



Graphical Editors User Manual

for the HDL Designer Series

Software Version 2010.3

June, 2011

© 1996-2011 Mentor Graphics Corporation
All rights reserved.

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

RESTRICTED RIGHTS LEGEND 03/97

U.S. Government Restricted Rights. The SOFTWARE and documentation have been developed entirely at private expense and are commercial computer software provided with restricted rights. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement provided with the software pursuant to DFARS 227.7202-3(a) or as set forth in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable.

Contractor/manufacturer is:

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.

Telephone: 503.685.7000

Toll-Free Telephone: 800.592.2210

Website: www.mentor.com

SupportNet: supportnet.mentor.com/

Send Feedback on Documentation: supportnet.mentor.com/user/feedback_form.cfm

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other third parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the respective third-party owner. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: www.mentor.com/terms_conditions/trademarks.cfm.

Table of Contents

Chapter 1

Introduction.....	17
The Design Creation Editors	18
DesignPad Text Editor	18
Block Diagram and IBD View Editors	18
Component Interface Editor	18
State Diagram and Algorithmic State Machine Editors	18
Flow Chart Editor	18
Truth Table Editor.....	19
Editor Windows	19
The Menu Bar	19
Toolbars	20
Standard Toolbar	20
Keyboard Shortcuts.....	21
Mnemonic Keys	22
Command Auto-repeat	22
Strokes	22
Common Features	23
Setting the Hardware Description Language	23
Setting Package References	23
Example VHDL Package List	25
Setting Compiler Directives	26
Formatting Text	27
Format Text Toolbar.....	27
Opening the Parent View	28
Editing the Parent Interface	28
Using the Same Window	29
Saving Graphic Editor Views	29
Automatic Backup and Recovery.....	31
Saving the Window Position and Size	32
Editing Object Properties	32
Redrawing a Window	32
Undo and Redo	32
Selecting Objects	33
Copying and Pasting Objects	34
Deleting Objects	34
Finding and Replacing Text Strings	35
More Search Options	35
Replacing a Text String	37
Object Linking and Embedding.....	38
Using Drag and Drop	40
Opening an OLE View	41

Generating HDL	41
VHDL Component Declarations	44
Setting a Black Box for Synthesis	44
Viewing the Generated HDL	44
Chapter 2	
Graphical Editor Windows	47
Diagram Editor Windows	48
Setting Preferences for Diagram Views	48
Setting Diagram Master Preferences	49
Setting Background Preferences	50
Moving and Copying Diagram Objects	51
Resizing Objects	53
Arranging Objects	53
Arrange Object Toolbar	53
Aligning or Distributing Objects	54
Rotating and Flipping Objects	54
Layering Comment Text and Graphics	55
Grouping Comment Text and Graphics	55
Adding Comment Text	56
Editing Text Properties	58
Adding Requirement Reference Object	59
Pasting in Editor	61
Pasting in Design Browser	63
Editing Text on a Diagram	64
Text Editing Shortcuts	65
Editing Text in the Text Editor	66
Moving Text	67
Changing Text Visibility	68
Adding Comment Graphics	72
Comment Graphics Toolbar	72
Adding a Line or Polyline	74
Adding an Arc	75
Adding a Rectangle or Polygon	75
Adding an Ellipse or Circle	76
Adding a Bitmap	76
Adding a Title Block	77
Displaying Object Information	78
Panels	78
Adding a Panel	79
Editing Panel Object Properties	79
Displaying a Panel	80
Viewing a Panel	80
Protecting Panels	81
Deleting a Panel	81
Printing a Panel	81
Editing Route Points	82
Setting Visual Attributes	83

Table of Contents

Appearance Toolbar	84
Setting Color Attributes	85
Toggling the Grid Visibility and Snapping	86
Changing the Diagram View	86
Table Editor Windows	87
Setting Preferences for Table Views	87
Selecting Table Cells	88
Editing a Table Cell	88
Changing the Table View	89
Resizing a Column or Row	90
Exporting a Table	90
The Diagram Browser	91
Browsing Diagram Structure	92
Browsing Diagram Content	94
Changing the Columns in the Content Pane	96
Sorting the Content Pane	97
Using Groups in the Content Pane	97
Using Flow Help	98
Signals Table	99
Signals Table Notation	100
Signal Declaration Columns	101
Signals Table Toolbars	101
Adding Port or Local Signal Declarations	102
Adding Comments to a Port or Local Signal Declaration	103
Resizing Columns	104
Hiding Columns	105
Filtering Columns	105
Grouping Signal Rows	105
Sorting Signal Rows	107

Chapter 3

Block Diagram and IBD Views	109
Editing Block Diagram and IBD Views	110
Adding Blocks and Components	110
Assigning Automatic Instance Names	111
Instantiating a Block	111
Instantiating a Component	112
Instantiating Verilog 2005 or System Verilog3.0 Text Components	114
Instantiating a ModuleWare Component	116
Instantiating an External HDL Model	120
Using a Soft Pathname for External HDL	123
Updating an External HDL Model	124
Adding an Embedded Block	124
Opening an Embedded View	125
Adding Embedded HDL Text	126
Updating an Instance	127
Reconciling Interfaces	128
Checking the Design	131

Checking Through Hierarchy	132
Editing Object Properties	133
Editing Component Properties	133
Editing Component Generics and Parameters	134
Editing Component Text Visibility	135
Editing Component Port map Frame	136
Setting Component Attributes and Embedded Constraints	138
Modifying Component Appearance	138
Editing Block Properties	139
Editing Block Generics and Parameters	140
Modifying Block Port Ordering	140
Setting Block Attributes and Embedded Constraints	141
Setting Block Appearance	141
Editing Embedded Block Properties	143
Editing Embedded Blocks HDL Text	144
Modifying Embedded Blocks Text Appearance	144
Modifying Embedded Blocks Text Box Appearance	144
Modifying Embedded Blocks Appearance	144
Editing ModuleWare Properties	145
Editing Moduleware Port map Frames	146
Setting ModuleWare Attributes and Embedded Constraints	146
Editing External IPs Properties	146
Editing External IP Generics	147
Editing External IP Text Visibility	147
Editing External IP Port map Frame	147
Setting External IP Attributes and Embedded Constraints	147
Modifying External IP Appearance	148
Editing Bundle Properties	148
Modifying Bundle Appearance	149
Editing Signal Properties	149
Editing VHDL Signal Declarations	150
Editing Verilog Signal Declarations	152
Editing Signal Text Visibility	153
Setting Signal Attributes and Embedded Constraints	154
Editing Signal Comments	155
Modifying Signal Appearance	155
Editing Port IOs Properties	156
Editing Port IO Text Visibility	156
Modifying Port IO Appearance	157
Editing Frame Properties	157
Modifying Frame Appearance	157
Editing Comment Text Properties	158
Modifying Comment Text Appearance	158
Modifying Comment Text Box Appearance	158
Editing Requirement Reference Properties	158
Modifying Requirement Reference Text Appearance	159
Editing Comment Graphics Properties	160
Editing User Declarations	161
Editing User Properties	162

Table of Contents

Setting the Scope for Net Changes	162
Adding Comments to a Signal or Port Declaration	163
Setting Attributes and Embedded Constraints	165
Propagating Net Changes	166
Inserting and Removing Nets	169
Ordering Port and Signal Declarations	171
Adding or Removing Design Hierarchy	172
Generics and Parameters	174
Generics and Parameters Tables	176
Accessing the Generics or Parameters Table	176
Generics Table Controls	177
Parameters Table Controls	178
Using the Generics and Parameters Table	179
Related Topics	184
Defining Generics and Parameters	184
Defining Generics for Components and Blocks	184
Defining Parameters for Components and Blocks	185
Editing Generics and Parameters for Instances	186
Editing VHDL Generic Values for Instances	187
Editing Verilog Parameter Values for Instances	189
Generics and Parameters Synchronization	192
Related Topics	194
Opening Block and Component Views	194
View Initialization	195
Setting the Default View	196
Mixed Language Designs	196
VHDL Instantiation of Verilog Components	197
Verilog Instantiation of VHDL Components	198
 Chapter 4	
Block Diagram Editor	199
Block Diagrams	200
Block Diagram Notation	201
Blocks and Components	202
Embedded Blocks and Embedded Views	203
Signals, Buses and Bundles	204
Ports and Signals	205
Changing the Display of Port Properties	206
Changing the Display of Signal Properties	208
Block Diagram Editor Toolbar	211
Adding Nets on a Block Diagram	212
Routing Nets	212
Adding a Signal or Bus on a Block Diagram	213
Ripping from a Bus	215
Adding Signal Stubs on a Block Diagram	217
Adding a Bundle on a Block Diagram	217
Adding Signals to a Bundle	218
Ripping from a Bundle	218

Using HDL Text to Combine or Split Signals	220
Adding Ports on a Block Diagram	220
Adding Ports to Existing Nets	221
Adding Ports from a Component	221
Changing the Mode of a Port	221
Rotating a Port	222
Rotating Signal Names	222
Adding a Global Connector on a Block Diagram	222
Connecting Overlapping Nets	222
Connecting Nets to a Block or Component	223
VHDL Port Mapping	224
Connecting Nets to a Port Map Frame	225
Highlighting a Net on a Block Diagram	225
Logic Shape Notation	227
Changing the Shape of a Block or Component	228
Choosing a Standard Shape	229
Hiding Ports on a Block or Component	230
Indicating Not or Clocked Ports	231
Setting Block Diagram Preferences	231
 Chapter 5	
IBD View Editor	245
Interface-Based Design	246
New Design Creation Flow	246
Code Re_Use Flow	247
Design Assembly Flow	248
IBD Working Environment	250
IBD View Matrix	250
IBD View Toolbar	251
Getting Designs into IBD Editor	252
Working on a Previously Created HDS Design	252
Creating a New Design View	253
Adding Design Elements	253
Adding Components	254
Adding Blocks	255
Adding Embedded Blocks	255
Adding Nets	256
Adding a Signal or Bus	256
Adding Ports	257
Adding a Net Slice	257
Adding Generate Frames	257
Using Generate Frames for Repeating Instances	258
Using Generate Frames for Repeating Structures	259
Using Generate Frames for Conditional Structures	259
Using a BLOCK Generate Frame	259
Adding Requirements References	259
Connecting Design Elements	261
Connecting Nets to Component Ports	261

Table of Contents

Connecting Existing Nets: Net_Centric_Connection Approach	261
Connecting Ports: Port_Centric_Connection Approach.	262
Mapping Expressions or Function Calls to Component Ports	263
Another Port_Centric_Connection Convention	265
Organizing View Layout	266
Expanding and Collapsing IBD Views	266
Moving Rows and Columns in an IBD View	267
Sorting Rows and Columns in an IBD View	267
Grouping IBD Rows and Columns	267
Showing/Hiding Columns in an IBD View	267
Adding Bundles to your IBD view	268
Adding Net/Component Comments	269
Adding Property Columns/Rows	269
Creating Filtered Views of the Design.	269
Defining Filter Settings and Logic	270
Creating Persistent Subset Views of the Design.	270
Pruning IBD Designs	272
Filtering Nets in an IBD view	272
Filtering Components in an IBD View	273
Managing Design Hierarchy	274
Adding a Level of Hierarchy	274
Flattening Design Hierarchy	275
Checking a Design in IBD Editor	276
Generating HDL from IBD views	276
Controlling the Generated HDL Code	276
Setting Generation Order	276
Setting the Style of the VHDL or Verilog code	277
Setting the Generation Hierarchy Level	277
Enforcing Generation	278
Cross Referencing Generation Errors	278
Setting a Black Box for Synthesis	278
Documenting IBD Design Views	279
Creating Visualization Views	279
Exporting to HTML	280
Exporting to TSV or CSV Files	280
Compiler Directives	281
Setting IBD Preferences.	281

Chapter 6

Port Map and Generate Frames	283
Port Map Frames	283
Adding a Port Map Frame.	283
Editing a Port Map	284
VHDL Port Map Example	285
Verilog Port Map Example.	286
Generate Frames	287
Adding a Generate Frame	288
Using Generate Frames for Repeating Instances	289

Using Generate Frames for Repeating Structures	291
Using Generate Frames for Conditional Structures	292
Using a BLOCK Generate Frame	296
Using Nested Generate Frames	298
Editing Generate Frame Properties	302
Chapter 7	
Component Interface Views	305
Opening a Component Interface	305
Tabular IO and Symbol Views	306
Tabular IO Notation	307
Hiding Columns	308
Filtering Columns	309
Tabular IO Toolbar	310
Sorting the Rows in a Tabular IO View	311
Adding Ports in the Tabular IO View	311
Grouping Port Rows	312
Setting Visual Attributes in the Tabular IO View	314
Symbol Notation	315
Symbol Toolbar	315
Adding Ports in the Symbol View	316
Customizing a Symbol	317
Editing Port Declarations	318
Changing the Port Declaration Order	319
Propagating Port Changes	320
Updating Instances	320
Adding Attributes to a Port Declaration	321
Adding Comments to a Port Declaration	321
Editing Symbol Generic or Parameter Declarations	322
Editing Symbol/Interface Object Properties	324
Editing Symbol User Declarations	324
Editing Symbol Body Properties	325
Setting Interface Preferences	327
Chapter 8	
Flow Chart Editor	339
Flow Chart Notation	340
Flow Chart Toolbar	342
Adding Objects on a Flow Chart	343
Adding a Start Point	345
Adding an Action Box	346
Adding a Decision Box	347
Adding a Wait Box	348
Adding a Loop	348
Breaking Out of a Loop	349
Adding a Case Box	349
Adding a Flow	351
Adding an End Point	352

Table of Contents

Adding Other Objects on a Flow Chart	352
Hierarchical Flow Charts	352
Concurrent Flow Charts	354
Adding a Concurrent Flow Chart	355
Opening a Concurrent Flow Chart	355
Renaming a Concurrent Flow Chart	355
Deleting a Concurrent Flow Chart	356
Editing Flow Chart Object Properties	356
Editing Action Box Object Properties	356
Editing Decision Box Object Properties	358
Editing Wait Box Object Properties	360
Editing Loop Object Properties	362
Editing Case Object Properties	363
Setting Flow Chart Properties	364
Setting Flow Chart Generation Properties	365
Sequential and Combinatorial Diagrams	366
Clock Signal	367
Reset Signal	367
Sensitivity List	367
Block Type	368
Animation	368
Editing Architecture or Module Declarations	369
Editing Concurrent Statements	370
Editing Process or Local Declarations	372
Setting Flow Chart Preferences	373
 Chapter 9	
Truth Table Editor	377
Truth Table Notation	379
Truth Table Toolbars	380
Editing a Truth Table Cell	380
Comparison Operators	380
Adding a Column or Row	381
Deleting a Column or Row	381
Setting Truth Table Properties	381
Setting Truth Table Generation Properties	382
Sequential and Combinatorial Diagrams	383
HDL Style	383
Clock Signal	384
Reset Signal	384
Sensitivity List	385
Full/Parallel Case	385
Assignment Type	385
Editing Architecture or Module Declarations	386
Editing Concurrent Statements	387
Editing Process or Local Declarations	387
Editing Global Actions	388
Case and IF Style Truth Tables	389

Case Style with a Single Input Expression	390
Case Style with Multiple Input Expressions	390
Setting Truth Table Preferences	391
Chapter 10	
Graphical Rendering	395
Design Extraction	395
Recovering Design Structure	396
Recovering Verilog Parameters	398
Recovering State Machines	398
Recognizing State Machines	399
Recovering Flow Charts	399
Incremental Recovery	401
Using the Convert to Graphics Wizard	401
Setting Convert to Graphics View Styles	402
Setting Libraries for Black Box Components	403
Setting Convert to Graphics Wizard Options	404
Setting Convert to Graphics Options	405
Block Diagram Options	406
Routing Options	407
Placement Options	408
Updating a Graphics View from Generated HDL	409
Visualizing HDL Text as Graphical Views	409
Block Diagram Layout and Routing	409
Changing the Layout of a Block Diagram	410
Automatic Routing	410
Autoroute	411
Autoconnect	411
Autobundle	411
Connect by Name	412
Bus Reconstruction	412
Chapter 11	
Simulation and Animation	415
Simulator Cross-Probing	415
Simulation Toolbar	416
Adding Signals to Simulator Windows	418
Removing Signals from Simulator Windows	418
Adding Signals to the Simulator Log	419
Highlighting Signals in the Simulator	419
Reporting Signal Information	419
Adding and Removing Breakpoints	419
Enabling and Disabling Breakpoints	420
Reporting Breakpoint Status	420
Adding and Removing Simulation Probes	421
Setting Probe Properties	423
Forcing Signal Values	423
Choosing the Simulation Instance	424

Table of Contents

Setting the Simulator Environment	425
Running the Simulator	426
Running a Simulation	426
Stepping Through a Simulation	426
Displaying Simulator Windows	427
Restarting the Simulator	427
Using the ModelSim Source Window	427
Cross-Probing from ModelSim.	428
State Diagram and Flow Chart Animation.	431
Animation Toolbar	432
Enabling Data Capture	433
Setting the Activity Trail	434
Graphical Highlighting	435
Reviewing Animation	436
Linking Diagrams for Animation	437
Mixed Language Animation	437

Chapter 12

Using a Test Bench	439
Test Benches	439
Creating a Test Bench	440
Defining Stimulus	441
Using ModuleWare Stimulus Generator Parts	441
Defining Stimulus on a Flow Chart	441
Wait Statements	442
Loop Statements	443
Case Statements	443
Defining Stimulus using Lookup Tables	443
Defining Stimulus using TextIO.	444
Defining Stimulus using a State Machine.	446
Generating a Clock using HDL Statements	446
Analyzing Results	447
Re-using a Test Bench	448

Glossary

Index 475

Index

End-User License Agreement

List of Tables

Table 1-1. Standard Toolbar	20
Table 1-2. Format Text Toolbar	27
Table 2-1. Arrange Object Toolbar	53
Table 2-2. Text Editing Shortcuts	65
Table 2-3. Comment Graphics Toolbar	72
Table 2-4. Comment Graphics Palettes	72
Table 2-5. Comment Graphics Menu Commands	73
Table 2-6. Appearance Toolbar	84
Table 2-7. Appearance Palettes	85
Table 2-8. Structure Navigator Notation — Block Diagram	92
Table 2-9. Structure Navigator Notation — State Machine, ASM, Flow Chart	93
Table 2-10. Structure Navigator Notation — Symbol	93
Table 2-11. Structure Navigator Notation — Text Objects	94
Table 2-12. Content List Notation — Block Diagram, IBD	95
Table 2-13. Content List Notation — Flow Chart	95
Table 2-14. Content List Notation — State Diagram	96
Table 2-15. Content List Notation — ASM	96
Table 2-16. Flow Help Notation	98
Table 2-17. Signals Table Toolbar	101
Table 2-18. Tabular IO View Commands for Adding Port or Local Signal Declarations ..	102
Table 3-1. Block Diagram/ IBD Commands for Adding Blocks and Components	110
Table 3-2. Supported Verilog2001/System Verilog 3.0 Types	114
Table 3-3. Generics Toolbar	177
Table 3-4. Generics Table Content	178
Table 3-5. Parameters Toolbar	178
Table 3-6. Parameters Table Content	178
Table 3-7. Object Properties — Parameters Page Controls	191
Table 4-1. Graphical Editor and HDL Text Views Notation	202
Table 4-2. Block Diagram Editor Toolbar	211
Table 4-3. Block Diagram Commands for Adding Nets	212
Table 5-1. IBD View Toolbar	251
Table 7-1. Tabular IO Toolbar	310
Table 7-2. Tabular IO View Commands for Adding Ports	311
Table 7-3. Symbol Toolbar	315
Table 7-4. Symbol View Commands for Adding Ports	316
Table 8-1. Flow Chart Notation	340
Table 8-2. Flow Chart Toolbar	342
Table 8-3. Verilog Declarations	369
Table 11-1. Simulation Toolbar Commands in State Diagram and Flow Chart Views	416
Table 11-2. Simulation Toolbar Commands in Block Diagram Views	417

List of Tables

Table 11-3. Simulation Toolbar Additional Commands in State Diagram	417
Table 11-4. ModelSim Main Window Commands	428
Table 11-5. ModelSim Source Window Commands	428
Table 11-6. ModelSim Wave Window Commands	428
Table 11-7. ModelSim Structure Window Commands	428
Table 11-8. ModelSim List Window Commands	429
Table 11-9. ModelSim Signals Window Commands	429
Table 11-10. Animation Toolbar	432

Chapter 1

Introduction

This chapter introduces the HDL Designer Series graphical design creation editors, their basic user interface and features that are common to all of the graphical editors.

The Design Creation Editors.....	18
DesignPad Text Editor	18
Block Diagram and IBD View Editors	18
Component Interface Editor	18
State Diagram and Algorithmic State Machine Editors	18
Flow Chart Editor	18
Truth Table Editor.....	19
Editor Windows	19
The Menu Bar	19
Toolbars	20
Keyboard Shortcuts.....	21
Common Features	23
Setting the Hardware Description Language	23
Setting Package References	23
Setting Compiler Directives	26
Formatting Text	27
Opening the Parent View	28
Using the Same Window	29
Saving Graphic Editor Views	29
Editing Object Properties	32
Redrawing a Window	32
Undo and Redo	32
Selecting Objects	33
Copying and Pasting Objects	34
Deleting Objects	34
Finding and Replacing Text Strings	35
Object Linking and Embedding.....	38
Using Drag and Drop	40
Opening an OLE View	41
Generating HDL.....	41
VHDL Component Declarations.....	44
Setting a Black Box for Synthesis	44
Viewing the Generated HDL	44

The Design Creation Editors

All of the *HDL Designer Series* tools include an integrated language sensitive HDL text editor and may also include one or more *graphical editors* for opening *diagram editor* and *table editor* views.

DesignPad Text Editor

The built-in *DesignPad* HDL text editor can be used for editing and viewing HDL text views (or viewing the HDL generated from the graphical views). This editor is described in a separate *DesignPad Text Editor User Guide*.

Block Diagram and IBD View Editors

The *block diagram* editor represents the design structure by blocks and re-usable components connected by signals, buses or bundles. The tabular *IBD view* editor represents the design structure by describing the signal interfaces between the blocks and components in the design. Blocks and components can be defined using *state diagram*, *ASM chart*, *flow chart* or *truth table* or *HDL text* views and HDL generated or compiled for individual views or hierarchies.

Component Interface Editor

The *tabular IO* view can be used to define component interfaces in the form of a table showing its inputs and outputs. The *symbol* diagram view can be used for creating and updating the graphical symbol used for a component on a block diagram.

State Diagram and Algorithmic State Machine Editors

The state diagram editor can be used to describe the behavior of a block or component view as a number of states and the transitions between them. The ASM chart editor describes an algorithmic state machine (*ASM*) in terms of a sequence of operations represented by flow chart style notation. Hierarchical or concurrent state machines are supported and HDL can be generated for the active state machine. These editors are described in a separate *State Machine Editor User Manual*.

Flow Chart Editor

The flow chart diagram editor can be used to describe a block or component view in terms of standard flow chart symbols including action boxes, decision boxes, loops, wait and case boxes. Hierarchical or concurrent flow charts are supported and HDL can be generated for the active flow chart.

Truth Table Editor

A tabular truth table editor which can be used to represent a block or component view as a spreadsheet defining output actions as a function of input conditions or expressions. Sequential or combinatorial HDL can be generated for the truth table using Case or If-then-else style HDL.

Editor Windows

A new window is opened for each new graphical block diagram, IBD view, state diagram, ASM chart, flow chart, truth table or symbol including separate windows for hierarchical, concurrent or embedded views.

Each window can be moved, resized or iconized and has its own menu bar, one or more toolbars and a status bar. Refer to the *HDL Designer Series User Manual* for general information about the graphical user interface.

You are prompted to save any changes to the active view when you close a window which is the last open view of a diagram or table and prompted to exit from the tool if the window is the last open window.

The Menu Bar

The following pulldown menus are provided in most graphical editor windows although some menus or commands may not always be available:



Note



The **Diagram** menu is available in a diagram editor or the **Table** editor in a table editor.

A short message describing the associated command is displayed in the status bar when the cursor is moved over any pulldown menu item or toolbar button. When tooltips are enabled, the command name appears in a small window beneath the button if the cursor is held over a toolbar button.

Many commands are also available in a context-sensitive popup menu which is displayed when you press and release the **Right** mouse button.

All menu items can be accessed by a keyboard shortcut using the **F10** key and the underlined mnemonic letter. For example, to save the current view, you can use the keyboard shortcut: **F10 F + S**.

There are additional keyboard shortcuts defined for standard commands (such as saving a view) using the **Ctrl** and **Shift** keys.

Commands which add an object normally repeat for you to add another similar object until you select another command or use the **Right** mouse button (or the **Esc** key) to terminate the command. However, you can set a preference in the **General** tab of the Main settings dialog box to remain active or activate only once or you can toggle this mode for the current command by holding down the **Ctrl** key while selecting the button or menu option.

Toolbars

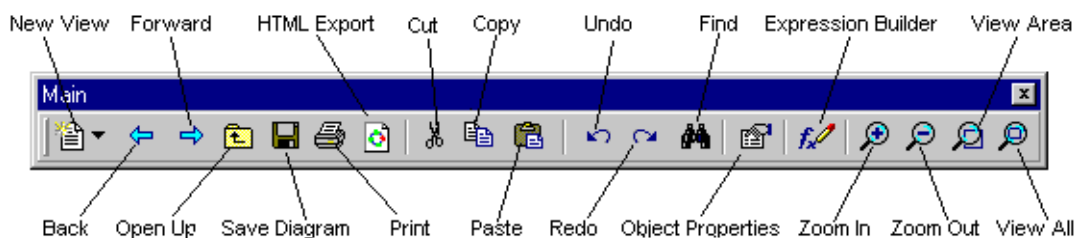
The most commonly used commands are available from toolbars. The toolbars are normally docked against the upper or lower edge of the active window but each toolbar can be moved independently to an alternative edge or allowed to float freely.

Refer to “Toolbars” in the *HDL Designer Series User Manual* for general information about the toolbars including procedures for “Docking and Undocking Toolbars”.

Additional toolbars are defined for tasks, HDL tools, version management and for commands specific to each editor window. Special simulation and animation toolbars are available in the graphical editors when a supported simulator is invoked. Refer to the *HDL Designer Series User Manual* for information about the tasks, HDL tools and version management toolbars. The other toolbars which are available in each graphical editor are described later in this manual.

Standard Toolbar

The standard toolbar in the graphical editors typically includes the buttons shown below:

























The following commands are usually available from the standard toolbar which is displayed in the block diagram, IBD view, state diagram, flow chart, truth table, tabular IO and symbol windows:

Table 1-1. Standard Toolbar

Icon	Description
	Create a new view
	Display the previous window when in same window mode
	Display the next window when in same window mode

Table 1-1. Standard Toolbar (cont.)

Icon	Description
	Open the parent view
	Save all changes made to the current view
	Print the current window
	Export the current view to create a website
	Move selection to the clipboard
	Copy selection to the clipboard
	Paste the contents of the clipboard
	Display the Object Properties dialog box
	Build a VHDL or Verilog expression
	Undo the last command
	Redo the last undo command
	Find a specified text string
	Increase the magnification of the active diagram
	Decrease the magnification of the active diagram
	View a specified diagram area
	View the entire diagram

The  button discloses a menu which allows you to choose the type of view to create. The  button is available in a state diagram or flow chart window only. The , ,  and  buttons are not available in truth table, tabular IO or IBD view windows.

Keyboard Shortcuts

Many commands are also available using keyboard shortcuts. Refer to “Keyboard Shortcuts” in the *HDL Designer Series User Manual* for general information about keyboard shortcuts including menu accelerator keys, dialog box shortcuts and mouse buttons.

Refer to the **Quick Reference Index** in the Help and Manuals tab of the HDS InfoHub for lists of the keyboard shortcuts supported in each window. To open the InfoHub, select **Help and Manuals** from the **Help** menu.

Mnemonic Keys

Single-press keys (which correspond to the underlined mnemonic character in the menu command) are defined to add specific objects in diagram editor windows.


Note



Note that mnemonic keys and shortcuts using the **Shift** key with an alphabetic character are not available in the table editors where these keys are used to enter characters in a table cell.

Command Auto-repeat

After adding an object in a graphical editor window using a toolbar or menu command, the command normally repeats until you use the **Esc** key (or **Right** mouse button) to terminate the repeating command. However, you can set a preference for the command to **Remain active** or **Activate only once** in the **General** tab of the Main Settings dialog box which is displayed when you choose **Main** from the **Options** menu.

Alternatively, you can toggle this mode for the current command by using the **Ctrl** key. For example, you can use **Ctrl**+ to add a single panel on a diagram when auto-repeat mode is on or to add multiple panels when auto-repeat mode is off.

Strokes

You can execute a command in a diagram editor using a stroke by simply holding the **Middle** mouse button down and dragging across the window. The command to be executed is shown on the screen while the mouse button is held down and executed when you release the button.

A further set of strokes can be performed by holding down the **Shift** key while dragging the **Middle** mouse button.

You can cancel a stroke by returning to the starting position before releasing the **Middle** mouse button or by using the **Esc** key.

Strokes can be enabled or disabled by setting an option in the Main Settings dialog box as described [“Setting Preferences for Diagram Views”](#) on page 48.

Refer to the **Quick Reference Index** in the Help and Manuals tab of the HDS InfoHub for lists of the supported strokes. To open the InfoHub, select **Help and Manuals** from the **Help** menu.

Note



If you use a mouse that has a wheel scrolling device, it may be necessary to modify the mouse setup to allow use of the wheel for strokes.

Common Features

This section describes features that are available in all of the graphical editors.

Setting the Hardware Description Language

You can set the default hardware description language (VHDL or Verilog) used for new *graphical editor* views from the **General** tab of the Main Settings dialog box which can be displayed by choosing **Main** from the **Options** menu.

You can also override the default language by choosing VHDL or Verilog in the File Creation wizard when you create a new graphic editor view.

Refer to “Using the Design Content Creation Wizard” in the *HDL Designer Series User Manual* for information about creating design views.

When *VHDL* is selected, a default *package list* is shown on all graphical diagrams and VHDL syntax is used in *port* and *signal* declarations. *VHDL generics* can be defined on a *symbol* or *tabular IO* interface and individual generic values set for each instance in a *block diagram*, *IBD view*, *state diagram*, *flow chart* and *truth table*.

When *Verilog* is selected, default *compiler directives* are shown and Verilog syntax is used in *port* and *signal* declarations. *Verilog parameters* can be defined on a *symbol* or *tabular IO* interface and individual parameter values set for each instance in a *block diagram*, *IBD view*, *state diagram*, *flow chart* and *truth table*.

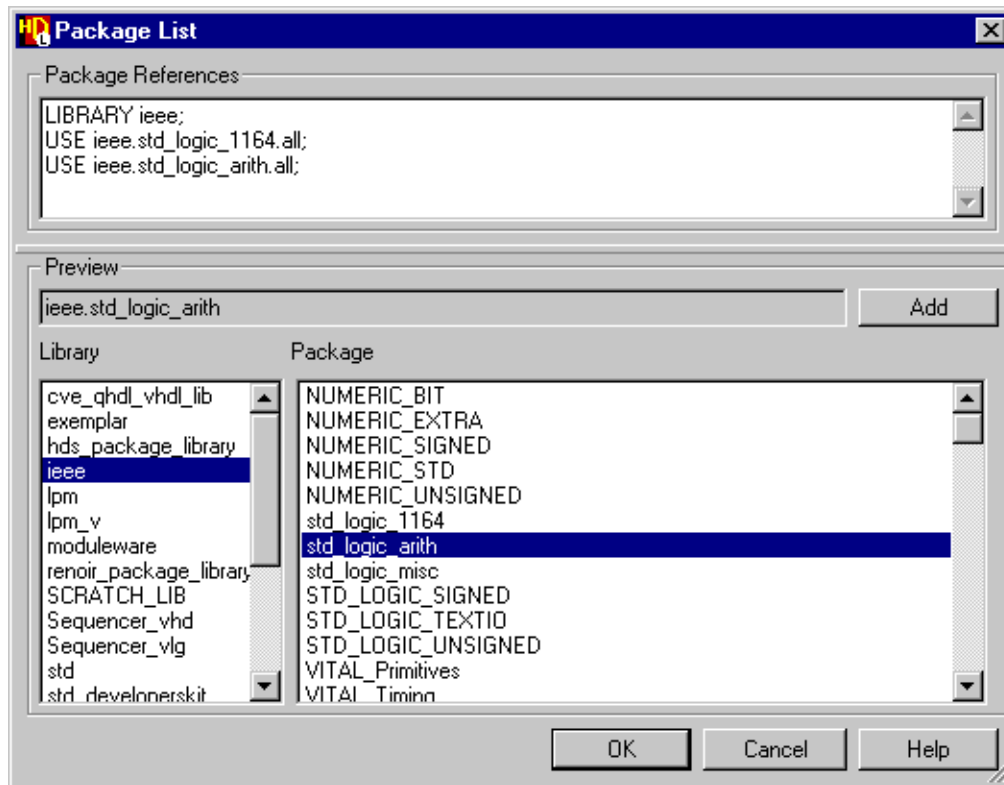
Setting Package References

When you are creating VHDL based designs, you can reference *VHDL packages* that define VHDL types subtypes, functions, procedures or constants. These packages can be referenced at any level of the design hierarchy and may be standard library packages containing pre-defined VHDL type definitions or user-defined packages in your design data libraries.

The default shared project file (*shared.hdp*) contains mappings for all the standard packages supplied with the HDL Designer Series tool.

You can also define your own standard packages or use an existing package supplied by another vendor. You can create your own local packages as HDL views stored with other block or component design unit views in a design data library or by referencing an existing external package.

You can set package references for a graphic editor view by double-clicking over the package list on a diagram, or by choosing **Package References** from the **Diagram**, **Table** or popup menu to display the Package List dialog box.



The dialog box displays any existing package references that are set for the current view (or the default packages references set in your preferences for a new view).

You can add references by selecting from the available libraries and choosing any of the packages contained in them. The supplied package libraries include all the standard types supported by ModelSim. Any packages contained in the currently mapped user-defined libraries will also be available.

Library and use statements for the selected package are added to the list of package references when you use the **Add** button and the updated list of references is applied to the active view when you choose the **Ok** button.

You can also add packages by entering any valid library or use statements in the list or by editing an existing library or use statement. For example, you could add a reference to the package *ieee.std_logic_unsigned* then replace the suffix *.all* by a function name (such as *CONV_INTEGER*) to explicitly add a reference:

```
USE ieee.std_logic_unsigned.CONV_INTEGER
```

You can remove references from the package list by simply deleting the reference in the dialog box.

The package list can also be edited by direct text entry and may optionally include comments or pragmas entered using the standard VHDL comment characters (--). The syntax is automatically checked on entry in a diagram editor unless syntax checking has been disabled in the master diagram preferences.

Note

Default package references can be set by choosing **VHDL** from the **Options** menu and selecting the **Default Package References** tab of the VHDL Options dialog box. The default packages are available on all editor views unless you have explicitly removed them.

The referenced packages are parsed during VHDL generation to verify the type definitions used in your design views.

Refer to “Setting Default Package References” in the [HDL Designer Series User Manual](#) for information about setting default package references. The default references are included for all views unless you have explicitly removed them.

Example VHDL Package List

The VHDL package list can be edited to include any valid LIBRARY and USE statements of the form:

```
LIBRARY <library_name>;  
USE <library_name>.<package_name>.<item_name>;  
USE <library_name>.<package_name>.all;  
USE <library_name>.all;
```

The package list can also include pragmas or comments prefixed by the VHDL comment characters (--).

For example:

```
LIBRARY ieee;  
-- Use the definition of std_logic from IEEE std_logic_1164  
USE ieee.std_logic_1164.std_logic;  
  
-- Use all contents of the IEEE std_logic_arith package  
USE ieee.std_logic_arith.all  
  
-- Ignore the following package references for synthesis  
-- pragma synthesis_off  
LIBRARY std_developerskit;  
USE std_developerskit.mempak.all;  
-- pragma synthesis_all  
  
LIBRARY my_parts; -- declare my library
```

Setting Compiler Directives

When you are creating Verilog based designs, you can insert compiler directives for the HDL generated from any graphic editor view.

In general, a compiler directive passes information to the Verilog compiler or other downstream tool and any directive recognized by your tools can be entered.

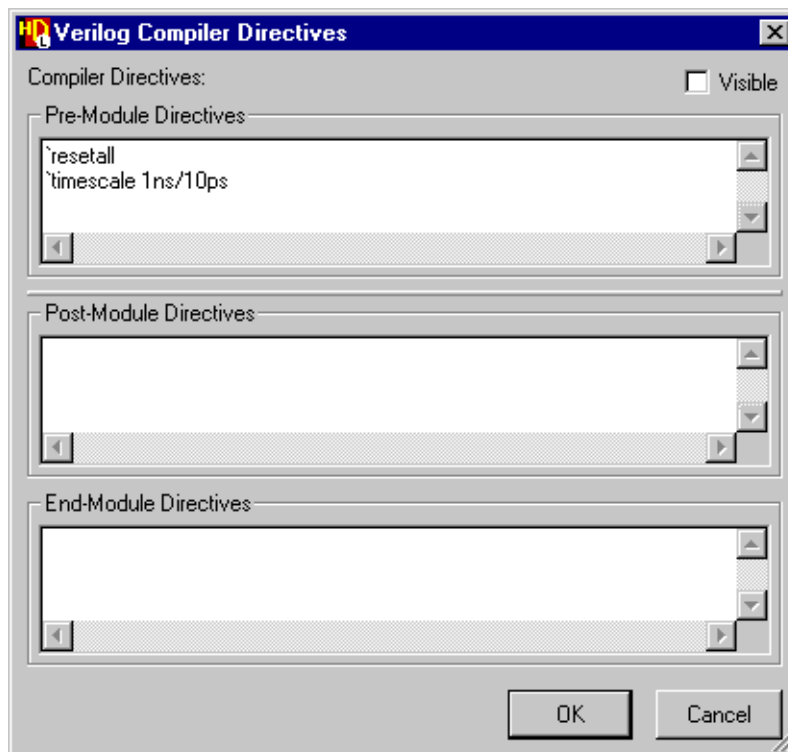
You can set Verilog compiler directives by double-clicking over the compiler directives list on a diagram or by choosing **Compiler Directives** from the **Diagram**, **Table** or popup menu to display the Verilog Compiler Directives dialog box listing any existing compiler directives that are set for the current view.

Note



The **Diagram** menu is available in a diagram editor or the **Table** editor in a table editor.

The dialog box allows you to enter pre-module directives which are included in the generated Verilog before the *module* keyword, post-module directives which are included after the *module* keyword and end-module directives which are included at the end of the Verilog module.



The compiler directives list can also be edited by direct text entry and may optionally include comments entered using the standard Verilog comment characters (*//*).

The directive syntax is automatically checked on entry in a diagram editor unless syntax checking has been disabled in the master diagram preferences.

Refer to “Setting Default Compiler Directives” in the *HDL Designer Series User Manual* for information about setting default compiler directives. The default directives are included for all views unless you have explicitly removed them.









Formatting Text

You can format text in the block diagram, state diagram, flow chart, symbol and truth table editors using the Format Text toolbar.

Format Text Toolbar






The following commands are available from the Format Text toolbar or as shortcut keys:




Table 1-2. Format Text Toolbar




Button	Shortcut	Description
	Ctrl + B	Applies bold formatting to text
	Ctrl + I	Applies italic formatting to text
	Ctrl + U	Underlines selected text
	none	Left aligns text
	none	Centers text in cell
	none	Right aligns text
	none	Increases size of the selected text
	none	Decreases size of the selected text



Note




The , , and  buttons are disabled in multi-line truth table cells. The  and  buttons are not available in a truth table.

If multiple cells are selected in a table, the formatting for each cell is toggled when you use the ,  or  buttons.

The , , and  buttons are available to align the text when one or more cells are selected in a table editor view and when embedded or comment text is selected on a diagram.


You can increase the font size used for the selected text by using the  button or decrease the font size using the  button. When scalable fonts are selected, the size is increased (or decreased) by one point size. For non-scalable fonts the next available font size is used.

You can change the text color used for any selected text object in a diagram editor window by using the  button in the Appearance toolbar.

The toolbar can be displayed or hidden by setting the **Format Text** option in the **Toolbars** cascade of the **View** menu.

Refer to “[Toolbars](#)” on page 20 for more information about toolbars.

Opening the Parent View

You can open up into the parent view using the  button, **Ctrl + Shift + o** shortcut or by choosing **Open Up** from the popup menu or the **Open** cascade of the **File** menu in any graphic editor window.

If the active window is a view of a block, the parent block diagram or IBD view is opened. If it is a view of a component, the component interface is opened in the symbol or tabular IO editor.

Note



You can choose whether the tabular IO or symbol editor is used to open the component interface by setting a preference in the **Miscellaneous** tab of the Symbol Master Preferences dialog box.

In a hierarchical state machine or hierarchical flow chart, the parent state diagram or flow chart is opened, unless you are already at the top level when the parent block diagram, IBD view, tabular IO or symbol editor view is opened.

You cannot open up from a symbol although you can use the popup menu to open down into any of its child views.

Editing the Parent Interface

You can edit the parent interface for any graphic editor view by choosing **Interface** from the **Open** cascade of the **File** menu.

If the view in the active window has a parent block diagram or IBD view, this view is opened.

If the parent view is a block diagram it is opened with the block representing the interface to the child view in the middle of the window.

If the view in the active window describes a component, the component interface is opened in the symbol or tabular IO editor.

Using the Same Window

You can re-use the current window when you open up or open down into an existing view from a graphic editor by setting the **Use Same Window** option in the **Window** menu. When this option is checked, the related diagram or table is opened in the same window without changing its position or size.


If the previous window has been edited and is the last open view of the diagram or table, you are prompted whether to save your changes before the view is closed. Note however, that a new tab is always opened when you create a concurrent view or hierarchical view for a state diagram or flow chart.



A new tab is also used when you display alternative block diagram and IBD views (or symbol and tabular IO views).

The window mode is saved as a preference and used as your default mode the next time you invoke a HDL Designer Series tool.



Tip: You can temporarily change the window mode by holding down the **Ctrl** key when you open the related view.


For example, **Ctrl** +  to open up, **Ctrl** + double-click on object to open down or **Ctrl** + menu option.

When **same** window mode is set, you can use the  button to go back to the previous window or the  button to go forward to the next window (if a forward travel log exists).

Travel log information is only recorded when a view has been named. Therefore, if you create a hierarchical state machine or flow chart and navigate around it without saving, no travel log information is recorded.

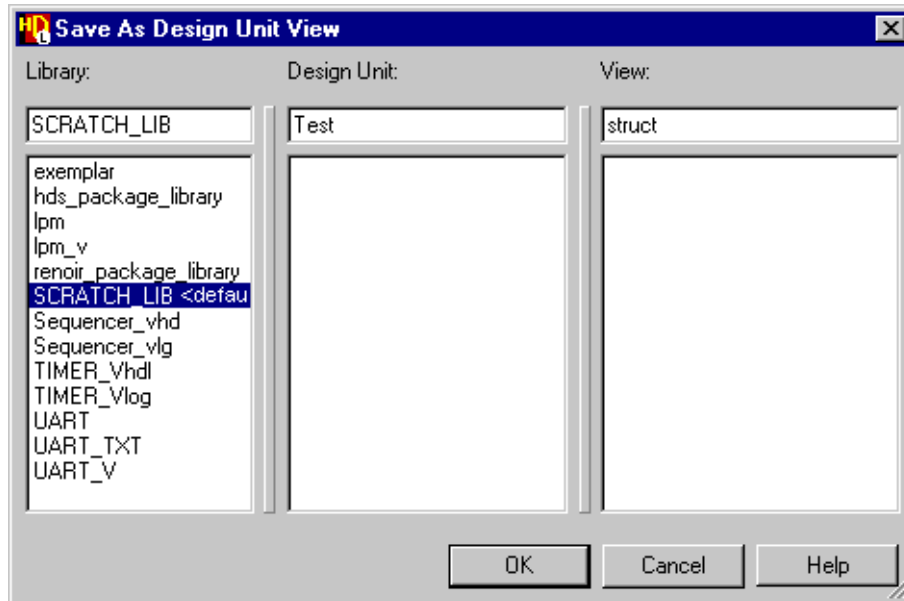
If you have renamed a design unit or design unit view since it was last traversed, the travel log will reference the old name and an error message is issued when you attempt to navigate into the diagram or table.

Saving Graphic Editor Views

You can save the active graphic editor view by using the  button, choosing **Save** from the **File** menu or using the **Ctrl + S** shortcut.

If the view has not been previously saved the Save As Design Unit View dialog box is displayed for you to enter a library name, design unit name and design unit view name. You can save an existing view with a new library, unit or view name, by using the **Save As** command. However, this option is not available in a block diagram or IBD view.

The current default library is automatically selected or you can choose from a list of other *Regular* libraries in the active project.



You can choose from the existing design units within a library or the existing views of each design unit. Alternatively, you can enter a new design unit or view name.

You can use any name for a library, design unit or view name but all names must be valid identifiers for the hardware description language you are using. A two or three character extension (*.bd* for a block diagram, *.ibd* for an IBD view, *.sm* for a state machine, *.fc* for a flow chart or *.tt* for a truth table) is added automatically to identify the type of view you are saving. For example: *struct.bd* or *flow.fc*.

If you do not specify a view name, the default names *struct* (for a block diagram or IBD view) or *fsm* (for a state machine view), *flow* for a flow chart view or *tbl* for a truth table view are used. These defaults can be changed by setting preferences.

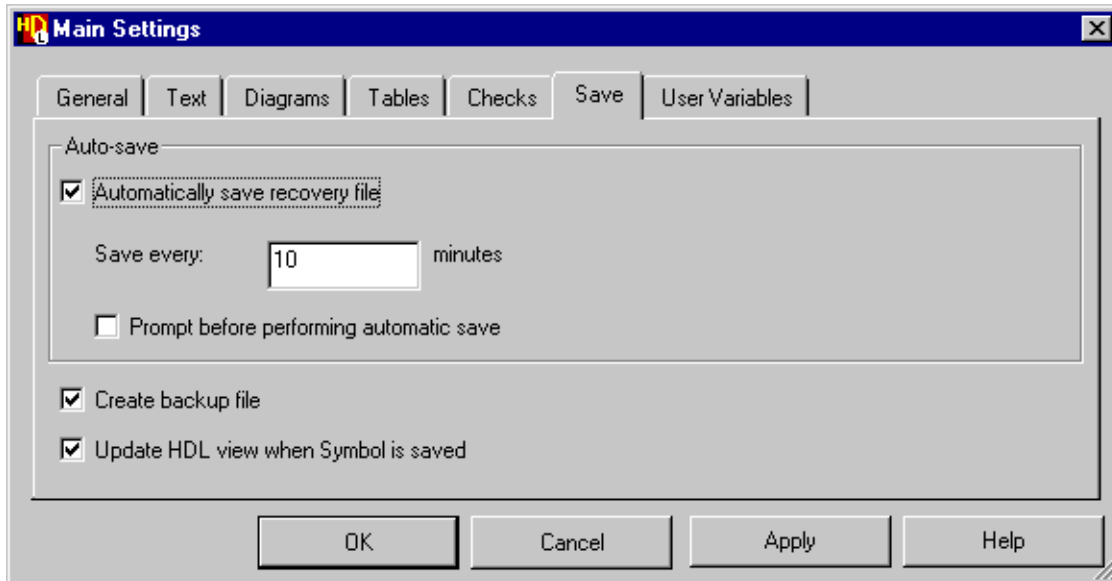
The library, design unit and view name of the view is shown in the title bar (or untitled for a new view which has not been saved). An asterisk (*) after the name in the title bar indicates that there are changes which need to be saved.

When you save a hierarchical state machine or flow chart, child views are named after the parent diagram and the hierarchical state that represents the child on its parent diagram.

Many hierarchical operations (for example, opening a child view or generating HDL) automatically perform a save on the parent design unit.

Automatic Backup and Recovery

You can set preferences to control whether backup and recovery files are saved for graphic editor views. These preferences can be set from the **Save** tab of the Main Settings dialog box which is displayed when you choose **Main** from the **Options** menu in any window.



You can choose to automatically save a recovery file (for example: *struct.bd.\$rec*) at saved at a specified time interval (minimum one minute) if the file has been edited since it was last saved.

Note



On a Windows workstation, you can also check an option to prompt before performing an automatic save.

You can choose to create a backup file by saving the existing version of each view (for example: *struct.bd.bak*) when the view is edited. The backup files are not deleted but are overwritten each time a view is explicitly saved.

You should not normally need to access a backup file, but it is possible to open these files after renaming the normal file and removing the *.bak* extension from the backup file.

The temporary recovery files are automatically deleted when a view is saved or closed normally. However, in the event of a power interruption or other system crash, the recovery file is detected when you attempt to re-open the view and you can choose to open the recovery file or to open the last normally saved file. (The unopened file is saved as a backup file in case the chosen file has been corrupted and cannot be opened.)

Autosave for a block diagram or IBD view does not update related interface data on disk and you may need to reconcile interfaces after recovering these views.

The save options are ignored for HDL text views. Save options for these views are determined by the preferences or defaults set in the text editor. However, when a text view containing HDL interface information exists, you can control whether the interface definition is updated when a symbol is edited by setting **Update HDL view when symbol is saved**.

Unsetting this option may be useful if your text editor is not sensitive to an open file being updated by another application. It can also be used to ensure that comments or code in the HDL text view header are not overwritten when the symbol is saved. However, if this option is unset, any interface changes must be made in the text file and propagated to the parent view by updating the component instances on the parent view.

These options are saved in the master preferences file when you confirm the dialog box and are used as default settings for the current and later sessions.

Note





Note that no backup or recovery files are written until a view has been saved.

Saving the Window Position and Size

When you save any graphic editor view, the current view, window position and size is also saved and used when the view is next opened.

Editing Object Properties

You can edit the properties for objects in a block diagram, IBD view, state diagram, flow chart, symbol, or tabular IO view by using the  button, **Alt** +  shortcut or by choosing **Object Properties** from the **Edit** or popup menus when one or more objects is selected.


An Object Properties dialog box is displayed which allows you to modify the properties of the selected objects in the diagram. Objects that do not exist in the selection set are disabled.

The Object Properties Dialog box pages or tabs are described in the appropriate sections for each editor later in this manual.

Redrawing a Window


A graphic editor window may sometimes become cluttered after moving text objects or when the window has been partially obscured by another application. If this occurs, you can redraw the window by choosing **Refresh** from the **View** menu.

Undo and Redo

At any time during an edit, you can reverse the previous command by using the **Undo** command in the **Edit** menu, the  toolbar button or the **Ctrl** + **Z** shortcut. The last command which can

be undone is shown on the menu (for example, after completing a move operation, the menu option **Undo (Move)** is available).

If any command cannot be undone you are warned before it is executed.

The **Redo** command (which also available from the **Edit** menu, using the  toolbar button or the **Ctrl + V** shortcut) allows you to restore the most recent 'undo' command.

Successive commands can be used to undo or redo any previous operations since the application was invoked.

Selecting Objects

You can select objects in a diagram editor or table editor window by clicking the **Left** mouse button with the cursor over the object and extend the current selection set using **Shift + Left** or add objects to (or remove objects from) the selection by using **Ctrl + Left**.

When an object is selected, small "handles" are displayed at each vertex. Solid handles are used for resizable objects or unfilled handles for non-resizable objects.



When there are several objects near the cursor in a diagram editor, the default select mode selects the closest object. Typically, this means that when you click over an object with associated text (for example, a block), the text (in this case the name text) is selected. To select an object and its associated text, hold the **Left** mouse button and drag a select rectangle around one or more objects. Any objects within (or partially within) the rectangle are included in the select set.






Note



When you click over a signal, bus or bundle in a block diagram (or a transition in a state diagram) the connector line is preferentially selected unless you explicitly click directly over the associated name text.

All objects in the active graphic editor window can be selected by choosing **Select All** from the **Edit** menu or using the **Ctrl + A** shortcut.

Three different selection modes are available in a diagram editor window. The default mode (indicated by  the toolbar button and  cursor) selects any object.

You can change the selection mode by using the  button to display a menu which allows you to select text only (the button changes to  and the cursor to ) or select shapes only (the button changes to  and the cursor to .

In a block diagram or flow chart, you can use the **Alt + Left** keys to select an individual segment of a net or flow. This combination can also be used with the **Shift** and **Ctrl** keys to select multiple segments. The selected segments can then be moved or deleted independently from other connected segments.

Copying and Pasting Objects

You can copy any graphic or text object in a graphic editor using the  toolbar button, the **Copy** command from the **Edit** menu or the **Ctrl + C** shortcut.

Alternatively, use the  toolbar button, the **Cut** command from the **Edit** menu or the **Ctrl + X** shortcut to move the object to the clipboard.


If any graphics objects are copied, an internal paste buffer is used. However, if any text objects are copied (or only text objects) they are copied to the system clipboard and can be pasted into an external application.

On a Windows PC, you can also use the **Copy Picture** command from the **Diagram** or **Table** menu to copy the entire window view into the system clipboard. This command can be used to copy a diagram or table into an external application as a Windows bitmap or enhanced metafile.

Note



The **Diagram** menu is available in a diagram editor or the **Table** editor in a table editor.

You can paste any graphic or text object that has been cut or copied to another position on the same graphic editor view (or in another window of the same type) using the  toolbar button, the **Ctrl + V** shortcut or the **Paste** command from the **Edit** menu or popup menu.

Graphic objects cannot be pasted using this command to diagrams of a different type or to another application. However, text objects can normally be pasted into any other window.

You can also use the **Paste from system clipboard** option which is available as a cascade from the **Paste Special** option in the popup menu to explicitly paste text objects into another text object, ignoring any graphic objects in the internal paste buffer.

When you paste a named object (such as a block or state) and the name already exists in the diagram, the new object is given a unique name by adding an integer to the object name. (For example: *Block*, *Block1*, *Block2*...). In a diagram editor, a ghosted image of the pasted object is attached to the cursor and can be placed by dragging the mouse and clicking at the required location.

You can also copy an object (or objects) by using the **Ctrl** key and dragging with the **Left** mouse button or by dragging with the **Right** mouse button and using the **Copy Here** option from the popup menu to paste a copy at the position of the cursor.


Deleting Objects

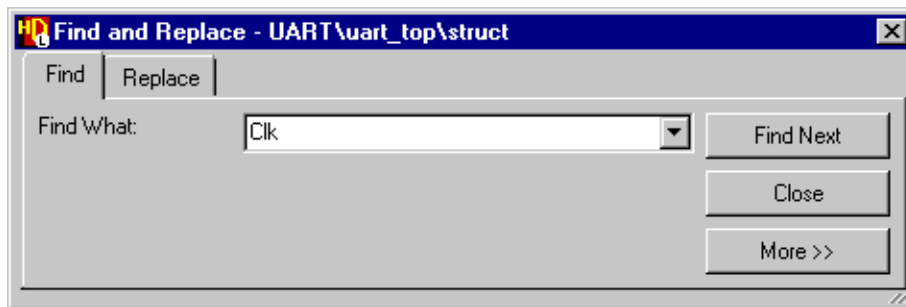
You can delete an object (or set of selected objects) from a diagram editor or table editor window by using the **Del** key or choosing the **Delete** command from the **Edit** or popup menus.

Unlike the **Cut** command, the erased objects are not copied to the clipboard but the delete operation can be undone during the current editing session.

When used in the **All** tab of an IBD view, the **Del** key deletes the selected row, column or text in all tabs. When used in a partial interconnect table, it deletes the selected objects from the active tab only. Alternatively, you can use the explicit **Delete from this table** or **Delete from all tables** commands.

Finding and Replacing Text Strings

You can search for a text string by using the  toolbar button, choosing **Find** from the **Edit** menu or using the **Ctrl + F** shortcut in any graphical window. The Find and Replace dialog box is displayed for you to specify a search string or choose from a dropdown list of text strings you have searched for in the current session.



If a text string is selected, it is used as the default search string in the dialog box. If the text has multiple lines, the first line is used. If the text string exceeds 40 characters, the search string is ended by the last space character inside this limit.

When you use the **Find Next** button or the **Enter** key, the next visible or hidden text object containing the specified string is selected.

More Search Options

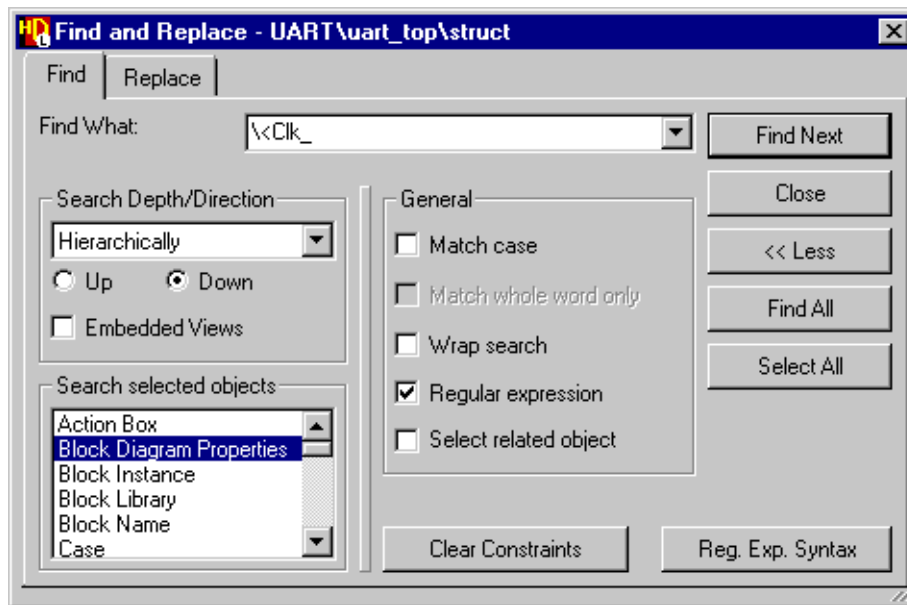
Additional search options are disclosed when you use the **More >>** button on the dialog box or can be hidden by using the **<< Less** button.

You can choose the search depth from the pulldown options **Current**, **Hierarchically** or **Through Components**, specify the search direction by choosing **Up** or **Down** and choose whether to search **Embedded Views**.

These options apply to your design hierarchy. If a flow chart or state machine in the search extent has hierarchical or concurrent views these are always searched. The **Up** or **Down** options are only available when you have chosen to search **Hierarchically**. If you choose to search up, the search stops when a symbol is reached.

You can choose to limit the search to one or more types of object by selecting from a list in the dialog box. (Use the **Clear Constraints** button to clear all selections from the object list.)

If you choose to search hierarchically, the constraints list includes all possible object types. For a non-hierarchical search, only object types for the active window are listed.



If you set the **Match case** option in the dialog box, a case-sensitive search is performed. For example, if you specified the string *Clk*, occurrences of *CLK* or *clk* would not be found. However, words containing *Clk* (such as *SysClk*) would be found.

You can also choose **Match whole words only** to restrict the search to occurrences where the search string is a complete word.

Note



The **Match whole word** option is not available when the **Regular Expression** option is set.

If you set the **Wrap search** option in the dialog box, the search repeats once all occurrences have been found. If unset, an "End of search" message indicates when all occurrences have been visited.

If you are searching hierarchically with wrap enabled, higher level views cannot be searched and once the bottom level view has been reached, wrap will be performed within the leaf level view only.

Find normally operates on simple text strings. However, if the **Regular expression** option is set, you can search for a regular or class expression instead of a text string. For example, the regular expression `\<Clk_` would find all strings starting with the characters *Clk_*.

You can display information about the supported regular expression and class expression syntax by using the **Reg exp Syntax** button.

If you set the **Select related object** option, the object related to the search string is selected instead of the text string itself. For example, the signal which matches a specified search string or the object associated with a comment text string.

When you use the **Find All** button, a scrollable list of matching text strings including the view and object name is displayed in a preview window. The status indicates whether the view is available for edit, read-only or locked by another user. You can select the actual string by double-clicking in the preview window and automatically opening the hierarchical view if necessary.

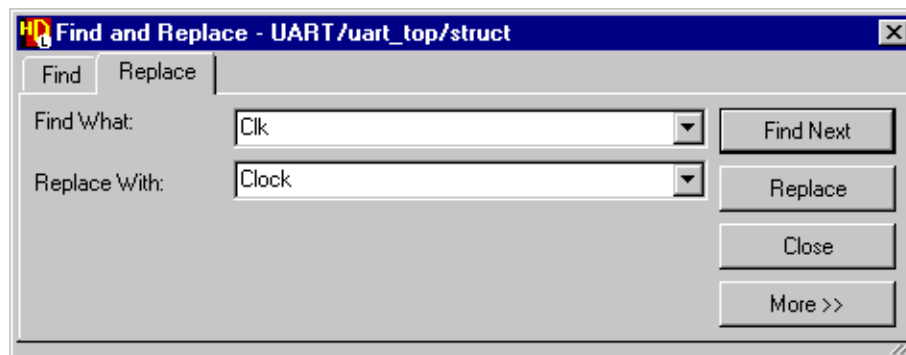
You can use the **Select All** button to select all the strings matching the specified string in the current view. The preview window is not updated when you use this button and the search hierarchically option is ignored.

If the search string is found outside the current window view, the window is panned so that the object containing the string is in the middle of the window. However, no panning is performed when the string is found within the current window view.

If the search string is not currently visible, it is made visible. If the search string is an editable characteristic which is not normally displayed (for example, the clock signal for a state machine), a Found dialog box is displayed indicating the text and location where it can be edited.

Replacing a Text String

You can replace a text string by choosing **Replace** from the **Edit** menu or using the **Ctrl + H** shortcut in any editor window. The **Replace** tab of the Find and Replace dialog box is displayed for you to specify the search string and a new string to replace it with.



You can choose from a drop down list of previous text strings you have searched for or replaced in the current session.

The **Replace** tab provides similar facilities to the **Find** tab with additional controls for replacing text strings. However, it only operates for replaceable text strings and any non-replaceable strings are ignored.

Always use the **Find** tab if you want to search for all occurrences of a text string. In general, any string that can be edited in place or by using a dialog box can be replaced.

You can choose to **Replace** the currently selected string or **Replace All** strings which match the current search constraints.

Note



Although you can choose to search for any GNU regular expression, only a literal text string or tagged expression can be entered in the **Replace With** box.

If the search string is an editable characteristic which is not normally displayed (for example, the clock signal for a state machine), a Found dialog box is displayed indicating the text and location where it can be edited.

When the Found dialog is displayed as a result of a replace operation, you can use the **Confirm** button to confirm the replacement or **Cancel** button to leave it unchanged.

Object Linking and Embedding

The HDL Designer Series diagram editor views (block diagram, state diagram and flow chart) support the Windows OLE (Object Linking and Embedding) standards and can be imported into any OLE compatible PC documentation tool including the Microsoft Office applications and Adobe FrameMaker.

Note



The table editor views (truth table, tabular IO and IBD view) support OLE only for documentation tools which recognize enhanced metafiles. These include the Microsoft Office XP tools but not older versions of Word or FrameMaker. Enhanced metafiles are also required to display vertical text in diagram editor views. If your documentation tool does not support enhanced metafiles, you may want to rotate any vertical text in diagram editor views to ensure that it will be correctly displayed in the documentation tool.

The interface used for importing OLE objects may vary between tools but is usually achieved by an **Insert Object** or **Import Object** command. Most OLE compatible tools also support direct import by "drag-and-drop". Refer to the user documentation for your documentation tool for information about importing OLE objects into your application.

If the diagram contains panels or multiple views (such as concurrent or hierarchical state machines) you are prompted which panel or view to insert.

The normal **File** menu and toolbar buttons for file operations are disabled when a diagram is edited as an OLE object from within the documentation tool. These are replaced by options to save a copy or return to the host document.

In order to import a design object using OLE, the *hds.hdp* project file which is used for library mapping information must be saved in a standard location which can be accessed from the documentation tool.

The default location is a user directory created beneath your *Profiles* directory. For example:

C:\Winnt\Profiles\<username>\Application Data\HDL Designer Series\
C:\Windows\Profiles\<username>\Application Data\HDL Designer Series\
C:\Documents and Settings\<username>\Application Data\HDL Designer Series\

Caution



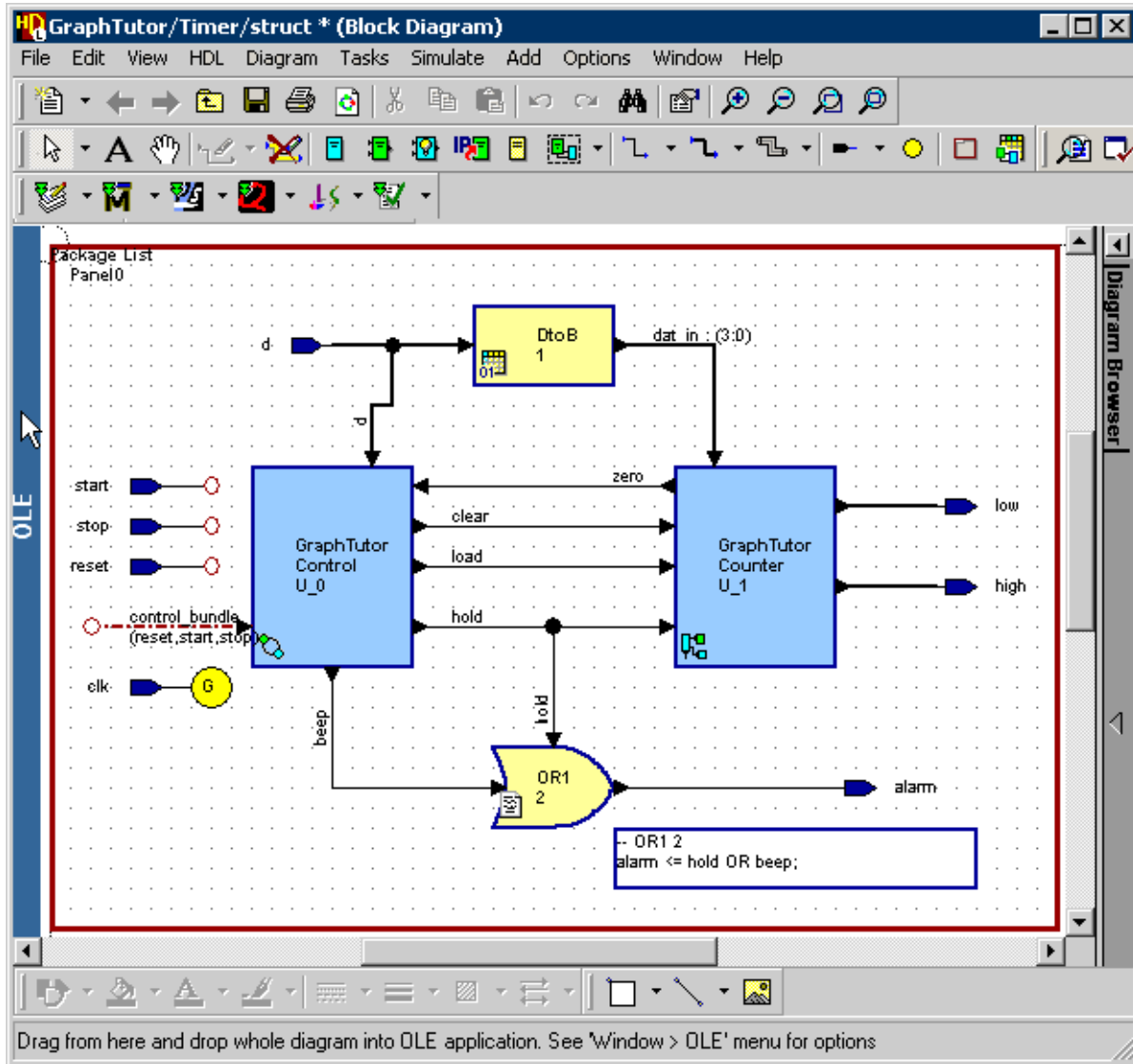
Initialization and preferences files saved in a working directory or at locations specified by the [HDS_LIBS](#) and [HDS_USER_HOME](#) environment variables cannot be used when you use the OLE facility.


If the embedded diagram is on a remote disk drive, you must ensure that full UNC pathnames are used in the project file. Pathnames using a mapped external drive are not supported when using OLE.


If a document containing an embedded or linked diagram is moved or copied to another workstation, the diagram can only be opened if a suitably licensed HDL Designer Series installation is available. If the document contains a linked diagram, the library containing the diagram must be defined in the *hds.hdp* project file on both workstations.

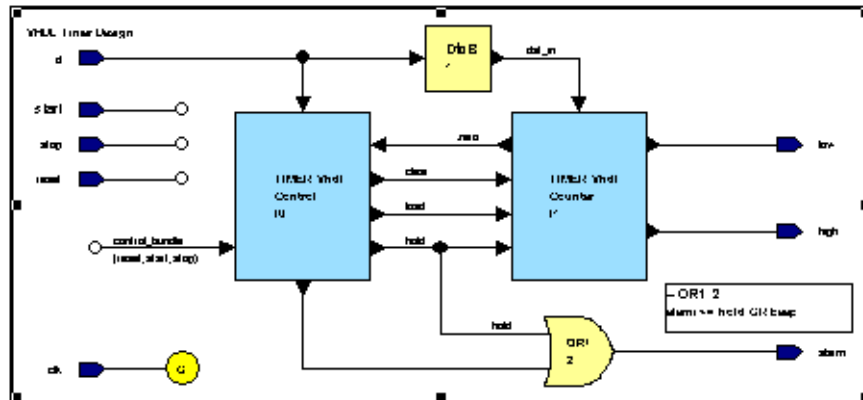
Using Drag and Drop

If your documentation tool supports OLE using drag and drop (for example, Microsoft Word or Adobe FrameMaker) you can import a diagram editor view from a graphical editor window by simply dragging and dropping it into the documentation tool window.



Click on the vertical blue drag bar at the left of the graphic editor window using the **Left** mouse button and drag with the button held down. The cursor changes to .

When you move over a valid destination in the documentation tool it changes to  and the graphic is inserted as an embedded object when you release the mouse button.





The entire diagram is inserted at the cursor location and may be clipped if it is larger than the document page size. However, you can resize the diagram by dragging its resize handles within the document window.

If the diagram contains panels (for example, *Panel0* in the example below) you can choose which panel to drag by clicking the **Right** mouse button over the gray border and selecting the required panel name from the **Set Drag Panel** cascade of the popup menu before performing the drag. You can also choose **Set Drag Panel** or reset **Set Drag All** from the **OLE** menu. The OLE drag bar can be shown or hidden by toggling **View Drag Bar** in the **OLE** menu.


Opening an OLE View


You can open a diagram or table view from within the document tool by double-clicking on the embedded OLE object. The normal editing capabilities are available if you have write permissions to the source files.

If a diagram contains multiple views (such as concurrent and hierarchical flow charts or state machines) or multiple panels you are prompted which view to edit.

If you open a tabular IO view from the OLE object, the port declaration table is normally displayed. You can access the generics table (in a VHDL view) by using the  button or choosing **Toggle Ports/Generics** from the **Table** menu. Similarly, you can access the parameters table (in a Verilog view) by using the  button or choosing **Toggle Ports/Parameters** from the **Table** menu. These commands are only available when the table is opened as an OLE object.

Generating HDL

You can generate HDL from graphical views by using the  button or by choosing one of the **Generate** options from the **Tasks** menu in any window.

You can use the  button to display a pulldown palette with options to run the task on a single design level, the hierarchy through blocks, the hierarchy through component or the hierarchy through components from the design root:



Run Single

Run Through Blocks

Run Through Components

Run Through Components from the Design Root

Corresponding commands are provided in the **Tasks** menu.

HDL is generated for any graphical views in the specified design hierarchy which have changed since HDL was last generated. However, you can force generation by setting **Set Generate Always** or **Generate All (One Shot)** in the **Tasks** menu.

Note



The generate always option remains set for the duration of the current work session (unless unset). The one shot option takes effect on the next generate command and is then automatically unset.

The design can include VHDL and Verilog views. Corresponding VHDL architecture and Verilog module files are generated in a single operation.

When a design explorer window is active, you can generate HDL for the selected object (or objects) or for the specified hierarchy of current views beneath the selected object(s).

When a block diagram or IBD view is active, you can choose to generate HDL for the hierarchy of current views beneath the active view. If a block or component is selected, the generation command operates on the selected object (or objects).

When a state diagram, flow chart or truth table is active, HDL is generated for the active view only unless you choose to generate from the design root.

If you choose to generate hierarchy from the design root, HDL is generated for all objects in the hierarchy below the design root. However, if no design root is set, HDL is generated for the hierarchy beneath the selected or active view.

Refer to “Setting the Design Root” in the *HDL Designer Series User Manual* for more information about the design root.

If no views have been defined for any block in the design hierarchy, a dummy HDL file is generated.

If a generated file would have the same name as an existing source HDL file, a Generation File Clash dialog box is displayed. You can choose to continue or cancel the HDL generation until you have checked whether both files are required. If you choose to continue, the existing HDL file is renamed with a *.replaced* suffix to ensure that no data is lost.

Note

A file name clash may occur when you migrate a pre-2003.1 design or if you have converted a HDL text view to graphics and made it the default view.

The HDL is created in the generated library directory specified by the library mapping which must exist for each library referenced by your design. Generated HDL files are given the extension specified as a preference in the **File** tab of the VHDL or Verilog Options dialog box.

The VHDL entity declaration and the VHDL architecture body can be generated as a combined file or you can choose to generate them as separate files by setting a preference in the **File** tab of the VHDL Options dialog box.

When the active view is a symbol and you have chosen to combine architectures and entities in a single file, the current view architecture is included when you generate HDL. If you have chosen separate files, only the entity is generated. The whole module is generated if you are using Verilog.

Refer to “Setting HDL File Options” of the [HDL Designer Series User Manual](#) for more information about VHDL and Verilog options.



Any generation errors are reported in the Task Log window. All views in the hierarchy to be generated must have been saved and you are prompted if any views need to be saved. Views which have not been changed since they were last generated are not generated unless you have set the HDL generation run options to force regeneration.

If no views need to be generated, a single dot is displayed for each processed hierarchy with a row of dashes to represent each selected hierarchy followed by a generation completed message.

You can set preferences to enable (or disable) HDL generation checks which carry out a full semantic error check on the generated code and optionally also report any warnings that are encountered.

Note

If an error is encountered during bulk parsing, the timestamp for the generated file is set to the oldest date allowed by the system. (January 1st 1970 on UNIX or January 1st 1980 on a PC). This ensures that the generated HDL file is retained with an older timestamp than the graphical view.

You can automatically display the generated HDL or graphics corresponding to an error message in the Task Log window by double-clicking on the message (or by explicitly clicking the  or  button).

Refer to “Task Log” in the *HDL Designer Series User Manual* for more information about the Task Log window.

If your design references a user-defined VHDL package, the application checks whether it has been compiled and compiles the package if required. Any package compilation errors are reported in the Task Log window and must be corrected before generation can be completed successfully.

VHDL Component Declarations

You can choose whether component declarations are created in the generated VHDL by setting **Create component declarations** in the **HDL** menu for *block diagram* and *IBD views*.

Refer to “Setting HDL Style Options” in the *HDL Designer Series User Manual* for information about using this option and setting its default.

Setting a Black Box for Synthesis


You can make the active *block diagram IBD view*, *state diagram*, *flow chart* or *truth table* a *black box* for synthesis by setting the **Black box for Synthesis** option in the **HDL** menu. When this option is set, synthesis control pragmas are included in the generated HDL so that the view is available for simulation but is ignored for synthesis.

For Verilog, the synthesis off pragma is inserted after the input/output statements and after any Verilog parameters declared in the symbol but before the type declarations. The synthesis on pragma is inserted immediately before the end module statement.

For VHDL, the synthesis off pragma is inserted after the architecture header and the synthesis on pragma after the end of the architecture.

Downstream operations (including synthesis) can also be disabled for a VHDL architecture view by using the "don't touch" property. However, "don't touch" cannot be used for Verilog or a combined VHDL entity and architecture view since the interface is included in the same generated HDL file. Refer to “Disabling Downstream Operations” in the *HDL Designer Series User Manual* for information about setting "don't touch" properties.

Viewing the Generated HDL

You can view the generated HDL for the active window by using the  button, **Ctrl + G** shortcut or choosing **View Generated HDL** from the **HDL** menu in any window. The text

editor is opened to display read-only views of the generated VHDL entity and architecture (or Verilog module) for the selected object.

In a block diagram or IBD view, the generated HDL for the active window is displayed if no blocks or components are selected. However, if one or more blocks or components are selected, the HDL for the current views of these design units is displayed.

If an object is selected and your text editor recognizes line number arguments, the generated HDL is opened at the line corresponding to the object. For example, if an embedded block is selected on a block diagram, the Source window is opened to display the corresponding code in the generated HDL for the block diagram.

You can also view the generated HDL files in the design explorer as described in the “HDL File Modes” section of the [HDL Designer Series User Manual](#).

Note

You should not normally edit the generated HDL since any changes will be overwritten the next time you generate HDL for the source design unit.

Chapter 2

Graphical Editor Windows

This chapter describes features that are available in the diagram and table editors. Later chapters describe features that are implemented in a particular editor.

Diagram Editor Windows	48
Setting Preferences for Diagram Views	48
Setting Diagram Master Preferences	49
Moving and Copying Diagram Objects	51
Resizing Objects	53
Arranging Objects	53
Adding Comment Text	56
Adding Requirement Reference Object	59
Editing Text on a Diagram	64
Editing Text in the Text Editor	66
Moving Text	67
Changing Text Visibility	68
Adding Comment Graphics	72
Adding a Title Block	77
Displaying Object Information	78
Panels	78
Editing Route Points	82
Setting Visual Attributes	83
Toggling the Grid Visibility and Snapping	86
Changing the Diagram View	86
Table Editor Windows	87
Setting Preferences for Table Views	87
Selecting Table Cells	88
Editing a Table Cell	88
Changing the Table View	89
Resizing a Column or Row	90
Exporting a Table	90
The Diagram Browser	91
Browsing Diagram Structure	92
Browsing Diagram Content	94
Signals Table	99
Signals Table Notation	100
Adding Port or Local Signal Declarations	102
Adding Comments to a Port or Local Signal Declaration	103
Resizing Columns	104

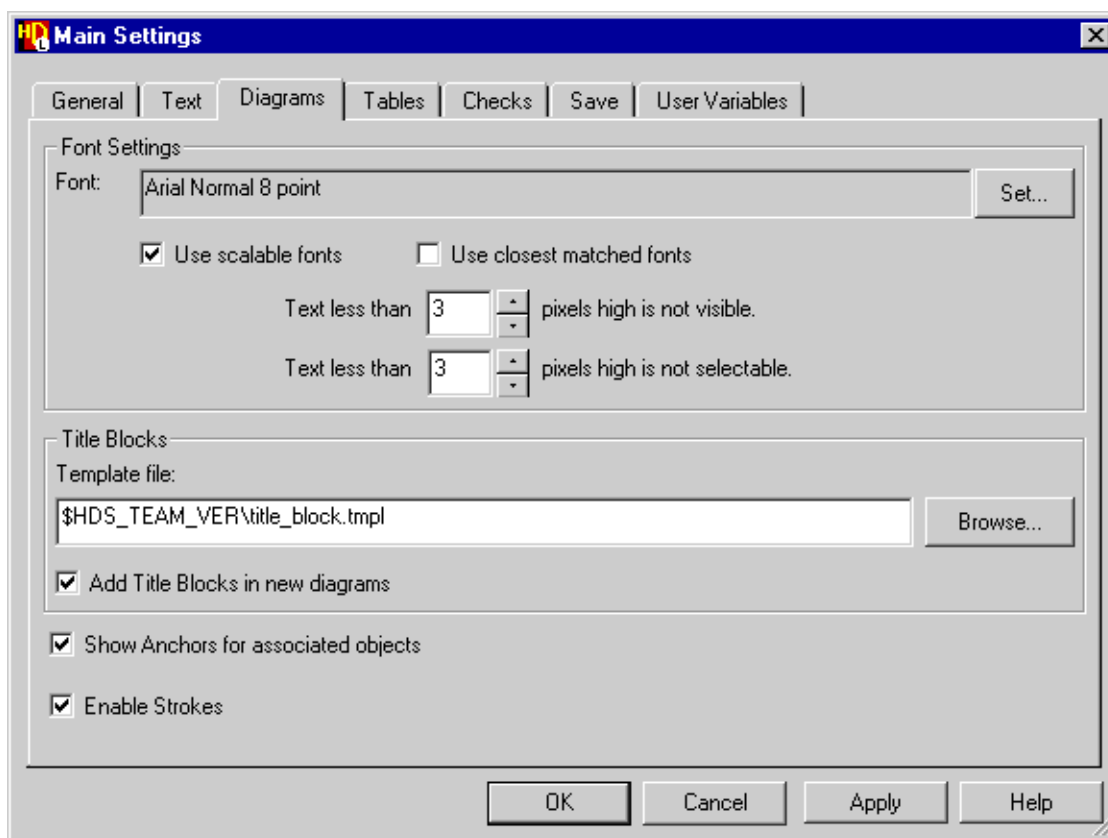
Hiding Columns	105
Filtering Columns	105
Grouping Signal Rows	105
Sorting Signal Rows	107

Diagram Editor Windows

This section describes features that are available in the *block diagram*, *flow chart*, *state diagram*, *ASM chart* and *symbol* diagram editors.

Setting Preferences for Diagram Views

You can set the preferences used for new *diagram editor* views in the **Diagrams** tab of the Main Settings dialog box which is displayed when you choose **Main** from the **Options** menu in any window.



You can change the default font by using the **Set...** button to choose from the fonts available on your system.

Scalable fonts are normally used. However, you can unset this option to use fixed size bitmap fonts on UNIX systems or true type fonts on PC systems.

You can also choose to match the default font specified in your preferences or in saved diagrams to the closest font available on your workstation.

You can specify the minimum number of pixel height for text visibility; if the text is less than the specified number, then it is not visible. Similarly, you can specify the minimum number of pixel height for text selectability; if the text is less than the specified number, then it cannot be selected. Note that the text selectability settings apply to the selection of single lines; in case of multiple lines, the pixel height you specified is automatically decreased by one pixel.

You can specify the location of the template used when you add a title block to a diagram and choose whether a title block is automatically included on new views. For more information about title blocks, refer to [“Adding a Title Block”](#) on page 77.

You can choose whether visible anchors are displayed connecting movable text selected on the diagram to its associated object.

You can also enable the stroke shortcuts which are described in [“Strokes”](#) on page 22.

Setting Diagram Master Preferences

You can change the master preferences for a *block diagram*, *symbol*, *flow chart*, *state diagram* or *truth table* editor view by choosing **Master Preferences** from the **Options** menu in the *design manager* window.

Note



The *table editors* for *tabular IO* do not have their own master preferences but use the same default values that are set for the symbol and block diagram.

You can change preferences for the active diagram by choosing **Diagram Preferences** from the **Options** menu in any *diagram editor* or truth table window.

Separate dialog boxes are used to set preferences for each editor but usually provide separate pages for setting the visual appearance of each graphical object, the default values used by new objects, the visual attributes of the background grid and other miscellaneous preferences for each editor.

State machine, flow chart and truth table preferences are described in the help topics that can be displayed by using the **Help** buttons which are available on each dialog box.

When you edit preferences for the active view, the dialog box allows you to choose whether the preferences are applied to new objects or to both new and existing objects in the diagram.

Note



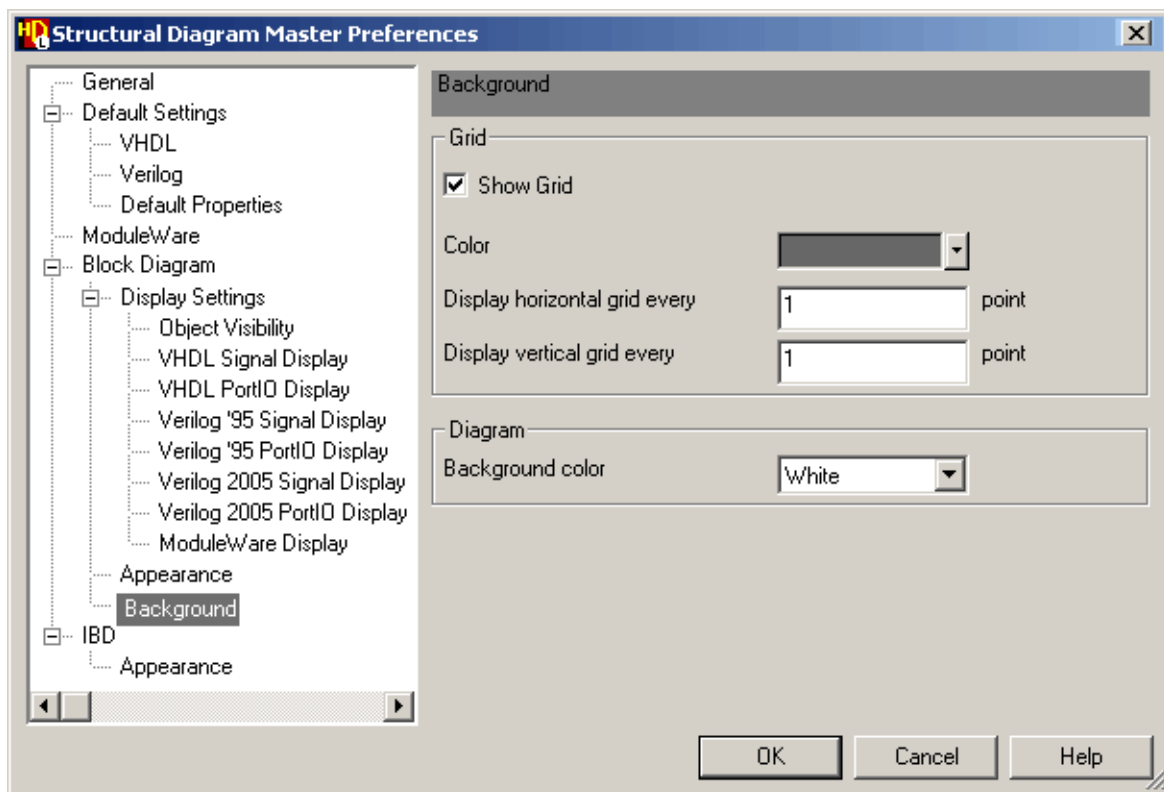
You can change appearance preferences in the active view but you cannot change preferences for default values, grid attributes or for many of the miscellaneous options. These preferences can only be accessed when you are editing master preferences from the [design manager](#).

The **Master Preferences** cascade menu in the graphic editor windows provides options to **Apply to New Objects** or to **Apply to New and Existing Objects** in the active diagram view. You can also choose **Update from Diagram** to update the master preferences using the current preferences from the active view. You are prompted for confirmation before the master preferences file is updated.

Refer to the “Default Preferences” appendix in the [HDL Designer Series User Manual](#) for lists of default preferences.

Setting Background Preferences


You can set preferences which control the background used for each diagram editor by using the **Background** page which is available in the master preferences dialog box for each editor.



You can choose whether the background grid is displayed or hidden.



Tip: You can also toggle the **Grid** display option for the active diagram from the **View** menu.

You can change the grid color by clicking on the  button to display a palette which allows you to choose from a set of 25 standard colors. You can also use the **Other** option on the palette to set color attributes using any of the colors available on your system.





The background grid is permanently set to 1000x1000 points. Grid snapping is always enabled for a symbol, block diagram or flow chart and cannot be unset. In a state diagram you can choose to unset the **Snap to Grid** option in the **View** menu.

You can choose a different granularity for the display of horizontal and vertical grid points. For example, setting the display factors to 10 would display every tenth grid point. The default display granularity is 1 in both the horizontal and vertical directions.

The background preferences can be set as defaults in the diagram master preferences dialog box or can be applied to the active diagram when accessed from the diagram preferences dialog box.

Moving and Copying Diagram Objects

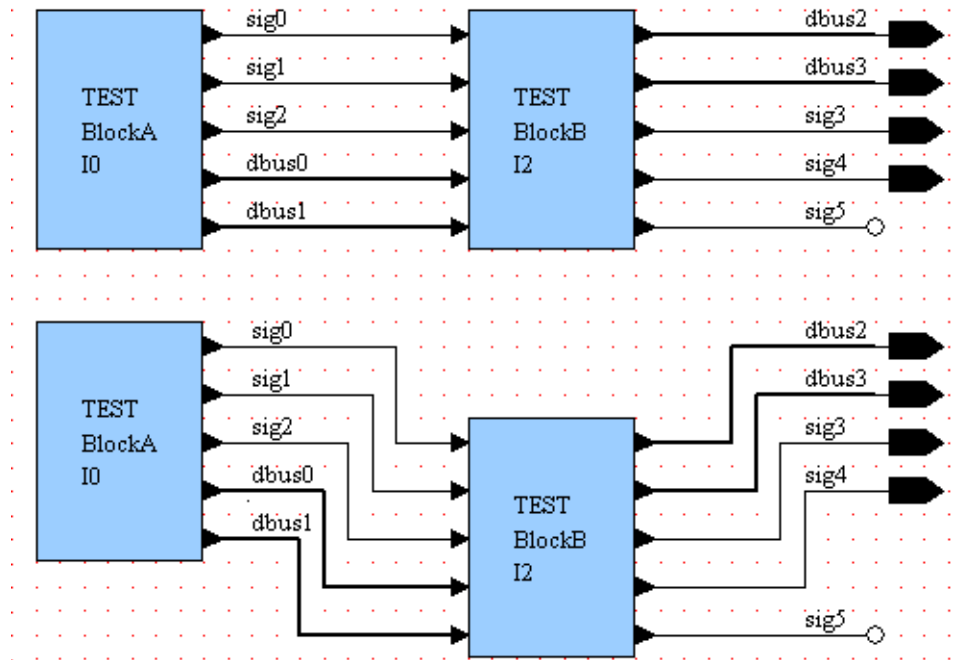
You can move an object (or group of selected objects) in a *diagram editor* by simply pressing down the **Left** mouse button and dragging the selected objects to a new position.

You can move a object to the next grid space by using the , ,  or  keys with the **Ctrl** key.

You can move a single segment of a net or flow in a block diagram or flow chart while retaining connectivity with adjacent orthogonal segments. You can also move a group of selected segments in different nets or flows while retaining the same relative separation and connections to unselected objects. Any attached ports in a block diagram are automatically moved with the selected segment.

Orthogonal routing is preserved when you move an object and route points added to nets or flows where necessary to preserve connectivity. Ports attached to an object are not moved but a dangling net connector will move with the attached net.

For example, the following picture shows the effect of moving a block in a block diagram.



In a state diagram, you can change the shape of a transition arc by dragging an individual route point to a new position with the **Left** mouse button.

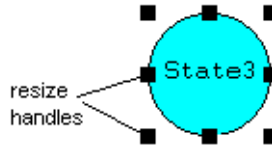
You can also move an object (or objects) by dragging with the **Right** mouse button and choosing the **Move Here** option from the popup menu to move the object to the position of the cursor. The popup menu also provides an option to **Copy Here** which creates a copy of the selected objects at the new position and a **Cancel** option to abort the copy or move operation.

You can use the **Shift** key with the **Left** or **Right** mouse button to constrain the movement to horizontal or vertical directions. For example, to prevent accidentally changing the length of a net during a horizontal or vertical move.

In a block diagram or flow chart, you can use the **Alt** key with the **Left** or **Right** mouse buttons to select an individual segment of a net or flow (or **Alt+Shift** or **Ctrl** and **Left** or **Right** to select multiple segments) and then drag the selected segments. Any connected but unselected segments are not moved but may be stretched to maintain connectivity.

Resizing Objects

Any diagram editor object can be resized by selecting the object and dragging one of its resize handles. However, an object which includes text cannot be resized to be smaller than the text it encloses.



When you resize a circular object (such as the state shown above) the shape is preserved.

When you resize any other shape (such as a rectangle or polygon) you can use the middle handles to resize horizontally or vertically and the corner handles to resize in both directions simultaneously. You can also use the **Shift** + **Left** mouse button to preserve the aspect ratio while resizing.

Comment text or comment graphics (or grouped comment text and graphics) can be resized, but the resize may be constrained by the grouped objects. For example, the aspect ratio of a circular comment graphics object must be preserved and the aspect ratio of group containing a circle is therefore constrained to preserve the aspect ratio of all objects in the group.

Arranging Objects

Any diagram object (or group of objects) can be aligned, distributed, rotated, flipped, layered or grouped with other diagram objects.





Arrange Object Toolbar




The following commands are available from the Arrange Object toolbar in the block diagram, state diagram, flow chart and symbol editor:

Table 2-1. Arrange Object Toolbar

Icon	Description
	Aligns the selected objects
	Distributes the selected objects
	Flips the selected objects horizontally about a vertical axis
	Flips the selected objects vertically about a horizontal axis
	Rotates the selected objects 90 degrees clockwise
	Rotates the selected objects 90 degrees anti-clockwise

Table 2-1. Arrange Object Toolbar (cont.)


Icon	Description
	Brings the selected objects to the front
	Brings the selected objects to the front
	Groups the selected objects
	Ungroups the selected objects




The  and  buttons perform a default command or you can use the  button to display a pulldown palette and choose the required operation. This choice then becomes the default operation (indicated by the icon shown on the button) until another operation is chosen.

The toolbar can be displayed or hidden by setting the **Arrange Object** option in the **Toolbars** cascade of the **View** menu.




Refer to “[Toolbars](#)” on page 20 for more information about toolbars.

Aligning or Distributing Objects

You can align the currently selected objects using the  button or by choosing **Align** from the **Edit** menu in a block diagram, state diagram or flow chart. The toolbar palette or cascade menu provides options to **Align Left**, **Align Center**, **Align Right**, **Align Top**, **Align Middle** or **Align Bottom**.



For example, if you choose  from the palette or **Align Left** from the menu all objects in the current selection set are aligned with the left edge of the left-most object. If you choose  or **Align Center** the selected objects are moved horizontally and aligned vertically about their vertical axis or if you choose  or **Align Middle**, the selected objects are moved vertically and aligned horizontally about their horizontal axis.

In general, text is moved with its associated object although selectable text groups (such as the signal name and type on a block diagram or actions text on a state diagram) can be aligned separately from their associated objects.

The **Align** cascade menu also includes options to **Distribute Horizontally** or **Distribute Vertically** which can be used to arrange the selected objects so they are equal distances from each other in the vertical or horizontal direction. These options are also available by using the  button and choosing  or  from its pulldown palette.


Rotating and Flipping Objects


When any non-symmetrically shaped object (or objects) is selected, the **Edit** and popup menus include a **Rotate** cascade which allows you to rotate the shape clockwise by **90**, **180** or **270** degrees. However the menu options are not available for symmetrical shapes such as circles.

You can also use the  button to rotate the selected objects by 90 degrees clockwise or the  button to rotate them by 90 degrees anti-clockwise.

Note



You cannot rotate any objects (other than comment text) on a flow chart and you cannot rotate the package list, compiler directives list or declarations text on any diagram.

You can flip an object horizontally about its vertical axis by using the  button or choosing **Horizontal** from the **Flip** cascade in the **Edit** or popup menus.

You can flip an object vertically about its horizontal axis by using the  button or choosing **Vertical** from the **Flip** cascade menu.

Layering Comment Text and Graphics


In general, if you add an object overlapping another object the new object is added over the existing object. However, you can change the layer of the selected comment graphics or comment text (or grouped comment text and graphics).

You can bring an object to the front using the  button or by choosing **Bring to front** from the **Layer** cascade of the popup menu. You can also push an object to the back using the  button or by choosing **Send to back** from the **Layer** cascade menu.

Only two layers are supported. If more than one object is overlapped, sending an object to the back may cause an overlapped object to be brought in front of other objects it previously overlapped.

If you send comment text to the back, the background color is moved behind any overlapping object but the text always remains in the foreground. Similarly, although you can place comment graphics in front of another diagram object (such as a block or component), the text associated with the object (for example, the name of the block) is always on the top layer.


Grouping Comment Text and Graphics

You can group one or more selected comment text or comment graphics objects using the  button or by choosing **Group** from the **Group** cascade of the popup menu.

Once grouped, all text and graphics in the group can be cut, copied, pasted, aligned, moved, hidden or deleted as one object.

Grouped comment graphics can also be resized, rotated or flipped although the aspect ratio must be preserved if the group contains circular objects. Grouped comment text can still be edited individually but a group containing comment text cannot be rotated.

Grouped objects do not need to be adjacent. For example, you could define a title block by grouping a heading comment text at the top of your diagram with a footer comment text containing version information near the bottom of the diagram.


You can ungroup objects by selecting the group and using the  button or by choosing **UnGroup** from the **Group** cascade of the popup menu. All comment text and graphics text objects are ungrouped although the group may have been created by grouping several existing groups.

Note



You cannot attach grouped comment text to another diagram object, although you can choose to include the group at the file start, in the file header or at the end of file in the generated HDL.

Adding Comment Text

You can use the  text tool button, the **Ctrl + F1** shortcut or choose **Comment Text** from the **Add** menu to add comment text as annotation on a block diagram, flow chart, state diagram or symbol.


The cursor changes to a cross-hair which allows you to open a text entry box by clicking at an empty location anywhere on the diagram and enter free-format text. Any line feeds or blank lines you enter are preserved on the diagram.

The text edit box has its own scroll bars which allow the text to be scrolled horizontally or vertically while editing.




To complete the text edit and return to the normal select mode, click the **Left** mouse button outside the text object. The text is drawn using the default font and can be moved, copied or resized in the same way as any other diagram object.

If the text is larger than can be displayed in the default box, this is indicated by the label <<--more -->>. However, you may want to resize the text object in order to fully display the enclosed text.

You can abort text mode without saving the text by clicking the **Right** mouse button or using the  button to return to the normal select mode.

You can change the appearance of comment text by setting visual attributes for the selected text or change the default appearance by setting default visual attributes for comment text in the diagram master preferences.

You can edit comment text directly, send it to the text editor or use the **Comment Text** page of the Object Properties dialog box which can be displayed using the  button, **Alt+ Enter** shortcut or by choosing **Object Properties** from the **Edit** menu when the text annotation is selected on the diagram.

When a comment text object is selected, you can attach it to any diagram object by choosing **Include in HDL** from the popup menu and selecting the **Before Object**, **After Object** or **End of Line** cascade option. An *anchor* is attached to the cursor which you can terminate on any other diagram object by clicking the **Left** mouse button over the required object. Comment text attached in this way is included as comments in the generated HDL next to the appropriate HDL for the associated object.

Note

You can attach separate comment texts containing synthesis pragmas before and after an object to include the pragmas in the generated HDL and turn synthesis processing off for the object.

You can associate comment text with the interface to a block by selecting the **In Block Interface** cascade option and attaching the anchor to a block. When this option is used, the note is included as comments in the generated VHDL entity or Verilog module for the block.

If the block is subsequently converted to a component, any block interface comments are moved into the symbol for the component. Any comments added before or after a block object are included before or after the instantiation statement for the parent diagram.

You can also associate comment text with the diagram itself by selecting the **At File Start**, **After File Header** or **At File End** cascade option. Comment text attached in this way is included as comments at the specified position in the generated HDL for the diagram.


You can detach comment text from an object by selecting the text and choosing **Do Not Include in HDL** from the popup menu. Detached comment text is ignored by HDL generation.

Note

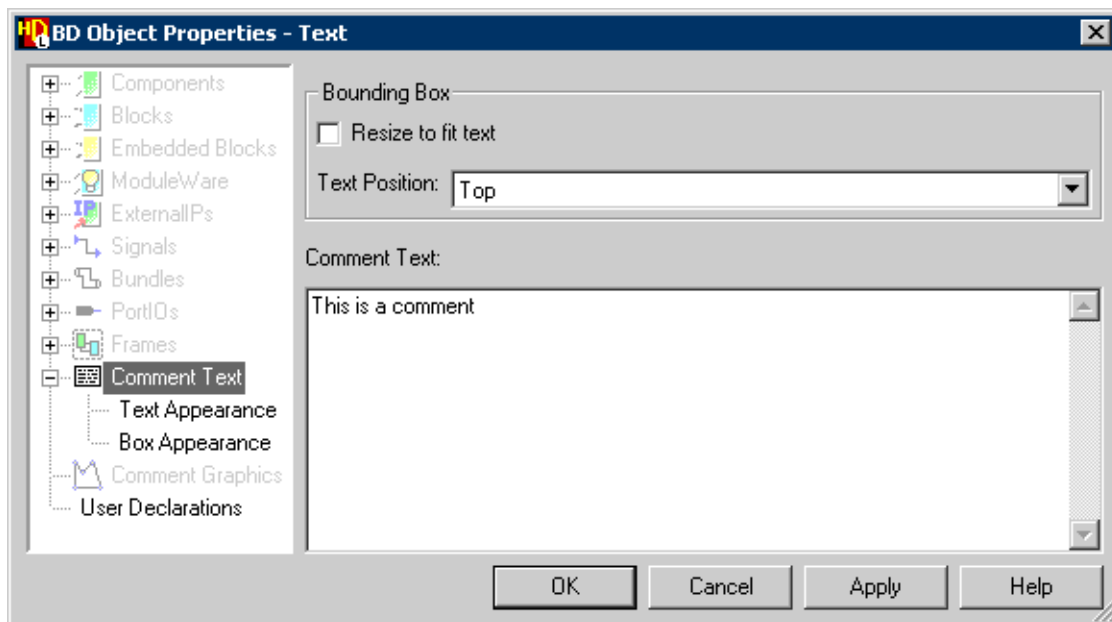
In a flow chart, comment text attached to the start point is included at the beginning of the generated HDL and comment text attached to the end point at the bottom. Comment text associated with a child diagram in a hierarchical flow chart can be used to label the HDL generated from the diagram. However, comment text in a state machine associated with a child state diagram is always added at the top of the generated HDL because the state machine is flattened before generation.

You can hide the selected comment text (or comment graphics) object by choosing **Hide** from the popup menu or from the **Comment Graphics** cascade in the **Diagram** menu and show all hidden comment text or graphics objects by choosing **Show All**.

Editing Text Properties

You can edit existing comment text by using the  button, **Alt + Enter** shortcut or choosing **Object Properties** from the **Edit** menu. If a single comment text object is selected on the diagram, the **Comment Text** node of the Object Properties dialog box is enabled and the Comment text page is displayed showing the existing text.

The Comment Text page allows you to resize the bounding box to fit the text and choose the text position from a pulldown list.



Any free-format text can be entered in the dialog box. Comment text can optionally be associated with the document header or attached to any diagram object and included as comments when HDL is generated for the diagram.

If the popup option to include the text comment **After File Header** is set, you can choose whether to **Add comment characters** before the text strings. If this option is unset, the text is treated as a HDL statement in the generated HDL.

You can include internal or user-defined variables (which are interpreted when the text is applied and automatically updated when you save the diagram). When you enter text using the dialog box or text editor, the text may also include any special characters (including for example, Kanji characters) if they are available on your system.

Note

The `%(d)`, `%(f)`, `%(p)`, `%(library)`, `%(unit)` and `%(view)` variables are only interpreted after the diagram has been saved and are shown as `<TBD>` in unsaved views.

Refer to “Using Internal Variables” in the [HDL Designer Series User Manual](#) for more information about internal variables.

You can protect a comment text (or grouped comment text) from changes to the diagram or master visual attribute preferences by unsetting the **Allow Visual Change** menu option in the popup menu (or **Text** cascade in the **Diagram** menu). This can be useful when you have saved comment text as a title block with different visual attributes from other comment texts on the diagram.

You can use the **Modify** check box to apply the new text to all selected comment text or embedded HDL text views on the diagram.

The **Text** page can also be used to edit an embedded HDL text view on a block diagram. When used to edit HDL text, and the syntax checking preference is enabled for block diagrams, the text is automatically syntax checked for the current hardware description language.

Caution

Internal variables are not interpreted when the **Text** page is used to enter HDL text. For example, `%d` is treated as an internal variable (translated as the current design unit) in comment text but as a Verilog variable (translated as a decimal value) in a Verilog HDL text view.

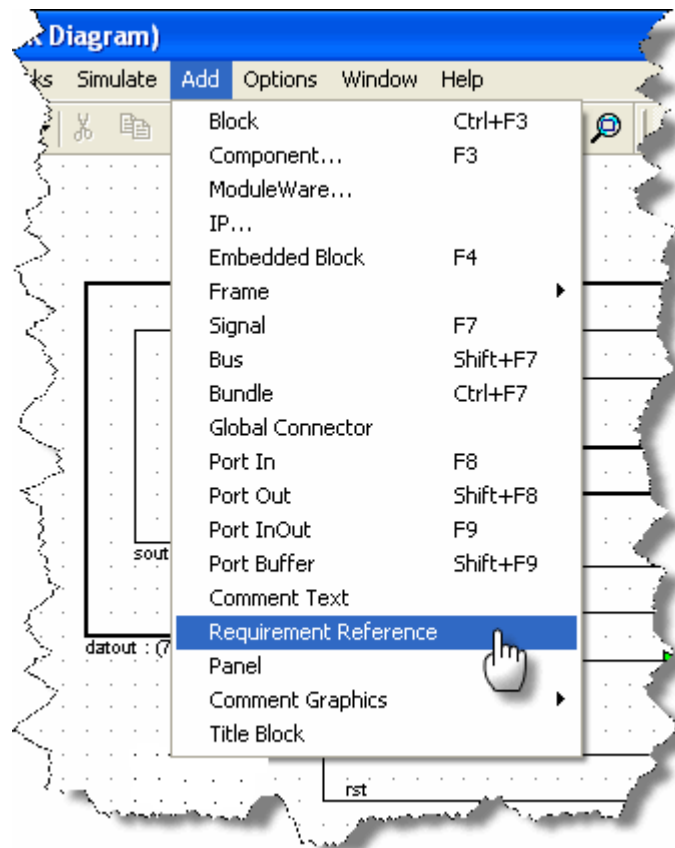
Adding Requirement Reference Object

This feature is disabled by default (along with all the features related to requirements tracing). You can add or paste a requirement reference only by enabling requirements referencing in the Requirements Referencing Settings dialog box which can be accessed by choosing **Requirements Referencing** menu item from the **Options** menu in the Design Manager of the HDL Designer Series. Refer to the [Enabling Requirements Referencing in HDS](#) in the [HDL Designer Series User Manual](#) for details on enabling requirements referencing.

There are different ways for adding requirements references to the design in one of the graphical editors (Symbol, Block Diagram, Finite State Machine, Algorithmic State Machine and Flow Chart excluding IBD which has a different approach and the Truth Table which doesn't support adding requirements). There are mainly two methods: Adding and Pasting the requirement reference object.

The Requirement Reference object is similar to the Comment Text box. Requirements references are written inside it and generated as comments. You can attach this object to any object in the diagram that can have a requirement associated with it.

To do this, choose **Add > Requirement Reference**.



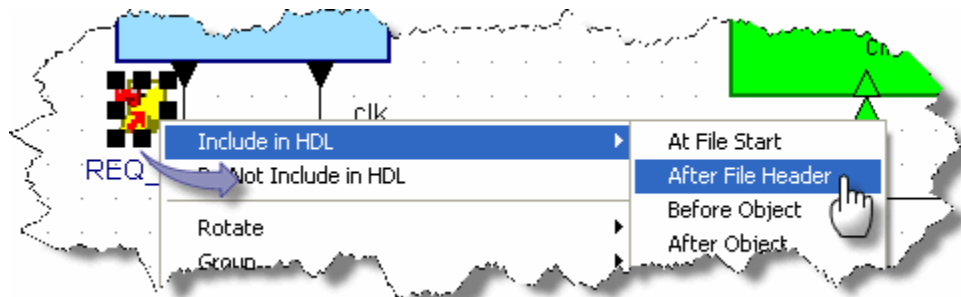
The cursor changes into a cross and you can click anywhere on the diagram to add the Requirement Reference object.

Write down the requirement reference into the displayed object. The requirement reference object has its own scroll bars which allow the text to be scrolled horizontally or vertically while editing.

To finish and return to the normal select mode, click the **Left** mouse button outside the requirement reference object.

You still have to attach the created object to any object in the diagram or in a specific location in the generated HDL file which is done by right clicking on the requirement reference object and

choosing an option from the cascade menu of the **Include in HDL** menu item in the popup menu.



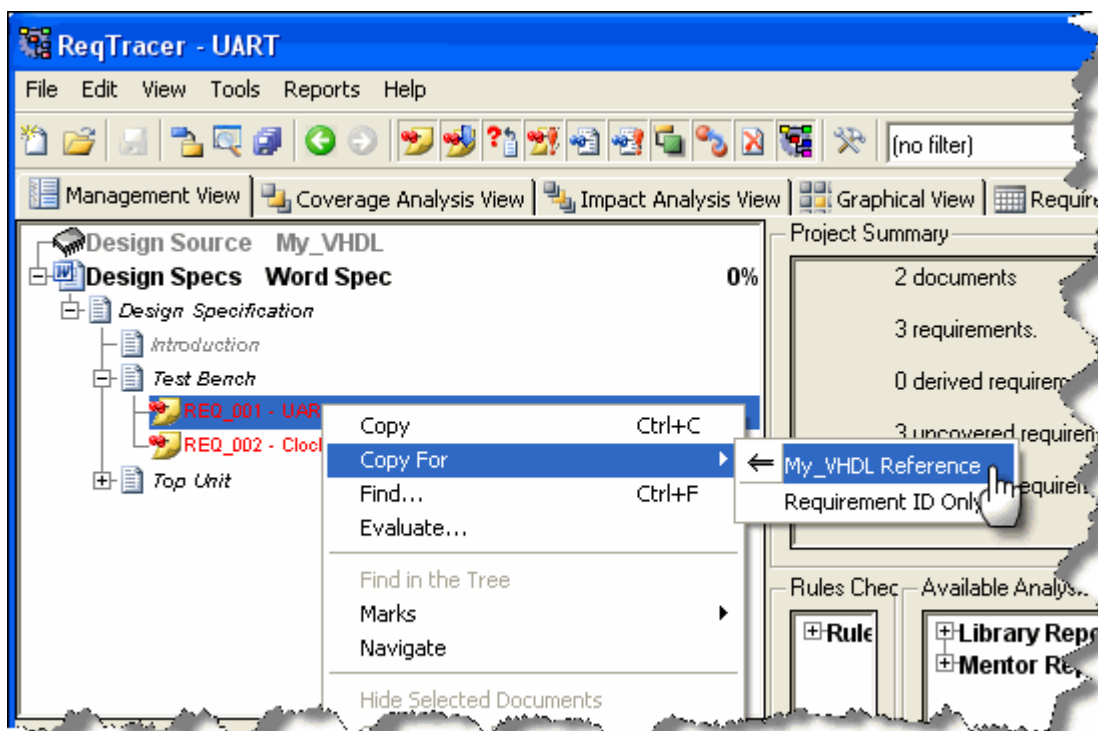
There are other ways by which requirement references can be added to your design; namely, [Pasting in Editor](#) and [Pasting in Design Browser](#).

Pasting in Editor

This method enables you to add requirement references to your graphical editor by copying the requirement from ReqTracer and pasting it onto a specific object in your design.

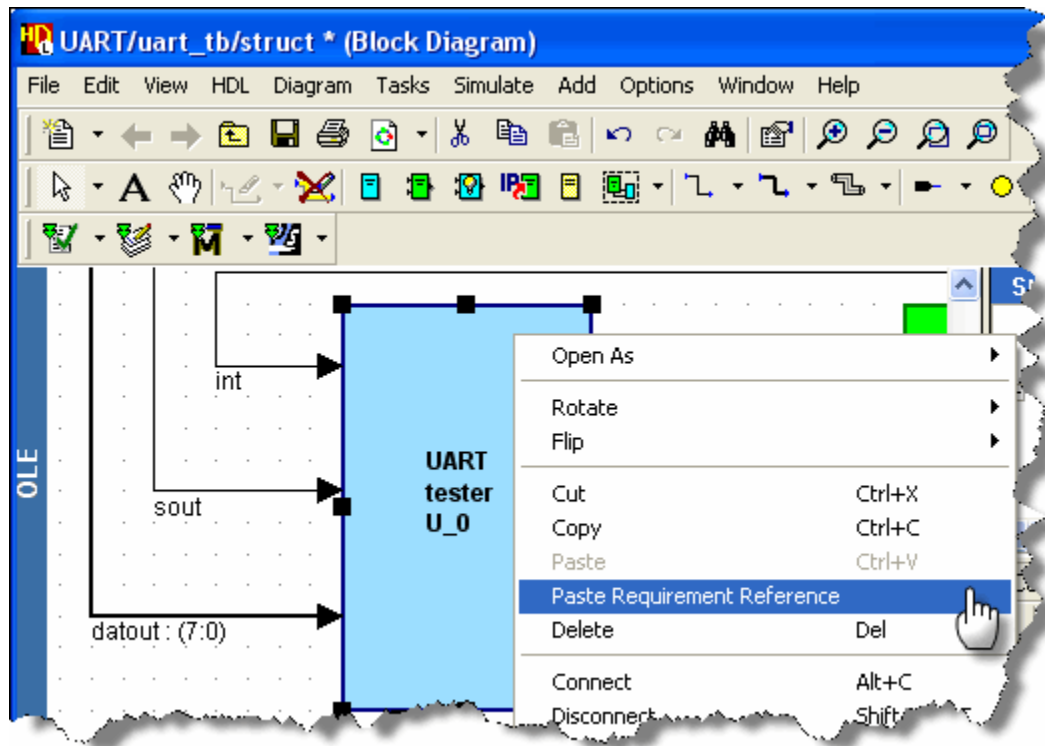
To copy the requirement from ReqTracer, do the following:

1. RMB on the requirement.
2. Choose **Copy For > My_VHDL Reference** from the popup menu.



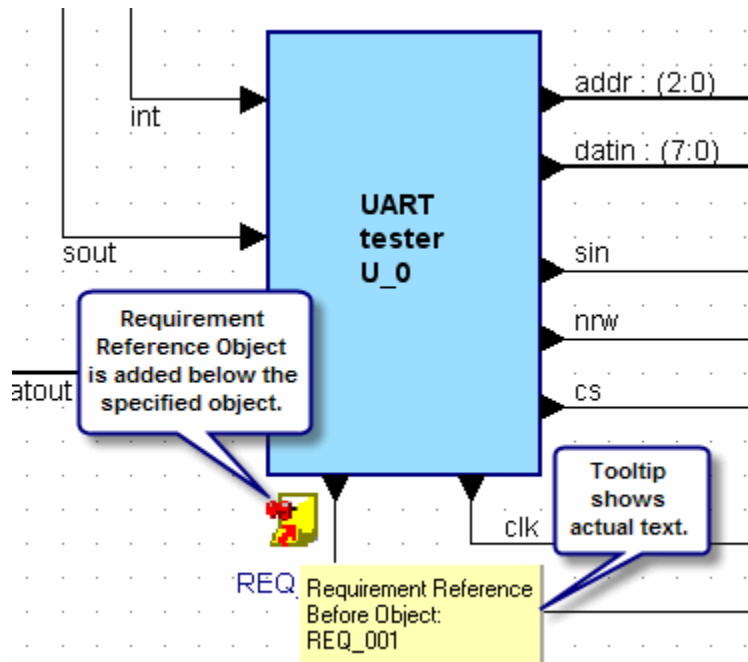
To paste the requirement reference onto a specific object in your design in HDS, do the following:

1. RMB on the desired object.
2. Choose **Paste Requirement Reference** from the popup menu.



The Requirement Reference object is added to your design next to the object you pasted it onto. Actual text of the requirement reference is shown under the Requirement Reference object.

Furthermore, if you hover with the mouse over the requirement reference object, the text is shown in the tooltip. The actual text is also visible in the Content Pane.



When the file is generated, the requirement references are shown as comments. The location where they appear in the generated file is specified in the **Object Default Generation Location** section of the “Requirements Referencing Settings Dialog Box” which is displayed by choosing **Options > Requirements Referencing** from the standard toolbar of the Design Manager.

Related Topics

- [Adding Requirement Reference Object](#)
- [Enabling Requirements Referencing in HDS](#)

Pasting in Design Browser

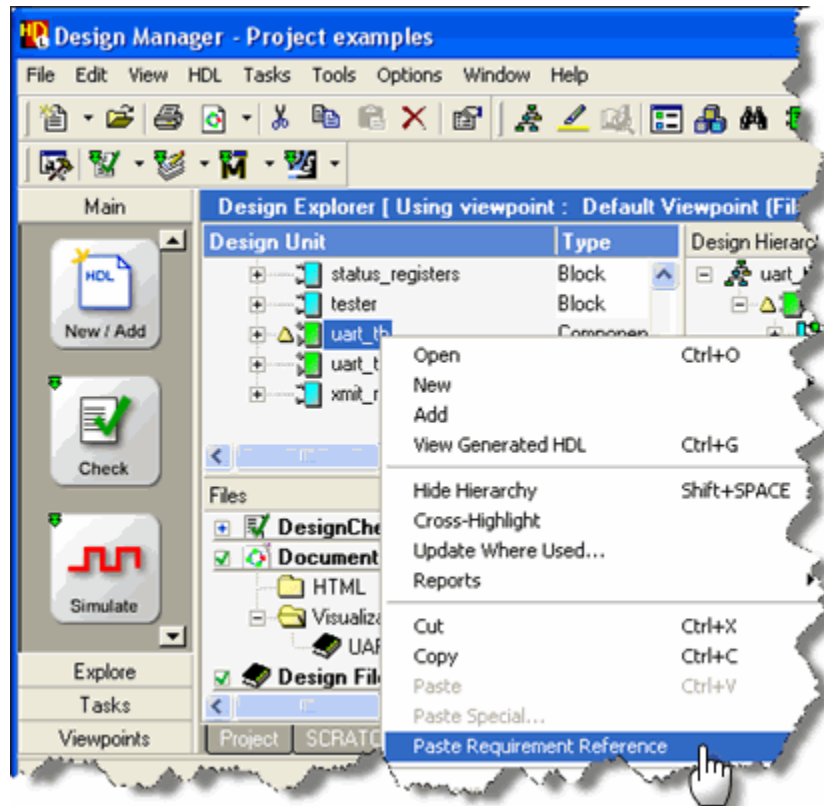
This method enables you to add requirements references to your design by copying the requirement from ReqTracer and pasting it into the design file from the Design Explorer without actually opening the file.

To copy the requirement from ReqTracer, do the following:

1. RMB on the requirement.
2. Choose **Copy For > My_VHDL Reference** from the popup menu.

To paste the requirement reference into the design file from the Design Explorer in HDS, do the following:

1. RMB on any design object (design unit, view, file...) that can have requirements associated with it.
2. Choose **Paste Requirement Reference** from the popup menu.



The requirement reference object is added to the file of the selected object without the file being opened.

When the file is generated, the requirements references are shown as comments. The location where they appear in the generated file is specified in the **File Default Generation Location section** of the “Requirements Referencing Settings Dialog Box” which is displayed by choosing **Options > Requirements Referencing** from the standard toolbar of the Design Manager..

Related Topics

- [Adding Requirement Reference Object](#)
- [Enabling Requirements Referencing in HDS](#)

Editing Text on a Diagram

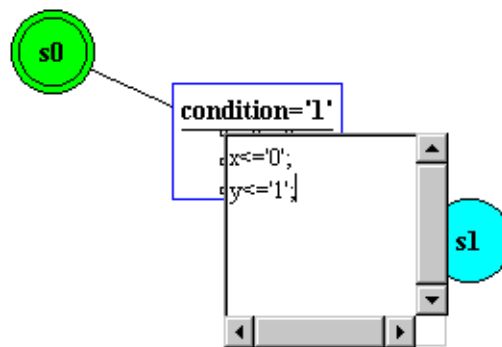
The text associated with any graphic object can be edited directly on the diagram in the following way:

1. Click on the text to select it.

2. Click on the selected text again. The selected text is highlighted and can be replaced by overwriting with new text. If you click again, the cursor changes to an I-beam and you can edit the existing text.
3. After editing, click outside the text to return to select mode.

There may be more than one text element associated with a graphic object and each text element can be edited separately. For example, a signal has a name and type; a block has a library name, a block name and an instance name; a transition may have condition and actions text.

Some text elements (such as the text in a comment text object, the HDL text for an embedded block on a block diagram or actions text in a state diagram) are multi-line objects. When one of these objects is edited, a popup text edit box allows you to enter free-format text.



Note



Tab characters are interpreted as spaces using the current setting of the tab width preference used for VHDL or Verilog views.

After editing, click with the **Left** mouse button outside the text object (or click anywhere with the **Right** mouse button) to return to select mode.

Text Editing Shortcuts

The following standard keyboard shortcuts are defined for in-place text editing:

Table 2-2. Text Editing Shortcuts












Shortcut	Description
Double-click Left button	Selects the word under the cursor
 + 	Selects all text in the selected object
 +  or 	Copy selected text to the clipboard
 +  or 	Paste text from the clipboard
 +  + 	Move selected text to the clipboard

Table 2-2. Text Editing Shortcuts (cont.)

Shortcut	Description
	Delete selection or delete next character if none selected.
	Delete selection or delete previous character if none selected.
	Move one character to left
	Move one character to right
	Move up one line
	Move down one line
+	Move one word to left
+	Move one word to right
+	Go to end of text box
+	Go to beginning of text box
+	Extend selection by one character to left
+	Extend selection by one character to right
+ +	Select to beginning of word
+ +	Select to end of word
+	Extend selection to next line above
+	Extend selection to next line below
+	Extend selection to beginning of line
+	Extend selection to end of line
	Terminate edit

The , and keys are only available on Sun workstations.

Editing Text in the Text Editor

You can send multi-line text such as comment text or embedded HDL text on a block diagram to the current text editor (or the current HDL viewer in a read-only diagram) by choosing **Send to Editor** from the popup menu or from the **Text** cascade of the **Diagram** menu.

If the ModelSim simulator is invoked, this command is replaced by **Send to Source** which opens the temporary HDL text file in the ModelSim Source window instead of using the current text editor.

The text is exported to a temporary location in the temporary directory specified by the TMP (Windows) or TMPDIR (UNIX) variable. (If these variables are not set, the root directory of the

current drive is used on Windows or `/usr/tmp` on UNIX.). All exported text files for the current session are saved in a temporary subdirectory which is deleted when the session is exited.

Text which has been exported to the text editor shown as dark gray text with a light gray background and cannot be edited directly (or from the Object Properties dialog box) until you save any changes in the text editor and choose **Finish Edits** from the menu to re-import the modified text.

If more than one text element is exported to the text editor, you can choose **Finish All Edits** to re-import all changed text.

If you save the diagram editor view while any text is exported, the tool will attempt to import the edited text automatically.

The syntax of Imported text is checked for the current diagram language unless the syntax checking preference has been unset in the diagram master preferences.

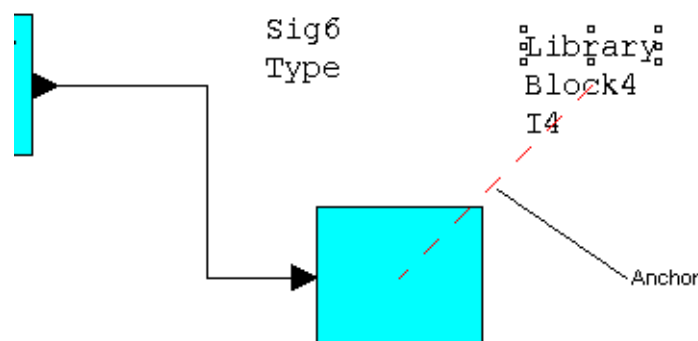
Moving Text

Many text elements (or grouped text elements such as the name, bounds and type of a net) can be moved independently from their associated object.

For example, block or component name text can be moved outside the object and transition or signal text can be moved from its default position.

The text is connected to its associated object by an *anchor*. The anchor is visible when the text is selected unless this preference is unset in the **Diagrams** tab of the Main Settings dialog box as described in “[Setting Preferences for Diagram Views](#)” on page 48.

If the associated object is moved, the text is also moved maintaining the same offset position from the object.



The following anchored text elements can be moved in this way:

Block Diagram

block	block, library and instance name
component	component, library and instance names port name, bounds and type VHDL generic or Verilog parameter values
signal	signal name, type and initial value or delay
bus	bus name, slice, type, bounds, initial value (or delay)
bundle	bundle name
embedded block	name, number and HDL text

Symbol

body	symbol name port name, slice, type, bounds, initial value (or delay) VHDL generic declarations, Verilog parameter declarations
------	--

State Diagram

state	state actions
transition	transition text (including conditions and actions)
link	link name

Flow Chart

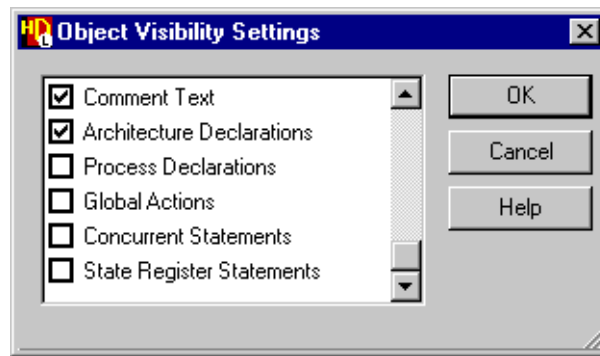
action box	name actions
decision box	name condition
case box	name expression port name
wait box	name statement
loop	name (anchored to start loop object)

Changing Text Visibility

You can change the visibility of multi-line text objects in a diagram editor using the Object Visibility Settings dialog box.

The dialog box is displayed when you choose **Object Visibility** from the background popup menu by using the mouse button when nothing is selected.

For example, the Object Visibility Settings dialog box in a VHDL state machine:



Note



The comment text option is available in the dialog box only when visible or hidden comment text exists in the diagram.

The Object Visibility Settings dialog box controls whether the following multi-line text objects are displayed on a graphical diagram:

	VHDL	Verilog
Block Diagram	Package List Port and Signal Declarations Comment Text	Compiler Directives Port and Signal Declarations Comment Text
State Diagram	Package List Architecture Declarations Global Actions Concurrent Statements State Register Statements Process Declarations Comment Text	Compiler Directives Module Declarations Global Actions Concurrent Statements State Register Statements Comment Text
Flow Chart	Package List Architecture Declarations Concurrent Statements Sensitivity List Process Declarations Comment Text	Compiler Directives Module Declarations Concurrent Statements Sensitivity List Local Declarations Comment Text

Many diagram objects have associated text which can be optionally hidden or displayed. You can hide individual text elements by choosing **Hide Text** from the popup menu (or from the **Text** cascade of the **Diagram** menu).

You can make all text elements associated with an object (including text that is hidden by default) visible by choosing **Show Text** from the menu when the object is selected. For example, you can choose a component and make all associated text visible.

You can hide or show associated text for the following objects on a block diagram:

Object	Associated Text Element
external port	name type (and bounds) initial value (VHDL) or delay (Verilog)
signal or bus	name (and slice) type (and bounds) initial value (VHDL) or delay (Verilog)
bundle	name signal names (and slices)
component	library name component name instance name port name (and slice) port type (and bounds) port initial value (VHDL) or delay (Verilog) VHDL generic or Verilog parameter mapping port mapping assignments
block	library name block name instance name VHDL generic or Verilog parameter declarations VHDL generic or Verilog parameter mapping
embedded block	name number HDL text statements
generate frame	frame declarations
declarations	port declarations pre-s signal declarations post-s

See also [“Changing the Display of Port Properties”](#) on page 206 and [“Changing the Display of Signal Properties”](#) on page 208.

You can hide or show associated text for the following objects on a symbol:

Object	Associated Text Element
declarations	port declarations s
symbol	library name cell name VHDL generic or Verilog parameter declarations

You can also change the visibility of the text associated with ports by using the Port Display Control dialog box as described in [“Changing the Display of Port Properties”](#) on page 206.

You can hide or show associated text for the following objects on a state diagram:

Object	Associated Text Element
state	actions
transition	condition, actions and transition priority

Note



You cannot hide the name of a state.

You can also change the visibility of the following text objects on a state diagram:

Global Actions	Process Declarations (VHDL)
Concurrent Statements	State Register Statements
Architecture Declarations (VHDL)	Package List (VHDL)
Module Declarations (Verilog)	Compiler Directives (Verilog)
Signals Status	

You cannot hide the titles for these text elements from the popup menu but you can hide the entire text block by clearing the corresponding **Visible** check box in the State Machine Properties or the Object Visibility dialog boxes.

You can hide or show associated text for the following objects on a flow chart:

Object	Associated Text Element
action box	name actions
decision box	name condition statement
wait box	name wait statement
loop	name control statement
case	name expression

You can also change the visibility of the following text objects on a flow chart:

Architecture (VHDL)	Process Declarations (VHDL)
Module Declarations (Verilog)	Local Declarations (Verilog)
Concurrent Statements	Package List (VHDL)
Sensitivity List	Compiler Directives (Verilog)

You cannot hide the titles for these text elements from the popup menu but you can hide the entire text block by clearing the corresponding **Visible** check box in the Flow Chart Properties or the Object Visibility dialog boxes.




Adding Comment Graphics

You can add comment graphics on a block diagram, flow chart, state diagram or symbol by using the Comment Graphics toolbar or choosing **Comment Graphics** from the **Add** menu.

Comment Graphics Toolbar

The following commands are available from the Comment Graphics toolbar in the block diagram, state diagram, flow chart and symbol editor:

Table 2-3. Comment Graphics Toolbar

Icon	Description
	Add a rectangle, ellipse, circle or polygon
	Add a line, polyline or arc
	Add a bitmap






The  and  buttons perform a default command or you can use the  button to display a pulldown palette and choose the required operation. This choice becomes the default operation (indicated by the icon on the button) until another operation is chosen. The palettes are shown below:

Table 2-4. Comment Graphics Palettes

Shapes	Operation	Lines	Operation
	Add a rectangle		Add a line
	Add an ellipse		Add a polyline
	Add a circle		Add an arc
	Add a polygon		

The toolbar can be displayed or hidden by setting the **Comment Graphics** option in the **Toolbars** cascade of the **View** menu.

Refer to “[Toolbars](#)” on page 20 for more information about toolbars.

Corresponding commands are also available from the **Comment Graphics** cascade of the **Add** menu:

Table 2-5. Comment Graphics Menu Commands

Command	Description
Line	Add a line
Polyline	Add a multi-segment line
Polygon	Add a polygon
Arc	Add an arc
Rectangle	Add a rectangle
Ellipse	Add an ellipse
Circle	Add a circle
Bitmap	Add a bitmap

When you use any of these commands, the cursor changes to a cross-hair which allows you to add the graphics object by clicking and dragging at the required location on the diagram.

For example, you can add a polyline by clicking the **Left** mouse button at a number of route points and clicking twice to terminate the polyline or you can add a rectangle by dragging across the diagonal of the required shape.

Comment graphics can be cut, copied, pasted, aligned, moved, resized, rotated, flipped, hidden or deleted in a similar way to other diagram objects.

You can reshape a comment graphics object by choosing **Edit Vertices** from the popup menu to display its vertices which can then be dragged to change the shape or you can add an additional vertex by clicking anywhere between the existing vertices. You can remove a vertex by moving it onto an existing vertex.

Note

Adding vertices to a line converts it into a polyline. You can also change a rectangle into a polygon (or a polygon into a rectangle). However, you cannot edit the vertices of a arc, circle or an ellipse.


Comment graphics can be grouped with other comment graphics or comment text by choosing **Group** (or **UnGroup**) from the **Group** cascade of the **Edit** or popup menu.

You can hide an individual comment graphics (or comment text) object by choosing **Hide** from the popup menu or from the **Comment Graphics** cascade in the **Diagram** menu and show all hidden comment text or graphics objects by choosing **Show All**.




You can hide a group of comment text and graphics object by choosing **Hide Group** from the popup menu.


You can change the appearance of comment graphics by using the Appearance toolbar or choosing **Appearance** from the popup or **Edit** menu to set visual attributes for the selected graphics or you can change the default appearance by setting default visual attributes for the object in the diagram master preferences.

Adding a Line or Polyline

You can add a comment graphics line on a block diagram, flow chart, state diagram or symbol by using the  button or by choosing **Line** from the **Comment Graphics** cascade in the **Add** menu.

The cursor changes to a cross-hair and you can start the line by pressing down the **Left** mouse button and dragging to the required destination. The line is completed when you release the mouse button.


You can choose an alternative button operation by using the  button to display a palette. After choosing an operation from the palette, the new default is indicated by the icon on the button. For example, the button changes to  when you choose  from the palette.




You can add a polyline by using the  button or by choosing **Polyline** from the **Comment Graphics** cascade in the **Add** menu. The cursor changes to a cross-hair and you can start the line by pressing down or clicking the **Left** mouse button and dragging to a required vertex. A vertex is added each time you click the mouse button and the line is completed when you double-click the mouse button at the last vertex.

Note







You can change a line to a polyline (and add or remove vertices to a polyline) by choosing **Edit Vertices** from the popup menu. If you end a polyline at its starting point the enclosed area automatically becomes a filled polygon.

You can add (or remove) arrowheads on a line or polyline by using the  button in the Appearance toolbar and choosing an arrowhead style from the pulldown palette to set arrowheads at the beginning, end, neither or both ends of the selected line. After choosing an operation from the palette, this becomes the default operation for the button.

You can also change the color, style and width for the selected line by using the ,  and  buttons in the Appearance toolbar to display a palette of alternative choices. After choosing an operation from the palette, this becomes the default operation for the button.

Adding an Arc

You can add a comment graphics arc on a block diagram, flow chart, state diagram or symbol by using the  button and choosing  from the pulldown palette or by choosing **Arc** from the **Comment Graphics** cascade in the **Add** menu.

After choosing an operation from the palette, the new default is indicated by the icon on the button. For example, the button changes to  when you choose  from the palette.


The cursor changes to a cross-hair and you can start the arc by pressing down the **Left** mouse button and dragging to the end of the required arc. A ghosted line is drawn between these two points and becomes an arc when you release the mouse button and move the cursor to either side of the arc. The arc is completed when you click the mouse button to specify a vertex on the required arc.

Note







All arcs are circular although you can change the radius by editing the start end or intermediate vertices.

Adding a Rectangle or Polygon



You can add a comment graphics rectangle on a block diagram, flow chart, state diagram or symbol by using the  button or by choosing **Rectangle** from the **Comment Graphics** cascade in the **Add** menu.

The cursor changes to a cross-hair and you can draw the rectangle by pressing down the **Left** mouse button and dragging across the diagonal of the required rectangle. A ghosted rectangle is drawn between these two points and becomes a filled rectangle when you release the mouse button.

You can choose an alternative button operation by using the  button to display a palette. After choosing an operation from the palette, the new default is indicated by the icon on the button. For example, the button changes to  when you choose  from the palette.



You can add a polygon by using the  button or by choosing **Polygon** from the **Comment Graphics** cascade in the **Add** menu.

The cursor changes to a cross-hair and you can start the polygon outline by pressing down or clicking the **Left** mouse button and dragging to a required vertex. A ghosted connection to the starting point indicates the outline of the polygon which can be completed by double-clicking at the last of any number of vertices.





You can change the fill color and pattern for the selected rectangle or polygon by using the  and  buttons from the Appearance toolbar.


An existing rectangle can be converted into a polygon by choosing **Edit Vertices** from the popup menu to add or move vertices.

Adding an Ellipse or Circle

You can add a comment graphics ellipse on a block diagram, flow chart, state diagram or symbol by using the  button and choosing  from the pulldown palette or by choosing **Ellipse** from the **Comment Graphics** cascade in the **Add** menu.

The cursor changes to a cross-hair and you can draw the ellipse by pressing down the **Left** mouse button and dragging across the diagonal of the required ellipse. A ghosted ellipse is drawn between these two points and becomes a filled ellipse when you release the mouse button.

After choosing an operation from the palette, the new default is indicated by the icon on the button. For example, the button changes to  when you choose  or to  when you choose .

You can add a circle by using the  button or by choosing **Circle** from the **Comment Graphics** cascade in the **Add** menu.

The cursor changes to a cross-hair and you can draw the circle by pressing down the **Left** mouse button and dragging across the radius of the required circle. A ghosted circle is drawn between these two points and becomes a filled circle when you release the mouse button.

Adding a Bitmap

You can add a comment graphics bitmap on a block diagram, flow chart, state diagram or symbol by using the  button or by choosing **Bitmap** from the **Comment Graphics** cascade in the **Add** menu.

A file browser is displayed for you to locate a bitmap file. This can be any standard windows bitmap (*.bmp*), portable network graphics (*.png*) or portable bitmap (*.pbm*) image.

Note



You are advised to use bitmaps which have been saved using 256 or less colors. Although any valid bitmap can be added, images containing more than 256 colors may cause color flashing problems unless you are using a high performance graphics card.

The cursor changes to a cross-hair and you can add the image on your diagram by pressing down the **Left** mouse button and dragging across the diagonal of the required image size. The image can be resized at any time by dragging its resize handles. If you use **Shift + Left** mouse button, the aspect ratio is preserved as you drag the image.

You can cut, copy, paste, align, move, group, layer, hide or delete a bitmap in a similar way to other comment graphics objects but you cannot flip or rotate a bitmap.

Bitmaps are included by reference and you can specify the pathname of the bitmap by browsing or by entering a hard or soft pathname. If you open a diagram which references a bitmap that cannot be found, a cross is displayed. You can change the pathname for an existing bitmap by selecting the bitmap and choosing **Load Bitmap** from the popup menu.

Adding a Title Block

You can add a title block by choosing **Title Block** from the **Add** menu in a block diagram, flow chart, state diagram or symbol. The cursor changes to a cross-hair which allows you to place the template title block defined in your preferences by clicking at a location anywhere on the diagram.

The template title block is read from the file specified in the **Diagrams** tab of the Main Settings dialog box. Refer to [“Setting Preferences for Diagram Views”](#) on page 48.

The current template title block is automatically included at the lower right side on a new diagram (including concurrent and hierarchical diagrams) when you create a new graphic editor view unless this option is unset in the **General** tab of the Main Settings dialog box.

The title block text can be included in the generated HDL by choosing **Include in HDL** from the popup menu.

You can create your own title block template by grouping one or more comment texts (which may optionally include internal or user-defined variables) and choosing **Save Title Block** from the **File** menu to save the title block at the location specified in your preferences.

For example, the default title block comprises ten grouped comment texts and substitutes the internal variables `%(project_name)`, `%(library)`, `%(unit)`, `%(view)`, `%(user)`, `%(dd)`, `%(month)` and `%(year)` by the project name, library, design unit, design unit view, username and modified date.

Refer to “Using Internal Variables” in the [HDL Designer Series User Manual](#) for more information about internal variables.

The `%(project_name)`, `%(user)`, `%(dd)`, `%(month)` and `%(year)` variables are interpreted when the text is applied to the diagram. (*examples, joans, 31, Mar* and *2003* in this example) but the `%(library)`, `%(unit)` and `%(view)` variables are shown as `<TBD>` until you save the diagram.

<company name>		Project: examples
Title:	<enter diagram title here>	<enter comments here>
Path:	<TBD>/<TBD>/<TBD>	
Edited:	by joans on 31 Mar 2003	

Although you cannot apply different formatting to individual words or characters in a comment text, you can apply different formatting to each separate comment text in the group (such as the `<company name>` text in the above example).

Displaying Object Information

When you move the cursor over any object in a diagram editor, summary information about the object is displayed in a popup *object tip* window. The tip remains displayed for approximately five seconds or until the cursor is moved to another object.

This feature can be disabled or enabled by setting the **Object Tips** option in the **View** menu.

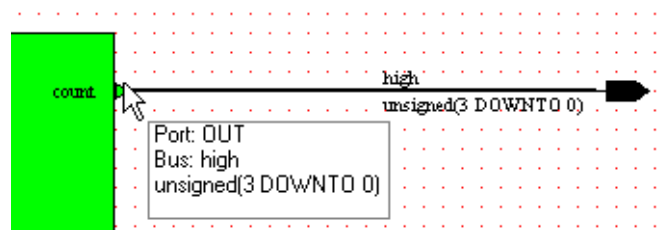
You can set preferences for the maximum number of characters and the maximum number of lines displayed for multi-line text in the popup information box.

These preferences can be set in the **Diagrams** tab of the Main Settings dialog box as described in “[Setting Preferences for Diagram Views](#)” on page 48.

If these limits are exceeded (for example, a state with multiple actions), an ellipsis is added to the end of each long line or to indicate that a multi-line field has more than three lines.

The *object tips* can be useful to identify objects on a diagram after you have used HDL2Graphics to convert a large design or when you have deliberately hidden text elements on the diagram.

The following example shows a tip for a component output port *count* connected to a VHDL bus named *high* with type *unsigned* and bounds *3 DOWNTO 0*.



Note




The slice, bounds and initial value (or delay) for a signal or bus are shown if they exist.

Panels

A panel is a defined and named area on a block diagram, flow chart, state diagram or symbol which facilitates viewing or printing the enclosed area.


Panels can be moved, copied or re-sized (by dragging their resize handles) and may partially or completely overlap other panels. You can also change the appearance of a panel by setting visual attributes for the panel box or name text.

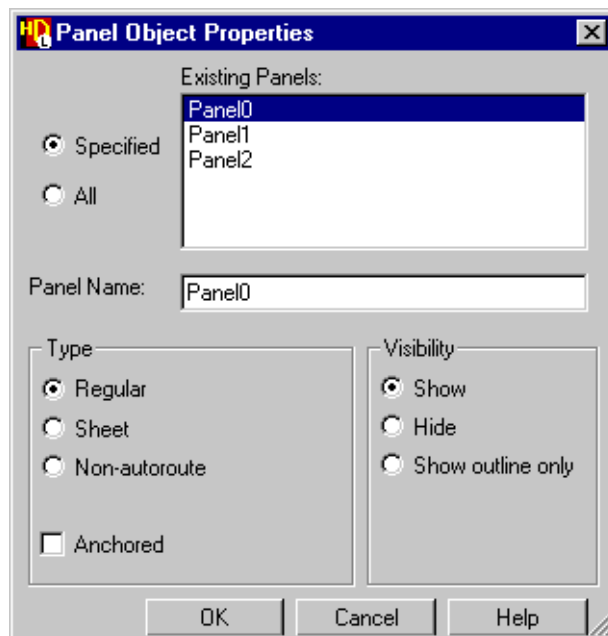
Adding a Panel

You can add a panel to a block diagram, flow chart, state diagram or symbol by using the  button, or by choosing **Panel** from the **Add** menu.

The cursor changes to a cross-hair and is attached to a ghosted panel which you can place anywhere on the diagram by clicking with the **Left** mouse button or you can hold down the **Left** mouse button and drag the cursor to resize the panel.

Editing Panel Object Properties

You can change the name of a panel by clicking on the panel name to select the text and clicking again to edit the text. Alternatively, you can double-click on the panel, use the  button, **Alt** + **Enter** shortcut or choose **Object Properties** from the **Edit** menu or popup menu, to display the Panel Object Properties dialog box.



If you do not change the name of a panel, each new panel is given a unique name by adding an integer to the default name (for example: *Panel0*, *Panel1*, *Panel2*...).

The dialog box can also be displayed when a panel is not selected by choosing **Object Properties** from the **Panels** cascade of the **Diagram** menu. (This can be useful when panels exist but are not visible in the current diagram view.)

The dialog box displays a list of all the panels that exist on the diagram. You can select one or more **Specified** panels or choose to edit the properties for **All** panels on the diagram. When a single panel is selected, you can use the dialog box to rename the panel by typing in a new panel name.

New panels are created as **Regular** panels but you can change any panel into a **Sheet** or **Non-autoroute** panel.

Note



A Sheet panel defines a diagram area and is used to support multi-sheet designs imported using guided HDL2Graphics placement. A Non-autoroute panel defines a diagram area to be excluded from autorouting operations.

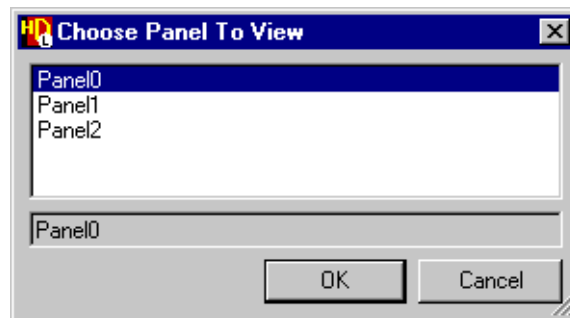
Any panel can also be **Anchored**. An anchored panel cannot be moved or re-sized and defines a fixed area of the diagram. An anchored panel may typically be used to define a page boundary or border. An unanchored panel can be freely moved around a diagram or re-sized to enclose other diagram objects.

You can also choose to **Show** or **Hide** the selected panels or to **Show outline only** (hiding the panel name).

Displaying a Panel

You can hide a panel by choosing **Hide** from the popup menu when a panel is selected or hide the panel name by choosing **Hide Text** from the popup menu or **Text** tab of the **Diagram** menu when the name is selected.

You can make a hidden panel visible by choosing **Show Panel** in the **Panels** cascade of the **Diagram** menu. If more than one panel exists on the diagram, the Choose Panel to Show dialog box is displayed for you to select the panel to make visible.



Viewing a Panel

You can change a diagram editor view to a named panel by choosing **View Panel** from the **View** menu or from the popup menu when a panel is selected.

If a panel is not selected and more than one panel exists on the diagram, the Choose Panel to View dialog box is displayed for you to select a panel. When you execute the dialog box, the diagram view is zoomed in or out to the extent of the specified panel.

The diagram view is also moved to the middle of the selected panel if you use the **Zoom In** or **Zoom Out** commands when a panel is selected.

Protecting Panels

You can protect all the existing panels on a diagram editor view by choosing **Protect All Panels** from the **Panels** cascade of the **Diagram** menu or from the popup menu when a panel is selected.


A protected panel cannot be selected, moved, copied re-sized or deleted (unless you use the explicit **Delete Panel** command).

Note

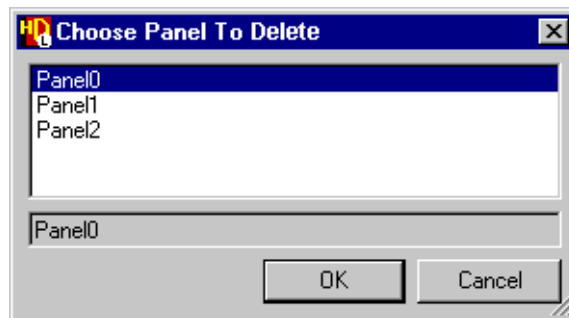


If you create another panel it is not automatically protected until you toggle protection to unprotect and then protect all panels on the diagram. You cannot explicitly protect or unprotect a single panel.


Deleting a Panel

You can delete an unprotected panel using the **Del** or  key but you cannot delete a protected panel in this way. However you can delete a protected or unprotected panel by choosing **Delete Panel** from the **Panels** cascade of the **Diagram** menu.

If no panels are selected when you choose this command and more than one panel exists on the diagram, the Choose Panel to Delete dialog box is displayed for you to choose which panel to delete.



Printing a Panel

You can print any named panel by using the  button, choosing **Print** from the **File** or popup menu or by using the **Ctrl + P** shortcut and selecting the required panel name in the Print dialog box.

You can also print the area within an unprotected panel by selecting the panel and choosing **Print** from the popup menu or from the **Panels** cascade of the **Diagram** menu. This can be

useful when you have added a temporary panel to define a print area. However, protected panels (which are not selectable) cannot be printed using this command.

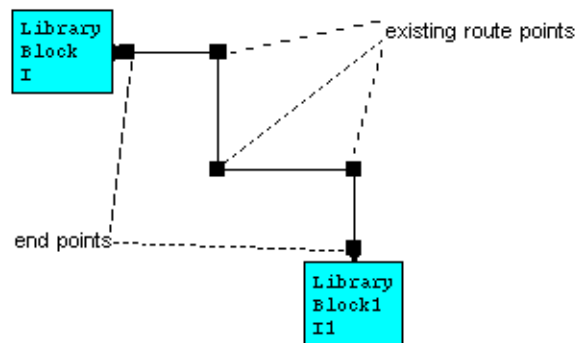
The panel is printed using the current scaling specified in the Page Setup dialog box. However, page breaks are not automatically calculated when you print a panel and unsatisfactory breaks may be imposed if your panel is larger than the current page size.

You can set the panel visibility in the Page Setup dialog box. For example, you can choose **Show Specified Panel** to print the specified panel as a border around the print area.

Editing Route Points

Signals, buses and bundles (in a block diagram), transition arcs (in a state diagram) or flows (in a flow chart) may have any number of route points.

When a net is selected, a filled square handle is displayed showing the position of each route point. These handles can be moved by dragging with the **Left** mouse button or deleted by moving the handle over another route point.



You can also remove route points by clicking on a route with the **Right** mouse button. A popup menu allows you to **Remove Route** or **Remove All Routes**.

If the cursor is further from a route point than the current grid spacing interval, the **Add Route** option allows you to add a new route point.

Route points are added when you single-click the **Left** mouse button while adding a signal, bus, bundle, transition arc or flow.

You can delete the last route point added while routing by using the **Del** key. The route is completed when you click over a destination object.



Tip: You can terminate a signal, bus or bundle without a destination object on a block diagram by double-clicking the Left mouse button.

When a route is connected using a curved spline (for example, the default type for a transition arc in a state diagram), you can also add a route point by dragging one of the unfilled diamond shaped handles which as shown midway between each route point when the spline is selected.

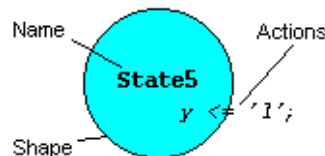
Transition arcs in a state diagram are drawn (by default) as splines but signals, buses and bundles in a block diagram or flows in a flow chart are drawn as orthogonal polylines.

The default drawing style and other attributes can be changed by setting diagram master preferences.

Existing orthogonal lines on a diagram can be made into diagonals by creating, dragging or deleting route points.

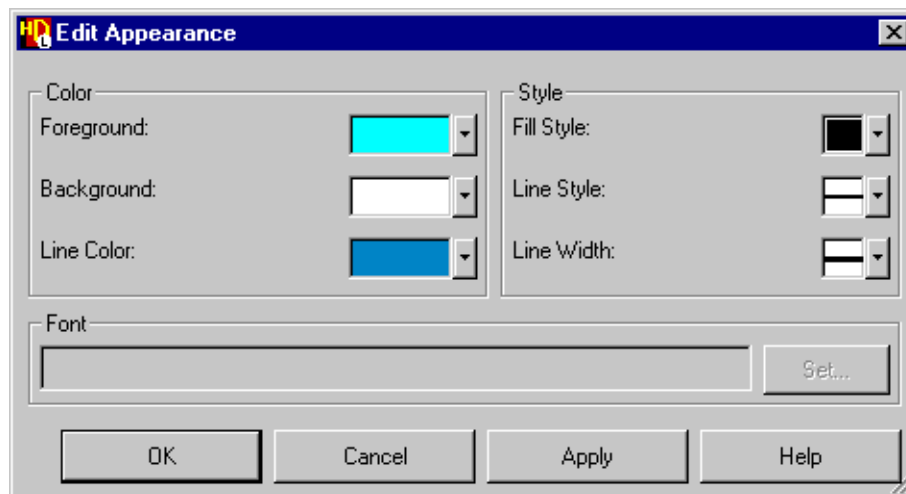
Setting Visual Attributes

Visual attributes can be set for individual elements of each diagram object. For example, separate visual attributes can be set for the name, shape, outline and actions text that comprise the symbol for a state.




You can set visual attributes for one or more objects in a diagram editor by selecting the required objects and using the Appearance toolbar or by choosing **Appearance** from the **Edit** or popup menu to display the Edit Appearance dialog box.


The dialog box indicates the current setting when a single object (or multiple objects with the same attribute) is selected.




When a single object is selected in a graphic editor, the dialog box shows the current **Foreground** and **Background** colors for the selected object. When a cell is selected in the interface editor, the dialog box shows the **Font** and **Background** cell colors.

You can click on the  button adjacent to each button to display a palette which allows you to choose from a set of 25 standard colors or use the **Other** option on the palette to set color attributes using any of the colors available on your system.

You can choose the **Line Color** used for object outlines, nets on a block diagram, transition arcs on a state machine or flows on a flow chart.

You can choose from a palette of nine alternative **Fill Style** patterns which combine the current foreground and background colors or choose the  fill pattern to make an object transparent.

You can also choose the **Line Style** and **Line Width** used for object outlines, nets, transition arcs and flows on a graphical diagram. The line widths include a no line  option which makes the line transparent.

In the interface editor, you can also set the alignment of text in the table cells.









You can set alternative text display fonts by using the **Set** button to choose from the fonts available on your window system. The available font sizes usually include the range 6 to 14 point in normal, bold, italic and bold italic typeface for a variety of font families.

The modified attributes are applied to all objects in the current selection set when you confirm the dialog box.


Appearance Toolbar


You can also set visual attributes using the following commands which are available from the Appearance toolbar in the block diagram, state diagram, flow chart and symbol editor:

Table 2-6. Appearance Toolbar

Button	Description
	Applies a logical symbol shape to the selected object
	Changes the foreground color of the selected object
	Changes the color of the selected text string
	Changes the color of the selected line or object outline
	Changes the line style used for the selected object
	Changes the line width used for the selected object
	Changes the fill pattern style used for the selected object
	Changes the arrowhead style used for the selected line or polyline

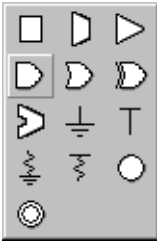



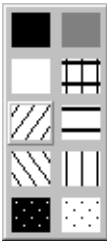

Note

The  button is not available in the state diagram and flow chart editors.

Each of these buttons performs a default command or you can use the  button to display a pulldown palette and choose the required operation. This choice becomes the default operation until another operation is chosen.

For example, the foreground, text font and line color buttons display a color choice palette and the last selected color becomes the default for the button. The palettes are shown below:

Table 2-7. Appearance Palettes

Logical Shapes	Foreground, Text or Line Color	Line Style	Line Width	Fill Pattern	Arrow Style
					

The toolbar can be displayed or hidden by setting the **Appearance** option in the **Toolbars** cascade of the **View** menu.

Refer to “[Toolbars](#)” on page 20 for more information about toolbars.

Setting Color Attributes

You can use the **Other** option on the color palette to display a Color selection dialog box which allows you to set any other color which is supported on your workstation.

On PC systems, the **Other** option displays a Color dialog box which provides a palette of 48 basic colors and an option to **Define Custom Colors** by setting red green blue (RGB), or hue, saturation and luminosity values. Up to sixteen custom colors can be defined and are saved as preferences.

On UNIX systems, the Color chooser dialog box provides slider controls to set colors using red, green, blue (**RGB**) or hue, saturation and value (**HSV**) color models. Alternatively, you can use the **Color Names** button to choose from a selection list of the named colors defined for your window system or display a **Color Disk** and gray scale which can be used to pick any available color.

Toggling the Grid Visibility and Snapping

You can toggle the grid visibility for the active block diagram, flow chart, state diagram or symbol by setting (or unsetting) the **Grid** option in the **View** menu.

You can also set the default grid visibility and snapping for each type of diagram by setting a preference.





Note





Grid snapping is always enabled for a block diagram or symbol and cannot be unset.

You can toggle grid snapping for the active state diagram by setting (or unsetting) the **Snap to Grid** option in the **View** menu.

Changing the Diagram View


You can change the view of the active diagram editor window by using the  button, the **Shift** +  shortcut or by choosing **Zoom In** from the **View** menu or popup menu to increase the magnification of the diagram and the  button, **Shift** +  shortcut or **Zoom Out** from the menu to decrease the magnification of the diagram.







You can restore the view before the last zoom command by choosing **Zoom Last** from the **View** menu. You can view the entire diagram by using the  button, the  shortcut key or choosing **View All** from the **View** menu.




Note




You can display popup information about the object under the cursor even if you have zoomed out and text elements are not visible.

You can choose an area to view using the  button, the **Shift** + **Home** shortcut or **View Area** menu option. When you choose one of these options, the cursor changes to a cross-hair and you can drag select the required area to view.

You can scroll the active diagram by using the vertical scroll bars  and  keys or in smaller increments by using the  and  keys. You can pan the diagram using the horizontal scroll bars or the  and  keys.

You can also scroll and pan the active diagram by using the  button and holding down the **Left** mouse button. In this mode, the cursor changes to  and the diagram moves with the cursor which changes to  until the mouse button is released.

The button is shown pressed while in panning mode until you return to normal select mode by using the  button or cancel the command using the **Esc** key.

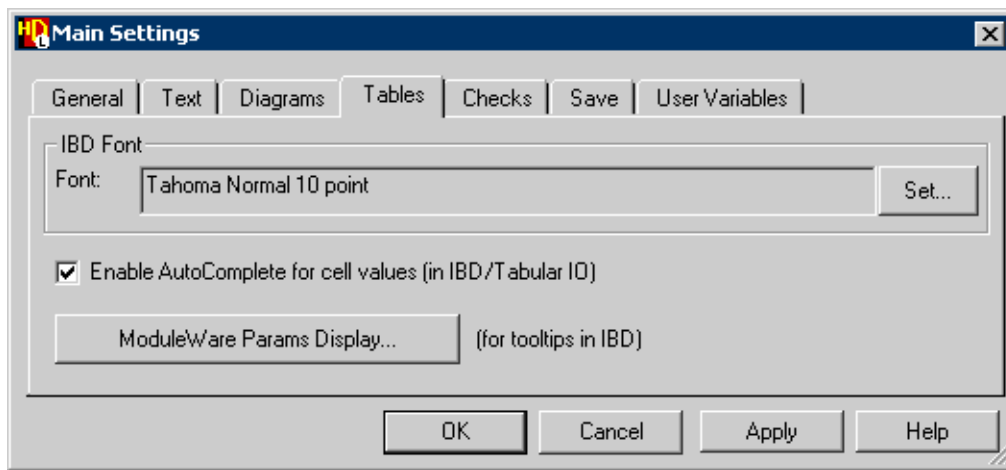
If you have a three-button mouse, you can also scroll and pan by holding down **Ctrl + Middle** mouse button.

Table Editor Windows

This section describes features that are available in the *truth table*, *tabular IO* and *IBD view* table editors. Later chapters describe features that are implemented in a particular editor.

Setting Preferences for Table Views

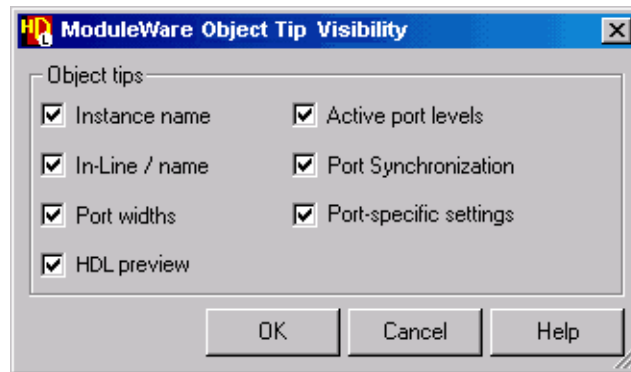
You can set the preferences used for new *tabular IO* and *IBD views* in the **Tables** tab of the Main Settings dialog box which is displayed when you choose **Main** from the **Options** menu in any window.



You can change the default font for IBD by using the **Set...** button to choose from the fonts available on your system.

You can enable automatic completion of cell values for both IBD and Tabular IO views. When this option is set and you enter the initial characters for a existing or previously entered string, the entry is completed automatically.

You can use the **ModuleWare Params Display** button to display the ModuleWare Parameters Object Tips Visibility dialog box and set the default *object tips* displayed in an *IBD view*:



Note





These preferences are used for tabular IO and IBD views only.
The default font for the text in truth tables can be set in the Truth Table Master Preferences dialog box.

Selecting Table Cells

You can select a single table editor cell by clicking with the **Left** mouse button or select multiple cells by clicking on a cell and dragging to extend the selection.

You can also extend the selection set in the tabular IO or IBD view editor by holding down the **Shift** key to add a range of cells to the existing selection or the **Ctrl** key to add individual cells to the selection.

Alternatively, you can use the **Shift** + , **Shift** + , **Shift** +  or **Shift** +  shortcuts to extend the selected range using the arrow keys.

Note

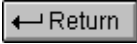





These shortcuts are not available in the truth table editor.





You can select an entire row or column by clicking on the first cell in the row or column. You can select multiple rows or columns by selecting multiple cells in the first row or column.

Editing a Table Cell

You can add text to a table editor cell by simply selecting the cell and typing the required expression. Any existing text is overwritten or you can double-click in a cell to explicitly add new or edit existing text characters.



The text is entered when you click in another cell or you can use the  key (which also selects the cell below), the **Shift** +  keys (which select the cell above), the  key (which selects the cell to the right) or the **Shift** +  keys (which select the cell to the left).

You can move to the first column in the current row by using the **Home** key, the first column in the first row by using the **Ctrl** + **Home** keys or the last column in the last row by using **Ctrl** + **End**.



You can also use the **End** key with the **Home**, , ,  or  keys to move between cells. Unlike other key modifiers, **End** is followed by the key and the keys do not need to be pressed together.





You can **Cut**, **Copy**, **Delete** or **Paste** text to or from one or more cells using the **Cut**, **Copy**, **Del** or **Paste** keys or the corresponding commands available from the **Edit** or popup menus. You can also clear the text from one or more selected cells by choosing **Clear** from the **Table** or popup menu.

You can use **Ctrl** with the **Left** mouse button to add cells to a selection set, or **Shift** to add all cells between the selected cell and the current cursor cell. You can select all cells in a column or row by clicking the **Left** mouse button in the ruler or select all cells in the table by clicking the origin cell in the top left corner of the table.

You can use the **Alt** +  shortcut in a tabular IO or IBD view cell to display a list of choices for the cell which match the entered characters. For example, if the Bounds column already contains: (0 to 9), (1 DOWNT0 0), (11 TO 16), (32 TO 64) and (100 DOWNT0 33) and you enter a 1 character followed by **Alt** + , the list of choices includes: (1 DOWNT0 0), (11 TO 16) and (100 DOWNT0 33).

Changing the Table View

You can scroll a table editor window using the vertical scroll bars or the  and  keys and pan the window using the horizontal scroll bars.

In a truth table editor window, you can also scroll the window using the **Shift** +  or **Shift** +  keys and pan the window using the **Shift** +  or **Shift** +  keys.

Note




These shortcuts are not supported in the tabular IO or IBD view editors.

Resizing a Column or Row

You can change the width of a column or row in a truth table, tabular IO or IBD view by dragging the sash in the horizontal or vertical ruler.

The diagram shows a table with 5 rows and 4 columns labeled A, B, C, and D. Row 1 has a black header cell, a light blue cell in column A, a light blue cell in column B, a yellow cell in column C, and a yellow cell in column D. Rows 2 through 5 have light blue cells in column A, light blue cells in column B, yellow cells in column C, and yellow cells in column D. A black cell is present in row 5, column B. Horizontal sashes are indicated by lines pointing to the top and bottom borders of the table. Vertical sashes are indicated by lines pointing to the left and right borders of the table.

	A	B	C	D
1				
2				
3				
4				
5				

You can automatically fit a column (or columns) to the width of the text contained in the selected cell (or cells) by using the  button or by choosing **Autofit** from the **Table** menu. If no cells are selected, then all columns in the table are re-sized.

Note



This command is not available in the truth table editor.

Exporting a Table

You can export the current table editor view as a tab-separated value (TSV) or comma-separated value (CSV) format file by choosing **TSV** or **CSV** from the **Export** cascade of the **Table** menu.

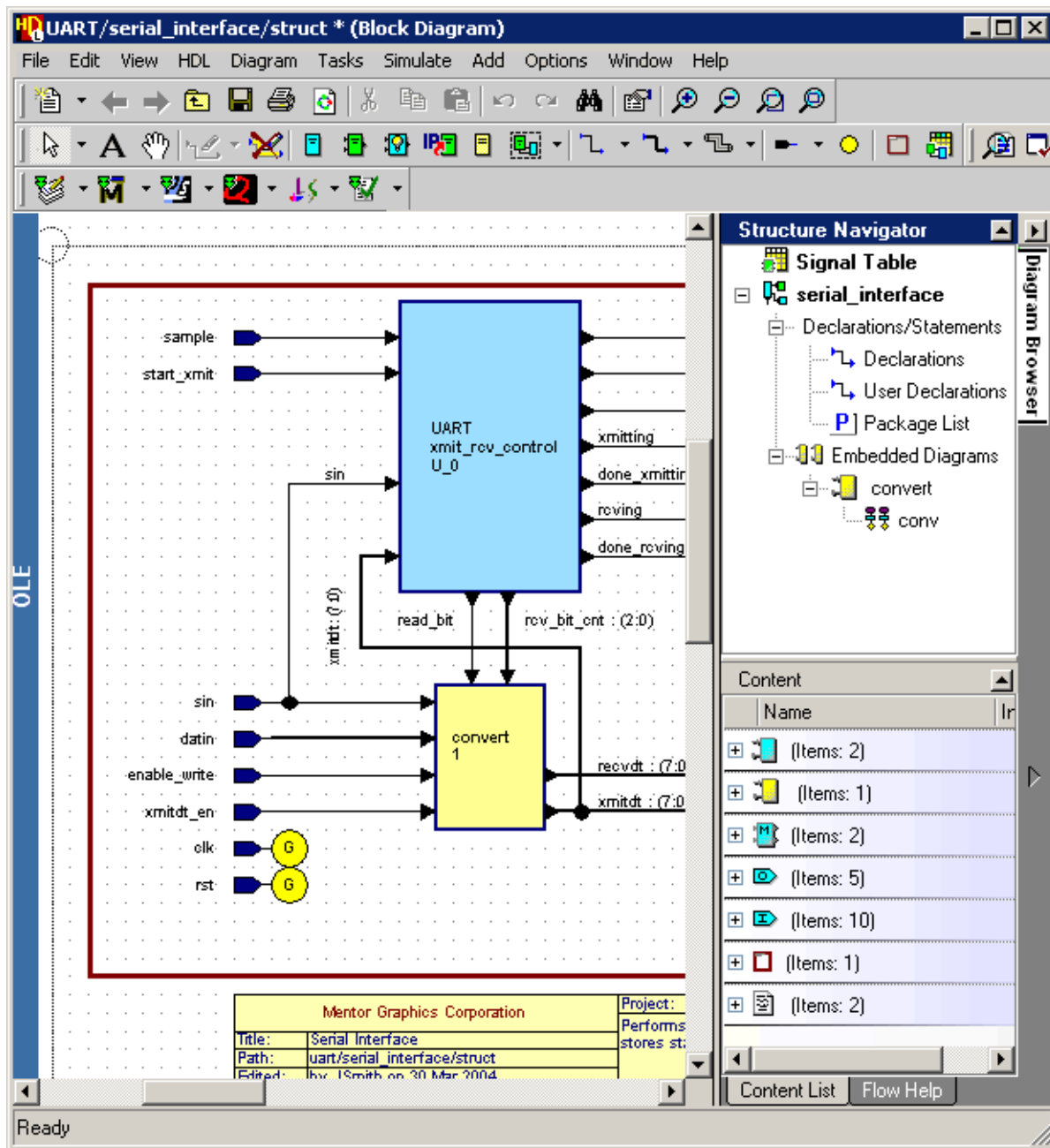
The currently displayed tab is exported in an IBD view. The **All** tab should be selected if you want to export all interface information for the current design unit view.

If you do not specify a file extension, *.csv* is used when you export a comma-separated value file or *.txt* when you export a tab-separated value file.

TSV and CSV files can be useful to preserve the column layout when you transfer tabular information into external documentation tools such as Adobe FrameMaker and Microsoft Word or Excel.



The Diagram Browser

The *diagram browser* is a sub-window that appears on the right of a graphical editor view. The sub-window is divided into two vertical panes. The upper pane displays the *Structure Navigator* and the lower pane displays either the *Content List* tab or the *Flow Help* tab.



You can hide the diagram browser by clicking on the right arrow icon in the window border or show the diagram browser by clicking on the left arrow icon. You can also hide or show the diagram browser by setting **Diagram Browser** in the **View** menu.

The diagram browser and main graphical editor panes can be resized by dragging the resize sashes between the panes. Horizontal or vertical scroll bars are automatically displayed when the contents of a browser are larger than its current size. The diagram browser panes are normally tiled to share the available window area but you can use the following buttons to change the layout:






-  Expand pane vertically to full height of window
-  Return pane to normal horizontal tiling

Browsing Diagram Structure

The *Structure Navigator* pane shows the structure of the active diagram view including any embedded blocks on a block diagram or concurrent and hierarchical views on a flow chart, ASM chart or state diagram view and the Signals table.

The following icons are used to identify views in the *Structure Navigator* pane for a block diagram:

Table 2-8. Structure Navigator Notation — Block Diagram









Icon	Description
	Block diagram design unit
	IBD view design unit
	Embedded diagrams or tables on a block diagram or IBD view
	Individual embedded block on a block diagram
	Signals table

When the design unit is selected, the contents of the diagram are shown in the *Content List* pane as described in “[Browsing Diagram Content](#)” on page 94. You can also access the diagram properties and package list or set object visibility on the diagram by using the **Right** mouse button to display a popup menu. When an individual embedded view, declaration or package list is selected, the diagram is panned and centered on the selected object.

When the signals table is selected, it becomes the active view in the diagram window.

The following icons are used to identify views in the *Structure Navigator* pane for a state machine, ASM chart, flow chart or for these views when they are embedded on a block diagram:

Table 2-9. Structure Navigator Notation — State Machine, ASM, Flow Chart





Icon	Description
	State diagram view
	Concurrent state diagram view
	ASM chart view
	Concurrent ASM chart view
	Flow chart view
	Concurrent flow chart view
	Truth table view (when embedded on a block diagram or IBD view)
	Signals table

When one of the diagram views is selected, it becomes the active diagram and the contents of the diagram are shown in the *Content* pane as described in “[Browsing Diagram Content](#)” on page 94. You can also access the diagram properties and package list, rename a concurrent view or set object visibility on the diagram by using the **Right** mouse button to display a popup menu. You can also create new Concurrent views by selecting **New Concurrent Machine** from the popup menu of a concurrent machine.

When the signals table is selected, it becomes the active view in the diagram window.

The following icons are used for to identify views in the *Structure* pane for a symbol:










Table 2-10. Structure Navigator Notation — Symbol


Icon	Description
	Tabular interface view
	Graphical symbol view
	Generic declaration table (VHDL only)
	Parameter declaration table (Verilog only)

You can toggle between the tabular interface, graphical symbol and VHDL generics or Verilog parameters view.

The *Structure Navigator* pane also shows text objects such as the package list or compiler directives. The following icons are used for text objects in the *Structure Navigator* pane:

Table 2-11. Structure Navigator Notation — Text Objects

Icon	Description
	Package list
	Compiler directives
	Signal, port and s, architecture or module declarations
	Process declarations
	Local declarations
	Concurrent statements
	Sensitivity list or state register statements
	Recovery state settings
	Global actions

An  icon is overlaid on any text object shown in the *Structure Navigator* pane of the diagram browser which is currently hidden on the graphical diagram. You can control which text objects are shown or hidden in the diagram browser by choosing **Show Object** from the popup menu for the design unit icon in the diagram browser.

When you select one of the text objects, the diagram is panned and centered on the selected object. You can use the **Right** mouse button to choose from a context-sensitive popup menu of available commands. For example, you can hide or show declarations text on the diagram or open an object properties dialog box to edit the declarations. If there is only one command available, it is automatically performed when you select the object. For example, the *s* dialog box is opened when you select a *s* object.

Declarations and statement blocks which apply to the whole diagram are shown in the hierarchy below the design unit icon. Declarations and statement blocks which apply to a concurrent diagram are shown in the hierarchy below the concurrent view icon.

Browsing Diagram Content













The *Content list* pane is displayed when the design unit or a concurrent state diagram, ASM chart or flow chart is selected in the *Structure Navigator* pane. The *Content List* pane lists all the design objects contained in the view.

When you select an icon in the Content List pane, the corresponding object is selected on the diagram. The diagram may be panned and centered if the selected object is not within the current diagram view. You can choose to autozoom on selected object by choosing **Auto Zoom on Select** from the popup menu to display.

You can edit the properties of a selected object in the Content List pane by choosing **Object Properties** from the popup menu to display the Object Properties dialog box.












The following icons are used in the *Content List* pane for a block diagram or IBD view

Table 2-12. Content List Notation — Block Diagram, IBD

Icon	Description
	Block
	Component
	External IP component
	ModuleWare component
	Embedded block
	Input port
	Output port
	Bidirectional port
	Buffer port
	Generate frame
	Panel
	Requirement Reference







The following icons are used in the *Content* pane for a flow chart:

Table 2-13. Content List Notation — Flow Chart

Icon	Description
	Action box
	Hierarchical action box
	Decision box
	Wait box
	Start point
	End point
	Start loop
	End loop
	Case box
	Panel
	Requirement Reference













The following icons are used in the *Content* pane for a state diagram:

Table 2-14. Content List Notation — State Diagram

Icon	Description
	Interrupt point
	State
	Hierarchical state
	Wait state
	Panel
	Requirement Reference

The following icons are used in the *Content* pane for an ASM chart:

Table 2-15. Content List Notation — ASM

Icon	Description
	Interrupt point
	Reset point
	Enable point
	Action box
	Hierarchical action box
	State box
	Hierarchical state box
	Link
	Case box
	If decode box
	Panel
	Requirement Reference

Changing the Columns in the Content Pane

You can change the columns displayed in the *Content* pane by checking options in the **Columns** cascade of the popup menu which is displayed if you click the **Right** mouse button over any column heading.

New columns are displayed in the order they are added. However, you can change the column display order by dragging the column header with the **Left** mouse button.

You can change the width of the columns by dragging the sashes between each column or automatically resize a column to fit its contents by double-clicking on the sash.

Sorting the Content Pane

The order of the objects *Content* pane can be changed by choosing **Sort Ascending** or **Sort Descending** from the popup menu over any column header to re-order the objects in ascending or descending order for the selected column.

The new sort order is indicated by a ▼ or ▲ indicator in the column header.



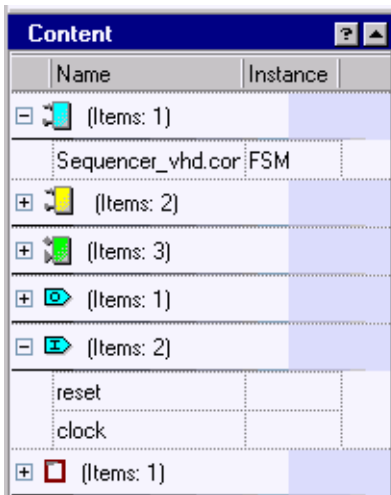
Tip: You can quickly toggle the existing order by clicking the **Left** mouse button in any column header.

Using Groups in the Content Pane

You can group design objects in the *Content List* pane by choosing **Group by this column** from the popup menu for a column header.

If you choose more than one column to group by, the groups are nested in the order that you selected them.

You can use the + icons to expand any group and reveal the objects it contains. The following example shows the *fibgen* block diagram in the *Sequencer_vhd* example design grouped by object type with the block and input port groups expanded:







You can choose **Ungroup** from the popup menu to remove all column groups.

Using Flow Help

The Flow Help pane provides you with a set of demos and instructions that guide you in using the active graphical editor and in navigating through the design.

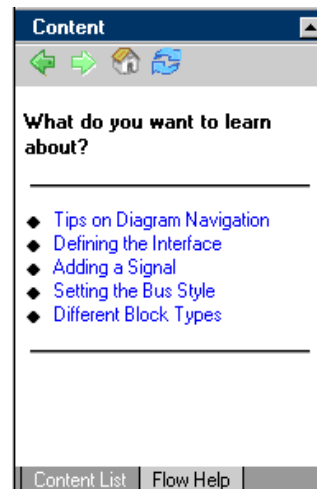
The following icons are used in the *Flow Help* pane:

Table 2-16. Flow Help Notation

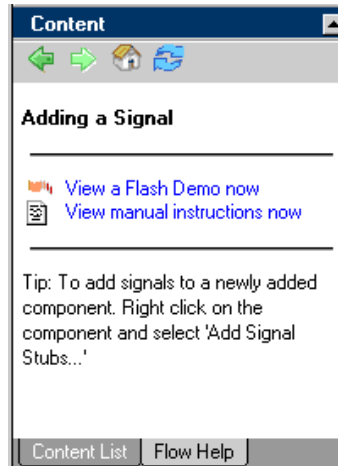
Icon	Description
	Move back to previously visited pages
	Move forward to recently visited pages
	Return to the home page
	Refresh the active page

To use flow help:

1. Make sure you have a web browser configured to display Macromedia Flash applications installed on your system.
2. Click on any of the topic links in the Flow Help pane.



3. A new page is displayed showing a list of demonstrations. Each topic is demonstrated by either self-running demos or a set of manual instructions.



4. Click on any of the demonstrations to view in your default web browser

Signals Table

The signal declarations associated with each graphical view can be displayed in a tabular view by selecting the **Signals** page in the [diagram browser](#). Refer to “[The Diagram Browser](#)” for information about browsing diagram structure and content.

The table is synchronized to show only the signals for the active graphical view including any ports which have been added to the symbol (if it exists). Any redundant ports are removed and the type, bounds and other properties updated from the symbol.

If you have edited the signals table, the graphical view is synchronized with the signal status. The table has a separate row for each signal defined in the interface plus additional rows for any locally defined signals.

The following example shows the signals table for a block diagram view of the *cpu_interface* design unit in the *UART* example design.

	A	B	C	D	E	F	G
	Group	Name	Mode	Type	Bounds	Initial	Comment
1		clk_div_en	IN	std_logic			
2		clr_int_en	IN	std_logic			
3		cs	IN	std_logic			
4		datout	OUT	std_logic_vector	(7 DOWNTO 0)		data to cpu
5		div_data	IN	std_logic_vector	(7 DOWNTO 0)		
6		nrw	IN	std_logic			
7		ser_if_data	IN	std_logic_vector	(7 DOWNTO 0)		
8		xmitdt_en	IN	std_logic			
9		clear_flags	OUT	std_logic			
10		clk	IN	std_logic			
11		enable_write	OUT	std_logic			
12		rst	IN	std_logic			
13		start_xmit	OUT	std_logic			

Signals Table Notation

Vertical and horizontal scroll bars are available if the signals table does not fit in the current window. However, the header row and the Group, Name and Mode columns are non-scrolling and are always shown.

Refer to “[Grouping Signal Rows](#)” on page 105 for information about using the Group column.

Note



Note that you can select an entire row by clicking the row number or an entire column by clicking the column letter. You can also select the entire table by clicking on cell A1. You can resize any cell by dragging the sashes between the columns.

Signal Declaration Columns

The signal declarations for interface ports are displayed in the Name, Mode, Type, Bounds, Delay or Initial and Comment columns.

New signal declarations can be added using an empty row at the bottom of the table.

Name	Port or locally declared signal name.
Mode	Signal mode: input, output, bi-directional, buffer (VHDL only) or local.
Type	VHDL type definition or Verilog net type.
Bounds	Range of the specified type (may use short or long format for VHDL).
Delay	Delay value for a Verilog signal.
Initial	Initial value of a VHDL signal.
Comment	Comment appended to a signal declaration.

Refer to the “[Component Interface Views](#)” chapter for more information about port signal declarations.

Signals Table Toolbars

The following commands are available from the Graphical Editors Tools toolbar in the signals table:

Table 2-17. Signals Table Toolbar















Icon	Description
	Add an input port
	Add an output port
	Add a bidirectional (inout) port
	Add a buffer port (VHDL only)
	Add a local signal
	Group the selected rows or add a group (in hierarchical mode)
	Ungroup
	Expand all groups
	Collapse all groups
	Toggle Filter
	Fit the cell width to the contents of the selected cell
	Sort in ascending order

Table 2-17. Signals Table Toolbar (cont.)





Icon	Description
	Sort in descending order
	Toggle between grouped and ungrouped mode

The Standard, HDL Tools and Tasks toolbars are also available in the signals table window. Refer to “[Toolbars](#)” on page 20 in the Graphical Editors User Manual for information about the Standard graphical editors toolbar. Refer to “Toolbars” in the [HDL Designer Series User Manual](#) for information about the HDL Tools and Tasks toolbars.

Adding Port or Local Signal Declarations

You can add ports to a component interface using the **Add** menu or the following buttons in the tabular IO view:


Table 2-18. Tabular IO View Commands for Adding Port or Local Signal Declarations

Button	Function Key	Description
	F8	Add an input port
	F9	Add an output port
	F11	Add a bidirectional (inout) port
	F12	Add a buffer port (VHDL only)

The port is added in the next available row with default name, type and bounds.

Alternatively, you can add ports by entering a declaration directly into the next row of Name, Mode, Type and Bounds cells. The mode defaults to the last mode used or you can choose from a list of available modes: input, output, bidirectional (inout) or buffer (VHDL only).

If you do not change the name of a port, each new port name is made unique by adding an integer to the default name. (For example: *In0*, *In1*, *In2*...).

If you add a port whose name suggests it might be a clock, reset, or enable, the port type, mode and category columns are populated with the signal default values. You can add a declaration for a local signal by using the  button or choosing **Local Signal** from the **Add** menu. A new declaration is added at the bottom of the table with the default name *Local* or *LocalN* (where *N* is automatically incremented if it exists).


You can also add a local signal declaration by entering the new signal name directly in the Name column of the empty row at the bottom of the signals table.

The port or local signal type defaults to the last type used or you can choose from a dropdown list of available types in the Type column. The bounds defaults to the last range used or you can choose from a dropdown list of recently entered ranges in the Bounds column.

A VHDL bounds can be entered in long or short format. The display format can be set by setting or unsetting the **Short Form** option in the **Table** menu.

If you enter a port or signal name followed by a valid bounds constraint, for example, *myport(7 DOWNT0 0)*, the constraint is automatically moved to the Bounds column.



Tip: You can automatically complete a row with default properties by using the  key after entering a port name to move to the name cell in the next row.

You can optionally enter a value in the Initial (VHDL) or Delay (Verilog) and Comment columns. The delay or initial value can be chosen from a dropdown list of recently entered values.

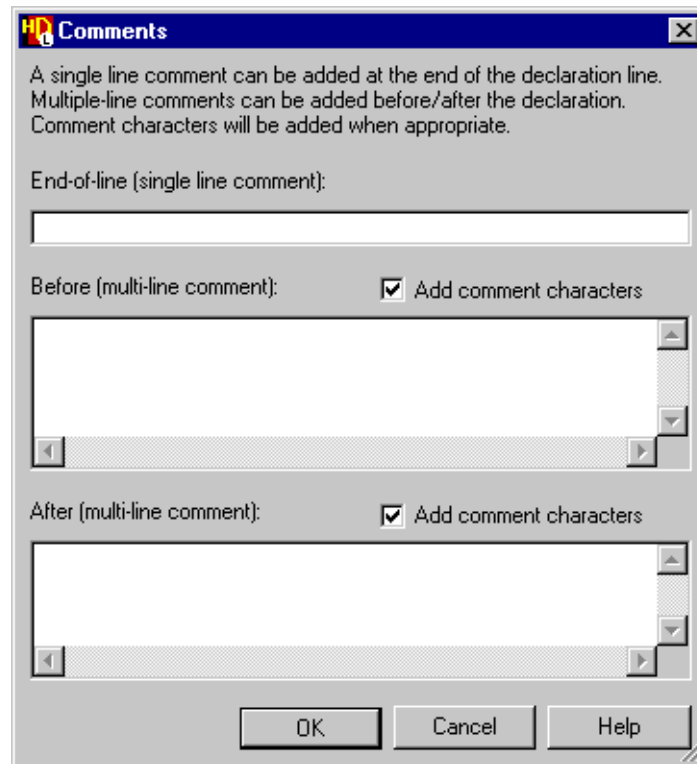
If you enter characters that match characters in an existing entry of the same column, the remaining characters are entered automatically.

If you have changed port declarations to the signal table, the interface is automatically synchronized when you save the state machine view and any new ports added to the parent view.

Adding Comments to a Port or Local Signal Declaration

You can add comments to a port or local signal declaration by choosing **Comments** from the popup menu when the declaration row is selected.

A free-format entry Comments dialog box is displayed which allows you to add a single line comment at the end of the declaration or you can enter a multi-line comment to be included before or after the declaration.




Comment characters for the current hardware description language (VHDL or Verilog) are automatically inserted if the **Add comment characters** check box is set.

When this option is unset, the comments must be valid HDL statements and are automatically syntax checked if checking is enabled.

If a declaration is deleted, the corresponding comments are also deleted.

Although multi-line comments can be added using the dialog box, these comments are not displayed in the table. However, end-of-line comments can be edited directly in the Comment column for the local signal declaration row.

Resizing Columns

You can automatically fit a column (or columns) to the width of the text contained in the selected cell (or cells) by using the  button or by choosing **Autofit** from the **Table** or popup menu. If no cells are selected, then all columns in the table are re-sized.


Hiding Columns

You can hide and show columns in the signals table by choosing **Hide Column** from the popup menu or the **Columns** cascade of the **Table** or menu.

If one or more columns are hidden, you can display a dialog box listing the hidden columns by choosing **Show Columns** from the popup or **Table** menu.

Refer to “[Hiding Columns](#)” in the Graphical Editors User Manual for more information.


Filtering Columns

You can filter the content of columns in the signals table by using the  button or setting the **Filter** option in the **Table** menu. When this option is set, an additional row is added in the non-scrolling area and a dropdown filter menu is available in each column.

You can apply filters to more than one column or set options to match case, match whole words or use regular expressions by choosing **Filter Settings** from the **Table** or popup menu to display the Filter Controls dialog box.

Refer to “[Filtering Columns](#)” in the Graphical Editors User Manual for more information.

Grouping Signal Rows

You can group rows in the signals table by selecting a row or rows and using the  button or choosing **Group** from the **Add** menu.


The selected rows are added to a new group with the default name *SmGroupN* or *AsmGroupN* (where *N* is automatically incremented if it already exists).

You can also add a group or create a new group by entering a name in the Group column for the ports you want to group.


Note





You can choose from a dropdown list of existing groups. If you type a partial string that matches the name of an existing group the name is automatically completed.


You can remove a group name by selecting a row (or rows) and using the  button or by deleting the name from the Group cell.

If you rename or remove an existing group cell and the group is no longer referenced, you are prompted to delete the old group name.

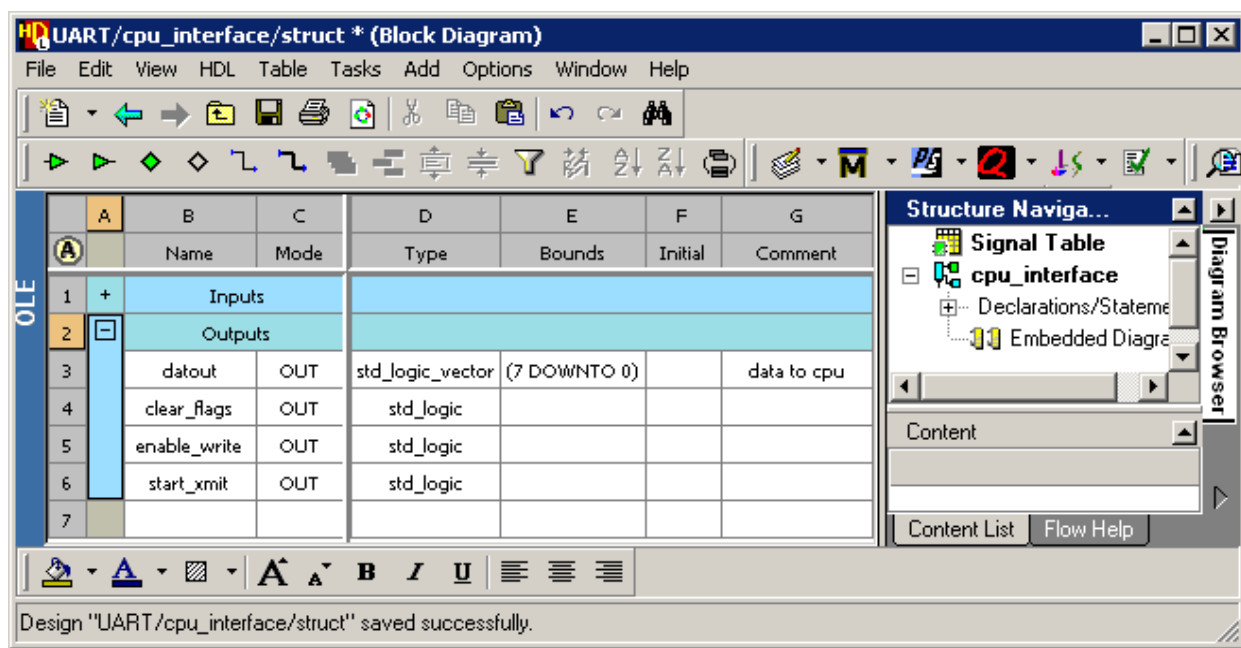
When the signals table includes one or more groups, you can use the  button or set **Show Grouped** in the **Table** menu to toggle between grouped and ungrouped mode.

All rows are displayed normally in flat mode but rows in the same group are shown as a single (but expandable) group in hierarchy mode.

You can expand all the group rows by using the  button or choosing **Expand All Groups** from the **Ports** cascade in the **Table** menu. Alternatively, you can expand an individual group by clicking on the  icon.

You can collapse all the group rows by using the  button or choosing **Collapse All Groups** from the **Ports** cascade in the **Table** menu.

When grouped mode is set, you can enter a comment in the group row as shown for the *Inputs* and *Outputs* groups in the example below. The *Inputs* and *Outputs* groups are both shown expanded but the *Locals* group is shown collapsed:



Note





Groups added in the signals table may be discarded and replaced by the groups defined in the tabular IO view when the table is synchronized with an updated symbol view.

If you delete a group which contains local signals only, they are deleted. If you delete a group which includes ports, only the local signals in the group are deleted.

You can also allow or disallow signals sorting within a group by choosing **Allow Sort/Disallow Sort** from the group popup menu or the **Group** cascade of the **Table** menu. Unsortable groups do not get sorted when you sort signals and have the **[SORT PROTECTED]** label displayed beside their name.

Sorting Signal Rows

You can sort the rows in a selected column of the signals table in ascending alphanumeric order of the cell data by using the  button or choosing **Sort Ascending** from the popup menu or the **Columns** cascade of the **Table** menu.

Alternatively, you can sort the data in descending order by using the  button or by choosing **Sort Descending** from the popup menu or the **Columns** cascade of the **Table** menu.

Note



Signals in Sort Protected Groups will not be sorted, but rather the group itself is placed among other signals or groups according to its name.

Chapter 3

Block Diagram and IBD Views

This chapter describes how the structure of a design can be represented using a graphical *block diagram* or tabular *IBD view*.


Editing Block Diagram and IBD Views	110
Adding Blocks and Components	110
Assigning Automatic Instance Names	111
Instantiating a Block	111
Instantiating a Component	112
Adding an Embedded Block	124
Updating an Instance	127
Reconciling Interfaces	128
Checking the Design	131
Checking Through Hierarchy	132
Editing Object Properties	133
Editing Component Properties	133
Editing Block Properties	139
Editing Embedded Block Properties	143
Editing ModuleWare Properties	145
Editing Bundle Properties	148
Editing Signal Properties	149
Editing External IPs Properties	146
Editing Port IOs Properties	156
Editing Frame Properties	157
Editing Comment Text Properties	158
Editing Requirement Reference Properties	158
Editing Comment Graphics Properties	160
Propagating Net Changes	166
Inserting and Removing Nets	169
Ordering Port and Signal Declarations	171
Adding or Removing Design Hierarchy	172
Generics and Parameters	174
Generics and Parameters Tables	176
Defining Generics and Parameters	184
Editing Generics and Parameters for Instances	186
Generics and Parameters Synchronization	192
Opening Block and Component Views	194
View Initialization	195

Setting the Default View.	196
Mixed Language Designs.	196
VHDL Instantiation of Verilog Components	197
Verilog Instantiation of VHDL Components	198

Editing Block Diagram and IBD Views

You can edit a graphical view describing the structural interfaces between design units using the block diagram or IBD view editors. These views are saved as the same design unit view (*struct.bd* if it is created as a block diagram or *struct.ibd* if it is created as an IBD view).

This chapter describes procedures that are common to block diagram and IBD views. Refer to Chapter 4, [Block Diagram Editor](#) or Chapter 5, [IBD View Editor](#) for procedures that are specific to each editor.

You can display an IBD view by using the  button or by choosing **Show IBD** from the **Diagram** menu in the block diagram editor.

When you display an IBD view from a block diagram, some objects (for example, bundles) will not be displayable and an ! is displayed in the Instance Ref row of the interface column in the IBD view.

Note



When you select a net or net declaration on a block diagram, the net name is selected on the IBD view.

When you select a net row on an IBD view, the net and the net declaration are selected on the block diagram. However, if you select a net name cell, only the net declaration is selected on the block diagram.

Adding Blocks and Components

You can add a block or component instance on a block diagram or IBD view by using the **Add** menu or by using one of the buttons in the block diagram or IBD view toolbars. Some objects can also be added using a shortcut or mnemonic key as shown in the following tab:

Table 3-1. Block Diagram/ IBD Commands for Adding Blocks and Components






Button	Shortcut	Mnemonic	Description
	Ctrl + F3	B	Add a block
	F3	C	Add a component
	none	none	Add a ModuleWare component
	none	none	Add an external HDL (IP) model

Table 3-1. Block Diagram/ IBD Commands for Adding Blocks and Components (cont.)

Button	Shortcut	Mnemonic	Description
	F4	E	Add an embedded block

Note



The mnemonic shortcut keys are not supported in an IBD view.

Assigning Automatic Instance Names

The instance names for blocks and components are normally derived from a base name specified in your block diagram preferences.

The base name can be a simple string (such as the default characters *U_*) which can be manually edited on a block diagram or IBD view.

Alternatively, you can use an automatic name derived from the design unit name which cannot be edited once you have created the design unit.

You can set automatic instance names for the active view by choosing **Automatic Instance Names** from the **Diagram** menu on a block diagram or the **Table** menu in an IBD view.


Instantiating a Block

When you add a *block* on a block diagram, a ghosted rectangle is attached to the cursor and can be placed on the diagram by clicking the **Left** mouse button at the required location. The block is added with a default size or you can hold down the mouse button and drag across the diagonal for a required size.

When you add a block on an IBD view, a new instance column is added to the table matrix.

The block has a library name, block name and instance name. You can change any of these names individually on a block diagram (or the instance name on an IBD view) by double-clicking on the name.

You can edit the name or library name (if the library row is displayed) of a block in an IBD view by choosing **Rename** from the popup menu.

You can also edit the library name, block name or instance name by using the **Blocks** page in the Object Properties dialog box which is displayed when you use the  button or choose **Object Properties** from the **Edit** menu.

If you do not change the library name for a block on an untitled view, it is automatically given the same library name as the parent view when the view is saved. Once a view has been saved, its library is used as the default library for new blocks added to the view.

Note



Each block must have a unique block name and instance name. The block and instance name cannot be the same when you are using VHDL but can be given the same name if you are using Verilog.

If you do not change the instance name, each new block is given a unique instance name by adding an integer to the default name (for example: *U_0*, *U_1*, *U_2*, *U_3*...).

The name text can be moved independently away from (or into) the block on a block diagram. If you want to contain the text inside the block outline, it may be necessary to resize the object.


Instantiating a Component

When you add a *component*, the *component browser* is displayed which allows you to choose a component (including ModuleWare components and components you have created) from an existing *library* and drag an instance of the component on to a *block diagram* or *IBD view*.

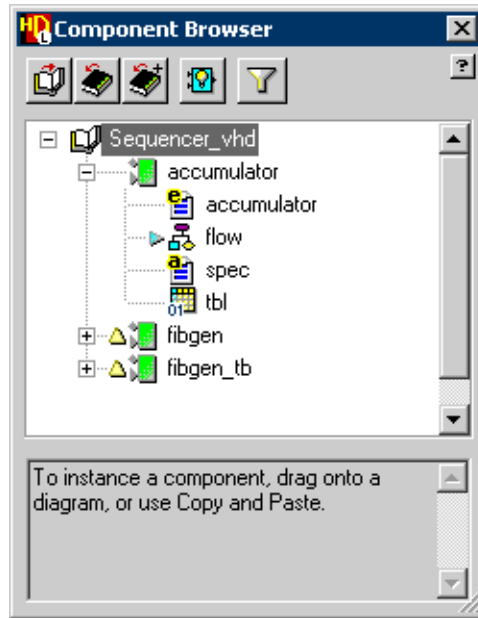
Note





The component browser can also be displayed by choosing **Component Browser** from the **Window** menu in a *graphical editor* window or from the **Tools** menu in the *design manager*. Refer to “Using the Component Browser” in the *HDL Designer Series User Manual* for more information.

You can instantiate a component by selecting the required *design unit* or view and holding down the **Left** mouse button to drag the  cursor over a block diagram or IBD view.

For example, the following picture shows the dialog box used to instantiate the truth table view of the *accumulator* component in the *Sequencer_vhd* library:



The  cursor changes to  when you move over a valid destination and is added when you release the mouse button.

If you drag a component design unit, the default view is instantiated. If you use the + icon to expand the design unit and drag a design unit view of a VHDL design unit, it is instantiated as a named view.

Note



You can instantiate a Verilog component in a VHDL design or a VHDL component in a Verilog design.

You can also add a component by copying or dragging a design unit or design unit view from a library displayed in a *design explorer* window.

When you add a component on a block diagram, a ghosted rectangle is attached to the cursor and can be placed on the diagram by clicking at the required location. The component shape and port positions on the block diagram are defined by its symbol.

When you add a component on an IBD view, a new instance column is added to the table matrix.

The component is added with its existing object name and a unique instance name. If you have selected a named VHDL view, the view name (for example: *struct.bd*) is shown after the instance name. You can change the instance name on a block diagram by double-clicking on the name.

You can also edit the instance name and other component properties by using the **Components** page in the Object Properties dialog box as described in “[Editing Component Properties](#)” on page 133.

If the component references any VHDL packages which are not already referenced in the package list, you are prompted whether to add them.

If you do not change the instance name, each new component is given a unique instance name by adding an integer to the default name (for example: *U_0*, *U_1*, *U_2*, *U_3*...).

The instance name can be automatically derived from the design unit name or it can be an editable string derived from a base name specified in your preferences as described in “[Assigning Automatic Instance Names](#)” on page 111.


The name text can be moved independently away from (or into) the component on a block diagram. If you want to contain the text inside the component outline, it may be necessary to resize the object.

You can move ports around the component symbol on a block diagram by dragging them with the mouse. (If you select more than one port, their relative separation is preserved.)

You can also space all the ports on each edge of a component evenly by choosing **Equidistant Ports** from the popup menu or **Ports** cascade of the **Diagram** menu when the component is selected.

Instantiating Verilog 2005 or System Verilog3.0 Text Components

A Verilog 2005 or System Verilog 3.0 text component can be instantiated in a Block diagram or IBD design if the component interface is logically compatible with the language of the diagram.

 **Tip:** A logically compatible interface is one whose port types can be mapped to Verilog 95 port types.

For compatibility reasons a level of type substitution will be provided. Following the type substitution, if the component interface is still logically incompatible i.e one or more port types failed to map to Verilog 95 port types, a message is issued and the operation is aborted.

Examples of port types that would fail to map are types void or enum.

The table below shows a list of Verilog2001/System Verilog 3.0 types that are supported as Verilog 95 compatible ones.

Table 3-2. Supported Verilog2001/System Verilog 3.0 Types

Type	Size
logic	1

Table 3-2. Supported Verilog2001/System Verilog 3.0 Types (cont.)

Type	Size
bit	1
byte	8
char	8
shortint	16
shortreal	32
integer	32
int	32
longint	64
time	64
real	64
realtime	64

For Verilog 95 diagrams, any type from the above table will be mapped to reg or wire of the corresponding size depending on the following

Mode of the Declaration to be Substituted

- Output ports are substituted with "reg"
- Input/Inout ports are substituted with "wire"

Type of Diagram

Verilog 95 structural diagrams override type of nets according to diagrams' master preferences while symbol types are not changed.

For VHDL diagrams, ports map to std_logic or std_logic_vector depending only on the size.

For BD/IBD, type substitution is applied to the newly created nets connected to a Verilog 2001 or System Verilog 3.0 instance. As for symbols created for Verilog 2001/System Verilog 3.0 text, type substitution is applied to the module ports themselves i.e. they would be replaced by Verilog 95 compatible types in the symbol.

Note

Verilog 2000/System Verilog 3.0 synchronization occurs only from text to graphics, any changes in the graphics are not reflected in the text.

Synchronization occurs between the instantiated component and the underlying text view according to the following rules:

- Graphical components with no symbols are updated only when you manually update them by choosing Component>Update>Interface from the component popup menu.
- Symbols of graphical components are updated on saving there underlying text components.
- A mixed design symbol is updated when changing the default view to Verilog 2005/System Verilog3.0




Tip: Symbol/ Interface updating can be allowed/disallowed through Options>Main>Save tab in the Design Explorer window or from the symbol properties dialog incase of mixed designs.

In all of the above cases the tool checks if the Verilog 2000 /System Verilog types in the text are logically compatible with the language of the diagram and updates the graphical interface doing any needed type substitution.

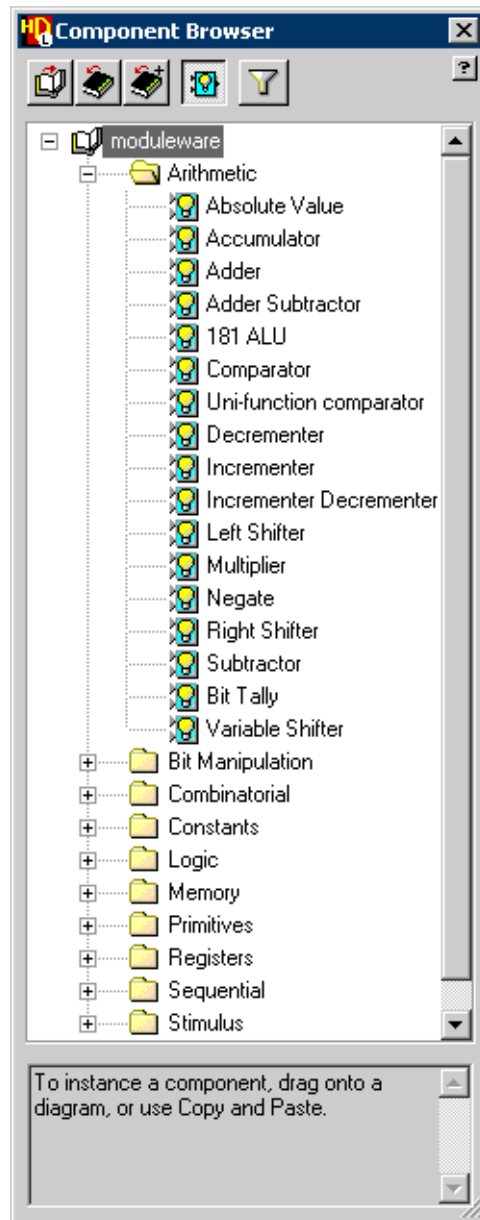
Instantiating a ModuleWare Component

HDL Author and HDL Designer include a parameterizable *ModuleWare* library which can be used to instantiate language-independent *component* parts representing a wide range of logic and arithmetic functions.

The *moduleware* library is automatically pre-selected in the *component browser* when you use the  button or choose **ModuleWare** from the **Add** menu.

When the *moduleware* library is displayed in the *component browser*, you can choose from a number of categories which each contain parts supporting a family of functions. Each category is displayed as an expandable folder.

For example, the *Arithmetic* folder contains parts which can be used to implement arithmetic functions.

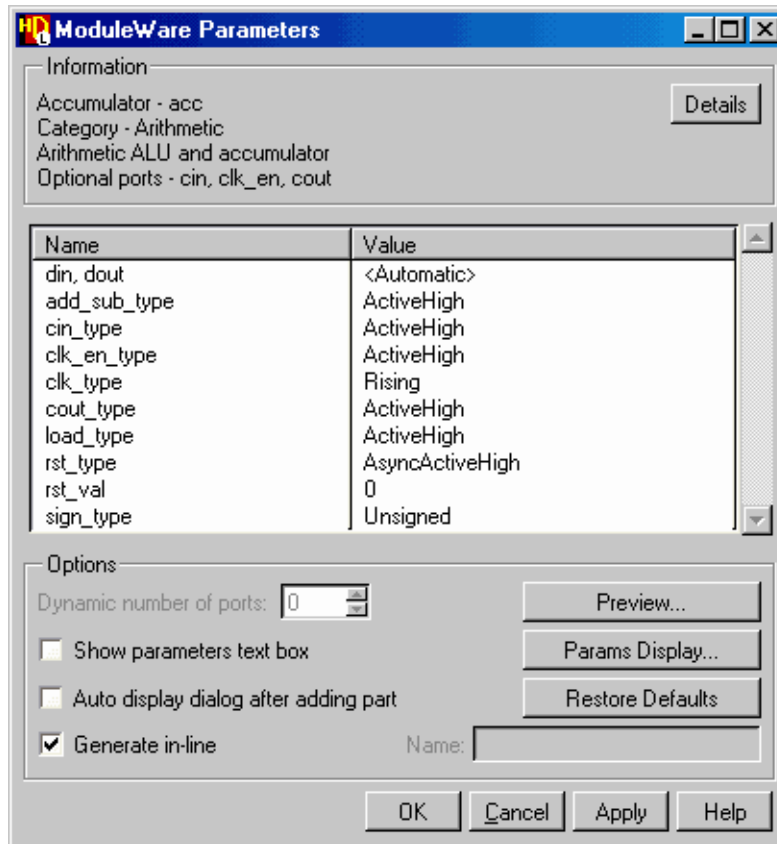


You can edit the instance name or other object properties for a ModuleWare part in the same way as any other component by using the **Components** tab in the Object Properties dialog box as described in [“Editing Component Properties”](#) on page 133.

Editing ModuleWare Parameters

You can edit ModuleWare parameters by using the ModuleWare page in the Object Properties dialog box.

For example, the following picture shows the ModuleWare Parameters dialog box for the *Accumulator* arithmetic function:



You can also display the ModuleWare Parameters dialog box by double-clicking on a ModuleWare component instance or by selecting **Parameters** from the **ModuleWare** cascade in the popup menu.

You can edit the ModuleWare parameters by entering the required values in the table or (for enumerated parameters) by choosing from a dropdown list.

You can preview the parameter settings and the generated HDL (or the error and warning messages that would be issued when HDL is generated) by using the **Preview** button to display the ModuleWare Preview dialog box.

You can also display the ModuleWare Preview dialog box by choosing **Preview** from the **ModuleWare** cascade of the popup menu when a ModuleWare instance is selected.

You can optionally display ModuleWare parameters in a comment text box when a part is instantiated on a block diagram and set the parameters which are displayed by using the **Params Display** button as described in [“Setting the Visibility of ModuleWare Parameters”](#) on page 120.

You can restore all parameters to their default values by using the **Restore Defaults** button.

You can also choose whether HDL for the ModuleWare instance is generated in-line or to a separate file. When set to in-line, the generated HDL is embedded in the HDL for the parent diagram. When the in-line option is unset, the generated HDL is written to a separate file with the name shown in the dialog box. (This name is a unique signature derived by adding a unique 32-bit hexadecimal number to the part name.)

Refer to the ModuleWare Parameters dialog box description for more information about setting ModuleWare parameters.

Note



Signals and buses can be connected to a ModuleWare instance in the same way as any other component. Note that the port size parameters default to *Automatic* and will be automatically set to the width of the connected net.

Note also, that you can connect bus slices directly to the bit manipulation models but not to any other type of ModuleWare or component instance.

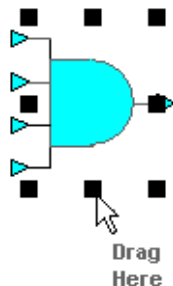
Refer to the [ModuleWare Reference Guide](#) for more information about using the ModuleWare library including full descriptions of each supported function.

Note that the relevant page in a HTML version of the reference guide can be accessed by using the **Details** button in the ModuleWare Parameters dialog box or by choosing **Details** from the popup menu in the [component browser](#).

Using Dynamic ModuleWare Components

A number of dynamic ModuleWare components (*and*, *nand*, *nor*, *or*, *sand*, *sor*, *sxor*, *xnor*, *xor*, *mux*, *omux*, *merge*, and *split*) are available. These components have two input or output ports when they are instantiated but can be extended to support any number of ports by resizing the instance to disclose the required number of ports or by setting the required number of ports in the ModuleWare Parameters dialog box.

To resize an instance, select the component by clicking with the left mouse button then drag on the top or bottom resize button as shown below until the required number of ports are available.



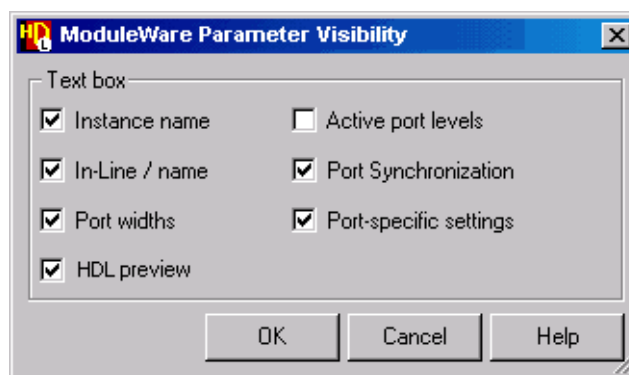
Additional port polarity parameters are available in the ModuleWare Parameters dialog box for each port.

You can also change the polarity by selecting a port and choosing **Active High** or **Active Low** from the **Port Type** cascade of the popup menu.

When a ModuleWare instance is selected, you can often change the function by choosing one of the **Change to** options from the **ModuleWare** cascade in the popup menu. For example, if an *and* gate is selected, you can change it to an *or* or *xor* gate with the same number of ports. The logical shape is automatically updated to represent the new function.

Setting the Visibility of ModuleWare Parameters

You can set the visibility of the *ModuleWare* parameters on a block diagram by using the **Params Display** button in the ModuleWare Parameters dialog box to display the Moduleware Parameters Visibility dialog box:



The dialog box can also be displayed by double-clicking on an existing ModuleWare Parameters text box.

The selected parameters are shown in a text box on the diagram which is usually displayed by default but can be hidden by choosing **Hide Text** from the popup menu for the text box. If not shown the text box can be displayed by choosing **Show Text** from the popup menu for the ModuleWare component.

You can set the default visibility of *ModuleWare* parameters on a block diagram and the properties which are shown in the *object tips* for a Moduleware instance using the *ModuleWare Params* page of the Block Diagram Master Preferences dialog box.

These preferences can also be accessed in an existing diagram to change the visibility for new and existing instances in the same diagram. Refer to [“Setting Block Diagram Preferences”](#) on page 231 for more information.

Instantiating an External HDL Model

Any existing external HDL model can be instantiated by reference as a component in a block diagram or IBD view. The HDL Designer Series installation includes HDL for Inventra, Seamless CVE and speedCHART models (in the *hdl_libs/src* subdirectory).

VHDL and Verilog source for synthesizable Inventra system level building blocks is provided in the *hdl_libs/src/inventra* installation subdirectory as two files:

inventra_soft_cores.vhd (VHDL entities)
inventra_soft_cores.v (Verilog modules)

These models are based on soft cores distributed by the Inventra Intellectual Properties (IP) business unit of Mentor Graphics Corporation.

More information including descriptions of all the currently available models can be obtained from the Inventra worldwide web site at:

<http://www.mentor.com/inventra/index.html>

VHDL 87 source for using the following Seamless CVE models is provided in the *hdl_libs/src/cve_qhdl_vhdl* installation subdirectory:

<i>arm7tdmi.vhd</i>	<i>dram.vhd</i>	<i>mc68040.vhd</i>	<i>register_mem.vhd</i>
<i>dpram.vhd</i>	<i>fifo.vhd</i>	<i>ppc603e.vhd</i>	<i>sram.vhd</i>

Refer to the Seamless CVE documentation for a full description of these models.

Note



Seamless expects these models to be compiled into the library:

cve_qhdl_vhdl.lib at *\$CVE_HOME/cve_qhdl_vhdl_87_lib*

where the *CVE_HOME* environment variable specifies the location of the Seamless software. You should add this library to your library mapping and enter *cve_qhdl_vhdl* as the compiled library in the Add Instance dialog box when instantiating a CVE model.

Please contact Mentor Graphics customer support for information about the latest available CVE models for other compilers (including Verilog language models) or visit the Seamless worldwide web site at:

<http://www.mentor.com/codesign/main-f/index.htm>

VHDL source for the following speed*CHART* component libraries can be found in the *hdl_libs/src/spdch/vhdl* installation subdirectory:

spdch_components.vhd
spdch_entities.vhd
spdch_configurations.vhd

The following Verilog file can be found in the *hdl_libs/src/spdch/verilog* installation subdirectory:

spdch_comp_lib.v


Note

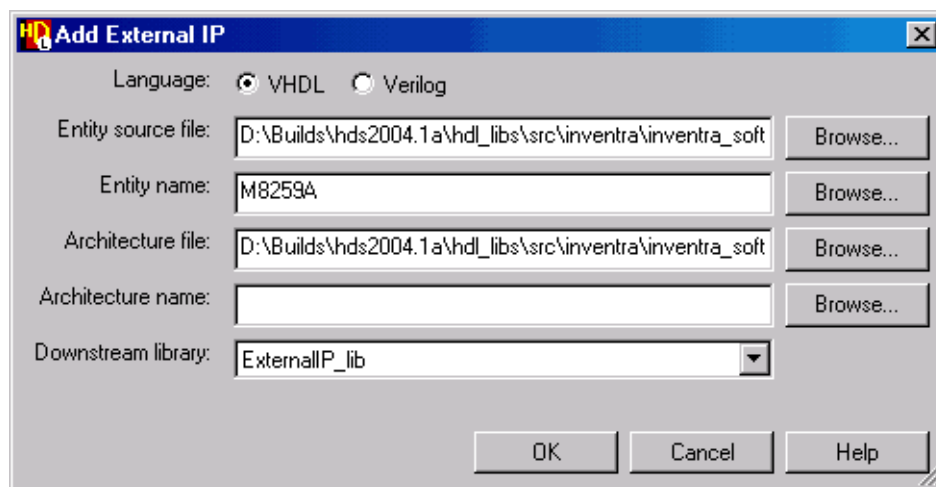


The models defined in these files are provided for users transitioning from speedCHART Project Designer to the HDL Designer Series and can be instantiated as external HDL models or converted using HDL import.

Any other models defined in a VHDL entity declaration, component definition in a VHDL package or Verilog module description on your file system can also be instantiated. For example, FPGA or 3Soft models.

Note that an external HDL model can be defined using a different language to the view it is instantiated in. VHDL models can be re-used in a Verilog block diagram or Verilog models in a VHDL diagram.

The Add External IP dialog box is displayed when you use the  button or choose **IP** from the **Add** menu:



You can enter (or browse for) the pathname of an existing HDL file which contains a VHDL entity declaration or Verilog module and browse a list of the entities (or modules) contained in the file.

Note



If the file extension is not recognized as one of those specified in the **File** tab of the VHDL or Verilog Options dialog box, it is assumed to be VHDL.

You can optionally enter a soft pathname which is defined as an environment variable or (on UNIX systems) in a location map. However, the full expanded pathname is shown when you use the **Browse** button.

For an external VHDL file, you can browse for the name of a separate architecture file and architecture name. (You can optionally omit the architecture name or enter a name which is not yet defined.) However, you must specify the downstream library which contains (or will

contain) the compiled object. This library can be set by choosing from a drop down list of currently mapped *Downstream Only* libraries.

The library name, entity name and an automatically generated unique instance name are shown on the component.

For an external Verilog file, you can optionally use a library to contain the compiled object. The Verilog module name, filename and an automatically generated unique instance name are shown on the component.

Caution

External VHDL or Verilog models must be compiled outside the HDL Designer Series tool.

The external HDL file is parsed to extract information about the interface including contained VHDL entities, VHDL architectures or Verilog modules, VHDL package references and VHDL generic or Verilog parameter declarations.

If the specified file contains a single HDL model (described by a VHDL entity, VHDL architecture or Verilog module) this model is automatically selected. An error message is issued if a parse error is encountered.

If a VHDL model requires any VHDL packages which are not already referenced, you are prompted whether to add them.

A symbol is automatically created and the external HDL model is added on a block diagram as a component instance with input ports automatically placed on the left, output ports on the right, bidirectional ports on the top and buffer ports on the bottom.

The default values of any VHDL generic or Verilog parameter declarations are shown on a block diagram and can be edited using the Object Properties dialog box.

Using a Soft Pathname for External HDL

If you want your design to be portable, it is advisable to specify the location of external HDL models using a soft pathname.

The soft pathname can be an environment variable or on UNIX systems a soft prefix defined in a location map.

The **Browse** buttons on the Add Instance and Update dialog boxes do not allow you to browse soft pathnames. However, you can enter a soft pathname directly or use the browser to locate the external HDL file and then substitute the appropriate prefix.

A leading \$ must be included when you are using an environment variable or location map prefix. For example, you could enter *\$External* in the file name entry box, then use the **Browse**

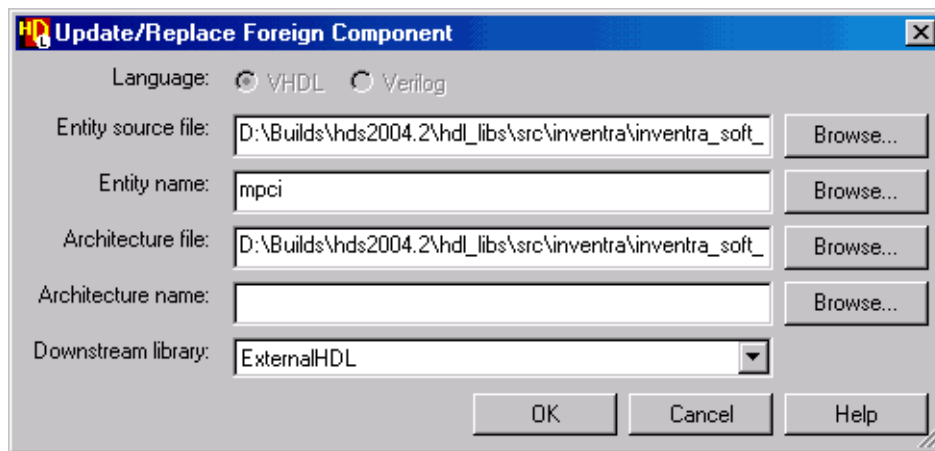
button to navigate below this location and select an external HDL file. Then substitute *\$External* in the hard pathname before confirming the dialog box.

Updating an External HDL Model

You can update or replace an external HDL model by choosing **From HDL** from the **Update** cascade in the popup menu when an external IP component is selected on a block diagram or in an IBD view.

The Update/Replace Foreign Component dialog box is displayed which allows you to update the pathnames for the external HDL source files or replace the name of the VHDL entity, VHDL architecture or Verilog module.

For example, the following dialog box is displayed when the external HDL is VHDL:



You can also change the downstream library for an external HDL model. However, the language can not be updated. If you want to replace an external HDL model by one of a different language, it must be deleted and re-instantiated in order to update the interface correctly.

Note



You do not need to update the pathnames if you used a soft pathname which is defined as an environment variable or (on UNIX systems) in a location map. The references are retained if the definition of the soft pathname changes.

Adding an Embedded Block


When you add an *embedded block* on a block diagram, a ghosted rectangle is attached to the cursor and can be placed on the diagram by clicking the **Left** mouse button at the required location.

The embedded block is added with a default size or you can hold down the mouse button and drag across the diagonal for a required size. When you add an embedded block on an IBD view, a new instance column is added to the table matrix.

The embedded block is added with a default name and instance number. The default name can be set as a preference but is made unique by adding a numeric suffix if the name already exists on the diagram.

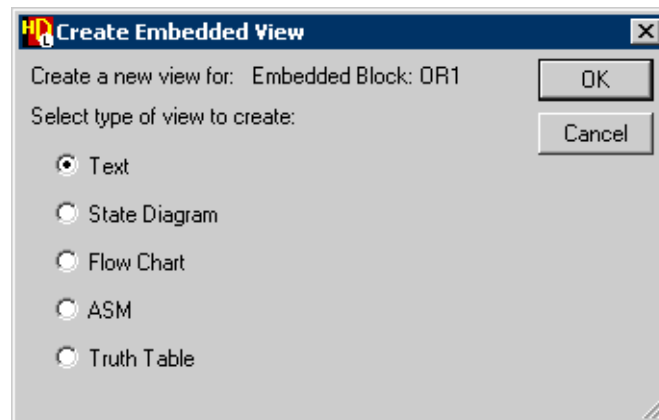
The instance number controls the ordering of the HDL code describing the embedded view in the generated HDL when there are multiple embedded views in the same diagram.

You can edit the name or instance number of the embedded block on a block diagram (or the instance name on an IBD view) by direct text editing. You can edit the name of an embedded block in an IBD view by choosing **Rename** from the popup menu.

You can also edit the name and number by using the **Embedded Blocks** page in the Object Properties dialog box which is displayed when you use the  button or choose **Object Properties** from the **Edit** menu.

Opening an Embedded View

You can open the existing *embedded view* of an embedded block by choosing **Open** from the popup menu or by double-clicking over the embedded block on the block diagram or IBD view. If no view exists, the Create Embedded View dialog box is displayed.



The dialog box allows you to create an embedded *flow chart*, *state machine*, *truth table* or *HDL text* view. However, each embedded block can have only one view.

Note



A flow chart, state machine or truth table created in this way is saved as part of the same design unit and not as a child hierarchical design unit. The embedded view shares the same undo log as the parent view and any Undo (or Redo) command will perform the last command executed in either view.

If an embedded block has an existing view, you can choose **Change Embedded View** from the popup menu. You are prompted whether to delete the existing view and the Create Embedded View dialog box is displayed for you to open a new view.

When you create an embedded flow chart, state machine or truth table, the embedded view is named by adding the embedded block name to the parent view name separated by a colon and displayed in a new tab added to the active window.

For example, if the block diagram *TEST\Counter\Struct* contains an embedded block named *Decode* which is defined by a truth table, the truth table is titled *TEST\Counter\Struct:Decode*.


An embedded state machine or flow chart can also include hierarchical or concurrent diagrams which are named in the usual way (by appending the object name in square brackets). However, if more than one embedded state machine block is included on any view, the state vector is not written and only one ENUM attribute is declared.

VHDL architecture declarations or Verilog module declarations are disabled in an embedded flow chart, state diagram or truth table. However, these can be entered on the parent view containing the embedded block.

Adding Embedded HDL Text

When you create an embedded HDL text view on a block diagram, a default text object containing the embedded view name and number as comment text is added on the diagram.

This object is associated to the embedded block by an *anchor* but can be moved independently by dragging with the mouse. You can resize the text object by dragging its resize handles or change the visibility of the text to hide or show it on the diagram.

You can edit the HDL text by direct text editing on a block diagram or by using the **Text** page of the Object Properties dialog box which is displayed when you use the  button or choose **Object Properties** from the **Edit** menu.

Refer to “[Editing Text Properties](#)” on page 58 for more information about the **Text** tab of the Object Properties dialog box.

You can also open an embedded HDL text view in the text editor by selecting the text object (or the embedded block instance in an IBD view) and choosing **Send To Editor** from the popup menu, by choosing **Open** from the popup menu or by double-clicking on the embedded block instance.

Note



If you double-click on an embedded block when the HDL text view is already open in the editor, you are prompted whether to finish the edits.

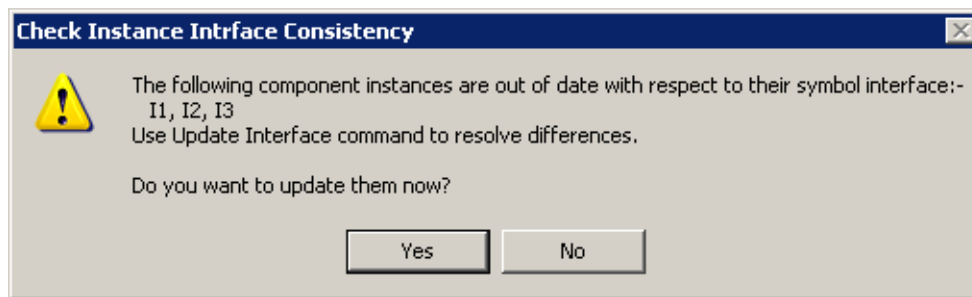
Any valid HDL statements for the current hardware description language can be entered in free format (including line breaks and indentation which are preserved when the text is displayed on a diagram) but each statement must be terminated by a semi-colon.

The syntax is automatically checked for the hardware description language of the active view. However, syntax checking can be disabled by unsetting a preference. If you change the language for the current view, syntax errors are reported when you close the HDL text view.

The HDL code is inserted after the first BEGIN statement in the generated HDL for the block diagram or IBD view. When there are more than one embedded HDL text views in the same view, they are treated as concurrent statements.

Updating an Instance

You can know the instances that need to be updated in IBD editor, by choosing **Instance Interface Consistency** from the **Table** menubar option. Instances that need to be updated will be highlighted in red. You will be prompted if all instances are up-to-date. If any instances are out-of-date, the following dialog box appears.



You can update instances later by clicking **No**. If you choose to update instances now, instances are updated but the red highlight will continue to appear until you clear it by clicking **Clear Net Highlight** from the menu bar.

Note



If you try generating an IBD that has out-of-date instances, an error message appears in the Log window listing the instances that need to be updated.

You can update a component instance with the latest interface defined by its symbol by choosing one of the options from the **Update** cascade of the **Diagram** or popup menu when the component is selected.

If you choose **Interface**, the Reconcile Interface dialog box is displayed as described in [“Reconciling Interfaces”](#) on page 128. This command examines the interface defined in the child view and allows you to reconcile any differences by updating either the child or parent views. New ports added in the graphics or text will be synchronized (i.e. added to the other description). Deleted ports will be synchronized (i.e. removed from the other description).

Note

Type differences (e.g. wire/reg etc.) are not updated in either direction for Verilog designs with text leaf-level descriptions.

You can choose **Interface and Graphics** when the symbol defining the interface to a component has changed and you want to update its instantiation on the block diagram or IBD view. This command attempts to preserve the instance size and port positions.

Alternatively, you can choose to update **From Symbol** when you want to replace the selected component including any graphical layout changes specified in the symbol. The connectivity is preserved although nets may have to be moved if the port positions have changed on the symbol.

You can also choose **Graphics To Symbol** when you want to update the symbol with changes to the layout, appearance, port placement or visibility from the component instance. You must have write permissions to the component to use this command. If there are any interface differences on the symbol, you are prompted to resolve them.

Similarly, you can update a block instance with the interface information from a child HDL text view by choosing **From HDL** from the **Update** cascade in the popup menu.

If any of the interface ports have been renamed, the port is disconnected when you update the instance but the overlapping ports and signals can be reconnected using the **Connect** option which is available from the **Diagram** menu or popup menu in a block diagram.

Note that Verilog is case sensitive but VHDL is case insensitive, therefore port names will be disconnected if you change the case of a Verilog port name but not if you change the case of a VHDL port name.

If VHDL generic or Verilog parameter declarations are defined on the symbol, their values are not updated on the component and may need to be edited using the Object Properties dialog box.

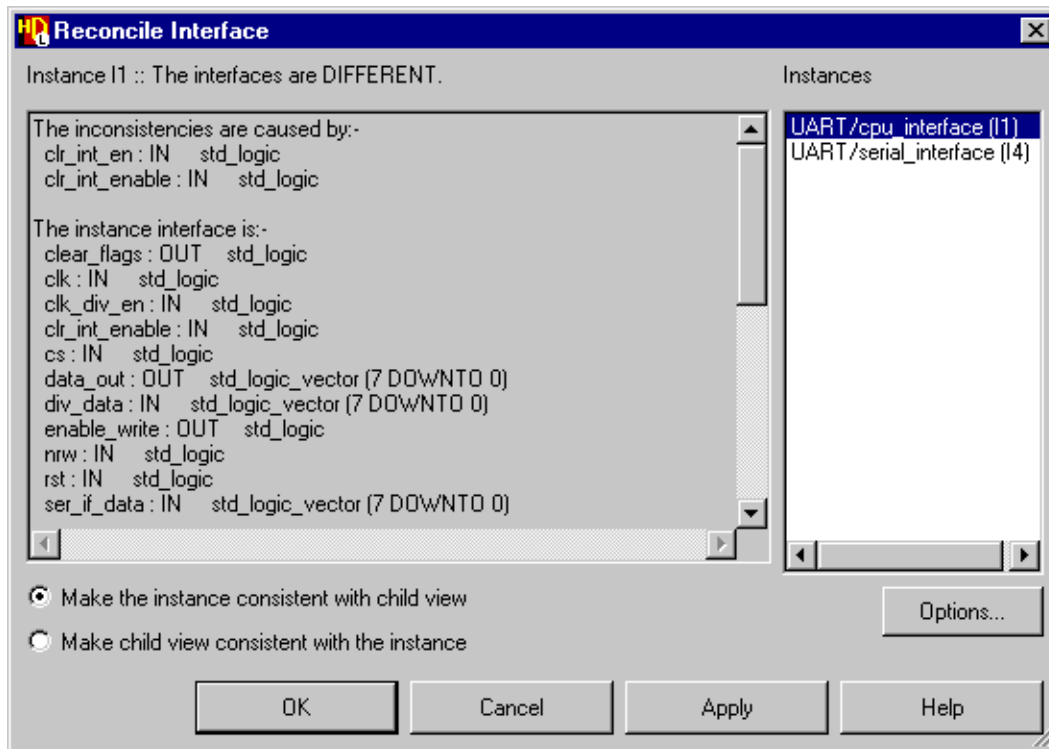
Reconciling Interfaces

When you create a new block diagram or IBD view its external ports are defined by the parent view. These ports are defined by the symbol for a component or by the connections to the parent view for a block. However, the interfaces may become inconsistent and need to be reconciled after a diagram has been edited.

Note

You are automatically prompted to update the interface when you save or close a block diagram or IBD view after changing the port declarations.

You can reconcile interfaces by choosing **Interface** from the **Update** cascade of the **Diagram** menu in a block diagram or from the **Table** menu in an IBD view to display the Reconcile Interface dialog box.



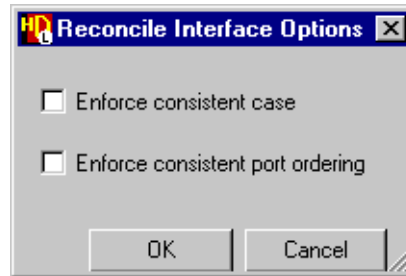
If no block or component instance is selected, this command reconciles interfaces with the parent view.

If a block or component instance is selected, the command is also available from the popup menu and you can reconcile the interface to the selected child view.

You can also compare the interface of a component representing external IP with the interface defined in the external HDL file. However, in this case only the instance can be updated.

If more than one block or component instance is selected, all the views are listed and can be reconciled by selecting an instance name from the list.

By default, reconcile interface ignores any differences due to inconsistent case or port ordering but you can set **Enforce consistent case** or **Enforce consistent port ordering** by using the Options button to display the Reconcile Interface Options dialog box.



If you change either of these options, the new default is saved as a preference.

Inconsistent case or port ordering is not reported until all other inconsistent interfaces have been successfully reconciled.

Note



If there are any changes to the ports defined in the symbol, any comments added in the symbol are reconciled. However, changes to the comments in the symbol are ignored if there are no changes to the interface declarations.

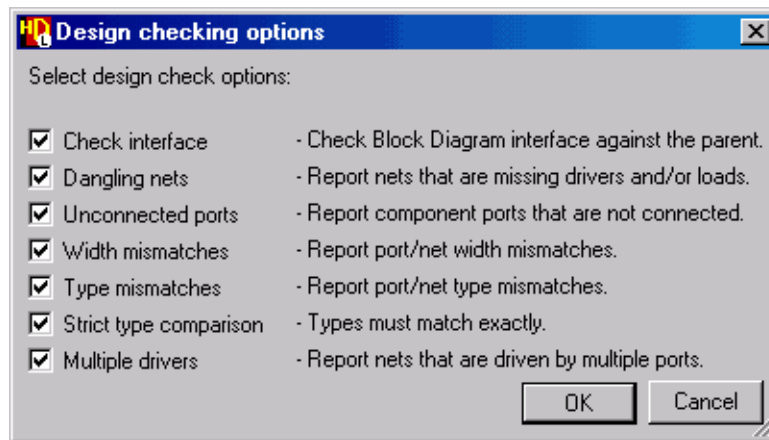
You can use reconcile interfaces to add or remove ports or propagate changes to signal properties between views. However, you cannot use reconcile interfaces to propagate changes to signal names. Refer to [“Propagating Net Changes”](#) on page 166 for information about propagating a signal change through hierarchical views.


If you change a signal name and reconcile the interfaces, a port and stub signal are added to the related view for the new name and the old signal is disconnected.

Any inconsistencies are reported and you can choose to automatically apply changes to make either the current view or the related (parent or child) interface consistent.

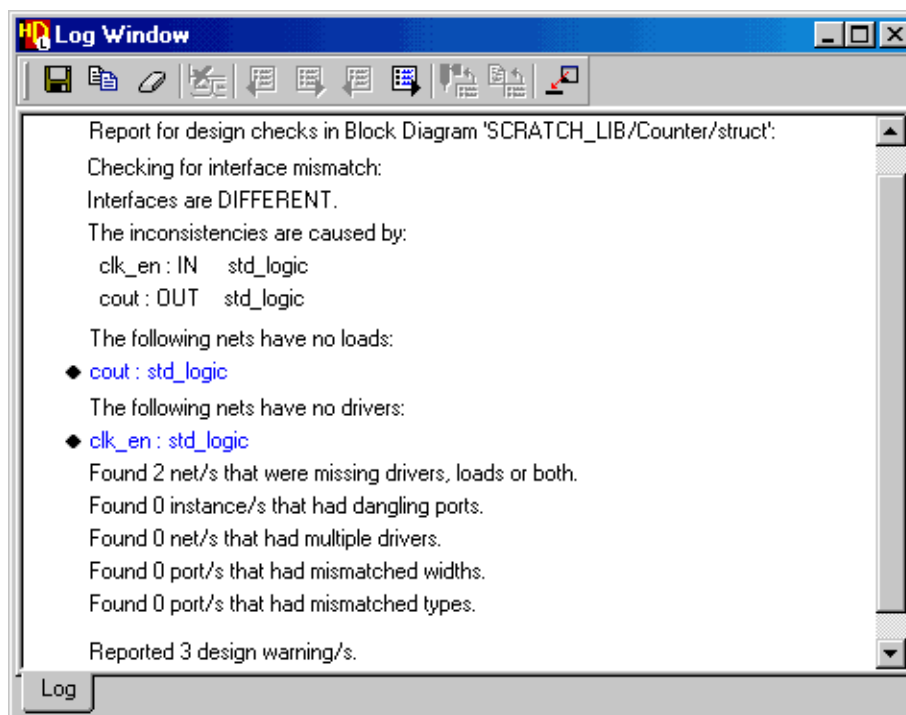
Checking the Design

You can set options for checking the connections in a block diagram or IBD view by choosing **Design Checking Options** from the **Diagram** menu in a block diagram or the **Table** menu in an IBD view to display the Design Checking Options dialog box:



You can check the connections in the design by using the  button or choosing **Check Design** from the **Diagram** menu in a block diagram or from the **Table** menu in an IBD view.

This command issues a report to the log window listing any mismatch with the interface and any nets which have no drivers (*source* is unconnected) or no loads (*destination* is unconnected). For example:



An error message is issued in the log window if the option to check the interface is set and the view has not been saved. If the view has been saved, any inconsistencies with the interface are reported.

You can check the block diagram or IBD view for unconnected (dangling) nets or ports and for any nets connected to ports with a different width or type. These checks include unconnected nets contained in a bundle or connected only to a *global connector* or *ripper*.

You can set strict type comparison to report any unmatched types or unset this option to allow nets connections with a related type (for example: *std_ulogic* and *std_logic*).

You can also check for nets that are driven by multiple ports.

Note that you can click on the net names reported in the log window to cross-reference to the nets on the block diagram or IBD view.

Checking Through Hierarchy

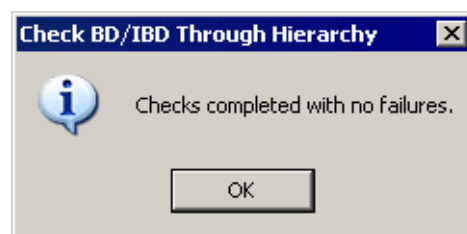
You can check all block diagram and IBD views hierarchically by selecting any design unit in the *design explorer* and choosing **Check BD/IBD Through Hierarchy** from the **Tools** or popup menu.

If any of the block diagrams or IBD views in the hierarchy violate the current design checking options, a report window is displayed in the design explorer showing these views in an expandable list.

Note



If no violations are detected, a message is raised informing you that the check is complete without any failures found as shown in the figure below.





You can open any of these views by double-clicking or choosing Open File from the popup menu. The design check violations for each view are shown in the Log window when they are opened.

Editing Object Properties

You can display and edit properties for objects in a Block Diagram or IBD view through the Object Properties dialog box.

To display the Object Properties dialog box:

1. Select an object in the Block Diagram or IBD view.
2. Do one of the following:
 - Click the  button on the shortcut bar.
 - Use the **Alt** +  shortcut.
 - Choose **Object Properties** from the **Edit** menu.
 - Choose **Object Properties** from the cascade popup menu.

The left pane of the dialog box displays a list of the block diagram objects. These include Components, Blocks, Embedded blocks, Moduleware, External IPs, Signals, Bundles, PortIOs, Frames, Comment Text, Comment Graphics and User Declarations. You can use + to expand or collapse an object list.

Note



The **Generics** Sub-page is available for Components, Blocks and External IPs if you are using VHDL or the **Parameters** page if you are using Verilog. The **Bundles**, **PortIO**, **Comment Text** and **Graphics** pages are not available in an IBD view.

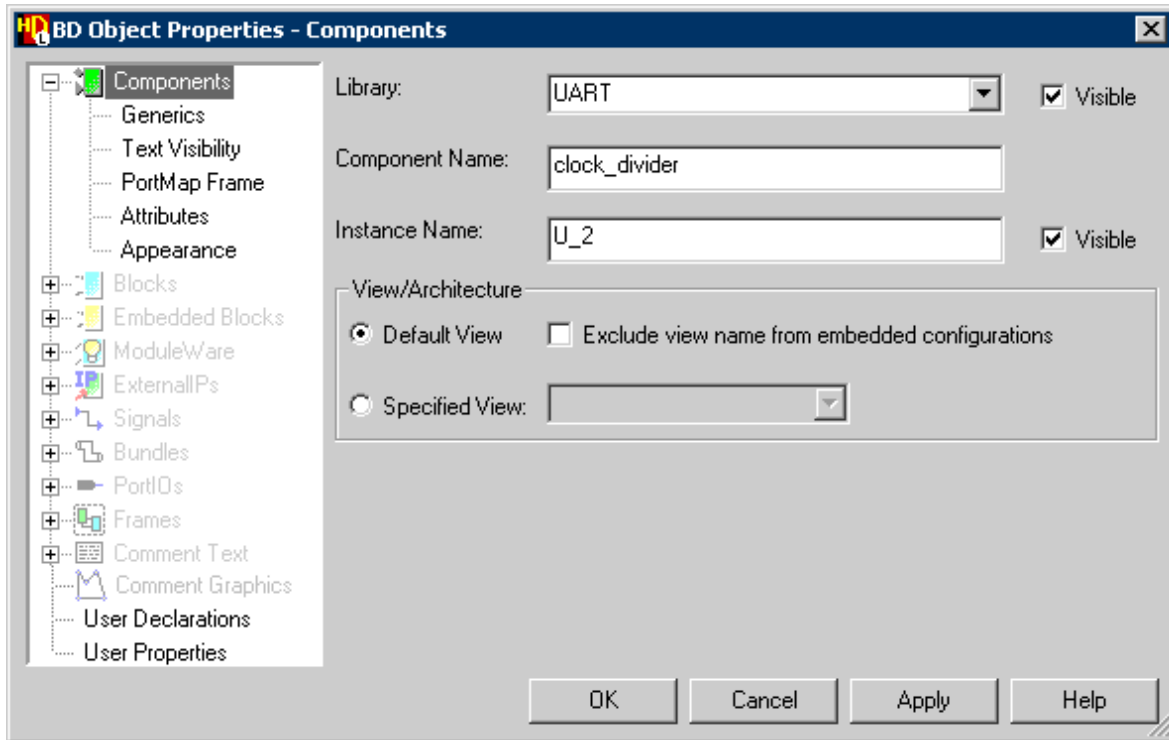
Selected objects are displayed in black and their corresponding pages are enabled. Objects not in the selection set are displayed in grey and their corresponding pages are disabled.

The right pane of the dialog box displays the selected object properties page.

Editing Component Properties

The **Components** object is selected if you display the Object Properties dialog box when one or more *component* is selected on a block diagram or IBD view.

The components page of the BD Object Properties dialog box allows you to specify the library, component name and Instance name.

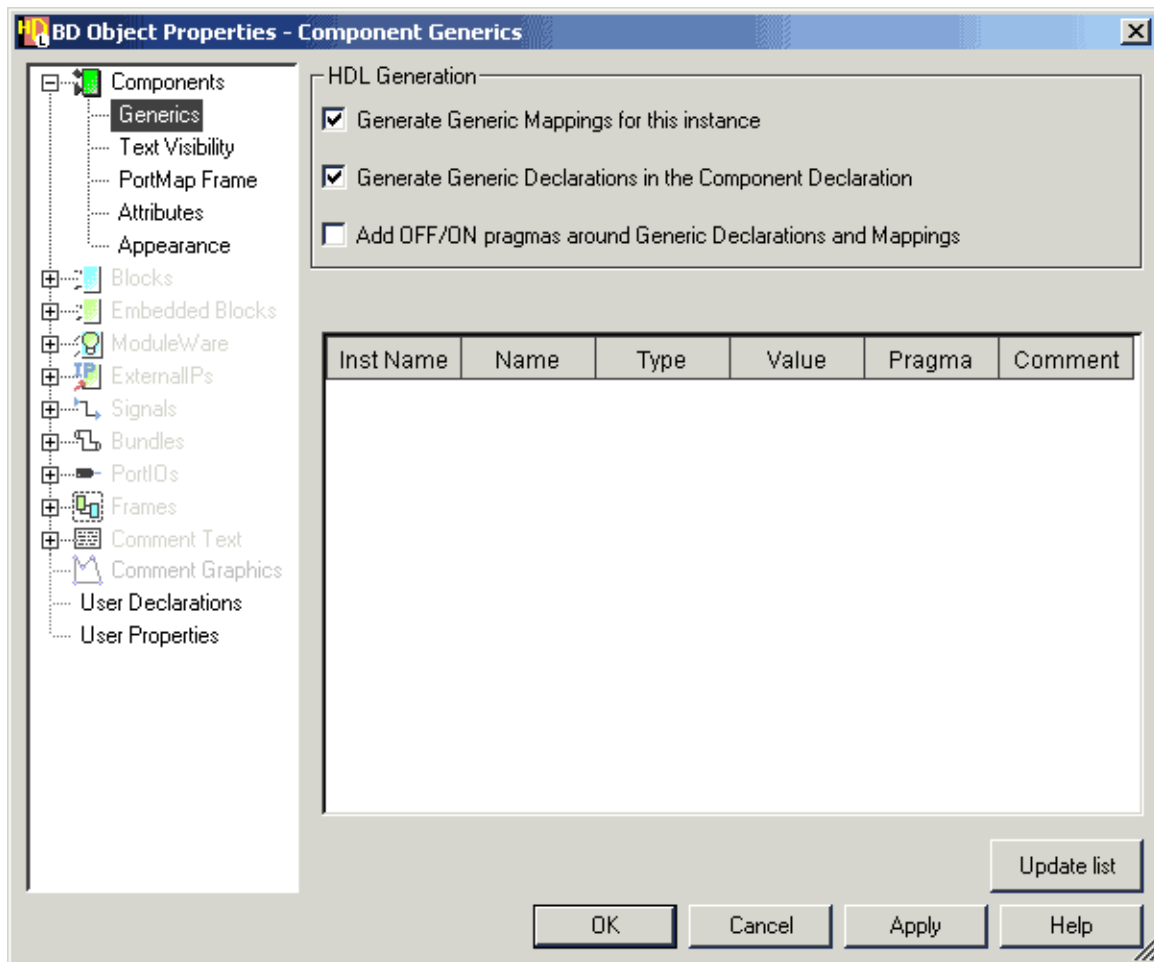


For a VHDL component, you can also change the instantiated view. This can be the default view or any other named view which exists for the component. If you choose the default view, it can optionally be included with no architecture reference. When you are using Verilog, the default view is always used and the Architecture options are not available.

Editing Component Generics and Parameters

The **Generics** page for the Component Object of the BD Object Properties dialog box allows you to override the default Generics that have been already defined on the component symbol using the Symbol editor. In addition it allows you to specify HDL Generation Settings.

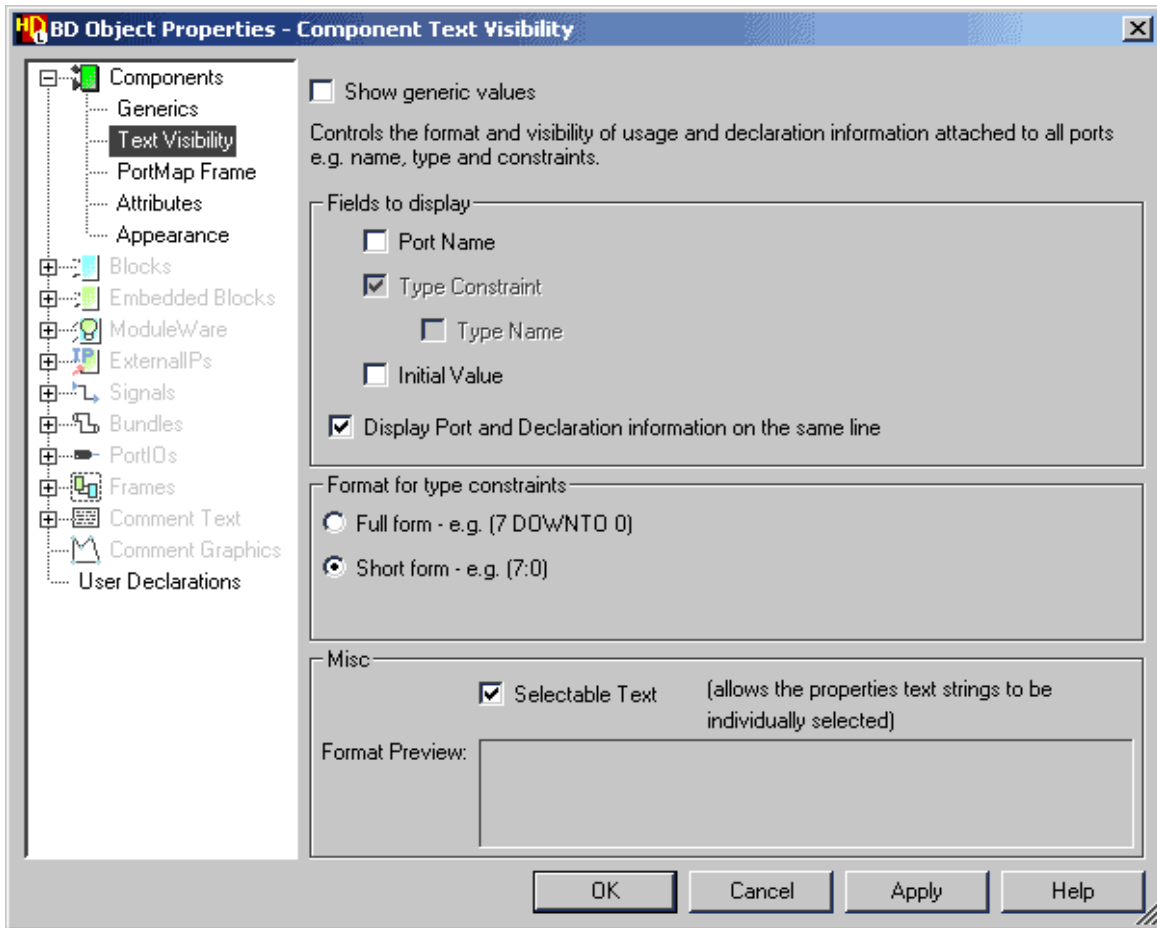
The same applies to the **Parameters** page which is available if you are using Verilog as your design language.



Refer to [“Defining Generics and Parameters”](#) on page 184 and [“Editing Generics and Parameters for Instances”](#) on page 186 for more information.

Editing Component Text Visibility

The **Text Visibility** page for the Component Object of the BD Object Properties dialog box allows you to set the visibility of VHDL generic or Verilog parameters and also allows you to modify port display properties for the selected component instance. Refer to [“Changing the Display of Port Properties”](#) on page 206 for more information.



Editing Component Port map Frame

The **Port Map Frame** page for the Component Object of the BD Object Properties dialog box allows you to enable (or disable) a port map frame and edit the mapping between formal ports on the component and the actual signals.

When **Enable Port Map Frame** is set, mapping generated automatically by direct connections is shown in read-only list with a separate editable list where you can explicitly map the ports and signals. When **Connection by Name** is set, any signal connected to the frame with the same name as a component port is implicitly connected.

When you enable a port map frame, a template port map for all unconnected component ports is shown as an editable comma separated list in the dialog box. For example:

VHDL

```
portA => ,  
portB => ,
```

Verilog

```
.portA() ,  
.portB() ,
```


VHDL

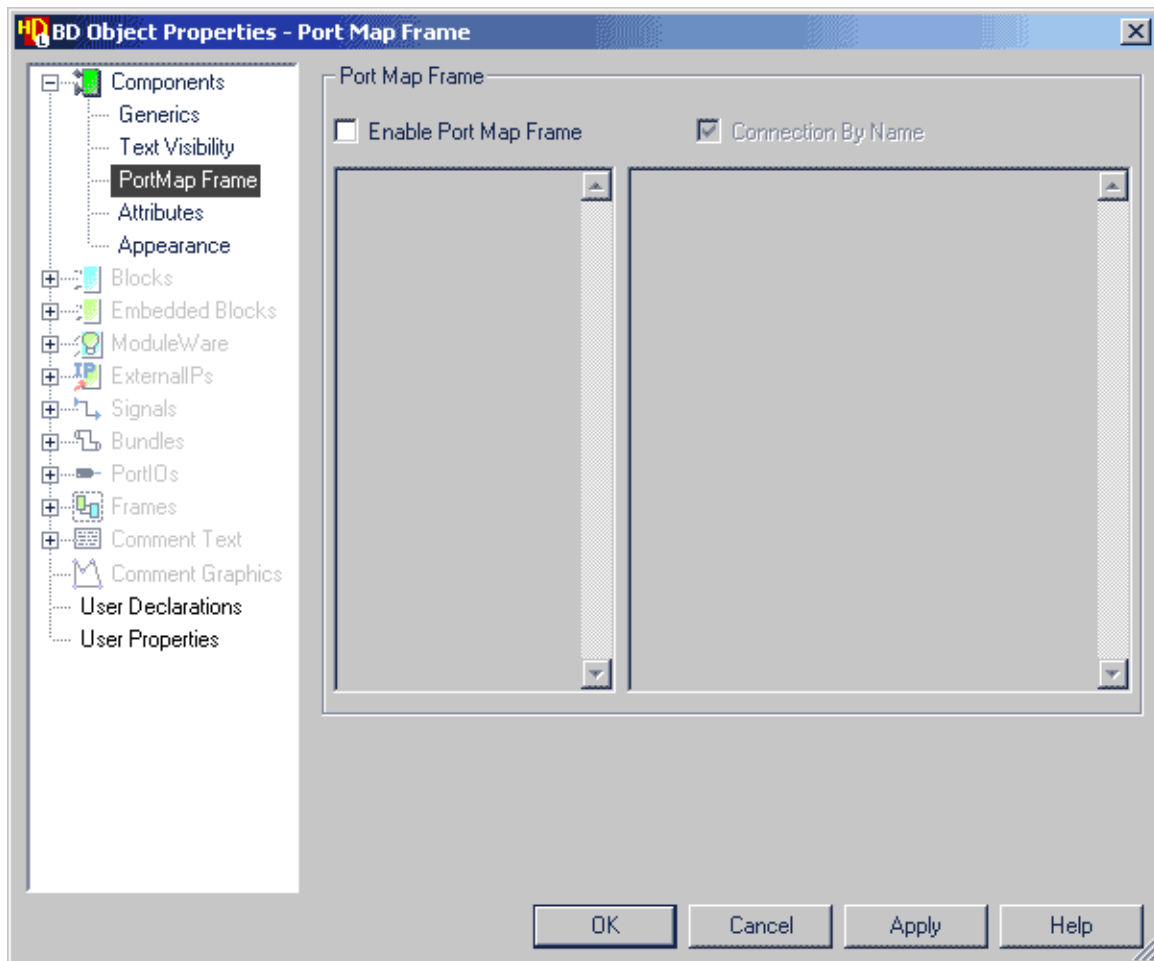
portC =>

Verilog

.portC()

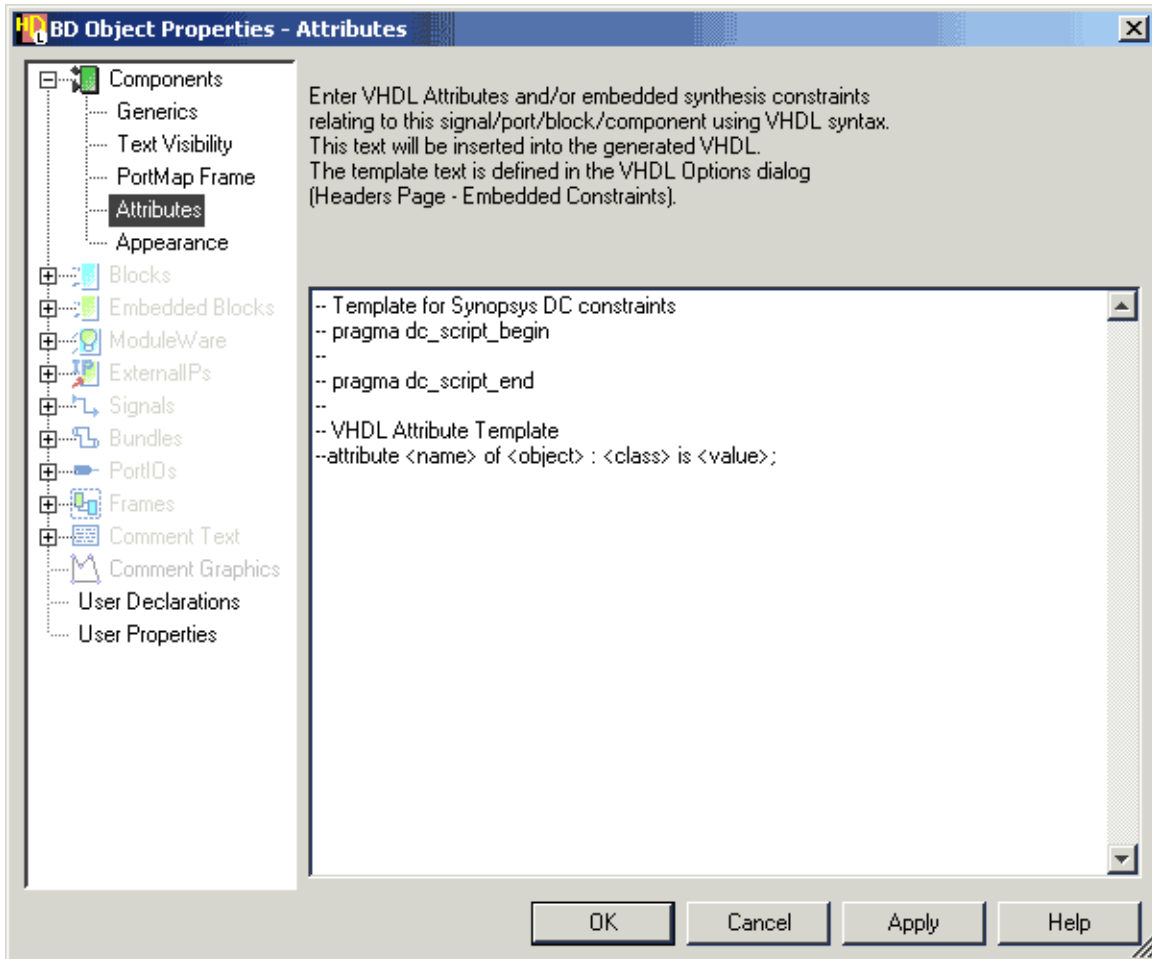
You can enter actual VHDL signal names after the => operator or actual Verilog signal names inside the parentheses (). Ensure that you remove any duplicate entries from the editable list if they already exist in the automatically generated list before attempting to generate HDL.

Refer to [“Port Map Frames”](#) on page 283 for more information about using port map frames.



Setting Component Attributes and Embedded Constraints

The **Attributes** page for the Component Object of the BD Object Properties dialog box allows you to set attributes and embedded constraints for the selected components. Refer to [“Setting Attributes and Embedded Constraints”](#) on page 165 for more information.



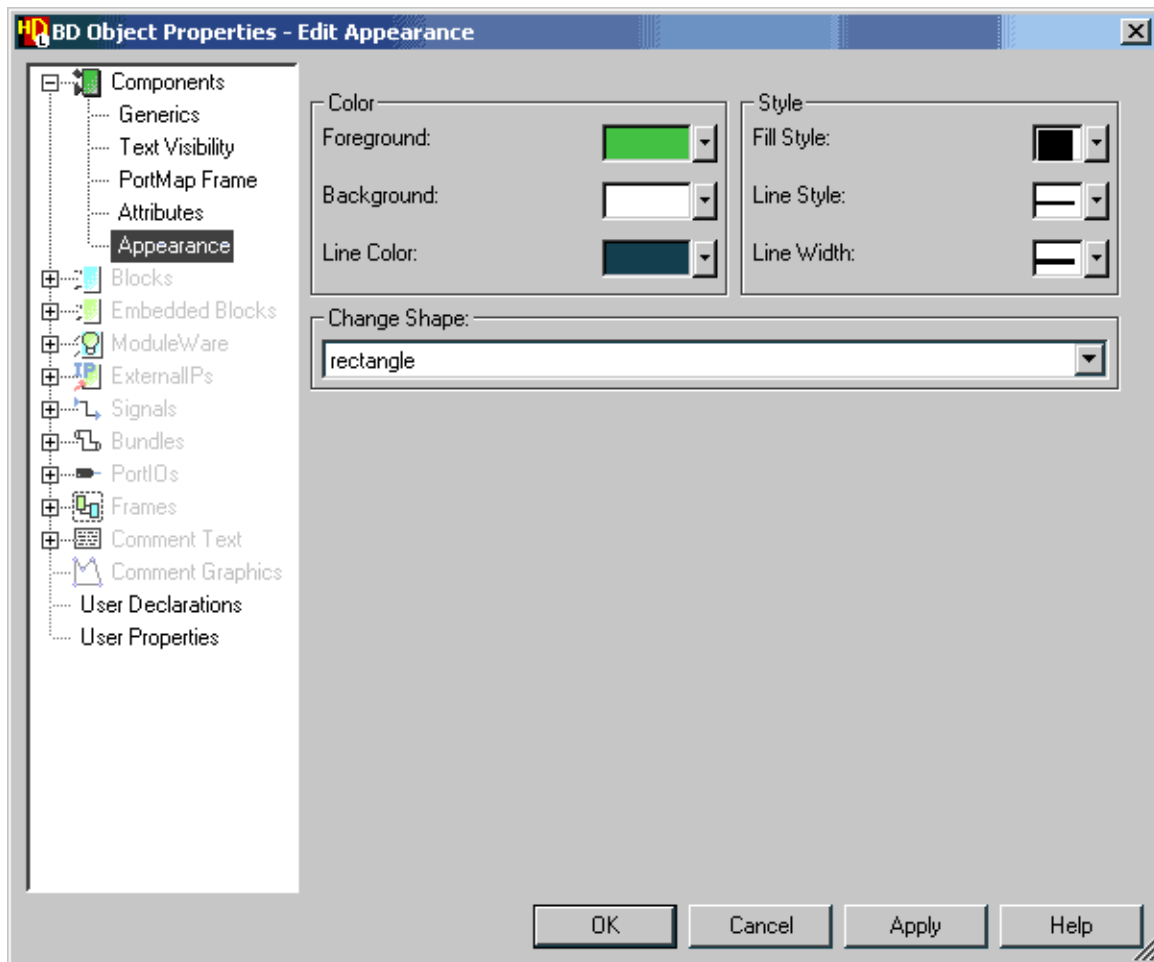
Modifying Component Appearance

The **Appearance** page for the Component Object of the BD Object Properties dialog box allows you to set visual attributes for the selected components.

The attributes include the foreground, background and line color, the line style, fill style, line width. You can set the font used. You can also change the appearance of a component by choosing **Appearance** from the popup menu.

You can change the shape of the selected blocks by choosing a standard logic shape from a drop-down list or by choosing **Autoshape** from the popup cascade menu to choose from a list of standard shapes. You can also define your own shape by selecting **Editing Shape** from the popup cascade menu.

Refer to “Changing the Shape of a Block or Component” on page 228 for more information.



Editing Block Properties

The **Blocks** object is selected if you display the Object properties dialog box when a block is selected on a block diagram or IBD view.

The Blocks page of the BD Object Properties dialog box allows you to specify the Library, Block name and Instance name.

Note



The block and instance name cannot be the same when you are using VHDL but can be given the same name if you are using Verilog.

You can change the library for all the selected blocks but the instance name and block name must be unique HDL identifiers and can only be edited if a single block is selected.

When the dialog box is displayed from a block diagram, you can choose whether the library and instance name are visible on the diagram.

You can also set the visibility of the declarations and values for VHDL generics or verilog parameters on the diagram and choose whether ports are shown for nets connected to the block.

Refer to [“Opening Block and Component Views”](#) on page 194 for more information.

Editing Block Generics and Parameters



The **Generics** page for the Block Object of the BD Object Properties dialog box allows you to specify HDL Generation Settings.

The dialog box also lists any existing VHDL generic declarations and allows you to change the values mapped for each instance. The same applies to the **Parameters** page which is available if you are using Verilog as your design language.

Refer to [“Defining Generics and Parameters”](#) on page 184 and [“Editing Generics and Parameters for Instances”](#) on page 186 for more information.

Modifying Block Port Ordering

The **Port Ordering** page for the Block Object of the BD Object Properties dialog box allows you to modify the port ordering by choosing the Manual option to disclose an ordered list. The list shows how the port declarations will be ordered in the generated HDL for the block

One or more port declarations can be selected and moved up and down the list by using the  and  buttons. If you choose **Automatic** the list is ordered alphanumerically.

You can modify the port ordering for a block by choosing the **Manual** option in the dialog box to disclose an ordered list showing how the port declarations will be ordered in the generated HDL for the block.

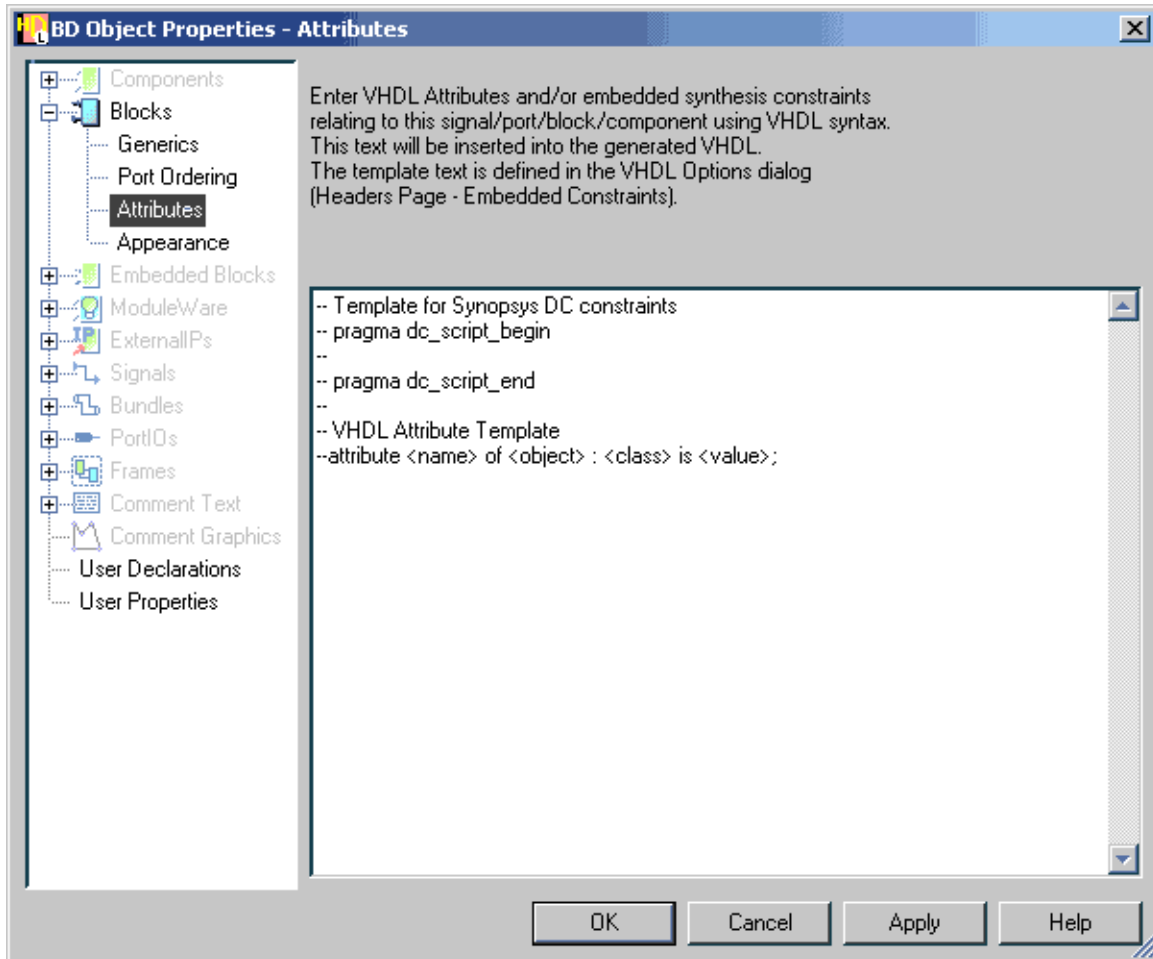
Note



If a block has been set to use manual port ordering, then manual ordering is used when you create a child block diagram or IBD view. Consistent port ordering can optionally be enforced when you reconcile interfaces.

Setting Block Attributes and Embedded Constraints

The **Attributes** page for the Block Object of the BD Object Properties dialog box allows you to set attributes and embedded constraints for the selected blocks. Refer to “[Setting Attributes and Embedded Constraints](#)” on page 165 for more information.

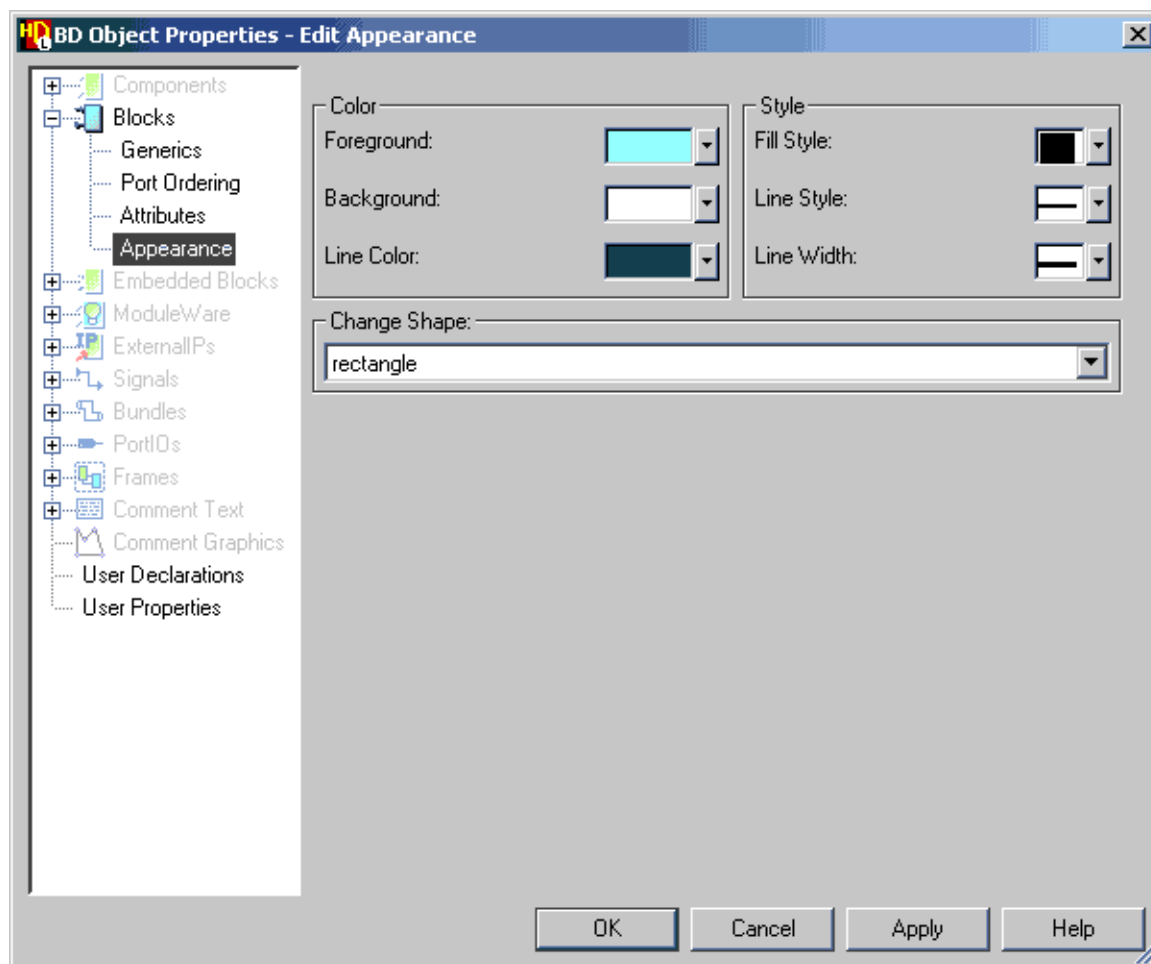


Setting Block Appearance

The **Appearance** page for the Block Object of the BD Object Properties dialog box allows you to set visual attributes for the selected blocks.

The attributes include the foreground, background and line color, the line style, fill style, line width. You can set the font used.

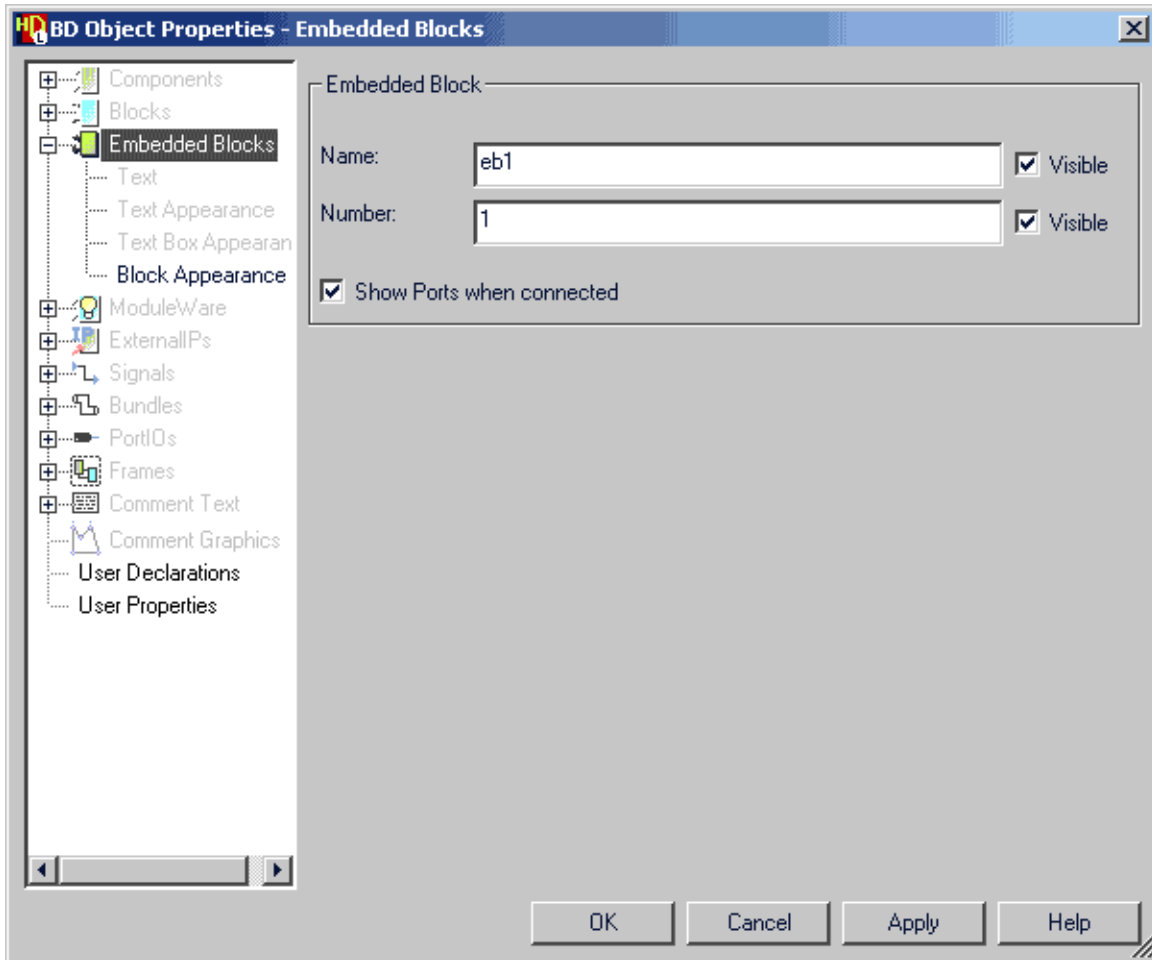
You can also change the shape of the selected blocks by choosing a standard logic shape from a drop-down list.



Refer to [“Changing the Shape of a Block or Component”](#) on page 228 for more information.

Editing Embedded Block Properties

The **Embedded Blocks** page is selected if you display the Object Properties dialog box when an embedded block is selected on a block diagram or IBD view.



The Embedded Blocks page allows you to specify the name and number of the embedded block. The name must be a unique HDL identifier in the view. The number must also be unique and determines the order in which the HDL for unconnected embedded blocks is included in the HDL for the view.

If you enter a number which is already used by another embedded block in the same design unit view, the numbers are swapped. The name and number can only be edited if a single block is selected.

When the dialog box is displayed from a block diagram, you can choose whether the name and number are visible on the diagram and choose whether ports are shown when nets are connected to the embedded block on the diagram.

Editing Embedded Blocks HDL Text

The **HDL Text** page for the Embedded Block Object of the BD Object Properties dialog box allows you to edit the existing HDL text.

Any valid HDL statements for the current hardware description language can be entered in free format (including line breaks and indentation) but each statement must be terminated by a semi-colon.

The syntax is automatically checked for hardware description language of the active view. However, syntax checking can be disabled by unsetting a preference.

You can use the modify check box to apply the new text to all the selected embedded HDL text views on the diagram.

You can also modify the text position in the bounding box.

Modifying Embedded Blocks Text Appearance

The **Text Appearance** page for the Embedded Block Object of the BD Object Properties dialog box allows you to modify the font of the comment text.

Modifying Embedded Blocks Text Box Appearance

The **Text Box Appearance** page for the Embedded Block Object of the BD Object Properties dialog box allows you to set visual attributes for the selected blocks text box.

The attributes include the foreground, background and line color, the line style, fill style, line width.

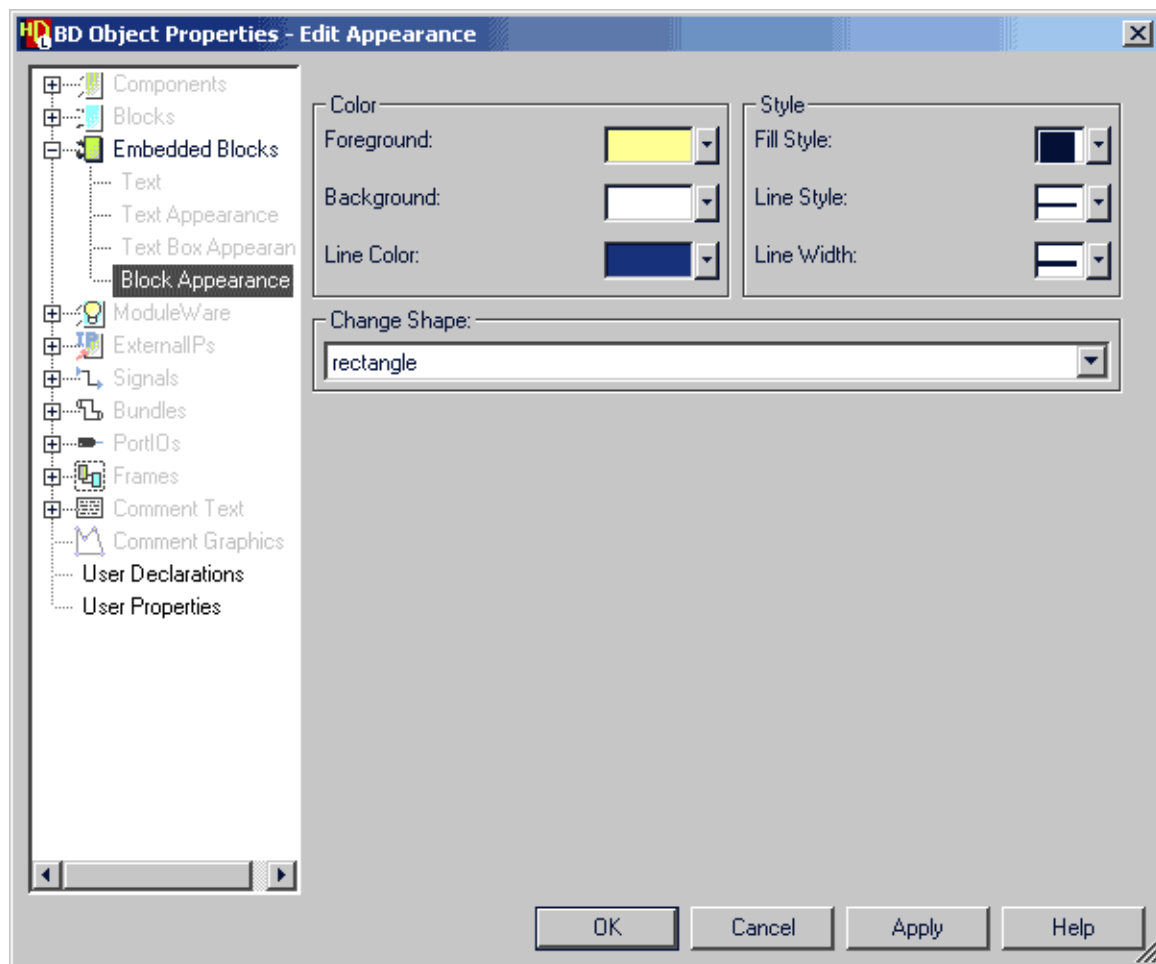
Modifying Embedded Blocks Appearance

The **Appearance** page for the Component Object of the BD Object Properties dialog box allows you to set visual attributes for the selected embedded blocks.

The attributes include the foreground, background and line color, the line style, fill style, line width. You can set the font used.

You can also change the shape of the selected blocks by choosing a standard logic shape from a drop-down list.

Refer to [“Changing the Shape of a Block or Component”](#) on page 228 for more information.



Editing ModuleWare Properties

The **ModuleWare** object is selected if you display the Object properties dialog box when a moduleware is selected on a block diagram.

The Moduleware page of the BD Object Properties dialog box allows you to specify the Moduleware Instance name. The library and the component name fields can not be modified.

You can choose to display the library and instance name on the moduleware symbol.

You can use the Moduleware parameters button to display the ModuleWare Parameters dialog box. Refer to [“Editing ModuleWare Parameters”](#) on page 117 for more information.

Editing Moduleware Port map Frames

The **Port Map Frame** page for the Moduleware Object of the BD Object Properties dialog box allows you to enable (or disable) a port map frame and edit the mapping between formal ports on the Moduleware and the actual signals.

When **Enable Port Map Frame** is set, mapping generated automatically by direct connections is shown in read-only list with a separate editable list where you can explicitly map the ports and signals. When Connection by Name is set, any signal connected to the frame with the same name as a component port is implicitly connected.

When you enable a port map frame, a template port map for all unconnected ModuleWare ports is shown as an editable comma separated list in the dialog box. For example:

VHDL	Verilog
portA => ,	.portA() ,
portB => ,	.portB() ,
portC =>	.portC()

You can enter actual VHDL signal names after the => operator or actual Verilog signal names inside the parentheses (). Ensure that you remove any duplicate entries from the editable list if they already exist in the automatically generated list before attempting to generate HDL.

Refer to “[Port Map Frames](#)” on page 283 for more information about using port map frames.

Setting ModuleWare Attributes and Embedded Constraints

The **Attributes** page for the Moduleware Object of the BD Object Properties dialog box allows you to set attributes and embedded constraints for the selected ModuleWare components. Refer to “[Editing ModuleWare Parameters](#)” on page 117 for more information.

Editing External IPs Properties

The **External IPs** object is selected if you display the Object properties dialog box when an External IP is selected on a block diagram or IBD view.

The External IPs page of the BD Object Properties dialog box allows you to specify the Instance name.

For a VHDL component, you can also change the instantiated view. This can be the default view or any other named view which exists for the External IP. If you choose the default view, it can optionally be included with no architecture reference. When you are using Verilog, the default view is always used and the view options are not available.

You can use the + to expand the objects list.

Editing External IP Generics

The **Generics** page for the External IP Object of the BD Object Properties dialog box allows you to specify HDL Generation Settings.

Editing External IP Text Visibility

The **Text Visibility** page for the External IP Object of the BD Object Properties dialog box allows you to set the visibility of VHDL generic or Verilog parameters.

The dialog box also allows you to modify port display properties for the selected External IP instance. Refer to [“Changing the Display of Port Properties”](#) on page 206 for more information.

Editing External IP Port map Frame

The **Port Map Frame** page for the External IP Object of the BD Object Properties dialog box allows you to enable (or disable) a port map frame and edit the mapping between formal ports on the External IP and the actual signals.

When **Enable Port Map Frame** is set, mapping generated automatically by direct connections is shown in read-only list with a separate editable list where you can explicitly map the ports and signals. When Connection by Name is set, any signal connected to the frame with the same name as a component port is implicitly connected.

When you enable a port map frame, a template port map for all unconnected component ports is shown as an editable comma separated list in the dialog box. For example:

VHDL	Verilog
portA => ,	.portA() ,
portB => ,	.portB() ,
portC =>	.portC()

You can enter actual VHDL signal names after the => operator or actual Verilog signal names inside the parentheses (). Ensure that you remove any duplicate entries from the editable list if they already exist in the automatically generated list before attempting to generate HDL.

Refer to [“Port Map Frames”](#) on page 283 for more information about using port map frames.

Setting External IP Attributes and Embedded Constraints

The **Attributes** page for the External IP Object of the BD Object Properties dialog box allows you to set attributes and embedded constraints for the selected External IPs. Refer to [“Setting Attributes and Embedded Constraints”](#) on page 165 for more information.

Modifying External IP Appearance

The **Appearance** page for the External IP Object of the BD Object Properties dialog box allows you to set visual attributes for the selected External IP.

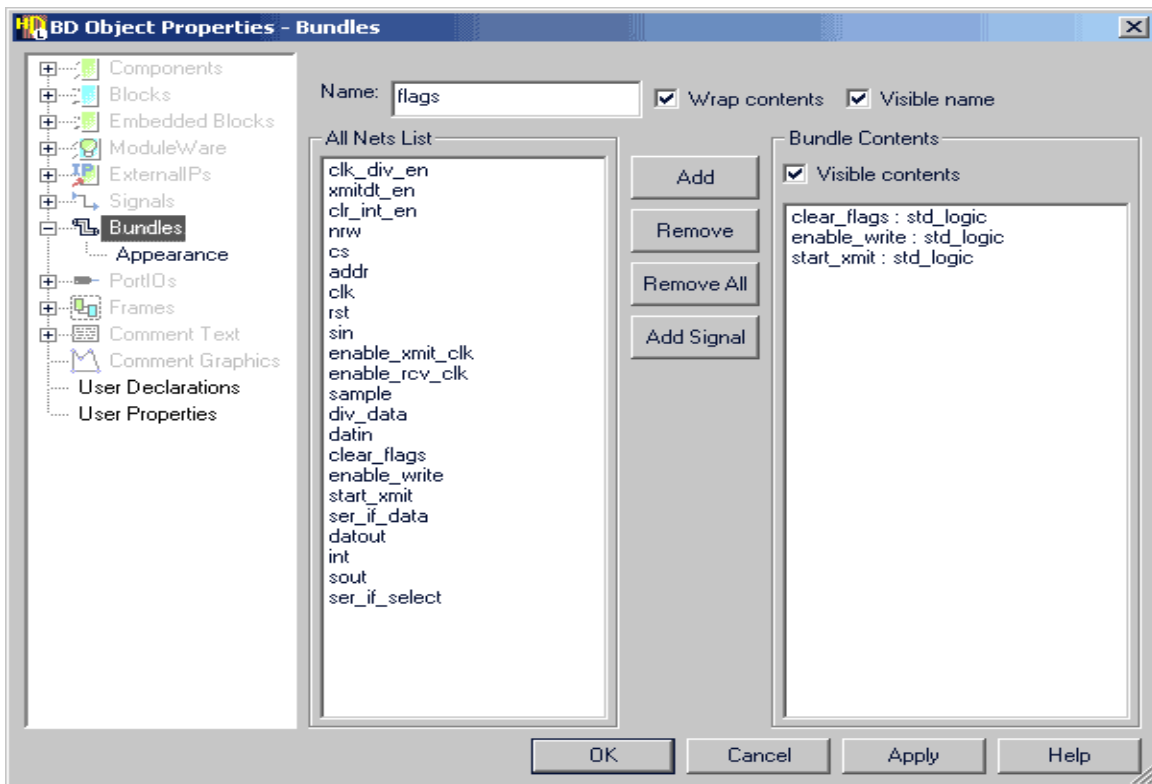
The attributes include the foreground, background and line color, the line style, fill style, line width. You can set the font used.

You can also change the shape of the selected External IP by choosing a standard logic shape from a drop-down list.

Editing Bundle Properties

The **Bundle** object is selected if you display the Object properties dialog box when a Bundle is selected on a block diagram.

The Bundle page allows you to specify the Bundle name. If you change the bundle name all occurrences of the name on the diagram are updated. The visible check allows you to choose whether the bundle name is displayed or hidden on the diagram.



A list of the bundle contents is normally shown on the diagram as a single line below the bundle name but you can check the wrap contents option to list each signal on a separate line.

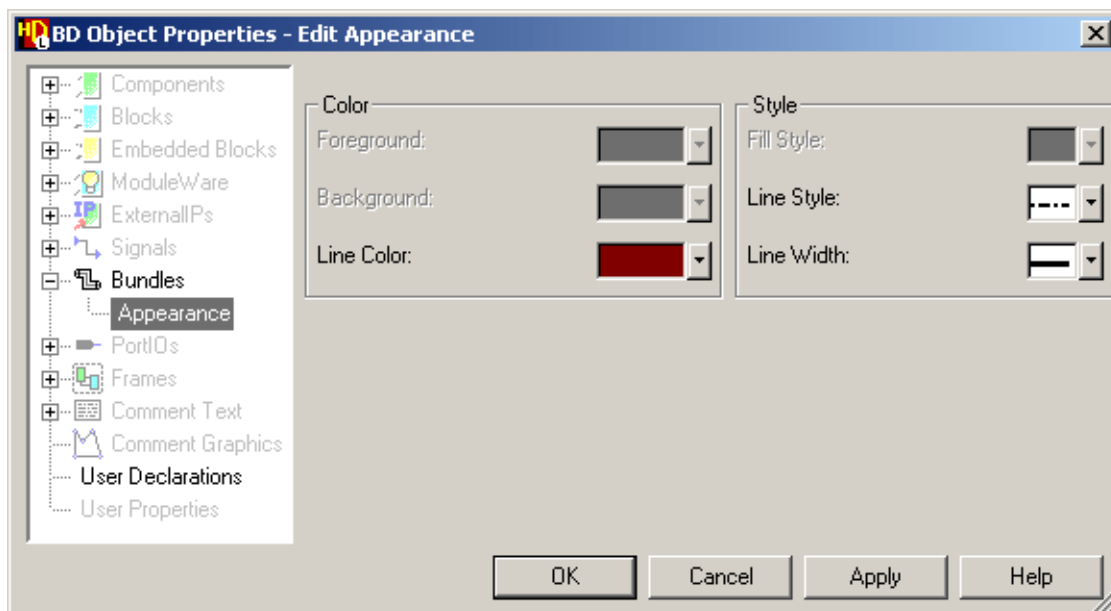
A list of declared nets is displayed on the left hand side of the page, and a list of the bundle contents is displayed on the right hand side. You can add a declared net to the bundle by selecting nets from the left hand side list and clicking “Add”. You can remove a net from the bundle declaration by selecting it from the right hand side list and clicking “Remove”. You can clear all the bundle signals by clicking on “Remove All” button.

You can add a new signal to the bundle contents by clicking the Add Signal button to display the Add Signal dialog box.

Modifying Bundle Appearance

The **Appearance** page for the Bundle Object of the BD Object Properties dialog box allows you to set visual attributes for the selected bundle.

The attributes include the line color, line style and line width.



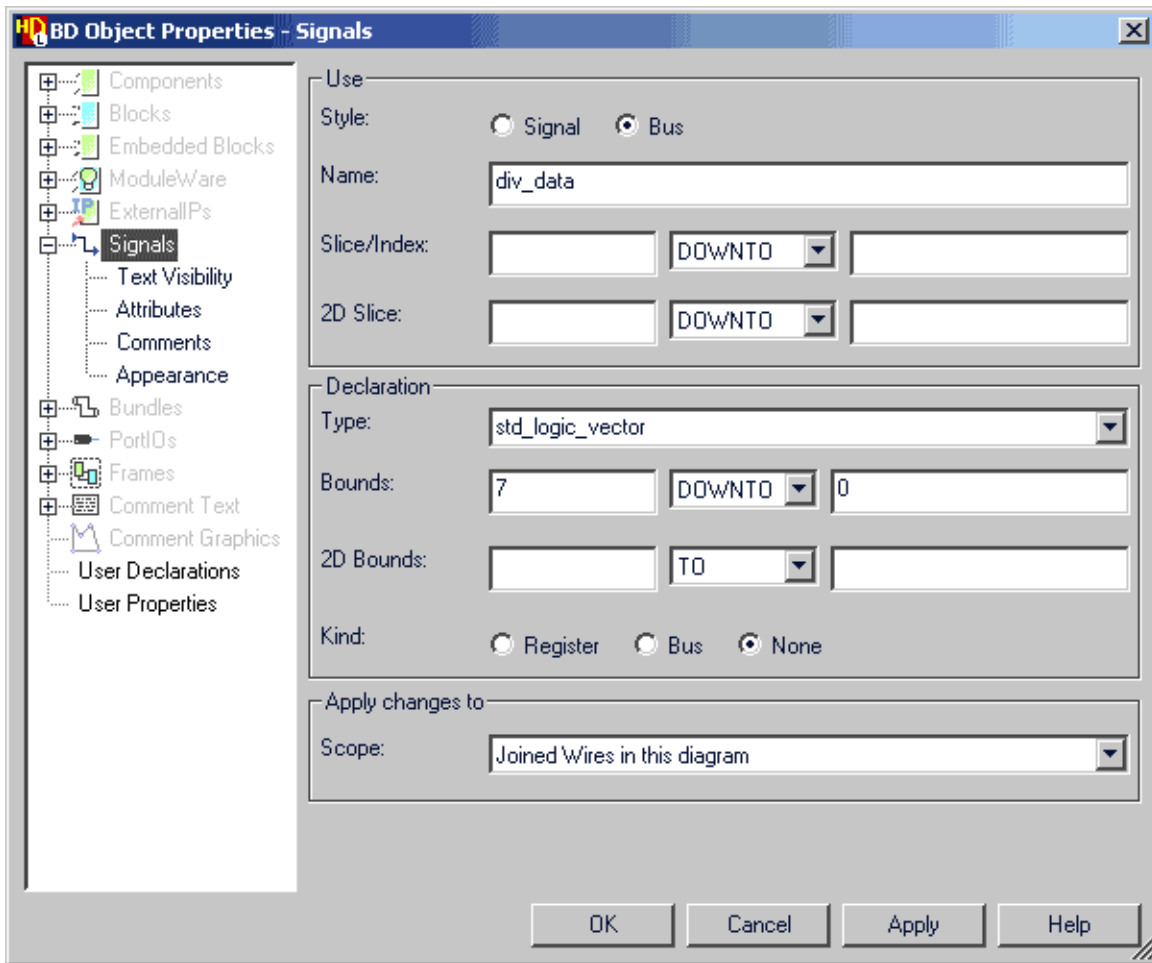
Editing Signal Properties

The **Signals** page is selected if you display the Object properties dialog box when a Signal is selected on a block diagram or IBD view.

The Signal page allows you to edit the declarations of all the signals on a block diagram or IBD view.

Editing VHDL Signal Declarations

The following page is available when you edit signal declarations for a VHDL view:



If you edit the net name, it is changed wherever it is used in the view and must be a valid VHDL identifier.

You can change the type by choosing from a pulldown list of VHDL types or by entering any other valid type. The type must be defined in a VHDL package referenced in the package list or be one of the standard predefined types.

Note



The pulldown list includes all the most commonly used types and any other types which have already been used in the active view. However, you may need to add a new package reference if you choose a type which is not in the currently referenced packages. For example, if you want to use the *signed* or *unsigned* types, the *ieee.numeric_std* package should be referenced.

The bounds can be used to specify the indexes for the elements in an array or the range for a scalar type (for example: *15 DOWNT0 0* or *0 TO 7*).

You can also use the first bounds entry box to enter a user specified range constraint such as an enumerated or integer type name or you can enter an array name or type name of the form:

`<array>'RANGE` or `<array>'REVERSE_RANGE`

You can also add the VHDL keywords **BUS** or **REGISTER** to the net declaration by choosing **Bus** or **Register** kind. (If not specified, the default is **None**.)

Note

Setting initial values for signals can be useful for simulation but may hide the behavior of a circuit that would be in an arbitrary state at power up.

Using 2D Signal Types

You can use the 2D bounds and 2D slice fields of the Object Properties dialog box when you have specified an unconstrained two dimensional array type. This type must be specified in a VHDL package referenced on the block diagram or IBD view or be defined in the user declarations for the view.

For example:

```
type my2dtype is array(natural range <>, natural range <>) of std_logic;
```

You can also define a two-dimensional "vector of vectors" by defining an array of a vector type.

For example:

```
type my_vtype is array(natural range <>) of std_logic_vector(3 downto 0);
```

A two dimensional bounds for signals using this type can then be entered in the dialog box.

Note

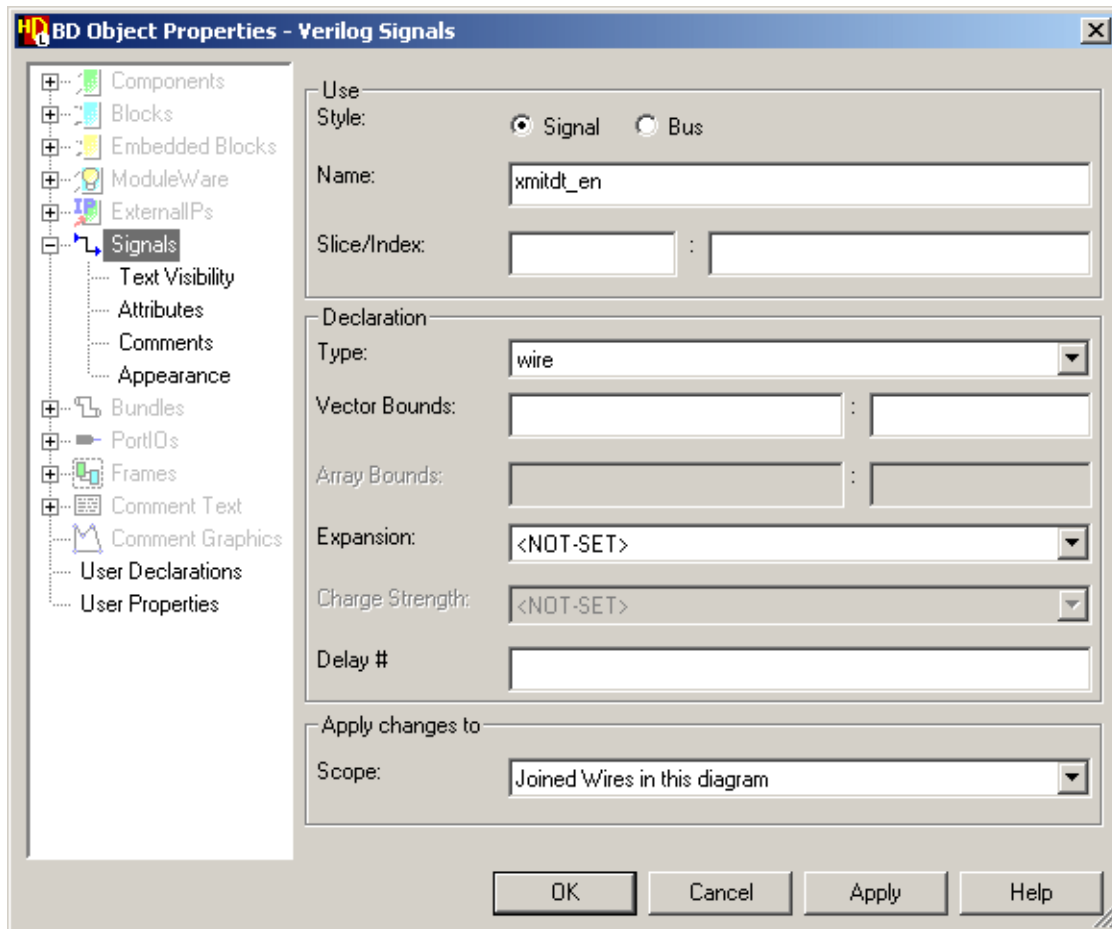
If you use a *std_logic_vector* signal and try to define a two dimensional bounds for this signal using the Object Properties dialog, HDL generation issues an error message of the form:

```
Error, std_logic_vector requires 1 index values.
```

This is because *std_logic_vector* is a one dimensional unconstrained array and cannot be given two ranges.

Editing Verilog Signal Declarations

The following page box is available when you edit net declarations for a Verilog view.



If you edit the signal name, it will be changed wherever it is used in the view and must be a valid Verilog identifier.

You can change the type by choosing from a pulldown list of supported Verilog types.

The vector bounds can be used to enter an expression specifying the index of an element in an array or to specify a range of values (for example: *15:0* or *0:7*).

You can specify the vector bounds for a *wire*, *tri*, *wor*, *trior*, *wand*, *triand*, *tri0*, *tri1*, *supply0*, *supply1*, *reg* or *triereg* type. For signals which connect between embedded blocks, you can specify the array bounds for *integer* or *time* types and for a *reg* type you can specify both a vector and array bounds.

You can specify expansion and charge strength options:

You can choose whether the expansion options are **scaled**, **vectored** or not set. The delay and expansion options are not available when *reg*, *integer*, *time*, *real* or *realtime* type is selected.

When the *trireg* type is selected, you can choose the charge strength from **small**, **medium**, **large** or not set.

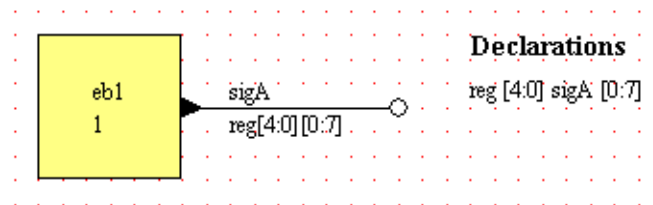
Verilog Arrays

Although Verilog does not support multi-dimensional arrays, it does distinguish between vectors (single elements which are n bits wide) and arrays (multiple elements which are 1 or n bits wide). For example:

<code>integer count[0:7]</code>	an array of 8 count variables
<code>reg bool[31:0]</code>	an array of 32 1-bit register variables
<code>time t_chk[1:100]</code>	an array of 100 time variables
<code>reg [4:0] port_id[0:7]</code>	an array of 8, 5-bit port_id vectors

Arrays can be defined for *reg*, *integer*, *time* and vector register data types and do not typically appear in the signal declarations although they may sometimes be required to connect between embedded blocks.

Arrays of registers are typically used to represent memories. The following example shows an array of eight five-bit vectors connected to an embedded block:



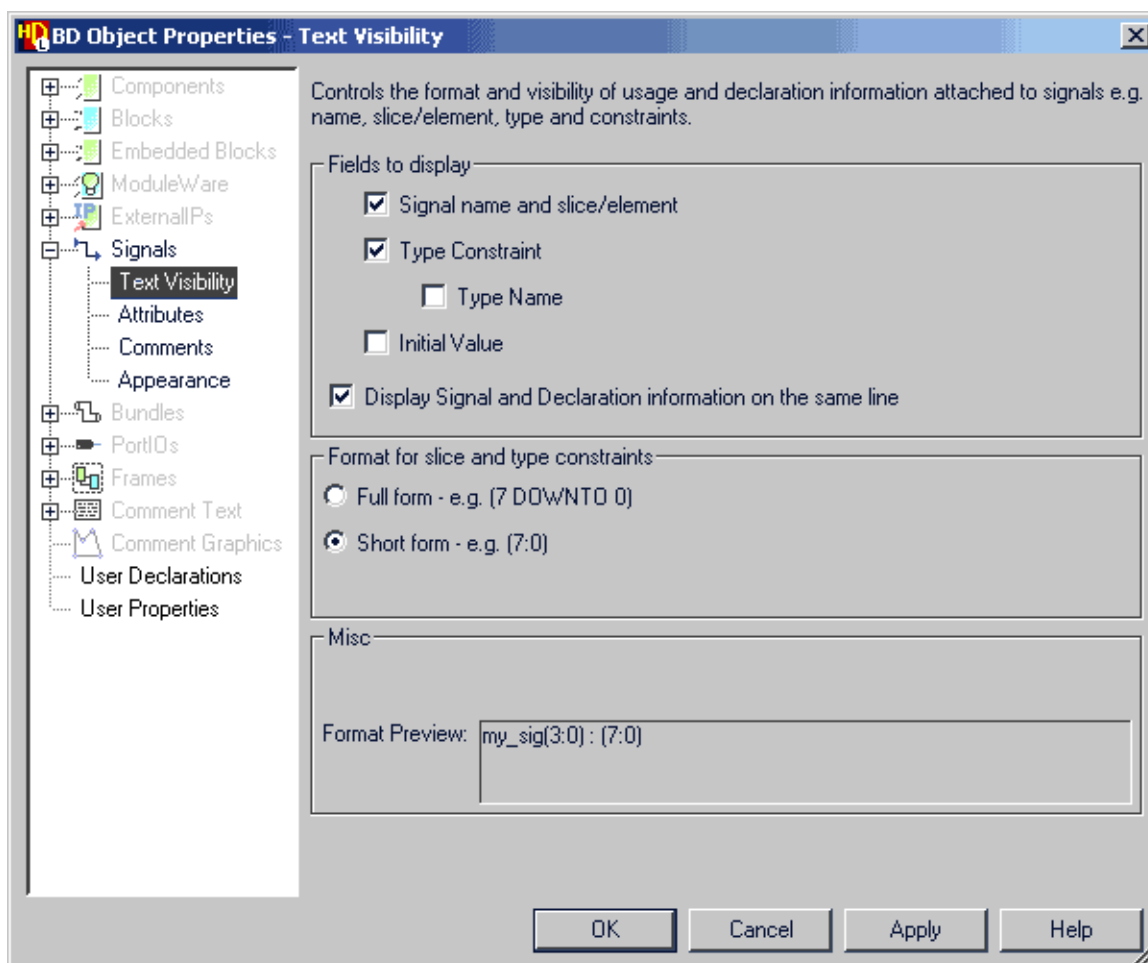
HDL2Graphics automatically adds the vector and array bounds to net declarations on a block diagram or IBD view when a vector with an array declaration and *reg* type is detected in the source Verilog. A vector bounds is recovered for an array of *integer* or *time*. If an array element is accessed in a Verilog instantiation, then a corresponding slice is added to the signal on the block diagram or IBD view.

You can also add the VHDL keywords *Bus* or *Register* to the net declaration by choosing bus or register kind (if not specified the default is none)

Editing Signal Text Visibility

The **Text Visibility** page for the Signal Object of the BD Object Properties dialog box allows you to modify the signal display properties.

Refer to [“Changing the Display of Signal Properties”](#) on page 208

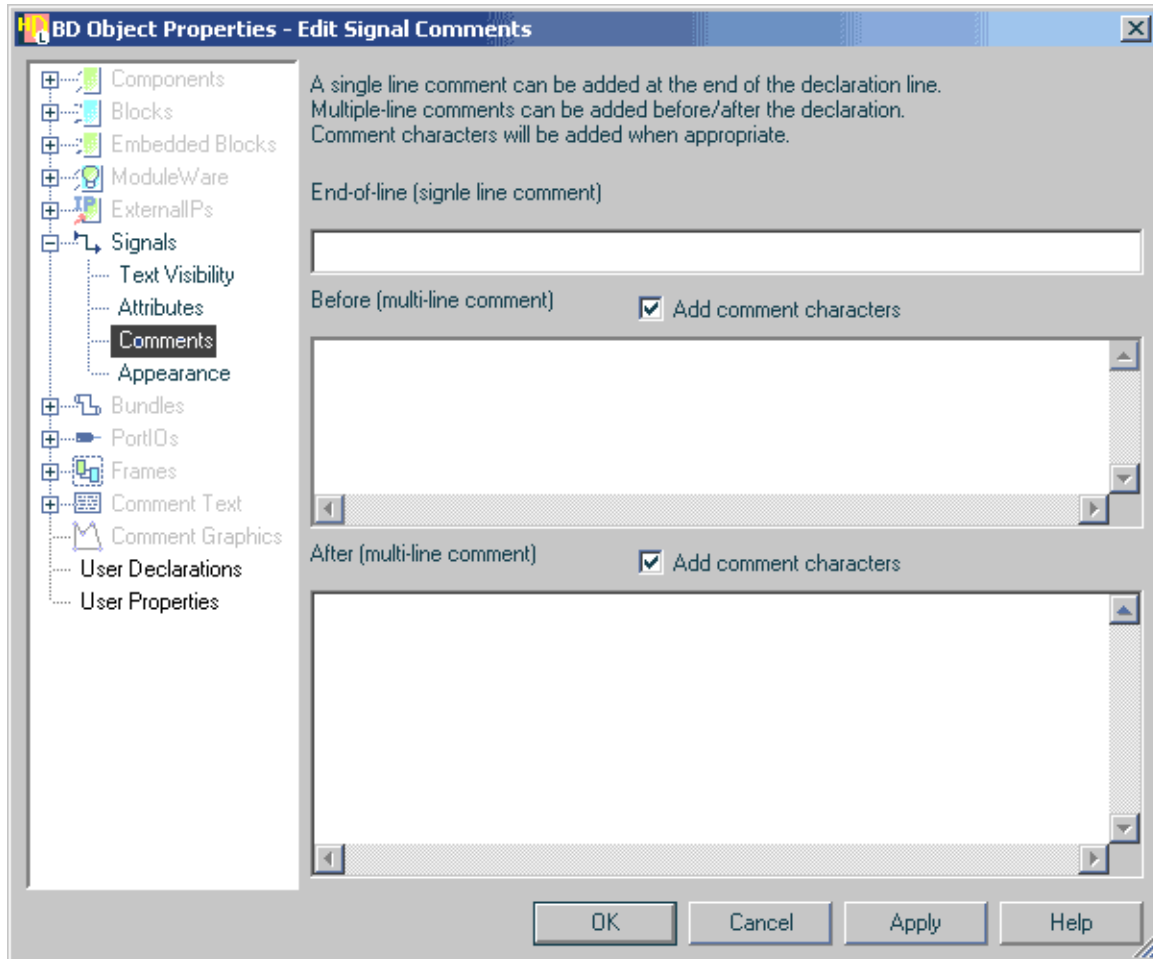


Setting Signal Attributes and Embedded Constraints

The **Attributes** page for the Signal Object of the BD Object Properties dialog box allows you to set attributes and embedded constraints for the selected signals. Refer to [“Setting Attributes and Embedded Constraints”](#) on page 165.

Editing Signal Comments

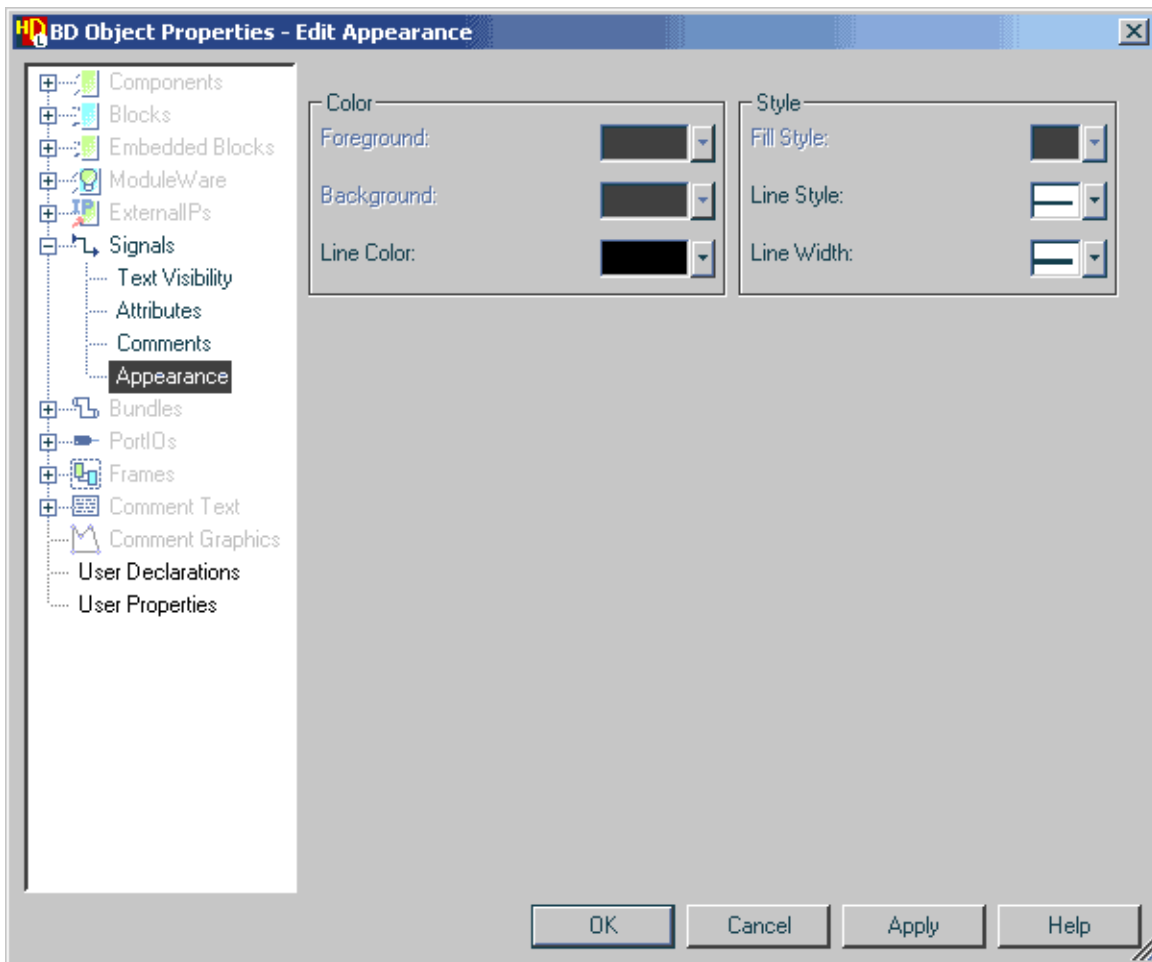
The Comments page for the Signal Object of the BD Object Properties dialog box allows you to add comments to a signal. Refer to [“Adding Comments to a Signal or Port Declaration”](#) on page 163.



Modifying Signal Appearance

The **Appearance** page for the Signal Object of the BD Object Properties dialog box allows you to set visual attributes for the selected bundle.

The attributes include the line color, line style and line width.



Editing Port IOs Properties

The **Port IO** object is selected if you display the Object properties dialog box when a Port is selected on a block diagram.

The Port IO page allows you to specify the port mode settings as In, Out, InOut or Buffer in VHDL designs and In, Out or InOut in verilog designs.

Editing Port IO Text Visibility

The **Text Visibility** page for the Port IO Object of the BD Object Properties dialog box allows you to modify the Port IO display properties.

Refer to [“Changing the Display of Port Properties”](#) on page 206.

Modifying Port IO Appearance

The **Appearance** page for the Port IO Object of the BD Object Properties dialog box allows you to set visual attributes for the selected Port.

The attributes include the foreground, background and line color, the line style, fill style, line width.

Editing Frame Properties

The **Frames** page is enabled if you display the Object properties dialog box when a frame is selected on a block diagram or an IBD view.

The Frames page of the BD Object Properties dialog box allows you to display and edit generate frames.

You can choose the frame style from a pulldown list that includes FOR, IF and ELSE when using Verilog or FOR, IF and BLOCK when using VHDL.

You can change the label, number and frame expression. When you are using VHDL, a Preview field in the dialog box shows the current frame expression. You can set the visibility of the frame title (label and expression) and the frame number.

When you are using VHDL, you can enter a Parameter and range for a FOR frame, a Condition for an IF frame or an optional Guard expression for a BLOCK frame.

When you are using Verilog, you can enter the Range for a repeating instance in a For frame or the Macroname for an IF frame

The syntax of the expression is checked on entry and the Generate Keyword is automatically included in VHDL FOR and IF expressions.

When the Declarations button is selected a free-format entry dialog box is displayed which allows you to specify local declarations which apply only within the frame. You can also choose whether these declarations are Visible on the diagram. The syntax of the declarations is checked on entry. For more information refer to [“Editing Generate Frame Properties”](#) on page 302.

Modifying Frame Appearance

The **Appearance** page for the Frame Object of the BD Object Properties dialog box allows you to set visual attributes for the selected frames.

The attributes include the line color, style and line width.

Editing Comment Text Properties

The **Comments** object is selected if you display the Object properties dialog box when a Comment is selected on a block diagram.

The **Comment Text** page allows you to modify the existing comment text and position.

You can also choose to resize the bounding box to fit the comment text.

Modifying Comment Text Appearance

The **Comment Text Appearance** page for the Comment Text Object of the BD Object Properties dialog box allows you to set visual attributes for the selected comment text

The attributes include the text color and font.

Modifying Comment Text Box Appearance

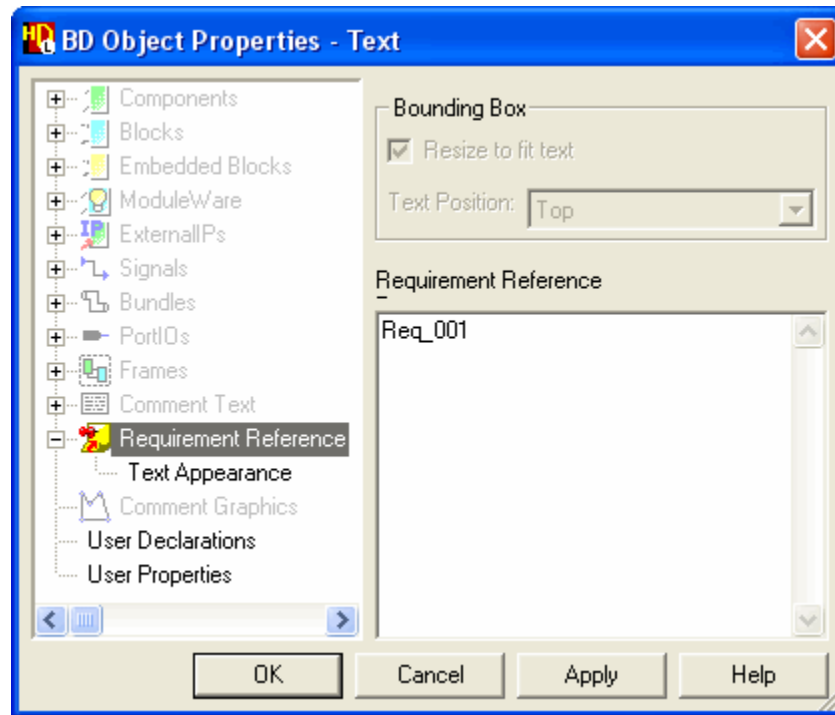
The **Comment Text Box Appearance** page for the Comment Text Box of the BD Object Properties dialog box allows you to set visual attributes for the selected comment text boxes.

The attributes include the foreground, background and line color, line style, fill style and line width.

Editing Requirement Reference Properties

The **Requirement Reference** object is selected if you display the Object Properties dialog box when a requirement reference object is selected on a block diagram.

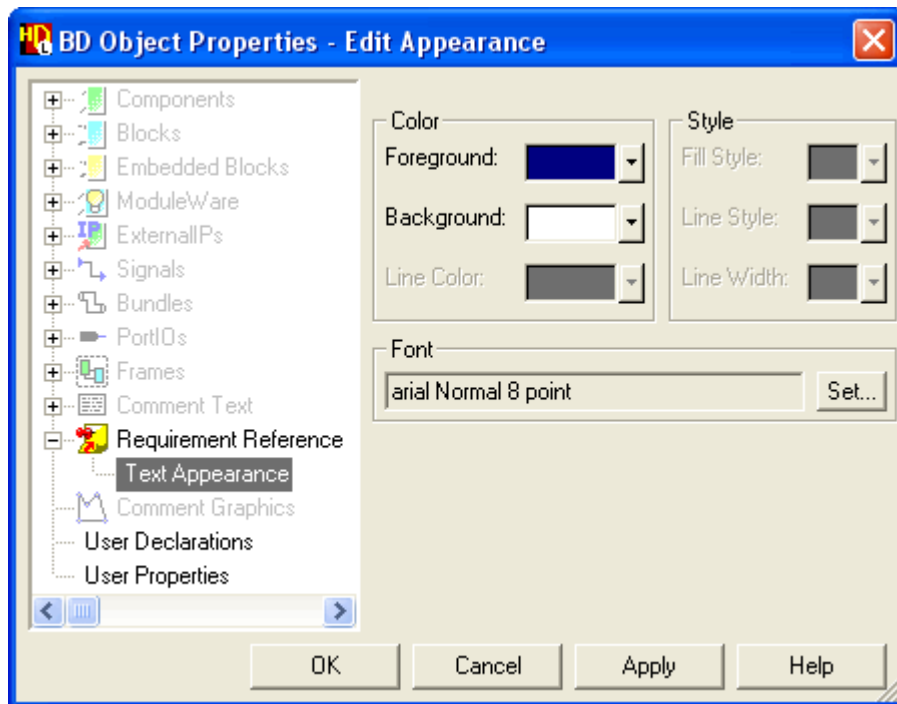
The **Requirement Reference** page allows you to modify the existing requirement reference object's text by typing in the Requirement Reference pane in the dialog box.



Modifying Requirement Reference Text Appearance

The **Requirement Reference Text Appearance** page for the Requirement Reference Object of the BD Object Properties dialog box allows you to set visual attributes for the selected Requirement Reference.

The attributes include the text color and font.



Editing Comment Graphics Properties

The **Comments Graphics** object is selected if you display the Object properties dialog box when a Comment Graphic is selected on a block diagram.

The **Comment Graphics** page allows you to set visual attributes for the selected comment graphics.

The attributes include the foreground, background and line color, line style, fill style and line width.

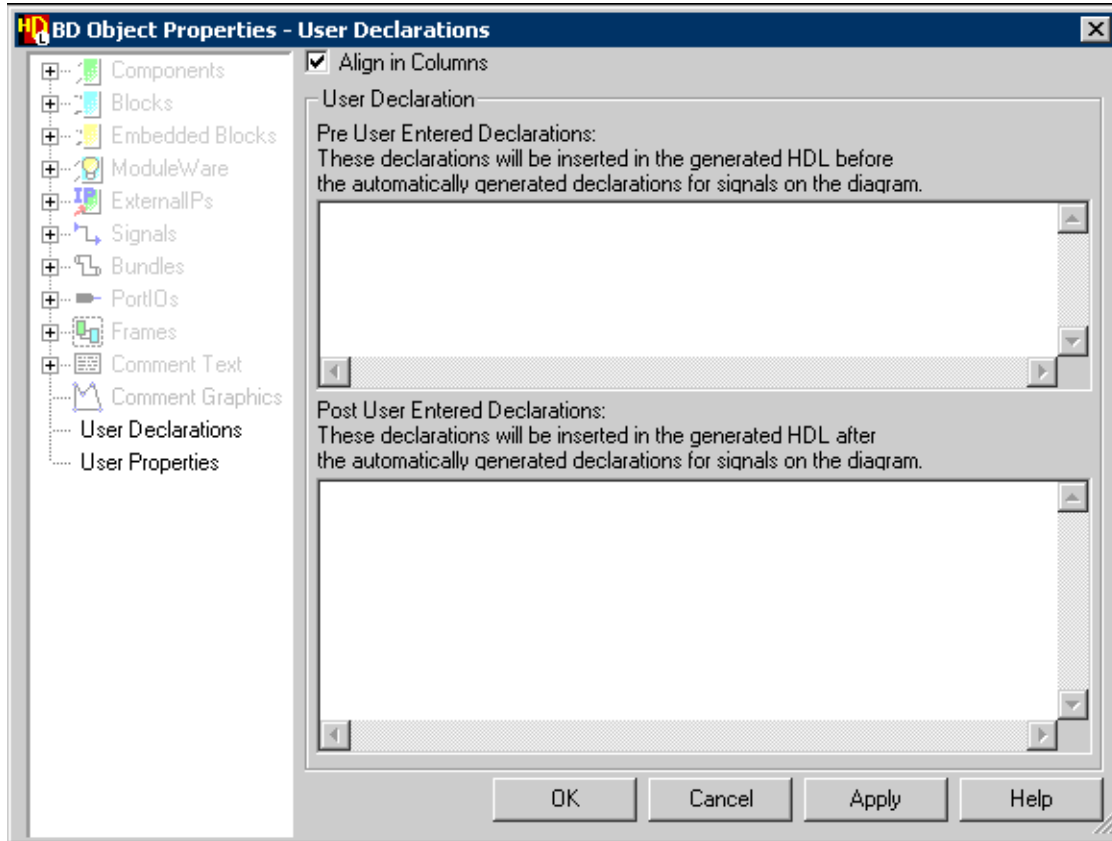
Note



The embedded VHDL style option to **Include View Name in Embedded Configurations** must be set if you want to instantiate a specified view.

Editing User Declarations

The **User Declaration** page of the BD Object Properties dialog box allows you to add or edit user-defined declarations for a block diagram or IBD view. You can enter free-format declarations before or after the signal declarations.



You can access the User Declarations dialog box directly in an IBD view by choosing **User Declarations** from the **Table** menu in an IBD view.

You can edit the declaration statements directly on a block diagram by clicking to select the text on the diagram and clicking again to edit the text.

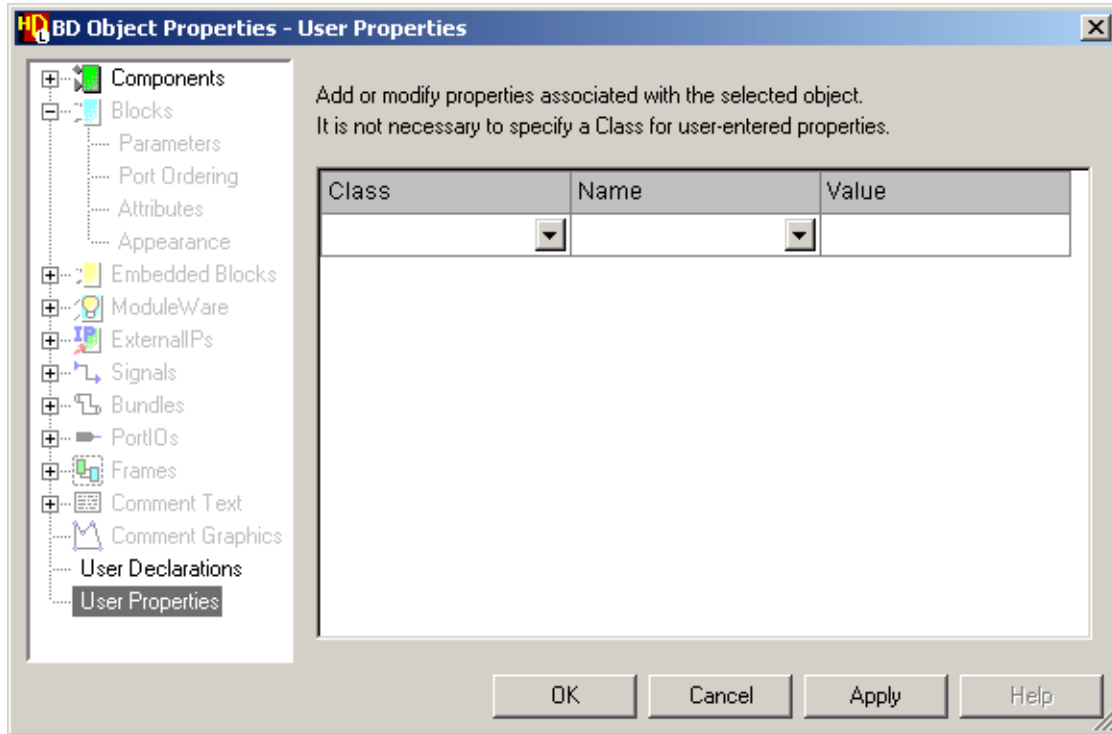
Pre user declarations can be referenced by signal declarations or by post user declarations. For example, you could use a pre user declaration for a bus width constant which is referenced by a graphically defined signal declaration.

Similarly a post user declaration can reference a signal declared in the graphically defined or pre user declarations.

The syntax is automatically checked when you confirm the dialog box. However, syntax checking can be disabled by unsetting a block diagram preference.

Editing User Properties

The **User Properties** page of the BD Object Properties dialog box allows you to define internal variables as a property of a selected design object. The defined Internal variables are described by Class and Name attributes to which values are then assigned for the selected design object.



Setting the Scope for Net Changes

When you edit a signal declaration on a block diagram or IBD view, you can choose whether the changes are applied only on the active view or are propagated to other design unit views in the hierarchy.

You can set the scope in a block diagram by choosing **Scope For Changes** from the popup menu or from the **Signals** cascade of the **Diagram** menu.

Note



You can also set the scope for changes in the **Signals** tab of the block diagram Object Properties dialog box.

You can choose **Joined Wires in this diagram** to apply changes to the explicitly joined net segments only (excluding any net segments connected by name), **Entire Net in diagram** to change all segments in the same net on the active diagram, or **Entire Net in hierarchy** to apply the net changes to all views in the hierarchy.

You can set the scope in an IBD view by choosing **Scope For Changes** from the popup menu or from the **Nets** cascade of the **Table** menu. For an IBD view, you can choose **Non Hierarchical** or **Hierarchical** scope.

The scope is saved as a preference and is used as the default until the next time a signal declaration is edited.

When you edit a net declaration and the scope for changes includes hierarchical views, a dialog box is displayed to control how changes are propagated to nets in the hierarchy.

Refer to [“Propagating Net Changes”](#) on page 166 for more information about propagating new signals and changes to the properties of a net.

Adding Comments to a Signal or Port Declaration

You can add comments to a signal or port declaration on a block diagram by choosing **Edit** from the **Comments** cascade for **Ports** or **Signals** in the **Diagram** menu or by choosing **Comments** from the popup menu when the net or its declaration is selected.

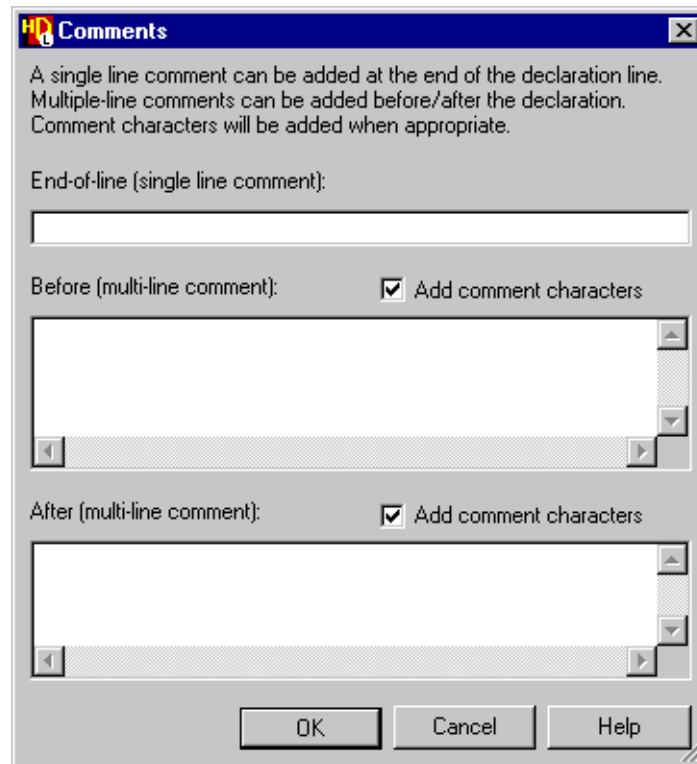
You can add comments to a signal declaration in an IBD view by choosing **Comments** from the **Nets** cascade in the **Table** menu or by choosing **Comments** from the **Add** or popup menu when one or more signal rows are selected in the table matrix.

A free-format entry Comments dialog box is displayed which allows you to add a single line comment at the end of the declaration or you can enter a multi-line comment to be included before or after the declaration.

Comment characters for the current hardware description language (VHDL or Verilog) are automatically inserted if the **Add comment characters** check box is set. When this option is unset, the comments must be valid HDL statements and are automatically syntax checked if checking is enabled.

The comments are displayed in the port or signal declarations list on a block diagram. If a declaration is deleted, the corresponding comments are also deleted.

You can control the display of comments on a block diagram by choosing **Show** or **Hide** from the **Comments** cascade of the popup menu (or **Ports** cascade of the **Diagram** menu) when the port or signal declaration is selected.



Although multi-line comments can be added to an IBD view using the dialog box, these comments are not displayed in the table. However, end-of-line comments can be edited directly in the Comment column for the port or signal declaration row.

Comments added to internal signals on a block diagram or IBD view are included in the generated HDL.

Comments added to port declarations are ignored when HDL is generated since the interface information is generated from the symbol. If you want to include comments in the generated HDL for interface ports, these should be added using the symbol or tabular IO editor.

Note

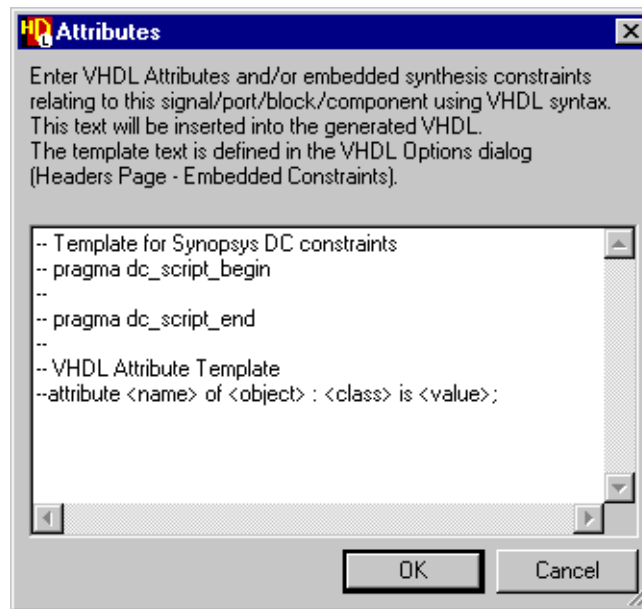
A special font is required to display Kanji characters in the dialog box. If Kanji characters are required in comment text, this font can be enabled by setting the **"HDS_KANJI_DIALOG"** environment variable.

Setting Attributes and Embedded Constraints

You can add embedded constraints or other attributes to a net or port declaration in a block diagram by choosing **Edit** from the **Attributes** cascade for **Ports** or **Signals** in the **Diagram** menu or by choosing **Attributes** from the popup menu when the net or its declaration is selected.

You can add attributes to a signal declaration in an IBD view by choosing **Attributes** from the **Nets** cascade in the **Table** menu or by choosing **Attributes** from the **Add** menu or popup when one or more signal rows are selected in the table matrix.

A free-format entry dialog box is displayed containing any existing attributes for the selected object. If no attributes have been set, the dialog box contains the template attributes and embedded constraints defined in your preferences.



You can add attributes to a port declaration in the symbol editor by choosing **Attributes** from the popup menu when the port or its declaration is selected.

You can add attributes to a port or locally declared signal declaration in a tabular IO view by choosing **Attributes** from the popup when one or more signal rows are selected in the table matrix.

You can also set attributes by using the **Block**, **Component**, **Signals**, **Bundles** or **Declarations** attributes pages of the block diagram or IBD view Object Properties dialog box.

The default template includes template begin and end pragmas which can be used to enclose embedded commands for the downstream tool. For example:

```
-- pragma dc_script_begin
-- set_max_area 2500.0
-- set_drive -rise 1 port_b
-- pragma dc_script_end
```

Note



Default embedded constraints can be set as VHDL or Verilog preferences from the **Headers** tab of the VHDL and Verilog Options dialog boxes.

The dialog box also contains a default template for in-line signal attributes. For example, having declared a signal *int_signal* you could add VHDL attributes such as:

```
attribute preserve_signal of int_signal:signal is TRUE ;
attribute modgen_sel of int_signal:signal is FAST ;
```

Attributes and embedded constraints are not normally shown on a block diagram. However, you can choose **Show** or **Hide** from the **Attributes** cascade of the popup menu (or **Ports** cascade of the **Diagram** menu) when a port or signal declaration is selected.

When shown, the attributes are displayed below the declaration of the corresponding port or signal in the Declarations list.

Note



You can change the default visibility by setting **Show Signal Attributes** in the **Miscellaneous** tab of the Block Diagram Preferences dialog box.

The contents of the edit box are associated with the selected signal declaration when you confirm the dialog box and will be included in the generated HDL for the view.

Attributes added to internal signals are included in the generated HDL. Although attributes can be added to a port declaration, these are ignored when HDL is generated since the interface information is generated from the symbol.

If you want to associate attributes to a port (for example, a pin number attribute) these should be added to the port declaration in the symbol or tabular IO editor.

Propagating Net Changes

By default, changes to nets are made non-hierarchically to the active view only.

However, you can set the scope for changes to include connected or hierarchical nets as described in [“Setting the Scope for Net Changes”](#) on page 162.

If a net property is changed when a hierarchical scope is set, the Net Propagation Options dialog box is displayed for you to choose how the changes are propagated to nets in other views.

Note

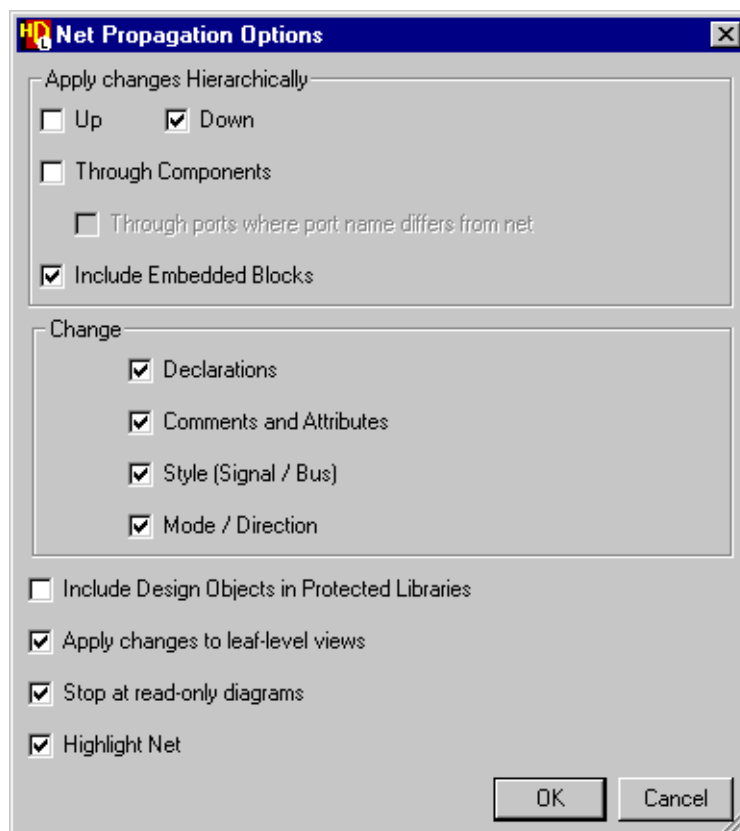
The dialog box is also displayed when you change the properties of a port in the symbol or tabular IO editor and the scope is set to apply the changes hierarchically.

All changes to the net or port properties (including any comments and attributes) can be propagated.

You can choose whether net changes are applied **Up** or **Down** the hierarchy and whether to propagate the changes **Through Components** or **Include Embedded Blocks**.

If you choose to propagate through components, an extra option controls whether the change is propagated **Through ports where port name differs from the net**.

If this option is set and you have write permission to any connected component, the port names in the component symbol are changed.



You can choose whether changes are made to **Declarations**, **Comments and Attributes**, net **Style** (represented as a signal or bus) or the **Mode** (in, out, bidirectional or buffer).

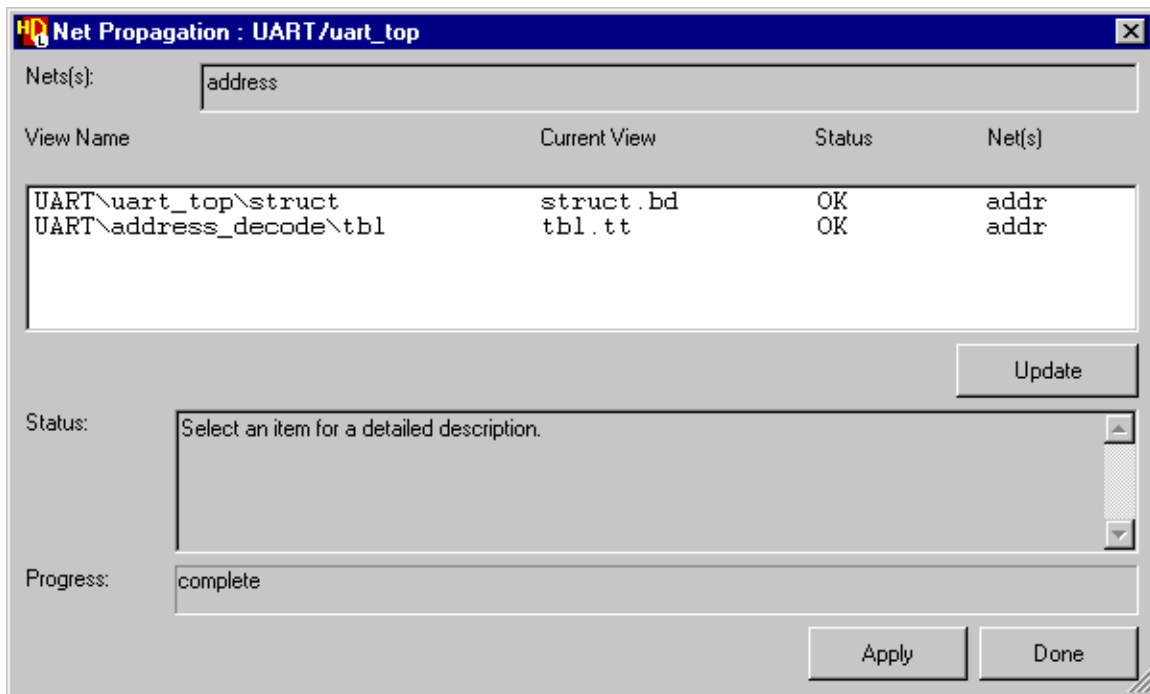
You can also choose to **Include design objects in Protected libraries**, **Apply changes to leaf-level views**, **Stop at read-only diagrams** or **Highlight the net**.

You cannot propagate up through components since the design traversal will stop when a symbol is reached.

You cannot propagate through a text view.

When you confirm your choices in the Net Propagation Options dialog box, the Net Propagation progress window is displayed which lists all the occurrences of the net in the specified hierarchy.

The progress is indicated in the dialog box and a complete indicator displayed when all views in the hierarchy have been traversed.



If an Error or Warning is encountered, these are indicated in the Status column and the full message appears in the Status box when the view name is selected. For example, a text or read-only view cannot be updated.

Note



Note that propagation will stop if an error is encountered to avoid creating an incorrect design.

You can display any view listed in the progress window by double-clicking on its name in the progress window.

You can use the **Update** button to change any of the net propagation options and update the preview window for changes made to any of the views in the hierarchy. For example, if you have made a read-only view available for edit.

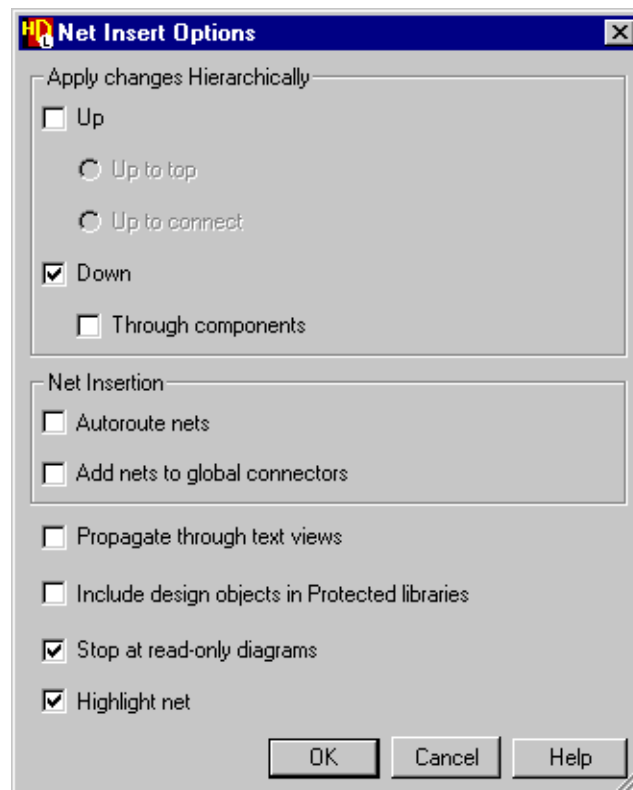
The changes are made when you confirm the preview window using the **Apply** button. The status column shows *Modified* for all views which have been changed.

Inserting and Removing Nets

You can propagate signals or buses on a block diagram or IBD view and ports on a symbol or tabular IO view to other views in the hierarchy by selecting the net and choosing **Single Level Up**, **Single Level Down** or **Hierarchical** from the **Insert Net** cascade in the popup menu.

The **Insert Net** cascade is also available in the **Signals** cascade of the **Diagram** menu in a block diagram, the **Ports** cascade of the **Diagram** menu in the symbol editor, the **Nets** cascade of the **Table** menu in an IBD view or the **Ports** cascade of the **Table** menu in a tabular IO view.

If you choose **Hierarchical**, the Net Insert Options dialog box is displayed.



If you choose the **Up** option in the dialog box and a net is connected to an external port, you can propagate the net up through the hierarchy to the top level symbol or until a connection can be made to an existing net with the same name and bounds.

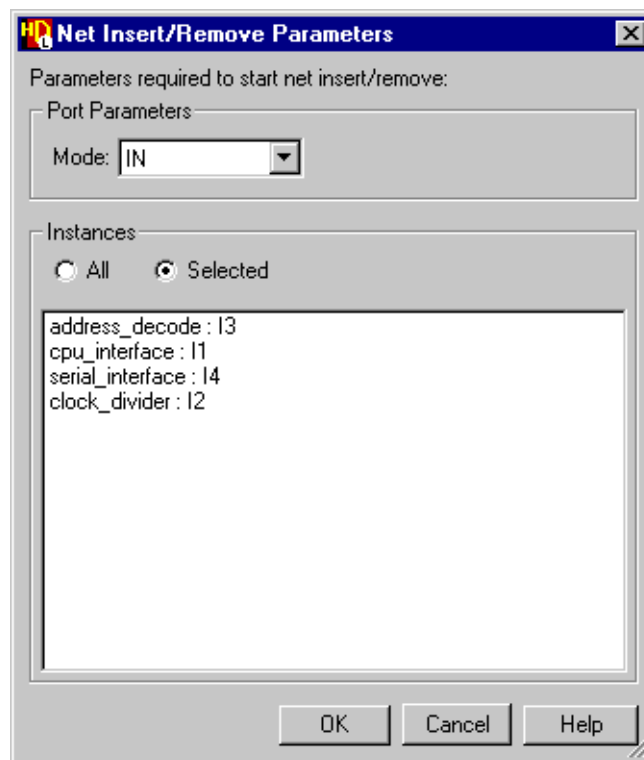
If you choose the **Down** option and a net is connected to an instance of a block or component, you can choose to propagate the net **Through Components** in the hierarchy.

You can also choose to **Autoroute nets** or to **Add nets to global connectors**. Global connectors can be added when you propagate down an input net and will be added to all blocks in the hierarchy.

If the global connectors and through components options are set, net stubs are added to each component in the hierarchy.

You can also choose whether to **Propagate through text views**, **Include design objects in Protected libraries**, **Stop at read-only diagrams** or **Highlight the net**.

If you select a net that is not connected to any instances on the active view, a Net Insert/Remove Parameters dialog box is displayed which allows you to choose the port mode and select one or more instances to traverse:

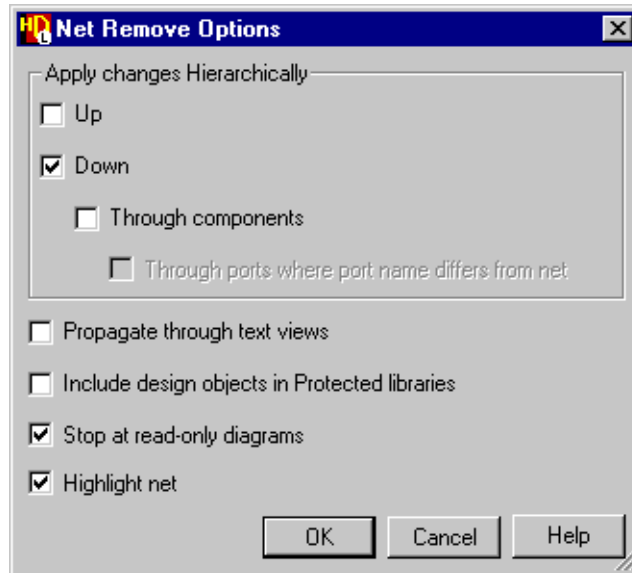


If there are more than one instances in the hierarchical view, you can choose whether the new net is connected to **All** instances or to **Selected** instances only.

You can remove signal or bus nets from related views in the hierarchy using the **Remove Net** cascade in the popup menu.

The **Remove Net** cascade is also available in the **Signals** cascade of the **Diagram** menu in a block diagram, the **Ports** cascade of the **Diagram** menu in the symbol editor, the **Nets** cascade of the **Table** menu in an IBD view or the **Ports** cascade of the **Table** menu in a tabular IO view.

If you choose **Hierarchical**, the Net Remove Options dialog box provides options to remove nets **Up** or **Down** and **Through Components** in the hierarchy.



When you remove nets through components, you can choose whether the change is propagated **Through ports where port name differs from net**. If this option is set and you have write permission to the component symbol, the ports are removed from the component.

The Net Remove options dialog boxes also allows you to **Propagate through text views**, **Include design objects in Protected libraries**, **Stop at read-only diagrams** or **Highlight the net**.

When you choose one of the single level options (or confirm the hierarchical options), the Net Propagation progress window is displayed which lists all the occurrences of the net in the specified hierarchy.

You can use the progress window to display views, update the net propagation options and apply changes in the same way as described in [“Propagating Net Changes”](#) on page 166.

Ordering Port and Signal Declarations

The declarations of signals and external ports are normally ordered automatically by mode (in, out, inout or buffer) and alphanumeric name as they are added in the block diagram or IBD view. However, you can enable manual ordering by choosing **Manual** from the popup or **Diagram** menu.

Note



This command is available from the **Ordering** cascade of the popup menu or **Ports** and **Signals** cascades of the **Diagram** menu in a block diagram or directly from the popup menu for the Order column in an IBD view.

When manual ordering is enabled in the block diagram, you can re-arrange the port or signal declarations by dragging one or more declarations with the **Left** mouse button. Any number of adjacent declarations can be moved in this way but you cannot mix port or signal declarations with user declarations (or port and signal declaration on a block diagram). The new order is preserved on the diagram and in the generated HDL.

When manual ordering is enabled in the IBD view, the Order column displays numbers which indicate the order used for the port and signal declarations in the HDL code. You can choose **Show Order** to re-order the table rows in declaration order.

You can re-order port rows in the IBD view by selecting an entire row and dragging it with the **Right** mouse button. However, you cannot update the declaration order from the IBD view.

Manual ordering is automatically set if you synchronize component interface ports with a text view to preserve the port ordering specified in the text view.

Manual port and signal ordering is normally used when diagrams are created by *HDL2Graphics* in order to preserve the order (and any in-line comments) from the source HDL code. If you choose **Automatic**, the original ordering is discarded and the declarations sorted by mode and alphanumeric name.

You can modify the port ordering for a block instance by editing the object properties for the block as described in “[Editing Block Properties](#)” on page 139.

Adding or Removing Design Hierarchy

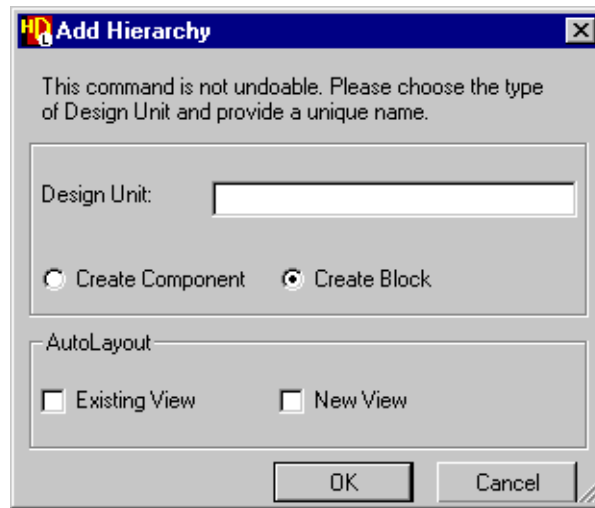
You can re-level a block diagram or IBD view by choosing the **Add Hierarchy** or **Remove Hierarchy** command from the **Re-level** cascade in the block diagram **Diagram** menu, IBD view **Table** menu or the popup menu in either view.

Adding hierarchy replaces the selected objects by a new block or component and moves the selected objects into the new design unit view.

When this command is used in a block diagram, a child block diagram view is created and the new block or component is attached to the cursor so that you can place it on the parent diagram. Any signals connected to the new object are automatically re-routed.

When used in an IBD view, a child IBD view is created and the objects replaced by a single new instance on the parent table.

The Add Hierarchy dialog box is displayed for you to specify the name of the new design unit containing the selected objects and whether to create it as a block or component.



You can also choose to automatically update the block diagram layout for the existing view and for the new hierarchy view. When these options are not set, the layout is preserved.

Any package references or compiler directives are copied to the new design unit. Global connectors are automatically included on the new view if any blocks or embedded blocks are selected.

If any of the selected objects are connected using a bundle and you choose to create a component, a port map frame is automatically added.

Removing hierarchy deletes the selected block or component instance and replaces it by the objects in the child hierarchical view.

If the parent and child views are block diagrams, the relative placement of the new objects is preserved and a ghosted view is attached to the cursor so that you can place it on the parent diagram.

If the parent view is a block diagram and the child view an IBD view, the new objects are automatically connected and the routing is completed when you place the ghosted objects on the diagram.

If the child view included other hierarchical views, their hierarchy is retained but can be removed by another re-level operation.

You are prompted to confirm a list of design objects which will be deleted by the re-level operation. For example:



If a component is deleted, you may need to manually update any other views that referenced the deleted component.

If there are global connectors in the child view, these become global connectors in the merged parent view and are connected to all blocks and embedded blocks in the parent view.

The package references or compiler directives for the child view are merged with those defined for the parent view.

If removing hierarchy replaces a component with port map frame expressions, these mapping expressions are preserved as an embedded HDL text view.

An error message is issued if you attempt to remove hierarchy for an instance which is not described by a block diagram or IBD view or which has a different language.

Generics and Parameters

Generics and Parameters are constants that you declare graphically and give a default value; they are later included in the generated HDL code. *VHDL generics* are a VHDL feature used to pass information to an instance of an entity. *Verilog parameters* can be used in a similar way when you are using the Verilog language.

Using generics and parameters involve the following two main procedures:

1. Generics and Parameters are primarily declared for the *components* symbols and *blocks*. Refer to “[Defining Generics and Parameters](#)” on page 184 for details.

2. Subsequently, the declared generics and parameters are edited on the level of the individual instances of the component or block. Refer to [“Editing Generics and Parameters for Instances”](#) on page 186.

Note

Note that you can also declare generics/parameters through the default view of the component and it will synchronize with the symbol. See [“Generics and Parameters Synchronization”](#) on page 192 for information.

Typical uses for VHDL generics and Verilog parameters include:

- Substitution in place of an integer value for the least or most significant field of a signal bounds specification.
- Declaration of a delay that can be varied on each instance of a re-usable component.

The following example illustrates the use of VHDL generics:

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

entity xorx is
  generic(width : integer; delay : time);
  port(x1, x2 : in std_logic_vector( width DOWNT0 0 ));
  xout : out std_logic_vector( width DOWNT0 0 ));
end xorx;
architecture generic_xorx of xorx is
  begin
    xout <= xin1 xor xin2 after delay;
end generic_xorx;

-- Test bench for instance generic xor
library ieee ;
use ieee.std_logic_1164.all;

entity testbench is
  generic(tb_width : integer := 4; tb_delay : time := 5 ns);
  port(ina, inb : in std_logic_vector(tb_width DOWNT0 0));
  outa : out std_logic_vector(tb_width DOWNT0 0));
end testbench;

architecture tb_arch of testbench is
  component xorx
    generic(width : INTEGER;
      delay : TIME );
    port(x1, x2 : in std_logic_vector(width DOWNT0 0));
    xout : out std_logic_vector(width DOWNT0 0));
  end component;
  begin
    U1 : xorx generic map ( tb_width, tb_delay)
      port map (ina, inb, out);
  end tb_arch;
```

The following example illustrates the use of Verilog parameters:

```

module xorx(xout, xin1, xin2);
    parameter width = 4,
              delay = 10;
    output [1:width] xout;
    input [1:width] xin1, xin2;
    assign #delay xout = xin1 ^ xin2;
endmodule

// Testbench for instance xor
module testbench;
    parameter tb_width = 4,
              tb_delay = 5;
    wire [1:tb_width] out1;
    reg [1:tb_width] in1, in2;
    xorx #(tb_width, tb_delay) U1(out1, in1, in2);
endmodule

```

Generics and Parameters Tables

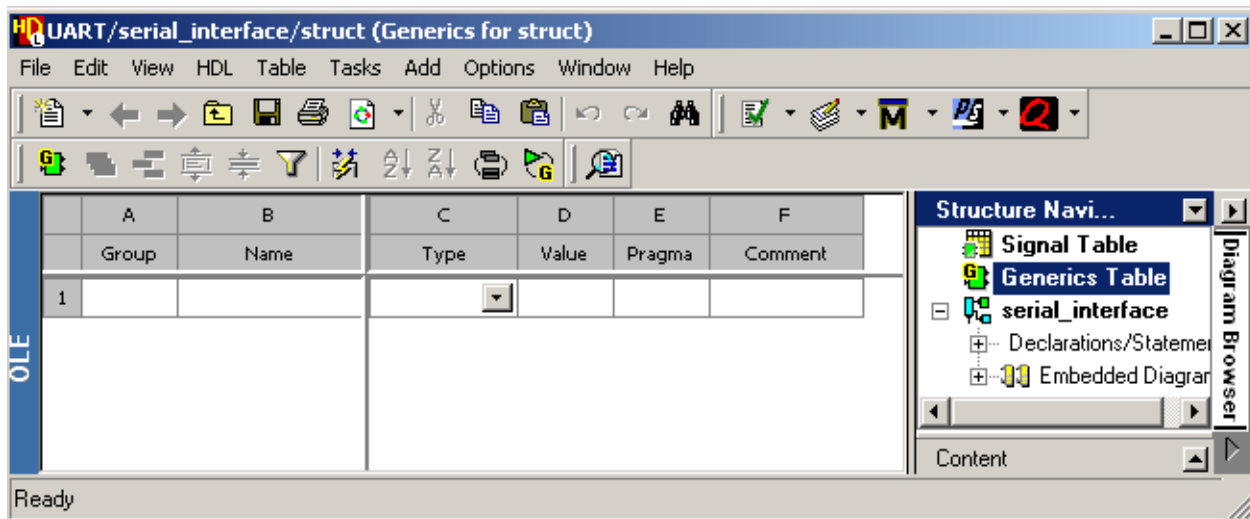
The Generics Table and Parameters Table allow you to declare VHDL generics and Verilog parameters respectively. The Generics Table is available if you are designing using VHDL and the Parameters Table is available if you are using Verilog.

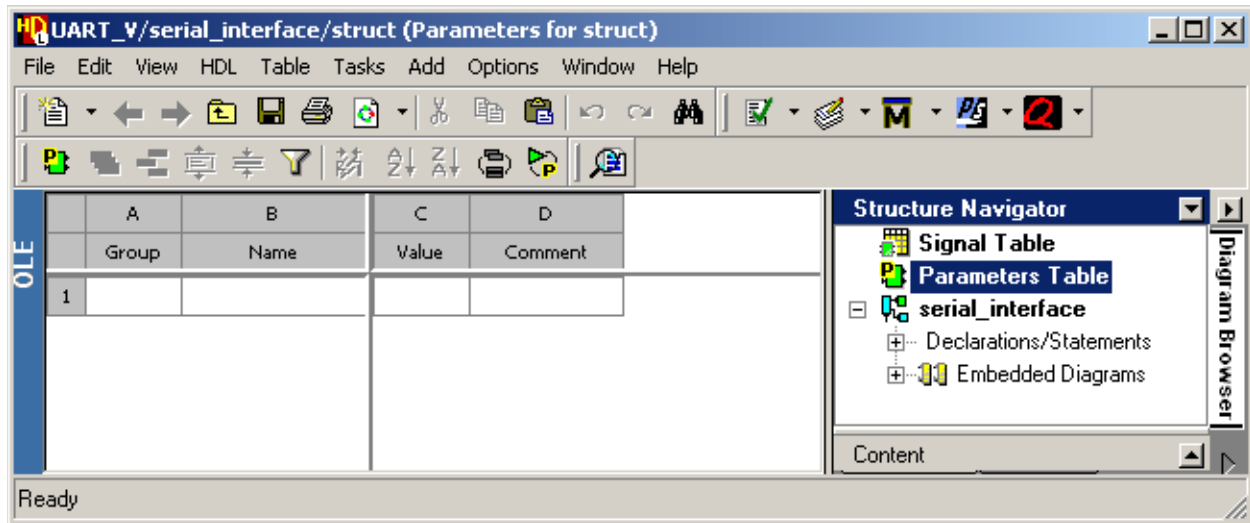
Accessing the Generics or Parameters Table

The Generics and Parameters Tables are found within any view window; for components, they can be mainly accessed through the symbol or the view. To open the Generics or Parameters Table, do one of the following:

- In the Structure Navigator pane of the Diagram Browser, click **Generics Table** to declare VHDL generics or click **Parameters Table** to declare Verilog parameters.
- From the **Diagram** menu, select **Generics** to declare VHDL generics or **Parameters** to declare Verilog parameters.

Generics Table:



Parameters Table:**Generics Table Controls**

You can manage the Generics Table using the generics toolbar which consists of the following buttons.

Table 3-3. Generics Toolbar

Button	Description
	Add Generic
	Group selected rows
	Ungroup selected rows
	Expand All Groups
	Collapse All Groups
	Toggle Filter
	Autofit column size to contents
	Sort column in ascending order
	Sort column in descending order
	Toggle show grouped rows
	Toggle between Ports/Generics table

To declare VHDL generics through the Generics Table, you have to enter the following information:












Table 3-4. Generics Table Content

Column	Description
Group	Specify a group name to categorize the VHDL generic if necessary.
Name	Specify the name of the VHDL generic.
Type	Choose a the type of the VHDL generic.
Value	Specify the default value of the generic.
Pragma	Choose whether to enclose the VHDL generic in pragmas or not.
Comment	Enter a general comment if necessary.

Parameters Table Controls

You can manage the Parameters Table using the parameters toolbar which consists of the following buttons.

Table 3-5. Parameters Toolbar

Button	Description
	Add Parameter
	Group selected rows
	Ungroup selected rows
	Expand All Groups
	Collapse All Groups
	Toggle Filter
	Autofit column size to contents
	Sort column in ascending order
	Sort column in descending order
	Toggle show grouped rows
	Toggle between Ports/Parameters table

To declare Verilog parameters through the Parameters Table, you have to enter the following information:

Table 3-6. Parameters Table Content

Column	Description
Group	Specify a group name to categorize the Verilog parameter if necessary.

Table 3-6. Parameters Table Content (cont.)

Column	Description
Name	Specify the name of the Verilog parameter.
Value	Specify the default value of the parameter.
Comment	Enter a comment if necessary.

Using the Generics and Parameters Table

This section describes the different operations that can be performed in the Generics/Parameters Table.

Add Generics/Parameters

To declare new generics or parameters, do the following:

1. Add the generic or parameter through one of these methods:
 - Click **Add Generic** in the toolbar if you are using VHDL or **Add Parameter** if you are using Verilog.
 - Choose **Add > Generic** in case of VHDL or **Add > Parameter** in case of Verilog.
 - Choose **Generic** or **Parameter** from the **Add** cascade of the popup menu.

A row is added with a default Name and Type in case of VHDL generics, or with a default Name only in case of Verilog parameters.

2. In the added row, insert the Name and Value of the generic/parameter, in addition to a Type in case of the generic.
3. Click **Save**.

Another method to add a generic/parameter is by typing the information directly in an empty row.

Grouping Generics/Parameters

Grouping enables you to organize the content of the generics/parameters table. You can specify a group for each generic/parameter declaration, and then different groups can be assembled and individually collapsed or expanded.

To group generics/parameters, select one row or more and then do one of the following:

- Click **Group** in the toolbar.
- Choose **Add > Group**.
- Choose **Table > Group > Group**.

- Choose **Group** from the popup menu.

The selected rows are added to a new group with the default name *GroupN* (where *N* is a number automatically incremented if it already exists).

- You can also enter a name directly in the Group column for the generic/parameter.

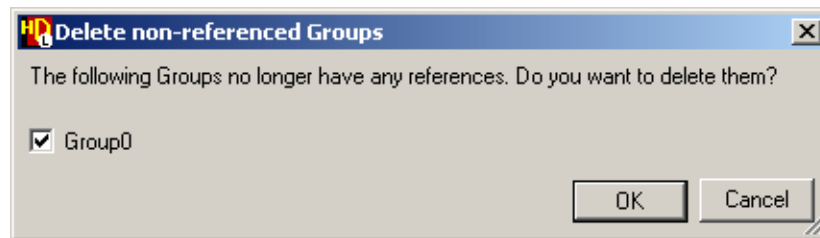
Note

You can choose from a dropdown list of existing groups. If you type a partial string that matches the name of an existing group, the name is automatically completed.

To remove a group for a generic or parameter, select the relevant row (or rows) and then do one of the following:

- Click **UnGroup** in the toolbar.
- Choose **Table > Group > UnGroup**.
- Choose **UnGroup** from the popup menu.
- You can also delete the name from the Group cell.

If you rename or remove an existing group cell and the group is no longer referenced, you are prompted to delete the old group name as shown in the figure below.



To assemble similar groups as collapsible and expandable groups, do one of the following:

- Click **Toggle Show Grouped** in the toolbar.
- Choose **Table > Show Grouped**.

By that, you have moved from the flat mode to the group mode. Rows in the same group are shown as a single (but expandable) group in hierarchy mode.

	A	B	C	D	E	F
		Name	Type	Value	Pragma	Comment
1	-	Integer				
2		width	integer ▾	15	NO	
3		delay	integer ▾	5	NO	
4	+	String				
5			▾			

To expand all groups and display their rows, do one of the following:

- Click **Expand All Groups** in the toolbar.
- Choose **Table > Group > Expand All Groups**.
- Click on the plus icon in the group row. This method enables you to expand an individual group.

On the other hand, to collapse all groups and hide their rows, do one of the following:

- Click **Collapse All Groups** in the toolbar.
- Choose **Table > Group > Collapse All Groups**.
- Click on the minus icon in the group row. This method enables you to collapse an individual group.

Note

When grouped mode is set, you can enter a comment in the group row.

Filtering Generics/Parameters

You can filter the content of any column, that is display specific data, by doing the following:

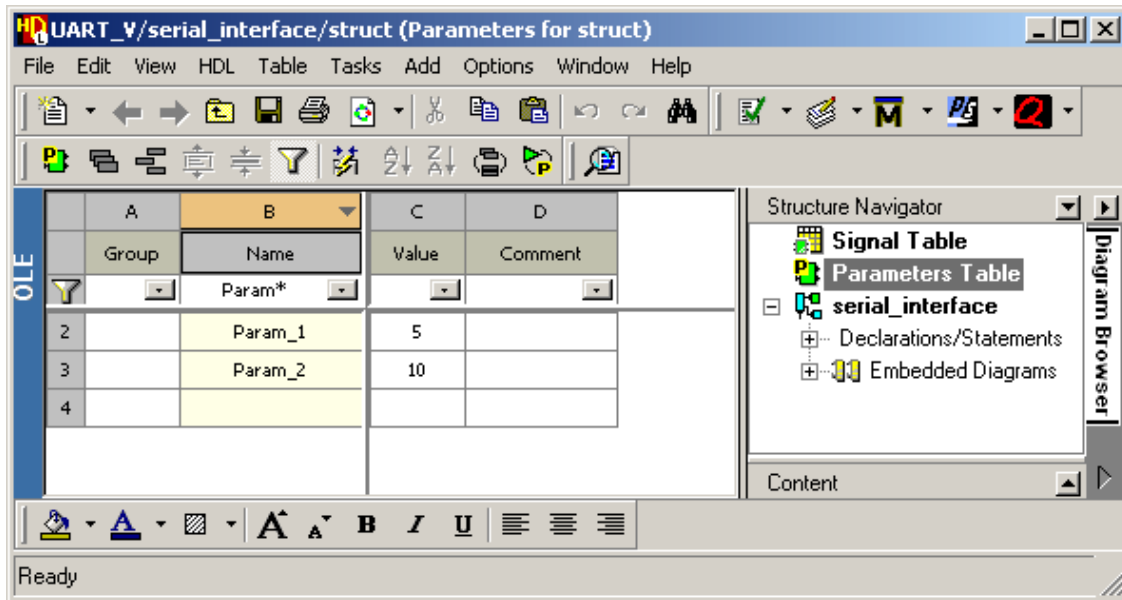
1. Turn on the filtering feature by doing one of the below steps:

- Click **Toggle Filter** in the toolbar.
- Set the **Filter** option in the **Table** menu.
- Choose **Filter** from the popup menu.

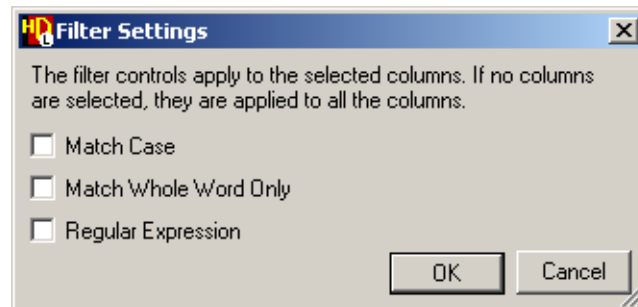
Consequently, a drop-down menu is displayed in each column through an additional filter row.

2. Through the drop-down menu of the column you want to filter, select the filter value. For example, if you choose *integer* in the Type column for a Verilog view, only the parameters of type integer are displayed.

You can also enter a simple match string in the filter cell to display only matching generics/parameters. For example, you can enter *param** in the Name column to display only parameters starting with the characters *param* as shown in the below figure.



You can apply filters to more than one column or set options to match case, match whole words or use regular expressions by choosing **Filter Settings** from the **Table** or popup menu to display the Filter Settings dialog box:



The filter settings are applied to the currently selected columns or to all columns if none is selected.

Sorting Generics/Parameters

You can arrange the rows in a selected column in ascending or descending alphanumeric order by either:

- Using the **Sort in ascending order** or **Sort in descending order** button in the toolbar.
- Choosing **Table > Columns > Sort Ascending** or **Table > Columns > Sort Descending**.
- Choosing **Sort Ascending** or **Sort Descending** from the selected column's popup menu.

You can also sort by clicking the triangular icon on the right side of the column header cell as shown in the following figure.

	A	B ▲	C	D
	Group	Name	Value	Comment
1		Param_2	10	
2		Param_1	5	
3				

Hiding Columns

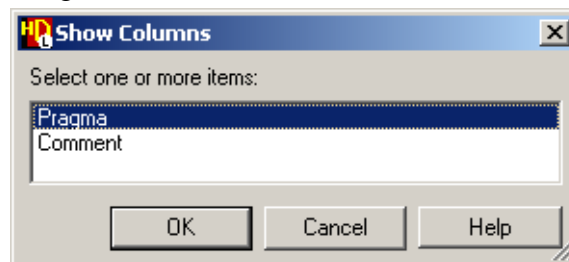
To hide columns, select one column or more and then do one of the following:

- Choose **Table > Columns > Hide Column**.
- Choose **Hide Column** from the popup menu.

To show the hidden columns, do one of the following:

- Choose **Table > Columns > Show Columns**.
- Choose **Show Columns** from the popup menu.

Consequently, a dialog box is displayed listing the hidden column (or columns). Select which columns you need to show again in the table and click **OK**.



Resizing Columns

To resize columns to the width of the text contained in the selected cell (or cells), do one of the following:

- Click **Autofit** in the toolbar.
- Choose **Table > Autofit**.
- Choose **Autofit** from the popup menu.

Note that if you have not selected any cells, then all the columns in the table are resized.

Note



You can also resize a column by dragging its vertical borders with your left mouse button and adjusting its width as required.

Related Topics

- [Defining Generics and Parameters](#)
- [Editing Generics and Parameters for Instances](#)

Defining Generics and Parameters

As mentioned earlier, VHDL generics can be declared for both components and blocks if you are using the VHDL language; the same applies to Verilog parameters if you are using the Verilog language.

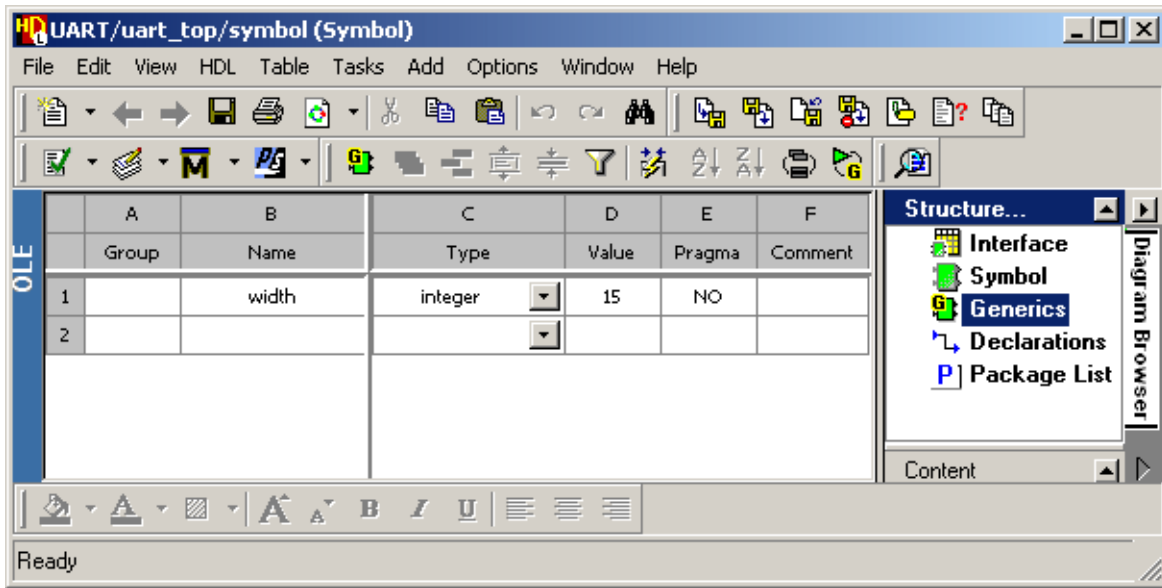
Defining Generics for Components and Blocks

Defining generics for components and blocks is done through the Generics Table which is a tabular view similar to the Signals Table; refer to “[Generics and Parameters Tables](#)” on page 176 for information. In case of components, the declaration of VHDL generics takes place on the level of the *symbol* which defines the interface to the *component*.

Follow this procedure to declare generic values whether for component symbols or for blocks.

Procedure

1. Open the symbol of the VHDL component for which you need to declare generic values. In case you are declaring generic values for blocks, open the block view.
2. In the Structure Navigator, click on **Generics**; the Generics table is displayed.
3. Click **Add Generic** in the toolbar; note that a row is added in the Generics table with a default Name and Type. Refer to “[Generics Table Controls](#)” on page 177 for information on the generics toolbar.
4. In the added row, insert the generic’s Name, Type and Value. Use the Group column to categorize your generics if necessary.



- Repeat the above steps to add more generics for the component or block as required.
- Click **Save**.

By that, you have declared generic values for the component or block. You can now use these generic values with individual instances; refer to [“Editing VHDL Generic Values for Instances”](#) on page 187.

Related Topics

- [Generics and Parameters Tables](#)
- [Editing VHDL Generic Values for Instances](#)

Defining Parameters for Components and Blocks

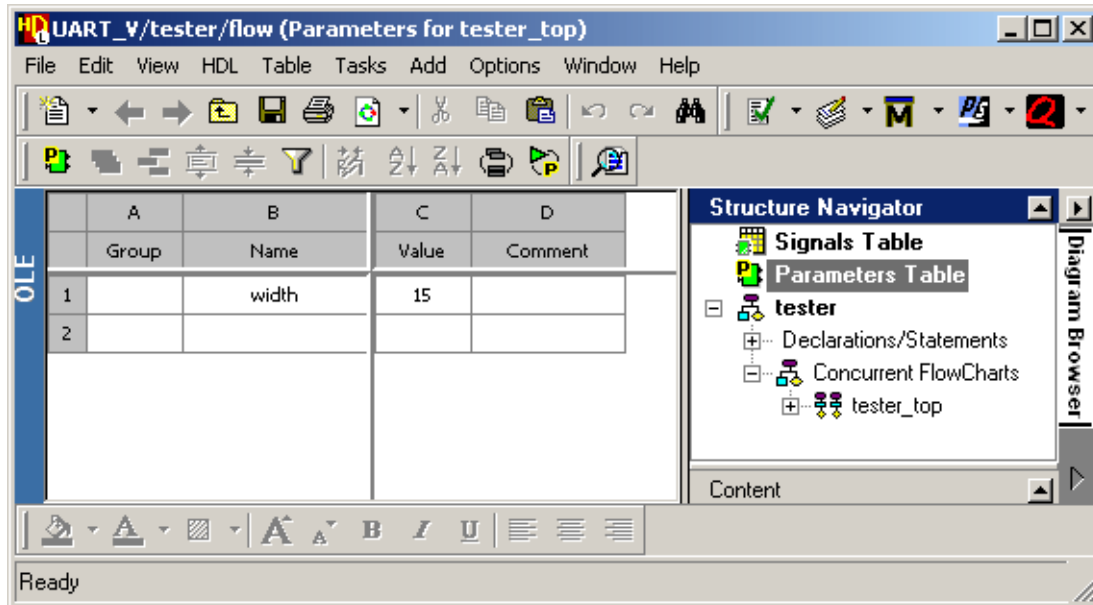
As the case is with VHDL generics, defining Verilog parameters for components and blocks is done through the Parameters Table which is a tabular view similar to the Signals Table; refer to [“Generics and Parameters Tables”](#) on page 176 for information. Note that in case of components, the declaration of Verilog parameters takes place on the level of the *symbol* which defines the interface to the *component*.

Follow this procedure to declare parameters whether for component symbols or for blocks.

Procedure

- Open the symbol of the Verilog component for which you need to declare parameters. In case you are declaring parameters for blocks, open the block view.
- In the Structure Navigator, click on **Parameters**; the parameters table is displayed.

3. Click **Add Parameter** in the toolbar; note that a row is added in the Parameters table with a default Name. Refer to [“Parameters Table Controls”](#) on page 178 for information on the parameters toolbar.
4. In the added row, insert the parameter’s Name and Value. Use the Group column to categorize your parameters if necessary.



5. Repeat the above steps to add more parameters for the component or block as required.
6. Click **Save**.

By that, you have declared parameters for the component or block. You can now use these parameters with individual instances; refer to [“Editing Verilog Parameter Values for Instances”](#) on page 189.

Note



It is worth mentioning that another method for defining verilog Parameters is through User Declarations. For information, refer to [“Editing User Declarations”](#) on page 161.

Related Topics

- [Generics and Parameters Tables](#)
- [Editing Verilog Parameter Values for Instances](#)

Editing Generics and Parameters for Instances

Having defined generics/parameters on the component or block level, you can now override these declarations for the instance through editing its Object Properties.

Editing VHDL Generic Values for Instances

Follow this procedure to edit VHDL generic declarations on the level of instances.

Prerequisites

- You must have the VHDL generic declarations preset for the component or block; if not, refer to [“Defining Generics for Components and Blocks”](#) on page 184.

Procedure

1. Open the top-level design of the component instance or block; that is, the component or block for which you previously declared VHDL generics. Refer to [“Defining Generics for Components and Blocks”](#) on page 184 for information on declaring VHDL generics.

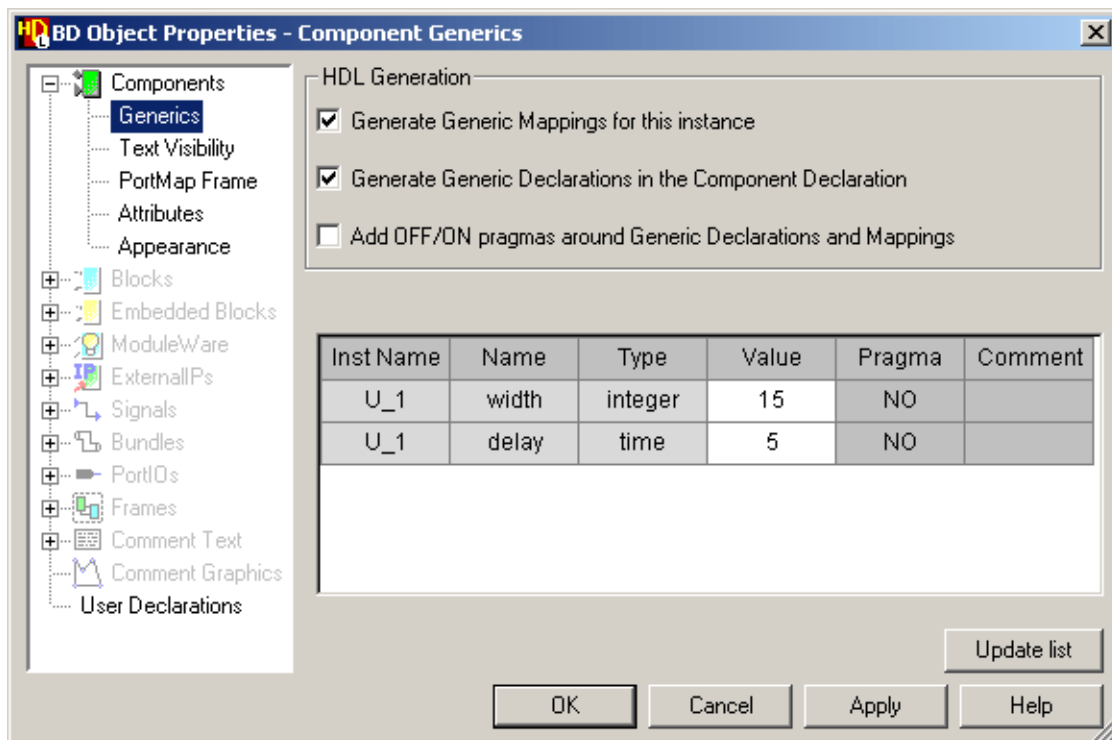
For example, if you have defined generic values for the symbol of the *uart_top* component, open the top-level *uart_tb* where an instance of *uart_top* is available with the name *U_1*.

2. Right-click on the instance and choose **Object Properties** from the popup menu.

Following the example in step 1, right-click on the *U_1* instance in the top-level *uart_tb*.

3. Open the **Generics** page and then click **Update List**.

The generics currently declared for the symbol of the component or for the block are consequently listed in the table as object properties of the instance as shown in the below figure.



Note that any changes made to the generic declarations of the component symbol or block are reflected on the Object Properties of the instance through clicking the **Update List** button. Any declarations that are no longer defined in the symbol or block are removed from the list and any new declarations are added. However, values that are already set on the instance for existing declarations are preserved.

Note



The **Generics** page of the Object Properties dialog box is available if the language of the selected instance is VHDL or the **Parameters** page if it is a Verilog instance.

4. Use the Value column to override the default declarations of the generics with mapped values. Type a new value or edit the existing value by double-clicking or using the **F2** key.

It is worth noting that this table is entirely read-only except for the Value column, that is, you cannot add generics through the Object Properties dialog box.

5. Set the following HDL Generation options as required:
 - **Generate Generic Mappings for this instance** — Setting this option leads to including the generic mappings in the generated HDL (that is, the mapped values inserted in the dialog box to override the default generic declarations).

For example, you can unset this option, when instantiating components (such as VITAL models) which have many generics but normally use the default values and the generics need to be omitted for efficient synthesis.

- **Generate Generic Declarations in the Component Declaration** — Setting this option leads to including the generic declarations in the generated component declarations.
 - **Add Off/On pragmas around Generic Declarations and Mappings** — Setting this option leads to enclosing the generic declarations and generic mappings with *translate_off* and *translate_on* pragmas in the generated HDL.
6. Click **OK** to execute your settings.

In the diagram, if you right-click on the instance and select **Show Text** from the popup menu, the generic mappings are consequently shown as text objects on the diagram. The current dialog box settings are also shown in the *object tips* of the mappings text.

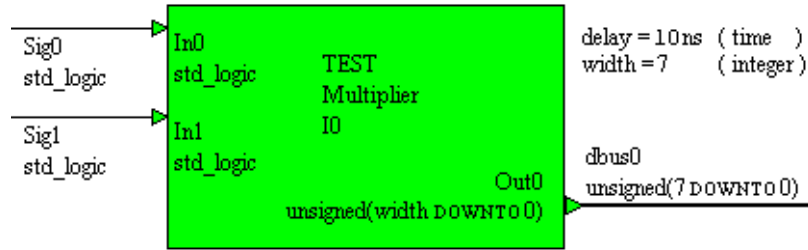
Note



You can set the visibility of VHDL generic values from the block or component Text Visibility pages of the Object Properties dialog box.

In the following example, the value of width is used to specify the upper bounds for the output bus and has been changed to 7 for this instance of the multiplier component. The declaration for

delay specifies a variable value for an internal signal used inside the component which has been set to the value 10 ns for this instance.



The mapping can be set to discrete value or an expression. For example, you could set width in the example above to the value breadth where breadth is declared as a VHDL generic for the parent design unit.



Tip: If you select more than one instance in a VHDL design unit, open the **Generics** page in the Object Properties dialog box and then click **Update List**, the table displays all the generics of all the selected instances.

In case of mixed-language designs, if you select a VHDL instance and a Verilog instance and invoke the Object Properties dialog box, the tool checks whether the language of the parent view is VHDL or Verilog and displays either the generics or parameters respectively. Also, a message is raised in the Log Window stating that you cannot view both VHDL generics and Verilog parameters simultaneously.

Related Topics

- [Generics and Parameters Tables](#)

Editing Verilog Parameter Values for Instances

Follow this procedure to edit Verilog parameter declarations on the level of instances.

Prerequisites

- You must have the Verilog parameter declarations preset for the component or block; if not, refer to [“Defining Parameters for Components and Blocks”](#) on page 185.

Procedure

1. Open the top-level design of the component instance or block; that is, the component or block for which you previously declared Verilog parameters. Refer to [“Defining Parameters for Components and Blocks”](#) on page 185 for information on declaring Verilog parameters.

For example, if you have defined parameter values for the *tester* block, open the top-level *uart_tb* where an instance of *tester* is available with the name *U_0*.

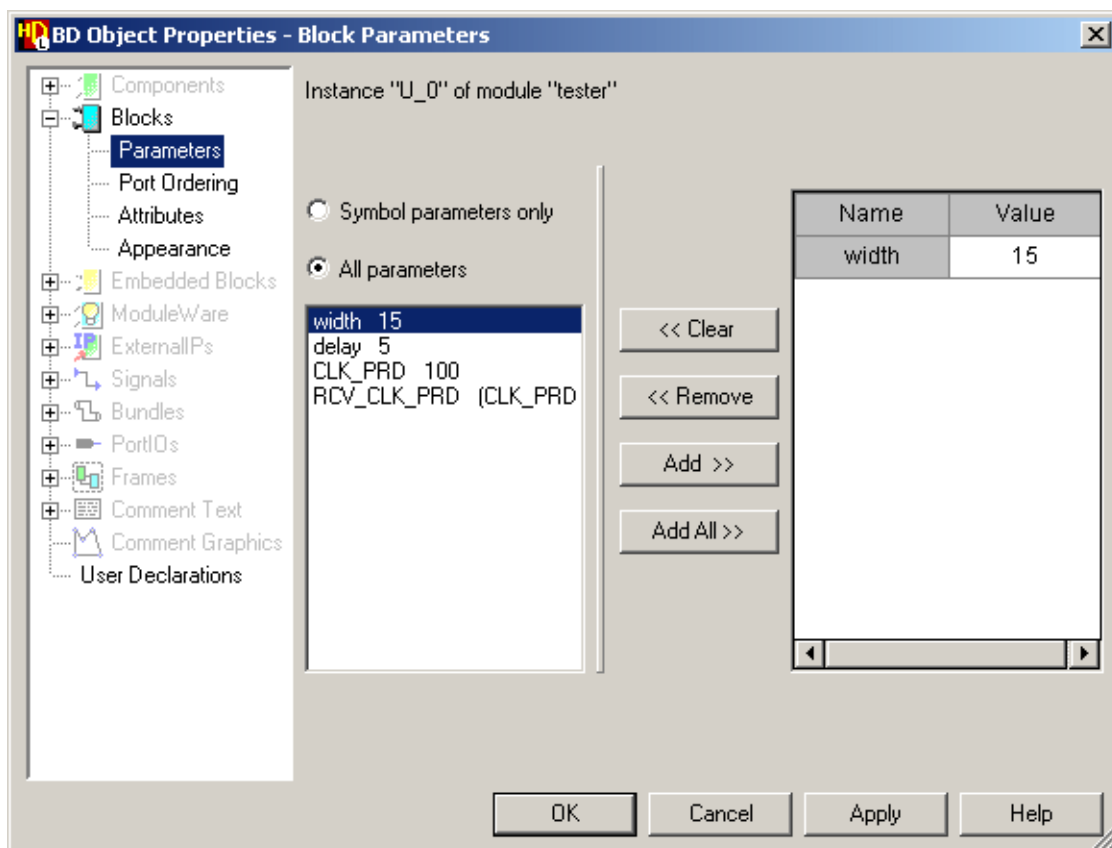
2. Right-click on the instance and choose **Object Properties** from the popup menu.

Following the example in step 1, right-click on the *U_0* instance in the top-level *uart_tb*.

3. Open the **Parameters** page and then select one of the following options to update the list of parameters: **Symbol parameters only** or **All Parameters**.

This means you can choose either to update the list from the Verilog parameters previously declared in the Parameters Table of the symbol only, or to include all parameters previously declared in the Parameters Table in addition to those defined in the User Declarations. The latter option may take several seconds to retrieve all the parameters for a large module.

According to the selected option, the corresponding parameters are displayed in the list along with their default declarations.



Note

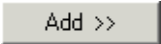

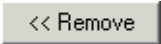
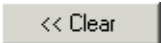


The **Parameters** page of the Object Properties dialog box is available if the language of the selected instance is Verilog or the **Generics** page if it is a VHDL instance.

4. Select the parameter you need to override and then click **Add**. If you want to override all the parameter values, click **Add All** directly (you can also add all parameters by pressing the **Shift** key with the **Add** button). Consequently, the parameters along with their values are moved to the right-hand-side table.

You can control which parameters to override through the following buttons:

Table 3-7. Object Properties — Parameters Page Controls

Button	Description
	Add selected parameter to the table
	Add all parameters to the table
	Remove selected parameter from the table
	Clear all parameters from the table

Note

Any Verilog parameters with already overridden values are listed in the table.

5. In the table, use the **Value** column to insert the mapped value which shall override the parameter's default declaration.

Apply step 4 and 5 to all parameters you need to override for the instance.

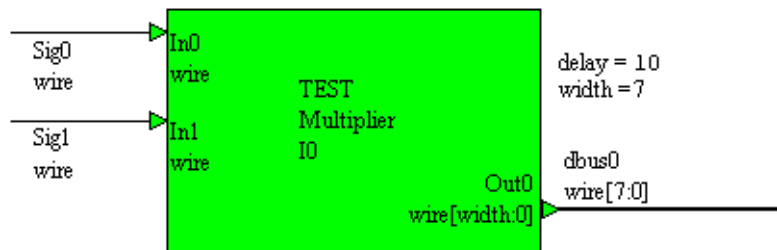
6. Click **OK** to execute your settings.

In the diagram, if you right-click on the instance and select **Show Text** from the popup menu, the parameter mappings are consequently shown as text objects on the diagram. The current dialog box settings are also shown in the *object tips* for the mappings text.

Note

You can set the visibility of Verilog parameter values from the block or component Text Visibility pages of the Object Properties dialog box.

In the following example, the value of width is used to specify the upper bounds for the output bus and has been changed to 7 for this instance of the multiplier component. The declaration for delay specifies a variable value for an internal signal used inside the component which has been set to the value 10 ns for this instance.



The mapping can be set to discrete value or an expression. For example, you could set width in the example above to the value breadth where breadth is declared as a Verilog parameter for the parent design unit.



Tip: If you select more than one instance in a Verilog design unit, and then open the **Parameters** page in the Object Properties dialog box, the table displays all the overridden parameters that are common to all the selected instances.

In case of mixed-language designs, if you select a Verilog instance and a VHDL instance and invoke the Object Properties dialog box, the tool checks whether the language of the parent view is VHDL or Verilog and displays either the generics or parameters respectively. Also, a message is raised in the Log Window stating that you cannot view both VHDL generics and Verilog parameters simultaneously.

Related Topics

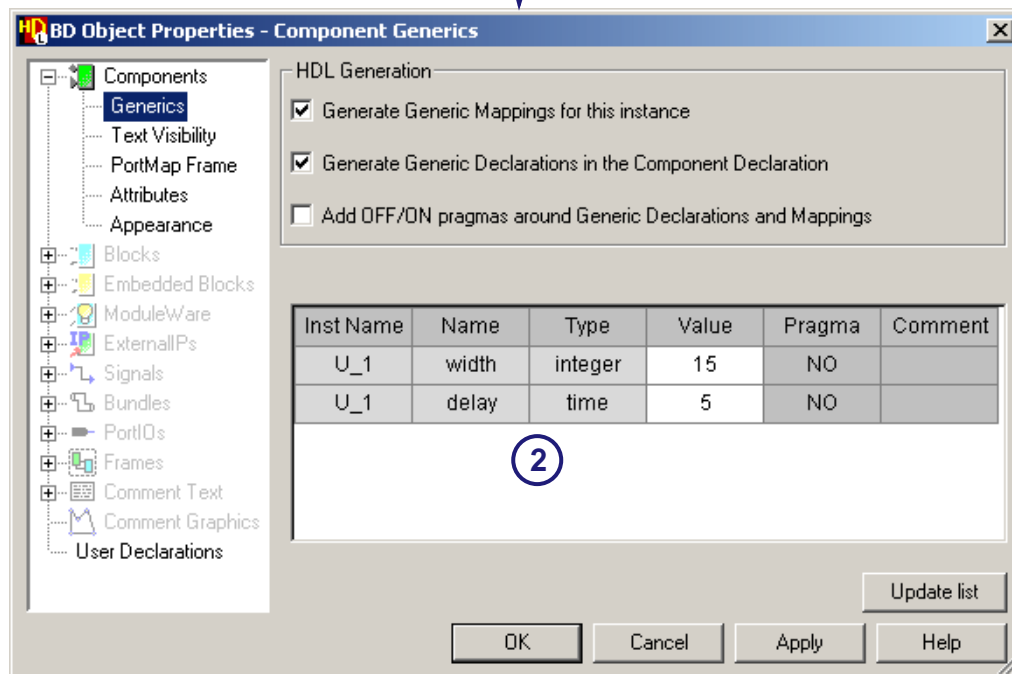
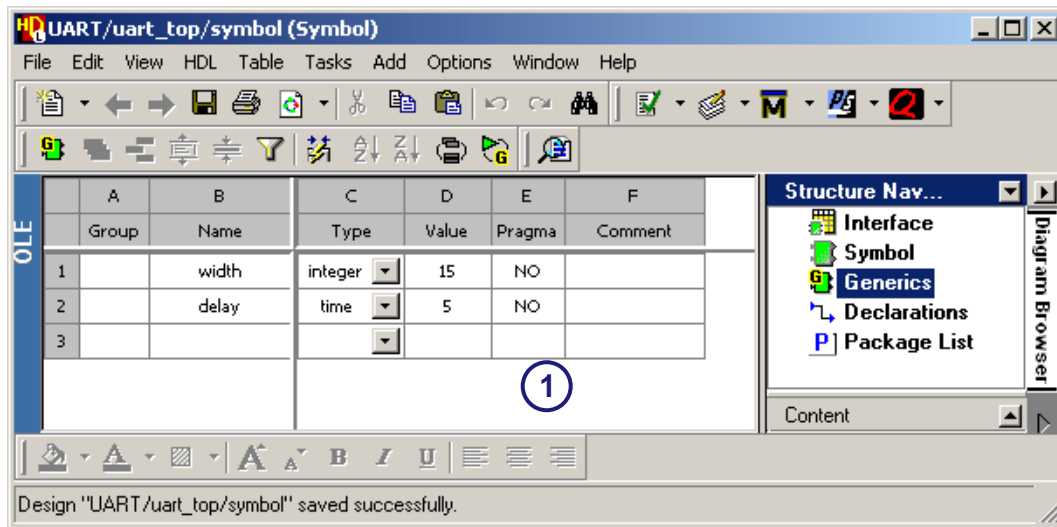
- [Generics and Parameters Tables](#)

Generics and Parameters Synchronization

VHDL Generics and Verilog Parameters are declared through the Generics Table and the Parameters Table respectively; refer to “[Generics and Parameters Tables](#)” on page 176.

The standard usage method is to declare generics/parameters first in the component symbol or block view, and then update the generics/parameters on the level of instances and override the

preset generics/parameters declarations with mapped values. Refer to “[Defining Generics and Parameters](#)” on page 184 and “[Editing Generics and Parameters for Instances](#)” on page 186.



Nevertheless, the Generics and Parameters Tables are available in all views of all types (block diagram, IBD view, state diagram, flow chart, truth table), thus enabling you to declare generics and parameters not only through the component symbol or block view, but also through other views of the design unit, whether default or non-default views.

For example, the following figure illustrates the *accumulator* component (in the *Sequencer_vlg* library) and its different graphical views; as shown in the figure, the *accumulator* component

has a symbol view, in addition to the default view *flow* and the non-default view *tbl*. According to the standard usage method, parameters would be declared through the Parameters table in the symbol view; however, a Parameters table is also available in the default and non-default view.

Design Unit	Type
Sequencer_vlg	
accumulator	Component
accumulator	Module
flow	Flow Chart
symbol	Symbol
tbl	Truth Table
control	Block
fibgen	Component
fibgen_tb	Component
fibgen_tester	Block

On declaring generics/parameters in one of the design unit views, the declarations are reflected on the rest of the views in most cases. This synchronization between views depends on the type of the view in which the declarations occurred as follows:

- If declarations are made in a component symbol, it automatically synchronizes with both the default and non-default views of the design unit on saving.
- If declarations are made in a default view, it automatically synchronizes with both the symbol and non-default views on saving.
- If declarations are made in the non-default view, no automatic synchronization occurs whether in the symbol or the default view on saving.

Related Topics

- [Generics and Parameters Tables](#)
- [Defining Generics and Parameters](#)
- [Editing Generics and Parameters for Instances](#)

Opening Block and Component Views

You can open down into a child view of a symbol or tabular IO view (or of a block, embedded block or component on a block diagram or IBD view) by using the **Right** mouse button to display an **Open As** popup menu for the selected object. The menu allows you to choose any existing views that exist for the object or open down to create a new child view.

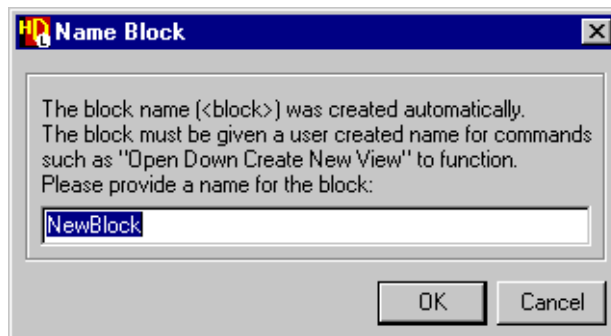
If the selected object is a component, you can also choose **Symbol** to open the graphical symbol editor or **Interface** to open the component interface in the tabular IO editor.

If one or more views already exist, double-clicking on the parent object opens the current view without displaying the menu. However, you can open any of the other existing views (or create a new view) from the menu.

When you create a new view, the File Creation Wizard is displayed. The wizard allows you to choose a graphical view, HDL view (with the same language as the parent view), registered view or text file. You can optionally select a template for a HDL text, registered or text file and specify the new view name.

Refer to “Using the Design Content Creation Wizard” in the [HDL Designer Series User Manual](#) for more information.

If you open down from a new unnamed block, the Name Block dialog box is displayed for you to enter the name for the block.



Once you have named the new block, the File Creation Wizard is displayed to create the new view.

View Initialization

When you create a new block diagram or IBD view by opening down from an existing block diagram the new view is initialized with the interface ports defined on the parent view.

In general, these ports have the same properties as the ports on the parent view. However, if you open down into a block diagram or IBD view from a Verilog symbol or tabular IO view, the *reg* type output ports on the symbol are automatically initialized as *wire* type ports on the child view.

Note



Reconcile interface ignores *reg* and *wire* differences between Verilog ports in the parent and child diagram.

Nets connected to ports in the child diagram can be explicitly changed to *reg*. Such a change is preserved even if the symbol port is a *wire* and other port changes are propagated from the symbol to the child diagram.

Setting the Default View

You can set the default view of a block or component instance by selecting **Change Default View** from the popup menu and selecting the required view from the list of available views in the cascade menu.

Mixed Language Designs

In HDS a design unit can be created as a block or a component. Any design unit whether a block or a component can be described as a state diagram, truth table, flow chart or HDL text view using the VHDL or Verilog language.



Tip: You cannot create VHDL and Verilog views for the same design unit.

You can create an unlimited number of component instances in block diagram and IBD views while you can only create a single instance of a block. The language of instanced components can be different from that of their parent views. That is, you can instance a Verilog component in a VHDL block diagram view or the opposite. Blocks can only be added to parent views of the same language.

If you are using a single kernel simulator (such as *ModelSim*), the entire design can be compiled and simulated in a single operation. For other tools, you can compile the design for one language, then change the downstream tool to compile the remaining components which are described using the other language.

In general, VHDL and Verilog data types are automatically mapped.

A VHDL instantiation of a Verilog component associates VHDL signals and values with the Verilog ports or Verilog parameters.

Similarly, a Verilog instantiation of a VHDL component associates Verilog signals and values with the VHDL ports and VHDL generics.

If a Verilog view is used in a VHDL design, the component declaration in the VHDL structural code is created using the following rules:

- The name of the port comes from the component symbol but the VHDL type comes from the connected signal. If no signal is connected, then the default signal or bus type is used.
- VHDL generics are given the type *string* if the first non-whitespace character is a quote character `"`, *integer* if it can be converted to an integer (for example: 3, 10) or *real* if it can be converted to a real (for example: 3.2, 2.1).

Refer to the "Mixed VHDL and Verilog Designs" section in the *ModelSim Users Manual* for more information about compiling mixed language HDL models.



Tip: In mixed-language designs, if you select a VHDL instance and a Verilog instance and then open the Object Properties dialog box, the tool checks whether the language of the parent view is VHDL or Verilog and displays either the generics or parameters respectively. Also, a message is raised in the Log Window informing you that you cannot view both VHDL generics and Verilog parameters simultaneously.

VHDL Instantiation of Verilog Components

The following VHDL types can be connected to Verilog ports:

<code>bit</code>	<code>std_logic</code>	<code>vl_logic</code>
<code>bit_vector</code>	<code>std_logic_vector</code>	<code>vl_logic_vector</code>

The *bit* and *std_logic* types are sufficient for most applications, but the *vl_logic* type is provided as a pre-compiled protected library (*verilog*) in case you need access to the full Verilog state set.

The *vl_logic* type is an enumeration that defines the full state set for Verilog nets, including ambiguous strengths.

A VHDL generic is generated for each Verilog parameter in the Verilog module which has an initial value that does not depend on any other parameters.

The VHDL generic type is determined by the Verilog parameter's initial value as follows:

Verilog Parameter	VHDL Generic
<code>integer</code>	<code>integer</code>
<code>real</code>	<code>real</code>
<code>string literal</code>	<code>string</code>

The default value of the VHDL generic is the same as the Verilog parameter's initial value. For example:

Verilog Parameter	VHDL Generic
<code>parameter p1 = 1 -3;</code>	<code>p1 : integer := -2;</code>
<code>parameter p2 = 3.0;</code>	<code>p2 : real := 3.000000;</code>
<code>parameter p3 = "Abc";</code>	<code>p3 : string := "Abc";</code>

A VHDL port clause is generated for each Verilog port. The VHDL port type can be *bit*, *std_logic*, and *vl_logic*. If the Verilog port has a range, then the VHDL port type can be

bit_vector, *std_logic_vector*, or *vl_logic_vector*. If the range does not depend on Verilog parameters, then the vector type is constrained accordingly, otherwise it is unconstrained.

Verilog Instantiation of VHDL Components

You can reference a VHDL entity or VHDL configuration from Verilog as though the design unit is a Verilog module of the same name (in lower case).

A VHDL view can be instantiated in Verilog if it meets the following criteria:

- The design unit has an entity and architecture or is a configuration declaration.
- The entity ports have any mix of the types: *bit*, *bit_vector*, *std_ulogic*, *std_ulogic_vector*, *vl_ulogic*, *vl_ulogic_vector* or their subtypes.
- VHDL generics are of type *integer*, *real*, *time*, *physical*, *enumeration* or *string*. (String is the only composite type allowed.)

Port associations may be named or positional. Use the same port names and port positions that appear in the entity.

Chapter 4

Block Diagram Editor

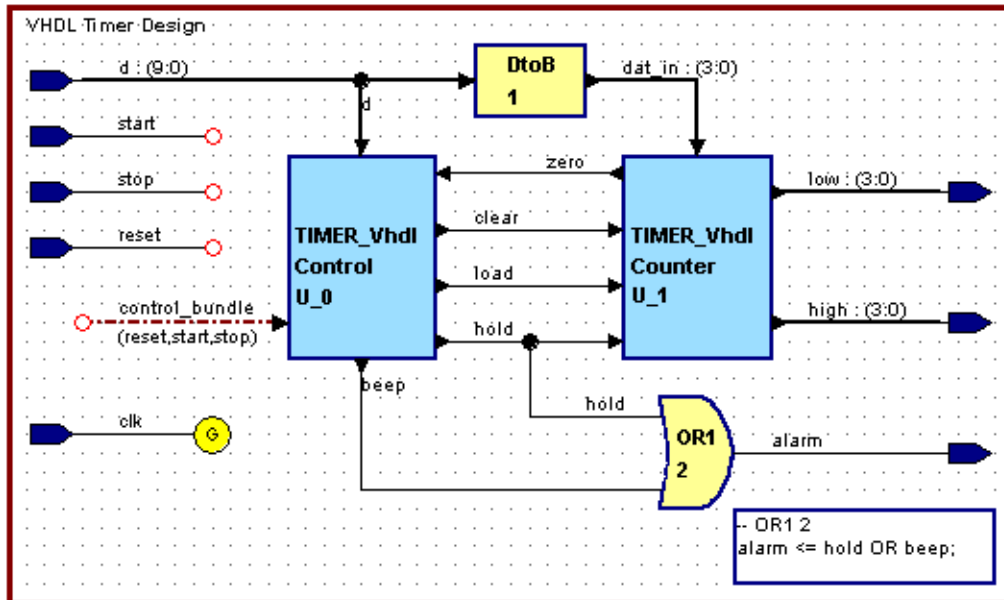
This chapter describes how the structure of a design can be represented using a graphical *block diagram*.

Block Diagrams	200
Block Diagram Notation	201
Blocks and Components	202
Embedded Blocks and Embedded Views	203
Signals, Buses and Bundles	204
Ports and Signals	205
Changing the Display of Port Properties	206
Changing the Display of Signal Properties	208
Block Diagram Editor Toolbar	211
Adding Nets on a Block Diagram	212
Routing Nets	212
Adding a Signal or Bus on a Block Diagram	213
Ripping from a Bus	215
Adding Signal Stubs on a Block Diagram	217
Adding a Bundle on a Block Diagram	217
Adding Signals to a Bundle	218
Ripping from a Bundle	218
Using HDL Text to Combine or Split Signals	220
Adding Ports on a Block Diagram	220
Adding a Global Connector on a Block Diagram	222
Connecting Overlapping Nets	222
Connecting Nets to a Block or Component	223
Connecting Nets to a Port Map Frame	225
Highlighting a Net on a Block Diagram	225
Logic Shape Notation	227
Changing the Shape of a Block or Component	228
Hiding Ports on a Block or Component	230
Indicating Not or Clocked Ports	231
Setting Block Diagram Preferences	231

Refer to the [Block Diagram and IBD Views](#) chapter for information about procedures which are common to the block diagram and IBD view editors.

Block Diagrams

A *block diagram* represents a design as a number of functional blocks connected by *signals*. For example, the top level block diagram for the *TIMER_Vhdl* design:



Each functional block may be defined by a *state machine*, *flow chart*, *truth table* or *HDL text* view, or be decomposed into a hierarchy of lower level blocks.

A *block* or *embedded block* with an interface defined by the connected signals can be created on the diagram or an existing *component* which has a fixed interface can be instantiated. A block can be converted to a component and re-used any number of times in the same (or different) designs.

A signal can be added as a thin polyline with a name and *type* representing a scalar connection (for example, the signal *start* in the picture has type *std_logic*) or as a thick polyline representing a vector bus with a name, type and *bounds* (for example, the bus *dat_in* has type *unsigned* and bounds *3 DOWNTO 0*).

If hierarchical operations are performed on signals connected to a block, the block interface is automatically modified.

A component has a fixed interface which can only be changed by editing its *symbol* or if the component is itself defined by a block diagram by editing this block diagram.

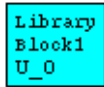
Note



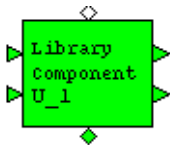
Mapping between actual signals on the block diagram and formal *ports* on a component with different properties can be achieved by using a port map frame.

Block Diagram Notation

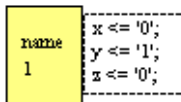
A block diagram is constructed from the following graphical objects:



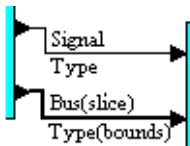
A newly added *block* has no input or output *ports*. Its interface is defined by the connections on the block diagram.



A *component* has fixed input and output ports which are shown on the *symbol* and can be used to connect *signals* and *buses* which have compatible properties.



An *embedded block* can be used to add concurrent *HDL* to a block diagram without creating hierarchy. Its interface is defined by the connections on the block diagram.



Groups of signals or buses can be combined into a *bundle* which can be connected to a block or be unconnected.



External connections are represented by input and output *ports*. Bidirectional signals are connected using an inout or buffered signals using a buffer port.



Signals connected to a *global connector* are implicitly connected to every block on the diagram.




A *net connector* joins together *signals* or *buses* with the same properties. An explicit connection is shown as a filled dot ● or a “dangling” net connector shown as an unfilled dot ○.is an implicit on-page connection to other signals or buses with the same name and type on the diagram.





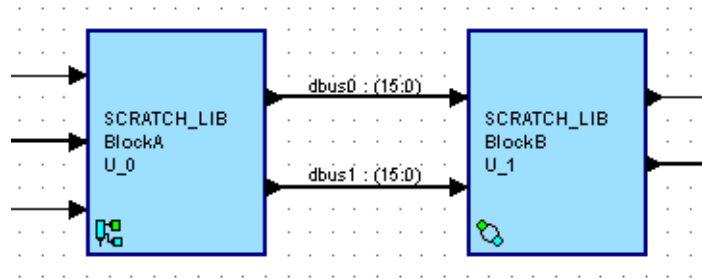
A *ripper* splits or combines slices or elements of a *net* which have the same name and *type*. It can also be used to add or remove nets from a bundle.

Blocks and Components

When you add a new *block* it has no *ports* (unlike a *component* which has fixed port interfaces defined by its *symbol*). Ports are automatically added when you connect *signals* and a connected block on a block diagram looks similar to a component (although they are normally drawn in a different default color).









An icon indicating the type of view is displayed by default. A  icon is displayed if the view type is unrecognized. You can choose to remove this icon by unsetting a display setting preference. Refer to “[Setting Block Diagram Preferences](#)” on page 231 for information about changing the display setting preferences.

For example, the following picture shows two blocks. *BlockA* is defined by a block diagram (indicated by the  icon); *BlockB* is defined by a state diagram (indicated by the  icon):



The following icons are used to identify graphical editor and HDL text views:

Table 4-1. Graphical Editor and HDL Text Views Notation

Icon	Description
	Block diagram view
	IBD view
	State diagram view
	ASM chart view
	Flow chart view
	Truth table view
	VHDL file
	Verilog file

If the default view is any other registered view, the appropriate icon is displayed.

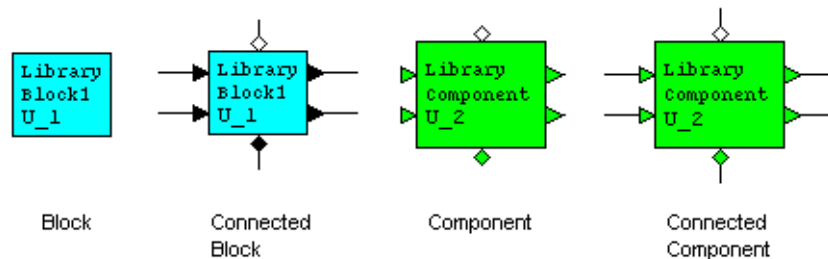
Refer to “File Registration” in the *HDL Designer Series User Manual* for information about registered view types.

A block can be converted into a re-usable component but a component cannot be converted back to a block. The interface for a block can be edited dynamically by adding or deleting the connected signals.

Note

You must rename a block before converting it to a component if it was under version management control. A prompt message will appear if you attempted converting blocks to components without renaming them while using RCS or Super CVS version management interfaces.

The interface for a component that is defined by a child block diagram can be edited by editing that diagram. Otherwise the symbol must be explicitly edited using the symbol editor.



The interface to a block is defined by its connections and all signals have the same names in its child views. The interface to a component is defined by its symbol and signals can be connected to ports with a different name provided that they have compatible properties. You can also connect to ports indirectly by using a *port map frame*.

A block or component can have one or more child *block diagram*, *IBD view*, *flow chart*, *state diagram*, *truth table*, *HDL text* or other registered views. When there is more than one child view, you can change the *current view* by setting the *default view* or by using a *VHDL configuration*.

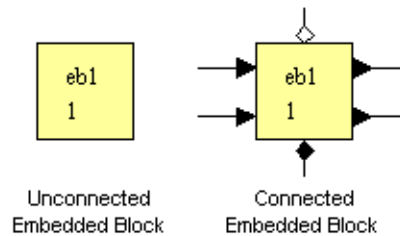
Embedded Blocks and Embedded Views

An *embedded view* is saved as part of the *block diagram* or *IBD view* on which it is instantiated and does not create a separate *design unit*. When HDL is generated, concurrent code is generated for the embedded view in the same files (*VHDL entity* and *VHDL architecture* or *Verilog module*) as the structural description of the design unit.

An embedded view has an interface described by the signals connected to the embedded block which represents it on the block diagram or IBD view.

An *embedded block* looks very similar to a block (although they are normally drawn in a different default color) but has no library name since it is always in the same name space as the

view. However, it does have a unique name and a number which determines the relative ordering when there are multiple unconnected embedded blocks in the generated HDL.



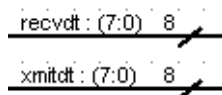
An embedded view can be used to represent a VHDL process, procedure or function, Verilog always or initial code, Verilog task or any other concurrent statements and is typically described by HDL text. The concurrent HDL can also be represented by a graphical state diagram, flow chart or truth table view which is normally hidden but can be displayed in an editor window by double-clicking on the embedded view.

Signals, Buses and Bundles

Connections can be added between objects on a block diagram using *signals*, *buses* or *bundles*. The only semantic difference between a signal and bus is the polyline style and you can change the style used at any time by editing the signal properties. Signal style is typically used to represent scalar information and bus style to represent a composite array.

A new signal (or bus) is automatically declared when it is added to a diagram and the declaration is stored as a property of the net. However, you can specify a *slice* or index for each segment of a net (for example, when slices are ripped from a bus). The declarations for each named net are shown as text Declarations on the diagram and the visibility of properties displayed for each net segment can be individually set.

You can optionally set a preference which displays a width label on bus nets. When enabled, the width is shown independently from the bounds which can be optionally displayed in the signal properties. The following example shows two bus nets with bounds (7 DOWNT0 0) and a width label indicating that the buses are 8 bit wide:

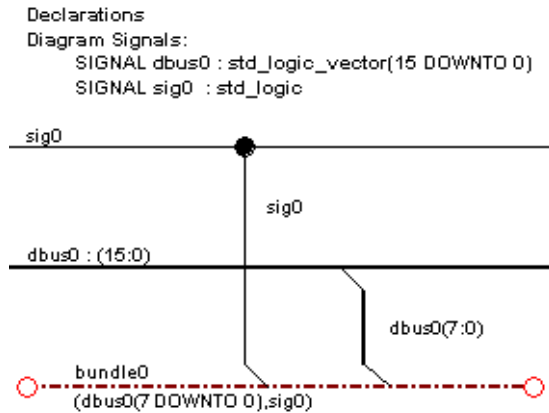


Refer to [“Setting Block Diagram Preferences”](#) on page 231 for information about setting this preference.

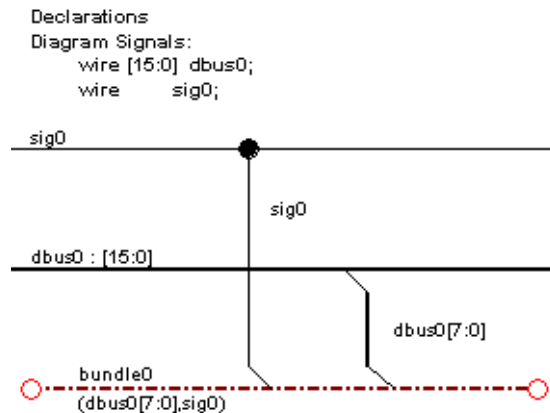
A bundle is a composite grouping of connections which may include any number of signals and buses and is normally shown on the diagram as a dotted line.

The following pictures illustrate how a default signal (*sig0*), bus (*dbus0*) and bundle (*bundle0*) are shown in VHDL and Verilog syntax. Notice how the name, type and bounds are shown for the first use of a bus. Only the name is initially shown for subsequent segments but the slice (or element) is shown if defined. The bundle shows the names (and slices if defined) of the signals it contains.

VHDL



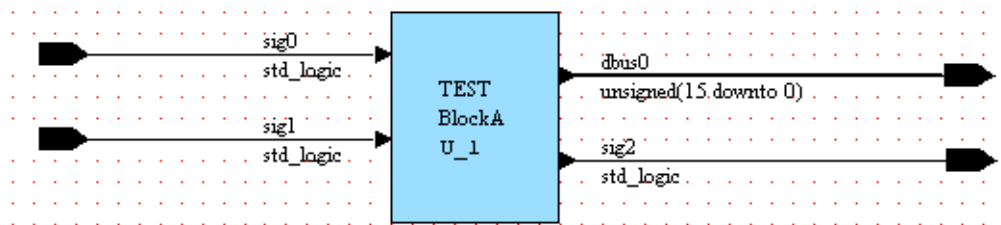
Verilog



Refer to [“Changing the Display of Port Properties”](#) on page 206 and [“Changing the Display of Signal Properties”](#) on page 208 for more information about the properties displayed on ports and signals.

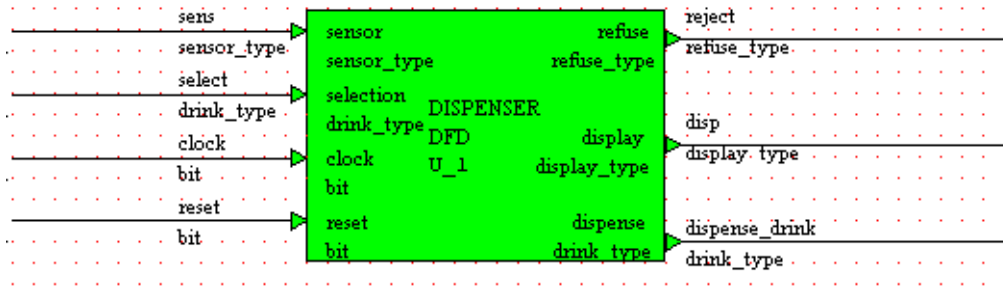
Ports and Signals

A port on a block or an external port on a block diagram has the same name and properties as the connected signal (or bus).

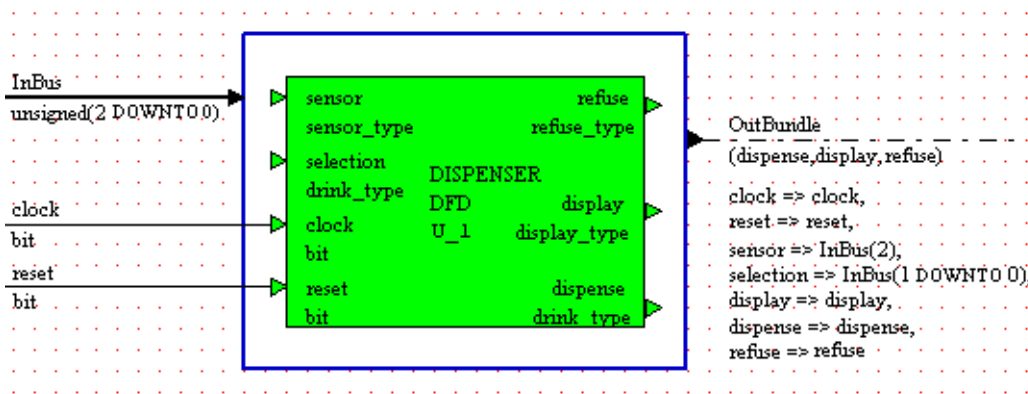


A port on a component has a name defined by its symbol and may be connected to a signal with a different name on the block diagram provided the port and signal have compatible properties.

For example, the signals *sens*, *select*, *reject*, *disp* and *dispense_drink* are connected to the ports *sensor*, *selection*, *refuse*, *display* and *dispense* in the following example:



You can also connect actual signals on a block diagram to formal ports on a component which have different properties by using a port map frame.



For more information about using port map frames, refer to [“Port Map Frames”](#) on page 283.

Changing the Display of Port Properties

You can change the properties that are displayed for any selected input/output (IO) port or component port on a block diagram.

Note

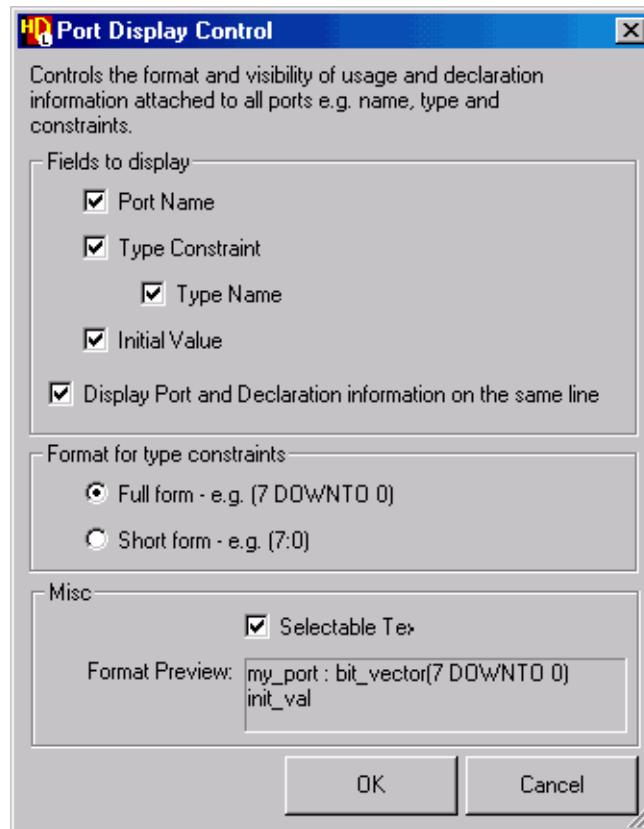


Refer to [“Setting Block Diagram Preferences”](#) on page 231 for information about setting default port display properties.

You can change the visibility of the properties for an input or output port on a block diagram by selecting the port and choosing **PortIO Text Visibility** from the popup menu to display the PortIO Display Control dialog box.

A similar dialog box is displayed when one or more component instances are selected and you choose **Port Visibility** from the popup menu.

This Port Display Control dialog box sets the properties displayed for all ports on the selected instance.



The same dialog box is also displayed when you choose **Port Display** from the **Diagram** or popup menu in the symbol editor or use one of the **Port Display** buttons in the **Symbol** tab of the Object Properties dialog box.

Note



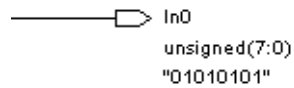
Note that you can set separate port properties for ports displayed in the symbol editor and the default port properties displayed when the symbol is instantiated on a block diagram.

You can choose to display the port name, type constraint (including the type name and bounds if defined) and VHDL initial value (if set) or Verilog delay (if set). Only the port name is shown by default but you can choose to display the type constraint and initial value (or delay).

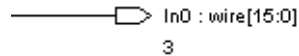
The name and type constraints are usually shown on a single line (separated by a colon) with the initial value or delay on a separate line. For example, a VHDL port *In0* with type *unsigned*, bounds *(7 DOWNTO 0)* and initial value *"01010101"* would be shown as:

— In0 : unsigned(7 DOWNTO 0)
"01010101"

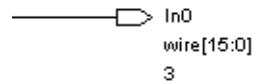
You can choose to display the port name and declaration information on separate lines and use short format (which omits the VHDL keywords *DOWNTO* or *TO*). For example, the example above becomes:



Verilog signals are similar except that the bounds is always shown with short format. For example, a Verilog bus *In0* with type *wire*, bounds *[15:0]* and delay *#3* would be shown as:



or with the signal and declaration on separate lines as:



You can choose whether to display the port name, type constraint, type name and initial value or delay for the selected ports by setting a tri-state check box.

When multiple ports which have different display properties are selected, the check box is dimmed and no changes are made for that property when you execute the dialog box.

Note



The full port declarations are available in the IBD view or (when *object tips* are enabled) in the object information displayed when the cursor is moved over a port in the block diagram editor.

If the **Selectable Text** option is set, the properties text strings can be individually selected when the symbol is instantiated as a component on a block diagram but if this option is unset, the properties cannot be selected. If selectable text is enabled, the port text can be moved independently but you cannot change the visibility of an individual port property. This option is not present when an external port is selected.

You can also hide any of the port properties by selecting the text and choosing **Hide Text** from the popup menu.

You can make all properties currently set in the port display properties visible by choosing **Show Text** from the popup menu when the port is selected.

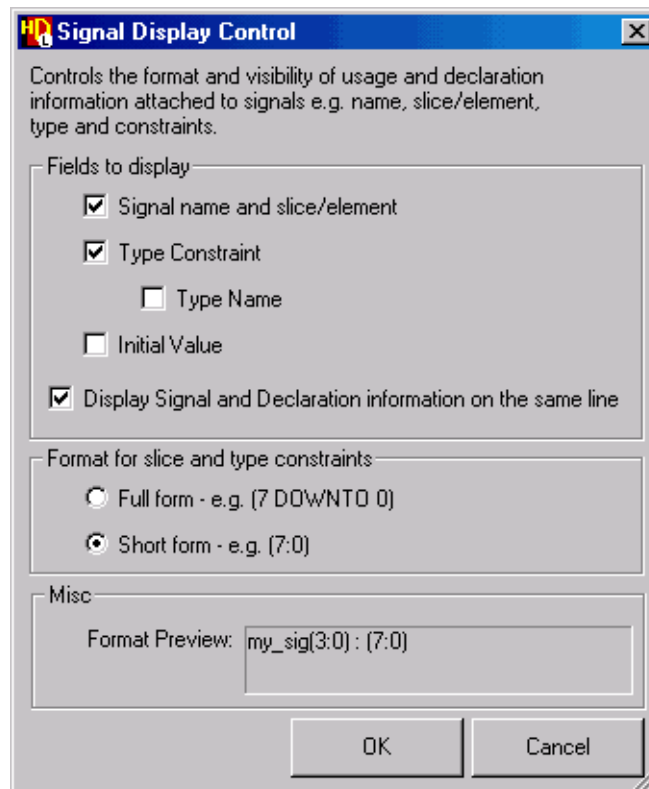
Changing the Display of Signal Properties

You can change the properties that are displayed for any selected signal or bus on a block diagram.

Note

Refer to [“Setting Block Diagram Preferences”](#) on page 231 for information about setting default signal display properties.

You can change the visibility of the properties for the selected segment of a signal or bus net on a block diagram by choosing **Signal Visibility** from the popup menu. The Signal Display Control dialog box is displayed:



You can also set the signal visibility by displaying the **Text Visibility** page of the Object Properties dialog box when a signal is selected.

You can choose to display the signal (or bus) name (including the slice or element if defined), type constraint (including the type name and bounds if defined) and VHDL initial value (if set) or Verilog delay (if set).

The name and type constraints are usually shown on a single line (separated by a colon) with the initial value or delay on a separate line.

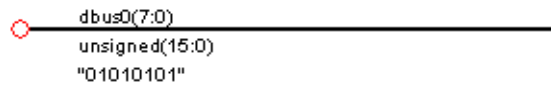
For example, a VHDL bus *dbus0* with slice (*7 DOWNTO 0*), type *unsigned*, bounds (*15 DOWNTO 0*) and initial value *"01010101"* would be shown as:

```

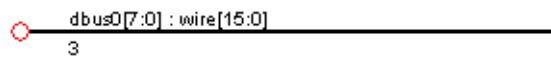
dbus0(7 DOWNTO 0) : unsigned(15 DOWNTO 0)
"01010101"

```

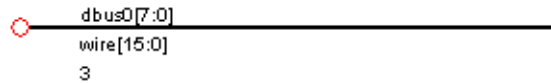
You can choose to display the signal name and declaration information on separate lines and use short format (which omits the VHDL keywords *DOWNTO* or *TO*). For example, the example above becomes:



Verilog signals are similar except that the bounds is always shown with short format. For example, a Verilog bus *dbus0* with slice *[7:0]*, type *wire*, bounds *[15:0]* and delay *#3* would be shown as:



or with the signal and declaration on separate lines as:



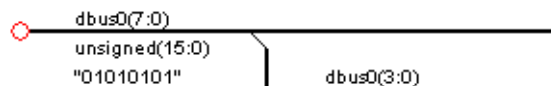
You can choose whether to display the signal name and slice, type constraint, type name and initial value or delay for the selected signals by setting a tri-state check box. When multiple signals which have different display properties are selected, the check box is dimmed and no changes are made for that property when you execute the dialog box.

Note



The full signal declarations are available in the IBD view or (when *object tips* are enabled) in the object information displayed when the cursor is moved over a signal or bus.

You can display different properties against each net segment on the diagram. Typically, the full properties are shown for the first use of a signal or bus but only the name and slice (or element) on ripped signals. For example:
























You can also hide any of the signal properties by selecting the text and choosing **Hide Text** from the popup menu. You can make all text elements associated with a signal visible by choosing **Show Text** from the popup menu when the object is selected.

Block Diagram Editor Toolbar



The following commands are available from the Block Diagram Tools toolbar:

Table 4-2. Block Diagram Editor Toolbar

Icon	Description
	Select text or object
	Select text only
	Select object only
	Add or modify comment text
	Pan the window
	Highlight all segments of the selected net on the block diagram
	Highlight all segments of the selected net in the hierarchy
	Clear net highlighting
	Add a block
	Add a component
	Add a ModuleWare component
	Add an external HDL (IP) model
	Add an embedded block
	Add a FOR, IF, ELSE or BLOCK generate frame
	Add a signal
	Add a bus
	Add a bundle
	Add a port
	Add a global connector
	Add a panel
	Show the block diagram as an IBD view

Note



Some buttons (for example, ) have a pulldown palette which allows you to change the command as shown in the table. The  button has a pulldown menu which allows you to choose a FOR, IF, BLOCK (VHDL only) or ELSE (Verilog only) generate frame.

The toolbar can be displayed or hidden by setting the **Block Diagram Tools** option in the **Toolbars** cascade of the **View** menu.





Refer to “[Toolbars](#)” on page 20 for more information about toolbars including the standard toolbar buttons which are available in more than one editor.

Adding Nets on a Block Diagram

You can add [signal](#) or [bus nets](#), [bundles](#), [ports](#) and [global connectors](#) on a [block diagram](#) by using the **Add** menu or by using one of the block diagram toolbar buttons.













Note



The , ,  and  buttons have a pulldown palette which allows you to change the command as shown in the table below.

Some objects can also be added using a shortcut or mnemonic keys shown in the table below:

Table 4-3. Block Diagram Commands for Adding Nets

Button	Shortcut	Mnemonic	Description
	F7	S	Add a signal net
	none	none	Add a signal net with a port
	Shift + F7	U	Add a bus net
	none	none	Add a bus net with a port
	none	none	Add a bus net with a ripper
	Ctrl + F7	N	Add a bundle
	none	none	Add a bundle with a ripper
	F8	I	Add an input port
	F9	O	Add an output port
	F11	T	Add a bidirectional (inout) port
	F12	F	Add a buffer port (VHDL only)
	F5	G	Add a global connector

Routing Nets

A net can be routed as a simple signal, vector bus or as a bundle which groups signals and buses together.

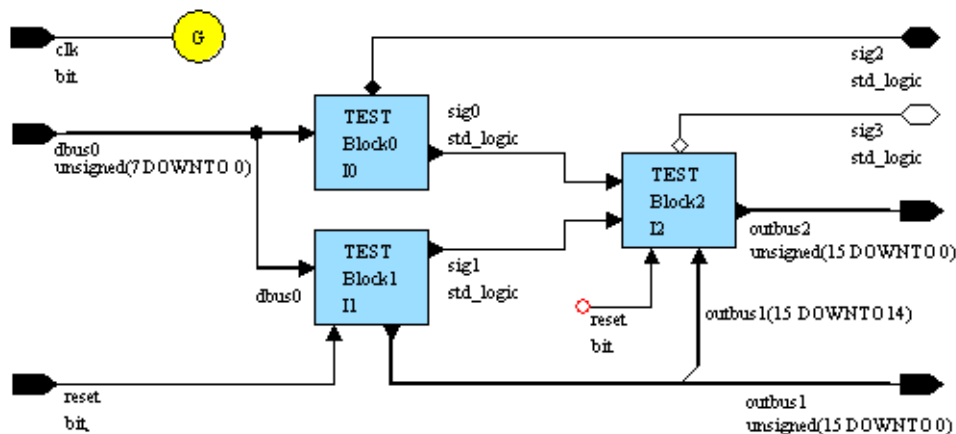
A net connection (shown as a filled dot ●) can be made by terminating a signal, bus or bundle on an existing net. When a junction between buses is made using a [net connector](#) in this way, all net segments have the same properties and only the net name is shown on the new net segment.

You can also rip an index or slice from a bus by using a ripper. For example, bits 15 and 14 are ripped from the 16-bit bus *outbus1* in the picture on the next page.

An input or output port is implicitly added to the symbol for a block when you connect a signal or bus to or from the block.

You can change the port direction (in, out, bidirectional or buffer) by selecting the port and choosing **Change Mode** from the popup menu which can be displayed using the **Right** mouse button.

Any signal, bus or buffer connected to a *global connector* is implicitly connected to every block in the block diagram. For example, *clk* is a clock signal connected to *Block0*, *Block1* and *Block2* in the picture.






A net connector can also be used to implicitly connect by name separate segments of a signal, bus or bundle which have the same name. For example, *reset* is a reset signal connected by name to *Block1* and *Block2* (but not *Block0*) although it not explicitly connected to *Block2*.

Adding a Signal or Bus on a Block Diagram

When you add a signal or bus, the cursor changes to a cross-hair which allows you to specify a *source*, *destination* and any number of *route points* by clicking the **Left** mouse button.

The source and destination can be a *block*, *embedded block*, *port* or *port map frame* on a *component*. You can also connect to an existing signal, bus, bundle or global connector on the block diagram.

If the source is in open space on the diagram or the signal is terminated by double-clicking in open space, a dangling *net connector* is created. However, you can use the  pull-down menus to change the default buttons to  or  and automatically create an input port if the source is an open space or an output port if a signal is terminated in open space.

Note



If the source object is an input, output, buffer or bidirectional port on a component, the default endpoint has the same mode.


If you add a signal or bus without a port, the name, *type* and *bounds* properties are displayed on the signal. However, if you add a signal or bus with a port, the signal properties are hidden and the properties are shown on the port.

If the source is over an existing signal or bus, the source name, type and bounds are used. However only the name is shown on the diagram for the new net segment. If the source is over a component port, the signal will have the same properties as the port (except in a Verilog diagram when wire type is used when connected to a port with reg type).

If the source is unconnected and the destination is over an existing signal, bus or component port, the destination name, type and bounds are used.


If both ends are connected, the source and destination must have compatible properties.

If neither end is connected, a signal is added with a default name and scalar type or a bus is added with a default name, vector type and bounds.

If the source of a signal is over a bus or if you connect a bus to an existing bus using the  button, the connection is made using a *ripper* and a dialog box is displayed for you to choose the required element index or slice. Refer to “[Ripping from a Bus](#)” on page 215 for more information.

If the source of a signal or bus is over a bundle, a dialog box is displayed for you to choose the required signal, bus and optionally an element index or slice. If the destination is over a bundle, the signal or bus is added to the bundle. Refer to “[Ripping from a Bundle](#)” on page 218 for more information.

You can change the name, type and bounds by clicking on the text to select it and clicking again to edit the text. The text is normally placed midway along the route but can be moved to any other position by dragging it with the mouse.


You can also edit the name, type, bounds and other properties by using the **Signals** tab in the Object Properties dialog box which is displayed when you by use the  button or choose **Object Properties** from the **Edit** menu.

The signal style (scalar signal or vector bus) can be changed in the Object Properties dialog box or you can change the appearance by setting its visual attributes.

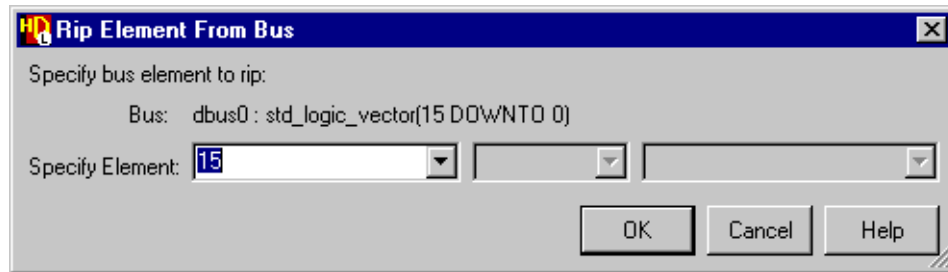
If you do not change the name, each new signal or bus is given a unique name by adding an integer to the default name (for example: *sig1*, *sig2*... for a signal; or *dbus1*, *dbus2*... for a bus). The default signal and bus name, type and bounds can be set by preferences.

The name and type constraints text (if visible) are aligned with the net orientation. Thus the name is horizontal for a horizontal net segment and vertical for a vertical net segment. However, the text can be rotated independently by choosing **Rotate Text** from the popup menu.


Ripping from a Bus

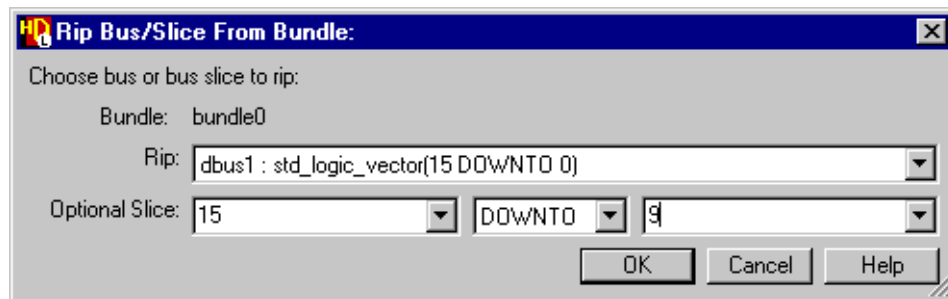
You can rip a single element of a bus by using the  button to connect a signal starting from any point on the bus using a *ripper*.

The Rip Element From Bus dialog box is displayed which allows you to specify the required slice. For example, the following dialog box is displayed when you are using VHDL:



The dialog box allows you to enter or select the index for the required element.

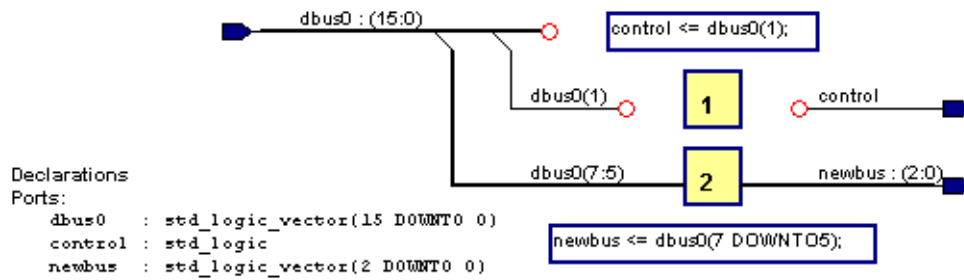
Alternatively, you can use the  button to rip a slice from the bus using the Rip Slice From Bus dialog box:



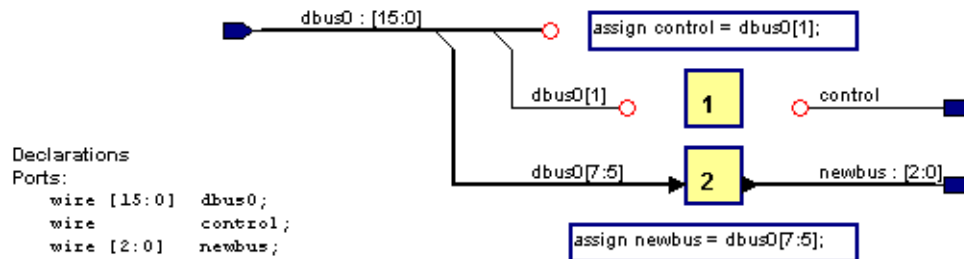
You can choose from a pulldown list of existing slices ripped from the net. You can also use separate pulldown lists to compose the required slice.

The following examples show a two-bit slice *7 DOWNTO 5* (VHDL) or *[7:5]* (Verilog) ripped from *dbus0* which has 15-bit bounds *15 DOWNTO 0* (VHDL) or *[15:0]* (Verilog). A one-bit slice *dbus0(1)* or *dbus0[1]* has also been ripped from the bus and drawn as a signal.

VHDL



Verilog



All other properties for the new net segment are the same as the source bus.



Do not attempt to change the name of a ripped signal or connect a slice or element of a bus directly to an output port. However, you can use embedded HDL text to assign the slice or element to an alternative output signal. In the example, bit 1 of *dbus0* is assigned to the *control* output and the slice to the *newbus* output.

Note



The embedded view containing HDL text can optionally be connected to the signals or buses or you can use connection by name as shown for the *control* signal in the example. You can also map ripped slices or elements of a bus to the ports on a component by using a port map frame.

Note that you can choose **Change to Junction** from the popup menu to replace a *ripper* by a *net connector* representing an unripped junction. You can also choose **Change to Ripper** from the popup menu to replace a *net connector* by a *ripper*. After using these commands, the signal properties for each net segment may need to be edited using the Object Properties dialog box.

You can flip the direction of a ripper joining two nets on a block diagram by using the  and  buttons in the Arrange Object toolbar or by choosing **Flip Ripper** from the popup menu when the ripper is selected.

Adding Signal Stubs on a Block Diagram

You can add signals and buses corresponding to the ports on a component by choosing **Add Signal Stubs** from the **Diagram** or popup menu.

If the component is selected, the Add Signal Stubs dialog box allows you to choose whether to add signal stubs on Input, Output, InOut or Buffer (VHDL only) ports. Signal stubs are added to all unconnected ports of the specified type.

If one or more ports are selected on a block diagram, signal stubs are added to the selected ports and any unselected ports are left unconnected.

Signal stubs are added with properties corresponding to the ports defined on the symbol for the component.

If the component has a different language from the parent view (for example, a Verilog component instantiated in a VHDL view) the signal properties are automatically mapped to the language of the parent view.

The signal stubs are implicitly connected by name to nets with the same name on the diagram and you are warned if any of the net names already exist.

Adding a Bundle on a Block Diagram

Bundles can be useful to connect a group of signals to different parts of a diagram without the need for long connecting nets.

A bundle allows a number of nets to be grouped as a single line on a block diagram. Unconnected bundle segments with the same name are implicitly connected.

When you add a bundle, the cursor changes to a cross-hair which allows you to specify a *source*, *destination* and any number of *route points* by clicking the **Left** mouse button.

The source can be a signal, bus, bundle, block, port map frame, global connector, embedded view or unconnected. The destination can be any object except a signal, bus or port but can also be left unconnected by double-clicking the **Left** mouse button to complete the route.

Although you cannot connect a bundle to a component port, you can connect a bundle to a port map frame and set **Connection By Name** in the object properties for the component to connect signals in the bundle to corresponding component ports with the same name.


A new bundle initially has no content (unless originated on an existing signal, bus or bundle) but you can add signals or buses to a bundle (which is then displayed with the constituent parts after the bundle name) by terminating them over any point on the bundle.

You can then rip signals or buses from the bundle by starting a new net at any point over the bundle.

If any signal or buses are selected when you add a new bundle, the selected signals are automatically included in the bundle. However, the signal names in the bundle are not updated if the discrete signal name is changed unless the signal is explicitly connected to the bundle.

If you connect a bundle to a block, then all of the constituent signals or buses in the block are connected to the block and are represented by separate ports in a child block diagram. Similarly, if you connect a bundle to a global connector, all of the signals and buses it contains are considered to be connected to every block on the diagram.

You can change the bundle name by clicking on the text to select it and clicking again to edit the text. The text is normally placed midway along the route but can be moved to any other position by dragging it with the mouse.

You can also edit the bundle name by using the **Bundles** tab in the Object Properties dialog box which is displayed when you use the  button or choose **Object Properties** from the **Edit** menu.

If you do not change the bundle name, each new bundle is given a unique name by adding an integer to the default name (for example: *Bundle1*, *Bundle2*...). The default bundle name can be set by a preference.

Adding Signals to a Bundle

If existing signals or buses are selected when you add a new bundle, they are automatically included in the bundle.


You can add signals or buses to a bundle by dragging the dangling *net connector* at the destination end of the existing net onto the bundle or by terminating a new signal or bus on the bundle.

Note



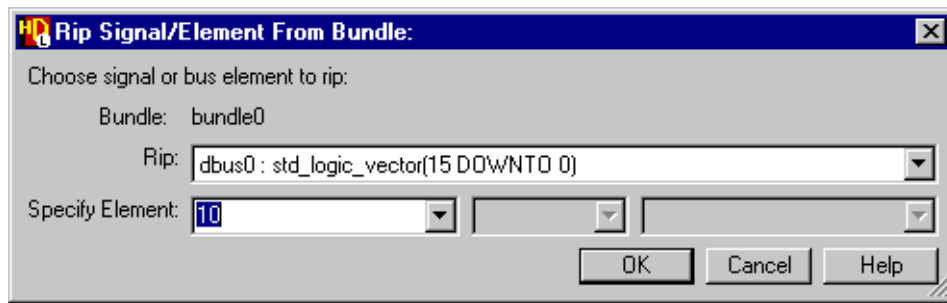
Signals or buses with the same properties are implicitly connected by name and need not be graphically connected.


Ripping from a Bundle

You can rip a signal (or element of a bus) from a bundle by using the  button to connect a signal starting from any point on the bundle using a *ripper*.

The Rip Signal/Element From Bundle dialog box is displayed which allows you to choose from a drop down list of signals and buses in the bundle and (when a bus is selected) specify the required element.

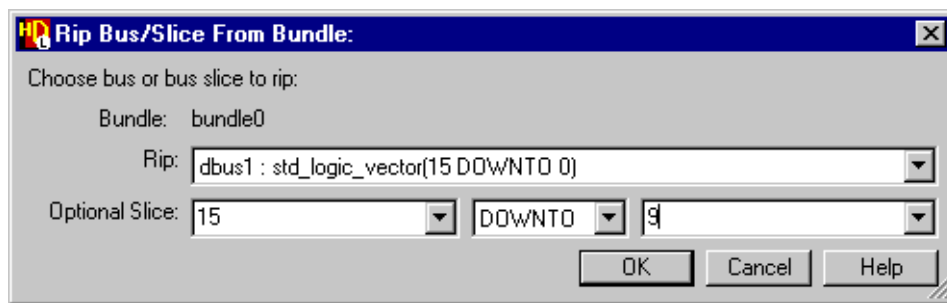
For example, the following dialog box is displayed when you are using VHDL:




You can rip a bus (or slice of a bus) from a bundle by using the  button to connect a bus starting from any point on the bundle.

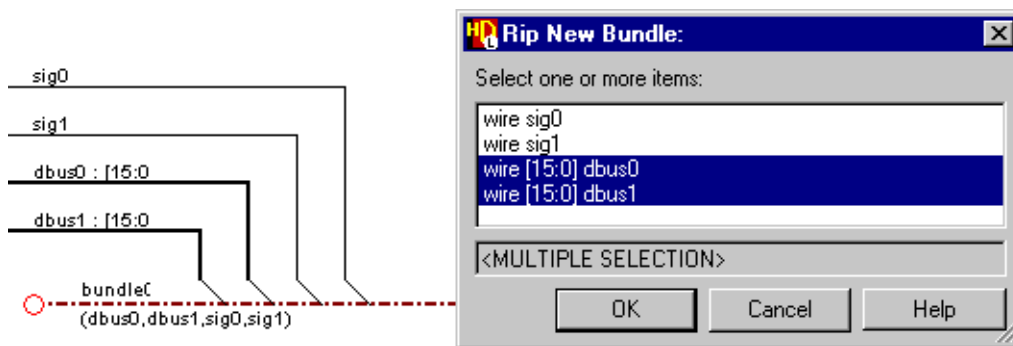
The Rip Bus/Slice From Bundle dialog box is displayed which allows you to choose from a drop down list of buses in the bundle and (optionally) specify the required slice.

For example, the following dialog box is displayed when you are using VHDL:



You can use the  button to rip a secondary bundle. The Rip New Bundle dialog box is displayed which allows you to select one or more signals and buses to include in the new bundle.

For example, the two buses *dbus0* and *dbus1* are being ripped to create a new bundle in the picture below:

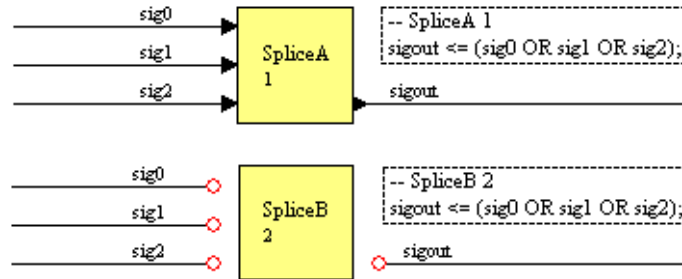


A bundle can be directly connected to a block (but not to a component) and all the nets in the bundle will be available as individual signals and buses in child views of the block.

Using HDL Text to Combine or Split Signals

An Embedded HDL text view can be used to combine or split signals which have different properties by using simple HDL assignment statements.

For example, the following picture shows embedded HDL text used on a block diagram as a splice to combine the signals *sig0*, *sig1* and *sig3* into signal *sigout*. The signals do not need to be explicitly connected to the embedded block and the same signal combination can be achieved with implicit connection as shown by the second example:



Note



If the text does not fit in the default size text box on a block diagram, the additional text is indicated by the word <<.. more ..>>. This text can be displayed by resizing the box.

You can hide the HDL text box on a block diagram by selecting the text and choosing **Hide Text** from the popup menu or re-display the text by choosing **Show Text** when the embedded block is selected.

You can also change the shape of an embedded block. For example, an embedded HDL text view which performs a logical OR or AND function can be represented by a corresponding logic shape.

Refer to “[Logic Shape Notation](#)” on page 227 for information about changing the shape of an embedded block.

Adding Ports on a Block Diagram

When you add an external interface *port* on a block diagram, the cursor changes to a cross-hair which allows you to add the port by clicking to drop the port at the required location on the diagram.

Note



Buffer ports are not available when the hardware description language is set to Verilog.

A port represents a connection for signal paths in or out of the block diagram and can be connected to a signal or bus. An unconnected port has no name, but when connected it is identified by the name of the connected signal or bus and this name is used as the port name on the default symbol for the block diagram.

If a port is added over the dangling *net connector* on an existing signal or bus, it is automatically connected to the net.

Adding Ports to Existing Nets

You can also add external interface ports to dangling net connectors by choosing **Add Port I/O** from the popup menu (or the **Signals** cascade of the **Diagram** menu) when one or more nets are selected. Appropriate ports are added for the signal properties. For example, an output port is added to an output signal or a buffer port to buffered signal. If both ends of a net are unconnected, an input port is connected to the source end.

Adding Ports from a Component

You can add external interface ports connected to signal stubs from the ports on a component by choosing **Add PortIO** from the **Diagram** or popup menu in a block diagram.

If the component is selected, the Add Signal Stubs dialog box allows you to choose whether to add signal stubs on Input, Output, InOut or Buffer (VHDL only) ports. Signal stubs are added to all unconnected ports of the specified type.

If one or more component ports are selected on a block diagram, signal stubs with ports are added to the selected ports and any unselected ports are left unconnected.

The signal stubs are added with properties corresponding to the ports defined on the symbol for the component. If the component has a different language from the parent view (for example, a Verilog component instantiated in a VHDL view) the properties are automatically mapped to the language of the parent view.

The signal stubs are implicitly connected by name to signals with the same name on the view and you are warned if any of the signal names already exist on the diagram.

Changing the Mode of a Port

When one or more ports are selected on a block diagram or in the symbol editor, the popup menu and the **Ports** cascade of the **Diagram** menu include a **Change Mode** cascade which allows you to change the mode (**In**, **Out**, **InOut** or **Buffer**) of the ports.

This option can be used for external ports defining the interface or ports on a block which define the interface to a child view.

Note



If you change the mode of a port to or from an input port, it is rotated automatically by 180 degrees.


Rotating a Port

When an external interface port on a block diagram is selected, the **Edit** and popup menus include a **Rotate Port** cascade which allows you to rotate the port clockwise by 90, 180 or 270 degrees.

Rotating Signal Names

You can rotate a signal name (or a port name in the symbol editor) by choosing **Rotate Text** from the popup menu to rotate the text clockwise by **90**, **180** or **270** degrees.

Adding a Global Connector on a Block Diagram

You can add a *global connector* on a block diagram using the  button or **G** shortcut key or by choosing **Global Connector** from the **Add** menu.

The cursor changes to a cross-hair which allows you to add the global connector by clicking at the required location on the diagram. A global connector represents a common connection to all blocks in the diagram. Any number of signals, buses or bundles can be connected to the same global connector. This does not imply any connection between them.

Note

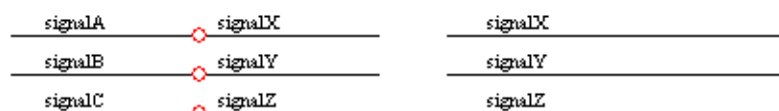


A global connector connects the attached net (or nets) to all blocks on the diagram but does not connect to components or ModuleWare instances.

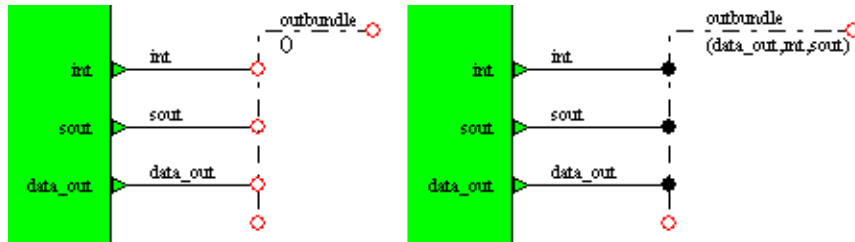
Connecting Overlapping Nets

A net can be individually connected to another net or other object by dragging with the **Left** mouse button over the net or object. You can also make a connection to one or more nets by overlapping the dangling net connectors and choosing **Connect** from the **Diagram** or popup menus.

When you connect nets in this way, the new net adopts the properties of the selected net. For example, the following nets become *signalX*, *signalY* and *signalZ* if the **Connect** command is used when *signalX*, *signalY* and *signalZ* are selected.

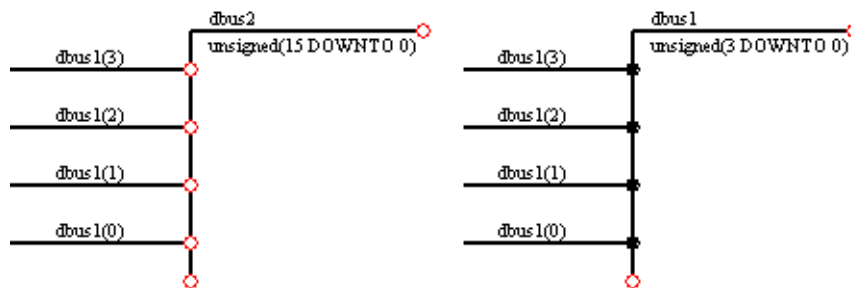


You can connect dangling net connectors to a bundle by drawing a bundle to overlap the net connectors and then choosing **Connect**. For example, after adding signal stubs to a component you can group the output signals into a single bundle for easy routing to another area of the diagram.



Similarly, if the dangling connectors represent slices of a bus, they can be connected to a single bus although the net properties may need to be edited to ensure that all connected net segments have the same name.

For example, in the following picture, *dbus2* has been connected and then edited to have the same name (and compatible range) as the slices *dbus1(0)* to *dbus1(3)*.



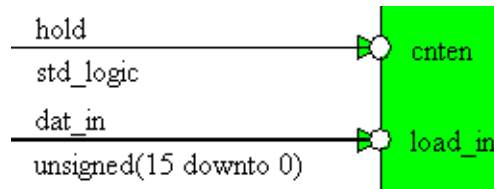
Connecting Nets to a Block or Component

Signals or buses can be individually connected to a block or component by dragging with the **Left** mouse button over the body of a block or over an existing port or port map frame on a component.

You can also make a connection by overlapping a block or component with the dangling net connectors on the end of existing signals or buses and then choosing **Connect** from the **Diagram** or **popup** menus.

This can be useful when you have created a new child block diagram and want to add a block or component connected to the nets created when the child diagram is initialized.

When you are using this method to connect a component port, the port must be fully overlapping as shown below:



VHDL Port Mapping

If a signal has no slice or element, then the port has the type and range (if any) of the signal. All other declarations are the same except that initial values on signals are not propagated to the port. The type and range are different if slices or elements are used.

If the signal has a slice, then the port has the type and the range of the slice. However, the type must be one of the following:

<code>std_logic_vector</code>	<code>integer</code>
<code>std_ulogic_vector</code>	<code>natural</code>
<code>bit_vector</code>	<code>positive</code>
<code>signed</code>	<code>real</code>
<code>unsigned</code>	<code>string</code>

If not one of these types, for example, a subtype defined in a VHDL package such as *slv7d0* IS *std_logic_vector(7 DOWNT0 0)*, an error is issued. To use such a type, you should either change the block to a component or assign the slice to an intermediate signal.

If the signal is an element of a bus, and the bus array type is one of those in the following list, then the corresponding port scalar type is used:

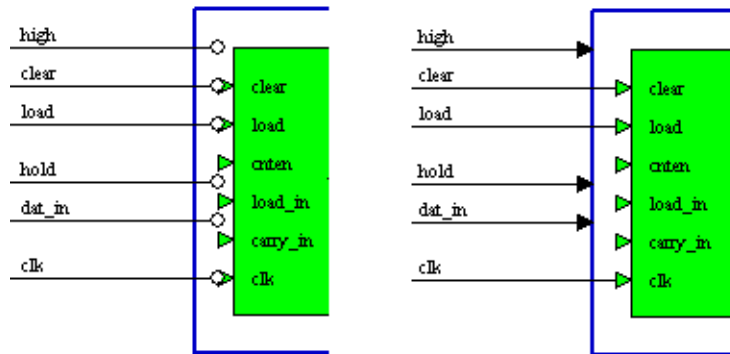
Bus Array Type	Port Scalar Type
<code>std_logic_vector</code>	<code>std_logic</code>
<code>std_ulogic_vector</code>	<code>std_ulogic</code>
<code>bit_vector</code>	<code>bit</code>
<code>signed</code>	<code>std_logic</code>
<code>unsigned</code>	<code>std_logic</code>
<code>integer</code>	<code>integer</code>
<code>natural</code>	<code>natural</code>
<code>positive</code>	<code>positive</code>
<code>real</code>	<code>real</code>
<code>string</code>	<code>character</code>

If not one of the above types, an error is issued and you should either change the block to a component or assign the element to an intermediate signal.

A signal is regarded as an element if the slice bounds is specified in a single field and is either an integer or a valid identifier. For two dimensional arrays, each array is treated accordingly, whether a slice or an element.

Connecting Nets to a Port Map Frame

If a port map frame is enabled and the nets or the component body are selected, then overlapping net connectors are connected to the frame unless they also overlap a port with compatible properties on the body of the component.



Note





If the port map frame is selected, all overlapping nets are connected to the frame.

Highlighting a Net on a Block Diagram

Any set of signals or buses which are connected explicitly or implicitly connected by name form a *net*.

You can highlight an entire net by selecting any net segment, port declaration or signal declaration on the block diagram and choosing **Highlight Net** from the popup menu or from the **Signals** cascade of the **Diagram** menu.

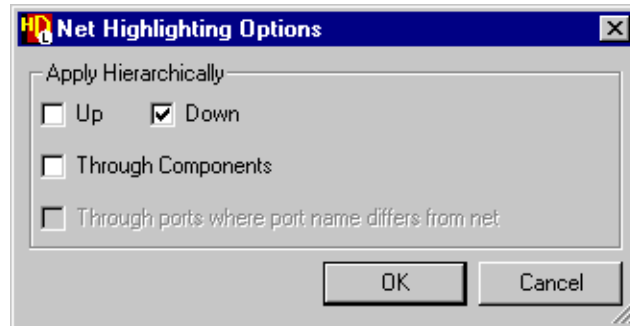
A cascade menu allows you to highlight all routes in the selected nets for a **Single Level** block diagram or to highlight the nets in any open **Hierarchical** diagram. These options are also available using the  and  buttons.

If multiple nets are selected, four different highlight colors are used. However, the colors may be repeated when more than four nets are selected and bundles are always highlighted in red.

If you choose the single level option, all segments on the active diagram in the selected net are highlighted even if they are connected only by name including any bundles which contain the net.

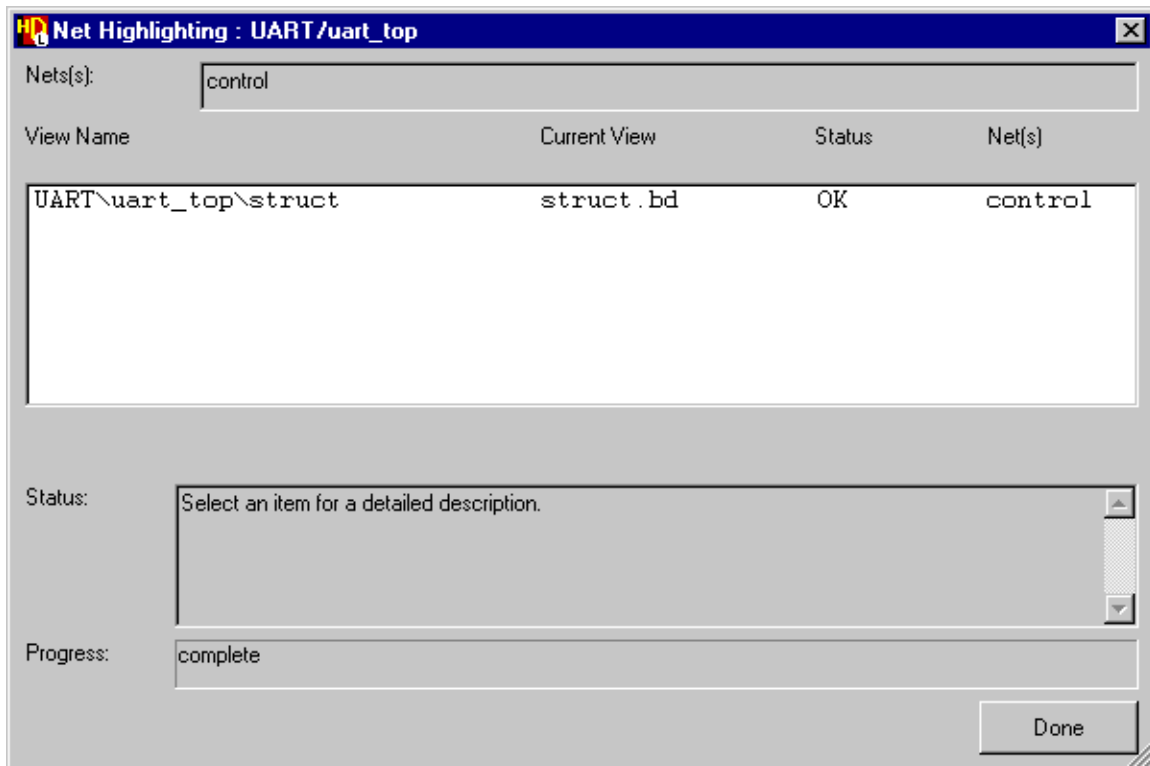
If a bundle is selected, any nets contained in the bundle are highlighted even if they are not graphically connected to the bundle.

If you choose the Hierarchical option, the Net Highlighting Options dialog box is displayed.



You can choose whether the highlighting is applied **Up** and **Down** the hierarchy and whether it should be traced **Through Components**. If you choose to highlight through components, an extra option controls whether the highlighted net is traced **Through ports where the port name differs from the net**.


When you confirm your choices in the Net Highlighting Options dialog box, the net is highlighted on the active diagram and a progress window is displayed which lists all the occurrences of the net in the specified hierarchy.



The progress is indicated in the dialog box and a complete indicator displayed when all views in the hierarchy have been traversed.

If any errors or warnings are encountered, these are indicated in the Status column and the full message appears in the Status box when the view name is selected.

You can display any view listed in the preview window by double-clicking on its name in the preview window. If the view is a block diagram, all occurrences of the net are highlighted and the window is zoomed to view the entire net.

You can clear all net highlighting by using the  button or by choosing **Clear All** from the menu.



Logic Shape Notation

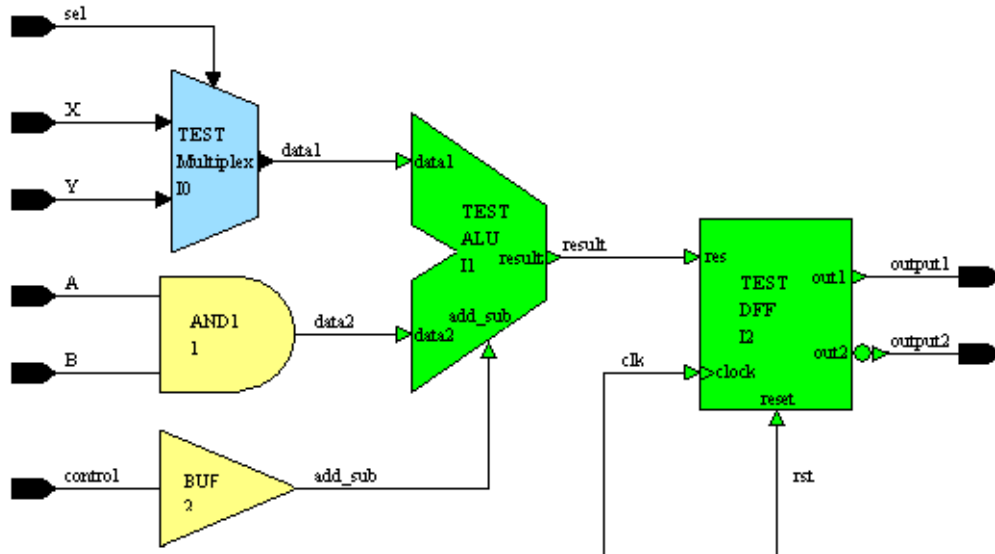
The standard block diagram notation uses simple rectangular shapes for blocks, embedded blocks and components connected by directed lines representing signals, buses or bundles.

This notation allows the designer to focus on the high level behavior of the design without prematurely committing to implementation details. However, there are many cases particularly in low level design or where existing components are being re-used when it is useful to use gate-level logic notation.

You can choose from a set of alternative standard shapes representing standard data path functions including multiplexer, buffer, AND, OR, XOR, arithmetic logic unit, ground, supply, pulldown and pullup.

These shapes can be used to represent inverted logic (including NAND, NOR, NXOR) by adding an active low (Not) indicator to the port or you can add a clock indicator to an edge-triggered input port.

For example, clock  and active low indicators  have been applied to the *clock* and *out2* ports on the *DFF* component in the picture below.



The shapes can also be applied to any block or embedded block and you can choose to hide the block ports when the flow is implied by using these logical shapes. For example, the ports have been hidden on the embedded blocks *AND1* and *BUF* in the picture.

Changing the Shape of a Block or Component

You can change the shape of the selected block, embedded block or component instance on a block diagram by choosing the **Shape** cascade of the **Diagram** or popup menu.



Note

Note that you can create a custom shape for a component symbol which will be applied to all instances of the component by using the symbol editor as described in [“Customizing a Symbol”](#) on page 317.

You can check **Edit Shape** from the **Shape** cascade menu to edit the existing shape directly or choose **Autoshapes** to choose from a palette of standard shapes.

If you choose **Edit Shape**, the block, embedded block or component is replaced by a resizable rectangular boundary enclosing the default shape. A customized shape can be created by adding comment graphics within this boundary.

The default shape can be deleted or you can superimpose any number of comment graphics objects. Alternatively, you can use the **Edit Vertices** command from the popup menu to modify shapes.

Any comment graphics or comment text added completely within the boundary is considered part of the customized shape. The block boundary is drawn in red while in edit mode and existing comment graphics or text can be moved inside the boundary.

When you uncheck **Edit Shape** in the menu, the boundary is redrawn in gray and all comment graphics or text within the boundary become part of the customized shape.

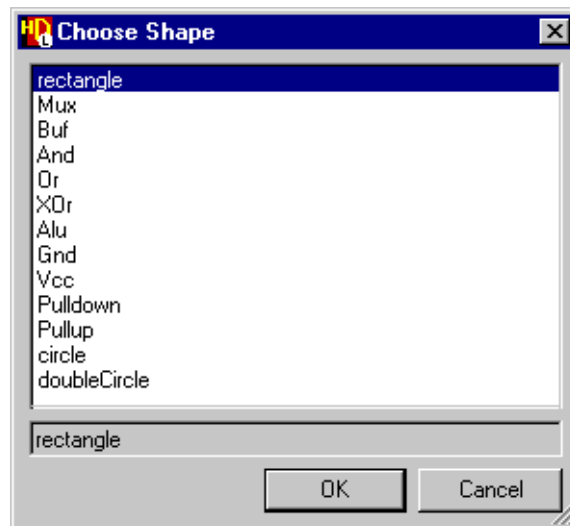
The cascade menu also includes an option to add autowhiskers. If **Autowhisker** is checked in the menu, whiskers are automatically added as orthogonal lines between the ports on the block boundary and the contained comment graphics shape. However, no whiskers are added if the line would not intersect with a graphics object.

Choosing a Standard Shape

You can apply a standard shape to the selected block, embedded block or component by choosing **Autoshapes** from the **Shape** cascade of the **Diagram** or popup menu or by using the **Change Shape** button in the block diagram Object Properties dialog box.











You can change the shape of a component symbol which will be applied to all instances of the component by choosing **Autoshapes** from the **Diagram** or popup menu when the body of the symbol is selected in the symbol editor or by using the **Change Shape** button in the **Symbol** tab of the Object Properties dialog box.


The Choose Shape dialog box allows you to change the default rectangle shape.



The following alternative standard shapes are available:

Name	Shape	Function	Name	Shape	Function
Mux		Multiplexer	Gnd		Ground

Buf		Buffer	Vcc		Supply
And		AND gate	Pd		Pull down
Or		OR gate	Pu		Pull up
XOr		Exclusive OR gate	circle		Circle
Alu		Arithmetic logic unit	doubleCircle		Double circle

You can also apply these shapes using a pulldown palette accessed from the  button in the Appearance toolbar.

You can use these shapes to represent other logic functions (such as NAND, NOR and NXOR gates) by setting an active low indicator for any selected port.

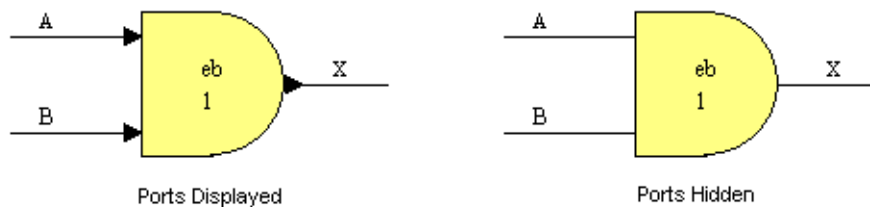
For example, an invertor can be represented by a buffer shape with a Not port:



Hiding Ports on a Block or Component

Ports are normally added automatically when a signal is connected to a block or embedded block. However, you can clear the **Show ports when connected** option in the **Blocks** or **Embedded Blocks** tab of the Object Properties dialog box to hide the ports. This can be useful when you have changed the shape of a block to represent a standard logic function.

For example, the following picture shows an how an embedded block representing a logical AND function would be displayed with the ports displayed or hidden:




You can also choose whether connected ports are displayed on a component by setting the **Show ports when connected** option in the **Symbol** tab of the Object Properties dialog box in the symbol editor.

Note


Unconnected component ports are always displayed.

Indicating Not or Clocked Ports

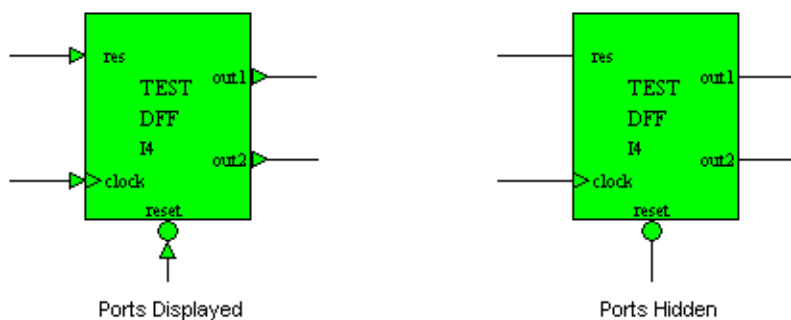
You can choose whether a selected port (or ports) on a block or embedded block in a block diagram is shown with active low (Not) polarity (indicated by ) by choosing **On** from the **Not** cascade in the popup menu (or **Ports** cascade of the **Diagram** menu) when the port is selected. Choose **Off** from the cascade menu to set active high polarity and remove the indicator.

If a port on a ModuleWare component which supports polarity control is selected, you can choose **Active High** or **Active Low** from the **Port Type** cascade of the popup menu or you can set the port type parameter in the ModuleWare Parameters dialog box. Many of the ModuleWare components also provide synchronous (**Sync**) and asynchronous (**Async**) polarity and **Rising** or **Falling** clock options.

You cannot directly change a regular component port but you can set an active low indicator by using the popup menu (or **Ports** cascade of the **Diagram** menu) in the symbol editor.

You can also choose whether a selected input port (or ports) is shown as an edge triggered clock signal (indicated by ) by choosing **On** from the **Clock** cascade in the **Diagram** or popup menu.

You can also choose to hide connected ports when you want to represent a standard data path logic function. For example, the following picture shows a component representing a flip-flop with an edge triggered clock and active low reset:



Setting Block Diagram Preferences

You can set *block diagram* preferences by choosing **Structural Diagram** from the **Master Preferences** cascade of the **Options** menu in the *design manager* to display the Structural Diagram Master Preferences dialog box.

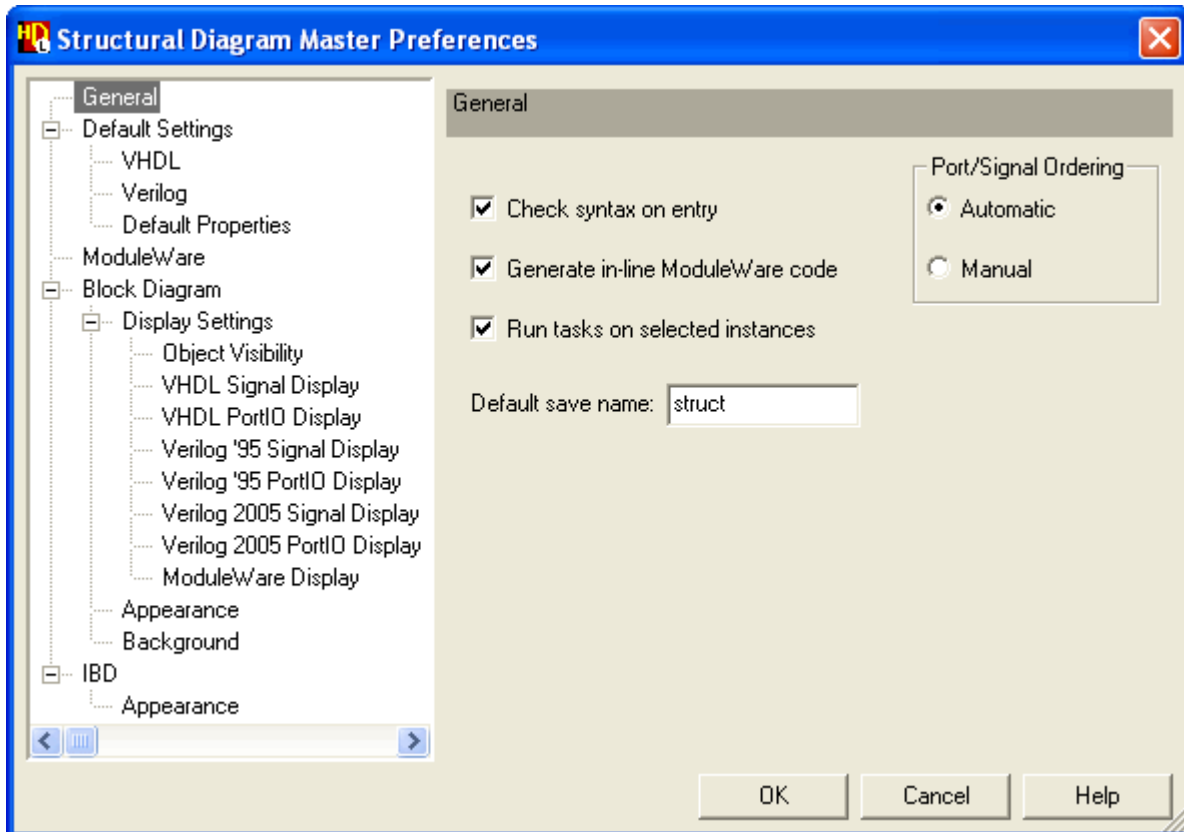
The dialog box has separate pages for setting **General** preferences, **Default Settings**, **ModuleWare** preferences, **Block Diagram** preferences (which include **Display Settings**, **Appearance** and **Background** preferences), in addition to **IBD** preferences.

You can set the block diagram appearance, display setting and background preferences (but not default values) for the active diagram by choosing **Diagram Preferences** from the **Options** menu in the block diagram editor to display the Block Diagram Preferences dialog box.

When you edit these preferences for the active diagram, the dialog box allows you to choose whether the preferences are applied to new objects or to both new and existing objects in the diagram.

Refer to “[Setting Diagram Master Preferences](#)” on page 49 for information about applying master preferences to existing views and updating the master preferences from the active view.

The **General** page allows you to set general preferences for the block diagram. You can set an option to check the HDL syntax of declarations and embedded HDL text objects on entry. Checks are also performed for unsynthesizable constructs if the common synthesis checks option is set in the **Checks** tab of the Main Settings dialog box.



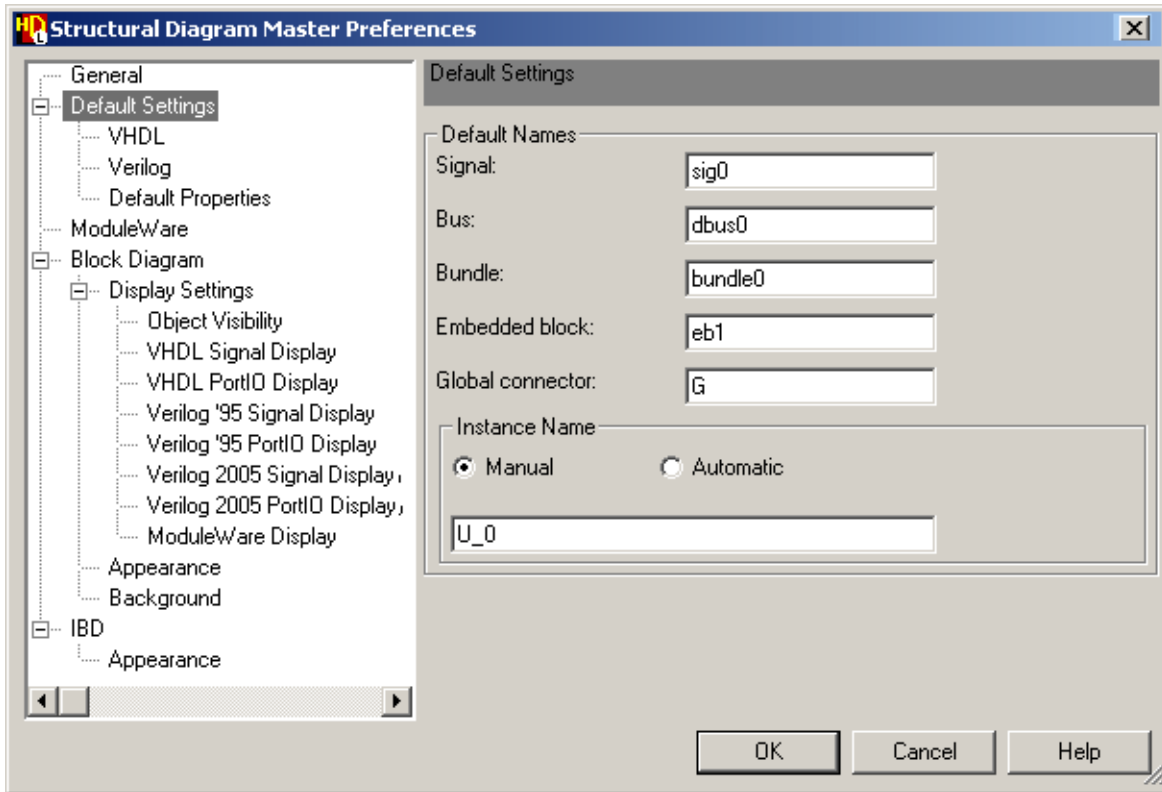
You can choose whether in-line or separate HDL code is generated by default for any instantiated ModuleWare components.

You can control whether the tasks are invoked on the selected instances or on the diagram itself.

You can specify the default save name used for new block diagrams.

You can also specify whether port declaration ordering is automatic (by mode and alphanumeric name) or if manual ordering is allowed.

The **Default Settings** page allows you to set the default names used for the signal, bus and bundle name, embedded block name and global connector name.



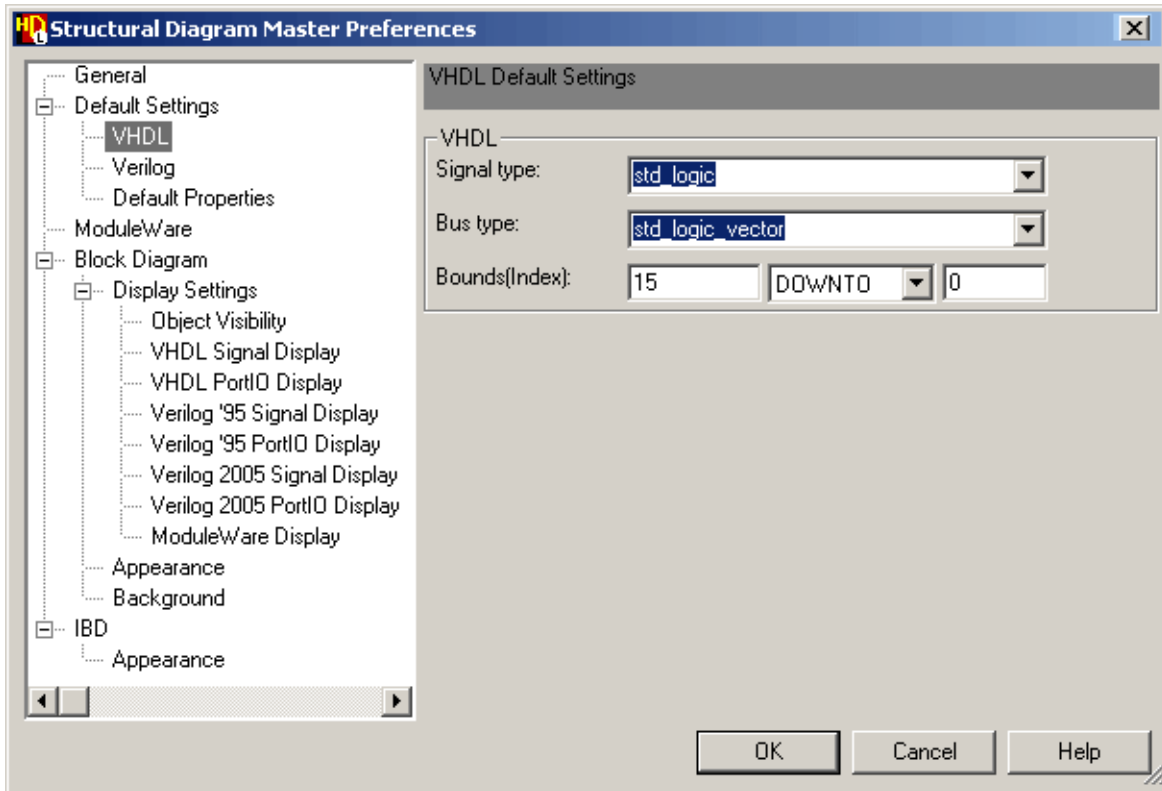
You can also choose whether the instance name for a block or component is derived from a specified root name or from the design unit name.

When set to **Manual**, the specified string is used as the root of the instance name. The default name can be changed or the instance name can be edited on a block diagram or IBD view and replaced by any other valid HDL identifier.

The internal variable `%(unit)` can be included to insert the design unit name and the instance name can be edited on a block diagram or IBD view.

When set to **Automatic**, the internal variable `%(unit)` is used. The default name can be modified but must always include the `%(unit)` variable. The variable is automatically replaced by the design unit name and the instance name cannot be edited on a block diagram or IBD view.

Separate **VHDL** and **Verilog** sub-pages can be used to set default types, constraints and bounds for VHDL and Verilog signals. For example, the following picture shows the VHDL sub-page:

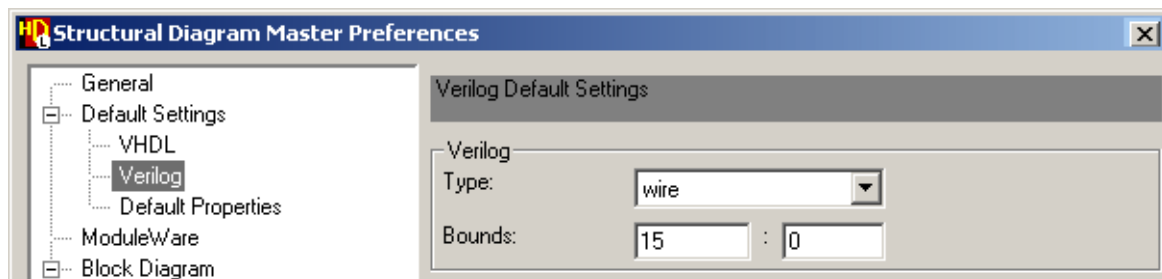


The bounds can be entered as a range (for example, *15 DOWNTO 0* or *0 TO 75* in VHDL or *0:7* in Verilog) or you can use the first bounds entry box as an index for a single element in an array leaving the second entry empty.

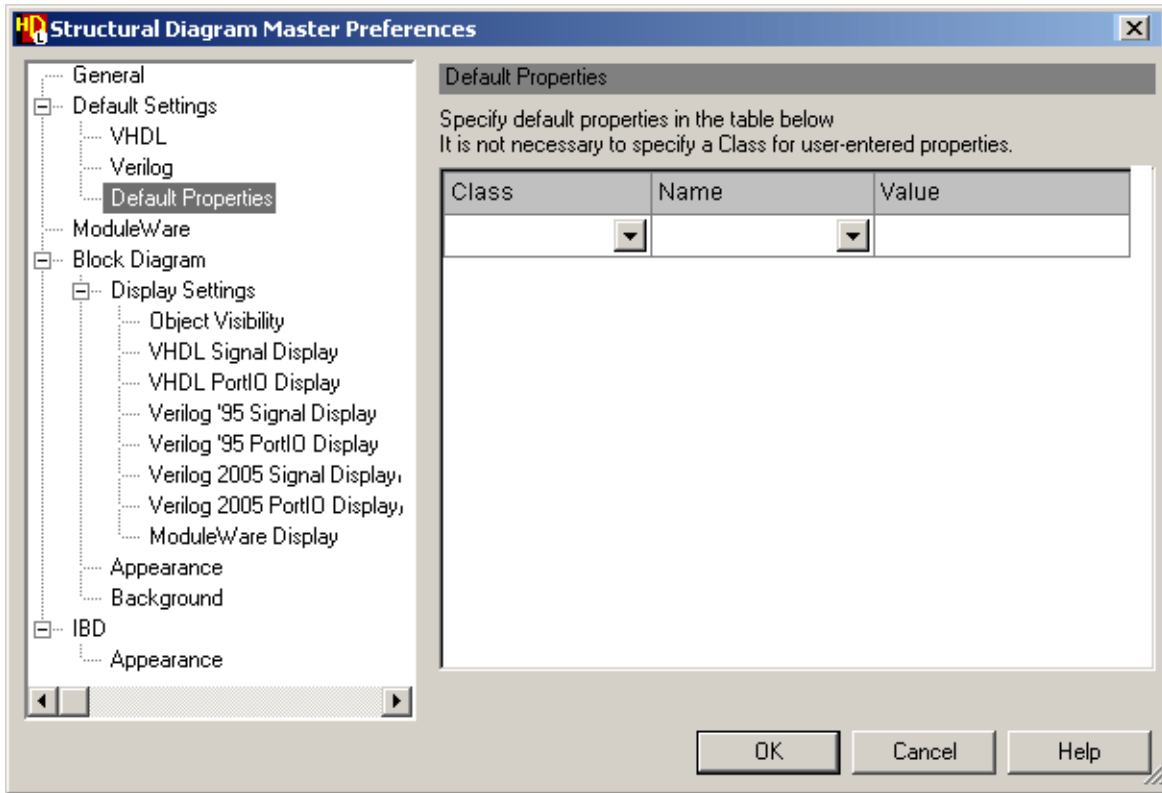
Note that for VHDL, you can use the first bounds entry box to enter a user specified constraint such as an enumerated or integer type name or you can enter an array name or type of the form:

`<array>'RANGE` or `<array>'REVERSE_RANGE`

The following picture shows the options on the Verilog sub-page



You can use the **Default Properties** sub-page to define default user properties for block diagram views.



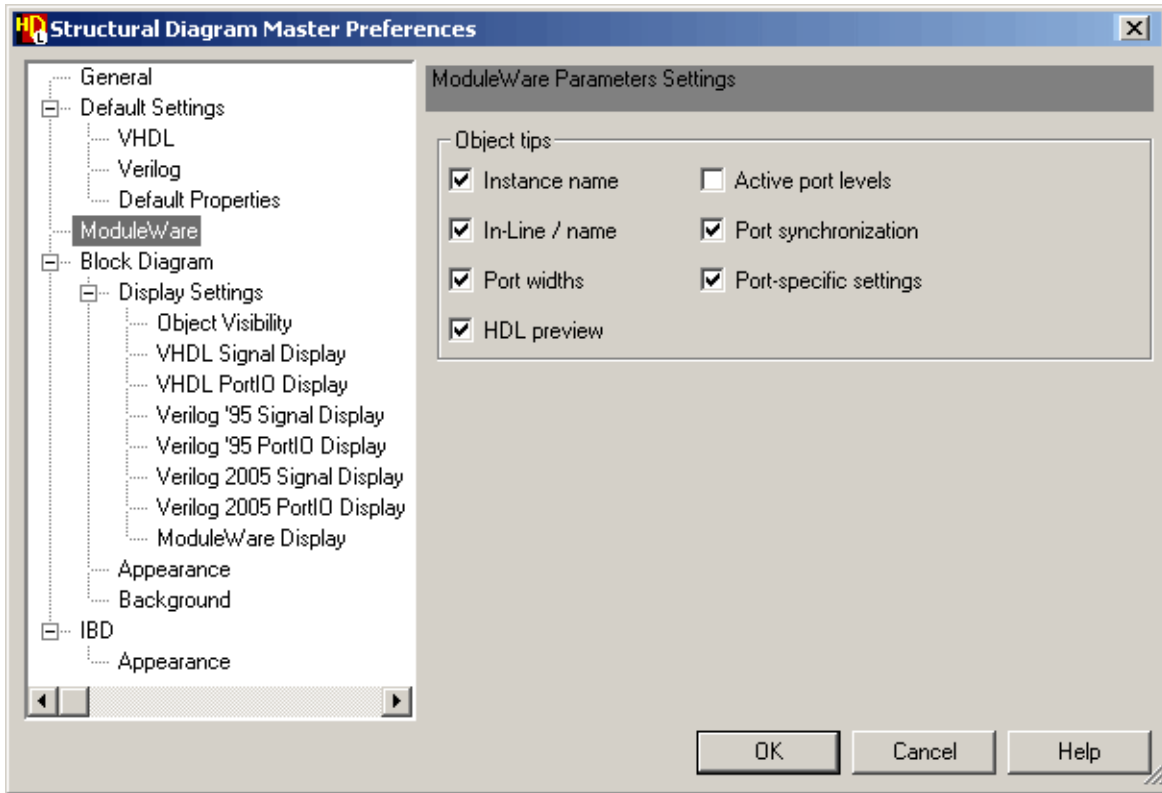
Refer to the [HDL Designer Series User Manual](#) for information about “Using View Property Variables”.

Note



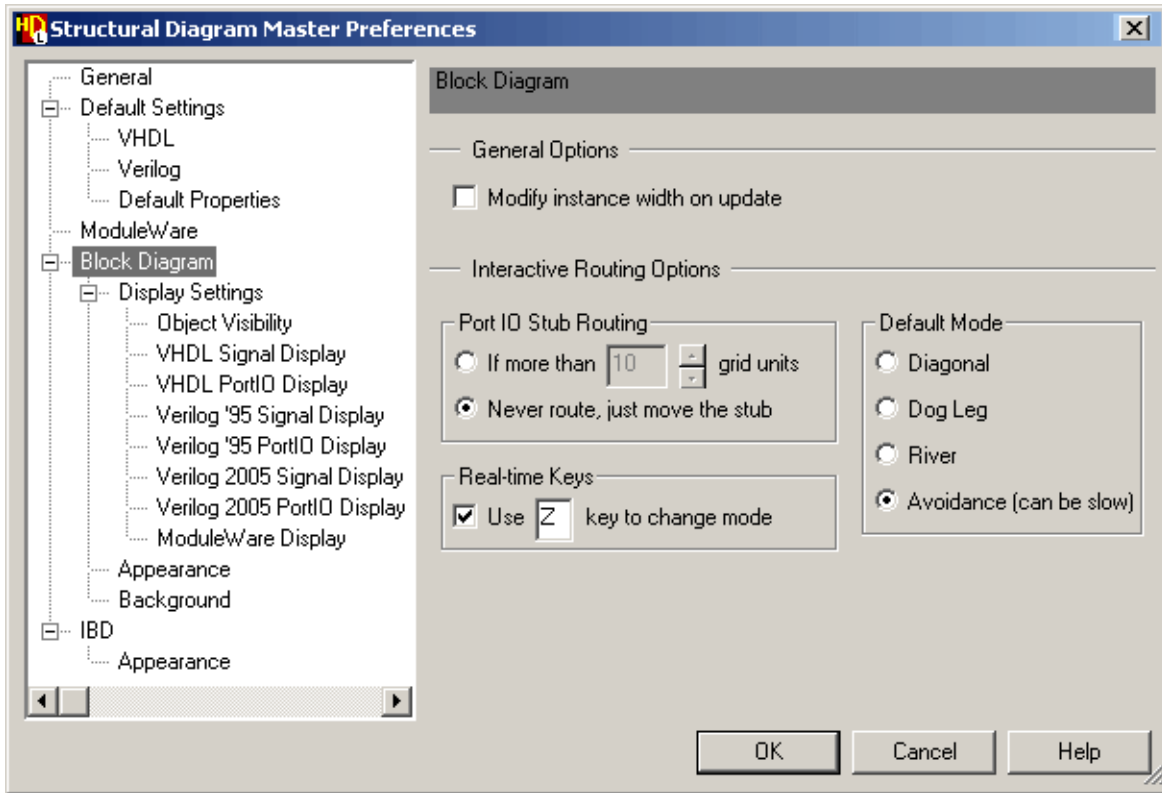
The default values take effect on the next block diagram you open and can only be edited from the **Master Preferences** cascade in the [design manager Options](#) menu.

The **ModuleWare** page allows you to set the default visibility for the content of the *object tip* displayed when the cursor is over a *ModuleWare* instance in a *block diagram* or *IBD view*.



Tip: When these settings are accessed from the editor, you can choose whether to apply the display settings to new objects only or to all new and existing objects on the diagram.

The **Block Diagram** page enables you to set general block diagram options in addition to routing preferences.



As part of the Block Diagram General Options, you can choose to automatically modify the width of an instance in a Block Diagram, on using the Update Interface option. This is applicable when you have added a port with a long name to a Symbol and then saved; subsequently, on right-clicking on the symbol's corresponding instance in the Block Diagram and choosing **Update Interface** from the popup menu, the width of the component is increased to accommodate the width of the added port name.

On the other hand, if the option “Modify instance width on update” is not selected, then on updating the interface, the component width shall not be increased to contain the long port name.

Note



You must set the Port Name option in the Port Visibility settings of the component, in order to have the name text visible on the component ports.

In the Interactive Routing Options section, you have the ability to set the default routing method as well as the stub routing options as follows:

1. You can set the Default Mode for net routing as Diagonal, Dog Leg, River, or Avoidance. The default routing mode is Avoidance.

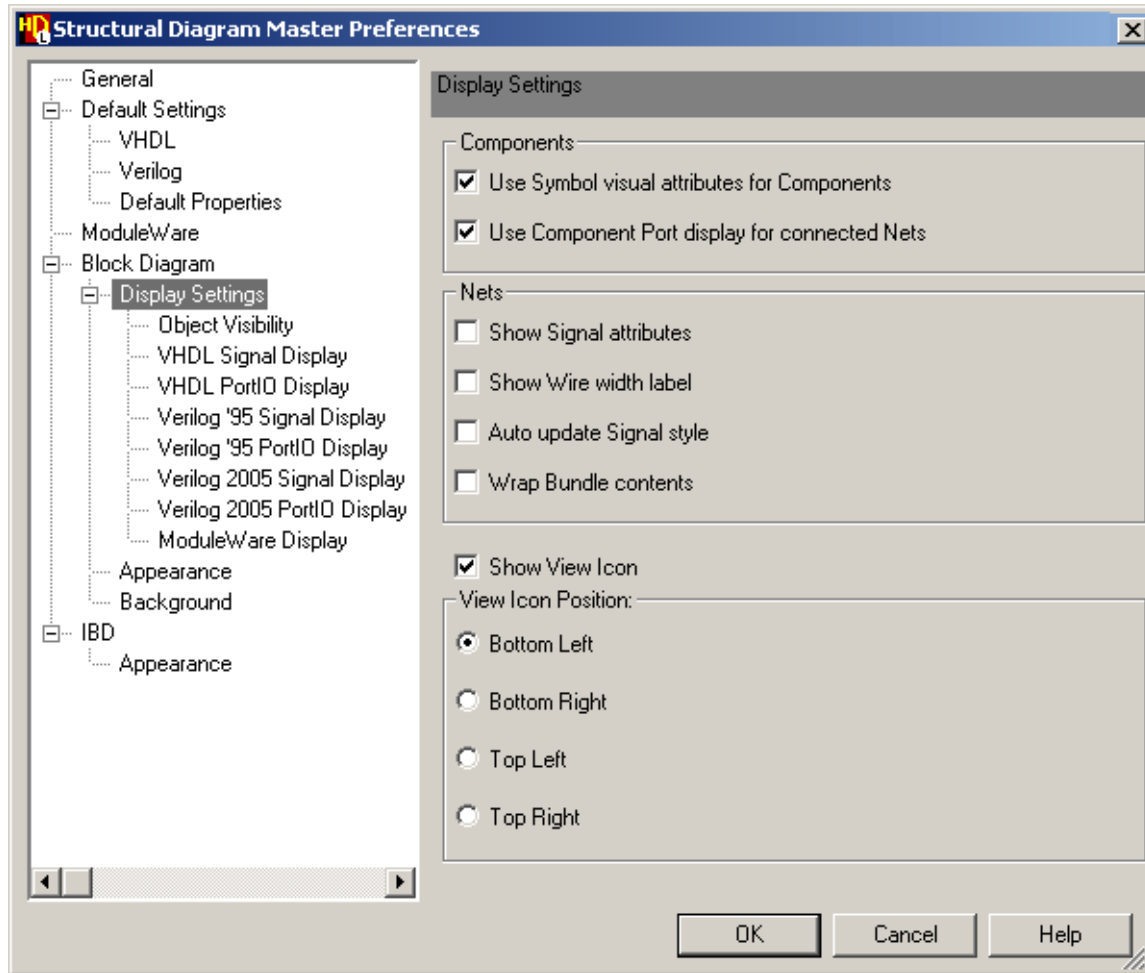
- **Diagonal:** allows routing in a straight-line mode.
- **Dog Leg:** allows routing in an angular mode with the ability to create overlapping between nets.
- **River:** allows routing in an angular mode with the ability to create net loops.
- **Avoidance:** does not allow any cross-overs of objects; that is, this mode avoids the overlapping of any existing objects.

Through the Real-time Keys section, you can specify a key through which you can toggle between the above mentioned routing modes while routing (by pressing the key you specified).

2. In the Port IO Stub Routing section, you can choose one of two options. First, you can allow stub routing only if the stub exceeds a specific length of your definition; that is to say, if the stub length is more than the number of grid units that you define, only then it will be routed.

On the other hand, you can choose not to route the stub, but rather to re-locate it directly on moving its relevant object.

The **Display Settings** sub-page contains options which control how objects are displayed on a diagram.



You can choose whether to use the symbol visual attributes when a symbol is instantiated in a block diagram as a component and whether to take display settings from component ports. This option makes the visibility of properties on a new net the same as the visibility of properties on the connected component port. Otherwise, the default signal display properties are used.

You can choose whether attributes added to port and signal net declarations are shown on the block diagram. You can choose whether to display an optional net width label and whether the signal style (signal or bus) is automatically updated when you change the bounds of a net. You can also choose to wrap the contents of bundles and show each signal on a single line or display all the signals on the same line below the bundle name.

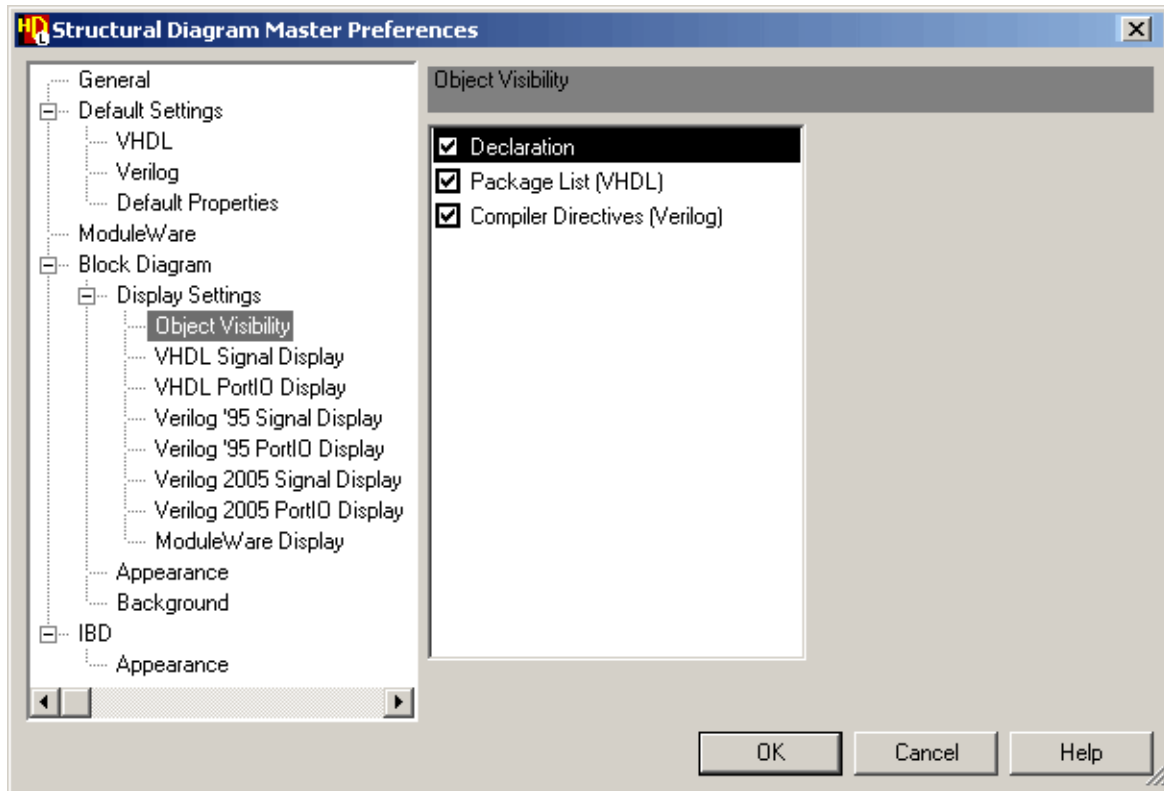
You can also enable or disable an icon which indicates the default view of a block or component and specify where this icon is displayed on the instance.

Note



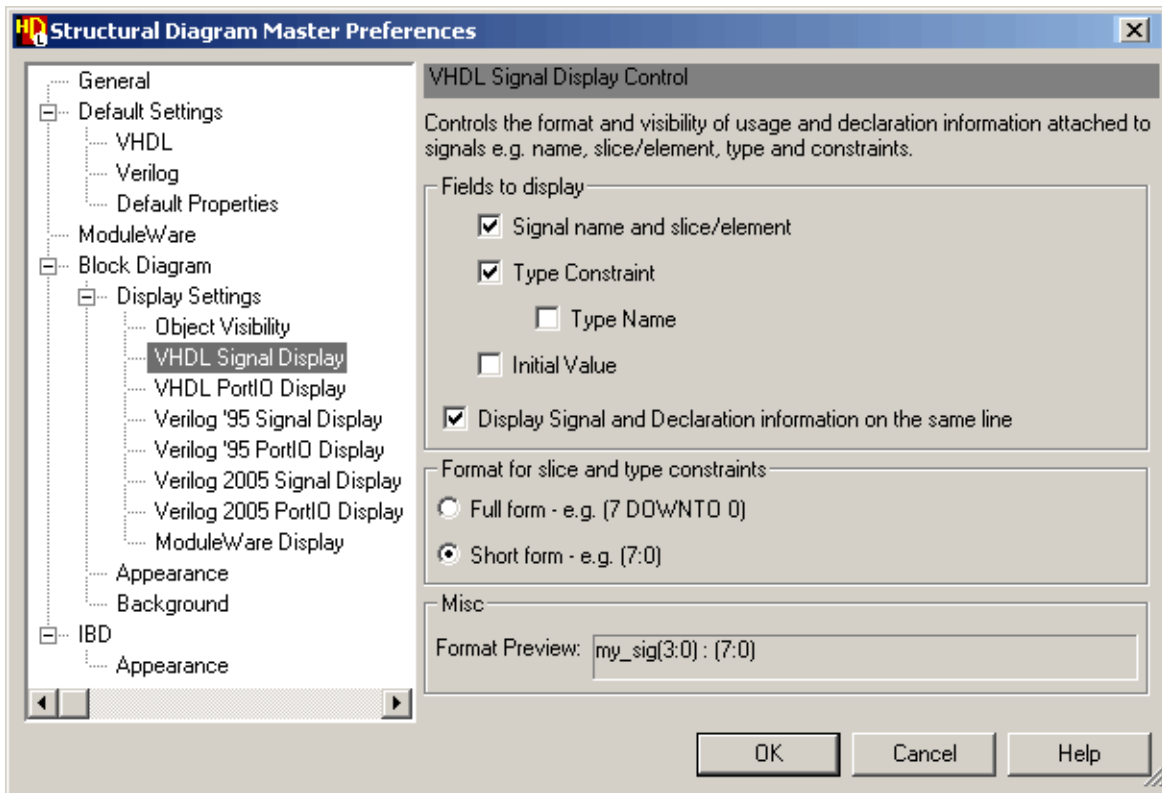
Many of the display settings can only be set in the master preferences and are permanently dimmed when the dialog box is accessed from the editor.

An **Object Visibility** sub-page is available in the Structural Diagram Master Preferences dialog box which allows you to set the default object visibility for multi-line text objects on the diagram.



Refer to [“Changing Text Visibility”](#) on page 68 for more information about the objects which can be displayed or hidden on a block diagram.

Separate **VHDL Signal**, **Verilog Signal**, **Verilog 2005 Signal**, **VHDL PortIO**, **Verilog PortIO** or **Verilog 2005** sub-pages can be opened to set the default display properties for signal nets and external ports. For example, the following picture shows the VHDL Signals page:



Note

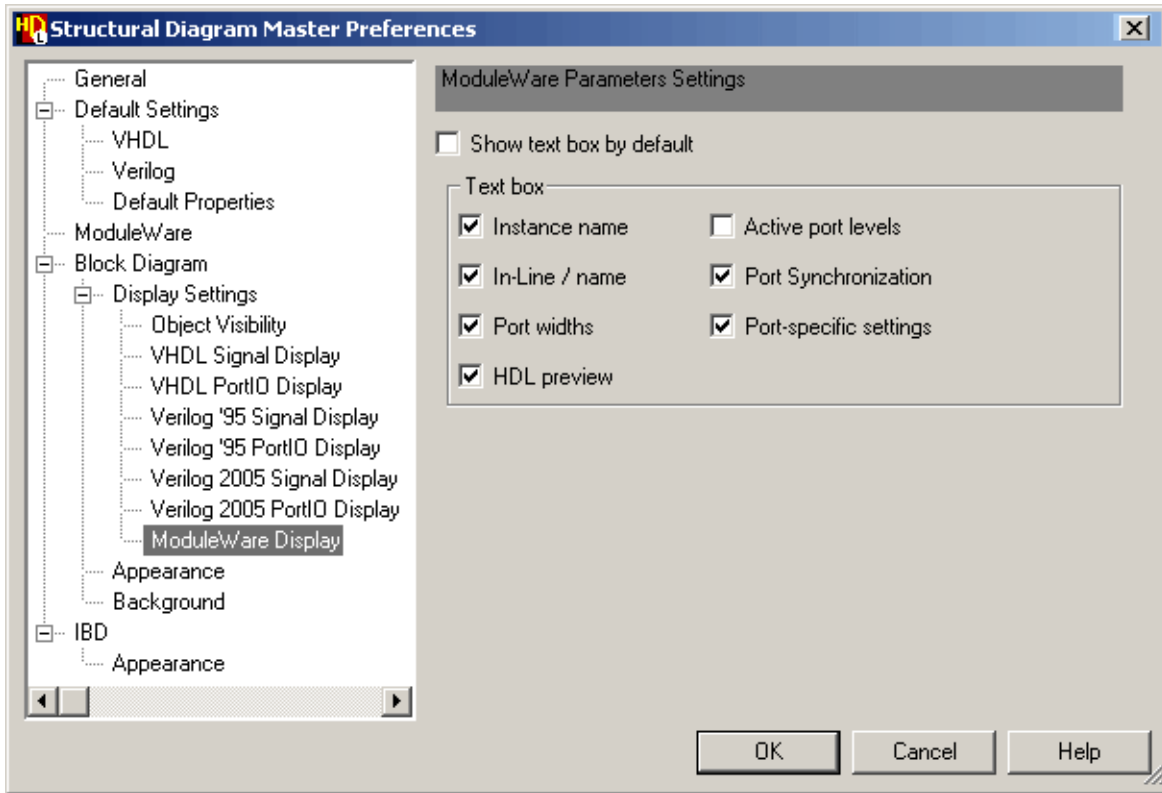


The display properties can be set separately for VHDL and Verilog when you edit the master preferences or for the language used by the active diagram when accessed from the editor. When accessed from the editor, you can choose whether to apply the signal and port display changes to new signals and ports only or to all new and existing signals and ports on the diagram.

Refer to [“Changing the Display of Signal Properties”](#) on page 208 and [“Changing the Display of Port Properties”](#) on page 206 for more information about the signal and port properties.

A **ModuleWare Display** sub-page allows you to set the visibility of the ModuleWare parameters that can be displayed as *comment text* adjacent to a ModuleWare instance on a block

diagram. Note that the settings in the ModuleWare Display sub-page apply to block diagrams only, whereas the settings in the ModuleWare page apply to structural diagrams in general.



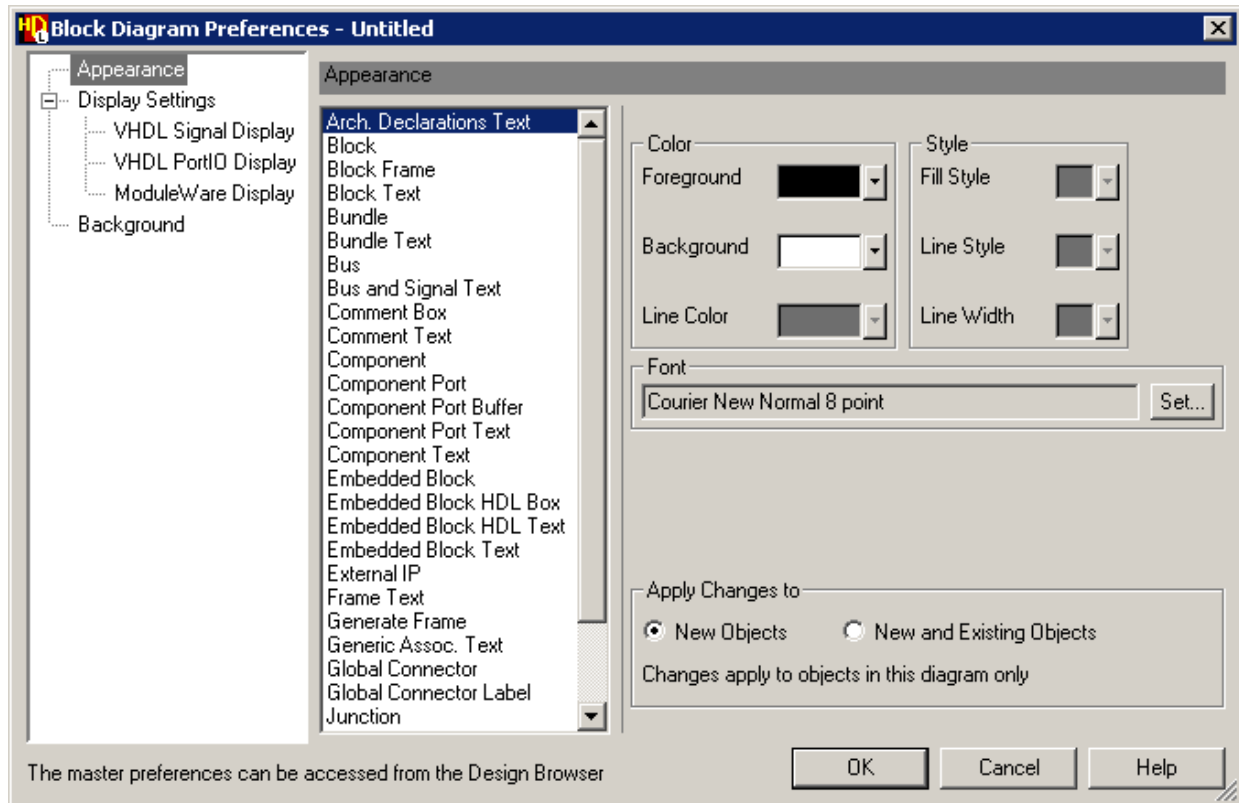
When this page is accessed from the Structural Diagram Master Preferences dialog box, you can choose whether the text box is displayed by default.



Tip: When accessed from the editor, you can choose whether to apply the display settings to new objects only or to all new and existing objects on the diagram.

Refer to [“Setting the Visibility of ModuleWare Parameters”](#) on page 120 for more information about displaying ModuleWare parameters.

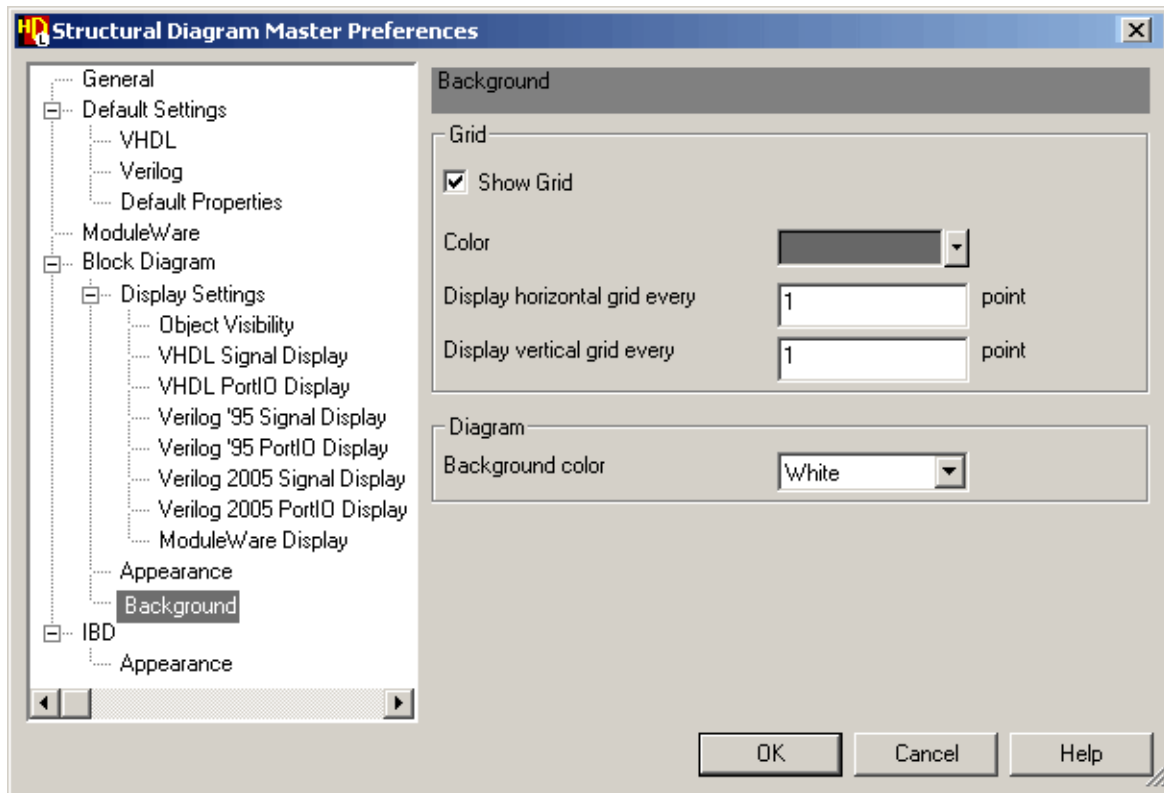
The **Appearance** page allows you to set visual attributes for individual block diagram objects.



These attributes include the foreground and background colors, line color, fill style, line style, line width and text font. Some attributes may not be available for all objects (for example, the line style, width and color attributes are not available for a text object).

Refer to [“Setting Visual Attributes”](#) on page 83 for more information about visual appearance attributes.

The **Background** page allows you to control the diagram background color and grid attributes used by the block diagram editor.



These preferences are described in [“Setting Background Preferences”](#) on page 50.

Refer to the “Default Preferences” appendix in the *HDL Designer Series User Manual* for lists of the default preferences set when you invoke a HDL Designer Series tool for the first time.

Chapter 5

IBD View Editor

This chapter describes how the structure of a design can be represented using a tabular IBD.

Refer to the [Block Diagram and IBD Views](#) chapter for information about procedures which are common to the block diagram and IBD view editors.

Interface-Based Design	246
New Design Creation Flow.....	246
Code Re_Use Flow	247
Design Assembly Flow	248
IBD Working Environment	250
IBD View Matrix	250
IBD View Toolbar	251
Getting Designs into IBD Editor.....	252
Working on a Previously Created HDS Design	252
Creating a New Design View	253
Adding Design Elements	253
Adding Components	254
Adding Blocks	255
Adding Embedded Blocks	255
Adding Nets	256
Adding Ports	257
Adding a Net Slice	257
Adding Generate Frames	257
Adding Requirements References.....	259
Connecting Design Elements.....	261
Connecting Nets to Component Ports.....	261
Mapping Expressions or Function Calls to Component Ports	263
Another Port_Centric_Connection Convention	265
Expanding and Collapsing IBD Views.....	266
Moving Rows and Columns in an IBD View.....	267
Sorting Rows and Columns in an IBD View	267
Grouping IBD Rows and Columns.....	267
Showing/Hiding Columns in an IBD View	267
Adding Bundles to your IBD view	268
Adding Net/Component Comments	269
Creating Filtered Views of the Design	269
Defining Filter Settings and Logic	270
Creating Persistent Subset Views of the Design.....	270

Pruning IBD Designs	272
Managing Design Hierarchy	274
Adding a Level of Hierarchy	274
Flattening Design Hierarchy	275
Generating HDL from IBD views	276
Controlling the Generated HDL Code	276
Enforcing Generation	278
Cross Referencing Generation Errors	278
Setting a Black Box for Synthesis	278
Documenting IBD Design Views	279
Creating Visualization Views	279
Exporting to HTML	280
Exporting to TSV or CSV Files	280
Setting IBD Preferences	281

Interface-Based Design

The Interface Based Design methodology represents the structure of a design by a tabular description of the interfaces between the lower level blocks, embedded blocks and components in a design. It enables you to manage the finest details of FPGA and ASIC SOC designs while still visualizing the big picture i.e you can see both the forest and the trees.

The methodology makes use of the editing and filtering mechanisms of tabular represented data thus giving you ultimate control over your design elements: Design blocks and nets are easily managed and organized: Specific areas and elements of the design can be fully explored and focused on: Connectivity problems can be easily detected.

The IBD editor is empowered with a synchronization feature by which you can smoothly move from an interface-based view to a block diagram view without losing any of your developed design features.

Whether you are using an old piece of code to build upon or you intend to specify a new design or even willing to assemble designs from diverse teams the simplicity and clarity of the IBD editor can make your job easy by following a series of defined steps.

IBD is tightly integrated with leading synthesis, analysis, verification and design creation solutions for advanced FPGA and ASIC design.

New Design Creation Flow

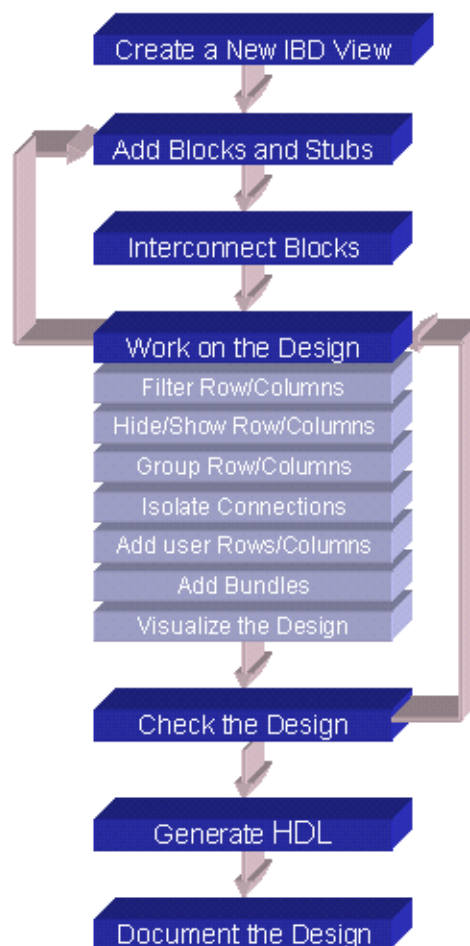
Starting a large design in the IBD editor can mean saving yourself a huge amount of time you would have otherwise wasted working on connecting design instances instead of focusing on design definition. The tabular representation of blocks and nets gives you visibility and control

over your design. The probability of wrong connections or undesired open ports leading to malfunction of the circuit and days of function debugging is aggressively reduced.

All you have to do is create a new graphical IBD view, add your blocks and stubs, interconnect your blocks then prune your design to focus on areas of interest. Once you are done, you can quickly check your design for any inconsistencies. A more complex design check can be configured through the invocation of the integrated DesignChecker tool.

On completing your design you are ready to generate HDL code. IBD gives you full control over the style, order and scope of the generated code.

The documentation options available act as an excellent mechanism by which designs can be further analyzed and reviewed.



Code Re_Use Flow

Starting a new project with an old piece of code can be a very challenging job. Entangling the design complexities, filtering elements of interest, detecting connectivity problems and

checking for certain constraints are all obstacles that may hinder and delay the actual design development process.

IBD offers a series of tools by which you can quickly and smoothly permeate the old design and fully understand it. Once you have done that you can then start customizing it to fulfill your requirements.

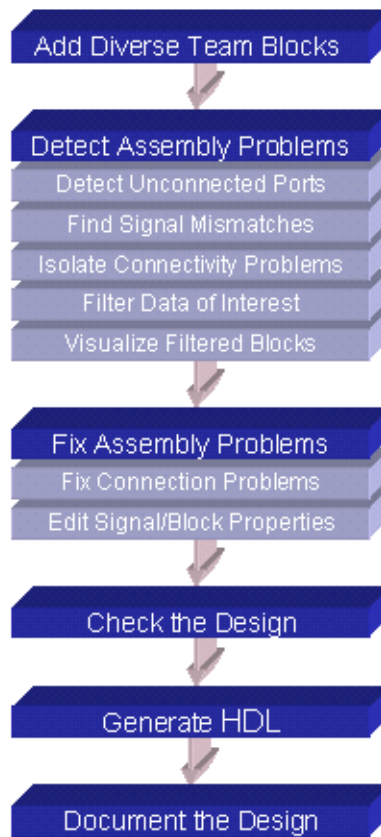
The illustration below defines a re-use flow approach that can be followed when dealing with previously developed designs.

Design Assembly Flow

Diversity of teams while being potentially beneficial or even in some cases a necessity due to the large size, complexity and sophistication of a project can sometimes lead to problems in the assembly stage.

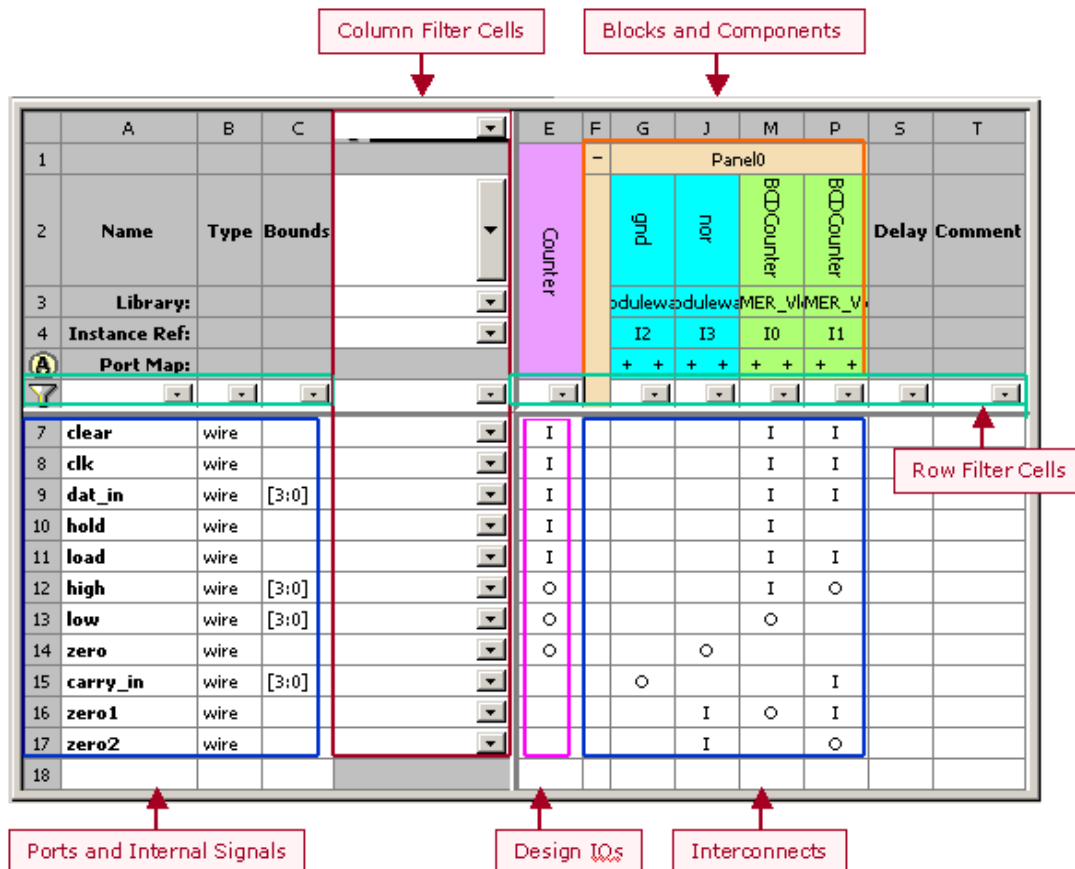
The probability of signal mismatches, connectivity problems, interface gaps in addition to the need to fully understand each area of the assembled design are all problems that can be tackled and resolved by the IBD editor.

IBD editor gives designers the tools to detect assembly problems as well as the tools to solve these problems.



IBD Working Environment

IBD View Matrix
























Net declarations are represented horizontally and are described by the Name, Slice, Type, Bounds, Delay, Initial and Comment columns.

Blocks, Embedded Blocks, Components and IPs are represented vertically and are described by the Name, Library, Instance Ref and Port Map (incase of components) rows.

The cells interconnecting net declarations with block, component and embedded block columns show the interface connections.


IBD View Toolbar

Table 5-1. IBD View Toolbar

Icon	Description
	Add a component
	Add a block
	Add moduleware component
	Add an external HDL (IP) model
	Add an embedded block
	Add a FOR, IF, ELSE or BLOCK generate frame
	Add a signal
	Add a bus
	Add a bundle
	Add a port map expression
	Add a property column
	Toggle filter
	Create viewpoint
	Add a net slice
	Fit the cell width to the contents of the selected cell
	Sort in ascending order
	Sort in descending order
	Clear net highlighting
	Edit BD
	Visualize as BD
	Requirement References (hidden by default and only visible when the requirement referencing is enabled).

Note



The  button has a pulldown menu which allows you to choose a FOR, IF, BLOCK (VHDL only) or ELSE (Verilog only) generate frame.

The toolbar can be displayed or hidden by setting the **IBD Tools** option in the **Toolbars** cascade of the **View** menu.

Getting Designs into IBD Editor

Working on a Previously Created HDS Design

You can get previously created HDL or block diagram HDS views into the IBD editor. Doing so you will be able to focus on areas of interest in your design using the different filtering mechanisms available. Refer to **Creating Filtered Views**. You will also easily spot any unconnected ports.

To get an HDL view into IBD editor:


1. Do one of the following to display the **Convert to Graphics** wizard:
 - Select an HDL view in the Design Manager window and choose **Convert to Graphics** from the popup menu or the **HDL** menu.
 - Open an HDL view in DesignPad and choose **Convert to Graphics** from the **Graphics** menu.
2. Choose IBD from the hierarchy description options and click finish.

Movie

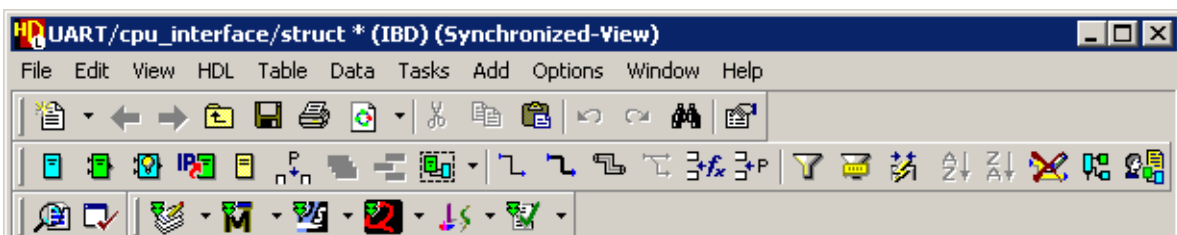


[How do I create an IBD using “Convert to Graphics”](#)

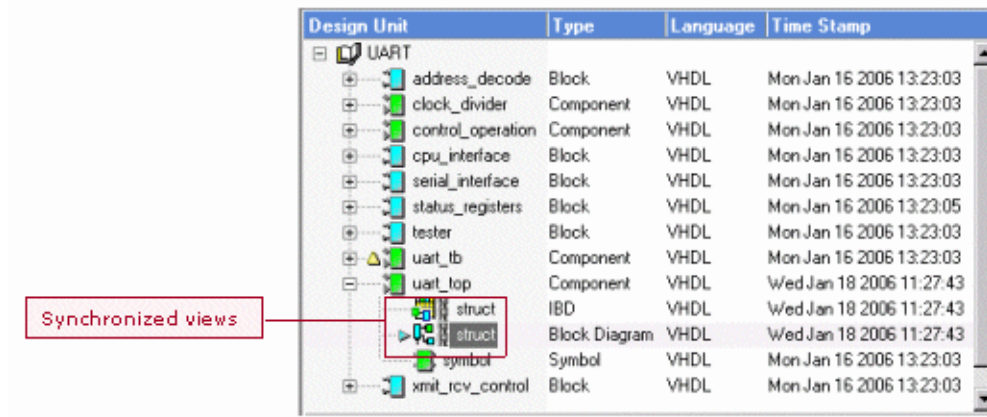
To get a block diagram view into IBD editor:

1. In the BD editor window click on the Edit as IBD  icon in the toolbar or choose **Edit as IBD** from the **Diagram** menu. A synchronized IBD view is created. Synchronized BD and IBD views are actually two representations of the same design in which saved changes in one view automatically appear in the other.

On choosing to create a synchronized view the title bar of the created view shows “Synchronized-View”. The next time you open your BD view and click the Edit as IBD icon your synchronized view will be automatically opened.





In the Design Manager synchronized views appear with a chain beside their icons.



Creating a New Design View

You can start creating a new design using the IBD editor.

To create a new IBD view using the Design Content Creation wizard:

1. Invoke the Design Content Creation Wizard by doing one of the following:
 - o Choose **Design Content** from the **New** cascade of the **File** menu or click the  icon in the Explore tab of the Design Manager left side bar.
 - o Click the  icon in the toolbar to display a menu and choose **Design Content**.
2. Choose Graphical View from the Categories pane and IBD from the File Type pane and click Next.
3. Specify view name and location and click **Next** to specify view interface or click **Finish**.

You can refer to **Design Content Creation Wizard**.

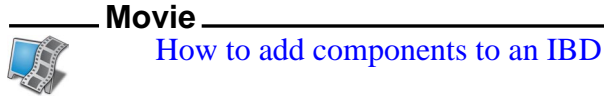
 **Tip:** You can click the  icon in the Design Explorer toolbar and choose **IBD** from the **Graphical View** cascade menu.

Adding Design Elements

Now that you have a design in the IBD editor the next step is to start working on this design. You can add new design elements, edit their properties, organize elements into groups and bundles or sort them.

Adding Components

You can instance components that are in an HDS library or you can choose to instance an external IP. You can also instance a component from a text file.



To add a component from an HDS library:

1. Select the Add Component icon from the toolbar or **Component** from the **Add** menu to display the component browser.
2. Add the library containing the component to the component browser then drag the required component to the IBD table.

You can also add components from the Design explorer.

1. Select one or more component from the Design Explorer window and choose **Copy** from the **Edit** or popup menu.
2. In the IBD Editor choose **Paste** from the **Edit** or popup menu.

You can simply drag a component from the Design explorer window to the IBD editor window.

To add a moduleware component:

1. You can do one of the following to display the component browser showing the ModuleWare library:
 - Click the ModuleWare icon in the IBD editor toolbar.
 - Select the Add Component icon from the shortcut bar or **Component** from the **Add** menu to display the component browser and click the ModuleWare icon.
2. Drag the selected component to the IBD editor window.

To add a component from an HDL text file:

You will first have to import your component into an HDS library.

1. In the Design Explorer select the library you would like to import your design file to.
2. Choose **File** from the **Import** cascade of the **File** menu to display the File Import dialog. Browse for your file, select Import and click **OK**.

Importing your design to an HDS library, allows you to deal with it as any HDS component. Refer to [To add a component from an HDS library:](#)

To add an external IP:

Use the  button or choose **IP** from the **Add** menu to display the Add External IP dialog.

Note



When adding a component to your design a row is added to the IBD matrix to show the number of unconnected ports. On expanding this row the unconnected ports' details are displayed.

Port Map:					+	+		-	-	Port	Actual
-	Unconnected Port filters							2			
										I: din(1:0)	
										O: dout	

To filter unconnected ports:

Choose a filter value from the component port column drop down list of the unconnected port group.

Adding Blocks

Use the  button or choose **Block** from the **Add** menu. You can then define the required view type.

To define the view type:

1. Select **New View** from the **Open As** cascade of the component popup menu to display the Open Down File Creation wizard.
2. Specify the new view type and click Next.
3. Type the view name and click finish.

Adding Embedded Blocks

Use the  button or choose **Embedded Block** from the **Add** menu.

Adding Nets


Adding a Signal or Bus

To add a signal or bus do one of the following:

1. Select **Signal** or **Bus** from the **Add** menu.
 - When you add a signal, a net declaration row is added to the table with a default name and scalar type while when you add a bus a net declaration row is added with a default name, vector type and bounds.
 - Each new signal or bus is given a unique name by adding an integer to the default name. (For example: *sig1*, *sig2*... for a signal or *dbus1*, *dbus2*... for a bus.)

You can change the default signal and bus name, type and bounds through the Default Settings page of the Structural Diagram Master Preferences Dialog. Refer to **Setting IBD Preferences**.
2. Type a net declaration directly into the Name, Bounds and Type cells for the empty row at the bottom of the table.

The following hints can help you enter signal and bus details:

- If you enter characters that match characters in an existing entry of the same column, the remaining characters are entered automatically.
- If you enter a net name followed by a valid bounds constraint, for example: *myNet(7 DOWNT0 0)*, the constraint is automatically moved to the Bounds column.
- Using the  key after entering a signal name automatically completes the row with default properties and moves the cursor to the name cell in the next row.

Note




Net declarations are automatically added to your design when you choose to connect unconnected ports. Refer to **Connecting Ports**.

Movie




[How do I create Nets, Slices and Bundles](#)

To edit a signal or bus do one of the following:

1. Click once and edit the cell contents. Note that you can choose from a pulldown list of standard types by clicking in a Type cell and using the  button. The current type or the type that most closely matches the current string is preselected in the list. For example, if the characters *st* are entered in a VHDL view, the *std_logic* type is selected.)


2. Use the  button or choose **Object Properties** from the **Edit** menu or popup menu to display the Object properties dialog box.

You can create a port on a block or embedded block by entering an I, B, O or U in the interconnect cell for the signal row and block instance column.

 **Tip:** The direction indicator (I, B, O or U) can also be selected from a drop down list.


Adding Ports

Enter an I (input), B (Bidirectional), O (Output) or U (Buffer for VHDL only) in the interconnect cell for the external interface column.

 **Tip:** The direction indicator (I, B, O or U) can also be selected from a drop down list.

You can add net declarations with external interface ports for all unconnected ports on a component. New net declaration rows are added with properties corresponding to the ports defined on the symbol for the component.

Adding a Net Slice

Select a net row and add one or more net slices by using the  button or by choosing **Net Slice** from the **Add** menu.

Specify the required index or slice range in the Slice column of the expanded net row. You can slice a net with no bounds to be able to connect it to more than one port.


Adding Generate Frames

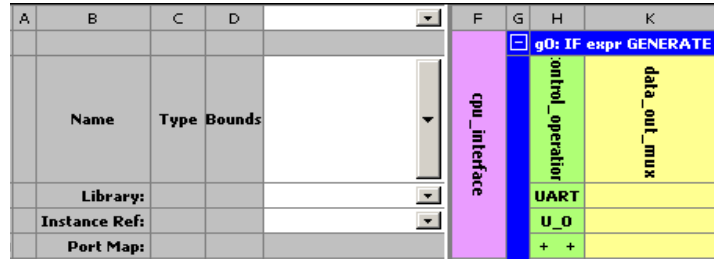
A generate frame can be used around one or more instances (including blocks, components and embedded blocks) on an IBD view to represent repeated, conditional or alternative structures in the HDL code.

For example, a FOR frame can be used to replicate a component which uses VHDL generics or Verilog parameters to define its interface or an IF frame may be used to implement the

contained instances only when the generate expression is true. IF and ELSE frames can be used to define alternative structures in a Verilog design.

To add a generate frame:

1. Select one or more instance columns.
2. Use the  or choose **Frame** from the **Add** menu to display the Frame options cascade menu.
3. Choose one of the following frame options



- | | |
|-------|---|
| FOR | A FOR frame can be used around a block or component to define a repeating instance. |
| IF | An IF frame can be used to define a conditional diagram area. |
| ELSE | An ELSE frame can be used with an IF frame which has the same label to define an alternative diagram. |
| BLOCK | A BLOCK frame can be used (in VHDL only) to define an area containing objects to be generated as concurrent HDL statements using the BLOCK keyword. |

The frame is added around the selected instances with a default title (comprising a label and expression). The label must be a valid HDL identifier. Example:

```
g0: FOR i IN 0 TO n GENERATE
```

The frame can be expanded and collapsed using the + and - buttons.

If you do not change the label, each new frame is given a unique label by adding an integer to the default name (for example: *g0*, *g1*, *g2*...). However, if you add an ELSE frame to a Verilog view and there is only one existing IF frame on the view, then the ELSE frame is given the same label.

Using Generate Frames for Repeating Instances

A FOR generate frame can be used in VHDL or Verilog to replicate a component where the relationship between the ports on the instances of the component and signals on the diagram can be described by an expression. For example, a number of instances attached to each element of a multi-bit bus where an instance is required connected in parallel for each element or group of elements.

Using Generate Frames for Repeating Structures

When you are using VHDL, it is possible to use a FOR generate frame around a diagram area containing a number of existing block and component objects.

Using Generate Frames for Conditional Structures

An IF generate frame can be used in Verilog to describe conditional structures. An alternative structure can be defined by using an ELSE frame which must have the same label as the corresponding IF frame.

Using a BLOCK Generate Frame

When you are using VHDL, a BLOCK generate frame can be used to cluster related concurrent statements in the generated HDL. All the HDL statements generated for the objects contained in a BLOCK frame are executed concurrently.

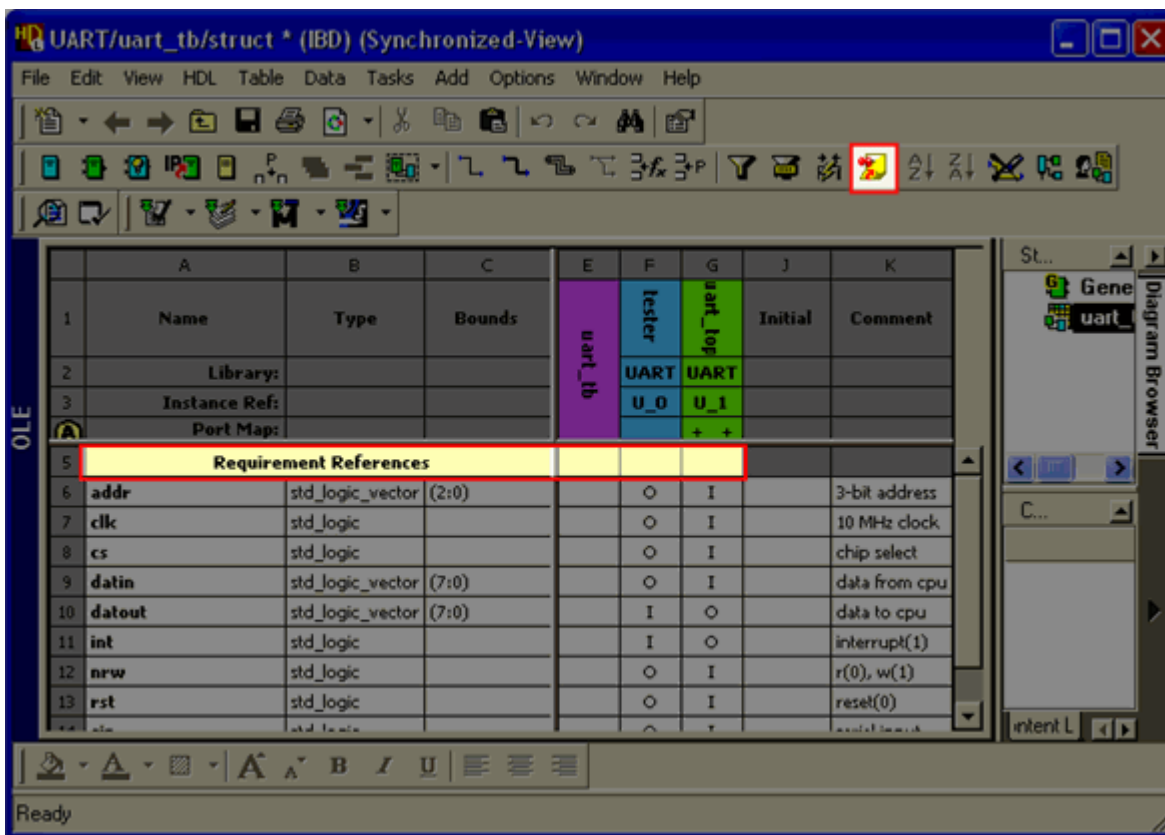
In the following example, separate BLOCK frames and frame declarations have been used to specify delay constants for the *ipdelay* and *opdelay* embedded blocks. The main functionality is implemented by two instantiations of *subcircuit* which are contained within a third BLOCK frame.

Adding Requirements References

IBD supports Requirements Referencing by a new dedicated row. The Requirement References row is hidden by default and is only shown when Requirement Referencing is enabled in the “Requirements Referencing Settings Dialog Box” (which is accessed by choosing **Options > Requirements Referencing** from the Design Manager’s toolbar) and HDS is reinvoked. Refer to [Enabling Requirements Referencing in HDS](#) in the [HDL Designer Series User Manual](#) for more information on the Requirements Referencing Settings dialog box.

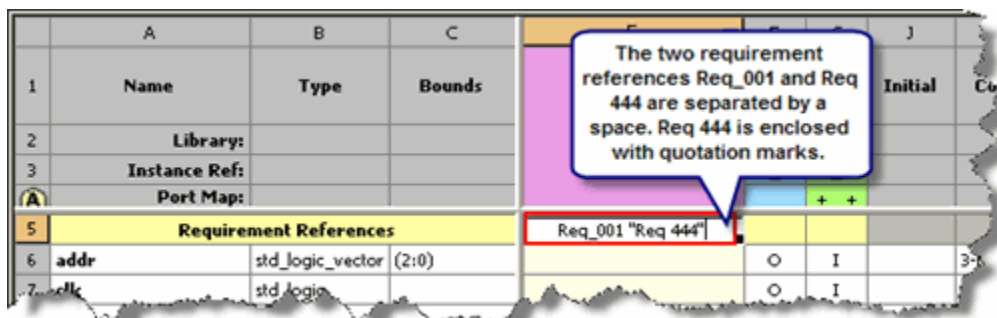
A Requirement References button is added in the toolbar to toggle the display of the Requirement References row. Like the row, the button is hidden by default and is only shown when Requirement Referencing is enabled.

Figure 5-1. Requirement References Row And Button In IBD



For an IBD, a requirement reference can only be added/pasted to columns (components, blocks and embedded blocks) as well as the interface column. Adding a requirement reference is simply done by clicking on the required cell and typing in the requirement reference. Pasting is available through the **Paste Requirement Reference** menu item of the **Edit** menu or through the RMB context menu of the supported columns.

Adding more than one requirement reference is possible only by inserting a space between them. If a requirement reference's name contains a space, then you need to enclose this requirement reference with quotation marks.



Like other graphical editors, generation and refreshing should follow for the requirements coverage information to be visible in the [Requirements References Column](#) of different containers of the Design Manager.

Connecting Design Elements

Having added your design elements to the IBD matrix, you can now start connecting nets, net slices and HDL expressions to blocks and components.

Connections are created either through direct connection of nets to ports or through port mapping. If you are using VHDL 93, you can map a function call or other valid HDL expression. If you are using Verilog, arbitrary expressions are allowed on inputs but outputs must be directly connected to nets.

Connecting Nets to Component Ports

You can connect nets and components using one of two approaches. The first approach is to connect existing nets to selected component ports i.e net-centric approach. The second approach is to select component ports then connect, add signals or add portIOs. This will create new signals connected to the selected component ports.i.e port-centric approach.

Connecting Existing Nets: Net_Centric_Connection Approach

To connect an existing net to a component port do one of the following:

- Click in the interconnect cell for the component and enter a direction indicator, the first available matching port with the specified direction is entered in the Port column.
- Use the + button to expand the Port column and click the interconnect cell to choose from a drop down list of available ports.

	A	B	C	D		F	G	J	K	L	N	O
1		Name	Type	Bounds		cpu_interface	control_operation	data_out_mux		fbitsel	Initial	Comment
2		Library:					UART			moduleware		
3		Instance Ref:					U_0			I0		
4		Port Map:					+	+	-	+	Port	
5												
6		+ Unconnected Port filters							2			
7		clk	std_log			I	I					
8		clk_div_en	std_log			I		I				
9		clr_int_en	std_log			I	I					
10		cs	std_log			I	I					
11		div_data	std_log (7:0)			I		I		I: din(1:0)		
12		nrv	std_log			I	I			O: dout		

You can connect to the full range of the port or enter the port name with an index or range, for example: *addr(5:4)*.

To connect an existing net to one or more component ports in one step:

1. Select a net, press the Ctrl key and select one or more unconnected ports.
2. Choose **Connect** from the popup menu of the last unconnected port.



Connecting Ports: Port_Centric_Connection Approach

In this approach, new net declarations are added to the IBD matrix with properties corresponding to the ports defined on the symbol of the component.

If the component has a different language from the parent view (for example, a VHDL component instantiated in a Verilog view) the signal properties are automatically mapped to the language of the parent view.

You are warned if any of the net names already exist in the IBD view and you can choose to create unique net names or connect the component to the existing net.



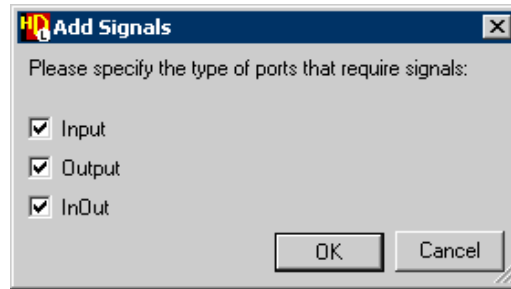
To connect unconnected component ports together:

Select one or more unconnected ports and choose **Connect** from the popup menu. The first chosen port is used to add a net of the same name, the subsequent ports are then connected to this net.

To connect unconnected component ports to new signals do one of the following:

1. Select the unconnected ports and choose **Add Signal** from the popup menu.

2. Select the component with unconnected ports. The **Add Signals** dialog box is displayed. Choose whether to add signals on Input, Output, InOut or Buffer (VHDL only) ports.



Mapping Expressions or Function Calls to Component Ports

Examples on Expressions:

Example 1: You can apply logical operations on signals or certain bits of a bus. For example, you can AND two signals and invert the result:

```
!(sig0 & sig1)
```

You can also OR bits in buses:

```
dbus0[3] | dbus5[3]
```

Example 2: You can also apply arithmetic operations, for example:

```
a + b
```


Example 3: You can use power. The expression below shows 2 raised to power *in_a*:

```
2**in_a
```

Example 4: You can call functions in expression rows. For example, the expression below calls a multiplier function:

```
mult(in_a,in_b)
```

To map an expression to a component port:

1. Choose **Expression Row** from the **Add** menu or use the  button and enter the expression in the Name column e.g **1'b1**.

- Use the + button to expand the Port column and click the interconnect cell to choose from a drop down list of available ports.

	A	B	C	D		F	G	I	J	K	L
1		Name	Type	Bounds		cpu_interface	ntrol_operati		data_out_mux		fbitsel
2		Library:					UART				moduleware
3		Instance Ref:					U_0				IO
4		Port Map:					+ - Actual				Port
5											
6		Unconnected Port filters									
7		1'b1									dout

You can also use the + button to expand the Actual column and enter the expression. A star sign * appears in the component interconnect cell to indicate that an expression is mapped to the port. Note that actuals take precedence over formal ports.

In some cases you may need to connect one expression to more than one port on the same component. Adding a slice to the expression will enable you to do that.

To connect an expression row to more than one port on the same component:

- Select expression row and choose **Slice** from the **Add** menu.
- Use the + button to expand the Port column of each expression row slice and click the interconnect cell to choose from a drop down list of available ports.

	A	B	C	D		F	G	I	J	K	L	N
1		Name	Type	Bounds		cpu_interface	ntrol_operati		data_out_mux		fbitsel	Initial
2		Library:					UART				moduleware	
3		Instance Ref:					U_0				IO	
4		Port Map:					+ - Actual				Port	
5												
6		+ Unconnected Port filters										
7		- 1'b1								I	din	
8												
9		clk	std_log			I	I				O: dout	
10		clk_div_en	std_log			I			I			

Movie



How do I create an expression to define complex port connections in IBD

To add ports from a component refer to [“Adding Ports”](#) on page 257.

Another Port_Centric_Connection Convention

In some cases it may be more convenient to connect a port by choosing from a dynamic list of design nets and expressions. A dynamic net/expression list is one that gives you the option to add new entries. The net /expression you select or add is automatically connected to your selected port.

To connect a single port by choosing from a dynamic list of design nets/expressions:

1. Expand the Unconnected Port Filters row. A list of unconnected ports appears in the component port column.
2. Click the actual column cell of the port you wish to connect. A dropdown list box is displayed. The content of the list box is sensitive to the type/range of the selected unconnected port.
3. According to what you select from the dropdown list the following occurs:
 - o If you choose to create a new net, a net declaration row is added to the IBD matrix with properties corresponding to the selected port defined on the symbol of the component and is connected to that port.
 - o If you choose to add a new expression, an expression row is added where you can type in your expression. The expression is automatically connected to the selected port.

Another important feature of this functionality is that it will automatically create child expression rows. That is to say, if you wanted to tie many unconnected ports on the same component high with the same expression '1'b1'. then after the first port, you will notice that child expression rows are added automatically.

VHDL Scalar Ports

```
<New Net>
<----->
<New Expression>
OPEN
'0'
'1'
List of the existing expression
List of existing nets
```

VHDL Vector Ports

```
<New Net>
<----->
<New Expression>
OPEN
(OTHERS => '0')
(OTHERS => '1')
List of the existing expression
List of existing nets
```

Verilog Scalar Ports

```
<New Net>
<----->
<New Expression>
OPEN
'0'
'1'
List of the existing expression
List of existing nets
```

Verilog Vector Ports

```
<New Net>
<----->
<New Expression>
OPEN
(OTHERS => '0')
(OTHERS => '1')
List of the existing expression
List of existing nets
```



Movie



Port-Centric Connectivity through Dynamic List of Nets/Expressions

Organizing View Layout

Expanding and Collapsing IBD Views

You can expand and collapse columns and rows in an IBD view by using the  and  buttons. Alternatively, you can use the following commands from the **Expand/Collapse Columns** and **Expand/Collapse Rows** cascades in the **Table** menu:

Expand/Collapse Columns

Expand All	Expand all components and generate frames
Collapse All	Collapse all components and generate frames
Expand All Frames	Expand all generate frames
Collapse All Frames	Collapse all generate frames
Expand Visible Components	Expand all visible components
Collapse Visible Components	Collapse all visible components

Expand/Collapse Rows

Expand All	Expand all rows
Collapse All	Collapse all rows



A visible component is one which is already visible in the table.

Moving Rows and Columns in an IBD View

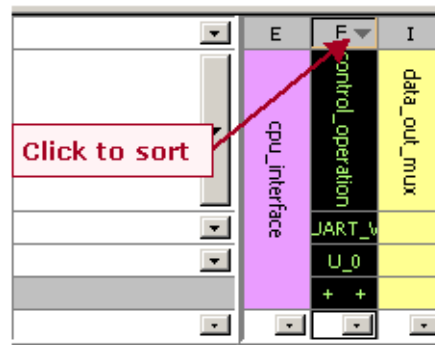
You can move a net row or instance column in an IBD view by clicking the row number or column letter and then dragging with the **Left** mouse button over the row number or column letter.

Sorting Rows and Columns in an IBD View


You can sort the rows in a selected column or columns in a selected row of an IBD view in ascending or descending alphanumeric order by either:

- Using the  /  button.
- Choosing **Data>Sort>Ascending** or **Data>Sort>Descending**.
- Choosing **Sort Ascending** or **Sort Descending** from the row/column popup menus.

You can also sort by clicking the triangular icon on the right side of the column header cell.



Grouping IBD Rows and Columns

You can group rows/columns by selecting a row or rows/column or columns and using the  button or choosing **Data>Group>Group**.

The selected rows/columns are added to a new group with the default name *GroupN* (where *N* is automatically incremented if it already exists).

Showing/Hiding Columns in an IBD View

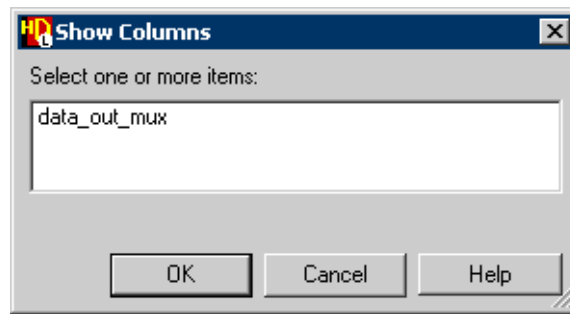
To hide a column:

Select the required column and choose **Hide Column** from the popup menu.

To show a hidden column:

1. Choose **Data>Show/Hide> Show Columns** to display the **Show Columns** dialog box.

2. Select the columns to show and click **OK**.



Adding Bundles to your IBD view

A bundle is a collection of logical signals. Connecting a bundle to a block or an embedded block will connect all the signals contained in that bundle to the specified block or embedded block. Changing the connectivity properties of the bundle changes the connectivity of the contained signals.

There are several way of creating a bundle and adding content to it.

1. Select a set of signals which you want in a Bundle. Then click the Bundle toolbar icon. A new collapsible bundle frame is created around the selected signals with a bundle name. Each new bundle is made unique by adding an integer to the default name. (For example: *bundle0*, *bundle1*, *bundle2*...). You can choose to edit the bundle name.
2. If you already have an existing Bundle, you can:
 - o Select the bundle, and hit the Add Signal or Add Bus toolbar icons.
 - o Drag existing signals into the bundle. Note that the connectivity of the signals has to be consistent with that of the bundle.
3. If you wish to define bundle connections enter the connection mode in the bundle interconnect cells.

Notice that the signals' interconnect cells are populated accordingly. If one of the signals is already connected you will receive an error.

To delete a bundle signal select the signal bundle row and delete the signal. To delete a bundle select the bundle frame row and press delete.

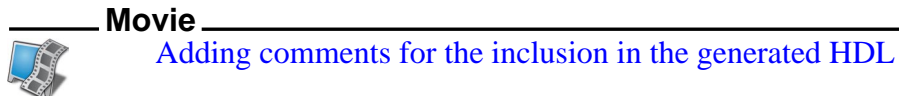
In the following example the created bundle includes the signals sig2 and sig3 that are inputs to Block1 and outputs to Block2.

A	B	C	D	E		G	H	I	J	K	L
1	Name	Slice	Type	Bounds		Manual	Block1	Block2	eb1	Initial	Comment
2	Library:						ATCH_ATCH_				
3	Instance Ref:						U_0	U_1			
4	Port Map:										
5											
6	sig0		std_log (15:0)					○	I		
7	sig1		std_log (15:0)				I	○			
8	Bundle						I	○			
9	sig2		std_log (15:0)								
10	sig3		std_log (15:0)								
11	sig2		std_log (15:0)				I	○	○		
12	sig3		std_log (15:0)				I	○	I		

Adding Net/Component Comments

To add a comment:

1. Choose **Comments** from the **popup** menu of the selected net or component to display the Comments dialog box.
2. Enter End of line, Before or After comments.



Adding Property Columns/Rows

You can add property rows/columns as additional comments or as sort/filter keys.



Creating Filtered Views of the Design

Large designs can often be very difficult to manage and deal with. You may need to focus and work on a subset of the design data. IBD filters enable you to display and work on only design data of interest.

You may find it handy to be able to refer to your filtered views: IBD viewpoints provide persistent views of filtered data that are easily retrievable.

Defining Filter Settings and Logic

To specify filter settings and logic do the following:

1. Choose **Data>Filters>Filter Settings**. The **Filter Settings** dialog is displayed.



Tip: You can display the Filter Settings dialog by choosing **Filter Settings** from the popup menu of the design filter row/column. If the filter row and/or column are not displayed toggle the filter icon in the design explorer toolbar.

2. In the General group box specify whether you want to match case and/or match whole word.
3. Choose the type of filter expression to be used (simple or regular). For more information on refer “Regular Expressions”
4. In the Filtering Logic group box specify the filtering logic you will be using as:
And: which displays all rows and columns that meet all filter expressions
Or: which displays rows/columns that display any filter expressions.
5. Click **OK**.

You can invert any of your defined filter expressions by choosing **Invert Expression** from the popup menu of the filter expression cell.

Creating Persistent Subset Views of the Design

On selecting to filter rows all columns connected to these rows are displayed. Similarly choosing to filter columns displays all rows connected to these columns

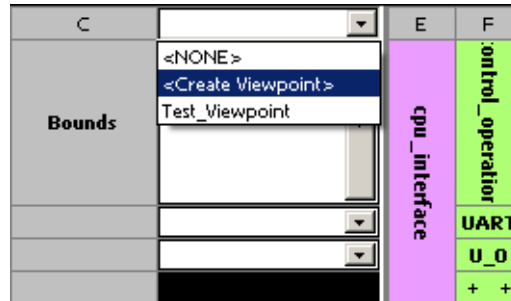
To create a viewpoint:

Select required rows and/or columns and do one of the following:

- Choose **Data>Viewpoints>Create from the cascade of the** menu.
- Click the Filter icon in the design manager toolbar.
- Choose **Create Viewpoint** from the row/column popup menu or the viewpoint dropdown box.

You can also take a snapshot of what’s currently visible without selecting any rows or columns.

A viewpoint *filterx*(ie *Filter1*, *Filter2*,...) is created. Viewpoints are saved in the drop down box of the filter cell.



To rename a viewpoint:

Viewpoints names can be edited in the dropdown box to have more indicative names.

On saving your work you can always refer back to your created filtered views by:

To delete a viewpoint:

1. Choose **Data>Viewpoints>Delete** or **Delete Viewpoint** from filter row/column popup menu to display the Delete Named Filter dialog.
2. Select the filter to be deleted and click OK.

Note



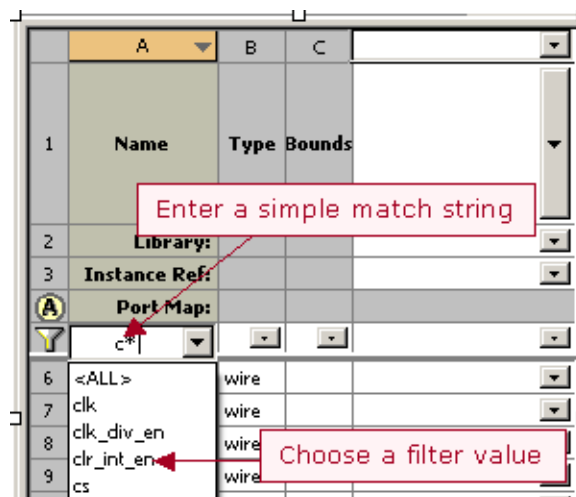
Any edits(add/remove/copy/paste) a named filter content undergoes are preserved when toggling between filters.

Pruning IBD Designs

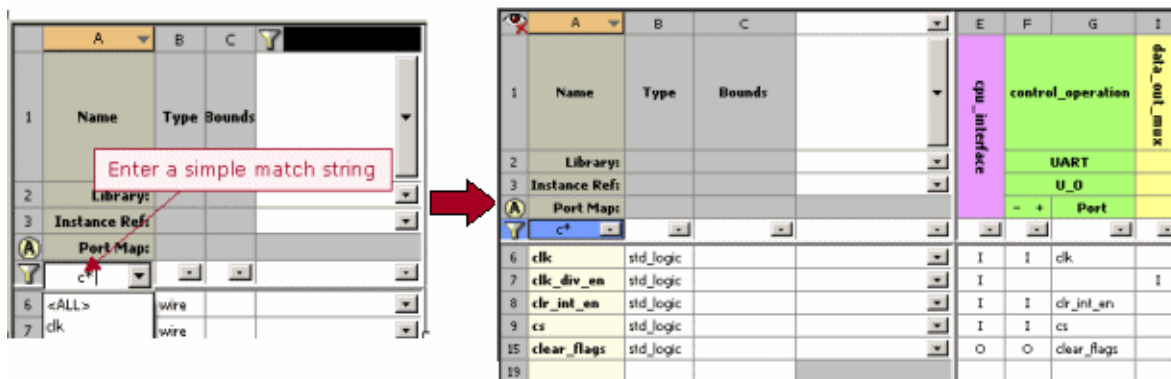
Filtering Nets in an IBD view

To filter IBD view nets do the following:

Select a column's filter cell. Choose a filter value from the drop down list or enter a simple match string in the drop-down entry box.



In the example below we have chosen to filter all nets starting with the letter C by entering the following expression C*. Notice the eye icon indicating that the view is filtered.



Select one of the following from the dropdown list of the cell at the intersection of the row and column filters.

	A	B	C		E	F	G	I
1	Name	Type	Bounds		cpu_interface	control_operation		data_out_mux
2	Library:					UART		
3	Instance Ref:					U_0		
4	Port Map:					-	+	Port
5								
6	clk	std_logic		<ALL>	I	I	clk	
7	clk_div_en	std_logic		Interconnected Nets	I			I
8	clr_int_en	std_logic		Common Nets	I	I	clr_int_en	
9	cs	std_logic		Connected Nets	I	I	cs	
10	div_data	std_logic_vector (7:0)		Floating Nets	I			I
				Unconnected rows				

- All: Shows all the rows.
- Interconnected Nets: Shows nets that are connected to more than one component/block with bidirectional or different modes.
- Connected Nets: Shows nets that are connected to at least one cell.
- Unconnected Rows: Shows nets, bundles, expressions, user rows that are not connected to any cell.
- Floating Nets: Shows nets that are connected to only one cell.
- Common Nets: Shows nets that are connected to more than one cell.

Movie



[How do I show unconnected signals and rows.](#)

Note



If a filter matches a net in a group or a net slice the parent net is included.

Filtering Components in an IBD View

To filter IBD view columns:

1. Identify the row you would like to filter your components based upon e.g. Name, Library, net/expression.

2. Choose a filter value from the corresponding column filter cell dropdown list.

The IBD matrix displays the columns that satisfy the selected filter values.

Managing Design Hierarchy

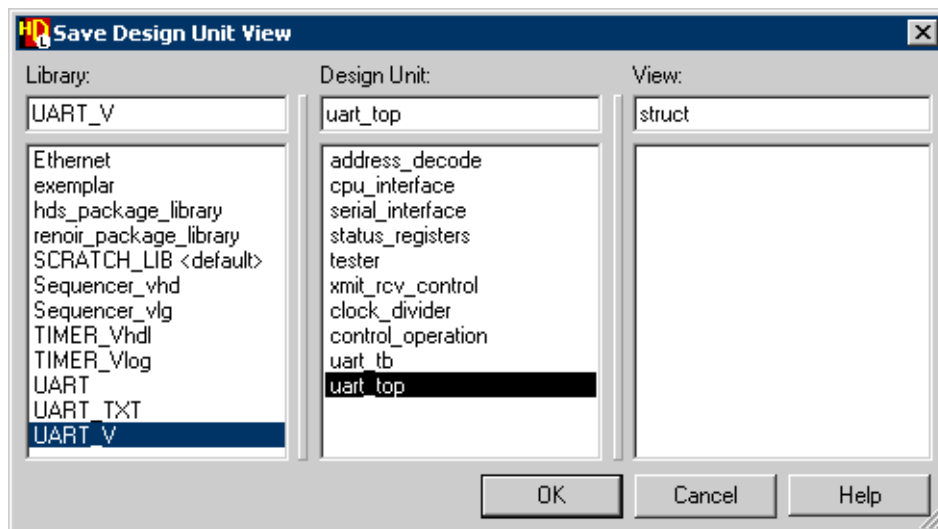
The IBD editor allows the hierarchy to be manipulated at any time. Hierarchy can easily be added or removed to facilitate design understanding or to improve performance of “down stream” tools such as synthesis.

Adding a Level of Hierarchy

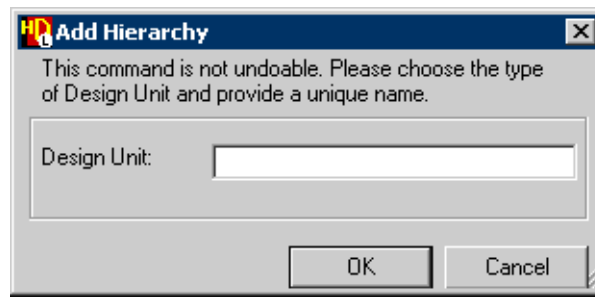
Adding hierarchy replaces the selected objects by a new component and moves the selected objects into the new design unit view.

To add a level of hierarchy:

1. Select components to be re-leveled.
2. Choose **Add Hierarchy** from the **Re-level** cascade of the **Table** menu or popup menu.
An error message is issued if:
 - Signals which constitute the interface of the new cells have slices.
 - Selected objects are not in the same frame scope.
3. If prompted to save the design view click **OK** and specify Library, Design Unit and View name in the Save Design Unit View dialog.



- Specify the name of the new design unit in the Add Hierarchy dialog and click **OK**.



A child IBD view is created and the objects replaced by a single new instance on the parent table.



Movie

[Click here to see a demonstration of adding an extra level of hierarchy](#)

Flattening Design Hierarchy

Removing hierarchy deletes the selected component instance and replaces it by the objects in the child hierarchal view.

To flatten a level of hierarchy:

- Select a single component instance to be re-leveled.
- Choose **Remove Hierarchy** from the **Re-level** cascade of the **Table** menu or popup menu. An error message is issued if you attempt to:
 - Remove hierarchy from an instance which is opened.
 - Remove hierarchy from an external IP or moduleware instance.
 - Select an instance which is not described by an IBD view.
 - Remove hierarchy from an instance which has a different language.
- Click **Yes** to confirm the design objects to be deleted by the re-level operation. You may need to manually update any other views that referenced the deleted component.

The child IBD view is deleted and its component instance is replaced by the objects in the child hierarchal view. If any of the instances included other hierarchal views, their hierarchy is retained but can be removed by another re-level operation.

The package references or compiler directives for the child view are merged with those defined for the parent view.

Movie



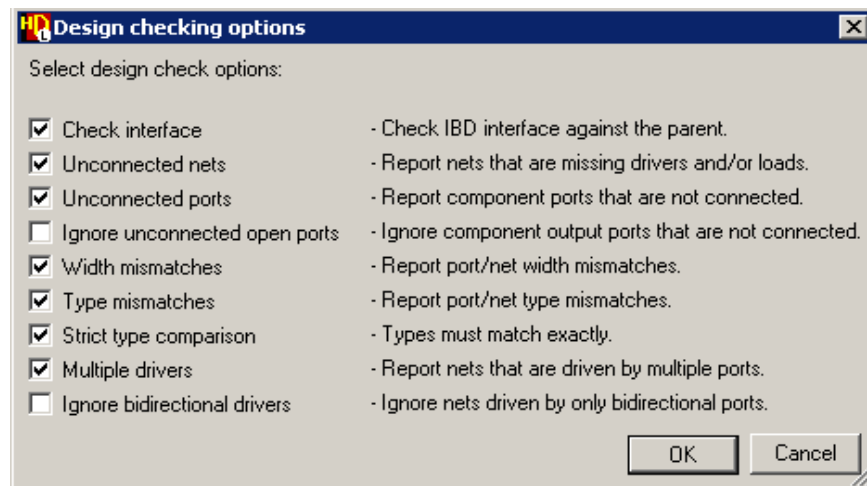
[Click here to see a demonstration of flattening a level of hierarchy](#)

To investigate a child view:

Select desired component column and double click to display child view.

Checking a Design in IBD Editor

1. Select **Design Checking Options** from the **Table** menu to display the Design Checking dialog.
2. Set the desired checks and click **OK**.



3. Select **Run Design Rule Checks** from the **Table** menu or click the  button.

Generating HDL from IBD views

Controlling the Generated HDL Code

This is really split into three areas, setting the order of generation within an individual IBD, specifying the hierarchical generation preference, and specifying the style of the generated code.


Setting Generation Order

The declarations of signals and external ports are either ordered automatically or manually. In automatic ordering generation order is set by mode(in,out,inout or buffer) and alphanumeric name irrespective of the rows display order. In manual ordering generation order is set by the rows display order.

In both automatic and manual ordering cell generation is set according to column order. Unlike BD objects there is no frame sequence number or embedded block number to specify generation order. The order can be modified simply by the use of a drag and drop approach to re-locate components to the desired order. The right mouse button must be used to select the column. When more than one column is used to describe a component the leftmost column should be selected.

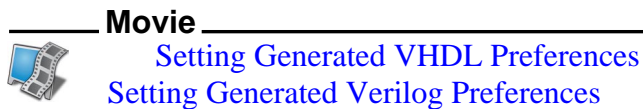
Additionally it is possible to sort rows or columns by order using the sort keys on the main task bar.

To enable manual ordering do one of the following:

- Select **Manual** from the popup menu of the  icon.
- Select **Manual** from the **Generation Order** cascade of the **Table** menu.



Setting the Style of the VHDL or Verilog code

The style of the VHDL or Verilog code is controlled through a number of preferences.



Setting the Generation Hierarchy Level

To generate HDL from IBD views:

1. Select **Main** from the **Options** menu to display the Main Settings dialog box.
2. Select the **Checks** tab and mark the Perform Checks option and click **OK**.
3. Use the  button or choose one of the **Generate** options from the **Tasks** menu. If a block or component is selected, the generation command operates on the selected object (or objects).
4. You can use the  button to display a pulldown palette with options to run the task on a single design level, the hierarchy through blocks, the hierarchy through component or the hierarchy through components from the design root:



Run Single

Run Through Blocks

Run Through Components

Run Through Components from the Design Root

Enforcing Generation

Views which have not been changed since they were last generated are not generated unless you have set the HDL generation run options to force regeneration.

If no views need to be generated, a single dot is displayed for each processed hierarchy with a row of dashes to represent each selected hierarchy followed by a generation completed message.

To Enforce Generation:

1. You can optionally enforce generation by choosing **Settings** from the **Generate** cascade of the **Tasks** menu to display the Generator Settings dialog.
2. Check the Set Generate Always when using this task option.



Cross Referencing Generation Errors

Any generation errors are reported in the Task Log window.

Note



If an error is encountered during bulk parsing, the timestamp for the generated file is set to the oldest date allowed by the system. (January 1st 1970 on UNIX or January 1st 1980 on a PC). This ensures that the generated HDL file is retained with an older timestamp than the graphical view.

You can automatically display the generated HDL or graphics corresponding to an error message in the Task Log window by double-clicking on the message (or by explicitly clicking the  or  button).

Refer to “Task Log” in the *HDL Designer Series User Manual* for more information about the Task Log window.

Setting a Black Box for Synthesis

You can make the active IBD view a black box for synthesis by setting the **Black box for Synthesis** option in the **HDL** menu. When this option is set, synthesis control pragmas are included in the generated HDL so that the view is available for simulation but is ignored for synthesis.

For Verilog, the synthesis off pragma is inserted after the input/output statements and after any Verilog parameters declared in the symbol but before the type declarations. The synthesis on pragma is inserted immediately before the end module statement.

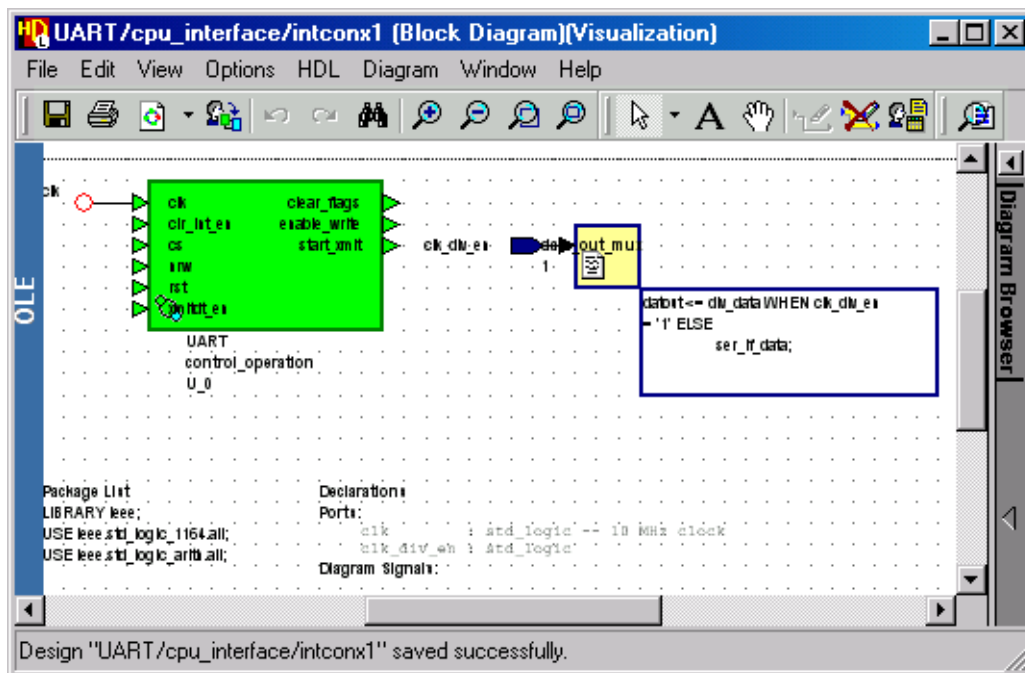
Documenting IBD Design Views

Creating Visualization Views

You can visualize your IBD views or subsets of your design as Block diagrams. Visualized views allow only non-logical edits.


To visualize your IBD views:

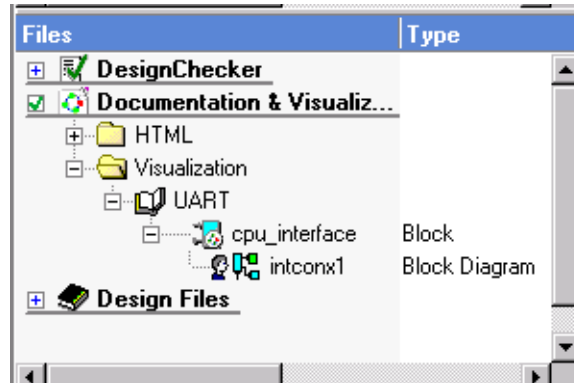
Choose Visualize as Block Diagram from the Table menu or click the Visualize as BD icon in the IBD toolbar and save your visualized view.



To open a visualized view:


1. In the Files pane, expand the Documentation and Visualization node to display the *Visualization* folder.

2. Click on the plus sign + to expand the *Visualization*; each *<library>* node, which is indicated by the book icon , holds the visualized design units.



Exporting to HTML

You can export IBD views or hierarchy of views as HTML pages which can be displayed in a compatible Web browser.

1. Use the  button or open **File > Document and Visualize > Single File/ Hierarchy Through Components/ Through Design Root** to display the Document & Visualize dialog box. Notice that the Visualize your Code option is disabled, while the Create a Website option is selected.
2. Browse for the target directory in which you wish to save your exported files.
3. Click the **Options** button to display the Documentation and Visualization Options dialog box. You can configure the HTML export preferences in the Website Options page as well as the HTML Settings and Graphics Settings sub-pages or leave the preset default options and click **OK**.
4. On the Document and Visualize dialog click **OK** to display your web browser showing the exported HTML pages.

Refer to “Exporting HTML Documentation” in the [HDL Designer Series User Manual](#).

Exporting to TSV or CSV Files

You can export IBD views to text files (TSV) or Excel files (CSV).

1. Select **TSV** or **CSV** from the **Export** cascade of the **Table** menu.
2. In the Select a File dialog enter the name of the file to which you wish to export your design and click **Save**.

Compiler Directives

When you are creating Verilog based designs you can insert compiler directives to pass information to the Verilog Compiler or any downstream tool.

To set compiler directives:

1. Choose **Compiler Directives** from the **Table** menu of the IBD editor to display the Compiler Directives dialog.
2. Enter Pre, Post or End Module Directives.

Setting IBD Preferences

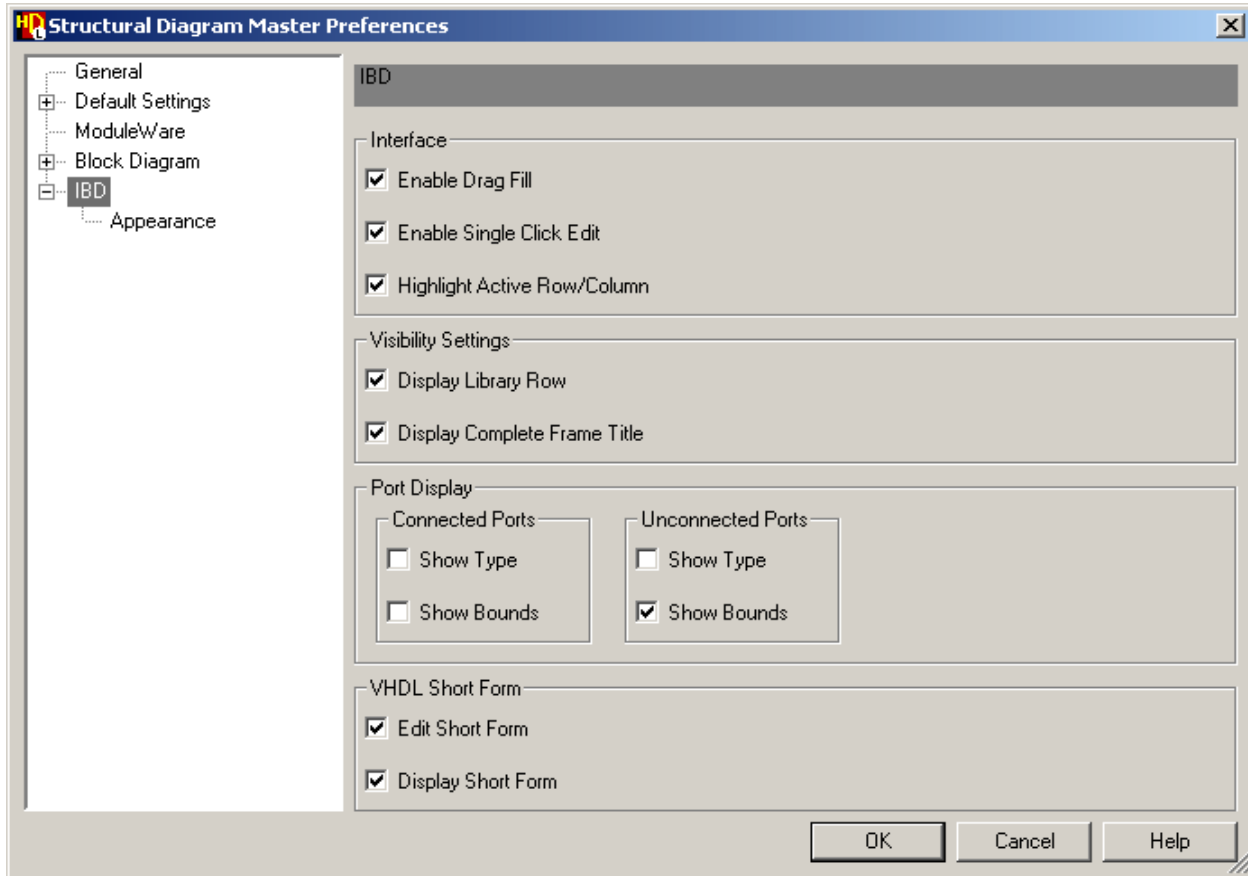
You can set IBD preferences by choosing **Options>Master Preferences>Structural Diagram** in the design manager to display the Structural Diagram Master Preferences dialog box.

The dialog box has separate pages for setting **General** preferences, **Default Settings**, **ModuleWare** preferences, **Block Diagram** preferences (which include **Display Settings**, **Appearance** and **Background** preferences), in addition to **IBD** preferences.

You can set the IBD appearance (but not default values) for the active diagram by choosing **Diagram Preferences** from the **Options** menu in the IBD editor to display the **IBD Preferences** dialog box.

When you edit these preferences for the active diagram, the dialog box allows you to choose whether the preferences are applied to new objects or to both new and existing objects in the diagram.

The **Appearance** page allows you to set visual attributes for individual block diagram objects.



These attributes include the foreground and background colors, line color, fill style, line style, line width and text font. Some attributes may not be available for all objects (for example, the line style, width and color attributes are not available for a text object).

Chapter 6

Port Map and Generate Frames

This chapter describes how *port map frames* and *generate frames* can be used on a graphical *block diagram*.

Port Map Frames	283
Adding a Port Map Frame.	283
Editing a Port Map	284
Generate Frames	287
Generate Frames	287
Adding a Generate Frame	288
Using Generate Frames for Repeating Instances	289
Using Generate Frames for Repeating Structures.	291
Using Generate Frames for Conditional Structures	292
Using a BLOCK Generate Frame	296
Using Nested Generate Frames.	298
Editing Generate Frame Properties.	302

Port Map Frames

Mapping between the actual signals on the block diagram and *formal* ports on a component with different properties may be shown by using a *port map frame*. For example, to connect individual slices of a signal to separate ports on the component.

Adding a Port Map Frame

You can add a port map frame to a component on a block diagram by choosing **Enable** from the **Port Map Frame** cascade in the **Diagram** or popup menu when a component is selected.

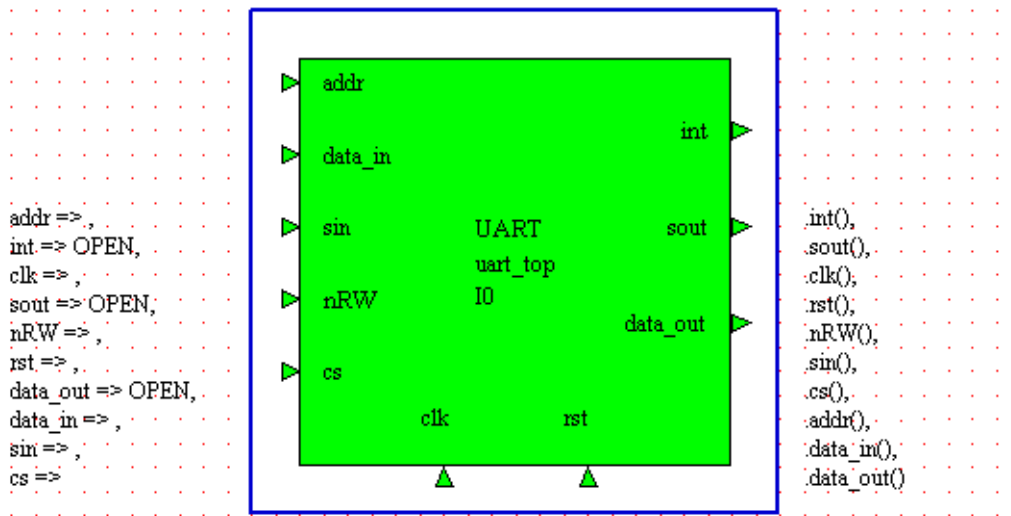
Note



You can also add a *port map frame* by setting the **Enable Port Map Frame** option in the **Components** tab of the Object Properties dialog box.

The following example shows the top level component from the UART example design with a port map enabled on the block diagram. The type information has been hidden in this illustration and the default mapping text is shown for both VHDL (on the left) and Verilog (on

the right). Note that for VHDL, unconnected output ports are shown with the default value OPEN.




The port map frame can be used to map connections between formal ports on the component and actual signals and buses on the block diagram.

When a port map frame is enabled, any signal or bus can still be connected directly to a component port provided that their properties are compatible.

You can also connect any signal, bus or bundle to the port map frame and edit the port map text to define the connections to the ports. For example, you can connect a bus to the port map frame and assign each slice or element of the bus to a separate formal port. A port can also be mapped to any valid HDL expression.

When port mapping is enabled on a block diagram, a frame is shown around the component and the port mapping list can be optionally shown or hidden on the diagram.

You can remove a port map frame on a block diagram by using the **Del** or  key when only the frame is selected or by choosing **Disable** from the **Port Map Frame** cascade. All connections to the frame are unconnected and shown as dangling net connectors.

Editing a Port Map

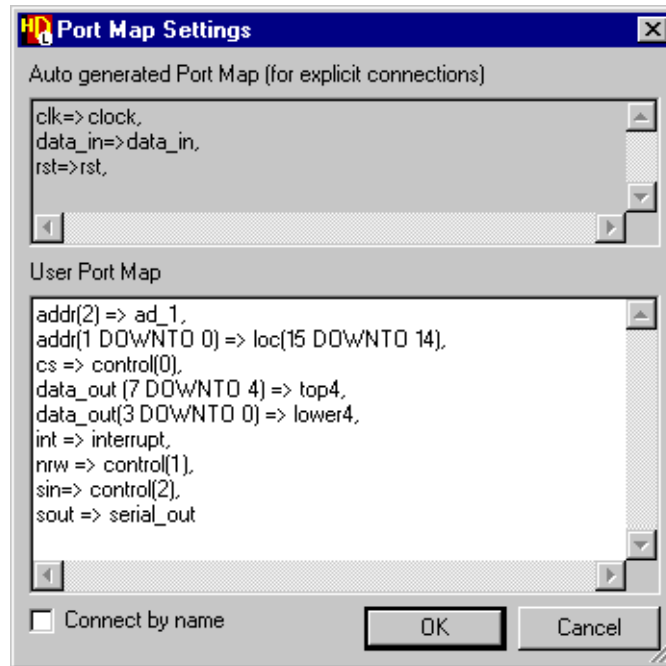
You can edit the port mapping by directly editing the text on the diagram, by using the **Components** tab of the Object Properties dialog box or by choosing **Edit** from the **Port Map Frame** cascade of the **Diagram** menu (in a block diagram).

The Port Map Settings dialog box shows any explicit connections which have been made on the block diagram in a read-only window. The editable window at the bottom of the dialog box can be used to enter user port mappings which must be specified using the correct language (VHDL or Verilog) syntax for your editor.

When **Connect by Name** is set, any signal with the same name as a component port is implicitly connected.

VHDL Port Map Example

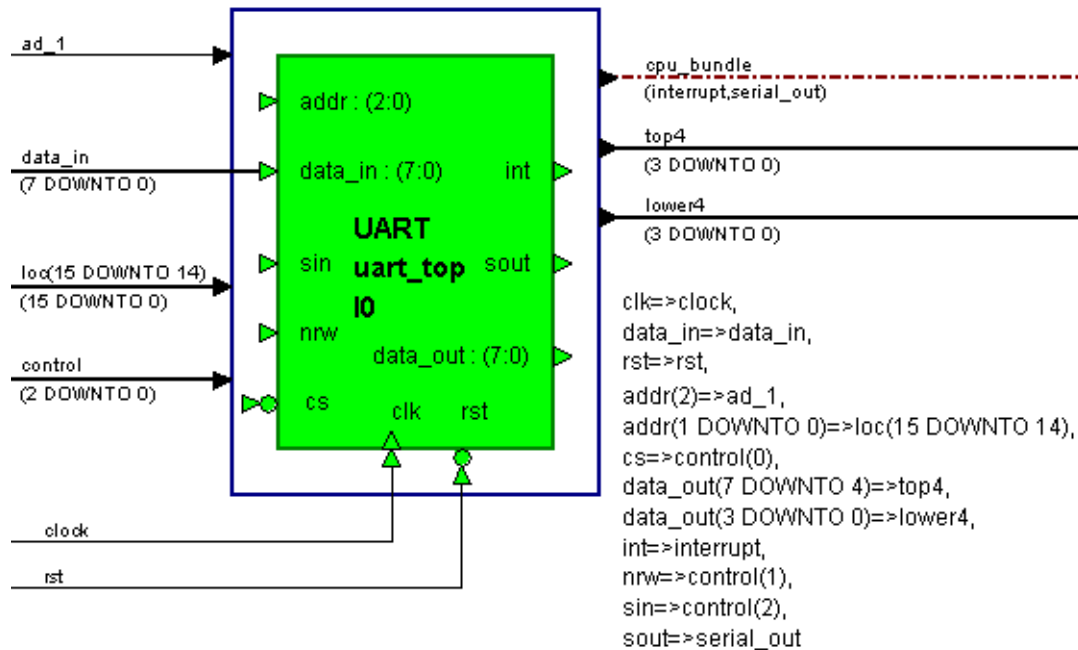
The following picture shows the VHDL UART example design connected using a port map:



The direct connections for *clk*, *rst* and *data_in* are shown in the read-only area at the top of the dialog box and the user port map connections have been entered in the lower entry box. The *ad_1* signal is connected to index 2 of port *addr* and slice 15 DOWNTO 14 of the *loc* bus is connected to bits 1 and 0 of bus *addr*.

The *control* bus is connected to the *sin*, *nRW* and *cs* ports and the *data_out* port mapped to separate *top4* and *lower4* buses. The *int* and *sout* output ports are mapped to the signals

interrupt and *serial_out*. This mapping corresponds to the following connections on a block diagram:



Verilog Port Map Example

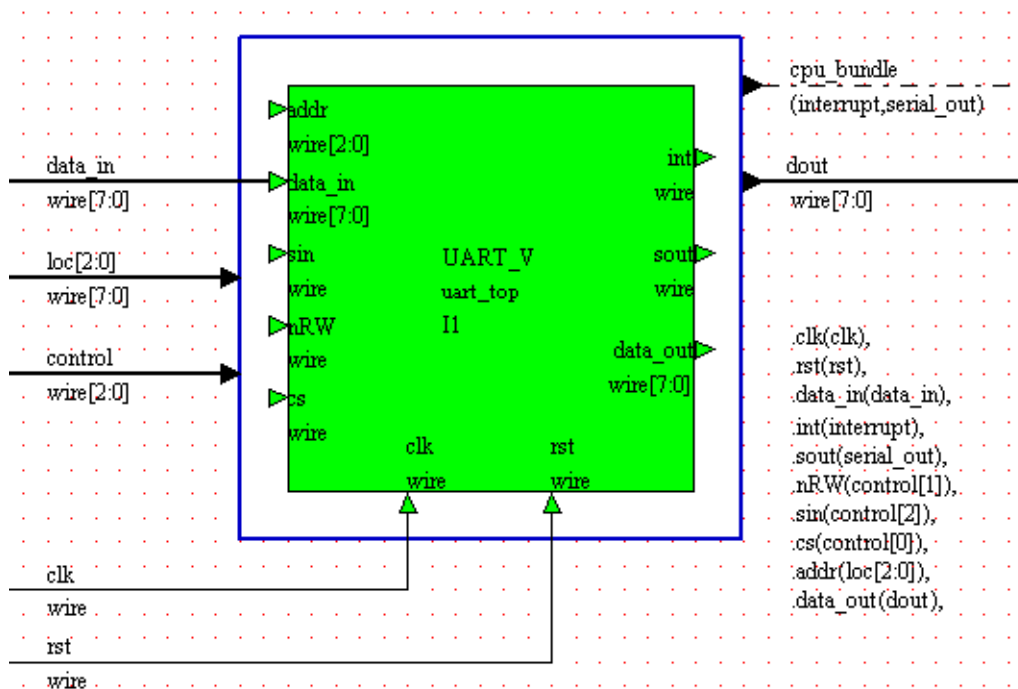
When you are using Verilog, the port mapping has the following syntax:

```

.clk(clk),
.data_in(data_in),
.rst(rst),
.addr(loc[2:0]),
.cs(control[0]),
.data_out(dout),
.int(interrupt),
.nrw(control[1]),
.sin(control[2]),
.sout(serial_out)

```

The following picture shows the Verilog UART example design connected using a port map frame.



Bits 2 to 0 of the *loc* bus are connected to the *addr* bus. The *control* bus is connected to the *sin*, *nRW* and *cs* ports and the *data_out* port mapped to the *dout* bus. The *int* and *sout* output ports are mapped to the signals *interrupt* and *serial_out* which are connected using the bundle *cpu_bundle*.

Generate Frames

A *generate frame* can be used around one or more instances (including blocks, components and embedded blocks) on a *block diagram* to represent repeated, conditional or alternative structures in the HDL code.

For example, a FOR frame can be used to replicate a component which uses VHDL generics or Verilog parameters to define its interface or an IF frame may be used to implement the contained instances only when the generate expression is true. IF and ELSE frames can be used to define alternative structures in a Verilog design.


Note



A generate FOR frame provides a static, compact representation of a design on a graphical block diagram or IBD view. When a VHDL code is loaded into a simulator, the generate statement is elaborated (expanded) into individual occurrences of the objects within the frame. However, most Verilog synthesis tools do not support automatic elaboration.

VHDL statements using the BLOCK keyword to define a cluster of concurrent statements can be represented by a BLOCK frame around an area containing any of the standard block diagram or IBD view objects. When HDL is generated, the frame contents are interpreted as a cluster of concurrent statements using the BLOCK keyword. A BLOCK frame (and FOR or IF frames defined using VHDL-93) may include local declarations which apply only within the frame.

Adding a Generate Frame

You can add a generate frame on a block diagram by using the  button or choosing **Frame** from the **Add** menu to display a cascade menu which provides the following frame options:

Option	Shortcut	Description
FOR	F6	A FOR frame can be used (in VHDL or Verilog) around a block or component to define a repeating instance or (in VHDL only) to define a repeating diagram area which may contain any number of other diagram objects.
IF	Shift + F6	An IF frame can be used (in VHDL or Verilog) to define a conditional diagram area.
ELSE	Ctrl + F6	An ELSE frame can be used (in Verilog only) with an IF frame which has the same label to define an alternative diagram area.
BLOCK	Alt + F6	A BLOCK frame can be used (in VHDL only) to define an area containing objects to be generated as concurrent HDL statements using the BLOCK keyword.

If one or more objects are selected on a block diagram, the frame is added around the selected objects. If nothing is selected, a ghosted frame is attached to the cursor and can be placed by clicking at the required location. You can resize the frame as it is added by holding down the **Left** mouse button and dragging with the cursor before you click for location or it can be resized later by dragging the resize handles.

A frame can be "nested" within another frame. However, a warning is issued when you save a block diagram if an instance is enclosed by overlapping frames. (This construction is considered to be an error by HDL generation.) If an instance is not completely enclosed by a frame, it is considered to be outside the frame.

All frames are added with a default title (comprising a label and expression) and default number. The label must be a valid HDL identifier.


If you do not change the label, each new frame is given a unique label by adding an integer to the default name (for example: *g0*, *g1*, *g2*...). However, if you add an ELSE frame to a Verilog view and there is only one existing IF frame on the view, then the ELSE frame is given the same label.

The frame number is used to determine the order of insertion in the generated HDL and can be any positive integer but must be unique. If you specify a number which is already used by another frame on the same view, the numbers are swapped.

Note



The frame label for a VHDL instance can be an extended identifier but you cannot use an escaped identifier for a Verilog instance.

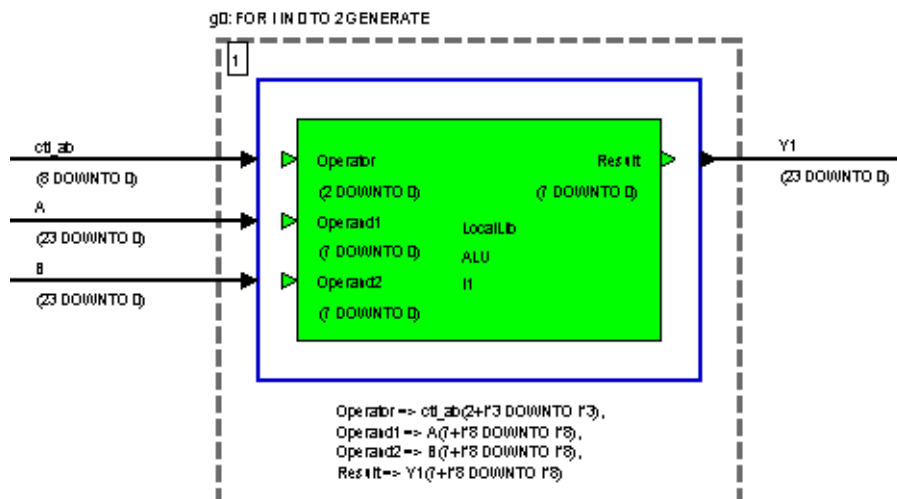
You can edit the title text and number by direct text editing or you can use the **Frames** tab in the Object Properties dialog box which is displayed when you use the  button or choose **Object Properties** from the **Edit** menu.

The syntax used in the frame title text is automatically checked for the hardware description language of the active view.

Using Generate Frames for Repeating Instances

A FOR generate frame can be used in VHDL or Verilog to replicate a component where the relationship between the ports on the instances of the component and signals on the diagram can be described by an expression. For example, a number of instances attached to each element of a multi-bit bus where an instance is required connected in parallel for each element or group of elements.

The following VHDL example shows an arithmetic logic unit (ALU) which is repeated three times with each instance connecting to the next group of elements in the bus. The repetition is defined by the VHDL expression *FOR i IN 0 TO 2 GENERATE* and each group of signals assigned to the ports by a port map frame.

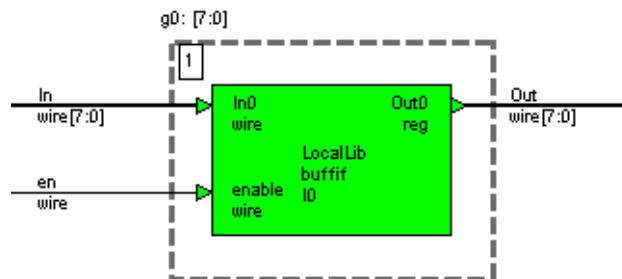


This example represents the following VHDL code:

```
ARCHITECTURE struct OF repetitive-instance IS
  COMPONENT ALU
  PORT (
    Operator: in unsigned(2 DOWNTO 0);
    Operand1: in unsigned(7 DOWNTO 0);
    Operand2: in unsigned(7 DOWNTO 0);
    Result: out unsigned(7 DOWNTO 0)
  );
  END COMPONENT ALU;
BEGIN
  -- Generates 3 instances of ALU.
  gen1: FOR i IN 0 TO 2 GENERATE
    I1: ALU
      PORT MAP (
        Operator => Ctl_ab(2+i*3 DOWNTO i*3),
        Operand1 => A(7+i*8 DOWNTO i*8),
        Operand2 => B(7+i*8 DOWNTO i*8),
        Result => Y1(7+i*8 DOWNTO i*8)
      );
    END GENERATE;
  END struct;
```

Repeating instances in Verilog are simpler (but less flexible than in VHDL) since only direct port mapping is supported for Verilog and a port map cannot be used.

The following Verilog example shows a tri-state buffer *Buffer1* which is repeated for each bit of the 8-bit input and output buses. In this case, each instance gets a part select of the range expression *[7:0]* starting from the right-hand index and the enable signal is connected to every instance.



This example represents the following Verilog code:

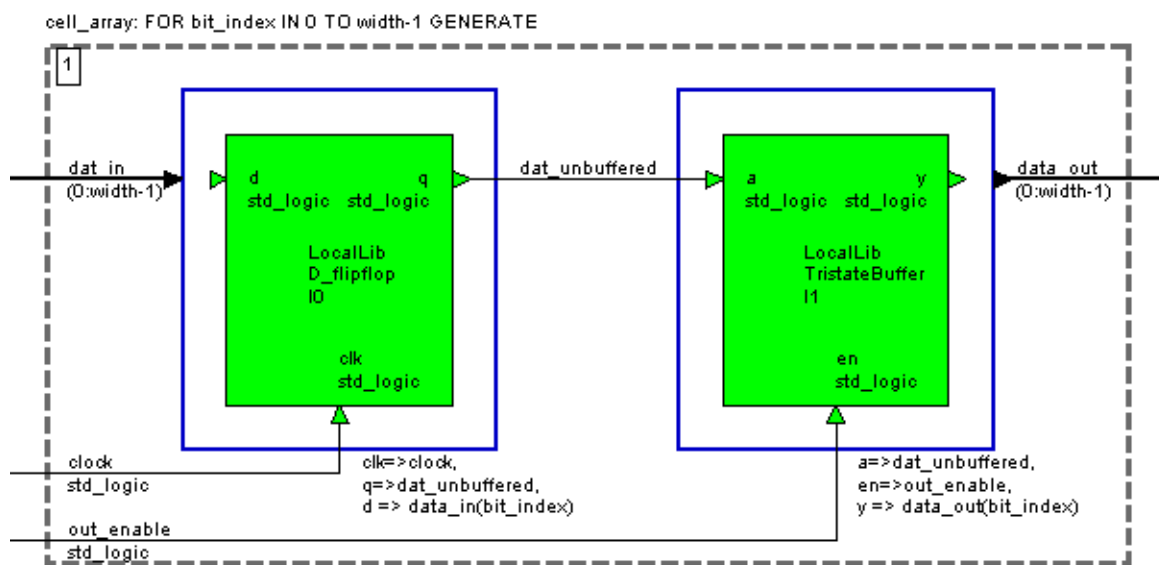
```
module Tristate(
  In,
  en,
  Out
);
input  [7:0] In;
input      en;
output [7:0] Out;
wire [7:0] In;
wire      en;
wire [7:0] Out;
```

```
buffif IO[7:0](
    .In0    (In),
    .enable (en),
    .Out0   (Out)
);
endmodule // Tristate
```

Using Generate Frames for Repeating Structures

When you are using VHDL, it is possible to use a FOR generate frame around a diagram area containing a number of existing block and component objects.

The following example shows a variable width bank of registers. Each register consists of a flip-flop whose output is connected to a tri-state buffer. The width of the input and output buses is specified by the VHDL generic *width* and the frame is controlled by the expression: *FOR bit_index IN 0 TO width-1 GENERATE*.



Signals which are connected to all occurrences of the objects in the generated code can be connected directly (*clk* and *en* in the example above). Separate port map frames for each component specify the connectivity expressions for each instance.

This example represents the following VHDL code:

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

ENTITY RegistersFrame IS
    GENERIC (width : positive);
    PORT (clock : IN std_logic;
          out_enable : IN std_logic;
          data_in : IN std_logic_vector(0 TO width - 1);
          data_out : OUT std_logic_vector(0 TO width - 1));
END RegistersFrame;
```

```
ARCHITECTURE struct of RegistersFrame IS
SIGNAL dat_unbuffered : std_logic;
  COMPONENT D_flipflop is
    PORT (clk : IN std_logic;
          d : IN std_logic;
          q : OUT std_logic);
  END COMPONENT;
  COMPONENT tristate_buffer IS
    PORT (a : IN std_logic;
          en : IN std_logic;
          y : OUT std_logic);
  END COMPONENT;
BEGIN
  cell_array : FOR bit_index IN 0 TO width - 1 GENERATE
    I0 : D_flipflop
      PORT MAP (
        clk => clock,
        d => data_in(bit_index),
        q => data_unbuffered
      );
    I1 : tristate_buffer
      PORT MAP (
        a => data_unbuffered,
        en => out_enable,
        y => data_out(bit_index)
      );
  END GENERATE cell_array;
END struct;
```

Note



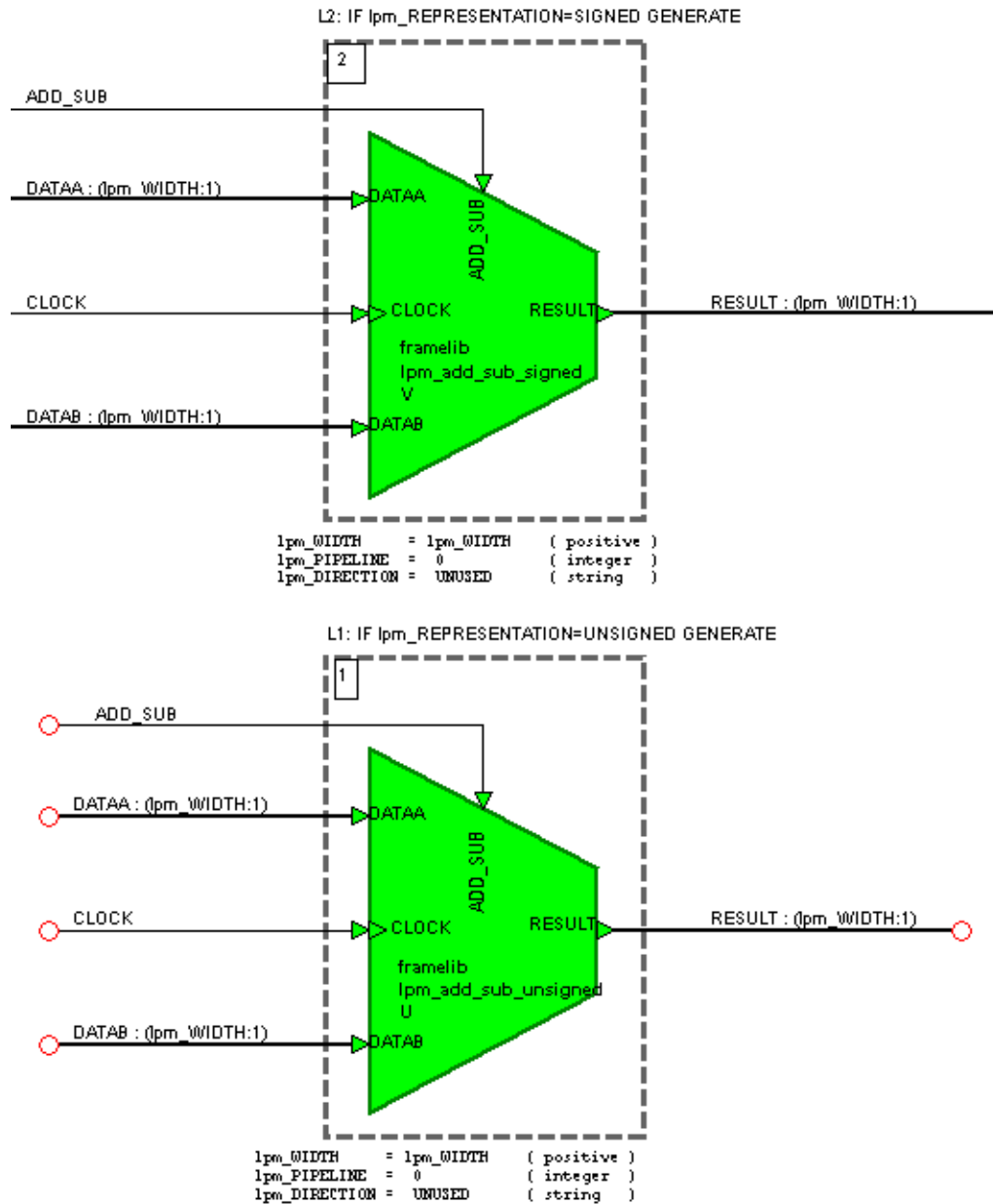
Repeating structures are not supported when you are using Verilog.

Using Generate Frames for Conditional Structures

An IF generate frame can be used in VHDL or Verilog to describe conditional or alternative structures. When you are using VHDL, there can be any number of conditional IF frames.

When you are using Verilog, an alternative structure can be defined by using an ELSE frame which must have the same label as the corresponding IF frame.

The following example instantiates different versions of the *lpm_add_sub* model determined by whether the variable *lpm_REPRESENTATION* is set to *SIGNED* or *UNSIGNED*.



This example represents the following VHDL code:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;

ENTITY lpm_add_sub IS
  GENERIC(
    lpm_WIDTH      : positive;
  
```

```
        lpm_REPRESENTATION : string := SIGNED
    );
    PORT(
        ADD_SUB : IN      std_logic := '1';
        CLOCK   : IN      std_logic := '0';
        DATAA  : IN      std_logic_vector (lpm_WIDTH DOWNT0 1);
        DATAB   : IN      std_logic_vector (lpm_WIDTH DOWNT0 1);
        RESULT  : OUT     std_logic_vector (lpm_WIDTH DOWNT0 1)
    );

END lpm_add_sub ;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;

LIBRARY framelib;

ARCHITECTURE struct OF lpm_add_sub IS

    COMPONENT lpm_add_sub_signed
    GENERIC (
        lpm_WIDTH      : positive;
        lpm_REPRESENTATION : string := SIGNED
    );
    PORT (
        ADD_SUB : IN      std_logic := '1';
        CLOCK   : IN      std_logic := '0';
        DATAA  : IN      std_logic_vector (lpm_WIDTH DOWNT0 1);
        DATAB   : IN      std_logic_vector (lpm_WIDTH DOWNT0 1);
        RESULT  : OUT     std_logic_vector (lpm_WIDTH DOWNT0 1)
    );
    END COMPONENT;
    COMPONENT lpm_add_sub_unsigned
    GENERIC (
        lpm_WIDTH      : positive;
    );
    PORT (
        ADD_SUB : IN      std_logic := '1';
        CLOCK   : IN      std_logic := '0';
        DATAA  : IN      std_logic_vector (lpm_WIDTH DOWNT0 1);
        DATAB   : IN      std_logic_vector (lpm_WIDTH DOWNT0 1);
        RESULT  : OUT     std_logic_vector (lpm_WIDTH DOWNT0 1)
    );
    END COMPONENT;

BEGIN
    L1: IF lpm_REPRESENTATION=UNSIGNED GENERATE
        U : lpm_add_sub_unsigned
        GENERIC MAP (
            lpm_WIDTH      => lpm_WIDTH,
        )
        PORT MAP (
            ADD_SUB => ADD_SUB,
            CLOCK   => CLOCK,
            DATAA  => DATAA,
            DATAB   => DATAB,
```

```

        RESULT => RESULT
    );
END GENERATE L1;

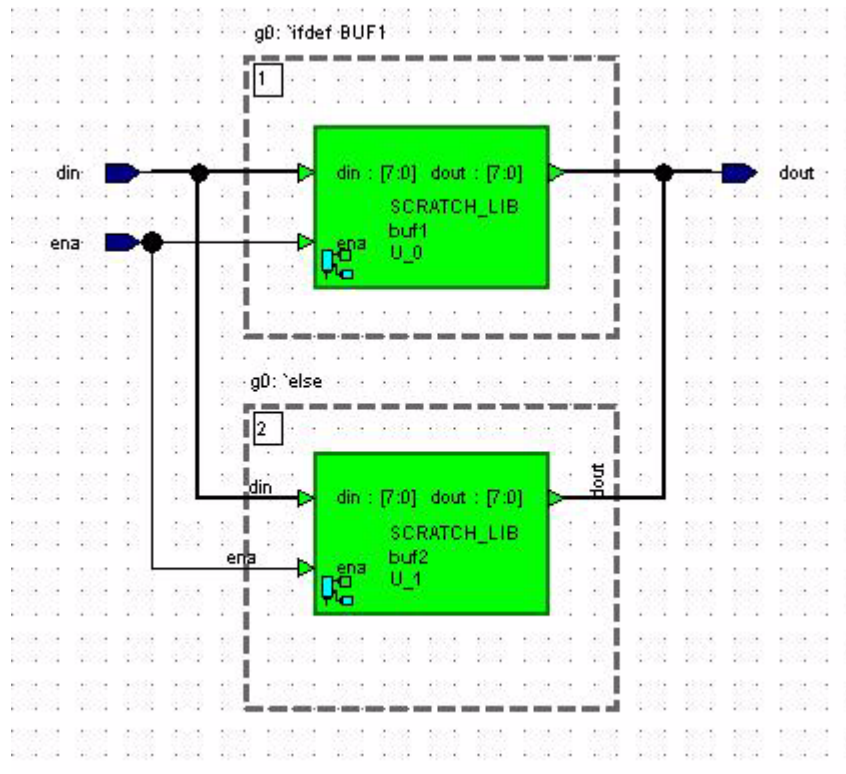
L2: IF lpm_REPRESENTATION=SIGNED GENERATE
    V : lpm_add_sub_signed
    GENERIC MAP (
        lpm_WIDTH      => lpm_WIDTH,
    )
    PORT MAP (
        ADD_SUB => ADD_SUB,
        CLOCK   => CLOCK,
        DATAA  => DATAA,
        DATAB   => DATAB,
        RESULT  => RESULT
    );
END GENERATE L2;

END struct;

```

When you are using Verilog, an alternative structure can be defined by using an ELSE frame which must have the same label as the corresponding IF frame.

The following example instantiates *Buf1* if the macro definition *BUF1* is true, otherwise *Buf2* is instantiated. Both frames have the same label *Gen1*.



This example represents the following Verilog code:

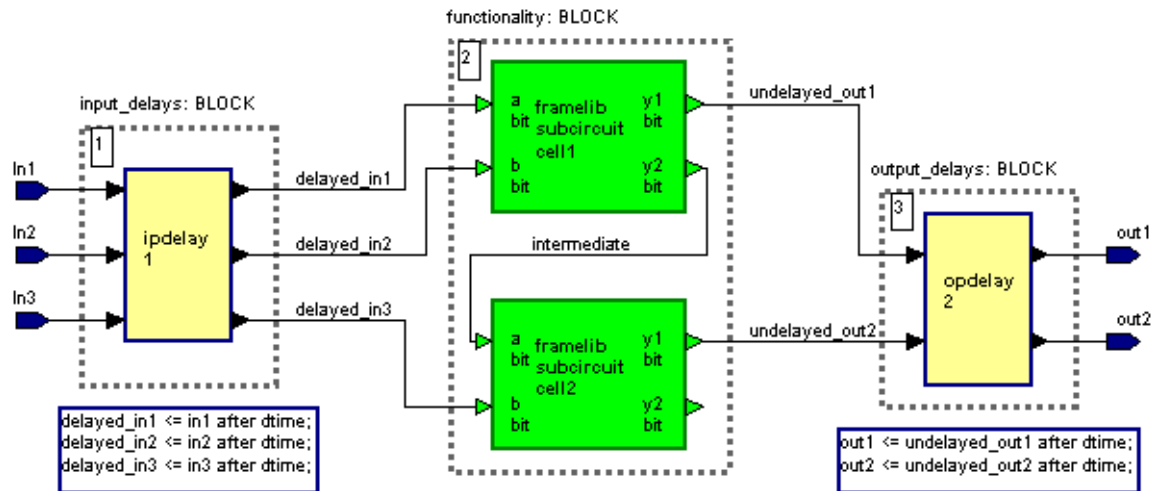
```
module tristate8(  
    ena,  
    in,  
    out  
);  
  
input      ena;  
input [7:0] in;  
output [7:0] out;  
wire      ena;  
wire [7:0] in;  
wire [7:0] out;  
  
`ifdef BUF1  
    buf1 U1(  
        .enable (ena),  
        .ip      (in),  
        .op      (out)  
    );  
`else  
    buf2 U2(  
        .enable (ena),  
        .ip      (in),  
        .op      (out)  
    );  
`endif  
  
endmodule // tristate8
```

Using a BLOCK Generate Frame

When you are using VHDL, a BLOCK generate frame can be used to cluster related concurrent statements in the generated HDL. All the HDL statements generated for the objects contained in a BLOCK frame are executed concurrently.

In the following example, separate BLOCK frames and frame declarations have been used to specify delay constants for the *ipdelay* and *opdelay* embedded blocks. The main functionality is

implemented by two instantiations of *subcircuit* which are contained within a third BLOCK frame.



This example represents the following VHDL code:

```

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

ENTITY block_circuit IS
    GENERIC(
        dtime : time := 10 ns
    );
    PORT(
        In1   : IN    bit;
        In2   : IN    bit;
        In3   : IN    bit;
        out1  : OUT   bit;
        out2  : OUT   bit
    );
END block_circuit ;

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

LIBRARY framelib;

ARCHITECTURE struct OF block_circuit IS

    SIGNAL delayed_in1    : bit;
    SIGNAL delayed_in2    : bit;
    SIGNAL delayed_in3    : bit;
    SIGNAL intermediate   : bit;
    SIGNAL undelayed_out1 : bit;
    SIGNAL undelayed_out2 : bit;

    COMPONENT subcircuit

```

```
    PORT (
        a : IN      bit ;
        b : IN      bit ;
        y1 : OUT     bit ;
        y2 : OUT     bit
    );
END COMPONENT;

BEGIN
    input_delays: BLOCK
    BEGIN
        delayed_in1 <= in1 after dtime;
        delayed_in2 <= in2 after dtime;
        delayed_in3 <= in3 after dtime;

    END BLOCK input_delays;

    functionality: BLOCK
    BEGIN
        cell1 : subcircuit
            PORT MAP (
                a => delayed_in1,
                b => delayed_in2,
                y1 => undelayed_out1,
                y2 => intermediate
            );
        cell2 : subcircuit
            PORT MAP (
                a => intermediate,
                b => delayed_in3,
                y1 => undelayed_out2,
                y2 => OPEN
            );
    END BLOCK functionality;

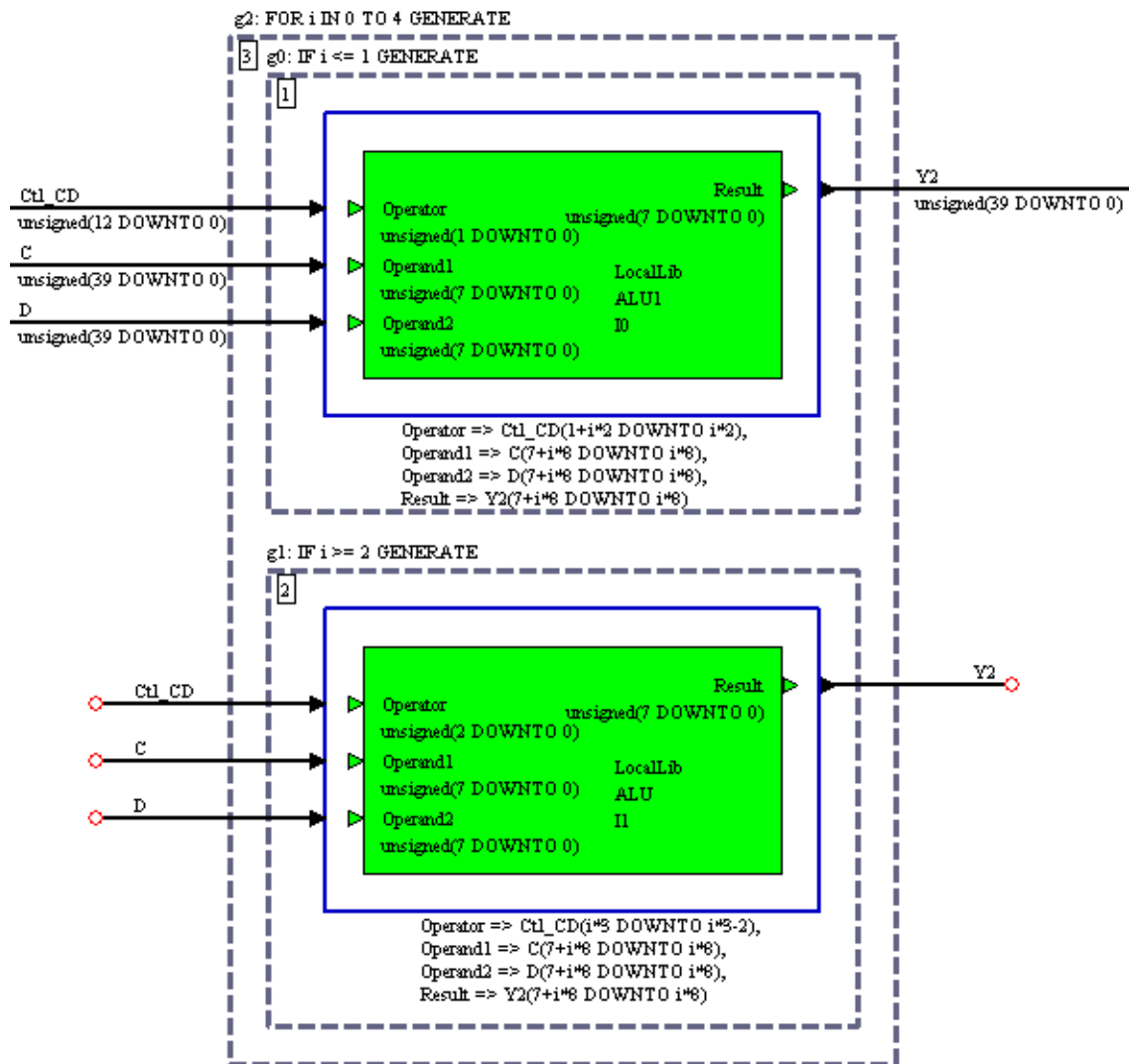
    output_delays: BLOCK
    BEGIN
        out1 <= undelayed_out1 after dtime;
        out2 <= undelayed_out2 after dtime;
    END BLOCK output_delays;

END struct;
```

Using Nested Generate Frames

A frame may be included within another frame. Typically a repeating structure may require different instantiations for parts of the structure.

The following VHDL example shows a FOR generate frame used to represent five instances where there are alternative instances determined by whether the value *i* is 0 or 1 (two instances) and 2, 3, or 4 (three instances).



This example represents the following VHDL code:

```

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

ENTITY NestedFrames IS
  PORT(
    C      : IN      unsigned (39 DOWNTO 0);
    Ctl_CD : IN      unsigned (12 DOWNTO 0);
    D      : IN      unsigned (39 DOWNTO 0);
    Y2     : OUT     unsigned (39 DOWNTO 0)
  );
END NestedFrames ;

```

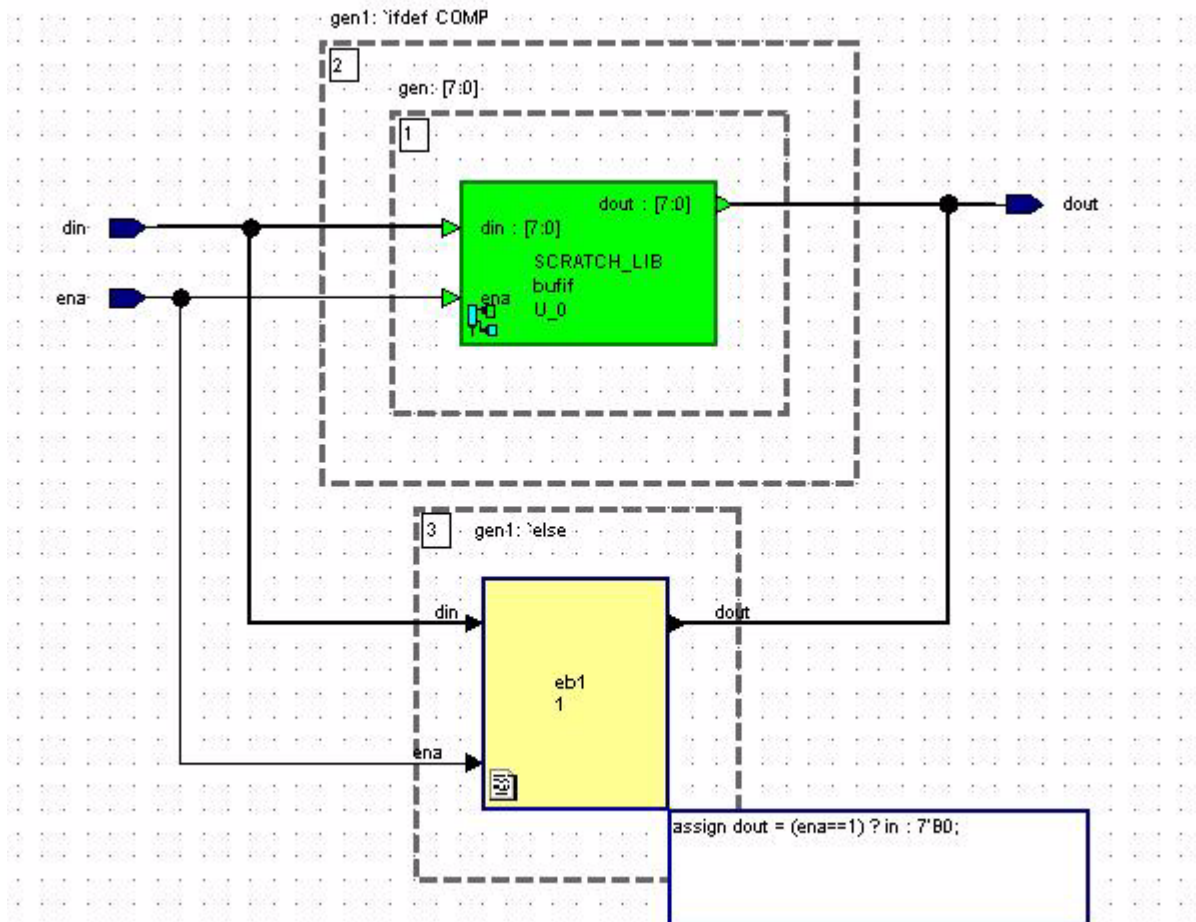
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ARCHITECTURE struct OF NestedFrames IS

    COMPONENT ALU1
    PORT (
        Operand1 : IN      unsigned (7 DOWNTO 0);
        Operand2 : IN      unsigned (7 DOWNTO 0);
        Operator  : IN      unsigned (1 DOWNTO 0);
        Result    : OUT     unsigned (7 DOWNTO 0)
    );
    END COMPONENT;
    COMPONENT ALU2
    PORT (
        Operand1 : IN      unsigned (7 DOWNTO 0);
        Operand2 : IN      unsigned (7 DOWNTO 0);
        Operator  : IN      unsigned (2 DOWNTO 0);
        Result    : OUT     unsigned (7 DOWNTO 0)
    );
    END COMPONENT;

BEGIN
    g2: FOR i IN 0 TO 4 GENERATE
        g0: IF i <= 1 GENERATE
            I0 : ALU1
            PORT MAP (
                Operator => Ctl_CD(1+i*2 DOWNTO i*2),
                Operand1 => C(7+i*8 DOWNTO i*8),
                Operand2 => D(7+i*8 DOWNTO i*8),
                Result  => Y2(7+i*8 DOWNTO i*8)
            );
        END GENERATE g0;
        g1: IF i >= 2 GENERATE
            I1 : ALU2
            PORT MAP (
                Operator => Ctl_CD(i*3 DOWNTO i*3-2),
                Operand1 => C(7+i*8 DOWNTO i*8),
                Operand2 => D(7+i*8 DOWNTO i*8),
                Result  => Y2(7+i*8 DOWNTO i*8)
            );
        END GENERATE g1;
    END GENERATE g2;
END struct;
```

You cannot include another frame within a FOR frame when using Verilog. However, you can have any level of nested IF and ELSE frames.



This example represents the following Verilog code:



```
module tristate_vlog(
    ena,
    in,
    out
);

input      ena;
input [7:0] in;
output [7:0] out;
wire      ena;
wire [7:0] in;
wire [7:0] out;

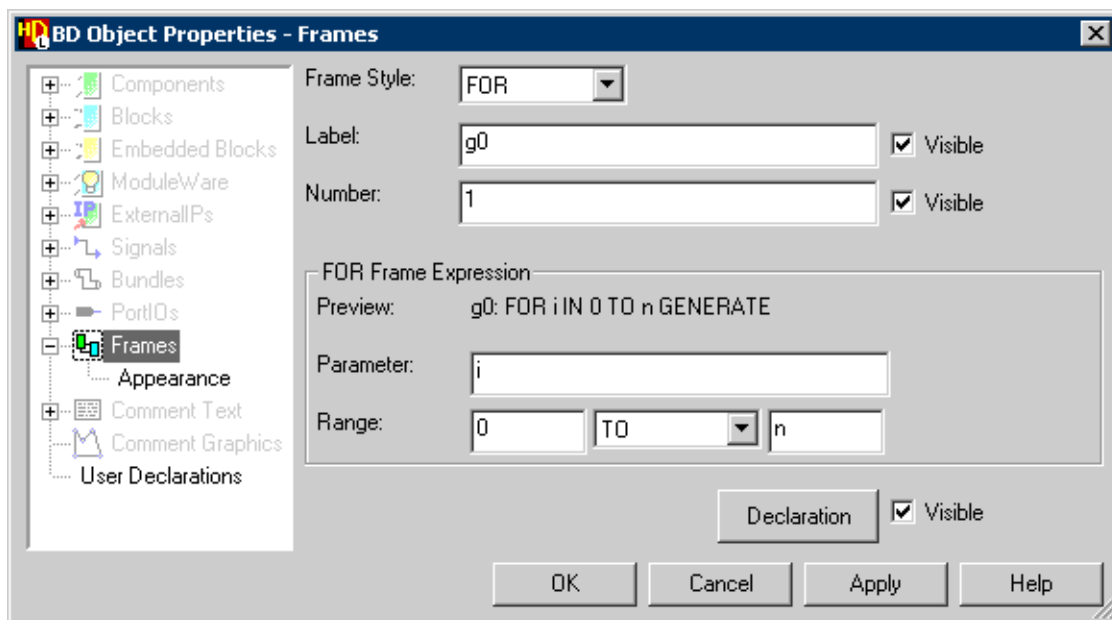
`ifdef COMP
    bufif U1[7:0](
        .enable (ena),
        .ip      (in),
        .op      (out)
    );
`endif
```

```
);  
  
`else  
    // HDL Embedded Block 1 ebl  
    assign out = (ena==1) ? in : 7'B0;  
`endif  
  
endmodule // tristate_vlog
```

Editing Generate Frame Properties

You can display and edit properties for generate frames in a block diagram or IBD view by using the  button, **Alt** +  shortcut or choosing **Object Properties** from the **Edit** menu. In IBD views the Frame number and Appearance fields do not exist.

If a frame is selected, the **Frames** page of the Object Properties dialog box is displayed listing its existing properties.



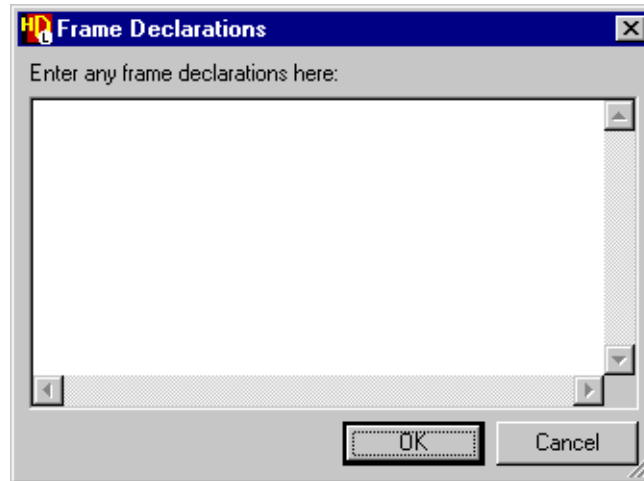
You can use the dialog box to edit the frame style, label, number and expression and specify whether the title (including the label and expression) and number are visible on the diagram.

The style can be selected from a pulldown list which includes FOR, IF and ELSE when using Verilog or FOR, IF and BLOCK when using VHDL.

Refer to the dialog box description for details of the options and syntax supported for each of these styles.

When you are using VHDL, you can use the **Declarations** button (or choose **Frame Declarations** from the popup menu in an IBD view) to display a free-format entry Frame

Declarations dialog box which allows you to specify local declarations which apply only within the frame.



You can also choose whether these declarations are visible on a block diagram. Frame declarations should only be used in a FOR or IF frame when **Other Dialect** is set as a preference in the **Style** tab of the VHDL Options dialog boxes and can be specified in a BLOCK frame when you are using any VHDL dialect.

The syntax of the declarations is checked on entry.



Tip: You can declare procedures, functions, shared variables, constants or other declarations which are allowed in generate statements for the current VHDL dialect. However, signal declarations local to generate frames are not supported.

Chapter 7


Component Interface Views


This chapter describes procedures for using the *tabular IO* and *symbol* views to edit a component interface.

Opening a Component Interface	305
Tabular IO and Symbol Views	306
Tabular IO Notation	307
Hiding Columns	308
Filtering Columns	309
Tabular IO Toolbar	310
Sorting the Rows in a Tabular IO View	311
Adding Ports in the Tabular IO View	311
Grouping Port Rows	312
Setting Visual Attributes in the Tabular IO View	314
Symbol Notation	315
Symbol Toolbar	315
Adding Ports in the Symbol View	316
Customizing a Symbol	317
Editing Port Declarations	318
Changing the Port Declaration Order	319
Propagating Port Changes	320
Updating Instances	320
Adding Attributes to a Port Declaration	321
Adding Comments to a Port Declaration	321
Editing Symbol Generic or Parameter Declarations	322
Editing Symbol/Interface Object Properties	324
Editing Symbol User Declarations	324
Editing Symbol Body Properties	325
Setting Interface Preferences	327

Opening a Component Interface


To create a new interface using the Design Content wizard:

1. Invoke the Design Content Creation Wizard by doing one of the following:
 - Choose **Design Content** from the **New** cascade of the **File** menu.
 - Click the  icon in the Design Manager left side bar.

- Click the  icon in the shortcut bar to display a menu and choose **Design Content** or choose **Graphical View** and select **Interface** from the cascade menu.
- 2. Choose Graphical View from the Categories pane and Interface from the File Type pane.

You can also choose **Interface** from the **Open As** cascade of the component popup menu.


To create a new interface:

Click the  icon in the shortcut bar and choose **Interface** from the **Graphical View** cascade menu to open a tabular IO view.

Refer to “Creating Design Views” in the *HDL Designer Series User Manual* for detailed information about creating new views.

To edit a component interface:

Do one of the following

- Select the *symbol* in the *design explorer* *Design Units* view and choose **Open File** from the **File** or popup menu.
- Select a component instance from a *block diagram* or *IBD view* and then open down by choosing **Interface** from the **Open As** cascade of the popup menu to edit a child component interface.
- Use the  button or choose **Open Up** or **Interface** from the **Open** cascade of the **File** menu to edit a parent interface for a block diagram, IBD view, state diagram, flow chart or truth table.

If the view is a *block* view, the parent block diagram or IBD view is opened. However, if it is a component, the component interface is opened.

When you open up, the interface is opened as a *tabular IO* view or as a *symbol* diagram view depending on an option set in the interface master preferences.

Note



If a component is described by a block diagram or IBD view, the component interface can also be updated by reconciling the interface from the block diagram as described in “[Reconciling Interfaces](#)” on page 128.

Tabular IO and Symbol Views

The *tabular IO* and *symbol* views are alternative views of the component interface. The tabular IO view is typically used to define the interface and the symbol view is used to define the graphical view when the component is instantiated in a block diagram.

You can display the tabular IO interface or symbol diagram view by choosing **Interface** or **Symbol** or in the diagram browser as described in “[Browsing Diagram Structure](#)” on page 92.

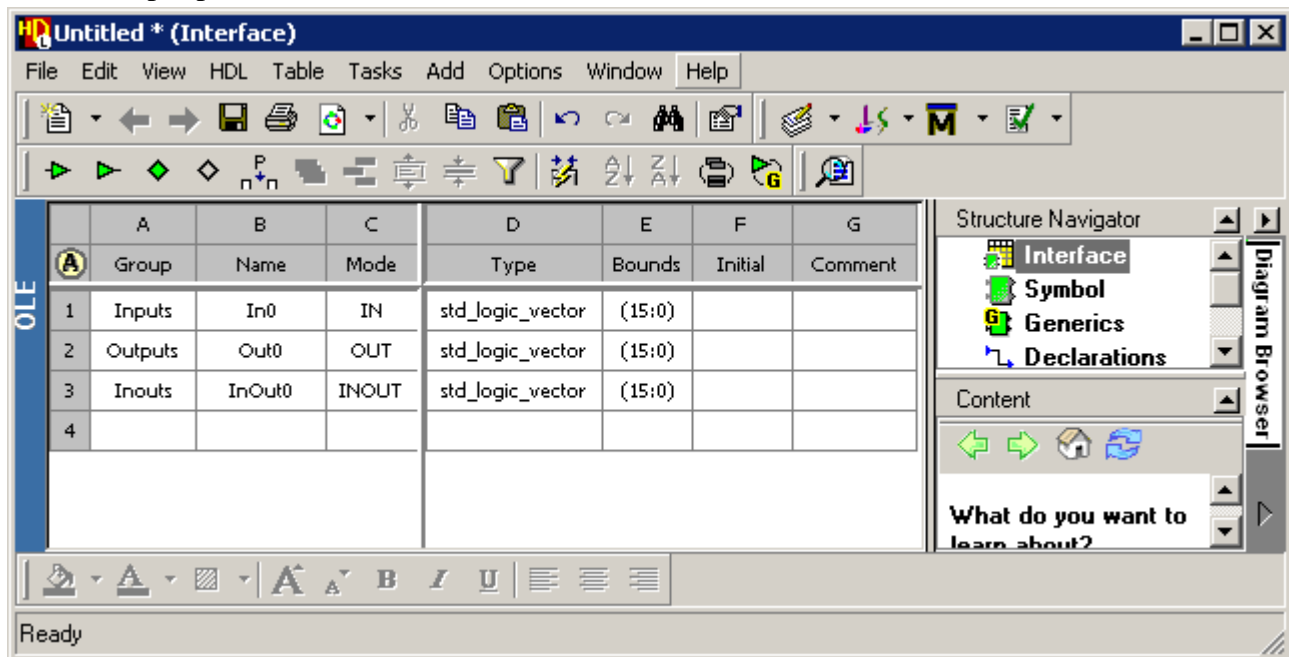
You can enter the interface port declarations for a component by using either view. Any changes made in one view are automatically made in the alternative view. Default port placement is used for the symbol when you add ports in the tabular IO view.

When you are using VHDL, you can set [VHDL package](#) references by double-clicking over the package list in the symbol view, or by choosing **Package References** from the **Diagram** menu in the symbol view or from the **Table** menu in the tabular IO view. Refer to “[Setting Package References](#)” on page 23 for more information.

Tabular IO Notation

The tabular IO view (when the **Ports** page is active) displays the port declarations for the interface as a matrix of seven columns with a separate row for each port.

For example, the following picture shows the interface to a Verilog symbol with four input and two output ports:



The following columns are normally displayed:

Group	Named groups of rows can be selectively displayed.
Name	Port name.
Mode	Port mode: input, output, bi-directional, buffer (VHDL only) or local.
Type	VHDL type definition or Verilog net type.


Bounds	Range of the specified type (may use short or long format for VHDL).
Delay	Delay value for a Verilog port.
Initial	Initial value of a VHDL port.
Comment	Comment appended to a port declaration.

For Verilog 2005 designs to extra columns are displayed:


Signed	Specify whether a net is signed or not.
Value	Initial value for a Verilog 2005 port.

If any synthesis properties have been added to a port declaration (for example, the *input_drive* column in the picture above), these are shown as additional columns.

Horizontal and vertical sashes divide the tabular IO view into scrolling and non-scrolling regions. Vertical and horizontal scroll bars are automatically displayed if the scrolling area does not fit in the current window.

You can change the non-scrolling area by moving the cursor over the sash between the scrolling areas in the letter row or number column. The cursor changes to  and allows you to move the sash by pressing down the **Left** mouse button and dragging it between the required column or row.

You can move a column or row (or several selected columns or rows) by pressing down the **Right** mouse button and dragging them to the required position. However, you cannot move the header row or the Group column and there must always be at least two columns in the non-scrolling area.

You can resize any column by using the  button to fit the column width to the text in the selected cell (or cells) or by dragging the sashes between the columns.

Note



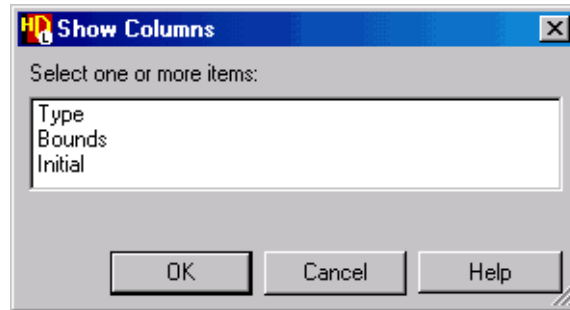
Note that you can select an entire row by clicking the row number or an entire column by clicking the column letter. You can also select the entire table by clicking on cell A1.

Hiding Columns

You can hide the selected column (or columns) by choosing **Hide Column** from the popup menu or the **Columns** cascade of the **Table** menu.


If one or more columns are hidden, you can display a dialog box listing the hidden columns by choosing **Show Column** from popup menu or the **Columns** cascade of the **Table** menu. For

example, the following Show Columns dialog box is displayed if the Type, Bounds and Initial columns are hidden:



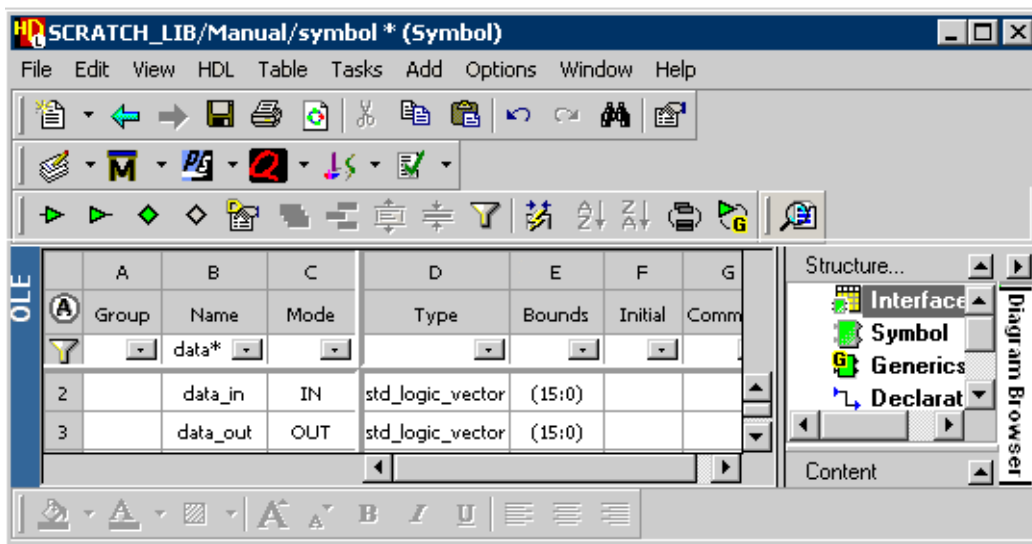
You can select one or more columns and confirm the dialog box to make them visible in the table.

Filtering Columns

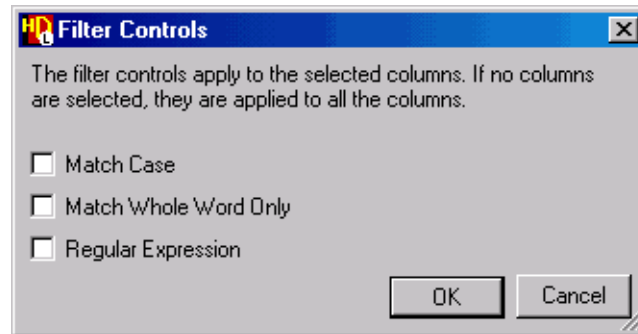
You can filter the content of any column by setting the **Filter** option in the **Table** menu or by using the  button to toggle the current filter mode. When this option is set a dropdown menu is available from an additional filter row in each column.

You can choose from any of the values in the dropdown menu to filter the column content. For example, if you choose *wire* in the Type column for a Verilog view, only ports with wire type are displayed.

You can also enter a simple match string in the drop entry box to display only matching ports. For example, you could enter *data** in the Name column to display only ports starting with the characters *data* as show by the following example.



You can apply filters to more than one column or set options to match case, match whole words or use regular expressions by choosing **Filter Settings** from the **Table** or popup menu to display the Filter Controls dialog box:
















The filter controls are applied to the currently selected columns or to all columns if none are selected.

Note that a list of supported regular expressions is available from the **Quick Reference Index** which can be accessed through the Help and Manuals tab of the HDS InfoHub. To open the InfoHub, select **Help and Manuals** from the **Help** menu.

Tabular IO Toolbar

The following commands are available from the Tabular IO Tools toolbar:


Table 7-1. Tabular IO Toolbar


Icon	Description
	Add an input port
	Add an output port
	Add a bidirectional (inout) port
	Group the selected rows or add a group
	Ungroup
	Expand all groups
	Collapse all groups
	Toggle Filter
	Fit the cell width to the contents of the selected cell
	Sort in ascending order
	Sort in descending order
	Toggle between grouped and ungrouped mode
	Toggle between the port and Verilog parameters table (OLE view only)

The toolbar can be displayed or hidden by setting the **Tabular IO Tools** option in the **Toolbars** cascade of the **View** menu.

Refer to “[Toolbars](#)” on page 20 for more information about toolbars including the standard toolbar buttons which are available in more than one editor.

Sorting the Rows in a Tabular IO View





You can sort the rows in a selected column of the tabular IO view in ascending alphanumeric order of the cell data by using the  button or choosing **Sort Ascending** from the popup menu or the **Columns** cascade of the **Table** menu.

Alternatively, you can sort the data in descending order by using the  button or by choosing **Sort Descending** from the popup menu or the **Columns** cascade of the **Table** menu.

Adding Ports in the Tabular IO View

You can add ports to a component interface using the **Add** menu or the following buttons in the tabular IO view:

Table 7-2. Tabular IO View Commands for Adding Ports

Button	Function Key	Description
	F8	Add an input port
	F9	Add an output port
	F11	Add a bidirectional (inout) port
	F12	Add a buffer port (VHDL only)

The port is added in the next available row with default name, type and bounds.

Alternatively, you can add ports by entering a declaration directly into the next row of Name, Mode, Type and Bounds cells. The mode defaults to the last mode used or you can choose from a list of available modes: input, output, bidirectional (inout) or buffer (VHDL only).

The type defaults to the last type used or you can choose from a dropdown list of available types in the Type column. The bounds defaults to the last range used or you can choose from a dropdown list of recently entered ranges in the Bounds column.

A VHDL bounds can be entered in long or short format. The display format can be set by setting or unsetting the **Short Form** option in the **Table** menu.

If you enter a port name followed by a valid bounds constraint, for example, *myport(7 DOWNT0 0)*, the constraint is automatically moved to the Bounds column.



Tip: You can automatically complete a row with default properties by using the key after entering a port name to move to the name cell in the next row.




You can optionally enter a value in the Initial (VHDL) or Delay (Verilog) and Comment columns. The delay or initial value can be chosen from a dropdown list of recently entered values.

If you enter characters that match characters in an existing entry of the same column, the remaining characters are entered automatically.

If you do not change the name of a port, each new port name is made unique by adding an integer to the default name. (For example: *In*, *In1*, *In2*...).

Grouping Port Rows

You can group rows in the tabular IO view by selecting a row or rows and using the  button or by choosing **Group** from the popup menu, the **Add** menu or from the **Group** cascade of the **Table** menu.

The selected rows are added to a new group with the default name *GroupN* (where *N* is automatically incremented if it already exists).

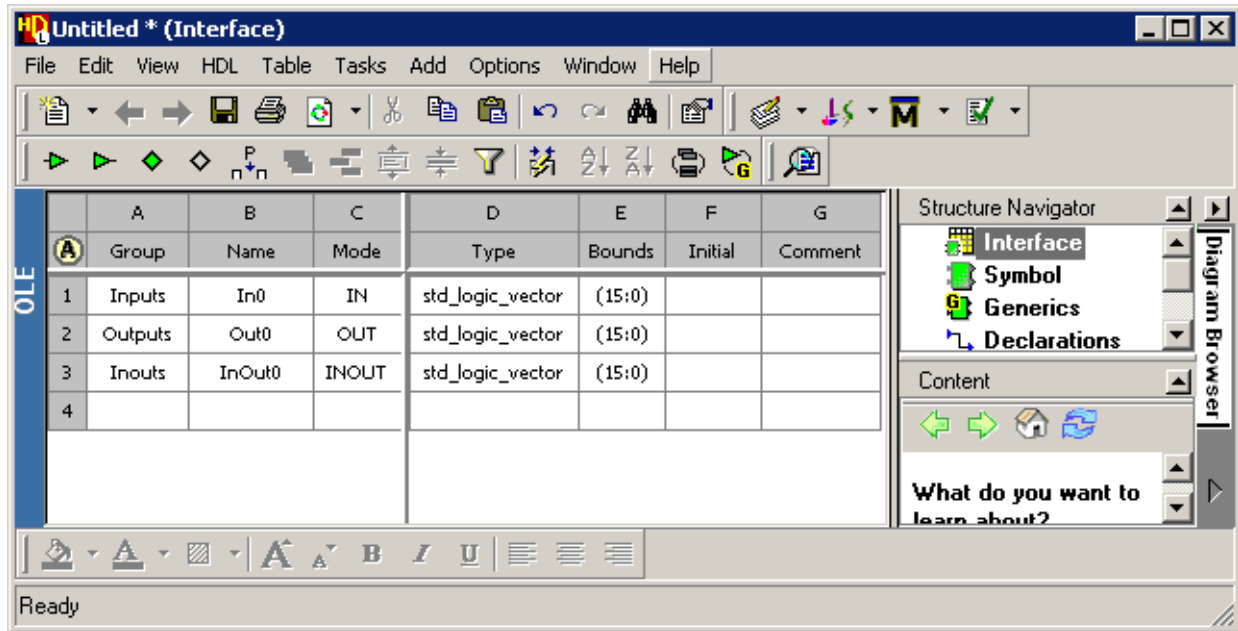
You can also add a group or create a new group by entering a name in the Group column for the ports you want to group.

Note




You can choose from a dropdown list of existing groups. If you type a partial string that matches the name of an existing group the name is automatically completed.


The following example shows groups defined for *Inputs*, *Outputs* and *InOuts*. The *Inputs* and *InOuts* groups are shown collapsed but the *Outputs* group is shown expanded. When grouped mode is set, you can enter a multi-line comment in the group row as shown below.






Tip: Note that you can select a port declaration row (or rows) and drag and drop the row(s) into an existing group.

You can remove a group name by selecting a row (or rows) and using the  button, choosing **UnGroup** from the popup menu, the **Group** cascade of the **Table** menu or by deleting the name from the Group cell.

If you rename or remove an existing group cell and the group is no longer referenced, you are prompted to delete the old group name.

When the view includes one or more groups, you can use the  button or set **Show Grouped** in the **Table** menu to toggle between grouped and ungrouped mode. All rows are displayed normally in flat mode but rows in the same group are shown as a single (but expandable) group in hierarchy mode.

You can expand all the group rows by using the  button or choosing **Expand All Groups** from the **Group** cascade of the **Table** menu. Alternatively, you can expand an individual group by clicking on the  icon.

You can collapse all the group rows by using the  button or choosing **Collapse All Groups** from the **Group** cascade of the **Table** menu.

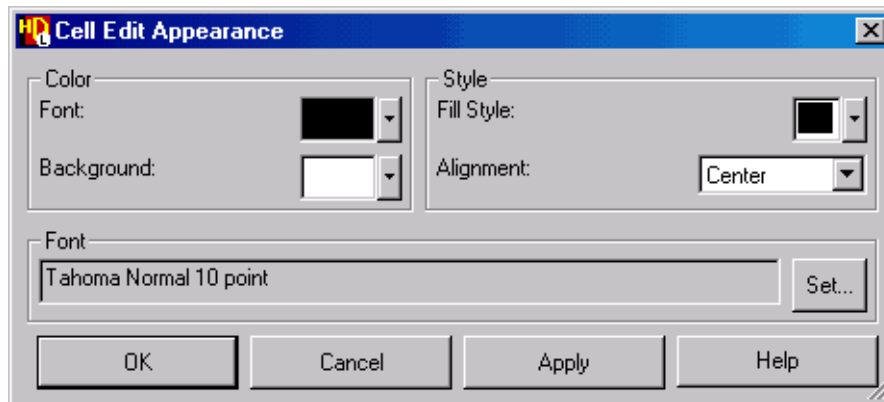
Note



If you delete a group, all its contents are deleted. If you copy a group, all ports in the copied group are automatically made unique.

Setting Visual Attributes in the Tabular IO View

You can set the visual attributes for the selected cell (or selected cells) by choosing **Appearance** from the **Edit** or popup menu. The Cell Edit Appearance dialog box is displayed:

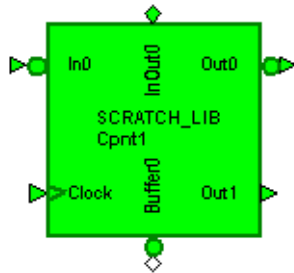




The dialog box allows you to set the font and cell colors. You can also set the fill style, default text font and cell text alignment.



You can also edit the visual attributes used in the table using the Appearance toolbar. Refer to [“Setting Visual Attributes”](#) on page 83 for more information.


Symbol Notation


When a component is defined by a child block diagram or IBD view, the ports shown on the symbol correspond to ports on the child view.



Input ports are shown by  entering the symbol and output ports by  exiting from the symbol.

A bidirectional (InOut) port is indicated by  and a buffer port by .

Any input port can be shown as an edge triggered clock signal (indicated by .

Any input, output, bidirectional or buffer port can be shown as an active low (Not) signal (indicated by .

By default the port name and other properties text is shown inside the symbol body but can optionally be moved outside the symbol.









You can choose to hide properties text in the symbol view or when the symbol is instantiated in a block diagram. You can also choose to automatically hide connected ports when the symbol is instantiated. (This option is typically used when the clocked and inverted attributes are set.)




When a symbol is automatically created, input ports are placed on the left, output ports on the right, bidirectional ports on the bottom edge and buffer ports on the top edge but these positions can be changed when you edit the symbol. Note that vertical text is automatically used for ports on the upper and lower edges.

Symbol Toolbar

The following commands are available from the Symbol Tools toolbar:

Table 7-3. Symbol Toolbar

Icon	Description
	Select text and shapes, text only or shapes only
	Add or modify comment text
	Pan the window
	Add an input port
	Add an output port
	Add a bidirectional (inout) port
	Add a buffer port (VHDL only)
	Add a panel

The  button sets normal selection mode (text or shapes) but has a pulldown menu which allows you to select text only  or shapes only . The button changes to indicate the active selection mode.





Refer to “[Adding Comment Text](#)” on page 56 for information about adding comment text and “[Panels](#)” on page 78 for information about adding panels.

Refer to “[Toolbars](#)” on page 20 for more information about toolbars including the standard toolbar buttons which are available in more than one editor.

Adding Ports in the Symbol View

You can add ports to a component interface using the **Add** menu or the following buttons in the symbol view:

Table 7-4. Symbol View Commands for Adding Ports

Button	Function Key	Mnemonic	Description
	F8	I	Add an input port
	F9	O	Add an output port
	F11	T	Add a bidirectional (inout) port
	F12	F	Add a buffer port (VHDL only)

The cursor changes to a cross-hair which allows you to add a port by clicking near the required location on the symbol outline.

The port is added to the symbol outline at the point nearest to the cursor. By default, the port name, type and bounds constraint are added inside the symbol alongside the new port but the text can be moved outside the outline.

Note that the command normally repeats until you use the **Esc** key (or **Right** mouse button) to terminate the repeating command.

The ports are added with default type and bounds but these properties can be changed by directly editing the text on the diagram or by double-clicking on the port declarations to display the tabular IO view.



You can change the mode of a port in the symbol view by choosing **Change Mode** from the popup menu (or **Ports** cascade of the **Diagram** menu) which is displayed when you use the **Right** mouse button with a single port selected.

If you do not change the name of a port, each new port is given a unique name by adding an integer to the default name. (For example: *In*, *In1*, *In2*...).

You can move ports around the symbol body by dragging with the **Left** mouse button. (If you select more than one port, their separation is preserved.)

You can space all the ports on each edge of a symbol evenly by choosing **Equidistant Ports** from the popup menu when the symbol body is selected or space the selected ports when one or more parts are selected.

The port name (and type constraints if visible) is aligned with the port orientation. Thus, the name text is horizontal for a port on the left or right of the symbol and vertical for a port on the top or bottom of the symbol. However, the text can be rotated independently by choosing **Rotate Text** from the popup menu.

You can add an active low  (Not) or edge-triggered clock  indicator on a symbol port as described in [“Indicating Not or Clocked Ports”](#) on page 231.

Customizing a Symbol

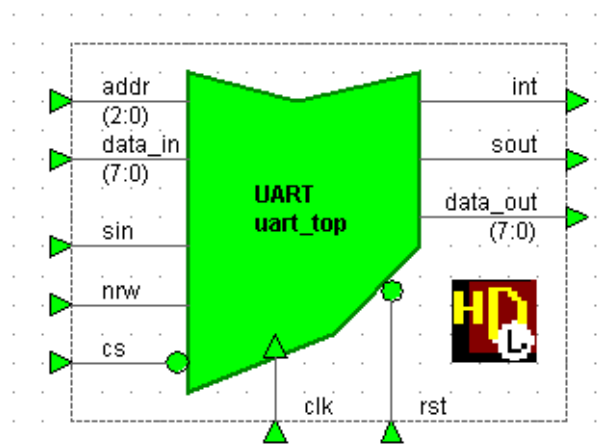
You can customize a symbol by checking **Custom Symbol** in the **Diagram** or popup menu in the symbol view or by checking the **Custom Symbol** option in the **Symbol** tab of the Symbol Object Properties dialog box.

When this option is active, the symbol body is replaced by a resizable rectangular boundary containing an editable comment graphic.

Any comment graphics object can be superimposed on the default shape or you can use the **Edit Vertices** command from the popup menu to modify the default shape.

Any comment graphics or comment text completely within the boundary is part of the custom symbol and will be displayed when the symbol is instantiated in a block diagram.

The following example shows a custom symbol for the *uart_top* design unit (with autowhiskers enabled and a HDS logo added as a custom graphics bitmap):



Comment graphics or text outside the boundary is visible only inside the symbol editor.

Refer to [“Adding Comment Graphics”](#) on page 72 for more information about using comment graphics.

You can check **Lock** from the **Diagram** or popup menu to lock all objects inside the symbol boundary so that they behave as a single object. (Note however, that all objects inside the boundary of an autoshape are automatically included.)

Ports can be added to the symbol boundary in the normal way. If **Autowhisker** is checked in the **Diagram** or popup menu or in the **Symbol** tab of the Symbol Object Properties dialog box.


Whiskers are automatically added as orthogonal lines between the ports on the symbol boundary and the contained comment graphics shape. However, no whisker is added if the line would not intersect with the contained comment graphics.

You can also change the shape of a symbol by applying a standard logic shape as described in [“Choosing a Standard Shape”](#) on page 229.

Editing Port Declarations

You can edit port declarations by direct text editing in the symbol view or by editing the cells in the tabular IO view.

You can edit the port name, mode, type, bounds, VHDL initial value or Verilog delay and comments for an existing port in the tabular IO view by editing the cell contents.

Note that you can choose from a pulldown list of standard VHDL or Verilog types by clicking in a Type cell and using the  button. The type which most closely matches the current string is automatically selected in the list. (For example, if the characters *st* are entered for a VHDL view, the type *std_logic* is selected.)

If you are using VHDL, the type must be defined in a VHDL package, referenced in the package references or be one of the standard predefined types.

The pulldown list includes the most commonly used types and any other types which have already been used in the view. However, you may need to add a new package reference if you choose a type which is not in the currently referenced packages. For example, *ieee.numeric_std* should be referenced if you want to use the *signed* or *unsigned* types.

The standard Verilog types can be selected if you are using Verilog. However, input or bidirectional ports cannot have type *reg* and automatically default to *wire*.

The bounds can be used to specify the indexes for the elements in an array or the range for a vector type. For example: *15 DOWNT0 0* or *0 TO 7* (if you are using VHDL); *15:0* or *0:7* (if you are using Verilog).

If you are using VHDL, you can also enter a user specified constraint such as an enumerated or integer type name or you can enter an array name or type of the form: *<array>'RANGE* or *<array>'REVERSE_RANGE*.


If you are using VHDL, you can use the Initial column to enter an expression defining an initial value for the net connected to the port.

Note



VHDL can be generated for a block diagram containing component interfaces with unconnected ports if the ports have been assigned an initial value. However, HDL generation issues an error for unconnected ports without an initial value.

If you are using Verilog, you can use the Delay column to enter a delay as one, two or three values. If two or three delay values are required, they must be separated by commas and enclosed in parentheses. For example: *(delay1, delay2, delay3)*.

Changing the Port Declaration Order

The port declarations are normally ordered automatically by mode (in, out, bidirectional or buffer) and alphanumeric name as they are added. This mode is indicated by an  icon in the first cell of the title row in the tabular IO view.

You can change the order of the port declaration rows in the table by selecting a row (or rows) and dragging it to a new position with the **Right** mouse button.


You can enable manual ordering in the tabular IO view by choosing **Switch to Manual** from the popup menu in the  cell (or **Manual** from the **Port Ordering** cascade of the **Table** menu). This mode is indicated by an  icon.

When manual ordering is enabled, you can choose **Update Generation Order** from the **Port Ordering** cascade of the **Table** or popup menu to update the declaration order to be the same as the row order. Alternatively, choose **Show Generation Order** to re-order the table rows in declaration order.

The new order is preserved on the symbol and in the generated HDL.

Note

Manual ordering is automatically set if you synchronize component interface ports with a text view to preserve the port ordering specified in the text view.
Manual ordering is also used when diagrams are created by [HDL2Graphics](#) in order to preserve the order (and any in-line comments) from the source HDL code.

If you choose **Switch to Automatic** from the popup menu in the  cell (or **Automatic** from the **Port Ordering** cascade of the **Table** menu, the original ordering is discarded and the declarations are sorted automatically by mode and alphanumeric name.

Propagating Port Changes

When you edit a port declaration in the tabular IO or symbol view, you can choose whether the changes are applied only to the interface or are propagated to connected nets in the views hierarchically below the symbol.

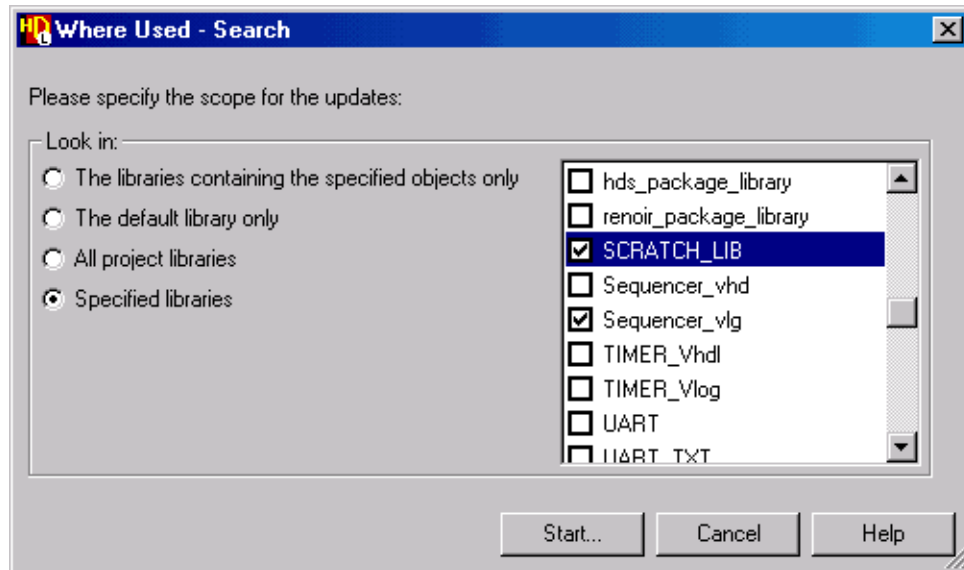
You can set **Hierarchical** or **Non Hierarchical** as the **Scope for Changes** from the popup menu or from the **Ports** cascade of the **Table** menu in the tabular IO view or the **Diagram** menu in the symbol view.

If hierarchical scope is set, the Net Propagation Options dialog box is displayed after you apply changes to the port properties for you to choose how the changes are propagated to nets in other views.

Refer to [“Propagating Net Changes”](#) on page 166 and [“Inserting and Removing Nets”](#) on page 169 for more information about propagating new signals and changes to the properties of a net.

Updating Instances

If you save a component interface after you have changed a port declaration or any of the symbol properties, you are prompted whether to update any instances where the component is used. If you answer **Yes** to the prompt, the Search page of the Where Used wizard is displayed:

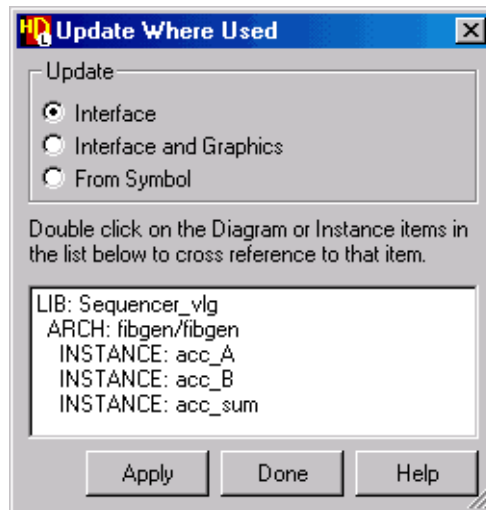


The Search page allows you to specify the search scope for referenced components.

You can choose to search for references in the same libraries as the selected objects, the default library only, all libraries in the current project or in specified libraries.

If you choose to specify libraries, the selection list is enabled and allows you to choose from any of the available libraries mapped for the current project.

The specified libraries are searched when you use the **Start** button and the Update Where Used dialog box is displayed listing any occurrences where an instance of the component is instantiated:



You can choose to update the interface only or also update any changes to the symbol graphics while preserving the symbol size and port positions. You can also choose to update all changes from the symbol. This option does a literal replacement of the symbol and nets may be moved if any ports have been moved on the symbol.

Adding Attributes to a Port Declaration

You can add attributes to a port declaration in a tabular IO view by choosing **Attributes** from the popup when one or more port rows are selected in the table matrix.

You can add attributes to a port declaration in the symbol view by choosing **Attributes** from the popup menu when the port or its declaration is selected.

Refer to [“Setting Attributes and Embedded Constraints”](#) on page 165 for more information.

Adding Comments to a Port Declaration

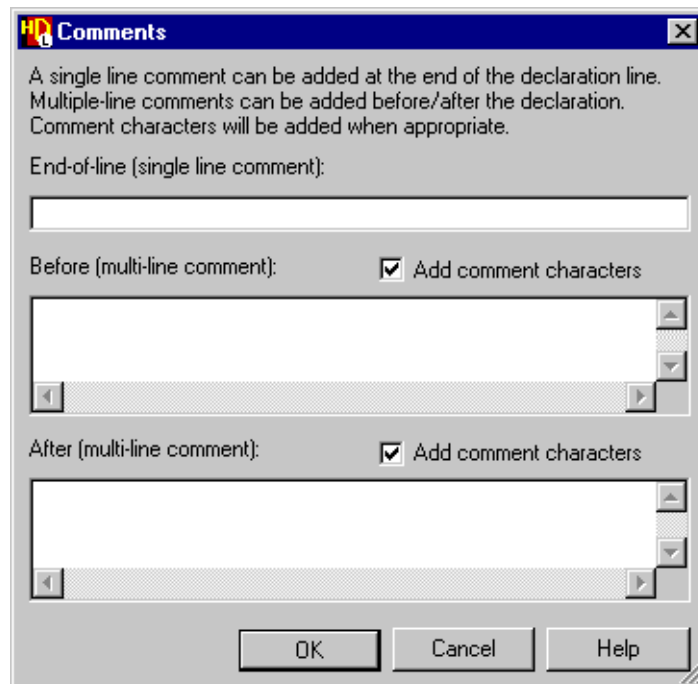
You can add comments to a port declaration in the tabular IO view by choosing **Comments** from the popup menu when the port declaration row is selected.

You can add comments to a port declaration in the symbol view by choosing **Comments** from the popup menu when the port or its declaration is selected.

A free-format entry Comments dialog box is displayed which allows you to add a single line comment at the end of the declaration or you can enter a multi-line comment to be included before or after the declaration.

Comment characters for the current hardware description language (VHDL or Verilog) are automatically inserted if the **Add comment characters** check box is set. When this option is unset, the comments must be valid HDL statements and are automatically syntax checked if checking is enabled.

The comments are displayed in the port declarations list in the symbol view. If a declaration is deleted, the corresponding comments are also deleted.



Although multi-line comments can be added to a tabular IO using the dialog box, these comments are not displayed in the table. However, end-of-line comments can be edited directly in the Comment column for the port declaration row.

Editing Symbol Generic or Parameter Declarations

A separate page in the interface view can be used to declare Verilog parameters (if the language is Verilog) or VHDL generics (if the language is VHDL). These pages can be opened from the diagram browser or by choosing **Generics** or **Parameters** from the **Diagram** menu in the symbol view.

Note



The **Generics** page is available if you are using VHDL or the **Parameters** page is available if you are using Verilog.

To set a new declaration, enter a name, type (not required when the language is Verilog) and default value in the table cells.

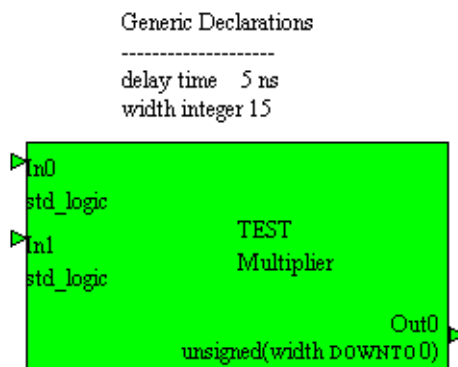
For example, the following picture shows VHDL generics *width* and *delay* defined in the **Generics** page:

	A	B	C	D	E	F
	Group	Name	Type	Value	Pragma	Comment
1		width	integer	15	NO	
2		delay	time	5	NO	
3						

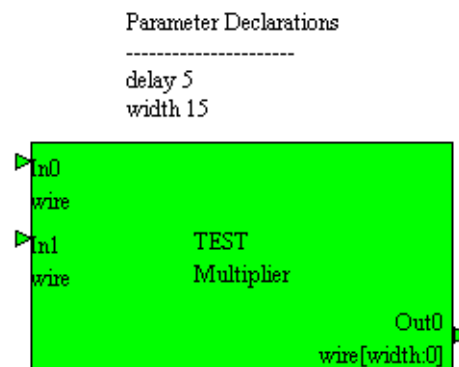
The *width* generic is used to specify the upper bounds for the *output* bus and *delay* specifies a variable value for an internal signal used within the component.

The new declarations are shown as a text object anchored to the symbol body in the symbol view. For example:

VHDL



Verilog



Note





The default value specified for a declaration can be a discrete value or an expression. Note that the *time* type used in the example above may not be supported by some compilers (for example, Synopsys Design Compiler).

Note that you can choose from a pulldown list of standard VHDL types in the Type column.

Refer to [“Generics and Parameters”](#) on page 174 for more information about using VHDL generics and Verilog Parameters.

Editing Symbol/Interface Object Properties

You can edit s and graphical properties for a component interface in the tabular IO or symbol view by using the  button, **Alt** +  shortcut or choosing **Object Properties** from the **Edit** menu.

The Symbol Object Properties dialog box is displayed with tab options for setting s, comment **Text** properties and **Symbol** body properties.

Note



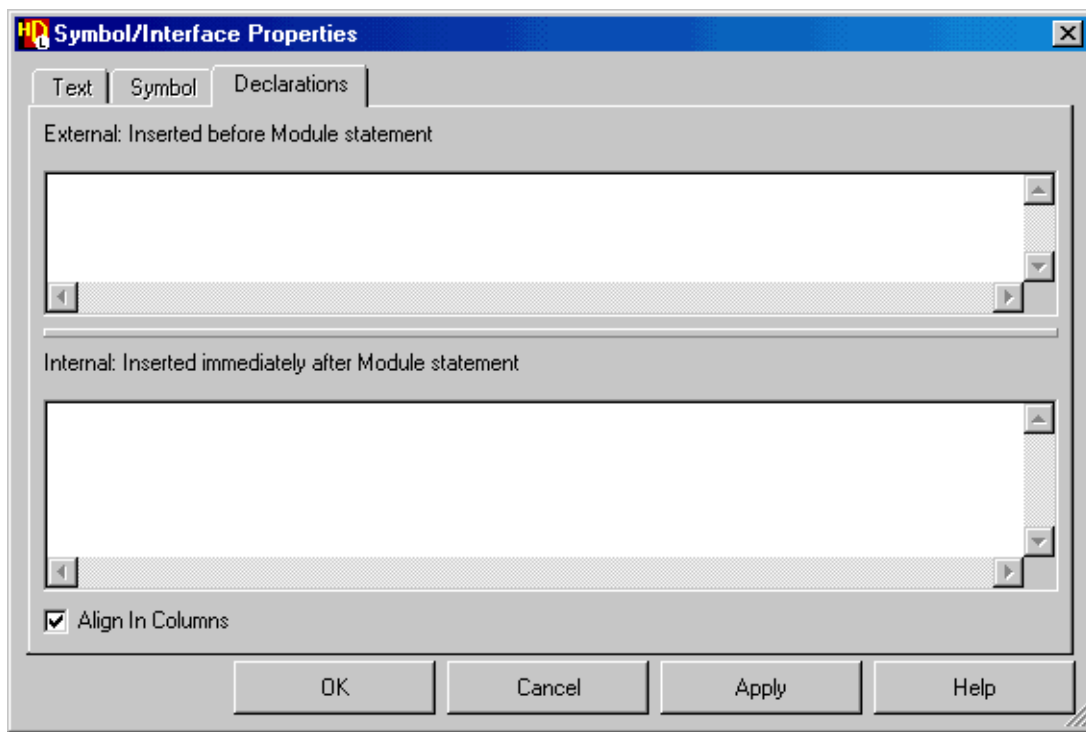
The **Text** tab is available in the symbol view when comment text is present on the diagram. The **Symbol** tab is only available in the symbol view.

Refer to “[Editing Text Properties](#)” on page 58 for information about editing comment text properties in the symbol view.

Editing Symbol User Declarations

You can use the **Declarations** tab of the Symbol/Interface Properties dialog box to add or edit User Declarations in the symbol or tabular IO view.

The dialog box can also be displayed directly by double-clicking on an existing declaration in the symbol view or by choosing **Edit Declarations** from the **Table** menu or from the popup menu when a complete port row is selected in the tabular IO view.



You can also edit the statements directly on the symbol view by clicking to select the text and clicking again to edit the text.

You can enter declarations in the dialog box using free format. The statements are added to the Declarations list on the diagram when you use the **Apply** button.

The syntax is automatically checked for the hardware description language of the active view. However, syntax checking can be disabled by unsetting a preference. If you change the language for the current view, syntax errors are reported when you edit a statement.

When you are using VHDL, there is a single entry box for declarations which are included as entity declarations in the generated VHDL entity.

When you are using Verilog, the dialog box has separate sections for external and internal declarations. The external declarations are inserted in the generated HDL before the Verilog module statement. The internal declarations (typically used to declare interface parameters) are inserted immediately after the module statement.

You can set or unset the **Align in Columns** option to control how the declarations are formatted on the diagram. Note that a monospace font must be used for the this option to be effective.

Editing Symbol Body Properties

You can edit the properties for a symbol body in the symbol view by using the **Symbol** tab of the Symbol/Interface Properties dialog box.

The dialog box displays the library and component names as read-only text fields and provides options to control how the symbol is displayed in the symbol view or when it is instantiated on a block diagram.

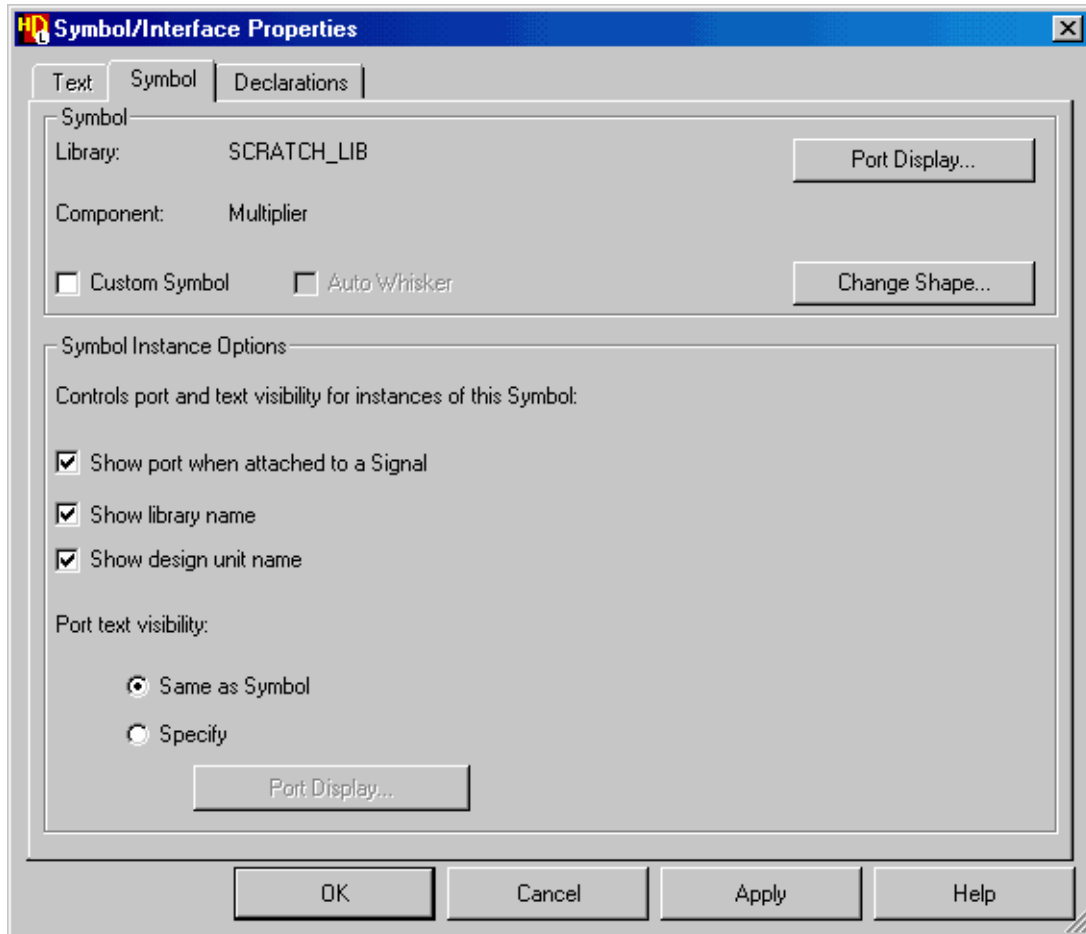
You can use the **Port Display** button to control how port properties are displayed in the symbol view as described in [“Changing the Display of Port Properties”](#) on page 206.

You can use the **Change Shape** button to choose from a list of alternative standard shapes as described in [“Choosing a Standard Shape”](#) on page 229.

You can enable **Custom Symbol** editing and choose whether to automatically add whiskers to signals as described in [“Customizing a Symbol”](#) on page 317.

If the **Show ports when attached to a signal** option is set, ports are displayed when a component instance is connected on a block diagram. (Unconnected ports are always visible but are automatically hidden when a signal is connected if this option is not set.)

The **Show library name** and **Show design unit name** options control whether the library name and design unit names are displayed on each instance.



If the **Same as symbol** option is set, the port text visibility for an instance displays the same properties that are visible in the symbol view.

Alternatively, you can set the **Specify** option and use the **Port Display** button to set the default visibility of port properties text displayed for an instance of the symbol on a block diagram.

i **Tip:** You can change the display of port properties for a selected component instance on a block diagram by using the **Port Display** button in the **Components** tab of the block diagram Object Properties dialog box (or by choosing Port Visibility from the popup menu).

Setting Interface Preferences

You can set preferences for the interface views by choosing **Interface** from the **Master Preferences** cascade of the **Options** menu in the *design manager* to display the Interface Master Preferences dialog box

The dialog box has separate pages for setting **General**, **Default Settings**, **Interface Table** and **Symbol** preferences.

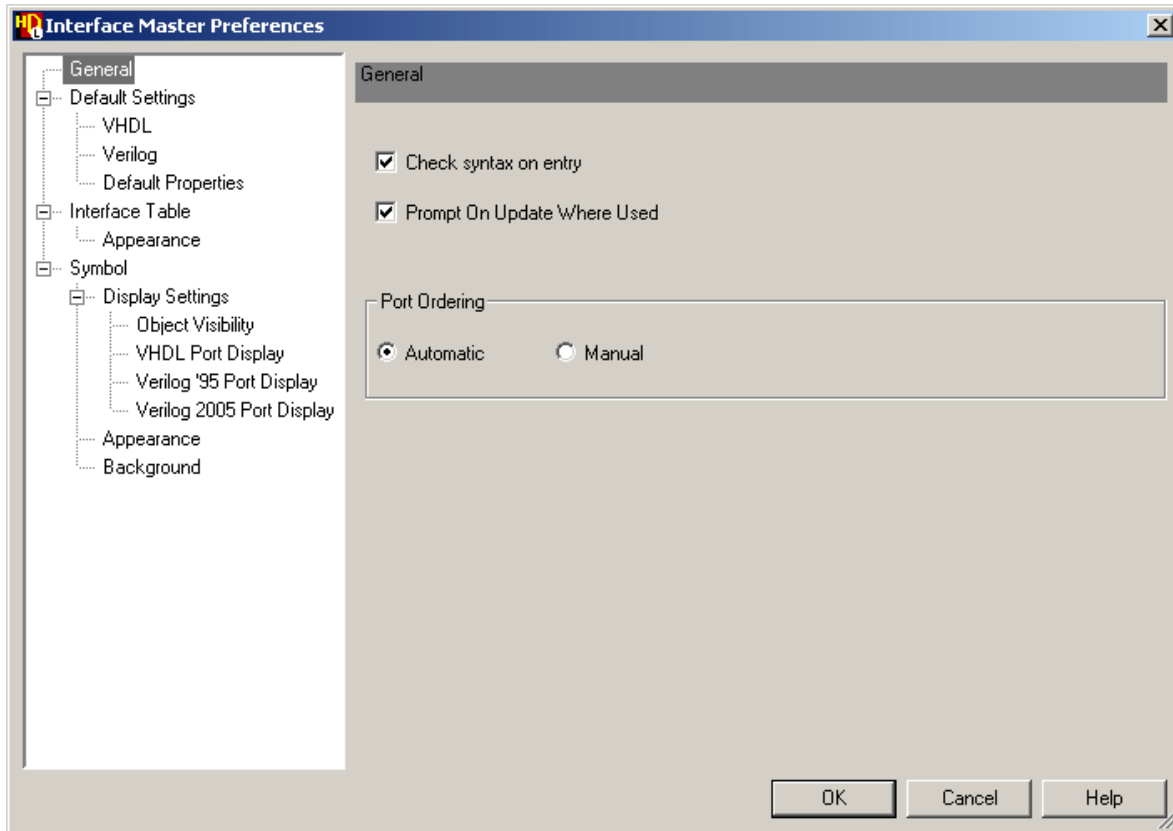
The master preferences take effect on the next interface you create. However, you can apply the current master preferences for the symbol appearance and background to the active view by choosing **Apply to New Objects** or **Apply to New and Existing Objects** from the **Master Preferences** cascade of the **Options** menu in the diagram.

You can set the symbol appearance and Interface table preferences for the active symbol by choosing **Diagram Preferences** from the **Options** menu in the symbol/Interface view.

When you edit these preferences for the active diagram, the dialog box allows you to choose whether the preferences are applied to new objects or to both new and existing objects in the diagram.

You can save the preferences for the active diagram as master preferences by choosing **Update from Diagram** in the **Master Preferences** cascade of the **Options** menu.

The **General** page allows you to set miscellaneous preferences for the interface and symbol view:



You can set an option to check the HDL syntax of declarations and embedded HDL text objects on entry.

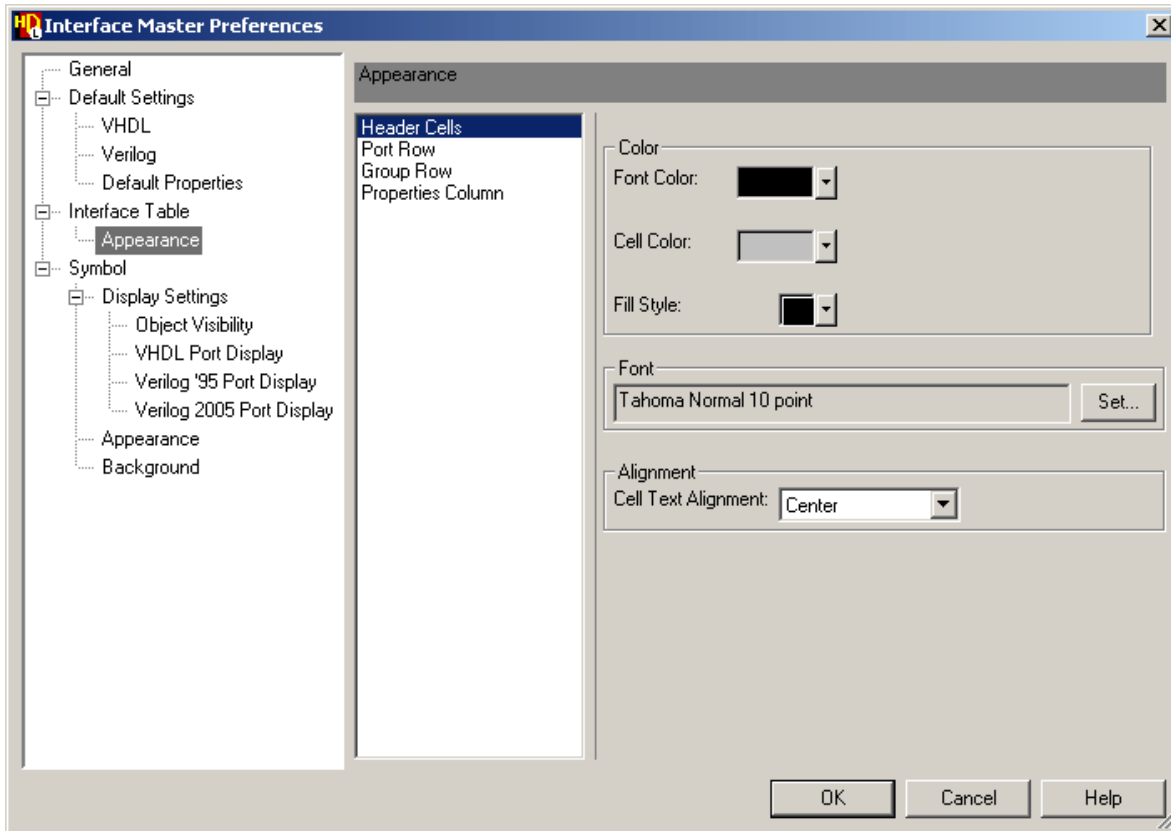
Note



Checks are also performed for unsynthesizable constructs when **Common synthesis checks** are set in the **Checks** tab of the Main Settings dialog box.

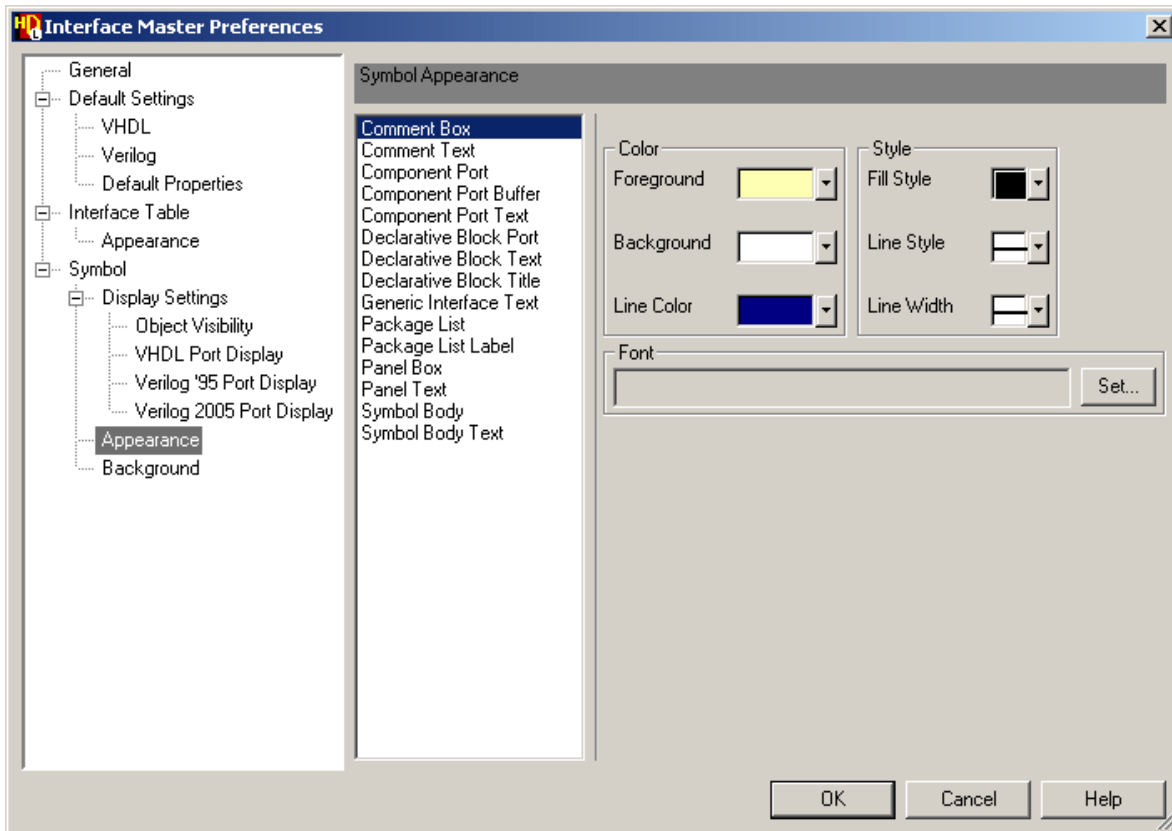
You can choose whether to automatically open the Where Used wizard when edits to the logical interface are saved.

You can also specify whether port declaration ordering is automatic (by mode and alphanumeric name) or if manual ordering is allowed and choose whether the interface view for a symbol is opened as a *tabular IO* interface or as a graphical *symbol*.



The **Interface Appearance** page allows you to set default visual attributes for the font and cell colors in the tabular IO view. You can also set the fill style, default text font and cell text alignment.

The appearance options are described in the description of the Cell Edit Appearance dialog box which can be used to change the appearance on individual cells. Refer to [“Setting Visual Attributes in the Tabular IO View”](#) on page 314 for more information.



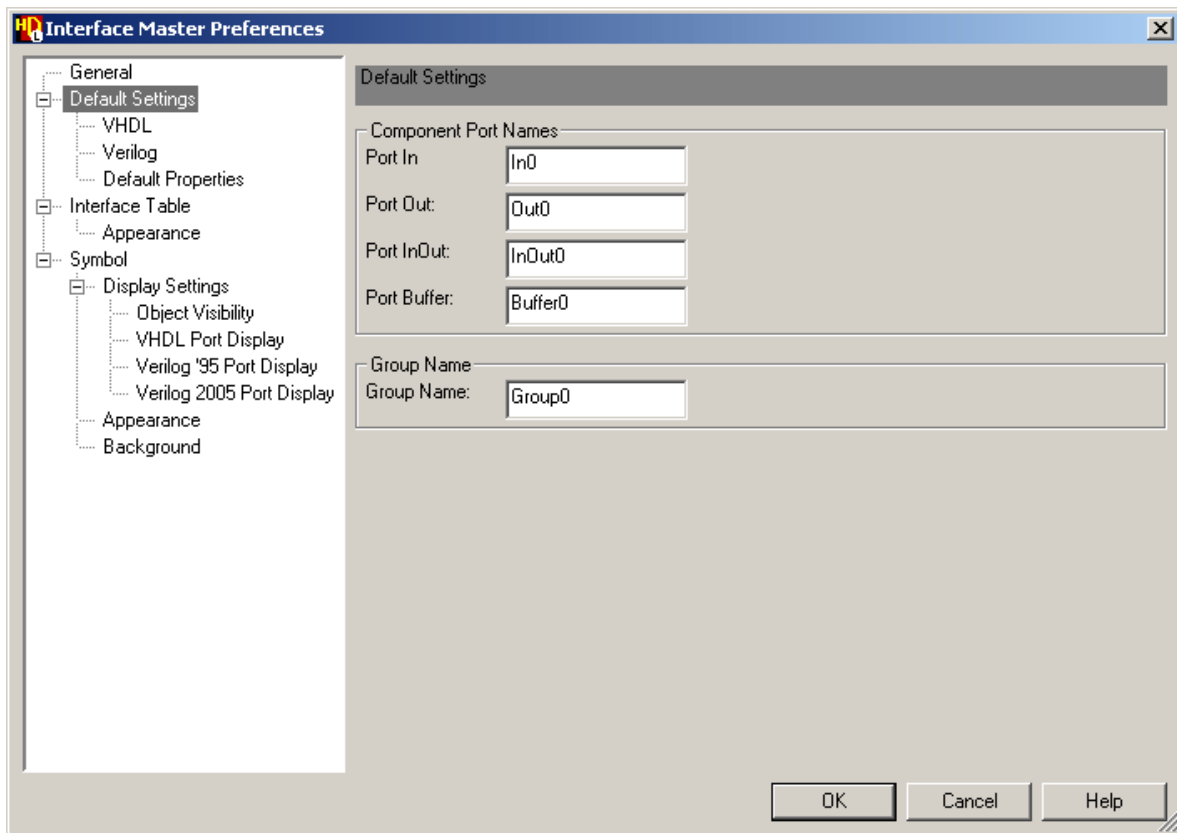
The **Symbol Appearance** page allows you to set default visual attributes for individual objects in the symbol view

The attributes that can be set include the foreground and background colors, line color and style, fill style and line width, and the text font. However, some attributes are not always available. For example, the line style, width and color attributes are not available for a text object.

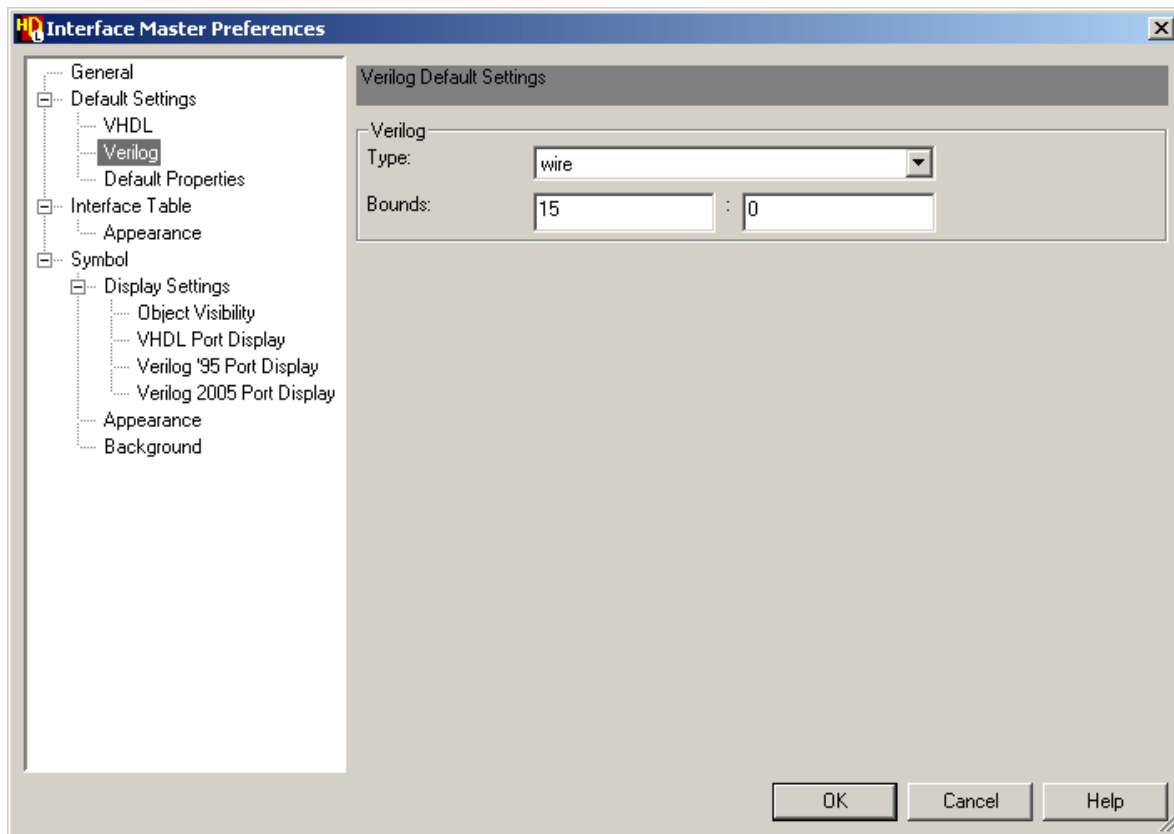
The appearance options are described in the description of the Edit Appearance dialog box which can be used to change the appearance of individual objects in a diagram. Refer to [“Setting Visual Attributes”](#) on page 83 for more information.

When accessed from the editor, you can choose whether to apply the symbol appearance changes to new objects only or to all new and existing objects on the diagram.

The **Default Settings** page allows you to change the port and group namedefaults.



Separate **VHDL** and **Verilog** sub-pages can be used to set the default constraints, type and bounds for VHDL and Verilog ports. For example, the following picture shows the Verilog sub-page:

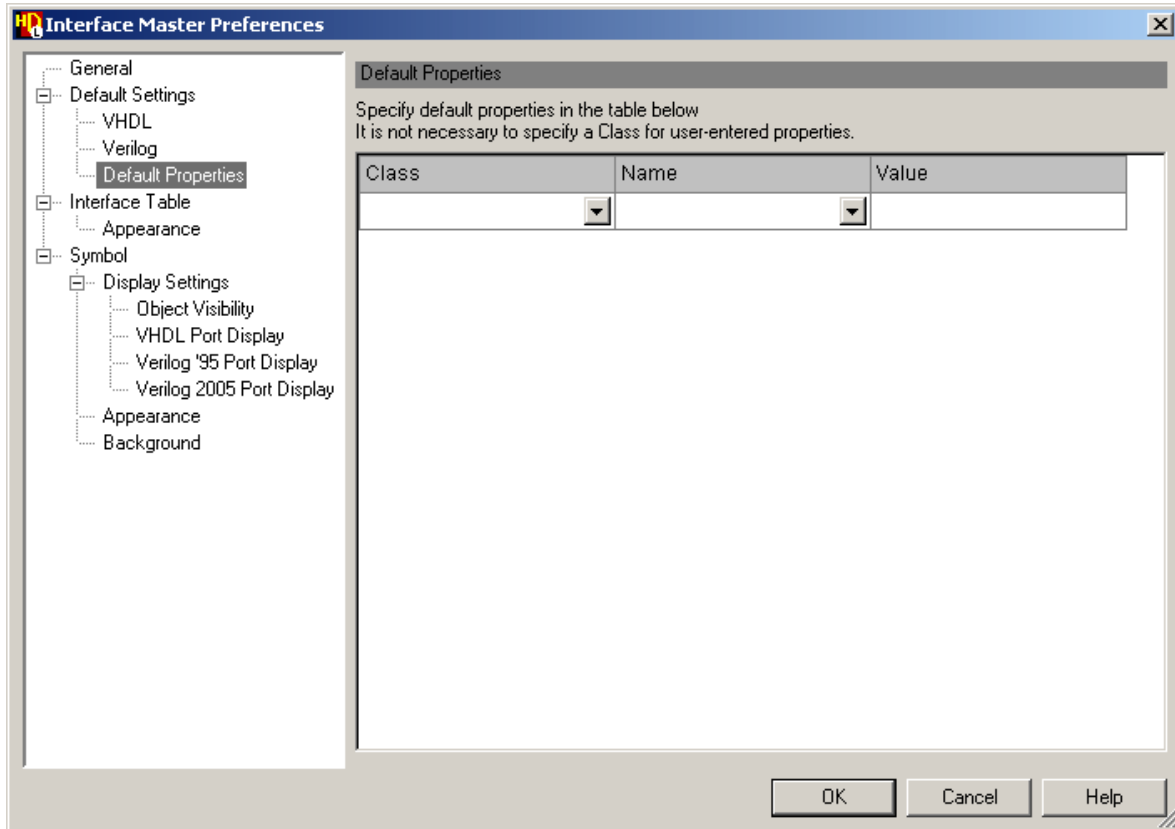


The bounds can be entered as a range (for example, *15 DOWNT0 0* or *0 TO 75* in VHDL or *0:7* in Verilog) or you can use the first bounds entry box as an index for a single element in an array leaving the second entry empty.

For VHDL, you can also use the first bounds entry box to enter a user specified constraint such as an enumerated or integer type name or you can enter an array name or type of the form:

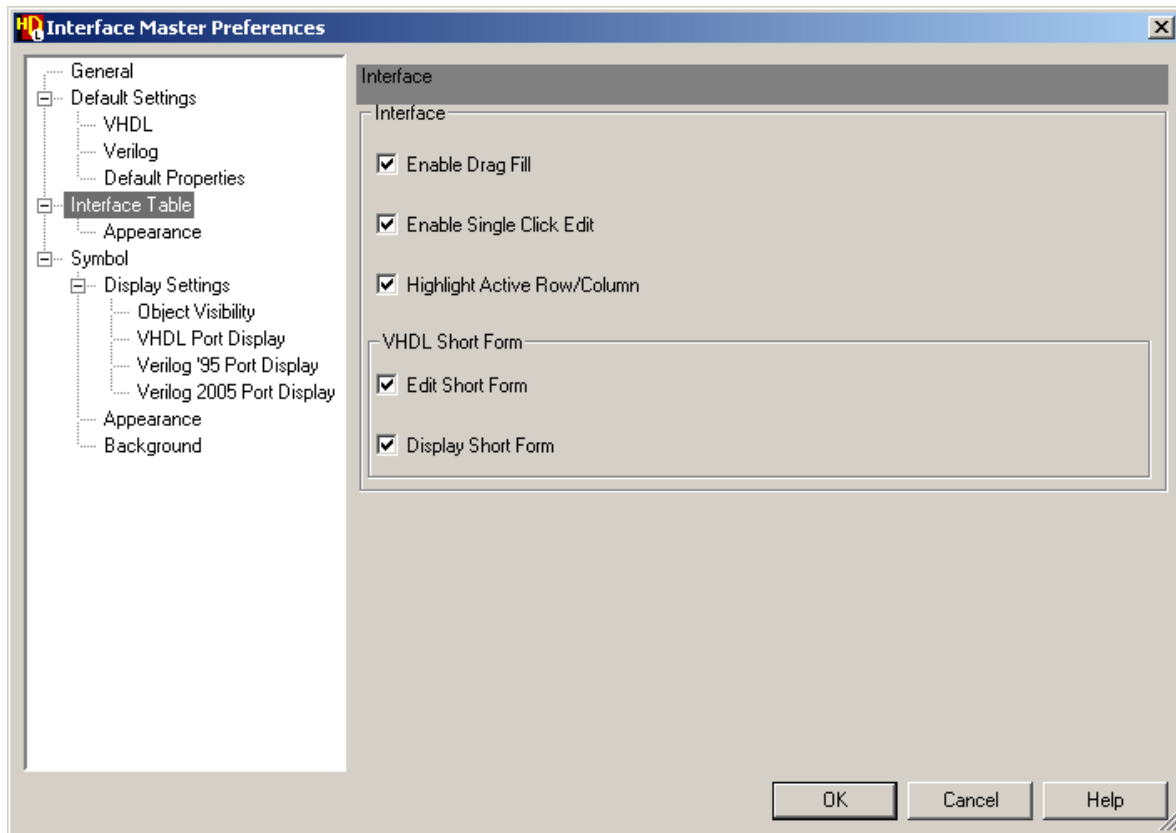
```
<array>'RANGE or <array>'REVERSE_RANGE
```

You can use the **Default Properties** sub-page to define default properties for tabular IO interface views.



Refer to the [HDL Designer Series User Manual](#) for information about “Using View Property variables”.

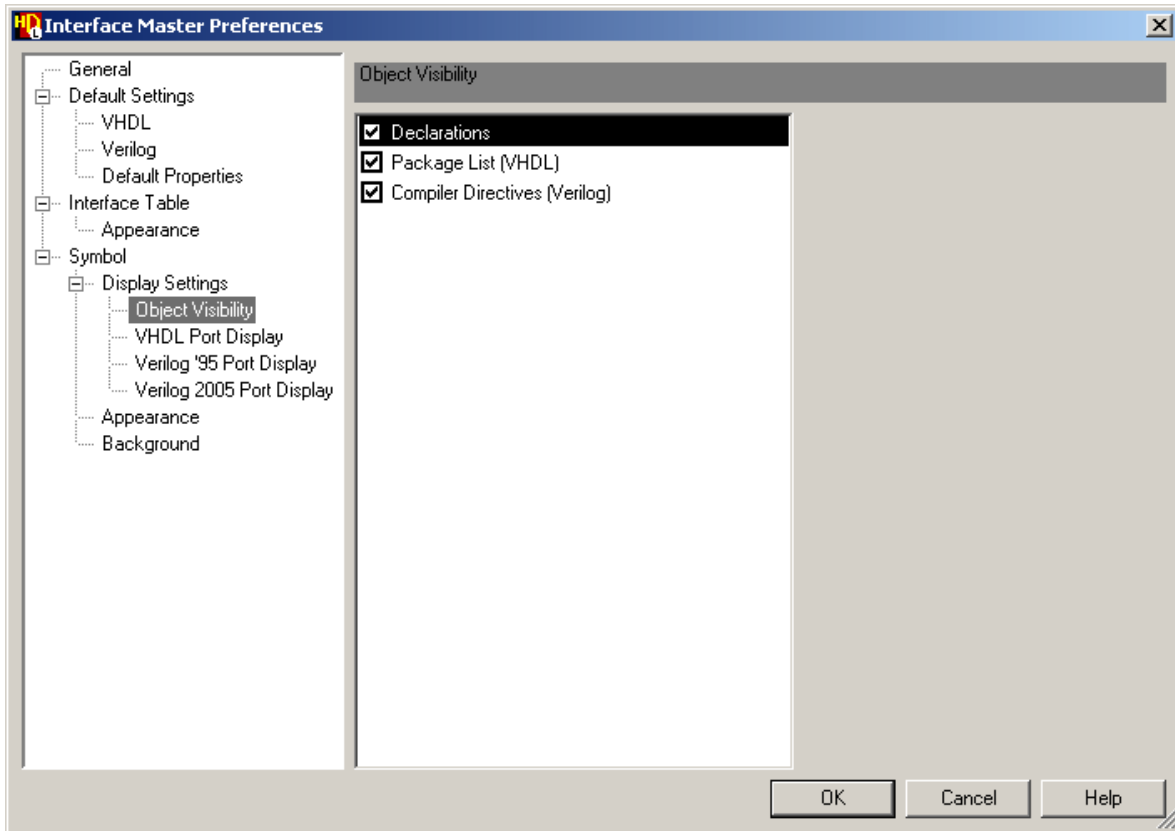
The **Interface** sub-page provides additional preferences which apply only to the interface view:



You can enable drag fill and enable single click editing in the interface view. You can enable an option to highlight the active row and column when a table cell is selected.

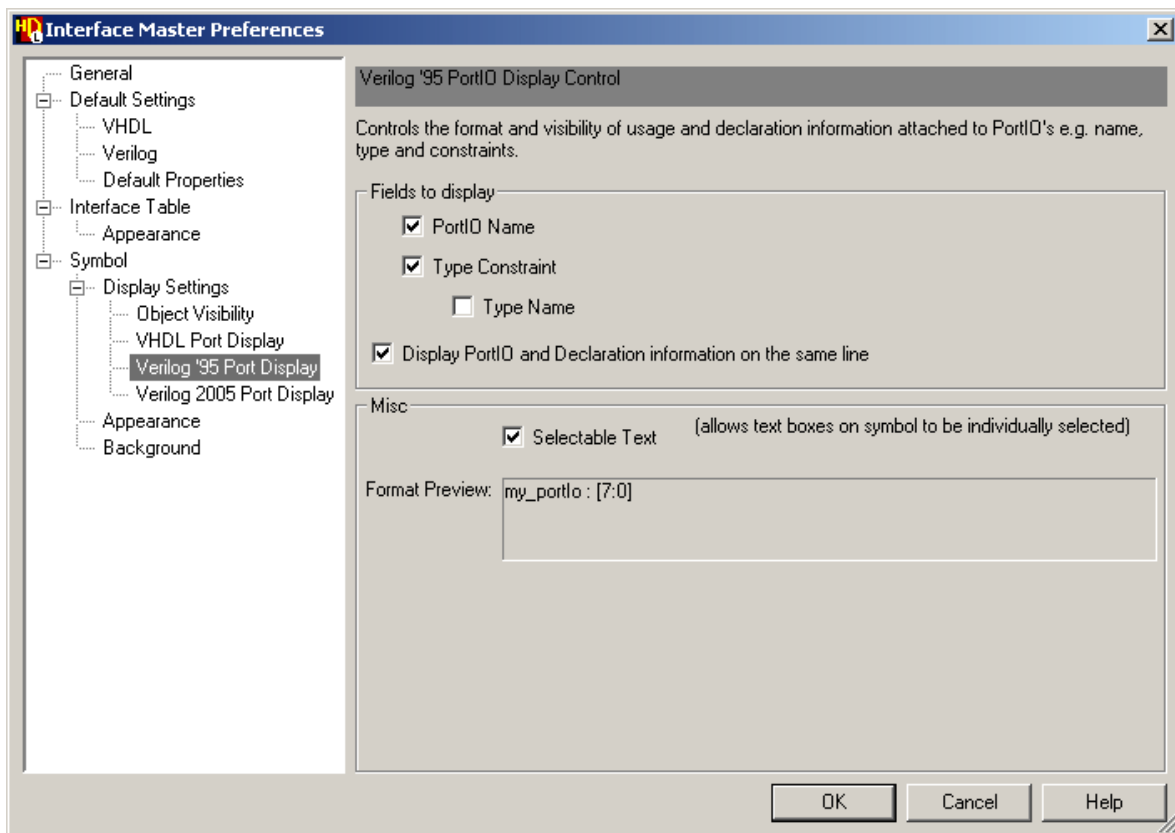
You can also choose whether to edit or display VHDL range constraints using short form notation.

The **Object Visibility** sub-page allows you to set the default object visibility for multi-line text objects on the diagram. Refer to [“Changing Text Visibility”](#) on page 68.

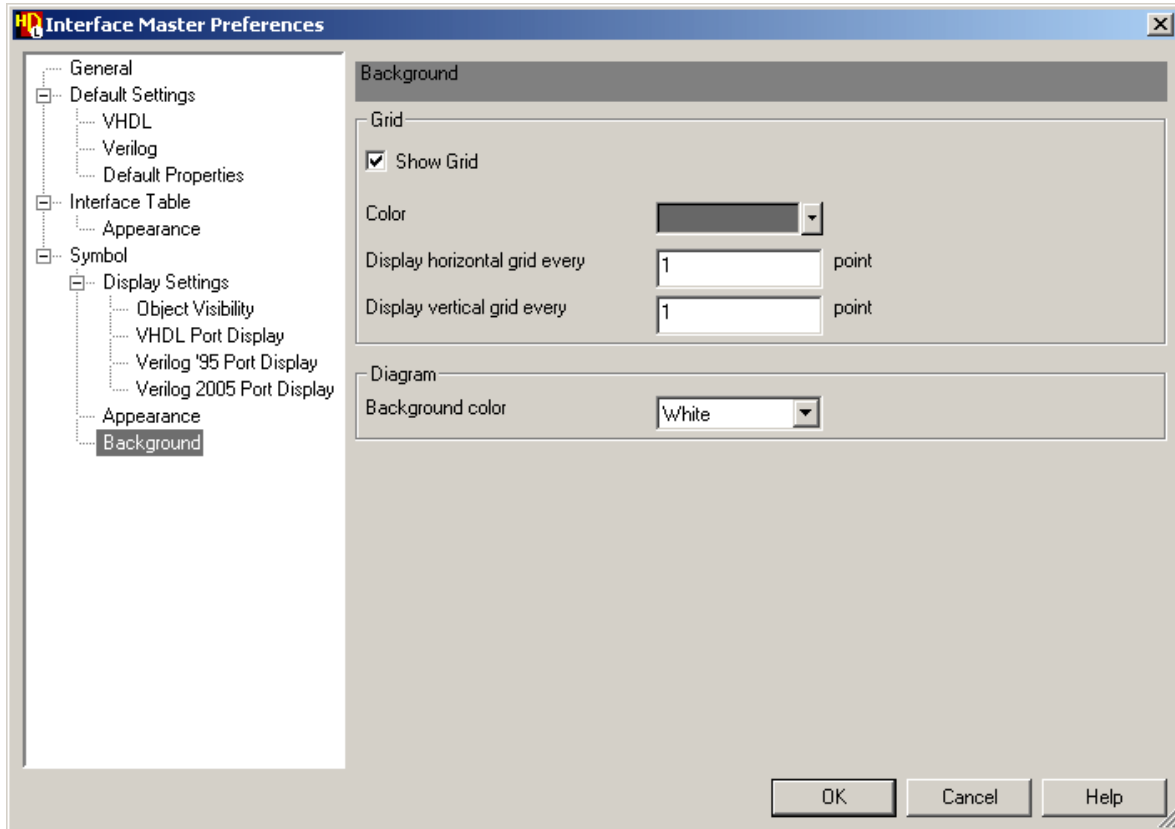


The **VHDL Port Display** and **Verilog Port Display** sub-pages buttons allow you to set default port display properties. For example, the following picture shows the Verilog Port Display page:

The port display properties can be set separately for VHDL and Verilog when you edit the master preferences or for the language used by the active symbol when accessed from the editor. Refer to [“Changing the Display of Port Properties”](#) on page 206 for more information.



The **Background** page allows you to control the diagram background color and grid attributes used in the symbol view.



These preferences are described in [“Setting Background Preferences”](#) on page 50.

Refer to the “Default Preferences” appendix in the *HDL Designer Series User Manual* for lists of the default preferences which are set when you invoke a HDL Designer Series tool for the first time.

Chapter 8

Flow Chart Editor


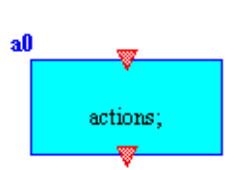
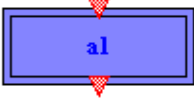
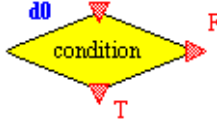

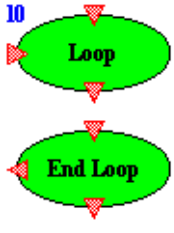
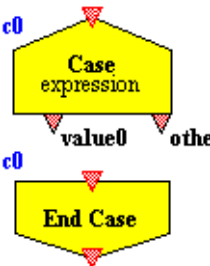
This chapter describes how a design process can be described by a *flow chart* in terms of graphical *action boxes*, *decision box*, *wait boxes*, *case boxes* and *loops* connected by *flows*. This technique can be used to describe a design unit view of any leaf-level *block* or *component* in a *block diagram* or *IBD view*.

Flow Chart Notation	340
Flow Chart Toolbar	342
Adding Objects on a Flow Chart	343
Adding a Start Point	345
Adding an Action Box	346
Adding a Decision Box	347
Adding a Wait Box	348
Adding a Loop	348
Adding a Case Box	349
Adding a Flow	351
Adding an End Point	352
Adding Other Objects on a Flow Chart	352
Hierarchical Flow Charts	352
Concurrent Flow Charts	354
Adding a Concurrent Flow Chart	355
Opening a Concurrent Flow Chart	355
Renaming a Concurrent Flow Chart	355
Deleting a Concurrent Flow Chart	356
Editing Flow Chart Object Properties	356
Editing Action Box Object Properties	356
Editing Decision Box Object Properties	358
Editing Wait Box Object Properties	360
Editing Loop Object Properties	362
Editing Case Object Properties	363
Setting Flow Chart Properties	364
Setting Flow Chart Generation Properties	365
Editing Architecture or Module Declarations	369
Editing Concurrent Statements	370
Editing Process or Local Declarations	372
Setting Flow Chart Preferences	373

Flow Chart Notation

The notation used for flow chart objects is show below.

Table 8-1. Flow Chart Notation

	<p>There must be one and only one <i>start point</i> which is always named <i>Start</i>.</p>
	<p>An <i>action box</i> contains HDL statements which are executed when the box is entered from a <i>flow</i>. There must be one input flow and one output flow.</p>
	<p>A <i>hierarchical action box</i> represents a <i>child</i> flow chart describing action logic.</p>
	<p>A <i>decision box</i> represents if-then-else statements and has two outputs: A True <i>flow</i> which is followed when its condition is satisfied or a False <i>flow</i> otherwise.</p>
	<p>A <i>wait box</i> defines a HDL wait or delay statement.</p>
	<p>A <i>loop</i> comprises start loop and end loop objects. The loop will continue for ever if no control properties are associated with the start loop object. Any number of other objects can be included in the flow between the start and end loop.</p>
	<p>A <i>decision box</i> represents if-then-else statements and has two outputs: A True <i>flow</i> which is followed when its condition is satisfied or a False <i>flow</i> otherwise.</p>

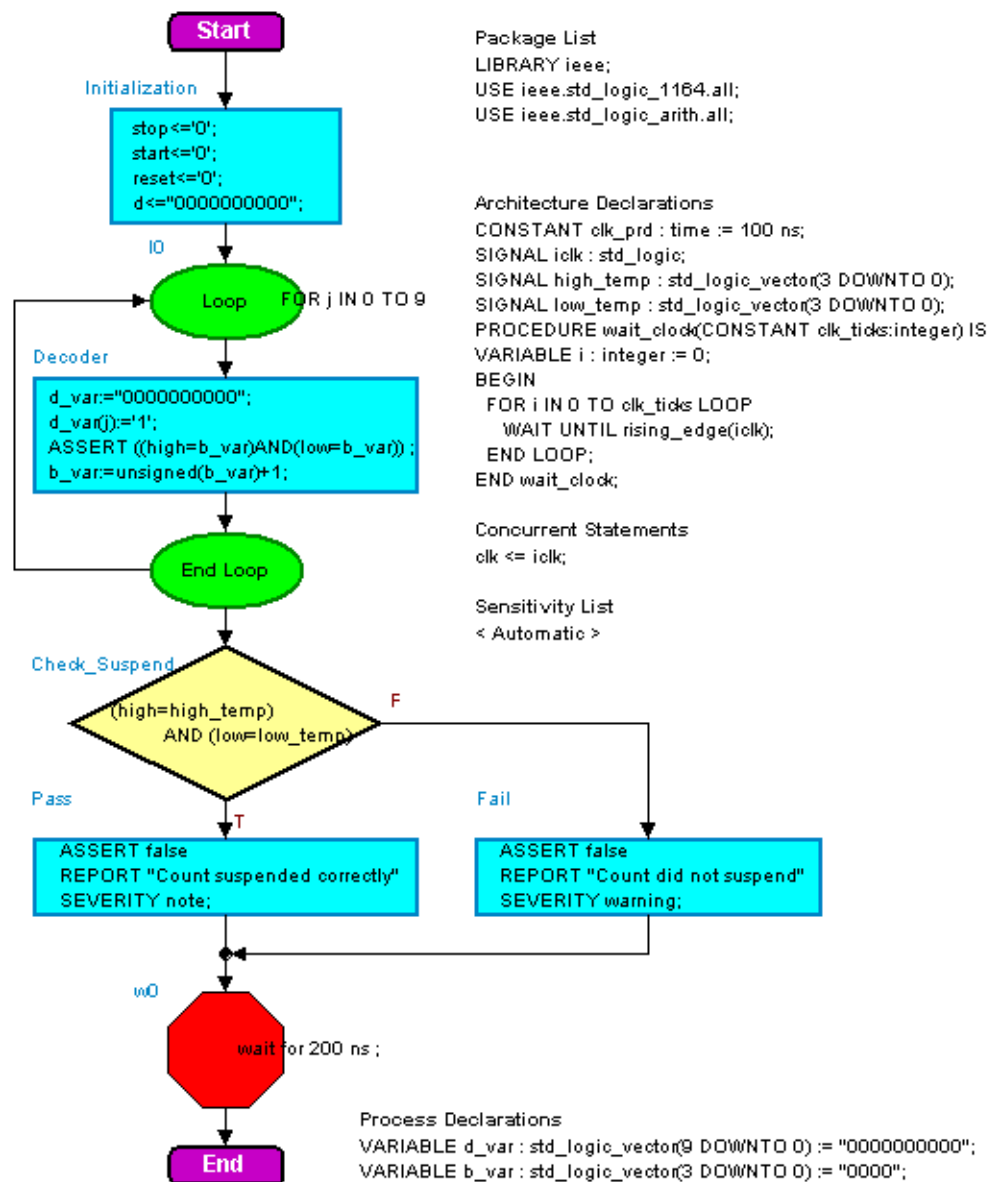


There must be at least one *end point*.

The ▼ on flow chart objects indicates *ports* where *flows* can be connected.

A flow chart may contain any combination of objects connected by flows but must contain one start point and one (or more) end points.





















A flow chart is typically used for the input stimulus and output checking blocks in a test bench. For example:



Flow Chart Toolbar



The following commands are available from the Flow Chart Tools toolbar:

Table 8-2. Flow Chart Toolbar

Icon	Description
	Select text or objects
	Select text only
	Select objects only
	Add or modify comment text
	Pan the window
	Add a new concurrent flow chart
	Open a named concurrent flow chart
	Delete a named concurrent flow chart
	Add an action box
	Add a hierarchical action box
	Add a decision box
	Add a wait box
	Add a start point
	Add an end point
	Add a start loop
	Add an end loop
	Add a case box
	Add a case port
	Add a flow
	Add a panel

Note



The  and  buttons have pulldown menus which allows you to choose from a list of concurrent flow chart names.

Refer to the [HDL Designer Series User Manual](#) for general information about toolbars and the HDL Designer Series user interface.

Refer to [Diagram Editor Windows](#) in Chapter 2 or information about selecting objects, resizing objects, adding comment text or graphics, panning the window, adding a panel and additional toolbars which are common to the other *diagram editors*.

Adding Objects on a Flow Chart

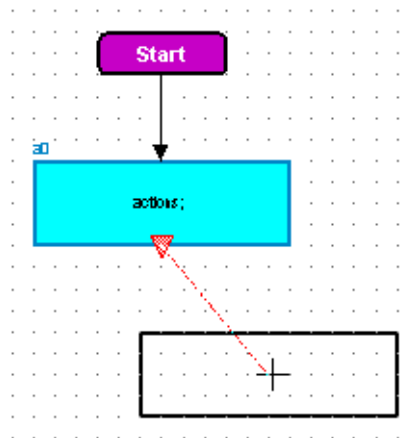
You can add objects on a *flow chart* using the **Add** menu or one of the buttons in the Flow Chart Tools toolbar. Some objects can also be added using a shortcut or mnemonic key.

You can find a list of supported Graphical Editor Shortcut Keys in the **Quick Reference Index** which is displayed using the -help switch.

The cursor changes to a cross-hair which allows you to add the object by clicking at the required location on the diagram.


When you add any object (except a *start point*) on a flow chart, a *flow* is automatically connected to the nearest unconnected *port* on an existing object.

The ghosting shows which port the object will connect to. If there are several available ports, the ghost flow snaps between them as you move the cursor.



This automatic connection mode can be set or unset by choosing **AutoConnect** from the **Diagram** menu. The current setting is saved as a preference.

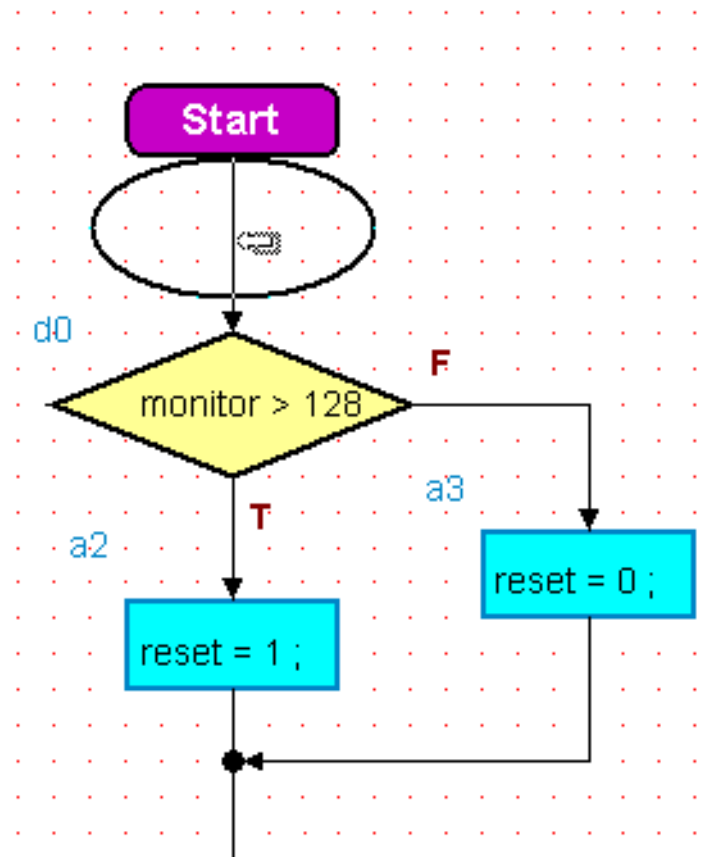
After adding an object, the command normally repeats until you use the **Esc** key (or **Right** mouse button) to terminate the command. However, you can set a preference for the command to remain active or activate only once and you can toggle this mode for the current command by using the **Ctrl** key.

If you move the cursor over an existing flow while you are adding an object, the cursor changes to  and the object is inserted into the flow between the existing objects.


If the new object is too close to the object above it, it will automatically snap to a position in free space below the object. Any existing objects below the new object are automatically moved down to make space.

This automatic insertion mode can be set or unset by choosing **AutoInsert** from the **Diagram** menu. The current setting is saved as a preference.

The following example shows a loop being inserted into the flow between a start point and a decision box:




Note

 Automatic insert mode works only for vertical flows.

If an object is resized so that it overlaps the next object in a flow, the next object is automatically moved down to make space.


Adding a Start Point

You can add a *start point* to a flow chart using the  button, **Shift + F8** or **S** shortcut keys or by choosing **Start Point** from the **Add** menu.

There must be one (but only one) start point on a flow chart.


The default name for a start point can be changed by setting a preference.

Adding an Action Box

You can add an *action box* to a flow chart using the  button or **T** shortcut key or by choosing **Action Box** from the **Add** menu.

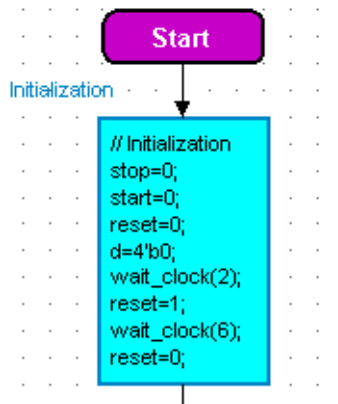
You can add a *hierarchical action box* to a flow chart using the  button, **Shift + F2** or **H** shortcut keys or by choosing **Hierarchical Action Box** from the **Add** menu.

You can change the name of an action box (and the enclosed actions) by clicking to select the text and clicking again to edit the text in-line.

Alternatively, you can double-click on the action box, use the  button or choose **Object Properties** from the **Edit** menu, to display the **Action Boxes** tab of the Object Properties dialog box as described in “[Editing Action Box Object Properties](#)” on page 356.

If you do not change the name of an action box, each new action box is given a unique name by adding an integer to the default name (for example: *a0*, *a1*, *a2*...). The default actions can be changed by setting preferences. Action boxes and hierarchical action boxes have the same default base name (*a*).

The default *actions* defined in your preferences are placed in the action box. The actions may overlap the action box outline but can be independently moved away from (or into) the action box. If you want to contain all your actions inside the box, it may be necessary to resize the object.




The actions syntax is automatically checked for the hardware description language of the active diagram. However, flow chart syntax checking can be disabled by unsetting a preference.


Note



Verilog system functions such as *\$display* or *\$stop* can be used in an action box.

Adding a Decision Box

You can add a [decision box](#) to a flow chart using the  button, **F3** or **D** shortcut keys or by choosing **Decision Box** from the **Add** menu.

You can change the name or condition by clicking to select the text and clicking again to edit the text. Alternatively, you can double-click on the decision box, use the  button or choose **Object Properties** from the **Edit** menu, to display the **Decision Boxes** tab in the Object Properties dialog box as described in [“Editing Decision Box Object Properties”](#) on page 358.

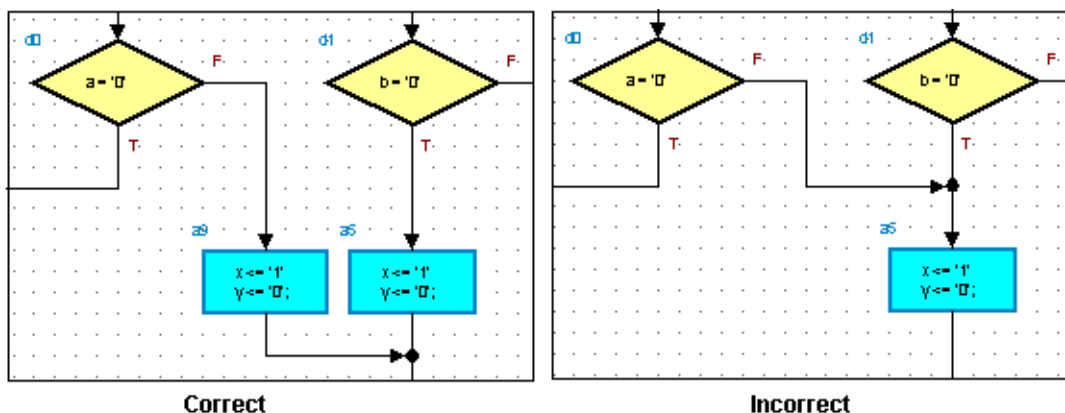
If you do not change the name of an decision box, each new decision box is given a unique name by adding an integer to the default name (for example: *d0*, *d1*, *d2*...). The default base name for new decision boxes, the default condition text or default labels for the True and False flows can be changed by setting preferences.

You can exchange the position of the output flows by choosing **Swap True and False** from the **Diagram** or popup menu when a decision box is selected.


The condition text is placed inside the decision box and after editing may overflow the box but can be moved independently away from (or into) the box. If you want to contain the whole condition inside the box, it may be necessary to resize the object.

When you enter a condition, the syntax is automatically checked for the hardware description language of the active diagram. However, flow chart syntax checking can be disabled by unsetting a preference.


A decision box corresponds to an If statement in HDL. You should not connect the output flows from two or more decision boxes directly together as this would generate invalid HDL. If necessary, copy the same action box and connect separately to both output flows. You can then recombine the flows using a flow join. End if statements will then be correctly generated for each decision arm.



Adding a Wait Box

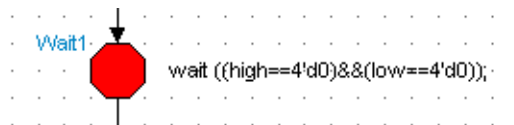
You can add a *wait box* to a flow chart using the  button, **F4** or **W** shortcut keys or by choosing **Wait Box** from the **Add** menu.

You can change the name of a wait box by clicking on the name to select the text and clicking again to edit the text. Similarly, you can change the wait statement by clicking on the wait statement text.

Alternatively, you can double-click on the wait box, use the  button or choose **Object Properties** from the **Edit** menu, to display the **Wait Boxes** tab in the Object Properties dialog box as described in “[Editing Wait Box Object Properties](#)” on page 360.


When you enter wait statements, the syntax is automatically checked for the hardware description language of the active diagram. However, flow chart syntax checking can be disabled by unsetting a preference.

If you do not change the name of an wait box, each new wait box is given a unique name by adding an integer to the default name (for example: *w0*, *w1*, *w2*...). The default base name for new wait boxes and the default wait statement text can be changed by setting preferences.




The wait statement text is normally offset overlapping the wait box but can be moved independently away from (or into) the box. If you want to contain the whole statement inside the box, it may be necessary to resize the object.

Adding a Loop

You can add a start *loop* to a flow chart using the  button or **L** shortcut key or by choosing **Loop** from the **Add** menu to place a start loop object.

Any combination of flow chart objects (including other loops) can be added below the start loop.


To complete the loop, you must add an end loop object using the  button or **O** shortcut keys or by choosing **End Loop** from the **Add** menu.

Note



If AutoConnect is enabled in the Diagram menu, the loop back flow is automatically connected to the nearest unconnected start loop.

You can change the name of a loop by clicking on the name to select the text and clicking again to edit the text. However, you cannot change the loop control text directly.

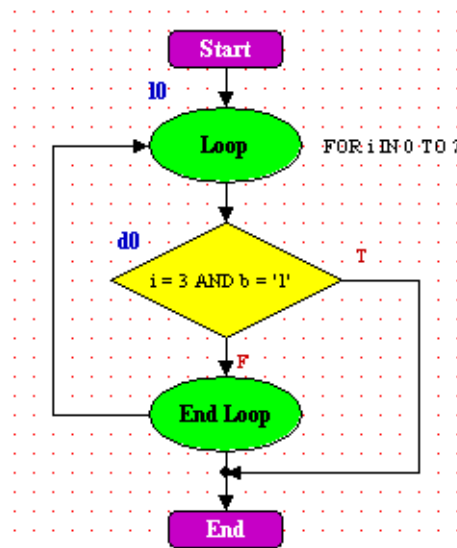
To edit the control text (or the loop name), double-click on the wait box, use the  button or choose **Object Properties** from the **Edit** menu, to display the **Loops** tab in the Object Properties dialog box as described in “[Editing Loop Object Properties](#)” on page 362.

If you do not change the name of a loop, each new loop is given a unique name by adding an integer to the default name (for example: (10, 11, 12...)). The default base name for new loops and the default labels for the start and end loop objects can be changed by setting preferences.

The loop control text is normally placed to the right of the start loop object but can be moved independently away from (or into) the object. If you want to contain the whole of the control text inside the loop object, it may be necessary to resize the object.

Breaking Out of a Loop

You can break out of a [loop](#) by using a [decision box](#) to set a "breakout" condition which is connected below the end loop using a [flow join](#) as shown below:




Note



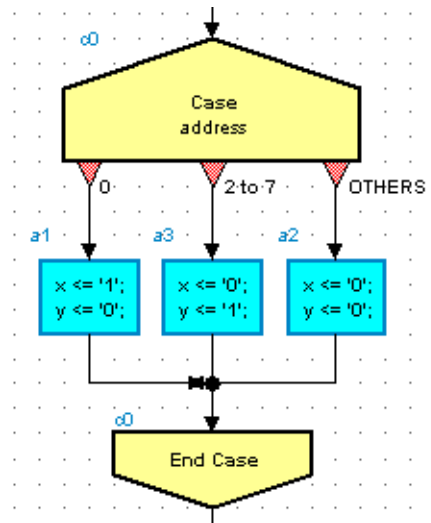
If there are multiple break-out conditions, these must be combined using a single break-out decision box. For example, in the example above, the loop is exited when the loop is on its third iteration if the signal b has the value '1'. Otherwise, the loop is exited normally after completing eight iterations.


Adding a Case Box

You can add a [case box](#) to a flow chart using the  button, **F6** or **C** shortcut keys or by choosing **Case** from the **Add** menu.

The start case box is added at the cursor location and an associated end case object with the same name is automatically added vertically below.


Any combination of other flow chart objects (including other case boxes) can be added between the start and end case objects.



A new start case box has a single output port (with a name which can be set as a preference) but you can add any number of unconnected case ports using the  button, **Shift + F6** or **P** shortcut keys or by choosing **Case Port** from the **Add** menu or **Add Port** from the popup menu when the case box is selected or simply by adding flows with their origin over the start case object.

You can delete a case port by using the **Del** key or by choosing **Delete** from the **Edit** or popup menu while the port is selected.

You can change the case name, case expression or the value for an existing case port by clicking on the name to select the text and clicking again to edit the text.


Alternatively, double-click on the case box or any case port, use the  button or choose **Object Properties** from the **Edit** menu to display the **Cases** tab in the Object Properties dialog box as described in “[Editing Case Object Properties](#)” on page 363.

When you enter a case expression or case port name, the syntax is automatically checked for the hardware description language of the active diagram. However, flow chart syntax checking can be disabled by unsetting a preference.


If you do not change the name of a case, each new case object is given a unique name by adding an integer to the default name (for example: *c0*, *c1*, *c2*...). The default case expression and port values for new case boxes can be changed by setting preferences.

The expression is normally placed inside the start case object but can be moved independently away from (or into) the object. If you want to contain the whole of the expression inside the case box, it may be necessary to resize the object.

Adding a Flow

You can add a *flow* to a flow chart using the  button, **F7** or **F** shortcut keys or by choosing **Flow** from the **Add** menu.

The cursor changes to a cross-hair which allows you to add a flow by clicking the **Left** mouse button with the cursor over a source and destination plus any number of route points.

Flows can only be connected between the connect ports shown by  on each flow chart object. However, you can move the loop back points on a loop object to the opposite side by dragging them with the mouse. Similarly, you can move the True and False flows from a decision box to an alternative vertex.

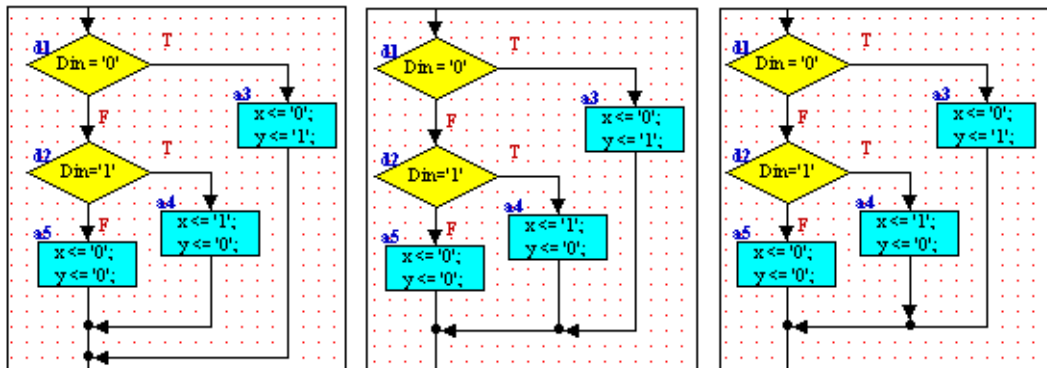
Note



You can dynamically create a port on a case box by adding a flow with its origin over the start case object.

A flow cannot originate on another flow but can be terminated on a flow (creating a flow join).

For example, the following constructs using flow joins are functionally equivalent and generate the same HDL:



Note that if you delete an object (such as an action box) which has one input flow and one output flow, a flow is automatically connected between the objects immediately above and below the deleted object.


All flow chart objects (except case ports) must be connected before you can generate HDL for a flow chart. Any unconnected case port is assigned a NULL statement by HDL generation.

Note



You cannot create a loop by connecting an output directly to the input of a previous object on the flow chart since this could create unreachable nodes. Use a start and end loop as described in [“Adding a Loop”](#) on page 348.

Adding an End Point

You can add an [end point](#) to a flow chart using the  button, **F8** or **C** shortcut keys or by choosing **End Point** from the **Add** menu. There must be at least one end point on a flow chart. The default name for a start point can be changed by setting a preference.

Adding Other Objects on a Flow Chart

You can also add other objects such as a title block, [comment text](#), [comment graphics](#) and [panels](#) on a flow chart.

Hierarchical Flow Charts

A large flow chart can be decomposed into a hierarchy of smaller more manageable hierarchical flow charts which are embedded below hierarchical action boxes in their parent flow chart. The child chart is connected to the hierarchical action box in the parent flow chart by its start point and end point.

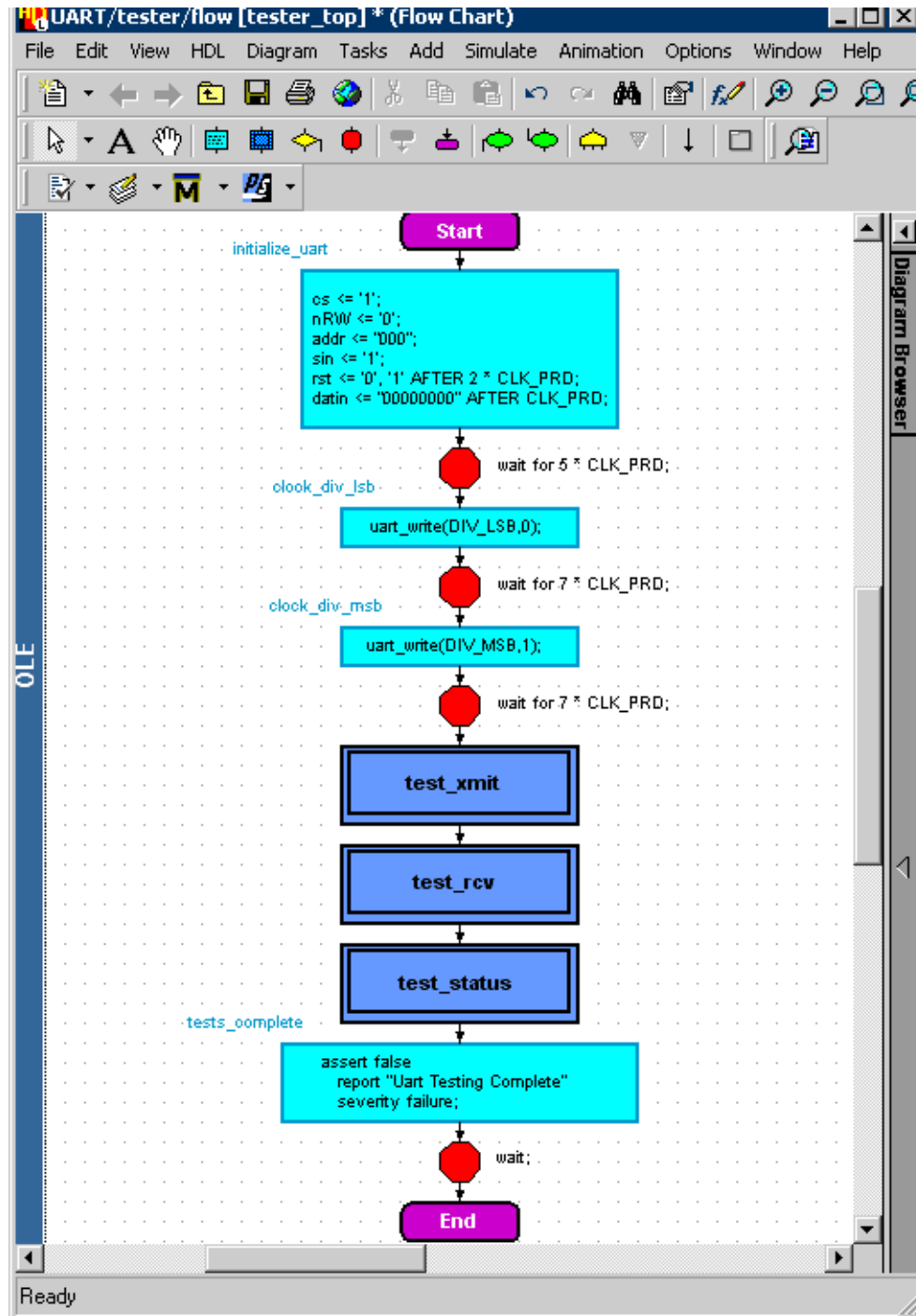
You can open down into a child flow chart by double-clicking on a hierarchical action box or by choosing **Open Down** from the **Open** cascade of the **File** menu (or popup menu) to explicitly open the selected hierarchical action box.

The child flow chart is opened for in the existing window. A new child flow chart comprises a start point, a single action box and an end point connected by flows. You can edit a hierarchical chart in the same way as any other flow chart including more hierarchical action boxes as well as any other flow chart objects.

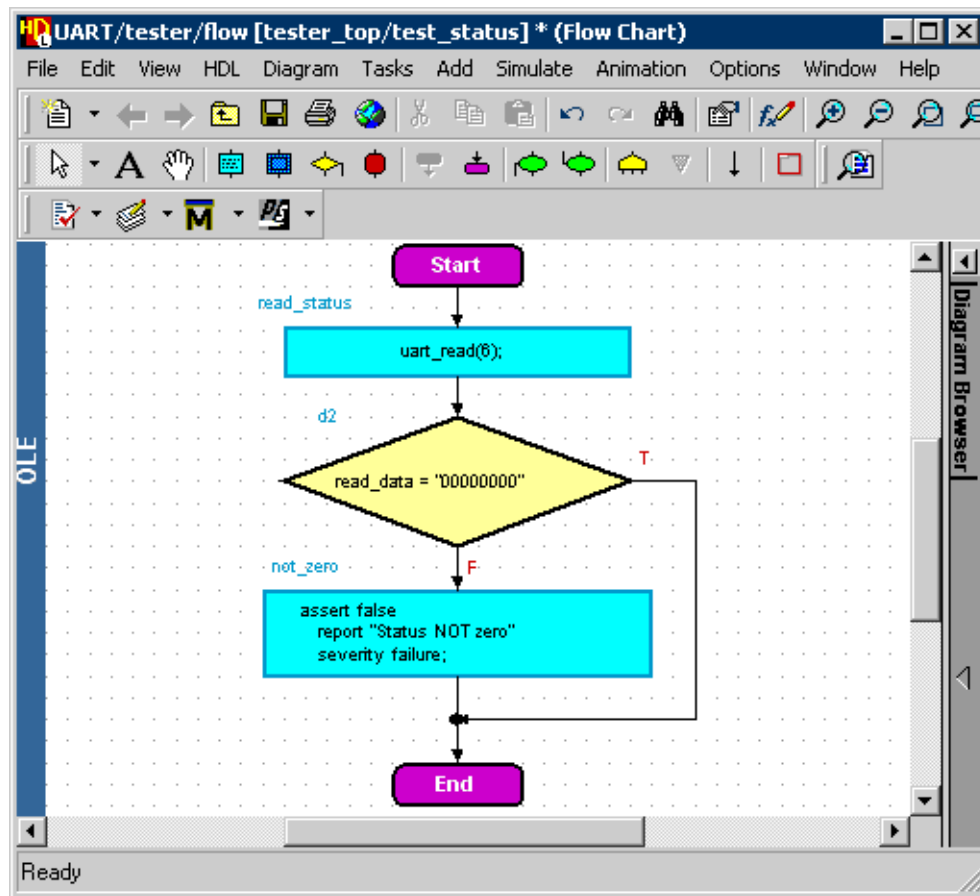
You can choose **Open Up** from the **File** menu or select the name of the parent diagram in the [diagram browser](#) to open the parent of the currently active flow chart.

Child flow charts are saved as part of the parent flow chart and named after the parent hierarchical action box by adding the name of the parent hierarchical action box to the concurrent flow chart name.

For example, the following picture shows a flow chart for a test bench which includes a *test_status*, *test_xmit*, *test_rcv* hierarchical action boxes represented by child flow charts:



The parent flow chart is named: *tester_top* and the child is named *tester_top/test_status*:



Concurrent Flow Charts

Any number of concurrent flow charts can be created from within an existing flow chart with the same interface.

The package list and any VHDL architecture declarations, Verilog module declarations or concurrent statements are shared by the concurrent flow charts but the sensitivity list and process declarations or local declarations can be set separately.

Each concurrent flow chart is given a unique name by adding an integer to the default name (for example: *process0*, *process1*, *process2*...) and identified in the title bar by appending this name and its position in the set of concurrent flow charts to the leaf flow chart name. For example, if you create three concurrent flow charts for the flow chart *DESIGNS\ConcFC\flow* the resulting set of four flow charts would be identified as follows:


DESIGNS\ConcFC\flow [*process0*' 1 of 4]
DESIGNS\ConcFC\flow [*process1*' 2 of 4]
DESIGNS\ConcFC\flow [*process2*' 3 of 4]
DESIGNS\ConcFC\flow [*process3*' 4 of 4]

A set of concurrent flow charts is treated as a single design object and all concurrent charts (including any hierarchical diagrams) are saved when any flow chart is saved. When HDL is generated for concurrent flow charts, separate VHDL processes (or *always* blocks in Verilog) are generated for each chart.

Note

Object names must be unique within the set of concurrent flow charts.


Adding a Concurrent Flow Chart

You can create a concurrent flow chart from within an existing flow chart using the  button or the **Ctrl + F2** shortcut keys or by choosing **Concurrent Flow Chart** from the **Add** menu. A new flow chart is created in the existing window with the same interface as the current chart.

The package list and any VHDL architecture declarations, Verilog module declarations or concurrent statements are shared by the concurrent flow charts but the sensitivity list and process declarations or local declarations can be set separately.

Each concurrent flow chart is given a unique name by adding an integer to the default name (for example: *process0*, *process1*, *process2*...). However, the base name for a new flow chart can be set as a preference in the **Default Values** tab of the Flow Chart Preferences dialog box.

Opening a Concurrent Flow Chart


You can open an existing concurrent flow chart from within an existing chart by using the  button or choosing **Open Flow Chart** from the **Diagram** menu and selecting from the menu of concurrent flow chart names. The concurrent flow chart is opened in the existing window. You can also open a concurrent flow chart by selecting it in the [diagram browser](#).

Renaming a Concurrent Flow Chart

You can change the name of the active concurrent flow chart by choosing **Rename Flow Chart** from the **Diagram** menu to display a Rename dialog box. The name is also shown in the window title for the flow chart. This name is used to uniquely identify concurrent flow charts in the generated HDL but can also be specified when there are no concurrent flow charts defined. If not specified, the name defaults to the value set in the flow chart preferences.

In a Verilog flow chart, the concurrent flow chart name is used to label the initial or always code for each chart and can be used (for example, by a disable statement) to reference this code.

Deleting a Concurrent Flow Chart

You can delete a concurrent flow chart from a set of concurrent flow charts by using the  button or choosing **Delete Flow Chart** from the **Diagram** menu and selecting from the menu of concurrent flow chart names.


When you select the name of a concurrent flow chart in this list, you are prompted for confirmation that the chart should be deleted. If you delete an open chart its window is closed. The titles for all other charts in the set of concurrent flow charts are updated. However, the *design explorer* view is not updated until you save the flow chart.

Note




You cannot delete the last concurrent flow chart in the set.

Editing Flow Chart Object Properties

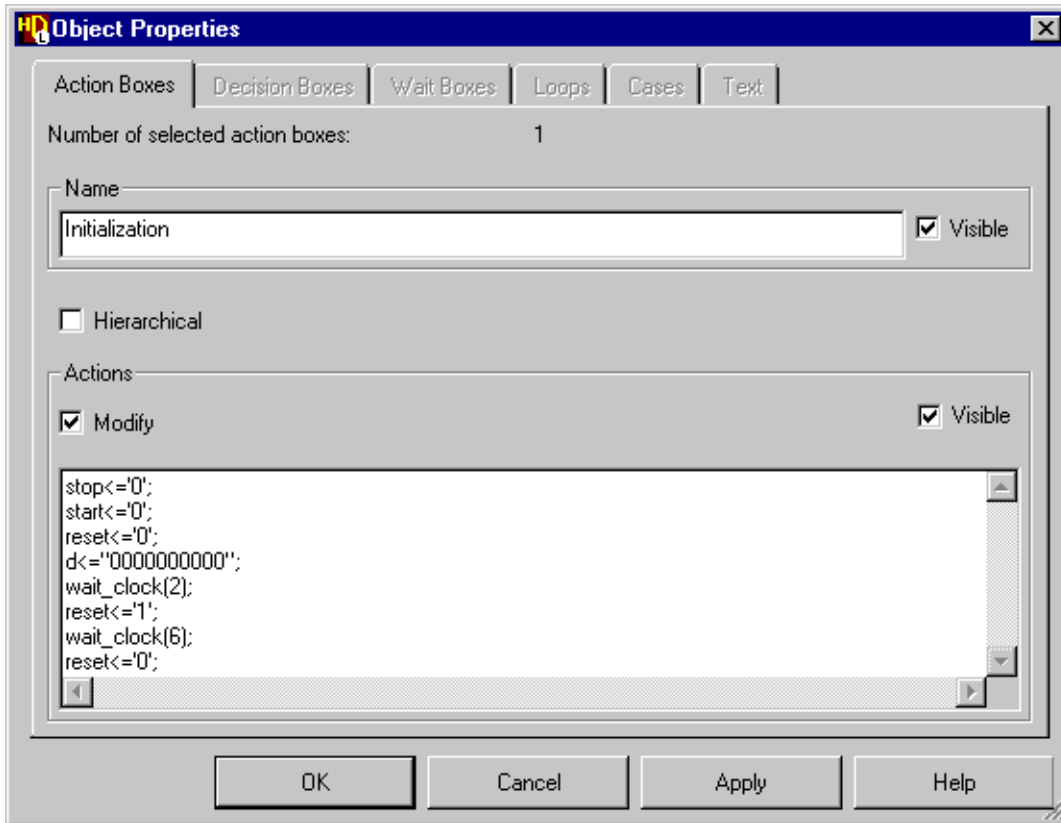
You can edit the properties for an object (or objects) on a flow chart by using the  button or choosing **Object Properties** from the **Edit** or popup menu. An Object Properties dialog box is displayed with separate tabs for **Action Boxes**, **Decision Boxes**, **Wait Boxes**, **Loops**, **Cases** and **Text** objects.

You can also edit most text properties (including object names, conditions and actions) directly on the diagram by clicking to select the text and clicking again to edit the text.

Editing Action Box Object Properties

You can edit the properties of one or more selected *action boxes* by using the  button or choosing **Object Properties** from the **Edit** menu or popup menu to display the **Action Boxes**

tab of the Object Properties dialog box. You can also display the tab directly by double-clicking on an existing action box on the diagram.



The action box name must be unique and can only be applied to a single selected action box.

If you change a hierarchical action box to a non-hierarchical action box, the child flow chart diagram (if it exists) and its contents are discarded. However, you can undo this change to recover the hierarchical action box and its child flow chart. If you change an action box to a hierarchical action box, any existing actions are transferred to a default action box in the child flow chart.

You can add or edit actions defined in the action box. If more than one action box is selected, you can use the Modify check box to choose whether the actions in the dialog box are applied to all the selected action boxes.

Note




An expression builder dialog box is automatically displayed when you begin to enter an action. Refer to “Building a HDL Expression” in the [State Machine Editors User Manual](#) for more information.

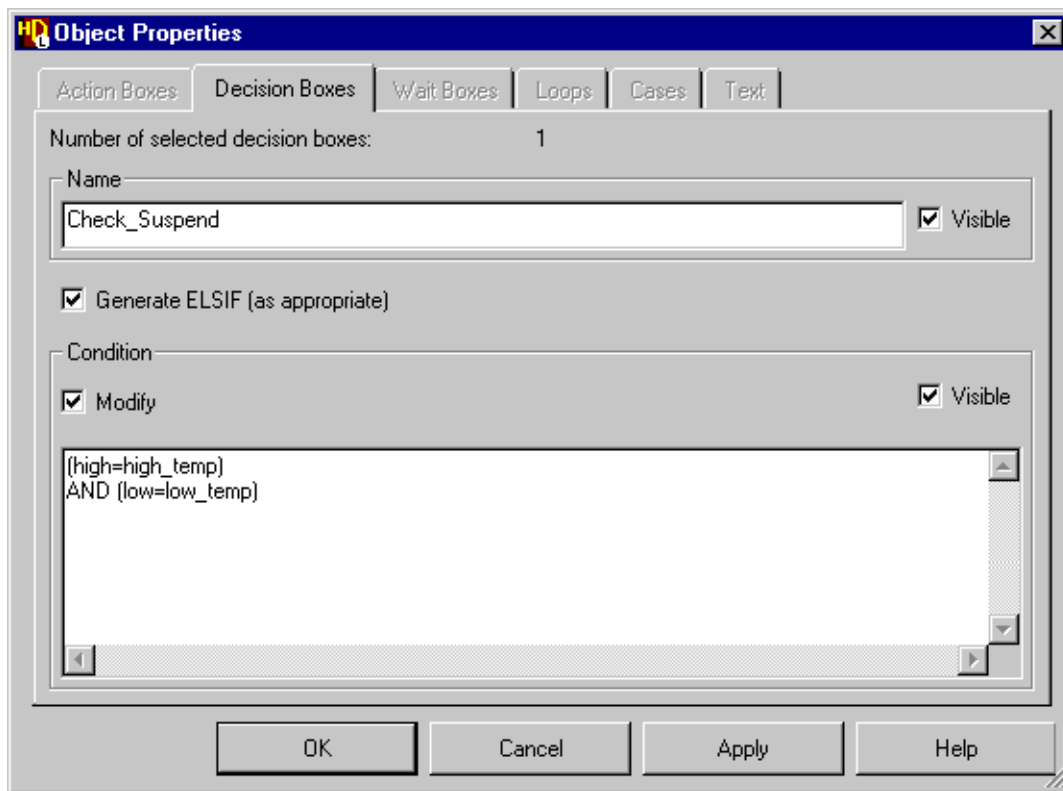
When you enter actions, the HDL syntax is automatically checked for the language of the diagram you are using (VHDL or Verilog). However, flow chart syntax checking can be

disabled by unsetting a preference. Note that you must include a terminating semi-colon for each statement although line breaks and indents can be used to improve legibility.

You can choose whether the action box name and actions are displayed or hidden on the diagram.

Editing Decision Box Object Properties

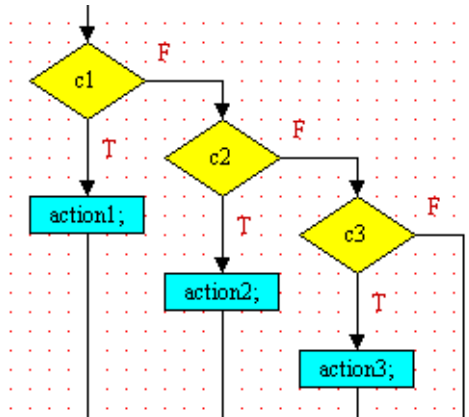
You can edit the properties of a one or more selected *decision boxes*) by using the  button or choosing **Object Properties** from the **Edit** menu or popup menu to display the **Decision Boxes** tab of the Object Properties dialog box. You can also display the tab directly by double-clicking on an existing decision box on the diagram.



The decision box name must be unique and can only be applied to a single selected decision box.

By default, if your flow chart uses nested decision boxes, combined *ELSIF* (VHDL) or *else if* (Verilog) statements are generated for the false branch if the decision box is the first and only statement on the false branch of another decision box.

For example, *c2* and *c3* in the following example:



VHDL:

```
IF c1 THEN
    action1;
ELSIF c2 THEN
    action2;
ELSIF c3 THEN
    action3;
END IF;
```

Verilog:

```
if (c1) begin
    action1;
end
else if (c2) begin
    action2;
end
else if (c3) begin
    action3;
end
```

If your downstream tool does not support combined else and if, you can unset this option in the dialog box or change the default behavior by unsetting the preference in the **Miscellaneous** tab of the Flow Chart Preferences dialog box. However, these options are ignored and separate instrumented else and if statements are always generated when the **Instrument for Animation** option is set in the **Generation** tab of the Flow Chart Properties dialog box.

You can add or edit the condition defined in the decision box. If more than one decision box is selected, you can use the Modify check box to choose whether the condition is applied to all the selected boxes.

Note




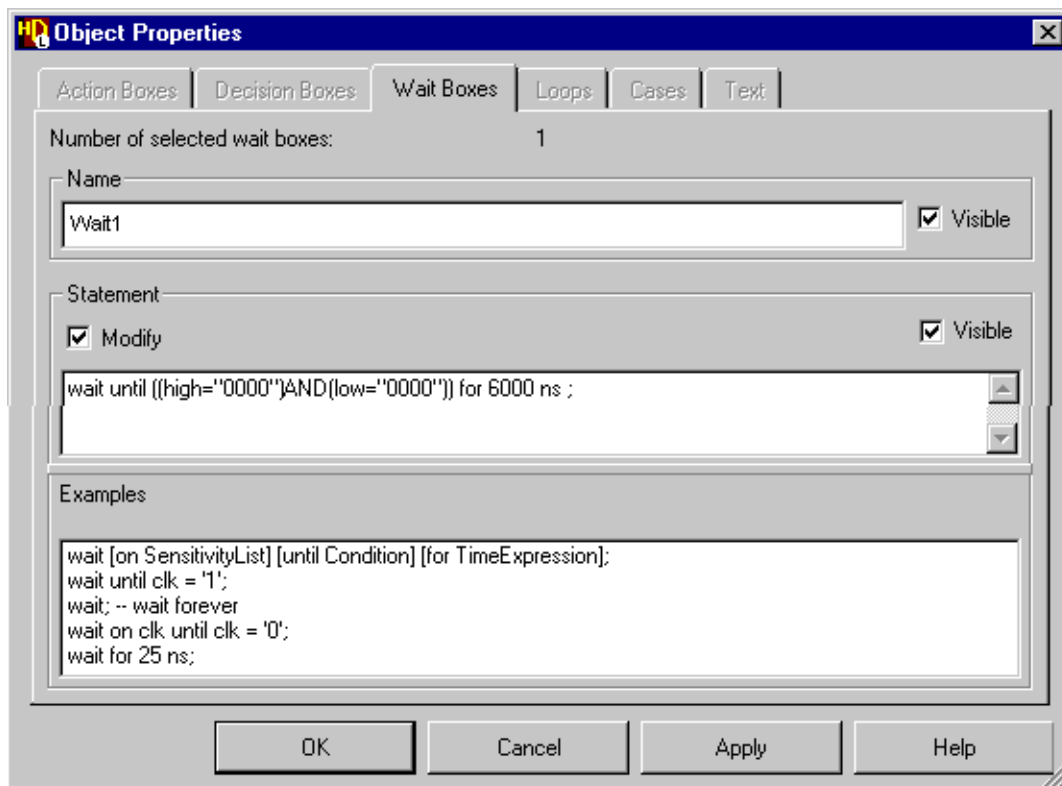
An expression builder dialog box is automatically displayed when you begin to enter a condition. Refer to “Building a HDL Expression” in the [State Machine Editors User Manual](#) for more information.

When you enter a condition, the HDL syntax is automatically checked for the language of the diagram you are using (VHDL or Verilog). However, flow chart syntax checking can be disabled by unsetting a preference.

You can choose whether the decision box name and condition are displayed or hidden on the diagram.

Editing Wait Box Object Properties

You can edit the properties of one or more selected *wait boxes* by using the  button or choosing **Object Properties** from the **Edit** menu or popup menu to display the **Wait Boxes** tab of the Object Properties dialog box. You can also display the tab directly by double-clicking on an existing wait box on the diagram.



The dialog box allows you to enter any valid HDL statements for the current hardware description language. Some examples of wait statements for the current language are given in the dialog box.

VHDL:

```
wait [on SensitivityList] [until Condition] [for TimeExpression];  
wait until clk = '1';  
wait; -- wait forever  
wait on clk until clk = '0';  
wait for 25 ns;
```

These can be inserted as templates by clicking on the required example with the **Left** mouse button.

The basic template for VHDL allows you to combine **on**, **until** and **for** clauses which may also include valid arithmetic operators.

For example:

```
wait until ((high="000")AND(low="0000")) for 6000ns;
```

In **Verilog**, you can also use the **@** operator to specify an event or **#** to specify a time delay:

```
wait (Expression);  
@Name;  
@(eventExpression);  
#UnsignedNumber;  
#ParameterName;  
#ConstantMinTypMaxExpression;  
#(MinTypMaxExpression);
```

For example:

```
wait ((high==4'd0)&&(low==4'd0));  
@(posedge clk);  
#12;  
#clk_prd;
```

Note




The default wait statement (*wait;*) corresponds to a wait for user interaction. Verilog system functions such as *\$stop* cannot be used in a wait box but can be used in an action box.

When you enter a wait statement, the HDL syntax is automatically checked for the language of the diagram you are using (VHDL or Verilog). However, flow chart syntax checking can be disabled by unsetting a preference.

You can choose whether the wait box name and statement are displayed or hidden on the diagram.

Editing Loop Object Properties

You can edit the properties of one or more selected *loops* by using the  button or choosing **Object Properties** from the **Edit** menu or popup menu to display the **Loops** tab of the Object Properties dialog box. You can also display the tab directly by double-clicking on an existing wait box on the diagram.

By default, a loop will repeat forever. However, you can clear this option in the dialog box and enter a specific loop control clause. Example clauses for *repeat* (if you are using Verilog), *for* and *while* loops are given in the dialog box.

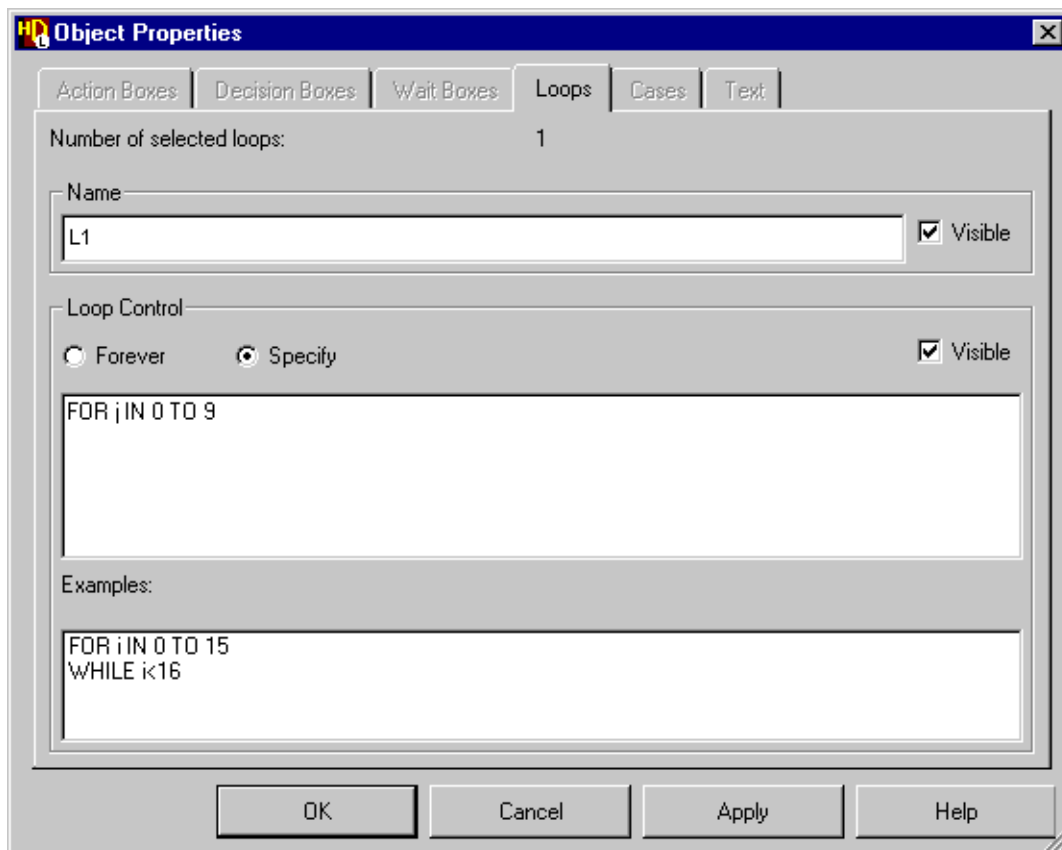
VHDL:

```
FOR i IN 0 TO 15  
WHILE i<16
```

Verilog:

```
repeat (n)  
for (i=0;i<16;i=i+1)  
while (i)
```


These can be inserted as templates by clicking on the required example with the **Left** mouse button.

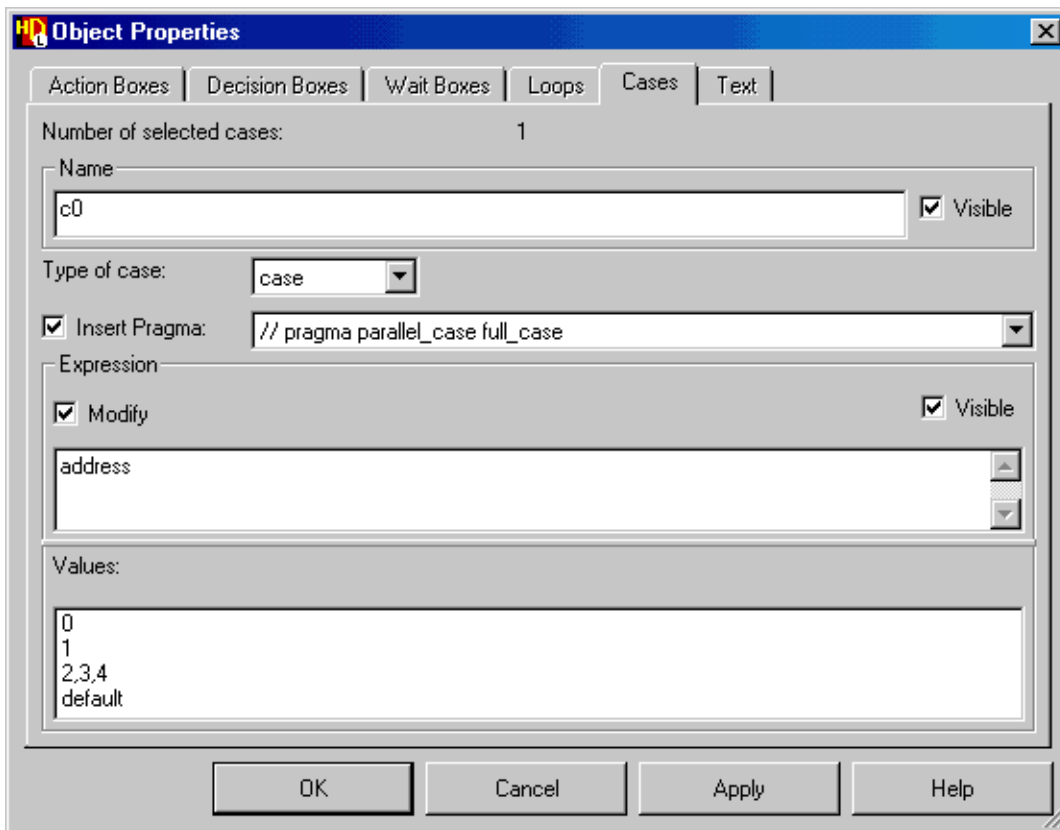


When you enter a loop statement, the HDL syntax is automatically checked for the language of the diagram you are using (VHDL or Verilog). However, flow chart syntax checking can be disabled by unsetting a preference.

You can also choose whether the loop name and loop control clause are displayed or hidden on the diagram.

Editing Case Object Properties

You can edit the properties of one or more selected *case boxes* by using the  button or choosing **Object Properties** from the **Edit** menu or popup menu to display the **Loops** tab of the Object Properties dialog box. You can also display the tab directly by double-clicking on an existing case box on the diagram.



The image shows the 'Object Properties' dialog box with the 'Cases' tab selected. The dialog has a title bar with the HDG logo and a close button. Below the title bar are tabs for 'Action Boxes', 'Decision Boxes', 'Wait Boxes', 'Loops', 'Cases', and 'Text'. The 'Cases' tab is active. Inside the dialog, it shows 'Number of selected cases: 1'. There is a 'Name' field containing 'c0' and a 'Visible' checkbox which is checked. Below this is a 'Type of case:' dropdown menu set to 'case'. Then there is a checked 'Insert Pragma:' checkbox followed by a dropdown menu showing '// pragma parallel_case full_case'. Below that is an 'Expression' section with a checked 'Modify' checkbox and a 'Visible' checkbox which is checked. The expression field contains 'address'. At the bottom is a 'Values:' list box containing '0', '1', '2,3,4', and 'default'. At the very bottom are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

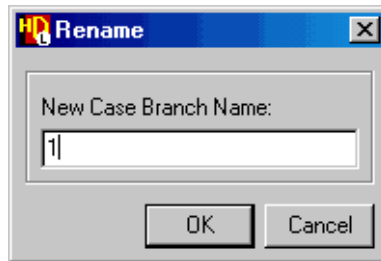
The case box name must be the same as the corresponding end case object and otherwise unique on the diagram.

If you are using Verilog, you can choose to use *casex* or *casez* comparison as an alternative to the default bit comparison *case* style and you can insert the following pragmas to specify full case or parallel case statements:

full_case	All possible branches have been specified, any missing branches cannot occur and a default branch need not be generated.
parallel_case	Branches are mutually exclusive.
parallel_case full_case	All possible branches have been specified and are mutually exclusive.

You can specify a case expression which can comprise any valid HDL statements entered as free-format text (using multiple lines if required).

The dialog box also lists the existing output port names which should correspond to possible values for the evaluated expression. These values can be edited by double-clicking over the existing name in the dialog box to display a Rename dialog box.



When you enter an expression or port name, the HDL syntax is automatically checked for the language of the diagram you are using (VHDL or Verilog). However, flow chart syntax checking can be disabled by unsetting a preference.

You can also choose whether the case box name and expression are displayed or hidden on the diagram.

Setting Flow Chart Properties

There are a number of properties associated with a flow chart which can be accessed from the Flow Chart Properties dialog box. The following properties are shown as text objects on the flow chart (or on the top-level diagram of a hierarchical flow chart):

Architecture or Module Declarations	A list of user defined VHDL architecture declarations or Verilog module declarations.
Concurrent Statements	A statement block containing a list of concurrent statements that are included in the generated HDL.

Sensitivity List	A list of signals which cause the corresponding process to execute if any of the signals change.
Process or Local Declarations	A list of statements which are included as process declarations in the generated VHDL or as local declarations in the generated Verilog.

Concurrent statements and architecture or module declarations are applied to all concurrent flow charts. However, process or local declarations and the sensitivity list are set separately for each diagram in a set of concurrent flow charts.

You can edit the flow chart properties by choosing **Flow Chart Properties** from the **Diagram** menu to display a dialog box with tab options for:

Generation

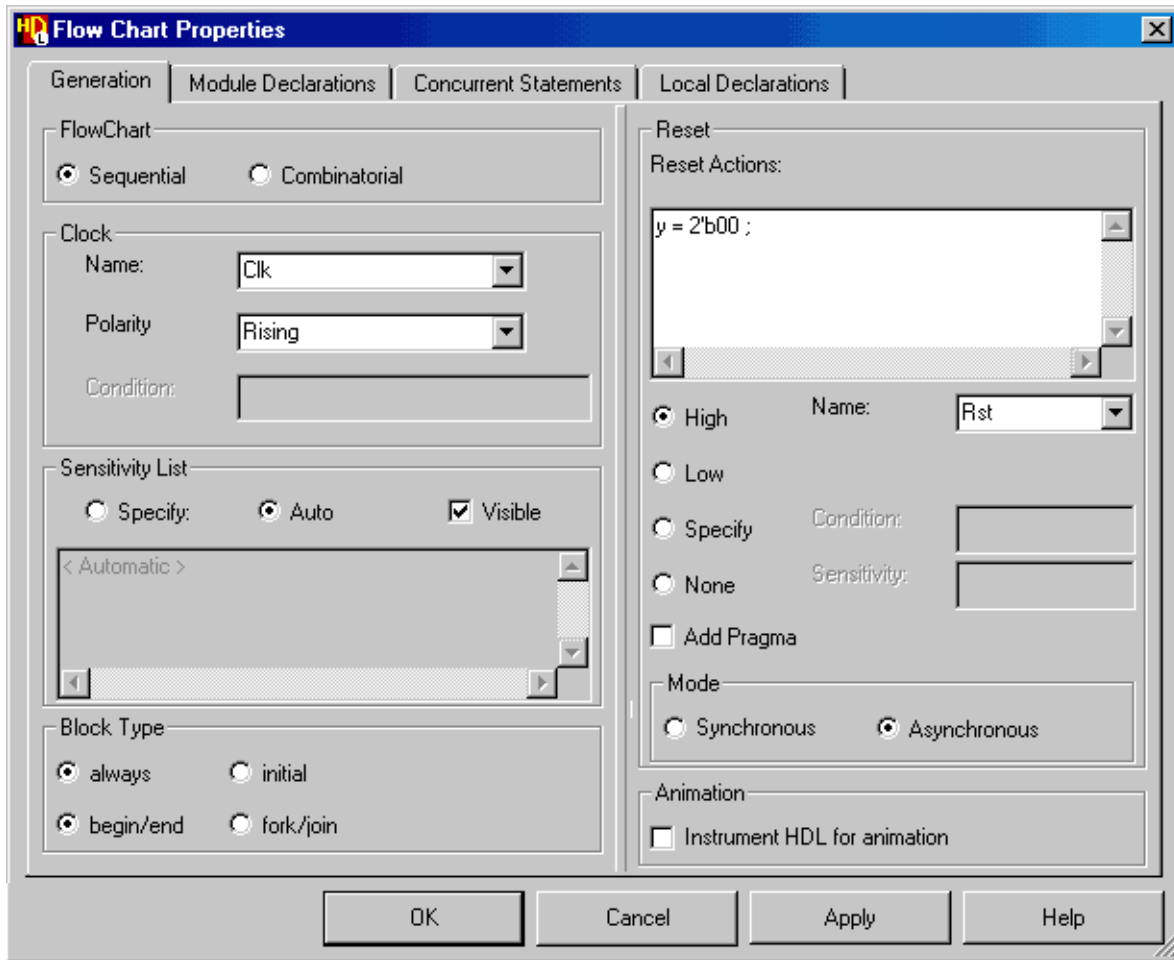
Architecture Declarations (VHDL only) or Module Declarations (Verilog only)

Process Declarations (VHDL only) or Local Declarations (Verilog only)

Setting Flow Chart Generation Properties

You can set the HDL generation properties by choosing **Flow Chart Properties** from the **Diagram** or popup menu and then selecting the **Generation** tab in the Flow Chart Properties dialog box.

For example, the following dialog box is displayed when you are using Verilog:



Note



Separate generation properties can be specified for each diagram in a set of concurrent flow charts.

Sequential and Combinatorial Diagrams

You can specify whether the flow chart is sequential or combinatorial.

A sequential flow chart changes state on active clock edges. An optional reset signal can be used to perform specified reset actions.

A combinatorial flow chart operates independently of any clock and is typically used for applying stimulus and testing results in a test bench.

Clock Signal

When the sequential option is selected, you can enter the clock signal or choose from a dropdown list of available input signals.

You can also set the clock edge sensitivity.

For a Verilog view, you can choose *Rising* or *Falling* corresponding to *posedge* or *negedge* sensitivity.

For a VHDL view, you can choose *Rising*, *Falling*, *Rising Last*, *Falling Last*. These options generate the following VHDL clock edge expressions:

Rising	<code>clk'EVENT AND clk = '1'</code>
Falling	<code>clk'EVENT AND clk = '0'</code>
Rising Last	<code>clk'EVENT AND clk = '1' AND clk'LAST_VALUE = '0'</code>
Falling Last	<code>clk'EVENT AND clk = '0' AND clk'LAST_VALUE = '1'</code>

Alternatively for either language, you can choose *Specify* to enter any other valid edge condition.

Reset Signal

When the sequential option is selected, you can enter a list of reset actions and enter a synchronous or asynchronous reset signal (or choose from a dropdown list of available input or internal signals). You can choose to trigger the reset on a *High* or *Low* level signal (or choose *Specify* to enter any other valid reset condition).

If you have specified a Verilog reset condition, you must also specify any additional signals required in the sensitivity list. (Multiple signals should be separated by an *OR* operator.)

You can optionally set pragmas (*sync_set_reset_local* or *async_set_reset_local*) which identify the name of the currently specified synchronous or asynchronous reset signal.

Note



The pragmas are entered using the keyword (*pragma*, *synopsys* or *exemplar*) preference set in the **Style** tab of the VHDL or Verilog Options dialog box.

Sensitivity List

You can also specify a sensitivity list of signals which cause the corresponding process to execute if any of the signals change. Multiple signals should be separated by the comma character for VHDL and by an *or* when using Verilog.

You can choose whether the sensitivity list is visible or hidden on the diagram. If the user-entered list is visible, it can also be edited by direct text editing on the diagram.

If the sensitivity list is not specified it is shown as *<Automatic>* in the dialog box and is automatically created during HDL generation using the following rules:

Sequential with synchronous or no reset	<clock name>
Sequential with asynchronous reset	<clock name>, <reset name>
Combinatorial	All input and internal signals

Block Type

For Verilog designs, you can specify whether *initial* or *always* style code is generated. You can also choose whether the generated code is contained between *begin* and *end* statements or using *fork* and *join* statements.

Animation

You can choose to instrument the HDL for animation. When this option is enabled, activity information is available for flow chart animation. This is achieved by additional code included in the generated HDL. The additional code is surrounded by translation control pragmas which ensure that it is ignored by downstream synthesis tools.

The HDL generated for nested decision boxes on a flow chart is normally created with ELSIF statements for each decision box. However, separate ELSE and IF statements instrumented by animation pragmas are generated when you choose the **Instrument HDL for Animation** option. You should regenerate HDL with the animation option unset if you want to use the generated HDL for synthesis.

Note



Note that you cannot instrument the generated HDL for animation when a flow chart is used to define an embedded view in a block diagram.

The new flow chart properties are set when you confirm the dialog box.

Note



The generation properties (other than the sensitivity list) are not normally shown on the diagram. However, you can set preferences in the **Headers** tab of the VHDL and Verilog Options dialog boxes to include the generation properties as comment text after the header in the generated HDL.

Editing Architecture or Module Declarations

You can edit architecture declarations (when using VHDL) or module declarations (when using Verilog) by choosing **Flow Chart Properties** from the **Diagram** or popup menu or by double-clicking over the Architecture Declarations (VHDL) or Module Declarations label (Verilog) on the diagram.

The **Architecture Declarations** (or the **Module Declarations**) tab of the Flow Chart Properties dialog box is displayed which allows you to enter any valid HDL statements for the current hardware description language in a free-format entry box.

Signal declarations, constants, variables, comments, procedures, functions or type definitions can be included in the declaration.

Typically a VHDL declaration, comprises a keyword (signal, constant or variable) followed by a name, type and value. The specified type must be one of the standard predefined types or a type defined in a VHDL package. An initial value is required when the declaration is a constant but is optional when you declare a signal or variable.

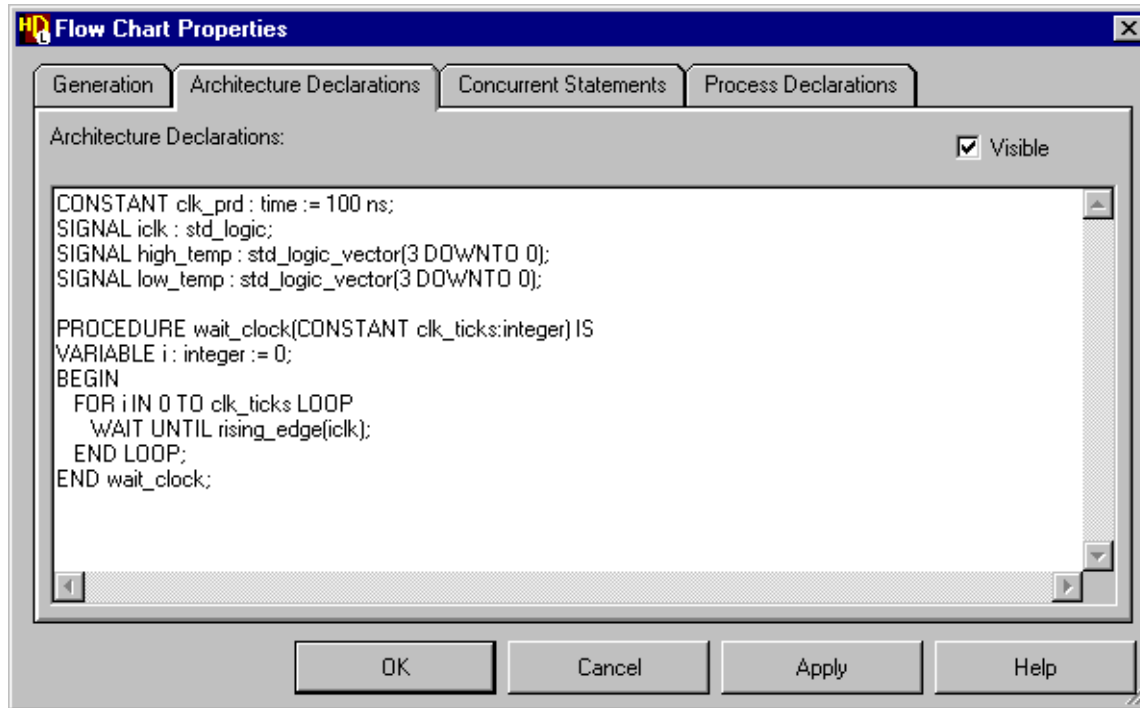
A typical Verilog declaration, comprises a keyword followed by appropriate parameters as given below:

Table 8-3. Verilog Declarations

Keyword	Parameters		
'define	name	value	
parameter	name	value	
reg	range (optional) ¹	name	array (optional)
integer	name	array (optional)	
real	name		
time	name	array (optional)	
wire	range (optional)	name	array (optional)

1. Verilog range and array parameters should be entered in the format *[m:n]*.

For example, the following picture shows the Flow Chart Properties dialog box being used to enter an architecture declaration:



The declarations are inserted at the top of the VHDL architecture or Verilog module in the generated HDL in the order they are listed and apply to all diagrams in a set of concurrent flow charts.

The syntax is checked and the declarations are added as a text object on the flow chart (or the top level diagram when you are editing a hierarchical flow chart) when you confirm the dialog box.

You can choose whether the declarations are visible or hidden on the diagram.

Note



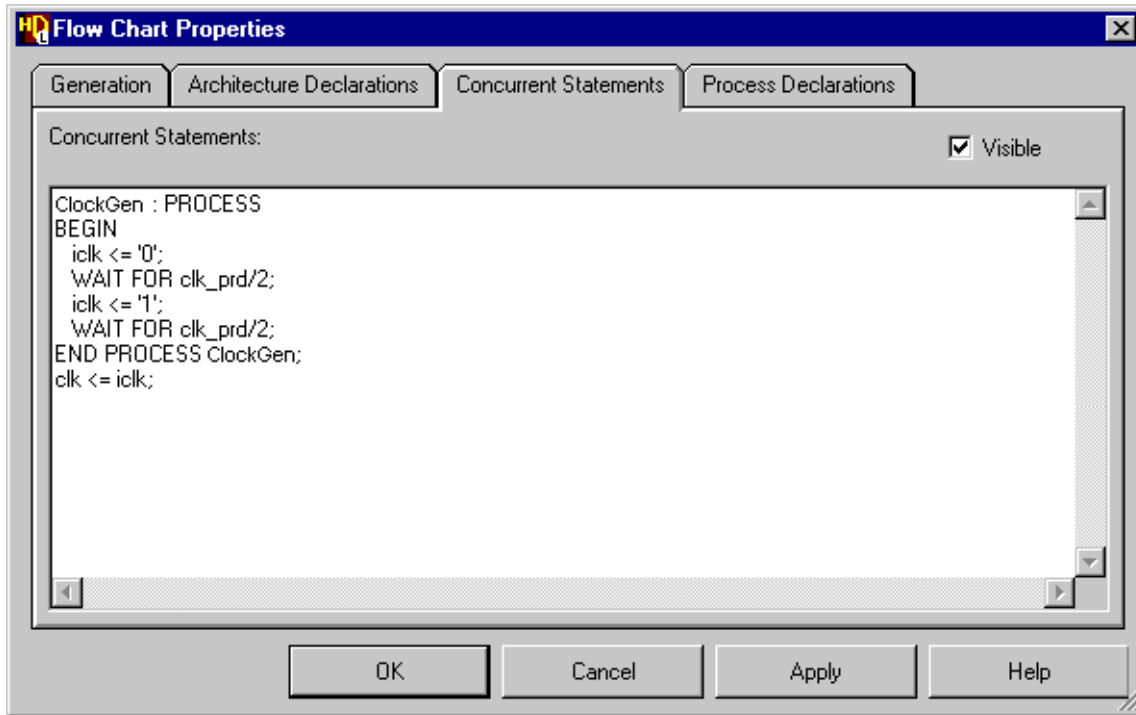
Architecture declarations or module declarations cannot be set when a flow chart is used to define an embedded view in a block diagram or IBD view. However, any declarations required by the embedded view can be set on the parent view.

Editing Concurrent Statements

You can edit *concurrent statements* by choosing **Flow Chart Properties** from the **Diagram** or popup menu or by double-clicking on an existing Concurrent Statements label on the diagram.

The **Concurrent Statements** tab of the Flow Chart Properties dialog box provides a free-format entry box for you to add or edit these HDL statements.

You can also choose whether the concurrent statements are visible or hidden on the diagram (or on the top level diagram when you are editing a hierarchical flow chart).



Note

An expression builder dialog box is automatically displayed when you begin to enter a concurrent statement. Refer to “Building a HDL Expression” in the [State Machine Editors User Manual](#) for more information.

The edited statements are added to the diagram when you confirm the dialog box.

When you enter statements, the syntax is automatically checked for the hardware description language of the active diagram. However, flow chart syntax checking can be disabled by unsetting a preference.

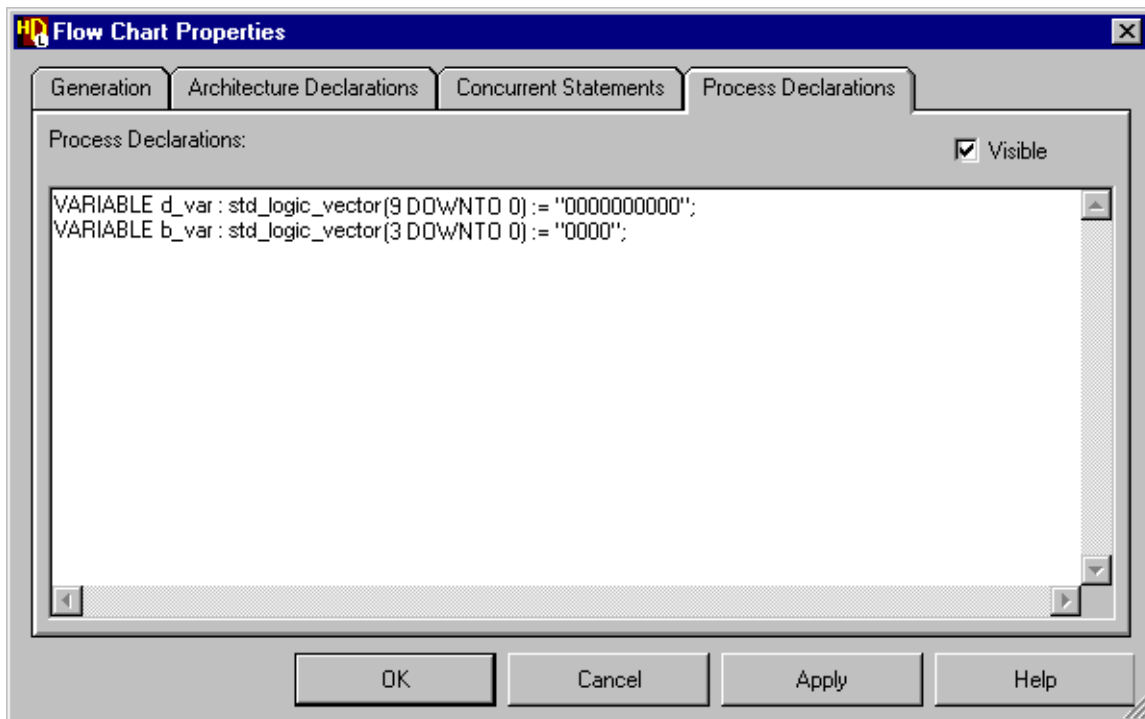
Concurrent statements are included in the generated HDL at the end of the VHDL architecture or Verilog module and are applied to all diagrams in a set of concurrent flow charts. They are executed concurrently with all of the processes (or *always* blocks) in the flow chart and are typically used for additional datapath operations or specialized clocking.

You can also edit the statements directly on the diagram by clicking on the action to select the text and clicking again to edit the text.

Editing Process or Local Declarations

You can add or edit flow chart process declarations (when using VHDL) or local declarations (when using Verilog) by choosing **Flow Chart Properties** from the **Diagram** menu or by double clicking on the Process Declarations (VHDL) or Local Declarations (Verilog) label on the flow chart.

The **Process Declarations** (or **Local Declarations**) tab of the Flow Chart Properties dialog box allows you to add or edit declaration statements within a VHDL process or within a local Verilog *initial* or *always* block. These tabs are typically used to declare constants and parameters (or for 'define statements when using Verilog).



You can also choose whether the declarations are visible or hidden on the diagram.

For a VHDL flow chart, process declarations are placed at the beginning of the process immediately before the **BEGIN** keyword in the generated HDL.

For a Verilog flow chart, local declarations are placed after the *initial* (or *always*) keyword in the generated HDL. (You can choose whether initial or always code is generated using the **Generation** tab of the dialog box.)

Separate process declarations can be specified for each diagram in a set of concurrent flow charts.

The edited statements are added to the flow chart when you confirm the dialog box.

You can also edit the statements directly on the diagram by clicking to select the text and clicking again to edit the text.

The syntax is automatically checked when you enter VHDL statements. However, flow chart syntax checking can be disabled by unsetting a preference.

Setting Flow Chart Preferences

You can set flow chart preferences by choosing **Flow Chart** from the **Master Preferences** cascade of the **Options** menu in the *design manager*.

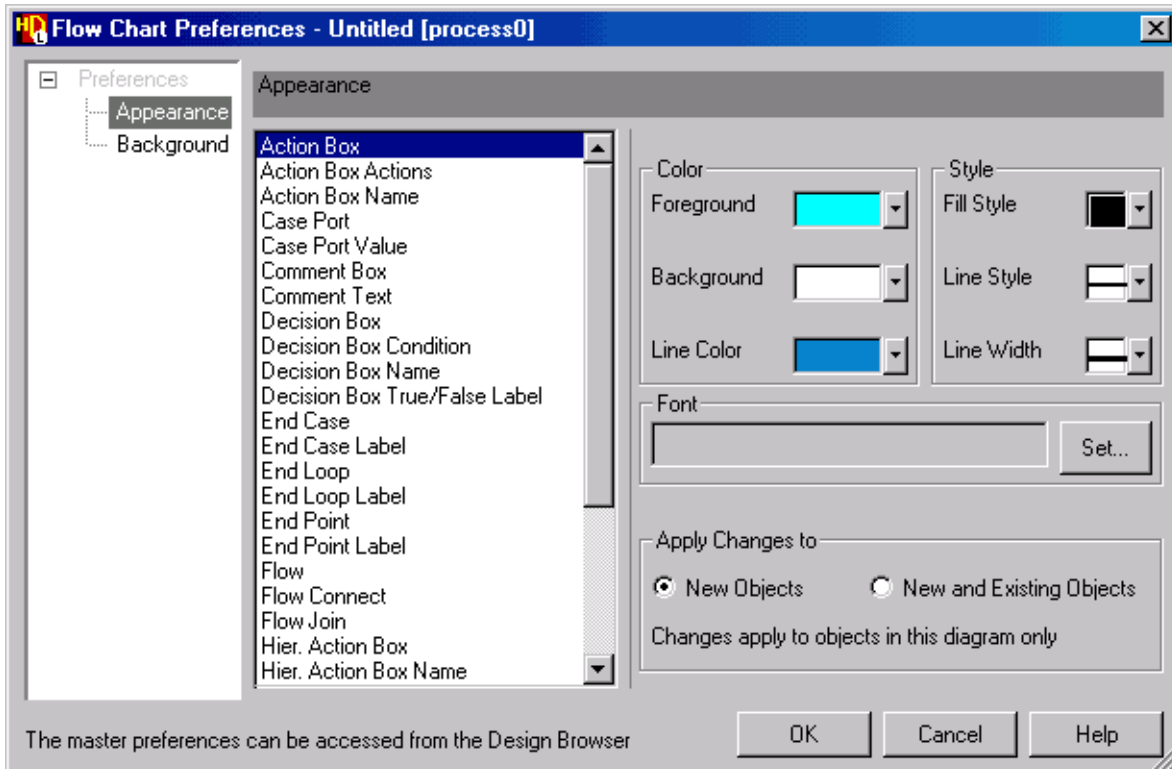
The Flow Chart Preferences dialog box has separate pages for **General**, **Default Settings**, **Object Visibility**, **Appearance** and **Background** preferences.

Note



The general, default settings and object visibility preferences take effect on the next flow chart you open and can only be edited when the dialog box is displayed from the **Master Preferences** cascade in the design manager **Options** menu. These pages are not available when you choose **Diagram Preferences** in a flow chart window.

The **Appearance** page allows you to set default visual attributes for individual flow chart objects.



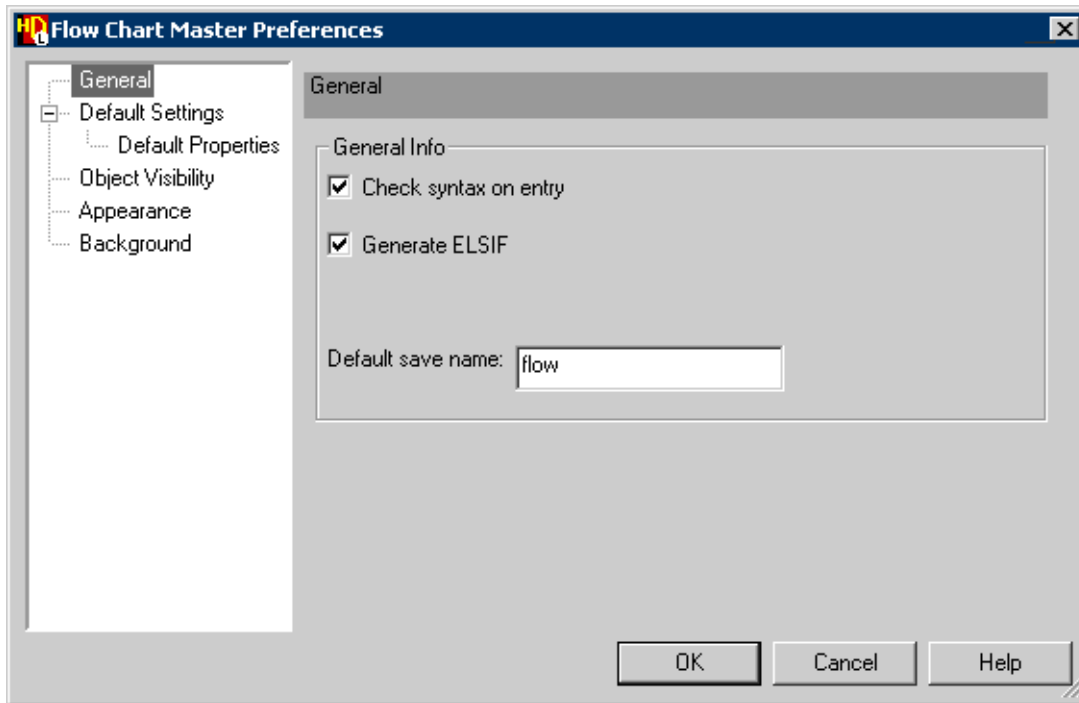
The attributes include the foreground and background colors, line color and style, fill style, line width and the text font. Some attributes may not always be available. For example, line style, width and color attributes are not available for a text object.

Refer to [“Setting Visual Attributes”](#) on page 83” for more information.

This page can also be edited by choosing **Diagram Preferences** from the **Options** menu in a flow chart. When you edit preferences for the active diagram, the dialog box allows you to choose whether the preferences are applied to new objects or to both new and existing objects in the flow chart (including concurrent or hierarchical diagrams).

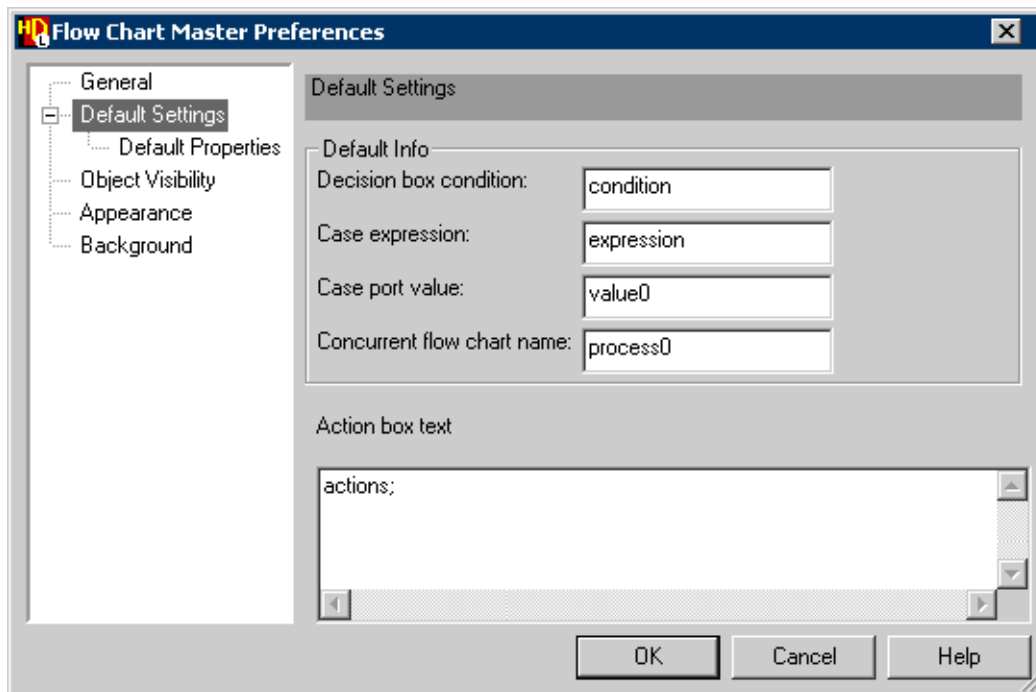
You can save the appearance preferences set on the active diagram as master preferences by choosing **Update from Diagram** in the **Master Preferences** cascade of the **Options** menu or you can apply the master preferences to the active diagram by choosing **Apply to New Objects** or **Apply to New and Existing Objects**.

The **General** page allows you to set other flow chart options:



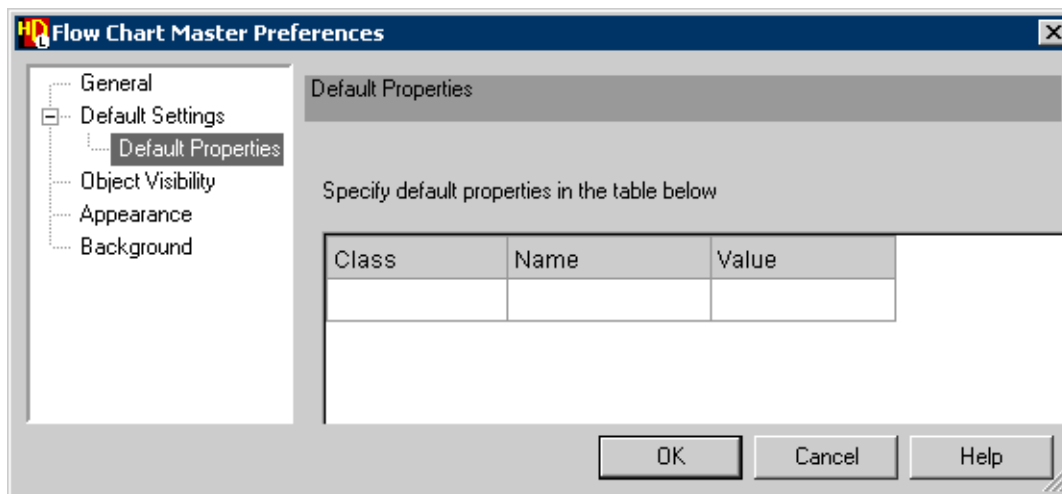
These options include syntax checking, an option to generate ELSIF statements and the default save name for flow charts.

The **Default Settings** page allows you to set flow chart default values:



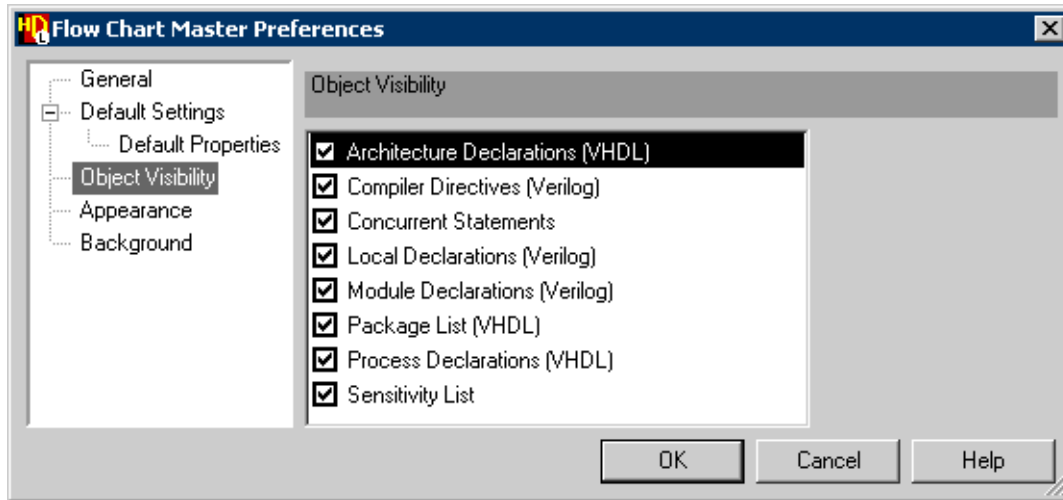
You can set default names for decision box conditions, case expressions and port values, the concurrent flow chart name and the default actions used in an action box.

You can use the **Default Properties** sub-page to define default properties for flow chart views.



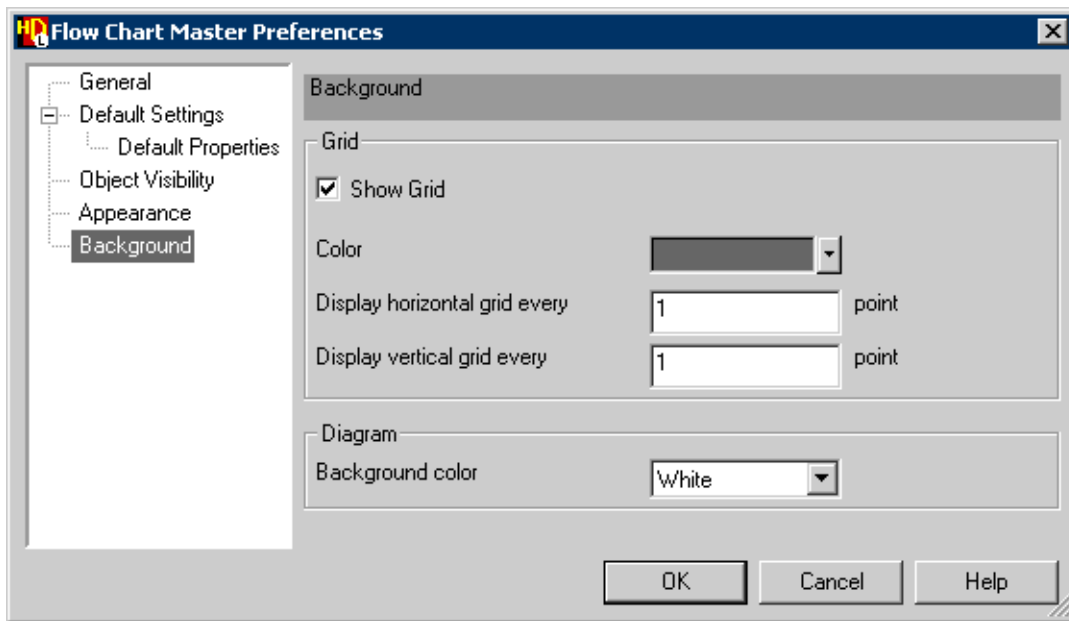
Refer to the [HDL Designer Series User Manual](#) for information about “Using View Property Variables”.

The **Object Visibility** allows you to set the default object visibility for multi-line text objects on the diagram.



Refer to [“Changing Text Visibility”](#) on page 68 for more information.

The **Background** page allows you to control the diagram background color and grid attributes used by the flow chart editor.



These preferences are described in [“Setting Background Preferences”](#) on page 50.

Refer to the “Default Preferences” appendix in the *HDL Designer Series User Manual* for lists of the default preferences set when you invoke a HDL Designer Series tool for the first time.

Chapter 9

Truth Table Editor

This chapter describes how a design function can be represented as a matrix of true or false input signals and their resultant output signal (or signals). This *truth table* can be used to describe a *design unit view* of any *block* or *component* in a *block diagram* or *IBD view* and *HDL* can be generated from the truth table.

Truth Table Notation.....	379
Comparison Operators	380
Truth Table Toolbars	380
Editing a Truth Table Cell	380
Adding a Column or Row	381
Deleting a Column or Row	381
Setting Truth Table Properties.....	381
Setting Truth Table Generation Properties	382
Editing Architecture or Module Declarations.....	386
Editing Concurrent Statements	387
Editing Process or Local Declarations	387
Editing Global Actions	388
Case and IF Style Truth Tables	389
Setting Truth Table Preferences.....	391

A truth table may typically be used to convert between digital and analog data input signals to describe a decoder or multiplexor function.

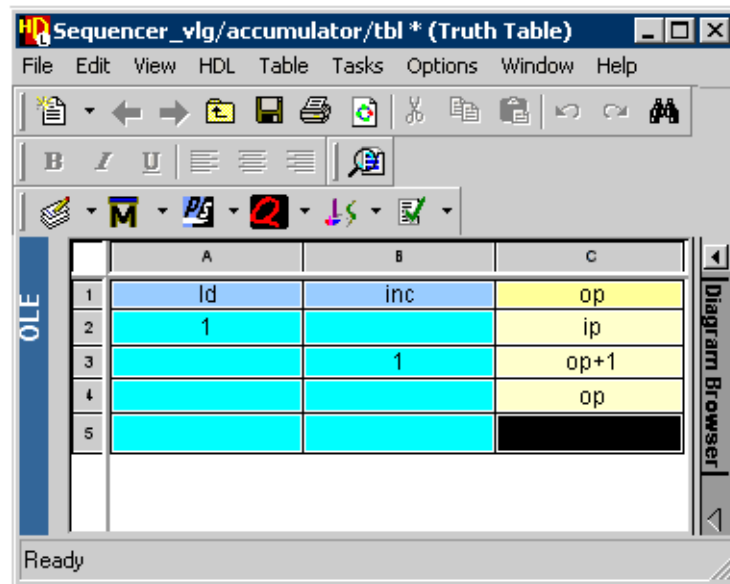
The truth table editor is implemented as a spreadsheet which supports extended behavioral and comparison functions as well as simple combinatorial or sequential truth tables.

A sequential truth table can be defined by setting clock and reset information as generation properties which are added to the VHDL process (or Verilog module) when HDL is generated for the truth table.

The syntax and semantics of the truth table are checked during HDL generation and the cells corresponding to any errors encountered are highlighted on the truth table.

When you create a new truth table as a child view from within a block diagram or IBD view, an input column is automatically created for each input port and an output column for each output

port. Alternatively, you can set preferences for the initial number of columns and rows in a new truth table which has no parent view.



In the following example, the output *ser_if_data* is assigned the value of the *xmitdt*, *recvdt* or *status* input registers depending on which enable signal is set. For any other inputs, the output is assigned the value "00000000".

	A	B	C	D
1	xmitdt_en	status_en	recvdt_en	ser_if_data
2	'1'			xmitdt
3		'1'		status
4			'1'	recvdt
5				"00000000"

Truth Table Notation

The truth table editor displays a spreadsheet containing input and output columns. The columns are numbered using alphabetic characters and rows using numeric characters so that any cell can be uniquely referenced. (For example, *C4* references the cell in column *C*, row 4.)

	A	B	C	D	E	F
1	Input0	Input1	Input2		Output	
2	'0'	'0'			0	
3	'0'	'1'			1	
4	'1'	'0'			2	
5					3	
6						
7	'0'		'0'		0	
8	'0'		'1'	Internal='0'	1	
9	'0'		'1'	Internal='1'	4	
10	'1'		'0'		2	Internal<='1'
11					3	

The first row is a header row which contains the names of input and output variables (typically an input, output or bidirectional net name). Subsequent rows may contain a value for the variable (which assumes the equality operator), an expression preceded by a comparison operator or a more complex expression.

For example, the scalar VHDL value '1' corresponds to the default expression *<input variable> = true*.

The output column (or columns) typically contain an expression defining the value of an output variable (or variables). An empty row of input cells (such as rows 5 or 11 in the picture above) represents the *OTHERS* condition which assigns output values when none of the input expressions are true.

You can use an empty row of input and output cells (such as row 6 in the picture above) to separate groups of cells. These cells are described by separate If or Case statements when HDL is generated for the truth table.

Additional conditions (for example, an expression using an internal signal or variable name) can be added in one or more unnamed input columns and additional actions added in unnamed output columns. The conditions or actions are generated in the order of the columns.

All values, expressions, conditions and actions must be valid for the hardware description language. If any syntax errors are encountered during HDL generation, an error message is issued and the corresponding cells are highlighted on the truth table.

Truth Table Toolbars

There is no toolbar specific to the truth table. However the standard, HDL tools, tasks and format text toolbars are available.

Refer to the [HDL Designer Series User Manual](#) for general information about these toolbars and general information about the HDL Designer Series user interface.

Editing a Truth Table Cell



You can add text to a truth table editor cell by simply selecting the cell and typing the required expression. Any existing text is overwritten or you can double-click in a cell to explicitly add new or edit existing text characters.

VHDL scalar values should be entered in single quotes (for example: '0' or '1'; VHDL array values should be entered in double quotes (for example: "00000000" or "101"). Verilog scalar values should be entered without quotes (for example: 0 or 1; VHDL array non-decimal values should be entered with a width and base prefix (for example: 3'b101, 8'b00000000 or 15'h107d).

By default, you can add or edit a single line in each cell. However, you can allow multiple lines in any cell by choosing **Change To Multi-line** from the **Table** or popup menu or return to single line mode by choosing **Change To Single-line**. Multi-line cells are outlined to distinguish them from single-line cells and are highlighted with a red background when selected.

Note



The  key is disabled and the  key creates a new line when multi-line mode is enabled. If you change a multi-line cell to single-line, the cell contents are concatenated on one line.

Refer to “[Table Editor Windows](#)” on page 87 for information about selecting, editing and resizing table cells.

Comparison Operators

When you enter a value in a truth table input column, the equality operator is assumed by default. However, any of the following operators can be used:

Operator	VHDL	Verilog
equal (default)	=	== or ===
not equal	/=	!= or !==
less than	<	<

greater than	>	>
less than or equal	<=	<=
greater than or equal	>=	>=

Note



Expressions using the === or !== operators are not synthesizable.

Adding a Column or Row

You can add a column to a truth table by choosing **Add Column** from the **Table** or popup menu.

If more than one cell is selected, the corresponding number of new columns are added to the left of the selected column(s). If the selected cells are input columns, the new columns are added as input columns. If the selected cells are output columns, the new columns are added as output columns. If the selected cells include both input and output columns the new columns are added as input columns.

You can add a row to a truth table by choosing **Add Row** from the **Table** or popup menu.

If more than one cell is selected, the corresponding number of new rows are added above the selected row(s). However you cannot add rows when the header row is selected.

Deleting a Column or Row

You can delete a column or group of selected columns from a truth table by choosing **Delete Column** from the **Table** or popup menu. However, you cannot delete all the columns. All truth tables must contain at least one input column and one output column.

You can delete a row or group of selected rows from a truth table by choosing **Delete Row** from the **Table** or popup menu. However, you cannot delete the header row and there must be at least one row for input comparison and output assignment.

Setting Truth Table Properties

You can edit the truth table properties by choosing **Truth Table Properties** from the **Table** menu to display the Truth Table Properties dialog box with tab options for:

Generation

You can specify a sequential or combinatorial truth table generated with If or Case statements and specify the sensitivity list.

Architecture or Module Declarations	A list of user defined declarations which are included at the start of the VHDL architecture or Verilog module in the generated HDL.
Concurrent Statements	A list of concurrent statements that are included at the end of the VHDL architecture or Verilog module in the generated HDL.
Process or Local Declarations	A list of statements which are included at the start of the process in the generated VHDL or at the start of the local <i>always</i> code in the generated Verilog.
Global Actions	Common actions that are always performed when the truth table is evaluated.

Note

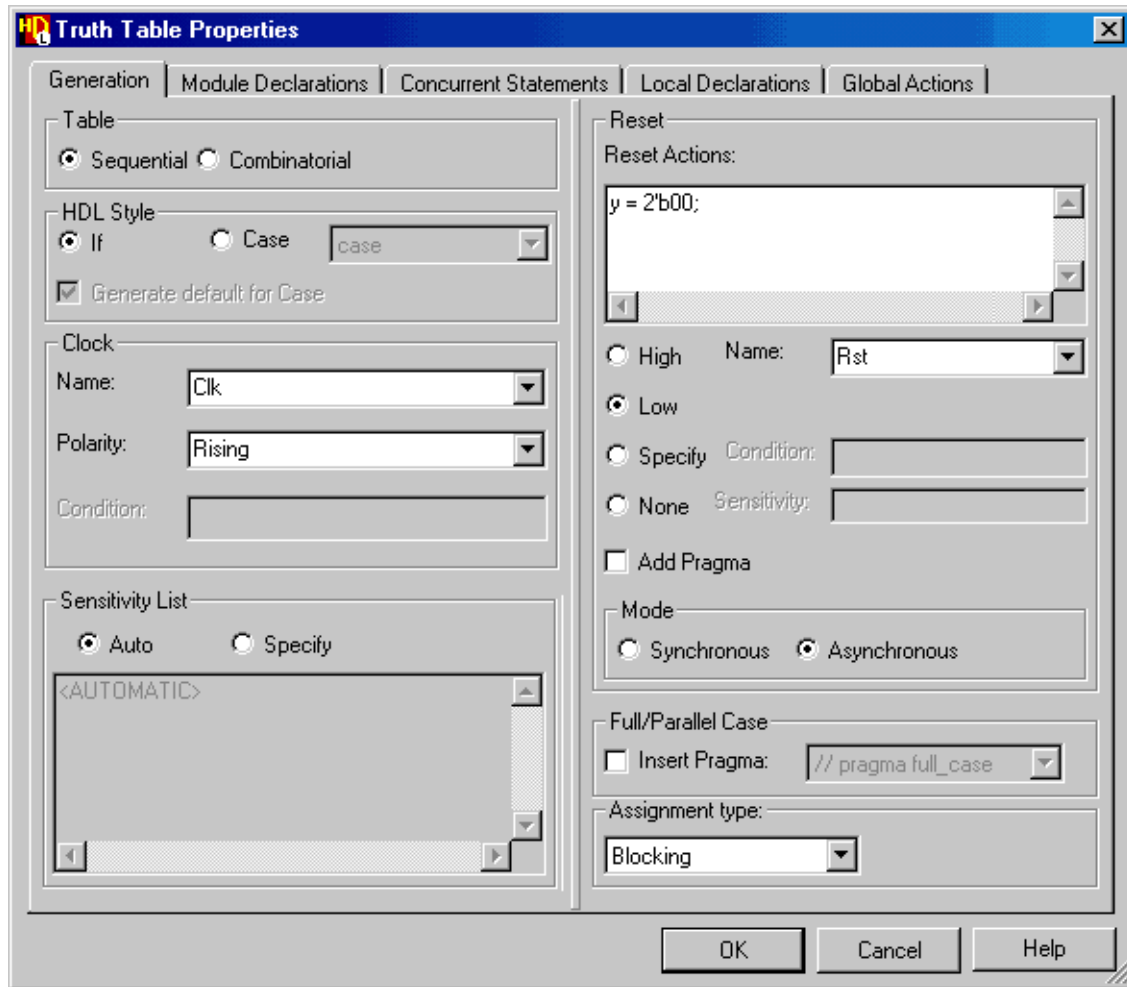


You can set preferences in the **Headers** tab of the VHDL and Verilog Options dialog boxes to include the generation properties as comment text after the header in the generated HDL.

Setting Truth Table Generation Properties

You can set the HDL generation properties by choosing **Truth Table Properties** from the **Table** menu and then selecting the **Generation** tab in the Truth Table Properties dialog box.

For example, the following dialog box is displayed when you are using Verilog:



Sequential and Combinatorial Diagrams

You can specify whether the truth table is sequential or combinatorial.

A sequential truth table changes state on active clock edges. An optional reset signal can be used to perform specified reset actions.

A combinatorial truth table operates independently of any clock and is typically used for decoding an address bus.

HDL Style

You can specify whether HDL is generated using If or Case styles. When Case style is selected you can choose to generate a default *Others* condition when not this condition is not specified by an empty input row in the truth table.

When using Verilog, you can also choose to use *casex* or *casez* comparisons as an alternative to bit comparison (*case*).

Note

If you choose Case style, and invalid cells are encountered during HDL generation, a warning is issued and generation is attempted using If style. If the truth table contains multiple blocks of cells, any cell blocks which are not valid for Case style may be generated using If style resulting in generated HDL which contains both If and Case constructs.

Refer to “[Case and IF Style Truth Tables](#)” on page 389 for more information.

Clock Signal

When the sequential option is selected, you can enter the clock signal or choose from a dropdown list of available input signals.

You can also set the clock edge sensitivity.

For a Verilog view, you can choose *Rising* or *Falling* corresponding to *posedge* or *negedge* sensitivity.

For a VHDL view, you can choose *Rising*, *Falling*, *Rising Last*, *Falling Last*. These options generate the following VHDL clock edge expressions:

Option	Expression
Rising	<code>clk'EVENT AND clk = '1'</code>
Falling	<code>clk'EVENT AND clk = '0'</code>
Rising Last	<code>clk'EVENT AND clk = '1' AND clk'LAST_VALUE = '0'</code>
Falling Last	<code>clk'EVENT AND clk = '0' AND clk'LAST_VALUE = '1'</code>
Rising edge	<code>rising_edge(clk)</code>
Falling Edge	<code>falling_edge(clk)</code>


Alternatively for either language, you can choose to specify any other valid edge condition.

Reset Signal

When the sequential option is selected, you can enter a list of reset actions and enter a synchronous or asynchronous reset signal (or choose from a dropdown list of available input or internal signals). You can choose to trigger the reset on a *High* or *Low* level signal (or choose *Specify* to enter any other valid reset condition).

If you have specified a Verilog reset condition, you must also specify any additional signals required in the sensitivity list. (Multiple signals should be separated by an *OR* operator.)

You can optionally set pragmas (*sync_set_reset_local* or *async_set_reset_local*) which identify the name of the currently specified synchronous or asynchronous reset signal.

 **Note** The pragmas are entered using the keyword (*pragma*, *synopsys* or *exemplar*) preference set in the **Style** tab of the VHDL or Verilog Options dialog box.

Sensitivity List

You can also specify a sensitivity list of signals which cause the corresponding process to execute if any of the signals change. Multiple signals should be separated by the comma character for VHDL and by an *or* when using Verilog.

If the sensitivity list is not specified it is shown as *<Automatic>* in the dialog box and is automatically created during HDL generation using the following rules:

Sequential with synchronous or no reset	<clock name>
Sequential with asynchronous reset	<clock name>, <reset name>
Combinatorial	All input and internal signals

Full/Parallel Case

When you are using Verilog, you can insert pragmas to specify full case or parallel case statements:

full_case	All possible branches have been specified, any missing branches cannot occur and a default branch need not be generated.
parallel_case	Branches are mutually exclusive.
parallel_case full_case	All possible branches have been specified and are mutually exclusive.

Assignment Type

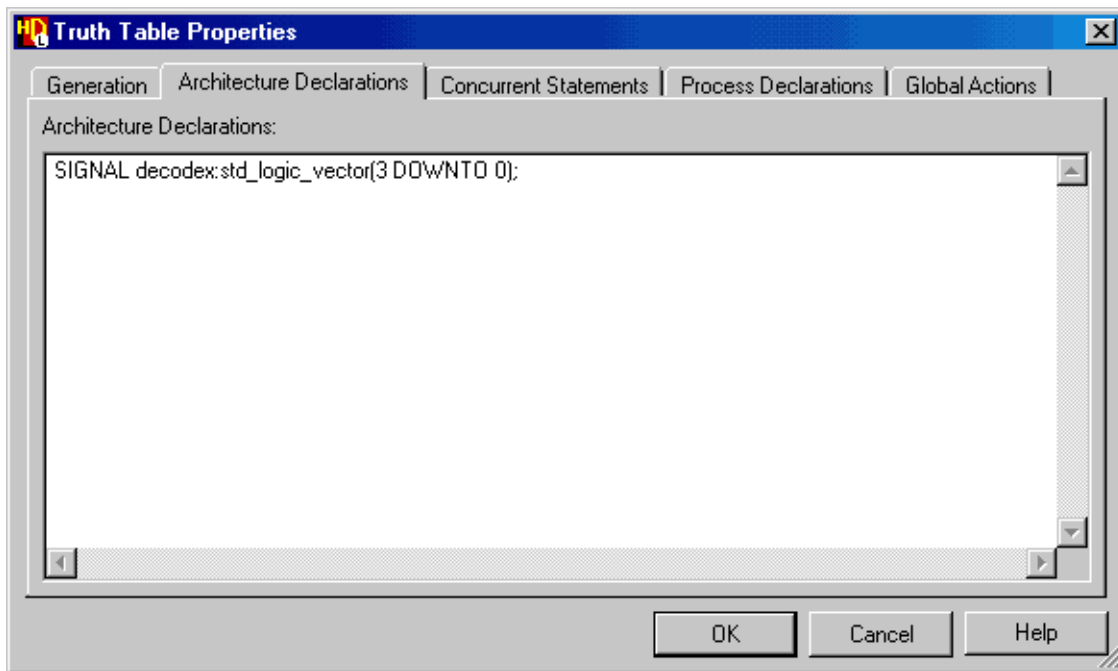
When you are using Verilog, you can also choose whether **Blocking** assignments, specified by the = operator or **Non Blocking** assignments specified by the <= operator are used in the generated HDL.

Editing Architecture or Module Declarations

You can edit *architecture declarations* (when using VHDL) or *module declarations* (when using Verilog) by choosing **Truth Table Properties** from the **Table** menu.

The **Architecture Declarations** (or the **Module Declarations**) tab of the Truth Table Properties dialog box is displayed which allows you to enter any valid HDL statements for the current hardware description language in a free-format entry box.

For example, the following picture shows the Truth Table Properties dialog box being used to enter an architecture declaration:



Terminating semi-colons should be included for all statements entered in the dialog box. No syntax checking is performed when the dialog box is applied to the truth table. However, syntax and semantic checks are performed when you generate HDL for the truth table.

Note



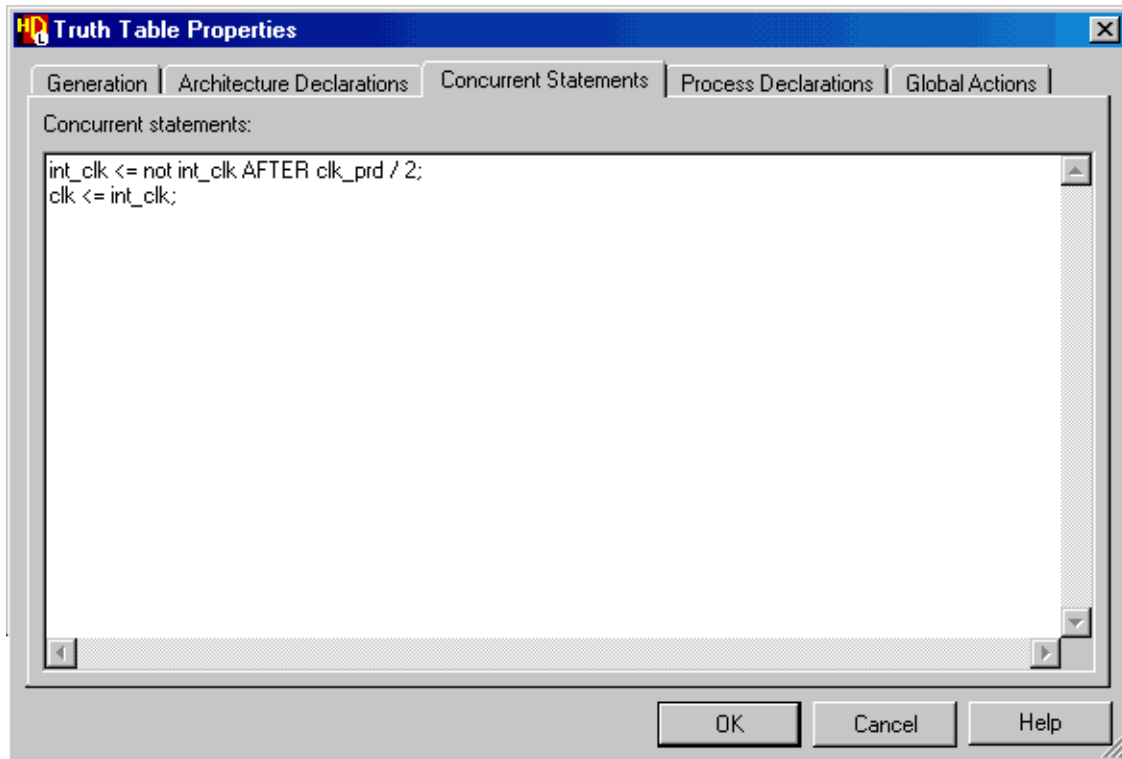
Architecture declarations or module declarations cannot be set when a truth table is used to define an embedded view in a block diagram or IBD view. However, any declarations required by the embedded view can be set on the parent view.

For more information about declarations refer to [“Editing Architecture or Module Declarations”](#) on page 369 in the Flow Chart Editor chapter.

Editing Concurrent Statements

You can edit *concurrent statements* by choosing **Truth Table Properties** from the **Table** menu.

The **Concurrent Statements** tab of the Truth Table Properties dialog box provides a free-format entry box for you to add or edit free-format HDL statements.



Terminating semi-colons should be included for all statements entered in the dialog box. No syntax checking is performed when the dialog box is applied to the truth table. However, syntax and semantic checks are performed when you generate HDL for the truth table.

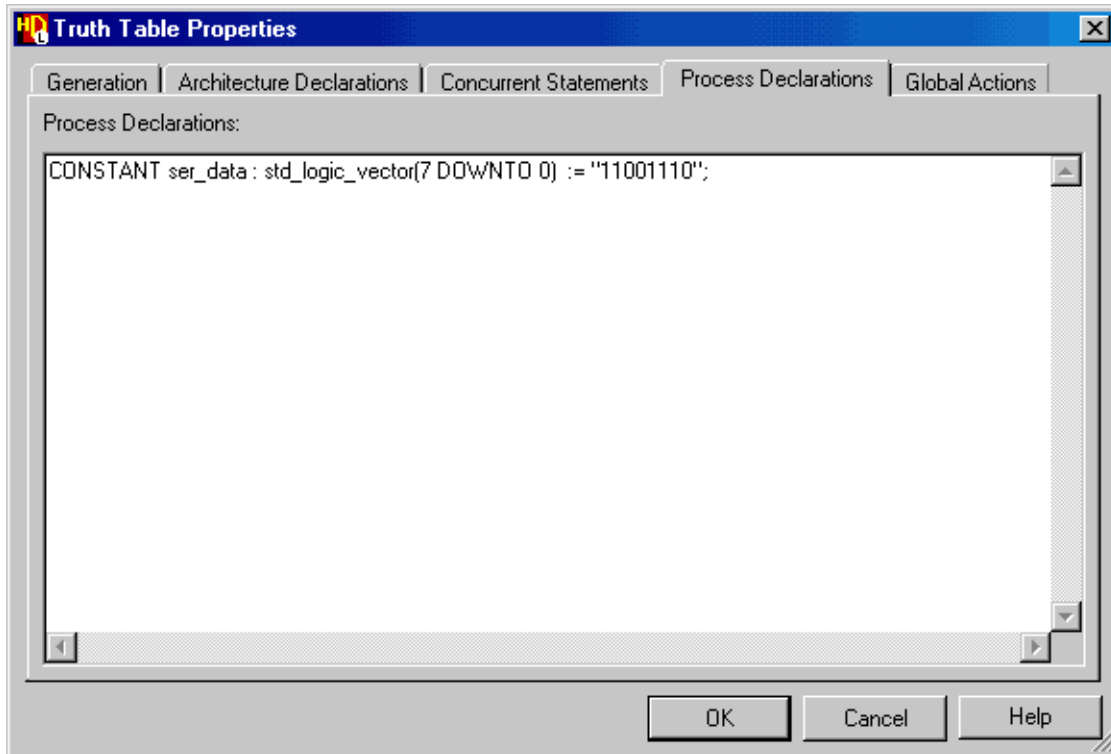
Concurrent statements are included in the generated HDL at the end of the VHDL architecture or Verilog module. They are executed concurrently with all of the processes (or *always* blocks) in the truth table and are typically used for additional datapath operations or specialized clocking.

Editing Process or Local Declarations

You can add or edit truth table *process declarations* (when using VHDL) or *local declarations* (when using Verilog) by choosing **Truth Table Properties** from the **Table** menu.

The **Process Declarations** (or **Local Declarations**) tab of the Truth Table Properties dialog box allows you to add or edit declaration statements within a VHDL process or within a local Verilog *always* block.

These tabs are typically used to declare constants and parameters (or for 'define statements when using Verilog). For example, the following picture shows the Truth Table Properties dialog box being used to enter a constant process declaration:



For a VHDL truth table, process declarations are placed at the beginning of the process immediately before the **BEGIN** keyword in the generated HDL.

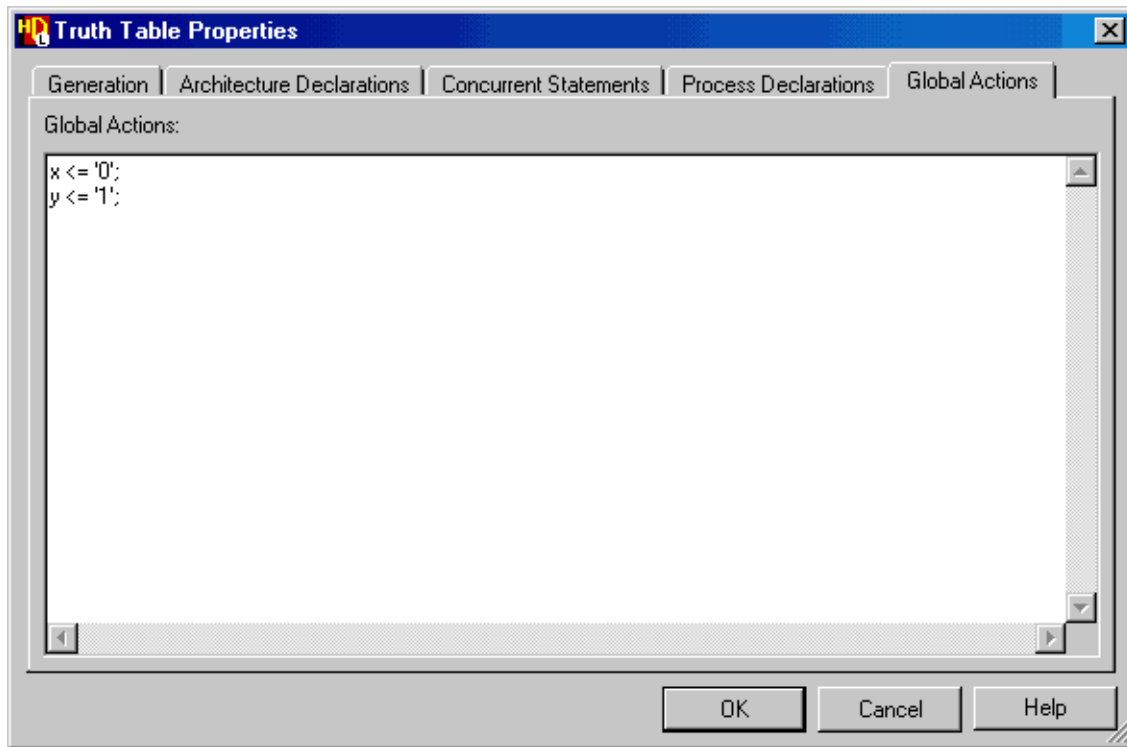
For a Verilog truth table, local declarations are placed after the *always* keyword in the generated HDL.

Terminating semi-colons should be included for all statements entered in the dialog box. No syntax checking is performed when the dialog box is applied to the truth table. However, syntax and semantic checks are performed when you generate HDL for the truth table.

Editing Global Actions

You can add or edit the [global actions](#) for a truth table by choosing **Truth Table Properties** from the **Table** menu to display the Truth Table Properties dialog box.

The **Global Actions** tab of the dialog box allows you to add or edit HDL statements which are included in the generated HDL for the truth table as default actions which are always performed.



Terminating semi-colons should be included for all statements entered in the dialog box.

No syntax checking is performed when the dialog box is applied to the truth table. However, syntax and semantic checks are performed when you generate HDL for the truth table.

Case and IF Style Truth Tables

The HDL generated for a truth table is usually described using *If-Then-Else* constructs but you can optionally set the generation properties to use *Case* style code.

Case statements can often be more efficiently synthesized because the *Case* statement does not imply priority between the available choices.

When you use an *If* statement, priority is imposed by the order in which conditions are specified.

When you are using *If-Then-Else* style code, individual input cells in the truth table matrix can be left empty and are treated as "Don't Care" values by HDL generation.

In a *Case* style truth table, values must be assigned for all input cells in the matrix (although you can use a specific "Don't Care" character if one is supported by your downstream tools).

However, each separate block of cells (which correspond to separate *Case* statements) may include empty columns.

A *Case* style truth table (or block of cells representing a *Case* statement) may contain a single input expression or multiple input expressions.

If the generation properties are set to *Case* style and invalid cells are encountered during HDL generation, a warning is issued and generation is attempted using *If-Then-Else* constructs.

If the truth table contains multiple blocks of cells, any cell blocks which are not valid for *Case* style are generated using *If-Then-Else* style (if valid) resulting in generated HDL which contains both *If* and *Case* constructs.

Note



Case style HDL cannot be generated when additional conditions have been entered in the truth table.

Case Style with a Single Input Expression

For a single input expression, possible values of the input expression are entered in a single column and the corresponding output values entered under each output column.

The resulting *CASE* construct is based purely on the values of this input expression.

An option *Others* (or Default) can be specified using a blank input value with output values specified. If not specified, the default VHDL value is "*null*".

Blank input expression cells are not allowed except for the *Others* row. Any type of input expression is allowed as for *If-Then-Else* style.

Case Style with Multiple Input Expressions

For multiple input expressions, possible values of each input expression are entered under each input column and the corresponding output values entered under each output column.

All scalar input expressions must be of the SAME type and either:

```
std_logic
std_ulogic
```

All array input expressions must be of the SAME type and either:

```
std_ulogic_vector
std_logic_vector
signed
unsigned
```

In VHDL, a case expression must be an enumeration, integer, physical or a one dimension array and locally static. Hence, a local variable is required to contain the concatenation of the input expressions being considered for a given block of cells.

The name of this local variable must have the form:

```
<blockname>_<input expression 1>_<input expression N>_...
```

The resulting name must be a valid VHDL identifier, so care must be taken to modify any characters which may be legal in an expression but not in the concatenated name.

The type of the concatenated variable will be either:

<code>std_ulogic_vector</code>	if all inputs are <i>std_ulogic</i> or <i>std_ulogic_vector</i>
<code>std_logic_vector</code>	if inputs are a mix of <i>std_logic_vector</i> and <i>std_ulogic_vector</i>
<code>std_logic_vector</code>	if all inputs are scalar and of type <i>std_logic</i> or the type of the inputs buses, if there are any.

You cannot mix *std_ulogic*, *std_ulogic_vector* with *signed* or *unsigned*.

Note



If scalar values (in single quotes) are concatenated with other scalar values or with array values (in double quotes) the result is always an array value (in double quotes).

The size of the concatenated variable is the sum of the individual sizes (ranges) of the input expressions being concatenated for that particular block of cells. However, this is not necessarily the width of the ports or input declarations since the input expression can be any expression.

Verilog does allow direct concatenation within the case expression itself, hence a concatenation variable is not required. However, the concatenated values must be prefixed with the total width, for example: *4'b 0101*

Setting Truth Table Preferences

You can set truth table preferences by choosing **Truth Table** from the **Master Preferences** cascade of the **Options** menu in the *design manager*.

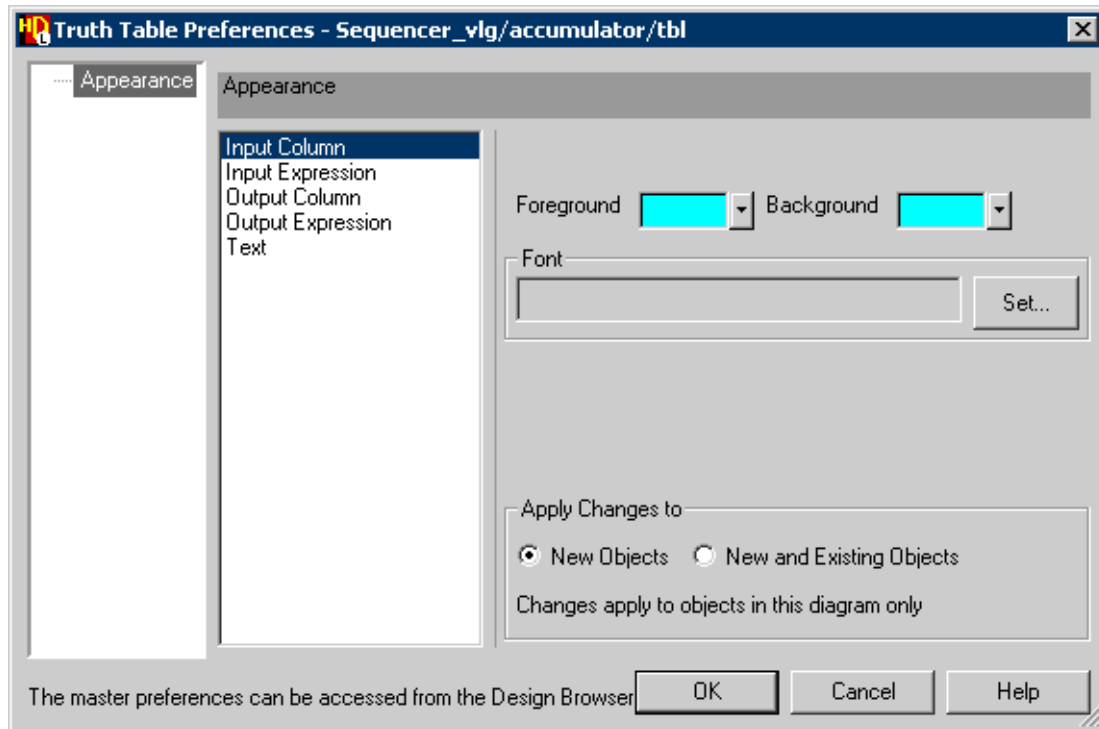
The Truth Table Preferences dialog box has separate pages for setting the default table **General** options, **Default Properties** and **Appearance**.

Note



The general and default preferences take effect on the next truth table you open and can only be edited when the dialog box is displayed from the **Master Preferences** cascade in the design manager **Options** menu. These pages are not available when you choose **Diagram Preferences** in a truth table window.

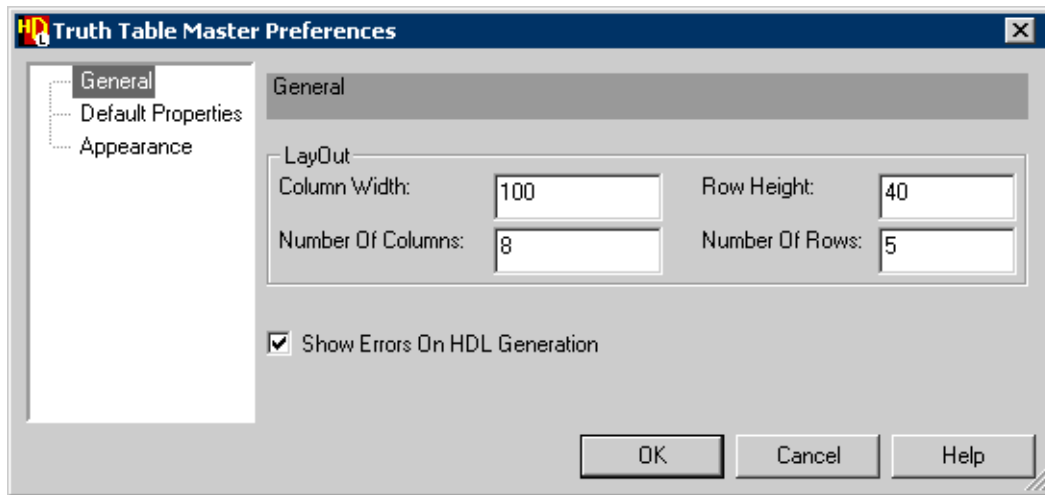
The **Appearance** page allows you to set default visual attributes for individual truth table objects.



The attributes include the foreground and background colors, line color and style, fill style, line width and the text font. Some attributes may not always be available. For example, line style, width and color attributes are not available for a text object. Refer to [“Setting Visual Attributes” on page 83](#) for more information.

When you are setting master preferences, you can also set the highlight color for syntax errors.

The **General** page allows you to set the column width and the row height. You can also set the initial number of columns and rows for a new truth table which has no parent view. There must be at least two columns which are split evenly between input and output columns.



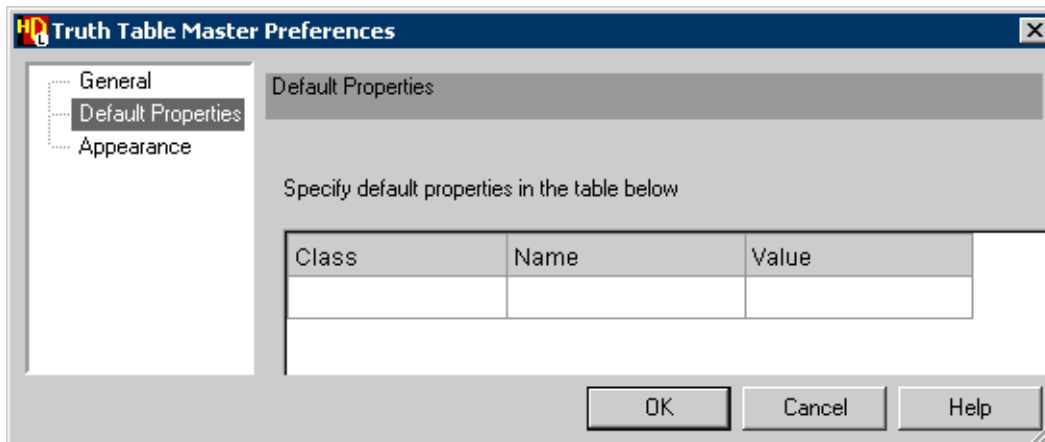
Note



Note that when a truth table is created as a child view from a block diagram, IBD view or symbol, an input column is created for each input signal and an output column for each output, buffer or bidirectional signal.

The HDL syntax in a truth table is not checked on entry. However, you can set a master preference to specify that the syntax is checked during HDL generation. If this option is enabled, cells with syntax errors are highlighted when HDL is generated for the view.

The **Default Properties** page allows you to define default user properties for truth table views.



Refer to the [HDL Designer Series User Manual](#) for information about “Using View Property Variables”.

The appearance and default properties preferences can also be edited by choosing **Diagram Preferences** from the **Options** menu in a truth table.

The diagram preferences dialog box allows you to set the background color for each type of cell in the truth table and the foreground color and font used for cell text.

Note



The foreground color for cells and background color for text is ignored.

When you edit preferences for the active diagram, the dialog box allows you to choose whether the preferences are applied to new objects or to both new and existing objects in the truth table.

Diagram preferences effect the active truth table view only. The master preferences take effect on the next truth table you open but do not effect any truth tables that are already open.

You can save the appearance preferences set on the active diagram as master preferences by choosing **Update from Diagram** in the **Master Preferences** cascade of the **Options** menu or you can apply the master preferences to the active diagram by choosing **Apply to New Cells** or **Apply to New and Existing Cells**.

Refer to the “Default Preferences” appendix in the [HDL Designer Series User Manual](#) for lists of the default preferences set when you invoke a HDL Designer Series tool for the first time.

Chapter 10

Graphical Rendering

This chapter describes how you can convert HDL text views to editable graphical views. You can also choose to visualize any HDL individual view as a graphical view.

Design Extraction.....	395
Recovering Design Structure	396
Recovering State Machines.....	398
Recovering Flow Charts	399
Incremental Recovery	401
Using the Convert to Graphics Wizard	401
Setting Convert to Graphics View Styles	402
Setting Libraries for Black Box Components.....	403
Setting Convert to Graphics Wizard Options	404
Setting Convert to Graphics Options.....	405
Block Diagram Options	406
Routing Options	407
Placement Options	408
Updating a Graphics View from Generated HDL	409
Visualizing HDL Text as Graphical Views	409
Block Diagram Layout and Routing	409
Changing the Layout of a Block Diagram	410
Automatic Routing	410

Design Extraction

The *HDL Designer Series* tools incorporate *HDL2Graphics* technology which can read any *VHDL*, *Verilog* (or mixed VHDL and Verilog) design and convert the structure of the source design code to a more easily maintainable hierarchy of *design unit views* while fully preserving the original behavior.

The HDL Designer Series can read HDL source files which fully comply with the VHDL (IEEE-1076-1987, IEEE-1076-1993 and IEEE-1076-2002) or Verilog (IEEE-1364-1995 and IEEE-1364-2005) and SystemVerilog standards but may fail if non-standard HDL is encountered.

Many tools support *translate_on* and *translate_off* pragmas which can be used to segregate application specific sections of *HDL* code. For example, *-- synopsis translate_off*, *-- exemplar translate_off* or *-- pragma translate_off*.

HDL2Graphics attempts to convert all HDL code including comments and code enclosed between these pragmas. However, you can isolate code you do not want converted by using the special pragmas *-- hds translate_off* and *-- hds translate_on*. All pragmas must include the appropriate comment characters (*--* in VHDL or *//* in Verilog).

Recovering Design Structure

The structure of the source HDL is recovered as a hierarchy of *design unit views* representing the relationship between *design units*.

Top level design units are recovered as *components* and marked with a 🚩 marker. Child design units are recovered as unmarked components.

Hierarchy descriptions in the HDL can be recovered as graphical views defined by a *block diagram* or *IBD view*. Primitive leaf level views defined as *state machines* in the HDL can be recovered as graphical *state diagrams* and other leaf-level code represented as graphical *flow charts*.

All diagram types can be recovered when you are using the *HDL Detective* or *HDL Designer* tools but are not automatically available when you are using *HDL Author*. However, if you attempt to use *HDL2Graphics* technology with HDL Author, a dialog box is displayed which allows you to acquire a temporary floating license for HDL Detective or HDL Designer if one is available. This license is released when the conversion to graphics operation has been completed.

Any concurrent HDL fragments within a hierarchy description are recovered as *embedded blocks*. Leaf-level HDL can optionally also be recovered as *embedded blocks* on a non-hierarchical block diagram. These embedded blocks are defined by *embedded views* which usually contain *HDL text* but if the HDL is recognized as a state machine can optionally be recovered as an embedded state diagram.

Each occurrence of a concurrent VHDL process or a block of Verilog always code is recovered in a separate embedded block connected by the signals used in the HDL fragment. All other concurrent HDL (such as assignments or generate statements) are recovered in a single embedded block which is unconnected to other objects in the recovered view.

Any required Verilog *compiler directives* or *VHDL package* references are automatically set on the graphical views.

If a design unit interface uses a *VHDL generic* or *Verilog parameter* declaration which requires port mapping it is instantiated in the parent view using a *port map frame*. Generate or Block keywords and other conditional HDL structures are recovered using a *generate frame*.

If a *VHDL entity* has multiple *VHDL architectures*, then all architectures are recovered as design unit views and the alphabetically last architecture name is set as the default view of the recovered design unit.

Any configuration information included in the HDL source (either as embedded declarations or as separate configuration specifications) is used to automatically recover multiple libraries and identify the instantiation of different views for a design unit. However, multiple libraries or multiple views cannot be recovered for Verilog designs.

A *VHDL configuration* defines the HDL library for a given entity and the architecture which is bound to a given instance within a structural architecture. For example, the following configuration statement specifies that instance *i0* of the *counter* component is bound to architecture *fsm* of entity *counter*, in library *timer_vhdl*:

```
for i0: counter
use entity timer_vhdl.counter(fsm);
end for;
```

The configuration may also specify additional port mapping and VHDL generics for an instance to those defined by the component instantiation. When multiple configurations exist for the same entity, only one configuration will be recovered. If a configuration is used in the source HDL to bind a component instantiation to alternative entities, this information is not recovered.

If a Verilog source file references an instance for which no corresponding Verilog module is provided, the port mapping does not indicate the mode or type of the ports for the instance. HDL import will guess the missing port information and recover a black box instance. This allows the recovered view to show the connectivity between the undefined instances.

The layout and routing for recovered block diagrams is determined by the default Document and Visualization options. However, you can change these options and run the diagram layout or autorouting commands interactively on any block diagram. Once changes have been made, the new options are saved as preferences and used the next time that you run a conversion to graphics operation.

The Verilog source code may contain many *Verilog modules* in the same file.

Compiler directives before the first *module* keyword are recovered as Pre-Module directives.

Compiler directives after a *module* keyword but before the first lexical token other than a comment or white space are recovered as Post-Module directives for the Verilog module.

Compiler directives after the last *endmodule* keyword in the file are recovered as End-Module directives.

Compiler directives between two Verilog modules are recovered as End-Module directives for the previous module if they occur before a *`nounconnected_drive* or *`endcelldefine* directive, otherwise they are recovered as Pre-Module directives for the next Verilog module.

Recovering Verilog Parameters

If a Verilog parameter is used in the declaration of an interface signal, it is recovered in the external declarations at the top of the Verilog module.

Other Verilog parameters which occur before syntactic text in the source Verilog are declared in the internal declarations for the Verilog module except parameters which are used for enumerated state encoding in a state machine.

Any other Verilog parameters are declared in the view declarations.

Any parameter on an instance with an overridden value is included in the Verilog parameter mapping shown on the instantiated block or component.

Recovering State Machines

When state machine recovery is enabled, a state diagram is created for each behavioral (leaf-level) HDL description below an instance in the recovered structure that is recognized as an explicit state machine and can be described by a state diagram.

Note



Implicit algorithmic state machines are not recovered. However, an algorithmic state machine can be recovered as a flow chart with clock conditions specified at the start of each action box representing a state.

An explicit state machine is implied when a block of HDL code contains:

- A *state variable*, which specifies the current *state* of the machine.
- A clock and optional synchronous or asynchronous reset condition (for synchronous machines only).
- Specification of state *transitions*.
- Specification of outputs.

A separate concurrent state machine is created for each state machine defined in the same VHDL architecture or Verilog module.

The recovered states are placed in a clockwise direction around a circle with the start state at the top. However, when there are only two states, they are placed horizontally with the start state on the left. The placement sequence is determined by the connections between states (not their order of occurrence in the HDL code).

Recognizing State Machines

When you enable the State Diagram option in the Convert to Graphics View Style Options, the HDL parser searches for HDL code with characteristics which represent an explicit state machine.

The following algorithm is used to analyze the HDL code:

- Find any clocked processes (which must have a sensitivity list).
The first body statement must be *IF* (but can be preceded by declarations) which must contain a condition. For example: *IF <condition> THEN*.
If there is no *ELSE* or *ELSIF*, then assume that the *IF* is testing the clock condition and extract this clock condition.
If there are *ELSE* or *ELSIF* branches, then assume the first *IF* contains the reset condition and check the *ELSE* or *ELSIF* to get the clock condition.

The reset condition must have the form *<condition> THEN <actions>* and must include a reset signal in the condition. For example: *rst='0'* or *rst/= '0'*. There must also be an assignment (signal or variable) to a state variable.

The clock condition must have the form *<condition> THEN <actions>* and must include a clock signal in the condition. For example: *clk'event AND ... or rising_edge(clk)*. If the first statement is an assignment, it must assign a signal or variable to a state variable. If the first statement is *CASE*, then it must test the state variable.
- Find transition (next state) processes for each clocked process, (clocked and transition processes may be combined) and find a process which assigns to the next state.
- Find output processes for each clocked process and the output process which assigns to a non state variable output signal.
- Build the state machine structure from the process information.
Store the first clock that was identified.
Store the first reset that was identified.
Store the state variable info for this process.
Store the state encoding, if it is not already stored.
Store target of WHEN OTHERS transition as recovery state
Ensure the reset strings are valid characters.
All assigned signals in the clocked process are tagged as registered.

Recovering Flow Charts

When *flow chart* recovery is enabled in the Convert to Graphics View Style Options, a flow chart is created for each behavioral (leaf-level) HDL description below an instance in the recovered structure. A separate concurrent flow chart is created for each VHDL process or Verilog procedure (where a procedure is any section of code represented by Verilog *always* or *initial* statements).

Note



After recovery, you can choose whether the Verilog regenerated from the graphical flow chart uses *initial* or *always* statements.

For VHDL, concurrent flow charts are named after the process, or if the process is unnamed, a name is generated using the format: *process*<*n*> where *n* is an integer starting from 1. Verilog procedures are not normally named in the source HDL and each concurrent flow chart is given a name using the format: *procedure*<*n*>.

Architecture declarations and process declarations (in VHDL) or module declarations and local declarations (in Verilog) are included in the corresponding flow chart as text blocks. Any concurrent statements in the recovered code relate to all concurrent diagrams for the same block instance and are included as text blocks on all the concurrent flow charts. A text block showing the signals *sensitivity list* for each concurrent flow chart is also created.

The objects in each concurrent flowchart are laid out from top to bottom with all *flows* (except for loopback flows) pointing downwards.

Conditional (*IF*) statements) are recovered as *decision boxes* with a true and false branch. If the conditional statement includes *ELSIF* (or Verilog *else if*) constructs these are represented by multiple cascaded decision boxes.

The choices (values) of a CASE statement are recovered as ports arranged on the bottom edge of a start *case box* from left to right as encountered in the code with the port values placed in the standard position relative the port. (For Verilog, *casex* and *casez* statements are recovered as start case objects with appropriate object attributes.)

Loops are represented with the continue and break statements shown to the right of a start *loop* and end loop pair. The loopback flow is connected between the left sides of these objects.

Wait statements are recovered as *wait boxes* with the appropriate delay statement. In Verilog, timing controls using the *wait* keyword are recovered as wait boxes but the # or @ timing controls are included in action box text.

Any other statements are treated as action statements and recovered into an *action box*. The action box contains as many statements as possible. For example, a section of code containing 100 lines none of which is a decision, case, loop or wait statement, is recovered into a single action box. The action box is automatically sized to contain the HDL text (with default padding added).

Comments within action statements or declarations are included on the flow chart but comments associated with if, case or loop statements are not recovered. In particular, the HDL translation pragmas *synopsys parallel_case* and *synopsys full_case* are not recovered.

Incremental Recovery

Convert to Graphics will normally attempt to recover the entire design described by the source HDL. However, the source HDL may be incomplete or may reference design units which have been previously recovered and already exist.

If an incomplete design is recovered, dummy design units are created in the database for undefined instances in the source HDL.

If the source HDL has been updated and you choose to reconvert the design, any new views are added (and dummy views overwritten). When a design unit is updated by incremental recovery, its interface to child or parent views in the database may also be changed.

In general when a design unit already exists, the following rules are followed:

Existing Unit	New Unit	Rules Applied
component	component	Add or overwrite views. If interface has changed, overwrite symbol.
component	block	Add or overwrite views. If interface has changed, overwrite symbol.
component	component	Add or overwrite views, convert block to component, update parent view.
block	block	If parent design units are unchanged, add or overwrite views. If parent design units changed, convert block to component and update parent view.
VHDL package	package	Add or overwrite package.


Note




An interface is considered to have changed if a port, VHDL generic or Verilog parameter has changed.

If the **Overwrite** option is set, you are not prompted before a view is overwritten. If not set, there is a single confirmation prompt for all views which will be overwritten. If a symbol is overwritten, you are warned that instances of the component will need to be reconciled.

Using the Convert to Graphics Wizard

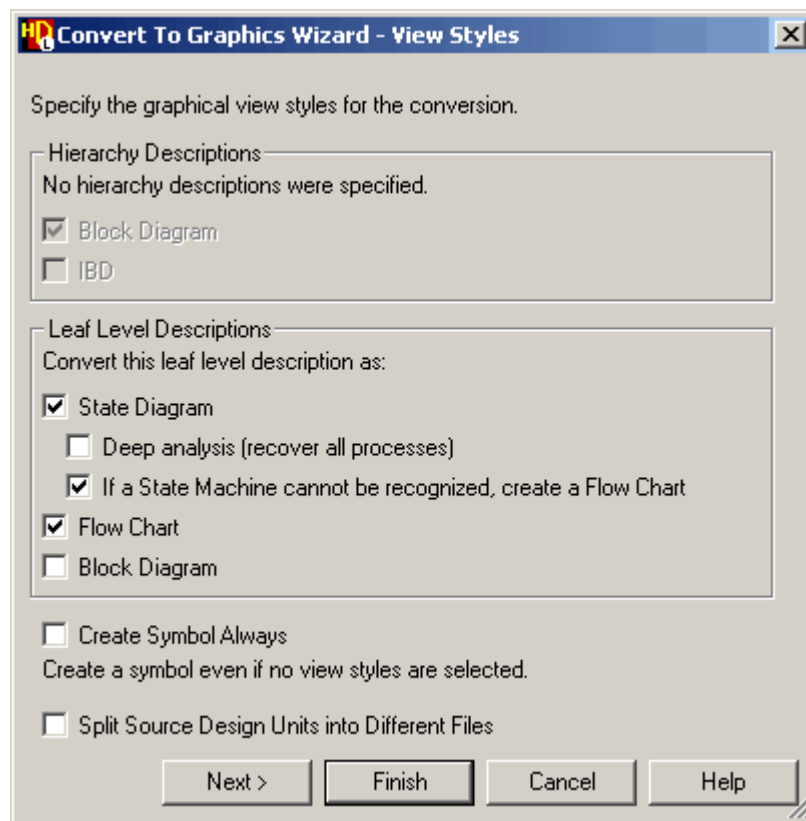
You can convert any existing *HDL text* views to editable graphical source views by selecting the views (or design units if the default views are HDL text views) in the *design explorer* and using the  button or by choosing **Single Level** from the **Convert To Graphics** cascade of the **HDL** or popup menu.

If the HDL text is a structural view which contains hierarchy descriptions, you can use the  button or choose **Hierarchy Through Components** from the HDL or popup menu to convert all views in the hierarchy below the selected view.

The Convert To Graphics wizard is displayed indicating the number of hierarchy descriptions and leaf level descriptions in the selected HDL files. For example, the hierarchy of the *uart_tb* test bench in the example *UART_TXT* library contains four hierarchy descriptions and six leaf level descriptions.

Setting Convert to Graphics View Styles

The first page of the Convert to Graphics wizard allows you to set the required graphical view styles:



Hierarchy descriptions can be recovered as block diagram or IBD views.

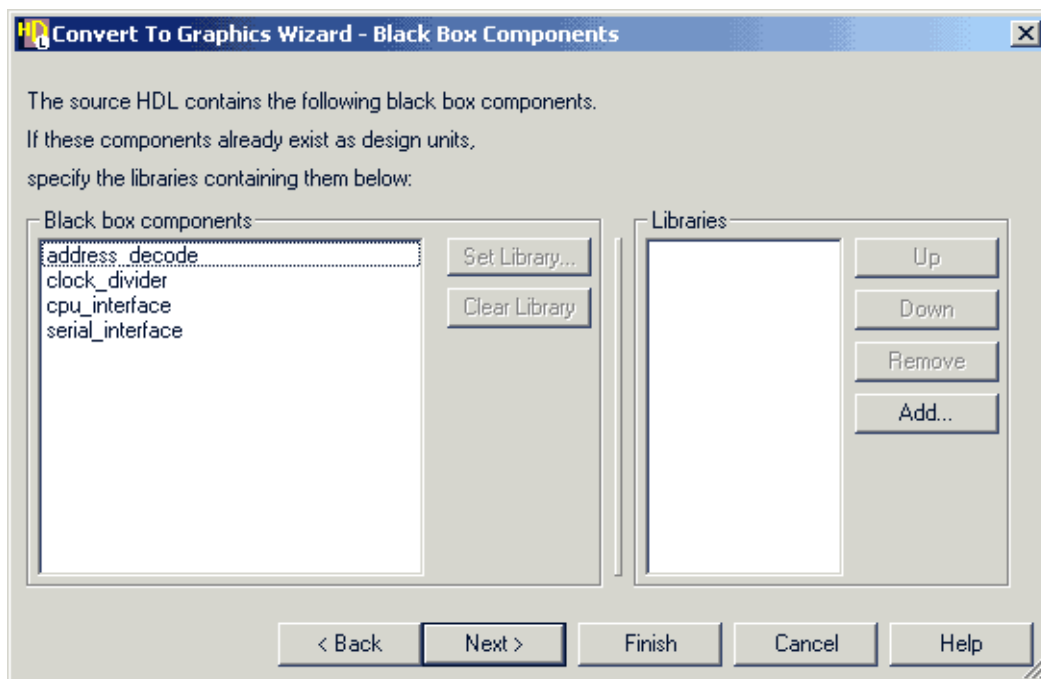
If a finite state machine is recognized in a leaf-level description it can be recovered as a state diagram. You can optionally choose the “Deep analysis” option whereby all process statements are analyzed to recover any relevant state machine information. Otherwise, the leaf level description can be saved as a flow chart or as a block diagram with the leaf level HDL codes represented by embedded HDL text views on the diagram.

You can also choose to create a symbol even when no other graphical view styles are selected. If this option is set, symbols are created although all views remain as HDL text. If unset, symbols are only created for the design units converted to graphic views.

Setting Libraries for Black Box Components

If you use the **Next** and the HDL code includes instantiations for components which are not defined in the current library (or in a library specified by a VHDL configuration) the Black Box Components page is displayed listing these *black box instances*.

For example, the following page is displayed when converting a HDL view which contains undefined black box components for the design units *address_decoder*, *clock_divider*, *cpu_interface* and *serial_interface*:

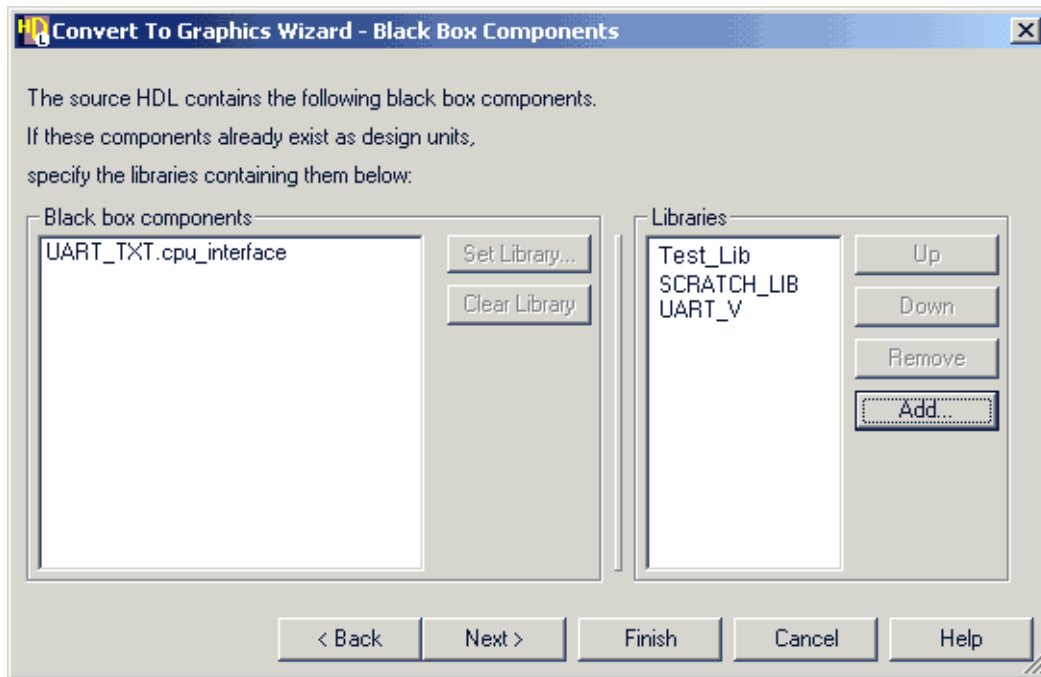


You can use the **Set Library** button to set a specific library by choosing from a list of existing libraries in the active project or the **Clear Library** button to clear a specific library specification.

Alternatively, you can use the **Add** button to create a list of libraries to search for matching components. The list can be ordered by using the **Up** and **Down** buttons or libraries removed from the list using the **Remove** button.

Components are automatically removed from the list of black box components if they exist in one of the libraries in the search list.

The following example shows the *cpu_interface* component has been explicitly assigned to the *UART_TXT* library while the other components are located by searching the *Test_Lib*, *SCRATCH_LIB* and *UART_TXT* libraries:



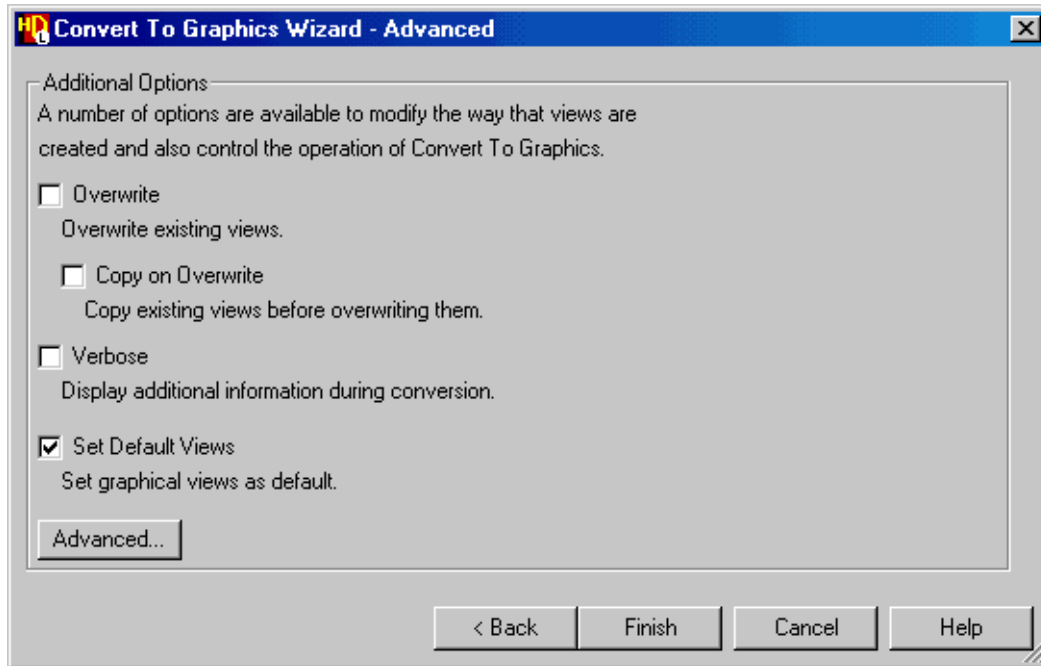
If you confirm the wizard while there are unassigned black box components, you are prompted to return to the wizard and specify the remaining libraries. If you override this prompt, the components are instantiated as black box components in the graphical view.

Setting Convert to Graphics Wizard Options

The last page of the wizard allows you to set additional options to overwrite existing graphical views and display verbose messages in the log window during conversion.

If you choose to overwrite an existing view, you can choose to make a copy of the previous view. (For example, if you overwrite a block diagram view named *struct*, the previous view is saved as *Copy_of_struct*.)

You can also choose to set the new graphical views as the *default views* or use the **Advanced** button to display the Documentation and Visualization Options dialog box.



When you confirm the wizard, the selected HDL files are read and the graphics views created.

Progress messages (including any errors if problems are encountered) are written to the Task Log window.

When conversion is complete a completion message is issued with a report of the number of design units saved and the number of components, block diagrams, state machines and flow charts that have been created.

Note

Note that a file name clash with the original source HDL file may occur if you generate HDL from a graphical view. You should rename one of these views if you want to keep both the graphical and HDL text views.

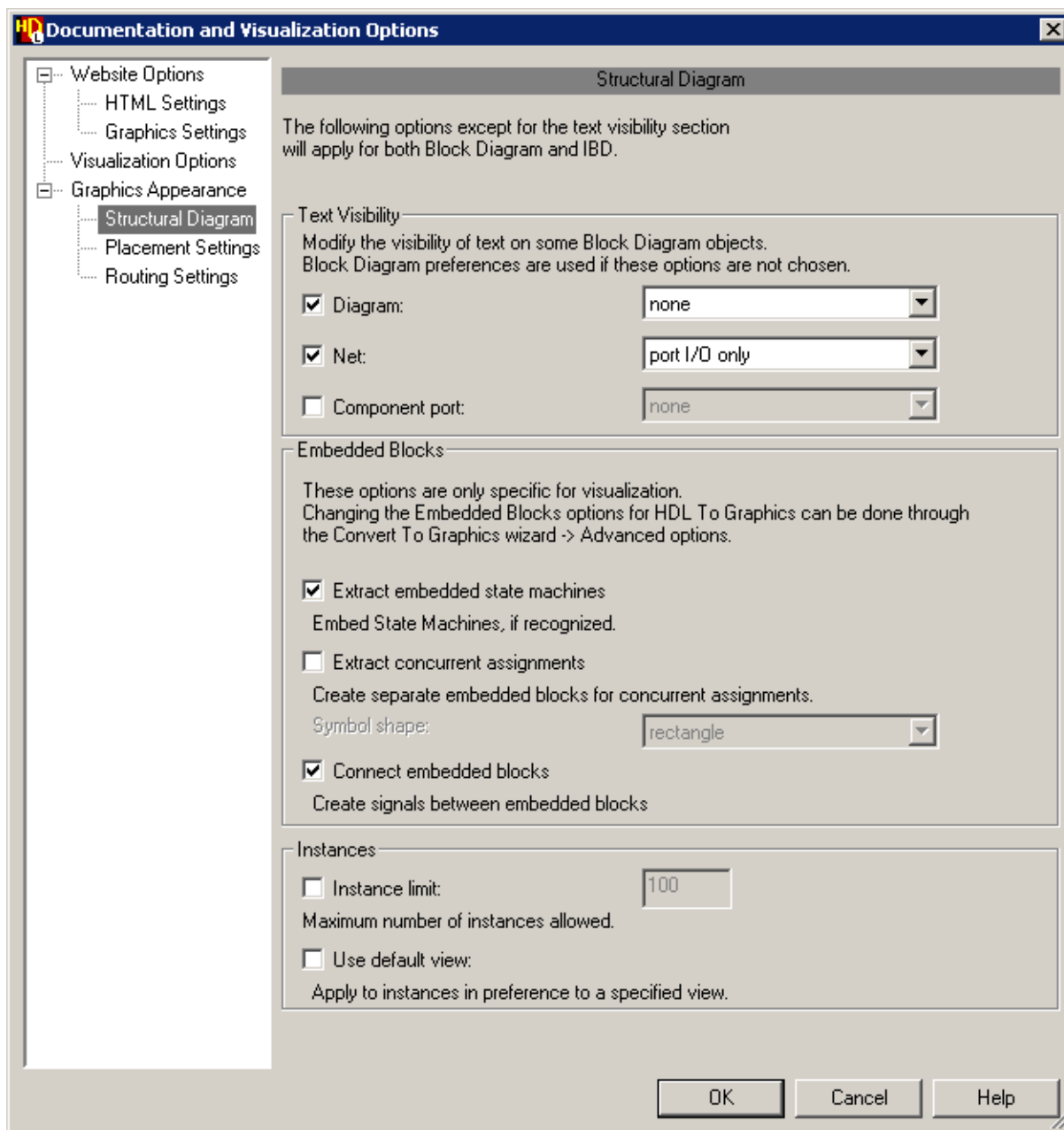
Setting Convert to Graphics Options

Convert to Graphics options can be set by using the **Advanced** button in the Convert to Graphics wizard or by choosing **Documentation and Visualization** from the **Options** menu in the *design manager* to display the Documentation and Visualization Options dialog box.

The options are saved and used as defaults the next time you convert or render HDL code as graphics.

Block Diagram Options

The **Structural Diagram** Node allows you to set options for text visibility, embedded blocks and instances on the recovered diagrams.

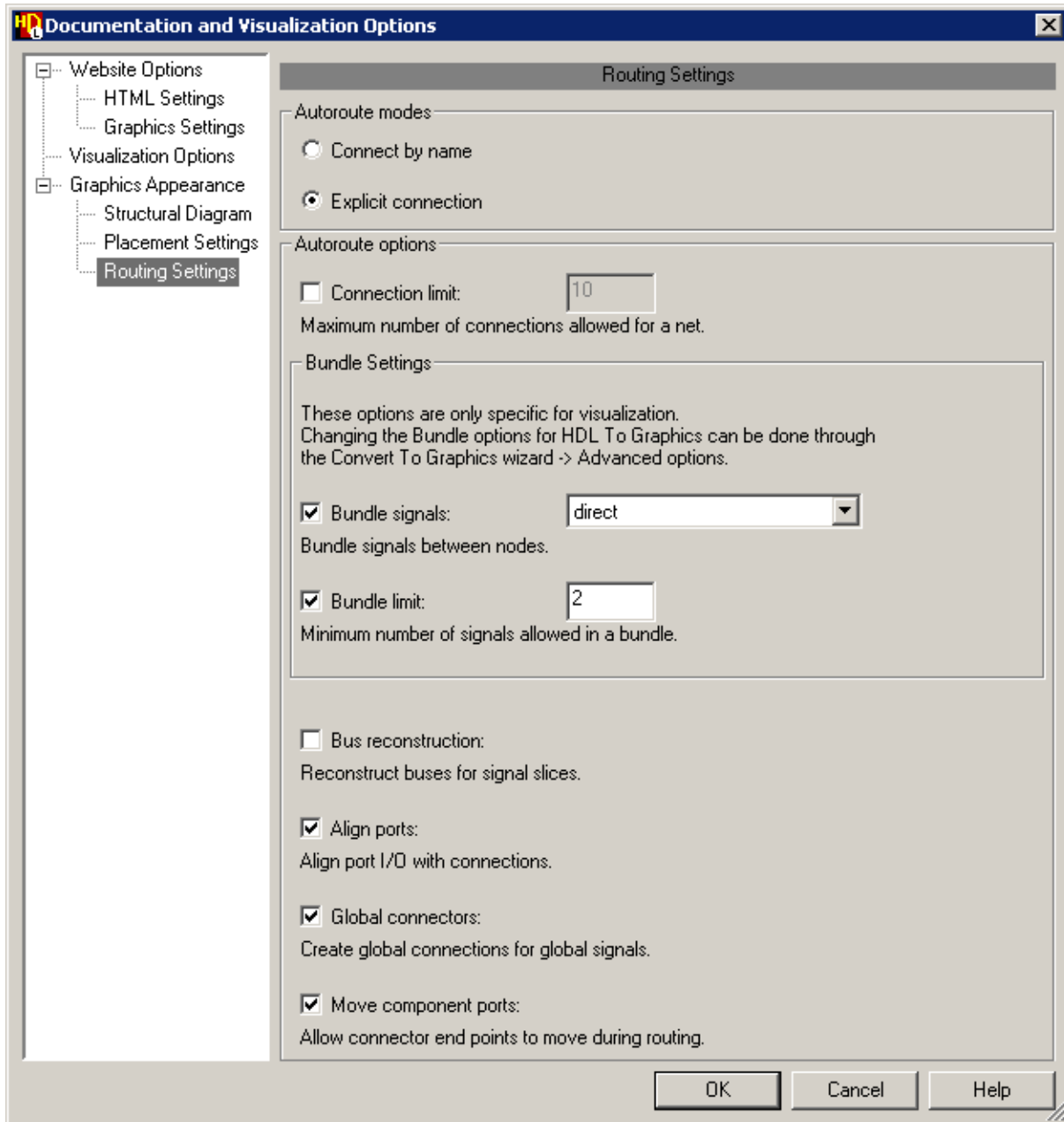


Note

Hiding text helps to optimize automatic routing and placement for compact layout. If the text objects are hidden, instances can be placed closer together to minimize the size of the recovered diagrams.

Routing Options

The **Routing** node allows you to set options which control whether nets are connected on a recovered block and how these nets are routed.



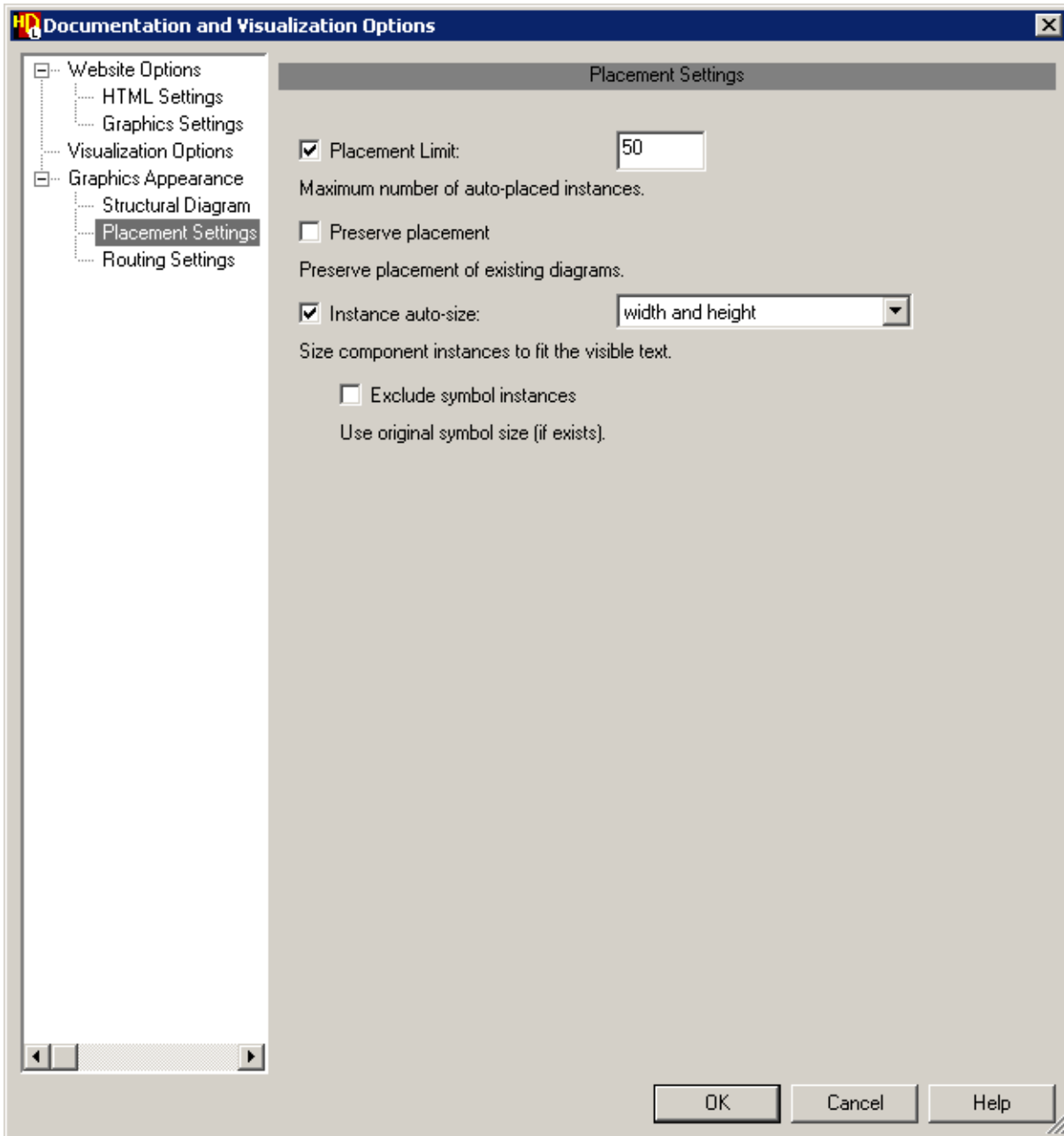
Note



The routing options are also available from the Block Diagram Layout and Routing Options dialog box.

Placement Options

The **Placement** node allows you to control the placement of instances on the recovered block diagrams.



Note



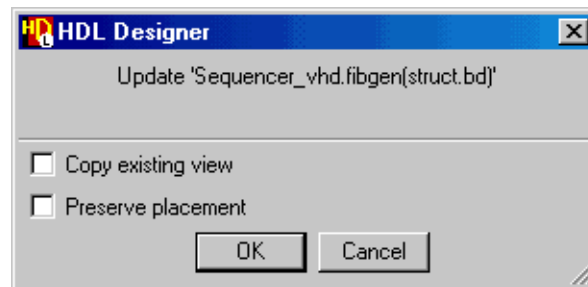
The placement options are also available from the Block Diagram Layout and Routing Options dialog box.

Updating a Graphics View from Generated HDL


If a graphical view is older than its generated HDL, you can select the generated file in the *design explorer* HDL Files view and choose **Update Graphics** from the popup menu to run Convert to Graphics and overwrite the graphical view.





You are prompted whether to retain a copy of the existing view (as *Copy_of<view>*) and whether to preserve the placement of objects from the existing view.

For example:



Visualizing HDL Text as Graphical Views

You can visualize a HDL text view as a graphical view by selecting the view (or a design unit if the default view of the design unit is a HDL text view) in the *design explorer* and using the  button.

A pulldown palette on the button allows you to choose a block diagram , IBD view , state diagram  or flow chart  view.

Alternatively you can choose **Block Diagram**, **IBD**, **State Diagram** or **Flow Chart** from the **Visualize Code As** cascade of the **HDL** or popup menu.

Although you can only perform non-logical edits to visualized views, the simulation, animation and cross-probing facilities described in “[Simulation and Animation](#)” on page 415 are fully functional.

Block Diagram Layout and Routing

Layout and routing is performed automatically when a block diagram is created from a HDL text view using the current layout and routing options or when you show a block diagram from an IBD view with the **Update Layout** option.

Block diagram layout and routing options can be set using a tabbed dialog box which is displayed by choosing **Layout/Routing Options** from the **Diagram** menu in a block diagram to display the Block Diagram Layout/Routing Options dialog box.



Tip: The **Routing** tab can also be accessed by choosing **Autoroute Options** from the **Autoroute** cascade of the **Diagram** or popup menu.

The dialog box provides access to the same options as provided in the Documentation and Visualization Options dialog box shown in “[Placement Options](#)” on page 408 and “[Routing Options](#)” on page 407.

Refer to the descriptions of the Layout Options and Routing Options which can be accessed using the **Help** buttons on each tab of this dialog box. These options are saved as preferences and used as defaults the next time you use the layout and routing commands.

Changing the Layout of a Block Diagram

You can automatically update the placement of blocks and components on a block diagram by choosing **Layout Diagram** from the **Diagram** menu or by using the **Ctrl + L** shortcut.

The placement algorithm attempts to rearrange the diagram, reducing empty space while preserving a left-to-right design flow. This command can be used in conjunction with the **Autoroute** command to optimize the diagram readability.

The placement algorithm is designed to reduce the number of feedback loops and reduce the complexity of the signal paths while showing association by placing ports on blocks as close as possible to their source.

The placement algorithm and block diagram layout options are used automatically to arrange diagrams recovered from HDL text views. However, the layout for a diagram may often be improved by using the **Layout Diagram** command after changing layout options.

Note



Note that any Non-autoroute panels on the diagram are deleted but Regular or Sheet panels are preserved.

Automatic Routing

You can re-route the nets on a block diagram by using the **Autoroute** cascade of the **Diagram** or popup menu and choosing **Autoroute**, **Autoconnect**, **Autobundle** or **Connect By Name**. Note that the **Autoroute** command can also be executed by using the **Ctrl + R** shortcut.

If nothing is selected, all connections on the diagram are re-routed. However if objects are selected, the selected nets and all nets connected to other selected objects are re-routed.

You can exclude parts of a diagram from the automatic routing commands by adding a panel around an area and setting the **Non-autoroute** panel property as described in “[Adding a Panel](#)” on page 79.

The percentage complete is indicated in the status bar while autorouting a large design.

The routing algorithm and current block diagram routing options are used automatically to route diagrams recovered from HDL text views. However, the routing for a diagram may be improved by using the **Autoroute** command after changing the routing options. For example, a complex diagram can often be simplified by enabling the options to automatically group signal nets into bundles or reconstruct a bus from bus slices.

Autoroute

The **Autoroute** command uses the current options set in the **Routing** tab of the Block Diagram Layout/Routing dialog box.

Autoconnect

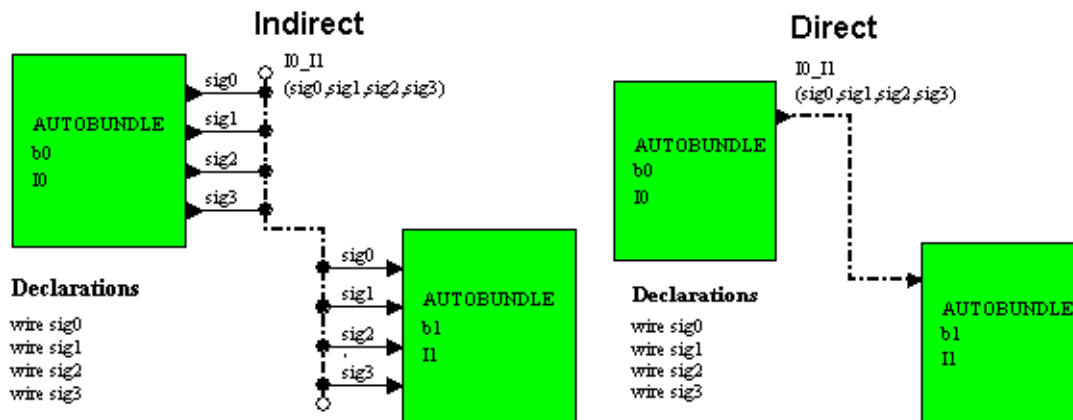
The **Autoconnect** command attempts to re-route the nets using simple routes which comprise the minimum number of vertical and horizontal segments to connect the source and destination but avoid other objects (or unselected nets).

If no route is found, or the source or destination is on another net, a direct line is drawn with no route points between the origin and destination. This command is equivalent to using **Autoroute** with **Explicit connection** set (but **Keep existing connectivity**, **Bundle signals** and **Connection limit** unset) in the routing options.

Autobundle

The **Autobundle** command automatically groups signal nets into bundles. This command is equivalent to using **Autoroute** with **Explicit connection** set (but **Keep existing connectivity** unset) and **Bundle signals** set. (The **Bundle limit** is set to the current number specified in the routing options.)

When you set the **Bundle signals** option, you can choose whether to use direct or indirect connection. For example, the following illustration shows the same two blocks connected using indirect and direct bundles:



When direct bundling is set, bundles are automatically created for any group of signals (which may include buses) between the same nodes.

For indirect bundling, the individual signals are shown as stubs with a common bundle connecting them across the diagram.

You can set a bundle limit to prevent buses being created with less than a specified number of signals. The most compact diagram layout is achieved when this limit is set to zero.

When the signals are connected to a component and direct bundling is set, the bundle is connected to the ports on the component using a port map frame. If two bundles have same contents, they are not connected but connections between the contained signals is implied by name.

Automatically created bundles are given a name derived from the source and destination nodes. For example, the bus **I0_I1** connects between instances **I0** and **I1** in the illustrations above.

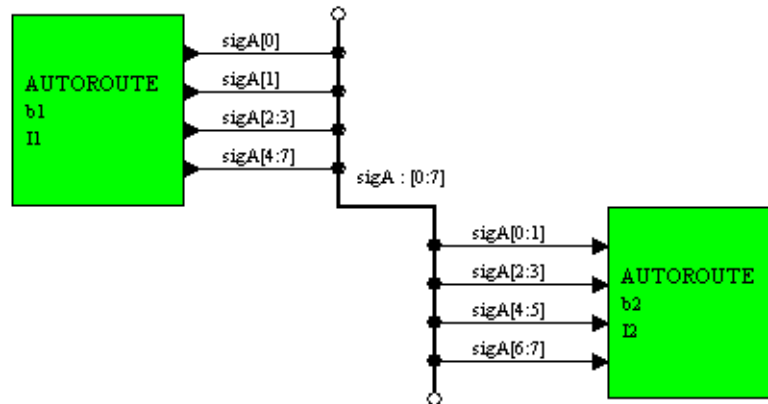
Connect by Name

The **Connect By Name** command reduces all nets to stub signals with all the connections implied by name. This command is equivalent to using **Autoroute** with **Connect by name** set in the routing options.

Bus Reconstruction

You can enable a block diagram layout option to reconstruct buses from slices of a signal with the same name. When set, automatic bus reconstruction is performed when you render a new block diagram or when you use the **Autoroute** command in an existing block diagram.

When you set the **Bus reconstruction** option, you can choose whether to use direct or indirect connection. However, reconstructed buses connected to signal stubs on a block are always connected indirectly since direct connections to a block would modify the block interface. The following example shows slices $sigA(0)$, $sigA(1)$, $sigA(2:3)$ and $sigA(4:7)$ combined into a single 8-bit bus $sigA$ then deconstructed into slices $sigA(0:1)$, $sigA(2:3)$, $sigA(4:5)$ and $sigA(6:7)$.



Chapter 11

Simulation and Animation

This chapter describes procedures for driving simulation from a graphical diagram and animating the state diagrams and flow charts in your design.

Simulator Cross-Probing	415
Simulation Toolbar	416
Adding Signals to Simulator Windows	418
Removing Signals from Simulator Windows	418
Adding Signals to the Simulator Log	419
Highlighting Signals in the Simulator	419
Reporting Signal Information	419
Adding and Removing Breakpoints	419
Enabling and Disabling Breakpoints	420
Reporting Breakpoint Status	420
Adding and Removing Simulation Probes	421
Choosing the Simulation Instance	424
Setting the Simulator Environment	425
Running the Simulator	426
Displaying Simulator Windows	427
Restarting the Simulator	427
Using the ModelSim Source Window	427
Cross-Probing from ModelSim	428
State Diagram and Flow Chart Animation	431
Animation Toolbar	432
Enabling Data Capture	433
Setting the Activity Trail	434
Graphical Highlighting	435
Reviewing Animation	436
Linking Diagrams for Animation	437
Mixed Language Animation	437

Simulator Cross-Probing

When you are using a supported simulator, an additional **Simulate** menu and toolbar are available in the block diagram, flow chart and state diagram windows which allow direct cross-probing between the simulator and the source design objects.

The features described in this chapter have been developed primarily for use with the ModelSim EE, SE or PE (version 5.5 or later) simulator. However, many of the features are also available

when you have invoked the NC-Sim (version 3.0 or later) simulator. Simulator cross-probing is not available when you are using VCS or VCSi.

















Refer to “Tasks, Tools and Flows” in the *HDL Designer Series User Manual* for information about setting up the simulator and other downstream tools.

The simulator should normally be invoked on a design unit at or above the level to be simulated. However, you can change the simulator environment to a lower level design unit and it is possible to synchronize with ModelSim when the simulator has been invoked on a root design which includes lower level design units.




Simulation Toolbar

The Simulation toolbar which is automatically displayed in the graphical state diagram and flow chart views when a supported simulator has been invoked and supports the following commands

Table 11-1. Simulation Toolbar Commands in State Diagram and Flow Chart Views












Icon	Description
	Run for a specified time
	Run forever
	Continue
	Run until next event
	Step over the current HDL line
	Step into procedure calls in the current HDL line
	Step over HDL in the selected object (state diagram only)
	Add a breakpoint to the selected object
	Remove a breakpoint from the selected object
	Remove all breakpoints
	Disable breakpoints
	Disable all breakpoints
	Enable breakpoints
	Enable all breakpoints
	Highlight signal in simulator corresponding to selected object
	Restarts the simulator

Note

The  button allows you to change the default timestep used by the  button by choosing from a pulldown list. The  button is available in the state diagram only.






The following additional commands are available from the Simulation toolbar which is displayed in the graphical block diagram.

Table 11-2. Simulation Toolbar Commands in Block Diagram Views


Icon	Description
	Add the selected signals to the simulator Wave window
	Remove the selected signals from the simulator Wave window
	Add the selected signals to the simulator List window
	Remove the selected signals from the simulator List window
	Reports signal information to the main ModelSim window
	Add a probe to the selected signal
	Delete a probe from the selected signal
	Delete all probes
	Enable or disable cursor tracking for probes
	Set probe properties
	Choose instance for simulation

The following additional commands are available from the Simulation toolbar which is displayed in the graphical state diagram:

Table 11-3. Simulation Toolbar Additional Commands in State Diagram

Icon	Description
	Add the current state to the simulator Wave window
	Remove the current state from the simulator Wave window
	Add the current state to the simulator List window
	Remove the current state from the simulator List window
	Enable or disable cursor tracking for state machines

Note


The  button is permanently dimmed in the Simulation toolbars if you are using ModelSim PE which does not support this integration feature.

The toolbar is normally displayed at the bottom of the diagram window but can be undocked, moved, docked or hidden in the same way as the other toolbars.

The toolbar can be displayed or hidden by setting the **Simulation Tools** option in the **Toolbars** cascade of the **View** menu.

Refer to “[Toolbars](#)” on page 20 for more information about toolbars.

Adding Signals to Simulator Windows

You can add signals to the simulator Wave window by selecting them on the block diagram using the  button from the Simulation toolbar or by choosing **Add Wave** from the **Display** cascade of the **Simulate** menu.



You can add selected signals to the simulator List window by using the  button from the Simulation toolbar or by choosing **Add List** from the **Display** cascade of the **Simulate** menu.

Note






If a block or component is selected, all the connected signals are added to the simulator window.


The Wave window is automatically displayed if it is not already open.


If you are using *ModelSim*, the  and  buttons (or commands in the **Simulate** menu) can be used in a state diagram to add a signal defining the current state of the state machine to the Wave or List windows.



For Verilog, a virtual signal showing the enumerated state is derived from the bit value. For VHDL, the signal represents the instance path of the state machine.

You can set these signals to track the *ModelSim* Wave window cursor by using the  button or choosing **Track Cursor** from the **Display** cascade. (The button is shown with a light background  when cursor tracking is enabled or a superimposed cross  when disabled.)

Removing Signals from Simulator Windows

You can remove a selected signal from the Wave window by using the  button or by choosing **Delete Wave** from the **Display** cascade of the **Simulate** menu.

You can remove signals from the List window by using the  button or by choosing **Delete List** from the **Display** cascade of the **Simulate** menu.

The  and  buttons (or commands in the **Simulate** menu) can be used in a state diagram to remove the current state signal from the Wave or List windows.


Adding Signals to the Simulator Log

You can add one or more signals to the simulator log by selecting the required signals on the block diagram and choosing **Add Log** from the **Display** cascade of the **Simulate** menu.

Signals added to the simulator log in this way are not displayed but their simulation activity is recorded and can be added to the Wave or List window for later analysis.


You can remove one or more selected signals from the simulator log by choosing **Delete Log** from the **Display** cascade of the **Simulate** menu.

Highlighting Signals in the Simulator

You can highlight the signal or signals in the simulator windows corresponding to a selected object in a block diagram, flow chart or state diagram by using the  button from the Simulation toolbar or by choosing **Highlight Object** from the **Display** cascade in the **Simulate** menu.


All occurrences of the selected signal in the ModelSim Structure, Source, Signals, List and Wave windows are highlighted.

Reporting Signal Information

You can report information about one or more selected signals by using the  button from the Simulation toolbar or by choosing **Signal Info** from the **Simulate** menu in a block diagram.

The ModelSim *examine*, *describe* and *drivers* commands are used to report current information about the current value, type information and a list of drivers to the main ModelSim window.

Adding and Removing Breakpoints

You can add breakpoints for any object which is referenced by the generated HDL from a block diagram, flow chart or state diagram by selecting the object and using the  button from the Simulation toolbar or by choosing **Add** from the **Breakpoints** cascade of the **Simulate** menu.

Breakpoints can be added to any object which corresponds to a executable line in the generated HDL. For example, a state or transition in a state diagram, an action box, decision box or wait box in a flow chart and a signal or bus in a block diagram.


If multi-line text (such as declarations or concurrent statements) is selected the breakpoint is added to the first line of the selected text.

Note



Line breakpoints may be corrected by up to ten lines if the selected object does not correspond to an executable line in the simulator.

If you select an object which has no corresponding executable HDL code, you are prompted to select another object.

Breakpoints are shown by an  icon on a diagram.


Breakpoints added to a HDL line in the ModelSim Source window are automatically added to the corresponding source object in the graphical view.


Note



If you set a breakpoint on a state name, HDL line breakpoints are added to all transitions which set the state as the next state.



For a hierarchical state machine, these transitions may be in another state diagram but will be shown when the diagram is displayed.

You can remove breakpoints from the selected object by using the  button or by choosing **Delete** from the **Breakpoints** cascade of the **Simulate** menu.


You can remove all breakpoints by using the  button or by choosing **Delete All** from the menu.


All breakpoints are automatically cleared when the simulator is closed.

Enabling and Disabling Breakpoints

You can disable breakpoints at any time during a simulation run for the selected signals by using the  button or by choosing **Disable** from the **Breakpoints** cascade of the **Simulate** menu and disable all breakpoints by using the  button or choosing **Disable All**.

The breakpoint icon for a disabled breakpoint is redrawn with no fill color.


You can enable breakpoints for the selected signals by using the  button or by choosing **Enable** from the menu.

You can enable all breakpoints by using the  button or by choosing **Enable All**.


Reporting Breakpoint Status

You can display information in the main ModelSim window about all currently set breakpoints by choosing **Report** from the **Breakpoints** cascade of the **Simulate** menu when you are using ModelSim.

Adding and Removing Simulation Probes

You can add simulation *probes* to signals on a block diagram by selecting one or more signals and using the  button from the Simulation toolbar or by choosing **Add** from the **Probes** cascade of the **Simulate** menu.

If one or more instances are selected, probes are added to all the signals connected to the instances.

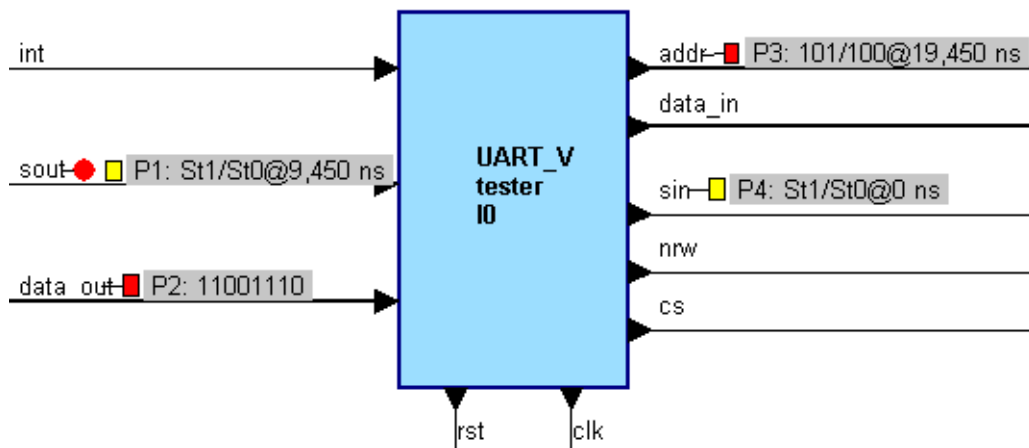
You can also add probes by using the  button in the ModelSim Wave window or by choosing **Add Probe** from the **Probes** menu in the ModelSim Wave, List or Signals windows.

A simulation probe displays the current value of the associated signal.

Simulation probes are added on a block diagram as yellow icons with associated text on a grey background adjacent to the signal name. However, you can move the probe independently and its associated signal is connected by an *anchor*.

You can set probe properties which control how each probe is displayed including an optional name, probe expression, display radix, previous value, time of change and visible anchor connecting it to the net.

The probes in the following example show a new value for the *data_out* and *addr* signals but the *sout* and *sin* signals have the same value as the last time the simulator stopped:



Probes *P1*, *P3* and *P4* in the example have been set to display the current/previous values and the time of the last change.

A breakpoint has also been attached to the *sout* signal.



The Probes and Breakpoints columns can be hidden or displayed by setting options in the **View** menu.



Note



The block diagram is automatically made read-only when a probe has been added and all probes must be removed before you can edit the view. All probes are automatically cleared when the view is closed and when you exit from the simulator.

The probes are updated whenever the simulator stops (at a breakpoint, forced stop or the end of a run). If the change indicator option is set and a signal has changed since the previous simulator step, the probe is shown as a red icon on a block diagram. Note that there is no color change if the signal value is the same although there may have been signal transitions during the simulation run and the signal has returned to its previous value.

You can remove probes from a selected signal by using the  button or by choosing **Delete** from popup menu or the **Probes** cascade of the **Simulate** menu and remove all probes by using the  button or by choosing **Delete All** from the menu.

If you are using ModelSim SE, you can set probes to track the ModelSim Wave window cursor by using the  button or choosing **Track Cursor** from the popup menu or from the **Probes** cascade of the **Simulate** menu. (The button has a light background  when cursor tracking is enabled or a superimposed cross when disabled.)


When cursor tracking is enabled for probes, the probe values track the current position of the active cursor in the ModelSim Wave window.

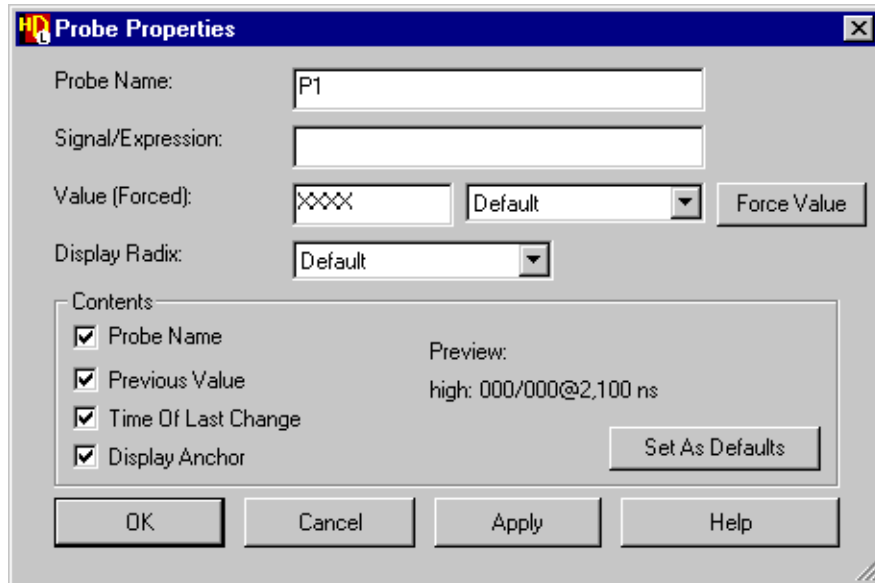
Note



Simulation probes are re-initialized if you restart the simulator but they are NOT automatically updated if you execute the ModelSim restore command in the simulator window. However, the probes are updated when the next run or step command is executed.

Setting Probe Properties

You can set simulation probe properties by using the  button in the Simulation toolbar or by choosing **Probe Properties** from the popup menu or the **Probes** cascade of the **Simulate** menu when one or more probes are selected to display the Probe Properties dialog box:



The dialog box allows you to specify the name of the probe and choose how it is displayed on the diagram.

You can optionally specify an expression if you want the probe to display a value other than the current signal value. For example, *dat_in(9:0)* would display only bits 9 DOWNT0 of the multi-bit bus *dat_in*.

The properties specified in the dialog box are applied to the selected probe or changes to the properties can be applied to all selected probes with unchanged properties remaining *<AS_IS>*.

The current probe properties for display radix and contents can be saved as defaults in your preferences using the **Set as Defaults** button.

Forcing Signal Values

You can force signal values for simulation by adding a probe and entering the required value in the Probe Properties dialog box using the **Force Value** button to force the required value.

You can also force a signal value by directly editing the value of a signal in a probe on a block diagram.

The signal is forced using the default strength currently set in *ModelSim* or you can choose the *-freeze*, *-drive* or *-deposit* drive strength arguments from a drop down list.

When the value of a signal has been forced, the probe is shown as a blue icon on a block diagram.

Choosing the Simulation Instance

When you send a command to the simulator, the signal or process is identified by a pathname that uniquely identifies its location in the design hierarchy.

This path is referenced relative to the component from which the current simulation was invoked. For example, if you have invoked simulation from the block diagram *DESIGNS/am2909/stack*, references will be relative to this path.


If your design hierarchy contains more than one occurrence of a component, you can open down into this component from each separate instance. When driving simulation from within such a component, it is necessary to specify the simulator hierarchy path in order to uniquely identify the signals and blocks it contains.


It is also possible to open separate windows on each instance in which case each instance can be animated separately.

Note



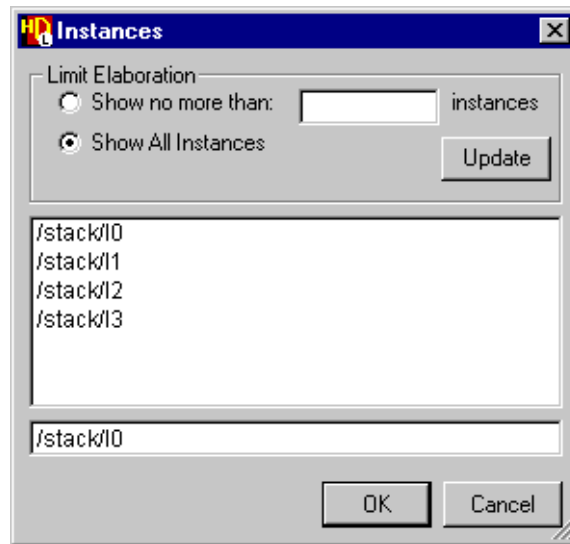
If the instance has not been selected, you are prompted to choose the instance when you use a command which requires the full hierarchy path.

You can display the Instances dialog box to set the simulator hierarchy path in a flow chart or state diagram by using the  button in the Animation toolbar or by selecting **Choose Instance** from the **Animation** menu.

In a block diagram, you can choose the  button in the Simulation toolbar or select **Choose Instance** from the **Simulate** menu.

For example, if there are four instances (*I0*, *I1*, *I2* and *I3*) of a component and simulation has been invoked from the parent block diagram, you can simulate any of the four instances by

setting the appropriate hierarchy path. You can also open block diagram windows for each instance and set a hierarchy path for each window.



The hierarchy path is also used when you add a breakpoint to an instance of a component which is used several times in the same diagram.

There can be many instances on a diagram (particularly if your design contains nested FOR generate frames) and the dialog box is initially empty. You can choose to show all available instances or specify a maximum number of instances to display in the dialog box.

This elaboration limit is saved as a preference and used when the dialog box is next displayed. The available instances determined by this limit are listed in the dialog box when you use the **Update** button.

The dialog box also allows you to explicitly enter a hierarchy path. For example, when your design contains many instances and you want to change the selected instance by simply incrementing its instance number.

This is also useful if a FOR generate frame has bounds which are not literal integers. Such a frame is treated as a single instance with an index of 0 and a warning message is issued when you select an instance path. However, you can then choose to accept the default index of 0 or explicitly edit the instance path to the required value.

Setting the Simulator Environment

The initial simulator environment is determined by the selected *design unit* in the *source browser* or the active view when you started the simulator. However, you can change the simulator environment to any lower level without having to re-invoke the simulator.

The following commands are available from the **Environment** cascade of the **Simulate** menu:

Selected	Set environment to the selected block or component instance in a block diagram.
Diagram	Set environment to the active block diagram, flow chart or state diagram view.
Top	Set environment to the top level of simulation.
Report	Reports the current simulator environment level. The pathname for the current simulator environment is reported in a popup dialog box.

Running the Simulator


You can run the simulator from a block diagram, flow chart or state diagram by using commands from the Simulation toolbar or the **Run** and **Step** cascades of the **Simulate** menu.

Running a Simulation

You can run the simulator using the following **Run** commands:



For Time

Run the simulator for the last specified time or you can use the  button to display a pulldown list. A pulldown list allows you to choose from the four most recently used timesteps, enter a time interval by displaying a dialog box or reset the default time step specified in the simulator. When you enter a time interval, the new interval replaces one of the default time steps in the pulldown list.



Forever

Run the until there are no more events scheduled.



Continue

Continue the simulation run after a step command or breakpoint.



To Next Event

Run the simulator until the next event.

Stepping Through a Simulation

You can step through a simulation using the following **Step** commands:



Over Line

Steps to the next HDL statement skipping any VHDL procedures, functions or Verilog tasks by treating them as simple statements instead of tracing each HDL line inside them.



Into Line

Steps into the next HDL statement.



Over Object

Steps to the next HDL statement representing a new graphical object (for example, the next state) by stepping over HDL lines for the current object. This command is available in a state diagram only.

Displaying Simulator Windows

You can open any simulator window by choosing one of the following options from the **View** cascade of the **Simulate** menu:

Source	Displays the simulator Source window
Structure	Displays the simulator Structure window
Variables	Displays the simulator Variables window
Signals	Displays the simulator Signals window
List	Displays the simulator List window
Process	Displays the simulator Process window
Wave	Displays the simulator Wave window
Dataflow	Displays the simulator Dataflow window
All	Displays all simulator windows

The Wave and List windows can also be opened automatically when you add signals as described in [“Adding Signals to Simulator Windows”](#) on page 418.

Restarting the Simulator


You can restart the simulator by using the  button or by choosing **Restart Simulator** from the **Simulate** menu in a block diagram, flow chart or state diagram.

When you restart the simulator, the design hierarchy is reloaded in the simulator, all captured data and animation views are cleared and the simulation time is reset to zero.

Using the ModelSim Source Window

If you have invoked *ModelSim* from a HDL Designer Series tool, the *ModelSim* Source window displays the source HDL files used for simulation.

The Source window is normally displayed in read-only mode.

If you need to make changes to a view which is described by a graphical view, the source design object should be modified in the HDL Designer Series tool and the  button used to regenerate, compile and restart simulation. However, if you want to directly modify a view which is described by a text source object, you can toggle the *ModelSim* Source window to editable mode from the **Edit** menu.

If you want to open a HDL text view of an embedded block from the HDL Designer Series tool you can choose **Send to Editor** from the popup menu over the HDL text or by double-clicking


over the embedded block. If *ModelSim* is invoked, this command sends the HDL text file to the *ModelSim* Source window instead of to the HDL Designer Series text editor.

Cross-Probing from ModelSim

If you have invoked *ModelSim* from a HDL Designer Series tool, the simulator toolbars and **Debug** menu provide additional commands which support cross-probing with the graphical views.


The following commands are provided in the main *ModelSim* window:

Table 11-4. ModelSim Main Window Commands

Button	Debug Menu	Description
none	Debug Detective Enabled	Enable or disable Debug Detective mode
none	Use Source Window	Enable Source window for HDL text views
	Trace To HDS	Open selected object as a graphical view




The following command is provided in the *ModelSim* Source window:

Table 11-5. ModelSim Source Window Commands

Button	Debug Menu	Description
	Trace To HDS	Open view corresponding to HDL line


The following commands are provided in the *ModelSim* Wave window:

Table 11-6. ModelSim Wave Window Commands

Button	Debug Menu	Description
	Trace To HDS	Open view where selected signal is declared
	Add Probe	Add a probe to the selected signal
	Cause	Show animation step at cursor time

The following command is provided in the *ModelSim* Structure window:

Table 11-7. ModelSim Structure Window Commands

Button	Debug Menu	Description
	Trace To HDS	Open selected object as a graphical view

The following menu commands are provided in the ModelSim List window:

Table 11-8. ModelSim List Window Commands

Debug Menu	Description
Trace To HDS	Open view where selected signal is declared
Add Probe	Add a probe to selected signal
Cause	Show animation at cursor time


The following menu commands are provided in the ModelSim Signals window:


Table 11-9. ModelSim Signals Window Commands


Debug Menu	Description
Trace To HDS	Open view where selected signal is declared
Add Probe	Add a probe to selected signal

If the **Use Source Window as Text Editor** option is set in the **Debug** menu for the main ModelSim window, all HDL text views are displayed in this window. If this option is unset, the HDS text editor is used.

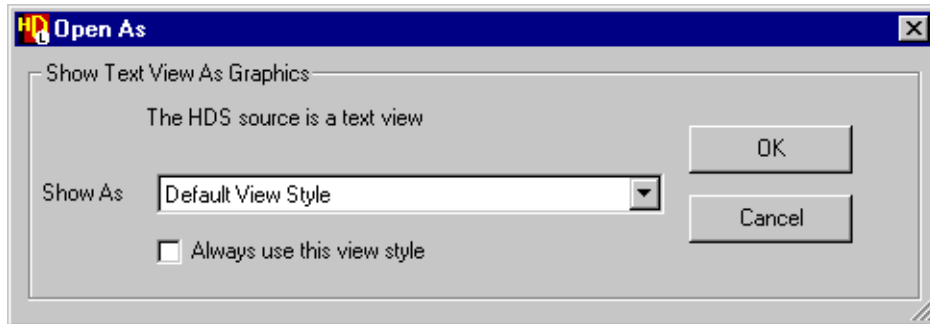
A  button is added to the toolbar and a **Trace To HDS** command to the **Debug** menu in the ModelSim Source window which can be used to open the source design object corresponding to the line of HDL code under the cursor.

The **Trace To HDS** command is also available in the **Debug** menu for the ModelSim Wave, List and Signals windows (or using the  button in the Wave window). It can be used in these windows to display the source object where the selected signal is declared. (Typically, this will be the test bench or top level block diagram for the design being simulated.) You can also trace a signal to HDS from the Wave or Signals windows by double-clicking on the signal.

A **Trace To HDS** command is also added in the **Debug** menu and a  button to the toolbar in the main ModelSim window and the ModelSim Structure window. They can be used in these windows to open the source object corresponding to the selected instance.

If the HDS source is a text file (and **Debug Detective Enabled** is set in the ModelSim **Debug** menu or the **Simulate** menu in any graphical view) the  button or **Trace To HDS** command can be used to display an Open As dialog box which allows you to render the source text file as a graphics view.

The Open As dialog box is also displayed when you double-click over a block or component instance in a block diagram, if the instance is described by a HDL text view.




You can choose to open the view using the default view style, or as a block diagram, state diagram, flow chart or text view. (The default view style is used if the specified view cannot be rendered.)


These views allow limited editing but fully support the simulation cross-probing and animation features.

Note



If you set the **Always use this view style** option in the dialog box, the dialog box is not displayed and HDS attempts to always render HDL text views using the last selected style.

You can add simulation probes to the active graphical block diagram view by using the  button in the ModelSim Wave window or by choosing **Add Probe** from the **Debug** menu in the ModelSim Wave, List or Signals windows. Refer to [“Adding and Removing Simulation Probes”](#) on page 421 for more information about using probes on graphical diagrams.

When you have invoked ModelSim from a HDL Designer Series tool, a  button is added to the ModelSim toolbar and a **Cause** command to the **Debug** menu in the simulator Wave window.

This command can be used to update all currently open animation windows to the simulation step or event immediately preceding the time marked by the Wave window cursor for the currently selected signal. If there is no cursor in the Wave window, the simulation time is set to zero.

If cursor tracking is enabled in a block diagram the values displayed by simulation probes in these views track the simulation time under the Wave window cursor. If cursor tracking is enabled in a state diagram and an animation view is displayed, the state machine animation tracks the cursor.

Note

The cursor tracking feature is not available if you are using ModelSim PE.

A **Cause** command is also added to the **Debug** menu in the simulator List window and can be used to update all open animation windows to the simulation step or event corresponding to the selected line in the List window.

ModelSim uses the *drivers* command to determine the driver of the signal. If the signal is driven from an instance which corresponds to a diagram that can be animated and which has data capture enabled, it is opened (or popped to the front if already open).

If the selected signal is a vector, the driver of the first element in the vector is chosen. This may be incorrect if the vector has more than one driver.

If the instance corresponds to a state machine or flow chart with concurrent diagrams, the default concurrent state machine or flow chart is opened although the process driving the signal may be contained in one of the other concurrent diagrams. The last view of the diagram is displayed and may not include the current animation state or action (especially if it is in a hierarchical diagram).

State Diagram and Flow Chart Animation

The state diagram and flow chart views in a VHDL or Verilog design can be animated if a ModelSim or NC-Sim simulator is available.

Animation exercises the generated HDL for your design and displays simulation behavior graphically on the source diagrams as well as in the monitoring windows provided within the simulator.

You can animate individual flow charts and state machines, one or more diagrams in a hierarchical branch of your design or the entire design. For example, you can invoke simulation from a test bench block diagram and simultaneously display animation for the flow chart which controls the test bench and state diagrams within the design under test.

Note

If you enable animation in a concurrent or hierarchical diagram, all the flow charts or state diagrams are animated. Flow chart or state diagram views describing an embedded block can be animated but you cannot animate an embedded view which is contained within a FOR generate frame.

The views can be animated in three usage modes:

- Animation of one or more diagrams or an overall system with simulation stimuli provided by a HDL test bench. The test bench runs scenarios which are selected at run

time by setting HDL signals and variables in the test bench. In this mode, simulation is initiated by a run command.
















- Animation of one or more diagrams with simulation stimuli provided by a "dofile" which contains force, breakpoint and run commands.
- Interactive animation of a single diagram using single stepping or run simulation commands controlled from the simulator command entry window. Forces and breakpoints are applied interactively.




The **Enable Communication with HDS** option must be set in the Simulator Invoke Settings dialog box if you want to use the simulation cross-probing or animation facilities.

Animation Toolbar

The Animation toolbar is automatically displayed in the state diagram and flow chart views when a supported simulator has been invoked and supports the following commands:

Table 11-10. Animation Toolbar

Icon	Description
	Step backward through animation history
	Step forward through animation history
	Sets review mode to step by states
	Sets review mode to step by simulation events
	Moves to a specified simulation time
	Moves to a specified simulation time
	Moves to the start of simulation time
	Moves to latest simulation time
	Enables capture of animation data
	Clears the animation data for the current instance
	Displays animation view
	Sets up the activity trail for capture and display
	Moves ModelSim Wave and List windows to current animation time
	Choose instance for simulation
	Links all currently animated diagrams

Note that the ,  or  buttons display a pulldown menu which allows you to change the mode. When one of these options is chosen the button changes to indicate the selected mode.

You can also step backward or forward through the animation by using the **Ctrl + Shift + <** or **Ctrl + Shift + >** shortcuts.

The toolbar is normally displayed at the bottom of the diagram window but can be undocked, moved, docked or hidden in the same way as the other toolbars.

The toolbar can be displayed or hidden by setting the **Animation Tools** option in the **Toolbars** cascade of the **View** menu.

Refer to “[Toolbars](#)” on page 20 for more information about toolbars.

Enabling Data Capture


Animation data capture can be enabled for any graphical state machine or flow chart within the current hierarchy being simulated. However, if you want to apply stimulus to a hierarchy of diagrams, you must invoke simulation from a structural HDL view at the appropriate level.


In order to animate a flow chart, the flow chart must have been generated with the **Instrument HDL for Animation** option enabled in the **Generation** tab of the Flow Chart Properties dialog box.

In order to fully animate a state diagram, the state machine must have been generated with the **Instrument HDL for Animation** option enabled in the **Generation** tab of the State Machine Properties dialog box. If this option was not enabled, the animation will show changes of state but full animation (for example, transitions taken and moving by clock cycle) is not available.

The extra HDL code generated for animation is enclosed between translation pragmas so that it can be ignored by downstream tools which recognize the pragmas.


If you invoke the simulator from within a flow chart or state diagram, you are prompted whether to enable data capture and show animation. If you confirm this dialog box, data capture is enabled and the animation view is displayed.


Alternatively, if the simulator is already invoked, you can use the  button in the Animation toolbar or choose **Data Capture** from the **Animation** menu to enable animation for the active state diagram. The button is shown pressed when data capture is enabled.

The animation view is automatically selected when you enable data capture but can be toggled by using the  button or choosing **Show Animation** from the **Animation** menu. The button is shown pressed when the animation view is displayed.

When data capture is enabled, simulation data is stored to capture animation activity. When the animation view is displayed, the diagram is set to be read-only and the graphics are updated directly from the simulation data whenever the simulator stops at a breakpoint or at the end of a run.

You can enable animation for all instances in the current simulation hierarchy by choosing **Global Capture On** from the **Animation** menu (or from the **Animation** cascade of the **Options** menu in the *design manager*). An animation view is automatically shown for any state diagrams or flow charts which are opened after this option has been set.

Data capture can be stopped for all diagrams by choosing **Global Capture Off** from the **Animation** menu (or from the **Animation** cascade of the **Options** menu in the *design manager*) and you can stop the capture of animation data for the active diagram by unsetting the  button or by clearing the **Data Capture** option in the **Animation** menu.


The stored data is cleared when the simulator is restarted or reloaded and when the session is exited. In addition, you can clear the animation data at any time by using the  button or by choosing **Clear Captured Events** from the **Animation** menu.

Setting the Activity Trail

The current and previous steps (for a flow chart animation) or the current and previous states plus the last transition (for a state machine animation) are normally highlighted in the animation view.

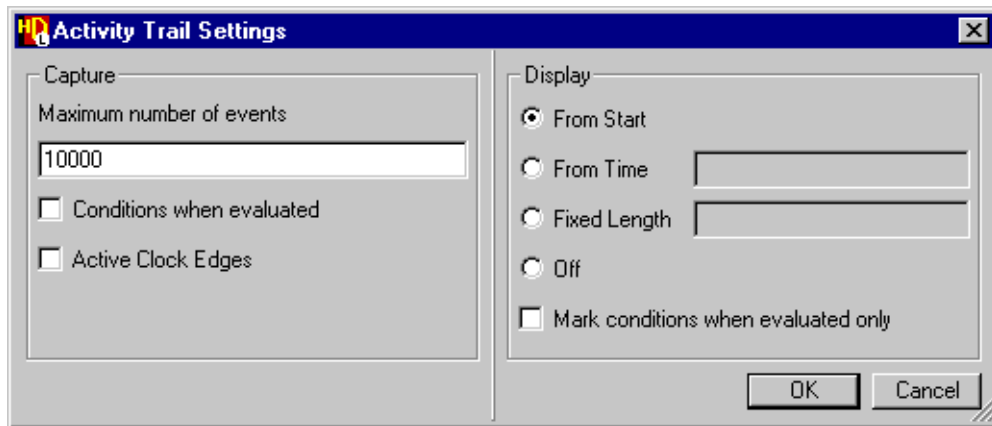
The activity trail allows you to control how much additional animation activity is displayed relative to a particular time in the simulation history or at a particular clock cycle in simulation history.

You can set the activity trail to highlight all visited steps in a flow chart or visited states and transitions in a state machine, a specified length (number of steps or transitions) or no activity trail.

You can change the activity trail settings by using the  button or choosing **Activity Trails** from the **Animation** menu to display the Activity Trail Settings dialog box.

If you are using ModelSim, you can specify the maximum number of simulation events captured for each diagram during an animation run. When this limit is exceeded, old events are discarded to reduce the total amount of stored data.

For a state machine, you can also choose whether to capture data about evaluated conditions or active clock edges. If all the capture options are unset, no data is saved, however, the animation is updated with the latest simulation data when the simulator stops.



You can specify the length of the displayed activity trail in terms of the simulation time or the number of visited objects (an object corresponds to a visited state in a state machine).

If you choose the **From Start** option, all objects visited since data capture was enabled are highlighted. You can also hide the activity trail except for the last step (or last state and transition) by selecting the **Off** option.

For a state machine, you can also choose whether conditions evaluated to be true but not followed are highlighted in the activity trail. This may occur if a condition is evaluated to be true but reverts to false before the next clock edge which would cause the transition to be completed.

Your activity trail choices are saved as preferences and re-used for the next animation run.

The initial data capture preferences for events captured, evaluated conditions and active clock edges are shown as a comment in the main simulator window which is repeated if the activity trail settings are modified.

For example, `hds_anim_prefs 8000 0 1` indicates that up to 8000 events can be captured, evaluated conditions are ignored but events on active clock edges are included.

Graphical Highlighting

The following color conventions are used in an animated flow chart:



- red fill: Current step
- yellow fill: Previous step
- blue fill: Previously visited steps
- white fill: Unvisited objects




The following color conventions are used in an animated state diagram:


- red fill: Current state or last transition taken
- yellow fill: Previous state or transitions
- blue fill: Previously visited states and transitions
- green fill: Transitions evaluated but not followed
- white fill: Unvisited objects

Reviewing Animation

You can use the animation view to investigate the cause and effect of animation events by moving to a specified point in the animation.

In an animated flow chart, you can move forwards or backwards to display the preceding and following steps by using the  or  buttons from the Animation toolbar or choosing **Goto Next** or **Goto Previous** from the **Animation** menu.



In an animated state diagram, you can choose whether to **Move By States**, **Move By Events** or **Move By Clocks** by selecting options in the **Animation** menu or by using the ,  or  buttons on the Animation toolbar.

You can change the current setting of these buttons by using the  icon to pulldown a choice menu. The current setting of these buttons is indicated on the button.

Note



The **Move By Clocks** option is not available unless the **Active Clock Edges** option is enabled in the Activity Trail Settings dialog box to capture events associated with active clock edges.


For each of these options, you can move forwards or backwards to display the preceding and consequential events (change of state, clock cycle or other animation event) by using the  or  buttons from the Animation toolbar or choosing the **Goto Next** or **Goto Previous** option (**Clock**, **State** or **Event**) from the **Animation** menu.

You can also use the **Ctrl + Shift + <** shortcut to **Goto Previous** or the **Ctrl + Shift + >** shortcut to **Goto Next**.


You can use the  button (or choose **Goto Time** from the menu) to view a specified animation time, the  button (or **Goto Start** from the menu) to return to the start of the animation or choose the  button (or **Goto Latest** from the menu) to view the latest animation event.

If you choose **Goto Time**, a dialog box is displayed for you to enter an absolute animation time: When you execute the dialog box, the animation time is set to the specified absolute time (or the time of the closest preceding event if no event occurred at the specified time).

When you use any of these commands, the animation status is redrawn and the activity trail is shown relative to the new animation time. The current animation time is displayed on the status line at the bottom of the animated window and an indication when you are at the start or end of an animation run.

When you are using *ModelSim*, you can use the  button or choose **Cause** from the **Animation** menu to move the Wave window cursor to the current animation time. (The current animation time should also be highlighted in the *ModelSim* List window.)

Linking Diagrams for Animation

You can link all open state diagrams and flow charts in the simulation hierarchy which are currently being animated by using the  button or choosing **Link Diagrams** for the **Animation** menu in any animated diagram.

When this command is enabled, all animation review commands executed in one diagram, also animate the linked diagrams.

The button is shown pressed and the link closed  when diagrams are linked.

Mixed Language Animation

Mixed language animation is supported when you are using the *ModelSim* simulator. Note however, that VHDL is case-insensitive and case sensitive Verilog names (for example, different instances named *I2* and *i2*) may cause problems in a mixed language design. You are advised to avoid using names that differ only by case in Verilog designs that need to co-exist with VHDL.

You can animate a mixed language design using the NC-Sim simulator but you cannot display animation views for both languages in the same simulation run. If the top level view is Verilog, you can instrument the generated HDL for all VHDL and Verilog views but can only display animation for the Verilog diagrams. If the top level view is VHDL, error messages are issued if the design includes any instrumented Verilog views and the design cannot be loaded.

Chapter 12

Using a Test Bench

This chapter describes how you can create a graphical test bench to validate your designs.

Test Benches	439
Creating a Test Bench	440
Defining Stimulus	441
Using ModuleWare Stimulus Generator Parts	441
Defining Stimulus on a Flow Chart	441
Defining Stimulus using Lookup Tables	443
Defining Stimulus using TextIO	444
Defining Stimulus using a State Machine	446
Generating a Clock using HDL Statements	446
Analyzing Results	447
Re-using a Test Bench	448

Test Benches

A test bench allows you to apply stimuli to your design and ensure that it fully meets the specified requirements.

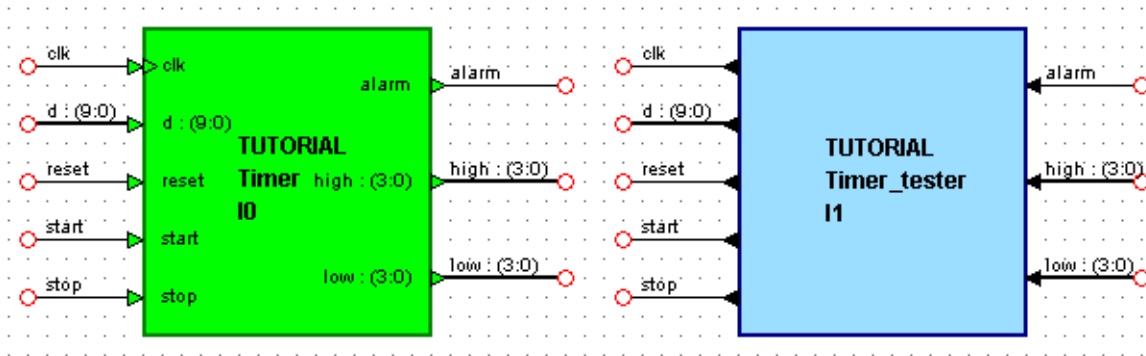
The input stimuli and corresponding output waveforms are just as important as the design description itself. If made available early in the design cycle, they can be used to test all phases of the design including behavioral, RTL and gate level designs.

A set of test stimuli available at the start of the design process reduces the number of design iterations by allowing the designer to check results for each level of abstraction and to verify the consistency of the design between levels.

You can implement a test bench by instantiating your design as a component in a block diagram or IBD view. Then connect the component to a block (or blocks) implementing the stimulus and output checking circuits.

These blocks can be described directly using VHDL or Verilog HDL text views or graphically by using block diagrams, IBD views, state diagrams, truth tables or flow charts.

For example, the test bench in the graphical design tutorial uses a single block *Timer_tester* described by a flow chart which provides both the test stimuli and output checking logic.

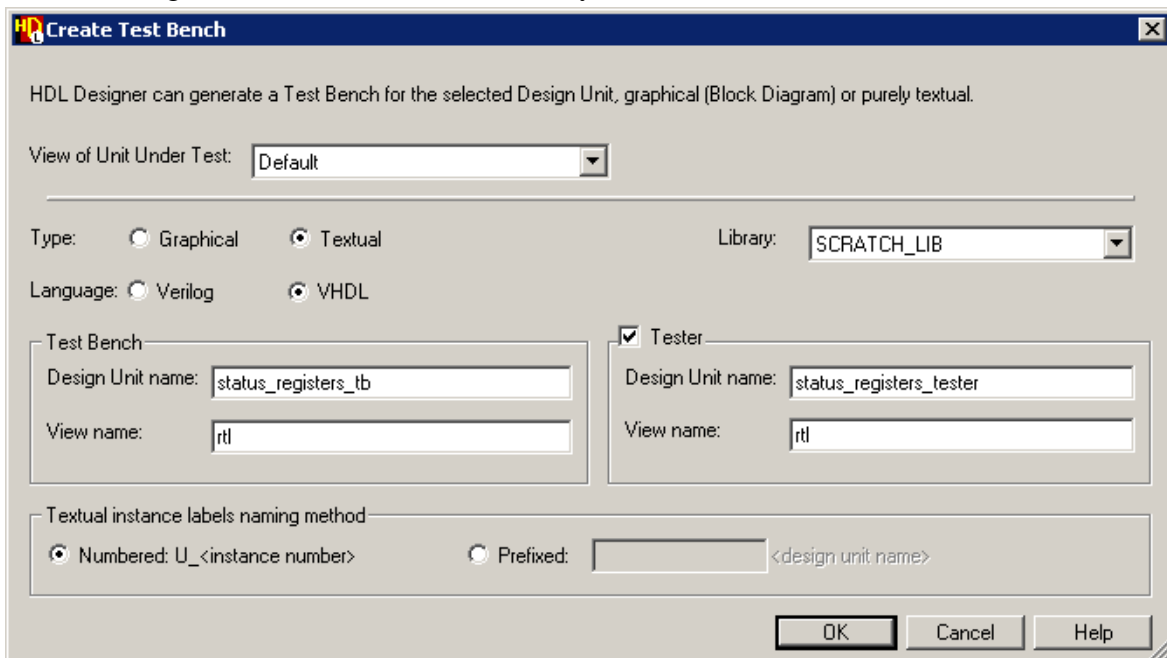


Creating a Test Bench

You can automatically create a test bench by choosing **Test Bench** from the **New** cascade in the **File** menu of any graphic editor window that represents a component. This command is also available in the *design explorer* when a component design unit or component design unit view is selected.

The Create Test Bench dialog box is displayed with a default name for the test bench created by adding the suffix *_tb* to the design unit name. You can change this name and choose a library to contain the test bench.

You can also create a tester unit to test your design unit by checking the **Tester checkbox**. The **tester module is created with a** default name adding the suffix *_tester* to the design unit name. You can change this name and choose a library to contain the tester unit.



When you confirm the dialog box, a block diagram is created containing an instance of the component under test and a tester block with ports corresponding to each port on the component which are implicitly connected to the component ports by name.

The tester block can be used to provide input stimulus and output checking signals by opening down to create a child view. This view can be any valid design unit view. For example, it may be a block diagram, IBD view, HDL text view, sequential flow chart or state machine.

Any required VHDL packages (for example, if the interfaces to the component are custom defined) are automatically included on the test bench block diagram.

Defining Stimulus

The HDL Designer Series ModuleWare library includes a number of waveform generator parts which can be instantiated in a block diagram, IBD view or HDL text view are used to generate stimulus waveforms for your test bench.

You can also define stimulus using concurrent and sequential processes in a flow chart or state machine or by writing a HDL text view using standard techniques such as lookup tables and TextIO.

When you are using VHDL, the *std_developerskit* standard library contains a number of procedures and functions which may be useful when you are designing a test harness. For example, these can be used to build Typical or Maximum timing into models, convert VHDL types to strings for use in File I/O and ASSERTS, perform arithmetic on unlimited length register sets and build memories of any size and program them from ASCII files.

Using ModuleWare Stimulus Generator Parts

The ModuleWare library *Stimulus* category provides a number of parameterized generators which can be used to setup clock, pulse and constant, random value or counter based waveforms.

Refer to the [ModuleWare Reference Guide](#) for a full description of these parts.

Refer to “[Instantiating a ModuleWare Component](#)” on page 116 for information about using ModuleWare parts in a block diagram or IBD view.

Refer to the [DesignPad Text Editor User Guide](#) for information about instantiating ModuleWare parts in a HDL text view.

Defining Stimulus on a Flow Chart

Stimulus generation within a test bench is predominantly sequential in nature.

Although separate processes are likely to be used for clock and reset generation, most of the code to generate stimuli is likely to be contained in a single process.

This process can conveniently be represented as a flow chart and separate action boxes can be used to represent procedures and functions for generic operations (such as 'write to register') or to separate out distinct blocks of code.

The result is a relatively short, top-level process which sequentially calls a series of procedures and functions to perform the main tasks in turn.

There are a number of techniques which can be used to define stimulus on a flow chart including Wait, Loop and Case statements.

Wait Statements

Wait statements can be used for results analysis where signal values are checked relative to one another or particular values are expected at certain times.

The VHDL WAIT command has four variants:

WAIT	causes an indefinite suspension of a process, for example, after an initial trigger pulse.
WAIT FOR <time>	suspends the process for the specified time.
WAIT UNTIL <condition>	suspends the process until a specified condition becomes true.
WAIT ON <signal>	suspends the process until an event occurs on the specified signal(s).

You can combine FOR, UNTIL and ON clauses which may also include valid arithmetic operators.

For example:

```
wait until ((high="000")AND(low="0000")) for 6000ns;
```

In Verilog, you can use wait statements and also use the @ operator to specify an event or # to specify a time delay.

For example:

```
wait ((high==4'd0)&&(low==4'd0));  
@(posedge clk);  
#12;  
#clk_prd;
```

Loop Statements

Loops can be used to apply repeating patterns, read and apply values from tables or files and progressively increment or decrement values. For example, a loop variable can be used to assign a value using a function:

VHDL

```
FOR J IN 0 TO 7 LOOP
  x <= conv_funcnt(J);
  WAIT FOR 20 ns;
END LOOP;
```

Verilog

```
for (j=0;j<=7;j=j+1)
  x = conv_function(J);
```

Case Statements

If stimulus is to be applied at different specific times, a counter can be used as an index within a Case statement to count clock cycles and determine which stimuli set is applied. For example:

```
CASE elapsed_counter IS
  WHEN 13 => x <= '1';
  y <= '0';
  WHEN 27 => x <= '0';
END CASE;
```

Defining Stimulus using Lookup Tables

A lookup table containing arrays or records can be defined using CONSTANTS in the declarations on a diagram or in a separate VHDL package. Such a table can be sequenced through or used to apply particular values depending on the value of other index signals.

Lookup tables exploit the power of HDL and can be easily related to the required results. However, recompilation is necessary if any changes are made to the code.

In the following example, the procedure *write_table* performs a series of write operations to load internal data tables and simulate the operation of an external microprocessor.

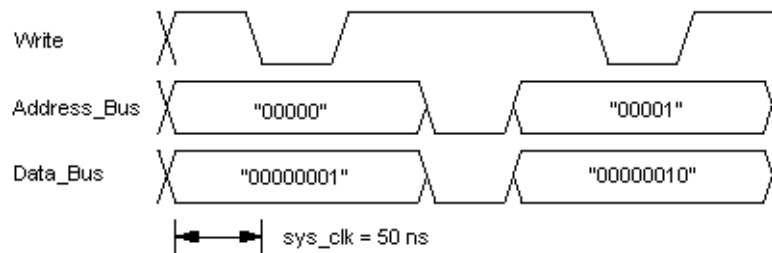
```
CONSTANT sys_clk : time := 50 ns;
CONSTANT row : INTEGER := 4;
CONSTANT col : INTEGER := 4;
TYPE table is ARRAY(1 TO row, 1 TO col) OF INTEGER;
CONSTANT lookup_table : table := (
  ( 1, 2, 3, 4 ),
  ( 5, 6, 7, 8 ),
  ( 9,10,11,12 ),
  ( 13,14,15,16 )
);
PROCEDURE write_table(
  SIGNAL Write : OUT slv_0d0;
  SIGNAL Address_Bus : OUT slv_4d0;
  SIGNAL Data_Bus : OUT slv_7d0) IS
BEGIN
  FOR i IN 1 TO row LOOP
```

```
FOR j IN 1 TO col LOOP
  Write <= '1' ;
  Address_Bus <=
    to_stdlogicvector((((i-1)*4)+(j-1)),5);
  Data_Bus <=
    to_stdlogicvector(lookup_table(i, j), 8) ;
  WAIT FOR sys_clk ;
  Write <= '0' ;
  WAIT FOR sys_clk ;
  Write <= '1' ;
  WAIT FOR sys_clk ;
  Address_Bus <= (OTHERS => '0') ;
  Data_Bus <= (OTHERS => '0') ;
  WAIT FOR sys_clk ;
END LOOP ;
END LOOP ;
END write_table ;
```

Note



Nested For loops are used to sequence through each row and column of the table. The values for *Address_Bus* are calculated from the row and column indices. All bits of each bus are set to '0' between write cycles by assigning (OTHERS => '0').



Defining Stimulus using TextIO

TextIO provides a mechanism to read stimulus files into a test bench at run time. This makes it easier to change values and does not need recompilation. In this way, a test bench can be made general purpose. For example, the same test bench can be used for a number of variants of the same basic design.

The VHDL 93 standard adds a number of enhancements to the basic TextIO capability, allowing files to be opened and closed for read or write at run time. The addition of pointers allows navigation within a file.

In the following example, a file *ctrl_registers.txt* contains values to be loaded into the three control registers *RT1*, *RT2* and *RT3*. The procedure *write_ctrl_registers* reads the data from the file and simulates the microprocessor writing values into the internal control registers.

```

FILE ctrl_file : TEXT IS IN "ctrl_registers.txt" ;

PROCEDURE write_ctrl_registers(
    SIGNAL Write : OUT slv_0d0;
    SIGNAL Address_Bus : OUT slv_4d0;
    SIGNAL Data_Bus : OUT slv_7d0) IS

    VARIABLE good : BOOLEAN ;
    VARIABLE RT : INTEGER ;
    VARIABLE ctrl_line : LINE ;

BEGIN
    IF NOT ENDFILE(ctrl_file) THEN
        READLINE(ctrl_file, ctrl_line) ;
        FOR i IN 0 TO 2 LOOP
            READ(ctrl_line, RT, good) ;
            IF good THEN
                Write <= '1' ;
                Address_Bus <= to_stdlogicvector((16 + i), 5) ;
                Data_Bus <= to_stdlogicvector(RT, 8) ;
                WAIT FOR sys_clk ;
                Write <= '0' ;
                WAIT FOR sys_clk ;
                Write <= '1' ;
                WAIT FOR sys_clk ;
                Address_Bus <= (OTHERS => '0') ;
                Data_Bus <= (OTHERS => '0') ;
                WAIT FOR sys_clk ;
            END IF ;
        END LOOP ;
    ELSE
        ASSERT false REPORT "END OF FILE reached"
        SEVERITY NOTE ;
    END IF ;
END write_ctrl_registers ;

```

The *ctrl_registers.txt* file contains the following data:

```

-- ctrl_registers.txt
--      RT1   RT2   RT3
--      1     2     3
--      4     5     6

```

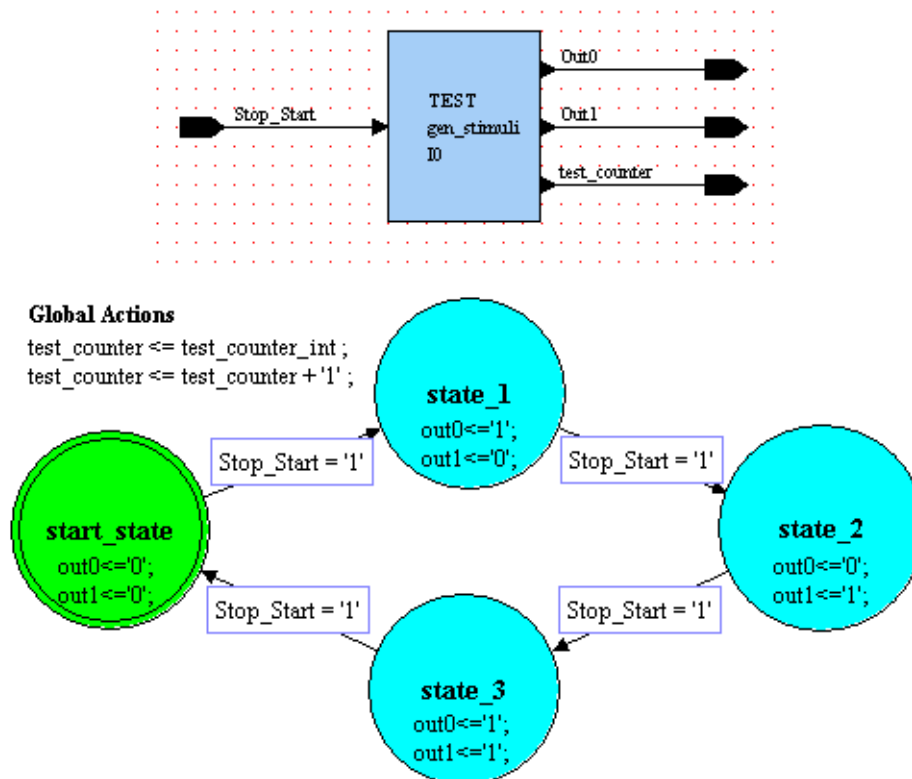
The file is declared and a line is read from the file into the variable *ctrl_line*. A loop then reads each of the three fields in turn into the variable *RT*. Variable *good* is the status returned by the read operation. The *Address_Bus* value is calculated from the loop index and the value of *RT* is assigned to the *Data_Bus*.

The TextIO function *ENDFILE* indicates when the end of the file has been reached. In this example, an ASSERT statement issues a note to that effect.

Defining Stimulus using a State Machine

A state machine can be used to apply a sequence of stimuli values, either on each clock or when certain conditions are true.

In the following example, an internal signal *test_counter_int* is created and used to count the clocks to the state machine. The count value is assigned to the *test_counter* output signal which can be used for stimulus in the test bench.



Generating a Clock using HDL Statements

A clock or repeating waveform can be generated using concurrent or sequential statements.

If your test bench is implemented as a flow chart, these statements can be included as concurrent statements on the diagram. Alternatively, the statements could be included in a HDL text view defining a clock generator block on a block diagram.

When you are using VHDL, a repeating clock waveform can be generated using concurrent statements. For example, the following architecture declarations and concurrent statements are used to generate a 10MHz clock by inverting the *int_clk* signal AFTER half the clock period:

```
Architecture Declarations
CONSTANT clk_prd : time := 100 ns;
```

```
SIGNAL int_clk : std_logic := '0';

Concurrent Statements
int_clk <= not int_clk AFTER clk_prd / 2;
clk <= int_clk;
```

Alternatively, you can use a sequential VHDL process. For example, the following architecture declarations and sequential statements are used to generate a 10MHz clock by looping around four sequential statements which alternately assign the values '0' and '1' to *iclk* after a WAIT of half the clock period:

```
Architecture Declarations
CONSTANT clk_prd : time := 100 ns;
SIGNAL iclk : std_logic;

Concurrent Statements
clock_gen : PROCESS
BEGIN
    iclk <= '0';
    WAIT FOR clk_prd/2;
    iclk <= '1';
    WAIT FOR clk_prd/2;
END PROCESS clock_gen;
clk <= iclk;
```

When you are using Verilog, a repeating clock waveform can be generated using initial and always statements.

For example, the following module declarations and concurrent statements are used to generate a 10MHz clock by inverting the *int_clk* signal after half the clock period:

```
Module Declarations
reg int_clk;
parameter clk_prd = 100 ;

Concurrent Statements
initial
begin
    int_clk = 0;
    forever #(clk_prd/2) int_clk = ~int_clk;
end
always @(int_clk)
    clk = int_clk;
```

Analyzing Results

The output checking logic in a test bench can perform analysis on the outputs from the design under test. It can also give a Pass or Fail indication for the design and provide messages indicating where results differ from those expected.

When you are using VHDL, ASSERT statements can be used to display messages in the main simulator window if a test condition is false. Assertions have three severity levels: Error, Warning or Note and you can specify what severity causes the simulator to stop.

The *std_iopack* package in the *std_developerskit* library contains string concatenation functions which can be used to include simulation values in messages as shown in the following example:

```
ASSERT false
REPORT "DR is " & to_string(DR,"%d") & " clks wide"
SEVERITY note;
ASSERT (DR = DR_REF)
REPORT "DR width incorrect!"
SEVERITY ERROR;
```

When you are using Verilog, messages can be displayed using the Verilog *\$display()* function. For example:

```
$display("DR width incorrect!");
```

For a control dominated system, the behavioral model used to generate the expected results can be represented by a state machine. For example, when using a master counter to count clock cycles and apply stimuli at certain counter values, the counter signal can also be fed to the results analysis process.

A state machine may be used to perform specific checks depending on the value of the counter or other sets of conditions on inputs.

In the same way that lookup tables can be used as a source of input stimuli, they can also be used to store reference values.

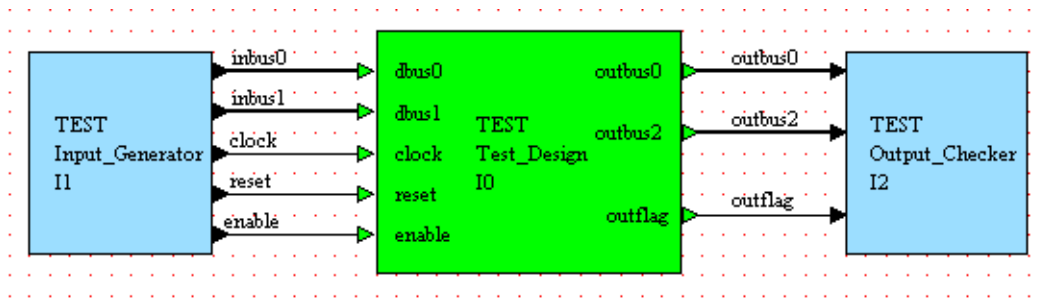
TextIO can be used to read in values corresponding to the expected results from ASCII files which were created manually, or from a behavioral model. It can also be used to write out results or errors to a file for subsequent analysis or documentation.

Re-using a Test Bench

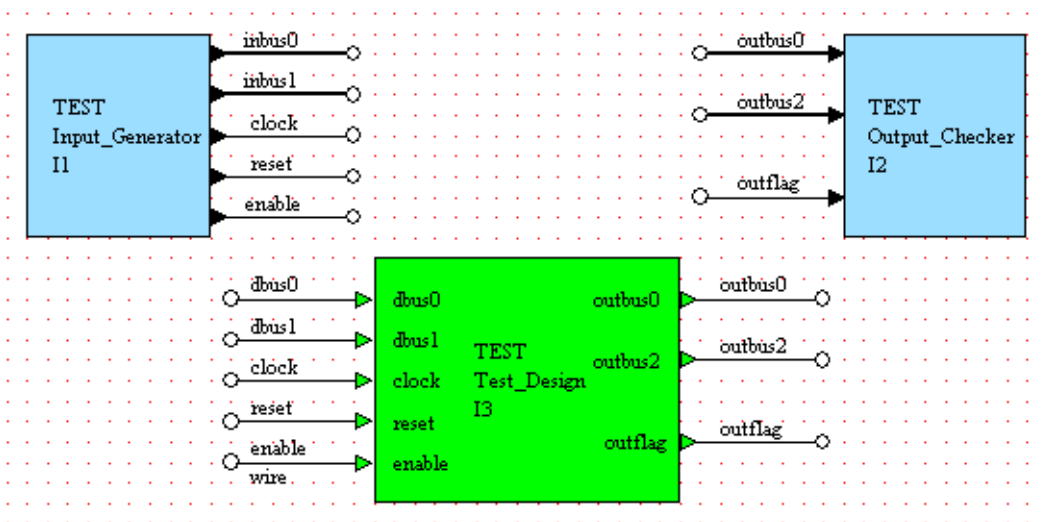
A test bench can be re-used to verify different stages of your design. For example, to check that the RTL version of your design performs identically with the initial behavioral version.

The design under test can be updated by changing the default view of the component representing the design under test or by instantiating a new test design into an existing test bench. If the interface signals correspond to those used in the test bench, you can add signal stubs to the test component and use connection by name to avoid having to re-connect signals between the test harness and the design under test.

For example, the following block diagram with explicit connections:



is functionally equivalent to the following diagram which is connected by name:



HDL Designer Series Glossary

This glossary defines the standard terminology used in the [HDL Designer Series](#) tools.

— A —

action box

A named object on a [flow chart](#) or [ASM chart](#) containing [actions](#) which are executed when the box is entered by a [flow](#). Each action box must have one input flow and one output flow. See also [case box](#), [decision box](#) and [wait box](#).

action

An operation performed by a [state machine](#), [flow chart](#) or [truth table](#) which modifies its output signals. In a [state diagram](#), there can be [transition actions](#) executed when an associated [condition](#) occurs or [state actions](#) executed when a [state](#) is entered. In a flow chart, the actions are executed when a [flow](#) entering the [action box](#) is followed. In a truth table, actions are generated from the values assigned to a variable in an output column or can be explicitly entered as additional actions in an unnamed output column. See also [global actions](#).

activity trail

A summary of simulation activity ([states](#) visited and [transitions](#) taken) displayed on an animated [state diagram](#).

anchor

An anchor attaches a text element to its parent object. For example, the name and [type](#) of a [signal](#) in a [block diagram](#) or the [transition text](#) and its [transition arc](#) in a [state diagram](#). An anchor is also used to attach a simulation [probe](#) to its associated signal.

architecture declarations

User-specified [VHDL](#) statements which can be entered in a [state diagram](#), [flow chart](#) or [truth table](#) and are declared for the corresponding [VHDL architecture](#) in the generated HDL. Architecture declarations are typically used to define local signals or constants. See also [entity declarations](#) and [process declarations](#).

ASIC

ASIC stands for Application Specific Integrated Circuit.

ASM

An algorithmic [state machine](#) describes the behavior of a system in terms of a defined sequence of operations which produce the required output from the given input data. These sequential operations can be represented using flow chart style notation as an [ASM chart](#).

ASM chart

A graphical representation of an algorithmic state machine which uses flow chart style objects to represent *states*, *conditions* and *actions*.

asynchronous

An asynchronous process is activated as soon as any of its inputs have any activity on them rather than only being activated on a clock edge. See also *clocking*.

— B —

black box

A view which has HDL translation pragmas set so that it is not analyzed or optimized for synthesis. See also *don't touch*.

black box instance

An instance of a *component* on a *block diagram* or *IBD view* which has no corresponding *design unit*. A black box instance may exist in a partial design which instantiates a view which has not been defined.

block

The representation of a functional object on a *block diagram* or *IBD view*. Also the *design unit* that contains the object definition. A block has a dynamic interface defined by the *signals* connected to it on the diagram and is typically defined by a *child* block diagram, IBD view, *state diagram*, *flow chart*, *truth table* or *HDL text* view. See also *embedded block* and *component*.

block diagram

A *diagram editor* view which defines a *design unit view* in terms of lower level *blocks* and *components* connected by *signals*. See also *IBD view*.

bottom-up design

The process of designing a system starting from the primitive or leaf-level views and progressing up through parent views until the design is completed. See also *top-down design*.

bounds

The *range* of possible values for a *signal* with integer, floating, enumeration or physical *type*. Also used to specify the index constraint for an array type. A *VHDL* range is normally shown in the format (15 DOWNT0 0) or (0 t0 7). A *Verilog* range is shown in the format [15 : 0] or [0 : 7].

breakpoint

A breakpoint can be used to interrupt the progress of a simulation at a specific point in the generated HDL. For example, you could set a breakpoint on a *signal* to interrupt the simulation when the signal changes value or on a *state* to interrupt the simulation when the state is entered.

bundle

A group of *signals* and/or *buses* with different *types* drawn as a composite line on a *block diagram*.

bus

A named vector *signal* with a *type* and *bounds* drawn as a composite line on a *block diagram*. See also *net* and *bundle*.

— C —

case box

A named object which represents a CASE statement on a *flow chart* or *ASM chart*. When used for decoding action logic each Case has an associated End Case object. A case box has one input *flow* and one or more output flows corresponding to the possible values for an evaluated CASE expression. See also *action box*, *decision box*, *if decode box* and *wait box*.

child

A view instantiated below its *parent* in the design hierarchy. A *component* or *block* on a *block diagram* or *IBD view* typically has a child *design unit view* which may be another block diagram, IBD view, *state diagram*, *flow chart*, *truth table* or a *HDL text* view. Also used for the embedded view representing a *hierarchical state* or “*hierarchical state box*” on page 460 in a hierarchical state machine or a *hierarchical action box* in a hierarchical flow chart or *ASM chart*.

clocked signal

A *signal* in a *state machine* whose value is assigned to an internal signal by the clocked process. This internal signal is continuously assigned to the real output signal. No default value need be specified. Typically used for an internal counter whose value is also required as an output. See also *combinatorial signal* and *registered signal*.

clocking

The timing aspects of behavior can be *asynchronous* or *synchronous* (explicitly clocked).

clock point

An object on an *ASM chart* which displays the clock *signal* name and *condition*. See also *enable point* and *reset point*.

clone window

A duplicate view of a *graphical editor* window. All select, highlighting and edit operations are made in both windows. However, you can display different parts of the diagram or table in each window.

combinatorial signal

A *signal* in a *state machine* whose value is directly assigned to the output port. See also *clocked signal* and *registered signal*.

comment graphics

Annotation graphics which can be used for illustration on a *block diagram*, *state diagram*, *flow chart* or *symbol*.

comment text

Annotation text on a *block diagram*, *state diagram*, *flow chart* or *symbol* which can optionally be attached to an object and included as comments or HDL code in the generated HDL for the diagram.

compiled library

A repository within a *library* containing downstream compiled objects usually created by compiling the *HDL* files in a *design data library*.

compiler directive

An instruction to the *Verilog* compiler. Typically used to define library cells or define a macro which controls conditional compilation. Also used to include specified Verilog file or define the simulation time units. The directive is effective from the place it appears in the Verilog code until it is superseded or reset.

complete transition path

The sequence of one or more *partial transitions* going from one *state* to another state (or itself) in a *state machine*. The *conditions* in the transition path are the collection of all the conditions on the individual *transitions*. The *action* in the transition path are the collection of all the actions on the individual transitions plus the actions of the origin state. When tracing the transition path, *links* are resolved to the referenced *start state*, state or *junction*. See also *partial transition*.

component

A *design unit* that contains a re-usable functional object definition or the instantiation of this object on a *block diagram* or *IBD view*. A component has a fixed interface and may be defined by a *child block diagram*, *IBD view*, *state diagram*, *flow chart*, *truth table*, *ModuleWare*, *HDL text*, *external HDL* or *foreign view*. See also *embedded block*, *block* and *port map frame*.

component browser

The component browser is a separate floating window which can be used to browse for *components* available in the current *library mapping*. Components can be instantiated in an editor view by copy and paste or drag and drop.

concurrent events

Occurrence of two or more events in the same clock cycle.

concurrent statements

Statements which can be entered in a *state diagram*, *flow chart* or *truth table* and are included in the generated HDL at the end of the *VHDL architecture* or *Verilog module*. Concurrent statements are applied to all diagrams in a set of concurrent state machines.

condition

A condition in a *state machine* is a boolean input expression which conforms to *HDL* syntax, and when it evaluates to TRUE, causes a *transition* to occur. The expression usually consists of a *signal* name, a relational operator and a value. In a flow chart, conditions are used in a *decision box* to determine which output *flow* is followed. In a *truth table*, conditions are generated from

the values assigned to a variable in an input column or can be explicitly entered as additional conditions in an unnamed input column. See also *transition priority*.

configuration

A definition of the *design unit views* that collectively describe a design by listing the included VHDL entities and architectures. A configuration may also include specification of the values for *VHDL generics* associated with *components* in the design. See also *VHDL configuration*.

connectable item

A *node* in a *block diagram*, *flow chart* or *state diagram* that can be the *source* or *destination* of a *signal*, *flow* or *transition*.

current view

The *design unit view* of a *block* or *component* that is currently used. This will be the *default view* unless a loaded *configuration* specifies otherwise.

— D —

decision box

A named object on a *flow chart* or *ASM chart* containing a *condition*. Each decision box has one input *flow* and two output flows (corresponding to the TRUE and FALSE conditions for an IF statement). See also *action box*, *case box* and *wait box*.

default view

The *design unit view* used in hierarchical operations, open commands and HDL generation (unless a loaded *configuration* specifies otherwise). See also *current view*.

design data library

A repository within a *library* containing source design data objects. There are usually different *library mappings* for *graphical editor* or *HDL text* source views. See also *compiled library*.

design explorer

The *source browser* design explorer windows can be used to browse the content and hierarchy of the source design data using user-defined *viewpoints* displayed in tree or list format.

design manager

The main *HDL Designer Series* window which is used for library management, data exploration, design flow and version control. The design manager includes a *shortcut bar*, *project manager*, *design explorer*, *side data browser*, *downstream browser*, *task manager* and *template manager*.

design unit

A subdirectory within a *design data library* which is represented by an icon in the *design explorer*. Design units may be *blocks*, *components* or *unknown design units*.

design unit view

A description of a *design unit*. Multiple views of *block* or *component* design units can describe alternative implementations. These can include *block diagram*, *IBD view*, *state diagram*, *flow chart*, *truth table* or *HDL text* views.

DesignPad

The built-in *VHDL* and *Verilog* sensitive editor and viewer for *HDL text* views.

destination

The *connectable item* at the end of a *signal*, *transition* or *flow*. See also *source*.

diagram browser

The diagram browser is an optional sub-window which displays the structure and content of the active *diagram editor* view.

diagram editor

An editable *block diagram*, *state diagram*, *flow chart* or *symbol* window which represents a *design unit view* using graphical objects. See also *graphical editor* and *table editor*.

don't touch

A control placed on a *design unit* or *design unit view* which disables specified downstream operations. See also *black box*.

downstream browser

The downstream browser displays the contents of the *compiled library* for the *design data library* currently open in the active *design explorer*. See also *source browser*, *side data browser* and *resource browser*.

downstream only library

A *library* which has *library mappings* defined only for downstream compiled data.

— E —**embedded block**

The representation of an *embedded view* on a *block diagram* or *IBD view* which has a dynamic interface defined by the *signals* connected to it but unlike a *block* or *component* does not add hierarchy to the design.

embedded view

An embedded view describes concurrent HDL statements on a *block diagram* or *IBD view* and is represented by an *embedded block* which can be defined by a *state diagram*, *flow chart*, *truth table* or *HDL text*.

enable point

An object on an *ASM chart* which displays an enable *signal* name and *condition*. See also *clock point* and *reset point*.

end point

A *flow chart* must have at least one end point which is always named *end*. See also *start point*.

entity declarations

User-specified *VHDL* statements which can be entered as properties in a *symbol* and are added to the corresponding *VHDL entity* declarations in the generated HDL. See also *architecture declarations* and *process declarations*.

entry point

A connector on a *child state diagram* which connects to a *source* in the *parent* state diagram. See also *exit point*.

exit point

A connector on a *child state diagram* which connects to a *destination* in the *parent* state diagram. See also *entry point*.

explicit clock

A *net* on a *block diagram* or *IBD view* which is used as a clock *signal* by the instantiated views of *blocks*, *embedded blocks* or *components*. See also *clocking*.

external HDL

A *HDL* description which was not created by a HDL Designer Series tool (for example, user-written *VHDL* or *Verilog*, gate-level HDL models created by synthesis, Inventra, FPGA or 3Soft core models). A port interface must exist for the referenced model as a *VHDL entity* or *Verilog module*. See also *HDL view* and *foreign view*.

— F —**flow**

An orthogonal line connecting objects on a *flow chart*. A flow can end on another flow (by creating a *flow join*) but cannot start from a flow.

flow chart

A *diagram editor* view which represents a process in terms of *action boxes*, *case boxes*, *decision boxes*, *wait boxes* and *loops* connected by *flows*. A flow chart must also contain one *start point* and one or more *end points*.

flow join

A connection between *flows* shown as a solid dot where the flows meet.

foreign view

A non-*HDL* description (for example, a C or C++ view) with a registered file type which requires an external HDL generator. See also *external HDL*.

formal

A *signal* or *bus* associated with a *port* on a *component*. Typically, a formal port is connected to an actual signal or bus on the *parent* view which has the same properties but may have a different name. Formal ports and actual signals with different properties can be connected using a *port map frame*.

FPGA

FPGA stands for Field Programmable Gate Array.

functional primitive

A *block* or *component* that is not further decomposed but fully defined by its own views. However, there may be both a *block diagram* or *IBD view* which describes its behavior in terms of lower level blocks or components and, for example, a *HDL text* view which fully defines its behavior. In this case, the *current view* determines whether the block or component is a functional primitive.

— G —

generate frame

An optional outline which can be used to replicate structure using a FOR frame or conditionally include structure using an IF frame (and ELSE frame in Verilog). Also used in VHDL to cluster concurrent objects using a BLOCK frame.

global actions

Explicit *action* in a *state diagram* or *truth table* which are always performed. In a state machine, global actions are executed on registered signals at an active clock edge or concurrently at a *transition* event on unregistered signals and are used to ensure that default output values are assigned for transitions with no explicit actions defined. See also *state actions* and *transition actions*.

global connector

Any *signal*, *bus*, or *bundle* connected to a global connector is considered to be connected (as an input) to every *block* in the *block diagram* or *IBD view*. It is typically used to connect clock or reset signals.

global net

A global net is a *signal* which can be used on a *block diagram* or *IBD view* but is declared externally in a *VHDL package* or *Verilog include* file. A global net can not be connected to a *block*, external *port* or *global connector*.

graphical editor

An editable window which displays a *diagram editor* or *table editor* view of a *design unit*. See *block diagram*, *IBD view*, *state diagram*, *flow chart*, *symbol*, *truth table* and *tabular IO*.

— H —

HDL

HDL stands for Hardware Description Language and is used in the documentation as a generic term for the *VHDL* or *Verilog* languages. It may also refer to any other language (for example, C) which is being used to describe the behavior of hardware.

HDL2Graphics

HDL2Graphics is a utility program used by *HDL Designer Series* tools to create graphical *block diagram*, *state diagram*, *flow chart* or *IBD view* from source *VHDL* or *Verilog* code.

HDL Author

HDL Author is an advanced environment for *HDL* design which supports design management, HDL text editing using the integrated *DesignPad* text editor, re-usable *ModuleWare* library, version management, and downstream tool interfaces. HDL Author includes *graphical editors* for maintaining the structure of a design as graphical *block diagram* or *IBD views* and a *symbol* or *tabular IO* editor for editing *design unit* interfaces. It also includes editors for *state diagram*, *flow chart*, *truth table*, *symbol* and *tabular IO* views which allow an entire design to be represented graphically. A simulation analyzer interface supports error cross-referencing and animation facilities to assist with design de-bug operations.

HDL Designer

The HDL Designer tool includes all the facilities provided by the *HDL Author* tool plus *HDL2Graphics* import which can automatically create editable diagrams from imported HDL code. HDL Designer supports the creation of *block diagram*, *state diagram*, *flow chart* and *IBD views*.

HDL Designer Series

The HDL Designer Series (HDS) is a family of tools for electronic system design using the *VHDL* and *Verilog* hardware description languages. See also *HDL Detective*, *HDL Author* and *HDL Designer*.

HDL Detective

HDL Detective is the *HDL Designer Series* visualization tool which allows you to import any complete or partial HDL text based design and convert the design into a hierarchy of graphical views. The design structure can be represented as graphical *block diagrams* or *IBD views*. Primitive leaf-level views can be viewed as block diagram, *state diagram*, *flow chart* or *HDL text* views. A *design manager* can be used to explore the relationship between individual *design units*.

HDL text

A textual *HDL* description of a design object. A HDL text *design unit view* may contain structural HDL or define the behavior of a leaf-level *block* or *component design unit*. HDL text may also be used by an *embedded view* on a *block diagram* or *IBD view* to contain concurrent HDL statements which are included in the generated structural code. See also *HDL view*.

HDL text editor

The tool used to edit or view *HDL text* views. The *HDL Designer Series* tools are initially configured to use the built-in *DesignPad* editor but can be set to use many other popular editors.

HDL view

A *design unit view* defined by structural or behavioral *HDL text*. See *Verilog module*, *VHDL entity* and *VHDL architecture*. Also the *VHDL package header* and *VHDL package body* views of a *VHDL package*.

HDM

The Hierarchical Data Model is the internal representation of design data used by the *HDL Designer Series* which allows design objects to be located anywhere in the hierarchy below a physical directory specified in the *library mapping*.

hierarchical action box

The representation on a *flow chart* or *ASM chart* of an embedded *child* diagram which describes *action* logic. See also *action box*.

hierarchical state

The representation on a *state diagram* of an embedded *child* diagram which describes state transitions. See also *simple state*.

hierarchical state box

The representation on an *ASM chart* of an embedded *child* diagram which describes state transitions. See also *state box*.



IBD view

A *design unit view* described using *Interface-Based Design* which represents the interfaces between instantiated *blocks*, *embedded blocks* and *components* as one or more *interconnect tables* showing the *signal* connections between them. See also *block diagram*.

if decode box

A named object which represents an IF statement on an *ASM chart*. When used for decoding action logic each If has an associated End If object. An if decode box has one input *flow* and one or more output flows each corresponding to an evaluated conditional expression. See also *action box*, *case box*, *decision box* and *wait box*.

interconnect cell

A cell at the intersection of a row and a column in an *IBD view*. The interconnect cells specify *ports* connecting *signals* or *buses* (defined by the rows) and *blocks*, *embedded blocks*, *components*, *external HDL* or *ModuleWare* instances (defined by the columns).

interconnect table

A *table editor* view which represents the connections between one or more *blocks*, *embedded blocks*, *components* or *ModuleWare* instances in an *IBD view*. May be abbreviated as ICT.

Interface-Based Design

A methodology which defines the structure of a design in terms of the interfaces between lower level *blocks* and *components*. See also *IBD view*.

interrupt condition

A *condition* associated with a *transition* from an *interrupt point* which applies to every *state* in the *state diagram* and has a higher *transition priority* than any other transitions.

interrupt point

A *node* on a *state diagram* or *ASM chart* that is implicitly connected to all *states* on the same diagram. Any *transition* from an interrupt point is treated as an *interrupt condition* from every other state in the diagram. A transition from an interrupt point in the top level diagram is treated as global interrupt condition and applies to all states in a hierarchical state machine. See also *junction* and *entry point*.

**junction**

A connector on a *state diagram* that enables a set of *transitions* between *states* to be replaced by a simpler set of *partial transitions* between the same states. See also *interrupt point* and *entry point*. Also used for a *net connector* joining two *nets* with the same properties on a *block diagram*.



No entries

**leaf view**

An undefined view of a *block* which has been added on a *block diagram* or *IBD view* but has not been defined by a *design unit view*.

library

A repository for source design data and compiled objects that has been assigned a logical name. See also *library mapping*, *regular library*, *protected library* and *downstream only library*.

library mapping

The mapping of a logical *library* name to physical locations. There are typically different mappings for the *design data library* containing *graphical editor* and *HDL text* source views and the *compiled library* containing downstream objects.

link

A connector used on a *state diagram* or *ASM chart* (or between *child* diagrams in the same hierarchy) to avoid long *transition arcs* or *flows*. A link is implicitly connected to the *state* or *junction* (on a *state diagram*) or to the *state box* (on an *ASM chart*) with the specified name. See also *exit point*.

local declarations

User-specified *Verilog* statements which can be entered as properties for a *flow chart* or *truth table*. These statements are declared at the top of the *always* code in the generated HDL for a truth table. When concurrent flow charts are defined, these declarations are local to each of the individual concurrent flow charts and you can choose whether they are inserted in the *initial* or *always* code. See also *module declarations*.

loop

A loop on a *flow chart* is defined by a start loop and stop loop object connected by a *flow*. A loop is used to repeat a set of sequential statements and can have *Repeat*, *For*, *While* or *Unconditional* control properties.

LPM

A library of parameterizable modules which can be instantiated as *components*. to implement common gate, arithmetic, storage or pad functions.

— M —

Mealy notation

A Mealy notation *state machine* is defined as a sequential network whose output is a function of both the present *state* and the inputs to the network (*conditions*). In Mealy notation, outputs (*action*) are associated with the *transitions* between states. See also *Moore notation* and *transition actions*.

module declarations

Locally defined *Verilog* statements which can be entered as properties in a *state diagram*, *flow chart*, *truth table* or *symbol* and are declared for the corresponding *Verilog module* in the generated HDL. Module declarations are typically used for 'define, parameter, reg, integer, real, time or wire declarations. See also *local declarations*.

ModuleWare

A library of technology-independent, synthesis-optimized *HDL* generators which can be used to implement many common logic, constant, combinatorial, bit manipulation, arithmetic, register, sequential, memory or primitive functions as instantiated *VHDL* or *Verilog* models.

Moore notation

A Moore notation *state machine* is defined as a sequential network whose outputs (*action*) are a function of the present *state* only. In Moore notation, actions are associated with the states. See also *Mealy notation* and *state actions*.

— N —

net

A set of *signals* or *buses* which have the same name and *type*. The net represents connections between objects in the design structure and has a value determined by the net's drivers. See also *wire*.

netlist

An ASCII representation of a circuit that lists all of the content of a design and shows how they are interconnected. Typically used for a gate level description as the input to a simulator or place and route tool.

net connector

A net connector can be used on a *block diagram* to join *nets* which have the same properties. It can also be used as an implicit on-page connector between nets with the same properties on the

same diagram or as a dangling connector to terminate nets which are left deliberately unconnected. See also *global connector*, *junction* and *ripper*.

node

A *connectable item* on a *block diagram*, *state diagram*, *ASM chart* or *flow chart*. On a block diagram, it can be a *block*, *embedded block*, *component*, *port map frame*, *global connector*, *port*, *ripper* or *net connector*. On a state diagram, it can be a *state*, *start state*, *hierarchical state*, *junction*, *interrupt point*, *link* and an *entry point* or *exit point* in a *child* hierarchical state diagram. On a flow chart, it can be a *start point*, *action box*, *loop*, *decision box*, *case box*, *wait box* or *end point*.



object

A general term used for a selectable item or selectable group of closely related items.

object tip

A popup window which displays information about the object under the cursor.



package list

A list of *VHDL packages* referenced by a *design unit view*. The package list is displayed as a text object on a *block diagram*, *state diagram*, *flow chart* or *symbol*.

panel

A defined and named area on a *block diagram*, *flow chart*, *state diagram* or *symbol* which facilitates viewing or printing the area.

parent

The view immediately above its *child* in the design hierarchy. A *design unit view* appears as a *block* or *component* on its parent *block diagram* or *IBD view*. Also used for the view containing the *hierarchical state* or *hierarchical action box* or *hierarchical state box* representing a hierarchical *state diagram*, *ASM chart* or *flow chart*.

partial condition

The *condition* associated with a *partial transition*.

partial transition

Any *transition* arriving at or leaving a *junction* or *interrupt point* on a *state diagram*. Also the transitions connected to an *entry point* or *exit point* in a *child* hierarchical state diagram. See also *complete transition path*.

polyline

A series of connected straight lines joining one or more points. Polylines may be orthogonal (horizontal and vertical lines only) or may include diagonals. See also *spline*.

port

The external connections for a *design unit* and their representation on a *symbol*, *tabular IO*, *block diagram* or *IBD view*. Also the connections to an instantiated *block*, *embedded block* or *component* on a block diagram or IBD view. The *signals* connected to *ports* may be inputs, outputs or bidirectional or (for VHDL) buffered. The connection points on objects in an *ASM chart* or *flow chart* are also described as ports.

port map frame

An optional outline around a *component* on a *block diagram* which allows mapping between actual *signals* on a *block diagram* and *formal ports* which have different properties.

probe

A probe is a text object which can be used to monitor the simulation activity of a *signal* on a *block diagram*. Although a probe can be moved independently, it is permanently attached to its associated signal by an *anchor*.

process declarations

User-specified *VHDL* statements which can be entered on a *flow chart*, *state machine* or *truth table* and are included at the beginning of the corresponding process in the generated HDL. When concurrent flow charts are defined, these declarations are local to each of the individual concurrent flow charts. See also *entity declarations* and *architecture declarations*.

project

The collection of *library mapping* information that the *HDL Designer Series* uses to locate and manage your designs.

project manager

The *source browser* project manager window can be used to set up a *project* and to define, load and configure the *library mapping* for your designs.

protected library

A *library* containing re-usable objects (such as standard VHDL type definitions or shared components) which cannot be edited, generated or compiled.

properties

A mechanism for storing additional information in the data model.

PSL

PSL is a Property Specification Language for the verification of *VHDL* or *Verilog* RTL designs.

— Q —

No entries

— R —**range**

The maximum and minimum *bounds* for an integer, floating, physical or enumeration *type*.

recovery state point

A *node* on an *ASM chart* that indicates the *flow* to the recovery *state* used when there is no other valid state assignment.

registered signal

A *signal* in a *state machine* whose value is held as an internal signal which is then assigned to the output port by the clocked process. A default value should be specified to avoid creating latches during synthesis. See also *combinatorial signal* and *clocked signal*.

regular expression

A regular expression is a pattern to be matched against a text string. When found, a string which matches the expression can optionally be replaced by another text string.

regular library

A *library* used for design creation which has *library mappings* for graphical and HDL text source design objects.

re-level

An operation available in the *state diagram* editor to add or remove hierarchy by moving *states* into or from a *child* diagram which is represented by a *hierarchical state* on the *parent* diagram.

requirement traceability

The process of tracking a requirement through a design to ensure that it is satisfied.

reset point

A *node* on an *ASM chart* that displays the reset *signal* name and *condition*. See also *clock point* and *enable point*.

resource browser

The resource browser provides a *task manager* for configuring and invoking tasks and a *template manager* for maintaining templates. See also *source browser*, *side data browser* and *downstream browser*.

ripper

A ripper can be used on a *block diagram* to split or combine *nets* which have the same name and *bounds* but represent a different *slice* or element. It can also be used to add or remove nets from a *bundle*. See also *net connector*.

route point

One of a series of points specifying the path of a *net* in a *block diagram* (or a *transition arc* in a *state diagram*). Route points can be connected using *polylines* or *splines*.

— S —**selection set**

A set of selected objects which are acted on by subsequent operations.

sensitivity list

A list of signals which can be entered in a *flow chart* or *truth table* and are used as the sensitivity list in the generated HDL. The signals defined in the sensitivity list cause the corresponding process to execute when any of the signals changes.

shortcut bar

A customizable control panel which provides shortcuts to *viewpoints*, *tasks* and *ModuleWare components*.

shortcut key

A keyboard key or key combination that invokes a particular command (also referred to as an accelerator key. See also *toolbar*.

side data

Supplementary source design data (such as EDIF, SDF and document header files) or user data (such as design documents or text files) which is saved with a *design unit view* and can be viewed using the *side data browser*.

side data browser

The *side data* browser displays an expandable indented list showing design and user data associated with the *design unit view* selected in the *design explorer*. See also *source browser*, *resource browser* and *downstream browser*.

signal

A connection or transfer of information between *blocks* or *components* which is represented as a *polyline* or *spline* (with a name and *type*) on a *block diagram*. A set of signals with the same name is called a *net*. See also *bus*.

signals status

A list of the output and locally declared signals in a *state machine* or *ASM chart* which shows the *type* (*VHDL* only), scope (output or local), default value, reset value and status (combinatorial, registered or clocked).

simple state

The representation on a *state diagram* of a *state* which has no *child* state diagram. See also *hierarchical state* and *wait state*.

slice

A slice is used to access a set of contiguous elements within an array type (such as *std_logic_vector*). The left and right limits of the slice must be consistent with the *bounds* of the object.

source

Source design data contained in a *library* as *graphical editor* or *HDL text* views. Also the *connectable item* at the start of a *signal*, *bus*, *transition* or *flow* on a *diagram editor* view. See also *destination*.

source browser

The source browser provides a *project manager* window and any number of *design explorers* for browsing *source* design objects. See also *side data browser*, *resource browser* and *downstream browser*.

spline

A curved line connecting two or more points. See also *polyline*.

start point

There is one and only one start point in a *flow chart* which is always named *start*. See also *end point*.

start state

The initial *state* of a *state machine*. The start state represents the status of the state machine before any *transitions* occur.

state

A state is a resting mode of a *state machine*. Also the representation of a state on a *state diagram*. Encoding information is shown if manual encoding is enabled and the state may have associated *actions*. See also *hierarchical state*, *simple state*, *start state*, *wait state*, *transition* and *condition*.

state actions

The *actions* associated with a *state* on a *state diagram* which are executed when the state is entered. See also *transition actions* and *global actions*.

state box

A state box is the representation of a *state* on an *ASM chart*. A state box may have associated entry, state and exit *actions*. See also *hierarchical state*.

state diagram

A *diagram editor* representation of a *state machine*. A state diagram typically consists of a number of *states*, *junctions*, *interrupt points* or *links* connected by *transitions*. The diagram may also include text blocks containing *global actions*, *concurrent statements*, *local declarations* and *comment text*. A hierarchical state machine may also include *hierarchical states*, *entry points* and *exit points*.

state machine

A *design unit view* of a *block* or *component* which defines its behavior in terms of a finite state machine (FSM). This is a mathematical model of a system. The system is represented by a finite number of *states* with a finite number of associated *transitions* between pairs of states. The state machine is represented graphically as a *state diagram*. State machines drawn using *Mealy notation* and *Moore notation* or a mixture of Mealy and Moore notation are supported.

state register statements

User entered statements which can be entered in a *state diagram* and are included in the generated HDL to replace the default state assignment for the *state machine* before the state decode statements at the beginning of a *VHDL* process or *Verilog* always code.

state variable

The name of a *signal* whose value that defines the current *state* of a *state machine*.

status bar

An area at the bottom of the *design manager*, *HDL text editor* or *graphical editor* window that displays information about the current command.

subtree

All objects directly or indirectly below a given object in the design hierarchy.

symbol

A *diagram editor* view which uses graphical objects to define the signal interface of a *component* and its representation when the component is instantiated on a *block diagram*. See also *tabular IO*.

synchronous

A synchronous process is activated on the next explicit clock edge rather than being activated only if any of its inputs are changed. See also *clocking*.

synthesis

The automatic generation of ASIC, FPGA or CPLD designs (circuits) from *HDL* descriptions.

system

Something that performs a specific function or set of functions with defined inputs and outputs. Typically, a self-contained electronic subsystem.

— T —**table editor**

An editable *truth table*, *IBD view* or *tabular IO* window which represents a *design unit view* using a tabular matrix of cells. See also *diagram editor* and *graphical editor*.

tabular IO

An alternative *table editor* view showing the interface of a *symbol*.

task

A customizable downstream tool or design flow which can be configured and invoked using the *task manager*.

task manager

The task manager window can be used to create, modify or run a *task*.

template manager

The template manager window can be used to create and modify the templates used for new *graphical editor* or *HDL text* views.

test bench

A test harness which allows a standard set of stimuli to be applied to a design.

toolbar

A group of buttons which provide shortcuts to commonly used commands. The *HDL Designer Series design manager* and *graphical editor* windows typically have several undockable toolbars each supporting a set of related commands. See also *shortcut key*.

tooltip

A small pop-up window that provides descriptive text for a *toolbar* button.

top-down design

The process of designing a system by identifying its major parts, decomposing them into lower level blocks and repeating the process until the desired level of detail is achieved. In electronic design automation, this process is applied to the top-down design of ASIC, FPGA and CPLD circuits using a hardware description language such as *VHDL* or *Verilog*. See also *bottom-up design*.

transition

A change of state within a *state machine*. The transition occurs when an associated *condition* is satisfied. A transition may have associated *transition actions* which are executed when the transition takes place. A transition is represented by a *transition arc* with associated *transition text* in a *state diagram*. See also *transition priority*.

transition actions

The *action* associated with a *transition* in a *state machine* which are executed when the transition occurs. A transition action is the consequence of a *condition*. See also *state actions*.

transition arc

A *polyline* or *spline* representing part of a *transition* between *states* on a *state diagram*. The direction of the transition is normally shown by an arrow head at its *destination* and the *transition text* is attached to the arc by an *anchor*.

transition order

The order in which CASE style *transitions* leaving a *state* are generated. CASE style transitions in *VHDL* are mutually exclusive and the order is ignored but the order is significant in *Verilog* since the first match in the generated code is taken.

transition priority

When there are more than one IF style *transitions* leaving a *state*, the associated *conditions* are evaluated in the order of their priority. The transition priority is shown by an integer on the *transition arc* adjacent to the *source* state. However, a *transition* with the *condition* OTHERS is always evaluated last.

transition text

The *condition* text (in a *Moore notation transition*) or the *condition* and *action* text (in a *Mealy notation transition*) which is attached to the *transition arc* by an *anchor*.

truth table

A *table editor* view which represents one or more output signals by the logical state of one or more input signals.

type

Specifies the characteristics and allowed values of a *net*. In VHDL, all *signals*, *buses*, variables and constants have a specific *VHDL* type definition which is defined in a *package list*. In *Verilog*, a net may have *wire*, *tri*, *wor*, *trior*, *wand*, *triand*, *tri0*, *tri1*, *supply0*, *supply1*, *reg*, *triereg*, *real*, *integer*, *time* or *realtime* type. The values for a *bus* may also be limited by a *bounds* constraint.

— U —

universe

The total area available for a diagram.

unknown design unit

A *design unit* which is not defined as a *block*, *component* or *package list*.

unknown design unit view

A *design unit view* representing data that is not defined as a *graphical editor*, *HDL text* or other registered view. Typically contains a text description and is treated as a text view for open, print or other file operations.

user directory

On UNIX, this is the home directory used when you login which contains your startup files and is normally located by the HOME environment variable. On a PC, an application data directory is created when you use a tool for the first time. On Windows NT, this is created in the profiles directory. For example:

C:\Winnt\Profiles\<user>\Application Data\HDL Designer Series

On a Windows XP machine, the application data directory is located below the *Documents and Settings* directory. For example:

C:\Documents and Settings\<user>\Application Data\HDL Designer Series Typically, the user directory will contain your preferences and library mapping files unless you have explicitly saved these files in alternative locations.

— V —

Verilog

A hardware description language (compliant with IEEE standard 1364-1995) that can be used to design, model and simulate electronic circuits. Verilog is a registered trademark of Cadence Design Systems Inc. See also *HDL* and *VHDL*.

Verilog include

A *Verilog* file containing global declarations or other Verilog code which can be included by reference using the ``include` *compiler directive*.

Verilog module

A *design unit view* of a *block* or *component* which defines its behavior using *Verilog* source code.

Verilog module body

Describes the boundaries and content of a *Verilog* logic block in structural, dataflow and behavioral constructs.

Verilog parameter

A Verilog parameter is a constant value used to parameterize a *Verilog* design description. Verilog parameters are used in a similar way to *VHDL generics*.

VHDL

VHDL stands for VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. VHDL is a design and modelling language (compliant with IEEE standards 1076-1987, 1076-1993 and 1076-2002) which was specifically created to describe (in machine and human-readable form) the organization and function of digital hardware systems and circuit boards. See also *HDL* and *Verilog*.

VHDL architecture

A *design unit view* of a *block* or *component* which defines its behavior using *VHDL* source code.

VHDL architecture body

Declares the items available inside a *VHDL design entity* and specifies the relationships between inputs and outputs. An architecture body describes the organization and operations performed inside the design entity. You can choose to store the VHDL architecture body in the same file or in a separate file from the *VHDL entity*.

VHDL configuration

A declaration which specifies the *VHDL architecture body* used to define a *VHDL design entity*. See also *configuration*.

VHDL design entity

A VHDL design entity is the primary abstraction level of a *VHDL* hardware model which typically represents a cell, chip, board or subsystem. A VHDL design entity comprises a *VHDL entity* declaration and a *VHDL architecture body*.

VHDL entity

Declares the interface between a *VHDL design entity* and its external environment. An entity declaration contains definitions of inputs to and outputs from the VHDL design entity. VHDL entity declarations can optionally be stored in the same file or a separate file from the associated *VHDL architecture body*.

VHDL generic

A VHDL generic is a constant value used to parameterize a *VHDL* design description. VHDL generics are used in a similar way to *Verilog parameters*.

VHDL package

A *VHDL* object that contains procedural definitions and declarations used by *design unit views* of *blocks* or *components*. Typically contains *type* and subtype definitions. Usually comprises a separate *VHDL package header* containing declarations and a *VHDL package body* containing any functions or procedures declared in the package header.

VHDL package body

The part of a *VHDL package* which defines the implementation of objects in the package. It contains data used when the design is evaluated. The package body typically contains constant definitions and function bodies.

VHDL package header

The part of a *VHDL package* which declares the objects defined in the package. It is referenced by *block* and *component* views.

viewpoint

A set of user-defined rules which examine particular aspects of a design.

VITAL

VITAL stands for the *VHDL* Initiative Towards ASIC Libraries which is an IEEE standard (IEEE1076.4) for *ASIC* library design.

— W —

wait box

A named object on a *flow chart* containing a conditional wait statement which controls the delay before an event occurs on a signal in the *sensitivity list*. See also *action box*, *case box* and *decision box*.

wait state

A wait state has similar properties to a *simple state* but introduces a delay of two or more clock cycles.

whisker

A line that extends between a *port* on the boundary of a customized *block* or *component symbol* and the body of the block or component symbol.

wire

A segment of a *net* on a *VHDL* or *Verilog block diagram*. A wire may have *signal* or *bus* style and scalar or vector *type* and should not be confused with the Verilog wire type.

working directory

On UNIX, the directory from which you invoked the application. On a PC, the working directory defaults to the *user directory* or can be set using the **Start In** option when you define the properties for a short cut to your application. Do not set a working directory using the **Start In** shortcut option if you want to use object linking and embedding (OLE) to import objects into a documentation tool as the application will not be able to access library mapping information from this location.

workspace

A working environment which allows common design data to be shared between multiple users. Typically, a project comprises one or more shared workspaces and a private workspace (often described as a sandbox) for each engineer working on the project.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

— X —

Xdefaults

A set of resources which can be used to set the default display characteristics on X server window systems.

— Y —

No entries

— Z —

No entries

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

— A —

- Action box
 - object properties, 356
- Actions
 - action box, 346
 - syntax, 357
- Activity trail
 - setting, 434
- Adding Ports, 257
- Anchor
 - comment text, 57
 - simulation probe, 421
 - text object, 67
- Animation
 - activity trail, 434
 - cause, 430, 437
 - clear captured events, 434
 - data capture, 433
 - enabling, 433
 - flow chart, 431
 - global capture, 433
 - goto latest, 436
 - goto next, 436
 - goto previous, 436
 - goto start, 436
 - goto time, 436
 - graphical highlighting, 436
 - highlight colors, 436
 - instrument for, 433
 - link diagrams, 437
 - linking diagrams, 437
 - mixed language, 437
 - move by clocks, 436
 - move by events, 436
 - move by states, 436
 - notation, 436
 - preferences, 435
 - reviewing, 436
 - showing the animation view, 433

- state diagram, 431

- ASM chart
 - editor, 18
- Attributes
 - in port declarations, 165
 - in signal declarations, 165
 - setting, 165
- Autoshape
 - see comment graphics

— B —

- Backup
 - automatic, 31
 - file, 31
- Black box
 - setting, 44, 278
- Block
 - automatic instance name, 111
 - autowhiskers, 229
 - changing the shape, 228
 - customizing, 228
 - definition of, 202
 - instantiating, 111
 - open down, 194
 - renaming, 111
 - updating the interface, 128
- Block diagram
 - adding hierarchy, 172
 - automatic routing, 410
 - bus reconstruction, 412
 - changing the layout, 410
 - definition of, 200
 - editor, 18
 - layout and routing options, 409
 - notation, 201
 - preferences, 231
 - re-level, 172
 - text visibility, 70
 - toolbar, 211
- Breakpoints

- adding, [419](#)
- disabling, [420](#)
- enabling, [420](#)
- graphical, [420](#)
- removing, [419](#)
- reporting, [420](#)

Browser

- diagram, [91](#)
 - content, [94](#)
 - columns, [96](#)
 - grouping, [97](#)
 - sorting, [97](#)
- structure, [92](#)

Bundle

- adding on a block diagram, [217](#)
- adding signals, [218](#)
- definition of, [204](#)
- highlight color, [225](#)
- ripping a bundle, [219](#)
- ripping a bus, [219](#)
- ripping a signal, [218](#)

Bus

- adding on a block diagram, [213](#)
- definition of, [204](#)
- endpoint, [213](#)
- ripping a slice, [215](#)
- ripping an element, [215](#)

— C —

Case box

- expression
 - syntax, [364](#)
- object properties, [363](#)

Cell

- autofit, [90](#)
- copying contents, [88](#)
- cut and paste, [88](#)
- deleting contents, [88](#)
- editing, [88](#)
- resizing a column or row, [90](#)
- selecting, [88](#)

Changing the Display of Signal Properties, [208](#)

Column

- collapsing, [266](#)
- expanding, [266](#)
- moving, [267](#)

Command

- auto-repeat, [22](#)

Command Auto-repeat, [22](#)

Comment graphics

- adding, [72](#)
- adding a bitmap, [76](#)
- adding a circle, [76](#)
- adding a line, [74](#)
- adding a polygon, [75](#)
- adding a polyline, [74](#)
- adding a rectangle, [75](#)
- adding an arc, [75](#)
- adding an ellipse, [76](#)
- adding arrowheads, [74](#)
- autoshape, [229](#)
- edit vertices, [72](#)
- grouping, [55](#)
- layering, [55](#)
- loading a bitmap, [76](#)

Comment text

- adding, [56](#)
- after file header, [57](#)
- after object, [57](#), [104](#), [163](#), [322](#)
- at file end, [57](#)
- at file start, [57](#)
- attaching, [57](#)
- before object, [57](#), [104](#), [163](#), [322](#)
- detaching, [57](#)
- editing, [57](#), [58](#)
- end of line, [57](#), [104](#), [163](#), [322](#)
- formatting, [58](#)
- grouping, [55](#)
- hiding, [58](#)
- in block interface, [57](#)
- in port declarations, [163](#), [321](#)
- in signal declarations, [163](#)
- including in HDL, [57](#)
- internal variables, [58](#)
- Kanji, [58](#)
- layering, [55](#)
- object properties, [58](#)
- properties, [58](#)
- showing, [58](#)

Compiler directive

- recovery, [397](#)

- setting, [26](#)
- Component
 - automatic instance name, [111](#)
 - definition of, [202](#)
 - drag and drop, [112](#)
 - instantiating, [112](#)
 - open down, [194](#)
 - updating an instance, [127](#)
 - where used, [320](#)
- Component browser
 - instantiating a component, [112](#)
 - instantiating a ModuleWare part, [116](#)
- Condition
 - syntax, [360](#)
- Constrained
 - move, [52](#)
 - resize, [53](#)
- Copy
 - object, [34](#)
 - picture, [34](#)
 - to clipboard, [34](#)
- Cursor tracking
 - probes, [422](#), [430](#)
 - state machine, [418](#), [422](#), [430](#)
- CVE models
 - source VHDL, [121](#)
- D —
- Dataflow window
 - displaying, [427](#)
- Decision box
 - object properties, [358](#)
- Declarations
 - align in columns, [324](#)
 - entity, [324](#)
 - external module declarations, [324](#)
 - internal module declarations, [324](#)
 - module, [324](#)
 - symbol, [324](#)
 - Verilog parameter, [322](#)
 - Verilog signals, [152](#)
 - VHDL generic, [322](#)
 - VHDL signals, [150](#)
- Default view
 - setting, [196](#)
- DesignPad
 - HDL text editor, [18](#)
- Diagram
 - saving, [29](#)
- Diagram editor
 - default font, [48](#)
 - template title block, [49](#)
 - visible anchors, [49](#)
- Dialog box
 - Activity Trail Settings, [435](#)
 - Add External IP, [122](#)
 - Add Hierarchy, [173](#)
 - Add Signal Stubs, [217](#), [221](#)
 - Attributes, [165](#)
 - Block Diagram Layout and Routing
 - Options, [409](#)
 - Block Diagram Master Preferences, [231](#), [281](#)
 - Block Diagram Preferences, [232](#), [281](#)
 - Appearance, [243](#), [282](#)
 - Background, [244](#)
 - Default Values, [232](#), [233](#)
 - Default Properties, [235](#)
 - Verilog, [234](#)
 - VHDL, [234](#)
 - Display Settings, [239](#)
 - ModuleWare Display, [241](#)
 - ModuleWare Params, [236](#)
 - Object Visibility, [240](#)
 - Verilog PortIO, [241](#)
 - Verilog Signals, [241](#)
 - VHDL PortIO, [241](#)
 - VHDL Signals, [241](#)
- Cell Edit Appearance, [314](#)
- Choose Instances, [424](#)
- Choose Panel to Delete, [81](#)
- Choose Panel to Show, [80](#)
- Choose Panel to View, [80](#)
- Choose Shape, [229](#)
- Color Selection, [83](#), [85](#)
- Comments, [104](#), [163](#), [322](#)
- Convert To Graphics, [402](#)
- Create Embedded View, [125](#)
- Create Test Bench, [440](#)
- Design Checking Options, [131](#)
- Diagram Master Preferences

- Background, [50](#)
- Documentation and Visualization Options, [405](#)
- Edit Appearance, [83](#)
- File Creation Wizard, [195](#)
- Filter Controls, [310](#)
- Filter Settings, [182](#)
- Find and Replace
 - Find, [35](#)
 - Replace, [37](#)
- Flow Chart Preferences, [373](#)
 - Appearance, [373](#)
 - Background, [376](#)
 - Default Values, [375](#)
 - Default Properties, [375](#)
 - Miscellaneous, [374](#)
 - Object Visibility, [376](#)
- Flow Chart Properties, [364](#)
 - Architecture Declarations, [369](#)
 - Concurrent Statements, [370](#)
 - Generation, [365](#)
 - Module Declarations, [369](#)
 - Process Declarations, [372](#)
- Font Select, [83](#)
- Found, [37](#), [38](#)
- Frame Declarations, [302](#)
- Generation File Clash, [43](#)
- Interface Master Preferences, [327](#)
- Interface Preferences
 - Default Values, [331](#)
 - Default Properties, [333](#)
 - Verilog, [332](#)
 - VHDL, [332](#)
- Interface Appearance, [329](#)
- Miscellaneous
 - Interface, [334](#)
 - Object Visibility, [335](#)
 - Verilog Port Display, [335](#)
 - VHDL Port Display, [335](#)
- Symbol Appearance, [330](#)
- Main Settings
 - Diagrams, [48](#), [67](#), [77](#), [78](#)
 - General, [22](#), [23](#)
 - Save, [31](#)
 - Tables, [87](#)
- ModuleWare Object Tips Visibility, [88](#)
- ModuleWare Parameter Visibility, [120](#)
- ModuleWare Preview, [118](#)
- Name Block, [195](#)
- Net Highlighting Options, [226](#)
- Net Insert Options, [169](#)
- Net Insert/Remove Parameters, [170](#)
- Net Propagation Options, [166](#)
- Net Remove Options, [171](#)
- Object Properties, [32](#)
 - Action Boxes, [357](#), [358](#), [360](#), [362](#), [363](#)
 - Components, [133](#)
 - Embedded Blocks, [143](#)
 - Frames, [302](#)
 - Text, [58](#)
- Open As (rendered view), [429](#)
- Package List, [24](#)
- Panel Object Properties, [79](#)
- Port Display Control, [207](#)
- Port Map Settings, [284](#)
- PortIO Display Control, [206](#)
- Probe Properties, [423](#)
- Reconcile Interface, [129](#)
- Reconcile Interface Options, [130](#)
- Rename, [355](#)
- Rip Bus/Slice From Bundle, [219](#)
- Rip Element From Bus, [215](#)
- Rip New Bundle, [219](#)
- Rip Signal/Element From Bundle, [218](#)
- Rip Slice From Bus, [215](#)
- Save As Design Unit View, [29](#)
- Show Columns, [105](#), [309](#)
- Signal Display Control, [209](#)
- Symbol Master Preferences, [327](#)
- Symbol Object Properties, [324](#)
 - Text, [58](#)
- Symbol Preferences
 - Background, [337](#)
 - Miscellaneous, [328](#)
- Symbol/Interface Properties
 - Declarations, [324](#)
 - Symbol, [325](#)
- Truth Table Preferences, [391](#)
 - Appearance, [392](#)
 - Default Properties, [393](#)

Truth Table Properties, 381
 Architecture Declarations, 386
 Concurrent Statements, 387
 Generation, 382
 Global Actions, 388
 Module Declarations, 386
 Process Declarations, 388
 Update Where Used, 321
 Update/Replace Foreign Component, 124
 Verilog Compiler Directives, 26
 Where Used wizard, 320

— E —

Embedded block
 adding, 124
 definition of, 203
 renaming, 125
 Embedded constraints
 setting, 165
 Embedded HDL text
 adding, 126
 editing, 58
 Embedded view, 203
 flow chart, 125
 HDL text, 125
 opening, 125
 state diagram, 125
 truth table, 125
 Environment variables
 CVE_HOME, 121
 Export
 comma separated value (CSV) file, 90
 tab separated value (TSV) file, 90
 table, 90
 External HDL
 instantiating, 120
 soft pathnames, 123
 updating or replacing, 124

— F —

Find
 class expression, 36
 match case, 36
 match word, 36
 regular expression, 36
 replace, 37

select all, 37
 select object, 37
 text, 35
 wrap search, 36

Flip

object, 55
 ripper, 216

Flow chart

adding a case box, 349
 adding a concurrent chart, 355
 adding a decision box, 347
 adding a flow, 351
 adding a loop, 348
 adding a start point, 345
 adding a wait box, 348
 adding an action box, 346
 adding an end point, 352
 adding objects, 343
 animation, 431
 automatic connection mode, 343
 automatic insertion mode, 344
 breaking out of a loop, 349
 concurrent, 354
 deleting a concurrent chart, 356
 editor, 18
 hierarchical, 352
 opening a concurrent chart, 355
 properties
 architecture declarations, 364, 369
 begin and end, 368
 clock, 367
 combinatorial, 366
 concurrent statements, 364, 370
 fork and join, 368
 generation characteristics, 365
 initial or always style code, 368
 instrument for animation, 368
 local declarations, 365, 372
 module declarations, 364, 369
 process declarations, 365, 372
 reset, 367
 sensitivity list, 365, 367
 sequential, 366
 renaming a concurrent chart, 355
 text visibility, 71

— G —

Generate frame

- adding, [288](#)
- BLOCK, [258](#), [259](#), [288](#), [296](#)
- block, [259](#), [296](#)
- declarations, [302](#)
- editing properties, [302](#)
- ELSE, [288](#)
- FOR, [288](#), [289](#), [298](#)
- IF, [288](#)
- nested, [298](#)
- object properties, [302](#)

Generic declarations

- editing, [322](#)

Global connector

- adding on a block diagram, [222](#)

Graphical views

- opening, [429](#)

Grid

- preferences, [51](#)
- snapping, [86](#)
- visibility, [86](#)

— H —

HDL

- bulk parsing, [43](#), [278](#)
- convert to graphics, [401](#)
- generating from graphical views, [41](#)
- setting the language, [23](#)
- show as graphics, [409](#)
- VHDL and Verilog, [23](#)
- viewing generated HDL, [44](#)

HDL2Graphics, [397](#)

- compiler directives, [397](#)
- flow chart recovery, [399](#)
- state machine recovery, [398](#), [399](#)
- structure recovery, [396](#)
- Verilog parameter, [398](#)

HDL2Graphics,compiler directives, [397](#)

Highlight

- color, [225](#)
- net, [225](#)

— I —

IBD view

- adding hierarchy, [172](#)

editor, [18](#)

- expanding and collapsing, [266](#)
- moving rows or columns, [267](#)
- remove hierarchy, [172](#)

Icons

- Animation toolbar, [432](#)
- Appearance toolbar, [84](#)
- Arrange Object toolbar, [53](#)
- Block Diagram Tools toolbar, [211](#)
- breakpoint, [420](#)
- Comment Graphics toolbar, [72](#)
- diagram browser
 - notation, [92](#), [93](#), [94](#), [95](#), [96](#), [98](#)
- Format text toolbar, [27](#)
- Simulation toolbar, [416](#)
- Standard toolbar, [20](#)
- Tabular IO toolbar, [310](#)

IF Frame

- using, [298](#)

Intellectual property

- instantiating, [120](#)

Interface

- enforce consistent case, [130](#)
- enforce consistent port ordering, [130](#)
- preferences, [327](#)
- reconciling, [128](#)

Internal variable

- in comment text, [58](#)

Inventra models

- source HDL, [121](#)

— K —

Kanji text

- in comment text, [58](#)

— L —

Library mapping

- standard packages, [23](#)

List window

- adding signals, [418](#)
- displaying, [427](#)

Log window

- Task log, [43](#), [278](#)

Logic function

- notation, [227](#)

Loop

object properties, 362
statement
syntax checking, 363

— M —

Mixed language

external HDL, 122
using, 196

ModelSim

Debug menu, 428
List window, 418, 419, 429
Main window, 428
Signals window, 419, 429
Source window, 419, 420, 428
Structure window, 419, 428
Wave window, 418, 419, 428

ModuleWare

default parameter visibility, 236
editing parameters, 117
instantiating, 116
object tips, 236
parameter visibility, 120
port polarity, 119, 231
resizing, 119
stimulus generators, 441

Mouse

strokes, 22

— N —

Net

adding a net slice on an IBD view, 257
adding on a block diagram, 212
connecting, 222
connecting to a block or component, 223
connecting to a port map frame, 225
highlight color, 225
highlighting, 225
inserting, 169
propagating, 169
propagating changes, 166
removing, 169
report if unconnected, 132
routing, 212

Net connector

change to ripper, 216

Notation

action box, 340
block, 201
block diagram, 201
BLOCK generate frame, 259, 296
bundle, 201
component, 201
decision box, 340
diagram browser
content
ASM chart, 96, 98
block diagram or IBD view, 95
flow chart, 95
state diagram, 96
structure
block diagram, 92
concurrent and hierarchy views, 92
symbol, 93
text objects, 94
ELSE generate frame, 259, 292, 295
embedded block, 201
end point, 341
FOR generate frame, 258, 289
global connector, 201
hierarchical action box, 340
IF generate frame, 259, 292
logic functions, 227
loop, 340
net connector, 201
port, 201
port map frame, 283
ripper, 201
start point, 340
symbol bidirectional port, 315
symbol buffer port, 315
symbol clock port, 315
symbol input port, 315
symbol inverted port, 315
symbol output port, 315
truth table, 379
wait box, 340

— O —

Object

aligning, 54
deleting, 34
Distributing, 54

- flipping, [54](#)
 - moving, [51](#)
 - moving to next grid point, [51](#)
 - resizing, [53](#)
 - rotating, [54](#)
 - selecting, [33](#)
 - selecting shapes, [33](#)
 - selecting text, [33](#)
 - Object properties
 - 2 dimensional bounds, [151](#)
 - 2 dimensional slice, [151](#)
 - action box, [356](#)
 - anchored panel, [79](#)
 - array bounds, [152](#)
 - attributes, [165](#)
 - block, [139](#)
 - block diagram, [133](#)
 - block port ordering, [140](#)
 - BUS keyword, [151](#)
 - case box, [363](#)
 - charge strength, [152](#)
 - comment bounding box, [58](#)
 - comment text, [58](#)
 - comments, [163](#)
 - component, [133](#)
 - decision box, [358](#)
 - editing, [32](#)
 - embedded block, [143](#)
 - embedded constraints, [165](#)
 - expansion, [152](#)
 - generate frame, [302](#)
 - IBD view, [133](#)
 - loop, [362](#)
 - net bounds, [151](#)
 - net type, [150](#)
 - non-autoroute panel, [79](#)
 - panel, [79](#)
 - panel visibility, [79](#)
 - Register keyword, [151](#)
 - regular panel, [79](#)
 - sheet panel, [79](#)
 - signal declarations, [150](#)
 - symbol, [324](#)
 - text, [58](#)
 - vector bounds, [152](#)
 - VHDL net declarations, [150](#)
 - wait box, [360](#)
 - Object tip
 - displaying, [78](#)
 - OLE
 - object linking and embedding, [38](#)
 - opening an OLE view, [41](#)
 - using drag and drop, [40](#)
 - Opening Block and Component Views, [194](#)
- ## — P —
- Panel
 - adding, [79](#)
 - anchored, [79](#)
 - deleting, [81](#)
 - displaying, [80](#)
 - hiding, [80](#)
 - non-autoroute, [79](#)
 - object properties, [79](#)
 - OLE, [41](#)
 - printing, [81](#)
 - protecting, [81](#)
 - sheet, [79](#)
 - showing, [80](#)
 - viewing, [80](#)
 - Panning
 - window, [86](#)
 - Parent view
 - editing, [28](#)
 - opening, [28](#)
 - Paste
 - here, [34](#)
 - object, [34](#)
 - special, [34](#)
 - Polyline
 - see comment graphics
 - Port
 - active high, [231](#)
 - active low (Not), [231](#)
 - adding in the signals table, [102](#)
 - adding in the symbol editor, [316](#)
 - adding in the tabular IO editor, [311](#)
 - adding on a block diagram, [220](#)
 - adding to a net, [221](#)
 - changing the mode, [221](#), [316](#)
 - display properties, [206](#)

- edge triggered (Clock), [231](#)
- falling edge clock, [231](#)
- mapping, [283](#)
- naming, [205](#)
- ordering, [319](#)
- polarity control, [231](#)
- propagating, [320](#)
- properties, [325](#)
- rising edge clock, [231](#)
- rotating, [222](#)
- spacing, [317](#)
- visibility, [230](#)
- Port map frame
 - editing the mapping, [284](#)
 - enabling, [283](#)
 - example, [286](#)
- Pragma
 - async_set_reset_local, [367](#), [385](#)
 - dc_script_begin, [166](#)
 - dc_script_end, [166](#)
 - hds, [396](#)
 - sync_set_reset_local, [367](#), [385](#)
 - synopsys full_case, [400](#)
 - synopsys parallel_case, [400](#)
 - translate_off, [396](#)
 - translate_on, [396](#)
- Preference
 - net width label, [204](#)
 - view type, [202](#)
- Preferences
 - animation, [435](#)
 - appearance, [243](#), [282](#)
 - applying master preferences, [50](#)
 - automatic completion in table cells, [87](#)
 - auto-update signal style, [239](#)
 - backup file, [31](#)
 - block diagram, [231](#)
 - block diagram default values, [233](#)
 - block diagram HDL types, [234](#)
 - bundle name, [233](#)
 - bus name, [233](#)
 - check syntax on entry, [328](#)
 - create component declarations, [44](#)
 - default font for diagram editor views, [48](#)
 - default font for table editor views, [87](#)
 - diagram background color, [244](#), [337](#), [376](#)
 - editing diagram preferences, [49](#)
 - editing master preferences, [49](#)
 - elaboration limit, [425](#)
 - embedded block name, [233](#)
 - generate in-line ModuleWare code, [232](#)
 - global connector name, [233](#)
 - grid, [244](#), [337](#), [376](#)
 - grid display, [51](#)
 - include title block in new diagrams, [49](#), [77](#)
 - instance name, [233](#)
 - interface appearance, [329](#)
 - interface default values, [331](#)
 - interface visual attributes, [329](#)
 - layout, [410](#)
 - net width label, [239](#)
 - object tips, [78](#)
 - open as symbol, [329](#)
 - open as tabular IO, [329](#)
 - port constraints, [332](#)
 - port display control, [241](#), [335](#)
 - port names, [331](#)
 - port ordering, [329](#)
 - reconcile enforce consistent case, [130](#)
 - reconcile enforce consistent port ordering, [130](#)
 - recovery file, [31](#)
 - routing, [410](#)
 - setting diagram background preferences, [50](#)
 - show anchors, [49](#), [67](#)
 - show signal attributes, [239](#)
 - signal constraints, [234](#)
 - signal display control, [241](#)
 - signal name, [233](#)
 - snap to grid, [51](#)
 - symbol appearance, [330](#)
 - symbol visual attributes, [330](#)
 - take display settings from component port, [239](#)
 - title block location, [49](#), [77](#)
 - update HDL view when symbol is saved, [32](#)
 - updating master preferences, [50](#)
 - use closest matched fonts, [49](#)

- use scalable fonts, [48](#)
- use symbol visual attributes for
 - components, [239](#)
- wrap bundle contents, [239](#)

Print

- panel, [81](#)

Probes

- adding, [421](#)
- removing, [421](#)
- setting properties, [423](#)
- tracking the ModelSim cursor, [422](#)

Process window

- displaying, [427](#)

— R —

Recovery

- file, [31](#)

Redo

- last undone command, [33](#)

Regular expression

- class expression, [36](#)
- searching for, [35](#)
- tagged expression, [38](#)

Replace

- see Find

Ripper

- change to net connector, [216](#)
- flip direction, [216](#)

Rotating

- objects, [54](#)
- text, [317](#)

Route point

- adding, [82](#)
- removing, [82](#)

Row

- collapsing, [266](#)
- expanding, [266](#)
- moving, [267](#)
- sorting in a tabular IO view, [311](#)
- sorting in the signals table, [107](#)

— S —

Save

- automatic, [31](#)

Seamless models

- see CVE models

Search

- see Find

Setting Background Preferences, [50](#)

Setting Compiler Directives, [26](#)

Setting Package References, [23](#)

Setting Preferences for Diagram Views, [48](#)

Setting Preferences for Table Views, [87](#)

Setting Visual Attributes, [83](#)

Shapes

- selecting, [33](#)

shared.hdp, [23](#)

Shortcuts

- in-line text editing, [65](#)
- mnemonic Keys, [22](#)

Signal

- adding in the signals table, [102](#)
- adding on a block diagram, [213](#)
- adding stubs on a block diagram, [217](#)
- adding the current state to simulator
 - windows, [418](#)
- adding to a bundle, [218](#)
- adding to simulator windows, [418](#)
- adding to the simulator log, [419](#)
- definition of, [204](#)
- display properties, [209](#)
- endpoint, [213](#)
- forcing, [423](#)
- highlighting in the simulator, [419](#)
- ordering, [171](#)
- reporting information, [419](#)
- rotating text, [222](#)

Signals table

- displaying, [99](#)
- filtering, [105](#)
- grouping, [105](#)
- notation, [100](#)

Signals window

- displaying, [427](#)

Simulation

- choosing the simulation instance, [424](#)
- continue, [426](#)
- driving, [418](#)
- forcing signals, [423](#)
- ModelSim cursor tracking, [418](#), [422](#), [430](#)
- removing probes, [422](#)

- reporting signal information, [419](#)
- restarting, [427](#)
- run for time, [426](#)
- run forever, [426](#)
- run to next event, [426](#)
- running, [426](#)
- step into line, [426](#)
- step over line, [426](#)
- step over object, [426](#)
- stepping, [426](#)
- Simulator environment
 - reporting, [426](#)
 - setting, [425](#)
- Simulator windows
 - displaying, [427](#)
- Source window
 - displaying, [427](#)
- speedCHART models
 - source HDL, [121](#)
- State diagram
 - animation, [431](#)
 - editor, [18](#)
 - text visibility, [71](#)
- Strokes
 - enabling, [49](#)
 - mouse shortcuts, [22](#)
- Structure window
 - displaying, [427](#)
- Symbol
 - autowhiskers, [317](#)
 - changing the shape, [317](#)
 - customize, [317](#)
 - editing declarations, [324](#)
 - lock, [317](#)
 - notation, [315](#)
 - text visibility, [70](#)
- Syntax
 - checking, [357](#), [360](#), [361](#), [363](#), [364](#)
- Synthesis
 - black box, [44](#), [278](#)
- T —
- Table
 - non-scrolling area, [308](#)
 - panning, [89](#)
 - scrolling, [89](#)
 - sorting rows, [107](#), [311](#)
- Table editor
 - default font, [87](#)
- Tabular IO
 - editor, [18](#)
 - filtering columns, [309](#)
 - grouping, [312](#)
 - hiding columns, [308](#)
 - VHDL range constraint format, [334](#)
- Test bench
 - analyzing results, [447](#)
 - creating, [440](#)
 - defining stimulus on a flow chart, [442](#)
 - defining stimulus on a state machine, [446](#)
 - defining stimulus using lookup tables, [443](#)
 - defining stimulus using textIO, [444](#)
 - definition of, [439](#)
 - generating a clock, [446](#)
 - re-using, [448](#)
- Text
 - alignment, [27](#)
 - anchor, [57](#), [67](#)
 - editing on a diagram, [64](#)
 - editor, [18](#)
 - embolden, [27](#)
 - finding, [35](#)
 - finish all edits, [67](#)
 - finish edits, [67](#)
 - font size, [27](#)
 - formatting, [27](#)
 - italicize, [27](#)
 - moving, [67](#)
 - pattern matching, [35](#)
 - regular expression, [35](#)
 - replacing, [37](#)
 - searching, [35](#)
 - selecting, [33](#)
 - send to editor, [66](#)
 - setting font, [83](#)
 - underline, [27](#)
 - visibility, [69](#)
- TextIO
 - defining stimulus, [444](#)
- The Diagram Browser, [91](#)
- Title block

- adding, [77](#)
- creating, [77](#)
- saving, [77](#)
- template, [77](#)

Toolbar

- Animation, [432](#)
- Appearance, [84](#)
- Arrange Object, [53](#)
- auto-repeat, [22](#)
- Block Diagram Tools, [211](#)
- Comment Graphics, [72](#)
- Format Text, [27](#)
- HDL Tools, [20](#)
- Simulation, [416](#)
- SM Signals Tools, [101](#)
- Standard, [20](#)
- Tabular IO Tools, [310](#)
- Tasks, [20](#)
- Version Management, [20](#)

Travel log

- window history, [29](#)

Truth table

- adding a column or row, [381](#)
- Case style, [389](#)
- comparison operators, [380](#)
- deleting a column or row, [381](#)
- editor, [19](#)
- If-Then-Else style, [389](#)
- notation, [379](#)
- preferences, [391](#)
- properties
 - architecture declarations, [382](#), [386](#)
 - clock, [384](#)
 - combinatorial, [383](#)
 - concurrent statements, [382](#)
 - generation, [381](#)
 - generation properties, [382](#)
 - global actions, [382](#), [388](#)
 - local declarations, [382](#), [387](#)
 - module declarations, [382](#), [386](#)
 - process declarations, [382](#), [387](#)
 - reset, [384](#)
 - sensitivity list, [385](#)
 - sequential, [383](#)

— U —

Undo

- last command, [32](#)

Using the Convert to Graphics Wizard, [401](#)

— V —

Variables window

- displaying, [427](#)

Verilog

- arrays, [153](#)

Verilog compiler directives

- setting, [26](#)

Verilog parameter

- declaration, [322](#)

- recovery, [398](#)

- usage example, [175](#)

- using, [174](#)

VHDL

- component declarations, [44](#)

VHDL generic

- declaration, [322](#)

- usage example, [175](#)

- using, [174](#)

VHDL package

- setting references, [23](#)

View

- all, [86](#)

- area, [86](#)

- diagram, [86](#)

- opening parent, [28](#)

- pan, [86](#)

- scroll, [86](#)

- zoom in, [86](#)

- zoom last, [86](#)

- zoom out, [86](#)

Visual attributes

- background color, [83](#)

- fill pattern, [83](#)

- foreground color, [83](#)

- line color, [83](#)

- line style, [83](#)

- line width, [83](#)

- setting, [83](#)

- setting color, [85](#)

- setting in the tabular IO view, [314](#)

text font, [83](#)

— W —

Wait box

object properties, [360](#)

Wait statement

syntax, [361](#)

Wave window

adding signals, [418](#)

displaying, [427](#)

Where used

component, [320](#)

Window

back, [29](#)

forward, [29](#)

panning, [86](#)

refreshing, [32](#)

re-using, [29](#)

saving position and size, [32](#)

scrolling, [86](#)

single, [29](#)

travel log, [29](#)

zooming, [86](#)

— Z —

Zoom

window, [86](#)

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. ORDERS, FEES AND PAYMENT.

- 1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.
- 1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
- 1.3. All Products are delivered FCA factory (Incoterms 2000), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer requests any change or enhancement to Software, whether in the course of

receiving support or consulting services, evaluating Software, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

- 4.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.
- 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
- 4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

- 5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use it except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark.
- 5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.
- 5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms

of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at <http://supportnet.mentor.com/about/legal/>.

7. **AUTOMATIC CHECK FOR UPDATES; PRIVACY.** Technological measures in Software may communicate with servers of Mentor Graphics or its contractors for the purpose of checking for and notifying the user of updates and to ensure that the Software in use is licensed in compliance with this Agreement. Mentor Graphics will not collect any personally identifiable data in this process and will not disclose any data collected to any third party without the prior written consent of Customer, except to Mentor Graphics' outside attorneys or as may be required by a court of competent jurisdiction.

8. **LIMITED WARRANTY.**

8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10. THE PROVISIONS OF THIS SECTION 11 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

12. **INFRINGEMENT.**

12.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance

to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

12.2. If a claim is made under Subsection 12.1 Mentor Graphics may, at its option and expense, (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.

12.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

12.4. THIS SECTION 12 IS SUBJECT TO SECTION 9 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS FOR DEFENSE, SETTLEMENT AND DAMAGES, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

13. **TERMINATION AND EFFECT OF TERMINATION.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term.

13.1. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

13.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.

14. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products and information about the products to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

15. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

16. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

17. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXIm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 17 shall survive the termination of this Agreement.

18. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, USA, if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International

Arbitration Centre (“SIAC”) to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. This section shall not restrict Mentor Graphics’ right to bring an action against Customer in the jurisdiction where Customer’s place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

19. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
20. **MISCELLANEOUS.** This Agreement contains the parties’ entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 100615, Part No. 246066