

Verification Continuum™

# **Synopsys Synplify Pro for Lattice**

## **Attribute Reference Manual**

---

November 2020



---

## Copyright Notice and Proprietary Information

© 2020 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

## 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)

November 2020



# Contents

---

## Chapter 1: Introduction

How Attributes and Directives are Specified	8
The SCOPE Attributes Tab	8
Summary of Attributes and Directives	10
Summary of Global Attributes	11

## Chapter 2: Attributes and Directives

Attributes and Directives Summary	13
black_box_pad_pin	15
black_box_tri_pins	20
full_case	23
loc	27
loop_limit	28
parallel_case	31
pragma translate_off/pragma translate_on	33
syn_allow_retiming	36
syn_allowed_resources	40
syn_black_box	44
syn_direct_enable	51
syn_direct_reset	56
syn_direct_set	61
syn_dspstyle	66
syn_encoding	69
syn_enum_encoding	79
syn_force_pads	85
syn_force_seq_prim	88
syn_gatedclk_clock_en	91
syn_gatedclk_clock_en_polarity	94
syn_global_buffers	99
syn_hier	106
syn_insert_buffer	116

---

syn_insert_pad	119
syn_isclock	123
syn_keep	127
syn_looplimit	133
syn_maxfan	135
syn_multstyle	141
syn_netlist_hierarchy	144
syn_noarrayports	150
syn_noclockbuf	154
syn_noclockpad	160
syn_noprune	165
syn_pad_type	181
syn_pipeline	185
syn_preserve	190
syn_probe	195
syn_ramstyle	203
syn_reduce_controlset_size	210
syn_reference_clock	219
syn_replicate	221
syn_romstyle	226
syn_safe_case	231
syn_safefsm_pipe	234
syn_sharing	237
syn_shift_resetphase	242
syn_smhigheffort	246
syn_srlstyle	251
syn_state_machine	254
syn_tco<n>	260
syn_tpd<n>	266
syn_tristate	272
syn_tsu<n>	275
syn_use_carry_chain	281
syn_useenables	294
syn_useioff	299
translate_off/translate_on	306

## CHAPTER 1

# Introduction

---

This document is part of a set that includes reference and procedural information for the Synopsys<sup>®</sup> FPGA synthesis tool.

This document describes the attributes and directives available in the tool. The attributes and directives let you direct the way a design is analyzed, optimized, and mapped during synthesis.

This chapter includes the following introductory information:

- [How Attributes and Directives are Specified](#), on page 8
- [Summary of Attributes and Directives](#), on page 10
- [Summary of Global Attributes](#), on page 11

## How Attributes and Directives are Specified

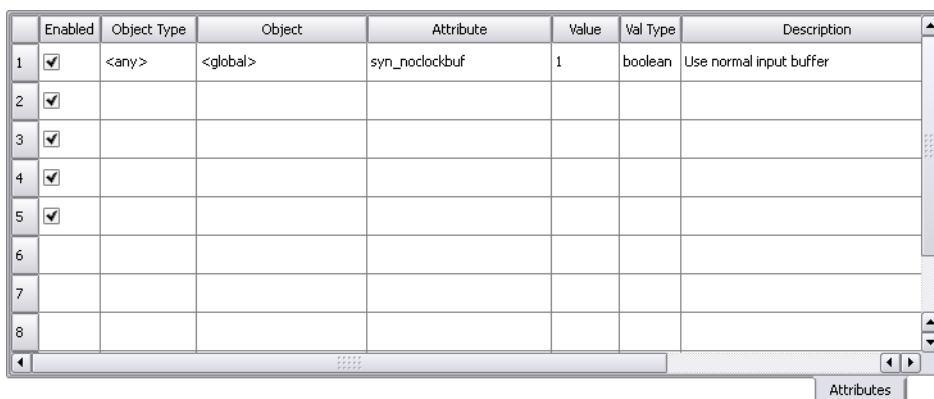
By definition, *attributes* control mapping optimizations and *directives* control compiler optimizations. Because of this difference, directives must be entered directly in the HDL source code or through a compiler design constraint file. Attributes can be entered either in the source code, in the SCOPE Attributes tab, or manually in a constraint file. For detailed procedures on different ways to specify attributes and directives, see [Specifying Attributes and Directives, on page 102](#) in the *User Guide*.

Verilog files are case sensitive, so attributes and directives must be entered exactly as presented in the syntax descriptions. For more information about specifying attributes and directives using C-style and Verilog 2001 syntax, see [Verilog Attribute and Directive Syntax, on page 132](#).

### The SCOPE Attributes Tab

This section describes how to enter attributes using the SCOPE Attributes tab. To use the SCOPE spreadsheet, use this procedure:

1. Start with a compiled design, then open the SCOPE window.
2. Scroll if needed and click the Attributes tab.



	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	<any>	<global>	syn_noclockbuf	1	boolean	Use normal input buffer
2	<input checked="" type="checkbox"/>						
3	<input checked="" type="checkbox"/>						
4	<input checked="" type="checkbox"/>						
5	<input checked="" type="checkbox"/>						
6							
7							
8							

3. Click in the Attribute cell and use the pull-down menus to enter the appropriate attributes and their values.

The Attributes panel includes the following columns.



Column	Description
Enabled	(Required) Turn this on to enable the constraint.
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 RTL or Technology view into this column.
Attribute	(Required) 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 synthesis tool filters the available choices 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 kind 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.

For more details on how to use the Attributes panel of the SCOPE spreadsheet, see [Specifying Attributes Using the SCOPE Editor](#), on page 105 in the *User Guide*.

When you use the SCOPE 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 whether your source code is in Verilog or VHDL format. See <a href="#">syn_black_box</a> , on page 44 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.

See [Summary of Global Attributes, on page 11](#) for more details on specifying global attributes in the synthesis environment.

## Summary of Attributes and Directives

The following sections summarize the synthesis attributes and directives:

- [Summary of Global Attributes, on page 11](#)
- [Chapter 2, Attributes and Directives](#)

For detailed descriptions of individual attributes and directives, see the individual attributes and directives, which are listed in alphabetical order.

# Summary of Global Attributes

Design attributes in the synthesis environment 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 the specified design object (module, view, port, instance, clock, and so on). When an attribute is set both globally and locally on a design object, the local specification overrides the global specification for the object.

In general, the syntax for specifying a global attribute in a constraint file is:

```
define_global_attribute attribute_name {value}
```

The table below contains a list of attributes that can be specified globally in the synthesis environment. For complete descriptions of any of the attributes listed below, see [Attributes and Directives Summary](#), on page 13.

Global Attribute	Can Also Be Set On Design Objects
<a href="#">syn_allow_retiming</a>	x
<a href="#">syn_allowed_resources</a>	x
<a href="#">syn_dspstyle</a>	x
<a href="#">syn_noarrayports</a>	
<a href="#">syn_ramstyle</a>	x
<a href="#">syn_reduce_controlset_size</a>	
<a href="#">syn_replicate</a>	x
<a href="#">syn_romstyle</a>	x
<a href="#">syn_srlstyle</a>	x
<a href="#">syn_useioff</a>	x



## CHAPTER 2

# Attributes and Directives

---

All attributes and directives supported for synthesis are listed in alphabetical order. Each command includes syntax, option and argument descriptions, and examples. You can apply attributes and directives globally or locally on a design object.

## Attributes and Directives Summary

The following attributes and directives are listed in alphabetical order.

<a href="#">black_box_pad_pin</a>	<a href="#">black_box_tri_pins</a>
<a href="#">full_case</a>	<a href="#">loc</a>
<a href="#">loop_limit</a>	<a href="#">parallel_case</a>
<a href="#">pragma translate_off/pragma translate_on</a>	<a href="#">syn_allow_retiming</a>
<a href="#">syn_allowed_resources</a>	<a href="#">syn_black_box</a>
<a href="#">syn_direct_enable</a>	<a href="#">syn_direct_reset</a>
<a href="#">syn_direct_set</a>	<a href="#">syn_dspstyle</a>
<a href="#">syn_encoding</a>	<a href="#">syn_enum_encoding</a>
<a href="#">syn_force_pads</a>	<a href="#">syn_force_seq_prim</a>
<a href="#">syn_gatedclk_clock_en</a>	<a href="#">syn_gatedclk_clock_en_polarity</a>

<a href="#">syn_global_buffers</a>	<a href="#">syn_hier</a>
<a href="#">syn_insert_buffer</a>	<a href="#">syn_insert_pad</a>
<a href="#">syn_isclock</a>	<a href="#">syn_keep</a>
<a href="#">syn_looplmit</a>	<a href="#">syn_maxfan</a>
<a href="#">syn_multstyle</a>	<a href="#">syn_netlist_hierarchy</a>
<a href="#">syn_noarrayports</a>	<a href="#">syn_noclockbuf</a>
<a href="#">syn_noclockpad</a>	<a href="#">syn_noprune</a>
<a href="#">syn_pad_type</a>	<a href="#">syn_pipeline</a>
<a href="#">syn_preserve</a>	<a href="#">syn_probe</a>
<a href="#">syn_ramstyle</a>	<a href="#">syn_reduce_controlset_size</a>
<a href="#">syn_reference_clock</a>	<a href="#">syn_replicate</a>
<a href="#">syn_romstyle</a>	<a href="#">syn_safe_case</a>
<a href="#">syn_safefsm_pipe</a>	<a href="#">syn_sharing</a>
<a href="#">syn_shift_resetphase</a>	<a href="#">syn_smhigh effort</a>
<a href="#">syn_srlstyle</a>	<a href="#">syn_state_machine</a>
<a href="#">syn_tco&lt;n&gt;</a>	<a href="#">syn_tpd&lt;n&gt;</a>
<a href="#">syn_tristate</a>	<a href="#">syn_tsu&lt;n&gt;</a>
<a href="#">syn_use_carry_chain</a>	<a href="#">syn_useenables</a>
<a href="#">syn_useioff</a>	<a href="#">translate_off/translate_on</a>

## black\_box\_pad\_pin

### *Directive*

Specifies that the pins on a black box are I/O pads visible to the outside environment.

### black\_box\_pad\_pin Values

Value	Description
<i>portName</i>	Specifies ports on the black box that are I/O pads.

### Description

Used with the `syn_black_box` directive and specifies that pins on black boxes 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, and without enclosed spaces.

To instantiate an I/O from your programmable logic vendor, you usually do not need to define a black box or this directive. The synthesis tool provides predefined black boxes for vendor I/Os. For more information, refer to your vendor section under FPGA and CPLD Support.

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 44](#) for a list of the associated directives.

### black\_box\_pad\_pin Values Syntax

The following support applies for the `black_box_pad_pin` attribute.

Global Support	Object
----------------	--------

No	Verilog module or VHDL architecture declared for a black box
----	--

This table summarizes the syntax in different files:

Verilog	<i>object</i> /* synthesis black_box_pad_pin = <i>portList</i> */;	<a href="#">Verilog Example</a>
VHDL	attribute black_box_pad_pin of <i>object</i> : <i>objectType</i> is <i>portList</i> ;	<a href="#">VHDL Example</a>

Where

- *object* is a module or architecture declaration of a black box.
- *portList* is a spaceless, comma-separated list of the names of the ports on black boxes that are I/O pads.
- *objectType* is a string in VHDL code.

## Verilog Example

This example shows how to specify this attribute in the following Verilog code segment:

```
module BBDLHS(D,E,GIN,GOUT,PAD,Q)
  /* synthesis syn_black_box black_box_pad_pin="GIN[2:0],Q" */;
```

## VHDL Example

This example shows how to specify this attribute in the following VHDL code:

```
library AI;
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 black_box_pad_pin : string;
attribute syn_black_box of test : component is true;
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 topl_arch;

```

## Effect of Using black\_box\_pad\_pin

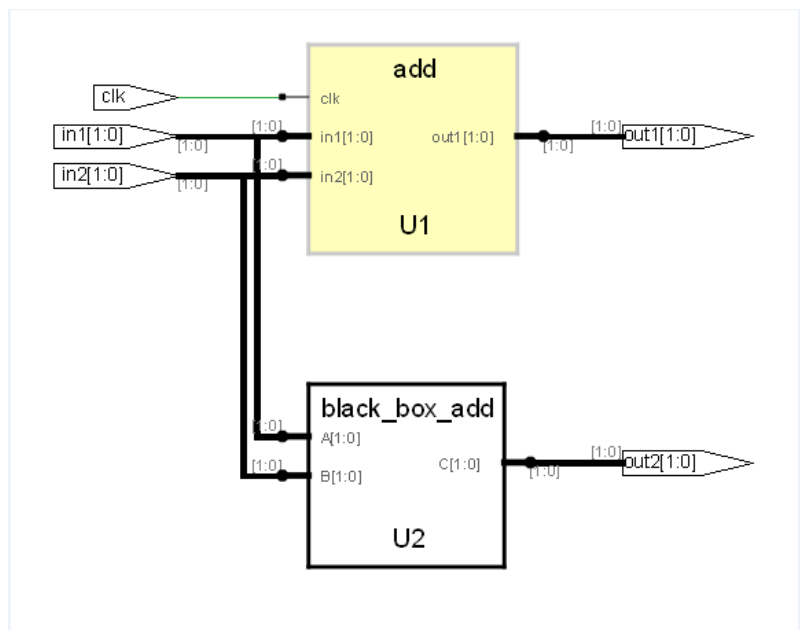
The following example shows the effect of applying the attribute.

### Before using black\_box\_pad\_pin

```

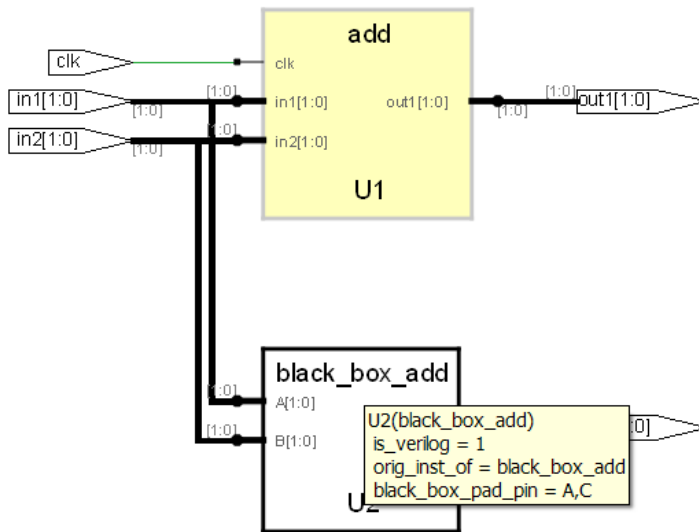
)
(cell black_box_add (cellType GENERIC)
  (view verilog (viewType NETLIST)
    (interface
      (port (array (rename A "A[1:0]") 2) (direction INPUT)
        (port (array (rename B "B[1:0]") 2) (direction INPUT)
          (port (array (rename C "C[1:0]") 2) (direction OUTPUT))
        )
      (property 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 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.

### black\_box\_tri\_pins Values

Value	Description
<i>portName</i>	Specifies an output port on the black box that is a tristate.

### Description

Used with the `syn_black_box` directive and specifies 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 (,), and without enclosed spaces.

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 44 for a list of the associated directives.

### black\_box\_tri\_pins Values Syntax

The following support applies for the `black_box_tri_pins` attribute.

#### Global Support    Object

No	Verilog module or VHDL architecture declared for a black box
----	--

This table summarizes the syntax in different files:

Verilog	<code>object /* synthesis black_box_tri_pins = portList */;</code>	<a href="#">Verilog Example</a>
VHDL	attribute black_box_tri_pins of <i>object</i> : <i>objectType</i> is <i>portList</i> ;	<a href="#">VHDL Example</a>

Where

- *object* is a module or architecture declaration of a black box.
- *portList* is a spaceless, comma-separated list of the tristate output port names.
- *objectType* is a string in VHDL code.

## Verilog Example

Here is an example with a single port name:

```
module BBDLHS(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 bbl(D,E,tri1,tri2,tri3,Q)
  /* synthesis syn_black_box black_box_tri_pins="tri1,tri2,tri3" */;
```

For a bus, you specify the port name followed by all the bits on the bus:

```
module bbl(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 BBDLHS
    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 BBDLHS : component is true;
  attribute black_box_tri_pins : string;
  attribute black_box_tri_pins of BBDLHS : 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 bbl : component is  
    "tri,tri2,tri3";
```

To apply this directive to a port that is a bus, specify all the bits on the bus:

```
attribute black_box_tri_pins of bbl : component is "bus1[7:0]";
```

## full\_case

### *Directive*

For Verilog designs only. Indicates that all possible values have been given, and that no additional hardware is needed to preserve signal values.

### full\_case Values

Value	Description
1 (Default)	All possible values have been given and no additional hardware is needed to preserve signal values.

### Description

For Verilog designs only. 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 needed to preserve signal values.

### full\_case Values Syntax

This table summarizes the syntax in the following file type:

Verilog	<code>object /* synthesis full_case */;</code>	<a href="#">Verilog Examples</a>
---------	--	----------------------------------

### 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;

```

```

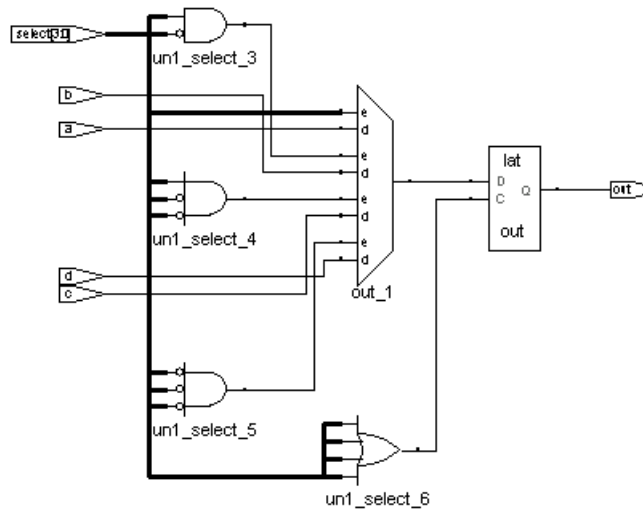
always @(select or a or b
or c or d)

```

```

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 the 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 like 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 them:

<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 <b>select</b> is assigned a 'bx	Can cause mismatches between pre- and post-synthesis simulation because the simulator does not use <b>full_case</b>

# loc

## *Attribute*

Specifies pin locations for Lattice I/Os, instances, and registers and forward-annotates them to the place-and-route tool.

Vendor	Technology
--------	------------

Lattice	All
---------	-----

## Description

The loc attribute specifies pin locations for Lattice I/Os, instances, and registers, and forward-annotates them to the place-and-route tool. If the attribute is on a bus, the software writes out bit-blasted constraints for forward-annotation. This attribute can only be specified in a constraint file.

Refer to the Lattice databook for valid pin location values.

## loc Syntax

Global Support	Object
----------------	--------

No	Lattice I/Os, instances, registers
----	------------------------------------

## FDC Example

```
define_attribute {portName} loc {pinLocations}
```

*pinLocations* is a comma-separated list of pin locations.

The following example assigns a pad location to all bits of a bus:

```
define_attribute {DATA0[3:0]} loc {P14,P12,P11,P5}
```

## loop\_limit

*Directive*

*Verilog*

Specifies a loop iteration limit for a for loop in a Verilog design when the loop index is a variable, not a constant.

### loop\_limit Values

Value	Description
1 - 1999	Overrides the default loop limit of 2000 in the RTL.

### Description

For Verilog designs only.

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 1999 when the exit or terminating condition does not compute a constant value, or to avoid infinite loops. 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. The limit cannot be an expression. The higher the value you set, the longer the runtime.

Alternatively, you can use the `set_option looplimit` command (Loop Limit GUI option) to set a global loop limit that overrides the default of 2000 loops in the RTL. To use the Loop Limit option on the Verilog tab of the Implementation Options panel, see [Verilog Panel, on page 378](#) in the *Command Reference*.

---

**Note:** VHDL applications use the `syn_looplimit` directive (see [syn\\_looplimit, on page 133](#)).

---

## loop\_limit Values Syntax

The following support applies for the loop\_limit directive.

Global Support	Object
Yes	Specifies the beginning of the loop statement.

This table summarizes the syntax in the following file:

Verilog */\* synthesis loop\_limit integer \*/ loopStatement*

[Verilog Example](#)

## Verilog Example

The following is an example where the loop limit is set to 2000:

```
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 2000 */
    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`

Code with more than 2000 loops will not produce 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 so that tag-matching logic can be parallel and independent, instead of chained.

### parallel\_case Syntax

The following support applies for the `parallel_case` directive.

Global Support	Object
No	A case, caseX, or caseZ statement declaration

This table summarizes the syntax in the following file type:

Verilog	object /* synthesis parallel_case */	<a href="#">Verilog Example</a>
---------	--------------------------------------	---------------------------------

### 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;

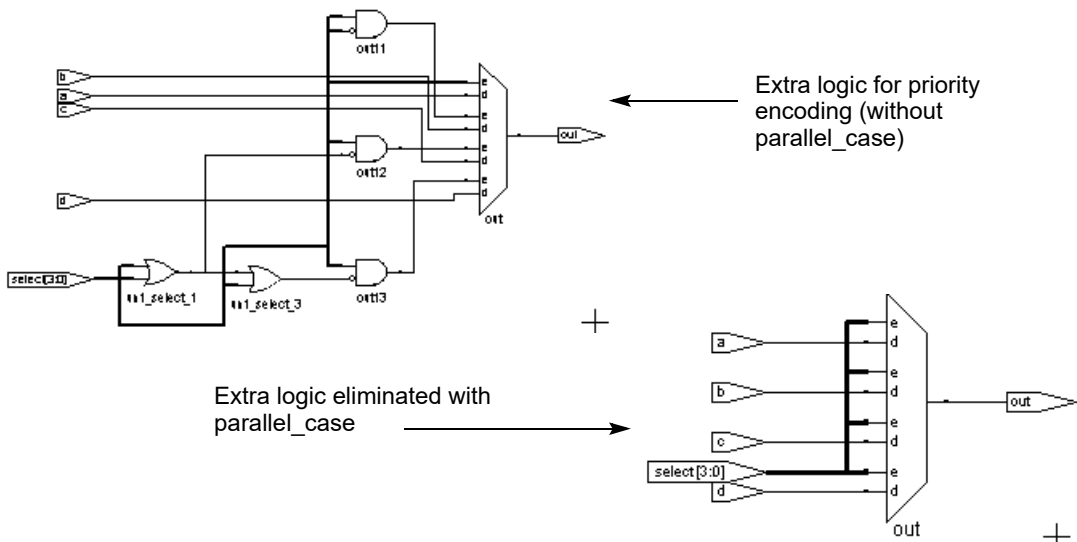
always @(select or a or b or c or d)

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 are not synthesizable.

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 306. These directives are implemented the same in the source code.

---

This table summarizes the syntax in the following file type:

Verilog	<pre>/* pragma translate_off */ /* pragma translate_on */ /*synthesis translate_off */ /*synthesis translate_on */</pre>	<a href="#">Verilog Example</a>
VHDL	<pre>--pragma translate_off --pragma translate_on --synthesis translate_off --synthesis translate_on</pre>	<a href="#">VHDL Example</a>

## 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
```

## 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;

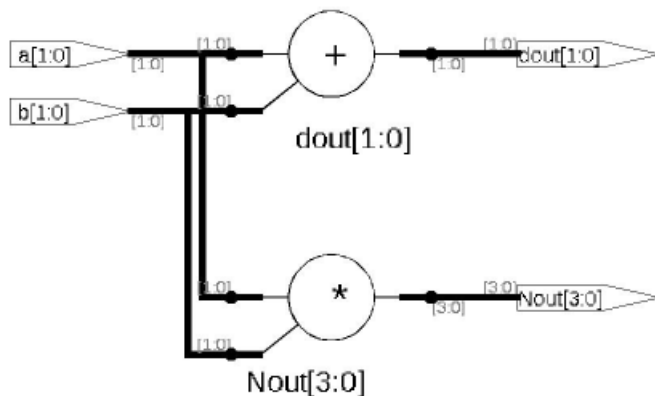
architecture rtl of test is
begin
  dout <= a + b;

  --Anything between pragma translate_off/translate_on is ignored by
  --the synthesis tool hence only
  --the adder circuit above is implemented not the multiplier circuit
  --below:

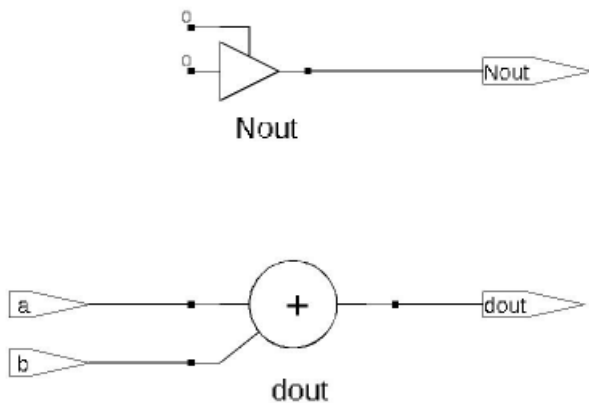
  --pragma translate_off
    Nout <= a * b;
  --pragma translate_on
end;
```

## Effect of Using `pragma translate_off/pragma translate_on`

Before applying the attribute:



After applying the attribute:



## syn\_allow\_retiming

### *Attribute*

Determines if registers can be moved across combinational logic to improve performance.

Vendor	Technology	Synthesis Tool
Lattice	iCE40, iCE40UP	Synplify Pro

### syn\_allow\_retiming values

1 | true    Allows registers to be moved during retiming.

0 | false    Does not allow retimed registers to be moved.

### Description

The `syn_allow_retiming` attribute determines if registers can be moved across combinational logic to improve performance.

The attribute can be applied either globally or to specific registers. Typically, you enable the global Retiming option in the UI (or the `set_option -retiming 1` switch in Tcl) and use the `syn_allow_retiming` attribute to disable retiming for specific objects that you do not want moved. Do not use the `syn_allow_retiming` attribute with the Fast Synthesis flow.

### syn\_allow\_retiming Syntax

Global	Object
Yes	Register

You can specify the attribute in the following files:

FDC	<b>define_attribute {<i>register</i>} syn_allow_retiming {1 0}</b> <b>define_global_attribute syn_allow_retiming {1 0}</b>	<a href="#">FDC Example</a>
Verilog	<i>object</i> /* <b>synthesis syn_allow_retiming = 0   1 *;</b>	<a href="#">Verilog Example</a>
VHDL	<b>attribute syn_allow_retiming of <i>object</i> : <i>objectType</i> is true   false;</b>	<a href="#">VHDL Example</a>

## FDC Example

```
define_attribute {register} syn_allow_retiming {1|0}
define_global_attribute syn_allow_retiming {1|0}
```

Enable	Object Type	Object	Attribute	Value	Value Type	Description
<input checked="" type="checkbox"/>	<any>	<Global>	syn_allow_retiming	1	boolean	Controls retiming of reg...

## Verilog Example

```
object /* synthesis syn_allow_retiming = 0 | 1 *;
```

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_retiming=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

**attribute syn\_allow\_retiming of object : objectType is true | false;**

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;

```

See [VHDL Attribute and Directive Syntax, on page 414](#) for different ways to specify VHDL attributes and directives.

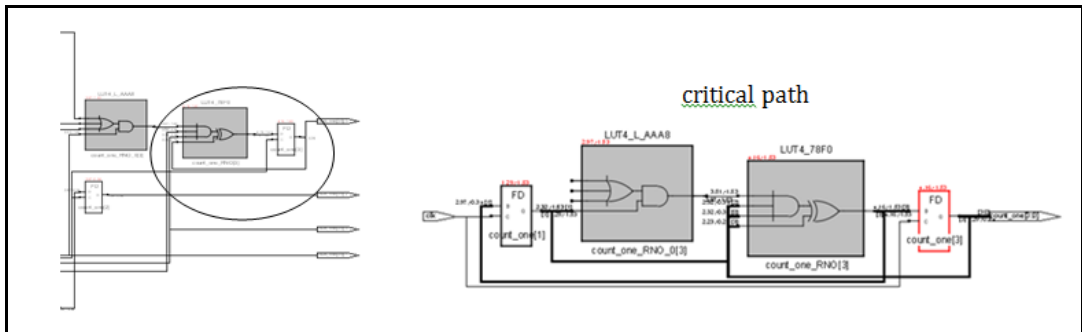
## Effect of using `syn_allow_retiming`

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.

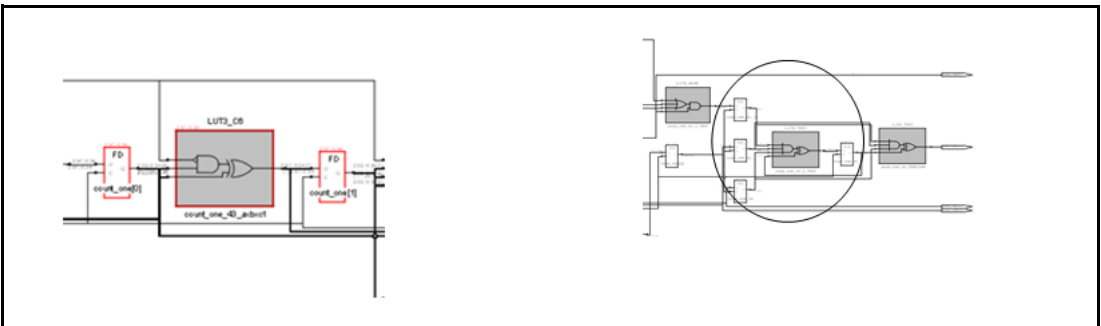


After applying `syn_allow_retiming`.

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.

Vendor	Devices
Lattice	ECP5U/UM, ECP3 families

### syn\_allowed\_resources Values

Default	Global	Object
Multipliers: blockmults RAM: blockrams and distributedrams	Yes	Module Compile points Verilog: register signals VHDL: signals

### 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 chip are used. This can result in exhausting all region or chip 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.

For information on compile points and the compile-point synthesis flow, see [Synthesizing Compile Points, on page 484](#) of the *User Guide*.

## syn\_allowed\_resources Syntax

FDC	<pre>define_attribute {v:module   architectureName} syn_allowed_resources {spec=n [, spec=n...]} define_global_attribute syn_allowed_resources {spec=n [, spec=n ... ]}</pre>	<a href="#">FDC Example</a>
Verilog	<pre>object /* synthesis syn_allowed_resources = "spec=N" */; object must be register definition (reg) signals.</pre>	<a href="#">Verilog Example</a>
VHDL	<pre>attribute syn_allowed_resources of object : objectType is "spec= N"; object can be a signal that defines a compile point or a label of a component instance. For descriptions of the spec values, see the table below.</pre>	<a href="#">VHDL Example</a>

- *module or architectureName* is the name of a Verilog module or VHDL architecture that has been defined as a compile point.
- *spec* can be any of the keywords in the table below depending on the technology you select.
- *n* is an integer that specifies the maximum number of resources of the specified type.

## Technology-Specific spec Values

Values	Devices	Resource
blockrams	Lattice ECP5U/UM, ECP3	Block RAM
blockmults	Lattice ECP3	Multiplier blocks
distributedrams	Lattice ECP5U/UM, ECP3	Distributed RAM

If you make multiple assignments with different types of resources, separate them with commas. See [FDC Example, on page 42](#).

## FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	view	v:work.top	syn_allowed_resources	blockmults=2	string	Control resource usage in a compile point

Tcl examples:

```
define_attribute
{v:work.test}{syn_allowed_resources}{blockmults=2}

define_attribute
{v:work.test}{syn_allowed_resources}{distributedrams=512,
    blockrams=0}

define_global_attribute {syn_allowed_resources}
{distributedrams=0,
    blockrams=25}
```

## Verilog Example

```
module test (clk, a, b, c) /* synthesis
syn_allowed_resources="blockrams=2" */

module test (clk, a, b, c) /* synthesis syn_allowed_resources=
    "blockrams=0|distributedrams=256" */
```

## VHDL Example

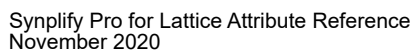
```
entity test is
port ( ... );
attribute syn_allowed_resources : string;
attribute syn_allowed_resources of top : entity is "blockrams=2";
end test;
```

For example:

```
attribute syn_allowed_resources : string;
attribute syn_allowed_resources of test : entity is
    "distributedrams=0";

attribute syn_allowed_resources : string;
attribute syn_allowed_resources of test : entity is "blockrams=2,
    distributedrams=1024";
```

The following figure shows a Lattice ECP3 device after applying the attribute with blockmult=2. Only two ALU54 blocks are used after running synthesis, although there are more ALU resources available for this device. One of the two ALU54 blocks is shown in the Technology view.



## syn\_black\_box

### *Directive*

Defines a module or component as a black box.

### syn\_black\_box Value

Value	Default	Description
<i>moduleName</i>	N/A	Defines an object 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 synthesis 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.

In certain cases, the tool does not honor a `syn_black_box` directive:

- In mixed language designs where a black box is defined in one language at the top level but where there is an existing description for it in another language, the tool can replace the declared black box with the description from the other language.
- If your project includes 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.

To override this and ensure that the attribute is honored, 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.

- If you want to define a black box when you have an `srs` or `edf` description for it, remove the description from the project.

Once you define a black box with `syn_black_box`, you use other source code directives to define timing 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 Source Code Directives

Use the following directives with `syn_black_box` to characterize black-box timing:

<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 Source Code Directives for Gated Clocks

To specify gated clocks on black boxes, you must specify the following directives in addition to the ones listed in [Black-box Source Code Directives, on page 45](#).

<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

You define the pins on a black box with these directives 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.

For more information on black boxes, see [Instantiating Black Boxes in Verilog, on page 123](#), and [Instantiating Black Boxes in VHDL, on page 412](#).

## syn\_black\_box Syntax Specification

Verilog	<code>object /* synthesis syn_black_box */;</code>	<a href="#">Verilog Example</a>
VHDL	attribute syn_black_box of <i>object</i> : <i>objectType</i> is true;	<a href="#">VHDL Example</a>

## 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;

```

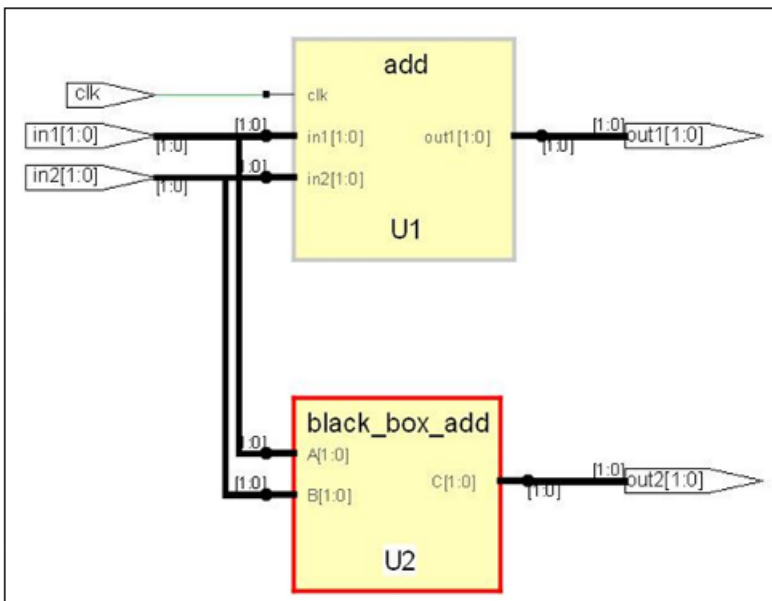
## Effect of Using `syn_black_box`

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);

```

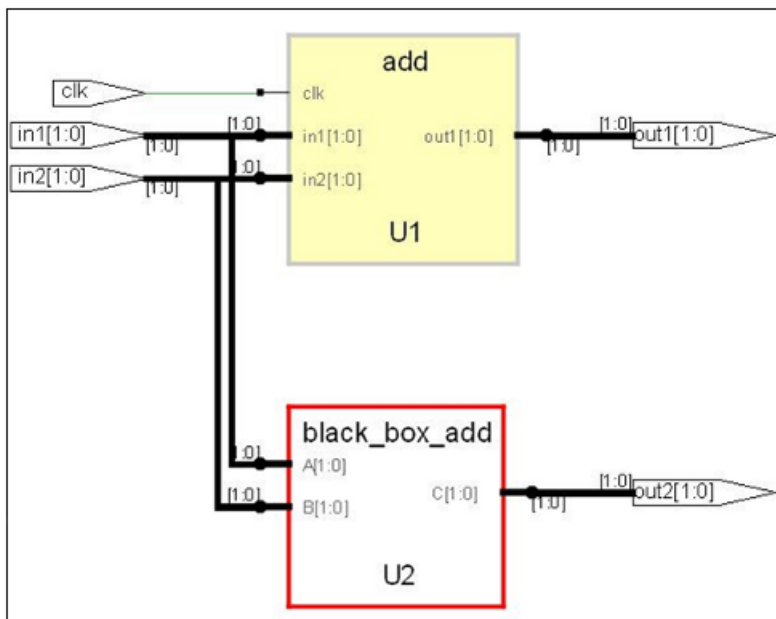


After applying `syn_black_box`, the contents of the black box are no longer visible:

```

module black_box_add(input [1:0]A, [1:0]B, output [1:0]C)/* synthesis
syn_black_box */;

```



## syn\_direct\_enable

*Attribute, Directive*

Controls the assignment of a clock enable net to the dedicated enable pin of a storage element (flip-flop).

Technology	Default Value	Global	Object	Synthesis Tool
Lattice: LatticeECP3, ECP2S, ECP2M, ECP2; LatticeECP, EC; LatticeXP2, XP; LatticeSC, SCM; MachXO	None	No	Net	Synplify Pro

### syn\_direct\_enable values

1   true	Enables nets to be assigned to the clock enable pin.
0   false	Does not assign nets to the clock enable pin.

### 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 SCOPE spreadsheet.

### syn\_direct\_enable Syntax

FDC	<b>define_attribute {<i>object</i>} syn_direct_enable {1}</b>	<a href="#">FDC Example</a>
Verilog	<b><i>object</i> /* synthesis syn_direct_enable = 1 */;</b>	<a href="#">Verilog Example</a>
VHDL	<b>attribute syn_direct_enable of <i>object</i> : <i>objectType</i> is true;</b>	<a href="#">VHDL Example</a>

## FDC Example

Enable	Object Type	Object	Attribute	Value	Value Type	Description
<input checked="" type="checkbox"/>	<any>	<Global>	syn_direct_enable	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;
```

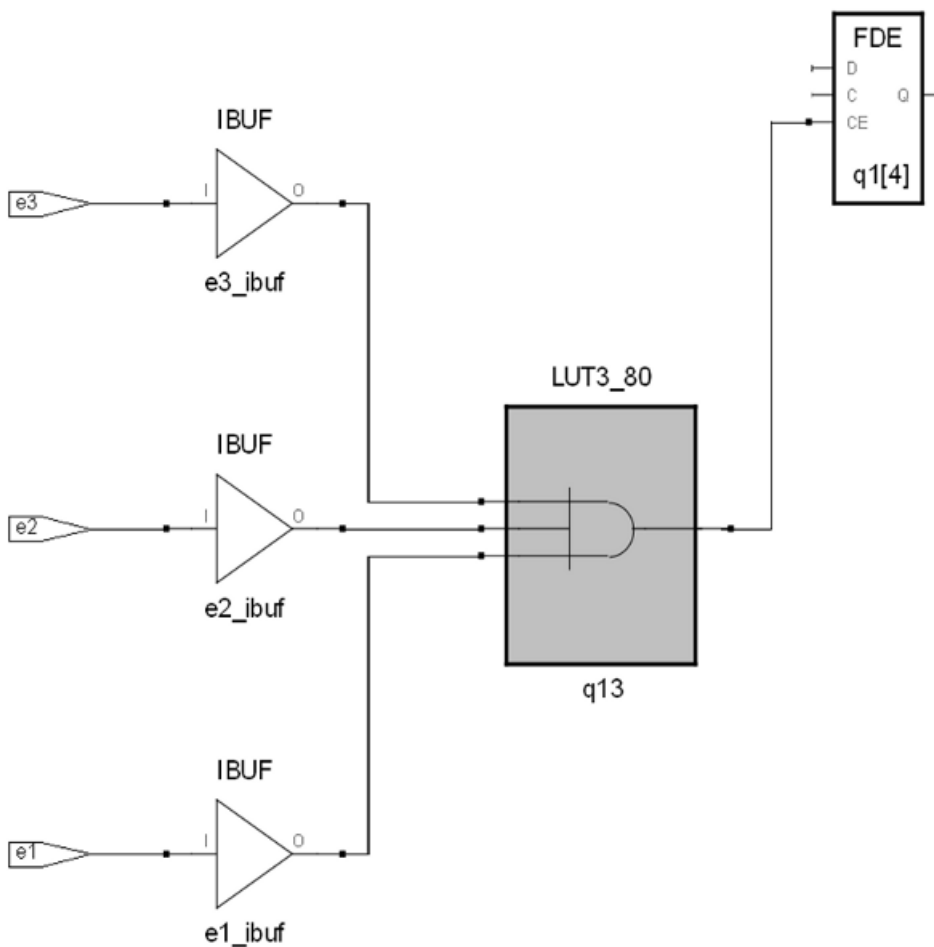
See [VHDL Attribute and Directive Syntax, on page 414](#) for different ways to specify VHDL attributes and directives.

## Effect of Using `syn_direct_enable`

### Before applying `syn_direct_enable`:

```
Verilog    input e3 /* synthesis syn_direct_enable = 0 */;
```

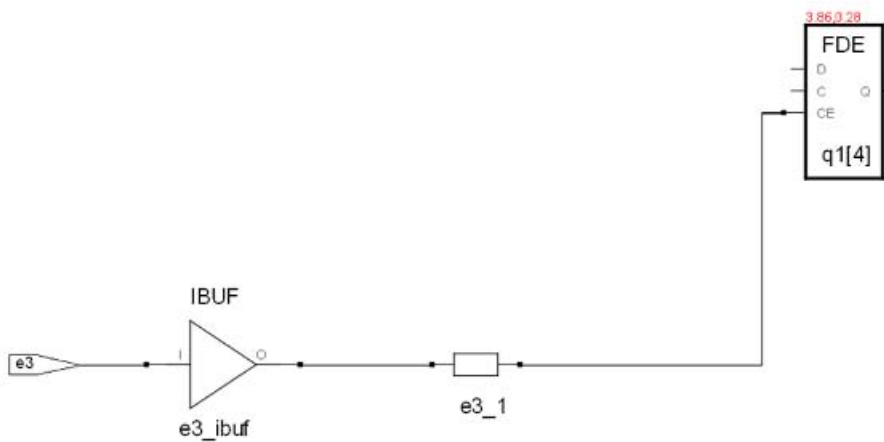
```
VHDL      attribute syn_direct_enable of e3: signal is false;
```



**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).

Vendor	Technology
--------	------------

Lattice	ECP, SC, XP, MachXO families
---------	------------------------------

### syn\_direct\_reset Values

Value	Description
1   true	Enables the software to assign a net to the dedicated reset pin of a synchronous storage element (flip-flop).
0   false (Default)	Disables the software from assigning 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

Global Attribute	Object
No	<code>p:portName</code>



The following table summarizes the syntax in different files:

FDC	<b>define_attribute syn_direct_reset {0 1}</b>	<a href="#">SCOPE Example</a>
Verilog	<i>object</i> /* <b>synthesis syn_direct_reset = 0 1</b> */	<a href="#">Example — Verilog syn_direct_reset</a>
VHDL	<b>attribute syn_direct_reset : boolean;</b> <b>attribute syn_direct_reset of <i>Object</i> : signal is</b> <b>true false;</b>	<a href="#">Example — VHDL syn_direct_reset</a>

## SCOPE Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>		p:a	syn_direct_reset	1		

## Example — Verilog syn\_direct\_reset

```
// Example 1: Verilog syn_direct_reset 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
```

**Example — VHDL syn\_direct\_reset**

```
-- Example 2: VHDL syn_direct_reset 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 architecture beh;
```

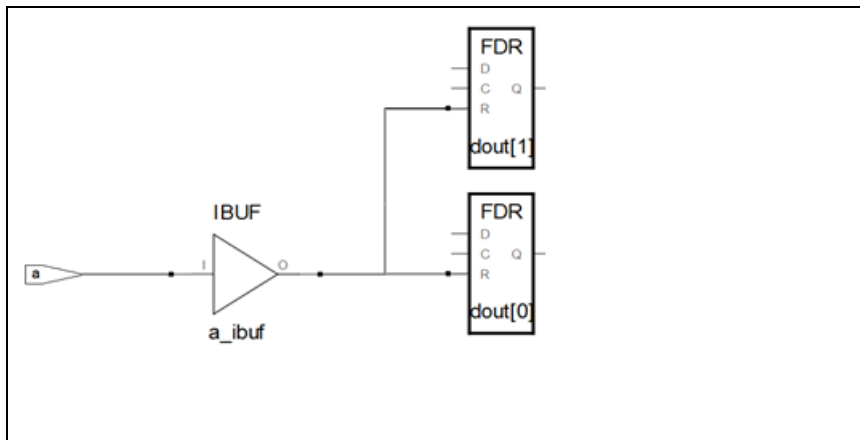
```
        end if;  
    end if;  
end process;  
end beh;
```

## 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*/;
```

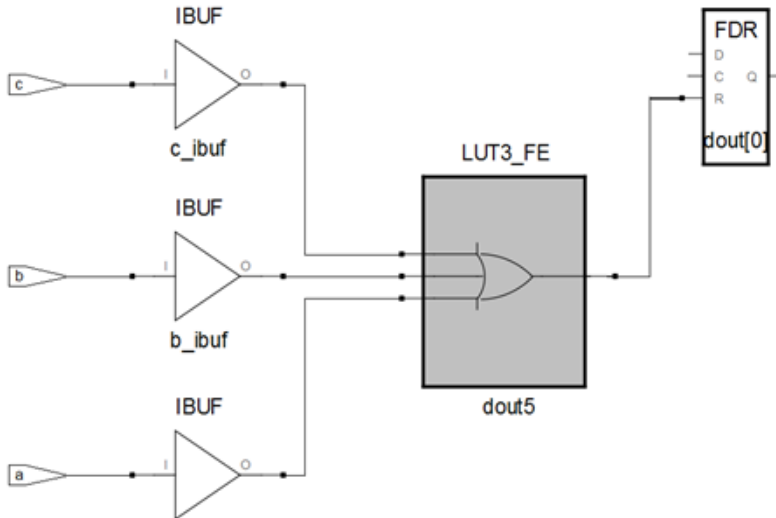
```
VHDL     attribute syn_direct_reset of a : signal is true;
```



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).

Vendor	Technology
--------	------------

Lattice	ECP, SC, XP, MachXO families
---------	------------------------------

### syn\_direct\_set Values

Value	Description
1   true (Default)	Enables the software to assign a net to the dedicated set pin of a synchronous storage element (flip-flop).
0   false	Disables the software from assigning 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

Global Attribute	Object
No	<code>p:portName</code>

The following table summarizes the syntax in different files:

FDC	<b>define_attribute syn_direct_set {0 1}</b>	<a href="#">SCOPE Example</a>
Verilog	<i>object</i> /* <b>synthesis syn_direct_set = 0 1*/</b>	<a href="#">Example — Verilog syn_direct_set</a>
VHDL	<b>attribute syn_direct_set : boolean;</b> <b>attribute syn_direct_set of <i>Object</i> : signal is true false;</b>	<a href="#">Example — VHDL syn_direct_set</a>

## SCOPE Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>		p:a	syn_direct_set	1		

### Example — Verilog syn\_direct\_set

```
// Example 1: Verilog syn_direct_set 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
```

### Example — VHDL syn\_direct\_set

```
-- Example 2: VHDL syn_direct_set 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 architecture;
```

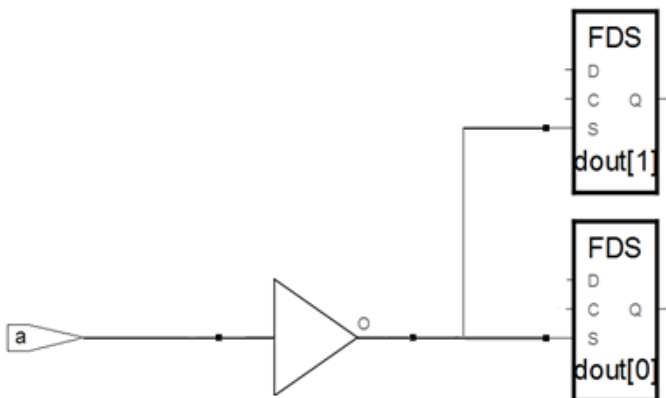
```
    end if;  
end if;  
end process;  
end beh;
```

## 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;
```

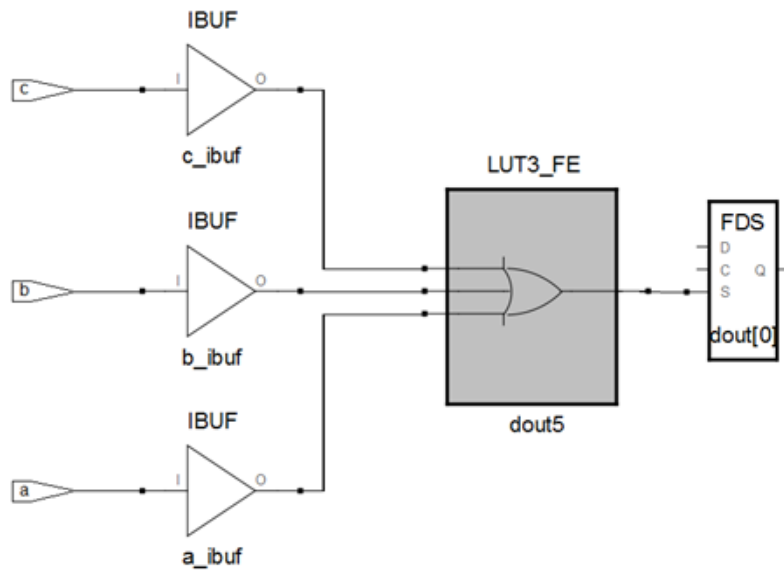




The next figure shows the attribute set to 0. The software does not assign a net for the design to the dedicated set pin of a synchronous storage element (flip-flop):

```
Verilog  input a /*synthesis syn_direct_set = 0*/;
```

```
VHDL    attribute syn_direct_set of a : signal is false;
```



## syn\_dspstyle

### *Attribute*

Determines how multipliers are implemented.

Vendor	Device
Lattice	iCE5LP

### syn\_dspstyle Values

Value	Description
DSP	Implements the multipliers as dedicated hardware blocks. For example: SB_MAC16 blocks in Lattice iCE5LP
Logic	Implements the multipliers as logic.

### syn\_dspstyle Syntax

Global Support	Object
Yes	Module or instance

The following shows the attribute syntax when specified in different files:

FDC	<pre>define_attribute {instance} syn_dspstyle {DSP   logic} Global attribute: define_global_attribute syn_dspstyle {DSP   logic}</pre>	<a href="#">SCOPE Example</a>
Verilog	<pre>input net /* synthesis syn_dspstyle = "DSP   logic" */;</pre>	<a href="#">Verilog Example</a>
VHDL	<pre>attribute syn_dspstyle of instance : signal is "DSP   logic";</pre>	<a href="#">VHDL Example</a>

## SCOPE Example

- This example shows multipliers globally implemented as logic.

Current Design: <Top Level>		Check Constraints					
	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	✓	<any>	<Global>	syn_dspstyle	logic	string	defines whether instance goes into dsp or not
2							
3							

Clocks   Generated Clocks   Collections   Inputs/Outputs   Registers   Delay Paths   Attributes   I/O Standards   Compile Points   TCL Vie

- This example specifies that multipliers be implemented as logic.

```
define_attribute {temp[15:0]} syn_dspstyle {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_dspstyle="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 mult;
architecture rtl of mult is
  signal mult_i : std_logic_vector(15 downto 0);
  attribute syn_dspstyle : string;
  attribute syn_dspstyle 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\_encoding

### *Attribute*

Overrides the default FSM Compiler encoding for a state machine and applies the specified encoding.

Vendor	Devices
Lattice	ECP3, Older devices

### 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:

Value	Description
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 <b>onehot</b> 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 <b>gray</b> 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 <i>attempts</i> 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 <b>gray</b> 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 <b>onehot</b> style if you have a large output decoder following a state machine.</p>

Value	Description
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: 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 <i>onehot</i> style if you have a large output decoder following a state machine.</p>
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 <i>safe</i> before the encoding style. The <i>safe</i> 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 <i>onehot</i> and the state machine reaches a state where all the bits are 0, which is an invalid state, the <i>safe</i> 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 <i>onehot</i>, <i>sequential</i> or <i>gray</i>, in order to force the state machine to reset. When you specify <i>safe</i>, 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 <i>safe</i> and you do not want the additional recovery logic (flip-flop on the inactive clock edge) inserted for this FSM, then use the <i>syn_shift_resetphase</i> attribute to remove it. See <a href="#">syn_shift_resetphase</a> , on page 242 for details.</li> </ul>
original	<p>This respects the encoding you set, but the software still does state machine and reachability analysis.</p>

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 79](#) and [syn\\_encoding Compared to syn\\_enum\\_encoding, on page 81](#) 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 254](#).
- For VHDL designs, see [syn\\_encoding Compared to syn\\_enum\\_encoding, on page 81](#) for comparative usage information.

## Syntax Specification

### Global    Object

Global	Object
No	Instance, register

This table shows how to specify the attribute in different files:

FDC	<code>define_attribute {object} syn_encoding {value}</code>	<a href="#">SCOPE 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.

## SCOPE 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`.

Although you cannot set this attribute globally, you can define a SCOPE collection and then apply the attribute to the collection. For example:

```
define_scope_collection sm {find -hier -inst * -filter
    @inst_of==statemachine}
define_attribute {$sm} {syn_encoding} {safe}
```

## 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

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;

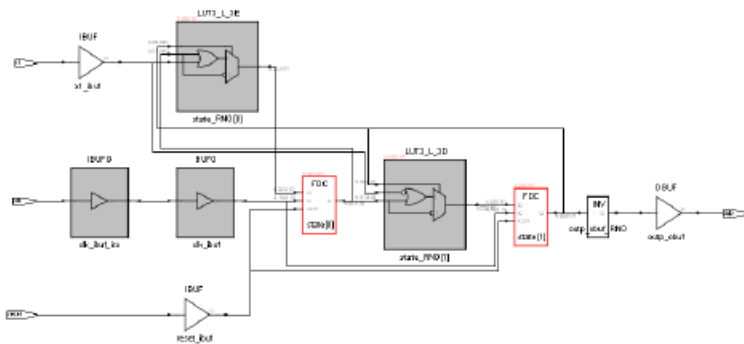
process (state)
begin
  case state is
    when s1 =>
      outp <= '1';
    when s2 =>
      outp <= '1';
    when s3 =>
      outp <= '0';
    when s4 =>
      outp <= '0';
  end case;
end process;
end rtl;
```

See [VHDL Attribute and Directive Syntax](#), on page 414 for different ways to specify VHDL attributes and directives.

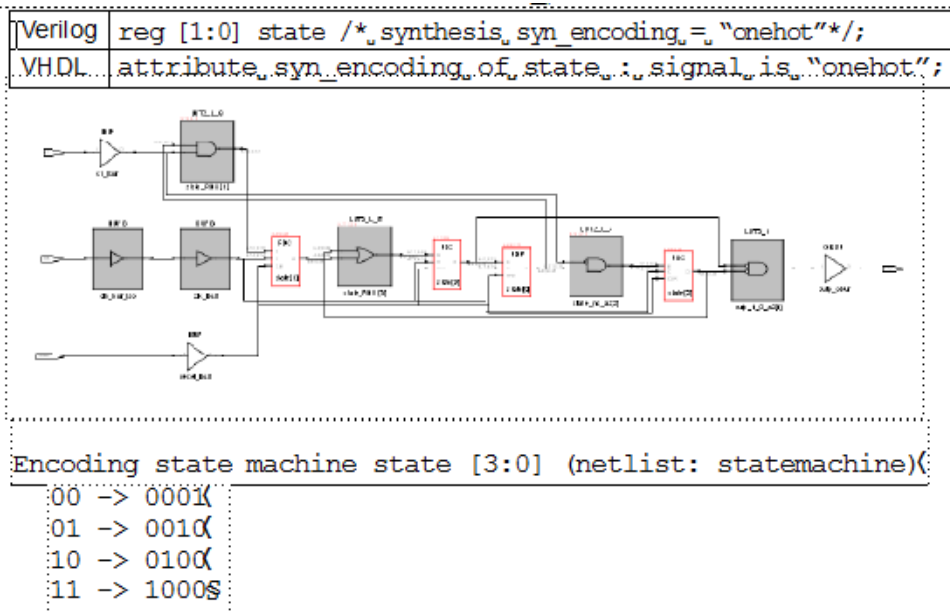
## Effect of Using `syn_encoding`

The following figure shows the default implementation of a state machine, with these encoding details reported:

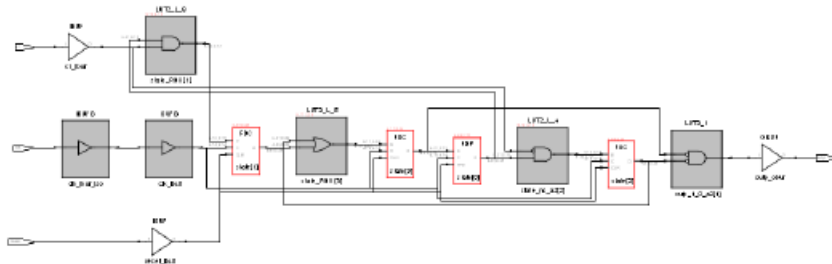
```
Encoding state machine state [3:0] (netlist: statemachine)
original code -> new code
  00 -> 00
  01 -> 01
  10 -> 10
  11 -> 11
```



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>

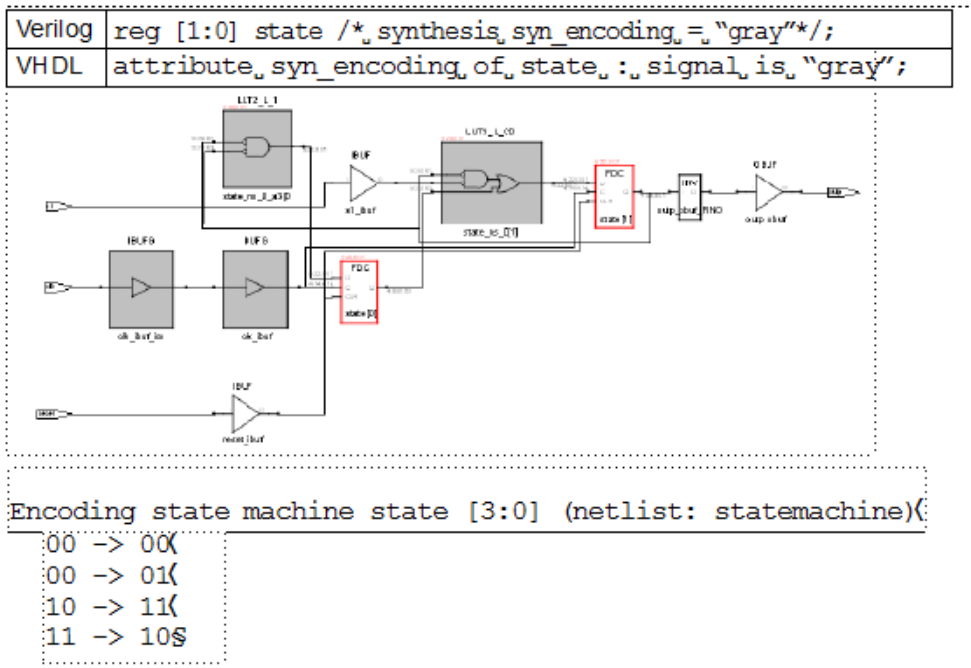


Encoding state machine state [3:0] (netlist: statemachine)

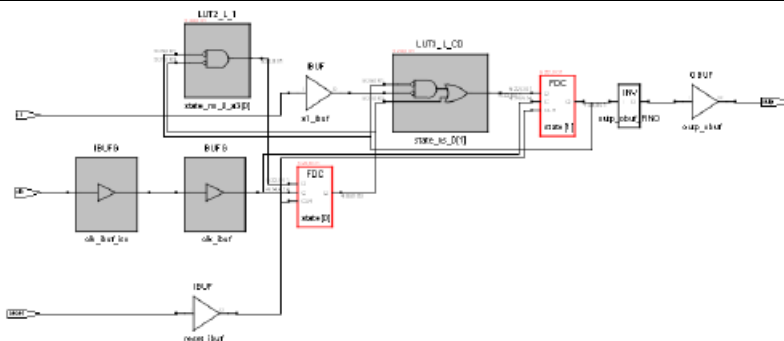
```

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	reg [1:0] state /* synthesis syn_encoding = "gray"*/;
VHDL	attribute syn_encoding of state : signal is "gray";



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

Value	Description
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="#">Example of syn_enum_encoding for User-Defined Encoding</a> , on page 81.

### 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 81](#) for more information. For step-by-step details about setting coding styles with this attribute see [Defining State Machines in VHDL, on page 410](#) of the *User Guide*.

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.

## Comparison of `syn_encoding` and `syn_enum_encoding`

<code>syn_encoding</code>	<code>syn_enum_encoding</code>
Attribute	Directive
Set on a state machine to specify a particular encoding	Set on VHDL enumerated data types only. If you use <code>syn_encoding</code> instead, you get a warning message (CD721).
Affects how the mapper implements state machines in the final netlist	Affects how the compiler interprets associated enumerated data types in VHDL; it is not automatically propagated to the implementation of the state machine.
Requires FSM Compiler to be enabled	Requires FSM Compiler to be disabled for the <code>syn_enum_encoding</code> value to be implemented in the final circuit.

## `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`, the Synopsys FPGA tools recognize `enum_encoding` and treat it just 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;
          0 : 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**

The following support applies for the syn\_enum\_encoding directive.

**Global Support    Object**

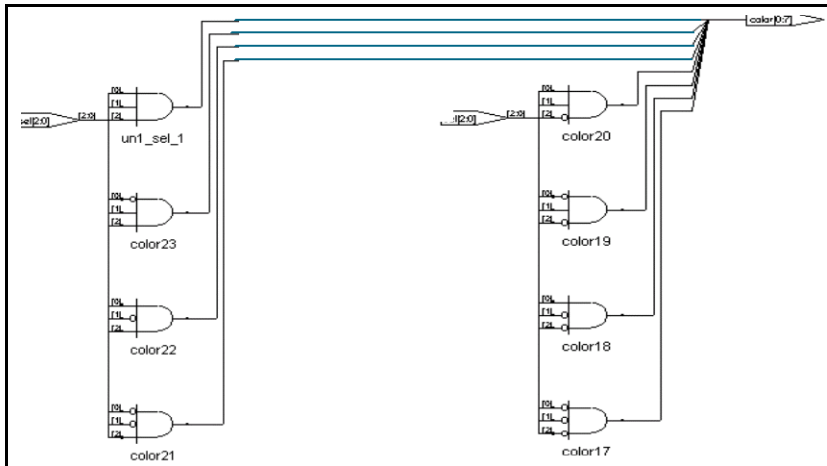
No/Yes	Enumerated data type.
--------	-----------------------

This table summarizes the syntax in the following file type:

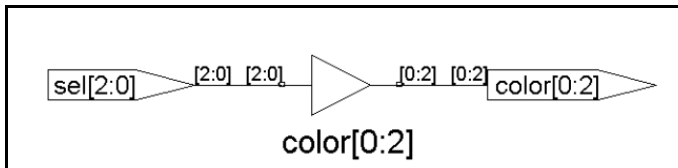
VHDL	attribute syn_enum_encoding of object : objectType is "value";	<a href="#">VHDL Example</a>
------	--	------------------------------

**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.

See the following section for the source code used to generate the schematics above.

## VHDL Example

See [VHDL Attribute and Directive Syntax, on page 414](#) for different ways to specify VHDL attributes and directives.

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;
```

## syn\_force\_pads

### Attribute

Prevents unused ports from being optimized.

Vendor	Technology
Lattice	LatticeECP3, ECP2S, ECP2M, ECP2; LatticeECP, EC; LatticeXP2, XP; LatticeSC, SCM; MachXO

### Description

Prevents unused ports from being optimized away to allow I/O pad insertion on the unused port. The attribute can be applied to individual ports or at the global level.

### syn\_force\_pads Values

Value	Default	Object	Description
0		port	Allows unused ports to be optimized.
1	Yes	port	Prevents unused ports to be optimized.

FDC	define_attribute { <i>object</i> } syn_force_pads {1   0}; define_global_attribute syn_force_pads {1   0};	<a href="#">FDC Example</a>
Verilog	<i>object</i> /* synthesis syn_force_pads = {1   0} */;	<a href="#">Verilog Example</a>
VHDL	attribute syn_force_pads of <i>object</i> : <i>objectType</i> is {true   false};	<a href="#">VHDL Example</a>

### FDC Example

```
define_attribute {object} syn_force_pads {1 | 0}
define_global_attribute syn_force_pads {1 | 0}
```

## Verilog Example

```
module test(input clk, a_in, b_in, c_in, d_in, output reg dout);
  //c_in and d_in are unconnected ports!

  always@(posedge clk)
    begin
        dout <= a_in & b_in;
    end
endmodule
```

## VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
entity test is
port (
    clk : in std_logic;
    a_in : in std_logic;
    b_in : in std_logic;
    c_in : in std_logic; --unconnected port
    d_in : in std_logic; --unconnected port
    d_out: out std_logic
);
end test;

architecture beh of test is
begin
    process (clk)
    begin
        if (clk'event and clk = '1') then
            d_out <= a_in and b_in;
        end if;
    end process;
end beh;
```

---

## Effect of Using `syn_force_pads`

When the attribute `syn_force_pads` is enabled, the output edn netlist contains pads attached to unconnected ports. For example:

```
(instance d_in_pad (viewRef PRIM (cellRef IBM (libraryRef LUCENT)))  
(instance c_in_pad (viewRef PRIM (cellRef IBM (libraryRef LUCENT)))
```

When `syn_force_pads` is disabled, the output edn netlist does not have pad referenced on the `c_in` and `d_in` unconnected ports.

## syn\_force\_seq\_prim

### Directive

Applies the “fix gated clocks” algorithm to the associated primitive.

### syn\_force\_seq\_prim Values

Value	Description
1	Applies the fixed gated clocks algorithm to the associated black-box primitive.
0	Does not apply the fixed gated clocks algorithm to the associated black-box primitive.

### Description

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 44 for a list of the associated directives.

For information about using this directive in working with gated clocks, see [Using Gated Clocks for Black Boxes](#), on page 613 of the *User Guide*.

### syn\_force\_seq\_prim Values Syntax

The following support applies for the `syn_force_seq_prim` directive.

Global Support	Object	Value
No	Module name of the black box	1 or 0



This table summarizes the syntax in the following file type:

Verilog	<i>object</i> /* synthesis syn_force_seq_prim = 1   0 */;	<a href="#">Verilog Example</a>
VHDL	attribute syn_force_seq_prim : boolean; attribute syn_force_seq_prim of <i>object</i> : architecture is true   false;	<a href="#">VHDL Example</a>

## Verilog Example

***object* /\* synthesis syn\_force\_seq\_prim = 1 \*/;**

where *object* is the module name of the black box.

```
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

**attribute syn\_force\_seq\_prim of *object*: *objectType* is true;**

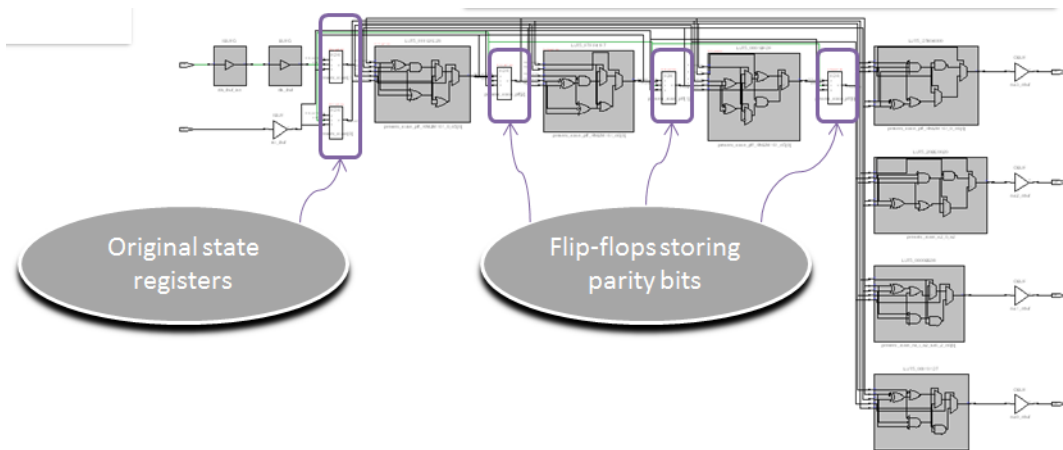
where *object* is the entity name of the black-box.

```
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\_gatedclk\_clock\_en

### *Directive*

Specifies the name of the enable pin inside a black box to support the fix gated clocks feature.

### syn\_gatedclk\_clock\_en Values

Value	Description
<i>enablePin</i>	Specifies the name of the enable pin within a black box that is to support the fixed gated clocks feature.

### Description

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 44](#) for a list of the associated directives.

For information about using this directive in working with gated clocks, see [Using Gated Clocks for Black Boxes, on page 613](#) of the *User Guide*.

### syn\_gatedclk\_clock\_en Values Syntax

The following support applies for the `syn_gatedclk_clock_en` directive.

Global Support	Object
No	Name of the enable pin in the black box.

This table summarizes the syntax in the following file type:

Verilog	<i>object</i> /* synthesis syn_gatedclk_clock_en = <i>enablePin</i> */;	<a href="#">Verilog Example</a>
VHDL	attribute syn_gatedclk_clock_en : string; attribute syn_gatedclk_clock_en of <i>object</i> : signal is <i>enablePin</i> ;	<a href="#">VHDL Example</a>

## Verilog Example

*object* /\* **synthesis syn\_gatedclk\_clock\_en = "value"** \*/;

where *object* is the module name of the black box and *value* is the name of the enable pin.

```
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

**attribute syn\_gatedclk\_clock\_en of *object*: *object*Type is *value*;**

where *object* is the entity name of the black-box.

See [VHDL Attribute and Directive Syntax, on page 414](#) for different ways to specify VHDL attributes and directives.

```
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";

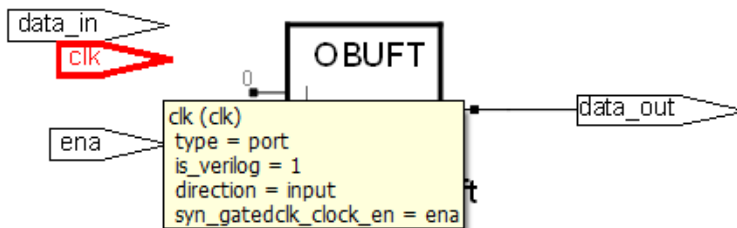
-- Other code

```

## Effect of Using syn\_gatedclk\_clock\_en

The `syn_gatedclk_clock_en` directive specifies the clock enable signal to use with the gated clock feature that defines timing for the black box. For example:

```
syn_gatedclk_clock_en = ena
```



## 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.

### syn\_gatedclk\_clock\_en\_polarity Values

Value	Description
1 (Default)	Indicates positive polarity of the enable signal (active high). If the attribute is not defined, the tool assumes a positive polarity by default.
0	Indicates negative polarity (active low).

### Description

If you do not set any 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 44 for a list of the associated directives.

### syn\_gatedclk\_clock\_en\_polarity Values Syntax

The following support applies for the `syn_gatedclk_clock_en_polarity` directive.

Global Support	Object
----------------	--------

Yes	Module name of the black box.
-----	-------------------------------

This table summarizes the syntax in the following file type:

## Verilog Example

Verilog	object /* synthesis syn_gatedclk_clock_en_polarity = 1   0 */;	<a href="#">Verilog Example</a>
VHDL	attribute syn_gatedclk_clock_en_polarity of object: objectType is true   false;	<a href="#">VHDL Example</a>

The following code segment provides an example.

```

module bbe1 (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;
  bbe1 u1 ( ena , clk_in , data_in , data_out );
  bbe2 u2 ( ena, clk_in2, data_in2, data_out2 );
endmodule

```

## VHDL Example

**attribute syn\_gatedclk\_clock\_en\_polarity of object: objectType is true | false;**

See [VHDL Attribute and Directive Syntax](#), on page 414 for different ways to specify VHDL attributes and directives.

```
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_bbel of bbel is
    attribute syn_black_box : boolean;
    attribute syn_black_box of arch_bbel: architecture is true;
    attribute syn_force_seq_prim of arch_bbel: architecture is true;
begin
end arch_bbel;

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.

Vendor	Devices
Lattice	iCE40, iCE40UP

### syn\_global\_buffers Values

Default	Global	Object
Maximum buffers available for a technology	Yes	Top-level module

Value	Description
An integer	Specifies an integer value for the number of global buffers.

### Description

The synthesis tool 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.

### Syntax Specification

FDC file	<code>define_attribute {view} syn_global_buffers {maximum}</code> <code>define_global_attribute syn_global_buffers {maximum}</code>
Verilog	<code>object /* synthesis syn_global_buffers = maximum */;</code>
VHDL	<code>attribute syn_global_buffers : integer;</code> <code>attribute syn_global_buffers of object : objectType is maximum;</code>

## SCOPE 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

```
define_global_attribute syn_global_buffers {10}
```

## Verilog Example

```
object /* synthesis syn_global_buffers = maximum */;
```

Here is a 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;

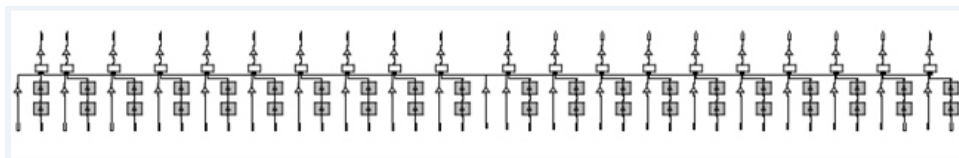
```

## Effect of Using syn\_global\_buffers

Before applying attribute:

Verilog Not applied

VHDL Not applied



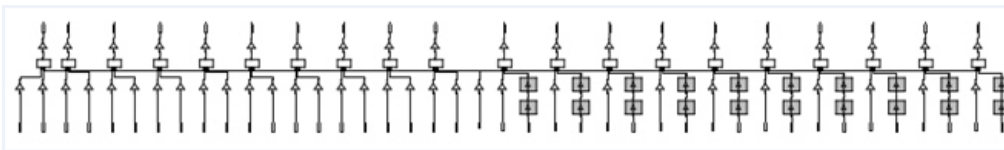
A message like the one below is generated:

```
@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) /* synthesis syn_global_buffers = 10 */;
```

```
VHDL attribute syn_global_buffers : integer;  
attribute syn_global_buffers of behave : architecture is  
10;
```



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 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.

Vendor	Devices
Lattice	ECP3, iCE40, iCE40UP and older families

### 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. For more information, see <a href="#">Using syn_hier fixed</a> , on page 108.

---

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 <code>macro</code> on the view (the <b>V:</b> 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="#">syn_netlist_hierarchy</a> , on page 144), set to <code>false</code>.</p> <p>You can use <code>flatten</code> in combination with other <code>syn_hier</code> values; the effects are described in <a href="#">Using syn_hier flatten with Other Values</a> , on page 115.</p> <p>If you apply <code>syn_hier</code> to a compile point, <code>flatten</code> 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 <code>flatten</code> 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	<pre>define_attribute {object} syn_hier {value} define_global_attribute syn_hier {flatten}</pre>
Verilog	<pre>object /* synthesis syn_hier = "value" */;</pre>
VHDL	<pre>attribute syn_hier of object : architecture is "value";</pre>

---

## SCOPE Example

	Object Type	Object	Attribute	Value	Val Type	Description	Comment
1	view		syn_hier	hard	string	Control hierarchy flattening	

```
define_global_attribute {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 109](#).

---

## 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.

- 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,
```

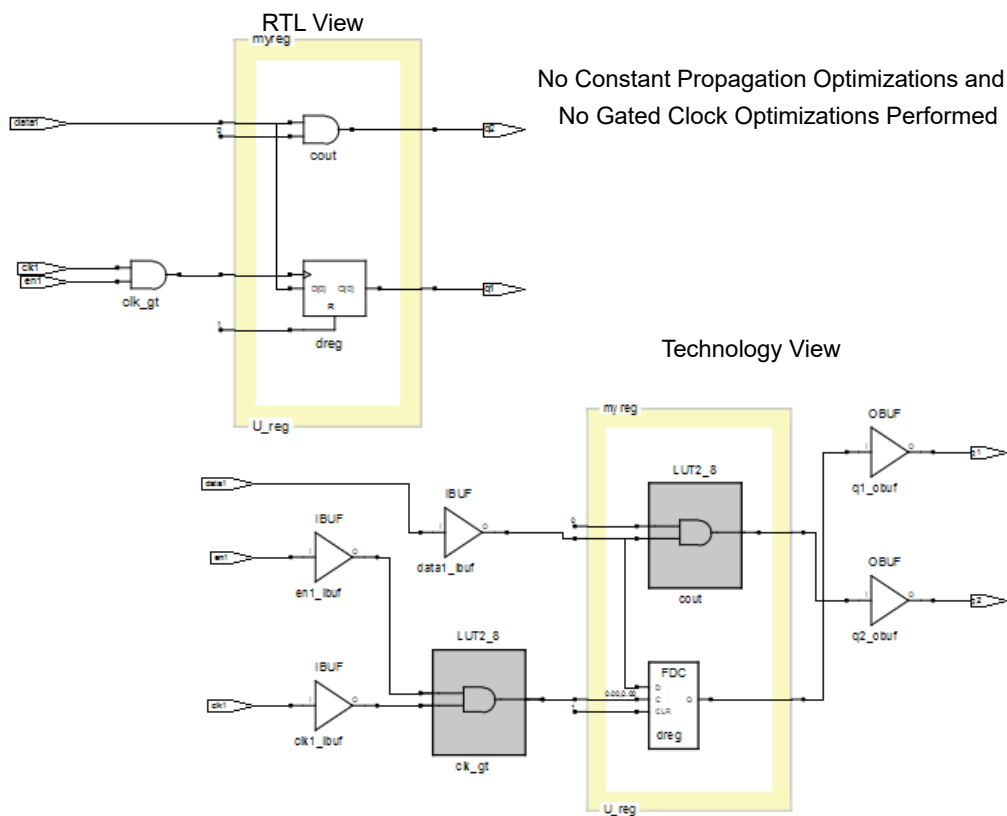
:

---

```
    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
```



```

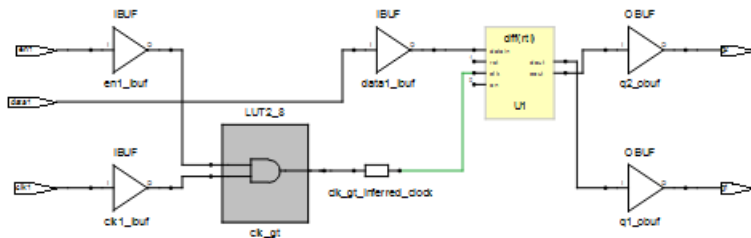
        process (clk, rst)
        begin
            if (rst = '1') then
                dreg<= '0';
            elsif (clk'event and clk ='1') then
                dreg<= datain;
            end if;
            dout <= dreg;
        end process;
    end;

```

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)

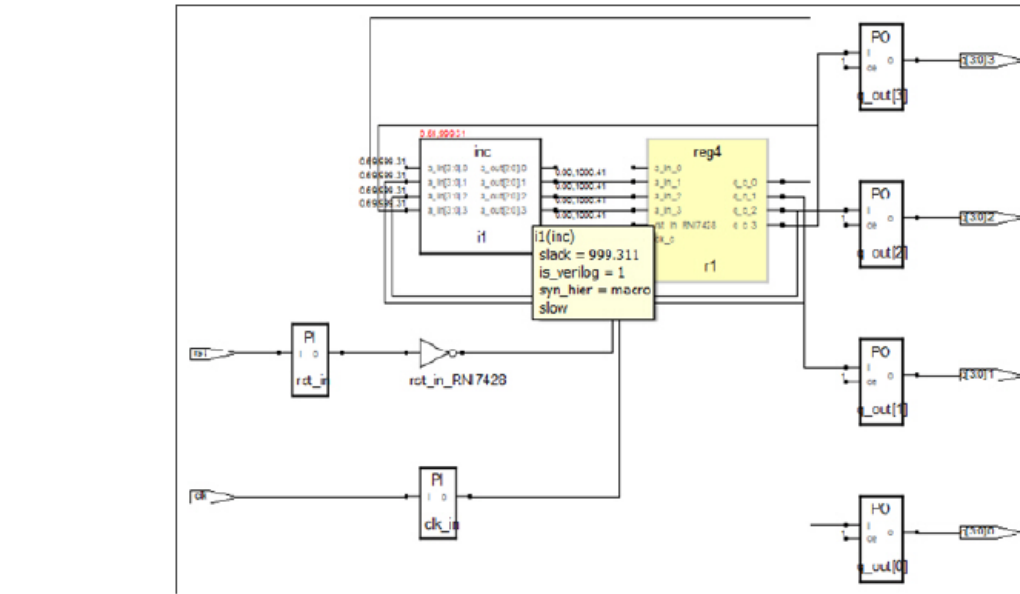
```

• •

After applying attribute with value *macro*:

```
Verilog    module inc(a_in, a_out) /* synthesis syn_hier = "macro" */;
```

```
VHDL      attribute syn_hier : string;
          attribute syn_hier of rtl: architecture is "macro";
```



## Using syn\_hier flatten with Other Values

You can combine `flatten` with other `syn_hier` values as shown below:

<code>flatten,soft</code>	Same as <code>flatten</code> .
<code>flatten,firm</code>	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.
<code>flatten,remove</code>	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 example 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\_insert\_buffer

### *Attribute*

Inserts a technology-specific clock buffer.

Vendor	Technologies
--------	--------------

Lattice	iCE40
---------	-------

### syn\_insert\_buffer Values

Vendor	Value	Description	Technology
Lattice	SB_GB_IO SB_GB	Use the value appropriate to the object: <ul style="list-style-type: none"><li>• Ports: SB_GB_IO</li><li>• Nets: SB_GB</li></ul>	iCE40

### 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. The synthesis tool inserts a technology-specific clock buffer. The object you attach the attribute to also varies with the vendor.

Vendor	Object	Description
Lattice	Port, net	Inserts a global clock buffer on a non-clock pin.

### syn\_insert\_buffer Syntax Specification

You cannot specify this attribute as a global value.

FDC	define_attribute object syn_insert_buffer value	<a href="#">FDC Example</a>
Verilog	object /* synthesis syn_insert_buffer = "value" */;	<a href="#">Lattice syn_insert_buffer Verilog Example</a>
VHDL	attribute syn_insert_buffer of object : objectType is "value";	<a href="#">VHDL Example</a>

## FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	<Global>	syn_insert_buffer	1	string	Insert a buffer on a signal

## Verilog Examples

Refer to the following syn\_insert\_buffer Verilog examples supported for various vendors.

### Lattice syn\_insert\_buffer Verilog Example

This section provides technology-specific examples.

```

module test
    (CLK, din1, din2, din3, din4, Q1, Q2, reset, gt1, gt2);

    input gt1, gt2;
    input CLK;
    input reset /* synthesis syn_insert_buffer = "SB_GB_IO" */;
    input din1;
    input din2 /* synthesis syn_insert_buffer = "SB_GB_IO" */;
    input din3 /* synthesis syn_insert_buffer = "SB_GB_IO" */;
    input din4;
    output reg Q1, Q2;

    wire gt1l /* synthesis syn_insert_buffer = "SB_GB" syn_keep = 1 */;

    assign gt1l = gt1;

    wire int_clk_glob;
    wire int_clk_core;

    wire int_clk_glob_gt;
    wire int_clk_core_gt;

```

:

---

```
reg reg_1, reg_2, reg_3, reg_4;

assign int_clk_glob_gt = CLK & gt11;
assign int_clk_core_gt = CLK & gt2;

always @(posedge int_clk_core_gt or negedge reset)
begin
    if (!reset)
        reg_1 <= 0;
    else
        begin
            reg_1 <= din1;
            reg_2 <= din2;
            Q1 <= reg_1 + reg_2;
        end
end

always @(posedge int_clk_glob_gt)
begin
    reg_3 <= din3;
    reg_4 <= din4;
    Q2 <= reg_3 + reg_4;
end

endmodule
```

This code specifies the `syn_insert_buffer` attribute, so the tool inserts `SB_GB_IO` buffers for the `reset`, `din2`, and `din3` ports. Without the attribute, these ports would use the `SB_IO` buffer and infer an `SB_GB` buffer on the `gt11` net.

## VHDL Example

The following shows a VHDL code snippet for this attribute:

```
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 rst : signal is "SB_GB_IO";
end prep2_2;
```

## syn\_insert\_pad

### *Attribute*

Removes an existing I/O buffer from a port or net when I/O buffer insertion is enabled.

Vendor	Technology
--------	------------

Lattice	iCE40, iCE40UP
---------	----------------

### syn\_insert\_pad Values

Value	Description	Default	Global	Object
0	Removes an IBUF/OBUF from a port or net	None	No	Port, net
1	Replaces a previously removed IBUF/OBUF on a port or net.	None	No	Port, net

### Description

The `syn_insert_pad` attribute is used when the Disable I/O Insertion option is not enabled (when buffers are automatically inserted) to allow users 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}

[SCOPE](#)  
[Example](#)

## SCOPE Example

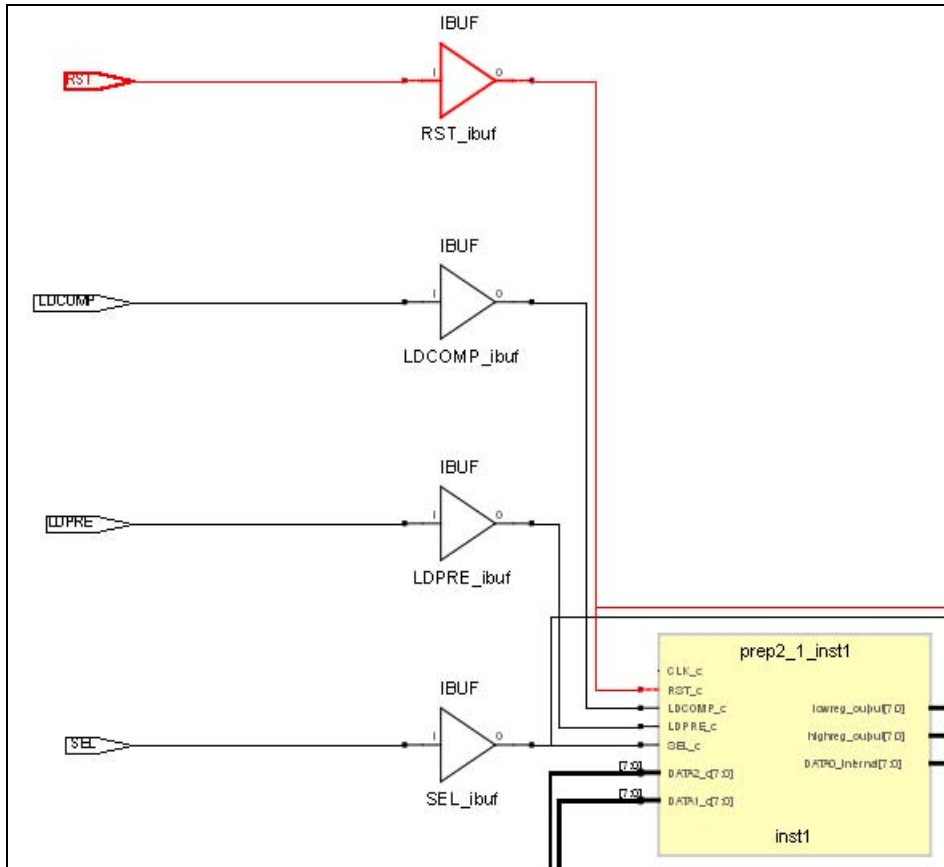
The following figure shows the attribute applied to the RST port using the SCOPE window:

Enable	Object Type	Object	Attribute	Value	Value Type
<input checked="" type="checkbox"/>	<any>	p:RST	syn_insert_pad	0	

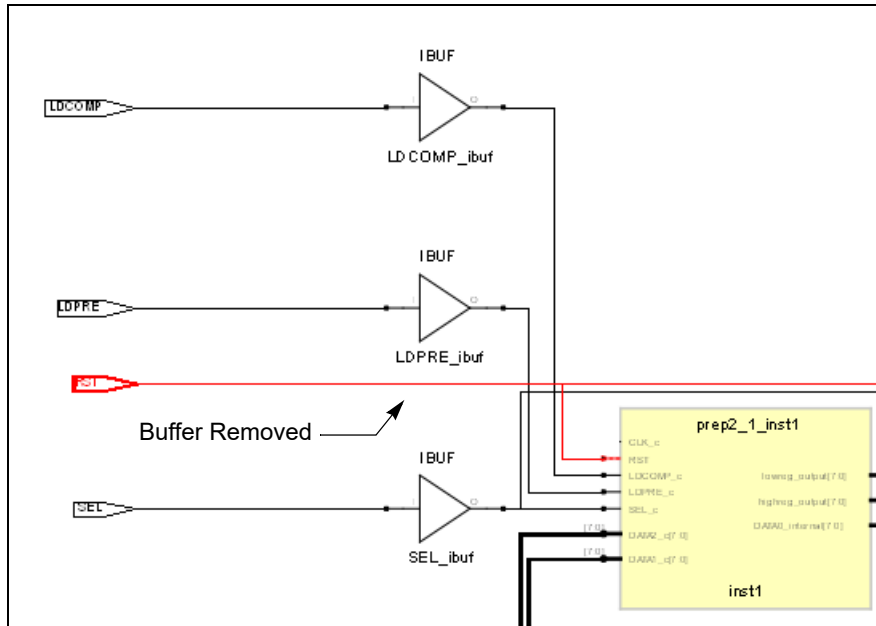
## Effect of Using syn\_insert\_pad

Original design before applying syn\_insert\_pad (or after applying syn\_insert\_pad with a value of 1 to replace a previously removed buffer).





Technology view after applying `syn_insert_pad` with a value of 0 to remove the original buffer from the RST input.



# syn\_isclock

## Directive

Specifies an input port on a black box as a clock.

## syn\_isclock Values

Value	Description	Object
1   true	Specifies input port is a clock.	Input port on a black box
0   false	Specifies input port is not a clock.	Input port on a black box

## Description

Used with the `syn_black_box` directive and specifies an input port on a black box as a clock. Use the `syn_isclock` directive to specify that an input port on a black box is a clock, even though its name does not correspond to one of the recognized names. Using this directive connects it to a clock buffer if appropriate. The data type is Boolean.

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 44 for a list of the associated directives.

## syn\_isclock Values Syntax

Verilog	<i>object</i> /* synthesis syn_isclock = 1 */;
VHDL	<b>attribute</b> syn_isclock of <i>object</i> : <i>objectType</i> 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

See [VHDL Attribute and Directive Syntax, on page 414](#) for different ways to specify VHDL attributes and directives.

```
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);
```

```

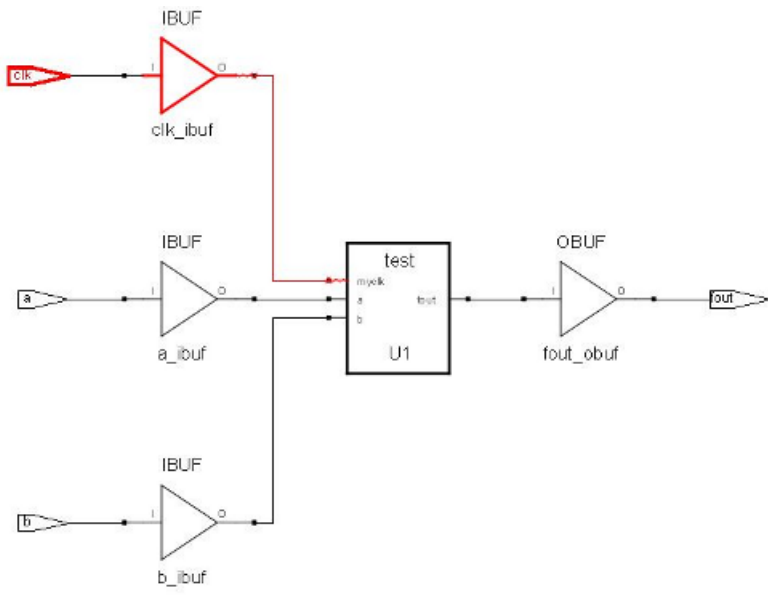
    a :    in std_logic_vector (size- 1 downto 0);
    b :    in std_logic_vector (size- 1 downto 0);
    myclk : in std_logic
  );
end component;

begin
  U1 : test port map (fout, a, b, clk);
end;

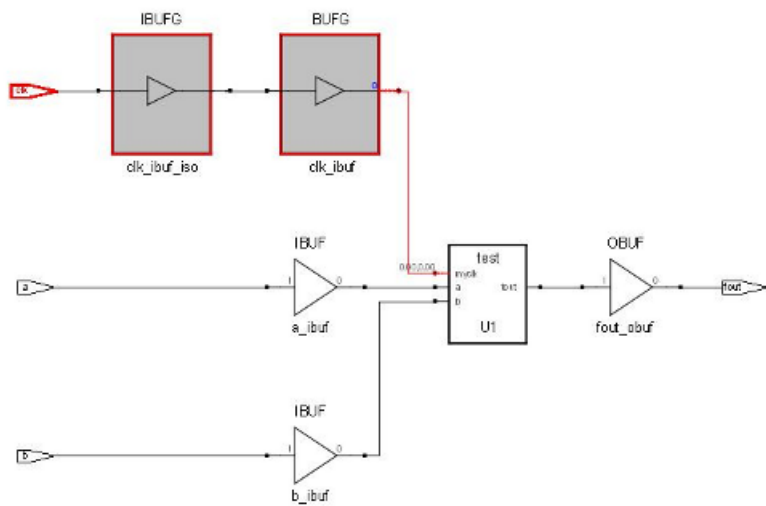
```

## Effect of Using syn\_isclock

This figure shows the HDL Analyst Technology view before using syn\_isclock:



This figure shows the HDL Analyst Technology view after using `syn_isclock`:



## syn\_keep

### *Directive*

Preserves the specified net and keeps it intact during optimization and synthesis.

Vendor	Technology	Global	Object
--------	------------	--------	--------

All	All	No	Net
-----	-----	----	-----

### syn\_keep Values

Value	Description
0   false (Default)	Allows nets to be optimized away.
1   true	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 132](#) 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 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;
```

For more information, see [Attribute Examples Using Verilog 2001 Parenthetical Comments](#), on page 135.

## syn\_keep and SystemVerilog Data Types

The `syn_keep` directive can be used for SystemVerilog data types, like logic, wire, or bit to preserve a net with the specified SystemVerilog data type. An example is provided below:

```
module test (input din1, din2, din3, input clk, output reg dout);  
  
  User defined data type  
  typedef logic signals;  
  
  struct {  
    signals A_1;  
    signals B_1;  
  } foo;  
  
  logic temp /* synthesis syn_keep = 1 */;
```



---

```

wire add;
assign add = din1 + din2;
assign temp= add /* synthesis syn_keep = 1 */;

always@(posedge clk)
begin
    dout <= temp;
end
endmodule

```

The following table shows examples of supported SystemVerilog data type assignments allowed with the `syn_keep` directive:

Assignment in always block, <code>syn_keep</code> works	<pre> 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; </pre>
Assignment outside always block, <code>syn_keep</code> does not work	<pre> 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; </pre>

For information about supported SystemVerilog data types, see [Data Types, on page 145](#).

## 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:

<code>syn_keep</code>	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.
<code>syn_preserve</code>	Ensures that registers are not optimized away.
<code>syn_noprune</code>	Ensures that a black box is not optimized away when its outputs are unused (i.e., when its outputs do not drive any logic).

See [Preserving Objects from Being Optimized Away, on page 445](#) in the *User Guide* for more information.

### `syn_keep` Syntax

Verilog	<code>object /* synthesis syn_keep = 1 */;</code>	<a href="#">Verilog Example</a>
VHDL	<code>attribute syn_keep : boolean attribute syn_keep of object : objectType is true;</code>	<a href="#">VHDL Example</a>

### Verilog Example

```
object /* synthesis syn_keep = 1 */;
```

`object` is a wire or reg declaration for combinational logic. Make sure that there is a space between the object name and the beginning of the comment slash (/).

Here is the source code used to produce the results shown in [Effect of Using `syn\_keep`, on page 132](#).

```
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

**attribute syn\_keep of *object* : *objectType* is true;**

*object* is a single or multiple-bit signal.

Here is the source code used to produce the schematics shown in [Effect of Using syn\\_keep, on page 132](#).

```

entity example2 is
    port (in1, in2 : in bit;
          clk : in bit;
          out1, out2 : out bit);
end example2;

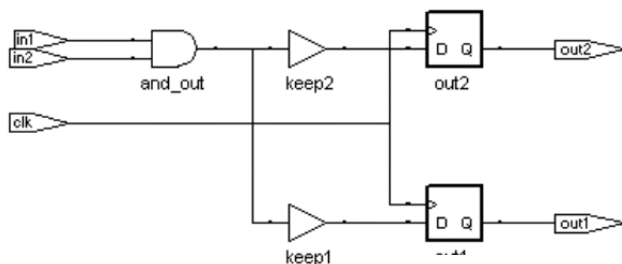
architecture rtl of example2 is
    attribute syn_keep : boolean;
    signal and_out, keep1, keep2: bit;
    attribute syn_keep of keep1, keep2 : signal is true;
begin
    and_out <= in1 and in2;
    keep1 <= and_out;
    keep2 <= and_out;
    process(clk)
    begin
        if (clk'event and clk = '1') then
            out1 <= keep1;
            out2 <= keep2;
        end if;
    end process;
end rtl;

```

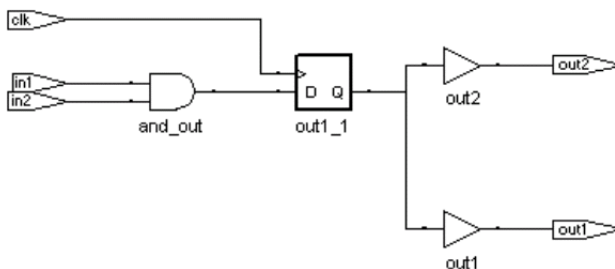
## Effect of Using syn\_keep

When you use `syn_keep` on duplicate logic, the tool retains it instead of optimizing it away. The following figure shows the Technology 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.



With `syn_keep`



Without `syn_keep`

# syn\_looplimit

*Directive*

*VHDL*

Specifies a loop iteration limit for while loops in the design.

## Description

VHDL only. For Verilog applications use the `loop_limit` directive (see [loop\\_limit](#), on page 28).

The `syn_looplimit` 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_looplimit` 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.

Alternatively, you can use the `set_option looplimit` command (Loop Limit GUI option) to set a global loop limit that overrides the default of 2000 loops. To use the Loop Limit option on the VHDL tab of the Implementation Options panel, see [VHDL Panel](#), on page 374 in the *Command Reference*.

## syn\_looplimit Summary

Technology	Global	Object
All	Yes	Architecture

## syn\_looplimit Syntax

VHDL      **attribute** `syn_looplimit` : *integer*;  
**attribute** `syn_looplimit` of *labelName* : **label is** *value*;

[VHDL Example](#)

## 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\_maxfan

### Attribute

Overrides the default (global) fanout guide for an individual input port, net, or register output.

Vendor	Technology	Default
Lattice	ECP3, ECP2S, ECP2M, ECP2 ECP/EC XP2/XP SC/SCM, MachXO	None

### syn\_maxfan Value

<i>value</i>	Integer for the maximum fanout
--------------	--------------------------------

### 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 Device panel on the Implementation Options dialog box or with the -fanout\_limit command. Use the syn\_maxfan attribute to 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.

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.

Similarly, you use `syn_maxfan` with the `syn_replicate` attribute in certain technologies to control replication.

## syn\_maxfan Syntax

### Global Object Type

No	Registers, instances, ports, nets
----	-----------------------------------

FDC	<b>define_attribute {object} syn_maxfan {integer}</b>	<a href="#">FDC Example</a>
Verilog	<b>object /* synthesis syn_maxfan = "value" */;</b>	<a href="#">Verilog Example</a>
VHDL	<b>attribute syn_maxfan of object : objectType is "value";</b>	<a href="#">VHDL Example</a>

## FDC Example

```
define_attribute {object} syn_maxfan {integer}
```

Enable	Object Type	Object	Attribute	Value	Value Type	Description
<input checked="" type="checkbox"/>	<any>	<Global>	syn_maxfan	1	integer	Overrides the default...

## Verilog Example

```
object /* synthesis syn_maxfan = "value" */;
```

For 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

**attribute syn\_maxfan of object : objectType is "value";**

See [VHDL Attribute and Directive Syntax, on page 414](#) for different ways to specify VHDL attributes and directives.

```

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';
            end if;
        end if;
    end process;
end architecture;

```

:

---

```
        else
            d <= not d;
        end if;
    end if;
end process;

process (d)
begin

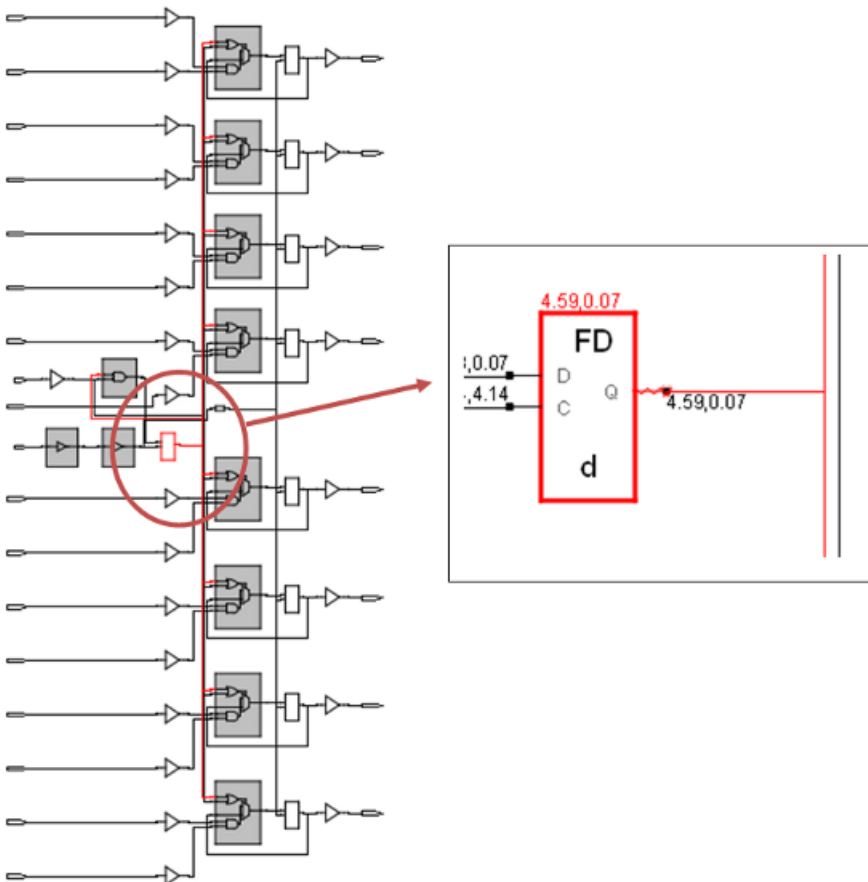
    if (d'event and d = '1') then

        c <= a and b;
    end if;
end process;

end rtl;
```

## Effect of Using syn\_maxfan

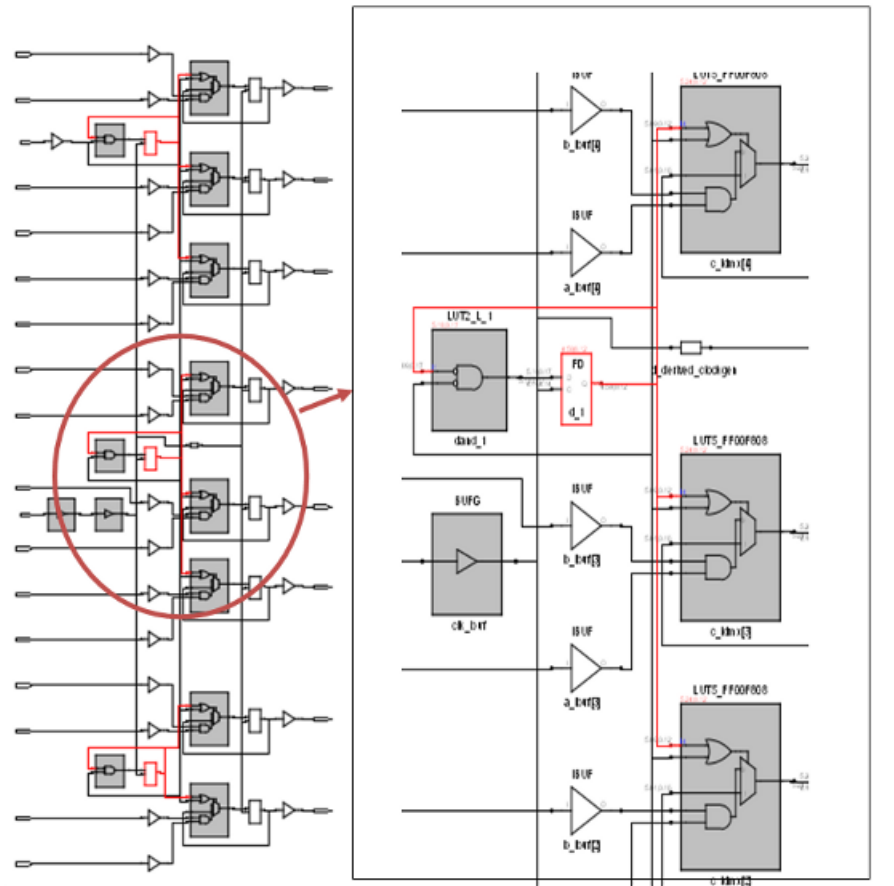
Before applying syn\_maxfan:



After applying the attribute `syn_maxfan`, the register `d` is replicated three times (shown in red) because its actual fanout is 8, but we have restricted it 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.

Vendor	Device	Values
Lattice	ECP3/ECP2S/ECP2M/ECP2 ECP/EC XP2/XP MACHXO	block_mult   logic
	iCE40UP	DSP   logic

### syn\_multstyle Values

Value	Description	Default
block_mult	Implements the multipliers as dedicated hardware blocks: Lattice—DSP blocks	X
logic	Implements the multipliers as logic.	-
DSP	Implements the multipliers as dedicated hardware blocks	

This table lists the valid values for each vendor:

Lattice	<ul style="list-style-type: none"> <li>• block_mult Uses dedicated hardware DSP blocks. This is the default.</li> <li>• logic Uses logic instead of dedicated resources.</li> </ul>
---------	---

### Description

This attribute specifies whether the multipliers are implemented as dedicated hardware blocks or as logic. The implementation varies with the technology, as shown in the preceding table.

## syn\_multstyle Syntax

### Global Attribute    Object

Yes	Module or instance
-----	--------------------

The following shows the attribute syntax when specified in different files:

FDC    `define_attribute {instance} syn_multstyle {block_mult | logic}`    [SCOPE Example](#)

Global attribute:

`define_global_attribute syn_multstyle {block_mult | logic}`

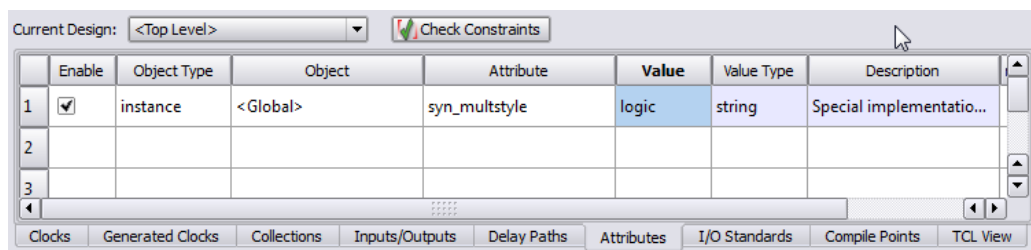
Verilog    `input net /* synthesis syn_multstyle = "block_mult | logic" */;`    [Verilog Example](#)

VHDL    `attribute syn_multstyle of instance : signal is "block_mult | logic";`    [VHDL Example](#)

See [VHDL Attribute and Directive Syntax](#), on page 414 for different ways to specify VHDL attributes and directives.

## SCOPE Example

This SCOPE example specifies that the multipliers be globally implemented as logic:



This 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 mult;

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.

Vendor	Technology
Lattice	EC, SC, XP families

### syn\_netlist\_hierarchy Values

Value	Description	Default
1/true	Allows hierarchy generation	Default
0/false	Flattens hierarchy in the netlist	

### Description

A global attribute that controls the generation of hierarchy in the 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

Global	Object
Yes	Module/Architecture

FDC	<b>define_global_attribute syn_netlist_hierarchy {0 1}</b>	<a href="#">SCOPE Example</a>
Verilog	<b>object /* synthesis syn_netlist_hierarchy = 0 1 */;</b>	<a href="#">Verilog Example</a>
VHDL	<b>attribute syn_netlist_hierarchy of object : objectType is true false;</b>	<a href="#">VHDL Example</a>



## SCOPE 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

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;

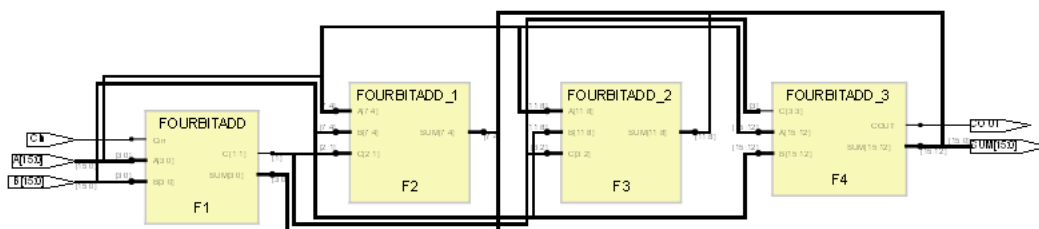
```

## 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\*/;

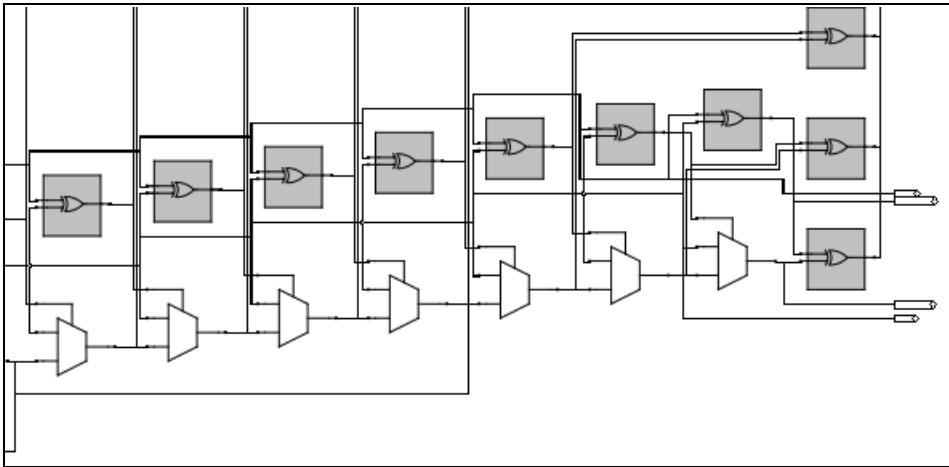
VHDL       attribute syn\_netlist\_hierarchy of adder\_structure :  
             architecture is true;



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 106](#) for information about this attribute.

## syn\_noarrayports

### Attribute

Specifies signals as scalar in the output file.

Vendor	Devices
Lattice	ECP3, iCE40, iCE40UP and older families

### syn\_noarrayports Values

Default	Global	Object
0	Yes	Module/Architecture

### Description

Use this attribute to specify that the ports of a design unit be treated as individual signals (scalars), not as buses (arrays) in the output file.

### Syntax Specification

SCOPE	define_global_attribute syn_noarrayports {0 1}
Verilog	object /* synthesis syn_noarrayports = 0   1;
VHDL	attribute syn_noarrayports of <i>object</i> : <i>objectType</i> is true   false;

### SCOPE 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;

---

(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;
------	---

---

```
(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.

Vendor	Technology
Lattice	all, including iCE40

### syn\_noclockbuf Values

Value	Description
0/false (Default)	Turns on clock buffering.
1/true	Turns off clock buffering.

### 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.

### Constraint File Syntax and Example

Global Support	Object
Yes	module/architecture

```
define_attribute {clock_port} syn_noclockbuf {0|1}
```

```
define_global_attribute syn_noclockbuf {0|1}
```

For example:

```
define_attribute {clk} syn_noclockbuf {1}
define_global_attribute syn_noclockbuf {1}
```

## FDC Example

The `syn_noclockbuf` attribute can be applied in the SCOPE 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 Syntax and Examples

```
object /* synthesis syn_noclockbuf = 1 | 0 */;

module ckbufg (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 Syntax and Examples

**attribute syn\_noclockbuf of *object* : *objectType* is true | false;**

```
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.

---

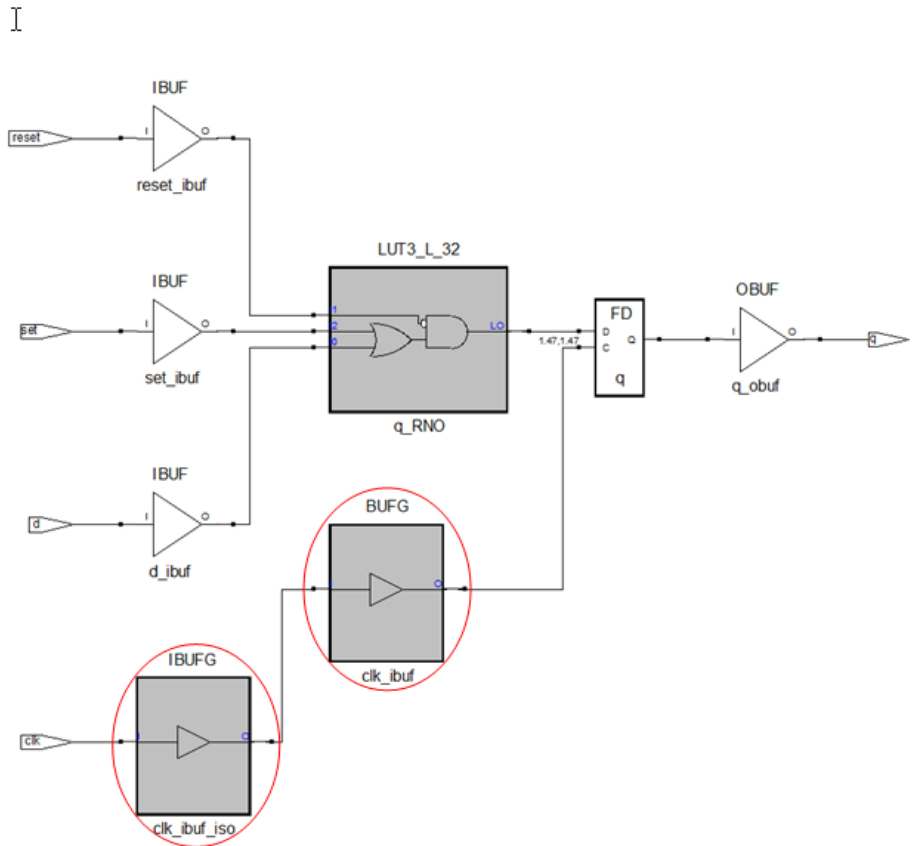
```
Verilog  input clk  /*synthesis syn_noclockbuf=0*/;
```

---

```
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.

:

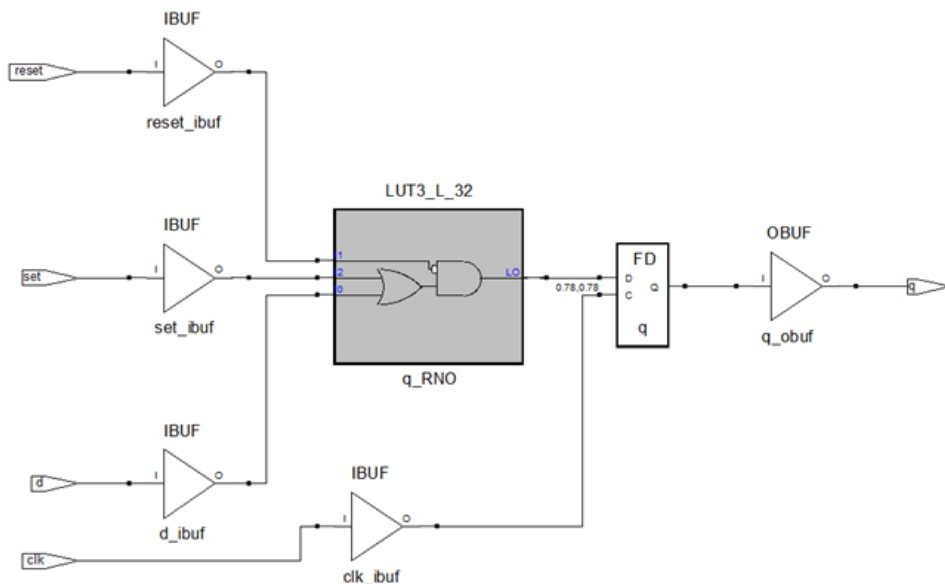
---

Verilog    `input clk   /*synthesis syn_noclockbuf=1*/;`

VHDL    `attribute syn_noclockbuf: Boolean;`  
`attribute syn_noclockbuf of clk : signal is true;`

---

No global buffers inferred



---

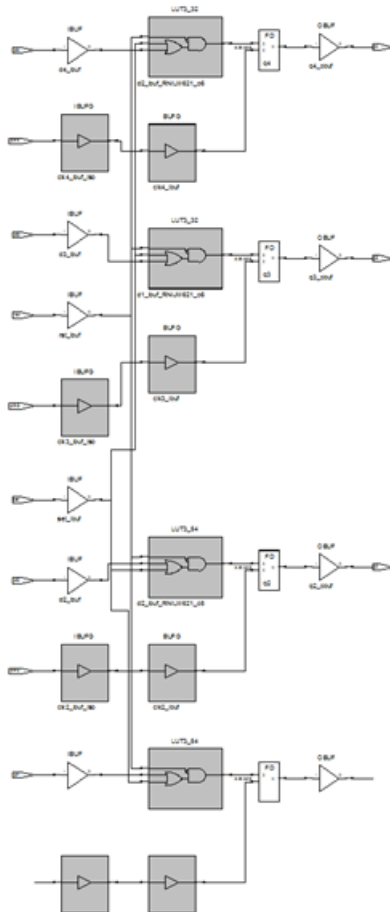
## Global Support

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.

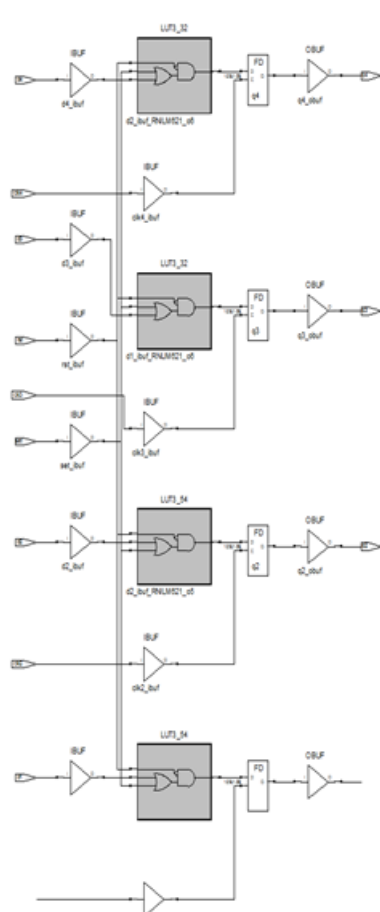
```
HDL  module
      ckbufg(d1,d2,d3,d4,clk1,clk2,clk3,clk4,rst,set,q1,q2,q3,q4)/*synthesis
      syn_noclockbuf=1*/;
```

```
FDC  define_global_attribute {syn_noclockbuf} {1}
```

Before Applying attribute



After applying attribute



## syn\_noclockpad

### Attribute

When you specify the `syn_noclockpad` attribute on a clock net connected to an input port, the software creates `SB_IO` and connects it to an input port.

Vendor	Technology
Lattice	iCE40 and iCE40LM families

### syn\_noclockpad Values

Global Support	Default	Object
Yes	0   false	Clock net

### Description

When you specify the `syn_noclockpad` attribute on a clock net connected to an input port, the software creates `SB_IO` and connects it to an input port. The software infers `SB_GB` and drives this buffer using the `SB_IO` connection. The `SB_GB` buffer is created if the number of global buffers already used in the design satisfies the resource limitation.

### syn\_noclockpad Syntax

The following table summarizes the syntax in different files.

FDC	<pre>define_attribute {p:clockName} syn_noclockpad {0   1} define_global_attribute syn_noclockpad {0   1}</pre>	<a href="#">FDC Example</a>
Verilog	<pre>object /* synthesis syn_noclockpad = "0   1"*/;</pre>	<a href="#">Example – Verilog syn_noclockpad</a>
VHDL	<pre>attribute syn_noclockpad : boolean; attribute syn_noclockpad of object : objectType is false   true;</pre>	<a href="#">Example – VHDL syn_noclockpad</a>



## FDC Example

Enable	Object Type	Object	Attribute	Value	Value Type	Description	Comment
<input checked="" type="checkbox"/>	<any>	p:CLK	syn_noclockpad	1	boolean	Convert SB_GB_IO to SB_IO and SB_GB	

```
define_attribute {p:CLK} {syn_noclockpad} {1}
```

## Example – Verilog syn\_noclockpad

```
// Example -- Verilog syn_noclock_pad

input CLK /* synthesis syn_noclockpad = 1 */;
input CLK2;

input [2:0] din1, din2, din3, din4;
output reg [2:0] Q1, Q2;

reg [2:0] reg_1, reg_2, reg_3, reg_4;

always @(posedge CLK)
begin
    reg_1 <= din1;
    reg_2 <= din2;
    Q1 <= reg_1 + reg_2;
end

always @(posedge CLK2)
begin
```

:

---

```
    reg_3 <= din3;
    reg_4 <= din4;
    Q2 <= reg_3 + reg_4;
end
endmodule
```

There are two clocks in this example; the `syn_noclockpad` attribute has been set on `CLK` so it does not use the buffer `SB_GB_IO`, but uses `SB_IO` with `SB_GB` buffers instead.

### Example – VHDL `syn_noclockpad`

```
-- Example -- VHDL syn_noclock_pad

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test is
port(CLK, CLK2 : in std_logic;
      din1, din2, din3, din4 : in std_logic_vector(2 downto
0);
      Q1, Q2 : out std_logic_vector(2 downto 0)

);

attribute syn_noclockpad : boolean ;
attribute syn_noclockpad of CLK : signal is true;
end test;
```

---

architecture rtl of test is

```
signal reg_1, reg_2, reg_3, reg_4 : std_logic_vector(2 downto 0);
begin
```

```
process(CLK)
```

```
begin
```

```
  if (CLK'event and CLK = '1') then
```

```
    reg_1 <= din1;
```

```
    reg_2 <= din2;
```

```
    Q1 <= reg_1 + reg_2;
```

```
  end if;
```

```
end process;
```

```
process(CLK2)
```

```
begin
```

```
  if (CLK2'event and CLK2 = '1') then
```

```
    reg_3 <= din3;
```

```
    reg_4 <= din4;
```

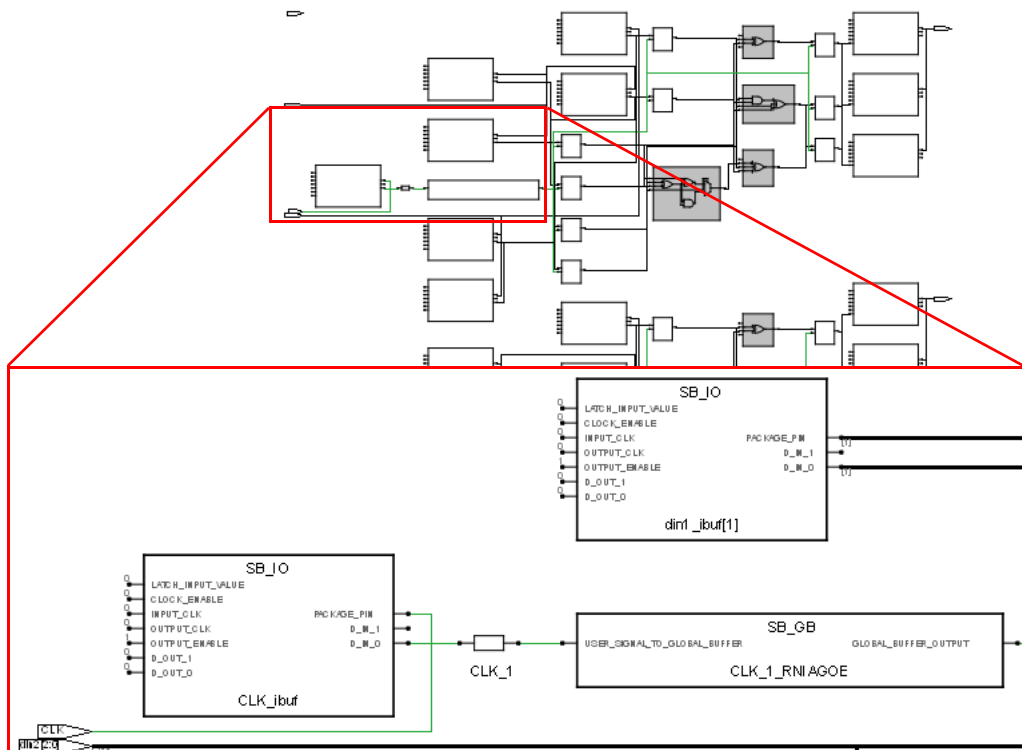
```
    Q2 <= reg_3 + reg_4;
```

```
  end if;
```

```
end process;
```

```
end rtl;
```

The `syn_noclockpad` attribute is specified on the input CLK, such that the software creates SB\_IO and connects it to an input port. The SB\_IO is used with SB\_GB buffers, as shown in the Technology view below.



## syn\_noprune

### *Directive*

Prevents optimizations for instances and black-box modules (including technology-specific primitives) with unused output ports.

Vendor	Technology	Global	Object
All	All	No	Verilog module/instance VHDL architecture/component

### syn\_noprune Values

Value	Description
0   false (Default)	Allows instances and black-box modules with unused output ports to be optimized away.
1   true	Prevents optimizations for instances and black-box modules 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, the synthesis tool 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 130](#), and [Preserving Objects from Being Optimized Away](#), on [page 445](#) in the *User Guide*.

## syn\_noprune Syntax

Verilog	<i>object</i> /* synthesis syn_noprune = 1 */;	<a href="#">Verilog Examples</a>
VHDL	attribute syn_noprune : boolean attribute syn_noprune of <i>object</i> : <i>objectType</i> is true;	<a href="#">VHDL Examples</a>

## Verilog Examples

This section contains code snippets and examples.

### Verilog Example 1: Module Declaration

```
// Verilog Example 1 -- Module Declaration

//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
```

`syn_noprune` 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.

```
module top (a,b,c,d,x,y); /* synthesis syn_noprune=1 */;

// Other code
```

The results for this example are shown in [Effect of Using `syn\_noprune`: Example 1, on page 175](#).

## Verilog 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 comma (,) 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 the instance `my_design2` will be removed if the output port is not mapped.

## Verilog Example 2: Hierarchical Design

```
// Verilog Example 2: Hierarchical Design
```

```
//Leaf level module

module subl (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 topl (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
```



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 176](#).

## VHDL Examples

This section contains 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 3: Component Declaration

```
-- VHDL Example 3: 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;

  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\*: Example 3, on page 178](#).

## VHDL Example: Component Instance Declaration

```
-- VHDL Example: Component Instance 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 `syn_noprune` directive works the same on component instances as with a component declaration.

---

```
-- 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`: Example 4, on page 178](#).

## Mixed Language Example

The `syn_noprune` directive can be specified on a module or architecture in a mixed Verilog and VHDL design.

### Example 5: Mixed Language Design

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;

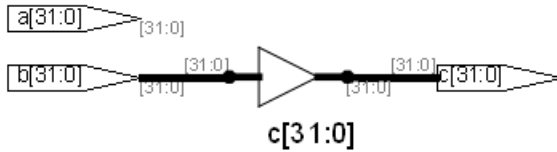
architecture behave of sub is
attribute syn_hier : string;
attribute syn_hier of behave : architecture is "hard";
begin
    x <= a and b;
    y <= c and d;
end behave;
```

The results for this example are shown in [Effect of Using `syn\_noprune` in a Mixed Language Design, on page 179](#).

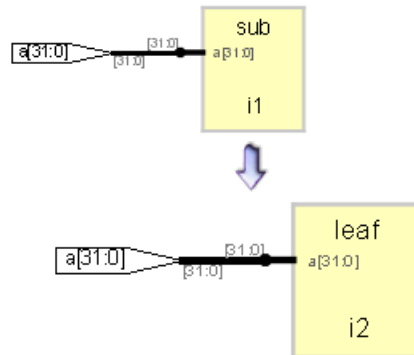
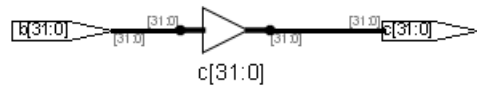
## Effect of Using syn\_noprune: Example 1

The following RTL 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`

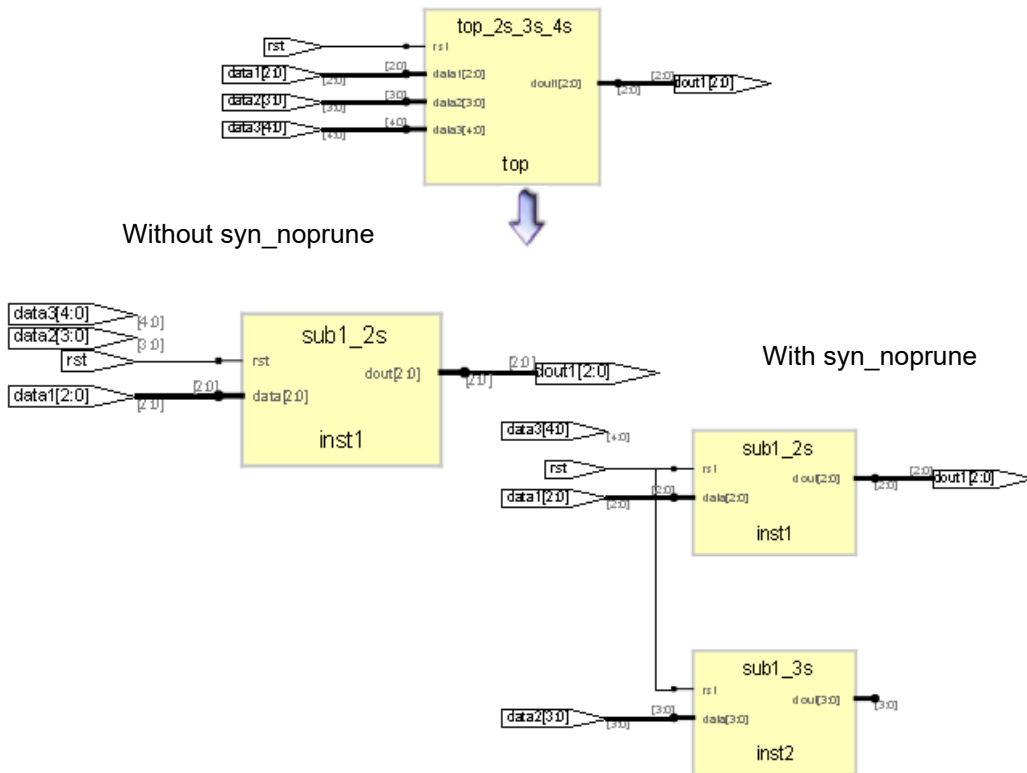


With `syn_noprune`



## 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 il (a);
endmodule
```



```

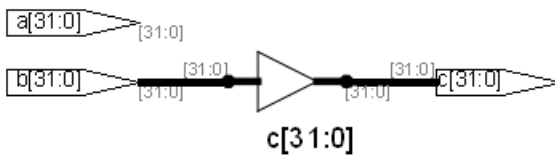
//Hier1
module sub (input int a);
interm1 i2 (a);
endmodule

//Hier2
module interm1 (input int a) /* synthesis syn_noprune=1*/;
interm2 i3 (a);
endmodule

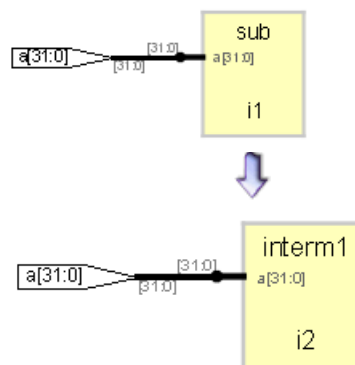
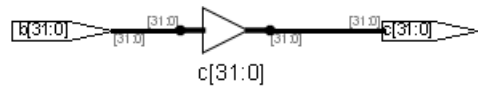
//Hier3
module interm2 (input int a);
leaf i4 (a);
endmodule

```

Without syn\_noprune

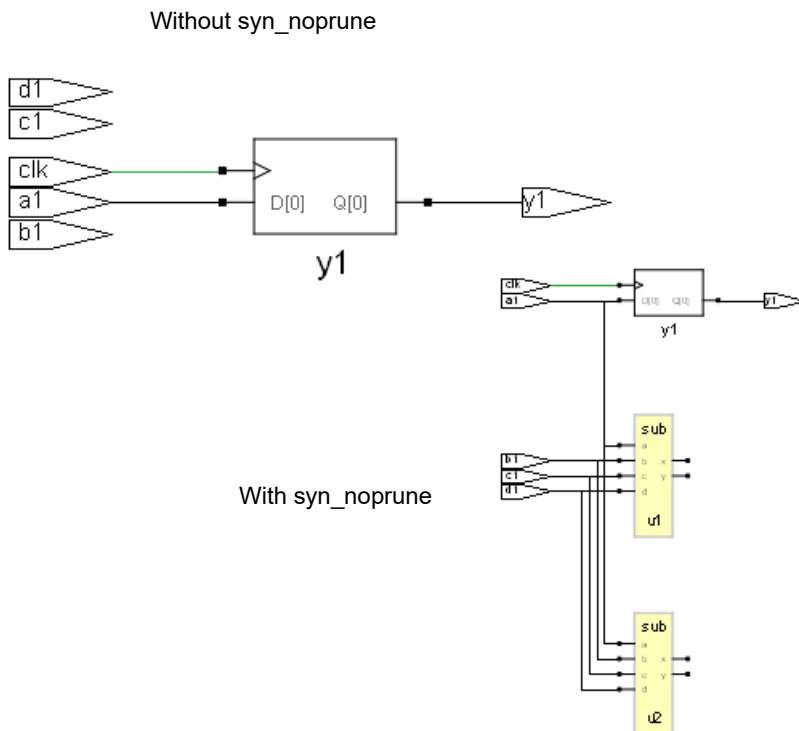


With syn\_noprune



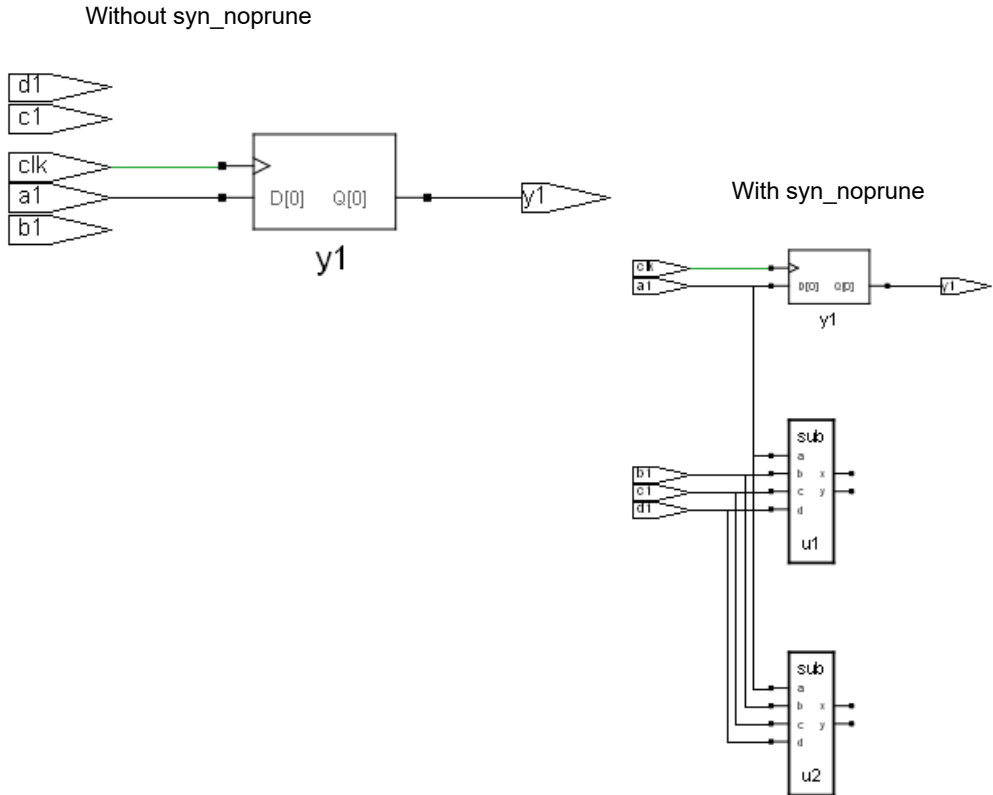
### Effect of Using syn\_noprune: Example 3

The following RTL 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 RTL views show that the instance and black box module are not optimized away when `syn_noprune` is applied.

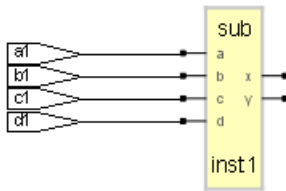
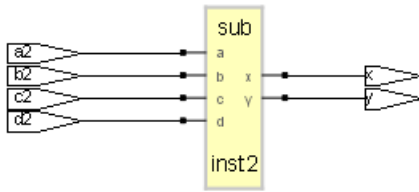


## Effect of Using syn\_noprune in a Mixed Language Design

The following RTL view shows that the design hierarchy is preserved when the syn\_noprune directive is applied on sub.

:

---



## syn\_pad\_type

### Attribute

Specifies an I/O buffer standard.

Vendor	Technology
Lattice	iCE40 family

### syn\_pad\_type Values

Value	Description
<code>{buffer}_{standard}</code> For example: <code>IBUF_LVCMOS_18</code>	Specifies the port I/O standard.

### Description

Specifies an I/O buffer standard. Refer to [I/O Standards, on page 174](#) and to the vendor-specific documentation for a list of I/O buffer standards available for the selected device family.

### syn\_pad\_type Syntax

Default	Global Attribute	Object
Not Applicable	No	Port
FDC	<b>define_io_standard -default portType {port} -delay_type portType syn_pad_type {io_standard}</b> For example: <code>define_io_standard {p} -delay_type output syn_pad_type {LVCMOS_18}</code>	<a href="#">FDC Example</a>
Verilog	<b>object !* synthesis syn_pad_type = io_standard *!</b>	<a href="#">Verilog Example</a>
VHDL	<b>attribute syn_pad_type of object : objectType is io_standard;</b>	<a href="#">VHDL Example</a>

## FDC Example

	Enable	Object Type	Object	Attribute	Value
1	<input checked="" type="checkbox"/>	port	p:output	syn_pad_type	LVCMOS_18
2					

**-default\_portType** *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 to them.

**-delay\_type portType** *PortType* can be input, output, or bidir.

**syn\_pad\_type {io\_standard}** Specifies I/O standard (see 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 CTT pad type	define_io_standard -default_bidir -delay_type bidir syn_pad_type {CTT}

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}
```

## 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      (instance m_2_4 (viewRef PRIM (cellRef LUT2_L (libraryRef VIRTEX)))
97        (property INIT (string "4'h5"))
98      )
99      (instance P_2_2 (viewRef PRIM (cellRef LUT2_L (libraryRef VIRTEX)))
100        (property INIT (string "4'h6"))
101      )
102      (instance P_obuf (viewRef PRIM (cellRef OBUF (libraryRef VIRTEX)))
103        (property IOSTANDARD (string "LVCMOS18"))
104      )
105      (instance PC_ibuf (viewRef PRIM (cellRef IBUF (libraryRef VIRTEX)))
106      )
```

### 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\_pipeline

## Attribute

Permits registers to be moved to improve timing.

Vendor	Technologies
Lattice	iCE40 and older

## syn\_pipeline Values

Value	Default	Global	Object	Description
0   False		Yes	Registers	Disables pipelining.
1   True	Default	Yes	Registers	Allows pipelining.

## 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 Fast Synthesis option.

## syn\_pipeline Syntax Specification

FDC	define_attribute { <i>register</i> } syn_pipeline {0 1}	<a href="#">FDC Example</a>
Verilog	<i>object</i> /* synthesis syn_pipeline = {1 0} */;	<a href="#">Verilog Example</a>
VHDL	attribute syn_pipeline of <i>object</i> : <i>objectType</i> is {true false};	<a href="#">VHDL Example</a>

## FDC Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	register	itemp2[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;

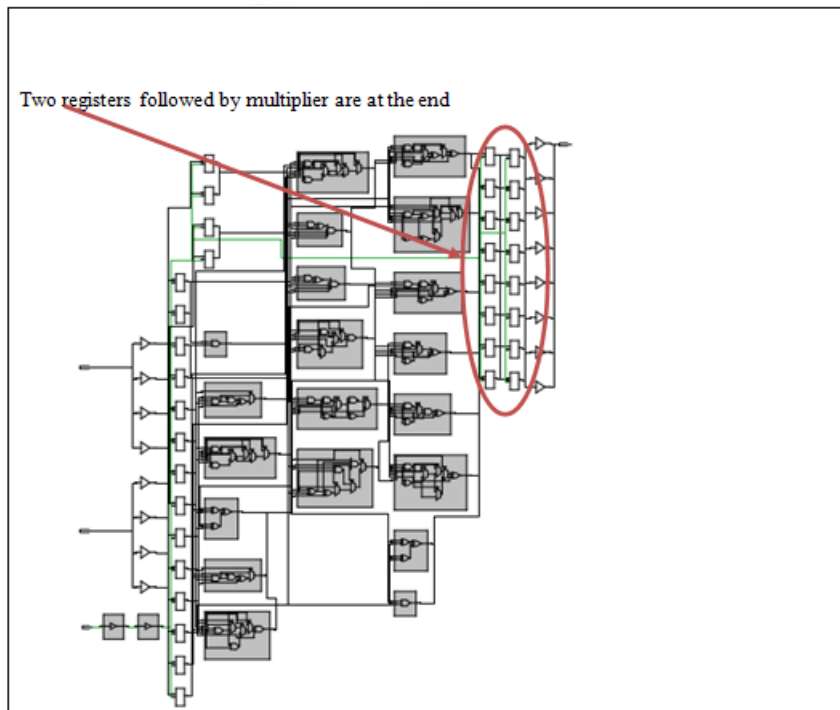
architecture rtl of pipeline is
signal a_reg : std_logic_vector(3 downto 0);
signal b_reg : std_logic_vector(3 downto 0);
signal temp1 : std_logic_vector(7 downto 0);
signal temp2 : std_logic_vector(7 downto 0);
signal temp3 : std_logic_vector(7 downto 0);
attribute syn_pipeline : string;
attribute syn_pipeline 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:

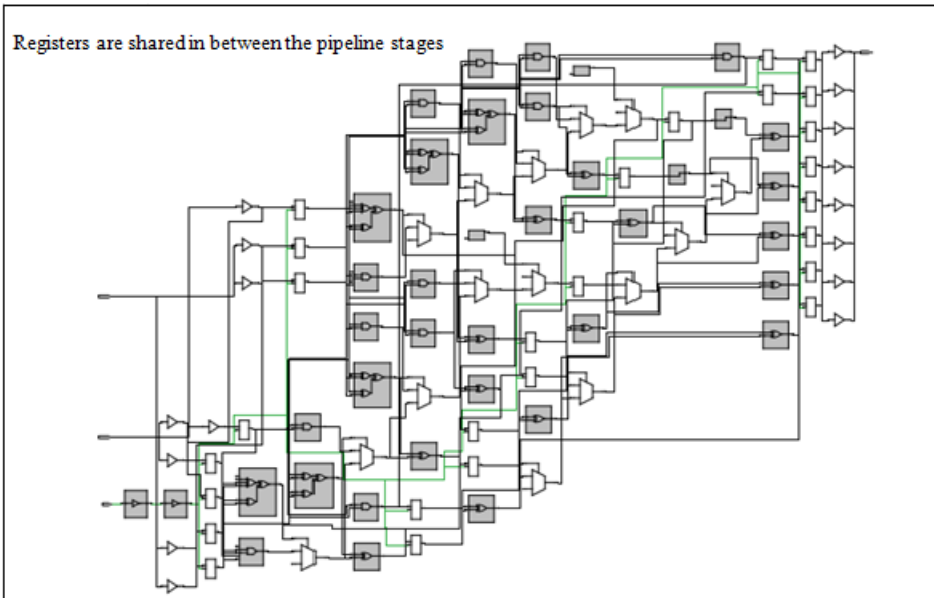
Verilog	reg [7:0] temp2/* synthesis syn_pipeline = 0 */;
VHDL	attribute syn_pipeline of temp2 : signal is "false";



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_pipeline of temp2 : signal is "true";`



## syn\_preserve

### *Directive*

Prevents sequential optimizations such as constant propagation, inverter push-through, and FSM extraction.

Technology	Global	Object
All	Yes	Register definition signal, module (Verilog) Output port or internal signal that holds the value of the register or architecture (VHDL)

### syn\_preserve Values

Value	Description
1   true	Preserves register logic.
0   false (Default)	Optimizes registers as needed.

### Description

The `syn_preserve` directive controls whether 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, the synthesis tool 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 find the 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 130](#), and [Preserving Objects from Being Optimized Away, on page 445](#) in the *User Guide*.

## syn\_preserve Syntax

Verilog	<code>object /* synthesis syn_preserve = 0   1 */</code>	<a href="#">Verilog Example</a>
VHDL	attribute <code>syn_preserve</code> of <code>object</code> : <code>objectType</code> is true   false;	<a href="#">VHDL Examples</a>

## 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 193](#).

```
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 some VHDL code examples:

### Example 1

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

entity simplifiedff 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 simplifiedff;

architecture behavior of simplifiedff 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 193](#).

```
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;

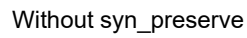
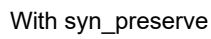
architecture behave of mod_preserve is
  attribute syn_preserve : boolean;
  attribute syn_preserve of behave: architecture is true;
  signal reg1 : std_logic;
  signal reg2 : std_logic;
begin
  process
  begin
    wait until clk'event and clk = '1';
    reg1 <= in1 and in2;
    reg2 <= in1 and in2;
    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

Value	Description
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. 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.

The table below shows how to apply syn\_probe values to nets, buses, and bus slices. It indicates what port names will 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] test_pt[1] 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, test_pt_0, test_pt_1	If a syn_probe value without brackets is applied to a bus, the port names are adjusted.

## syn\_probe Syntax

Global	Object	Default
No	Net	None

The following table shows the syntax used to define this attribute in different files:

FDC	<b>define_attribute {n:netName} syn_probe {probePortname 1 0}</b>	<a href="#">FDC Example</a>
Verilog	<b>object /* synthesis syn_probe = "string"   1   0 */;</b>	<a href="#">Verilog Example</a>
VHDL	<b>attribute syn_probe of object : signal is "string"   1   0;</b>	<a href="#">VHDL Example</a>

## FDC Example

The following examples insert a probe signal into 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}
```

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;

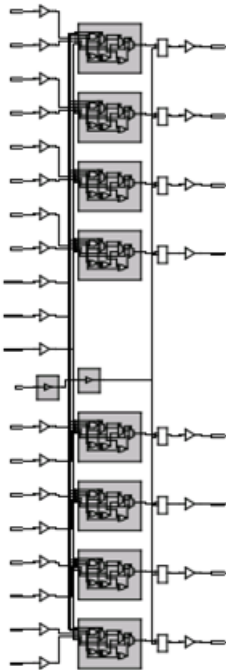
end rtl;

```

## Effect of Using syn\_probe

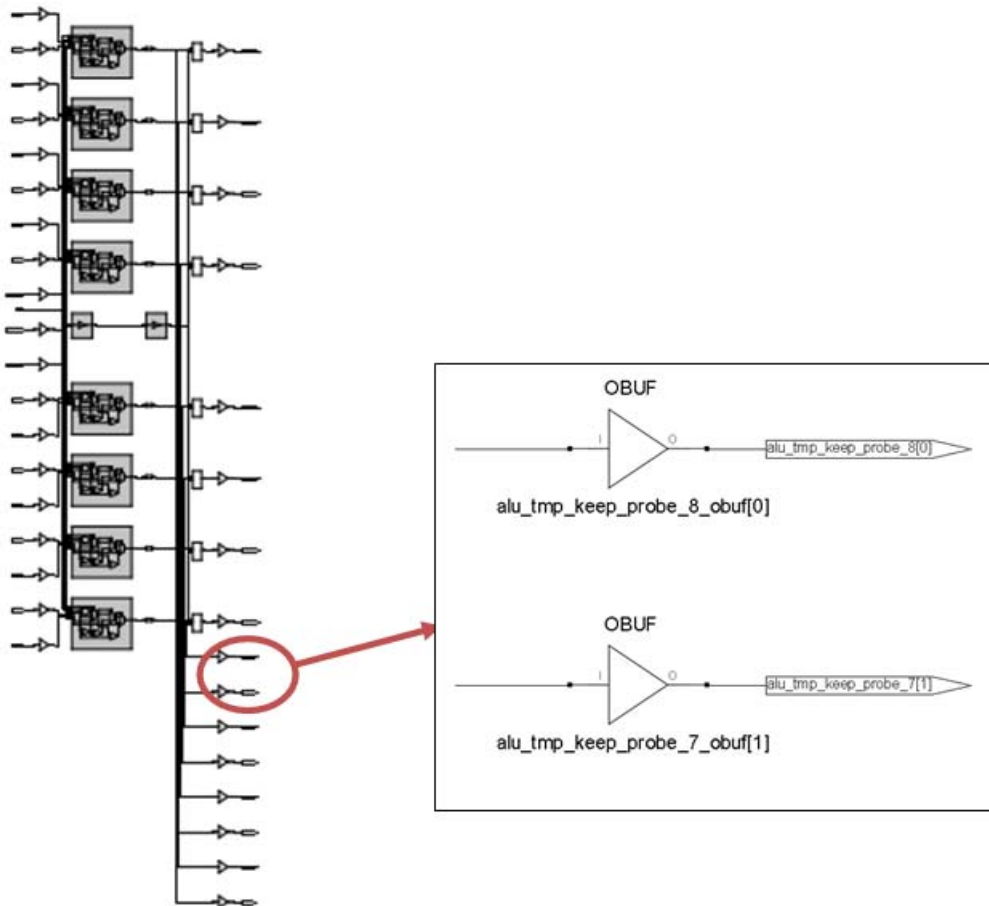
Before applying syn\_probe:

Verilog	<code>reg [7:0] alu_tmp /* synthesis syn_probe="0"*/</code>
VHDL	<code>attribute syn_probe of alu_tmp : signal is "0";</code>



After applying syn\_probe with 1:

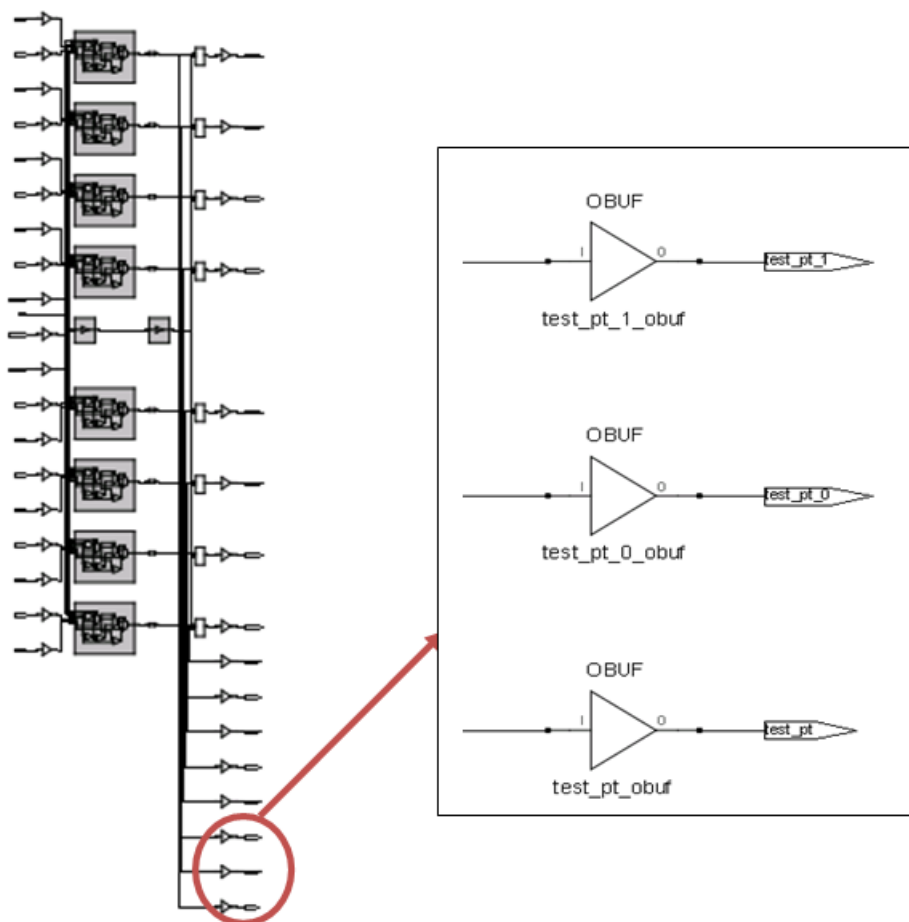
Verilog	<code>reg [7:0] alu_tmp /* synthesis syn_probe="1"*/</code>
VHDL	<code>attribute syn_probe of alu_tmp : signal is "1";</code>



After applying syn\_probe with test\_pt:

Verilog	reg [7:0] alu_tmp /* synthesis syn_probe="test_pt"*/
VHDL	attribute syn_probe of alu_tmp : signal is "test_pt";

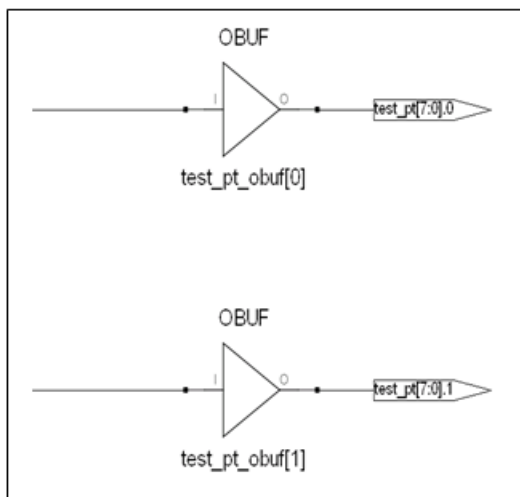
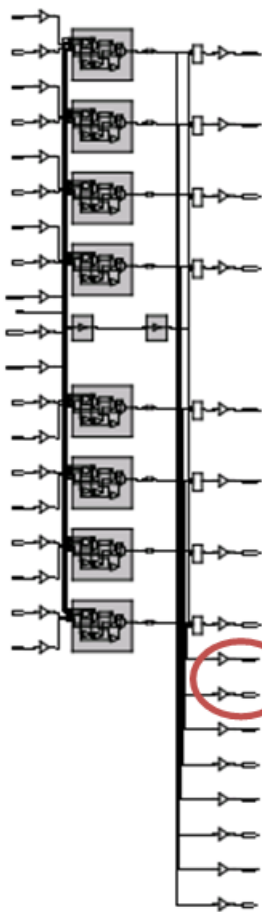




After applying syn\_probe with test\_pt[]:

Verilog	reg [7:0] alu_tmp /* synthesis syn_probe="test_pt[]" */
---------	---

VHDL	attribute syn_probe of alu_tmp : signal is "test_pt[]";
------	---



## syn\_ramstyle

### Attribute

Specifies the implementation for an inferred RAM.

Vendor	Devices
Lattice	ECP3, older devices iCE40, iCE40UP

### syn\_ramstyle Values

Default	Global Attribute	Object
block_ram	Yes	View, module, entity, RAM instance

The values for `syn_ramstyle` vary with the target technology. The following table lists all the valid `syn_ramstyle` values, some of which apply only to certain technologies. For details about using `syn_ramstyle`, see [RAM Attributes, on page 415](#) in the *User Guide*.

block_ram	<p>Specifies that the inferred RAM be mapped to the appropriate device-specific memory. It uses dedicated memory resources in the FPGA.</p> <p>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.</p> <p>Alternatively, you can specify a <code>ramType</code> value. See <a href="#">RAM Type Values and Implementations</a>, on page 205 for details of how memory is implemented for different devices.</p>
-----------	---

no_rw_check	<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 M512, or with the power value for supported technologies. You cannot use it with the rw_check option, as the two are mutually exclusive.</p> <p>There are other read-write check controls. See <a href="#">Read-Write Address Checks</a> , on page 206 for details about the differences.</p>
no_rw_check_diff_clk	<p>When enabled, the synthesis tool prevents the insertion of bypass logic around the RAM. If you know your design has RAM that has a read clock and a write clock that are asynchronous, use no_rw_check_diff_clk to prevent the insertion of bypass logic. If this option is enabled, you should not set the asynchronous clock groups in your FDC file. For example, if you set the following, do not use this option:</p> <pre>create_clock {p:clkr} -period {10} create_clock {p:clkw} -period {20} set_clock_groups -derive -asynchronous -name {async_clkgroup} -group { {c:clkw} }</pre> <p>Note: The no_rw_check, rw_check, and no_rw_check_diff_clk options for the syn_ramstyle attribute are mutually exclusive and must not be used together. Whenever synthesis conflicts exist, the software uses the following order of precedence: first the syn_ramstyle attribute, the syn_rw_conflict attribute, and then the Automatic Read/Write check Insertion for RAM option on the Implementation Option panel.</p>
ramType	<p>Specifies a device-specific RAM implementation. Valid values vary from vendor to vendor as they are based on device architecture:</p> <ul style="list-style-type: none"> <li>• Lattice: distributed</li> </ul> <p>See <a href="#">RAM Type Values and Implementations</a> , on page 205 for details of how memory is implemented for different devices.</p>
registers	<p>Specifies that an inferred RAM be mapped to registers (flip-flops and logic), not technology-specific RAM resources.</p>

**rw\_check**

When enabled, the synthesis tool inserts bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations.

You can use this option on its own or in conjunction with a RAM type value such as M512, or with the power value for supported technologies. You cannot use it with the no\_rw\_check option, as the two are mutually exclusive.

Do not enable this option for RAMs with asynchronous read/write clocks. If rw\_check is enabled on block RAM with an asynchronous read clock (rclk) and write clock (wclk), the tool inserts extra logic and a timing path between wclk and rclk. If the clocks are asynchronous to each other, this path can produce glitches on hardware.

There are other read-write check controls. See [Read-Write Address Checks](#), on page 206 for details about the differences.

## RAM Type Values and Implementations

The table lists vendor-specify RAM implementation information, including vendor-specific *ramType* values.

Vendor	Values	Implementation	Technology
Lattice		Default: Synchronous dual-port memory	LatticeECP3/ ECP2/ECP2M/ ECP2S
	registers	Registers	LatticeECP/EC
	block_ram	Device-specific RAMs	LatticeSC/SCM
	block_ram, no_rw_check/ rw_check	RAMs without/with glue logic	LatticeXP2/XP MachXO
	distributed	Distributed RAM or PFU resources	
		Default: block_ram	iCE40
	registers	Registers	iCE40
	block_ram	Device-specific RAMs	
	block_ram, no_rw_check/ rw_check	RAMs without/with glue logic	

## 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.

The `syn_ramstyle` values vary with the technology.

## 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:

Read Write Control	Use when ...
<code>syn_ramstyle</code>	<p>You know your 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:</p> <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>
Read Write Check on RAM	<p>You want to globally enable or disable glue logic insertion for all the RAMs in the design.</p>

If there is a conflict, the software uses the following order of precedence:

- `syn_ramstyle` attribute settings
- Read Write Check on RAM option on the Device panel of the Implementation Options dialog box.

## syn\_ramstyle Syntax

FDC	<code>define_attribute {<i>signalname</i> [<i>bitRange</i>]} -syn_ramstyle <i>value</i></code> <code>define_global_attribute syn_ramstyle <i>value</i></code>	<a href="#">FDC Example</a>
Verilog	<code><i>object</i> /* synthesis syn_ramstyle = <i>value</i> */</code>	<a href="#">Verilog Example</a>
VHDL	<code>attribute syn_ramstyle of <i>object</i> : <i>objectType</i> is <i>value</i> ;</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_ramstyle	block_ram	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="block_ram" */;
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
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 "block_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;
```

## syn\_reduce\_controlset\_size

### Attribute

Specifies the minimum size of the unique control-set, to customize resource usage.

Vendor	Devices
Lattice	iCE40

### syn\_reduce\_controlset\_size Values

Vendor	Global	Value	Object
Lattice	Yes	Integer up to the maximum number of control sets. Default: 8	Top-level module or architecture

### 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.

The implementation of this attribute varies slightly with the vendor:

Lattice	The tool sets the default number of control-sets to 8. You cannot apply this attribute to asynchronous set/reset registers.
---------	--

## syn\_reduce\_controlset\_size Syntax

FDC	define_attribute {v:object} syn_reduce_controlset_size value define_global_attribute syn_reduce_controlset_size value	<a href="#">FDC Example</a>
Verilog	object /* synthesis syn_reduce_controlset_size = value */	<a href="#">Verilog Example</a>
VHDL	attribute syn_reduce_controlset_size of object : objectType is value;	<a href="#">VHDL Example</a>

## FDC Example

	Enabled	Object Type	Object	Attribute	Value	Val Type
1	<input checked="" type="checkbox"/>	global	<global>	syn_reduce_controlset_size	3	

```
define_global_attribute syn_reduce_controlset_size 2
```

## 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 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` in Lattice Designs

The following examples show how the `syn_reduce_controlset_size` attribute affects control-set implementation in Lattice designs.

### Example 1: `syn_reduce_controlset_size=2`, Verilog

Suppose you have four registers: SB\_DFFSR. with two registers driven by R1 reset and two registers driven by R2 reset. If `syn_reduce_controlset_size` is set to 2, then no control-set optimizations will occur since all control-sets have two registers. However, when `syn_reduce_controlset_size` is set to 3 or greater, then all the registers are converted to SB\_DFFs.

```
module test(i1, i2, i3, i4, r1, r2, clk, o1, o2, o3, o4)
    /* synthesis syn_useioff = 0 syn_reduce_controlset_size = 2 */;
    input i1, i2, i3, i4;
    input clk;
    input r1, r2;
    output o1, o2, o3, o4;

    reg reg1, reg2, reg3, reg4;

    always@(posedge clk)
        if (r1)
            reg1 <= 1'b0;
        else
            reg1 <= i1;

    always@(posedge clk)
        if (r1)
            reg2 <= 1'b0;
        else
            reg2 <= i2;

    always@(posedge clk)
        if (r2)
            reg3 <= 1'b0;
        else
            reg3 <= i3;

    always@(posedge clk)
        if (r2)
            reg4 <= 1'b0;
        else
            reg4 <= i4;
```

```

assign o1 = reg1;
assign o2 = reg2;
assign o3 = reg3;
assign o4 = reg4;
endmodule

```

## Example 2: syn\_reduce\_controlset\_size=3, Verilog

Suppose you have five registers with two registers sharing reset signal r1 and three registers sharing reset signal r2. If `syn_reduce_controlset_size` is set to 2, then no control-set optimizations will occur on either the r1 control-set or the r2 control-set, since each of these control-sets has the specified minimum of two registers. However, when `syn_reduce_controlset_size` is set to 3 the two registers sharing r1 are converted to SB\_DFFs and r1 is implemented in combinational logic. The three registers sharing r2 are unaffected since the r2 control-set has the specified minimum of three registers.

```

module test(i1, i2, i3, i4, i5, r1, r2, clk, o1, o2, o3, o4, o5)
  /* synthesis syn_useioff = 0 syn_reduce_controlset_size = 3
  */;input i1, i2, i3, i4, i5;
  input clk;
  input r1, r2;
  output o1, o2, o3, o4, o5;

  reg reg1, reg2, reg3, reg4, reg5;

  always@(posedge clk)
  if (r1)
    reg1 <= 1'b0;
  else
    reg1 <= i1;

  always@(posedge clk)
  if (r2)
    reg2 <= 1'b0;
  else
    reg2 <= i2;

  always@(posedge clk)
  if (r2)
    reg3 <= 1'b0;
  else
    reg3 <= i3;

  always@(posedge clk)
  if (r1)

```

:

---

```
        reg4 <= 1'b0;
    else
        reg4 <= i4;

    always@(posedge clk)
    if (r2)
        reg5 <= 1'b0;
    else
        reg5 <= i5;

    assign o1 = reg1;
    assign o2 = reg2;
    assign o3 = reg3;
    assign o4 = reg4;
    assign o5 = reg5;

endmodule
```

### Example 3: syn\_reduce\_controlset\_size=7, VHDL

```
Library ieee;
use ieee.std_logic_1164.all;

entity reg_top is
port (clk : in std_logic;
      reset : in std_logic_vector (1 downto 0);
      ina,inb : in std_logic;
      outa,outb: out std_logic);
end;

architecture rtl of reg_top is
signal a_reg,b_reg: std_logic_vector (3 downto 0);
attribute syn_useioff : boolean;
attribute syn_useioff of rtl : architecture is false;
attribute syn_reduce_controlset_size : integer;
attribute syn_reduce_controlset_size of rtl : architecture is 7;

begin
process (clk,reset)
begin
    if (clk'event and clk='1') then
    if (reset(0)= '1') then
        a_reg(0)<='0';
        b_reg(0)<='0';
    else
        a_reg(0)<=ina;
```



```
        b_reg(0)<=inb;
    end if;
end if;
end process;

process (clk,reset)
begin
if (clk'event and clk='1') then
    if (reset(1)= '1') then
        a_reg(1)<='0';
        b_reg(1)<='0';
    else
        a_reg(1)<=a_reg(0);
        b_reg(1)<=b_reg(0);
    end if;
end if;
end process;

process (clk,reset)
begin
if (clk'event and clk='1') then
    if (reset(0)= '0') then
        a_reg(2)<='1';
        b_reg(2)<='1';
    else
        a_reg(2)<=a_reg(1);
        b_reg(2)<=b_reg(1);
    end if;
end if;
end process;

process (clk,reset)
begin
if (clk'event and clk='1') then
    if (reset(0)= '0') then
        a_reg(3)<='1';
        b_reg(3)<='1';
    else
        a_reg(3)<=a_reg(2);
        b_reg(3)<=b_reg(2);

    end if;
end if;
```

:

---

```
end process;  
  
outa<=a_reg(3);  
outb<=b_reg(3);  
end;
```

## syn\_reference\_clock

### Attribute

Specifies a clock frequency other than the one implied by the signal on the clock pin of the register.

Vendor	Technology	Default Value	Global	Object
Lattice	iCE40, ECP3, older families	-	-	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.

FDC	<b>define_attribute {register} syn_reference_clock {clockName}</b>	<a href="#">FDC Example</a>
-----	--	-----------------------------

### FDC Example

```
define_attribute {register} syn_reference_clock {clockName}
```

For example:

```
define_attribute {myreg[31:0]} syn_reference_clock {sloClock}
```

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.

The following example shows how you can apply the constraint to all registers with the enable signal `en40`:

:

```
define_scope_collection clock_enable_col {find -seq * -filter
  (@clock_enable==en40)}
define_attribute {$clock_enable_col} syn_reference_clock {clk2}
```

Enable	Object Type	Object	Attribute	Value	Value Type	Description
<input checked="" type="checkbox"/>	<any>	<Global>	syn_reference_clock	1	string	Override the default ...

---

**Note:** You apply `syn_reference_clock` only in a constraint file; you cannot use it in source code.

---

## Effect of using `syn_reference_clock`

The following figure shows the report before applying the attribute:

Performance Summary							
*****							
Worst slack in design: 499.379							
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							
*****							
Worst slack in design: 999.379							
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\_replicate

### *Attribute*

Controls replication of registers during optimization.

Vendor	Technologies
Lattice	iCE40 and older families

### syn\_replicate values

Value	Default	Global	Object	Description
0	No	Yes	Register	Disables duplication of registers
1	Yes	Yes	Register	Allows duplication of registers

### 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.

## syn\_replicate Syntax Specification

FDC	define_global_attribute syn_replicate {0   1};	<a href="#">FDC Example</a>
Verilog	<i>object</i> /* synthesis syn_replicate = 1   0 */;	<a href="#">Verilog Example</a>
VHDL	attribute syn_replicate : boolean; attribute syn_replicate of <i>object</i> : signal is true false;	<a href="#">VHDL Example</a>

## FDC 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;

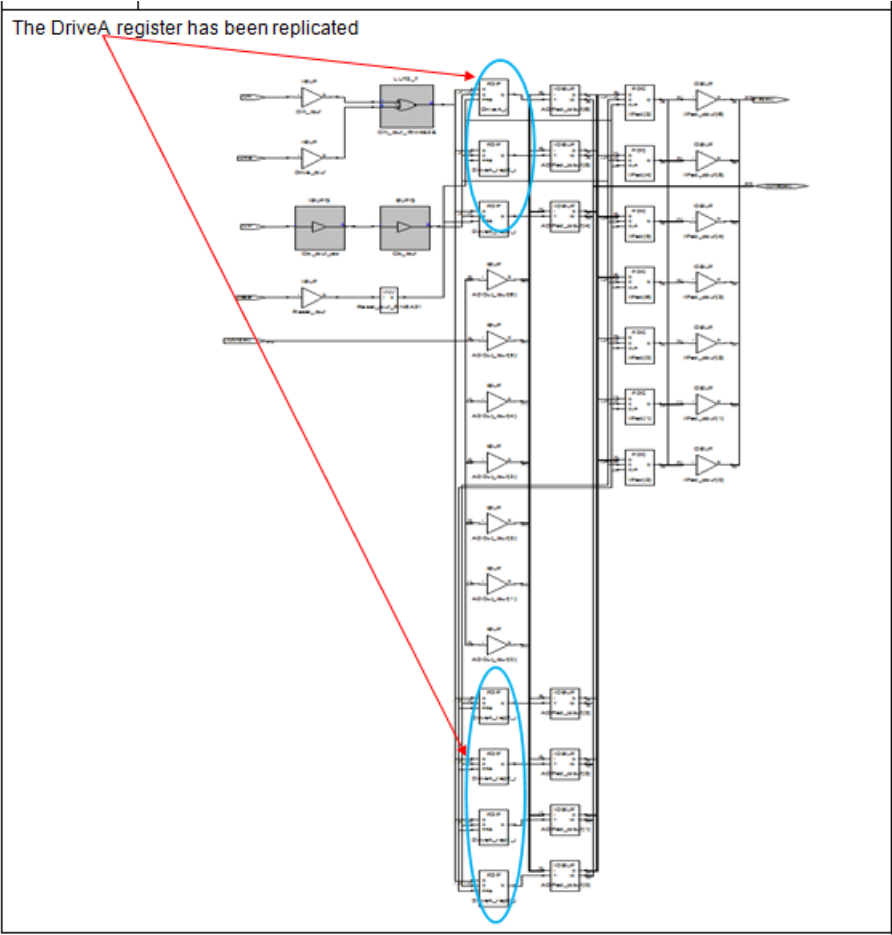
architecture archnorep of norep is
  signal DriveA : std_logic;
  attribute syn_replicate : boolean;
  attribute syn_replicate of DriveA : signal is false;
begin
  ADPad <= ADOut when DriveA='1' else (others => 'Z');
  process (Clk, Reset)
  begin
    if Reset='0' then
      DriveA <= '0';
      IPad <= (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	<code>reg DriveA /*synthesis syn_replicate=1*/</code>
VHDL	<code>attribute syn_replicate : boolean; attribute syn_replicate of DriveA : signal is true;</code>

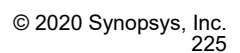


When you apply `syn_replicate`, the registers are not duplicated:

Verilog      `reg DriveA /*synthesis syn_replicate=0*/`

VHDL      `attribute syn_replicate : boolean;`  
             `attribute syn_replicate of DriveA : signal is false;`





## syn\_romstyle

### *Attribute*

This attribute determines how ROM architectures are implemented.

Vendor	Device	Values
Lattice	ECP3, iCE40 and older families	logic   block_rom   distributed
	iCE40UP	auto   logic   EBR

### syn\_romstyle Values

Value	Description
logic	Uses discrete logic primitives.
block_rom	Specifies that the inferred ROM be mapped to the appropriate vendor-specific memory.
distributed	Implements the ROM structure as distributed ROM.
EBR	Specifies that the inferred ROM be mapped to the appropriate vendor-specific memory.
auto	Default mode.

### 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.

You can infer ROM architectures using a `case` statement in your code. For the synthesis tool to implement a ROM, at least half of the available addresses in the `case` statement must be assigned a value. For example, consider a ROM with six address bits (64 unique addresses). The `case` statement for this ROM must specify values for at least 32 of the available addresses.

### syn\_romstyle Values Syntax

The following support applies for the `syn_romstyle` attribute.

Default	Global Support	Object
logic	Yes	v: module or entity

This table summarizes the syntax in different files:

FDC	<code>define_attribute {romPrimitive} syn_romstyle {logic   block_rom   distributed}</code>	<a href="#">SCOPE Example</a>
Verilog	<code>object /* synthesis syn_romstyle = "logic   block_rom   distributed" */;</code>	<a href="#">Verilog Example</a>
VHDL	<code>attribute syn_romstyle of object: objectType is "logic   block_rom   distributed";</code>	<a href="#">VHDL Example</a>

## SCOPE 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'b00000101;
            5'b00010: dataout <= 8'b00001001;
            5'b00011: dataout <= 8'b00001101;
            5'b00100: dataout <= 8'b00010001;
        endcase
    end
endmodule

```

:

---

```
5'b00101: dataout <= 8'b00011001;
5'b00110: dataout <= 8'b00100001;
5'b00111: dataout <= 8'b10110100;
5'b01000: dataout <= 8'b11000000;
5'b01000: dataout <= 8'b00011011;
5'b01001: dataout <= 8'b10110001;
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'b10110000;
5'b10000: dataout <= 8'b11111011;
5'b10001: dataout <= 8'b00010001;
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
```

## 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 is
    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;

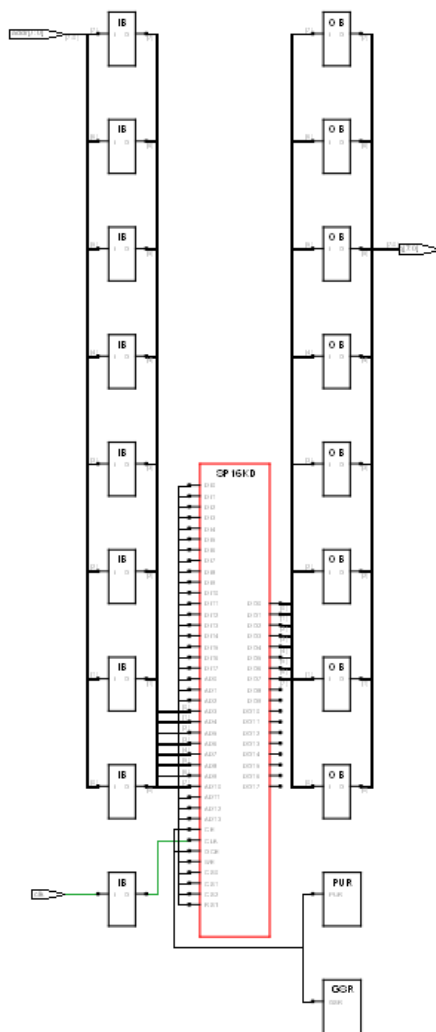
  begin
    process(clk)
    begin
      if(rising_edge(clk)) then
        q <= rom(addr);
      end if;
    end process;
  end rtl;

```

## Effect of Using `syn_romstyle` for Lattice

```
Verilog  module test (clock,addr,dataout) /*synthesis syn_romstyle =
        "block_rom" */;
```

```
VHDL    attribute syn_romstyle: string;
        attribute syn_romstyle of q : signal is "block_rom";
```



## syn\_safe\_case

### Directive

This directive enables/disables the safe case option.

Vendor	Technologies
Lattice	ECP3, MachXO2

### syn\_safe\_case Values

Value	Description	Default	Global
false   0	Turns off the safe case option.	false   0	No
true   1	Turns on the safe case option.		

### Description

This directive enables/disables the safe case option. When enabled, the high reliability safe case option turns off sequential optimizations for counters, FSM, and sequential logic to increase the reliability of the circuit. If you set this directive on a module or architecture, the module or architecture is treated as safe and all case statements within it are implemented as safe.

The `syn_safe_case` directive can perform operations on FSMs and pmuxes to preserve default states and inject fault recovery logic to the default case. Using this directive might produce different results than the Preserve and Decode Unreachable States option.

### syn\_safe\_case Syntax

Verilog `module /* syn_safe_case = "1 | 0" */;`

[Verilog Example](#)

VHDL `attribute syn_safe_case : boolean;  
attribute syn_safe_case of architectureName: architecture  
is "true | false";`

[VHDL Example](#)

## Verilog Example

For example:

```
module top (input a, output b) /* synthesis syn_safe_case =1 */
```

## VHDL Example

For example:

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

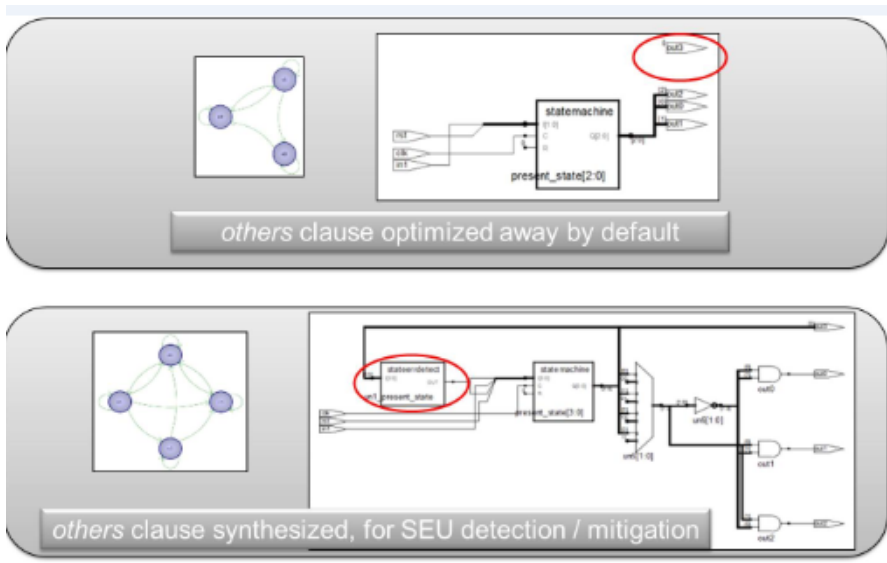
entity test is
port (a input std_logic;
      b: out std_logic);
end test;

architecture rtl of test is
attribute syn_safe_case: boolean;
attribute syn_safe_case of rtl : architecture is "TRUE";
```

## Effect of Using syn\_safe\_case

This example shows the others clause optimized away; then synthesized for SEU detection and mitigation when the syn\_safe\_case directive is enabled.





# syn\_safe fsm\_pipe

## Attribute

Removes the pipeline register on the error recovery path for the Preserve and Decode Unreachable States option.

## Vendor Technologies

Lattice	ECP3 and MachXO2 families
---------	---------------------------

## syn\_safe fsm\_pipe Values

Value	Description	Default	Global
true   1	Preserves the pipeline register on the error recovery path.	true   1	Yes
false   0	Removes the pipeline register on the error recovery path.		

## Description

The `syn_safe fsm_pipe` directive removes the pipeline register on the error recovery path for the Preserve and Decode Unreachable States option. This allows for better simulation matching to minimize clock synchronization issues and post place and route timing violations. The attribute can either be specified on an FSM instance or globally.

## syn\_safe fsm\_pipe Syntax

SCOPE	define_attribute <i>moduleName</i> syn_safe fsm_pipe {0} define_global_attribute syn_safe fsm_pipe {0}	<a href="#">FDC Example</a>
Verilog	module /* syn_safe fsm_pipe = "0" */;	<a href="#">Verilog Example</a>
VHDL	attribute syn_safe fsm_pipe : boolean; attribute syn_safe fsm_pipe of <i>architectureName</i> : architecture is "false";	<a href="#">VHDL Example</a>

## FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	global	<Global>	syn_safe fsm_pipe	0	integer	Remove pipeline register on recovery path for safe case FSMs

```
define_global_attribute syn_safe fsm_pipe {0}
```

## Verilog Example

The syn\_safe fsm\_pipe attribute is used in the following Verilog code snippet:

```
module fsm (clk, reset, x1, outp);
input clk, reset, x1;
output outp;
reg outp;
reg [1:0] state /* synthesis syn_safe fsm_pipe = 0 */;
parameter s1 = 2'b00; parameter s2 = 2'b01;
parameter s3 = 2'b10; parameter s4 = 2'b11;
```

## VHDL Example

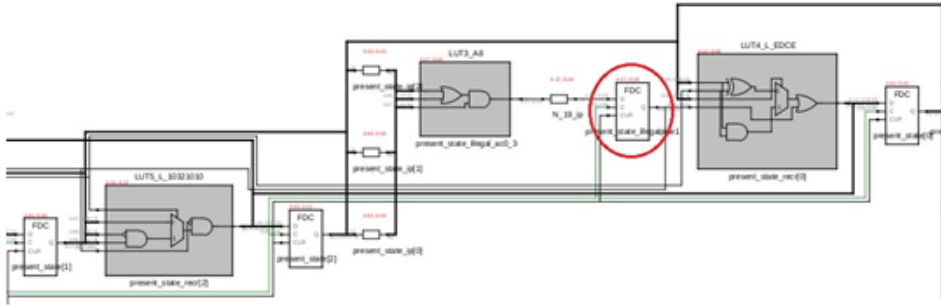
The syn\_safe fsm\_pipe attribute is used in the following VHDL code snippet:

```
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;

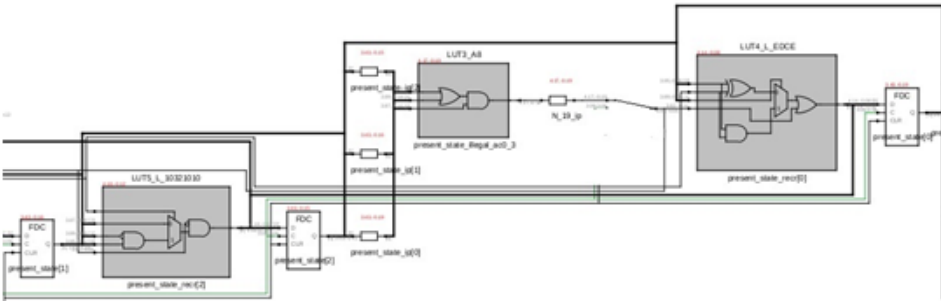
architecture rtl of fsm is
signal state : std_logic_vector(1 downto 0);
constant s1 : std_logic_vector := "00";
constant s2 : std_logic_vector := "01";
constant s3 : std_logic_vector := "10";
constant s4 : std_logic_vector := "11";
attribute syn_safe fsm_pipe : string;
attribute syn_safe fsm_pipe of state : signal is "false";
```

## Effects of Using syn\_saferfm\_pipe

By default, when Preserve and Decode Unreachable States is enabled, a pipeline register is inserted in the recovery path.



When the syn\_saferfm\_pipe attribute is set to 0, the pipeline register on the error recovery path is removed.



## syn\_sharing

### Directive

Enables or disables the sharing of operator resources during the compilation stage of synthesis.

Technology	Default Value	Global	Object
All	On	Yes	Component, module

### syn\_sharing Values

Value	Description
off	Does not share resources during the compilation stage of synthesis.
on (Default)	Optimizes the design to perform resource sharing during the compilation stage of synthesis.

### Description

The `syn_sharing` directive controls resource sharing during the compilation stage of synthesis. 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 Resource Sharing option in the Project view, from the Project->Implementation Options->Options panel, or with the `set_option -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. See [Sharing Resources, on page 454](#) in the *User Guide* for a detailed procedure.

### syn\_sharing Syntax

Verilog	<code>object /* synthesis syn_sharing="on   off" */;</code>	<a href="#">Verilog Example</a>
VHDL	attribute <code>syn_sharing</code> of <code>object</code> : <code>objectType</code> is "true   false";	<a href="#">VHDL Example</a>

## 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
    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";

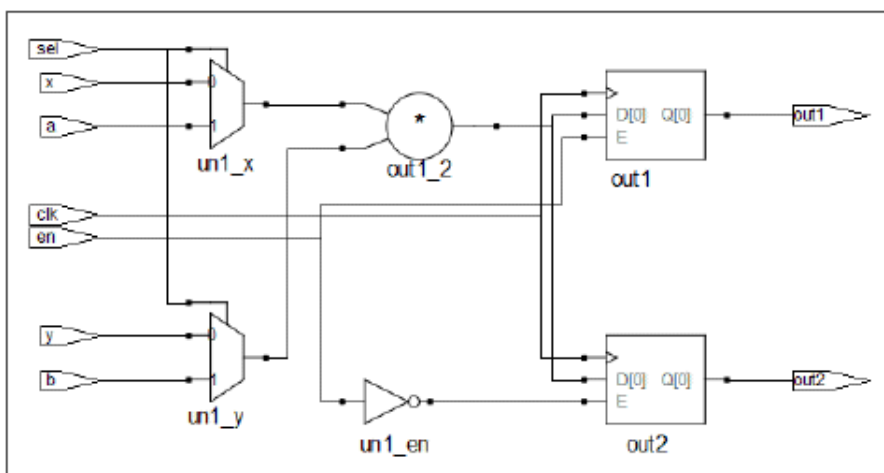
  signal tmp1, tmp2: std_logic_vector(3 downto 0);
begin
  tmp1 <= a * b;
  tmp2 <= x * y;

  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 : architecture is "on";
```

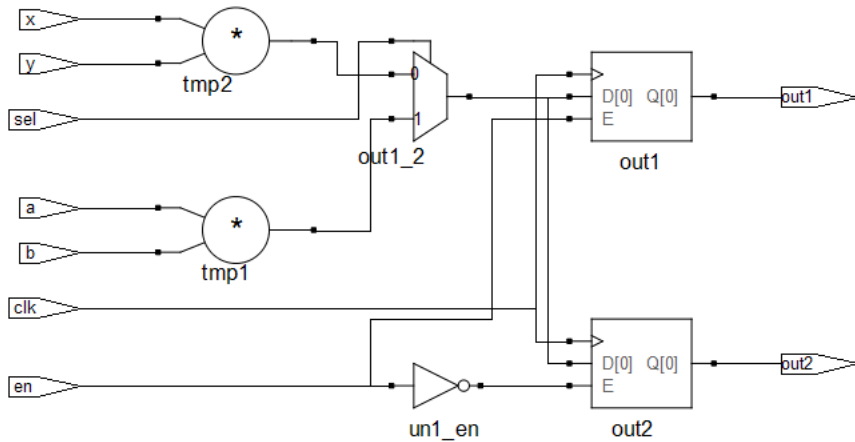




The next figure shows the same design when resource sharing is off, and two adders are inferred:

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.

Vendor	Technology
Lattice	ECP3, MachXO2

### syn\_shift\_resetphase Values

Value	Description
1 (Default)	The flip-flop on the inactive clock edge is present.
0	Removes the flip-flop on the inactive clock edge.

### 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, isolating the reset. You can use the `syn_shift_resetphase` attribute 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 69.

### syn\_shift\_resetphase Syntax

Global Support	Object
Yes	FSM instance

The following table summarizes the syntax in different files:

FDC	define_attribute <i>object</i> {syn_shift_resetphase} {1 0} define_global_attribute {syn_shift_resetphase} {1 0}	<a href="#">SCOPE Example</a>
Verilog	<i>object</i> /* synthesis syn_shift_resetphase = "1   0" */;	<a href="#">Verilog Example</a>
VHDL	attribute syn_shift_resetphase of <i>state</i> : signal is "true   false";	<a href="#">VHDL Example</a>

## SCOPE 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

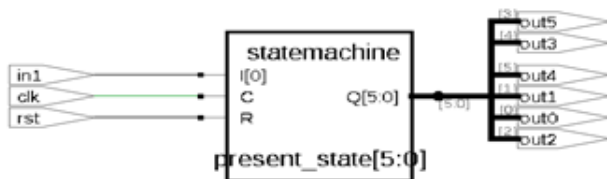
    ...

  end rtl;

```

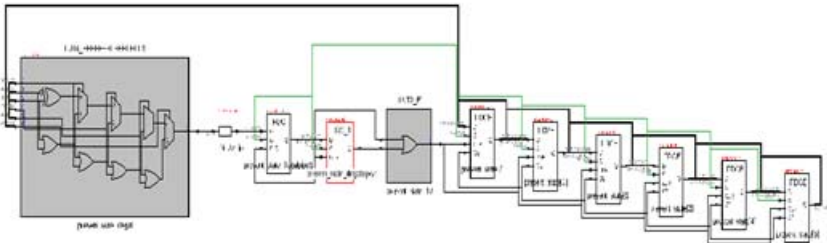
## Effect of Using syn\_shift\_resetphase

Safe encoding is implemented for the following state machine.



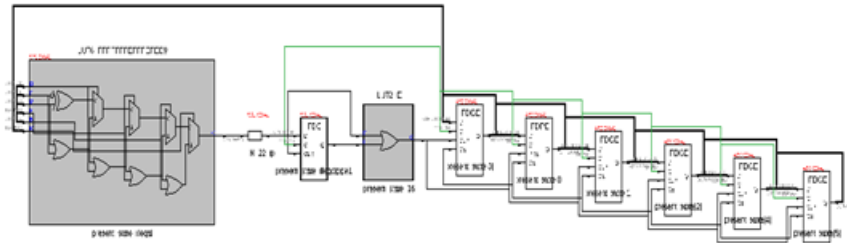
This example shows Technology view results before the `syn_shift_resetphase` attribute is applied.

Enable	Object Type	Object	Attribute	Value
✓	<any>	ipresent_state[5:0]	syn_encoding	safe,onehot



This example shows Technology view results after the syn\_shift\_resetphase attribute is applied.

Enable	Object Type	Object	Attribute	Value
✓	<any>	ipresent_state[5:0]	syn_encoding	safe,onehot
✓	<any>	<Global>	syn_shift_resetphase	0



## syn\_smhigheffort

### Attribute

Uses higher threshold effort when the tool extracts a state-machine on individual state registers.

Technology	Default Value	Global	Object
All	Default is 0   false	Yes	Component, module

### syn\_smhigheffort Values

Value	Description
0   false	Does not increase effort to extract the state machines.
1   true	Allows increase in effort to extract the state machines.

### Description

Increases effort to extract a state-machine on individual state registers by using a higher threshold. Use this attribute when state machine extraction is enabled, but they are not automatically extracted. To increase effort to extract some state machines, use this attribute with a value of 1 with higher threshold. The Compiler devotes more effort to attempt state machine extraction but this also increases runtime. By default, `syn_smhigheffort` is set with a value of 0. This attribute can be used when a state machine extraction is enabled but it is not automatically extracted.

### syn\_smhigheffort Syntax

Verilog	<code>object /* synthesis syn_smhigheffort = "0   1" */;</code>
VHDL	<code>attribute syn_smhigheffort of &lt;object_name&gt;: signal is "false   true";</code>

For Verilog:

- `object` is a state register.
- Data type is Boolean: 0 does not extract an FSM, 1 extracts an FSM.

```
reg [7:0] current_state /* synthesis syn_smhigheffort=1 */;
```

For VHDL:

- *state* is a signal that holds the value of the state machine.
- Data type is Boolean: *false* does not extract an FSM, *true* extracts an FSM.

```
attribute syn_smhigheffort of current_state: signal is true;

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_smhigheffort = 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
end
```

```

end
endcase
endmodule

```

This is the Verilog 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_smhigh effort : boolean;
    attribute syn_smhigh effort 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 =>

```



```

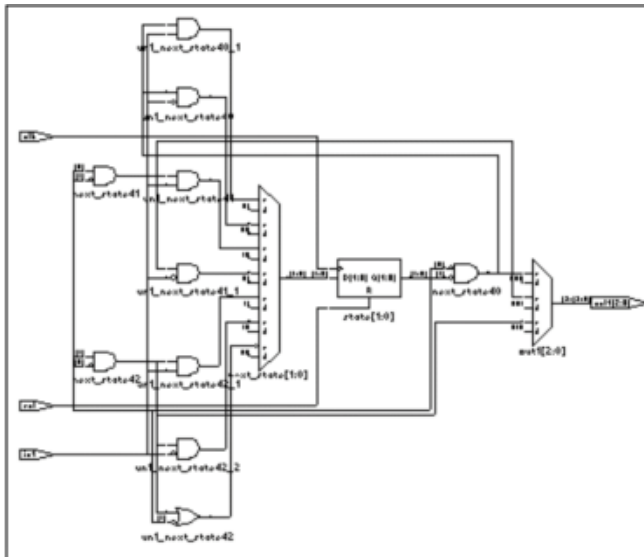
        out1 <= "XXX"; next_state <= s0;
    end case;
end process;
end behave;

```

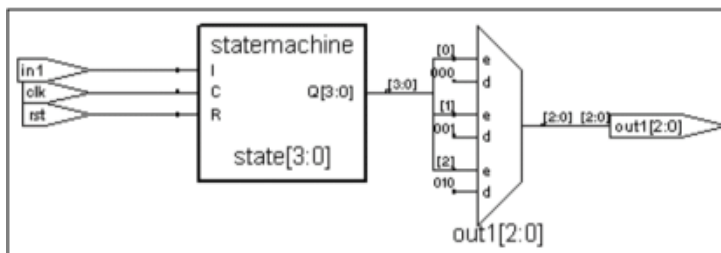
This is the VHDL source code used for the example in the following figure.

## Effect of Using syn\_smhigheffort

The following figure shows an example of two implementations of a state machine: one with the syn\_smhigheffort attribute enabled, the other with the attribute disabled.



syn\_smhigheffort = 0



syn\_smhigheffort = 1

See [syn\\_state\\_machine](#), on page 254 for information on enabling/disabling state-machine optimization on individual state registers.

## syn\_srstyle

### *Attribute*

Determines how to implement the sequential shift components.

Vendor	Technology
Lattice	LatticeEC/ECP/ECP2 families LatticeSC/SCM families LatticeXP/XP2 families MachXO family

### syn\_srstyle Values

Technology	Value	Implements ...
Lattice		
All supported families	registers	Maps seqShift register components to registers.
	distributed	Maps shift registers to distributed RAM.
	block_ram	Maps shift registers to block RAM.

### Description

The tool infers sequential shift components based on threshold limits. The `syn_srstyle` attribute can be used to override the default behavior of `seqshift` implementation depending on how you set the values.

The `syn_srstyle` 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\_srstyle Syntax

SCOPE	<pre>define_attribute {object} syn_srstyle {register   block_ram   distributed} define_global_attribute syn_srstyle {register   block_ram   distributed}}</pre>	<a href="#">SCOPE Example</a>
Verilog	<pre>object /* synthesis syn_srstyle = "register   block_ram   distributed" */;</pre>	
VHDL	<pre>attribute syn_srstyle: string; attribute syn_srstyle of object : signal is "register   block_ram   distributed";</pre>	

## SCOPE Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	register	i:special_regs.w[7:0]	syn_srstyle	registers	string	Determines how seq. shift comp. are implemented
2							

For example:

```
define_attribute {i:regBank[15:0]} syn_srstyle {registers}
```

## HDL Example

In the HDL file, you must apply the `syn_srstyle` attribute on the final stage of the shift register. In the following example, apply the `syn_srstyle` attribute on register `pll_status_ck245_s`. The constraint is not honored if it is placed on other registers in the shifting 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_srstyle : string;
    attribute syn_srstyle 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;
```

# syn\_state\_machine

## Directive

Enables/disables state-machine optimization on individual state registers in the design.

Technology	Default Value	Global	Object
All	Default is determined by the global FSM Compiler option. set_option -symbolic_fsm_compiler 1	Yes	Component, module

## syn\_state\_machine Values

Value	Description
off   false	Does not extract state machines automatically.
on   true	Automatically extracts state machines.

## 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" */;`

[Example – Verilog syn\\_state\\_machine](#)

VHDL `attribute syn_state_machine of state : signal is "false | true";`

[Example – VHDL syn\\_state\\_machine](#)

For Verilog:

- *object* is a state register.
- Data type is Boolean: 0 does not extract an FSM, 1 extracts an FSM.

```
reg [7:0] current_state /* synthesis syn_state_machine=1 */;
```

For VHDL:

- *state* is a signal that holds the value of the state machine.
- Data type is Boolean: false does not extract an FSM, true extracts an FSM.

```
attribute syn_state_machine of current_state: signal is true;
```

## Example – Verilog syn\_state\_machine

```
// Example: Verilog syn_state_machine

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
```

This is the Verilog source code used for the example in the following figure.

### Example – VHDL syn\_state\_machine

```
-- Example: VHDL syn_state_machine

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;

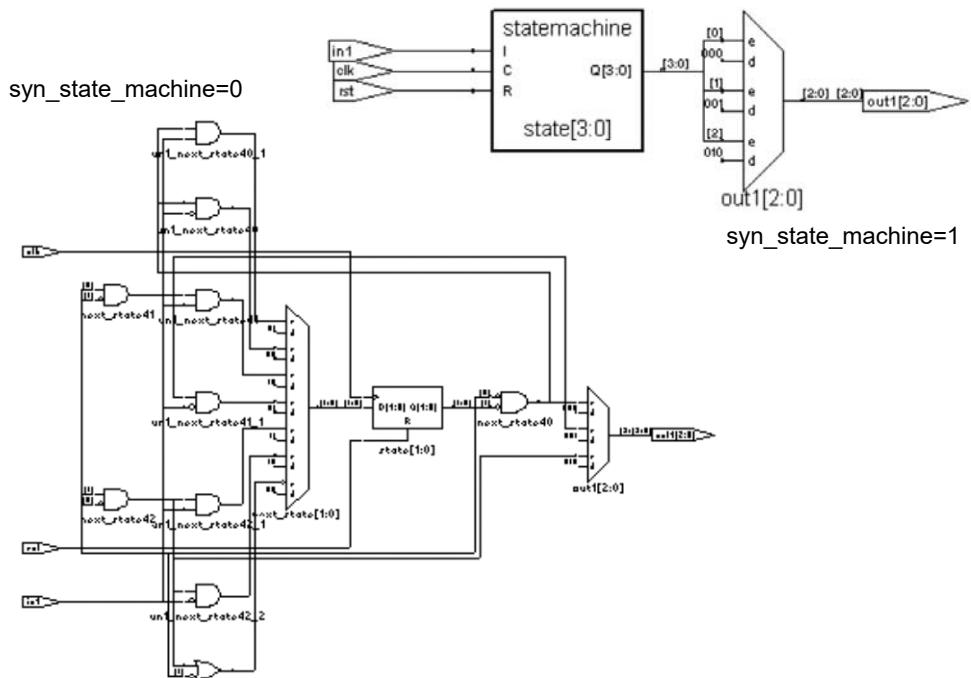
    process (state, in1) begin
```

```
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;
```

This is the VHDL source code used for the example in the following figure.

## 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 the following HDL syntax and example sections for the source code used to generate the schematics above. See also:

- [syn\\_encoding](#), on page 69 for information on overriding default encoding styles for state machines.
- For VHDL designs, [syn\\_encoding Compared to syn\\_enum\\_encoding](#), on page 81 for usage information about these two directives.

## 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 timing-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 44](#) for a list of the associated directives.

### syn\_tco<n> Syntax

Verilog     *object* !\* **syn\_tcon** = "[!]clock -> bundle = value" \*!;

VHDL        **attribute syn\_tcon of** *object* : *objectType* **is** "[!]clock -> bundle = value";

The syn\_tco<n> directive can be entered as an attribute using the Attributes panel of the SCOPE editor. The information in the Object, Attribute, and Value fields must be manually entered. This is the constraint file syntax for the directive:

```
define_attribute {v:blackboxModule} syn_tcon {![]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.
<i>blackboxModule</i>	The symbol name of the black-box.
<i>n</i>	A numerical suffix that lets you specify different clock to output timing delays for multiple signals/bundles.
<b>!</b>	The optional exclamation mark indicates that the clock is active on its falling (negative) edge.
<i>clock</i>	The name of the clock signal.

**bundle** A bundle is 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**

The values are in ns.

---

**value** Clock to output delay value in ns.

---

Constraint file example:

```
define_attribute {v:work.test} {syn_tsu4} {clk->tout=1.0}
```

## Verilog Example

**object** /\* **syn\_tcon** = "[!]clock -> bundle = value" \*/;

See [syn\\_tco<n> Syntax, on page 260](#) for syntax explanations. 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_tpd1="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 synply 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;
```

See [syn\\_tco<n> Syntax, on page 260](#) for other syntax explanations.

See [VHDL Attribute and Directive Syntax, on page 414](#) for alternate methods for specifying VHDL attributes and directives.

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_tsu1 : string;  
attribute syn_tsu1 of test : component is  
    "a-> myclk = 1.2";  
begin  
U1 : test port map(fout, a, b, clk);  
end;
```

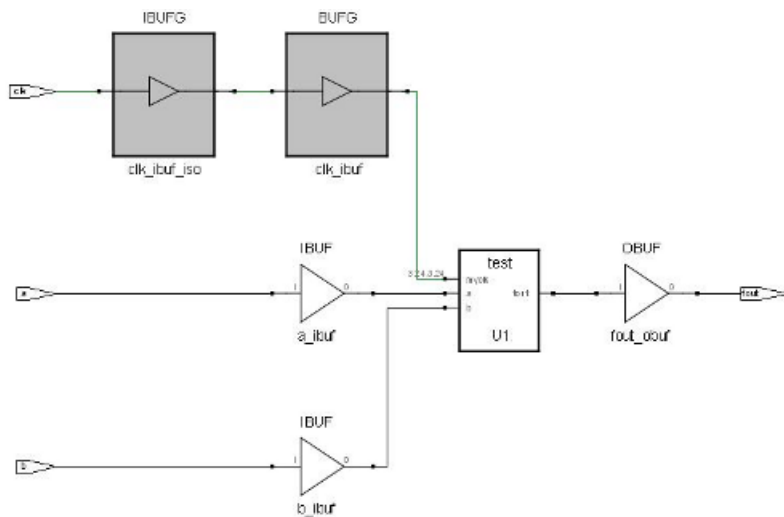
## Verilog-Style Syntax in VHDL for Black Box Timing

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_tcol of inputfifo_coregen : component is  
    "rd_clk->dout[48:0]=3.0";
```

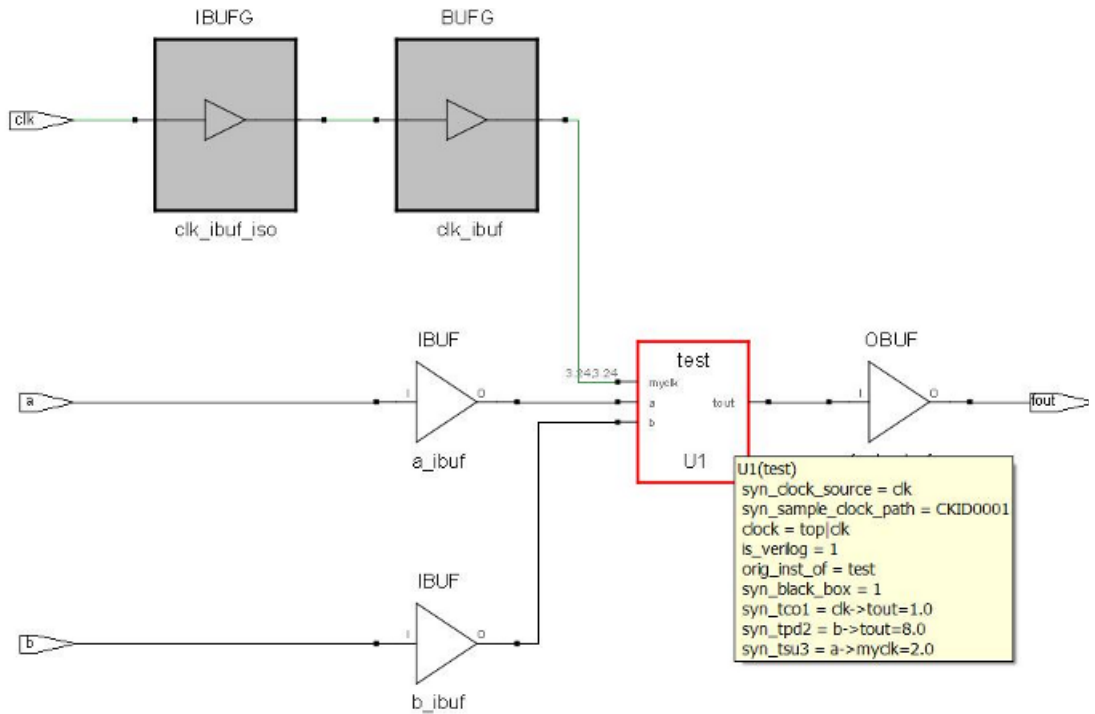
## Effect of using syn\_tco

This figure shows the HDL Analyst Technology view before using syn\_tco:





This figure shows the HDL Analyst Technology 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 44 for a list of the associated directives.

### syn\_tpd<n> Syntax

Verilog     *object !\* **syn\_tpd**n = "bundle -> bundle = value" \*!;*

VHDL        **attribute syn\_tpd**n of *object* : *objectType* is "bundle -> bundle = value";

You can enter the syn\_tpd<n> directive as an attribute using the Attributes panel of the SCOPE editor. The information in the Object, Attribute, and Value fields must be manually entered. This is the constraint file syntax:

**define\_attribute {v:blackboxModule} syn\_tpd**n {bundle->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.
<i>blackboxModule</i>	The symbol name of the black-box.

<i>n</i>	A numerical suffix that lets you specify different input to output timing delays for multiple signals/bundles.
<i>bundle</i>	<p>A bundle is 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.</p> <p style="text-align: center;"><b>"<i>bundle</i>-&gt;<i>bundle</i>=<i>value</i>"</b></p> <p>The values are in ns.</p>
<i>value</i>	Input to output delay value in ns.

Constraint file example:

```
define_attribute {v:MEM} syn_tpd1 {MEM_RD->DATA_OUT[63:0]=20}
```

## Verilog Example

See [syn\\_tpd<n> Syntax, on page 266](#) for an explanation of the syntax. This is an example of syn\_tpd<n> along with some of the other black-box timing constraints:

```
module test(myclk, a, b, tout,)
  /*synthesis syn_black_box syn_tcol="clk->tout=1.0"
    syn_tpd1="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";
```

See [syn\\_tpd<n> Syntax, on page 266](#) for an explanation of the syntax.

See [VHDL Attribute and Directive Syntax, on page 414](#) for different ways to specify VHDL attributes and directives.

The following is an example of assigning syn\_tpd<n> along with some of the black box constraints. See [Verilog-Style Syntax in VHDL for Black Box Timing, on page 263](#) for another way.

```
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;
```

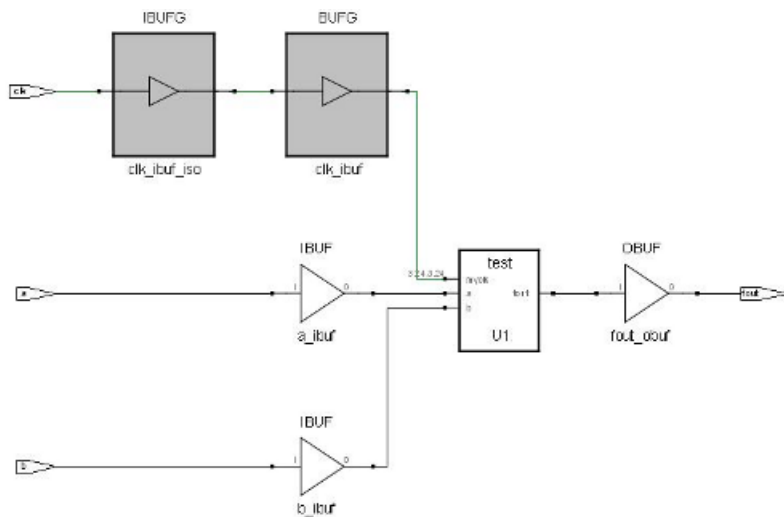
```
-- TOP Level--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity top is
generic (size: integer := 8);
port (fout : out std_logic_vector(size- 1 downto 0);
      a : in std_logic_vector (size- 1 downto 0);
      b : in std_logic_vector (size- 1 downto 0);
      clk : 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_tsu1 : string;
attribute syn_tsu1 of test : component is
  "a-> myclk = 1.2";
begin
U1 : test port map(fout, a, b, clk);
end;
```

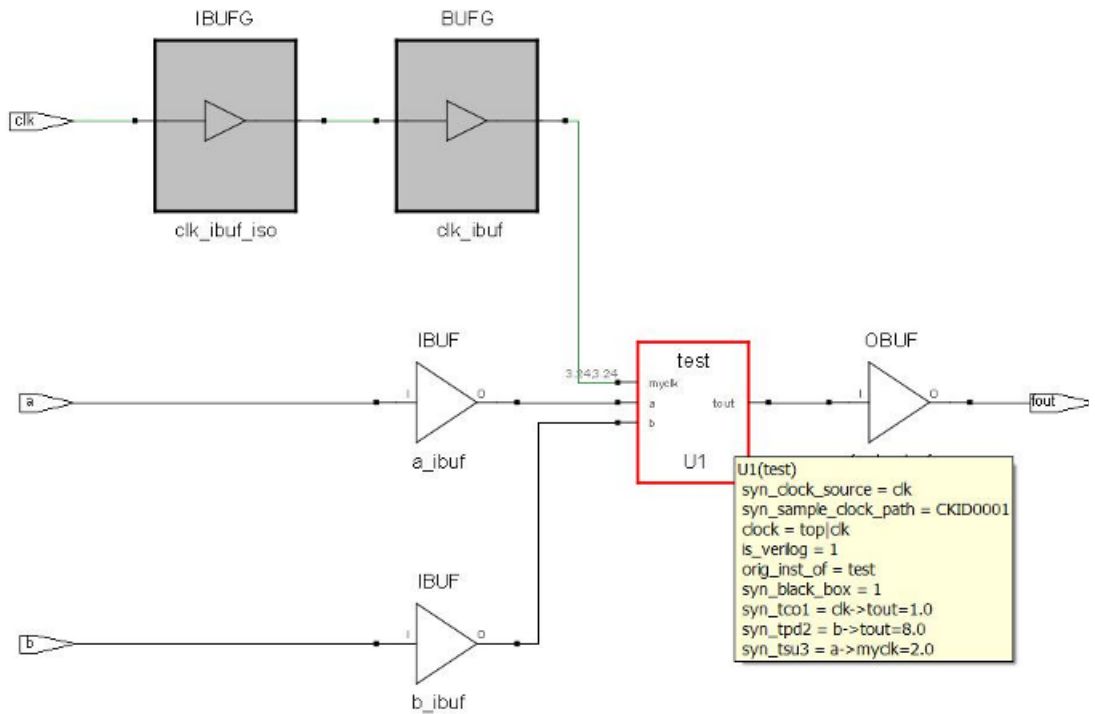
## Effect of using syn\_tpd

This figure shows the HDL Analyst Technology view before using syn\_tpd:



## After using syn\_tpd

This figure shows the HDL Analyst Technology view after using syn\_tpd:



## syn\_tristate

### Directive

Specifies that an output port on a black box is a tristate.

### syn\_tristate Values

Value	Default
0	Yes
1	

### 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	<i>object</i> /* synthesis syn_tristate = 1 */;
VHDL	<b>attribute syn_tristate : boolean;</b> <b>attribute syn_tristate of tout: signal is true;</b>

### 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);
end;
```

## 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 44](#) for a list of the associated directives.

### syn\_tsu<n> Syntax

Verilog     *object* /\* **syn\_tsun** = "bundle -> [!]clock = value" \*/;

VHDL       **attribute syn\_tsun** of *object* : *objectType* is "bundle -> [!]clock = value";

The syn\_tsu<n> directive can be entered as an attribute using the Attributes panel of the SCOPE editor. The information in the Object, Attribute, and Value fields must be manually entered. The constraint file syntax for the directive is:

**define\_attribute {v:blackboxModule} syn\_tsun {bundle->[!]clock=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.
<i>blackboxModule</i>	The symbol name of the black-box.
<i>n</i>	A numerical suffix that lets you specify different clock to output timing delays for multiple signals/bundles.
<i>bundle</i>	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. The values are in ns. This is the syntax to define a bundle:

*bundle->[!]clock=value*

<b>!</b>	The optional exclamation mark indicates that the clock is active on its falling (negative) edge.
<i>clock</i>	The name of the clock signal.
<i>value</i>	Input to clock setup delay value in ns.

Constraint file example:

```
define_attribute {v:RTRV_MOD} syn_tsu4 {RTRV_DATA[63:0]->!CLK=20}
```

## Verilog Example

For syntax explanations, see [syn\\_tsu<n> Syntax](#), on page 275.

This is an example that defines syn\_tsu<n> along 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_tpd1="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,dil -> clk = 2.0";
attribute syn_tsu12 : string;
attribute syn_tsu12 of bitreg : component is
    "di2,di3 -> clk = 1.8";
```

For other syntax explanations, see [syn\\_tsu<n> Syntax, on page 275](#).

See [VHDL Attribute and Directive Syntax, on page 414](#) for different ways to specify VHDL attributes and directives. For this directive, you can also use the following Verilog-style syntax to specify it, as described in [Verilog-Style Syntax in VHDL for Black Box Timing, on page 263](#).

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;

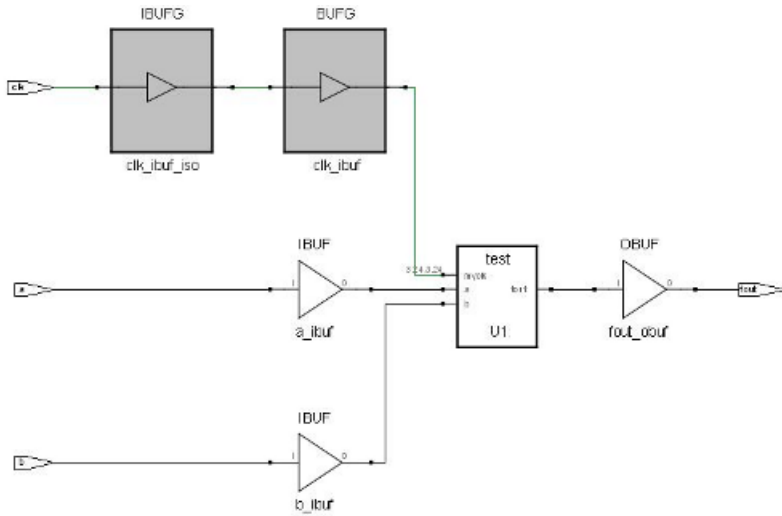
entity top is
generic (size: integer := 8);
port (fout : out std_logic_vector (size- 1 downto 0);
      a : in std_logic_vector (size- 1 downto 0);
      b : in std_logic_vector (size- 1 downto 0);
      clk : in std_logic
    );
end;

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;

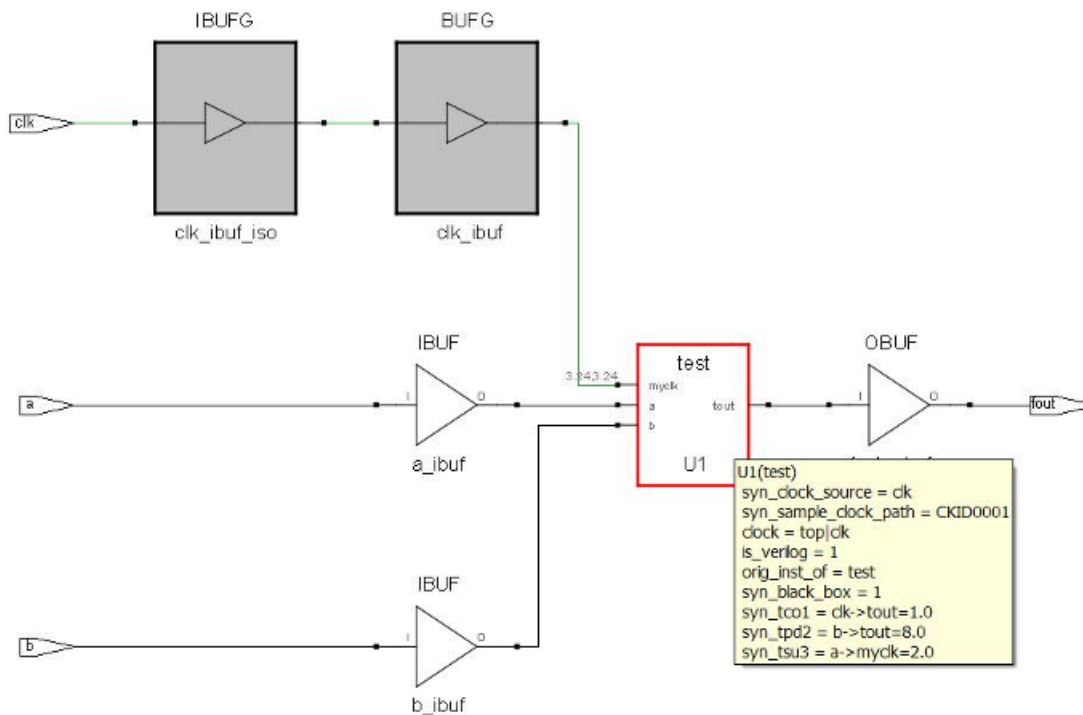
attribute syn_tcol : string;
attribute syn_tcol of test : component is
  "clk->tout = 1.0";
attribute syn_tpd1 : string;
attribute syn_tpd1 of test : component is
  "b->tout= 2.0";
attribute syn_tsul : string;
attribute syn_tsul of test : component is
  "a-> myclk = 1.2";
begin
U1 : test port map (fout, a, b, clk);
end;
```

## Effect of using syn\_tsu

This figure shows the HDL Analyst Technology view before using syn\_tsu:



This figure shows the HDL Analyst Technology view after using syn\_tsu:





## syn\_use\_carry\_chain

### Attribute

Turns on or off the carry chain implementation for arithmetic and combinational operators, depending on their input or output widths.

Vendor	Technology
Lattice	iCE40, iCE40UP, iCE40LM families

### Description

Use the attribute to turn on or off carry chain implementation for arithmetic and combinational operators, depending on their input or output widths. This attribute can be applied globally or on a particular instance. You can use any of the following operators: such as an adder, addmux, subtractor, accumulator, counter, or wide AND/OR gate and set the `syn_use_carry_chain` attribute value in the constraint file. This value controls whether or not carry chain is inferred.

### syn\_use\_carry\_chain Syntax

Global Support	Object
Yes	Instance

The following table summarizes the syntax in different files.

FDC	<pre>define_attribute {instanceName} syn_use_carry_chain {integerValue} define_global_attribute syn_use_carry_chain {integerValue}</pre>	<a href="#">FDC Example</a>
Verilog	<pre>object /* synthesis syn_use_carry_chain = integerValue */;</pre>	<a href="#">Verilog Examples</a>
VHDL	<pre>attribute syn_use_carry_chain : string; attribute syn_use_carry_chain of object : objectType is integerValue;</pre>	<a href="#">VHDL Examples</a>

Where:

- *instanceName* can be specified for an operator, such as an adder, addmux, subtractor, accumulator, counter, or wide AND/OR gate.
- *integerValue* can be specified as follows:
  - To disable carry chain (SB\_CARRY) usage, the value for this attribute should be greater than the input or output width of the logical or arithmetic operator.

Also, for example:

- For adders, carry chain inferencing occurs when the output width for the adder is less than or equal to the threshold value applied.
- For comparators with only one output such as the > operator, the threshold value should be set according to input widths of the comparator.

## FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	i:unl_state[1:0]	syn_use_carry_chain	10	integer	inference of carry chains

```
define_attribute {i:unl_state[1:0]} {syn_use_carry_chain} {10}
define_global_attribute {syn_use_carry_chain} {15}
```

## Verilog Examples

The following examples apply `syn_use_carry_chain` on an instance.

### Example 1: Verilog `syn_use_carry_chain` (Applied on a Instance)

```
// Example 1: Verilog syn_use_carry_chain (Applied on an Instance)
```

```
module test (dout, din1, din2, din3, clk);
    output [16:0] dout;
    input [15:0] din1, din2, din3;
    input clk;
```

```

reg [15:0] din1_reg, din2_reg;
reg [16:0] dout_reg, dout_reg1;
wire [16:0] data1;
assign data1 = dout_reg & din2_reg;
assign dout = dout_reg;
always @(posedge clk)
begin
  din1_reg <= din1;
  din2_reg <= din2;
  dout_reg1 <= din1_reg + data1;
  dout_reg <= dout_reg1 + din3;
end
endmodule

```

This code example contains two adders `dout_reg1` and `dout_reg`. Also, the following attribute is set in the constraint file:

```
define_attribute {i:dout_reg1_l[16:0]} {syn_use_carry_chain} {18}
```

The `syn_use_carry_chain` attribute defines the carry chain inference limit for `dout_reg1` to 18, which is more than its output width of 17. Therefore, this particular adder will not be mapped using `SB_CARRY`. To infer `SB_CARRY`, set the carry chain limit to 17 or less. However, the other adder instance infers the standard carry chain.

## Example 2: Verilog `syn_use_carry_chain` (Applied on a Instance)

```
// Example 2: Verilog syn_use_carry_chain (Applied on an Instance)
```

```

module test (a, b, c, d, gr, ngr, gr2, ngr2);

  parameter size = 8;

  input[size-1:0] a, c, d;

```

```
input[size-1:0]b;
output gr, gr2;
output ngr, ngr2;
```

```
reg gr, gr2, ngr2;
reg ngr;
```

```
always@(a or b)
begin
    if(a >= b)
        begin
            gr = 1'b1;
            ngr = 1'b0;
        end
    else
        begin
            gr = 1'b0;
            ngr = 1'b1;
        end
    end
end
```

```
always@(c or d)
begin
    if(c >= d)
        begin
            gr2 = 1'b1;
            ngr2 = 1'b0;
        end
    end
end
```

```

        end
    else
        begin
            gr2 = 1'b0;
            ngr2 = 1'b1;
        end
    end
end
endmodule

```

This code example contains two comparators; note that comparators must take into account their input width. Here the two comparators have an input width of 8; for which the ngr instance comparator is set to following constraint:

```
define_attribute {i:ngr} {syn_use_carry_chain} {100}
```

In this case, the syn\_use\_carry\_chain value is greater than the input width of the comparator. Therefore, the software does not infer SB\_CARRY for instance ngr. However, instance ngr2 infers the standard carry chain.

The following examples apply syn\_use\_carry\_chain globally.

### Example 1: Verilog syn\_use\_carry\_chain (Applied Globally)

```
// Example 3: Verilog syn_use_carry_chain (Applied Globally)
```

```

module adder_12 (
    a_in,
    b_in,
    add_out,
    c_out
);

    input [11:0] a_in;

```

```
input [11:0] b_in;

output [11:0] add_out;
output c_out;

assign {c_out,add_out} = a_in + b_in;

endmodule
```

For this example, the output for the adder has 13 bits (this includes c\_out). To infer the carry chain, set this attribute to the following:

```
define_global_attribute {syn_use_carry_chain} {13}
```

Therefore, any value that is less than or equal to 13 will infer carry chain. Carry chain is disabled when you set the attribute value greater than 13; so you can set this value to 14 in this case.

## Example 2: Verilog syn\_use\_carry\_chain (Applied Globally)

```
// Example 4: Verilog syn_use_carry_chain (Applied Globally)
```

```
module comp ( a, b, y );
input [7:0] a,b;
output y;

assign y = ( a > b )? 1'b1 : 1'b0;

endmodule
```

In this example, the output width is 1 bit. For comparators such as >, you must take into account the input width of 8. To infer carry chain, set this attribute to the following:

```
define_global_attribute {syn_use_carry_chain} {8}
```

---

Therefore, any value that is less than or equal to 8 will infer carry chain. Carry chain is disabled when you set the attribute value greater than 8; so you can set this value to 9 in this case.

## VHDL Examples

Here are VHDL code examples for the comparable Verilog examples above.

### Example 1: VHDL syn\_use\_carry\_chain (Applied on an Instance)

```
-- Example 1: VHDL syn_use_carry_chain (Applied on an Instance)
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity test is
    port (
        dout : out std_logic_vector( 16 downto 0 );
        din1 : in std_logic_vector( 16 downto 0 );
        din2 : in std_logic_vector( 16 downto 0 );
        din3 : in std_logic_vector( 16 downto 0 );
        clk : in std_logic
    );
end entity;

architecture rtl of test is
    signal din1_reg : std_logic_vector( 16 downto 0 );
    signal din2_reg : std_logic_vector( 16 downto 0 );
    signal dout_reg : std_logic_vector( 16 downto 0 );
    signal dout_reg1 : std_logic_vector( 16 downto 0 );
    signal data1 : std_logic_vector( 16 downto 0 );
begin
```



```

    data1 <= ( dout_reg and din2_reg ) ;
    dout <= dout_reg;
    process
    begin
        wait until ( clk'EVENT and ( clk = '1' ) ) ;
        din1_reg <= din1;
        din2_reg <= din2;
        dout_reg1 <= ( din1_reg + data1 ) ;
        dout_reg <= ( dout_reg1 + din3 ) ;
    end process;
end;
```

## Example 2: VHDL syn\_use\_carry\_chain (Applied on an Instance)

-- Example 2: VHDL syn\_use\_carry\_chain (Applied on an Instance)

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    generic (
        size : INTEGER := 8
    );
    port (
        a : in std_logic_vector( ( size - 1 ) downto 0 );
        c : in std_logic_vector( ( size - 1 ) downto 0 );
        d : in std_logic_vector( ( size - 1 ) downto 0 );
        b : in std_logic_vector( ( size - 1 ) downto 0 );
        gr : out std_logic;
```

```
        gr2 : out std_logic;
        ngr : out std_logic;
        ngr2 : out std_logic
    );
end entity;
```

architecture rtl of test is

```
begin
    process
    begin
        wait on a, b;
        if ( ( a >= b ) ) then
            gr <= '1';
            ngr <= '0';
        else
            gr <= '0';
            ngr <= '1';
        end if;
    end process;
    process
    begin
        wait on c, d;
        if ( ( c >= d ) ) then
            gr2 <= '1';
            ngr2 <= '0';
        else
```

```

        gr2 <= '0';
        ngr2 <= '1';
    end if;
end process;
end;
```

### Example 1: VHDL syn\_use\_carry\_chain (Applied Globally)

-- Example 3: VHDL syn\_use\_carry\_chain (Applied Globally)

```

library ieee;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity adder_12 is
    port (
        a_in : in std_logic_vector( 11 downto 0 );
        b_in : in std_logic_vector( 11 downto 0 );
        add_out : out std_logic_vector( 11 downto 0 );
        c_out : out std_logic_vector( 11 downto 0 )
    );
end entity;

architecture rtl of adder_12 is
    begin

        add_out <= (a_in xor b_in);
```

```
c_out    <= (a_in and b_in);
```

```
end;
```

## Example 2: VHDL syn\_use\_carry\_chain (Applied Globally)

-- Example 4: VHDL syn\_use\_carry\_chain (Applied Globally)

```
library ieee;
```

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

```
use ieee.numeric_std.all;
```

```
entity comp is
```

```
    port (
```

```
        a : in std_logic_vector( 7 downto 0 );
```

```
        b : in std_logic_vector( 7 downto 0 );
```

```
        y : out std_logic
```

```
    );
```

```
end entity;
```

```
architecture rtl of comp is
```

```
begin
```

```
    process (a,b)
```

```
    begin
```

```
        if (a > b) then
```

```
            y <= '1';
```

```
        else
```

```
            y <= '0';
```

```
        end if;
```

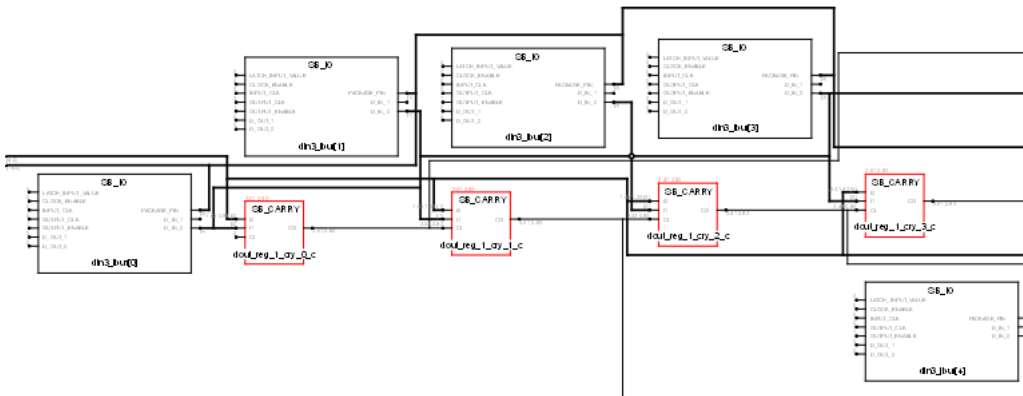
```

end process;

end;
```

## Effect of Using syn\_use\_carry\_chain

In the following example, the carry chain inference limit is less than the output limit for one of the adders (dout\_reg\_1). Therefore, the SB\_CARRY primitives are inferred for this adder as shown as shown in the Technology view below:



## syn\_useenables

### *Attribute*

Controls the use of clock-enable registers within a design.

Vendor	Technology
Lattice	EC, SC, XP families

### syn\_useenables Values

Default	Global	Object Type
1/true	No	Register

Value	Description
1/true	Infers registers with clock-enable pins
0/false	Uses external logic to generate the clock-enable function for the register

### 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	<b>define attribute {<i>register signal</i>} syn_useenables {0 1}</b>	<a href="#">SCOPE Example</a>
Verilog	<i>object</i> /* synthesis syn_useenables = "0 1" */;	<a href="#">Verilog Example</a>
VHDL	<b>attribute syn_useenables of <i>object</i> : <i>objectType</i> is "true/false";</b>	<a href="#">VHDL Example</a>

## SCOPE 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;

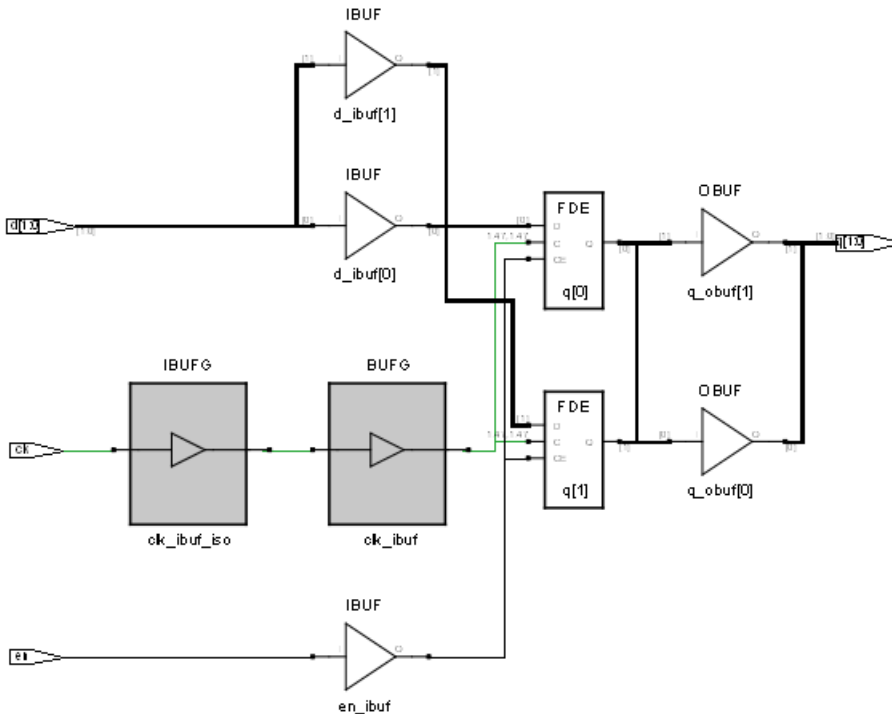
entity syn_useenables is
  port (d : in std_logic_vector(1 downto 0);
        en,clk : in std_logic;
        q : out std_logic_vector(1 downto 0) );
  attribute syn_useenables: boolean;
  attribute syn_useenables of q: signal is false;
end;

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;
```

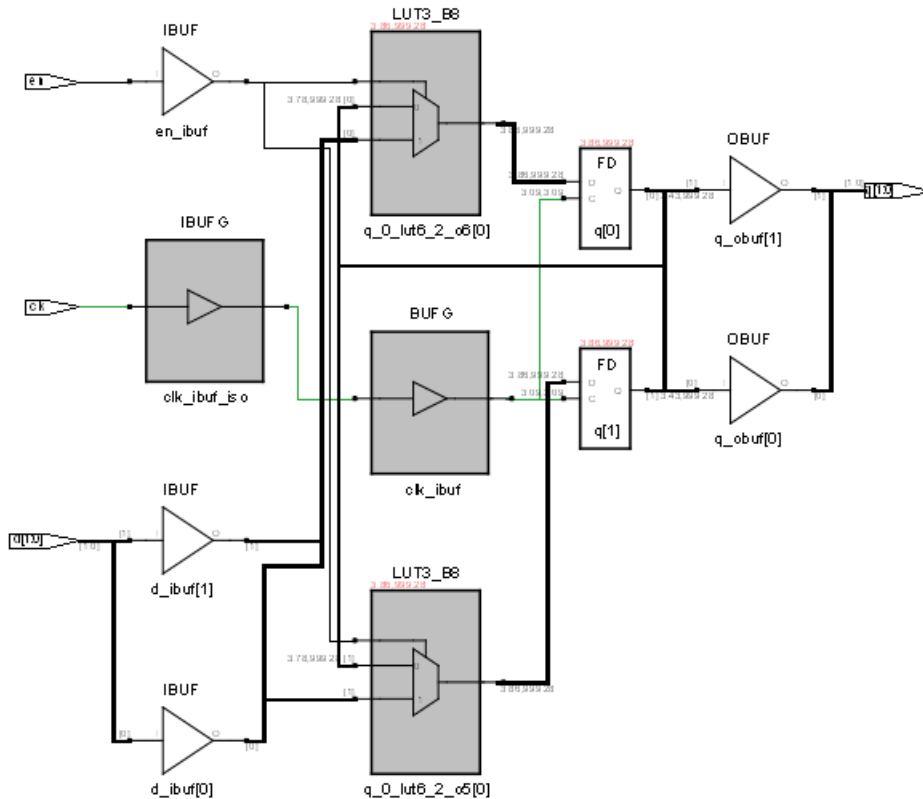


## Effect of Using syn\_useenables

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.

Vendor	Technologies
--------	--------------

Lattice	iCE40, iCE40UP
---------	----------------

### syn\_useioff Values

Value	Description
-------	-------------

1/true	Packs the registers into I/O pad cells.
--------	---

0/false	Do not pack the registers into 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.

### 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.

## 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

### 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

### 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

## syn\_useioff Syntax Specification

The following table summarizes the syntax in different files. See [Object Considerations, on page 299](#) [Examples of syn\\_useioff on Different Ports, on page 303](#) for descriptions of the ports where you can set the attribute.

FDC	define_attribute { <i>object</i> } syn_useioff {1 0} define_global_attribute syn_useioff {1 0}	<a href="#">SCOPE Example</a>
Verilog	<i>object</i> /* synthesis syn_useioff = {1 0} */;	<a href="#">Verilog Example</a>
VHDL	attribute syn_useioff of <i>object</i> : <i>objectType</i> is {true false};	<a href="#">VHDL Example</a>

## SCOPE Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	port	p:q	syn_useioff	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 clk_c (direction INPUT))
        (port q_c (direction OUTPUT))
      )
      (contents
        (instance qreg (viewRef PRIM (cellRef FD (libraryRef UNILIB)))
          (property IOB (string "FALSE"))
        )
      )
    )
  )
)
```

# 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 33. These directives are implemented the same in the source code.

## translate\_off/translate\_on Syntax

Verilog	<code>/* synthesis translate_off */</code> <code>/* synthesis translate_on */</code>
---------	---

VHDL	<code>synthesis translate_off</code> <code>synthesis translate_on</code>
------	---

## 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. See [synthesis Macro, on page 127](#) for information.

## 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.

See [VHDL Attribute and Directive Syntax, on page 414](#) for different ways to specify VHDL attributes and 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 synthesis translate_off/translate_on is ignored
  --by the synthesis tool hence only
  --the adder circuit above is implemented not the multiplier circuit

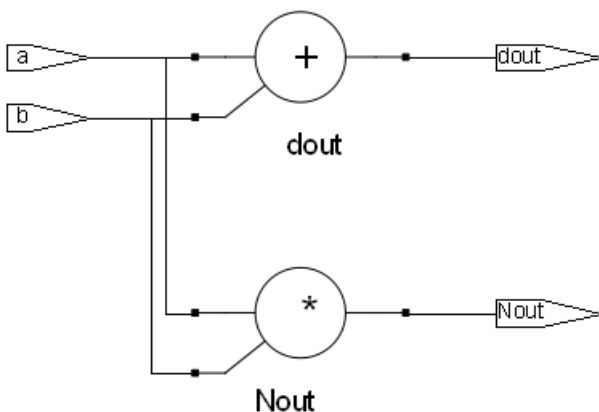
```

below:

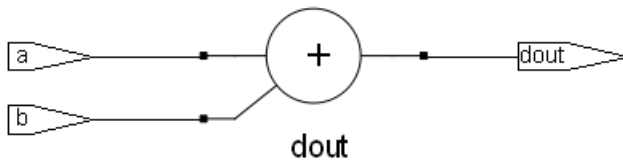
```
--synthesis translate_off
  Nout <= a * b;
--synthesis translate_on
end;
```

## Effects of Using translate\_off/translate\_on

Here is the RTL view before applying the attribute.



This is the RTL view after applying the attribute.



:

translate\_off/translate\_on

---

# Index

---

## A

attributes  
    custom [80](#)  
    global attribute summary [11](#)  
    specifying in the SCOPE spreadsheet [8](#)  
    specifying, overview of methods [8](#)  
    syn\_direct\_reset [56](#)  
Attributes panel, SCOPE spreadsheet [8](#)

## B

black box directives  
    black\_box\_pad\_pin [15](#)  
    black\_box\_tri\_pins [20](#)  
    syn\_black\_box [44](#)  
    syn\_force\_seq\_prim [88](#)  
    syn\_isclock [123](#)  
    syn\_tco [260](#)  
    syn\_tpd [266](#)  
    syn\_tristate [272](#)  
    syn\_tsu [275](#)  
black boxes  
    directives. *See* black box directives  
    source code directives [45](#)  
    syn\_gatedclk\_clock\_en directive [91](#)  
    timing directives [266](#)  
black\_box\_pad\_pin directive [15](#)  
black\_box\_tri\_pins directive [20](#)  
buffers  
    clock. *See* clock buffers  
    global. *See* global buffers

## C

case statement  
    default [25](#)  
clock buffers  
    assigning resources [154](#)  
clock enables  
    inferring with syn\_direct\_enable [51](#)  
    net assignment [51](#)

clocks  
    on black boxes [123](#)  
code  
    ignoring with pragma translate off/on [33](#)  
compile points  
    allowed resources [40](#)  
compiler  
    loop iteration, loop\_limit [28](#)  
    loop iteration, syn\_looplevelimit [133](#)  
custom attributes [80](#)

## D

define\_attribute  
    syntax [9](#)  
define\_false\_path  
    using with syn\_keep [127](#)  
define\_global\_attribute  
    summary [11](#)  
    syntax [9](#)  
define\_multicycle\_path  
    using with syn\_keep [127](#)

## E

edif file  
    scalar and array ports [150](#)  
    syn\_noarrayports attribute [150](#)  
enumerated types  
    syn\_enum\_encoding directive [79](#)

## F

fanout limits  
    overriding default [135](#)  
    syn\_maxfan attribute [135](#)  
fix gated clocks  
    syn\_gatedclk\_clock\_en directive [91](#)  
FSMs  
    syn\_encoding attribute [69](#)

full\_case directive [23](#)

## G

gated clocks

    syn\_force\_seq\_prim directive [88](#)

global attributes summary [11](#)

global buffers

    defining [99](#)

## H

hierarchy

    flattening with syn\_hier [106](#)

## I

I/O buffers

    inserting [119](#)

    specifying I/O standards [181](#)

I/O packing

    disabling with syn\_replicate [221](#)

instances

    preserving with syn\_noprune [165](#)

## L

Lattice

    loc attribute [27](#)

loc attribute (Lattice) [27](#)

loop\_limit directive [28](#)

## M

multicycle paths

    syn\_reference\_clock [219](#)

## N

nets

    preserving with syn\_keep [127](#)

## P

parallel\_case directive [31](#)

pin locations

    Lattice [27](#)

pipelining

    syn\_pipeline attribute [185](#)

pragma translate\_off directive [33](#)

pragma translate\_on directive [33](#)

priority encoding [31](#)

probes

    inserting [195](#)

## R

RAMs

    implementation styles [203](#)

    technology support [205](#)

registers

    preserving with syn\_preserve [190](#)

replication

    disabling [221](#)

resource sharing

    syn\_sharing directive [237](#)

retiming

    syn\_allow\_retiming attribute [36](#)

## S

SCOPE spreadsheet

    Attributes panel [8](#)

sequential optimization, preventing with syn-  
    \_preserve [190](#)

simulation mismatches

    full\_case directive [26](#)

state machines

    enumerated types [79](#)

    extracting [246](#), [254](#)

syn\_allow\_retiming attribute [36](#)

syn\_black\_box directive [44](#)

syn\_direct\_enable attribute [51](#)

syn\_direct\_reset Attribute [56](#)

syn\_direct\_set attribute [61](#)

syn\_encoding

    compared with syn\_enum\_encoding  
    directive [80](#), [81](#)

    using with enum\_encoding [81](#)

syn\_encoding attribute [69](#)

syn\_enum\_encoding

    using with enum\_encoding [81](#)

syn\_enum\_encoding directive [79](#)

    compared with syn\_encoding attribute  
    [80](#), [81](#)

syn\_force\_pads attribute [85](#)

syn\_force\_seq\_prim directive [88](#)

syn\_gatedclk\_clock\_en directive [91](#)

syn\_gatedclk\_clock\_en\_polarity directive [94](#)

syn\_global\_buffers attribute [99](#)



[syn\\_hier attribute 106](#)  
[syn\\_insert\\_buffer attribute 116](#)  
[syn\\_insert\\_pad attribute 119](#)  
[syn\\_isclock directive 123](#)  
[syn\\_keep](#)  
     [compared with syn\\_preserve and syn\\_noprune directives 130](#)  
[syn\\_keep directive 127](#)  
[syn\\_looplimit directive 133](#)  
[syn\\_maxfan attribute 135](#)  
[syn\\_multstyle attribute 141](#)  
[syn\\_netlist\\_hierarchy attribute 144](#)  
[syn\\_noarrayports attribute 150](#)  
[syn\\_noclockbuf attribute 154](#)  
     [using with fanout guides 136](#)  
[syn\\_noclockpad attribute 160](#)  
[syn\\_noprune directive 165](#)  
[syn\\_pad\\_type attribute 181](#)  
[syn\\_pipeline attribute 185](#)  
[syn\\_preserve](#)  
     [compared with syn\\_keep and syn\\_noprune 191](#)  
[syn\\_preserve directive 190](#)  
[syn\\_probe attribute 195](#)  
[syn\\_ramstyle attribute 203](#)  
[syn\\_reference\\_clock attribute 219](#)  
[syn\\_replicate](#)  
     [using with fanout guides 136](#)  
[syn\\_replicate attribute 221](#)  
[syn\\_romstyle attribute 226](#)  
[syn\\_safe\\_case directive 231](#)  
[syn\\_safefsm\\_pipe directive 234](#)  
[syn\\_sharing directive 237](#)  
[syn\\_shift\\_resetphase 242](#)  
[syn\\_smhigh effort attribute 246](#)  
[syn\\_srlstyle attribute 251](#)  
[syn\\_state\\_machine attribute 246](#)  
[syn\\_state\\_machine directive 254](#)  
[syn\\_tco directive 260](#)  
[syn\\_tpd directive 266](#)  
     [black-box timing 266, 275](#)  
[syn\\_tristate directive 272](#)  
[syn\\_tsu directive 275](#)  
     [black-box timing 275](#)  
[syn\\_use\\_carry\\_chain attribute 281](#)  
[syn\\_useenables attribute 294](#)  
[synthesis\\_off directive 307](#)

[synthesis\\_on directive 307](#)  
 SystemVerilog  
     [ignoring code with synthesis\\_off/on 307](#)

## T

[timing](#)  
     [syn\\_tco directive 260](#)  
     [syn\\_tpd directive 266](#)  
     [syn\\_tsu directive 275](#)  
[translate\\_off directive 306](#)  
[translate\\_on directive 306](#)  
[tristates](#)  
     [black\\_box\\_tri\\_pins directive 20](#)  
     [syn\\_tristate directive 272](#)

## V

[Verilog](#)  
     [ignoring code with translate off/on 306](#)  
     [syn\\_keep on multiple nets 128](#)

## W

[wires, preserving with syn\\_keep directive 127](#)

