

Verification Continuum™

# **HAPS® Prototyping**

## **Compiler and Mapper Guide**

---

April 2022

**SYNOPSYS®**

## Copyright Notice and Proprietary Information

© 2022 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

## Free and Open-Source Licensing Notices

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

## Destination Control Statement

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

## Disclaimer

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

## Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at

<http://www.synopsys.com/Company/Pages/Trademarks.aspx>.

All other product or company names may be trademarks of their respective owners.

## **Third-Party Links**

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

Synopsys, Inc.  
690 East Middlefield Road  
Mountain View, CA 94043  
[www.synopsys.com](http://www.synopsys.com)

April 2022

## **Synopsys Statement on Inclusivity and Diversity**

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

# Contents

---

## Chapter 1: Specifying Constraints

Using the Constraints Editor . . . . .	16
Specifying FDC Constraints . . . . .	20
Entering and Editing Constraints from the Constraints Editor . . . . .	20
Defining Clocks and Path Constraints from the GUI . . . . .	39
Defining Clocks in the FDC File . . . . .	40
Defining Clock Groups . . . . .	41
Defining Clocks for Clock Muxes . . . . .	44
Defining Input and Output Constraints . . . . .	47
Specifying Standard I/O Pad Types . . . . .	48
Specifying Limits for FDC Wildcard Reporting . . . . .	49
Using the TCL View of Constraints GUI . . . . .	51
Guidelines for Entering and Editing Constraints . . . . .	53
Specifying Timing Exceptions . . . . .	56
Defining From/To/Through Points for Timing Exceptions . . . . .	56
Defining Multicycle Paths . . . . .	62
Defining False Paths . . . . .	63
Conflict Resolution for Timing Exceptions . . . . .	64
Finding Objects with Tcl find and expand . . . . .	68
Specifying Search Patterns for Tcl find . . . . .	68
Refining Tcl Find Results with -filter . . . . .	70
Using the Tcl Find Command to Define Collections . . . . .	71
Using the Tcl expand Command to Define Collections . . . . .	72
Using Collections . . . . .	74
Creating and Using Constraints Editor Collections . . . . .	75
Creating Collections using Tcl Commands . . . . .	77
Viewing and Manipulating Collections with Tcl Commands . . . . .	80
<b>Chapter 2: Analyzing with HDL Analyst</b>	
Working in the Schematic . . . . .	86

Opening Different Views . . . . .	86
Cloning Schematics . . . . .	89
Viewing Object Properties . . . . .	89
Viewing Objects with Constant Values . . . . .	92
Viewing Objects in a Source File . . . . .	93
Selecting Objects in Schematics . . . . .	96
Grouping Objects in Schematic . . . . .	98
Moving Between Views in a Schematic Window . . . . .	102
Setting the Schematic Preferences . . . . .	103
Exploring Design Hierarchy . . . . .	106
Traversing Design Hierarchy with the Hierarchy Browser . . . . .	106
Exploring Object Hierarchy with Push/Pop Commands . . . . .	108
Finding Objects . . . . .	113
Browsing to Find Objects in HDL Analyst Views . . . . .	113
Using Wildcards with the Find Command . . . . .	123
Crossprobing . . . . .	124
Crossprobing within a View . . . . .	124
Crossprobing from an HDL Analyst View . . . . .	125
Crossprobing to the Source Code . . . . .	127
Crossprobing from the Text Editor Window . . . . .	129
Crossprobing from the Log File . . . . .	131
Analyzing With the HDL Analyst Tool . . . . .	132
Viewing Design Hierarchy and Context . . . . .	132
Filtering Schematics . . . . .	138
Expanding Pin and Net Logic . . . . .	140
Dissolving and Partial Dissolving of Buses and Pins . . . . .	146
Dissolving of Ports . . . . .	149
Flattening Schematic Hierarchy . . . . .	150
Using the FSM Viewer . . . . .	153
Working in the Standard HDL Analyst Schematic . . . . .	157
Opening the Schematics . . . . .	157
Viewing Object Properties . . . . .	158
Selecting Objects in Schematics . . . . .	161
Working with Multi-sheet Schematics . . . . .	162
Moving Between Views in a Schematic Window . . . . .	163
Setting Schematic Preferences . . . . .	164
Exploring Design Hierarchy (Standard) . . . . .	166
Traversing Design Hierarchy with the Hierarchy Browser . . . . .	166
Exploring Object Hierarchy by Pushing/Popping . . . . .	167

Exploring Object Hierarchy of Transparent Instances . . . . .	171
Finding Objects (Standard) . . . . .	172
Browsing to Find Objects in HDL Analyst Views . . . . .	172
Using Find for Hierarchical and Restricted Searches . . . . .	174
Using Wildcards with the Find Command . . . . .	177
Combining Find with Filtering to Refine Searches . . . . .	180
Using Find to Search the Output Netlist . . . . .	181
Crossprobing (Standard) . . . . .	183
Crossprobing within a View . . . . .	183
Crossprobing from an HDL Analyst View . . . . .	184
Crossprobing from the Text Editor Window . . . . .	186
Analyzing With the Standard HDL Analyst Tool . . . . .	190
Viewing Design Hierarchy and Context . . . . .	190
Filtering Schematics . . . . .	194
Expanding Pin and Net Logic . . . . .	196
Expanding and Viewing Connections . . . . .	199
Flattening Schematic Hierarchy . . . . .	201
Minimizing Memory Usage While Analyzing Designs . . . . .	205

## Chapter 3: Defining Objects for Inference

Defining RAM in the HDL Code . . . . .	208
Block RAM Basics . . . . .	208
Setting Attributes to Guide RAM Inference . . . . .	209
Inferring Block RAM . . . . .	211
Inferring Distributed RAM . . . . .	216
Inferring Asymmetric RAM . . . . .	220
Inferring Byte-Enable RAM . . . . .	226
Initializing RAMs . . . . .	230
Initializing RAMs Using VHDL Signal or Variable Declarations . . . . .	230
Initializing RAM in VHDL with the INIT Property . . . . .	233
Initializing RAM Using Verilog \$readmemh or \$readmemb . . . . .	234
Initializing RAM in Verilog Using the INIT Property . . . . .	238
Initializing Xilinx RAM . . . . .	241
Specifying the INIT Property with Attributes . . . . .	241
Specifying Register INIT Values . . . . .	243
Generating the RAM MMI File . . . . .	245
Inferring Shift Registers . . . . .	248

Inferring Wide Adders . . . . .	251
Defining State Machines . . . . .	253
Defining State Machines in Verilog . . . . .	253
Defining State Machines in VHDL . . . . .	254
Specifying FSMs with Attributes and Directives . . . . .	255
Defining Black Boxes for Synthesis . . . . .	257
Using Black Boxes . . . . .	257
Instantiating Black Boxes and I/Os in Verilog . . . . .	260
Instantiating Black Boxes and I/Os in VHDL . . . . .	261
Adding Black Box Timing Constraints . . . . .	264
Adding Other Black Box Attributes . . . . .	267
Instantiating Xilinx Macros and Cores . . . . .	267
Specifying Xilinx Macros . . . . .	268
Instantiating CoreGen Cores . . . . .	270
Instantiating Virtex PCI Cores . . . . .	271
Packing Registers for Xilinx I/Os . . . . .	274
Inserting Xilinx I/Os and Specifying Pin Locations . . . . .	277
Working with Buffers . . . . .	284
Inferring Buffers . . . . .	284
Inferring BUFGDLL Clock Buffers . . . . .	285
Inferring Regional Clock Buffers . . . . .	286
Instantiating Special I/O Standard Buffers . . . . .	287
Specifying RLOCs . . . . .	289
Specifying RLOCs with xc_map, xc_rloc, and xc_usest . . . . .	289
Specifying RLOCs and RLOC_ORIGINS with the synthesis Attribute . . . . .	290

## **Chapter 4: Optimizing Synthesis Results**

Tips for Optimization . . . . .	292
General Optimization Tips . . . . .	292
Optimizing for Area . . . . .	293
Optimizing for Timing . . . . .	295
Optimizing Fanout . . . . .	297
Setting Fanout Limits . . . . .	297
Controlling Buffering and Replication . . . . .	299
Inserting BUFG Global Buffers . . . . .	300
Pipelining and Retiming . . . . .	303
Pipelining the Design . . . . .	303
Retiming the Design . . . . .	305

Preserving Objects from Being Optimized Away .....	311
Using syn_keep for Preservation or Replication .....	312
Controlling Hierarchy Flattening .....	315
Preserving Hierarchy .....	315
Sharing Resources .....	317
Inserting I/Os .....	318

## **Chapter 5: Working with Compile Points**

Automatic Compile Point Basics .....	320
ACPs and Runtime .....	320
Nested Compile Points .....	322
Compile Point Types .....	322
Automatic Compile Point Generation .....	327
Automatic Constraint Extraction and Interface Timing .....	328
Using Automatic Compile Points .....	330
Creating a Top-Level Constraints File for ACP .....	333
Compile Point Mapping .....	333
Incremental Compile Point Synthesis .....	334
Forward-annotation of Compile Point Timing Constraints .....	336
Analyzing Compile Point Results .....	336

## **Chapter 6: Working with IP**

Incorporating DesignWare IP .....	340
Importing Vivado IP .....	344
Creating Vivado IP .....	345
Importing Vivado IP Netlists .....	352
Importing IP from a Block Design File .....	356
Including IP Collections .....	360
Debugging XDC Conversion Messages .....	362
Working with Third-Party IP .....	363
Importing VCS Simulated Designs .....	366
The Synopsys FPGA IP Encryption Flow .....	368
Overview of the Synopsys FPGA IP Encryption Flow .....	368
Preparing and Encrypting IP .....	374
Preparing the IP Package .....	375
Working with IEEE 1735 Encryption .....	379
Encrypting IP Using IEEE 1735-2014 .....	380
IEEE 1735-Encryption Limitation .....	385

Including IEEE 1735-Encrypted IP in a Synthesis Flow . . . . .	385
Including IEEE 1735-Encrypted IP in a Partitioned Design . . . . .	386
The encryptP1735 Script . . . . .	388
IEEE 1735 Encryption Use Models . . . . .	391
Encrypting Bit Files . . . . .	398
Encrypting IP Using OpenIP (encryptIP) . . . . .	400
Encrypting IP with the OpenIP Scheme . . . . .	400
The encryptIP Script . . . . .	404
Working with Syncenc-encrypted IP . . . . .	407

## **Chapter 7: HDL Compiler Language Constructs**

Verilog Language Support . . . . .	410
Support for Verilog Language Constructs . . . . .	410
Compiler Directives . . . . .	412
Verilog 2001 Language Support . . . . .	413
Verilog Synthesis Guidelines . . . . .	416
General Synthesis Guidelines . . . . .	416
Library Support in Verilog . . . . .	417
Sets and Resets . . . . .	419
SRL Inference . . . . .	420
Verilog State Machines . . . . .	421
Instantiating Black Boxes in Verilog . . . . .	423
Hierarchical or Structural Verilog Designs . . . . .	424
Verilog Attribute and Directive Syntax . . . . .	427
SystemVerilog Language Support . . . . .	428
SystemVerilog Limitations . . . . .	431
SystemVerilog Assertions . . . . .	436
SystemVerilog Keyword Support . . . . .	437
VHDL Language Support . . . . .	439
Supported VHDL Language Constructs . . . . .	439
Unsupported VHDL Language Constructs . . . . .	440
Partially-supported VHDL Language Constructs . . . . .	441
Ignored VHDL Language Constructs . . . . .	441
VHDL Language Constructs . . . . .	442
Libraries and Packages . . . . .	442
VHDL Implicit Data-type Defaults . . . . .	446
Language Construct Restrictions . . . . .	450
VHDL 2008 Language Support . . . . .	451
Operators . . . . .	451

Unconstrained Data Types . . . . .	453
Unconstrained Record Elements . . . . .	453
Predefined Functions . . . . .	453
Generics . . . . .	454
Packages . . . . .	454
Generics in Packages . . . . .	455
Context Declarations . . . . .	456
Case-generate Statements . . . . .	456
Matching case and select Statements . . . . .	458
else/elsif Clauses . . . . .	458
Syntax Conventions . . . . .	459

## Chapter 8: Attributes and Directives

Specifying Attributes and Directives . . . . .	462
Specifying Attributes and Directives in VHDL . . . . .	463
Specifying Attributes and Directives in Verilog . . . . .	465
Specifying Directives in a CDC File . . . . .	466
Specifying Attributes Using the Constraints Editor . . . . .	467
Specifying Attributes in a Constraints File . . . . .	471
Attributes and Directives Summary . . . . .	473
black_box_pad_pin . . . . .	475
black_box_tri_pins . . . . .	479
CLOCK_DEDICATED_ROUTE . . . . .	481
diff_term . . . . .	483
full_case . . . . .	485
loop_limit . . . . .	489
parallel_case . . . . .	491
pragma translate_off/pragma translate_on . . . . .	493
syn_allow_retiming . . . . .	496
syn_allowed_resources . . . . .	500
syn_assign_to_region . . . . .	513
syn_assign_to_slr . . . . .	515
syn_async_reg . . . . .	517
syn_auto_insert_bufg . . . . .	520
syn_auto_insert_bufgmux . . . . .	522
syn_black_box . . . . .	526
syn_bram_cascade_height . . . . .	533
syn_clean_reset . . . . .	536
syn_clock_gmux_proxy . . . . .	539
syn_clock_priority . . . . .	543
syn_connect_hrefs . . . . .	547
corrupt_pd . . . . .	555

syn_cp_use_fast_synthesis . . . . .	559
syn_diff_io . . . . .	560
syn_direct_enable . . . . .	564
syn_direct_reset . . . . .	568
syn_direct_set . . . . .	572
syn_disable_purifyclock . . . . .	576
syn_DSPstyle . . . . .	579
syn_edif_bit_format . . . . .	591
syn_edif_scalar_format . . . . .	594
syn_encoding . . . . .	603
syn_enum_encoding . . . . .	611
syn_fast_auto . . . . .	616
syn_force_seq_prim . . . . .	622
syn_formal_blackbox . . . . .	624
syn_forward_io_constraints . . . . .	627
syn_gatedclk_clock_en . . . . .	630
syn_gatedclk_clock_en_polarity . . . . .	632
syn_global_buffers . . . . .	636
syn_hier . . . . .	642
syn_implement . . . . .	652
syn_insert_buffer . . . . .	653
syn_insert_pad . . . . .	659
syn_latch_ramstyle . . . . .	662
syn_isclock . . . . .	664
syn_keep . . . . .	667
syn_loc . . . . .	672
syn_looplimit . . . . .	676
syn_macro . . . . .	678
syn_map_dffrs . . . . .	683
syn_maxfan . . . . .	687
syn_multstyle . . . . .	692
syn_netlist_hierarchy . . . . .	695
syn_noarrayports . . . . .	701
syn_noclockbuf . . . . .	704
syn_no_compile_point . . . . .	709
syn_noprune . . . . .	712
syn_packer_effort_level . . . . .	724
syn_pad_type . . . . .	725
syn_partition_keep . . . . .	729
syn_partition_preserve . . . . .	730
syn_pipeline . . . . .	732
syn_pnr_preserve_regs . . . . .	736

syn_preserve . . . . .	738
syn_probe . . . . .	743
syn_ramstyle . . . . .	751
syn_ram_write_mem . . . . .	760
syn_reduce_controlset_size . . . . .	764
syn_reference_clock . . . . .	771
syn_rename_module . . . . .	773
syn_replicate . . . . .	775
syn_resources . . . . .	780
syn_ret_type . . . . .	784
syn_ret_lib_cell_type . . . . .	786
syn_romstyle . . . . .	787
syn_rw_conflict_logic . . . . .	792
syn_sharing . . . . .	797
syn_shift_resetphase . . . . .	801
syn_slow . . . . .	805
syn_srl_mindepth . . . . .	808
syn_srlstyle . . . . .	811
syn_state_machine . . . . .	815
syn_tco< <i>n</i> > . . . . .	819
syn_tpd< <i>n</i> > . . . . .	824
syn_trace_attr . . . . .	830
syn_tristate . . . . .	832
syn_tsu< <i>n</i> > . . . . .	835
syn_unconnected_inputs . . . . .	840
syn_unique_inst_module . . . . .	844
syn_upf_ret_port_type . . . . .	846
syn_useenables . . . . .	847
syn_useioff . . . . .	851
syn_useoddr . . . . .	858
syn_user_instance . . . . .	863
translate_off/translate_on . . . . .	869
xc_area_group . . . . .	872
xc_clockbuftype . . . . .	877
xc_fast . . . . .	881
xc_fast_auto . . . . .	885
xc_global_buffers . . . . .	888
xc_isgsr . . . . .	891
xc_loc . . . . .	894
xc_map . . . . .	897
xc_padtype . . . . .	901
xc_pullup/xc_pulldown . . . . .	902

## *Contents*

---

xc_rloc .....	906
xc_slow .....	910
xc_use_keep_hierarchy .....	911
xc_use_rpms .....	916
xc_use_timespec_for_io .....	919
xc_uset .....	923

## CHAPTER 1

# Specifying Constraints

---

This chapter describes how to specify constraints and attributes for the compile and map design stages. It covers the following:

- [Using the Constraints Editor](#), on page 16
- [Specifying FDC Constraints](#), on page 20
- [Specifying Timing Exceptions](#), on page 56
- [Conflict Resolution for Timing Exceptions](#), on page 64
- [Finding Objects with Tcl find and expand](#), on page 68
- [Using Collections](#), on page 74

# Using the Constraints Editor

The Constraints Editor produces a spreadsheet-like editor with a number of panels for entering and managing timing constraints and synthesis attributes. The constraints GUI is good for editing most constraints, but there are some constraints (like black box constraints) which can only be entered as directives in the source files. The constraints GUI also includes an advanced text editor that can help you edit constraints easily.

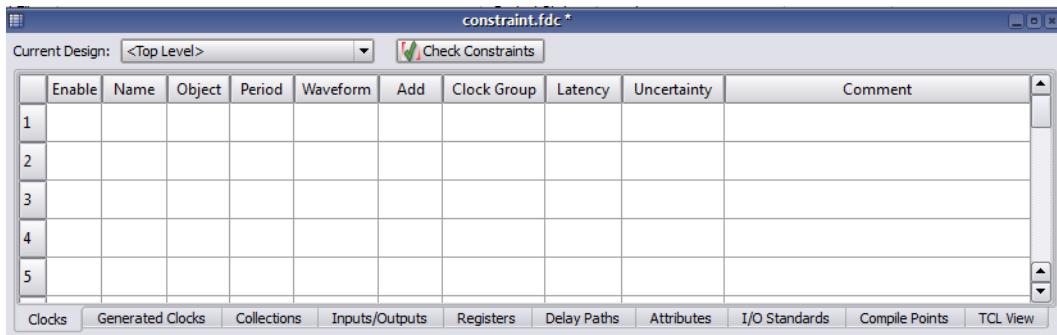
These constraints are saved to the FPGA Design Constraint (FDC) file. The FDC file contains *Synopsys SDC Standard* timing constraints (for example, `create_clock`, `set_input_delay`, and `set_false_path`), along with the non-timing constraints (design constraints) (for example, `define_attribute`, `define_collection`, and `define_io_standard`). When working with these constraints, see [Creating Constraints in the Constraints Editor, on page 16](#) for information on how to use the constraints editor.

## Creating Constraints in the Constraints Editor

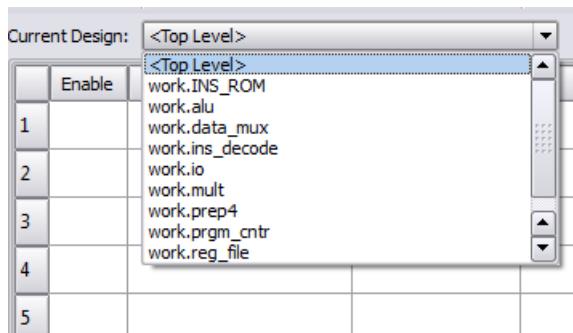
The following procedure shows you how to use the constraints editor to create constraints for the FDC constraint file.

1. To create a new constraint file, follow these steps:
  - Compile the design.
  - Open the constraints window by specifying the following command:  
`edit fdc fileName.fdc`

An empty constraints window opens when you specify a new constraint file. The tabs along the bottom of the window list the different kinds of constraints you can add. For each kind of constraint, the columns contain specific data.



2. Select if you want to apply the constraint to the top-level or for modules from the Current Design option drop-down menu located at the top of the constraints editor.



3. You can enter or edit the following types of constraints:
  - Timing constraints – on the Clocks, Generated Clocks, Inputs/Outputs, Registers, or Delay Paths tab.
  - Design constraints – on the Collections, Attributes, I/O Standards, or Compile Points tab.

For details about these constraints, see [Specifying FDC Constraints](#), on page 20.

For information about ways to enter constraints within the constraints editor, see [Guidelines for Entering and Editing Constraints](#), on page 53. The editor is located in the TCL View tab, which is the last tab in the constraints editor. This text editor has a help window on the right-hand

side. For more information about this text editor, see [Using the TCL View of Constraints GUI, on page 51](#).

4. Click on the Check Constraints button to run the constraint checker. Alternatively, use the report constraint\_check command.

The output provides information on how the constraints are interpreted by the tool.

All constraint information is saved in the same FPGA Design Constraint file (FDC) with clearly marked beginning and ending for each section. Do not manually modify these pre-defined constraints sections.

The following example shows the contents of an FDC file.

```
# These sections are generated from the spreadsheet tabs.

##### END Header

##### BEGIN Collections - (Populated from tab in SCOPE, do not edit)
##### END Collections

##### BEGIN Clocks - (Populated from tab in SCOPE, do not edit)
create_clock {p:clk_a} -period {10} -waveform {0 5.0}
create_clock {p:clk_b} -period {6.667} -waveform {0 3.3335}
set_clock_groups -derive -name default_clkgroup_0 -asynchronous -group [get_clocks {clk_a}];
set_clock_groups -derive -name default_clkgroup_1 -asynchronous -group [get_clocks {clk_b}];

##### END Clocks

##### BEGIN "Generated Clocks" - (Populated from tab in SCOPE, do not edit)
##### END "Generated Clocks"

##### BEGIN Inputs/Outputs - (Populated from tab in SCOPE, do not edit)
set_input_delay -clock {c:clk_a} -clock_fall -add_delay 0.00 $all_inputs_fdc
set_output_delay -clock {c:clk_a} -add_delay 0.000 $all_outputs_fdc
set_input_delay -clock {c:clk_a} -add_delay 2.00 {p:a[7:0]}
set_input_delay -clock {c:clk_a} -add_delay 2.00 {p:rst}
##### END Inputs/Outputs

##### BEGIN Registers - (Populated from tab in SCOPE, do not edit)
##### END Registers

##### BEGIN "Delay Paths" - (Populated from tab in SCOPE, do not edit)
##### END "Delay Paths"
set_multicycle_path 3 -end -from $fcc_cmd_0
set_false_path -comment {false foo [0] free {iddhh}} -from {p:ena}
set_false_path -from $fdc_cmd_2 -to {i:abc.def.g reg} -through {n:bar}
##### BEGIN Attributes - (Populated from tab in SCOPE, do not edit)
##### END Attributes

##### BEGIN "I/O Standards" - (Populated from tab in SCOPE, do not edit)
##### END "I/O Standards"

##### BEGIN "Compile Points" - (Populated from tab in SCOPE, do not edit)
##### END "Compile Points"
```

# Specifying FDC Constraints

Timing constraints define the performance goals for a design. The tool supports a subset of the Synopsys SDC Standard timing constraints (for example, `create_clock`, `set_input_delay`, and `set_false_path`).

Design constraints let you add attributes, define collections and specify constraints for them, and select specific I/O standard pad types for your design.

You can define both timing and design constraints in the constraints editor. For the different types of constraints, see the following topics:

- [Entering and Editing Constraints from the Constraints Editor](#)
- [Defining Clocks and Path Constraints from the GUI](#)
- [Defining Clock Groups](#)
- [Defining Clocks for Clock Muxes](#)
- [Defining Input and Output Constraints](#)
- [Specifying Standard I/O Pad Types](#)

To set constraints for timing exceptions like false paths and multi-cycle paths, see [Specifying Timing Exceptions, on page 56](#).

For information about collections, see [Using Collections, on page 74](#).

## Entering and Editing Constraints from the Constraints Editor

This table and the following subsections describe the timing and design constraints you can enter from the various panels of constraints editor GUI. The constraints are then saved to an FDC file.

Constraint Editor Panel	See ...
Clocks and Clock Groups	<a href="#">Clocks, on page 21</a>
Generated Clocks	<a href="#">Generated Clocks, on page 23</a>
Collections	<a href="#">Collections, on page 24</a>
Inputs/Outputs	<a href="#">Inputs/Outputs, on page 27</a>

**Constraint Editor Panel****See ...**

Registers	<a href="#">Registers</a> , on page 30
Delay Paths	<a href="#">Delay Paths</a> , on page 31
Attributes	<a href="#">Attributes</a> , on page 33
I/O Standards	<a href="#">I/O Standards</a> , on page 34
Compile Points	<a href="#">Compile Points</a> , on page 35
TCL View	<a href="#">TCL View</a> , on page 38

**Clocks**

Use the Clocks panel to designate specified signals as clocks.

	Enable	Name	Object	Period	Waveform	Add	Clock Group	Latency	Uncertainty	Comment
1										
2										
3										
4										

**Clocks**

The Clocks panel includes the options shown below. For more about defining clocks, see [Defining Clocks, on page 44](#) in the *User Guide*.

Field	Description
Name	Specifies the clock object name which can be defined on pin, port, or net objects. For virtual clocks, the field must contain a unique name not associated with any port, pin, or net in the design.
Period	Specifies the clock period in nanoseconds which is defined as the minimum time over which the clock waveform repeats. The period must be greater than zero.
Waveform	Specifies the rise and fall edge times for the clock waveforms of the clock in nanoseconds, over an entire clock period. The first time in the list is a rising transition, typically the first rising transition after time zero. There must be two edges, and they are assumed to be rise and then fall. The edges must be monotonically increasing. If you do not specify this option, a default waveform with a rising edge of 0.0 and a falling edge of the period divided by 2 is assumed.
Add Delay	Specifies whether to add this delay to the existing clock or to overwrite it. Use this option when multiple clocks must be specified on the same source for simultaneous analysis with different clock waveforms. When you use this option, you must also specify the clock; clocks with the same source must have different names.
Clock Group	Assigns clocks to asynchronous clock groups. The clock grouping is inclusive (for example, clk2 and clk3 can each be related to clk1 without being related to each other). For details, see <a href="#">Defining Clock Groups , on page 41</a> .
Latency	Specifies the clock latency applied to clock ports and clock aliases. Applying the latency constraint on a port can be used to model the off-chip clock delays in a multi-chip environment. Clock latency can only: <ul style="list-style-type: none"> <li>• apply to clocks defined on input ports</li> <li>• be used for source latency</li> <li>• apply to port clock objects</li> </ul>
Uncertainty	Specifies the clock uncertainty (skew characteristics) of the specified clock networks. You can only apply latency to clock objects.

## Generated Clocks

Use the Generated Clocks panel of the constraint editor to define a signal as a generated clock. For more about using generated clocks, see [Converting Gated Clocks, on page 862](#) in the *User Guide*.

	Enable	Name	Source	Object	Master Clock	Generate Type	Generate Parameters	Generate Modifier	Modifier Parameters	Invert	Add	Comment
1												
2												
3												
4												

Generated Clocks

The Generated Clocks panel includes the following options:

Field	Description
Name	Specifies the name of the generated clock. If this option is not used, the clock gets the name of the first clock source specified in the source.
Source	Specifies the master clock pin, which is either a master clock source pin or a fanout pin of the master clock driving the generated clock definition pin. The clock waveform at the master pin is used for deriving the generated clock waveform.
Object	Generated clocks can be defined on pin, port, net, and instance objects when the instance has only one output (for example, a BUFG).
Master Clock	Specifies the master clock to be used for this generated clock, when multiple clocks fan into the master pin.

Field	Description
Generate Type	<p>Specifies one of the following:</p> <p>edges – Specifies a list of integers that represents edges from the source clock that are to form the edges of the generated clock. The edges are interpreted as alternating rising and falling edges and each edge must not be less than its previous edge. The number of edges must be an odd number and not less than 3 to make one full clock cycle of the generated clock waveform. For example, 1 represents the first source edge, 2 represents the second source edge, and so on.</p> <p>divide_by – Specifies the frequency division factor. If the divide factor value is 2, the generated clock period is twice as long as the master clock period.</p> <p>multiply_by – Specifies the frequency multiplication factor. If the multiply factor value is 3, the generated clock period is one-third as long as the master clock period.</p>
Generate Parameters	Specifies integers that define the type of generated clock.
Generate Modifier	Defines the secondary characteristics of the generated clock.
Modify Parameters	Defines modifier values of the generated clock.
Invert	Specifies whether to use invert – Inverts the generated clock signal (in the case of frequency multiplication and division).
Add	Either add this clock to the existing clock or overwrite it. Use this option when multiple generated clocks must be specified on the same source, because multiple clocks fan into the master pin. Ideally, one generated clock must be specified for each clock that fans into the master pin. If you specify this option, you must also specify the clock and master clock. The clocks with the same source must have different names.

## Collections

The Collections tab allows you to set constraints for a group of objects defined as a collection with the Tcl command. For more about using Tcl collections, see [Using Collections, on page 74](#).

Enabled	Collection Name	Command	Command Arguments	Comment
1				
2				
3				
4				
-				

Collections

Field	Description
Name	Enter the collection name.
Command	Select a collection creation command from the drop-down menu.
Command Arguments	Specify the Tcl syntax for the constraint you want to apply to the collection.
Comment	Enter comments that are included in the constraints file.

## Collection Commands

You can use the collection commands on collections or Tcl lists. The following table summarizes the collection commands; see [Using Collections, on page 74](#) for step-by-step details.

---

To ...	Use this command ...
Create a collection	<p><code>set modules</code> To create and save a collection, assign it to a variable. You can also use this command to create a collection from any combination of single elements, TCL lists and collections:</p> <pre>set modules [define_collection {v:top} {v:cpu} \$mycoll \$mylist]</pre> <p>Once you have created a collection, you can assign constraints to it in the constraint editor.</p>
Copy a collection	<p><code>set modules_copy \$modules</code> Copies the collection so that any change to \$modules does not affect \$modules_copy.</p>
Evaluate a collection	<p><code>c_print</code> Returns all objects in a column format. Use this for visual inspection.</p> <p><code>c_list</code> Returns a Tcl list of objects. Use this to convert a collection to a list. You can manipulate a Tcl list with standard Tcl list commands.</p>
Concatenate a list to a collection	<code>c_union</code>
Identify differences between lists or collections	<p><code>c_diff</code> Identifies differences between a list and a collection or between two or more collections. Use the -print option to display the results.</p>
Identify objects common to a list and a collection	<p><code>c_intersect</code> Identifies objects common to a list and a collection. Use the -print option to display the results.</p>
Identify objects common to two or more collections	<p><code>c_sub</code> Identifies objects common to two or more collections. Use the -print option to display the results.</p>
Identify objects that belong exclusively to only one list or collection	<p><code>c_symdiff</code> Identifies unique objects in a list and a collection, or two or more collections. Use the -print option to display the results.</p>

---

## Inputs/Outputs

The Inputs/Outputs panel models the interface of the FPGA with the outside environment. Use this panel to specify delays outside the device.

	Enable	Delay Type	Port	Rise	Fall	Max	Min	Clock	Clock Fall	Add Delay	Value	Comment
1												
2												
3												
4												

Inputs/Outputs

The Inputs/Outputs panel includes the following options:

Field	Description
Delay Type	Specifies if the delay is an input or output delay.
Port	Specifies the name of the port.
Rise	Specifies that the delay is relative to the rising transition on specified port. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst-case path delay is used instead. However, the -rise option is preserved and forward annotated to the place-and-route tool.
Fall	Specifies that the delay is relative to the falling transition on specified port. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst-case path delay is used instead. However, the -fall option is preserved and forward annotated to the place-and-route tool.
Max	Specifies that the delay value is relative to the longest path. <b>Note:</b> The -max delay values are reported in the top-level log file and are forward annotated to the place-and-route tool.
Min	Specifies that the delay value is relative to the shortest path. <b>Note:</b> The synthesis tool does not optimize for hold-time violations and only reports -min delay values in the synlog/topLevel_fpga_mapper.srr_Min timing report section of the log file. The -min delay values are forward annotated to the place-and-route tool.

Field	Description
Clock	Specifies the name of a clock for which the specified delay is applied. If you specify the clock fall, you must also specify the name of the clock.
Clock Fall	Specifies that the delay is relative to the falling edge of the clock.
Add Delay	Specifies whether to add delay information to the existing input delay or overwrite the input delay. For examples, see <a href="#">Input Delays , on page 28</a> and <a href="#">Output Delays , on page 29</a> .
Value	Specifies the delay path value.

## Input Delays

Certain constraint options affect input delays:

- Clock Fall – The default is the rising edge or rising transition of a reference pin. If you specify clock fall, you must also specify the name of the clock.
- Add Delay – Use this option to capture information about multiple paths leading to an input port relative to different clocks or clock edges.

For example, the command `set_input_delay 5.0 -max -rise -clock phi1 {A}` removes all maximum rise input delay from A because the `-add_delay` option is not specified. Other input delays with different clocks or with `-clock_fall` are removed.

If the `-add_delay` option is specified (`set_output_delay 5.0 -max -rise -clock phi1 -add_delay {Z}`) and there is an input maximum rise delay for A relative to clock phi1 rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is maximum rise input delay relative to a different clock or different edge of the same clock, it remains with the new delay.

## Output Delays

Certain constraint options affect output delays:

- Clock Fall – If you specify clock fall, you must also specify the name of the clock.
- Add Delay – By using this option, you can capture information about multiple paths leading from an output port relative to different clocks or clock edges.

For example, the `set_output_delay 5.0 -max -rise -clock phi1 {OUT1}` command removes all maximum rise output delays from OUT1, because the `-add_delay` option is not specified. Other output delays with a different clock or with the `-clock_fall` option are removed.

If the `-add_delay` option is specified (`set_output_delay 5.0 -max -rise -clock phi1 -add_delay {Z}`) and if there is an output maximum rise delay for Z relative to the clock phi1 rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is a maximum rise output delay relative to a different clock or different edge of the same clock, it remains with the new delay.

## Conflict Resolution for Multiple I/O Constraints

You can specify multiple input and output delay constraints for the same I/O port which can be useful for cases where a port is driven by or feeds multiple clocks. The priority of a constraint and its use is determined by several factors:

- The timing report reports all applicable constraints.
- The tool applies the tightest constraint for a given clock edge and ignores all others.
- You can apply I/O constraints on three levels with the more specific overriding the more global:
  - Global (top-level netlist) for all inputs and outputs
  - Port-level for the entire bus
  - Bit-level for single bits

If there are bit constraints and port constraints defined for the same bit, the bit constraints override the port constraints for that bit. Other bits

that do not have bit constraints take the port constraints. For example, take the following constraints:

```
a[3:0]3 clk1:r
a[3:0]3 clk2:r
a[0]2 clk1:r
```

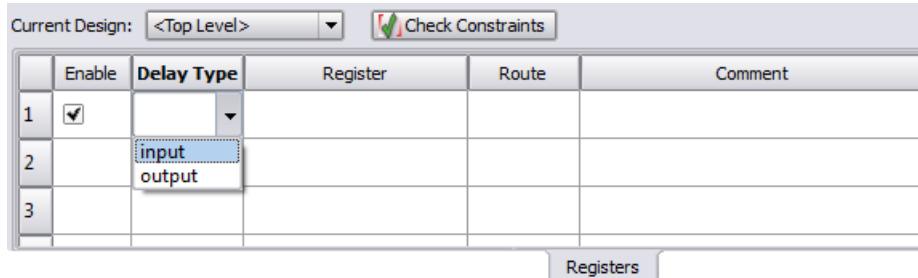
In this case, port a[0] only gets one constraint of 2 ns. Ports a[1], a[2], and a[3] get two constraints of 3 ns each.

- If at any given level (bit, port, or global), there is a constraint with a reference clock specified, then any constraint without a reference clock is ignored. In this example, the 1 ns constraint on port a[0] is ignored.

```
a[0]2 clk1:r
a[0]1
```

## Registers

This panel lets the advanced user add delays to paths feeding into/out of registers, in order to further constrain critical paths. You use this constraint to speed up the paths feeding a register.



The screenshot shows a software interface for managing register delays. At the top, there is a dropdown menu labeled "Current Design: <Top Level>" and a "Check Constraints" button. Below this is a table with the following columns: Enable, Delay Type, Register, Route, and Comment. The table has four rows, indexed 1, 2, and 3. Row 1 has an checked "Enable" checkbox and a dropdown menu under "Delay Type". Row 2 has an unchecked "Enable" checkbox and a dropdown menu containing "input" and "output", with "input" currently selected. Row 3 has an unchecked "Enable" checkbox and an empty "Delay Type" field. At the bottom right of the table area is a "Registers" button.

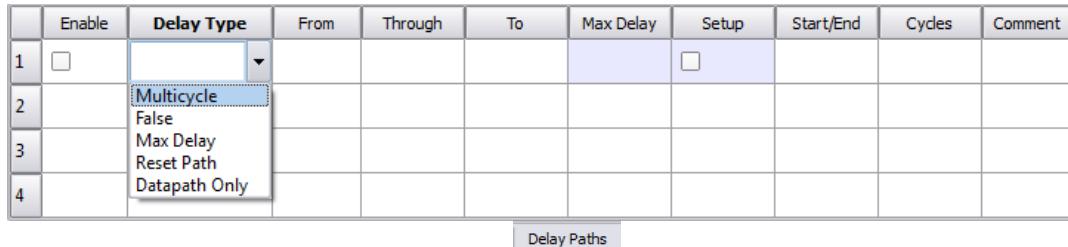
	Enable	Delay Type	Register	Route	Comment
1	<input checked="" type="checkbox"/>				
2	<input type="checkbox"/>	input output			
3	<input type="checkbox"/>				

The Registers constraint panel includes the following fields:

Field	Description
Delay Type	Specifies if the delay is an input or output delay.
Register	Specifies the name of the register. If you have initialized a compiled design, you can choose from the pull-down list.
Route	Improves the speed of the paths to or from the register by the given number of nanoseconds. The value shrinks the effective period for the constrained registers without affecting the clock period that is forward-annotated to the place-and-route tool.
Comment	Lets you enter comments that are included in the constraints file.

## Delay Paths

Use the Delay Paths panel to define any timing exceptions.



The screenshot shows a table titled "Delay Paths" with four rows. The columns are labeled: Enable, Delay Type, From, Through, To, Max Delay, Setup, Start/End, Cycles, and Comment. Row 1 has an unchecked checkbox in the Enable column and a dropdown menu open in the Delay Type column, showing options: Multicycle, False, Max Delay, Reset Path, and Datapath Only. Row 2 has a checked checkbox in the Enable column. Rows 3 and 4 are empty. A button labeled "Delay Paths" is located at the bottom right of the table.

	Enable	Delay Type	From	Through	To	Max Delay	Setup	Start/End	Cycles	Comment
1	<input type="checkbox"/>	Multicycle False Max Delay Reset Path Datapath Only					<input type="checkbox"/>			
2	<input checked="" type="checkbox"/>									
3	<input type="checkbox"/>									
4	<input type="checkbox"/>									

The Path Delay panel includes the following options:

Field	Description
Delay Type	Specifies the type of delay path you want the synthesis tool to analyze from one of the following types: <ul style="list-style-type: none"> <li>• Multicycle</li> <li>• False</li> <li>• Max Delay</li> <li>• Reset Path</li> <li>• Datapath Only</li> </ul>
From	Starting point for the path. From points define timing start points and can be defined for clocks (c:), registers (i:), top-level input or bi-directional ports (p:), black box output pins (i:) or sequential cell clock pins.
Through	Specifies the intermediate points for the timing exception. Intermediate points can be combinational nets (n:), hierarchical ports (t:), or instantiated cell pins (t:). If you click the arrow in a column cell, you open the Product of Sums (POS) interface where you can set through constraints.
To	Ending point of the path. To points must be timing end points and can be defined for clocks (c:), registers (i:), top-level output or bidirectional ports (p:), or black box input pins (i:).
Max Delay	Specifies the maximum delay value for the specified path in nanoseconds.
Setup	Specifies the setup (maximum delay) calculations used for specified path.
Start/End	Used for multicycle paths with different start and end clocks. This option determines the clock period to use for the multiplicand in the calculation for clock distance. If you do not specify a start or end clock, the end clock is the default.
Cycles	Specifies the number of cycles required for the multicycle path.

## Attributes

You can assign attributes directly in the editor.

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description	more
1	<input checked="" type="checkbox"/>	output_port	<global>	syn_nodclockbuf				
2	<input checked="" type="checkbox"/>			syn_clean_reset				
3	<input checked="" type="checkbox"/>			syn_dspstyle				
4	<input checked="" type="checkbox"/>			syn_edif_bit_format				
5	<input checked="" type="checkbox"/>			syn_edif_scalar_format				
				syn_forwar...onstraints				
				syn_multstyle				
				syn_netlist_hierarchy				
				syn_noarrayports				
				syn_nodclockbuf				
				syn_ramstyle				

Attributes

Here are descriptions for the Attributes columns:

Column	Description
Object Type	Specifies the type of object to which the attribute is assigned. Choose from the pull-down list, to filter the available choices in the Object field.
Object	Specifies the object to which the attribute is attached. This field is synchronized with the Attribute field, so selecting an object here filters the available choices in the Attribute field.
Attribute	Specifies the attribute name. You can choose from a pull-down list that includes all available attributes for the specified technology. This field is synchronized with the Object field. If you select an object first, the attribute list is filtered. If you select an attribute first, the Synopsys FPGA synthesis tool filters the available choices in the Object field. You must select an attribute before entering a value. If a valid attribute does not appear in the pull-down list, simply type it in this field and then apply appropriate values.
Value	Specifies the attribute value. You must specify the attribute first. Clicking in the column displays the default value; a drop-down arrow lists available values where appropriate.
Val Type	Specifies the type of value for the attribute. For example, string or boolean.
Description	Contains a one-line description of the attribute.
Comment	Lets you enter comments about the attributes.

Enter the appropriate attributes and their values, by clicking in a cell and choosing from the pull-down menu. To specify an object to which you want to assign an attribute, you may also drag-and-drop it from the RTL or Technology view into a cell in the Object column. After you have entered the attributes, save the constraint file.

## I/O Standards

You can specify a standard I/O pad type to use in the design. Define an I/O standard for any port appearing in the I/O Standards panel.

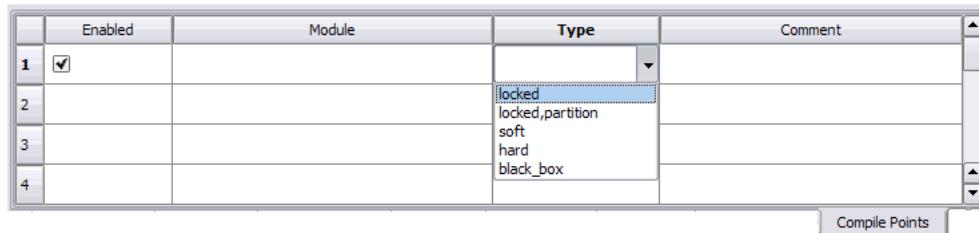
	Enabled	Port	Type	I/O Standard	DCI	DV2	Slew Rate	Drive Strength	Termination	Description
1	<input checked="" type="checkbox"/>	<input default>	input	LVCMS_15			fast	8	pullup	1.5 volt - C...
2	<input checked="" type="checkbox"/>	<output default>	output							
3	<input type="checkbox"/>	<bidir default>	bidir							
4	<input type="checkbox"/>	resetn	input							

Field	Description
Enabled	Turn this on to enable the constraint, or off to disable a previous constraint.
Port	Specifies the name of the port. If you have initialized a compiled design, you can select a port name from the pull-down list. The first two entries let you specify global input and output delays, which you can then override with additional constraints on individual ports.
Type	Specifies whether the delay is an input or output delay.
I/O Standard	Supported I/O standards by Synopsys FPGA products.

DCI	The values for these parameters are based on the selected I/O standard.
DV2	
Slew Rate	
Drive Strength	
Termination	
Power	
Schmitt	
Description	Describes the selected I/O Standard.
Comment	Enter comments about an I/O standard.

## Compile Points

Use the Compile Points panel to specify compile points in your design and to enable/disable them. This panel is used to define a top-level constraint file.

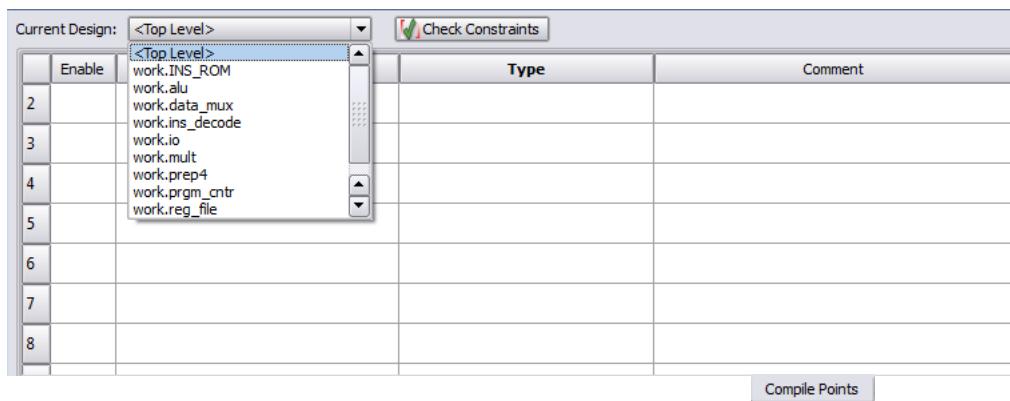


Here are the descriptions of the fields in the Compile Points panel.

Field	Description
Enabled	Turn this on to enable the constraint.
Module	Specifies the name of the compile-point module. You must specify a view module, with a v: prefix to identify the module as a view. For example: v:alu.
Type	<p>Specifies the type of compile point:</p> <ul style="list-style-type: none"> <li>• locked (default) – no timing reoptimization is done on the compile point. The hierarchical interface is unchanged and an interface logic model is constructed for the compile point.</li> <li>• locked, partition – locked compile point, for which compile point information is forward annotated to the place and route tool. This mode provides place and route runtime advantages and allows for obtaining stable results for a completed design.</li> <li>• soft – compile point is included in the top-level synthesis, boundary optimizations can occur.</li> <li>• hard – compile point is included in the top-level synthesis, boundary optimizations can occur, however, the boundary remains unchanged. Although, the boundary is not modified, instances on both sides of the boundary can be modified using top-level constraints.</li> <li>• black_box – the compile point is treated as a black box. The software ignores the contents of the compile point that includes all instances in the module; only its ports exist for synthesis. Black box compile point modules only write port definitions to the netlist files.</li> </ul>
Comment	Lets you enter a comment about the compile point.

## Constraints for Compile Points

You can set constraints at the top-level or for modules to be used as the compile points from the Current Design pull-down menu shown below. Use the Compile Points tab to select compile points and specify their types.



## TCL View

The TCL View is an advanced text file editor for defining FPGA timing and design constraints.

The screenshot shows the Synopsys HAPS Prototyping Compiler and Mapper Guide's "TCL View" window. The main area is a text editor containing a series of TCL commands for specifying FDC constraints. The commands include setting RTL FF names, creating clocks, and defining clock groups. A red arrow points from the text "Click on Hide Syntax Help to close this browser" down to a "Syntax Help" button at the bottom of the window.

```

146
147 set_rtl_ff_names {#}
148 set qffs [define_collection           [find -seq -hier {qb[*]}]
149 set f_max 150
150 create_clock -name {clk_a} [get_ports {clk_a}] -period 10 -wa
151 create_clock -name {clk_b} [get_ports {clk_b}] -period 6.6666
152 set_input_delay -clock [get_clocks {clk_a}] -clock_fall -add
153 set_output_delay -clock [get_clocks {clk_a}] -add_delay 0.00
154 set_input_delay -clock [get_clocks {clk_a}] -add_delay 2.00
155 set_input_delay -clock [get_clocks {clk_a}] -add_delay 0 [ge
156 set mcp 4
157 set_multicycle_path $mcp -start \
158   -from \
159     [get_ports \
160       {a* \
161         b*} \
162       ]
163   -to \
164     [find -seq -hier {q?[*]} ]
165
166 set_multicycle_path 3 -end \
167   -from \
168     [find -seq {*y*.q[*]} ]
169
170 set_clock_groups -name default_clkgroup_0 -asynchronous \
171   -group [get_clocks {clk_a dcm|clk_0_derived_clock dcm|clk

```

Current Design: <Top Level> Check Constraints

Name

- FDC Constraints
  - bus\_dimension\_separat...
  - bus\_naming\_style
  - create\_clock
  - create\_generated\_clock
  - define\_attribute
  - define\_compile\_point
  - define\_global\_attribute
  - define\_io\_standard
  - define\_scope\_collection
  - read\_sdc
  - reset\_path
  - set\_clock\_groups
  - set\_clock\_latency
  - set\_clock\_uncertainty

Constraint Syntax:

```
set_rtl_ff_names
  value <string value>
```

Hide Syntax Help Ln 1 Col 1 Total 173 Ovr Block

Click on Hide Syntax Help to close this browser

Syntax Help

This text editor provides the following capabilities:

- Uses dynamic keyword expansion and tool tips for commands that
  - Automatically completes the command from a popup list
  - Displays complete command syntax as a tool tip
  - Displays parameter options for the command from a popup list
  - Includes a keyword command syntax help

- Checks command syntax and uses color indicators that
  - Validate commands and command syntax
  - Identifies FPGA design constraints and SCOPE legacy constraints
- Allows for standard editor commands, such as copy, paste, comment/un-comment a group of lines, and highlighting of keywords.

## Defining Clocks and Path Constraints from the GUI

The following table summarizes how to set different clock and path constraints from the constraints editor GUI.

To define ...	Pane	Do this to set the constraint ...
Clocks	Clock	<p>Select the clock object (Clock).</p> <p>Specify a clock name (Clock Alias), if required.</p> <p>Type a period (Period).</p> <p>Set the rise and fall edge times for the clock.</p> <p>Specify waveforms of the clock in nanoseconds, if needed.</p> <p>Define a clock group, if needed (see below).</p> <p>Check the Enabled box.</p>
Clock Groups	Clock	<p>Select the clock object (Clock).</p> <p>Define the clocks as usual (see above).</p> <p>For synchronous clock groups, assign the clocks to the same group, or leave them in the default clock group.</p> <p>For asynchronous clock groups, assign the clock to a new group. Each clock group defined is asynchronous to the other clock groups and to the default clock group.</p> <p>Check the Enabled box.</p> <p>See <a href="#">Defining Clock Groups , on page 41</a> for details.</p>
Generated Clocks	Generated Clocks	<p>Select the generated clock object.</p> <p>Specify the master clock source (a clock source pin in the design).</p> <p>Specify whether to use invert for the generated clock signal.</p> <p>Specify whether to use: edges, divide_by, or multiply_by.</p> <p>Check the Enabled box.</p>
Input/output delays	Inputs/Outputs	See <a href="#">Defining Input and Output Constraints , on page 47</a> for information about setting I/O constraints.

To define ...	Pane	Do this to set the constraint ...
Maximum path delay	Delay Paths	Select the Delay Type path of Max Delay. Select the start/from point for either a port or register (From/Through). See <a href="#">Defining Through Points , on page 59</a> for more information. Select the end/to point for either an output port or register. Specify a through point for a net or hierarchical port/pin (To/Through). Set the delay value (Max Delay). Check the Enabled box.
Multicycle paths	Delay Paths	See <a href="#">Defining Multicycle Paths , on page 62</a> .
False paths	Delay Paths	See <a href="#">Defining False Paths , on page 63</a> for details.
Global attributes	Attributes	Set Object Type to <global>. Select the object (Object). Set the attribute (Attribute) and its value (Value). Check the Enabled box.
Attributes	Attributes	Do either of the following: <ul style="list-style-type: none"> <li>• Select the type of object (Object Type). Select the object (Object). Set the attribute (Attribute) and its value (Value). Check the Enabled box.</li> <li>• Set the attribute (Attribute) and its value (Value). Select the object (Object). Check the Enabled box.</li> </ul>

## Defining Clocks in the FDC File

Define clocks in the FDC file so that the synthesis process uses the low-skew resources built into the FPGA for the clocks.

1. For multi-FPGA designs, specify the HAPS clock information in the TSS and PCF files, as described in [Defining Clocks, on page 44](#).

You can then define the clocks for synthesis, as described in the subsequent steps below.

2. Specify all master clocks with `create_clock`:

```
create_clock -name myDesignClk [get_ports myDesignClk] -period 20
```

- Make sure to specify the clock object. Clocks can be defined on pins, ports, and nets.

To find the names for the objects to be used in the constraint, use the schematic viewer (HDL Analyst). From a compiled state, right-click in the Database View and select View Schematic to open the viewer. In the viewer, right-click on the net you want to designate as a clock, select Copy, and paste the name into your constraints file,

- Specify a period.
- Optionally, specify the rise and fall times for the clock.
- If you do not define the clocks, the tool can infer them. Inferred clocks negatively affect performance. After an initial run, check for inferred clocks and provide clock definitions for them.

For details about the command syntax, see [create\\_clock, on page 290](#) in the *Command Reference*.

### 3. Optionally, define a clock group.

Each clock group defined is asynchronous to the other clock groups and to the default clock group. For synchronous clock groups, assign the clocks to the same group, or leave them in the default clock group. For asynchronous clock groups, assign the clock to a new group. You must define clock groups for asynchronous clocks. See [Defining Clock Groups, on page 41](#) for details.

### 4. Define generated or divided clocks with `create_generated_clock`.

See [Defining Clocks for Gated Clock Conversion, on page 872](#) in the *User Guide* for details.

## Defining Clock Groups

The clock group to which a clock is assigned determines whether it is synchronous and asynchronous. Clocks in the same group are synchronous with each other but asynchronous to clocks in other groups. By default, the tool assigns all clocks to the default clock group and all clocks in the group are synchronous. Assigning a clock to another group makes it asynchronous to the default clock group as well as all other named clock groups. Clock grouping is associative; two clocks can be asynchronous to each other but both can be synchronous with a third clock.

1. Define the clocks with `create_clock` and `create_generated_clock` constraints.
2. Define the clock group with the `set_clock_group` Tcl constraint or through the Constraint Editor GUI, as described below:

For details of the Tcl constraint syntax, [set\\_clock\\_groups, on page 307](#) in the *Command Reference*. For example:

```
set_clock_groups -derive -asynchronous -name {group1}  
-group {{c:clk1} {c:clk2}}
```

- To enter the constraint in the GUI, select the clock object in the Clocks pane in the GUI.
  - Define the clocks as usual.
  - For synchronous clocks, assign the clocks to the same group, or leave them in the default clock group. Each additional clock group defined is asynchronous to the other clock groups and to the default clock group.
  - Check the Enabled box to set the constraint.
3. To define asynchronous clocks, assign the clocks to different clock groups with the `-group` argument to the `set_clock_groups` constraint.

There are two ways to do this.

- To make clocks asynchronous to all other clocks outside the group, assign them using a single `-group` argument. In the following example, `clk1` is asynchronous to all other clocks in the design.

```
set_clock_groups -asynchronous -group {clk1}
```

- To make clocks asynchronous to selected clocks, use one constraint with multiple `-group` arguments to separate the clocks in different groups. The following example makes the clocks in one group asynchronous with clocks from any other group defined in the constraint. Clocks in the same group remain synchronous with each other. Thus, in the following example, `clk1` is synchronous with `clk2`, but asynchronous to `clk3`, `clk4`, `clk5`, and `clk6`.

```
set_clock_groups -asynchronous -group {clk1 clk2}  
-group {clk3 bclk4} -group {clk5 cclk6}
```

The following GUI example shows three clocks (`clk1`, `clk2`, `clk3`), defined so that the first two are synchronous with each other, but asynchronous to `clk3`. `clk1` and `clk2` are assigned to clock group `group1`,

so that they are synchronous to each other but asynchronous to clk3, which remains in the default clock group.

Enable	Name	Object	Period	Waveform	Add	Clock Group	Latency	U
1 <input checked="" type="checkbox"/>	clk1	clk1	7		<input type="checkbox"/>	group1		
2 <input checked="" type="checkbox"/>	clk2	n:clk2	10		<input type="checkbox"/>	group1		
3 <input checked="" type="checkbox"/>	clk3	clk3	12		<input type="checkbox"/>	<default>		
4								
5								
6								

4. Use one of these `set_clock_groups` constructions to define clock groups for UMR clocks.

```
set_clock_groups -asynchronous -group {usrclk1} -group {usrclk2 ...}
set_clock_groups -aysnchronous -group {usrclk1 HAPS_umr_clk}
```

Avoid this construction:

```
set_clock_groups -asynchronous -group {usrclk1}
```

By default UMR clocks are synchronous to all other clocks, so any true user paths with UMRBus/CAPIM logic are visible. If these paths need to be constrained, do not inadvertently mask these paths when using `set_clock_groups`.

5. To time a path whose start point and end points are in asynchronous clock domains, use `reset_path` in conjunction with `set_clock_groups` and define the clock groups as shown below.
  - Put the asynchronous clocks in the same clock domain.
  - However, do not use the typical assignment shown below to make the four clocks asynchronous to each other, because `reset_path` does not override the `set_clock_groups` definition.

```
set_clock_groups -derive -asynchronous -name {my_group1} -group {clk1}
set_clock_groups -derive -asynchronous -name {my_group2} -group {clk2}
set_clock_groups -derive -asynchronous -name {my_group3} -group {clk3}
set_clock_groups -derive -asynchronous -name {my_group4} -group {clk4}
```

These constraints make the clock domains asynchronous to each other, but it does not allow for timing an individual path between clock domains.

- Instead define false paths between the asynchronous paths in the group. The following example shows how to define the clocks for a scenario where you have to time a signal that is generated in clk1 and then used in clk2:

```
set_clock_groups -derive -asynchronous -name {my_group1} -group
  {{clk1} {clk2}}
set_false_path -from [get_clocks {clk1}] -to [get_clocks {clk2}]
set_false_path -from [get_clocks {clk2}] -to [get_clocks {clk1}]
set_clock_groups -derive -asynchronous -name {my_group3} -group {clk3}
set_clock_groups -derive -asynchronous -name {my_group4} -group {clk4}
```

Defining paths between domains with `set_false_path` still accomplishes the goal of defining false paths between the domains, but `set_false_path` can be overridden by `reset_path`.

- Add FDC constraints to allow a path from a specific register that is clocked by clk1 (`my_async_rst` in the following example) to be timed with respect to clk2.

```
reset_path -from [get_pins {my_async_rst.Q[0]}] -to [get_clocks {clk2}]
set_max_delay -from [get_pins {my_async_rst.Q[0]}] -to [get_clocks {clk2}]
  80
```

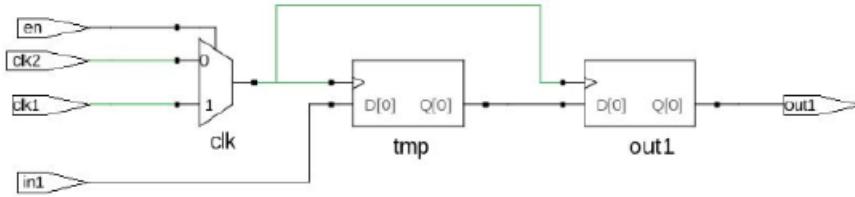
The `reset_path` exception times the path as a single-cycle path with respect to the clock definitions. The `set_max_delay` constraint lets you specify a specific delay, in this case 80ns.

6. To propagate exception constraints with various clocking structure elements, add the `-include_generated_clocks` argument to the constraints.

```
set_clock_groups -asynchronous -group [get_clocks -include_generated_clocks
GCLK2] -group [get_clocks -include_generated_clocks GCLK1]
```

## Defining Clocks for Clock Muxes

The definition of clocks that are to be muxed together varies slightly, depending on whether the clocks have the same frequency or not. The following procedures use this example as an illustration:



## Defining Muxed Clocks with Different Frequencies

If the clocks are asynchronous, separate clock paths must be defined, as described below.

1. Define the clocks with `create_clock` constraints.

For the example, two clocks are defined:

```
create_clock -name {clk1} [get_nets {clk1}] -period 10.0
            -waveform {0 5.0}
create_clock -name {clk2} [get_nets {clk2}] -period 10.0 -waveform {0 5.0}
```

2. Use multiple `set_clock_groups` constraints to mark them as asynchronous to each other:

```
set_clock_groups -derive -asynchronous -name {default_clkgroup_0} -group
                  [get_clocks {clk1}]
set_clock_groups -derive -asynchronous -name {default_clkgroup_1} -group
                  [get_clocks {clk2}]
```

3. Check the timing report.

For the example, the tool reports two separate clock paths, one for each clock.

## Defining Muxed Clocks with the Same Frequency

If the clocks have the same phase and frequency, follow this procedure to define the clocks.

1. Define the clock at the net connected to the output pin of the mux.

For example:

```
create_clock -name {clk} [get_nets {clk}] -period 10.0
            -waveform {0 5.0}
```

2. Define the mux output clock as asynchronous to all other clocks, using a `set_clock_groups` constraint:

```
set_clock_groups -derive -asynchronous -name {default_clkgroup_2} -group  
[get_clocks {clk}]
```

3. Check the timing report.

In this case, there should be a single clock path, instead of separate paths.

## Defining Input and Output Constraints

In addition to setting I/O delays in the constraints window as described in [Defining Clocks and Path Constraints from the GUI, on page 39](#), you can set them as follows:

- Open the constraints window, click Inputs/Outputs, and select the port (Port). You can set the constraint for
  - All inputs and outputs (globally in the top-level netlist)
  - For a whole bus
  - For single bits

You can specify multiple constraints for the same port. The software applies all the constraints; the tightest constraint determines the worst slack. If there are multiple constraints from different levels, the most specific overrides the more global. For example, if there are two bit constraints and two port constraints, the two bit constraints override the two port constraints for that bit. The other bits get the two port constraints.

- Specify the constraint value in the constraints window:
  - Select the type of delay: input or output (Type).
  - Type a delay value (Value).
  - Check the Enabled box, and save the constraint file in the project.

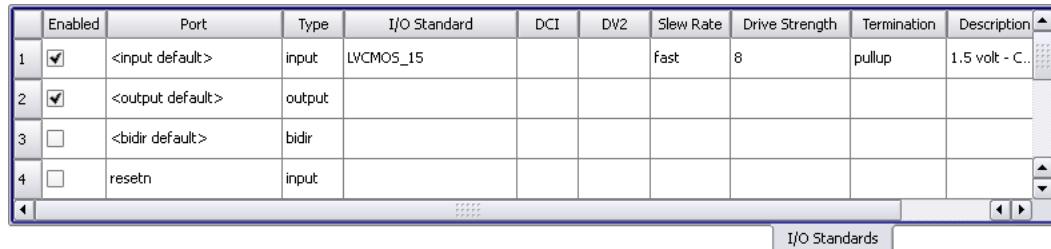
Make sure to specify explicit constraints for each I/O path you want to constrain.

- To determine how the I/O constraints are used during synthesis, do the following:
  - To use only the explicitly defined constraints set the autoconstrain\_io option to 0 (option set command).
  - To synthesize with all the constraints using the clock period for all I/O paths that do not have an explicit constraint enable, set the autoconstrain\_io option to 1.
  - Compile and map the design. When you forward-annotate the constraints to place and route, the constraints are forward-annotated.

- Input or output ports with explicitly defined constraints, but without a reference clock (-ref option) are included in the System clock domain and are considered to belong to every defined or inferred clock group.
- If you do not meet timing goals after place and route and you need to adjust the input constraints; do the following:
  - Open the constraints window with the input constraint.
  - Use the `set_clock_route_delay` command to translate the -route option for the constraint, so that you can specify the actual route delay (in nanoseconds), as obtained from the place-and-route results. Adding this constraint is equivalent to putting a register delay on the input register. Use it as a fudge factor to force the synthesis engine to accommodate a routing delay that is larger than expected.
  - Resynthesize your design.

## Specifying Standard I/O Pad Types

You can specify a standard I/O pad type to use in the design using the I/O Standards panel. The equivalent Tcl command is `define_io_standard`.



The screenshot shows a software interface for defining I/O standards. At the bottom right is a tab labeled "I/O Standards". Above it is a table with the following data:

	Enabled	Port	Type	I/O Standard	DCI	DV2	Slew Rate	Drive Strength	Termination	Description
1	<input checked="" type="checkbox"/>	<input default>	input	LVCMS_15			fast	8	pullup	1.5 volt - C...
2	<input checked="" type="checkbox"/>	<output default>	output							
3	<input type="checkbox"/>	<bidir default>	bidir							
4	<input type="checkbox"/>	resetn	input							

To define an I/O standard for any port appearing in the I/O Standards panel:

1. Open the constraints window and go to the I/O Standard tab.
2. In the Port column, select the port. This determines the port type in the Type column.
3. Enter an appropriate I/O pad type in the I/O Standard column. The Description column shows a description of the I/O standard you selected. Supported I/O standards are listed in the drop-down menu.
4. Where applicable, set other parameters such as drive strength, slew rate, and termination as outlined in the following table. You cannot set

the parameter values for industry I/O standards whose parameters are predefined for the standard.

Field	Description
Port	Specifies the name of the port from the pull-down list. The first two entries let you specify global input and output default delays, which you can then override with additional constraints on individual ports.
Type	Specifies the port direction.
I/O Standard	Supported I/O standards by Synopsys FPGA products.
DCI, DV2, Slew Rate, Drive Strength, Termination, Power Schmitt	The values for these parameters are based on the selected I/O standard.
Description	Describes the selected I/O Standard.
Comment	Enter comments about an I/O standard.

The software stores the pad type specification and the parameter values with the `syn_pad_type` attribute. When you synthesize the design, the I/O specifications are mapped to the appropriate I/O pads within the technology.

## Specifying Limits for FDC Wildcard Reporting

For FDC constraints specified with wildcard expressions, you can set a maximum limit to the number of such objects when they are reported in the constraint checker.

1. To reduce the number of objects, in the FDC file, specify the `set ::wildcard_report_limit` command with the value set to the upper limit you want to see.

The default is 1000, but the following command restricts the maximum to 150 objects:

```
set ::wildcard_report_limit 150
```

You can set it to 0, or any positive value that will fit into a 32-bit signed integer.

2. If you see want to see more objects than are reported (message MF892), you can increase the limit. For example:

```
set ::wildcard_report_limit 1000000000
```

## Using the TCL View of Constraints GUI

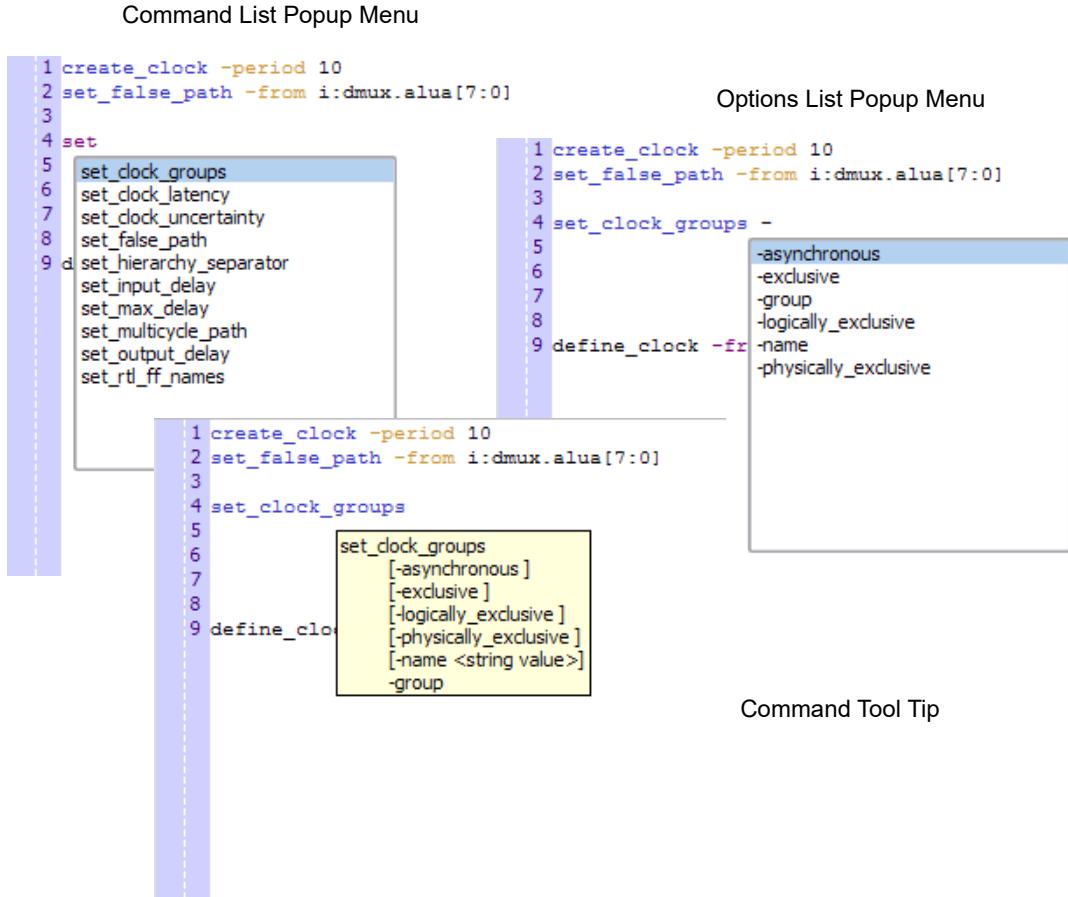
The TCL View of the constraints GUI is an advanced text file editor used for synthesis timing and design constraints. To use the TCL View of the constraints GUI, follow these steps:

1. Click on **TCL View**.

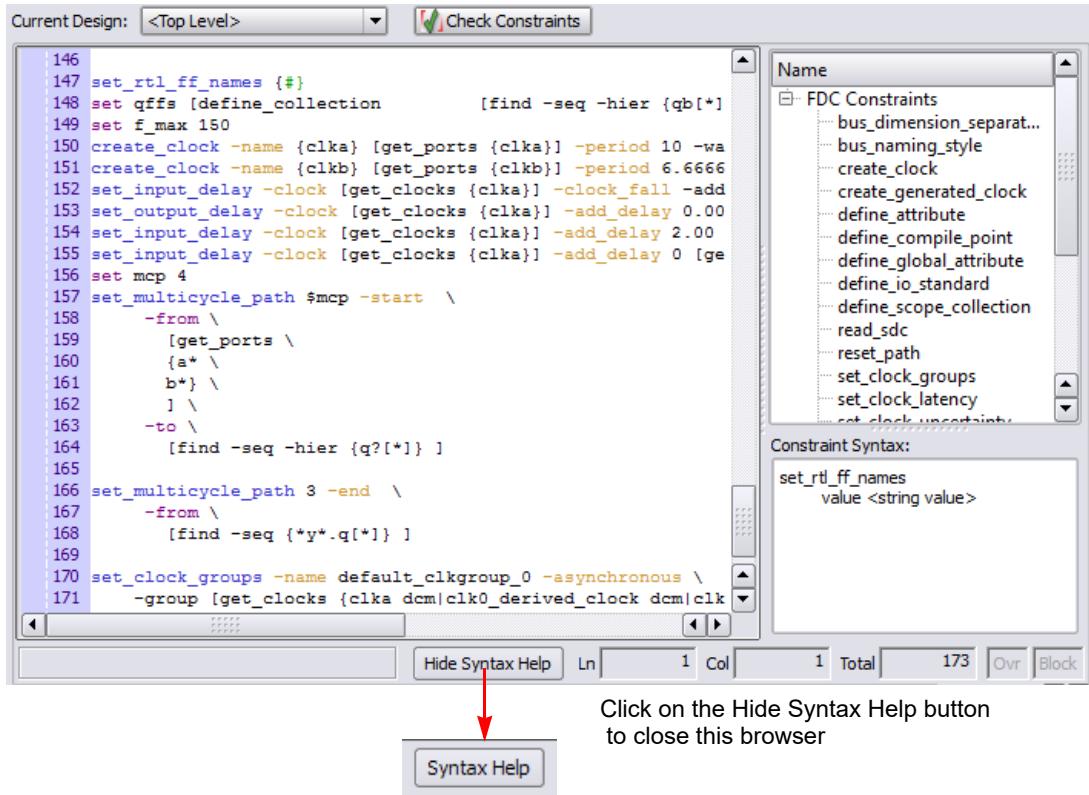
The text editor opens. It features standard copy, paste, comment and uncomment commands, and auto-complete. The window includes keyword highlighting and uses color indicators to validate the syntax. It also has keyword-based syntax help, displays parameter options for a command in popup lists, or displays the complete command syntax as a tool tip.

2. Specify FPGA design constraints as follows:

- Type the command; after you type three characters a popup menu displays the design constraint command list. Select a command.
- When you type a dash (-), the options popup menu list is displayed. Select an option.
- When you hover over a command, a tool tip is displayed for the selected commands.



3. Alternatively, specify a command by using the constraints browser that displays a constraints command list and associated syntax.
  - Double-click on the specified constraint to add the command to the editor window.
  - Then, use the constraint syntax window to help you specify the options for this command.
  - Click on the Hide Syntax Help button at the bottom of the editor window to close the syntax help browser.



- Save this file.

## Guidelines for Entering and Editing Constraints

- Enter or edit constraints as follows:
  - For attribute cells in the spreadsheet, click in the cell and select from the pull-down list of available choices.
  - For object cells in the spreadsheet, click in the cell and select from the pull-down list. When you select from the list, the objects automatically have the proper prefixes in the constraints window.

Alternatively, you can drag and drop an object from an HDL Analyst view into the cell, or type in a name. If you drag a bus, the software enters the whole bus (busA). To enter busA[3:0], select the appropriate

bus bits before you drag and drop them. If you drag and drop or type a name, make sure that the object has the proper prefix identifiers:

Prefix Identifiers	Description for ...
v: <i>design_name</i>	hierarchies or “views” (modules)
c: <i>clock_name</i>	clocks
i: <i>instance_name</i>	instances (blocks)
p: <i>port_name</i>	ports (off-chip)
t: <i>pin_name</i>	hierarchical ports, and pins of instantiated cells
b: <i>name</i>	bits of a bus (port)
n: <i>net_name</i>	internal nets

- For cells with values, type in the value or select from the pull-down list.
- Click the check box in the Enabled column to enable the constraint or attribute.
- Make sure you have entered all the essential information for that constraint. Scroll horizontally to check. For example, to set a clock constraint in the Clocks tab, you must fill out Enabled, Clock, Period, and Clock Group. The other columns are optional. For details about setting different kinds of constraints, go to the appropriate section listed in [Specifying FDC Constraints, on page 20](#).

2. For common editing operations, refer to this table:

To ...	Do ...
Cut, copy, paste, undo, or redo	Select the command from the popup (hold down the right mouse button to get the popup) or from the Edit menu.
Copy the same value down a column	Select Fill Down (Ctrl-d) from the Edit or popup menus.
Insert or delete rows	Select Insert Row or Delete Rows from the Edit or popup menus.
Find text	Select Find from the Edit or popup menus. Type the text you want to find, and click OK.

3. Edit your constraint file, if needed. If your naming conventions do not match these defaults, add the appropriate command specifying your naming convention to the beginning of the file, as shown in these examples:

	<b>Default</b>	<b>You use</b>	<b>Add this to your file</b>
Hierarchy separator	A.B	Slash: A/B	set_hierarchy_separator {}
Naming bit 5 of bus ABC	ABC[5]	Underscore	bus_naming_style {%s_%d}
Naming row 2 bit 3 of array ABC [2x16]	ABC [2] [3]	Underscore ABC [2_3]	bus_dimension_separator_style {}

# Specifying Timing Exceptions

You can specify the following timing exception constraints, either from the constraints GUI or by manually entering the Tcl commands in a file:

- Multicycle Paths – Paths with multiple clock cycles.
- False Paths – Clock paths that you want the synthesis tool to ignore during timing analysis and assign low (or no) priority during optimization.
- Max Delay Paths – Point-to-point delay constraints for paths.

By default, exception constraints are not propagated through MMCM/PLL or to generated/derived clocks. To propagate exception constraints with various clocking structure elements including MMCMs, BUFGs, BUFGCE\_DIVs, BUFG\_MUXEs, GCCs, XMRs, TDMs, and the clocks generated by flip-flops, add the `-include_generated_clocks` argument to the constraint.

The following shows you how to specify timing exceptions in the constraints GUI.

- [Defining From/To/Through Points for Timing Exceptions](#), on page 56
- [Defining Multicycle Paths](#), on page 62
- [Defining False Paths](#), on page 63

## Defining From/To/Through Points for Timing Exceptions

For multicycle path, false path, and maximum path delay constraints, you must define paths with a combination of from/to/through points. For more information about defining these points, see the following:

- [Defining From and To Points](#), on page 56
- [Defining Through Points](#), on page 59

### Defining From and To Points

1. Determine the appropriate object for the from/to point.

You must have a valid object. A from point sets the starting point for a timing exception and a to point sets the ending point for a timing exception. You can set these constraints on the following objects:

From Points	To Points
Clocks. See step 4 for more information.	Clocks. See step 4 for more information.
Registers	Registers
Top-level input or bi-directional ports	Top-level output or bi-directional ports
Instantiated library primitive cells (gate cells)	Instantiated library primitive cells (gate cells)
Black box outputs	Black box inputs

You can specify the from/to point by entering the syntax in an fdc constraint file (step 2) or in the SCOPE editor (step 3).

2. To specify the constraint in an fdc file, do the following:

- Add the constraint to an fdc file using the appropriate multicycle path, false path or max delay path syntax.
- Specify the kind of object in the constraint specification by explicitly specifying the object type with the appropriate prefix: n: (net) or i: (instance). For example:

```
set_multicycle_path -from {i:aq i:bq} 2
set_multicycle_path -from {i:aq i:bq} -through {n:xor_all} 2
```

- Specify multiple from/to points in a single exception by enclosing them in square brackets.

For example, you can specify constraints that apply to all the bits in a bus: from A[0:15]. You can also specify multiple starting and ending points in the same constraint:

```
set_multicycle_path -from {i:inst2.lowreg_output[7]} -to
{i:inst1.DATA0[7]} 2
```

From/to constraints apply from any of the defined start points to any of the end points. If you set from A[0:15] to B[0:15], it defines a timing exception that applies to all 16 \* 16, or 256 combinations of start/end points.

3. To specify a from/to point in the SCOPE GUI, do the following:

- Select the first point from the HDL Analyst view (see step 1), then drag and drop this instance into the From/To cell in the constraints editor. For each subsequent instance, press the Shift key as you drag

and drop the instance into the From/To cell in the constraints editor. For example, valid Tcl command format include:

- You do not need to explicitly specify the object type prefix; the SCOPE interface adds this automatically.
  - Specify multiple from/to points in a single exception by enclosing them in square brackets, as described in the previous step.
4. To specify clocks as from/to points in timing exception constraints, use this syntax:

```
set_multicycle_path | false_path | max_delay -from | -to { c:clock_name [:r | f] }
```

- Multicycle clock constraints apply to all registers clocked by the specified clock. The following multicycle constraint allows two clock periods for all paths from the rising edge of the flip-flops clocked by clk1:

```
set_multicycle_path -from {c:clk1:r} 2
```

- False path clock constraints apply to registers clocked by the specified clock. The following false path constraint disables all paths from the rising edge of the flip-flops clocked by clk1:

```
set_false_path -from {c:clk1:r}
```

- Path delay clock constraints apply to all paths of the registers clocked by the specified clock. This path delay constraint sets a max delay of 2 ns on all paths to the falling edge of the flip-flops clocked by clk1:

```
set_max_delay -to {c:clk1:f} 2
```

- You cannot specify a clock as a through point. However, you can set a constraint from or to a clock and through an object (net, pin, or hierarchical port). The following constraint allows two clock periods for all paths to the falling edge of the flip-flops clocked by clk1 and through bit 9 of the hierarchical net:

```
set_multicycle_path -to {c:clk1:f} -through  
{n:MYINST.mybus2[9]} 2
```

## Defining Through Points

You can specify a through point in one of the ways listed in the table.

Single point	Applies to any path passing through the specified point
Single list of points	Works as an OR and applies to any path that passes through any point in the list
Multiple through points	Works as an AND and applies to all paths that pass through the specified points
Multiple lists of points	Works as an AND/OR and applies to any paths that pass through combinations of points (AND) from all lists. See the procedure below for details.

See the steps below for details:

1. Determine the appropriate object for the through point.

You must select a valid object. Set through points on combinational nets, hierarchical ports, or pins of instantiated cells. You can define the through point by entering the appropriate constraint syntax or by using the GUI.

2. To specify the constraint in an fdc file, do the following:

- Add the constraint to an fdc file using the appropriate multicycle path, false path or max delay path syntax.
- Specify the kind of object in the constraint specification by explicitly specifying the object type with the appropriate prefix: n: (net), i: (instance), t: (hierarchical port), or p: (top-level port). For example:

```
set_multicycle_path -through {i:aq i:bq} 2
```

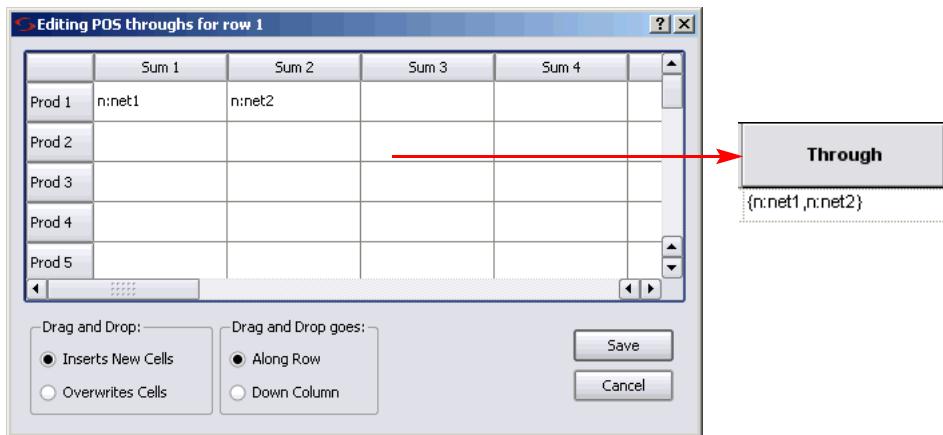
If you do not use the prefix to identify a net, the constraint will not apply to valid nets.

- Use any of the following ways to specify the through point. The constraint applies to all paths that pass through the specified points. When multiple points are specified, it applies to any path that passes through any of the listed points.

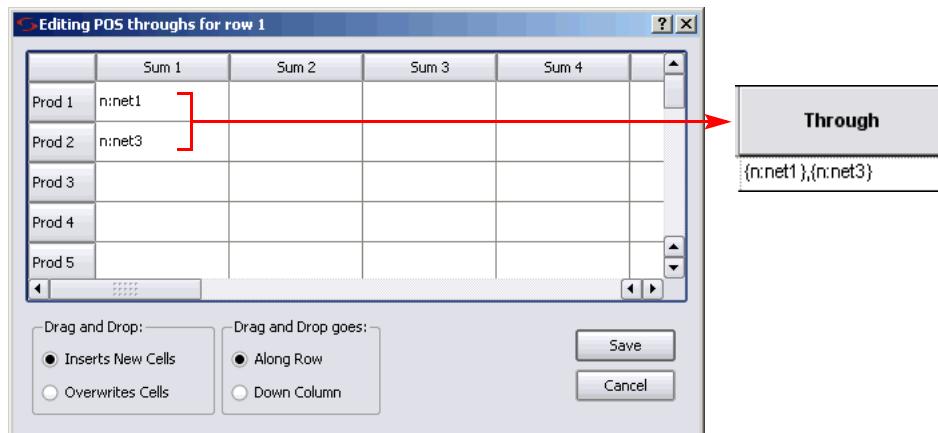
The following table shows some examples:

Single point	Use the -through option: <code>set_false_path -through n:regs_mem[2]</code> <code>set_false_path -through [get_nets {regs_mem[2]}]</code>
Single list of points (OR)	Use the -through option to specify a list of points: <code>set_max_delay -through {t:regs_mem[2] t:prgcntr.pc[7] t:dmux.alub[0]} 5</code>
Multiple through points (AND)	Use a separate -through option for each point: <code>set_max_delay -through t: regs_mem[2] -through t:prgcntr.pc[7] -through t:dmux.alub[0] 5</code>
Multiple lists of points (AND/OR)	Use a separate -through option for each list of points: <code>set_multicycle_path -through {n:net1 n:net2} -through {n:net3 n:net4} 2</code> The example sets a 2-cycle constraint on all paths that pass through net1 and net3 or net1 and net4 or net2 and net3 or net2 and net4.

- Combine through points with from/to points in a single constraint as needed.
3. To specify a through point from the GUI, follow these steps:
- Open the SCOPE editor, click in the Through field and click the arrow. This opens the Product of Sums (POS) interface.

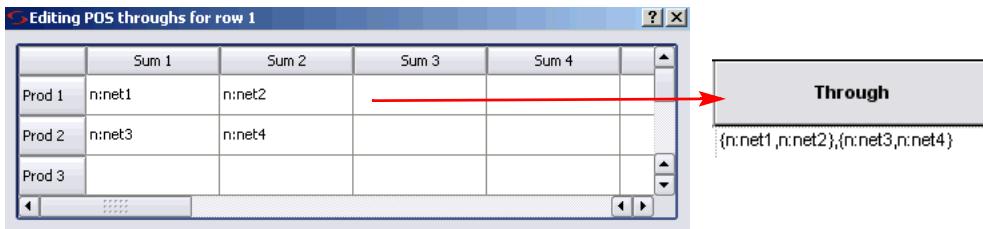


- Either type the net name with the n: prefix in the first cell or drag the net from an HDL Analyst view into the cell. If you specify n:net1, the through constraint applies to any path passing through net1.
  - Specify multiple through points or lists of points as described in step 4.
  - Combine through points with from/to points in a single constraint as needed.
4. Do the following in the POS GUI to specify multiple through points or lists of points.
- To specify a single list of points (OR), type the net name or drag and drop the net as described in step 4. Repeat this step along the same row, adding other nets in the Sum columns. The nets in each row form an OR list. The constraint applies to any path passing through any of the specified nets. In the example shown in the previous figure, the constraint applies to any path that passes through net1 or net2.
- Alternatively, select Along Row in the constraints editor POS interface. In an HDL Analyst view, select all the nets you want in the list of through points. Drag the selected nets and drop them into the POS interface. The tool fills in the net names along the row. The nets in each row form an OR list.
- To specify multiple through points (AND), type the names in the Through field or drag the net from an HDL Analyst view into the cell as described in step 4. Repeat this step down the same Sum column. The constraint works as an AND function and applies to any path passing through all the specified nets.



Alternatively, select Down Column in the constraints editor POS interface. In an HDL Analyst view, select all the nets you want in the list of through points. Drag the selected nets and drop them into the POS interface. The tool fills in the net names down the column.

- To specify an AND/OR constraint for a list of through points, type the names in the Through field or create multiple lists as described previously.



In this example, the software applies the constraint to the paths through all points in the lists as follows:

```
net1 AND net3
OR net1 AND net4
OR net2 AND net3
OR net2 AND net4
```

- Click Save.

## Defining Multicycle Paths

To define a multi-cycle path constraint, use the Tcl `set_multicycle_path` command, or select the constraints editor Delay Paths tab and do the following;

- From the Delay Type pull-down menu, select Multicycle.
- Select a port or register in the From or To columns, or a net in the Through column. You must set at least one From, To, or Through point. You can use a combination of these points. See [Defining Through Points, on page 59](#) for more information.
- Select another port or register if needed (From/To/Through).
- Type the number of clock cycles or nets (Cycles).

5. Specify the clock period to use for the constraint by going to the Start/End column and selecting either Start or End.

If you do not explicitly specify a clock period, the software uses the end clock period. The constraint is now calculated as follows:

$$\text{multicycle\_distance} = \text{clock\_distance} + (\text{cycles} - 1) * \text{reference\_clock\_period}$$

In the equation, `clock_distance` is the shortest distance between the triggering edges of the start and end clocks, `cycles` is the number of clock cycles specified, and `reference_clock_period` is either the specified start clock period or the default end clock period.

6. Check the Enabled box.

## Defining False Paths

You define false paths by setting constraints explicitly on the Delay Paths tab or implicitly on the Clock tab. See [Defining Through Points, on page 59](#) for object naming and specifying through points.

1. To define a false path between ports or registers, select the constraints editor Delay Paths tab, and do the following:
  - From the Delay Type pull-down menu, select False.
  - Use the pull-down to select the port or register from the appropriate column (From/To/Through).
  - Check the Enabled box.

The software treats this as an explicit false constraint and assigns it the highest priority. Any other constraints on this path are ignored.

2. To define a false path between two clocks, select the constraints editor Clocks tab, and assign the clocks to different clock groups:

The software implicitly assumes a false path between clocks in different clock groups. This false path constraint can be overridden by a maximum path delay constraint, or with an explicit constraint.

# Conflict Resolution for Timing Exceptions

The term *timing exceptions* refers to false path, max path delay, and multicycle path timing constraints. If there are conflicts in the way timing exceptions are specified in the constraint file, the tool uses some priority rules to resolve them. There are four tiers where conflicts can arise, so accordingly, there are four conflict resolution categories, with one being the highest level. This table below summarizes conflict resolution for constraints.

Level	Constraint Conflict	Priority	For Details, See ...
1	Different timing exceptions set on the same object	1 – False path 2 – Path delay 3 – Multicycle path	<i>Conflicting Timing Exceptions , on page 64</i>
2	Timing exceptions of the same constraint type, using different semantics (from/to/through)	1 – From 2 – To 3 – Through	<i>Same Constraint Type with Different Semantics , on page 66</i>
3	Timing exceptions of the same constraint type using the same semantics, but set on different objects	1 – Ports/instances/pins 2 – Clocks	<i>Same Constraint and Semantics with Different Objects , on page 67</i>
4	Identical timing constraints with different constraint values	Tightest, or most constricting constraint	<i>Identical Constraints with Different Values , on page 67</i>

In addition to the four levels of conflict resolution for timing exceptions, there are priorities for the way the tool handles multiple I/O delays set on the same port and implicit and explicit false path constraints. For information on resolving these types of conflicts, see *Conflict Resolution for Multiple I/O Constraints, on page 29*.

## Conflicting Timing Exceptions

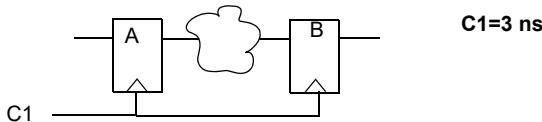
The first (and highest) level of resolution occurs when timing exceptions conflict with each other. The tool follows this priority for applying timing exceptions:

1. False path

2. Path delay
3. Multicycle path

For example:

```
set_false_path -from {c:C1:r}  
set_max_delay -from {i:A} -to {i:B} 10  
set_multicycle_path -from {i:A} -to {i:B} 2
```



These constraints conflict because the path from A to B has three different constraints set on it. With these types of conflict, the false path constraint (`set_false_path`) is honored because it has the highest priority of all timing exceptions and the other timing exceptions are ignored.

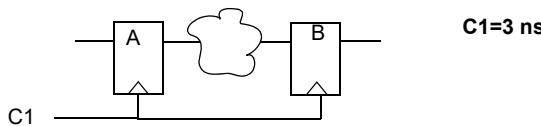
## Same Constraint Type with Different Semantics

The second level of resolution occurs when there are multiple timing exceptions of the same constraint type, but which use different semantics (from/to/through). This is the priority for these conflicts:

1. from
2. to
3. through

If there are two multicycle constraints set on the same path, one specifying a from point and the other specifying a to point, the constraint using -from takes precedence, as in the following example.

```
set_multicycle_path -from {i:A} 3
set_multicycle_path -to {i:B} 2
```



In this case, the -to constraint is ignored even though it sets a tighter constraint and the tool uses the -from constraint because it has higher priority:

```
set_multicycle_path -from {i:A} 3
```

## Same Constraint and Semantics with Different Objects

The third level of constraint conflict arises when multiple timing exceptions of the same type use the same semantic, but are set on different objects. The priority for design objects is as follows:

1. Ports/instances/pins
2. Clocks

In the following example above, the tool uses the first constraint set on the instance and ignores the constraint set on the clock from i:mac1.datax[0], even though the clock constraint is tighter. It ignores the constraint on the clock because this constraint has a lower priority.

```
set_multicycle_path -from {i:mac1.datax[0]} -start 4  
set_multicycle_path -from {c:clk1:r} 2
```

For details on how the tool prioritizes multiple I/O delays set on the same port or implicit and explicit false path constraints, see [Conflict Resolution for Multiple I/O Constraints, on page 29](#).

## Identical Constraints with Different Values

Where timing constraints are identical except for the constraint value, the tool uses the tightest or most constricting constraint. In the following example, the tool uses the constraint specifying two clock cycles:

```
set_multicycle_path -from {i:special_regs.trisa[7:0]} 2  
set_multicycle_path -from {i:special_regs.trisa[7:0]} 3
```

# Finding Objects with Tcl find and expand

The `Tcl find` and `expand` commands are powerful search tools that you can use to quickly identify the objects you want. The following sections describe how to use these commands effectively:

- [Specifying Search Patterns for `Tcl find`, on page 68](#)
- [Refining `Tcl Find` Results with `-filter`, on page 70](#)
- [Using the `Tcl Find` Command to Define Collections, on page 71](#)
- [Using the `Tcl expand` Command to Define Collections, on page 72](#)
- [`Tcl find` Command, on page 372 in the \*Command Reference\*](#)

Once you have located objects with the `find` or `expand` commands, you can group them into collections, as described in [Using Collections, on page 74](#), and apply constraints to all the objects in the collection at the same time.

## Specifying Search Patterns for `Tcl find`

The usage tips in the following table apply for `Tcl find` search patterns, regardless of whether you specify the `find` command in the constraints window or as a `Tcl` shell command. For full details of the command syntax, refer to [`Tcl find` Command, on page 372](#) in the *Command Reference*.

Case rules	<p>Use the case rules for the language from which the object was generated:</p> <ul style="list-style-type: none"><li>• VHDL: case-insensitive</li><li>• Verilog: case-sensitive. Make sure that the object name you type in the constraints window matches the Verilog name.</li></ul> <p>For mixed language designs, use the case rules for the parent module. The top level for this example is VHDL, so the following command finds any object in the current view that starts with either a or A:</p> <pre>find {a*} -nocase</pre>
Pattern matching	<p>You have two pattern-matching choices:</p> <ul style="list-style-type: none"><li>• Specify the <code>-regexp</code> argument, and then use regular expressions for pattern matching.</li><li>• Do not specify <code>-regexp</code>, and use only the * and ? wildcards for pattern matching.</li></ul> <p>For hierarchical instance names that use dots as separators, the dots must be escaped with a backward slash (\). For example: abc\l.d.</p>
Scope of the search	<p>The scope of the search varies, depending on where you enter the command. If you enter it in the constraints editor environment, the scope of the search is the entire database, but if it is entered in the Tcl shell, the default scope of the search is the current HDL Analyst view. See <a href="#">Comparison of Methods for Defining Collections , on page 74</a> for a list of the differences.</p> <p>To set the scope to include the hierarchical levels below the current view in HDL Analyst, use the <code>-hier</code> argument. This example finds all objects below the current view that begin with a:</p> <pre>find {a*} -hier</pre>

---

Restricting search by type of object	Use the <code>-objectType</code> argument. The following command finds all nets that contain syn:
Restricting search by object property	Use the <code>-filter</code> option, as described in <a href="#">Refining Tcl Find Results with -filter , on page 70</a> .
Extending search through the hierarchy	Use the <code>-flat</code> option. With this option, the <code>*</code> wildcard matches hierarchy separators as well as regular characters. In the following example, the command finds all instances that include <code>fft_stages</code> in the name, whether it just matches an instance name ( <code>inst1_fft_stages_2</code> ) or matches a hierarchical name that includes the hierarchy separator ( <code>a1.fft_stages_xy</code> ):

---

`find -net {*syn*}`

---

`find -seq -flat *fft_stages* -print`

---

## Refining Tcl Find Results with -filter

The `-filter` option of the `find` command lets you further refine the objects located by the `find` command, according to their properties. When used with other commands, it can be a powerful tool for generating statistics and for evaluation. To filter your `find` results, follow these steps:

1. Specify the command using the `find` pattern as usual, and then specify the `-filter` option as the last argument:

```
find searchPattern -filter expression
find searchPattern -filter !expression
```

With this command, the tool first finds objects that match the `find searchPattern`, and then further filters the found objects the according to the property criteria specified in `-filter expression`. Use the `!` character before `expression` if you want to select objects that do not match the properties specified in the filter `expression`.

`expression` can be a property name, specified as `@propertyName`, or a property name and value pair, specified as `@propertyName operator value`.

The following example finds registers in the current view that are clocked by `myclk`:

```
find -seq {*} -filter {@clock==myclk}
```

For further information about the command, see the following:

For ...	See
Tips on using find search patterns	<i>Specifying Search Patterns for Tcl find , on page 68</i>
find syntax details	<i>Tcl find Command , on page 372</i> in the <i>Command Reference</i>
find -filter syntax details	<i>find -filter , on page 385</i> in the <i>Command Reference</i>

## Examples of Useful Find -filter Commands

To find ...	Use a command like this example ...
Instances by slack value	set slack [find -hier -inst {*} -filter @slack <= {-1.000}]
Instances with negative slack	set negFF [find -hier -inst {*} -filter @slack <= {0.0}]
Instances within a slack range	set slackRange [find -hier -inst {*} -filter @slack <= {-1.000} && @slack >= {+1.000}]
Pins by fanout value	set pinResult [find -pin *.CE -hier -filter {@fanout > 15 && @slack < 0.0} -print]
Sequential elements within a clock domain	set clk1FF [find -hier -seq * -filter {@clock==clk1}]
Sequential components by primitive type	set fdrse [find -hier -seq {*} -filter @view=={FDRSE}]

## Using the Tcl Find Command to Define Collections

It is recommended that you use the constraints GUI rather than the Tcl shell described here to specify the find command, for the reasons described in *Comparison of Methods for Defining Collections, on page 74*.

The Tcl find command returns a collection of objects. If you want to create a collection of connectivity-based objects, use the Tcl expand command instead of find (*Specifying Search Patterns for Tcl find , on page 68*). This section lists some tips for using the Tcl find command.

1. Create a collection by typing the set command and assigning the results to a variable. The following example finds all instances with a primitive type DFF and assigns the collection to the variable \$result:

```
set result [find -hier -inst {*} -filter @ view == DFF]
```

The result is a random number like `s:49078472`, which is the collection of objects found. The following table lists some usage tips for specifying the `find` command. For full details of the syntax, refer to [Tcl find Command, on page 372](#) in the *Command Reference Manual*.

2. Check your `find` constraints.
3. Once you have defined the collection, you can view the objects in the collection, using one of the following methods, which are described in more detail in [Viewing and Manipulating Collections with Tcl Commands, on page 80](#):
  - Print the collection using the `-print` option to the `find` command.
  - Print the collection without carriage returns or properties, using `c_list`.
  - Print the collection in columns, with optional properties, using `c_print`.
4. To manipulate the objects in the collection, use the commands described in [Viewing and Manipulating Collections with Tcl Commands, on page 80](#).
5. Combine the Tcl `find` command with other commands:

To ...	Combine with ...
Create or copy objects; create collections	<code>set</code> <code>define_collection</code>
Generate reports for evaluation	<code>c_list</code> <code>c_print</code>
Generate statistics	<code>c_info</code>

## Using the Tcl `expand` Command to Define Collections

The Tcl `expand` command returns a list of objects that are logically connected between the specified expansion points. This section contains tips on using the Tcl `expand` command to generate a collection of objects that are related by their connectivity.

1. Specify at least one `from`, `to`, or `thru` point as the starting point for the command. You can use any combination of these points.

The following example expands the cone of logic between `reg1` and `reg2`.

```
expand -from {i:reg1} -to {i:reg2}
```

If you only specify a thru point, the expansion stops at sequential elements. The following example finds all elements in the transitive fanout and transitive fanin of a clock-enable net:

```
expand -thru {n:cen}
```

2. To specify the hierarchical scope of the expansion, use the **-hier** argument.

If you do not specify this argument, the command only works on the current view. The following example expands the cone of logic to reg1, including instances below the current level:

```
expand -hier -to {i:reg1}
```

If you only specify a thru point, you can use the **-level** argument to specify the number of levels of expansion. The following example finds all elements in the transitive fanout and transitive fanin of a clock-enable net across one level of hierarchy:

```
expand -thru {n:cen} -level 1
```

3. To restrict the search by type of object, use the **-object\_type** argument.

The following command finds all pins driven by the specified pin.

```
expand -pin -from {t:i_and3.z}
```

4. To print a list of the objects found, either use the **-print** argument for the **expand** command, or use the **c\_print** or **c\_list** commands (see [Creating Collections using Tcl Commands, on page 77](#)).

# Using Collections

A collection is a defined group of objects. The advantage offered by collections is that you can operate on all the objects in the collection at the same time. A collection can consist of a single object, multiple objects, or even other collections. You can either define collections in the constraints GUI or type the commands in the Tcl shell.

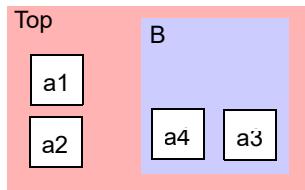
- [Creating and Using Constraints Editor Collections](#), on page 75
- [Creating Collections using Tcl Commands](#), on page 77
- [Viewing and Manipulating Collections with Tcl Commands](#), on page 80

## Comparison of Methods for Defining Collections

You can enter the find and expand Tcl commands that are used to define collections in either the Tcl shell or in the constraints editor. It is recommended that you use the constraints GUI for the reasons outlined below:

	Constraints GUI	Tcl Shell
Database used	Top level; includes all objects.  See the example below.	Current Analyst view, which might be a lower-level view. If the current view is the Technology view after mapping, objects might be renamed, replicated, or removed.
Persistence	Collection saved in project file.	Collection only valid for the current session; you must redefine it the next time you open the project.
Constraints	Can apply to collection.	Cannot apply to collection.

In the design shown below, if you push down into B, and then type `find -hier a*` in the Tcl shell, the command finds a3 and a4. However if you cut and paste the same command into the constraints editor Collections tab, your results would include a1, a2, a3, and a4, because the constraints GUI uses the top-level database and searches the entire hierarchy.



## Creating and Using Constraints Editor Collections

The following procedure shows you how to define collections in the constraints GUI. The constraints editor method is preferred over typing the commands in the Tcl shell ([Creating Collections using Tcl Commands, on page 77](#)) for the reasons described in [Comparison of Methods for Defining Collections, on page 74](#).

1. Define a collection by doing the following:
  - Open the constraints editor and click the Collections tab.
  - In the Name column, type a name for the collection.

	Enable	Name	Command
1	<input checked="" type="checkbox"/>	find_all	find -hier -inst {*usbSlaveControl.u_endpMux.*}
2	<input checked="" type="checkbox"/>	find_reg	find -hier -seq {*usbSlaveControl.u_endpMux.*}
3	<input checked="" type="checkbox"/>	find_comb	find -hier -inst {*usbSlaveControl.u_endpMux.*} -filter@is_combination

- In the Command column, enter the command. Additional information about specifying search patterns is described in [Specifying Search Patterns for Tcl find, on page 68](#).

You can also paste in a command. If you cut and paste a Tcl Find command from the Tcl shell into the constraints editor Collections tab, remember that the constraints GUI works on the top-level database, while the `find` command in the Tcl shell works on the current level displayed in the Analyst view.

Objects in a collection do not have to be of the same type. The collections shown in the preceding figure do the following:

Collection	Finds ...
find_all	All components in the module endpMux
find_reg	All registers in the module endpMux
find_comb	All combinatorial components under endpMux

The collections you define appear in the constraints editor pull-down object lists, so you can use them to define constraints.

2. To create a collection that is made up of other collections, do this:
  - Define the collections as described in the previous step. These collections must be defined before you can concatenate them or add them together in a new collection.
  - To concatenate collections or add to collections, type a name for the new collection in the Name column. Type the appropriate operator command like c\_union or c\_diff in the Command column. See [Creating Collections using Tcl Commands, on page 77](#) for a list of available commands.

The software saves the collection information in the constraint file for the project.

3. To apply constraints to a collection do the following:
  - Define a collection as described in the previous steps.
  - Go to the appropriate constraints editor tab and specify the collection name where you would normally specify the object name. Collections defined in the constraints GUI are available from the pull-down object lists. The following figure shows the collections defined in step 1 available for setting a false path constraint.

	Enabled	Delay Type	From	To	Through	Start/End	Cycles	Max Delay(ns)	Comment
1	<input checked="" type="checkbox"/>	False	special_regs.status[7:0]						
2			i:decode(opcode_goto						
3			i:decode(opcode_call						
4			i:decode(opcode_retw						
5			i:decode.skip						
6			i:decode.decodes[13:0]						
			i:prgmcntr..r_0[10:0]						
			i:prgmcntr..r_1[10:0]						
			i:prgmcntr..r_2[10:0]						
			i:prgmcntr..r_3[10:0]						
			i:prgmcntr..r_4[10:0]						

- Specify the rest of the constraint as usual. The software applies the constraint to all the objects in the collection.

## Example: Attribute Attached to a Collection

The following example shows the `xc_area_group` attribute applied to `$find_reg`, which results in all the registers in this collection being placed in the same region. Check the `srr` file and the netlist to see that the attribute is honored.

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description	Comment
1	<input checked="" type="checkbox"/>	instance	\$find reg	xc_area_group	rr#cc#rr#cc	string	Specifies the region where instance should be placed	
2								
3								
4								
5								
6								

Attributes

## Creating Collections using Tcl Commands

This section describes how to use the Tcl collection commands at the Tcl shell or in a script instead of entering them in the constraints GUI ([Creating and Using Constraints Editor Collections, on page 75](#)). There are differences in operation depending on where the collection commands are entered, and it is recommended that you use the constraints GUI, for the reasons described in [Comparison of Methods for Defining Collections, on page 74](#).

- To create a collection using a Tcl shell command, name it with the `set` command and assign it to a variable.

A collection can consist of individual objects, Tcl lists (which can consist of a single element), or other collections. You can embed the Tcl find and expand commands in the set command to locate objects for the collection (see [Using the Tcl Find Command to Define Collections, on page 71](#) and [Specifying Search Patterns for Tcl find, on page 68](#)). The following example creates a collection called my\_collection which consists of all the modules (views) found by the embedded find command:

```
set my_collection [find -view {*}]
```

2. To create collections derived from other collections, do the following:

- Define a new variable for the collection.
- Create the collection with one of the operator commands from this table:

To ...	Use this command ...
Add objects to a collection	c_union. See <a href="#">Examples: c_union Command , on page 79</a>
Concatenate collections	c_union. See <a href="#">Examples: c_union Command , on page 79.</a>
Isolate differences between collections	c_diff. See <a href="#">Examples: c_diff Command , on page 79.</a>
Find common objects between collections	c_intersect. See <a href="#">Examples: c_intersect Command , on page 79.</a>
Find objects that belong to just one collection	c_symdiff. See <a href="#">Examples: c_symdiff Command , on page 80.</a>

3. If your Tcl collection includes instances that use special characters, make sure to use extra curly braces or use a backslash to escape the special character.

Curly Braces {}	<pre>define_collection GRP_EVENT_PIPE2 {find -seq     {EventMux[2].event_inst?_sync[*]} -hier} define_collection mytn {find -inst {i:count1.co[*]}}</pre>
-----------------	---

Backslash Escape Character (\)	<pre>define_collection mytn {find -inst i:count1.co[*]}</pre>
--------------------------------	---

Once you have created a collection, you can do various operations on the objects in the collection (see [Viewing and Manipulating Collections with Tcl Commands, on page 80](#)), but you cannot apply constraints to the collection.

## Examples: c\_union Command

This example adds the reg3 instance to collection1, which contains reg1 and reg2 and names the new collection sumCollection.

```
set sumCollection [c_union $collection1 {i:reg3}]
c_list $sumCollection
{"i:reg1" "i:reg2" "i:reg3"}
```

If you added reg2 and reg3 with the c\_union command, the command removes the redundant instances (reg2) so that the new collection would still consist of reg1, reg2, and reg3.

This example concatenates collection1 and collection2 and names the new collection combined\_collection:

```
set combined_collection [c_union $collection1 $collection2]
```

## Examples: c\_diff Command

This example compares a list to a collection (collection1) and creates a new collection called subCollection from the list of differences:

```
set collection1 {i:reg1 i:reg2}
set subCollection [c_diff $collection1 {i:reg1}]
c_print $subCollection
"i:reg2"
```

You can also use the command to compare two collections:

```
set reducedCollection [c_diff $collection1 $collection2]
```

## Examples: c\_intersect Command

This example compares a list to a collection (collection1) and creates a new collection called interCollection from the objects that are common:

```
set collection1 {i:reg1 i:reg2}
set interCollection [c_intersect $collection1 {i:reg1 i:reg3}]
c_print $interCollection
"i:reg1"
```

You can also use the command to compare two collections:

```
set common_collection [c_intersect $collection1 $collection2]
```

### Examples: c\_symdiff Command

This example compares a list to a collection (collection1) and creates a new collection called diffCollection from the objects that are different. In this case, reg1 is excluded from the new collection because it is in the list and collection1.

```
set collection1 {i:reg1 i:reg2}
set diffCollection [c_symdiff $collection1 {i:reg1 i:reg3}]
c_list $diffCollection
{"i:reg2" "i:reg3"}
```

You can also use the command to compare two collections:

```
set symdiff_collection [c_symdiff $collection1 $collection2]
```

### Examples: Names with Special Characters

Your instance names might include special characters, as for example when your HDL code uses a generate statement. If your instance names have special characters, do the following:

Make sure that you include extra curly braces {}, as shown below:

```
define_collection GRP_EVENT_PIPE2 {find -seq
    {EventMux\[2\].event_inst?_sync[*]} -hier}
define_collection mytn {find -inst {i:count1.co[*]}}
```

Alternatively, use a backslash to escape the special character:

```
define_collection mytn {find -inst i:count1.co\[*\]}
```

## Viewing and Manipulating Collections with Tcl Commands

The following section describes various operations you can do on the collections you defined.

1. To view the objects in a collection, use one of the methods described in subsequent steps:
  - Select the collection in an HDL Analyst view (step 2).
  - Print the collection without carriage returns or properties (step 3).

- Print the collection in columns (step 4).
  - Print the collection in columns with properties (step 5).
2. To select the collection in an HDL Analyst view, type `select <collection>`.  
 For example, `select $result` highlights all the objects in the `$result` collection.
3. To print a simple list of the objects in the collection, uses the `c_list` command, which prints a list like the following:
- ```
{i:EP0RxFifo.u_fifo.dataOut[0]} {i:EP0RxFifo.u_fifo.dataOut[1]}
{i:EP0RxFifo.u_fifo.dataOut[2]} ...
```

The `c_list` command prints the collection without carriage returns or properties. Use this command when you want to perform subsequent Tcl commands on the list. See [Example: c\\_list Command, on page 83](#).

4. To print a list of the collection objects in column format, use the `c_print` command. For example, `c_print $result` prints the objects like this:

```
{i:EP0RxFifo.u_fifo.dataOut[0]}
{i:EP0RxFifo.u_fifo.dataOut[1]}
{i:EP0RxFifo.u_fifo.dataOut[2]}
{i:EP0RxFifo.u_fifo.dataOut[3]}
{i:EP0RxFifo.u_fifo.dataOut[4]}
{i:EP0RxFifo.u_fifo.dataOut[5]}
```

5. To print a list of the collection objects and their properties in column format, use the `c_print` command as follows:

- In the Tcl shell, type the `c_print` command with the `-prop` option. For example, typing `c_print -prop slack -prop view -prop clock $result` lists the objects in the `$result` collection, and their `slack`, `view` and `clock` properties.

| Object Name                     | slack  | view  | clock |
|---------------------------------|--------|-------|-------|
| {i:EP0RxFifo.u_fifo.dataOut[0]} | 0.3223 | "FDE" | clk   |
| {i:EP0RxFifo.u_fifo.dataOut[1]} | 0.3223 | "FDE" | clk   |
| {i:EP0RxFifo.u_fifo.dataOut[2]} | 0.3223 | "FDE" | clk   |
| {i:EP0RxFifo.u_fifo.dataOut[3]} | 0.3223 | "FDE" | clk   |
| {i:EP0RxFifo.u_fifo.dataOut[4]} | 0.3223 | "FDE" | clk   |
| {i:EP0RxFifo.u_fifo.dataOut[5]} | 0.3223 | "FDE" | clk   |
| {i:EP0RxFifo.u_fifo.dataOut[6]} | 0.3223 | "FDE" | clk   |
| {i:EP0RxFifo.u_fifo.dataOut[7]} | 0.3223 | "FDE" | clk   |
| {i:EP0TxFifo.u_fifo.dataOut[0]} | 0.1114 | "FDE" | clk   |
| {i:EP0TxFifo.u_fifo.dataOut[1]} | 0.1114 | "FDE" | clk   |

- To print out the results to a file, use the `c_print` command with the `-file` option. For example, `c_print -prop slack -prop view -prop clock $result -file results.txt` writes out the objects and properties listed above to a file called `results.txt`. When you open this file, you see the information in a spreadsheet format.
6. You can do a number of operations on a collection, as listed in the following table.

| To ...                                             | Do this ...                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Copy a collection                                  | <p>Create a new variable for the copy and copy the original collection to it with the <code>set</code> command. When you make changes to the original, it does not affect the copy, and vice versa.</p> <p><b><code>set my_collection_copy \$my_collection</code></b></p>                                                                                                                                                                                                      |
| List the objects in a collection                   | <p>Use the <code>c_print</code> command to view the objects in a collection, and optionally their properties, in column format:</p> <p style="padding-left: 40px;">"v:top"<br/>"v:block_a"<br/>"v:block_b"</p> <p>Alternatively, you can use the <code>-print</code> option to an operation command to list the objects.</p>                                                                                                                                                   |
| Generate a Tcl list of the objects in a collection | <p>Use the <code>c_list</code> command to view a collection or to convert a collection into a Tcl list. You can manipulate a Tcl list with standard Tcl commands. In addition, the Tcl collection commands work on Tcl lists.</p> <p>This is an example of <code>c_list</code> results:</p> <p style="padding-left: 40px;">{"v:top" "v:block_a" "v:block_b"}</p> <p>Alternatively, you can use the <code>-print</code> option to an operation command to list the objects.</p> |

## Example: c\_list Command

The following provides a practical example of how to use the `c_list` command. This example first finds all the CE pins with a negative slack that is less than 0.5 ns and groups them in a collection:

```
set get_components_list [c_list [find -hier -pin {*.CE} -filter  
@slack < {0.5}]]
```

The `c_list` command returns a list:

```
{t:EP0RxFifo.u_fifo.dataOut[0].CE}  
{t:EP0RxFifo.u_fifo.dataOut[1].CE}  
{t:EP0RxFifo.u_fifo.dataOut[2].CE} ...
```

You can use the list to find the terminal (pin) owner:

```
proc terminal_to_owner_instance {terminal_name terminal_type} {  
    regsub -all $terminal_type$ $terminal_name {} suffix  
    regsub -all {^t:} $suffix {i:} prefix  
    return $prefix  
}  
  
foreach get_component $get_components_list {  
    append owner [terminal_to_owner_instance $get_component {.CE}]  
"  
"  
}  
  
puts "terminal owner is $owner"
```

This returns the following, which shows that the terminal (pin) has been converted to the owning instance:

```
terminal owner is i:EP0RxFifo.u_fifo.dataOut[0]  
i:EP0RxFifo.u_fifo.dataOut[1] i:EP0RxFifo.u_fifo.dataOut[2]
```



## CHAPTER 2

# Analyzing with HDL Analyst

---

This chapter describes how to analyze logic in the HDL Analyst. See the following for detailed procedures:

- [Working in the Schematic](#), on page 86
- [Exploring Design Hierarchy](#), on page 106
- [Finding Objects](#), on page 113
- [Crossprobing](#), on page 124
- [Analyzing With the HDL Analyst Tool](#), on page 132
- [Working in the Standard HDL Analyst Schematic](#), on page 157
- [Exploring Design Hierarchy \(Standard\)](#), on page 166
- [Finding Objects \(Standard\)](#), on page 172
- [Crossprobing \(Standard\)](#), on page 183
- [Analyzing With the Standard HDL Analyst Tool](#), on page 190

## Working in the Schematic

The HDL Analyst tool includes single-page schematics, which can help you graphically analyze and navigate your entire design easier. This table shows where to find information for performing common tasks in the schematics:

|                             |                                                                                   |
|-----------------------------|-----------------------------------------------------------------------------------|
| Cloning schematics          | <a href="#">Cloning Schematics</a> , on page 89                                   |
| Instance groups             | <a href="#">Grouping Objects in Schematic</a> , on page 98                        |
| Partial dissolve            | <a href="#">Grouping Objects in Schematic</a> , on page 98                        |
| Net-based filtering         | <a href="#">Filtering Schematics</a> , on page 138                                |
| Unfiltering                 | <a href="#">Filtering Schematics</a> , on page 138                                |
| Multi-threaded Find command | <a href="#">Browsing to Find Objects in HDL Analyst Views</a> , on page 113       |
| Mouse strokes               | <a href="#">Mouse Stroke Conventions</a> , on page 88                             |
| Push View Tab               | <a href="#">Cloning Schematics</a> , on page 89                                   |
| Peek                        | <a href="#">Viewing Design Hierarchy and Context</a> , on page 132                |
| Displaying contents         | <a href="#">Viewing Design Hierarchy and Context</a> , on page 132                |
| Bus display                 | <a href="#">Dissolving and Partial Dissolving of Buses and Pins</a> , on page 146 |

For information on specific tasks like analyzing critical paths, see the following sections:

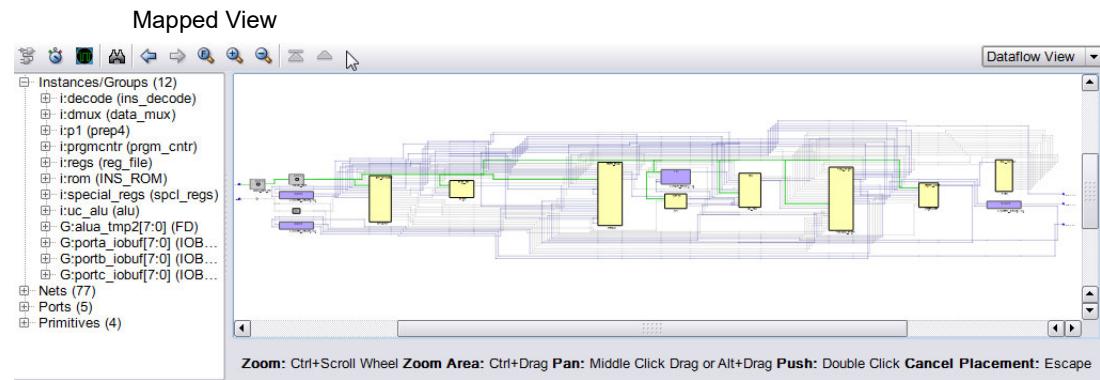
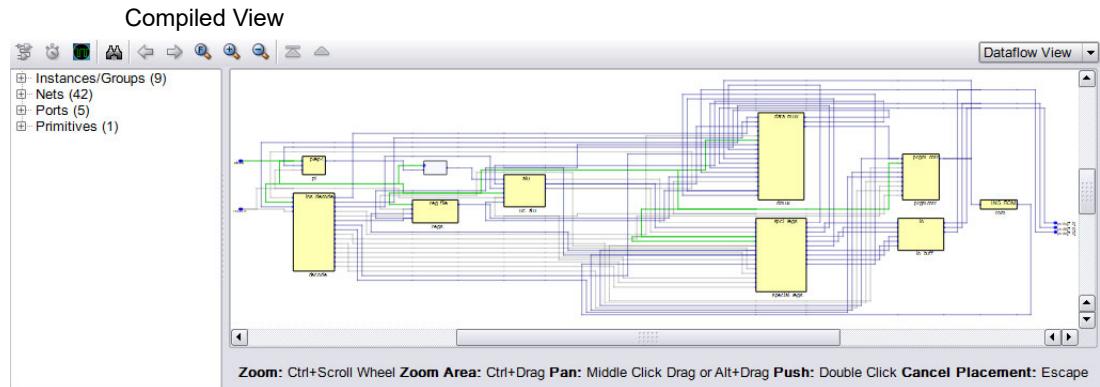
- [Traversing Design Hierarchy with the Hierarchy Browser](#), on page 106
- [Exploring Object Hierarchy with Push/Pop Commands](#), on page 108
- [Browsing to Find Objects in HDL Analyst Views](#), on page 113
- [Crossprobing](#), on page 124
- [Analyzing With the HDL Analyst Tool](#), on page 132

## Opening Different Views

The procedure for opening a view is the same at different design stages; the main difference is the content that is available at the different design database states.

1. Start at the database state you want.
2. To enable the new HDL Analyst tool from the UI, select Options->Use New HDL Analyst option. By default, this option is enabled. Open the schematic using one of the following commands:
  - At the Tcl command line, type view schematic.
  - Click the View the schematic icon ()

All schematics have the schematic on the right and a pane on the left that contains a hierarchical list of the objects in the design. This pane is called the Hierarchy Browser.

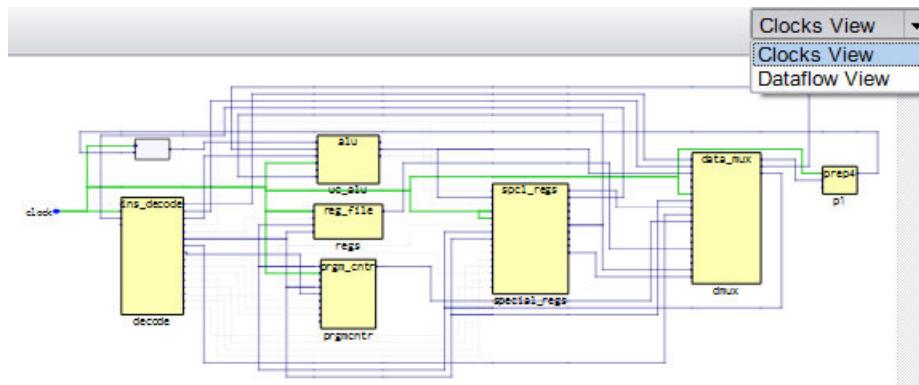


## Dataflow View

Both the compiled and mapped views have a Dataflow View. Use this view to display objects from a left to right datapath flow as shown above. You can display a Clock View and Compact View as well.

## Clock View

To display all sequential elements connected to clock nets and debug the clocks in the design use the view selector. Select Clocks View from the drop-down menu in the upper right corner of the schematic view. Clock nets are displayed with the color green.



## Mouse Stroke Conventions

Use the mouse strokes to control navigation and the display, which are listed at the bottom of the schematic window. They include the following:

Zoom            Ctrl-scroll wheel

Zoom area     Ctrl-drag

Pan            Middle-click and drag or Alt-drag

Push            Double-click

Pop            Double-click on an empty space

Cancel display    Esc  
For large designs. Cancels netlist display, but netlist is still accessible from the hierarchy browser, Tcl window, and Find dialog box.

## Cloning Schematics

Most operations performed in any of the HDL Analyst views (Clock View or Dataflow View) are displayed in the current view. To create a new view of the netlist, use the clone commands.

1. To clone the current view displayed, right-click and select Clone Schematic from the drop-down menu. This view opens in a new window. You can open multiple clone views.

Tcl equivalent: `analyst clone_view`

To close a clone view: `analyst close_design designID`.

For example: `analyst close_design d:3`

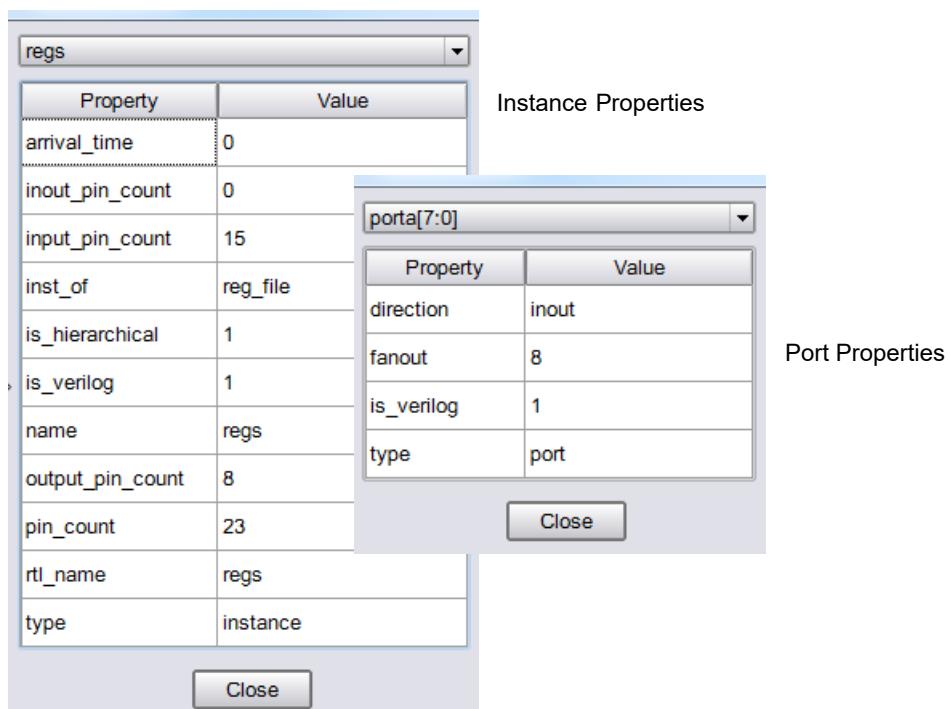
2. To push into an object and create a new view, select an object then right-click and Push in New Tab from the drop-down menu. For more information, see [Exploring Object Hierarchy with Push/Pop Commands, on page 108](#).
3. To filter objects and create a new view, select the objects then right-click and select Filter in a New Tab from the drop-down menu. For more information, see [Filtering Schematics, on page 138](#).

## Viewing Object Properties

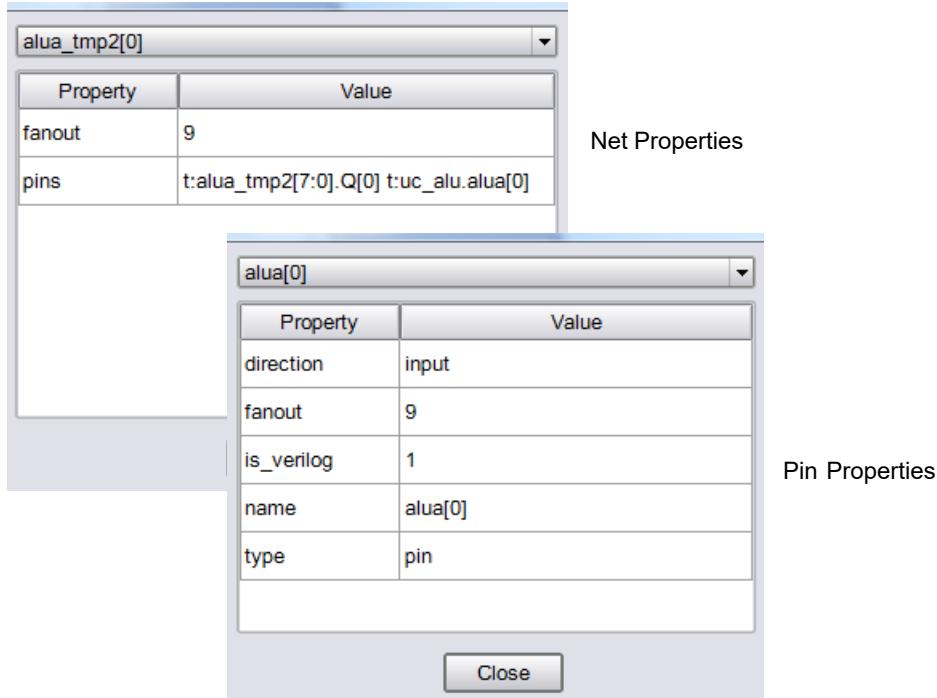
There are a few ways in which you can view the properties of objects.

1. To temporarily display the properties of a particular object, hold the cursor over the object.
2. Select the object, right-click, and select Properties. The properties and their values are displayed in a table.

For example, you can view the properties for instances and ports as shown below.



Similarly, you can view the properties for pins and nets.



3. You can copy any number of fields from the Properties dialog box and paste the properties to the Tcl window or a text file from within the tool.

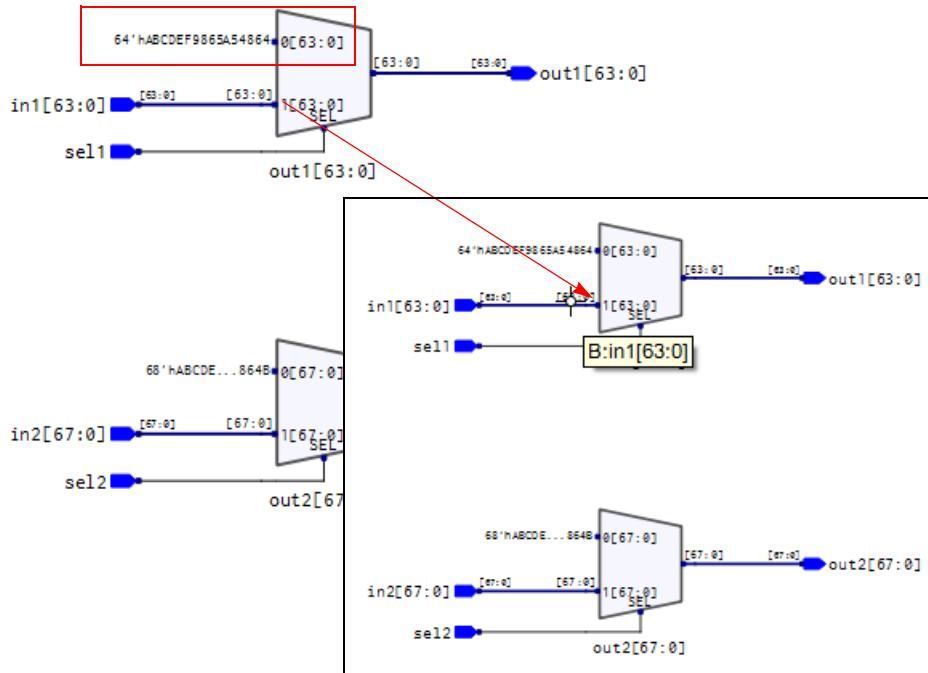
For example, use this field with the collection commands to identify groups of objects in the schematic.

| Property        | Value                   |
|-----------------|-------------------------|
| arrival_time    | 0                       |
| inout_pin_count | 5                       |
| input_pin_count | 15                      |
| inst_of         | ins_decode              |
| is_hierarchical | 1                       |
| is_verilog      | 1                       |
| lib.cell.view   | work.ins_decode.verilog |

**Close**

## Viewing Objects with Constant Values

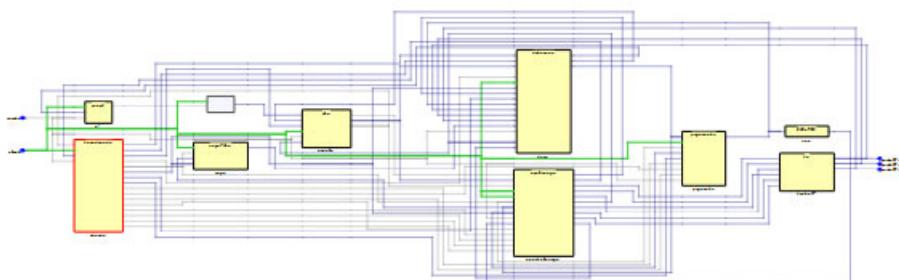
You can display constants and view their constant values. Individual functions of the constant are displayed. Use tool tips to see the full value for the constant.



## Viewing Objects in a Source File

The HDL Analyst view provides various ways to view objects in a source file. For example:

1. Select an object in the schematic view.



2. Then, right-click and select one of the following:

- View Instance in Source – Opens the RTL source file and finds the instantiated instance selected.

```

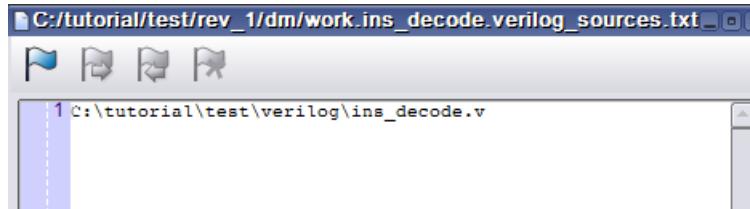
4 input clock, resetn;
5
6 inout [7:0] porta, portb, portc;
7
8 reg [7:0] alua_tmp2;
9 wire resetn, aluz;
10 wire [11:0] inst;
11 wire f_we, w_we, status_z_we, status_c_we, tris_we, skip;
12 wire [7:0] k;
13 wire [4:0] fsel;
14 wire [8:0] longk;
15 wire [3:0] aluop;
16 wire [1:0] alua_sel, alub_sel;
17 wire bdpol;
18 wire opcode_call, opcode_goto, opcode_retlw;
19 wire [2:0] b_mux;
20 wire [17:0] fin;
21 wire [10:0] pc;
22 wire [7:0] regfile_out;
23 wire [7:0] far, rtcc;
24 wire [7:0] status, porta, portb, portc, w;
25 wire [7:0] alua, alub;
26 wire alu_out;
27 wire [7:0] aluout;
28 wire [11:0] romdata;
29 wire [7:0] port_int_a;
30 wire [7:0] port_int_b;
31 wire [7:0] port_int_c;
32 wire [7:0] trisa;
33 wire [7:0] trisb;
34 wire [7:0] trisc;
35 wire clk1, clk2, clk3, clk4;
36 wire [7:0] alua_tmp;
37 assign clk1 = clock;
38 assign clk2 = clock;
39 assign clk3 = clock;
40 assign clk4 = clock;
41 assign rtcc = 0;
42 assign b_mux = 0;
43 // instantiating prep4 block
44 prep4 p1 (alu_tmp, alua, clk3, resetn);
45
46 // instantiating decode block
47 ins_decode decode (clk2, resetn, aluz, inst, f_we, w_we,
48                      status_z_we, status_c_we, tris_we,
49                      skip, k, fsel, longk, aluop,
50                      alua_sel, alub_sel, bdpol, opcode_goto, opcode_call, opcode_retlw );
51

```

- View Module in Source – Opens the RTL source file and finds the instantiated module selected.

```
1 module ins_decode( clk2,resetn ,aluz, inst, f_we, w_we,
2                      status_z_we, status_c_we, tris_we,
3                      skip, k, fsel, longk, aluop,
4                      alua_sel, alub_sel, bdpol, opcode_goto,
5                      opcode_call, opcode_retlw);
6 input clk2, resetn;
7 input aluz;
8 input [11:0] inst;
9
10 inout f_we;
11
12
13 output w_we, status_z_we, status_c_we, tris_we, skip;
14 output [7:0] k;
15
16 inout [4:0] fsel;
17
18 output [8:0] longk;
19
20 output [3:0] aluop;
21
22 output [1:0] alua_sel, alub_sel;
23
24 output bdpol;
25
26 output opcode_call, opcode_retlw, opcode_goto;
27
28 reg [13:0] decodes, decodes_in;
29
30 wire reset = ~resetn;
31 // this is never used either take it out
32 // or hook it up if necessary
33 //reg [7:0] bit_decoder;
```

- View dependent source file list – A source file (*moduleName\_source.txt*) containing the specified module is created in the dm directory of the Implementation Results directory.



- View source file list – A source file (*designName\_source.txt*) containing the specified modules for the design is created in the dm directory of the Implementation Results directory.

```

1 C:\tutorial\test\verilog\mult.v
2 C:\tutorial\test\verilog\alu.v
3 C:\tutorial\test\verilog\spcl_regs.v
4 C:\tutorial\test\vhdl\ins_rom.vhd
5 C:\tutorial\test\verilog\reg_file.v
6 C:\tutorial\test\verilog\pc.v
7 C:\tutorial\test\verilog\state_mc.v
8 C:\tutorial\test\verilog\io.v
9 C:\tutorial\test\verilog\data_mux.v
10 C:\tutorial\test\verilog\ins_decode.v
11 C:\tutorial\test\verilog\eight_bit_uc.v

```

## Selecting Objects in Schematics

For mouse selection, standard object selection rules apply:

| To select ...                               | Do this ...                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Single objects                              | <p>Click the object in the schematic, or click the object name in the Hierarchy Browser.</p> <p>Tcl equivalent: <b>select {i:instanceName}</b></p> <p>For a net: <b>select {n:netName}</b></p> <p>For a pin: <b>select {t:pinName}</b></p> <p>For a port: <b>select {p:portName}</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Multiple objects                            | <p>Use one of these methods:</p> <ul style="list-style-type: none"> <li>Draw a rectangle around the objects.</li> <li>Select an object, press Ctrl, and click other objects you want to select.</li> <li>Select multiple objects in the Hierarchy Browser. See <a href="#">Browsing With the Hierarchy Browser , on page 113</a>.</li> <li>Use Find to select the objects you want. See <a href="#">Finding Objects , on page 113</a>.</li> </ul> <p>Tcl equivalent: <b>select {{i:instance1} {i:instance2}}</b></p> <ul style="list-style-type: none"> <li>Select all instances in the current view or press Ctrl+A:</li> </ul> <p>Tcl equivalent: <b>select -instances</b></p> <ul style="list-style-type: none"> <li>Select all primitives in the current view or press Ctrl+Alt+A:</li> </ul> <p>Tcl equivalent: <b>select -primitives</b></p> |
| Objects by type<br>(instances, ports, nets) | <p>Use Find to select the objects (see <a href="#">Browsing With the Find Command , on page 118</a>), or use the Hierarchy Browser, which lists objects by type.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

| To select ...                                                 | Do this ...                                                                                                                                            |
|---------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| No objects<br>(deselect all<br>currently<br>selected objects) | Click the left mouse button in a blank area of the schematic.<br>Deselected objects are no longer highlighted.<br>Tcl equivalent: <b>select -clear</b> |

The HDL Analyst view highlights selected objects in red. If you have other windows that are cloned, the selected object is highlighted in the other windows as well (crossprobing).

## Selecting a Sequence of Objects in the Schematic

You can select a series of objects in the schematic, then traverse back and forth through these selections in the order they were chosen. Use the backwards and forwards icons (⬅ ➡) to move between each selection. This is handy to help you undo or redo any changes with your selections. Once you select an operation, the selection becomes unavailable unless the view does not change.

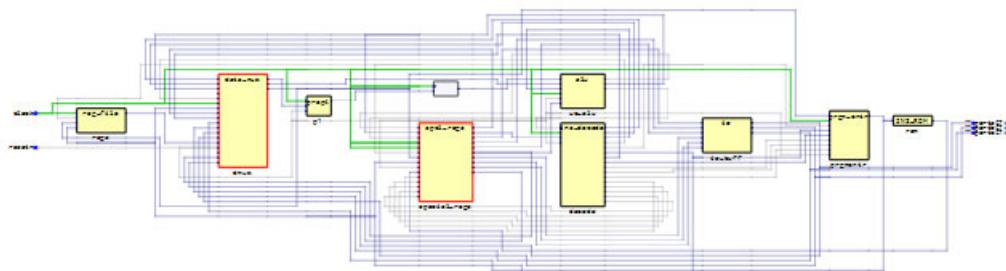
When an object is selected in the schematic, you can use the following command to print the name of the object (instance, port, pin, or net):

```
analyst get_selected [-inst] [-net] [-port] [-pin]
```

In the following example, two instances are selected. Specify the following command to display their names in the Tcl window:

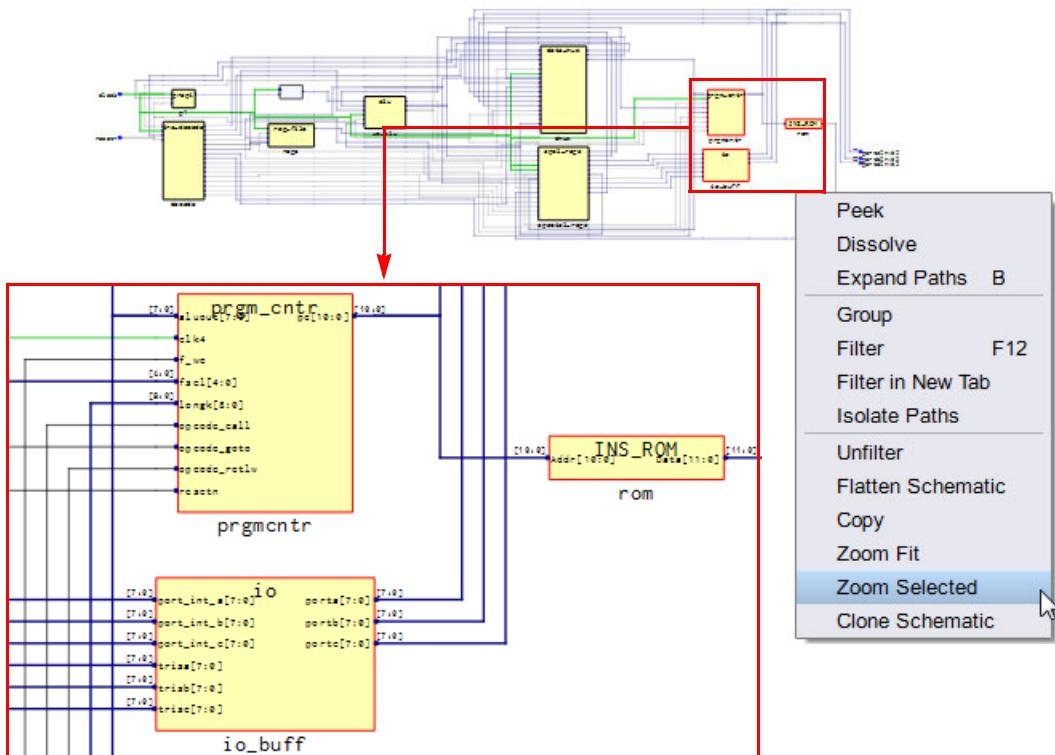
```
% analyst get_selected -inst
{i:dmux} {i:special_regs}
```

This command returns a Tcl list of selected objects in the current view.



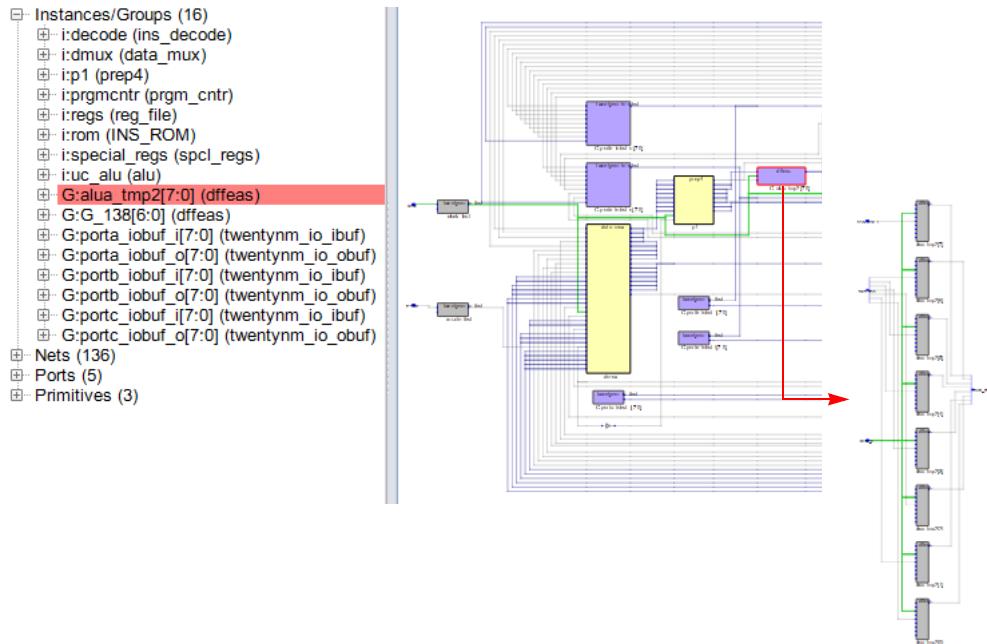
## Zooming in on Selected Objects

Once you have selected objects in the schematic view, you can automatically zoom in on these objects. To do this, highlight the required objects, then right-click and select Zoom Selected from the drop-down menu.



## Grouping Objects in Schematic

You can group objects in the schematic. Sometimes the HDL Analyst tool automatically determines groups shown with the color purple below. When you push into the group block, the content of its objects is displayed.

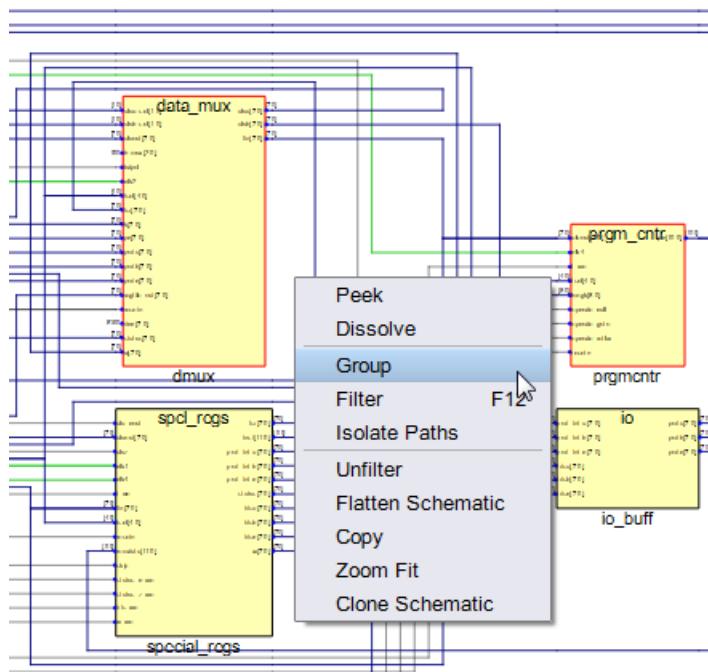


The HDL Analyst tool automatically groups instances with similar names at all levels of hierarchy, when you enable the Allow Automatic Grouping option on the HDL Analyst Options dialog box. For example, suppose there are three registers with the names `out_reg[1]`, `out_reg[2]`, and `out_reg[3]`. A group will be created with the registers having the name `out_reg[3:1]`.

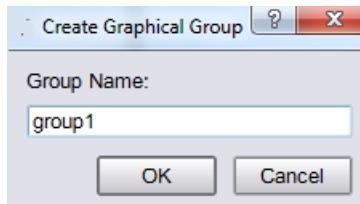
To create your own groups:

1. Select the specified objects in the view.
2. Right-click and select the Group option from the pop-up menu.

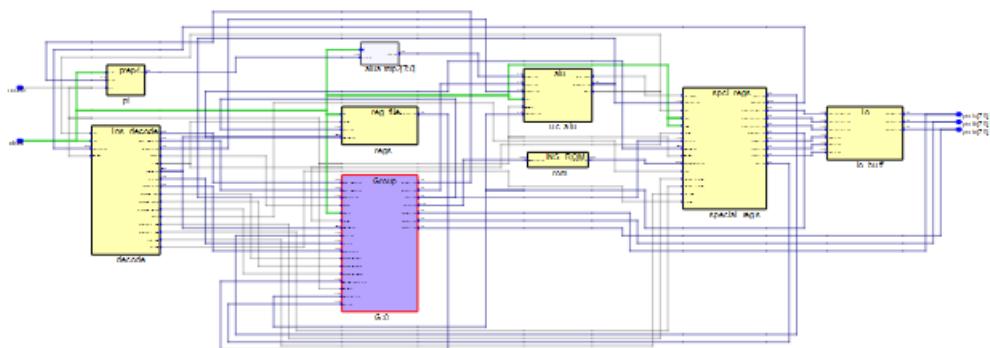
This creates a dummy hierarchy that groups the objects together, which is only displayed in the HDL Analyst GUI. It does not generate any hierarchical changes to the design.



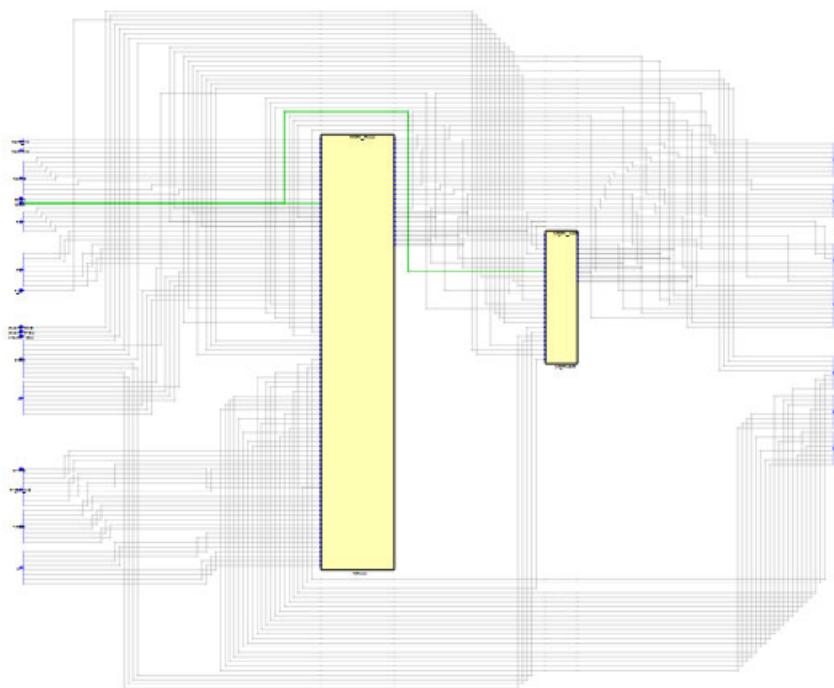
3. Specify a group name in the dialog box that pops up.



This forms a group of objects in group1 that is displayed with the purple block.

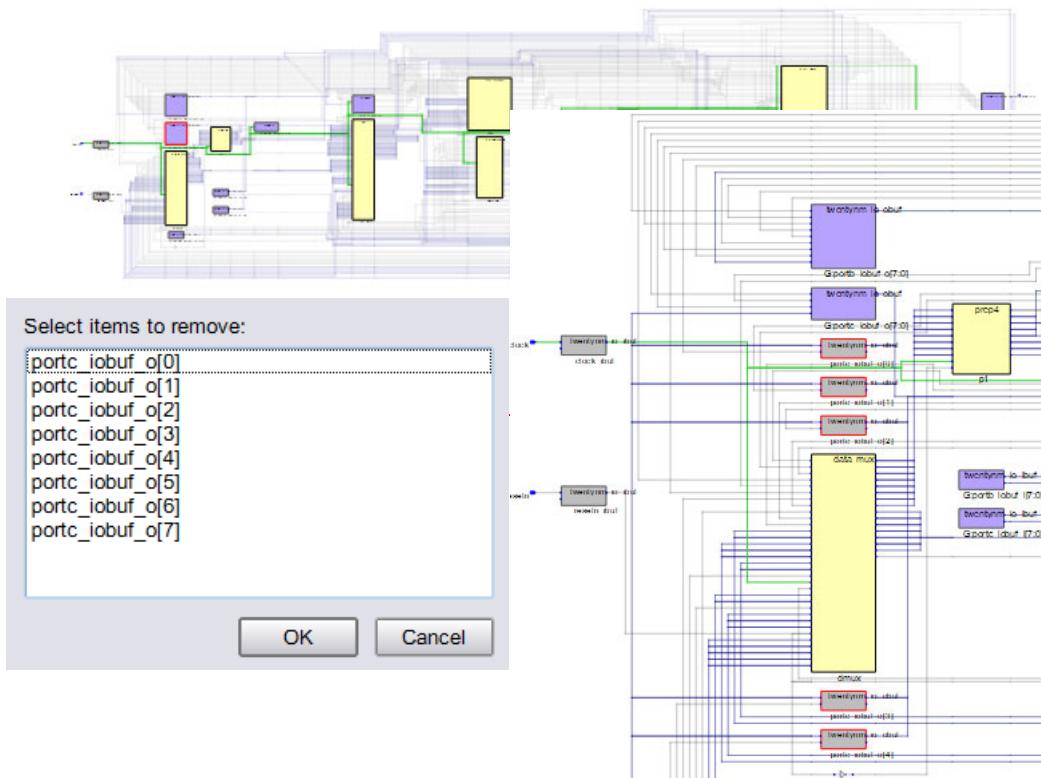


4. Push into the purple block to see the grouped objects.



5. To ungroup selections, right-click and select Dissolve from the drop-down menu.

- To remove individual instances of a group, right-click and select Partial Dissolve from drop-down menu. A dialog box is displayed where you can select the items to remove.



- User-created groups are not saved when you close and re-open the same netlist.

## Moving Between Views in a Schematic Window

When you filter or expand your design, you move through a number of different design views in the same schematic window. For example, you might start with a view of the entire design, then filter an object and finally expand a connection in the filtered view, for a total of three views. You can also move back after flattening a view.

- To move back to the previous view, click the Back icon ().

The software displays the last view.

2. To move forward again, click the Forward icon ( ➤ ).

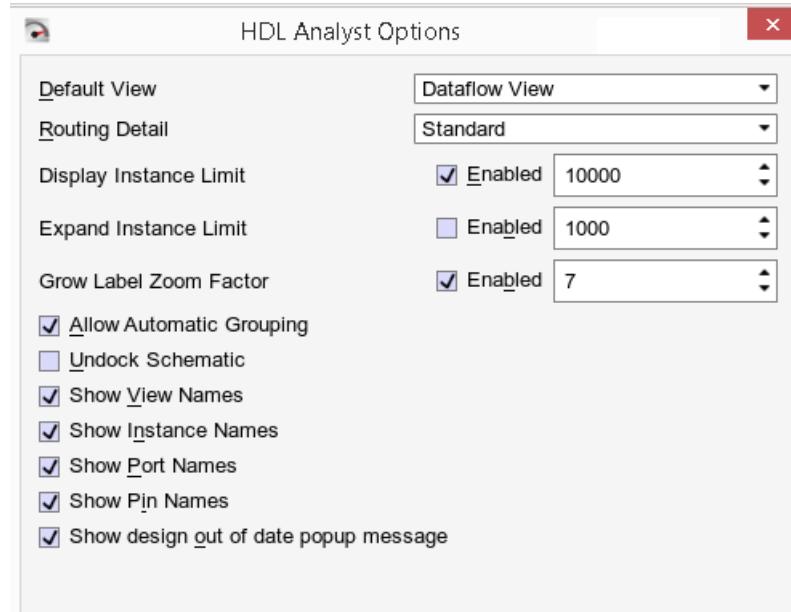
The software displays the next view in the display history.

## Setting the Schematic Preferences

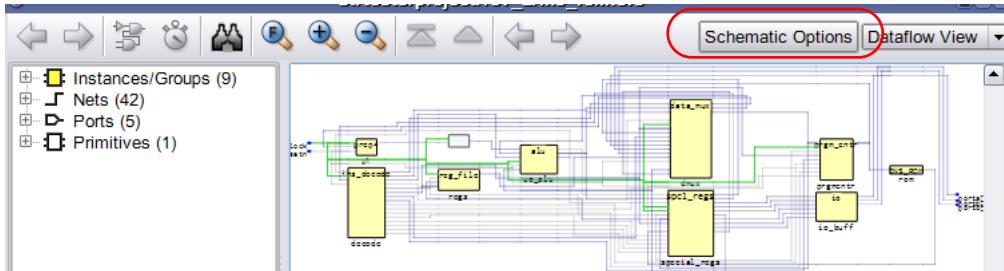
You can set various preferences for the schematic from the user interface.

1. Select Options->Schematic Options.

For a description of all the options on this form, see [HDL Analyst Options Dialog Box, on page 72](#) in the *Reference Manual*.



- Also, for your convenience you can simply select the Schematic Options button from the top of the HDL Analyst view.



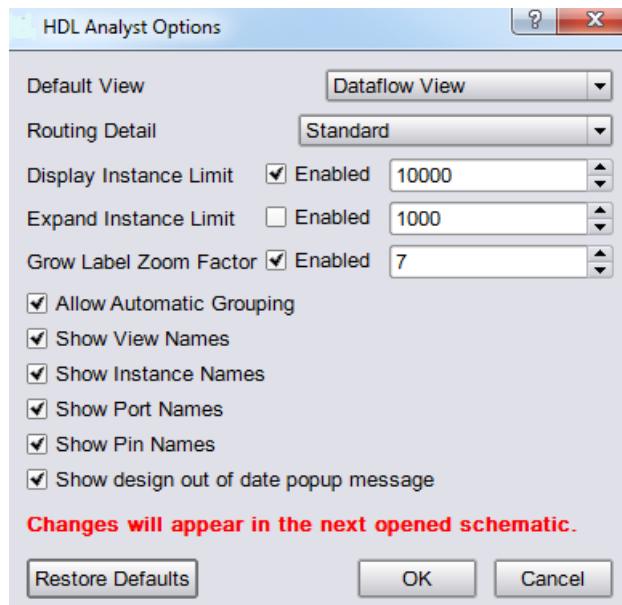
2. The table details the following operations:

| To ...                                                               | Do this ...                                                                                                                                                                     |
|----------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Specify how you want the schematic to display.                       | Select Clock View or Dataflow View (default).                                                                                                                                   |
| Specify how the tool determines the detailed routing for the design. | Select Standard (default) or Quick (direct connection).                                                                                                                         |
| Show names for the view, instances, ports, and pins.                 | Enable any of the following: <ul style="list-style-type: none"><li>• Show View Names</li><li>• Show Instance Names</li><li>• Show Port Names</li><li>• Show Pin Names</li></ul> |
| Show the out of date popup message for the design.                   | When enabled, shows the design out of date popup message if the design file has changed while the HDL Analyst view was opened.                                                  |
| Specify limits when displaying or expanding instances.               | Set the limit and select Enabled.                                                                                                                                               |
| Specify a zoom factor for labels displayed in the schematic view.    | Select a value between 1 and 10, where labels are shown increasing in size respectively. Changes will appear in the next opened schematic view. The default is 2.               |
| Determine if you want automatic grouping for the design.             | When Allow Automatic Grouping is enabled, the tool automatically groups instances with similar names at every level in the design.                                              |

3. Enable the Show design out of date popup message option to ensure that the correct version of the HDL Analyst view is being displayed. You might be

looking at inconsistent results, if the design netlist file (srs) has changed. You can choose to close the current HDL Analyst view and reload the updated version.

When you select this option, the following warning message is displayed at the bottom of this dialog box.



# Exploring Design Hierarchy

Schematics generally have a certain amount of design hierarchy. You can move between hierarchical levels using the Hierarchy Browser mode. For additional information, see *Analyzing With the HDL Analyst Tool*, on page 132. See *Traversing Design Hierarchy with the Hierarchy Browser*, on page 106.

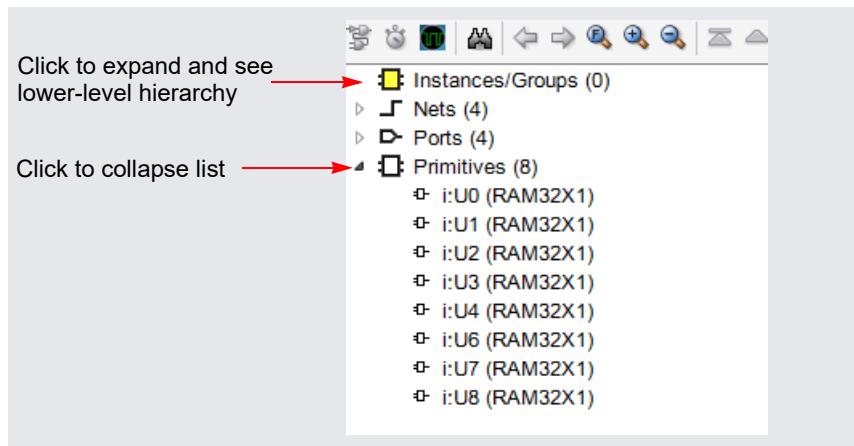
## Traversing Design Hierarchy with the Hierarchy Browser

The Hierarchy Browser is the list of objects on the left side of the schematic. It is best used to get an overview, or when you need to browse and find an object. If you want to move between design levels of a particular object, using the Push command is more direct. Refer to *Exploring Object Hierarchy with Push/Pop Commands*, on page 108 for details.

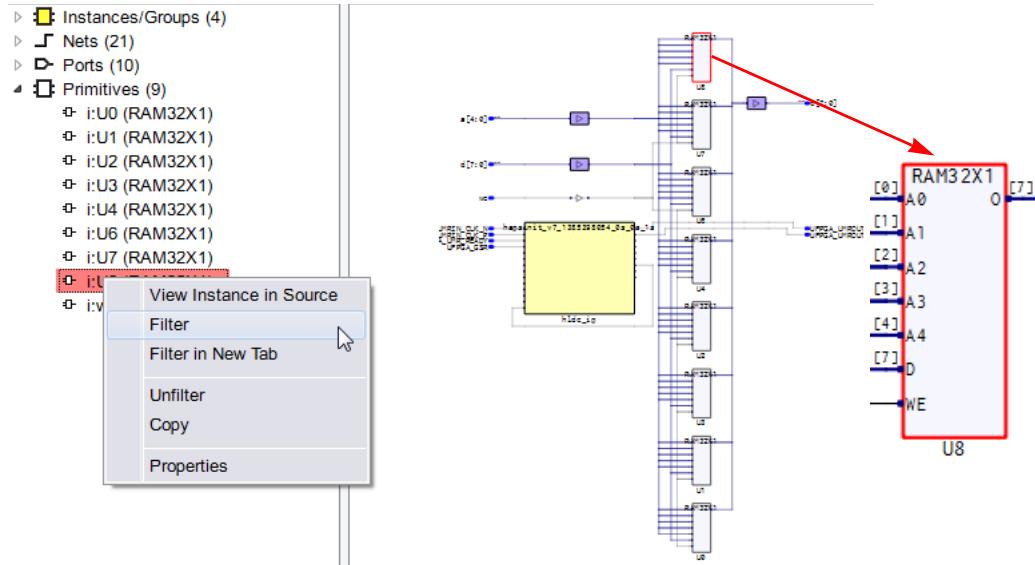
The hierarchy browser allows you to traverse and select the following:

- Instances or Groups
- Ports
- Internal nets

The browser lists the objects by type. Use the expand ( ▲ ) and collapse ( ▾ ) signs to ascend or descend the hierarchy.



1. You can perform some similar operations as done in the schematic, such as filtering an object from the Hierarchy Browser. For example, highlight an instance, then right-click and select Filter as shown below.



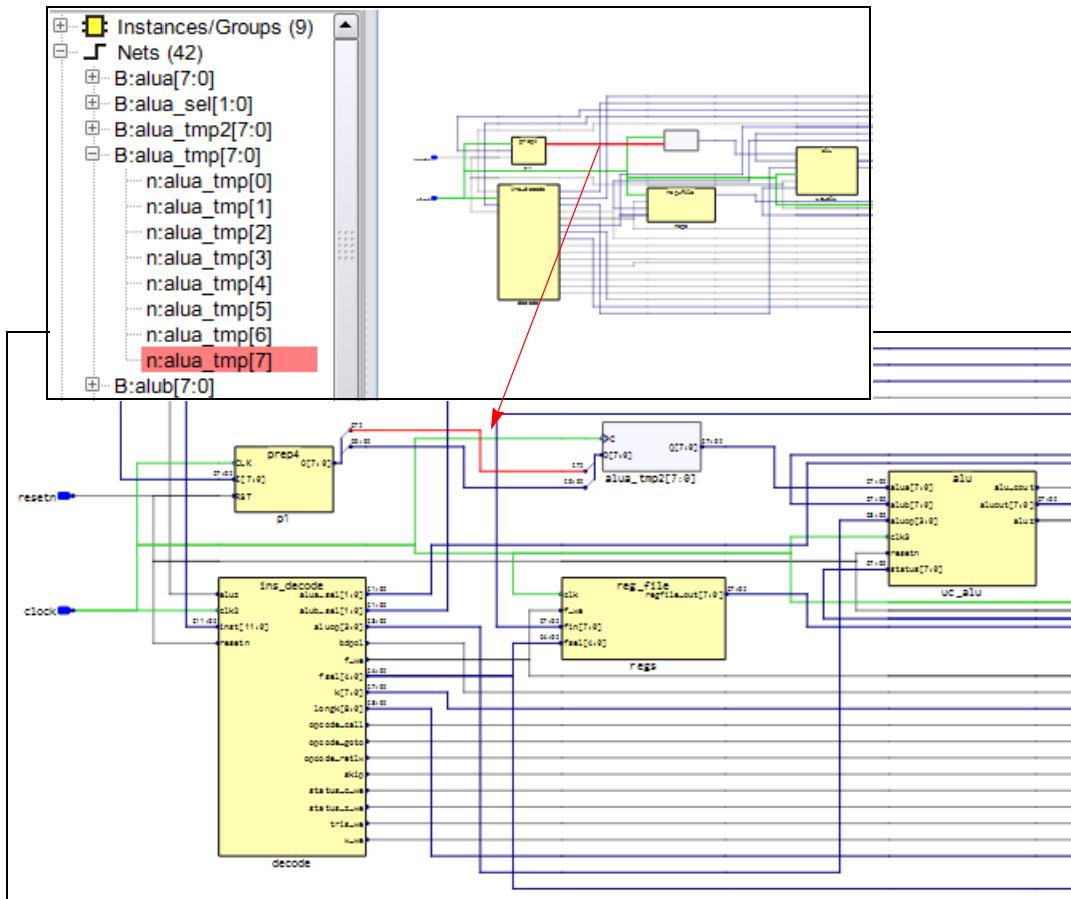
2. You can also crossprobe to instances and modules in the source file from the Hierarchy Browser. Right-click and select either:

- View Instance in Source
- View Module in Source

For details, see [Crossprobing, on page 124](#).

3. To extract logic for a partially dissolved net:

- Select a partially selected net from the hierarchy browser.
- Right-click and select Extract Net from the drop-down menu in the HDL Analyst view.



## Exploring Object Hierarchy with Push/Pop Commands

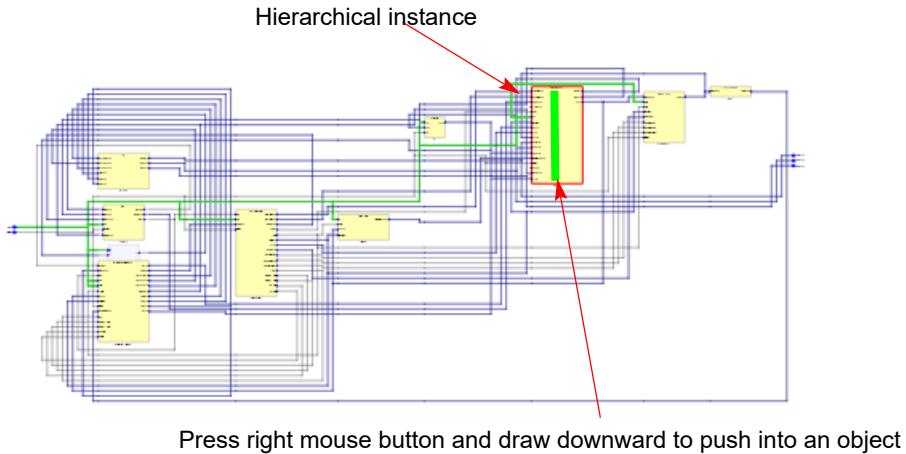
To view the internal hierarchy of a specific instance, use the Push/Pop commands from the drop-down menu or mouse strokes. When combined with other commands like filtering and expansion commands, Push/Pop can be a very powerful tool for isolating and analyzing logic. See [Filtering Schematics, on page 138](#) and [Expanding Pin and Net Logic, on page 140](#) for details about filtering and expansion. See the following sections for information about pushing down and popping up in hierarchical design objects:

- [Pushing into Objects, on page 109](#)
- [Popping up a Hierarchical Level, on page 111](#)

## Pushing into Objects

In the schematic, you can push into instances and view the lower-level hierarchy. You can use a mouse stroke or the command to push into objects:

1. To move down a level (push into an object) with a mouse stroke, put your cursor near the top of the object, hold down the right mouse button, and draw a vertical stroke from top to bottom. You can push into the following objects; see step 3 for examples of pushing into different types of objects.
  - Hierarchical instances. They can be displayed as pale yellow boxes (opaque instances).



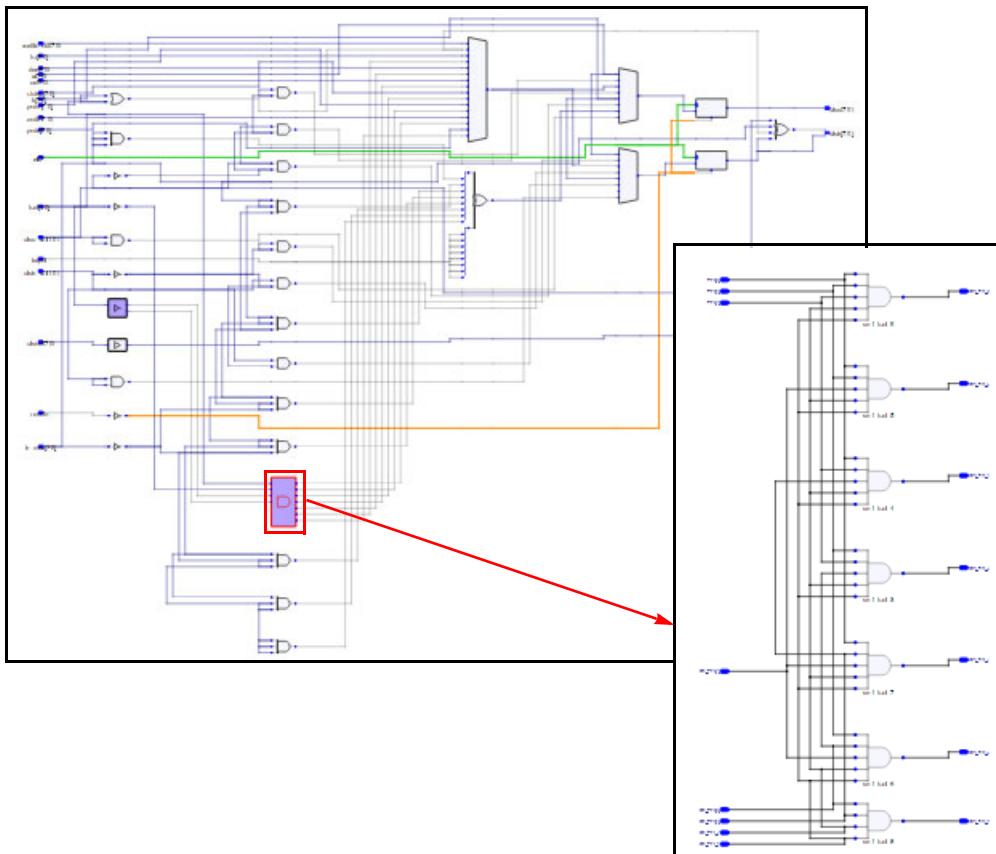
- Technology-specific primitives. The primitives are listed in the Hierarchy Browser in the schematic, under `i:instanceNames ->Primitives`.

Instances formed into a group. The remaining steps show you how to use the icon or command to push into an object.

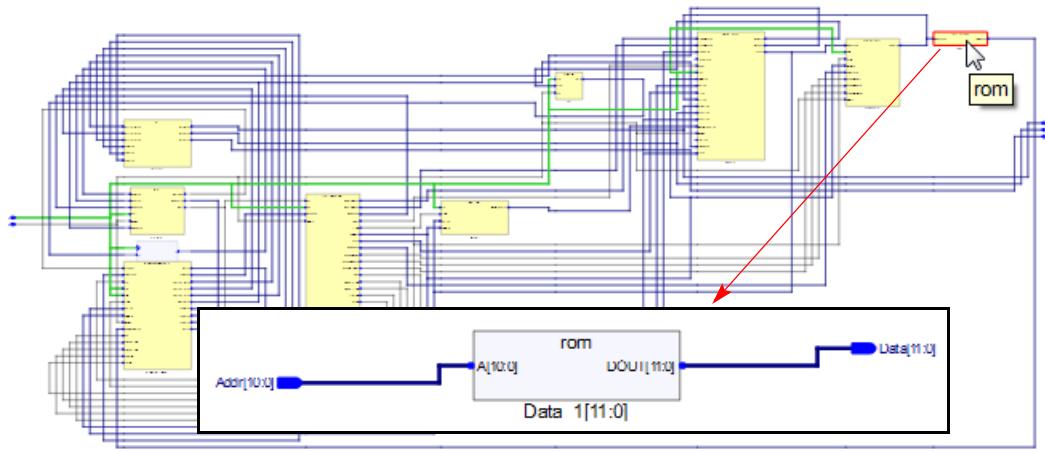
2. Enable the Push/Pop command by doing one of the following:

- Double-click on the object.
- Right-click in the view and select Push/Pop from the drop-down menu.
- Use the mouse strokes.

After pushing into an instance, the following schematic is displayed. Notice the purple block that groups gates with similar names together. You can push into this block as well.

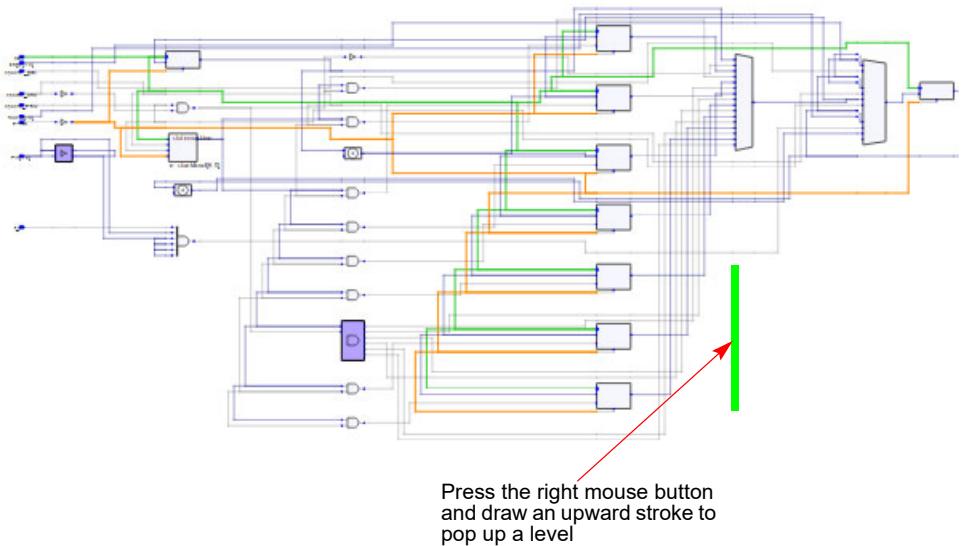


3. To push (descend) into an object, double-click on the hierarchical object. The following figure shows the result of pushing into a ROM.



## Popping up a Hierarchical Level

1. To move up a level (pop up a level), put your cursor anywhere in the design,
  - Use the Pop Hierarchy icon (  ).
  - Hold down the right mouse button, and draw a vertical mouse stroke, moving from the bottom upwards.



- The software moves up a level, and displays the next level of hierarchy.
2. Alternatively, you can double-click on any whitespace in the view to pop up a level from where you pushed into the hierarchy.

# Finding Objects

In the schematic, you can use the Hierarchy Browser or the Find command to find objects, as explained in these sections:

- [Browsing to Find Objects in HDL Analyst Views, on page 113](#)
- [Using Wildcards with the Find Command, on page 123](#)

## Browsing to Find Objects in HDL Analyst Views

You can always zoom in to find an object in the schematic. Use Zoom Fit to quickly fit all objects into the schematic. The following procedure shows you how to browse through design objects and find an object at any level of the design hierarchy. You can use the Hierarchy Browser or the Find command to do this. If you are familiar with the design hierarchy, the Hierarchy Browser can be the quickest method to locate an object. The Find command is best used to graphically browse and locate the object you want.

### Browsing With the Hierarchy Browser

1. In the Hierarchy Browser, click the name of the net, port, or instance you want to select.

The object is highlighted in the schematic.

2. To select a range of objects, you can press and hold the Shift key while clicking the selected objects in the range.

The software selects and highlights all the objects in the range.

3. If the object is on a lower hierarchical level, do either of the following:
  - Expand the appropriate higher-level object by clicking the collapsed symbol next to it, and then select the object you want.
  - Push down into the higher-level object, and then select the object from the Hierarchy Browser.

The selected object is highlighted in the schematic. However, you may have to filter the object to view it in the design hierarchy.

4. To select all objects of the same type, select them from the Hierarchy Browser. For example, you can find all the nets in your design.

## Browsing With the New Hierarchy Browser

*Synplify Pro, Synplify Premier  
Beta*

Using the HDL Analyst tool, you can display a hierarchy of design objects in the Hierarchy Browser. Typically, the time taken to display a design hierarchy depends on the size of the design.

To speed up this process, the new Hierarchy Browser traverses the entire (text-based) netlist to quickly extract hierarchical instance data. This helps to display the entire netlist hierarchy quickly and also facilitates the viewing of custom instances on demand, instead of traversing down the design hierarchy. This flow is advantageous in large designs, to display the hierarchy and view any instance, quickly.

To enable the new Hierarchy Browser, follow these steps:

1. To set preference in the schematic from the user interface, do the following:

Options->Schematics Options

The HDL Analyst Options dialog box is displayed.

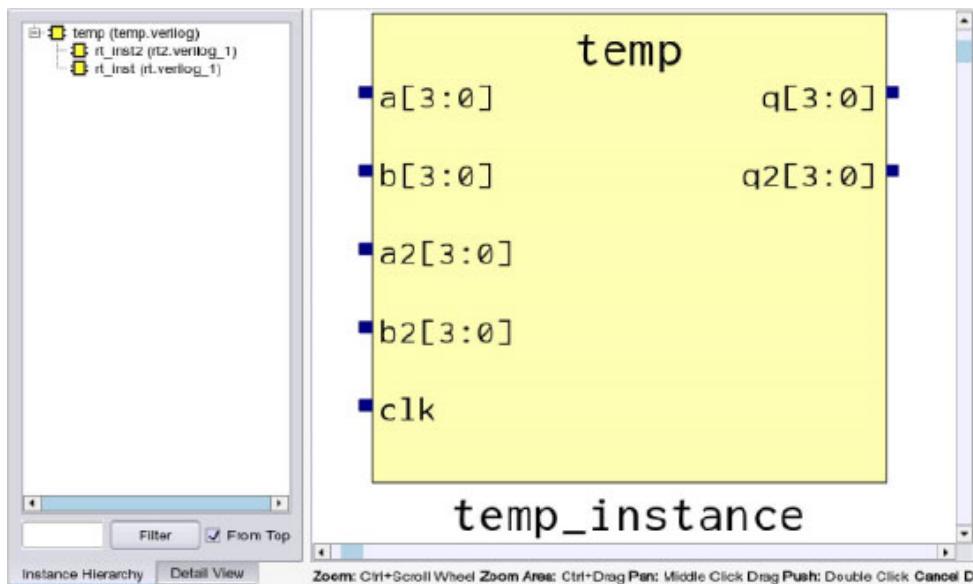
2. Enable the [BETA] Use new hierarchy browser option on the HDL Analyst Options dialog box. A warning is displayed, as shown below:



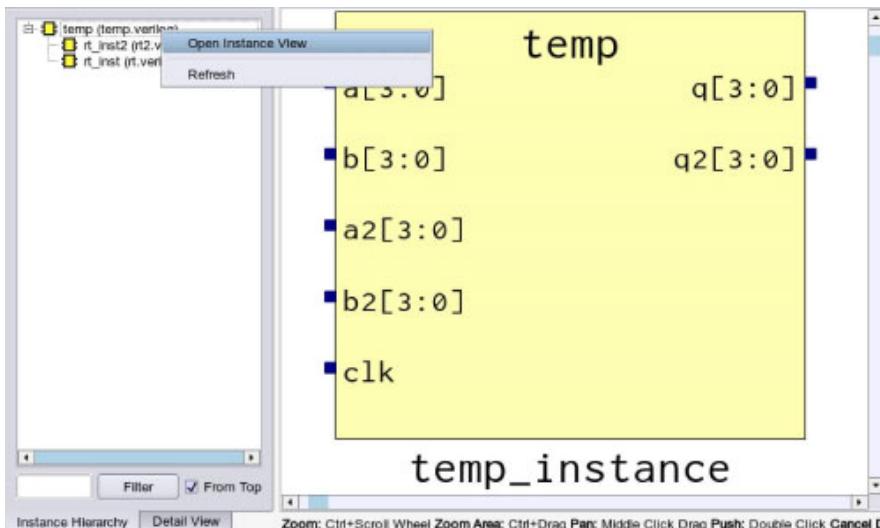
When you enable the Use new hierarchy browser option in the HDL Analyst Options dialog box, the HDL Analyst tool displays the RTL schematic at the instance level (Instance Hierarchy tab) and design level (Design View tab), when the next schematic displayed.

The Instance Hierarchy tab is visible only when the Use new hierarchy browser option is enabled in the Schematic Options (HDL Analyst Options) dialog box.

3. Click OK to close the dialog box. Now each design you view will follow the updated option settings.
4. Select the RTL View icon or select RTL->Hierarchical View from the HDL-Analyst menu, to view the RTL view of the design.

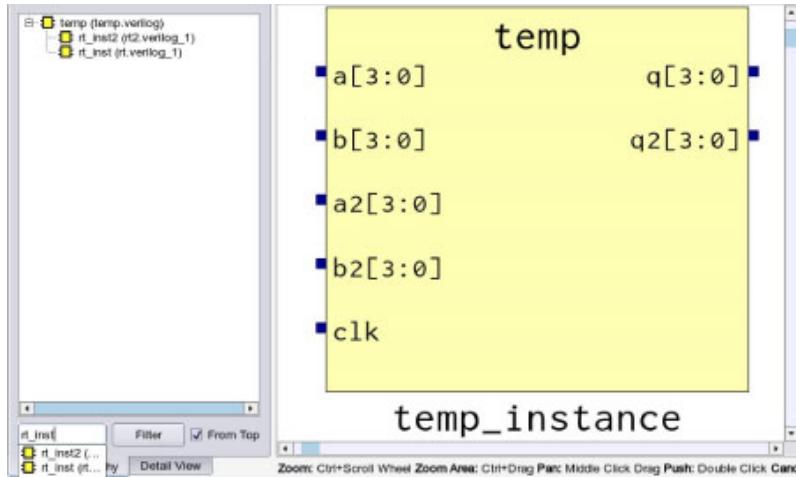


5. Open the instance by one of these methods:
  - Double-click on the instance.
  - Right-click and select Open Instance View.



## Browsing an Object by Filtering or Loading

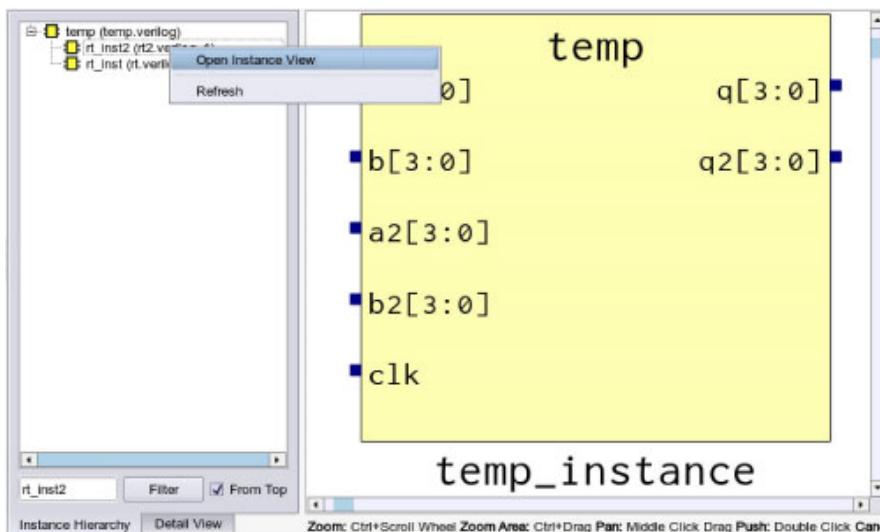
1. To filter a particular instance, type the instance name into the text box available below the Hierarchy Browser pane as shown below:



The instance names become visible in the drop-down as you continue to type.

2. Choose the instance and click Filter to display the instance in the RTL schematic view.
3. To search through a hierarchical path, select From Top. If this is not checked, the command searches the entire design.
4. To load an instance from the Hierarchy Browser, right-click on the desired instance and select the Open Instance option.

The instance is displayed in the RTL schematic view.



The Instance Hierarchy tab is visible only when the Use new hierarchy browser option is enabled in the Schematic Options (HDL Analyst Options) dialog box.

## New Hierarchy Browser Limitations:

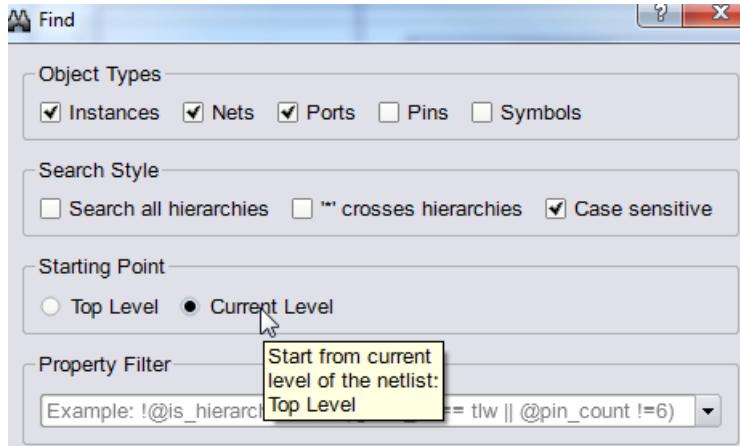
Consider these limitations before using the new Hierarchy Browser flow:

- Currently, usable only with compiled-stage/mapped-stage netlists.
- In map designs, primitives and black boxes appear in the hierarchy but not in the RTL view.
- Certain view options while filtering or loading an instance from the new Hierarchy Browser do not work correctly.

## Browsing With the Find Command

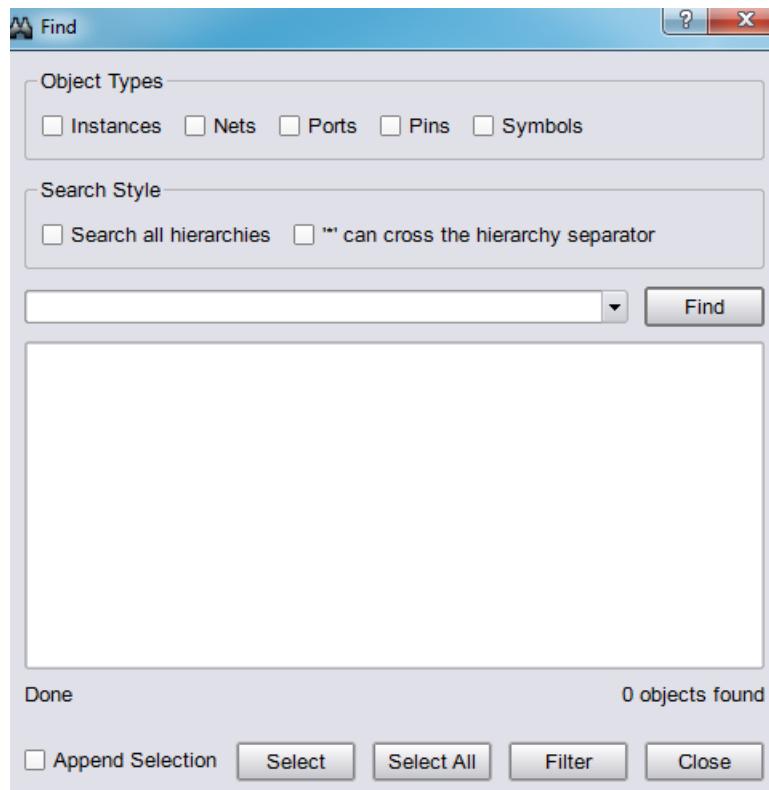
1. In a schematic, select the search icon ( ) or press Ctrl-f to open the Find dialog box.
2. Do the following in the dialog box:
  - Select the type of objects to find: instances, nets, ports, pins, and/or symbols.
  - Specify how you want the search to occur: for all hierarchies and/or allow \* to search across the hierarchy separator.

- Specify whether to search using case-sensitive designations for objects.
- Start the search either from the top level or current level of the schematic view. Use the tool tip in this dialog box to display the current level starting point.



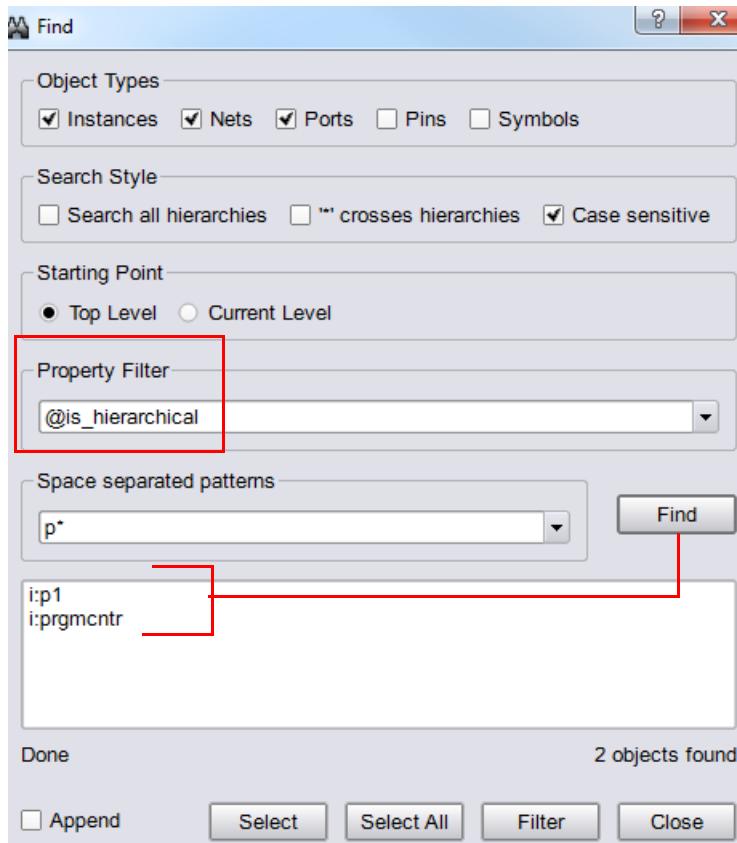
When searching, note the following:

- Double-click on a selected object from the dialog box to filter it in the schematic view.
  - Use the Space separated patterns field to search for multiple patterns, specifying the patterns with spaces in the search field.
  - Once multiple objects have been selected from the dialog box, you can highlight them, then copy and paste them to the Tcl window or in a text file.
3. Select on an object displayed in the dialog box below, then click the Select button. Click the Filter button, to select the specified objects and filter them in the HDL Analyst view.
- Click the Close button to end the Find search. Then, you can use the Filter command to display the objects.



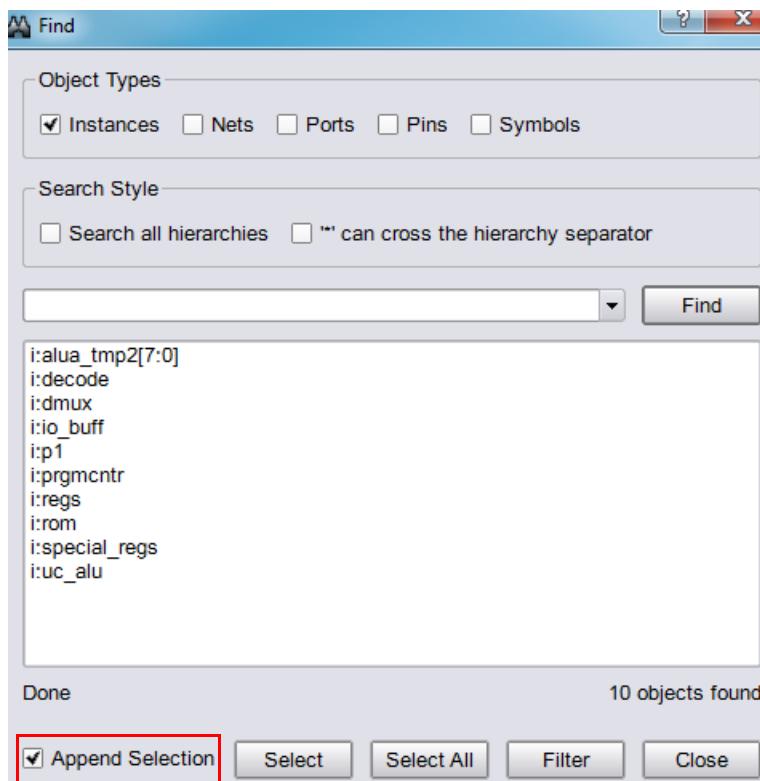
When the search style options (Search all hierarchies and ".,"can cross the hierarchy separator) are not enabled, the software searches for objects at the top level.

4. You can filter on the results found for objects based on the Property Filter field and using the search patterns specified in the Space separated patterns field. For example, suppose you want to search for the @is\_hierarchical property for the design. Specify the pattern as shown in the following dialog box and click Find. The results are displayed below:



Select and filter as needed.

5. If you enable the Append option, objects selected in the current display window are appended to each other when you click the Select or Select All button. Otherwise, objects will be overridden after each selection.



6. You can also search for multiple patterns, then filter them in the schematic view by clicking the Filter button.
7. If you determine that the search is taking too long to run, notice that the Find button changes to Stop. Click Stop.

The multi-threaded Find command can be interrupted and canceled once the search term has been identified. If you let the search complete, you will see Finishing and Done appearing under the display window.

## Using Wildcards with the Find Command

Use the following wildcards when you search the schematics:

- \*     The asterisk matches any sequence of characters.
- ?     The question mark matches any single character, but not the hierarchy separator by default.
- .     The dot explicitly matches a hierarchy separator, so type one dot for each level of hierarchy. To use the dot as a pattern and not a hierarchy separator, type a backslash before the dot: \.

# Crossprobing

Crossprobing is the process of selecting an object in one view and having the object or the corresponding logic automatically highlighted in other views. Crossprobing helps you visualize where coding changes or timing constraints might help to reduce area or improve performance.

This section describes how to crossprobe from different views. It includes the following:

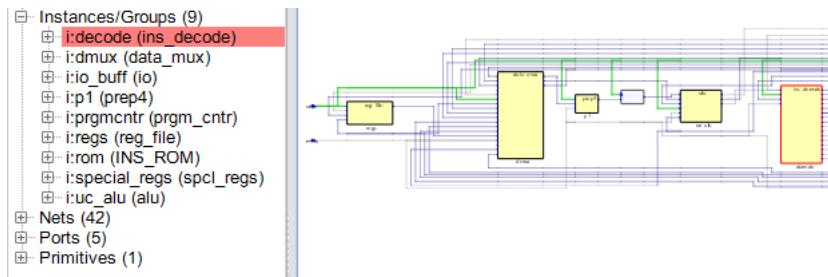
- [Crossprobing within a View, on page 124](#)
- [Crossprobing from an HDL Analyst View, on page 125](#)
- [Crossprobing to the Source Code, on page 127](#)
- [Crossprobing from the Text Editor Window, on page 129](#)
- [Crossprobing from the Log File, on page 131](#)

## Crossprobing within a View

Selecting an object name in the Hierarchy Browser highlights the object in the schematic, and vice versa.

| Selected Object                      | Highlighted Object               |
|--------------------------------------|----------------------------------|
| Instance in schematic (single-click) | Module icon in Hierarchy Browser |
| Net in schematic                     | Net icon in Hierarchy Browser    |
| Port in schematic                    | Port icon in Hierarchy Browser   |
| Logic icon in Hierarchy Browser      | Instance in schematic            |
| Net icon in Hierarchy Browser        | Net in schematic                 |
| Port icon in Hierarchy Browser       | Port in schematic                |

In this example, when you select the DECODE module in the Hierarchy Browser, the DECODE module is automatically selected in the view.

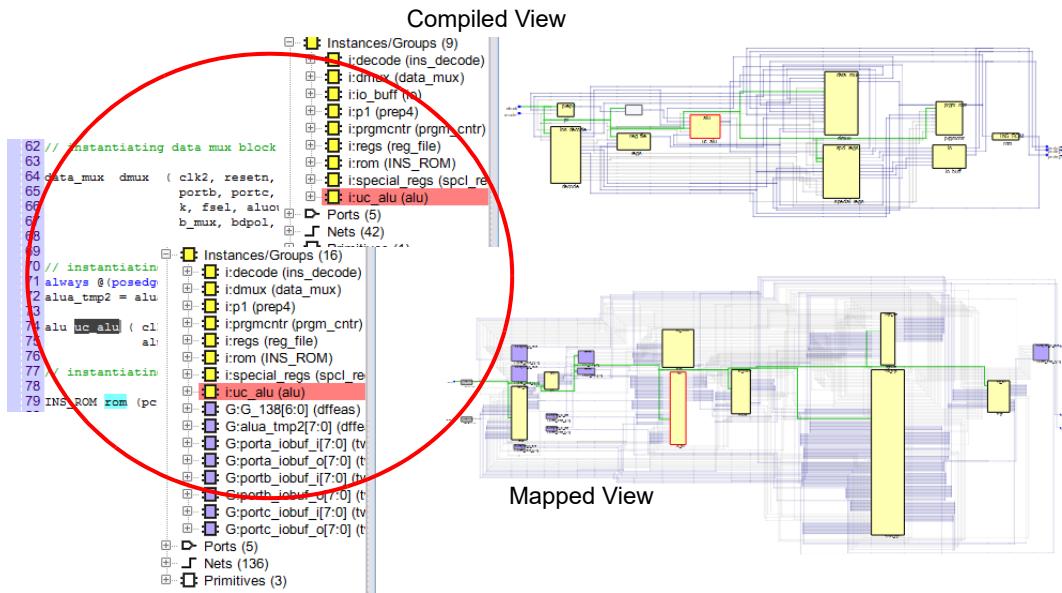


## Crossprobing from an HDL Analyst View

To crossprobe from the schematic to other open views or the source code files, select the object by clicking on it.

| Crossprobing              | Description                                                                                                                                                                                                                         |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Between HDL Analyst views | You can crossprobe: <ul style="list-style-type: none"><li>Between the compiled and mapped views</li><li>Between the compiled/mapped and hierarchy browser views</li><li>Between the pre-partitioned and partitioned views</li></ul> |
| To the source code        | For details, see <a href="#">Crossprobing to the Source Code , on page 127</a>                                                                                                                                                      |
| From the text editor      | For details, see <a href="#">Crossprobing from the Text Editor Window , on page 129</a>                                                                                                                                             |
| From the log file         | For details, see <a href="#">Crossprobing from the Log File , on page 131</a>                                                                                                                                                       |

The software automatically highlights the object in all open views. If the open view is a schematic, the software highlights the object in the Hierarchy Browser on the left as well as in the schematic. If the highlighted object is in another hierarchy of a schematic, the view does not automatically track to the hierarchy. You may have to filter the schematic.

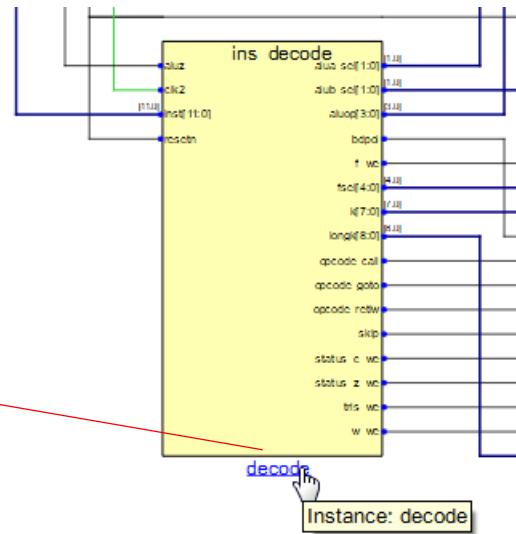


To crossprobe from the schematic to a source file when the source file is not open, the instance names must be the same. Notice that when you hover over an instance name in the schematic, it turns blue. You can click on this link, to automatically open the editor window of the source code file and highlight the appropriate code as shown below. A message is generated if a match cannot be found.

```

32 wire [7:0] trisa;
33 wire [7:0] trisb;
34 wire [7:0] trisc;
35 wire clk1, clk2, clk3, clk4;
36 wire [7:0] alua_tmp;
37 assign clk1 = clock;
38 assign clk2 = clock;
39 assign clk3 = clock;
40 assign clk4 = clock;
41 assign rtcc = 0;
42 assign b_mux = 0;
43 // instantiating prep4 block
44 prep4 p1 (alua_tmp, alua, clk3, resetn);
45
46 // instantiating decode block
47 ins_decode decode //clk2, resetn, aluz, in:
48     status_z_we, status_c,
49     skip, k, fsel, longk,
50     alua_sel, alub_sel, b
51

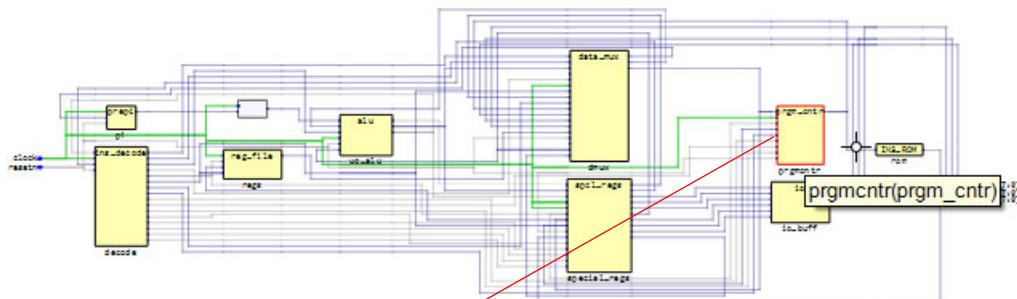
```



## Crossprobing to the Source Code

You can easily crossprobe instances or modules in the HDL Analyst view to the source code. To do this, choose either to:

- Highlight an instance in the HDL Analyst view, then right-click and select View Instance in Source from the drop-down menu. The tool automatically crossprobes to this instance in the source code.

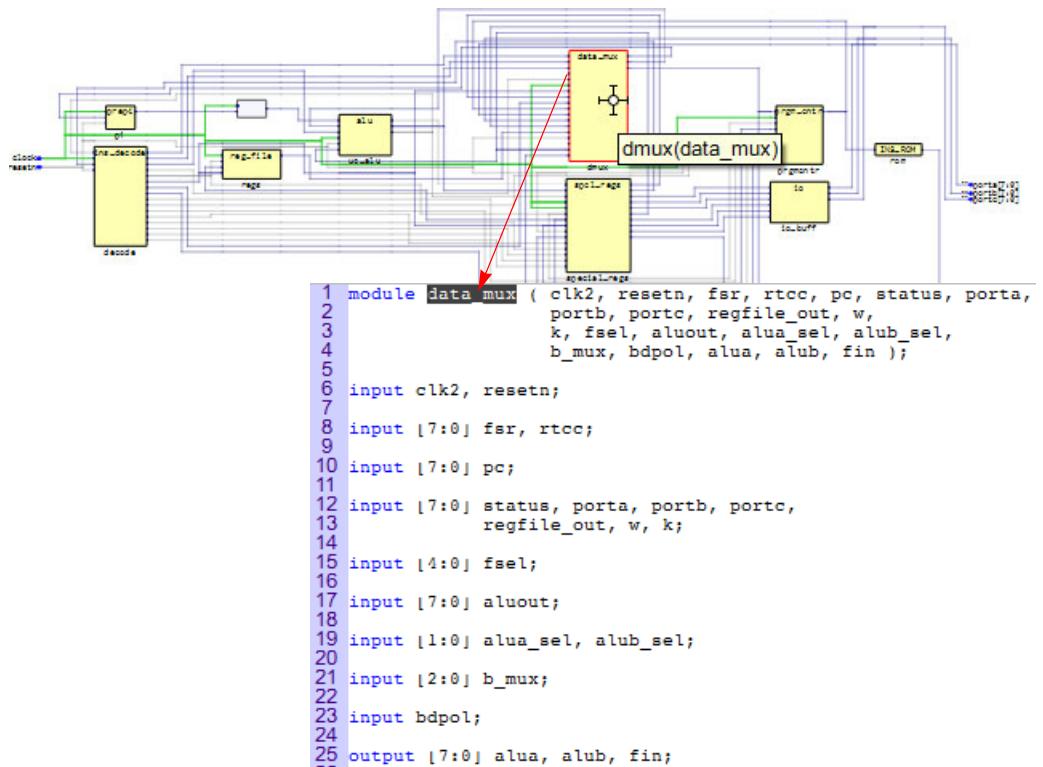


```

47 ins_decode decode (clk2, resetn, aluz, inst, f_we, w_we,
48                      status_z_we, status_c_we, tris_we,
49                      skip, k, fsel, longk, aluop,
50                      alua_sel, alub_sel, bdpol, opcode_goto, opcode_call, opcode_retlw );
51
52
53 // instantiating program counter block
54 prgm_cntr prgm_cntr ( clk4, resetn, f_we, longk, fsel,
55                         opcode_goto, opcode_call, opcode_retlw, fin, pc );
56
57
58 // instantiating regs block
59 reg_file regs (clk1, f_we, fsel, fin, regfile_out);
60
61
62 // instantiating data mux block
63
64 data_mux dmux ( clk2, resetn, fsr, rtcc, pc[7:0], status, porta,
65                  portb, portc, regfile_out, w,
66                  k, fsel, aluout, alua_sel, alub_sel,
67                  b_mux, bdpol, alua, alub, fin );

```

- Highlight a module in the HDL Analyst view, then right-click and select View Module in Source from the drop-down menu. The tool automatically crossprobes to this module in the source code.



Note that you can crossprobe to instances and modules in the source code from the Hierarchy Browser as well. Highlight an object, then right-click and select View Instance in Source or View Module in Source from the drop-down menu.

## Crossprobing from the Text Editor Window

To crossprobe source in the text editor window or from the log file to a schematic, use this procedure. You can use this method to crossprobe from any text file with objects that have the same instance names as in the synthesis software.

1. Open the schematic to which you want to crossprobe.
2. Select the appropriate portion of text in the Text Editor window. In some cases, it may be necessary to select an entire block of text to crossprobe.

3. You can choose either to:

- Highlight the objects in the path, right-click and select Filter in Analyst from the drop-down menu. The tool automatically filters the schematic so that you see just the selected objects in the view.
- Highlight the objects in the path, right-click and select Select in Analyst from the drop-down menu. You might have to filter the selected objects to see them displayed in the schematic.

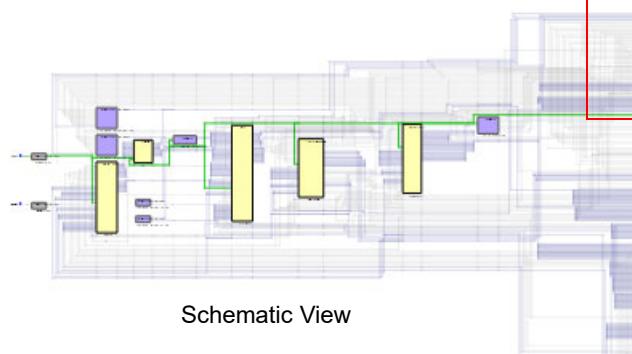
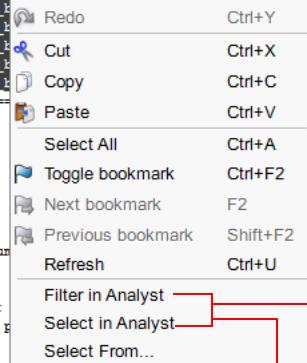
For example:

#### Text Editor

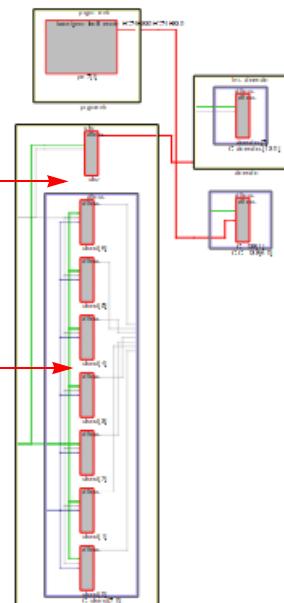
```

635 ****
636
637           Starting
638 Instance      Reference   Type     Pin
639          Clock
640 -----
641 uc_alu.aluz    eight_bit_uc|clock  dffeas   d
642 uc_alu.aluout[2] eight_bit_uc|clock  dffeas   d
643 uc_alu.aluout[1] eight_bit_uc|clock  dffeas   d
644 uc_alu.aluout[3] eight_bit_uc|clock  dffeas   d
645 uc_alu.aluout[0] eight_bit_uc|clock  dffeas   d
646 uc_alu.aluout[4] eight_bit_uc|clock  dffeas   d
647 uc_alu.aluout[5] eight_bit_uc|clock  dffeas   d
648 uc_alu.aluout[6] eight_bit_uc|clock  dffeas   d
649 decode.decode[7] eight_bit_uc|clock  dffeas   d
650 G_138[1]        eight_bit_uc|clock  dffeas   d
651
652
653
654
655 Worst Path Information
656 ****
657
658
659 Path information for path number 1
660 Requested Period:
661 - Setup time:
662 + Intrinsic clock delay:
663 + Clock delay at ending register
664 = Required time:
665

```



#### Schematic View (Filtered)



## Crossprobing from the Log File

The log file contains handy links, such as, clock trees driving clock pins of sequential elements and worst paths for the design to the HDL Analyst view. For example:

- Click on the View Worst Path in Analyst link in the log file.

```

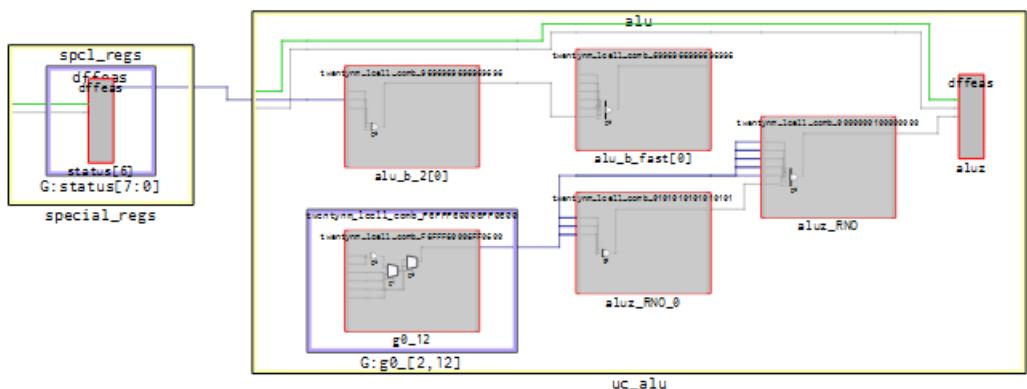
Worst Path Information
View Worst Path in Analyst
*****  

Path information for path number 1:
  Requested Period: 3.970
  - Setup time: -0.020
  + Intrinsic clock delay: 0.618
  + Clock delay at ending point: 0.000 (ideal)
  = Required time: 4.608

  - Propagation time: 6.815
  - Intrinsic clock delay: 0.618
  - Clock delay at starting point: 0.000 (ideal)
  = Slack (critical) : -2.825

  Number of logic level(s): 6
  Starting point: special_regs.status[6] / q
  Ending point: uc_alu.aluz / d
  The start point is clocked by eight_bit_ue|clock [rising] on pin clk
  The end point is clocked by eight_bit_ue|clock [rising] on pin clk
*****
```

- The schematic for this critical path is automatically displayed in the mapped view shown below.



# Analyzing With the HDL Analyst Tool

The HDL Analyst tool is a graphical productivity tool that helps you visualize your synthesis results. It displays schematics of the design at different stages, allowing you to graphically view and analyze your design. At an early design stage, the schematic displays high-level structures like RAM, ROM, operators, and FSM as abstractions. Later in the cycle, these structures are converted to gates and mapped to technology-specific resources.

To analyze information or compare views with the log file, the FSM view, and the source code, you can use techniques like crossprobing, flattening, and filtering. See the following for more information about analysis techniques.

- [Viewing Design Hierarchy and Context, on page 132](#)
- [Filtering Schematics, on page 138](#)
- [Expanding Pin and Net Logic, on page 140](#)
- [Dissolving and Partial Dissolving of Buses and Pins, on page 146](#)
- [Flattening Schematic Hierarchy, on page 150](#)

For additional information about navigating the HDL Analyst views or using other techniques like crossprobing, see the following:

- [Working in the Schematic, on page 86](#)
- [Exploring Design Hierarchy, on page 106](#)
- [Finding Objects, on page 113](#)
- [Crossprobing, on page 124](#)

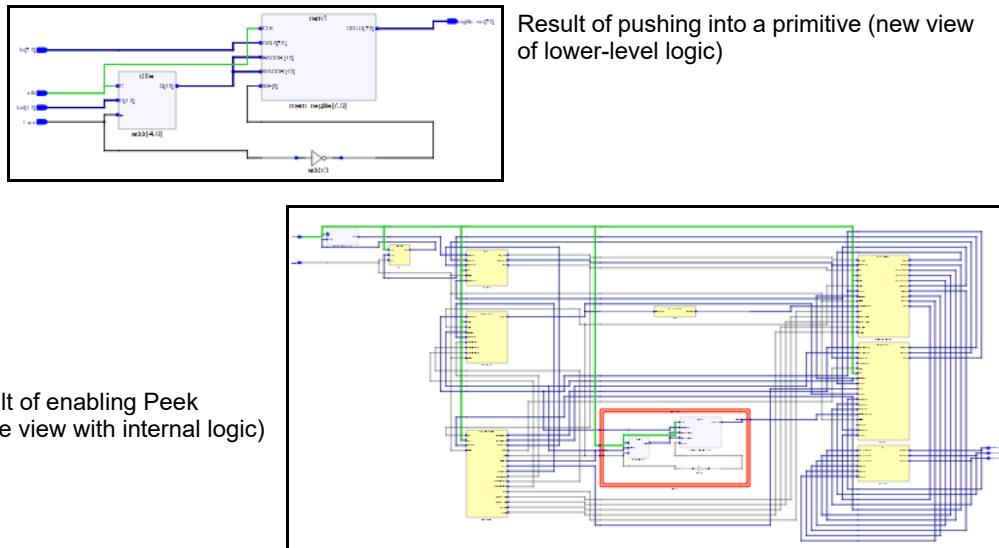
## Viewing Design Hierarchy and Context

Most large designs are hierarchical, so the software provides tools that help you view hierarchy details or put the details in context. Alternatively, you can browse and navigate hierarchy with the Push/Pop command, or flatten the design to view internal hierarchy.

This section describes how to use interactive hierarchical viewing operations to better analyze your design. Automatic hierarchy viewing operations that are built into other commands are described in the context in which they appear.

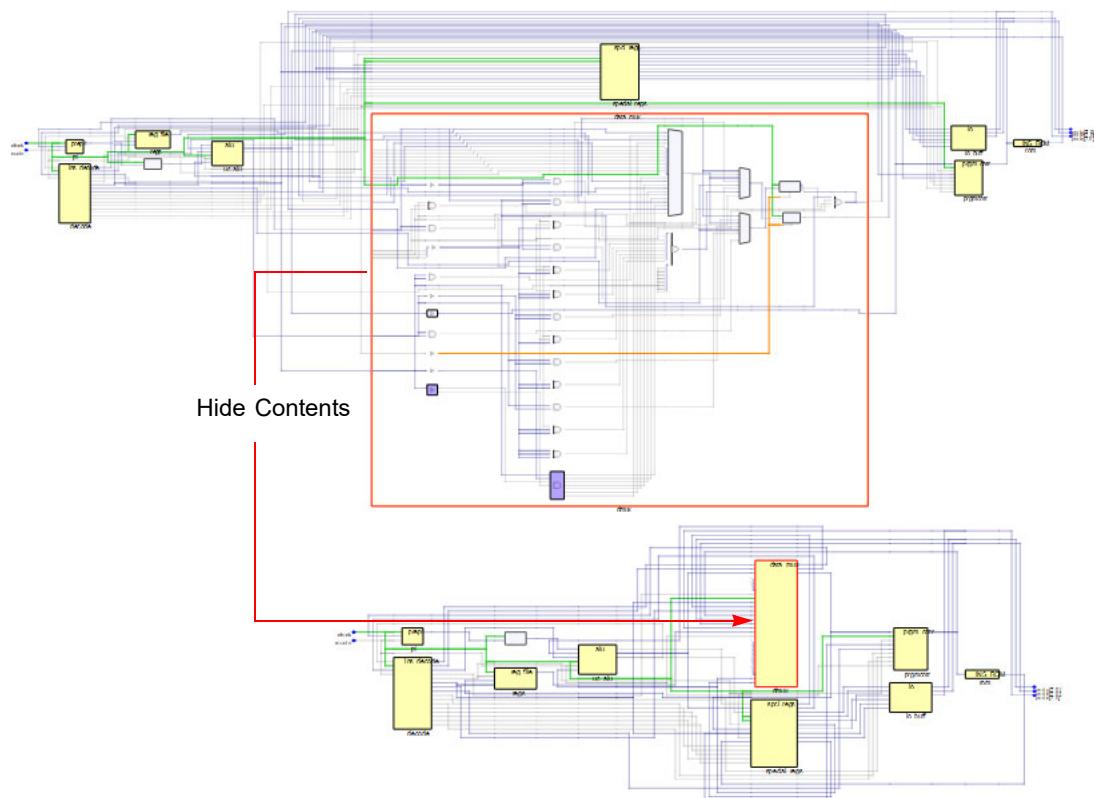
1. To view the internal logic of instances in your design, do either of the following:
  - To view the logic of an individual instance, push into it. This generates a new schematic with the internal details. Click the Back icon to return to the previous view.
  - To view the logic of all instances in the design, select all the required instances and right-click then select Peek. This command lets you see internal logic in context, by adding the internal details to the current schematic. If the view is too cluttered with this option on, filter the view (see [Filtering Schematics, on page 138](#)) or push into the primitive. Click the Back icon to return to the previous view after filtering or pushing into the object.

The following figure compares these two methods:



The technology primitives that are contained in a block are displayed automatically when you zoom in on them in the schematic view.

2. Suppose you just used the peek option to see the internal logic of an instance. To return back to the schematic state before using peek and while the peek objects are still highlighted, right-click and select Hide Contents from the drop-down menu.



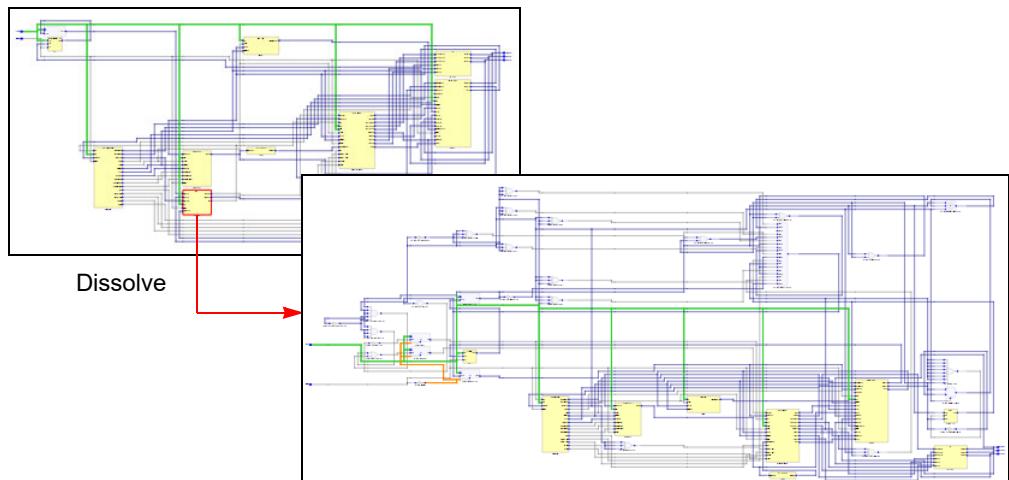
3. To view the internal logic of a hierarchical instance, you can push into the instance, dissolve the selected instance with the Dissolve command, or flatten the design.

---

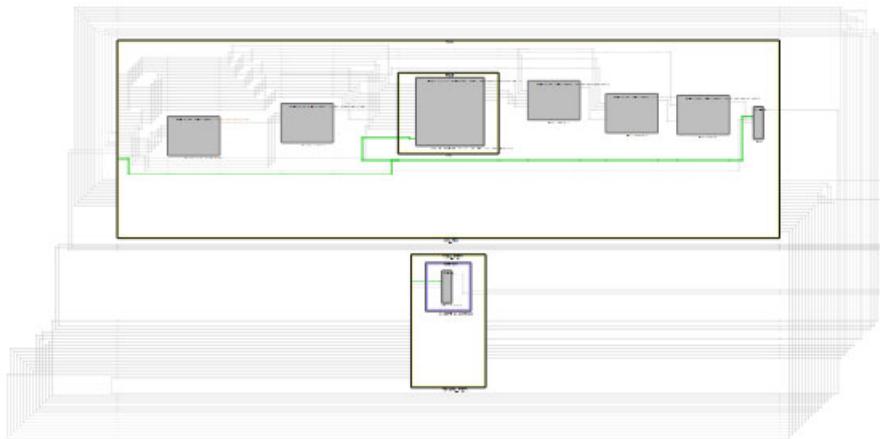
|                                      |                                                                                                                                                                                                                                                        |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Pushing into an instance             | Generates a view that shows only the internal logic. You do not see the internal hierarchy in context. To return to the previous view, click Back. See <a href="#">Exploring Object Hierarchy with Push/Pop Commands</a> , on page 108 for details.    |
| Flattening the entire design         | Opens a view where the entire design is flattened. Large flattened designs can be overwhelming. See <a href="#">Flattening Schematic Hierarchy</a> , on page 150 for details about flattening designs.                                                 |
| Flattening an instance by dissolving | Generates a view where the hierarchy of the selected instances is flattened, but the rest of the design is unaffected. This provides context. See <a href="#">Flattening Schematic Hierarchy</a> , on page 150 for details about dissolving instances. |

---

The following schematic shows an instance that has been dissolved in the view.



4. The software automatically traces a critical path through different hierarchical levels using hollow boxes with nested internal logic (transparent instances) to indicate levels in hierarchical instances.



## Browsing With the New Hierarchy Browser

*Beta*

Using the HDL Analyst tool, you can display a hierarchy of design objects in the Hierarchy Browser. Typically, the time taken to display a design hierarchy depends on the size of the design.

To speed up this process, the new Hierarchy Browser traverses the entire (text-based) netlist to quickly extract hierarchical instance data. This helps to display the entire netlist hierarchy quickly and also facilitates the viewing of custom instances on demand, instead of traversing down the design hierarchy. This flow is advantageous in large designs, to display the hierarchy and view any instance, quickly.

To enable the new Hierarchy Browser, follow these steps:

1. To set preference in the schematic from the user interface, select one of the following:
  - HDL Analyst->Schematic Options
  - Options->Schematics Options

The HDL Analyst Options dialog box is displayed.

2. Enable the Use new hierarchy browser option on the HDL Analyst Options dialog box. A warning is displayed, as shown below.



When you enable the Use new hierarchy browser option in the HDL Analyst Options dialog box, the HDL Analyst tool displays the RTL schematic at the instance level (Instance Hierarchy tab) and design level (Design View tab), when the next schematic displayed.

The Instance Hierarchy tab is visible only when the Use new hierarchy browser option is enabled in the Schematic Options (HDL Analyst Options) dialog box.

3. Click OK to close the dialog box. Now, each design you view will follow the updated option settings.
4. Select the RTL View icon to view the RTL view of the design.

#### Browsing an Object by Filtering or Loading

1. To filter a particular instance, type the instance name into the text box available below the Hierarchy Browser pane. The instance names become visible in the drop-down as you continue to type.
2. Choose the instance and click Filter to display the instance in the RTL schematic view.
3. To search through a hierarchical path, select From Top. If this is not checked, the command searches the entire design.
4. To load an instance from the Hierarchy Browser, right-click on the desired instance and select the Open Instance View ... option. The instance is displayed in the RTL schematic view. The Instance Hierarchy tab is visible only when the Use new hierarchy browser option is enabled in the Schematic Options (HDL Analyst Options) dialog box.

New Hierarchy Browser Limitations:

- Consider these limitations before using the new Hierarchy Browser flow:
- Currently, usable only with compiled-stage/mapped-stage netlists.
- In map designs, primitives and black boxes appear in the hierarchy but not in the RTL view.
- Certain view options while filtering or loading an instance from the new Hierarchy Browser do not work correctly.

## Filtering Schematics

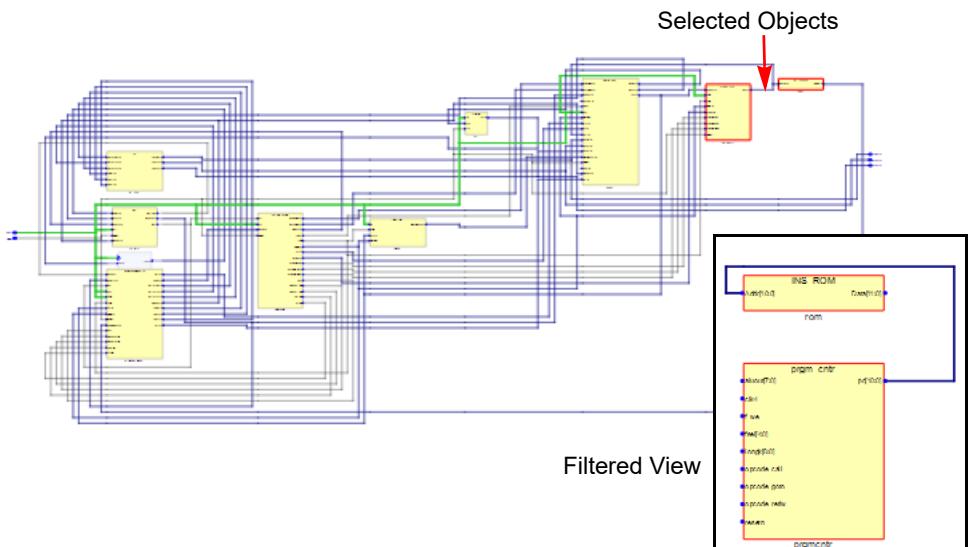
Filtering is a useful first step in analysis, because it focuses analysis on the relevant parts of the design. Some commands, like the Expand commands, automatically generate filtered views; this procedure only discusses manual filtering, where you use the Filter command to isolate selected objects.

This table lists the advantages of using filtering over flattening:

| Filter Schematic Command                                                                                                       | Flatten Commands                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| Loads part of the design; better memory usage                                                                                  | Loads entire design                                                                                  |
| Combine filtering with the Push/Pop command, and history buttons (Back and Forward) to move freely between hierarchical levels | You can use the Back arrow or Show Top View icon to return to previous view that has been flattened. |

1. Select the objects that you want to isolate. For example, you can select two connected objects.
2. Select the Filter command, using one of these methods:
  - Right-click and select Filter from the popup menu.
  - Click the Filter icon (buffer gate) (

The software filters the design and displays the selected objects in a filtered view. These objects are isolated in the schematic displayed. Select Unfilter to take you back to the view where objects are at the same level.



You can now analyze the problem, and do operations like the following:

---

|                               |                                                                                                                 |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Trace paths, build up logic   | See <a href="#">Expanding Pin and Net Logic</a> , on page 140                                                   |
| Filter further                | Select objects and filter again                                                                                 |
| Find objects                  | See <a href="#">Finding Objects</a> , on page 113                                                               |
| Flatten                       | See <a href="#">Flattening Schematic Hierarchy</a> , on page 150. You can hide transparent or opaque instances. |
| Crossprobe from filtered view | See <a href="#">Crossprobing from an HDL Analyst View</a> , on page 125                                         |

---

3. To return to the previous schematic, click the Back arrow. If you flattened the hierarchy, right-click and select the Back arrow or the Show Top View icon to return to the top-level unflattened view.

For additional information about filtering schematics, see [Filtering Schematics, on page 138](#) and [Flattening Schematic Hierarchy, on page 150](#).

## Expanding Pin and Net Logic

When you are working in a filtered view, you might need to include more logic in your selected set to debug your design.

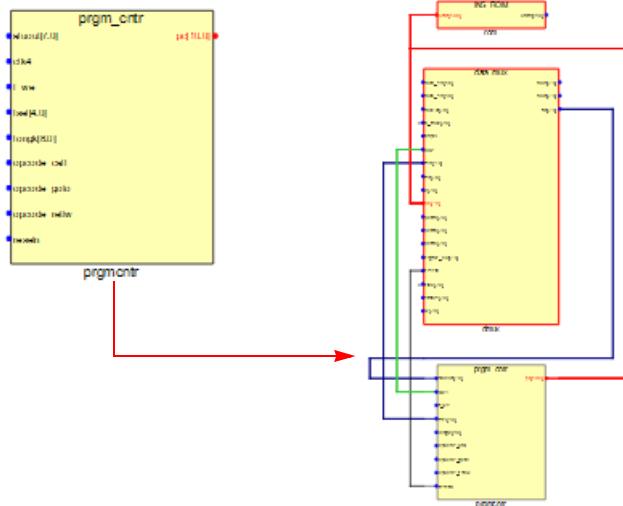
Use the Expand commands with the Filter and Flatten commands to isolate just the logic that you want to examine. Filtering isolates logic, flattening removes hierarchy. See [Filtering Schematics, on page 138](#) and [Flattening Schematic Hierarchy, on page 150](#) for details.

1. To expand logic from a pin hierarchically across boundaries, use the following commands.

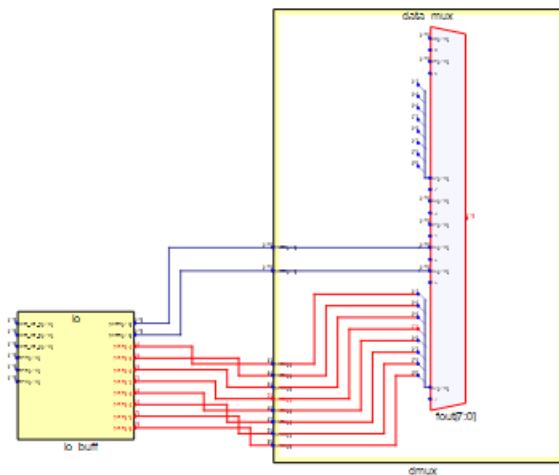
| To ...                                                                                                 | Do this ...                                                                                          |
|--------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| See the first-level cells connected to a pin in the same hierarchy                                     | Select a pin and select Expand. See <a href="#">Expanding Filtered Logic Example , on page 141</a> . |
| See the first-level cells connected to a pin at any level of hierarchy                                 | Select a pin and select Hierarchical Expand.                                                         |
| See all cells until a register or port is connected to the selected pin at the same level of hierarchy | Select a pin and select Expand to Reg/Port.                                                          |

| To ...                                                                                            | Do this ...                                                                                                                                                                                  |
|---------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| See internal cells connected to a pin                                                             | Select a pin and select Expand Inwards. The software filters the schematic and displays the internal cells closest to the port. See <a href="#">Expanding Inwards Example , on page 142.</a> |
| Select only one object connected to a port or pin                                                 | Select a pin or port and select Expand to One Object.                                                                                                                                        |
| See all cells until a register or port is connected to the selected pin at any level of hierarchy | Select a pin and select Hierarchical Expand to Reg/Port.                                                                                                                                     |

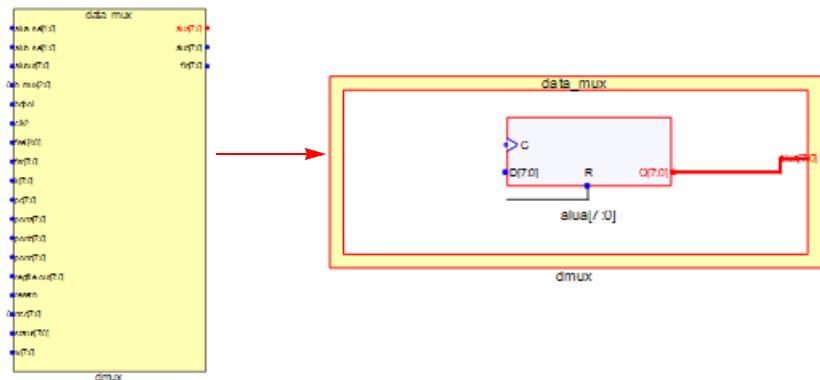
## Expanding Filtered Logic Example



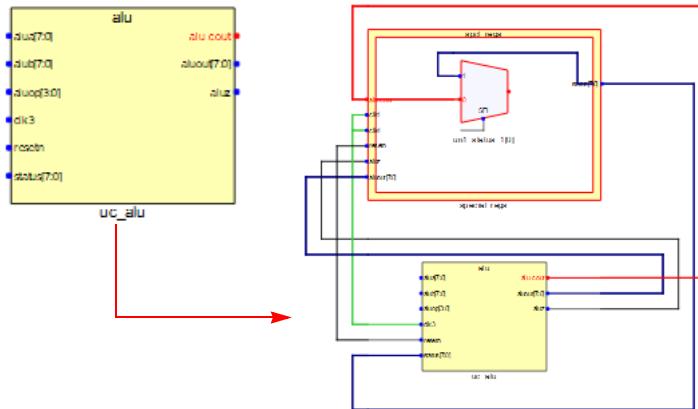
## Expand to One Object



## Expanding Inwards Example



## Expanding Hierarchically

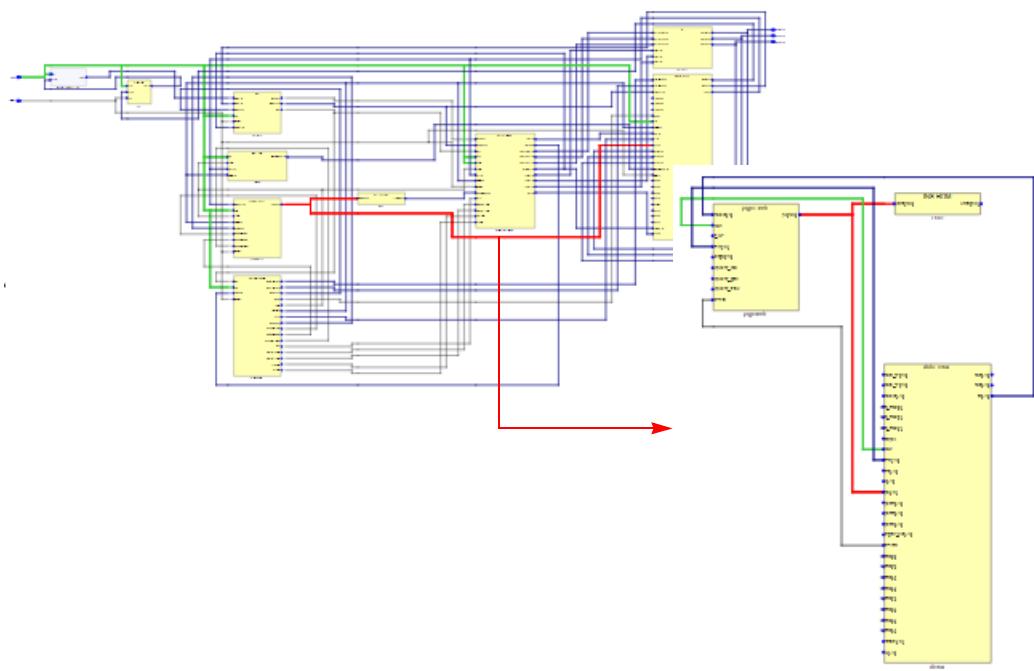


2. To expand logic from a net, do the following:
  - Use the commands shown in the following table.
  - Select a net, then right-click and select the command from the right-click options.

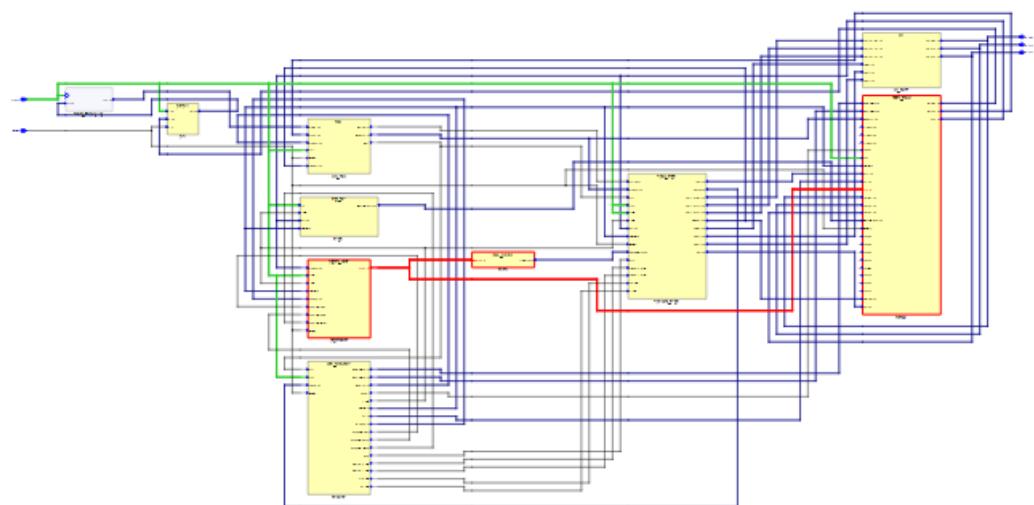
| To ...                                                                                                                                                                     | Do this ...                                                            |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|
| See all instances connected to the selected net being filtered                                                                                                             | Select a net and select Filter by Nets.                                |
| Select the instances in the same hierarchy connected to the selected net                                                                                                   | Select a net and select Expand Nets.                                   |
| Select and show instances connected to the selected net at any level of hierarchy. The instance that drives the net and the instance which is driven by the net are shown. | Select a net and select Hierarchical Expand Nets.                      |
| Select and show instances connected to the selected net at any level of hierarchy. Instances that are not connected are removed from the view.                             | Select a net, then Filter by Net, and select Hierarchical Expand Nets. |

The following figures illustrate this.

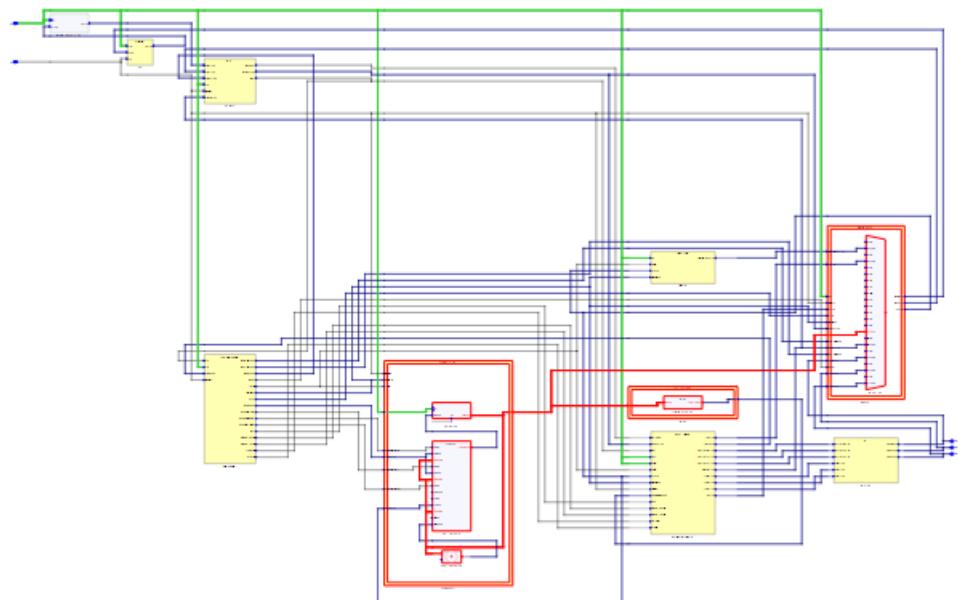
## Filter by Nets



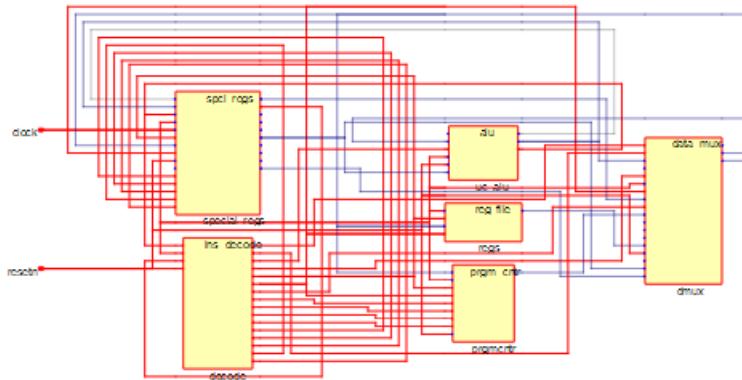
## Expand Nets



## Hierarchical Expand Nets



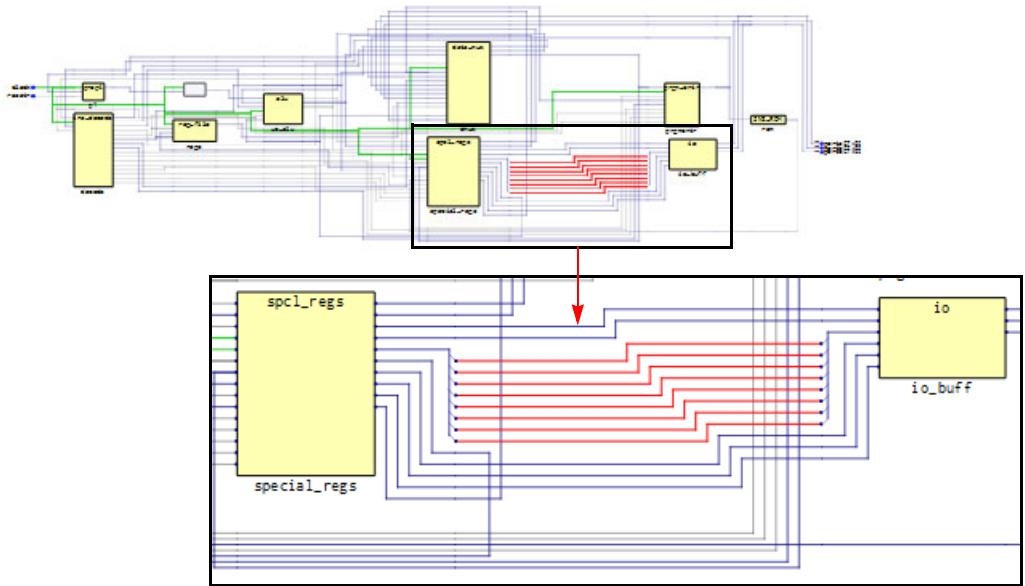
3. You can also isolate the paths to generate a schematic for a path between objects. To display connections to and from the selected instance, highlight it then right-click and select Isolate Paths from the drop-down menu.



## Dissolving and Partial Dissolving of Buses and Pins

The HDL Analyst tool has options for handling buses and pins in the display that can help you analyze your design easier. You can expand logic from a bus port or specific bits of a port.

1. To expand logic for all nets of a bus:
  - Select a bus.
  - Right-click and select Dissolve from the drop-down menu.
  - Filter by net and choose an operation to expand as needed; see [Expanding Pin and Net Logic, on page 140](#).

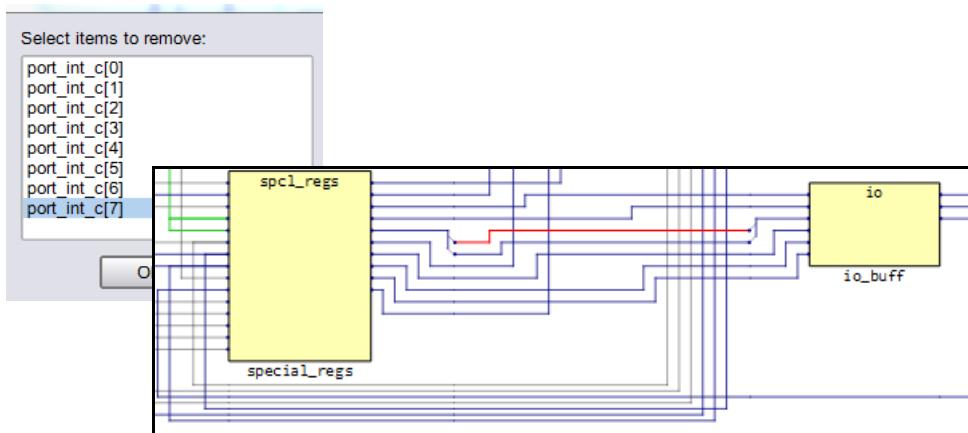


2. To expand logic from nets of a bus:

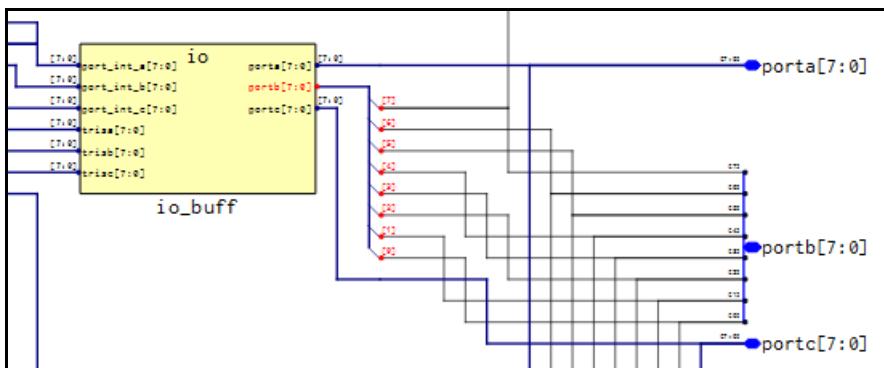
- Select a bus.
- Right-click and select Partial Dissolve from the drop-down menu.
- Select the net (`port_int_c[7]`) to be removed.
- Click OK.

The selected net is now removed from the bus.

- Choose an operation to expand as needed; see [Expanding Pin and Net Logic, on page 196](#).



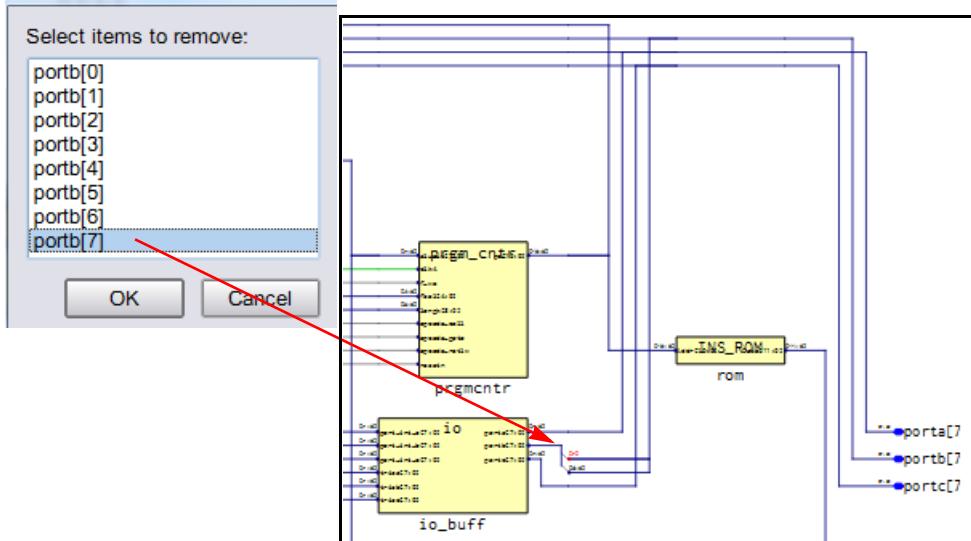
3. To expand logic for all the pins of a bus pin:
  - Select a bus pin.
  - Right-click and select Dissolve from the drop-down menu.
  - Choose an operation to expand as needed; see [Expanding Pin and Net Logic, on page 196](#).



4. To expand logic from specific bits pins of a bus pin:
  - Select a bus pin.
  - Right-click and select Partial Dissolve Pin from the drop-down menu.
  - Select the pin to be removed.
  - Click OK.

The selected pin is removed from the bus pin.

- Choose an operation to expand as needed; see [Expanding Pin and Net Logic, on page 196](#).

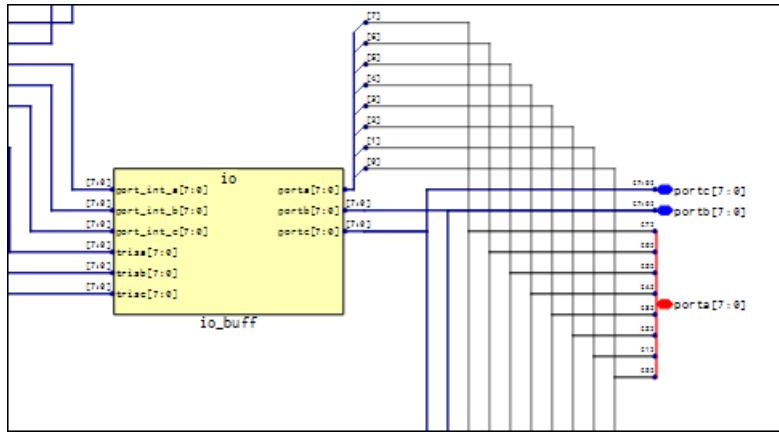


## Dissolving of Ports

The HDL Analyst tool has options for handling ports in the display that can help you analyze your design easier. You can expand logic for all bits of a port.

To expand logic for a port:

- Select a port.
- Right-click and select Dissolve from the drop-down menu.
- Choose an operation to expand as needed; see [Expanding Pin and Net Logic, on page 140](#).



## Flattening Schematic Hierarchy

Flattening removes hierarchy so you can view the logic without hierarchical levels. In most cases, you do not have to flatten your hierarchical schematic to debug and analyze your design, because you can use a combination of filtering and expanding to view logic at different levels. However, if you must flatten the design use the following techniques, which include flattening and dissolving instances.

1. To flatten any level of hierarchy to logic cells below the current level, right-click and select Flatten Schematic from the drop-down menu.

The software flattens the design hierarchy and displays it in the window. To return to the previous level, select the Back arrow.

2. To selectively flatten some hierarchical instances in your design by dissolving them, do the following:

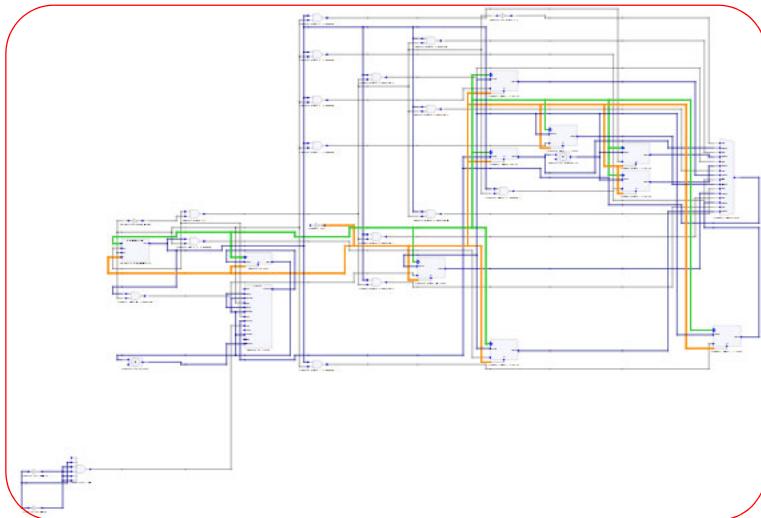
- Select the instances to be flattened.
- Right-click and select Dissolve.

The results differ slightly, depending on the kind of view from which you dissolve instances.

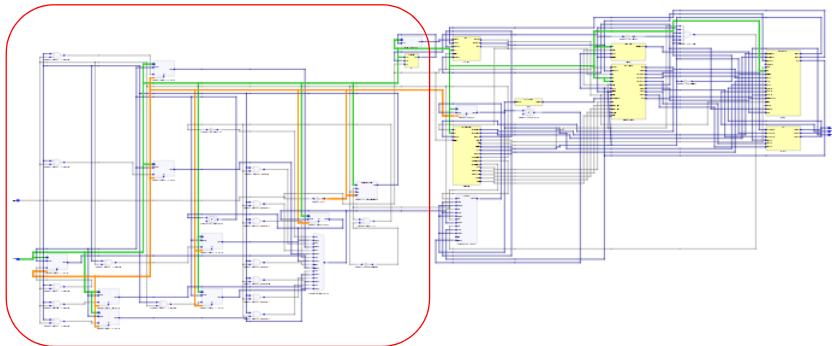
| Starting View | Software Generates a ...                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Filtered      | Filtered view with the internal logic of dissolved instances displayed within hollow bounding boxes (transparent instances), and the hierarchy of the rest of the design unchanged. If the transparent instance does not display internal logic, use one of the alternatives described in step 4 of <a href="#">Viewing Design Hierarchy and Context , on page 132</a> . Use the Back button to return to the undissolved view. |
| Unfiltered    | New, flattened view with the dissolved instances flattened in place (no nesting) to Boolean logic, and the hierarchy of the rest of the design unchanged. You can use the Back button to return to previous or the top-level views.                                                                                                                                                                                             |

The following figure illustrates this.

Dissolved logic for prgmcntr shown filtered when started from filtered view



Dissolved logic for prgmcntr shown flattened in context when you start from an unfiltered view



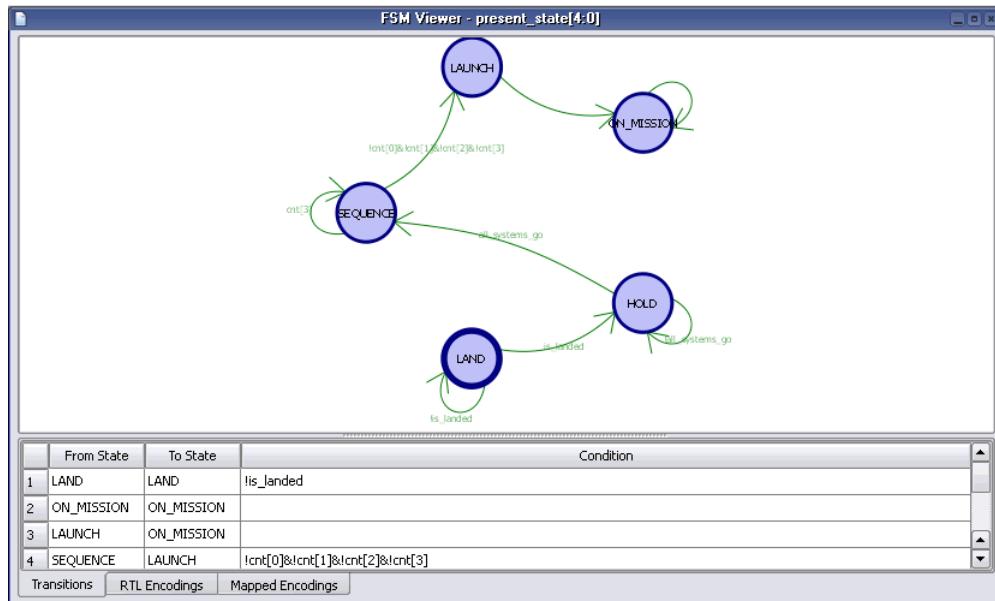
Use this technique if you only want to flatten part of your design while retaining the hierarchical context. If you want to flatten most of the design, use the technique described in the previous step. Instead of dissolving instances, you can use a combination of the filtering commands and the Push/Pop command.

## Using the FSM Viewer

The FSM viewer displays state transition bubble diagrams for FSMs in the design, along with additional information about the FSM. You can use this viewer to view state machines.

1. To start the FSM viewer, open the compiled view and highlight the FSM instance, click the right mouse button and select View State Machine from the popup menu.

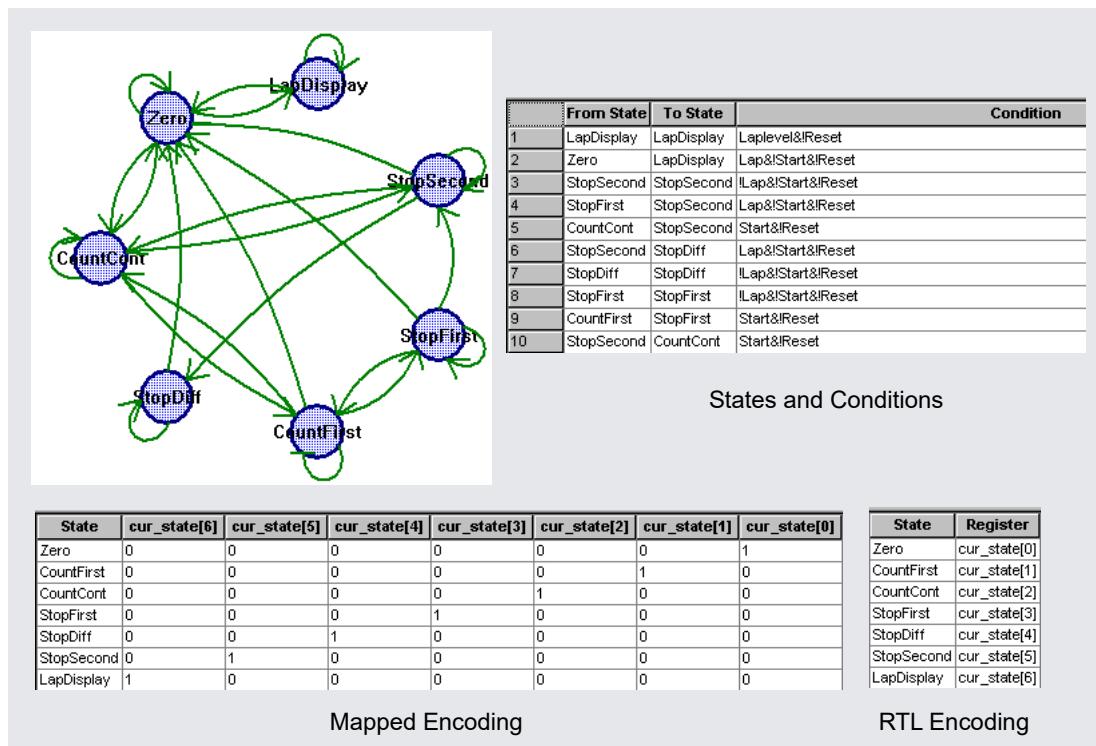
The FSM viewer opens. The viewer consists of a transition bubble diagram and a table for the encodings and transitions. If you used Verilog to define the FSMs, the viewer displays binary values for the state machines if you defined them with the 'define' keyword, and actual names if you used the parameter keyword.



2. The following table summarizes basic viewing operations.

| To view ...                                                                    | Do ...                                                                                               |
|--------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| from and to states, and conditions for each transition                         | Click the Transitions tab at the bottom of the table.                                                |
| the correspondence between the states and the FSM registers in the RTL view    | Click the RTL Encoding tab.                                                                          |
| the correspondence between the states and the registers in the Technology View | Click the Mapped Encodings tab (available after synthesis).                                          |
| only the transition diagram without the table                                  | Select View->FSM table or click the FSM Table icon. You might have to scroll to the right to see it. |

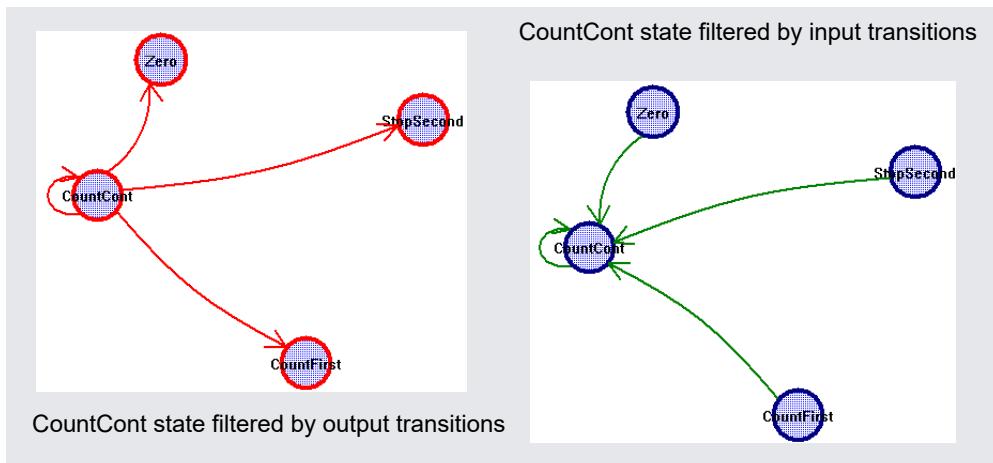
This figure shows you the mapping information for a state machine. The Transitions tab shows you simple equations for conditions for each state. The RTL Encodings tab has a State column that shows the state names in the source code, and a Registers column for the corresponding RTL encoding. The Mapped Encoding tab shows the state names in the code mapped to actual values.



3. To view just one selected state,

- Select the state by clicking on its bubble. The state is highlighted.
- Click the right mouse button and select the filtering criteria from the popup menu: output, input, or any transition.

The transition diagram now shows only the filtered states you set. The following figure shows filtered views for output and input transitions for one state.



Similarly, you can check the relationship between two or more states by selecting the states, filtering them, and checking their properties.

4. To view the properties for a state,

- Select the state.
- Click the right mouse button and select Properties from the popup menu. A form shows you the properties for that state.

To view the properties for the entire state machine like encoding style, number of states, and total number of transitions between states, deselect any selected states, click the right mouse button outside the diagram area, and select Properties from the popup menu.

# Working in the Standard HDL Analyst Schematic

The standard HDL Analyst includes schematics, which are schematics used to graphically analyze your design. This section describes basic procedures you use in the schematics. The information is organized into these topics:

- [Opening the Schematics, on page 157](#)
- [Viewing Object Properties, on page 158](#)
- [Selecting Objects in Schematics, on page 161](#)
- [Working with Multi-sheet Schematics, on page 162](#)
- [Moving Between Views in a Schematic Window, on page 163](#)
- [Setting Schematic Preferences, on page 164](#)

For information on specific tasks like analyzing critical paths, see the following sections:

- [Exploring Object Hierarchy by Pushing/Popping, on page 167](#)
- [Exploring Object Hierarchy of Transparent Instances, on page 171](#)
- [Browsing to Find Objects in HDL Analyst Views, on page 172](#)
- [Crossprobing \(Standard\), on page 183](#)
- [Analyzing With the Standard HDL Analyst Tool, on page 190](#)

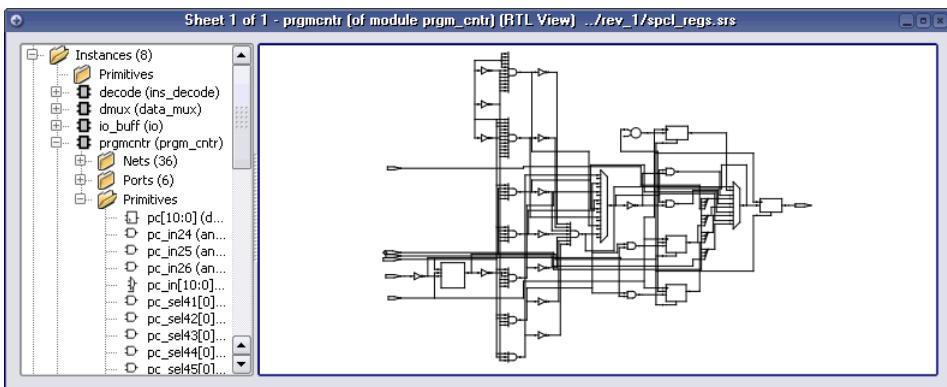
## Opening the Schematics

The procedure for opening a schematic is the same at different design stages; the main difference is the content that is available at the different design database states.

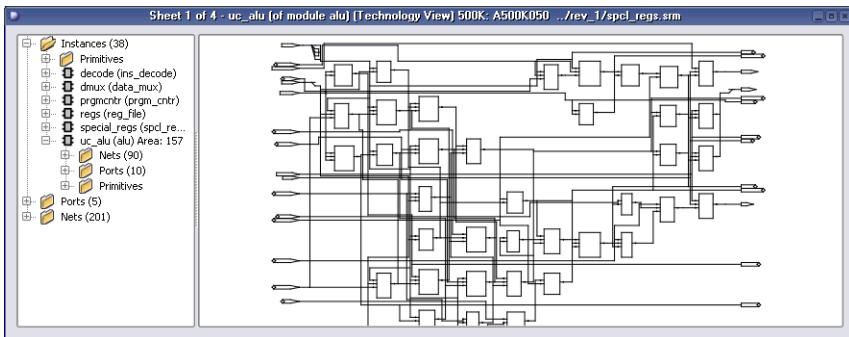
1. Start at the database state you want.
2. Open the schematic using one of the following methods:
  - At the Tcl command line, type `view schematic`.
  - Click the View the schematic icon () , a plus sign in a circle.

All schematics have the schematic on the right and a pane on the left that contains a hierarchical list of the objects in the design. This pane is called the Hierarchy Browser.

### Compiled View



### Mapped View

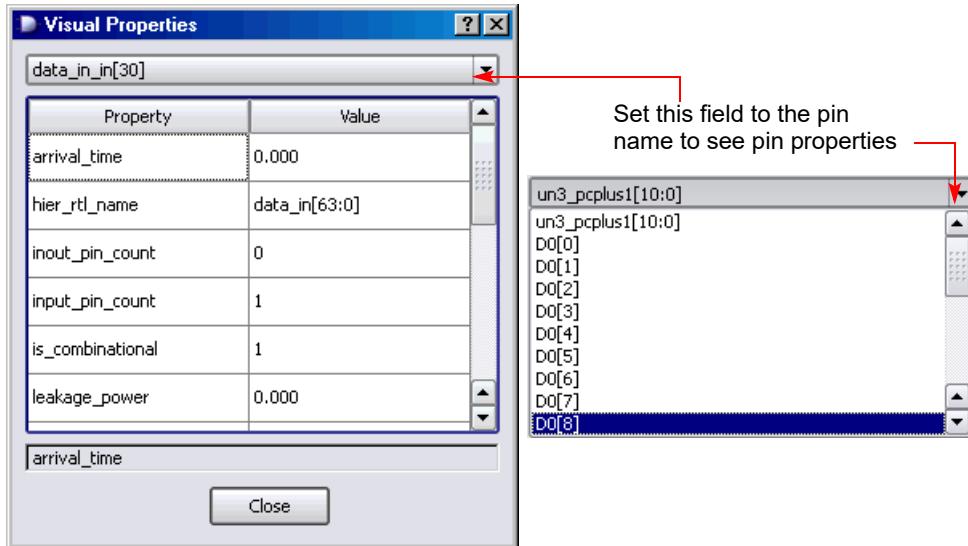


## Viewing Object Properties

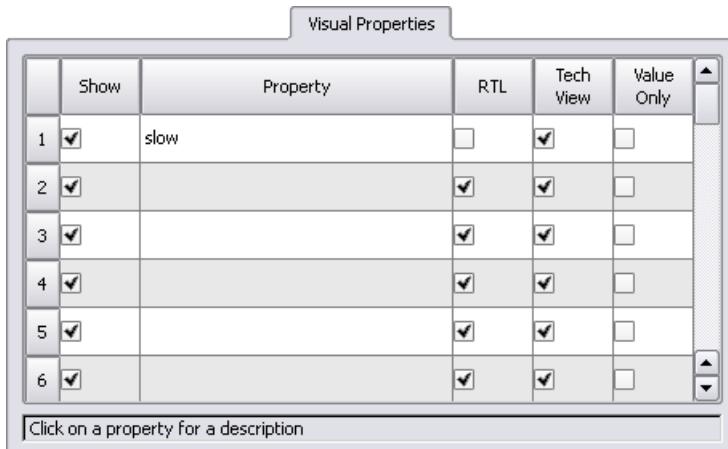
There are a few ways in which you can view the properties of objects.

1. To temporarily display the properties of a particular object, hold the cursor over the object. A tooltip temporarily displays the information, at the cursor and in the status bar at the bottom of the tool window.
2. Select the object, right-click, and select **Properties**. The properties and their values are displayed in a table.

If you select an instance, you can view the properties of the associated pins by selecting the pin from the list. Similarly, if you select a port, you can view the properties on individual bits.



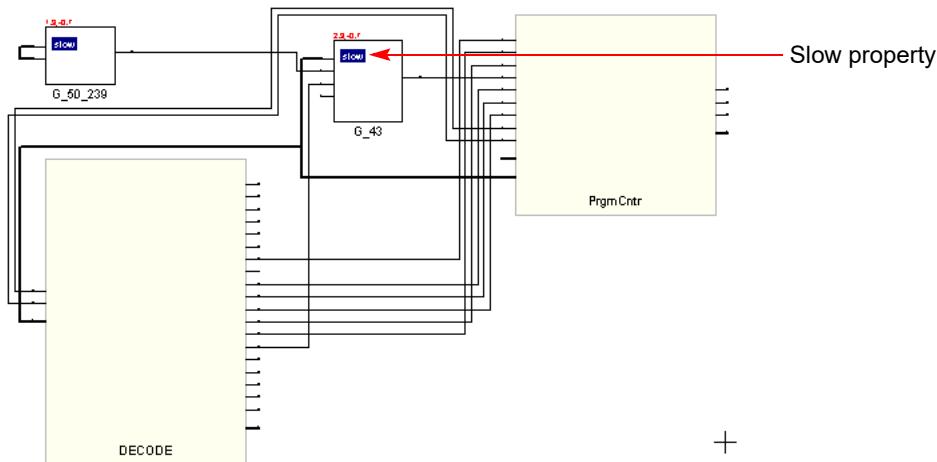
3. To flag objects by property, follow these steps:
  - Open a schematic.
  - Right-click and select HDL Analyst Options->Visual Properties, and select the properties you want to view from the pull-down list. Some properties are only available in certain views.



- Close the HDL Analyst Options dialog box.

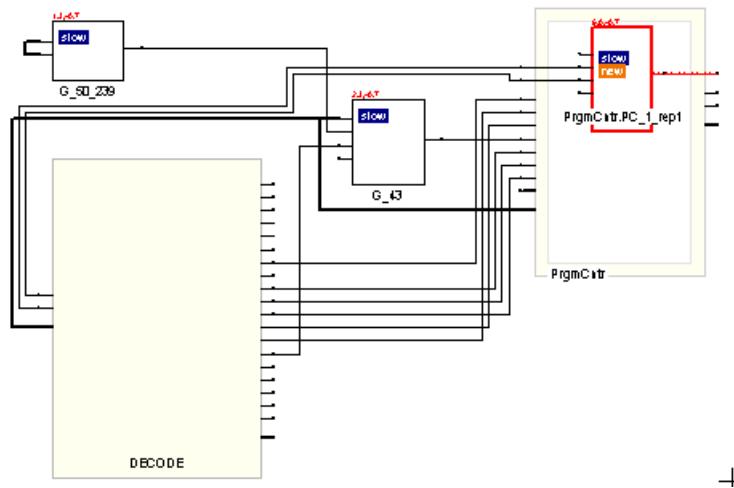
## Example: Slow and New Properties

The slow property is useful for analyzing your critical path, because it denotes objects that do not meet the timing criteria. The following figure shows a filtered view of a critical path, with slow instances flagged in blue.



The New property helps with debugging because it quickly identifies objects that have been added to the current schematic with commands like Expand. You can step through successive filtered views to determine what was added at each step.

The next figure expands one of the pins from the previous filtered view. The new instance added to the view has two flags: new and slow.



## Selecting Objects in Schematics

For mouse selection, standard object selection rules apply: In selection mode, the pointer is shaped like a cross-hair.

| To select ...                               | Do this ...                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Single objects                              | Click on the object in the schematic, or click the object name in the Hierarchy Browser.                                                                                                                                                                                                                                                                                                                                                                                     |
| Multiple objects                            | Use one of these methods: <ul style="list-style-type: none"> <li>Draw a rectangle around the objects.</li> <li>Select an object, press Ctrl, and click other objects you want to select.</li> <li>Select multiple objects in the Hierarchy Browser. See <a href="#">Browsing With the Hierarchy Browser</a>, on page 172.</li> <li>Use Find to select the objects you want. See <a href="#">Using Find for Hierarchical and Restricted Searches</a>, on page 174.</li> </ul> |
| Objects by type<br>(instances, ports, nets) | Use Edit->Find to select the objects (see <a href="#">Browsing With the Find Command</a> , on page 174), or use the Hierarchy Browser, which lists objects by type.                                                                                                                                                                                                                                                                                                          |

| To select ...                                          | Do this ...                                                                                                                                                                                                                                                                  |
|--------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| All objects of a certain type (instances, ports, nets) | To select all objects of a certain type, do either of the following: <ul style="list-style-type: none"> <li>Right-click and choose the appropriate command from the Select All Schematic/Sheet popup menus.</li> <li>Select the objects in the Hierarchy Browser.</li> </ul> |
| No objects (deselect all currently selected objects)   | Click the left mouse button in a blank area of the schematic or click the right mouse button to bring up the pop-up menu and choose Unselect All. Deselected objects are no longer highlighted.                                                                              |

The HDL Analyst view highlights selected objects in red. If the object you select is on another sheet of the schematic, the schematic tracks to the appropriate sheet. If you have other windows open, the selected object is highlighted in the other windows as well (crossprobing), but the other windows do not track to the correct sheet. Selected nets that span different hierarchical levels are highlighted on all the levels. See [Crossprobing \(Standard\), on page 183](#) for more information about crossprobing.

Some commands affect selection by adding to the selected set of objects: the Expand commands, the Select All commands, and the Select Net Driver and Select Net Instances commands.

## Working with Multi-sheet Schematics

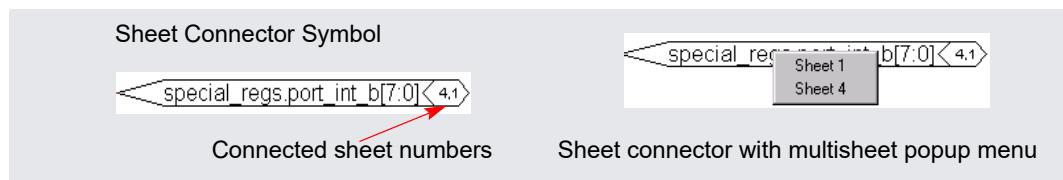
In a multi-sheet schematic, nets that span multiple sheets are indicated by sheet connector symbols, which you can use for navigation.

1. To reduce the number of sheets in a schematic, right-click and select HDL Analyst Options and increase the values set for Sheet Size Options - Instances and Sheet Size Options - Filtered Instances. To display fewer objects per sheet (increase the number of sheets), increase the values.

These options set a limit on the number of objects displayed on an unfiltered and filtered schematic sheet, respectively. A low Filtered Instances value can cause lower-level logic inside a transparent instance to be displayed on a separate sheet. The sheet numbers are indicated inside the empty transparent instance.

2. To navigate through a multi-sheet schematic, refer to this table. It summarizes common operations and ways to navigate.

| To view ...                                                    | Use one of these methods ...                                                                                                                                                                                                                                                                                         |
|----------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Next sheet or previous sheet                                   | Press the right mouse button and draw a horizontal mouse stroke (left to right for next sheet, right to left for previous sheet).<br>Click the icons: Next Sheet (↗) or Previous Sheet (↖)<br>Press Shift-right arrow (Next Sheet) or Shift-left arrow (Previous sheet).                                             |
| Lower-level logic of a transparent instance on separate sheets | Check the sheet numbers indicated inside the empty transparent instance. Use the sheet navigation commands like Next Sheet or View Sheets to move to the sheet you need.                                                                                                                                             |
| All objects of a certain type                                  | To highlight all the objects of the same type in the schematic, right-click and select the appropriate command from the Select All Schematic popup menu.<br>To highlight all the objects of the same type on the current sheet, right-click and select the appropriate command from the Select All Sheet popup menu. |
| Selected items only                                            | Filter the schematic as described in <i>Filtering Schematics</i> , on page 194.                                                                                                                                                                                                                                      |
| A net across sheets                                            | If there are no sheet numbers displayed in a hexagon at the end of the sheet connector, select HDL Analyst Options->General tab and enable Show Sheet Connector Index. Right-click the sheet connector and select the sheet number from the popup as shown in the following figure.                                  |

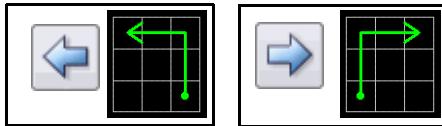


## Moving Between Views in a Schematic Window

When you filter or expand your design, you move through a number of different design views in the same schematic window. For example, you might start with a view of the entire design, zoom in on an area, then filter an object, and finally expand a connection in the filtered view, for a total of four views.

1. To move back to the previous view, click the Back icon or draw the appropriate mouse stroke.

The software displays the last view, including the zoom factor. This does not work in a newly generated view (for example, after flattening) because there is no history.



2. To move forward again, click the Forward icon or draw the appropriate mouse stroke.

The software displays the next view in the display history.

## Setting Schematic Preferences

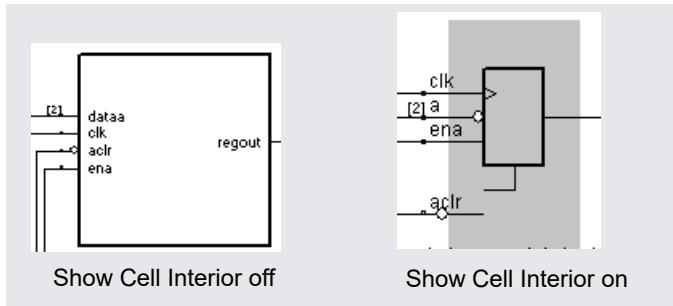
You can set various preferences for the schematics from the user interface.

1. Select HDL Analyst Options.
2. The following table details some common operations:

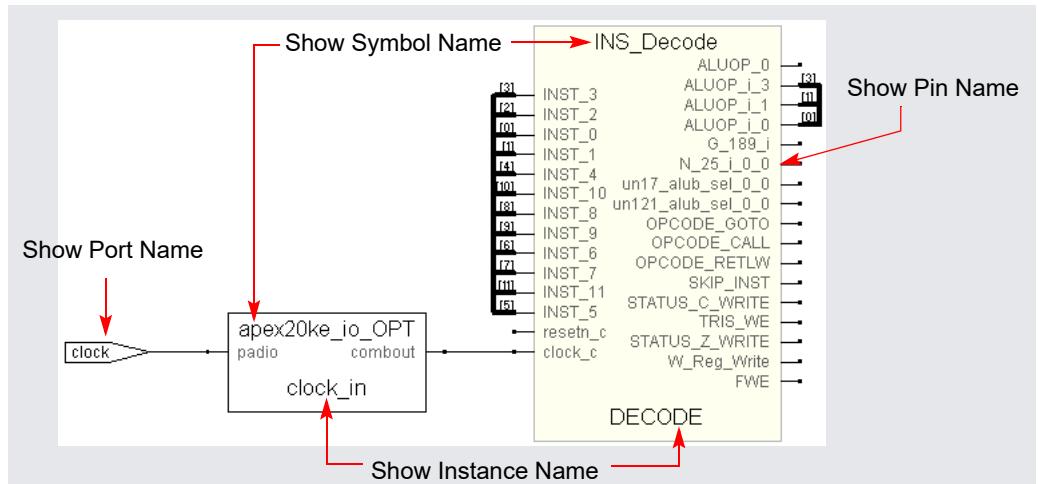
| To ...                                                                   | Do this ...                                                                                                                                                                                              |
|--------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Display the Hierarchy Browser                                            | Enable Show Hierarchy Browser (General tab).                                                                                                                                                             |
| Determine the number of objects displayed on a sheet.                    | Set the value with Maximum Instances on the Sheet Size tab. Increase the value to display more objects per sheet.                                                                                        |
| Determine the number of objects displayed on a sheet in a filtered view. | Set the value with Maximum Filtered Instances on the Sheet Size tab. Increase the number to display more objects per sheet. You cannot set this option to a value less than the Maximum Instances value. |

Some of these options do not take effect in the current view, but are visible in the next schematic you open.

3. To view hierarchy within a cell, enable the General->Show Cell Interiors option.



4. To control the display of labels, first enable the Text->Show Text option, and then enable the Label Options you want. The following figure illustrates the label that each option controls.



5. Click OK on the HDL Analyst Options form.

The software writes the preferences you set to the `ini` file, and they remain in effect until you change them.

# Exploring Design Hierarchy (Standard)

Schematics generally have a certain amount of design hierarchy. You can move between hierarchical levels using the Hierarchy Browser or Push/Pop command. For additional information, see [Analyzing With the Standard HDL Analyst Tool, on page 190](#). The topics include:

- [Traversing Design Hierarchy with the Hierarchy Browser, on page 166](#)
- [Exploring Object Hierarchy by Pushing/Popping, on page 167](#)
- [Exploring Object Hierarchy of Transparent Instances, on page 171](#)

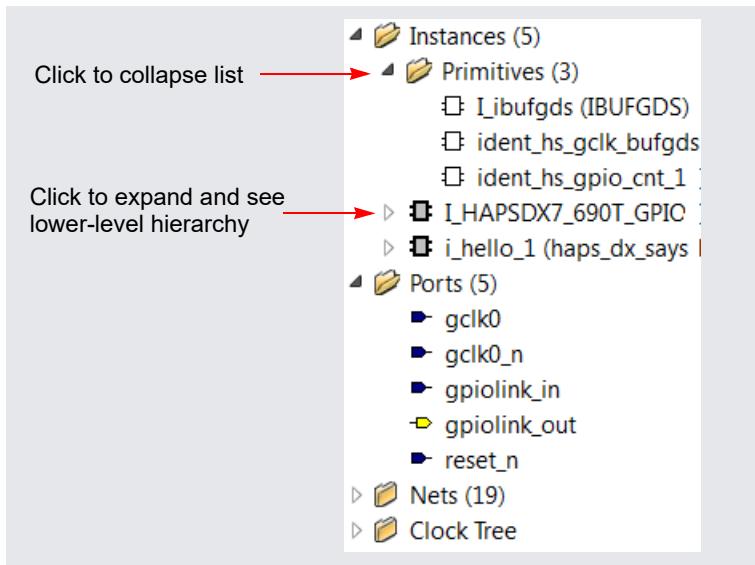
## Traversing Design Hierarchy with the Hierarchy Browser

The Hierarchy Browser is the list of objects on the left side of the schematic. It is best used to get an overview, or when you need to browse and find an object. If you want to move between design levels of a particular object, the Push/Pop command is more direct. Refer to [Exploring Object Hierarchy by Pushing/Popping, on page 167](#) for details.

The hierarchy browser allows you to traverse and select the following:

- Instances and submodules
- Ports
- Internal nets

Clock trees The browser lists the objects by type. Use the expand ( ▲ ) and collapse ( ▾ ) signs to ascend or descend the hierarchy.



## Exploring Object Hierarchy by Pushing/Popping

To view the internal hierarchy of a specific object, it is best to use the Push/Pop command or examine transparent instances, instead of using the Hierarchy Browser described in [Traversing Design Hierarchy with the Hierarchy Browser, on page 166](#). You can access Push/Pop command with the Push/Pop Hierarchy command or mouse strokes.

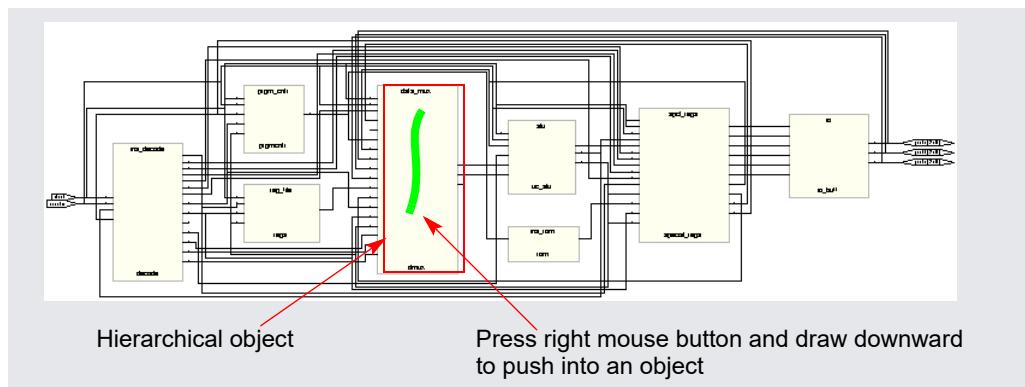
When combined with other commands like filtering and expansion commands, Push/Pop mode can be a very powerful tool for isolating and analyzing logic. See [Filtering Schematics, on page 194](#), [Expanding Pin and Net Logic, on page 196](#), and [Expanding and Viewing Connections, on page 199](#) for details about filtering and expansion. See the following sections for information about pushing down and popping up in hierarchical design objects:

- [Pushing into Objects, on page 168](#)
- [Popping up a Hierarchical Level, on page 170](#)

## Pushing into Objects

In the schematics, you can push into objects and view the lower-level hierarchy. You can use a mouse stroke, the command, or the icon to push into objects:

1. To move down a level (push into an object) with a mouse stroke, put your cursor near the top of the object, hold down the right mouse button, and draw a vertical stroke from top to bottom. You can push into the following objects; see step 3 for examples of pushing into different types of objects.
  - Hierarchical instances. They can be displayed as pale yellow boxes (opaque instances) or hollow boxes with internal logic displayed (transparent instances). You cannot push into a hierarchical instance that is hidden with the Hide Instance command (internal logic is hidden).



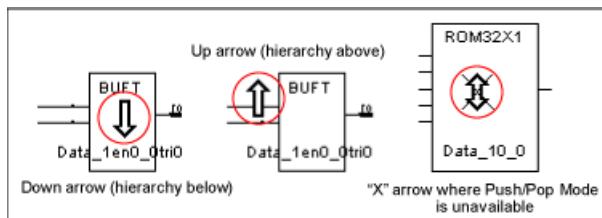
- Technology-specific primitives. The primitives are listed in the Hierarchy Browser in the schematic, under Instances - Primitives.
- Inferred ROMs and state machines.

The remaining steps show you how to use the icon or command to push into an object.

2. Enable Push/Pop mode by doing one of the following:

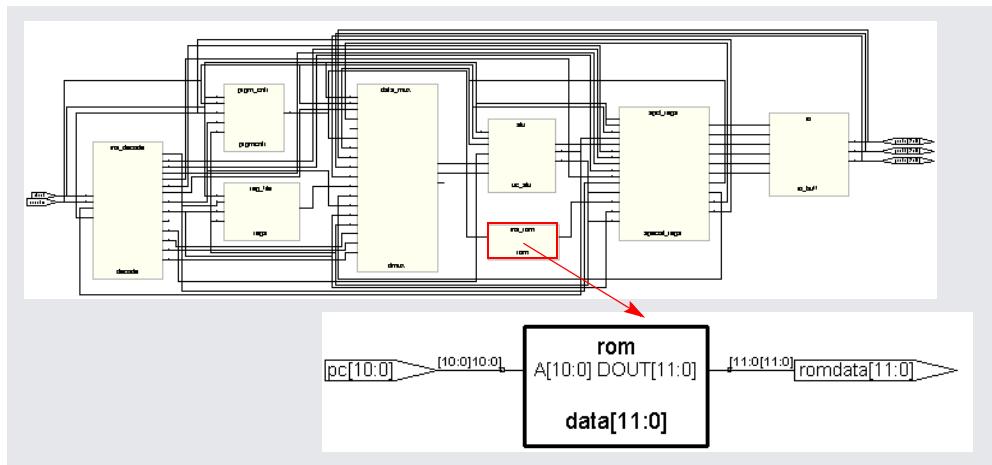
- Right-click in the view and select Push/Pop Hierarchy from the popup menu.
- Click the Push/Pop Hierarchy icon ( ) in the toolbar (two arrows pointing up and down).

The cursor changes to an arrow. The direction of the arrow indicates the underlying hierarchy, as shown in the following figure. The status bar at the bottom of the window reports information about the objects over which you move your cursor.



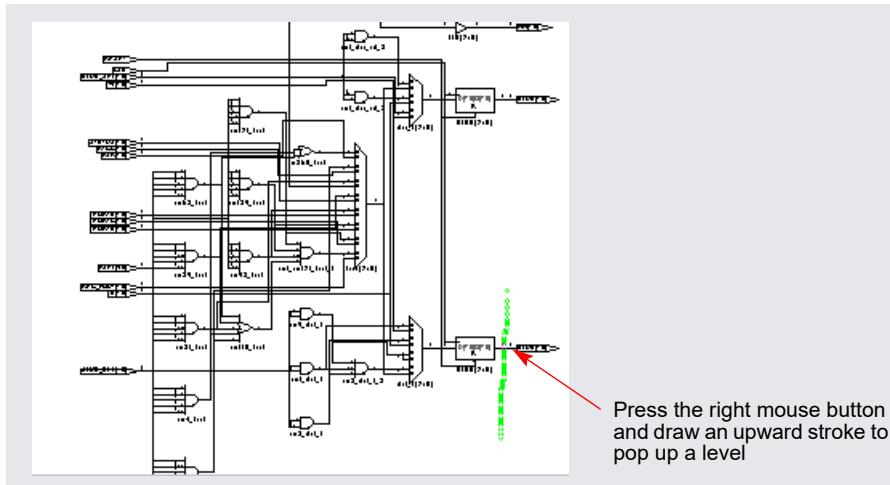
3. To push (descend) into an object, click on the hierarchical object. For a transparent instance, you must click on the pale yellow border. The following figure shows the result of pushing into a ROM.

When you descend into a ROM, you can push into it one more time to see the ROM data table. The information is in a view-only text file called rom.info.



## Popping up a Hierarchical Level

1. To move up a level (pop up a level), put your cursor anywhere in the design, hold down the right mouse button, and draw a vertical mouse stroke, moving from the bottom upwards.



The software moves up a level, and displays the next level of hierarchy.

2. To pop (ascend) a level using the commands or icon, do the following:
  - Select the command or icon if you are not already in Push/Pop mode. See [Pushing into Objects, on page 168](#)for details.
  - Move your cursor to a blank area and click.
3. To exit Push/Pop mode, do one of the following:
  - Click the right mouse button in a blank area of the view.
  - Deselect the Push/Pop Hierarchy icon.

## Exploring Object Hierarchy of Transparent Instances

Examining a transparent instance is one way of exploring the design hierarchy of an object. The following table compares this method with pushing (described in [Exploring Object Hierarchy by Pushing/Popping, on page 167](#)).

|                | <b>Pushing</b>                                                  | <b>Transparent Instance</b>                                                                                                      |
|----------------|-----------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| User control   | You initiate the operation through the command or icon.         | You have no direct control; the transparent instance is automatically generated by some commands that result in a filtered view. |
| Design context | Context lost; the lower-level logic is shown in a separate view | Context maintained; lower-level logic is displayed inside a hollow yellow box at the hierarchical level of the parent.           |

# Finding Objects (Standard)

In the schematics, you can use the Hierarchy Browser or the Find command to find objects, as explained in these sections:

- [Browsing to Find Objects in HDL Analyst Views, on page 172](#)
- [Using Find for Hierarchical and Restricted Searches, on page 174](#)
- [Using Wildcards with the Find Command, on page 177](#)
- [Using Find to Search the Output Netlist, on page 181](#)

## Browsing to Find Objects in HDL Analyst Views

You can always zoom in to find an object in the schematic. The following procedure shows you how to browse through design objects and find an object at any level of the design hierarchy. You can use the Hierarchy Browser or the Find command to do this. If you are familiar with the design hierarchy, the Hierarchy Browser can be the quickest method to locate an object. The Find command is best used to graphically browse and locate the object you want.

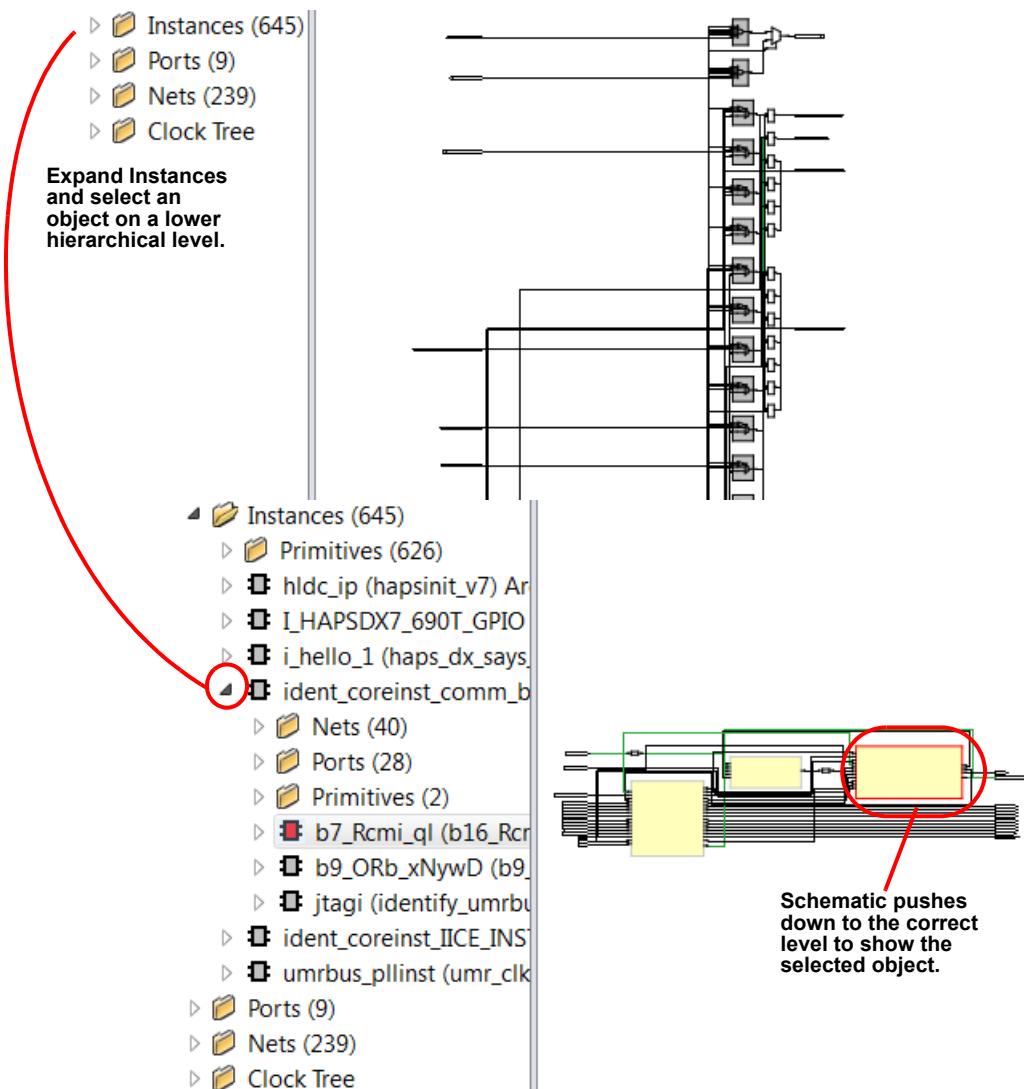
### Browsing With the Hierarchy Browser

1. In the Hierarchy Browser, click the name of the net, port, or instance you want to select.

The object is highlighted in the schematic.
2. To select a range of objects, select the first object in the range. Then, scroll to display the last object in the range. Press and hold the Shift key while clicking the last object in the range.

The software selects and highlights all the objects in the range.
3. If the object is on a lower hierarchical level, do either of the following:
  - Expand the appropriate higher-level object by clicking the collapsed symbol next to it, and then select the object you want.
  - Push down into the higher-level object, and then select the object from the Hierarchy Browser.

The selected object is highlighted in the schematic. The following example shows how moving down the object hierarchy and selecting an object causes the schematic to move to the sheet and level that contains the selected object.



4. To select all objects of the same type, select them from the Hierarchy Browser. For example, you can find all the nets in your design.

## Browsing With the Find Command

1. In a schematic, right-click and select Find or press Ctrl-f to open the Object Query dialog box.
2. Do the following in the dialog box:
  - Select objects in the selection box on the left. You can select all the objects or a smaller set of objects to browse. If length makes it hard to read a name, click the name in the list to cause the software to display the entire name in the field at the bottom of the dialog box.
  - Click the arrow to move the selected objects over to the box on the right.

The software highlights the selected objects.

3. In the Object Query dialog box, click on an object in the box on the right.

The software tracks to the schematic page with that object.

## Using Find for Hierarchical and Restricted Searches

You can always zoom in to find an object in the schematic, or use the Hierarchy Browser (see [Browsing to Find Objects in HDL Analyst Views, on page 172](#)). This procedure shows you how to use the Find command to do hierarchical object searches or restrict the search to the current level or the current level and its underlying hierarchy.

Note that Find only adds to the current selection; it does not deselect anything that is already selected. You can use successive searches to build up exactly the selection you need, before filtering.

1. If needed, restrict the range of the search by filtering the view.

See [Viewing Design Hierarchy and Context, on page 190](#) and [Filtering Schematics, on page 194](#) for details. With a filtered view, the software only searches the filtered instances, unless you set the scope of the search to Entire Design, as described below, in which case Find searches the entire design.

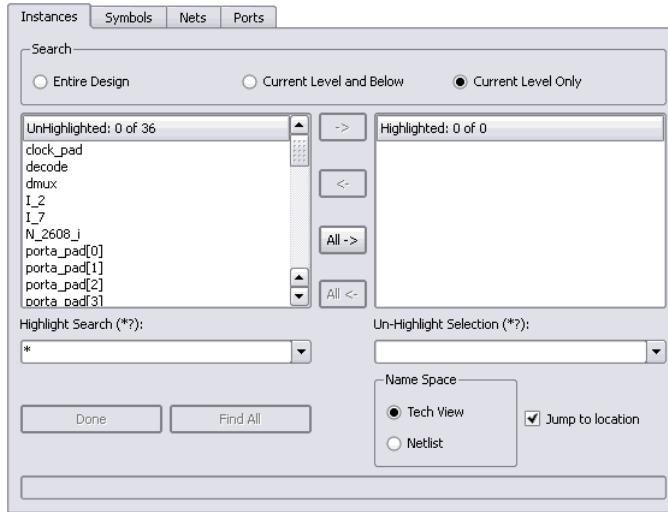
You can use the filtering technique to restrict your search to just one schematic sheet. Select all the objects on one sheet and filter the view. Continue with the procedure.

2. To further restrict the range of the search, hide instances you do not need.

You can do this in addition to filtering the view, or instead of filtering the view. Hidden instances and their hierarchy are excluded from the search. When you have finished the search, use the Unhide Instances command to make the hierarchy visible again.

3. Open the Object Query dialog box.

- Do one of the following: Right click in the view and select Find from the popup menu or press Ctrl-f.
- Reposition the dialog box so you can see both your schematic and the dialog box.



4. Select the tab for the type of object. The Unhighlighted box on the left lists all objects of that type (instances, symbols, nets, or ports).

For fastest results, search by Instances rather than Nets. When you select Nets, the software loads the whole design, which could take some time.

5. Click one of these buttons to set the hierarchical range for the search: Entire Design, Current Level & Below, or Current Level Only, depending on the hierarchical level of the design to which you want to restrict your search.

The range setting is especially important when you use wildcards. See [Effect of Hierarchy and Range on Wildcard Searches, on page 177](#) for

details. Current Level Only or Current Level & Below are useful for searching filtered schematics or critical path schematics.

The lower-level details of a transparent instance appear at the current level and are included in the search when you set it to Current Level Only. To exclude them, temporarily hide the transparent instances, as described in step 2.

Use Entire Design to hierarchically search the whole design. For large hierarchical designs, reduce the scope of the search by using the techniques described in the first step.

The Unhighlighted box shows available objects within the scope you set. Objects are listed in alphabetical order, not hierarchical order.

6. To search for objects in the mapped database or the output netlist, set the Name Space option.

The name of an object might be changed because of synthesis optimizations or to match the place-and-route tool conventions, so that the object name may no longer match the name in the original netlist. Setting the Name Space option ensures that the Find command searches the correct database for the object. For example, if you set this option to Tech View, the tool searches the mapped database (`srm`) for the object name you specify. For information about using this feature to find objects from an output netlist, see [Using Find to Search the Output Netlist, on page 181](#).

7. Do the following to select objects from the list. To use wildcards in the selection, see the next step.

- Click on the objects you want from the list. If length makes it hard to read a name, click the name in the list to cause the software to display the entire name in the field at the bottom of the dialog box.
- Click Find 200 or Find All. The former finds the first 200 matches, and then you can click the button again to find the next 200.
- Click the right arrow to move the objects into the box on the right, or double-click individual names.

The schematic displays highlighted objects in red.

8. Do the following to select objects using patterns or wildcards.

- Type a pattern in the Highlight Wildcard field. See [Using Wildcards with the Find Command, on page 177](#) for a detailed discussion of wildcards.

The Unhighlighted list shows the objects that match the wildcard criteria. If length makes it hard to read a name, click the name in the list to cause the software to display the entire name in the field at the bottom of the form.

- Click the right arrow to move the selections to the box on the right, or double-click individual names. The schematic displays highlighted objects in red.

You can use wildcards to avoid typing long path names. Start with a general pattern, and then make it more specific. The following example browses and uses wildcards successively to narrow the search.

|                                                                                          |            |
|------------------------------------------------------------------------------------------|------------|
| Find all instances three levels down                                                     | *.*.*      |
| Narrow search to find instances that begin with i_                                       | i_*.*.*    |
| Narrow search to find instances that begin with un2 after the second hierarchy separator | i_*.*.un2* |

9. You can leave the dialog box open to do successive Find operations. Close the dialog box when you are done.

## Using Wildcards with the Find Command

Use the following wildcards when you search the schematics:

- \* The asterisk matches any sequence of characters.
- ? The question mark matches any single character.
- . The dot explicitly matches a hierarchy separator, so type one dot for each level of hierarchy. To use the dot as a pattern and not a hierarchy separator, type a backslash before the dot: \.

## Effect of Hierarchy and Range on Wildcard Searches

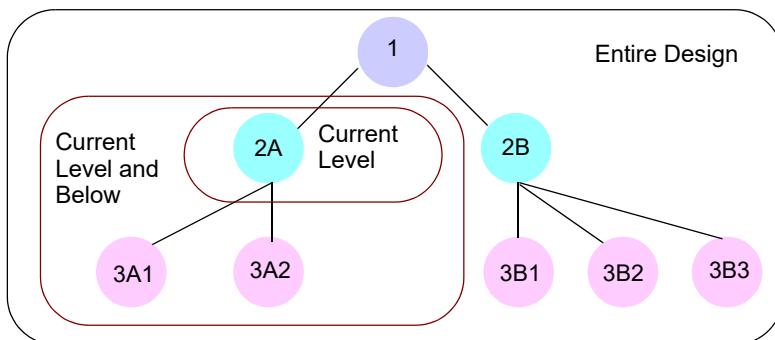
The asterisk and question mark wildcards do not cross hierarchical boundaries, but search each level of hierarchy individually with the search pattern. This default is affected by the following:

- Hierarchical separators

Dots match hierarchy separators, unless you use the backslash escape character in front of the dot (\.). Hierarchical search patterns with a dot (l\*.\* ) are repeated at each level included in the scope. If you use the \*.\* pattern with Current Level, the software matches non-hierarchical names at the current level that include a dot.

- Search range

The scope of the search determines the starting point for the searches. Some times the starting point might make it appear as if the wildcards cross hierarchical boundaries. If you are at 2A in the following figure and the scope of the search is set to Current Level and Below, separate searches start at 2A, 3A1, and 3A2. Each search does not cross hierarchical boundaries. If the scope of the search is Entire Design, the wildcard searches run from each hierarchical point (1, 2A, 2B, 3A1, 3A2, 3B1, 3B2, and 3B3). The result of an asterisk search (\*) with Entire Design is a list of all matches in the design, regardless of the current level.



See [Wildcard Search Examples](#), on page 179 for examples.

## How a Wildcard Search Works

1. The starting point of a wildcard search depends on the range set for the search.

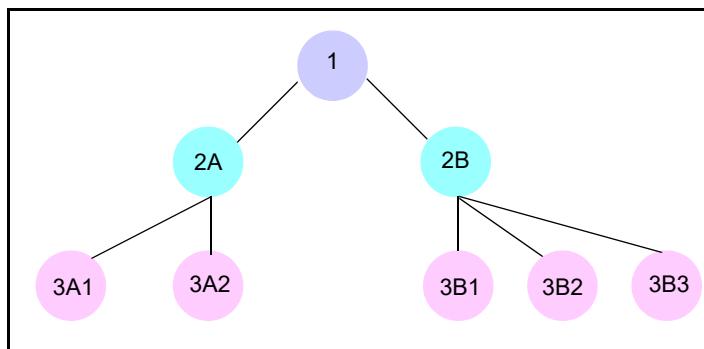
|                         |                                                                                                                                                                                                                                                                    |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Entire Design           | Starts at top level and uses the pattern to search from that level. It then moves to any child levels below the top level and searches them. The software repeats the search pattern at each hierarchical point in the design until it searches the entire design. |
| Current Level           | Starts at the current hierarchical level and searches that level only. A search started at 2A only covers 2A.                                                                                                                                                      |
| Current Level and Below | Starts at the current hierarchical level and searches that level. It then moves to any child levels below the starting point and conducts separate searches from each of these starting points.                                                                    |

2. The software applies the wildcard pattern to all applicable objects within the range. For Current Level and Current Level and Below, the current level determines the starting point.

Dots match hierarchy separators, unless you use the backslash escape character in front of the dot (\.). Hierarchical search patterns with a dot (1\*.\* ) are repeated at each level included in the scope. See [Effect of Hierarchy and Range on Wildcard Searches, on page 177](#) and [Wildcard Search Examples, on page 179](#) for details and examples, respectively. If you use the \*./\* pattern with Current Level, the software matches non-hierarchical names at the current level that include a dot.

## Wildcard Search Examples

The figure shows a design with three hierarchical levels, and the table shows the results of some searches on this design.



| Scope                   | Pattern | Starting Point | Finds Matches in ...                                                                                                                                                              |
|-------------------------|---------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Entire Design           | *       | 3A1            | 1, 2A, 2B, 3A1, 3A2, 3B1, 3B2, and 3B3 (* at all levels)                                                                                                                          |
|                         | *.*     | 2B             | 2A and 2B (*.*) from 1)<br>3A1, 3A2, 3B1, 3B2, and 3B3 (*.*) from 2A and 2B)<br>No matches in 1 (because of the hierarchical dot), unless a name includes a non-hierarchical dot. |
| Current Level           | *       | 1              | 1 only (no hierarchical boundary crossing)                                                                                                                                        |
|                         | *.*     | 2B             | 2B only. No search of lower levels even though the dot is specified, because the scope is Current Level. No matches, unless a 2B name includes a non-hierarchical dot.            |
| Current Level and Below | *       | 2A             | 2A only (no hierarchical boundary crossing)                                                                                                                                       |
|                         | *.*     | 1              | 2A and 2B (*.*) from 1)<br>3A1, 3A2, 3B1, 3B2, and 3B3 (*.*) from 2A and 2B)<br>No matches from 1, because the dot is specified.                                                  |
|                         | *.*     | 2B             | 3B1, 3B2, and 3B3 (*.*) from 2B)                                                                                                                                                  |
|                         | *.*     | 3A2            | No matches (no hierarchy below 3A2)                                                                                                                                               |
|                         | *.*.*   | 1              | 3A1, 3A2, 3B1, 3B2, and 3B3 (*.*) from 1)<br>Search ends because there is no hierarchy two levels below 2A and 2B.                                                                |

## Combining Find with Filtering to Refine Searches

You can combine the Find command with the filtering commands for a better effect. Depending on what you want to do, use the Find command first, or a filtering command.

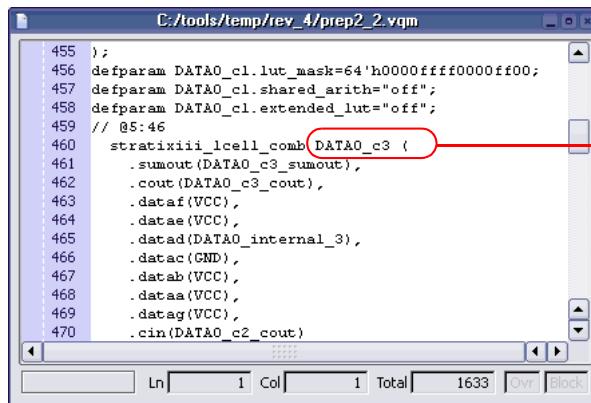
1. To limit the range of a search, do the following:
  - Filter the design.
  - Use the Find command on the filtered view, but set the search range to Current Level Only.

2. Select objects for a filtered view.
  - Use the Find command to browse and select objects.
  - Filter the objects to display them.

## Using Find to Search the Output Netlist

When you debug your design for place and route looking for a particular object, use the Name Space option in the Object Query dialog box to locate the optimized names in the output netlist. The following procedure shows you how to locate an object, highlight and filter it in the mapped view, and crossprobe to the source code for editing.

1. After you synthesize your design, open your output netlist file and select the name of the object you want to find.



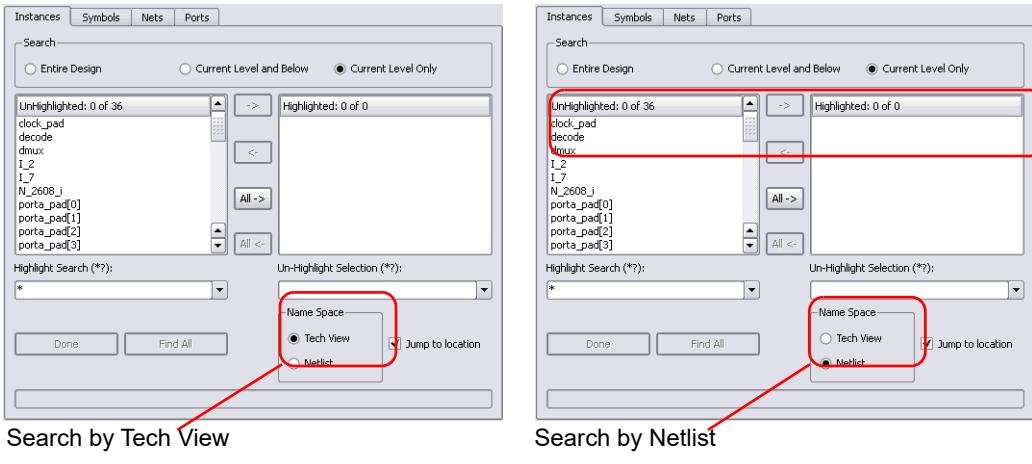
C:/tools/temp/rev\_4/prep2\_2.vqm

```
455 );
456 defparam DATA0_c1.lut_mask=64'h0000ffff0000ff00;
457 defparam DATA0_c1.shared_arith="off";
458 defparam DATA0_c1.extended_lut="off";
459 // @5:46
460     stratixiii_lcell_comb DATA0_c3 (
461         .sumout(DATA0_c3_sumout),
462         .cout(DATA0_c3_cout),
463         .dataf(VCC),
464         .datae(VCC),
465         .datad(DATA0_internal_3),
466         .datac(GND),
467         .datab(VCC),
468         .dataa(VCC),
469         .datag(VCC),
470         .cin(DATA0_c2_cout)
```

Ln 1 Col 1 Total 1633 Ovr Block

Copy Name

2. Copy the name and open the schematic for the mapped design.
3. In the view, select Find to open the Object Query dialog box and do the following:
  - Paste the object name you copied into the Highlight Search field.
  - Set the Name Space option to Netlist and click Find All.

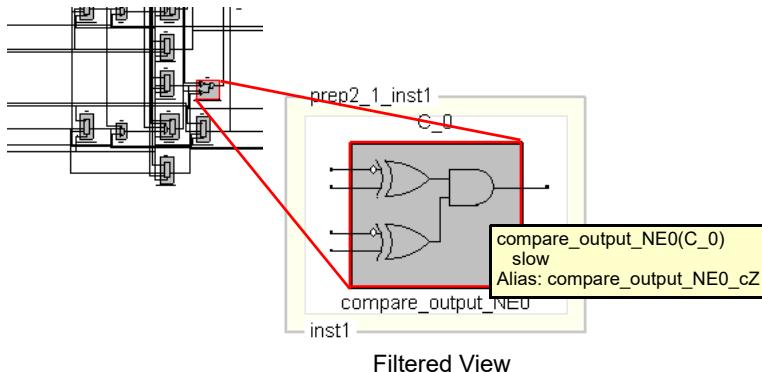


If you leave the Name Space option set to the default of Tech View, the tool does not find the name because it is searching the mapped database instead of the output netlist.

- Double click the name to move it into the Highlighted field and close the dialog box.

In the mapped view, the name is highlighted in the schematic.

4. Select Filter Schematic to view only the highlighted portion of the schematic.



The tooltip shows the equivalent name in the mapped view.

5. Double click on the filtered schematic to crossprobe to the corresponding code in the HDL file.

# Crossprobing (Standard)

Crossprobing is the process of selecting an object in one view and having the object or the corresponding logic automatically highlighted in other views. Highlighting a line of text, for example, highlights the corresponding logic in the schematics. Crossprobing helps you visualize where coding changes or timing constraints might help to reduce area or improve performance.

You can crossprobe between the schematics, the log file, the source files, and some external text files from place-and-route tools. However, not all objects or source code crossprobe to other views, because some source code and RTL logic is optimized away during subsequent compilation or mapping processes.

This section describes how to crossprobe from different views. It includes the following:

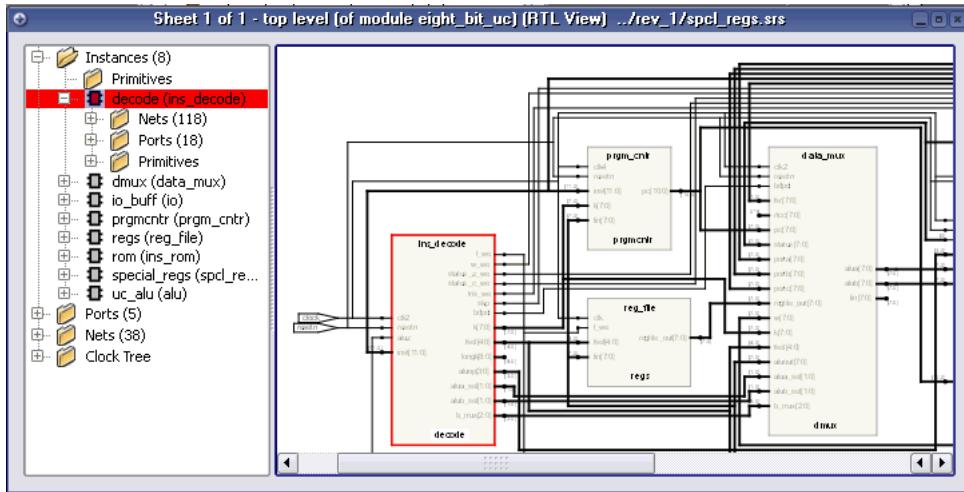
- [Crossprobing within a View, on page 183](#)
- [Crossprobing from an HDL Analyst View, on page 184](#)
- [Crossprobing from the Text Editor Window, on page 186](#)

## Crossprobing within a View

Selecting an object name in the Hierarchy Browser highlights the object in the schematic, and vice versa.

| Selected Object                      | Highlighted Object               |
|--------------------------------------|----------------------------------|
| Instance in schematic (single-click) | Module icon in Hierarchy Browser |
| Net in schematic                     | Net icon in Hierarchy Browser    |
| Port in schematic                    | Port icon in Hierarchy Browser   |
| Logic icon in Hierarchy Browser      | Instance in schematic            |
| Net icon in Hierarchy Browser        | Net in schematic                 |
| Port icon in Hierarchy Browser       | Port in schematic                |

In this example, when you select the DECODE module in the Hierarchy Browser, the DECODE module is automatically selected in the view.

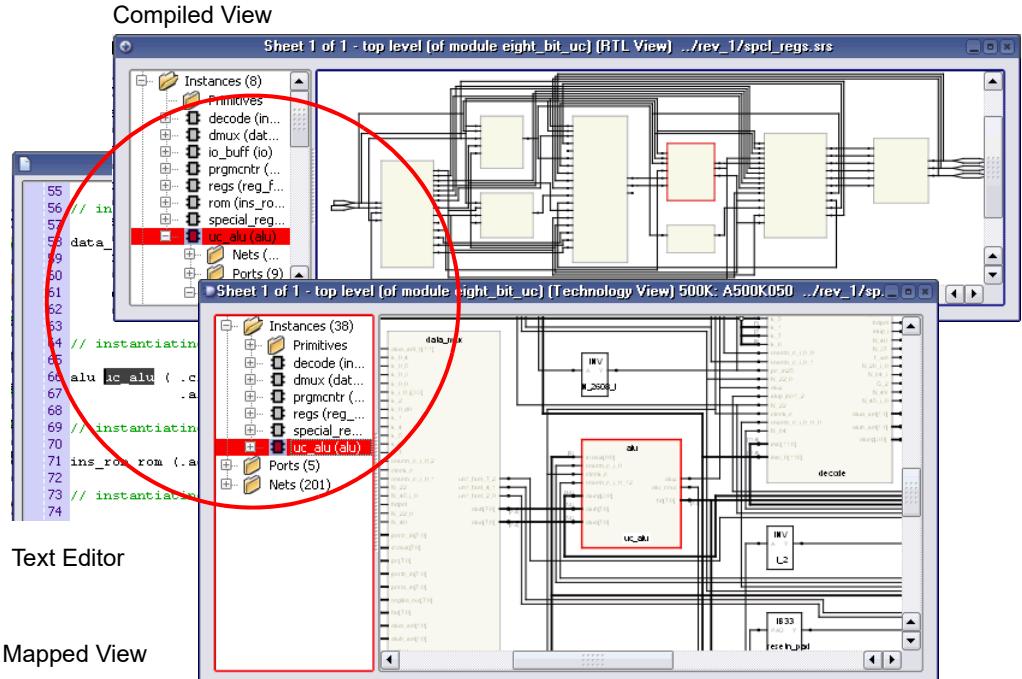


## Crossprobing from an HDL Analyst View

1. To crossprobe from an HDL Analyst view to other open views, select the object by clicking on it.

The software automatically highlights the object in all open views. If the open view is a schematic, the software highlights the object in the Hierarchy Browser on the left as well as in the schematic. If the highlighted object is on another sheet of a multi-sheet schematic, the view does not automatically track to the page. If the cross-probed object is inside a hidden instance, the hidden instance is highlighted in the schematic.

If the open view is a source file, the software tracks to the appropriate code and highlights it. The following figure shows crossprobing between HDL Analyst and Text Editor (source code) views.



2. To crossprobe from the HDL Analyst view to the source file when the source file is not open, double-click on the object in the view.

Double-clicking automatically opens the appropriate source code file and highlights the appropriate code. For example, if you double-click an object in a mapped view, the HDL Analyst tool automatically opens an editor window with the source code and highlights the code that contains the selected register.

The following table summarizes the crossprobing capability from the HDL Analyst view.

| From        | To          | Procedure                                                                                                                                                                                                                                                 |
|-------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HDL Analyst | Source code | Double-click an object. If the source code file is not open, the software opens the Text Editor window to the appropriate section of code. If the source file is already open, the software scrolls to the correct section of the code and highlights it. |
| HDL Analyst | HDL Analyst | Both views must be open. Click the object to highlight and crossprobe.                                                                                                                                                                                    |

## Crossprobing from the Text Editor Window

To crossprobe from a source code window or from the log file to an HDL Analyst or FSM view, use this procedure. You can use this method to crossprobe from any text file with objects that have the same instance names as in the synthesis software. For example, you can crossprobe from place-and-route files. See [Example of Crossprobing a Path from a Text File, on page 186](#) for a practical example of how to use crossprobing.

1. Open the HDL Analyst or FSM view to which you want to crossprobe.
2. To crossprobe from an error, warning, or note in the html log file, click on the file name to open the corresponding source code in another Text Editor window; to crossprobe from a text log file, double-click on the text of the error, warning, or note. Select the appropriate portion of text in the Text Editor window. In some cases, it may be necessary to select an entire block of text to crossprobe.

The software highlights the objects corresponding to the selected code in all the open windows. If an object is on another schematic sheet or on another hierarchical level, the highlighting might not be obvious. If you filter the schematic (right-click in the source code window with the selected text and select Filter Schematic from the popup menu), you can isolate the highlighted objects for easy viewing.

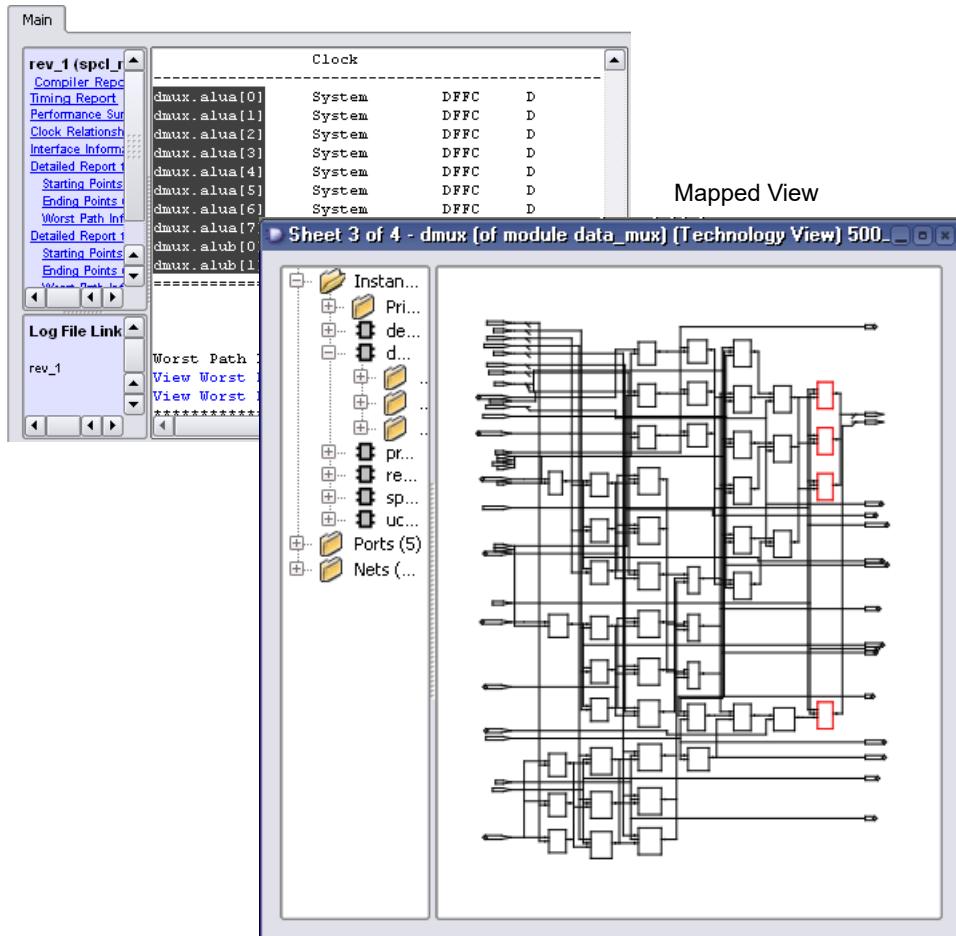
### Example of Crossprobing a Path from a Text File

This example selects a path in a log file and crossprobes it in the mapped view. You can use the same technique to crossprobe from other text files like place-and-route files, as long as the instance names in the text file match the instance names in the synthesis tool.

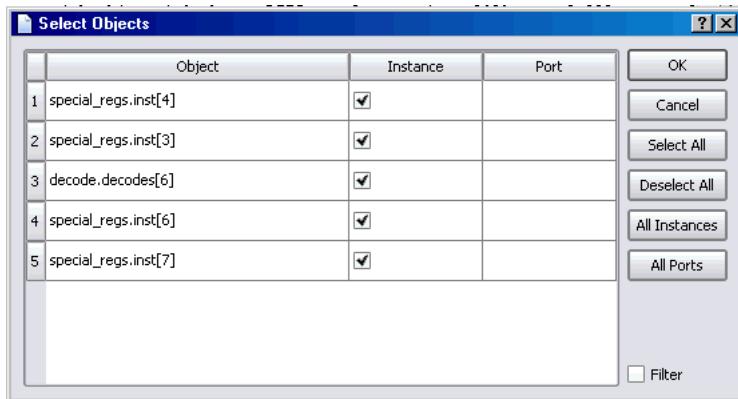
1. Open the log file and schematics.
2. Select the path objects in the log file.
  - Select the column by pressing Alt and dragging the cursor to the end of the column. On the Linux platform, use the key to which the Alt function is mapped; this is usually the Ctrl-Alt key combination.
  - To select all the objects in the path, right-click and choose Select in Analyst from the popup menu. Alternatively, you can select certain objects only, as described next.

The software selects the objects in the column, and highlights the path in the open schematics.

## Text Editor



- To further filter the objects in the path, right-click and choose Select From from the popup menu. On the form, check the objects you want, and click OK. Only the corresponding objects are highlighted.



3. To isolate and view only the selected objects, do this in the mapped view: right-click and select the Filter Schematic command from the popup menu. You see just the selected objects.

# Analyzing With the Standard HDL Analyst Tool

The HDL Analyst tool is a graphical productivity tool that helps you visualize your synthesis results. It displays schematics of the design at different stages, allowing you to graphically view and analyze your design. At an early design stage, the schematic displays high-level structures like RAMs, ROMs, operators, and FSMs as abstractions. Later in the cycle, these structures are converted to gates and mapped to technology-specific resources.

To analyze information or compare views with the log file, the FSM view, and the source code, you can use techniques like crossprobing, flattening, and filtering. See the following for more information about analysis techniques.

- [Viewing Design Hierarchy and Context, on page 190](#)
- [Filtering Schematics, on page 194](#)
- [Expanding Pin and Net Logic, on page 196](#)
- [Expanding and Viewing Connections, on page 199](#)
- [Flattening Schematic Hierarchy, on page 201](#)
- [Minimizing Memory Usage While Analyzing Designs, on page 205](#)

For additional information about navigating the HDL Analyst views or using other techniques like crossprobing, see the following:

- [Working in the Standard HDL Analyst Schematic, on page 157](#)
- [Exploring Design Hierarchy \(Standard\), on page 166](#)
- [Finding Objects \(Standard\), on page 172](#)
- [Crossprobing \(Standard\), on page 183](#)

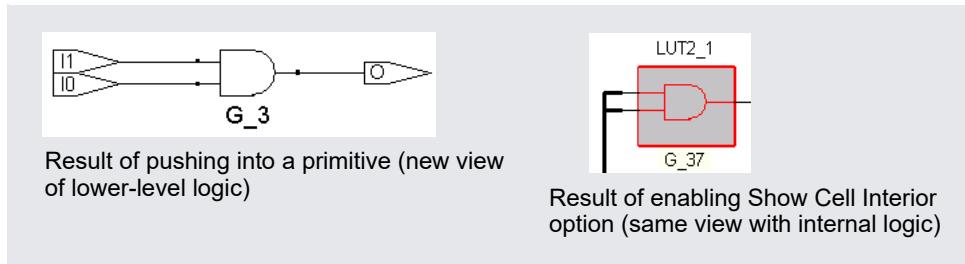
## **Viewing Design Hierarchy and Context**

Most large designs are hierarchical, so the synthesis software provides tools that help you view hierarchy details or put the details in context. Alternatively, you can browse and navigate hierarchy with Push/Pop mode, or flatten the design to view internal hierarchy.

This section describes how to use interactive hierarchical viewing operations to better analyze your design. Automatic hierarchy viewing operations that are built into other commands are described in the context in which they appear. For example, the software automatically traces a critical path through different hierarchical levels using hollow boxes with nested internal logic (transparent instances) to indicate levels in hierarchical instances.

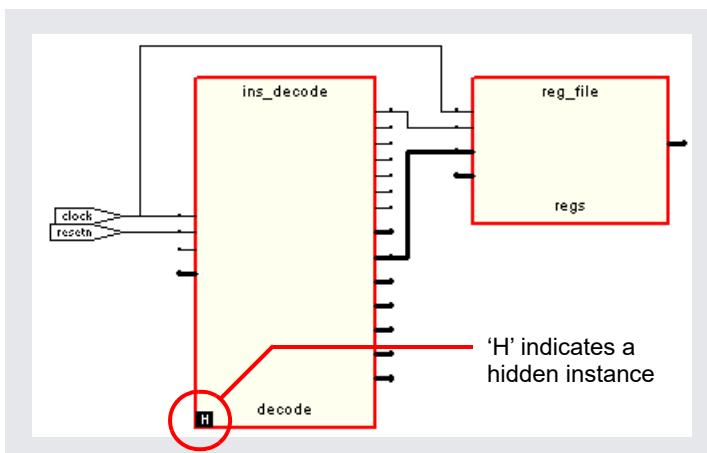
1. To view the internal logic of primitives in your design, do either of the following:
  - To view the logic of an individual primitive, push into it. This generates a new schematic with the internal details. Click the Back icon to return to the previous view.
  - To view the logic of all primitives in the design, select HDL Analyst Options->General, and enable Show Cell Interior. This command lets you see internal logic in context, by adding the internal details to the current schematic and all subsequent views. If the view is too cluttered with this option on, filter the view (see [Filtering Schematics, on page 194](#)) or push into the primitive. Click the Back icon to return to the previous view after filtering or pushing into the object.

The following figure compares these two methods:



2. To hide selected hierarchy, select the instance whose hierarchy you want to exclude, and then select Hide Instances from the HDL Analyst menu or the right-click popup menu in the schematic.

You can hide opaque (solid yellow) or transparent (hollow) instances. The software marks hidden instances with an H in the lower left. Hidden instances are like black boxes; their hierarchy is excluded from filtering, expanding, dissolving, or searching in the current window, although they can be cross-probed. An instance is only hidden in the current view window; other view windows are not affected. Temporarily hiding unnecessary hierarchy focuses analysis and saves time in large designs.



Before you save a design with hidden instances, select **Unhide Instances** from the HDL Analyst menu or the right-click popup menu and make the hidden internal hierarchy accessible again. Otherwise, the hidden instances are saved as black boxes, without their internal logic. Conversely, you can use this feature to reduce the scope of analysis in a large design by hiding instances you do not need, saving the reduced design to a new name, and then analyzing it.

3. To view the internal logic of a hierarchical instance, you can push into the instance, dissolve the selected instance with the **Dissolve Instances** command, or flatten the design. You cannot use these methods to view the internal logic of a hidden instance.

|                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Pushing into an instance             | Generates a view that shows only the internal logic. You do not see the internal hierarchy in context. To return to the previous view, click Back. See <a href="#">Exploring Object Hierarchy by Pushing/Popping</a> , on page 167 for details.                                                                                                                                                                                          |
| Flattening the entire design         | Opens a new view where the entire design is flattened, except for hidden hierarchy. Large flattened designs can be overwhelming. See <a href="#">Flattening Schematic Hierarchy</a> , on page 201 for details about flattening designs.<br><br>Because this is a new view, you cannot use Back to return to the previous view. To return to the top-level unflattened schematic, right-click in the view and select Unflatten Schematic. |
| Flattening an instance by dissolving | Generates a view where the hierarchy of the selected instances is flattened, but the rest of the design is unaffected. This provides context. See <a href="#">Flattening Schematic Hierarchy</a> , on page 201 for details about dissolving instances.                                                                                                                                                                                   |

4. If the result of filtering or dissolving is a hollow box with no internal logic, try either of the following, as appropriate, to view the internal hierarchy:
  - Right-click and select HDL Analyst Options->Sheet Size and increase the value of Maximum Filtered Instances. Use this option if the view is not too cluttered.
  - Use the sheet navigation commands to go to the sheets indicated in the hollow box.

If there is too much internal logic to display in the current view, the software puts the internal hierarchy on separate schematic sheets. It displays a hollow box with no internal logic and indicates the schematic sheets that contain the internal logic.

5. To view the design context of an instance in a filtered view, select the instance, right-click, and select Show Context from the popup menu.

The software displays an unfiltered view of the hierarchical level that contains the selected object, with the instance highlighted. This is useful when you have to go back and forth between different views during analysis. The context differs from the Expand commands, which show connections. To return to the original filtered view, click Back.

## Filtering Schematics

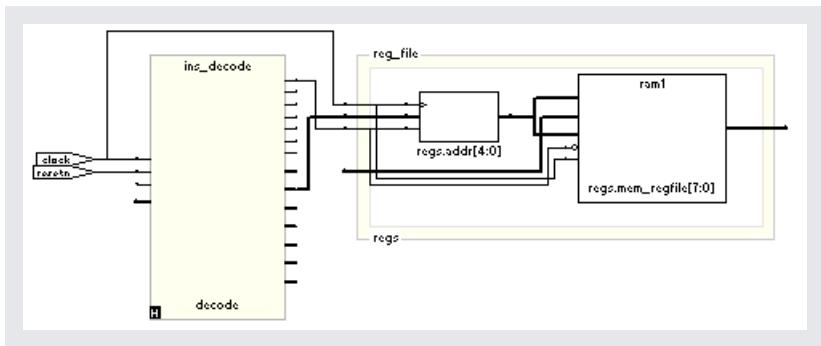
Filtering is a useful first step in analysis, because it focuses analysis on the relevant parts of the design. Some commands, like the Expand commands, automatically generate filtered views; this procedure only discusses manual filtering, where you use the Filter Schematic command to isolate selected objects.

This table lists the advantages of using filtering over flattening:

| Filter Schematic Command                                                                                                | Flatten Commands                                                                                                                                                                      |
|-------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Loads part of the design; better memory usage                                                                           | Loads entire design                                                                                                                                                                   |
| Combine filtering with Push/Pop mode, and history buttons (Back and Forward) to move freely between hierarchical levels | Must use Unflatten Schematic to return to top level, and flatten the design again to see lower levels. Cannot return to previous view if the previous view is not the top-level view. |

1. Select the objects that you want to isolate. For example, you can select two connected objects.

If you filter a hidden instance, the software does not display its internal hierarchy when you filter the design. The following example illustrates this.

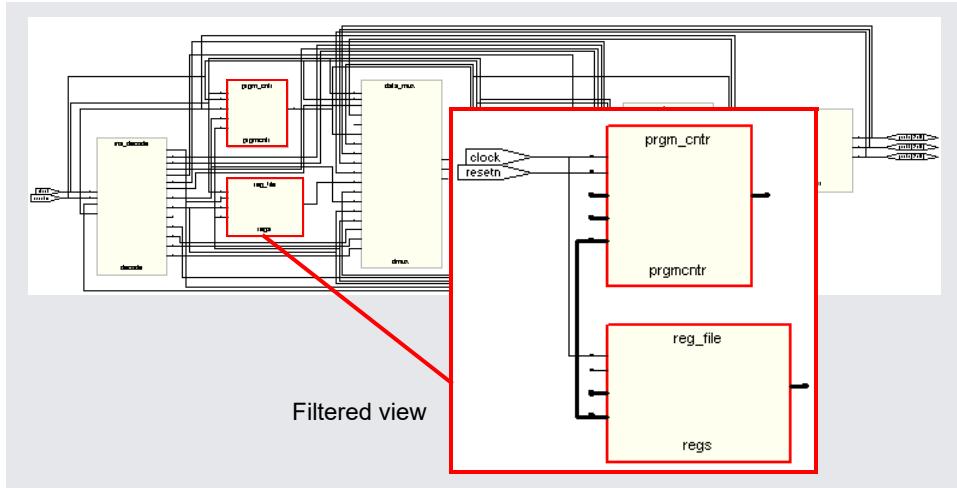


2. Select the Filter Schematic command, using one of these methods:

- Right-click and select Filter Schematic from the popup menu.
- Click the Filter Schematic icon (buffer gate) (

Alternatively, run the filter command from the Tcl window. For details on filtering the schematic, see the `analyst` command in the ProtoCompiler Command Reference Guide.

The software filters the design and displays the selected objects in a filtered view. The title bar indicates that it is a filtered view. Hidden instances have an H in the lower left. The view displays other hierarchical instances as hollow boxes with nested internal logic (transparent instances). If the transparent instance does not display internal logic, use one of the alternatives described in [Viewing Design Hierarchy and Context, on page 190](#), step 4.



3. If the filtered view does not display the pin names of technology primitives and transparent instances that you want to see, do the following:
  - Right-click and select HDL Analyst Options->Text and enable Show Pin Name.
  - To temporarily display a pin name, move the cursor over the pin. The name is displayed as long as the cursor remains over the pin. Alternatively, select a pin. The software displays the pin name until you make another selection. Either of these options can be applied to individual pins. Use them to view just the pin names you need and keep design clutter to a minimum.
  - To see all the hierarchical pins, select the instance, right-click, and select Show All Hier Pins.

You can now analyze the problem, and do operations like the following:

|                               |                                                                                                                                   |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Trace paths, build up logic   | See <a href="#">Expanding Pin and Net Logic</a> , on page 196 and <a href="#">Expanding and Viewing Connections</a> , on page 199 |
| Filter further                | Select objects and filter again                                                                                                   |
| Find objects                  | See <a href="#">Finding Objects (Standard)</a> , on page 172                                                                      |
| Flatten, or hide and flatten  | See <a href="#">Flattening Schematic Hierarchy</a> , on page 201. You can hide transparent or opaque instances.                   |
| Crossprobe from filtered view | See <a href="#">Crossprobing from an HDL Analyst View</a> , on page 184                                                           |

4. To return to the previous schematic, click the Back icon. If you flattened the hierarchy, right-click and select Unflatten Schematic to return to the top-level unflattened view.

For additional information about filtering schematics, see [Filtering Schematics](#), on page 194 and [Flattening Schematic Hierarchy](#), on page 201.

## Expanding Pin and Net Logic

When you are working in a filtered view, you might need to include more logic in your selected set to debug your design. This section describes commands that expand logic fanning out from pins or nets; to expand paths, see [Expanding and Viewing Connections](#), on page 199.

Use the Expand commands with the Filter Schematic, Hide Instances, and Flatten commands to isolate just the logic that you want to examine. Filtering isolates logic, flattening removes hierarchy, and hiding instances prevents their internal hierarchy from being expanded. See [Filtering Schematics](#), on page 194 and [Flattening Schematic Hierarchy](#), on page 201 for details.

1. To expand logic from a pin hierarchically across boundaries, use the following commands.

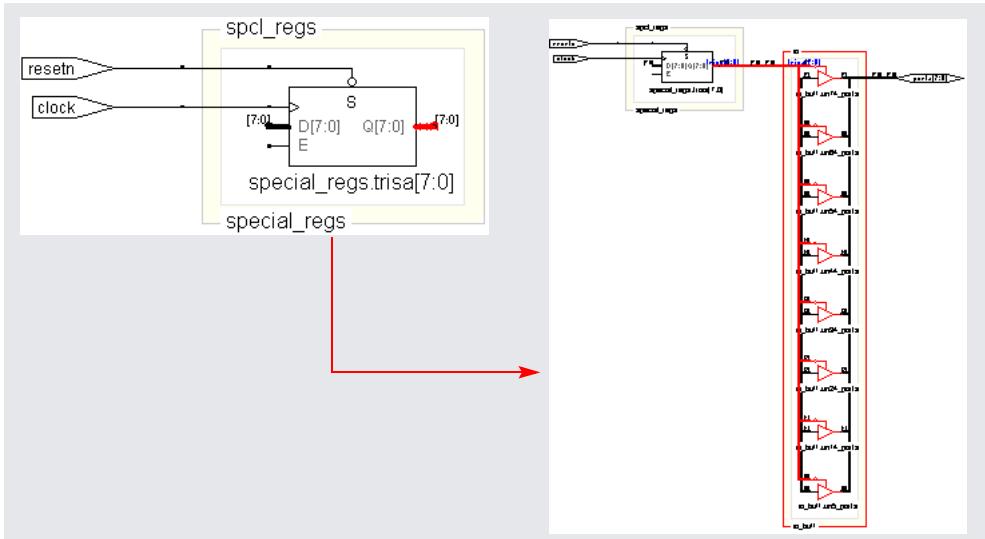
| To ...                                                             | Do this ...                                                                                                                                                                                   |
|--------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| See all cells connected to a pin                                   | Select a pin and select Expand. See <a href="#">Expanding Filtered Logic Example , on page 198</a> .                                                                                          |
| See all cells that are connected to a pin, up to the next register | Select a pin and select Expand to Register/Port. See <a href="#">Expanding Filtered Logic to Register/Port Example , on page 198</a> .                                                        |
| See internal cells connected to a pin                              | Select a pin and select Expand Inwards. The software filters the schematic and displays the internal cells closest to the port. See <a href="#">Expanding Inwards Example , on page 199</a> . |

The software expands the logic as specified, working on the current level and below or working up the hierarchy, crossing hierarchical boundaries as needed. Hierarchical levels are shown nested in hollow bounding boxes. The internal hierarchy of hidden instances is not displayed.

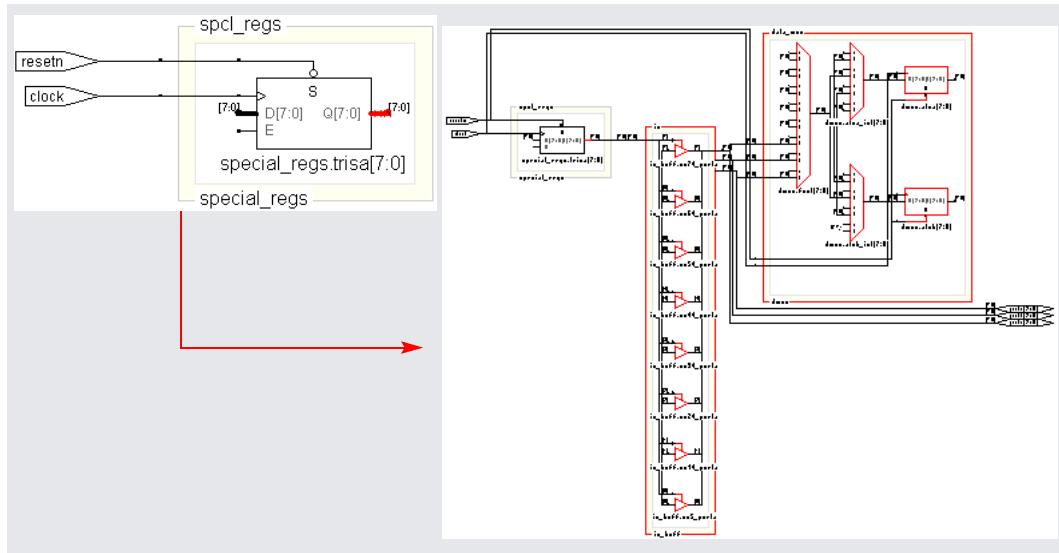
2. To expand logic from a pin at the current level only, do the following:
  - Select a pin, then right-click and select the popup menu Current Level.
  - Select Expand or Expand to Register/Ports. The commands work as described in the previous step, but they do not cross hierarchical boundaries.
3. To expand logic from a net at the current level only, do the following:
  - Use the commands shown in the following table.
  - Select a net, then right-click and select the command from the popup menu Current Level.

| To ...                                    | Do this ...                                                                                                                                                               |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Select the driver of a net                | Select a net and select Select Net Driver. The result is a filtered view with the net driver selected ( <a href="#">Selecting the Net Driver Example , on page 199</a> ). |
| Trace the driver, across sheets if needed | Select a net and select Go to Net Driver. The software shows a view that includes the net driver.                                                                         |
| Select all instances on a net             | Select a net and select Select Net Instances. You see a filtered view of all instances connected to the selected net.                                                     |

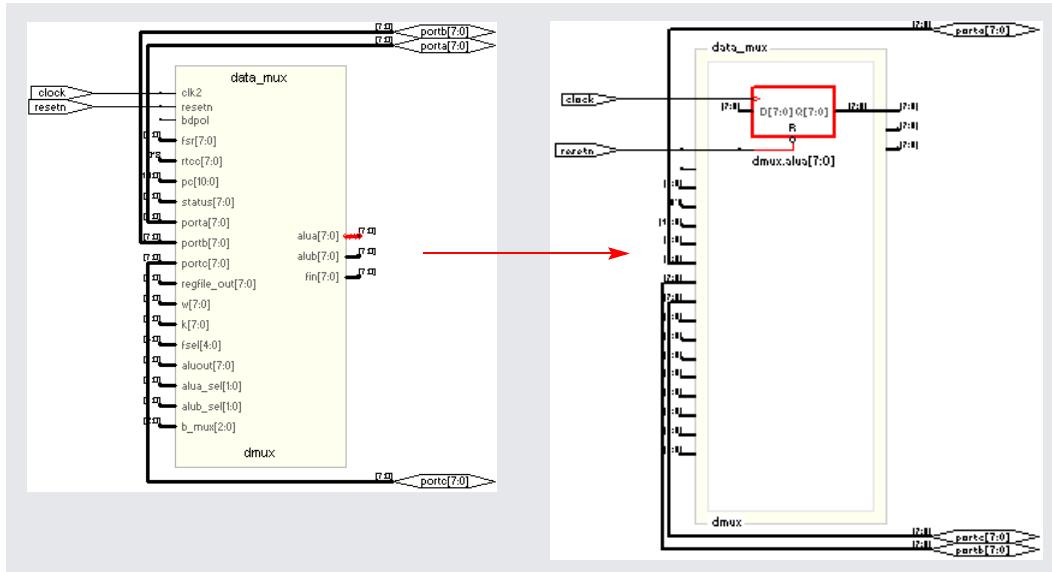
## Expanding Filtered Logic Example



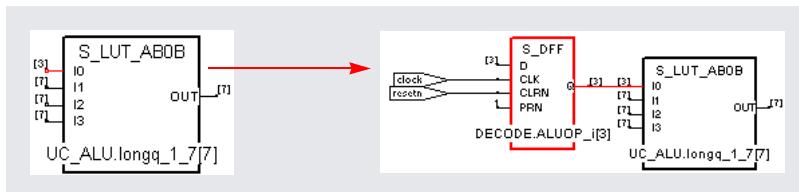
## Expanding Filtered Logic to Register/Port Example



## Expanding Inwards Example



## Selecting the Net Driver Example



## Expanding and Viewing Connections

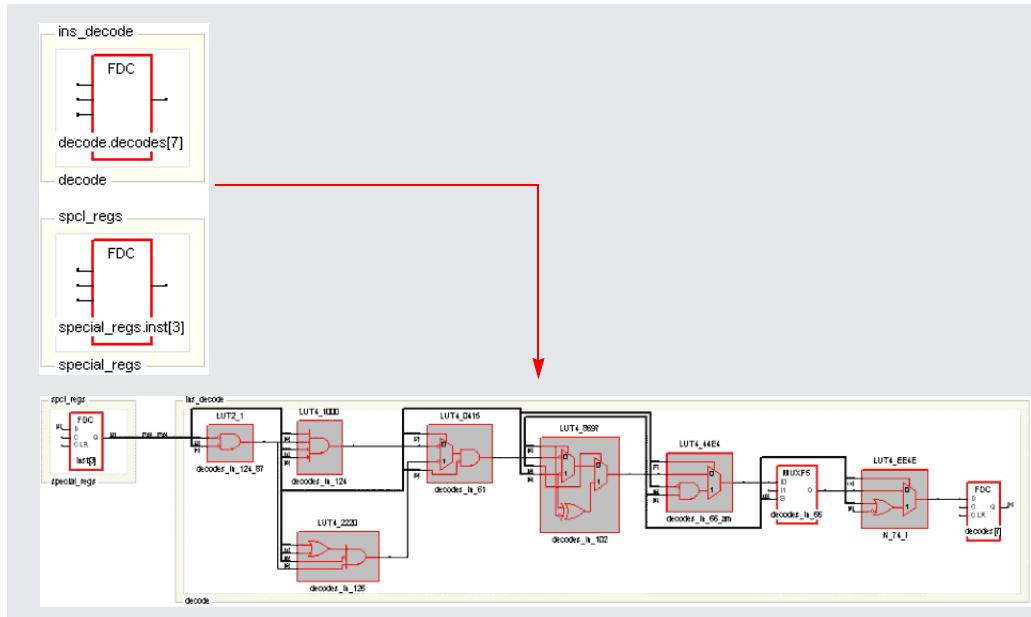
This section describes commands that expand logic between two or more objects; to expand logic out from a net or pin, see [Expanding Pin and Net Logic, on page 196](#).

Use the following path commands with the Filter Schematic and Hide Instances commands to isolate just the logic that you want to examine. The two techniques described here differ: Expand Paths expands connections between selected objects, while Isolate Paths pares down the current view to only display connections to and from the selected instance.

1. To expand and view connections between selected objects, do the following:

- Select two or more points.

To expand the logic at the current level only, right-click and select the popup menu Current Level Expand Paths.



2. To view connections from all pins of a selected instance, right-click and select Isolate Paths from the popup menu.

**Starting Point** The Filtered View Traces Paths (Forward and Back) From All Pins of the Selected Instance...

**Filtered view** Traces through all sheets of the filtered view, up to the next port, register, hierarchical instance, or black box.

**Unfiltered view** Traces paths on the current schematic sheet only, up to the next port, register, hierarchical instance, or black box.

Unlike the Expand Paths command, the connections are based on the schematic used as the starting point; the software does not add any objects that were not in the starting schematic.

## Flattening Schematic Hierarchy

Flattening removes hierarchy so you can view the logic without hierarchical levels. In most cases, you do not have to flatten your hierarchical schematic to debug and analyze your design, because you can use a combination of filtering, Push/Pop mode, and expanding to view logic at different levels. However, if you must flatten the design, use the following techniques., which include flattening, dissolving, and hiding instances.

1. To flatten an entire design down to logic cells, right-click and select Flattened View from the popup menu.

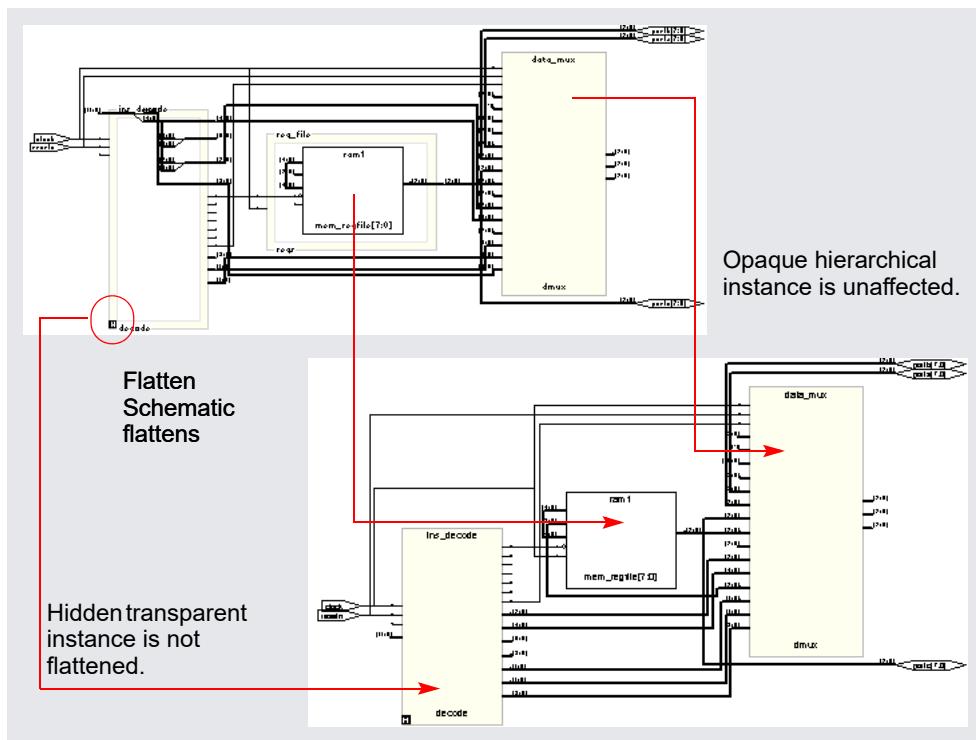
The software flattens the top-level design and displays it in a new window. To return to the top-level design, right-click and select Unflatten Schematic.

Unless you really require the entire design to be flattened, use Push/Pop mode and the filtering commands (*Filtering Schematics*, on page 194) to view the hierarchy. Alternatively, you can use one of the selective flattening techniques described in subsequent steps.

2. To selectively flatten transparent instances when you analyze critical paths or use the Expand commands, select Flatten Schematic from the right-click popup menu.

The software generates a new view of the current schematic in the same window, with all transparent instances at the current level and below flattened. To control the number of hierarchical levels that are flattened, use the Dissolve Instances command described in step 4.

If your view only contains hidden hierarchical instances or pale yellow (opaque) hierarchical instances, nothing is flattened. If you flatten an unfiltered (usually the top-level design) view, the software flattens all hierarchical instances (transparent and opaque) at the current level and below. The following figure shows flattened transparent instances.



Because the flattened view is a new view, you cannot use Back to return to the unflattened view or the views before it. Use Unflatten Schematic to return to the unflattened top-level view.

- To selectively flatten the design by hiding instances, select hierarchical instances whose hierarchy you do not want to flatten, right-click, and select Hide Instances. Then flatten the hierarchy using one of the Flatten commands described above.

Use this technique if you want to flatten most of your design. If you want to flatten only part of your design, use the approach described in the next step.

When you hide instances, the software generates a new view where the hidden instances are not flattened, but marked with an H in the lower left corner. The rest of the design is flattened. If unhidden hierarchical instances are not flattened by this procedure, use the Flattened View command described in step 1 instead of the Flatten Current Schematic

command described in step 2, which only flattens transparent instances in filtered views.

You can select the hidden instances, right-click, and select Unhide Instances to make their hierarchy accessible again. To return to the unflattened top-level view, right-click in the schematic and select Unflatten Schematic.

4. To selectively flatten some hierarchical instances in your design by dissolving them, do the following:

- If you want to flatten more than one level, right-click and select HDL Analyst Options->General and change the value of Dissolve Levels. If you want to flatten just one level, leave the default setting.
- Select the instances to be flattened.
- Right-click and select Dissolve Instances.

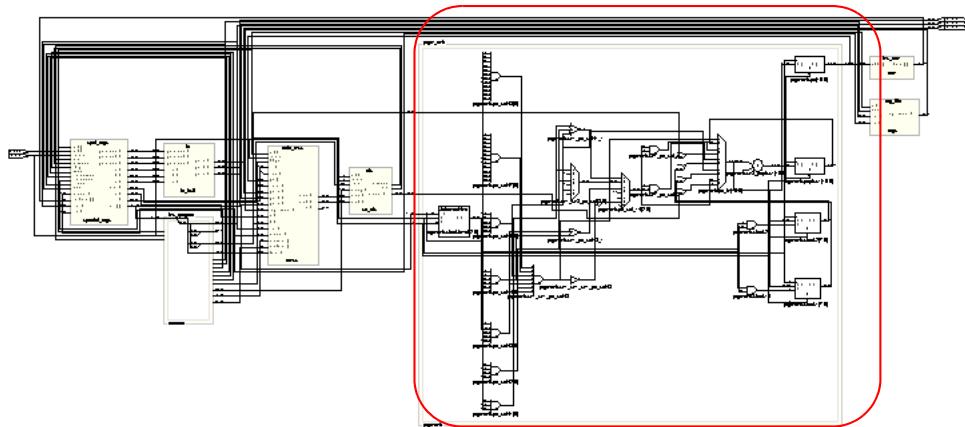
The results differ slightly, depending on the kind of view from which you dissolve instances.

| Starting View | Software Generates a ...                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Filtered      | Filtered view with the internal logic of dissolved instances displayed within hollow bounding boxes (transparent instances), and the hierarchy of the rest of the design unchanged. If the transparent instance does not display internal logic, use one of the alternatives described in step 4 of <a href="#">Viewing Design Hierarchy and Context , on page 190</a> . Use the Back button to return to the undissolved view. |
| Unfiltered    | New, flattened view with the dissolved instances flattened in place (no nesting) to Boolean logic, and the hierarchy of the rest of the design unchanged. Select Unflatten Schematic to return to the top-level unflattened view. You cannot use the Back button to return to previous views because this is a new view.                                                                                                        |

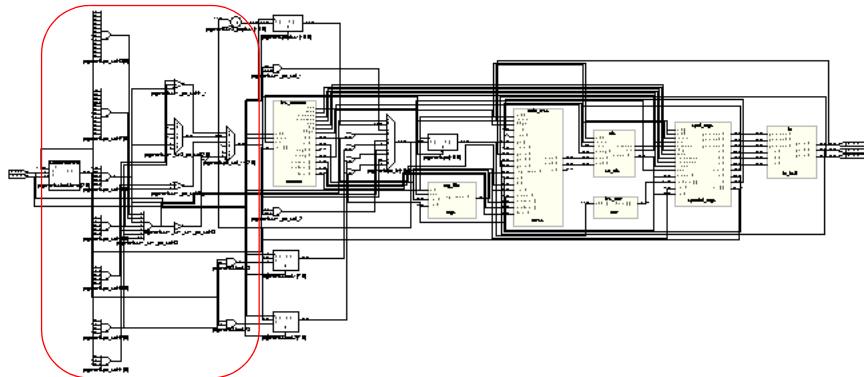
| Starting View | Software Generates a ...                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Filtered      | Filtered view with the internal logic of dissolved instances displayed within hollow bounding boxes (transparent instances), and the hierarchy of the rest of the design unchanged. If the transparent instance does not display internal logic, use one of the alternatives described in step 4 of <a href="#">Viewing Design Hierarchy and Context , on page 190</a> . Use the Back button to return to the undissolved view. |
| Unfiltered    | New, flattened view with the dissolved instances flattened in place (no nesting) to Boolean logic, and the hierarchy of the rest of the design unchanged. Select Unflatten Schematic to return to the top-level unflattened view. You cannot use the Back button to return to previous views because this is a new view.                                                                                                        |

The following figure illustrates this.

Dissolved logic for prgmcntr shown nested when started from filtered view



Dissolve



Use this technique if you only want to flatten part of your design while retaining the hierarchical context. If you want to flatten most of the design, use the technique described in the previous step. Instead of dissolving instances, you can use a combination of the filtering commands and Push/Pop mode.

## Minimizing Memory Usage While Analyzing Designs

When working with large hierarchical designs, use the following techniques to use memory resources efficiently.

- Before you do any analysis operations such as searching, flattening, expanding, or pushing/popping, hide (Hide Instances) the hierarchical instances you do not need. This saves memory resources, because the software does not load the hierarchy of the hidden instances.
- Temporarily divide your design into smaller working files. Before you do any analysis, hide the instances you do not need. Save the design. The `srs` and `srm` files generated are smaller because the software does not save the hidden hierarchy. Close any open HDL Analyst windows to free all memory from the large design. Analyze the design using the smaller, working schematics.
- Filter your design instead of flattening it. If you must flatten your design, hide the instances whose hierarchy you do not need before flattening, or use the Dissolve Instances command. See *Flattening Schematic Hierarchy*, on page 201 for details.
- When searching your design, search by instance rather than by net. Searching by net loads the entire design, which uses memory.
- Limit the scope of a search by hiding instances you do not need to analyze. You can limit the scope further by filtering the schematic in addition to hiding the instances you do not want to search.



## CHAPTER 3

# Defining Objects for Inference

---

This chapter how to set up the HDL code so that the compiler can automatically infer high-level objects or logic structures from it.

- [Defining RAM in the HDL Code](#), on page 208
- [Initializing RAMs](#), on page 230
- [Specifying Register INIT Values](#), on page 243
- [Inferring Shift Registers](#), on page 248
- [Inferring Wide Adders](#), on page 251
- [Defining State Machines](#), on page 253
- [Defining Black Boxes for Synthesis](#), on page 257
- [Instantiating Xilinx Macros and Cores](#), on page 267
- [Working with Buffers](#), on page 284
- [Specifying RLOCs](#), on page 289

# Defining RAM in the HDL Code

You can let the compiler and mapper infer RAM directly from the HDL source code and map it to the appropriate RAM resources on the FPGA. This approach lets you maintain portability. You can also specifically designate RAM to be inferred using attributes in the HDL code.

For further details about defining RAM and automatic RAM inference, see these topics:

- [Block RAM Basics](#), on page 208
- [Setting Attributes to Guide RAM Inference](#), on page 209
- [Inferring Block RAM](#), on page 211
- [Inferring Distributed RAM](#), on page 216
- [Inferring Asymmetric RAM](#), on page 220
- [Inferring Byte-Enable RAM](#), on page 226

For additional examples, see SolvNetPlus articles 039555 (*Inferring Xilinx RAM*), 030578 (*Byte-Enable RAM Support*), and 2560210 (*Verilog RTL Coding Style for True Dual-Port Byte-Enabled RAM*).

## Block RAM Basics

The tool can implement the block RAM it infers using different types of block RAM and different block RAM modes.

### Types of Block RAM

The software can infer different kinds of block RAM, according to how the code is set up. For details about block RAM inference, see [Inferring Block RAM](#), on page 211.

The tool can infer the following kinds of block RAM:

- Single-port RAM
- Dual-port RAM

Based on how the read and write ports are used, dual-port RAM can be further classified as follows:

- Simple dual-port
- Dual-port
- True dual-port
- Multi-port RAM (NRAM)

## Block RAM Modes for Single-Port and Dual-Port RAM

There are three supported Block RAM operating modes, which determine the output of the RAM when write enable is active. The tool infers the mode from the RTL. It is best to explicitly describe the RAM behavior in the code, so as to correctly infer the operating mode you want. Refer to the examples for recommended coding styles.

The block RAM operating modes are described in the following table:

| <b>Mode</b> | <b>When write enable (WE) is active ...</b>                                                                                                                                                                          |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WRITE_FIRST | This is a transparent mode, and the input data is simultaneously written into memory and stored in the RAM data output (DO). DO uses the value of the RAM data input (DI).                                           |
| READ_FIRST  | This mode is read before write. The data previously stored at the write address appears at the RAM data output (DO) first, and then the RAM input data is stored in memory. DO uses the value of the memory content. |
| NO_CHANGE   | RAM data output (DO) remains the same during a write operation, with DO containing the last read data.                                                                                                               |

See SolvNetPlus articles 039555, *Inferring Xilinx RAM* for examples of the modes.

## Setting Attributes to Guide RAM Inference

In addition to the automatic inference by the tool, you can explicitly specify RAM inference with the `syn_ramstyle` and `syn_rw_conflict_logic` attributes.

1. To explicitly define RAM for inference, use the `syn_ramstyle` or `syn_rw_conflict_logic` attributes.

The `syn_ramstyle` attribute explicitly specifies the kind of RAM you want, while the `syn_rw_conflict_logic` attribute specifies that you want to infer a RAM, but leave it to the tool to select the kind of RAM, as appropriate.

- If you use `syn_ramstyle` attribute to define block RAM ([syn\\_ramstyle, on page 751](#)), the following values for the attribute are most relevant:

| <b>syn_ramstyle Value</b> | <b>Description</b>                                                                                                                                                                 |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>block_ram</code>    | Enforces the inference and implementation of a technology-specific RAM.                                                                                                            |
| <code>registers</code>    | Prevents inference of a RAM, and maps the RAM to flip-flops and logic.                                                                                                             |
| <code>no_rw_check</code>  | Does not create overhead logic to account for read-write conflicts.                                                                                                                |
| <code>uram</code>         | Forces the inference of the UltraRAM block.                                                                                                                                        |
| <code>no_uram</code>      | When the <code>no_uram</code> option is specified, the tool prevents the RAM from inferring the UltraRAM (URAM) block. The RAM is mapped to block RAM (RAMB36E2/RAMB18E2) instead. |

- If you specify the `syn_rw_conflict_logic` attribute ([syn\\_rw\\_conflict\\_logic, on page 792](#)), the compiler automatically infers block RAM, according to the design. You can also set this attribute globally. The inferred block RAM does not have inserted bypass logic to account for read-write conflicts and prevent simulation mismatches. In this way its functionality is the same as `syn_ramstyle` with `no_rw_check`, which does not insert bypass logic either.

## 2. Set the attribute in the HDL source code or in an fdc constraint file.

- HDL Source Code

Set the attribute on the Verilog register or VHDL signal that holds the output values of the RAM. The following syntax shows how to specify the attribute in Verilog and VHDL code:

```
Verilog reg [7:0] ram_dout [127:0]
/*synthesis syn_ramstyle = "block_ram"*/;
reg [d_width-1:0] mem [mem_depth-1:0]
/*synthesis syn_rw_conflict_logic = 0*/;
```

```
VHDL attribute syn_ramstyle of ram_dout : signal is "block_ram";
```

- Constraints File

In the fdc Tcl constraints file, set the `syn_ramstyle` attribute on the register mem signal of the RAM, and the `syn_rw_conflict_logic` attribute on the view. You can also set `syn_rw_conflict_logic` globally. See the examples below:

```
define_attribute {i:mem[7:0]} {syn_ramstyle} {block_ram}
define_attribute {v:mem[0:7]} syn_rw_conflict_logic {0}
define_global_attribute syn_rw_conflict_logic {0}
```

## Inferring Block RAM

Based on the design and how you code it, the tool can infer the following kinds of block RAM: single-port, simple dual-port, dual-port, and true dual-port. The details about RAM inference and setup guidelines are described here:

- [Setting up the RTL and Inferring Block RAM](#), on page 211
- [Inferring True Dual-Port RAM](#), on page 214
- [Inferring True Dual-Port Byte-Enabled RAM](#), on page 215

For additional examples, see SolvNetPlus article 039555 (*Inferring Xilinx RAM*) and 2560210 (*Verilog RTL Coding Style for True Dual-Port Byte-Enabled RAM*).

## Setting up the RTL and Inferring Block RAM

To ensure that the tool infers the kind of block RAM you want, do the following:

1. Set up the RAM HDL code in accordance with the following guidelines:
  - The RAM must be synchronous. It must not have any asynchronous control signals connected. The tool does not infer asynchronous block RAM.
  - You must register either the read address or the output.
  - The RAMs must not be too small, as the tool does not infer block RAM for small-sized RAMs. The size threshold varies with the target technology.

2. Check the clocks and read and write ports, so that the tool infers the kind of RAM you want.
  - The following table summarizes how the RAM must be architecture so that the tool can infer it:

| <b>RAM</b>       | <b>Clock</b>         | <b>Read Ports</b>     | <b>Write Ports</b>     |
|------------------|----------------------|-----------------------|------------------------|
| Single-port      | Single clock         | One; same as write    | One; same as read      |
| Simple dual-port | Single or dual clock | One dedicated read    | One dedicated write    |
| Dual-port        | Single or dual clock | Two independent reads | One dedicated write    |
| True dual-port   | Single or dual clock | Two independent reads | Two independent writes |

- To infer simple dual-port (SDP) block RAM, the read and write addresses must be different. In addition, the read and write clocks and enable signals can be different. The tool maps SDP RAMs to RAM primitives in the architecture, using a unique set of addresses, clocks, and enable signals for each port. It might also set the `RAM_MODE` property on the RAM to indicate the RAM mode.
- To infer dual-port block RAM, the read and write addresses must be different. In addition, the read and write clocks and enable signals can be different. The tool uses a unique set of addresses, clocks, and enable signals for each port. It sets properties on the RAM to indicate the RAM mode.
- To infer true dual-port (TDP) RAM, make sure the RAM follows the requirements in the table above. If the writes are made in different processes, you might need to add an explicit `syn_ramstyle` attribute to infer the RAM. See [Inferring True Dual-Port RAM, on page 214](#) for an explanation.

For illustrative code examples, see SolvNetPlus article 039555, *Inferring Xilinx RAM*.

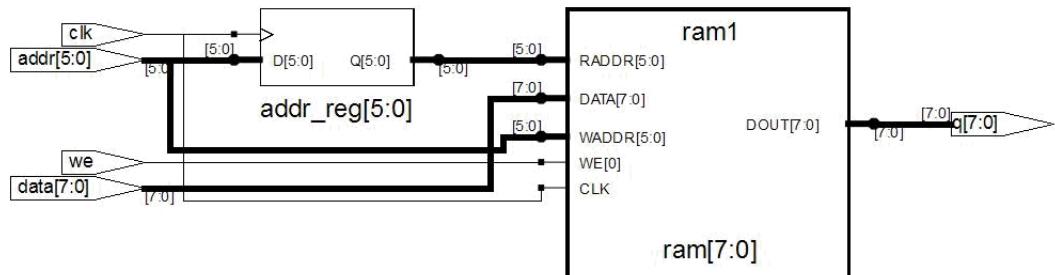
3. If needed, guide automatic inference with the `syn_ramstyle` attribute:
  - To force the inference of block RAM, specify `syn_ramstyle=blockram`.
  - To prevent block RAM from being inferred or if your resources are limited, use `syn_ramstyle=registers`.

- If you know your design does not read and write to the same address simultaneously, specify `syn_ramstyle=no_rw_check` to ensure that the tool does not unnecessarily create bypass logic for resolving conflicts.

For information about setting the `syn_ramstyle` attribute, see [Setting Attributes to Guide RAM Inference, on page 209](#). For the syntax, see [syn\\_ramstyle](#), on page 751.

#### 4. Synthesize the design.

The tool first compiles the design and infers the RAMs, which it represents as abstract technology-independent primitives like `RAM1` and `RAM2`. You can view these RAMs in the schematic view:



It is important that the compiler first infer the RAM, because the tool only maps inferred RAM primitives to block RAM. Any RAM that is not inferred is mapped to registers. You can view the mapped RAMs in the schematic view after mapping.

#### Example: SDP RAM Inference

```

module Read_First_RAM (
    read_clk,
    read_address,
    data_in,
    write_clk,
    rd_en,
    wr_en,
    reg_en,
    write_address,
    data_out);

```

```

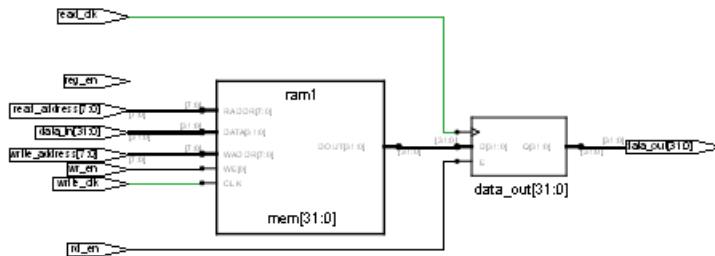
parameter address_width = 8;
parameter data_width = 32;
parameter depth = 256;
input read_clk, write_clk;
input rd_en;
input wr_en;
input reg_en;
input [address_width-1:0] read_address, write_address;
input [data_width-1:0] data_in;
output [data_width-1:0] data_out;
//wire [data_width-1:0] data_out;
reg [data_width-1:0] mem [depth -1 : 0]
/* synthesis syn_ramstyle="no_rw_check" */;
reg [data_width-1:0] data_out;

always @(posedge write_clk)
if(wr_en)
    mem[write_address] <= data_in;

always @(posedge read_clk)
if(rd_en)
    data_out <= mem[read_address];

endmodule

```



## Inferred True Dual-Port RAM

True dual-port RAMs (TDP) are block RAMs with two write ports and two read ports. The compiler extracts a RAM2 primitive for RAMs with two write ports or two read ports and the mapper maps this primitive to TDP RAM. The ports operate independently, with different clocks, addresses and enables.

The tool also sets the **RAM\_MODE** property on the RAM to indicate the RAM mode.

The compiler infers TDP block RAM based on the write processes. The implementation depends on whether the write enables use one process or multiple processes:

- When all the writes are made in one process, there are no address conflicts, and the compiler generates an nram that is later mapped to either true dual-port block RAM. The following coding results in an nram with two write ports, one with write address waddr0 and the other with write address waddr1:

```
always @ (posedge clk)
begin
    if (we1) mem[waddr0] <= data1;
    if (we2) mem[waddr1] <= data2;
end
```

- When the writes are made in multiple processes, the software does not infer a multiport RAM unless you explicitly specify the `syn_ramstyle` attribute with a value that indicates the kind of RAM to implement, or with the `no_rw_check` value. If the attribute is not specified as such, the software does not infer an nram, but infers a RAM with multiple write ports. You get a warning about simulation mismatches when the two addresses are the same.

In the following case, the compiler infers an nram with two write ports because the `syn_ramstyle` attribute is specified. The writes associated with `waddr0` and `waddr1` are `we1` and `we2`, respectively.

```
reg [1:0] mem [7:0] /* synthesis syn_ramstyle="no_rw_check" */;
always @ (posedge clk1)
begin
    if (we1) mem[waddr0] <= data1;
end

always @ (posedge clk2)
begin
    if (we2) mem[waddr1] <= data2;
end
```

## Inferring True Dual-Port Byte-Enabled RAM

The procedure below describes how to specify RAM where you can read/write each byte into a specific address location independently, and how to implement it as block RAM. See SolvNetPlus article 2560210, *Verilog RTL Coding Style for True Dual-Port Byte-Enabled RAM*, for an example.

1. Instantiate the true dual-port RAM  $n$  number of times, where  $n$  is the number of bytes for a particular RAM address.

In the following example, `ram_dp` is instantiated twice because there are two bytes in the address:

```
ram_dp u1 (clk1, clk2, dia[7:0], addra, wea[0], doa[7:0], dib[7:0], addrb, web[0],  
dob[7:0]);  
ram_dp u2 (clk1, clk2, dia[15:8], addra, wea[1], doa[15:8], dib[15:8], addrb,  
web[1], dob[15:8]);
```

2. To map the true dual-port RAM into a block RAM, add the `syn_ramstyle="block_ram"` attribute to the true dual-port RAM module.
3. Run compile.

The RTL schematic shows two instantiations, as specified.

4. Run map.

After synthesis, check the resource utilization report to make sure that two block RAMs were inferred, as specified.

## Cascading Block RAM

### Xilinx UltraScale

The synthesis tools provide support for multi-level cascading of block RAM (RAMB36E2) for Xilinx UltraScale+ devices. Multi-level cascading provides QOR improvement. Block RAMs are built from the bottom-up directly in the block ram column. The length of the block RAM chain is controlled by the `syn_bram_cascade_height` attribute. See [syn\\_bram\\_cascade\\_height, on page 533](#) for more information.

## Inferring Distributed RAM

Distributed RAMs are inferred memories that the tool does not map to the dedicated block RAM memory resources. Instead, they are implemented with regular lookup tables (LUTs) within a slice of the configurable logic block (CLB). Typically, distributed RAMs are used for small embedded memory blocks, like synchronous and asynchronous FIFOs, for example.

Distributed RAMs have a single clock. Reads can be asynchronous, but writes are synchronous. The tool supports the following memory types for distributed RAM:

- Single-port distributed RAM
- Dual-port distributed RAM
- Multiport distributed RAM

If you specify the `syn_rw_conflict_logic` attribute, the compiler automatically infers distributed RAM, depending on the design.

For information about setting up your design to infer distributed RAM, see [Inferring Distributed RAM, on page 217](#) and [Setting Attributes to Guide RAM Inference, on page 209](#).

## Inferring Distributed RAM

Based on the design and how you code it, the tool can infer the following kinds of distributed RAM: single-port, dual-port, and multiport. To ensure that the tool infers the kind of RAM you want, do the following:

1. Set up the RAM HDL code in accordance with the following guidelines:
  - The RAM can be synchronous or asynchronous.
  - Do not register the read address or the output. If you add a register, the software implements block RAM, not distributed RAM.
  - To be automatically inferred, make sure the RAM is above the minimum size threshold.

You can also guide inference with the `syn_ramstyle` attribute, as described in step 3.

2. Check that the clocks and read and write ports are set up to infer the kind of RAM you want.

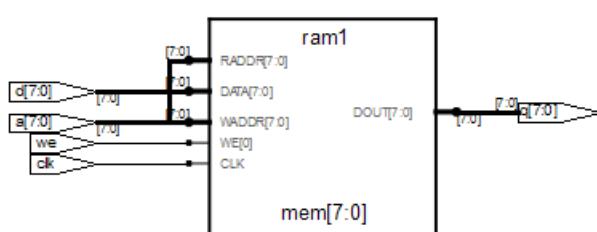
The following table summarizes how the RAM must be set up:

| Distributed RAM | Read Ports                    | Write Ports           |
|-----------------|-------------------------------|-----------------------|
| Single Port     | One; same as write            | One; same as read     |
| Dual Port       | One dedicated read            | One dedicated write   |
| Multiport       | 3 or 4 independent read ports | One shared write port |

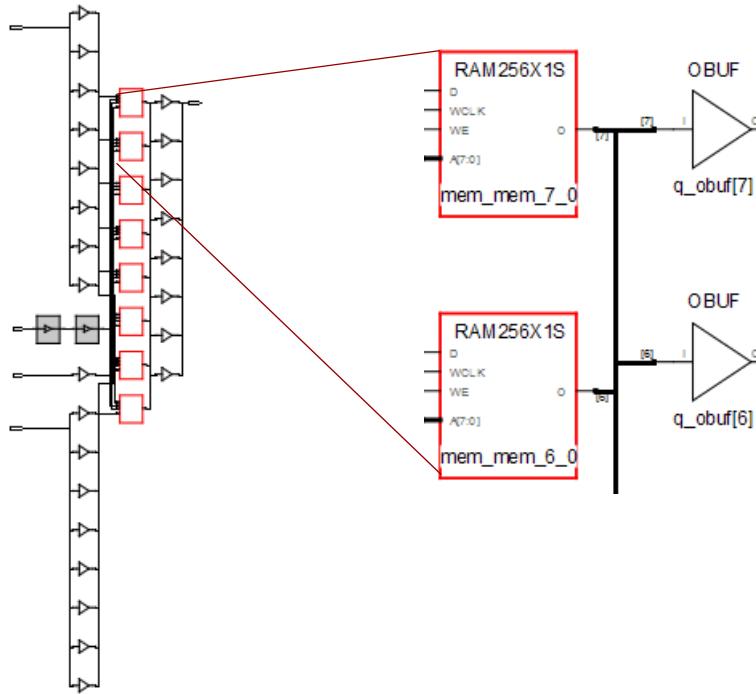
The inference of multiport distributed RAM depends on whether one or multiple write processes are used. See [Inferring True Dual-Port RAM, on page 214](#) for details.

3. If needed, guide automatic inference with the `syn_ramstyle` attribute:
  - To force the inference of distributed RAM, specify `syn_ramstyle=select_ram`.
  - To prevent a distributed RAM from being inferred, use `syn_ramstyle=logic`.
4. Synthesize the design.

The tool first compiles the design and infers the RAMs, which it maps to abstract primitives. The following figure shows an inferred RAM in the schematic view after compiling:



The tool then maps the inferred RAM primitives to technology-specific distributed RAM. The following view after mapping shows the previously inferred RAM mapped to RAM resources. The RAMs are highlighted in red:



## Single-Port Distributed RAM Inference

For single-port RAM, the same address is used for reading and writing operations. The tool automatically infers single-port distributed RAM in the following cases:

- The block RAM resources have been exhausted
- The RAM contains an asynchronous read port
- The `syn_ramstyle` attribute is set to `select_ram`

The RTL view shows a RAM1 primitive inferred.

## Dual-Port Distributed RAM Inference

For distributed RAM, dual-port inference is the same as for block RAM.

## Multiport Distributed RAM Inference

Multiport distributed RAM primitives, such as RAM32M and RAM64M, have four ports, generally configured as one write and three read ports. The RAM configurations follow the DRC rules provided by Xilinx.

When the compiler encounters more than two ports in the RTL, it infers an nram primitive. The nrams are mapped to RAM32M or RAM64M primitives by the mapper.

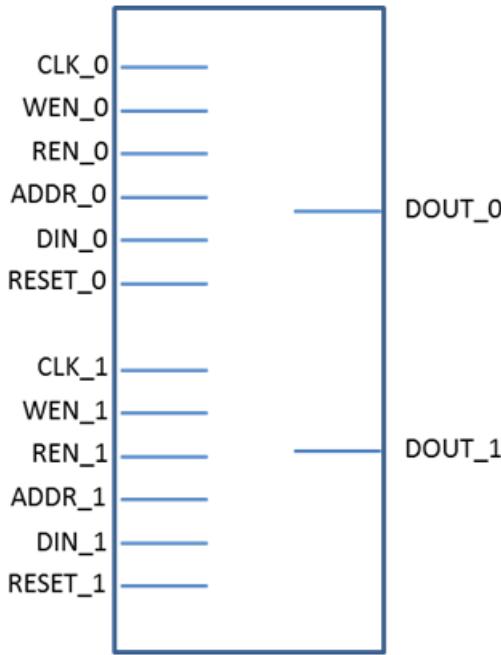
## Inferring Asymmetric RAM

RAMs with different port widths for read and write operations are called asymmetric RAMs. The tool supports different port widths for read and write ports within a single block RAM primitive. The read and write widths vary by  $2^{**n}$ .

1. To infer asymmetric RAM automatically, code it as a contiguous write operation.

For code examples of asymmetric RAM, see SolvNetPlus article 039555, *Inferring Xilinx RAM*.

2. To infer asymmetric RAM as a black box model, do the following:
  - Instantiate the `syn_asym_tdp_ram_model` black box model in your design.



- Specify parameter values for it, according to your design:

#### Asymmetric RAM Parameter Descriptions

| Parameter        | Type    | Valid Values         | Description                          |
|------------------|---------|----------------------|--------------------------------------|
| PORT_0_DEPTH     | Integer | $2^{ADDR\_0\_WIDTH}$ | Port 0 RAM depth                     |
| ADDR_0_WIDTH     | Integer | 32*                  | Port 0 Address width                 |
| DIN_0_WIDTH      | Integer | Up to 128**          | Port 0 Data width                    |
| PORT_1_DEPTH     | Integer | $2^{ADDR\_1\_WIDTH}$ | Port 1 RAM depth                     |
| ADDR_1_WIDTH     | Integer | 32                   | Port 1 Address width                 |
| DIN_1_WIDTH      | Integer | Up to 128            | Port 1 Data width                    |
| OUTPUT_REG       | Integer | 0   1                | Inserts output data registers        |
| READ_ADDRESS_REG | Integer | 0   1                | Inserts input read address registers |

### Asymmetric RAM Parameter Descriptions

|                    |         |                                   |                                                                                                     |
|--------------------|---------|-----------------------------------|-----------------------------------------------------------------------------------------------------|
| <b>RESET_TYPE</b>  | String  | None   Synchronous   Asynchronous | Indicates the reset port type                                                                       |
| <b>WRITE_FIRST</b> | Integer | 0   1                             | Indicates the RAM mode.<br>See <a href="#">Specify RAM mode configuration values: , on page 222</a> |

\* - 12 bits. Currently, the maximum address width is limited by the depth of a single block RAM.  
 \*\* - 36 bits. Currently, the maximum data width is limited by the width of the Xilinx block RAM.

### 3. Specify RAM mode configuration values:

You can specify WRITE\_FIRST and READ\_FIRST modes.

- For WRITE\_FIRST mode, set WRITE\_FIRST and OUTPUT\_REG to 1, or set READ\_ADDRESS\_REG to 1. For WRITE\_FIRST mode, if a read-during-write occurs, then the new data is read out.
- In READ\_FIRST mode, set both READ\_ADDRESS\_REG and WRITE\_FIRST must be set to 0. For READ\_FIRST mode, if a read-during-write occurs, then the old data is read out.

This table shows how the RAM ports can be mapped for Ports 0 and 1.

### Asymmetric RAM Port Descriptions

| <b>Port</b> | <b>Direction</b> | <b>Description</b>                                                                      |
|-------------|------------------|-----------------------------------------------------------------------------------------|
| CLK_0       | Input            | Port 0 Clock                                                                            |
| WEN_0       | Input            | Port 0 Write enable. (Optional port)                                                    |
| REN_0       | Input            | Port 0 Read enable. (Optional port)                                                     |
| RESET_0     | Input            | Port 0 Reset. RESET_TYPE parameter specifies whether the reset is enabled and its type. |
| ADDR_0      | Input            | Port 0 Address. Address width is set using the ADDR_0_WIDTH parameter.                  |
| DIN_0       | Input            | Port 0 Data in. Data width is set using the DIN_0_WIDTH parameter.                      |

### Asymmetric RAM Port Descriptions

|         |        |                                                                                         |
|---------|--------|-----------------------------------------------------------------------------------------|
| DOUT_0  | Output | Port 0 Data out. Data width is set using the DIN_0_WIDTH parameter.                     |
| CLK_1   | Input  | Port 1 Clock                                                                            |
| WEN_1   | Input  | Port 1 Write enable. (Optional port)                                                    |
| REN_1   | Input  | Port 1 Read enable. (Optional port)                                                     |
| RESET_1 | Input  | Port 1 Reset. RESET_TYPE parameter specifies whether the reset is enabled and its type. |
| ADDR_1  | Input  | Port 1 Address. Address width is set using the ADDR_1_WIDTH parameter.                  |
| DIN_1   | Input  | Port 1 Data in. Data width is set using the DIN_1_WIDTH parameter.                      |
| DOUT_1  | Output | Port 1 Data out. Data width is set using the DIN_1_WIDTH parameter.                     |

4. Set parameter requirements to infer asymmetric RAM correctly:
- True dual-port asymmetric RAM must satisfy the following equation:  

$$(\text{PORT\_0\_DEPTH} * \text{DIN\_0\_WIDTH}) = (\text{PORT\_1\_DEPTH} * \text{DIN\_1\_WIDTH})$$
  - Port depths must be a power of 2.

### Verilog Example: Asymmetric RAM Instantiation Model

This is a Verilog template that you can use to create an instantiation model for the asymmetric RAM.

```
syn_asym_tdp_ram_model
#(
    .PORT_0_DEPTH(PORT_0_DEPTH),
    .ADDR_0_WIDTH(ADDR_0_WIDTH),
    .DIN_0_WIDTH(DIN_0_WIDTH),

    // Port 0
    .PORT_1_DEPTH(PORT_1_DEPTH),
    .ADDR_1_WIDTH(ADDR_1_WIDTH),
    .DIN_1_WIDTH(DIN_1_WIDTH),

    // RAM Control Signals
```

```

        .OUTPUT_REG(OUTPUT_REG),
        .READ_ADDRESS_REG(READ_ADDRESS_REG),
        .RESET_TYPE(RESET_TYPE),
        .WRITE_FIRST(WRITE_FIRST)
    )
Model_inst (
    .CLK_0(CLK_0),
    .WEN_0(WEN_0),
    .REN_0(REN_0),
    .RESET_0(RESET_0),
    .ADDR_0(ADDR_0),
    .DIN_0(DIN_0),
    .DOUT_0(DOUT_0),

    // Port 1
    .CLK_1(Clk_1),
    .WEN_1(WEN_1),
    .RESET_1(RESET_1),
    .ADDR_1(ADDR_1),
    .DIN_1(DIN_1),
    .DOUT_1(DOUT_1)
);

```

## VHDL Example: Asymmetric RAM Instantiation Model

This is a VHDL template you can use to create an instantiation model for the asymmetric RAM.

```

Model_inst : syn_asym_tdp_ram_model
generic map (
    PORT_0_DEPTH => PORT_0_DEPTH,
    ADDR_0_WIDTH => ADDR_0_WIDTH,
    DIN_0_WIDTH => DIN_0_WIDTH,

    -- Port 0
    PORT_1_DEPTH => PORT_1_DEPTH,
    ADDR_1_WIDTH => ADDR_1_WIDTH,
    DIN_1_WIDTH => DIN_1_WIDTH,

    -- RAM Control Signals
    OUTPUT_REG => OUTPUT_REG,
    READ_ADDRESS_REG => READ_ADDRESS_REG,
    RESET_TYPE => RESET_TYPE,
    WRITE_FIRST => WRITE_FIRST
)
port map(

```

```

CLK_0 => CLK_0,
WEN_0 => WEN_0,
REN_0 => REN_0,
RESET_0 => RESET_0,
ADDR_0 => ADDR_0,
DIN_0 => DIN_0,
DOUT_0 => DOUT_0,

-- Port 1
CLK_1 => Clk_1,
WEN_1 => WEN_1,
REN_1 => REN_1,
RESET_1 => RESET_1,
ADDR_1 => ADDR_1,
DIN_1 => DIN_1,
DOUT_1 => DOUT_1
);

```

## Instantiation of Parameters Example

This example shows how the Verilog or VHDL templates above can actually specify parameters for the RAM. The following code snippet is an explicit WRITE\_FIRST style RAM with asynchronous reset:

```

PORT_0_DEPTH(16),
.ADDR_0_WIDTH(4),
.DIN_0_WIDTH(32),
// Port 0
.PORT_1_DEPTH(32),
.ADDR_1_WIDTH(5),
.DIN_1_WIDTH(16),
// RAM Control Signals
.OUTPUT_REG(1),
.READ_ADDRESS_REG(1),
.RESET_TYPE("Asynchronous"),
.WRITE_FIRST(1)

```

## Limitation

Xilinx device limitations for the RAM include the following:

- Only Virtex-7 devices are supported.
- The maximum depth for the RAM is 1K and maximum width is 36 bits.
- For higher capacity devices, you must multiplex individual block RAMs.

## Inferring Byte-Enable RAM

Instead of using a single enable bit to write data serially to one location at a time, byte-enable RAM uses multiple enable signals to read or write data to multiple locations in a block RAM simultaneously. This allows data to be processed in parallel, increasing the speed and overall efficiency of the FPGA. The tool infers byte-enable RAM.

The following procedure describes how to write the RTL code so that the tool infers byte-enable RAM. For information about true dual-port byte-enabled RAM, see [Inferring True Dual-Port Byte-Enabled RAM, on page 215](#).

1. Use exclusive if or loop conditional statements to define the enable input of a block RAM as a byte-enable pin.

You can define one enable per byte, up to a maximum of four enables. Use 1, 2, 4, or 8 bits to describe the enable input. This Verilog example defines a 4-bit byte enable using exclusive if statements.

```
module test(clock, rden, address, byteena, data, q);
    input clock;
    input rden;
    input [8:0] address;
    input [3:0] byteena;
    input [31:0] data;
    output reg [31:0] q;
    reg [31:0] mem [511:0];

    always @ (posedge clock)
    begin
        if (byteena[3])
            mem[address][31:24] <= data[31:24];
        if (byteena[2])
            mem[address][23:16] <= data[23:16];
        if (byteena[1])
            mem[address][15:8] <= data[15:8];
        if (byteena[0])
            mem[address][7:0] <= data[7:0];
    end
    ...
endmodule
```

This is an example of VHDL code to infer byte-enable RAM:

```
proc1: process (clk)      begin
    if (rising_edge(clk)) then
        if (sel1 = '1') then
            mem(waddr) (dwidth/2-1 downto 0) := data;
        elsif (sel2 = '1') then
            mem(waddr) (dwidth/2-1 downto 0) := data2;
        end if;   end if;
end process;
```

2. Do not mix the byte-enable logic with other signal statements in the HDL code:

If you do, it might result in the creation of glue logic that is implemented in LUTs outside the block RAM.

3. Compile and map as usual. The tool infers byte-enable block RAM.
4. Check for RAM inference in the compiler report.

For details and examples on inferring byte-wide write enable RAM, refer to SolvNetPlus article 030578, *Byte-Enable RAM Support*.





# Initializing RAMs

You can specify startup values for RAMs and pass them to place and route tools. There are different ways to set the initial values:

## VHDL

- Signal or variable declarations  
See [Initializing RAMs Using VHDL Signal or Variable Declarations , on page 230.](#)
- INIT property on label  
See [Initializing RAM in VHDL with the INIT Property , on page 233.](#)

## Verilog

- \$readmemb or \$readmemh system tasks  
See [Initializing RAM Using Verilog \\$readmemh or \\$readmemb , on page 234.](#)
- INIT property in defparam statement or global comment /\* synthesis INIT or /\*synthesis INIT\_xx=<value>  
See [Initializing RAM in Verilog Using the INIT Property , on page 238.](#)

## Attributes

- INIT property in SCOPE
  - define\_attribute statements in the fdc file
- See [Specifying the INIT Property with Attributes , on page 241.](#)

Note the following differences between using the INIT property and the HDL code-specific methods:

- You can use the INIT property with either VHDL or Verilog code. By contrast, the \$readmemb and \$readmemh system tasks are only applicable in Verilog.
- The Verilog initial values only affect the output of the compiler, not the mapper. They ensure that the synthesis results match the simulation results, and are not forward-annotated.

## Initializing RAMs Using VHDL Signal or Variable Declarations

There are two ways to initialize the RAM using declarations in the VHDL code: with signal declarations or with variable declarations.

## Initializing VHDL Rams with Signal Declarations

The following example shows a single-port RAM that is initialized with signal initialization statements. For alternative methods, see the table in [Initializing RAMs, on page 230](#).

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

entity w_r2048x28 is
port (
    clk : in std_logic;
    adr : in std_logic_vector(10 downto 0);
    di : in std_logic_vector(26 downto 0);
    we : in std_logic;
    dout : out std_logic_vector(26 downto 0));
end;

architecture arch of w_r2048x28 is

-- Signal Declaration --

type MEM is array(0 to 2047) of std_logic_vector (26 downto 0);
signal memory : MEM := (
"1111111111111000000000000000",
,"111110011011101010011110001"
,"111001111000111100101100111"
,"110010110011101110011110001"
,"10100111100011111100110111"
,"10000000000000111111111111"
,"010110000111001111100110111"
,"00110100110011110011110001"
,"000110000111001100101100111"
,"000001100100011010011110001"
,"000000000000001000000000000"
,"0000001100100010101100001110"
,"000110000111000011010011000"
,"001101001100010001100001110"
,"010110000111000000011001000"
,"0111111111111000000000000000"
,"101001111000110000011001000"
,"110010110011100001100001110"
,"11100111100011001101001100"
,"111110011011100101100001110"
,"11111111111110111111111111"
,"111110011011101010011110001"
```

```
, "11100111000111100101100111"
, "110010110011101110011110001"
, "10100111100011111100110111"
, "1000000000000001111111111111"
, others => (others => '0'));

begin
process(clk)

begin
    if rising_edge(clk) then
        if (we = '1') then
            memory(conv_integer(adr)) <= di;
        end if;
        dout <= memory(conv_integer(adr));
    end if;
end process;

end arch;
```

## Initializing VHDL Rams with Variable Declarations

The following example shows a RAM that is initialized with variable declarations. For alternative methods, see the table in [Initializing RAMs, on page 230](#).

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

entity one is
generic (data_width : integer := 6;
         address_width :integer := 3
);
port (data_a :in std_logic_vector(data_width-1 downto 0);
      raddr1 :in unsigned(address_width-2 downto 0);
      waddr1 :in unsigned(address_width-1 downto 0);
      we1 :in std_logic;
      clk :in std_logic;
      out1 :out std_logic_vector(data_width-1 downto 0));
end;

architecture rtl of one is
type mem_array is array(0 to 2**address_width -1) of
    std_logic_vector(data_width-1 downto 0);
begin
```

```
WRITE1_RAM : process (clk)
    variable mem : mem_array := (1 => "111101", others => (1=>'1',
        others => '0'));
begin
    if rising_edge(clk) then
        out1 <= mem(to_integer(raddr1));
        if (we1 = '1') then
            mem(to_integer(waddr1)) := data_a;
        end if;
    end if;
end process WRITE1_RAM;
end rtl;
```

## Initializing RAM in VHDL with the INIT Property

You can also define the INIT property as an attribute ([Specifying the INIT Property with Attributes, on page 241](#)). See [Initializing RAMs, on page 230](#) for alternative methods to initialize RAMs.

Follow these steps to include the INIT property in VHDL code:

1. Add the INIT property.
  - Attach the INIT property to the label as shown:

|           |                                                                 |
|-----------|-----------------------------------------------------------------|
| RAM       | attribute INIT of <i>object</i> : label is " <i>value</i> ";    |
| Block RAM | attribute INIT_xx of <i>object</i> : label is " <i>value</i> "; |

---

- Keep the entire statement on one line. Let the editor wrap lines if it supports line wrap, but do not press Enter until the end of the statement.

2. For RAM, specify hex values for the INIT statement as shown:

```
attribute INIT of RAM1 : label is "0000";:
```

3. For Virtex block RAM, specify 16 different INIT statements.

- Define the `INIT_xx=value` property as follows:

`xx` Indicate the part of the RAM you are initializing with a number from 00 to FF.

`value` Set the initialization value, in hex. You have 64 hex values in each INIT ( $64 \times 4 = 256$  and  $256 \times 16 = 4K$ ), because there are 16 INIT statements.

---

All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16.

- End the initialization data with a semicolon.

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx place-and-route software. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

## Initializing RAM Using Verilog \$readmemh or \$readmemb

In Verilog, you specify startup values using initial statements, which are procedural assign statements guaranteed by the language to be executed by the simulator at the start of simulation. This means that any assignment to a variable within the body of the initial statement is treated as if the variable was initialized with the corresponding LHS value. You can then initialize memories using the built-in load memory system tasks `$readmemb` (binary) and `$readmemh` (hex).

The following procedure is the recommended method for specifying initial values in Verilog. See [Initialization RAMs, on page 230](#) for alternative methods to initialize RAMs.

1. Create a data file with an initial value for every address in the memory array.

This file can be a binary file or a hex file, to match the task enable statement you use in the next step. See [Initialization Data File, on page 235](#) for details of the formats.

2. Do the following in the Verilog file to define the module:

- Include the appropriate task enable statement, \$readmemb or \$readmemh, in the initial statement for the module:

```
$readmemb ("fileName", memoryName [, startAddress [, stopAddress]]);
```

```
$readmemh ("fileName", memoryName [, startAddress [, stopAddress]]);
```

Use \$readmemb with a binary initialization file and \$readmemh with a hex file.

- Make sure the array declaration matches the order in the initial value data file you specified. As the file is read, each number encountered is assigned to a successive word element of the memory. The software starts with the left-hand address in the memory declaration, and loads consecutive words until the memory is full or the data file has been completely read. The loading order is the order in the declaration. For example, with the following memory definition, the first line in the data file corresponds to address 0:

```
reg [7:0] mem_up [0:63]
```

With this next definition, the first line in the data file applies to address 63:

```
reg [7:0] mem_down [63:0]
```

See [Address Initialization, on page 237](#) for more information.

3. Include one of the task enable statements, \$readmemb or \$readmemh, in the initial statement for the module:

```
$readmemb ("fileName", memoryName [, startAddress [, stopAddress]]);  
$readmemh ("fileName", memoryName [, startAddress [, stopAddress]]);
```

Use \$readmemb for a binary file and \$readmemh for a hex file. The tool forward-annotates initial values to place and route.

## Initialization Data File

The initialization data file read by the \$readmemb and \$readmemh system tasks, contains the initial values to be loaded into the memory array. You can reference the file with an include path.

The data initialization file contains the following:

- Initial values for every memory task, binary for \$readmemb tasks, or hexadecimal for \$readmemh tasks

- The format for the binary initialization file (`$readmemb`) looks like this:

```
1111111111111111111111111111111100110111 /* data for address 0 */
1111111111111111111111111111111101100111 /* data for address 1 */
11111111111111111111111111111111000010
1111111111111111111111111111111100100001
1111111111111111111111111111111101110000
11111111111111111111111111111111011100110
11111111111111111111111111111111011100110
... /* continues until Address 1999 */
```

- The format for the hexadecimal data file (`$readmemh`) looks like this:

```
FFFFF37    /* data for address 0 */
FFFFF63    /* data for address 1 */
FFFFFC2
FFFFF21
.../* continues until Address 1999 */
```

If an initial value is not specified, the tool initializes unaddressed memory locations to 0. If there is a width mismatch between an initialization value and the memory width, the tool terminates the loading of the memory array, but any values initialized before the mismatch are retained.

- Embedded hexadecimal addresses

Embedded hexadecimal addresses must be prefaced with an at sign (@) as shown in the example below.

```
FFFFF37 /* data for address 0 */
FFFFF63 /* data for address 1 */
@0EA    /* memory address 234
FFFFFC2 /* data for address 234*/
FFFFF21 /* data for address 235*/
...
@0A7    /* memory address 137
FFFFF77 /* data for address 137*/
FFFFF7A /* data for address 138*/
...
```

When the \$readmemb or \$readmemh system task encounters an embedded address specification, it starts loading subsequent data at that memory location.

You can use either uppercase or lowercase in the address, but there must no space between the @ and the hex address. You can include any number of address specifications in the file, in any order.

When addressing information is specified both in the system task and in the data file, the addresses in the data file must be within the address range specified by the system task arguments; otherwise, you get an error message and the load operation is terminated.

- White space (spaces, new lines, tabs, and form-feeds)
- Comments (both comment formats are allowed)

## Address Initialization

Unless you specify an embedded internal address, the tool assigns each value to a successive word element of the memory. If no addressing information is specified either with the task statement or with an embedded address in the initialization file itself, the default starting address is the lowest available address in the memory. Consecutive words are loaded until either the highest address in the memory is reached or the data file is completely read.

If a start address is specified without a finish address, loading starts at the specified start address and continues upward toward the highest address in the memory. If both start and finish addresses are specified, loading begins at the start address and continues until the finish address is reached (or until all initialization data is read).

For example:

```
initial
begin
  //$/readmemh ("mem.ini", ram_bank1)
    /* Initialize RAM with contents from locations 0 thru 31*/;

  //$/readmemh ("mem.ini", ram_bank1,0)
    /* Initialize RAM with contents from locations 0 thru 31*/;

  $readmemh ("mem.ini", ram_bank1, 0, 31)
    /* Initialize RAM with contents from locations 0 thru 31*/;

  $readmemh ("mem.ini", ram_bank2, 31, 0)
    /* Initialize RAM with contents from locations 31 thru 0*/;
```

## Initializing RAM in Verilog Using the INIT Property

You can initialize and forward-annotate the values for Xilinx Verilog RAMs by specifying the INIT property. In Verilog, there are two ways to do this: with the defparams statement or by specifying the property in a global comment. The following procedures describe these two methods.

- [Using defparam to Specify Initialization Values for Xilinx RAMs](#), on page 238
- [Using Global Comments to Specify Initialization Values for Xilinx RAM](#), on page 239

### Using defparam to Specify Initialization Values for Xilinx RAMs

You can also define the INIT property in a comment or as an attribute ([Specifying the INIT Property with Attributes](#), on page 241). See [Initializing RAMs](#), on page 230 for alternative methods to initialize RAMs.

1. Include defparam statements in the Verilog file, using one statement for each word. Use the following syntax for the INIT property:

**defparam name.INIT\_xx=value;**

**name** Is the name of the RAM.

**xx** Indicates the part of the RAM you are initializing. It can be any hex number from 00 to FF.

**value** Sets the initialization value in hex.

The following example for Virtex block RAM would have 16 statements, because it is 4K bits in size. Each statement has 64 hex values in each INIT, because there are 16 INIT statements (64 x 4 and 256 x 16 = 4K).

```
RAMB4_S4 pkt_len_ram_lo (
    .CLK    (clock),
    .RST    (1'b0),
    .EN     (1'b1),
    .WE     (we),
    .ADDR   (address),
    .DI     (data),
    .DO     (q)
);
defparam pkt_len_ram_lo.INIT_00=
```

```
"00170016001500140013001200110010000f000e000d000c000b000a00090008 ";
defparam pkt_len_ram_lo.INIT_01=
"00270026002500240023002200210020001f001e001d001c001b001a00190018";
defparam pkt_len_ram_lo.INIT_02=
"00370036003500340033003200310030002f002e002d002c002b002a00290028";
...
defparam pkt_len_ram_lo.INIT_OF=
"0107010601050104010301020101010000ff00fe00fd00fc00fb00fa00f900f8";
```

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx P&R software in the EDIF netlist. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

## Using Global Comments to Specify Initialization Values for Xilinx RAM

You can also define the INIT property in a defparam statement or as an attribute ([Specifying the INIT Property with Attributes, on page 241](#)). See [Initializing RAMs, on page 230](#) for alternative methods to initialize RAMs.

### 1. Add the INIT property.

- Attach the INIT property to the instance as shown:

|           |                                   |
|-----------|-----------------------------------|
| RAM       | /* synthesis INIT = "value" */    |
| Block RAM | /* synthesis INIT_xx = "value" */ |

---

See step 2 for details on RAM and step 3 for block RAM.

- Keep the entire statement on one line. Let the editor wrap lines if it supports line wrap, but do not press Enter until the end of the statement.

### 2. For RAM, specify a hex value for the INIT statement as shown here:

```
RAM16X1S RAM1(...) /* synthesis INIT = "0000" */;
```

3. Do the following to use the INIT property with block RAM:

- Define the INIT\_xx=value property as follows:

**xx** Indicates the part of the RAM to initialize. Use a number from 00 to FF.

**value** Sets the initialization value, in hex. There are 64 hex values in each INIT ( $64 \times 4 = 256$  and  $256 \times 16 = 4K$ ), because there are 16 INIT statements.

- Specify 16 different INIT statements.

All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16.

In the following example, each statement has 64 hex values in each INIT, because there are 16 INIT statements ( $64 \times 4$  and  $256 \times 16 = 4K$ ).

```
RAMB4_S4 pkt_len_ram_lo (
    .CLK  (clock),
    .RST  (1'b0),
    .EN   (1'b1),
    .WE   (we),
    .ADDR (address),
    .DI    (data),
    .DO    (q)
)

/* synthesis
INIT_00="00170016001500140013001200110010000f000e000d000c000b000a00090008
INIT_01="00270026002500240023002200210020001f001e001d001c001b001a00190018"
INIT_02="00370036003500340033003200310030002f002e002d002c002b002a00290028"
...
INIT_0F="0107010601050104010301020101010000ff00fe00fd00fc00fb00fa00f900f8"
*/;
```

- End the initialization data with a semicolon.

When you synthesize the design, the software forward-annotates the RAM initialization information to the Xilinx place-and-route software. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before place and route.

## Initializing Xilinx RAM

In addition to the methods described in [Initializing RAMs, on page 230](#), you can also define and forward-annotate Xilinx RAM initialization values as summarized in this table:

### Verilog

- \$readmemebh or \$readmemh  
See [Initializing RAM Using Verilog \\$readmemh or \\$readmemb , on page 234](#).
- INIT property in defparam statement  
See [Initializing RAM in Verilog Using the INIT Property , on page 238](#).
- INIT property in global comment /\* synthesis INIT or /\*synthesis INIT xx=<value>  
See [Initializing RAM in Verilog Using the INIT Property , on page 238](#).

### VHDL

- INIT property on label  
See [Initializing RAM in VHDL with the INIT Property , on page 233](#).

### Attributes

- INIT property in SCOPE  
See [Specifying the INIT Property with Attributes , on page 241](#).
- define\_attribute statements in the sdc file  
See [Specifying the INIT Property with Attributes , on page 241](#).

Note the following differences between the above methods:

- You can use the INIT property with any code. The \$readmemb and \$readmemhb system tasks are only applicable in Verilog.
- The Verilog initial values only affect the output of the compiler, not the mapper. They ensure that the synthesis results match the simulation results, and are not forward-annotated.

## Specifying the INIT Property with Attributes

When you set the INIT property in the source code, it is difficult to pass on the values if the RAM instances are mapped to registers. When you specify INIT as an attribute, either in the SCOPE window or the constraint file, you are working with a mapped RAM, and the values are passed to the P&R tool. You can use this method to specify the initialization values for a RAM whether you are using Verilog or VHDL.

1. Compile and map the design.
2. To specify the INIT property in the fdc file, use define\_attribute:
  - For RAM, define the INIT property:

```
define_attribute {RAM1} INIT {0000}
```

- For block RAM, define the INIT\_xx = value property.

**xx** Indicates the part of the RAM you are initializing with a number from 00 to FF.

**value** Sets the initialization value, in hex. There are 64 hex values in each INIT (64 x 4 = 256 and 256 x 16 = 4K), because there are 16 INIT statements.

---

All Virtex block RAMs have 16 INIT statements because they are all 4Kbits in size, although they are configured differently: 4Kx1, 2Kx2, 1Kx4, 512x8, and 256x16. The following example shows a value of ABBADABAABBADABA defined for INIT\_00 and INIT\_01 on mem.mem\_0\_0 in the fdc file:

```
define_attribute {i:mem.mem_0_0} INIT_00 {ABBADABAABBADABA}  
define_attribute {i:mem.mem_0_0} INIT_01 {ABBADABAABBADABA}
```

3. To specify the INIT property in the constraints window, follow these steps:
  - Select the RAM in .
  - Either select the RAM in a HDL Analyst view and drag it into the appropriate column, or type in the attribute.
  - Enter the initialization information as described in the previous step.
4. Define the INIT (RAM) or INIT\_xx = value (Block RAM) property in SCOPE. Alternatively you can edit the fdc file using define\_attribute statements.
5. Compile and map the design.

When you synthesize the design, the software forward-annotates the initialization values as constraints to the place-and-route software. The INIT values are forward-annotated as is, and you must ensure that the values are in the right format before P&R.

# Specifying Register INIT Values

You can specify initial values for registers so that the RTL, gate-level simulation, and the final implementation results match. You can specify INIT values for registers either with the HDL initialization specification built into Verilog or VHDL, or by adding the synthesis attribute. You can then pass the values to the Xilinx P&R tools.

Both methods are described in the following procedure, but the HDL specification method is recommended.

1. To ensure that the register is not optimized away during synthesis, set the `syn_preserve` directive on the register in the source code.

Use this directive even if you define the INIT values with a constraint in the `sdc` file. If you do not have this directive, the register can be removed during optimization.

2. To set a register value using the HDL initialization feature, use the following syntax:

**HDL Initialization**    `reg myreg=0;`  
                            `initial myreg=0; (Verilog only)`

---

**Verilog HDL Initialization**    `reg error_reg = 1'b0;`  
                                    `reg [7:0] address_reg = 8'hff;`

**VHDL HDL Initialization**    `signal tmp: std_logic = '0';`

---

This is the preferred method to pass INIT values to the Xilinx place-and-route tools.

3. To set a register value using the `synthesis` attribute, add the attribute to the register in the source code or the constraint file, and specify the INIT value as a string:

---

**Verilog**    reg [3:0] rst\_cntr /\* synthesis INIT="1" \*/;

---

**VHDL**    attribute INIT: string;  
              attribute INIT of rst\_cntr : signal is "1";

---

**SDC**    define\_attribute {i:rst\_cntr} INIT {"1"}

---

4. To specify different INIT values for each register bit on a bus, do the following:
  - Set `syn_preserve` on the register as described in step 1, so that it is not optimized away. You can now either use the HDL specification or set an attribute.
  - To specify the values using the HDL specification, use the syntax as shown in the following examples:

---

**Verilog HDL Bus Initialization**    reg [7:0] address\_reg = 8'hff;

---

**VHDL HDL Bus Initialization**    signal q: std\_logic\_vector  
(11 downto 0) := X"755";

---

- To specify the value with the INIT attribute in the `sdc` constraint file, set INIT values for the individual register bits on the bus. Specify the register using the `i:` prefix, with periods as hierarchy separators.

The following specifies INIT values for individual bits of `rst_cntr`, which is part of the `init_attrver` module, under the top-level module:

```
define_attribute {i:init_attrver.rst_cntr[0]} INIT {"0"}  
define_attribute {i:init_attrver.rst_cntr[1]} INIT {"1"}  
define_attribute {i:init_attrver.rst_cntr[2]} INIT {"0"}  
define_attribute {i:init_attrver.rst_cntr[3]} INIT {"1"}
```

5. Synthesize the design.

The tool forward-annotates the values to the Xilinx P&R tool in the EDIF netlist. Note that the INIT value is forward-annotated as is (as an integer, not binary). You must ensure that the value is specified in the correct format for P&R.

If the register is an asynchronous output register with an initial value, the mapper preserves the initial value and packs the register into the block RAM.

# Generating the RAM MMI File

The tool can be used to generate memory map information (MMI) file for block RAM. The MMI file is supported by the UpdateMEM utility of the Xilinx Vivado tool and provides a way to modify the contents of the RAM without having to rerun synthesis and place and route. The MMI file uses XML format, which contains a physical description of the block RAM and includes placement information for the RAM resources.

To generate the RAM MMI file:

1. Specify a supported device for your database. For example:

```
% database load db0 -autocreate -technology HAPS-80
```

2. Apply the `syn_ram_write_mem` attribute on the RAM in the source code. For example:

```
reg [data_width-1:0] mem [2**addr_width-1:0]
/* synthesis syn_ram_write_mem = 1 */;
```

You can apply this attribute only on RAM objects, otherwise a warning is generated.

3. Run compile and map.

4. Export the files required to run place and route. For example:

```
export vivado -path par
```

5. Launch the Xilinx Vivado tool. For example:

```
launch vivado -script ./par/run_vivado_bitstream.tcl
```

6. Backannotate the place-and-route information. For example:

```
database apply_state -backannotate -dcp par/test.dcp
```

7. Export the files required to run place and route. For example:

```
export vivado -path par
```

A separate MMI file is created for each RAM instance and can be found in the place-and-route directory automatically created after the place-and-route run. For example—`par/mmifiles/ram_0.mmi`.

8. Launch the Xilinx Vivado UpdateMEM utility to update the \*.bit file with the new memory content, using the `updatemem` command. For example:

```
/global/snps_apps/vivado_2017.2.1/Vivado/2017.2/bin/updatemem  
-meminfo ./par/mmifiles/ram_0.mmi -data ./wrc.mem -proc ram_inst  
-bit ./par/test.bit -out ./par/test_updated.bit -force -debug
```

You can check the results of the run in the `updatemem.log` file:

```
#-----  
source /global/snps_apps/vivado_2017.2.1/Vivado/2017.2/scripts/updatemem/main.tcl -notrace  
Command: update_mem -meminfo ./par/mmifiles/ram_0.mmi -data ./wrc.mem -proc ram_inst -bit .  
Loading bitfile ./par/test.bit  
Loading data files...  
Updating memory content...  
Creating bitstream...  
Bitstream compression saved 102814304 bits.  
Bitstream compression saved 101776352 bits.  
Bitstream compression saved 102737952 bits.  
Bitstream compression saved 102543264 bits.  
Writing bitstream ./par/test_updated.bit...  
0 Infos, 0 Warnings, 0 Critical Warnings and 0 Errors encountered.  
update_mem completed successfully  
update_mem: Time (s): cpu = 00:00:33 ; elapsed = 00:00:39 . Memory (MB): peak = 1861.109 ;  
INFO: [Common 17-206] Exiting updatemem at Tue Sep 26 14:36:50 2017...
```

Make sure that the UpdateMEM job completes successfully.

For complete details about the Xilinx UpdateMEM utility, refer to the section called *Using UpdateMEM to Update BIT files with MMI and ELF Data* of the *Xilinx Vivado Design Suite User Guide*.

## Limitations

RAM modes and configurations for single-port, simple dual-port, and true dual-port RAM are supported. Currently, only BRAM18E/BRAM36E blocks that map to a single primitive are supported with the following limitations:

- Cascaded block RAM is not supported.
- Byte-Wide Write Enable mode is not supported.
- Block RAM using parity bits is not supported.
- ROM is not supported.
- The following will not be supported:
  - Asymmetric RAM
  - Block RAM that have been merged

- Block RAM that include encrypted IP

# Inferring Shift Registers

The software infers shift registers for certain architectures when you use the following procedure.

1. Set up the HDL code for the sequential shift components. See [Shift Register Examples, on page 249](#) for examples.

Note the following:

- The new component represents a set of three or more registers that can be shifted left (from a low address to a higher address).
- The contents of only one register can be seen at a time, based on the read address.
- For static components, the software only taps the output of the last register. The read address of the inferred component is set to a constant.

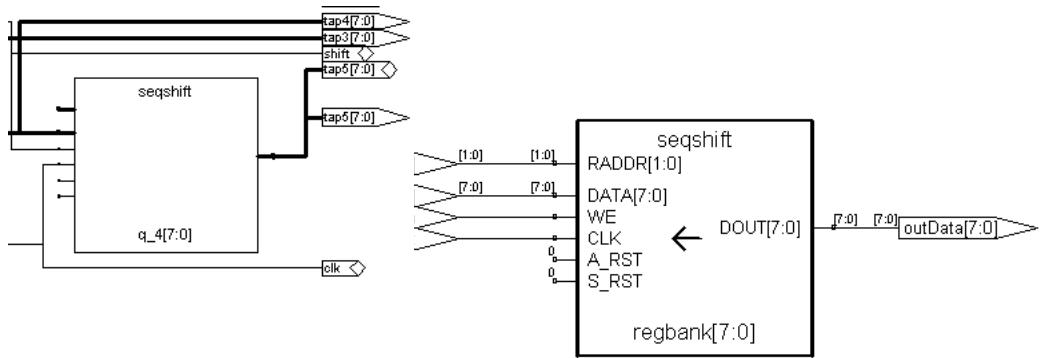
2. Specify the implementation style with the `syn_srlstyle` attribute.

| <b><code>syn_srlstyle</code> Value</b> | <b>Implemented as ...</b>                  |
|----------------------------------------|--------------------------------------------|
| <code>registers</code>                 | registers                                  |
| <code>select_srl</code>                | SRL16 primitives                           |
| <code>no_extractff_srl</code>          | SRL16 primitives without output flip-flops |

You can set the value globally or on individual registers. For example, if you do not want the components automatically mapped to shift registers, set it globally to `registers`. You can then override this with specific settings on individual registers as needed. See [syn\\_srlstyle, on page 811](#) for the syntax.

3. Run compile.

After compilation, the software displays the components as `seqShift` components in the RTL view. The following figure shows the components in the schematic view:



After mapping, the components are implemented as SRL16 primitives or registers, depending on the attribute values you set.

4. Check the results in the log file and the technology file.

The log file reports the shift registers and the number of registers packed in them.

## Shift Register Examples

The following VHDL and Verilog examples show a shift register with no resets. It has four 8-bit wide registers and a 2-bit wide read address. Registers shift when the write enable is 1.

### VHDL Shift Register Example

```

library IEEE;
use IEEE.std_logic_1164.all;

entity srltest is
    port (inData: std_logic_vector(7 downto 0);
          clk, en : in std_logic;
          outStage : in integer range 3 downto 0;
          outData: out std_logic_vector(7 downto 0)
        );
end srltest;

architecture rtl of srltest is
    type dataAryType is array(3 downto 0) of std_logic_vector(7
downto 0);
    signal regBank : dataAryType;

```

```
begin
    outData <= regBank(outStage);
    process(clk, inData)
        begin
            if (clk'event and clk = '1') then
                if (en='1') then
                    regBank <= (regBank(2 downto 0) & inData);
                end if;
            end if;
        end process;
    end rtl;
```

## Verilog Shift Register Example

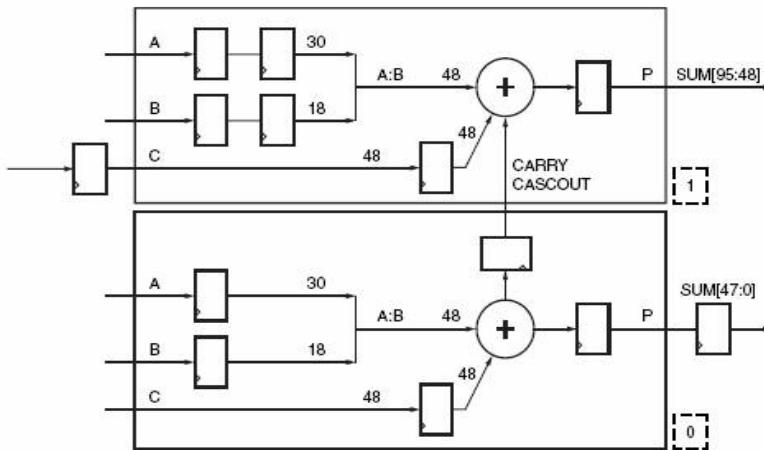
```
module test_srl(clk, enable, dataIn, result, addr);
    input clk, enable;
    input [3:0] dataIn;
    input [3:0] addr;
    output [3:0] result;
    reg [3:0] regBank[15:0];
    integer i;

    always @ (posedge clk) begin
        if (enable == 1) begin
            for (i=15; i>0; i=i-1) begin
                regBank[i] <= regBank[i-1];
            end
            regBank[0] <= dataIn;
        end
    end
    assign result = regBank[addr];
endmodule
```

# Inferring Wide Adders

You can map wide adder/subtractor structures to DSP48Es. Xilinx architectures let you use cascading DSP48Es and the CARRYCASOUT pin to support a structure with up to three pipeline registers with different synchronous control signals. It supports two or three signed/unsigned inputs (with carry/borrow).

The following shows the implementation of wide adders with one pipelined register and no pipelined registers as DSP48Es:

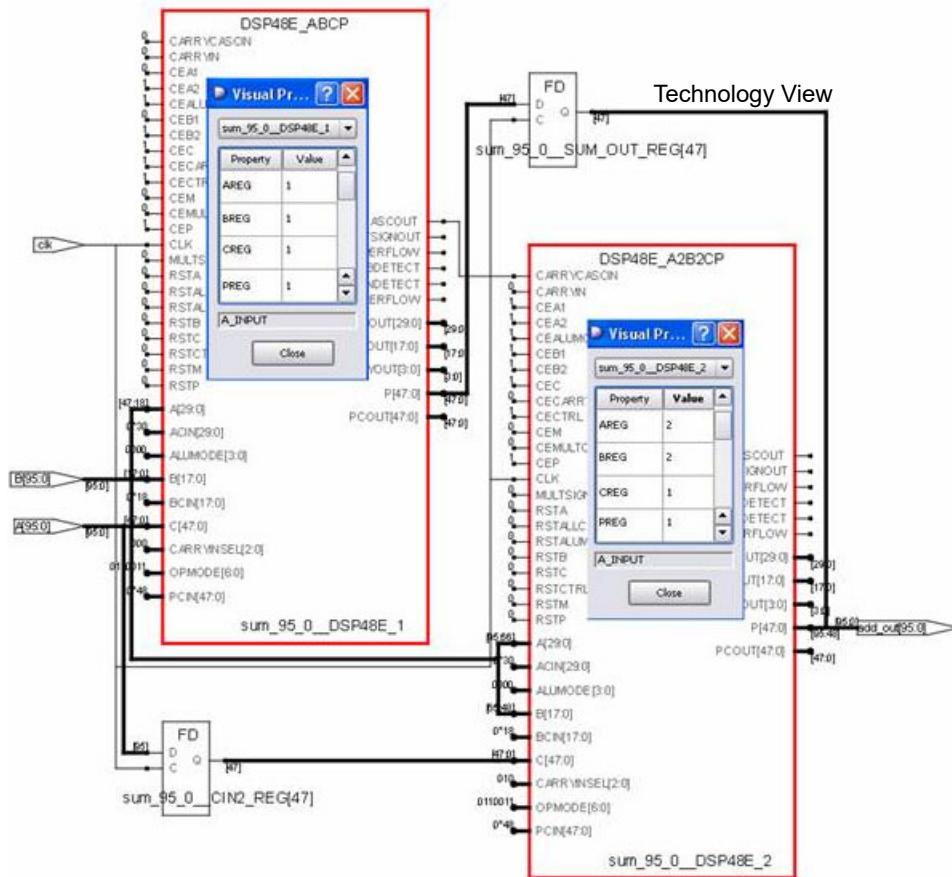
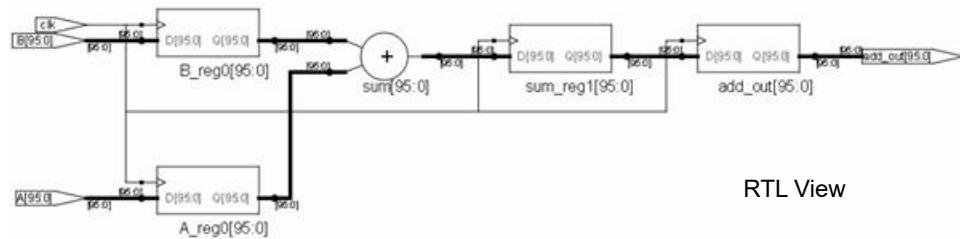


To automatically map to DSP48Es in the synthesis tools, do the following:

1. Make sure the structure you want to map conforms with these rules:
    - The adder/subtractor has fewer than 96 bits.
    - All registers share the same control signals (enables, clocks, reset). Registers with different control signals are mapped to the DSP48E, but they are kept outside the DSP48E.
    - The adder does not have a 48-bit input and a 49-bit output.
  2. Set `syn_dspstyle` to `dsp48`.
- You must set this attribute, or the tool does not infer a DSP48E. See [syn\\_dspstyle, on page 579](#) for the syntax for this attribute.
3. Synthesize the design.

If the structure has less than three pipelined registers, you see an advisory message in the log file, because three pipelined registers give the best performance.

The following is an example of how the synthesis tool maps an adder->register->register structure with 96-bit signed input and output to a DSP48E



# Defining State Machines

A finite state machine (FSM) is a piece of hardware that advances from state to state at a clock edge. The compiler recognizes and extracts the state machines from the HDL source code. For guidelines on setting up the source code, see the following:

- [Defining State Machines in Verilog](#), on page 253
- [Defining State Machines in VHDL](#), on page 254
- [Specifying FSMs with Attributes and Directives](#), on page 255

## Defining State Machines in Verilog

The compiler recognizes and automatically extracts state machines from the Verilog source code if you follow the coding guidelines listed below. The software attaches the `syn_state_machine` attribute to each extracted FSM.

For alternative ways to define state machines, see [Defining State Machines](#), on page 253.

Follow these Verilog coding guidelines:

- In Verilog, model the state machine with `case`, `cased`, or `casez` statements in `always` blocks. Check the current state to advance to the next state and then set output values. Do not use `if` statements.
- Always use a default assignment as the last assignment in the `case` statement, and set the state variable to '`bx`'. This is a "don't care" statement and ensures that the software can remove unnecessary decoding and gates.
- Make sure the state machines have a synchronous or asynchronous reset to set the hardware to a valid state after power-up, or to reset the hardware when you are operating.
- Specify explicit state values for states with parameter or 'define' statements. This is an example of a parameter statement that sets the current state to `2'h2`:

```
parameter state1 = 2'h1, state2 = 2'h2;  
...  
current_state = state2;
```

This example shows how to set the current state value with `define statements:

```
'define state1 2'h1
'define state2 2'h2
...
current_state = 'state2;
```

- Make state assignments using parameter with symbolic state names. Use parameter over `define, because `define is applied globally while parameter definitions are local. Local definitions make it easier to reuse common state names in multiple FSM designs, like RESET, IDLE, READY, READ, WRITE, ERROR, and DONE.

If you use `define to assign the names, you cannot reuse a state name because it has already been used in the global name space. To reuse the same names in this scenario, you have to use `undef and `define statements between modules to redefine the names. This method makes it difficult to probe the internal values of FSM state buses from a testbench and compare them to the state names.

## Defining State Machines in VHDL

The compiler recognizes and automatically extracts state machines from the VHDL source code if you follow the coding guidelines below. For alternative ways to define state machines, see [Defining State Machines, on page 253](#).

The following are VHDL guidelines for coding. The software attaches the `syn_state_machine` attribute to each extracted FSM.

- Use case statements to check the current state at the clock edge, advance to the next state, and set output values. You can also use if-then-else statements, but case statements are preferable.
- If you do not cover all possible cases explicitly, include a when others assignment as the last assignment of the case statement, and set the state vector to some valid state.
- If you create implicit state machines with multiple WAIT statements, the software does not recognize them as state machines.
- Make sure the state machines have a synchronous or asynchronous reset to set the hardware to a valid state after power-up, or to reset the hardware when you are operating.

- To choose an encoding style, attach the `syn_encoding` attribute to the enumerated type. The software automatically encodes your state machine with the style you specified.

## Specifying FSMs with Attributes and Directives

If your design has state machines, the software can extract them automatically with the FSM Compiler, or you can manually attach attributes to state registers to define them as state machines. See [Defining State Machines, on page 253](#) for other ways to specify FSMs.

The following steps show you how to manually attach attributes to define FSMs for extraction.

1. To determine how state machines are extracted, set attributes in the source code as shown in the following table:

| To ...                                                    | Attribute                        |
|-----------------------------------------------------------|----------------------------------|
| Specify a state machine for extraction and optimization   | <code>syn_state_machine=1</code> |
| Prevent state machines from being extracted and optimized | <code>syn_state_machine=0</code> |
| Prevent the state machine from being optimized away       | <code>syn_preserve=1</code>      |

For information about how to add attributes, see [Specifying Attributes and Directives, on page 462](#).

2. To determine the encoding style for the state machine, set the `syn_encoding` attribute in the source code or in the SCOPE window. For VHDL users there are alternative methods, described in the next step.

The FSM Compiler honors the `syn_encoding` setting. The different values for this attribute are briefly described here.

| Situation: If ...                           | syn_encoding Value                                                                            | Explanation                                                                                                                                                                                                                    |
|---------------------------------------------|-----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Area is important                           | sequential                                                                                    | One of the smallest encoding styles.                                                                                                                                                                                           |
| Speed is important                          | onehot                                                                                        | Usually the fastest style and suited to most FPGA styles.                                                                                                                                                                      |
| Recovery from an invalid state is important | safe, plus another style. For example:<br>/* synthesis<br>syn_encoding =<br>"safe, onehot" */ | Forces the state machine to reset in certain situations. For example, if an alpha particle hit in a hostile operating environment causes a spontaneous register change, you can use safe to reset the state machine.           |
| There are <5 states                         | sequential                                                                                    | Default encoding.                                                                                                                                                                                                              |
| A large output decoder follows the FSM      | sequential   gray                                                                             | Could be faster than onehot, even though the value must be decoded to determine the state. For sequential, more than one bit can change at a time; for gray, only one bit changes at a time, but more than one bit can be hot. |
| There are a large number of flip-flops      | onehot                                                                                        | Fastest style, because each state variable has one bit set, and only one bit of the state register changes at a time.                                                                                                          |

3. If you are using VHDL, you have two choices for defining encoding:

- Use syn\_encoding as described above, and enable the FSM compiler.
- Use the syn\_enum\_encoding attribute to define the states (sequential, onehot, gray, and safe). This attribute supports user-defined FSM encoding. For example:

```
attribute syn_enum_encoding of state_type : type is "001 010 101";
```

# Defining Black Boxes for Synthesis

Black boxes are predefined components for which the interface is specified, but whose internal architectural statements are ignored. They are used as place holders for IP blocks, legacy designs, or a design under development. If a black box is defined, the tool uses the interface information only.

This section discusses the following topics:

- [Using Black Boxes](#), on page 257
- [Instantiating Black Boxes and I/Os in Verilog](#), on page 260
- [Instantiating Black Boxes and I/Os in VHDL](#), on page 261
- [Adding Black Box Timing Constraints](#), on page 264
- [Adding Other Black Box Attributes](#), on page 267

## Using Black Boxes

Use black boxes to isolate portions of the design that are subject to frequent change or not yet stable. Black boxes offer more isolation and user control than compile points. Using black boxes also reduces runtime, because the black-boxed parts can be ignored, reducing design complexity and memory requirements, or can be run in parallel. Independence also means that tool issues can be isolated for debugging.

The following procedure describes how to use a bottom-up, black box flow:

1. Identify parts of the design for black-boxing.
  - Find candidates that are likely to be modified frequently, like ASIC memories early in the conversion process. Typical candidates for black-boxing are technology-specific primitives and macros, including I/Os; or user-designed macros whose functionality is defined in an input source where the place-and-route tool merges design netlists from different sources.
  - Do not black-box objects that have predefined black box definitions.
  - Make sure black boxes do not contain cross-module references (XMRs).
2. Automatically infer some black boxes with `auto_infer_blackbox`.

For example you might want to isolate ASIC memories that have not yet been converted for FPGAs. Remove the memories from the source files and set the `auto_infer_blackbox` option to 1. This automatically creates black boxes for the memories.

- Set these basic constraints for black boxes at the top level:

|                                   |                                                                                                               |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>syn_black_box</code>        | Identifies the black box at the top level.                                                                    |
| <code>disable_io_insertion</code> | Prevents the inference of I/O pads on the black box in addition to the top level.                             |
| <code>syn_resources</code>        | Use the block is large enough to affect partition. The entire black box must be partitioned in the same FPGA. |

- Set other constraints for black box timing and other information in the source code, as described in [Adding Black Box Timing Constraints, on page 264](#) and [Adding Other Black Box Attributes, on page 267](#).

You must add all other black box directives in the source code because information like timing models is specific to individual instances. This table summarizes the black box controls:

| <b>Black Box Directive</b>                               |                                                                           |
|----------------------------------------------------------|---------------------------------------------------------------------------|
| <code>syn_black_box</code>                               | Identifies the black box at the top level.                                |
| <b>Black Box Source Code Timing Directives</b>           |                                                                           |
| <code>syn_isclock</code>                                 | Specifies a clock port on a black box.                                    |
| <code>syn_tpd&lt;n&gt;</code>                            | Sets timing propagation for combinational delay through the black box.    |
| <code>syn_tsu&lt;n&gt;</code>                            | Defines timing setup delay required for input pins relative to the clock. |
| <code>syn_tco&lt;n&gt;</code>                            | Defines the timing clock to output delay through the black box.           |
| <b>Black Box Source Code Directives for Gated Clocks</b> |                                                                           |
| <code>syn_force_seq_prim</code>                          | Indicates that gated clocks should be fixed for this black box.           |
| <code>syn_gatedclk_clock_en</code>                       | Specifies the enable pin to be used in fixing the gated clocks.           |

`syn_gatedclk_clock_en_polarity` Indicates the polarity of the clock enable port on a black box so that the software can fix gated clocks.

### Black Box Pin Definitions in the Source Code

|                                 |                                                                      |
|---------------------------------|----------------------------------------------------------------------|
| <code>black_box_pad_pin</code>  | Indicates that a black box is an I/O pad for the rest of the design. |
| <code>black_box_tri_pins</code> | Indicates tristates on black boxes.                                  |

## 5. Run through the usual design flow.

When partitioning, make sure that the entire black box is partitioned into one FPGA.

The `syn_black_box` directive might not be honored in some cases:

- In a mixed design, if a black box is defined in one language at the top level but there is a description for it in another language, the tool can replace the declared black box with the description from the other language.
- If your source files include black box descriptions in `srs`, `ngc`, or `edf` formats, the tool uses these black box descriptions even if you have specified `syn_black_box` at the top level.

## 6. To ensure that the directive is honored even when other descriptions are available for the black box, use these methods:

- Set a `syn_black_box` directive on the module or entity in the HDL file that contains the description, not at the top level. The contents will be black-boxed.
- Use the compiler constraints file (`.cdc`) to set a comprehensive black box directive at the top level. For example, the following sets the `syn_black_box` attribute on all architectures of the sub entity in the `MyLib` library:

```
define_directive {v:MyLib.sub} {syn_black_box} {1}
```

- If you have `srs`, `ngc`, or `edf` description for a module you want to define as a black box, remove the description.

## 7. Use the black box output for place-and-route runs:

- Include the synthesis output of black box runs in the place-and-route directory, in the same directory as the top-level netlist.

- Run place and route. The tool finds the black box and inserts the information when it runs place and route.

## Instantiating Black Boxes and I/Os in Verilog

Verilog black boxes for macros and I/Os come from two sources: commonly-used or vendor-specific components that are predefined in Verilog macro libraries, or black boxes that are defined in another input source like a schematic. For information about instantiating black boxes in VHDL, see [Instantiating Black Boxes and I/Os in VHDL](#), on page 261.

The following process shows you how to instantiate both types as black boxes.

1. To instantiate a predefined Verilog module as a black box:
  - Select the library file with the macro you need. Files are named *technology.v*. Most vendor architectures provide macro libraries that redefine the black boxes for primitives and macros.
  - Make sure the library macro file is the first file in the source file list for your project.
2. To instantiate a module that has been defined in another input source as a black box:
  - Create an empty macro that only contains ports and port directions.
  - Put the `syn_black_box` directive just before the semicolon in the module declaration.

```
module myram (out, in, addr, we) /* synthesis syn_black_box */;
    output [15:0] out;
    input [15:0] in;
    input [4:0] addr;
    input we;
endmodule
```

- Make an instance of the stub in your design.
- Compile the stub along with the module containing the instantiation of the stub.
- To simulate with a Verilog simulator, you must have a functional description of the black box. To make sure the software ignores the functional description and treats it as a black box during the compile

and map stages, use the `translate_off` and `translate_on` constructs. For example:

```
module adder8(cout, sum, a, b, cin);
  // Code that you want to synthesize
  /* synthesis translate_off */
  // Functional description.
  /* synthesis translate_on */
  // Other code that you want to synthesize.
endmodule
```

3. To instantiate a vendor-specific (black box) I/O that has been defined in another input source:
  - Create an empty macro that only contains ports and port directions.
  - Put the `syn_black_box` directive just before the semicolon in the module declaration.
  - Specify the external pad pin with the `black_box_pad_pin` directive, as in this example:

```
module BBDLHS(D,E,GIN,GOUT,PAD,Q)
  /* synthesis syn_black_box black_box_pad_pin="PAD"
```

  - Make an instance of the stub in your design.
  - Compile the stub along with the module containing the instantiation of the stub.
4. Add timing constraints and attributes as needed. See [Adding Black Box Timing Constraints, on page 264](#) and [Adding Other Black Box Attributes, on page 267](#).

## Instantiating Black Boxes and I/Os in VHDL

VHDL black boxes for macros and I/Os come from two sources: commonly-used or vendor-specific components that are predefined in VHDL macro libraries, or black boxes that are defined in another input source like a schematic. For information about instantiating black boxes in VHDL, see [Instantiating Black Boxes and I/Os in Verilog, on page 260](#).

The following process shows you how to instantiate both types as black boxes.

1. To instantiate a predefined VHDL macro (for a component or an I/O),

- Select the library file with the macro you need. Files are named *family.vhd*. Most vendor architectures provide macro libraries that predefined the black boxes for primitives and macros.
- Add the appropriate library and use clauses to the beginning of your design units that instantiate the macros.

```
library family;
use family.components.all;
```

2. To create a black box for a component from another input source:

- Create a component declaration for the black box.
- Declare the `syn_black_box` attribute as a boolean attribute.
- Set the attribute to true.

```
library synplify;
use synplify.attributes.all;
entity top is
    port (clk, rst, en, data: in bit; q: out bit);
end top;

architecture structural of top is
component bbox
    port(Q: out bit; D, C, CLR: in bit);
end component;

attribute syn_black_box of bbox: component is true;
...
```

- Instantiate the black box and connect the ports.

```
begin
my_bbox: bbox port map (
    Q => q,
    D => data,
    C => clk,
    CLR => rst);
```

- To simulate with a VHDL simulator, you must have the functional description of a black box. To make the software ignores the functional description and treats it as a black box during the compile and map stages, use the `translate_off` and `translate_on` constructs. For example:

```

architecture behave of ram4 is
begin
    -- synthesis translate_off
    stimulus: process (clk, a, b)
        -- Functional description
    end process;
    -- synthesis translate_on

    -- Other source code you WANT synthesized

```

3. To create a black box I/O for an I/O defined in another input source:
  - Create a component declaration for the I/O.
  - Declare the `black_box_pad_pin` attribute as a string attribute.
  - Set the attribute value on the component to be the external pin name for the pad.

```

library synplify;
use synplify.attributes.all;
...

component mybuf
    port(O: out bit; I: in bit);
end component;
attribute black_box_pad_pin of mybuf: component is "I";

```

- Instantiate the pad and connect the signals.

```

begin
data_pad: mybuf port map (
    O => data_core,
    I => data);

```

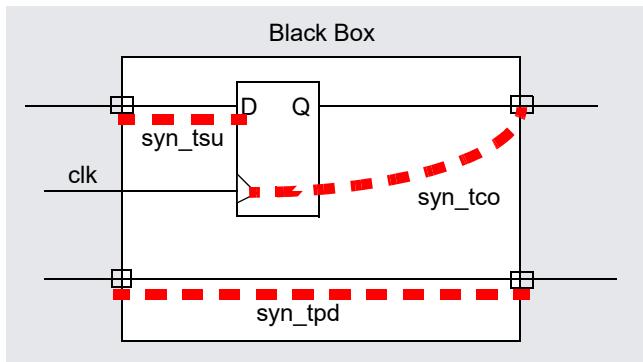
4. Add timing constraints and attributes.

See [Adding Black Box Timing Constraints, on page 264](#) and [Adding Other Black Box Attributes, on page 267](#).

## Adding Black Box Timing Constraints

A black box does not provide the software with any information about internal timing characteristics. You must characterize black box timing accurately, because it can critically affect the overall timing of the design. To do this, you add constraints in the source code or in the SCOPE interface.

You attach black box timing constraints to instances that have been defined as black boxes. There are three black box timing constraints, syn\_tpd, syn\_tsu, and syn\_tco. There are additional attributes for black box pins and black boxes with gated clocks; see [Adding Other Black Box Attributes](#), on page 267.



1. Define the instance as a black box, as described in [Instantiating Black Boxes and I/Os in Verilog](#), on page 260 or [Instantiating Black Boxes and I/Os in VHDL](#), on page 261.
2. Determine the kind of constraint for the information you want to specify:

| To define ...                                      | Use ... |
|----------------------------------------------------|---------|
| Propagation delay through the black box            | syn_tpd |
| Setup delay (relative to the clock) for input pins | syn_tsu |
| Clock-to-output delay through the black box        | syn_tco |

3. In VHDL, use the following syntax for the constraints.
  - Use the predefined attributes package by adding this syntax

```
library synplify;
use synplify.attributes.all;
```

In VHDL, you must use the predefined attributes package. For each directive, there are ten predeclared constraints in the attributes package, from *directive\_name1* to *directive\_name10*. If you need more constraints, declare the additional constraints using integers greater than 10. For example:

```
attribute syn_tco11 : string;
attribute syn_tco12 : string;
```

- Define the constraints in either of these ways:

|                |                                                       |
|----------------|-------------------------------------------------------|
| VHDL<br>syntax | attribute <i>attributeName&lt;n&gt;</i> : "att_value" |
|----------------|-------------------------------------------------------|

|                           |                                                                                           |
|---------------------------|-------------------------------------------------------------------------------------------|
| Verilog-style<br>notation | attribute <i>attributeName&lt;n&gt;</i> of <i>bbox_name</i> :<br>component is "att_value" |
|---------------------------|-------------------------------------------------------------------------------------------|

The following table shows the appropriate syntax for att\_value. See the attribute descriptions ([Attributes and Directives Summary, on page 473](#)) for complete syntax information.

| <b>Attribute</b>              | <b>Value Syntax</b>                        |
|-------------------------------|--------------------------------------------|
| <code>syn_tsu&lt;n&gt;</code> | <code>bundle -&gt; [!]clock = value</code> |
| <code>syn_tco&lt;n&gt;</code> | <code>[!]clock -&gt; bundle = value</code> |
| <code>syn_tpd&lt;n&gt;</code> | <code>bundle -&gt; bundle = value</code>   |

• <*n*> is a numerical suffix.

• *bundle* is a comma-separated list of buses and scalar signals, with no intervening spaces. For example, A,B,C.

• ! indicates (optionally) a negative edge for a clock.

• *value* is in ns.

The following is an example of black box attributes, using VHDL signal notation:

```

architecture top of top is
component rcf16x4z port(
    ad0, ad1, ad2, ad3 : in std_logic;
    di0, di1, di2, di3 : in std_logic;
    wren, wpe : in std_logic;
    tri : in std_logic;
    do0, do1, do2, do3 : out std_logic;
end component

attribute syn_tpd1 of rcf16x4z : component is
    "ad0,ad1,ad2,ad3 -> do0,do1,do2,do3 = 2.1";
attribute syn_tpd2 of rcf16x4z : component is
    "tri -> do0,do1,do2,do3 = 2.0";
attribute syn_tsu1 of rcf16x4z : component is
    "ad0,ad1,ad2,ad3 -> ck = 1.2";
attribute syn_tsu2 of rcf16x4z : component is
    "wren,wpe,do0,do1,do2,do3 -> ck = 0.0";

```

4. In Verilog, add the directives as comments, as shown in the following example.

```

module ram32x4 (z, d, addr, we, clk)
    /* synthesis syn_black_box
    syn_tpd1="addr[3:0]->z[3:0]=8.0"
    syn_tsu1="addr[3:0]->clk=2.0"
    syn_tsu2="we->clk=3.0" */;
    output [3:0] z;
    input [3:0] d;
    input [3:0] addr;
    input we;
    input clk;
endmodule

```

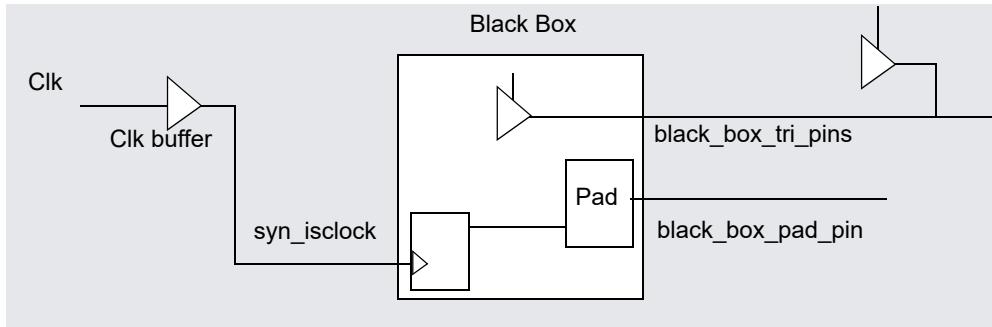
5. To add black box attributes to the fdc file, use the following syntax, and save the constraint file.

**define\_attribute v:{blackboxModule} attribute<n> {attributeValue}**

6. Synthesize the design, and check black box timing.

## Adding Other Black Box Attributes

Besides black box timing constraints, you can also add other attributes to define pin types on the black box or define gated clocks. You cannot use the attributes for all technologies. Check the [Attributes and Directives Summary, on page 473](#) for syntax details about the attributes mentioned below.



1. To specify that a clock pin on the black box has access to global clock routing resources, use `syn_isclock`.

The tool inserts BUFGs.

2. To specify that the software need not insert a pad for a black box pin, use `black_box_pad_pin`.
3. To define a tristate pin so that you do not get a mixed driver error when there is another tristate buffer driving the same net, use `black_box_tri_pins`.
4. To ensure consistency between synthesized black box netlist names and the names generated by third party tools or IP cores, use the following attributes:
  - `syn_edif_bit_format`
  - `syn_edif_scalar_format`

## Instantiating Xilinx Macros and Cores

See the following for more information:

- [Specifying Xilinx Macros](#), on page 268
- [Instantiating CoreGen Cores](#), on page 270
- [Instantiating Virtex PCI Cores](#), on page 271

## Specifying Xilinx Macros

The synthesis tool provides Xilinx macro libraries that you can use to instantiate components like I/Os, I/O pads, gates, counters, and flip-flops. Using the macros from these libraries allows you to perform a subsequent simulation run without changing your code.

1. To use the Verilog macro library, review the `unisim.v` macro library in the `installDirectory/lib/xilinx` directory for the available macros.
2. To use a VHDL library, do the following:
  - Review the `unisim.vhd` macro library in the `installDirectory/lib/xilinx` directory to check the macros that are available.
  - Add the corresponding library and use clauses to the beginning of the design units that instantiate the macros, as in the following example:

```
library unisim;
use unisim.vcomponents.all;
```

You do not need to add the macro library files to your the source files for your project.

3. Instantiate the macro component in your design.
4. To instantiate an I/O pad with different I/O standards, do the following:
  - Specify the macro library as described in the first two steps.
  - Instantiate the I/O pad component in your design. You can instantiate IBUF, IBUFG, OBUF, OBUFT, and IOBUF components.
  - In the source files, define the generic or parameter values for the I/O standard. Use an `IOSTANDARD` generic/parameter to specify the I/O standard you want. Refer to the Xilinx documentation for a list of supported `IOSTANDARD`s. For certain pad types, you can also specify the output slew rate (`SLEW`) and output drive strength (`DRIVE`). See [OBUF Instantiation Example](#), on page 269 for an example.

## OBUF Instantiation Example

The following examples show the declaration of OBUF in macro library files:

|      |                                                                                                                                                                                                                                                                       |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VHDL | <pre>component OBUF   generic (     IOSTANDARD : string := "default";     SLEW : string := "SLOW";     DRIVE : integer := 12   );   port (     O : out std_logic;     I : in std_logic;   ); end component; attribute syn_black_box of OBUF : component is true</pre> |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

|         |                                                                                                                                                                     |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Verilog | <pre>module OBUF(O, I); /* synthesis syn_black_box */ parameter IOSTANDARD="default"; parameter SLEW="SLOW"; parameter DRIVE=12; output O; input I; endmodule</pre> |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

To use the macro libraries to instantiate I/O pad types, define the generic/parameter values in the Verilog or VHDL source files. The following examples show how to instantiate OBUF pads with an I/O standard value of LVCMOS2, an output slew value of FAST, and an output drive strength of 24.

|      |                                                                                                                                                                       |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VHDL | <pre>Data : OBUF   generic map (     IOSTANDARD =&gt; "LVCMOS2",     SLEW =&gt; "FAST",     DRIVE =&gt; 24   )   port map (     O =&gt; o1,     I =&gt; i1   );</pre> |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

|         |                                                                                                                                    |
|---------|------------------------------------------------------------------------------------------------------------------------------------|
| Verilog | <pre>OBUF Data(.O(o1), .I(i1)); defparam Data.IOSTANDARD = "LVCMOS2"; defparam Data.SLEW = "FAST"; defparam Data.DRIVE = 24;</pre> |
|---------|------------------------------------------------------------------------------------------------------------------------------------|

---

The resulting EDIF file contains the following, which corresponds to the instantiations:

```
instance (
    rename dataZ0 "data")
    (viewRef PRIM (cellRef OBUF (libraryRef VIRTEX)))
    (property iostandard (string "LVCMOS2"))
    (property slew (string "FAST"))
    (property drive (integer 24))
)
```

## Instantiating CoreGen Cores

Predesigned IP cores save on design effort and improve performance. The process for handling IP cores is slightly different for CoreGen and Virtex PCI cores. The following procedure describes how to instantiate a CoreGen module. For Virtex PCI cores, see [Instantiating Virtex PCI Cores, on page 271](#).

1. Use the Xilinx CORE generator to create structural EDIF netlists and generate timing and resource usage information for synthesis.
  - For legacy cores, generate a single flat `edf` netlist file.
  - For newer cores, generate a top-level flat `edn` or `edf` netlist file that instantiates `ndf` files for each hierarchical level in the design.
2. Open the synthesis software, and add the generated files (`edf` only for legacy cores; `edn` or `edf` and `ndf` for newer cores) to your project.
3. Define the core as a black box by adding the `syn_black_box` attribute to the module definition line, or by using the Coregen v file. The following is an example of the attribute:

```
module ram64x8(din, addr, we, clk, dout)/* synthesis syn_black_box */;
    input [7:0] din;
    input [5:0] addr;
    input we, clk;
    output [7:0] dout;
endmodule;
```

4. Make sure the bus format matches the bus format in the core generator, using the `syn_edif_bit_format` and `syn_edif_scalar_format` directives if needed.

```
module ram64x8(din, addr, we, clk, dout)
/* synthesis syn_black_box syn_edif_bit_format = "%u<%i>"
syn_edif_scalar_format ="%u" */;
```

5. Instantiate the black box in the module or architecture.

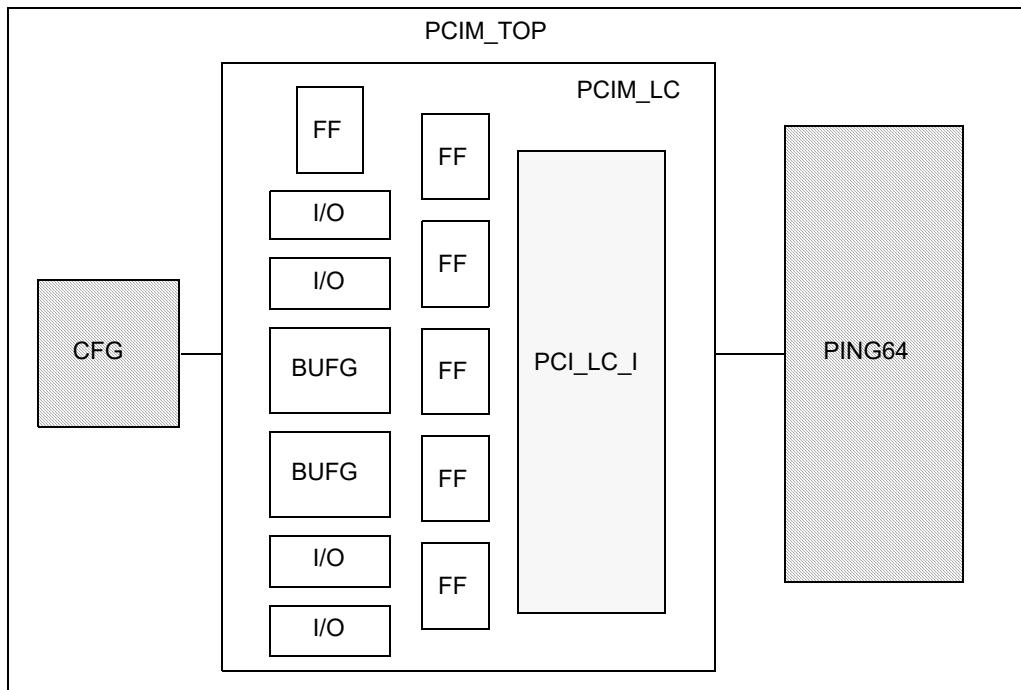
```
ram64x8 r1(din, addr, we, clk, dout);
```

6. Synthesize the design.

If you supplied structural EDIF netlists, the software optimizes the design based on the information in the structural netlists. The generated reports contain the optimization information.

## Instantiating Virtex PCI Cores

To instantiate Virtex PCI cores, you can use either a top-down or bottom-up methodology. This figure shows a design that is used in the explanations of both methodologies, below.



## Bottom-Up Method

The bottom-up method synthesizes lower-level modules first. The synthesized modules are then treated as black boxes and synthesized at the next level. The following procedure refers to the figure shown above.

1. Synthesize the user-defined application (**PING64**) by itself.
  - Make sure that the Disable I/O Insertion option is on.
  - Specify the `syn_edif_bit_format = "%u<%i>"` and `syn_edif_scalar_format = "%u"` attributes. These attributes ensure that the EDIF bus names match the Xilinx upper-case, angle bracket style bus names and the Xilinx upper-case net names, respectively.

The software generates an EDIF file for this module.

2. Synthesize the top-level module that contains the PCI core, with the Disable I/O Insertion option enabled and the EDIF naming attributes described in the previous step. Use the following files to synthesize:
  - The top-level module (PCIM\_LC) file, with the PCI core (PCI\_LC\_I) declared as a black box with the `syn_black_box` attribute.
  - A black box file for the core (PCI\_LC\_I), that only contains information about the PCI core ports. This file is the source file that is generated for simulation, not the `ngo` file.
  - The appropriate synthesis Virtex file (`installDirectory/lib/xilinx`) that contains module definitions of the I/O pads in the top-level module, PCIM\_LC.

The software generates an EDIF file for this module.

3. Synthesize the top level (PCIM\_TOP) with Disable I/O Insertion off. Use the following files:
  - The source file for CFG.
  - A black box file for PING64.
  - A black box file for PCIM\_LC.
  - A top-level file that contains black box declarations for PING64 and PCIM\_LC.

The software generates an EDIF file for the top level.

4. Place and route using the Xilinx `ngo` file for the core, and the three EDIF files generated from synthesis: one for each of the modules PING64 and PCIM\_LC, and the top-level EDIF file. Select the top-level EDIF file when you run place-and-route.

## Top-down Methodology

The top-down method instantiates user application blocks and synthesizes all the source files in one synthesis run. This method can result in a smaller, faster design than with the bottom-up method, because the tool can do cross-boundary optimizations. The following procedure refers to the design shown in the previous figure.

1. Create your own configuration file for your application model (CFG).
2. Edit the top-level source file to do the following:
  - Instantiate your application block (PING64) in the top-level source file.

- Add the ports from your application.
3. Add the appropriate synthesis Virtex file (*installDirectory/lib/xilinx*) to the project. This file contains module definitions of the I/O pads in the PCIM\_LC module.
  4. Specify the top-level file in the project.
  5. Synthesize your design with the following files:
    - Virtex module definition file (previous step)
    - Source files for top-level design, user application (PING64), PCIM\_LC, and CFG
    - Simulation wrapper file for PCI core
- The software generates an EDIF file for the top level.
6. Place and route the design using the top-level EDIF file from synthesis and the Xilinx ngo file for the PCI core.

## Packing Registers for Xilinx I/Os

When a register drives an input or output, you might want to pack it in an IOB instead of a CLB, as in these cases:

- When the chip interfaces with another, and you have to minimize the register-to-output or input-to-register delay.
- When there are limited CLB resources, and packing the registers in an IOB can free up some resources.

To pack registers in an IOB, you set the syn\_useioff attribute.

1. To globally embed all the registers into IOBs, attach the syn\_useioff attribute to the module in one of these ways:
  - Add the attribute in the SCOPE window, attaching it to the module, architecture, or the top level. Check the Enable box, set the Attribute column to syn\_useioff, the Object column to <global>, and the attribute value to 1. The constraint file syntax looks like this:

```
define_global_attribute syn_useioff 1
```
  - To add the attribute in the Verilog source code, add this syntax to the top level:

```
module global_test(d, clk, q) /* synthesis syn_useioff = 1 */;
```

- To add the attribute in the VHDL source code, add this syntax to the top level architecture declaration:

```
architecture rtl of global_test is
attribute syn_useioff : boolean;
attribute syn_useioff of rtl : architecture is true;
```

For details about attaching attributes using the SCOPE interface and in the source code, see [Attributes and Directives Summary, on page 473](#).

When set globally, all boundary registers and (OE) registers associated with the data registers are marked with the Xilinx IOB property. This property is forward annotated in the EDIF netlist and used by the Xilinx place-and-route tools to determine how the registers are packed. All marked registers are packed in the corresponding IOBs.

2. To apply syn\_useioff to individual registers or ports, use one of these methods:
  - Add the attribute in the SCOPE window, attaching it to the ports you want to pack, and set the attribute value to 1. The resulting constraint file syntax looks like this:

```
define_attribute {p:q[3:0]} syn_useioff 1
```

- To add the attribute in the Verilog source code, add this syntax:

```
module test is (d, clk, q);
  input [3:0] d;
  input clk;
  output [3:0] q /* synthesis syn_useioff = 1 */;
  reg q;
```

- To add the attribute in the VHDL source code, add syntax as shown inside the entity for the local port:

```
entity test is
  port (d : in std_logic_vector(3 downto 0);
        clk : in std_logic;
        q : out std_logic_vector(3 downto 0));
  attribute syn_useioff : boolean;
  attribute syn_useioff of q : signal is true;
end test;
```

The software attaches the IOB property as described in the previous step, but only to the specified flip-flops. Packing for ports and registers

without the attribute is determined by timing preferences. If a register is to be packed into an IOB, the IOB property is attached and forward annotated. If it is to be packed into a CLB, the IOB property is not forward annotated.

In Virtex designs where the synthesis software duplicates OE registers, setting the `syn_useioff` attribute on a boundary register only enables the associated OE register for packing. The duplicate is not packed, but placed in a CLB. The packed registers are used for data path, and the CLB registers are used for counter implementation.

In Virtex designs where a shift register is at a boundary edge and the `syn_useioff` attribute is enabled, the software extracts only the initial or final SRL16 shift register from the LUT for packing. The shift register that is implemented in the technology view is smaller because of the extraction.

3. If you set multiple `syn_useioff` attributes at different levels of the design, the tool uses the most specific setting (highest priority).

This table summarizes `syn_useioff` priority settings, from the highest priority (register) to the lowest (global):

| I/O Type | <code>syn_useioff</code> Value | Description                                                                           |
|----------|--------------------------------|---------------------------------------------------------------------------------------|
| Register | 1                              | Packs registers into the I/O pad cells, overriding port or global specifications.     |
|          | 0                              | Does not pack registers into I/O pad cells, overriding port or global specifications. |
| Port     | 1                              | Packs registers into the I/O pad cells, overriding any global specification.          |
|          | 0                              | Does not pack registers into I/O pad cells, overriding any global specification.      |
| Global   | 1                              | Packs registers into the I/O pad cells.                                               |
|          | 0                              | Does not pack registers into I/O pad cells.                                           |

The `syn_useioff` attribute is supported in the compile point flow.

## Inserting Xilinx I/Os and Specifying Pin Locations

By default, the synthesis tools automatically insert I/Os for inputs, outputs, and bidirectionals (such as IBUFs and OBUFs). You can control this with the Disable IO Insertion option. You can also insert I/Os manually by instantiating them.

Whether you use the automatic or manual method, you can specify pin locations for the I/Os with the `xc_loc` attribute. By default, or if no location is specified, the Xilinx tool assigns pin locations automatically.

The following provide details:

- [Assigning Pin Locations for Automatically Inserted Xilinx I/Os](#), on page 277
- [Manually Inserting Xilinx I/Os in Verilog](#), on page 280
- [Manually Inserting Xilinx I/Os in VHDL](#), on page 281

### Assigning Pin Locations for Automatically Inserted Xilinx I/Os

The synthesis tool automatically inserts the I/Os (unless you have checked Disable IO Insertion in the Device tab of the Implementation Options dialog box). The following procedure shows you how to assign pin locations for automatically inserted I/Os in a Verilog or VHDL design.

1. Create a new top-level module or entity and instantiate it in your Verilog or VHDL design.

This module/entity holds I/O placement information. Creating this lets you keep your vendor-specific information separate from the rest of your design. Your original design remains technology-independent.

For example, this is a Verilog counter definition:

```
module cnt4 (cout, out, in, ce, load, clk, rst);  
// Counter definition  
endmodule
```

Create a top-level module that instantiates your design:

```
module cnt4_xilinx (cout, out, in, ce, load, clk, rst);
```

2. Specify inputs and outputs.

- If you do not want to specify locations, specify the inputs or outputs as usual, and the place-and-route tool determines placement automatically. The following is an example of Verilog inputs in the top-level module:
 

```
input ce, load, clk, rst;
```
- Optionally, specify I/O locations in the new top-level module, by setting the `xc_loc` attribute. You can specify the `xc_loc` attribute in the Attribute panel of the SCOPE spreadsheet, as shown below.

|   | Enabled                             | Object Type | Object   | Attribute | Value   | Val Type | Description    |
|---|-------------------------------------|-------------|----------|-----------|---------|----------|----------------|
| 2 | <input checked="" type="checkbox"/> | port        | in1[2:0] | xc_loc    | P2,P3,P | string   | Port placement |
| 3 | <input checked="" type="checkbox"/> | port        | p:out1   | xc_loc    | R1      | string   | Port placement |
| 4 | <input checked="" type="checkbox"/> |             |          |           |         |          |                |

Attributes

Alternatively, you can specify it in the HDL files, as described in [Manually Inserting Xilinx I/Os in Verilog, on page 280](#) and [Manually Inserting Xilinx I/Os in VHDL, on page 281](#). See `xc_loc`, on page 894 in the *Reference Manual* for syntax details.

The following Verilog code includes `xc_loc` attributes that specify the following locations: cout at A1, out in the top left (TL) of the chip, in[3] at P20, in[2] at P19, in[1] at P18, and in[0] at P17.

```
output cout /* synthesis xc_loc="A1" */;
output [3:0] out /* synthesis xc_loc="TL" */;
input [3:0] in /* synthesis xc_loc="P20,P19,P18,P17" */;
```

3. Instantiate the top-level module or entity with the placement information you specified in your design. For example:

```
cnt4 my_counter (.cout(cout), .out(out), .in(in),
    .ce(ce), .load(load), .clk(clk), .rst(rst));
endmodule
```

4. Compile, map, place, and route the design.

During compile and map, the tool automatically insert I/Os for inputs, outputs, and bidirectionals (such as IBUFs and OBUFs). The place-and-route tool automatically selects locations for I/Os with no `xc_loc` attribute defined. If you specified `xc_loc` settings, they are honored.

## VHDL Automatic I/O Insertion Example

```
library synplify;
entity cnt4 is
    port (cout: out bit;
          output: out bit_vector (3 downto 0);
          input: in bit_vector (3 downto 0);
          ce, load, clk, rst: in bit );
end cnt4;

architecture behave of cnt4 is
begin
-- Behavioral description of the counter.
end behave;

-- New top level entity, created specifically
-- to place I/Os for Xilinx. This entity is typically
-- in another file, so that your original
-- design stays untouched and technology independent.

entity cnt4_xilinx is
    port (cout: out bit;
          output: out bit_vector (3 downto 0);
          input: in bit_vector (3 downto 0);
          ce, load, clk, rst: in bit );

-- Place a single I/O for cout at location A1.
attribute xc_loc : string;
attribute xc_loc of cout: signal is "A1";

-- Place all bits of "output" in the
-- top-left of the chip.
attribute xc_loc of output: signal is "TL";

-- Place input(3) at P20, input(2) at P19,
-- input(1) at P18, and input(0) at P17
attribute xc_loc of input: signal is "P20, P19, P18, P17";

-- Let Xilinx place the rest of the inputs.
end cnt4_xilinx;

-- New top level architecture instantiates your design.
architecture structural of cnt4_xilinx is
-- Component declaration for your entity.
```

```
component cnt4
    port (cout: out bit;
          output: out bit_vector (3 downto 0);
          input: in bit_vector (3 downto 0);
          ce, load, clk, rst: in bit );
end component;
begin
    -- Instantiate your VHDL design here:
    my_counter: cnt4 port map (cout, output, input,
                                ce, load, clk, rst);
end structural;
```

## Manually Inserting Xilinx I/Os in Verilog

To insert a Xilinx I/O manually, you must instantiate a black box macro for that I/O from the Xilinx library file. Then, either assign it a location or have the Xilinx tool automatically select one for it. Follow these steps:

1. Add the *installDirectory/lib/xilinx/unisim.v* macro library file to the *top* of the source files list for your design.
2. Create instances of I/Os by instantiating a black box in your Verilog source code.

These black boxes are empty Verilog module descriptions, taken from the Xilinx macro library you specified in step 1. You can stop at this step, and the Xilinx tool will automatically assign locations for the I/Os you specified.

To continue on and specify pin locations, do the following:

1. Create a new top-level module and instantiate your Verilog design.
2. Add the *installDirectory/lib/xilinx/unisim.v* macro library file to the *top* of the source files list for your project.
3. Create instances of I/Os by instantiating a black box in your Verilog source code.
4. Specify I/O locations by adding the `xc_loc` attribute to the I/Os.

See [Verilog Manual I/O Insertion Example](#), on page 281 for an example of the code. The Xilinx tool honors any locations assigned with the `xc_loc` attribute, and automatically selects locations for any remaining I/Os without definitions.

## Verilog Manual I/O Insertion Example

```

module cnt4 (cout, out, in, ce, load, clk, rst);
/* Your counter definition goes here, */
endmodule
/* Create a top level to place I/Os specifically
   for Xilinx. Any top level pins which do not have
   I/Os will be automatically inserted */
module cnt4_xilinx(cout, out, in, ce, load, clk, rst);
output [3:0] out;
output cout;
input [3:0] in;
input ce, load, clk, rst;wire [3:0] out_c, in_c;
wire cout_c;

/* The xc_loc attribute can be added right after the
   instance name like that shown below, or right before
   the semicolon. */

IBUF i3 /* synthesis xc_loc="P20" */ (.O(in_c[3]), .I(in[3]));
IBUF i2 /* synthesis xc_loc="P19" */ (.O(in_c[2]), .I(in[2]));
IBUF i1 /* synthesis xc_loc="P18" */ (.O(in_c[1]), .I(in[1]));
IBUF i0 /* synthesis xc_loc="P17" */ (.O(in_c[0]), .I(in[0]));

OBUF o3 /* synthesis xc_loc="TL" */ (.O(out[3]), .I(out_c[3]));
OBUF o2 /* synthesis xc_loc="TL" */ (.O(out[2]), .I(out_c[2]));
OBUF o1 /* synthesis xc_loc="TL" */ (.O(out[1]), .I(out_c[1]));
OBUF o0 /* synthesis xc_loc="TL" */ (.O(out[0]), .I(out_c[0]));

OBUF cout_p /* synthesis xc_loc="BL" */ (.O(cout), .I(cout_c));
cnt4 it(.cout(cout_c), .out(out_c), .in(in_c),
       .ce(ce), .load(load), .clk(clk), .rst(rst));
endmodule

```

## Manually Inserting Xilinx I/Os in VHDL

To insert an I/O manually and then use automatic location assignment, do the following:

1. Add the corresponding library and use clauses to the beginning of your design units that instantiate the macros.

```

library unisim;
use unisim.vcomponents.all;

```

The Xilinx `unisim.vhd` macro library is always visible in the synthesis tool, so do not add this library file to the source files list for your project. To

see which design units are available, use a text editor to view the file located in the *installDirectory/lib/xilinx* directory. Do not edit this file in any way.

2. Create instances of I/Os by instantiating a black box in your Verilog source code.

These black boxes are empty Verilog module descriptions, taken from the Xilinx macro library you specified in step 1. You can stop at this step, and the Xilinx tool will automatically assign locations for the I/Os you specified.

To continue and specify pin locations, do the following:

1. Create a new top-level module and instantiate your VHDL design.
2. Instantiate the Xilinx I/Os.
3. Add the appropriate library and use clauses to the beginning of design units that instantiate the I/Os.

```
library unisim;
use unisim.vcomponents.all;
```

See the source code in [VHDL Manual I/O Insertion Example, on page 282](#) for an example.

4. To specify I/O locations, add the `xc_loc` attribute to the I/O instances for which you want to specify the locations.

If you leave out the `xc_loc` attribute, the Xilinx place-and-route tool will choose the locations.

## VHDL Manual I/O Insertion Example

The following example is a behavioral D flip-flop with instantiated data input I/O. The other ports will have synthesized I/Os.

```
library ieee, synplify;
use synplify.attributes.all;
use ieee.std_logic_1164.all;
-- Library and use clauses for access to the Xilinx Macro Library.
library unisim;
use unisim.vcomponents.all;
```

```
entity place_example is
    port (q: out std_logic;
          d, clk: in std_logic );
end place_example;

architecture behave of place_example is
    signal dz: std_logic;

attribute xc_loc of I1: label is "P3";
begin
    I1: IBUF port map (I=>d,O=>dz);

process (clk) begin
    if rising_edge(clk) then
        q<=dz;
    end if;
end process;

end behave;
```

# Working with Buffers

By default, the synthesis tools do not automatically infer buffers. If you want the tools to infer buffers, you must use attributes. The following procedures describe how to work with different kinds of buffers:

- [Inferring Buffers](#), on page 284
- [Inferring BUFGDLL Clock Buffers](#), on page 285
- [Inferring Regional Clock Buffers](#), on page 286
- [Instantiating Special I/O Standard Buffers](#), on page 287
- [Inserting BUFG Global Buffers](#), on page 300

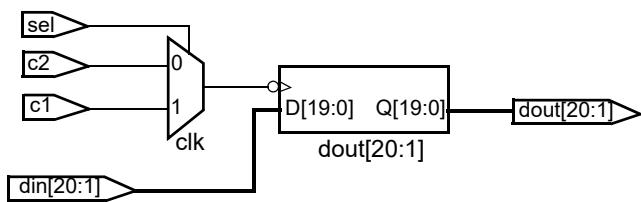
## Inferring Buffers

Attach attributes to infer buffers. The following procedure describes how to infer BUFGMUX, IBUFDS, IBUFGDS, OBUFDS, OBUFTDS, and IOBUFDS components, using `syn_insert_buffer` and `syn_diff_io`.

1. To infer BUFGMUX components, do the following:
  - Attach the `syn_insert_buffer` attribute to the mux instance. If you need information on how to do this, see [Specifying Attributes and Directives](#), on page 462.
  - Set the attribute value to `bufgmxu`. When you set this value, the tool infers a `BUFGMUX_1` if the muxed clock operates on the negative edge; otherwise it infers a `BUFGMUX`. If you do not specify this value, by default the tool infers the LUT that drives the BUFG.

```
module
bufgmxu_1(c1,c2,sel,din,d
out);
input c1,c2,sel;
input [20:1] din;
output reg [20 : 1] dout;
wire clk;

assign clk = sel ? c1 :
```



For details about the `syn_insert_buffer` syntax, see [syn\\_insert\\_buffer, on page 653](#).

2. To infer IBUFDS, IBUFGDS, OBUFDS, OBUFTDS, and IOBUFDS differential buffers, do the following:
  - Attach the `syn_diff_io` attribute to the inputs of the buffer.
  - Set the value to 1 or true.

For details about the `syn_diff_io` syntax, see [syn\\_diff\\_io, on page 560](#).

The `syn_diff_io` attribute is supported in the compile point flow.

## Inferring BUFGDLL Clock Buffers

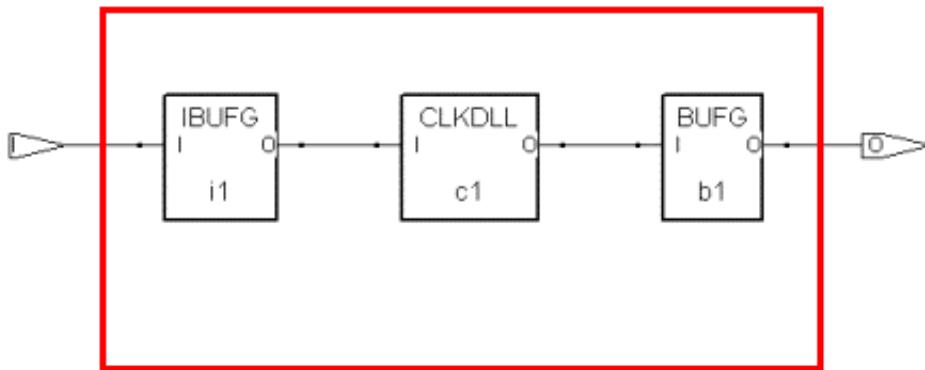
The tool can infer BUFGDLL clock buffers, which include the CLKDLL primitive. BUFGDLL consists of an IBUFG followed by a CLKDLL (Clock Delay Locked Loop) followed by a BUFG. To use this CLKDLL primitive, you must specify the `xc_clockbuftype` attribute.

The following steps show you how to add the attribute in HDL or through the SCOPE interface.

1. To specify the `xc_clockbuftype` attribute in Verilog, add the attribute as shown in this example.

```
module test(d, clk, rst, q);
    input [1:0] d;
    input clk /* synthesis xc_clockbuftype = "BUFGDLL" */, rst;
    output [1:0] q;
    //other coding
```

The software infers a buffer as shown in the following figure.



2. To specify the attribute in VHDL, add the attribute as shown in this example.

```
entity test_clkbuftype is
    port (d: in std_logic_vector(3 downto 0);
          clk, rst : in std_logic;
          q : out std_logic_vector(3 downto 0)
        );
    attribute xc_clockbuftype of clk : signal is "BUFGDLL";
end test_clkbuftype
```

3. To specify the attribute in the SCOPE window, use the Attributes panel to add the **xc\_clockbuftype** attribute to a port.
4. Check the output EDIF netlist for text like the following:

```
(instance clk_ibuf (viewRef PRIM (cellRef BUFGDLL (libraryRef VIRTEX) ))
```

## Inferring Regional Clock Buffers

The regional clock buffer (BUFR) is a special buffer used to connect clock nets in the same region and adjacent regions, independent of the global clock tree. The BUFR can drive the I/O logic and logic resources (CLB, block RAM, and DSP) in existing and adjacent clock regions. They can be driven by clock-capable pins, local interconnects, gigabit transceiver (GT) clocks, and Mixed-Mode Clock Manager (MMCM) clocks. You can also use BUFR as a clock divider in low-power and low-skew operations.

By default, the synthesis tools do not automatically infer BUFR. If you want the tools to infer regional clock buffers, you must use the `syn_insert_buffer` attribute, as described below.

1. To infer BUFR primitives, do the following:
  - Apply the `syn_insert_buffer` attribute on a port or net.
  - Set the attribute value to BUFR. For details about the syntax, see [syn\\_insert\\_buffer, on page 653](#).
2. Check the log file.
  - Check that the output of the BUFR used on the clock net is defined as the derived clock.
  - Check that there is a timing report for the clock signal.
  - Check BUFR inference. If the BUFR is inferred, you see a message like this one:

Clock Buffers:

Inserting Clock buffer on net `clk1_en`, Inserting Clock buffer for port `clk`,

If it is not inferred, there is a warning message:

Warning: BUFR not inserted on net <name>

- Check the resource utilization for the number of BUFRs connected in the region.

Cell usage:

BUFR 2 uses

## Instantiating Special I/O Standard Buffers

The software supports all the I/O standard buffers, like HSTL\_\*, CTT, AGP, PC133\_\*, PC166\_\*, etc. You can either instantiate these primitives directly, or specify them with the `xc_padtype` attribute.

1. To instantiate I/O buffers, use code like the following to specify them.

```
module inst_padtype(a, b, clk, rst, en, bidir, q) ;
  input [0:0] a, b;
  input clk, rst, en;
  inout bidir;
  output [0:0] q;
```

```
reg [0:0] q_int;
wire a_in, q_in;
IBUF_AGP i1 (.O(a_in), .I(a) ) ;
IOBUF_CTT i2 (.O(q_in), .IO(bidir) , .I(q_int), .T(en) ) ;
OBUF_F_12 o1 (.O(q), .I(q_in) ) ;

always @ (posedge clk or posedge rst)
  if (rst)
    q_int <= 1'b0;
  else
    q_int <= a_in & b;

endmodule
```

2. To specify the I/O buffers with the `xc_padtype` attribute, either add the attribute in the Attributes panel of the constraints window or add it to the HDL code, as shown below:

```
module inst_padtype(a, b, clk, rst, en, bidir, q) ;
  input [0:0] a /* synthesis xc_padtype = "IBUF_AGP" */, b;
  input clk, rst, en;
  inout bidir /* synthesis xc_padtype = "IOBUF_CTT" */;
  output [0:0] q /* synthesis xc_padtype = "OBUF_F_12" */;

  reg [0:0] q_int;

  assign q = bidir;
  assign bidir = en ? q_int : 1'bz;
  always @ (posedge clk or posedge rst)
    if (rst)
      q_int <= 1'b0;
    else
      q_int <= a & b;
endmodule
```

See [xc\\_padtype](#), on page 901 for syntax details.

# Specifying RLOCs

RLOCs are relative location constraints. They let you control placement in critical areas, thus improving performance.

See the following for more information on specifying RLOCs:

- [Specifying RLOCs with xc\\_map, xc\\_rloc, and xc\\_uset, on page 289](#)
- [Specifying RLOCs and RLOC\\_ORIGINS with the synthesis Attribute, on page 290](#)

## Specifying RLOCs with xc\_map, xc\_rloc, and xc\_uset

With this method, you use three attributes, xc\_map, xc\_rloc, and xc\_uset, to define RLOCs. As with other attributes, you can define them in the source code, or in the SCOPE window. You can also specify RLOCs directly, as described in [Specifying RLOCs and RLOC\\_ORIGINS with the synthesis Attribute, on page 290](#).

1. Create the modules you want to constrain, and attach the xc\_map attribute with a value of lut to them.  
The modules can have up to four inputs, but only one output.
2. Instantiate the modules you created at a higher hierarchy level.
3. Group the instances together with the xc\_uset attribute and specify the relative locations of instances in the group with the xc\_rloc attribute.

This example shows the Verilog code for the top-level CLB that includes the 4-input module in the previous example:

```
module xor9(z, a);
    output z;
    input [8:0] a;
    wire x03, x47;
    fmap_xor4 x03 /*synthesis xc_uset="SET1" xc_rloc="R0C0.S0" */
        (z03, a[0], a[1], a[2], a[3]);
    fmap_xor4 x47 /*synthesis xc_uset="SET1" xc_rloc="R0C0.S0" */
        (z47, a[4], a[5], a[6], a[7]);
    hmap_xor3 zz /*synthesis xc_uset="SET1" xc_rloc="R0C0.S1" */
        (z, z03, z47, a[8]);
endmodule
```

4. Create a top-level design and instantiate your design.

## Specifying RLOCs and RLOC\_ORIGINs with the synthesis Attribute

You can specify RLOCs and RLOC\_ORIGINs with the synthesis attribute, and then pass them to the Xilinx P&R tools. Alternatively, you can specify RLOCs using the three attributes described in *Specifying RLOCs with xc\_map, xc\_rloc, and xc\_uset, on page 289*.

1. In the source code, use the synthesis attribute to specify the RLOC and RLOC\_ORIGIN values:

---

|         |                                                                                                                                                                     |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Verilog | /* synthesis RLOC_ORIGIN="X0Y2" RLOC="X0Y0" */;                                                                                                                     |
| VHDL    | attribute RLOC_ORIGIN : string;<br>attribute RLOC_ORIGIN of behave : architecture is "X0Y2";<br>attribute RLOC : string;<br>attribute RLOC of q : signal is "X0Y0"; |

---

2. To specify different RLOC and RLOC\_ORIGIN values for bits on a bus, do the following:

- Specify the RLOC\_ORIGIN for the Verilog module or VHDL architecture in the source file. See step 1 for the syntax.
- Define RLOCs for the individual register bits as constraints in the sdc file. Do not define RLOCs for individual bits in the source code, or you will get a Xilinx ISE error.

```
define_attribute {i:tmp[0]} RLOC {"X3Y0"}  
define_attribute {i:tmp[1]} RLOC {"X2Y0"}  
define_attribute {i:tmp[2]} RLOC {"X1Y0"}  
define_attribute {i:tmp[3]} RLOC {"X0Y0"}
```

3. Synthesize the design.

The tool forward-annotates the values to the place-and-route tool.

## CHAPTER 4

# Optimizing Synthesis Results

---

This chapter contains information on various optimization techniques to improve performance results.

- [Tips for Optimization](#), on page 292
- [Optimizing Fanout](#), on page 297
- [Pipelining and Retiming](#), on page 303
- [Preserving Objects from Being Optimized Away](#), on page 311
- [Sharing Resources](#), on page 317
- [Inserting I/Os](#), on page 318

# Tips for Optimization

The software automatically makes efficient trade-offs to achieve the best results. However, you can optimize your results by using the appropriate control parameters. This section describes general design guidelines for optimization. The topics have been categorized as follows:

- [General Optimization Tips](#), on page 292
- [Optimizing for Area](#), on page 293
- [Optimizing for Timing](#), on page 295

## General Optimization Tips

This section contains general optimization tips that are not directly area or timing-related. For area optimization tips, see [Optimizing for Area](#), on page 293. For timing optimization, see [Optimizing for Timing](#), on page 295.

- In your source code, remove any unnecessary priority structures in timing-critical designs. For example, use CASE statements instead of nested IF-THEN-ELSE statements for priority-independent logic.
- When using VHDL, specify a UNISIM library using the following syntax:

```
library unisim;
use unisim.vcomponents.all;
```

Remove any other package files with user-defined UNISIM primitives.

- If your design includes safe state machines, use the syn\_encoding attribute with a value of safe. This ensures that the synthesized state machines never lock in an illegal state.
- For FSMs coded in VHDL using enumerated types, use the same encoding style (syn\_enum\_encoding attribute value) on both the state machine enumerated type and the state signal. This ensures that there are no discrepancies in the type of encoding to negatively affect the final circuit.
- Make sure that the source code supports inferencing or instantiation by using architecture-specific resources like memory blocks.

- Some designs benefit from hierarchical optimization techniques. To enable hierarchical optimization on your design, set the `syn_hier` attribute to `firm`.
- For accurate results with timing-driven synthesis, explicitly define clock frequencies with a constraint, instead of using a global clock frequency.
- For critical paths, attach the `xc_fast` attribute to the I/Os.
- To ensure that frequency constraints from register to output pads are forward annotated to the place-and-route tools, add default `input_delay` and `output_delay` constraints of 0.0 in the synthesis tool. The synthesis tool forward-annotates the frequency constraints as PERIOD constraints (register-to-register) and OFFSET constraints (input-to-register and register-to-output). The place-and-route tools use these constraints.
- Run successive place-and-route iterations with progressively tighter timing constraints to get the best results possible.

## Optimizing for Area

This section contains information on optimizing to reduce area. Optimizing for area often means larger delays, and you will have to weigh your performance needs against your area needs to determine what works best for your design. For tips on optimizing for performance, see [Optimizing for Timing, on page 295](#). General optimization tips are in [General Optimization Tips, on page 292](#).

- Increase the fanout limit when you set options (Tools->Edit Options->Fanout Guide). A higher limit means less replicated logic and fewer buffers inserted during synthesis, and a consequently smaller area. In addition, as P&R tools typically buffer high fanout nets, there is no need for excessive buffering during synthesis. See [Setting Fanout Limits, on page 297](#) for more information.
- Enable the Resource Sharing option when you set options (Tools->Edit Options->Resource Sharing). With this option checked, the software shares hardware resources like adders, multipliers, and counters wherever possible, and minimizes area. This is a global setting, but you can also specify resource sharing on an individual basis for lower-level modules. See [Sharing Resources, on page 317](#) for details.

- For designs with large FSMs, use the gray or sequential encoding styles, because they typically use the least area. For details, see [\*Specifying FSMs with Attributes and Directives, on page 255\*](#).
- If you are mapping into a CPLD and do not meet area requirements, set the default encoding style for FSMs to sequential instead of onehot. For details, see [\*Specifying FSMs with Attributes and Directives, on page 255\*](#).
- For small CPLD designs (less than 20K gates), you might improve area by using the `syn_hier` attribute with a value of flatten. When specified, the software optimizes across hierarchical boundaries and creates smaller designs.

## Optimizing for Timing

This section contains information on optimizing to meet timing requirements. Optimizing for timing is often at the expense of area, and you will have to balance the two to determine what works best for your design. For tips on optimizing for area, see [Optimizing for Area, on page 293](#). General optimization tips are in [General Optimization Tips, on page 292](#).

- Use realistic design constraints, about 10 to 15 percent of the real goal. Over-constraining your design can be counter-productive because you can get poor implementations. Typically, you set timing constraints like clock frequency, clock-to-clock delay paths, I/O delays, register I/O delays and other miscellaneous path delays. Use clock, false path, and multi-cycle path constraints to make the constraints realistic.
- Enable the Retiming option. This optimization moves registers into I/O buffers if this is permitted by the technology and the design. However, it may add extra registers when clouds of logic are balanced across more than one register-to-register timing path. Extra registers are only added in parallel within the timing path and only if no extra latency is added by the additional registers. For example, if registers are moved across a 2x1 multiplexer, the tool adds two new registers to accommodate the select and data paths.

You can set this option globally or on specific registers. When it is enabled, it automatically enables pipelining as well.

- Enable the Pipelining option. With this optimization enabled, the tool moves existing registers into a ROM or multiplier. Unlike retiming, it does not add any new logic. Pipelining reduces routing and delay and the extra instance delay of the external register by moving it into the ROM or multiplier and making it a built-in register.
- Select a balanced fanout constraint. A large constraint creates nets with large fanouts, and a low fanout constraint results in replicated logic. See [Setting Fanout Limits, on page 297](#) for information about setting limits and using the syn\_maxfan attribute. You can use this in conjunction with the syn\_replicate attribute that controls register duplication and buffering.
- Control register duplication and buffering criteria with the syn\_replicate attribute. The tool automatically replicates registers during optimization, and you can use this attribute globally or locally on a specific register to turn off register duplication. See [Controlling Buffering and Replication, on page 299](#) for a description. Use syn\_replicate in conjunction with the syn\_maxfan attribute that controls fanout.

- If the critical path goes through arithmetic components, try disabling Resource Sharing. You can get faster times at the expense of increased area, but use this technique carefully. Adding too many resources can cause longer delays and defeat your purpose.
- If the P&R and synthesis tools report different critical paths, use a timing constraint with the `-route` option. With this option, the software adds route delay to its calculations when trying to meet the clock frequency goal. Use realistic values for the constraints.
- For FSMs, use the onehot encoding style, because it is often the fastest implementation. If a large output decoder follows an FSM, gray or sequential encoding could be faster.
- For designs with black boxes, characterize the timing models accurately, using the `syn_tpd`, `syn_tco`, and `syn_tso` directives.
- If you see warnings about feedback muxes being created for signals when you compile your source code, make sure to assign set/resets for the signals. This improves performance by eliminating the extra mux delay on the input of the register.
- Make sure that you pass your timing constraints to the place-and-route tools, so that they can use the constraints to optimize timing.
- If performance and quality of results (QoR) are not essential to the application (as with early prototyping and “what if” scenarios), use the Fast Synthesis option. For details, refer to [Running Fast Compiler Mode \(Standard Compiler\), on page 86](#).

The Synthesis Strategy fast option reduces the amount and number of mapper optimizations performed so that you get faster synthesis runtimes. Once the design has been evaluated with fast synthesis strategy, the mapper optimization effort can be returned to its normal, default level for optimum routability and congestion reduction techniques.

# Optimizing Fanout

You can optimize your results with attributes and directives, some of which are specific to the technology you are using. Similarly, you can use specify objects or hierarchy that you want to preserve during synthesis. For a complete list of all the directives and attributes, see the *Attribute Reference Manual*. This section describes the following:

- [Setting Fanout Limits](#), on page 297
- [Controlling Buffering and Replication](#), on page 299
- [Inserting BUFG Global Buffers](#), on page 300

## Setting Fanout Limits

Optimization affects net fanout. If your design has critical nets with high fanout, you can set fanout limits. You can only do this in certain technologies. For details specific to individual technologies, see the *Reference Manual*.

1. To set a global fanout limit for the whole design, do either of the following:
  - Either specify a maximum limit with the option set `maxfan` command, or select Tools-> Edit Options and set a value for Fanout Guide.
  - Apply the `syn_maxfan` attribute to the top-level view or module.

The value sets the maximum number of fanouts for a given driver, and applies to all the nets in the design. Select a balanced fanout value. A large constraint creates nets with large fanouts, and a low fanout constraint results in replicated or buffered logic. Both extremes affect routing and design performance. The right value depends on your design. The same value of 32 might result in fanouts of 11 or 12 and large delays on the critical path in one design or in excessive replication in another design.

The software uses the value as a soft limit, or a guide. It traverses the inverters and buffers to identify the fanout, and tries to ensure that all fanouts are under the limit by replicating or buffering where needed (see [Controlling Buffering and Replication](#), on page 299 for details). However, the tool does not respect the fanout limit absolutely; it ignores the limit if the limit imposes constraints that interfere with optimization.

2. To override the global fanout guideline and set a soft fanout limit at a local level, set the `syn_maxfan` attribute on modules, views, or non-primitive instances.

These limits override the global limits for that object. However, the local limits are also soft limits, and nets can be replicated or buffered, as described in [Controlling Buffering and Replication, on page 299](#).

| Attribute specified on ...              | Effect                                                   |
|-----------------------------------------|----------------------------------------------------------|
| Module or view                          | Soft limit for the module; overrides the global setting. |
| Non-primitive instance                  | Soft limit; overrides global and module settings         |
| Clock nets or asynchronous control nets | Soft limit.                                              |

- |  |  |
| --- | --- |
| Attribute specified on ... | Effect |
- |  |  |
| --- | --- |
| Module or view | Soft limit for the module; overrides the global setting. |
- |  |  |
| --- | --- |
| Non-primitive instance | Soft limit; overrides global and module settings |
- |  |  |
| --- | --- |
| Clock nets or asynchronous control nets | Soft limit. |
3. To set a hard or absolute limit, set the `syn_maxfan` attribute on a port, net, register, or primitive instance.

Fanouts that exceed the hard limit are buffered or replicated, as described in [Controlling Buffering and Replication, on page 299](#).

4. To preserve net drivers from being optimized, attach the `syn_keep` or `syn_preserve` attributes.

For example, the software does not traverse a `syn_keep` buffer (inserted as a result of the attribute), and does not optimize it. However, the software can optimize implicit buffers created as a result of other operations; for example, it does not respect an implicit buffer created as a result of `syn_direct_enable`.

5. Check the results of buffering and replication in the following:

- The log file. The log file reports the number of buffered and replicated objects and the number of segments created for the net.
- The HDL Analyst views. The software might not follow DRC rules when buffering or replicating objects, or when obeying hard fanout limits.

## Controlling Buffering and Replication

To honor fanout limits (see [Setting Fanout Limits, on page 297](#)) and reduce fanout, the software either replicates components or adds buffers. The tool uses buffering to reduce fanout on input ports, and uses replication to reduce fanout on nets driven by registers or combinatorial logic. The software first tries replication, replicating the net driver and splitting the net into segments. This increases the number of register bits in the design. When replication is not possible, the software buffers the signals. Buffering is more expensive in terms of intrinsic delay and resource consumption. The following table summarizes the behavior.

| Replicates When ...                                 | Creates Buffers When ...                                                                                                                                                                                                                                                 |
|-----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>syn_maxfan</code> is set on a register output | <code>syn_maxfan</code> is set on input ports in Microsemi ProASIC (500K), ProASIC PLUS (PA) and ProASIC3 families.                                                                                                                                                      |
| <code>syn_replicate</code> is 1                     | <code>syn_replicate</code> is 0.<br>Note that the <code>syn_replicate</code> attribute must be used in conjunction with the <code>syn_maxfan</code> attribute for Microsemi families. The <code>syn_replicate</code> attribute is used only to turn off the replication. |
|                                                     | <code>syn_maxfan</code> is set on a port/net that is driven by a port or I/O pad                                                                                                                                                                                         |
|                                                     | The net driver has a <code>syn_keep</code> or <code>syn_preserve</code> attribute                                                                                                                                                                                        |
|                                                     | The net driver is not a primitive gate or register                                                                                                                                                                                                                       |

You can control whether high fanout nets are buffered or replicated, using the techniques described here:

- To use buffering instead of replication, set `syn_replicate` with a value of 0 globally, or on modules or registers. The `syn_replicate` attribute prevents replication, so that the software uses buffering to satisfy the fanout limit. For example, you can prevent replication between clock boundaries for a register that is clocked by `clk1` but whose fanin cone is driven by `clk2`, even though `clk2` is an unrelated clock in another clock group.
- To specify that high-fanout clock ports should not be buffered, set `syn_noclockbuf` globally, or on individual input ports. Use this if you want to save clock buffer resources for nets with lower fanouts but tighter constraints.

- Inverters merged with fanout loads increase fanout on the driver during placement and routing. A distinction is made between a keep buffer created as the result of the `syn_keep` attribute being applied by the user (explicit keep buffer) and a keep buffer that exists as the result of another attribute (implicit keep buffer). For example, the `syn_direct_enable` attribute inserts a keep buffer. When a `syn_maxfan` attribute is applied to the output of an explicit keep buffer, the signal is buffered (the keep buffer is not traversed so that the driver is not replicated). When the `syn_maxfan` attribute is applied to the output of an implicit keep buffer, the keep buffer is traversed and the driver is replicated.
- You can handle extremely large clock fanout nets by inserting a global buffer (BUFG) in your design. A global buffer reduces delay for a large fanout net and can free up routing resources for other signals. See [Inserting BUFG Global Buffers, on page 300](#).
- Turn off buffering and replication entirely, by setting `syn_maxfan` to a very high number, like 1000.

## Inserting BUFG Global Buffers

Depending on available resources after global buffers have been allocated for critical nets like gated clocks, the tool automatically inserts BUFG global buffers on a maximum of two non-critical control nets, like reset or enable. Use the `syn_auto_insert_bufg` attribute to prevent automatic BUFG insertion on control signals with heavy loads, or to control which nets gets buffered.

1. Understand the criteria that determine automatic BUFG insertion:
  - To be considered for BUFG insertion, the net must be a high-fanout net, which means that the fanout must exceed the maximum fanout set by the global fanout limit.
  - Buffer insertion for critical nets like gated clocks must use less than fifty percent of the BUFG resources; more than fifty percent of the BUFG resources must still be available after buffer insertion for critical nets.
  - The values for the user-specified `syn_insert_buffer` and `syn_global_buffers` attributes are taken into account.
  - The control signal nets must have mixed loads (control and datapath signals).

- Candidate nets must drive loads that are more than 50% control signals, either synchronous or asynchronous. If a net drives both the control and datapath signals, the BUFG is not inserted.
  - BUFGs are inserted on the two nets with the highest fanouts. The tool only buffers up to two BUFG components on non-clock nets for the entire design, per Xilinx guidelines.
2. Map the design and check the log file for informational FX1035 messages like this one:

```
@N: FX1035 | Inserting BUFG on non-clock net dut_module_inst.dut_rst  
(fanout: 6060). To prevent insertion of a BUFG on this net, copy  
and paste the string  
'define_attribute {n:dut_module_inst.dut_rst} {syn_auto_insert_bufg} {0}'  
into your FDC file.
```

By default the tool inserts BUFGs automatically, according to the criteria described in step 1.

3. To prevent a BUFG from being automatically inserted, do the following.
- Copy and paste the `syn_auto_insert_bufg` attribute syntax with the 0 value from the note into an fdc file, as directed. You could also directly add `syn_auto_insert_bufg` attributes with a value of 0 to the fdc file for nets that you do not want to buffer. See [syn\\_auto\\_insert\\_bufg, on page 520](#) for the syntax for the attribute.
  - Consider the effect of other buffer settings when deciding whether to prevent buffer insertion with `syn_auto_insert_bufg`:

## Global Settings

|                                  |                                                                                                                                                                                                                                                                                    |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>syn_global_buffers</code>  | Global attribute that sets an upper limit on the number of buffers that can be inserted, including BUFGs inserted with <code>syn_auto_insert_bufg</code> . Clock net buffering takes a higher priority than buffering for control signals with <code>syn_auto_insert_bufg</code> . |
| <code>fanout_limit</code> Option | Global fanout limit set for the design. Once the limit is exceeded, the tool uses other techniques besides buffering to reduce fanout, like replicating the driver or splitting the net into segments.                                                                             |

## Local Settings

---

|                                |                                                                                                                                                 |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>syn_insert_buffer</code> | Local attribute set on nets and ports, without checking total BUFG resource usage.                                                              |
| <code>syn_maxfan</code>        | Local attribute for a port, net, or register that overrides the global fanout limit.                                                            |
| <code>syn_noclockbuf</code>    | When attached to a port or net of a hierarchy that will not be dissolved, this attribute turns off the automatic use of clock buffer resources. |

For information about using other buffer attributes, see [Working with Buffers, on page 284](#). For information about controlling fanout, see [Optimizing Fanout, on page 297](#).

- Rerun mapping, using the updated constraint file. The tool does not insert BUFGs for the constrained nets.

Based on your knowledge of the available resources and the insertion criteria listed in step 1, you can selectively prevent automatic BUFG insertion on the nets.

4. Alternatively, you can use `define_global_attribute` to set `syn_auto_insert_bufg` to 0 globally, and then specify the `syn_insert_buffer` attribute locally to insert buffers on selected nets.

```
define_global_attribute syn_auto_insert_bufg {0}
define_attribute syn_insert_buffer enable_1 {BUFG}
```

# Pipeline and Retiming

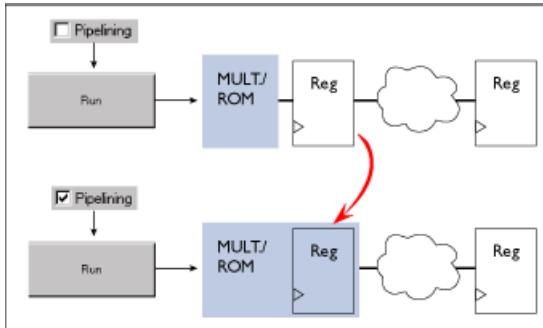
Pipeline and retiming are techniques that improve design performance by moving design objects. Pipeline splits logic into stages to speed up processing, and retiming moves objects from critical paths to improve timing while still maintaining the logical behavior. See the following topics for details:

- [Pipeline the Design](#), on page 303
- [Retiming the Design](#), on page 305

## Pipeline the Design

Pipeline is the process of splitting logic into stages so that the first stage can begin processing new inputs while the last stage is finishing the previous inputs. This ensures better throughput and faster circuit performance. For pipeline, the software splits the logic by moving registers into the multiplier or ROM:

The following procedure shows you how to control pipeline.



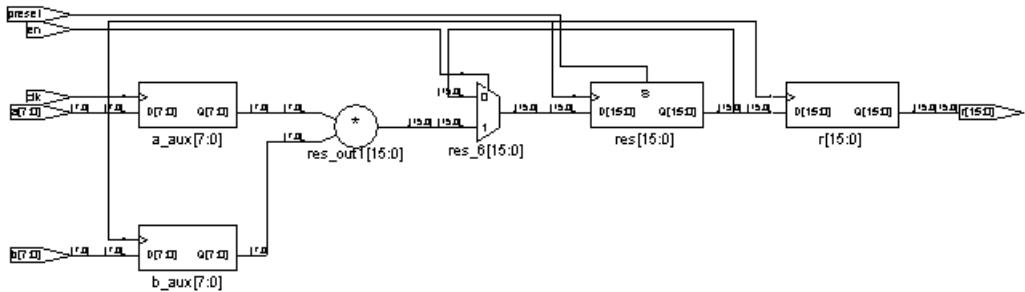
1. To pipeline, make sure the objects meet the following criteria:
  - You can pipeline ROMs and multipliers, but you can only pipeline multipliers if the adjacent register has a synchronous reset.
  - ROMs to be pipelined must be at least 512 words. Anything below this limit is too small.
  - You can push any kind of flip-flop into the module, as long as all the flip-flops in the pipeline have the same clock, the same set/reset signal or lack of it, and the same enable control or lack of it.

- To enable pipelining globally for the whole design, use the option set pipe 1 command.

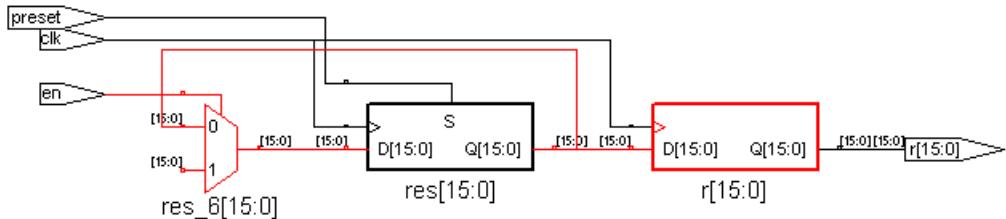
Use this approach as a first pass to get a feel for which modules you can pipeline. If you know exactly which registers you want to pipeline, add the attribute to the registers in the source code or interactively using the SCOPE interface.

- To check whether individual registers are suitable for pipelining, do the following:

- Open the schematic for a compiled view of the design.
- Select the register and press F12 to filter the schematic view.



- In the filtered schematic view, select the output and type e (or select Expand from the popup menu). Check that the register is suitable for pipelining.



- To enable pipelining on selected registers, use one of these techniques:

- Enable pipelining globally (option set pipe 1) and then set the syn\_pipeline attribute with a value of 0 or false on any registers that you do not want the software to move. This attribute specifies that the register cannot be moved for pipelining.
- Do not enable pipelining. Attach the syn\_pipeline attribute with a value of 1 or true on any registers you want the software to consider for pipelining.

pipelining. This attribute marks the register as one that could be moved during pipelining, but it might not be moved if it is not needed.

See [syn\\_pipeline](#), on page 732 for details of the syntax.

#### 5. Map the design.

The software looks for registers where all the flip-flops of the same row have the same clock, no control signal, or the same unique control signal, and pushes them inside the module marked for pipelining. It attaches the syn\_pipeline attribute to all these registers. If there already is a syn\_pipeline attribute on a register, the software implements it.

#### 6. Check the log file.

Use the Find command for occurrences of the word pipelining to find out which modules got pipelined. The log file entries look like this:

```
@N: | Pipelining module res_out1  
@N: |res_i is level 1 of the pipelined module res_out1  
@N: |r is level 2 of the pipelined module res_out1
```

## Retiming the Design

Retiming improves the timing performance of sequential circuits without modifying the source code. It automatically moves registers (register balancing) across combinatorial gates or LUTs to improve timing while maintaining the original behavior as seen from the primary inputs and outputs of the design. Retiming moves registers across gates or LUTs, but does not change the number of registers in a cycle or path from a primary input to a primary output. However, it can change the total number of registers in a design.

The retiming algorithm retimes only edge-triggered registers. It does not retime level-sensitive latches. Note that registers associated with RAMS, DSPs, and the mapping for generated clocks may be moved, regardless of the retiming option setting.

These sections contain details about using retiming.

- [Controlling Retiming](#), on page 306
- [Retiming Example](#), on page 307
- [How Retiming Works](#), on page 308

## Controlling Retiming

The following procedure shows you how to use retiming.

1. To enable retiming globally for the whole design, use the option set retiming 1 command.

Retiming works globally on the design, and moves edge-triggered registers as needed to balance timing. Retiming is a superset of pipelining, so when you select it, you automatically select pipelining (*Pipelining the Design*, on page 303).

2. To enable retiming on selected registers, use either of the following techniques:
  - Enable retiming globally (option set retiming 1) and then attach the `syn_allow_retimimg` attribute with a value of 0 or false to any registers you do not want to move. This attribute setting specifies that a register cannot be moved for retiming. Refer to *How Retiming Works*, on page 308 for a list of components the retiming algorithm will move.
  - Do not enable retiming. Attach the `syn_allow_retimimg` attribute with a value of 1 or true to any registers you want the software to consider for retiming. You can do this in the SCOPE interface or in the source code. This attribute setting marks the register as one that can be moved during retiming, but does not necessarily force it to be moved during retiming. If you apply the attribute to an FSM, RAM or SRL that is decomposed into flip-flops and logic, the software applies the attribute to all the resulting flip-flops.
3. You can fine-tune retiming using attributes:
  - To preserve the power-on state of flip-flops without sets or resets (FD or FDE) during retiming, set `syn_preserve=1` or `syn_allow_retimimg=0` on these flip-flops.
  - To force flip-flops to be packed in I/O pads, globally set `syn_useioff=1`. This prevents the flip-flops from being moved during retiming.
4. Set other options for the run. Retiming might affect some constraints and attributes. See *How Retiming Works*, on page 308 for details.
5. Map the design.

After the LUTs are mapped, the software moves registers to optimize timing. See [Retiming Example, on page 307](#) for an example. The software honors other attributes you set, like `syn_preserve`, `syn_useioff`, and `syn_ram-style`. See [How Retiming Works, on page 308](#) for details.

Note that the tool might retime registers associated with RAMs, DSPs, and generated clocks, regardless of whether retiming is enabled or disabled.

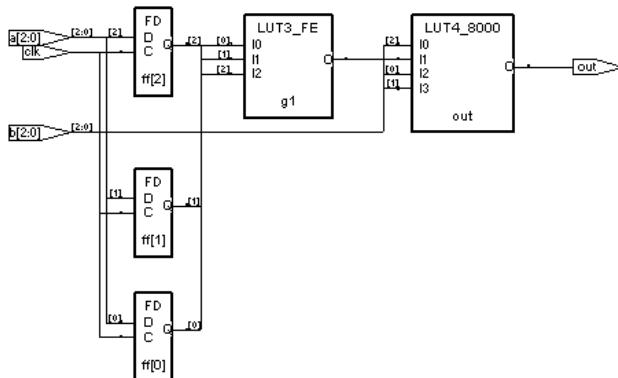
6. Check the log file for retiming information.

The log file includes a retiming report that you can analyze to understand the retiming changes. The retiming report includes the following information:

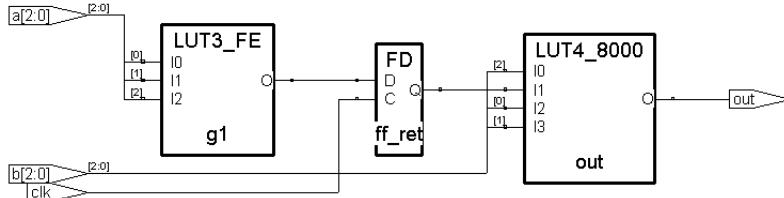
- The number of registers added, removed, or untouched by retiming.
- Names of the original registers that were moved by retiming and which no longer exist in the mapped view.
- Names of the registers created as a result of retiming, and which did not exist in the compiled view. These retimed registers have a `_ret` suffix.

## Retiming Example

The first figure is not retimed, and shows two levels of logic between the registers and the output, and no levels of logic between the inputs and the registers.



The second figure shows the results of retiming the three registers at the input of the OR gate. The two logic levels from the register to the output are reduced to one. The retimed circuit has better performance than the original circuit. Timing is improved by transferring one level of logic from the critical part of the path (register to output) to the non-critical part (input to register).



## How Retiming Works

This section describes how retiming affects sequential components (flip-flops). Registers associated with RAMs, DSPs, and the mapping for fixing generated clocks might be moved, whether retiming is enabled or not. Here are some implications and results of retiming:

- Flip-flops with no control signals (resets, presets, and clock enables) can be retimed. Flip-flops with minimal control logic can also be retimed. Multiple flip-flops with reset, set or enable signals that need to be retimed together are only retimed if they have exactly the same control logic.
- The software does not retime the following combinatorial sequential elements: flip-flops with both set and reset, flip-flops with attributes like `syn_preserve`, flip-flops packed in I/O pads, level-sensitive latches, registers that are instantiated in the code, SRLs, and RAMs. If a RAM with combinatorial logic has `syn_ramstyle` set to registers, the registers can be retimed into the combinatorial logic.
- Retimed flip-flops are only moved through combinatorial logic. The software does not move flip-flops across the following objects: black boxes, sequential components, tristates, I/O pads, instantiated components, carry and cascade chains, and keepbufs. For Altera designs, registers that are in counter modes are not retimed to preserve the performance benefit of the counter mode.
- There may not be a one-to-one correspondence between registers in a compiled design and the same retimed design after retiming. A single

register in the compiled view might now correspond to multiple registers in the retimed view.

- Retiming affects or is affected by, these attributes and constraints:

| Attribute/Constraint             | Effect                                                                                                                                                                                                                                                                                               |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| False path constraint            | Does not retime flip-flops with different false path constraints. Retimed registers affect timing constraints.                                                                                                                                                                                       |
| Multicycle constraint            | Does not retime flip-flops with different multicycle constraints. Retimed registers affect timing constraints.                                                                                                                                                                                       |
| Register constraint              | Does not maintain <code>set_reg_input_delay</code> and <code>set_reg_output_delay</code> constraints. Retimed registers affect timing constraints.                                                                                                                                                   |
| from/to timing exceptions        | Does not retime timing register constraints set with a from/to specification. <code>max_delay</code> constraints are the exception to this rule. In this case, retiming is performed but the constraint is not forward-annotated, because the <code>max_delay</code> value would no longer be valid. |
| <code>syn_hier=macro</code>      | Does not retime registers in a macro with this attribute.                                                                                                                                                                                                                                            |
| <code>syn_keep</code>            | Does not retime across keepbufs generated because of this attribute.                                                                                                                                                                                                                                 |
| <code>syn_hier=macro</code>      | Does not retime registers in a macro with this attribute.                                                                                                                                                                                                                                            |
| <code>syn_pipeline</code>        | Automatically enabled if retiming is enabled.                                                                                                                                                                                                                                                        |
| <code>syn_preserve</code>        | Does not retime flip-flops with this attribute set.                                                                                                                                                                                                                                                  |
| <code>syn_probe</code>           | Does not retime net drivers with this attribute. If the net driver is a LUT or gate, no flip-flops are retimed across it.                                                                                                                                                                            |
| <code>syn_reference_clock</code> | On a critical path, does not retime registers with different <code>syn_reference_clock</code> values together, because the path effectively has two different clock domains.                                                                                                                         |
| <code>syn_useioff</code>         | Does not override attribute-specified packing of registers in I/O pads. If the attribute value is false, the registers can be retimed. If the attribute is not specified, the timing engine determines whether the register is packed into the I/O block.                                            |
| <code>syn_allow_retimming</code> | Does not retime registers if the value is 0.                                                                                                                                                                                                                                                         |

- Retiming does not change simulation behavior (as observed from primary inputs and outputs) of your design. However if you are monitoring (probing) values on individual registers inside the design, you might need to modify your test bench if the probe registers are retimed.
- If retiming is enabled, registers connected to unconstrained I/O pins are not retimed by default. If you want to retime I/O paths, add constraints to all input/output ports, and constrain each I/O pin separately as required.

# Preserving Objects from Being Optimized Away

Synthesis can collapse or remove nets during optimization. If you want to retain a net for simulation, probing, or for a different synthesis implementation, you must specify this with an attribute. Similarly, the software removes duplicate registers or instances with unused output. If you want to preserve this logic for simulation or analysis, you must use an attribute. The following table lists the attributes to use in each situation. For details about the attributes and their syntax, see the *Attributes Reference Manual*.

| To Preserve ...         | Use ...                                                                                                                                                                          | Result                                                                                                                                                                                                                               |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nets                    | <code>syn_keep</code> on wire or reg (Verilog), or signal (VHDL).<br>For Microsemi designs (except 500K and PA), use <code>alspreserve</code> as well as <code>syn_keep</code> . | Keeps net for simulation, a different synthesis implementation, or for passing to the place-and-route tool.                                                                                                                          |
| Nets for probing        | <i>Synplify Pro</i> , <i>Synplify Premier</i><br><code>syn_probe</code> on wire or reg (Verilog), or signal (VHDL)                                                               | Preserves internal net for probing.                                                                                                                                                                                                  |
| Shared registers        | <code>syn_keep</code> on input wire or signal of shared registers                                                                                                                | Preserves duplicate driver cells, prevents sharing. See <a href="#">Using <code>syn_keep</code> for Preservation or Replication</a> , on page 312 for details on the effects of applying <code>syn_keep</code> to different objects. |
| Sequential components   | <code>syn_preserve</code> on reg or module (Verilog), signal or architecture (VHDL)                                                                                              | Preserves logic of constant-driven registers, keeps registers for simulation, prevents sharing                                                                                                                                       |
| FSMs                    | <code>syn_preserve</code> on reg or module (Verilog), signal (VHDL)                                                                                                              | Prevents the output port or internal signal that holds the value of the state register from being optimized                                                                                                                          |
| Instantiated components | <code>syn_noprune</code> on module or component (Verilog), architecture or instance (VHDL)                                                                                       | Keeps instance for analysis, preserves instances with unused outputs                                                                                                                                                                 |

See the following for more information:

- [Using syn\\_keep for Preservation or Replication](#), on page 312
- [Controlling Hierarchy Flattening](#), on page 315
- [Preserving Hierarchy](#), on page 315

## Using syn\_keep for Preservation or Replication

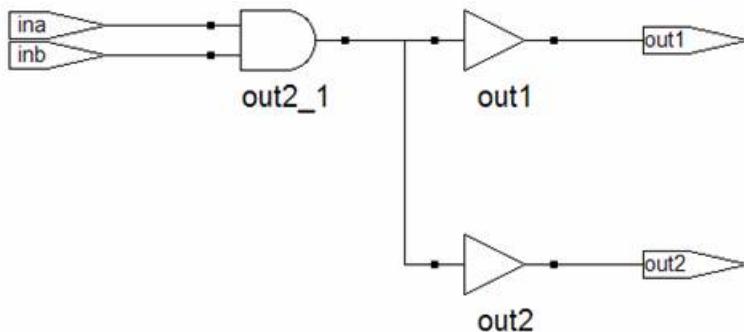
By default the tool considers replicated logic redundant, and optimizes it away. If you want to maintain the redundant logic, use `syn_keep` to preserve the logic that would otherwise be optimized away.

The following Verilog code specifies a replicated AND gate:

```
module redundant1(ina,inb,out1);
  input ina,inb;
  output out1,out2;
  wire out1;
  wire out2;

  assign out1 = ina & inb;
  assign out2 = ina & inb;
endmodule
```

The compiler implements the AND function by replicating the outputs `out1` and `out2`, but optimizes away the second AND gate because it is redundant.



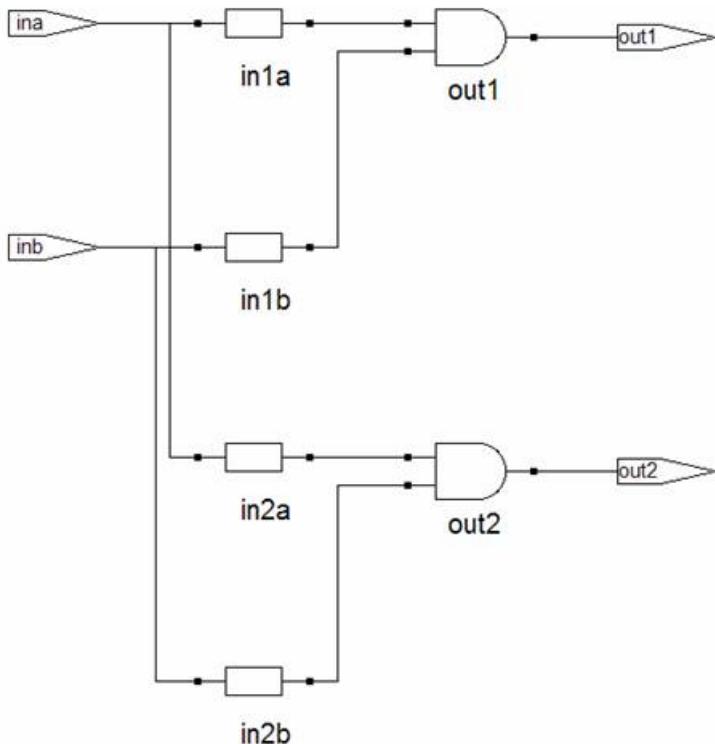
To replicate the AND gate in the previous example, apply `syn_keep` to the input wires, as shown below:

```
module redundant1d(ina,inb,out1,out2);
    input ina,inb;
    output out1,out2;
    wire out1;
    wire out2;

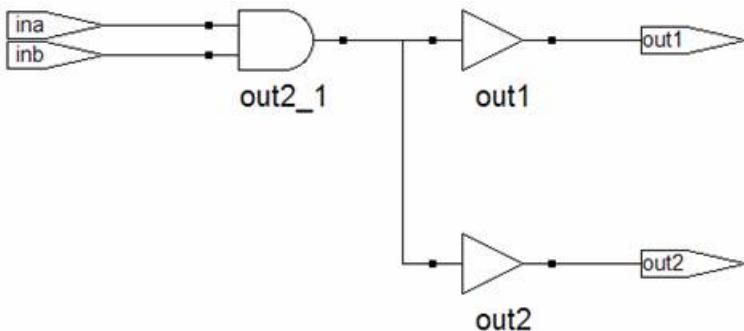
    wire in1a /*synthesis syn_keep = 1*/;
    wire in1b /*synthesis syn_keep = 1*/;
    wire in2a /*synthesis syn_keep = 1*/;
    wire in2b /*synthesis syn_keep = 1 */;

    assign in1a = ina;
    assign in1b = inb;
    assign in2a = ina;
    assign in2b = inb;
    assign out1 = in1a & in1b;
    assign out2 = in2a & in2b;
endmodule
```

Setting `syn_keep` on the input wires ensures that the second AND gate is preserved:



You must set `syn_keep` on the input wires of an instance if you want to preserve the logic, as in the replication of this AND gate. If you set it on the outputs, the instance is not replicated, because `syn_keep` preserves the nets but not the function driving the net. If you set `syn_keep` on the outputs in the example, you get only one AND gate, as shown in the next figure.



## Controlling Hierarchy Flattening

Optimization flattens hierarchy. To control the flattening, use the `syn_hier` attribute as described here. You can also use the attribute to prevent flattening, as described in [Preserving Hierarchy, on page 315](#).

1. Attach the `syn_hier` attribute with the value you want to the module or architecture you want to preserve.

| To ...                                                                   | Value ...                    |
|--------------------------------------------------------------------------|------------------------------|
| Flatten all levels below, but not the current level                      | <code>flatten</code>         |
| Remove the current level of hierarchy without affecting the lower levels | <code>remove</code>          |
| Remove the current level of hierarchy and the lower levels               | <code>flatten, remove</code> |
| Flatten the current level (if needed for optimization)                   | <code>soft</code>            |

You can also add the attribute in SCOPE instead of the HDL code. If you use SCOPE to enter the attribute, make sure to use the `v:` syntax. For details, see [syn\\_hier, on page 642](#).

The software flattens the design as directed. If there is a lower-level `syn_hier` attribute, it takes precedence over a higher-level one.

2. If you want to flatten the entire design, use the `syn_netlist_hierarchy` attribute set to `false`, instead of the `syn_hier` attribute.

This flattens the entire netlist and does not preserve any hierarchical boundaries. See [syn\\_netlist\\_hierarchy, on page 695](#) for the syntax.

## Preserving Hierarchy

The synthesis process includes cross-boundary optimizations that can flatten hierarchy. To override these optimizations, use the `syn_hier` attribute as described here. You can also use this attribute to direct the flattening process as described in [Controlling Hierarchy Flattening, on page 315](#).

1. Attach the `syn_hier` attribute to the module or architecture you want to preserve. You can also add the attribute in SCOPE. If you use SCOPE to enter the attribute, make sure to use the `v:` syntax.

## 2. Set the attribute value:

| To ...                                                                                                                               | Value ...     |
|--------------------------------------------------------------------------------------------------------------------------------------|---------------|
| Preserve the interface but allow cell packing across the boundary                                                                    | firm          |
| Preserve the interface with no exceptions (Altera, Microsemi, and Xilinx only)                                                       | hard          |
| Preserve the interface and contents with no exceptions (Microsemi (except PA, 500K, and ProASIC3 families), Altera and Lattice only) | macro         |
| Flatten lower levels but preserve the interface of the specified design unit                                                         | flatten, firm |

The software flattens the design as directed. If there is a lower-level `syn_hier` attribute, it takes precedence over a higher-level one.

# Sharing Resources

One of the ways to optimize area is to use resource sharing in the compiler. With resource sharing, the software uses the same arithmetic operators for mutually exclusive statements; for example, with the branches of a case statement. Conversely, you can improve timing by disabling resource sharing, but at the expense of increased area.

Compiler resource sharing is on by default. You can set it globally and then override the global setting on individual modules.

1. To disable resource sharing globally for the whole design, use one of the methods below.

Leave the default setting to improve area; disable the option to improve timing.

- Use the following command before compiling the design:

```
option set -resource_sharing 0
```

- Apply the `syn_sharing` directive to the top-level module or architecture in the HDL code before compiling. See [syn\\_sharing, on page 797](#) for syntax and examples.

---

```
Verilog module top(out, in, clk_in) /* synthesis syn_sharing = "off" */;
```

```
VHDL architecture rtl of top is
    attribute syn_sharing : string;
    attribute syn_sharing of rtl : architecture is "false";
```

---

The resource sharing setting does not affect the mapper, so even if resource sharing is disabled, the tool can share resources during the mapping phase to optimize the design and improve results.

2. To specify resource sharing for individual modules or override the global setting, specify the `syn_sharing` attribute for individual modules/architectures.

# Inserting I/Os

You can control I/O insertion globally, or on a port-by-port basis.

1. To control the insertion of I/O pads at the top level of the design, use the option set disable\_io\_insertion command as follows:
  - Enable the option (option set disable\_io\_insertion 1) if you want to do a preliminary run and check the area taken up by logic blocks, before synthesizing the entire design. With this setting, no I/O pads are inserted in your design, unless you manually instantiate them.  
For the most control, enable the option and then manually instantiate the I/O pads for specific pins, as needed.
  - Disable the option (option set disable\_io\_insertion 0) if you want to automatically insert I/O pads for all the inputs, outputs and bidirectionals. With this setting, the software inserts I/O pads for inputs, outputs, and bidirectionals in the output netlist. You can override an automatically inserted I/O pad by directly instantiating another I/O pad.
2. Optionally specify the syn\_insert\_pad attribute in an fdc file to override a global setting. Set the attribute on a port or a net.
  - Set the attribute to 0 to remove an existing I/O buffer or prevent an I/O buffer from being automatically inserted. For example:

```
define_attribute {p:RST} syn_insert_pad {0}
```
  - Set the attribute to 1 to replace a previously removed I/O buffer.

For the syntax of this attribute, see [syn\\_insert\\_pad, on page 659](#).

## CHAPTER 5

# Working with Compile Points

---

Automatic compile points (ACP) are RTL partitions of the design. For discussions about various aspects of ACPS and their use, see the following topics:

- [Automatic Compile Point Basics](#), on page 320
- [Using Automatic Compile Points](#), on page 330

# Automatic Compile Point Basics

Automatic compile points (ACP) are RTL partitions of the design, which are generated automatically by the synthesis tool.

The tool makes the decisions and automatically creates compile points based on various parameters, like the size of the design, the sizes of hierarchical modules, their boundary logic, the number of ports driven by constants, and so on. The down side to using automatic compile points is that they might increase area if the partition boundaries prevent many cross-boundary optimizations.

All compile points have a compile point type, based on the amount of cross-boundary optimizations. For Xilinx targets, the tool automatically sets the ACP type to locked.

All compile points can be synthesized, optimized, placed and routed independently.

For details about automatic compile points, see the following topics:

- [ACPs and Runtime](#), on page 320
- [Nested Compile Points](#), on page 322
- [Compile Point Types](#), on page 322
- [Automatic Constraint Extraction and Interface Timing](#), on page 328

## ACPs and Runtime

Runtime reduction techniques like multiprocessing, distributed processing, and incremental compilation or mapping rely on compile points to define subdivisions. You can also reduce runtime by defining ACPs that consist of logic that is reused in different parts of the design.

- Distributed processing or multiprocessing  
Distributed processing runs parallel processes across multiple machines using the Synopsys CDPL mechanism, and multiprocessing runs multiple parallel processes on different processors. Both methods use compile points to determine where to divide the design into separate processes, and both synthesize the compile points in parallel on multiple processors.

For both distributed processing and multiprocessing, the type of ACP affects how they are processed. It is recommended that you use hard or locked compile points because they offer the best runtime advantage with minimal QoR loss and area impact.

Soft ACPs might not be processed in parallel, unless they are independent (at the leaf level). Upper-level compile points that contain soft lower-level compile points cannot be processed until the lower level has been mapped, with the top level being processed last. By contrast, if you have hard or locked compile points, they are all processed in parallel, including the top level.

- Incremental compilation and mapping

Incremental synthesis uses compile points to determine which portions of the design to rerun, only resynthesizing compile points that have been modified. The following describe some design situations when incremental synthesis becomes necessary and compile points are useful:

- During the initial design phase, design modules are being designed. Use compile points to preserve unchanged design modules and evaluate the effects of modifications to parts of the design that are still changing.
- During design integration, use compile points to preserve the main design modules and only allow the glue logic to be remapped.
- If your design contains IP, synthesize the IP, and use compile points to preserve them while you run incremental synthesis on the rest of the design.
- In the final stages of the design, use compile points to preserve design modules that do not need to be updated while you work through minor RTL changes in another part of the design.

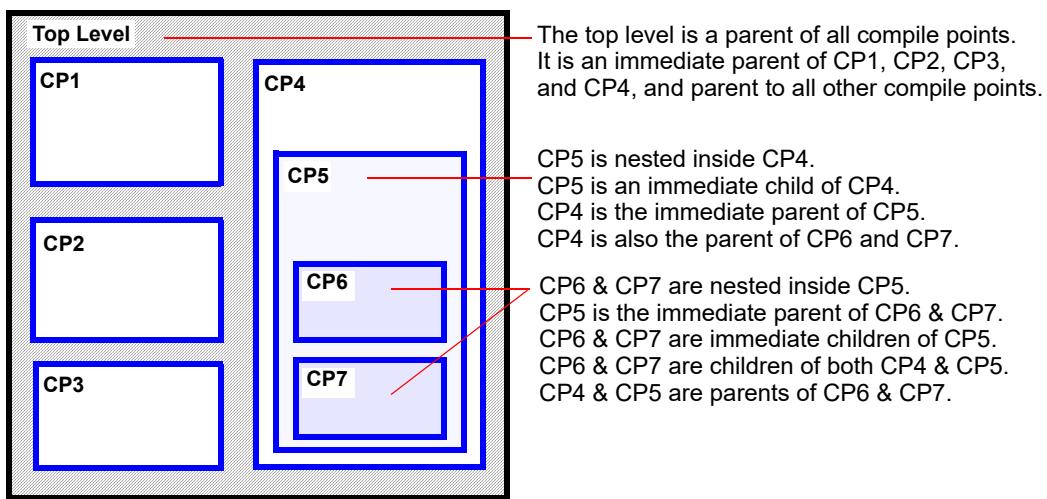
- Compile point definition

If a block that is instantiated multiple times in a design is defined as a compile point, the tool can synthesize it once and then re-use the results wherever it applies in the design, without resynthesizing that logic. This reduces runtime, especially if the block is instantiated many times in the design. Similarly it can help reduce runtime during incremental synthesis, if it is one of the compile points that was modified.

## Nested Compile Points

A design can have any number of compile points, and compile points can be nested inside other compile points. In the following figure, compile point CP6 is nested inside compile point CP5, which is nested inside compile point CP4.

To simplify things, the term *child* is used to refer to a compile point that is contained inside another compile point; the term *parent* is used to refer to a container compile point that contains a child. These terms are not used in their strict sense of direct, immediate containment: If a compile point A is nested in B, which is nested in C, then A and B are both considered children of C, and C is a parent of both A and B. The top level is considered the parent of all compile points. In the figure above, both CP5 and CP6 are children of CP4; both CP4 and CP5 are parents of CP6; CP5 is an immediate child of CP4 and an immediate parent of CP6.



## Compile Point Types

Compile point boundaries limit optimizations. Cross-boundary optimizations typically improve area and timing, but take longer to run. The compile point type determines whether boundary optimizations are allowed. The type of compile point can also affect runtime (see [ACPs and Runtime, on page 320](#)).

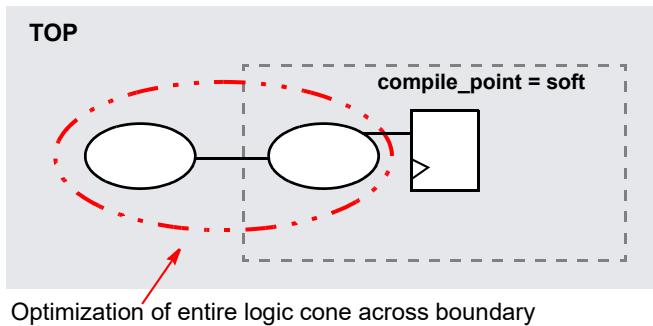
The tool marks ACPs as locked by default. These are descriptions of the soft, hard, locked, locked,partition, and black\_box compile types:

- Soft

Compile point boundaries can be reoptimized during top-level mapping. Timing optimizations like sizing, buffering, and DRC logic optimizations can modify boundary instances of the compile point and combine them with functions from the next higher level of the design. The compile point interface can also be modified. Multiple instances are uniquified. Any optimization changes can propagate both ways: into the compile point and from the compile point to its parent.

Using soft mode usually yields the best quality of results, because the software can utilize boundary optimizations. On the other hand, soft compile points can take a longer time to run than the same design with hard or locked compile points. Unless they are at the leaf level, soft compile points are not processed in parallel. Upper levels that contain soft compile points cannot be processed until the lower level has been mapped, with the top level processed last.

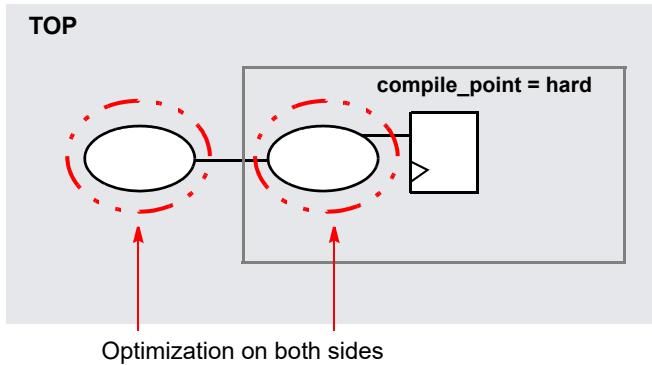
The following figure shows the soft compile point with a dotted boundary to show that logic can be moved in or out of the compile point.



- Hard

For hard compile points, the compile point boundary can be reoptimized during top-level mapping and instances on both sides of the boundary can be modified by timing and DRC optimizations using top-level constraints. However, the boundary is not modified. Any changes can propagate in either direction while the compile point boundary (port/interface) remains unchanged. Multiple instances are uniquified. For performance improvements, constant propagation and removal of unused logic optimizations are performed across hard compile points.

In the following figure, the solid boundary on the hard compile point indicates that no logic can be moved in or out of the compile point.



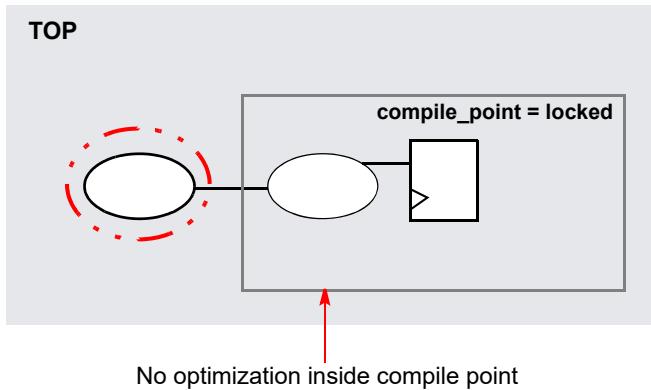
The hard compile point type allows for optimizations on both sides of the boundary without changing the boundary. There is a trade-off in quality of results to keep the boundaries. Using hard also allows for hierarchical equivalence checking for the compile point module.

- **Locked**

This is the default compile point type. With a locked compile point, the tool does not make any interface changes or reoptimize the compile point during top-level mapping. An interface logic model (ILM) of the compile point is created (see [Interface Logic Models, on page 328](#)) and included for the top-level mapping. The ILM remains unchanged during top-level mapping.

The locked value indicates that all instances of the same compile point are identical and unaffected by top-level constraints or critical paths. As a result, multiple instances of the compile point module remain identical even though the compile point is unqualified. The schematic view after mapping shows unique names for multiple instances, but the final Verilog netlist (vma file) restores the original module names for the multiple instances.

Timing optimization can only modify instances outside the compile point. Although the compile point is used to time the top-level netlist, changes do not propagate into or out of a locked compile point. The following figure shows a solid boundary for the locked compile point to indicate that no logic is moved in or out of the compile point during top-level mapping.



This mode has the largest trade-off in terms of QoR, because there are no boundary optimizations. So, it is very important to provide accurate constraints for locked compile points. The following table lists some advantages and limitations with the locked compile point:

| Advantages                                                                                                                     | Limitations                                                              |
|--------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| Consumes smallest amount of memory.                                                                                            | Interface timing                                                         |
| Fastest to process; reduces runtime.                                                                                           | Constant propagation                                                     |
| Lets you achieve stable results for a completed part of the design.                                                            | BUFG insertion                                                           |
| Allows for hierarchical place and route with multiple output netlists for each compile point and the top-level output netlist. | GSR hookup                                                               |
| Allows for hierarchical simulation.                                                                                            | Do not instantiate I/O pads (e.g. IBUFs and OBUFs) within compile points |

- Locked, partition

You can also specify a compile point type to be locked, partition. With this setting and depending on the technology specified, the tool creates an `xpartition.sxml` file for the compile points that are defined and includes timestamps for each compile point. The contents of this file is used by Xilinx incremental place and route.

This mode offers place-and-route runtime advantages and lets you converge on stable results for a completed design. However, this mode has the largest trade-off of quality of results because boundary optimizations are not allowed.

- **Black Box**

The tool treats `black_box` compile points as black boxes. It ignores the contents of the compile point and only uses its ports for synthesis. Black box compile point modules only write port definitions to the netlist files. The contents are not written to any of the netlist files (`srm`, `edf`, `vqm`, `edn`, `vm`, or `vhm`). This compile point type supports all black box directives.

You can change the type of a compile point to `black_box` at any time. The previous compile point results are retained from intermediate mapping, but the parent compile point might be remapped. This table shows the results when the RTL is unchanged and a compile point is changed to `black_box` for the second run:

| Original ACP | Effect of changing to <code>black_box</code>                                                                                                                                                                                                |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| soft, hard   | The top level is remapped because of cross-boundary optimizations between the top-level and compile point A during the previous synthesis run. For the second run, since A is a black box, no logic from the top-level can be moved into A. |
| locked       | The mapper ignores the ACP contents. When mapping other hierarchical compile points, the tool uses only the black box port information. It writes port definitions only to the output netlist files for the black box compile points.       |

If you specify the `continue on error` option, the tool creates black box compile points when it encounters an error, instead of stopping the synthesis process. Resolve the errors before mapping.

## Compile Point Type Summary

The following table summarizes how the tool handles different compile points during mapping:

| Features                                    | Compile Point Type |              |              |              |
|---------------------------------------------|--------------------|--------------|--------------|--------------|
|                                             | Soft               | Hard         | Locked       | Black Box    |
| Boundary optimizations                      | Yes                | Limited      | No           | No           |
| Uniquification of multiple instance modules | Yes                | Yes          | Limited      | No           |
| Compile point interface (port definitions)  | Modified           | Not modified | Not modified | Not modified |
| Hierarchical simulation                     | No                 | Yes          | Yes          | Yes*         |
| Hierarchical equivalence checking           | No                 | Yes          | Yes          | Yes*         |
| Interface Logic Model (created/used)        | No                 | Yes          | Yes          | No           |

\* If you replace the black box with the original RTL, you can run hierarchical simulation or hierarchical equivalence checking on the rest of the design.

## Automatic Compile Point Generation

The process of ACP generation follows certain rules:

- The ACP process does not generate new hierarchy. It honors existing hierarchy, so if you have RAMs, ROMs or DSPs that cross hierarchies, it does not disturb them. The automatic compile point process does not generate automatic compile points for IP modules like NGCs and EDIFs.
- The automatic compile point process overrides the `syn_hier` setting, if one is applied to a module. If `syn_hier` is set to soft, the tool determines the best optimization across hierarchical boundaries. If the tool makes the same module into an automatic compile point, it applies the `syn_hier=fixed` attribute to that module, and sets the compile point type to locked.

The tool goes through these stages to generate automatic compile points and their constraints.

1. It first identifies compile points based on factors like the size of hierarchical modules, their boundary logic, and the number of hierarchical ports driven by constants.
2. It extracts compile point constraints from the top-level timing constraints, and propagates this interface timing automatically to the automatic compile points. See *Automatic Constraint Extraction and Interface Timing*, on page 328 for details. For black box compile point modules, the tool only writes port definitions to the netlist files. It does not write the contents of the module to any of the netlist files like srm, edf, vqm, edn, vm, or vhm.

## Automatic Constraint Extraction and Interface Timing

Once you specify the top level constraints for your design, the tool automatically derives and extracts the constraints for the ACPs from the top-level constraints and applies them when it synthesizes the compile points. It creates an environment around the compile point and applies the top-level constraints to it to derive the compile point-level timing constraints. It then uses these constraints to map the compile point.

When the tool calculates interface timing, it uses the clock frequency set for the top level. If the top-level clock frequency is 100 MHz, the tool uses this top-level value for the ACP.

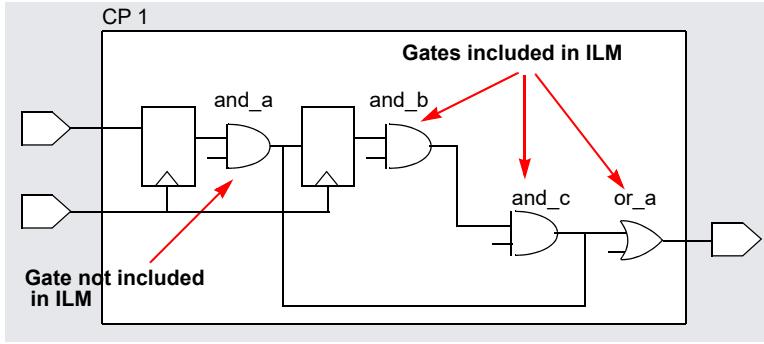
Automatic compile points prohibit some optimizations, because they are locked. To permit cross-boundary optimizations, you must set it to soft or hard.

## Interface Logic Models

The interface logic model (ILM) of a locked or hard compile point is a timing model that contains only the interface logic necessary for accurate timing. An ILM is a partial gate-level netlist that represents the original design accurately while requiring less memory during mapping. Using ILMs improves the runtime for static timing analysis without compromising timing accuracy.

The tool does not do any timing optimizations on an ILM. The interface logic is preserved with no modifications. All logic required to recreate timing at the top level is included in the ILM. ILM logic includes any paths from an input/inout port to an internal register, an internal register to an output/inout port, and an input/inout port to an output/inout port.

The tool removes internal register-to-register paths, as shown in this example. In this design, `and_a` is not included in the ILM because the timing path that goes through `and_a` is an internal register-to-register path.



# Using Automatic Compile Points

You can run normal or incremental synthesis on designs with ACPs. For details about using automatic compile points, see the following topics:

- [Adding ACPs to the Design](#), next
- [Creating a Top-Level Constraints File for ACP](#), on page 333
- [Compile Point Mapping](#), on page 333
- [Incremental Compile Point Synthesis](#), on page 334
- [Forward-annotation of Compile Point Timing Constraints](#), on page 336
- [Analyzing Compile Point Results](#), on page 336

## Adding ACPs to the Design

The following procedure gives you an overview of how to incorporate ACPs in your design.

1. Create a top-level constraint file with constraints for the entire design.

This file is required, because the definitions of the ACPs are added to this file and the top-level constraints defined here are propagated to the compile points, as described in [Automatic Constraint Extraction and Interface Timing](#), on page 328. For information about creating a constraints file, see [Creating a Top-Level Constraints File for ACP](#), on page 333.

The tool writes a `define_compile_point` command for each ACP. For example:

```
define_compile_point {v:work.prgm_cntr} -type {locked}
```

```
13
14 ##### BEGIN Collections - (Populated from tab in SCOPE, do not edit)
15 ##### END Collections
16
17 ##### BEGIN Clocks - (Populated from tab in SCOPE, do not edit)
18 create_clock {p:clock} -period {10}
19
20 ##### END Clocks
21
22 ##### BEGIN "Generated Clocks" - (Populated from tab in SCOPE, do not edit)
23 ##### END "Generated Clocks"
24
25 ##### BEGIN Inputs/Outputs - (Populated from tab in SCOPE, do not edit)
26 ##### END Inputs/Outputs
27
28 ##### BEGIN "Delay Paths" - (Populated from tab in SCOPE, do not edit)
29 ##### END "Delay Paths"
30
31 ##### BEGIN Attributes - (Populated from tab in SCOPE, do not edit)
32 ##### END Attributes
33
34 ##### BEGIN "I/O Standards" - (Populated from tab in SCOPE, do not edit)
35 ##### END "I/O Standards"
36
37 ##### BEGIN "Compile Points" - (Populated from tab in SCOPE, do not edit)
38 define_compile_point {v:work.prgm_cntr} -type {locked}
39 ##### END "Compile Points"
40
```

2. Specify that you want to generate compile points with the option set command in a Tcl file:

```
option set automatic_compile_point 1
```

For more information about the process, refer to [Automatic Compile Point Generation, on page 327](#).

3. If you want to continue compiling the remainder of the design if an error is encountered within a compile point, set the `continue_on_error` option with the option set command.

When this is enabled, the tool automatically black-boxes any compile points that have mapper errors during synthesis, and continues to configure the compile points. Note that if you have compiler errors in the RTL, you must correct them before proceeding with synthesis.

4. If you do not want a module to be made into an automatic compile point, set the `syn_no_compile_point` attribute on that design module in the top-level constraint file.

Set this attribute if you want the ACP to be optimized. By default the ACP is locked and does not allow for optimizations. If the compile point is a nested compile point, apply the attribute to each level of hierarchy to prevent the tool from creating a compile point from a sub-module.

5. If you want to specify resources for a compile point, use the `syn_allowed_resources` command.

Apply the `syn_allowed_resources` attribute globally or to an individual compile point to specify its allowed resources. When a compile point is synthesized, the resources of its siblings and parents cannot be taken into account because it stands alone as an independent synthesis unit. This attribute limits dedicated resources such as block RAMs or DSPs that the compile point can use, so that there are adequate resources available for all parts of the design.

6. If you want to run processes in parallel, do the following:
  - Set `max_parallel_jobs` to define the number of parallel jobs to run.
  - To use CDPL distributed processing set up the hosts, and then set option set `cdpl`.

If you have not specified CDPL, the jobs run in parallel on the available processors (multiprocessing). If you specified CDPL, the parallel jobs run on the specified machines.

7. Compile and map the design.

The tool generates automatic compile points when you compile the design. See [Compile Point Mapping, on page 333](#) for details of the process. The generated compile points are also used as the basis for multiprocessing, distributed processing, and incremental runs.

For subsequent runs, if the design changes are minor ones within a compile point, the tool only recompiles or remaps those compile points that have changed from the last run. However, for major changes like the introduction of a new module, the tool regenerates compile points. See [Incremental Compile Point Synthesis, on page 334](#) for additional information.

After synthesis for all the automatic compile points are done, the software reloads all the automatic compile point results and writes out a single output netlist and one constraint file for the entire design. See [Forward-annotation of Compile Point Timing Constraints, on page 336](#) for a description of the constraints that are forward-annotated.

### 8. Check results.

Check the top-level log file for the number of ACPs identified, the number of parallel jobs, and the status on incremental runs. See [Analyzing Compile Point Results, on page 336](#) for details.

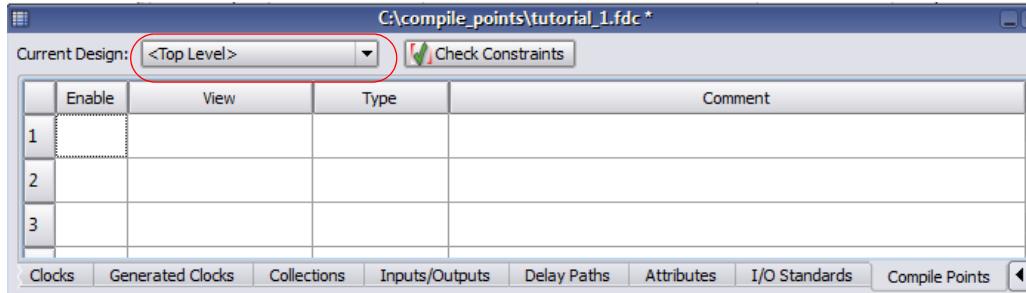
## Creating a Top-Level Constraints File for ACP

All compile points require a top-level constraints file. If you have manual compile points, define them in this file. The top-level file also contains design-level constraints.

The following procedure describes how to create a top-level constraints file for ACP.

### 1. Create the top-level constraints file.

You can type in the constraints, or use the constraints editor to enter them. Click the Constraints Editor icon to open the GUI. It includes a Current Design field, where you can specify constraints for the top-level design from the drop-down menu.



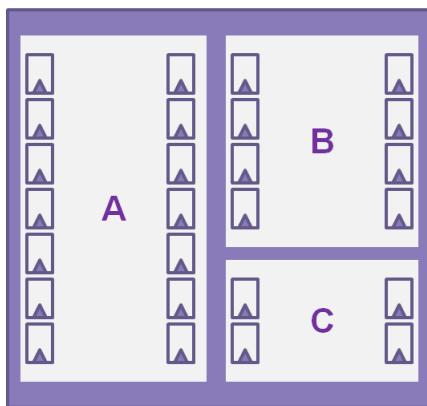
- Set top-level constraints like input/output delays, clock frequencies or multicycle paths.
- Save the top-level constraints file and specify it with the run compile command.

## Compile Point Mapping

The tool maps automatic compile points and the top level simultaneously.

A compile point stands on its own, and is optimized separately from its parent environment (the compile point container or the top level). This means that critical paths from a higher level do not propagate downwards, and they are unaffected by them. Automatic compile points have constraints automatically assigned from the top level.

When it synthesizes a compile point, the tool considers all other compile points as black boxes and only uses their interface timing information. In the following figure, when the tool is synthesizing compile point A, it applies relevant timing information to the boundary registers of B and C, because it treats them as black boxes.



By default, synthesis stops if the tool encounters an error while synthesizing a compile point. You can specify that the tool ignore the error (continue on error option) and continue synthesizing other compile points. If you use this option, the tool black boxes the compile point with the error and continues with the rest of the design. Resolve the errors in these black boxes before mapping.

## Incremental Compile Point Synthesis

The tool treats compile points as blocks for incremental synthesis. On subsequent synthesis runs, the tool runs incrementally and only resynthesizes those compile points that have changed, and the top level. The synthesis tool automatically detects design changes and resynthesizes compile points only if necessary. For example, it does not resynthesize a compile point if you only add or change a source code comment, because this change does not really affect the design functionality.

| Summary of Compile Points |        |             |
|---------------------------|--------|-------------|
| Name                      | Status | Reason      |
| mult                      | Mapped | No database |
| comb_logic                | Mapped | No database |
| alu                       | Mapped | No database |
| eight_bit_uc              | Mapped | No database |

=====

| Incremental Run Log Summary |           |                |
|-----------------------------|-----------|----------------|
| Summary of Compile Points   |           |                |
| Name                        | Status    | Reason         |
| mult                        | Unchanged | -              |
| comb_logic                  | Remapped  | Design changed |
| alu                         | Unchanged | -              |
| eight_bit_uc                | Unchanged | -              |

=====

The tool resynthesizes a compile point that has already been synthesized, in any of these cases:

- The HDL source code defining the compile point is changed in such a way that the design logic is changed.
- The constraints applied to the compile point are changed.
- Any of the options on the Device panel of the Implementation Options dialog box, except Update Compile Point Timing Data, are changed. In this case the entire design is resynthesized, including all compile points.
- You intentionally force the resynthesis of your entire design, including all compile points, with the Run->Resynthesize All command.
- The Update Compile Point Timing Data device mapping option is enabled and at least one child of the compile point (at any level) has been remapped. The option requires that the parent compile point be resynthesized using the updated timing model of the child. This includes the possibility that the child was remapped earlier, while the option was disabled. The newly enabled option requires that the updated timing model of the child be taken into account, by resynthesizing the parent.

Once generated, the model file is not updated unless there is an interface design change, or the file is deleted.

## Forward-annotation of Compile Point Timing Constraints

Constraints are forward-annotated to placement and routing from the top-level. However, not all compile point constraints are forward-annotated, as explained below. For example, constraints on top-level ports are always forward annotated, but compile point port constraints are not forward annotated.

- Top-level constraints are forward-annotated.
- Constraints applied to the interface (ports and bit ports) of the compile point are not forward-annotated.  
These include `input_delays`, `output_delays`, and clock definitions on the ports. Such constraints are only used to map the compile point itself, not its parents. They are not used in the final timing report, and they are not forward-annotated.
- Constraints applied to instances inside the compile point are forward-annotated  
Constraints like timing exceptions and internal clocks are used to map the compile point and its parents. They are used in the final timing report, and they are forward-annotated.

## Analyzing Compile Point Results

The software writes all timing and area results to a single log file in the implementation directory. You can check this file and the RTL and Technology views to determine if your design has met the goals for area and performance. You can also view and isolate the critical paths, search for and highlight design objects and crossprobe between the schematics and source files.

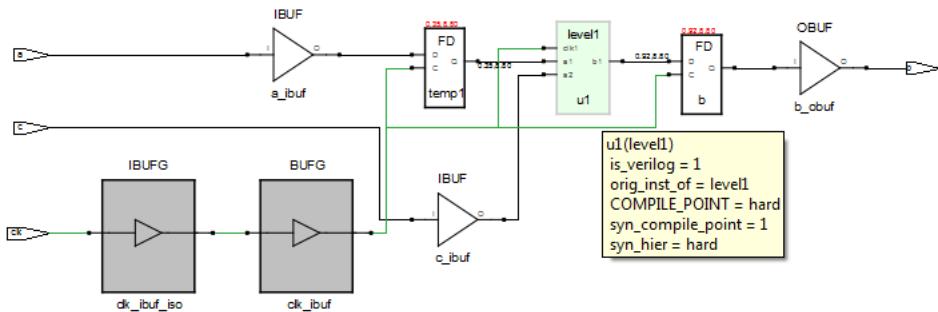
1. Check that the design meets the target frequency for the design.
2. Open the log file and check the following:
  - Check top-level and compile point boundary timing. You can also check this visually using the RTL and Technology view schematics. If you find negative slack, check the critical path. If the critical path crosses the compile point boundary, you might need to improve the compile point constraints.
  - If the design was resynthesized, check the Summary of Compile Points section to see if compile points were preserved or remapped.

| Summary of Compile Points : |          |                         |                          |                          |            |            |
|-----------------------------|----------|-------------------------|--------------------------|--------------------------|------------|------------|
| -----                       | Status   | Reason                  | Start Time               | End Time                 | Runtime    | CPU Time   |
| user_top                    | Remapped | Mapping options changed | Mon Mar 14 04:43:42 2011 | Mon Mar 14 04:44:27 2011 | 0h:00m:45s | 0h:00m:33s |

Note that this section reports black box compile points as Not Mapped, and lists the reason as Black Box.

- Review all warnings and determine which should be addressed and which can be ignored.
  - Review the area report in the log file and determine if the cell usage is acceptable for your design.
  - Check all DRC information.
3. Check other files:
- Check the individual compile point module log files. The tool creates a separate directory for each compile point module under the implementation directory. Check the compile point log file in this directory for synthesis information about the compile point synthesis run.
  - Check the compile point timing report. This report is located in the compile point results directory of the implementation directory for each compile point.
4. Check the RTL and Technology view schematics for a graphic view of the design logic. Even though instantiations of compile points do not have unique names in the output netlist, they have unique names in the Technology view. This is to facilitate timing analysis and the viewing of critical paths.

**Note:** Compile points of type {hard} and {locked, partition} are easily identified in the Technology view by their green coloring.



## 5. Fix any errors.

Remember that the mapper reports an error if synthesis at a parent level requires that interface changes be made to a locked compile point. The software does not change the compile point interface, even if changes are required to fix DRC violations.

## CHAPTER 6

# Working with IP

---

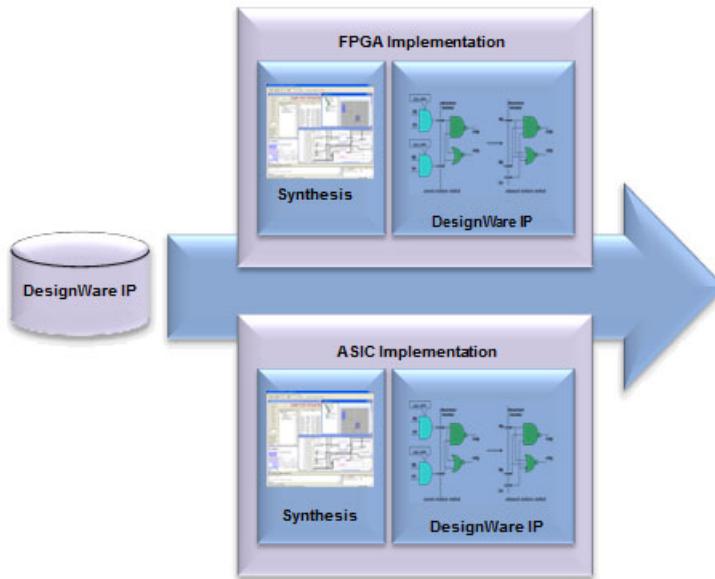
ASIC designs can include different types of IP (intellectual property) which you might want to integrate into your prototype, or revalidate by including in a prototype. This chapter describes the details about incorporating DesignWare® IP and Xilinx Vivado IP into prototyping designs, and discusses the use of encrypted IP.

- [Incorporating DesignWare IP](#), on page 340
- [Importing Vivado IP](#), on page 344
- [Working with Third-Party IP](#), on page 363
- [The Synopsys FPGA IP Encryption Flow](#), on page 368
- [Working with IEEE 1735 Encryption](#), on page 379
- [Encrypting IP Using OpenIP \(encryptIP\)](#), on page 400
- [Working with Synenc-encrypted IP](#), on page 407

Regardless of which IP encryption scheme you choose, be aware that encryption, like any security measure, may become vulnerable to unauthorized access and circumvention. The encryption technology and this documentation are supplied "As Is", and Synopsys makes no warranties or representations (whether express or implied) regarding the efficacy or security of such technology.

# Incorporating DesignWare IP

The prototyping tool lets you directly incorporate DesignWare IP, so you can reduce risk by using the same IP in the prototype and the final ASIC.



There are three kinds of DesignWare IP you can include automatically: functions, digital cores, and building blocks. See the following for details about incorporating these types of IP in your prototype:

- [Incorporating DesignWare IP Functions](#), on page 341
- [Incorporating DesignWare Library Building Block IP](#), on page 341
- [Incorporating DesignWare Digital Cores](#), on page 342
- [Working with Primitives Used by Synopsys IP](#), on page 343

You can also use DesignWare Foundation components with the unified compiler. For details, see [Including DesignWare Foundation Components](#), on page 114.

## Incorporating DesignWare IP Functions

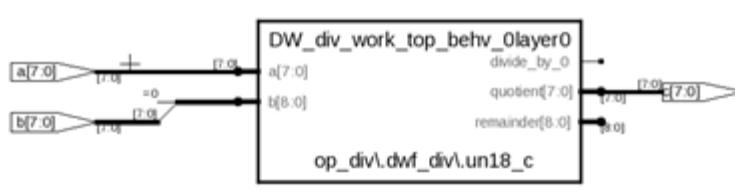
To incorporate DesignWare IP functions, specify the path to the DesignWare foundation library in the HDL code. For example:

```
library ieee,dware;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use dware.DW_foundation_arith.all;

entity top is
port (a : in signed (7 downto 0);
      b : in unsigned (7 downto 0);
      c : out signed(7 downto 0) );
end entity;

architecture behv of top is
begin
  c <= a / b ;
end architecture;
```

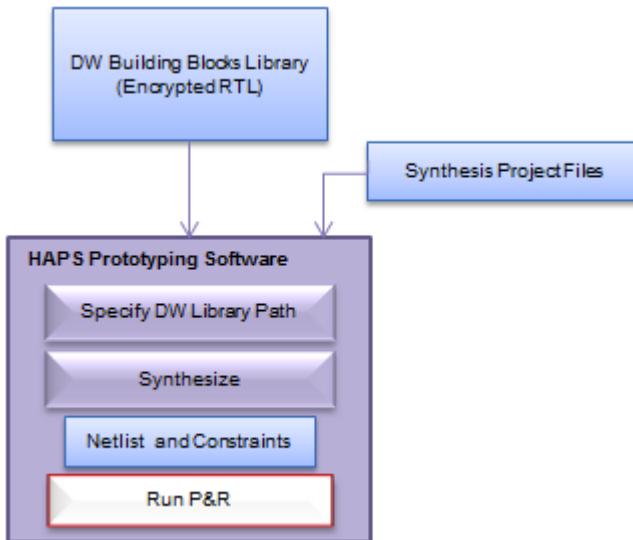
The tool automatically infers the appropriate logic for the DesignWare function during FPGA synthesis:



## Incorporating DesignWare Library Building Block IP

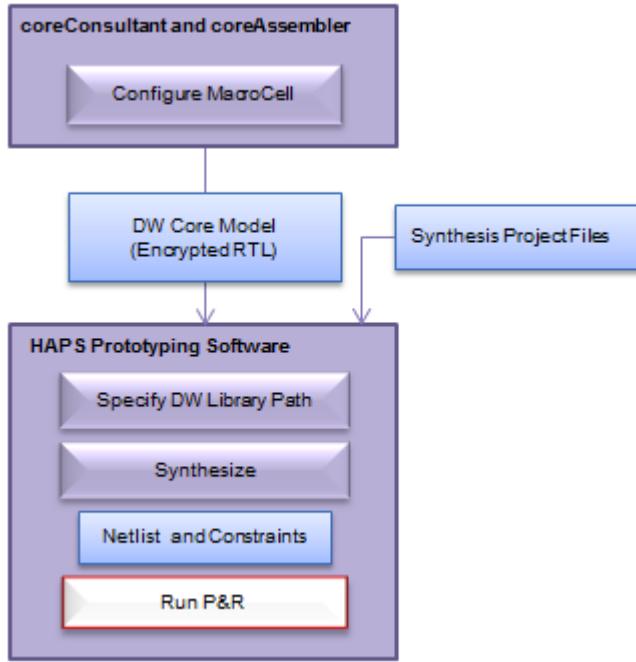
For DesignWare components and interface IP, the tool can access the encrypted RTL for the IP. Specify the path to the DesignWare libraries in the prototyping software, and the tool automatically infers the appropriate logic for the FPGA.

For a list of the available building blocks, see  
<http://www.synopsys.com/dw/buildingblock.php>.



## Incorporating DesignWare Digital Cores

For DesignWare digital cores, first configure them with the Synopsys coreConsultant tool. Include them in your design, and synthesize them directly. Use this methodology to incorporate DesignWare micro-controllers, AMBA coreTools synenc encrypted cores, and digital cores configured by Synopsys coreTools.



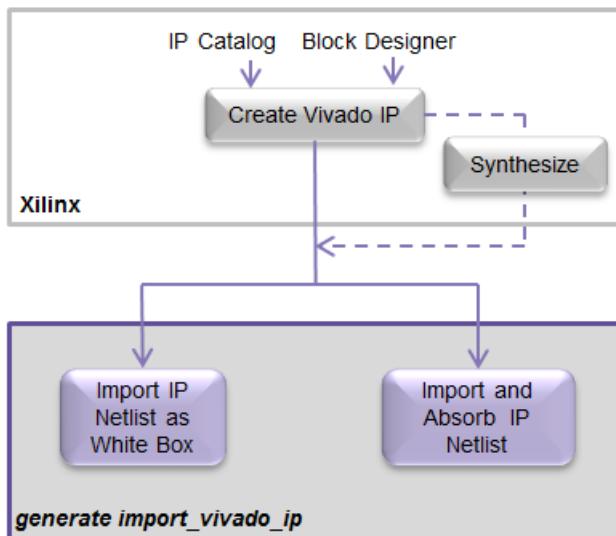
## Working with Primitives Used by Synopsys IP

Note that you cannot instantiate primitives that are used by Synopsys IP as part of your design. If you do, you get a placement error. For example, do not instantiate the USR\_ACCESE2 primitive, or you will get an error at the placement stage.

# Importing Vivado IP

The recommended approach for incorporating Vivado IP depends on the type of IP generated, interface IP (white box method) or data path IP (absorb method). In general, the RTL method is not recommended.

Instead, use one of the methods shown in the following figure which summarizes the flow:



For details about the flow, see the following topics:

- [Types of Vivado IP and Import Modes](#), on page 345
- [Creating Vivado IP](#), on page 345
- [Importing Vivado IP Netlists](#), on page 352
- [Importing IP from a Block Design File](#), on page 356
- [Including IP Collections](#), on page 360
- [Debugging XDC Conversion Messages](#), on page 362

## Types of Vivado IP and Import Modes

There are two categories of Vivado IP, which are handled differently when imported for synthesis:

- Interface IP

This category includes PCIE and other IP with complex constraints. The white box flow is recommended because the constraints might not be handled correctly otherwise.

In White Box mode, the IP netlist is used for timing estimations. The IP is not optimized and will be an empty box in the synthesized netlist, but the timing information is used to optimize around the IP. A DCP file is added to the place-and-route options file to fill in the IP functionality. This file contains all the original design constraints. This means that nothing is left untranslated and therefore avoids naming or translation issues introduced by netlist optimizations and constraint translation.

- Data path IP

This category includes IP with simpler constraints, like mults, DSPs, RAMs, and accumulators. Use the absorb method here, because these IPs can be easily integrated into the design and optimized.

With this method, the original timing constraints are converted from the original XDC format into FDC constraints. The post-synthesis netlist contains the entire design and the original XDC constraints for the IP blocks are forward annotated to the place-and-route options file.

The absorb and white box methods are specified as -mode options to the generate import\_vivado\_ip command. See [Importing Vivado IP Netlists, on page 352](#) for a step-by-step procedure.

## Creating Vivado IP

You can create Vivado IP using the Xilinx IP Catalog or Block Designer tools:

| Xilinx IP Creation Tools                               | For more information...                                             |
|--------------------------------------------------------|---------------------------------------------------------------------|
| IP Catalog<br>(individual IP blocks)                   | <a href="#">Creating Vivado IP with IP Catalog, on page 346</a>     |
| Block Designer<br>(multiple IP blocks for a subsystem) | <a href="#">Creating Vivado IP with Block Designer, on page 347</a> |

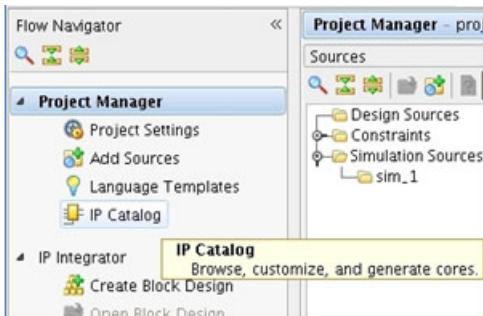
## Creating Vivado IP with IP Catalog

With this method, you select IP from a menu and create individual IP blocks. The following steps briefly describe how to generate Vivado IP with the Vivado IP Catalog; for comprehensive information, refer to the Xilinx documentation.

1. To run Vivado IP Catalog, start the Vivado tool, either from the GUI or with a Tcl script as shown below:

```
launch vivado -scriptt tc/ScriptName -run_dir extractionDirectory
```

2. Open the Vivado GUI, then select Create New Project and set the options to configure your project.
3. Once in the Vivado GUI, generate the IP by doing the following:
  - Click the IP Catalog button in the left column.



- Select your IP in IP Catalog and double-click to start configuration. The configuration wizard opens.
- Work through this wizard to configure your IP.
- Click Generate when you are done.

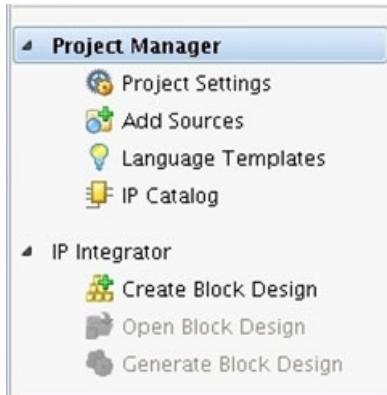
The tool generates the IP. The files include the IP RTL files, one or more xdc constraint files and an xml metadata file. The tool writes the IP RTL files to the ipcores directory.

You can now import the IP into your FPGA synthesis design, using the procedures described in [Importing Vivado IP Netlists, on page 352](#).

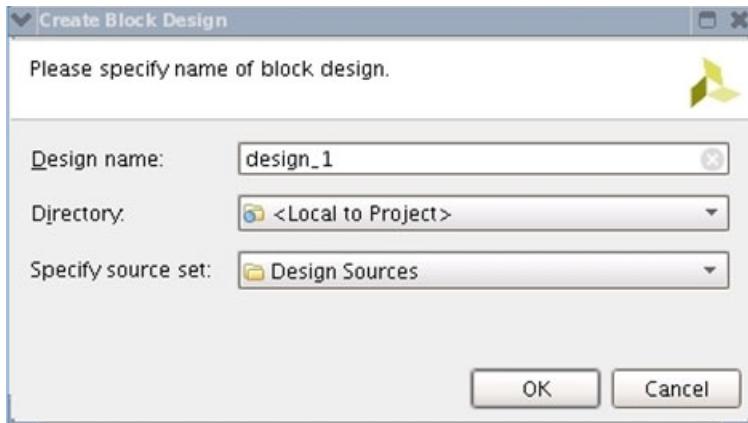
## Creating Vivado IP with Block Designer

With this method, you drag and drop IP components into a design panel, connect the components, validate the IP subsystem, and generate the IP out of context. The following steps briefly describe how to generate Vivado IP with the Block Designer tool; for comprehensive information, refer to the Xilinx documentation.

1. Select IP Integrator->Create Block Design from the left side of the Vivado GUI.

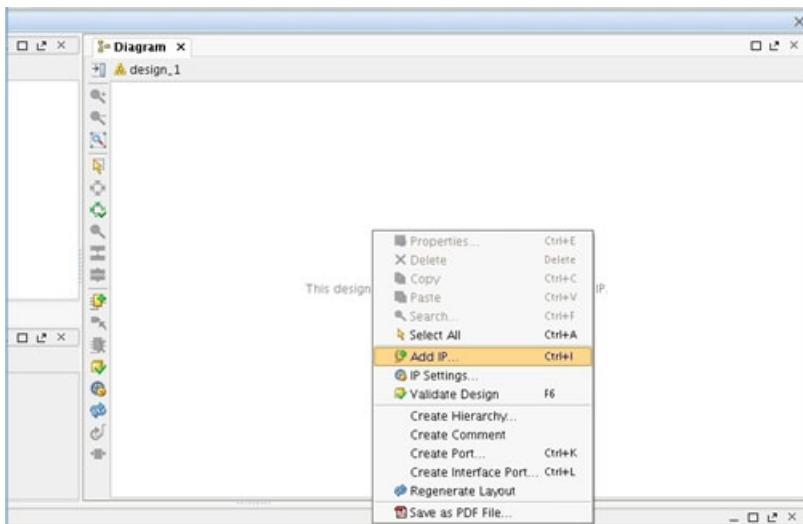


2. Specify the name of the block design and optionally a location for the generated IP files in Directory. Otherwise, the location defaults to your current Vivado directory.

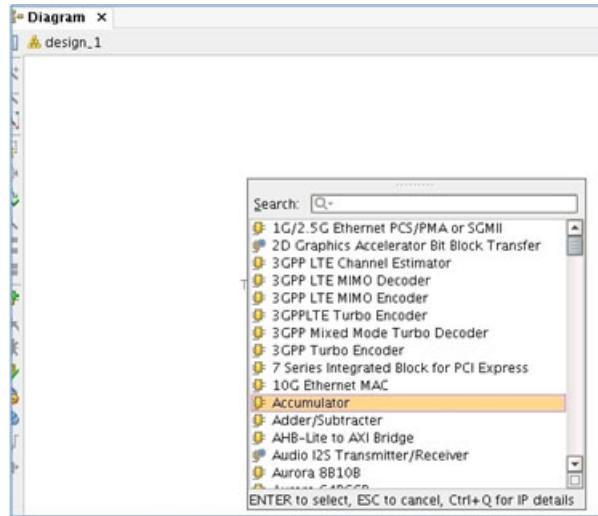


3. Add the IP you want to generate.

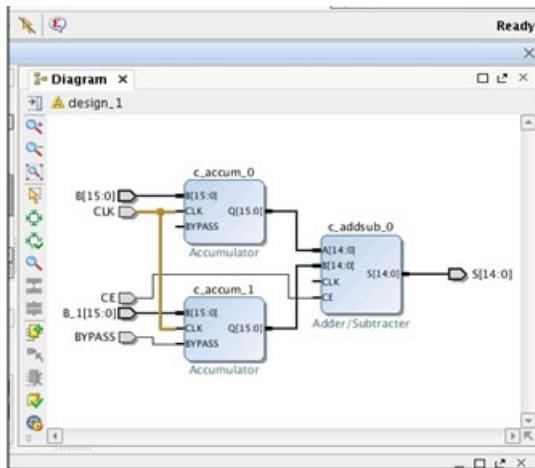
- On the right side of the Diagram panel, right-click and select Add IP.



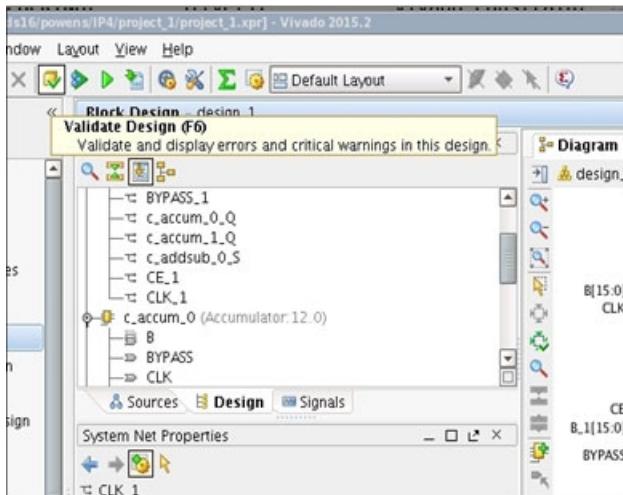
- Select the IP blocks to be placed on the Diagram panel.



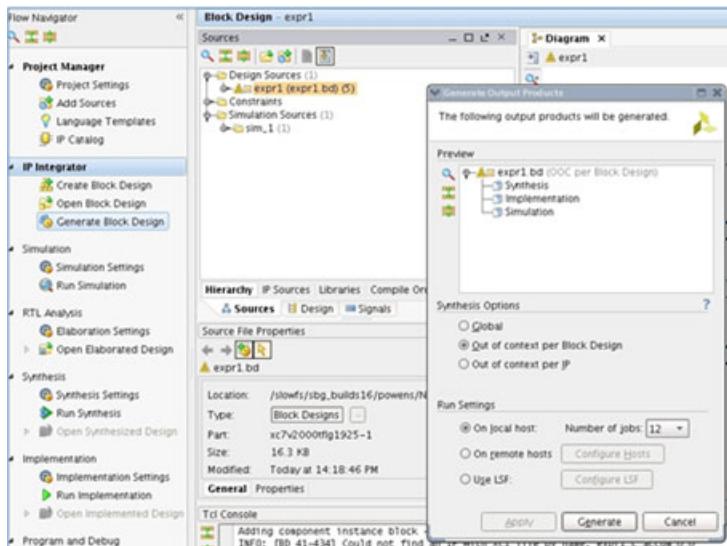
- Continue to add and connect the IP, making ports external as needed.



- Once your IP subsystem is complete, you can validate the design by selecting the folder with the green check.

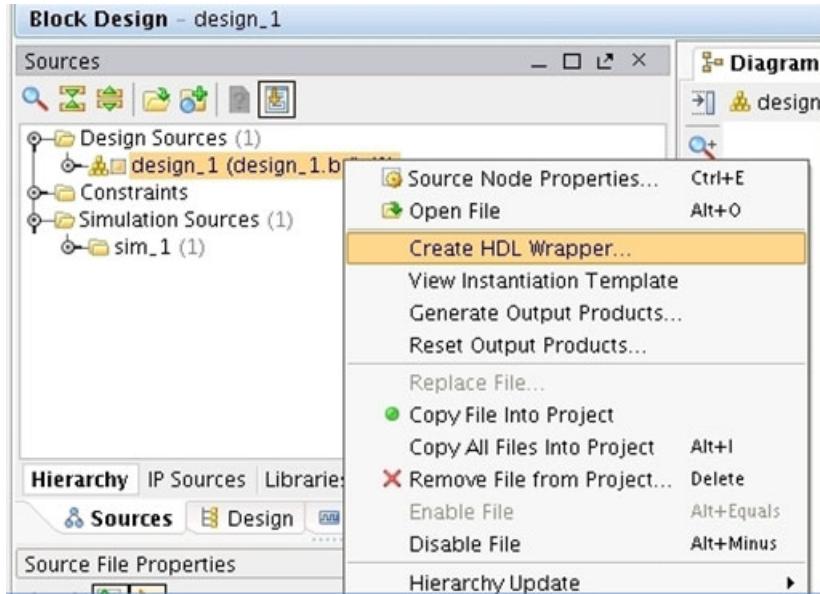


- Specify Out of Context per Block Design for the Synthesis option, then generate the block design (*topDesign.dcp*).



- Additionally, you can create an HDL wrapper for the IP subsystem.

On the Sources tab, highlight the design from the Design Sources hierarchy, then right-click and select Create HDL Wrapper.



## 6. Synthesize the design in Vivado.

Vivado creates individual DCP files for the IP blocks, as well as an XCI file to use with generate import\_vivado\_ip command.



It creates the following directories for the IP:

- A `bd/` sub-directory containing the `design.bd` file
- An `hdl/` directory for the wrappers
- An `ip/` directory containing the IPs for the subsystem

To process the block design (BD) file, see [Importing IP from a Block Design File, on page 356](#).

## Importing Vivado IP Netlists

You can automatically import Vivado IP into your current directory without having to manually create the IP files and translate constraints. Do not use the RTL flow, as this could cause issues with syntax and with Vivado RTL attributes not being recognized. Instead, use the absorb or white box methods described in the following procedure.

1. Use the Vivado tool to create IP with either the Vivado IP Catalog or Block Designer.

For details, see [Creating Vivado IP, on page 345](#).

2. From the prototyping tool, import the IP.

- Using the `generate import_vivado_ip` command at the command line, or specify it from the GUI ([Specifying Vivado IP Import Options from the GUI, on page 354](#)).

The following example shows the basic command:

```
generate import_vivado_ip -xci add_sub.xci -mode absorb  
-path ./ipcores
```

The basic options that you must specify are input (-xci), import mode (-mode absorb or -mode white\_box), and output directory (-path). The default location for the IP files is the current directory. For a discussion of the import modes, see [Types of Vivado IP and Import Modes, on page 345](#).

For both absorb and white box modes, the `generate import_vivado_ip` command converts XDC timing constraints to FDC format. The DCP file that contains both the constraints and netlist information is used for placement and routing.

- Optionally, specify other options from the command line.

For example, specify -dir to import a collection of IPs (see [Including IP Collections, on page 360](#)), -synthesize to run Vivado synthesis before importing the IP, or -dcp to specify a checkpoint file. For the complete command syntax, refer to [generate import\\_vivado\\_ip, on page 64](#) in the *Command Reference Manual*.

The tool creates one directory for each IP in the specified import directory. For example:

```
design_1_c_accum_0_0
design_1_c_accum_1_0
design_1_c_addsub_0_0
```

Each IP directory contains three files:

- `design_1_c_accum_0_0_src.sfl` (list of files)
  - `edf`
  - `verilog wrapper`
- `design_1_c_accum_0_0_constraint.sfl` (list of constraint files)
  - `fdc` (translated from IP `xdc`)
  - `macro.fdc` white-box mode)

For white box IP, the tool automatically sets `syn_macro` to 1 in the `macro.fdc` file, to prevent optimizations within the IP.

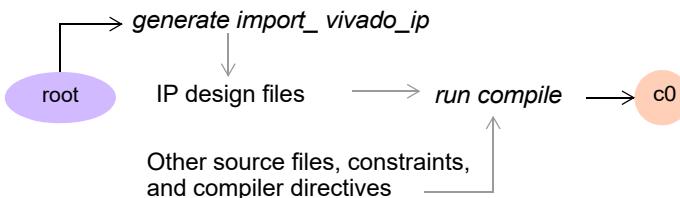
- `design_1_c_accum_0_0_par_options.tcl` (single line used by white box only)
 

```
read_checkpoint design_1_c_accum_0_0.dcp
```

### 3. Run compile with IP blocks added to the top-level design.

Add IP blocks to the top-level design. This is an example of the command:

```
run compile \
-srclist ipcores/design_1_c_accum_0_0/design_1_c_accum_0_0_src.sfl \
-srclist ipcores/design_1_c_accum_1_0/design_1_c_accum_1_0_src.sfl \
-srclist ipcores/design_1_c_addsub_0_0/design_1_c_addsub_0_0_src.sfl \
-srclist my_design.sfl \
-top_module design_1
```



### 4. Run pre-map with the IP constraints. For example:

```
run pre_map \
-fdclist ipcores/design_1_c_accum_0_0/design_1_c_accum_0_0_constraint.sfl \
-fdclist ipcores/design_1_c_accum_1_0/design_1_c_accum_1_0_constraint.sfl \
-fdclist ipcores/design_1_c_addsub_0_0/design_1_c_addsub_0_0_constraint.sfl
```

For absorb mode where Xilinx XDC constraints are translated to FDC constraints, check the log file to make sure there were no problems with

applying translated constraints. See [Debugging XDC Conversion Messages, on page 362](#) for more information.

5. Run map.
6. Export database files to run place and route. For example:

```
export vivado \
-path par_dir \
-vivado_option_file \
./ipcores/design_1_c_accum_0_0/design_1_c_accum_0_0_par_options.tcl \
-vivado_option_file \
./ipcores/design_1_c_accum_1_0/design_1_c_accum_1_0_par_options.tcl \
-vivado_option_file \
./ipcores/design_1_c_addsub_0_0/design_1_c_addsub_0_0_par_options.tcl
```

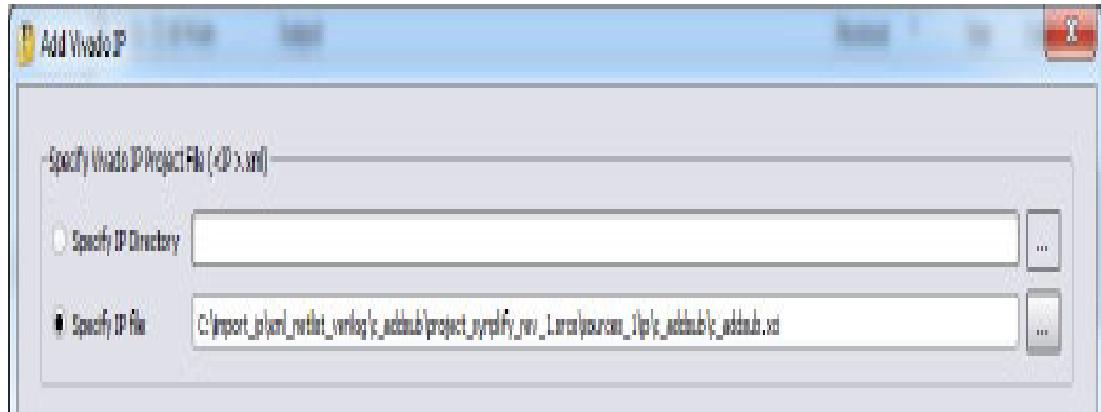
The IP par\_options.tcl files are sourced by the run\_vivado\_haps.tcl script.

7. Run place and route.

## Specifying Vivado IP Import Options from the GUI

To import an IP using the GUI, follow these steps:

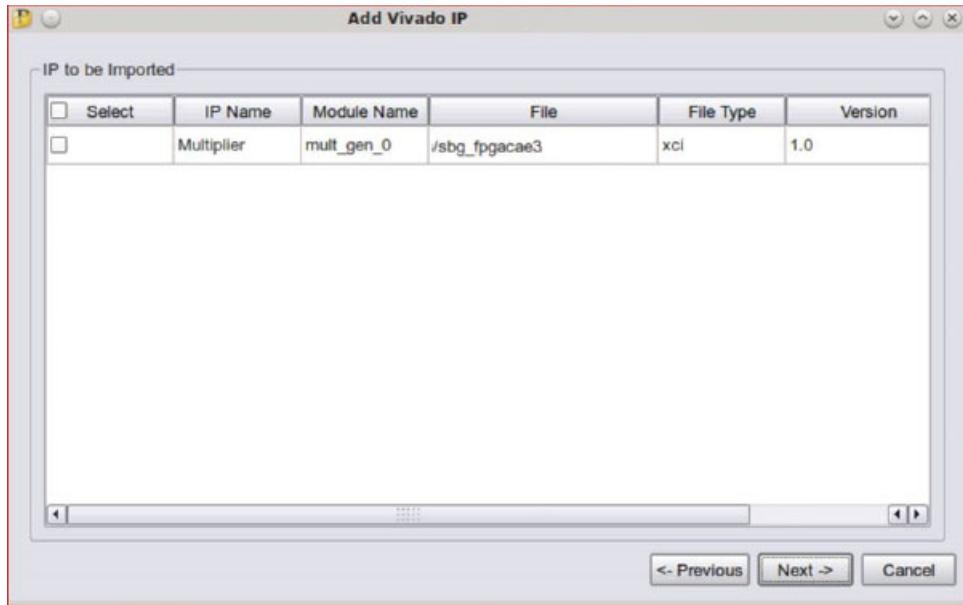
1. Select Tools->Import Vivado IP.
  - To import an IP directory, click Specify IP Directory. For one IP file, or a file list of IPs to import, click Specify IP File.



- Specify the XCI or DCP file and the name of the IP. Click Next.
- Optionally, you can click the Use Defaults button to choose the default synthesis mode for this Vivado run and directly go to the step to import the IP. That is:

- White Box mode to incorporate the IP.
  - VM format netlist is generated.
  - The location for the generated IP files (Import Location) is <projectDir>/ipcores.
2. Set these options on the next page.
- Select the IP block from the table on the second page and check the Settings column. In general, set it to Synthesize & Import. Click Next.

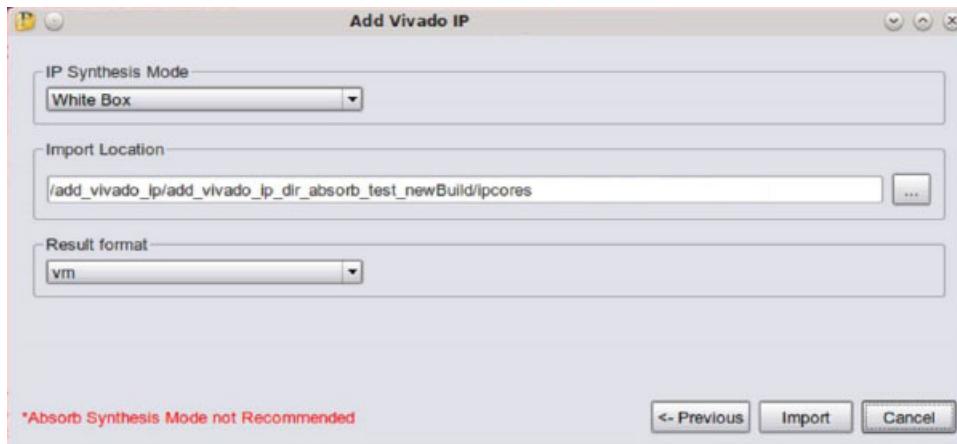
If you choose XCI as the input, then selecting Import calls Vivado synthesis if the DCP file is not already there. If a pre-generated DCP file exists, Vivado synthesis is not called unless you choose Synthesize & Import to force the DCP file to be regenerated.



3. On the last page, set the import mode and any optional arguments you might want for the run.
- For data path IP  
Set IP Synthesis Mode to Absorb if you want to optimize the IP netlist. In this case the IP is included in the final netlist, and the XDC file generated after synthesis (based on the original XDC file) is forward-annotated after mapping. Specify this mode to get the best

QoR, because the IP netlist is optimized with this mode. It is best for IP with relatively simple constraints.

- For interface IP  
Set IP Synthesis Mode to White Box if you only want the IP netlist for timing estimation. In this case the IP is not optimized and will be an empty box in the synthesized netlist. Use white box mode for IP with complex constraints. With this mode, the original IP constraints are preserved exactly, because the DCP file is added to the place-and-route options file to fill in the IP functionality and define the functions and constraints for the IP. This mode avoids potential naming issues that might occur because of optimizations or converted synthesis constraints.
- Specify a location for the generated IP files in Import Location.



4. Click Import.

IP files are generated as described in the procedure for [Importing Vivado IP Netlists, on page 352](#).

## Importing IP from a Block Design File

You can automatically process the Vivado block design (BD) file and convert it to a DCP file in the prototyping tool. After completion, you can use this DCP file to import the IP into a design database. To process the BD IP, you can run it from the GUI or in batch mode.

- [Importing IP from a DCP File \(Command Line Mode\), on page 357](#)

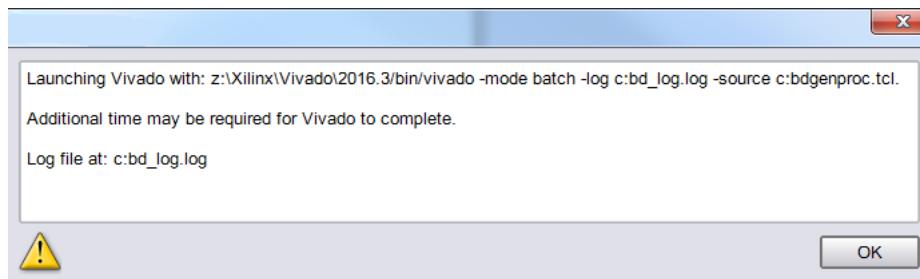
- [Importing IP from a DCP File \(GUI Mode\)](#), on page 358

## Importing IP from a DCP File (Command Line Mode)

The following steps describe how to use a Tcl procedure to automatically process the block design (BD) file and convert it to a DCP file in the prototyping tool using batch mode.

1. Set up the files.
  - Put the BD file in a separate directory than your working database directory; for example, bdsources/. Make sure you have write access to this directory.
  - Use the same version of the Vivado tool to process the BD file as was used to create the BD file.
2. Start the prototyping tool, and specify the BD file with the following command:

```
process_bd_ip -bd bdFileName
```



This launches Vivado and runs it in background mode. This processing might take time, depending on the number of IPs used to create the BD file for the design.

Vivado generates the IP in the source directory where the *design.bd* file is located

- bdsources/ contains *design.dcp* and *design\_stub.v*
- hdl/ contains *design\_wrapper.v*
- ip/ contains sub-directories for each IP block within the IP subsystem.

3. Add the IP to your design using generate import\_vivado\_ip and specifying the *design.dcp* file.
  - Specify the *design.dcp* file for the IP to import. Note that a list of all the IP blocks are displayed in the dialog box. Selecting only the *design.dcp* file is sufficient.
  - Optionally, specify whether to add the IP in absorb mode or white box mode. See *Importing Vivado IP Netlists*, on page 352 for details.
  - Optionally, specify the directory where the *design.bd* file is located.

If different versions of Vivado tool were used to generate and process the IP, you might encounter error messages specifying that the IP cannot be found.

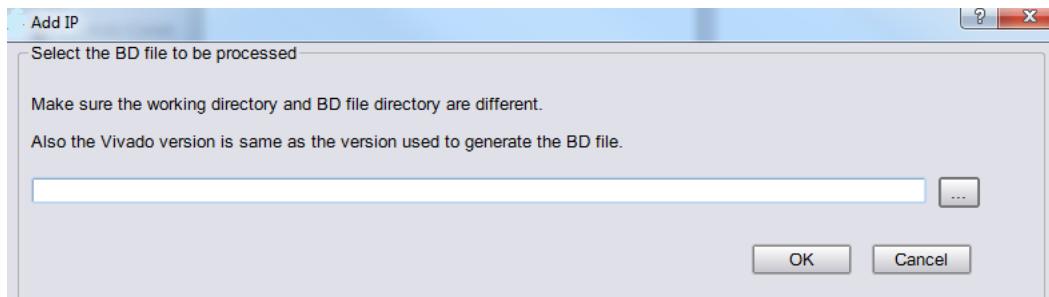
```
INFO: [BD 41-434] Could not find an IP with XCI file by name: design_1_microblaze_0
ERROR: [BD 41-50] Could not find an IP with the given vlnv: xilinx.com:ip:microblaze:9.6
ERROR: [BD 41-595] Failed to add ip repository block <microblaze_0>
ERROR: [BD 41-425] Failed to read Diagram <design_1> from BD file </slowfs/sbg.b
```

## Importing IP from a DCP File (GUI Mode)

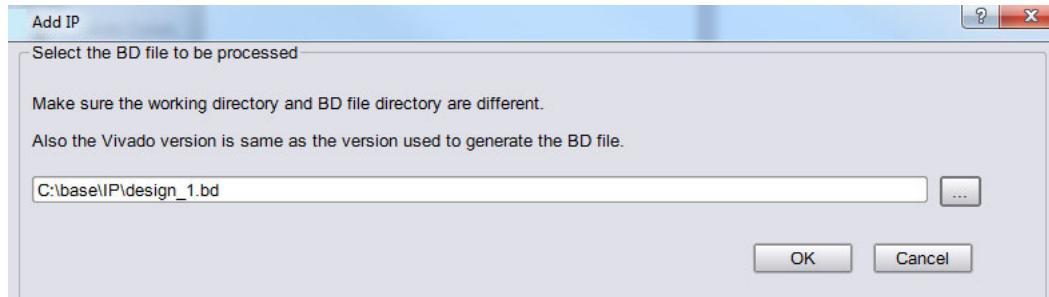
The following procedure shows how to use the GUI to generate a DCP file from a BD file and import the IP for prototyping.

1. Set up the files.
  - Put the BD file in a separate directory than your working database directory. Make sure you have write access to this directory.
  - Use the same version of the Vivado tool to process the BD file as was used to create the BD file.
2. Start the prototyping tool, and select Tools->Process BD IP from the main menu of the tool.

The Add IP dialog box opens.

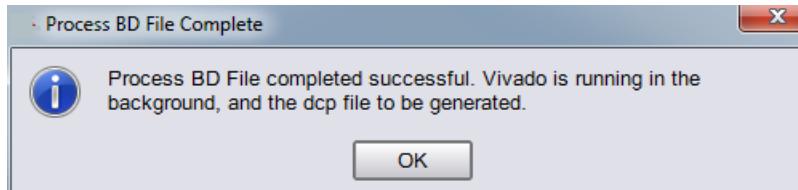


3. Select the specified BD IP file and click OK.



This launches Vivado tool in background mode to process the file. Processing might take some time, depending on the number of IPs used to create the BD file for the design.

4. Click OK in the dialog box that opens after BD file processing has successfully completed.



However, note that the Vivado tool might still be running in batch mode while it continues to generate the DCP file. The process generates a DCP file.

5. Check for the DCP file in the directory where the BD file is located.

Vivado might still be required for this conversion to complete, because all the IPs must be synthesized again to create the DCP file.

|                          |                   |             |
|--------------------------|-------------------|-------------|
| hdl                      | 1/20/2017 3:13 PM | File folder |
| hw_handoff               | 1/20/2017 3:13 PM | File folder |
| ip                       | 1/20/2017 3:13 PM | File folder |
| ipshared                 | 1/20/2017 3:13 PM | File folder |
| design_1.bd              | 1/20/2017 3:13 PM | BD File     |
| design_1.bxml            | 1/20/2017 3:13 PM | BXML File   |
| design_1.dcp             | 1/20/2017 3:37 PM | DCP File    |
| design_1_ooc.xdc         | 1/20/2017 3:13 PM | XDC File    |
| design_1_sim_netlist.v   | 1/20/2017 3:37 PM | V File      |
| design_1_sim_netlist.vhd | 1/20/2017 3:37 PM | VHDL File   |
| design_1_stub.v          | 1/20/2017 3:37 PM | V File      |
| design_1_stub.vhd        | 1/20/2017 3:37 PM | VHDL File   |

6. You can now import the DCP file into your design database. For details, see [Importing Vivado IP Netlists, on page 352](#).

## Including IP Collections

If you have a large number of IP blocks you want to add to a design, you can import them in one operation instead of importing each one independently. Follow the steps described below.

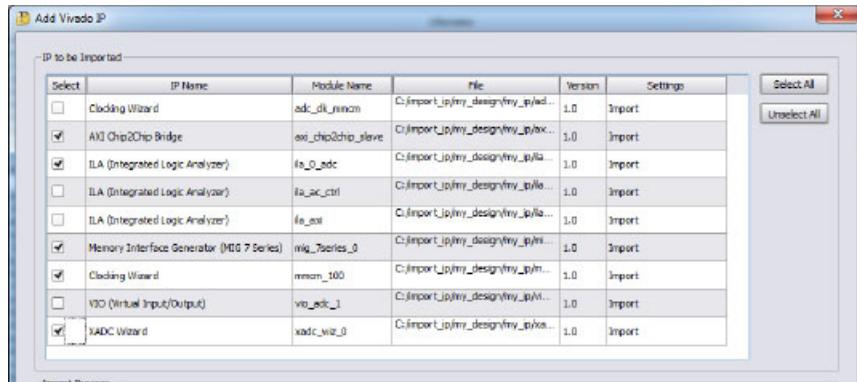
1. Generate the IP blocks using Vivado IP Catalog or Block Designer.  
This is required, if you want to import a collection of IP blocks.
2. Gather the blocks you want to import using these techniques:
  - Put the IP blocks in the same directory, so that you can specify the directory in the next step.
  - Create a text file that contains a list of IP blocks to be imported. For example, the ip.txt file lists these IP files:

```

./my_ip/adc_clk_mmcmm/adc_clk_mmcmm.xci
./my_ip/axi_chip2chip_slave/axi_chip2chip_slave.xci
./my_ip/ila_0_adc/ila_0_adc.xci
./my_ip/ila_ac_ctrl/ila_ac_ctrl.xci
./my_ip/ila_axi/ila_axi.xci
./my_ip/mig_7series_0/mig_7series_0.xci
./my_ip/mmcmm_100/mmcmm_100.xci
./my_ip/vio_adc_1/vio_adc_1.xci
./my_ip/xadc_wiz_0/xadc_wiz_0.xci

```

3. To import the IP from the GUI, select Tools->Add Vivado IP and do the following:
  - If you are using an IP directory, specify it in Specify IP Directory. If you are using a text file list, specify the file in Specify IP File. Click Next.
  - On the next page of the dialog box, select the IP blocks you want to import. For each block to be imported, set the Settings column to either Import or Synthesize and Import, depending on whether the netlist was already synthesized. Click Next.



- On the last page, set optional arguments for the run. You can specify whether you want the IP to be absorbed into the design or treated as a white box (IP Synthesis Mode options). You can also specify a location for the generated IP files (Import Location).

For more information about the command syntax, see [generate import\\_vivado\\_ip](#), on page 64 in the *ProtoCompiler Command Reference*.

- Click Import.

Once the collection of IP blocks have been generated in a file, run compile, pre-map, and map to complete the flow as described in [Importing Vivado IP Netlists, on page 352](#).

## Debugging XDC Conversion Messages

The tool automatically converts XDC constraints when you import Vivado IP using the methods described in [Importing Vivado IP, on page 344](#). You do not need to convert constraints manually for each IP. The tool reports any xdc constraints it fails to translate. Some of these situations are described here:

- Inability to translate a constraint

If the tool does not translate every constraint from the input XDC file, the untranslated constraints are flagged by messages like the one shown below. Manually modify any untranslated constraints.

Untranslated constraint at line 124 of XDC file.

- Possible errors in using original xdc file for place and route

If you use the converted fdc file for compile and map, but the original xdc file for place-and-route, be aware that the tool might rename a constrained element during synthesis if absorb mode is chosen for the IP.

The command generates a warning like the following when it detects this issue:

Warning: The following constraint has an element with a \_reg suffix in its name. Synthesis may rename this element, and this can cause the constraint not to work if you use the original Xilinx XDC file for P&R. It is suggested that you edit the XDC file and remove \_reg from the element name to avoid this problem.

- Incorrect translations

Manually modify the incorrectly translated constraints in the fdc file.

# Working with Third-Party IP

Use these methods to incorporate third-party IP blocks. IP blocks vary, and you might have to use different approaches, depending on the type of IP.

## Digital IP

1. Include the RTL source code, the encrypted source code, FPGA netlist, or hard IP.
  - If you are using RTL source code, it must be licensed or open-sourced through licensing schemes. If the RTL is the original SoC implementation, it will perform at lower speeds in the FPGA. If you are using vendor RTL that is customized for FPGA, it might have lesser functionality.
  - If you are using encrypted source code from a vendor, the encryption scheme must be supported by the prototyping software. Use an industry standard scheme like Synopsys synenc or IEEE P1735.
  - If you use an FPGA netlist provided by the vendor, the type of IP netlist affects the optimizations performed during synthesis. If you have a simple FPGA netlist that is targeted for a specific technology, the prototyping tool can freely optimize it. If you use a secure netlist, the prototyping tool can only use the timing information and optimize the logic around the IP, but not the IP itself. If the netlist is encrypted, the synthesis tool can only treat it as a black box; it can only be optimized by the place-and-route software.
  - If you have hard IP where the vendor has only provided test chips for prototyping, the prototyping software cannot implement it in the FPGA. You must remove the IP from the design, provide the top-level ports, and create top-level I/O ports to interface with the IP test chips.
2. Use the bottom-up flow described in [Using the Bottom-Up IP Flow, on page 364](#).

## Analog IP

Exclude analog IP from the FPGA prototype; the tool does not support it.

## Using the Bottom-Up IP Flow

You can run the flow with individual commands as described below, or put the commands into a script and run the script.

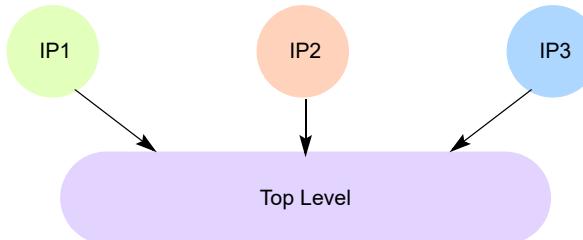
1. Compile each IP separately, with each IP as a top module.

For example:

```
run compile -srclist f1.txt -out ip1
run compile -srclist f2.txt -out ip2
run compile -srclist f3.txt -out ip3
```

2. Instantiate the IPs at the top level, and compile the top-level design using multiple -srcstate arguments:

```
run compile -srcstate IP1|ip1 -srcstate IP2|ip2 -srcstate IP3|ip3} -out top_stitch
```



3. Run through the rest of the flow as usual.

See [Running Bottom-Up Compile, on page 90](#) in the *User Guide*.

### Example Tcl Script for Bottom-Up IP Flow

```
##### sub2_2 #####
database load spc1reg1 -autocreate
source set_options0.tcl
run compile -srclist vlg/add_files_spreg.txt -top_module
    spc1_regs -out ip_spc1_regs1
export report -all -path spc1reg1_reports

##### sub2_1 #####
database load alu1 -autocreate
source set_options0.tcl
run compile -srclist vlg/add_files_alu.txt -top_module alu
    -out ip_alu1
export report -all -path alu1_reports
```

```
##### sub2 #####
database load dmux1 -autocreate
source set_options0.tcl
run compile -srclist vdl/add_files_dmux.txt -top_module
    data_mux_wrapper -out ip_data_mux1
export report -all -path dmux1_reports

##### sub1_2 #####
database load ins_decode1 -autocreate
source set_options0.tcl
run compile -srclist vdl/add_files_dec.txt -top_module ins_decode
    -out ip_ins_decode1
export report -all -path ins_decode1_reports

##### sub1_1 #####
database load ins_rom1 -autocreate
source set_options0.tcl
run compile -srclist vlg/add_files_rom.txt -top_module
    ins_rom_wrapper -out ip_ins_rom1
export report -all -path ins_rom1_reports

##### sub1 #####
database load iol -autocreate
source set_options0.tcl
run compile -srclist vlg/add_files_iol.txt
    -top_module io_wrapper -out ip_iol
export report -all -path iol_reports

##### sub1 #####
database load prgm_cntrl -autocreate
source set_options0.tcl
run compile -srclist vdl/add_files_cntr.txt
    -top_module prgm_cntr -out ip_prgm_cntrl
export report -all -path prgm_cntrl_reports

### sub1 #####
database load reg_file1 -autocreate
source set_options0.tcl
run compile -srclist vdl/add_files_reg_f.txt
    -top_module reg_file_wrapper -out ip_reg_file
export report
-all -path reg_file1_reports
```

```
##### top_stitch #####
database load top_stitch1 -autocreate
source set_options0.tcl
run compile -type srcstate -srclist vdl/add_files_8bituc.txt
  -src {spc1reg1|ip_spcl_regs1 alu1|ip_alu1 dmux1|ip_data_mux1
  ins_decode1|ip_ins_decode1 ins_rom1|ip_ins_rom1 io1|ip_io1
  prgm_cntrl1|ip_prgm_cntrl1 reg_file1|ip_reg_file1}
  -top_module eight_bit_uc -out top_stitch1

run pre_map

run map

export report -all -path top_reports
```

## Importing VCS Simulated Designs

Many ASIC designs for prototyping will have been simulated already. The following procedure shows you how to take a design that was simulated with the Synopsys VCS tool and get it ready for prototyping.

1. Use the dh\_module\_sources attribute to get a list of source files defining a module hierarchy. Edit the RTL and use ‘define’ to isolate RTL differences and to drive system configuration. The ‘defines’ in the following snippet specify other models for FPGA memories:

```
`define OR1200 ASIC
/////////////////////////////
// Typical configuration for an ASIC
//`ifdef OR1200 ASIC
// Target ASIC memories
//`define A_SSP
//`define A_SD
//`define A_STP

`define V_SSP
`define V_STP
```

2. Add parameterized models for memories and clocks with the add\_files command.
3. Load constraints.

4. If you have DesignWare IP, point to the DC directory. You can define the location with the \$SYNOPSYS variable.
5. Import the project files.
6. Run compile and map.

# The Synopsys FPGA IP Encryption Flow

The Synopsys FPGA IP encryption flow is a design flow that encourages interoperability while protecting IP implementations using encryption/decryption technologies. This flow offers the following advantages: interoperability, protection of IP, reuse of IP, and a standard flow for IP encryption.

See the following for information about the encryption flow:

- [Overview of the Synopsys FPGA IP Encryption Flow](#), on page 368
- [Preparing and Encrypting IP](#), on page 374

The following encryption standards are supported, but the recommended scheme is the IEEE 1735 standard.

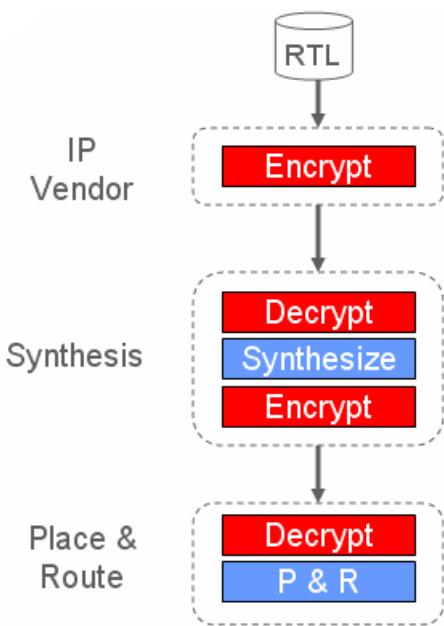
| Encryption Standard                         | Details                                                              |
|---------------------------------------------|----------------------------------------------------------------------|
| IEEE 1735-2014 with key-block (Recommended) | <a href="#">Working with IEEE 1735 Encryption</a> , on page 379      |
| OpenIP                                      | <a href="#">Encrypting IP Using OpenIP (encryptIP)</a> , on page 400 |
| Synenc-encrypted IP                         | <a href="#">Working with Synenc-encrypted IP</a> , on page 407       |

Regardless of which IP encryption scheme you choose, please be aware that encryption, like any security measure, may become vulnerable to unauthorized access and circumvention. The encryption technology and this documentation are supplied "As Is", and Synopsys makes no warranties or representations (whether express or implied) regarding the efficacy or security of such technology.

- [Preparing the IP Package](#), on page 375

## Overview of the Synopsys FPGA IP Encryption Flow

The complete flow for protecting IP requires a partnership between the IP author, Synopsys, and any other downstream tool vendor that consumes the IP. The following figure shows an FPGA synthesis flow, which requires the IP to be handed off from the synthesis tool to the place-and-route tool.

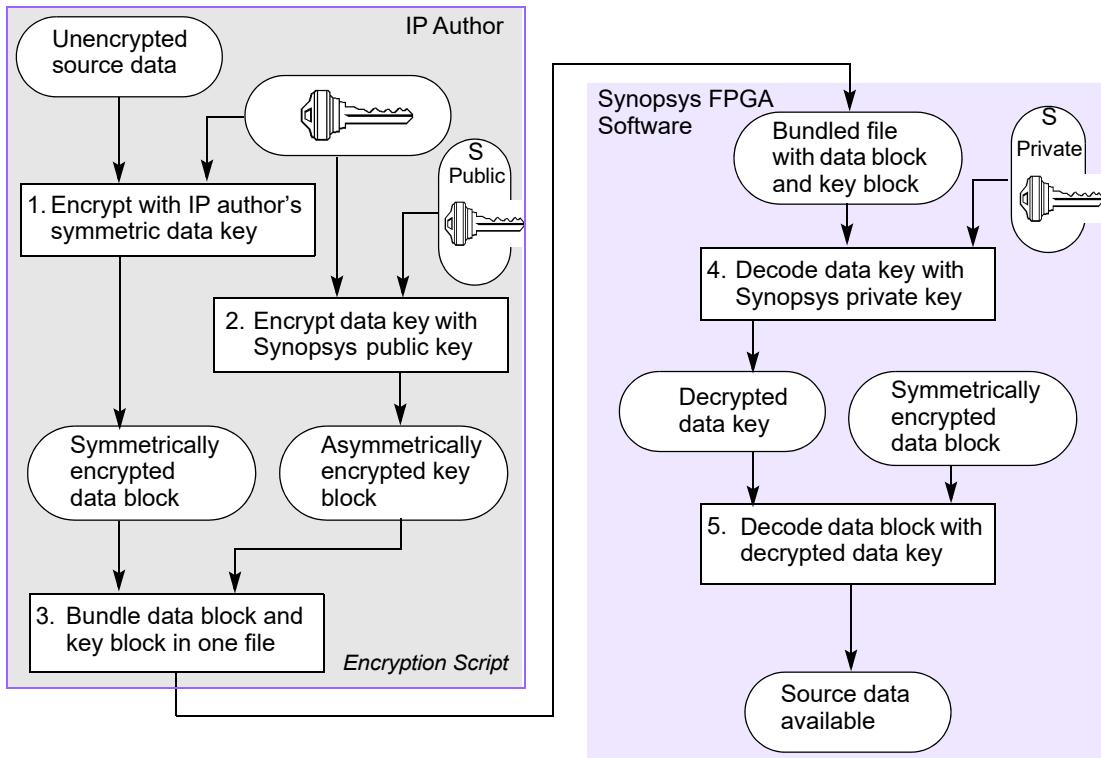


The details of the encryption and decryption hand-offs are described in [Encryption and Decryption, on page 369](#).

## Encryption and Decryption

There are two major classes of encryption/decryption algorithms: symmetric, and asymmetric. Each method has its own advantages and disadvantages. The approach for the Synopsys FPGA IP flow is a hybrid one that uses both asymmetric and symmetric encryption to leverage the strengths of each scheme.

The following figure illustrates the steps in this encryption/decryption methodology, showing the handoff from an IP author to a Synopsys FPGA tool.



The following describes these phases in more detail:

- [Data Encryption, on page 371](#)
- [Data Decryption, on page 373](#)
- [Re-encryption in the Synopsys FPGA IP Flow, on page 373](#)

Synopsys provides the IEEE 1735-2014 and OpenIP scripts to simplify and automate the process of encrypting data for the IP vendor.

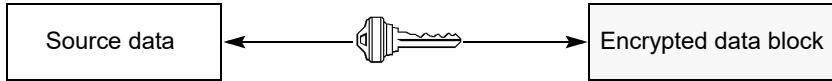
## Data Encryption

Data encryption is a three-step process that uses both symmetric and asymmetric encryption to encrypt the data.

### Step 1: Data Encryption (Symmetric)

Symmetric encryption uses a special number as a key to encrypt the files. The same key is used to decrypt the file, so the software must have access to the same key.

Symmetric Encryption/Decryption with the Same Data Key



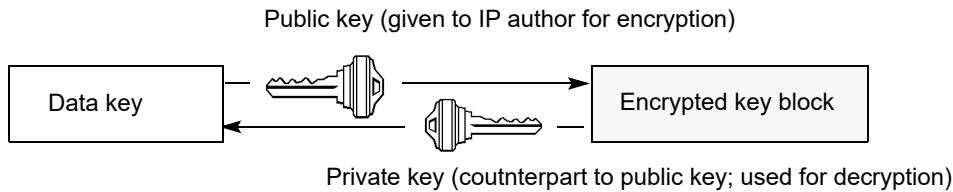
IP authors encrypt the IP data using their own symmetric key. This key is called the *data key*. The result of encoding is a *data block*. Using symmetric encryption offers two advantages to the IP author: fast data encryption because it is symmetric, and freedom to use any symmetric scheme they choose: Data Encryption Standard (DES), Triple DES, or Advanced Encryption Standard (AES).

### Step 2: Data Key Encryption (Asymmetric)

Next, the IP author encrypts the data key used to encode the IP block, and generates a *key block*. For this operation, the IP author uses RSA asymmetric encryption and the public key provided by the downstream consumer of the IP; for example Synopsys.

Asymmetric encryption uses different keys to encode and decode data. The IP consumer or tool vendor generates and makes a public key for encryption available to the IP author. The public key cannot be used for decryption. The IP consumer has a corresponding private key that is used to decrypt the data. The asymmetric encryption cipher used is RSA.

### Asymmetric Encryption and Decryption with Public and Private Keys

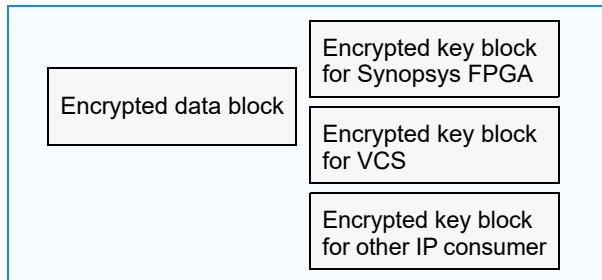


Asymmetric encryption offers the following advantages:

- Although asymmetric encryption is compute-intensive, the data key itself is small, so this is not time-intensive.
- The IP author can use public keys from different IP consumers to encrypt the IP data key (and therefore the IP data) for each IP consumer. This capability ensures that IP consistency is maintained, because there is no need for multiple copies. There is just one encrypted IP data block, with multiple keys, one for each specific IP consumer.
- Downstream IP consumers only need to pass their specific public key to the IP author, so that the data key can be encrypted for later retrieval.

### Step 3: Combining the Encrypted Data Block and Data Key

Example of Decryption Envelope



The IP author bundles the encrypted data block with the key block into one decryption envelope file for handoff to the IP consumer. Note that this methodology allows the IP author to create just one version of the IP, and add key blocks for each supported downstream consumer; for example, add key

blocks for place-and-route and simulation. Also, this approach eliminates the need to securely transmit the symmetric key, because this is included in the file. Security is maintained because both the key and the data are encrypted.

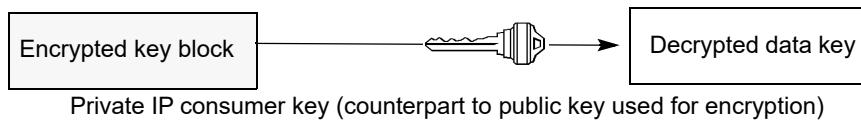
This is the point at which the IP author hands off the IP to the synthesis tool.

## Data Decryption

Decryption is a two-stage process.

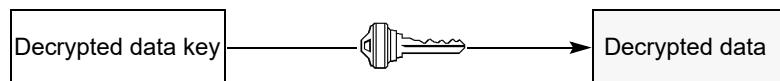
### Step 1: Data Key Decryption

In the FPGA tool, the first step as an IP consumer is to decrypt the data key from the IP author. The IP author encrypted the data key with the asymmetric public key from the FPGA tool, so the tool decodes this using the private key counterpart to the public key used for encryption, and extracts the data key.



### Step 2: IP Data Access

The second step is to use the extracted data key to access the IP data. As the data key is the original symmetric key used to encode the IP, the process is quick. The tool can now use the unencrypted IP.



## Re-encryption in the Synopsys FPGA IP Flow

After synthesis, the IP can be re-encrypted if the downstream IP consumer has adopted one of the Synopsys methodologies.

Re-encryption of the synthesized IP for other consumers downstream requires that the public key for the other consumer be included when the IP is first encrypted. If there is a key block included for a downstream consumer, that consumer can access the re-encrypted data. If such an agree-

ment is not in place, the IP is treated as a black box, and the output netlists, plaintext netlists, or encrypted netlists contain black boxes instead of the encrypted IP.

## Preparing and Encrypting IP

IP authors can use any of the supported Synopsys FPGA IP schemes to provide IP for prototypers and FPGA implementers to evaluate and use. Synopsys provides scripts to simplify this process.

To prepare and encrypt your IP as an IP author, do the following:

1. Gather your RTL files.

You only encrypt the RTL. You can encrypt any number of Verilog, VHDL, or mixed RTL files to form your encrypted IP, and each file can be encrypted in its entirety or in part.

2. Determine your file setup for each IP.

- Create a single set of files for the IP if your IP has no vendor-specific or vendor-optimized content and if the output method is supported by all intended consumers.
- Create multiple versions of your protected IP if you are using FPGA device-family specific RTL like architecture-specific instantiations, or if you optimized your RTL or constraints for use with a specific FPGA vendor device family or FPGA vendor.

3. Encrypt the files with the appropriate encryption script:

|                           |                                                                 |
|---------------------------|-----------------------------------------------------------------|
| IEEE 1735-2014 encryption | <a href="#">Working with IEEE 1735 Encryption</a> , on page 379 |
|---------------------------|-----------------------------------------------------------------|

|                   |                                                                      |
|-------------------|----------------------------------------------------------------------|
| OpenIP encryption | <a href="#">Encrypting IP Using OpenIP (encryptIP)</a> , on page 400 |
|-------------------|----------------------------------------------------------------------|

|                   |                                                                |
|-------------------|----------------------------------------------------------------|
| Synenc encryption | <a href="#">Working with Synenc-encrypted IP</a> , on page 407 |
|-------------------|----------------------------------------------------------------|

All the schemes uses a two-stage encryption process:

- First, encrypt your IP files using a symmetric encryption algorithm and your own data key to create an encrypted data block. See [Step 1: Data Encryption \(Symmetric\)](#), on page 371 for a general description.

- Next, encrypt the session key for the encrypted data block using an asymmetric algorithm and the Synopsys public key. All the Synopsys encryption methodologies support RSA encryption. See [Step 2: Data Key Encryption \(Assymmetric\), on page 371](#) for a general description.
4. Package your IP, as described in [Preparing the IP Package, on page 375](#).
  5. Verify that your IP works with the tool by going through the procedure that the user would use.
    - Start the tool and add the IP into a design.
    - Run the normal synthesis implementation or partitioning flow and check that the IP works.

## Preparing the IP Package

Do the following to package your IP and make it accessible to an authorized consumer like Synopsys:

1. Collect the files for the package.
  - Encrypt the files you need, as described in [Preparing and Encrypting IP, on page 374](#).
  - Make sure your package includes the files listed in [IP Package File List, on page 376](#).
  - Structure the files as described in [Suggested Directory Structure, on page 376](#).
2. If the IP package is intended for synthesis only, without subsystem assembly, create a compressed package for download, using one of these methods:
  - Create a compressed tarball (.tar.gz), which is a tar archive compressed with the gzip utility, using one of these commands:

```
tar cf -fileList | gzip -c > compressed-tarball
gtar -cf compressed-tarball fileList
```

Preserve the directory structure when you run gzip.
  - Create a zip file (zip) by running WinZip. WinZip archives and preserves your directory hierarchy.
3. Post the packaged IP on your website for downloading.

The user generally downloads the package and then untars or unzips it into a top-level directory. The IP can then be used by the tool.

4. Supply Synopsys with the following:
  - The URL for the download package.
  - Vendor and advertising information you wish to display on the Synopsys website. See *Supplying Vendor Information*, on page 377 for details.

## IP Package File List

Your IP package must contain the following files:

| Files                           | Description                                                                                                                                                                                                                     |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ipinfo.txt                      | Text file that lists the name of the IP, the version, restrictions for use, support contact information, and an email alias to request a license for the full RTL for your IP.                                                  |
| Documentation, preferably a PDF | Documents the IP, and includes detailed information about usage restrictions like vendor, device family, etc.                                                                                                                   |
| Readme                          | An optional text file that contains instructions on use of the IP for assembly and/or synthesis, and hints on how to use it correctly.                                                                                          |
| Encrypted HDL or EDIF           | Protected RTL for the IP, created using the Synopsys encryptIP script.                                                                                                                                                          |
| FDC constraints                 | Unencrypted design constraints for the IP. You need only maintain a single file for both the Synopsys synthesis tools, as the Synplify Pro software ignores any constraints that are specific to the Synplify Premier software. |

## Suggested Directory Structure

Follow these recommendations when you structure the IP package:

- Always use relative paths to reference a file.
- Always preserve directory structure when you run gzip.
- Place IP-XACT `.xml` files in the top-level directory or in a common subdirectory. You can have multiple files or a single file for the same component or variants of a component. However, it is preferred that you keep

all IP-XACT components that are in one library at the same directory level, even if it is many levels deep in the directory hierarchy.

## Supplying Vendor Information

To make your IP accessible for downloads and evaluation from the Synopsys tools, you must supply Synopsys with some vendor information as well as information for each of the cores or IPs to be used.

1. Supply Synopsys with the following general information to advertise your company and IP on the Synopsys website:

|                         |                                                                                                                                                                                                                                                              |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IP vendor name and logo | Your vendor name and logo for display.                                                                                                                                                                                                                       |
| Optional IP description | Short paragraph describing the IP and key features.                                                                                                                                                                                                          |
| Email alias             | Synopsys sends leads to this alias when evaluation cores are requested on the Synopsys IP website.                                                                                                                                                           |
| Website URL             | Unique URL for accessing IP. After the user has filled out lead information on the website, the Synopsys tool directs the user to this URL to download the IP. The lead form on your website can be pre-filled by prior arrangement with Synopsys Marketing. |

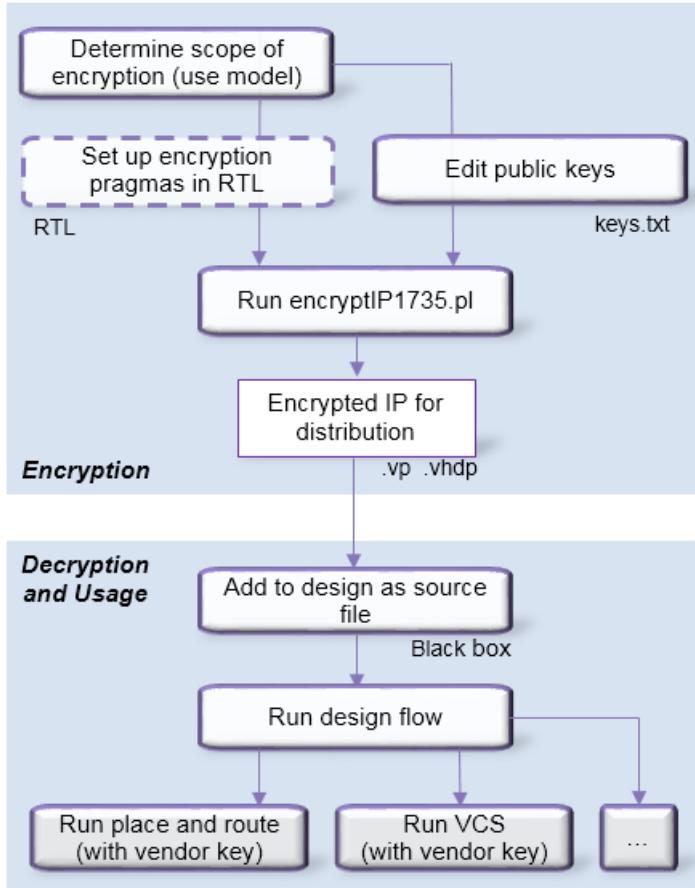
2. Supply Synopsys with the following information about each core or IP to be used:

|                          |                                                                                                                         |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------|
| IP name                  | Name of the IP.                                                                                                         |
| IP short description     | Sentence describing the IP, which is displayed in the summary view on the Synopsys website.                             |
| IP paragraph description | More detailed description of the IP, covering functional description and compatibility with other cores or peripherals. |
| Notes about usage        | Any other information, like licensing requirements                                                                      |

|                                    |                                                                                                                                                                                                      |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Core datasheet (HTML or PDF)       | Information about the characteristics, features, functions, and interfaces.                                                                                                                          |
| Supported FPGA vendors and devices | List of the targeted vendors and devices that the core supports.                                                                                                                                     |
| IP-XACT compatibility information  | List of the IP-XACT version number supported, the IP-XACT VLNV, and the IP-XACT VLNVs of all the bus definitions required for the core, along with a link to download each of these bus definitions. |

# Working with IEEE 1735 Encryption

The recommended method for encrypting IP is to use the IEEE 1735-2014 standard. The following figure summarizes the steps for encrypting and decrypting the IP.



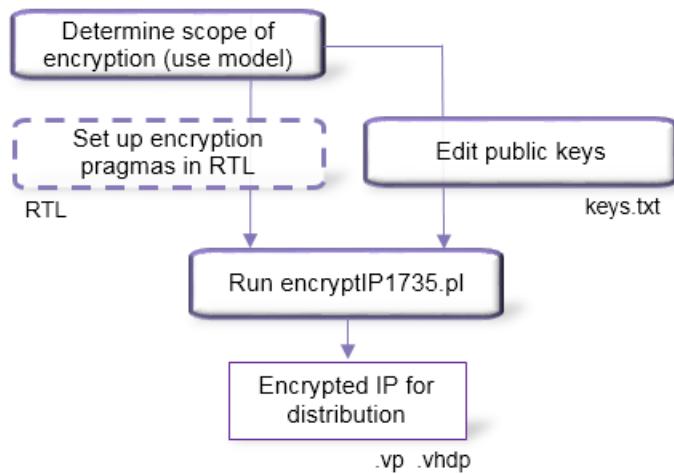
See the following for details about the stages in the flow shown above:

- [Encrypting IP Using IEEE 1735-2014](#), on page 380
- [Including IEEE 1735-Encrypted IP in a Synthesis Flow](#), on page 385
- [Including IEEE 1735-Encrypted IP in a Partitioned Design](#), on page 386

- [Adding Encrypted IP to a Unified Compile Database](#), on page 118 in the *User Guide*
- [The encryptP1735 Script](#), on page 388
- [IEEE 1735 Encryption Use Models](#), on page 391

## Encrypting IP Using IEEE 1735-2014

The following figure summarizes the steps an IP author must follow to encrypt and package data with the IEEE 1735-2014 standard. You can encrypt an entire file or parts of it. According to the encryption model, you add encryption pragmas to the source files and edit the encryption key file. The `encryptIP1735` Perl script, which is included in the tool hierarchy along with a default encryption key file, simplifies the process of encrypting the IP and generating an envelope around it.



1. Install the `encryptP1735.pl` script.
  - Make sure Perl is installed; otherwise you cannot run the script. Several commercial and free versions are available from <http://www.perl.org/get.html>.
2. Determine the scope of the encryption, and add encryption pragmas to the source files according to the encryption use model you use.
  - For Verilog source files, enclose the code you want to encrypt between `pragma protect begin` and `pragma protect end` statements.

- For VHDL source files, enclose the code you want to encrypt between protect begin and protect end statements.

**Encryption Model    Details of Use**

- |           |                                                                                                                                                                                                                                                                                      |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Full file | <ul style="list-style-type: none"><li>• Do not add any pragmas in the HDL because the entire source file is encrypted.</li><li>• Add the public key information in the keys.txt file, as described in step 3. Add information from each IP consumer that will have access.</li></ul> |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

See [Full-File Use Model](#), on page 392.

|                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Partial file with minimal pragmas                             | <ul style="list-style-type: none"><li>In the HDL, define pragma protect begin and pragma protect end pragmas for each data block you want to encrypt.</li><li>Add the public key information in the keys.txt file, as described in step 3. Add information for each IP consumer that will have access. The marked data is encrypted for all the keys in the keys file.</li></ul> <p>See <a href="#">Partial File Encryption with Minimal Pragmas Use Model</a>, on page 392.</p>                                                                                                                                                                                                                                                         |
| Partial file with standard pragmas<br><br>(Recommended model) | <ul style="list-style-type: none"><li>In the HDL, precede each data block you want to encrypt with the names of the keys which can access that block, using pragma protect statements. The values must match the values in the key file. Do not include the key block.</li><li>In the HDL, define pragma protect begin and pragma protect end pragmas for each data block you want to encrypt.</li><li>Add the public key information in the keys.txt file, as described in step 3. Add information for each IP consumer that will have access. Each block of data is only encrypted for the keys specified before the block.</li></ul> <p>See <a href="#">Partial File Encryption with Standard Pragmas Use Model</a>, on page 393.</p> |
| Partial file with IEEE pragmas                                | <ul style="list-style-type: none"><li>Define pragma protect begin and pragma protect end pragmas for each data block you want to encrypt.</li><li>In the HDL, precede each data block you want to encrypt with the names of the keys which can access that block, using pragma protect statements.</li><li>In the HDL, also include the public key information for each IP consumer that will have access.</li><li>You do not need the key file, because all the public key information is included in the HDL.</li></ul> <p>See <a href="#">Partial File Encryption with IEEE Pragmas Use Model</a>, on page 396.</p>                                                                                                                   |

- The recommended use model is the partial file with standard pragmas use model.
  - For an encryption scheme that is portable and can be used with any 1735 encryptor, use the partial file with IEEE pragmas use model.
3. Add public key information and other encryption pragma information to the keys.txt file to define which tools have access to the IP.

The keys.txt file contains the public key information and other encryption pragmas.

### Guidelines for All Use Models Except Partial File with IEEE Pragmas

- Copy the keys.txt file, which is included with the *installDir/lib/encryptP1735.pl* script, to a local directory so that you can edit the file.
- Obtain public key information for each tool. If you want the IP to be used by other tools, you must add key information for each tool that will be able to access the IP. For example, contact Xilinx for Vivado, or Synopsys for VCS. Note that the Synopsys VCS key is different from the one used by the Synopsys FPGA tools.
- Add the public key information obtained from the IP consumers to the keys.txt file.
- Add it after the default information, between the comment lines indicated in the file. The following example shows information for a dummy key. You must use actual information from the vendor.

```
// Add additional public keys below this line
`pragma protect key_keyowner="XYZ",
    key_keyname="DUMMY", key_method="rsa"
`pragma protect key_public_key
...<tool_vendor_public_key_information>...
// Add additional public keys above this line
```

- You must add the information for each downstream consumer at this time, as the encrypted IP cannot be passed to tools that do not have public key information included.

### Additional Guidelines for Partial File with Standard Pragmas Model

- Follow the general guidelines above.
- In addition, make sure to specify the version:

```
`pragma protect version=1
```

- In addition, add encryption information to the HDL. Before each block to be encrypted, add key\_keyowner entries, making sure that the information matches what is in the key block in the keys.txt file. For example:

```
`protect key_keyowner="Synopsys", key_keyname="SYNP15_1",
    key_method="rsa", key_block
```

### Guidelines for Partial File with IEEE Pragmas Model

- Do not use the keys.txt file. Add all encryption information in the HDL only.
- In the HDL, before each block to be encrypted, add the key\_keyowner and key\_block information for each tool allowed to access that block. For example:

```
'protect key_keyowner="Synopsys", key_keyname="SYNP15_1",
    key_method="rsa", key_block
`pragma protect key_public_key
...<vendor_public_key_information>...
```

#### 4. Run the encryptP1735.pl Perl script.

- Find the encryptP1735.pl script in the *installDir/lib* directory of the tool.
- Run the script. Below is an example of a command to run the script, where keys.txt is the file with the public keys and mylist is a file that contains a list of the files to encrypt. The list can name a single file to encrypt or multiple files; for multiple files, put each file name on a separate line. This command also prints messages to a log file.

```
perl encryptP1735.pl -list mylist -pk keys.txt -log encryptP1735.log
```

For the complete syntax to run the script, refer to [The encryptP1735 Script, on page 388](#).

This script automates the two-stage encryption process described in [The Synopsys FPGA IP Encryption Flow, on page 368](#). It first encrypts the IP files using a symmetric encryption algorithm and a random data key to create an encrypted data block. It then encrypts the session key for the encrypted data block using an asymmetric algorithm and the Synopsys public key. The tool currently uses RSA encryption.

The script then creates a “decryption envelope” file, so that the encrypted IP can be passed on and used by other tools. Verilog decryption envelopes have a .vp extension, and VHDL decryption envelopes have a .vhdp extension. For information about using the encrypted IP, see [Including IEEE 1735-Encrypted IP in a Synthesis Flow, on page 385](#) and [Including IEEE 1735-Encrypted IP in a Partitioned Design, on page 386](#).

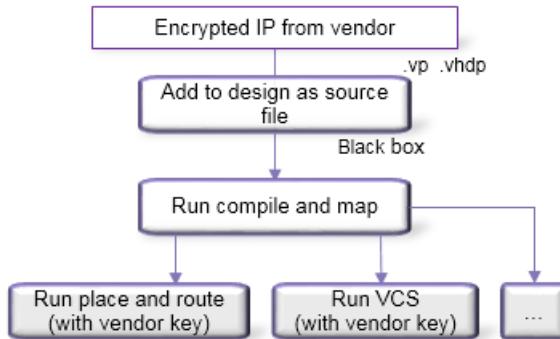
## IEEE 1735-Encryption Limitation

- Encryption on the top module is not supported. If top module encryption is conducted, the following error is observed:

The top module should not be encrypted

## Including IEEE 1735-Encrypted IP in a Synthesis Flow

This figure summarizes how to incorporate IP encrypted with IEEE 1735 in a synthesis implementation for single-FPGA designs. For information about encrypting the IP, see [Encrypting IP Using IEEE 1735-2014, on page 380](#).



- Add the encrypted file along with other source files when you compile the design.
  - To add a file, use the `-src` or `-srclist` arguments to the `run compile` command. For example:

```
run compile -srclist./myDesign/filelist.txt -top_module "myTop"
```

See [Specifying Source Files Using a Command Argument, on page 97](#) in the *User Guide* for more information about adding files.

- For a `.vp` Verilog file, make sure to specify that the file is a Verilog file, using the `-type` argument from the command line. Alternatively, you can right-click the file name in the GUI and specify the file type.

- Run through the design flow and compile and map as usual.

The tool decrypts the protected IP and uses it in the design, while protecting the IP data from disclosure. The IP remains encrypted as a

black box in compiled views and no LUT initialization values are displayed in mapped views.

3. To use the encrypted IP in other tools, like place and route or VCS simulation, the IP author must include the public keys for these tools when the IP was first encrypted.

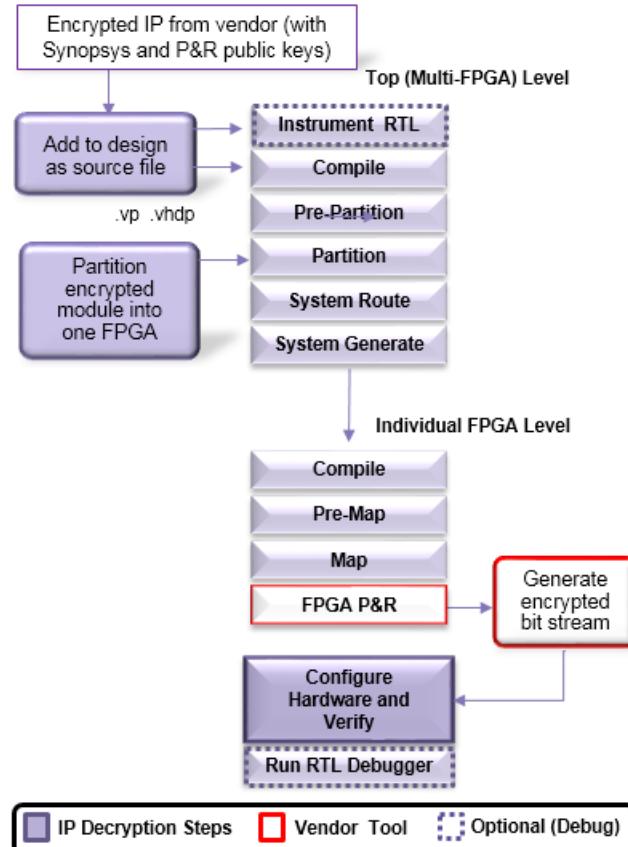
If the keys were included, the encrypted IP is passed on and can be used in the other tools. If the appropriate keys were not included, the IP must be re-encrypted with all the public keys required before it can be used. In this case the output from the FPGA tool includes a black box for the IP.

4. After placing and routing the design and generating the bit stream, you can encrypt the Xilinx bit files for added security, before loading them on the HAPS systems.

For more information, see [Encrypting Bit Files](#), on page 398.

## Including IEEE 1735-Encrypted IP in a Partitioned Design

The following figure shows the partitioning flow and summarizes how IP encrypted with IEEE 1735 is incorporated into it with a few extra steps or guidelines. For information about encrypting the IP to use in this flow, see [Encrypting IP Using IEEE 1735-2014](#), on page 380.



1. Add the encrypted file along with other source files at the compile or pre-instrument stage.
  - To add a file, use the `-src` or `-srclist` arguments to the run `compile` or `run pre_instrument` commands. For example:

```
run compile -srclist./myDesign/filelist.txt -top_module "myTop"
```

```
run pre_instrument-srclist./myDesign/filelist.txt -top_module "myTop"
```

See [Specifying Source Files Using a Command Argument, on page 97](#) in the *User Guide* for more information about adding files.

- For a `.vp` Verilog file, make sure to specify that the file is a Verilog file, using the `-type` argument from the command line. Alternatively, you can right-click the file name in the GUI and specify the file type.

2. If you are instrumenting the design, do not instrument IP signals.  
Make sure the instrumentation .idc file only includes instrumentation signals on non-encrypted modules.
3. At the partition stage, assign the module to a single FPGA bin to make sure that the encrypted module is not partitioned across multiple FPGAs.
4. Continue with the rest of the flow.
  - After partitioning, continue with the implementation flow for each partition, and synthesize, place and route each one.
  - After generating the bit stream, you can encrypt the Xilinx bit files for added security, before loading them on the HAPS systems.

For more information, see [Encrypting Bit Files, on page 398](#).

## The encryptP1735 Script

Run the encryptP1735 script directly from Perl.

The script supports different models for encrypting RTL files and accessing the encrypted information (see [IEEE 1735 Encryption Use Models, on page 391](#)). The use model is determined by how blocks are marked for encryption in the HDL, combined with the information in the public keys file, which is described in [Public Keys File, on page 389](#).

### Perl encryptP1735 Script Syntax

```
encryptP1735
  [-l | list /listofFiles]
  [-pk | public_keys keyFileName]
  [-sk | showkey]
  [-verbose]
  [-verilog]
  [-vhdl]
  [-log logFileName]
  [-h | -help]
```

The following table describes the command-line arguments.

---

|                                     |                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-l   list <i>listofFiles</i></b> | Specifies a list of the files to be encrypted; <i>listofFiles</i> is a list of the non-encrypted HDL input files with each filename entry on a separate line.                                                                                                                                                                                                                            |
| <b>-pk   public_keys</b>            | Specifies the public keys repository file. This file contains public keys for various tools. If the encryption envelope contains a key block with a particular keyowner and keyname, the script searches the public keys file to find a corresponding public key to use during key-block generation. See <a href="#">Public Keys File</a> , on page 389 for information about this file. |
| <b>-sk   showkey</b>                | When used, the encryption script displays the session key in use. This is useful when random keys are used and you want to know which key is being used.                                                                                                                                                                                                                                 |
| <b>-verbose</b>                     | Prints more detailed messages to the screen or log file.                                                                                                                                                                                                                                                                                                                                 |
| <b>-verilog</b>                     | Specifies Verilog HDL file format when filename does not include a default .v or .sv extension.                                                                                                                                                                                                                                                                                          |
| <b>-vhdl</b>                        | Specifies VHDL HDL file format when filename does not include a default .vhd or .vhdl extension.                                                                                                                                                                                                                                                                                         |
| <b>-log</b>                         | Prints messages to the specified log file.                                                                                                                                                                                                                                                                                                                                               |

---

## Public Keys File

The encryptP1735.pl encryption script requires public key information, which is specified in a designated file (-public\_keys or -pk) option). This file includes public keys for each of the tools that are allowed access to the envelope with the encrypted data. The default keys file is called keys.txt and is located with the encryption script in the lib directory of the tool installation.

```
// Use verilog pragma syntax in this file
`pragma protect version=1
`pragma protect author="default"
`pragma protect author_info="default"

`pragma protect key_keyowner="Synopsys", key_keyname="SYNP15_1",
key_method="rsa"
`pragma protect key_public_key
<public_key_block>

// Add additional public keys below this line
// Add additional public keys above this line
```

```

`pragma protect data_keyowner="default-ip-author"
`pragma protect data_keyname="default-ip-key"
`pragma protect data_method="aes128-cbc"

// End of file

```

For the partial file with all pragmas use model, the following pragma attribute values must match the corresponding values in the key-block section of the encryption envelope:

```

`pragma protect key_keyowner="Synopsys", key_keyname="SYNP15_1",
key_method="rsa"

```

For information on the pragmas supported, see *Pragmas Used by Encryption Scripts, on page 390*.

## Pragmas Used by Encryption Scripts

Both the encryptIP1735 and encryptIP (OpenIP) schemes use the pragmas described in the following tables. Note the following:

- The %%% protect directive must be placed at the exact beginning of a line.
- Exactly one white-space character must separate the %%% sequence from the command that follows.

The following table lists the pragmas used. In Verilog, the pragma must be preceded by the word `pragma`; this is not required in VHDL.

### General Pragmas

|                                    |                                                 |
|------------------------------------|-------------------------------------------------|
| %%% protect protected_file 1.0     | Line 1 of file with encrypted data              |
| %%% protect begin_protected        | Marks the beginning of data to be encrypted     |
| %%% protect end_protected          | Marks the end of data to be encrypted           |
| %%% protect comment <i>comment</i> | Single-line plain-text comment                  |
| %%% protect begin_comment          | Marks the beginning of plain-text comment block |
| %%% protect end_comment            | Marks the end of plain-text comment block       |

### Data Block Pragmas (IP Author Data Encryption Information)

|                                   |                         |
|-----------------------------------|-------------------------|
| %%% protect author= <i>string</i> | Lists name of IP author |
|-----------------------------------|-------------------------|

---

|                                                                             |                                                                                                                                                                                                                             |
|-----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>%%% protect version=1</code>                                          | Specifies encryption version; required only for IEEE 1735 Partial File with Standard Pragmas encryption use model                                                                                                           |
| <code>%%% protect<br/>data_method=des-cbc   3des-cbc<br/> aes128-cbc</code> | Specifies the DES encryption method used: <ul style="list-style-type: none"><li>• des-cbc: Data Encryption Standard (DES)</li><li>• 3des-cbc: Triple DES</li><li>• aes128-cbc: Advanced Encryption Standard (AES)</li></ul> |
| <code>%%% protect data_block</code>                                         | Immediately precedes the encrypted data block                                                                                                                                                                               |
| <b>Key Block Pragmas (IP Consumer Public Key Information)</b>               |                                                                                                                                                                                                                             |
| <code>%%% protect<br/>key_keyowner=string</code>                            | Lists the owner of the key                                                                                                                                                                                                  |
| <code>%%% protect<br/>key_keyname=string</code>                             | Name recognized by the Synopsys software to select the key block                                                                                                                                                            |
| <code>%%% protect key_method=string</code>                                  | Encryption algorithm (RSA currently supported)                                                                                                                                                                              |
| <code>w %%% protect key_block</code>                                        | Immediately precedes encrypted key block                                                                                                                                                                                    |

---

## IEEE 1735 Encryption Use Models

Encryption models determine the scope of what gets encrypted and who can access the files. The `encryptP1735` script lets you use these use models to encrypt RTL files:

| Encryption Model                                 | Details                                                                               |
|--------------------------------------------------|---------------------------------------------------------------------------------------|
| Full file                                        | <a href="#">Full-File Use Model</a> , on page 392                                     |
| Partial file with minimal pragmas                | <a href="#">Partial File Encryption with Minimal Pragmas Use Model</a> , on page 392  |
| Partial file with standard pragmas (Recommended) | <a href="#">Partial File Encryption with Standard Pragmas Use Model</a> , on page 393 |
| Partial file with IEEE pragmas                   | <a href="#">Partial File Encryption with IEEE Pragmas Use Model</a> , on page 396     |

## Full-File Use Model

Use this model to encrypt the entire file. The entire RTL file is included in the decryption envelope. This model uses the `keys.txt` file to define which consumers have access to the encrypted data.

- |          |                                                                                                                                                                                           |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RTL      | <ul style="list-style-type: none"><li>• Contains no encryption pragmas (entire file is encrypted)</li></ul>                                                                               |
| keys.txt | <ul style="list-style-type: none"><li>• Contains public key information for multiple downstream tools</li><li>• Owners of all public keys listed have access to the entire file</li></ul> |
- 

This Verilog example encrypts the whole file (`tb_encrypt.v`), including the module named `secret` that it contains.

```
module secret (a, b, clk);
    input a, clk;
    output b;
    reg b=0;

    always @ (posedge clk) begin
        b = a;
    end
endmodule
```

Run the script to encrypt the file:

```
perl encryptP1735.pl -list mylist -log encryptP1735.log
```

This command runs the script on a file (`mylist`), which lists the single Verilog file `tb_encrypt.v`. The command uses the default `keys.txt` file from the `lib` directory, and creates the decryption envelope file `tb_encrypt.vp`. Messages from the run are written to the `encryptP1735.log` file.

## Partial File Encryption with Minimal Pragmas Use Model

With this encryption model, `pragma protect begin` and `pragma protect end` pragmas are used to indicate the start and end points of encryption regions. The encryption region enclosed by the pragmas cannot be part of a module; it must include the entire module. This model is best suited for cases where every encryption region must be encrypted for every key in the key file. When using this model, you must encrypt the entire module. The `encryptP1735.pl` script checks all the begin and end pragmas and generates the decryption envelope for each tool specified in the `keys` file.

|          |                                                                                                                                                                                                |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RTL      | <ul style="list-style-type: none"><li>Contains individual blocks marked for encryption (partial file)</li><li>Contains no public key information</li></ul>                                     |
| keys.txt | <ul style="list-style-type: none"><li>Contains public key information for multiple downstream tools</li><li>Owners of all public keys listed have access to all encrypted RTL blocks</li></ul> |

To illustrate this use model, consider a single, Verilog file (`tb_encrypt.v`) to be encrypted with only begin and end pragmas. This file contains a single module named `secret`. Note that the pragmas are defined outside the module.

```
`pragma protect begin

module secret (a, b, clk);
    input a, clk;
    output b;
    reg b=0;
    always @(posedge clk) begin
        b = a;
    end
endmodule

`pragma protect end
```

When you run the script with the following command, it uses the begin and end pragmas specified in the RTL file to encrypt the file:

```
perl encryptP1735.pl -list mylist -pk keys.txt
```

Here, the list file (`mylist`) names the Verilog file `tb_encrypt.v`. The command encrypts the data between the begin and end pragmas and creates the decryption envelope file `tb_encrypt.vp` for all tools listed in the key file. No log file (-log option) is specified, so messages are not written to a log file.

## Partial File Encryption with Standard Pragmas Use Model

This is the recommended encryption use model. It is the most flexible because you can choose to encrypt individual blocks instead of the entire file and specify which tools can access each encrypted block on a per-block basis. When using this model, you must encrypt the entire module. This model requires a side file (`keys.txt`) that contains public key information for IP consumers.

- |          |                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RTL      | <ul style="list-style-type: none"><li>• Marks individual blocks for encryption (partial file)</li><li>• Includes public key encryption pragmas for each tool that is allowed access to an encrypted block, except the key itself (<code>key_public_key</code>)</li><li>• Key information must match the information in the <code>keys.txt</code> file</li><li>• Can allow different keys access to different blocks</li></ul> |
| keys.txt | <ul style="list-style-type: none"><li>• Contains public key information for multiple downstream tools</li><li>• Must include all public key encryption pragmas, as well as the public key itself</li><li>• Only those owners of public keys listed in the RTL before the encrypted block have access to that block; all public keys listed in <code>keys.txt</code> need not be used in the RTL</li></ul>                     |
- 

If there are conflicting pragmas defined in the RTL and the `keys.txt` file, the RTL pragma takes precedence over the corresponding pragma in the `keys.txt` file. For example, if `data_method` in the RTL is defined as `des-cbc` but the same pragma in the `keys.txt` file defines it as `aes128-cbc`, the RTL definition is used and copied to the decryption envelope:

```
data_method="des-cbc"
```

## Verilog Example

This example encrypts a single Verilog file (`tb_encrypt.v`). The file contains a module named `secret` and includes all the encryption-related pragmas in the RTL, with the exception of `key_public_key`. The pragmas are defined outside the module, to encrypt the entire module.

```
`pragma protect version=1
`pragma protect encoding=(enctype="base64")
`pragma protect author="author-a", author_info="author-a-details"
`pragma protect encrypt_agent="encryptP1735.pl",
encrypt_agent_info="Synplify encryption scripts"
`pragma protect key_keyowner="Synopsys", key_keyname="SYNP15_1",
key_method="rsa", key_block
`pragma protect
data_keyowner="ip-vendor-a", data_keyname="fpga-ip",
data_method="des-cbc"
`pragma protect begin
```

```

module secret (a, b, clk);
  input a, clk;
  output b;
  reg b=0;
  always @ (posedge clk) begin
    b = a;
  end
endmodule

`pragma protect end

```

The script is then run with the following command, where the list file (mylist) names a single file, tb\_encrypt.v. The command uses the default keys.txt file from the *installLocation/lib* directory as the public keys file to create the decryption envelope file tb\_encrypt.vp. No log file is specified, so messages from the run are not sent to a log file.

```
perl encryptP1735.pl -list mylist -pk keys.txt
```

## VHDL Example

This example partially encrypts a VHDL file (tb\_encrypt.vhd) where all encryption pragmas are specified in the file, except for key\_public\_key. The file contains a single entity/architecture pair named secret. For VHDL pragmas, just use the keyword protect. The pragmas must enclose the entire module.

```

library IEEE;
use IEEE.std_logic_1164.all;

entity secret is
  port (clk : in std_logic;
        a : in std_logic;
        b : out std_logic);
end entity;

`protect version=1
`protect author="author-a", author_info="author-a-details"
`protect encrypt_agent="encryptP1735.pl",
encrypt_agent_info="Synplify encryption scripts"
`protect encoding=(enctype="base64")
`protect key_keyowner="Synopsys", key_keyname="SYNP15_1",
key_method="rsa", key_block
`protect data_keyowner="ip-vendor-a", data_keyname="fpga-ip",
data_method="des-cbc"
`protect begin

```

```

architecture rtl of secret is
signal b_reg: std_logic;
begin
    process (clk) is
    begin
        if rising_edge(clk) then
            b_reg <= a;
        end if;
    end process;
    b <= b_reg;
end architecture;

`protect end

```

Encrypt the file with the following command, where the list file (mylist) names a single VHDL file, tb\_encrypt.vhd. The command uses the default keys.txt file from the directory *installLocation/lib* as the public keys file to create the decryption envelope file tb\_encrypt.vhdp. Messages are not captured in a log file.

```
perl encryptP1735.pl -list mylist -pk keys.txt
```

## Partial File Encryption with IEEE Pragmas Use Model

Like the partial file with standard pragmas model, this use model is flexible, but it does not require a side file with the key block information. This makes it the most portable model, because all the information, including the key block information for each IP consumer, is included in the source code. When using this model, you must encrypt the entire module.

- |          |                                                                                                                                                                                                                                                                                                                                               |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RTL      | <ul style="list-style-type: none"> <li>• Marks individual blocks for encryption (partial file)</li> <li>• Includes public key encryption pragmas for each tool that is allowed access to an encrypted block, including the key itself (<code>key_public_key</code>)</li> <li>• Can allow different keys access to different blocks</li> </ul> |
| keys.txt | <ul style="list-style-type: none"> <li>• Not required</li> </ul>                                                                                                                                                                                                                                                                              |

The pragmas must include the entire module.

```

module top (qa, qb, a, b, clk);
    input a, b, clk;
    output qa, qb;

    enc_and iand (qa, a, b, clk);
    enc_or ior (qb, a, b, clk);

endmodule

```

```
`pragma protect version=1
`pragma protect author="author-a", author_info="author-a-details"

`pragma protect key_keyowner="Synplicity", key_keyname="SYNP15_1",
key_method="rsa"

`pragma protect key_public_key
<public_key_block>

`pragma protect key_keyowner = "XYZ"
`pragma protect key_method = "rsa"
`pragma protect key_keyname = "XYZ8_001"

`pragma protect key_public_key
<public_key_block>

`pragma protect data_method="aes128-cbc"
`pragma protect begin

module enc_and (q, a, b, clk);
input a, b, clk;
output q;

reg q=0;
always @ (posedge clk) begin
    q = a & b;
end

endmodule
`pragma protect end

`pragma protect version=1
`pragma protect author="author-a", author_info="author-a-details"
`pragma protect key_keyowner="Synplicity", key_keyname="SYNP15_1",
key_method="rsa"
`pragma protect key_public_key
<public_key_block>

`pragma protect key_keyowner = "XYZ"
`pragma protect key_method = "rsa"
`pragma protect key_keyname = "XYZ8_001"
`pragma protect key_public_key
<public_key_block>

`pragma protect data_method="aes128-cbc"
`pragma protect begin

module enc_or (q, a, b, clk);
input a, b, clk;
output q;
```

```
reg q=0;
always @ (posedge clk) begin
    q = a | b;
end

endmodule
`pragma protect end
```

## Encrypting Bit Files

You can encrypt the bitstream to further protect the design. Xilinx FPGAs include on-chip Advanced Encryption Standard (AES) decryption and authentication logic, which allows you to encrypt the bit files.

There are two ways to save the encryption key files: the reprogrammable key method which uses dedicated BRAM, backed up by an external battery; or the eFuse-based, non-volatile method, where the key is permanently programmed once and cannot be changed. Refer to the Xilinx documentation for details.

The BRAM method is recommended, and the following procedure describes the steps to follow to use this method to encrypt the bit files:

1. To automatically generate the AES-encrypted bit stream in Vivado after place and route, add these constraints to the run\_vivado script:

```
set_property BITSTREAM.ENCRIPTION.ENCRIPT Yes [current_design]
set_property BITSTREAM.ENCRIPTION.ENCRIPTKEYSELECT bbram
    [current_design]
set_property BITSTREAM.ENCRIPTION.KEYFILE <.nkyfile> [current_design]
set_property BITSTREAM.GENERAL.COMPRESS {True} [current_design]
write_bitstream-force ${DesignName}_enc_comp.bit
```

This creates a \*.nkyfile with a random key.

Consult the Xilinx documentation, *UG570/UltraScale architecture Configuration User Guide*, for the most current information.

2. Use the Vivado Hardware Manager to load the \*.nkyfile key into the HAPS system over JTAG.

Once loaded, the button battery built into the HAPS system is used for backup storage of the encryption key. The battery is used to store the encryption key when you switch off the system. The details of the HAPS

system architecture, including the battery, are described in the HAPS hardware manuals.

The encryption key cannot be read out of the device by any means.

The key can be retained for several years, until you clear it. The encryption key can be only cleared after configuring BBRAM.

3. Use Confpro or an SD Card to load the encrypted bit files.

The FPGA now accepts the encrypted bit stream. It only loads encrypted bit files with the correct key and any unencrypted bit files.

4. To check that everything is working correctly before you distribute the HAPS system with the encrypted key, do the following:
  - Power cycle the HAPS system after loading the key into BBRAM.
  - Check that you can configure the HAPS system with an encrypted bit file.
  - Check that you can configure the HAPS system with a non-encrypted bit file.

# Encrypting IP Using OpenIP (encryptIP)

OpenIP encryption is a scheme developed by Synopsys and donated to the standards body. You can use it to encrypt modules or components, which can then be downloaded for evaluation or use by the Synopsys FPGA user. Synopsys provides a script (encryptIP) to encrypt your data with this scheme. The script is run with the encryptIP Perl command.

For details, see the following:

- [Encrypting IP with the OpenIP Scheme](#), on page 400
- [The encryptIP Script](#), on page 404

## Encrypting IP with the OpenIP Scheme

Synopsys provides a script to encrypt your data with the OpenIP scheme. The encryptIP Perl script is provided to IP vendors who wish to provide IP to synthesis users. The script automates the two-stage encryption process described in the Synopsys FPGA IP methodology ([The Synopsys FPGA IP Encryption Flow](#), on page 368). The following procedure shows you how to encrypt your data with the encryptIP script.

Do the following to use the encryptIP script to encrypt IP:

1. Install the encryptIP Perl script.
  - Install Perl on your machine. You cannot run the script if you do not have Perl installed.
  - Download the encryptIP Perl script from this SolvNetPlus article:  
<https://solvnetplus.synopsys.com/s/article/encryptIP-Perl-Script-1576173366118>.
2. Make sure that the encryptIP script specifies the decryption key and the matching key length:
  - Specify the symmetric data decryption key with the -k option. Optionally, you can also specify a symmetric encryption key in hexadecimal format with the -kx option.
  - Make sure you specify the right key length for the encryption algorithm with the -c option. For example, TEST1234 becomes a 64-bit key, so you specify the des-cbc algorithm.

See [The encryptIP Script, on page 404](#) or full details of the encryptIP syntax.

3. Make sure you specify the appropriate output method (-om) when you run the script.

This is important because the output method (-om) determines what is encrypted. If the output method is plaintext for example, the entire output netlist is unencrypted, and includes the IP netlist in an unencrypted and readable form. See [Specifying the Script Output Method for OpenIP Encryption, on page 402](#) for more information.

The script encrypts the IP with the standard symmetric encryption algorithm you specified, and produces a `data_block`. The data key used for encrypting the HDL is then encrypted with an asymmetric algorithm and the Synopsys public key, and produces a `key_block`. The `data_block` and the `key_block` are combined with the appropriate pragmas for the flow being used, and the script creates an encrypted HDL file. See [Encryption and Decryption, on page 369](#) for a general discussion, and [Example of encryptIP \(OpenIP\) Script Output, on page 405](#) for an example.

All other output files from synthesis, including `srm`, `srd`, and `srs` files, are encrypted using the same encryption method specified for the input to synthesis. Output constraints are not encrypted.

4. Run the encryptIP script on each RTL file you want to encrypt.

The following example encrypts the Verilog `plain_ip.v` file into an encrypted file called `protected_ip.v`, using AES128-cbc encryption. The session key is `MY_AES_SAMPLEKEY`. See [The encryptIP Script, on page 404](#) for details about the syntax and required parameters.

```
perl encryptIP -in plain_ip.v -out protected_ip.v -c aes128-cbc  
-k MY_AES_SAMPLEKEY -bd 16OCT2007 -om plaintext -v
```

First, it encrypts the IP files using a symmetric encryption algorithm and a random session or data key. This creates an encrypted data block.

Next, it encrypts the session key for the encrypted data block using an asymmetric algorithm and the public key for the FPGA software and any other public keys you might have added for other tools.

5. Check the encrypted RTL file to make sure that there is only one key block present.

## Specifying the Script Output Method for OpenIP Encryption

You can control access to the IP encrypted with OpenIP by setting the appropriate output method. You specify the output method using the `-om` parameter, as described in [The encryptIP Script, on page 404](#).

The output method mainly affects the output netlist. The following are guidelines for setting the output method for the `encryptIP` script, and detail the effects of different settings:

- When using the `encrypyIP` script, set `-om` to `persistent_key` if you have an agreement in place with Synopsys and want the output netlist to be encrypted.
- 6. Set `-om` to `plaintext` in the following cases:
  - If you want to allow the IP to be incorporated in a logic synthesis design

Setting the output method to `plaintext` allows the tool to synthesize, run gate-level simulations, place and route, and implement an FPGA (that includes the IP) on a board.
  - If you want the IP to be freely optimized by the synthesis tools

Although IP cores are already optimized, the synthesis tools can effect additional optimizations based on the design context in which it will be used. When the synthesis tool is allowed to optimize the IP, it can prune away IP logic that is unused or unnecessary in the current design context. Or take the case where the output of an instantiated IP core is timing-critical because it drives hundreds of user loads. If the synthesis tool can freely optimize, it can replicate sources within the core and fix the problem.
- 7. To let the IP be incorporated in a logic synthesis design, set `-om` to `plaintext` or `blackbox`.

Setting the output method to `plaintext` allows the tool to synthesize, run gate-level simulations, place and route, and implement an FPGA (that includes the IP) on a board. Setting the output method to `blackbox` does not allow the tool to run gate-level simulations or place and route the IP, because it only uses the port and connectivity information.
- 8. If you have set `-om` to `plaintext` and you want to specify individual cores as white boxes, set the `syn_macro` directive to 1 on the view for the IP.

Note that you must set this on the view, not the instance. When this is set, the tool treats the IP as a white box and only uses the timing and connection information from the IP. The synthesis tool maintains the IP boundary and only trims unused logic inside the IP.

9. During synthesis, the IP contents appear as a black box in the RTL view, irrespective of the output method selected. When the output method is set to plaintext, you can push down into the IP from the Technology view.
10. After synthesis, the output method affects the results in the following ways:
  - Output constraints for an IP are in the standard Synopsys format and are not encrypted.
  - The output method affects the contents of the output netlist and its format. This table summarizes the encryptIP behavior with different output methods.

| <b>Method (-om)</b> | <b>Output Netlist After Synthesis</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| blackbox            | <p>The output netlist contains the IP interface only, and no IP contents. It only includes IP ports and connections. The IPs are treated as black boxes, and there are no nets or instances shown inside the IP. This applies to all the netlist formats generated for different vendors, whether it is HDL (vm or vhm), EDIF (edf or edn), or vqm.</p> <p>Output constraints are not encrypted. Output resource utilization and timing information includes IP information.</p> <p>You cannot run gate-level simulation on the output netlist or place and route the IP, because there is no information about the contents of the IP.</p> |
| plaintext           | <p>The output netlist contains the unencrypted synthesized IP, which is completely readable (nothing is encrypted). With this method, you can synthesize, run gate-level simulation, place, route, and implement the IP on an FPGA on a board.</p>                                                                                                                                                                                                                                                                                                                                                                                          |
| persistent_key      | <p>The output netlist includes encrypted versions of the IP. The IP is re-encrypted using the same session key and cipher that was used to encrypt the IP. The encrypted IP can be passed to place and route if that tool also uses the OpenIP scheme.</p>                                                                                                                                                                                                                                                                                                                                                                                  |

## The encryptIP Script

The encryptIP script lets you encrypt data with the OpenIP scheme.

Download the script (<https://solvnet-plus.synopsys.com/s/article/encryptIP-Perl-Script-1576173366118>) and run it directly from Perl. The Perl command line syntax for running the script is as follows:

### Perl Script Syntax

```
encryptIP
  -in | input inputFile
  -out | output outputFileName
  -c | cipher "{des-cbc|3des-cbc |aes128-cbc}"
  -k | key symmetricEncryptionKeyInTextFormat
  -kx | keyx symmetricEncryptionKeyInHexadecimalFormat
  -bd | build_date ddmmmyyyy
  -om | outputmethod "{plaintext|blackbox |persistent_key}"
  -incv | includevendor vendorKeyBlock
  -dkn | datakeyname sessionKeyName
  -dko | datakeyowner sessionKeyOwner
  -a | author dataAuthor
  -v | verbose
```

You must specify all required parameters.

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -in   input   | Names the input RTL file to be encrypted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| -out   output | Names the output file generated after encryption.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| -c   cipher   | Specifies the symmetric encryption cipher. The key length must match the algorithm being used, with each character using 8 bits. <ul style="list-style-type: none"><li>• des-cbc specifies the Data Encryption Standard (DES); uses a 64-bit key.</li><li>• 3des-cbc specifies the Triple Data Encryption Standard (Triple DES); uses a 192-bit key.</li><li>• aes128-cbc specifies the Advanced Encryption Standard (AES Rijndael); uses a 128-bit key. See <a href="#">Encryption and Decryption , on page 369</a> for an overview.</li></ul> |
| -k   key      | Specifies the symmetric data decryption key used to encode your RTL data block. The key is in text format, and can be any string (e.g. ABCDEFG). The exact length of the key depends on the data method you use.                                                                                                                                                                                                                                                                                                                                |

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -kx   keyx*           | Optional parameter. Specifies the symmetric encryption key in hexadecimal format.                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| -bd   build_date      | Specifies a date (ddmmmyyyy), so that the IP only works in FPGA software released after that date. This option lets you force users to use newer Synopsys releases that contain more security features. Contact Synopsys if you need help in deciding what build date to use.                                                                                                                                                                                                                                                                          |
| -om   outputmethod    | Determines how the IP is treated in the output after synthesis: <ul style="list-style-type: none"><li>• plaintext specifies that the IP is unencrypted in the synthesis netlist.</li><li>• blackbox specifies that the IP is treated as a black box, and only interface information is in the output.</li></ul> <i>persistent_key</i> is the default setting and includes encrypted versions of the IP. See <a href="#">Specifying the Script Output Method for OpenIP Encryption , on page 402</a> for more information about the use of this option. |
| -incv   includevendor | Optional parameter that specifies a key block for an EDA vendor, so that IP can be read by the vendor tools. C                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| -dkn   datakeyname    | Specifies a string that denotes your session key, that was used to encrypt your IP.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| -dko   datakeyowner   | Optional parameter that names the owner of the session key. The value can be any string.                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| -a   author           | Optional parameter that names the author of the session key. The value can be any string.                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| -v   verbose          | Specifies that the script run in verbose mode.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

---

## Example of encryptIP (OpenIP) Script Output

The following is an example of the script output:

```
Key block with session key          Key block header  
with key information  
%%% protect protected_file 1.0  
<optional unencrypted HDL>  
  
%%% protect begin_protected  
%%% protect key_keyowner=Synopsys  
%%% protect key_keyname=SYNP05_001  
%%% protect key_block  
U9n263KwF7RWb8GSz7C+700tKshqQgTmb8UdRxISekIJdfon/  
...  
  
<other key blocks>  
  
%%% protect data_method=aes128-cbc  
%%% protect data_block  
UWhcm3CPmGz27DXAWQZF8rY7hSsvLwedXiP59HYZHJfoMIM/  
...  
...  
%%% protect end_protected  
  
<optional unencrypted HDL>  
  
Data block           Data block header with  
                    encryption method
```

For brief descriptions of the pragmas used in the output of the encryptIP script, see [Pragmas Used by Encryption Scripts, on page 390](#).

# Working with Synenc-encrypted IP

Synenc encryption is a proprietary Synopsys encryption scheme for RTL cores, and is used for encryption by many Synopsys products. It includes DesignWare library macrocells and proprietary RTL cores encrypted using Synopsys coreTools. On Linux platforms, the FPGA tool reads Synenc-encrypted IP. As long as the Synenc-encrypted data does not include licensed components, you can read in the IP and synthesize or prototype it in the FPGA tool.

The following steps describe how to use these encrypted cores:

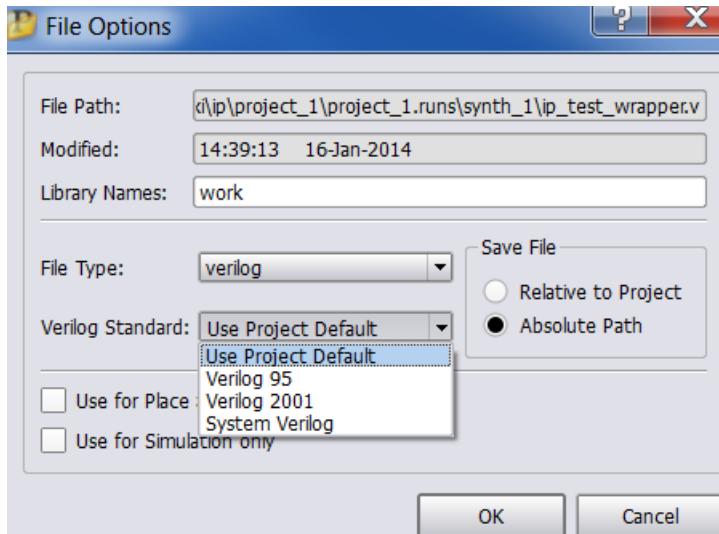
1. For cores created with coreConsultant, follow these steps:
  - Create a synthesis project file in coreConsultant. This file includes the Synenc-encrypted DesignWare core files in the correct order.
  - Add this project file to the synthesis project as a subproject, using the project -insert command.
2. For existing Synenc-encoded source files where you cannot go back to coreConsultant and create a project file, add the core files manually to the synthesis project.

File order is critical, because incorrect order causes the compiler to error out with a message about unknown macros. Ensure correct file order by doing one of the following:

- Use the original `lst` file from coreConsultant to set up your project. The `lst` file gives the proper order of files. This is the typical path to the `lst` file:  
**`ip_core_name/src/ip_core_name.lst`**
  - If the `lst` file is unavailable, make sure that the params and constants files for each core are listed first, and make sure that the undef file for the core is listed last.
3. Make sure that encrypted IP generated from coreConsultant is specified with the correct file types and Verilog standards to avoid a compiler error.

Use one of the following methods:

- Open the project file in the synthesis tool and highlight IP files. Right click and select File Options, then specify the applicable File Type and Verilog Standard on the dialog box.



All files are automatically updated in the project.

- Manually open the project file and edit the encrypted file with the proper file type and Verilog standard. For example, if the top.v file uses the Verilog 2001 standard specify the following:

```
add_file -vlog_std v2001 "./top.v"
```

Similarly, specify the following for an encrypted SystemVerilog top.sv file:

```
add_file -verilog -vlog_std sysv "./top.sv"
```

## CHAPTER 7

# HDL Compiler Language Constructs

---

This chapter describes the following language constructs supported by the ProtoCompiler products:

- [Verilog Language Support](#), on page 410
- [Verilog Synthesis Guidelines](#), on page 416
- [SystemVerilog Language Support](#), on page 428
- [VHDL Language Support](#), on page 439
- [VHDL Language Constructs](#), on page 442
- [VHDL 2008 Language Support](#), on page 451

# Verilog Language Support

This section describes Verilog support in the ProtoCompiler products. SystemVerilog support is described separately, in [SystemVerilog Language Support](#), on page 428. This section includes the following topics:

- [Support for Verilog Language Constructs](#), on page 410
- [Compiler Directives](#), on page 412
- [Verilog 2001 Language Support](#), on page 413
- [Verilog Synthesis Guidelines](#), on page 416
- [Verilog State Machines](#), on page 421
- [Instantiating Black Boxes in Verilog](#), on page 423
- [Hierarchical or Structural Verilog Designs](#), on page 424
- [Verilog Attribute and Directive Syntax](#), on page 427

## Support for Verilog Language Constructs

This section describes support for various Verilog language constructs:

- [Supported and Unsupported Verilog Constructs](#), on page 410
- [Ignored Verilog Language Constructs](#), on page 411

## Supported and Unsupported Verilog Constructs

The following table lists the supported and unsupported Verilog constructs. If the tool encounters an unsupported construct, it generates an error message and stops.

| Supported Verilog Constructs                                                                                                 | Unsupported Verilog Constructs                       |
|------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
| Net types:<br>wire, tri, tri0, tri1, wand, wor                                                                               | Net types:<br>trireg, triand, trior, charge strength |
| Register types:<br><ul style="list-style-type: none"><li>• reg, integer, time (64-bit reg)</li><li>• arrays of reg</li></ul> | Register types:<br>real                              |

|                                                                                                                                                                                                                                                                                                                                                               |                                                                                   |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| Gate primitive, module, and macromodule instantiations, and built-in switch primitives - pmos, cmos                                                                                                                                                                                                                                                           | Built-in unidirectional and bidirectional switches, and pull-up/pull-down         |
| inputs, outputs, and inout to a module                                                                                                                                                                                                                                                                                                                        | UDPs and specify blocks                                                           |
| All operators<br>+, -, *, /, %, **, <, >, <=, >=, ==, !=, ===, !==, ==?, !=?, &&,   , !, ~, &, ~&,  , ~ , ^~, ~^, ^, <<, >>, ?:, {}, {{ }}                                                                                                                                                                                                                    | Net names:<br>release net names (for simulation only)                             |
| Net names:<br>hierarchical net names                                                                                                                                                                                                                                                                                                                          |                                                                                   |
| Procedural statements:<br>assign, if-else-if, case, casex, casez, for, repeat, while, forever, begin, end, fork, join, disable                                                                                                                                                                                                                                | Procedural statements:<br>deassign, wait                                          |
| Procedural assignments: <ul style="list-style-type: none"><li>• always blocks, user tasks, user functions</li><li>• bind and force functions</li><li>• Blocking assignments =</li><li>• Non-blocking assignments &lt;=</li></ul> Do not use = with <= for the same register.<br>Use parameter override: # and defparam<br>(down one level of hierarchy only). | <ul style="list-style-type: none"><li>• Named events and event triggers</li></ul> |
| Continuous assignments                                                                                                                                                                                                                                                                                                                                        |                                                                                   |
| Compiler directives:<br>'begin_keywords, `celldefine, `define,<br>'endcelldefine, `endif, `end_keywords, `ifdef,<br>'ifndef, `else, `elsif, `include, `line, `undef                                                                                                                                                                                           |                                                                                   |
| Miscellaneous: <ul style="list-style-type: none"><li>• Parameter ranges</li><li>• Local declarations to begin-end block</li><li>• Variable indexing of bit vectors on the left and right sides of assignments</li></ul>                                                                                                                                       |                                                                                   |

## Ignored Verilog Language Constructs

When it encounters certain Verilog constructs, the tool ignores them and continues the synthesis run. The following constructs are ignored:

- delay, delay control, and drive strength

- scalared, vectored
- initial block
- Compiler directives (except for `begin\_keywords, `celldesign, `define, `ifdef, `ifndef, `else, `elsif, `endcelldefine, `endif, `end\_keywords, `include, `line, and `undef, which are supported)
- Calls to system tasks and system functions (they are only for simulation)

## Limitations

The following functions are not supported:

- Direct Entity Instantiation
- Configurations for Verilog Instances

## Compiler Directives

Compiler directives control compilation within an EDA environment. These directives are prefixed with an accent grave (`) or “tick mark.” Compiler directives are not Verilog statements and, as such, do not require the semicolon terminator. A compiler directive remains active until it is modified or disabled by another directive. The following table lists the supported compiler directives:

|                 |                                                                                                                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'begin_keywords | Specifies a pair of directives `begin_keywords and `end_keywords to identify keywords reserved within a block of source code, based on a specific version of IEEE Std 1364 or IEEE Std 1800. |
| 'celldesign     | Identifies the source code limited by 'cellname and 'endcelldefine as a cell.                                                                                                                |
| 'define         | Creates a macro for text substitution                                                                                                                                                        |
| 'else           | Indicates an alternative to the previous `ifdef or `ifndef condition                                                                                                                         |
| 'elsif          | Indicates an alternative to the previous `ifdef or `ifndef condition                                                                                                                         |
| 'endcelldefine  | Identifies the source code limited by 'cellname and 'endcelldefine as a cell.                                                                                                                |
| 'endif          | Indicates the end of an `ifdef or `ifndef conditional procedural statement                                                                                                                   |

---

|               |                                                                                                                                                                                              |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'end_keywords | Specifies a pair of directives `begin_keywords and `end_keywords to identify keywords reserved within a block of source code, based on a specific version of IEEE Std 1364 or IEEE Std 1800. |
| 'ifdef        | Executes a conditional procedural statement based on a defined macro                                                                                                                         |
| 'ifndef       | Executes a conditional procedural statement in the absence of a text macro                                                                                                                   |
| 'include      | File inclusion; the contents of the referenced file are inserted at the location of the 'include directive.                                                                                  |
| 'line         | Maintains the reference to line numbers for the original source or include file.                                                                                                             |
| 'undef        | Removes the definition of a previously defined text macro                                                                                                                                    |

---

## Verilog 2001 Language Support

You can choose the Verilog standard to use for a project or given files within a project: Verilog '95 or Verilog 2001. The following Verilog 2001 features are supported:

---

| Feature                                          | Description                                                                                                                                                                                                 |
|--------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Combined Data, Port Types (ANSI C-style Modules) | Module data and port type declarations can be combined for conciseness.                                                                                                                                     |
| Comma-separated Sensitivity List                 | Commas are allowed as separators in sensitivity lists (as in other Verilog lists).                                                                                                                          |
| Wildcards (*) in Sensitivity List                | Use @* or @(*) to include all signals in a procedural block to eliminate mismatches between RTL and post-synthesis simulation.                                                                              |
| Signed Signals                                   | Data types net and reg, module ports, integers of different bases and signals can all be signed. Signed signals can be assigned and compared. Signed operations can be performed for vectors of any length. |
| Inline Parameter Assignment by Name              | Assigns values to parameters by name, inline.                                                                                                                                                               |
| Constant Function                                | Builds complex values at elaboration time.                                                                                                                                                                  |

---

| Feature                                    | Description                                                                                                                                                  |
|--------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Configuration Blocks                       | Specifies a set of rules that defines the source description applied to an instance or module. See <a href="#">Configuration Limitations , on page 414</a> . |
| Localparams                                | A constant that cannot be redefined or modified.                                                                                                             |
| \$signed and \$unsigned Built-in Functions | Built-in Verilog 2001 function that converts types between signed and unsigned.                                                                              |
| \$clog2 Constant Math Function             | Returns the value of the log base-2 for the argument passed.                                                                                                 |
| Automatic Task Declaration                 | Dynamic allocation and release of storage for tasks.                                                                                                         |
| Multidimensional Arrays                    | Groups elements of the declared element type into multi-dimensional objects.                                                                                 |
| Variable Partial Select                    | Supports indexed part select expressions (+: and -:), which use a variable range to provide access to a word or part of a word.                              |
| Cross-Module Referencing                   | Accesses elements across modules. See <a href="#">Cross-Module Limitations , on page 415</a> .                                                               |
| ifndef and elsif Compiler Directives       | 'ifndef and 'elsif compiler directive support.                                                                                                               |

## Configuration Limitations

Configuration limitation include:

- Nested configuration is not supported.
- Top-level design name in the project file must match the top-level design name in the design clause of the configuration construct.
- A **use** clause with the cell name or library name omitted is not supported.
- The case where the configuration name and the module name are the same is not supported.
- Mixed HDL configuration is not supported.
- Multiple top levels in the design clause are not supported.
- Compiling the same configuration file to multiple libraries is not supported.

## Cross-Module Limitations

The following limitations currently exist with cross-module referencing:

- Cross-module referencing through an array of instances is not supported. In upward cross-module referencing, the reference must be an absolute path (an absolute path is always from the top-level module).
- Functions and tasks cannot be accessed through cross-module reference notation.
- You can only use cross-module referencing with Verilog/SystemVerilog elements. You cannot access VHDL elements with hierarchical references.

# Verilog Synthesis Guidelines

This section provides guidelines for synthesis using Verilog and covers the following topics:

- [General Synthesis Guidelines](#), on page 416
- [Library Support in Verilog](#), on page 417
- [Constant Function Syntax Restrictions](#), on page 418
- [Multi-dimensional Array Syntax Restrictions](#), on page 418
- [Sets and Resets](#), on page 419
- [SRL Inference](#), on page 420
- [Verilog State Machines](#), on page 421
- [Instantiating Black Boxes in Verilog](#), on page 423
- [Hierarchical or Structural Verilog Designs](#), on page 424
- [Verilog Attribute and Directive Syntax](#), on page 427

## General Synthesis Guidelines

Some general guidelines are presented here to help you synthesize your Verilog design.

- Top-level module – The synthesis tool picks the last module compiled that is not referenced in another module as the top-level module. Module selection can be overridden from the Verilog panel of the Implementation Options dialog box.
- Simulate your design before synthesis to expose logic errors. Logic errors that you do not catch are passed through the synthesis tool, and the synthesized results will contain the same logic errors.
- Simulate your design after placement and routing – Have the place-and-route tool generate a post placement and routing (timing-accurate) simulation netlist, and do a final simulation before programming your devices.
- Avoid asynchronous state machines – To use the synthesis tool for asynchronous state machines, make a netlist of technology primitives from your target library.

- Level-sensitive latches – For modeling level-sensitive latches, use continuous assignment statements.

## Library Support in Verilog

Verilog libraries are used to compile design units; this is similar to VHDL libraries. Use the libraries in Verilog to support mixed-HDL designs, where the VHDL design includes instances of a Verilog module that is compiled into a specific library. Library support in Verilog can be used with Verilog 2001 and SystemVerilog designs.

### Compiling Design Units into Libraries

By default, the Verilog source files are compiled into the work library. You can compile these Verilog source files into any user-defined library.

To compile a Verilog file into a user-defined library, specify the library to use for subsequent files (default is work). The *libName* argument is a string value that specifies the full path name to the indicated library. You can compile multiple files into the same library and also compile the same file into multiple libraries.

### Searching for Verilog Design Units in Mixed-HDL Designs

When a VHDL file references a Verilog design unit, the compiler first searches the corresponding library for which the VHDL file was compiled. If the Verilog design unit is not found in the user-defined library for which the VHDL file was compiled, the compiler searches the work library and then all the other Verilog libraries.

Therefore, to use a specific Verilog design unit in the VHDL file, compile the Verilog file into the same user-defined library for which the corresponding VHDL file was compiled. You cannot use the VHDL library clause for Verilog libraries.

### Specifying the Verilog Top-level Module

To set the Verilog top-level module for a user-defined library, include a -top\_module *moduleName* argument with one of the following commands:

- run compile

- run pre\_instrument
- report rtl\_diagnostics
- report syntax\_check

## Constant Function Syntax Restrictions

For Verilog 2001, the syntax for constant functions is identical to the existing function definitions in Verilog. Restrictions on constant functions are as follows:

- No hierachal references are allowed
- Any function calls inside constant functions must be constant functions
- System tasks inside constant functions are ignored
- System functions inside constant functions are illegal
- Any parameter references inside a constant function should be visible
- All identifiers, except arguments and parameters, should be local to the constant function
- Constant functions are illegal inside the scope of a generate statement

## Multi-dimensional Array Syntax Restrictions

For Verilog 2001, the following examples show multi-dimensional array syntax restrictions.

```
reg [3:0] arrayb [7:0][0:255];  
  
arrayb[1] = 0;  
// Illegal Syntax - Attempt to write to elements [1][0]..[1][255]  
  
arrayb[1][12:31] = 0;  
// Illegal Syntax - Attempt to write to elements [1][12]..[1][31]  
  
arrayb[1][0] = 0;  
// Okay. Assigns 32'b0 to the word referenced by indices [1][0]  
  
arrayb[22][8] = 0;  
// Semantic Error, There is no word 8 in 2nd dimension.
```

When using multi-dimension arrays, the association is always from right-to-left while declarations are left-to-right.

## Sets and Resets

A set signal is an input to a flip-flop that, when activated, sets the state of the flip-flop to a logic one. Asynchronous sets take place independent of the clock, whereas synchronous sets only occur on an active clock edge.

A reset signal is an input to a flip-flop that, when activated, sets the state of the flip-flop to a logic zero. Asynchronous resets take place independent of the clock, whereas synchronous resets take place only at an active clock edge.

### Asynchronous Sets and Resets

Asynchronous sets and resets are independent of the clock. When active, they set flip-flop outputs to one or zero (respectively), without requiring an active clock edge. Therefore, list them in the event control of the always block, so that they trigger the always block to execute, and so that you can take the appropriate action when they become active. The event-control syntax is:

```
always @ (edgeKeyword clockSignal or edgeKeyword resetSignal or  
edgeKeyword setSignal )
```

### Synchronous Sets and Resets

Synchronous sets and resets set flip-flop outputs to logic 1 or 0 (respectively) on an active clock edge.

Do not list the set and reset signal names in the event expression of an always block so they do not trigger the always block to execute upon changing. Instead, trigger the always block on the active clock edge, and check the reset and set inside the always block first.

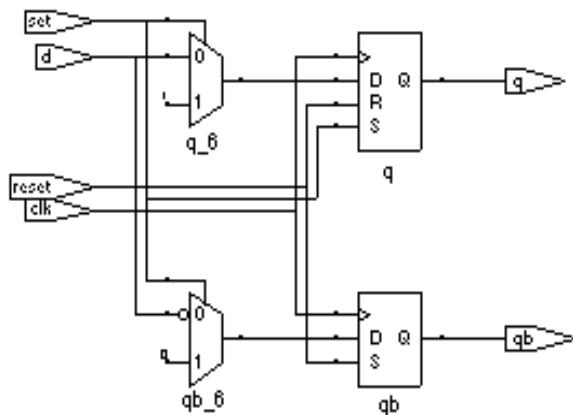
## RTL View Primitives

The Verilog compiler can detect and extract the following flip-flops with synchronous sets and resets and display them in the RTL schematic view:

- sdffr – flip-flop with synchronous reset
- sdffs – flip-flop with synchronous set
- sdffrs – flip-flop with both synchronous set and reset
- sdffpat – vectored flip-flop with synchronous set/reset pattern

- `sdffre` – enabled flip-flop with synchronous reset
  - `sdffse` – enabled flip-flop with synchronous set
  - `sdfppate` – enabled, vectored flip-flop with synchronous set/reset pattern

You can check the name (type) of any primitive by placing the mouse pointer over it in the RTL view: a tooltip displays the name. The following figure shows flip-flops with synchronous sets and resets.



The event-control syntax for synchronous flip-flops is:

**always @ (edgeKeyword *clockName* )**

In the syntax line, *edgeKeyword* is posedge for a positive-edge triggered clock or negedge for a negative-edge triggered clock.

# SRL Inference

Sequential elements can be mapped into SRLs using an initialization assignment in the Verilog code. You can infer SRLs with initialization values.

Enable SystemVerilog by including a `-vlog_std sysv` argument with one of the following commands:

- run compile
  - run pre\_instrument

- report rtl\_diagnostics
- report syntax\_check

This is an example of an SRL with no resets. The SRL has four 4-bit wide registers and a 4-bit wide read address. Registers shift when the write enable is 1.

```
module test_srl(clk, enable, dataIn, result, addr);
    input clk, enable;
    input [3:0] dataIn;
    input [3:0] addr;
    output [3:0] result;
    reg [3:0] regBank[3:0]='{4'h0,4'h1,4'h2,4'h3};
    integer i;

    always @ (posedge clk) begin
        if (enable == 1) begin
            for (i=3; i>0; i=i-1) begin
                regBank[i] <= regBank[i-1];
            end
            regBank[0] <= dataIn;
        end
    end

    assign result = regBank[addr];
endmodule
```

## Verilog State Machines

This section describes Verilog state machines: guidelines for using them, defining state values, and dealing with asynchrony. The topics include:

- [State Machine Guidelines](#), on page 421
- [State Values](#), on page 423

### State Machine Guidelines

A finite state machine (FSM) is hardware that advances from state to state at a clock edge.

Synthesis works best with synchronous state machines. You typically write a fully synchronous design and avoid asynchronous paths such as paths through the asynchronous reset of a register.

- The state machine must have a synchronous or asynchronous reset, to be inferred. State machines must have an asynchronous or synchronous reset to set the hardware to a valid state after power-up, and to reset your hardware during operation (asynchronous resets are available freely in most FPGA architectures).
- You can define state machines using multiple event controls in an always block only if the event control expressions are identical (for example, `@(posedge clk)`). These state machines are known as implicit state machines. However it is better to use the explicit style.
- Separate the sequential from the combinational always block statements. Besides making it easier to read, it makes what is being registered very obvious. It also gives better control over the type of register element used.
- Represent states with defined labels or enumerated types.
- Use a case statement in an always block to check the current state at the clock edge, advance to the next state, then set the output values. You can use if statements in an always block, but stay with case statements, for consistency.
- Always use a default assignment as the last assignment in your case statement and set the state variable to 'bx. Assigning 'bx to the state variable (a “don't care” for synthesis) tells the tool that you have specified all the used states in your case statement. Any remaining states are not used, and the synthesis tool can remove unnecessary decoding and gates associated with the unused states. You do not have to add any special, non-Verilog directives.

If you set the state to a used state for the default case (for example, `default state = state1`), the tool generates the same logic as if you assign 'bx, but there will be pre- and post-synthesis simulation mismatches until you reset the state machine. These mismatches occur because all inputs are unknown at start up on the simulator. You therefore go immediately into the default case, which sets the state variable to state1. When you power up the hardware, it can be in a used state, such as state2, and then advance to a state other than state1. Post-synthesis simulation behaves more like hardware with respect to initialization.

- Set encoding style with the `syn_encoding` directive. This attribute overrides the default encoding assigned during compilation. When you specify a particular encoding style with `syn_encoding`, that value is used during the mapping stage to determine encoding style.

One-hot implementations are not always the best choice for state machines, even in FPGAs. For example, one-hot state machines might result in larger implementations that can cause fitting problems. An example in an FPGA where one-hot implementation can be detrimental is a state machine that drives a large decoder, generating many output signals. In a 16-state state machine, for instance, the output decoder logic can reference up to sixteen signals in a one-hot implementation, but only four signals in a sequential representation.

## State Values

In Verilog, you must give explicit state values for states. You do this using parameter or `define statements. It is recommended that you use parameter for the following reasons:

- The `define is applied globally whereas parameter definitions are local. With global `define definitions, you cannot reuse common state names that you might want to use in multiple designs, like RESET, IDLE, READY, READ, WRITE, ERROR and DONE. Local definitions make it easier to reuse certain state names in multiple FSM designs. If you work around this restriction by using `undef and then redefining them with `define in the new FSM modules, it makes it difficult to probe the internal values of FSM state buses from a testbench and compare them to state names.
- The tool only displays state names in the FSM Viewer if they are defined using parameter.

## Instantiating Black Boxes in Verilog

Black boxes are modules with just the interface specified; internal information is ignored by the software. Black boxes can be used to directly instantiate:

- Vendor primitives and macros (including I/Os).
- User-designed macros whose functionality was defined in a schematic editor, or another input source. (When the place-and-route tool can merge design netlists from different sources.)

Black boxes are specified with the `syn_black_box` directive. If the macro is an I/O, use `black_box_pad_pin=1` on the external pad pin. The input, output, and delay through a black box are specified with special black box timing directives (see [syn\\_black\\_box, on page 526](#)).

Macro libraries are provided (in *installDirectory/lib/xilinx/unisim\_vivado.v*) that predefine the black boxes for their primitives and macros (including I/Os).

Verilog simulators require a functional description of the internals of a black box. To ensure that the functional description is ignored and treated as a black box, use the `translate_off` and `translate_on` directives. See [translate\\_off/translate\\_on, on page 869](#) for information.

If the black box has tristate outputs, you must define these outputs with a `black_box_tri_pins` directive (see [black\\_box\\_tri\\_pins, on page 479](#)).

## Hierarchical or Structural Verilog Designs

This section describes the creation and use of hierarchical Verilog designs:

- [Using Hierarchical Verilog Designs, on page 424](#)
- [Creating a Hierarchical Verilog Design, on page 424](#)
- [Include Files, on page 425](#)
- [synthesis Macro, on page 425](#)
- [text Macro, on page 426](#)

### Using Hierarchical Verilog Designs

The software accepts and processes hierarchical Verilog designs. You create hierarchy by instantiating a module or a built-in gate primitive within another module.

The signals connect across the hierarchical boundaries through the port list, and can either be listed by position (the same order that you declare them in the lower-level module), or by name (where you specify the name of the lower-level signals to connect to).

Connecting by name minimizes errors, and can be especially advantageous when the instantiated module has many ports.

### Creating a Hierarchical Verilog Design

To create a hierarchical design:

1. Create modules.

2. Instantiate the modules within other modules. (When you instantiate modules inside of other modules, the ones that you have instantiated are sometimes called “lower-level modules” to distinguish them from the “top-level” module that is not inside another module.)
3. Connect signals in the port list together across the hierarchy either “by position” or “by name”.

## Include Files

The `include compiler directive can be used to insert the entire contents of a source file within another source file during compilation. The result appears as though the contents of the included source file replaces the `include compiler directive.

The included file is compiled with the same options (i.e., Verilog standard or defines) as the file in which it is included. Suppose that the file a.h has option set to vlog\_std v95 and is included within top.v, where vlog\_std sysv has been added from the Tcl command line. For this example, the compiler uses the sysv option since the command line has higher precedence than option set.

## synthesis Macro

Use this text macro along with the Verilog `ifdef compiler directive to conditionally exclude part of your Verilog code from being synthesized. The most common use of the synthesis macro is to avoid synthesizing stimulus that only has meaning for logic simulation.

The synthesis (or SYNTHESIS; either all upper case or all lower case is accepted) macro is defined so that the statement `ifdef synthesis is true. The statements in the `ifdef branch are compiled; the stimulus statements in the `else branch are ignored.

---

**Note:** Because Verilog simulators do *not* recognize a synthesis macro, the compiler uses the stimulus in the `else branch.

---

In the following example, an AND gate is used for synthesis because the tool recognizes the synthesis macro to be defined (as true); the assign c = a & b branch is taken. During simulation, an OR gate is used instead, because the simulator does not recognize the synthesis macro to be defined; the assign c = a | b branch is taken.

---

**Note:** A macro in Verilog has a non-zero value only if it is defined.

---

```
module top (a,b,c);
    input a,b;
    output c;
`ifdef synthesis
    assign c = a & b;
`else
    assign c = a | b;
`endif
endmodule
```

## text Macro

The directive `define` creates a macro for text substitution. The compiler substitutes the text of the macro for the string *macroName*. A text macro is defined using arguments that can be customized for each individual use.

The syntax for a text macro definition is as follows.

*textMacroDefinition* ::= **define** *textMacroName* *macroText*

*textMacroName* ::= *textMacroIdentifier*[*(formalArgumentList)*]

*formalArgumentList* ::= *formalArgumentIdentifier* {, *formalArgumentIdentifier*}

When formal arguments are used to define a text macro, the scope of the formal argument is extended to the end of the macro text. You can use a formal argument in the same manner as an identifier.

A text macro with one or more arguments is expanded by replacing each formal argument with the actual argument expression.

## Example 1

```
`define MIN(p1, p2) (p1)<(p2) ? (p1) : (p2)

module example1(i1, i2, o);
    input i1, i2;
    output o;
    reg o;
```

```
always @(i1, i2) begin
  o = `MIN(i1, i2);
end
endmodule
```

## Example 2

```
`define SQR_OF_MAX(a1, a2) (^MAX(a1, a2))*(^MAX(a1, a2))
`define MAX(p1, p2) (p1)<(p2)?(p1):(p2)

module example2(i1, i2, o);
  input i1, i2;
  output o;
  reg o;

  always @(i1, i2) begin
    o = `SQR_OF_MAX(i1, i2);
  end
endmodule
```

## Verilog Attribute and Directive Syntax

Verilog attributes and directives allow you to associate information with your design to control the way it is analyzed, compiled, and mapped.

- *Attributes* direct the way your design is optimized and mapped during synthesis.
- *Directives* control the way your design is analyzed prior to mapping. They must therefore be included directly in your source code; they cannot be specified in a constraint file like attributes.

For information on individual attributes and directives, see the *Attributes and Directives Reference*.

# SystemVerilog Language Support

SystemVerilog is a IEEE (P1800) standard with extensions to the IEEE Std.1364-2001 Verilog standard. The extensions integrate features from C, C++, VHDL, OVA, and PSL. The following table summarizes the SystemVerilog features currently supported. See [SystemVerilog Limitations, on page 431](#) for a list of limitations.

| Feature                                                                                                                                                                                                                                                | Brief Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Unsized Literals                                                                                                                                                                                                                                       | Specification of unsized literals as single-bit values without a base specifier.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Data Types <ul style="list-style-type: none"> <li>• TypeDefs (with unique names)</li> <li>• Enumerated Types</li> <li>• Struct Construct</li> <li>• Union Construct (currently, only packed unions are supported)</li> <li>• Static Casting</li> </ul> | Data types that are a hybrid of Verilog and C, including the following types: <ul style="list-style-type: none"> <li>• User-defined types that allow you to create new type definitions from existing types</li> <li>• Variables and nets defined with a specific set of named values</li> <li>• Structure data type to represent collections of variables referenced as a single name</li> <li>• Data type collections sharing the same memory location</li> <li>• Conversion of one data type to another data type.</li> <li>• Iteration over enumerated type values; methods include first, last, next, previous, name, and num.</li> </ul> |
| Arrays <ul style="list-style-type: none"> <li>• Arrays</li> <li>• Arrays of Structures</li> </ul>                                                                                                                                                      | Packed, unpacked, and multi-dimensional arrays of structures.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Data Declarations <ul style="list-style-type: none"> <li>• Constants</li> <li>• Variables</li> <li>• Nets</li> <li>• Data Types in Parameters</li> <li>• Type Parameters</li> </ul>                                                                    | Data declarations including constant, variable, net, and parameter data types.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

| Feature                                                                                                                                                                                                                                                                                             | Brief Description                                                                                                                                                                                         |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Operators and Expressions <ul style="list-style-type: none"> <li>• Operators</li> <li>• Aggregate Expressions</li> <li>• Streaming Operator</li> <li>• Set Membership Operator</li> <li>• Set Membership Case Inside Operator</li> <li>• type/\$typeof Operator</li> </ul>                          | C assignment operators and special bit-wise assignment operators.                                                                                                                                         |
| Procedural Statements and Control Flow <ul style="list-style-type: none"> <li>• Do-While Loops</li> <li>• For Loops</li> <li>• Foreach Loops</li> <li>• Unnamed Blocks</li> <li>• Block Name on end Keyword</li> <li>• Unique and Priority Modifiers</li> <li>• bind and force Functions</li> </ul> | Procedural statements including variable declarations and block functions.                                                                                                                                |
| Processes <ul style="list-style-type: none"> <li>• always_comb</li> <li>• always_latch</li> <li>• always_ff</li> </ul>                                                                                                                                                                              | Specialized procedural blocks that reduce ambiguity and indicate the intent.                                                                                                                              |
| Tasks and Functions <ul style="list-style-type: none"> <li>• Implicit Statement Group</li> <li>• Formal Arguments</li> <li>• endtask/endfunction Names</li> </ul>                                                                                                                                   | Information on implicit grouping for multiple statements, passing formal arguments, and naming end statements for functions and tasks.                                                                    |
| Hierarchy <ul style="list-style-type: none"> <li>• Compilation Units</li> <li>• Packages</li> <li>• Port Connection Constructs</li> <li>• Extern Module</li> </ul>                                                                                                                                  | Permits sharing of language-defined data types, user-defined types, parameters, constants, function definitions, and task definitions among one or more compilation units, modules, or interfaces (pkgs). |
| Interface <ul style="list-style-type: none"> <li>• Interface Construct</li> <li>• Modports</li> </ul>                                                                                                                                                                                               | Interface data type to represent port lists and port connection lists as single name.                                                                                                                     |
| System Tasks and System Functions <ul style="list-style-type: none"> <li>• \$bits System Function</li> <li>• Array Querying Functions</li> </ul>                                                                                                                                                    | Queries to returns number of bits required to hold an expression as a bit stream or array.                                                                                                                |

| Feature                                                                                                                                        | Brief Description                                                                                             |
|------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| Generate Statements <ul style="list-style-type: none"><li>• Multiple Configuration Support</li><li>• Conditional Generate Constructs</li></ul> | Creates multiple instances of an object in a module (see <a href="#">Generate Statements , on page 435</a> ). |
| Parameters                                                                                                                                     | Parameters to define constants                                                                                |
| Assertions                                                                                                                                     | <a href="#">SystemVerilog Assertions</a>                                                                      |
| Keyword Support                                                                                                                                | <a href="#">SystemVerilog Keyword Support</a>                                                                 |

## SystemVerilog Limitations

The following SystemVerilog limitations are present in the current release.

### Interface/modport Structures

The following restrictions apply when using interface/modport structures:

- Access of array type elements outside of the interface is not supported.
- Modport definitions within a Generate block are not supported.
- Declaring an interface within another interface is not supported
- Direction information in modports has no effect on synthesis.Exported (export keyword) interface functions and tasks are not supported.
- Virtual interfaces are not supported.
- Full hierarchical naming of interface members is not supported.
- An extern module declaration is not supported within a module.Compilation Unit and Package

Write access to the variable defined in package/compilation unit is not supported.

### Union Constructs

The following union constructs are not supported:

- unpacked union
- tagged packed union
- tagged unpacked union

Currently, support is limited to packed unions, arrays of packed unions, and nested packed unions.

## Nets Restrictions

Using wire with a 2-state data type (for example, int or bit) results in the following error message:

```
CG1205 | Net data types must be 4-state values
```

A lexical restriction also applies to a net or port declaration in that the net type keyword `wire` cannot be followed by `reg`.

## **type/\$typeof Operator**

An *expression* inside a type or \$typeof operator results in a self-determined type of expression; the expression is not evaluated. Also the *expression* cannot contain any hierarchical references. The type or \$typeof operator is not supported on complex expressions (for example type(d1\*d2)).

## **always\_comb**

The `always_comb` process block models combinational logic, and the logic inferred from the `always_comb` process must be combinational logic. The compiler warns you if the behavior does not represent combinational logic.

The semantics of an `always_comb` block are different from a normal `always` block in these ways:

- It is illegal to declare a sensitivity list in tandem with an `always_comb` block.
- An `always_comb` statement cannot contain any block, timing, or event controls and fork, join, or wait statements.

Note the following about the `always_comb` block:

- There is an inferred sensitivity list that includes all the variables from the RHS of all assignments within the `always_comb` block and variables used to control or select assignments.
- The variables on the LHS of the expression cannot be written by any other processes.
- `always_comb` is sensitive to changes within the contents of a function and not just the function arguments, unlike the `always@(*)` construct of Verilog 2001.

## **always\_latch**

The SystemVerilog `always_latch` process models latched logic, and the logic inferred from the `always_latch` process must only be latches (of any kind). The compiler warns you if the behavior does not follow the intent.

It is illegal for `always_latch` statements to contain a sensitivity list, any block, timing, or event controls, and fork, join, or wait statements. The sensitivity list of an `always_latch` process is automatically inferred by the compiler and the inferring rules are similar to the `always_comb` process (see [always\\_comb, on page 432](#)).

## **always\_ff**

The SystemVerilog `always_ff` process block models sequential logic that is triggered by clocks. The compiler warns you if the behavior does not represent the intent. The `always_ff` process has the following restrictions:

- An `always_ff` block must contain only one event control and no blocking timing controls.
- Variables on the left side of assignments within an `always_ff` block must not be written to by any other process.

## **Package Variables**

The variables declared in packages can only be accessed or read; package variables cannot be written between a module statement and its end module statement.

## **.name Connection**

Restrictions to the `.name` feature are the same as the restrictions for named associations in Verilog. In addition, the following restrictions apply:

- Currently, the `.name` port connection is not supported for mixed HDL source code.
- Named associations and positional associations cannot be mixed.
- Sizes must match in mixed named and positional associations. The example below is not valid because of the size mismatch on `in2`.

```
myand mand3(.in1, .out, .in2);
```

- The identifier referred by `.name` must not create an implicit declaration, regardless of the compiler directive `'default_nettype`.
- You cannot use the `.name` connection to create an implicit cast.

## .\* Connection

Restrictions to the `.*` feature are the same as the restrictions for the `.name` feature. See [.name Connection, on page 433](#). In addition, the following restrictions apply:

- Currently, the `.*` port connection is not supported for mixed HDL source code.
- Named associations and positional associations cannot be mixed.
- Named associations where there is a mismatch of variable sizes or names generate an error.
- You can only use the `.*` once per instantiation, although you can mix the `.*` connection with `.name` and `.port_identifier(name)` type connections.
- If you use a `.*` construction, but all remaining ports are explicitly connected, the compiler ignores the `.*` construct.

## Nested Interface

With the nested interface feature, nesting of interface is possible by either instantiating one interface in another or by using one interface as a port in another interface. Generic interface is not supported for nested interface; array of interface when using interface as a port also is not supported.

## \$bits System Function

The `$bits` system function is not supported under the following conditions:

- Passing an interface member as an argument to the `$bits` function. In the example

```
parameter logic[2:0] din = $bits(ff_if_0.din);
```

interface instance `ff_if_0.din` is one of the ports of the modport. To avoid the limitation, use the actual value as the argument in place of the interface member.

- `$bits` cannot be used within a module instantiation.

- \$bits is not supported with params/localparams.

## Generate Statements

The generate statement is a Verilog 2005 feature; to use this statement, you must enable SystemVerilog. The statement creates multiple instances of an object in a module. You can use generate statements with loop, conditional, and case statements with parameters and function/task declarations; a configuration rule can be used to specify how the instance is to be configured.

Single and multiple configurations on generated instances are supported within generate statements. Multiple-configuration support allows submodules to be compiled to a library other than the instantiating top-module library with each generated instance configured through a separate configuration file.

The following generate statement functions are not currently supported:

- Defparam support for generate instances
- Hierarchical access for interface
- Hierarchical access of function/task defined within a generate block

---

**Note:** Whenever the generate statement contains symbolic hierarchies separated by a hierarchy separator (.), the instance name includes the (\) character before this hierarchy separator (.).

---

## Conditional Generate Constructs

The if-generate and case-generate conditional generate constructs allow the selection of, at most, one generate block from a set of alternative generate blocks based on constant expressions evaluated during elaboration. The generate and endgenerate keywords are optional.

Generate blocks in conditional generate constructs can be either named or unnamed and can consist of only a single item. It is not necessary to enclose the blocks with begin and end keywords; the block is still a generate block and, like all generate blocks, comprises a separate scope and a new level of hierarchy when it is instantiated. The if-generate and case-generate constructs can be combined to form a complex generate scheme.

## Bind and Force Statements

The bind statement is a non-invasive function that allows you to insert an instance in the design hierarchy; the force statement is a non-invasive function that allows you to override existing drivers of internal signals in the design hierarchy with new drivers. To enable the use of either of these functions, set the bindandforce option to 1.

## SystemVerilog Assertions

The parsing of SystemVerilog Assertions (SVA) is supported as outlined in the following table.

| Assertion Construct                   | Support Level                | Comment                                                                                                   |
|---------------------------------------|------------------------------|-----------------------------------------------------------------------------------------------------------|
| Immediate assertions                  | Supported                    |                                                                                                           |
| Concurrent assertions                 | Partially Supported, Ignored | Multi-clock properties are not supported                                                                  |
| Boolean expressions                   | Partially Supported, Ignored | In the boolean expressions, \$rose function having a clocking event is not supported.                     |
| Sequence                              | Supported, ignored           |                                                                                                           |
| Declaring sequences                   | Partially Supported, Ignored | Sequence with ports declared in global space is not supported                                             |
| Sequence operations                   | Partially Supported, Ignored | All variations of first_match, within and intersect in a sequence is not supported.                       |
| Manipulating data in a sequence       | Partially Supported, Ignored | More than one assignment in the parenthesis is not supported.                                             |
| Calling subroutines on sequence match | Partially Supported, Ignored | Calling of more than one tasks is not supported                                                           |
| System functions                      | Partially Supported          | System functions \$onehot, \$onehot1, \$countones, and \$countbits supported; \$isunknown not supported   |
| Declaring properties                  | Partially Supported, Ignored | Declaring of properties in a package and properties with ports declared in global space are not supported |

| Assertion Construct                      | Support Level                | Comment |
|------------------------------------------|------------------------------|---------|
| Multi-clock support                      | Partially Supported, Ignored |         |
| Expect statement                         | Not Supported                |         |
| Final blocks                             | Partially Supported, Ignored |         |
| Property blocks                          | Supported, Ignored           |         |
| Checker                                  | Partially Supported, Ignored |         |
| Default clocking and default disable iff | Supported, Ignored           |         |
| Let statement                            | Partially Supported, Ignored |         |
| Program                                  | Partially Supported, Ignored |         |

## SystemVerilog Keyword Support

This table lists supported SystemVerilog keywords:

|              |              |              |              |
|--------------|--------------|--------------|--------------|
| always_comb  | always_ff    | always_latch | assert*      |
| assume*      | automatic    | bind*        | bit          |
| break        | byte         | checker*     | clocking*    |
| const        | continue     | cover*       | do           |
| endchecker*  | endclocking* | endinterface | endproperty* |
| endsequence* | enum         | expect*      | extern       |
| final*       | function     | global*      | import       |
| inside       | int          | interface    | intersect*   |
| let*         | logic        | longint      | modport      |
| packed       | package      | parameter    | priority     |

\* Reserved keywords for SystemVerilog assertion parsing; cannot be used as identifiers or object names

|                |           |         |             |
|----------------|-----------|---------|-------------|
| property*      | restrict* | return  | sequence*   |
| shortint       | struct    | task    | throughout* |
| timeprecision* | timeunit* | typedef | union       |
| unique         | void      | within* |             |

\* Reserved keywords for SystemVerilog assertion parsing; cannot be used as identifiers or object names

---

# VHDL Language Support

This section describes how the synthesis tool relates to different VHDL language constructs. The topics include:

- [Supported VHDL Language Constructs](#), on page 439
- [Unsupported VHDL Language Constructs](#), on page 440
- [Partially-supported VHDL Language Constructs](#), on page 441
- [Ignored VHDL Language Constructs](#), on page 441

## Supported VHDL Language Constructs

The following is a compact list of language constructs that are supported.

- Entity, architecture, and package design units
- Function and procedure subprograms
- All IEEE library packages, including:
  - std\_logic\_1164
  - std\_logic\_unsigned
  - std\_logic\_signed
  - std\_logic\_arith
  - numeric\_std and numeric\_bit
  - standard library package (std)
- In, out, inout, buffer, linkage ports
- Signals, constants, and variables
- Aliases
- Integer, physical, and enumeration data types; subtypes of these
- Arrays of scalars and records
- Record data types
- File types
- All operators (-, -, \*, /, \*\*, mod, rem, abs, not, =, /=, <, <=, >, >=, and, or, nand, nor, xor, xnor, sll, srl, sla, sra, rol, ror, &)

**Note:** With the `**` operator, arguments are compiler constants. When the left operand is 2, the right operand can be a variable.

---

- Sequential statements: signal and variable assignment, wait, if, case, loop, for, while, return, null, function, and procedure call
- Concurrent statements: signal assignment, process, block, generate (for and if), component instantiation, function, and procedure call
- Component declarations and four methods of component instantiations
- Configuration specification and declaration
- Generics; attributes; overloading
- Next and exit looping control constructs
- Predefined attributes: `t'base`, `t'left`, `t'right`, `t'high`, `t'low`, `t'succ`, `t'pred`, `t'val`, `t'pos`, `t'leftof`, `t'rightof`, `integer'image`, `a'left`, `a'right`, `a'high`, `a'low`, `a'range`, `a'reverse_range`, `a'length`, `a'ascending`, `s'stable`, `s'event`
- Unconstrained ports in entities
- Global signals declared in packages
- Dynamic ranges on for LHS, RHS, or both sides of a slice in an expression; available for concurrent, procedural, and function/procedure assignments, comparison statements (with if only), and concatenations (see [Language Construct Restrictions, on page 450](#)).

## Unsupported VHDL Language Constructs

If any of these constructs are found, an error message is reported and the synthesis run is cancelled.

- Register and bus kind signals
- Guarded blocks
- Expanded (hierarchical) names
- User-defined resolution functions. The synthesis tool only supports the resolution functions for `std_logic` and `std_logic_vector`.
- Slices with range indices that do not evaluate to constants

## Partially-supported VHDL Language Constructs

When one of the following constructs is encountered, compilation continues, but will subsequently error out if logic must be generated for the construct.

- real data types (real data expressions are supported in VHDL-2008 IEEE float\_pkg.vhd) – real data types are supported as constant declarations or as constants used in expressions as long as no floating point logic must be generated
- access types – limited support for file I/O
- null arrays – null arrays are allowed as operands in concatenation expressions

## Ignored VHDL Language Constructs

The synthesis tool ignores the following constructs in your design. If found, the tool parses and ignores the construct (provided that no logic is required to be synthesized) and continues with the synthesis run.

- disconnect
- report
- initial values on inout ports
- assert on ports and signals
- after

# VHDL Language Constructs

This section describes the synthesis language support that the synthesis tool provides for each VHDL construct. The language information is taken from the most recent VHDL Language Reference Manual (Revision ANSI/IEEE Std 1076-1993). The section names match those from the LRM, for easy reference.

## Libraries and Packages

When you want to synthesize a design in VHDL, include the HDL files in the source files list of your synthesis tool project. Often your VHDL design will have more than one source file. List all the source files in the order you want them compiled; the files at the top of the list are compiled first.

### Compiling Design Units into Libraries

All design units in VHDL, including your entities and packages are compiled into libraries. A library is a special directory of entities, architectures and/or packages. You compile source files into libraries by adding them to the source file list. In VHDL, the library you are compiling has the default name work. All entities and packages in your source files are automatically compiled into work. You can keep source files anywhere on your disk, even though you add them to libraries. You can have as many libraries as are needed.

1. To add files to a library, specify the file or files in a ProtoCompiler file list (PFL file)
2. Use a run compile or run pre\_instrument command with an -src/-srclist argument to identify the files to be added and include a -lib argument to specify the library for the files.

If you want to use a design unit that you compiled into a library (one that is no longer in the default work library), you must use a library clause in the VHDL source code to reference the library.

For example, if you add a source file for the entity ram16x8 to library my\_rams, and instantiate the 16x8 RAM in the design called top\_level, you must add library my\_rams; immediately before defining top\_level.

## Predefined Packages

The synthesis tool supports the two standard libraries, std and ieee, that contain packages containing commonly used definitions of data types, functions, and procedures. These libraries and their packages are built in to the synthesis tool, so you do not compile them. The predefined packages are described in the following table.

| Library | Package            | Description                                                                                        |
|---------|--------------------|----------------------------------------------------------------------------------------------------|
| std     | standard           | Defines the basic VHDL types including bit and bit_vector                                          |
| ieee    | std_logic_1164     | Defines the 9-value std_logic and std_logic_vector types                                           |
| ieee    | numeric_bit        | Defines numeric types and arithmetic functions. The base type is BIT.                              |
| ieee    | numeric_std        | Defines arithmetic operations on types defined in std_logic_1164                                   |
| ieee    | std_logic_arith    | Defines the signed and unsigned types, and arithmetic operations for the signed and unsigned types |
| ieee    | std_logic_signed   | Defines signed arithmetic for std_logic and std_logic_vector                                       |
| ieee    | std_logic_unsigned | Defines unsigned arithmetic for std_logic and std_logic_vector                                     |

The synthesis tools also have built-in macro libraries for components like gates, counters, flip-flops, and I/Os. The libraries are located in *installDirectory/lib/xilinx*. Use the built-in macro libraries to instantiate these macros directly into the VHDL designs and forward-annotate them to the output netlist.

Additionally, the synthesis tool library contains an attributes package of built-in attributes and timing constraints (*installDirectory/lib/vhd/synattr.vhd*) that you can use with VHDL designs. The package includes declarations for timing constraints (including black-box timing constraints), vendor-specific attributes and synthesis attributes. To access these built-in attributes, add the following two lines to the beginning of each of the VHDL design units that uses them:

```
library synplify;
use synplify.attributes.all;
```

If you want the addition operator (+) to take two std\_ulogic or std\_ulogic\_vector as inputs, you need the function defined in the std\_logic\_arith package (the cdn\_arith.vhd file in *installDirectory/lib/vhd/*). Add this file to the project. To successfully compile, the VHDL file that uses the addition operator on these input types must have include the following statement:

```
use work.std_logic_arith.all;
```

## Accessing Packages

To gain access to a package include a library clause in your VHDL source code to specify the library the package is contained in, and a use clause to specify the name of the package. The library and use clauses must be included immediately before the design unit (entity or architecture) that uses the package definitions.

### Syntax

```
library library_name;
use library_name.package_name.all;
```

To access the data types, functions and procedures declared in std\_logic\_1164, std\_logic\_arith, std\_logic\_signed, or std\_logic\_unsigned, you need a library ieee clause and a use clause for each of the packages you want to use.

### Example

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

-- Other code
```

## Library and Package Rules

To access the standard package, no library or use clause is required. The standard package is predefined (built-in) in VHDL, and contains the basic data types of bit, bit\_vector, Boolean, integer, real, character, string, and others along with the operators and functions that work on them.

If you create your own package and compile it into the work library to access its definitions, you still need a use clause before the entity using them, but not a library clause (because work is the default library.)

To access packages other than those in work and std, you must provide a library and use clause for each package as shown in the following example of creating a resource library.

```
-- Compile this in library mylib
library ieee;
use ieee.std_logic_1164.all;

package my_constants is
constant max: std_logic_vector(3 downto 0):="1111";
.
.
.
end package;

-- Compile this in library work
library ieee, mylib;
use ieee.std_logic_1164.all;
use mylib.my_constants.all;

entity compare is
    port (a: in std_logic_vector(3 downto 0);
          z: out std_logic );
end compare;

architecture rtl of compare is
begin
    z <= '1' when (a = max) else '0';
end rtl;
```

The rising\_edge and falling\_edge functions are defined in the std\_logic\_1164 package. If you use these functions, your clock signal must be declared as type std\_logic.

## Instantiating Components in a Design

No library or use clause is required to instantiate components (entities and their associated architectures) compiled in the default work library. The files containing the components must be listed in the source files list before any files that instantiate them.

To instantiate components from the built-in technology-vendor macro libraries, you must include the appropriate use and library clauses in your source code. Refer to the section for your vendor for more information.

To create a separate resource library to hold your components, put all the entities and architectures in one source file, and assign that source file the library components in the synthesis tool Project view. To access the components from your source code, put the clause library components; before the designs that instantiate them. There is no need for a use clause. The database must include both the files that create the package components and the source files that access them.

## Package Definitions

A package is a unit that groups various declarations to be shared among several designs. Packages are stored in libraries for greater convenience. A package consists of package declaration as shown in the following syntax:

```
package packageName is  
    package_declarations  
end package packageName;
```

The purpose of a package is to declare shareable types, subtypes, constants, signals, files, aliases, component attributes, and groups. Once a package is defined, it can be used in multiple independent designs. Items declared in a package declaration are visible in other design units if the use clause is applied.

## VHDL Implicit Data-type Defaults

Type default propagation avoids potential simulation mismatches that are the result of differences in behavior with how initial values for registers are treated in the synthesis tools and how they are treated in the simulation tools.

With implicit data-type defaults, when there is no explicit initial-value declaration for a signal being registered, the VHDL compiler passes an init value through a syn\_init property to the mapper, and the mapper then propagates the value to the respective register. Compiler requirements are based on specific data types. These requirements can be broadly grouped based on the different data types available in the VHDL language.

Implicit data-type defaults are enabled through the `-supporttypedfit` argument to the option set command.

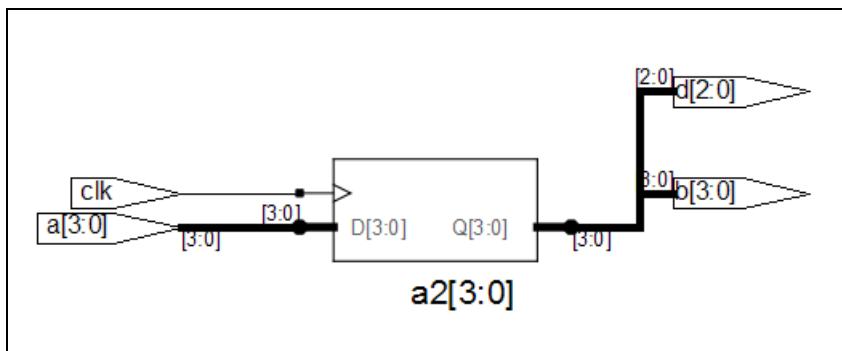
To illustrate the use of implicit data-type defaults, consider the following example.

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

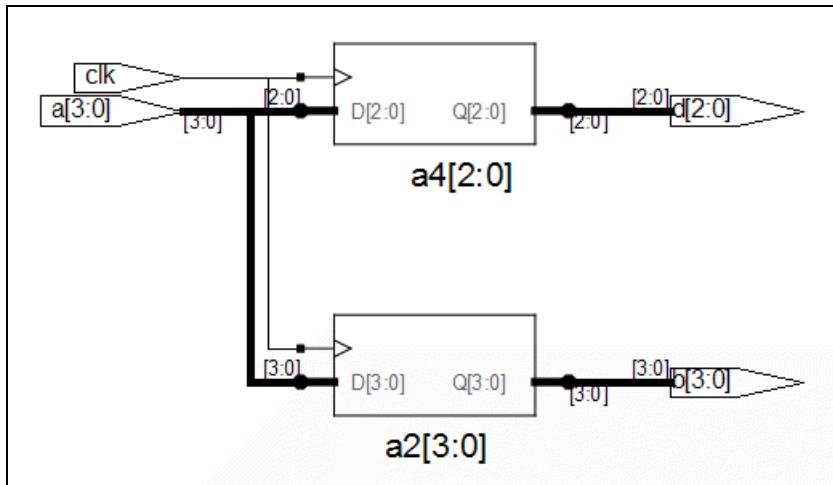
entity top is
    port (clk:in std_logic;
          a : in integer range 1 to 8;
          b : out integer range 1 to 8;
          d : out positive range 1 to 7);
end entity top;

architecture rtl of top is
signal a1,a2 : integer range 1 to 8;
signal a3,a4 : positive range 1 to 7;
begin
a1 <= a;
a3 <= a;
b <= a2;
d <= a4;
process(clk)
begin
    if (rising_edge(clk))then
        a2 <= a1;
        a4 <= a3;
    end if;
end process;
end rtl;
```

In the above example, two signals (a2 and a4) with different type default values are registered. Without implicit data-type defaults, if the values of the signals being registered are not the same, the compiler merges the redundant logic into a single register as shown in the figure below.



Enabling implicit data-type defaults prevents certain compiler and mapper optimizations to preserve both registers as shown in the following figure.



### Example – Impact on Integer Ranges

The default value for the integer type when a range is specified is the minimum value of the range specified, and size is the upper limit of that range. With implicit data-type defaults, the compiler is required to propagate the minimum value of the range as the init value to the mapper. Consider the following example:

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

```

entity top is
    port (clk,set:in std_logic;
          a : in integer range -6 to 8;
          b : out integer range -6 to 8);
end entity top;

architecture rtl of top is
signal a1,a2: integer range -6 to 8;
begin
a1 <= a ;
    process(clk,set)
begin
    if (rising_edge(clk))then
        if set = '1' then
            a2 <= a;
        else
            a2 <= a1;
        end if;
    end if;
    end process;
b <= a2;
end rtl;

```

In the example,

```
signal a1, a2: integer range -6 to 8;
```

the default value is -6 (FA in 2's complement) and the range is -6 to 8. With a total of 15 values, the size of the range can be represented in four bits.

### Example – Impact on RAM Inferencing

When inferencing a RAM with implicit data-type defaults, the compiler propagates the type default values as init values for each RAM location. The mapper must check if the block RAMs of the selected technology support initial values and then determine if the compiler-propagated init values are to be considered. If the mapper chooses to ignore the init values, a warning is issued stating that the init values are being ignored. Consider the following VHDL design:

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

```

```

entity top is
    port (clk : in std_logic;
          addr : in std_logic_vector (6 downto 0);
          din : in positive;
          wen : in std_logic;
          dout : out positive);
end top;

architecture behavioral of top is
-- RAM
type trram is array(0 to 127) of positive;
signal ram : trram ;
begin
-- Contents of RAM has initial value = 1
process (clk)
begin
    if clk'event and clk = '1' then
        if wen = '1' then
            ram(conv_integer(addr)) <= din;
        end if;
        dout <= ram(conv_integer(addr));
    end if;
end process;
end behavioral;

```

In the above example:

- The type of signal a1 is bit\_vector
- The default value for type integer is 1 when no range is specified

Accordingly, a value of x00000001 is propagated by the compiler to the mapper with a syn\_init property.

## Language Construct Restrictions

Currently, the following limitations apply to dynamic range assignments:

- There is no support for selected signal assignment; i.e., with *expression* Select.
- Only comparisons with if statements are supported; no other comparison statements can be used.

# VHDL 2008 Language Support

This section describes support for the VHDL 2008 standard. For information on the VHDL standard, see [VHDL Language Support, on page 439](#) and the IEEE 1076™-2008 standard. The following sections describe the current level of VHDL 2008 support.

- [Operators](#), on page 451
- [Unconstrained Data Types](#), on page 453
- [Unconstrained Record Elements](#), on page 453
- [Predefined Functions](#), on page 453
- [Generics](#), on page 454
- [Packages](#), on page 454
- [Generics in Packages](#), on page 455
- [Context Declarations](#), on page 456
- [Case-generate Statements](#), on page 456
- [Matching case and select Statements](#), on page 458
- [else/elsif Clauses](#), on page 458
- [Syntax Conventions](#), on page 459

## Operators

VHDL 2008 includes support for the following operators:

- Logical Reduction operators – the logic operators: and, or, nand, nor, xor, and xnor can now be used as unary operators
- Condition operator (??) – converts a bit or std\_ulogic value to a boolean value
- Matching Relational operators (?=, ?/=, ?<, ?<=, ?>, ?>=) – similar to the normal relational operators, but return bit or std\_ulogic values in place of Boolean values
- Bit-string literals – bit-string characters other than 0 and 1 and string formats including signed/unsigned and string length

## Logical Reduction Operators

The logical operators and, or, nand, nor, xor, and xnor can be used as unary operators.

## Condition Operator

The condition operator (??) converts a bit or std\_ulogic value to a boolean value. The operator is implicitly applied in a condition where the expression would normally be interpreted as a boolean value as shown in the if statement in the two examples below.

The implicit use of the ?? operator occurs in the following conditional expressions:

- after if or elsif in an if statement
- after if in an if-generate statement
- after until in a wait statement
- after while in a while loop
- after when in a conditional signal statement
- after assert in an assertion statement
- after when in a next statement or an exit statement

## Matching Relational Operators

The matching relational operators return a bit or std\_ulogic result in place of a Boolean.

## Bit-string Literals

Bit-string literal support in VHDL 2008 includes:

- Support for characters other than 0 and 1 in the bit string, such as X or Z.
- Optional support for a length specifier that determines the length of the string to be assigned.
- Optional support for a signed (S) or unsigned (U) qualifier that determines how the bit-string value is expanded/truncated when a length specifier is used.

- Additional support for a base specifier for decimal numbers (D). The number of characters in the bit string can be determined by using the expression  $(\log_2 n) + 1$ ; where  $n$  is the decimal integer.

For complete descriptions of bit-string literal requirements, see the VHDL 2008 LRM.

## Unconstrained Data Types

VHDL 2008 allows the element types for arrays and the field types for records to be unconstrained. In addition, VHDL 2008 includes support for partially constrained subtypes in which some elements of the subtype are constrained, while others elements are unconstrained. Specifically, VHDL 2008:

- Supports unconstrained arrays of unconstrained arrays (i.e., element types of arrays can be unconstrained)
- Supports the VHDL 2008 syntax that allows a new subtype to be declared that constrains any element of an existing type that is not yet constrained
- Supports the ‘element attribute that returns the element subtype of an array object
- Supports the new ‘subtype attribute that returns the subtype of an object

## Unconstrained Record Elements

VHDL 2008 allows element types for records to be unconstrained (earlier versions of VHDL required that the element types for records be fully constrained). In addition, VHDL 2008 supports the concept of partially constrained subtypes in which some parts of the subtype are constrained, while others remain unconstrained.

## Predefined Functions

VHDL 2008 adds the minimum and maximum predefined functions. The behavior of these functions is defined in terms of the “<” operator for the operand type. The functions can be binary to compare two elements, or unary when the operand is an array type.

## Generics

VHDL 2008 introduces several types of generics that are not present in VHDL IEEE Std 1076-1993. These types include generic types, generic packages, and generic subprograms.

### Generic Types

Generic types allow logic descriptions that are independent of type. These descriptions can be declared as a generic parameter in both packages and entities. The actual type must be provided when instantiating a component or package.

### Generic Packages

Generic packages allow descriptions based on a formal package. These descriptions can be declared as a generic parameter in both packages and entities. An actual package (an instance of the formal package) must be provided when instantiating a component or package.

### Generic Subprograms

Generic subprograms allow descriptions based on a formal subprogram that provides the function prototype. These descriptions can be declared as a generic parameter in both packages and entities. An actual function must be provided when instantiating a component or package.

## Packages

VHDL 2008 includes several new packages and modifies some of the existing packages. The new and modified packages are located in the \$LIB/vhd2008 folder instead of \$LIB/vhd.

### New Packages

The following packages are supported in VHDL 2008:

- fixed\_pkg.vhd, float\_pkg.vhd, fixed\_generic\_pkg.vhd, float\_generic\_pkg.vhd, fixed\_float\_types.vhd – IEEE fixed and floating point packages
- numeric\_bit\_unsigned.vhd – Overloads for bit\_vector to have all operators defined for ieee.numeric\_bit,unsigned

- numeric\_std\_unsigned.vhd – Overloads for std\_ulogic\_vector to have all operators defined for ieee.numeric\_std.unsigned

String and text I/O functions in the above packages are not to be supported. These functions include read(), write(), to\_string().

## Modified Packages

The following modified IEEE packages are supported with the exception of the new string and text I/O functions (the previously supported string and text I/O functions are unchanged):

- std.vhd – new overloads
- std\_logic\_1164.vhd – std\_logic\_vector is now a subtype of std\_ulogic\_vector; new overloads
- numeric\_std.vhd – new overloads
- numeric\_bit.vhd – new overloads

## Unsupported Packages/Functions

The following packages and functions are not currently supported:

- string and text I/O functions in the new packages
- The fixed\_pkg\_params.vhd or float\_pkg\_params.vhd packages, which were temporarily supported to allow the default parameters to be changed for fixed\_pkg.vhd and float\_pkg.vhd packages, have been obsoleted by the inclusion of the fixed\_generic\_pkg.vhd or float\_generic\_pkg.vhd packages.

## Using the Packages

The following option is used to enable the VHDL 2008 packages and to enable the use of the ?? operator:

```
option set vhdl2008 1
```

## Generics in Packages

In VHDL 2008, packages can include generic clauses. These generic packages can then be instantiated by providing values for the generics.

## Context Declarations

VHDL 2008 provides a new type of design unit called a context declaration. A context is a collection of library and use clauses. Both context declarations and context references are supported. In VHDL 2008, a context clause cannot precede a context declaration. Similarly, VHDL 2008 does not allow reference to the library name work in a context declaration.

VHDL 2008 supports the following two, standard context declarations in the IEEE package:

- IEEE\_BIT\_CONTEXT
- IEEE\_STD\_CONTEXT

## Case-generate Statements

The case-generate statement is a new type of generate statement incorporated into VHDL 2008. Within the statement, alternatives are specified similar to a case statement. A static (computable at elaboration) select statement is compared against a set of choices as shown in the following syntax:

```
caseLabel: case expression generate
    when choice1 =>
        -- statement list
    when choice2 =>
        -- statement list
    ...
end generate caseLabel;
```

To allow for configuration of alternatives in case-generate statements, each alternative can include a label preceding the choice value (e.g., labels L1 and L2 in the syntax below):

```
caseLabel: case expression generate
    when L1: choice1 =>
        -- statement list
    when L2: choice2 =>
        -- statement list
    ...
end generate caseLabel;
```

## Example – Case-generate Statement with Alternatives

```

entity myTopDesign is
    generic (instSel: bit_vector(1 downto 0) := "10");
    port (in1, in2, in3: in bit; out1: out bit);
end myTopDesign;

architecture myarch2 of myTopDesign is
component mycomp
    port (a: in bit; b: out bit);
end component;

begin
a1: case instSel generate
    when "00" =>
        inst1: component mycomp port map (in1,out1);
    when "01" =>
        inst1: component mycomp port map (in2,out1);
    when others =>
        inst1: component mycomp port map (in3,out1);
    end generate;
end myarch2;

```

## Example – Case-generate Statement with Labels for Configuration

```

entity myTopDesign is
generic (selval: bit_vector(1 downto 0) := "10");
    port (in1, in2, in3: in bit; tstIn: in bit_vector(3 downto 0);
          out1: out bit);
end myTopDesign;

architecture myarch2 of myTopDesign is
component mycomp
    port (a: in bit; b: out bit);
end component;

begin
a1: case selval generate
    when spec1: "00" | "11"=> signal inRes: bit;
        begin
            inRes <= in1 and in3;
            inst1: component mycomp port map (inRes,out1);
        end;
    when spec2: "01" =>
        inst1: component mycomp port map (in1, out1);
    when spec3: others =>
        inst1: component mycomp port map (in3,out1);
    end generate;
end myarch2;

```

```
entity mycomp is
    port (a : in bit;
          b : out bit);
end mycomp;

architecture myarch of mycomp is
begin
    b <= not a;
end myarch;

architecture zarch of mycomp is
begin
    b <= '1';
end zarch;

configuration myconfig of myTopDesign is
for myarch2
    for al1 (spec1)
        for inst1: mycomp use entity mycomp(myarch);
            end for;
    end for;
    for al1 (spec2)
        for inst1: mycomp use entity mycomp(zarch);
            end for;
    end for;
    for al1 (spec3)
        for inst1: mycomp use entity mycomp(myarch);
            end for;
    end for;
end for;
end configuration myconfig;
```

## Matching case and select Statements

Matching case and matching select statements are supported – `case?` (matching case statement) and `select?` (matching select statement). The statements use the `?=` operator to compare the case selector against the case options.

## else/elsif Clauses

In VHDL 2008, `else` and `elsif` clauses can be included in `if-generate` statements. You can configure specific `if/else/elsif` clauses using configurations by adding a label before each condition. In the code example below, the labels on the

branches of the if-generate statement are spec1, spec2, and spec3. These labels are later referenced in the configuration myconfig to specify the appropriate entity/architecture pair. This form of labeling allows statements to be referenced in configurations.

## Syntax Conventions

The following syntax conventions are supported in VHDL 2008:

- All keyword
- Comment delimiters
- Extended character set

### All Keyword

VHDL 2008 supports the use of an all keyword in place of the list of input signals to a process in the sensitivity list.

### Comment Delimiters

VHDL 2008 supports the /\* and \*/ comment-delimiter characters. All text enclosed between the beginning /\* and the ending \*/ is treated as a comment, and the commented text can span multiple lines. The standard VHDL “--” comment-introduction character string is also supported.

### Extended Character Set

The extended ASCII character literals (ASCII values from 128 to 255) are supported.



## CHAPTER 8

# Attributes and Directives

---

This chapter describes the attributes and directives available in the ProtoCompiler products. Attributes and directives allow you to direct the way in which a design is analyzed, optimized, and mapped during synthesis.

This chapter includes the following introductory information:

- [Specifying Attributes and Directives](#), on page 462
- [Attributes and Directives Summary](#), on page 473

# Specifying Attributes and Directives

Attributes and directives are specifications that you assign to design objects to control the way your design is analyzed, optimized, and mapped.

Attributes control mapping optimizations, and directives control compiler optimizations. Because of this difference, directives must be entered directly in the HDL source code or through a compiler design constraint (CDC) file. Attributes can be entered either in the source code, from the constraints editor Attributes tab, or manually in a constraint file. This table describes the methods available to create attribute and directives specifications:

| Entry Method             | Attributes | Directives            |
|--------------------------|------------|-----------------------|
| VHDL Source              | Yes        | Yes                   |
| Verilog Source           | Yes        | Yes                   |
| Compiler Directives File | No         | Supported directives* |
| Constraints Editor       | Yes        | No                    |
| Constraints File         | Yes        | No                    |

It is better to specify attributes in the constraints editor or the constraints file to avoid having to recompile the design if a specification changes. For directives, you must compile the design for them to take effect.

When a compiler directives file, a constraints file, and source code entries are all specified within a design, constraints have the highest priority when there are conflicts followed by CDC compiler directives which take precedence over any HDL source code entries.

Different constraints apply to different database states; some constraints only work for particular database states. See [Valid Constraints for Specific States](#), on page 55 for more information.

For further details, refer to the following:

- [Specifying Attributes and Directives in VHDL](#), on page 463
- [Specifying Attributes and Directives in Verilog](#), on page 465
- [Specifying Directives in a CDC File](#), on page 466

- [Specifying Attributes Using the Constraints Editor](#), on page 467
- [Specifying Attributes in a Constraints File](#), on page 471

## Specifying Attributes and Directives in VHDL

The supported VHDL directives and attributes are predefined in the attributes package in the tool library. This library package contains the built-in attributes and declarations for timing constraints (including black-box timing constraints). The file is located in the install directory at:

*installDirectory/lib/vhd/synattr.vhd*

While there are several methods for adding attributes to objects, you can specify directives only in the source code. There are two ways of defining attributes and directives in VHDL:

- [Using the Predefined VHDL Attributes Package](#), on page 463
- [Declaring VHDL Attributes and Directives](#), on page 464

You can either use the attributes package or re-declare the types of directives and attributes each time you use them. You typically use the attributes package.

### Using the Predefined VHDL Attributes Package

This is the most typical way to specify the attributes. The advantage to using the predefined package is that you avoid redefining the attributes and directives each time you include them in source code. The disadvantage is that your source code is less portable.

1. To use the predefined attributes package from the software library, include these lines ahead of the entity statement:

```
library synplify;
use synplify.attributes.all;
```

2. Add the attribute or directive you want after the design unit declaration.

```
library synplify;
use synplify.attributes.all;
-- design_unit_declarations
attribute productname_attribute of object : object_type is value;
```

The following is an example using `syn_noclockbuf` from the attributes package:

```
library synplify;
use synplify.attributes.all;

entity simpledff is
    port (q: out bit_vector(7 downto 0);
          d : in bit_vector(7 downto 0);
          clk : in bit );
    attribute syn_noclockbuf of clk : signal is true;
```

## Declaring VHDL Attributes and Directives

The alternative method is to declare the attributes to explicitly define them. If you do not use the attributes package, you must redefine the attributes each time you include them in your source code. Every time you use an attribute or directive, define it immediately after the design unit declarations using the following syntax:

```
design_unit_declaration ;
attribute attributeName : dataType ;
attribute attributeName of objectName : objectType is value ;
```

Here is an example using the `syn_noclockbuf` attribute:

```
entity simpledff is
    port (q: out bit_vector(7 downto 0);
          d : in bit_vector(7 downto 0);
          clk : in bit );
    attribute syn_noclockbuf : boolean;
    attribute syn_noclockbuf of clk :signal is true;
```

## Case Sensitivity

Although VHDL is case-insensitive, directives, attributes, and their values are case sensitive and must be declared in the code using the correct case. This rule applies especially for port names in directives.

For example, if a port in VHDL is defined as `GIN`, the following code does not work:

```
attribute black_box_tri_pin : string;
attribute black_box_tri_pin of BBDSLHS : component is "gin";
```

The following code is correct because the case of the port name is correct:

---

```
attribute black_box_tri_pin : string;
attribute black_box_tri_pin of BBDSLHS : component is "GIN";
```

## Specifying Attributes and Directives in Verilog

You can use other methods to add attributes to objects, as described in [For further details, refer to the following:, on page 462](#). However, you can specify directives only in the source code.

Verilog does not have predefined synthesis attributes and directives, so you must add them as comments. The attribute or directive name is preceded by the keyword **synthesis**. Verilog files are case sensitive, so attributes and directives must be specified exactly as presented in their syntax descriptions.

1. To add an attribute or directive in Verilog, use Verilog line or block comment (C-style) syntax directly following the design object. Block comments must precede the semicolon, if there is one.

---

| Verilog Block Comment Syntax                       | Verilog Line Comment Syntax                     |
|----------------------------------------------------|-------------------------------------------------|
| <code>/* synthesis attributeName = value */</code> | <code>// synthesis attributeName = value</code> |
| <code>/* synthesis directoryName = value */</code> | <code>// synthesis directoryName = value</code> |

---

The following are examples of the Verilog syntax:

```
module fifo(out, in) /* synthesis syn_hier = "hard" */;
module b_box(out, in); // synthesis syn_black_box
```

2. To attach multiple attributes or directives to the same object, separate the attributes with white spaces, but do not repeat the **synthesis** keyword. Do not use commas. For example:

```
case state /* synthesis full_case parallel_case */;
```

3. If multiple registers are defined using a single Verilog **reg** statement and an attribute is applied to them, then the synthesis software only applies the last declared register in the **reg** statement. For example:

```
reg [5:0] q, q_a, q_b, q_c, q_d /* synthesis syn_preserve=1 */;
```

The **syn\_preserve** attribute is only applied to **q\_d**. This is the expected behavior for the synthesis tools. To apply this attribute to all registers,

you must use a separate Verilog `reg` statement for each register and apply the attribute.

## Specifying Directives in a CDC File

You can use a compiler directives file to specify certain directives to be added to the source code without making changes directly to your HDL files. During compilation, the tool passes all active compiler constraint files to the compiler. The compiler references the object names in these files with the original RTL objects to assign the corresponding directive.

You can specify the following CDC-compatible directives on views, entities, architectures and modules:

- `syn_black_box`
- `syn_keep`
- `syn_noprune`
- `syn_preserve`
- `syn_rename_module`
- `syn_sharing`
- `syn_unique_inst_module`

Use this procedure to create a CDC file and specify directives:

1. Create a constraints file with a `.cdc` extension that contains the Tcl directives you want.
2. Use the following syntax for the directives you want. Use the syntax that matches the HDL source code you are using.

VHDL    `define_directive {v:[libraryName.]entityName[(architectureName)]} {directive} {value}`

Verilog    `define_directive {v:[libraryName.]moduleName} {directive} {value}`

---

The following example sets the `syn_black_box` attribute on all architectures of the sub entity in the `MyLib` library:

```
define_directive {v:MyLib.sub} {syn_black_box} {1}
```

You must specify the attribute or directive on a view (v:). The *libraryName* and *architectureName* arguments are optional. If you do not specify a library, the tool defaults to all design libraries. If you include an architecture, make sure to enclose it in parentheses. Note that Verilog objects are case-sensitive, but VHDL objects are not.

3. Add the file or files to the database. CDC files are added through the report rtl\_diagnostics and report syntax\_check commands or by the run compile and run pre\_instrument commands using the -cdc or -cdclist argument.
4. Running any of the above commands with the -cdc or -cdclist argument passes the specified CDC files to the compiler. During compilation, the compiler references the object names in these files with the original RTL objects and assigns the corresponding directives.

## Specifying Attributes Using the Constraints Editor

The constraints editor window provides an easy-to-use interface to add any attribute. You cannot use it for adding directives, because they must be added to the source files. (See [Specifying Attributes and Directives in VHDL, on page 463](#) or [Specifying Attributes and Directives in Verilog, on page 465](#)). The following procedure shows how to add an attribute directly in the constraints editor window.

1. Start with a compiled design, then open the constraint editor window:  

```
edit fdc -mode gui filename.fdc
```
2. Click the Attributes tab at the bottom of the constraints editor window. You can either select the object first (step 3) or the attribute first (step 4).

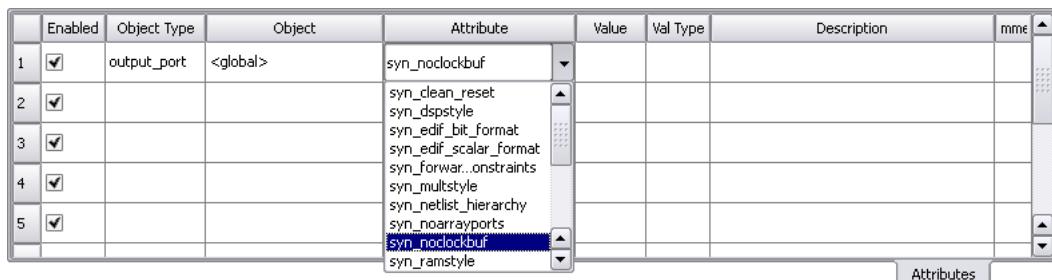
|   | Enable                              | Object Type | Object   | Attribute      | Value | Value Type | Description             | Comment |
|---|-------------------------------------|-------------|----------|----------------|-------|------------|-------------------------|---------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Global> | syn_noclockbuf | 1     | boolean    | Use normal input buffer |         |
| 2 |                                     |             |          |                |       |            |                         |         |
| 3 |                                     |             |          |                |       |            |                         |         |

3. To specify the object, do one of the following in the Object column. If you already specified the attribute, the Object column lists only valid object choices for that attribute.

- Select the type of object in the Object Filter column, and then select an object from the list of choices in the Object column. This is the best way to ensure that you are specifying an object that is appropriate, with the correct syntax.
- Drag the object to which you want to attach the attribute from the HDL Analyst view to the Object column in the constraints editor window. For some attributes, dragging and dropping may not select the desired object. For example, if you want to set syn\_hier on a module or entity such as an and gate, you must set the attribute on the view for that module/entity. The object would have the syntax v:*moduleName* in Verilog or v:*library.moduleName* in VHDL, where you can have multiple libraries.
- Enter the name of the object in the Object column. If you do not know the name, use the Find command or the Object Filter column. Make sure to enter the appropriate prefix for the object where it is needed. For example, to set an attribute on a view, add the v: prefix to the module or entity name. For VHDL, it may be necessary to specify the library as well as the module name.

If you specified the object first, you can now specify the attribute. The list shows only the valid attributes for the object type selected.

4. Specify the attribute by holding down the mouse button in the Attribute column and selecting the desired attribute from the list displayed.



The screenshot shows a table-based interface for specifying attributes. The columns are: Enabled, Object Type, Object, Attribute, Value, Val Type, Description, and mme. The 'Enabled' column has checkboxes. The 'Object Type' column contains 'output\_port'. The 'Object' column contains '<global>'. The 'Attribute' column is a dropdown menu. The dropdown menu lists several attributes: syn\_noclockbuf, syn\_clean\_reset, syn\_dspstyle, syn\_edif\_bit\_format, syn\_edif\_scalar\_format, syn\_forwar\_onstraints, syn\_multstyle, syn\_netlist\_hierarchy, syn\_noarrayports, syn\_noclockbuf (which is highlighted in blue), and syn\_ramstyle. A scroll bar is visible on the right side of the dropdown menu. At the bottom right of the table area, there is a button labeled 'Attributes'.

If you select the object first, the choices available are determined by the selected object.

When you select an attribute, the constraints editor window tells you the type of value you must enter for that attribute and provides a brief description of the attribute. If you selected the attribute first, make sure to go back and specify the object.

5. Fill out the value. Hold down the mouse button in the Value column, and select from the list. You can also type in a value.
6. Save the file. The software creates a Tcl constraint file composed of `define_attribute` statements for the attributes you specified. .
7. Include the constraint file with any of the following database commands using an `-fdc` or `-fdclist` argument:
  - `report constraint_check`
  - `run pre_map`
  - `run pre_partition`
  - `run system_generate`

## Attributes Panel

The Attributes panel includes the following columns.

| Column      | Description                                                                                                                                                                                                                                                                                                                                                                            |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Enabled     | (Required) Enables the selection.                                                                                                                                                                                                                                                                                                                                                      |
| Object Type | Specifies the type of object to which the attribute is assigned. Choose from the pull-down list to filter the available choices in the Object field.                                                                                                                                                                                                                                   |
| Object      | (Required) Specifies the object to which the attribute is attached. This field is synchronized with the Attribute field, so selecting an object here filters the available choices in the Attribute field. You can also drag and drop an object from the HDL Analyst view into this column.                                                                                            |
| Attribute   | (Required) Specifies the attribute name. You can choose from a pull-down list that includes all available attributes. This field is synchronized with the Object field. If you select an object first, the attribute list is filtered. If you select an attribute first, the available choices are filtered in the Object field. You must select an attribute before entering a value. |
| Value       | (Required) Specifies the attribute value. You must specify the attribute first. Clicking in the column displays the default value; a drop-down arrow lists available values where appropriate.                                                                                                                                                                                         |

|             |                                                                                |
|-------------|--------------------------------------------------------------------------------|
| Val Type    | Specifies the type of value for the attribute. For example, string or boolean. |
| Description | Contains a one-line description of the attribute.                              |
| Comment     | Contains any comments you want to add about the attributes.                    |

When you use the constraints editor spreadsheet to create and modify a constraint file, the proper `define_attribute` or `define_global_attribute` statement is automatically generated for the constraint file. The following shows the syntax for these statements as they appear in the constraint file.

```
define_attribute {object} attributeName {value}  
define_global_attribute attributeName {value}
```

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>object</i>        | The design object, such as module, signal, input, instance, port, or wire name. The object naming syntax varies, depending on if your source code is in Verilog or VHDL format. See <a href="#">syn_black_box</a> , on page 526 for details about the syntax conventions. If you have mixed input files, use the object naming syntax appropriate for the format in which the object is defined. Global attributes, since they apply to an entire design, do not use an <i>object</i> argument. |
| <i>attributeName</i> | The name of the synthesis attribute. This must be an attribute, not a directive, as directives are not supported in constraint files.                                                                                                                                                                                                                                                                                                                                                           |
| <i>value</i>         | String, integer, or boolean value.                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

## Specifying Attributes in a Constraints File

When you use the constraints editor window ([Specifying Attributes Using the Constraints Editor, on page 467](#)), the attributes are automatically written to a constraint file using the Tcl `define_attribute` syntax. This is the preferred method for defining constraints as the syntax is determined for you.

However, the following procedure explains how you can specify attributes directly in the constraint file.

1. Open a file in a text editor.
2. Enter the desired attributes. For example,

```
define_attribute {objectName} attributeName value
```

3. Save the constraints in a file using the FDC file extension.

## Global Attributes

A number of attributes can be defined either globally, (values are applied to all objects of the specified type in the design) or locally (values are applied only to a specified design object such as a module, view, port, instance, or clock). When an attribute is set both globally and locally on a design object, the local specification overrides the global specification for the object.

The syntax for specifying a global attribute in a constraint file is:

```
define_global_attribute attributeName {value}
```

The following code excerpt provides an example of attributes defined in the constraint file:

```
# Assign a location for scalar port "sel".
define_attribute {sel} xc_loc "P139"

# Assign a pad location to all bits of a bus.
define_attribute {b[7:0]} xc_loc "P14, P12, P11, P5, P21,
    P18, P16, P15"

# Assign a fast output type to the pad.
define_attribute {a[5]} xc_fast 1

# Use a regular buffer instead of a clock buffer
# for clock "clk_slow".
define_attribute {clk_slow} syn_noclockbuf 1
```

```
# Relax timing by not buffering "clk_slow", because it is
# the slow clock. Set the maximum fanout to 10000.
define_attribute {clk_slow} syn_maxfan 10000
```

# Attributes and Directives Summary

All attributes and directives supported for synthesis are listed in alphabetical order. Each command includes syntax, option and argument descriptions, and examples. The following attributes and directives are listed in alphabetical order:

|                            |                                          |                                |
|----------------------------|------------------------------------------|--------------------------------|
| black_box_pad_pin          | black_box_tri_pins                       | CLOCK_DEDICATED_ROUTE          |
| diff_term                  | full_case                                | loop_limit                     |
| corrupt_pd                 | pragma translate_off/pragma translate_on | syn_allow_retiming             |
| parallel_case              | syn_assign_to_region                     | syn_assign_to_srl              |
| syn_allowed_resources      | syn_auto_insert_bufg                     | syn_auto_insert_bufgmux        |
| syn_async_reg              | syn_bram_cascade_height                  | syn_clean_reset                |
| syn_black_box              | syn_clock_priority                       | syn_connect_hrefs              |
| syn_clock_gmux_proxy       | syn_cp_use_fast_synthesis                | syn_diff_io                    |
| syn_direct_enable          | syn_direct_reset                         | syn_direct_set                 |
| syn_disable_purifyclock    | syn_dspstyle                             | syn_edif_bit_format            |
| syn_edif_scalar_format     | syn_encoding                             | syn_enum_encoding              |
| syn_fast_auto              | syn_force_seq_prim                       | syn_formal_blackbox            |
| syn_forward_io_constraints | syn_gatedclk_clock_en                    | syn_gatedclk_clock_en_polarity |
| syn_global_buffers         | syn_hier                                 | syn_implement                  |
| syn_insert_buffer          | syn_insert_pad                           | syn_isclock                    |
| syn_keep                   | syn_latch_ramstyle                       | syn_loc                        |
| syn_loopleft               | syn_macro                                | syn_map_dffrs                  |
| syn_maxfan                 | syn_multstyle                            | syn_netlist_hierarchy          |
| syn_noarrayports           | syn_noclockbuf                           | syn_no_compile_point           |
| syn_noprune                | syn_packer_effort_level                  | syn_pad_type                   |
| syn_partition_keep         | syn_partition_preserve                   | syn_pipeline                   |
| syn_pnr_preserve_regs      | syn_preserve                             | syn_probe                      |
| syn_ramstyle               | syn_ram_write_mem                        | syn_reduce_controlset_size     |

|                                    |                                     |                                         |
|------------------------------------|-------------------------------------|-----------------------------------------|
| <code>syn_reference_clock</code>   | <code>syn_rename_module</code>      | <code>syn_replicate</code>              |
| <code>syn_resources</code>         | <code>syn_ret_lib_cell_type</code>  | <code>syn_ret_type</code>               |
| <code>syn_romstyle</code>          | <code>syn_rw_conflict_logic</code>  | <code>syn_sharing</code>                |
| <code>syn_shift_resetphase</code>  | <code>syn_slow</code>               | <code>syn_srl_mindepth</code>           |
| <code>syn_srlstyle</code>          | <code>syn_state_machine</code>      | <code>syn_tco&lt;n&gt;</code>           |
| <code>syn_tpd&lt;n&gt;</code>      | <code>syn_trace_attr</code>         | <code>syn_tristate</code>               |
| <code>syn_tsu&lt;n&gt;</code>      | <code>syn_unconnected_inputs</code> | <code>syn_unique_inst_module</code>     |
| <code>syn_upf_ret_port_type</code> | <code>syn_useenables</code>         | <code>syn_useioff</code>                |
| <code>syn_useoddr</code>           | <code>syn_user_instance</code>      | <code>translate_off/translate_on</code> |
| <code>xc_area_group</code>         | <code>xc_clockbuftype</code>        | <code>xc_fast</code>                    |
| <code>xc_fast_auto</code>          | <code>xc_global_buffers</code>      | <code>xc_isgsr</code>                   |
| <code>xc_loc</code>                | <code>xc_map</code>                 | <code>xc_padtype</code>                 |
| <code>xc_pullup/xc_pulldown</code> | <code>xc_rloc</code>                | <code>xc_slow</code>                    |
| <code>xc_use_keep_hierarchy</code> | <code>xc_use_rpms</code>            | <code>xc_use_timespec_for_io</code>     |
| <code>xc_usest</code>              |                                     |                                         |

## black\_box\_pad\_pin

### Directive

Specifies that the pins on a black box are I/O pads visible to the outside environment.

### Description

Used with the `syn_black_box` directive to specify which pins on a black box are I/O pads visible to the outside environment. To specify more than one port as an I/O pad, list the ports inside double-quotes ("") separated by commas. To instantiate an I/O, you usually do not need to define a black box or this directive as the synthesis software provides predefined black boxes for vendor I/Os.

The `black_box_pad_pin` directive is one of several directives that you can use with the `syn_black_box` directive to define timing for a black box. See [syn\\_black\\_box, on page 526](#) for a list of the related directives.

### black\_box\_pad\_pin Syntax

This table summarizes the syntax:

|                                                                                   |                                 |
|-----------------------------------------------------------------------------------|---------------------------------|
| Verilog <code>object /* synthesis black_box_pad_pin = portList */;</code>         | <a href="#">Verilog Example</a> |
| VHDL <code>attribute black_box_pad_pin of object : objectType is portList;</code> | <a href="#">VHDL Example</a>    |

In the above syntax, `object` is a module or architecture declaration of a black box and `portList` is a comma-separated list of the names of the black box ports that are I/O pads.

### Verilog Example

This code segment shows how to specify the directive in Verilog:

```
module BBDLHS (D,E,GIN,GOUT,PAD,Q)
    /* synthesis syn_black_box black_box_pad_pin="GIN[2:0],Q" */;
```

### VHDL Example

The example below shows how to specify this directive in VHDL code:

```

library ieee;
use ieee.std_logic_1164.all;

entity top is
generic (width : integer := 4);
port (in1,in2 : in std_logic_vector(width downto 0);
      clk : in std_logic;
      q : out std_logic_vector (width downto 0) );
end top;

architecture top1_arch of top is
component test is
generic (width1 : integer := 2);
port (in1,in2 : in std_logic_vector(width1 downto 0);
      clk : in std_logic;
      q : out std_logic_vector (width1 downto 0) );
end component;
attribute syn_black_box : boolean;
attribute syn_black_box of test : component is true;
attribute black_box_pad_pin : string;
attribute black_box_pad_pin of test : component is
  "in1(4:0),in2[4:0],q(4:0)";
begin
  test123 : test generic map (width) port map (in1,in2,clk,q);
end top1_arch;

```

## Effect of Using `black_box_pad_pin`

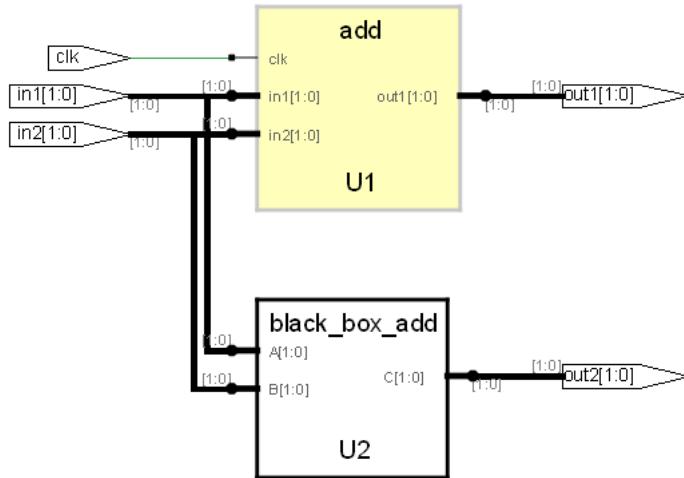
The following examples show a netlist before and after applying the attribute.

### Before using `black_box_pad_pin`

```

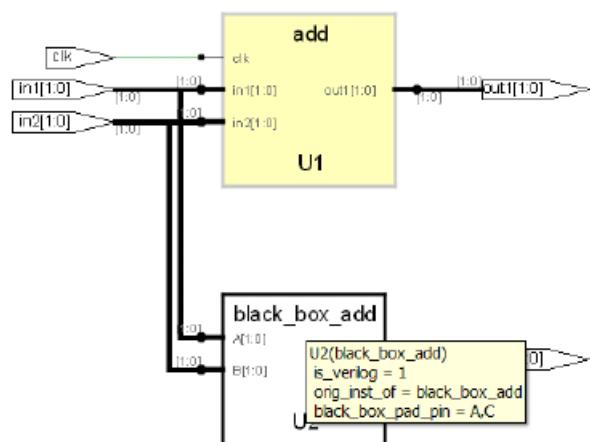
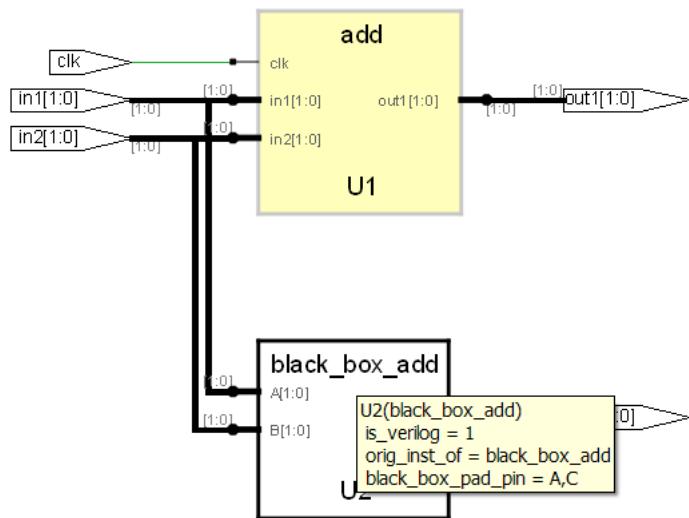
}
(cell black_box_add (cellType GENERIC)
  (view verilog (viewType NETLIST)
    (interface
      (port (array (rename A "A[1:0]" ) 2) (direction INPUT)
        (port (array (rename B "B[1:0]" ) 2) (direction INPUT)
          (port (array (rename C "C[1:0]" ) 2) (direction OUTPUT))
      )
      (property orig_inst_of (string "black_box_add"))
    )
  )
)

```



After using `black_box_pad_pin`

```
)  
(cell black_box_add (cellType GENERIC)  
  (view verilog (viewType NETLIST)  
    (interface  
      (port (array (rename A "A[1:0]") 2) (direction INPUT)  
        (port (array (rename B "B[1:0]") 2) (direction INPUT)  
          (port (array (rename C "C[1:0]") 2) (direction OUTPUT))  
      )  
      (property black_box_pad_pin (string "A,C"))  
      (property orig_inst_of (string "black_box_add"))  
    )  
  )
```



## black\_box\_tri\_pins

### *Directive*

Specifies that an output port on a black box component is a tristate.

### Description

Used with the `syn_black_box` directive to specify that an output port on a black box component is a tristate. This directive eliminates multiple-driver errors when the output of a black box has more than one driver. To specify more than one tristate port, list the ports inside double-quotes ("") separated by commas.

The `black_box_tri_pins` directive is one of several directives that you can use with the `syn_black_box` directive to define timing for a black box. See [syn\\_black\\_box, on page 526](#) for a list of the associated directives.

### black\_box\_tri\_pins Syntax

This table summarizes the syntax in different files:

|         |                                                                               |                                 |
|---------|-------------------------------------------------------------------------------|---------------------------------|
| Verilog | <code>object /* synthesis black_box_tri_pins = portList */;</code>            | <a href="#">Verilog Example</a> |
| VHDL    | <code>attribute black_box_tri_pins of object : objectType is portList;</code> | <a href="#">VHDL Example</a>    |

In the above syntax, `object` is a module or architecture declaration of a black box and `portList` is a comma-separated list of the names of the tristate output port names.

### Verilog Example

Here is an example with a single port name:

```
module BBDSLHS(D,E,GIN,GOUT,PAD,Q)
/* synthesis syn_black_box black_box_tri_pins="PAD" */;
```

Here is an example with a list of multiple pins:

```
module bb1(D,E,tri1,tri2,tri3,Q)
/* synthesis syn_black_box black_box_tri_pins="tri1,tri2,tri3" */;
```

For a bus, specify the port name followed by the bit range:

```
module bb1(D,bus1,E,GIN,GOUT,Q)
/* synthesis syn_black_box black_box_tri_pins="bus1[7:0]" */;
```

## VHDL Example

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

package my_components is
component BBDSLHS
    port (D: in std_logic;
          E: in std_logic;
          GIN : in std_logic;
          GOUT : in std_logic;
          PAD : inout std_logic;
          Q: out std_logic );
end component;
attribute syn_black_box : boolean;
attribute syn_black_box of BBDSLHS : component is true;
attribute black_box_tri_pins : string;
attribute black_box_tri_pins of BBDSLHS : component is "PAD";
end package my_components;
```

Multiple pins on the same component can be specified as a list:

```
attribute black_box_tri_pins of bb1 : component is
    "tri,tri2,tri3";
```

To apply this directive to a port that is a bus, specify the name followed by the bit range:

```
attribute black_box_tri_pins of bb1 : component is "bus1[7:0]";
```

# CLOCK\_DEDICATED\_ROUTE

## *Attribute*

Allows the place-and-route tool to override clock placement rules. The attribute can only be applied to clock objects through a constraint file.

## Description

The default is TRUE, which enforces clock-placement rules to report clock components that are not in compliance. To ignore the rules and any related violations, set the attribute to FALSE. In this mode, place and route converts the CLOCK\_DEDICATED\_ROUTE value to an XDC constraint, and continues even if there are violations.

Use the FALSE setting only when absolutely necessary. It is recommended that you fix all clock-placement errors before proceeding.

You can specify BACKBONE when location constraints are assigned that violate basic clock placement rules (but is not generally recommended) or when an MMCM or PLL is placed far from the source CCIP pin. The extra wire length may add delay to the timing path from the CCIP pin to the MMCM or PLL. Use BACKBONE if the design meets timing with the added delay.

## Constraint File Syntax

|          |                                                                                              |                                            |
|----------|----------------------------------------------------------------------------------------------|--------------------------------------------|
| FDC File | <code>define_attribute {objectName}<br/>CLOCK_DEDICATED_ROUTE {FALSE  TRUE  BACKBONE}</code> | <a href="#">Constraints Editor Example</a> |
|----------|----------------------------------------------------------------------------------------------|--------------------------------------------|

In the above syntax, *objectName* is the name of the clock object. The clock object can include net of a clock driver instance, I/O pin that is the clock driver, input or output pins on clock primitives BUFG, BUFR, DCM, PLLPMCD, GT11, GT11 DUAL, GT11 CLK, or GTP DUAL.

For example:

```
define_attribute {n:clk} CLOCK_DEDICATED_ROUTE FALSE
```

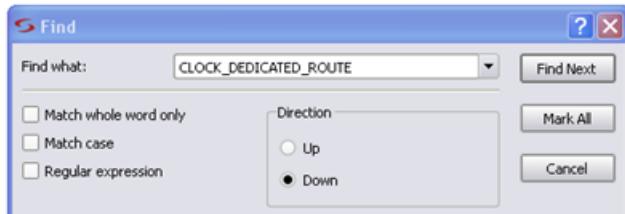
## Constraints Editor Example

|   | Enable                              | Object Type | Object | Attribute             | Value | Value Type | Description |
|---|-------------------------------------|-------------|--------|-----------------------|-------|------------|-------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | n:clk  | CLOCK_DEDICATED_ROUTE | FALSE |            |             |

## Effect of Using CLOCK\_DEDICATED\_ROUTE

The following example shows the effect of applying the attribute. The constraints are forward-annotated to the place-and-route stage. Check for the presence of the CLOCK\_DEDICATED\_ROUTE attribute in the edif file.

```
(net (rename q_c_8 "q_c[8]") (joined
  (portRef Q (instanceRef q202_8))
  (portRef I (instanceRef qobuf_8)))
)
(net (rename q_8 "q[8]") (joined
  (portRef O (instanceRef qobuf_8))
  (portRef (member q 1)))
)
(net (rename q_c_9 "q_c[9]") (joined
  (portRef Q (instanceRef q202_9))
  (portRef I (instanceRef qobuf_9)))
)
(net (rename q_9 "q[9]") (joined
  (portRef O (instanceRef qobuf_9))
  (portRef (member q 0)))
)
(net clk_c (joined
  (portRef I (instanceRef clk_keep_cb))
  (portRef O (instanceRef clk_ibuf)))
)
(property CLOCK_DEDICATED_ROUTE (integer 0))
)
(net (rename cnt_cry202_1 "cnt_cry[1]") (joined
  (portRef L0 (instanceRef cnt_cry_1))
  (portRef CI (instanceRef cnt_cry_2))
  (portRef CI (instanceRef cnt_s_2)))
)
(net (rename cnt_s202_1 "cnt_s[1]") (joined
  (portRef O (instanceRef cnt_s_1))
  (portRef D (instanceRef cnt_1)))
)
```



## diff\_term

### *Attribute*

Use this attribute to specify Differential Termination for true differential input I/O standards. You can use this attribute with inputs for the buffer, such as IBUFDS and OBUFDS. Set `dff_term` in the constraint file or HDL source code.

### Constraint File Syntax

```
define_attribute {input} diff_term {true|false}
```

In the above syntax:

- *input* can be specified for any of the following inferred primitives: IBUFDS, IBUFGDS, OBUFDS, OBUFGDS, or IOBUFGDS.
- The `diff_term` data type is Boolean.
  - A value of `true` enables differential termination on the pin.
  - A value of `false` disables differential termination on the pin.

For example:

```
define_attribute {p:in1_p} {diff_term} {true}  
define_attribute {p:in1_n} {diff_term} {true}
```

### Verilog Syntax and Example

```
object /* synthesis diff_term=true|false */
```

```
module syn_diff_io (in1_p, in1_n, d, out1);  
    input in1_p /*synthesis diff_term = true*/,  
          in1_n /*synthesis diff_term = true*/, d;  
    output reg out1;  
    reg in1;  
  
    always@(in1_p, in1_n)  
        if (in1_p != in1_n)  
            in1 = in1_p;  
        else  
            in1 = in1;
```

```
always@(posedge in1)
    out1 <= d;

endmodule
```

## VHDL Syntax and Example

```
attribute diff_term of object : objectType is true | false;

library IEEE;
use IEEE.std_logic_1164.all;

entity test is
    port (in1_p : in std_logic;
          in1_n : in std_logic;
          d : in std_logic;
          out1 : out std_logic );
attribute diff_term: boolean;
attribute diff_term of in1_p,in1_n: signal is true;
end test;

architecture arc of test is
signal in1 : std_logic;
begin
    process(in1_p,in1_n)
    begin
        if (in1_p /= in1_n) then
            in1 <= in1_p;
        else
            in1 <= in1;
        end if;
    end process;

    process(in1)
    begin
        if (in1'event and in1='1') then
            out1 <= d;
        end if;
    end process;
end arc;
```

## **full\_case**

### *Directive*

For Verilog designs only. Indicates that all possible values have been given, and that no additional hardware is required to preserve signal values.

### **Description**

When used with a case, casex, or casez statement, this directive indicates that all possible values have been given, and that no additional hardware is required to preserve signal values.

### **full\_case Values Syntax**

This table summarizes the syntax in the following file type:

Verilog

*object /\* synthesis full\_case \*/;*

[Verilog Examples](#)

### **Verilog Examples**

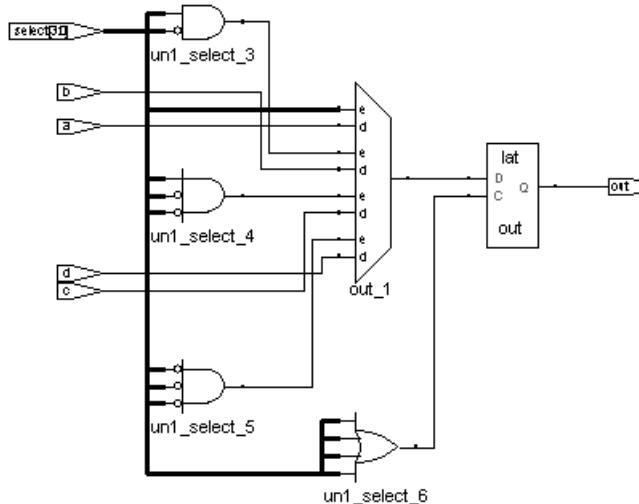
The following casez statement creates a 4-input multiplexer with a pre-decoded select bus (a decoded select bus has exactly one bit enabled at a time):

```

module muxnew1 (out, a,
b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;

always @ (select or a or
b or c or d)
begin
  casez (select)
    4'b???1: out = a;
    4'b??1?: out = b;
    4'b?1??: out = c;
    4'b1???: out = d;
  endcase
end
endmodule

```



This code does not specify what to do if the `select` bus has all zeros. If the `select` bus is being driven from outside the current module, the current module has no information about the legal values of `select`, and the synthesis tool must preserve the value of output `out` when all bits of `select` are zero. Preserving the value of `out` requires the tool to add extraneous level-sensitive latches if `out` is not assigned elsewhere through every path of the `always` block. A warning message such as the following is issued:

"Latch generated from always block for signal out, probably missing assignment in branch of if or case."

If you add the `full_case` directive, it instructs the synthesis tool not to preserve the value of `out` when all bits of `select` are zero.

```

module muxnew3 (out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;

```

```

always @(select or a or b or c or d) begin
  casez (select) /* synthesis full_case */
    4'b???1: out = a;
    4'b??1?: out = b;
    4'b?1???: out = c;
    4'b1????: out = d;
  endcase
end
endmodule

```

If the select bus is decoded in the same module as the case statement, the synthesis tool automatically determines that all possible values are specified, so the `full_case` directive is unnecessary.

## Assigned Default and `full_case`

As an alternative to `full_case`, you can assign a default in the case statement. The default is assigned a value of 'bx (a 'bx in an assignment is treated as a “don't care”). The software assigns the default at each pass through the `casez` statement in which the select bus does not match one of the explicitly given values; this ensures that the value of `out` is not preserved and no extraneous level-sensitive latches are generated.

The following code shows a default assignment in Verilog:

```

module muxnew2 (out, a, b, c, d, select);
  output out;
  input a, b, c, d;
  input [3:0] select;
  reg out;

  always @(select or a or b or c or d)
  begin
    casez (select)
      4'b???1: out = a;
      4'b??1?: out = b;
      4'b?1???: out = c;
      4'b1????: out = d;
      default: out = 'bx;
    endcase
  end
endmodule

```

Both techniques help keep the code concise because you do not need to declare all the conditions of the statement. The following table compares the techniques:

| <b>Default Assignment</b>                                                                        | <b>full_case</b>                                                                                             |
|--------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| Stays within Verilog to get the desired hardware                                                 | Must use a synthesis directive to get the desired hardware                                                   |
| Helps simulation debugging because you can easily find that the invalid select is assigned a 'bx | Can cause mismatches between pre- and post-synthesis simulation because the simulator does not use full_case |

## loop\_limit

### *Directive*

Specifies a loop iteration limit for a for loop in a Verilog design when the loop index is a variable, not a constant.

### Description

Verilog designs only. For VHDL applications ,use the syn\_looplimate directive (see [syn\\_looplimate, on page 676](#)).

Specifies a loop iteration limit for a for loop on a per-loop basis when the loop index is a variable, not a constant. The compiler uses the default iteration limit of 2000 when the exit or terminating condition does not compute a constant value, or to avoid an infinite loop. The default limit ensures the effective use of runtime and memory resources.

If your design requires a variable loop index or if the number of loops is greater than the default limit, use the loop\_limit directive to specify a new limit for the compiler. If you do not, you get a compiler error. You must hard code the limit at the beginning of the loop statement, and the limit cannot be an expression. The higher the value you set, the longer the runtime.

Alternatively, you can use the option set looplimate command (Loop Limit option) to set a global loop limit that overrides the default of 2000 loops in the RTL.

### loop\_limit Syntax

This table summarizes the syntax in the following file:

Verilog    *start\_of\_loop\_statement /\* synthesis loop\_limit integer \*/;*

[Verilog Example](#)

## Verilog Example

The following is an example where the loop limit is set to 2010:

```
module test(din,dout,clk);
    input[1999 : 0] din;
    input clk;
    output[1999 : 0] dout;
    reg[1999 : 0] dout;
    integer i;

    always @ (posedge clk)
    begin
        /* synthesis loop_limit 2010 */
        for(i=0;i<=1999;i=i+1)
        begin
            dout[i] <= din[i];
        end
    end
endmodule
```

## Effect of Using `loop_limit`

### Before using `loop_limit`

If the code has more than 2000 loops and the attribute is not set, the tool will produce an error.

```
@E:CS162 : loop_limit.v(10) | Loop iteration limit 2000 exceeded
      - add '// synthesis loop_limit 4000' before the loop construct
```

### After using `loop_limit`

The number of loops will not generate the `loop_limit` error.

## parallel\_case

### *Directive*

For Verilog designs only. Forces a parallel-multiplexed structure rather than a priority-encoded structure.

### Description

case statements are defined to work in priority order, executing (only) the first statement with a tag that matches the select value. The parallel\_case directive forces a parallel-multiplexed structure rather than a priority-encoded structure.

If the select bus is driven from outside the current module, the current module has no information about the legal values of select, and the software must create a chain of disabling logic so that a match on a statement tag disables all following statements.

However, if you know the legal values of select, you can eliminate extra priority-encoding logic with the parallel\_case directive. In the following example, the only legal values of select are 4'b1000, 4'b0100, 4'b0010, and 4'b0001, and only one of the tags can be matched at a time. Specify the parallel\_case directive to a case, casex, or casez statement declaration so that tag-matching logic can be parallel and independent, instead of chained.

### parallel\_case Syntax

This table summarizes the syntax in the following file type:

| Verilog | object /* synthesis parallel_case */ | Verilog Example |
|---------|--------------------------------------|-----------------|
|---------|--------------------------------------|-----------------|

### Verilog Example

You specify the directive as a comment immediately following the select value of the case statement.

```
module muxnew4 (out, a, b, c, d, select);
    output out;
    input a, b, c, d;
    input [3:0] select;
    reg out;
```

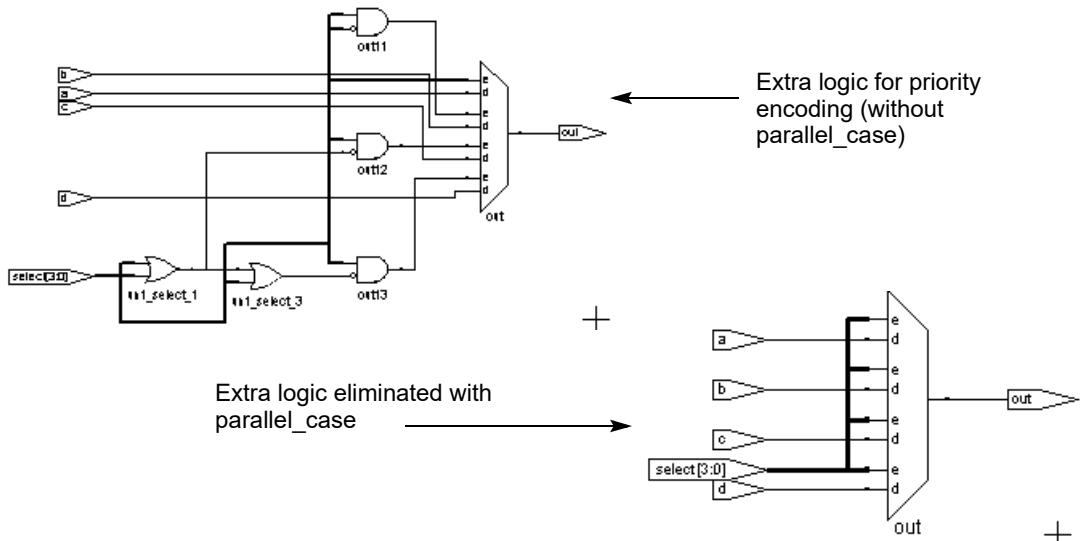
```

always @(select or a or b or c or d) begin
  casez (select) /* synthesis parallel_case */
    4'b???1: out = a;
    4'b??1?: out = b;
    4'b?1???: out = c;
    4'b1????: out = d;
    default: out = 'bx;
  endcase
end
endmodule

```

If the select bus is decoded within the same module as the case statement, the parallelism of the tag matching is determined automatically, and the parallel\_case directive is unnecessary.

## Effect of Using parallel\_case



## pragma translate \_off/pragma translate \_on

### *Directive*

Allows you to synthesize designs originally written for use with other synthesis tools without needing to modify source code. All source code that is between these two directives is ignored during synthesis.

### Description

Another use of these directives is to prevent the synthesis of stimulus source code that only has meaning for logic simulation. You can use pragma translate\_off/translate\_on to skip over simulation-specific lines of code that cannot be synthesized.

When you use pragma translate\_off in a module, synthesis of all source code that follows is halted until pragma translate\_on is encountered. Every pragma translate\_off must have a corresponding pragma translate\_on. These directives cannot be nested, therefore, the pragma translate\_off directive can only be followed by a pragma translate\_on directive.

---

**Note:** See also, [translate\\_off/translate\\_on, on page 869](#). These directives are implemented the same in the source code.

---

This table summarizes the syntax in the following file types:

|         |                                                                                                                                        |                                 |
|---------|----------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| Verilog | <code>/* pragma translate_off */<br/>/* pragma translate_on */<br/>/*synthesis translate_off */<br/>/*synthesis translate_on */</code> | <a href="#">Verilog Example</a> |
| VHDL    | <code>--pragma translate_off<br/>--pragma translate_on<br/>--synthesis translate_off<br/>--synthesis translate_on</code>               | <a href="#">VHDL Example</a>    |

## Verilog Example

```
module test (input a, b, output dout, Nout);
  assign dout = a + b;

  // Anything between synthesis translate_off and translate_on is
  // ignored during synthesis so that only the adder circuit
  // above is implemented and not the enclosed multiplier
  // circuit below:

  /* synthesis translate_off */
  assign Nout = a * b;
  /* synthesis translate_on */
endmodule
```

## VHDL Example

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

entity test is
  port (a : in std_logic_vector(1 downto 0);
        b : in std_logic_vector(1 downto 0);
        dout : out std_logic_vector(1 downto 0);
        Nout : out std_logic_vector(3 downto 0));
end;

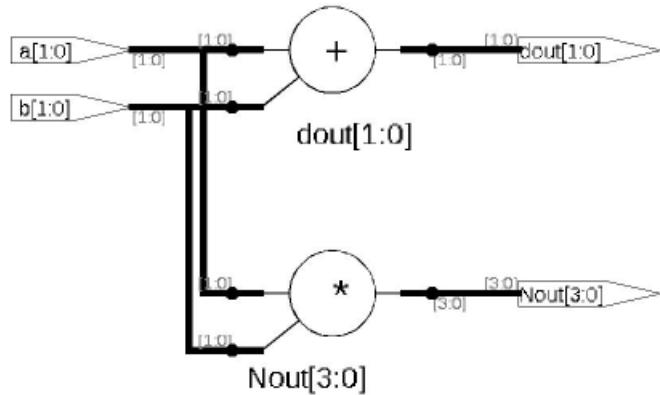
architecture rtl of test is
begin
  dout <= a + b;

  -- Anything between pragma translate_off and translate_on is
  -- ignored during synthesis so that only the adder circuit
  -- above is implemented and not the enclosed multiplier circuit
  -- below:

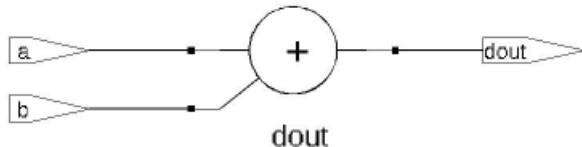
  --pragma translate_off
  --  Nout <= a * b;
  --pragma translate_on
end;
```

## Effect of Using pragma translate\_off/pragma translate\_on

Before applying the attribute:



After applying the attribute:



## **syn\_allow\_retimining**

### *Attribute*

Determines if registers can be moved across combinational logic to improve performance.

### **Description**

The `syn_allow_retimining` attribute determines if registers can be moved across combinational logic to improve performance.

The attribute can be applied either globally through a constraint file or on specific registers. Typically, you enable global Retiming using the option set `retiming 1` switch in Tcl and use the `syn_allow_retimining` attribute to selectively disable retiming for specific registers that you do not want moved. Do not use the `syn_allow_retimining` attribute with the Synthesis Strategy fast option.

### **syn\_allow\_retimining Syntax**

You can specify the attribute in the following files:

|         |                                                                                                                                        |                                 |
|---------|----------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| FDC     | <code>define_attribute {register} syn_allow_retimining {1 0}</code><br><code>define_global_attribute syn_allow_retimining {1 0}</code> | <a href="#">FDC Example</a>     |
| Verilog | <code>object /* synthesis syn_allow_retimining = 1 0 */ ;</code>                                                                       | <a href="#">Verilog Example</a> |
| VHDL    | <code>attribute syn_allow_retimining of object : objectType is true   false ;</code>                                                   | <a href="#">VHDL Example</a>    |

### **FDC Example**

```
define_attribute {register} syn_allow_retimining {1|0}
define_global_attribute syn_allow_retimining {1|0}
```

| Enable                              | Object Type | Object   | Attribute            | Value | Value Type | Description                 |
|-------------------------------------|-------------|----------|----------------------|-------|------------|-----------------------------|
| <input checked="" type="checkbox"/> | <any>       | <Global> | syn_allow_retimining | 1     | boolean    | Controls retiming of reg... |

## Verilog Example

Here is an example of applying it to a register:

```
module parity_check (clk,data,count_one);
    input clk;
    input [20:0]data ;
    output reg [3:0]count_one /* synthesis syn_allow_retimming=1 */;
    integer i;
    reg parity= 1'b1;

    always @ (posedge clk)
    begin
        for (i=0; i<21; i=i+1)
            if (data[i] == parity)
                count_one<=count_one+1;
    end
endmodule
```

## VHDL Example

The data type is Boolean. Here is an example of applying it to a register:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY ones_cnt IS
    PORT (vin : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          vout : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
          clk : IN STD_LOGIC );
END ones_cnt;

ARCHITECTURE lan OF ones_cnt IS
    signal vout_reg : STD_LOGIC_VECTOR (3 DOWNTO 0);
    attribute syn_allow_retiming : boolean;
    attribute syn_allow_retiming of vout_reg : signal is true;
    BEGIN
        gen_vout: PROCESS(clk,vin)
        VARIABLE count : STD_LOGIC_VECTOR(vout'RANGE);
        BEGIN
            if rising_edge(clk) then
                count := (OTHERS => '0');
                FOR I IN vin'RANGE LOOP
                    count := count + vin(i);
                END LOOP;
```

```

        vout_reg <= count;
    end if;
    vout <= vout_reg;
END PROCESS gen_vout;
END lan;
```

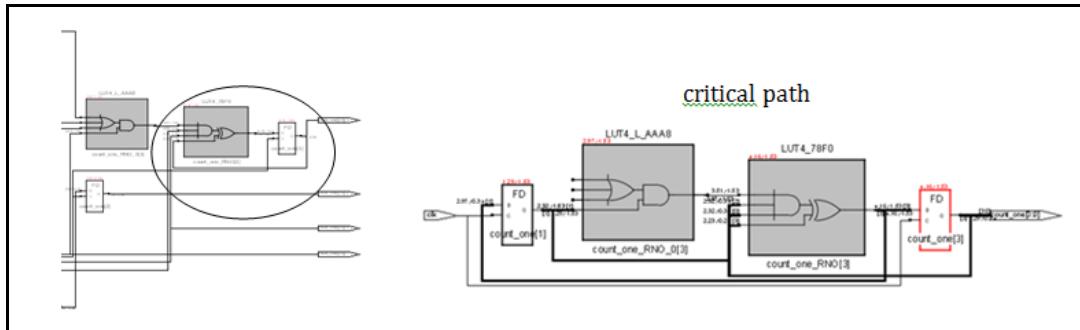
## Effect of Using syn\_allow\_retiming

Before applying syn\_allow\_retiming.

Verilog    output reg [3:0]count\_one /\* synthesis syn\_allow\_retiming=0\*/;

VHDL    attribute syn\_allow\_retiming of vout\_reg : signal is false;

The critical path and the worst slack for this scenario are given below along with the original count\_one [3] register (before being retimed) as found in the design.

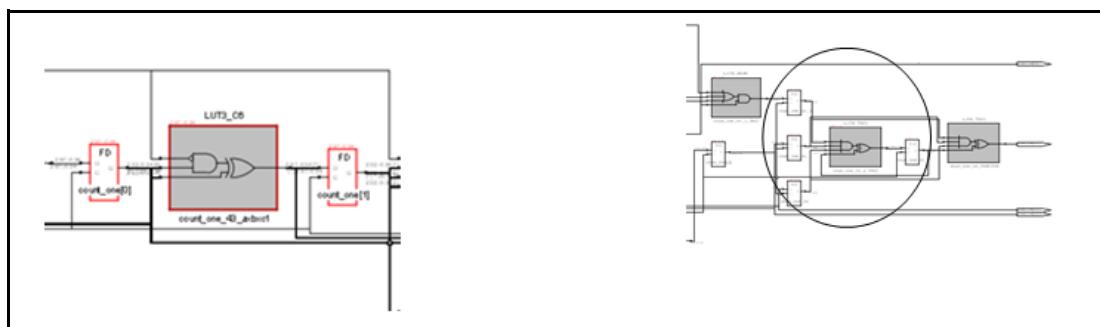


After applying syn\_allow\_retiming.

Verilog    output reg [3:0]count\_one /\* synthesis syn\_allow\_retiming=1\*/;

VHDL    attribute syn\_allow\_retiming of vout\_reg : signal is true;

The critical path and the worst slack for this scenario are shown along with the four '\_ret' retimed registers.



## **syn\_allowed\_resources**

### *Attribute*

Specifies the maximum number of technology-specific resources available for use in a design.

### **Description**

The `syn_allowed_resources` attribute allows you to specify the maximum number of available resources that can be assigned. Apply the attribute globally in the top-level design (with or without compile points) or to a compile point to specify its allowed resources.

When a compile point is synthesized, the resources of its siblings and parents cannot be taken into account because it stands alone as an independent synthesis unit. This attribute lets you account for this usage by limiting the resources a compile point can use.

If you do not set this attribute for a given compile point, the default maximum values for the region or FPGA are used which can result in exhausting all region or FPGA resources for a single compile point.

The attribute value assigned to a given compile point includes the resources used by its children (at all levels). For example, if a compile point is limited by this attribute to a maximum of four block RAMs and it contains a compile point that uses two block RAMs, there are two block RAMs remaining for the parent compile point itself.

For RAM resources, you can use multiple values to specifically define how many of each type of RAM can be used.

## syn\_allowed\_resources Syntax

|         |                                                                                                                                                                                                                                                                          |                                           |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| FDC     | <code>define_attribute {v:module   architectureName}<br/>syn_allowed_resources {spec=n [, spec=n ...] }<br/>define_global_attribute syn_allowed_resources<br/>{spec=n [, spec=n ... ] }</code>                                                                           | <a href="#">Constraint Editor Example</a> |
| Verilog | <code>object /* synthesis syn_allowed_resources = "spec=N" */ ;<br/>object must be register definition (reg) signals.</code>                                                                                                                                             | <a href="#">Verilog Example</a>           |
| VHDL    | <code>attribute syn_allowed_resources of object : objectType is<br/>"spec= N";</code><br><br><code>object</code> can be a signal that defines a compile point or a label of a component instance. For descriptions of the <code>spec</code> values, see the table below. | <a href="#">VHDL Example</a>              |

In the above syntax;

- *module or architectureName* is the name of a Verilog module or VHDL architecture that has been defined as a compile point.
- *spec* is dsps (digital signal processing blocks) or blockmults for backwards compatibility.
- *n* is an integer that specifies the maximum number of resources of the specified type.

## Constraint Editor Example

|   | Enabled                             | Object Type | Object     | Attribute             | Value        | Val Type | Description                               |
|---|-------------------------------------|-------------|------------|-----------------------|--------------|----------|-------------------------------------------|
| 1 | <input checked="" type="checkbox"/> | view        | v:work.top | syn_allowed_resources | dsps=4       | string   | Control resource usage in a compile point |
| 1 | <input checked="" type="checkbox"/> | view        | v:work.top | syn_allowed_resources | blockmults=5 | string   | Control resource usage in a compile point |
| 1 | <input checked="" type="checkbox"/> | view        | v:work.top | syn_allowed_resources | blockrams=3  | string   | Control resource usage in a compile point |

## Verilog Example

```
module top (datain, dataout, clk, addr, a, b, c);
    output[7:0] dataout;
    input clk;
    input[7:0] datain;
    input[7:0] addr;
    input [79:0] a, b;
    output reg [139:0] c
    /* synthesis syn_allowed_resources = "dsp=2 | blockmults=3 |
       dsp_blocks=4" */;

    reg [7:0] mem [255:0]
    /* synthesis syn_allowed_resources= "M20K = 1| blockrams=3|
       M-RAM=3| M512s=3" */;

    always @ (posedge clk)
        begin
            mem [addr]<=datain[7:0];
        end
    assign dataout= mem [addr];

    always @ (posedge clk)
        begin
            c[13:0] <= a[7:0] * b[7:0];
            c[27:14] <= a[15:8]* b[15:8];
            c[41:28] <= a[23:16]* b[23:16];
            c[55:42] <= a[31:24]* b[31:24];
            c[69:56] <= a[39:32]* b[39:32];
            c[83:70] <= a[47:40]* b[47:40];
            c[97:84] <= a[55:48]* b[55:48];
            c[111:98] <= a[63:56]* b[63:56];
            c[125:112] <= a[71:64]* b[71:64];
            c[139:126] <= a[79:72]* b[79:72];
        end
    endmodule
```

See [Constraint Editor Example](#), on page 501 for ways to use the attribute.

## VHDL Example

```

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

entity top_ra_mux is
    port (ADDR1: in std_logic_vector(7 downto 0);
          data_in : in std_logic_vector(7 downto 0);
          A : in std_logic_vector (79 downto 0);
          CLK : in std_logic;
          B : in std_logic_vector (79 downto 0);
          EN : in std_logic;
          C : out std_logic_vector (159 downto 0);
          data_out : out std_logic_vector(7 downto 0) );
end top_ra_mux;

architecture rtl of top_ra_mux is
type mem_type is array (255 downto 0) of
    std_logic_vector (7 downto 0);
signal mem : mem_type;
signal mult_i : std_logic_vector(159 downto 0);
attribute syn_allowed_resources : string;
attribute syn_allowed_resources of mem : signal is "blockrams=1";
attribute syn_allowed_resources of mult_i : signal is
    "dsp_blocks=3";
begin
process (CLK)
begin
    IF (CLK'event AND CLK = '1') THEN
        IF (EN = '1') THEN
            data_out <= mem(to_integer(unsigned(ADDR1)));
        END IF;
    END IF;
end process;

process (CLK)
begin
    IF (CLK'event AND CLK = '1') THEN
        IF (EN = '1') THEN
            mem(to_integer(unsigned(ADDR1))) <= data_in;
        END IF;
    END IF;
end process;

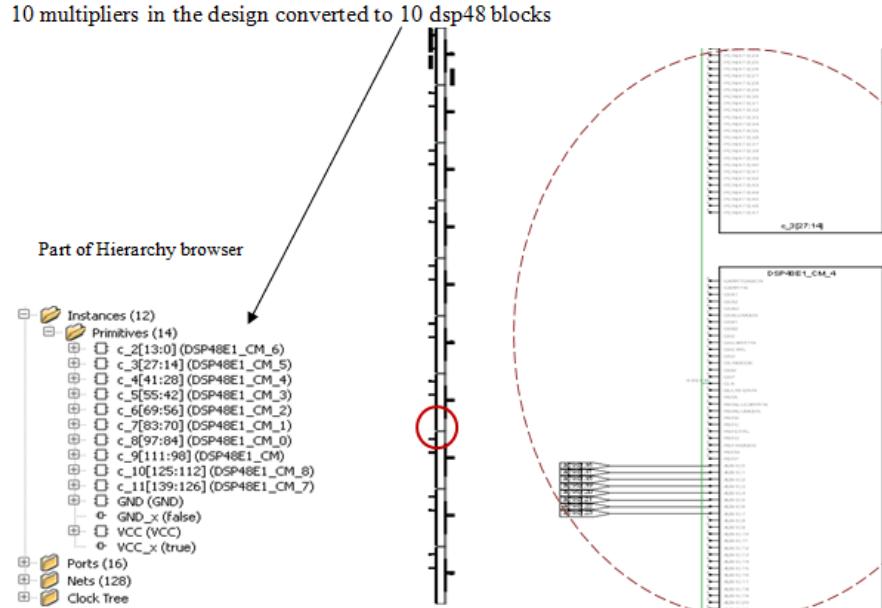
```

```
mult_i(15 downto 0) <= std_logic_vector
  (unsigned(a(7 downto 0))*unsigned(b(7 downto 0)));
mult_i(31 downto 16) <= std_logic_vector
  (unsigned(a(15 downto 8))*unsigned(b(15 downto 8)));
mult_i(47 downto 32) <= std_logic_vector
  (unsigned(a(23 downto 16))*unsigned(b(23 downto 16)));
mult_i(63 downto 48) <= std_logic_vector(
  unsigned(a(31 downto 24))*unsigned(b(31 downto 24)));
mult_i(79 downto 64) <= std_logic_vector
  (unsigned(a(39 downto 32))*unsigned(b(39 downto 32)));
mult_i(95 downto 80) <= std_logic_vector
  (unsigned(a(47 downto 40))*unsigned(b(47 downto 40)));
mult_i(111 downto 96) <= std_logic_vector(unsigned(
  a(55 downto 48))*unsigned(b(55 downto 48)));
mult_i(127 downto 112) <= std_logic_vector(
  unsigned(a(63 downto 56))*unsigned(b(63 downto 56)));
mult_i(143 downto 128) <= std_logic_vector(
  unsigned(a(71 downto 64))*unsigned(b(71 downto 64)));
mult_i(159 downto 144) <= std_logic_vector
  (unsigned(a(79 downto 72))*unsigned(b(79 downto 72)));

process (CLK)
begin
  IF (CLK'event AND CLK = '1') THEN
    C <= mult_i;
  end if;
end process;
end rtl;
```

## Effect of Using syn\_allowed\_resources

The following figure shows a Xilinx Virtex-7 device before applying the attribute, with the `dsp48` keyword:



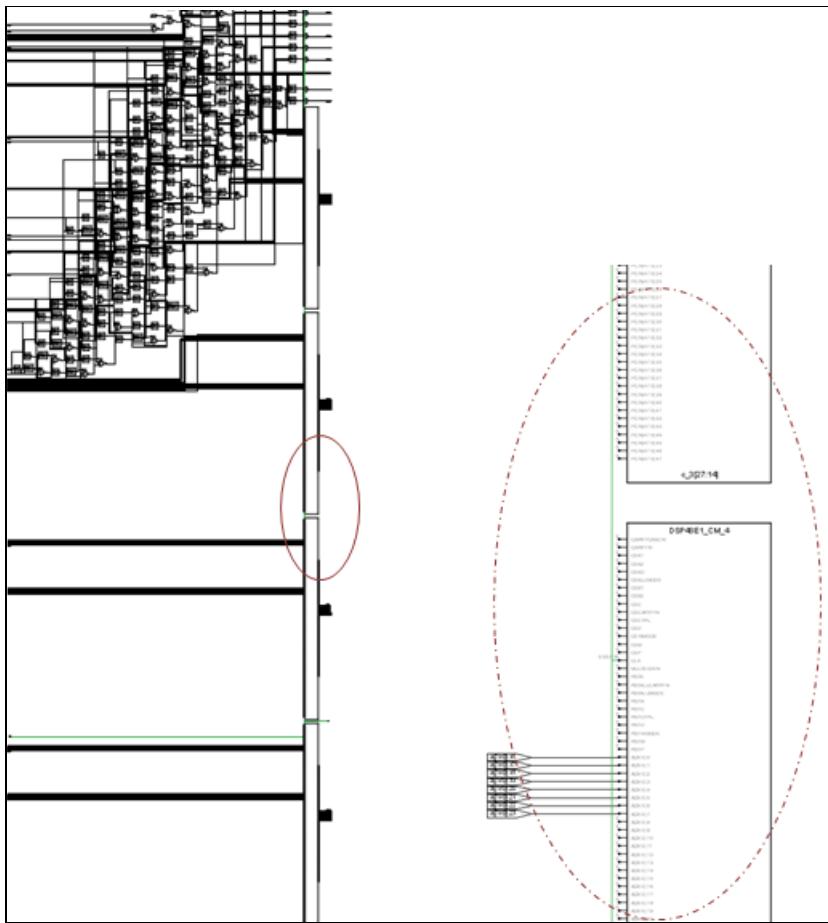
The following example shows the attribute used with the `dsps` keyword.

Verilog `output reg [139:0] c/* synthesis syn_allowed_resources = "dsps=4" */;`

VHDL `attribute syn_allowed_resources : string;  
attribute syn_allowed_resources of mult_i : signal is "dsps=4";`

FDC `define_attribute {v:work.top} syn_allowed_resources {dsps=4}`

---



---

## Examples of Resource Limitations

The following examples show resource limitations you might encounter for your design.

## Example 1: DSP Resources

This example allows the design v.work.TOP to have two DSP blocks.

```
define_attribute {v:work.TOP} syn_allowed_resources {dsps = 2}
```

For backward compatibility, DSP48 resources are treated as blockmults resources. To manually limit the use of blockmults, apply `syn_allowed_resources` on the top view. For example,

```
define_attribute {v:work.dsp_48_route_check}
    syn_allowed_resources {blockmults=1}
```

The resource limit for the `syn_allowed_resources` value includes both inferred and instantiated components. If you set DSP usage to 1 and your design infers one and instantiates one DSP48E primitive, both DSP48E primitives are honored.

```
module top (input clk,
            input [10:0] a1,
            input [10:0] a2,
            input [10:0] a3,
            input [10:0] b1,
            input [10:0] b2,
            input [10:0] b3,
            input [47:0] c1,
            input [47:0] c2,
            input [47:0] c3,
            input [6:0] opmode,
            output reg [47:0] out_1,
            output reg [47:0] out_2,
            output reg [47:0] out_3)
    /*synthesis syn_allowed_resources="dsps=1" */;

    // always @ (posedge clk)
    //     out_2 <= (a2*b2);

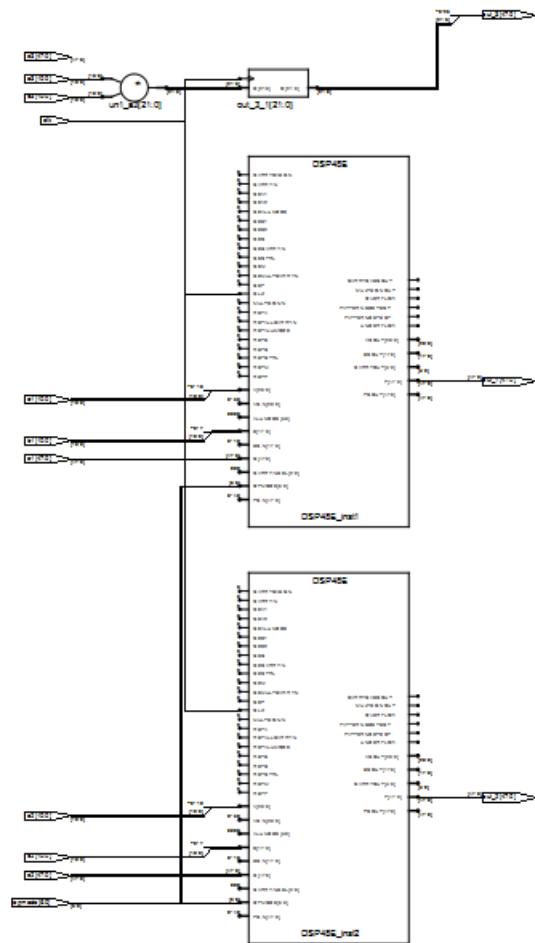
    always @ (posedge clk)
        out_3 <= (a3*b3);

    // always @ (posedge clk)
    //     out_1 <= (a1*b1);
    //

    DSP48E DSP48E_inst1 (
        .A(a1),
        .B(b1),
```

```
.C(c1),  
.P(out_1),  
.CLK(clk),  
.OPMODE(opmode)  
);  
  
DSP48E DSP48E_inst2 (  
.A(a2),  
.B(b2),  
.C(c2),  
.P(out_2),  
.CLK(clk),  
.OPMODE(opmode)  
);  
  
endmodule
```

The following schematic shows that two DSP48E primitives are honored.



If you set DSP usage to 2 and your design infers only one DSP48E primitive, then this resource limit cannot be honored.

```
module top (input clk,
    input [10:0] a1,
    input [10:0] a2,
    input [10:0] a3,
    input [10:0] b1,
    input [10:0] b2,
    input [10:0] b3,
    input [47:0] c1,
    input [47:0] c2,
    input [47:0] c3,
    input [6:0] opmode,
    output reg [47:0] out_1,
    output reg [47:0] out_2,
    output reg [47:0] out_3)
    /*synthesis syn_allowed_resources="dsps=2" */;

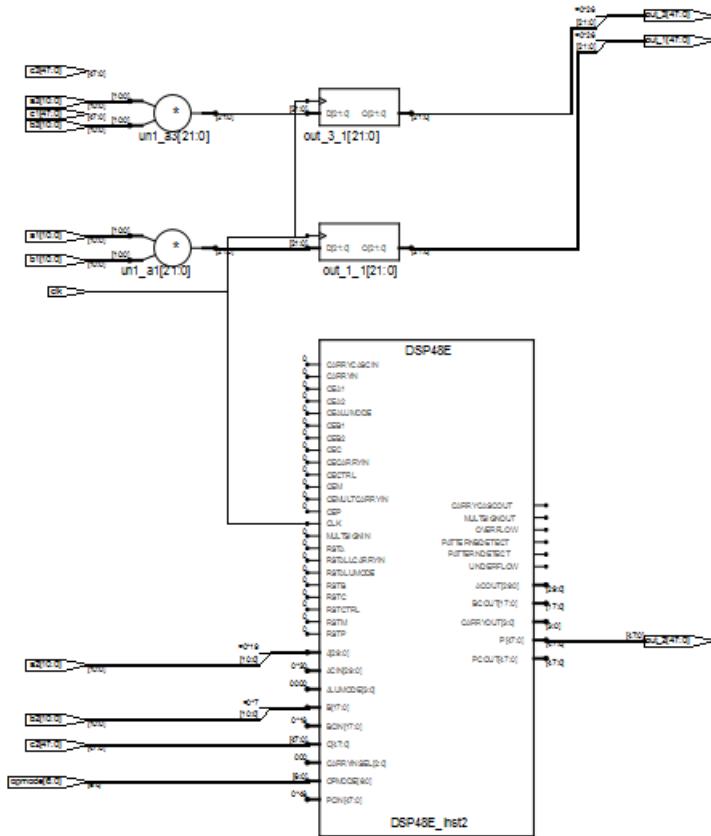
// always @ (posedge clk)
// out_2 <= (a2*b2);

always @ (posedge clk)
    out_3 <= (a3*b3);

always @ (posedge clk)
    out_1 <= (a1*b1);

// DSP48E DSP48E_inst1 (
//     .A(a1),
//     .B(b1),
//     .C(c1),
//     .P(out_1),
//     .CLK(clk), comp
//     .OPMODE(opmode)
// );
DSP48E DSP48E_inst2 (
    .A(a2),
    .B(b2),
    .C(c2),
    .P(out_2),
    .CLK(clk),
    .OPMODE(opmode)
);
endmodule
```

The following schematic shows that only one DSP48E primitive was honored.



## Example 2: Block RAM Resources

Resource management cannot always honor the resource limitations you specify. For example, the synthesis software may not be able to separate block RAM and logic assignments for 64-bit wide RAM that use multiple RAM components. Therefore, block RAM resources can become overutilized.

The following Resource Usage Report in the log file shows that the block RAM sites are overutilized.

```
I/O ports: 44
I/O primitives: 44
IBUF          22 uses
IBUFG         1 use
OBUF          21 uses

BUFG          1 use

I/O Register bits:           42
Register bits not including I/Os: 3173 (16%)
```

```
RAM/ROM usage summary
Occupied Block RAM sites (RAMB36) : 38 of 36 (105%)
```

```
Global Clock Buffers: 1 of 32 (3%)
```

```
Total load per clock:
clk: 3291
```

```
Mapping Summary:
Total LUTs: 2235 (11%)
```

## **syn\_assign\_to\_region**

### *Attribute*

Allows you to assign logic to physical regions specified on the device that is supported by the Stacked Silicon Interconnect (SSI) technology.

### **Description**

This region constraint must be provided in the FDC file generated from logic synthesis.

### **syn\_assign\_to\_region Syntax**

FDC      **define\_attribute {i:*instance*} {syn\_assign\_to\_region} {value}**      [FDC Example](#)

In the above syntax:

- *object* – The instance name for the logic to be assigned to a region. Specify as *i:instanceName*.
- *sliceLLloc:sliceURloc* – Specifies the lower-left and upper-right SLICE coordinate system locations for the region to be created on the device. Format for this constraint is *SLICE\_XlYl:SLICE\_XurYur*.

### **FDC Example**

The following example assigns instance or1200\_cpu to the region defined by the lower-left and upper-right SLICE coordinate system X12Y233 and X127Y281 for the specified architecture.

```
define_attribute {i:or1200_cpu} {syn_assign_to_region}
    {SLICE_X12Y233:SLICE_X127Y281}
```

### **Effect of using syn\_assign\_to\_region**

Applying the *syn\_assign\_to\_region* attribute to an instance through the constraints file results in the constraint being forward annotated to the place-and-route tool in the XDC file shown below:

```
#User specified region constraints
create_pblock rgn_12_233_127_281
resize_pblock [get_pblocks rgn_12_233_127_281]
    -add {SLICE_X12Y233:SLICE_X127Y281}
add_cells_to_pblock [get_pblocks rgn_12_233_127_281]
    [get_cells M2] -clear_locs
```

## syn\_assign\_to\_slr

### Attribute

Assigns logic to a specific SLR (Super Logic Region) for a device.

### Description

Xilinx place-and-route software treats the SLICE coordinate system for SSI as a monolithic device. However, there is a timing penalty for signals that cross SLR boundaries, so place and route tries to automatically partition logic to the SLR. If these results are not optimal, you can use the `syn_assign_to_slr` attribute to assign logic to a specific SLR for the device. You can use this attribute with logic synthesis flows targeting Virtex-7 SSI devices.

**Note:** No resource estimation is performed for SLR assignments. If too much logic is assigned to a specific SLR, an error will occur in the Xilinx place-and-route tool.

The attribute must be specified in a constraint file. This attribute can be applied to instances or ports in the design.

### syn\_assign\_to\_slr Syntax

```
define_attribute {object} syn_assign_to_slr {slrValue}
```

In the above syntax:

- *object* – specified for the following:
  - **i:instanceName**
  - **p:portName**
- *slrValue* – The SLR specified as 0, 1, 2, or 3 where 3 is the top-most SLR

Each SLR follows the same SLICE coordinate system for the architecture. Depending on the device selected, the number of SLRs can range between two and four.

**Note:** When SLR assignment conflicts exist, the software honors lower-level hierarchical instances within a hierarchy. For example, instance A contains instances B and C. If instance A is assigned to SLR0 and instance B is assigned to SLR1, then the assignments to SLR0 are maintained for instances A and C. However, the synthesis software still honors instance B assignment to SLR1.

## FDC Example

```
define_attribute {i:inst_abc} {syn_assign_to_slr} {0}
```

### Effect of using **syn\_assign\_to\_slr**

Applying the `syn_assign_to_slr` attribute to an instance through the constraints file results in the constraint being forward annotated to the place-and-route tool.

## **syn\_async\_reg**

### *Attribute*

The `syn_async_reg` attribute controls the minimum number of registers used to synchronize a signal across two asynchronous clock domains.

### **Description**

The `syn_async_reg` attribute allows you to set the minimum number of registers used to synchronize a signal across two asynchronous clock domains. If a launch register is in one clock domain and a chain of capture registers is in another clock domain, the synthesis tool infers that this configuration is being used to synchronize a signal between the clock domains based on clock definitions in the FDC constraint file.

When a value is set for the `syn_async_reg` attribute (default is 2), the software reserves the number of capture registers specified for synchronization by applying an `ASYNC_REG` property to these registers. This property prevents the registers from being mapped to SRL blocks and is forward annotated to the XDC file to allow the registers to be placed in the same region. Any registers in the chain beyond the specified number do not have the property applied and are free to be packed into an SRL.

To prevent registers in a synchronization chain from receiving and forward annotating the `ASYNC_REG` property, apply a `syn_srlstyle` attribute with a value of registers to the first capture register in the chain that should not have the applied property. For example, specifying the following attribute on register instance `a3[0]` allows any subsequent registers in the chain (for example, `a4[0]`, `a5[0]`, etc.) to be packed into an SRL.

```
define_attribute {i:a3[0]} {syn_srlstyle} {registers}
```

The `syn_srlstyle` attribute takes precedence over the global `syn_async_reg` attribute.

### **syn\_async\_reg Syntax**

The `syn_async_reg` attribute is a global attribute that can only be specified in the FDC constraint file.

FDC      **define\_global\_attribute {syn\_async\_reg} {numOfRegs}**      [Constraints Editor Example](#)

## Constraints Editor Example

|   | Enable                              | Object Type | Object   | Attribute     | Value | Value Type | Description |
|---|-------------------------------------|-------------|----------|---------------|-------|------------|-------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Global> | syn_async_reg | 3     |            |             |
| 2 |                                     |             |          |               |       |            |             |

### Effect of Using syn\_async\_reg

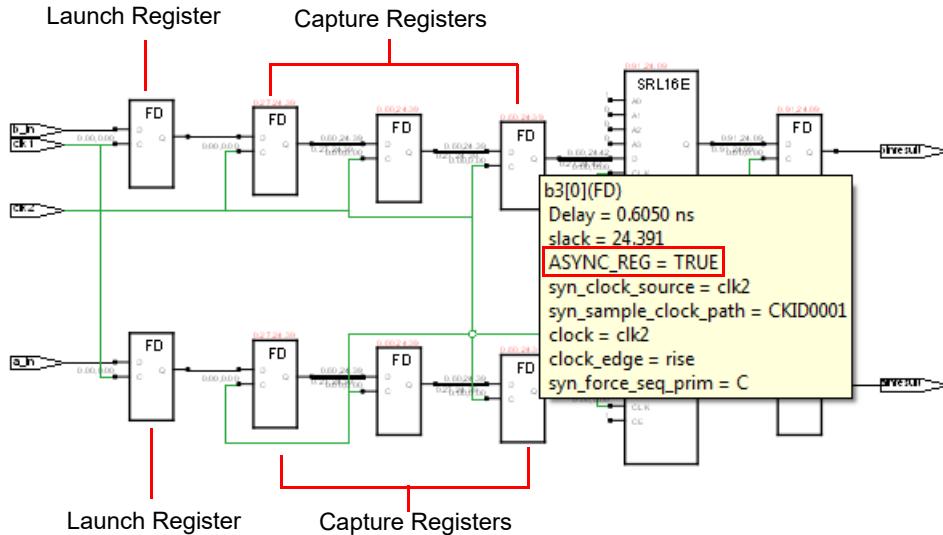
As an example, assume that the following information is included in the FDC constraint file. The clocks are specified as:

```
create_clock -name {clk1} {p:clk1} -period {20}
create_clock -name {clk2} {p:clk2} -period {25}
set_clock_groups -derive -asynchronous -name {clk1}
    -group {{c:clk1}}
set_clock_groups -derive -asynchronous -name {clk2}
    -group {{c:clk2}}
```

The `syn_async_reg` attribute used in the example is specified as:

```
define_global_attribute {syn_async_reg} {3}
```

The following view shows that three capture registers for each clock group (clk1 and clk2) have the applied ASYNC\_REG property in the capture register synchronization chain following design synthesis.



A message in the log file (FX1019) identifies capture register chains with the `ASYNC_REG` property. You can use `Edit->Find` to locate specific instances with the property or use `Tcl` find commands to manipulate collections of registers with the `ASYNC_REG` property. For example, you can use the following commands from the `Tcl` window:

- Create a collection of all FD registers clocked by `clk2` with the property `ASYN_REG==TRUE`:

```
set c1 [find * -hier -filter
    {@inst_of == FD && @clock == clk2 && @ASYNC_REG == TRUE}]
```

For the example, the collection includes `{i:a1[0]}` `{i:a2[0]}` `{i:b1[0]}` `{i:b2[0]}`.

- Create a collection of all registers clocked by `clk1` from net `b_in`:

```
set c2 [find * -in [c_list [expand -through {n:b_in}]]
    -filter @clock == clk1]
```

For the example, the collection includes `{i:b_inreg}`.

- Create a collection of all registers on clock `clk2` from the previous `c2` collection with the `ASYN_REG==TRUE` property:

```
set c3 [find * -in [c_list [expand -from $c2]]
    -filter @clock == clk2 && @ASYNC_REG == TRUE]
```

## **syn\_auto\_insert\_bufg**

### *Attribute*

Automatically inserts a BUFG primitive on high-fanout control signal nets such as set, reset, enable, or clear).

### **Description**

The `syn_auto_insert_bufg` attribute is globally enabled by default. It automatically inserts BUFG primitives on high-fanout nets that are non-critical or less critical, such as set, reset, enable, or clear.

To disable automatic BUFG insertion, apply this attribute locally on nets in the design. Alternatively, disable this attribute globally, and then use the `syn_insert_buffer` attribute to apply specific types of buffers on nets in the design. See [Inserting BUFG Global Buffers, on page 300](#) for details on using this attribute, and criteria for automatic BUFG insertion.

### **syn\_auto\_insert\_bufg Syntax**

Apply the `syn_auto_insert_bufg` attribute globally or on individual nets of the design in the FDC constraint file. You can only specify this attribute in an FDC constraint file.

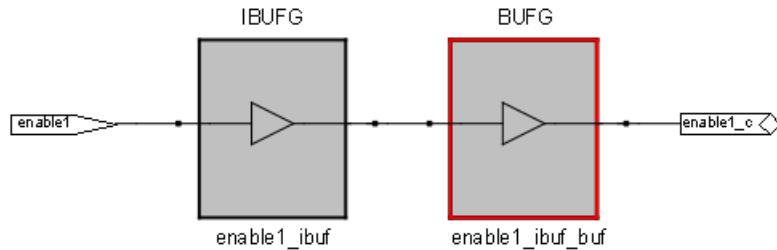
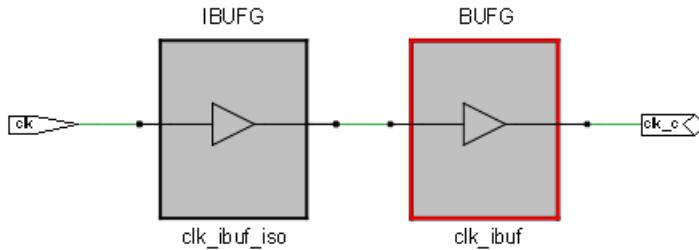
|     |                                                                  |                                            |
|-----|------------------------------------------------------------------|--------------------------------------------|
| FDC | <code>define_global_attribute syn_auto_insert_bufg {1 0}</code>  | <a href="#">Constraints Editor Example</a> |
|     | <code>define_attribute syn_auto_insert_bufg netName {1 0}</code> |                                            |

### **FDC Example**

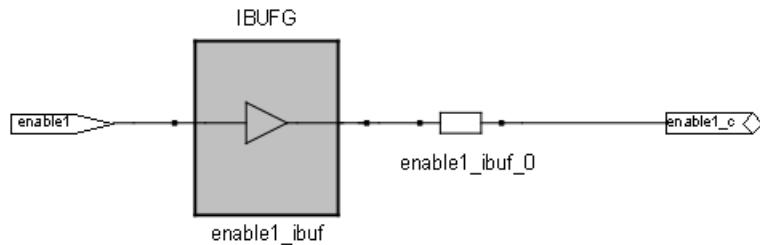
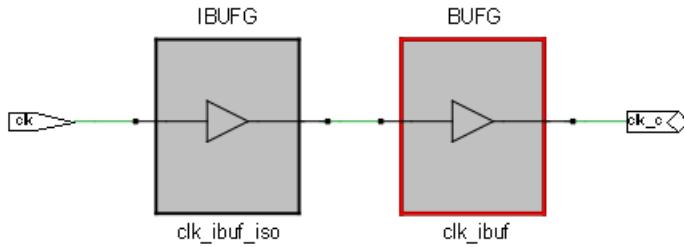
|   | Enable                              | Object Type | Object   | Attribute                         | Value | Value Type | Description | Comment |
|---|-------------------------------------|-------------|----------|-----------------------------------|-------|------------|-------------|---------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Global> | <code>syn_auto_insert_bufg</code> | 1     |            |             |         |
| 2 |                                     |             |          |                                   |       |            |             |         |

### **Effect of Using syn\_auto\_insert\_bufg**

This example shows the results when `syn_auto_insert_bufg` is set to 1 globally for this design:



Here are the results with `syn_auto_insert_bufg` set to 0 globally for this design:



## **syn\_auto\_insert\_bufgmux**

### *Attribute*

Controls global automatic BUFGMUX insertion. If inserted, you can specify synchronous (SYNC) or asynchronous (ASYNC) clocks for the BUFGMUX primitives.

| <b>Vendor</b> | <b>Technology</b> |
|---------------|-------------------|
|---------------|-------------------|

|        |     |
|--------|-----|
| Xilinx | All |
|--------|-----|

### **syn\_auto\_insert\_bufgmux Value**

| <b>Default</b> | <b>Global</b> |
|----------------|---------------|
|----------------|---------------|

|       |     |
|-------|-----|
| ASYNC | Yes |
|-------|-----|

| <b>Value</b> | <b>Description</b> |
|--------------|--------------------|
|--------------|--------------------|

|       |                                      |
|-------|--------------------------------------|
| false | Disables automatic BUFGMUX insertion |
|-------|--------------------------------------|

|       |                                                                  |
|-------|------------------------------------------------------------------|
| ASYNC | Forward annotates the BUFGMUX property as CLK_SEL_TYPE = "ASYNC" |
|-------|------------------------------------------------------------------|

|      |                                                                 |
|------|-----------------------------------------------------------------|
| SYNC | Forward annotates the BUFGMUX property as CLK_SEL_TYPE = "SYNC" |
|------|-----------------------------------------------------------------|

The BUFGMUX will not be inferred when there are more than two unique clocks (for example, three clocks are feeding the MUX).

### **Description**

Use the syn\_auto\_insert\_bufgmux attribute to either globally disable automatic BUFGMUX insertion (false) or insert BUFGMUX primitives for synchronous (SYNC) or asynchronous (ASYNC) clocks. ASYNC is the default setting. When two unique clocks feed the MUX, you can select either BUFGMUX primitives with synchronous (CLK\_SEL\_TYPE="SYNC") or asynchronous (CLK\_SEL\_TYPE="ASYNC") clocks for insertion. This BUFGMUX clock property is forward-annotated to the place-and-route tool.

### **syn\_auto\_insert\_bufgmux Syntax**

Apply the syn\_auto\_insert\_bufgmux attribute globally in the FDC constraint file.

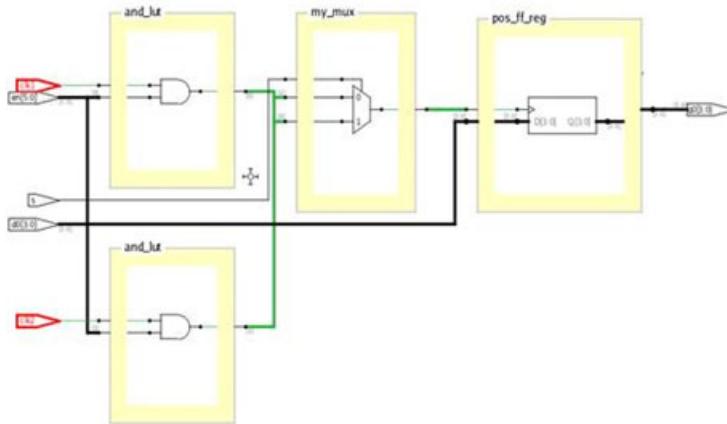
FDC            `define_global_attribute syn_auto_insert_bufgmux {false|ASYNC|SYNC}`

The following example disables auto insertion of the BUFGMUX primitive.

```
define_global_attribute {syn_auto_insert_bufgmux} {false}
```

### Effect of Using `syn_auto_insert_bufgmux`

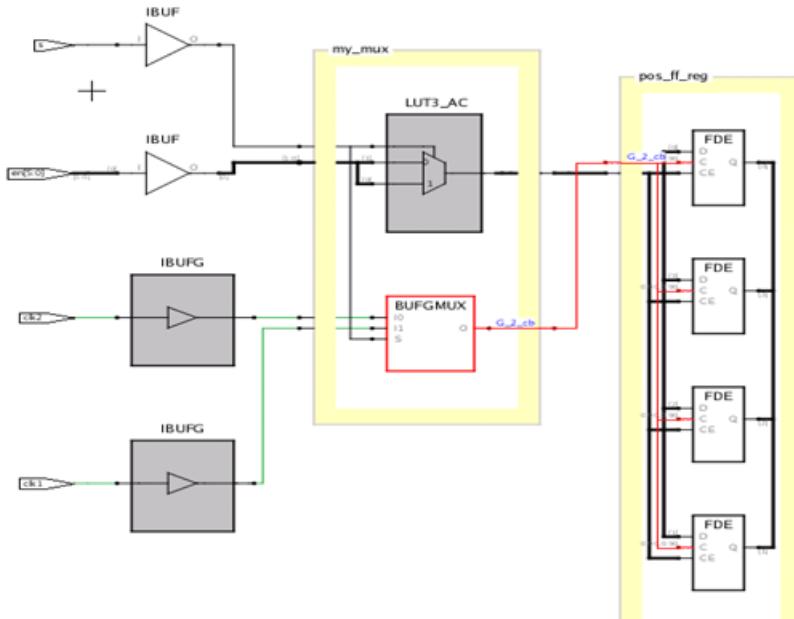
This example has two clocks defined on the top-level ports `clk1` and `clk2`. These clocks are gated with enable signals (`en[1]` and `en[0]`) driving the MUX inputs. This condition allows for BUFGMUX inference.



You can set the attribute for inference for the design:

```
define_global_attribute {syn_auto_insert_bufgmux} {SYNC}
```

The BUFGMUX is then inferred for the clock MUX after you run synthesis (see the following figure). The enable signals are multiplexed and fed into the clock enable pins of the register loads. You can verify the `CLK_SEL_TYPE="SYNC"` property for the BUFGMUX primitive in the EDIF file generated.



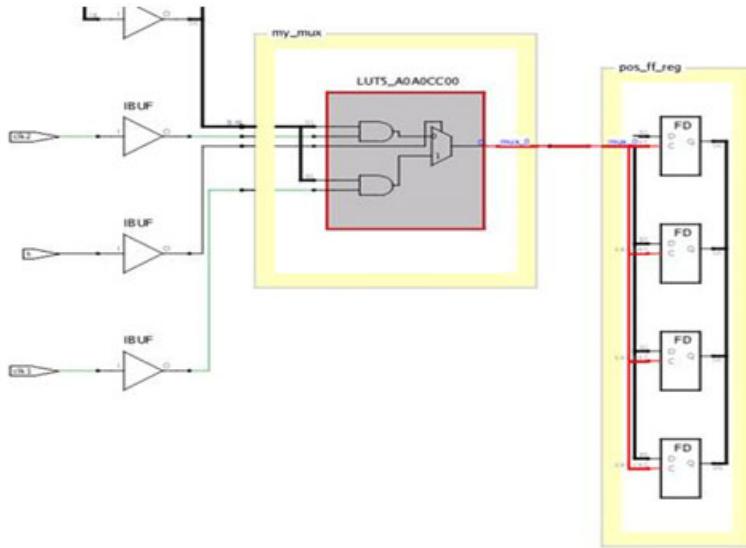
An example of the clock properties forward-annotated to the place-and-route tool is shown in the EDIF and simulation netlists below.

```
(contents
  (instance q_cb (viewRef PRIM (cellRef BUFGMUX (libraryRef VIRTEX)))
    (property CLK_SEL_TYPE (string "SYNC"))
  )

  BUFGMUX q_cb_c (
    .O(q_cb),
    .I0(clk0_c),
    .I1(clk1_c),
    .S(s_c)
  );
  defparam q_cb_c.CLK_SEL_TYPE = "SYNC";
  
```

For the same design, the MUX is implemented as a simple LUT if you specify that BUFGMUXes should not be inferred:

```
define_global_attribute {syn_auto_insert_bufgmulx} {false}
```



## **syn\_black\_box**

### *Directive*

Defines a module or component as a black box.

### **Description**

Specifies that a module or component is a black box for synthesis. A black box module has only its interface defined for synthesis; its contents are not accessible and cannot be optimized during synthesis. A module can be a black box whether or not it is empty.

Typically, you set `syn_black_box` on objects like the ones listed below. You do not need to define a black box for such an object if the tool includes a predefined black box for it.

- Vendor primitives and macros (including I/Os).
- User-designed macros whose functionality is defined in a schematic editor, IP, or another input source where the place-and-route tool merges design netlists from different sources.

For more information about using this attribute to define black boxes, see [Defining Black Boxes for Synthesis, on page 257](#).

### **Other Black Box Source Code Directives**

Once you define a black box with `syn_black_box`, use other source code directives to define timing and other information for the black box. You must add the directives to the source code because the timing models are specific to individual instances. There are no corresponding Tcl directives you can add to a constraint file.

#### **Black Box Timing Directives**

|                               |                                                                           |
|-------------------------------|---------------------------------------------------------------------------|
| <code>syn_isclock</code>      | Specifies a clock port on a black box.                                    |
| <code>syn_tpd&lt;n&gt;</code> | Sets timing propagation for combinational delay through the black box.    |
| <code>syn_tsu&lt;n&gt;</code> | Defines timing setup delay required for input pins relative to the clock. |

|                               |                                                                 |
|-------------------------------|-----------------------------------------------------------------|
| <code>syn_tco&lt;n&gt;</code> | Defines the timing clock to output delay through the black box. |
|-------------------------------|-----------------------------------------------------------------|

### Black Box Directives for Gated Clocks

|                                             |                                                                                                           |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| <code>syn_force_seq_prim</code>             | Indicates that gated clocks should be fixed for this black box.                                           |
| <code>syn_gatedclk_clock_en</code>          | Specifies the enable pin to be used in fixing the gated clocks.                                           |
| <code>syn_gatedclk_clock_en_polarity</code> | Indicates the polarity of the clock enable port on a black box so that the software can fix gated clocks. |

### Black Box Pin Definitions

|                                 |                                                                      |
|---------------------------------|----------------------------------------------------------------------|
| <code>black_box_pad_pin</code>  | Indicates that a black box is an I/O pad for the rest of the design. |
| <code>black_box_tri_pins</code> | Indicates tristates on black boxes.                                  |

## syn\_black\_box Syntax

|          |                                                                       |                                 |
|----------|-----------------------------------------------------------------------|---------------------------------|
| CDC File | <code>define_directive {object} {syn_black_box} {1}</code>            | <a href="#">CDC Example</a>     |
| Verilog  | <code>object /* synthesis syn_black_box */;</code>                    | <a href="#">Verilog Example</a> |
| VHDL     | <code>attribute syn_black_box of object : objectType is true ;</code> | <a href="#">VHDL Example</a>    |

### CDC Example

```
define_directive {v:MyLib.sub} {syn_black_box} {1}
```

## Verilog Example

```
module top (clk, in1, in2, out1, out2);
    input clk;
    input [1:0]in1;
    input [1:0]in2;
    output [1:0]out1;
    output [1:0]out2;
    add U1 (clk, in1, in2, out1);
    black_box_add U2 (in1, in2, out2);
endmodule

module add (clk, in1, in2, out1);
    input clk;
    input [1:0]in1;
    input [1:0]in2;
    output [1:0]out1;
    reg [1:0]out1;

    always@(posedge clk)
        begin
            out1 <= in1 + in2;
        end
    endmodule

module black_box_add(A, B, C)/* synthesis syn_black_box */;
    input [1:0]A;
    input [1:0]B;
    output [1:0]C;
    assign C = A + B;
endmodule
```

## VHDL Example

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

entity add is
    port(
        in1 : in std_logic_vector(1 downto 0);
        in2 : in std_logic_vector(1 downto 0);
        clk : in std_logic;
        out1 : out std_logic_vector(1 downto 0) );
end;

architecture rtl of add is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            out1 <= (in1 + in2);
        end if;
    end process;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity black_box_add is
    port (A : in std_logic_vector(1 downto 0);
          B : in std_logic_vector(1 downto 0);
          C : out std_logic_vector(1 downto 0) );
end;

architecture rtl of black_box_add is
attribute syn_black_box : boolean;
attribute syn_black_box of rtl: architecture is true;
begin
    C <= A + B;
end;

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

```
entity top is
    port (in1 : in std_logic_vector(1 downto 0);
          in2 : in std_logic_vector(1 downto 0);
          clk : in std_logic;
          out1 : out std_logic_vector(1 downto 0);
          out2 : out std_logic_vector(1 downto 0) );
end;

architecture rtl of top is
component add is
    port (in1 : in std_logic_vector(1 downto 0);
          in2 : in std_logic_vector(1 downto 0);
          clk : in std_logic;
          out1 : out std_logic_vector(1 downto 0) );
end component;

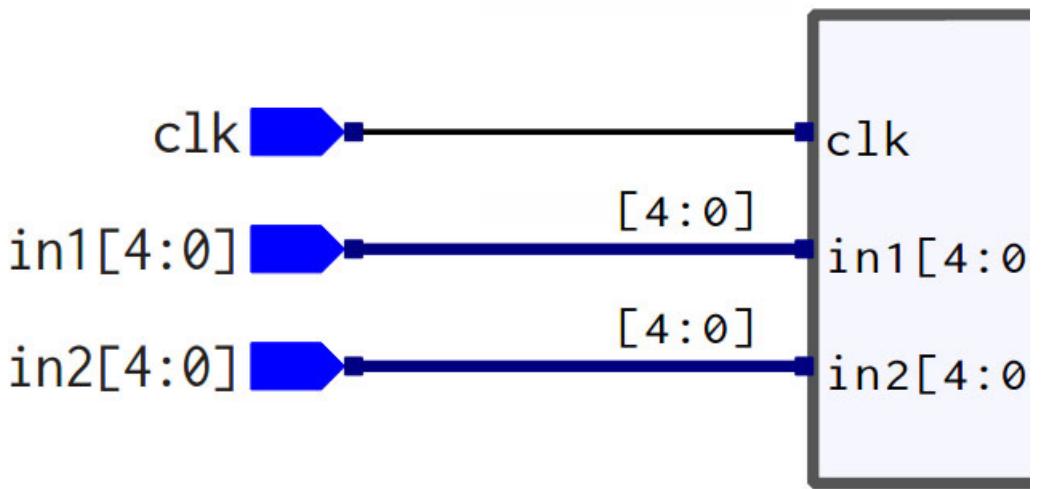
component black_box_add
    port (A : in std_logic_vector(1 downto 0);
          B : in std_logic_vector(1 downto 0);
          C : out std_logic_vector(1 downto 0) );
end component;

begin
    U1: add port map(in1, in2, clk, out1);
    U2: black_box_add port map(in1, in2, out2);
end;
```

## Effect of Using `syn_black_box`

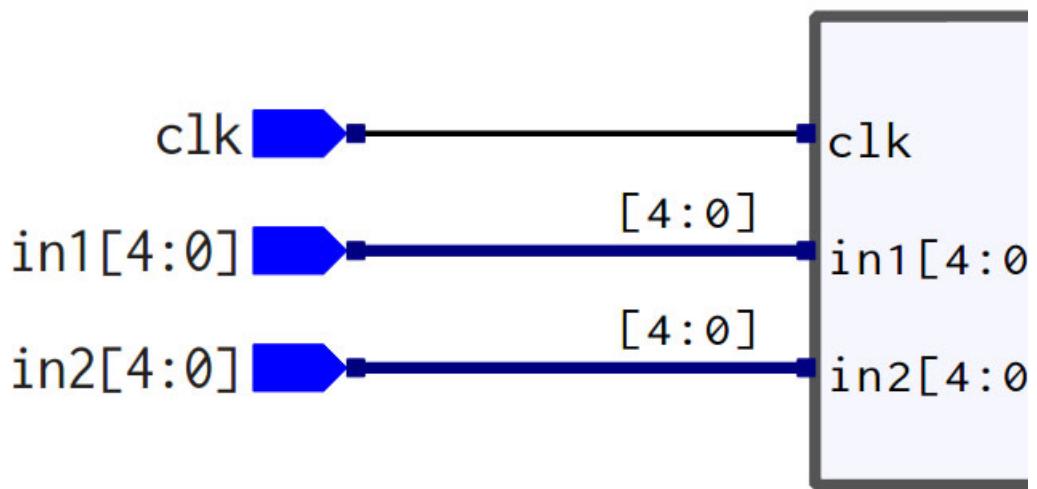
When the `syn_black_box` attribute is not set on the `black_box_add` module, its content are accessible, as shown in the example below:

```
module black_box_add (input [1:0]A, [1:0]B, output [1:0]C);
```



After applying `syn_black_box`, the contents of the black box are no longer visible:

```
module black_box_add (input [1:0]A, [1:0]B,  
                      output [1:0]C) /* synthesis syn_black_box */;
```



## **syn\_bram\_cascade\_height**

### *Attribute*

Controls Block RAM cascading.

| <b>Vendor</b> | <b>Devices</b>                                                                       |
|---------------|--------------------------------------------------------------------------------------|
| Xilinx        | Kintex UltraScale, UltraScale+, Virtex UltraScale, UltraScale+, and Zynq UltraScale+ |

## **syn\_bram\_cascade\_height Values**

| <b>Value</b> | <b>Description</b>                   |
|--------------|--------------------------------------|
| 1 - 8        | The number of Block RAMs to cascade. |

## **Description**

The `syn_bram_cascade_height` attribute sets the number of Block RAMs to cascade.

For information on Block RAM cascading, see [Cascading Block RAM, on page 216](#).

## **syn\_bram\_cascade\_height Syntax**

|         |                                                                                                                         |                                 |
|---------|-------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| FDC     | <code>define_attribute {mem[3:0]} {syn_bram_cascade_height}<br/>{Value}</code>                                          | FDC                             |
| Verilog | <code>reg [data_width-1:0] mem [2**addr_width-1:0] /* synthesis<br/>syn_bram_cascade_height = Value */;</code>          | <a href="#">Verilog Example</a> |
| VHDL    | <code>attribute syn_bram_cascade_height: integer;<br/>attribute syn_bram_cascade_height of mem: signal is Value;</code> | <a href="#">VHDL Example</a>    |

## **Verilog Example**

```
module test (addra,addrb,clka,clkb,wea,web,din,out);
parameter data_width = 4;
parameter addr_width = 18;
```

```

input [addr_width-1:0] addra, addrb;
input    clka, clkb, wea, web;
input [data_width-1:0] din;
output reg [data_width-1:0]  out;

reg [data_width-1:0] mem [2**addr_width-1:0] /* synthesis
syn_bram_cascade_height = 8 */;

// ram code

always@ (posedge clka)
begin
    if (wea)
        mem[addra] <= din;
end

always@ (posedge clkb)
begin
    if (web)
        out <= mem[addrb];
    else
        out <= mem[addrb];
end
endmodule

```

## VHDL Example

```

Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Dual_Port_ReadFirst is
generic (data_width: integer :=4;
address_width: integer :=18);

port (write_enable: in std_logic;
      write_clk, read_clk: in std_logic;
      data_in: in std_logic_vector (data_width-1 downto 0);
      data_out: out std_logic_vector (data_width-1 downto 0);
      write_address: in std_logic_vector (address_width-1 downto 0);
      read_address: in std_logic_vector (address_width-1 downto 0)
      );
end Dual_Port_ReadFirst;

```

```
architecture behavioral of Dual_Port_ReadFirst is
type memory is array (2**address_width-1) downto 0) of
std_logic_vector (data_width-1 downto 0);
signal mem: memory;

signal reg_write_address: std_logic_vector (address_width-1 downto
0);
signal reg_write_enable: std_logic;

attribute syn_bram_cascade_height: integer;
attribute syn_bram_cascade_height of mem: signal is 8;

begin
    register_enable_and_write_address:
    process (write_clk,write_enable,write_address,data_in)
begin
    if (rising_edge(write_clk)) then
        reg_write_address <= write_address;
        reg_write_enable <= write_enable;
    end if;
end process;

write:
process (read_clk,write_enable,write_address,data_in)
begin
    if (rising_edge(write_clk)) then
        if (write_enable='1') then
            mem(conv_integer(write_address)) <= data_in;
        end if;
    end if;
end process;

read:
process (read_clk,write_enable,read_address,write_address)
begin
    if (rising_edge(read_clk)) then
        data_out <= mem(conv_integer(read_address));
    end if;
end process;

end behavioral;
```

## syn\_clean\_reset

*Attribute*

| Vendor | Technologies                 |
|--------|------------------------------|
| Xilinx | UltraScale and older devices |

### syn\_clean\_reset Values

| Value                  | Description                                   | Object                | Default | Global |
|------------------------|-----------------------------------------------|-----------------------|---------|--------|
| <i>resetLogicValue</i> | Specifies the synchronous reset logic values. | Module / Architecture | None    | Yes    |

### Description

The `syn_clean_reset` attribute changes asynchronous reset registers, which cannot go into DSP48 blocks, into synchronous reset logic. The resulting implementation is not logically equivalent to the one specified in the RTL code and does not match the RTL and gate-level simulation.

**Note:** This attribute can only be applied on registers that get packed into the DSP48 blocks. This attribute does not work for all registers in the design.

### syn\_clean\_reset Syntax

|         |                                                                                                                                 |                                 |
|---------|---------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| FDC     | <code>define_global_attribute syn_clean_reset<br/>{resetLogicValue}</code>                                                      | <a href="#">FDC Example</a>     |
| Verilog | <code>object /* syn_clean_reset = "resetLogicvalue" */;</code>                                                                  | <a href="#">Verilog Example</a> |
| VHDL    | <code>attribute syn_clean_reset : string;<br/>attribute syn_clean_reset of object : objectType is<br/>"resetLogicvalue";</code> | <a href="#">VHDL Example</a>    |

### FDC Example

| Enable                              | Object Type | Object   | Attribute       | Value                      | Value Type | Description                                                                        |
|-------------------------------------|-------------|----------|-----------------|----------------------------|------------|------------------------------------------------------------------------------------|
| <input checked="" type="checkbox"/> | <any>       | <Global> | syn_clean_reset | dsp48_no_simulate,one_flop | string     | allows synplify to convert async resets to sync resets so registers go into DSP48s |

The following is an example of the syntax:

```
define_global_attribute syn_clean_reset
  {dsp48_no_simulate,one_flop}
```

## Verilog Example

```
object /* synthesis syn_clean_reset = "dsp48_no_simulate,one_flop" */;
```

With the following Verilog code:

```
module syn_clean_RST( d1,d2,rst,clk,out);
  input [7:0] d1,d2;
  input rst;
  input clk;
  output reg [14:0]out;
  wire [14:0] mul_out;

  assign mul_out = d1 * d2;

  always@(posedge clk or posedge rst)
  begin
    if (rst)
      out <= 15'b0;
    else
      out<= mul_out;
  end
endmodule
```

## VHDL Example

```
attribute syn_clean_reset of object : objectType is "dsp48_no_simulate,one_flop";
```

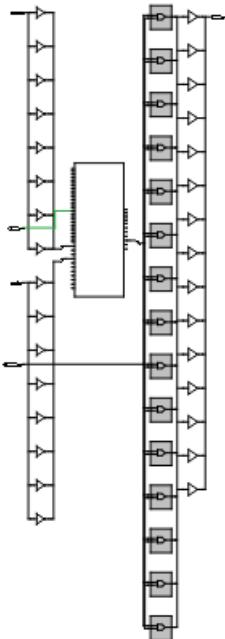
In the VHDL code (apply to top-level architecture):

```
entity test is
  port (in1_p, in1_n : in std_logic;
        clk : in std_logic;
        out1 : out std_logic);
  attribute syn_clean_reset : string;
  attribute syn_clean_reset of test : architecture is
    "dsp48_no_simulate,one_flop";
end test;
```

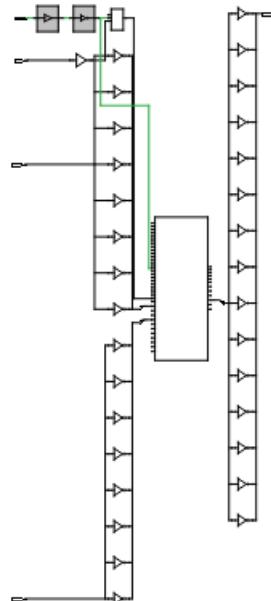
## Effect of Using syn\_clean\_reset

The `syn_clean_reset` attribute can be specified to convert asynchronous reset to synchronous reset allowing registers to be packed into a DSP block. The mapped schematic views are shown with and without the `syn_clean_reset` value.

Without `syn_clean_reset`



With `syn_clean_reset`



## **syn\_clock\_gmux\_proxy**

### *Attribute*

Allows the software to automatically generate proxy clocks at the output of the BUFGMUX.

### **Description**

This attribute, when enabled, allows the tool to automatically generate proxy clocks at the output of the BUFGMUX to provide a one-to-one correspondence for the clocks on the inputs. The proxy clocks exist at the output of the BUFGMUX and are grouped asynchronously to each other, while inheriting any other clock grouping specifications from their source/master clock.

You can use the `set_case_analysis` constraint to specify which clocks to propagate through the BUFGMUX. However, the disadvantage of using this constraint is that

- You must identify the BUFGMUX and set its value.
- Optimizations can only be done for one clock.

For details, see the `set_case_analysis` constraint in the *Reference Manual*. For these reasons, using the `syn_clock_gmux_proxy` attribute should be used whenever possible.

Note the following conditions:

- If the clock at the output of the BUFGMUX is already specified, then automatic inferencing of generated clocks does not occur for this BUFGMUX instance.
- If the `set_case_analysis` constraint has been applied on a select pin of a BUFGMUX, then automatic inferencing of generated clocks does not happen.
- The appropriate `create_generated_clock` and `set_clock_groups` constraints are forward-annotated to the Vivado XDC file.

## syn\_clock\_gmux\_proxy Syntax

|          |                                                                                                                                          |                                 |
|----------|------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| FDC file | <code>define_attribute object {syn_clock_gmux_proxy} {0 1}</code><br><code>define_global_attribute {syn_clock_gmux_proxy} {0 1}</code>   | <a href="#">FDC Example</a>     |
| Verilog  | <code>object /* synthesis syn_clock_gmux_proxy = 0 1 */;</code>                                                                          | <a href="#">Verilog Example</a> |
| VHDL     | <code>attribute syn_clock_gmux_proxy : boolean;</code><br><code>attribute syn_clock_gmux_proxy object : objectType is true false;</code> | <a href="#">VHDL Example</a>    |

## FDC Example

|   | Enable                              | Object Type | Object           | Attribute            | Value | Value Type | Description | Comment |
|---|-------------------------------------|-------------|------------------|----------------------|-------|------------|-------------|---------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Global>         | syn_clock_gmux_proxy | 1     |            |             |         |
| 2 | <input checked="" type="checkbox"/> | <any>       | i: BUFGMUX_INST1 | syn_clock_gmux_proxy | 1     |            |             |         |

It is recommended that you apply this attribute on specific BUFGMUX instances to minimize its effect on runtime. However, when you apply this attribute globally and disable a specific instance, the software can handle this situation successfully. For example:

```
define_global_attribute {syn_clock_gmux_proxy} {1}
define_attribute {i:BUFGMUX_INST1} {syn_clock_gmux_proxy} {1}
```

## Verilog Example

Here is a code segment that shows how to specify this attribute in Verilog:

```
BUFGMUX BUFGMUX_hier(.S(S),.I0(I0),.I1(I1),.O(O))
/*synthesis syn_clock_gmux_proxy=1 */;
```

## VHDL Example

Here is a code segment that shows how to specify this attribute in VHDL:

```
entity bufmux_vhd is
  port (OCLK : out std_logic;
        I0 : in std_logic;
        I1 : in std_logic;
        S : in std_logic );
end bufmux_vhd;
```

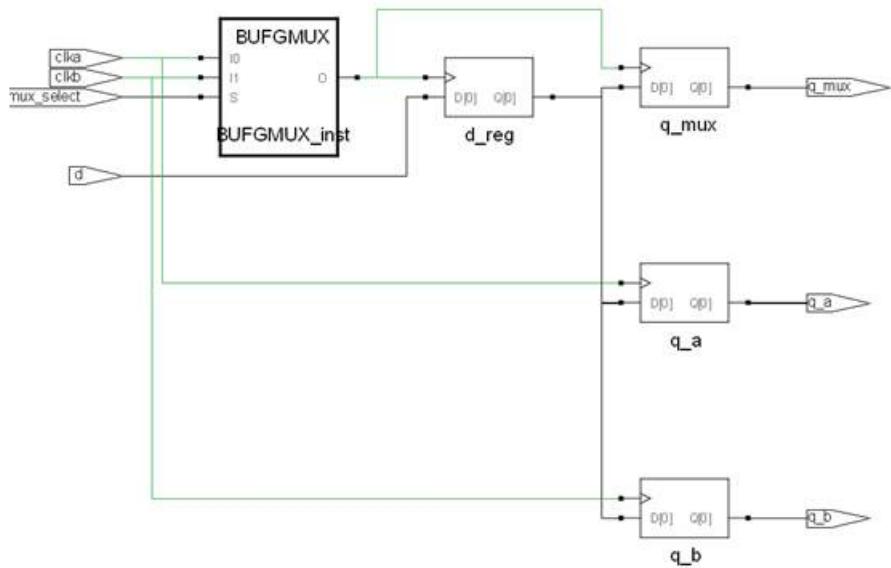
```
architecture struct of bufgmux_vhd is
component BUFGMUX
    port (O : out std_logic;
          I0 : in std_logic;
          I1 : in std_logic;
          S : in std_logic );
end component;

attribute syn_clock_gmux_proxy : boolean;
attribute syn_clock_gmux_proxy of U1 : label is true;

begin
    U1 : BUFGMUX
        port map (O  => OCLK,
                  I0 => I0,
                  I1 => I1,
                  S  => S );
end struct;
```

## Effect of Using `syn_clock_gmux_proxy`

For the circuit below, the synthesis tool propagates both `clka` and `clkb` through the BUFGMUX during timing analysis. However, since timing analysis does not automatically treat the clock signals as mutually exclusive on the output of the BUFGMUX, paths between `clka` and `clkb` after the mux are timed even though both clocks are not actually propagating simultaneously through the mux in the real circuit. Therefore, false paths will be timed after the mux unless you explicitly constrain `clka` and `clkb` in the FDC file to be a false path or asynchronous to each other. Using this attribute allows `clka` and `clkb` to remain synchronous for paths before the mux, but their proxy clocks (after the mux) are treated as asynchronous. When you use this attribute, more clocks may be created in your design and the timing reports, but can also provide better area and/or timing quality of results (QoR).



## **syn\_clock\_priority**

### *Attribute*

Lets you set a clock priority to resolve logical or physical clock conflicts.

### **Description**

The `syn_clock_priority` attribute lets you set a clock priority to resolve logical or physical clock conflicts to ensure that the clocks are forward-annotated correctly in the UCF file. In the UCF file, a TIMESPEC statement with a priority value overrides a competing TIMESPEC without a priority value. For competing TIMESPECs where both have priority values, the one with the lowest positive integer value takes priority. If the priority values are the same, no priority is established between them.

Use the `syn_clock_priority` attribute to ensure that the correct clock is forward-annotated. For example, you can use it in the following cases:

- Overriding DCM clocks with declared clocks is not honored by the place-and-route tool. Instead of manually editing the UCF file, you can use the `syn_clock_priority` attribute to assign a priority to a particular clock.
- Synthesis allows multiple clocks to propagate along a single net, either through unate logic or through a mux for timing all possible clocks and clock combinations. The place-and-route tool cannot do this, and uses the `PRIORITY` keyword in UCF to indicate clock priority in case of conflict. You can set clock priority with this attribute.

The tool forward-annotates the clock priority to the UCF file in a TIMESPEC or PERIOD statement. See [Effect of syn\\_clock\\_priority, on page 545](#) for details.

The synthesis tools handle derived and declared clocks slightly differently when they forward-annotate the priority:

#### **Attribute set on    Synthesis Forward-annotation**

|                |                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Declared clock | Directly forward-annotated to the TIMESPEC/PERIOD for the clock.                                                                        |
| Derived clock  | First generates a TIMESPEC/PERIOD that represents the nature of the derived clock, and then forward-annotates this as a declared clock. |

## Messages and Warnings

The software generates messages in certain situations:

| Situation                                                                                                     | Message                                                                           |
|---------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| When derived clocks are overridden                                                                            | Message to set clock priority with the <code>syn_clock_priority</code> attribute. |
| When the tool detects multiple timing for a path                                                              |                                                                                   |
| When the priority set on the DCM base clock is higher than the priority of the override clock on a DCM output | Usage warnings.                                                                   |
| When you assign the same priority to both inputs of a mux                                                     |                                                                                   |

## Syntax Specification

FDC file   **define\_attribute {n:netName|p:portName|i:BUFGname} {syn\_clock\_priority} {value}**

You can only set this attribute in the constraint file. You set the attribute on ports or nets, and the tool applies the value to the clock. Clocks can be declared or derived clocks. You can also set it on a BUFG instance, but not on any other instances.

The `syn_clock_priority` value must be greater than or equal to 1, with 1 being the highest priority. Consider the following clock declaration:

```
define_clock {n:u_fx_clkrstgen.clk_100_dcm} -name {clk_100}
             -period 10 -clockgroup sys_group -rise 0 -fall 5
```

If you apply `syn_clock_priority` as follows, the clock on the specified net is assigned the highest priority, 1:

```
define_attribute {n:u_fx_clkrstgen.clk_100_dcm}
                 {syn_clock_priority} {1}
```

The following example sets the highest priority for the DCM CLKFX output:

```
define_attribute {n:dcm_module_b.clk0fx} {syn_clock_priority} {1}
```

## Constraints Editor Example

|   | Enabled                             | Object Type | Object             | Attribute          | Value | Val Type | Description                                                               |
|---|-------------------------------------|-------------|--------------------|--------------------|-------|----------|---------------------------------------------------------------------------|
| 1 | <input checked="" type="checkbox"/> | net         | n:inst1.CLK180_BUF | syn_clock_priority | 1     | integer  | Establishes clock priority to be forward annotated to the Xilinx UCF file |

```
define_attribute {n:inst1.CLK180_BUF} {syn_clock_priority} {1}
```

### Effect of syn\_clock\_priority

Before applying the attribute, content in UCF file:

```
# Constraints generated by Synplify Pro maprc, Build 1020R
# Product Version "F-2012.03-SP1"
#
# Period Constraints
#Begin clock constraints

# 1001 : define_clock {clk} -name {clk} -freq {200} -clockgroup {default_clkgroup_0}
# c:\xilinx\syn_clock_priority.sdc
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 5.000 ns HIGH 50.00%;

# 1002 : define_clock {n:inst1.CLK180_BUF} -name {clk_400} -period {10} -clockgroup {default_clkgroup_2}
# c:\syn_clock_priority.sdc
NET "inst1/CLK180_BUF" TNM_NET = "inst1_CLK180_BUF";
TIMESPEC "TS_inst1_CLK180_BUF" = PERIOD "inst1_CLK180_BUF" 10.000 ns HIGH 50.00%;
#End clock constraints

# I/O Registers Packing Constraints
INST "out2" IOB=TRUE;
INST "out1" IOB=FALSE;
INST "b1" IOB=FALSE;
```

After applying the attribute, content in UCF file:

```
# Constraints generated by Synplify Pro maprc, Build 1020R
# Product Version "F-2012.03-SP1"
#
# Period Constraints
#Begin clock constraints
# 1001 : define_clock {clk} -name {clk} -freq {200} -clockgroup {default_clkgroup_0}
# c:\xilinx\syn_clock_priority.sdc
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 5.000 ns HIGH 50.00%;  
# 1002 : define_clock {inst1.CLK180_BUF} -name {clk_400} -period {10} -clockgroup {default_clkgroup_2} -priority {1}
# c:\xilinx\syn_clock_priority.sdc
NET "inst1/CLK180_BUF" TNM_NET = "inst1_CLK180_BUF";
TIMESPEC "TS_inst1_CLK180_BUF" = PERIOD "inst1_CLK180_BUF" 10.000 ns HIGH 50.00% PRIORITY 1;
#End clock constraints

# I/O Registers Packing Constraints
INST "out2" IOB=TRUE;
INST "out1" IOB=FALSE;
INST "b1" IOB=FALSE;
INST "a1" IOB=FALSE;

# End of generated constraints
```

## **syn\_connect\_hrefs**

*Directive*

Allows cross-module referencing (XMR) of signals in a Verilog or VHDL design using a CDC file, without having to modify or update the design.

### **syn\_connect\_hrefs Syntax**

CDC      **syn\_connect\_hrefs** {-readers *{hierPathOfReader}*}  
              -writer *{hierPathOfWriter}*

[CDC File: XMR Example With a Verilog Design](#)

[CDC File: XMR Example With a VHDL Design](#)

### **syn\_connect\_hrefs Values**

| <b>Value</b>            | <b>Description</b>                                                                                                                                              |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -readers                | Hierarchical path of the reader (connect). Specify one or more readers.                                                                                         |
| <i>hierPathOfReader</i> | Specifies the hierarchical path for the readers.<br><i>topModule.instance1.instance2.instance3.signal</i><br>or<br><i>top.inputPortName</i>                     |
| -writer                 | Hierarchical path of the writer (source). Specify only one writer.                                                                                              |
| <i>hierPathOfWriter</i> | Specifies the format of hierarchical path for the writer as follows:<br><i>topModule.instance1.instance2.instance3.signal</i><br>or<br><i>top.inputPortName</i> |

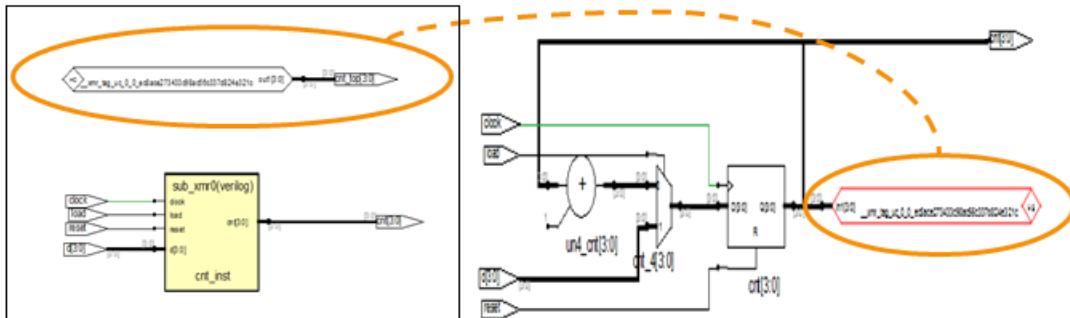
## Description

Allows cross-module referencing (XMR) of signals in a design using a CDC file, without having to modify or update the Verilog or VHDL design. You can specify the `syn_connect_hrefs` directive to add signals for debugging or reference signals deep within the hierarchy of the design that drive the connections.

## CDC File: XMR Example With a Verilog Design

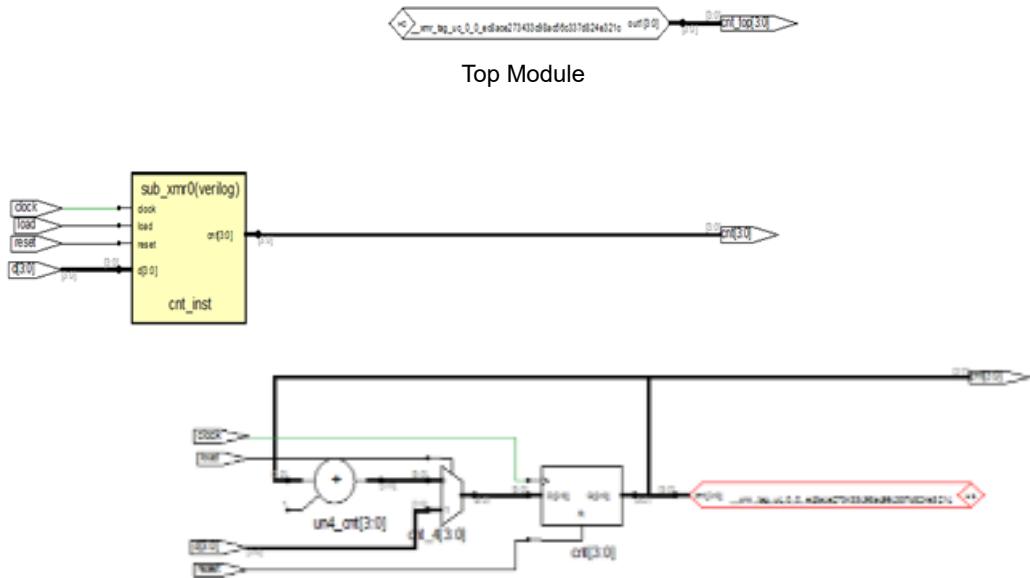
This example shows an assignment within the RTL that came from outside the RTL. For example:

```
assign cnt_top = top.cnt_inst.cnt;
```



The CDC equivalent is shown below:

```
syn_connect_hrefs -readers {top.cnt_top}
                     -writer {top.cnt_inst.cnt}
```



## CDC File: XMR Example With a VHDL Design

The CDC file can be specified as shown below:

```

syn_connect_hrefs -readers {top.xmr_net.we_int}
                     -writer {top.wr_en}

syn_connect_hrefs -readers {work.myTopDesign.out2}
                     -writer {work.myTopDesign.inst1.iz.b}

```

Where: *hierPathOfReader* and *hierPathOfRWriter* values can be specified as shown in the table below.

| Value                                     | Description               |
|-------------------------------------------|---------------------------|
| Library name <sup>1</sup>                 | work                      |
| Entity name <i>topModule</i> <sup>2</sup> | myTopDesign, top          |
| Instance name                             | xmr_net, inst1, iz        |
| Signal name                               | we_int, wr_en, out2(0), b |

1. Library name is optional when the default library is work.
2. If multiple architectures exist for the entity, then the last architecture is selected.

For the effect of using the `syn_connect_hrefs` attribute with a VHDL design, see [VHDL Example: Top-Level top.vhd, on page 551](#).

## Effect of Using `syn_connect_hrefs`

These examples show how XMR can be used in a CDC file to update a Verilog or VHDL design without modifying the RTL.

### Verilog Example: Top-Level top.sv

Use XMR to connect `top.scale` to `top.I1.scale_fac` in the test case below:

```
module top (input clk, [3:0] din1, [3:0] din2, [3:0] scale,
             output [3:0] dout);
    sub I1(clk, din1, din2, dout);
endmodule

module sub(input clk,[3:0] din1, din2, output reg [3:0] dout);
    //Signal accessed using XMR
    logic [3:0] scale_fac;

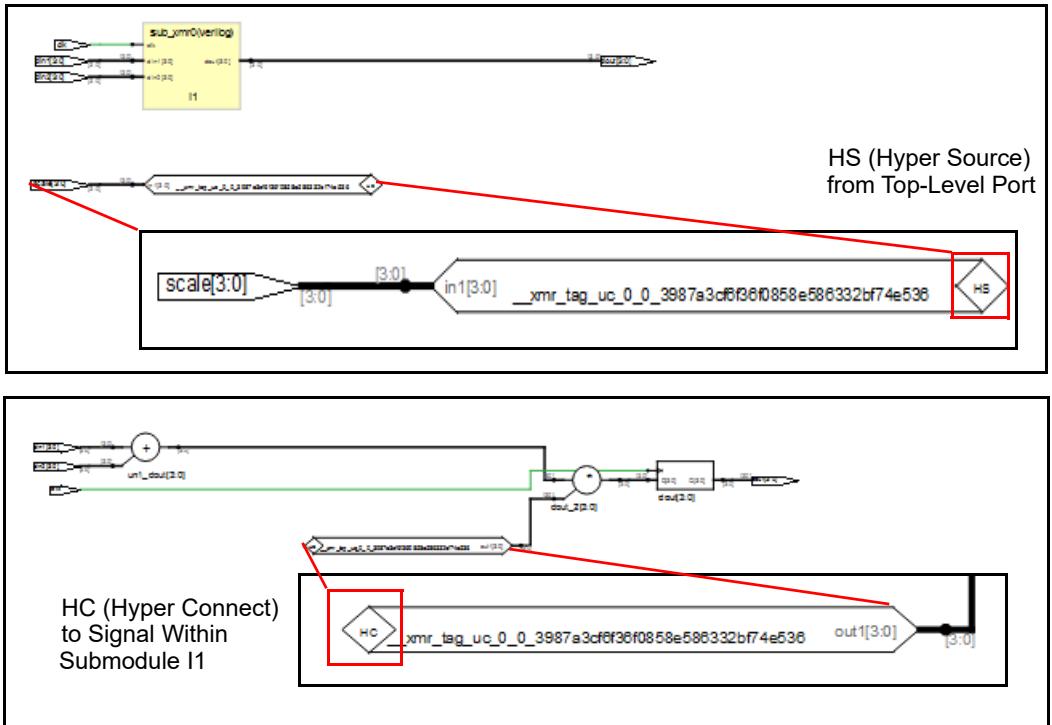
    always @ (posedge clk)
        dout <= (din1 + din2) * scale_fac;
endmodule
```

### Example: hrefs.cdc

You can specify that the top-level port (`top.scale`) is connected to the internal signal (`top.I1.scale_fac`) of the submodule `I1` in a CDC file:

```
syn_connect_hrefs -readers {top.I1.scale_fac} -writer {top.scale}
```

After you run compile, the top-level port uses hyper source to hyper connect to the signal within submodule `I1`, when you push into it as shown in the schematic views below.



### VHDL Example: Top-Level top.vhd

Use XMR to connect `top.wr_en` to `top.xmr_net.we_int` of the submodule `test` in the test case below:

```

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

entity top is
  port(
    clk,rd_en,wr_en : in std_logic;
    rd_addr_ip      : in std_logic_vector(10 downto 0);
    wr_addr_ip      : in std_logic_vector(10 downto 0);
    data_ip         : in std_logic_vector(8 downto 0);
    data_op         : out std_logic_vector(8 downto 0));
end entity top;

```

```

architecture rtl of top is
component test
  port(
    clk,rd_en : in std_logic;
    rd_addr_ip,wr_addr_ip : in std_logic_vector(10 downto 0);
    data_ip : in std_logic_vector(8 downto 0);
    data_op : out std_logic_vector(8 downto 0));
end component test;

begin
---- submodule test.vhd instantiated.
xmr_net : test
  port map (clk => clk,
            rd_en      => rd_en,
            rd_addr_ip => rd_addr_ip,
            wr_addr_ip => wr_addr_ip,
            data_ip     => data_ip,
            data_op     => data_op);
end rtl;

```

## VHDL Example: Submodule test.vhd

```

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

entity test is
  port(
    clk,rd_en      : in std_logic;
    rd_addr_ip    : in std_logic_vector(10 downto 0);
    wr_addr_ip    : in std_logic_vector(10 downto 0);
    data_ip       : in std_logic_vector(8 downto 0);
    data_op       : out std_logic_vector(8 downto 0));
end entity test;

architecture rtl of test is
type ram_array is array(2047 downto 0) of
  std_logic_vector(8 downto 0);
signal ram_data : ram_array;
signal we_int : std_logic; --- XMR signal defined for XMR
connection.

```

```
begin
    write_to_ram: process(clk) begin
        if rising_edge(clk) then
            if (we_int = '1') then
                ram_data(to_integer(unsigned(wr_addr_ip))) <= data_ip;
            end if;
        end if;
    end process write_to_ram;

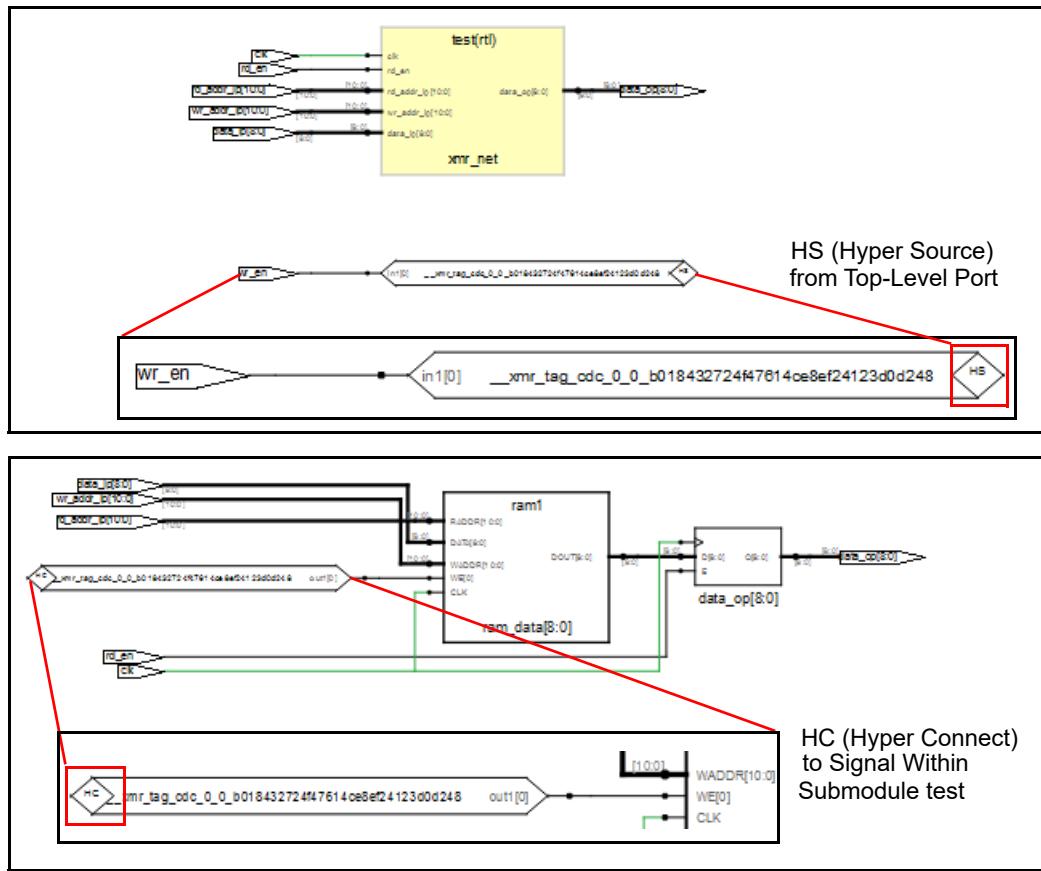
    read_from_ram: process(clk)      begin
        if (rising_edge (clk)) then
            if (rd_en = '1') then
                data_op <= (ram_data(to_integer(unsigned(rd_addr_ip))));
            end if;
        end if;
    end process read_from_ram;
end rtl;
```

### Example: hrefs.cdc

You can specify that the top-level port (`top.wr_en`) is connected to the internal signal (`top.xmr_net.we_int`) of the submodule test in a CDC file:

```
syn_connect_hrefs -readers {top.xmr_net.we_int}
                    -writer {top.wr_en}
```

After you run `compile`, the top-level port uses hyper source to hyper connect to the signal within submodule test, when you push into it as shown in the schematic views below.



## Limitations

Cross-module referencing support includes the following limitations:

- Mixed language (Verilog to VHDL or VHDL to Verilog) designs are not supported.
- The XMR signals cannot be used for logic functions, such as and, or, or xnor.
- The XMR signals cannot be used for arithmetic functions, such as add, subtract, or multiply.

## corrupt\_pd

### *UPF Attribute*

Specifies how corruption is to be inserted into sequential elements (RAMs and flip-flops) when power is switched off.

### Description

The corrupt\_pd attribute is used with all power domains that have a create\_power\_switch command, to specify how corruption is to be implemented.

Power-domain corruption is based on parameters specified with the create\_power\_switch command and the value specified by the corrupt\_pd attribute. The create\_power\_switch command is defined in the UPF file, and the attribute is included in an FDC constraint file. For details on the syntax for the create\_power\_switch command, see [create\\_power\\_switch, on page 267](#).

Corruption is not enabled for power domains that do not have a create\_power\_switch command. For power domains with a create\_power\_switch command, the corrupt\_pd attribute defines one of four corruption modes:

| Attribute Setting | Description                                                                                                                            |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b>none</b>       | Disables corruption insertion regardless of the create_power_switch command parameters; this is the default setting for the attribute. |
| <b>all1s</b>      | Sets all sequential elements to all 1's                                                                                                |
| <b>all0s</b>      | Sets all sequential elements to all 0's                                                                                                |
| <b>random</b>     | Randomly sets all sequential elements to 1's and 0's.                                                                                  |

### Constraint File Syntax and Example

To set the corruption mode for all power domains, specify the corrupt\_pd attribute as a global attribute. You can set it to one of the four corruption modes:

```
define_global_attribute {corrupt_pd} {all1s |all0s |random |none}
```

You can concurrently set the corrupt\_pd attribute at the register level to override the global corruption setting on that register. The value none is the only legal value at the register level. The complete syntax and examples are shown below:

**define\_attribute {registerInstance} {corrupt\_pd} {none}**

The following sequence sets all1s corruption for the power domains and then disables this setting on addrb\_reg:

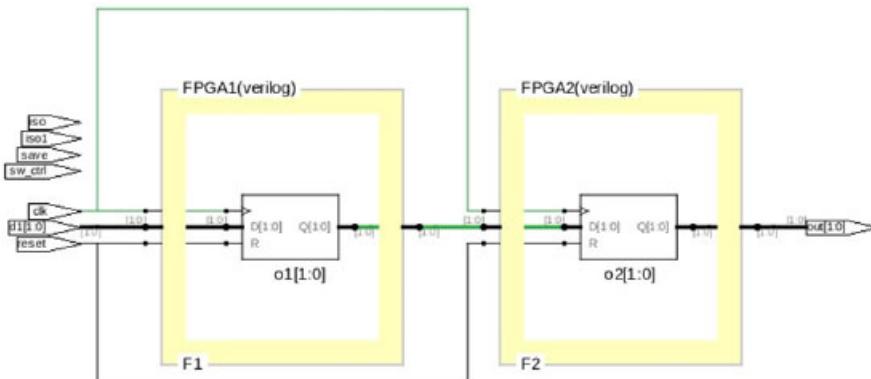
```
define_global_attribute {corrupt_pd} {all1s}
define_attribute {i:addrb_reg} {corrupt_pd} {none}
```

## FDC Example

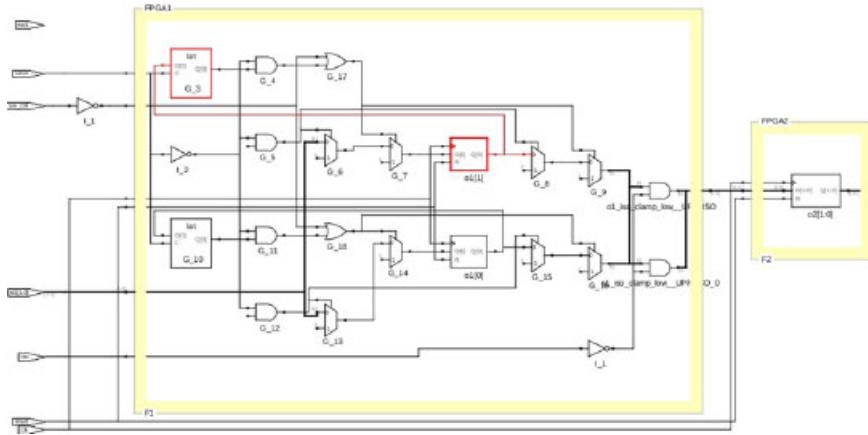
|   | Enable                              | Object Type | Object   | Attribute      | Value | Value Type | Description                  |
|---|-------------------------------------|-------------|----------|----------------|-------|------------|------------------------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Global> | syn_corrupt_pd | all1s | string     | Power domain corruption type |
| 2 |                                     |             |          |                |       |            |                              |

## Examples of Extra Corruption Logic

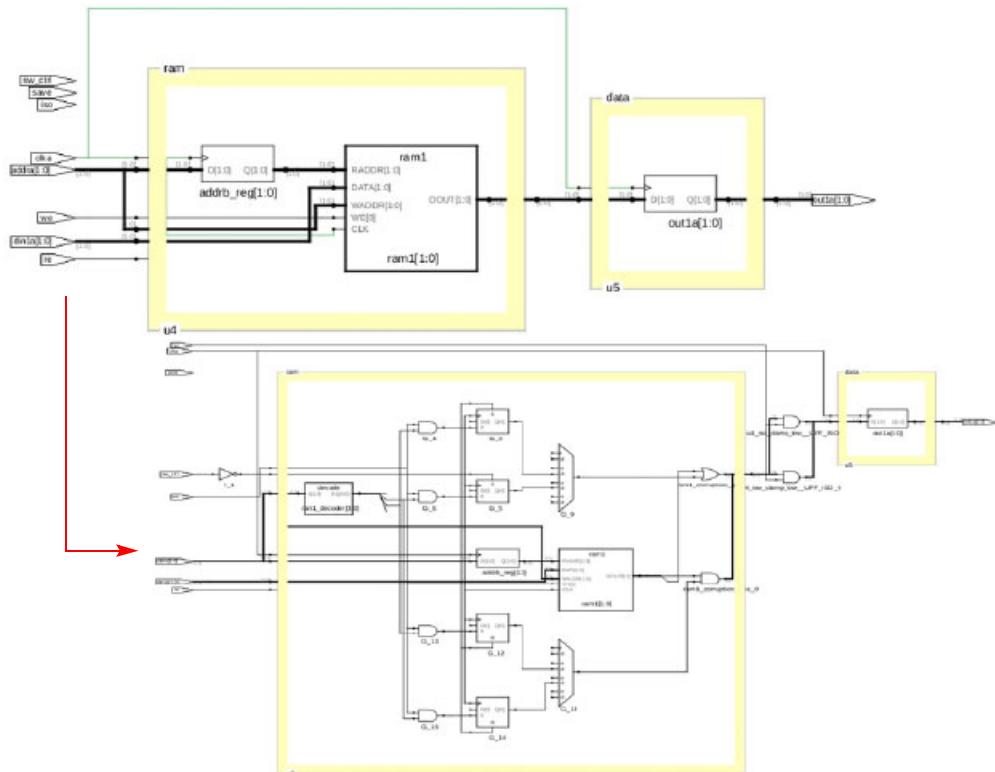
This constraint adds extra logic as required by the corruption mode. The following figure shows a flip-flop setup:



The following figure shows the extra logic added for flip-flop corruption when the corruption mode is set to all1s:



The following figure shows a RAM setup and the extra logic added for RAM corruption when the global setting is random and `addrb_reg` is set to none:



## **syn\_cp\_use\_fast\_synthesis**

### *Attribute*

Specifies manual compile points in the manual or automatic compile point flow.

Return to [Attributes and Directives Summary](#).

### **Description**

Use the Synthesis Strategy fast option when specifying manual compile points in the manual compile point and automatic compile point flows. The fast synthesis strategy is applied globally when set. However, you can set the syn\_cp\_use\_fast\_synthesis attribute to apply the fast synthesis strategy to specific manual compile points.

To enable this attribute, set syn\_cp\_use\_fast\_synthesis to 1. The attribute can be applied on a selected manual compile point view/module for the design. Define the compile points in the top-level constraint file.

### **syn\_cp\_use\_fast\_synthesis Syntax**

FDC            **define\_attribute {v:cp\_name} syn\_cp\_use\_fast\_synthesis {1|0}**

## **syn\_diff\_io**

### *Attribute*

The `syn_diff_io` attribute controls differential I/O buffer inferencing.

### **Description**

The `syn_diff_io` attribute controls differential I/O buffer inferencing in designs: IBUFDS, IBUFGDS, OBUFDS, OBUFTDS and IOBUFDS. Attach the attribute to the inputs of the buffer. Use this attribute to identify differential input buffers when using either combinational or sequential style to code the buffer.

The `syn_diff_io` data type is Boolean. A value of 1 or true enables automatic differential I/O buffer inferencing. The default is false or 0, no automatic inferencing.

The `syn_diff_io` attribute is supported in the compile point flow.

### **syn\_diff\_io Syntax**

|         |                                                                                                                              |                                            |
|---------|------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute object {syn_diff_io} {1 0}</code><br><code>define_global_attribute {syn_diff_io} {1 0}</code>         | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_diff_io = 1 0 */</code>                                                                        | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_diff_io : true false;</code><br><code>attribute syn_diff_io of object : objectType is true false;</code> | <a href="#">VHDL Example</a>               |

### **Constraints Editor Example**

|   | Enable                              | Object Type | Object  | Attribute   | Value | Value Type | Description                               |
|---|-------------------------------------|-------------|---------|-------------|-------|------------|-------------------------------------------|
| 1 | <input checked="" type="checkbox"/> | port        | p:in1_p | syn_diff_io | 1     | boolean    | Allows inference of differential I/O pads |
| 2 | <input checked="" type="checkbox"/> | port        | p:in1_n | syn_diff_io | 1     | boolean    | Allows inference of differential I/O pads |
| 3 | <input checked="" type="checkbox"/> | port        | p:out1  | syn_diff_io | 1     | boolean    | Allows inference of differential I/O pads |
| 4 | <input checked="" type="checkbox"/> | port        | p:out2  | syn_diff_io | 1     | boolean    | Allows inference of differential I/O pads |

## Verilog Example

```
module test(in1_p, in1_n, clk, out1,out2);
    input in1_p/* synthesis syn_diff_io = 1*/,
          in1_n/* synthesis syn_diff_io = 1*/;
    input clk;
    output out1/* synthesis syn_diff_io = 1*/,
          out2/* synthesis syn_diff_io = 1*/;
    reg out1,out2;
    reg in1;

    always@(in1_p or in1_n)
        if (in1_p != in1_n) in1 = in1_p;
    always@(posedge clk)
        out1 <= in1;
    assign out2 = ~out1;
endmodule
```

## VHDL Example

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

entity test is
    port (in1_p : in std_logic;
          in1_n : in std_logic;
          clk : in std_logic;
          out1 : out std_logic;
          out2 : out std_logic );
attribute syn_diff_io: boolean;
attribute syn_diff_io of in1_p,in1_n,out1,out2: signal is true;
end test;

architecture arch of test is
signal in1 : std_logic;
begin
    process(in1_p,in1_n)
begin
    if(in1_p /= in1_n)then
        in1 <= in1_p;
    end if;
end process;
process(clk)
begin
    if(clk'event and clk = '1')then
```

```

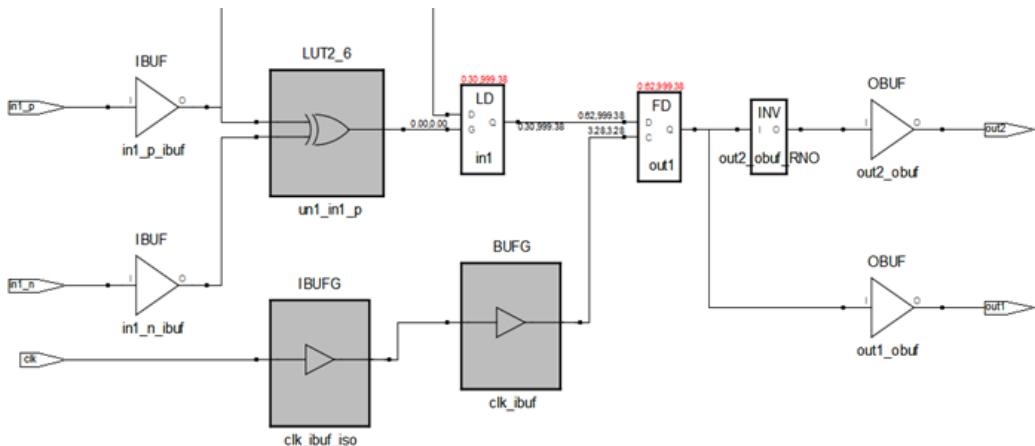
        out1 <= in1;
    end if;
end process;
out2 <= not out1;
end arch;
```

## Effect of Using syn\_diff\_io

The following figure shows the attribute set to 0. The software disables automatic differential I/O buffer inferencing:

Verilog    input in1\_p /\*synthesis syn\_diff\_io = 0\*/;  
           in1\_n /\*synthesis syn\_diff\_io = 0\*/;  
           output out1 /\*synthesis syn\_diff\_io = 0\*/;  
           out2 /\*synthesis syn\_diff\_io = 0\*/;

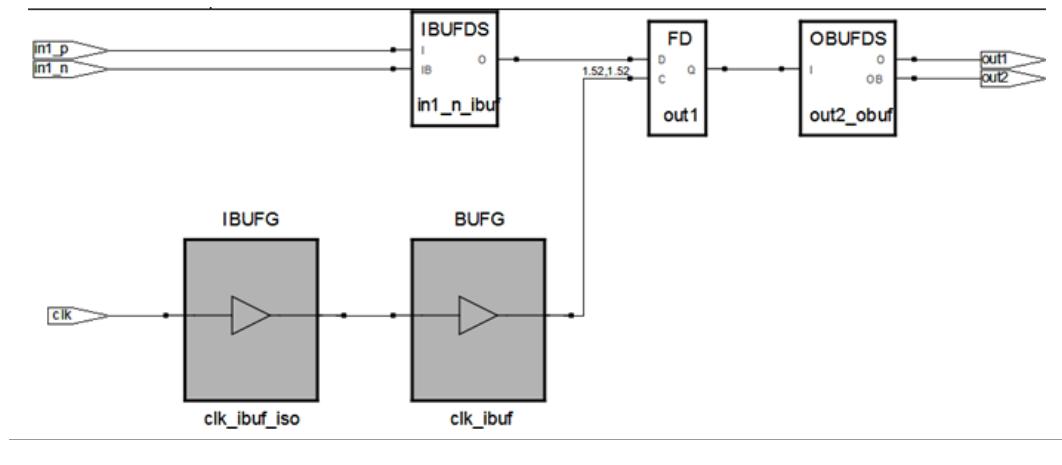
VHDL    attribute syn\_diff\_io of in1\_p,in1\_n,out1,out2:signal is false;



The next figure shows the attribute set to 1. The software enables automatic differential I/O buffer inferencing:

```
Verilog input in1_p /*synthesis syn_diff_io = 1*/;
in1_n /*synthesis syn_diff_io = 1*/;
output out1 /*synthesis syn_diff_io = 1*/;
out2 /*synthesis syn_diff_io = 1*/;
```

```
VHDL attribute syn_diff_io of in1_p,in1_n,out1,out2:signal is true;
```



## **syn\_direct\_enable**

*Attribute, Directive*

Controls the assignment of a clock enable net to the dedicated enable pin of a storage element (flip-flop).

### Description

The `syn_direct_enable` attribute controls the assignment of a clock enable net to the dedicated enable pin of a storage element (flip-flop). Using this attribute, you can direct the mapper to use a particular net as the only clock enable when the design has multiple clock-enable candidates.

As a directive, you use `syn_direct_enable` to infer flip-flops with clock enables. To do so, enter `syn_direct_enable` as a directive in source code, not the constraints editor spreadsheet.

### **syn\_direct\_enable Syntax**

|         |                                                                          |                                 |
|---------|--------------------------------------------------------------------------|---------------------------------|
| FDC     | <code>define_attribute {object} syn_direct_enable {1}</code>             | <a href="#">FDC Example</a>     |
| Verilog | <code>object /* synthesis syn_direct_enable = 1 */;</code>               | <a href="#">Verilog Example</a> |
| VHDL    | <code>attribute syn_direct_enable of object : objectType is true;</code> | <a href="#">VHDL Example</a>    |

### **FDC Example**

| Enable                              | Object Type | Object   | Attribute                      | Value | Value Type | Description            |
|-------------------------------------|-------------|----------|--------------------------------|-------|------------|------------------------|
| <input checked="" type="checkbox"/> | <any>       | <Global> | <code>syn_direct_enable</code> | 1     | boolean    | Preferred clock enable |

### **Verilog Example**

```
module direct_enable(q1, d1, clk, e1, e2, e3);
parameter size=5;
input [size-1:0] d1;
input clk;
input e1,e2;
input e3 /* synthesis syn_direct_enable = 1 */;
```

```
output reg [size-1:0] q1;
(posedge clk)
    if (e1&e2&e3)
        q1 = d1;
endmodule
```

## VHDL Example

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

entity direct_enable is
    port (d1 : in std_logic_vector(4 downto 0);
          e1,e2,e3,clk : in std_logic;
          q1 : out std_logic_vector(4 downto 0) );
attribute syn_direct_enable: boolean;
attribute syn_direct_enable of e3: signal is true;
end;

architecture d_e of direct_enable is
begin
    process (clk) begin
        if (clk = '1' and clk'event) then
            if (e1='1' and e2='1' and e3='1') then
                q1<=d1;
            end if;
        end if;
    end process;
end architecture;
```

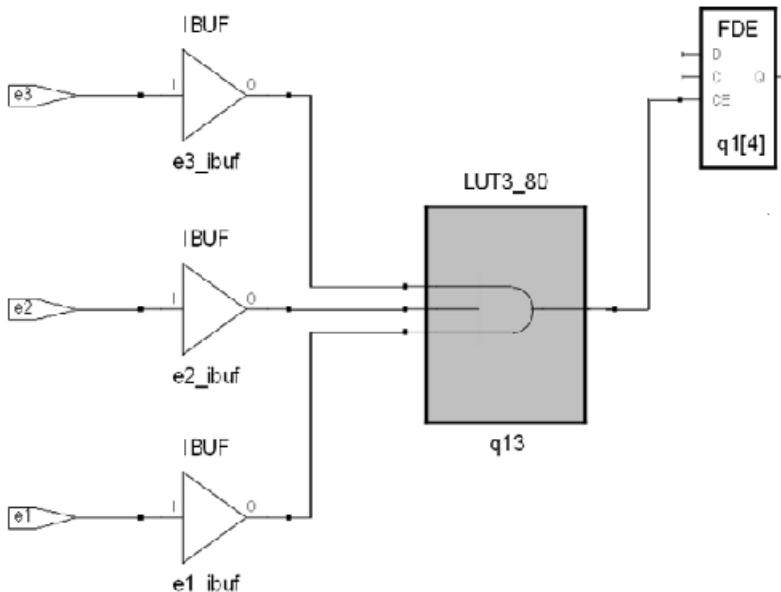
## Effect of Using syn\_direct\_enable

Before applying syn\_direct\_enable:

Verilog      input e3 /\* synthesis syn\_direct\_enable = 0 \*/;

VHDL      attribute syn\_direct\_enable of e3: signal is false;

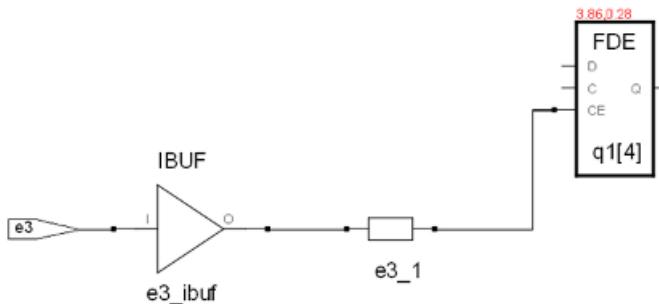
---



**After applying syn\_direct\_enable:**

Verilog      input e3 /\* synthesis syn\_direct\_enable = 1 \*/;

VHDL      attribute syn\_direct\_enable of e3: signal is true;



## **syn\_direct\_reset**

*Attribute/Directive*

Controls the assignment of a net to the dedicated reset pin of a synchronous storage element (flip-flop).

### **Description**

The `syn_direct_reset` attribute controls the assignment of a net to the dedicated reset pin of a synchronous storage element (flip-flop). You can direct the mapper to only use a particular net as the reset when the design has a conditional reset on multiple candidates. This attribute can be applied to separate AND or OR logic and is valid for only one input of the reset logic cone.

### **syn\_direct\_reset Syntax**

The following table summarizes the syntax in different files:

|         |                                                                                                                                 |                                           |
|---------|---------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| FDC     | <code>define_attribute syn_direct_reset {0 1}</code>                                                                            | <a href="#">Constraint Editor Example</a> |
| Verilog | <code>object /* synthesis syn_direct_reset = 0 1 */</code>                                                                      | <a href="#">Verilog Example</a>           |
| VHDL    | <code>attribute syn_direct_reset : boolean;</code><br><code>attribute syn_direct_reset of object : signal is true false;</code> | <a href="#">VHDL Example</a>              |

### **Constraint Editor Example**

|   | Enabled                             | Object Type | Object | Attribute        | Value | Val Type | Description |
|---|-------------------------------------|-------------|--------|------------------|-------|----------|-------------|
| 1 | <input checked="" type="checkbox"/> |             | p:a    | syn_direct_reset | 1     |          |             |

### **Verilog Example**

```
module test (
    input a /* synthesis syn_direct_reset = 1 */,
    input b,
    input c,
    input clk,
    input [1:0] din,
    output reg [1:0] dout );
```

```
always @ (posedge clk)
  if (a == 1'b1 || b == 1'b1 || c == 1'b1)
    dout = 2'b00;
  else
    dout = din;
endmodule
```

## VHDL Example

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

entity test is
  port (a : in std_logic;
        b : in std_logic;
        c : in std_logic;
        clk : in std_logic;
        din : in std_logic_vector(1 downto 0);
        dout : out std_logic_vector(1 downto 0) );
attribute syn_direct_reset : boolean;
attribute syn_direct_reset of a : signal is true;
end entity test;

architecture beh of test is
signal rst : std_logic;
begin
rst <= a or b or c;
process(clk, rst)
begin
  if (clk='1' and clk'event) then
    if (rst = '1') then
      dout <= (others => '0');
    else
      dout <= din;
    end if;
  end if;
end process;
end beh;
```

## Effect of Using syn\_direct\_reset

The following figure shows the attribute set to 1. The software assigns a net for the design to the dedicated reset pin of a synchronous storage element (flip-flop):

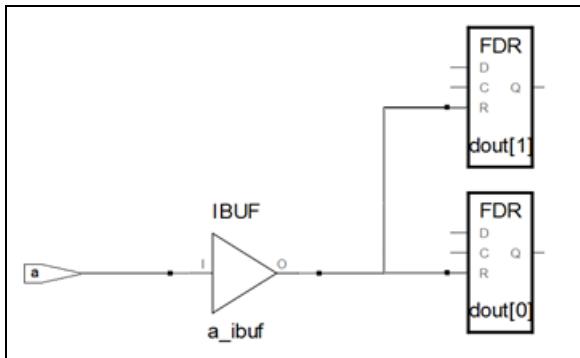
---

```
Verilog    input a /*synthesis syn_direct_reset=1*/;
```

---

```
VHDL    attribute syn_direct_reset of a : signal is true;
```

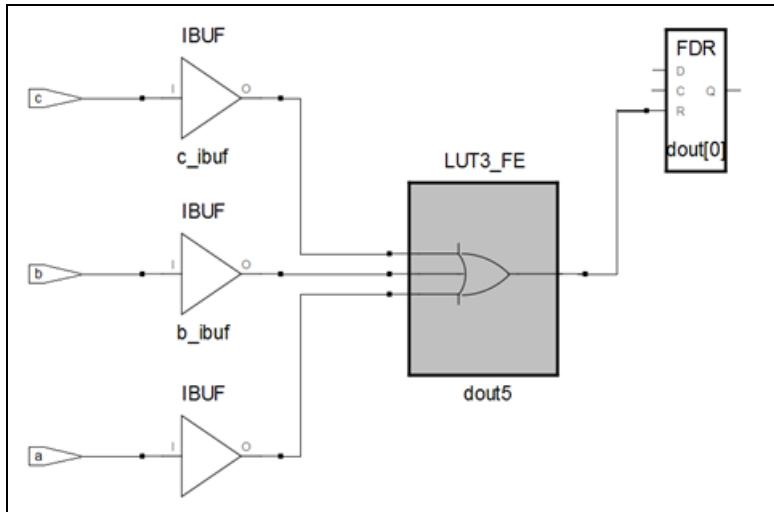
---



The next figure shows the attribute set to 0. The software does not assign a net for the design to the dedicated reset pin of a synchronous storage element (flip-flop):

Verilog    input a /\*synthesis syn\_direct\_reset = 0\*/;

VHDL    attribute syn\_direct\_reset of a : signal is false;



## **syn\_direct\_set**

*Attribute/Directive*

Controls the assignment of a net to the dedicated set pin of a synchronous storage element (flip-flop).

### Description

The `syn_direct_set` attribute controls the assignment of a net to the dedicated set pin of a synchronous storage element (flip-flop). You can direct the mapper to only use a particular net as the set when the design has a conditional set on multiple candidates. This attribute can be applied to separate OR logic and is valid for only one input of the set logic cone.

### **syn\_direct\_set Syntax**

The following table summarizes the syntax in different files:

|         |                                                                                                                             |                                           |
|---------|-----------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| FDC     | <code>define_attribute syn_direct_set {0 1}</code>                                                                          | <a href="#">Constraint Editor Example</a> |
| Verilog | <code>object /* synthesis syn_direct_set = 0 1 */;</code>                                                                   | <a href="#">Verilog Example</a>           |
| VHDL    | <code>attribute syn_direct_set : boolean;</code><br><code>attribute syn_direct_set of object : signal is true false;</code> | <a href="#">VHDL Example</a>              |

### Constraints Editor Example

|   | Enabled                             | Object Type | Object | Attribute      | Value | Val Type | Description |
|---|-------------------------------------|-------------|--------|----------------|-------|----------|-------------|
| 1 | <input checked="" type="checkbox"/> |             | p:a    | syn_direct_set | 1     |          |             |

### Verilog Example

```
module test (
    input a /* synthesis syn_direct_set = 1 */,
    input b,
    input c,
    input clk,
    input [1:0] din,
    output reg [1:0] dout );
```

```
always @ (posedge clk)
  if (a == 1'b1 || b == 1'b1 || c == 1'b1)
    dout = 2'b11;
  else
    dout = din;
endmodule
```

## VHDL Example

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

entity test is
  port (a : in std_logic;
        b : in std_logic;
        c : in std_logic;
        clk : in std_logic;
        din : in std_logic_vector(1 downto 0);
        dout : out std_logic_vector(1 downto 0) );
attribute syn_direct_set : boolean;
attribute syn_direct_set of a : signal is true;
end entity test;

architecture beh of test is
signal set : std_logic;
begin
set <= a or b or c;
  process(clk, set)
begin
  if (clk='1' and clk'event) then
    if (set = '1') then
      dout <= (others => '1');
    else
      dout <= din;
    end if;
  end if;
  end process;
end beh;
```

## Effect of Using syn\_direct\_set

The following figure shows the attribute set to 1. The software assigns a net for the design to the dedicated set pin of a synchronous storage element (flip-flop):

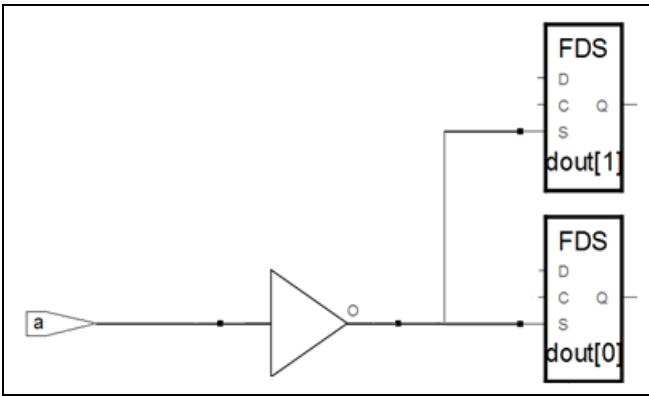
---

```
Verilog    input a /*synthesis syn_direct_set=1*/;
```

---

```
VHDL    attribute syn_direct_set of a : signal is true;
```

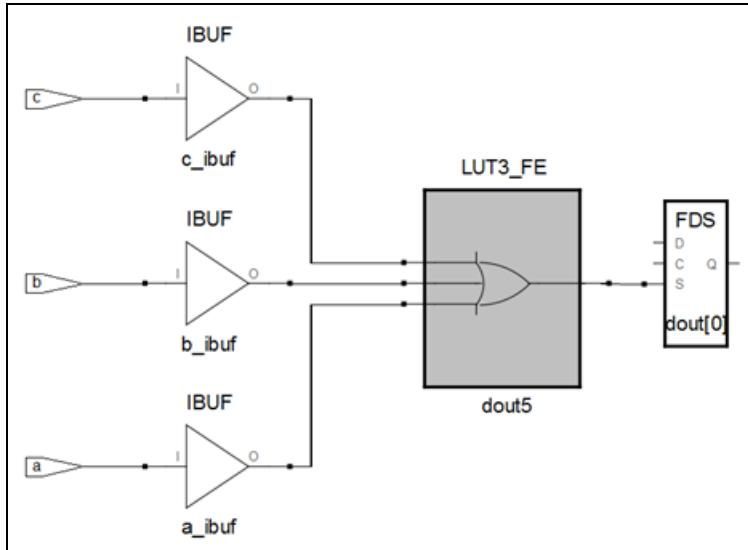
---



The next figure shows the attribute set to 0. The software does not assign a net for the design to the dedicated set pin of a synchronous storage element (flip-flop):

Verilog    input a /\*synthesis syn\_direct\_set = 0\*/;

VHDL    attribute syn\_direct\_set of a : signal is false;



## **syn\_disable\_purifyclock**

### *Attribute*

Disables the purification of clocks.

### **Description**

The `syn_disable_purifyclock` attribute disables purification of clocks. Clock purification is on by default. Clock purification duplicates nets that drive clock and data pins simultaneously, to ensure clock nets do not drive data pins. Currently, clock purification applies for generated clocks. The attribute can be set globally or on a net.

FDC      `define_attribute {object} syn_disable_purifyclock {1}`

[FDC Example](#)

Verilog    `object /* synthesis syn_disable_purifyclock ="{0|1}" */;`

[Verilog Example](#)

VHDL     `attribute syn_disable_purifyclock of object :objectType is true ;`

[VHDL Example](#)

### **FDC Example**

The constraint file syntax for the attribute is

```
define_attribute {object} syn_disable_purifyclock {1}
```

```
define_global_attribute syn_disable_purifyclock {1}
```

Where `object` is a net.

### **Verilog Example**

```
module FF2LUT2FF
  input wire clk,
  input wire din_r,
  output dout,dout_r
  wire gen_clk /* synthesis syn_disable_purifyclock=1 */
```

```

/* synthesis syn_keep=1 */;
FD u0(.C(clk),.D(~gen_clk),.Q(gen_clk));
FD ud(.C(clk),.D(gen_clk),.Q(dout));
FD uc(.D(din_r),.Q(dout_r),.C(gen_clk));
endmodule

```

## VHDL Example

```

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

entity FF2LUT2FF is
    port (clk : in std_logic;
          din_r : in std_logic;
          dout : out std_logic;
          dout_r : out std_logic );
end;

architecture beh of FF2LUT2FF is
signal gen_clk : std_logic;
attribute syn_keep : boolean;
attribute syn_keep of gen_clk : signal is true;
attribute syn_disable_purifyclock : boolean;
attribute syn_disable_purifyclock of gen_clk :
    signal is true;
component FD
generic (INIT : bit := '0');
    port (Q : out std_ulogic;
          C : in std_ulogic;
          D : in std_ulogic );
end component;

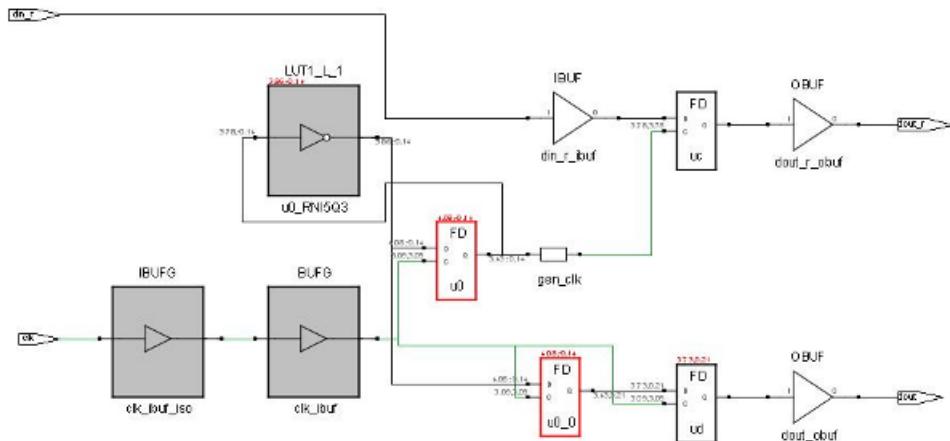
begin
u0: FD port map(C=>clk,D=>not gen_clk,Q=>gen_clk);
ud: FD port map(C=>clk,D =>gen_clk,Q=>dout);
uc: FD port map(D=>din_r,Q=>dout_r,C=>gen_clk);

```

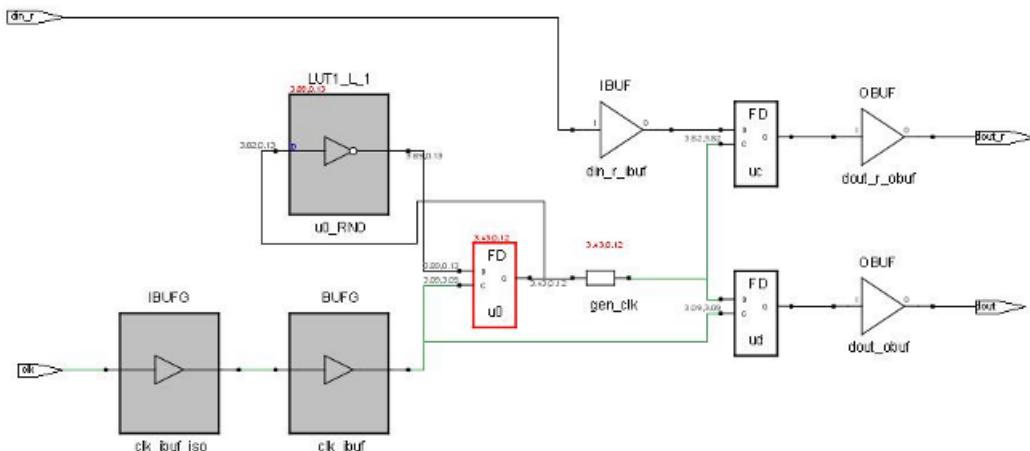
Use the `syn_keep` attribute on the nets along with this attribute to ensure that net names are preserved.

## Effect of Using syn\_disable\_purifyclock

The following shows a design before applying the `syn_disable_purifyclock` attribute:



The following shows the same design after applying the `syn_disable_purifyclock` attribute:



## syn\_dspstyle

### *Attribute*

Determines whether the object is mapped to a technology-specific DSP component.

The syn\_dspstyle options for designs with DSPs vary. See the following for specific details:

- [syn\\_dspstyle](#), on page 580
  - [syn\\_dspstyle Syntax](#), on page 581
  - [Constraint Editor Examples](#), on page 581
  - [Verilog Example](#), on page 582
  - [VHDL Example](#), on page 582
  - [Effect of Using syn\\_dspstyle](#), on page 583
  - [syn\\_dspstyle in SIMD Mode](#), on page 586

## syn\_dspstyle Values

| Value              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| logic              | Maps the object to logic instead of the DSP resources.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| dsp48<br>(Default) | Maps the specified object to a DSP48. See <a href="#">syn_dspstyle</a> , on page 580 for a description of how different objects are handled.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| simd               | Specifies the DSP configuration available in Single Instruction Multiple Data (SIMD) mode, and leaves it to the synthesis software to determine how to combine adders and pack them into one DSP slice. See <a href="#">syn_dspstyle in SIMD Mode</a> , on page 586 for details.                                                                                                                                                                                                                                                                                                                                                               |
| simd: <i>i</i>     | Specifies the DSP configuration available in SIMD mode. <i>i</i> is an index value that is an integer less than or equal to 24. Specify an index value to pair and pack multiple adders into the same DSP slice. You can pack the following components: <ul style="list-style-type: none"><li>• Two adders or subtractors with the same index value and functionality, with an input size less than or equal to 24 bits</li><li>• Four adders or subtractors with the same index value and functionality, with an input size less than or equal to 12 bits.</li></ul> See <a href="#">syn_dspstyle in SIMD Mode</a> , on page 586 for details. |

## **syn\_dspstyle**

This attribute determines if the object to which the attribute is applied is mapped to a DSP48 component. The value of this attribute is either logic or dsp48. The object to which you apply this attribute determines what gets mapped into the DSP48.

| Object                           | Behavior                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Multiplier                       | <p>By default, multipliers are mapped to DSP48, so setting <code>syn_dspstyle</code> to <code>dsp48</code> on a multiplier is redundant. When a multiplier is mapped, the mapper also packs the adder/subtractor driven by the multiplier into the same DSP48 if possible.</p> <p>If your design has constants driving multiplier logic, the tool could perform constant propagation before DSP48 packing. Use <code>syn_keep</code> to prevent constant propagation optimizations.</p> <p>When you set the attribute value to <code>logic</code>, the mapper uses logic to implement the multiplier. For simple multipliers, <code>syn_dspstyle</code> has the same effect as using <code>syn_multstyle</code>. For example, <code>syn_dspstyle = dsp48</code> results in the same implementation as <code>syn_multstyle = block_mult</code>. In both cases, the tool implements a block multiplier.</p>                                                                                                                                                                                                                                                                              |
| Adder,<br>Subtractor,<br>Counter | <p>By default an adder/subtractor without a multiplier feeding it or a counter is mapped to logic. You do not need to explicitly set <code>syn_dspstyle=logic</code> on an adder/subtractor unless you want to decouple the mult-add or mult-subtract structure and map the multiplier into the DSP48 without the adder/subtractor.</p> <p>When the attribute is set to <code>dsp48</code>, the adder/subtractor or counter is mapped into the DSP48. If your design has constants driving multiplier logic, the tool could perform constant propagation before DSP48 packing. Use the <code>syn_keep</code> directive to prevent any constant propagation optimizations.</p> <p>By default, only one adder is packed into a single DSP48 slice. The DSP structure supports two 24-bit adders and four 12-bit adders packed into a single DSP48 slice. For details, see <a href="#"><i>syn_dspstyle in SIMD Mode , on page 586</i></a>.</p> <p>By default, the tool automatically maps counters with a large bit size that are not timing-critical to DSP48s. If you want to map these counters to logic, you must explicitly set <code>syn_dspstyle</code> to <code>logic</code>.</p> |

| Object                           | Behavior                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Register                         | <p>When a register is associated with an adder/subtractor or multiplier, the mapper determines if the register needs to be mapped into the DSP48. Setting <code>syn_DSPstyle</code> to <code>dsp48</code> has no effect.</p> <p>Setting <code>syn_DSPstyle</code> to <code>logic</code> on a register prevents the register from being mapped into the DSP48. If you do not want to map a register to a DSP48, set <code>syn_DSPstyle</code> to <code>logic</code> for the register. Note that when the attribute is attached to a register, it only applies to the register, not to the associated multipliers, adders, or other objects.</p> <p>You can set the SIMD values on a view.</p> |
| View,<br>Module,<br>Architecture | <p>Setting the attribute on a view, module, or architecture applies the attribute to the entire view.</p> <p>You can set the SIMD values on a view.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

## **syn\_DSPstyle Syntax**

This table shows how to specify the attribute in different files:

|         |                                                                                                                                          |                                            |
|---------|------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute {object} syn_DSPstyle {dsp48   logic}</code><br><code>define_global_attribute syn_DSPstyle {dsp48   logic}</code> | <a href="#">Constraint Editor Examples</a> |
| Verilog | <code>object /* synthesis syn_DSPstyle = "dsp48   logic" */;</code>                                                                      | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_DSPstyle of object : objectType is dsp48   logic;</code>                                                             | <a href="#">VHDL Example</a>               |

## **Constraint Editor Examples**

|   | Enabled                             | Object Type | Object          | Attribute    | Value | Val Type |
|---|-------------------------------------|-------------|-----------------|--------------|-------|----------|
| 1 | <input checked="" type="checkbox"/> | <any>       | i:P_1[16:0]     | syn_DSPstyle | dsp48 | string   |
| 2 | <input checked="" type="checkbox"/> | <any>       | i:un1_b_d[15:0] | syn_DSPstyle | logic | string   |

```
define_global_attribute {syn_DSPstyle} {simd}
define_attribute {reg1} syn_DSPstyle {dsp48}
define_attribute {my_add1} syn_DSPstyle {simd:1}
define_attribute {v:work.mult} syn_DSPstyle {logic}
```

## Verilog Example

```

module dsp_style (clk,A,B,PC,P);
  input clk;
  input [7:0] A,B,PC;
  output reg [16:0] P;
  reg [7:0]a_d,b_d;
  reg [16:0] m /* synthesis syn_DSPstyle = "logic" */;

  always @ (posedge clk) begin
    a_d <= A;
    b_d <= B;
    m   <= a_d * b_d;
    P   <= m + PC;
  end

endmodule

```

See [Example 1 – Specifying SIMD, on page 587](#) and [Example 2 – Specifying a SIMD Index Value, on page 589](#) for Verilog examples of the attribute specified with SIMD mode values.

## VHDL Example

```

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

entity dsp_style is
  port (clk : in std_logic ;
        A : in std_logic_vector(7 downto 0) ;
        B : in std_logic_vector(7 downto 0) ;
        PC : in std_logic_vector(7 downto 0) ;
        P : out std_logic_vector(15 downto 0) );
end dsp_style ;

architecture rtl of dsp_style is
signal m : std_logic_vector(15 downto 0) ;
attribute syn_DSPstyle : string ;
attribute syn_DSPstyle of m : signal is "logic" ;
begin
  process(clk)
begin
  if (clk'event and clk = '1') then

```

```

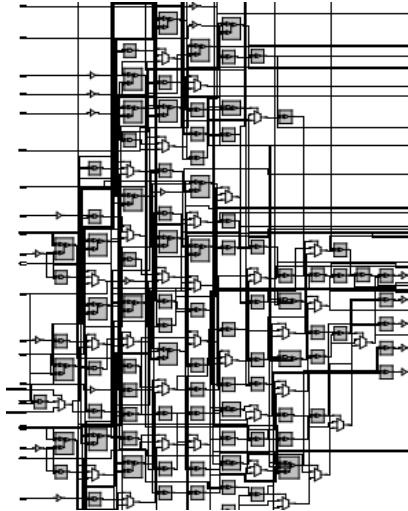
        m <= A * B;
        P <= m + PC;
    end if ;
end process ;
end rtl ;

```

## Effect of Using syn\_dspstyle

In the example below, the `syn_dspstyle` attribute is set to `logic` globally on the module, so all four multipliers in the design are implemented as logic.

|         |                                                                           |
|---------|---------------------------------------------------------------------------|
| FDC     | define_global_attribute syn_dspstyle {logic}                              |
| Verilog | module dsp (a,b,c,d,e,f,g,h,r1,r2)<br>/*synthesis syn_dspstyle="logic"*/; |



```

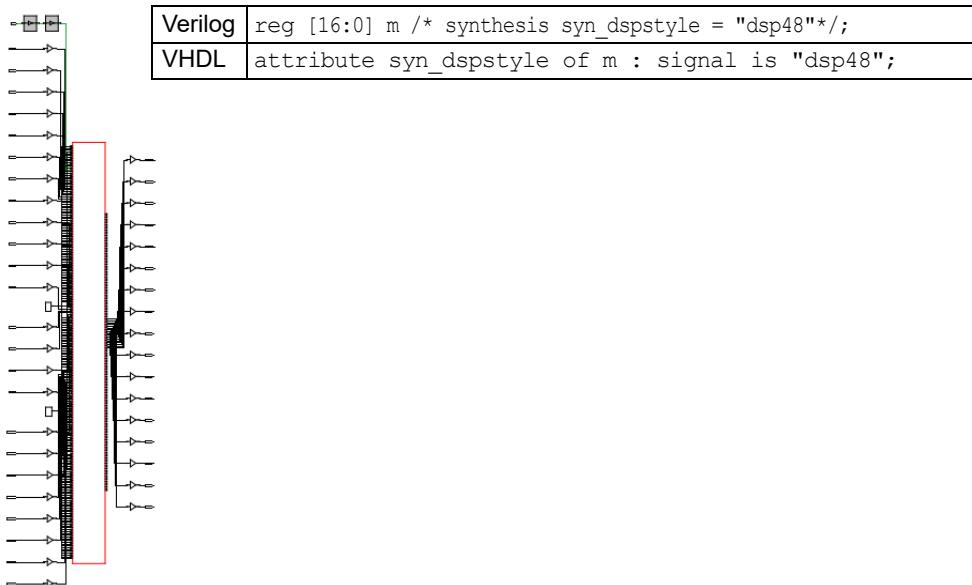
module dsp(a,b,c,d,e,f,g,h,r1,r2) /* synthesis
syn_dspstyle="logic"*/;
input [7:0] a,b,c,d,e,f,g,h;
output [15:0] r1;
output [15:0] r2;
wire [15:0] temp1;
wire [15:0] temp2;
wire [15:0] temp3;
wire [15:0] temp4;
assign temp1 = a*b;
assign temp2 = c*d;

```

```
assign temp3 = e*f;
assign temp4 = g*h;
assign r1 = temp1 + temp2;
assign r2 = temp3 + temp4;
endmodule
```

### syn\_dspstyle=dsp48

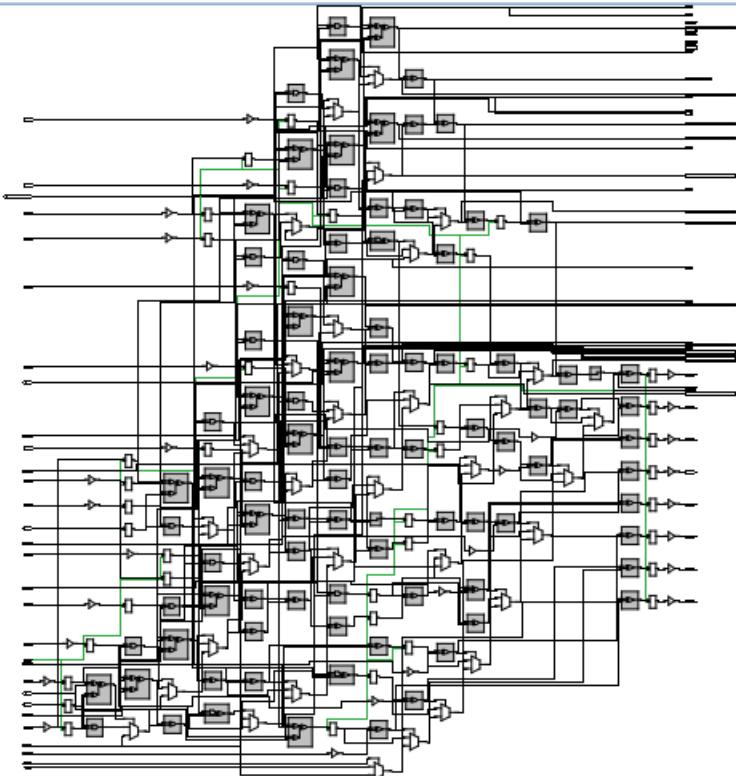
The following figure shows a multiplier, adder, and registers implemented in a DSP48 block, the default setting:



**syn\_DSPstyle=logic**

The next figure shows how the multipliers and adders are implemented when the attribute is set to logic:

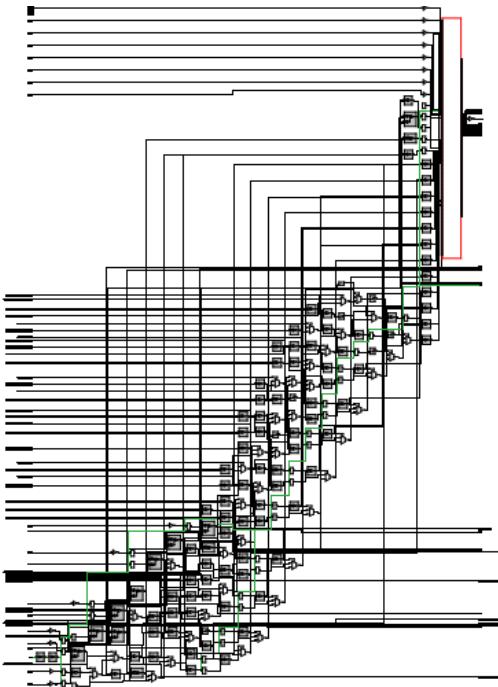
|         |                                                      |
|---------|------------------------------------------------------|
| Verilog | reg [16:0] m /* synthesis syn_DSPstyle = "logic" */; |
| VHDL    | attribute syn_DSPstyle of m : signal is "logic";     |



## Mixed Values for syn\_dspstyle

The following figure shows the implementation when syn\_dspstyle is set to logic for the multiplier and dsp48 for the adder:

|         |                                                                                                                             |
|---------|-----------------------------------------------------------------------------------------------------------------------------|
| Verilog | <pre>output reg [16:0] P /* synthesis syn_dspstyle = "dsp48" */; reg [16:0] m /* synthesis syn_dspstyle = "logic" */;</pre> |
| VHDL    | <pre>attribute syn_dspstyle of P : signal is "dsp48"; attribute syn_dspstyle of m : signal is "logic";</pre>                |



## syn\_dspstyle in SIMD Mode

Single Instruction Multiple Data (SIMD) mode is supported. By default, only one adder is packed into a single DSP slice. In SIMD mode, the DSP structure can support two 24-bit adders and four 12-bit adders in one DSP48 slice. To implement this DSP configuration, use the syn\_dspstyle attribute with one of the SIMD mode values, as described in [syn\\_dspstyle Values, on page 579](#). There are some limitations to DSP packing in SIMD mode; see [Limitations with SIMD Mode Packing, on page 587](#) for details.

The following Verilog syntax shows the same index set on two adders to map them into a single DSP slice:

```
wire [9:0] my_add1 /* synthesis syn_dspstyle = simd:1 */;
wire [9:0] my_add2 /* synthesis syn_dspstyle = simd:1 */;
```

See the examples below for more detail.

## Limitations with SIMD Mode Packing

DSP-packing in SIMD mode has the following limitations:

- Accumulators are packed only if they have the same reset, enable, and clock pins for the output registers.
- Add-Sub functions are packed only if they have the same select pin for dynamically choosing the adder or subtractor.
- Add-Mux functions are not supported with SIMD mode.

## Example 1 – Specifying SIMD

The following example shows four 12-bit adders specified with SIMD mapped to one DSP48 block.

```
module test(a_ip, b_ip, c_ip, d_ip, e_ip, f_ip, g_ip, h_ip,
            dout_op0, dout_op1, dout_op2, dout_op3);

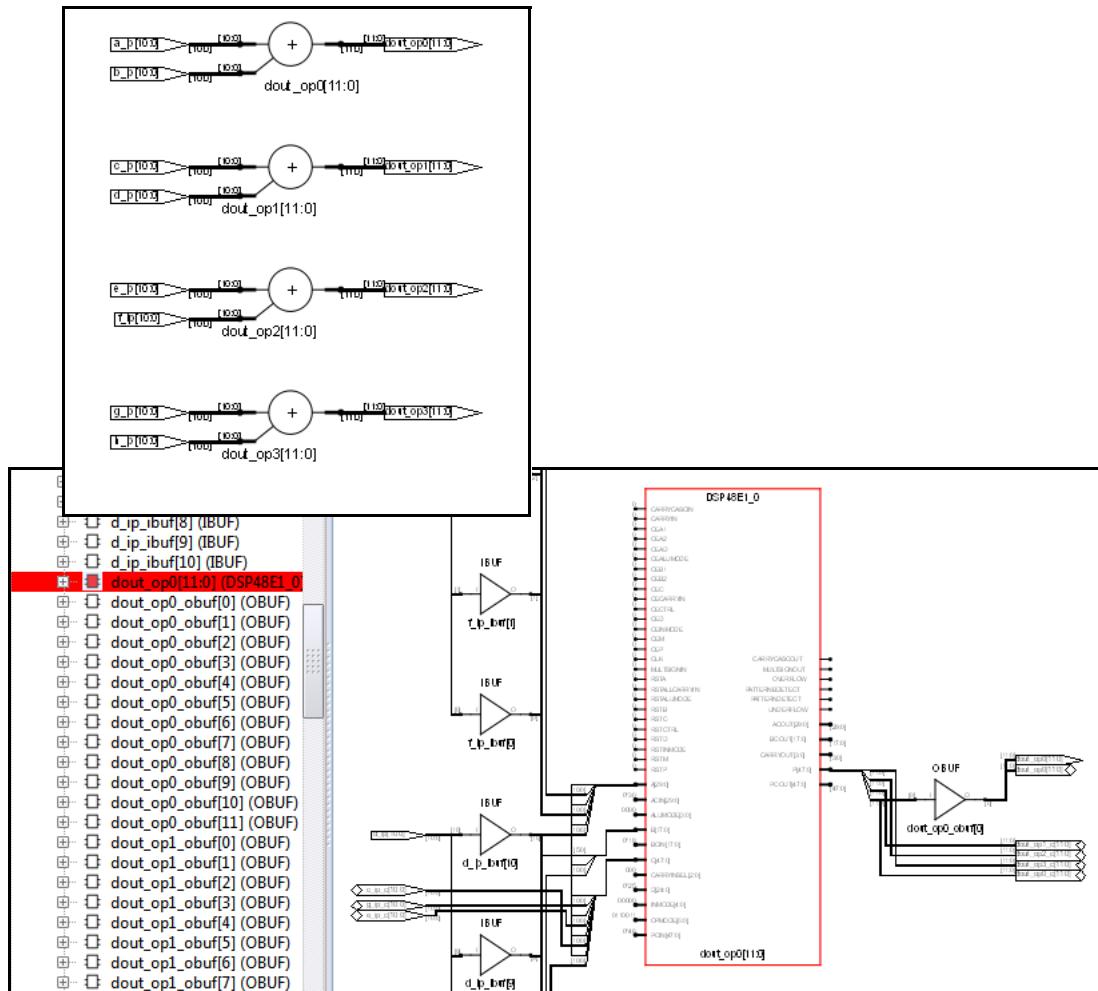
    //// Input declaration
    input [10 : 0] a_ip ;
    input [10 : 0] b_ip ;
    input [10 : 0] c_ip ;
    input [10 : 0] d_ip ;
    input [10 : 0] e_ip ;
    input [10 : 0] f_ip ;
    input [10 : 0] g_ip ;
    input [10 : 0] h_ip ;

    //// Output declaration
    output [11 : 0] dout_op0/* synthesis syn_dspstyle="simd" */;
    output [11 : 0] dout_op1/* synthesis syn_dspstyle="simd" */;
    output [11 : 0] dout_op2/* synthesis syn_dspstyle="simd" */;
    output [11 : 0] dout_op3/* synthesis syn_dspstyle="simd" */;
```

```

assign dout_op0 = a_ip + b_ip ;
assign dout_op1 = c_ip + d_ip ;
assign dout_op2 = e_ip + f_ip ;
assign dout_op3 = g_ip + h_ip ;
endmodule

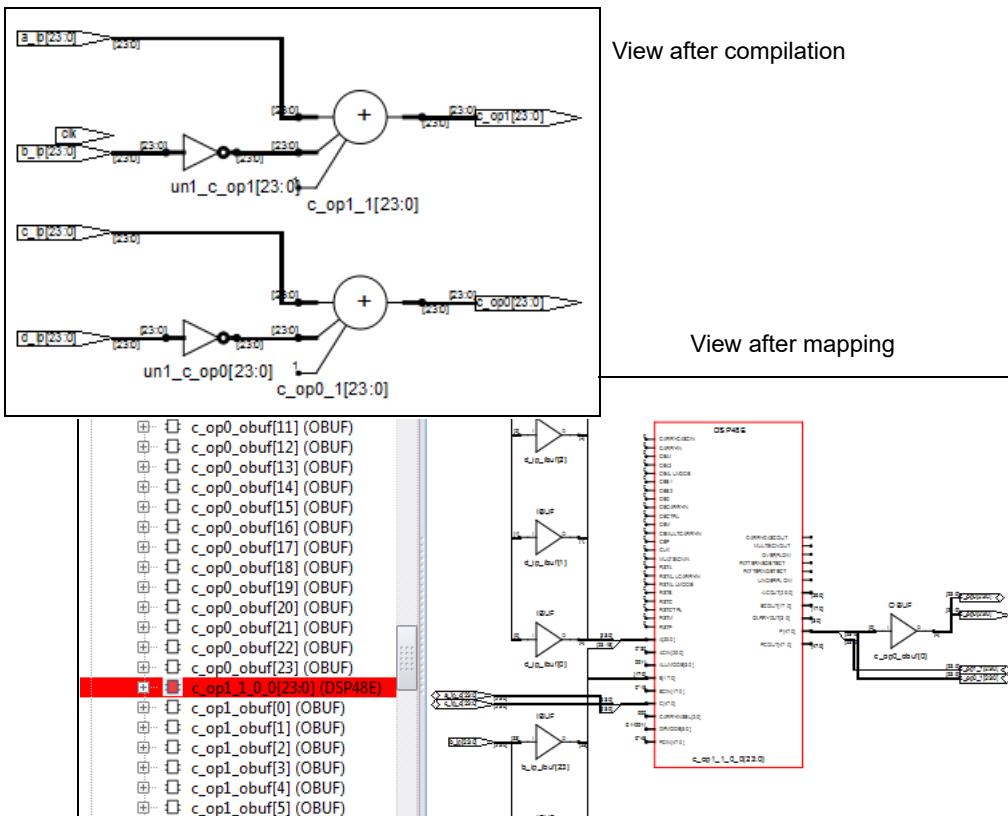
```



## Example 2 – Specifying a SIMD Index Value

The following example shows two 24-bit adders specified with a SIMD index value mapped to one DSP48 block.

```
module test(clk,a_ip,b_ip,c_ip,d_ip,c_op0,c_op1);  
  
    //// Input declaration  
    input clk;  
    input signed [23 : 0] a_ip;  
    input signed [23 : 0] b_ip;  
    input signed [23 : 0] c_ip;  
    input signed [23 : 0] d_ip;  
  
    ///Output declaration  
    output signed [23 : 0] c_op0/  
        /* synthesis syn_dspstyle = "simd:1" */;  
    output signed [23 : 0] c_op1  
        /* synthesis syn_dspstyle = "simd:1" */;  
  
    assign c_op0 = a_ip - b_ip ;  
    assign c_op1 = c_ip - d_ip ;  
endmodule
```



## syn\_edif\_bit\_format

### Attribute

Changes or preserves the naming style of bus ports in the EDIF output file.

### Description

`syn_edif_bit_format` is a character formatting attribute for vector (bus) port names that controls their naming style in the EDIF output netlist file. The first argument (%n, %d, or %u) controls the case of bus names, and the second argument (%i) controls the appearance of vector ranges. The default preserves the character case of names as they appear in source code with angle brackets (Verilog) or parentheses (VHDL) delineating the vector range. To change the character formatting of scalar ports, see [syn\\_edif\\_scalar\\_format, on page 594](#).

### syn\_edif\_bit\_format Syntax

| Constraint File | <code>define_global_attribute syn_edif_bit_format<br/>{[%n %u %d][<u>string</u>]({%i} [%i] &lt;%i&gt;})</code>               | Constraints File Example        |
|-----------------|------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| Verilog         | <code>object /* synthesis syn_edif_bit_format =<br/>"[%n %u %d] [<u>string</u>]({%i} [%i] &lt;%i&gt;)" */ ;</code>           | <a href="#">Verilog Example</a> |
| VHDL            | <code>attribute syn_edif_bit_format of object : objectType is<br/>"[%n %u %d][<u>string</u>]({%i} [%i] &lt;%i&gt;)" ;</code> | <a href="#">VHDL Example</a>    |

| Value                                       | Description                                                                                                                   |
|---------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <code>%n(%i)   %n[%i]   %n&lt;%i&gt;</code> | Preserves the name case (%n); parenthesis ((%i)), square brackets ([%i]), or angle brackets (<%i>) used as bus delimiters.    |
| <code>%u(%i)   %u[%i]   %u&lt;%i&gt;</code> | Shifts names to upper case (%u); parenthesis ((%i)), square brackets ([%i]), or angle brackets (<%i>) used as bus delimiters. |
| <code>%d(%i)   %d[%i]   %d&lt;%i&gt;</code> | Shifts names to lower case (%d); parenthesis ((%i)), square brackets ([%i]), or angle brackets (<%i>) used as bus delimiters. |

In addition to the above syntax definitions, %n, %u, and %d accept a `_string` argument to append a character string to the end of bus names.

## Constraints File Example

|   | Enable                              | Object Type | Object   | Attribute           | Value  | Value Type | Description      |
|---|-------------------------------------|-------------|----------|---------------------|--------|------------|------------------|
| 1 | <input checked="" type="checkbox"/> | global      | <Global> | syn_edif_bit_format | %u<%i> | string     | Format bus names |

## Verilog Example

```
module edif_bit_format(aBcd, clk, rst, q)
    /* synthesis syn_edif_bit_format ="%u<%i>" */;
    input[31:0]aBcd;
    input clk,rst;
    output reg[31:0]q;

    always@(posedge clk)
    begin
        if(rst)
            q<=0;
        else
            q<=aBcd;
    end
endmodule
```

## VHDL Example

```
library IEEE;
use IEEE.std_logic_1164.all;

entity d_ff_srss is
    port (aBcd: in std_logic_vector(31 downto 0);
          clk: in std_logic;
          q: out std_logic_vector(31 downto 0) );
end d_ff_srss;

architecture d_ff of d_ff_srss is
attribute syn_edif_bit_format : string ;
attribute syn_edif_bit_format of d_ff : architecture is "%u<%i>";
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            q <= aBcd;
        end if;
    end process;
end d_ff;
```

## Effect of Using syn\_edif\_bit\_format Attribute

The following examples illustrate the effects of using the syn\_edif\_bit\_format attribute on the EDIF output netlist.

### Example – Changing Bus Name Case and Delimiter

|                        |                                                                                                        |
|------------------------|--------------------------------------------------------------------------------------------------------|
| <b>VHD Source</b>      | input[31:0]aBcd; (Verilog)<br>port (aBcd: in std_logic_vector(31 downto 0); (VHDL)                     |
| <b>Attribute Value</b> | % u<%i><br>Converts bus names to all upper case (%u) and uses angle brackets for bus delimiters (<%>). |
| <b>EDIF Output</b>     | (port (array (rename aBcd "ABCD<31:0>") 32) (direction INPUT))                                         |

### Example – Appending Bus Name String

|                        |                                                                                                                                                |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>VHD Source</b>      | input[31:0]aBcd; (Verilog)<br>port (aBcd: in std_logic_vector(31 downto 0); (VHDL)                                                             |
| <b>Attribute Value</b> | % d_sub2A[%i]<br>Converts names to all lower case (%d), appends string (_s2A) to bus names, and uses square brackets for bus delimiters ([%]). |
| <b>EDIF Output</b>     | (port (array (rename aBcd "abcd_s2a[31:0]") 32) (direction INPUT))                                                                             |

## **syn\_edif\_scalar\_format**

### *Attribute*

Helps change the case of the object name in an EDIF file automatically.

### **Description**

Global character formatting attribute to use with scalar port names to up-shift (%u) or down-shift (%d) the character case of the name in the EDIF output file. The default (`syn_edif_scalar_format %n`) preserves the character case of the names as they appear in source code. To change the character formatting and style of vector (bus) signals and ports, see [syn\\_edif\\_bit\\_format](#).

### **syn\_edif\_scalar\_format Syntax**

The following table summarizes the syntax in different files:

| FDC     | <code>define_global_attribute syn_edif_scalar_format {%u %d %n}</code>            |
|---------|-----------------------------------------------------------------------------------|
| Verilog | <code>object /*synthesis syn_edif_scalar_format = " value" */;</code>             |
| VHDL    | <code>attribute syn_edif_scalar_format of object : objectType is " value";</code> |

### **Constraints Editor Example**

|   | Enabled                             | Object Type | Object   | Attribute              | Value | Val Type | Description         | Comment |
|---|-------------------------------------|-------------|----------|------------------------|-------|----------|---------------------|---------|
| 1 | <input checked="" type="checkbox"/> | global      | <global> | syn_edif_scalar_format | %u    | string   | Format scalar names |         |

### **Verilog Example**

```
module edif_case(aBcd,clk,q)
    /*synthesis syn_edif_scalar_format = " %u" */;
    input aBcd,clk;
    output reg q;

    always@ (posedge clk)
        q<=aBcd;
endmodule
```

## VHDL Example

```

library IEEE;
use IEEE.std_logic_1164.all;

entity d_ff_srss is
    port (aBcd,clk: in bit;
          q : out bit );
end d_ff_srss;

architecture d_ff of d_ff_srss is
attribute syn_edif_scalar_format : string ;
attribute syn_edif_scalar_format of d_ff : architecture is "%u";
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            q <= aBcd;
        end if;
    end process;
end d_ff;

```

## Effect of Using `syn_edif_scalar_format`

The following table shows the resulting netlist before applying the `syn_edif_scalar_format` attribute:

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Verilog | <pre>module edif_case(aBcd,clk,q) /*synthesis syn_edif_scalar_format=" %n"*/;</pre>                                                                                                                                                                                                                                                                                                                                                                                                       |
| VHDL    | <pre>attribute syn_edif_scalar_format : string ; attribute syn_edif_scalar_format of d_ff : architecture is " %n";</pre>                                                                                                                                                                                                                                                                                                                                                                  |
|         | <pre> (library work   (edifLevel 0)   (technology (numberDefinition ))   (cell edif_case (cellType GENERIC)     (view verilog (viewType NETLIST)       (interface         (port aBcd (direction INPUT))         (port clk (direction INPUT)            (port q (direction OUTPUT))           (net aBcd_c (joined             (portRef O (instanceRef aBcd_ibuf))             (portRef D (instanceRef qZ0)))))        )))     (net (rename abcd "aBcd") (joined       (portRef aBcd)</pre> |

```

        (portRef I (instanceRef aBcd_ibuf))
    ))
    (net clk_c (joined
    (portRef O (instanceRef clk_ibuf))
    (portRef C (instanceRef qZ0)))
))
    (net clk (joined
    (portRef clk)
    (portRef I (instanceRef clk_ibuf_iso)))
))
    (net q_c (joined
    (portRef Q (instanceRef qZ0))
    (portRef I (instanceRef q_obuf)))
))
    (net q (joined
    (portRef O (instanceRef q_obuf))
    (portRef q)
)))
    (net (rename clk_ibuf_isoZ0 "clk_ibuf_iso") (joined
    (portRef O (instanceRef clk_ibuf_iso))
    (portRef I (instanceRef clk_ibuf)))
))
)
)
(property mapper_option (string ""))
)

```

The following table shows the resulting netlist after applying the `syn_edif_scalar_format` attribute with the `%u` value:

---

**Verilog**    module edif\_case(aBcd,clk,q)  
/\*synthesis syn\_edif\_scalar\_format=%u\*/;

---

**VHDL**    attribute syn\_edif\_scalar\_format : string ;  
attribute syn\_edif\_scalar\_format of d\_ff : architecture is "%u";

---

```

(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell edif_case (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port ABCD (direction INPUT))
        (port CLK (direction INPUT))
        (port Q (direction OUTPUT))

        (net (rename aBcd_c "ABCD_C") (joined
          (portRef O (instanceRef aBcd_ibuf))
          (portRef D (instanceRef qZ0)))
        ))
      (net (rename abcd "ABCD") (joined
        (portRef ABCD)
        (portRef I (instanceRef aBcd_ibuf)))
    )))

```

```

        (net (rename clk_c "CLK_C") (joined
        (portRef O (instanceRef clk_ibuf))
        (portRef C (instanceRef qZ0)))
    ))
    (net (rename clk "CLK") (joined
    (portRef CLK)
    (portRef I (instanceRef clk_ibuf_iso)))
    ))
    (net (rename q_c "Q_C") (joined
    (portRef Q (instanceRef qZ0))
    (portRef I (instanceRef q_obuf)))
    ))
    (net (rename q "Q") (joined
    (portRef O (instanceRef q_obuf))
    (portRef Q)
    ))
    (net (rename clk_ibuf_isoZ0 "CLK_IBUF_ISO") (joined
    (portRef O (instanceRef clk_ibuf_iso))
    (portRef I (instanceRef clk_ibuf)))
    ))
)
(property mapper_option (string ""))
)

```

The following table shows the resulting netlist after applying the `syn_edif_scalar_format` attribute with the `%d` value:

|         |                                                                                                                                                           |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Verilog | <code>module edif_case(aBcd,clk,q)</code><br><code>/*synthesis syn_edif_scalar_format="%d"*/;</code>                                                      |
| VHDL    | <code>attribute syn_edif_scalar_format : string ;</code><br><code>attribute syn_edif_scalar_format of d_ff : architecture is</code><br><code>"%d";</code> |

```

(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell edif_case (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port abcd (direction INPUT))
        (port clk (direction INPUT))
        (port q (direction OUTPUT))

        (net (rename aBcd_c "abcd_c") (joined
        (portRef O (instanceRef aBcd_ibuf))
        (portRef D (instanceRef qZ0)))
    ))
    (net abcd (joined
    (portRef abcd)
    (portRef I (instanceRef aBcd_ibuf)))
  )
    (net clk_c (joined
    (portRef O (instanceRef clk_ibuf)))
  )

```

```
        (portRef C (instanceRef qZ0))
    ))
    (net clk (joined
    (portRef clk)
    (portRef I (instanceRef clk_ibuf_iso)))
))
    (net q_c (joined
    (portRef Q (instanceRef qZ0))
    (portRef I (instanceRef q_obuf)))
))
    (net q (joined
    (portRef O (instanceRef q_obuf))
    (portRef q)
))
    (net (rename clk_ibuf_isoz0 "clk_ibuf_iso") (joined
    (portRef O (instanceRef clk_ibuf_iso))
    (portRef I (instanceRef clk_ibuf)))
))
)
(property mapper_option (string ""))
)
```

## Global Support

If a design has more than one EDIF netlist, apply this attribute only on the top level netlist. It has the same impact as when it is applied on individual netlists. For example, if two modules have been instantiated, the attribute should be applied only on the top-level module.

## Verilog Example

```
module mux(out, a, b, sel); // mux
    output out;
    input a, b;
    input sel;
    assign out = sel ? a : b;
endmodule

module reg8(q, data, clk, rst); // register
    output q;
    input data;
    input clk, rst;
    reg q;
```

```

always @(posedge clk or posedge rst)
begin
    if (rst)
        q = 0;
    else
        q = data;
end
endmodule

module top1(q1, a, b, sel, clk, rst)
    /*synthesis syn_edif_scalar_format="%u"*/;
output q1;
input a, b;
input sel, clk, rst;
wire mux_out;
mux mux_1 (mux_out, a, b, sel);
reg8 reg8_1 (q1, mux_out, clk, rst);
endmodule

```

The following table shows the resulting netlist with the `syn_edif_scalar_format` attribute applied to the top-level module.

|          |                                                                                                 |
|----------|-------------------------------------------------------------------------------------------------|
| FDC File | <code>define_global_attribute {syn_edif_scalar_format} {%u}</code>                              |
| FDC File | <code>module top1(q1, a, b, sel, clk, rst)/*synthesis<br/>syn_edif_scalar_format="%u"*/;</code> |

```

(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell reg8 (cellType GENERIC)
    (view netlist (viewType NETLIST)
      (interface
        (port RST_C (direction INPUT))
        (port CLK_C (direction INPUT))
        (port MUX_OUT (direction INPUT))
        (port Q1_C (direction OUTPUT))
      )
      (contents
        (instance q (viewRef PRIM (cellRef FDC (
          libraryRef UNILIB)))
        )
        (net (rename q1_c "Q1_C") (joined
          (portRef Q (instanceRef q))
          (portRef Q1_C)
        ))
        (net (rename mux_out "MUX_OUT") (joined
          (portRef MUX_OUT)
          (portRef D (InstanceRef q))
        ))
        (net (rename clk_c "CLK_C") (joined

```



```
(cellRef IBUFG (libraryRef VIRTEX)))
)
(instance q1_obuf (viewRef PRIM
    (cellRef OBUF (libraryRef VIRTEX)))
)
(instance rst_ibuf (viewRef PRIM
    (cellRef IBUF (libraryRef VIRTEX)))
)
(instance sel_ibuf (viewRef PRIM
    (cellRef IBUF (libraryRef VIRTEX)))
)
(instance b_ibuf (viewRef PRIM
    (cellRef IBUF (libraryRef VIRTEX)))
)
(instance a_ibuf (viewRef PRIM (cellRef IBUF
    (libraryRef VIRTEX)))
)
(instance mux_1 (viewRef netlist (cellRef mux)))
)
(instance reg8_1 (viewRef netlist (cellRef reg8)))
)
(net (rename mux_out "MUX_OUT") (joined
(portRef MUX_OUT (instanceRef mux_1))
(portRef MUX_OUT (instanceRef reg8_1)))
))
(net (rename a_c "A_C") (joined
(portRef O (instanceRef a_ibuf))
(portRef A_C (instanceRef mux_1)))
)
(net (rename a "A") (joined
(portRef A)
(portRef I (instanceRef a_ibuf)))
))
(net (rename b_c "B_C") (joined
(portRef O (instanceRef b_ibuf))
(portRef B_C (instanceRef mux_1)))
))
(net (rename b "B") (joined
(portRef B)
(portRef I (instanceRef b_ibuf)))
))
(net (rename sel_c "SEL_C") (joined
(portRef O (instanceRef sel_ibuf))
(portRef SEL_C (instanceRef mux_1)))
))
(net (rename sel "SEL") (joined
(portRef SEL)
(portRef I (instanceRef sel_ibuf)))
))
(net (rename clk_c "CLK_C") (joined
(portRef O (instanceRef clk_ibuf))
(portRef CLK_C (instanceRef reg8_1)))
))
(net (rename clk "CLK") (joined
(portRef CLK)
(portRef I (instanceRef clk_ibuf_iso)))
))
```

```
(net (rename rst_c "RST_C") (joined
  (portRef O (instanceRef rst_ibuf))
  (portRef RST_C (instanceRef reg8_1)))
  )
  (net (rename rst "RST") (joined
    (portRef RST)
    (portRef I (instanceRef rst_ibuf)))
  )
  (net (rename q1_c "Q1_C") (joined
    (portRef Q1_C (instanceRef reg8_1))
    (portRef I (instanceRef q1obuf)))
  )
  (net (rename q1 "Q1") (joined
    (portRef O (instanceRef q1obuf))
    (portRef Q1)))
  )
  (net (rename clk_ibuf_isoz0 "CLK_IBUF_ISO") (joined
    (portRef O (instanceRef clk_ibuf_isoz0))
    (portRef I (instanceRef clk_ibuf)))
  )
)
(property mapper_option (string ""))
```

## **syn\_encoding**

### *Attribute*

Overrides the default FSM Compiler encoding for a state machine and applies the specified encoding.

### **syn\_encoding Values**

The default is that the tool automatically picks an encoding style that results in the best performance. To ensure that a particular encoding style is used, explicitly specify that style, using the values below:

| <b>Value</b> | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| onehot       | <p>Only two bits of the state register change (one goes to 0, one goes to 1) and only one of the state registers is hot (driven by 1) at a time. For example:</p> <p>0001, 0010, 0100, 1000</p> <p>Because onehot is not a simple encoding (more than one bit can be set), the value must be decoded to determine the state. This encoding style can be slower than a gray style if you have a large output decoder following a state machine.</p>                                                                                                 |
| gray         | <p>More than one of the state registers can be hot. The synthesis tool attempts to have only one bit of the state registers change at a time, but it can allow more than one bit to change, depending upon certain conditions for optimization. For example:</p> <p>000, 001, 011, 010, 110</p> <p>Because gray is not a simple encoding (more than one bit can be set), the value must be decoded to determine the state. This encoding style can be faster than a onehot style if you have a large output decoder following a state machine.</p> |
| sequential   | <p>More than one bit of the state register can be hot. The synthesis tool makes no attempt at limiting the number of bits that can change at a time. For example:</p> <p>000, 001, 010, 011, 100</p> <p>This is one of the smallest encoding styles, so it is often used when area is a concern. Because more than one bit can be set (1), the value must be decoded to determine the state. This encoding style can be faster than a onehot style if you have a large output decoder following a state machine.</p>                               |

| Value    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| safe     | <p>This implements the state machine in the default encoding and adds reset logic to force the state machine to a known state if it reaches an invalid state. This value can be used in combination with any of the other encoding styles described above. You specify safe before the encoding style. The safe value is only valid for a state register, in conjunction with an encoding style specification.</p> <ul style="list-style-type: none"> <li>For example, if the default encoding is onehot and the state machine reaches a state where all the bits are 0, which is an invalid state, the safe value ensures that the state machine is reset to a valid state.</li> <li>If recovery from an invalid state is a concern, it may be appropriate to use this encoding style, in conjunction with onehot, sequential or gray, in order to force the state machine to reset. When you specify safe, the state machine can be reset from an unknown state to its reset state.</li> <li>If an FSM with asynchronous reset is specified with the value safe and you do not want the additional recovery logic (flip-flop on the inactive clock edge) inserted for this FSM, then use the <a href="#">syn_shift_resetphase</a>, on page 801 for details.</li> </ul> |
| original | This respects the encoding you set, but the software still does state machine and reachability analysis.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

You can specify multiple values. This snippet uses safe,gray. The encoding style for register OUT is set to gray, but if the state machine reaches an invalid state the synthesis tool will reset the values to a valid state.

```
module prep3 (CLK, RST, IN, OUT);
  input CLK, RST;
  input [7:0] IN;
  output [7:0] OUT;
  reg [7:0] OUT;
  reg [7:0] current_state /* synthesis syn_encoding="safe,gray" */;
  // Other code
```

## Description

This attribute takes effect only when FSM Compiler is enabled. It overrides the default FSM Compiler encoding for a state machine. For the specified encoding to take effect, the design must contain state machines that have been inferred by the FSM Compiler. Setting this attribute when `syn_state_machine` is set to 0 will not have any effect.

The default encoding style automatically assigns encoding based on the number of states in the state machine. Use the `syn_encoding` attribute when you want to override these defaults. You can also use `syn_encoding` when you want to disable the FSM Compiler globally but there are a select number of state registers in your design that you want extracted. In this case, use this attribute with the `syn_state_machine` directive on for just those specific registers.

The encoding specified by this attribute applies to the final mapped netlist. For other kinds of enumerated encoding, use `syn_enum_encoding`. See [syn\\_enum\\_encoding, on page 611](#) and [syn\\_encoding Compared to syn\\_enum\\_encoding, on page 613](#) for more information.

## Encoding Style Implementation

The encoding style is implemented during the mapping phase. A message appears when the synthesis tool extracts a state machine, for example:

```
@N: CL201 : "c:\design\..."|Trying to extract state machine for
register current_state
```

The log file reports the encoding styles used for the state machines in your design. This information is also available in the FSM Viewer.

See also the following:

- For information on enabling state machine optimization for individual modules, see [syn\\_state\\_machine, on page 815](#).
- For VHDL designs, see [syn\\_encoding Compared to syn\\_enum\\_encoding, on page 613](#) for comparative usage information.

## Syntax Specification

This table shows how to specify the attribute in different files:

|         |                                                                        |                                            |
|---------|------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute {object} syn_encoding {value}</code>            | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_encoding = "value" */;</code>            | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_encoding of object : objectType is "value";</code> | <a href="#">VHDL Example</a>               |

If you specify the `syn_encoding` attribute in Verilog or VHDL, all instances of that FSM use the same `syn_encoding` value. To have unique `syn_encoding` values for each FSM instance, use different entities or modules, or specify the `syn_encoding` attribute in a constraint file.

## Constraints Editor Example

|   | Enabled                             | Object Type | Object       | Attribute    | Value | Val Type | Description                                             |
|---|-------------------------------------|-------------|--------------|--------------|-------|----------|---------------------------------------------------------|
| 1 | <input checked="" type="checkbox"/> | fsm         | i:state[3:0] | syn_encoding | gray  | string   | FSM encoding (onehot, sequential, gray, original, safe) |

The *object* must be an instance prefixed with `i:`, as in `i:instance`. The instance must be a sequential instance with a view name of statemachine.

## Verilog Example

The object can be a register definition signals that hold the state values of state machines.

```
module fsm (clk, reset, x1, outp);
    input clk, reset, x1;
    output outp;
    reg outp;
    reg [1:0] state /* synthesis syn_encoding = "onehot" */;
    parameter s1 = 2'b00; parameter s2 = 2'b01;
    parameter s3 = 2'b10; parameter s4 = 2'b11;

    always @ (posedge clk or posedge reset)
    begin
        if (reset)
            state <= s1;
        else begin
            case (state)
                s1: if (x1 == 1'b1)
                    state <= s2;
                else
                    state <= s3;
                s2: state <= s4;
                s3: state <= s4;
                s4: state <= s1;
            endcase
        end
    end
end
```

```
always @(state) begin
    case (state)
        s1: outp = 1'b1;
        s2: outp = 1'b1;
        s3: outp = 1'b0;
        s4: outp = 1'b0;
    endcase
end
endmodule
```

## VHDL Example

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

entity fsm is
    port (x1 : in std_logic;
          reset : in std_logic;
          clk : in std_logic;
          outp : out std_logic );
end fsm;

architecture rtl of fsm is
signal state : std_logic_vector(1 downto 0);
constant s1 : std_logic_vector := "00";
constant s2 : std_logic_vector := "01";
constant s3 : std_logic_vector := "10";
constant s4 : std_logic_vector := "11";
attribute syn_encoding : string;
attribute syn_encoding of state : signal is "onehot";
begin
process (clk,reset)
begin
    if (clk'event and clk = '1') then
        if (reset = '1') then
            state <= s1 ;
        else
            case state is
                when s1 =>
                    if x1 = '1' then
                        state <= s2;
                    else
                        state <= s3;
                    end if;
                when s2 =>
```

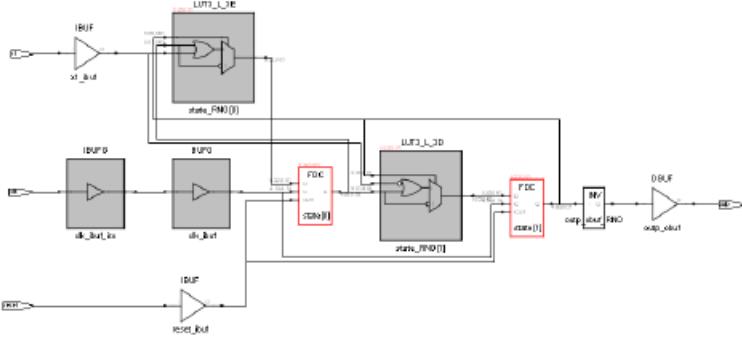
```
        state <= s4;
        when s3 =>
            state <= s4;
        when s4 =>
            state <= s1;
        end case;
    end if;
end if;
end process;

process (state)
begin
    case state is
        when s1 =>
            outp <= '1';
        when s2 =>
            outp <= '1';
        when s3 =>
            outp <= '0';
        when s4 =>
            outp <= '0';
        end case;
    end process;
end rtl;
```

## Effect of Using `syn_encoding`

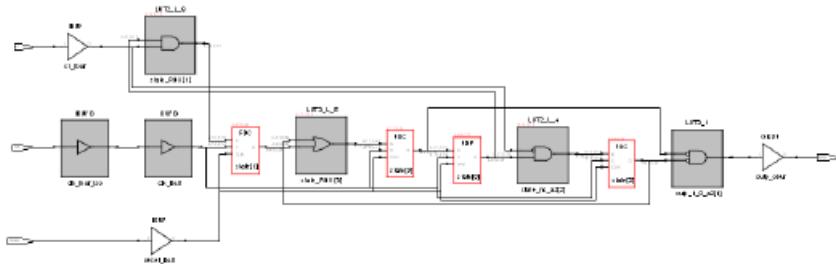
The following figure shows the default implementation of a state machine, with these encoding details reported:

```
Encoding state machine state [3:0] (netlist: statemachine)
original code -> new code
 00 -> 00
 01 -> 01
 10 -> 10
 11 -> 11
```



The next figure shows the state machine when the `syn_encoding` attribute is set to `onehot`, and the accompanying changes in the code:

|         |                                                                       |
|---------|-----------------------------------------------------------------------|
| Verilog | <code>reg [1:0] state /* synthesis syn_encoding = "onehot" */;</code> |
| VHDL    | <code>attribute syn_encoding of state : signal is "onehot";</code>    |



Encoding state machine state [3:0] (netlist: statemachine)

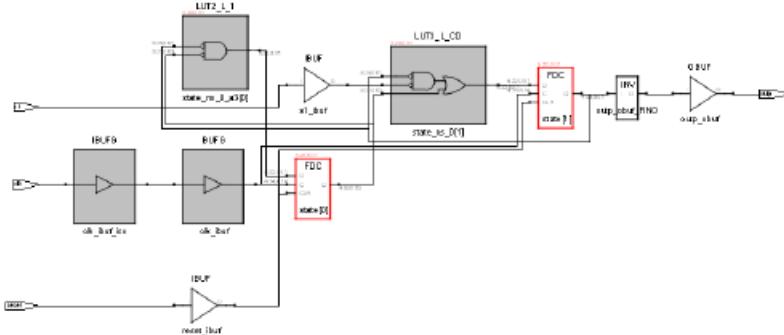
```

00 -> 0001
01 -> 0010
10 -> 0100
11 -> 1000

```

The next figure shows the state machine when the syn\_encoding attribute is set to gray:

|         |                                                                     |
|---------|---------------------------------------------------------------------|
| Verilog | <code>reg [1:0] state /* synthesis syn_encoding = "gray" */;</code> |
| VHDL    | <code>attribute syn_encoding of state : signal is "gray";</code>    |



Encoding state machine state [3:0] (netlist: statemachine)

```

00 -> 00
00 -> 01
10 -> 11
11 -> 10

```

## **syn\_enum\_encoding**

*Directive*

For VHDL designs. Defines how enumerated data types are implemented. The type of implementation affects the performance and device utilization.

### **syn\_enum\_encoding Values**

| <b>Value</b> | <b>Description</b>                                                                                                                                                                              |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| default      | Automatically assigns an encoding style that results in the best performance.                                                                                                                   |
| sequential   | More than one bit of the state register can change at a time, but because more than one bit can be hot, the value must be decoded to determine the state. For example: 000, 001, 010, 011, 100. |
| onehot       | Only two bits of the state register change (one goes to 0; one goes to 1) and only one of the state registers is hot (driven by a 1) at a time. For example: 0000, 0001, 0010, 0100, 1000.      |
| gray         | Only one bit of the state register changes at a time, but because more than one bit can be hot, the value must be decoded to determine the state. For example: 000, 001, 011, 010, 110.         |
| string       | This can be any value you define. For example: 001, 010, 101. See <a href="#"><i>Example of syn_enum_encoding for User-Defined Encoding , on page 613.</i></a>                                  |

### **Description**

If FSM Compiler is enabled, this directive has no effect on the encoding styles of extracted state machines; the tool uses the values specified in the `syn_encoding` attribute instead.

However, if you have enumerated data types and you turn off the FSM Compiler so that no state machines are extracted, the `syn_enum_encoding` style is implemented in the final circuit. See [\*syn\\_encoding Compared to syn\\_enum\\_encoding, on page 613\*](#) for more information.

A message appears in the log file when you use the `syn_enum_encoding` directive; for example:

```
CD231: Using onehot encoding for type mytype (red="10000000")
```

When using an application such as an equivalence checker, the encoding value automatically reverts to the sequential standard interpretation for the enumerations. Using a value other than sequential cannot guarantee that the application will use the same value. A message (CD233) is written to the log file as notification of the value change.

## **syn\_enum\_encoding, enum\_encoding, and syn\_encoding**

Custom attributes are attributes that are not defined in the IEEE specifications, but which you or a tool vendor define for your own use. They provide a convenient back door in VHDL, and are used to better control the synthesis and simulation process. `enum_encoding` is one of these custom attributes that is widely used to allow specific binary encodings to be attached to objects of enumerated types.

The `enum_encoding` attribute is declared as follows:

```
attribute enum_encoding: string;
```

This can be either written directly in your VHDL design description, or provided to you by the tool vendor in a package. Once the attribute has been declared and given a name, it can be referenced as needed in the design description:

```
type statevalue is (INIT, IDLE, READ, WRITE, ERROR);
attribute enum_encoding of statevalue: type is
    "000 001 011 010 110";
```

When this is processed by a tool that supports the `enum_encoding` attribute, it uses the information about the `statevalue` encoding. Tools that do not recognize the `enum_encoding` attribute ignore the encoding.

Although it is recommended that you use `syn_enum_encoding`, `enum_encoding` is recognized and treated like `syn_enum_encoding`. The tool uses the specified encoding when the FSM compiler is disabled, and ignores the value when the FSM Compiler is enabled.

If `enum_encoding` and `syn_encoding` are both defined and the FSM compiler is enabled, the tool uses the value of `syn_encoding`. If you have both `syn_enum_encoding` and `enum_encoding` defined, the value of `syn_enum_encoding` prevails.

## syn\_encoding Compared to syn\_enum\_encoding

To implement a state machine with a particular encoding style when the FSM Compiler is enabled, use the `syn_encoding` attribute. The `syn_encoding` attribute affects how the technology mapper implements state machines in the final netlist. The `syn_enum_encoding` directive only affects how the compiler interprets the associated enumerated data types. Therefore, the encoding defined by `syn_enum_encoding` is *not propagated* to the implementation of the state machine. However, when FSM Compiler is disabled, the value of `syn_enum_encoding` is implemented in the final circuit.

## Example of syn\_enum\_encoding for User-Defined Encoding

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

entity shift_enum is
    port (clk, rst : bit;
          O : out std_logic_vector(2 downto 0) );
end shift_enum;

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "001 010 101";
signal machine : state_type;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            machine <= S0;
        elsif clk = '1' and clk'event then
            case machine is
                when S0 => machine <= S1;
                when S1 => machine <= S2;
                when S2 => machine <= S0;
            end case;
        end if;
    end process;
    with machine select
        O <= "001" when S0,
        "010" when S1,
        "101" when S2;
    end behave;
```

## **syn\_enum\_encoding Values Syntax**

This table summarizes the syntax in the following file type:

VHDL **attribute syn\_enum\_encoding of object : objectType is "value";** [VHDL Example](#)

---

### **VHDL Example**

Here is the code used to generate the second schematic in the previous figure. (The first schematic will be generated instead, if sequential is replaced by onehot as the syn\_enum\_encoding value.)

```
package testpkg is
    type mytype is (red, yellow, blue, green, white,
                    violet, indigo, orange);
    attribute syn_enum_encoding : string;
    attribute syn_enum_encoding of mytype : type is "sequential";
end package testpkg;

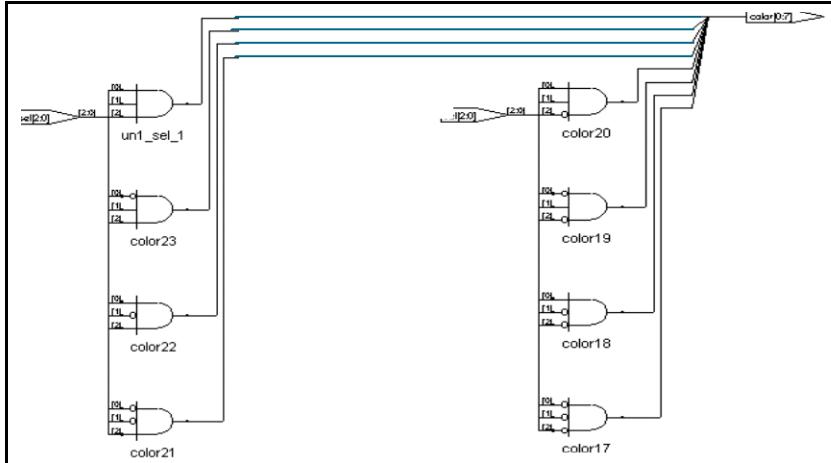
library IEEE;
use IEEE.std_logic_1164.all;
use work.testpkg.all;

entity decoder is
    port (sel : in std_logic_vector(2 downto 0);
          color : out mytype );
end decoder;

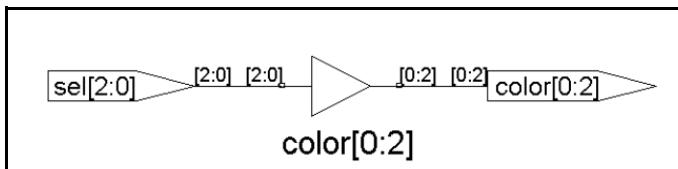
architecture rtl of decoder is
begin
    process(sel)
    begin
        case sel is
            when "000" => color <= red;
            when "001" => color <= yellow;
            when "010" => color <= blue;
            when "011" => color <= green;
            when "100" => color <= white;
            when "101" => color <= violet;
            when "110" => color <= indigo;
            when others => color <= orange;
        end case;
    end process;
end rtl;
```

## Effect of Encoding Styles

The following figure provides an example of two versions of a design: one with the default encoding style, the other with the `syn_enum_encoding` directive overriding the default enumerated data types that define a set of eight colors.



`syn_enum_encoding = "default"` Based on 8 states, onehot assigned



`syn_enum_encoding = "sequential"`

In this example, using the default value for `syn_enum_encoding`, onehot is assigned because there are eight states in this design. The onehot style implements the output color as 8 bits wide and creates decode logic to convert the input `sel` to the output. Using sequential for `syn_enum_encoding`, the logic is reduced to a buffer. The size of output color is 3 bits.

## **syn\_fast\_auto**

*Attribute.*

Speeds up signal transitions on output ports.

### Description

The synthesis tool infers the fast output buffers OBUF\_F\_24, OBUFT\_F\_24, and IOBUF\_F\_24. Use the `syn_fast_auto` global attribute to force inference of slow buffers. In the HDL source code, specify `syn_fast_auto` for the top-level module or architecture.

You can override this attribute on an individual basis by using the `syn_padtype` attribute. The `syn_fast_auto` attribute has no effect on output ports specified with the `syn_padtype` attribute.

### **syn\_fast\_auto Syntax**

The following table summarizes the syntax in different files:

FDC           **define\_global\_attribute syn\_fast\_auto {0|1}**

Verilog       **object /\*synthesis syn\_fast\_auto = 0|1 \*/;**

VHDL         **attribute syn\_fast\_auto : boolean;**  
**attribute syn\_fast\_auto of object : objectType is 0|1;**

### Constraint Editor Example

|   | Enabled                             | Object Type | Object   | Attribute    | Value | Val Type | Description                             |
|---|-------------------------------------|-------------|----------|--------------|-------|----------|-----------------------------------------|
| 1 | <input checked="" type="checkbox"/> | global      | <global> | xc_fast_auto | 1     | boolean  | Enable automatic fast output buffer use |

### Verilog Example

```
module ramtest (q,d,addr,we,clk) /*synthesis syn_fast_auto=0 */;
  input we,clk;
  input [2:0]addr;
  input [3:0]d;
  output reg [3:0]q;
  reg [3:0] mem [7:0];
```

```
always @(posedge clk)
begin
    q <= mem[addr];
    if (we == 1)
        mem[addr] <= d;
    end
endmodule
```

## VHDL Example

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

entity ramtest is
port (q : out std_logic_vector(3 downto 0);
      d : in std_logic_vector(3 downto 0);
      addr : in std_logic_vector(2 downto 0);
      we : in std_logic;
      clk : in std_logic );
end ramtest;

architecture rtl of ramtest is
attribute syn_fast_auto : boolean;
attribute syn_fast_auto of rtl : architecture is false;

type mem_type is array (7 downto 0) of std_logic_vector
  3 downto 0);
signal mem : mem_type;
begin
q <= mem(conv_integer(addr));
process (clk) begin
  if rising_edge(clk) then
    if (we = '1') then
      mem(conv_integer(addr)) <= d;
    end if;
  end if;
end process;
end rtl;
```

## Effect of Using syn\_fast\_auto

The following table shows a design before applying the syn\_fast\_auto attribute.

Verilog      No attribute statement.

VHDL      No attribute statement.

Constraint    Attribute definition disabled:

|   | Enabled                             | Object Type | Object   | Attribute    | Value | Val Type | Description                             |
|---|-------------------------------------|-------------|----------|--------------|-------|----------|-----------------------------------------|
| 1 | <input checked="" type="checkbox"/> | global      | <global> | xc_fast_auto | 1     | boolean  | Enable automatic fast output buffer use |

Check the following lines in the generated \*.edf file:

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell ramtest (cellType GENERIC)
    (view rtl (viewType NELLIST)
      (
        .....
        .....
        .....
        (property mapper_option (string ""))
      )
    )
  (design ramtest (cellRef ramtest (libraryRef work)
    (property mapper_option (string ""))
    (property PART (string "xc5vlx20tff323-1") (owner "Xilinx")))
  )
)
```

The following table shows the same design after applying the syn\_fast\_auto attribute.

Verilog      module ramtest (q,d,addr,we,clk) /\*synthesis syn\_fast\_auto=1 \*/;

VHDL      `attribute syn_fast_auto : boolean;`  
`attribute syn_fast_auto of rtl : architecture is true;`

#### Constraint

|   | Enabled                             | Object Type | Object   | Attribute    | Value | Val Type | Description                             |
|---|-------------------------------------|-------------|----------|--------------|-------|----------|-----------------------------------------|
| 1 | <input checked="" type="checkbox"/> | global      | <global> | xc_fast_auto | 1     | boolean  | Enable automatic fast output buffer use |

Search for the string "syn\_fast\_auto" in the \*.edf file.

```
(library work
  (ediLevel 0)
  (technology (numberDefinition ))
  (cell ramtest (cellType GENERIC)
    (view rtl (viewType NETLIST)
      (
        -----
        -----
        (property mapper_option (string ""))
        (property syn_fast_auto (integer 1))
      )
    )
  (design ramtest (cellRef ramtest (libraryRef work)
    (property mapper_option (string ""))
    (property PART (string "xc5vlx20tff323-1") (owner "Xilinx")))
  )
)
```

The following table shows a design with the value of syn\_fast\_auto attribute set to false.

Verilog      `module ramtest (q,d,addr,we,clk) /*synthesis syn_fast_auto=0 */;`

VHDL      `attribute syn_fast_auto : boolean;`  
`attribute syn_fast_auto of rtl : architecture is false;`

#### Constraint

|   | Enabled                             | Object Type | Object   | Attribute    | Value | Val Type | Description                             |
|---|-------------------------------------|-------------|----------|--------------|-------|----------|-----------------------------------------|
| 1 | <input checked="" type="checkbox"/> | global      | <global> | xc_fast_auto | 0     | boolean  | Enable automatic fast output buffer use |

```

(library work
  (ediLevel 0)
  (technology (numberDefinition ))
  (cell ramtest (cellType GENERIC)
    (view rtl (viewType NETLIST)
      (
        -----
        -----
        (property mapper_option (string ""))
        (property syn_fast_auto (integer 0))
      )
    )
  (design ramtest (cellRef ramtest (libraryRef work)
    (property mapper_option (string ""))
    (property PART (string "xc5vlx20tf323-1") (owner "Xilinx"))
  )
)

```

The 'FAST' keyword is not forward annotated to any of the place-and-route files. The following files have forward annotated "SLOW" keyword attached to output ports in the design.

In the file, "...pr\_1\ramtest\_map.mrp", the following statement appears:

INFO:LIT:244—All the single ended outputs in this design use slew rate limited output drivers. The delay on speed critical single ended outputs can be dramatically reduced by designating them as fast outputs.

```

..\rev_1\pr_1\ramtest_map.xrpt(281): <item label="Slew&#xA;Rate" stringID="SLEW_RATE" value="SLOW" />
..\rev_1\pr_1\ramtest_pad.csv(274):R1,q[2],IOB,IO_L0P_CC_RS1_2,OUTPUT,LVCMOS25*,2,12,SLOW,,,UNLOCATE
D,NO,NONE,
..\rev_1\pr_1\ramtest_pad(274):
R1|q[2]|IOB|IO_L0P_CC_RS1_2|OUTPUT|LVCMOS25*|2|12|SLOW|||UNLOCATED|NO|NONE|


..\rev_1\pr_1\ramtest_map.mrp(128): | q[0] | IOB | OUTPUT | LVCMOS25 | | 12 | SLOW | | |
..\rev_1\pr_1\ramtest_pad.txt(259): |P3|q[1]|IOB|IO_L2N_A22_2|OUTPUT|LVCMOS25*|2|12|SLOW|||
|UNLOCATED|NO|NONE|
..\rev_1\pr_1\ramtest_pad(258):P3|q[1]|IOB|IO_L2N_A22_2|OUTPUT|LVCMOS25*|2|12|SLOW|||UNLOCATED|NO|NO
|NE|

```

The attribute can also be applied individually in the FDC file as follows:

```
define_attribute {p:q[3:0]} {xc_fast_auto} {0}
```

## Global Support

The `syn_fast_auto` attribute is defined globally by default as shown in the above examples and can be overridden individually by using the `syn_pad_type` attribute.

The following example shows the `syn_fast_auto` attribute when applied on collections.

```
FDC      define_scope_collection output_bufs {q[3:0]}
          define_attribute {$output_bufs} {xc_fast_auto} {0}
```

### Constraint

Collections Tab:

|   | Enabled                             | Collection Name | Command | Command Arguments |
|---|-------------------------------------|-----------------|---------|-------------------|
| 1 | <input checked="" type="checkbox"/> | output_bufs     | q[3:0]  | q[3:0]            |

Attributes Tab:

|   | Enabled                             | Object Type | Object        | Attribute    | Value | Val Type | Description                             |
|---|-------------------------------------|-------------|---------------|--------------|-------|----------|-----------------------------------------|
| 1 | <input checked="" type="checkbox"/> | input_port  | \$output_bufs | xc_fast_auto | 0     | boolean  | Enable automatic fast output buffer use |

The following lines can be seen in the log file when the attribute is applied using collections.

```
-----+
Adding property xc_fast_auto, value 0, to inst q[3:0]
Adding property xc_fast_auto, value 0, to port q[3:0]
Adding property xc_fast_auto, value 0, to net q[0]
Adding property xc_fast_auto, value 0, to net q[1]
Adding property xc_fast_auto, value 0, to net q[2]
Adding property xc_fast_auto, value 0, to net q[3]
-----+
```

## **syn\_force\_seq\_prim**

### *Directive*

Applies the “fix gated clocks” algorithm to the associated primitive.

To use the `syn_force_seq_prim` directive with a black box, you must also identify the clock signal using the `syn_isclock` directive and the enable signal using the `syn_gatedclk_clock_en` directive. The data type is Boolean.

The `syn_force_seq_prim` directive is one of several directives that you can use with the `syn_black_box` directive to define timing for a black box. See [syn\\_black\\_box, on page 526](#) for a list of the associated directives.

### **syn\_force\_seq\_prim Syntax**

|         |                                                                                          |                                 |
|---------|------------------------------------------------------------------------------------------|---------------------------------|
| Verilog | <code>object /* synthesis syn_forward_io_constraints = 1 0 */;</code>                    | <a href="#">Verilog Example</a> |
| VHDL    | <code>attribute syn_forward_io_constraints of object : objectType is true false ;</code> | <a href="#">VHDL Example</a>    |

In the above syntax, `object` is the module name of the black box.

### **Verilog Example**

```
module bbe (ena, clk, data_in, data_out)
    /* synthesis syn_black_box */
    /* synthesis syn_force_seq_prim=1 */;
    input clk /* synthesis syn_isclock = 1 */;
    /* synthesis syn_gatedclk_clock_en="ena" */;
    input data_in,ena;
    output data_out;
endmodule
```

### **VHDL Example**

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

entity bbram is
    port (addr: IN std_logic_VECTOR(6 downto 0);
          din: IN std_logic_VECTOR(7 downto 0);
          dout: OUT std_logic_VECTOR(7 downto 0);
          clk: IN std_logic;
```

```
        en: IN std_logic;
        we: IN std_logic );
attribute syn_black_box : boolean;
attribute syn_black_box of bbram : entity is true;
attribute syn_isclock : boolean;
attribute syn_isclock of clk: signal is true;
attribute syn_gatedclk_clock_en : string;
attribute syn_gatedclk_clock_en of clk : signal is "en";
end entity bbram;

architecture bb of bbram is
attribute syn_force_seq_prim : boolean;
attribute syn_force_seq_prim of bb : architecture is true;
begin
end architecture bb;
```

## syn\_formal\_blackbox

### Directive

Defines a module or component as a black box for formal verification. The module or component will not be implemented as a black box for synthesis.

### syn\_formal\_blackbox Syntax

Verilog    *object /\* synthesis syn\_formal\_blackbox = 1|0 \*/;*

Verilog  
Example

---

### Description

Specifies that the module or component is a black box for formal verification. A black box module only has its interface defined for formal verification. Its contents cannot be accessed or verified during formal verification. A module can be a black box whether or not it is empty.

### Verilog Example

```
module myreg ( datain, rst, clk, en, dout, cout )
    /* synthesis syn_formal_blackbox=1 */;
    input clk, rst, datain, en;
    output dout;
    output cout;
    reg dreg;
    assign cout = en & datain;
    always @ (posedge clk or posedge rst)
    begin
        if (rst)
            dreg <= 'b0;
        else
            dreg <= datain;
    end
    assign dout = dreg;
endmodule

module top( clk1,en1, data1, q1, q2 );
    input clk1, en1;
    input data1;
    output q1, q2;
    wire cwire, rwire;
    wire clk_gt;
```

```
assign clk_gt = en1 & clk1;
// Register module
myreg U_reg (
    .datain(data1),
    .rst(1'b1),
    .clk(clk_gt),
    .en(1'b0),
    .dout(rwire),
    .cout(cwire)
);
assign q1 = rwire;
assign q2 = cwire;
endmodule

library ieee;
use ieee.std_logic_1164.all;
entity top is
    port (data1: in std_logic;
          clk1: in std_logic;
          en1: in std_logic;
          q1: out std_logic;
          q2: out std_logic);
end;

architecture rtl of top is
signal cwire, rwire: std_logic;
signal clk_gt: std_logic;
component dff is
    port (datain: in std_logic;
          rst: in std_logic;
          clk: in std_logic;
          en: in std_logic;
          dout: out std_logic;
          cout: out std_logic);
end component;
begin
    U1 : dff port map(datain => data1, rst => '1', clk =>
clk_gt, en => '0', dout => rwire, cout => cwire);
    q1 <= rwire;
    q2 <= cwire;
    clk_gt <= en1 and clk1;
end;
```

## Effect of Using syn\_formal\_blackbox

The following examples compare results when `syn_formal_blackbox` is applied or not. This example shows verification results if `syn_formal_blackbox` has been applied to the `myreg` module.

```
***** Verification Results *****
Verification SUCCEEDED
ATTENTION: synopsys_auto_setup mode was enabled.
See Synopsys Auto Setup Summary for details.

Reference design: r:/WORK/top
Implementation design: i:/WORK/top
3 Passing compare points

Matched Compare Points BBPin Loop BBNNet Cut Port DFF LAT TOTAL
-----|-----|-----|-----|-----|-----|-----|-----|-----|
Passing (equivalent) 1 0 0 0 2 0 0 3
Failing (not equivalent) 0 0 0 0 0 0 0 0
Not Compared
Don't verify 3 0 0 0 0 0 0 3
*****
```

The following example shows verification results if `syn_formal_blackbox` has not been applied to the `myreg` module.

```
***** Verification Results *****
Verification SUCCEEDED
ATTENTION: synopsys_auto_setup mode was enabled.
See Synopsys Auto Setup Summary for details.

Reference design: r:/WORK/top
Implementation design: i:/WORK/top
2 Passing compare points

Matched Compare Points BBPin Loop BBNNet Cut Port DFF LAT TOTAL
-----|-----|-----|-----|-----|-----|-----|-----|-----|
Passing (equivalent) 0 0 0 0 2 0 0 2
Failing (not equivalent) 0 0 0 0 0 0 0 0
Not Compared
Constant reg 0 1 1
*****
```

## **syn\_forward\_io\_constraints**

### *Attribute*

Controls forward annotation of the define\_input\_delay and define\_output\_delay timing constraints in the Tcl or NCF file for place-and-route tools.

### **Description**

Enables or disables forward annotation of the define\_input\_delay and define\_output\_delay timing constraints in the UCF file.

Use this attribute on the top level of a VHDL or Verilog file. Alternatively you can specify syn\_forward\_io\_constraints on a global object in the constraint file.

### **syn\_forward\_io\_constraints Syntax**

|         |                                                                                        |                                            |
|---------|----------------------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <b>define_global_attribute syn_forward_io_constraints {1 0}</b>                        | <a href="#">Constraints Editor Example</a> |
| Verilog | <b>object /* synthesis syn_forward_io_constraints = 1 0 */;</b>                        | <a href="#">Verilog Example</a>            |
| VHDL    | <b>attribute syn_forward_io_constraints of object :<br/>objectType is true false ;</b> | <a href="#">VHDL Example</a>               |

### **Constraints Editor Example**

|   | Enabled                             | Object Type | Object   | Attribute                  | Value | Val Type | Description                     |
|---|-------------------------------------|-------------|----------|----------------------------|-------|----------|---------------------------------|
| 1 | <input checked="" type="checkbox"/> | global      | <global> | syn_forward_io_constraints | 0     | boolean  | Forward annotate IO constraints |

### **Verilog Example**

```
module test (a,b,c,d,clk,rst)
    /* synthesis syn_forward_io_constraints = 0 */;
    input a,b,c;
    input clk, rst;
    output d;
    wire temp;
    reg d;
    assign temp = a && b || c;
```

```
always@(posedge clk or posedge rst)
if (rst)
    d = 1 'b0;
else
    d=temp;
endmodule
```

## VHDL Example

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

entity test is
    port (a : in std_logic;
          b : in std_logic;
          clk :in std_logic;
          rst : in std_logic;
          d: out std_logic );
end test;

architecture arch of test is
attribute syn_forward_io_constraints : boolean;
attribute syn_forward_io_constraints of all : architecture
    is false;
signal temp : std_logic;
begin
temp <= a and b;
process (clk, rst)begin
    if (rst ='1')then
        d <='0';
    elsif (clk'event and clk ='1')then
        d <= temp;
    end if;
end process;
end arch;
```

## Effect of Using syn\_forward\_io\_constraints

The default value of syn\_forward\_io\_constraints is 0. The table below shows the corresponding results in the UCF file:

---

|         |                                                                                  |
|---------|----------------------------------------------------------------------------------|
| Verilog | module test (a,b,c,d,clk,rst)<br>/* synthesis syn_forward_io_constraints = 0 */; |
|---------|----------------------------------------------------------------------------------|

---

|      |                                                                         |
|------|-------------------------------------------------------------------------|
| VHDL | attribute syn_forward_io_constraints of all :<br>architecture is false; |
|------|-------------------------------------------------------------------------|

---

```
# Period Constraints
#Begin clock constraints
# 1001 : define_clock {clk} -name {clk} -freq {100} -clockgroup {default_clkgroup_0}
# line 13 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_0/syn_forward_io_constraints_0.sdc
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 10.000 ns HIGH 50.000;
#End clock constraints

# I/O Registers Packing Constraints
INST "d" IOB=TRUE;

# End of generated constraints
```

The following example shows the results after setting `syn_forward_io_constraints` to 1 for the same design:

**Verilog**    module test (a,b,c,d,clk,rst)
/\* synthesis syn\_forward\_io\_constraints = 1 \*/;

**VHDL**    attribute syn\_forward\_io\_constraints of all :
architecture is true;

```
# Period Constraints
#Begin clock constraints
# 1001 : define_clock {clk} -name {clk} -freq {100} -clockgroup {default_clkgroup_0}
# line 13 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_1/syn_forward_io_constraints_1.sdc
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 10.000 ns HIGH 50.000;
#End clock constraints

# 1002 : define_input_delay {a} {1.00} -improve {0.00} -route {0.00}
# line 22 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_1/syn_forward_io_constraints_1.sdc
NET "a" TNM = "iodelay_1002_0";
TIMEGRP "iodelay_1002_0" OFFSET = IN 9.000 ns BEFORE "clk" RISING;

# 1003 : define_input_delay {b} {1.00} -improve {0.00} -route {0.00}
# line 23 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_1/syn_forward_io_constraints_1.sdc
NET "b" TNM = "iodelay_1003_0";
TIMEGRP "iodelay_1003_0" OFFSET = IN 9.000 ns BEFORE "clk" RISING;

# 1004 : define_input_delay {c} {1.00} -improve {0.00} -route {0.00}
# line 24 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_1/syn_forward_io_constraints_1.sdc
NET "c" TNM = "iodelay_1004_0";
TIMEGRP "iodelay_1004_0" OFFSET = IN 9.000 ns BEFORE "clk" RISING;

# 1005 : define_output_delay {d} {1.00} -improve {0.00} -route {0.00}
# line 25 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_1/syn_forward_io_constraints_1.sdc
NET "d" TNM = "iodelay_1005_0";
TIMEGRP "iodelay_1005_0" OFFSET = OUT 9.000 ns AFTER "clk" RISING;

# I/O Registers Packing Constraints
INST "d" IOB=TRUE;

# End of generated constraints
```

## **syn\_gatedclk\_clock\_en**

### *Directive*

Specifies the name of the enable pin inside a black box to support the “fix gated clocks” feature.

To use the `syn_gatedclk_clock_en` directive with a black box, you must also identify the clock signal with the `syn_isclock` directive and indicate that the fix gated clocks algorithm can be applied with the `syn_force_seq_prim` directive. The data type is String.

The `syn_gatedclk_clock_en` directive is one of several directives that you can use with the `syn_black_box` directive to define timing for a black box. See [syn\\_black\\_box, on page 526](#) for a list of the associated directives.

### **syn\_gatedclk\_clock\_en Syntax**

|         |                                                                                  |                 |
|---------|----------------------------------------------------------------------------------|-----------------|
| Verilog | <code>object /* synthesis syn_gatedclk_clock_en = "value" */;</code>             | Verilog Example |
| VHDL    | <code>attribute syn_gatedclk_clock_en of object : objectType is "value" ;</code> | VHDL Example    |

In the above syntax, `object` is the module name of the black box and `value` is the name of the enable pin.

### **Verilog Example**

```
module bbe (ena, clk, data_in, data_out)
    /* synthesis syn_black_box */
    /* synthesis syn_force_seq_prim=1 */;
    input clk
    /* synthesis syn_isclock = 1 */
    /* synthesis syn_gatedclk_clock_en="ena" */;
    input data_in,ena;
    output data_out;
endmodule
```

## VHDL Syntax and Examples

```
architecture top of top is
component bbram
    port (myclk : in bit;
          opcode : in bit_vector(2 downto 0);
          a, b : in bit_vector(7 downto 0);
          rambus : out bit_vector(7 downto 0) );
end component;

attribute syn_black_box : boolean;
attribute syn_black_box of bbram: component is true;
attribute syn_force_seq_prim : boolean
attribute syn_force_seq_prim of bbram: component is true;
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;
attribute syn_gatedclk_clock_en : string;
attribute syn_gatedclk_clock_en of bbram: signal is "ena";

-- Other code
```

## **syn\_gatedclk\_clock\_en\_polarity**

### *Directive*

Indicates the polarity of the clock enable port on a black box, so that the software can apply the algorithm to fix gated clocks.

### **Description**

Specifies the polarity of the clock enable port on a black box with a value of 0 or false indicating a negative clock polarity. If you do not set a polarity with this attribute, the software assumes a positive polarity.

The `syn_gatedclk_clock_en_polarity` directive is one of several directives that you can use with the `syn_black_box` directive to define timing for a black box. See [syn\\_black\\_box, on page 526](#) for a list of the associated directives.

### **syn\_gatedclk\_clock\_en\_polarity Syntax**

This table summarizes the syntax:

|         |                                                                                               |                 |
|---------|-----------------------------------------------------------------------------------------------|-----------------|
| Verilog | <code>object /* synthesis syn_gatedclk_clock_en_polarity = 1   0 */;</code>                   | Verilog Example |
| VHDL    | <code>attribute syn_gatedclk_clock_en_polarity of object : objectType is true   false;</code> | VHDL Example    |

### **Verilog Example**

```
module bbel (ena, clk, data_in, data_out)
    /* synthesis syn_black_box */
    /* synthesis syn_force_seq_prim=1 */;
    input clk /* synthesis syn_isclock = 1 */
        /* synthesis syn_gatedclk_clock_en="ena" */
        /* synthesis syn_gatedclk_clock_en_polarity = 0 */;
    input data_in,ena;
    output data_out;
endmodule
```

```

module bbe2 (ena, clk, data_in, data_out)
    /* synthesis syn_black_box */
    /* synthesis syn_force_seq_prim=1 */;
    input clk /* synthesis syn_isclock = 1 */
        /* synthesis syn_gatedclk_clock_en_polarity = 1 */
        /* synthesis syn_gatedclk_clock_en="ena" */;
    input data_in,ena;
    output data_out;
endmodule

module top (ena, clk, data_in , data_in2, data_out, data_out2 ) ;
    input ena , clk , data_in , data_in2;
    output data_out, data_out2 ;
    wire clk_in ;
    wire inv_enable;
    wire clk_in2 ;
    assign inv_enable = ~ena;
    assign clk_in = inv_enable & clk ;
    assign clk_in2 = ena & clk ;
    bbel u1 ( ena , clk_in , data_in , data_out ) ;
    bbe2 u2 ( ena, clk_in2, data_in2, data_out2 ) ;
endmodule

```

## VHDL Example

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity bbel is
    port (ena : in std_logic;
          clk : in std_logic;
          data_in : in std_logic;
          data_out : out std_logic );
attribute syn_black_box : boolean;
attribute syn_force_seq_prim : boolean;
attribute syn_gatedclk_clock_en_polarity : boolean;
attribute syn_gatedclk_clock_en : string;
attribute syn_isclock : boolean;
attribute syn_isclock of clk : signal is true;
attribute syn_gatedclk_clock_en_polarity of clk: signal is false;
attribute syn_gatedclk_clock_en of clk: signal is "ena";
attribute syn_force_seq_prim of clk: signal is true ;
end bbel;

```

```
architecture arch_bbe1 of bbe1 is
attribute syn_black_box : boolean;
attribute syn_black_box of arch_bbe1: architecture is true;
attribute syn_force_seq_prim of arch_bbe1: architecture is true;
begin
end arch_bbe1;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity bbe2 is
    port (ena : in std_logic;
          clk : in std_logic;
          data_in2 : in std_logic;
          data_out2 : out std_logic );
attribute syn_black_box : boolean;
attribute syn_gatedclk_clock_en_polarity : boolean;
attribute syn_force_seq_prim : boolean;
attribute syn_gatedclk_clock_en : string;
attribute syn_isclock : boolean;
attribute syn_isclock of clk : signal is true;
attribute syn_gatedclk_clock_en_polarity of clk: signal is true;
attribute syn_gatedclk_clock_en of clk: signal is "ena";
attribute syn_force_seq_prim of clk: signal is true ;
end bbe2;

architecture arch_bbe2 of bbe2 is
attribute syn_black_box : boolean;
attribute syn_black_box of arch_bbe2: architecture is true;
attribute syn_force_seq_prim of arch_bbe2: architecture is true;
begin
end arch_bbe2;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity top is
    port (ena : in std_logic;
          clk : in std_logic;
          data_in, data_in2 : in std_logic;
          data_out, data_out2 : out std_logic );
end top;
```

```
architecture arch_top of top is
component bbe1 is
    port (ena : in std_logic;
          clk : in std_logic;
          data_in : in std_logic;
          data_out : out std_logic );
end component;

component bbe2 is
    port (ena : in std_logic;
          clk : in std_logic;
          data_in2 : in std_logic;
          data_out2 : out std_logic);
end component;

signal clk_in, inv_enable, clk_in2 : std_logic;
begin
inv_enable <= not(ena);
clk_in <= inv_enable and clk;
clk_in2 <= ena and clk;
U1 : bbe1 port map (ena => ena, clk => clk_in,
                     data_in => data_in, data_out => data_out);
U2 : bbe2 port map (ena => ena, clk => clk_in2,
                     data_in2 => data_in2, data_out2 => data_out2);
end arch_top;
```

## **syn\_global\_buffers**

### *Attribute*

Specifies the number of global buffers to be used in a design.

### **Description**

Synthesis automatically adds global buffers for clock nets with high fanout; use this attribute to specify a maximum number of buffers and restrict the amount of global buffer resources used. Also, if there is a black box in the design that has global buffers, you can use syn\_global\_buffers to prevent the synthesis tool from inferring clock buffers or exceeding the number of global resources.

You specify the attribute globally on the top-level module/entity or view. The syn\_global\_buffers attribute works just like the xc\_global\_buffers attribute ([xc\\_global\\_buffers, on page 888](#)); the only difference is that you can specify syn\_global\_buffers in the source code or in a constraint file. If both attributes are specified for the same design, syn\_global\_buffers overwrites xc\_global\_buffers. Use this attribute instead of the global xc\_global\_buffers attribute.

### **syn\_global\_buffers Syntax**

FDC file      **define\_attribute {view} syn\_global\_buffers {maximum}**  
**define\_global\_attribute syn\_global\_buffers {maximum}**

Verilog      **object /\* synthesis syn\_global\_buffers = maximum \*/;**

VHDL      **attribute syn\_global\_buffers : integer;**  
**attribute syn\_global\_buffers of object : objectType is maximum;**

### **Constraints Editor Example**

|   | Enabled                             | Object Type | Object   | Attribute          | Value | Val Type | Description              |
|---|-------------------------------------|-------------|----------|--------------------|-------|----------|--------------------------|
| 1 | <input checked="" type="checkbox"/> | global      | <global> | syn_global_buffers | 10    | integer  | Number of global buffers |

## Verilog Example

```
module top (clk1, clk2, clk3, clk4, clk5, clk6, clk7, clk8, clk9,
            clk10, clk11, clk12, clk13, clk14, clk15, clk16, clk17, clk18,
            clk19, clk20, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11,
            d12, d13, d14, d15, d16, d17, d18, d19, d20, q1, q2, q3, q4, q5,
            q6, q7, q8, q9, q10, q11, q12, q13, q14, q15, q16, q17, q18,
            q19, q20, reset) /* synthesis syn_global_buffers = 10 */;

input clk1, clk2, clk3, clk4, clk5, clk6, clk7, clk8, clk9, clk10,
      clk11, clk12, clk13, clk14, clk15, clk16, clk17, clk18,
      clk19, clk20;
input d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14,
      d15, d16, d17, d18, d19, d20;
output q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, q13, q14,
      q15, q16, q17, q18, q19, q20;
input reset;
reg q1, q2, q3, q4, q5, q6, q7, q8, q9, q10,
      q11, q12, q13, q14, q15, q16, q17, q18, q19, q20;

always @ (posedge clk1 or posedge reset)
  if (reset)
    q1 <= 1'b0;
  else
    q1 <= d1;

always @ (posedge clk2 or posedge reset)
  if (reset)
    q2 <= 1'b0;
  else
    q2 <= d2;

always @ (posedge clk3 or posedge reset)
  if (reset)
    q3 <= 1'b0;
  else
    q3 <= d3;

always @ (posedge clk4 or posedge reset)
  if (reset)
    q4 <= 1'b0;
  else
    q4 <= d4;
```

```
always @ (posedge clk5 or posedge reset)
  if (reset)
    q5 <= 1'b0;
  else
    q5 <= d5;

always @ (posedge clk6 or posedge reset)
  if (reset)
    q6 <= 1'b0;
  else
    q6 <= d6;

always @ (posedge clk7 or posedge reset)
  if (reset)
    q7 <= 1'b0;
  else
    q7 <= d7;

always @ (posedge clk8 or posedge reset)
  if (reset)
    q8 <= 1'b0;
  else
    q8 <= d8;

always @ (posedge clk9 or posedge reset)
  if (reset)
    q9 <= 1'b0;
  else
    q9 <= d9;

always @ (posedge clk10 or posedge reset)
  if (reset)
    q10 <= 1'b0;
  else
    q10 <= d10;

always @ (posedge clk11 or posedge reset)
  if (reset)
    q11 <= 1'b0;
  else
    q11 <= d11;

always @ (posedge clk12 or posedge reset)
  if (reset)
    q12 <= 1'b0;
  else
    q12 <= d12
```

```
always @ (posedge clk13 or posedge reset)
  if (reset)
    q13 <= 1'b0;
  else
    q13 <= d13;

always @ (posedge clk14 or posedge reset)
  if (reset)
    q14 <= 1'b0;
  else
    q14 <= d14;

always @ (posedge clk15 or posedge reset)
  if (reset)
    q15 <= 1'b0;
  else
    q15 <= d15;

always @ (posedge clk16 or posedge reset)
  if (reset)
    q16 <= 1'b0;
  else
    q16 <= d16;

always @ (posedge clk17 or posedge reset)
  if (reset)
    q17 <= 1'b0;
  else
    q17 <= d17;

always @ (posedge clk18 or posedge reset)
  if (reset)
    q18 <= 1'b0;
  else
    q18 <= d18;

always @ (posedge clk19 or posedge reset)
  if (reset)
    q19 <= 1'b0;
  else
    q19 <= d19;

always @ (posedge clk20 or posedge reset)
  if (reset)
    q20 <= 1'b0;
  else
    q20 <= d20;
```

```
endmodule
```

## VHDL Example

Here is a VHDL example:

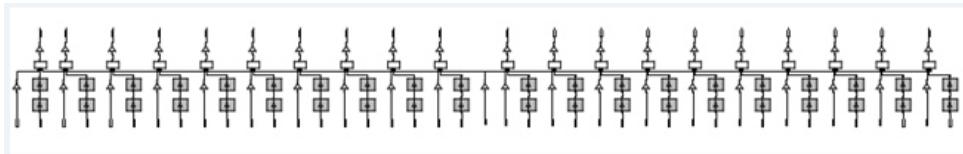
```
library ieee;
use ieee.std_logic_1164.all;

entity top is
    port (clk : in std_logic_vector(19 downto 0);
          d : in std_logic_vector(19 downto 0);
          q : out std_logic_vector(19 downto 0);
          reset : in std_logic );
end top;

architecture behave of top is
attribute syn_global_buffers : integer;
attribute syn_global_buffers of behave : architecture is 10;
begin
    process (clk, reset)
    begin
        for i in 0 to 19 loop
            if (reset = '1') then
                q(i) <= '0';
            elsif clk(i) = '1' and clk(i)' event then
                q(i) <= d(i);
            end if;
        end loop;
    end process;
end behave;
```

## Effect of Using `syn_global_buffers`

Without applying the attribute:

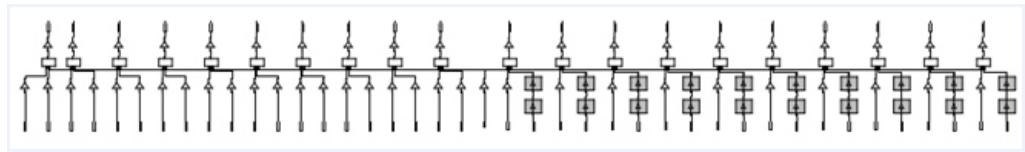


A message like the one below is generated:

```
@W:FX726: | Ignoring out-of-range global buffer count of 33 for
chip view:work.top(behave)
```

After applying attribute:

|         |                                                                                                                                                                                                                                                                                                                                                                                         |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Verilog | module top (clk1, clk2, clk3, clk4, clk5, clk6, clk7, clk8, clk9, clk10, clk11, clk12, clk13, clk14, clk15, clk16, clk17, clk18, clk19, clk20, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14, d15, d16, d17, d18, d19, d20, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, q13, q14, q15, q16, q17, q18, q19, q20, reset)<br>/* synthesis syn_global_buffers = 10 */; |
| VHDL    | attribute syn_global_buffers : integer;<br>attribute syn_global_buffers of behave : architecture is 10;                                                                                                                                                                                                                                                                                 |



Verify the results in the log file.

```
@N:FX112 : | Setting available global buffers in chip view:work.top(behave) to 10
Clock Buffers:
  Inserting Clock buffer for port clk[0],
  Inserting Clock buffer for port clk[1],
  Inserting Clock buffer for port clk[2],
  Inserting Clock buffer for port clk[3],
  Inserting Clock buffer for port clk[4],
  Inserting Clock buffer for port clk[5],
  Inserting Clock buffer for port clk[6],
  Inserting Clock buffer for port clk[7],
  Inserting Clock buffer for port clk[8],
  Inserting Clock buffer for port clk[9],
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on n
et clk_c[10] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on n
et clk_c[11] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on n
et clk_c[12] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on n
et clk_c[13] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on n
et clk_c[14] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on n
et clk_c[15] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on n
et clk_c[16] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on n
et clk_c[17] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on n
et clk_c[18] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on n
et clk_c[19] in view view:work.top(behave) (fanout 1)
@N:FX112 : | Setting available global buffers in chip view:work.top(behave) to 10
```

## **syn\_hier**

*Attribute/Directive*

Controls the amount of hierarchical transformation across boundaries on module or component instances during optimization.

### **syn\_hier Values**

| Default | Global | Object |
|---------|--------|--------|
| Soft    | No     | View   |

| Value           | Description                                                                                                                                                                                                                                                                                                                                       |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| soft (default)  | The synthesis tool determines the best optimization across hierarchical boundaries. This attribute affects only the design unit in which it is specified.                                                                                                                                                                                         |
| firm            | Preserves the interface of the design unit. However, when there is cell packing across the boundary, it changes the interface and does not guarantee the exact RTL interface. This attribute affects only the design unit in which it is specified.                                                                                               |
| hard            | Preserves the interface of the design unit and prevents most optimizations across the hierarchy. However, the boundary optimization for constant propagation is performed. Additionally, if all the clock logic is contained within the hard hierarchy, gated clock conversion can occur. This attribute affects only the specified design units. |
| fixed           | Preserves the interface of the design unit with no exceptions. Fixed prevents all optimizations performed across hierarchical boundaries and retains the port interfaces as well.<br>For more information, see <a href="#">Using syn_hier fixed , on page 644</a> .                                                                               |
| partition_fixed | Preserves the interface of the module and prevents optimizations across the hierarchy during the system level compile stage. The hierarchy is optimized during the synthesis stage.                                                                                                                                                               |

---

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| remove  | Removes the level of hierarchy for the design unit in which it is specified. The hierarchy at lower levels is unaffected. This only affects synthesis optimization. The hierarchy is reconstructed in the netlist and Technology view schematics.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| macro   | Preserves the interface and contents of the design with no exceptions. This value can only be set on structural netlists. (In the constraint file, or using the SCOPE editor, set <code>syn_hier</code> to macro on the view (the <code>V:</code> object type)).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| flatten | <p>Flattens the hierarchy of all levels below, but not the one where it is specified. This only affects synthesis optimization. The hierarchy is reconstructed in the netlist and Technology view schematics. To create a completely flattened netlist, use the <code>syn_netlist_hierarchy</code> attribute (<a href="#"><i>syn_netlist_hierarchy , on page 695</i></a>), set to false.</p> <p>You can use flatten in combination with other <code>syn_hier</code> values; the effects are described in <a href="#"><i>Using syn_hier flatten with Other Values , on page 651</i></a>.</p> <p>If you apply <code>syn_hier</code> to a compile point, flatten is the only valid attribute value. All other values only apply to the current level of hierarchy. The compile point hierarchy is determined by the type of compile point specified, so a <code>syn_hier</code> value other than flatten is redundant and is ignored.</p> |

---

## Description

During synthesis, the tool dissolves as much hierarchy as possible to allow efficient logic optimization across hierarchical boundaries while maintaining optimal run times. The tool then rebuilds the hierarchy as close as possible to the original source to preserve the topology of the design.

Use the `syn_hier` attribute to address specific needs to maintain the original design hierarchy during optimization. This attribute gives you manual control over flattening/preserving instances, modules, or architectures in the design.

It is advised that you avoid using `syn_hier="fixed"` with tri-states.

## Syntax Specification

---

|          |                                                         |
|----------|---------------------------------------------------------|
| FDC file | <code>define_attribute {object} syn_hier {value}</code> |
|----------|---------------------------------------------------------|

---

|         |                                                         |
|---------|---------------------------------------------------------|
| Verilog | <code>object /* synthesis syn_hier = "value" */;</code> |
|---------|---------------------------------------------------------|

---

|      |                                                                      |
|------|----------------------------------------------------------------------|
| VHDL | <code>attribute syn_hier of object : architecture is "value";</code> |
|------|----------------------------------------------------------------------|

---

## SCOPE Example

|   | Enable                              | Object Type | Object     | Attribute | Value | Value Type | Description                  | Comment |
|---|-------------------------------------|-------------|------------|-----------|-------|------------|------------------------------|---------|
| 1 | <input checked="" type="checkbox"/> | view        | v:work.alu | syn_hier  | hard  | string     | Control hierarchy flattening |         |

```
define_attribute {v:work.alu} {syn_hier} {hard}
```

### Example of Applying syn\_hier Attribute Globally

The `syn_hier` attribute is not supported globally. However, you can apply this attribute globally on design hierarchies using Tcl collection commands.

To do this, create a global collection of the design views in the FDC constraint file. Then, apply the attribute to the collection as shown below:

```
define_scope_collection all_views {find {v:*}}
define_attribute {$all_views} {syn_hier} {hard}
```

### syn\_hier in the SCOPE Window

If you use the SCOPE window to specify the `syn_hier` attribute, do not drag and drop the object into the SCOPE spreadsheet. Instead, first select `syn_hier` in the Attribute column, and then use the pull-down menu in the Object column to select the object. This is because you must set the attribute on a view (v:). If you drag and drop an object, you might not get a view object. Selecting the attribute first ensures that only the appropriate objects are listed in the Object column.

### Using syn\_hier fixed

When you use the fixed value with `syn_hier`, hierarchical boundaries are preserved with no exceptions. For example, optimizations such as constant propagation and gated or generated clock conversions are not performed across these boundaries.

---

**Note:** It is recommended that you do not use `syn_hier` with the fixed value on modules that have ports driven by tri-state gates. For details, see [When Using Tri-states, on page 645](#).

---

## When Using Tri-states

It is advised that you avoid using `syn_hier="fixed"` with tri-states. However, if you do, here is how the software handles the following conditions:

- Tri-states driving output ports

If a module with `syn_hier="fixed"` includes tri-state gates that drive a primary output port, then the synthesis software retains a tri-state buffer so that the P&R tool can pack the tri-state into an output port.

For example, the software can infer a Xilinx BUFT so that the ISE P&R tool packs the tri-state into an OBUFT output port.

- Tri-states driving internal logic

If a module with `syn_hier="fixed"` includes tri-state gates that drive internal logic, then the synthesis software converts the tri-state gate to a MUX and optimizes within the module accordingly.

In the following code example, `myreg` has `syn_hier` set to `fixed`.

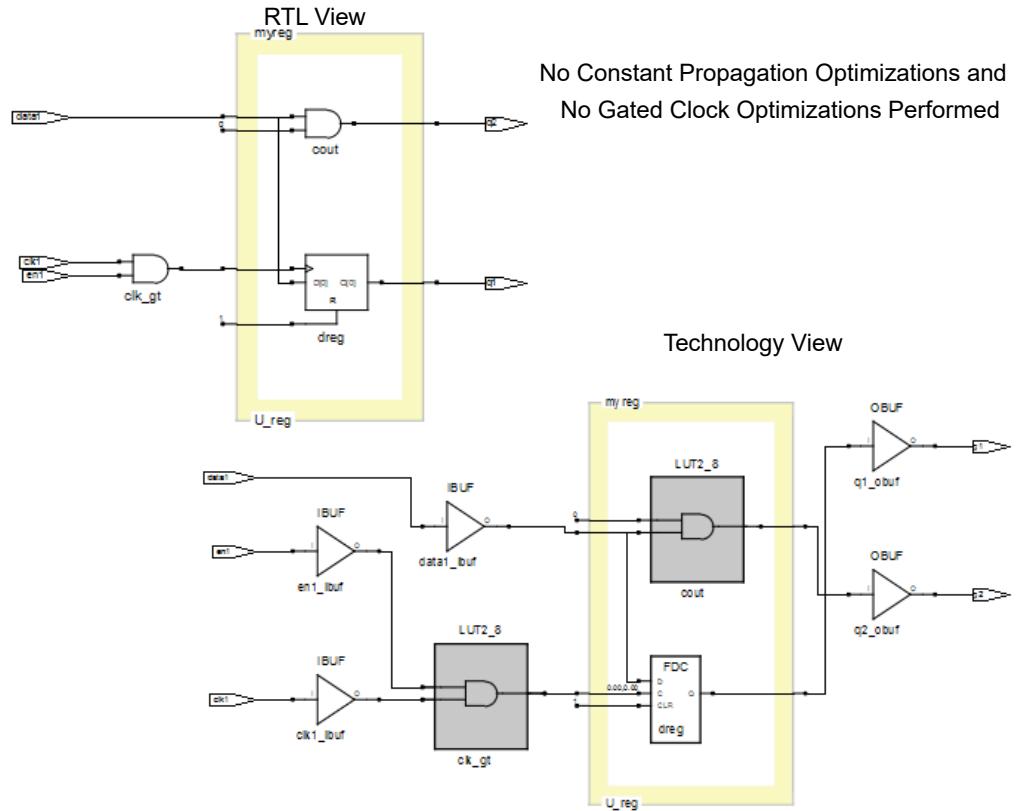
```
module top(
    clk1, en1, data1,
    q1, q2
);
    input clk1, en1;
    input data1;
    output q1, q2;
    wire cwire, rwire;
    wire clk_gt;
    assign clk_gt = en1 & clk1;

    // Register module
    myreg U_reg (
        .datain(data1),
        .rst(1'b1),
        .clk(clk_gt),
        .en(1'b0),
        .dout(rwire),
        .cout(cwire)
    );
    assign q1 = rwire;
    assign q2 = cwire;
endmodule
```

```
module myreg (
    datain,
    rst,
    clk,
    en,
    dout,
    cout
) /* synthesis syn_hier = "fixed" */;
input clk, rst, datain, en;
output dout;
output cout;
reg dreg;
assign cout = en & datain;

always @ (posedge clk or posedge rst)
begin
    if (rst)
        dreg <= 'b0;
    else
        dreg <= datain;
end
assign dout = dreg;
endmodule
```

The HDL Analyst views show that myreg preserves its hierarchical boundaries without exceptions and prevents constant propagation and gated clock conversions optimizations.



## Effect of Using `syn_hier`

The following VHDL and Verilog examples show the effects of using the fixed and macro values with the `syn_hier` attribute.

## VHDL Example 1

```
library ieee;
use ieee.std_logic_1164.all;
entity top is
port (data1: in std_logic;
      clk1: in std_logic;
      en1: in std_logic;
      q1: out std_logic;
      q2: out std_logic);
end;

architecture rtl of top is
signal cwire, rwire: std_logic;
signal clk_gt: std_logic;
component dff is
port (datain: in std_logic;
      rst: in std_logic;
      clk: in std_logic;
      en: in std_logic;
      dout: out std_logic;
      cout: out std_logic);
end component;
begin
U1 : dff port map(datain => data1, rst => '1', clk =>
      clk_gt, en => '0', dout => rwire, cout => cwire);
q1 <= rwire;
q2 <= cwire;
clk_gt <= en1 and clk1;
end;

library ieee;
use ieee.std_logic_1164.all;
entity dff is
port (datain: in std_logic;
      rst: in std_logic;
      clk: in std_logic;
      en: in std_logic;
      dout: out std_logic;
      cout: out std_logic);
end;

architecture rtl of dff is
signal dreg: std_logic;
attribute syn_hier : string;
attribute syn_hier of rtl: architecture is "fixed";
begin
```

```

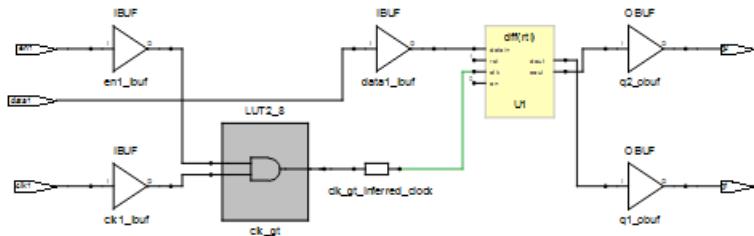
process (clk, rst)
begin
  if (rst = '1') then
    dreg<= '0';
  elsif (clk'event and clk ='1') then
    dreg<= datain;
  end if;
  dout <= dreg;
end process;
end;

```

After applying attribute with the value *fixed*:

Verilog    Module myreg(datain,rst,clk,en,dout,cout)/\*synthesis syn\_hier="fixed"\*/;

VHDL    attribute syn\_hier : string;
attribute syn\_hier of rtl: architecture is "fixed";



## Verilog Example 2

```

module inc(a_in, a_out) /* synthesis syn_hier = "macro" */;
  input [3:0] a_in;
  output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
  input [3:0] d;
  input clk, rst;
  output [3:0] q;
  reg [3:0] q;
  always @ (posedge clk or posedge rst)

```

```

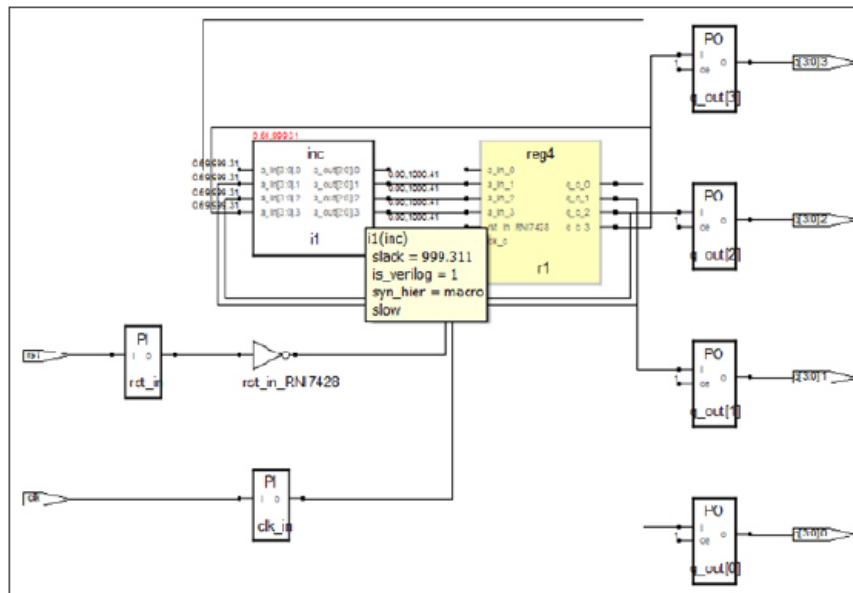
if(rst)
q <= 0;
else
q <= d;
endmodule

module top(clk, rst, q);
input clk, rst;
output [3:0] q;
wire [3:0] a_in;
inc i1(q, a_in);
reg4 r1(clk, rst, a_in, q);
endmodule

```

After applying attribute with value *macro*:

|         |                                                                                                               |
|---------|---------------------------------------------------------------------------------------------------------------|
| Verilog | <code>module inc(a_in, a_out) /* synthesis syn_hier = "macro" */;</code>                                      |
| VHDL    | <code>attribute syn_hier : string;</code><br><code>attribute syn_hier of rtl: architecture is "macro";</code> |



## Using syn\_hier flatten with Other Values

You can combine flatten with other syn\_hier values as shown below:

|                |                                                                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| flatten,soft   | Same as flatten.                                                                                                                                                                           |
| flatten,firm   | Flattens all lower levels of the design but preserves the interface of the design unit in which it is specified. This option also allows optimization of cell packing across the boundary. |
| flatten,remove | Flattens all lower levels of the design, including the one on which it is specified.                                                                                                       |

If you use flatten in combination with another option, the tool flattens as directed until encountering another syn\_hier attribute at a lower level. The lower level syn\_hier attribute then takes precedence over the higher level one.

These examples demonstrate the use of the flatten and remove values to flatten the current level of the hierarchy and all levels below it (unless you have defined another syn\_hier attribute at a lower level).

```
Verilog module top1 (Q, CLK, RST, LD, CE, D)
/* synthesis syn_hier = "flatten,remove" */;
// Other code
```

```
VHDL architecture struct of cpu is
attribute syn_hier : string;
attribute syn_hier of struct: architecture is "flatten,remove";
-- Other code
```

## **syn\_implement**

### *Attribute*

Ensures that the log file preserves user-defined custom retention models after compilation.

### **Syntax**

**syn\_implement = "1"**

1 is the only valid value.

### **Example**

```
module ret_dffrse (Q, CLK, SAVE, RESTORE, D, S, R, E)
    /* synthesis syn_implement = "1" syn_upf_ret_type = "DFFRSE" */;
    <Other logic>
endmodule
```

## syn\_insert\_buffer

### *Attribute*

Inserts a technology-specific clock buffer.

### Description

Use this attribute to insert a clock buffer. You can also use it on a non-clock high fanout net, such as reset or common enable that needs global routing, to insert a global buffer for that port.

### syn\_insert\_buffer Syntax

|         |                                                                             |                                            |
|---------|-----------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute object syn_insert_buffer {value}</code>              | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_insert_buffer = "value" */;</code>            | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_insert_buffer of object : objectType is "value";</code> | <a href="#">VHDL Example</a>               |

The syntax term *value* is defined in the following table:

| Object   | Value           | Description                                                                                                                                                                                                                                                                                                   |
|----------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Instance | BUFGMUX<br>none | BUFGMUX inference is disabled by default. If the attribute is not specified, the default behavior is to infer the LUT driving the BUFG. Or, specify one of these values: <ul style="list-style-type: none"> <li>• none: Infers only the LUT, without the BUFG</li> <li>• BUFGMUX: Infers a BUFGMUX</li> </ul> |
| Port/Net | BUFG            | This is the default buffer inferred if nothing is specified.                                                                                                                                                                                                                                                  |
|          | BUFH            | Explicitly specify this value to infer a BUFH.                                                                                                                                                                                                                                                                |
|          | BUFR            | Explicitly specify this value to infer a regional clock buffer, BUFR.                                                                                                                                                                                                                                         |

### Constraints Editor Example

|   | Enabled                             | Object Type | Object    | Attribute         | Value   | Val Type | Description |
|---|-------------------------------------|-------------|-----------|-------------------|---------|----------|-------------|
| 1 | <input checked="" type="checkbox"/> |             | i:clk_mux | syn_insert_buffer | BUFGMUX |          |             |

## Verilog Example

```

`define SIZE 16
module cr_top (
    input rst,
    input [`SIZE:0] in1,in2,
    output reg [`SIZE:0] out,
    input clk1,
    input clk2,
    input sel_clk );
    wire clk_mux /*synthesis syn_insert_buffer="BUFMUX"*/;
    assign clk_mux = sel_clk ? clk1 : clk2;

    always @ (posedge clk_mux or posedge rst)
    begin
        if(rst)
            out <= 0;
        else
            out <= in1&in2 ;
        end
    endmodule

```

## VHDL Example

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity prep2_1 is
    port (clk : in bit;
          rst : in bit;
          sel : in bit;
          ldcomp : in bit;
          ldpre : in bit;
          data1,data2 : in std_logic_vector(7 downto 0);
          data0 : out std_logic_vector(7 downto 0) );
end prep2_1;

architecture behave of prep2_1 is
    signal equal: bit;
    signal mux_output: std_logic_vector(7 downto 0);
    signal lowreg_output: std_logic_vector(7 downto 0);
    signal highreg_output: std_logic_vector(7 downto 0);
    signal data0_i: std_logic_vector(7 downto 0);
begin
    compare: process(data0_i, lowreg_output)

```

```
begin
    if data0_i = lowreg_output then
        equal <= '1';
    else
        equal <= '0';
    end if;
end process compare;

mux: process(sel, data1, highreg_output)
begin
    case sel is
        when '0' =>
            mux_output <= highreg_output;
        when '1' =>
            mux_output <= data1;
    end case;
end process mux;

registers: process (rst,clk)
begin
    if (rst = '1') then
        highreg_output <= "00000000";
        lowreg_output <= "00000000";
    elsif clk = '1' and clk'event then
        if ldpre = '1' then
            highreg_output <= data2;
        end if;
        if ldcomp = '1' then
            lowreg_output <= data2;
        end if;
    end if;
end process registers;

counter: process (rst,clk)
begin
    if rst = '1' then
        data0_i <= "00000000";
    elsif clk = '1' and clk'event then
        if equal = '1' then
            data0_i <= mux_output;
        elsif equal = '0' then
            data0_i <= data0_i + "00000001";
        end if;
    end if;
end process counter;
data0 <= data0_i;
end behave;
```

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

entity prep2_2 is
    port (CLK : in bit;
          RST : in bit;
          SEL : in bit;
          LDCOMP : in bit;
          LDPRE : in bit;
          DATA1,DATA2 : in std_logic_vector(7 downto 0);
          DATA0 : out std_logic_vector(7 downto 0) );
attribute syn_insert_buffer : string;
attribute syn_insert_buffer of clk : signal is "BUFGMUX";
end prep2_2;

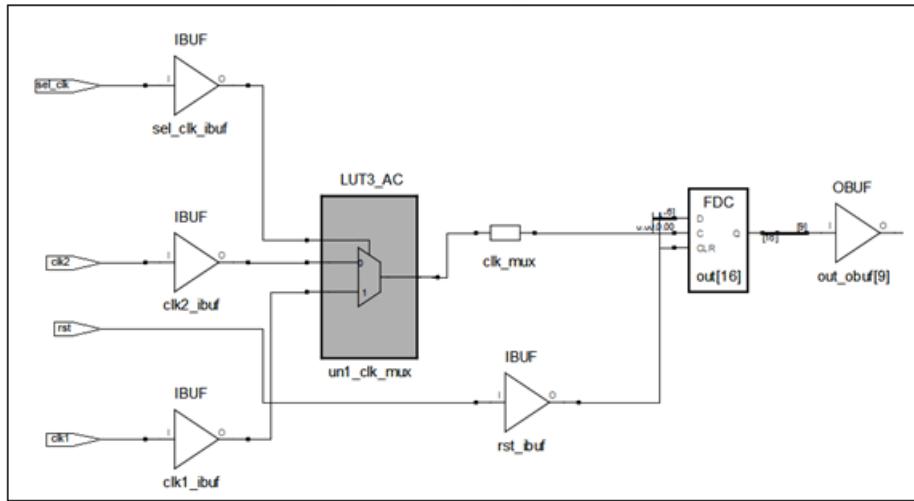
architecture behave of prep2_2 is
component prep2_1
    port (clk : in bit;
          rst : in bit;
          sel : in bit;
          ldcomp : in bit;
          ldpre : in bit;
          data1,data2 : in std_logic_vector(7 downto 0);
          data0 : out std_logic_vector(7 downto 0) );
end component;

signal data0_internal : std_logic_vector (7 downto 0);

begin
inst1: prep2_1 port map(clk => CLK, rst => RST, sel => SEL,
ldcomp => LDCOMP, ldpre => LDPRE, data1 => DATA1,
data2 => DATA2, data0 => data0_internal );
inst2: prep2_1 port map(clk => CLK, rst => RST, sel => SEL,
ldcomp => LDCOMP, ldpre => LDPRE, data1 => data0_internal,
data2 => DATA2, data0 => DATA0);
end behave;
```

## Effect of Using syn\_insert\_buffer

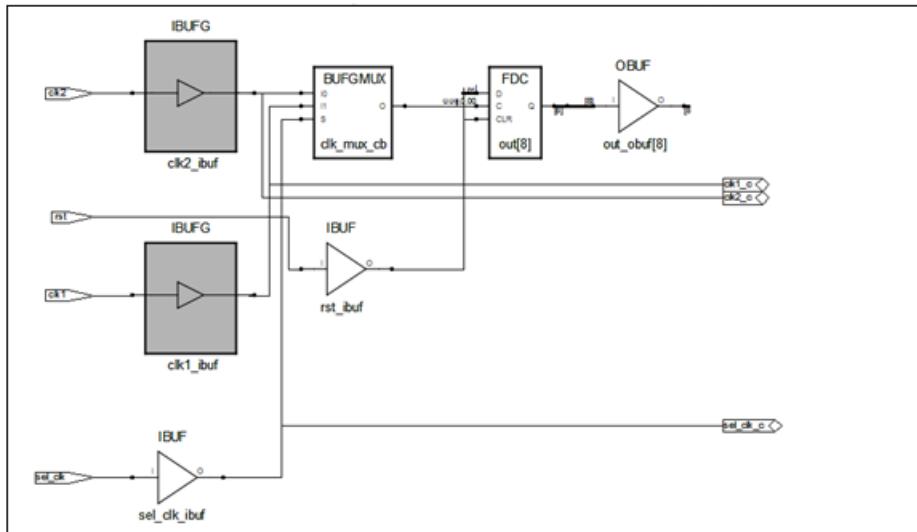
This example shows a device in the default state, before syn\_insert\_buffer is applied. There is no BUFGMUX on the clock.



After applying the `syn_insert_buffer` attribute, the tool infers a BUFGMUX, as shown below.

Verilog      `wire clk_mux /*synthesis syn_insert_buffer="BUFGMUX"*/;`

VHDL      `attribute syn_insert_buffer of clk_out : signal is "BUFGMUX";`



## syn\_insert\_pad

### *Attribute*

Removes an existing I/O buffer from a port or net when I/O buffer insertion is enabled.

### Description

Use the syn\_insert\_pad attribute when the Disable I/O Insertion option is disabled (when buffers are automatically inserted), to selectively remove an individual buffer from a port or net or to replace a previously removed buffer.

- Setting the attribute to 0 on a port or net removes the I/O buffer (or prevents an I/O buffer from being automatically inserted).
- Setting the attribute to 1 on a port or net replaces a previously removed I/O buffer.

The syn\_insert\_pad attribute can only be applied through a constraint file.

### syn\_insert\_pad Syntax

FDC      **define\_attribute {object} syn\_insert\_pad {1|0}**

[Constraints  
Editor  
Example](#)

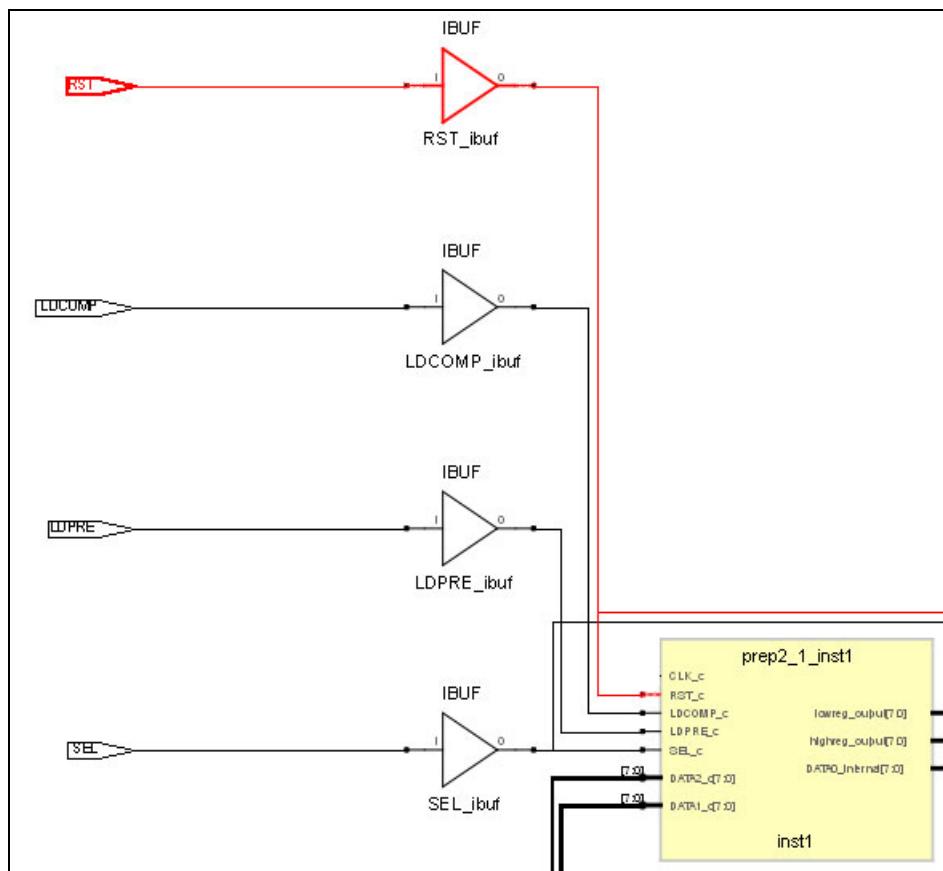
### Constraints Editor Example

The following figure shows the attribute applied to the RST port using the constraints editor window:

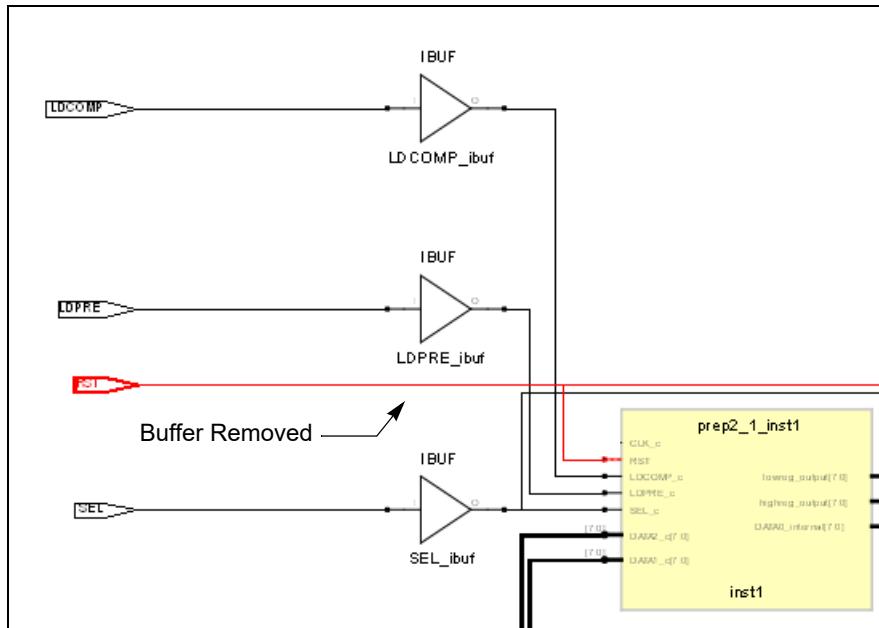
| Enable                              | Object Type | Object | Attribute      | Value | Value Type |
|-------------------------------------|-------------|--------|----------------|-------|------------|
| <input checked="" type="checkbox"/> | <any>       | p:RST  | syn_insert_pad | 0     |            |

### Effect of Using syn\_insert\_pad

Original design before applying syn\_insert\_pad (or after applying syn\_insert\_pad with a value of 1 to replace a previously removed buffer).



This is the view after applying `syn_insert_pad` with a value of 0 to remove the original buffer from the RST input:



## **syn\_latch\_ramstyle**

### *Attribute*

Controls the mapping of primitives.

[Return to Attributes and Directives Summary.](#)

### **Description**

This attribute allows compiler primitives to be mapped to block ram, enabling automatic inference and runtime improvement.

Values are **block\_ram** and **logic**.

### **syn\_latch\_ramstyle Syntax**

---

FDC      **define\_global\_attribute syn\_latch\_ramstyle value**

---

### **syn\_latch\_ramstyle Values**

**block\_ram**      Sets the mapper to map to block RAM.

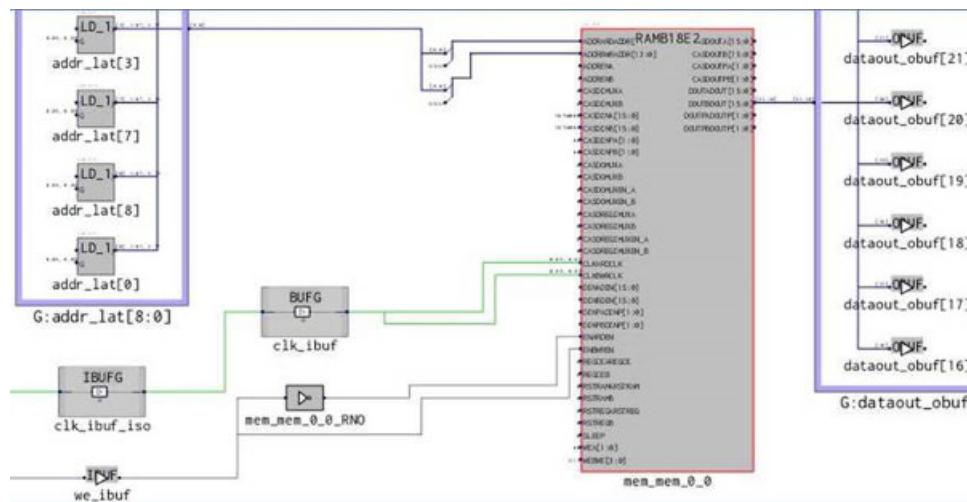
---

**logic**      Sets the mapper to map latchram primitive to logic (i.e. latches).

---

### **Effect of using syn\_latch\_ramstyle**

Applying the `syn_latch_ramstyle` attribute results in compiler primitives being mapped to block ram.



## syn\_isclock

*Directive*

Specifies an input port on a black box as a clock.

### Description

Used with the `syn_black_box` directive to specify an input port on a black box as a clock even though the port name does not reflect a recognized clock. Using this directive connects the port to a clock buffer if appropriate.

The `syn_isclock` directive is one of several directives that you can use with the `syn_black_box` directive to define timing for a black box. See [syn\\_black\\_box, on page 526](#) for a list of the associated directives.

### syn\_isclock Values Syntax

Verilog    `object /* synthesis syn_isclock = 1 */;`

VHDL    `attribute syn_isclock of object : objectType is true;`

---

### Verilog Example

```
module test (myclk, a, b, tout,) /* synthesis syn_black_box */;
  input myclk /* synthesis syn_isclock = 1 */;
  input a, b;
  output tout;
endmodule

//Top Level
module top (input clk, input a, b, output fout);
  test U1 (clk, a, b, fout);
endmodule
```

### VHDL Example

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

```
entity test is
generic (size: integer := 8);
    port (tout : out std_logic_vector (size- 1 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          myclk : in std_logic );
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;
end;

architecture rtl of test is
attribute syn_black_box : boolean;
attribute syn_black_box of rtl: architecture is true;
begin
end;

-- TOP Level--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

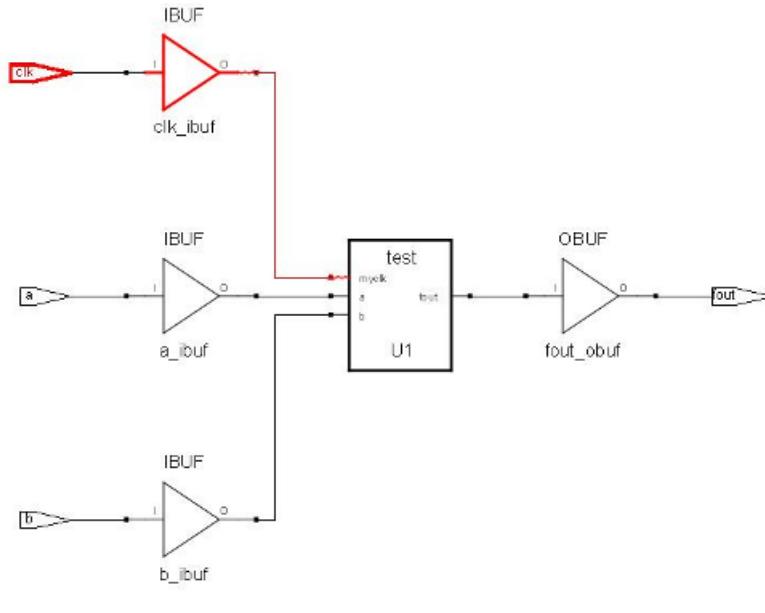
entity top is
generic (size: integer := 8);
    port (fout : out std_logic_vector (size- 1 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          clk : in std_logic );
end;

architecture rtl of top is
component test
generic (size: integer := 8);
    port (tout : out std_logic_vector (size- 1 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          myclk : in std_logic );
end component;

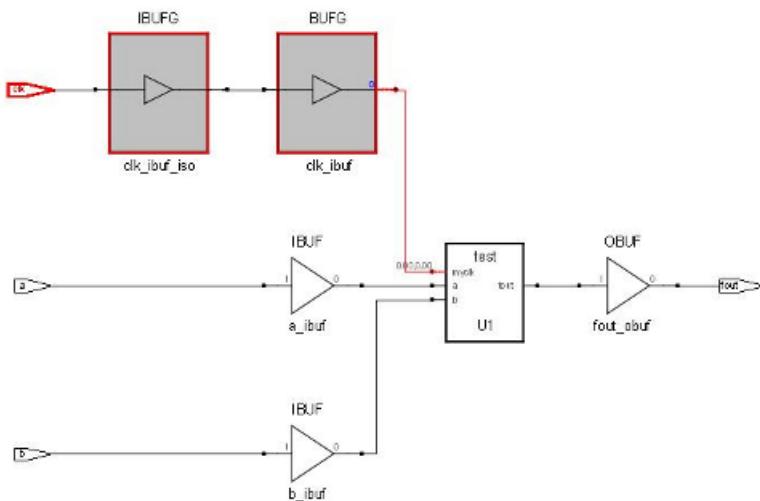
begin
U1 : test port map (fout, a, b, clk);
end;
```

## Effect of Using syn\_isclock

This figure shows the view before using `syn_isclock`:



This figure shows the HDL Analyst view after using `syn_isclock`:



## **syn\_keep**

### *Directive*

Preserves the specified net and keeps it intact during optimization and synthesis.

### **Description**

With this directive, the tool preserves the net without optimizing it away by placing a temporary keep buffer primitive on the net as a placeholder. You can view this buffer in the schematic views (see [Effect of Using syn\\_keep, on page 671](#) for an example). The buffer is not part of the final netlist, so no extra logic is generated. There are various situations where this directive is useful:

- To preserve a net that would otherwise be removed as a result of optimization. You might want to preserve the net for simulation results or to obtain a different synthesis implementation.
- To prevent duplicate cells from being merged during optimization. You apply the directive to the nets connected to the input of the cells you want to preserve.
- As a placeholder to apply the -through option of the `set_multicycle_path` or `set_false_path` timing constraint. This allows you to specify a unique path as a multiple-cycle or false path. Apply the constraint to the keep buffer.
- To prevent the absorption of a register into a macro. If you apply `syn_keep` to a `reg` or signal that will become a sequential object, the tool keeps the register and does not absorb it into a macro.

### **syn\_keep Syntax**

|          |                                                                                                               |                                 |
|----------|---------------------------------------------------------------------------------------------------------------|---------------------------------|
| CDC File | <code>define_directive {n:libName.moduleName   netName} {syn_keep} {0 1}</code>                               | <a href="#">CDC Example</a>     |
| Verilog  | <code>object /* synthesis syn_keep = 1 */;</code>                                                             | <a href="#">Verilog Example</a> |
| VHDL     | <code>attribute syn_keep : boolean;</code><br><code>attribute syn_keep of object : objectType is true;</code> | <a href="#">VHDL Example</a>    |

In the Verilog syntax, *object* is a wire or reg declaration. Make sure that there is a space between the object name and the beginning of the comment slash (/). In the VHDL syntax, *object* is a single or multiple-bit signal.

## **syn\_keep with Multiple Nets in Verilog**

In the following statement, syn\_keep only applies to the last variable in the wire declaration, which is net c:

```
wire a,b,c /* synthesis syn_keep=1 */;
```

To apply syn\_keep to all the nets, use one of the following methods:

- Declare each individual net separately as shown below.

```
wire a /* synthesis syn_keep=1 */;
wire b /* synthesis syn_keep=1 */;
wire c /* synthesis syn_keep=1 */;
```

- Use Verilog 2001 parenthetical comments, to declare the syn\_keep directive as a single line statement.

```
(* syn_keep=1 *) wire a,b,c;
```

## **syn\_keep and SystemVerilog Data Types**

The SystemVerilog data types behave like logic or reg, and SystemVerilog allows them to be assigned either inside or outside an always block. If you want to use syn\_keep to preserve a net with a SystemVerilog data type, like bit, byte, longint or shortint for example, you must make sure that continuous assigns are made inside an always block, not outside.

The following table shows examples of SystemVerilog data type assignments:

Assignment in always block,  
syn\_keep works correctly

```
assign keep1_wireand_out;
assign keep2_wireand_out;
always @(*) begin
    keep1_bitand_out;
    keep2_bitand_out;
    keep1_byteand_out;
    keep2_byteand_out;
    keep1_longintand_out;
    keep2_longintand_out;
    keep1_shortintand_out;
    keep2_shortintand_out;
```

Assignment outside always block,  
syn\_keep does not work

```
assign keep1_wireand_out;
assign keep2_wireand_out;
assign keep1_bitand_out;
assign keep2_bitand_out;
assign keep1_byteand_out;
assign keep2_byteand_out;
assign keep1_longintand_out;
assign keep2_longintand_out;
assign keep1_shortintand_out;
assign keep2_shortintand_out;
```

## Comparison of syn\_keep, syn\_preserve, and syn\_noprune

Although these directives all work to preserve logic from optimization, syn\_keep, syn\_preserve, and syn\_noprune work on different objects:

|                     |                                                                                                                                                                                                                                                                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>syn_keep</b>     | Only works on nets and combinational logic. It ensures that the wire is kept during synthesis, and that no optimizations cross the wire. This directive is usually used to prevent unwanted optimizations and to ensure that manually created replications are preserved. When applied to a register, the register is preserved and not absorbed into a macro. |
| <b>syn_preserve</b> | Ensures that registers are not optimized away.                                                                                                                                                                                                                                                                                                                 |
| <b>syn_noprune</b>  | Ensures that a black box is not optimized away when its outputs are unused (i.e., when its outputs do not drive any logic).                                                                                                                                                                                                                                    |

## CDC Example

```
define_directive {n:work.sub_new|temp1} {syn_keep} {1}
```

## Verilog Example

```
module example2(out1, out2, clk, in1, in2);
    output out1, out2;
    input clk;
    input in1, in2;
    wire and_out;
    wire keep1 /* synthesis syn_keep=1 */;
    wire keep2 /* synthesis syn_keep=1 */;
    reg out1, out2;
    assign and_out=in1&in2;
    assign keep1=and_out;
    assign keep2=and_out;

    always @(posedge clk) begin;
        out1<=keep1;
        out2<=keep2;
    end
endmodule
```

## VHDL Example

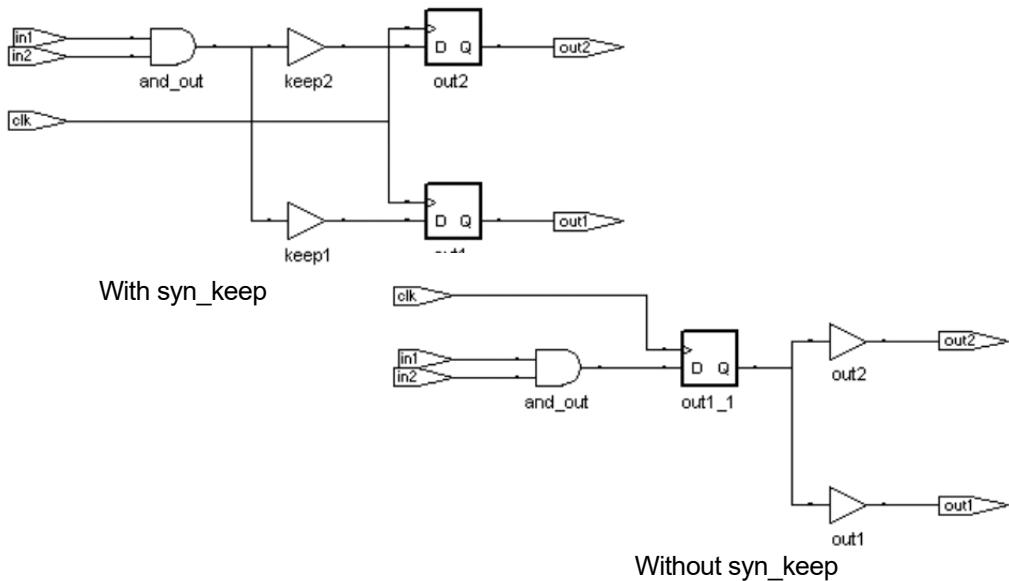
```
entity example2 is
    port (in1, in2 : in bit;
          clk : in bit;
          out1, out2 : out bit );
end example2;

architecture rtl of example2 is
attribute syn_keep : boolean;
signal and_out, keep1, keep2: bit;
attribute syn_keep of keep1, keep2 : signal is true;
begin
    and_out <= in1 and in2;
    keep1 <= and_out;
    keep2 <= and_out;
    process(clk)
    begin
        if (clk'event and clk = '1') then
            out1 <= keep1;
            out2 <= keep2;
        end if;
    end process;
end rtl;
```

## Effect of Using syn\_keep

When you use `syn_keep` on duplicate logic, the logic is retained instead of being optimized away. The following figure shows the view for two versions of a design.

In the first, `syn_keep` is set on the nets connected to the inputs of the registers `out1` and `out2`, to prevent sharing. The second figure shows the same design without `syn_keep`. Setting `syn_keep` on the input wires for the registers ensures that the design has duplicate registered outputs for `out1` and `out2`. If you do not apply `syn_keep` to `keep1` and `keep2`, the software optimizes `out1` and `out2`, and only has one register.



## **syn\_loc**

### *Attribute*

The `syn_loc` attribute specifies the location (placement) of ports.

### **Description**

Specifies pin locations for I/O pins and cores, and forward-annotates this information to the place-and-route tool. This attribute can only be specified in a top-level source file or a constraint file.

### **syn\_loc Syntax**

|         |                                                                        |                                            |
|---------|------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute portDesignName {syn_loc} {pinNumbers}</code>    | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_loc = "pinNumbers" */</code>             | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_loc of object : objectType is "pinNumbers";</code> | <a href="#">VHDL Example</a>               |

In the above syntax, `pinNumbers` is a comma-separated list of pin or placement numbers. Refer to the vendor data book for valid values.

### **Constraints Editor Example**

|   | Enable                              | Object Type | Object    | Attribute | Value       | Value Type | Description                |
|---|-------------------------------------|-------------|-----------|-----------|-------------|------------|----------------------------|
| 1 | <input checked="" type="checkbox"/> |             | out1[2:0] | syn_loc   | P14 P12,P11 | string     | Assign the object location |
| 2 |                                     |             |           |           |             |            |                            |

The following is an example of using this attribute:

```
define_attribute {DATA0[4:0]} syn_loc {P14,P12,P11,P5,P21}
```

You can also specify locations for individual bus bits with this attribute:

Specify individual bits:

```
define_attribute {valid[0]} syn_loc {R6}
define_attribute {valid[1]} syn_loc {T2}
define_attribute {valid[2]} syn_loc {R5}
define_attribute {valid[3]} syn_loc {R1}
```

Specify the b: prefix and the bit slice:

```
define_attribute {b:valid[0]} syn_loc {R6}
define_attribute {b:valid[1]} syn_loc {T2}
define_attribute {b:valid[2]} syn_loc {R5}
define_attribute {b:valid[3]} syn_loc {R1}
```

## Verilog Example

```
module test(a ,b, clk, out1);
  input clk;
  input [2:0]a;
  input [2:0]b;
  output reg [2:0] out1/* synthesis syn_loc = "P14,P12,P11" */;

  always@ (posedge clk)
  begin
    out1 <= a + b;
  end
endmodule
```

## VHDL Example

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

entity test is
  generic (s : integer := 2);
  port (clk: in std_logic;
        in1: in std_logic_vector(s downto 0);
        in2: in std_logic_vector(s downto 0);
        d_out: out std_logic_vector(5 downto 0) );
attribute syn_loc : string;
attribute syn_loc of d_out:signal is" P14,P12,P11,P5,P21,P13";
end test;
```

```
architecture beh of test is
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= in1 & in2;
        end if;
    end process;
end beh;
```

## Effect of Using syn\_loc

This is the netlist output before the attribute is applied:

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell test (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port (array (rename a "a[2:0]" 3) (direction INPUT))
          (port (array (rename b "b[2:0]" 3) (direction INPUT))
            (port (array (rename out1 "out1[2:0]" 3) (direction OUTPUT))
              (port clk (direction INPUT)
            )
          )
        (contents
          (instance clk_ibuf (viewRef PRIM (cellRef BUFG (libraryRef VIRTEX)))      )
          (instance clk_ibuf_iso (viewRef PRIM (cellRef IBUFG (libraryRef VIRTEX)))
        )
        (instance (rename out1Z0Z_0 "out1[0]" ) (viewRef PRIM (cellRef FD (libraryRef UNILIB)))
        )
        (instance (rename out1Z0Z_1 "out1[1]" ) (viewRef PRIM (cellRef FD (libraryRef UNILIB)))
        )
        (instance (rename out1Z0Z_2 "out1[2]" ) (viewRef PRIM (cellRef FD (libraryRef UNILIB)))
      )
    )
  )
)
```

When the attribute is applied, the output netlist shows the LOC property:

**VERILOG EDIF FILE:**

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell test (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port (array (rename a "a[2:0]") 3) (direction INPUT))
        (port (array (rename b "b[2:0]") 3) (direction INPUT))
        (port (array (rename out1 "out1[2:0]") 3) (direction OUTPUT)
          (property LOC (string "P14,P12,P11"))
```

**VHDL EDIF FILE:**

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell test (cellType GENERIC)
    (view beh (viewType NETLIST)
      (interface
        (port clk (direction INPUT)
      )
      (port (array (rename in1 "in1[2:0]") 3) (direction INPUT))
      (port (array (rename in2 "in2[2:0]") 3) (direction INPUT))
      (port (array (rename d_out "d_out[5:0]") 6) (direction OUTPUT)
        (property LOC (string "P14,P12,P11,P5,P21,P13"))
      )
    )
  )
```

## syn\_looplimit

### Directive

Specifies a loop iteration limit in VHDL for a `while` loop in the design when the loop index is a variable, not a constant.

### Description

VHDL only. For Verilog applications use the `loop_limit` directive (see [loop\\_limit, on page 489](#)).

The `syn_looplmit` directive specifies a loop iteration limit for a `while` loop on a per-loop basis, when the loop index is a variable, not a constant. If your design requires a variable loop index, use the `syn_looplmit` directive to specify a limit for the compiler. If you do not, you can get a “while loop not terminating” compiler error.

The limit cannot be an expression. The higher the value you set, the longer the runtime.

Alternatively, you can use the option set `looplmit` command (Loop Limit GUI option) to set a global loop limit that overrides the default setting of 2000 loops.

### syn\_looplmit Syntax

|      |                                                                                                          |                              |
|------|----------------------------------------------------------------------------------------------------------|------------------------------|
| VHDL | <b>attribute syn_looplmit : integer;</b><br><b>attribute syn_looplmit of labelName : label is value;</b> | <a href="#">VHDL Example</a> |
|------|----------------------------------------------------------------------------------------------------------|------------------------------|

---

### VHDL Example

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

entity test is
    port (clk : in std_logic;
          d_in : in std_logic_vector(2999 downto 0);
          d_out: out std_logic_vector(2999 downto 0) );
end test;
```

```
architecture beh of test is
attribute syn_looplimit : integer;
attribute syn_looplimit of loopabc: label is 3000;

begin
  process (clk)
    variable i, k: integer := 0;
  begin
    if (clk'event and clk = '1') then
      k:=0;
      loopabc: while (k<2999) loop
        k:= k+ 1;
        d_out(k) <= d_in(k);
      end loop loopabc;
      d_out(0) <= d_in(0);
    end if;
  end process;
end beh;
```

## syn\_macro

### Attribute

Determines whether instantiated macros are optimized and included in the final output netlist.

### Description

The syn\_macro attribute determines whether the macros are optimized and whether the contents of instantiated macros are included in the final output netlist. You can use this attribute with IP cores (edn, ngo, ngc, or structural Verilog netlist). The attribute can only be set on an instantiated macro or IP core in a constraint file, and you must set this attribute on a view, not an instance.

The effect of syn\_macro varies according to whether the core is encrypted or not. For details, see [syn\\_macro Setting and Cores, on page 680](#) and [syn\\_macro, syn\\_user\\_instance, Enhanced Optimization and Cores, on page 680](#).

### syn\_macro Syntax

|         |                                                                        |                    |
|---------|------------------------------------------------------------------------|--------------------|
| FDC     | <code>define_attribute {v:macroName} syn_macro {1   0}</code>          | Constraint Example |
| Verilog | <code>module /* synthesis syn_macro = 1  0 */</code>                   | Verilog Example    |
| VHDL    | <code>attribute syn_macro of component : label is true   false;</code> | VHDL Example       |

In the above syntax, the Boolean values are defined as outlined in the following table:

| Value                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0   false<br>(default) | <ul style="list-style-type: none"> <li>• Macro optimization:<br/>Instantiated macros are optimized, but this behavior can be affected by the Enhanced Optimization option and the <code>syn_user_instance</code> attribute. See <a href="#">syn_macro, syn_user_instance, Enhanced Optimization and Cores , on page 680</a>.</li> <li>• Macro netlist:<br/>Core netlist generation is affected by the <code>syn_user_instance</code> attribute. See <a href="#">syn_macro, syn_user_instance, Enhanced Optimization and Cores , on page 680</a> for details.</li> </ul>                                                                                                                                                                                        |
| 1   true               | <ul style="list-style-type: none"> <li>• Macro optimization:<br/>Instantiated primitives are not optimized with associated edn, ngo, or nge files. The contents and the macro boundary are preserved and treated as a white box using the timing and resource utilization information from these files. Optimization is affected by the Enhanced Optimization option and <code>syn_user_instance</code> attribute. See <a href="#">syn_macro, syn_user_instance, Enhanced Optimization and Cores , on page 680</a>.</li> <li>• Macro netlist:<br/>Core netlist generation is affected by the <code>syn_user_instance</code> attribute. See <a href="#">syn_macro, syn_user_instance, Enhanced Optimization and Cores , on page 680</a> for details.</li> </ul> |

## Constraint Example

The following example prevents optimizations for the instantiated fifo macro. You must set the attribute on the view (`v:`) for the macro or IP core in the constraint file; if you set it on another object, such as an instance, the attribute will not work.

```
define_attribute {v:fifo} syn_macro {1}
```

## Verilog Example

The attribute must be set on a module:

```
module /* synthesis syn_macro = 1 */
```

If the module has the `syn_black_box` attribute:

```
module /* synthesis syn_black_box syn_macro = 1 |0 */
```

## VHDL Example

You must specify `syn_macro` on the component name. The following code specifies `syn_macro` correctly on the component name:

```
architecture top of top is
component decoder_macro
    port (clk : in bit;
          opcode : in bit_vector(2 downto 0);
          a : in bit_vector(7 downto 0);
          data0 : out bit_vector(7 downto 0) );
end component;

attribute syn_macro : boolean;
attribute syn_macro of decoder_macro : label is true;
```

## `syn_macro` Setting and Cores

The `syn_macro` setting affects how encrypted and unencrypted cores are handled:

| Core        | <code>syn_macro</code> | Core Optimization | Core Netlist                       |
|-------------|------------------------|-------------------|------------------------------------|
| Unencrypted | 0                      | No                | Absorbed in top-level netlist      |
|             | 1                      | No                | Not absorbed in top level netlist  |
| Encrypted   | 0                      | No                | Separate encrypted core netlist    |
|             | 1                      | No                | No separate encrypted core netlist |

## `syn_macro`, `syn_user_instance`, Enhanced Optimization and Cores

Like `syn_macro`, the `syn_user_instance` attribute ([syn\\_user\\_instance, on page 863](#)) affects whether a macro is optimized and how core netlists are written. As both `syn_macro` and `syn_user_instance` are supported, follow these usage guidelines when using the two attributes:

- Use `syn_macro` to disable optimizations to the core and to prevent the core contents from being written to the top-level output netlist (unencrypted cores) or as a separate netlist (encrypted cores).

- Use `syn_user_instance` to disable optimizations to the core, but still write out the contents as appropriate for encrypted and unencrypted cores.

If both attributes are applied to the same object, the `syn_user_instance` attribute is ignored. In addition, the Enhanced Optimization option affects whether cores are optimized. The following table summarizes the behavior of these variables when applied to cores that only contain primitives:

#### Unencrypted Cores, Enhanced Optimization = 0

| <code>syn_user_instance</code> | <code>syn_macro</code> | Optimization | Core Netlist                      |
|--------------------------------|------------------------|--------------|-----------------------------------|
| 0                              | 0                      | No           | Absorbed in top-level netlist     |
| 0                              | 1                      | No           | Not absorbed in top level netlist |
| 1                              | 0                      | No           | Absorbed in top-level netlist     |
| 1                              | 1                      | No           | Absorbed in top-level netlist     |

#### Unencrypted Cores, Enhanced Optimization = 1

| <code>syn_user_instance</code> | <code>syn_macro</code> | Optimization | Core Netlist                      |
|--------------------------------|------------------------|--------------|-----------------------------------|
| 0                              | 0                      | Yes          | Absorbed in top-level netlist     |
| 0                              | 1                      | No           | Not absorbed in top level netlist |
| 1                              | 0                      | No           | Absorbed in top-level netlist     |
| 1                              | 1                      | No           | Absorbed in top-level netlist     |

#### Encrypted Cores, Enhanced Optimization = 0

| <code>syn_user_instance</code> | <code>syn_macro</code> | Optimization | Core Netlist               |
|--------------------------------|------------------------|--------------|----------------------------|
| 0                              | 0                      | No           | Separate encrypted netlist |
| 0                              | 1                      | No           | No separate core netlist   |
| 1                              | 0                      | No           | Separate encrypted netlist |
| 1                              | 1                      | No           | Separate encrypted netlist |

#### Encrypted Cores, Enhanced Optimization = 1

| <code>syn_user_instance</code> | <code>syn_macro</code> | Optimization | Core Netlist               |
|--------------------------------|------------------------|--------------|----------------------------|
| 0                              | 0                      | Yes          | Separate encrypted netlist |

|   |   |    |                            |
|---|---|----|----------------------------|
| 0 | 1 | No | No separate core netlist   |
| 1 | 0 | No | Separate encrypted netlist |
| 1 | 1 | No | Separate encrypted netlist |

## syn\_map\_dffrs

### Attribute

Allows the synthesis software to handle registers with both asynchronous set and asynchronous reset.

### Description

If this attribute is set to 0 and the design contains registers with asynchronous set and reset, the synthesis software generates an error message:

`@W: FX707 : Register regName has both asynchronous set and reset. Your design may not work on the board. Xilinx recommends you change your RTL.`

This warning is only issued once, although it may apply to other registers as well.

You must set the attribute `syn_map_dffrs` to 1 for the top-level design. When you set this attribute to 1, instead of generating an error, the software takes either the set or reset and maps it to logic which might affect the quality of results. The default value is 1.

### Syntax Specification

|         |                                                                              |                                            |
|---------|------------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_global_attribute {syn_map_dffrs} {1 0}</code>                   | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_map_dffrs = 1 0 */;</code>                     | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_map_dffrs of object : objectType is true   false;</code> | <a href="#">VHDL Example</a>               |

### Constraints Editor Example

|   | Enabled                             | Object Type | Object   | Attribute     | Value | Val Type | Description |
|---|-------------------------------------|-------------|----------|---------------|-------|----------|-------------|
| 1 | <input checked="" type="checkbox"/> |             | <global> | syn_map_dffrs | 0     |          |             |

## Verilog Example

```
moduledff_aload(q, d, ad, clk, load)
/*synthesis syn_map_dffrs = 0*/;
input d, ad, clk, load;
output q;
reg q;

always @ (posedge clk or posedge load)
begin
    if (load)
        q = ad;
    else
        q = d;
end
endmodule
```

## VHDL Example

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

ENTITY clatches IS
    PORT (d,c,r,s : IN std_logic;
          qrs: OUT std_logic );
END clatches;

ARCHITECTURE behave OF clatches IS
SIGNAL qrs_i: std_logic;
attribute syn_map_dffrs : boolean;
attribute syn_map_dffrs of behave : architecture is true;
BEGIN
    qrs_i <= '0' WHEN r = '1' ELSE
    '1' WHEN s = '1' ELSE
    d WHEN c = '1' ELSE
    qrs_i;
    qrs <= qrs_i;
END BEHAVE;
```

## Effect of using syn\_map\_dffrs

The following shows the content in log file, before applying the attribute:

**Verilog**    module dff\_aload(q, d, ad, clk, load)  
/\*synthesis syn\_map\_dffrs = 1\*/;

**VHDL**    attribute syn\_map\_dffrs of behave: architecture is true;

---

```
# Tue Jun 12 11:40:00 2012
#####
***** Premap Report *****
Synopsys Xilinx Technology Pre-mapping, Version map630dev, Build 633R, Built Jun 8 2012 22:05:25
Copyright (C) 1994-2012, Synopsys, Inc. This software and the associated documentation are confidential and proprietary to Synopsys, Inc. Your use or disclosure of this software subject to the terms and conditions of a written license agreement between you, or your company, and Synopsys, Inc.
Product Version F-2012.09

Mapper Startup Complete (Time elapsed 0h:00m:00s; Memory used current: 91MB peak: 92MB)
Reading constraint file: C:\syn_map_dffrs\syn_map_dffrs_0.sdc
Adding property syn_map_dffrs, value 1 to viewnetwork_dff_aload(versilog)
Writing Xilinx I/O parameter type table from file <R:\syn201209_098\lib\xilinx\v_i_o_tb1.txt>
Priming clock summary report in "C:\syn_map_dffrs\rev_1\syn_map_dffrs_scck.rpt" file
IN: MF248 |Running in 64-bit mode.
IN: MF257 |Gated clock conversion enabled
IN: MF546 |Generated clock conversion enabled

Design Input Complete (Time elapsed 0h:00m:00s; Memory used current: 91MB peak: 92MB)
Mapper Initialization Complete (Time elapsed 0h:00m:00s; Memory used current: 93MB peak: 92MB)
Start loading timing files (Time elapsed 0h:00m:00s; Memory used current: 92MB peak: 94MB)
Finished loading timing files (Time elapsed 0h:00m:00s; Memory used current: 92MB peak: 94MB)

Reading Xilinx I/O pad type table from file <R:\syn201209_098\lib\xilinx\v_i_o_tb1.txt>
IN: FX707 :c:\syn_map_dffrs\syn_map_dffrs.v:6:0:6:5!Register unl_d has both asynchronous set and reset. Your design may not work on the board. Xilinx recommends you change your RTL.

Clock Summary
*****
Start Requested Requested Clock Clock
Clock Frequency Period Type Type
-----
System 1.0 MHz 1000.000 system system_clockgroup
dff_aloadclk 1.0 MHz 1000.000 inferred Inferred_clockgroup_0

IN: MTS29 :c:\syn_map_dffrs\syn_map_dffrs.v:6:0:6:5!Found inferred clock dff_aloadclk which controls 2 sequential elements including unl_d. This clock has no specified timing constraint which may prevent conversion of gated or generated clocks and may adversely impact design performance.

syn_allowed_resources : blockrams=2060 set on top level netlist dff_aload
Finished Pre Mapping Phase. (Time elapsed 0h:00m:00s; Memory used current: 174MB peak: 200MB)
IN: BN225 |Writing default property annotation file C:\Users\anantnag\attributes\syn_map_dffrs\xilinx\syn_map_dffrs_0\rev_1\syn_map_dffrs.sap.
Pre-mapping successful!

Mapper Exit (Time elapsed 0h:00m:00s; Memory used current: 112MB peak: 200MB)

Process took 0h:00m:00s realtime, 0h:00m:01s cputime
# Tue Jun 12 11:40:03 2012
#####
```

---

The following shows the content in log file, after applying the attribute:

---

**Verilog** module dff\_aload(q, d, ad, clk, load)  
/\*synthesis syn\_map\_dffrs = 0\*/;

---

**VHDL** attribute syn\_map\_dffrs of behave: architecture is false;

---

```
# Tue Jun 12 11:40:00 2012
#####
Premap Report

Synopsys Xilinx Technology Pre-mapping, Version map60dev, Build 613R, Built: Jun 8 2012 22:05:25
Copyright (C) 1994-2012, Synopsys, Inc. This software and the associated documentation are confidential and proprietary to Synopsys, Inc. Your use or disclosure of this software subject to the terms and conditions of a written license agreement between you, or your company, and Synopsys, Inc.
Product Version F-2012.09

Mapper Startup Complete (Time elapsed 0h:00m:00s; Memory used current: 91MB peak: 92MB)

Reading constraint file: C:\syn_map_dffrs\syn_map_dffrs_0.sdc
Adding property syn_map_dffrs, value 1 to vieworkflow.dff_aLoad(verilog)
@L: C:\syn_map_dffrs\rev\syn_map_dffrs.scck.rpt
Printing clock summary report in "C:\syn_map_dffrs\rev\syn_map_dffrs_scck.rpt" file
@N: MF248 |Running in 64-bit mode.
@N: MF257 |Cated clock conversion enabled
@N: MF346 |Generated clock conversion enabled

Design Input Complete (Time elapsed 0h:00m:00s; Memory used current: 91MB peak: 92MB)
Mapper Initialization Complete (Time elapsed 0h:00m:00s; Memory used current: 91MB peak: 92MB)
Start loading timing files (Time elapsed 0h:00m:00s; Memory used current: 92MB peak: 92MB)
Finished timing files (Time elapsed 0h:00m:00s; Memory used current: 92MB peak: 94MB)

Reading Xilinx I/O pad type table from file <C:\nt\syn201209_098R\lib\xilinx\x_i_o_tb1.txt>
Reading Xilinx Rocket I/O parameter type table from file <C:\nt\syn201209_098R\lib\xilinx\gttype.txt>
@W: FX707 :<C:\syn_map_dffrs\syn_map_dffrs.v>:6:0:6|Register uni_d has both asynchronous set and reset. Your design may not work on the board. Xilinx recommends you change your RTL.

Clock Summary
*****
Start Requested Requested Clock Clock
Clock Frequency Period Type Group
-----
System 1.0 MHz 1000.000 system system_clkgroup
dff_aload|clk 1.0 MHz 1000.000 inferred Inferred_clkgroup_0

@W: HT529 :<C:\syn_map_dffrs\syn_map_dffrs.v>:6:0:6|Found inferred clock dff_aload|clk which controls 2 sequential elements including uni_d. This clock has no specified timing constraint which may prevent conversion of gated or generated clocks and may adversely impact design performance.

syn_allowed_resources : blockrams=2060 set top level netlist dff_aload

Finished Pre Mapping Phase. (Time elapsed 0h:00m:00s; Memory used current: 174MB peak: 200MB)

@N: BN225 |Writing default property annotation file C:\Users\anantnag\attributes\syn_map_dffrs\xilinx\syn_map_dffrs_0\rev\syn_map_dffrs.sap.
Pre-mapping successful!

At Mapper Exit (Time elapsed 0h:00m:00s; Memory used current: 112MB peak: 200MB)

Process took 0h:00m:01s realtime, 0h:00m:01s cputime
# Tue Jun 12 11:40:03 2012
#####
```

---

## **syn\_maxfan**

### *Attribute*

Overrides the default (global) fanout guide for an individual input port, net, or register output.

### **Description**

`syn_maxfan` overrides the global fanout for an individual input port, net, or register output. You set the default Fanout Guide for a design through the option `set_fanout_limit` command. Use the `syn_maxfan` attribute to override the global default and specify a different (local) value for individual I/Os.

Generally, `syn_maxfan` and the default fanout guide are suggested guidelines only, but in certain cases they function as hard limits.

- When they are guidelines, the synthesis tool takes them into account, but does not always respect them absolutely. The synthesis tool does not respect the `syn_maxfan` limit if the limit imposes constraints that interfere with optimization.
- The attribute value functions as a hard limit when it is attached to nets, ports, primitive instances, and registers.

You can apply the `syn_maxfan` attribute to the following objects:

- Registers or instances.
- Ports or nets. If you apply the attribute to a net, the synthesis tool creates a `KEEPBUF` component and attaches the attribute to it to prevent the net itself from being optimized away during synthesis.

The `syn_maxfan` attribute is often used along with the `syn_noclockbuf` attribute on an input port that you do not want buffered. There are a limited number of clock buffers in a design, so if you want to save these special clock buffer resources for other clock inputs, put the `syn_noclockbuf` attribute on the clock signal. If timing for that clock signal is not critical, you can turn off buffering completely to save area. To turn off buffering, set the maximum fanout to a very high number; for example, 1000. Note, do not use the `syn_maxfan` attribute with the Synthesis Strategy fast option.

Similarly, you use `syn_maxfan` with the `syn_replicate` attribute in certain technologies to control replication.

## syn\_maxfan Syntax

|         |                                                                       |                                            |
|---------|-----------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute {object} syn_maxfan {value}</code>             | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_maxfan = "value" */;</code>             | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_maxfan of object : objectType is "value" ;</code> | <a href="#">VHDL Example</a>               |

## Constraints Editor Example

| Enable                              | Object Type | Object   | Attribute  | Value | Value Type | Description              |
|-------------------------------------|-------------|----------|------------|-------|------------|--------------------------|
| <input checked="" type="checkbox"/> | <any>       | <Global> | syn_maxfan | 1     | integer    | Overrides the default... |

## Verilog Example

```
module syn_maxfan (clk,rst,a,b,c);
    input clk,rst;
    input [7:0] a,b;
    output reg [7:0] c;
    reg d/* synthesis syn_maxfan=3 */;

    always @ (posedge clk)
        begin
            if(rst)
                d <= 0;
            else
                d <= ~d;
        end

    always @ (posedge d)
        begin
            c <= a^b;
        end
endmodule
```

## VHDL Example

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

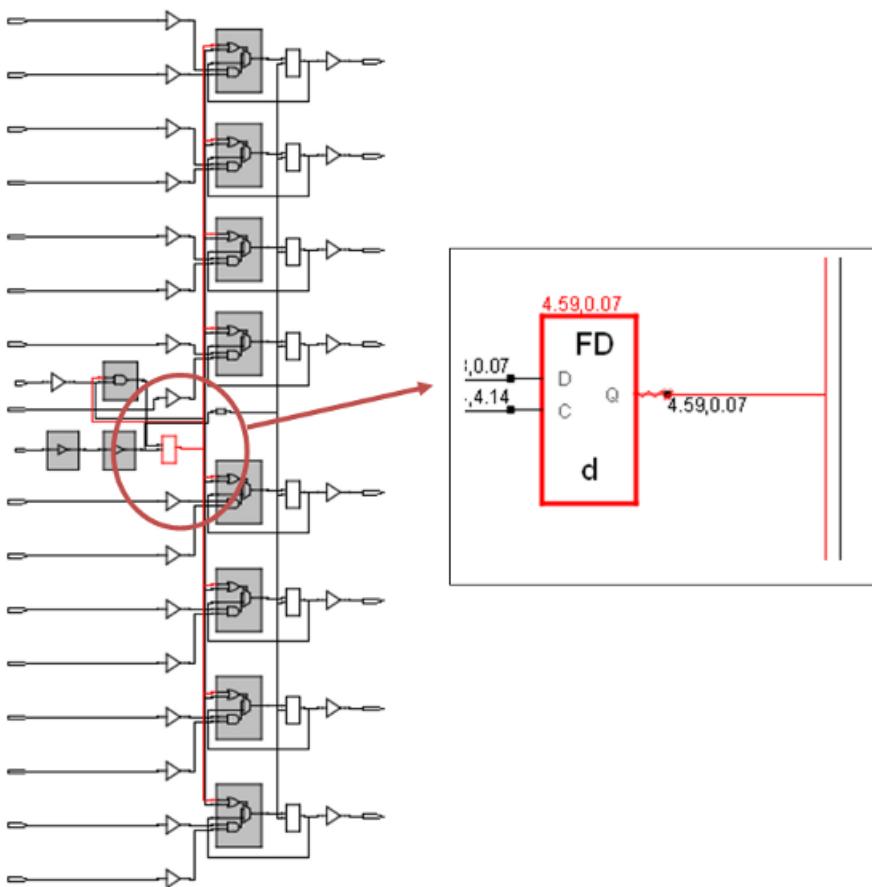
```
entity maxfan is
    port (a : in std_logic_vector(7 downto 0);
          b : in std_logic_vector(7 downto 0);
          rst : in std_logic;
          clk : in std_logic;
          c : out std_logic_vector(7 downto 0) );
end maxfan;

architecture rtl of maxfan is
signal d : std_logic;
attribute syn_maxfan : integer;
attribute syn_maxfan of d : signal is 3;
begin
process (clk)
begin
    if (clk'event and clk = '1') then
        if (rst = '1') then
            d <= '0';
        else
            d <= not d;
        end if;
    end if;
end process;

process (d)
begin
    if (d'event and d = '1') then
        c <= a and b;
    end if;
end process;
end rtl;
```

## Effect of Using `syn_maxfan`

Before applying `syn_maxfan`:



After applying the `syn_maxfan` attribute, register **d** is replicated three times (shown in red) because its actual fanout is 8, but it is restricted to 3.

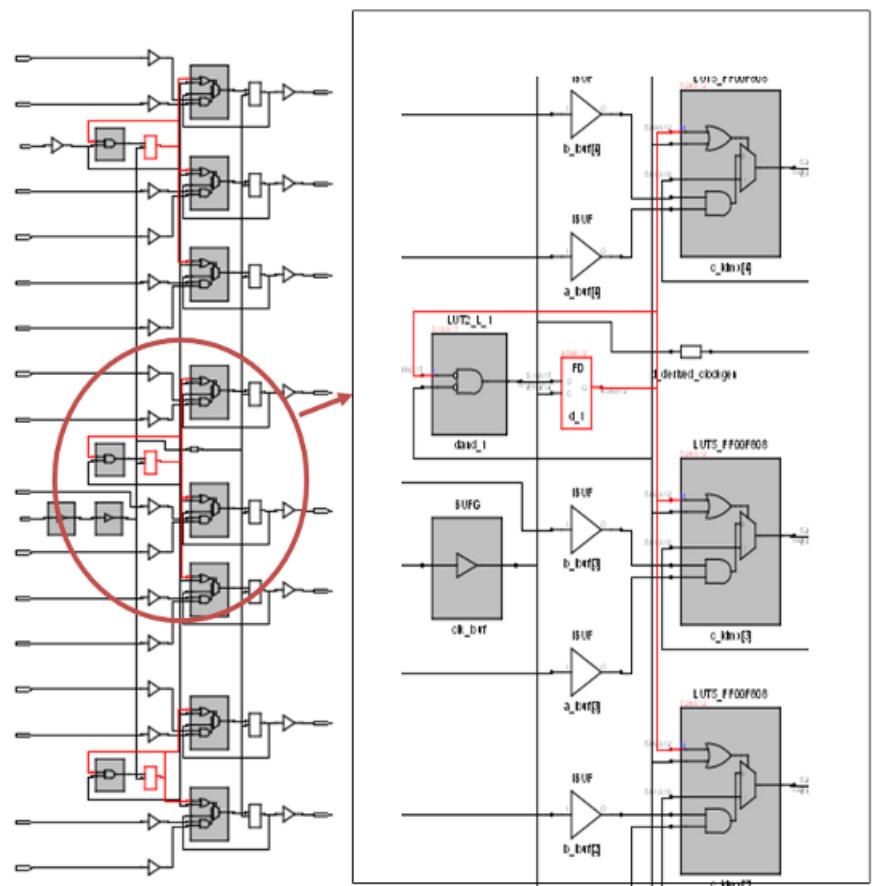
---

Verilog      `reg d/* synthesis syn_maxfan=3 */;`

---

VHDL      `attribute syn_maxfan of d : signal is 3;`

---



## syn\_multstyle

### Attribute

Determines how multipliers are implemented.

### Description

This attribute specifies if multipliers are implemented as dedicated hardware blocks or as logic.

### syn\_multstyle and syn\_DSPstyle

syn\_multstyle results in the same logic or block multiplier implementations as syn\_DSPstyle ([syn\\_DSPstyle, on page 579](#)) when applied on simple multipliers that have dedicated DSP resources. Typically, you use syn\_multstyle for older technologies that do not support DSPs.

### syn\_multstyle Syntax

|         |                                                                                  |                                    |
|---------|----------------------------------------------------------------------------------|------------------------------------|
| FDC     | <code>define_attribute {instance} syn_multstyle {block_mult logic}</code>        | <a href="#">Constraints Editor</a> |
|         | Global attribute:                                                                | <a href="#">Example</a>            |
|         | <code>define_global_attribute syn_multstyle {block_mult logic}</code>            |                                    |
| Verilog | <code>input netName /* synthesis syn_multstyle = "block_mult logic" */;</code>   | <a href="#">Verilog Example</a>    |
| VHDL    | <code>attribute syn_multstyle of instance : signal is "block_mult logic";</code> | <a href="#">VHDL Example</a>       |

In the above syntax, block\_mult implements the multipliers as dedicated block multipliers (the default), and logic implements the multipliers as logic instead of dedicated resources.

### Constraints Editor Example

This constraints editor example specifies that the multipliers be globally implemented as logic:

|   | Enable                              | Object Type | Object   | Attribute     | Value | Value Type | Description                           |
|---|-------------------------------------|-------------|----------|---------------|-------|------------|---------------------------------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Glob... | syn_multstyle | logic | string     | Special implementation of multipliers |
| 2 |                                     |             |          |               |       |            |                                       |

This FDC file example specifies that multipliers be implemented as logic.

```
define_attribute {temp[15:0]} syn_multstyle {logic}
```

## Verilog Example

```
module mult(a,b,c,r,en);
  input [7:0] a,b;
  output [15:0] r;
  input [15:0] c;
  input en;
  wire [15:0] temp /* synthesis syn_multstyle="logic" */;
  assign temp = a*b;
  assign r = en ? temp: c;
endmodule
```

## VHDL Example

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

entity mult is
  port (clk : in std_logic ;
        a : in std_logic_vector(7 downto 0);
        b : in std_logic_vector(7 downto 0);
        c : out std_logic_vector(15 downto 0) );
end mults;

architecture rtl of mult is
  signal mult_i : std_logic_vector(15 downto 0);
  attribute syn_multstyle : string ;
  attribute syn_multstyle of mult_i : signal is "logic";
begin
  mult_i <= std_logic_vector(unsigned(a)*unsigned(b));
```

```
process(clk)
begin
    if (clk'event and clk = '1') then
        c <= mult_i;
    end if;
end process;
end rtl;
```

## **syn\_netlist\_hierarchy**

*Attribute*

Determines if the generated netlist is to be hierarchical or flat.

### **Description**

A global attribute that controls the generation of hierarchy in the EDIF output netlist when assigned to the top-level module in your design. The default (1/true) allows hierarchy generation, and setting the attribute to 0/false flattens the hierarchy and produces a completely flattened output netlist.

### **Syntax Specification**

|         |                                                                                        |                                            |
|---------|----------------------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_global_attribute syn_netlist_hierarchy {0 1}</code>                       | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_netlist_hierarchy = 0 1 */;</code>                       | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_netlist_hierarchy of object :<br/>objectType is true false;</code> | <a href="#">VHDL Example</a>               |

### **Constraints Editor Example**

| Enable                              | Object Type | Object   | Attribute             | Value | Value Type | Description                     |
|-------------------------------------|-------------|----------|-----------------------|-------|------------|---------------------------------|
| <input checked="" type="checkbox"/> | global      | <Global> | syn_netlist_hierarchy | 1     | boolean    | Enable hierarchy reconstruction |

## Verilog Example

```
module fu_add(input a,b,cin,output su,cy);
  assign su = a ^ b ^ cin;
  assign cy = (a & b) | ((a^b) & cin);
endmodule 4

module rca_adder#(parameter width =4)
  (input [width-1:0]A,B,input CIN,
   output [width-1:0]SU,output COUT );
  wire [width-2:0]CY;
  fu_add FA0(.su(SU[0]),.cy(CY[0]),.cin(CIN),.a(A[0]),.b(B[0]));
  fu_add FA1(.su(SU[1]),.cy(CY[1]),.cin(CY[0]),.a(A[1]),.b(B[1]));
  fu_add FA2(.su(SU[2]),.cy(CY[2]),.cin(CY[1]),.a(A[2]),.b(B[2]));
  fu_add FA3(.su(SU[3]),.cy(COUT),.cin(CY[2]),.a(A[3]),.b(B[3]));
endmodule

module rp_top#(parameter width =16)
  (input [width-1:0]A1,B1,input CIN1,
   output [width- 1:0]SUM,output COUT1) /*synthesis
   syn_netlist_hierarchy=0*/;
  wire[2:0]CY1;
  rca_adder RA0 (.SU(SUM[3:0]),.COUT(CY1[0]),.CIN(CIN1),
    .A(A1[3:0]),.B(B1[3:0]));
  rca_adder RA1(.SU(SUM[7:4]),.COUT(CY1[1]),.CIN(CY1[0]),
    .A(A1[7:4]),.B(B1[7]));
  rca_adder RA2 (.SU(SUM[11:8]),.COUT(CY1[2]),.CIN(CY1[1]),
    .A(A1[11:8]),.B(B1[11:8]));
  rca_adder RA3(.SU(SUM[15:12]),.COUT(COUT1),.CIN(CY1[2]),
    .A(A1[15:12]),.B(B1[15:12]));
endmodule
```

## VHDL Example

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

entity FULLADDER is
    port (a, b, c : in std_logic;
          sum, carry: out std_logic );
end FULLADDER;

architecture fulladder_behav of FULLADDER is
begin
    sum <= (a xor b) xor c ;
    carry <= (a and b) or (c and (a xor b));
end fulladder_behav;

library ieee;
use ieee.std_logic_1164.all;

entity FOURBITADD is
    port (a, b : in std_logic_vector(3 downto 0);
          Cin  : in std_logic;
          sum  : out std_logic_vector (3 downto 0);
          Cout, V : out std_logic );
end FOURBITADD;

architecture fouradder_structure of FOURBITADD is
signal c: std_logic_vector (4 downto 1);
component FULLADDER
    port (a, b, c: in std_logic;
          sum, carry: out std_logic );
end component;
begin
    FA0: FULLADDER
        port map (a(0), b(0), Cin, sum(0), c(1));
    FA1: FULLADDER
        port map (a(1), b(1), C(1), sum(1), c(2));
    FA2: FULLADDER
        port map (a(2), b(2), C(2), sum(2), c(3));
    FA3: FULLADDER
        port map (a(3), b(3), C(3), sum(3), c(4));
    V <= c(3) xor c(4);
    Cout <= c(4);
end fouradder_structure;
```

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

entity BITADD is
    port (A, B: in std_logic_vector(15 downto 0);
          Cin : in std_logic;
          SUM : out std_logic_vector (15 downto 0);
          COUT: out std_logic );
end BITADD;

architecture adder_structure of BITADD is
attribute syn_netlist_hierarchy : boolean;
attribute syn_netlist_hierarchy of adder_structure:
    architecture is false;
signal C: std_logic_vector (4 downto 1);

component FOURBITADD
    port (a, b: in std_logic_vector(3 downto 0);
          Cin : in std_logic;
          sum : out std_logic_vector (3 downto 0);
          Cout, V: out std_logic );
end component;

begin
    F1: FOURBITADD
        port map (A(3 downto 0),B(3 downto 0),
                  Cin, SUM(3 downto 0),C(1));
    F2: FOURBITADD
        port map (A(7 downto 4),B(7 downto 4),
                  C(1), SUM(7 downto 4),C(2));
    F3: FOURBITADD
        port map (A(11 downto 8),B(11 downto 8),
                  C(2), SUM(11 downto 8),C(3));
    F4: FOURBITADD
        port map (A(15 downto 12),B(15 downto 12),
                  C(3), SUM(15 downto 12),C(4));
    COUT <= c(4);
end adder_structure;
```

## Effect of Using syn\_netlist\_hierarchy

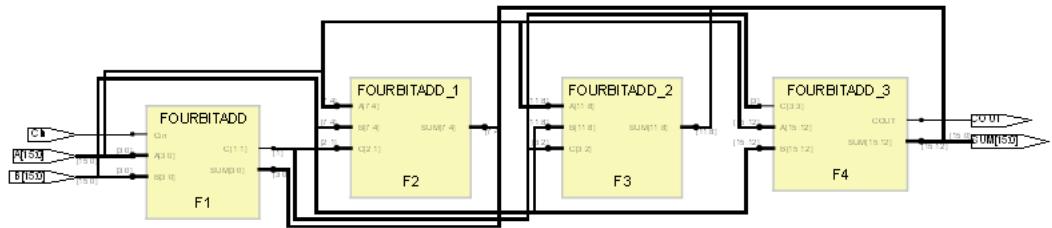
Without applying the attribute (default is to allow hierarchy generation) or setting the attribute to 1/true creates a hierarchical netlist.

**Verilog**

```
output[width-1:0]SUM, output COUT1)
/*synthesis syn_netlist_hierarchy=1*/;
```

**VHDL**

```
attribute syn_netlist_hierarchy of adder_structure :
architecture is true;
```

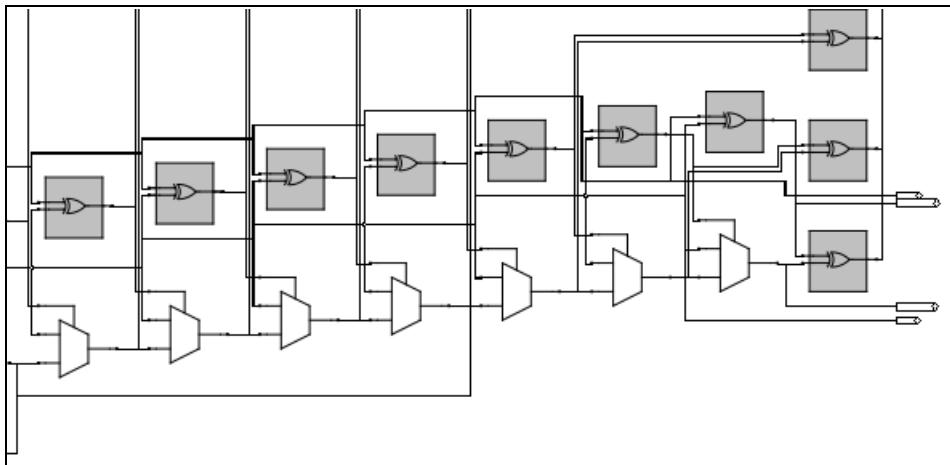


Applying the attribute with a value of 0/false creates a flattened netlist.

**Verilog**      output [width-1:0]SUM, output COUT1)  
/\*synthesis syn\_netlist\_hierarchy=0\*/;

**VHDL**      attribute syn\_netlist\_hierarchy of adder\_structure :  
architecture is false;

---



### **syn\_hier flatten and syn\_netlist\_hierarchy**

The `syn_hier=flatten` attribute and the `syn_netlist_hierarchy=false` attributes both flatten hierarchy, but work slightly differently. Use the `syn_netlist_hierarchy` attribute if you want a completely flattened netlist (this attribute flattens all levels of hierarchy). When you set `syn_hier=flatten`, you flatten the hierarchical levels below the component on which it is set, but you do not flatten the current hierarchical level where it is set. Refer to [syn\\_hier, on page 642](#) for information about this attribute.

## **syn\_noarrayports**

### *Attribute*

Specifies signals as scalar in the output file.

### **Description**

Use this attribute to specify that the ports of a design unit are to be treated as individual signals (scalars), not as buses (arrays) in the output file.

### **syn\_noarrayports Syntax**

|          |                                                |
|----------|------------------------------------------------|
| FDC File | define_global_attribute syn_noarrayports {0 1} |
|----------|------------------------------------------------|

|         |                                              |
|---------|----------------------------------------------|
| Verilog | object /* synthesis syn_noarrayports = 0 1 ; |
|---------|----------------------------------------------|

|      |                                                                   |
|------|-------------------------------------------------------------------|
| VHDL | attribute syn_noarrayports of object : objectType is true false ; |
|------|-------------------------------------------------------------------|

### **Constraints Editor Example**

|   | Enabled                             | Object Type | Object   | Attribute        | Value | Val Type | Description         | Comment |
|---|-------------------------------------|-------------|----------|------------------|-------|----------|---------------------|---------|
| 1 | <input checked="" type="checkbox"/> | global      | <global> | syn_noarrayports | 1     | boolean  | Disable array ports |         |

### **Verilog Example**

```
module adder8(cout,sum,a,b,cin)
    /* synthesis syn_noarrayports = "1" */;
    input[7:0] a,b;
    input cin;
    output reg[7:0] sum;
    output reg cout;

    always@(*)
    begin
        {cout,sum}=a+b+cin;
    end
endmodule
```

## VHDL Example

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ADDER is
generic(n: natural :=8);
port (A: in std_logic_vector(n-1 downto 0);
      B: in std_logic_vector(n-1 downto 0);
      carry: out std_logic;
      sum: out std_logic_vector(n-1 downto 0) );
end ADDER;

architecture adder_struct of ADDER is
attribute syn_noarrayports : boolean;
attribute syn_noarrayports of adder_struct : architecture is true;
signal result: std_logic_vector(n downto 0);
begin
  result <= ('0' & A)+('0' & B);
  sum <= result(n-1 downto 0);
  carry <= result(n);
end adder_struct ;

```

## Effect of Using `syn_noarrayports`

This example shows the netlist before applying the attribute:

---

Verilog module adder8(cout,sum,a,b,cin)/\* synthesis syn\_noarrayport="0" \*/

---

VHDL attribute syn\_noarrayports : boolean;  
attribute syn\_noarrayports of adder\_struct : architecture is false;

---

Netlist (library work  
 (edifLevel 0)  
 (technology (numberDefinition ))  
 (cell ADDER (cellType GENERIC)  
 (view behv (viewType NETLIST)  
 (interface  
 (port (array (rename A "A(7:0)" 8) (direction INPUT))  
 (port (array (rename B "B(7:0)" 8) (direction INPUT))  
 (port (array (rename sum "sum(7:0)" 8) (direction OUTPUT))  
 (port carry (direction OUTPUT))  
 )

---

This example shows the netlist after applying the attribute:

**Verilog** module adder8(cout,sum,a,b,cin)/\* synthesis syn\_noarrayport="1" \*/

**VHDL** attribute syn\_noarrayports : boolean;  
attribute syn\_noarrayports of adder\_struct : architecture is true;

**Netlist** (library work  
    (edifLevel 0)  
    (technology (numberDefinition ))  
    (cell ADDER (cellType GENERIC)  
        (view behv (viewType NETLIST)  
            (interface  
                (port (rename A\_0 "A(0)") (direction INPUT))  
                (port (rename A\_1 "A(1)") (direction INPUT))  
                (port (rename A\_2 "A(2)") (direction INPUT))  
                (port (rename A\_3 "A(3)") (direction INPUT))  
                (port (rename A\_4 "A(4)") (direction INPUT))  
                (port (rename A\_5 "A(5)") (direction INPUT))  
                (port (rename A\_6 "A(6)") (direction INPUT))  
                (port (rename A\_7 "A(7)") (direction INPUT))  
                (port (rename B\_0 "B(0)") (direction INPUT))  
                (port (rename B\_1 "B(1)") (direction INPUT))  
                (port (rename B\_2 "B(2)") (direction INPUT))  
                (port (rename B\_3 "B(3)") (direction INPUT))  
                (port (rename B\_4 "B(4)") (direction INPUT))  
                (port (rename B\_5 "B(5)") (direction INPUT))  
                (port (rename B\_6 "B(6)") (direction INPUT))  
                (port (rename B\_7 "B(7)") (direction INPUT))  
                (port carry (direction OUTPUT))  
                (port (rename sum\_0 "sum(0)") (direction OUTPUT))  
                (port (rename sum\_1 "sum(1)") (direction OUTPUT))  
                (port (rename sum\_2 "sum(2)") (direction OUTPUT))  
                (port (rename sum\_3 "sum(3)") (direction OUTPUT))  
                (port (rename sum\_4 "sum(4)") (direction OUTPUT))  
                (port (rename sum\_5 "sum(5)") (direction OUTPUT))  
                (port (rename sum\_6 "sum(6)") (direction OUTPUT))  
                (port (rename sum\_7 "sum(7)") (direction OUTPUT))  
            )  
    )

## **syn\_noclockbuf**

### *Attribute*

Turns off automatic clock buffer usage.

### **Description**

The synthesis tool uses clock buffer resources, if they exist in the target module, and puts them on the highest fanout clock nets. You can turn off automatic clock buffer usage by using the `syn_noclockbuf` attribute. For example, you can put a clock buffer on a lower fanout clock that has a higher frequency and a tighter timing constraint.

You can turn off automatic clock buffering for nets or specific input ports. Set the Boolean value to 1 or true to turn off automatic clock buffering.

You can attach this attribute to a port or net in any hard architecture or module whose hierarchy will not be dissolved during optimization.

### **syn\_noclockbuf Syntax**

|          |                                                                                                                          |
|----------|--------------------------------------------------------------------------------------------------------------------------|
| FDC File | <code>define_attribute {object} syn_noclockbuf {0 1}</code><br><code>define_global_attribute syn_noclockbuf {0 1}</code> |
|----------|--------------------------------------------------------------------------------------------------------------------------|

|         |                                                         |
|---------|---------------------------------------------------------|
| Verilog | <code>object /* synthesis syn_noclockbuf = 0 1 ;</code> |
|---------|---------------------------------------------------------|

|      |                                                                              |
|------|------------------------------------------------------------------------------|
| VHDL | <code>attribute syn_noclockbuf of object : objectType is true false ;</code> |
|------|------------------------------------------------------------------------------|

### **Constraints Editor Example**

The `syn_noclockbuf` attribute can be applied in the constraints editor window as shown:

|   | Enabled                             | Object Type | Object   | Attribute      | Value | Val Type | Description             |
|---|-------------------------------------|-------------|----------|----------------|-------|----------|-------------------------|
| 1 | <input checked="" type="checkbox"/> | global      | <global> | syn_noclockbuf | 1     | boolean  | Use normal input buffer |

## Verilog Example

```
module ckbuflg (d,clk,rst,set,q);
  input d,rst,set;
  input clk /*synthesis syn_noclockbuf=1*/;
  output reg q;

  always@ (posedge clk)
  begin
    if(rst)
      q<=0;
    else if(set)
      q<=1;
    else
      q<=d;
  end
endmodule
```

## VHDL Example

```
library IEEE;
use IEEE.std_logic_1164.all;

entity d_ff_srss is
  port (d,clk,reset,set : in STD_LOGIC;
        q : out STD_LOGIC);
attribute syn_noclockbuf: Boolean;
attribute syn_noclockbuf of clk : signal is false;
end d_ff_srss;

architecture d_ff_srss of d_ff_srss is
begin
  process(clk)
  begin
    if clk'event and clk='1' then
      if reset='1' then
        q <= '0';
      elsif set='1' then
        q <= '1';
      else
        q <= d;
      end if;
    end if;
  end process;
end d_ff_srss;
```

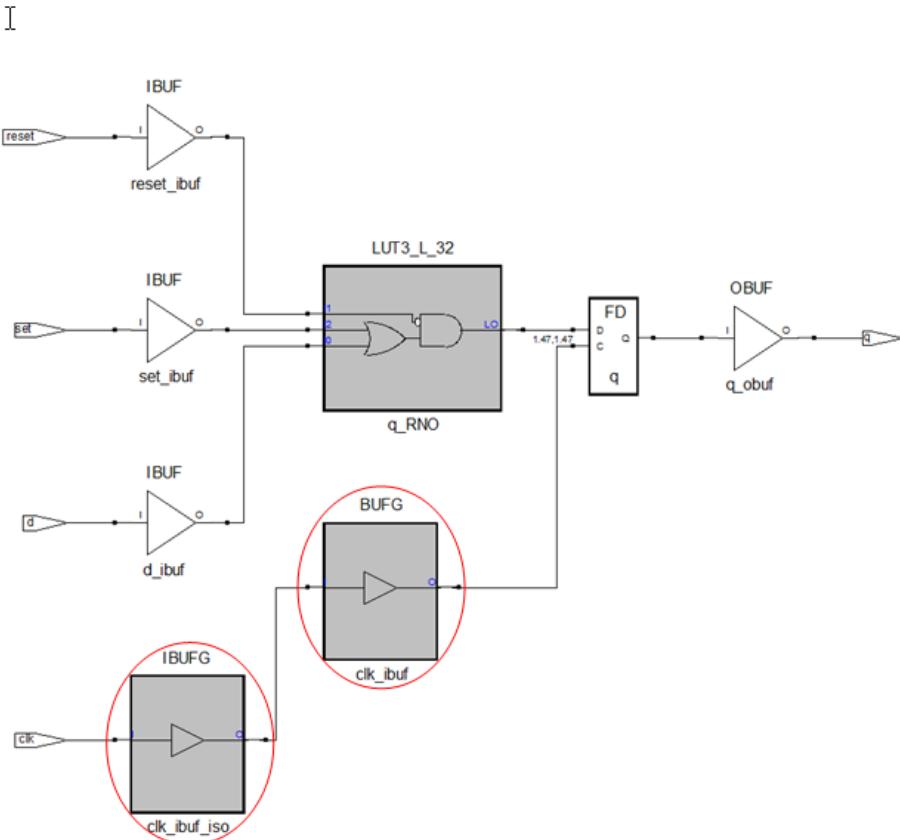
## Effect of Using syn\_noclockbuf

The following graphic shows a design without the `syn_noclockbuf` attribute.

**Verilog**    `input clk /*synthesis syn_noclockbuf = 0*/;`

**VHDL**    `attribute syn_noclockbuf: Boolean;`  
`attribute syn_noclockbuf of clk : signal is false;`

Global buffers are inferred



The following graphic shows a design with the `syn_noclockbuf` attribute.

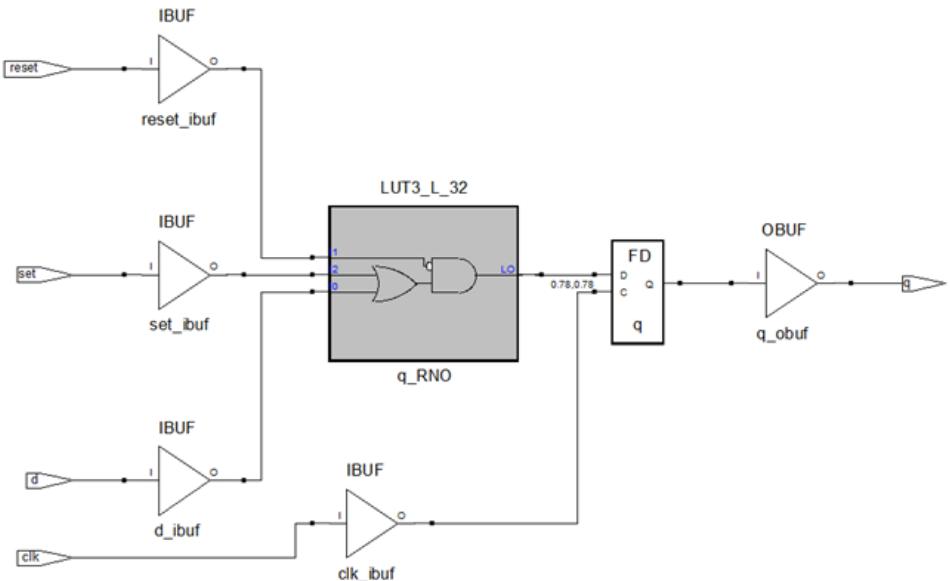
---

Verilog    input clk /\*synthesis syn\_noclockbuf=1\*/;

VHDL    attribute syn\_noclockbuf: Boolean;  
attribute syn\_noclockbuf of clk : signal is true;

---

No global buffers inferred




---

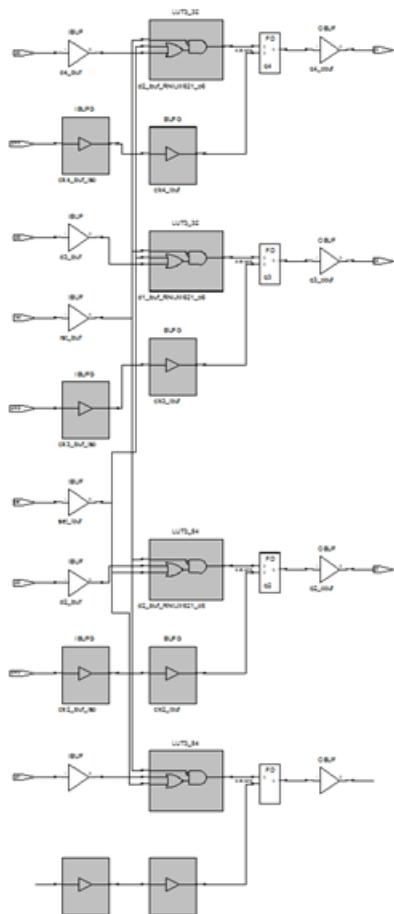
## Global Support

When `syn_noclockbuf` attribute is applied globally, global buffers are inferred by default. If the `syn_noclockbuf` attribute value is set to 1, global buffers are not inferred.

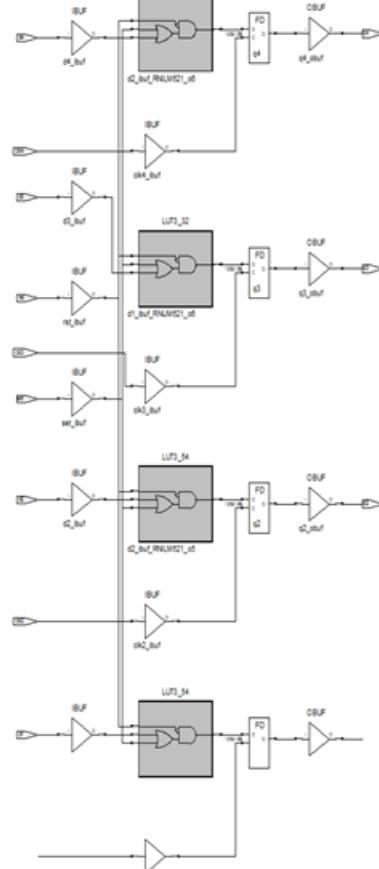
HDL      module ckbuffg (d1,d2,d3,d4,clk1,clk2,clk3,clk4,rst,set,q1,q2,q3,q4)  
/\*synthesis syn\_noclockbuf=1\*/;

FDC      define\_global\_attribute {syn\_noclockbuf} {1}

Before Applying attribute



After applying attribute



## **syn\_no\_compile\_point**

### *Attribute*

Use this attribute with the Automatic Compile Point (ACP) feature. The software automatically identifies modules as compile points in the design based on their size, number of I/Os, and hierarchical levels. However, if you do not want the software to create a compile point for a particular view or module, then apply this attribute.

### **Description**

Use this attribute when the Auto Compile Point option is enabled. The software automatically identifies modules as compile points in the design based on its size, number of I/Os, and hierarchical levels.

However, if you do not want the software to create a compile point for a particular view or module, then apply this attribute. This design view or module is ignored by the Automatic Compile Point software, ensuring that a compile point is not generated for it during synthesis. You must explicitly set this attribute to 1 or true. When you specify `syn_no_compile_point` on a module, be aware that this does not prevent ACP from identifying compile points for other modules instantiated within that module.

### **syn\_no\_compile\_point Syntax**

|         |                                                                                                                                 |                                            |
|---------|---------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute {v:moduleName} syn_no_compile_point {0 1}</code>                                                         | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_no_compile_point = 0 1 */;</code>                                                                 | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_no_compile_point : boolean;<br/>attribute syn_no_compile_point of object : objectType is false true;</code> | <a href="#">VHDL Example</a>               |

In the above syntax, `object` must be a view with the syntax `v:moduleName`. For example:

```
define_attribute {v:fifo} syn_no_compile_point {1}
```

## Constraints Editor Example

| Enable                              | Object Type | Object           | Attribute            | Value | Value Type | Description                           |
|-------------------------------------|-------------|------------------|----------------------|-------|------------|---------------------------------------|
| <input checked="" type="checkbox"/> | <any>       | v:work.prgm_cntr | syn_no_compile_point | 1     | boolean    | Do not mark the view as compile-point |

## Verilog Example

```
module add(input clk, input [4:0]a,[4:0]b, output reg [4:0]dout);
    always@(posedge clk)
    begin
        dout <= a + b;
    end
endmodule

module mult(input clk, input [4:0]a, [4:0]b,
            output reg [9:0]dout)/* synthesis syn_no_compile_point="1" */;

    always@(posedge clk)
    begin
        dout <= a * b;
    end
endmodule
```

## VHDL Example

```
--Multiplier Module
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity mult is
    generic (size: integer :=5);
    port (f_out : out std_logic_vector(9 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          clk : in std_logic );
end;
architecture rtl of mult is
attribute syn_no_compile_point: boolean;
attribute syn_no_compile_point of rtl: architecture is true;
begin
    process (clk)
    begin
```

```

        if (clk'event and clk = '1') then
            f_out <= a * b;
        end if;
    end process;
end;

--Add Module
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity add is
generic (size: integer :=5);
port (f_out : out std_logic_vector(4 downto 0);
      a : in std_logic_vector (size- 1 downto 0);
      b : in std_logic_vector (size- 1 downto 0);
      clk : in std_logic );
end;

architecture rtl of add is
begin
    process (clk)
begin
    if (clk'event and clk = '1') then
        f_out <= a + b;
    end if;
end process;
end;

```

## **Effect of Using syn\_no\_compile\_point**

This attribute can be used when the Auto Compile Point option is set:

```
option set automatic_compile_point 1
```

The Automatic Compile Point (ACP) flow is applied globally and creates compile points automatically for large modules of a design. If you do not want this to occur for individual modules in the design, set the `syn_no_compile_point` attribute to 1. This turns off the effects of automatically creating a compile point for the specified modules, which prevents extensive optimizations within the design units.

The effects of ACP synthesis for the Verilog/VHDL code segments above can be seen in the view, where a module displayed in green (for example, `v:add`) is a compile point. A module displayed in yellow (for example, `v:mult`) is not considered a compile point and is specified with `syn_no_compile_point=1`.

## syn\_noprune

### Directive

Prevents optimizations for instances and black-box modules (including technology-specific primitives) with unused output ports.

### Description

Use this directive to prevent the removal of instances, black-box modules, and technology-specific primitives with unused output ports during optimization.

By default, synthesis removes any module that does not drive logic as part of the synthesis optimization process. If you want to keep such an instance in the design, use the `syn_noprune` directive on the instance or module, along with `syn_hier` set to `hard`.

The `syn_noprune` directive can prevent a hierarchy from being dissolved or flattened. To ensure that a design with multiple hierarchies is preserved, apply this directive on the leaf hierarchy, which is the lower-most hierarchical level. This is especially important when hierarchies cannot be accessed or edited.

For further information about this and other directives used for preserving logic, see [Comparison of syn\\_keep, syn\\_preserve, and syn\\_noprune, on page 669](#).

### syn\_noprune Syntax

| CDC File | <code>define_directive {object} {syn_noprune} {0 1}</code>                                                          | CDC Example      |
|----------|---------------------------------------------------------------------------------------------------------------------|------------------|
| Verilog  | <code>object /* synthesis syn_noprune = 1 */;</code>                                                                | Verilog Examples |
| VHDL     | <code>attribute syn_noprune : boolean;</code><br><code>attribute syn_noprune of object : objectType is true;</code> | VHDL Examples    |

### CDC Example

```
define_directive {v:bb} {syn_noprune} (1)
```

## Verilog Examples

This section contains Verilog code snippets and examples.

### Verilog Example 1: Module Declaration

The `syn_noprune` attribute can be applied in two places: on the module declaration or in the top-level instantiation. The most common place to use `syn_noprune` is in the declaration of the module. By placing it here, all instances of the module are protected.

```
//Top module
module top (input int a, b, output int c);
    assign c=b;
    sub i1 (a);
endmodule

//Intermediate sub level which does not specify syn_noprune
module sub (input int a);
    leaf i2 (a,);
endmodule

//Leaf level with syn_noprune directive
module leaf (input int a, output int b)
    /* synthesis syn_noprune=1 */;
    assign b = a;
endmodule
```

The results for this example are shown in [Effect of Using `syn\_noprune`: Example 1, on page 718](#).

### Black Box Declaration

Here is a snippet showing `syn_noprune` used on black-box instances. If your design uses multiple instances with a single module declaration, the `synthesis` comment must be placed before the semicolon (`;`) following the port list for each of the instances.

```
my_design my_design1(out,in,clk_in) /* synthesis syn_noprune=1 */;
my_design my_design2(out,in,clk_in) /* synthesis syn_noprune=1 */;
```

In this example, only instance `my_design2` is removed if the output port is not mapped.

## Verilog Example 2: Hierarchical Design

In this example, `syn_noprune` is applied on the leaf-level module `sub1`. Although `syn_noprune` has not been applied to the intermediate level hierarchy, the directive is specified on an instance of module `sub1` that includes `inst1`, `inst2`, and `inst3`. The software propagates this directive upwards in the hierarchy chain. See [Effect of Using `syn\_noprune`: Example 2, on page 718](#).

```
//Leaf level module
module sub1 (data, rst, dout);
parameter width = 1;
input [width :0] data;
input rst;
output [width : 0] dout;
assign dout = rst?1'b0:data;
endmodule

//Intermediate Top level with 3 instances of sub1
module top (data1,data2,data3, rst, dout1);
parameter width1 = 2;
parameter width2 = 3;
parameter width3 = 4;
input [width1 :0] data1;
input [width2 :0] data2;
input [width3 :0] data3;
input rst;
output [width1 : 0] dout1;
sub1 #(width1) inst1 (data1,rst,dout1);
sub1 #(width2) inst2 (data2,rst,) /* synthesis syn_noprune=1 */;
sub1 #(width3) inst3 (data3,rst,);
endmodule

//Top level
module top1 (data1,data2,data3, rst, dout1);
parameter width1 = 2;
parameter width2 = 3;
parameter width3 = 4;
input [width1 :0] data1;
input [width2 :0] data2;
input [width3 :0] data3;
input rst;
output [width1 : 0] dout1;
top #(width1, width2, width3) top (data1, data2, data3,
rst, dout1);
endmodule
```

## VHDL Examples

This section contains VHDL code snippets and examples.

### Architecture Declaration

The `syn_noprune` directive is normally associated with the names of architectures. Once it is associated, any component instantiation of the architecture (design unit) is protected from being deleted.

```
library ieee;
architecture mydesign of rtl is

attribute syn_noprune : boolean;
attribute syn_noprune of mydesign : architecture is true;

-- Other code
```

### VHDL Example 1: Component Instance Declaration

The `syn_noprune` directive works the same on component instances as with a component declaration.

```
library ieee;
use ieee.std_logic_1164.all;
entity sub is
port (a, b, c, d : in std_logic;
      x,y : out std_logic);
end sub;

architecture behave of sub is
attribute syn_hier : string;
attribute syn_hier of behave : architecture is "hard";
begin
  x <= a and b;
  y <= c and d;
end behave;

--Top level
library ieee;
use ieee.std_logic_1164.all;
entity top is
port (a1, b1 : in std_logic;
      c1,d1,clk : in std_logic;
      y1 :out std_logic);
end;
```

```

architecture behave of top is
component sub
port (a, b, c, d : in std_logic;
x,y : out std_logic);
end component;

signal x2,y2,x3,y3 : std_logic;
attribute syn_noprune : boolean;
attribute syn_noprune of u1 : label is true;

begin
u1: sub port map(a1, b1, c1, d1, x2, y2);
--Instance with syn_noprune directive
u2: sub port map(a1, b1, c1, d1, x3, y3);

process begin
wait until (clk = '1') and clk'event;
y1 <= a1;
end process;

end;

```

The results for this example are shown in [Effect of Using syn\\_noprune: Example 3, on page 720](#)

#### VHDL Example 4: Black Box

```

--Top level
library ieee;
use ieee.std_logic_1164.all;

entity top is
port (a1, b1 : in std_logic;
      c1,d1,clk : in std_logic;
      y1 :out std_logic );
end;

architecture behave of top is
component sub
port (a, b, c, d : in std_logic;
      x,y : out std_logic );
end component;

attribute syn_noprune : boolean;
attribute syn_noprune of sub : component is true;

signal x2,y2,x3,y3 : std_logic;
begin

```

```

u1: sub port map(a1, b1, c1, d1, x2, y2);
u2: sub port map(a1, b1, c1, d1, x3, y3);
    process begin
        wait until (clk = '1') and clk'event;
        y1 <= a1;
    end process;
end;

```

The results for this example are shown in [Effect of Using syn\\_noprune in a Mixed Language Design , on page 722](#). **Mixed Language Example**

The `syn_noprune` directive can be specified on a module or architecture in a mixed Verilog and VHDL design. In the following example, the `syn_noprune` directive is specified on module `sub` in the top-level Verilog file.

```

module top (input a1,b1,c1,d1,
            input a2,b2,c2,d2,
            output x,y
        );

    sub inst1 (a1,b1,c1,d1,,)/*synthesis syn_noprune=1*/;
    sub inst2 (a2,b2,c2,d2,x,y);

endmodule

```

The architecture `sub` is defined in the following VHDL library file.

```

library ieee;
use ieee.std_logic_1164.all;

entity sub is
    port (a, b, c, d : in std_logic;
          x,y : out std_logic );
end sub;

architecture behave of sub is
attribute syn_hier : string;
attribute syn_hier of behave : architecture is "hard";
begin
    x <= a and b;
    y <= c and d;
end behave;

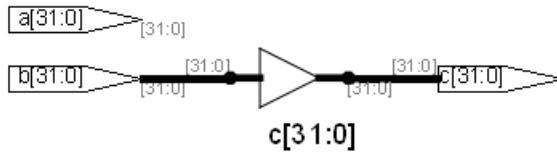
```

The results for this example are shown in [Effect of Using syn\\_noprune in a Mixed Language Design , on page 722](#).

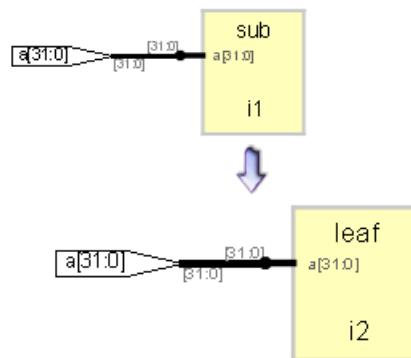
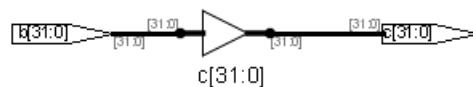
## Effect of Using syn\_noprune: Example 1

The following view shows that the design hierarchy is preserved when the syn\_noprune directive is applied on the module leaf. Otherwise, the design hierarchies are dissolved.

Without syn\_noprune

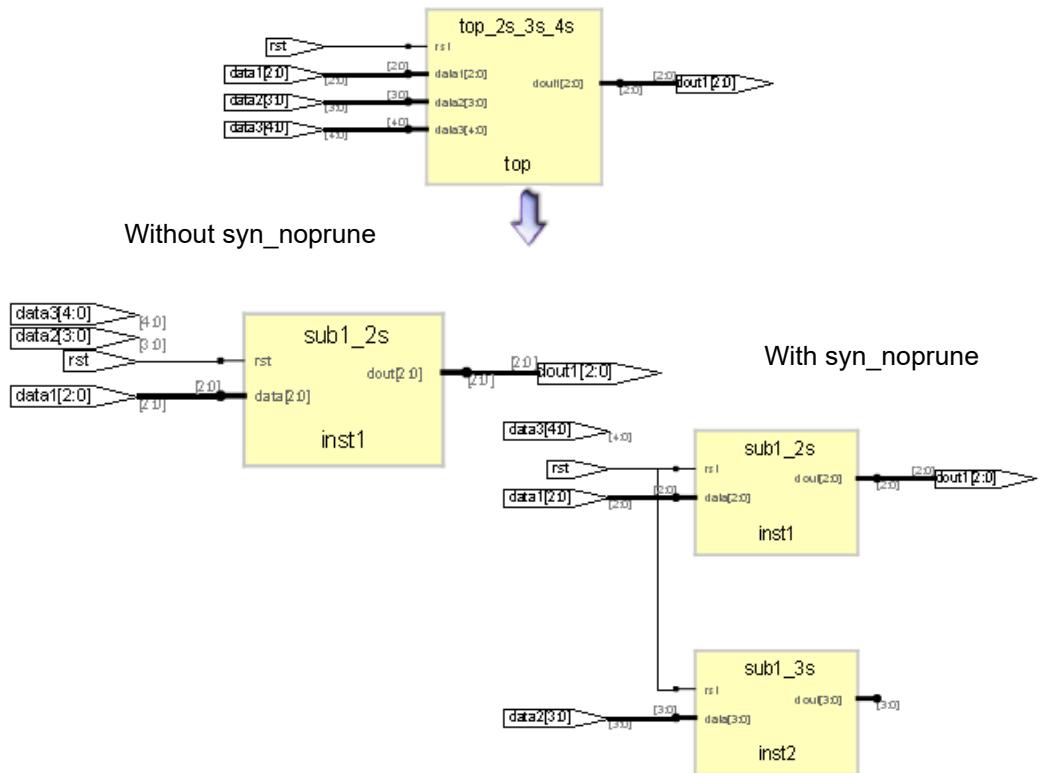


With syn\_noprune



## Effect of Using syn\_noprune: Example 2

In this example, the software preserves the lower-most leaf hierarchy inst2 and the hierarchy above it. When syn\_noprune is not applied, inst2 is not preserved.



In this example, the software propagates the `syn_noprune` directive downwards in the hierarchy chain.

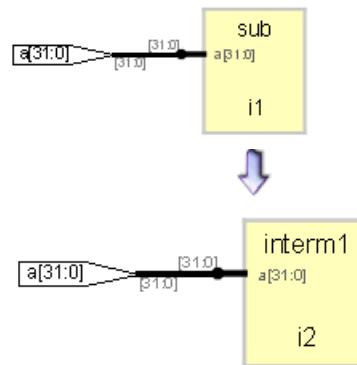
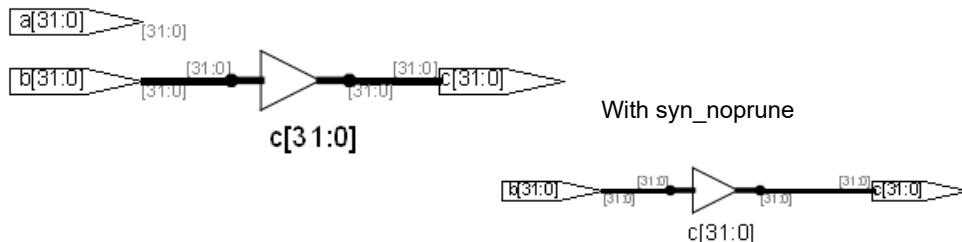
```
//Top module
module top (input int a, b, output int c);
assign c=b;
sub i1 (a);
endmodule

//Hier1
module sub (input int a);
interm1 i2 (a);
endmodule
```

```
//Hier2
module interm1 (input int a) /* synthesis syn_noprune = 1*/;
  interm2 i3 (a);
endmodule

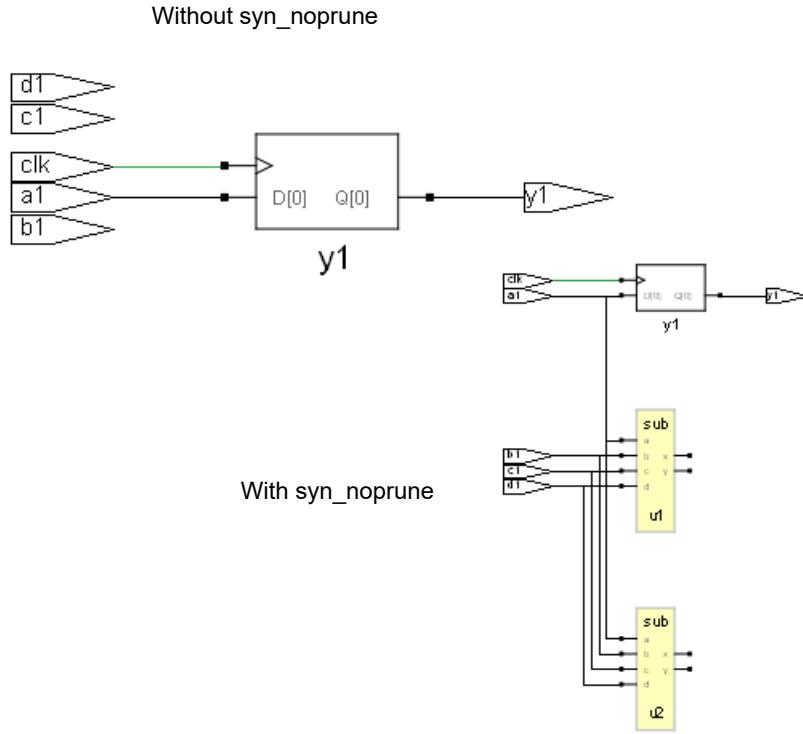
//Hier3
module interm2 (input int a);
  leaf i4 (a);
endmodule
```

Without syn\_noprune



### Effect of Using syn\_noprune: Example 3

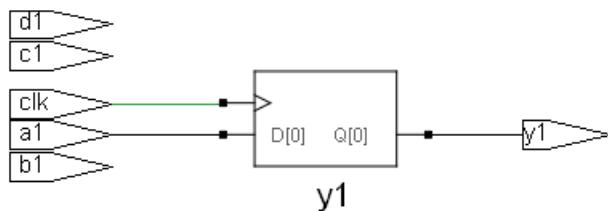
The following views show that the design hierarchy is preserved when the `syn_noprune` directive is applied for the component `sub`.



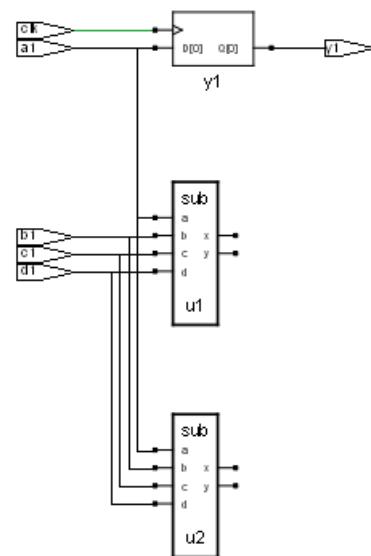
### Effect of Using syn\_noprune: Example 4

The following views show that the instance and black box module are not optimized away when syn\_noprune is applied.

Without syn\_noprune

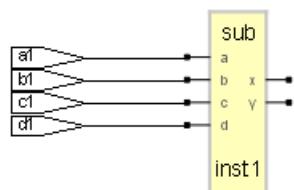
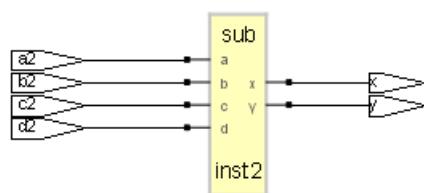


With syn\_noprune



## Effect of Using syn\_noprune in a Mixed Language Design

The following view shows that the design hierarchy is preserved when the syn\_noprune directive is applied on sub.



## **syn\_packer\_effort\_level**

### *Attribute*

Determines utilization by specifying a packing effort level for global placement on the top-level module or architecture.

### **Description**

Determines utilization by specifying a packing effort level for global placement. You can only specify this attribute in the constraint file.

The higher the value, the more densely the components are packed. If you have a design with components occupying most of the chip, you can set a higher value so that the tool packs the components and frees up area. Conversely, if you have a congested design, set a lower value to spread out the design and reduce congestion.

### **Constraint File Syntax**

```
define_global_attribute syn_packer_effort_level {value}
```

In the above syntax, *value* is an integer ranging from 0 to 9. The higher the value, the greater the packing effort, and the more densely the components are packed. Set a higher value to reduce area, and a lower value to reduce congestion. Typical values for medium to high-utilization designs are 2 or 3. If you do not specify the attribute, the tool automatically picks an effort level based on the design.

For example:

```
define_global_attribute syn_packer_effort_level 5
```

## syn\_pad\_type

### *Attribute*

Specifies an I/O buffer standard for a port.

### Description

Specifies an I/O buffer standard. Refer to the vendor documentation for a list of I/O buffer standards. Use this attribute instead of the I/O buffer standard attribute `xc_padtype`.

Some I/O standards for more recent devices require additional information to be specified with the `define_io_standard` constraint in the FDC file. See [define\\_io\\_standard, on page 300](#) in the *Command Reference* for details.

### syn\_pad\_type Syntax

|          |                                                                                                              |                            |
|----------|--------------------------------------------------------------------------------------------------------------|----------------------------|
| FDC File | <code>define_io_standard -default_portType {portName} -delay_type portType syn_pad_type {io_standard}</code> | Constraints Editor Example |
| Verilog  | <code>object /* synthesis syn_pad_type = io_standard */</code>                                               | Verilog Example            |
| VHDL     | <code>attribute syn_pad_type of object : objectType is io_standard;</code>                                   | VHDL Example               |

- *portType* can be input, output, or bidir. Setting `default_input`, `default_output`, or `default_bidir` causes all ports of that type to have the same I/O standard applied.
- *io\_standard* specifies I/O standard (see examples in the following table).

## Constraint File Examples

| To set ...                                            | Use this syntax ...                                                           |
|-------------------------------------------------------|-------------------------------------------------------------------------------|
| The default for all input ports to the AGP1X pad type | define_io_standard -default_input -delay_type input syn_pad_type {AGP1X}      |
| All output ports to the GTL pad type                  | define_io_standard -default_output -delay_type output syn_pad_type {GTL}      |
| All bidirectional ports to the LVC<MOS_18 pad type    | define_io_standard -default_bidir -delay_type bidir syn_pad_type {LVC MOS_18} |

The following are examples of pad types set on individual ports. You cannot assign pad types to bit slices.

```
define_io_standard {in1} -delay_type input
    syn_pad_type {LVCMOS_15}

define_io_standard {out21} -delay_type output
    syn_pad_type {LVCMOS_33}

define_io_standard {bidirbit} -delay_type bidir
    syn_pad_type {LVTTL_33}
```

## Constraints Editor Example

|   | Enable                              | Object Type | Object   | Attribute    | Value     |
|---|-------------------------------------|-------------|----------|--------------|-----------|
| 1 | <input checked="" type="checkbox"/> | port        | p:output | syn_pad_type | LVCMOS_18 |
| 2 |                                     |             |          |              |           |

## Verilog Example

```
module top (clk,A,B,PC,P);
  input clk;
  input A ;
  input B,PC;
  output reg P/* synthesis syn_pad_type = "OBUF_LVCMOS_18" */;
  reg a_d,b_d;
  reg m;
```

```
always @ (posedge clk)
begin
    a_d <= A;
    b_d <= B;
    m   <= a_d + b_d;
    P   <= m + PC;
end
endmodule
```

## VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library synplify;
use synplify.attributes.all;

entity top is
    port (clk : in std_logic ;
          A : in std_logic_vector(1 downto 0);
          B : in std_logic_vector(1 downto 0);
          PC : in std_logic_vector(1 downto 0);
          P : out std_logic_vector(1 downto 0) );
attribute syn_pad_type : string;
attribute syn_pad_type of P : signal is "OBUF_LVCMOS_18";
end top ;

architecture rtl of top is
signal m : std_logic_vector(1 downto 0);
begin
    process(clk)
    begin
        if (clk'event and clk = '1') then
            m <= A + B;
            P <= m + PC;
        end if;
    end process;
end rtl;
```

## Effect of Using syn\_pad\_type

The following figure shows the netlist output after the attribute is applied:

Verilog `output reg P /*synthesis syn_pad_type = "OBUF_LVCMOS_18"*/;`

VHDL `attribute syn_pad_type of P : signal is "OBUF_LVCMOS_18";`

---

### Net list

```
95
96     )
97     (instance m_2_4 (viewRef PRIM (cellRef LUT2_L (libraryRef VIRTEX)))
98         (property INIT (string "4'h6"))
99     )
100    (instance P_2_2 (viewRef PRIM (cellRef LUT2_L (libraryRef VIRTEX)))
101        (property INIT (string "4'h6"))
102    )
103    (instance P_obuf (viewRef PRIM (cellRef OBUF (libraryRef VIRTEX)))
104        (property IOSTANDARD (string "LVCMOS18"))
105    )
106    (instance PC_ibuf (viewRef PRIM (cellRef IBUF (libraryRef VIRTEX)))
107  )
```

### P&R Files

We can see the effect of `syn_pad_type` in the following P&R files

```
<projectdirectory>\rev_1\pr_1\top.pad(412):
T17|P|IOB|IO_L1P_GC_24|OUTPUT|LVCMOS18|24|12|SLOW|||UNLOCATED|NO|NONE|
```

```
<projectdirectory>\rev_1\pr_1\top_pad.txt(413
|T17|P|IOB|IO_L1P_GC_24|OUTPUT|LVCMOS18|24|12|SLOW|||UNLOCATED
```

## **syn\_partition\_keep**

### *Attribute*

Preserves the specified net and keeps it intact during optimization and synthesis during the system-level compile stage.

### **Description**

With this attribute, the tool preserves the net without optimizing it away by placing a temporary keep buffer primitive on the net as a placeholder, by a constant 1 or 0. The default is 0. You can view this buffer in the schematic views. The buffer is not part of the final netlist, so no extra logic is generated. The effects of this attribute is not be seen in the individual FPGA mapping stage. This attribute cannot be applied on some design objects (such as nets and module instantiations) in the system-level compilation stage.

There are various situations where this attribute is useful:

- To preserve a net that would otherwise be removed as a result of optimization. You might want to preserve the net for simulation results or to obtain a different synthesis implementation.
- To prevent duplicate cells from being merged during optimization. You apply the attribute to the nets connected to the input of the cells you want to preserve.
- As a placeholder to apply the -through option of the `set_multicycle_path` or `set_false_path` timing constraint. This allows you to specify a unique path as a multiple-cycle or false path. Apply the constraint to the keep buffer.
- To prevent the absorption of a register into a macro. If you apply `syn_partition_keep` to a reg or signal that will become a sequential object, the tool keeps the register and does not absorb it into a macro.

### **syn\_partition\_keep Syntax**

CDC File    `define_attribute {n:libName.moduleName | netName} {syn_partition_keep} {0|1}`

## **syn\_partition\_preserve**

### *Attribute*

Prevents sequential optimizations such as constant propagation, inverter push-through, and FSM extraction at system-level partition compile stage. These optimizations are allowed during the synthesis stage.

### **Description**

The `syn_partition_preserve` attribute determines when objects are optimized away. Use `syn_partition_preserve` to retain registers for simulation, or to preserve the logic of registers driven by a constant 1 or 0. The default is 0. The effects of this attribute is not be seen in the individual FPGA mapping stage. This attribute cannot be applied on some design objects (such as nets and module instantiations) in the system-level compilation stage.

You can set `syn_partition_preserve` on individual registers or on the module/architecture so that the directive is applied to all registers in the module.

For example, assume that the input of a flip-flop is always driven to the same value, such as logic 1. By default, synthesis ties that signal to VCC and removes the flip-flop. Using `syn_partition_preserve` on the registered signal prevents the removal of the flip-flop. This is useful when you are not finished with the design, but want to do a preliminary run to determine area utilization.

Another use for this attribute is to preserve a particular state machine. When the FSM compiler is enabled, it performs various state-machine optimizations. Use `syn_partition_preserve` to retain a particular state machine and prevent it from being optimized away.

When registers are removed during synthesis, the tool issues a warning message in the log file. For example:

```
@W:...Register bit out2 is always 0, optimizing ...
```

The `syn_partition_preserve` attribute is similar to `syn_partition_keep` and `syn_no-prune` in that it preserves logic. For more information, see [\*syn\\_partition\\_keep\*, on page 729](#).

## **syn\_partition\_preserve Syntax**

CDC  
File      **define\_attribute {n:*libName.moduleName* | netName} {syn\_partition\_preserve}**  
          {0|1}

---

## syn\_pipeline

### Attribute

Permits registers to be moved to improve timing.

### Description

Specifies that registers that are outputs of multipliers can be moved to improve timing. Depending on the criticality of the path, the tool moves the output register either into the multiplier or back to the input side.

Do not use the syn\_pipeline attribute with the Synthesis Strategy fast option.

### syn\_pipeline Syntax Specification

|         |                                                                              |                                            |
|---------|------------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute {register} syn_pipeline {0 1}</code>                  | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_pipeline = 1 0 */;</code>                      | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_pipeline of object : objectType is "true false" ;</code> | <a href="#">VHDL Example</a>               |

### Constraints Editor Example

|   | Enabled                             | Object Type | Object       | Attribute    | Value | Val Type | Description                      |
|---|-------------------------------------|-------------|--------------|--------------|-------|----------|----------------------------------|
| 1 | <input checked="" type="checkbox"/> | register    | i:temp2[7:0] | syn_pipeline | 1     | boolean  | Controls pipelining of registers |

### Verilog Example

```
module pipeline (a, b, clk,r);
  input [3:0] a,b;
  input clk;
  output [7:0] r;
  reg [3:0] a_reg,b_reg;
  reg [7:0] temp2/* synthesis syn_pipeline = 1 */;
  reg [7:0] temp3;
  wire [7:0] temp1;
  assign temp1 = a_reg * b_reg;
```

```
always @ (posedge clk)
begin
    a_reg <= a;
    b_reg <= b;
    temp2 <= temp1;
    temp3 <= temp2;
end
assign r = temp3;
endmodule
```

## VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity pipeline is
    port (clk : in std_logic;
          a : in std_logic_vector(3 downto 0);
          b : in std_logic_vector(3 downto 0);
          r : out std_logic_vector(7 downto 0) );
end pipeline;

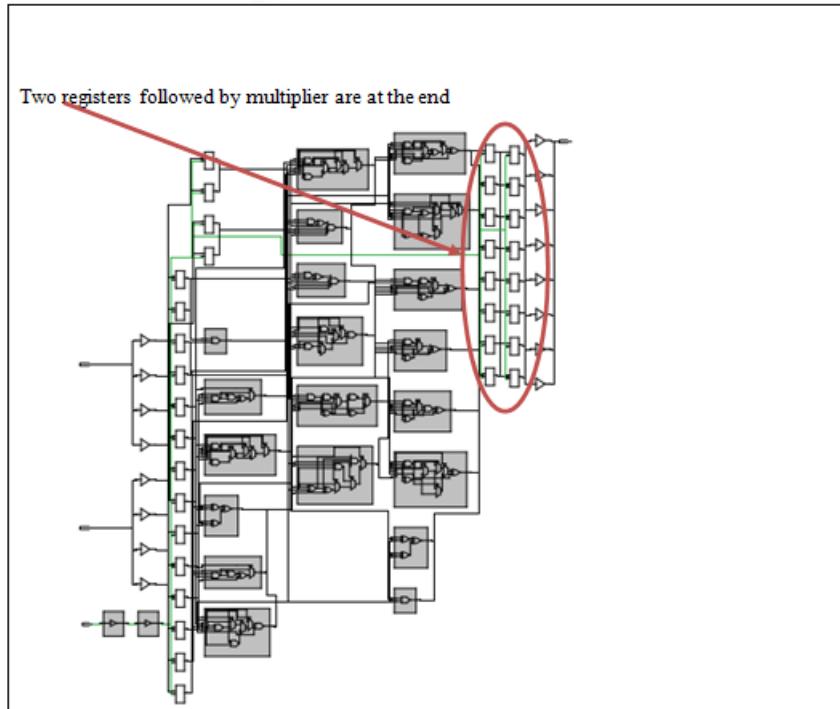
architecture rtl of pipeline is
signal a_reg : std_logic_vector(3 downto 0);
signal b_reg : std_logic_vector(3 downto 0);
signal temp1 : std_logic_vector(7 downto 0);
signal temp2 : std_logic_vector(7 downto 0);
signal temp3 : std_logic_vector(7 downto 0);
attribute syn_pipeline : string;
attribute syn_pipeline of temp2 : signal is "true";
begin
    process(clk)
    begin
        if (clk'event and clk = '1') then
            temp1 <= a_reg * b_reg;
            a_reg <= a;
            b_reg <= b;
            temp2 <= temp1;
            temp3 <= temp2;
            r <= temp3;
        end if;
    end process;
end rtl;
```

## Effect of Using syn\_pipeline

The following example shows a design where syn\_pipeline is set to 0:

Verilog      `reg [7:0] temp2/* synthesis syn_pipeline = 0 */;`

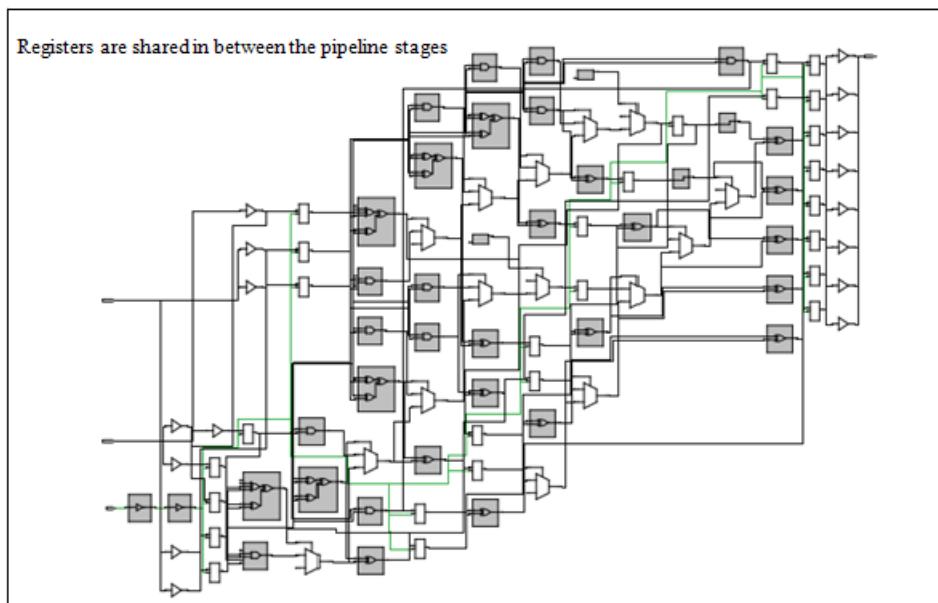
VHDL      `attribute syn_dspstyle of temp2 : signal is "false" ;`



The next example shows the results when syn\_pipeline is set to 1:

Verilog      `reg [7:0] temp2/* synthesis syn_pipeline = 1 */;`

VHDL      `attribute syn_dspstyle of temp2 : signal is "true" ;`



## **syn\_pnr\_preserve\_regs**

*Attribute/Directive*

Preserves registers in Vivado.

### **Description**

Preserves registers in Vivado. This attribute/directive can be applied only on modules and views.

Boolean values 1 and 0 are not supported. Use only on and off.

### **syn\_pnr\_preserve\_regs Syntax**

---

|          |                                                                          |                             |
|----------|--------------------------------------------------------------------------|-----------------------------|
| CDC File | <code>define_directive {v:my_module} {syn_pnr_preserve_regs} {on}</code> | <a href="#">CDC Example</a> |
|----------|--------------------------------------------------------------------------|-----------------------------|

---

|          |                                                                          |                             |
|----------|--------------------------------------------------------------------------|-----------------------------|
| FDC File | <code>define_attribute {v:my_module} {syn_pnr_preserve_regs} {on}</code> | <a href="#">FDC Example</a> |
|----------|--------------------------------------------------------------------------|-----------------------------|

---

|         |                                                       |                                 |
|---------|-------------------------------------------------------|---------------------------------|
| Verilog | <code>/*synthesis syn_pnr_preserve_regs="on"*/</code> | <a href="#">Verilog Example</a> |
|---------|-------------------------------------------------------|---------------------------------|

---

### **CDC Example**

Preserve registers in Vivado by using `syn_pnr_preserve_regs` on submodule by applying directive on view:

```
#top module off
define_directive {v:work.test} {syn_pnr_preserve_regs} {off}

#sub module on
define_directive {v:work.flop} {syn_pnr_preserve_regs} {on}
```

### **FDC Example**

Preserve registers in Vivado by using `syn_pnr_preserve_regs` on submodule by applying attribute on view:

```
#top module off
define_attribute {v:work.test} {syn_pnr_preserve_regs} {off}
```

```
#sub module on
define_attribute {v:work.flop} {syn_pnr_preserve_regs} {on}
```

## Verilog Example

Preserve registers in Vivado by enabling `syn_pnr_preserve_regs` on submodule in the RTL and by enabling attribute on submodule in FDC at pre-map:

```
module test(clk,rst,in,out);
  input clk,rst,in;
  output [15:0] out;
  module test_rtl(in1, in2, in3, in4, out1);

    wire [6:0] a,b;
    assign b = a ^ (~a);
    flop d_inst (clk,rst,{8'h55,b,in},out);
  endmodule

  module flop (clk,rst,din,dout )/*synthesis syn_pnr_preserve_regs =
  "on"*/;
    input clk,rst;
    input [15:0] din;
    output reg [15:0] dout;

    always @ (posedge clk or negedge rst)
    begin
      if (!rst)
        dout <= 0;
      else
        dout <= din;
    end
  endmodule
endmodule
```

## syn\_preserve

### Directive

Prevents sequential optimizations such as constant propagation, inverter push-through, and FSM extraction.

### Description

The `syn_preserve` directive determines when objects are optimized away. Use `syn_preserve` to retain registers for simulation, or to preserve the logic of registers driven by a constant 1 or 0. You can set `syn_preserve` on individual registers or on the module/architecture so that the directive is applied to all registers in the module.

For example, assume that the input of a flip-flop is always driven to the same value, such as logic 1. By default, synthesis ties that signal to VCC and removes the flip-flop. Using `syn_preserve` on the registered signal prevents the removal of the flip-flop. This is useful when you are not finished with the design, but want to do a preliminary run to determine area utilization.

Another use for this attribute is to preserve a particular state machine. When the FSM compiler is enabled, it performs various state-machine optimizations. Use `syn_preserve` to retain a particular state machine and prevent it from being optimized away.

When registers are removed during synthesis, the tool issues a warning message in the log file. For example:

```
@W:...Register bit out2 is always 0, optimizing ...
```

The `syn_preserve` directive is similar to `syn_keep` and `syn_noprune` in that it preserves logic. For more information, see [Comparison of syn\\_keep, syn\\_preserve, and syn\\_noprune, on page 669](#).

### syn\_preserve Syntax

|          |                                                                             |                                 |
|----------|-----------------------------------------------------------------------------|---------------------------------|
| CDC File | <code>define_directive {object} {syn_preserve} {0 1}</code>                 | <a href="#">CDC Example</a>     |
| Verilog  | <code>object /* synthesis syn_preserve = 0  1 */</code>                     | <a href="#">Verilog Example</a> |
| VHDL     | <code>attribute syn_preserve of object : objectType is true   false;</code> | <a href="#">VHDL Examples</a>   |

## CDC Example

```
define_directive {v:mod_preserve}{syn_preserve}{1}
```

## Verilog Example

In the following example, `syn_preserve` is applied to all registers in the module to prevent them from being optimized away. For the results, see [Effect of using `syn\_preserve`, on page 741](#).

```
module mod_preserve (out1,out2,clk,in1,in2)
    /* synthesis syn_preserve=1 */;
    output out1, out2;
    input clk;
    input in1, in2;
    reg out1;
    reg out2;
    reg reg1;
    reg reg2;

    always@ (posedge clk)begin
        reg1 <= in1 &in2;
        reg2 <= in1&in2;
        out1 <= !reg1;
        out2 <= !reg1 & reg2;
    end
endmodule
```

This is an example of setting `syn_preserve` on a state register:

```
reg [3:0] curstate /* synthesis syn_preserve = 1 */ ;
```

## VHDL Examples

This section contains VHDL code examples:

### Example 1

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

```

entity simpledff is
    port (q : out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic );

-- Turn on flip-flop preservation for the q output
attribute syn_preserve : boolean;
attribute syn_preserve of q : signal is true;
end simpledff;

architecture behavior of simpledff is
begin
    process(clk)
    begin
        if rising_edge(clk) then
-- Notice the continual assignment of "11111111" to q.
            q <= (others => '1');
        end if;
    end process;
end behavior;

```

## Example 2

In this example, `syn_preserve` is used on the signal `curstate` that is later used in a state machine to hold the value of the state register.

```

architecture behavior of mux is
begin
    signal curstate : state_type;
    attribute syn_preserve of curstate : signal is true;

-- Other code

```

## Example 3

The results for the following example are shown in [Effect of using `syn\_preserve`, on page 741](#).

```

library ieee;
use ieee.std_logic_1164.all;

entity mod_preserve is
    port (out1 : out std_logic;
          out2 : out std_logic;
          in1,in2,clk : in std_logic );
end mod_preserve;

```

```
architecture behave of mod_preserve is
attribute syn_preserve : boolean;
attribute syn_preserve of behave: architecture is true;
signal reg1 : std_logic;
signal reg2 : std_logic;
begin
  process
  begin
    wait until clk'event and clk = '1';
    reg1 <= in1 and in2;
    reg2 <= in1 and in2;
    out1 <= not (reg1);
    out2 <= (not (reg1) and reg2);
  end process;
end behave;
```

## Effect of using syn\_preserve

The following figure shows reg1 and out2 are preserved during optimization with syn\_preserve.

When syn\_preserve is not set, reg1 and reg2 are shared because they are driven by the same source. out2 gets the result of the AND of reg2 and NOT reg1. This is equivalent to the AND of reg1 and NOT reg1, which is a 0. As this is a constant, the tool removes out2 and the output out2 is always 0.

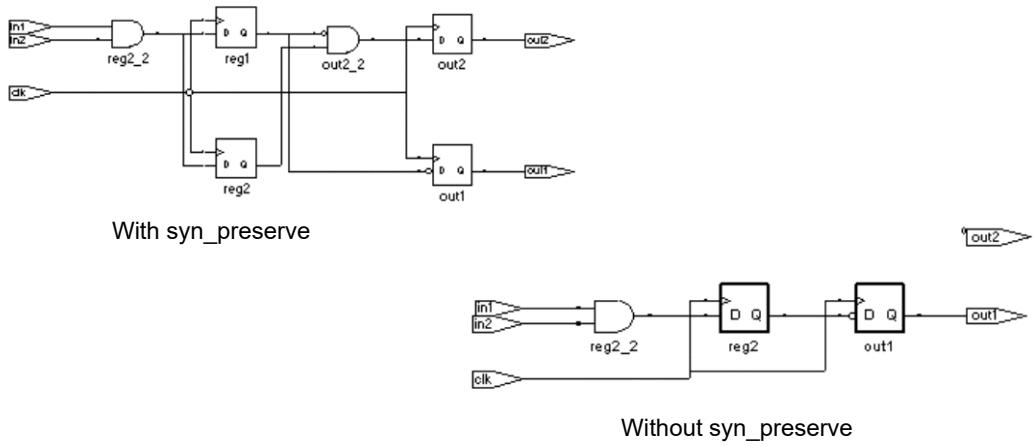
---

Verilog      mod\_preserve /\* synthesis syn\_preserve = 1 \*/

---

VHDL      attribute syn\_preserve of behave : architecture is true;

---



## **syn\_probe**

### *Attribute*

Inserts probe points for testing and debugging the internal signals of a design.

### **syn\_probe Values**

| <b>Value</b>    | <b>Description</b>                                                                                                                                                  |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1/true          | Inserts a probe, and automatically derives a name for the probe port from the net name.                                                                             |
| 0/false         | Disables probe generation.                                                                                                                                          |
| <i>portName</i> | Inserts a probe and generates a port with the specified name. If you include empty square brackets, [ ], the probe names are automatically indexed to the net name. |

### **Description**

`syn_probe` works as a debugging aid, inserting probe points for testing and debugging the internal signals of a design. The probes appear as ports at the top level. When you use this attribute, the tool also applies `syn_keep` to the net.

You can specify values to name probe ports and assign pins to named ports. Pin-locking properties of probed nets will be transferred to the probe port and pad. If empty square brackets [] are used, probe names will be automatically indexed, according to the index of the bus being probed.

You can specify pin locations for the probe signals. Pin locations cannot be applied to bus slices.

## syn\_probe Syntax

|         |                                                                               |                 |
|---------|-------------------------------------------------------------------------------|-----------------|
| FDC     | <code>define_attribute {n:netName} syn_probe {probePort 1 0}</code>           | FDC Example     |
| Verilog | <code>object /* synthesis syn_probe = "probePort"  1 0 */;</code>             | Verilog Example |
| VHDL    | <code>attribute syn_probe of object : objectType is "probePort"  1 0 ;</code> | VHDL Example    |

The table below shows how to apply syn\_probe values to nets, buses, and bus slices. It indicates what port names appear at the top level. When the syn\_probe value is 0, probe generation is disabled; when syn\_probe is 1, the probe port name is derived from the net name.

| Net Name      | syn_probe Value | Probe Port                             | Comments                                                                                |
|---------------|-----------------|----------------------------------------|-----------------------------------------------------------------------------------------|
| n:ctrl        | 1               | ctrl_probe_1                           | Probe port name generated by the synthesis tool.                                        |
| n:ctr         | test_pt         | test_pt                                | For string values on a net, the port name is identical to the syn_probe value.          |
| n:aluout[2]   | test_pt         | test_pt                                | For string values on a bus slice, the port name is identical to the syn_probe value.    |
| n:aluout[2]   | test_pt[ ]      | test_pt[2]                             | The empty square brackets [ ] indicate that port names will be indexed to net names.    |
| n:aluout[2:0] | test_pt[ ]      | test_pt[2]<br>test_pt[1]<br>test_pt[0] | The empty square brackets [ ] indicate that port names will be indexed to net names.    |
| n:aluout[2:0] | test_pt         | test_pt,<br>test_pt_0,<br>test_pt_1    | If a syn_probe value without brackets is applied to a bus, the port names are adjusted. |

## FDC Example

The following examples insert a probe signal onto a net and assign pin locations to the ports.

```
define_attribute {n:inst2.DATA0_*[7]} syn_probe {test_pt[]}
define_attribute {n:inst2.DATA0_*[7]} syn_loc
{14,12,11,5,21,18,16,15}
```

## Constraints Editor Example

| Enable                              | Object Type | Object   | Attribute | Value | Value Type | Description             |
|-------------------------------------|-------------|----------|-----------|-------|------------|-------------------------|
| <input checked="" type="checkbox"/> | <any>       | <Global> | syn_probe | 1     | string     | Send a signal to out... |

## Verilog Example

The following example inserts probes on bus alu\_tmp [7:0] and assigns pin locations to each of the ports inserted for the probes.

```
module alu(out1, opcode, clk, a, b, sel);
output [7:0] out1;
input [2:0] opcode;
input [7:0] a, b;
input clk , sel;
reg [7:0] alu_tmp /* synthesis syn_probe="alu1_probe[]" */
    syn_loc="A5,A6,A7,A8,A10,A11,A13,A14" */;
reg [7:0] out1;
// Other code
always @(opcode or a or b or sel)
begin
    case (opcode)
        3'b000:alu_tmp <= a+b;
        3'b000:alu_tmp <= a-b;
        3'b000:alu_tmp <= a^b;
        3'b000:alu_tmp <= sel ? a:b;
        default:alu_tmp <= a|b;
    endcase
end
always @ (posedge clk)
out1 <= alu_tmp;
endmodule
```

## VHDL Example

The following example inserts probes on bus alu\_tmp(7 downto 0) and assigns pin locations to each of the ports inserted for the probes.

```

library ieee;
use ieee.std_logic_1164.all;

entity alu is
    port (a : in std_logic_vector(7 downto 0);
          b : in std_logic_vector(7 downto 0);
          opcode : in std_logic_vector(2 downto 0);
          clk : in std_logic;
          out1 : out std_logic_vector(7 downto 0));
end alu;

architecture rtl of alu is
signal alu_tmp : std_logic_vector (7 downto 0);
attribute syn_probe : string;
attribute syn_probe of alu_tmp : signal is "test_pt";
attribute syn_loc : string;
attribute syn_loc of alu_tmp : signal is
    "A5,A6,A7,A8,A10,A11,A13,A14";
begin
    process (clk)
    begin
        if (clk'event and clk = '1') then
            out1 <= alu_tmp;
        end if;
    end process;
    process (opcode,a,b)
    begin
        case opcode is
            when "000" => alu_tmp <= a and b;
            when "001" => alu_tmp <= a or b;
            when "010" => alu_tmp <= a xor b;
            when "011" => alu_tmp <= a nand b;
            when others => alu_tmp <= a nor b;
        end case;
    end process;
end rtl;

```

## Effect of Using syn\_probe

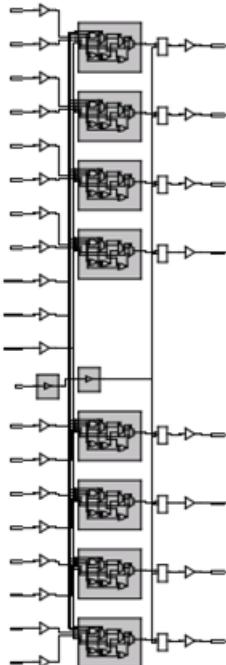
Before applying syn\_probe:

---

Verilog      `reg [7:0] alu_tmp /* synthesis syn_probe = "0" */`

VHDL      `attribute syn_probe of alu_tmp : signal is "0";`

---



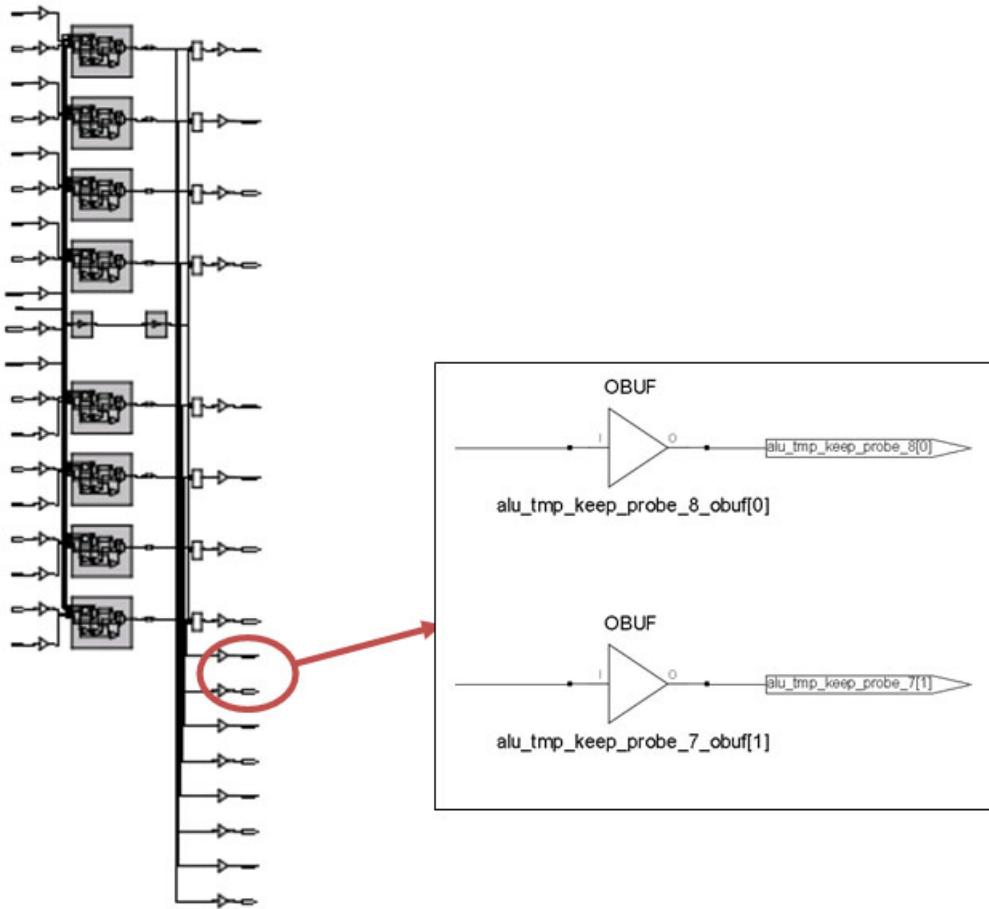
After applying `syn_probe` with 1:

---

Verilog      `reg [7:0] alu_tmp /* synthesis syn_probe = "1" */`

VHDL      `attribute syn_probe of alu_tmp : signal is "1";`

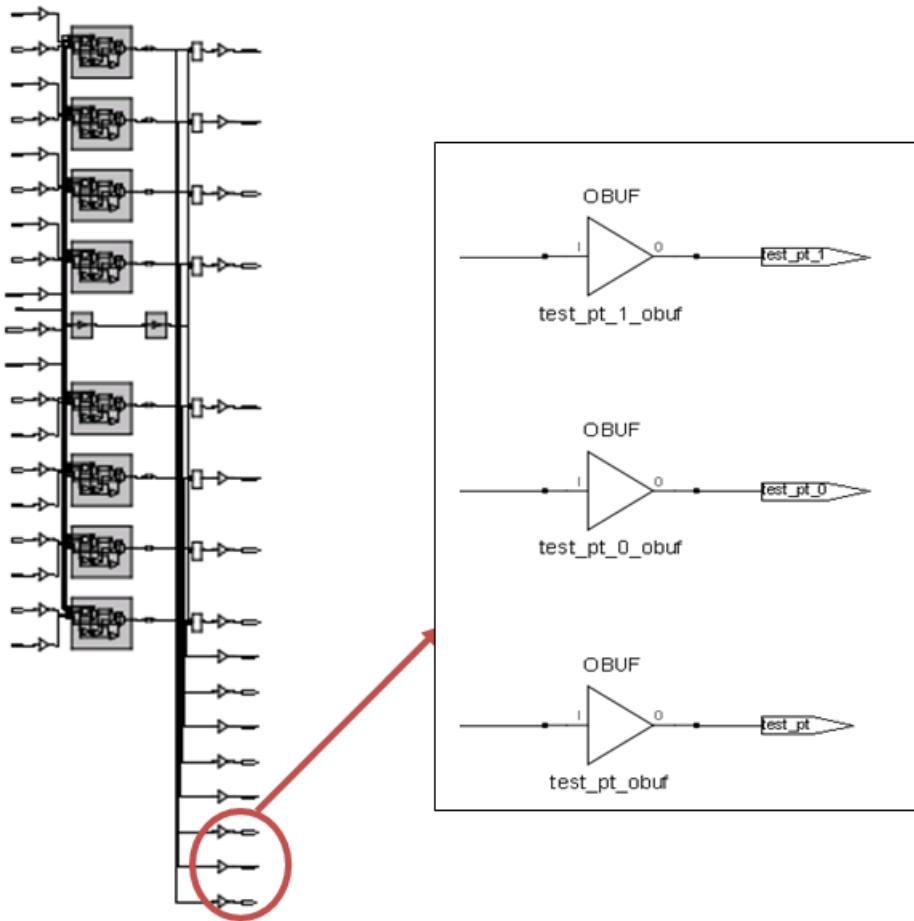
---



After applying `syn_probe` with `test_pt`:

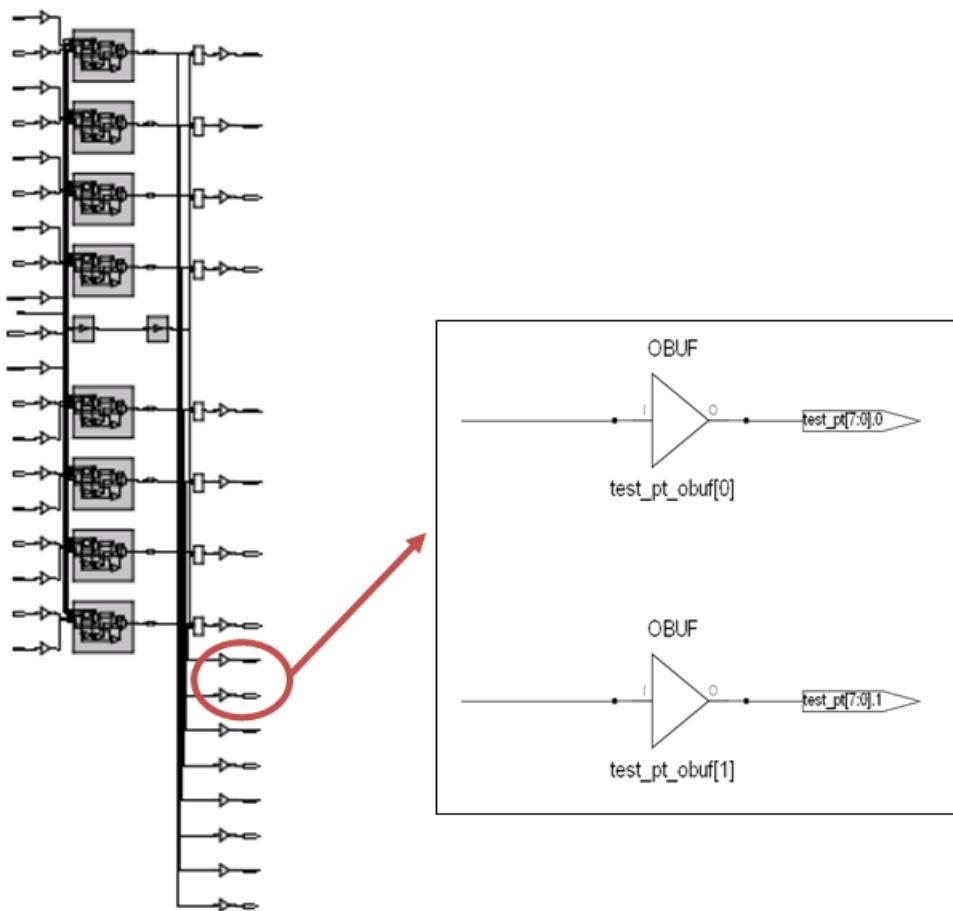
|         |                                                                   |
|---------|-------------------------------------------------------------------|
| Verilog | <pre>reg [7:0] alu_tmp /* synthesis syn_probe= "test_pt" */</pre> |
| VHDL    | <pre>attribute syn_probe of alu_tmp : signal is "test_pt";</pre>  |

---



After applying `syn_probe` with `test_pt[]`:

|         |                                                                       |
|---------|-----------------------------------------------------------------------|
| Verilog | <code>reg [7:0] alu_tmp /* synthesis syn_probe="test_pt[]" */;</code> |
| VHDL    | <code>attribute syn_probe of alu_tmp : signal is "test_pt[]" ;</code> |



## **syn\_ramstyle**

### *Attribute*

Specifies the implementation for an inferred RAM.

### **Description**

The `syn_ramstyle` attribute specifies the implementation to use for an inferred RAM. You can apply the attribute globally, to a module, or a RAM instance. You can also use `syn_ramstyle` to prevent the inference of a RAM, by setting it to registers. If your RAM resources are limited, you can map additional RAMs to registers instead of RAM resources using this setting.

### **Read-Write Address Checks**

When reads and writes are made to the same address, the output could be indeterminate, and this can cause simulation mismatches. By default, the synthesis tool inserts bypass logic around an inferred RAM to avoid these mismatches. The synthesis tool offers multiple ways to specify how to handle read-write address checking:

| <b>Read Write Control</b>           | <b>Use when ...</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>syn_ramstyle</code>           | The design does not read and write to the same address simultaneously and you want to specify the RAM implementation. The attribute has two mutually-exclusive read-write check options: <ul style="list-style-type: none"><li>• Use <code>no_rw_check</code> to eliminate bypass logic. If you enable global RAM inference with the Read Write Check on RAM option, you can use <code>no_rw_check</code> to selectively disable glue logic insertion for individual RAMs.</li><li>• Use <code>rw_check</code> to insert bypass logic. If you disable global RAM inference with the Read Write Check on RAM option, you can use <code>rw_check</code> to selectively enable glue logic insertion for individual RAMs.</li></ul> |
| <code>syn_rw_conflict_logic</code>  | The design does not read and write to the same address simultaneously and you want to let the tool select the RAM implementation. See <a href="#"><i>syn_rw_conflict_logic</i>, on page 792</a> for details.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <code>ReadWrite Check on RAM</code> | You want to globally enable or disable glue logic insertion for all the RAMs in the design.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

If there is a conflict, the software uses the following order of precedence:

- `syn_ramstyle` attribute settings
- `syn_rw_conflict_logic` attribute
- Read Write Check on RAM option on the Device panel of the Implementation Options dialog box.

## **syn\_ramstyle Syntax**

|         |                                                                                                                                         |                                           |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| FDC     | <code>define_attribute {signalname [bitRange]} syn_ramstyle {value}</code><br><code>define_global_attribute syn_ramstyle {value}</code> | <a href="#">Constraint Editor Example</a> |
| Verilog | <code>object /* synthesis syn_ramstyle = "value" */</code>                                                                              | <a href="#">Verilog Example</a>           |
| VHDL    | <code>attribute syn_ramstyle of object : objectType is "value" ;</code>                                                                 | <a href="#">VHDL Example</a>              |

The following table lists all the valid `syn_ramstyle` values.

|                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>block_ram</code><br>(default) | Specifies that the inferred RAM be mapped to the appropriate device-specific memory. It uses the dedicated memory resources in the FPGA.<br><br>By default, the software uses deep block RAM configurations instead of wide configurations to get better timing results. Using deeper RAMs reduces the output data delay timing by reducing the MUX logic at the output of the RAMs. By default the software does not use the parity bit for data with this option.<br><br>Alternatively, you can specify a <code>ramType</code> value. See <a href="#">RAM Type Values and Implementations</a> , on page 754 for details of how memory is implemented for different devices.<br><br>As the parity bit is not used for data by default, use <code>block_ram</code> with <code>power</code> for a more area-efficient RAM inferencing scheme.<br><br>This option implements block SelectRAM+ with additional glue logic to resolve read/write address conflicts. |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

|                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>no_rw_check</code> | <p>By default, the synthesis tool inserts bypass logic around the inferred RAM to avoid simulation mismatches caused by indeterminate output values when reads and writes are made to the same address. When this option is specified, the synthesis tool does not insert glue logic around the RAM.</p> <p>You can use this option on its own or in conjunction with a RAM type value such as <code>select ram</code>, or with the <code>power</code> value for supported technologies. You cannot use it with the <code>rw_check</code> option, as the two are mutually exclusive.</p> <p>There are other read-write check controls. See <a href="#">Read-Write Address Checks , on page 751</a> for details about the differences.</p> |
| <code>area</code>        | <p>When used with the <code>block_ram</code> or <code>no_rw_check</code> values, specifies that the inferred RAM be mapped to area-efficient Xilinx memory. The software uses wider block RAM configurations that are more area-efficient, instead of the default deeper RAM configurations. It also uses the parity bit for data. If you want the RAM implementations optimized for timing, use the <code>block_ram</code> or <code>no_rw_check</code> values.</p> <p>The synthesis tool tries to map memory to a Block SelectRAM, depending on the properties of the memory and the available resources. If this is not possible, the memory is mapped using <code>select_ram</code>.</p>                                               |
| <code>no_uram</code>     | <p><i>Xilinx</i></p> <p>When the <code>no_uram</code> option is specified, the tool prevents the inference of UltraRAM (URAM) block. When <code>no_uram</code> is used with <code>block_ram</code>, the memory is mapped to the block RAM instead of the URAM block.</p> <p>Usage: <b>/* synthesis syn_ramstyle = "block_ram,no_uram" */</b></p>                                                                                                                                                                                                                                                                                                                                                                                          |
| <code>power</code>       | <p>When used with the <code>block_ram</code> or <code>no_rw_check</code> values, specifies that the inferred RAM be mapped to power and area-efficient memory. The tool implements RAM using area-efficient wider block RAM configurations, instead of the default deeper RAM configurations.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>ramType</code>     | Specifies a RAM implementation ( <code>select_ram</code> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <code>registers</code>   | Specifies that an inferred RAM be mapped to registers (flip-flops and logic), not technology-specific RAM resources.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

---

---

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>rw_check</code>   | <p>When enabled, the synthesis tool inserts bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations.</p> <p>You can use this option on its own or in conjunction with a RAM type value such as <code>select_ram</code>, or with the <code>power</code> value for supported technologies. You cannot use it with the <code>no_rw_check</code> option, as the two are mutually exclusive.</p> <p>Do not enable this option for RAMs with asynchronous read/write clocks. If <code>rw_check</code> is enabled on block RAM with an asynchronous read clock (<code>rclk</code>) and write clock (<code>wclk</code>), the tool inserts extra logic and a timing path between <code>wclk</code> and <code>rclk</code>. If the clocks are asynchronous to each other, this path can produce glitches on hardware.</p> |
|                         | <p>There are other read-write check controls. See <a href="#">Read-Write Address Checks , on page 751</a> for details about the differences.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <code>select_ram</code> | Implements distributed RAM using the resources in the CLBs. For distributed RAMs, the write operation must be synchronous and the read operation asynchronous.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <code>uram</code>       | When the <code>uram</code> option is specified, the tool forces the RAM to infer the UltraRAM (URAM) block.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

---

## RAM Type Values and Implementations

The table lists RAM implementation information including `ramType` values.

| Values                                                                          | Implementation                                                                                                            |
|---------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
|                                                                                 | Default: <code>block_ram</code> . If not possible, <code>select_ram</code> .                                              |
| <code>registers</code>                                                          | Registers and combinational logic                                                                                         |
| <code>block_ram</code>                                                          | Block RAM                                                                                                                 |
| <code>block_ram</code> ,<br><code>no_rw_check</code> /<br><code>rw_check</code> | Block RAM with/without glue logic                                                                                         |
| <code>block_ram</code> ,<br><code>area</code>                                   | Same as <code>block_ram</code> , but with wider, more area-efficient block RAMs instead of the default deeper block RAMs. |
| <code>block_ram</code> ,<br><code>power</code>                                  | Power and area-efficient block RAM                                                                                        |
| <code>select_ram</code>                                                         | Distributed RAM                                                                                                           |

| Values                | Implementation                                  |
|-----------------------|-------------------------------------------------|
| no_rw_check,<br>power | Power and area-efficient RAM with no glue logic |
| uram, no_uram         | UltraRAM block                                  |

## Constraint Editor Example

|   | Enabled                             | Object Type | Object   | Attribute    | Value       | Val Type | Description                            |
|---|-------------------------------------|-------------|----------|--------------|-------------|----------|----------------------------------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <global> | syn_ramstyle | select_uram | string   | Special implementation of inferred RAM |

If you edit a constraint file to apply `syn_ramstyle`, be sure to include the range of the signal with the signal name. For example:

```
define_attribute {mem[7:0]} syn_ramstyle {registers};
define_attribute {mem[7:0]} syn_ramstyle {block_ram};
```

## Verilog Example

```
module RAMB4_S4 (data_out, ADDR, data_in, EN, CLK, WE, RST);
    output [3:0] data_out;
    input [7:0] ADDR;
    input [3:0] data_in;
    input EN, CLK, WE, RST;
    reg [3:0] mem [255:0] /* synthesis syn_ramstyle="select_uram" */;
    reg [3:0] data_out;

    always@ (posedge CLK)
        if (EN)
            if (RST == 1)
                data_out <= 0;
            else
                begin
                    if (WE == 1)
                        data_out <= data_in;
                    else
                        data_out <= mem[ADDR];
                end
        always @ (posedge CLK)
            if (EN && WE) mem[ADDR] = data_in;
endmodule
```

## VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.ALL;
library synplify;

entity RAMB4_S4 is
    port (ADDR: in std_logic_vector(7 downto 0);
          data_in : in std_logic_vector(3 downto 0);
          WE : in std_logic;
          CLK : in std_logic;
          RST : in std_logic;
          EN : in std_logic;
          data_out : out std_logic_vector(3 downto 0) );
end RAMB4_S4;

architecture rtl of RAMB4_S4 is
type mem_type is array (255 downto 0) of std_logic_vector (3 downto 0);
signal mem : mem_type;
-- mem is the signal that defines the RAM
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "select_ram"; begin
process (CLK)
begin
    IF (CLK'event AND CLK = '1') THEN
        IF (EN = '1') THEN
            IF (RST = '1') THEN
                data_out <= "0000";
            ELSE
                IF (WE = '1') THEN
                    data_out <= data_in;
                ELSE
                    data_out <= mem(to_integer(unsigned(ADDR)));
                END IF;
            END IF;
        END IF;
    END IF;
end process;
```

```

process (CLK)
begin
  IF (CLK'event AND CLK = '1') THEN
    IF (EN = '1' AND WE = '1') THEN
      mem(to_integer(unsigned(ADDR))) <= data_in;
    END IF;
  END IF;
end process;

end rtl;

```

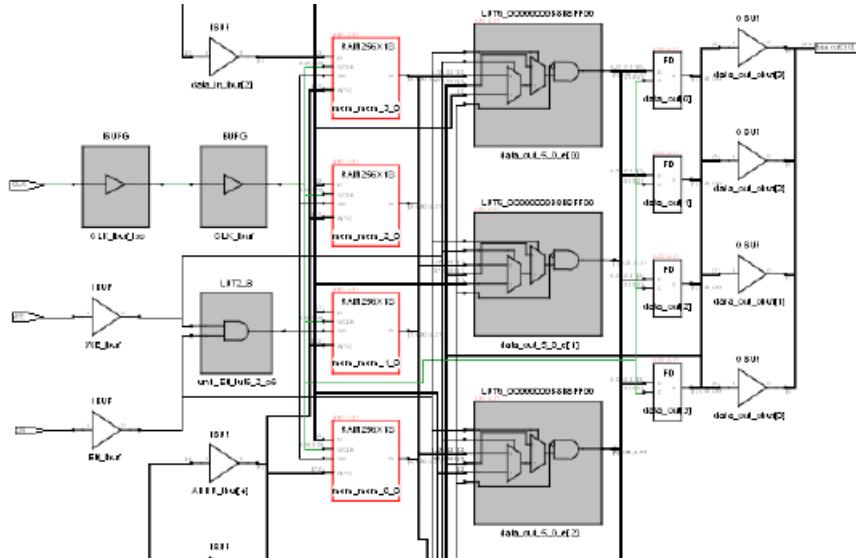
## Effect of Setting syn\_ramstyle

The following are some examples of how the syn\_ramstyle setting directs the implementation of RAM.

## Distributed RAM (Select RAM) Example

**Verilog**    reg [3:0] mem [255:0] /\* synthesis syn\_ramstyle = "select\_ram" \*/;

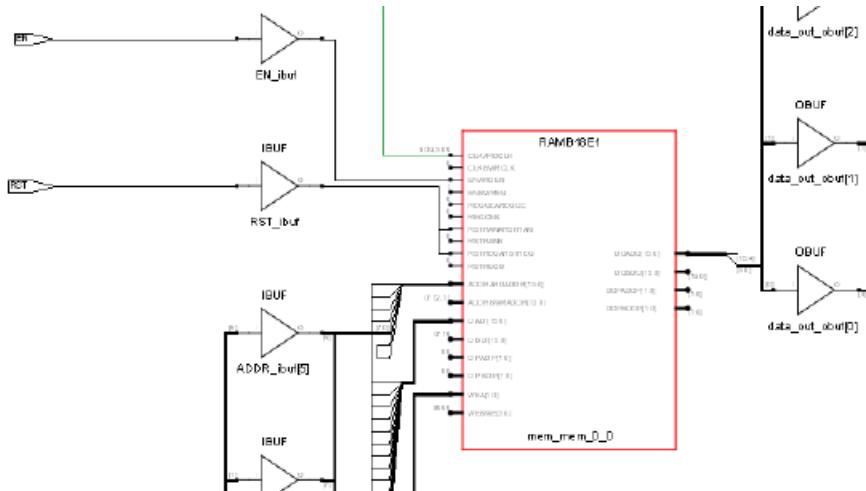
**VHDL**    attribute syn\_ramstyle of mem : signal is "select\_ram";



## Block RAM Example

**Verilog**    `reg [3:0] mem [255:0] /* synthesis syn_ramstyle = "select_ram" */;`

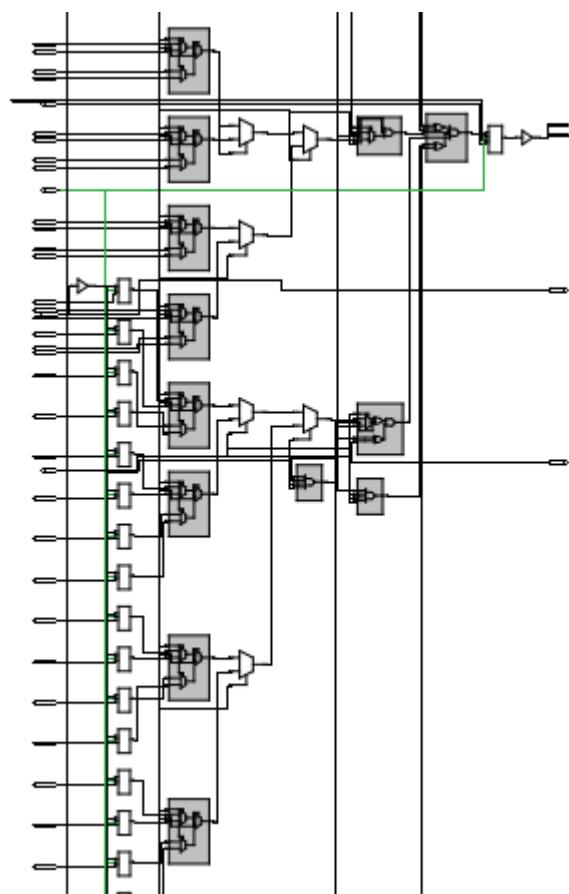
**VHDL**    `attribute syn_ramstyle of mem : signal is "select_ram";`



## Registers Example

**Verilog**    `reg [3:0] mem [255:0] /* synthesis syn_ramstyle = "registers" */;`

**VHDL**    `attribute syn_ramstyle of mem : signal is "registers";`



## **syn\_ram\_write\_mem**

### *Attribute*

Generates the memory map information (MMI) file for block RAM.

### **Description**

Generates the memory map information (MMI) file for block RAM. This MMI file is supported by the UpdateMEM utility of the Xilinx Vivado tool and provides a way to modify the contents of the RAM without having to rerun synthesis and place and route. This flow produces consistent quality of results, when only the memory contents has changed.

### **syn\_ram\_write\_mem Values**

| Value | Description                                                              | Object       | Default | Global |
|-------|--------------------------------------------------------------------------|--------------|---------|--------|
| 1 0   | Enables or disables that the MMI file is generated for the RAM instance. | RAM Instance | None    | No     |

### **syn\_ram\_write\_mem Syntax**

|         |                                                            |                                 |
|---------|------------------------------------------------------------|---------------------------------|
| FDC     | define_attribute {objectName {syn_ram_write_mem {0 1}}     | <a href="#">FDC Example</a>     |
| Verilog | object /* synthesis syn_ram_write_mem = 0 1 */             | <a href="#">Verilog Example</a> |
| VHDL    | attribute syn_ram_write_mem of object : objectType is 0 1; | <a href="#">VHDL Example</a>    |

### **FDC Example**

```
define_attribute {i:bram} {syn_ram_write_mem} {1|0}
```

### **Verilog Example**

```
module test (addra,addrb,clka,clkb,wra,web,din,out);
parameter data_width = 32;
parameter addr_width = 10;
```

```

input [addr_width-1:0] addra, addrb;
input clka, clkb, wea, web;
input [data_width-1:0] din;
output reg [data_width-1:0] out;

reg [data_width-1:0] mem [2**addr_width-1:0]
/* synthesis syn_ram_write_mem = 1 */;

// ram code

always@ (posedge clka)
begin
    if (wea)
        mem[addra] <= din;
end

always@ (posedge clkb)
begin
    if (web)
        out <= mem[addrb];
    else
        out <= mem[addrb];
end
endmodule

```

## VHDL Example

```

Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Dual_Port_ReadFirst is
generic (data_width: integer :=32;
address_width: integer :=10);

port (write_enable: in std_logic;
write_clk, read_clk: in std_logic;
data_in: in std_logic_vector (data_width-1 downto 0);
data_out: out std_logic_vector (data_width-1 downto 0);
write_address: in std_logic_vector (address_width-1 downto 0);
read_address: in std_logic_vector (address_width-1 downto 0));
end Dual_Port_ReadFirst;

```

```
architecture behavioral of Dual_Port_ReadFirst is
type memory is array (2**address_width-1) downto 0)
    of std_logic_vector (data_width-1 downto 0);
signal mem: memory;
signal reg_write_address: std_logic_vector
    (address_width-1 downto 0);
signal reg_write_enable: std_logic;
attribute syn_ram_write_mem: boolean;
attribute syn_ram_write_mem of mem: signal is true;
begin
register_enable_and_write_address:
process (write_clk,write_enable,write_address,data_in)
begin
if (rising_edge(write_clk)) then
reg_write_address <= write_address;
reg_write_enable <= write_enable;
end if;
end process;

write:
process (read_clk,write_enable,write_address,data_in)
begin
if (rising_edge(write_clk)) then
if (write_enable='1') then
mem(conv_integer(write_address)) <= data_in;
end if;
end if;
end process;

read:
process (read_clk,write_enable,read_address,write_address)
begin
if (rising_edge(read_clk)) then
data_out <= mem(conv_integer(read_address));
end if;
end process;
end behavioral;
```

## Effect of Using `syn_ram_write_mem`

For this example, look for the `ram_0.mmi` file in the `par_1/mmifiles` directory of the Implementation Results directory. Notice that the MMI file contains the physical RAM description and placement information.

```
1 ix;<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
2 <MemInfo Minor="0" Version="1">
3
4   <!--Copyright (C) 1994-2017 Synopsys, Inc. This Synopsys
5 This file is generated by Synplify Premier, version N-2017
6
7   <Processor Endianness="little" InstPath="ram_inst">
8     <AddressSpace Begin="0" End="4096" Name="dummy">
9       <BusBlock>
10      <BitLane MemType="RAMB32" Placement="X13Y18">
11        <!--Instance Path = mem_mem_0_0-->
12        <DataWidth LSB="0" MSB="31"/>
13        <AddressRange Begin="0" End="1023"/>
14        <Parity NumBits="0" ON="false"/>
15      </BitLane>
16      </BusBlock>
17    </AddressSpace>
18  </Processor>
19
20  <Config>
21    <Option Name="Part" Val="xcvu065-ffvc1517-3-e"/>
22  </Config>
23
24</MemInfo>
```

## **syn\_reduce\_controlset\_size**

### *Attribute*

Specifies the minimum size of the unique control-set to customize resource usage.

### **syn\_reduce\_controlset\_size Values**

### **Description**

Control sets are unique combinations of clock, clock-enable, and synchronous reset/set signals. There can be packing problems with some architectures because they have more registers with the same control signals per slice. To help eliminate packing problems, the control-set optimizations move some or all of the control pins for the registers to the D inputs.

The `syn_reduce_controlset_size` attribute sets the minimum size of the unique control-set on which control-set optimizations can occur. It lets you override the default settings for the tool. The control-set optimizations are not implemented for registers with timing critical paths and IOB registers.

Use the `syn_reduce_controlset_size` attribute with caution. If you do not choose a value correctly for this attribute, utilization can increase and the design may not fit into the target technology.

### **syn\_reduce\_controlset\_size Syntax**

|             |                                                                                                                                                        |                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| FDC<br>File | <code>define_attribute {v:object} syn_reduce_controlset_size {value}</code><br><code>define_global_attribute syn_reduce_controlset_size {value}</code> | FDC<br>Example     |
| Verilog     | <code>object /* synthesis syn_reduce_controlset_size = "value" */</code>                                                                               | Verilog<br>Example |
| VHDL        | <code>attribute syn_reduce_controlset_size of object :<br/>objectType is "value";</code>                                                               | VHDL<br>Example    |

### **FDC Example**

```
define_global_attribute syn_reduce_controlset_size 2
```

## Constraints Editor Example

|   | Enabled                             | Object Type | Object   | Attribute                  | Value | Val Type |
|---|-------------------------------------|-------------|----------|----------------------------|-------|----------|
| 1 | <input checked="" type="checkbox"/> | global      | <global> | syn_reduce_controlset_size | 3     |          |

## Verilog Example

```

module test(i1, r1,s1, e1, clk, o1) /* synthesis
    syn_reduce_controlset_size = 3 */;
    input [3:1] i1;
    input clk;
    input r1;
    input s1;
    input [2:1] e1;
    output [3:1] o1;
    reg reg1, reg2, reg3;

//reg1 FDRSE
always@ (posedge clk)
if (r1)
    reg1 <= 1'b0;
else if (s1)
    reg1 <= 1'b1;
else if (e1[1])
    reg1 <= i1[1];

//reg2 FDRSE
always@ (posedge clk)
if (r1)
    reg2 <= 1'b0;
else if (s1)
    reg2 <= 1'b1;
else if (e1[1])
    reg2 <= i1[2];

//reg3 FDRSE
always@ (posedge clk)
if (r1)
    reg3 <= 1'b0;
else if (s1)
    reg3 <= 1'b1;
else if (e1[2])
    reg3 <= i1[3];

assign o1 = {reg3,reg2,reg1};
endmodule

```

## VHDL Example

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

entity test is
    port (rst : in std_logic;
          set : in std_logic;
          en : in std_logic_vector(1 downto 0);
          clk : in std_logic;
          din : in std_logic_vector (2 downto 0);
          dout : out std_logic_vector (2 downto 0) );
end entity test;

architecture test_arch of test is
attribute syn_reduce_controlset_size : integer;
attribute syn_reduce_controlset_size of test_arch :
    architecture is 3; begin
process(clk, rst, set, en(0))
begin
    if (clk='1' and clk'event) then
        if (rst = '1') then
            dout(0) <= '0';
        else if (set = '1') then
            dout(0) <= '1';
        else if (en(0) = '1') then
            dout(0) <= din(0);
        end if;
    end if;
    end if;
    end if;
end process;

process(clk, rst, set, en(0))
begin
    if (clk='1' and clk'event) then
        if (rst = '1') then
            dout(1) <= '0';
        else if (set = '1') then
            dout(1) <= '1';
        else if (en(0) = '1') then
            dout(1) <= din(1);
        end if;
    end if;
    end if;
    end if;
end process;
```

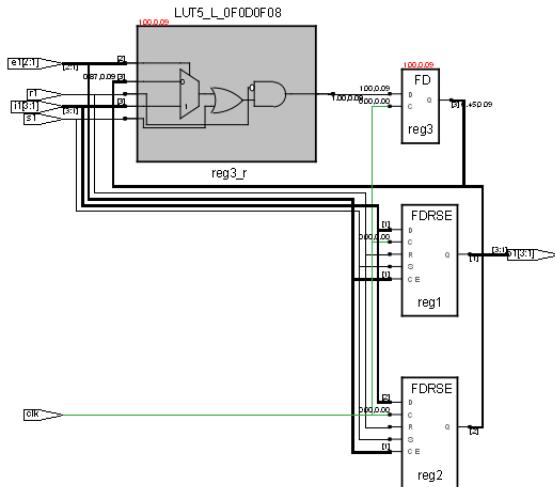
```

process(clk, rst, set, en(1))
begin
    if (clk='1' and clk'event) then
        if (rst = '1') then
            dout(2) <= '0';
        else if (set = '1') then
            dout(2) <= '1';
        else if (en(1) = '1') then
            dout(2) <= din(2);
        end if;
    end if;
    end if;
end process;
end test_arch;

```

## Effect of Using syn\_reduce\_controlset\_size

The following examples show how syn\_controlset\_size affects the implementation of control-sets. This is the design without the attribute. The default value is 2.



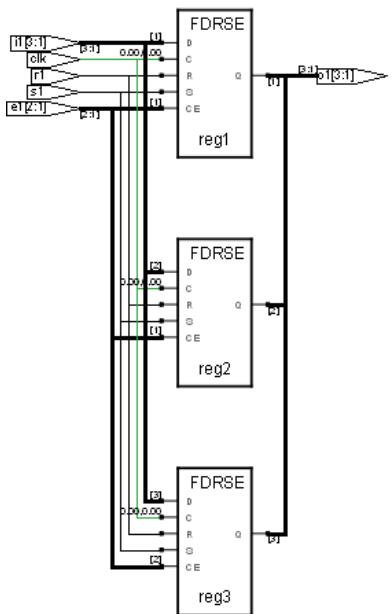
### Example 1: syn\_reduce\_controlset\_size = 1

Verilog   /\* synthesis syn\_reduce\_constsolt\_size=1 \*/;

---

VHDL   attribute syn\_reduce\_constsolt\_size of test\_arch :  
architecture is 1;

---



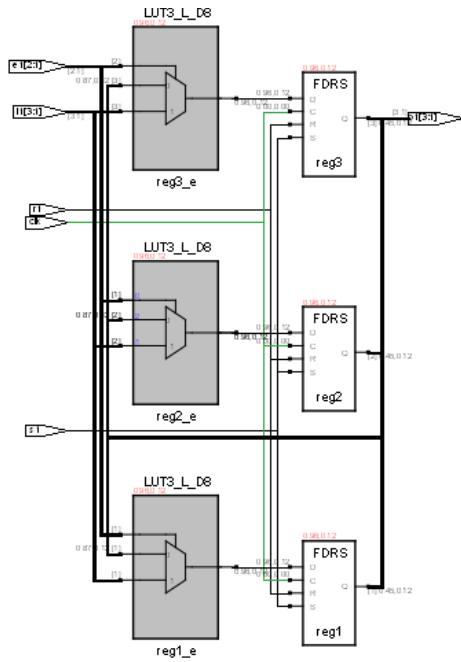
### Example 2: syn\_reduce\_controlset\_size = 3

Verilog   /\* synthesis syn\_reduce\_constsolt\_size=3 \*/;

---

VHDL   attribute syn\_reduce\_constsolt\_size of test\_arch :  
architecture is 3;

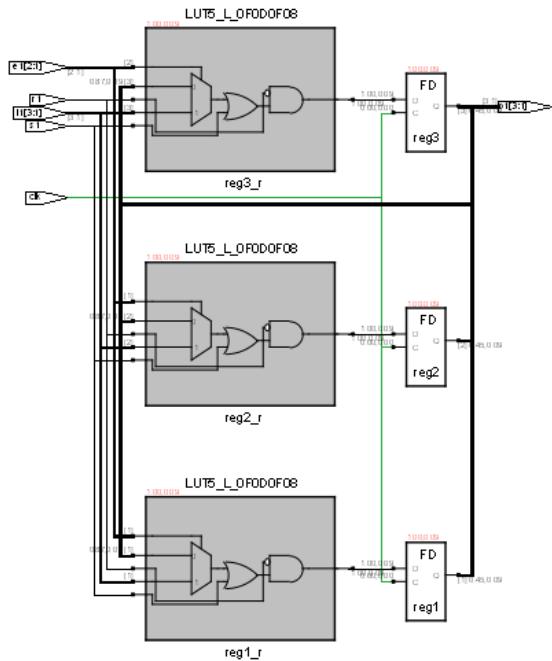
---



### Example 3: syn\_reduce\_controlset\_size = 4

**Verilog**   `/* synthesis syn_reduce_constrolset_size=4 */;`

**VHDL**   `attribute syn_reduce_controlset_size of test_arch : architecture is 4;`



## **syn\_reference\_clock**

### *Attribute*

Specifies a clock frequency other than the one implied by the signal on the clock pin of the register.

### **Description**

`syn_reference_clock` is a way to change clock frequencies other than using the signal on the clock pin. For example, when flip-flops have an enable with a regular pattern, such as every second clock cycle, use `syn_reference_clock` to have timing analysis treat the flip-flops as if they were connected to a clock at half the frequency.

To use `syn_reference_clock`, define a new clock, then apply its name to the registers you want to change. You apply `syn_reference_clock` only in a constraint file; you cannot use it in source code.

You can also use `syn_reference_clock` to constrain multiple-cycle paths through the enable signal. Assign the `find` command to a collection (`clock_enable_col`), then refer to the collection when applying the `syn_reference_clock` constraint.

### **syn\_reference\_clock Syntax**

FDC      `define_attribute {register} syn_reference_clock {clockName}`

[FDC Example](#)

### **FDC Example**

```
define_attribute {myreg[31:0]} syn_reference_clock {sloClock}
```

### **Constraints Editor Example**

| Enable                              | Object Type | Object   | Attribute           | Value | Value Type | Description              |
|-------------------------------------|-------------|----------|---------------------|-------|------------|--------------------------|
| <input checked="" type="checkbox"/> | <any>       | <Global> | syn_reference_clock | 1     | string     | Override the default ... |

### **Effect of using `syn_reference_clock`**

The following figure shows the report before applying the attribute:

| Performance Summary |                     |                     |                  |                  |         |            |                    |
|---------------------|---------------------|---------------------|------------------|------------------|---------|------------|--------------------|
| Starting Clock      | Requested Frequency | Estimated Frequency | Requested Period | Estimated Period | Slack   | Clock Type | Clock Group        |
| clk                 | 2.0 MHz             | 1609.5 MHz          | 500.000          | 0.621            | 499.379 | declared   | default_clkgroup_0 |
| ref_clk             | 1.0 MHz             | NA                  | 1000.000         | NA               | NA      | declared   | default_clkgroup_1 |

This is the report after applying the attribute:

| Performance Summary |                     |                     |                  |                  |         |            |                    |
|---------------------|---------------------|---------------------|------------------|------------------|---------|------------|--------------------|
| Starting Clock      | Requested Frequency | Estimated Frequency | Requested Period | Estimated Period | Slack   | Clock Type | Clock Group        |
| clk                 | 2.0 MHz             | NA                  | 500.000          | NA               | NA      | declared   | default_clkgroup_0 |
| ref_clk             | 1.0 MHz             | 1609.5 MHz          | 1000.000         | 0.621            | 999.379 | declared   | default_clkgroup_1 |

## **syn\_rename\_module**

*Directive*

Renames a module.

### **Description**

This directive renames the specified module. It is typically used to rename submodules to avoid possible naming conflicts between a generated name and the original module name. The syn\_rename\_module directive is specified through the CDC compiler directives file. The synthesis tool automatically replicates any necessary registers during optimization when fixing fanouts, packing I/Os, or improving the quality of results.

### **syn\_rename\_module Syntax**

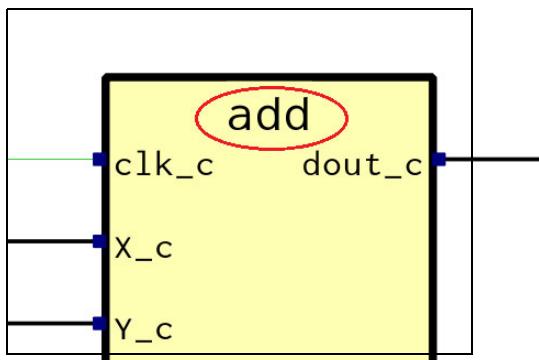
CDC File      **define\_directive {modName} syn\_rename\_module {newModName}**      [CDC File Example](#)

### **CDC File Example**

```
define_directive {v:work.add} syn_rename_module {add_new}
```

### **Effect of using syn\_rename\_module**

The following figure shows the original design, before applying syn\_rename\_module:



The next figure shows the modules with new names after applying `syn_rename_module`:



## **syn\_replicate**

### *Attribute*

Controls replication of registers during optimization.

### **Description**

The synthesis tool automatically replicates registers while optimizing the design and fixing fanouts, packing I/Os, or improving the quality of results.

If area is a concern, you can use this attribute to disable replication either globally or on a per-register basis. When you disable replication globally, it disables I/O packing and other QoR optimizations. When it is disabled, the synthesis tool uses only buffering to meet maximum fanout guidelines.

To disable I/O packing on specific registers, set the attribute to 0. Similarly, you can use it on a register between clock boundaries to prevent replication. Take an example where the tool replicates a register that is clocked by clk1 but whose fanin cone is driven by clk2, even though clk2 is an unrelated clock in another clock group. By setting the attribute for the register to 0, you can disable this replication.

Do not use the syn\_replicate attribute with the Synthesis Strategy fast option.

### **syn\_replicate Syntax**

|         |                                                                                                                   |                                            |
|---------|-------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_global_attribute syn_replicate {0 1};</code>                                                         | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_replicate = 1 0 */;</code>                                                          | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_replicate : boolean;<br/>attribute syn_replicate of object : objectType is true false;</code> | <a href="#">VHDL Example</a>               |

### **Constraints Editor Example**

| Enabled                             | Object Type | Object   | Attribute     | Value | Val Type | Description                       | Comment |
|-------------------------------------|-------------|----------|---------------|-------|----------|-----------------------------------|---------|
| <input checked="" type="checkbox"/> | global      | <global> | syn_replicate | 0     | boolean  | Controls replication of registers |         |

## Verilog Example

```

module norep (Reset, Clk, Drive, OK, ADPad, IPad, ADOut);
    input Reset, Clk, Drive, OK;
    input [6:0] ADOut;
    inout [6:0] ADPad;
    output [6:0] IPad;
    reg [6:0] IPad;
    reg DriveA /* synthesis syn_replicate = 0 */;
    assign ADPad = DriveA ? ADOut : 32'bz;

    always @ (posedge Clk or negedge Reset)
        if (!Reset)
            begin
                DriveA <= 0;
                IPad <= 0;
            end
        else
            begin
                DriveA <= Drive & OK;
                IPad <= ADPad;
            end
    endmodule

```

## VHDL Example

```

library IEEE;
use ieee.std_logic_1164.all;

entity norep is
    port (Reset : in std_logic;
          Clk : in std_logic;
          Drive : in std_logic;
          OK : in std_logic;
          ADPad : inout std_logic_vector (6 downto 0);
          IPad : out std_logic_vector (6 downto 0);
          ADOut : in std_logic_vector (6 downto 0) );
end norep;

architecture archnorep of norep is
    signal DriveA : std_logic;
    attribute syn_replicate : boolean;
    attribute syn_replicate of DriveA : signal is false;
begin
    ADPad <= ADOut when DriveA='1' else (others => 'Z');
    process (Clk, Reset)
    begin

```

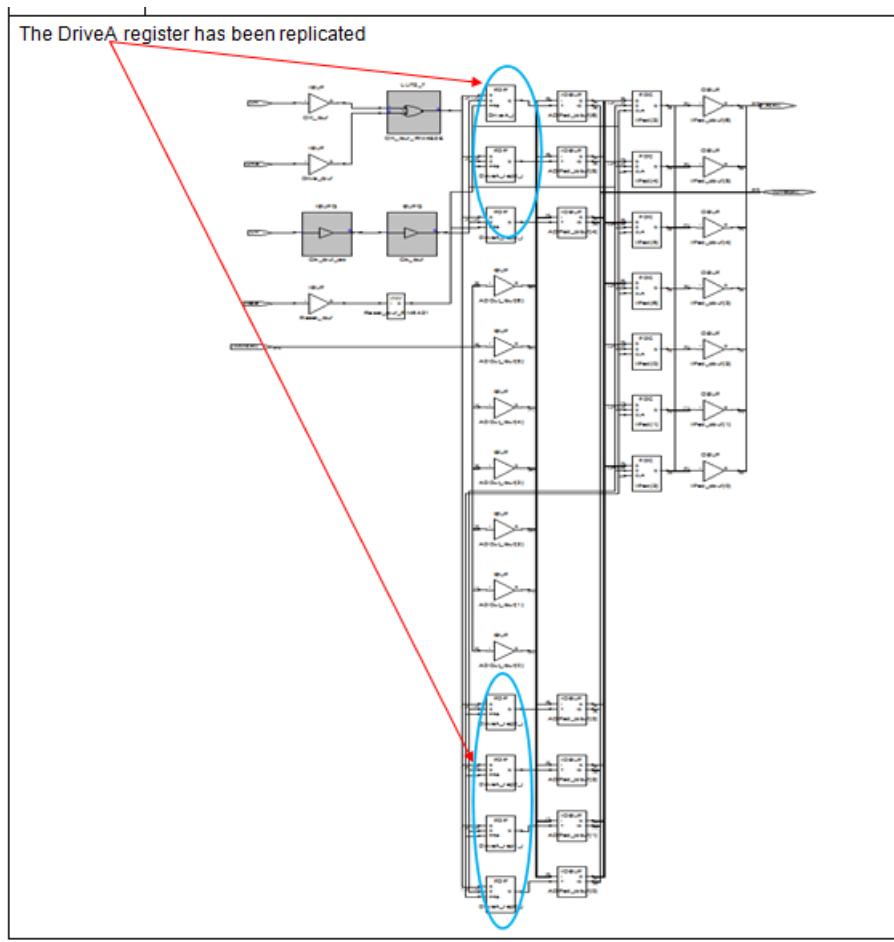
```
if Reset='0' then
    DriveA <= '0';
    IPad <= (others => '0');
elsif rising_edge(clk) then
    DriveA <= Drive and OK;
    IPad <= ADPad;
end if;
end process;
end archnorep;
```

## Effect of Using syn\_replicate

The following example shows a design without the syn\_replicate attribute:

Verilog      `reg DriveA /*synthesis syn_replicate = 1*/`

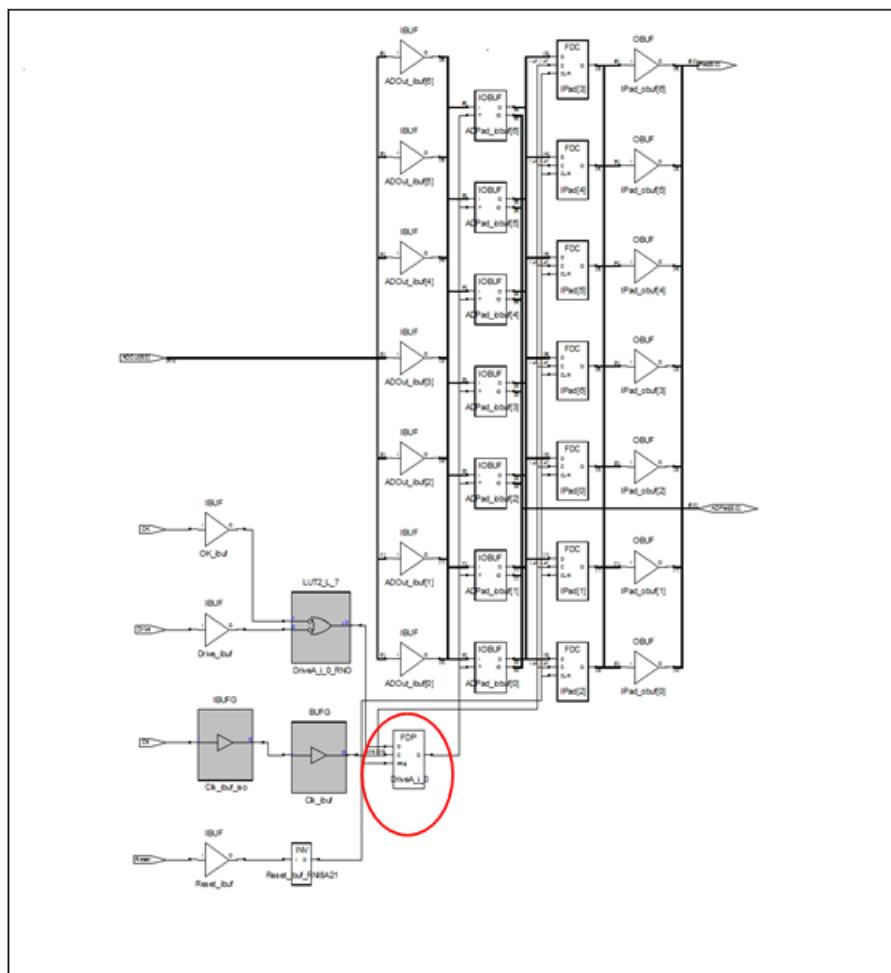
VHDL      `attribute syn_replicate : boolean;`  
             `attribute syn_replicate of DriveA : signal is true;`



When you apply `syn_replicate`, the registers are not duplicated:

Verilog      `reg DriveA /*synthesis syn_replicate = 0*/`

VHDL      `attribute syn_replicate : boolean;`  
             `attribute syn_replicate of DriveA : signal is false;`



## **syn\_resources**

### *Attribute*

Specifies the resources used inside a black box.

### **Description**

Specifies the resources used inside a black box. This attribute is applied to Verilog black-box modules and VHDL architectures or component definitions.

### **syn\_resources Syntax**

The following table summarizes the syntax in different files.

|         |                                                                                                                        |  | FDC Example     |
|---------|------------------------------------------------------------------------------------------------------------------------|--|-----------------|
| Verilog | <code>object /* synthesis syn_resources = "resource=value" */;</code>                                                  |  | Verilog Example |
| VHDL    | <code>attribute syn_resources : string;<br/>attribute syn_resources of object : objectType is "resource=value";</code> |  | VHDL Example    |

The `resource=value` entry for this attribute can be specified with any combination of the following:

| Resource = Value                | Description                          |
|---------------------------------|--------------------------------------|
| <code>luts=integer</code>       | Number of lookup tables              |
| <code>regs=integer</code>       | Number of registers                  |
| <code>blockrams=integer</code>  | Number of RAM resources              |
| <code>blockmults=integer</code> | Number of block multiplier resources |
| <code>dsp_blocks=integer</code> | Number of DSP or DSP48 blocks        |

The value listed in the area usage report is the larger of the `luts` or `regs` value.

## FDC Example

You can apply the attribute to more than one type of resource at a time by separating assignments with a comma (,). For example:

```
define_attribute {v:bb} syn_resources {corecells=300, blockrams=5}
define_attribute {v:bb} syn_resources {luts=500, blockrams=10}
```

## Constraints Editor Example

|   | Enable                              | Object Type | Object | Attribute     | Value                      | Value Type | Description                                     |
|---|-------------------------------------|-------------|--------|---------------|----------------------------|------------|-------------------------------------------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | v:bb   | syn_resources | corecells=300, blockrams=5 | string     | Specifies the resources used inside a black box |

## Verilog Example

In Verilog, you can only attach this attribute to a module. Here is an example:

```
module bb (o,i) /* synthesis syn_black_box syn_resources =
    "luts=500,regs=463,blockrams=10,blockmults=2" */;
  input i;
  output o;
endmodule

module top_bb (o,i);
  input i;
  output o;
  bb u1 (o,i);
endmodule
```

## VHDL Example

In VHDL, this attribute can be placed on either an architecture or a component declaration.

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

entity top is
  port (o : out std_logic;
        i : in std_logic
       );
end top;
```

```
architecture top_rtl of top is
component bb
    port (o : out std_logic;
          i : in std_logic);
end component;

begin
    U1: bb port map(o, i);
end top_rtl;

--black box entity
library ieee;
use ieee.std_logic_1164.all;

entity bb is
port (o : out std_logic;
      i : in std_logic
     );
end bb;

architecture rtl of bb is
attribute syn_resources : string;
attribute syn_resources of rtl: architecture is
    "luts=500, regs=463, blockrams=10, blockmults=2";

begin
end rtl;
```

## Effect of Using syn\_resources

You can check the Resource Utilization report in the log file to verify how resources are actually mapped.

```
Resource Usage Report for top_bb

Mapping to part: xc7vx485tffg1157-1
Cell usage:
bb           1 use

I/O ports: 2
I/O primitives: 2
IBUF          1 use
OBUF          1 use

I/O Register bits:          0
Register bits not including I/Os: 463 (0%)

RAM/ROM usage summary
Occupied Block RAM sites (RAMB36) : 10 of 1030 (0%)

DSP48s: 2 of 2800 (0%)
Total load per clock:

Mapping Summary:
Total LUTs: 500 (0%)

Distribution of All Consumed LUTs = RAM
Distribution of All Consumed Luts 500 = 500
```

## syn\_ret\_type

### Directive

Specifies the type of sequential element, such as a flip-flop, latch, RAM, or seqshift, for the custom retention model definition.

### Syntax

**syn\_ret\_type = "type"**

In the above syntax, *type* is the type of sequential element as described in the following table:

| Type of Sequential Element | Description                                   |
|----------------------------|-----------------------------------------------|
| Flip-flops                 |                                               |
| DFF                        | Standard flip-flop                            |
| DFFE                       | Flip-flop with enable                         |
| DFFR                       | Flip-flop with async reset                    |
| DFFRE                      | Flip-flop with enable and async reset         |
| DFFS                       | Flip-flop with async set                      |
| DFFSE                      | Flip-flop with enable and async set           |
| DFFRS                      | Flip-flop with async reset and set            |
| DFFRSE                     | Flip-flop with enable and async set and reset |
| SDFFR                      | Flip-flop with sync reset                     |
| SDFFRE                     | Flip-flop with enable and sync reset          |
| SDFFS                      | Flip-flop with sync set                       |
| SDFFSE                     | Flip-flop with enable and sync set            |
| SDFFRS                     | Flip-flop with sync reset and set             |
| SDFFRSE                    | Flip-flop with enable and sync reset and set  |

| Type of Sequential Element                                                                         | Description              |
|----------------------------------------------------------------------------------------------------|--------------------------|
| <b>Latches</b>                                                                                     |                          |
| LAT                                                                                                | Latch                    |
| LATR                                                                                               | Latch with reset         |
| LATS                                                                                               | Latch with set           |
| LATRS                                                                                              | Latch with reset and set |
| <b>RAM</b>                                                                                         |                          |
| RAM primitives are decomposed and mapped into one of the DFF or SDFF cell types, accordingly.      |                          |
| <b>SeqShift</b>                                                                                    |                          |
| SEQSHIFT primitives are decomposed and mapped into one of the DFF or SDFF cell types, accordingly. |                          |

## Example

```
module ret_dffrse (Q, CLK, SAVE, RESTORE, D, S, R, E)
  /* synthesis syn_implement = "1" syn_upf_ret_type = "DFFRSE" */;
  logic
endmodule
```

## **syn\_ret\_lib\_cell\_type**

### *Directive*

The custom retention model must include this directive and its value must be the same as specified by the -lib\_cell\_type option of the map\_retention\_cell command.

### **Syntax**

```
syn_ret_lib_cell_type = "asicLibCellType"
```

In the above syntax, *asicLibCellType* is the same type of ASIC library cell as specified by the -lib\_cell\_type option of the map\_retention\_cell command.

### **Example**

```
module ret_dffrse (Q, CLK, SAVE, RESTORE, D, S, R, E)
  /* synthesis syn_ret_lib_cell_type = "asicLibCellType" */;
```

## syn\_romstyle

### Attribute

This attribute determines how ROM architectures are implemented.

### Description

By applying the `syn_romstyle` attribute to the signal output value, you can control whether the ROM structure is implemented as discrete logic or technology-specific RAM blocks. By default, small ROMs (less than seven address bits) are implemented as logic, and large ROMs (seven or more address bits) are implemented as RAM.

### syn\_romstyle Syntax

This table summarizes the syntax:

|         |                                                                           |                                            |
|---------|---------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute {romPrimitive} syn_romstyle {logicType}</code>     | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_romstyle = "logicType" */;</code>           | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_romstyle of object: objectType is "logicType";</code> | <a href="#">VHDL Example</a>               |

In the above syntax, the `logicType` variable is defined in the following table:

| Variable   | Description                                                                          |
|------------|--------------------------------------------------------------------------------------|
| logic      | Uses discrete logic primitives.                                                      |
| block_rom  | Specifies that the inferred ROM be mapped to the appropriate vendor-specific memory. |
| select_rom | Implements ROMs using the distributed ROM resources in the CLBs.                     |

### Constraints Editor Example

|   | Enable                              | Object Type | Object   | Attribute    | Value     | Value Type | Description                      | Comment |
|---|-------------------------------------|-------------|----------|--------------|-----------|------------|----------------------------------|---------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Global> | syn_romstyle | block_rom | string     | Controls mapping of inferred ROM |         |
| 2 |                                     |             |          |              |           |            |                                  |         |

## Verilog Example

This Verilog code example applies the `syn_romstyle` value of `block_rom`.

```
module test (clock,addr,dataout)
    /* synthesis syn_romstyle = "block_rom" */;
    input clock;
    input [4:0] addr;
    output [7:0] dataout;
    reg [7:0] dataout;
    reg [4:0] addr_reg;

    always @ (posedge clock)
    begin
        addr_reg<=addr;
        case (addr_reg)
            5'b00000: dataout <= 8'b10000011;
            5'b00001: dataout <= 8'b000000101;
            5'b00010: dataout <= 8'b000001001;
            5'b00011: dataout <= 8'b000001101;
            5'b00100: dataout <= 8'b000010001;
            5'b00101: dataout <= 8'b000011001;
            5'b00110: dataout <= 8'b000100001;
            5'b00111: dataout <= 8'b000110100;
            5'b01000: dataout <= 8'b110000000;
            5'b01000: dataout <= 8'b00011011;
            5'b01001: dataout <= 8'b101100001;
            5'b01010: dataout <= 8'b00110101;
            5'b01011: dataout <= 8'b01110010;
            5'b01100: dataout <= 8'b11100011;
            5'b01101: dataout <= 8'b00111111;
            5'b01110: dataout <= 8'b01010101;
            5'b01111: dataout <= 8'b00110100;
            5'b10000: dataout <= 8'b101100000;
            5'b10000: dataout <= 8'b11111011;
            5'b10001: dataout <= 8'b000010001;
            5'b10010: dataout <= 8'b10110011;
            5'b10011: dataout <= 8'b00101011;
            5'b10100: dataout <= 8'b11101110;
            5'b10101: dataout <= 8'b01110111;
            5'b10110: dataout <= 8'b01110101;
            5'b10111: dataout <= 8'b01000011;
            5'b11000: dataout <= 8'b01011100;
            5'b11000: dataout <= 8'b11101011;
            5'b11001: dataout <= 8'b00010100;
            5'b11010: dataout <= 8'b00110011;
            5'b11011: dataout <= 8'b00100101;
```

```

5'b11100: dataout <= 8'b01001110;
5'b11101: dataout <= 8'b01110100;
5'b11110: dataout <= 8'b11100101;
5'b11111: dataout <= 8'b01111110;
default: dataout <= 8'b00000000;
endcase
end
endmodule

```

## VHDL Example

The following VHDL code example applies the `syn_romstyle` value of `block_rom`.

```

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

entity single_port_rom is
    generic
        (DATA_WIDTH : natural := 8;
         ADDR_WIDTH : natural := 8 );
    port (clk : in std_logic;
          addr : in natural range 0 to 2**ADDR_WIDTH - 1;
          q : out std_logic_vector((DATA_WIDTH-1) downto 0) );
attribute syn_romstyle : string;
attribute syn_romstyle of q : signal is "block_rom";
end entity;

architecture rtl of single_port_rom is
subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;
function init_rom
return memory_t
variable tmp : memory_t := (others => (others => '0'));
begin
    for addr_pos in 0 to 2**ADDR_WIDTH - 1 loop
        tmp(addr_pos) := std_logic_vector(to_unsigned
            (addr_pos, DATA_WIDTH));
    end loop;
    return tmp;
end init_rom;
signal rom : memory_t := init_rom;
begin
    process(clk)
    begin

```

```
if(rising_edge(clk)) then
    q <= rom(addr);
end if;
end process;
end rtl;
```

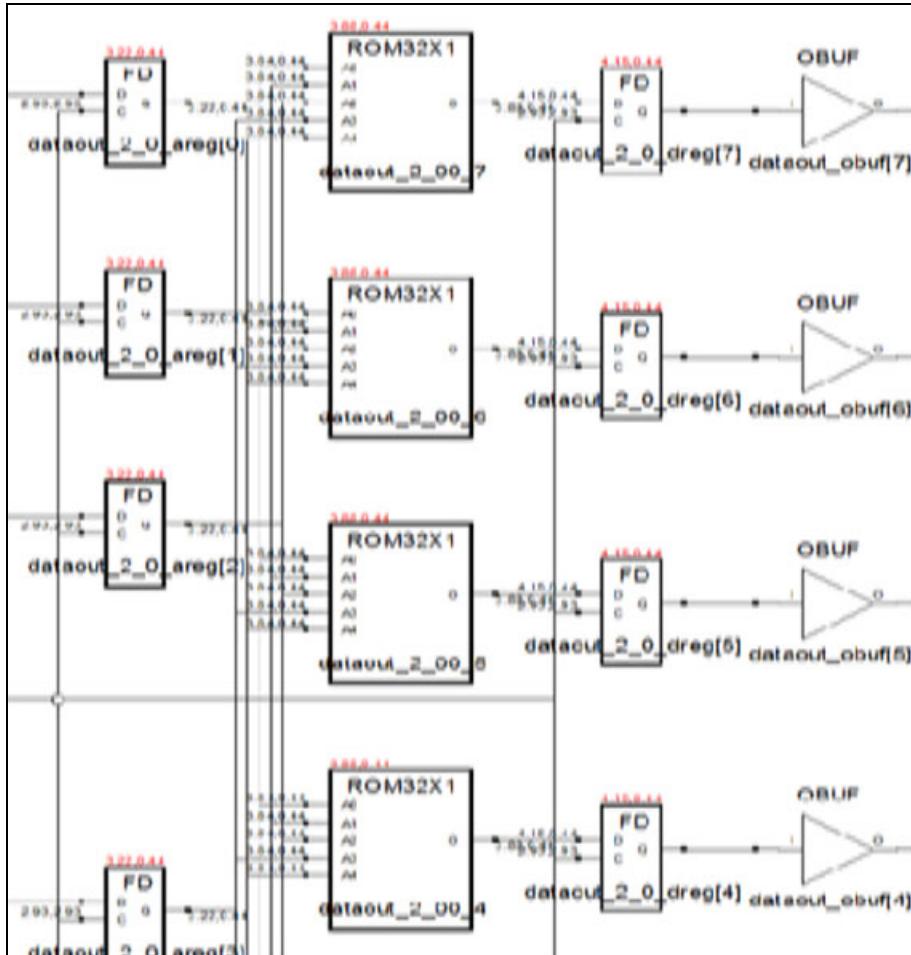
## Effect of Using syn\_romstyle

**Verilog**

```
module test (clock,addr,dataout)
/*synthesis syn_romstyle = "select_rom" */;
```

**VHDL**

```
attribute syn_romstyle: string;
attribute syn_romstyle of q : signal is "select_rom";
```



## **syn\_rw\_conflict\_logic**

*Attribute/Directive*

Infers an appropriate RAM implementation, and controls the insertion of glue logic if the inferred RAM is block RAM.

### Description

The `syn_rw_conflict_logic` allows the synthesis tool to choose other RAM implementations as appropriate, instead of automatically inferring block RAM.

When you read and write to the same block RAM address, the value of the output is indeterminate. An indeterminate output can create a simulation mismatch between RTL and post-synthesis simulations. With the default setting for this attribute, if block RAM is inferred, the synthesis tools automatically generate bypass logic around the RAM to prevent mismatches.

You can specify that the bypass logic not be inserted if the tool implements block RAM, by setting the attribute value to 0. Glue logic insertion only applies to block RAM, not for any other RAM implementations. Use this setting only when you cannot simultaneously read and write to the same RAM location and you want to minimize overhead logic.

This attribute is similar in functionality to the `syn_ramstyle` attribute and its `no_rw_check` value. For a comparison of the different read-write address check options available, see [Read-Write Address Checks, on page 751](#).

### **syn\_rw\_conflict\_logic Syntax**

You can specify `syn_rw_conflict_logic` as a compiler directive or as a constraint in the constraint file. The following table summarizes the syntax:

|         |                                                                                                                                               |                                                               |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
| FDC     | <code>define_attribute {object} syn_rw_conflict_logic {0 1}</code><br><code>define_global_attribute syn_rw_conflict_logic {0 1}</code>        | <a href="#">Constraints Editor</a><br><a href="#">Example</a> |
| Verilog | <code>object /* synthesis syn_rw_conflict_logic = 0 1 */</code>                                                                               | <a href="#">Verilog Example</a>                               |
| VHDL    | <code>attribute syn_rw_conflict_logic : boolean;</code><br><code>attribute syn_rw_conflict_logic of object : objectType is true false;</code> | <a href="#">VHDL Example</a>                                  |

## Constraints Editor Example

|   | Enabled                             | Object Type | Object   | Attribute             | Value |
|---|-------------------------------------|-------------|----------|-----------------------|-------|
| 1 | <input checked="" type="checkbox"/> |             | <global> | syn_rw_conflict_logic | 0     |

## Verilog Example

```
module ram2Kx8 (data0, waddr0,raddr0, we0, clk0, dataout);
parameter d_width = 8;
parameter addr_width = 11;
parameter mem_depth = 2048;
input [d_width-1:0] data0;
input [addr_width-1:0] waddr0, raddr0;
input we0, clk0;
output [d_width-1:0] dataout;
reg [addr_width-1:0] reg_addr0;
reg [d_width-1:0] mem [mem_depth-1:0] /* synthesis
syn_rw_conflict_logic = 0 */;
assign dataout = mem[reg_addr0];

always @ (posedge clk0)
begin
    reg_addr0 <= raddr0;
    if (we0)
        mem[waddr0] <= data0;
end
endmodule
```

## VHDL Example

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

entity ram2Kx8 is port(
data_out : out std_logic_vector(7 downto 0);
data_in : in std_logic_vector(7 downto 0);
raddr0 : in std_logic_vector(10 downto 0);
waddr0 : in std_logic_vector(10 downto 0);
clk, we : in std_logic);
end entity;
```

```
architecture beh of ram2Kx8 is
type mem_type is array (2047 downto 0) of
    std_logic_vector (7 downto 0);
signal raddr0_in : std_logic_vector(10 downto 0);
signal mem : mem_type;
attribute syn_rw_conflict_logic : boolean;
attribute syn_rw_conflict_logic of mem : signal is false ;
begin
data_out <= mem(conv_integer(raddr0_in));
process(clk)
begin
if clk'event and clk='1' then
    if (we = '1') then
        mem(conv_integer(waddr0)) <= data_in;
    end if;
end if;
end process;

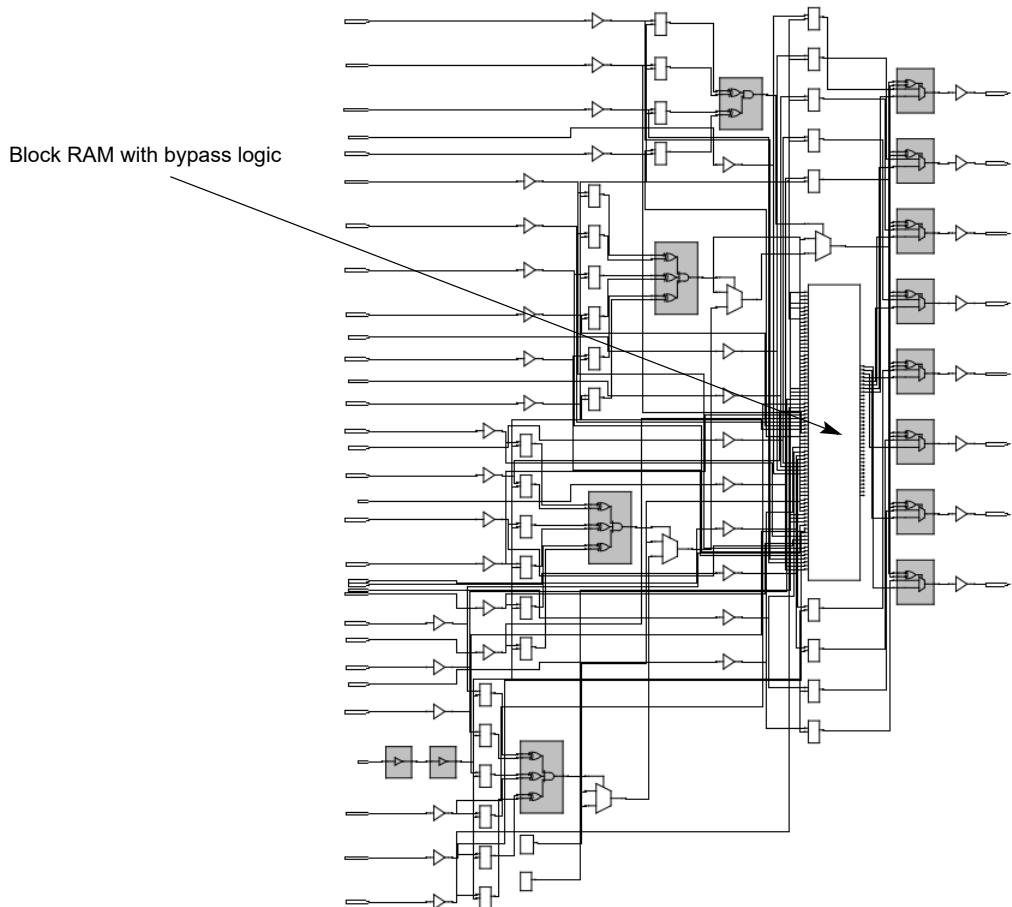
process(clk)
begin
if clk'event and clk='1' then
    raddr0_in <= raddr0;
end if;
end process;
end beh;
```

## Effect of Using syn\_rw\_conflict\_logic

The following figure shows the attribute set to 1. The design shows a block RAM with bypass logic inserted:

Verilog    `reg[d_width-1:0]mem[mem_dep-1:0]`  
          `/*synthesis syn_rw_conflict_logic = 1*/;`

VHDL    `attribute syn_rw_conflict_logic of mem : signal is true ;`

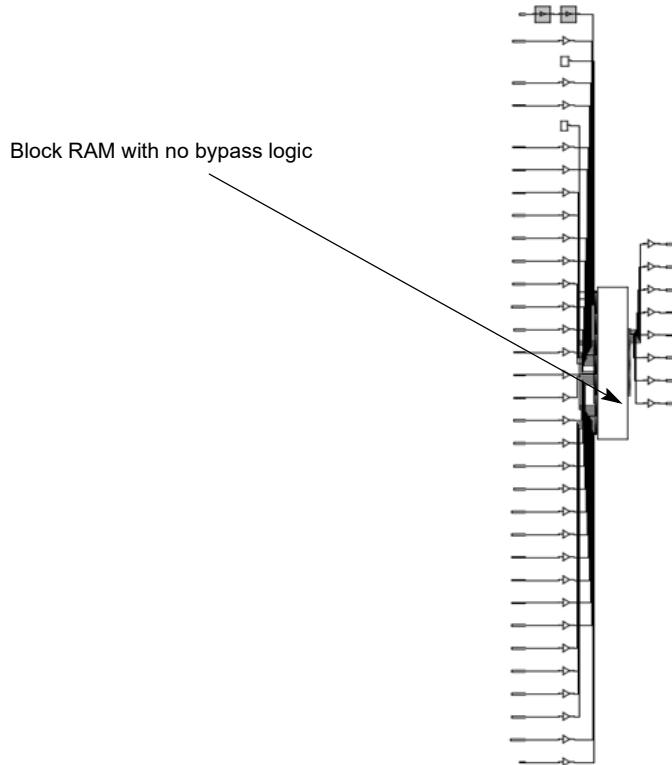


The next figure shows the attribute set to 0. The tool does not insert bypass logic with this value:

Verilog    `reg[d_width-1:0]mem[mem_dep-1:0]  
/*synthesis syn_rw_conflict_logic = 0*/;`

VHDL    `attribute syn_rw_conflict_logic of mem : signal is false ;`

---



## syn\_sharing

### *Directive*

Enables or disables the sharing of resources during the compilation stage of synthesis.

### Description

The syn\_sharing directive controls resource sharing during the compilation stage of synthesis. The directive is on by default (resource sharing is enabled). This is a compiler-specific optimization that does not affect the mapper; this means that the mapper might still perform resource sharing optimizations to improve timing, even if syn\_sharing is disabled.

You can also specify global resource sharing with the option set resource\_sharing Tcl command.

If you disable resource sharing globally, you can use the syn\_sharing directive to turn on resource sharing for specific modules or architectures.

### syn\_sharing Syntax

|          |                                                                         |                                 |
|----------|-------------------------------------------------------------------------|---------------------------------|
| CDC File | <code>define_directive {object} {syn_sharing} {on off}</code>           | <a href="#">CDC Example</a>     |
| Verilog  | <code>object /* synthesis syn_sharing = "on off" */ ;</code>            | <a href="#">Verilog Example</a> |
| VHDL     | <code>attribute syn_sharing of object : objectType is "on off" ;</code> | <a href="#">VHDL Example</a>    |

### CDC Example

```
define_directive {v:module_add} {syn_sharing} {on}
```

## Verilog Example

```
module add (a, b, x, y, out1, out2, sel, en, clk)
    /* synthesis syn_sharing=off */;
    input a, b, x, y, sel, en, clk;
    output out1, out2;
    wire tmp1, tmp2;
    assign tmp1 = a * b;
    assign tmp2 = x * y;
    reg out1, out2;

    always@(posedge clk)
        if (en)
            begin
                out1 <= sel ? tmp1: tmp2;
            end
        else
            begin
                out2 <= sel ? tmp1: tmp2;
            end
    endmodule
```

## VHDL Example

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

entity add is
    port (a, b : in std_logic_vector(1 downto 0);
          x, y : in std_logic_vector(1 downto 0);
          clk, sel, en: in std_logic;
          out1 : out std_logic_vector(3 downto 0);
          out2 : out std_logic_vector(3 downto 0));
end add;

architecture rtl of add is
attribute syn_sharing : string;
attribute syn_sharing of rtl : architecture is "on";

signal tmp1, tmp2: std_logic_vector(3 downto 0);
begin
    tmp1 <= a * b;
    tmp2 <= x * y;
```

```

process(clk) begin
    if clk'event and clk='1' then
        if (en='1') then
            if (sel='1') then
                out1 <= tmp1;
            else
                out1 <= tmp2;
            end if;
        else
            if (sel='1') then
                out2 <= tmp1;
            else
                out2 <= tmp2;
            end if;
        end if;
    end if;
end process;
end rtl;

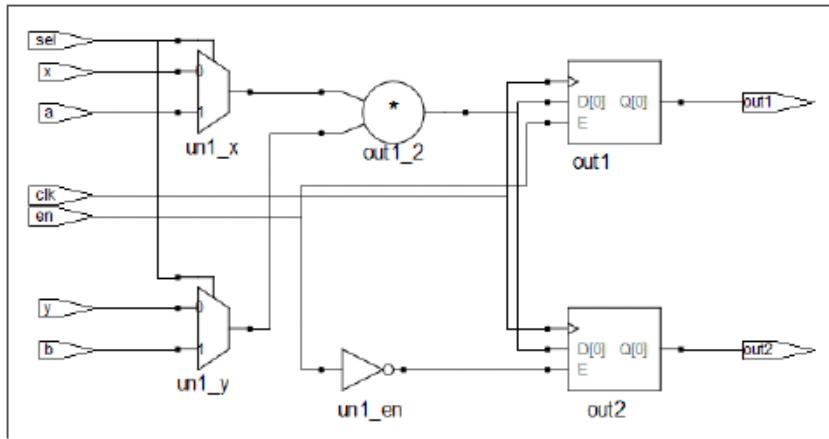
```

## Effect of Using syn\_sharing

The following example shows the default setting, where resource sharing in the compiler is on:

Verilog    module add /\* synthesis syn\_sharing = "on" \*/;

VHDL    attribute syn\_sharing of add : component is "on";



The next figure shows the same design when resource sharing is off, and two adders are inferred:

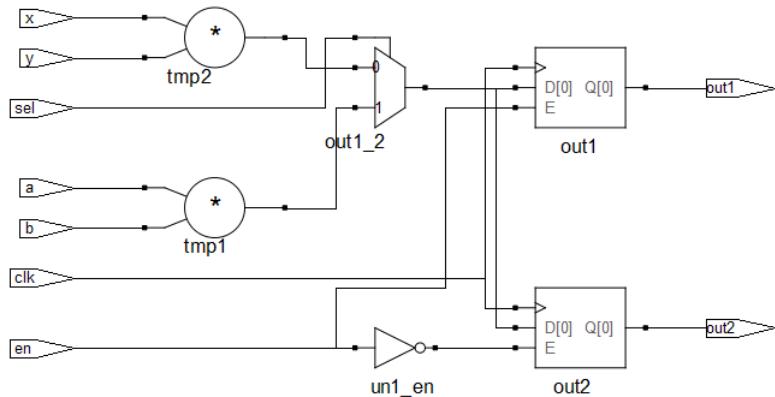
---

Verilog    module add /\* synthesis syn\_sharing = "off" \*/;

---

VHDL    attribute syn\_sharing of add : component is "off";

---



## **syn\_shift\_resetphase**

### *Attribute*

Allows you to remove the flip-flop on the inactive clock edge, built by the reset recovery logic for an FSM when a single event upset (SEU) fault occurs.

### **Description**

When a single event upset (SEU) fault occurs, the FSM can transition to an unreachable state. The `syn_encoding` attribute with a value of `safe` provides a mechanism to build additional logic for recovery to the specified reset state. For an FSM with asynchronous reset, the software inserts an additional flip-flop to the recovery logic path on the opposite edge of the design clock by default to isolate the reset. You can set the `syn_shift_resetphase` attribute to 0 to remove this additional flip-flop on the inactive clock edge, if necessary.

For more information about the `syn_encoding` attribute, see [syn\\_encoding](#), on page 603.

### **syn\_shift\_resetphase Syntax**

|          |                                                                                                                                        |                            |
|----------|----------------------------------------------------------------------------------------------------------------------------------------|----------------------------|
| FDC File | <code>define_attribute object {syn_shift_resetphase} {1 0}</code><br><code>define_global_attribute {syn_shift_resetphase} {1 0}</code> | Constraints Editor Example |
| Verilog  | <code>object /* synthesis syn_shift_resetphase = "1 0" */;</code>                                                                      | Verilog Example            |
| VHDL     | <code>attribute syn_shift_resetphase of object : signal is "true false";</code>                                                        | VHDL Example               |

### **Constraints Editor Example**

|   | Enable                              | Object Type | Object                | Attribute            | Value | Value Type | Description | Comment |
|---|-------------------------------------|-------------|-----------------------|----------------------|-------|------------|-------------|---------|
| 1 | <input checked="" type="checkbox"/> | instance    | i:present_state[11:0] | syn_shift_resetphase | 0     |            |             |         |

The Tcl equivalent is shown below:

```
define_attribute {i:present_state[11:0]}{syn_shift_resetphase}{0}
```

## Verilog Example

Apply the `syn_shift_resetphase` attribute on the top module or state register as shown in the Verilog code segment below.

```
module test (clk, rst, in, out)
    /* synthesis syn_shift_resetphase = 0 */;
    ...

    reg [3:0] present_state
    /* synthesis syn_shift_resetphase = 0 */, next_state;
    ...

endmodule
```

## VHDL Example

Here is a VHDL code segment showing how to use the `syn_shift_resetphase` attribute.

```
entity fsm is
    ...
end fsm;

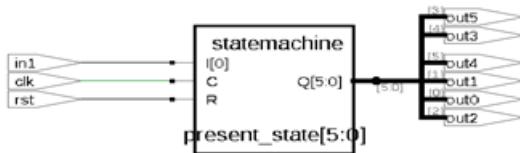
architecture rtl of fsm is
signal present_state : std_logic_vector(3 downto 0);

-- Specifying on the architecture
attribute syn_shift_resetphase : boolean;
attribute syn_shift_resetphase of rtl : architecture is false;

-- Specifying on the state signal
attribute syn_shift_resetphase : boolean;
attribute syn_shift_resetphase of present_state : signal is false;
begin
    ...
end rtl;
```

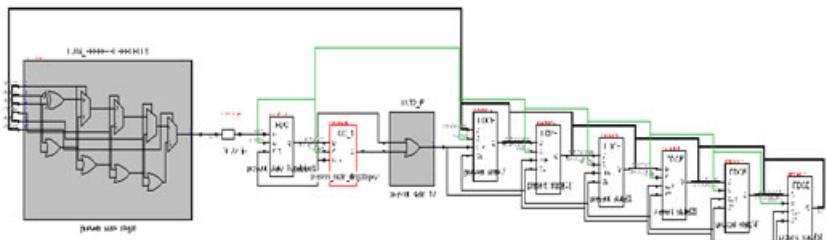
## Effect of Using `syn_shift_resetphase`

Safe encoding is implemented for the following state machine.



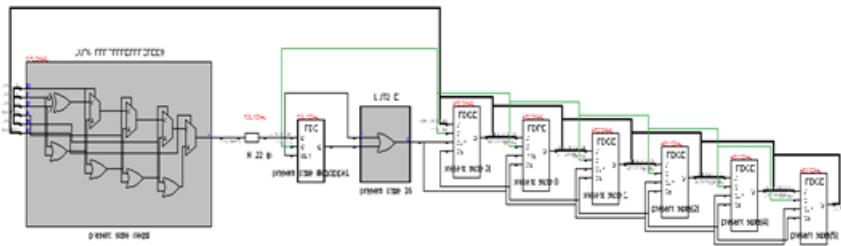
This example shows the results before the `syn_shift_resetphase` attribute is applied.

| Enable | Object Type | Object              | Attribute    | Value       |
|--------|-------------|---------------------|--------------|-------------|
| ✓      | <any>       | ipresent_state[5:0] | syn_encoding | safe,onehot |



This example shows the view results after the `syn_shift_resetphase` attribute is applied.

| Enable | Object Type | Object               | Attribute            | Value       |
|--------|-------------|----------------------|----------------------|-------------|
| ✓      | <any>       | i:present_state[5:0] | syn_encoding         | safe,onehot |
| ✓      | <any>       | <Global>             | syn_shift_resetphase | 0           |



## **syn\_slow**

### *Attribute*

Increases transition time of a specific output port or ports.

### **Description**

The transition time of the output driver can be programmed as either fast or slow. The **syn\_slow** attribute increases the transition time which reduces the noise level in the circuit.

You can speed up the transition time with the **syn\_fast** attribute.

The synthesis tool provides attributes, passed into the XNF/EDIF netlist for Xilinx placement and routing, that affect the input setup times and output transition times for your I/Os. The **syn\_slow** timing attribute gets passed to the Xilinx I/O Block parameter as SLOW in the XNF/EDIF netlist.

### **syn\_slow Syntax**

|          |                                                                     |                                           |
|----------|---------------------------------------------------------------------|-------------------------------------------|
| FDC File | <code>define_attribute {outputPort} syn_slow {1}</code>             | <a href="#">Constraint Editor Example</a> |
| Verilog  | <code>outputPort /* synthesis syn_slow = 1 */;</code>               | <a href="#">Verilog Example</a>           |
| VHDL     | <code>attribute syn_slow of outputPort : objectType is true;</code> | <a href="#">VHDL Example</a>              |

### **Constraint Editor Example**

|   | Enable                              | Object Type | Object       | Attribute | Value | Value Type | Description | Comment |
|---|-------------------------------------|-------------|--------------|-----------|-------|------------|-------------|---------|
| 1 | <input checked="" type="checkbox"/> | <any>       | p:portc[7:0] | syn_slow  | 1     |            |             |         |

```
define_attribute {p:portc[7:0]} {syn_slow} {1}
```

### **Verilog Example**

```
module counter (CLK, RST, DATA0);
  input CLK, RST;
  output [7:0] DATA0 /* synthesis syn_slow = 1 */;
  reg [7:0] DATA0;
```

```
always @ (posedge CLK or posedge RST)
begin
    if (RST)
        DATA0 = 0;
    else
        DATA0 = DATA0 + 1;
end
endmodule
```

## VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity count is
    port (clk : in std_logic;
          rst : in std_logic;
          data0 : out std_logic_vector(7 downto 0) );
end count;

architecture behave of count is
signal data0_i: std_logic_vector(7 downto 0);
attribute syn_slow : boolean;
attribute syn_slow of data0 : signal is true;
begin
    process (rst,clk)
    begin
        if (clk'event and clk = '1') then
            if (rst = '1') then
                data0_i <= "00000000";
            else
                data0_i <= data0_i + "00000001";
            end if;
        end if;
    end process;
    data0 <= data0_i;
end behave;
```

## Effect of Using `syn_slow`

The following table shows the resultant netlist without the `syn_slow` attribute.

---

Verilog    output [7:0] DATA0 /\* synthesis syn\_slow = 0 \*/;

VHDL    attribute syn\_slow of data0 : signal is false;

---

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell counter (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port (array (rename DATA0 "DATA0[7:0]" ) 8) (direction OUTPUT))
        (port CLK (direction INPUT))
      )
      (port RST (direction INPUT))
    )
  )
)
```

The following table shows the resultant netlist for the same design with the syn\_slow attribute.

---

Verilog    output [7:0] DATA0 /\* synthesis syn\_slow = 1 \*/;

VHDL    attribute syn\_slow of data0 : signal is true;

---

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell counter (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port (array (rename DATA0 "DATA0[7:0]" ) 8) (direction OUTPUT)
          (property xc_slow (integer 1))
        )
      )
    )
  )
)
```

When place and route is run, a line is added for each output port bit attached to the syn\_slow attribute in the following files:

```
<projectdirectory>\rev_1\pr_1\counter.pad(257):P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMS25*|2|12
|SLOW|||UNLOCATED|NO|NONE|
<projectdirectory>\rev_1\pr_1\counter_pad.txt(258):|P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMS25*|2|1
2|SLOW|||UNLOCATED|NO|NONE|
<projectdirectory>\rev_1\pr_1\syn_slow.edf(193):(property SLOW (string ""))
```

## **syn\_srl\_mindepth**

### *Attribute*

The `syn_srl_mindepth` attribute lets you specify the threshold number of registers to pack into an SRL.

### **Description**

This attribute lets you specify the threshold number of registers to pack into an SRL (the default is 3). This attribute helps guide the tool when it encounters a chain of registers.

Like `syn_srlstyle`, `syn_srl_mindepth` offers a shift register control. When `syn_srlstyle` is set globally to `select_srl`, the software infers an SRL for two-bit shift registers. With a global `syn_srl_mindepth` attribute, you can specify a register chain of three or more registers. Only Xilinx FD registers without set, reset, and enable logic can be used with this attribute.

Note that the synthesis software keeps the last register in a chain that feeds an output port unpacked, for better quality of results (QoR). You must adjust the `syn_srl_mindepth` value to account for how the tool implements this attribute. You can override this behavior by setting the `syn_srlstyle` attribute to `noextractff_srl`. This allows the final register of the chain to be packed into the SRL.

### **syn\_srl\_mindepth Syntax**

FDC File    `define_global_attribute {syn_srl_mindepth} {value}`

[Constraints  
Editor Example](#)

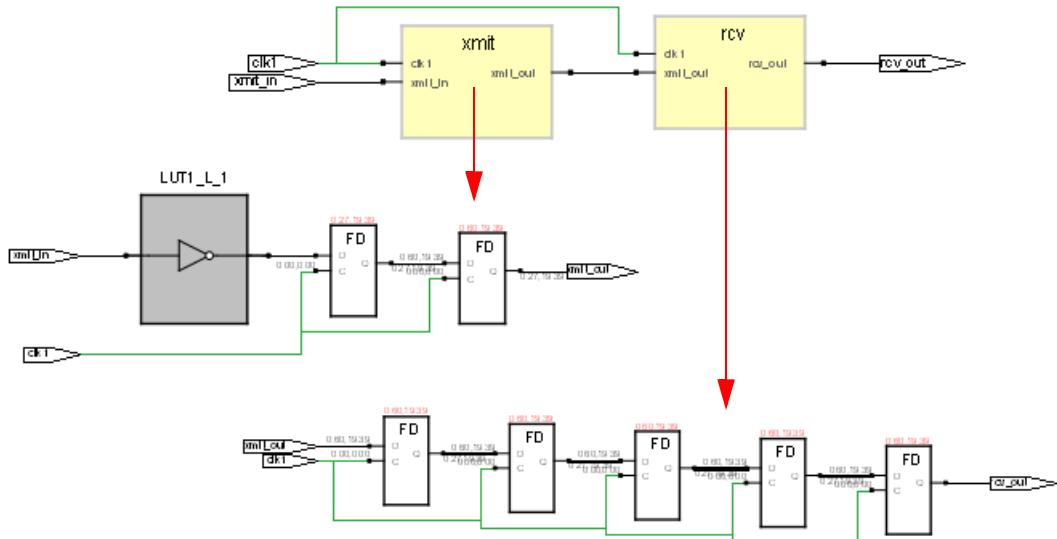
### **Constraints Editor Example**

|   | Enable                              | Object Type | Object   | Attribute                     | Value | Value Type | Description |
|---|-------------------------------------|-------------|----------|-------------------------------|-------|------------|-------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Global> | <code>syn_srl_mindepth</code> | 7     |            |             |
| 2 |                                     |             |          |                               |       |            |             |

## Effect of Using syn\_srl\_mindepth

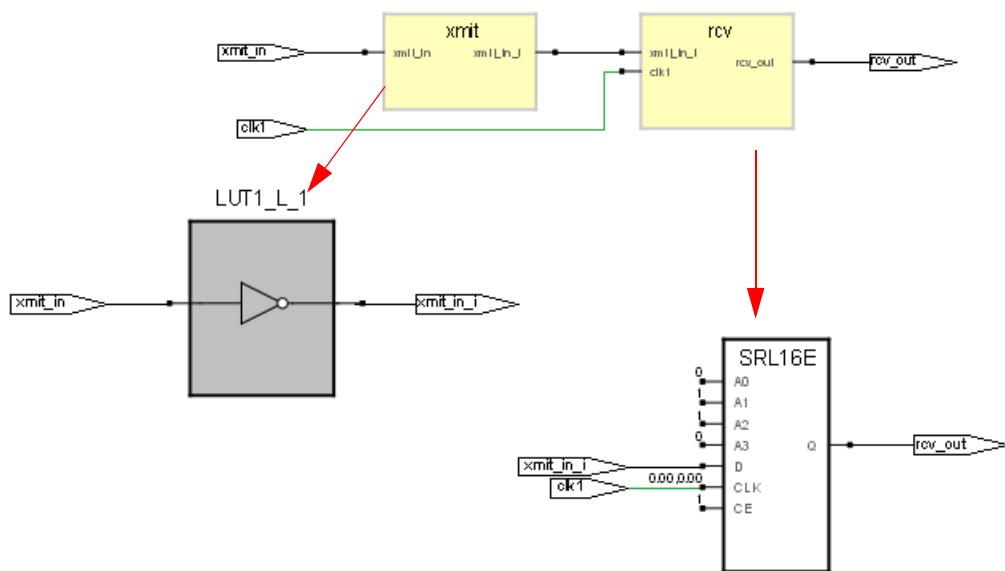
When you push down into the xmit and rcv blocks in the view for the following example, you see they have two FD registers and five FD registers respectively. All seven registers are not packed into an SRL, despite the global attribute set in the FDC file:

```
define_global_attribute {syn_srl_mindepth} {7}
```



To ensure that all seven registers are packed into the SRL16E primitive, set the attributes as shown below. The `syn_srstyle` attribute overrides the default behavior of the tool and allows the last register to be packed into the SRL:

```
define_global_attribute {syn_srl_mindepth} {7}
define_global_attribute {syn_srstyle} {noextractff_srl}
```



## syn\_srlstyle

### Attribute

Determines how to implement the sequential shift components.

### Description

The tool infers sequential shift components based on threshold limits. The `syn_srlstyle` attribute can be used to override the default behavior of `seqshift` implementation depending on how you set the values.

The `syn_srlstyle` attribute can be set globally, either on a module or a register instance. The global attribute can be overridden by the attribute set on the module or instances.

### syn\_srlstyle Syntax

|         |                                                                                                                                          |                                            |
|---------|------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute {object} syn_srlstyle {componentType}</code><br><code>define_global_attribute syn_srlstyle {componentType}</code> | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_srlstyle = "componentType" */;</code>                                                                      |                                            |
| VHDL    | <code>attribute syn_srlstyle: string;</code><br><code>attribute syn_srlstyle of object : signal is "componentType";</code>               |                                            |

In the above syntax, `componentType` is one of the values defined in the following table:

| Value           | Implements ...                                                                                                                                                 |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| registers       | Maps seqShift register components to registers.                                                                                                                |
| select_srl      | Maps seqShift components using 16-bit shift register lookup table primitives (SRL16) with flip-flops inserted ahead of the output buffers for improved timing. |
| noextractff_srl | Maps seqShift components using 16-bit shift register lookup table primitives (SRL16) without output flip-flops.                                                |

If you do not specify `syn_srlstyle`, the default behavior depends on the timing at the SRL output. If the output has very positive slack, the tool uses `syn_srlstyle=noextractff_srl`. In all other cases, the software implements `syn_srlstyle=select_srl` behavior.

## Constraints Editor Example

|   | Enable                              | Object Type | Object                | Attribute    | Value     | Value Type | Description                                     |
|---|-------------------------------------|-------------|-----------------------|--------------|-----------|------------|-------------------------------------------------|
| 1 | <input checked="" type="checkbox"/> | register    | i:special_regs.w[7:0] | syn_srlstyle | registers | string     | Determines how seq. shift comp. are implemented |
| 2 |                                     |             |                       |              |           |            |                                                 |

The following are examples of FDC constraint file entries.

```
define_global_attribute syn_srlstyle {noextractff_srl}
define_attribute {i:regBank[15:0]} syn_srlstyle {registers}
```

## HDL Example

In the HDL file, you must apply the `syn_srlstyle` attribute on the final stage of the shift register. In the following example, apply the `syn_srlstyle` attribute on register `pll_status_ck245_s`. The constraint is not honored if it is placed on other registers in the shift chain.

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

entity test is
    port (pll_status, lbdr_clk : in std_logic;
          pll_status_ck245_s: out std_logic );
attribute syn_srlstyle : string;
attribute syn_srlstyle of pll_status_ck245_s :
    signal is "registers";
end test;

architecture behave of test is
signal pll_status_ck245_r : std_logic;
signal pll_status_ck245_r1 : std_logic;
begin
resynchro_ck245_reg: process(lbdr_clk)
BEGIN
    if_clk: IF lbdr_clk'EVENT AND lbdr_clk = '1' THEN
        pll_status_ck245_r <= pll_status;
        pll_status_ck245_r1 <= pll_status_ck245_r;
        pll_status_ck245_s <= pll_status_ck245_r1;
    END IF if_clk;
END PROCESS resynchro_ck245_reg;
end behave;
```

## Verilog Example

This example implements seqShift components as SRL16 without output flip-flops.

```
module test_srl(clk, enable, dataIn, result, addr);
    input clk, enable;
    input [3:0] dataIn;
    input [3:0] addr;
    output [3:0] result;
    reg [3:0] regBank[15:0]
        /* synthesis syn_srlstyle="noextractff_srl" */;
    integer i;

    always @ (posedge clk) begin
        if (enable == 1) begin
            for (i=15; i>0; i=i-1) begin
                regBank[i] <= regBank[i-1];
            end
            regBank[0] <= dataIn;
        end
    end
    assign result = regBank[addr];
endmodule
```

## VHDL Example

The example below implements seqShift components as SRL16 primitives without inserting timing improvement flip-flops between the SRL outputs and the output buffers.

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

entity d_p is
    port (clk : in std_logic;
          data_out : out std_logic_vector(127 downto 0) );
end d_p;

architecture rtl of d_p is
type dataAryType is array(3 downto 0) of
    std_logic_vector(127 downto 0);
signal h_data_pip_i : dataAryType;
attribute syn_srlstyle : string;
attribute syn_srlstyle of h_data_pip_i :
    signal is "noextractff_srl";
```

```
begin
  process (Clk)
  begin
    if (Clk'Event And Clk = '1') then
      h_data_pip_i <= (h_data_pip_i(2 DOWNTO 0)) &
        h_data_pip_i(3);
    end if;
  end process;
  data_out <= h_data_pip_i(0);
end rtl;
```

## syn\_state\_machine

### *Directive*

Enables/disables state-machine optimization on individual state registers in the design.

### Description

Enables/disables state-machine optimization on individual state registers in the design. When you disable the FSM Compiler, state-machines are not automatically extracted. To extract some state machines, use this directive with a value of 1 on just those individual state-registers to be extracted. Conversely, when the FSM Compiler is enabled and there are state machines in your design that you do not want extracted, use syn\_state\_machine with a value of 0 to override extraction on just those individual state registers.

Also, when the FSM Compiler is enabled, all state machines are usually detected during synthesis. However, on occasion there are cases in which certain state machines are not detected. You can use this directive to declare those undetected registers as state machines.

### syn\_state\_machine Syntax

Verilog    *object* /\* synthesis syn\_state\_machine = "0|1" \*/;

[Verilog Example](#)

VHDL    attribute syn\_state\_machine of *state* : signal is "false|true";

[VHDL Example](#)

In the Verilog syntax, *object* is a state register; in the VHDL syntax, *state* is a signal that holds the value of the state machine.

### Verilog Example

This is the Verilog source code used for the example in the following figure.

```
module FSM1 (clk, in1, rst, out1);
  input clk, rst, in1;
  output [2:0] out1;

  `define s0 3'b000
  `define s1 3'b001
  `define s2 3'b010
  `define s3 3'bxxx
```

```
reg [2:0] out1;
reg [2:0] state /* synthesis syn_state_machine = 1 */;
reg [2:0] next_state;

always @ (posedge clk or posedge rst)
    if (rst) state <= `s0;
    else state <= next_state;

// Combined Next State and Output Logic
always @ (state or in1)
    case (state)
        `s0 : begin
            out1 <= 3'b000;
            if (in1) next_state <= `s1;
            else next_state <= `s0;
        end
        `s1 : begin
            out1 <= 3'b001;
            if (in1) next_state <= `s2;
            else next_state <= `s1;
        end
        `s2 : begin
            out1 <= 3'b010;
            if (in1) next_state <= `s3;
            else next_state <= `s2;
        end
        default : begin
            out1 <= 3'bxxx;
            next_state <= `s0;
        end
    endcase
endmodule
```

## VHDL Example

This is the VHDL source code used for the example in the following figure.

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

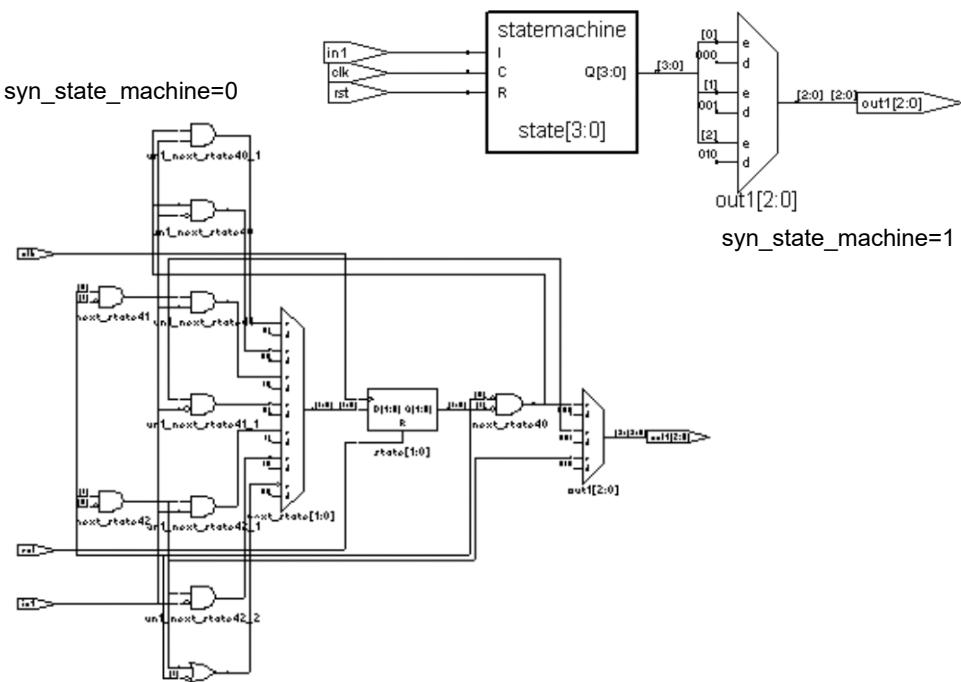
entity FSM1 is
    port (clk,rst,in1 : in std_logic;
          out1 : out std_logic_vector (2 downto 0) );
end FSM1;
```

```
architecture behave of FSM1 is
type state_values is (s0, s1, s2,s3);
signal state, next_state: state_values;
attribute syn_state_machine : boolean;
attribute syn_state_machine of state : signal is false;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            state <= s0;
        elsif rising_edge(clk) then
            state <= next_state;
        end if;
    end process;

    process (state, in1) begin
        case state is
            when s0 =>
                out1 <= "000";
                if in1 = '1' then next_state <= s1;
                    else next_state <= s0;
                end if;
            when s1 =>
                out1 <= "001";
                if in1 = '1' then next_state <= s2;
                    else next_state <= s1;
                end if;
            when s2 =>
                out1 <= "010";
                if in1 = '1' then next_state <= s3;
                    else next_state <= s2;
                end if;
            when others =>
                out1 <= "XXX"; next_state <= s0;
        end case;
    end process;
end behave;
```

## Effect of Using syn\_state\_machine

The following figure shows an example of two implementations of a state machine: one with the `syn_state_machine` directive enabled, the other with the directive disabled.



See [syn\\_encoding](#), on page 603 for information on overriding default encoding styles for state machines..

## **syn\_tco<n>**

*Directive*

Supplies the clock to output timing-delay through a black box.

### Description

Used with the `syn_black_box` directive; supplies the clock-to-output delay through a black box. The `syn_tco<n>` directive is one of several directives that you can use with the `syn_black_box` directive to define timing for a black box. See [syn\\_black\\_box, on page 526](#) for a list of the associated directives.

### **syn\_tco<n> Syntax**

---

FDC File    **define\_attribute {v:object} syn\_tcon {[!]clock->bundle=value}**

---

Verilog    **object /\* syn\_tcon = "[!]clock->bundle=value" \*/;**

---

VHDL    **attribute syn\_tcon of object : objectType is "[!]clock->bundle=value" ;**

---

For details about the syntax, see the following table:

|               |                                                                                                                                                                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>v:</b>     | Constraint file syntax that indicates the directive is attached to the view.                                                                                                                                                                                       |
| <b>object</b> | The symbol name of the black-box.                                                                                                                                                                                                                                  |
| <b>n</b>      | A numerical suffix that lets you specify different clock to output timing delays for multiple signals/bundles.                                                                                                                                                     |
| <b>!</b>      | An optional exclamation mark that indicates that the clock is active on its falling (negative) edge.                                                                                                                                                               |
| <b>clock</b>  | The name of the clock signal.                                                                                                                                                                                                                                      |
| <b>bundle</b> | A collection of buses and scalar signals. The objects of a bundle must be separated by commas with no intervening spaces. A valid bundle is A,B,C, which lists three signals. To assign values to bundles, use the following syntax:<br><br>[!]clock->bundle=value |
| <b>value</b>  | Clock-to-output delay value in ns.                                                                                                                                                                                                                                 |

The `syn_tco<n>` directive can be entered as an attribute using the Attributes panel of the Constraints Editor window. The information in the Object, Attribute, and Value fields must be manually entered. This is a constraint file example for the directive:

```
define_attribute {v:work.test} {syn_tsu4} {clk->tout=1.0}
```

## Verilog Example

The following example defines `syn_tco<n>` and other black-box constraints:

```
module test(myclk, a, b, tout,) /*synthesis syn_black_box
    syn_tco1="clk->tout=1.0" syn_tpdl="b->tout=8.0"
    syn_tsu1="a->myclk=2.0" */;
    input myclk;
    input a, b;
    output tout;
endmodule

//Top Level
module top (input clk, input a, b, output fout);
    test U1 (clk, a, b, fout);
endmodule
```

## VHDL Example

In VHDL, there are ten predefined instances of each of these directives in the `synplify` library: `syn_tco1`, `syn_tco2`, `syn_tco3`, ... `syn_tco10`. If you are entering the timing directives in the source code and you require more than 10 different timing delay values for any one of the directives, declare the additional directives with an integer greater than 10. For example:

```
attribute syn_tco11 : string;
attribute syn_tco12 : string;
```

The following example defines `syn_tco<n>` and other black-box constraints:

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

```
entity test is
generic (size: integer := 8);
    port (tout : out std_logic_vector (size- 1 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          myclk : in std_logic );
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;
end;

architecture rtl of test is
attribute syn_black_box : boolean;
attribute syn_black_box of rtl: architecture is true;
begin
end;

-- TOP Level--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity top is
generic (size: integer:= 8);
    port (fout : out std_logic_vector(size- 1 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          clk : in std_logic );
end;

architecture rtl of top is
component test
generic (size: integer := 8);
    port (tout : out std_logic_vector(size- 1 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          myclk : in std_logic );
end component;
attribute syn_tcol : string;
attribute syn_tcol of test : component is "clk->tout = 1.0";
attribute syn_tpd1 : string;
attribute syn_tpd1 of test : component is "b->tout= 2.0";
attribute syn_tsul : string;
attribute syn_tsul of test : component is "a-> myclk = 1.2";
begin
    U1 : test port map(fout, a, b, clk);
end;
```

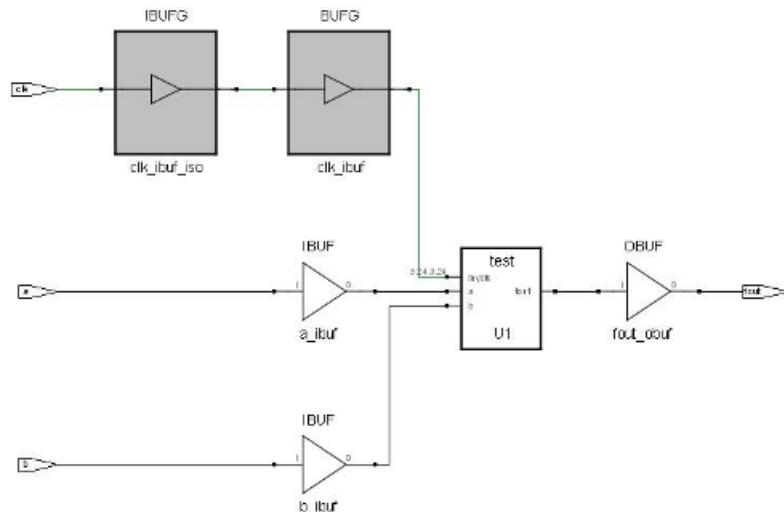
## Verilog-Style Syntax in VHDL for Black Box Timing

In addition to the syntax used in the code above, you can also use the following Verilog-style syntax to specify black-box timing constraints:

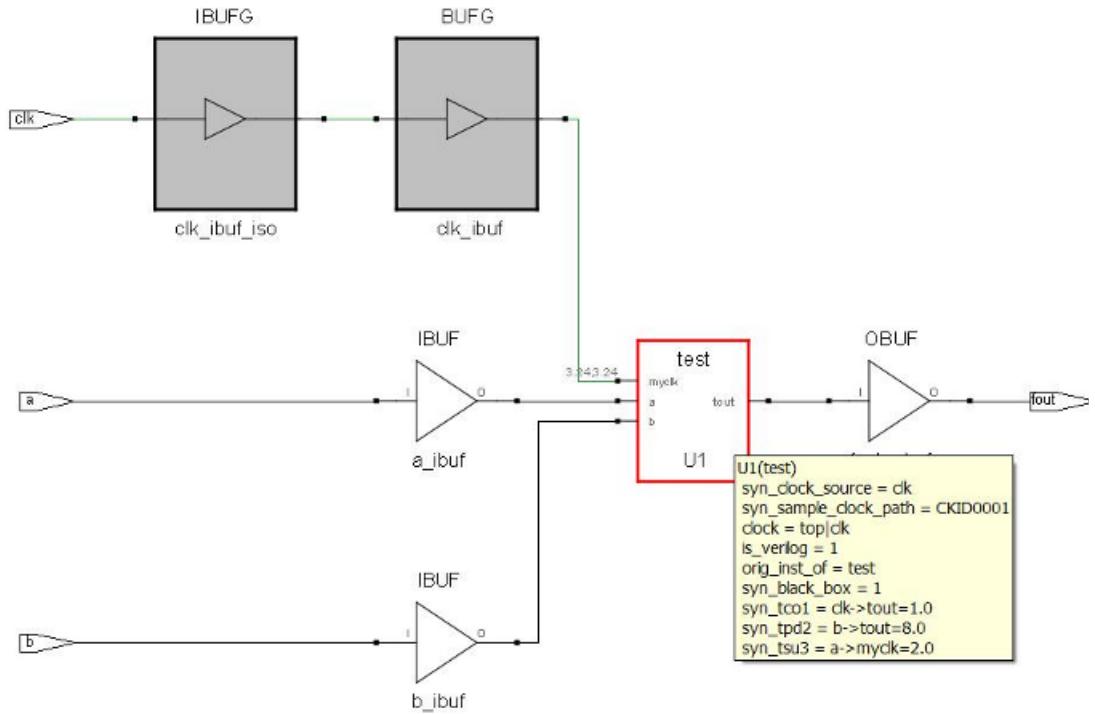
```
attribute syn_tco1 of inputfifo_coregen : component is  
  "rd_clk->dout[48:0]=3.0";
```

### Effect of using syn\_tco

This figure shows the view before using syn\_tco:



This figure shows the view after using syn\_tco:



## **syn\_tpd<n>**

### *Directive*

Supplies information on timing propagation for combinational delays through a black box.

### **Description**

Used with the syn\_black\_box directive; supplies information on timing propagation for combinational delay through a black box. The **syn\_tpd<n>** directive is one of several directives that you can use with the syn\_black\_box directive to define timing for a black box. See [syn\\_black\\_box, on page 526](#) for a list of the associated directives.

### **syn\_tco<n> Syntax**

---

FDC File   **define\_attribute {v:object} syn\_tpdn {[!]clock->bundle=value}**

---

Verilog   **object /\* syn\_tpdn = "[!]clock->bundle=value" \*/;**

---

VHDL   **attribute syn\_tpdn of object : objectType is "[!]clock->bundle=value" ;**

---

For details about the syntax, see the following table:

|               |                                                                                                                                                                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>v:</b>     | Constraint file syntax that indicates the directive is attached to the view.                                                                                                                                                                                       |
| <b>object</b> | The symbol name of the black-box.                                                                                                                                                                                                                                  |
| <b>n</b>      | A numerical suffix that lets you specify different input-to-output timing delays for multiple signals/bundles.                                                                                                                                                     |
| <b>!</b>      | An optional exclamation mark that indicates that the clock is active on its falling (negative) edge.                                                                                                                                                               |
| <b>clock</b>  | The name of the clock signal.                                                                                                                                                                                                                                      |
| <b>bundle</b> | A collection of buses and scalar signals. The objects of a bundle must be separated by commas with no intervening spaces. A valid bundle is A,B,C, which lists three signals. To assign values to bundles, use the following syntax:<br><br>[!]clock->bundle=value |
| <b>value</b>  | input-to-output delay value in ns.                                                                                                                                                                                                                                 |

The `syn_tpd<n>` directive can be entered as an attribute using the Attributes panel of the Constraints Editor window. The information in the Object, Attribute, and Value fields must be manually entered. This is a constraint file example for the directive:

```
define_attribute {v:MEM} syn_tpdl {MEM_RD->DATA_OUT[63:0]=20}
```

## Verilog Example

The following example defines `syn_tpd<n>` and other black-box constraints:

```
module test(myclk, a, b, tout,) /*synthesis syn_black_box
    syn_tco1="clk->tout=1.0" syn_tpdl="b->tout=8.0"
    syn_tsul="a->myclk=2.0" */;
    input myclk;
    input a, b;
    output tout;
endmodule

//Top Level
module top(input clk, input a, b, output fout);
    test U1 (clk, a, b, fout);
endmodule
```

## VHDL Example

In VHDL, there are 10 predefined instances of each of these directives in the synplify library, for example: syn\_tpd1, syn\_tpd2, syn\_tpd3, ... syn\_tpd10. If you are entering the timing directives in the source code and you require more than 10 different timing delay values for any one of the directives, declare the additional directives with an integer greater than 10. For example:

```
attribute syn_tpd11 : string;
attribute syn_tpd11 of bitreg : component is
    "di0,di1 -> do0,do1 = 2.0";
attribute syn_tpd12 : string;
attribute syn_tpd12 of bitreg : component is
    "di2,di3 -> do2,do3 = 1.8";
```

The following example defines syn\_tpd<n> and other black-box constraints.

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

entity test is
generic (size: integer := 8);
port (tout : out std_logic_vector(size- 1 downto 0);
      a : in std_logic_vector (size- 1 downto 0);
      b : in std_logic_vector (size- 1 downto 0);
      myclk : in std_logic );
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;
end;

architecture rtl of test is
attribute syn_black_box : boolean;
attribute syn_black_box of rtl: architecture is true;
begin
end;

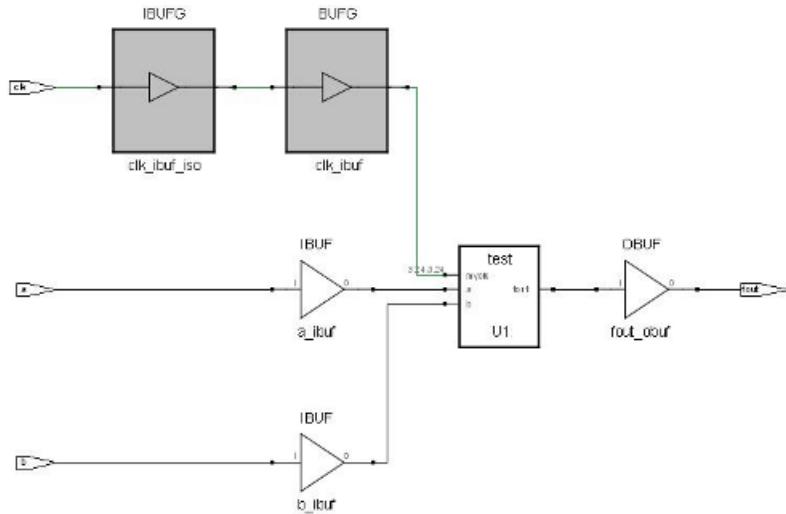
-- TOP Level--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity top is
generic (size: integer := 8);
    port (fout : out std_logic_vector(size- 1 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          clk : in std_logic );
end;

architecture rtl of top is
component test
generic (size: integer := 8);
    port (tout : out std_logic_vector(size- 1 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          myclk : in std_logic );
end component;
attribute syn_tcol : string;
attribute syn_tcol of test : component is "clk->tout = 1.0";
attribute syn_tpdl : string;
attribute syn_tpdl of test : component is "b->tout= 2.0";
attribute syn_tsul : string;
attribute syn_tsul of test : component is "a-> myclk = 1.2";
begin
    U1 : test port map(fout, a, b, clk);
end;
```

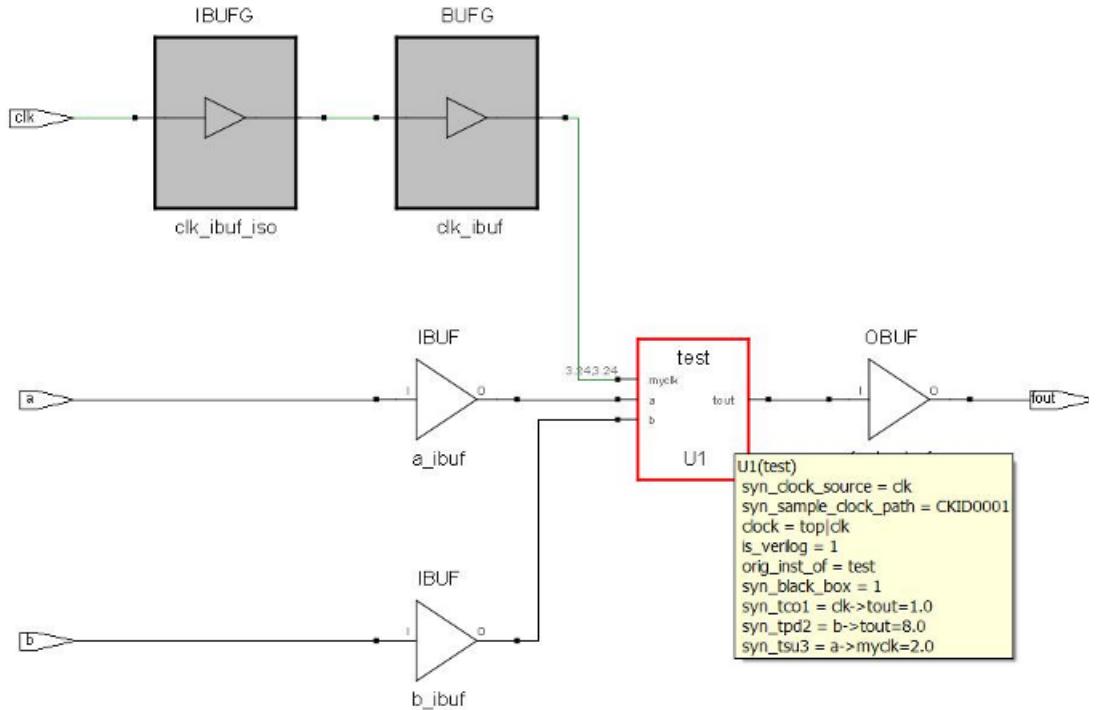
## Effect of using syn\_tpd

This figure shows the view before using syn\_tpd:



## After using syn\_tpd

This figure shows the view after using syn\_tpd:



## **syn\_trace\_attr**

Black boxes can be used in board files to indicate certain attributes on traces of boards. Use the `syn_trace_attr` in the `_guts.v` file to define attributes such as trace names, IOSTD, and `syn_keep`. Properties on these traces are honored by the tool and may also be available to Vivado.

| Parameter  | Value                                             | Description                                                                                                                                                                    |
|------------|---------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| width      | 1   0                                             | Defines the trace width.                                                                                                                                                       |
| priority   | Any integer                                       | Defines the priority of the attribute.<br>If more than one attribute is assigned to a port, then the attribute with higher priority is used to provide the name for the trace. |
| attributes | Any synthesis attribute or user defined attribute | Assigns specific attributes and properties to the trace and connected pins.                                                                                                    |
| reserved   | User defined                                      | Reserves the trace for specific purpose.                                                                                                                                       |
| first_bit  | User defined                                      | Indicates the index of the first bit.                                                                                                                                          |

### Example

In this example, attributes with names `GCLK0` and `GCLK0N` are used to set trace-names to traces connected to ports `A_GCLKP[0]` and `A_GCLKN[0]`, and to set a higher priority to these traces. All other attributes are applied on these ports as `syn_diff_io`, `syn_hstdm_clk_pin`.

```
// Attributes for clocks
// Name 100MHz / HSTDM clock
syn_trace_attr #(2, 1, "syn_diff_io=1")
CLK_100MHz({A_GCLKP[0], A_GCLKN[0]});

// Use 100MHz clock for HSTDM
syn_trace_attr #(1, -1, "syn_hstdm_clk_pin=100 syn_noprune=1")
hstdm_clk(A_GCLKP[0]);

syn_trace_attr #(1, -1, "syn_noprune=1") hstdm_clk_n(A_GCLKN[0]);
// Keep N trace

// Clock DIFF type attributes
```

```
syn_trace_attr #(2,-1,"syn_haps_differential_clock=in  
syn_haps_funcgroup=GCLK0_100MHz") gclk_0({A_GCLKP[0],A_GCLKN[0]});  
  
syn_trace_attr #(1,3,"syn_haps_differential_type=N")  
GCLK0N(A_GCLKN[0]);  
  
// Clock trace assign name attributes  
  
syn_trace_attr #(1,3,"haps_allowed_iostandards=LVDS_18")  
GCLK0(A_GCLKP[0]);
```

## syn\_tristate

### *Directive*

Specifies that an output port on a black box is a tristate.

### Description

You can use this directive to specify that an output port on a module defined as a black box is a tristate. This directive eliminates multiple driver errors if the output of a black box has more than one driver. A multiple driver error is issued unless you use this directive to specify that the outputs are tristate.

### syn\_tristate Syntax

---

Verilog    *object /\* synthesis syn\_tristate = 1 \*/;*

---

VHDL    **attribute syn\_tristate : boolean;**  
         **attribute syn\_tristate of *object* : *objectType* is true;**

---

### Verilog Example

```
module test(myclk, a, b, tout) /* synthesis syn_black_box */;
  input myclk;
  input a, b;
  output tout/* synthesis syn_tristate = 1 */;
endmodule

//Top Level
module top(input [1:0]en, input clk, input a, b, output reg fout);
  wire tmp;
  assign tmp = en[0] ? (a & b) : 1'bz;
  assign tmp = en[1] ? (a | b) : 1'bz;

  always@(posedge clk)
  begin
    fout <= tmp;
  end
  test U1 (clk, a, b, tmp);
endmodule
```

## VHDL Example

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

entity test is
    port (tout : out std_logic;
          a : in std_logic;
          b : in std_logic;
          myclk : in std_logic );
attribute syn_tristate : boolean;
attribute syn_tristate of tout: signal is true;
end;

architecture rtl of test is
attribute syn_black_box : boolean;
attribute syn_black_box of rtl: architecture is true;
begin
end;

-- TOP Level--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity top is
    port (fout : out std_logic;
          a : in std_logic;
          b : in std_logic;
          en: in std_logic_vector(1 downto 0);
          clk : in std_logic );
end;

architecture rtl of top is
signal tmp : std_logic;
component test
    port (tout : out std_logic;
          a : in std_logic;
          b : in std_logic;
          myclk : in std_logic );
end component;
begin
tmp <= (a and b)when en(0) = '1' else 'Z';
tmp <= (a or b) when en(1) = '1' else 'Z';
process (clk)
begin
```

```
if (clk = '1' and clk'event ) then
    fout <= tmp;
end if;
end process;
U1 : test port map(fout, a, b, clk);
```

## **syn\_tsu< n >**

*Directive*

Sets information on timing setup delay required for input pins in a black box.

### Description

Used with the `syn_black_box` directive; supplies information on timing setup delay required for input pins (relative to the clock) in the black box. The `syn_tsu< n >` directive is one of several directives that you can use with the `syn_black_box` directive to define timing for a black box. See [syn\\_black\\_box, on page 526](#) for a list of the associated directives.

### **syn\_tsu< n > Syntax**

---

FDC File    **define\_attribute {v:object} syn\_tsun {bundle->[!]clock=value}**

---

Verilog      **object /\* syn\_tsun = "bundle->[!]clock=value" \*/;**

---

VHDL        **attribute syn\_tsun of object : objectType is "bundle->[!]clock=value";**

---

For details about the syntax, see the following table:

---

**v:** Constraint file syntax that indicates the directive is attached to the view.

---

**object** The symbol name of the black-box.

---

**n** A numerical suffix that lets you specify different input-to-output timing delays for multiple signals/bundles.

---

**!** An optional exclamation mark that indicates that the clock is active on its falling (negative) edge.

---

**clock** The name of the clock signal.

---

**bundle** A collection of buses and scalar signals. The objects of a bundle must be separated by commas with no intervening spaces. A valid bundle is A,B,C, which lists three signals. To assign values to bundles, use the following syntax:

---

**[!]clock->bundle=value**

---

**value** setup delay value in ns.

---

The `syn_tsu<n>` directive can be entered as an attribute using the Attributes panel of the Constraints Editor window. The information in the Object, Attribute, and Value fields must be manually entered. This is a constraint file example for the directive:

```
define_attribute {v:RTRV_MOD} syn_tsu4 {RTRV_DATA[63:0]->!CLK=20}
```

## Verilog Example

This is an example that defines `syn_tsu<n>` with some of the other black-box constraints:

```
module test(myclk, a, b, tout,) /*synthesis syn_black_box
    syn_tcol="clk->tout=1.0" syn_tpdl="b->tout=8.0"
    syn_tsu1="a->myclk=2.0" */;
    input myclk;
    input a, b;
    output tout;
endmodule

//Top Level
module top (input clk, input a, b, output fout);
    test U1 (clk, a, b, fout);
endmodule
```

## VHDL Examples

In VHDL, there are 10 predefined instances of each of these directives in the synplify library, for example: `syn_tsu1`, `syn_tsu2`, `syn_tsu3`, ... `syn_tsu10`. If you are entering the timing directives in the source code and you require more than 10 different timing delay values for any one of the directives, declare the additional directives with an integer greater than 10:

```
attribute syn_tsu11 : string;
attribute syn_tsu11 of bitreg : component is
    "di0,di1 -> clk = 2.0";
attribute syn_tsu12 : string;
attribute syn_tsu12 of bitreg : component is
    "di2,di3 -> clk = 1.8";
```

The following is an example of assigning `syn_tsu<n>` along with some of the other black-box constraints:

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

```
entity test is
generic (size: integer := 8);
    port (tout : out std_logic_vector(size- 1 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          myclk : in std_logic );
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;
end;

architecture rtl of test is
attribute syn_black_box : boolean;
attribute syn_black_box of rtl: architecture is true;
begin
end;

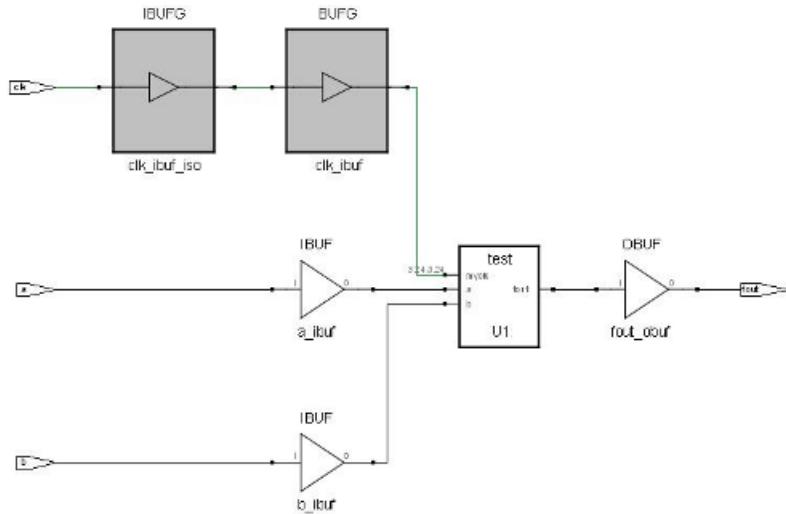
-- TOP Level--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity top is
generic (size: integer := 8);
    port (fout : out std_logic_vector (size- 1 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          clk : in std_logic );
end;

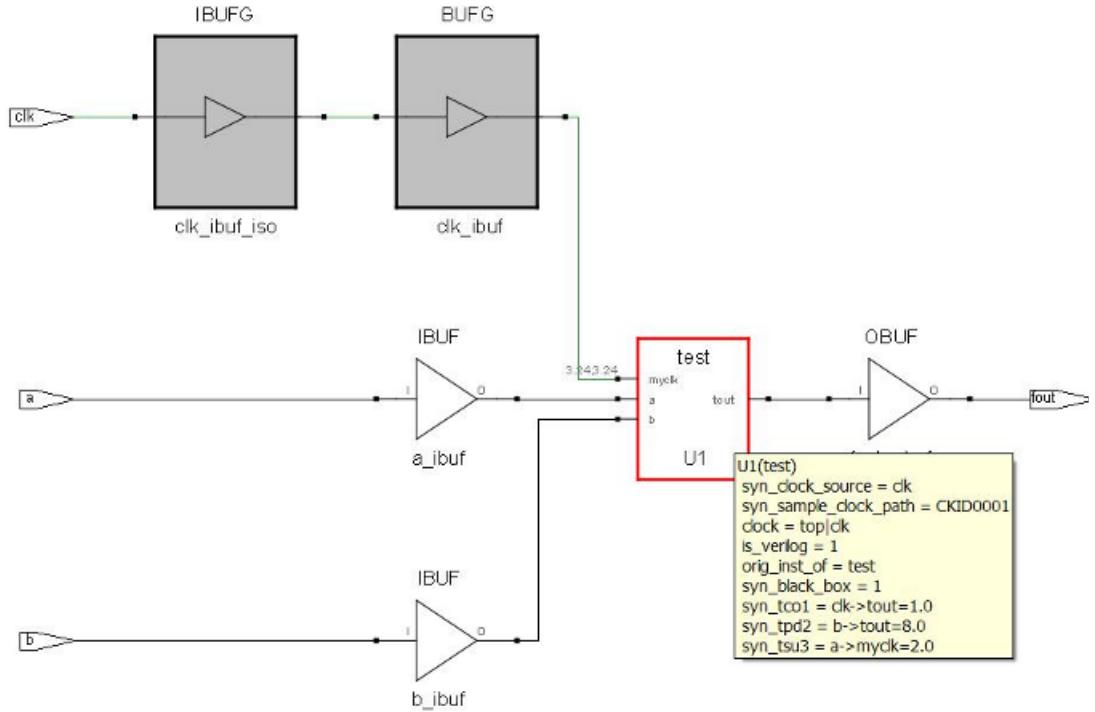
architecture rtl of top is
component test
generic (size: integer := 8);
    port (tout : out std_logic_vector(size- 1 downto 0);
          a : in std_logic_vector (size- 1 downto 0);
          b : in std_logic_vector (size- 1 downto 0);
          myclk : in std_logic );
end component;
attribute syn_tcol : string;
attribute syn_tcol of test : component is "clk->tout = 1.0";
attribute syn_tpd1 : string;
attribute syn_tpd1 of test : component is "b->tout= 2.0";
attribute syn_tsul : string;
attribute syn_tsul of test : component is "a-> myclk = 1.2";
begin
    U1 : test port map (fout, a, b, clk);
end;
```

## Effect of using syn\_tsu

This figure shows the view before using syn\_tsu:



This figure shows the view after using syn\_tsu:



## **syn\_unconnected\_inputs**

### *Attribute*

Specifies that input pins of an instance are to be left floating (unassigned in the RTL).

### **Description**

When an instance has a floating input pin, the synthesis software does not tie the pin to GND in the output netlist. However when the input pin is driven by a wire that subsequently is unassigned, then the synthesis tool ties the pin to GND. Connecting floating input pins to GND causes routing problems in the place-and-route tool. Use the `syn_unconnected_inputs` attribute to ensure that input pins with undriven wires are left floating. This attribute can only be specified in the constraint file.

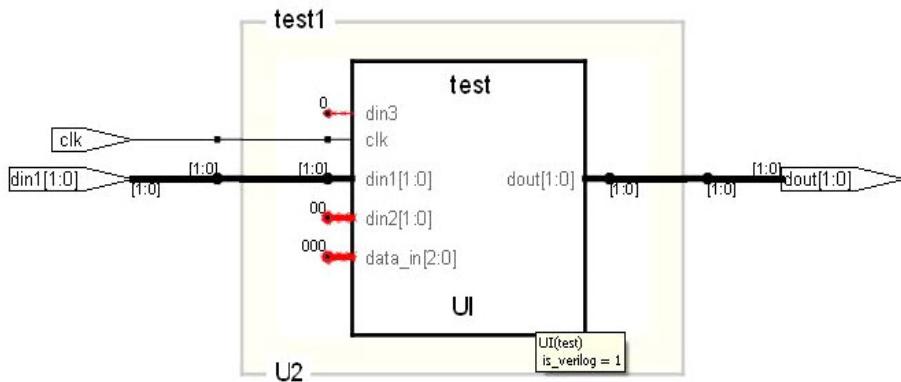
### **syn\_unconnected\_inputs Syntax**

Specify the syntax for this attribute in the constraint file as follows:

```
define_attribute {instanceName} syn_unconnected_inputs {string}
```

### **Verilog Example**

The following example shows an instance which you can drag-and-drop from the view into the Constraints user interface.



The following examples of Verilog source code are defined for this attribute.

```

module test (din1, din2, din3, data_in, dout, clk)
    /*synthesis syn_black_box*/;
    input [1:0] din1;
    input [1:0] din2;
    input [2:0] data_in;
    input din3;
    input clk;output [1:0] dout;
endmodule

module test1 (din1, din2, din3, data_in, dout, clk);
    input [1:0] din1;
    input [1:0] din2;
    input din3;
    input [2:0] data_in;
    input clk;
    output [1:0] dout;

    //Following wires are left floating
    wire [1:0] d2_wire;
    wire [2:0] data_wire;
    wire d3_wire;

```

```

test UI (
    .din1(din1),
    .din2(d2_wire),
    .din3(d3_wire),
    .data_in(data_wire),
    .dout(dout),
    .clk(clk) );
endmodule

module top (din1, din2, din3, data_in, dout, clk);
    input [1:0] din1;
    input [1:0] din2;
    input [2:0] data_in;
    input din3;
    input clk;
    output [1:0] dout;
endmodule

test1 U2 (
    .din1(din1),
    .din2(din2),
    .din3(din3),
    .data_in(data_in),
    .dout(dout),
    .clk(clk) );
endmodule

```

## VHDL Syntax and Example

```
define_attribute {i:UI} syn_unconnected_inputs
{din2, din3, data_in}
```

The following examples of VHDL source code are defined for this attribute.

```

Library ieee;
use ieee.std_logic_1164.all;

entity top is
    port (din1: in std_logic_vector(1 downto 0);
          din2: in std_logic_vector(1 downto 0);
          data_in: in std_logic_vector(2 downto 0);
          din3: in std_logic;
          clk: in std_logic;
          dout: out std_logic_vector(1 downto 0)) ;
end top;

```

```
architecture top_a of top is
component test is
    port (din1: in std_logic_vector(1 downto 0);
          din2: in std_logic_vector(1 downto 0);
          data_in: in std_logic_vector(2 downto 0);
          din3: in std_logic;
          clk: in std_logic;
          dout: out std_logic_vector(1 downto 0) );
end component;

-----
-- Wires left floating
-----

signal d2_wire: std_logic_vector(1 downto 0);
signal d3_wire: std_logic;
signal data_wire: std_logic_vector(2 downto 0);

begin
UI: test
    port map (din1 => din1,
              din2 => d2_wire,
              data_in => data_wire,
              din3 => d3_wire,
              clk => clk,
              dout => dout);
end top_a;
```

## **syn\_unique\_inst\_module**

### *Directive*

Renames the module for a particular instance, leaving other instances and the original module unchanged.

### **Description**

The `syn_unique_inst_module` directive, which is applied through a compiler directives (CDC) file, renames the module for a particular instance without affecting the other instances of the original module or the original module.

Use this directive in a hierarchical design flow where subprojects are created for instance-based export. This flow requires that the instance uniquely link to a particular module. You can achieve this by only renaming the module for the instance, leaving the other instances of the module as is, pointing to the original module.

### **syn\_unique\_inst\_module Syntax**

CDC File    `define_directive {instanceName} syn_unique_inst_module {newModuleName}`

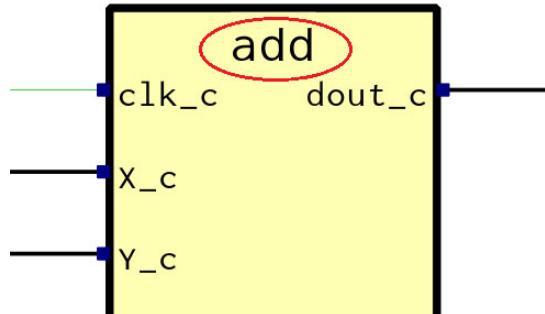
---

### **CDC Example**

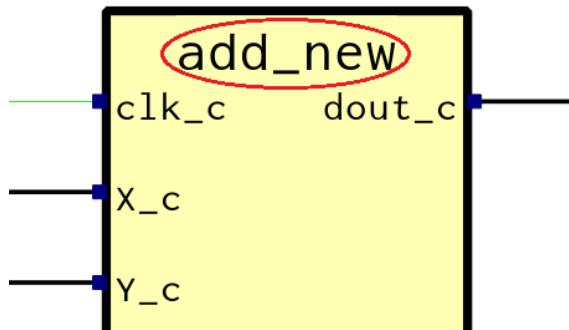
```
define_directive {add_instance1} syn_unique_inst_module {add_new}
```

### **Effect of Using syn\_unique\_inst\_module**

The following figure shows a design before `syn_unique_inst_module` is applied.



This figure shows the results after applying `syn_unique_inst_module`:



## **syn\_upf\_ret\_port\_type**

### *Directive*

Identifies the port type for the sequential element.

### Syntax

**syn\_upf\_ret\_port\_type = "portType [portTypeDescriptor]"**

*portType* is the port for the sequential element as described in the following table. Some port types require an additional descriptor (*portTypeDescriptor*).

Note that for the custom or user-defined retention model or for the dual-signal retention model, define ports for both save and restore. For the single-signal retention model, define either the save or the restore.

| Argument 1 | Argument 2   | Description                       |
|------------|--------------|-----------------------------------|
| CLK        |              | Clock                             |
| RESET      | SYNC   ASYNC | Synchronous or Asynchronous Reset |
| SET        | SYNC   ASYNC | Synchronous or Asynchronous Set   |
| EN         |              | Enable                            |
| SAVE       |              | Retention Save Control            |
| RESTORE    |              | Retention Restore Control         |
| D          |              | Input Data                        |
| Q          |              | Output Data                       |

### Example

```
input clk /* synthesis syn_upf_ret_port_type = "CLK" */;
input reset /* synthesis syn_upf_ret_port_type =
    "RESET, ASYNC" */;
input SAVE /* synthesis syn_upf_ret_port_type = "SAVE" */;
input RESTORE /* synthesis syn_upf_ret_port_type = "RESTORE" */;
```

## **syn\_useenables**

### *Attribute*

Controls the use of clock-enable registers within a design.

### **Description**

By default, the synthesis tool uses registers with clock enable pins where applicable. Setting the `syn_useenables` attribute to 0 on a register creates external clock-enable logic to allow the tool to infer a register that does not require a clock-enable.

By eliminating the need for a clock-enable, designs can be mapped into less complex registers that can be more easily packed into RAMs or DSPs. The trade-off is that while conserving complex registers, the additional external clock-enable logic can increase the overall logic-unit count.

### **Syntax Specification**

|          |                                                                               |                                            |
|----------|-------------------------------------------------------------------------------|--------------------------------------------|
| FDC File | <code>define attribute {register signal} syn_useenables {0 1}</code>          | <a href="#">Constraints Editor Example</a> |
| Verilog  | <code>object /* synthesis syn_useenables = "0 1" */;</code>                   | <a href="#">Verilog Example</a>            |
| VHDL     | <code>attribute syn_useenables of object : objectType is "true false";</code> | <a href="#">VHDL Example</a>               |

### **Constraints Editor Example**

| Enable                              | Object Type | Object   | Attribute      | Value | Value Type | Description                    |
|-------------------------------------|-------------|----------|----------------|-------|------------|--------------------------------|
| <input checked="" type="checkbox"/> | register    | i:q[1:0] | syn_useenables | 0     | boolean    | Generate with clock enable pin |

### **Verilog Example**

```
module useenables(d,clk,q,en);
  input [1:0] d;
  input en,clk;
  output [1:0] q;
  reg [1:0] q /* synthesis syn_useenables = 0 */;
```

```
always @ (posedge clk)
  if (en)
    q<=d;
endmodule
```

## VHDL Example

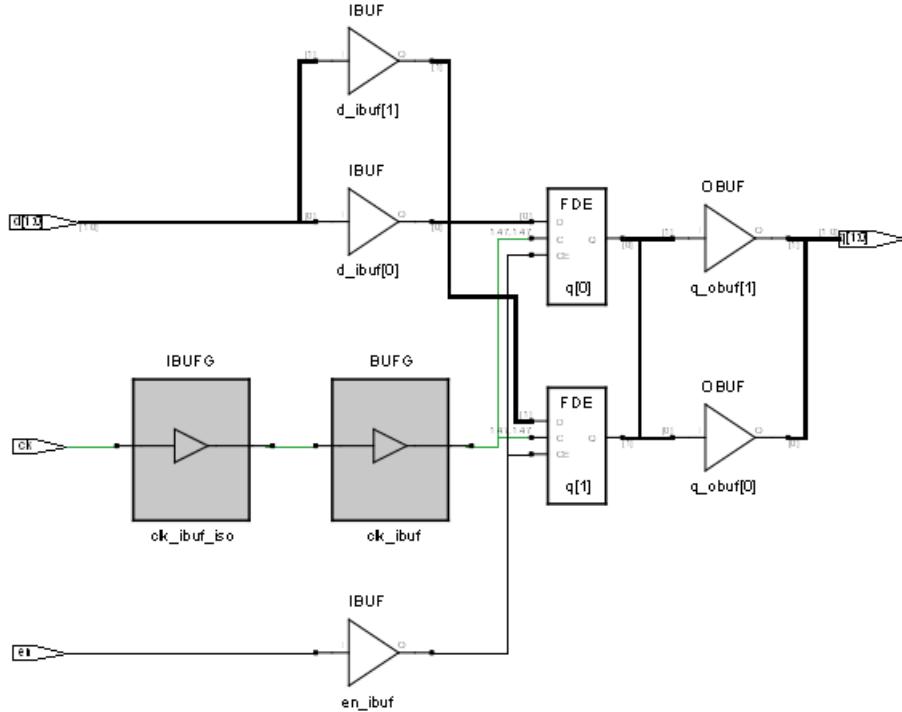
```
library ieee;
use ieee.std_logic_1164.all;

entity syn_useenables is
  port (d : in std_logic_vector(1 downto 0);
        en,clk : in std_logic;
        q : out std_logic_vector(1 downto 0) );
attribute syn_useenables: boolean;
attribute syn_useenables of q: signal is false;
end;

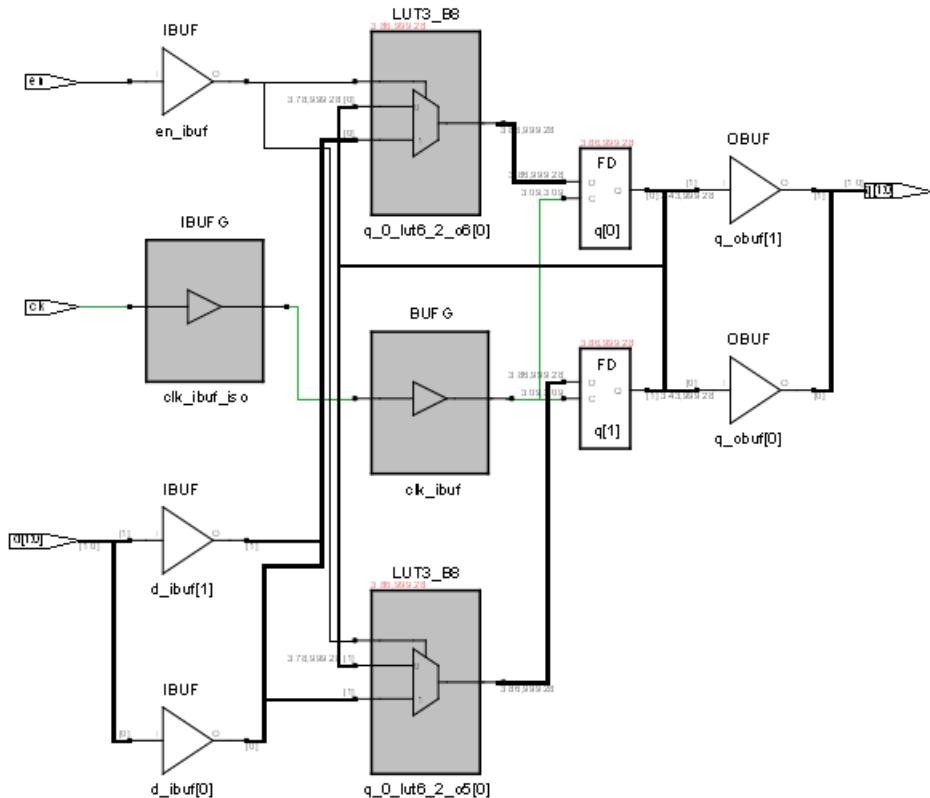
architecture syn_ue of syn_useenables is
begin
  process (clk) begin
    if (clk = '1' and clk'event) then
      if (en='1') then
        q <= d;
      end if;
    end if;
  end process;
end architecture;
```

## Effect of Using syn\_useenables

Without applying the attribute (default is to use registers with clock-enable pins) or setting the attribute to 1/true uses registers with clock-enable pins (FDEs in the below schematic).



Applying the attribute with a value of 0/false uses registers without clock-enable pins (FDs in the below schematic) and creates external clock-enable logic.



## syn\_useioff

### *Attribute*

Determines the packing of flip-flops in I/O pad cells.

### Description

By default, the software attempts to pack registers into I/O pad cells based on timing requirements. Setting the syn\_useioff attribute to 0/false overrides the default behavior and prevents register packing. The attribute can be applied to individual registers or ports and can also be applied globally. When applied at the register level, the register is excluded from I/O pad cell packing, and when applied at the port level, all registers attached to the port are excluded.

The syn\_useioff attribute is supported in the compile point flow.

### syn\_useioff Syntax

See [Object Considerations, on page 852 Examples of syn\\_useioff on Different Ports, on page 854](#) for descriptions of the ports where you can set the attribute.

|         |                                                                                                                                |                                            |
|---------|--------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute {object} syn_useioff {1 0 force}</code><br><code>define_global_attribute syn_useioff {1 0 force}</code> | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis syn_useioff = "1 0 force" */;</code>                                                                 | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute syn_useioff of object :</code><br><code>objectType is "true false force";</code>                               | <a href="#">VHDL Example</a>               |

In the above syntax, the values are defined as follows:

| Value   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1/true  | Packs the registers into I/O pad cells.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 0/false | Do not pack the registers into I/O pad cells.                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| force   | Packs the registers into I/O pad cells. Xilinx place and route requires that registers be packed into I/O pad cells with this option. The results of packing registers are not different if you use force or true. However, when a register is ineligible to be packed into an IOB, specifying IOB=force generates an error message in the Vivado tool. If IOB=true is used, a warning message is generated instead. For more information about how IOB constraints control register packing, see the Xilinx documentation. |

## Object Considerations

The `syn_useioff` attribute can be set on the following objects:

- Top-level ports (see [Example 1: Top-Level Port](#)). If `syn_useioff` is applied on a top-level port, the registers can be included in a lower-level module.
- Registers that drive top-level ports (see [Example 2: Register Driving Top-Level Port](#)). If `syn_useioff` is applied on registers that drive top-level ports, the registers can be included in a lower-level module.
- Lower-level ports, if the register is specified as part of the port declaration (see [Example 3: Lower-Level Port](#)). The tool does not apply the attribute if the register driving the port is declared independently.
  - If `syn_useioff` is applied on a lower-level port for which a register is defined as part of the port definition, the port should be driven within a clocked block.
  - If `syn_useioff` is applied on a lower-level port for which a register is defined independently and is not driven within a clocked block, this attribute will not be applied.
- When specified, IOB properties are added to the output register in the HapsTrak II view to reflect the state of the `syn_useioff` attribute.

## Register Packing Precedence

When using the `syn_useioff` attribute to control register packing, the order of precedence is registers, followed by ports, and then global as outlined below:

### 1. Registers (highest priority)

|                                    |                                                                                         |
|------------------------------------|-----------------------------------------------------------------------------------------|
| <code>syn_useioff = 1/true</code>  | Packs register into the I/O pad cell regardless of the port or global specification     |
| <code>syn_useioff = 0/false</code> | Does not pack register into I/O pad cell regardless of the port or global specification |
| <code>syn_useioff = force</code>   | Packs register into the I/O pad cell regardless of the port or global specification     |

### 2. Ports

|                                    |                                                                              |
|------------------------------------|------------------------------------------------------------------------------|
| <code>syn_useioff = 1/true</code>  | Packs registers into the I/O pad cell regardless of global specification     |
| <code>syn_useioff = 0/false</code> | Does not pack registers into I/O pad cell regardless of global specification |
| <code>syn_useioff = force</code>   | Packs registers into the I/O pad cell regardless of global specification     |

### 3. Global (lowest priority)

|                                    |                                            |
|------------------------------------|--------------------------------------------|
| <code>syn_useioff = 1/true</code>  | Packs registers into the I/O pad cells     |
| <code>syn_useioff = 0/false</code> | Does not pack registers into I/O pad cells |
| <code>syn_useioff = force</code>   | Packs registers into the I/O pad cells     |

## Constraints Editor Example

|   | Enabled                             | Object Type | Object | Attribute                | Value | Val Type | Description                     |
|---|-------------------------------------|-------------|--------|--------------------------|-------|----------|---------------------------------|
| 1 | <input checked="" type="checkbox"/> | port        | p:q    | <code>syn_useioff</code> | 1     | boolean  | Embed flip-flops in the IO ring |

## Verilog Example

```
module test_top (input clk, input d, output q );
test U (.clk(clk), .d(d), .q(q) );
endmodule

module test (input clk, input d, output q );
reg temp;
reg qreg/*synthesis syn_useioff = 0*/;
assign q = qreg;

always@(posedge clk)
begin
    temp <= d;
    qreg <= temp;
end
endmodule
```

## VHDL Example

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

entity dff is
    port (data_in: in std_logic;
          clock: in std_logic;
          data_out: out std_logic );
attribute syn_useioff : boolean;
attribute syn_useioff of data_out: signal is false;
end dff;

architecture behv of dff is
begin
    process(data_in, clock)
    begin
        if (clock='1' and clock'event) then
            data_out <= data_in;
        end if;
    end process;
end behv;
```

## Examples of syn\_useioff on Different Ports

The following examples show syn\_useioff set on ports at different levels in the design.

## Example 1: Top-Level Port

The `syn_useioff` attribute is applied on a top-level port.

```
module good2_top (input clk, input d, output q
                  /* synthesis syn_useioff=1 */ );
  good_2 U (.clk(clk), .d(d), .q(q) );
endmodule

module good_2 (input clk, input d, output q);
  reg temp;
  reg qreg;
  assign q = qreg;

  always@ (posedge clk)
    begin
      temp <= d;
      qreg <= temp;
    end
endmodule
```

## Example 2: Register Driving Top-Level Port

The `syn_useioff` attribute is applied on a register driving the top-level port.

```
module good1_top (input clk, input d, output q);
  good_1 U (.clk(clk), .d(d), .q(q) );
endmodule

module good_1 (input clk, input d, output q);
  reg temp;
  reg qreg/* synthesis syn_useioff=1 */;
  assign q = qreg;

  always@ (posedge clk)
    begin
      temp <= d;
      qreg <= temp;
    end
endmodule
```

## Example 3: Lower-Level Port

This attribute is applied on a lower-level port, for which the register is defined as part of the port declaration. However, the attribute is not applied if the register driving the port is declared independently.

```

module good_top (input clk, input d, output q);
good U (.clk(clk), .d(d), .q(q) );
endmodule

module good (input clk, input d, output reg q
    /* synthesis syn_useioff=1 */;
reg temp;
//reg qreg;
//assign q = qreg;

always@ (posedge clk)
begin
    temp <= d;
    q <= temp;
end
endmodule

```

## Effect of using syn\_useioff

Setting the `syn_useioff` attribute to 0/false prevents the software from packing registers into I/O pad cells. When you set the attribute on a top-level register or port, the synthesis software writes out the `syn_useioff` attribute to the output netlist file as an `IOB=FALSE/TRUE` statement.

Example of a netlist when `syn_useioff` is set to 0/false:

```

(library work
  (edifLevel 0)
  (technology (numberDefinition))
  (cell good_1 (cellType GENERIC)
    (view netlist (viewType NETLIST)
      (interface
        (port d_c (direction INPUT))
        (port c̄lk_c (direction INPUT))
        (port q_c (direction OUTPUT))
      )
      (contents
        (instance qreg (viewRef PRIM (cellRef FD (libraryRef UNILIB)))
          (property IOB (string "FALSE")))
      )
    )
  )
)
```

Setting the `syn_useioff` attribute to force ensures that the software attempts to pack registers into an IOB. When you set the attribute on a top-level register or port, the synthesis software writes out the `syn_useioff` attribute to the output netlist file as an `IOB=FORCE` statement to be used by the place-and-route tool.

Example of an EDIF netlist when `syn_useioff` is set to force:

```
(library work
  (edifLevel 0)
  (technology (numberDefinition))
  (cell VLOG_INR_OUTR_FD_1 (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port io (direction INOUT))
        (port clk (direction INPUT))
        (port tst (direction INPUT)))
      )
      (contents
        (instance clk_ibuf (viewRef PRIM (cellRef BUFG
          (libraryRef VIRTEX)))
        )
        (instance clk_ibuf_iso (viewRef PRIM (cellRef IBUFG
          (libraryRef VIRTEX)))
        )
        (instance io_iobuf_RNO (viewRef PRIM (cellRef INV
          (libraryRef UNILIB)))
        )
        (instance inff (viewRef PRIM (cellRef FD_1 (libraryRef UNILIB)))
          (property IOB (string "FORCE"))
        )
      )
    )
  )
)
```

Also, the `IOB=FORCE` property is forward annotated in the XDC constraint file, specifically when a VM netlist is generated. For example:

```
1 # 1000 : define_clock [get_ports {clk}] -name {VHDL_INR_OUTR_FDCE|clk} -ref_rise
2
3 create_clock -name {VHDL_INR_OUTR_FDCE|clk} [get_ports {clk}] -period {1.000}
4
5
6 set_property IOB force [get_cells {inff_Z}]
7 set_property IOB force [get_cells {outff_Z}]
8
9 #Constraints which are not forward annotated in XDC and intentionally commented out
10
```

## **syn\_useoddr**

### *Attribute*

Allows the inference of an ODDRE1/ODDR primitive on specific output ports.

| Vendor | Technologies                  |
|--------|-------------------------------|
| Xilinx | UltraScale and older devices. |

### **Description**

The `syn_useoddr` attribute allows the inference of an ODDRE1/ODDR primitive on specific output ports. You can direct the mapper to enable or disable the inference of the ODDRE1/ODDR primitive.

This attribute should be applied only to the output ports. For HAPS-70 systems, the ODDR primitive is inferred and for HAPS-80/HAPS-100 systems, the ODDRE1 primitive is inferred. If the output port is already driven by an ODDRE1/ODDR primitive, the `syn_useoddr` attribute will not infer another ODDRE1/ODDR primitive.

When the ODDRE1/ODDR primitive is inferred, the following message appears in the `map.srr` log file. Here the attribute `syn_useoddr` is applied on the output port, `clkout[0]`.

```
@N: FX126 |ODDR(E1) primitive inserted at port clkout[0] as
specified by user
```

### **syn\_useoddr Syntax**

You can specify the attribute in the following files:

---

FDC      `define_attribute {p:clkout[0]} {syn_useoddr} {1}`

---

Verilog    `object /* synthesis syn_useoddr = 0 | 1 */`

---

VHDL      `attribute syn_useoddr of object : objectType is false | true;`

## FDC Example

```
create_clock -name {clk} -period 10 [get_ports {clk}]  
define_attribute {p:clkout[0]} {syn_useoddr} {1}
```

## Verilog Example

```
module test  
#(  
    parameter integer DW = 1,  
    parameter integer CLKOUT_W = 1,  
    parameter integer CLKEN_W = 1  
) ()  
    input [DW-1:0] d,  
    input clk,  
    input [CLKEN_W-1:0] clk_en,  
    input rst,  
    input set,  
    output [CLKOUT_W-1:0] clkout /* synthesis syn_useoddr=1 */, // Attribute applied  
to "clkout"  
    output reg [DW-1:0] q  
);  
wire gclk ;  
icgp icg_u1 (gclk, clk, clk_en);  
  
assign clkout = gclk ;  
  
// DFF with set and rst  
always@(posedge gclk)  
begin  
    if(rst)  
        q <= {DW{1'b0}};  
    else if (set)  
        q <= {DW{1'b1}};  
    else  
        q <= d ;  
end  
endmodule
```

## VHDL Example

```
entity test is
  generic(
    DW : integer := 1;
    CLKOUT_W : integer := 1;
    CLKEN_W : integer := 1
  );
  port(
    d :in std_logic_vector(DW-1 downto 0);
    clk :in std_logic;
    clk_en :in std_logic_vector(CLEN_W-1 downto 0);
    rst: in std_logic;
    set: in std_logic;
    clkout :out std_logic_vector(CLKOUT_W-1 downto 0);
    q : out std_logic_vector(DW-1 downto 0)
  );
-- Attribute applied to "clkout"

attribute syn_useoddr : boolean;
attribute syn_useoddr of clkout : signal is true;
end test;

architecture Behavioral of test is
signal gclk :std_logic_vector(CLKOUT_W-1 downto 0);

component icgp port(
  c : in std_logic;
  en : in std_logic;
  gclk : out std_logic
);
end component;
begin
icg_u1 : icgp port map(
  c => clk, en => clk_en(0), gclk => gclk(0)
);
clkout <= gclk;
process(gclk)
begin
  if(rst='1') then
    q <= b"0";
  elsif(set='1') then
    q <= b"1";
```

```

elsif(rising_edge(gclk(0))) then
    q <= d;
end if;
end process;
end Behavioral;

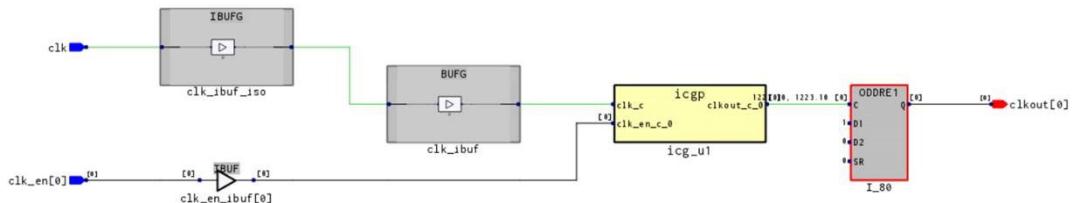
```

## Effect of using syn\_useoddr

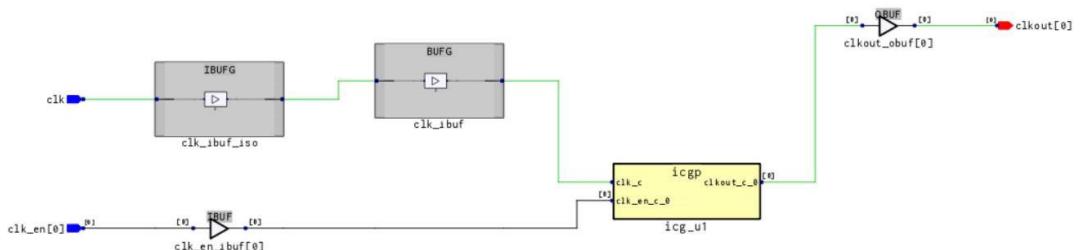
Pre-map design view:



The following figure shows when the attribute **syn\_useoddr** is set to 1. The tool infers the ODDRE1/ODDR primitive on the output port.



The following figure shows when the attribute **syn\_useoddr** is set to 0. The ODDRE1/ODDR primitive is not inferred.





## **syn\_user\_instance**

### *Attribute*

Prevents optimizations on user-instantiated primitives, but does not apply to inferred logic.

### **Description**

The `syn_user_instance` attribute determines whether a user-instantiated primitive, such as LUTs, flip-flops, or block RAM is optimized. It does not affect inferred logic. When this attribute is set to prevent optimizations on user-instantiated primitives, the logic around the primitive can still be optimized. For example, if the design has an instantiated input buffer primitive that drives an instantiated flip-flop primitive, the tool might insert a clock buffer between the two.

The `syn_user_instance` attribute is similar to the `syn_macro` attribute, but there are differences in how the objects are handled. In addition, you can set `syn_user_instance` on an instance, but you must set `syn_macro` on a view or module. You can use the `syn_user_instance` attribute to selectively disable optimizations for instantiated user-instantiated primitives when the global Enhanced Optimization switch is enabled.

If both attributes are applied to the same object, `syn_user_instance` takes precedence. See [syn\\_macro](#), [syn\\_user\\_instance](#), [Enhanced Optimization and Cores](#), on page 680 for further details about how the attributes affect optimization when used together.

### **syn\_user\_instance Syntax**

|     |                                                                                                                                                                                                              |                              |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| FDC | <code>define_attribute {i: instName} syn_user_instance {1 0}</code><br><code>define_attribute {v: moduleName} syn_user_instance {1 0}</code><br><code>define_global_attribute syn_user_instance {1 0}</code> | <a href="#">FDC Examples</a> |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|

|         |                                                             |                                  |
|---------|-------------------------------------------------------------|----------------------------------|
| Verilog | <code>object /* synthesis syn_user_instance = 1 0 */</code> | <a href="#">Verilog Examples</a> |
|---------|-------------------------------------------------------------|----------------------------------|

|      |                                                                                |                               |
|------|--------------------------------------------------------------------------------|-------------------------------|
| VHDL | <code>attribute syn_user_instance of object : objectType is true false;</code> | <a href="#">VHDL Examples</a> |
|------|--------------------------------------------------------------------------------|-------------------------------|

In the above syntax:

| Value                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0   false<br>(default) | The tool optimizes user-instantiated primitives as needed, as long as Enhanced Optimization is enabled. See <a href="#">syn_macro</a> , <a href="#">syn_user_instance</a> , <a href="#">Enhanced Optimization and Cores</a> , on page 680 for details about how this attribute works with other attributes.                                                                                                                                                          |
| 1   true               | Prevents optimizations to user-instantiated primitives. If you specify this attribute globally, it prevents optimizations to all user-instantiated primitives in the entire design.<br>If the attribute is set on a module or instance, it prevents optimization to primitives within the module or instance at all hierarchical levels. If the module or instance contains RTL in addition to user-instantiated primitives, the RTL portion can still be optimized. |

## FDC Examples

|                             |                                                         |
|-----------------------------|---------------------------------------------------------|
| Global setting              | define_global_attribute syn_user_instance {1}           |
| Macro view                  | define_attribute {v:work.prep2_2} syn_user_instance {1} |
| User-instantiated primitive | define_attribute {i:LUTinst0} syn_user_instance {1}     |

## Verilog Examples

### Verilog Example 1: syn\_user\_instance on a View

```
module LutRestructure(
    input din,
    input in_lut0,in_lut1,in_lut2,in_lut3,in_lut4,in_lut5,
    input in2_lut0,in2_lut1,in2_lut2,in2_lut3,
    output out );

    LutRestructure_mod inst0(
        .din(din),.in_lut0(in_lut0),.in_lut1(in_lut1),
        .in_lut2(in_lut2),.in_lut3(in_lut3),.in_lut4(in_lut4),
        .in_lut5(in_lut5),.in2_lut0(in2_lut0),.in2_lut1(in2_lut1),
        .in2_lut2(in2_lut2),.in2_lut3(in2_lut3),.out(out) );
endmodule
```

```

module LutRestructure_mod(
    input din,
    input in_lut0,in_lut1,in_lut2,in_lut3,in_lut4,in_lut5,
    input in2_lut0,in2_lut1,in2_lut2,in2_lut3,
    output out ) /*synthesis syn_user_instance = 1*/;
wire out_1;

LUT6 LUTinst0(
    .O(out_1),.I5(in_lut5),.I2(in_lut4),.I0(in_lut3),
    .I4(in_lut2),.I3(in_lut1),.I1(in_lut0) );
defparam LUTinst0.INIT = 64'h8000000000000000;

LUT5 LUTinst1(
    .O(out),.I2(in2_lut0),.I0(din),.I4(in2_lut1),
    .I1(in2_lut2),.I3(out_1) );
defparam LUTinst1.INIT = 32'h80000000;
endmodule

```

## Verilog Example 2: syn\_user\_instance on an Instance

```

module LutRestructure(
    input din,
    input in_lut0,in_lut1,in_lut2,in_lut3,in_lut4,in_lut5,
    input in2_lut0,in2_lut1,in2_lut2,in2_lut3,
    output out );

LutRestructure_mod inst0(
    .din(din),.in_lut0(in_lut0),.in_lut1(in_lut1),
    .in_lut2(in_lut2),.in_lut3(in_lut3),.in_lut4(in_lut4),
    .in_lut5(in_lut5),.in2_lut0(in2_lut0),.in2_lut1(in2_lut1),
    .in2_lut2(in2_lut2),.in2_lut3(in2_lut3),.out(out) );
endmodule

module LutRestructure_mod(
    input din,
    input in_lut0,in_lut1,in_lut2,in_lut3,in_lut4,in_lut5,
    input in2_lut0,in2_lut1,in2_lut2,in2_lut3,
    output out );
wire out_1;

LUT6 LUTinst0(
    .O(out_1),.I5(in_lut5),.I2(in_lut4),.I0(in_lut3),.I4(in_lut2),
    .I3(in_lut1),.I1(in_lut0) )
/*synthesis syn_user_instance = 1*/;
defparam LUTinst0.INIT = 64'h8000000000000000;

```

```
LUT5 LUTinst1(
    .O(out),.I2(in2_lut0),.I0(din),.I4(in2_lut1),.I1(in2_lut2),
    .I3(out_1) /*synthesis syn_user_instance = 1*/;
defparam LUTinst1.INIT = 32'h80000000;
endmodule
```

## VHDL Examples

### VHDL Example 1: syn\_user\_instance on a View

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library synplify;
use synplify.components.all;
library UNISIM;
use UNISIM.VCOMPONENTS.all;

entity LutRestructure is
    port (out_c : out std_logic;
          in2_lut1_c : in std_logic;
          in2_lut0_c : in std_logic;
          in2_lut2_c : in std_logic;
          din_c : in std_logic;
          in_lut5_c : in std_logic;
          in_lut2_c : in std_logic;
          in_lut1_c : in std_logic;
          in_lut4_c : in std_logic;
          in_lut0_c : in std_logic;
          in_lut3_c : in std_logic );
end LutRestructure;

architecture beh of LutRestructure is
signal OUT_L : std_logic ;
signal GND : std_logic ;
signal VCC : std_logic ;
attribute syn_user_instance : boolean;
attribute syn_user_instance of beh : architecture is true;

begin
LUTINST0: LUT6
generic map (INIT => X"8000000000000000")
port map (
    I0 => in_lut3_c,
    I1 => in_lut0_c,
    I2 => in_lut4_c,
```

```

I3 => in_lut1_c,
I4 => in_lut2_c,
I5 => in_lut5_c,
O => OUT_L );
LUTINST1: LUT5
generic map (INIT => X"80000000")
port map (
  I0 => din_c,
  I1 => in2_lut2_c,
  I2 => in2_lut0_c,
  I3 => OUT_L,
  I4 => in2_lut1_c,
  O => out_c );
GND <= '0';
VCC <= '1';
end beh;

```

### VHDL Example 2: syn\_user\_instance on an Instance

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library synplify;
use synplify.components.all;
library UNISIM;
use UNISIM.VCOMPONENTS.all;

entity LutRestructure is
  port (out_c : out std_logic;
        in2_lut1_c : in std_logic;
        in2_lut0_c : in std_logic;
        in2_lut2_c : in std_logic;
        din_c : in std_logic;
        in_lut5_c : in std_logic;
        in_lut2_c : in std_logic;
        in_lut1_c : in std_logic;
        in_lut4_c : in std_logic;
        in_lut0_c : in std_logic;
        in_lut3_c : in std_logic);
end LutRestructure;

architecture beh of LutRestructure is
signal OUT_L : std_logic ;
signal GND : std_logic ;
signal VCC : std_logic ;
attribute syn_user_instance : boolean;
attribute syn_user_instance of LUTINST0 : label is true;

```

```
attribute syn_user_instance of LUTINST1 : label is true;
begin
LUTINST0: LUT6
generic map (INIT => X"8000000000000000")
port map (
    I0 => in_lut3_c,
    I1 => in_lut0_c,
    I2 => in_lut4_c,
    I3 => in_lut1_c,
    I4 => in_lut2_c,
    I5 => in_lut5_c,
    O => OUT_L );
LUTINST1: LUT5
generic map (INIT => X"80000000")
port map (
    I0 => din_c,
    I1 => in2_lut2_c,
    I2 => in2_lut0_c,
    I3 => OUT_L,
    I4 => in2_lut1_c,
    O => out_c );
GND <= '0';
VCC <= '1';
end beh;
```

## translate\_off/translate\_on

### *Directive*

Synthesizes designs originally written for use with other synthesis tools without needing to modify source code.

### Description

Allows you to synthesize designs originally written for use with other synthesis tools without needing to modify source code. All source code that is between these two directives is ignored during synthesis.

Another use of these directives is to prevent the synthesis of stimulus source code that only has meaning for logic simulation. You can use translate\_off/translate\_on to skip over simulation-specific lines of code that are not synthesizable.

When you use translate\_off in a module, synthesis of all source code that follows is halted until translate\_on is encountered. Every translate\_off must have a corresponding translate\_on. These directives cannot be nested, therefore, the translate\_off directive can only be followed by a translate\_on directive.

See also, [pragma translate\\_off/pragma translate\\_on, on page 493](#). These directives are implemented the same in the source code.

### translate\_off/translate\_on Syntax

Verilog    **/\* synthesis translate\_off \*/**  
            **/\* synthesis translate\_on \*/**

VHDL    **synthesis translate\_off**  
            **synthesis translate\_on**

### Verilog Example

```
module test(input a, b, output dout, Nout);
    assign dout = a + b;

    //Anything between pragma translate_off/translate_on is ignored by
    //the synthesis tool hence only
    //the adder circuit above is implemented not the multiplier circuit
    //below:
```

```

/* synthesis translate_off */
assign Nout = a * b;
/* synthesis translate_on */

endmodule

```

For SystemVerilog designs, you can alternatively use the `synthesis_off`/`synthesis_on` directives. The directives function the same as the `translate_off`/`translate_on` directives to ignore all source code contained between the two directives during synthesis.

For Verilog designs, you can use the `synthesis` macro with the Verilog `'ifdef` directive instead of the `translate on/off` directives.

## VHDL Example

For VHDL designs, you can alternatively use the `synthesis_off`/`synthesis_on` directives. Select Project->Implementation Options->VHDL and enable the Synthesis On/Off Implemented as Translate On/Off option. This directs the compiler to treat the `synthesis_off`/`on` directives like `translate_off`/`on` and ignore any code between these directives.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
    port (a : in std_logic_vector(1 downto 0);
          b : in std_logic_vector(1 downto 0);
          dout : out std_logic_vector(1 downto 0);
          Nout : out std_logic_vector(3 downto 0) );
end;

architecture rtl of test is
begin
    dout <= a + b;

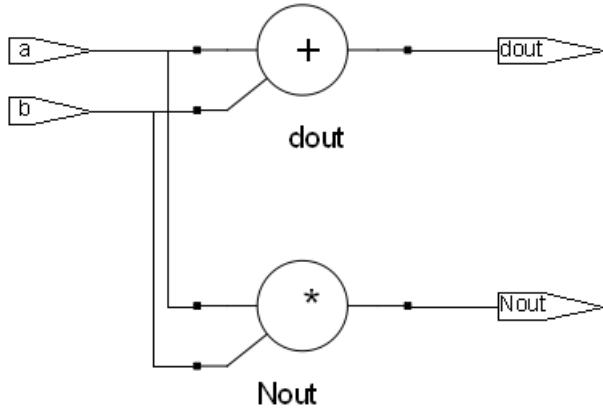
    -- Anything between pragma translate_off/translate_on is ignored
    -- by the synthesis tool hence only the adder circuit above is
    -- implemented not the multiplier circuit below:

    -- pragma translate_off
    Nout <= a * b;
    -- pragma translate_on
end;

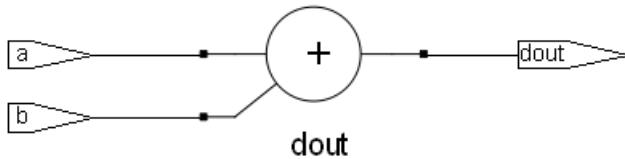
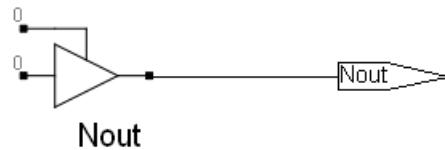
```

## Effects of Using translate\_off/translater\_on

Here is the view before applying the attribute to the examples.



This is the view after applying the attribute.



## xc\_area\_group

*Attribute/Directive*

Specifies the region where the instance should be placed.

### Description

xc\_area\_group is used to assign a compile point to a Xilinx area group; by specifying its location and area. Physical regions you define are automatically assigned to corresponding Xilinx area groups. During fitting, the Xilinx place-and-route tool places all of the compile-point logic within the assigned area group.

Apply this attribute to a compile point. The attribute assigns the compile point to a Xilinx area group, specifying its location and area. During fitting, the Xilinx place-and-route tool places all of the compile point logic within the assigned area group. You must not use this attribute on a module that has been instantiated more than once.

The attribute value specifies the location and area of the area group, by locating its top-left and bottom-right corners. N1 and N2 are integers that correspond to X and Y coordinates of the corners, respectively (for example, SLICE\_X N1 Y N2).

### Syntax

|          |                                                                                                                                 |                                            |
|----------|---------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| FDC File | <code>define_attribute {v:module architectureName}<br/>xc_area_group {topleft:bottomright}</code>                               | <a href="#">Constraints Editor Example</a> |
| Verilog  | <code>/* synthesis xc_area_group = "topleft:bottomright" */;</code>                                                             | <a href="#">Verilog Example</a>            |
| VHDL     | <code>attribute xc_area_group : string;<br/>attribute xc_area_group of object : objectType is<br/>"topleft:bottomright";</code> | <a href="#">VHDL Example</a>               |

### Constraints Editor Example

|   | Enable                              | Object Type | Object       | Attribute     | Value                 | Value Type | Description                                          |
|---|-------------------------------------|-------------|--------------|---------------|-----------------------|------------|------------------------------------------------------|
| 1 | <input checked="" type="checkbox"/> | instance    | v:work.mod10 | xc_area_group | CLB_R10C29:CLB_R16C35 | string     | Specifies the region where instance should be placed |

```
define_attribute {v:work.mod10} {xc_area_group}
{CLB_R10C29:CLB_R16C35}
```

## Verilog Example

```
module xc_area_group (a,b,c,clk,out1,out2)/* synthesis
    xc_area_group = "CLB_R10C29:CLB_R16C35" */;
    input a,b,c,clk;
    output reg out1,out2;
    mod10 inst1 (.clk(clk),.c(c),.d(out2));
    always @ (posedge clk)
    out1 <= a + b;
endmodule

module mod10 (clk,c,d);
    input c,clk;
    output reg d;
    always @ (posedge clk)
    d <= c;
endmodule
```

## VHDL Example

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

entity test is
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          clk : in std_logic;
          out1 : out std_logic;
          out2 : out std_logic );
end test;

architecture arch of test is
attribute xc_area_group : string;
attribute xc_area_group of arch: architecture is
    "CLB_R10C29:CLB_R16C35";
component test1
    port (c : in std_logic;
          clk : in std_logic;
          d : out std_logic );
end component;
begin
inst1 : test1 port map (c =>c, d=>out2,clk=>clk);
```

```
process (clk)
begin
    if (clk'event and clk ='1')then
        out1 <= a and b;
    end if;
end process;
end arch;

library IEEE;
use ieee.std_logic_1164.all;

entity test1 is
    port (c : in std_logic;
          clk : in std_logic;
          d : out std_logic );
end test1;

architecture arch of test1 is
begin
    process (clk)
begin
    if (clk'event and clk ='1')then
        d <= c;
    end if;
end process;
end;
```

## Effect of Using `xc_area_group`

The following netlist is generated, before applying the attribute:

```
library work
(edifLevel 0)
(technology (numberDefinition))
(cell mod10 (cellType GENERIC)
(view verilog (viewType NETLIST)
(interface
(port clk (direction INPUT))
(port c (direction INPUT))
(port d (direction OUTPUT))
)
(contents
(instance (rename dZ0 "d") (viewRef PRIM (cellRef FD
(libraryRef UNILIB)))
)
(net clk (joined
(portRef clk)
(portRef C (instanceRef dZ0))
))
(net c (joined
(portRef c)
(portRef D (instanceRef dZ0))
))
(net d (joined
(portRef Q (instanceRef dZ0))
(portRef d)
```

The following netlist is generated, after applying the attribute:

```
FDC      define attribute {v:work.mod10} {xc_area_group}
          {CLB_R10C29:CLB_R16C3}

          (library work
          (edifLevel 0)
          (technology (numberDefinition))
          (cell mod10 (cellType GENERIC)
          (view verilog (viewType NETLIST)
          (interface
          (port clk (direction INPUT))
          (port c (direction INPUT))
          (port d (direction OUTPUT)))
          )
          (contents
          (instance (rename dZ0 "d") (viewRef PRIM (cellRef FD (libraryRef
          UNILIB)))
          )
          (net clk (joined
          (portRef clk)
          (portRef C (instanceRef dZ0)))
          )
          (net c (joined
          (portRef c)
          (portRef D (instanceRef dZ0)))
          )
          (net d (joined
          (portRef Q (instanceRef dZ0))
          (portRef d)))
          )
          (property xc_area_group (string "SLICE_X10Y29:SLICE_X16Y35"))
          )
          )
```

---

## xc\_clockbuftype

*Attribute/Directive*

Controls the use of the BUFGDLL buffer for a clock port.

### Description

Uses the Clock Delay Locked Loop primitive (CLKDLL) for a clock port. This is inferred as a buffer called BUFGDLL, which includes the CLKDLL primitive. BUFGDLL consists of an IBUFG, followed by a CLKDLL, followed by a BUFG. It is a special purpose clock delay locked loop buffer for clock skew management.

### xc\_clockbufftype Syntax

|          |                                                                            |                                            |
|----------|----------------------------------------------------------------------------|--------------------------------------------|
| FDC File | <code>define_attribute {port} xc_clockbuftype {BUFGDLL}</code>             | <a href="#">Constraints Editor Example</a> |
| Verilog  | <code>object /* synthesis xc_clockbuftype = "BUFGDLL" */;</code>           | <a href="#">Verilog Example</a>            |
| VHDL     | <code>attribute xc_clockbuftype of object : objectType is "BUFGDLL"</code> | <a href="#">VHDL Example</a>               |

### Constraints Editor Example

| Enabled | Object Type | Object | Attribute       | Value   | Val Type | Description                         | Comment |
|---------|-------------|--------|-----------------|---------|----------|-------------------------------------|---------|
| 1       | input_port  | pclk   | xc_clockbuftype | BUFGDLL | string   | Use the Xilinx BUFGDLL clock buffer |         |

### Verilog Example

```
module xc_clock (d,rst,clk,q);
  input [3:0] d;
  input rst;
  input clk /* synthesis syn_clockbuftype = "BUFGDLL" */;
  output reg[3:0] q;
```

```
always@ (posedge clk)
begin
    if(rst)
        q<=0;
    else
        q<=d;
end
endmodule
```

## VHDL Example

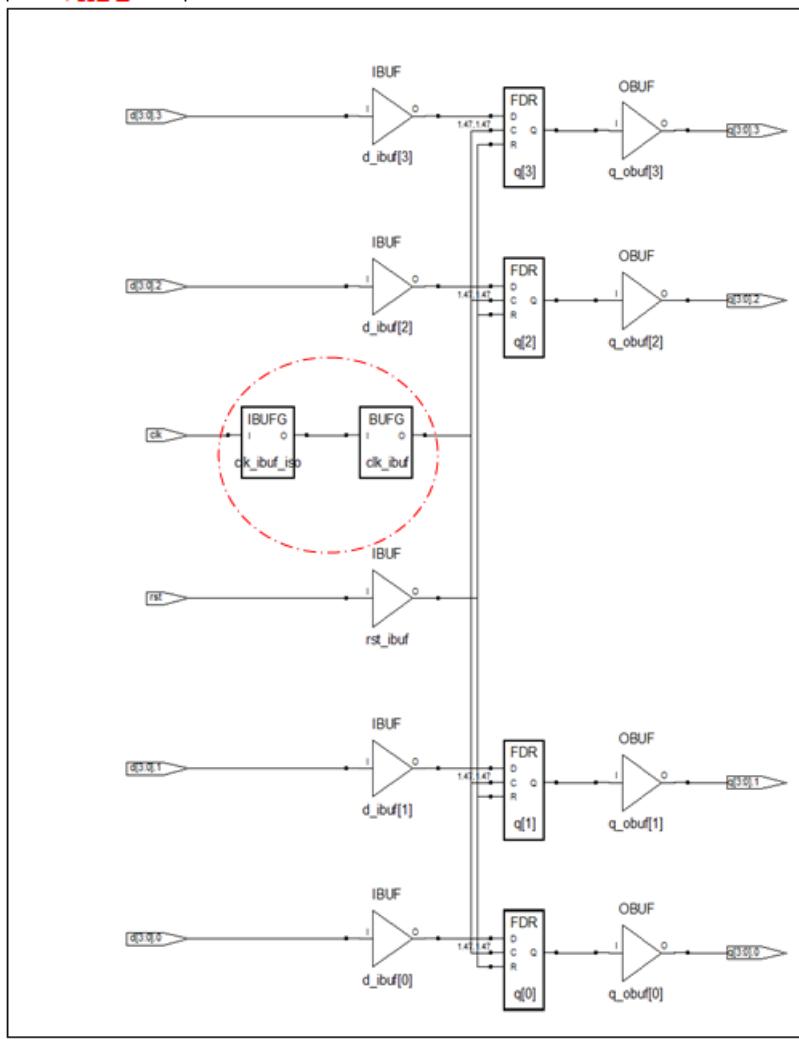
```
library ieee ;
use ieee.std_logic_1164.all;
use work.all;

entity dff is
    port (data_in : in std_logic_vector (3 downto 0);
          clock,rst : in std_logic;
          data_out: out std_logic_vector (3 downto 0) );
attribute syn_clockbuftype : string;
attribute syn_clockbuftype of clock : signal is "BUFGDLL";
end dff;

architecture behv of dff is
begin
    process(data_in, clock)
    begin
        if (clock'event and clock='1') then
            if rst='1' then
                data_out <= "0000";
            else
                data_out <= data_in;
            end if;
        end if;
    end process;
end behv;
```

## Effect of Using `xc_clockbufftype`

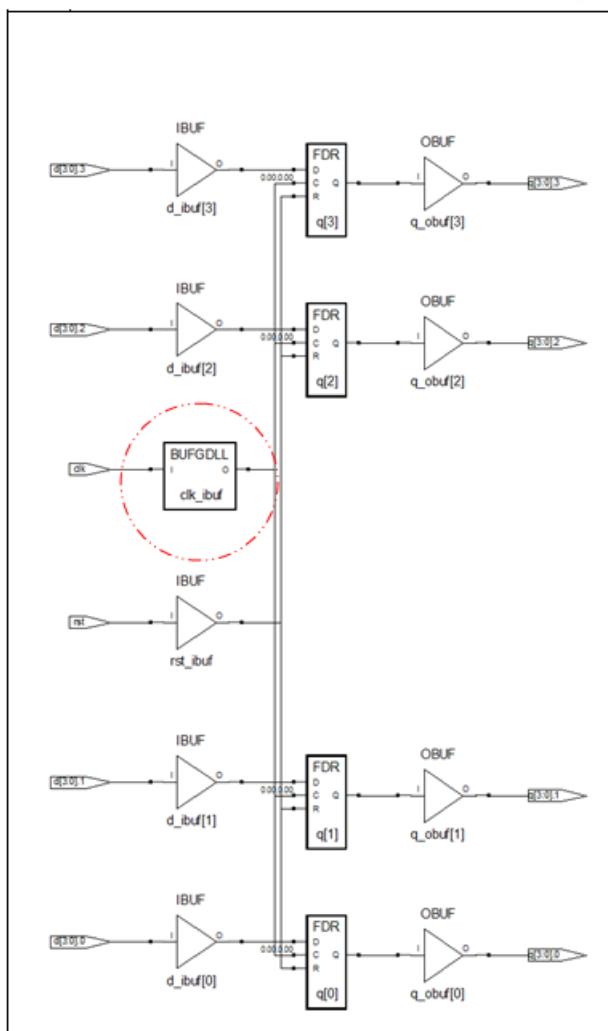
The following example shows a design without the `xc_clockbufftype` attribute:



The next figure shows the same design with a BUFGDLL after the `xc_clockbuftype` attribute is applied:

**Verilog**    `input clk /* synthesis xc_clockbuftype = "BUFGDLL" */;`

**VHDL**    `attribute xc_clockbuftype : string;`  
`attribute xc_clockbuftype of clock : signal is "BUFGDLL";`



## **xc\_fast**

*Attribute.*

Speeds up transition time of the output driver.

### **xc\_fast Values**

| Value               | Description                     |
|---------------------|---------------------------------|
| 0   false (Default) | Keeps the transition time slow. |
| 1   true            | Speeds up the transition time.  |

### **Description**

Use this attribute to make the transition time of the output driver fast. The default transition time is slow (see [xc\\_slow, on page 910](#)).

The synthesis tool provides attributes that are passed into the XNF or EDIF netlist for placement and routing. These attributes affect the input setup times and output transition times for your I/Os. The transition time of the output driver can be programmed as fast or slow. Use the `xc_fast` attribute to decrease the transition time for the output driver. The `xc_fast` attribute is passed to the Xilinx I/O Block Parameter as `FAST` in the XNF or EDIF netlist.

### **xc\_fast Syntax**

FDC      **define\_attribute {object} xc\_fast {0|1}**

[Constraints Editor Example](#)

Verilog    **object /\* synthesis xc\_fast = "0|1" \*/;**

[Verilog Example](#)

VHDL     **attribute xc\_fast of object : objectType is "0|1";**

[VHDL Example](#)

### **Constraints Editor Example**

|   | Enabled                             | Object Type | Object       | Attribute | Value | Val Type | Description |
|---|-------------------------------------|-------------|--------------|-----------|-------|----------|-------------|
| 1 | <input checked="" type="checkbox"/> |             | p:DATA0[7:0] | xc_fast   | 1     |          |             |

## Verilog Example

```
module counter (CLK, RST, DATA0);
    input CLK, RST;
    output [7:0] DATA0 /* synthesis xc_fast = 1 */;
    reg [7:0] DATA0;

    always @ (posedge CLK or posedge RST)
    begin
        if (RST)
            DATA0 = 0;
        else
            DATA0 = DATA0 + 1;
    end
endmodule
```

## VHDL Example

```
entity count is
    port (clk : in bit;
          rst : in bit;
          data0 : out std_logic_vector(7 downto 0) );
attribute xc_fast : Boolean;
attribute xc_fast of data0 : signal is true;
end count;

architecture behave of count is
signal data0_i: std_logic_vector(7 downto 0);
begin
    counter: process (rst,clk)
    begin
        if  rst = '1' then
            data0_i <= "00000000";
        elsif clk = '1' and clk'event then
            data0_i  <= data0_i + "00000001";
        end if;
    end process counter;
    data0 <= data0_i;
end behave;
```

## Effect of Using xc\_fast

The following table shows a design without the xc\_fast attribute.

---

Verilog    output [7:0] DATA0 /\* synthesis xc\_fast = 0 \*/;

---

VHDL    attribute xc\_fast of data0 : signal is false;

---

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell counter (cellType GENERIC)
    (view verilog (viewType NELIST)
      (interface
        (port (array (rename DATA0 "DATA0[7:0]") 8) (direction OUTPUT))
        (port CLK (direction INPUT)
      )
      (port RST (direction INPUT))
    )
  )
)
```

The following table shows a design with the xc\_fast attribute.

---

Verilog    output [7:0] DATA0 /\* synthesis xc\_fast = 1 \*/;

---

VHDL    attribute xc\_fast of data0 : signal is true;

---

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell counter (cellType GENERIC)
    (view verilog (viewType NELIST)
      (interface
        (port (array (rename DATA0 "DATA0[7:0]") 8) (direction OUTPUT)
          (property xc_fast (integer 1))
      )
    )
  )
)
```

When place and route is run, a line is added for each output port bit with an xc\_fast attribute in the following files:

```
E:\srm\xc_fast\rev_1\pr_1\counter.pad(257):P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMOS25*|2|12
|FAST||||UNLOCATED|NO|NONE|
E:\srm\xc_fast\rev_1\pr_1\counter_map.mrp(172): | DATA0[0] | IOB| OUTPUT | LVCMOS25|| 12| FAST ||
E:\srm\xc_fast\rev_1\pr_1\counter_pad.txt(258):|P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMOS25*|2|12|
FAST||||UNLOCATED|NO|NONE |
E:\srm\xc_fast\rev_1\pr_1\proj_3.edf(193):      (property FAST (string ""))
E:\srm\xc_fast\rev_1\pr_1\counter_map.xrpt(239): <item label="SlewxA;Rate" stringID="SLEW_RATE"
value="FAST"/>
E:\srm\xc_fast\rev_1\pr_1\counter_par.xrpt(1450): <item label="SlewxA;Rate" stringID="Slew_Rate"
value="FAST"/>
E:\srm\xc_fast\rev_1\pr_1\proj_3.xise(223): <property xil_pn:name="Output Slew Rate" xil_pn:value=
"Fast" il_pn:valueState="default"/>
```

## xc\_fast\_auto

### *Attribute*

Forces inference of slow buffers.

### Description

By default (xc\_fast\_auto value of 1), the synthesis tool infers the fast output buffers OBUF\_F\_24, OBUFT\_F\_24, and IOBUF\_F\_24. Use this global attribute to force inference of slow buffers. In HDL source code you specify xc\_fast\_auto for the top-level module or architecture. You can override this attribute on an individual basis by using the [xc\\_padtype](#) attribute.

The xc\_fast\_auto attribute has no affect on output ports specified with the xc\_padtype attribute.

### xc\_fast\_auto Syntax

|          |                                                                                                                 |                                            |
|----------|-----------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| FDC File | <code>define_global_attribute xc_fast_auto {0 1}</code>                                                         | <a href="#">Constraints Editor Example</a> |
| Verilog  | <code>object /* synthesis xc_fast_auto = 0 1 */;</code>                                                         | <a href="#">Verilog Example</a>            |
| VHDL     | <code>attribute xc_fast_auto : boolean;<br/>attribute xc_fast_auto of object : objectType is false true;</code> | <a href="#">VHDL Example</a>               |

### Constraints Editor Example

|   | Enable                              | Object Type | Object   | Attribute    | Value | Value Type | Description                            | Comment |
|---|-------------------------------------|-------------|----------|--------------|-------|------------|----------------------------------------|---------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Global> | xc_fast_auto | 0     | boolean    | Enable automatic fast output buffer... |         |

### Verilog Example

```
module top( input a, b, clk, input gsrin, output fout)
    /*synthesis xc_fast_auto=0 */;
    test T1(a, b, clk, gsrin, fout);
endmodule
```

```
//module test
module test(a, b, clk, gsrin, out)/* synthesis syn_black_box */;
    input a, b, clk;
    input gsrin;
    output out;
endmodule
```

## VHDL Example

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

entity top is
    port (a : in std_logic;
          b : in std_logic;
          clk : in std_logic;
          gsrin : in std_logic;
          dout : out std_logic );
end top;

architecture rtl of top is
component test is
    port (a : in std_logic;
          b : in std_logic;
          clk : in std_logic;
          gsrin : in std_logic;
          o : out std_logic );
end component;
begin
    U1 : test port map(a, b, clk, gsrin, dout);
end rtl;

--Black Box module
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
    port (a : in std_logic;
          b : in std_logic;
          clk : in std_logic;
          gsrin : in std_logic;
          o : out std_logic );
end;
```

```
architecture rtl of test is
attribute syn_black_box : boolean;
attribute syn_black_box of rtl : architecture is true;
attribute xc_fast_auto : boolean;
attribute xc_fast_auto of rtl : architecture is false;
begin
end;
```

## Effect of Using `xc_fast_auto`

The tool infers fast output buffers by default, however, you can control this behavior using the `xc_fast_auto` attribute. If you set `xc_fast_auto=0`, fast buffer inferencing is turned off and this value is forward annotated to the output EDIF netlist as:

```
(property xc_fast_auto (integer 0))
```

## **xc\_global\_buffers**

### *Attribute*

Controls the number of global buffers used in a device.

### **Description**

Controls the number of global buffers used in a device. Synthesis automatically adds global buffers for clock nets with high fanout. Use this attribute to specify a maximum number of buffers and restrict the amount of global buffer resources added. Also, if there is a black box in the design that has global buffers, you can use `xc_global_buffers` to prevent the synthesis tool from inferring clock buffers or exceeding the number of global resources. You can only specify this attribute through a constraint file (cannot be specified in HDL source code).

The `xc_global_buffers` attribute has the same effect as the `syn_global_buffers` attribute. If both attributes are specified in the same design, `syn_global_buffers` overwrites `xc_global_buffers`. See [syn\\_global\\_buffers, on page 636](#) for details about this attribute.

It is recommended that you use the global buffers attribute `syn_global_buffers` instead; see [syn\\_global\\_buffers, on page 636](#).

### **xc\_global\_buffers Syntax**

The following table summarizes the syntax in different files.

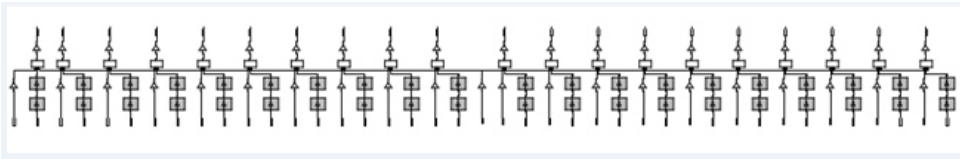
|          |                                                                     |                            |
|----------|---------------------------------------------------------------------|----------------------------|
| FDC File | <code>define_global_attribute xc_global_buffers {maxbuffers}</code> | Constraints Editor Example |
|----------|---------------------------------------------------------------------|----------------------------|

### **Constraints Editor Example**

|   | Enable                              | Object Type | Object   | Attribute         | Value | Value Type | Description              | Comment |
|---|-------------------------------------|-------------|----------|-------------------|-------|------------|--------------------------|---------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Global> | xc_global_buffers | 3     | integer    | Number of global buffers |         |

### **Effect of Using xc\_global\_buffers**

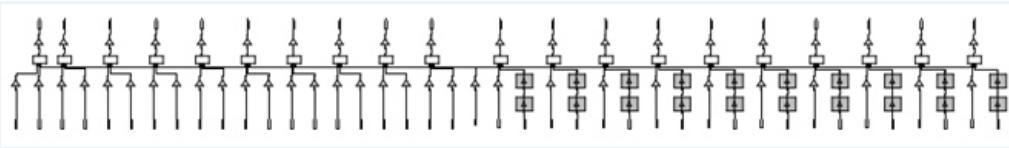
Before applying attribute:



A message such as the one below is generated:

```
@W:FX726: | Ignoring out-of-range global buffer count of 33 for  
chip view:work.top(behave)
```

After applying attribute:



Verify results in the log file.

```
@N:FX112 : | Setting available global buffers in chip view:work.top(behave) to 10
Clock Buffers:
  Inserting Clock buffer for port clk[0],
  Inserting Clock buffer for port clk[1],
  Inserting Clock buffer for port clk[2],
  Inserting Clock buffer for port clk[3],
  Inserting Clock buffer for port clk[4],
  Inserting Clock buffer for port clk[5],
  Inserting Clock buffer for port clk[6],
  Inserting Clock buffer for port clk[7],
  Inserting Clock buffer for port clk[8],
  Inserting Clock buffer for port clk[9],
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[10] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[11] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[12] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[13] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[14] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[15] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[16] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[17] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[18] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[19] in view view:work.top(behave) (fanout 1)
@N:FX112 : | Setting available global buffers in chip view:work.top(behave) to 10
```

## **xc\_isgsr**

### *Attribute*

Specifies that a port on a black box is connected to an internal STARTUP block.

### **Description**

Specifies that a port on a black box is connected to an internal STARTUP block. Use this directive with designs targeting Xilinx-specific families where the synthesis tool infers a STARTUP block.

### **xc\_isgsr Syntax**

|         |                                                                                                        |                                            |
|---------|--------------------------------------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute {instance.resetPort} xc_isgsr {0 1}</code>                                      | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis xc_isgsr = 0 1 */;</code>                                                    | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute xc_isgsr : string;<br/>attribute xc_isgsr of object : objectType is false true;</code> | <a href="#">VHDL Example</a>               |

### **Constraints Editor Example**

|   | Enable                              | Object Type | Object      | Attribute | Value | Value Type | Description | Comment |
|---|-------------------------------------|-------------|-------------|-----------|-------|------------|-------------|---------|
| 1 | <input checked="" type="checkbox"/> | <any>       | bbgsr.gsrin | xc_isgsr  | 1     |            |             |         |

### **Verilog Example**

```
module top( input a, b, clk, input gsrin, output fout);
  test T1(a, b, clk, gsrin, fout);
endmodule

module test(a, b, clk, gsrin, out)/* synthesis syn_black_box */;
  input a, b, clk;
  input gsrin /* synthesis xc_isgsr = 1 */;
  output out;
endmodule
```

## VHDL Example

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

entity top is
    port (a : in std_logic;
          b : in std_logic;
          clk : in std_logic;
          gsrin : in std_logic;
          dout : out std_logic );
end top;

architecture rtl of top is
component test is
    port (a : in std_logic;
          b : in std_logic;
          clk : in std_logic;
          gsrin : in std_logic;
          o : out std_logic );
end component;
begin
    U1 : test port map(a, b, clk, gsrin, dout);
end rtl;

--Black Box module
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
    port (a : in std_logic;
          b : in std_logic;
          clk : in std_logic;
          gsrin : in std_logic;
          o : out std_logic );

attribute xc_isgsr : boolean;
attribute xc_isgsr of gsrin : signal is true;
end;
```

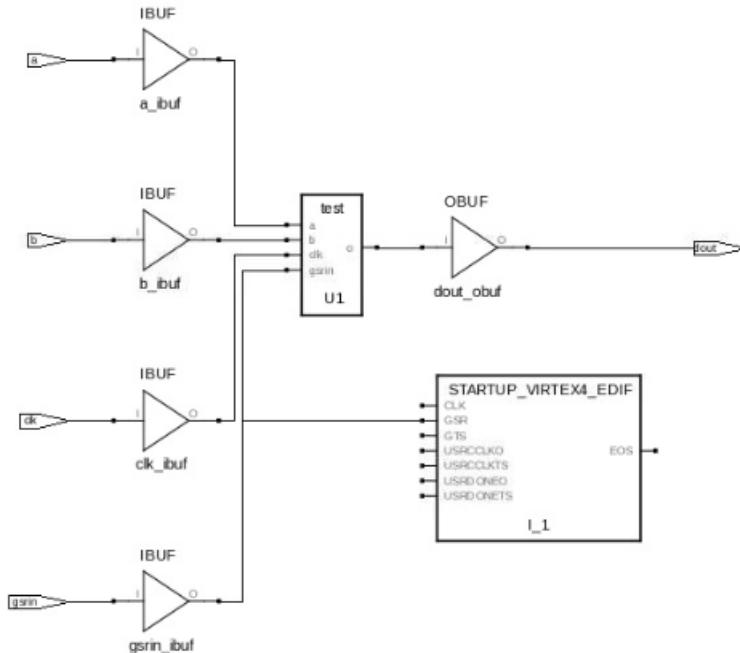
```

architecture rtl of test is
attribute syn_black_box : boolean;
attribute syn_black_box of rtl : architecture is true;
begin
end;

```

## Effect of Using xc\_isgsr

The reset input pin of the black box is connected to a GSR block as shown in the view below:



## **xc\_loc**

### *Attribute*

Specifies the location (placement) of ports.

### **Description**

Specifies the location (placement) of ports. Refer to the Xilinx databook for valid placement locations. This attribute can be specified in a top-level source file or the constraint file.

It is recommended that you use the Xilinx location attribute `syn_loc` instead; see [syn\\_loc, on page 672](#).

### **xc\_loc Syntax**

|         |                                                                        |                                            |
|---------|------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute {portDesignName} xc_loc {placements}</code>     | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis xc_loc = "placements" */;</code>             | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute xc_loc of object : objectType is "placements" ;</code> | <a href="#">VHDL Example</a>               |

### **Constraints Editor Example**

|   | Enable                              | Object Type | Object       | Attribute | Value                  | Value Type | Description    |
|---|-------------------------------------|-------------|--------------|-----------|------------------------|------------|----------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | p:data0[4:0] | xc_loc    | P5, P11, P12, P14, P21 | string     | Port placement |

The following example is a constraint file assignment of a pad location to all bits of a 5-bit bus.

```
define_attribute {DATA0[4:0]} xc_loc {P5,P11,P12,P14,P21}
```

## Verilog Example

```
module test(a ,b, clk, out1);
    input clk;
    input [2:0]a;
    input [2:0]b;
    output [2:0] out1;
    reg [2:0] out1/* synthesis xc_loc = "P14,P12,P11" */;

    always@(posedge clk)
    begin
        out1 <= a + b;
    end
endmodule
```

## VHDL Example

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

entity test is
generic (s : integer := 4);
    port (clk: in std_logic;
          in1: in std_logic_vector(s downto 0);
          in2: in std_logic_vector(s downto 0);
          d_out: out std_logic_vector(9 downto 0) );
attribute xc_loc : string;
attribute xc_loc of d_out : signal is "P14,P12,P11,P5,P21";
end test;

architecture beh of test is
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= in1 & in2;
        end if;
    end process;
end beh;
```

## Effect of Using `xc_loc`

The specified ports are locked to particular pins of the FPGA. This information, along with the output EDIF netlist, is forward-annotated to the place-and-route tool through the XDC file.

After applying the attribute:

```
create_clock [get_ports {clk}] -period {5}
set_property LOC P11 [get_cells {out1obuf[0]}]
set_property LOC P12 [get_cells {out1obuf[1]}]
set_property LOC P14 [get_cells {out1obuf[2]}]

set_property LOC P11 [get_ports {out1[0]}]
set_property LOC P12 [get_ports {out1[1]}]
set_property LOC P14 [get_ports {out1[2]}]
set_property IOB true [get_cells {out1[2]}]
set_property IOB true [get_cells {out1[1]}]
set_property IOB true [get_cells {out1[0]}]
```

## Examples: xc\_loc on Individual Bus Bits

You can use xc\_loc to specify locations of individual bits of the bus in one of two ways. This example specifies the bit:

```
define_attribute {valid[0]} xc_loc {R6}
define_attribute {valid[1]} xc_loc {T2}
define_attribute {valid[2]} xc_loc {R5}
define_attribute {valid[3]} xc_loc {R1}
```

This example uses the b: prefix as well as the bit slice:

```
define_attribute {b:valid[0]} xc_loc {R6}
define_attribute {b:valid[1]} xc_loc {T2}
define_attribute {b:valid[2]} xc_loc {R5}
define_attribute {b:valid[3]} xc_loc {R1}
```

## xc\_map

### *Attribute*

Specifies RLOCs. RLOCs are relative location constraints.

### Description

RLOCs are relative location constraints. They let you control placement in critical sections, thus improving performance.

Specify RLOCs using xc\_map along with xc\_rloc and xc\_uset. As with other attributes, you can define them in the source code, or in the Constraints Editor window.

### xc\_map Syntax

|          |                                                                           |                                            |
|----------|---------------------------------------------------------------------------|--------------------------------------------|
| FDC File | <code>define_attribute {v:primitiveName} xc_map {fmap hmap lut}</code>    | <a href="#">Constraints Editor Example</a> |
| Verilog  | <code>object /* synthesis xc_map = "fmap hmap lut" */;</code>             | <a href="#">Verilog Example</a>            |
| VHDL     | <code>attribute xc_map of object : objectType is "fmap hmap lut" ;</code> | <a href="#">VHDL Example</a>               |

### Constraints Editor Example

|   | Enable                              | Object Type | Object      | Attribute | Value | Value Type | Description                 |
|---|-------------------------------------|-------------|-------------|-----------|-------|------------|-----------------------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | v:hmap_xor4 | xc_map    | fmap  | string     | Map entity to fmap/hmap/lut |

### Verilog Example

```
module fmap_xor4 (z, a, b, c, d)/* synthesis xc_map=fmap */;
output z;
input a, b, c, d;
assign z = a ^ b ^ c ^ d;
endmodule

module hmap_xor3 (z, a, b, c) /* synthesis xc_map=hmap */;
output z;
input a, b, c;
assign z = a ^ b ^c;
endmodule
```

```

module clb_xor9 (z, a);
output z;
input [8:0] a;
wire z03, z47;
fmap_xor4 x03 /* synthesis xc_uset="SET1" xc_rloc="R0C0.f" */
(z03, a[0], a[1], a[2], a[3]);
hmap_xor3 zz /* synthesis xc_uset="SET1" xc_rloc="R0C0.h" */
(z, z47, a[4], a[5]);
fmap_xor4 x47 /* synthesis xc_uset="SET1" xc_rloc="R0C0.g" */
(z47, z03, a[5], a[6], a[7]);
endmodule

module xor9top (z, a);
output z;
input [8:0] a;
clb_xor9 x (z, a);
endmodule

```

## VHDL Example

```

--hmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity hmap_xor4 is
    port (a, b, c, d : in std_logic;
          z : out std_logic );
end hmap_xor4;

architecture rtl of hmap_xor4 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "hmap";
begin
    z <= a xor b xor c xor d;
end rtl;

-- fmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity fmap_xor3 is
port ( a, b, c : in std_logic;
       z : out std_logic);
end fmap_xor3;

```

```
architecture rtl of fmap_xor3 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "fmap";
begin
    z <= (a and b) xor c;
end rtl;

-- hmap & fmap instantiations
library IEEE;
use IEEE.std_logic_1164.all;

entity clb_xor9 is
    port (a : in std_logic_vector(8 downto 0);
          z : out std_logic );
end clb_xor9;

architecture rtl of clb_xor9 is
signal z03, z47 : std_logic;
component fmap_xor3
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          z : out std_logic );
end component;

component hmap_xor4
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          d : in std_logic;
          z : out std_logic );
end component;
attribute xc_uset : string;
attribute xc_rloc : string;
attribute xc_uset of x03 : label is "SET1";
attribute xc_rloc of x03 : label is "R0C0.f";
attribute xc_uset of x47 : label is "SET1";
attribute xc_rloc of x47 : label is "R0C0.g";
attribute xc_uset of zz : label is "SET1";
attribute xc_rloc of zz : label is "R0C0.h";
begin
    x03 : hmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
    x47 : hmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
    zz : fmap_xor3 port map(z03, z47, a(8), z);
end rtl;
```

```
--Top level entity definition
library IEEE;
use IEEE.std_logic_1164.all;

entity top is
    port ( a : in std_logic_vector(8 downto 0);
           z : out std_logic );
end top;

architecture rtl of top is
component clb_xor9
    port ( a : in std_logic_vector (8 downto 0);
           z : out std_logic );
end component;
begin
    U1: clb_xor9 port map (a, z);
end rtl;
```

## **Effect of Using xc\_map**

The following example shows forward-annotated regional constraints in the output EDIF netlist after applying the attribute.

```
(property xc_map (string "hmap"))
(property orig_inst_of (string "hmap_xor4"))
)
)
(cell clb_xor9 (cellType GENERIC)
(view_netlist (viewType NETLIST)
(interface
    (port (array (rename a_c "a_c(8:0)") 9) (direction INPUT))
    (port z_c (direction OUTPUT))
)
(contents
(instance zz (viewRef PRIM (cellRef LUT3 (libraryRef VIRTEX)))
(property RLOC (string "R0C0.h"))
(property HU_SET (string "U1$SET1"))
(property INIT (string "8'h78"))
)
(instance x03 (viewRef netlist (cellRef hmap_xor4))
(property RLOC (string "R0C0.f"))
(property HU_SET (string "SET1"))
)
(instance x47 (viewRef netlist (cellRef hmap_xor4_0))
(property RLOC (string "R0C0.g"))
(property HU_SET (string "SET1"))
)
```

## **xc\_padtype**

### *Attribute*

Specifies an I/O buffer standard.

### **Description**

It is recommended that you use the Xilinx I/O buffer standard attribute `syn_pad_type` instead; see [\*syn\\_pad\\_type\*, on page 725](#).

## xc\_pullup/xc\_pulldown

### Attribute

Specifies that a port is either a pullup or pulldown.

### Description

Use this attribute to specify that a port is either a pullup or pulldown.

### Syntax Specification

FDC File    `define_attribute {portName} xc_pullup {1}`  
`define_attribute {portName} xc_pulldown {1}`

Verilog    `object /* synthesis xc_pulldown = 1 */;`  
`object /* synthesis xc_pullup = 1 */;`

VHDL    `attribute xc_pulldown of object : objectType is true;`  
`attribute xc_pullup of object : objectType is true;`

### Constraints Editor Example

|   | Enabled                             | Object Type | Object    | Attribute   | Value | Val Type | Description    |
|---|-------------------------------------|-------------|-----------|-------------|-------|----------|----------------|
| 1 | <input checked="" type="checkbox"/> | port        | p:A[7:0]  | xc_pullup   | 1     | boolean  | Add a pullup   |
| 2 | <input checked="" type="checkbox"/> | port        | p:P[16:0] | xc_pulldown | 1     | boolean  | Add a pulldown |

For example, you can apply the xc\_pullup/xc\_pulldown attribute as follows:

- On a multi-bit port such as RXD[3:0]

```
define_attribute {RXD[3:0]} xc_pullup {1}
```

- On a single-bit port such as RXA0

```
define_attribute {RXA0} xc_pullup {1}
```

- To define this attribute on the  $n^{\text{th}}$  bit of the multi-bit port RXD[3:0], separate out the output port RXD[n]. For example, apply the attribute on the single-bit port RXD\_n.

```
define_attribute {RXD_1} xc_pullup {1}
```

## Verilog Example

```
module pullup_down (clk, A,B,PC,P);
input clk;
input [7:0] A /* synthesis syn_pullup = 1 */;
input [7:0] B, PC;
output reg [16:0] P /* synthesis syn_pulldown = 1 */;
reg [7:0] a_d, b_d;
reg [16:0] m;

always @ (posedge clk) begin
    a_d <= A;
    b_d <= B;
    m <= a_d * b_d;
    P <= m + PC;
end
endmodule
```

## VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity dsp_style is
    port (clk : in std_logic;
          A : in std_logic_vector(7 downto 0);
          B : in std_logic_vector(7 downto 0);
          PC : in std_logic_vector(7 downto 0);
          P : out std_logic_vector(15 downto 0) );
end dsp_style;

architecture rtl of dsp_style is
signal m : std_logic_vector(15 downto 0);
attribute syn_pullup : boolean;
attribute syn_pullup of A : signal is true;
attribute syn_pulldown : boolean ;
attribute syn_pulldown of P : signal is true;
begin
process(clk)
begin
    if (clk'event and clk = '1') then
```

```
m <= A * B;  
P <= m + PC;  
end if;  
end process;  
end rtl;
```

## Effect of Using **xc\_pullup**/**xc\_pulldown**

Before applying the attribute:

---

Verilog      input [7:0] A /\* synthesis syn\_pullup = 0 \*/;  
                output reg [16:0] P /\* synthesis syn\_pulldown = 0 \*/;

---

VHDL      attribute syn\_pullup of A : signal is false ;  
                attribute syn\_pulldown of P : signal is false ;

---

## Log File

```

Mapper Initialization Complete (Time elapsed 0h:00m:00s; Memory used current: 49MB peak: 49MB)

Adding property xc_pullup, value 0, to port A[7:0]
Adding property xc_pulldown, value 0, to port P[16:0]

Start loading timing files (Time elapsed 0h:00m:00s; Memory used current: 49MB peak: 50MB)

```

## Net List

```

(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell pullup_down (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port (array (rename A "A[7:0]" ) 8) (direction INPUT)
          (property xc_pullup (integer 0))
        )
        (port (array (rename B "B[7:0]" ) 8) (direction INPUT))
        (port (array (rename PC "PC[7:0]" ) 8) (direction INPUT))
        (port (array (rename P "P[16:0]" ) 17) (direction OUTPUT)
          (property xc_pulldown (integer 0))
        )
        (port clk (direction INPUT)
      )
    )
  )

```

## P&R Files

```

When P&R is run, there is no syn_pullup and syn_pulldown attributes in the following
files:

<projectdirectory>\rev_1\pr_1\pullup_down.pad(262):P7|A[0]|IOB|IO_L9N_CC_GC_4|INPUT|LVCMS25*|4|||
|NONE||UNLOCATED|NO|NONE|
(318): U9|P[4]|IOB|IO_L2N_GC_D10_4|OUTPUT|LVCMS25*|4|12|SLOW|||UNLOCATED|NO|NONE|
<projectdirectory>\rev_1\pr_1\pullup_down_pad.txt(263):|P7|A[0]|IOB|IO_L9N_CC_GC_4|INPUT|LVCMS25*|4|||NONE|UNLOCATED|NO|NONE|
(288): R14|P[0]|IOB|IO_L14P_17|OUTPUT|LVCMS25*|17|12|SLOW|||UNLOCATED|NO|NONE

```

## After applying the attribute:

Verilog      input [7:0] A /\* synthesis syn\_pullup = 1 \*/;  
               output reg [16:0] P /\* synthesis syn\_pulldown = 1 \*/;

VHDL      attribute syn\_pullup of A : signal is true ;  
               attribute syn\_pulldown of P : signal is true ;

## **xc\_rloc**

### *Attribute*

Specifies the relative locations for component instances.

### **Description**

Specifies the relative locations for component instances. This attribute is used in conjunction with [xc\\_uset](#) and [xc\\_map](#). Use this attribute to specify locations for components grouped together using [xc\\_uset](#).

### **xc\_rloc Syntax**

|         |                                                                           |                                            |
|---------|---------------------------------------------------------------------------|--------------------------------------------|
| FDC     | <code>define_attribute {designName} xc_rloc {instanceName}</code>         | <a href="#">Constraints Editor Example</a> |
| Verilog | <code>object /* synthesis xc_rloc = "instanceName" */;</code>             | <a href="#">Verilog Example</a>            |
| VHDL    | <code>attribute xc_rloc of object : objectType is "instanceName" ;</code> | <a href="#">VHDL Example</a>               |

### **Constraints Editor Example**

|   | Enable                              | Object Type | Object | Attribute | Value  | Value Type | Description                                        |
|---|-------------------------------------|-------------|--------|-----------|--------|------------|----------------------------------------------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | i:x03  | xc_uset   | SET1   | string     | Assign group name for placement, use with xc_rloc  |
| 2 | <input checked="" type="checkbox"/> | <any>       | i:x03  | xc_rloc   | R0C0.f | string     | Relative placement specification, use with xc_uset |

FDC file examples:

```
define_attribute {x03} xc_uset {SET1}
define_attribute {x03} xc_rloc {R0C0.f}

define_attribute {x47} xc_uset {SET1}
define_attribute {x47} xc_rloc {R0C0.g}

define_attribute {zz} xc_uset {SET1}
define_attribute {zz} xc_rloc {R0C0.h}
```

## Verilog Example

```

module fmap_xor4 (z, a, b, c, d) /* synthesis xc_map=fmap */;
output z;
input a, b, c, d;
assign z = a ^ b ^ c ^ d;
endmodule

module hmap_xor3 (z, a, b, c) /* synthesis xc_map=hmap */;
output z;
input a, b, c;
assign z = a ^ b ^ c;
endmodule

module clb_xor9 (z, a);
output z;
input [8:0] a;
wire z03, z47;
fmap_xor4 x03 /* synthesis xc_uset="SET1" xc_rloc="R0C0.f" */
(z03, a[0], a[1], a[2], a[3]);
hmap_xor3 zz /* synthesis xc_uset="SET1" xc_rloc="R0C0.h" */
(z, z47, a[4], a[5]);
fmap_xor4 x47 /* synthesis xc_uset="SET1" xc_rloc="R0C0.g" */
(z47, z03, a[5], a[6], a[7]);
endmodule

module xor9top (z, a);
output z;
input [8:0] a;
clb_xor9 x (z, a);
endmodule

```

## VHDL Example

```

--hmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity hmap_xor4 is
    port (a, b, c, d : in std_logic;
          z : out std_logic );
end hmap_xor4;

```

```
architecture rtl of hmap_xor4 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "hmap";
begin
    z <= a xor b xor c xor d;
end rtl;

-- fmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity fmap_xor3 is
    port (a, b, c : in std_logic;
          z : out std_logic );
end fmap_xor3;

architecture rtl of fmap_xor3 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "fmap";
begin
    z <= (a and b) xor c;
end rtl;

-- hmap & fmap instantiations
library IEEE;
use IEEE.std_logic_1164.all;

entity clb_xor9 is
    port (a : in std_logic_vector(8 downto 0);
          z : out std_logic );
end clb_xor9;

architecture rtl of clb_xor9 is
    signal z03, z47 : std_logic;
component fmap_xor3
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          z : out std_logic );
end component;
component hmap_xor4
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          d : in std_logic;
          z : out std_logic );
end component;
```

```

attribute xc_uset : string;
attribute xc_rloc : string;
attribute xc_uset of x03 : label is "SET1";
attribute xc_rloc of x03 : label is "R0C0.f";
attribute xc_uset of x47 : label is "SET1";
attribute xc_rloc of x47 : label is "R0C0.g";
attribute xc_uset of zz : label is "SET1";
attribute xc_rloc of zz : label is "R0C0.h";
begin
x03 : hmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
x47 : hmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
zz : fmap_xor3 port map(z03, z47, a(8), z);
end rtl;

--Top level entity definition
library IEEE;
use IEEE.std_logic_1164.all;

entity top is
    port (a : in std_logic_vector(8 downto 0);
          z : out std_logic );
end top;

architecture rtl of top is
component clb_xor9
    port (a : in std_logic_vector (8 downto 0);
          z : out std_logic );
end component;
begin
    U1: clb_xor9 port map (a, z);
end rtl;

```

## Effect of Using xc\_rloc

After applying the attribute, the output constraint is forward-annotated to the output EDIF netlist:

```
(instance x03 (viewRef PRIM (cellRef LUT4 (libraryRef VIRTEX)))
  (property RLOC (string "R0C0.f"))
  (property HU_SET (string "x$SET1"))
  (property INIT (string "16'h6996"))
```

## **xc\_slow**

### *Attribute*

Specifies that the transition time of the driver of an output port is slow (the default).

This attribute is the same as the *syn\_slow* attribute. See [\*syn\\_slow, on page 805\*](#).

## xc\_use\_keep\_hierarchy

### *Attribute*

Preserves the hierarchy of the marked module for diagnostic purposes and timing simulation.

### Description

This attribute lets you preserve the hierarchy of a marked module for diagnostic purposes and timing simulation. The marked modules are not flattened in the place-and-route tool, and their hierarchy is preserved.

By default, the software does not add the Xilinx attribute (KEEP\_HIERARCHY (string "TRUE")) in the EDIF file for all marked compile points and hierarchical blocks. When the attribute is active, the software adds the Xilinx attribute KEEP\_HIERARCHY=TRUE in the EDIF file for all marked compile points and hierarchical blocks.

When using the xc\_use\_keep\_hierarchy attribute:

- Make sure that all blocks with this attribute are registered.
- You must use xc\_use\_keep\_hierarchy together with the syn\_hier attribute and a value of "hard" ([syn\\_hier, on page 642](#)) to preserve the hierarchy of the module.
- Use this attribute in conjunction with the xc\_area\_group compile point attribute ([xc\\_area\\_group, on page 872](#)).
- When this attribute is applied, the synthesis tool does not do boundary optimizations, so use the attribute with caution, because it can compromise design performance.
- When specified as a global attribute, all compile points in the design are marked (KEEP\_HIERARCHY=TRUE) as independent hierarchical modules.

## xc\_use\_keep\_hierarchy Syntax

|         |                                                                                   |                                         |
|---------|-----------------------------------------------------------------------------------|-----------------------------------------|
| FDC     | <code>define_global_attribute {xc_use_keep_hierarchy} {1 0}</code>                | <a href="#">Constraint File Example</a> |
| Verilog | <code>object /* synthesis xc_use_keep_hierarchy = 1 0 */</code>                   | <a href="#">Verilog Example</a>         |
| VHDL    | <code>attribute xc_use_keep_hierarchy of all : architecture is true false;</code> | <a href="#">VHDL Example</a>            |

## Constraint File Example

The following Constraints Editor example shows that the `xc_use_keep_hierarchy` attribute must be used in conjunction with the `syn_hier` attribute.

|   | Enable                              | Object Type | Object       | Attribute                          | Value | Value Type | Description                           |
|---|-------------------------------------|-------------|--------------|------------------------------------|-------|------------|---------------------------------------|
| 1 | <input checked="" type="checkbox"/> | view        | <Global>     | <code>xc_use_keep_hierarchy</code> | 1     | boolean    | Preserves the hierarchy of the marked |
| 2 | <input checked="" type="checkbox"/> | view        | v:work.mod10 | <code>syn_hier</code>              | hard  | string     | Control hierarchy flattening          |

The Tcl equivalent commands for these attributes are shown below:

```
define_global_attribute {xc_use_keep_hierarchy} {1}
define_attribute {v:work.mod10} {syn_hier} {hard}
```

## Verilog Example

```
module keep_hier (a,b,c,clk,out1,out2)
    /* xc_use_keep_hierarchy= 1 */;
    input a,b,c,clk;
    output reg out1,out2;
    mod10 inst1 (.clk(clk),.c(c),.d(out2));

    always @ (posedge clk)
        out1 <= a + b;
    endmodule

module mod10 (clk,c,d) /* synthesis syn_hier = "hard" */;
    input c,clk;
    output reg d;

    always @ (posedge clk)
        d <= c;
    endmodule
```

## VHDL Example

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

entity test is
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          clk : in std_logic;
          out1 : out std_logic;
          out2 : out std_logic );
end test;

architecture arch of test is
attribute syn_use_keep_hierarchy : boolean;
attribute syn_use_keep_hierarchy of all : architecture is true;
component test1
    port (c : in std_logic;
          clk : in std_logic;
          d : out std_logic );
end component;
begin
inst1 : test1 port map (c =>c, d=>out2,clk=>clk);
process (clk)
begin
    if (clk'event and clk ='1')then
        out1 <= a and b;
    end if;
end process;
end arch;

library IEEE;
use ieee.std_logic_1164.all;

entity test1 is
    port (c : in std_logic;
          clk : in std_logic;
          d : out std_logic );
end test1;

architecture asrch of test1 is
attribute syn_hier : string;
attribute syn_hier of all : architecture is "hard";
begin
process (clk)
begin
```

```

        if (clk'event and clk ='1') then
            d <= c;
        end if;
    end process;
end arch;

```

## Effects of Using xc\_use\_keep\_hierarchy

This example shows the contents of the EDIF file before applying the xc\_use\_keep\_hierarchy attribute.

**Verilog**

```

module keep_hier (a, b, c, clk, out1,out2)
/* synthesis xc_use_keep_hierarchy = 0 */;
module mod10 (clk, c, d) /*synthesis syn_hier = "hard";

```

**VHDL**

```

attribute xc_use_keep_hierarchy of all : architecture is false;
attribute syn_hier of all : architecture is "hard";

```

```

(library work
(edinLevel 0)
(technology (numberDefinition ))
(cell hier (cellType GENERIC)
  (view verilog (viewType NETLIST)
    (interface
      (port clk (direction INPUT))
      (port c (direction INPUT))
      (port d (direction OUTPUT))
    )
    (contents
      (instance d (viewRef PRIM (cellRef FD (libraryRef UNILIB)))
      )
      (net clk (joined
        (portRef clk)
        (portRef C (instanceRef d))
      ))
      (net c (joined
        (portRef c)
        (portRef D (instanceRef d))
      ))
      (net (rename d0 "d") (joined
        (portRef Q (instanceRef d))
        (portRef d)
      ))
    )
  )
)

```

This example shows the contents of the EDIF file after applying the xc\_use\_keep\_hierarchy attribute.

Verilog

```
module keep_hier (a, b, c, clk, out1, out2);
/* synthesis xc_use_keep_hierarchy = 1 */;
module mod10 (clk, c, d) /*synthesis syn_hier = "hard";
```

VHDL

```
attribute xc_use_keep_hierarchy of all : architecture is true;
attribute syn_hier of all : architecture is "hard";
```

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell hier (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port clk (direction INPUT))
        (port c (direction INPUT))
        (port d (direction OUTPUT))
      )
      (contents
        (instance d (viewRef PRIM (cellRef FD (libraryRef UNILIB)))
        )
        (net clk (joined
          (portRef clk)
          (portRef C (instanceRef d))
        )))
        (net c (joined
          (portRef c)
          (portRef D (instanceRef d))
        )))
        (net (rename dZ0 "d") (joined
          (portRef Q (instanceRef d))
          (portRef d)
        )))
      )
      (property KEEPHIERARCHY (string "TRUE"))
    )
  )
```

## **xc\_use\_rpms**

### *Attribute*

Specifies how to handle relationally placed macros (RPMs) during global placement and physical synthesis.

### **Description**

Specifies how to handle relationally placed macros (RPMs) during global placement and physical synthesis. Xilinx IP cores may contain RPMs that can have illegal placements that would cause the synthesis tool to error out and generate a message about a workaround for the placement error.

### **xc\_use\_rpms Syntax**

FDC File **define\_global\_attribute {xc\_use\_rpms} {0|1|2}**

[Constraints Editor Example](#)

In the above syntax::

| <b>Value</b>   | <b>Description</b>                                                                                                                                                                                                                                                                        |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0              | Removes all RPMs for global placement and physical synthesis. With this value, RPMs can be moved during global placement and physical synthesis. This eliminates the chance of physical synthesis errors related to illegal placement, however, QoR advantages for using RPMs are missed. |
| 1<br>(Default) | Maintains RPMs for global placement and physical synthesis. With this value, the QoR advantages of using RPMs are maintained, however, if any of the RPMs are illegally placed, physical synthesis will error out.                                                                        |
| 2              | Honors RPMs during global placement and removes them for physical synthesis. With this value, some QoR advantages can be maintained for global placement, however, the tool has the option to optimize placement during physical synthesis.                                               |

### **Constraints Editor Example**

|   | Enable                              | Object Type | Object   | Attribute   | Value | Value Type | Description                                                                     | Comment |
|---|-------------------------------------|-------------|----------|-------------|-------|------------|---------------------------------------------------------------------------------|---------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Global> | xc_use_rpms | 1     | boolean    | Allows Synplify Premier either to ignore or honour the Relatively Placed Macros |         |
| 2 |                                     |             |          |             |       |            |                                                                                 |         |

An FDC file entry example is

```
define_global_attribute xc_use_rpms {2}
```

## Effects of Using xc\_use\_rpms

This example occurs before the xc\_use\_rpms attribute is applied, which shows the content of the log file with the default value 1.

```
#####
# Premap Report

Adding property xc_use_rpms, value 1 to view:work.test(verilog)

Adding property RLOC, value 'XOY0', to instance u1
Adding property RLOC, value 'XOY1', to instance u2
Adding property RLOC, value 'XOY4', to instance u5

#####
# Map & Optimize Report

'

#E: FX779 |Error during Initial Placement with code=|. Please examine the Initial Placement report test.bld, test_map.map and test.par in 'xplace'
#####


```

The following example occurs after the xc\_use\_rpms attribute is applied, which shows the content of the log file with a value 0.

```
#####
[Premap Report

Adding property xc_use_rpms, value 0 to view:work.test(verilog)

Adding property RLOC, value 'XOY0', to instance u1
Adding property RLOC, value 'XOY1', to instance u2
Adding property RLOC, value 'XOY4', to instance u5

#####
[Map & Optimize Report

@N: FX391 |Deleting all RLOCs before initial placement because xc_use_rpms=0

@N: FX356 |Ignoring RPMs in Physical Synthesis
```

This last example shows the content of the log file with the value 2.

```
#####
[Premap Report

Adding property xc_use_rpms, value 2 to view:work.test(verilog)

Adding property RLOC, value 'XOY0', to instance u1
Adding property RLOC, value 'XOY1', to instance u2
Adding property RLOC, value 'XOY4', to instance u5
```

## **xc\_use\_timespec\_for\_io**

### *Attribute*

Uses the TIMESPEC FROM ... TO command for writing flop-to-out and in-to-flop I/O constraints to the NCF constraint file.

### **Description**

By default, the Xilinx OFFSET keyword is used for writing flop-to-out and in-to-flop I/O constraints to the NCF constraint file for forward annotation to placement and routing. To use the TIMESPEC FROM ... TO command instead, you globally assign the `xc_use_timespec_for_io` attribute a value of 1. The default value of the attribute is 0 (use OFFSET).

When TIMESPEC FROM ... TO is used, only the data delay is taken into account by the Xilinx place-and-route tool; when OFFSET is used, the clock arrival time and clock delay are also taken into account. For more information on the behavior of these keywords, refer to the appropriate Xilinx documentation.

---

**Note:** When the `xc_use_timespec_for_io` attribute is enabled (1), there is no relaxation of constraints that are forward-annotated to the NCF file.

---

### **xc\_use\_timespec\_for\_io Syntax**

|          |                                                                                         |                                            |
|----------|-----------------------------------------------------------------------------------------|--------------------------------------------|
| FDC File | <code>define_global_attribute xc_use_timespec_for_io {0 1}</code>                       | <a href="#">Constraints Editor Example</a> |
| Verilog  | <code>object /* synthesis xc_use_timespec_for_io = 0 1 */;</code>                       | <a href="#">Verilog Example</a>            |
| VHDL     | <code>attribute xc_use_timespec_for_io of object :<br/>objectType is false true;</code> | <a href="#">VHDL Example</a>               |

## Constraints Editor Example

|   | Enable                              | Object Type | Object   | Attribute              | Value | Value Type | Description                                                         |
|---|-------------------------------------|-------------|----------|------------------------|-------|------------|---------------------------------------------------------------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | <Global> | xc_use_timespec_for_io | 1     | boolean    | Enable use of from-to timespec instead of offset for I/O constraint |

FDC file entry example:

```
define_global_attribute xc_use_timespec_for_io {1}
```

## Verilog Example

```
module fmap_xor4 (z, a, b, c, d) /* synthesis xc_map=fmap */;
  output z;
  input a, b, c, d;
  assign z = a ^ b ^ c ^ d;
endmodule

module hmap_xor3 (z, a, b, c) /* synthesis xc_map=hmap */;
  output z;
  input a, b, c;
  assign z = a ^ b ^ c;
endmodule

module clb_xor9 (z, a);
  output z;
  input [8:0] a;
  wire z03, z47;
  fmap_xor4 x03 /* synthesis xc_uset="SET1" xc_rloc="R0C0.f" */
    (z03, a[0], a[1], a[2], a[3]);
  hmap_xor3 zz /* synthesis xc_uset="SET1" xc_rloc="R0C0.h" */
    (z, z47, a[4], a[5]);
  fmap_xor4 x47 /* synthesis xc_uset="SET1" xc_rloc="R0C0.g" */
    (z47, z03, a[5], a[6], a[7]);
endmodule

module xor9top (z, a);
  output z;
  input [8:0] a;
  clb_xor9 x (z, a);
endmodule
```

## VHDL Example

```
--hmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity hmap_xor4 is
    port (a, b, c, d : in std_logic;
          z : out std_logic );
end hmap_xor4;

architecture rtl of hmap_xor4 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "hmap";
begin
    z <= a xor b xor c xor d;
end rtl;

-- fmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity fmap_xor3 is
    port (a, b, c : in std_logic;
          z : out std_logic );
end fmap_xor3;

architecture rtl of fmap_xor3 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "fmap";
begin
    z <= (a and b) xor c;
end rtl;

-- hmap & fmap instantiations
library IEEE;
use IEEE.std_logic_1164.all;

entity clb_xor9 is
    port (a : in std_logic_vector(8 downto 0);
          z : out std_logic );
end clb_xor9;
```

```
architecture rtl of clb_xor9 is
signal z03, z47 : std_logic;
component fmap_xor3
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          z : out std_logic );
end component;
component hmap_xor4
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          d : in std_logic;
          z : out std_logic );
end component;
attribute xc_uset : string;
attribute xc_rloc : string;
attribute xc_uset of x03 : label is "SET1";
attribute xc_rloc of x03 : label is "R0C0.f";
attribute xc_uset of x47 : label is "SET1";
attribute xc_rloc of x47 : label is "R0C0.g";
attribute xc_uset of zz : label is "SET1";
attribute xc_rloc of zz : label is "R0C0.h";
begin
x03 : hmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
x47 : hmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
zz : fmap_xor3 port map(z03, z47, a(8), z);
end rtl;

--Top level entity definition
library IEEE;
use IEEE.std_logic_1164.all;

entity top is
    port (a : in std_logic_vector(8 downto 0);
          z : out std_logic );
end top;

architecture rtl of top is
component clb_xor9
    port (a : in std_logic_vector (8 downto 0);
          z : out std_logic );
end component;
begin
U1: clb_xor9 port map (a, z);
end rtl;
```

## **xc\_uset**

### *Attribute*

Assigns a group name to component instances.

### **Description**

Assigns a group name to component instances. This attribute is used with the [xc\\_rloc](#) and [xc\\_map](#) attributes to specify a relative location to a group of components instances.

### **xc\_uset Syntax**

|          |                                                                        |                                            |
|----------|------------------------------------------------------------------------|--------------------------------------------|
| FDC File | <code>define_attribute {instanceName} xc_uset {groupName}</code>       | <a href="#">Constraints Editor Example</a> |
| Verilog  | <code>object /* synthesis xc_uset = "groupName" */;</code>             | <a href="#">Verilog Example</a>            |
| VHDL     | <code>attribute xc_uset of object : objectType is "groupName" ;</code> | <a href="#">VHDL Example</a>               |

### **Constraints Editor Example**

|   | Enable                              | Object Type | Object | Attribute | Value  | Value Type | Description                                        |
|---|-------------------------------------|-------------|--------|-----------|--------|------------|----------------------------------------------------|
| 1 | <input checked="" type="checkbox"/> | <any>       | i:x03  | xc_uset   | SET1   | string     | Assign group name for placement, use with xc_rloc  |
| 2 | <input checked="" type="checkbox"/> | <any>       | i:x03  | xc_rloc   | ROCO.f | string     | Relative placement specification, use with xc_uset |

FDC file entry examples:

```
define_attribute {x03} xc_uset {SET1};
define_attribute {x03} xc_rloc {ROCO.f};

define_attribute {x44} xc_uset {SET1};
define_attribute {x44} xc_rloc {ROCO.g};

define_attribute {zz} xc_uset {SET1};
define_attribute {zz} xc_rloc {ROCO.h};
```

## Verilog Example

```
module clb_xor9(z, a);
  output z;
  input [8:0] a;
  wire z03, z47;
  fmap_xor4 x03 /* synthesis xc_uset = "SET1" xc_rloc = "R0C0.f" */
    (z03, a[0], a[1], a[2], a[3]);
  fmap_xor4 x47 /* synthesis xc_uset = "SET1" xc_rloc = "R0C0.g" */
    (z47, a[4], a[5], a[6], a[7]);
  hmap_xor3 zz /* synthesis xc_uset = "SET1" xc_rloc = "R0C0.h" */
    (z, z03, z47, a[8]);
endmodule
```

## VHDL Example

```
library synplify;

architecture rtl of clb_xor9 is
  signal z03, z47 : std_logic;
  component hmap_xor3
    port (a, b, c : in std_logic;
          z : out std_logic );
  end component;
  component fmap_xor4
    port (a, b, c : in std_logic;
          z : out std_logic );
  end component;
  attribute xc_uset : string;
  attribute xc_rloc : string;
  attribute xc_uset : string;
  attribute xc_uset of x03 : label is "SET1";
  attribute xc_rloc of x03 : label is "R0C0.f";
  attribute xc_uset of x47 : label is "SET1";
  attribute xc_rloc of x47 : label is "R0C0.g";
  attribute xc_uset of zz : label is "SET1";
  attribute xc_rloc of zz : label is "R0C0.h";

  begin
    x03 : fmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
    x47 : fmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
    zz : hmap_xor3 port map(z03, z47, a(8), z);
  end rtl;
```

# Index

---

## Symbols

.cdc file  
  examples 466  
.edf file 270  
.ndf file 270  
`ifdef 425

## A

adders  
  wide. *See* wide adders/subtractors.  
AES 404  
AES Rijndael 404  
all keyword, VHDL 2008 459

custom 612  
effects of retiming 309  
for FSMs 255  
inferring RAM 209  
pipelining 304  
specifying, overview of methods 462  
syn\_assign\_to\_region 513  
syn\_clock\_priority 543  
syn\_direct\_reset 568  
syntax, Verilog 427  
VHDL package 463  
Attributes panel, SCOPE 33  
Attributes panel, SCOPE  
  spreadsheet 469  
automatic compile points  
  *See also* compile points  
  constraints 328  
  optimizing 331  
  optimizing with syn\_hier 327  
  preventing generation 331  
  setting up 331  
  with incremental synthesis 332

## B

backslash  
  escaping dot wildcard in Find  
  command 123, 177  
bind statement 436  
bit files  
  encrypting 398  
bitstream. *See* bit files  
bit-string literals 452  
black box directives  
  black\_box\_pad\_pin 475  
  black\_box\_tri\_pins 479  
  syn\_black\_box 526  
  syn\_isclock 664  
  syn\_resources 780  
  syn\_tco 819  
  syn\_tpd 824

- 
- syn\_tristate** 832
  - black box instantiation**
    - true dual-port asymmetric RAM 220
  - black boxes** 257
    - adding timing constraints 264
    - directives. *See* black box directives
    - EDIF naming consistency 267
    - for IP cores 270
    - functional description for simulation (Verilog) 260
    - functional description for simulation (VHDL) 262
    - instantiating in Verilog 260
    - instantiating in VHDL 261
    - instantiating, Verilog 423
    - internal startup box (Xilinx) 891
    - pin attributes 267
    - source code directives 526
    - specifying timing information for Xilinx cores 270
    - syn\_gatedclk\_clock\_en** directive 630
    - timing directives 824
    - Verilog 423
  - black\_box** compile point 326
  - black\_box\_pad\_pin** directive 475
  - black\_box\_tri\_pins** directive 479
  - block RAM**
    - inferring 211
    - modes 208
    - types 208
  - BRAM**
    - encrypting bit files 398
  - browsers** 166
  - buffering**
    - controlling 299
  - buffers**
    - clock. *See* clock buffers
    - global 888
    - global. *See* global buffers
    - output 885
  - BUFG**
    - for fanouts 300
  - BUFGDLL** 285
  - BUFGMUX\_1** inference 284
  - BUFGMUX/BUFGMUX\_1** inference 284
  - BUFR** clock buffers 286
  - buses**
    - INIT values for bits 244
    - RLOC values for bits 290
  - byte-enable RAM inference** 226
  - byte-wide write enable RAM.** *See* byte-enable RAM
- C**
- c\_diff** command (collections) 26
  - c\_diff** command, examples 79
  - c\_intersect** command (collections) 26
  - c\_intersect** command, examples 79
  - c\_list** command
    - different from **c\_print** 81
    - example 83
    - using 82
  - c\_print** command
    - different from **c\_list** 81
    - using 82
  - c\_print** command (collections) 26
  - c\_sub** command (collections) 26
  - c\_symdiff** command (collections) 26
  - c\_symdiff** command, examples 80
  - c\_union** command (collections) 26
  - c\_union** command, examples 79
  - case sensitivity**
    - Find command (Tcl) 69
  - case statement**
    - default 487
  - cdc file syntax** 466
  - clock and path constraints**
    - setting 39
  - clock buffers** 285
    - assigning resources 704
  - clock constraints**
    - setting 39, 41, 42
  - clock DLLs** 285
  - clock enables**
    - inferring with **syn\_direct\_enable** 564
    - net assignment 564
  - clock groups**
    - effect on false path constraints 63
  - clock groups, SCOPE** 41

---

**clock priority**  
 syn\_clock\_priority 543  
**clocks**  
 implicit false path 63  
 on black boxes 664  
**Clocks** panel, SCOPE 21  
**code**  
 ignoring with pragma translate off/on 493  
**collection commands**  
 SCOPE 25  
**collections**  
 adding attributes to 77  
 adding objects 78  
 concatenating 78  
 constraints 76  
 copying 82  
 creating from common objects 78  
 creating from other collections 76  
 creating in Constraints Editor 75  
 creating in Tcl 77  
 definition 74  
 diffing 78  
 highlighting in HDL Analyst views 81  
 listing objects 82  
 listing objects and properties 81  
 listing objects in a file 82  
 listing objects in columnar format 81  
 listing objects with c\_list 81  
 special characters 80  
 Tcl window and Constraints Editor comparison 74  
 using Tcl expand command 72  
 using Tcl find command 71  
 viewing 80  
**Collections** panel, SCOPE 24  
**compile point types**  
 black\_box 326  
 hard 323  
 locked 324  
 locked,partition 325  
**compile points**  
 allowed resources 500  
 analyzing results 336  
 child 322  
 combining with distributed processing 320  
 combining with multiprocessing 320  
 constraints for forward-annotation 336  
 constraints, internal 336  
 feature summary 327  
 for blocks with multiple instantiations 321  
 nested 322  
 optimization 334  
 parent 322  
 preserving hierarchy for Xilinx Place and Route 911  
 resynthesis 334  
 synthesis process 333  
 type 320  
 types 322  
 using syn\_allowed\_resources attribute 332  
 using with incremental synthesis 321  
**Compile Points** panel, SCOPE 35  
**compile-point synthesis**  
 interface logic models 328  
**compiler**  
 loop iteration, loop\_limit 489  
 loop iteration, syn\_loopleft 676  
**congestion**  
 syn\_packer\_effort\_level 724  
**constant function**  
 syntax restrictions 418  
**constraint files**  
 applying to a collection 76  
 compile point 336  
 editing 53  
 effects of retiming 309  
**constraints**  
 specifying through points 59, 60  
 types 20  
**Constraints Editor**  
 collections compared to Tcl script window 74  
 drag and drop 53  
 editing operations 54  
 multicycle paths 62  
 setting constraints (FDC) 16  
 specifying constraints 20  
 using 16  
 using TCL View 51  
**Constraints Editor panels**  
 entering and editing constraints 20  
**Constraints Editor TCL View**  
 using 51

constraints. *See* timing constraints 328

context  
for object in filtered view 193

context declarations  
VHDL 2008 456

CoreGen 270

cores, instantiating in Xilinx designs 270

critical paths  
using -route 296

crossprobing 183  
compiled view 184  
filtering text objects for 188  
from text files 186  
Hierarchy Browser 124, 183  
mapped view 184  
new HDL Analyst views 125  
paths 186  
schematic views 124  
Text Editor view 184  
text file example 186  
Verilog file 184  
VHDL file 184

current level  
expanding logic from pin 197  
searching current level and below 174

custom attributes 612

**D**

data block 371

data encryption standard (DES) 404

data key 371

DCM clock priority 543

default propagation 446

define\_attribute 471  
syntax 470

define\_false\_path  
using with syn\_keep 667, 729

define\_global\_attribute  
summary 471  
syntax 470

define\_input\_delay  
forward-annotation with syn\_forward\_io\_constraints 627

define\_multicycle\_path  
using with syn\_keep 667, 729

define\_output\_delay  
forward-annotation with syn\_forward\_io\_constraints 627

defining I/O standards 34

Delay Paths panel, SCOPE 31

delays  
forward annotating 627  
forward-annotating with syn\_forward\_io\_constraints 627

DES 404

design guidelines 292

design hierarchy  
viewing 132, 190

design hierarchy, automatic compile points 327

design size  
amount displayed on a sheet 164

design views  
moving between views 102, 164

DesignWare 340  
building block IP 341  
digital cores 342  
functions 341  
importing cores 407

diff\_term attribute 483

directives  
adding 462  
adding black box constraints 264  
adding in Verilog 465  
adding in VHDL 463  
black box 264  
FSM 255  
syntax, Verilog 427

dissolving instances for flattening hierarchy 203

distributed processing  
need for compile points 320

distributed RAM inference 216

dot wildcard  
Find command 123, 177

drivers  
preserving duplicates with syn\_keep 311  
selecting 199

DSP (Xilinx) 579, 580

DSP48 (Xilinx)  
 converting asynchronous reset  
 registers [536, 543, 760](#)

## E

edf file. *See* edif file  
**EDIF**  
 structural, for Xilinx IP cores [270](#)  
**EDIF** and automatic compile points [327](#)  
**edif** file  
 bus name styles [591](#)  
 character case [594](#)  
 scalar and array ports [701](#)  
 syn\_noarrayports attribute [701](#)  
**else/elsif** clauses  
 VHDL 2008 [458](#)  
**encoding**  
 state machine  
 guidelines  
**Verilog** [421](#)  
**encryption**  
 asymmetric [371](#)  
 bit files [398](#)  
 methodologies [369](#)  
 partial (IEEE 1735) [392](#)  
 symmetric [371](#)  
 synenc [407](#)  
 encryption algorithms [371](#)  
**encryptip** output constraints [403](#)  
**encryptip** output method  
 effect on output netlists [403](#)  
**encryptIP** script [400](#)  
 command-line arguments [404](#)  
 controlling output [402](#)  
 encrypting IP [400](#)  
 output methods [402](#)  
 syntax [404](#)  
**encryptP1735** script [388, 404](#)  
 command-line arguments [388](#)  
 public keys repository file [389](#)  
 syntax [388](#)  
 use models [391](#)  
**encryptP1735.pl** script [380](#)  
**enhanced optimization**  
 effect on syn\_macro and  
 syn\_user\_instance [681](#)

enumerated types  
 syn\_enum\_encoding directive [611](#)  
**Expand** command  
 connection logic [199](#)  
 pin and net logic [140, 196](#)  
 using [197](#)  
**expand** command (Tcl). *See* Tcl expand command  
**Expand Inwards** command  
 using [141, 197](#)  
**Expand Paths** command  
 different from Isolate Paths [199](#)  
**Expand to Register/Port** command  
 using [197](#)  
**expanding**  
 connections [199](#)  
 pin and net logic [140, 196](#)

## F

**false paths**  
 impact of clock group assignments [63](#)  
 ports [63](#)  
 registers [63](#)  
 setting constraints [63](#)  
**fanout limits**  
 overriding default [687](#)  
 syn\_maxfan attribute [687](#)  
**fanouts**  
 buffering vs replication [299](#)  
 hard limits [298](#)  
 soft global limit [297](#)  
 soft module-level limit [298](#)  
 using syn\_keep for replication [312](#)  
 using syn\_maxfan [297](#)  
**fdc** file  
 adding black box timing  
 constraints [266](#)  
**files**  
 .edf. *See* edif file  
 rom.info [169](#)  
**Filter Schematic** command,  
 using [139, 194](#)  
**Filter Schematic icon**, using [139, 194](#)  
**filtering** [138, 194](#)  
 advantages over flattening [138, 194](#)  
 using to restrict search [174](#)

Find command  
**174**  
 browsing with [174](#)  
 hierarchical search [175](#)  
 long names [174](#)  
 reading long names [177](#)  
 search scope, effect of [177](#)  
 search scope, setting [175](#)  
 searching the mapped database [176](#)  
 setting limit for results [176](#)  
 using wildcards [123, 177](#)  
 wildcard examples [179](#)

find command (Tcl)  
*See* Tcl find command

fix gated clocks  
`syn_gatedclk_clock_en` directive [630](#)

Flatten Current Schematic command  
 transparent instances [201](#)

Flatten Schematic command  
 using [201](#)

flattening [150, 201](#)  
*See also* dissolving  
 compared to filtering [138, 194](#)  
 dissolving instances [150, 203](#)  
 hidden instances [202](#)  
 transparent instances [201](#)  
 using `syn_hier` [315](#)  
 using `syn_netlist_hierarchy` [315](#)

force statement [436](#)

forward annotation  
 frequency constraints in Xilinx [293](#)

forward-annotation  
 compile point constraints [336](#)  
`syn_forward_io_constraints`  
 attribute [627](#)

from constraints [57](#)

FSM encoding  
 user-defined [256](#)  
 using `syn_enum_encoding` [256](#)

FSM view  
 crossprobing from source file [186](#)

FSMs  
 attributes and directives [255](#)  
 defining in Verilog [253](#)  
 defining in VHDL [254](#)  
 definition [253](#)  
`syn_encoding` attribute [603](#)

`full_case` directive [485](#)  
 functions  
 VHDL 2008 predefined [453](#)

## G

Generated Clocks panel, SCOPE [23](#)  
 generating [245](#)  
 generics  
 VHDL 2008 packages [455](#)  
 global buffers [888](#)  
 defining [636](#)  
`xc_global_buffers` attribute [888](#)  
 global comments  
 initializing Xilinx RAM [239](#)

## H

HDL Analyst  
*See also* RTL view, Technology view  
 crossprobing [124, 183](#)  
 filtering schematics [138, 194](#)  
 Push/Pop mode [169, 170](#)  
 traversing hierarchy with mouse  
 strokes [167](#)  
 traversing hierarchy with Push/Pop  
 mode [110, 169](#)  
 using [132, 190](#)

HDL Analyst tool  
 deselecting objects [97, 162](#)  
 selecting/deselecting objects [96, 161](#)

HDL Analyst views  
 highlighting collections [81](#)

hidden instances  
 consequences of saving [192](#)  
 flattening [202](#)  
 restricting search by hiding [175](#)  
 specifying [191](#)  
 status in other views [191](#)

hierarchical design  
 expanding logic from pins [140, 196](#)

hierarchical design, creating  
 Verilog [424](#)

hierarchical designs  
 using include files [425](#)

hierarchical instances  
 dissolving [150, 203](#)

- hiding. *See* hidden instances, Hide Instances command
- multiple sheets for internal logic 193
- pin name display 195
- viewing internal logic 134, 192
- hierarchical objects
  - pushing into with mouse stroke 109, 168
  - traversing with Push/Pop mode 110, 169
- hierarchical search 174
- hierarchy
  - flattening 150, 201
  - flattening with syn\_hier 642
  - preserving for Xilinx Place and Route 911
  - traversing 106, 166
  - Verilog 424
- hierarchy browser
  - controlling display 164
  - crossprobing from 124, 183
  - defined 106, 166
  - finding objects 113, 172
  - traversing hierarchy 166
- I**
- I/O buffers
  - inserting 659
  - specifying I/O standards 725
- I/O constraints
  - multiple on same port 29
- I/O insertion 318
  - VHDL manual (Xilinx) 282
- I/O locations
  - assigning automatically (Xilinx) 277
  - manually assigning (Xilinx) 281
- I/O packing
  - disabling with syn\_replicate 775
- I/O pads
  - specifying I/O standards 48
- I/O standards
  - specifying 48
- I/O Standards panel, SCOPE 34
- I/Os
  - constraining 47
  - inferring Xilinx buffers with syn\_diff\_io 560
- packing in Xilinx designs 274
- specifying pad type (Xilinx) 287
- Verilog black boxes 260
- VHDL black boxes 261
- IBUFDS
  - inference 285
- IBUFGDS
  - inference 285
- IEEE 1735
  - encrypting multiple files 384
- ieee library (VHDL) 443
- ignored language constructs (Verilog) 411
- ignored language constructs (VHDL) 441
- include files
  - hierarchical designs 425
- incremental synthesis
  - locked,partition compile points 325
  - need for compile points 321
- inference
  - BUFGMUX/BUFGMUX\_1 284
  - Xilinx BUFGMUX/BUFGMUX\_1 284
  - Xilinx I/O buffers 284
- INIT property
  - initializing Xilinx RAMs, Verilog 238
  - initializing Xilinx RAMs, VHDL 233
  - specifying with attributes 241
- INIT values
  - Xilinx registers 243
- init values
  - in RAMs 449
- initial value data file 235
- initializing RAM 230
- Input and output constraints
  - defining 47
- input constraints, setting 47
- Inputs/Outputs panel, SCOPE 27
- Instance Hierarchy tab 115
- instances
  - controlling optimization 863
  - preserving with syn\_noprune 311, 712
  - properties 89, 158
  - properties of pins 158
- instantiating black boxes (Verilog) 423
- instantiating components (VHDL) 445

- ILM *See* interface logic models  
 interface logic models 328
- IOBUFDS  
 inference 285
- IP 339  
 DesignWare 340  
 encryption-decryption flow 369  
 re-encryption 373  
 third-party 363  
 Vivado 352
- IP cores 270
- IP encryption  
 IEEE 1735 380
- IP encryption flow overview 368
- IP encryption scheme 374
- IP vendors  
 directory structure for package 376  
 encrypting IP 374  
 package file list for encrypted IP flow 376  
 packaging for evaluation 375  
 supplying vendor information 377
- IPs  
 encrypting 374  
 encryption flow 368
- Isolate Paths command  
 different from Expand Paths 199, 200

## K

- key block 371
- keywords  
 all (VHDL 2008) 459  
 SystemVerilog 437

## L

- language constructs (Verilog) 410
- language constructs (VHDL) 439, 442
- latches  
 SystemVerilog always\_latch 433
- libraries  
 macro, built-in 443  
 Verilog  
 macro 424
- VHDL  
 attributes and constraints 443

- IEEE, supported 439
- libraries (VHDL) 442
- library and package rules, VHDL 444
- library packages (VHDL), accessing 444
- library statement (VHDL) 444
- literal  
 bit string 452
- location constraints  
 RLOC\_ORIGIN 290  
 RLOCs with synthesis attribute 290  
 RLOCs with xc\_attributes 289
- log files  
 pipelining description 305  
 retiming report 307
- logic  
 expanding between objects 199  
 expanding from net 143, 197  
 expanding from pin 140, 196
- logic preservation  
 syn\_hier 315  
 syn\_keep for nets 311  
 syn\_keep for registers 311  
 syn\_noprune 311  
 syn\_preserve 311
- logical operators  
 VHDL 2008 452
- loop\_limit directive 489

## M

- macro libraries (Xilinx) 268
- macromodule 411
- macros  
 libraries 443  
 preserving with syn\_macro 678
- macros (Xilinx) 268
- map primitive 897
- memory map information (MMI file) 245
- memory usage  
 maximizing with HDL Analyst 205
- modules  
 partial IEEE 1735 encryption 392  
 renaming 773, 844
- mouse strokes  
 pushing/popping objects 108, 167

- 
- multicycle paths
    - setting constraints [40](#)
    - `syn_reference_clock` [771](#)
  - multidimensional array
    - syntax restrictions [418](#)
  - multipliers
    - pipelining restriction [303](#)
  - multipliers, pipelining [303](#)
  - multiprocessing
    - need for compile points [320](#)
  - multi-sheet schematics [162](#)
  - multisheet schematics
    - for nested internal logic [193](#)
    - searching just one sheet [174](#)
    - transparent instances [163](#)
- N**
- name spaces
    - technology view [176](#)
  - navigating among design views [102](#), [164](#)
  - nets
    - expanding logic from [143](#), [197](#)
    - preserving for probing with `syn_probe` [311](#)
    - preserving with `syn_keep` [311](#), [667](#), [729](#)
    - properties [89](#), [158](#)
    - selecting drivers [199](#)
  - new Hierarchy Browser [114](#)
  - New property [160](#)
  - NGCs and automatic compile points [327](#)
  - numeric\_bit IEEE package (VHDL) [443](#)
  - numeric\_std IEEE package (VHDL) [443](#)
- O**
- objects
    - finding on current sheet [174](#)
    - flagging by property [159](#)
    - selecting/deselecting [161](#)
  - OBUFDS
    - inference [285](#)
  - OBUFTDS
    - inference [285](#)
  - OFFSET I/O constraints (Xilinx) [919](#)
  - operators
- Verilog [411](#)
  - VHDL 2008 logical [452](#)
  - VHDL 2008 relational [452](#)
  - optimization
    - for area [293](#)
    - for timing [295](#)
    - logic preservation. *See* logic preservation.
    - mapper effort. *See* fast synthesis [296](#)
    - preserving hierarchy [315](#)
    - preserving objects [311](#)
    - tips for [292](#)
  - output buffers
    - selection [885](#)
  - output constraints, setting [47](#)
  - output drivers
    - slowing transition times [910](#)
    - transition time [881](#)
  - output files
    - .edf. *See* edif file
- P**
- packages
    - VHDL 2008 [454](#)
    - VHDL 2008 generics [455](#)
  - packages, VHDL [442](#)
  - pad types
    - industry standards [48](#)
    - Xilinx [901](#)
  - parallel\_case directive [491](#)
  - path constraints
    - false paths [63](#)
  - pathnames
    - using wildcards for long names (Find) [177](#)
  - paths
    - crossprobing [186](#)
    - tracing between objects [199](#)
    - tracing from net [143](#), [197](#)
    - tracing from pin [140](#), [196](#)
  - pattern matching
    - Find command (Tcl) [69](#)
  - pin locations
    - forward annotating [672](#)
    - specifying (Xilinx) [277](#)
  - pin names, displaying [195](#)

pins  
     expanding logic from 140, 196  
     properties 89, 158  
 pipelining  
     adding attribute 304  
     definition 303  
     multipliers 303  
     prerequisites 303  
     syn\_pipeline attribute 730, 732  
     whole design 304  
 port placement 894  
 ports  
     false path constraint 63  
     properties 89, 158  
 POS interface  
     using 60  
 pragma translate\_off directive 493  
 pragma translate\_on directive 493  
 predefined functions  
     VHDL 2008 453  
 predefined packages (VHDL) 443  
 preferences  
     displaying Hierarchy Browser 164  
     displaying labels 165  
     sheet size (UI) 164  
 primitives  
     pin name display 195  
     pushing into with mouse  
         stroke 109, 168  
     viewing internal hierarchy 191  
 priority encoding 491  
 private key 371  
 probes  
     inserting 743  
     retiming 310  
 Product of Sums interface. *See* POS interface  
 properties  
     displaying with tooltip 89, 158  
     finding objects with Tcl find -filter 70  
     viewing for individual objects 89, 158  
 public key 371  
 Push/Pop mode  
     HDL Analyst 167  
     using 108, 110, 167, 169

## Q

question mark wildcard, Find command 123, 177

## R

radiation effects. *See* high reliability  
 RAM  
     MMI file 245  
 RAM inference 208  
     using attributes 209  
 RAM MMI file  
     syn\_ram\_write\_mem attribute 760  
 RAMs  
     implementation styles 751  
     inferring block RAM 211  
     initializing 230  
     initializing values (Xilinx) 241  
     technology support 754  
 region clock buffers (BUFR) 286  
 register balancing. *See* retiming  
 register packing  
     *See also* syn\_useioff attribute 274  
     Xilinx 274  
 registers  
     false path constraint 63  
     INIT value 243  
     preserving with syn\_preserve 730, 738  
 Registers panel, SCOPE 30  
 relational operators  
     VHDL 2008 452  
 relative location  
     xc\_map (Xilinx) 897  
     xc\_rloc 906  
     xc\_uset 923  
 relative placement. *See* RLOCs  
 replication  
     controlling 299  
     disabling 775  
 resets  
     Verilog 419  
 resolving conflicting timing constraints 64  
 resource library (VHDL), creating 445  
 resource sharing

- 
- optimization technique 293
  - overriding option with `syn_sharing` 317
  - results example 317
  - `syn_sharing` directive 797
  - using 317
  - resynthesis
    - compile points 335
    - forcing with Resynthesize All 335
    - forcing with Update Compile Point Timing Data 335
  - retiming
    - effect on attributes and constraints 309
    - example 307
    - overview 305
    - probes 310
    - report 307
    - simulation behavior 310
    - `syn_allow_retimining` attribute 496
    - whole design 306
  - RLOC\_ORIGINS
    - specifying 290
  - RLOCs 289, 290
    - specifying with synthesis attribute 290
    - specifying with xc attributes 289
  - rom.info file 169
  - ROMs
    - pipelining 303
    - viewing data table 169
  - RTL view
    - crossprobing from Text Editor 186
    - filtering 138, 194
    - finding objects with Hierarchy Browser 113
    - flattening hierarchy 201
    - highlighting collections 81
    - opening 87
    - primitives
      - Verilog 419
  - rules
    - library and package, VHDL 444
  - runtime
    - using compile points to reduce 320
  - S**
  - Schematic Options 115
  - schematics
    - multisheet. *See* multisheet schematics
  - page size 164
  - selecting/deselecting objects 96, 161
  - SCOPE
    - adding attributes 467
    - assigning Xilinx pin locations 278
    - Attributes panel 33
    - clock groups 41
    - Clocks panel 21
    - Collections panel 24
    - Compile Points panel 35
    - defining compile points 333
    - Delay Paths panel 31
    - Generated Clocks panel 23
    - I/O Standards panel 34
    - Inputs/Outputs panel 27
    - pipelining attribute 304
    - Registers panel 30
    - specifying RLOCs 289, 290
    - TCL View 38
  - SCOPE spreadsheet
    - Attributes panel 469
  - search
    - browsing objects with the Find command 174
    - browsing with the Hierarchy Browser 113, 172
    - finding objects on current sheet 174
    - setting limit for results 176
    - setting scope 175
    - using the Find command in HDL Analyst views 174
  - search in Analyst
    - browsing objects with the Find command 118
  - See also* search
  - sequential logic
    - SystemVerilog
      - sequential logic 433
  - sequential optimization, preventing with `syn_preserve` 730, 738
  - sequential shift components *See* shift registers
  - set command
    - collections 82
  - set modules command (collections) 26
  - set modules\_copy command (collections) 26

---

**set\_option**  
 -automatic\_compile\_point 331  
**sets and resets (Verilog)** 419  
**sheet connectors**  
 navigating with 163  
**sheet size**  
 setting number of objects 164  
**shematic view**  
 setting preferences (New Anallyst) 103  
**shift register lookup table.** See **shift registers**  
**shift registers**  
 inferring 248  
 mapping 248  
 SRL16 primitives 248  
 Verilog 250  
 VHDL 249  
**Show Cell Interior option** 191  
**Show Context command**  
 different from Expand 193  
 using 193  
**SIMD mode**  
 using **syn\_dspstyle** attribute 586  
**simulation mismatches**  
 full\_case directive 488  
**simulation, effect of retiming** 310  
**Slow property** 160  
**source code**  
 adding pipelining attribute 304  
 defining FSMs 253  
 opening automatically to crossprobe 185  
 optimizing 292  
 specifying RLOCs 289, 290  
**source files**  
 adding to VHDL design library 442  
 crossprobing 186  
 state machine attributes 255  
**special characters**  
 Tcl collections 80  
**SRLs** See **shift registers**  
**standard IEEE package (VHDL)** 443  
**startup blocks**  
 Xilinx 891  
**state machines**  
 attributes 255  
 encoding  
     **syn\_encoding** attribute  
     **Verilog** 422  
 enumerated types 611  
 extracting 815  
 parameter and 'define comparison 254  
 Verilog 422, 423  
**state machines (Verilog)** 421  
**state values (FSM), Verilog** 423  
**std IEEE library (VHDL)** 443  
**std\_logic\_1164** IEEE package (VHDL) 443  
**std\_logic\_arith** IEEE package (VHDL) 443  
**std\_logic\_signed** IEEE package (VHDL) 443  
**std\_logic\_unsigned** IEEE package (VHDL) 443  
**structural designs, Verilog** 424  
**supported language constructs (Verilog)** 410  
**supported language constructs (VHDL)** 439  
**syn\_allow\_retiming**  
 using for retiming 306  
**syn\_allow\_retiming** attribute 496  
**syn\_allowed\_resources**  
 compile points 332  
**syn\_assign\_to\_region** attribute 513  
**syn\_assign\_to\_slr** attribute 515  
**syn\_async\_reg** attribute 517  
**syn\_auto\_insert\_bufg** attribute 520  
**syn\_black\_box** directive 526  
**syn\_clean\_reset** attribute 536  
**syn\_clock\_gmux\_proxy** attribute 539  
**syn\_clock\_priority** attribute 543  
**syn\_connect\_hrefs** directive 547  
**syn\_corrupt\_pd** attribute 555  
**syn\_cp\_use\_fast\_synthesis** attribute 559  
**syn\_diff\_io** attribute 560  
**syn\_direct\_enable** attribute 564  
**syn\_direct\_reset** attribute 568

---

**syn\_direct\_set** attribute 572  
**syn\_disable\_purifyclock** attribute 576  
**syn\_DSPstyle** attribute 579, 580  
 inferring wide adders/subtractors 251  
 using SIMD mode 586  
**syn\_EDIF\_bit\_format** attribute 270, 591  
**syn\_EDIF\_scalar\_format** attribute 270, 594  
**syn\_encoding**  
 compared with **syn\_enum\_encoding** directive 613  
 using with **enum\_encoding** 612  
**syn\_encoding** attribute 603  
 FSM encoding style  
 Verilog 422  
**syn\_enum\_encoding**  
 using with **enum\_encoding** 612  
**syn\_enum\_encoding** directive 611  
 compared with **syn\_encoding** attribute 613  
 FSM encoding 256  
**syn\_fast\_auto** attribute 616  
**syn\_force\_seq\_prim** directive 622  
**syn\_formal\_blackbox** directive 624  
**syn\_forward\_io\_constraints** attribute 627  
**syn\_gatedclk\_clock\_en** directive 630  
**syn\_gatedclk\_clock\_en\_polarity** directive 632  
**syn\_global\_buffers**  
 and **xc\_global\_buffers** 636  
**syn\_global\_buffers** attribute 636  
**syn\_hier**  
 using with automatic compile points 327  
**syn\_hier** attribute 642  
 and **xc\_use\_keep\_hierarchy** 911  
 controlling flattening 315  
 preserving hierarchy 315  
**syn\_implement** directive 652  
**syn\_insert\_buffer** attribute 652  
 BUFGMUX 284  
**syn\_insert\_pad** attribute 659  
**syn\_isclock**  
 black box clock pins 267  
**syn\_isclock** directive 664  
**syn\_keep**  
 compared with **syn\_preserve** and **syn\_noprune** directives 669  
 replicating redundant logic 312  
**syn\_keep** attribute  
 preserving nets 311  
 preserving shared registers 311  
**syn\_keep** directive 667  
 effect on buffering 299  
**syn\_loc** attribute 672  
**syn\_looplimit** directive 676  
**syn\_macro**  
 specifying encrypted IP as white box 402  
**syn\_macro** directive 678  
**syn\_map\_dffrs** attribute 683  
**syn\_maxfan** attribute 687  
 setting fanout limits 297  
 Xilinx buffers 300  
**syn\_multstyle** attribute 692  
**syn\_netlist\_hierarchy** attribute 695  
**syn\_no\_compile\_point** 331  
**syn\_no\_compile\_point** attribute 709  
**syn\_noarrayports** attribute 701  
**syn\_noclockbuf** attribute 704  
 using with fanout guides 687  
**syn\_noprune** directive 712  
 preserving instances 311  
**syn\_packer\_effort\_level** attribute 724  
**syn\_pad\_type** attribute 725  
**syn\_pipeline** attribute 304, 730, 732  
**syn\_pnr\_preserve\_regs** 736  
**syn\_preserve**  
 compared with **syn\_keep** and **syn\_noprune** 730, 738  
 effect on buffering 299  
 preserving power-on for retiming 306  
 preserving registers with INIT values 243  
**syn\_preserve** directive 738  
 preserving FSMs from optimization 255  
 preserving logic 311  
**syn\_probe** attribute 743

- preserving nets 311  
 syn\_ram\_write\_mem attribute 760  
 syn\_ramstyle attribute 751  
 syn\_reduce\_controlset\_size attribute 764  
 syn\_reference\_clock attribute 771  
     effect on multiple I/O constraints 30  
 syn\_rename\_module directive 773  
 syn\_replicate  
     using with fanout guides 687  
 syn\_replicate attribute 775  
     using buffering 299  
 syn\_resources attribute 780  
 syn\_ret\_lib\_cell\_type directive 786  
 syn\_ret\_type directive 784  
 syn\_romstyle attribute 787  
 syn\_rw\_conflict\_logic attribute 792  
 syn\_sharing directive 797  
     overriding default 317  
 syn\_shift\_resetphase 801  
 syn\_slow attribute 805  
 syn\_srl\_mindepth attribute 808  
 syn\_srlstyle attribute 811  
     mapping sequential shift components  
         to registers 248  
     setting shift register style 248  
 syn\_state\_machine directive 815  
 syn\_tco directive 819  
     adding black box constraints 264  
 syn\_tpd directive 824  
     adding black box constraints 264  
     black-box timing 835  
 syn\_tristate directive 832  
 syn\_tsu directive 835  
     adding black box constraints 264  
     black-box timing 835  
 syn\_unconnected\_inputs attribute 840  
 syn\_unique\_inst\_module directive 844  
 syn\_upf\_ret\_port\_type directive 846  
 syn\_use\_carry\_chain attribute 847  
 syn\_useenables attribute 847  
 syn\_useioff  
     preventing flops from moving during  
         retiming 306
- syn\_useioff attribute  
     packing registers (Xilinx) 274  
 syn\_user\_instance  
     comparison with syn\_macro 680  
 syn\_user\_instance attribute 863  
 synchronous sets and resets  
     Verilog 419  
 syncenc encryption 407  
 syntax restrictions  
     constant function 418  
     multidimensional array 418  
 synthesis  
     attributes and directives, Verilog 427  
     guidelines  
         Verilog 416  
 synthesis macro, Verilog 425  
 synthesis\_off directive 870  
 synthesis\_on directive 870  
 SystemVerilog  
     always\_ff 433  
     always\_latch 433  
     ignoring code with  
         synthesis\_off/on 870  
     keywords 437  
 SystemVerilog data types  
     assignment for syn\_keep 668

**T**

- Tcl commands  
     collections 25  
 Tcl expand  
     using 68  
 Tcl expand command  
     usage tips 72  
     using in Constraints Editor 75  
 Tcl find  
     filtering results by property 70  
     search patterns 68  
     using 68  
 Tcl find command  
     case sensitivity 69  
     database differences 75  
     pattern matching 69  
     Tcl window vs Constraints Editor 74  
     usage tips 71

---

- useful -filter examples [71](#)
- using in Constraints Editor [75](#)
- Tcl script window
  - collections compared to Constraints Editor [74](#)
- TCL View [51](#)
  - using [51](#)
- TCL View, SCOPE [38](#)
- Technology view
  - crossprobing from source file [186](#)
  - filtering [138, 194](#)
  - finding objects with Hierarchy Browser [113](#)
  - flattening hierarchy [201](#)
  - highlighting collections [81](#)
- Text Editor view
  - crossprobing [184](#)
- text files
  - crossprobing [186](#)
- text macro
  - Verilog [426](#)
- through constraints [59, 60](#)
  - AND lists [61](#)
  - OR list [61](#)
  - point-to-point delays [32](#)
- TIMESPEC I/O constraint (Xilinx)
  - xc\_use\_timespec\_for\_io attribute [919](#)
- timing
  - syn\_tco directive [819](#)
  - syn\_tpd directive [824](#)
  - syn\_tsu directive [835](#)
- timing constraints
  - conflict resolution [64](#)
  - constraint priority [64](#)
  - using with automatic compile points [328](#)
- timing exceptions
  - priority [64](#)
- timing optimization [295](#)
- tips
  - memory usage [205](#)
- to constraints
  - specifying [56](#)
- translate\_off directive [869](#)
- translate\_on directive [869](#)
- transparent instances
  - flattening [201](#)
  - lower-level logic on multiple sheets [163](#)

triple DES [404](#)

tristates

black\_box\_tri\_pins directive [479](#)

syn\_tristate directive [832](#)

typedefs [428](#)

## **U**

UCF

clock priority with syn\_clock\_priority [543](#)

UNISIM library [292](#)

unsupported language constructs (VHDL) [440](#)

UpdateMEM

using [245](#)

Use OFFSET for I/O constraints option [919](#)

use statement (VHDL) [444](#)

utilization

syn\_packer\_effort\_level [724](#)

## **V**

Verilog

'ifdef [425](#)

adding attributes and directives [465](#)

asynchronous sets and resets [419](#)

attribute syntax [427](#)

black boxes [260, 423](#)

case sensitivity for Tcl Find command [69](#)

clock DLLs [285](#)

crossprobing from HDL Analyst view [184](#)

defining FSMs [253](#)

defining state machines with parameter and 'define [254](#)

directive syntax [427](#)

hierarchical design [424](#)

hierarchy [424, 425](#)

ignored language constructs [411](#)

ignoring code with 'ifdef [425](#)

ignoring code with translate off/on [869](#)

initial value data file [235](#)

initializing RAMs [234](#)

instantiating

  black boxes [423](#)

language

  constructs [410](#)

macro library (Xilinx) [268](#)

operators [411](#)

sets and resets [419](#)

shift registers [250](#)

---

- state machines [421](#)
- state values (FSM) [423](#)
- structural Verilog [424](#)
  - supported language constructs [410](#)
  - syn\_keep on multiple nets [668](#)
  - synchronous sets and resets [419](#)
  - synthesis macro [425](#)
  - synthesis text macro [425](#)
  - text macro [426](#)
  - wildcard (\*) in sensitivity list [413](#)
- Verilog 2001 support [413](#)
- Verilog include files
  - hierarchical designs [425](#)
- Verilog model (.vmd) [328](#)
- Verilog synthesis guidelines [416](#)
- VHDL
  - accessing packages [444](#)
  - adding attributes and directives [463](#)
  - adding source files to design library [442](#)
  - attributes package [463](#)
  - black boxes [261](#)
  - case sensitivity for Tcl Find command [69](#)
  - clock DLLs [285](#)
  - compiling design units into libraries [442](#)
  - crossprobing from HDL Analyst view [184](#)
  - defining FSMs [254](#)
  - design libraries [442](#)
  - ignored language constructs [441](#)
  - initializing RAM with variable declarations [232](#)
  - initializing with signal declarations [231](#)
  - instantiating
    - components [445](#)
  - instantiating components [445](#)
  - language
    - constructs [439, 442](#)
  - libraries [442](#)
    - attributes, supplied with synthesis tool [443](#)
  - library and package rules [444](#)
  - library packages
    - accessing [444](#)
    - attributes package [463](#)
    - IEEE support [439](#)
    - predefined [443](#)
  - library statement [444](#)
  - macro library (Xilinx) [268](#)
  - packages [442](#)
  - predefined packages [443](#)
  - resource library, creating [445](#)
  - sequential shift components [249](#)

statements  
library 444  
use 444  
supported language constructs 439  
unsupported language constructs 440  
use statement 444  
VHDL 2008 451  
enabling 455  
operators 451  
packages 454  
VHDL components  
creating resource library 446  
instantiating 445  
vendor macro libraries 446  
VHDL libraries  
compiling design units 442  
Virtex  
clock buffers 285  
I/O buffers 287  
PCI core 270  
Vivado IP 352  
creating 345, 346  
Vivado IP Block Designer 347  
Vivado IP Catalog 346

## W

warnings  
feedback muxes 296  
wide adders/subtractors  
example 252  
inferring 251  
prerequisites for inference 251  
wildcards  
effect of search scope 177  
Find command (Tcl) 69  
Verilog sensitivity list 413  
wildcards (Find)  
examples 179  
how they work 123, 177  
wires, preserving with syn\_keep directive 667, 729

## X

xc\_area\_group (Xilinx) 872  
xc\_clockbuftype attribute 877  
specifying 286

---

- `xc_fast` attribute [881](#)
  - for critical paths [293](#)
- `xc_fast_auto` attribute [885](#)
- `xc_global_buffers`
  - and `syn_global_buffers` [888](#)
- `xc_global_buffers` attribute [888](#)
- `xc_isgsr` directive [891](#)
- `xc_loc` attribute [894](#)
  - assigning locations in SCOPE [278](#)
- `xc_map` attribute [897](#)
  - relative location [289](#)
- `xc_padtype` attribute [901](#)
  - specifying I/Os [287](#)
- `xc_pulldown` attribute [902](#)
- `xc_pullup` attribute [902](#)
- `xc_rloc` attribute [906](#)
  - specifying relative location [289](#)
- `xc_slow` attribute [910](#)
- `xc_use_keep_hierarchy` attribute [911](#)
- `xc_use_rpms` attribute [916](#)
- `xc_use_timespec_for_io` attribute [919](#)
- `xc_uset` attribute [923](#)
  - grouping instances for relative placement [289](#)
  - using to group instances [289](#)

**Xilinx**

- clock buffers [285](#)
- CoreGen [270](#)
- DSP component [579, 580](#)
- I/O buffers [287](#)
- I/O insertion, manual [281](#)
- I/O locations [277](#)
- INIT property [238](#)
- INIT property, VHDL [233](#)
- IP cores [270](#)
- macro libraries [268](#)
- macros [268](#)
- OFFSET for I/O constraints [919](#)
- packing registers [274](#)
- pad types [901](#)
- specifying pin location [277](#)
- TIMESPEC for I/O constraints [919](#)
- using BUFR [286](#)

Xilinx differential I/O buffer inference [284](#)

Xilinx DSP48 register conversion [536, 543, 760](#)

