

Verification Continuum™

FPGA Synthesis

Attribute Reference Manual

March 2021

SYNOPSYS®

<http://solvnet.synopsys.com>

Synopsys Confidential Information

Copyright Notice and Proprietary Information

© 2021 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Free and Open-Source Licensing Notices

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

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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

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

Third-Party Links

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

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

March 2021

Synopsys Statement on Inclusivity and Diversity

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

Contents

Chapter 1: Introduction

How Attributes and Directives are Specified	10
The SCOPE Attributes Tab	10
The Compiler Directives File	12
Summary of Attributes and Directives	19
Attribute and Directive Summary Table	20
Summary of Global Attributes	22
ace_useioff	25
alsloc	31
alspin	35
alspreserve	39
altera_io_powerup	43
altera_logicclock_location	47
altera_logicclock_size	51
black_box_pad_pin	55
black_box_tri_pins	61
CLOCK_DEDICATED_ROUTE	65
diff_term	69
full_case	73
loc	77
loop_limit	79
orca_padtype	83
orca_props	87
parallel_case	91
pragma translate_off/pragma translate_on	95
syn_allow_retimig	99
syn_allowed_resources	105
syn_append_submodules	129
syn_assign_to_region	131
syn_assign_to_slr	133
syn_async_reg	137
syn_auto_insert_bufg	141

syn_auto_insert_bufgmux	145
syn_black_box	149
syn_bram_cascade_height	157
syn_clean_reset	161
syn_clock_gmux_proxy	165
syn_clock_priority	169
syn_connect_hrefs	173
syn_cp_use_fast_synthesis	181
syn_diff_io	183
syn_direct_enable	189
syn_direct_reset	195
syn_direct_set	201
syn_disable_purifyclock	207
syn_donot_infer_oddr	211
syn_donot_infer_iddr	215
syn_DSPstyle	219
syn_edif_name_length	239
syn_edif_scalar_format	243
syn_edif_bit_format	255
syn_encoding	259
syn_enum_encoding	269
syn_fast_auto	275
syn_force_pads	283
syn_force_seq_prim	287
syn_forward_io_constraints	291
syn_fsm_correction	295
syn_gatedclk_clock_en	301
syn_gatedclk_clock_en_polarity	305
syn_global_buffers	311
syn_hier	319
syn_highrel_ioconnector	329
syn_implement	337
syn_insert_buffer	339
syn_insert_pad	351
syn_isclock	355
syn_keep	359
syn_loc	367
syn_looplimit	373
syn_macro	375
syn_map_dffrs	381
syn_max_memsize_reg	387
syn_maxfan	393

syn_multstyle	401
syn_netlist_hierarchy	413
syn_no_compile_point	421
syn_noarrayports	425
syn_noclockbuf	429
syn_noclockpad	435
syn_noprune	441
syn_pad_type	459
syn_pipeline	465
syn_preserve	471
syn_preserve_rom	477
syn_preserve_sr_priority	481
syn_probe	487
syn_radhardlevel	495
syn_ram_write_mem	505
syn_ramstyle	509
syn_reduce_controlset_size	525
syn_reduce_controlset_size (Achronix)	539
syn_reference_clock	545
syn_register_correction	549
syn_rename_module	555
syn_replicate	557
syn_resources	563
syn_ret_lib_cell_type	573
syn_ret_type	575
syn_romstyle	577
syn_romstyle (Microchip)	586
syn_rw_conflict_logic	591
syn_safe_case	599
syn_safefsm_pipe	603
syn_sharing	607
syn_shift_resetphase	613
syn_slow	619
syn_smhigheffort	623
syn_srl_mindepth	629
syn_srlstyle	633
syn_state_machine	643
syn_tco<n>	651
syn_tpd<n>	657
syn_tristate	663
syn_tristatetomux	667
syn_tsu<n>	675

syn_unconnected_inputs	681
syn_unique_inst_module	691
syn_upf_ret_port_type	695
syn_use_carry_chain	697
syn_useenables	711
syn_useioff	717
syn_useioff and ace_useioff	726
syn_user_instance	733
syn_vote_loops	739
syn_vote_register	743
translate_off/translate_on	749
xc_area_group	753
xc_clockbuftype	759
xc_fast	765
xc_fast_auto	769
xc_global_buffers	775
xc_isgsr	779
xc_loc	785
xc_map	789
xc_padtype	795
xc_props	797
xc_pullup/xc_pulldown	805
xc_rloc	811
xc_slow	817
xc_use_keep_hierarchy	819
xc_use_timespec_for_io	825
xc_uset	831

CHAPTER 1

Introduction

This document is part of a set that includes reference and procedural information for the Synopsys® FPGA synthesis tools.

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 10
 - [Summary of Attributes and Directives](#), on page 19

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 137](#) 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 133](#).

See the following for descriptions of the interfaces available to enter attribute information:

- [The SCOPE Attributes Tab, on page 10](#)
- [The Compiler Directives File, on page 12](#)

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.
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 143](#) 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 syn_black_box, on page 149 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 22](#) for more details on specifying global attributes in the synthesis environment.

The Compiler Directives File

Synplify Premier

You can use a compiler directives file to specify directives to be added to the source code. During compilation, the tool passes all active compiler constraint files to the compiler. The compiler references the object names in these files with the original RTL objects to assign the corresponding directive. For details about specifying directives with this file, see [Specifying Directives in a CDC File, on page 141](#) in the *User Guide*.

The remainder of this section consists of code examples.

Example - Verilog Compiled into Default Library

```
//Example -- Verilog compiled into default library
//Entry in .cdc file:
// define_directive {v:sub} {syn_black_box} {1}
module top (
    input clock,
    input reset,
    input din,
```

```
input din1,
output dout );
sub UUT (clock, reset, din, din1, dout);
endmodule

module sub (
    input clock,
    input reset,
    input din,
    input din1,
    output reg dout );
    always@ (posedge clock)
    begin
        if (reset == 1'b1)
            dout = 0;
        else
            dout = din | din1;
    end
endmodule
```

Example - Verilog Compiled into a Defined Library

```
//Example -- Verilog compiled into defined library
//Entry in .cdc file (compiles submodule into MyLib):
// define_directive {v:MyLib.sub} {syn_black_box} {1}
//top.v
module top (
    input a,
    input b,
```

```
output c,  
output d );  
sub inst1 (.a(a), .b(b), .c(c), .d(d) );  
endmodule  
//sub.v  
  
module sub (  
input a,  
input b,  
output c,  
output d );  
assign c = a & b;  
assign d = top.a;  
endmodule
```

Example - VHDL Compiled into Default Library

```
--Example -- VHDL compiled into default library  
--Entry in .cdc file:  
-- define_directive {v:sub} {syn_black_box} {1}  
--top.vhd  
  
library ieee;  
use ieee.std_logic_1164.all;  
entity top is  
port (clk : in std_logic;  
din : in std_logic_vector(3 downto 0);  
din1 : in std_logic_vector(3 downto 0);  
dout : out std_logic_vector(3 downto 0) );  
end top;
```

```

architecture RTL of top is
component sub
port (clk : in std_logic;
din : in std_logic_vector(3 downto 0);
din1 : in std_logic_vector(3 downto 0);
dout : out std_logic_vector(3 downto 0) );
end component;
begin
UUT : sub port map (
clk => clk,
din => din,
din1 => din1,
dout => dout );
end RTL;
--sub.vhd
library ieee;
use ieee.std_logic_1164.all;
entity sub is
port (clk : in std_logic;
din : in std_logic_vector(3 downto 0);
din1 : in std_logic_vector(3 downto 0);
dout : out std_logic_vector(3 downto 0) );
end sub;
architecture RTL of sub is
begin
process (clk)
begin

```

```
if rising_edge(clk) then
  dout <= din or din1;
end if;
end process;
end RTL;
```

Example - VHDL Compiled into a Defined Library

```
--Example -- VHDL compiled into defined library
--Entry in .cdc file (compiles submodule into MyLib):
-- define_directive {v:MyLib.sub(RTL_1)} {syn_black_box} {1}
--Top.vhd
library ieee;
use ieee.std_logic_1164.all;
library MyLib;
use MyLib.all;
entity top is
port (clk : in std_logic;
      din : in std_logic;
      dout : out std_logic );
end top;
architecture RTL of top is
signal inter : std_logic;
component sub
port (clk : in std_logic;
      din : in std_logic;
      dout : out std_logic );
```

```
end component;

for UUT1 : sub
use entity work.sub(RTL_1)
port map ( clk => clk, din => din, dout => dout);

for UUT2 : sub
use entity work.sub(RTL_2)
port map ( clk => clk, din => din, dout => dout);

begin
  UUT1 : entity MyLib.sub(RTL_1)
  port map (
    clk => clk,
    din => din,
    dout => inter );
  UUT2 : sub
  port map (
    clk => clk,
    din => inter,
    dout => dout );
end RTL;
--sub.vhd

library ieee;
use ieee.std_logic_1164.all;
entity sub is
port (
  clk : in std_logic;
  din : in std_logic;
  dout : out std_logic );
```

```
end sub;

architecture RTL_1 of sub is
begin
process (clk)
begin
if rising_edge(clk) then
dout <= din;
end if;
end process;
end RTL_1;

architecture RTL_2 of sub is
begin
process (clk)
begin
if rising_edge(clk) then
dout <= not din;
end if;
end process;
end RTL_2;
```

Summary of Attributes and Directives

The following sections summarize the synthesis attributes and directives:

- [Attribute and Directive Summary Table](#), on page 20
- [Summary of Global Attributes](#), on page 22

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

Attribute and Directive Summary Table

ace_useiooff	alsloc	alspin
alspreserve	altera_io_powerup	altera_logiclock_location
altera_logicclock_size	black_box_pad_pin	black_box_tri_pins
CLOCK_DEDICATED_ROU TE	diff_term	full_case
loc	loop_limit	orca_padtype
orca_props	parallel_case	pragma translate_off/pragma translate_on
syn_allow_retiming	syn_allowed_resources	syn_append_submodule s
syn_assign_to_region	syn_assign_to_slr	syn_async_reg
syn_auto_insert_bufg x	syn_auto_insert_bufgmu	syn_black_box
syn_bram_cascade_height	syn_clean_reset	syn_clock_gmux_proxy
syn_clock_priority	syn_connect_hrefs	syn_cp_use_fast_synthet is
syn_diff_io	syn_direct_enable	syn_direct_reset
syn_direct_set	syn_disable_purifyclock	syn_donot_infer_oddr
syn_donot_infer_iddr	syn_DSPstyle	syn_edif_name_length
syn_edif_scalar_format	syn_edif_bit_format	syn_encoding
syn_enum_encoding	syn_fast_auto	syn_force_pads
syn_force_seq_prim	syn_forward_io_constraint	syn_fsm_correction
syn_gatedclk_clock_en	syn_gatedclk_clock_en_p olarity	syn_global_buffers
syn_hier	syn_highrel_ioconnector	syn_implement
syn_insert_buffer	syn_insert_pad	syn_isclock
syn_keep	syn_loc	syn_looplmit
syn_macro	syn_map_dffrs	syn_max_memsize_reg

syn_maxfan	syn_multstyle	syn_netlist_hierarchy
syn_no_compile_point	syn_noarrayports	syn_noclockbuf
syn_noclockpad	syn_noprune	syn_pad_type
syn_pipeline	syn_preserve	syn_preserve_rom
syn_preserve_sr_priority	syn_probe	syn_radhardlevel
syn_ram_write_mem	syn_ramstyle	syn_reduce_controlset_size
syn_reference_clock	syn_register_correction	syn_rename_module
syn_replicate	syn_resources	syn_ret_lib_cell_type
syn_ret_type	syn_romstyle	syn_rw_conflict_logic
syn_safe_case	syn_safefsm_pipe	syn_sharing
syn_shift_resetphase	syn_slow	syn_smhigheffort
syn_srl_minddepth	syn_srlstyle	syn_state_machine
syn_tco< n >	syn_tmr_retain_dram_sr 1	syn_tpd< n >
syn_tristate	syn_tristatetomux	syn_tsu< n >
syn_unconnected_inputs	syn_unique_inst_module	syn_upf_ret_port_type
syn_use_carry_chain	syn_useenables	syn_useioff
syn_useioff and ace_useioff	syn_user_instance	syn_vote_loops
syn_vote_register	translate_off/translate_on	xc_area_group
xc_clockbuftype	xc_fast	xc_fast_auto
xc_global_buffers	xc_isgsr	xc_loc
xc_map	xc_padtype	xc_props
xc_pullup/xc_pulldown	xc_rloc	xc_slow
xc_use_keep_hierarchy	xc_use_timespec_for_io	xc_uset

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](#), on page 17.

Global Attribute	Can Also Be Set On Design Objects
<code>syn_allow_retiming</code>	x
<code>syn_allowed_resources</code>	x
<code>syn_async_reg</code>	x
<code>syn_auto_insert_bufgmx</code>	x
<code>syn_auto_insert_bufgmx</code>	
<code>syn_clean_reset</code>	
<code>syn_clock_gmux_proxy</code>	x
<code>syn_disable_purifyclock</code>	x
<code>syn_dspstyle</code>	x
<code>syn_edif_bit_format</code>	
<code>syn_edif_name_length</code>	
<code>syn_edif_scalar_format</code>	
<code>syn_force_pads</code>	x
<code>syn_forward_io_constraints</code>	
<code>syn_global_buffers</code>	x

Global Attribute	Can Also Be Set On Design Objects
syn_hier	x
syn_map_dffrs	
syn_max_memsize_reg	
syn_multstyle	x
syn_netlist_hierarchy	
syn_noarrayports	
syn_noclockbuf	x
syn_preserve_sr_priority	
syn_ramstyle	x
syn_reduce_controlset_size	
syn_replicate	x
syn_romstyle	x
syn_safefsm_pipe	x
syn_rw_conflict_logic	x
syn_srl_mindepth	
syn_srlstyle	x
syn_useioff	x
syn_useioff and ace_useioff	
syn_user_instance	x
xc_fast_auto	
xc_global_buffers	
xc_use_keep_hierarchy	x
xc_use_timespec_for_io	

ace_useioff

Attribute

You can use ace_useioff in conjunction with the syn_useioff attribute.

The tool supports the ace_useioff=0 or 1 or syn_useioff=0 or 1 property and forward annotates it in the netlist (.vm file).

Vendor	Technologies
Achronix	Speedster7t

Attribute Support

When you apply the syn_useioff=1|0 or ace_useioff=1|0 attribute on registers, instances, or ports, the tool forward annotates the property on that instance in the netlist (.vm file). Additionally, the ace_useioff=1|0 attribute can be applied on nets to leverage retiming on the corresponding register.

- If the ace_useioff/syn_useioff attribute is applied globally, then the property is forward annotated to the top level.
- Do not apply the syn_useioff attribute on nets; use the ace_useioff attribute instead. The tool will issue a warning message, when this rule is violated.
- The tool does not retime registers when the ace_useioff or syn_useioff attribute is set to 1.
- The tool does not retime registers driving or driven by the nets when the ace_useioff or syn_useioff attribute is set to 1.
- Retiming on registers is allowed only when both the attributes are set to 0.
- If the ace_useioff or syn_useioff attribute is set to 1 globally, then retiming will not be disabled. Retiming prevents the tool from achieving the benefits of better performance/timing QoR. Only registers that are

connected directly or indirectly (separated by hierarchy) to the top-level ports are prevented from being retimed.

- If the attribute is applied at the submodule level, then the tool ignores this property and issues a warning message.

Syntax Specification

Apply the `syn_useioff` and `ace_useioff` attributes in the HDL source code or constraint file (.fdc file).

Verilog Syntax

Valid syntax for applying these attributes in the Verilog source code are shown below.

- Attribute is applied on input/output registers:

Output Register `d /*synthesis syn_useioff=1|0*/ OR /*ace_useioff=1 or 0*/;`

Input Register `din /*synthesis syn_useioff=1|0*/ OR /*ace_useioff=1 or 0*/;`

- Attribute is applied on input/output ports and nets.

Output `d /*synthesis syn_useioff=1|0*/ OR /*ace_useioff=1 or 0*/;`

Input `din /*synthesis syn_useioff=1|0*/ OR /*ace_useioff=1 or 0*/;`

Wire `tmp /*ace_useioff=1*/`

VHDL Syntax

You can apply these attributes in the VHDL source code as follows:

`attribute syn_useioff of data_out : signal is "true|false";1`

1. Similarly, `ace_useioff` can be declared and applied.

FDC File Syntax

The `syn_useioff/ace_useioff` attribute can be applied on the input/output ports in the constraint file is shown below.

Output	define_attribute {p:dout} {syn_useioff} {0 1}
Input	define_attribute {p:din[7:0]} {ace_useioff} {0 1}
Register	define_attribute {i:U1.areg[7:0]} {syn_useioff} {1}
Global	define_global_attribute {syn_useioff} {0 1}

Example 1

In the following example, the `syn_useioff` attribute is set on the top-level output port `q`. The top-level output port `q` is marked with the property `syn_useioff="1"` in the `vm` netlist. Since the top-level output port is not driven within the sequential block, the attribute is only forward annotated on the port. In this case, the top-level output port can be connected to a register from within the lower-level submodules.

```
module test_top (
  input clk,
  input d,
  output q/* synthesis syn_useioff=1 */;
    test U (
      .clk(clk),
      .d(d),
      .q(q));
endmodule
```

For the example above, the `syn_useioff` property is forward annotated to the output port ‘q’ in the .vm netlist.

(*syn useioff = 1*) output q;

Example 2

The following Verilog example applies the `syn_useioff` attribute on a register of the submodule.

```

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=1 */;
    assign q = qreg;
    always @ (posedge clk)
    begin
        temp <= d;
        qreg <= temp;
    end
endmodule

```

For the example above, the `syn_useioff=1` property is forward annotated on the register instance in the .vm netlist.

```
(*syn_useioff = 1*) DFF qreg;
```

Example 3

The following Verilog example applies the `syn_useioff` attribute on an output port in a lower-level module. Since the output port is driven from within the sequential block, the attribute is forward annotated on the output register.

```

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 reg q/* synthesis syn_useioff=1 */;
  reg temp;

  //reg qreg;

  //assign q = qreg;
  always @(posedge clk)
    begin
      temp <= d;
      q <= temp;
    end
end
endmodule
```

For the example above, the `syn_useoff=1` property is forward annotated to register instance `q` in the netlist.

(*syn useioff = 1*) DDF q;

Example 4

This example applies the `syn_useioff` attribute in the VHDL source code.

```
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;
```

:

In the example above, the `syn_useioff=0` property is forward annotated to the register `data_out` in the netlist.

```
(*syn_useioff = 0*) dff data_out;
```

Example 5

The `syn_useioff` and `ace_useioff` attributes are applied globally at the top level in this example.

```
module test_top (
    input clk,
    input d,
    output q) /* synthesis syn_useioff=1 */;
    test U (
        .clk(clk),
        .d(d),
        .q(q));
endmodule

module test(
    input clk,
    input d,
    output q);
    reg temp;
    reg qreg;
    assign q = qreg;
    always @ (posedge clk)
    begin
        temp <= d;
        qreg <= temp;
    end
endmodule
```

In the example above, the properties are forward annotated at the top level in the netlist.

```
(*syn_useioff = 1/0*) module test_top(... )
```

alsloc

Attribute

Preserves relative placements of macros and IP blocks in the Microchip Designer place-and-route tool.

Vendor	Technology
Microchip	Legacy Antifuse families: ACT1, ACT2, ACT3, 3200DX, 40MX, 42MX, 54SX, 54SXA, Axcelerator, and eX

Description

Preserves relative placements of macros and IP blocks in the Microchip Designer place-and-route tool. The alsloc attribute has no effect on synthesis, but is passed directly to Microchip Designer.

The alsloc constraint is passed directly to the post synthesis EDN netlist as the following:

```
(property alsloc (string "R15C6"))
(property alsloc (string "R35C6"))
```

alsloc Syntax Specification

Name	Global	Object	Synthesis Tool
Alsloc	No	Macro or IP block	Synplify, Synplify Pro Synplify Premier

alsloc Value

Value	Default	Description
location	None	Location of macro or IP block.

This table summarizes the syntax in different files:

FDC	define_attribute {object} alsloc {location}	SCOPE Example
Verilog	object /* synthesis alsloc = "location" */;	Verilog Example
VHDL	attribute alsloc of object : label is "location";	VHDL Example

SCOPE Example

Following is an example of setting alsloc on a macro (u1).

```
define_attribute {u1} alsloc {R15C6}
```

Verilog Example

```
module test(in1, in2, in3, clk, q);
    input in1, in2, in3, clk;
    output q;
    wire out1 /* synthesis syn_keep = 1 */, out2;
    and2a u1 (.A (in1), .B (in2), .Y (out1))
        /* synthesis alsloc="R15C6" */;
    assign out2 = out1 & in3;
    df1 u2 (.D (out2), .CLK (clk), .Q (q))
        /* synthesis alsloc="R35C6" */;
endmodule

module and2a(A, B, Y); // synthesis syn_black_box
    input A, B;
    output Y;
endmodule

module df1(D, CLK, Q); // synthesis syn_black_box
    input D, CLK;
    output Q;
endmodule
```

VHDL Example

```
library IEEE;
use IEEE.std_logic_1164.all;

entity test is
port (in1, in2, in3, clk : in std_logic;
      q : out std_logic);
end test;

architecture rtl of test is
signal out1, out2 : std_logic;

component AND2A
port (A, B : in std_logic;
      Y : out std_logic);
end component;

component df1
port (D, CLK : in std_logic;
      Q : out std_logic);
end component;

attribute syn_keep : boolean;
attribute syn_keep of out1 : signal is true;
attribute alsloc: string;
attribute alsloc of U1: label is "R15C6";
attribute alsloc of U2: label is "R35C6";
attribute syn_black_box : boolean;
attribute syn_black_box of AND2A, df1 : component is true;
begin
U1: AND2A port map (A => in1, B => in2, Y => out1);
out2 <= in3 and out1;
U2: df1 port map (D => out2, CLK => clk, Q => q);
end rtl;
```


alspin

Attribute

Assigns the scalar or bus ports of the design to Microchip I/O pin numbers.

Vendor	Technology
Microchip	Legacy Antifuse families: ACT1, ACT2, ACT3, 3200DX, 40MX, 42MX, 54SX, 54SXA, Axcelerator, and eX

Description

The alspin attribute assigns the scalar or bus ports of the design to Microchip I/O pin numbers (pad locations). Refer to the Microchip databook for valid pin numbers. If you use alspin for bus ports or for slices of bus ports, you must also use the [syn_noarrayports](#) attribute. See [Specifying Locations for Microchip Bus Ports, on page 822](#) of the Reference for information on assigning pin numbers to buses and slices.

The alspin pin location is passed as a property string to the output EDN netlist as the following:

```
(instance (rename dataoutZ0 "dataout") (viewRef netlist (cellRef df1 (libraryRef &54SXA)))
(property alspin (string "48")))
```

alspin Syntax Specification

Name	Global	Object	Synthesis Tool
alspin	No		Synplify, Synplify Pro Synplify Premier

alspin Value

Value	Default	Description
<i>pin_number</i>	None	The Microchip I/O pin

This table summarizes the syntax in different files:

FDC	define_attribute { <i>port_name</i> } alspin { <i>pin_number</i> }	Constraint File Example
Verilog	<i>object</i> /* synthesis alspin = " <i>pin_number</i> " */;	Verilog Example
VHDL	attribute alspin of <i>object</i> : <i>objectType</i> is " <i>pin_number</i> ";	VHDL Example

Constraint File Example

In the attribute syntax, *port_name* is the name of the port and *pin_number* is the Microchip I/O pin.

```
define_attribute {DATAOUT} alspin {48}
```

Verilog Example

Where *object* is the port and *pin_number* is the Microchip I/O pin. For example:

```
module comparator (datain, clk, dataout);
  output reg dataout /* synthesis alspin="48" */;
  input [7:0] datain;
  input clk;

  always@ (posedge clk)
    begin
      dataout <= datain;
    end
endmodule
```

VHDL Example

See [VHDL Attribute and Directive Syntax, on page 401](#) for different ways to specify VHDL attributes and directives.

Where *object* is the port, *objectType* is signal, and *pin_number* is the Microchip I/O pin. For example:

```
library ieee;
use ieee.std_logic_1164.all;
entity comparator is
    port (datain : in std_logic_vector(7 downto 0);
          clk : in std_logic;
          dataout : out std_logic_vector(7 downto 0));
attribute alspin : string;
attribute alspin of dataout : signal is "48";
end;

architecture rtl of comparator is
begin

process(clk)
begin
    if clk'event and clk = '1' then
        dataout <= datain;
    end if;
end process;

end rtl;
```


alspreserve

Attribute

Specifies a net that you do not want removed by the Microchip Designer place-and-route tool.

Vendor	Technology
Microchip	Legacy Antifuse families: ACT1, ACT2, ACT3, 3200DX, 40MX, 42MX, 54SX, 54SXA, Axcelerator, and eX

Description

The alspreserve attribute specifies a net that you do not want removed (optimized away) by the Microchip Designer place-and-route tool. The alspreserve attribute has no effect on synthesis, but is passed directly to the Microchip Designer place-and-route software. However, to prevent the net from being removed during the synthesis process, you must also use the [syn_keep](#) directive.

The alspreserve attribute is passed to the output EDN netlist file as the following:

```
(net (rename and_outZ0Z3 "and_out3") (joined
    (portRef b (instanceRef outZ0Z1))
    (portRef y (instanceRef and_out3_1)))
)
(property alspreserve (integer 1)))
```

alspreserve Syntax Specification

Name	Global	Object	Synthesis Tool
alspreserve	No	Net	Synplify, Synplify Pro Synplify Premier

alspreserve Value

Value	Default	Description
<i>object</i>	None	Name of the net to preserve

This table summarizes the syntax in different files:

FDC	define_attribute {n: <i>net_name</i> } alspreserve {1}	Constraint File Example
Verilog	<i>object</i> /* synthesis alspreserve = 1 */;	Verilog Example
VHDL	attribute alspreserve of <i>object</i> : signal is true;	VHDL Example

Constraint File Example

```
define_attribute {n:and_out3} alspreserve {1};
define_attribute {n:or_out1} alspreserve {1};
```

Verilog Example

```
module complex (in1, out1);
  input [6:1] in1;
  output out1;
  wire out1;
  wire or_out1 /* synthesis syn_keep=1 alspreserve=1 */;
  wire and_out1;
  wire and_out2;
  wire and_out3 /* synthesis syn_keep=1 alspreserve=1 */;
  assign and_out1 = in1[1] & in1[2];
  assign and_out2 = in1[3] & in1[4];
  assign and_out3 = in1[5] & in1[6];
  assign or_out1 = and_out1 | and_out2;
  assign out1 = or_out1 & and_out3;
endmodule
```

VHDL Example

See [VHDL Attribute and Directive Syntax](#), on page 401 for different ways to specify VHDL attributes and directives.

```

library ieee;
use ieee.std_logic_1164.all;
library synplify;
use synplify.attributes.all;

entity complex is
port (input : in std_logic_vector (6 downto 1);
      output : out std_logic);
end complex;

architecture RTL of complex is
signal and_out1 : std_logic;
signal and_out2 : std_logic;
signal and_out3 : std_logic;
signal or_out1 : std_logic;
attribute syn_keep of and_out3 : signal is true;
attribute syn_keep of or_out1 : signal is true;
attribute alspreserve of and_out3 : signal is true;
attribute alspreserve of or_out1 : signal is true;

begin
    and_out1 <= input(1) and input(2);
    and_out2 <= input(3) and input(4);
    and_out3 <= input(5) and input(6);
    or_out1 <= and_out1 or and_out2;
    output <= or_out1 and and_out3;
end;

```


altera_io_powerup

Attribute

Controls the power-up mode of registers in the I/O port.

Vendor	Technology
Intel FPGA	APEX20KE, APEX II, Stratix

Description

Controls the power-up mode of registers in the I/O port (bit or bus). However, you can only specify this attribute if the I/O register does not have a defined preset or clear condition.

If the I/O register has a preset, powerup is set to high. If the I/O register has a clear, powerup is set to low. You can only specify the altera_io_powerup attribute to control the powerup mode if the I/O register is not defined.

By default, Quartus II sets the power-up mode of the I/O to low.

If the register cannot be packed into an I/O, the synthesis tool issues a warning saying it has ignored the attribute.

altera_io_powerup Syntax Specification

Name	Global	Object	Synthesis Tool
<i>altera_io_powerup</i>	No	Port, bus,* bus slice*	Synplify, Synplify Pro Synplify Premier

* Bus and bus slice in constraint file only, not HDL

This table summarizes the syntax in different files:

FDC	<code>define_attribute {port} altera_io_powerup {high low}</code>	Constraint File Example
Verilog	<code>object /*synthesis altera_io_powerup = "high low" */;</code>	Verilog Example
VHDL	<code>attribute altera_io_powerup of object : objectType is "high low";</code>	VHDL Example

Constraint File Example

You can apply the `altera_io_powerup` attribute to a port, bus, or slice of a bus in the SCOPE spreadsheet, whereas in the source code this attribute can only be applied to a port definition.

1. In the SCOPE spreadsheet, click the port in the Object column.
2. In the Attribute column pulldown menu, choose `altera_io_powerup`.
3. Enter the appropriate string in the Value column: `high` or `low`.

Following is an example:

```
define_attribute {seg [31:0]} altera_io_powerup {high}
```

Verilog Example

The following example demonstrates how to apply the `altera_io_powerup` attribute to a single-bit port in Verilog.

```
moduledff_or(q, a, b, clk);
output q /* synthesis altera_io_powerup="low" */;
input a, b, clk;
reg q;

always@ (posedge clk)
begin
q= a + b;
end
endmodule
```

VHDL Example

The following demonstrates how to apply the altera_io_powerup attribute to a 32-bit bus port in VHDL.

```
library ieee;
use ieee.std_logic_1164.all;
entity dff_or is
port (
    q : out std_logic;
    a : in std_logic;
    b : in std_logic;
    clk: in std_logic
);

attribute altera_io_powerup : string;
attribute altera_io_powerup of q : signal is "high";
end ;

architecture rtl of dff_or is
begin
    process(clk)
    begin
        if clk'event and clk ='1' then
            q<= a or b;
        end if;
    end process;
end rtl;
```

See [VHDL Attribute and Directive Syntax, on page 401](#) for different ways to specify VHDL attributes and directives.

altera_logiclock_location

Attribute

Assigns a compile point to a Logic Lock Region.

Vendor	Technology
Intel FPGA	APEX, APEX II, Stratix, Cyclone

Description

This attribute is used by Synplify Pro to assign a compile point to a Logic Lock Region, specifying the location. In the Synplify Premier tool, this assignment is done during design planning.

When applied to a compile point, this attribute assigns that compile point to a Logic Lock Region, specifying the location. The synthesis tool provides a separate VQM file (VQM) for the compile point to Quartus II. During fitting, Quartus II places all logic assigned to a Logic Lock Region within the region.

In the Synplify Premier tool, physical assignments are made in the Synplify Premier Design Planner view, not by assigning attributes. The Synplify Premier tool ignores applications of this attribute in constraint files; it does not forward-annotate them to the Quartus II place-and-route tool.

altera_logiclock_location Syntax Specification

Name	Object	Default	Synthesis Tool
<code>altera_logicclock_location</code>	<code>module</code>	<code>floating</code>	Synplify, Synplify Pro

Specifying a Compile Point Region Using the Synplify Premier Tool

Instead of applying this attribute, to specify a compile-point region as a Logic Lock Region you use the Synplify Premier Design Planner view popup menu command **LogicLock -> Enabled** - see [Design Planner Popup Menus, on page 673](#). The Synplify Premier tool then forward-annotates the region location and size in a **TCL** file for treatment by the Quartus II place-and-route tool as a Logic Lock Region.

A physical region can be designated as a Logic Lock Region only if you have defined the region as a compile point. For more information on compile points and the compile point synthesis flow, see and [Synthesizing Compile Points, on page 626](#) of the *User Guide*.

When you specify a compile point as a Logic Lock Region, the Synplify Premier tool provides a separate VQM file (**VQM**) for the compile point to Quartus II. During fitting, Quartus II places all logic assigned to a Logic Lock Region within the region.

Applying `altera_logiclock_location` to a Compile Point

Available only in the Synplify Pro tool, when you assign this attribute to a compile point, you typically also assign the attribute `altera_logiclock_size` to it - see [altera_logicclock_size, on page 51](#). If you assign only one of these two attributes, a default value is used for the unassigned attribute.

The attribute value `floating` means that Quartus II determines the actual LogicLock region location during its optimization process. The default value is `floating`.

This table summarizes the syntax in different files:

FDC	<code>define_attribute {v:module architectureName} altera_logicclock_location {floating}</code>	Constraint File Example
-----	---	---

Constraint File Example

In the attribute syntax, *module* or *architectureName* is the name of a Verilog module or VHDL architecture that has been defined as a compile point.

Following is an example of specifying a floating LogicLock region location:

```
define_attribute {v.work.a1} altera_logicclock_location {floating}
```

The `altera_logicclock_size` attribute is applied in a constraint file, it cannot be used in source code.

altera_logiclock_size

Attribute

Assigns a compile point to a Logic Lock Region, specifying the size.

Vendor	Technology
Intel FPGA	Stratix, Cyclone

Description

This attribute is used by Synplify Pro to assign a compile point to a Logic Lock Region, specifying the size. In the Synplify Premier tool, this assignment is done during design planning instead of with this attribute.

This attribute is applied to a compile point. The attribute assigns the compile point to a Logic Lock Region, specifying the size. The synthesis tool provides a separate VQM file (VQM) for the compile point to Quartus II. During fitting, Quartus II places all logic assigned to a Logic Lock Region within the region.

The Synplify Premier tool ignores applications of this attribute in constraint files; it does not forward-annotate them to the Quartus II place-and-route tool.

altera_logiclock_size Syntax Specification

Name	Object	Default	Synthesis Tool
<i>altera_logicclock_size</i>	<i>module</i>	<i>auto</i>	Synplify Synplify Pro

Specifying a Compile Point Region Using the Synplify Premier Tool

To specify a compile-point region as a Logic Lock Region you use the Synplify Premier Design Planner view popup menu command LogicLock -> Enabled - see [Design Planner Popup Menus, on page 673](#). The Synplify Premier tool then forward-annotates the region location and size in a `TCL` file for treatment by the Quartus II place-and-route tool as a Logic Lock Region.

You can only designate a physical region as a Logic Lock Region if you have defined the region as a compile point. For more information on compile points and the compile point synthesis flow, see [Synthesizing Compile Points, on page 626](#) of the *User Guide*.

When you specify a compile point as a Logic Lock Region, the synthesis tool provides a separate `VQM` file (`VQM`) for the compile point to Quartus II. During fitting, Quartus II places all logic assigned to a Logic Lock Region within the region.

Applying `altera_logiclock_size` to a Compile Point

When you assign this attribute to a compile point, you typically also assign the attribute `altera_logicclock_location` to it - see [altera_logicclock_location, on page 47](#). If you assign only one of these two attributes, a default value is used for the unassigned attribute.

The attribute value `auto` means that Quartus II determines the actual LogicLock region size during its optimization process. The default value is `auto`. If the value is a comma-separated pair of numbers, it explicitly specifies the size of the region.

FDC `define_attribute {v:module|architectureName}
altera_logicclock_size {auto|width,height}`

[Constraint File Example](#)

Constraint File Example

In the attribute syntax:

- `module` or `architectureName` is the name of a Verilog module or VHDL architecture that has been defined as a compile point
- `width` and `height` are the width and height, respectively, of the Logic Lock Region

Following is an example of specifying an explicit LogicLock region size:

```
define_attribute {v.work.a1} altera_logicclock_size {30,20}
```

The `altera_logicclock_size` attribute is applied in a constraint file, it cannot be used in source code.

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 149](#) 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	<code>object /* synthesis black_box_pad_pin = portList */;</code>	Verilog Example
VHDL	<code>attribute black_box_pad_pin of object: objectType is portList;</code>	VHDL Example

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)";

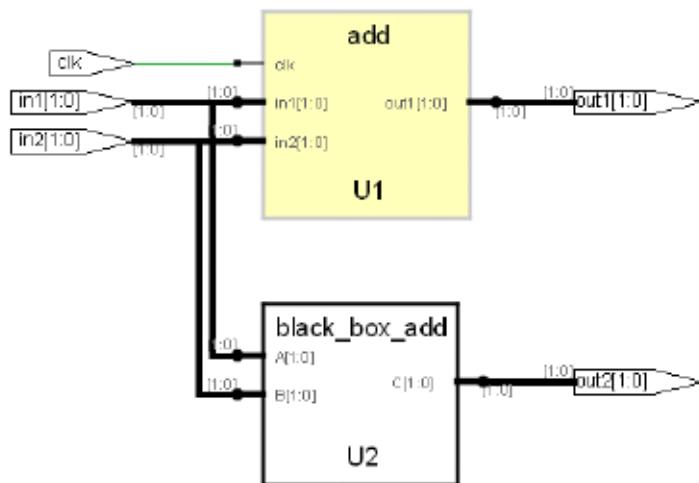
begin
    test123 : test generic map (width) port map (in1,in2,clk,q);
end top1_arch;
```

Effect of Using `black_box_pad_pin`

The following example shows the effect of applying the attribute.

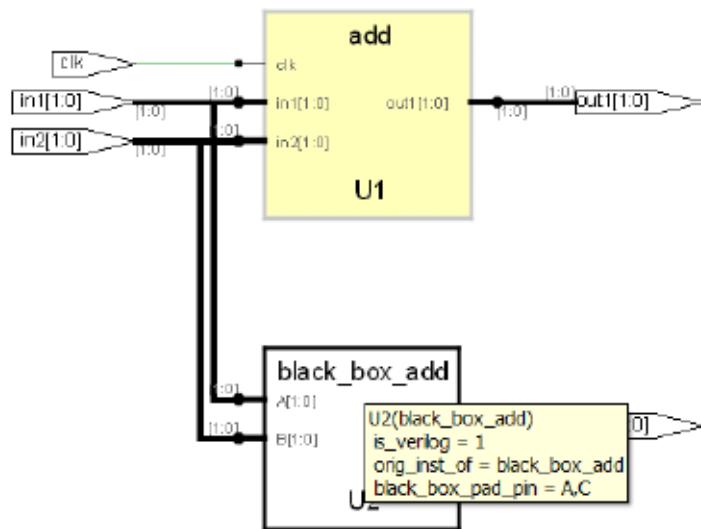
Before using `black_box_pad_pin`

```
)  
(cell black_box_add (cellType GENERIC)  
  (view verilog (viewType NELIST)  
    (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 149](#) 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>	Verilog Example
VHDL	<code>attribute black_box_tri_pins of object: objectType is portList;</code>	VHDL Example

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 BBDSLHS(D,E,GIN,GOUT,PAD,Q)
/* synthesis syn_black_box black_box_tri_pins="PAD" */;
```

Here is an example with a list of multiple pins:

```
module bb1(D,E,tril,tri2,tri3,Q)
/* synthesis syn_black_box black_box_tri_pins="tril,tri2,tri3" */;
```

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

```
module bb1(D,bus1,E,GIN,GOUT,Q)
/* synthesis syn_black_box black_box_tri_pins="bus1[7:0]" */;
```

VHDL Example

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

package my_components is
component BBDSLHS
    port (D: in std_logic;
          E: in std_logic;
          GIN : in std_logic;
          GOUT : in std_logic;
          PAD : inout std_logic;
          Q: out std_logic);
```

```
end component;

attribute syn_black_box : boolean;
attribute syn_black_box of 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 bb1 : 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 bb1 : component is "bus1[7:0]";
```


CLOCK_DEDICATED_ROUTE

Attribute

Synplify Pro, Synplify Premier

Directs the place-and-route tool to not follow clock placement rules.

Vendor	Devices
Xilinx	Spartan-3, Spartan-3E, Spartan-3A, Spartan-3A DSP, Spartan-6, Virtex-4 and newer devices

CLOCK_DEDICATED_ROUTE Values

Default	Global	Object
TRUE	No	<ul style="list-style-type: none"> • Net of an instance that is the driver • I/O pin that is the driver • Input or output pins of the following primitives: BUFG, BUFR, DCM, PLLPMCD, GT11, GT11 DUAL, GT11 CLK, or GTP DUAL

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

TRUE (Default)	Enforces clock placement rules.
FALSE	Lets you override clock placement rules. Use when a clock component (i.e. BUFG, MMCM, or PLL) is placed such that the dedicated fast clock route cannot be used.
BACKBONE	Lets you use this value when location constraints are assigned that violate basic clock placement rules (but is not generally recommended) or when an MMCM or PLL is placed far from the source CCIP pin. The extra wire length may add delay to the timing path from the CCIP pin to the MMCM or PLL. Use BACKBONE if the design meets timing with the added delay.

Description

The default is TRUE, and the tool follows clock placement rules in this mode. If clock components are not optimally placed, you can get errors.

To ignore the rules and violations, set this attribute to FALSE. In this mode, the tool continues with placement & routing, even if there are errors. Use this setting only when absolutely necessary. It is recommended that you fix all clock placement errors before proceeding.

CLOCK_DEDICATED_ROUTE Syntax

FDC	define_attribute {object_name}	SCOPE Example
	CLOCK_DEDICATED_ROUTE {TRUE FALSE BACKBONE}	

SCOPE Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	n:clk	CLOCK_DEDICATED_ROUTE	FALSE		

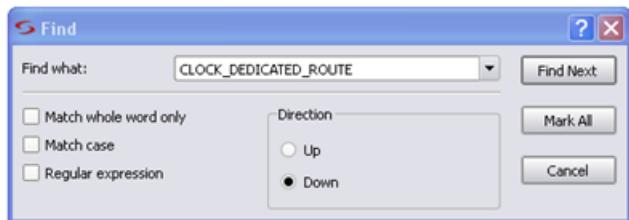
```
define_attribute {n:clk} CLOCK_DEDICATED_ROUTE FALSE
```

Effect of Using CLOCK_DEDICATED_ROUTE

The following example shows the effect of applying the attribute.

The constraints are forward-annotated to the place-and-route stage. Check for the CLOCK_DEDICATED_ROUTE attribute in the edif file.

```
(net (rename q_c_8 "q_c[8]") (joined
  (portRef Q (instanceRef q202_8))
  (portRef I (instanceRef q_obuf_8)))
())
(net (rename q_8 "q[8]") (joined
  (portRef O (instanceRef q_obuf_8))
  (portRef (member q 1)))
())
(net (rename q_c_9 "q_c[9]") (joined
  (portRef Q (instanceRef q202_9))
  (portRef I (instanceRef q_obuf_9)))
())
(net (rename q_9 "q[9]") (joined
  (portRef O (instanceRef q_obuf_9))
  (portRef (member q 0)))
())
(net clk_c (joined
  (portRef I (instanceRef clk_keep_cb))
  (portRef O (instanceRef clk_ibuf)))
)
(property CLOCK_DEDICATED_ROUTE (integer 0))
)
(net (rename cnt_cry202_1 "cnt_cry[1]") (joined
  (portRef L0 (instanceRef cnt_cry_1))
  (portRef CI (instanceRef cnt_cry_2))
  (portRef CI (instanceRef cnt_s_2)))
())
(net (rename cnt_s202_1 "cnt_s[1]") (joined
  (portRef O (instanceRef cnt_s_1))
  (portRef D (instanceRef cnt_1)))
())
```



diff_term

Attribute

Specifies Differential Termination for true differential input I/O standards.

Vendor	Technology
Xilinx	Virtex-II Pro and Spartan-3 and newer devices

diff_term Values

Value	Description
true	Enables differential termination on the pin.
false	Disables differential termination on the pin.

Description

Use this attribute to specify Differential Termination for true differential input I/O standards. You can use this attribute with inputs for the buffer, such as IBUFDS and OBUFDS. Set `dff_term` in the constraint file or HDL source code.

diff_term Syntax

Object

IBUFDS, IBUFGDS, OBUFDS, OBUFGDS, and IOBUFGDS.

The following table summarizes the syntax in different files:

FDC	define_attribute {input} diff_term {true false}	FDC Example
Verilog	object /* synthesis diff_term=true false */	Verilog Example
VHDL	attribute diff_term of object : objectType is true false;	VHDL Example

FDC Example

For example:

```
define_attribute {p:in1_p} {diff_term} {true}
define_attribute {p:in1_n} {diff_term} {true}
```

Verilog Example

```
module syn_diff_io (in1_p, in1_n, d, out1);
    input in1_p /*synthesis diff_term = true*/,
          in1_n /*synthesis diff_term = true*/,
          d;
    output reg out1;

    reg in1;

    always@(in1_p, in1_n)
        if (in1_p != in1_n)
            in1 = in1_p;
        else
            in1 = in1;

    always@(posedge in1)
        out1 <= d;

endmodule
```

VHDL Example

```
library IEEE;
use IEEE

E.std_logic_1164.all;

entity test is
    port  in1_p : in std_logic;
          in1_n : in std_logic;
          d     : in std_logic;
          out1  : out std_logic
    );
attribute diff_term: boolean;
attribute diff_term of in1_p,in1_n: signal is true;
end test;

architecture arc of test is
signal in1 : std_logic;

begin

process(in1_p,in1_n)
begin
    if(in1_p /= in1_n)then
        in1 <= in1_p;
    else
        in1 <= in1;
    end if;
end process;

process(in1)
begin
    if (in1'event and in1='1') then
        out1 <= d;
    end if;
end process;

end arc;
```


full_case

Directive

For Verilog designs only. Indicates that all possible values have been given, and that no additional hardware is 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:

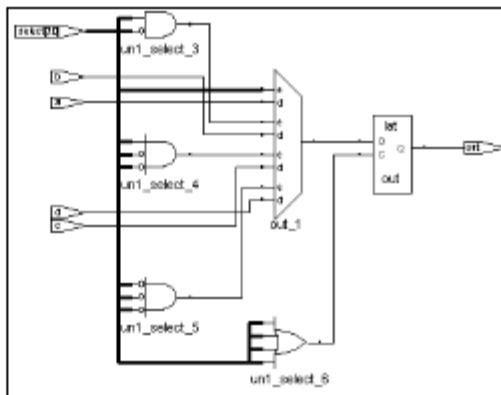
Verilog	<i>object /* synthesis full_case */;</i>	Verilog Examples
---------	--	----------------------------------

Verilog Examples

The following casez statement creates a 4-input multiplexer with a pre-decoded select bus (a decoded select bus has exactly one bit enabled at a time):

```
module muxnew1 (out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;

always @ (select or a or b
or c or d)
begin
  casex (select)
    4'b???1: out =
    a;
    4'b??1?: out = b;
    4'b?1???: out = c;
    4'b1????: out = d;
  endcase
end
```



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:

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

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

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 , on page 284 in the *Command Reference*.

Note: VHDL applications use the `syn_looplmt` directive (see [syn_looplmt, on page 373](#)).

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
            begin
                dout[i] <= din[i];
            end
        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.

orca_padtype

Attribute

The `orca_padtype` attribute specifies the pad type for a Lattice ORCA I/O.

Vendor	Technologies
Lattice	ORCA FPSC, Series 2, Series 3, and Series 4

orca_padtype Values

Value	Description	Default	Global
<code>portName</code>	Specifies the name of the port for which the pad type applies.	None	No
<code>padType</code>	Specifies the pad type values to be used for the port. For example: <ul style="list-style-type: none"> • Inputs: IBM • Outputs: OB6 		

Description

The `orca_padtype` attribute specifies the pad type for a Lattice ORCA I/O.

Values for `padType` are defined in the Lattice FPGA databook. The default for inputs is IBM, which is a CMOS pad type. The default for outputs is OB6.

orca_padtype Syntax

FDC	define_attribute {portName} orca_padtype {padType}	FDC Example
Verilog	object /* orca_padtype = "padType" */;	Verilog Example
VHDL	attribute orca_padtype : string; attribute orca_padtype of object : objectType is "padType";	VHDL Example

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description	Comment
1	<input checked="" type="checkbox"/>	<any>	p:ain[3]	orca_padtype	IBT	string	Pad type for I/O	
2								

This example defines bit 3 of AIN as an ORCA TTL input pad.

```
define_attribute {AIN[3]} orca_padtype {IBT}
```

Verilog Example

```
object /* synthesis orca_padtype = "padType" */;
```

This is an example of the orca_padtype attribute in a Verilog source file:

```
module counter4 (cout, out, in, ce, load, clk, rst);

// Choose IBT padtype to select a TTL input pad for "load"
input load /* synthesis orca_padtype="IBT" */;
endmodule
```

VHDL Example

```
attribute orca_padtype of object : objectType is "padType";
```

This is an example of the orca_padtype attribute in a VHDL source file:

```
entity counter4 is
  port (cout: out bit;
        output_vector: out bit_vector (3 downto 0);
        input_vector: in bit_vector (3 downto 0);
        ce, load, clk, rst: in bit);
  -- Choose OB12 padtype for "output_vector".
```

```
-- The OB12 padtype is being set for all four
-- bits in "output_vector".
attribute orca_padtype : string;
attribute orca_padtype of output_vector: signal is "OB12";
end counter4;
```

Effect of Using orca_padtype

The synthesis software overrides the default I/O cell and uses the I/O cell specified for the `orca_padtype` attribute instead.

orca_props

Attribute

Specifies properties for forward-annotation. Add the attribute to ports or I/O pads.

Vendor	Technologies
Lattice	ORCA FPSC, Series 2, Series 3, and Series 4

orca_props Values

Value	Description	Default	Global
<i>portName</i>	Specifies the name of ports or pads for which the property values apply.	None	No
<i>property=value</i>	Specifies the property values to be used for the specified ports or pads.		

Description

Specifies properties for forward-annotation. Add the attribute to ports or I/O pads. For more information about the properties and their values, see [I/O Properties and Values, on page 88](#).

orca_props Syntax

FDC	<code>define_attribute {p:<i>portName</i>} orca_props {<i>property=value</i>}</code>	FDC Example
Verilog	<code>object /* orca_props = "<i>property=value</i>" */;</code>	Verilog Example
VHDL	<code>attribute orca_props : string; attribute orca_props of <i>object</i> : <i>objectType</i> is "<i>property=value</i>";</code>	VHDL Example

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	port	p:data_in	orca_props	levelmode=lvds	string	Forward annotate attributes to ORCA back-end
2							

The following is an example of the syntax:

```
define_attribute {p:data_in} orca_props {LEVELMODE=LVDS}
```

I/O Properties and Values

The following table lists the properties and values currently supported.

Property	Valid Values	Attached to	Notes
AMPSMODE	6 12 24	Bidirectional or output ports	Must be set in conjunction with levelmode set to LVTTL or LVCMOS2.
BUFMODE	SLOW FAST	Bidirectional or output ports	Must be set in conjunction with levelmode set to LVTTL or LVCMOS2.
CLKMODE	SCLK ECLK	Bidirectional or output ports, and I/O pads	
DELAYMODE	0 1	Input ports	
KEEPERMODE	off on	Bidirectional or tristate ports	
LEVELMODE	LVTTL LVCMOS2 LVCMOS18 PCI SSTL2 SSTL3 HSTL1 HSTL3 GTL GTLPLUS PECL LVPECL LVDS	Any port	
RESISTOR	off on	Input or output ports	Must be set in conjunction with levelmode set to LVDS or LVPECL.

Verilog Example

```
object /* synthesis orca_props = "property=value" */;
```

where *object* can be a port (input, output, bidirectional, or tristate) or an I/O pad. For more information about the properties and their values, see [I/O Properties and Values, on page 88](#). The following are examples of the syntax:

```
output data_out /* synthesis orca_props = "LEVELMODE=LVDS" */;
output clk_out_ddr /* synthesis orca_props = "CLKMODE = SCLK" */;
```

If multiple properties are required, separate them with commas. This is an example of an object with more than one property:

```
output data_out
/* synthesis orca_props = "BUFMODE=FAST,LEVELMODE=LVTTL" */;
```

VHDL Example

```
attribute orca_props : string;
attribute orca_props of object : objectType is "property=value";
```

For more information about the properties and their values, see [I/O Properties and Values, on page 88](#). The following is an example of the VHDL syntax.

```
Entity test is port (a : in std_logic; b : out std_logic);
attribute orca_props : string;
attribute orca_props of a :signal is "LEVELMODE=LVDS";
```

This is an example of the syntax when you have multiple properties:

```
Entity test is port (a : in std_logic; b : out std_logic);
attribute orca_props : string;
attribute orca_props of a :signal is "LEVELMODE=LVTTL,
AMPSMODE=6";
```

Effect of Using orca_props

The synthesis tool forward annotates the properties specified for the *orca_props* attribute to the EDIF file.

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 */

[Verilog Example](#)

Verilog Example

You specify the directive as a comment immediately following the **select** value of the **case** statement.

```
module muxnew4 (out, a, b, c, d, select);
    output out;
    input a, b, c, d;
    input [3:0] select;
    reg out;

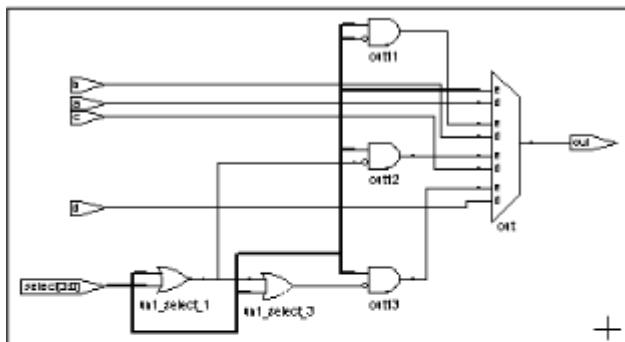
    always @ (select or a or b or c or d)

    begin
        casez (select) /* synthesis parallel_case */
            4'b???1: out = a;
            4'b??1?: out = b;
            4'b?1??: out = c;
            4'b1???: out = d;
            default: out = 'bx;
        endcase
    end
endmodule
```

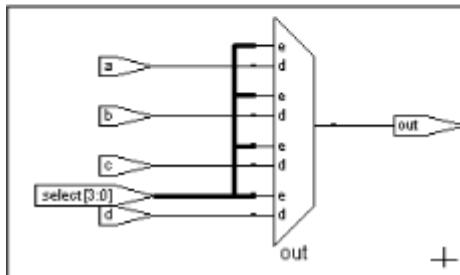
If the **select** bus is decoded within the same module as the **case** statement, the parallelism of the tag matching is determined automatically, and the **parallel_case** directive is unnecessary.

Effect of Using parallel_case

Extra logic for priority encoding (without parallel_case)



Extra logic eliminated with 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 749](#). 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>	Verilog Example
VHDL	<pre>--pragma translate_off --pragma translate_on --synthesis translate_off --synthesis translate_on</pre>	VHDL Example

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;

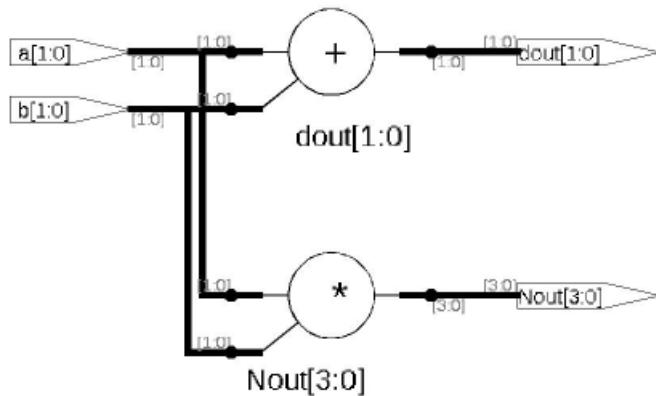
architecture rtl of test is
begin
    dout <= a + b;

    --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:

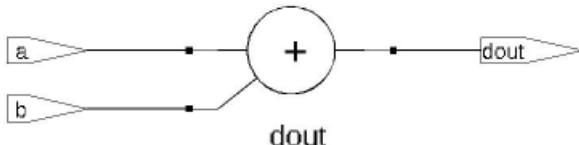
    --pragma translate_off
    Nout <= a * b;
    --pragma translate_on
end;
```

Effect of Using pragma translate_off/pragma translate_on

Before applying the attribute:



After applying the attribute:



syn_allow_retimining

Attribute

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

Vendor	Technology	Synthesis Tool
Achronix	Speedster7t	Synplify Pro
Intel FPGA	Stratix and Cyclone families	Synplify Pro, Synplify Premier
Lattice	Lattice EC,ECP, iCE40, iCE5LP, LIFMD, SC, SCM, XP and later families	Synplify Pro
Microchip	Axcelerator, Fusion, IGLOO, PolarFire, ProASIC, ProASICPLUS, ProASIC3, RTG4, SmartFusion and later families	Synplify Pro
Xilinx	All Virtex technologies Spartan-II/IIE, Spartan-3/3E, Spartan-6	Synplify Pro, Synplify Premier

syn_allow_retimining values

1 | true Allows registers to be moved during retiming.

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

Description

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

The attribute can be applied either globally 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_retimining` attribute to disable retiming for specific objects that you do not want moved. Do not use the `syn_allow_retimining` attribute with the fast synthesis mode.

syn_allow_retimining Syntax

Global Object

Yes	Register
-----	----------

You can specify the attribute in the following files:

FDC `define_attribute {register} syn_allow_retimining {1|0}`
`define_global_attribute syn_allow_retimining {1|0}`

[FDC Example](#)

Verilog `object /* synthesis syn_allow_retimining = 0 | 1 */;`

[Verilog Example](#)

VHDL `attribute syn_allow_retimining of object : objectType is true | false;`

[VHDL Example](#)

FDC Example

```
define_attribute {register} syn_allow_retimining {1|0}
```

```
define_global_attribute syn_allow_retimining {1|0}
```

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

Verilog Example

object /* synthesis syn_allow_retimining = 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_retimining=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_retimining 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_retimining : boolean;
    attribute syn_allow_retimining 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 401](#) for different ways to specify VHDL attributes and directives.

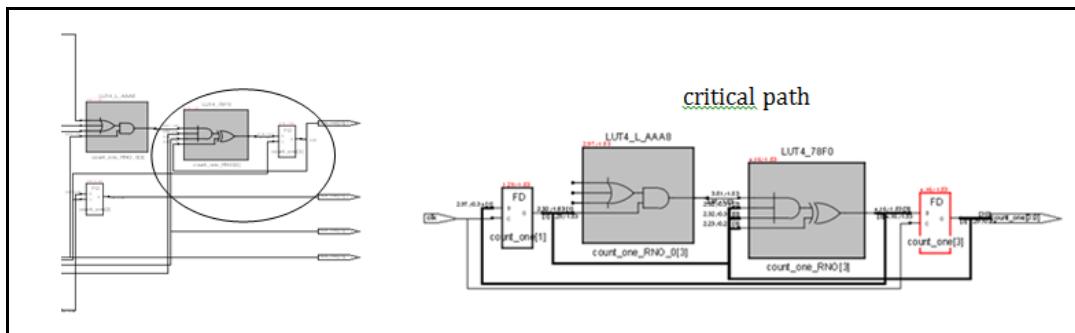
Effect of using syn_allow_retiming

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.

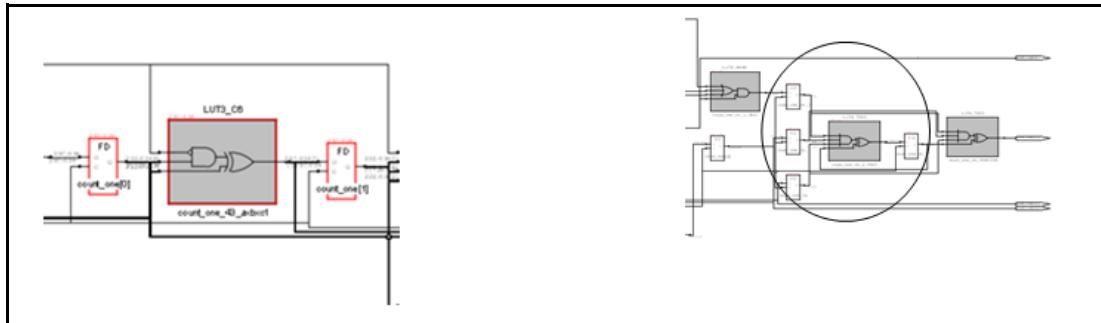


After applying syn_allow_retiming.

Verilog output reg [3:0]count_one /* synthesis syn_allow_retiming=1 */;

VHDL attribute syn_allow_retimimg 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
Intel FPGA	All Arria, Cyclone, Stratix, Stratix Hardcopy Companion, and newer families
Lattice	ECP3 and later families
Xilinx	All Xilinx families

Intel FPGA	All Arria, Cyclone, Stratix, Stratix Hardcopy Companion, and newer families
Lattice	ECP3 and later families
Xilinx	All Xilinx families

syn_allowed_resources Values

Vendor	Default	Global	Object
Intel FPGA	Multipliers RAM	Yes	Module Compile points Verilog: register signals VHDL: signals

Vendor	Default	Global	Object
Lattice	blockmults	Yes	Module Compile points Verilog: register signals VHDL: signals

Vendor	Default	Global	Object
Xilinx	Multipliers: dssps RAM: blockrams	Yes	Module Compile points Verilog: register signals VHDL: signals

Description

The `syn_allowed_resources` attribute allows you to specify the maximum number of available resource 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 626](#) of the *User Guide*.

syn_allowed_resources Syntax

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

- *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
blockram	Intel FPGA devices	block RAMs
M4K	Intel Stratix, Stratix GX, Stratix II/Hardcopy II, Stratix II GX, Cyclone, Cyclone II	M4K RAM blocks (128 x 36 bits)
M512	Intel Stratix, Stratix GX, Stratix II/Hardcopy II, Stratix II GX, Arria GX, Arria II GX	M512 RAM blocks (32 x 18 bits)
M9K	Intel Stratix III, Stratix IV, Stratix IV GT, Cyclone III	M9K RAM blocks (256 x 36 bits)
M144K	Intel Stratix III, Stratix IV, Stratix IV GT	M144K RAM blocks (4K x 36 bits)

Values	Devices	Resource
M-RAM	Intel Stratix, Stratix GX, Stratix II/Hardcopy II, Stratix II GX, Arria GX, Arria II GX	Mega RAM blocks (4K x 144 bits)
M20K	Intel Arria GZ and Stratix V, Stratix 10, Agilex	M20K RAM blocks
M10K	Intel Arria V and Cyclone V	M10K RAM blocks
dsp_blocks	Intel all Stratix and Arria families	Digital signal processing blocks

Values	Devices	Resource
blockmults	Lattice ECP3	Multiplier blocks

Values	Devices	Resource
blockram	Xilinx devices	block RAMs
dsps	Xilinx families with DSP and DSP48 blocks	Digital signal processing blocks. For newer technologies, from Virtex 4 on, use this value instead of blockmults.
blockmults	Xilinx families with DSP48 blocks	Block mults for DSP48 blocks. This is for backwards compatibility; use dsps.

If you make multiple assignments with different types of resources, separate them with commas. See [FDC Example, on page 108](#).

FDC Example

The following example is for Intel FPGA.

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	view	v:work.top	syn_allowed_resources	dsp_blocks=4	string	Control resource usage in a compile point
1	<input checked="" type="checkbox"/>	view	v:work.top	syn_allowed_resources	MLAB=3	string	Control resource usage in a compile point
1	<input checked="" type="checkbox"/>	view	v:work.top	syn_allowed_resources	M-RAM=3	string	Control resource usage in a compile point
1	<input checked="" type="checkbox"/>	view	v:work.top	syn_allowed_resources	MS12s=3	string	Control resource usage in a compile point

The following example is for Lattice.

	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

The following example is for Xilinx.

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	view	v:work.top	syn_allowed_resources	dsps=4	string	Control resource usage in a compile point
1	<input checked="" type="checkbox"/>	view	v:work.top	syn_allowed_resources	blockmults=5	string	Control resource usage in a compile point
1	<input checked="" type="checkbox"/>	view	v:work.top	syn_allowed_resources	blockrams=3	string	Control resource usage in a compile point

Verilog Example

```
module top (datain, dataout, clk, addr, a, b, c);
output [7:0] dataout;
input clk;
input [7:0] datain;
input [7:0] addr;
input [79:0] a, b;
output reg [139:0] c
/* synthesis syn_allowed_resources = "dsps=2 | blockmults=3 |
dsp_blocks=4" */;
```

```

reg [7:0] mem [255:0]
/* synthesis syn_allowed_resources= "M20K = 1| blockrams=3|
M-RAM=3| M512s=3"*/;

always @ (posedge clk)
begin
    mem [addr]<=datain[7:0];
end
assign dataout= mem [addr];

always @ (posedge clk)
begin
    c[13:0] <= a[7:0] * b[7:0];
    c[27:14] <= a[15:8]* b[15:8];
    c[41:28] <= a[23:16]* b[23:16];
    c[55:42] <= a[31:24]* b[31:24];
    c[69:56] <= a[39:32]* b[39:32];
    c[83:70] <= a[47:40]* b[47:40];
    c[97:84] <= a[55:48]* b[55:48];
    c[111:98] <= a[63:56]* b[63:56];
    c[125:112] <= a[71:64]* b[71:64];
    c[139:126] <= a[79:72]* b[79:72];
end
endmodule

```

See [FDC Example](#), on page 108 for ways to use the attribute.

VHDL Example

```

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

entity top_ra_mux is
port (ADDR1: in std_logic_vector(7 downto 0);
      data_in : in std_logic_vector(7 downto 0);
      A : in std_logic_vector (79 downto 0);
      CLK : in std_logic;
      B : in std_logic_vector (79 downto 0);
      EN : in std_logic;
      C : out std_logic_vector (159 downto 0);
      data_out : out std_logic_vector(7 downto 0));
end top_ra_mux;

```

```
architecture rtl of top_ra_mux is
type mem_type is array (255 downto 0) of std_logic_vector (7 downto
0);
signal mem : mem_type;
signal mult_i : std_logic_vector(159 downto 0);
attribute syn_allowed_resources : string;
attribute syn_allowed_resources of mem : signal is "blockrams=1";
attribute syn_allowed_resources of mult_i : signal is
"dsp_blocks=3";
begin
    process (CLK)
    begin
        IF (CLK'event AND CLK = '1') THEN
            IF (EN = '1') THEN
                data_out <= mem(to_integer(unsigned(ADDR1)));
            END IF;
        END IF;
    end process;

    process (CLK)
    begin
        IF (CLK'event AND CLK = '1') THEN
            IF (EN = '1') THEN
                mem(to_integer(unsigned(ADDR1))) <= data_in;
            END IF;
        END IF;
    end process;

    mult_i(15 downto 0) <= std_logic_vector(unsigned(a(7 downto
0))*unsigned(b(7 downto 0)));
    mult_i(31 downto 16) <= std_logic_vector(unsigned(a(15 downto
8))*unsigned(b(15 downto 8)));
    mult_i(47 downto 32) <= std_logic_vector(unsigned(a(23 downto
16))*unsigned(b(23 downto 16)));
    mult_i(63 downto 48) <= std_logic_vector(unsigned(a(31 downto
24))*unsigned(b(31 downto 24)));
    mult_i(79 downto 64) <= std_logic_vector(unsigned(a(39 downto
32))*unsigned(b(39 downto 32)));
    mult_i(95 downto 80) <= std_logic_vector(unsigned(a(47 downto
40))*unsigned(b(47 downto 40)));
    mult_i(111 downto 96) <= std_logic_vector(unsigned(a(55 downto
48))*unsigned(b(55 downto 48)));
    mult_i(127 downto 112) <= std_logic_vector(unsigned(a(63 downto
```

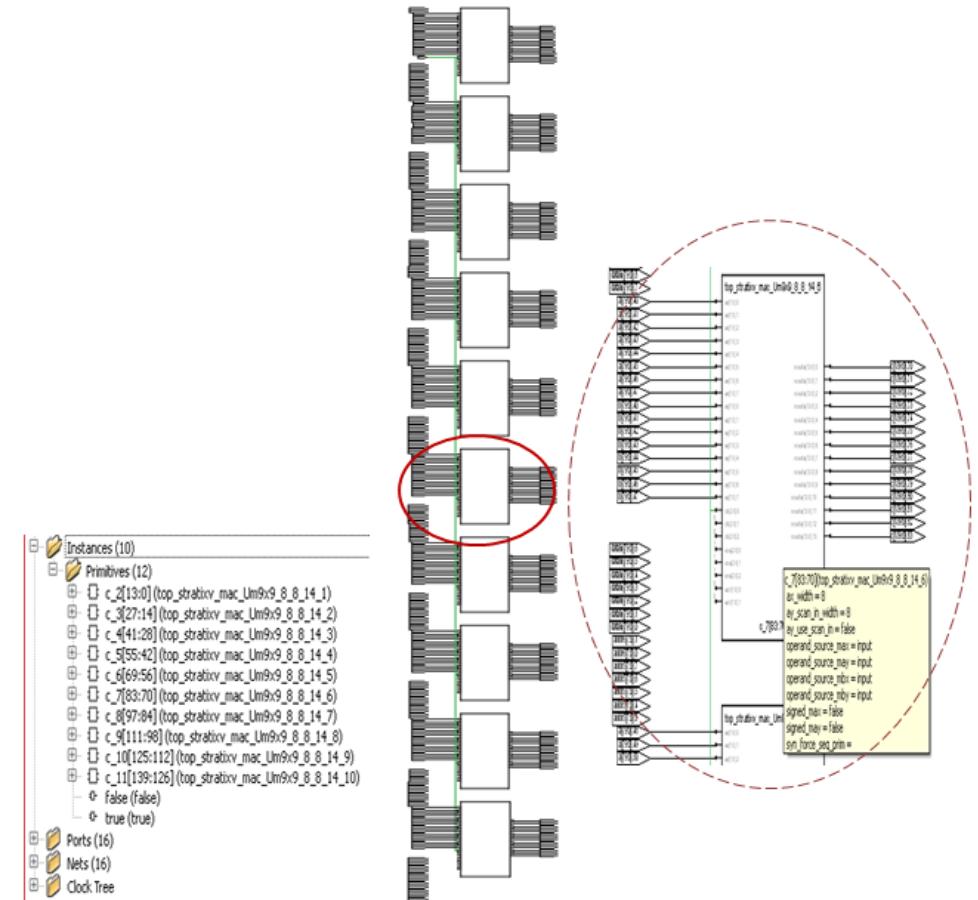
```
:  
  
      56) *unsigned(b(63 downto 56));  
mult_i(143 downto 128) <= std_logic_vector(unsigned(a(71 downto  
64)) *unsigned(b(71 downto 64)));  
mult_i(159 downto 144) <= std_logic_vector(unsigned(a(79 downto  
72)) *unsigned(b(79 downto 72)));  
  
process (CLK)  
begin  
    IF (CLK'event AND CLK = '1') THEN  
        C <= mult_i;  
    end if;  
end process;  
  
end rtl;
```

Effect of Using syn_allowed_resources on Intel FPGA Devices

The following figure shows an Intel Stratix V device before the attribute is applied with the `dsp_blocks` keyword.

Change Altera

The 10 multipliers in the design is inferred as 10 MAC blocks in Intel Stratix V device



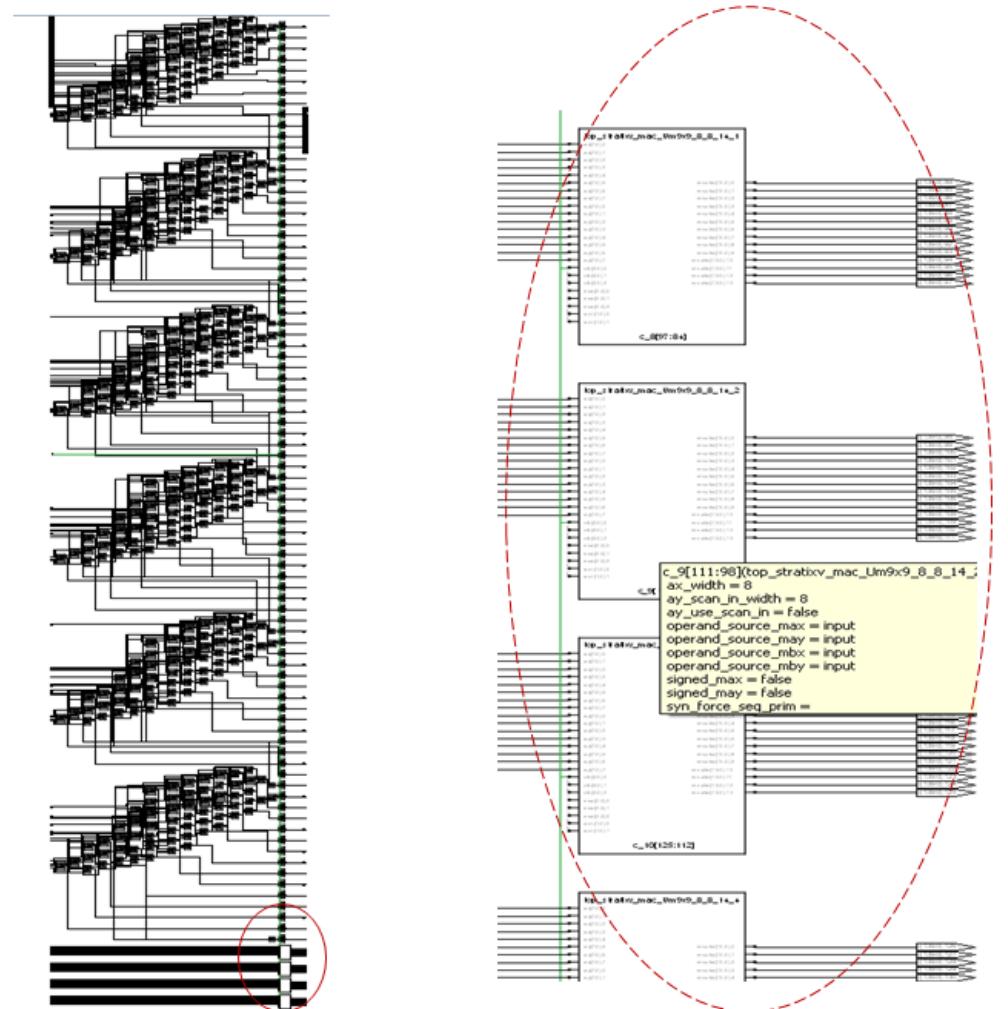
The following figure shows the attribute used with the `dsp_blocks` keyword.

Verilog `output reg [139:0] c /* synthesis syn_allowed_resources = "dsp_blocks=4" */;`

VHDL `attribute syn_allowed_resources : string;`
`attribute syn_allowed_resources of mult_i : signal is "dsp_blocks=4";`

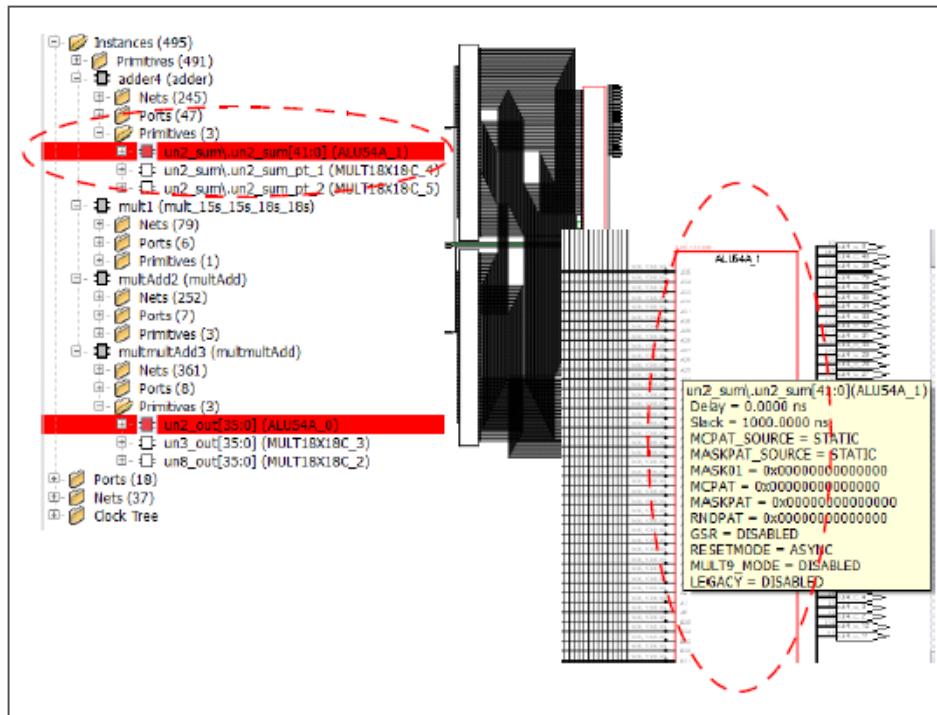
FDC

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	view	v:work.top	syn_allowed_resources	dsp_blocks=4	string	Control resource usage point



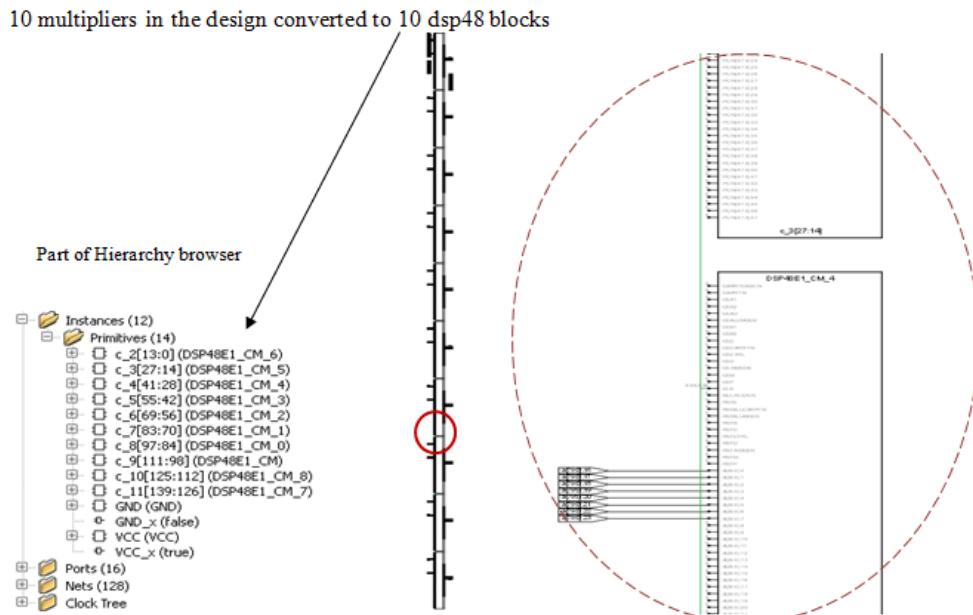
Effect of Using syn_allowed_resources on Lattice Devices

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.



Effect of Using syn_allowed_resources on Xilinx Devices

The following figure shows a Xilinx Virtex-7 device before applying the attribute, with the `dsp48` keyword:



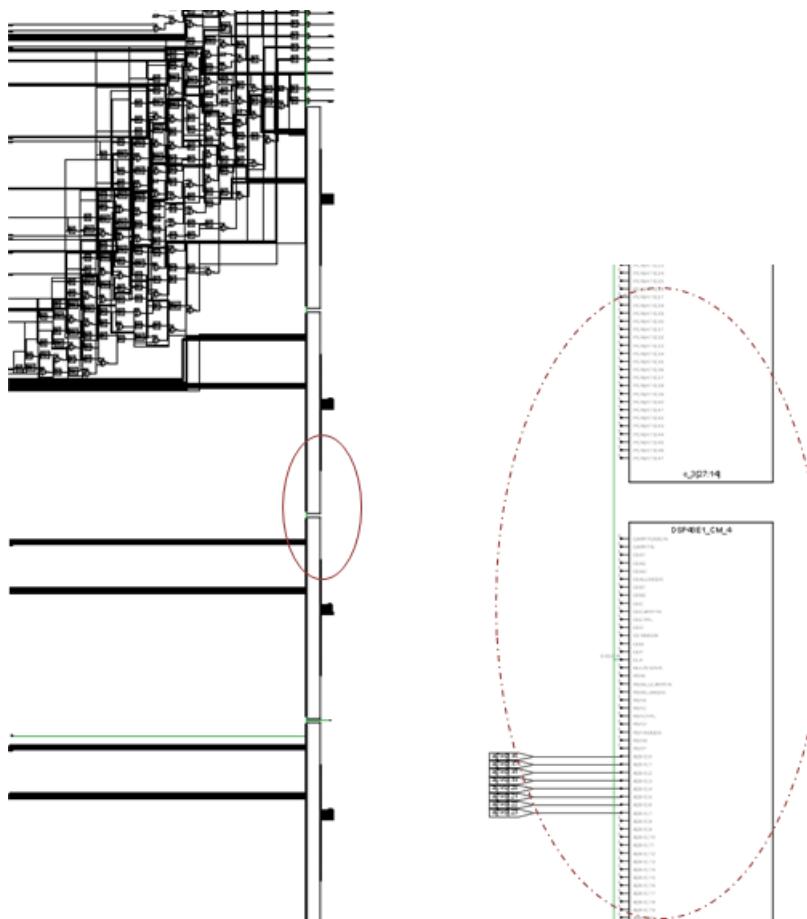
The following example shows the attribute used with the `dsp48` keyword.

```
Verilog output reg [139:0] c /* synthesis syn_allowed_resources = "dsp48=4" */;
```

```
VHDL attribute syn_allowed_resources : string;
attribute syn_allowed_resources of mult_i : signal is "dsp=4";
```

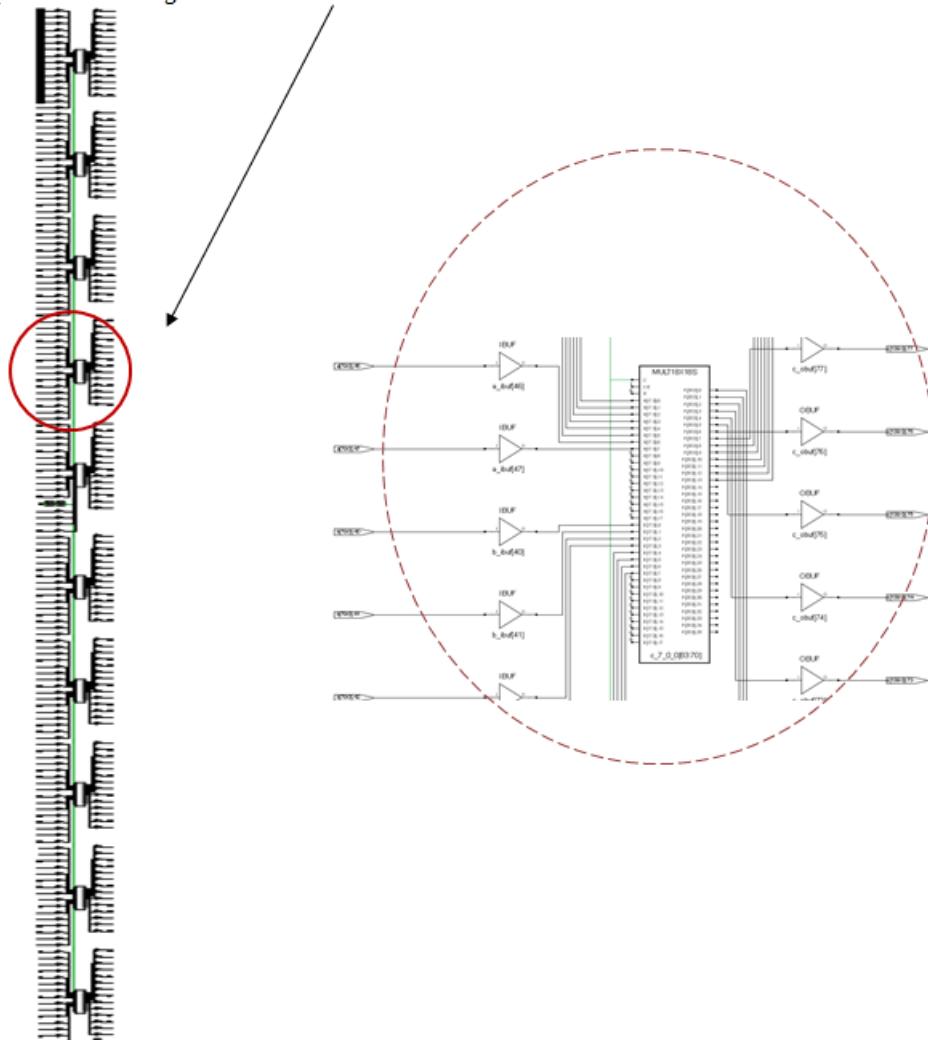
FDC

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	view	v:work.top	syn_allowed_resources	dsp=4	string	Control resource usage in a compile point



The following figure shows a Xilinx Virtex-2p device before applying the attribute, with the `blockmults` keyword.

10 multipliers in the design converted to 10 blockmults



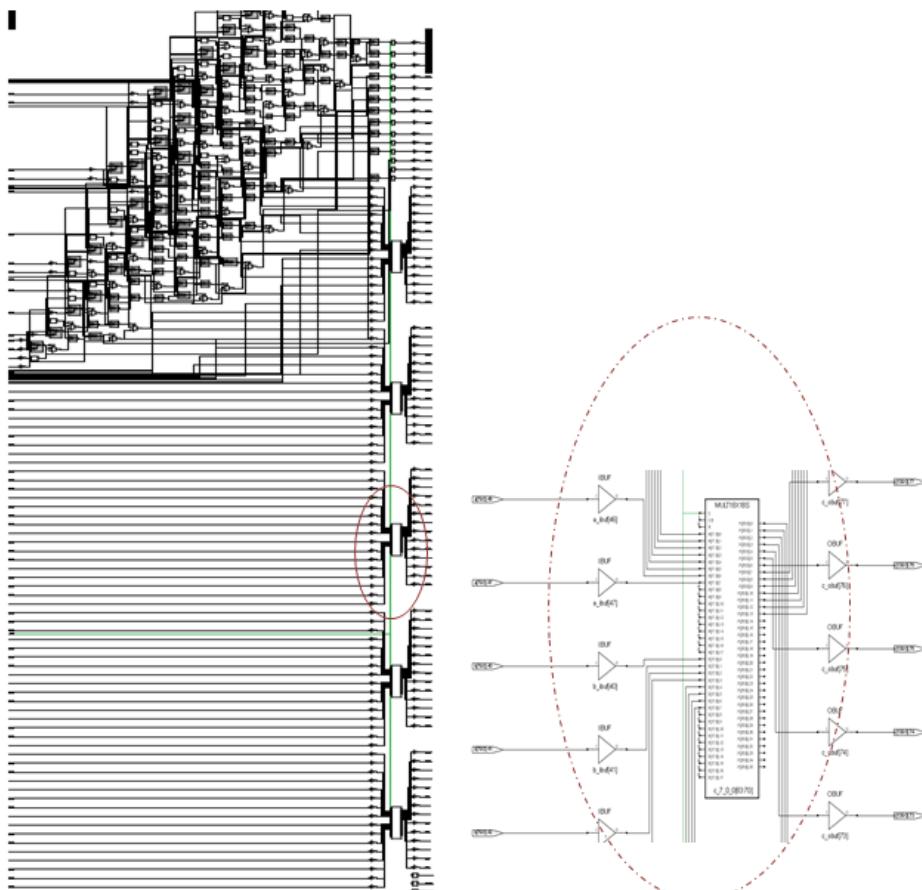
The following figure shows the attribute used with the blockmults keyword.

Verilog **output reg [139:0] c /* synthesis syn_allowed_resources = "blockmults=5" */;**

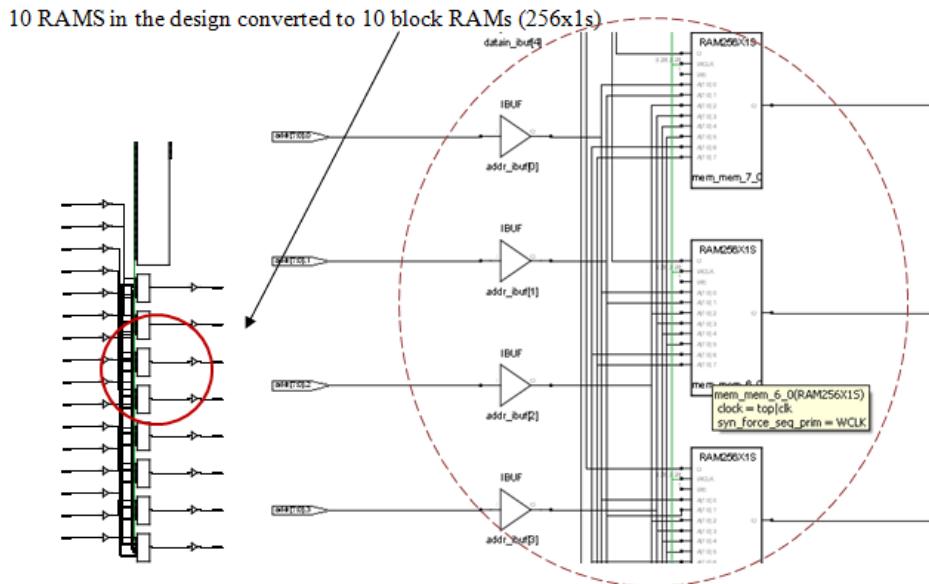
VHDL **attribute syn_allowed_resources : string;**
attribute syn_allowed_resources of mult_i : signal is "blockmults=5";

FDC

Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1 <input checked="" type="checkbox"/>	view	vwork.top	syn_allowed_resources	blockmults=5	string	Control resource usage in a compile point



The following figure shows a Xilinx Virtex-5 device before the attribute is used with the blockrams keyword.

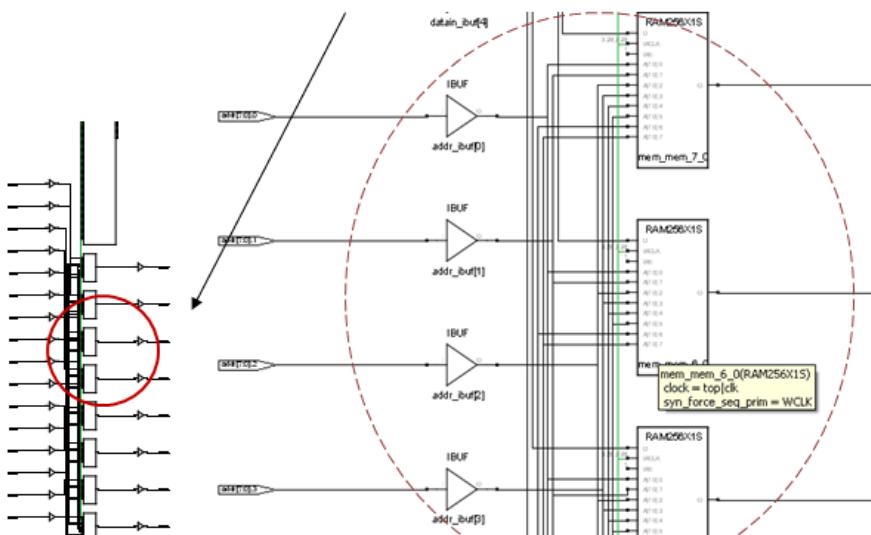


The following figure shows the attribute used with the `blockrams` keyword.

Verilog `reg [7:0] mem [255:0] /* synthesis syn_allowed_resources= "blockrams = 5**/;`

VHDL `attribute syn_allowed_resources : string;
attribute syn_allowed_resources of mem : signal is "blockrams=1";`

10 RAMS in the design converted to 5 block RAMs (256x1s) & again another 5 block RAMS (since default style is also block RAM)



Examples of Resource Limitations

The following examples show resource limitations you might encounter for your design.

Example 1 (Intel FPGA): One RAM Resource

This example (for certain Intel FPGA devices) specifies only one M512 RAM resource for the compile point v.work.single. Other memory elements in the compile point are mapped to logic resources. You can overuse the logic resources if the allowed RAM resources are inadequate for the available memory elements inferred in a given compile point.

```
define_attribute {v:work.single} syn_allowed_resources  
{M4Ks=0, M512s=1}
```

Example 2 (Intel FPGA): No RAM Resources

This example (for certain Intel FPGA devices) specifies that no Intel FPGA specific RAM resources are allowed for the compile point v.work.single. All memory elements in the compile point will be mapped to logic resources.

```
define_attribute {v:work.single} syn_allowed_resources  
{M4Ks=0, M512s=0}
```

You can overuse the logic resources if the allowed RAM resources are inadequate for the available memory elements inferred in a given compile point.

Example 3 (Xilinx): DSP Resources

This example (for certain Xilinx devices) allows the design v.work.TOP to have two DSP blocks.

```
define_attribute {v:work.TOP} syn_allowed_resources {dsps = 2}
```

For backward compatibility, DSP48 resources are treated as blockmults resources. To manually limit the use of blockmults, apply syn_allowed_resources on the top view. For example,

```
define_attribute {v:work.dsp_48_route_check}  
syn_allowed_resources {blockmults=1}
```

The resource limit for the syn_allowed_resources value includes both inferred and instantiated components. If you set DSP usage to 1 and your design infers one and instantiates one DSP48E primitive, both DSP48E primitives are honored.

```
module top (input clk,
            input [10:0] a1,
            input [10:0] a2,
            input [10:0] a3,
            input [10:0] b1,
            input [10:0] b2,
            input [10:0] b3,
            input [47:0] c1,
            input [47:0] c2,
            input [47:0] c3,
            input [6:0] opmode,
            output reg [47:0] out_1,
            output reg [47:0] out_2,
            output reg [47:0] out_3)
    /*synthesis syn_allowed_resources="dsps=1" */;

    // always @ (posedge clk)
    //     out_2 <= (a2*b2);

    always @ (posedge clk)
        out_3 <= (a3*b3);

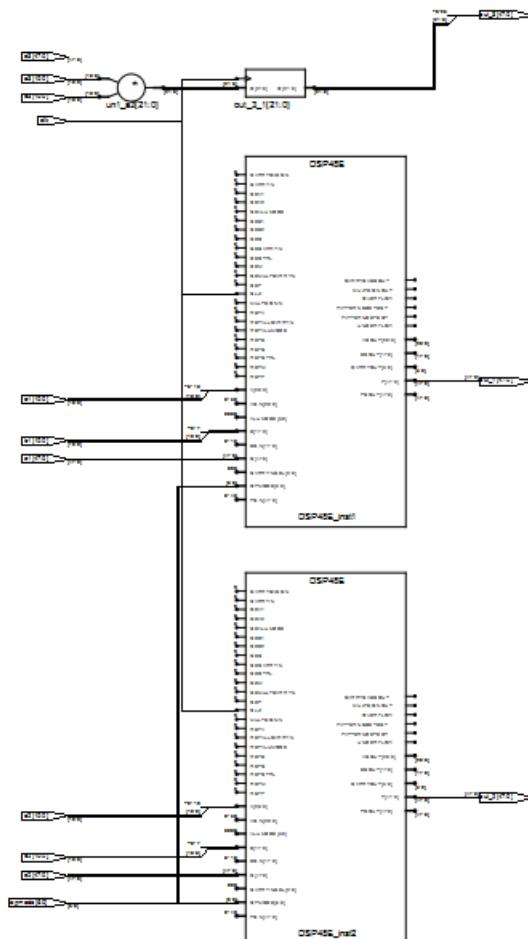
    // always @ (posedge clk)
    //     out_1 <= (a1*b1);
    //

    DSP48E    DSP48E_inst1 (
        .A(a1),
        .B(b1),
        .C(c1),
        .P(out_1),
        .CLK(clk),
        .OPMODE(opmode)
    );

    DSP48E    DSP48E_inst2 (
        .A(a2),
        .B(b2),
        .C(c2),
        .P(out_2),
        .CLK(clk),
        .OPMODE(opmode)
    );

endmodule
```

The following schematic shows that two DSP48E primitives are honored.



If you set DSP usage to 2 and your design infers only one DSP48E primitive, then this resource limit cannot be honored.

```
module top (input clk,
    input [10:0] a1,
    input [10:0] a2,
    input [10:0] a3,
    input [10:0] b1,
    input [10:0] b2,
    input [10:0] b3,
    input [47:0] c1,
    input [47:0] c2,
    input [47:0] c3,
    input [6:0] opmode,
    output reg [47:0] out_1,
    output reg [47:0] out_2,
    output reg [47:0] out_3)
    /*synthesis syn_allowed_resources="dsps=2" */;

//always @ (posedge clk)
//out_2 <= (a2*b2);

always @ (posedge clk)
    out_3 <= (a3*b3);

always @ (posedge clk)
    out_1 <= (a1*b1);

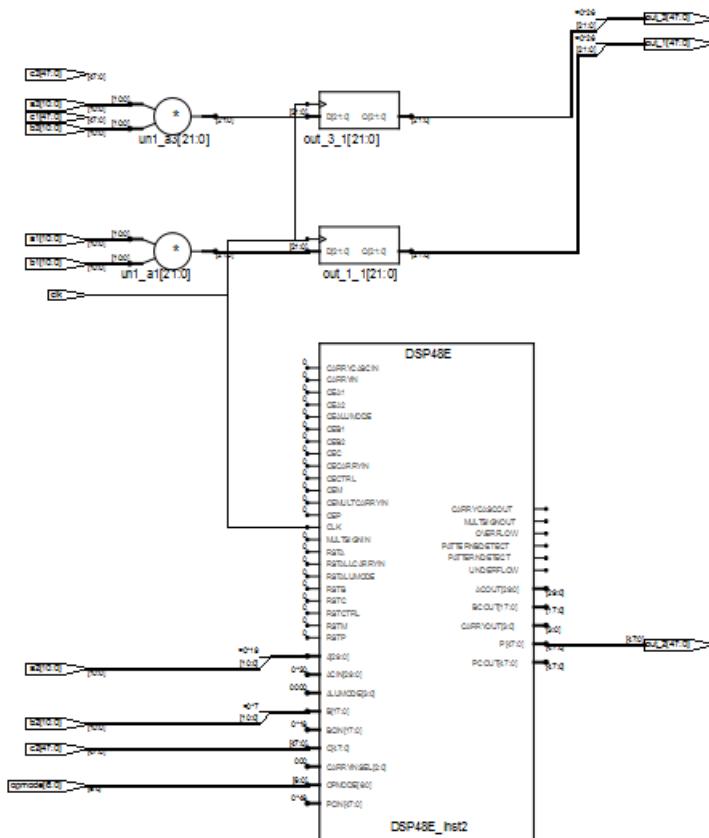
//
//DSP48E    DSP48E_inst1 (
//    .A(a1),
//    .B(b1),
//    .C(c1),
//    .P(out_1),
//    .CLK(clk), comp
//    .OPMODE(opmode)
//    );

DSP48E    DSP48E_inst2 (
    .A(a2),
    .B(b2),
    .C(c2),
    .P(out_2),
```

```
.CLK(clk),  
.OPMODE(opmode)  
);
```

endmodule

The following schematic shows that only one DSP48E primitive was honored.



Example 4: Block RAM Resources

Resource management cannot always honor the resource limitations you specify. For example, the synthesis software may not be able to separate block RAM and logic assignments for 64-bit wide RAM that use multiple RAM components. Therefore, block RAM resources can become over utilized.

The following Resource Usage Report in the log file shows that the block RAM sites are over utilized.

```
I/O ports: 44
I/O primitives: 44
IBUF          22 uses
IBUFG         1 use
OBUF          21 uses

BUFG          1 use

I/O Register bits:           42
Register bits not including I/Os: 3173 (16%)

RAM/ROM usage summary
Occupied Block RAM sites (RAMB36) : 38 of 36 (105%)
Global Clock Buffers: 1 of 32 (3%)

Total load per clock:
  clk: 3291

Mapping Summary:
Total LUTs: 2235 (11%)
```

syn_append_submodules

Directive

Renames all the modules within the hierarchy.

syn_append_submodules Values

Value	Description
<i>appendModuleName</i>	Appends a new name for all modules in the hierarchy.

Description

This directive renames all modules within the specified hierarchy. For the top-level module specified, the suffix for the new module name is appended to the submodules in the hierarchy. It is typically used to rename submodules to avoid possible naming conflicts between a generated name and the original module name. The `syn_append_submodules` directive is specified through the `cvc` compiler directives file.

You must specify this attribute on the top-level module. However, the top-level module is not renamed. To rename the top-level module, use the `syn_rename_module` attribute (see [syn_rename_module, on page 555](#)).

syn_append_submodules Syntax Specification

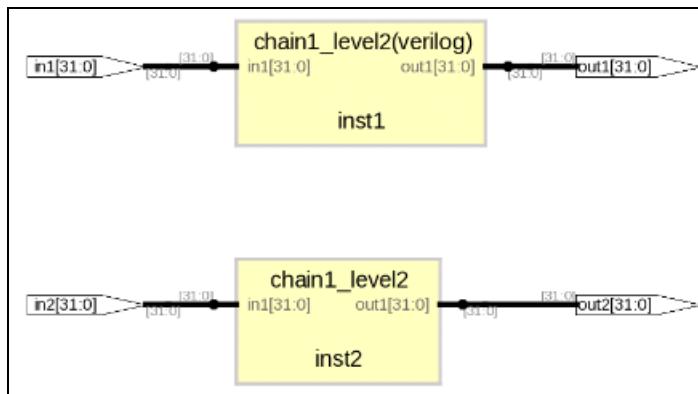
CDC File `define_directive {moduleName} syn_append_submodules
{newModuleName}` [CDC File Example](#)

CDC File Example

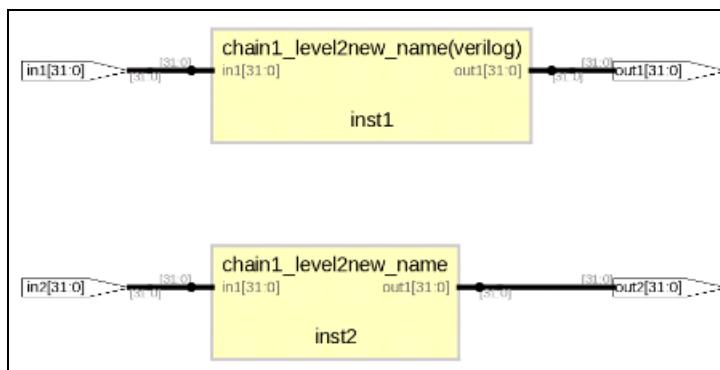
```
define_directive {v:work.chain1_level2} syn_append_submodules  
{new_name}
```

Effect of using syn_append_submodules

The following figure shows the original design, before applying syn_append_submodules:



The next figure shows the modules with new names after applying syn_append_submodules:



syn_assign_to_region

Attribute

Synplify Premier

Assigns logic to physical regions specified on the device.

Vendor	Technologies
Xilinx	Virtex-7 and later including Stacked Silicon Interconnect (SSI) technology that allows multiple dies to be combined in a single device.

Description

This region constraint must be provided in the *design_synplify.fdc* file generated from logic synthesis.

Alternatively, you can use the Design Planner to create regions and assign logic to them. The Synplify Premier tool recognizes and integrates the placement of these region constraints.

syn_assign_to_region Values

Name	Global	Object
syn_assign_to_region	No	define_attribute {object} syn_assign_to_region {sliceLLloc:sliceURloc}

Where:

- *object* - The instance name for the logic to be assigned to a region. Specify as *i:instanceName*.
- *sliceLLloc:sliceURloc* - Specifies the lower-left and upper-right SLICE coordinate system locations for the region to be created on the device. Format for this constraint is *SLICE_XlYl:SLICE_XuYu*.

syn_assign_to_region Syntax Specification

FDC	define_attribute {i: <i>instance</i> } {syn_assign_to_region} { <i>value</i> }	FDC Example
-----	---	-----------------------------

FDC Example

In the following example, instance or1200_cpu is assigned to a region created by the lower-left and upper-right SLICE coordinate system X12Y233 and X127Y281 for the specified architecture.

```
define_attribute {i:or1200_cpu} {syn_assign_to_region}  
{SLICE_X12Y233:SLICE_X127Y281}
```

Effect of using syn_assign_to_region

Applying the syn_assign_to_region attribute to an instance through the constraints file results in the constraint being forward annotated to the place-and-route tool.

Example XDC file:

```
#User specified region constraints  
create_pblock rgn_12_233_127_281  
resize_pblock [get_pblocks rgn_12_233_127_281] -add {SLICE_X-  
12Y233:SLICE_X127Y281}  
add_cells_to_pblock [get_pblocks rgn_12_233_127_281] [get_cells M2]  
-clear_locs
```

syn_assign_to_slr

Attribute

Assigns logic to a specific SLR for a device.

Vendor	Technologies
Xilinx	Virtex-7 and later including stacked Silicon Interconnect (SSI) technology that allows multiple dies to be combined in a single device

syn_assign_to_slr Values

Value	Default	Description
0	No	The bottom-most SLR
1	No	
2	No	
3	No	The top-most SLR

Description

Xilinx place-and-route software treats the SLICE coordinate system for SSI as a monolithic device. However, there is a timing penalty for signals that cross SLR boundaries, so place and route tries to automatically partition logic to the SLR. If these results are not optimal, you can use the `syn_assign_to_slr` attribute to assign logic to a specific SLR for the device. You can use this attribute with logic synthesis flows targeting Virtex-7 SSI devices.

Note: No resource estimation is performed for SLR assignments. If too much logic is assigned to a specific SLR, an error will occur in the Xilinx place-and-route tool.

The attribute must be specified in a constraint file. This attribute can be applied to instances or ports in the design. For more information about assigning SLR regions for the device, see [Xilinx Stacked Silicon Interconnect \(SSI\) Technology, on page 839](#).

syn_assign_to_slr Syntax

Name

```
define_attribute {object} syn_assign_to_slr {slrValue}
```

Where:

- *object* - Can be specified for the following:
 - **i:instance**
 - **p:port**
- *slrValue* - The SLR can be specified as 0, 1, 2, or 3.

Each SLR follows the same SLICE coordinate system for the architecture. Depending on the device selected, the number of SLRs can range between two and four.

```
define_attribute {i:inst_abc} {syn_assign_to_slr} {0}
```

Note: When SRL assignment conflicts exist, the software honors lower-level hierarchical instances within a hierarchy. For example, instance A contains instances B and C. If instance A is assigned to SLR0 and instance B is assigned to SLR1, then the assignments to SLR0 are maintained for instances A and C. However, the synthesis software still honors instance B assignment to SLR1.

syn_assign_to_slr Syntax Specification

FDC

```
define_attribute {i:instance} {syn_assign_to_slr}
{slrValue}
```

[FDC Example](#)

FDC Example

```
define_attribute {i:inst_abc} {syn_assign_to_slr} {0}
```

Effect of using syn_assign_to_slr

Applying the `syn_assign_to_slr` attribute to an instance through the constraints file results in the constraint being forward annotated to the place-and-route tool.

syn_async_reg

Attribute

The `syn_async_reg` attribute controls the minimum number of registers used to synchronize a signal across two asynchronous clock domains.

Vendor Technology

Xilinx	Virtex-6 and newer families
--------	-----------------------------

syn_async_reg Value

Value	Description
<code>numOfRegs</code>	Specifies the number of registers to be configured in the synchronization chain.

Description

The `syn_async_reg` attribute allows you to set the minimum number of registers used to synchronize a signal across two asynchronous clock domains. If a launch register is in one clock domain and a chain of capture registers is in another clock domain, the synthesis tool infers that this configuration is being used to synchronize a signal between the clock domains based on clock definitions in the FDC constraint file.

When a value is set for the `syn_async_reg` attribute (default is 2), the software reserves the number of capture registers specified for synchronization by applying an `ASYNC_REG` property to these registers. This property prevents the registers from being mapped to SRL blocks and is forward annotated to the Xilinx `.xdc` file to allow the registers to be placed in the same region. Any registers in the chain beyond the specified number do not have the property applied and are free to be packed into an SRL. For more information, see [Asynchronous Registers, on page 861](#) in the *Reference* manual.

To prevent registers in a synchronization chain from receiving and forward annotating the ASYNC_REG property, apply a syn_srlstyle attribute with a value of registers to the first capture register in the chain that should not have the applied property. For example, specifying the following attribute on register instance a3[0] allows any subsequent registers in the chain (for example, a4[0], a5[0], etc.) to be packed into an SRL.

```
define_attribute {i:a3[0]} {syn_srlstyle} {registers}
```

The syn_srlstyle attribute takes precedence over the global syn_async_reg attribute.

syn_async_reg Syntax

The syn_async_reg attribute is a global attribute that can only be specified in the FDC constraint file.

Global Attribute Default Value

Yes	If this global attribute is not applied, the default number of synchronization registers is two.
-----	--

FDC define_global_attribute {syn_async_reg} {numOfRegs} [FDC Example](#)

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	<Global>	syn_async_reg	3		
2							

Effect of Using syn_async_reg

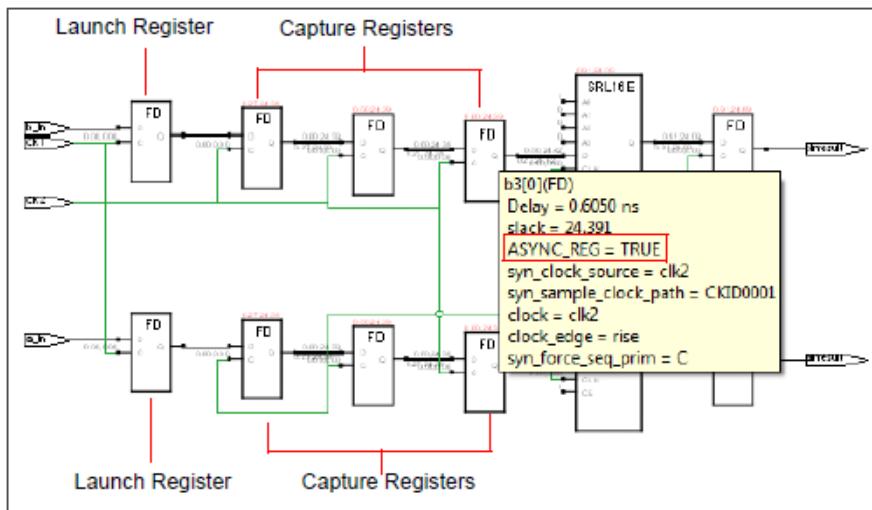
As an example, assume that the following information is included in the FDC constraint file. The clocks are specified as:

```
create_clock -name {clk1} {p:clk1} -period {20}
create_clock -name {clk2} {p:clk2} -period {25}
set_clock_groups -derive -asynchronous -name {clk1}
    -group {{c:clk1}}
set_clock_groups -derive -asynchronous -name {clk2}
    -group {{c:clk2}}
```

The `syn_async_reg` attribute used in the example is specified as:

```
define_global_attribute {syn_async_reg} {3}
```

The following view shows that three capture registers for each clock group (`clk1` and `clk2`) have the applied `ASYNC_REG` property in the capture register synchronization chain following design synthesis.



A message in the log file (FX1019) identifies capture register chains with the `ASYNC_REG` property. You can use `Edit->Find` to locate specific instances with the property or use `Tcl` find commands to manipulate collections of registers with the `ASYNC_REG` property. For example, you can use the following commands from the `Tcl` window:

- Create a collection of all FD registers clocked by `clk2` with the property `ASYN_REG==TRUE`:

:

```
set c1 [find * -hier -filter  
{@inst_of == FD && @clock == clk2 && @ASYNC_REG == TRUE}]
```

For the example, the collection includes {i:a1[0]} {i:a2[0]} {i:b1[0]} {i:b2[0]}.

- Create a collection of all registers clocked by clk1 from net b_in:

```
set c2 [find * -in [c_list [expand -through {n:b_in}]]  
-filter @clock == clk1]
```

For the example, the collection includes {i:b_inreg}.

- Create a collection of all registers on clock clk2 from the previous c2 collection with the ASYN_REG==TRUE property:

```
set c3 [find * -in [c_list [expand -from $c2]]  
-filter @clock == clk2 && @ASYNC_REG == TRUE]
```

syn_auto_insert_bufg

Attribute

Synplify Pro, Synplify Premier

Use this attribute to automatically insert a BUFG primitive on high fanout nets that are non-critical or less critical control signals (such as set, reset, enable, or clear).

Vendor	Technology
Xilinx	Virtex-6 and newer families UltraScale family

syn_auto_insert_bufg Value

Default	Global	Object
1	Yes	n:net

Value	Description
0	Disables automatic BUFG insertion.
1	Enables automatic BUFG insertion.

Description

Use this attribute to automatically insert a BUFG primitive on high fanout nets that are non-critical or less critical control signals (such as set, reset, enable, or clear).

Fanout criteria used to determine automatic BUFG insertion is based on the following:

- Must exceed the Fanout Guide specified on the Device tab of the Implementation Options panel or the value specified for the `syn_maxfan` attribute.
- Inserts a BUFG on nets driving loads that have more than 50% control signals, either synchronous or asynchronous as follows:
 - Inserts a BUFG on the control net when the fanout threshold is exceeded and less than 50% of the BUFG resources are used.
 - Is limited by BUFG resources; Xilinx guidelines specify two BUFG components on non-clock nets for the entire design.
 - Applies the BUFG on the larger of two high fanout nets.
 - Inserts a BUFG only on the control signal for nets with mixed loads (control and datapath signals).
- If a net drives both the control and datapath signals, then the BUFG is not inserted.
- Considers the values for the user-specified `syn_insert_buffer` and `syn_global_buffers` attributes.

The `syn_auto_insert_bufg` attribute is globally enabled by default. To disable BUFG insertion, apply this attribute locally on nets in the design. Message FX1035 identifies and provides the syntax to prevent BUFG insertion on nets, which you can use to copy and paste into a constraint file easily. Otherwise, if this attribute is disabled globally, you can use the `syn_insert_buffer` attribute to apply specific types of buffers on nets in the design instead.

You can only specify this attribute in the FDC constraint file.

syn_auto_insert_bufg Syntax

Apply the `syn_auto_insert_bufg` attribute globally or on individual nets of the design in the FDC constraint file.

FDC	<code>define_global_attribute syn_auto_insert_bufg {1 0}</code>	FDC Example
	<code>define_attribute syn_auto_insert_bufg netName {1 0}</code>	

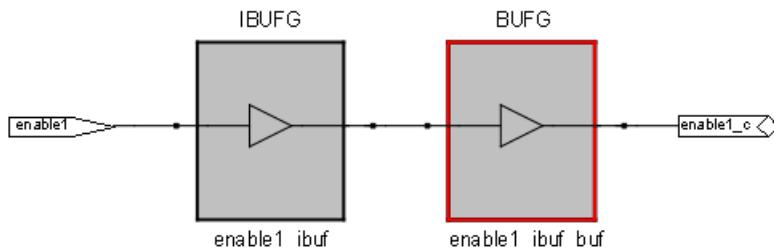
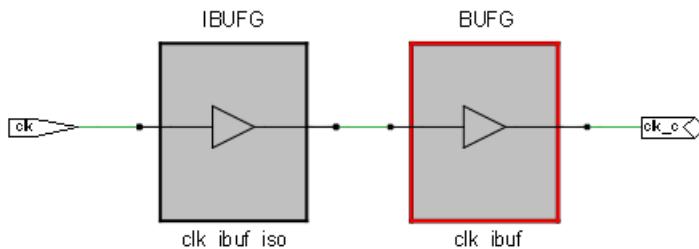
```
define_global_attribute {syn_auto_insert_bufg} {1}
define_attribute {n:rst} {syn_auto_insert_bufg} {0}
```

FDC Example

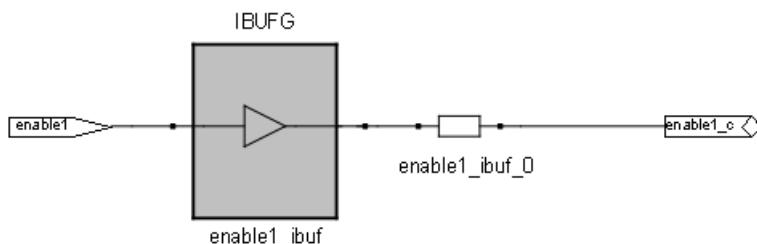
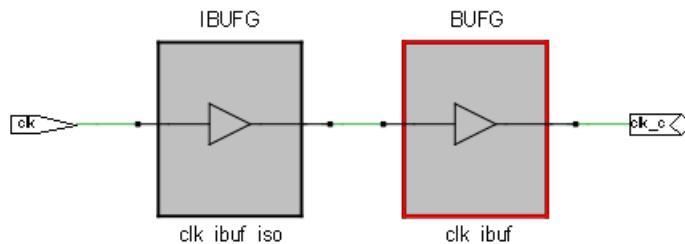
	Enable	Object Type	Object	Attribute	Value	Value Type	Description	Comment
1	<input checked="" type="checkbox"/>	<any>	<Global>	syn_auto_insert_bufg	1			
2								

Effect of Using syn_auto_insert_bufg

This example shows the Technology view results when syn_auto_insert_bufg is set to 1 globally for this design:



Here is the Technology view results with `syn_auto_insert_bufg` set to 0 globally for this design:



syn_auto_insert_bufgmux

Attribute

Controls global automatic BUFGMUX insertion. If inserted, you can specify synchronous (SYNC) or asynchronous (ASYNC) clocks for the BUFGMUX primitives.

Vendor	Technology
Xilinx	All

syn_auto_insert_bufgmux Value

Default	Global
ASYNC	Yes

Value	Description
false	Disables automatic BUFGMUX insertion
ASYNC	Forward annotates the BUFGMUX property as CLK_SEL_TYPE = "ASYNC"
SYNC	Forward annotates the BUFGMUX property as CLK_SEL_TYPE = "SYNC"

The BUFGMUX will not be inferred when there are more than two unique clocks (for example, three clocks are feeding the MUX).

Description

Use the syn_auto_insert_bufgmux attribute to either globally disable automatic BUFGMUX insertion (false) or insert BUFGMUX primitives for synchronous (SYNC) or asynchronous (ASYNC) clocks. ASYNC is the default setting. When two unique clocks feed the MUX, you can select either BUFGMUX primitives

with synchronous (`CLK_SEL_TYPE="SYNC"`) or asynchronous (`CLK_SEL_TYPE="ASYNC"`) clocks for insertion. This BUFGMUX clock property is forward-annotated to the place-and-route tool.

syn_auto_insert_bufgmux Syntax

Apply the `syn_auto_insert_bufgmux` attribute globally in the FDC constraint file.

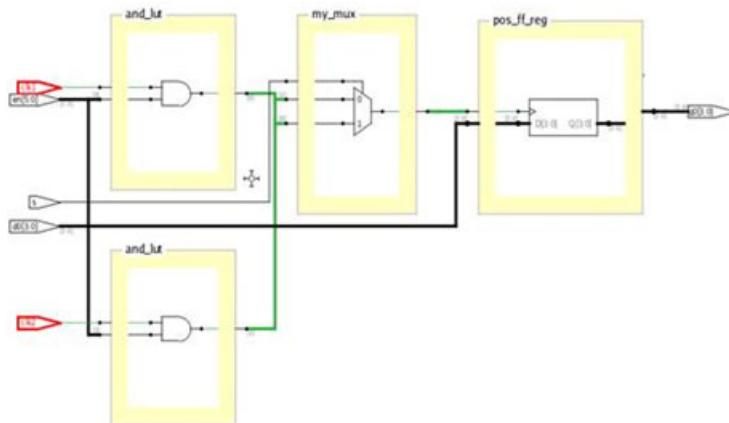
FDC	<code>define_global_attribute syn_auto_insert_bufgmux {false ASYNC SYNC}</code>
-----	---

The following example disables auto insertion of the BUFGMUX primitive.

```
define_global_attribute {syn_auto_insert_bufgmux} {false}
```

Effect of Using syn_auto_insert_bufgmux

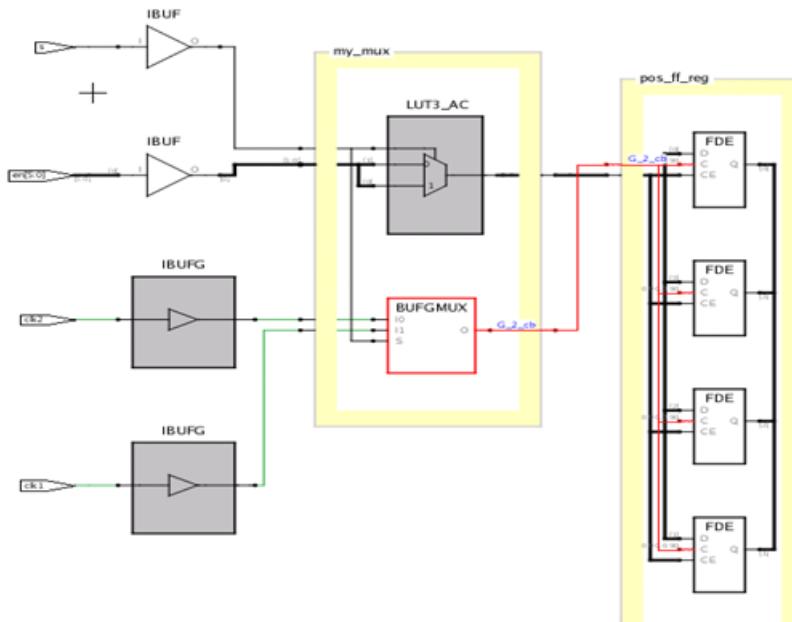
This example has two clocks defined on the top-level ports `clk1` and `clk2`. These clocks are gated with enable signals (`en[1]` and `en[0]`) driving the MUX inputs. This condition allows for BUFGMUX inference.



You can set the attribute for inference for the design:

```
define_global_attribute {syn_auto_insert_bufgmx} {SYNC}
```

The BUFGMUX is then inferred for the clock MUX after you run synthesis (see the following figure). The enable signals are multiplexed and fed into the clock enable pins of the register loads. You can verify the CLK_SEL_TYPE="SYNC" property for the BUFGMUX primitive in the EDIF file generated.



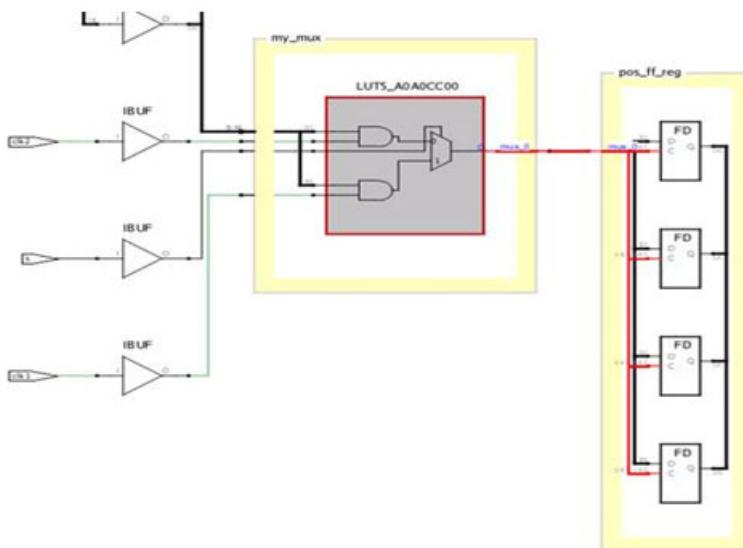
An example of the clock properties forward-annotated to the place-and-route tool is shown in the EDIF and simulation netlists below.

```
(contents
  (instance q_cb (viewRef PRIM (cellRef BUFGMUX (libraryRef VIRTEX)))
    (property CLK_SEL_TYPE (string "SYNC"))
  )

  BUFGMUX q_cb_c (
    .O(q_cb),
    .I0(clk2_c),
    .I1(clk1_c),
    .S(s_c)
  );
  defparam q_cb_c.CLK_SEL_TYPE = "SYNC";
```

For the same design, the MUX is implemented as a simple LUT if you specify that BUFGMUXes should not be inferred:

```
define_global_attribute {syn_auto_insert_bufgmux} {false}
```



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.
- Use the compiler constraints file (`cdc`) to set a comprehensive black box directive at the top level. For example, the following sets the syn_black_box attribute on all architectures of the sub entity in the MyLib library:

```
define_directive {v:MyLib.sub} {syn_black_box} {1}
```

See [Specifying Directives in a CDC File, on page 141](#) in the *User Guide* for the procedure and syntax details, and [The Compiler Directives File, on page 12](#) for examples.

- If you want to define a black box when you have an srs, ngc, 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:

syn_isclock	Specifies a clock port on a black box.
syn_tpd<n>	Sets timing propagation for combinational delay through the black box.
syn_tsu<n>	Defines timing setup delay required for input pins relative to the clock.
syn_tco<n>	Defines the timing clock to output delay through the black box.

If the black-box timing constraints are not defined, the tool times paths to/from the black box with the system clock.

Black Box Source Code Directives for Gated Clocks

Synplify Pro, Synplify Premier

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 150.

<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:

black_box_pad_pin	Indicates that a black box is an I/O pad for the rest of the design.
black_box_tri_pins	Indicates tristates on black boxes.

For more information on black boxes, see Instantiating Black Boxes in Verilog, on page 124, and Instantiating Black Boxes in VHDL, on page 399.

syn_black_box Syntax Specification

CDC File	<code>define_directive {object} {syn_black_box} {1}</code>	CDC Example
Verilog	<code>object /* synthesis syn_black_box */;</code>	Verilog Example
VHDL	<code>attribute syn_black_box of object : objectType is true;</code>	VHDL Example

CDC Example

```
define_directive {v:MyLib.sub} {syn_black_box} {1}
```

Verilog Example

```
module top(clk, in1, in2, out1, out2);  
    input clk;  
    input [1:0]in1;  
    input [1:0]in2;  
  
    output [1:0]out1;  
    output [1:0]out2;  
  
    add      U1 (clk, in1, in2, out1);  
    black_box_add U2 (in1, in2, out2);  
  
endmodule  
  
module add (clk, in1, in2, out1);  
  
    input clk;  
    input [1:0]in1;  
    input [1:0]in2;  
  
    output [1:0]out1;  
    reg [1:0]out1;  
  
    always@(posedge clk)  
        begin  
            out1 <= in1 + in2;  
        end  
endmodule  
  
module black_box_add(A, B, C) /* synthesis syn_black_box */;  
    input [1:0]A;  
    input [1:0]B;  
  
    output [1:0]C;  
  
    assign C = A + B;  
  
endmodule
```

VHDL Example

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

entity add is
    port(
        in1 : in std_logic_vector(1 downto 0);
        in2 : in std_logic_vector(1 downto 0);
        clk : in std_logic;
        out1 : out std_logic_vector(1 downto 0));
end;

architecture rtl of add is
begin

process(clk)
begin
    if(clk'event and clk='1') then
        out1 <= (in1 + in2);
    end if;
end process;
end;

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

entity black_box_add is
    port(
        A : in std_logic_vector(1 downto 0);
        B : in std_logic_vector(1 downto 0);
        C : out std_logic_vector(1 downto 0));
end;

architecture rtl of black_box_add is

attribute syn_black_box : boolean;
attribute syn_black_box of rtl: architecture is true;
begin

C <= A + B;
end;

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

```
entity top is
  port(
    in1 : in std_logic_vector(1 downto 0);
    in2 : in std_logic_vector(1 downto 0);
    clk : in std_logic;
    out1 : out std_logic_vector(1 downto 0);
    out2 : out std_logic_vector(1 downto 0));
  end;

  architecture rtl of top is

  component add is
    port(
      in1 : in std_logic_vector(1 downto 0);
      in2 : in std_logic_vector(1 downto 0);
      clk : in std_logic;
      out1 : out std_logic_vector(1 downto 0));
    end component;

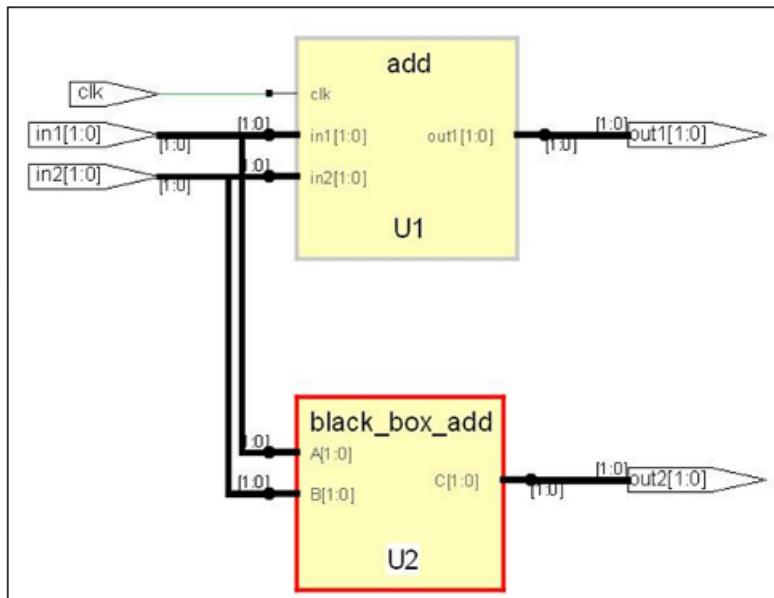
  component black_box_add
    port(
      A : in std_logic_vector(1 downto 0);
      B : in std_logic_vector(1 downto 0);
      C : out std_logic_vector(1 downto 0));
    end component;

  begin
    U1: add port map(in1, in2, clk, out1);
    U2: black_box_add port map(in1, in2, out2);
  end;
```

Effect of Using syn_black_box

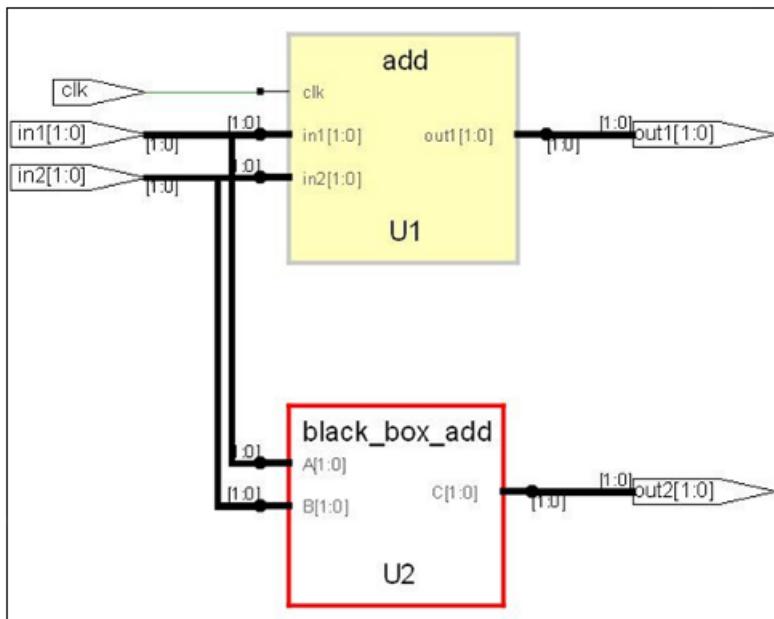
When the `syn_black_box` attribute is not set on the `black_box_add` module, its content are accessible, as shown in the example below:

```
module black_box_add(input [1:0]A, [1:0]B, output [1:0]C);
```



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

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



syn_bram_cascade_height

Attribute

Controls block RAM cascading.

Vendor	Devices
Xilinx	Kintex UltraScale, UltraScale+, Virtex UltraScale, UltraScale+, and Zynq UltraScale+

syn_bram_cascade_height Values

Value	Description
1 - 8	The number of Block RAMs to cascade.

Description

The `syn_bram_cascade_height` attribute sets the number of block RAMs to cascade.

For information on block RAM cascading, see [Cascading Block RAM](#) in the *Reference Manual*.

syn_bram_cascade_height Syntax

FDC	<code>define_attribute {mem[3:0]} {syn_bram_cascade_height} {value}</code>	FDC
Verilog	<code>reg [data_width-1:0] mem [2**addr_width-1:0] /* synthesis syn_bram_cascade_height = value */;</code>	Verilog Example
VHDL	<code>attribute syn_bram_cascade_height: integer; attribute syn_bram_cascade_height of mem: signal is value;</code>	VHDL Example

Verilog Example

```
module test (addr_a,addr_b,clk_a,clk_b,wra,web,din,out);  
  
parameter data_width = 4;  
parameter addr_width = 18;  
  
input [addr_width-1:0] addr_a, addr_b;  
input clk_a, clk_b, wra, web;  
input [data_width-1:0] din;  
output reg [data_width-1:0] out;  
  
reg [data_width-1:0] mem [2**addr_width-1:0] /* synthesis  
syn_bram_cascade_height = 8 */;  
  
// ram code  
  
always@ (posedge clk_a)  
begin  
    if (wra)  
        mem[addr_a] <= din;  
  
end  
  
always@ (posedge clk_b)  
begin  
    if (web)  
        out <= mem[addr_b];  
    else  
        out <= mem[addr_b];  
end  
  
endmodule
```

VHDL Example

```
Library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.std_logic_arith.all;  
use IEEE.std_logic_unsigned.all;  
  
entity Dual_Port_ReadFirst is  
generic (data_width: integer :=4;  
address_width: integer :=18);
```

```
port (write_enable: in std_logic;
      write_clk, read_clk: in std_logic;
      data_in: in std_logic_vector (data_width-1 downto 0);
      data_out: out std_logic_vector (data_width-1 downto 0);
      write_address: in std_logic_vector (address_width-1 downto 0);
      read_address: in std_logic_vector (address_width-1 downto 0)
    );
end Dual_Port_ReadFirst;

architecture behavioral of Dual_Port_ReadFirst is
type memory is array (2**address_width-1) downto 0) of
std_logic_vector (data_width-1 downto 0);
signal mem: memory;

signal reg_write_address: std_logic_vector (address_width-1 downto 0);
signal reg_write_enable: std_logic;

attribute syn_bram_cascade_height: integer;
attribute syn_bram_cascade_height of mem: signal is 8;

begin
  register_enable_and_write_address:
  process (write_clk, write_enable, write_address, data_in)
begin
  if (rising_edge(write_clk)) then
    reg_write_address <= write_address;
    reg_write_enable <= write_enable;
  end if;
end process;

write:
process (read_clk, write_enable, write_address, data_in)
begin
  if (rising_edge(write_clk)) then
    if (write_enable='1') then
      mem(conv_integer(write_address)) <= data_in;
    end if;
  end if;
end process;

read:
process (read_clk, write_enable, read_address, write_address)
begin
  if (rising_edge(read_clk)) then
    data_out <= mem(conv_integer(read_address));
  end if;
end process;
```

end behavioral;

syn_clean_reset

Attribute

The `syn_clean_reset` attribute changes asynchronous reset registers, which cannot go into DSP48 blocks, into synchronous reset logic.

Vendor	Technologies
Xilinx	UltraScale and earlier devices

syn_clean_reset Values

Value	Description	Object	Default	Global
<code>resetLogicValue</code>	Specifies the synchronous reset logic values.	Module / Architecture	None	Yes

Description

The `syn_clean_reset` attribute changes asynchronous reset registers, which cannot go into DSP48 blocks, into synchronous reset logic. The resulting implementation is not logically equivalent to the one specified in the RTL code and does not match the RTL and gate-level simulation.

Note: This attribute can only be applied on registers that get packed into the DSP48 blocks. This attribute does not work for all registers in the design.

:

syn_clean_reset Syntax

FDC	define_global_attribute syn_clean_reset {resetLogicValue}	FDC Example
Verilog	object /* syn_clean_reset = "resetLogicvalue" */;	Verilog Example
VHDL	attribute syn_clean_reset : string; attribute syn_clean_reset of object : objectType is "resetLogicvalue";	VHDL Example

FDC Example

Enable	Object Type	Object	Attribute	Value	Value Type	Description
<input checked="" type="checkbox"/>	<any>	<Global>	syn_clean_reset	dsp48_no_simulate,one_flop	string	allows synplify to convert async resets to sync resets so registers go into DSP48s

The following is an example of the syntax:

```
define_global_attribute syn_clean_reset
{dsp48_no_simulate,one_flop}
```

Verilog Example

```
object /* synthesis syn_clean_reset = "dsp48_no_simulate,one_flop" */;
```

With the following Verilog code:

```
module syn_clean_rst( d1,d2,rst,clk,out);
  input [7:0] d1,d2;
  input rst;
  input clk;
  output reg [14:0]out;
  wire [14:0] mul_out;

  assign mul_out = d1 * d2;

  always@(posedge clk or posedge rst)
  begin
    if (rst)
      out <= 15'b0;
    else
      out<= mul_out;
  end
endmodule
```

VHDL Example

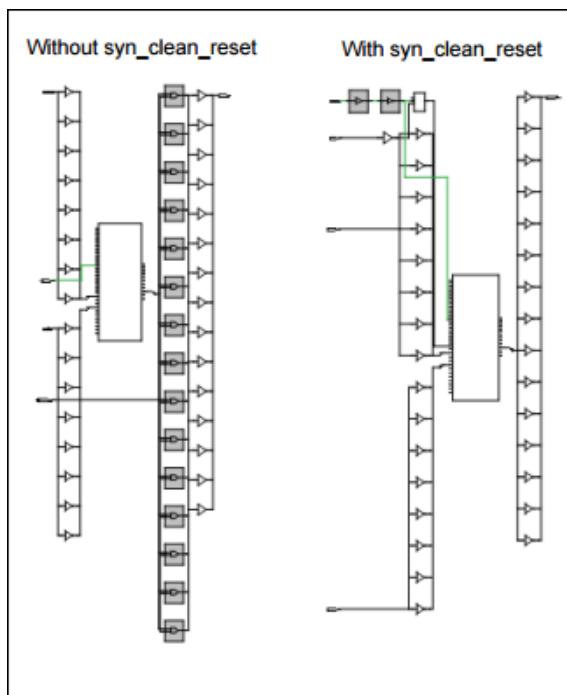
```
attribute syn_clean_reset of object : objectType is "dsp48_no_simulate,one_flop";
```

In the VHDL code (apply to top-level architecture):

```
entity test is
    port (in1_p, in1_n : in std_logic;
          clk : in std_logic;
          out1 : out std_logic);
    attribute syn_clean_reset : string;
    attribute syn_clean_reset of test : architecture is
        "dsp48_no_simulate,one_flop";
end test;
```

Effect of Using syn_clean_reset

The `syn_clean_reset` attribute can be specified to convert asynchronous reset to synchronous reset allowing registers to be packed into a DSP block. The mapped schematic views are shown with and without the `syn_clean_reset` value.



syn_clock_gmux_proxy

Attribute

Allows the software to automatically generate proxy clocks at the output of the BUFGMUX.

Vendor	Devices
Xilinx	7 Series (Virtex-7, Kintex-7, and Artix-7) devices

syn_clock_gmux_proxy Values

Default	Global	Object
0	Yes	BUFGMUX instance

Value	Description
0 false	Disables automatic generation of proxy clocks at the output of the BUFGMUX.
1 true	Enables automatic generation of proxy clocks at the output of the BUFGMUX.

Description

This attribute allows the tool to automatically generate proxy clocks at the output of the BUFGMUX, providing a one-to-one correspondence for the clocks on the inputs. The proxy clocks exist at the output of the BUFGMUX and will be grouped asynchronously to each other, while inheriting any other clock grouping specifications of their source/master clock.

You can use the `set_case_analysis` constraint to specify which clocks to propagate through the BUFGMUX. However, the disadvantage of using this constraint is that

- You must identify the BUFGMUX and set its value.
- Optimizations can only be done for one clock.

For details, see [set_case_analysis, on page 351](#).

So use the `syn_clock_gmux_proxy` attribute instead, whenever possible.

Note the following conditions:

- If the clock at the output of the BUFGMUX is already specified, then automatic inferencing of generated clocks does not occur for this BUFGMUX instance.
- If the `set_case_analysis` constraint has been applied on a select pin of a BUFGMUX, then automatic inferencing of generated clocks does not happen.
- The appropriate `create_generated_clock` and `set_clock_groups` constraints are forward-annotated to the Vivado `xdc` file.

`syn_clock_gmux_proxy` Syntax

FDC	<code>define_attribute object {syn_clock_gmux_proxy} {1 0}</code>	FDC Example
	<code>define_global_attribute {syn_clock_gmux_proxy} {1 0}</code>	

Verilog	<code>object /* synthesis syn_clock_gmux_proxy = 1 0 */</code>	Verilog Example
---------	--	---------------------------------

VHDL	<code>attribute syn_clock_gmux_proxy : boolean;</code> <code>attribute syn_clock_gmux_proxy object : objectType is</code> <code>true false;</code>	VHDL Example
------	--	------------------------------

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description	Comment
1	<input checked="" type="checkbox"/>	<any>	<Global>	<code>syn_clock_gmux_proxy</code>	1			
2	<input checked="" type="checkbox"/>	<any>	i: BUFGMUX_INST1	<code>syn_clock_gmux_proxy</code>	1			

It is recommended that you apply this attribute on specific BUFGMUX instances to minimize its effect on runtime. However, when you apply this attribute globally and disable a specific instance, the software can handle this situation successfully. For example:

```
define_global_attribute {syn_clock_gmux_proxy} {1}
```

```
define attribute {i:BUFGMUX INST1} {syn clock gmux proxy} {1}
```

Verilog Example

Here is a code snippet that shows how to specify this attribute in Verilog:

```
BUFGMUX BUFGMUX_hier(.S(S), .IO(IO), .I1(I1), .O(O));
    /*synthesis syn clock gmux proxy=1 */;
```

VHDL Example

Here is a code snippet that shows how to specify this attribute in VHDL:

```

entity bufgmux_vhd is
  port(
    OCLK : out std_logic;
    I0 : in std_logic;
    I1 : in std_logic;
    S : in std_logic
  );
end bufgmux_vhd;

architecture struct of bufgmux_vhd is

component BUFGMUX
  port
    O : out std_logic;
    I0 : in std_logic;
    I1 : in std_logic;
    S : in std_logic
  );
end component;

attribute syn_clock_gmux_proxy : boolean;
attribute syn_clock_gmux_proxy of U1 : label is true;

begin
  U1 : BUFGMUX
  port map (
    O => OCLK,
    I0 => I0,

```

```

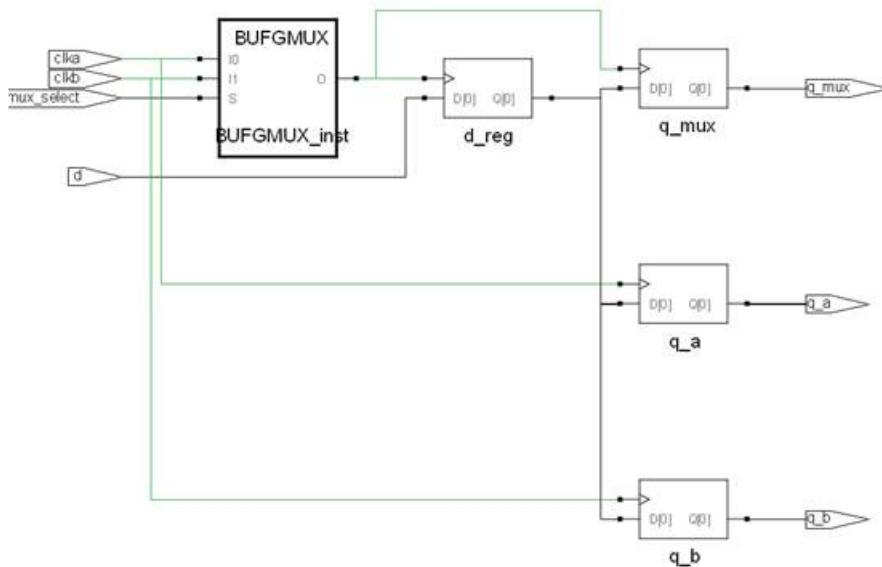
    I1 => I1,
    S  => S
);

end struct;

```

Effect of Using syn_clock_gmux_proxy

For the circuit below, the synthesis tool propagates both clka and clkb through the BUFGMUX during timing analysis. However, since timing analysis does not automatically treat the clock signals as mutually exclusive on the output of the BUFGMUX, paths between clka and clkb after the mux are timed even though both clocks are not actually propagating simultaneously through the mux in the real circuit. Therefore, false paths will be timed after the mux unless you explicitly constrain clka and clkb in the FDC file to be a false path or asynchronous to each other. Using this attribute allows clka and clkb to remain synchronous for paths before the mux, but their proxy clocks (after the mux) are treated as asynchronous. When you use this attribute, more clocks may be created in your design and the timing reports, but can also provide better area and/or timing quality of results (QoR).



syn_clock_priority

Attribute

Lets you set a clock priority to resolve logical or physical clock conflicts.

Vendor	Devices
Xilinx	Virtex 6 and older families

syn_clock_priority Values

Default	Global	Object
N/A	Yes	port/net

Value	Description
Integer	In the ucf file, a TIMESPEC statement with a priority value overrides a competing TIMESPEC without a priority value. For competing TIMESPECS where both have priority values, the one with the lowest positive integer value wins. If the priority values are the same, no priority is established between them.

Description

The `syn_clock_priority` attribute lets you set a clock priority to resolve logical or physical clock conflicts. This ensures that the clocks are forward-annotated correctly in the UCF file.

Use the `syn_clock_priority` attribute to ensure that the correct clock is forward-annotated. For example, you can use it in the following cases:

- When you override DCM clocks with declared clocks, ISE does not honor this. Instead of manually editing the UCF file, you can use the `syn_clock_priority` attribute to assign a priority to a particular clock.

-
- The FPGA synthesis tools allow multiple clocks to propagate along a single net, either through `unate` logic or through a mux for timing all possible clocks and clock combinations. Xilinx ISE cannot do this, and uses the `PRIORITY` keyword in UCF to indicate clock priority in case of conflict. You can set clock priority with this attribute.

See [Setting Clock Priority in Xilinx Designs \(Legacy\)](#), on page 271 in the *User Guide* for more information.

The tool forward-annotates the clock priority to the UCF file in a `TIMESPEC` or `PERIOD` statement. See [Effect of Using syn_clock_priority](#), on page 172 for details.

The FPGA tools handle derived and declared clocks slightly differently when they forward-annotate the priority:

Attribute set on **FPGA Synthesis Forward-annotation**

Declared clock	Directly forward-annotated to the <code>TIMESPEC</code> / <code>PERIOD</code> for the clock.
Derived clock	First generates a <code>TIMESPEC</code> / <code>PERIOD</code> that represents the nature of the derived clock, and then forward-annotates this as a declared clock.

Messages and Warnings

The tool generates messages in certain situations:

Situation	Message
When derived clocks are overridden	Message to set clock priority with the <code>syn_clock_priority</code> attribute.
When the tool detects multiple timing for a path	

When the priority set on the DCM base clock is higher than the priority of the override clock on a DCM output

When you assign priority for a clock that is one of multiple clocks specified for the same object with the `create_clock` native SDC command

When you assign the same priority to both inputs of a mux

Usage warnings.

Syntax Specification

SDC file `define_attribute {n|p|j:netName|portName|BUFGname} {syn_clock_priority} {value}`

You can only set this attribute in the constraint file. You set the attribute on ports or nets, and the tool applies the value to the clock. Clocks can be declared or derived clocks. You can also set it on a BUFG instance, but not on any other instances.

The `syn_clock_priority` value must be greater than or equal to 1, with 1 being the highest priority. Take the following clock declaration:

```
define_clock {n:u_fx_clkrstgen.clk_100_dcm} -name {clk_100}
-period 10 -clockgroup sys_group -rise 0 -fall 5
```

If you apply `syn_clock_priority` as follows, the clock on the specified net is assigned the highest priority, 1:

```
define_attribute {n:u_fx_clkrstgen.clk_100_dcm}
{syn_clock_priority} {1}
```

The following example sets the highest priority for the DCM CLKFX output:

```
define_attribute {n:dcm_module_b.clk0fx} {syn_clock_priority} {1}
```

SCOPE Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	net	n:inst1.CLK180_BUF	syn_clock_priority	1	integer	Establishes clock priority to be forward annotated to the Xilinx UCF file

```
define_attribute {n:inst1.CLK180_BUF} {syn_clock_priority} {1}
```

Effect of Using syn_clock_priority

Before applying the attribute, content in synplicity.ucf file:

```
# Constraints generated by Synplify Pro maprc, Build 1020R
# Product Version "F-2012.03-SP1"
#
# Period Constraints
#Begin clock constraints
# 1001 : define_clock {clk} -name {clk} -freq {200} -clockgroup {default_clkgroup_0}
# c:\xilinx\syn_clock_priority.sdc
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 5.000 ns HIGH 50.00%;

# 1002 : define_clock {n:inst1.CLK180_BUF} -name {clk_400} -period {10} -clockgroup {default_clkgroup_2}
# c:\syn_clock_priority.sdc
NET "inst1/CLK180_BUF" TNM_NET = "inst1_CLK180_BUF";
TIMESPEC "TS_inst1_CLK180_BUF" = PERIOD "inst1_CLK180_BUF" 10.000 ns HIGH 50.00%;
#End clock constraints

# I/O Registers Packing Constraints
INST "out2" IOB=TRUE;
INST "out1" IOB=FALSE;
INST "b1" IOB=FALSE;
INST "a1" IOB=FALSE;

# End of generated constraints
```

After applying the attribute, content in synplicity.ucf file:

```
# Constraints generated by Synplify Pro maprc, Build 1020R
# Product Version "F-2012.03-SP1"
#
# Period Constraints
#Begin clock constraints
# 1001 : define_clock {clk} -name {clk} -freq {200} -clockgroup {default_clkgroup_0}
# c:\xilinx\syn_clock_priority.sdc
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 5.000 ns HIGH 50.00%;

# 1002 : define_clock {n:inst1.CLK180_BUF} -name {clk_400} -period {10} -clockgroup {default_clkgroup_2} -priority {1}
# c:\xilinx\syn_clock_priority.sdc
NET "inst1/CLK180_BUF" TNM_NET = "inst1_CLK180_BUF";
TIMESPEC "TS_inst1_CLK180_BUF" = PERIOD "inst1_CLK180_BUF" 10.000 ns HIGH 50.00% PRIORITY 1;
#End clock constraints

# I/O Registers Packing Constraints
INST "out2" IOB=TRUE;
INST "out1" IOB=FALSE;
INST "b1" IOB=FALSE;
INST "a1" IOB=FALSE;

# End of generated constraints
```

syn_connect_hrefs

Directive

Synplify Premier

Allows cross-module referencing (XMR) of signals in a design using a CDC file, without having to modify or update the Verilog or VHDL design.

syn_connect_hrefs Values

Value	Description
-readers	Hierarchical path of the reader (connect). Specify one or more readers.
<i>hierPathOfReader</i>	Specifies the hierarchical path for the readers. <i>topModule.instance1.instance2.instance3.signal</i> or <i>top.inputPortName</i>
-writer	Hierarchical path of the writer (source). Specify only one writer.
<i>hierPathOfWriter</i>	Specifies the format of hierarchical path for the writer as follows: <i>topModule.instance1.instance2.instance3.signal</i> or <i>top.inputPortName</i>

Description

Allows cross-module referencing (XMR) of signals in a design using a CDC file, without having to modify or update the Verilog or VHDL design. You can specify the `syn_connect_hrefs` directive to add signals for debugging or reference signals deep within the hierarchy of the design that drive the connections.

:

syn_connect_hrefs Syntax Specification

CDC File `syn_connect_hrefs -readers {hierPathOfReader}
-writer {hierPathOfWriter}`

[CDC File: XMR
Example With a
Verilog Design](#)

[CDC File: XMR
Example With a
VHDL Design](#)

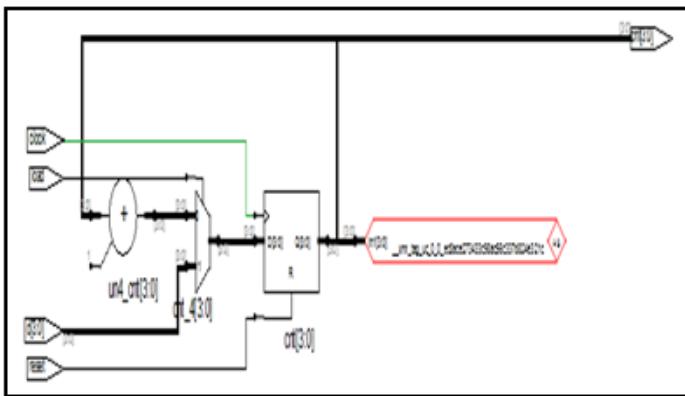
CDC File: XMR Example With a Verilog Design

This Verilog example shows an assignment within the RTL that came from outside the RTL. For example:

```
assign cnt_top = top.cnt_inst.cnt;
```

The CDC equivalent is shown below:

```
syn_connect_hrefs -readers {top.cnt_top}  
-writer {top.cnt_inst.cnt}
```



For the effect of using the `syn_connect_hrefs` attribute with a Verilog design, see [Verilog Example: Top-Level top.sv, on page 176](#).

CDC File: XMR Example With a VHDL Design

The CDC file can be specified as shown below:

```
syn_connect_hrefs -readers {top.xmr_net.we_int}
                     -writer {top.wr_en}

syn_connect_hrefs -readers {work.myTopDesign.out2}
                     -writer {work.myTopDesign.inst1.iz.b}
```

Where: `hierPathOfReader` and `hierPathOfRWriter` values can be specified as shown in the table below.

Value**Description**

Library name ¹	work
Entity name <i>topModule</i> ²	myTopDesign, top
Instance name	xmr_net, inst1, iz
Signal name	we_int, wr_en, out2(0), b

1. Library name is optional when the default library is work.
2. If multiple architectures exist for the entity, then the last architecture is selected.

For the effect of using the `syn_connect_hrefs` attribute with a VHDL design, see [VHDL Example: Top-Level *top.vhd*, on page 177](#).

Effect of Using `syn_connect_hrefs`

These examples show how XMR can be used in a CDC file to update a Verilog or VHDL design without modifying the RTL.

Verilog Example: Top-Level *top.sv*

Use XMR to connect `top.scale` to `top.I1.scale_fac` in the test case below:

```
module top (input clk, [3:0] din1, [3:0] din2, [3:0] scale,
            output [3:0] dout);
    sub I1(clk, din1, din2, dout);
endmodule

module sub(input clk,[3:0] din1, din2, output reg [3:0] dout);
    //Signal accessed using XMR
    logic [3:0] scale_fac;

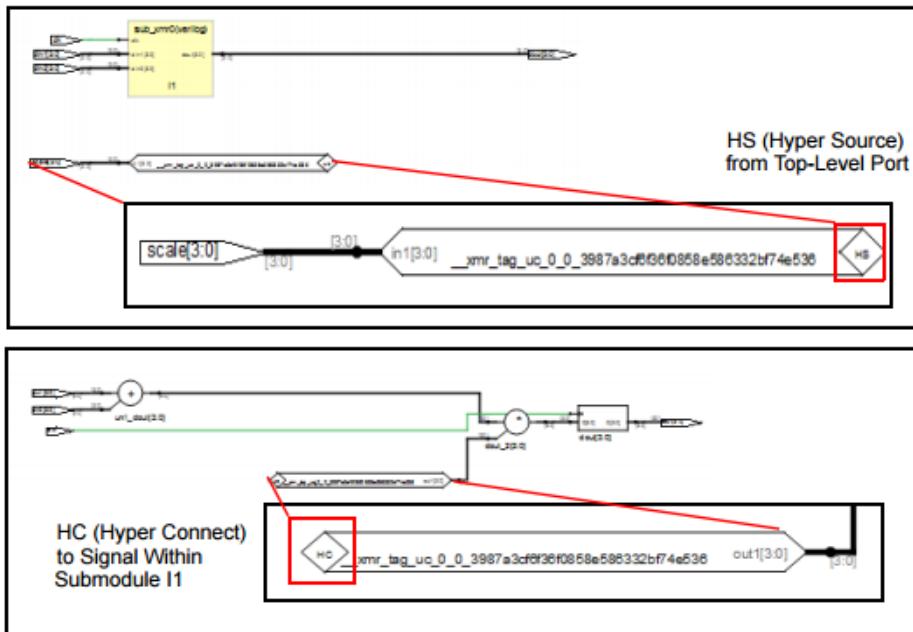
    always @ (posedge clk)
        dout <= (din1 + din2) * scale_fac;
endmodule
```

Example: *hrefs.cdc*

You can specify that the top-level port (`top.scale`) is connected to the internal signal (`top.I1.scale_fac`) of the submodule `I1` in a CDC file:

```
syn_connect_hrefs -readers {top.I1.scale_fac} -writer {top.scale}
```

After you run compile, the top-level port uses hyper source to hyper connect to the signal within submodule I1, when you push into it as shown in the schematic views below.



VHDL Example: Top-Level top.vhd

Use XMR to connect top.wr_en to top.xmr_net.we_int of the submodule test in the test case below:

```

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

entity top is
  port(
    clk,rd_en,wr_en : in std_logic;
    rd_addr_ip      : in std_logic_vector(10 downto 0);
    wr_addr_ip      : in std_logic_vector(10 downto 0);
    data_ip          : in std_logic_vector(8 downto 0);
    data_op          : out std_logic_vector(8 downto 0));
end entity top;

```

```

architecture rtl of top is
component test
    port(
        clk,rd_en : in std_logic;
        rd_addr_ip,wr_addr_ip : in std_logic_vector(10 downto 0);
        data_ip : in std_logic_vector(8 downto 0);
        data_op : out std_logic_vector(8 downto 0));
    end component test;

begin
---- submodule test.vhd instantiated.
xmr_net : test
    port map (clk => clk,
              rd_en      => rd_en,
              rd_addr_ip => rd_addr_ip,
              wr_addr_ip => wr_addr_ip,
              data_ip     => data_ip,
              data_op     => data_op);
end rtl;

```

VHDL Example: Submodule test.vhd

```

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

entity test is
    port(
        clk,rd_en      : in std_logic;
        rd_addr_ip    : in std_logic_vector(10 downto 0);
        wr_addr_ip    : in std_logic_vector(10 downto 0);
        data_ip       : in std_logic_vector(8 downto 0);
        data_op       : out std_logic_vector(8 downto 0));
    end entity test;

architecture rtl of test is
type ram_array is array(2047 downto 0) of
    std_logic_vector(8 downto 0);
signal ram_data : ram_array;
signal we_int : std_logic; --- XMR signal defined for XMR
connection.

```

```
begin
    write_to_ram: process(clk) begin
        if rising_edge(clk) then
            if (we_int = '1') then
                ram_data(to_integer(unsigned(wr_addr_ip))) <= data_ip;
            end if;
        end if;
    end process write_to_ram;

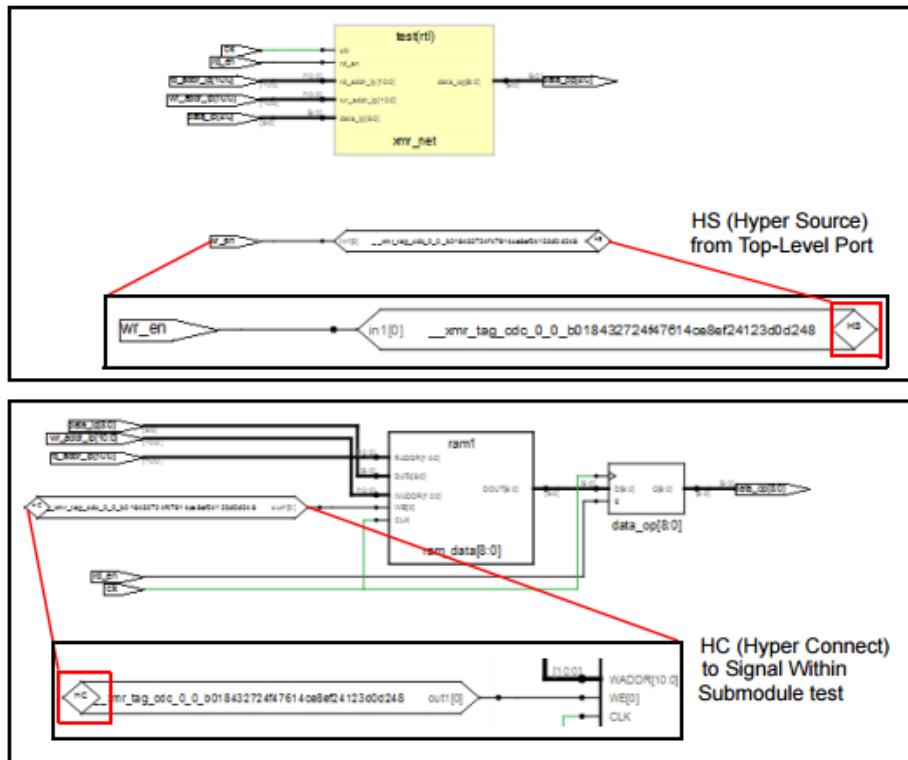
    read_from_ram: process(clk)      begin
        if (rising_edge (clk)) then
            if (rd_en = '1') then
                data_op <= (ram_data(to_integer(unsigned(rd_addr_ip))));
            end if;
        end if;
    end process read_from_ram;
end rtl;
```

Example: hrefs.cdc

You can specify that the top-level port (`top.wr_en`) is connected to the internal signal (`top.xmr_net.we_int`) of the submodule test in a CDC file:

```
syn_connect_href -readers {top.xmr_net.we_int}
                    -writer {top.wr_en}
```

After you run `compile`, the top-level port uses hyper source to hyper connect to the signal within submodule test, when you push into it as shown in the schematic views below.



Limitations

Cross-module referencing support includes the following limitations:

- Mixed language (Verilog to VHDL or VHDL to Verilog) designs are not supported.
- The XMR signals cannot be used for logic functions, such as and, or, or xnor.
- The XMR signals cannot be used for arithmetic functions, such as add, subtract, or multiply.

syn_cp_use_fast_synthesis

Attribute

Synplify Premier

Specifies manual compile points in the manual or automatic compile point flow.

Vendor	Technology
Intel FPGA	Stratix III, Cyclone III, and newer families
Xilinx	Virtex-4, Spartan-3, and newer families

Description

You can use fast synthesis mode when specifying manual compile points in the manual compile point flow and automatic compile point flow. However, you can set the `syn_cp_use_fast_synthesis` attribute to apply fast synthesis mode for specific manual compile points. The fast synthesis option is available for the Synplify Premier tool.

To enable this attribute, set `syn_cp_use_fast_synthesis` to 1. The attribute can be applied on a selected manual compile point view/module for the design. Define the compile points in the top-level constraint file.

For information about Automatic Compile Points, see [The Automatic Compile Point Flow, on page 627](#).

syn_cp_use_fast_synthesis Summary

Value	Global	Object
0	Yes	Compile point view
1	Yes	Compile point view

Constraint File Syntax and Example

FDC

```
define_attribute {v:cp_name} syn_cp_use_fast_synthesis {1|0}
```

syn_diff_io

Attribute

The `syn_diff_io` attribute controls differential I/O buffer inferencing for Xilinx designs.

Vendor	Technology
Xilinx	Virtex-7 and older families

syn_diff_io Values

Value	Description
0 false (Default)	Disables automatic differential I/O buffer inferencing.
1 true	Enables automatic differential I/O buffer inferencing.

Description

The `syn_diff_io` attribute controls differential I/O buffer inferencing in Xilinx designs: IBUFDS, IBUFGDS, OBUFDS, OBUFTDS and IOBUFDS. Attach the attribute to the inputs of the buffer. Use this attribute to identify differential input buffers when using either combinational or sequential style to code it.

The `syn_diff_io` data type is Boolean. A value of 1 or true enables automatic differential I/O buffer inferencing. The default is false or 0, no automatic inferencing.

The `syn_diff_io` attribute is supported in the compile point flow.

syn_diff_io Syntax

Global Attribute	Object
Yes	Port

FDC `define_attribute object {syn_diff_io} {1|0}` [FDC Example](#)
`define_global_attribute {syn_diff_io} {1|0}`

Verilog `object /* synthesis syn_diff_io = 1 | 0 */` [Verilog Example](#)

VHDL `attribute syn_diff_io : true|false;`
`attribute syn_diff_io of object : objectType is true|false;` [VHDL Example](#)

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	port	pin1_p	syn_diff_io	1	boolean	Allows inference of differential I/O pads
2	<input checked="" type="checkbox"/>	port	pin1_n	syn_diff_io	1	boolean	Allows inference of differential I/O pads
3	<input checked="" type="checkbox"/>	port	piout1	syn_diff_io	1	boolean	Allows inference of differential I/O pads
4	<input checked="" type="checkbox"/>	port	piout2	syn_diff_io	1	boolean	Allows inference of differential I/O pads

Verilog Example

```
module test(in1_p, in1_n, clk, out1,out2);
    input in1_p/* synthesis syn_diff_io = 1*/,
          in1_n/* synthesis syn_diff_io = 1*/;
    input clk;
    output out1/* synthesis syn_diff_io = 1*/,
          out2/* synthesis syn_diff_io = 1*/;
    reg out1,out2;
    reg in1;
    always@(in1_p or in1_n)
        if (in1_p != in1_n) in1 = in1_p;
    always@(posedge clk)
        out1 <= in1;
    assign out2 = ~out1;
endmodule
```

VHDL Example

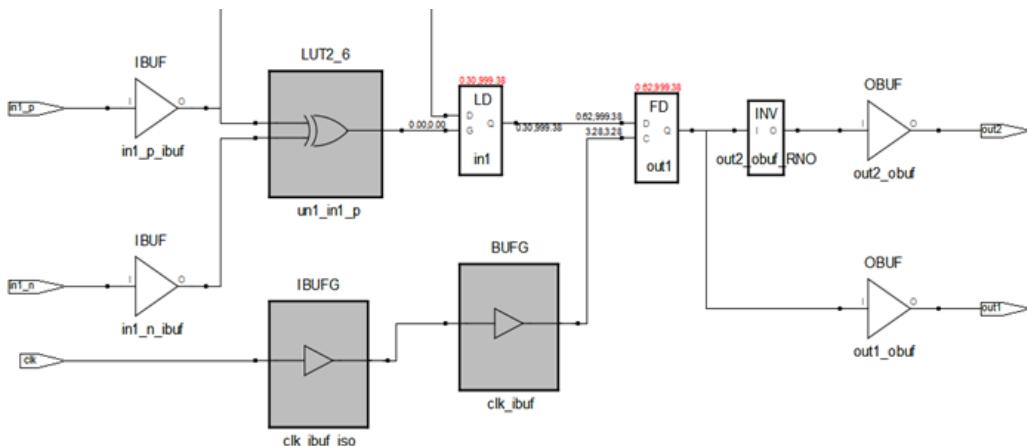
```
library ieee;
use ieee.std_logic_1164.all;
entity test is
    port (in1_p : in std_logic;
          in1_n : in std_logic;
          clk : in std_logic;
          out1 : out std_logic;
          out2 : out std_logic);
attribute syn_diff_io: boolean;
attribute syn_diff_io of in1_p,in1_n,out1,out2: signal is true;
end test;
architecture arch of test is
    signal in1 : std_logic;
    signal tmp1 : std_logic;
begin
    process(in1_p,in1_n)
    begin
        if(in1_p /= in1_n)then
            in1 <= in1_p;
        end if;
    end process;
    process(clk)
    begin
        if(clk'event and clk = '1')then
            out1 <= in1;
            tmp1 <= in1;
        end if;
    end process;
    out2 <= not tmp1;
end arch;
```

Effect of Using syn_diff_io

The following figure shows the attribute set to 0. The software disables automatic differential I/O buffer inferencing:

```
Verilog input in1_p /*synthesis syn_diff_io = 0*/;
           in1_n /*synthesis syn_diff_io = 0*/;
           output out1 /*synthesis syn_diff_io = 0*/;
           out2 /*synthesis syn_diff_io = 0*/;
```

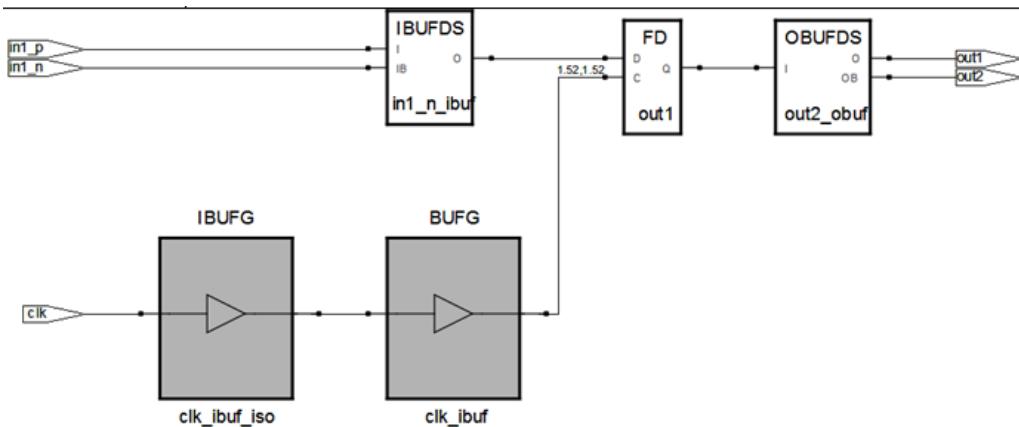
```
VHDL attribute syn_diff_io of in1_p,in1_n,out1,out2:signal is
false;
```



The next figure shows the attribute set to 1. The software enables automatic differential I/O buffer inferencing:

Verilog input in1_p /*synthesis syn_diff_io = 1*/;
 in1_n /*synthesis syn_diff_io = 1*/;
 output out1 /*synthesis syn_diff_io = 1*/;
 out2 /*synthesis syn_diff_io = 1*/;

VHDL attribute syn_diff_io of in1_p,in1_n,out1,out2:signal is
 true;



syn_direct_enable

Attribute, Directive

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

Technology	Default Value	Global	Object
Achronix: All	None	No	Net
Intel FPGA: All	None	No	Net
Lattice: LIFMD, Platform Manager, ECP, SC, XP, MachXO and newer families	None	No	Net
Microchip: PolarFire, RTG4, SmartFusion, Fusion, IGLOO, Accelerator, ProASIC, and newer families	None	No	Net
Xilinx: All	None	No	Net

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	<code>define_attribute {object} syn_direct_enable {1}</code>	FDC Example
Verilog	<code>object /* synthesis syn_direct_enable = 1 */;</code>	Verilog Example
VHDL	<code>attribute syn_direct_enable of object : objectType is true;</code>	VHDL Example

FDC Example

Enable	Object Type	Object	Attribute	Value	Value Type	Description
<input checked="" type="checkbox"/>	<any>	<Global>	<code>syn_direct_enable</code>	1	boolean	Preferred clock enable

Verilog Example

```
module direct_enable(q1, d1, clk, e1, e2, e3);
parameter size=5;
input [size-1:0] d1;
input clk;
input e1,e2;
input e3 /* synthesis syn_direct_enable = 1 */;
output reg [size-1:0] q1;

(posedge clk)
  if (e1&e2&e3)
    q1 = d1;
endmodule
```

VHDL Example

```

library ieee;
use ieee.std_logic_1164.all;

entity direct_enable is
    port (
        d1 : in std_logic_vector(4 downto 0);
        e1,e2,e3,clk : in std_logic;
        q1 : out std_logic_vector(4 downto 0));
attribute syn_direct_enable: boolean;
attribute syn_direct_enable of e3: signal is true;
end;

architecture d_e of direct_enable is
begin
    process (clk) begin
        if (clk = '1' and clk'event) then
            if (e1='1' and e2='1' and e3='1') then
                q1<=d1;
            end if;
        end if;
    end process;
end architecture;

```

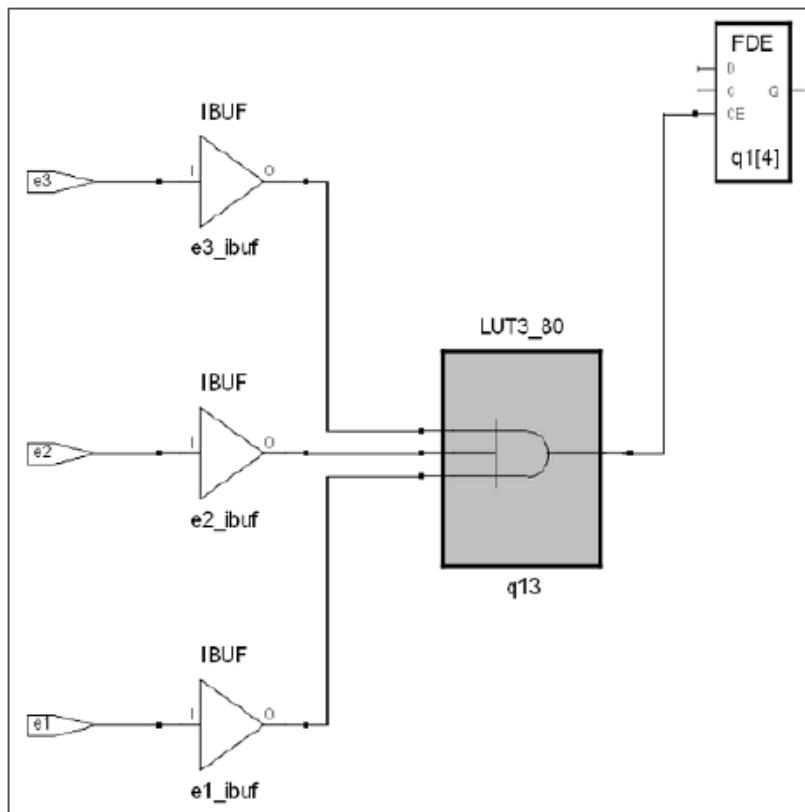
See [VHDL Attribute and Directive Syntax](#), on page 401 for different ways to specify VHDL attributes and directives.

Effect of Using syn_direct_enable

Before applying syn_direct_enable:

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

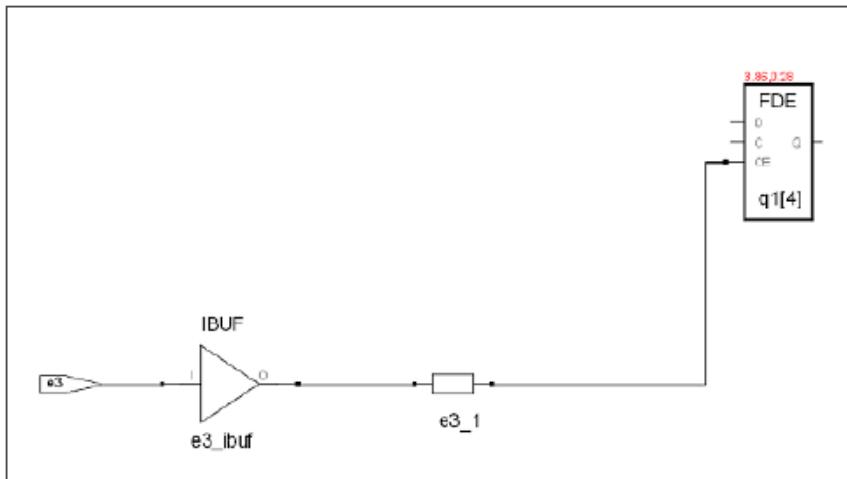
VHDL attribute syn_direct_enable of e3: signal is false;



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	LIFMD, Platform Manager, ECP, SC, XP, MachXO and newer families
Microchip	PolarFire, RTG4, SmartFusion, Fusion, IGLOO, Accelerator, ProASIC, and newer families
Xilinx	Virtex-4 and newer 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	p: <i>portName</i>

The following table summarizes the syntax in different files:

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

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
```

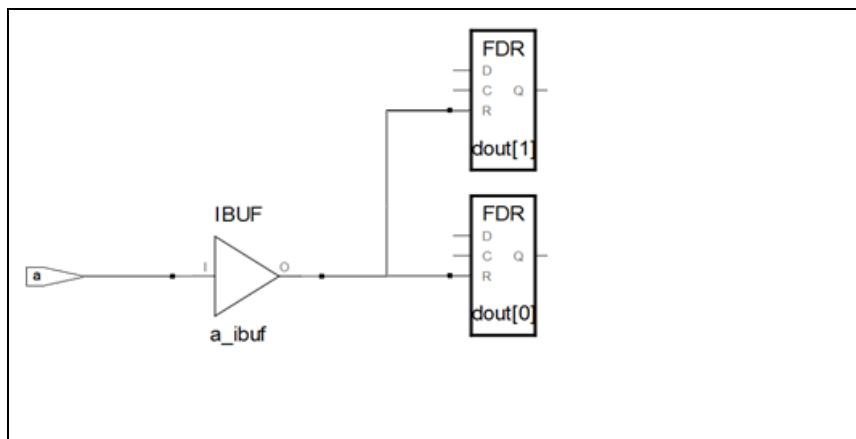
```
if (clk='1' and clk'event) then
    if (rst = '1') then
        dout <= (others => '0');
    else
        dout <= din;
    end if;
end if;
end process;
end beh;
```

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):

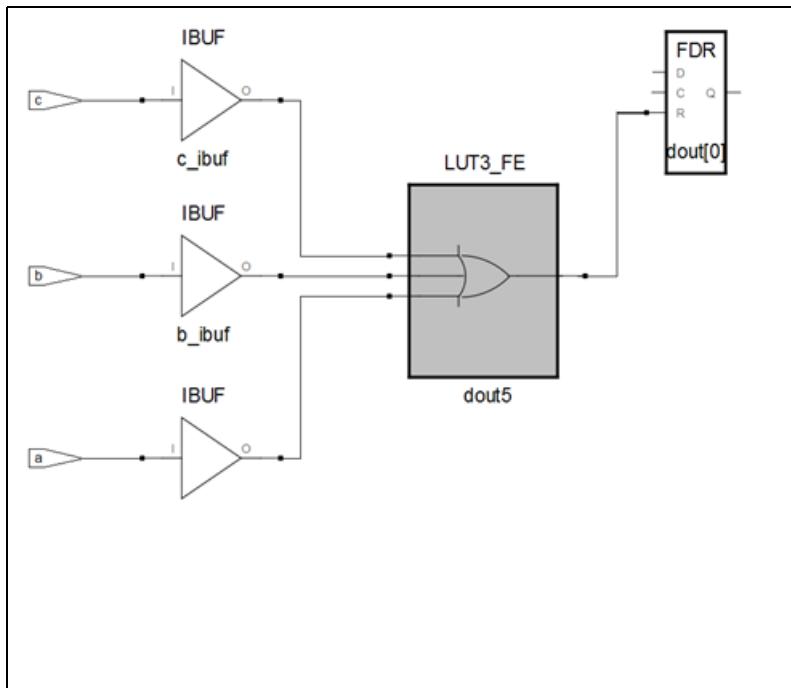
Verilog input a /*synthesis syn_direct_reset=1*/;

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	LIFMD, Platform Manager, ECP, SC, XP, MachXO and newer families
Microchip	PolarFire, RTG4, SmartFusion, Fusion, IGLOO, Accelerator, ProASIC, and newer families
Xilinx	Virtex-4 and newer 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	p: <i>portName</i>

The following table summarizes the syntax in different files:

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

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
```

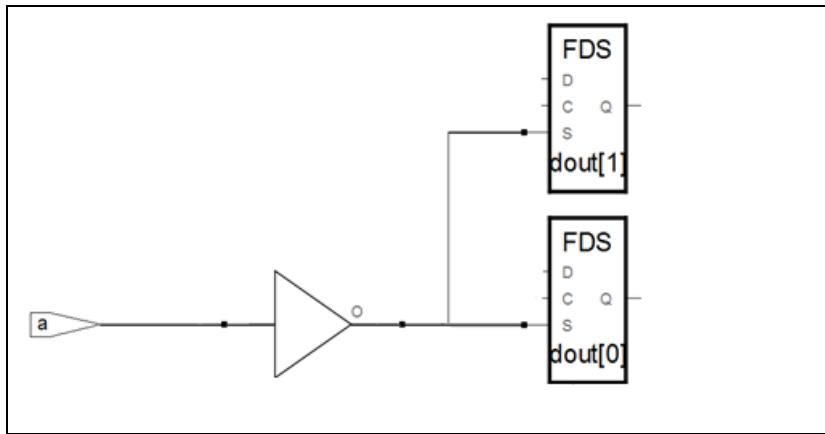
```
if (set = '1') then
    dout <= (others => '1');
else
    dout <= din;
end if;
end if;
end process;
end beh;
```

Effect of Using syn_direct_set

The following figure shows the attribute set to 1. The software assigns a net for the design to the dedicated set pin of a synchronous storage element (flip-flop):

Verilog input a /*synthesis syn_direct_set=1*/;

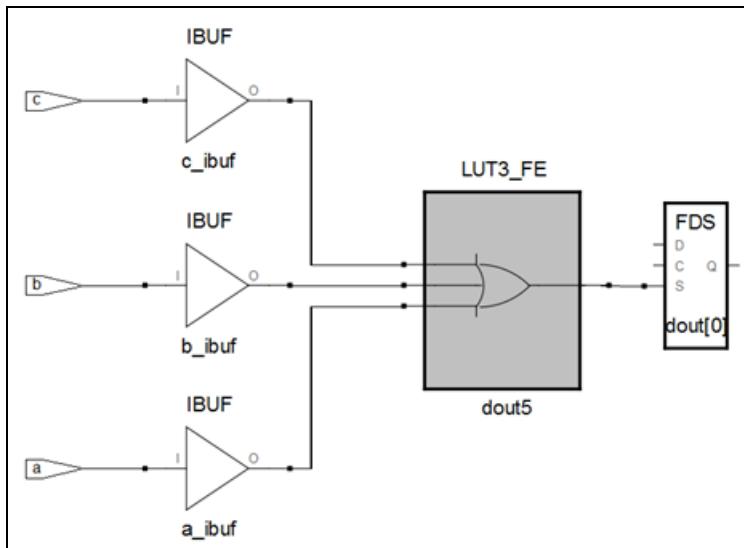
VHDL attribute syn_direct_set of a : signal is true;



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

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

VHDL attribute syn_direct_set of a : signal is false;



syn_disable_purifyclock

Attribute

Disables the purification of clocks.

Vendor	Technology	Synthesis Tool
Xilinx	Virtex-4 and newer families	Synplify Premier

Description

The `syn_disable_purifyclock` attribute disables purification of clocks. Clock purification is on by default. Clock purification duplicates nets that drive clock and data pins simultaneously, to ensure clock nets do not drive data pins. Currently, clock purification applies for generated clocks.

This attribute can be set globally or on a net.

Global Attribute	Object
Yes	Net

FDC	<code>define_attribute {object} syn_disable_purifyclock {1}</code>	FDC Example
Verilog	<code>object /* synthesis syn_disable_purifyclock = "{0 1}" */;</code>	Verilog Example
VHDL	<code>attribute syn_disable_purifyclock of object:objectType is true;</code>	VHDL Example

FDC Example

The constraint file syntax for the attribute is

```
define_attribute {object} syn_disable_purifyclock {1}
define_global_attribute syn_disable_purifyclock {1}
```

Where *object* is a net. For example:

```
n:clk1
```

Verilog Example

```
...
input wire clk,
input wire din_r,
output dout,dout_r
);
wire gen_clk /* synthesis syn_disable_purifyclock=1 */
/* synthesis syn_keep=1 */;
FD u0(.C(clk),.D(~gen_clk),.Q(gen_clk));
FD ud(.C(clk),.D(gen_clk),.Q(dout));
FD uc(.D(din_r),.Q(dout_r),.C(gen_clk));
```

VHDL Example

```
...
entity FF2LUT2FF is
port (
    clk      : in  std_logic;
    din_r   : in  std_logic;
    dout    : out std_logic;
    dout_r  : out std_logic
);
end;

architecture beh of FF2LUT2FF is
signal gen_clk : std_logic;
attribute syn_keep : boolean;
attribute syn_keep of gen_clk : signal is true;
attribute syn_disable_purifyclock : boolean;
attribute syn_disable_purifyclock of gen_clk : signal is true;

component FD
    generic (
        INIT : bit := '0'
    );

```

```

port (
    Q : out std_ulogic;
    C : in std_ulogic;
    D : in std_ulogic
);
end component;

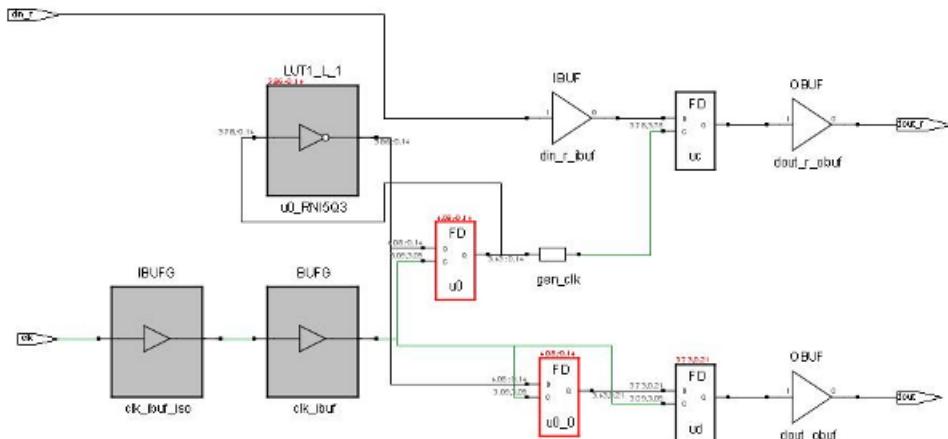
begin
    u0: FD port map(C=>clk,D=>not gen_clk,Q=>gen_clk);
    ud: FD port map(C=>clk,D =>gen_clk,Q=>dout);
    uc: FD port map(D=>din_r,Q=>dout_r,C=>gen_clk);

```

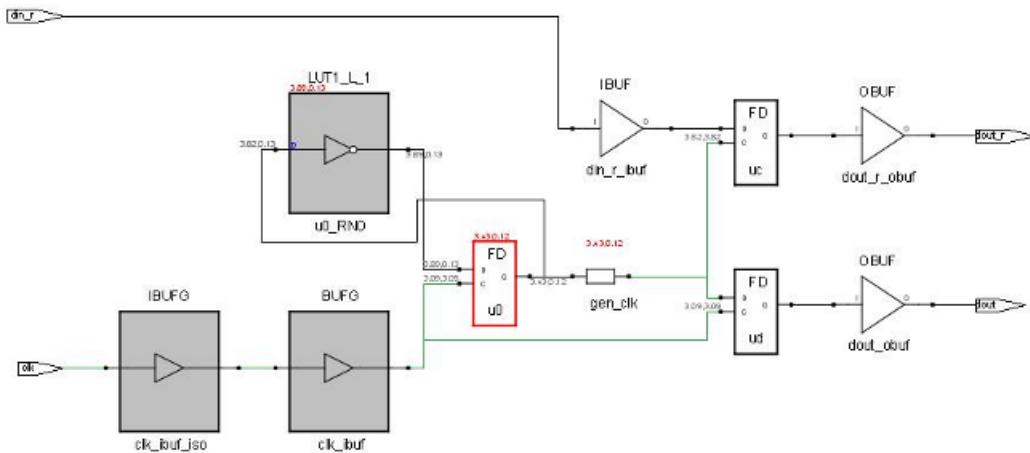
Use the `syn_keep` attribute on the nets along with this attribute to ensure that net names are preserved.

Effect of Using `syn_disable_purifyclock`

The following shows a design before applying the `syn_disable_purifyclock` attribute:



The following shows a design after applying the `syn_disable_purifyclock` attribute:



syn_donot_infer_addr

Attribute

Prevents the inference of ODDR in the design.

Vendor	Technology	Synthesis Tool
Xilinx	Virtex 5 and newer families	Synplify Pro, Synplify Premier

syn_donot_infer_addr values

1 | true Disables the software from inferring ODDR.

0 | false (Default) Enables the software to infer ODDR.

Description

The syn_donot_infer_addr attribute prevents the inference of ODDR. You can direct the mapper to enable or disable the inference of the ODDR. This attribute can be applied globally or to the module.

syn_donot_infer_addr Syntax

You can specify the attribute in the following files:

FDC `define_attribute {object} syn_donot_infer_addr {0|1}`
`define_global_attribute syn_donot_infer_addr {0|1}`

[FDC Example](#)

Verilog `object /* synthesis syn_donot_infer_addr = 0 | 1 */`

[Verilog Example](#)

FDC Example

```
define_attribute {v:work.ddr_input} syn_donot_infer_addr {1|0}
```

define_global_attribute {syn_donot_infer_addr} {1|0}

	Enable	Object Type	Object	Attribute	Value
1	<input checked="" type="checkbox"/>	global	<Global>	syn_donot_infer_addr	1
2					

Verilog Example

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

module ddr_input (input data_in, in1, in2, clk, rst,
output reg data_out) /*synthesis syn_donot_infer_addr=0*/;

reg data_in1_reg, data_in2_reg;
wire data_in1, data_in2;
reg parity= 1'b1;

always @ (posedge clk, posedge rst)
begin
    if (rst)
        q1 <= 1'b0;
    else
        q1 <= data_in;
end

assign data_in1 = q1 & q2;
assign data_in2 = in1 & in2;

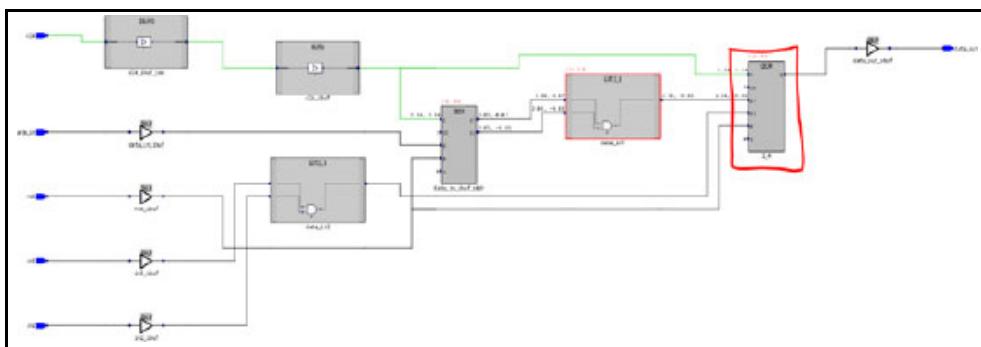
always @ (posedge clk, posedge rst)
begin
    if (rst)
        data_in1_reg <= 1'b0;
    else
        data_in1_reg <= data_in1;
end

always @ (negedge clk, posedge rst)
begin
    if (rst)
        data_in2_reg <= 1'b0;
    else
        data_in2_reg <= data_in2;
end

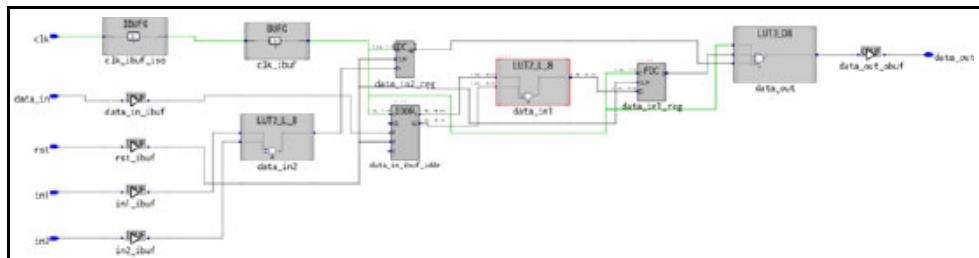
assign data_out = clk ? data_in1_reg : data_in2_reg;
endmodule
```

Effect of using syn_donot_infer_addr

The following figure shows when the attribute syn_donot_infer_addr is set to 0. The tool infers ODDR.



The following figure shows when the attribute `syn_donot_infer_oeaddr` is set to 1. This prevents the tool from inferring ODDR.



syn_donot_infer_iddr

Attribute

Prevents the inference of IDDR in the design.

Vendor	Technology	Synthesis Tool
Xilinx	Virtex 5 and newer families	Synplify Pro, Synplify Premier

syn_donot_infer_addr values

1 | true Disables the software from inferring IDDR.

0 | false (Default) Enables the software to infer IDDR.

Description

The syn_donot_infer_iddr attribute prevents the inference of IDDR. You can direct the mapper to enable or disable the inference of the IDDR. This attribute can be applied globally or to the module.

syn_donot_infer_iddr Syntax

You can specify the attribute in the following files:

FDC `define_attribute {object} syn_donot_infer_iddr {0|1}`
`define_global_attribute syn_donot_infer_iddr {0|1}`

[FDC Example](#)

Verilog `object /* synthesis syn_donot_infer_iddr = 0 | 1 */`

[Verilog Example](#)

FDC Example

```
define_attribute {v:work.ddr_input} syn_donot_infer_iddr {1|0}
```

```
define_global_attribute {syn_donot_infer_iddr} {1|0}
```

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	<Global>	syn_donot_infer_iddr	1		
2							

Verilog Example

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

module ddr_input (input data_in, in1, in2, clk, rst,
output reg data_out) /*synthesis syn_donot_infer_iddr=1*/;

reg q1, q2;
wire data_in1, data_in2;
reg data_in1_reg, data_in2_reg;

always @ (posedge clk, posedge rst)
begin
    if (rst)
        q1 <= 1'b0;
    else
        q1 <= data_in;
end

always @ (negedge clk, posedge rst)
begin
    if (rst)
        q2 <= 1'b0;
    else
        q2 <= data_in;
end

assign data_in1 = q1 & q2;
assign data_in2 = in1 & in2;

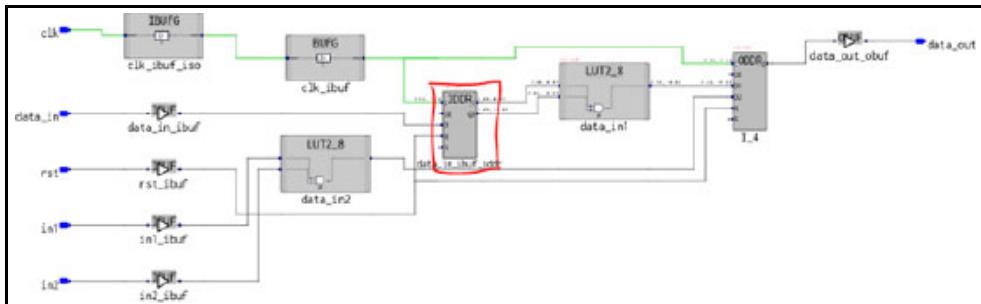
always @ (posedge clk, posedge rst)
begin
    if (rst)
        data_in1_reg <= 1'b0;
    else
        data_in1_reg <= data_in1;
end

always @ (negedge clk, posedge rst)
begin
    if (rst)
        data_in2_reg <= 1'b0;
    else
        data_in2_reg <= data_in2;
end

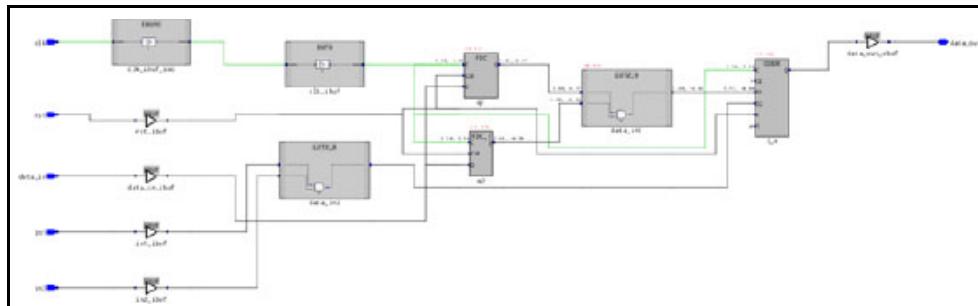
assign data_out = clk ? data_in1_reg : data_in2_reg;
endmodule
```

Effect of using syn_donot_infer_iddr

The following figure shows when the attribute syn_donot_infer_iddr is set to 0. The tool infers IDDR.



The following figure shows when the attribute syn_donot_infer_iddr is set to 1. This prevents the tool from inferring IDDR.



syn_dspstyle

Attribute

Determines whether the object is mapped to a technology-specific DSP component.

Vendor	Technologies
Achronix	Speedster7t
Lattice	iCE5LP
Xilinx	Spartan-2 and newer families Virtex and newer families

The `syn_dspstyle` options for designs with DSPs vary, according to the technology family selected. See the following for specific details:

- [syn_dspstyle for Achronix](#), on page 221
 - [Achronix syn_dspstyle Syntax](#), on page 221
 - [Achronix Verilog Example](#), on page 222
 - [Achronix VHDL Example](#), on page 223
- [syn_dspstyle for Lattice](#), on page 224
 - [Lattice syn_dspstyle Syntax](#), on page 224
 - [Lattice SCOPE Example](#), on page 224
 - [Lattice Verilog Example](#), on page 225
 - [Lattice VHDL Example](#), on page 225
- [syn_dspstyle for Xilinx](#), on page 227
 - [Xilinx syn_dspstyle Syntax](#), on page 228
 - [Xilinx Constraint Examples](#), on page 229
 - [Xilinx Verilog Example](#), on page 229

-
- [Xilinx VHDL Example, on page 229](#)
 - [Effect of Using syn_DSPStyle in a Xilinx Design, on page 230](#)
 - [syn_DSPStyle in Xilinx SIMD Mode, on page 233](#)

syn_DSPStyle Values

Value	Description
logic	Maps the object to logic instead of the DSP resources.
dsp	<p><i>Achronix</i></p> <p>Maps the specified object to MLP primitives. See syn_DSPStyle for Achronix , on page 221 for a description of how different objects are handled.</p> <p><i>Lattice</i></p> <p>Maps the multiplier as dedicated hardware blocks (SB_MAC16 blocks for iCE5LP devices)</p>
dsp48 (Default)	<p><i>Xilinx</i>.</p> <p>Maps the specified object to a DSP48. See syn_DSPStyle for Xilinx , on page 227 for a description of how different objects are handled.</p>
simd	<p><i>Xilinx Virtex 5 and later</i></p> <p>Specifies the DSP configuration available in Single Instruction Multiple Data (SIMD) mode, and leaves it to the synthesis software to determine how to combine adders and pack them into one DSP slice. See syn_DSPStyle in Xilinx SIMD Mode , on page 233 for details.</p>
simd: <i>i</i>	<p><i>Xilinx Virtex 5 and later</i></p> <p>Specifies the DSP configuration available in SIMD mode. <i>i</i> is an index value that is an integer less than or equal to 24. Specify an index value to pair and pack multiple adders into the same DSP slice. You can pack the following components:</p> <ul style="list-style-type: none"> • Two adders or subtractors with the same index value and functionality, with an input size less than or equal to 24 bits • Four adders or subtractors with the same index value and functionality, with an input size less than or equal to 12 bits. <p>See syn_DSPStyle in Xilinx SIMD Mode , on page 233 for details.</p>

syn_dspstyle for Achronix

For Achronix designs, the value of this attribute is either logic or dsp. The object you apply the attribute to affects what gets mapped into the DSP resource. The following table provides details:

Object	Behavior
Multiplier	<p>Setting syn_dspstyle to dsp maps the multiplier logic structures into the MLP primitives. You can apply this attribute to multipliers, which are packed into the MLP primitives component. By default, the tool implements MLP primitives when the input width of the multiplier is greater than 2-bits wide.</p> <p>When you set the attribute value to logic on the multiplier logical structures prevents the register from being mapped into the MLP primitives component.</p>
Register	<p>Setting syn_dspstyle to dsp maps the multiplier logic structures into the MLP primitives. You can apply this attribute to modules or registers, which are packed into the MLP primitives component. By default, the tool implements MLP primitives when the input width of the multiplier is greater than 2-bits wide.</p> <p>When you set the attribute value to logic on the multiplier logical structures prevents the register from being mapped into the MLP primitives component.</p>
View, Module / Architecture	Setting the attribute on a view, module, or architecture applies the attribute to the entire view. This attribute can be used globally.

Achronix syn_dspstyle Syntax

Global Support	Objects
Yes	<p>Multipliers, registers, views, modules, architectures</p> <p>See syn_dspstyle for Achronix , on page 221 for details.</p>

:

The following table shows how to specify the attribute in different files:

FDC	define_attribute {object} syn_DSPstyle {dsp logic} define_global_attribute syn_DSPstyle {dsp logic}	
Verilog	object /* synthesis syn_DSPstyle = "dsp logic" */;	Achronix Verilog Example
VHDL	attribute syn_DSPstyle of object : objectType is dsp logic;	Achronix VHDL Example

Achronix Verilog Example

```
module test (clk,reset,enable,a,b,q);
    input clk;
    input reset;
    input enable;
    input signed [27:0] a;
    input signed[27:0] b;
    reg signed [27:0] a_reg;
    reg signed[27:0] b_reg;
    output signed [55:0] q;
    wire signed [55:0] q1;
    reg signed [55:0] q/*synthesis syn_DSPstyle = logic*/;

    assign q1 = a_reg * b_reg; // Multiply

    always @(posedge clk)
    begin
        if (reset)
            begin
                a_reg <= 0;
                b_reg <= 0;
                q <= 0;
            end
        else if (enable)
            begin
                a_reg <= a;
                b_reg <= b;
                q <= q1 + q; // Accumulate
            end
    end
endmodule
```

Achronix VHDL Example

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

entity test is
generic (widtha : integer := 2;
          widthb : integer := 2;
          width_out : integer := 4
        );
port (clk : in std_logic;
       ina : in std_logic_vector(widtha-1 downto 0);
       inb : in std_logic_vector(widthb-1 downto 0);
       dout : out std_logic_vector(width_out-1 downto 0)
      );
end entity test;

architecture arc of test is
  signal dout_t : signed (width_out-1 downto 0);
  attribute syn_dspstyle : string ;
  attribute syn_dspstyle of dout_t : signal is "dsp";
begin

  dout_t <= signed(ina) * signed(inb);

  process (clk)
begin
  if(clk'event and clk='1')then
    dout <= CONV_STD_LOGIC_VECTOR(dout_t,4);
  end if;
end process;

end arc;
```

syn_dspstyle for Lattice

For Lattice designs, the value of this attribute is either logic or dsp. The object you apply the attribute to affects what gets mapped into the DSP resource. The following table provides details:

Object	Behavior
Multiplier	Setting syn_dspstyle to dsp maps the multiplier logic structures into the SB_MAC16. When you set the attribute value to logic on the multiplier logical structures prevents the register from being mapped into the SB_MAC16 component.
View, Module / Architecture	Setting the attribute on a view, module, or architecture applies the attribute to the entire view. This attribute can be used globally.

Lattice syn_dspstyle Syntax

Global Support	Object	
Yes	Module or instance	
FDC	define_attribute {instance} syn_dspstyle {DSP logic}	Lattice SCOPE Example
	Global attribute: define_global_attribute syn_dspstyle {DSP logic}	
Verilog	input net /* synthesis syn_dspstyle = "DSP logic" */;	Lattice Verilog Example
VHDL	attribute syn_dspstyle of instance : signal is "DSP logic";	Lattice VHDL Example

Lattice SCOPE Example

This example shows multipliers globally implemented as logic:

Current Design: <Top Level> 

	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 View

This example specifies that multipliers be implemented as logic.

```
define_attribute {temp[15:0]} syn_DSPStyle {logic}
```

Lattice 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
```

Lattice VHDL Example

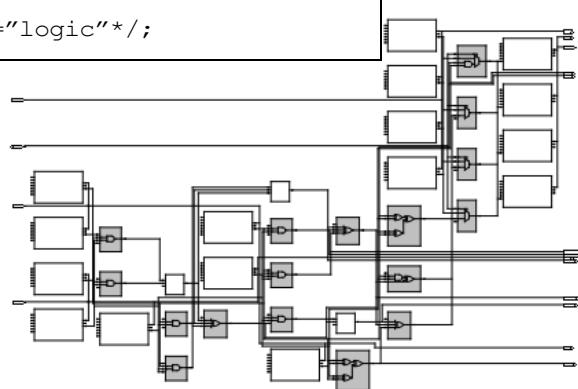
```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
entity mult is
port (
    clk : in std_logic;
    a : in std_logic_vector(7 downto 0);
    b : in std_logic_vector(7 downto 0);
    c : out std_logic_vector(15 downto 0)
);
end mult;
architecture rtl of mult is
signal mult_i : std_logic_vector(15 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;
```

Effect of Using syn_DSPstyle in a Lattice Design

In the example below, the syn_DSPstyle attribute is set to logic on the multiplier, so that it is implemented as logic in the design.

FDC	define attribute {temp[15:0]} syn_DSPstyle {logic}
Verilog	wire [15:0] temp /*synthesis syn_DSPstyle="logic"*/;



The diagram illustrates the internal logic implementation of a multiplier. It shows a complex network of logic cells (represented by small squares) and interconnecting wires. The input signals are processed through a series of logic stages, including AND gates, OR gates, and other combinational logic blocks, to produce the final output. This implementation is specifically for logic cells, as indicated by the Verilog code's use of the syn_DSPstyle attribute.

syn_dspstyle for Xilinx

This attribute determines if the object to which the attribute is applied is mapped to a Xilinx DSP48 component. The value of this attribute is either logic or dsp48. The object to which you apply this attribute determines what gets mapped into the DSP48.

Object	Behavior
Multiplier	<p>By default, multipliers are mapped to DSP48, so setting <code>syn_dspstyle</code> to <code>dsp48</code> on a multiplier is redundant. When a multiplier is mapped, the mapper also packs the adder/subtractor driven by the multiplier into the same DSP48 if possible.</p> <p>If your design has constants driving multiplier logic, the tool could perform constant propagation before DSP48 packing. Use <code>syn_keep</code> to prevent constant propagation optimizations.</p> <p>When you set the attribute value to <code>logic</code>, the mapper uses logic to implement the multiplier.</p> <p><code>syn_dspstyle</code> and <code>syn_multstyle</code>:</p> <ul style="list-style-type: none"> • Virtex-4, and newer technologies: For simple multipliers in these newer technologies, <code>syn_dspstyle</code> has the same effect as using <code>syn_multstyle</code>. For example, <code>syn_dspstyle = dsp48</code> results in the same implementation as <code>syn_multstyle = block_mult</code>. In both cases, the tool implements a block multiplier. • Older technologies: Use <code>syn_multstyle</code> for older technologies that do not support dedicated DSP resources. See syn_multstyle, on page 401.
Adder, Subtractor, Counter	<p>By default an adder/subtractor without a multiplier feeding it or a counter is mapped to logic. You do not need to explicitly set <code>syn_dspstyle=logic</code> on an adder/subtractor unless you want to decouple the mult-add or mult-subtract structure and map the multiplier into the DSP48 without the adder/subtractor.</p> <p>When the attribute is set to <code>dsp48</code>, the adder/subtractor or counter is mapped into the DSP48. If your design has constants driving multiplier logic, the tool could perform constant propagation before DSP48 packing. Use the <code>syn_keep</code> directive to prevent any constant propagation optimizations.</p> <p>By default, only one adder is packed into a single DSP48 slice. The Xilinx DSP structure supports two 24-bit adders and four 12-bit adders packed into a single DSP48 slice. For details, see syn_dspstyle in Xilinx SIMD Mode, on page 233.</p> <p>By default, the tool automatically maps counters with a large bit size that are not timing-critical to DSP48s. If you want to map these counters to logic, you must explicitly set <code>syn_dspstyle</code> to <code>logic</code>.</p>

Object	Behavior
Register	<p>When a register is associated with an adder/subtractor or multiplier, the mapper determines if the register needs to be mapped into the DSP48. Setting <code>syn_dspstyle</code> to <code>dsp48</code> has no effect.</p> <p>Setting <code>syn_dspstyle</code> to <code>logic</code> on a register prevents the register from being mapped into the DSP48. If you do not want to map a register to a DSP48, set <code>syn_dspstyle</code> to <code>logic</code> for the register. Note that when the attribute is attached to a register, it only applies to the register, not to the associated multipliers, adders, or other objects.</p> <p>You can set the SIMD values on a view.</p>
View, Module, Architecture	<p>Setting the attribute on a view, module, or architecture applies the attribute to the entire view.</p> <p>You can set the SIMD values on a view.</p>

Xilinx `syn_dspstyle` Syntax

Global Support	Valid Objects
Yes	<p>Multipliers, adders, subtractors, counters, registers, views, modules, architectures.</p> <p>See syn_dspstyle for Xilinx , on page 227 for details.</p>

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

FDC	<code>define_attribute {object} syn_dspstyle {dsp48 logic}</code> <code>define_global_attribute syn_dspstyle {dsp48 logic}</code>	Xilinx Constraint Examples
Verilog	<code>object /* synthesis syn_dspstyle = "dsp48 logic" */;</code>	Xilinx Verilog Example
VHDL	<code>attribute syn_dspstyle of object : objectType is dsp48 logic;</code>	Xilinx VHDL Example

See [VHDL Attribute and Directive Syntax, on page 401](#) for different ways to specify VHDL attributes and directives.

Xilinx Constraint Examples

	Enabled	Object Type	Object	Attribute	Value	Val Type
1	<input checked="" type="checkbox"/>	<any>	i:P_1[16:0]	syn_DSPStyle	dsp48	string
2	<input checked="" type="checkbox"/>	<any>	i:un1_b_d[15:0]	syn_DSPStyle	logic	string

```
define_global_attribute {syn_DSPstyle} {simd}
define_attribute {reg1} syn_DSPstyle {dsp48}
define_attribute {my_addr1} syn_DSPstyle {simd:1}
define_attribute {v:work.mult} syn_DSPstyle {logic}
```

Xilinx Verilog Example

```

module dsp_style (clk,A,B,PC,P);
input clk;
input [7:0] A,B,PC;
output reg [16:0] P;
reg [7:0]a_d,b_d;
reg [16:0] m /* synthesis syn_DSPstyle = "logic" */;

always @ (posedge clk) begin
    a_d <= A;
    b_d <= B;
    m   <= a_d * b_d;
    P   <= m + PC;
end

endmodule

```

See [Example 1 - Specifying SIMD](#), on page 234 and [Example 2 - Specifying a SIMD Index Value](#), on page 236 for Verilog examples of the attribute specified with SIMD mode values.

Xilinx VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
```

```

entity dsp_style is
port (clk : in std_logic;
      A : in std_logic_vector(7 downto 0);
      B : in std_logic_vector(7 downto 0);
      PC : in std_logic_vector(7 downto 0);
      P : out std_logic_vector(15 downto 0));
end dsp_style;

architecture rtl of dsp_style is
signal m : std_logic_vector(15 downto 0);
attribute syn_DSPstyle : string ;
attribute syn_DSPstyle of m : signal is "logic";
begin
  process(clk)
  begin
    if (clk'event and clk = '1') then
      m <= A * B;
      P <= m + PC;
    end if;
  end process;
end rtl;

```

Effect of Using syn_DSPstyle in a Xilinx Design

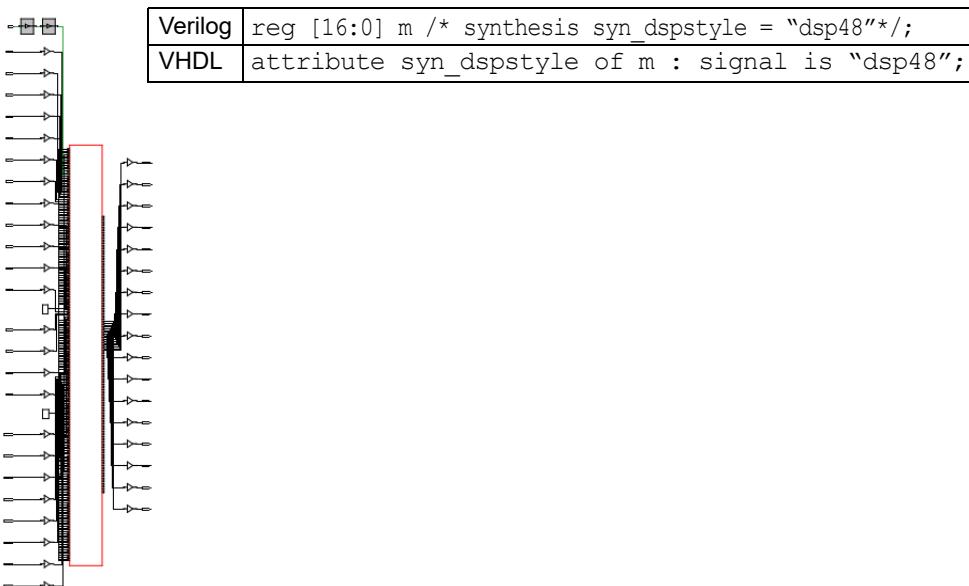
In the example below, the `syn_DSPstyle` attribute is set to logic globally on the module, so all four multipliers in the design are implemented as logic.

FDC	define_global_attribute syn_DSPstyle {logic}
Verilog	module dsp (a,b,c,d,e,f,g,h,r1,r2) /*synthesis syn_DSPstyle="logic"*/;

```
module dsp(a,b,c,d,e,f,g,h,r1,r2) /* synthesis
syn_dspstyle="logic" */;
input [7:0] a,b,c,d,e,f,g,h;
output [15:0] r1;
output [15:0] r2;
wire [15:0] temp1;
wire [15:0] temp2;
wire [15:0] temp3;
wire [15:0] temp4;
assign temp1 = a*b;
assign temp2 = c*d;
assign temp3 = e*f;
assign temp4 = g*h;
assign r1 = temp1 + temp2;
assign r2 = temp3 + temp4; endmodule
```

syn_dspstyle=dsp48

The following figure shows a multiplier, adder, and registers implemented in a DSP48 block, the default setting:



syn_dspstyle=logic

The next figure shows how the multipliers and adders are implemented when the attribute is set to logic:

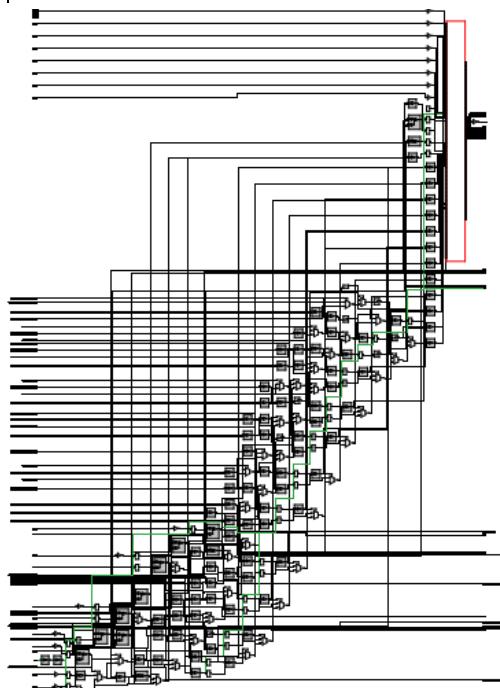
Verilog	reg [16:0] m /* synthesis syn_dspstyle = "logic" */;
VHDL	attribute syn_dspstyle of m : signal is "logic";



Mixed Values for syn_dspstyle

The following figure shows the implementation when `syn_dspstyle` is set to logic for the multiplier and `dsp48` for the adder:

Verilog	<code>output reg [16:0] P /* synthesis syn_dspstyle = "dsp48"*/;</code> <code>reg [16:0] m /* synthesis syn_dspstyle = "logic"*/;</code>
VHDL	<code>attribute syn_dspstyle of P : signal is "dsp48";</code> <code>attribute syn_dspstyle of m : signal is "logic";</code>



syn_dspstyle in Xilinx SIMD Mode

Single Instruction Multiple Data (SIMD) mode is supported for newer Xilinx devices. By default, only one adder is packed into a single DSP slice. In SIMD mode, the Xilinx DSP structure can support two 24-bit adders and four 12-bit adders in one DSP48 slice. To implement this DSP configuration, use the `syn_dspstyle` attribute with one of the SIMD mode values, as described in

syn_DSPStyle Values, on page 220. There are some limitations to DSP packing in SIMD mode; see *Limitations with SIMD Mode Packing*, on page 234 for details.

The following Verilog syntax shows the same index set on two adders to map them into one DSP slice:

```
wire [9:0] my_add1 /* synthesis syn_DSPStyle = simd:1 */;
wire [9:0] my_add2 /* synthesis syn_DSPStyle = simd:1 */;
```

See the examples below for more detail.

Limitations with SIMD Mode Packing

DSP-packing in SIMD mode has the following limitations:

- Accumulators are packed only if they have the same reset, enable, and clock pins for the output registers.
- Add-Sub functions are packed only if they have the same select pin for dynamically choosing the adder or subtractor.
- Add-Mux functions are not supported with SIMD mode.

Example 1 - Specifying SIMD

The following example shows four 12-bit adders specified with SIMD mapped to one DSP48 block.

```
module test(a_ip, b_ip, c_ip, d_ip, e_ip, f_ip, g_ip, h_ip,
dout_op0, dout_op1, dout_op2, dout_op3);

    //// Input declaration
    input [10 : 0] a_ip;
    input [10 : 0] b_ip;
    input [10 : 0] c_ip;
    input [10 : 0] d_ip;
    input [10 : 0] e_ip;
    input [10 : 0] f_ip;
    input [10 : 0] g_ip;
    input [10 : 0] h_ip;

    //// Output declaration
    output [11 : 0] dout_op0/* synthesis syn_DSPStyle="simd" */;
    output [11 : 0] dout_op1/* synthesis syn_DSPStyle="simd" */;
    output [11 : 0] dout_op2/* synthesis syn_DSPStyle="simd" */;
```

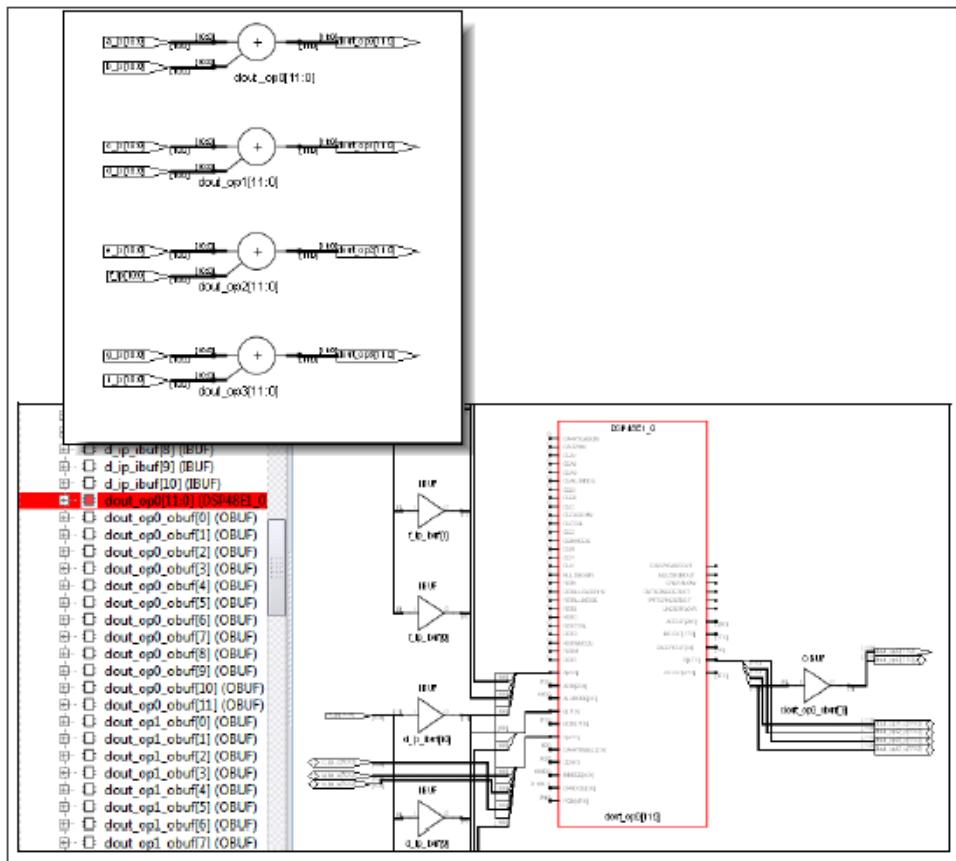
```

output [11 : 0] dout_op3/* synthesis syn_DSPstyle="simd" */;

assign dout_op0 = a_ip + b_ip;
assign dout_op1 = c_ip + d_ip;
assign dout_op2 = e_ip + f_ip;
assign dout_op3 = g_ip + h_ip;

endmodule

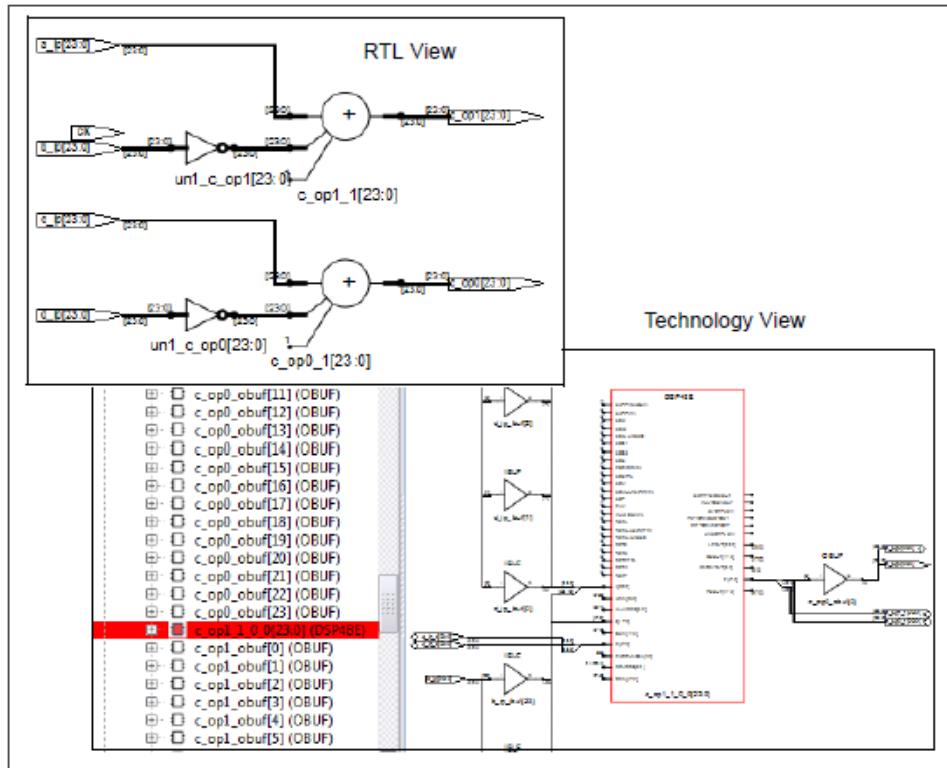
```



Example 2 - Specifying a SIMD Index Value

The following example shows two 24-bit adders specified with a SIMD index value mapped to one DSP48 block.

```
module test(clk,a_ip,b_ip,c_ip,d_ip,c_op0,c_op1);  
  
    //// Input declaration  
    input clk;  
    input signed [23 : 0] a_ip;  
    input signed [23 : 0] b_ip;  
    input signed [23 : 0] c_ip;  
    input signed [23 : 0] d_ip;  
  
    //Output declaration  
    output signed [23 : 0] c_op0/  
        * synthesis syn_dspstyle = "simd:1" */;  
    output signed [23 : 0] c_op1  
        * synthesis syn_dspstyle = "simd:1" */;  
  
    assign c_op0 = a_ip - b_ip;  
    assign c_op1 = c_ip - d_ip;  
  
endmodule
```



syn_edif_name_length

Attribute

Determines the length of port names written to the output file for Intel FPGA designs.

Vendor Technology

Intel All
FPGA

syn_edif_name_length Values

Value	Description
Restricted	Use with Max+Plus II.
Unrestricted	Use with Quartus II.

Description

Global truncation attribute that determines the length of port names written to the output file for Intel FPGA designs. If your target place-and-route tool is Quartus II, the length of the port name does not matter.

You specify this attribute globally on the top-level module or architecture. The value of the attribute can be either restricted or unrestricted. The restricted setting (the default) limits the port names to 30 characters. Use this setting with Max+Plus II. If you set the attribute to unrestricted, the port names are not truncated. Use this value if you are targeting Quartus II as the place-and-route tool.

syn_edif_name_length Syntax

Global Support Object

Yes Top-level module or architecture.

The following table summarizes the syntax in different files:

FDC	define_global_attribute syn_edif_name_length {restricted unrestricted}	FDC Example
Verilog	object /* synthesis syn_edif_name_length = "restricted unrestricted" */;	Verilog Example
VHDL	attribute syn_edif_name_length of object:objectType is "restricted unrestricted";	VHDL Example

FDC Example

Enable	Object Type	Object	Attribute	Value	Value Type	Description
<input checked="" type="checkbox"/>	<any>	<Global>	syn_edif_name_length	Unrestrict...	string	Use Restricted for MAXII; Unrestricted for quartus

Verilog Example

Attach the attribute to the top-level module. The following is an example:

```
module test(in1, in2, out)
    /* synthesis syn_edif_name_length = "restricted" */;
input in1, in2;
output out;
testbox inst1(.in1(in1),.in2(in2),
    .abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstuvwxyz(out));
endmodule
```

:

VHDL Example

attribute syn_edif_name_length of object:objectType is "restricted | unrestricted";

Attach the attribute to the top-level entity, as shown in this example:

```
architecture beh of test is
component testbox is port(
  in1, in2 : in std_logic;
  abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstuvwxyz : out
    std_logic);
end component;

attribute syn_black_box : boolean;
attribute syn_black_box of testbox : component is true;
attribute syn_edif_name_length : string;
attribute syn_edif_name_length of beh :
  architecture is "Unrestricted";
begin
  u0: testbox port map (in1 => in1, in2 => in2,
    abcdefghijklmnopqrstuvwxyz_abcdefghijklmnopqrstuvwxyz
    => out1);
end beh;
```

See [VHDL Attribute and Directive Syntax, on page 401](#) for different ways to specify VHDL attributes and directives.

syn_edif_scalar_format

Attribute

Helps change the case of the object name in an edif file automatically.

Vendor	Technology
Xilinx	All

syn_edif_scalar_format Values

Value	Description
%n (Default)	Preserves case in the names of edif objects
%u	Makes the names of edif objects to uppercase
%d	Makes the names of edif objects to lowercase

Description

Global character formatting attribute to use with scalar port names to up-shift (%u) or down-shift (%d) the character case of the name in the EDIF output file. The default (`syn_edif_scalar_format %n`) preserves the character case of the names as they appear in source code. To change the character formatting and style of vector (bus) signals and ports, see [syn_edif_bit_format](#).

syn_edif_scalar_format Syntax

Global Support	Object
Yes	Verilog module or VHDL architecture

The following table summarizes the syntax in different files:

FDC	define_global_attribute syn_edif_scalar_format {%u %d %n}
Verilog	<i>object</i> /*synthesis syn_edif_scalar_format = "value" */;
VHDL	attribute syn_edif_scalar_format of <i>object</i> : <i>objectType</i> : architecture is "value";

Constraint Example

Enabled	Object Type	Object	Attribute	Value	Val Type	Description	Comment
1 <input checked="" type="checkbox"/>	global	<global>	syn_edif_scalar_format	%u	string	Format scalar names	

Verilog Example

```
module edif_case(aBcd,clk,q) /*synthesis syn_edif_scalar_format =
" %u" */;
  input aBcd,clk;
  output reg q;
  always@(posedge clk)
    q<=aBcd;
endmodule
```

VHDL Example

```
library IEEE;
use IEEE.std_logic_1164.all;
entity d_ff_srss is
port (
  aBcd, clk: in bit;
  q : out bit);
end d_ff_srss;
architecture d_ff of d_ff_srss is
attribute syn_edif_scalar_format : string;
attribute syn_edif_scalar_format of d_ff : architecture is "%u";
begin
process(clk)
begin
  if clk'event and clk='1' then
    q <= aBcd;
  end if;
end process;
end d_ff;
```

Effect of Using syn_edif_scalar_format

The following table shows a Xilinx design before applying the `syn_edif_scalar_format` attribute:

Verilog	<pre>module edif_case(aBcd,clk,q) /*synthesis syn_edif_scalar_format="%n"*/;</pre>
VHDL	<pre>attribute syn_edif_scalar_format : string; attribute syn_edif_scalar_format of d_ff : architecture is "%n";</pre>
	<pre>(library work (edifLevel 0) (technology (numberDefinition)) (cell edif case (cellType GENERIC) (view verilog (viewType NETLIST) (interface (port aBcd (direction INPUT)) (port clk (direction INPUT) (port q (direction OUTPUT)) (net aBcd_c (joined (portRef O (instanceRef aBcd_ibuf)) (portRef D (instanceRef qZ0)))))) (net (rename abcd "aBcd") (joined (portRef aBcd) (portRef I (instanceRef aBcd_ibuf)))))) (net clk_c (joined (portRef O (instanceRef clk_ibuf)) (portRef C (instanceRef qZ0)))))) (net clk (joined (portRef clk) (portRef I (instanceRef clk_ibuf_iso)))))) (net q_c (joined (portRef Q (instanceRef qZ0)) (portRef I (instanceRef q_obuf)))))) (net q (joined (portRef O (instanceRef q_obuf)) (portRef q))))))</pre>

```

        (net (rename clk_ibuf_isoz0 "clk_ibuf_iso") (joined
        (portRef O (instanceRef clk_ibuf_iso))
        (portRef I (instanceRef clk_ibuf)))
      ))
    )
  (property mapper_option (string ""))
)

```

The following table shows a Xilinx design after applying the `syn_edif_scalar_format` attribute with the `%u` value:

Verilog module edif_case(aBcd,clk,q) /*synthesis syn_edif_scalar_format="%u"*/;

VHDL attribute syn_edif_scalar_format : string;
attribute syn_edif_scalar_format of d_ff : architecture is "%u";

```

library work
(edifLevel 0)
(technology (numberDefinition))
(cell edif_case (cellType GENERIC)
  (view verilog (viewType NETLIST)
    (interface
      (port ABCD (direction INPUT))
      (port CLK (direction INPUT))
      (port Q (direction OUTPUT))

      (net (rename aBcd_c "ABCD_C") (joined
        (portRef O (instanceRef aBcd_ibuf))
        (portRef D (instanceRef qZ0)))
      ))
    (net (rename abcd "ABCD") (joined
      (portRef ABCD)
      (portRef I (instanceRef aBcd_ibuf)))
    ))
    (net (rename clk_c "CLK_C") (joined
      (portRef O (instanceRef clk_ibuf))
      (portRef C (instanceRef qZ0)))
    ))
    (net (rename clk "CLK") (joined
      (portRef CLK)
      (portRef I (instanceRef clk_ibuf_iso)))
    ))
    (net (rename q_c "Q_C") (joined
      (portRef Q (instanceRef qZ0)))
  )

```

```

    (portRef I (instanceRef q_obuf))
  ))
  (net (rename q "Q") (joined
    (portRef O (instanceRef q_obuf))
    (portRef Q)
  ))
  (net (rename clk_ibuf_isoz0 "CLK_IBUF_ISO") (joined
    (portRef O (instanceRef clk_ibuf_iso))
    (portRef I (instanceRef clk_ibuf))
  ))
)
)
(property mapper_option (string ""))
)

```

The following table shows a Xilinx design after applying the `syn_edif_scalar-format` attribute with the `%d` value:

Verilog	module edif_case(aBcd,clk,q) /*synthesis syn_edif_scalar_format="%d"*/;
VHDL	attribute syn_edif_scalar_format : string; attribute syn_edif_scalar_format of d_ff : architecture is " %d";

```

(library work
  (edifLevel 0)
  (technology (numberDefinition))
  (cell edif_case (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port abcd (direction INPUT))
        (port clk (direction INPUT))
        (port q (direction OUTPUT)))
      (net (rename aBcd_c "abcd_c") (joined
        (portRef O (instanceRef aBcd_ibuf))
        (portRef D (instanceRef qZ0)))
      )))
      (net abcd (joined
        (portRef abcd)
        (portRef I (instanceRef aBcd_ibuf)))
    )
    (net clk_c (joined
      (portRef O (instanceRef clk_ibuf))
      (portRef C (instanceRef qZ0)))
  )))

```

```
:  
  
        (net clk (joined  
        (portRef clk)  
        (portRef I (instanceRef clk_ibuf_iso))  
    ))  
        (net q_c (joined  
        (portRef Q (instanceRef qZ0))  
        (portRef I (instanceRef q_obuf))  
    ))  
        (net q (joined  
        (portRef O (instanceRef q_obuf))  
        (portRef q)  
    ))  
        (net (rename clk_ibuf_isoz0 "clk_ibuf_iso") (joined  
        (portRef O (instanceRef clk_ibuf_iso))  
        (portRef I (instanceRef clk_ibuf)))  
    ))  
)  
(property mapper_option (string ""))  
)
```

Global Support

If a design has more than one edif netlist, apply this attribute only on the top level netlist. It has the same impact as when it is applied on individual netlists. For example, if two modules have been instantiated, the attribute should be applied only on the top-level module.

Verilog Example

```
module mux(out, a, b, sel); // mux  
output out;  
input a, b;  
input sel;  
assign out = sel ? a : b;  
endmodule  
  
module reg8(q, data, clk, rst); // register  
output q;  
input data;  
input clk, rst;  
reg q;  
always @ (posedge clk or posedge rst)  
begin  
if (rst)
```

```

q = 0;
else
q = data;
end
endmodule

module top1(q1, a, b, sel, clk, rst)/*synthesis
syn_edif_scalar_format="%u"*/;
output q1;
input a, b;
input sel, clk, rst;
wire mux_out;
mux mux_1 (mux_out, a, b, sel);
reg8 reg8_1 (q1, mux_out, clk, rst);
endmodule

```

The following table shows a Xilinx design with the `syn_edif_scalar_format` attribute applied to the top-level module.

FDC File	<code>define_global_attribute {syn_edif_scalar_format} {%u}</code>
----------	--

FDC File	<code>module top1(q1, a, b, sel, clk, rst)/*synthesis syn_edif_scalar_format="%u"*/;</code>
----------	---

(library work (edifLevel 0) (technology (numberDefinition)) (cell reg8 (cellType GENERIC) (view netlist (viewType NETLIST) (interface (port RST_C (direction INPUT)) (port CLK_C (direction INPUT)) (port MUX_OUT (direction INPUT)) (port Q1_C (direction OUTPUT))) (contents (instance q (viewRef PRIM (cellRef FDC (libraryRef UNILIB)))) (net (rename q1_c "Q1_C") (joined (portRef Q (instanceRef q)) (portRef Q1_C))) (net (rename mux_out "MUX_OUT") (joined (portRef MUX_OUT) (portRef D (instanceRef q)))

```
)  
    (net (rename clk_c "CLK_C") (joined  
        (portRef CLK_C)  
        (portRef C (instanceRef q))  
    ))  
    (net (rename rst_c "RST_C") (joined  
        (portRef RST_C)  
        (portRef CLR (instanceRef q))  
    ))  
    )  
)  
(cell mux (cellType GENERIC)  
(view netlist (viewType NETLIST)  
    (interface  
        (port MUX_OUT (direction OUTPUT))  
        (port SEL_C (direction INPUT))  
        (port B_C (direction INPUT))  
        (port A_C (direction INPUT))  
    )  
(contents  
    (instance out (viewRef PRIM (cellRef LUT3_L  
        (libraryRef VIRTEX)))  
    (property INIT (string "8'hAC"))  
    )  
    (net (rename a_c "A_C") (joined  
        (portRef A_C)  
        (portRef I0 (instanceRef out))  
    ))  
    (net (rename b_c "B_C") (joined  
        (portRef B_C)  
        (portRef I1 (instanceRef out))  
    ))  
    (net (rename sel_c "SEL_C") (joined  
        (portRef SEL_C)  
        (portRef I2 (instanceRef out))  
    ))  
    (net (rename mux_out "MUX_OUT") (joined  
        (portRef LO (instanceRef out))  
        (portRef MUX_OUT)  
    ))  
)  
)  
(cell top1 (cellType GENERIC)  
(view verilog (viewType NETLIST)
```

```
(interface
  (port Q1 (direction OUTPUT))
  (port A (direction INPUT))
  (port B (direction INPUT))
  (port SEL (direction INPUT))
  (port CLK (direction INPUT)
  )
  (port RST (direction INPUT))
  )
(contents
  (instance clk_ibuf (viewRef PRIM
    (cellRef BUFG (libraryRef VIRTEX)
    (instance clk_ibuf_iso (viewRef PRIM
      (cellRef IBUFG (libraryRef VIRTEX)))
    )
    (instance q1_obuf (viewRef PRIM
      (cellRef OBUF (libraryRef VIRTEX)))
    )
    (instance rst_ibuf (viewRef PRIM
      (cellRef IBUF (libraryRef VIRTEX)))
    )
    (instance sel_ibuf (viewRef PRIM
      (cellRef IBUF (libraryRef VIRTEX)))
    )
    (instance b_ibuf (viewRef PRIM
      (cellRef IBUF (libraryRef VIRTEX)))
    )
    (instance a_ibuf (viewRef PRIM (cellRef IBUF
      (libraryRef VIRTEX)))
    )
    (instance mux_1 (viewRef netlist (cellRef mux))
    )
    (instance reg8_1 (viewRef netlist (cellRef reg8))
    )
    (net (rename mux_out "MUX_OUT") (joined
      (portRef MUX_OUT (instanceRef mux_1))
      (portRef MUX_OUT (instanceRef reg8_1))
    ))
    (net (rename a_c "A_C") (joined
      (portRef O (instanceRef a_ibuf))
      (portRef A_C (instanceRef mux_1))
    ))
    (net (rename a "A") (joined
      (portRef A)
      (portRef I (instanceRef a_ibuf))
    )))
  ))
```

```
:  
  
        (net (rename b_c "B_C") (joined  
        (portRef O (instanceRef b_ibuf))  
        (portRef B_C (instanceRef mux_1))  
    ))  
        (net (rename b "B") (joined  
        (portRef B)  
        (portRef I (instanceRef b_ibuf))  
    ))  
        (net (rename sel_c "SEL_C") (joined  
        (portRef O (instanceRef sel_ibuf))  
        (portRef SEL_C (instanceRef mux_1))  
    ))  
        (net (rename sel "SEL") (joined  
        (portRef SEL)  
        (portRef I (instanceRef sel_ibuf))  
    ))  
        (net (rename clk_c "CLK_C") (joined  
        (portRef O (instanceRef clk_ibuf))  
        (portRef CLK_C (instanceRef reg8_1))  
    ))  
        (net (rename clk "CLK") (joined  
        (portRef CLK)  
        (portRef I (instanceRef clk_ibuf_iso))  
    ))  
        (net (rename rst_c "RST_C") (joined  
        (portRef O (instanceRef rst_ibuf))  
        (portRef RST_C (instanceRef reg8_1))  
    ))  
        (net (rename rst "RST") (joined  
        (portRef RST)  
        (portRef I (instanceRef rst_ibuf))  
    ))  
        (net (rename q1_c "Q1_C") (joined  
        (portRef Q1_C (instanceRef reg8_1))  
        (portRef I (instanceRef q1obuf))  
    ))  
        (net (rename q1 "Q1") (joined  
        (portRef O (instanceRef q1obuf))  
        (portRef Q1)  
    ))  
        (net (rename clk_ibuf_isoz0 "CLK_IBUF_ISO") (joined  
        (portRef O (instanceRef clk_ibuf_iso))  
        (portRef I (instanceRef clk_ibuf))  
    ))  
    )  
    (property mapper_option (string ""))
```


syn_edif_bit_format

Attribute

Changes or preserves the naming style of bus ports in the EDIF output file.

Vendor	Devices
Xilinx	All

syn_edif_bit_format Values

Language	Default	Global	Object
Verilog	%n[%i]	Yes	Module/Architecture
VHDL	%n(%i)	Yes	Module/Architecture

Description

syn_edif_bit_format is a character formatting attribute for vector (bus) port names that controls their naming style in the EDIF output netlist file. The first argument (%n, %d, or %u) controls the case of bus names, and the second argument (%i) controls the appearance of vector ranges. The default preserves the character case of names as they appear in source code with angle brackets (Verilog) or parentheses (VHDL) delineating the vector range. To change the character formatting of scalar ports, see [syn_edif_scalar_format, on page 243](#).

syn_edif_bit_format Syntax

SCOPE	<code>define_global_attribute syn_edif_bit_format {[%n %u %d] [_string]{(%i) [%i] <%i>}}</code>	SCOPE Example
Verilog	<code>object /* synthesis syn_edif_bit_format = "[%n %u %d] [_string] {(%i) [%i] <%i>}" */;</code>	Verilog Example
VHDL	<code>attribute syn_edif_bit_format of object : objectType is "[%n %u %d] [_string] {(%i) [%i] <%i>}";</code>	VHDL Example

Value	Description
<code>%n(%i) %n[%i] %n<%i></code>	Preserves the name case (%n); parenthesis ((%i)), square brackets ([%i]), or angle brackets (<%i>) used as bus delimiters.
<code>%u(%i) %u[%i] %u<%i></code>	Shifts names to upper case (%u); parenthesis ((%i)), square brackets ([%i]), or angle brackets (<%i>) used as bus delimiters.
<code>%d(%i) %d[%i] %d<%i></code>	Shifts names to lower case (%d); parenthesis ((%i)), square brackets ([%i]), or angle brackets (<%i>) used as bus delimiters.

In addition to the above syntax definitions, %n, %u, and %d accept a *_string* argument to append a character string to the end of bus names.

SCOPE Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	global	<Global>	syn_edif_bit_format	%u<%i>	string	Format bus names

Verilog Example

```
module edif_bit_format(aBcd, clk, rst, q)
    /* synthesis syn_edif_bit_format ="%u<%i>" */;
    input[31:0]aBcd;
    input clk,rst;
    output reg[31:0]q;
```

```
always@ (posedge clk)
begin
    if(rst)
        q<=0;
    else
        q<=aBcd;
end
endmodule
```

VHDL Example

```
library IEEE;
use IEEE.std_logic_1164.all;

entity d_ff_srss is
    port (aBcd: in std_logic_vector(31 downto 0);
          clk: in std_logic;
          q: out std_logic_vector(31 downto 0));
end d_ff_srss;

architecture d_ff of d_ff_srss is
attribute syn_edif_bit_format : string;
attribute syn_edif_bit_format of d_ff : architecture is "%u<%i>";
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            q <= aBcd;
        end if;
    end process;
end d_ff;
```

Effect of Using `syn_edif_bit_format` Attribute

The following examples illustrate the effects of using the `syn_edif_bit_format` attribute on the EDIF output netlist.

Example - Changing Bus Name Case and Delimiter

VHD Source	input[31:0]aBcd; (Verilog) port (aBcd: in std_logic_vector(31 downto 0); (VHDL)
Attribute Value	% u<% i> Converts bus names to all upper case (%u) and uses angle brackets for bus delimiters (<%>).
EDIF Output	(port (array (rename aBcd "ABCD<31:0>") 32) (direction INPUT))

Example - Appending Bus Name String

VHD Source	input[31:0]aBcd; (Verilog) port (aBcd: in std_logic_vector(31 downto 0); (VHDL)
Attribute Value	% d_sub2A[% i] Converts names to all lower case (%d), appends string (_s2A) to bus names, and uses square brackets for bus delimiters ([%]).
EDIF Output	(port (array (rename aBcd "abcd_s2a[31:0]") 32) (direction INPUT))

syn_encoding

Attribute

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

Vendor	Devices
Achronix	Speedster
Intel FPGA	Stratix 10, Agilex, Stratix V, Cyclone V, newer devices
Lattice	ECP3, ECP4, newer devices
Microchip	ProASIC3, Fusion, SmartFusion2, newer devices
Xilinx	Virtex 6, Spartan6, newer 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 onehot is not a simple encoding (more than one bit can be set), the value must be decoded to determine the state. This encoding style can be slower than a gray style if you have a large output decoder following a state machine.</p>

Value	Description
gray	<p>More than one of the state registers can be hot. The synthesis tool attempts to have only one bit of the state registers change at a time, but it can allow more than one bit to change, depending upon certain conditions for optimization. For example:</p>
	000, 001, 011, 010, 110
	<p>Because gray is not a simple encoding (more than one bit can be set), the value must be decoded to determine the state. This encoding style can be faster than a onehot style if you have a large output decoder following a state machine.</p>
sequential	<p>More than one bit of the state register can be hot. The synthesis tool makes no attempt at limiting the number of bits that can change at a time. For example:</p>
	000, 001, 010, 011, 100
	<p>This is one of the smallest encoding styles, so it is often used when area is a concern. Because more than one bit can be set (1), the value must be decoded to determine the state. This encoding style can be faster than a onehot style if you have a large output decoder following a state machine.</p>
safe	<p>safe – 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.</p>
	<p>This value can be used in combination with any of the other encoding styles described above. You specify safe before the encoding style. The safe value is only valid for a state register, in conjunction with an encoding style specification.</p>
	<ul style="list-style-type: none"> For example, if the default encoding is onehot and the state machine reaches a state where all the bits are 0, which is an invalid state, the safe value ensures that the state machine is reset to a valid state. If recovery from an invalid state is a concern, it may be appropriate to use this encoding style, in conjunction with onehot, sequential or gray, in order to force the state machine to reset. When you specify safe, the state machine can be reset from an unknown state to its reset state. If an FSM with asynchronous reset is specified with the value safe and you do not want the additional recovery logic (flip-flop on the inactive clock edge) inserted for this FSM, then use the syn_shift_resetphase attribute to remove it. See syn_shift_resetphase, on page 613 for details.
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. The default encoding is determined by the number of states, except in the case of Xilinx designs, where a non-default encoding is implemented if it produces better results. 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, such as enumerated data types, use the `syn_enum_encoding` directive. If you use `syn_encoding` instead of `syn_enum_encoding`, you get a warning message (CD721). See [Comparison of syn_encoding and syn_enum_encoding, on page 270](#) for a comparison summary.

The tool honors `enum_encoding` attributes, but uses the value of `syn_encoding` instead if that is specified. See [syn_enum_encoding, enum_encoding, and syn_encoding, on page 270](#) 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. In the Synplify Pro and Synplify Premier tools, 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 643](#).
- For VHDL designs, see [Comparison of syn_encoding and syn_enum_encoding, on page 270](#) for comparative usage information.

Syntax Specification

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>	SCOPE Example
Verilog	<code>Object /* synthesis syn_encoding = "value" */;</code>	Verilog Example
VHDL	<code>attribute syn_encoding of object: objectType is "value";</code>	VHDL Example

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;

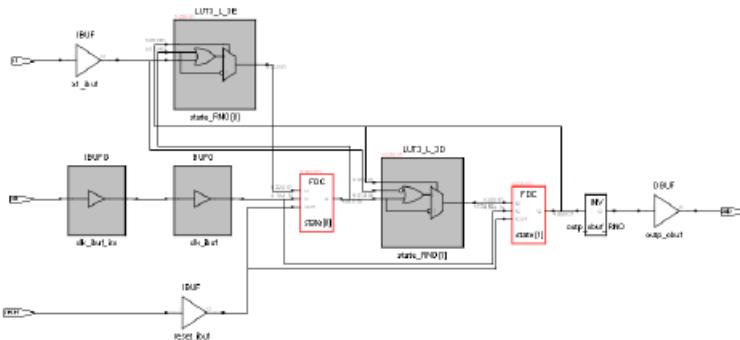
```

See [VHDL Attribute and Directive Syntax](#), on page 401 for different ways to specify VHDL attributes and directives.

Effect of Using syn_encoding

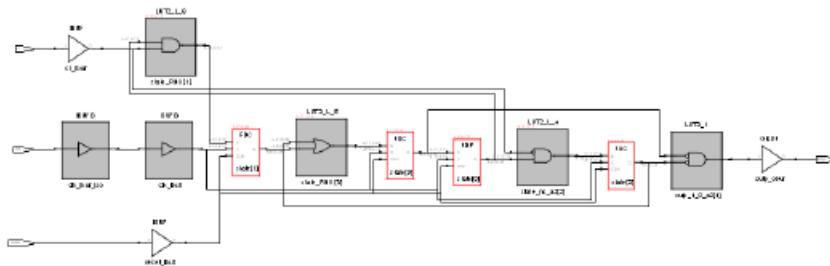
The following figure shows the default implementation of a state machine, with these encoding details reported:

```
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)

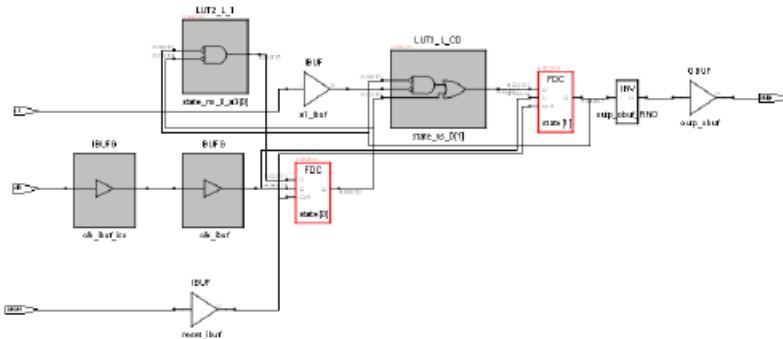
```

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



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

00	\rightarrow	00
00	\rightarrow	01
10	\rightarrow	11
11	\rightarrow	10

syn_enum_encoding

Directive

In 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 Example of syn_enum_encoding for User-Defined Encoding , on page 271 .

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 [Comparison of syn_encoding and](#)

[syn_enum_encoding](#), on page 270 for more information. For step-by-step details about setting coding styles with this attribute see [Defining State Machines in VHDL](#), on page 524 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`

`enum_encoding` is a custom attributes that is widely used to allow specific binary encodings to be attached to enumerated type objects. 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 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.

- `syn_enum_encoding` and `enum_encoding`

The `syn_enum_encoding` directive is the Synopsys equivalent of `enum_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 you have both `syn_enum_encoding` and `enum_encoding` defined, the value of `syn_enum_encoding` prevails.

- `syn_encoding` and `enum_encoding`

The Synopsys `syn_encoding` attribute specifies an implementation for a state machine. The tool uses this setting over the default if 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`.

Example of `syn_enum_encoding` for User-Defined Encoding

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

entity shift_enum is
    port (clk, rst : bit;
          O : out std_logic_vector(2 downto 0));
end shift_enum;
```

```

architecture behave of shift_enum is
type state_type is (S0, S1, S2);
attribute syn_enum_encoding: string;
attribute syn_enum_encoding of state_type : type is "001 010 101";
signal machine : state_type;
begin
    process (clk, rst)
    begin
        if rst = '1' then
            machine <= S0;
        elsif clk = '1' and clk'event then
            case machine is
                when S0 => machine <= S1;
                when S1 => machine <= S2;
                when S2 => machine <= S0;
            end case;
        end if;
    end process;
    with machine select
        O <= "001" when S0,
        "010" when S1,
        "101" when S2;
    end behave;

```

syn_enum_encoding Values Syntax

The following support applies for the `syn_enum_encoding` directive.

Global Support Object

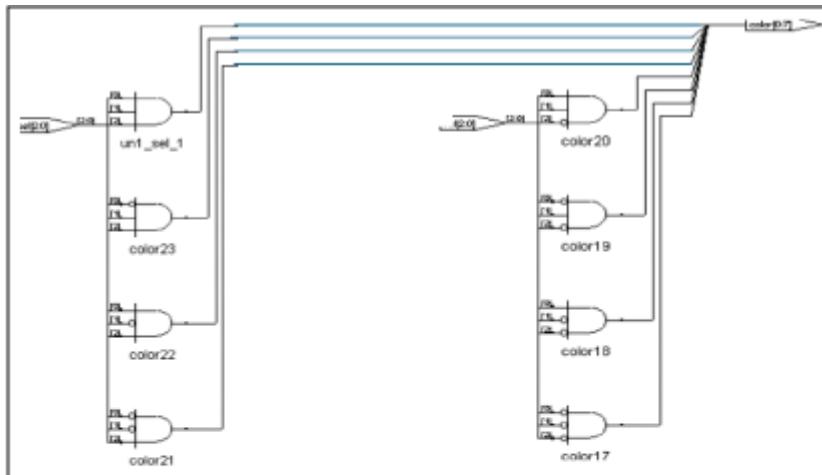
No/Yes	Enumerated data type.
--------	-----------------------

This table summarizes the syntax in the following file type:

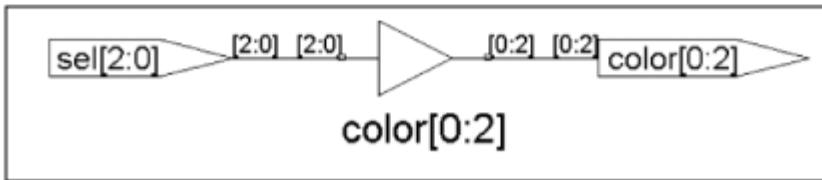
VHDL	attribute syn_enum_encoding of object : objectType is "value";	VHDL Example
------	---	------------------------------

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 401](#) for different ways to specify VHDL attributes and directives.

Here is the code used to generate the second schematic in the previous figure. (If "onehot" is used as the `syn_enum_encoding` value, the first schematic is generated instead.)

```
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_fast_auto

Attribute

Speeds up signal transitions on output ports.

Vendor	Technology
Xilinx	All

syn_fast_auto Values

Value	Description
False 0	Keeps the transition time slow.
True 1	Infers fast output buffers.

Description

The synthesis tool infers the fast output buffers OBUF_F_24, OBUFT_F_24, and IOBUF_F_24. Use the `syn_fast_auto` global attribute to force inference of slow buffers. In the HDL source code, specify `syn_fast_auto` for the top-level module or architecture.

You can override this attribute on an individual basis by using the `syn_padtype` attribute. The `syn_fast_auto` attribute has no effect on output ports specified with the `syn_padtype` attribute.

syn_fast_auto Syntax

Default	Global	Object
True 1	Yes	Module/Architecture

The following table summarizes the syntax in different files:

FDC	define_global_attribute syn_fast_auto {0 1}
Verilog	object /*synthesis syn_fast_auto= value 0 1 */;
VHDL	attribute syn_fast_auto : boolean; attribute syn_fast_auto of object : objectType is {value};

FDC Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	global	<global>	xc_fast_auto	1	boolean	Enable automatic fast output buffer use

Verilog Example

```
module ramtest (q,d,addr,we,clk) /*synthesis syn_fast_auto=0 */;
input we,clk;
input [2:0]addr;
input [3:0]d;
output reg [3:0]q;
reg [3:0] mem [7:0];

always @(posedge clk)
begin
q <= mem[addr];
if (we == 1)
mem[addr] <= d;
end
endmodule
```

VHDL Example

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

entity ramtest is
port (q : out std_logic_vector(3 downto 0);
      d : in std_logic_vector(3 downto 0);
      addr : in std_logic_vector(2 downto 0);
      we : in std_logic;
      clk : in std_logic);
end ramtest;
```

```
architecture rtl of ramtest is
attribute syn_fast_auto : boolean;
attribute syn_fast_auto of rtl : architecture is false;

type mem_type is array (7 downto 0) of std_logic_vector (3 downto 0);
signal mem : mem_type;
begin

q <= mem(conv_integer(addr));
process (clk) begin
  if rising_edge(clk) then
    if (we = '1') then
      mem(conv_integer(addr)) <= d;
    end if;
  end if;
end process;
end rtl;
```

Effect of Using syn_fast_auto

The following table shows a Xilinx design before applying the `syn_fast_auto` attribute.

Verilog No attribute statement.

VHDL No attribute statement.

Constraint Scope attribute definition disabled:

Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1 <input checked="" type="checkbox"/>	global	<global>	xc_fast_auto	1	boolean	Enable automatic fast output buffer use

Check the following lines in the generated *.edf file:

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell ramtest (cellType GENERIC)
    (view rtl (viewType NETLIST)
      (
        .....
        .....
        (property mapper_option (string ""))
      )
    )
  (design ramtest (cellRef ramtest (libraryRef work)
    (property mapper_option (string ""))
    (property PART (string "xc5vlx20tff323-1") (owner "Xilinx")))
  )
)
```

The following table shows a Xilinx design after applying the syn_fast_auto attribute.

Verilog module ramtest (q,d,addr,we,clk)/*synthesis syn_fast_auto=1 */;

```
VHDL      attribute syn_fast_auto : boolean;
           attribute syn fast auto of rtl : architecture is true;
```

Constraint

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	global	<global>	xc_fast_auto	1	boolean	Enable automatic fast output buffer use.

Search for the word "syn fast auto" in the *.edf file.

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell ramtest (cellType GENERIC)
    (view rtl (viewType NETLIST)
      (
        .....
        .....
        .....
        (property mapper_option (string ""))
        (property syn_fast_auto (integer 1))
      )
    )
  )
  (design ramtest (cellRef ramtest (libraryRef work)
    (property mapper_option (string ""))
    (property PART (string "xc6vlx20tff323-1") (owner "Xilinx")))
  )
)
```

The following table shows a Xilinx design with the value of `syn_fast_auto` attribute set to false.

```
Verilog module ramtest (q,d,addr,we,clk)/*synthesis syn_fast_auto=0
*:/;
```

```
VHDL      attribute syn_fast_auto : boolean;  
          attribute syn_fast_auto of rtl : architecture is false;
```

Constraint

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	global	<global>	xc_fast_auto	0	boolean	Enable automatic fast output buffer use

```

(library work
  (ediLevel 0)
  (technology (numberDefinition ))
  (cell ramtest (cellType GENERIC)
    (view rtl (viewType NETLIST)
      (
        -----
        -----
        (property mapper_option (string ""))
        (property syn_fast_auto (integer 0))
      )
    )
  (design ramtest (cellRef ramtest (libraryRef work)
    (property mapper_option (string ""))
    (property PART (string "xc5vlx20tf323-1") (owner "Xilinx"))
  )
)

```

The 'FAST' keyword is not forward annotated to any of the Xilinx P&R files. The following files have forward annotated "SLOW" keyword attached to output ports in the design.

In the file, "...pr_1\ramtest_map.mrp", the following statement appears:

INFO:LIT:244—All the single ended outputs in this design use slew rate limited output drivers. The delay on speed critical single ended outputs can be dramatically reduced by designating them as fast outputs.

```

..\rev_1\pr_1\ramtest_map.xrpt(281): <item label="SlewxA:Rate" stringID="SLEW_RATE" value="SLOW" />
..\rev_1\pr_1\ramtest_pad.csv(274): R1,q[2],IOB,IO_L0P_CC_RS1_2,OUTPUT,LVCMOS25*,2,12,SLOW,,,UNLOCATE
D,NO,NONE,
..\rev_1\pr_1\ramtest_pad(274):
R1|q[2]|IOB|IO_L0P_CC_RS1_2|OUTPUT|LVCMOS25*|2|12|SLOW|||UNLOCATED|NO|NONE|

```

```

..\rev_1\pr_1\ramtest_map.mrp(128): | q[0] | IOB | OUTPUT | LVCMOS25 | | 12 | SLOW | | |
..\rev_1\pr_1\ramtest_pad.txt(259): | P3 | q[1] | IOB | IO_L2N_A22_2 | OUTPUT | LVCMOS25* | 2 | 12 | SLOW | |
| UNLOCATED | NO | NONE |
..\rev_1\pr_1\ramtest_pad(258): P3|q[1]|IOB|IO_L2N_A22_2|OUTPUT|LVCMOS25*|2|12|SLOW|||UNLOCATED|NO|NO
|NE|

```

The attribute can also be applied individually in the fdc file as follows:

```
define_attribute {p:q[3:0]} {xc_fast_auto} {0}
```

Global Support

The `syn_fast_auto` attribute is defined globally by default as shown in the above examples and can be overridden individually by using the `syn_pad_type` attribute.

The following example shows the `syn_fast_auto` attribute when applied on collections.

```
FDC      define_scope_collection output_bufs {q[3:0]}
          define_attribute {$output_bufs} {xc_fast_auto} {0}
```

Constraint

Collections Tab:

	Enabled	Collection Name	Command	Command Arguments
1	<input checked="" type="checkbox"/>	output_bufs	q[3:0]	q[3:0]

Attributes Tab:

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	input_port	\$output_bufs	xc_fast_auto	0	boolean	Enable automatic fast output buffer use

The following lines can be seen in the log file when the attribute is applied using collections.

```
Adding property xc_fast_auto, value 0, to inst q[3:0]
Adding property xc_fast_auto, value 0, to port q[3:0]
Adding property xc_fast_auto, value 0, to net q[0]
Adding property xc_fast_auto, value 0, to net q[1]
Adding property xc_fast_auto, value 0, to net q[2]
Adding property xc_fast_auto, value 0, to net q[3]
```


syn_force_pads

Attribute

Prevents unused ports from being optimized.

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

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

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 149](#) 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 872](#) 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 Example

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

Attribute

Controls forward annotation of the define_input_delay and define_output_delay timing constraints in the .tcl or .ncf file for place-and-route tools.

Vendor	Technology
Intel FPGA	Stratix and newer
Xilinx	Virtex and newer

Value	Default	Global	Object	Description
0 false	Yes	Global		Does not forward annotate I/O constraints.
1 true	Default			Forward-annotates I/O constraints.

syn_forward_io_constraints Values

Value	Default	Global	Object	Description
-------	---------	--------	--------	-------------

0 false	Yes	Global		Does not forward annotate I/O constraints.
1 true	Default			Forward-annotates I/O constraints.

Description

Enables or disables forward annotation of the define_input_delay and define_output_delay timing constraints in the .scf file (for the Intel FPGA place and route tool) or .ucf file (for the Xilinx place and route tool).

Use this attribute on the top level of a VHDL or Verilog file. Alternatively you can specify syn_forward_io_constraints on a global object in the constraint file.

syn_forward_io_constraints Syntax

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

FDC Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	global	<global>	syn_forward_io_constraints	0	boolean	Forward annotate IO constraints

Verilog Example

```
module test (a,b,c,d,clk,rst)
    /* synthesis syn_forward_io_constraints = 0 */;
    input a,b,c;
    input clk, rst;
    output d;
    wire temp;
    reg d;
    assign temp = a && b || c;

    always@(posedge clk or posedge rst)
    if (rst)
        d = 1 'b0;
    else
        d=temp;
endmodule
```

VHDL Example

```

library IEEE;
use ieee.std_logic_1164.all;

entity test is
    port (a : in std_logic;
          b : in std_logic;
          clk :in std_logic;
          rst : in std_logic;
          d: out std_logic);
end test;

architecture arch of test is
attribute syn_forward_io_constraints : boolean;
attribute syn_forward_io_constraints of all : architecture is
false;
signal temp : std_logic;
begin
temp <= a and b;
process (clk, rst)begin
    if (rst ='1')then
        d <='0';
    elsif (clk'event and clk ='1')then
        d <= temp;
    end if;
end process;
end arch;

```

Effect of Using syn_forward_io_constraints

Xilinx

The default value of `syn_forward_io_constraints` is 0. The table below shows the corresponding results in the `synplicity.ucf` file:

Verilog	module test (a,b,c,d,clk,rst) /* synthesis syn_forward_io_constraints = 0 */;
VHDL	attribute syn_forward_io_constraints of all: architecture is false;

```

# Period Constraints
#Begin clock constraints
# 1001 : define_clock {clk} -name {clk} -freq {100} -clockgroup {default_clkgroup_0}
# line 13 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_0/syn_forward_io_constraints_0.sdc
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 10.000 ns HIGH 50.00%;

#End clock constraints

# I/O Registers Packing Constraints
INST "d" IOB=TRUE;
# End of generated constraints

```

The following example shows the results after setting syn_forward_io_constraints to 1 for the same design:

Verilog module test (a,b,c,d,clk,rst)
 /* synthesis syn_forward_io_constraints = 1 */;

VHDL attribute syn_forward_io_constraints of all: architecture is true;

```

# Period Constraints
#Begin clock constraints
# 1001 : define_clock {clk} -name {clk} -freq {100} -clockgroup {default_clkgroup_0}
# line 13 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_1/syn_forward_io_constraints_1.sdc
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 10.000 ns HIGH 50.00%;

#End clock constraints

# 1002 : define_input_delay {a} {1.00} -improve {0.00} -route {0.00}
# line 22 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_1/syn_forward_io_constraints_1.sdc
NET "a" TNM = "iodelay_1002_0";
TIMEGRP "iodelay_1002_0" OFFSET = IN 9.000 ns BEFORE "clk" RISING;

# 1003 : define_input_delay {b} {1.00} -improve {0.00} -route {0.00}
# line 23 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_1/syn_forward_io_constraints_1.sdc
NET "b" TNM = "iodelay_1003_0";
TIMEGRP "iodelay_1003_0" OFFSET = IN 9.000 ns BEFORE "clk" RISING;

# 1004 : define_input_delay {c} {1.00} -improve {0.00} -route {0.00}
# line 24 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_1/syn_forward_io_constraints_1.sdc
NET "c" TNM = "iodelay_1004_0";
TIMEGRP "iodelay_1004_0" OFFSET = IN 9.000 ns BEFORE "clk" RISING;

# 1005 : define_output_delay {d} {1.00} -improve {0.00} -route {0.00}
# line 25 in c:/users/anantnag/attributes/syn_forward_io_constraints/xilinx/syn_forward_io_constraints_1/syn_forward_io_constraints_1.sdc
NET "d" TNM = "iodelay_1005_0";
TIMEGRP "iodelay_1005_0" OFFSET = OUT 9.000 ns AFTER "clk" RISING;

# I/O Registers Packing Constraints
INST "d" IOB=TRUE;
# End of generated constraints

```

syn_fsm_correction

Attribute

Synplify Premier

Implements FSM double-bit error detection (DED) and single-bit error correction (SEC) logic using Hamming 3 encoding.

syn_fsm_correction Values

Value	Description
hamming3	Builds single-bit error correction logic with Hamming 3 encoding.
hamming3_ded	Builds single-bit error correction and double-bit error detection logic with Hamming 3 encoding.
hamming3_ded_recovery	Builds single-bit error correction and double-bit error detection logic with Hamming 3 encoding along with default state recovery for double-bit error.

Description

This attribute implements single-bit error correction (hamming3) or single-bit error correction along with double-bit error detection (hamming3_ded) or a single-bit error correction along with default state recovery for double-bit error detection (hamming3_ded_recovery) using Hamming 3 encoding. With hamming3_ded value, it is required to specify the error monitoring TCL commands (`syn_create_err_net` and `syn_connect`) for connecting the generated DED flag to a user-defined pin/port.

When you specify the attribute value as hamming3, hamming3_ded, or hamming3_ded_recovery the value is propagated to the state machines in the module.

If the hamming3 or hamming3_ded value and distributed TMR (see [syn_radhard-level, on page 495](#)) are both applied to a module that contains an FSM, the FSM is triplicated along with the rest of the module, and the Hamming 3 encoding specification is ignored. You also get a warning message in the log file.

syn_fsm_correction Syntax

Global Support Object

No	FDC: FSM instance HDL: signal/module/architecture
----	--

The following table summarizes the syntax in different files:

FDC	define_attribute {i:state[3:0]} {syn_fsm_correction} {hamming3 hamming3_ded}	FDC Example
Verilog	module /* syn_fsm_correction="hamming3 hamming3_ded hamming3_ded_recovery" */;	Verilog Example
VHDL	attribute syn_fsm_correction : string; attribute syn_fsm_correction of architectureName : architecture is "hamming3 hamming3_ded hamming3_ded_recovery";	VHDL Example

FDC Example

Enable	Object Type	Object	Attribute	Value	Value Type	Description
<input checked="" type="checkbox"/>	<any>	iinst_test.present_state[3:0]	syn_fsm_correction	hamming3		

Verilog Example

```
module top (input a, output b)
  /* synthesis syn_fsm_correction="hamming3" */
```

VHDL Example

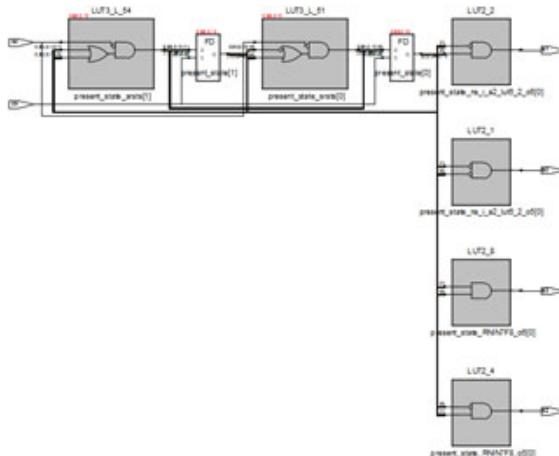
```
library ieee;
use ieee.std_logic_1164.all;
```

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

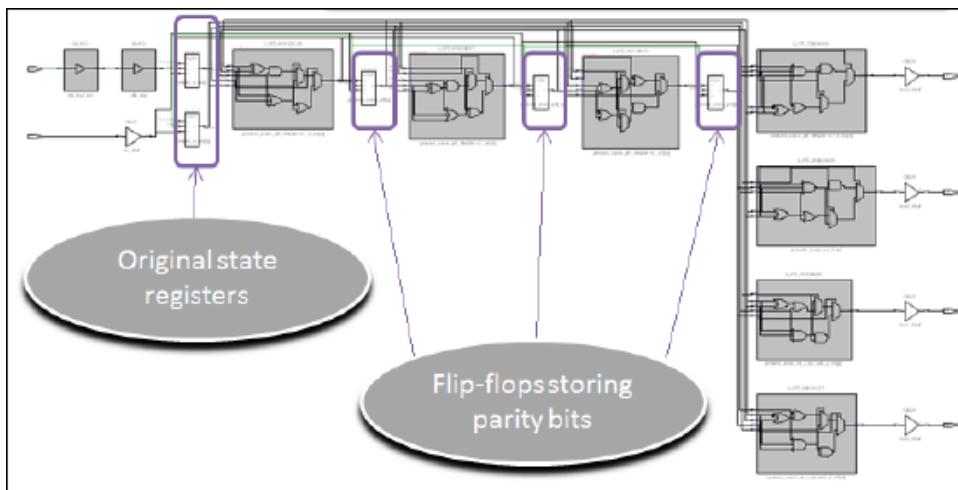
architecture rtl of test is
attribute syn_fsm_correction : string;
attribute syn_fsm_correction of rtl : architecture is "hamming3";
```

Effect of Using syn_fsm_correction

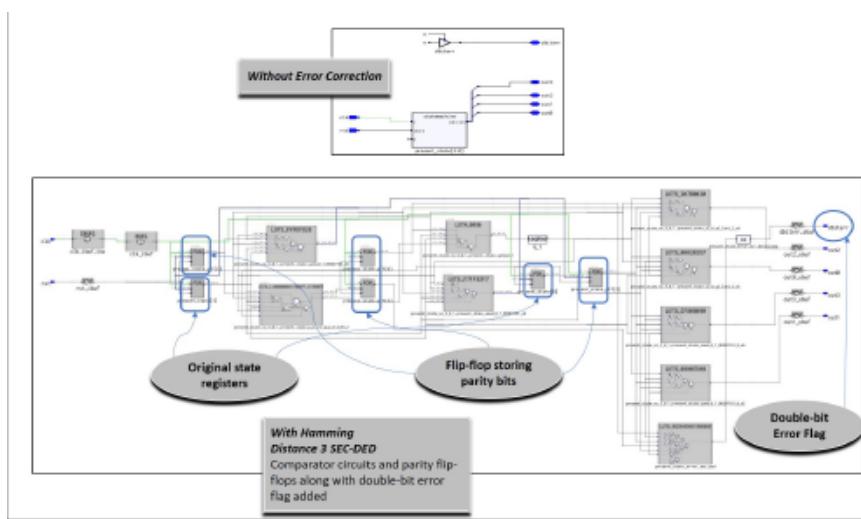
Without error correction:



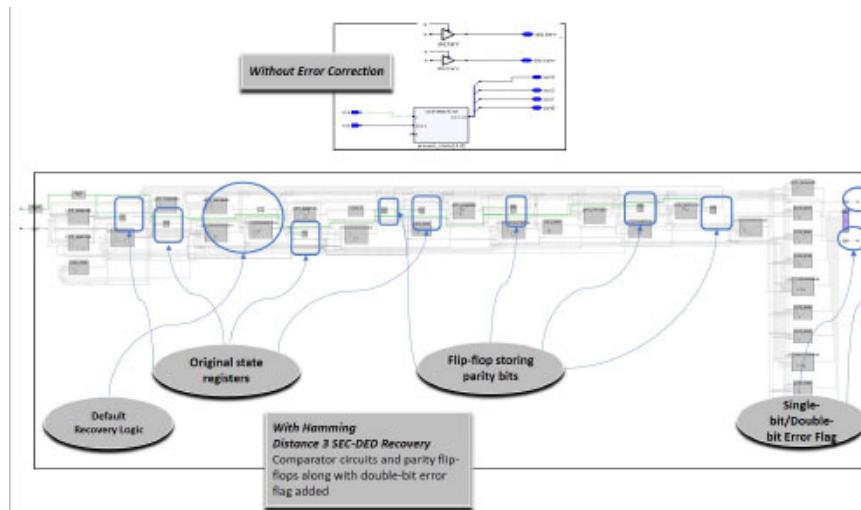
With Hamming-3 EC comparator circuits and parity flip-flops added:



With Hamming3_DED and error monitoring TCL commands, EC comparator circuits and parity flip-flops along with double error flag are added:



With Hamming3_DED_RECOVERY, EC comparator circuits and parity flip-flops along with default state recovery on double error detection are added:



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
enablePin	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 149](#) 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 872](#) 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	<code>object /* synthesis syn_gatedclk_clock_en = enablePin */;</code>	Verilog Example
VHDL	<code>attribute syn_gatedclk_clock_en : string;</code> <code>attribute syn_gatedclk_clock_en of object : signal is</code> <code>enablePin;</code>	VHDL Example

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: objectType is value;
```

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

See [VHDL Attribute and Directive Syntax, on page 401](#) 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;
```

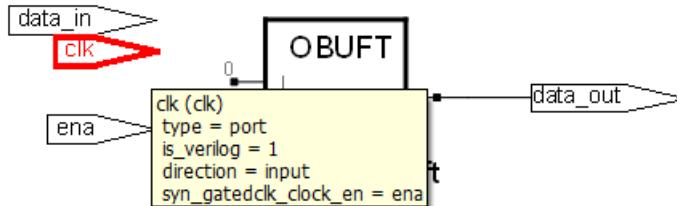
```
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;
attribute syn_gatedclk_clock_en : string;
attribute syn_gatedclk_clock_en of bbram: signal is "ena";

-- 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 149](#) 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	object /* synthesis syn_gatedclk_clock_en_polarity = 1 0 */;	Verilog Example
---------	--	---------------------------------

VHDL	attribute syn_gatedclk_clock_en_polarity of object: objectType is true false;	VHDL Example
------	---	------------------------------

Verilog Example

The following code segment provides an example.

```

module bbel (ena, clk, data_in, data_out)
    /* synthesis syn_black_box */
    /* synthesis syn_force_seq_prim=1 */;
    input clk /* synthesis syn_isclock = 1 */
        /* synthesis syn_gatedclk_clock_en="ena" */
        /* synthesis syn_gatedclk_clock_en_polarity = 0 */;
    input data_in,ena;
    output data_out;
endmodule

module bbe2 (ena, clk, data_in, data_out)
    /* synthesis syn_black_box */
    /* synthesis syn_force_seq_prim=1 */;
    input clk /* synthesis syn_isclock = 1 */
        /* synthesis syn_gatedclk_clock_en_polarity = 1 */
        /* synthesis syn_gatedclk_clock_en="ena" */;
    input data_in,ena;
    output data_out;
endmodule

module top (ena, clk, data_in , data_in2, data_out, data_out2 );
input ena, clk, data_in, data_in2;
output data_out, data_out2;
wire clk_in;
wire inv_enable;
wire clk_in2;
assign inv_enable = ~ena;
assign clk_in = inv_enable & clk;
assign clk_in2 = ena & clk;
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 401](#) 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 bbel 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 and newer families
Microchip	IGLOO/IGLOOe/IGLOO+, ProASIC3/3E, Fusion/SmartFusion
Xilinx	Virtex and newer devices

syn_global_buffers Values

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

Value	Description
An integer	For Lattice designs, specifies an integer value for the number of global buffers.
	For Microchip designs, it can be any integer between 6 and 18.
	For Xilinx designs, the value varies with the technology.

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.

You specify the attribute globally on the top-level module/entity or view.

For Microchip designs, it can be any integer between 6 and 18. If you specify an integer less than 6, the software infers six global buffers.

For Xilinx designs, the `syn_global_buffers` attribute works just like the Xilinx `xc_global_buffers` attribute ([xc_global_buffers, on page 775](#)); the only difference is that you can specify `syn_global_buffers` in the source code or in a constraint file. If both attributes are specified for the same Xilinx design, `syn_global_buffers` overwrites `xc_global_buffers`. Use this attribute instead of the Xilinx global buffers attribute `xc_global_buffers`.

Syntax Specification

FDC file `define_attribute {view} syn_global_buffers {maximum}`
`define_global_attribute syn_global_buffers {maximum}`

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

VHDL `attribute syn_global_buffers : integer;`
`attribute syn_global_buffers of object : objectType is maximum;`

SCOPE Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	global	<global>	<code>syn_global_buffers</code>	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:

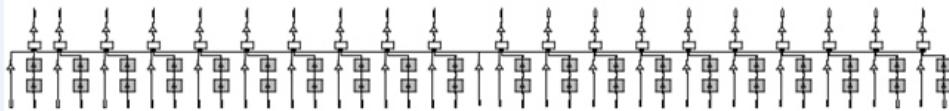
```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity top is  
    port (clk : in std_logic_vector(19 downto 0);  
          d : in std_logic_vector(19 downto 0);  
          q : out std_logic_vector(19 downto 0);  
          reset : in std_logic);  
end top;  
  
architecture behave of top is  
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)
```

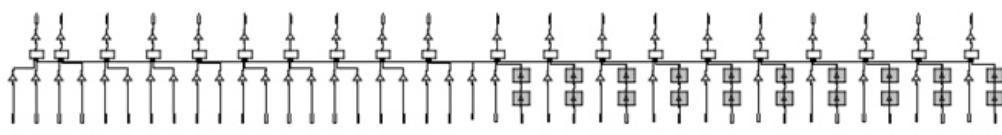
After applying attribute:

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 net clk_c[10] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[11] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[12] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[13] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[14] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[15] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[16] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[17] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[18] in view view:work.top(behave) (fanout 1)
@W:FX434 : global.vhd(4) | Because of resource limitations, clock buffer insertion could not be done on net clk_c[19] in view view:work.top(behave) (fanout 1)

@N:FX112 : | Setting available global buffers in chip view:work.top(behave) to 10
```

syn_hier

Attribute/Directive

Controls the amount of hierarchical transformation across boundaries on module or component instances during optimization.

Vendor	Devices
Achronix	Speedster7t and newer families
Intel FPGA	Stratix, Cyclone, and newer families
Lattice	ECP4, ECP3, iCE40 and newer families
Microchip	SmartFusion, ProASIC3 and newer families
Xilinx	Virtex and newer 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 Using syn_hier fixed , on page 322 .
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 syn_hier to macro on the view (the V: object type).
flatten	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 syn_netlist_hierarchy attribute (syn_netlist_hierarchy , on page 413), set to false. You can use flatten in combination with other syn_hier values; the effects are described in Using syn_hier flatten with Other Values , on page 328 . If you apply syn_hier to a compile point, flatten is the only valid attribute value. All other values only apply to the current level of hierarchy. The compile point hierarchy is determined by the type of compile point specified, so a syn_hier value other than flatten is redundant and is ignored.

Description

During synthesis, the tool dissolves as much hierarchy as possible to allow efficient logic optimization across hierarchical boundaries while maintaining optimal run times. The tool then rebuilds the hierarchy as close as possible to the original source to preserve the topology of the design.

Use the syn_hier attribute to address specific needs to maintain the original design hierarchy during optimization. This attribute gives you manual control over flattening/preserving instances, modules, or architectures in the design.

It is advised that you avoid using `syn_hier="fixed"` with tri-states.

Syntax Specification

FDC file	<code>define_attribute {object} syn_hier {value}</code>
Verilog	<code>object /* synthesis syn_hier = "value" */;</code>
VHDL	<code>attribute syn_hier of object : architecture is "value";</code>

SCOPE Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description	Comment
1	<input checked="" type="checkbox"/>	view	v:work.alu	syn_hier	hard	string	Control hierarchy flattening	

```
define_attribute {v:work.alu} {syn_hier} {hard}
```

Example of Applying `syn_hier` Attribute Globally

The `syn_hier` attribute is not supported globally. However, you can apply this attribute globally on design hierarchies using Tcl collection commands.

To do this, create a global collection of the design views in the FDC constraint file. Then, apply the attribute to the collection as shown below:

```
define_scope_collection all_views {find {v:*}}
define_attribute {$all_views} {syn_hier} {hard}
```

`syn_hier` in the SCOPE Window

If you use the SCOPE window to specify the `syn_hier` attribute, do not drag and drop the object into the SCOPE spreadsheet. Instead, first select `syn_hier` in the Attribute column, and then use the pull-down menu in the Object column to select the object. This is because you must set the attribute on a view (v:). If you drag and drop an object, you might not get a view object. Selecting the attribute first ensures that only the appropriate objects are listed in the Object column.

Using syn_hier fixed

When you use the fixed value with `syn_hier`, hierarchical boundaries are preserved with no exceptions. For example, optimizations such as constant propagation and gated or generated clock conversions are not performed across these boundaries.

Note: It is recommended that you do not use `syn_hier` with the fixed value on modules that have ports driven by tri-state gates. For details, see [When Using Tri-states, on page 322](#).

When Using Tri-states

It is advised that you avoid using `syn_hier="fixed"` with tri-states. However, if you do, here is how the software handles the following conditions:

- Tri-states driving output ports

If a module with `syn_hier="fixed"` includes tri-state gates that drive a primary output port, then the synthesis software retains a tri-state buffer so that the P&R tool can pack the tri-state into an output port.

For example, the software can infer a Xilinx BUFT so that the ISE P&R tool packs the tri-state into an OBUFT output port.

- Tri-states driving internal logic

If a module with `syn_hier="fixed"` includes tri-state gates that drive internal logic, then the synthesis software converts the tri-state gate to a MUX and optimizes within the module accordingly.

In the following code example, `myreg` has `syn_hier` set to fixed.

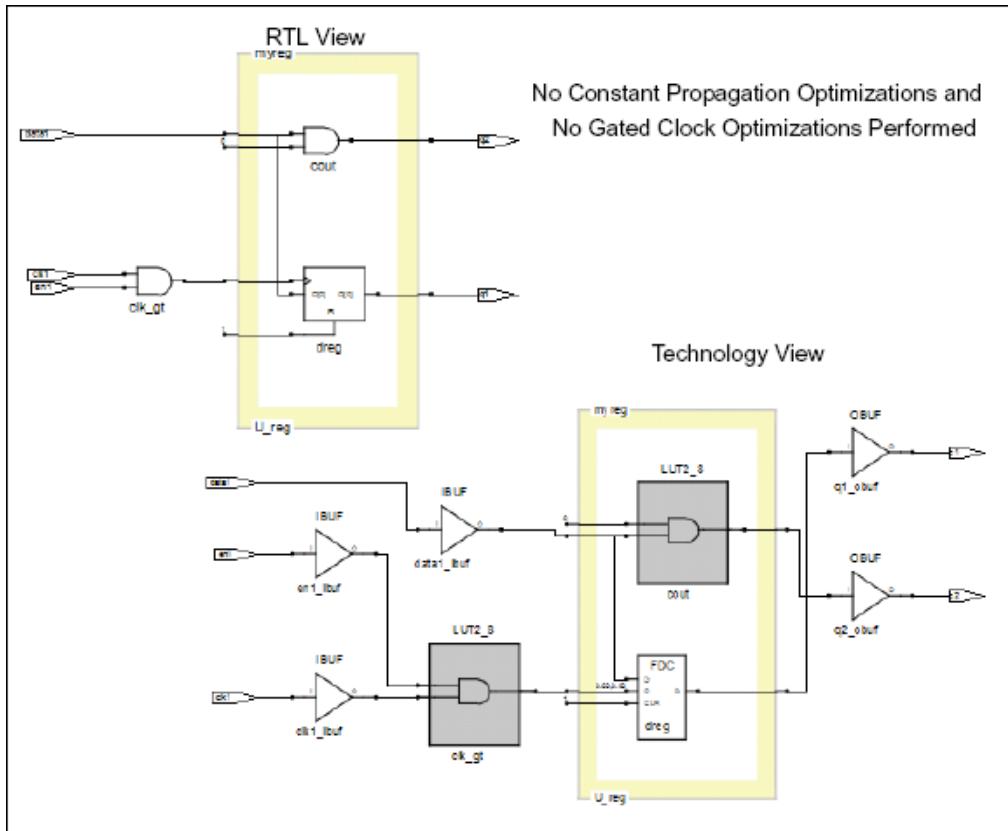
```
module top(
    clk1, en1, data1,
    q1, q2
);
    input clk1, en1;
    input data1;
    output q1, q2;
    wire cwire, rwire;
    wire clk_gt;
    assign clk_gt = en1 & clk1;
```

```
// Register module
myreg U_reg (
    .datain(data1),
    .rst(1'b1),
    .clk(clk_gt),
    .en(1'b0),
    .dout(rwire),
    .cout(cwire)
);
assign q1 = rwire;
assign q2 = cwire;
endmodule

module myreg (
    datain,
    rst,
    clk,
    en,
    dout,
    cout
) /* synthesis syn_hier = "fixed" */;
input clk, rst, datain, en;
output dout;
output cout;
reg dreg;
assign cout = en & datain;

always @ (posedge clk or posedge rst)
begin
    if (rst)
        dreg <= 'b0;
    else
        dreg <= datain;
end
assign dout = dreg;
endmodule
```

The HDL Analyst views show that `myreg` preserves its hierarchical boundaries without exceptions and prevents constant propagation and gated clock conversions optimizations.



Effect of Using syn_hier

The following VHDL and Verilog examples show the effects of using the fixed and macro values with the `syn_hier` attribute.

VHDL Example 1

```
library ieee;
use ieee.std_logic_1164.all;
entity top is
port (data1: in std_logic;
      clk1: in std_logic;
      en1: in std_logic;
      q1: out std_logic;
      q2: out std_logic);
end;

architecture rtl of top is
signal cwire, rwire: std_logic;
signal clk_gt: std_logic;
component dff is
port (datain: in std_logic;
      rst: in std_logic;
      clk: in std_logic;
      en: in std_logic;
      dout: out std_logic;
      cout: out std_logic);
end component;
begin
U1 : dff port map(datain => data1, rst => '1', clk =>
      clk_gt, en => '0', dout => rwire, cout => cwire);
q1 <= rwire;
q2 <= cwire;
clk_gt <= en1 and clk1;
end;

library ieee;
use ieee.std_logic_1164.all;
entity dff is
port (datain: in std_logic;
      rst: in std_logic;
      clk: in std_logic;
      en: in std_logic;
      dout: out std_logic;
      cout: out std_logic);
end;

architecture rtl of dff is
signal dreg: std_logic;
attribute syn_hier : string;
attribute syn_hier of rtl: architecture is "fixed";
begin
```

```

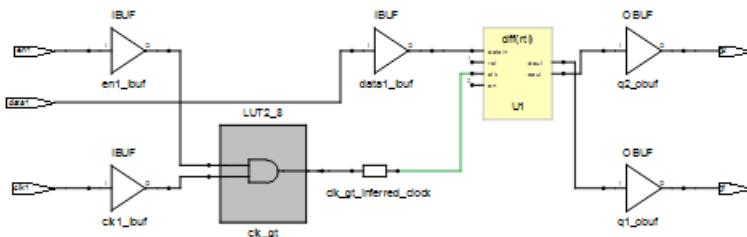
process (clk, rst)
begin
  if (rst = '1') then
    dreg<= '0';
  elsif (clk'event and clk ='1') then
    dreg<= datain;
  end if;
  dout <= dreg;
end process;
end;

```

After applying attribute with the value *fixed*:

Verilog Module myreg(datain,rst,clk,en,dout,cout)/*synthesis syn_hier="fixed"*/;

VHDL attribute syn_hier : string;
attribute syn_hier of rtl: architecture is "fixed";



Verilog Example 2

```

module inc(a_in, a_out) /* synthesis syn_hier = "macro" */;
  input [3:0] a_in;
  output [3:0] a_out;
endmodule

module reg4(clk, rst, d, q);
  input [3:0] d;
  input clk, rst;
  output [3:0] q;
  reg [3:0] q;
  always @ (posedge clk or posedge rst)

```

```

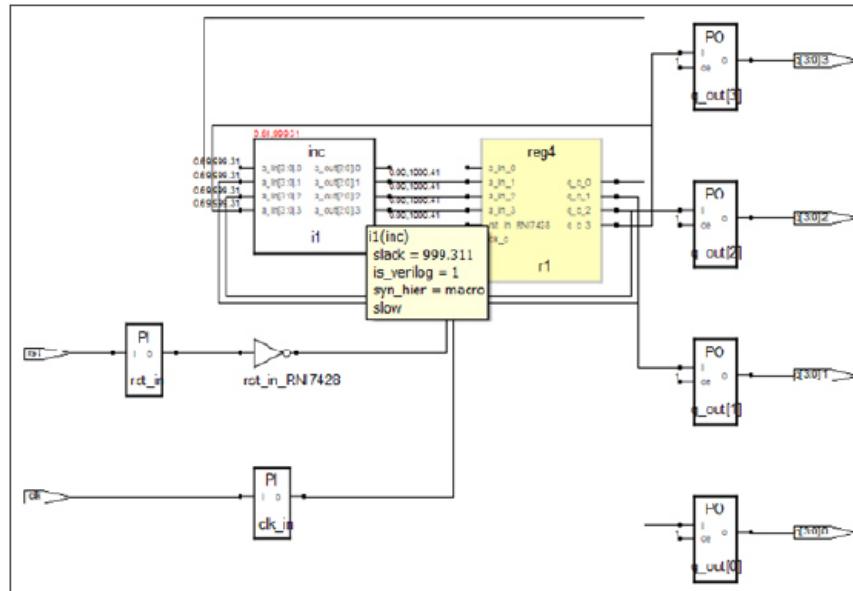
if(rst)
q <= 0;
else
q <= d;
endmodule

module top(clk, rst, q);
input clk, rst;
output [3:0] q;
wire [3:0] a_in;
inc i1(q, a_in);
reg4 r1(clk, rst, a_in, q);
endmodule

```

After applying attribute with value *macro*:

Verilog	<code>module inc(a_in, a_out) /* synthesis syn_hier = "macro" */;</code>
VHDL	<code>attribute syn_hier : string;</code> <code>attribute syn_hier of rtl: architecture is "macro";</code>



Using syn_hier flatten with Other Values

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

flatten,soft	Same as flatten.
flatten,firm	Flattens all lower levels of the design but preserves the interface of the design unit in which it is specified. This option also allows optimization of cell packing across the boundary.
flatten,remove	Flattens all lower levels of the design, including the one on which it is specified.

If you use flatten in combination with another option, the tool flattens as directed until encountering another syn_hier attribute at a lower level. The lower level syn_hier attribute then takes precedence over the higher level one.

These examples demonstrate the use of the flatten and remove values to flatten the current level of the hierarchy and all levels below it (unless you have defined another syn_hier attribute at a lower level).

```
Verilog module top1 (Q, CLK, RST, LD, CE, D)
/* synthesis syn_hier = "flatten,remove" */;
// Other code
```

```
VHDL architecture struct of cpu is
attribute syn_hier : string;
attribute syn_hier of struct: architecture is "flatten,remove";
-- Other code
```

syn_highrel_ioconnector

Attribute

Synplify Premier

Identifies I/O connectors that interface to modules with distributed TMR or DWC to ensure their inputs and outputs are not a single point of failure.

Vendor	Technologies
Intel FPGA	Arria 10, Arria V, Cyclone V, Cyclone IV, Cyclone III, Stratix V, Stratix IV, Stratix III, Stratix10
Microchip	IGLOOE, IGLOO+, IGLOO 2, ProASIC3L, ProASIC3E, ProASIC3, SmartFusion 2
Xilinx	Artix-7, Kintex-7, Virtex7 and UltraScale/UltraScale+ families

syn_highrel_ioconnector Values

Value	Description	Default	Global	Object
1	Identifies I/O connectors that interface to modules with distributed TMR or DWC.	None	No	Module/ Architecture

Description

Identifies I/O connectors that interface to modules with distributed TMR or DWC, which allows you to access the signals within its boundaries ensuring their inputs and outputs are not a single point of failure.

syn_highrel_ioconnector Syntax

You cannot specify this attribute as a global value.

FDC	define_attribute syn_highrel_ioconnector = {1 0}	FDC Example
Verilog	object /* syn_highrel_ioconnector="1 0" */;	Verilog Example
VHDL	attribute syn_highrel_ioconnector : string; attribute syn_highrel_ioconnector of <i>architectureName</i> : architecture is "true" "false";	VHDL Example

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	view	v:work.ipconn_dwc_2s	syn_highrel_ioconnector	1	integer	Defines an I/O connector for high reliability feature
2	<input checked="" type="checkbox"/>	view	v:work.opconn_dwc_2s	syn_highrel_ioconnector	1	integer	Defines an I/O connector for high reliability feature

For example:

```
define_attribute {v:work.ipconn_dwc_2s}
  {syn_highrel_ioconnector} {1}

define_attribute {v:work.opconn_dwc_2s}
  {syn_highrel_ioconnector} {1}
```

Verilog Example

For example:

```
module ipconn_dtmr (in1, in2, in3, out)
  /* synthesis syn_highrel_ioconnector="1" */
```

VHDL Example

For example:

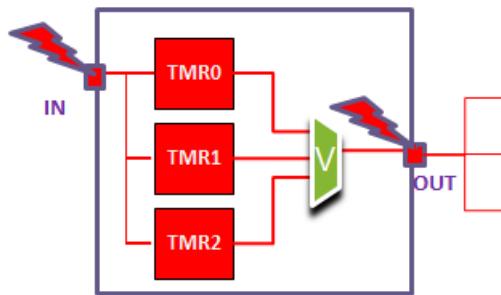
```
architecture rtl of ipconn_dtmr is
attribute syn_highrel_ioconnector : string;
attribute syn_highrel_ioconnector of rtl;
architecture is "true";
```

Effect of Using syn_highrel_ioconnector

The following examples show how the syn_highrel_ioconnector attribute can be used for either distributed TMR or DWC with I/O connectors.

Distributed TMR with I/O Connectors

I/Os can be a single point of failure as shown below.



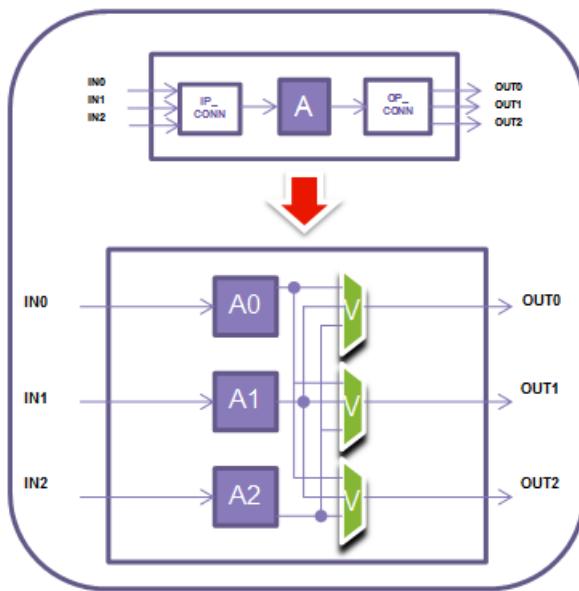
You can use the syn_highrel_ioconnector attribute to create redundant ports and instantiate connector objects in RTL, by specifying the

- Valid input connector - 3 inputs and 1 output with majority voter logic.
- Valid output connector - 1 input and 3 outputs with feedthrough logic.

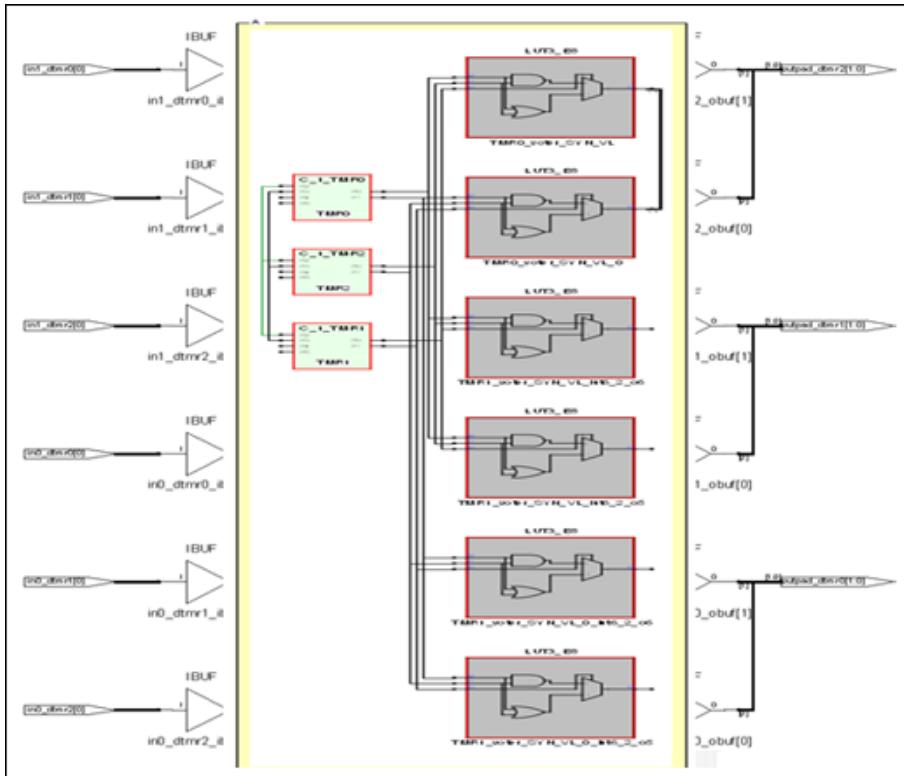
Use this attribute in conjunction with the syn_radhardlevel attribute, specifying the distributed_tmr value on the appropriate module.

For this example, you can enable DTMR with I/O connectors as follows:

```
define_attribute {v:IP_CONN} {syn_highrel_ioconnector} {1}
define_attribute {v:OP_CONN} {syn_highrel_ioconnector} {1}
define_attribute {v:work.A} {syn_radhardlevel} {distributed_tmr}
```



The HDL Analyst view shows that *_dtmr0 ports are connected to the TMR0 instance, *_dtmr1 ports are connected to the TMR1 instance, and *_dtmr2 ports are connected to the TMR2 instance, when dissolving hierarchy for the connector objects.



DWC with I/O Connectors

You can use the `syn_highrel_ioc`nnector attribute to create redundant ports and instantiate connector objects in RTL, by specifying the

- Valid input connector - 2 inputs and 1 output with AND/OR logic.
- Valid output connector - 1 input and 2 outputs with feedthrough logic.

Use this attribute in conjunction with the `syn_radhardlevel` attribute, specifying the `duplicate_with_compare` value on the appropriate module.

For this example, you can enable DWC with I/O connectors as follows:

```

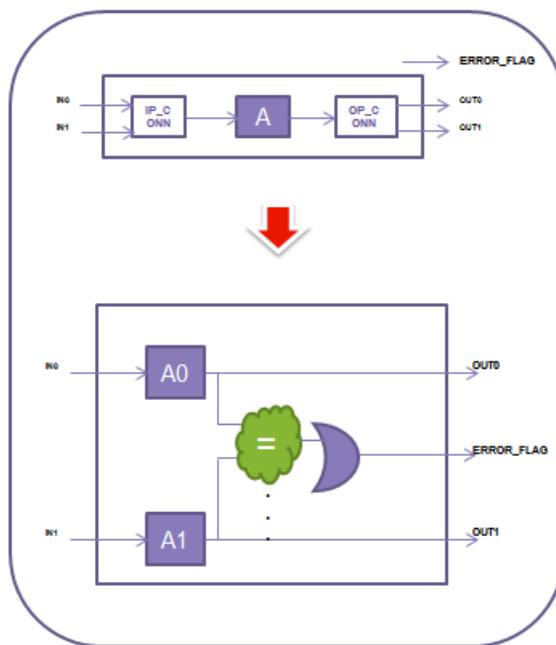
define_attribute {v:IP_CONN} {syn_highrel_iocnector} {1}
define_attribute {v:OP_CONN} {syn_highrel_iocnector} {1}
define_attribute {v:work.A} {syn_radhardlevel} {duplicate_with_compare}
syn_create_err_net {-name {error_flag_net} -inst {i:inst_A}}
syn_connect {-from {n:error_flag_net} -to {p:ERROR_FLAG}}

```

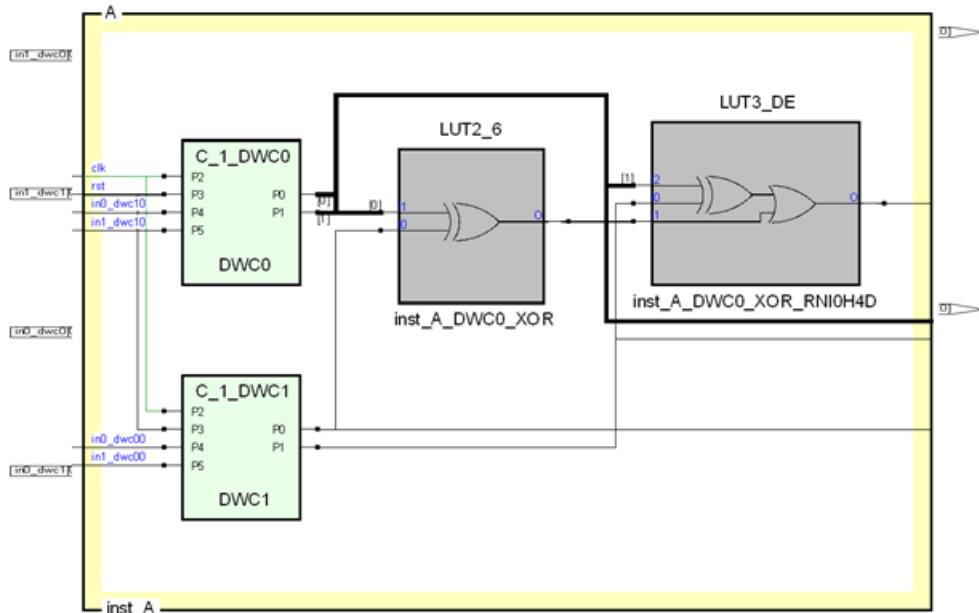
Two identical copies of module A are created: A0 and A1 with their outputs compared (XOR/OR) and connected to the error monitoring port ERROR_FLAG. Use the command

- syn_create_err_net to specify the instance name of the DWC module, for which the error flag needs to be instrumented and connected to the specified net.
- syn_connect to associate the net specified for the syn_create_err_net command with the error monitoring port ERROR_FLAG.

Note: You can create optional pipeline registers using other options of the syn_create_err_net command.



The HDL Analyst view shows that *_dwc0 ports are connected to the DWC0 instance and *_dwc1 ports are connected to the DWC1 instance, when dissolving hierarchy for the connector objects.



syn_implement

UPF Directive

Ensures that the user-defined custom retention models are preserved after compile in the srs file.

Syntax

syn_implement = "1"

Where 1 is the only valid value.

Example

```
module ret_dffrse (Q, CLK, SAVE, RESTORE, D, S, R, E)
    /* synthesis syn_implement = "1" syn_upf_ret_type = "DFFRSE" */;
    logic
endmodule
```


syn_insert_buffer

Attribute

Inserts a technology-specific clock buffer.

Vendor	Technologies
Lattice	iCE40 and newer families
Microchip	Axcelerator IGLOO/IGLOOe/IGLOO+/IGLOO2 ProASICPLUS, ProASIC3/3E/3L SmartFusion2 and newer families
Xilinx	Virtex and newer families

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"> • Ports: SB_GB_IO • Nets: SB_GB 	iCE40

Vendor	Value	Description	Technology
Microchip	CLKBUF	Pads: CLKBUF	SmartFusion2, IGLOO2, IGLOO+, IGLOOe, IGLOO, and ProASICPLUS
	HCLKBUF	Pads: HCLKBUF (for nets that drive the clock pins of sequential primitives)	IGLOO+, IGLOOe, IGLOO, and ProASICPLUS
	CLKBIBUF	Pads: CLKBIBUF	SmartFusion2, IGLOO2 only
	CLKINT	Nets: CLKINT	SmartFusion2, IGLOO2 only
	GCLKINT	Nets: HCLKINT (for nets that drive the clock pins of sequential primitives)	IGLOO+, IGLOOe, IGLOO, and ProASICPLUS
	RCLKINT	Nets: RCLKINT	SmartFusion2, IGLOO2 only

Vendor	Value	Description	Technology
Xilinx (Instance)	BUFGMUX none	BUFGMUX inference is disabled by default. If the attribute is not specified, the default behavior is to infer the LUT driving the BUFG. Or, specify one of these values: <ul style="list-style-type: none">• none: Infers only the LUT, without the BUFG• BUFGMUX: Infers a BUFGMUX	All supported technologies
	BUFG	This is the default buffer inferred if nothing is specified.	Valid for Virtex-4 and later devices.
	BUFH	Explicitly specify this value to infer a BUFH.	Valid for Virtex-4 and later devices.
	BUFR	Explicitly specify this value to infer a BUFR.	Valid for Virtex-4 and later devices.

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.
Microchip	Instance	Inserts the specified clock buffer.
Xilinx	Instance, port, net	Inserts the specified clock buffer.

syn_insert_buffer Syntax Specification

You cannot specify this attribute as a global value.

FDC	<code>define_attribute object syn_insert_buffer value</code>	FDC Example
Verilog	<code>object /* synthesis syn_insert_buffer = "value" */;</code>	Verilog Examples
VHDL	<code>attribute syn_insert_buffer of object : objectType is "value";</code>	VHDL Examples

FDC Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>		i:dk_mux	syn_insert_buffer	BUFGMUX		

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 gt11 /* synthesis syn_insert_buffer = "SB_GB" syn_keep = 1 */;
assign gt11 = 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.

Microchip `syn_insert_buffer` Verilog Example

In the following example, the attribute is attached to `LPDRE`, `SEL`, `RST`, `LDCOMP`, and `CLK`.

```

module prep2_2 (DATA0, DATA1, DATA2, LPDRE, SEL, RST, CLK, LDCOMP);
output [7:0] DATA0;
input [7:0] DATA1, DATA2;
input LPDRE, SEL, RST, CLK
    /* synthesis syn_insert_buffer = "GL25" */, LDCOMP;
wire [7:0] DATA0_internal;
prep2_1 inst1 (CLK, RST, SEL, LDCOMP, LPDRE, DATA1, DATA2,
    DATA0_internal);
prep2_1 inst2 (CLK, RST, SEL, LDCOMP, LPDRE, DATA0_internal,
    DATA2, DATA0);
endmodule

module prep2_1 (CLK, RST, SEL, LDCOMP, LPDRE, DATA1, DATA2, DATA0);
input CLK, RST, SEL, LDCOMP, LPDRE;
input [7:0] DATA1, DATA2;
output [7:0] DATA0;
reg [7:0] DATA0;
reg [7:0] highreg_output, lowreg_output; // internal registers
wire compare_output = (DATA0 == lowreg_output); // comparator
wire [7:0] mux_output = SEL ? DATA1 : highreg_output;

// mux registers
always @ (posedge CLK or posedge RST)
begin
    if (RST) begin
        highreg_output = 0;
        lowreg_output = 0;
    end else begin
        if (LPDRE)
            highreg_output = DATA2;
    end
end
endmodule

```

```
        if (LDCOMP)
            lowreg_output = DATA2;
    end
end

// counter
always @ (posedge CLK or posedge RST)
begin
    if (RST)
        DATA0 = 0;
    else if (compare_output) // load
        DATA0 = mux_output;
    else
        DATA0 = DATA0 + 1;
end
endmodule
```

Xilinx syn_insert_buffer Verilog Example

```
`define SIZE 16
module cr_top (
    input rst,
    input [`SIZE:0] in1,in2,
    output reg [`SIZE:0] out,
    input clk1,
    input clk2,
    input sel_clk
);

wire clk_mux/*synthesis syn_insert_buffer="BUFGMUX"*/;

assign clk_mux = sel_clk ? clk1 : clk2;

always @ (posedge clk_mux or posedge rst)
begin
    if(rst)
        out <= 0;
    else
        out <= in1&in2;
end
endmodule
```

VHDL Examples

This section provides technology-specific examples.

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

entity prep2_1 is
    port (clk : in bit;
          rst : in bit;
          sel : in bit;
          ldcomp : in bit;
          ldpre : in bit;
          data1,data2 : in std_logic_vector(7 downto 0);
          data0 : out std_logic_vector(7 downto 0));
end prep2_1;

architecture behave of prep2_1 is
signal equal: bit;
signal mux_output: std_logic_vector(7 downto 0);
signal lowreg_output: std_logic_vector(7 downto 0);
signal highreg_output: std_logic_vector(7 downto 0);
signal data0_i: std_logic_vector(7 downto 0);
begin
    compare: process(data0_i, lowreg_output)
    begin
        if data0_i = lowreg_output then
            equal <= '1';
        else
            equal <= '0';
        end if;
    end process compare;

    mux: process(sel, data1, highreg_output)
    begin
        case sel is
            when '0' =>
                mux_output <= highreg_output;
            when '1' =>
                mux_output <= data1;
        end case;
    end process mux;
```

```
registers: process (rst,clk)
begin
    if (rst = '1') then
        highreg_output <= "00000000";
        lowreg_output <= "00000000";
    elsif clk = '1' and clk'event then
        if ldpre = '1' then
            highreg_output <= data2;
        end if;
        if ldcomp = '1' then
            lowreg_output <= data2;
        end if;
    end if;
end process registers;

counter: process (rst,clk)
begin
    if rst = '1' then
        data0_i <= "00000000";
    elsif clk = '1' and clk'event then
        if equal = '1' then
            data0_i <= mux_output;
        elsif equal = '0' then
            data0_i <= data0_i + "00000001";
        end if;
    end if;
end process counter;
data0 <= data0_i;
end behave;

library ieee;
use ieee.std_logic_1164.all;

entity prep2_2 is
    port (CLK : in bit;
          RST : in bit;
          SEL : in bit;
          LDCOMP : in bit;
          LDPRE : in bit;
          DATA1,DATA2 : in std_logic_vector(7 downto 0);
          DATA0 : out std_logic_vector(7 downto 0));
attribute syn_insert_buffer : string;
attribute syn_insert_buffer of clk : signal is "GL25";
end prep2_2;
```

```
architecture behave of prep2_2 is
component prep2_1
    port (clk : in bit;
          rst : in bit;
          sel : in bit;
          ldcomp : in bit;
          ldpre : in bit;
          data1,data2 : in std_logic_vector(7 downto 0);
          data0 : out std_logic_vector(7 downto 0));
end component;

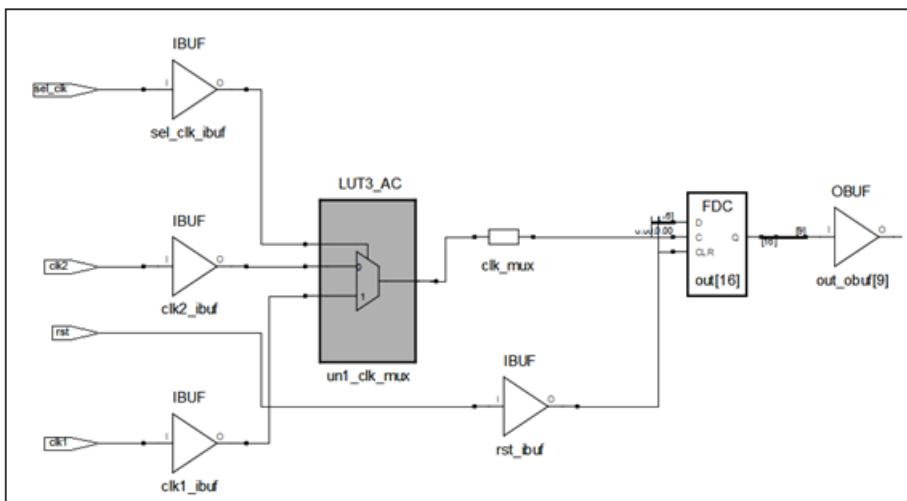
signal data0_internal : std_logic_vector (7 downto 0);

begin
inst1: prep2_1 port map(clk => CLK, rst => RST, sel => SEL,
ldcomp => LDCOMP, ldpre => LDPRE, data1 => DATA1,
data2 => DATA2, data0 => data0_internal);
inst2: prep2_1 port map(clk => CLK, rst => RST, sel => SEL,
ldcomp => LDCOMP, ldpre => LDPRE, data1 => data0_internal,
data2 => DATA2, data0 => DATA0);
end behave;
```

Effect of Using syn_insert_buffer

Xilinx

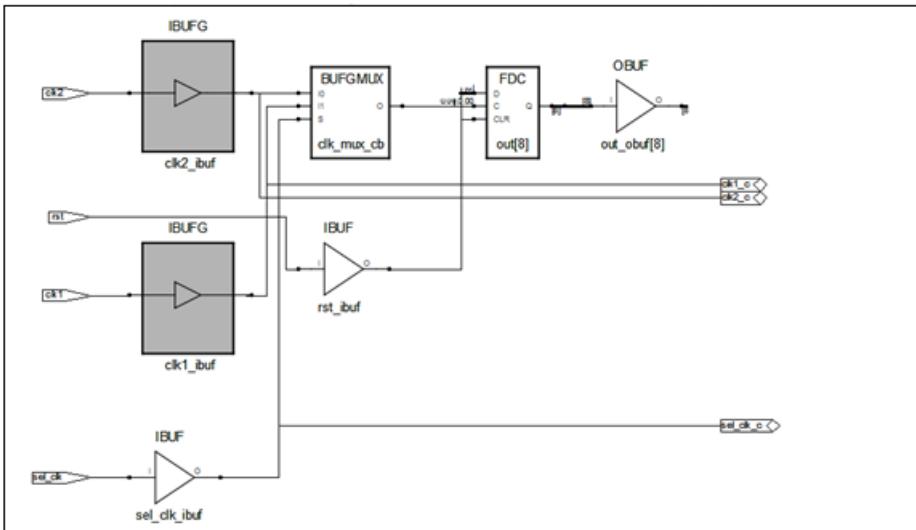
This example shows a Virtex-7 device in the default state, before syn_insert_buffer is applied. There is no BUFGMUX on the clock.



After applying the `syn_insert_buffer` attribute, the tool infers a BUFGMUX, as shown below.

Verilog `wire clk_mux/*synthesis syn_insert_buffer="BUFGMUX"*/;`

VHDL `attribute syn_insert_buffer of clk_out : signal is "BUFGMUX";`



syn_insert_pad

Attribute

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

Vendor	Technology
Achronix	Speedster7t and newer families
Lattice	iCE40 and newer families
Microchip	SmartFusion2, IGLOO2 and newer families
Xilinx	Virtex and newer families Spartan and newer families

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](#)

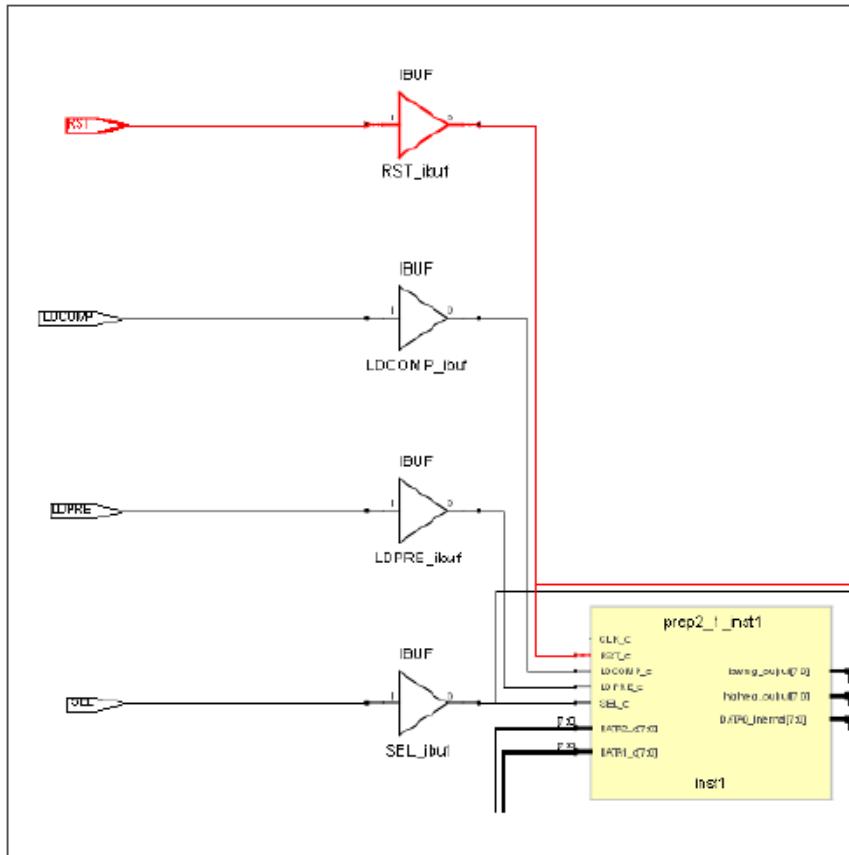
SCOPE Example

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

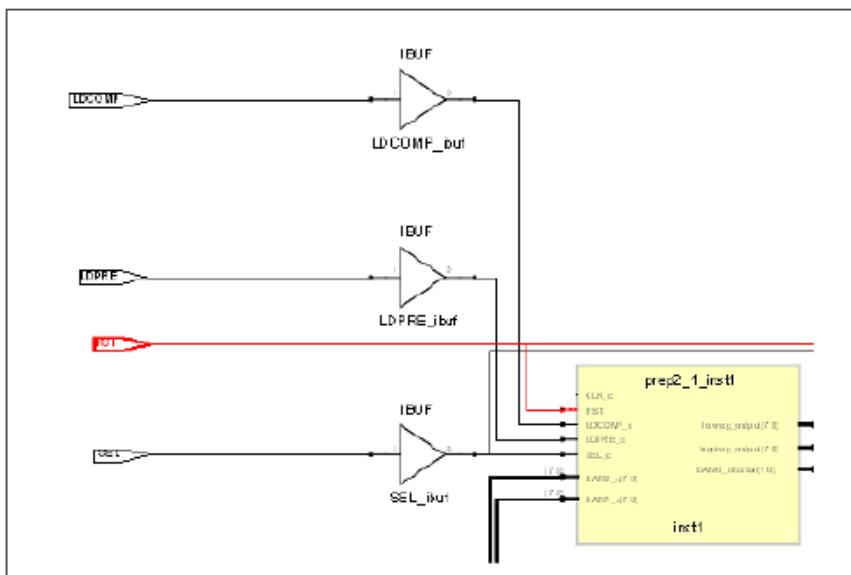
Enable	Object Type	Object	Attribute	Value	Value Type
<input checked="" type="checkbox"/>	<any>	pRST	<code>syn_insert_pad</code>	0	

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 149 for a list of the associated directives.

syn_isclock Values Syntax

Verilog *object /* synthesis syn_isclock = 1 */;*

VHDL attribute syn_isclock of *object*: *objectType* is true;

Verilog Example

```
module test (myclk, a, b, tout,) /* synthesis syn_black_box */;
  input myclk /* synthesis syn_isclock = 1 */;
  input a, b;
  output tout;
endmodule

//Top Level
module top (input clk, input a, b, output fout);
  test U1 (clk, a, b, fout);
endmodule
```

VHDL Example

See [VHDL Attribute and Directive Syntax, on page 401](#) 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;

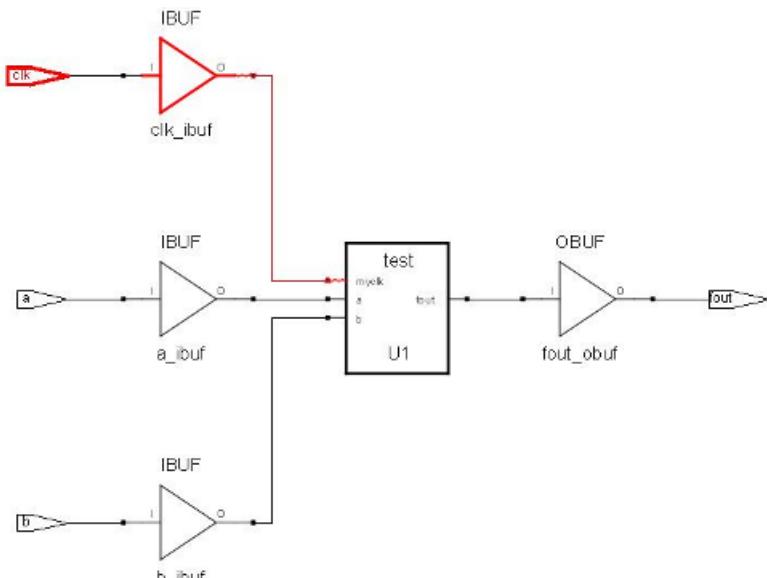
-- TOP Level--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity top is
  generic (size: integer := 8);
  port (fout : out std_logic_vector (size- 1 downto 0);
        a : in std_logic_vector (size- 1 downto 0);
        b : in std_logic_vector (size- 1 downto 0);
        clk : in std_logic
```

```
 );
end;

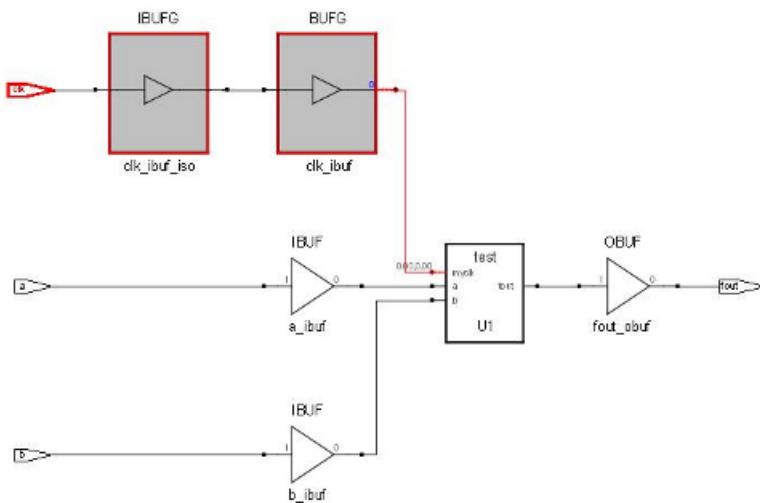
architecture rtl of top is
component test
generic (size: integer := 8);
port (tout :  out std_logic_vector (size- 1 downto 0);
      a :  in std_logic_vector (size- 1 downto 0);
      b :  in std_logic_vector (size- 1 downto 0);
      myclk : in std_logic
    );
end component;
begin
U1 : test port map (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 364](#) 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.
 - To prevent the inferencing of `altnet_add` and `altnet_accum` (Intel FPGA). Add the `syn_keep` directive between the adder and the multiplier of a Multiply/Accumulate (MAC) block to prevent this inferencing while preserving the inference of plain multipliers.

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:

logic	logic temp /* synthesis syn_keep = 1 */;
wire	wire temp /* synthesis syn_keep = 1 */;
bit	bit temp /* synthesis syn_keep = 1 */;

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:

syn_keep 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.

syn_preserve Ensures that registers are not optimized away.

syn_noprune 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 571](#) in the *User Guide* for more information.

syn_keep Syntax

CDC File	CDC Example for a Verilog Design define_directive {n:libName.moduleName 1netName} {syn_keep} {0 1}	CDC Examples
	CDC Example for a VHDL Design define_directive {n:libName.moduleName.[architectureName] ² netName} {syn_keep} {0 1}	

Verilog	object /* synthesis syn_keep = 1 */;	Verilog Example
VHDL	attribute syn_keep : boolean attribute syn_keep of object : objectType is true;	VHDL Example

1. A pipe (|) separator character is required with no spaces between the net name and the module name
2. Optionally, specifies the name of the library for VHDL designs. Use for single entity or multiple architecture-based designs.

CDC Examples

Verilog and VHDL .cdc examples with the syn_keep attribute are shown below:

Verilog CDC Example

```
define_directive {n:work.example1|keep1} {syn_keep} {1}
define_directive {n:work.example1|keep2} {syn_keep} {1}
```

VHDL CDC Example

```
define_directive {n:work.example2.rtl|keep1} {syn_keep} {1}
define_directive {n:work.example2.rtl|keep2} {syn_keep} {1}
```

For more information about using compiler directives, see [Using the Compiler Directives Editor, on page 60](#) or [Specifying Directives in a CDC File, on page 141](#).

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 364](#).

```
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 364](#).

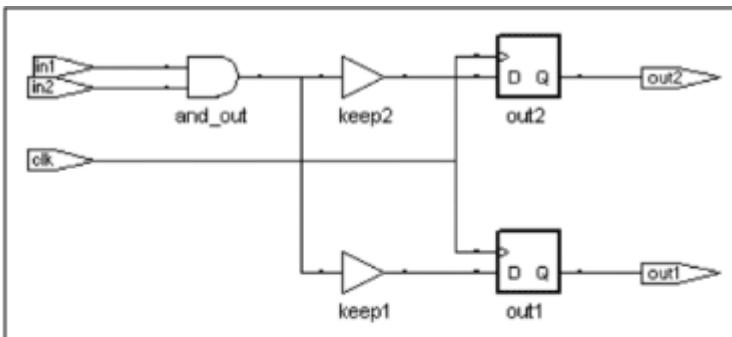
```
entity example2 is
    port (in1, in2 : in bit;
          clk : in bit;
          out1, out2 : out bit);
end example2;

architecture rt1 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 rt1;
```

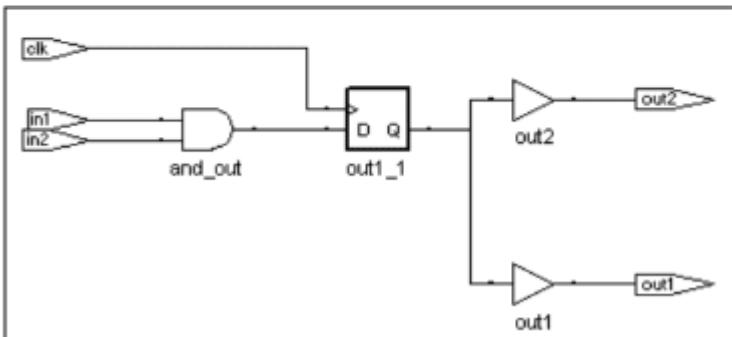
Effect of Using syn_keep

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_loc

Attribute

The `syn_loc` attribute specifies the location (placement) of ports.

Vendor	Technology
Intel FPGA	Stratix and newer families
Microchip	SmartFusion, ProASIC
Xilinx	Virtex and newer families. Use this attribute in place of <code>xc_loc</code> .

syn_loc Values

Value	Description
Pin numbers	Assigns pin numbers to ports.

Description

Specifies pin locations for I/O pins and cores, and forward-annotates this information to the place-and-route tool. This attribute can only be specified in a top-level source file or a constraint file.

Use the QSF2SDC utility to set this attribute when it translates Intel FPGA QSF `set_location_assignment` and `set_instance_assignment` constraints.

syn_loc Syntax

Default	Global Attribute	Object
---------	------------------	--------

pinNumbers is a comma-separated list of pin or placement numbers. Refer to the vendor data book for valid values.

FDC	define_attribute <i>portDesignName</i> {syn_loc} { <i>pinNumbers</i> }	FDC Example
Verilog	<i>object</i> /* synthesis syn_loc = "pinNumbers" */	Verilog Example
VHDL	attribute syn_loc of <i>object</i> : <i>objectType</i> is "pinNumbers";	VHDL Example

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>		out1[2:0]	syn_loc	P14 P12,P11	string	Assign the object location
2							

The following are examples of using this attribute:

Intel FPGA	define_attribute {DATA0[7:0]} syn_loc {14,12,11,5,21,18,16,15}
Microchip	define_attribute {CR_DIN[3:0]} syn_loc {M7, Y6, B6, D10}
Xilinx	define_attribute {DATA0[4:0]} syn_loc {P14,P12,P11,P5,P21}

You can also specify locations for individual bus bits with this attribute:

Intel FPGA	Xilinx
<pre>define_attribute {DATA0[7]} syn_loc {14} define_attribute {DATA0[6]} syn_loc {12} define_attribute {DATA0[5]} syn_loc {11} define_attribute {DATA0[4]} syn_loc {5} define_attribute {DATA0[3]} syn_loc {21} define_attribute {DATA0[2]} syn_loc {18} define_attribute {DATA0[1]} syn_loc {16} define_attribute {DATA0[0]} syn_loc {15}</pre>	<p>Specify individual bits:</p> <pre>define_attribute {valid[0]} syn_loc {R6} define_attribute {valid[1]} syn_loc {T2} define_attribute {valid[2]} syn_loc {R5} define_attribute {valid[3]} syn_loc {R1}</pre> <p>Specify the b: prefix and the bit slice:</p> <pre>define_attribute {b:valid[0]} syn_loc {R6} define_attribute {b:valid[1]} syn_loc {T2} define_attribute {b:valid[2]} syn_loc {R5} define_attribute {b:valid[3]} syn_loc {R1}</pre>

Microchip

```
define_attribute {CR_DIN[3]} syn_loc {M7}
define_attribute {CR_DIN[2]} syn_loc {Y6}
define_attribute {CR_DIN[1]} syn_loc {B6}
define_attribute {CR_DIN[0]} syn_loc {D10}
```

Specify the b: prefix and the bit slice:
define_attribute {b:CR_DIN[0]} syn_loc {D10}
define_attribute {b:CR_DIN[1]} syn_loc {B6}
define_attribute {b:CR_DIN[2]} syn_loc {Y6}
define_attribute {b:CR_DIN[3]} syn_loc {M7}

Verilog Example

```
Intel FPGA output [7:0] DATA0
/* synthesis syn_loc = "14,12,11,5,21,18,16,15" */;

Microchip input [3:0] CR_DIN /* synthesis syn_loc = "M7,Y6, B6, D10" */

Xilinx output reg [2:0] out1/*synthesis syn_loc = "P14,P12,P11"*/;
```

```
module test(a, b, clk, out1);
    input clk;
    input [2:0]a;
    input [2:0]b;
    output reg [2:0] out1/* synthesis syn_loc = "P14,P12,P11"*/;

    always@ (posedge clk)
    begin
        out1 <= a + b;
    end
endmodule
```

VHDL Example

```
Intel      attribute syn_loc : string;
FPGA      attribute syn_loc of data0: signal is "14,12,11,5,21";

Microchip attribute syn_loc : string;
              attribute syn_loc of CR_DIN : signal is "M7, Y6, B6, D10";

Xilinx   attribute syn_loc : string;
              attribute syn_loc of d_out:signal is
" P14,P12,P11, P5,P21,P13";
```

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

entity test is
    generic (s : integer := 2);

    port (
        clk: in std_logic;
        in1: in std_logic_vector(s downto 0);
        in2: in std_logic_vector(s downto 0);
        d_out: out std_logic_vector(5 downto 0));
    attribute syn_loc : string;
    attribute syn_loc of d_out:signal is "P14,P12,P11,P5,P21,P13";
end test;

architecture beh of test is
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= in1 & in2;
        end if;
    end process;
end beh;
```

Effect of Using syn_loc

This is the netlist output before the attribute is applied:

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell test (cellType GENERIC)
    (view verilog (viewType NELIST)
      (interface
        (port (array (rename a "a[2:0]" ) 3) (direction INPUT))
        (port (array (rename b "b[2:0]" ) 3) (direction INPUT))
        (port (array (rename out1 "out1[2:0]" ) 3) (direction OUTPUT))
        (port clk (direction INPUT)
      )
      (contents
        (instance clk_ibuf (viewRef PRIM (cellRef BUFG (libraryRef VIRTEX)))
        (instance clk_ibuf_iso (viewRef PRIM (cellRef IBUFG (libraryRef VIRTEX)))
      )
      (instance (rename out1Z0Z_0 "out1[0]" ) (viewRef PRIM (cellRef FD (libraryRef UNILIB)))
      )
      (instance (rename out1Z0Z_1 "out1[1]" ) (viewRef PRIM (cellRef FD (libraryRef UNILIB)))
      )
      (instance (rename out1Z0Z_2 "out1[2]" ) (viewRef PRIM (cellRef FD (libraryRef UNILIB))))
```

When the attribute is applied, the output netlist shows the LOC property:

VERILOG EDIF FILE:

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell test (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port (array (rename a "a[2:0]" 3) (direction INPUT))
          (port (array (rename b "b[2:0]" 3) (direction INPUT))
            (port (array (rename out1 "out1[2:0]" 3) (direction OUTPUT)
              (property LOC (string "P14,P12,P11"))
```

VHDL EDIF FILE:

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell test (cellType GENERIC)
    (view beh (viewType NETLIST)
      (interface
        (port clk (direction INPUT)
      )
        (port (array (rename in1 "in1[2:0]" 3) (direction INPUT))
          (port (array (rename in2 "in2[2:0]" 3) (direction INPUT))
            (port (array (rename d_out "d_out[5:0]" 6) (direction OUTPUT)
              (property LOC (string "P14,P12,P11,P5,P21,P13"))
```

syn_looplmit

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 79](#)).

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 458](#) in the *Command Reference*.

syn_looplmit Summary

Technology	Global	Object
All	Yes	Architecture

syn looplimit Syntax

VHDL	attribute syn_looplimit : integer; attribute syn_looplimit of <i>labelName</i> : label is <i>value</i> ;	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_macro

Attribute

Synplify Pro, Synplify Premier

Determines whether instantiated macros are optimized and included in the final output netlist.

Vendor	Devices
Xilinx	All

syn_macro Values

Value	Description	Default
<i>Synplify Pro</i>		
0 false	Macro netlist: It absorbs the contents of unencrypted macros into the top-level netlist. For encrypted macros, it writes out a separate encrypted netlist for the core.	Yes
1 true	Macro netlist: For both encrypted and unencrypted cores, the original unoptimized core netlist is used for place and route. Unencrypted macros are not absorbed into the top-level netlist, and no netlists are written out for encrypted cores.	

Value	Description	Default
<i>Synplify Premier</i>		
0 false	<ul style="list-style-type: none"> Macro optimization: The tool optimizes instantiated macros, but this behavior can be affected by the advanced synthesis strategy and the <i>syn_user_instance</i> attribute. See <i>syn_macro, syn_user_instance, Advanced Synthesis, and Cores</i>, on page 378. If the advanced synthesis mode is disabled, the tool does not optimize instantiated primitives with associated edn, ngo, or ngc files. It preserves the contents and the macro boundary and treats it as a white box, using the timing and resource utilization information from these files. If advanced synthesis mode is enabled and <i>syn_user_instance</i> is 0, the tool optimizes instantiated macros. 	Yes
1 true	<ul style="list-style-type: none"> Macro optimization: The tool does not optimize instantiated primitives with associated edn, ngo, or ngc files. It preserves the contents and the macro boundary and treats it as a white box, using the timing and resource utilization information from these files. Optimization is affected by the advanced synthesis mode and <i>syn_user_instance</i> attribute. See <i>syn_macro, syn_user_instance, Advanced Synthesis, and Cores</i>, on page 378. Macro netlist: Core netlist generation is affected by the <i>syn_user_instance</i> attribute. See <i>syn_macro, syn_user_instance, Advanced Synthesis, and Cores</i>, on page 378 for details. 	

Description

The *syn_macro* attribute determines whether the macros are optimized and whether the contents of instantiated macros are included in the final output netlist. You can use this attribute with IP cores (edn, ngo, ngc, or structural Verilog netlist). The attribute can only be set on an instantiated macro or IP core in a constraint file, and you must set this attribute on a view, not an instance.

The effect of `syn_macro` varies according to whether the core is encrypted or not. There are also slight differences in the way the attribute works in different FPGA synthesis tools. For details, see [syn_macro Setting and Cores, on page 378](#) and [syn_macro, syn_user_instance, Advanced Synthesis, and Cores, on page 378](#).

For more information about using the `syn_macro` attribute with the Xilinx IP flow for secure and non-secure cores, see [Including Xilinx Cores for Synthesis, on page 759](#) in the *User Guide*.

syn_macro Syntax

FDC	<code>define_attribute {v: macroName} syn_macro {1 0}</code>	Constraint Example
Verilog	<code>module /* synthesis syn_macro = 1 0 */</code>	Verilog Example
VHDL	<code>attribute syn_macro of component : label is true false;</code>	VHDL Example

Constraint Example

You must set it on the view (`v:`) for the macro or IP core; if you set it on another object, like an instance, the attribute will not work.

This example prevents optimizations for the instantiated fifo macro.

```
define_attribute {v:fifo} syn_macro {1}
```

Verilog Example

The attribute must be set on a module:

```
module /* synthesis syn_macro = 1 */
```

If the module has the `syn_black_box` attribute, apply as follows:

```
module /* synthesis syn_black_box syn_macro = 1 |0 */
```

VHDL Example

You must specify `syn_macro` on the component name. The following specification in the code will not work because it is not specified on the component name:

```
attribute syn_macro of IP1 : component is false;
```

The following code specifies `syn_macro` correctly on the component name:

```
architecture top of top is
component decoder_macro
port (clk : in bit;
      opcode : in bit_vector(2 downto 0);
      a : in bit_vector(7 downto 0);
      data0 : out bit_vector(7 downto 0));
end component;

attribute syn_macro : boolean;
attribute syn_macro of decoder_macro : label is true;
```

syn_macro Setting and Cores

Synplify Pro

The `syn_macro` setting affects how encrypted and unencrypted cores are handled in the Synplify Pro software:

Core	<code>syn_macro</code>	Core Optimization	Core Netlist
Unencrypted	0	No	Absorbed in top-level netlist
	1	No	Not absorbed in top level netlist
Encrypted	0	No	Separate encrypted core netlist
	1	No	No separate encrypted core netlist

syn_macro, syn_user_instance, Advanced Synthesis, and Cores

Synplify Premier

Like `syn_macro` the `syn_user_instance` attribute ([syn_user_instance, on page 733](#)) affects whether a macro is optimized and how core netlists are written. This tool includes both `syn_macro` and `syn_user_instance`, so follow these usage guidelines when using the two attributes:

- Use `syn_macro` to disable optimizations to the core and to prevent the core contents from being written to the top-level output netlist (unencrypted cores) or as a separate netlist (encrypted cores).
- Use `syn_user_instance` to disable optimizations to the core, but still write out the contents as appropriate for encrypted and unencrypted cores.

If both attributes are applied to the same object, the `syn_user_instance` attribute is ignored. In addition, the advanced synthesis strategy affects whether cores are optimized. The following table summarizes the behavior of these variables when applied to cores that only contain primitives:

Unencrypted Cores, advanced synthesis strategy is disabled

<code>syn_user_instance</code>	<code>syn_macro</code>	Optimization	Core Netlist
0	0	No	Absorbed in top-level netlist
0	1	No	Not absorbed in top-level netlist
1	0	No	Absorbed in top-level netlist
1	1	No	Not absorbed in top-level netlist

Unencrypted Cores, advanced synthesis strategy is enabled

<code>syn_user_instance</code>	<code>syn_macro</code>	Optimization	Core Netlist
0	0	Yes	Absorbed in top-level netlist
0	1	No	Not absorbed in top-level netlist
1	0	No	Absorbed in top-level netlist
1	1	No	Not absorbed in top-level netlist

Encrypted Cores, advanced synthesis strategy is disabled

<code>syn_user_instance</code>	<code>syn_macro</code>	Optimization	Core Netlist
0	0	No	Separate encrypted netlist
0	1	No	No separate core netlist
1	0	No	Separate encrypted netlist
1	1	No	No separate encrypted netlist

Encrypted Cores, advanced synthesis strategy is enabled

<code>syn_user_instance</code>	<code>syn_macro</code>	Optimization	Core Netlist
0	0	Yes	Separate encrypted netlist
0	1	No	No separate core netlist
1	0	No	Separate encrypted netlist
1	1	No	No separate encrypted netlist

syn_map_dffrs

Attribute

Allows the synthesis software to handle registers with both asynchronous set and asynchronous reset.

Vendor	Devices
Xilinx	Virtex 7, Virtex 6, and Spartan 6

This attribute is applicable for Xilinx Virtex-7, Virtex-6, and Spartan-6 devices only, for which registers with asynchronous set and reset are not supported.

syn_map_dffrs Values

Default	Global	Object
1/true	Yes	module

Value	Description
1/true	The software takes either the set or reset and maps it to logic which might affect the quality of results.
0/false	If the design contains registers with asynchronous set and reset, the synthesis software generates an error message.

Description

If this attribute is set to 0 when the design contains registers with asynchronous set and reset, the synthesis software generates an error message:

@W: FX707 :"dff_aload.v":5:0:5:5|Register un1_d has both asynchronous set and reset.
Your design may not work on the board. Xilinx recommends you change your RTL.

:

This warning is only issued once, although it may apply to other registers as well.

You must set the attribute `syn_map_dffrs` to 1 for the top-level design. When you set this attribute to 1, instead of generating an error, the software takes either the set or reset and maps it to logic which might affect the quality of results. The default value is 1.

Syntax Specification

FDC `define_global_attribute {syn_map_dffrs} {1|0}`

Verilog `object /* synthesis syn_map_dffrs = 1 | 0 */;`

VHDL `attribute syn_map_dffrs of object : objectType is true | false;`

SCOPE Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>		<global>	<code>syn_map_dffrs</code>	0		

Verilog Example

```
module dff_aload (q, d, ad, clk, load)
/*synthesis syn_map_dffrs = 0*/;
input d, ad, clk, load;
output q;
reg q;

always @ (posedge clk or posedge load)
begin
    if (load)
        q = ad;
    else
        q = d;
end
endmodule
```

VHDL Example

`attribute syn_map_dffrs of object : objectType is 1;`

```

library ieee;
use ieee.std_logic_1164.all;
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY clatches IS
    PORT (d,c,r,s : IN std_logic;
          qrs: OUT std_logic);
END clatches;

ARCHITECTURE behave OF clatches IS
    SIGNAL qrs_i: std_logic;
    attribute syn_map_dffrs : boolean;
    attribute syn_map_dffrs of behave : architecture is true;
BEGIN
    qrs_i <= '0' WHEN r = '1' ELSE
    '1' WHEN s = '1' ELSE
        d WHEN c = '1' ELSE
        qrs_i;
    qrs <= qrs_i;
END BEHAVE;

```

Effect of using syn_map_dffrs

The following shows the content in log file, before applying the attribute:

Verilog moduledff_aload(q, d, ad, clk, load) /*synthesis syn_map_dffrs = 1*/;

VHDL attribute syn_map_dffrs of behave: architecture is true;

```
# Tue Jun 12 11:40:00 2012
#####
Premap Report
Synopsys Xilinx Technology Pre-mapping, Version map610dev, Build 613R, Built Jun 8 2012 22:05:25
Copyright (C) 1994-2012, Synopsys, Inc. This software and the associated documentation are confidential and proprietary to Synopsys, Inc. Your use or disclosure of this software subject to the terms and conditions of a written license agreement between you, or your company, and Synopsys, Inc.
Product Version F-2012.09

Mapper Startup Complete (Time elapsed 0h:00m:00s; Memory used current: 91MB peak: 92MB)
Reading constraint file: C:\syn_map_dffrs\syn_map_dffrs_0.sdc
Adding property syn_map_dffrs, value 1 to viewwork.dff_aload(verilog)
BL: C:\syn_map_dffrs\rev_1\syn_map_dffrs_sck.rpt
Printing Clock summary report in "C:\syn_map_dffrs\rev_1\syn_map_dffrs_sck.rpt" file
#N: M531 [Unused] in port ad
#N: M532 [Unused] in port load
#N: M537 Generated clock conversion enabled
#N: M546 Generated clock conversion enabled

Design Input Complete (Time elapsed 0h:00m:00s; Memory used current: 91MB peak: 92MB)
Mapper Initialization Complete (Time elapsed 0h:00m:00s; Memory used current: 92MB peak: 92MB)
Start Loading timing files (Time elapsed 0h:00m:00s; Memory used current: 92MB peak: 92MB)
Finished Loading timing files (Time elapsed 0h:00m:00s; Memory used current: 92MB peak: 94MB)

Reading Xilinx I/O pad type table from file <C:\ntsyn202209_0988\lib\xilinx\v1_0_iob.txt>
Reading Xilinx Rocket I/O parameter type table from file <C:\ntsyn202209_0988\lib\xilinx\gttype.txt>
#W: FX07 : "c:\syn_map_dffrs\syn_map_dffrs.v":6:0:5|Register uni_d has both asynchronous set and reset. Your design may not work on the board. Xilinx recommends you change your RTL.

Clock Summary
#####
Start          Requested      Requested      Clock      Clock
Clock          Frequency     Period        Type       Group
#####
System_dff_aload|clk 1.0 MHz    1000.000   system    system_clkgroup
System_dff_aload|clk 1.0 MHz    1000.000   inferred  inferred_clkgroup_0

#W: HT529 : "c:\syn_map_dffrs\syn_map_dffrs.v":6:0:6:5|Found inferred clock dff_aload|clk which controls 2 sequential elements including uni_d. This clock has no specified timing constraint which may prevent conversion of gated or generated clocks and may adversely impact design performance.

syn_allowed_resources : blockram=2060 set on top level netlist dff_aload
Finished Pre Mapping Phase. (Time elapsed 0h:00m:00s; Memory used current: 174MB peak: 200MB)
#N: BN225 |Writing default property annotation file C:\Users\anantnag\attributes\syn_map_dffrs\xilinx\syn_map_dffrs_0\rev_1\syn_map_dffrs.sap.
Pre-mapping successful!

At Mapper Exit (Time elapsed 0h:00m:00s; Memory used current: 112MB peak: 200MB)
Process took 0h:00m:01s realtime, 0h:00m:01s cputime
# Tue Jun 12 11:40:03 2012
#####
```

The following shows the content in log file, after applying the attribute:

Verilog module dff_aload(q, d, ad, clk, load) /*synthesis syn_map_dffrs = 0*/;

VHDL attribute syn_map_dffrs of behave: architecture is false;

```
# Tue Jun 12 11:40:00 2012
#####
# Premap Report
#
# Copyright (C) 1994-2012, Synopsys, Inc. This software and the associated documentation are confidential and proprietary to Synopsys, Inc. Your use or disclosure of this software subject to the terms and conditions of a written license agreement between you, or your company, and Synopsys, Inc.
# Product Version F-2012.09

Mapper Startup Complete (Time elapsed 0h:00m:00s; Memory used current: 91MB peak: 92MB)
Reading constraint file: C:\syn_map_dffrs\syn_map_dffrs_0.sdc
Adding property syn_map_dffrs, value 1 to viewwork.dff_aload(verilog)
@l: C:\syn_map_dffrs\rev_1\syn_map_dffrs.scck.rpt
Property clock conversion in "C:\syn_map_dffrs\rev_1\syn_map_dffrs_scck.rpt"
@W: MF248 | Running in 64-bit mode
@N: MF257 | Gated clock conversion enabled
@N: MF546 | Generated clock conversion enabled

Design Input Complete (Time elapsed 0h:00m:00s; Memory used current: 91MB peak: 92MB)
Mapper Initialization Complete (Time elapsed 0h:00m:00s; Memory used current: 91MB peak: 92MB)
Start loading timing files (Time elapsed 0h:00m:00s; Memory used current: 92MB peak: 92MB)
Finished Loading timing files (Time elapsed 0h:00m:00s; Memory used current: 92MB peak: 94MB)

Reading Xilinx I/O pad Type table from file <C:\nt\xlmn00209_0988\lib\xilinxx_iq.tbt>
Reading Xilinx Rocket I/O parameter type table from file <C:\nt\xlmn00209_0988\lib\xilinxx_gtttype.txt>
@W: FX07? :<C:\syn_map_dffrs\syn_map_dffrs.v>:6:0:6:5|Register unl_d has both asynchronous set and reset. Your design may not work on the board. Xilinx recommends you change your RTL.

Clock Summary
*****
Start Clock Requested Frequency Requested Period Clock Type Clock Group
-----
System dff_aload[clk] 1.0 MHz 1000.000 system inferred system_clkgroup_0
-----

@W: MF529 :<C:\syn_map_dffrs\syn_map_dffrs.v>:6:0:6:5|Found inferred clock dff_aload[clk] which controls 2 sequential elements including unl_d. This clock has no specified timing constraint which may prevent conversion of gated or generated clocks and may adversely impact design performance.

syn_allowed_resources : blockrams=2060 set on top level netlist dff_aload

Finished Pre Mapping Phase. (Time elapsed 0h:00m:00s; Memory used current: 174MB peak: 200MB)
@N: BN225 |Writing default property annotation file C:\Users\anantnag\attributes\syn_map_dffrs\xilinx\syn_map_dffrs_0\rev_1\syn_map_dffrs.sap.
Pre-mapping successful

At Mapper Exit (Time elapsed 0h:00m:00s; Memory used current: 112MB peak: 200MB)
Process took 0h:00m:01s realtime, 0h:00m:01s cputime
# Tue Jun 12 11:40:03 2012
#####
#
```


syn_max_memsize_reg

Attribute

The `syn_max_memsize_reg` attribute allows you to specify the maximum number of registers that can be mapped to an inferred RAM above the limit for which the tool errors out. The default value is 128.

Vendor	Technologies
Achronix	Speedster7t and newer families

syn_max_memsize_reg Values

Value	Description	Default	Global
<code>integerValue</code>	Specifies the maximum number of registers that can be mapped to an inferred RAM above the limit for which an error is generated.	128	Yes

Description

The `syn_max_memsize_reg` attribute allows you to specify the maximum number of registers that can be mapped to an inferred RAM above the limit for which the tool errors out. The default value is 128.

You can apply the `syn_max_memsize_reg` attribute globally in the top-level design to change the default threshold to a new value above the limit for which the tool generates an error message.

Note: If the `syn_ramstyle` attribute is set to "registers" for an inferred RAM, then the `syn_max_memsize_reg` attribute has no effect on the RAM and the tool always maps the RAM to registers.

syn_max_memsize_reg Syntax

FDC	define_global_attribute syn_max_memsize_reg {integer}	FDC Example
Verilog	object /*synthesis syn_max_memsize_reg={integer}*/	Verilog syn_max_memsize _reg Example
VHDL	attribute syn_max_memsize_reg: integer; attribute syn_max_memsize_reg of {object}:architecture is {integer};	VHDL syn_max_memsize _reg Example

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	global	<Global>	syn_max_memsize_reg	100	integer	Defines the threshold value of memory size for memory to be mapped to registers; beyond which tool will error out
2							

```
define_global_attribute syn_max_memsize_reg {1030}
```

Verilog syn_max_memsize_reg Example

```
//Example: Verilog syn_max_memsize_reg

module test (data0,data1,waddr0,waddr1,we0,we1,clk,q0, q1)/*
synthesis syn_max_memsize_reg=1030 */;

`else

module test_RTL (data0,data1,waddr0,waddr1,we0,we1,clk,q0, q1);

`endif


parameter d_width = 1;
parameter addr_width = 10 ;
parameter mem_depth = 1024;

input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
```

```
input we0, we1, clk;

reg [d_width-1:0] mem [mem_depth-1:0]/* synthesis
syn_ramstyle=registers */;
reg [addr_width-1:0] reg_waddr0, reg_waddr1;
output [d_width-1:0] q0, q1;

assign q0 = mem[waddr0];
assign q1 = mem[waddr1];

always @ (posedge clk)
begin
//reg_waddr0 <= waddr0;
//reg_waddr1 <= waddr1;
    if (we0) mem[waddr0] <= data0;
    if (we1) mem[waddr1] <= data1;
end

endmodule
```

Apply the `syn_max_memsize_reg` attribute globally on the top-level module.

VHDL `syn_max_memsize_reg` Example

```
--Example: VHDL syn_max_memsize_reg
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity test is
generic
(d_width : integer := 1;
addr_width : integer := 10;
mem_depth : integer :=1023);
Port (
    clk : in std_logic;
    we0,we1 : in std_logic;
    data0, data1 : in STD_LOGIC_VECTOR ((d_width-1) downto 0);
    waddr0, waddr1 : in STD_LOGIC_VECTOR ((addr_width-1) downto 0);
    q0, q1 : out STD_LOGIC_VECTOR ((d_width-1) downto 0)
);
end test;

architecture Behavioral of test is
type mem_type is array(0 to mem_depth) of std_logic_vector
((d_width-1) downto 0);
signal mem : mem_type;
attribute syn_max_memsize_reg : integer;
attribute syn_max_memsize_reg of Behavioral: architecture is 1030;
begin
```

```
q0<=mem(conv_integer(waddr0));  
q1<=mem(conv_integer(waddr1));  
process (clk)  
begin  
  if (rising_edge(clk)) then  
    if (we0 = '1') then  
      mem(conv_integer(waddr0)) <= data0;  
    end if;  
    if (we1 = '1') then  
      mem(conv_integer(waddr1)) <= data1;  
    end if;  
  end if;  
end process;  
end Behavioral;
```

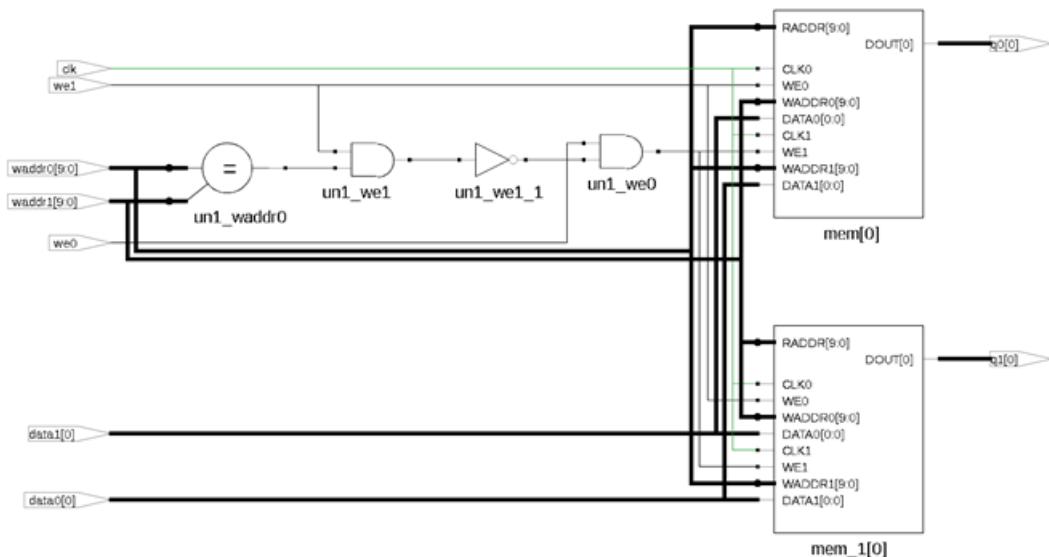
Apply the `syn_max_memsize_reg` attribute globally on the top-level module.

Effect of Using `syn_max_memsize_reg`

When the `syn_max_memsize_reg` attribute is not used, the software generates an error if the RAM size is greater than the default threshold of 128. For example:

```
module test (data0,data1,waddr0,waddr1,we0,we1,clk,q0,q1);
```

The RAM of size 1024x1 shown in the HDL Analyst view below should get mapped to registers. However, an error is generated because its size is greater than the default threshold.



When you apply the `syn_max_memsize_reg` attribute with a value of 1030 as shown below, the software maps the RAM to registers.

```
module test (data0,data1,waddr0,waddr1,we0,we1,clk,q0, q1)
/* synthesis syn_max_memsize_reg=1030 */;
```

syn_maxfan

Attribute

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

Vendor	Technology	Default
Achronix	All	None
Intel FPGA	All	None
Lattice	ECP3, ECP2S, ECP2M, ECP2 ECP/EC XP2/XP SC/SCM, MachXO, ORCA and later families	None
Microchip	All	None
Xilinx	All	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.
 - The attribute value functions as a hard limit when it is attached to nets, ports, primitive instances, and registers in the designs. See [Setting Fanout Limits, on page 577](#) of the *User Guide* for details.

You can apply the `syn_maxfan` attribute to the following objects:

- Registers or instances.
- Ports or nets. If you apply the attribute to a net, the synthesis tool creates a `KEEPBUF` component and attaches the attribute to it to prevent the net itself from being optimized away during synthesis.

The `syn_maxfan` attribute is often used along with the `syn_noclockbuf` attribute on an input port that you do not want buffered. There are a limited number of clock buffers in a design, so if you want to save these special clock buffer resources for other clock inputs, put the `syn_noclockbuf` attribute on the clock signal. If timing for that clock signal is not critical, you can turn off buffering completely to save area. To turn off buffering, set the maximum fanout to a very high number; for example, 1000. Note, do not use the `syn_maxfan` attribute with the fast synthesis option.

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 define_attribute {object} syn_maxfan {integer}

[FDC Example](#)

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

[Verilog Example](#)

VHDL attribute syn_maxfan of object : objectType is "value";

[VHDL Example](#)

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 value of 1.

Verilog Example

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

For example:

```
module syn_maxfan (clk,rst,a,b,c,y);
  input clk,rst;
  input [7:0] a,b;
  output reg [7:0] c,y;

  reg d/* synthesis syn_maxfan=3 */;
  always @ (posedge clk)
    begin
      if(rst)
        d <= 0;
      else
        d <= ~d;
    end

  always @ (posedge d)
    c <= a&b;
endmodule
```

```
always @ (posedge clk)
begin
    if(d)
        y<=0;
    else
        y <= a&b;
end
endmodule
```

VHDL Example

```
attribute syn_maxfan of object : objectType is "value";
```

See [VHDL Attribute and Directive Syntax, on page 401](#) for different ways to specify VHDL attributes and directives.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity test is
generic (n : integer := 10;
         m : integer := 7
);
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 test;

architecture rtl of test is
signal d : std_logic;

attribute syn_maxfan : integer;
attribute syn_maxfan of d : signal is (n-m);

begin

process (clk)
begin
    if (clk'event and clk = '1') then
        if (rst = '1') then
            d <= '0';
        else
            d <= a and b;
        end if;
    end if;
end process;
end;
```

```

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

process (clk)
begin

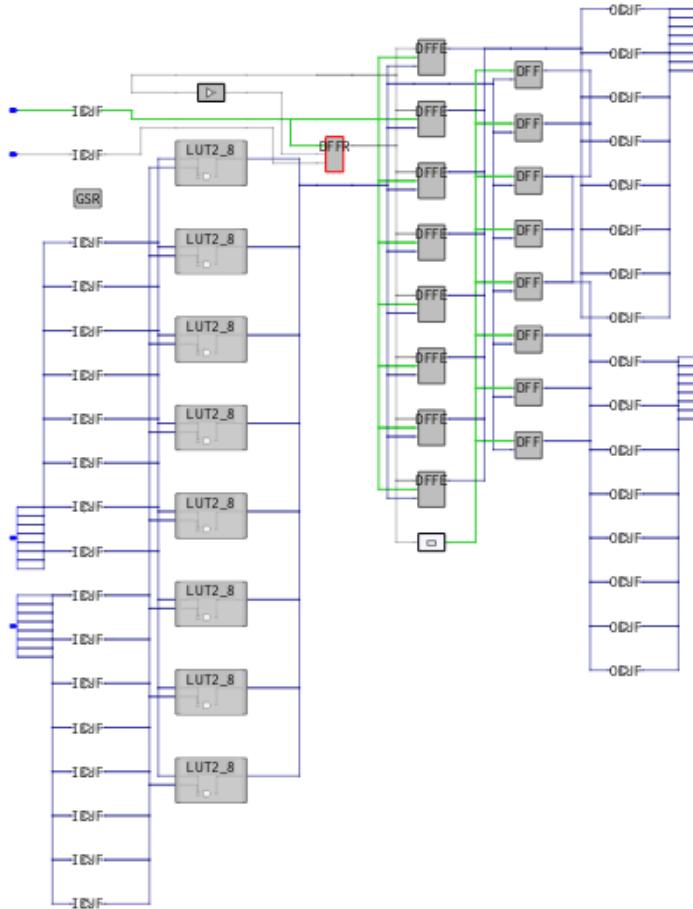
if (clk'event and clk = '1') then
    if (d = '1') then
        c <= a and b;
    end if;
    end if;
end process;

end rtl;

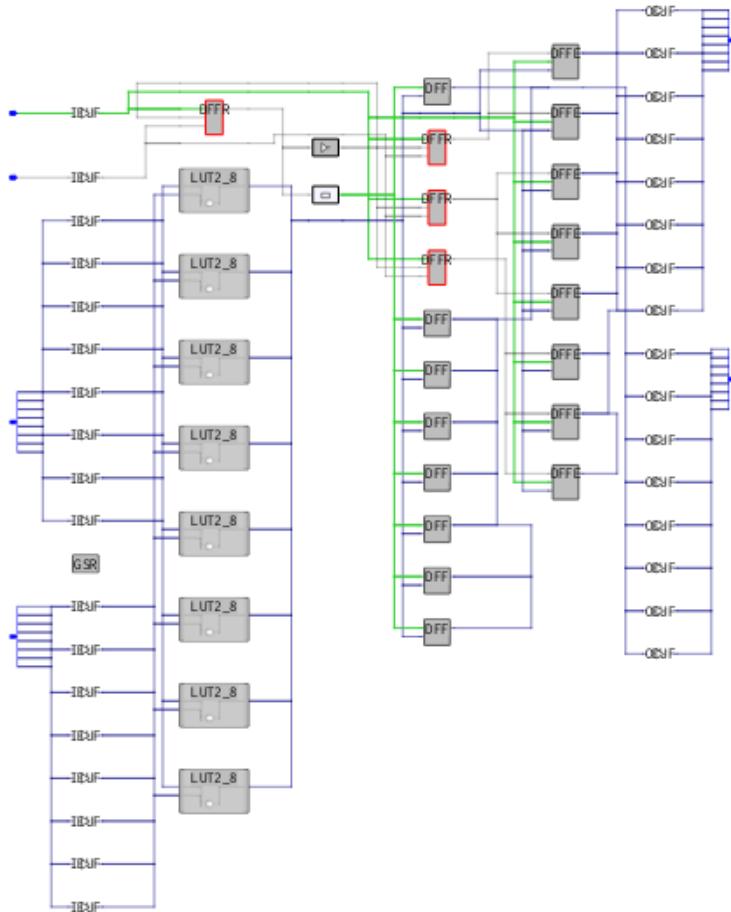
```

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.



syn_multstyle

Attribute

Determines how multipliers are implemented.

Vendor	Device	Values
Intel FPGA	Stratix and Cyclone families	block_mult logic lpm_mult simd
Lattice	ECP3/ECP2S/ECP2M/ECP2 ECP/EC XP2/XP MACH XO, ORCA and newer families	block_mult logic
	iCE40UP, LIFCL (Lattice Radiant software)	DSP logic
Microchip	SmartFusion2 IGLOO2 Axcelerator RTAX2000D Axcelerator RTAX4000D and newer families	dsp logic
Xilinx	Virtex-II and newer families	block_mult logic

syn_multstyle Values

Value	Description	Default
block_mult	<i>Intel FPGA, Xilinx, Lattice</i> Implements the multipliers as dedicated hardware blocks: Intel FPGA: DSP MACs Xilinx: Block mults Lattice: DSP blocks	X
logic	Implements the multipliers as logic.	-
dsp	<i>Microchip</i> Implements the multipliers as DSP blocks.	X
simd:n	Implements the multipliers in Single Instruction Multiple Data (SIMD) mode, in PolarFire devices. n denotes an integer value.	
lpm_mult	<i>Intel FPGA</i> Implements the multipliers as LPMs.	-
simd / simd: <i>i</i>	<i>Intel FPGA</i> Specifies the DSP configuration available in SIMD mode.	-

This table lists the valid values for each vendor:

Intel Stratix and Cyclone families	<ul style="list-style-type: none">• block_mult Uses DSP MAC blocks. This is the default.• lpm_mult Uses LPMs. If you want an lpm_mult implementation, you must explicitly apply this value; if you do not specify this, the software implements the multipliers as alt_mult_add. <p>Limitations: No support for packing of registers and pipeline stages inside the lpm_mult implementation when syn_multstyle is applied. The attribute is honored only for multipliers with width less than or equal to 256.</p> <ul style="list-style-type: none">• logic Uses logic instead of dedicated resources.• simd / simd:i Implements two multipliers in a DSP block. For details, see syn_multstyle in Intel FPGA SIMD Mode, on page 408
Lattice	<ul style="list-style-type: none">• block_mult Uses dedicated hardware DSP blocks. This is the default.• logic Uses logic instead of dedicated resources.
Microchip	<ul style="list-style-type: none">• dsp Uses dedicated hardware DSP blocks. This is the default.• logic Uses logic instead of dedicated resources.
Xilinx Virtex-II and later families	<ul style="list-style-type: none">• block_mult Uses block multipliers. This is the default.• logic Uses logic instead of dedicated resources.

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 and syn_DSPstyle in Xilinx Designs

syn_multstyle results in the same logic or block multiplier implementations as syn_DSPstyle ([syn_DSPstyle, on page 219](#)) when they are applied on simple multipliers in newer technologies that have dedicated DSP resources. Typically, you use syn_multstyle for older technologies that do not support DSPs.

syn_multstyle Syntax

Global Attribute Object

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

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

FDC	<code>define_attribute {instance} syn_multstyle {block_mult logic dsp lpm_mult}</code>	SCOPE Example
	Global attribute: <code>define_global_attribute syn_multstyle {block_mult logic dsp lpm_mult}</code>	
Verilog	<code>input net /* synthesis syn_multstyle = "block_mult logic dsp lpm_mult" */;</code>	Verilog Example
VHDL	<code>attribute syn_multstyle of instance : signal is "block_mult logic dsp lpm_mult";</code>	VHDL Example

See [VHDL Attribute and Directive Syntax](#), on page 401 for different ways to specify VHDL attributes and directives.

SCOPE Example

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

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	<Glob...	syn_multstyle	logic	string	Special implementation of multipliers
2							

This example specifies that multipliers be implemented as logic.

```
define_attribute {temp[15:0]} syn_multstyle {logic}
```

Verilog Example

```
module mult(a,b,c,r,en);
  input [7:0] a,b;
  output [15:0] r;
  input [15:0] c;
  input en;
  wire [15:0] temp /* synthesis syn_multstyle="logic" */;
  assign temp = a*b;
  assign r = en ? temp: c;
endmodule
```

VHDL Example

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

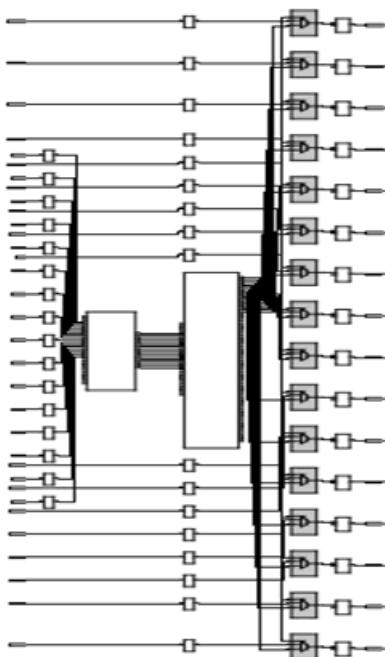
entity mult is
  port (clk : in std_logic;
        a : in std_logic_vector(7 downto 0);
        b : in std_logic_vector(7 downto 0);
        c : out std_logic_vector(15 downto 0))
end mults;

architecture rtl of mult is
signal mult_i : std_logic_vector(15 downto 0);
attribute syn_multstyle : string;
attribute syn_multstyle of mult_i : signal is "logic";
begin
mult_i <= std_logic_vector(unsigned(a)*unsigned(b));
process(clk)
begin
  if (clk'event and clk = '1') then
    c <= mult_i;
  end if;
end process;
end rtl;
```

Effect of Using syn_multstyle in an Intel FPGA Design

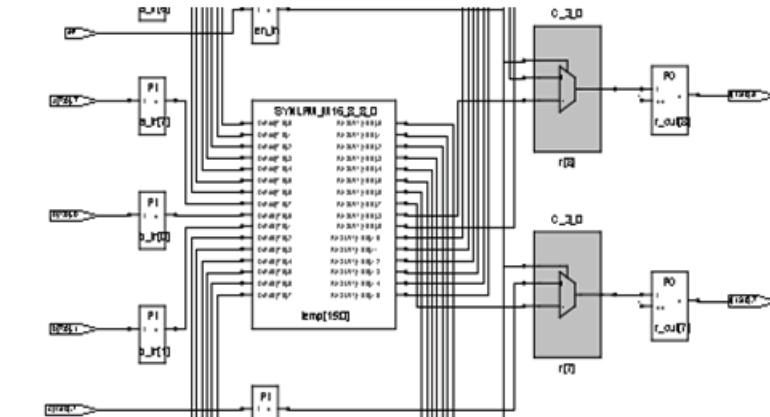
The following figure shows the multiplier implemented as a DSP block in an Intel FPGA design, when the attribute is set to block_mult:

Verilog	wire [15:0] temp /* synthesis syn_multstyle = "block_mult" */;
VHDL	attribute syn_multstyle of mult_i : signal is "block_mult";



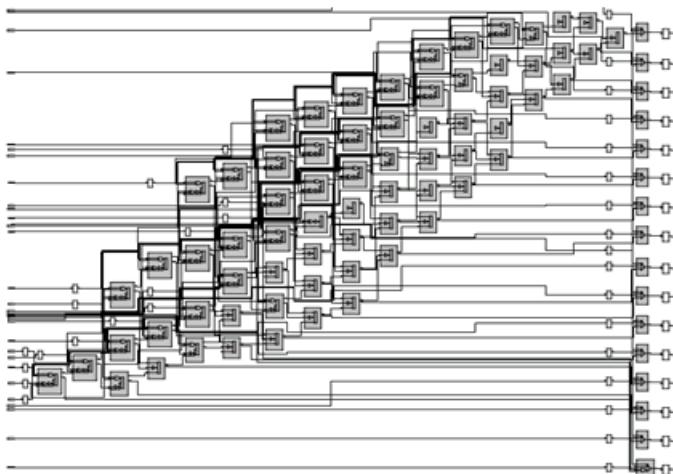
The next figure shows the same Intel FPGA design with the multiplier implemented as LPM when the attribute is set to lpm_mult:

Verilog	<code>wire [15:0] temp /* synthesis syn_multstyle = "lpm_mult" */;</code>
VHDL	<code>attribute syn_multstyle of mult_i : signal is "lpm_mult";</code>



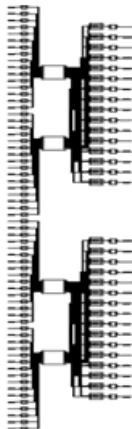
The following figure shows the Intel FPGA design with the multiplier implemented as logic when the attribute is set to logic:

Verilog	<code>wire [15:0] temp /* synthesis syn_multstyle = "logic" */;</code>
VHDL	<code>attribute syn_multstyle of mult_i : signal is "logic";</code>



The next figure shows the effect of applying the `lpm_mult` value globally to the Intel FPGA design. All four multipliers are implemented as LPMs.

HDL	<code>module mult (a,b,c,de,e,f,g,h,r1,r2 /* synthesis syn_multstyle = "lpm_mult" */;</code>
FDC	<code>define_global_attribute syn_multstyle {lpm_mult}</code>



syn_multstyle in Intel FPGA SIMD Mode

Arria 10, Arria V, and Cyclone V

For Intel FPGA designs, the value of this attribute can be either `simd` or `simd:i`. The object you apply the attribute to affects what gets mapped into the DSP resource. The following table provides details:

Object	Behavior
<code>simd</code>	Specifies the DSP configuration available in Single Instruction Multiple Data (SIMD) mode, and leaves it to the synthesis software to determine how to pack two multipliers into one DSP block.
<code>simd:<i>i</i></code>	Specifies the DSP configuration available in SIMD mode. <i>i</i> is an index value that is a valid integer. Specify an index value to pair and pack two multipliers into one DSP block.

Single Instruction Multiple Data (SIMD) mode is supported for newer Intel FPGA devices. In SIMD mode, the Intel FPGA DSP structure supports packing two multipliers into one DSP block as shown in the table below:

Technology	DSP Modes Supported ...
Arria 10/Arria V/ Cyclone V	M18X18_FULL Two independent M18X18 (signed/unsigned) or M18X19 (signed) multiplications with/without pre-adders/subtractors in a single DSP block with full 37-bit outputs for each multiplication

Warning messages are generated for the following conditions:

- Technology was specified for other than the supported devices.
 - If the multiplier specified with SIMD mode is greater than M18X19 (signed) and M18X18 (unsigned), then SIMD packing cannot be performed.
 - When a multiplier is followed by an adder, mult-add has higher priority than SIMD packing. SIMD packing does not occur; the mult-add gets mapped instead.

syn_multstyle Syntax in Intel FPGA SIMD Mode

Global Support Objects

Yes Multipliers, instances, modules, architectures

The following table shows how to specify the attribute in different files:

FDC `define_attribute {object} syn_multstyle {simd | simd: i}` **SCOPE Example**
 `define_global_attribute syn_multstyle {simd | simd: i}`

Verilog *object* /* synthesis syn multstyle = "simd | simd:i" */;

VHDL attribute syn multstyle of object : *objectType* is simd | SIMD:i;

SCOPE Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	<Global>	syn_multstyle	simd	string	Inferred multiplier implementation style
2							

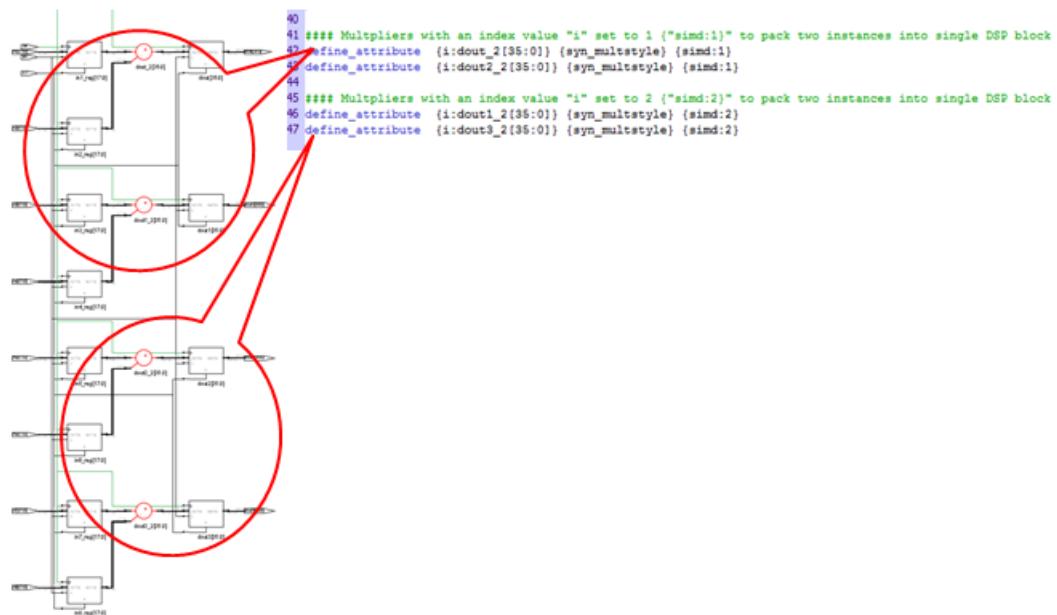
```
define global attribute {syn multstyle} {simd}
```

Effect of Using syn_multstyle in Intel FPGA SIMD Mode

For this example only one DSP block is implemented for the RTL M18X18_FULL multipliers when SIMD is specified globally.



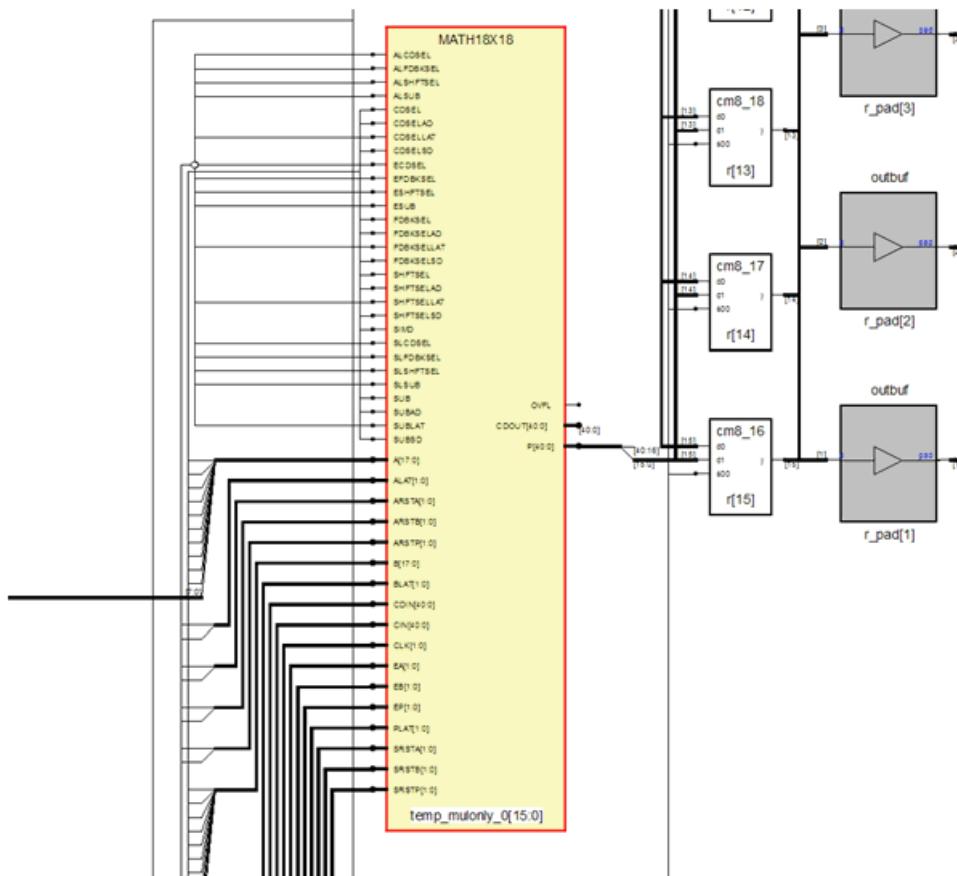
The example below shows how two DSP blocks are implemented for the RTL M18X18_FULL multipliers instead of four when SIMD is applied on the instances.



Effect of Using syn_multstyle in a Microchip Design

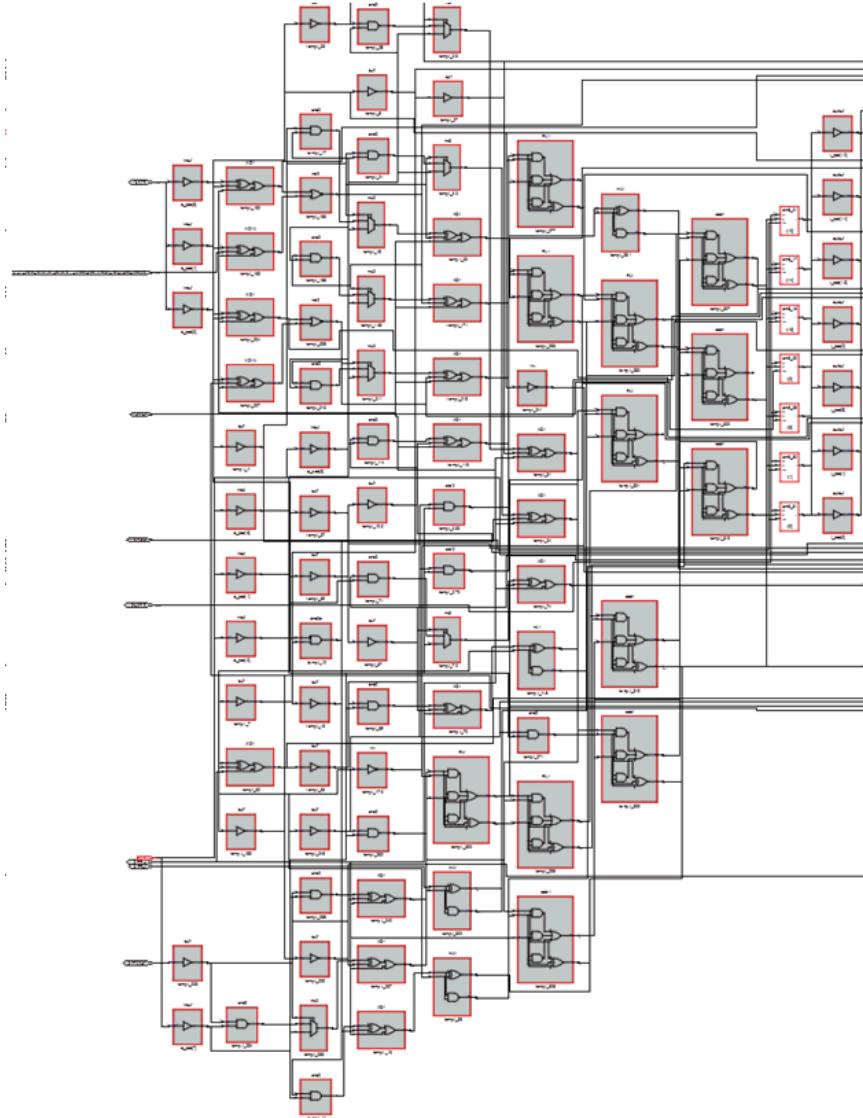
In a Microchip design, you can specify that the multipliers be implemented as logic or as dedicated DSP blocks. The following figure shows a multiplier implemented as DSP:

Verilog	<code>wire [15:0] temp /* synthesis syn_multstyle = "dsp" */;</code>
VHDL	<code>attribute syn_multstyle of mult_i : signal is "dsp";</code>



The following figure shows the same Microchip design with the multiplier implemented as logic when the attribute is set to logic:

Verilog	wire [15:0] temp /* synthesis syn_multstyle = "logic" */;
VHDL	attribute syn_multstyle of mult_i : signal is "logic";



syn_netlist_hierarchy

Attribute

Determines if the generated netlist is to be hierarchical or flat.

Vendor	Technology
Achronix	Speedster families
Intel FPGA	Stratix, Cyclone, Arria, and newer families
Lattice	EC, SC, XP, ORCA, isp and newer families
Microchip	ProASIC, IGLOO and newer families
Xilinx	Virtex and newer families Spartan and newer 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 **define_global_attribute syn_netlist_hierarchy {0|1}**

[SCOPE Example](#)

Verilog **object /* synthesis syn_netlist_hierarchy = 0|1 */;**

[Verilog Example](#)

VHDL **attribute syn_netlist_hierarchy of object : objectType is true|false;**

[VHDL Example](#)

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 4

module rca_adder#(parameter width =4)
    (input[width-1:0]A,B,input CIN,
     output[width-1:0]SU,output COUT);
wire[width-2:0]CY;
fu_add FA0(.su(SU[0]),.cy(CY[0]),.cin(CIN),.a(A[0]),.b(B[0]));
fu_add FA1(.su(SU[1]),.cy(CY[1]),.cin(CY[0]),.a(A[1]),.b(B[1]));
fu_add FA2(.su(SU[2]),.cy(CY[2]),.cin(CY[1]),.a(A[2]),.b(B[2]));
fu_add FA3(.su(SU[3]),.cy(COUT),.cin(CY[2]),.a(A[3]),.b(B[3]));
endmodule

module rp_top#(parameter width =16)
    (input[width-1:0]A1,B1,input CIN1,
     output[width- 1:0]SUM,output COUT1) /*synthesis
                                              syn netlist_hierarchy=0*/;
wire[2:0]CY1;
rca_adder RA0 (.SU(SUM[3:0]),.COUT(CY1[0]),.CIN(CIN1),
               .A(A1[3:0]),.B(B1[3:0]));
rca_adder RA1(.SU(SUM[7:4]),.COUT(CY1[1]),.CIN(CY1[0]),
               .A(A1[7:4]),.B(B1[7]));
rca_adder RA2 (.SU(SUM[11:8]),.COUT(CY1[2]),.CIN(CY1[1]),
               .A(A1[11:8]),.B(B1[11:8]));
rca_adder RA3(.SU(SUM[15:12]),.COUT(COUT1),.CIN(CY1[2]),
               .A(A1[15:12]),.B(B1[15:12]));
endmodule
```

VHDL Example

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

entity FULLADDER is
    port (a, b, c : in std_logic;
          sum, carry: out std_logic);
end FULLADDER;

architecture fulladder_behav of FULLADDER is
begin
    sum <= (a xor b) xor c ;
    carry <= (a and b) or (c and (a xor b));
end fulladder_behav;

library ieee;
use ieee.std_logic_1164.all;

entity FOURBITADD is
    port (a, b : in std_logic_vector(3 downto 0);
          Cin  : in std_logic;
          sum  : out std_logic_vector (3 downto 0);
          Cout, V : out std_logic);
end FOURBITADD;

architecture fouradder_structure of FOURBITADD is
signal c: std_logic_vector (4 downto 1);
component FULLADDER
    port (a, b, c: in std_logic;
          sum, carry: out std_logic);
end component;
begin
    FA0: FULLADDER
        port map (a(0), b(0), Cin, sum(0), c(1));
    FA1: FULLADDER
        port map (a(1), b(1), C(1), sum(1), c(2));
    FA2: FULLADDER
        port map (a(2), b(2), C(2), sum(2), c(3));
    FA3: FULLADDER
        port map (a(3), b(3), C(3), sum(3), c(4));
    V <= c(3) xor c(4);
    Cout <= c(4);
end fouradder_structure;
```

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

entity BITADD is
    port (A, B: in std_logic_vector(15 downto 0);
          Cin : in std_logic;
          SUM : out std_logic_vector (15 downto 0);
          COUT: out std_logic);
end BITADD;

architecture adder_structure of BITADD is
attribute syn_netlist_hierarchy : boolean;
attribute syn_netlist_hierarchy of adder_structure:
    architecture is false;
signal C: std_logic_vector (4 downto 1);

component FOURBITADD
    port (a, b: in std_logic_vector(3 downto 0);
          Cin : in std_logic;
          sum : out std_logic_vector (3 downto 0);
          Cout, V: out std_logic);
end component;

begin
    F1: FOURBITADD
        port map (A(3 downto 0),B(3 downto 0),
                  Cin, SUM(3 downto 0),C(1));
    F2: FOURBITADD
        port map (A(7 downto 4),B(7 downto 4),
                  C(1), SUM(7 downto 4),C(2));
    F3: FOURBITADD
        port map (A(11 downto 8),B(11 downto 8),
                  C(2), SUM(11 downto 8),C(3));
    F4: FOURBITADD
        port map (A(15 downto 12),B(15 downto 12),
                  C(3), SUM(15 downto 12),C(4));
    COUT <= c(4);
end adder_structure;
```

Effect of Using syn_netlist_hierarchy

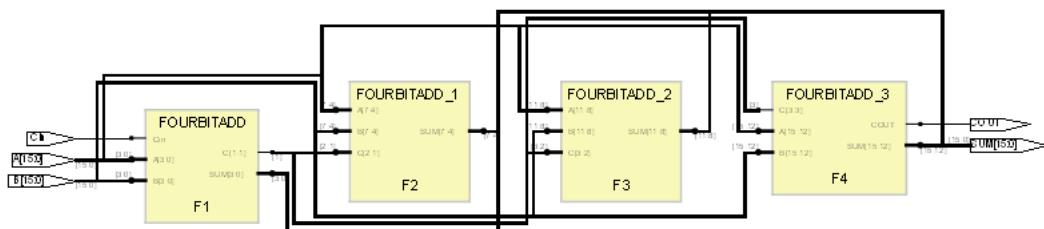
Without applying the attribute (default is to allow hierarchy generation) or setting the attribute to 1/true creates a hierarchical netlist.

Verilog

```
output [width-1:0]SUM, output COUT1)
/*synthesis syn_netlist_hierarchy=1*/;
```

VHDL

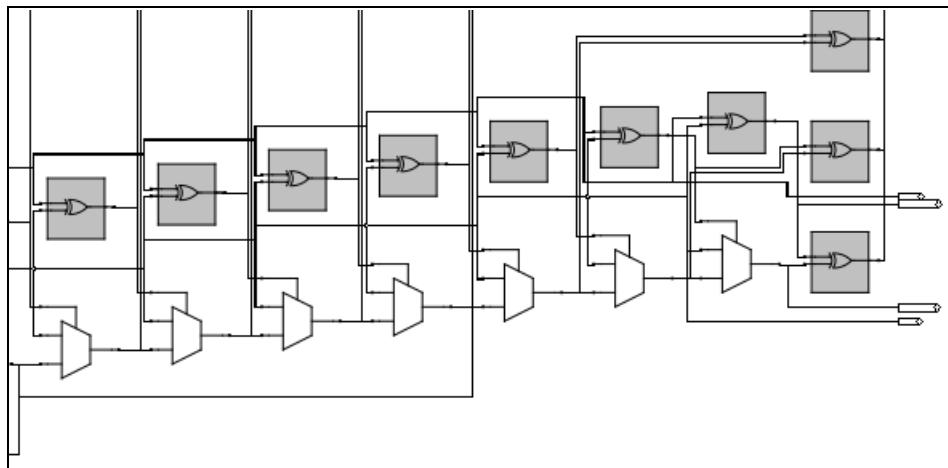
```
attribute syn_netlist_hierarchy of adder_structure :
architecture is true;
```



Applying the attribute with a value of 0/false creates a flattened netlist.

Verilog output [width-1:0] SUM, output COUT1)
/*synthesis syn_netlist_hierarchy=0*/;

VHDL attribute syn_netlist_hierarchy of adder_structure :
architecture is false;



syn_hier flatten and syn_netlist_hierarchy

The `syn_hier=flatten` attribute and the `syn_netlist_hierarchy=false` attributes both flatten hierarchy, but work slightly differently. Use the `syn_netlist_hierarchy` attribute if you want a completely flattened netlist (this attribute flattens all levels of hierarchy). When you set `syn_hier=flatten`, you flatten the hierarchical levels below the component on which it is set, but you do not flatten the current hierarchical level where it is set. Refer to [syn_hier, on page 319](#) for information about this attribute.

syn_no_compile_point

Attribute

Use this attribute with the Automatic Compile Point (ACP) feature. The software automatically identifies modules as compile points in the design based on its size, number of I/Os, and hierarchical levels. However, if you do not want the software to create a compile point for a particular view or module, then apply this attribute.

Vendor	Technology
Achronix	Speedster7t family
Intel FPGA	Cyclone III and Stratix III and newer families
Xilinx	Spartan-3 and Virtex-4 and newer families

syn_no_compile_point Values

Global Support	Default	Object
No	0 false	Module or architecture

Description

Use this attribute when the Auto Compile Point option is enabled. The software automatically identifies modules as compile points in the design based on its size, number of I/Os, and hierarchical levels. For details about this feature, see the [The Automatic Compile Point Flow, on page 627](#).

However, if you do not want the software to create a compile point for a particular view or module, then apply this attribute. This design view or module is ignored by the Automatic Compile Point software, ensuring that a compile point is not generated for it during synthesis. You must explicitly set

this attribute to 1 or true. When you specify syn_no_compile_point on a module, be aware that this does not prevent ACP from identifying compile points for other modules instantiated within that module.

syn_no_compile_point Syntax

The following table summarizes the syntax in different files.

FDC	define_attribute {v: <i>moduleName</i> } syn_no_compile_point {0 1}	FDC Example
Verilog	<i>object</i> /* synthesis syn_no_compile_point = 0 1 */;	Verilog Example
VHDL	attribute syn_no_compile_point : boolean; attribute syn_no_compile_point of <i>object</i> : <i>objectType</i> is false true;	VHDL Example

Where:

- *object* must be a view with the syntax v:*moduleName*
- *value* must be 1 or true

```
define_attribute {v:fifo} syn_no_compile_point {1}
```

You cannot apply this attribute globally.

FDC Example

Enable	Object Type	Object	Attribute	Value	Value Type	Description
<input checked="" type="checkbox"/>	<any>	vwork.prgm_cntr	syn_no_compile_point	1	boolean	Do not mark the view as compile-point

```
define_attribute {v:work.prgm_cntr} {syn_no_compile_point} {1}
```

Verilog Example

The following Verilog code segment contains the module, mult, which should not be treated as a compile point during the ACP synthesis flow.

```
module add(input clk, input [4:0]a,[4:0]b, output reg [4:0]dout);
  always@(posedge clk)
    begin
      dout <= a + b;
    end
  endmodule
```

```
module mult(input clk, input [4:0]a, [4:0]b,
            output reg [9:0]dout) /* synthesis syn_no_compile_point="1" */;

  always@(posedge clk)
  begin
    dout <= a * b;
  end
endmodule
```

VHDL Example

The following VHDL code segment contains the architecture, mult, which should not be treated as a compile point during the ACP synthesis flow.

```
--Multiplier Module
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity mult is
  generic (size: integer :=5);
  port (f_out : out std_logic_vector(9 downto 0);
        a : in std_logic_vector (size- 1 downto 0);
        b : in std_logic_vector (size- 1 downto 0);
        clk : in std_logic
      );
end;

architecture rtl of mult is
attribute syn_no_compile_point: boolean;
attribute syn_no_compile_point of rtl: architecture is true;

begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      f_out <= a * b;
    end if;
  end process;
end;

--Add Module
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity add is
generic (size: integer :=5);
port (f_out : out std_logic_vector(4 downto 0);
      a : in std_logic_vector (size- 1 downto 0);
      b : in std_logic_vector (size- 1 downto 0);
      clk : in std_logic
      );
end;

architecture rtl of add is
begin
process (clk)
begin
  if (clk'event and clk = '1') then
    f_out <= a + b;
  end if;
end process;
end;
```

Effect of Using `syn_no_compile_point`

This attribute can be used when the Auto Compile Point option is turned on or if it is set in the project (prj) file as:

```
set_option -automatic_compile_point 1
```

The Automatic Compile Point (ACP) flow is applied globally and creates compile points automatically for large modules of a design. If you do not want this to occur for individual modules in the design, then you must set the `syn_no_compile_point` attribute to 1. This turns off the effects of automatically creating a compile point for the specified modules, which prevents extensive optimizations within the design units.

The effects of ACP synthesis for the Verilog/VHDL code segments above can be shown in the Technology view, where a module displayed with the color green (for example, v:add) is a compile point and a module displayed with the color yellow (for example, v:mult) is not considered a compile point and was specified with `syn_no_compile_point=1`.

syn_noarrayports

Attribute

Specifies signals as scalar in the output file.

Vendor	Devices
Achronix	Speedster7t and newer family
Intel FPGA	Stratix and newer families
Lattice	ECP4, ECP3, iCE40 and newer families
Microchip	SmartFusion, ProASIC3E and newer devices
Xilinx	Virtex and newer 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
Achronix	all
Intel FPGA	all
Lattice	all, including iCE40
Microchip	all
Xilinx	all

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 ckbuffg (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

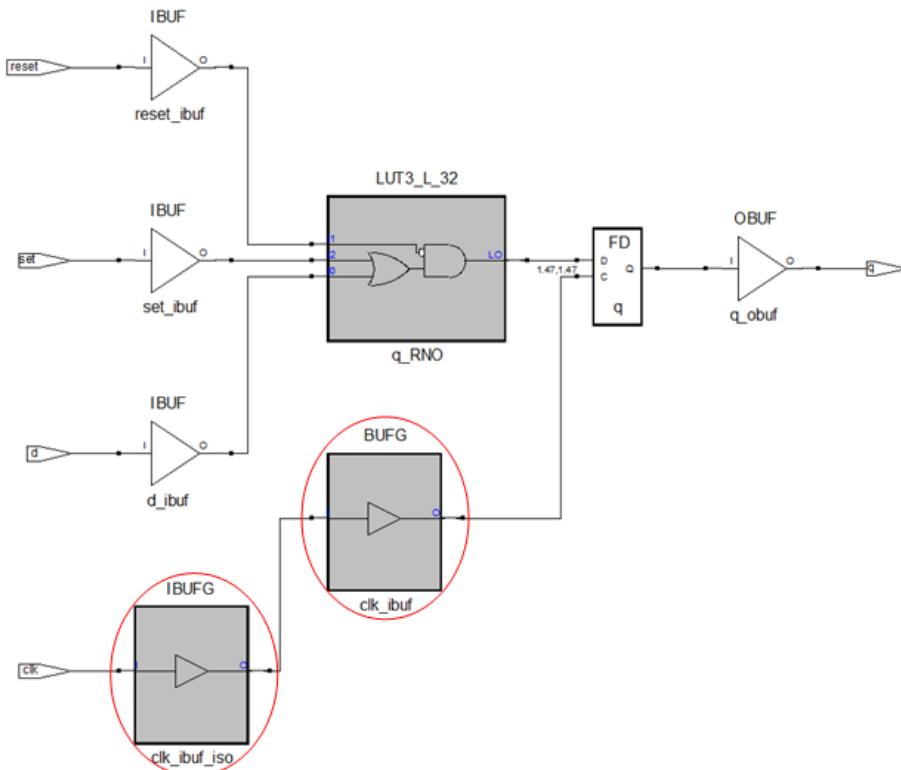
The following graphic shows a design without the `syn_noclockbuf` attribute.

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

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

Global buffers are inferred

||

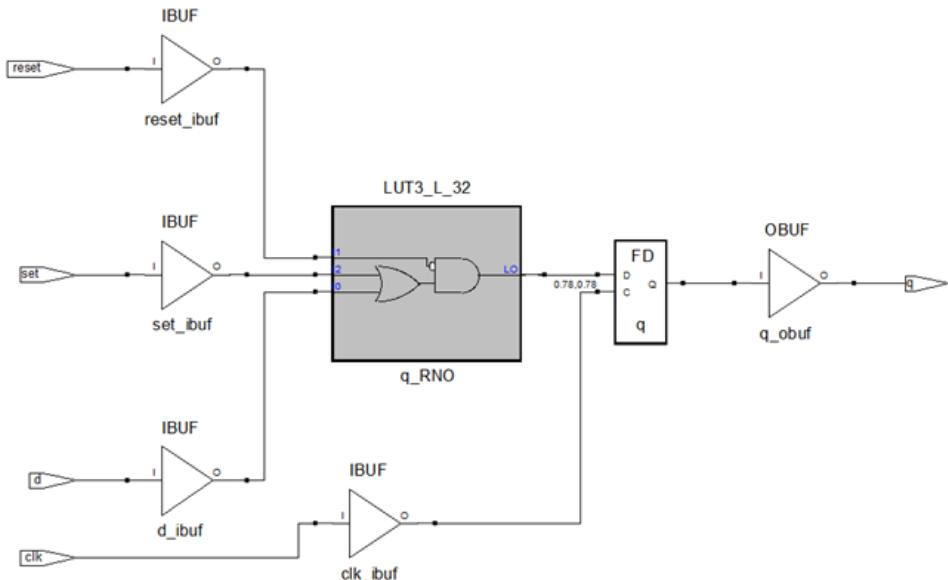


The following graphic shows a design with the `syn_noclockbuf` attribute.

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

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

No global buffers inferred



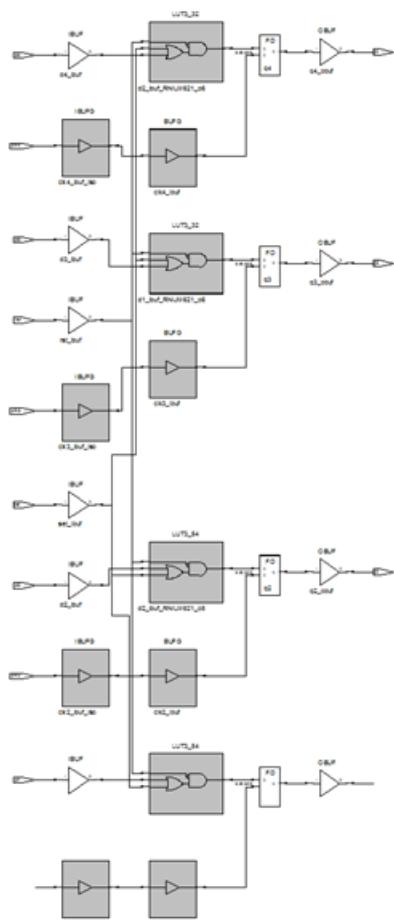
Global Support

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.

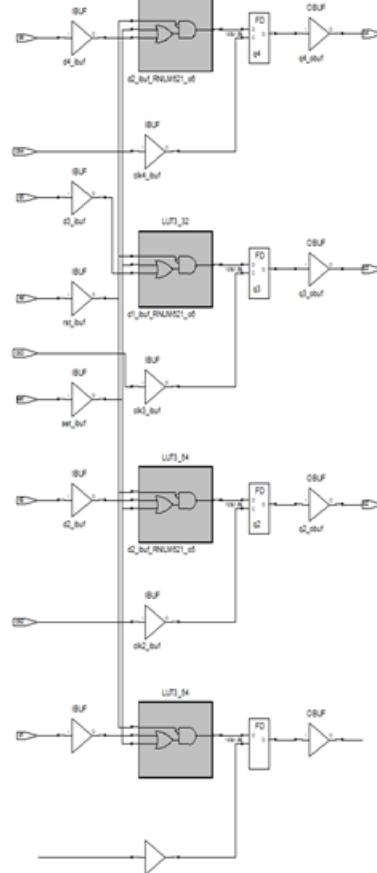
HDL module
 ckbufg(d1,d2,d3,d4,clk1,clk2,clk3,clk4,rst,set,q1,q2,q3,q4)/*synthesis
 syn_noclockbuf=1*/;

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 `define_attribute {p:clockName} syn_noclockpad {0 | 1}` [FDC Example](#)

Verilog `object /* synthesis syn_noclockpad = "0 | 1" */;` [Example - Verilog syn_noclockpad](#)

VHDL `attribute syn_noclockpad : boolean;`
`attribute syn_noclockpad of object : objectType is false | true;` [Example - VHDL syn_noclockpad](#)

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;

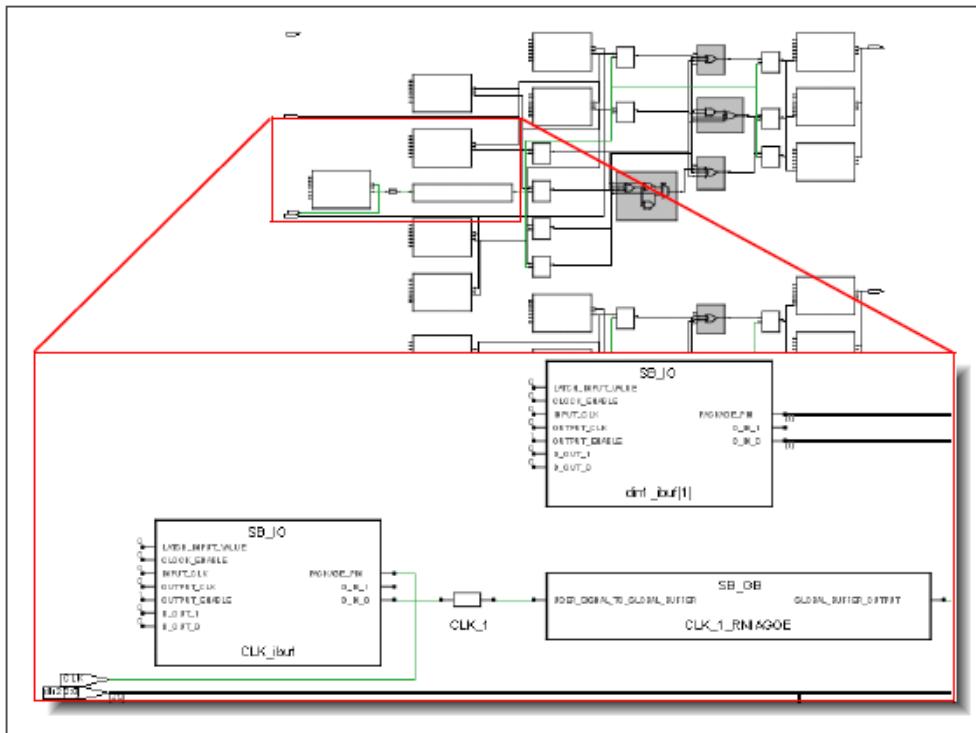
process(CLK2)
begin
if (CLK2'event and CLK2 ='1') then


```

```
reg_3 <= din3;  
reg_4 <= din4;  
Q2 <= reg_3 + reg_4;  
end if;  
end process;  
  
end rtl;
```

Effect of Using syn_noclockpad

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 361](#), and [Preserving Objects from Being Optimized Away, on page 571](#) in the *User Guide*.

syn_noprune Syntax

CDC File	<code>define_directive {object} {syn_noprune} {0 1}</code>	CDC Example
Verilog	<code>object /* synthesis syn_noprune = 1 */;</code>	Verilog Examples
VHDL	<code>attribute syn_noprune : boolean</code> <code>attribute syn_noprune of object : objectType is true;</code>	VHDL Examples

CDC Example

```
define_directive {v:bb} {syn_noprune} (1)
```

Verilog Examples

This section contains 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 451](#).

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 sub1 (data, rst, dout);
parameter width = 1;
input [width :0] data;
input rst;
output [width : 0] dout;
assign dout = rst?1'b0:data;
endmodule

//Intermediate Top level with 3 instances of sub1
module top (data1,data2,data3, rst, dout1);
parameter width1 = 2;
parameter width2 = 3;
parameter width3 = 4;
input [width1 :0] data1;
input [width2 :0] data2;
input [width3 :0] data3;
input rst;
output [width1 : 0] dout1;
sub1 #(width1) inst1 (data1,rst,dout1);
sub1 #(width2) inst2 (data2,rst,) /* synthesis syn_noprune=1 */;
sub1 #(width3) inst3 (data3,rst,);
endmodule

//Top level
module top1 (data1,data2,data3, rst, dout1);
parameter width1 = 2;
parameter width2 = 3;
parameter width3 = 4;
input [width1 :0] data1;
```

```
input [width2 :0] data2;
input [width3 :0] data3;
input rst;
output [width1 : 0] dout1;
top #(width1, width2, width3) top (data1,data2,data3, rst, dout1);
endmodule
```

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 452](#).

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 454](#).

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

--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 no(prune) : boo
```

```
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 455](#).

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.

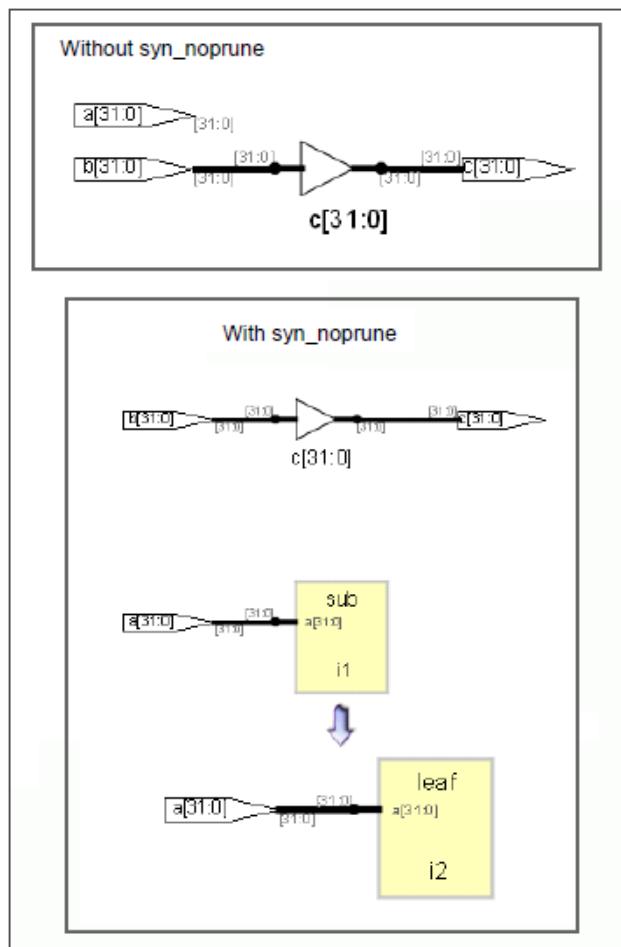
```
library ieee;
use ieee.std_logic_1164.all;
entity sub is
port (a, b, c, d : in std_logic;
x,y : out std_logic);
end sub;

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 456](#).

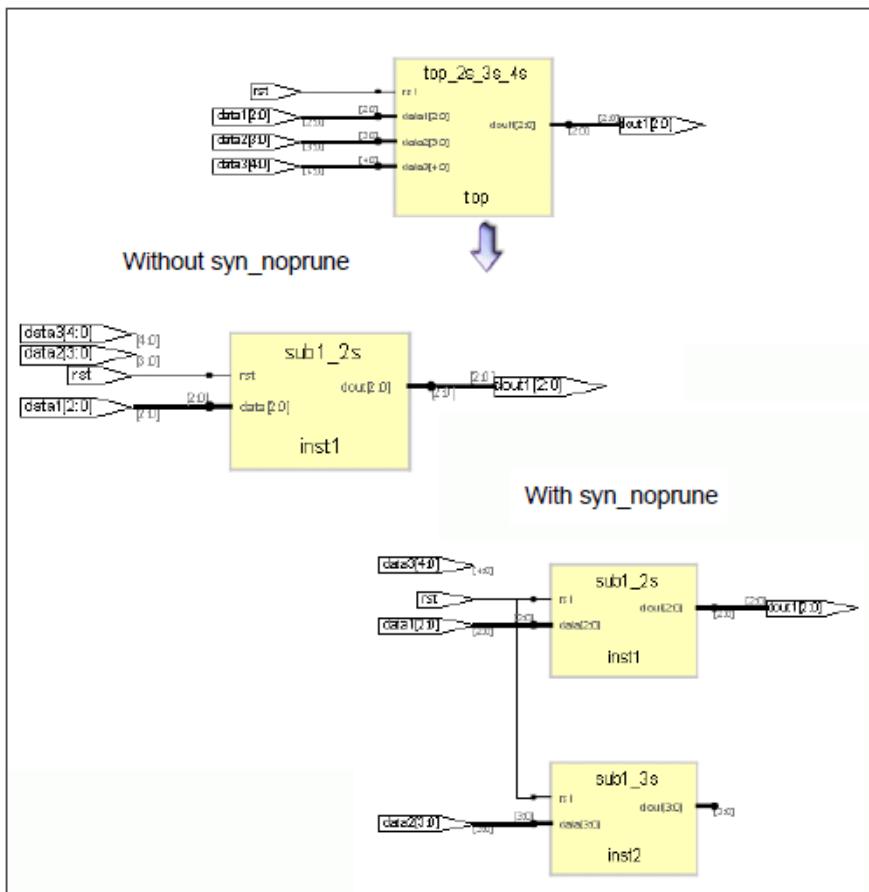
Effect of Using syn_noprune: Example 1

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.



Effect of Using syn_noprune: Example 2

In this example, the software preserves the lower-most leaf hierarchy inst2 and the hierarchy above it. When syn_noprune is not applied, inst2 is not preserved.



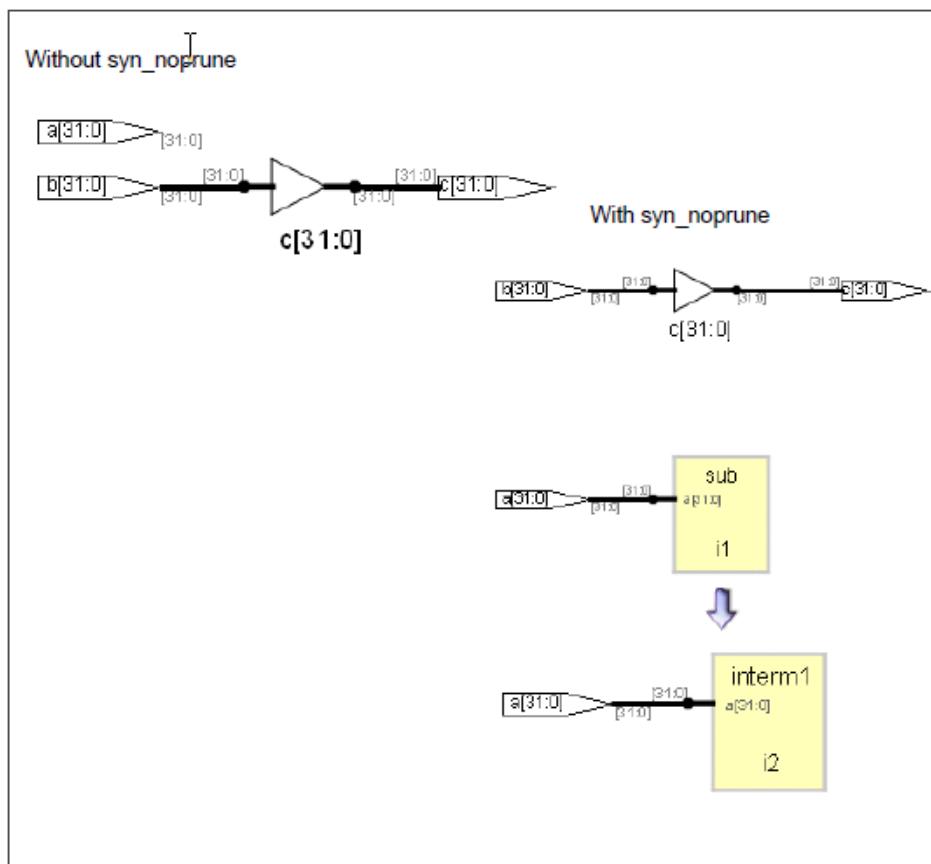
In this example, the software propagates the `syn_noprune` directive downwards in the hierarchy chain.

```
//Top module
module top (input int a, b, output int c);
assign c=b;
sub i1 (a);
endmodule

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

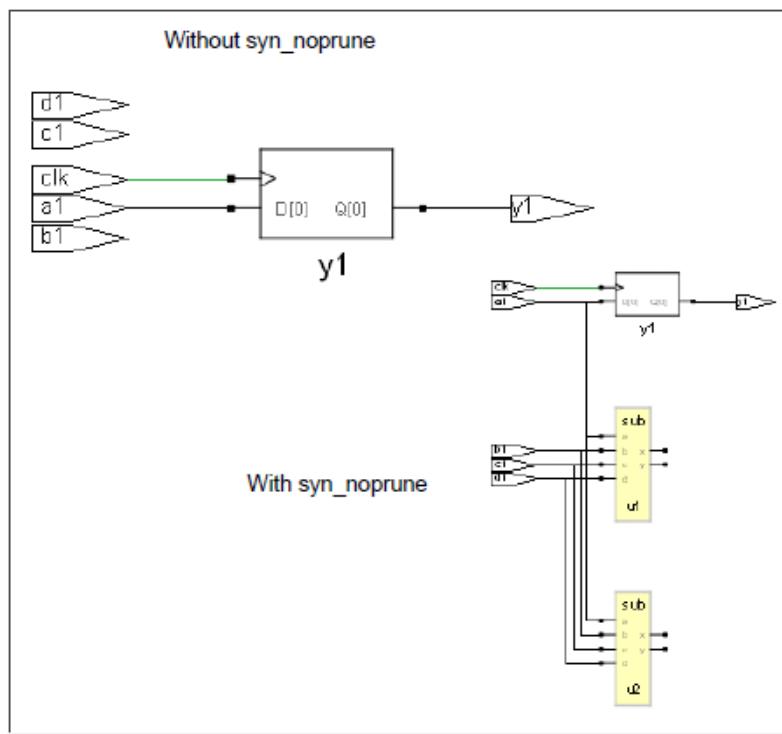
```
//Hier2
module interm1 (input int a) /* synthesis syn_noprune=1 */;
  interm2 i3 (a);
endmodule

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



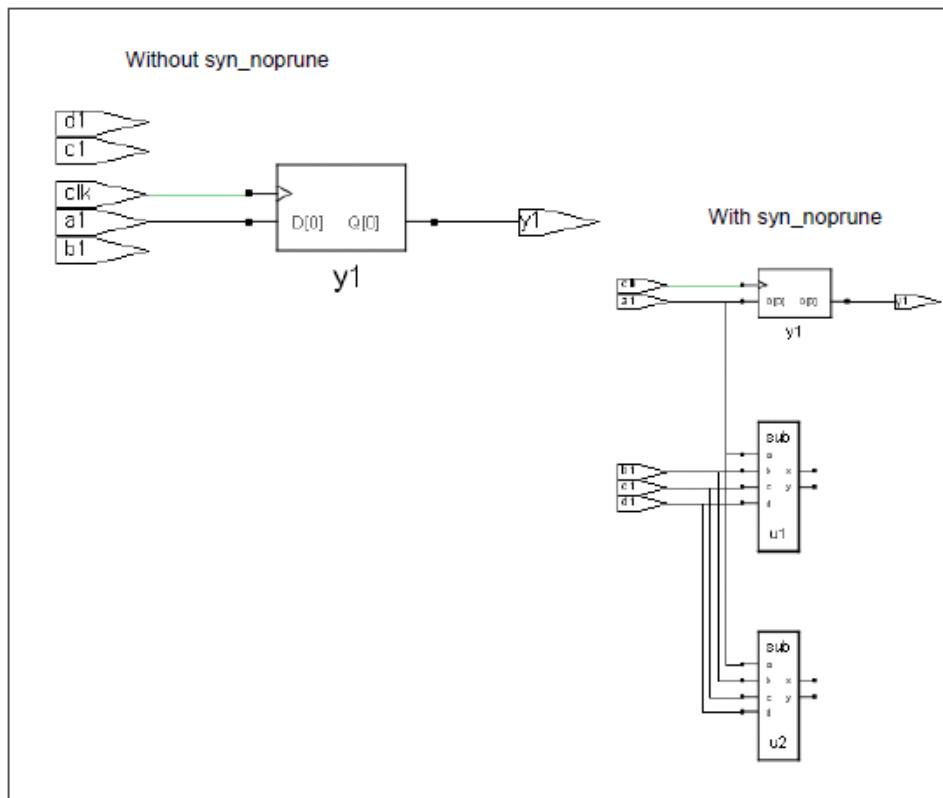
Effect of Using `syn_noprune`: Example 3

The following RTL views show that the design hierarchy is preserved when the `syn_noprune` directive is applied for the component `sub`.



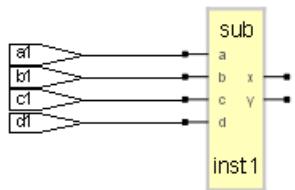
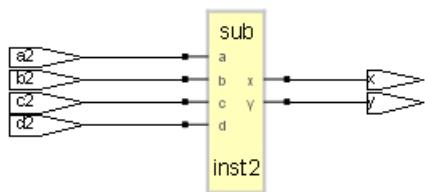
Effect of Using `syn_noprune`: Example 4

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
Intel FPGA	Cyclone and Stratix and newer families
Lattice	iCE40 and newer families
Microchip	Axcelerator, IGLOO, and ProASIC and newer families
Xilinx	Spartan-3 and Virtex-II and newer families

syn_pad_type Values

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

Description

Specifies an I/O buffer standard. Refer to [Industry I/O Standards, on page 321](#) and to the vendor-specific documentation for a list of I/O buffer standards available for the selected device family. Use this attribute instead of the Xilinx I/O buffer standard attribute `xc_padtype`.

syn_pad_type Syntax

Default	Global Attribute	Object
Not Applicable	No	Port

FDC	<code>define_io_standard -default_portType {port} -delay_type portType syn_pad_type {io_standard}</code>	FDC Example
	For example: <code>define_io_standard {p} -delay_type output syn_pad_type {LVCMOS_18}</code>	
Verilog	<code>object /* synthesis syn_pad_type = io_standard */</code>	Verilog Example

VHDL	<code>attribute syn_pad_type of object : objectType is io_standard;</code>	VHDL Example
------	--	------------------------------

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	<code>define_io_standard -default_input -delay_type input syn_pad_type {AGP1X}</code>
All output ports to the GTL pad type	<code>define_io_standard -default_output -delay_type output syn_pad_type {GTL}</code>
All bidirectional ports to the CTT pad type	<code>define_io_standard -default_bidir -delay_type bidir syn_pad_type {CTT}</code>

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
97
98
99
100
101
102
103
104
105
106
```

)
(instance m_2_4 (viewRef PRIM (cellRef LUT2_L (libraryRef VIRTEX)))
(property INIT (string "4'h6"))
)
(instance P_2_2 (viewRef PRIM (cellRef LUT2_L (libraryRef VIRTEX)))
(property INIT (string "4'h6"))
)
(instance Pobuf (viewRef PRIM (cellRef OBUF (libraryRef VIRTEX)))
(property IOSTANDARD (string "LVCMOS18"))
)
(instance PC_ibuf (viewRef PRIM (cellRef IBUF (libraryRef VIRTEX))

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

Synplify Pro, Synplify Premier

Permits registers to be moved to improve timing.

Vendor	Technologies
Achronix	Speedster families
Intel FPGA	Stratix, Cyclone, and newer families
Lattice	iCE40, ECP4 and newer families
Microchip	ProASIC3, Fusion, SmartFusion and newer families
Xilinx	Virtex, Spartan, and newer families

syn_pipeline Values

Value	Default	Global	Object	Description
0 False		Yes	Registers Xilinx Virtex families: Registers and ROMs	Disables pipelining.
1 True	Default	Yes	Registers Xilinx Virtex families: Registers and ROMs	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 {register} syn_pipeline {0|1}`

[FDC Example](#)

Verilog `object /* synthesis syn_pipeline = {1|0} */;`

[Verilog Example](#)

VHDL `attribute syn_pipeline of object : objectType is {true|false};`

[VHDL Example](#)

FDC Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	register	i:temp2[7:0]	syn_pipeline	1	boolean	Controls pipelining of registers

Verilog Example

```
module pipeline (a, b, clk,r);
    input [3:0] a,b;
    input clk;
    output [7:0] r;
    reg [3:0] a_reg,b_reg;
    reg [7:0] temp2/* synthesis syn_pipeline = 1 */;
    reg [7:0] temp3;
    wire [7:0] temp1;

    assign temp1 = a_reg * b_reg;

    always @ (posedge clk)
    begin
        a_reg <= a;
        b_reg <= b;
        temp2 <= temp1;
        temp3 <= temp2;
    end

    assign r = temp3;
endmodule
```

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity pipeline is
port (clk : in std_logic;
      a : in std_logic_vector(3 downto 0);
      b : in std_logic_vector(3 downto 0);
      r : out std_logic_vector(7 downto 0));
end pipeline;

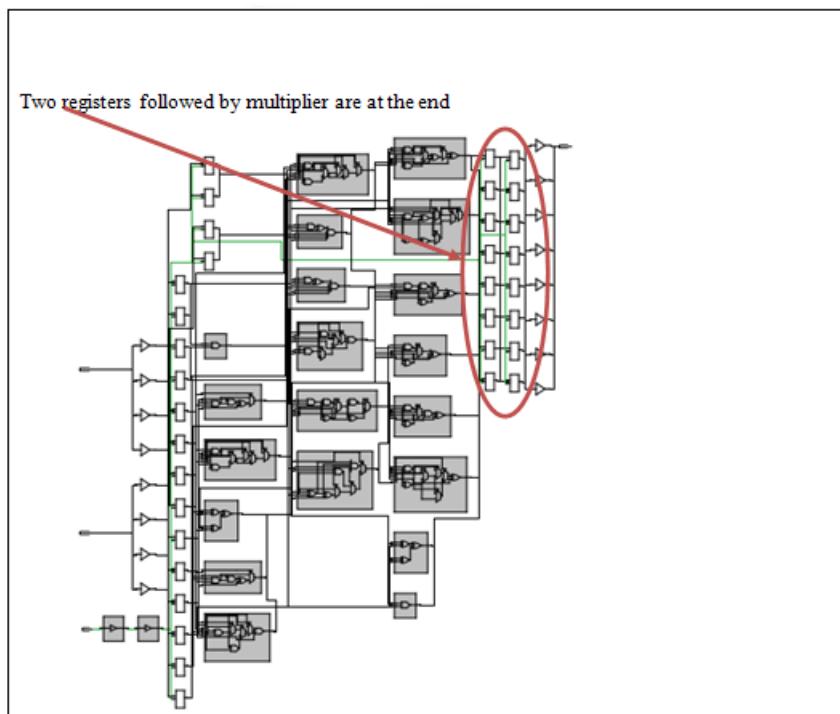
architecture rtl of pipeline is
signal a_reg : std_logic_vector(3 downto 0);
signal b_reg : std_logic_vector(3 downto 0);
signal temp1 : std_logic_vector(7 downto 0);
signal temp2 : std_logic_vector(7 downto 0);
signal temp3 : std_logic_vector(7 downto 0);
attribute syn_pipeline : string;
attribute syn_pipeline of temp2 : signal is "true";
begin
  process(clk)
  begin
    if (clk'event and clk = '1') then
      temp1 <= a_reg * b_reg;
      a_reg <= a;
      b_reg <= b;
      temp2 <= temp1;
      temp3 <= temp2;
      r <= temp3;
    end if;
  end process;
end rtl;
```

Effect of Using syn_pipeline

The following example shows a design where syn_pipeline is set to 0:

Verilog reg [7:0] temp2/* synthesis syn_pipeline = 0 */;

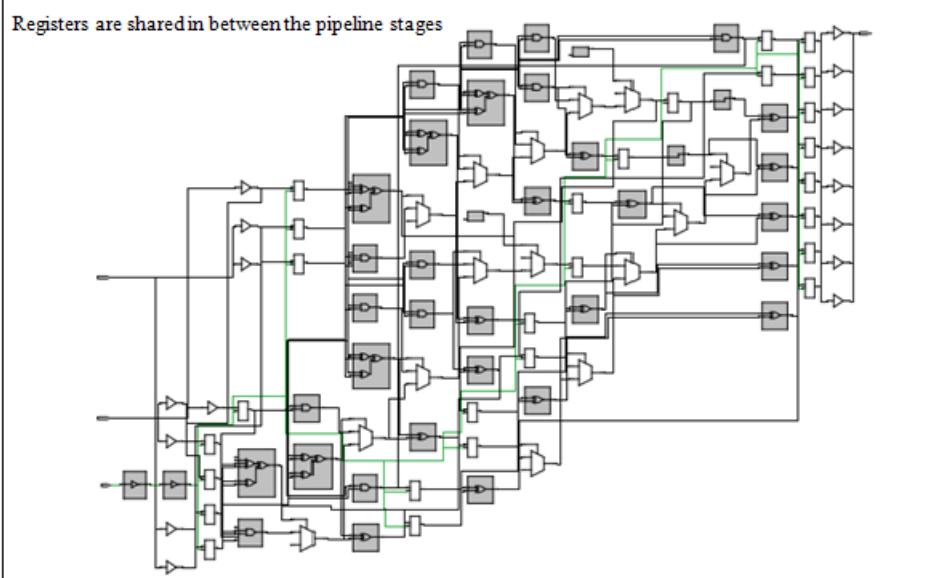
VHDL attribute syn_pipeline of temp2 : signal is "false";



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 361](#), and [Preserving Objects from Being Optimized Away, on page 571](#) in the *User Guide*.

syn_preserve Syntax

CDC File	<code>define_directive {object} {syn_preserve} {0 1}</code>	CDC Example
Verilog	<code>object /* synthesis syn_preserve = 0 1 */</code>	Verilog Example
VHDL	<code>attribute syn_preserve of object : objectType is true false;</code>	VHDL Examples

CDC Example

```
define_directive {v:mod_preserve} {syn_preserve}{1}
```

Verilog Example

In the following example, `syn_preserve` is applied to all registers in the module to prevent them from being optimized away. For the results, see [Effect of using syn_preserve, on page 474](#).

```
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 simpledff is
    port (q : out std_logic_vector(7 downto 0);
          d : in std_logic_vector(7 downto 0);
          clk : in std_logic);

-- Turn on flip-flop preservation for the q output
attribute syn_preserve : boolean;
attribute syn_preserve of q : signal is true;
end simpledff;

architecture behavior of simpledff is
begin
    process(clk)
    begin
        if rising_edge(clk) then
-- Notice the continual assignment of "11111111" to q.
            q <= (others => '1');
        end if;
    end process;
end behavior;

```

Example 2

In this example, `syn_preserve` is used on the signal `curstate` that is later used in a state machine to hold the value of the state register.

```
architecture behavior of mux is
begin
  signal curstate : state_type;
  attribute syn_preserve of curstate : signal is true;

  -- Other code
```

Example 3

The results for the following example are shown in [Effect of using `syn_preserve`, on page 474](#).

```
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;
```

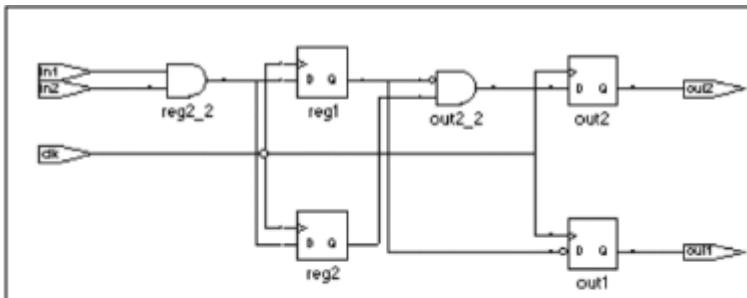
Effect of using `syn_preserve`

The following figure shows `reg1` and `out2` are preserved during optimization with `syn_preserve`.

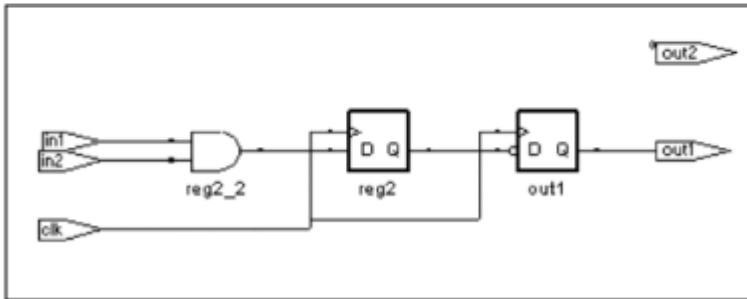
When syn_preserve is not set, reg1 and reg2 are shared because they are driven by the same source. out2 gets the result of the AND of reg2 and NOT reg1. This is equivalent to the AND of reg1 and NOT reg1, which is a 0. As this is a constant, the tool removes out2 and the output out2 is always 0.

Verilog mod_preserve /* synthesis syn_preserve = 1 */

VHDL attribute syn_preserve of behave : architecture is true;



With syn_preserve



Without syn_preserve

syn_preserve_rom

Directive

Apply on a ROM instance to control optimization when ROM contents are read from the memory initialization file.

Vendor	Technology
Achronix	Speedster7t

Value	Description
1 (Default)	Preserves ROM
0	Enables ROM optimization based on initial values specified in the file.

syn_preserve_rom Values

Value	Description
1 (Default)	Preserves ROM
0	Enables ROM optimization based on initial values specified in the file.

Description

The syn_preserve_rom directive is supported only when applied on the ROM instance in the RTL. If syn_preserve_rom is set to 0, then the tool optimizes the ROM based on initial values specified in the memory initialization file.

syn_preserve_rom Syntax

The following support applies for the syn_preserve_rom directive.

Global Support	Object
No	Register arrays in Verilog with initial values specified in file.

This table specifies the syntax that is used for the attribute:

Verilog Examples

In the following examples, the `syn_preserve_rom` directive is applied on a ROM with initial values specified in the memory initialization file.

Example 1

The example below shows ROM of size 2Kx40. By default, the tool does not interpret the initial values in the `data.dat`, which preserves the size of ROM after synthesis.

```
module rom(addr, clk, data_out);
parameter ADDR_WIDTH = 11;
parameter DATA_WIDTH = 40;
parameter MEM_DEPTH = 2048;

input [ADDR_WIDTH-1:0]addr;
input clk;
output reg [DATA_WIDTH-1 : 0]data_out;

reg [ADDR_WIDTH-1:0]addr_reg;
reg [DATA_WIDTH-1 : 0]mem[MEM_DEPTH-1:0];

initial
begin
    $readmemb("data.dat",mem); // $readmemh is also supported
end

always @ (posedge clk)
begin
    addr_reg <= addr;
    data_out <= mem[addr_reg];
end
endmodule
```

Example 2

The example below has ROM of size 2Kx40, for which optimization is enabled using the directive `syn_preserve_rom=0`. The tool interprets the content from the `data.dat` file to optimize the ROM size.

```
module rom(addr, clk, data_out);
parameter ADDR_WIDTH = 11;
parameter DATA_WIDTH = 40;
parameter MEM_DEPTH = 2048;

input [ADDR_WIDTH-1:0]addr;
input clk;
output reg [DATA_WIDTH-1 : 0]data_out;

reg [ADDR_WIDTH-1:0]addr_reg;
reg [DATA_WIDTH-1 : 0]mem[MEM_DEPTH-1 : 0]
    /*synthesis syn_preserve_rom=0*/;

initial
begin
    $readmemb("data.dat",mem); // $readmemh is also supported
end

always @ (posedge clk)
begin
    addr_reg <= addr;
    data_out <= mem[addr_reg];
end
endmodule
```


syn_preserve_sr_priority

Attribute

Globally implements hardware that forces set/reset flip-flops to honor the priority for the set or reset, as coded in the design. This attribute can increase the area of your design.

Vendor	Technologies
Microchip	ACT1, 40MX

syn_preserve_sr_priority Values

Value	Description	Default	Global
0 false	Does not force set/reset flip-flops to honor the priority for the set or reset, as coded in the design.	None	Yes
1 true	Forces set/reset flip-flops to honor the priority for the set or reset, as coded in the design.		

Description

Globally implements hardware that forces set/reset flip-flops to honor the priority for the set or reset, as coded in the design. This attribute can increase the area of your design.

Microchip sequential components do not have a well defined behavior when both the set and reset signals are active at the same time, but you can have the synthesis tool automatically add hardware to force the priority that you have coded in the source code of your design.

syn_preserve_sr_priority Syntax

FDC	define_global_attribute syn_preserve_sr_priority {1 0}	FDC Example
Verilog	<i>object</i> /* syn_preserve_sr_priority = "1 0" */;	Verilog Example
VHDL	attribute syn_preserve_sr_priority : boolean; attribute syn_preserve_sr_priority of <i>object</i> : <i>objectType</i> is "true false";	VHDL Example

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description	Comment
1	<input checked="" type="checkbox"/>	global	<Global>	syn_preserve_sr_priority	1	boolean	Controls the prio of set vs reset	
2								

For example:

```
define_global_attribute syn_preserve_sr_priority {1}
```

Most language models for a set/reset level-sensitive latch or D flip-flop define the behavior for this condition. Using this attribute sets the priority of set/reset as it is specified in your HDL source code. The following Verilog code implies that reset has priority over set if both are active:

```
if (reset)
    q=0;
else if (set)
    q=1;
else
    q=d;
end if;
```

Verilog Example

The following example sets the syn_preserve_sr_priority attribute on the latch3 module. The value 1 indicates that the attribute is on.

```
module latch3(q,data,set,reset,clk)
/* synthesis syn_preserve_sr_priority = 1 */;
output q;
input data, clk, set, reset;
reg q;
```

```
always @ (clk or data or set or reset)
begin
    if (reset)
        q = 0;
    else if (set)
        q = 1;
    else if (clk)
        q = data;
    end
endmodule
```

VHDL Example

Where *object* is the entity. Here is an example:

```

library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
port (data, clk, reset, set : in std_logic;
      qrs: out std_logic);
end dff1;

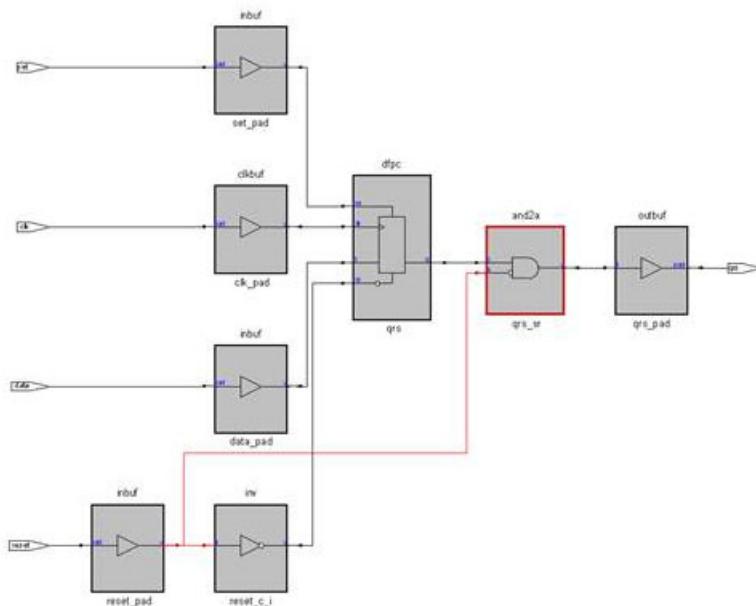
architecture async_set_reset of dff1 is

-- Set the attribute to "true" for async_set_reset architecture
attribute syn_preserve_sr_priority : boolean;
attribute syn_preserve_sr_priority of async_set_reset:
    architecture is true;
begin
setreset: process (clk, reset, set)
begin
    if reset = '1' then
        qrs <= '0';
    elsif set = '1' then
        qrs <= '1';
    elsif rising_edge(clk)then
        qrs <= data;
    end if;
end process setreset;
end async set reset;

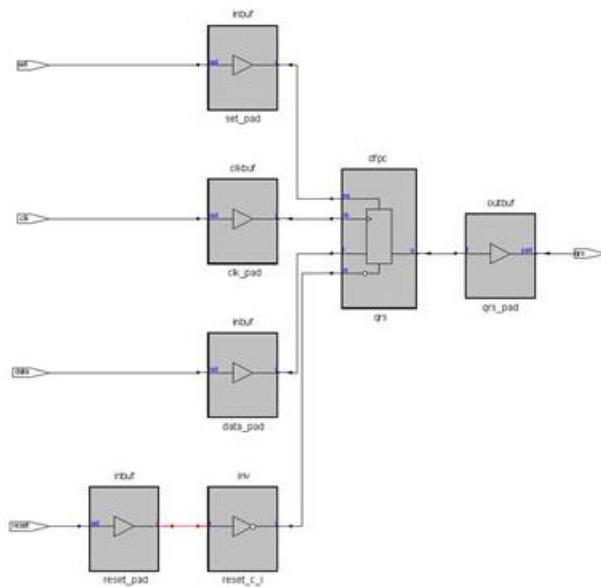
```

Effect of Using syn_preserve_sr_prioritiy

When the attribute is set to true, the tool ensures that the priority specified in the RTL is implemented in the netlist and infers any extra logic to handle the priority.



When the attribute is set to false, the tool does not check for priority in the RTL and does not infer any extra logic.



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 and assign pins to named ports for selected technologies. 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.

For Intel FPGA and Xilinx families, you can specify pin locations for the probe signals. Pin locations cannot be applied to bus slices.

:

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 <code>syn_probe</code> value.
n:aluout[2]	test_pt	test_pt	For string values on a bus slice, the port name is identical to the <code>syn_probe</code> 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 <code>syn_probe</code> 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	define_attribute {n:netName} syn_probe {probePortname 1 0}	FDC Example
Verilog	object /* synthesis syn_probe = "string" 1 0 */;	Verilog Example
VHDL	attribute syn_probe of object : signal is "string" 1 0;	VHDL Example

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. The example uses the pin assignments for a Xilinx Virtex device.

```
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. The example uses the pin assignments for a Xilinx Virtex-4 device.

```
library ieee;
use ieee.std_logic_1164.all;
entity alu is
port (a : in std_logic_vector(7 downto 0);
      b : in std_logic_vector(7 downto 0);
      opcode : in std_logic_vector(2 downto 0);
      clk : in std_logic;
      out1 : out std_logic_vector(7 downto 0));
end alu;
architecture rtl of alu is
signal alu_tmp : std_logic_vector (7 downto 0);

attribute syn_probe : string;
attribute syn_probe of alu_tmp : signal is "test_pt";
attribute syn_loc : string;
attribute syn_loc of alu_tmp : signal is
  "A5,A6,A7,A8,A10,A11,A13,A14";

begin
process (clk)
begin
  if (clk'event and clk = '1') then
    out1 <= alu_tmp;
  end if;
end process;
process (opcode,a,b)
begin
  case opcode is
  when "000" => alu_tmp <= a and b;
  when "001" => alu_tmp <= a or b;
  when "010" => alu_tmp <= a xor b;
  when "011" => alu_tmp <= a nand b;
  when others => alu_tmp <= a nor b;
  end case;
end process;

end rtl;
```

Effect of Using syn_probe

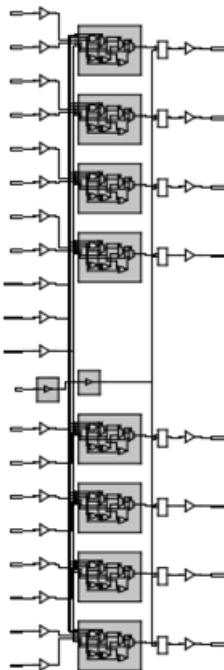
Before applying syn_probe:

Verilog

```
reg [7:0] alu_tmp /* synthesis syn_probe="0" */
```

VHDL

```
attribute syn_probe of alu_tmp : signal is "0";
```



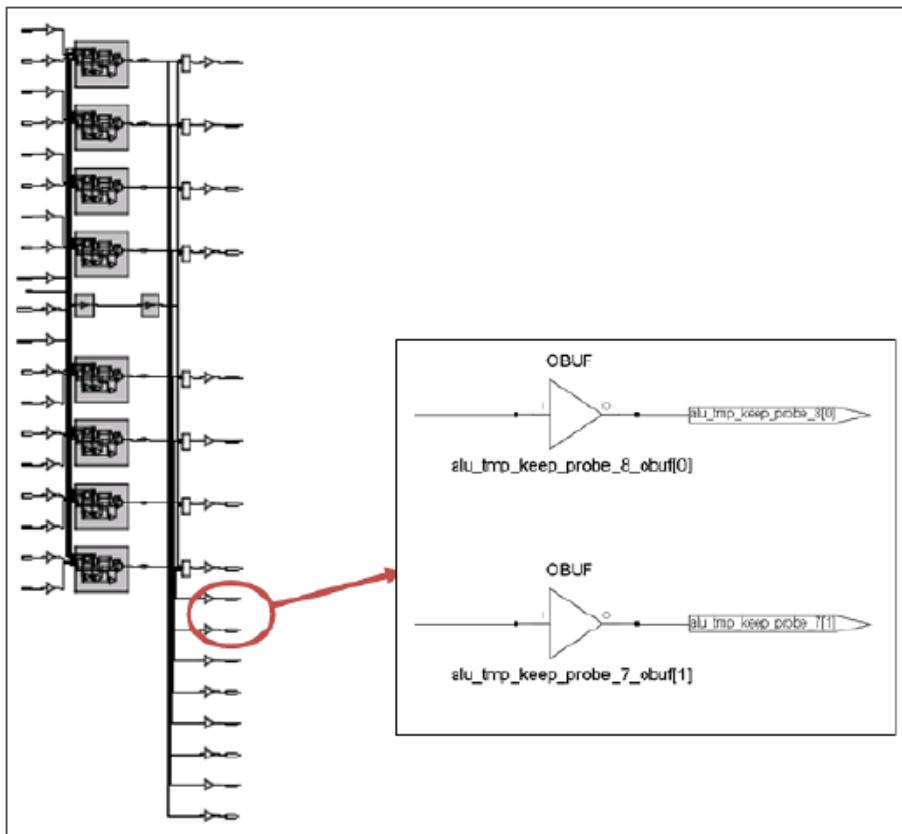
After applying syn_probe with 1:

Verilog

```
reg [7:0] alu_tmp /* synthesis syn_probe="1" */
```

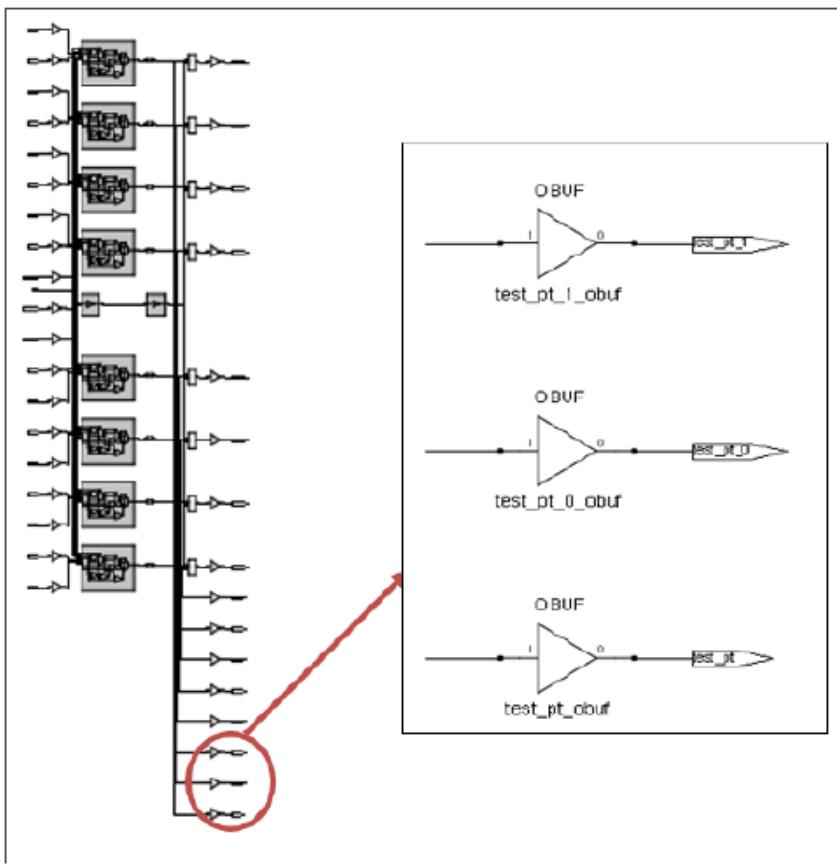
VHDL

```
attribute syn_probe of alu_tmp : signal is "1";
```



After applying `syn_probe` with `test_pt`:

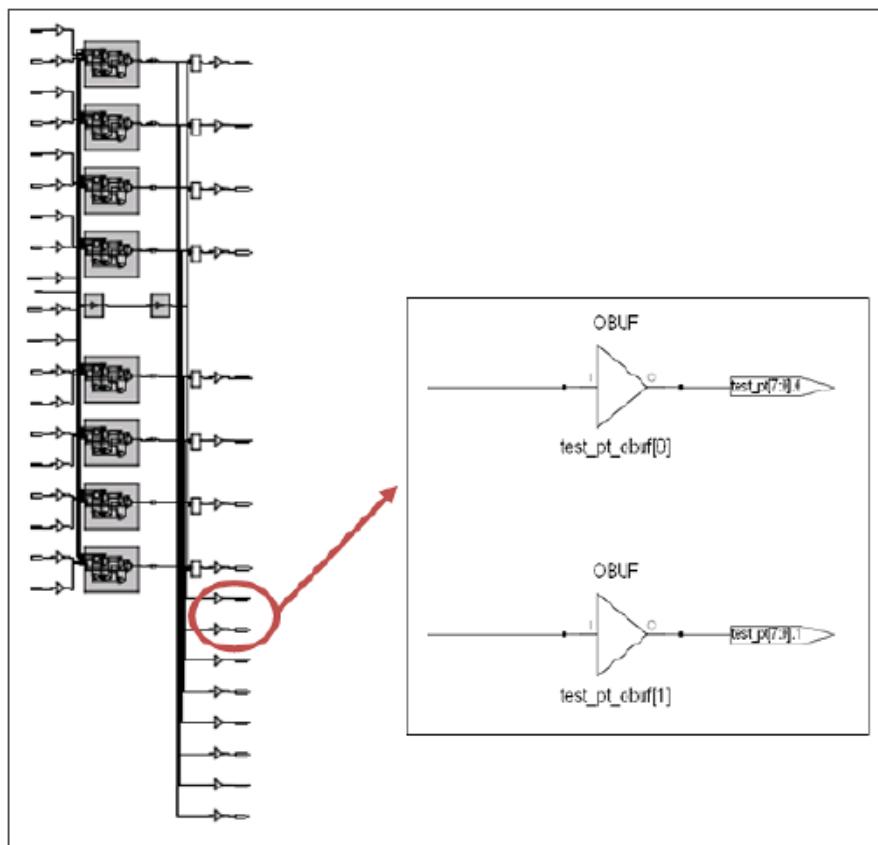
Verilog	<code>reg [7:0] alu_tmp /* synthesis syn_probe="test_pt" */</code>
VHDL	<code>attribute syn_probe of alu_tmp : signal is "test_pt";</code>



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_radhardlevel

Attribute

Implements designs with high reliability, using radiation-resistant techniques.

Vendor	Technologies	Tool
Intel FPGA	Arria 10, Arria V, Cyclone V, Cyclone IV, Cyclone III, Stratix V, Stratix IV, Stratix III, Stratix10	Synplify Premier
Lattice	ECP3 and Mach XO2	Synplify Premier
Microchip	Anti-fuse (RD, RH, and RT radhard devices) Fusion, IGLOO, IGLOO+, IGLOOE, IGLOO2, ProASIC3, ProASIC3E, ProASIC3L, RTG4, Smartfusion, SmartFusion2, PolarFire Axcelerator RT (attribute ignored)	Synplify Pro Synplify Premier
Xilinx	Artix-7, Kintex-7, Spartan-6, Virtex and later, and UltraScale/UltraScale+ families	Synplify Premier

Description

This attribute enables triple modular redundancy (TMR) for hierarchical modules in the design and implements distributed TMR, block TMR, or local TMR. You can also use this attribute to enable Duplicate with Compare (DWC), which duplicates modules and compares the output results to determine if an SEU error occurs.

syn_radhardlevel Values

You can apply the syn_radhardlevel attribute at the module (view) level or you can set it globally on a top-level module for distributed TMR and block TMR. You can only apply this attribute at the module level for the Duplicate with Compare feature.

:

The `syn_radhardlevel` attribute can use the following options:

`syn_radhardlevel = none | block_tmr | distributed_tmr | duplicate_with_compare | tmr`

The applicable `syn_radhardlevel` options for fault-tolerant design vary, according to the vendor and the technology family. See the following for specific details:

none	<i>Intel FPGA, Lattice, Microchip, Xilinx</i> Default* Uses standard design techniques, and does not insert any triple register logic. *For Microchip Axcelerator RT devices, the tool ignores the <code>syn_radhardlevel</code> attribute.
block_tmr	<i>Intel FPGA, Lattice, Microchip, Xilinx</i> Triplicates modules and inserts majority voting logic only on the module outputs.
cc	<i>Microchip Anti-fuse</i> Implements combinational cells with feedback as storage, rather than flip-flop or latch primitives.
distributed_tmr	<i>Intel FPGA, Lattice, Microchip, Xilinx</i> Triplicates modules and inserts majority voting logic on the sequential loops and the module outputs. This option can potentially impact area and timing QoR because of the additional logic inserted, so be sure to check your area and timing goals when you use this option. See Specifying Local TMR for RAMs, on page 943 in the <i>User Guide</i> for a procedure.

<code>duplicate_with_compare</code>	Intel FPGA, <i>Lattice</i> , <i>Microchip</i> , <i>Xilinx</i> Duplicates modules and compares the outputs of this circuit to determine if an SEU failure occurs. Signal nets are also duplicated to be the same as the original connectivity. However, this option limits the redundancy performed, so it has less impact on area and QoR than distributed TMR.
<code>tmr</code>	<i>Microchip Anti-fuse (RD, RH, and RT radhard devices), Fusion, IGLOO, IGLOO+, IGLOOE, IGLOO2, ProASIC3, ProASIC3E, ProASIC3L, RTG4, SmartFusion, SmartFusion2, PolarFire</i> Intel FPGA, <i>Lattice</i> , <i>Xilinx</i> Uses triple module redundancy or triple voting to implement registers. Each register is implemented by three flip-flops or latches that “vote” to determine the state of the register. This option can potentially affect area and timing QoR because of the additional logic inserted, so be sure to check your area and timing goals when you use this option.
<code>tmr_cc</code>	<i>Microchip Anti-fuse</i> Uses triple module redundancy, where each voting register is composed of combinational cells with feedback rather than flip-flop or latch primitives.

syn_radhardlevel Syntax

Name	Global Attribute	Object
<code>syn_radhardlevel - for Distributed TMR and Block TMR</code>	Yes	View, module, architecture
<code>syn_radhardlevel - for Duplicate with Compare</code>	No	View, module, architecture
<code>syn_radhardlevel - for Local TMR or Register-based TMR</code>	Yes	FDC: view, register instance Verilog: module, register VHDL: architecture, signal

syn_radhardlevel Syntax for Local TMR (Microchip)

Some high reliability techniques are not available or appropriate for all Microchip families. Use a design technique that is valid for the project. Contact Microchip technical support for details.

:

You can apply `syn_radhardlevel` globally to the top-level module/architecture or on an individual register output signal (or inferred register in VHDL), and the tool uses the attribute value in conjunction with the Microchip macro files supplied with the software. For more details about using this attribute, see [Specifying syn_radhardlevel in the Source Code](#), on page 928 and [Working with Microchip Radhard Designs](#), on page 927 in the *User Guide*.

The following table summarizes the syntax in different files:

FDC	<code>define_global_attribute {v:topLevelModule object}</code> <code>syn_radhardlevel {none cc tmr tmr_cc}</code> <code>define_attribute {object} syn_radhardlevel</code> <code>{none cc tmr tmr_cc}</code>	FDC File Example
Verilog	<code>object /* synthesis syn_radhardlevel = none cc tmr tmr_cc */</code>	Verilog Example
VHDL	<code>attribute syn_rardhardlevel : boolean;</code> <code>attribute syn_radhardlevel of object : object type is</code> <code>none cc tmr tmr_cc;</code>	VHDL Example

FDC File Example

```
define_attribute {i:dataout[3:0]} syn_radhardlevel {tmr}
```

Verilog Example

```
//Top level
module top (clk, dataout, a, b);
    input clk;
    input a;
    input b;
    output [3:0] dataout;
    M1 inst_M1 (a1, M3_out1, clk, rst, M1_out);
    // Other code

//Sub modules subjected to DTMR
module M1 (a1, a2, clk, rst, q)
    /* synthesis syn_radhardlevel="tmr" */;
    input clk;
    input signed [15:0] a1,a2;
    input clk, rst;
    output signed [31:0] q;
    // Other code
```

VHDL Example

See [VHDL Attribute and Directive Syntax, on page 401](#) for alternate methods for specifying VHDL attributes and directives.

```
library synplify;
architecture top of top is
attribute syn_radhardlevel : string;
attribute syn_radhardlevel of top: architecture is "tmr";
-- Other code
```

syn_radhardlevel Syntax for Local TMR (Registers)

The following table summarizes the syntax in different files for local TMR or register-based TMR:

FDC	define_global_attribute {syn_radhardlevel} {tmr} define_attribute {object} syn_radhardlevel {none tmr}	
Verilog	<i>object</i> /* synthesis syn_radhardlevel = none tmr */	Verilog Example: Local TMR
VHDL	attribute syn_radhardlevel : boolean; attribute syn_radhardlevel of <i>object</i> : object type is none tmr;	

Verilog Example: Local TMR

```
//Example: FFs with Enable

//FFs with Enable
`timescale 1ns/10ps
`ifdef synthesis
    module test(clock,en,datain,dataout);
`else
    module test_rtl(clock,en, datain,dataout);
`endif
parameter width=1;
input clock,en;
```

```

input  datain;
output reg dataout /*synthesis syn_radhardlevel="tmr"*/;
always @ (posedge clock )
begin
    if(en)
        dataout <= datain;
end

```

syn_radhardlevel Syntax for Block TMR, Distributed TMR, or Duplicate with Compare

The following table summarizes the syntax used for different files to specify distributed or block TMR on the top-level module of a design.

FDC	<code>define_global_attribute syn_radhardlevel {none block_tmrv distributed_tmrv} define_attribute {{v:<i>topLevelModule</i>} <i>object</i>} syn_radhardlevel {none block_tmrv distributed_tmrv duplicate_with_compare}</code>	FDC Example
Verilog	<code><i>topLevelModule</i> <i>object</i> /* synthesis syn_radhardlevel = none block_tmrv distributed_tmrv duplicate_with_compare */</code>	Verilog Example
VHDL	<code>attribute syn_radhardlevel : string; attribute syn_radhardlevel of <i>topLevelModule</i>: architecture is none block_tmrv distributed_tmrv duplicate_with_compare; attribute syn_radhardlevel: boolean; attribute syn_radhardlevel of <i>Object</i>:<i>Object Type</i> is none block_tmrv distributed_tmrv duplicate_with_compare;</code>	VHDL Example

FDC Example

```

define_global_attribute {syn_radhardlevel}
{distributed_tmrv | block_tmrv}

define_attribute {v:work.top} {syn_radhardlevel}
{distributed_tmrv | block_tmrv | duplicate_with_compare}

```

Verilog Example

```
// Top level
module top (clk, dataout, a, b)
    /* synthesis syn_radhardlevel="distributed_tmr | block_tmr" */;
    input clk;
    input a;
    input b;
    output [3:0] dataout;
    M1 inst_M1 (a1, M3_out1, clk, rst, M1_out);
```

VHDL Example

```
library synplify;
-- Top level
architecture inst_top of top is
attribute syn_radhardlevel : string;
attribute syn_radhardlevel of inst_top: architecture is
"distributed_tmr | block_tmr";
-- Other code
```

Effect of Using `syn_radhardlevel`

Examples showing the effects of using the `syn_radhardlevel` attribute are provided for the following:

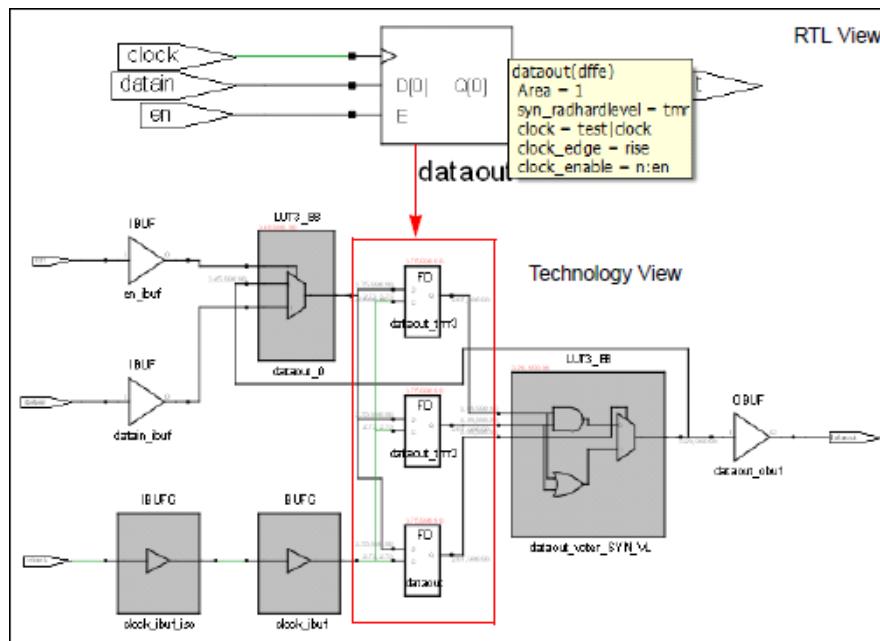
- [Effect of Using `syn_radhardlevel` \(Local TMR\), on page 501](#)
- [Effect of Using `syn_radhardlevel` for Duplicate with Compare, on page 502](#)
- [Effect of Using `syn_radhardlevel` for Distributed TMR and Block TMR, on page 504](#)

Effect of Using `syn_radhardlevel` (Local TMR)

This example shows how TMR is implemented for the register `datout` in the RTL and Technology views of the HDL Analyst tool.

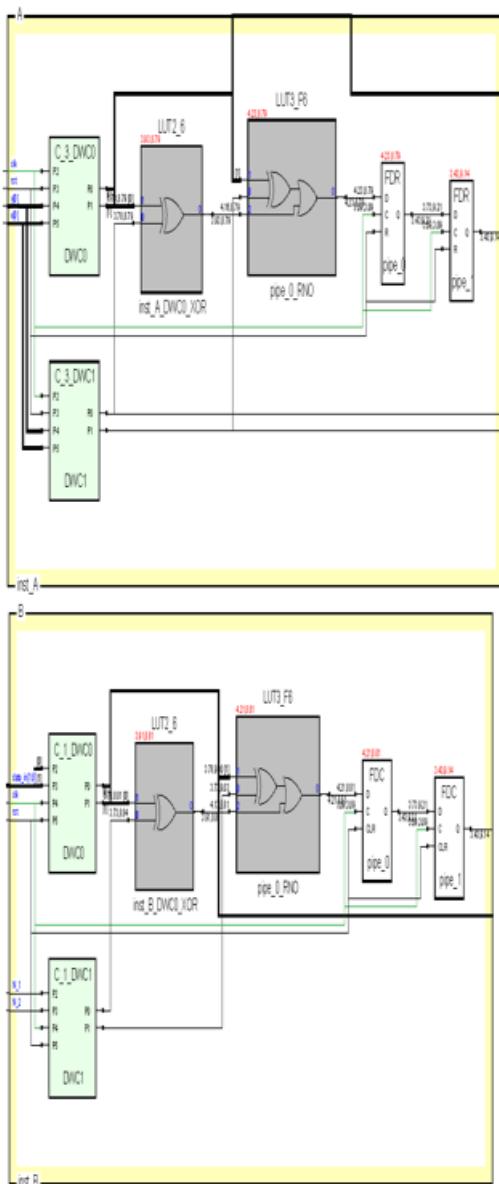
For this example, specify the FDC syntax below:

```
define_attribute {i:dataout} {syn_radhardlevel} {tmr}
```



Effect of Using syn_radhardlevel for Duplicate with Compare

For this example, the `syn_radhardlevel` attribute specifies `duplicate_with_compare` for modules A and B:



Effect of Using syn_radhardlevel for Distributed TMR and Block TMR

For this example, the `syn_radhardlevel` attribute specifies distributed TMR on the top-level module top in the constraint file:

```
define_attribute {v:work.top} {syn_radhardlevel} {distributed_tmr}
```

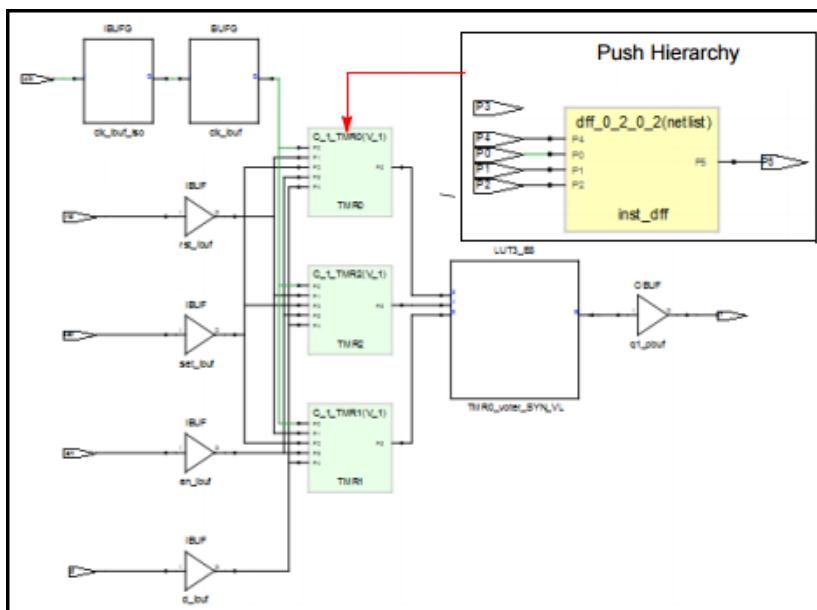
The top-level module is defined as follows:

```
module top (clk, rst, set, en, d, q1);
    input clk;
    input rst;
    input set;
    input en;
    input d;
    output q1;

    dff inst_dff (clk, rst, set, en, d, q1);

endmodule
```

After synthesis, distributed TMR has been applied on the top-level module as shown in the schematic below:



syn_ram_write_mem

Attribute

Generates the memory map information (MMI) file for block RAM.

Vendor	Technologies	Tool
Xilinx	UltraScale/UltraScale+, 7 Series, and Zynq families	Synplify Premier

Description

Generates the memory map information (MMI) file for block RAM. This MMI file is supported by the UpdateMEM utility of the Xilinx Vivado tool and provides a way to modify the contents of the RAM without having to rerun synthesis and place and route. This flow produces consistent quality of results, when only the memory contents has changed.

syn_ram_write_mem Values

Value	Description	Object	Default	Global
1 0	Enables or disables that the MMI file is generated for the RAM instance.	RAM Instance	None	No

syn_ram_write_mem Syntax

FDC	define_attribute {objectName {syn_ram_write_mem {0 1}}}	FDC Example
Verilog	object /* synthesis syn_ram_write_mem = 0 1 */	Verilog Example
VHDL	attribute syn_ram_write_mem of object : objectType is 0 1;	VHDL Example

FDC Example

```
define_attribute {i:bram} {syn_ram_write_mem} {1|0}
```

Verilog Example

```
module test (addr_a,addr_b,clk_a,clk_b,wra,web,din,out);  
  
parameter data_width = 32;  
parameter addr_width = 10;  
  
input [addr_width-1:0] addr_a, addr_b;  
input clk_a, clk_b, wra, web;  
input [data_width-1:0] din;  
output reg [data_width-1:0] out;  
  
reg [data_width-1:0] mem [2**addr_width-1:0]  
/* synthesis syn_ram_write_mem = 1 */;  
  
// ram code  
  
always@(posedge clk_a)  
begin  
    if (wra)  
        mem[addr_a] <= din;  
end  
  
always@(posedge clk_b)  
begin  
    if (web)  
        out <= mem[addr_b];  
    else  
        out <= mem[addr_b];  
end  
endmodule
```

VHDL Example

```
Library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.std_logic_arith.all;  
use IEEE.std_logic_unsigned.all;  
  
entity Dual_Port_ReadFirst is  
generic (data_width: integer :=32;  
address_width: integer :=10);
```

```
port (write_enable: in std_logic;
      write_clk, read_clk: in std_logic;
      data_in: in std_logic_vector (data_width-1 downto 0);
      data_out: out std_logic_vector (data_width-1 downto 0);
      write_address: in std_logic_vector (address_width-1 downto 0);
      read_address: in std_logic_vector (address_width-1 downto 0));
end Dual_Port_ReadFirst;

architecture behavioral of Dual_Port_ReadFirst is
type memory is array (2**(address_width-1) downto 0)
    of std_logic_vector (data_width-1 downto 0);
signal mem: memory;

signal reg_write_address: std_logic_vector
    (address_width-1 downto 0);
signal reg_write_enable: std_logic;

attribute syn_ram_write_mem: boolean;
attribute syn_ram_write_mem of mem: signal is true;

begin
register_enable_and_write_address:
process (write_clk,write_enable,write_address,data_in)
begin
if (rising_edge(write_clk)) then
reg_write_address <= write_address;
reg_write_enable <= write_enable;
end if;
end process;

write:
process (read_clk,write_enable,write_address,data_in)
begin
if (rising_edge(write_clk)) then
if (write_enable='1') then
mem(conv_integer(write_address)) <= data_in;
end if;
end if;
end process;

read:
process (read_clk,write_enable,read_address,write_address)
begin
if (rising_edge(read_clk)) then
data_out <= mem(conv_integer(read_address));
end if;
end process;
end behavioral;
```

Effect of Using syn_ram_write_mem

For this example, look for the ram_0.mmi file in the par_1/mmifiles directory of the Implementation Results directory. Notice that the MMI file contains the physical RAM description and placement information.

```
1 i»;<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
2 <MemInfo Minor="0" Version="1">
3
4   <!--Copyright (C) 1994-2017 Synopsys, Inc. This Synopsys
5 This file is generated by Synplify Premier, version N-2017
6
7   <Processor Endianness="little" InstPath="ram_inst">
8     <AddressSpace Begin="0" End="4096" Name="dummy">
9       <BusBlock>
10         <BitLane MemType="RAMB32" Placement="X13Y18">
11           <!--Instance Path = mem_mem_0_0-->
12           <DataWidth LSB="0" MSB="31"/>
13           <AddressRange Begin="0" End="1023"/>
14           <Parity NumBits="0" ON="false"/>
15         </BitLane>
16       </BusBlock>
17     </AddressSpace>
18   </Processor>
19
20   <Config>
21     <Option Name="Part" Val="xcvu065-ffvc1517-3-e"/>
22   </Config>
23
24 </MemInfo>
```

syn_ramstyle

Attribute

Specifies the implementation for an inferred RAM.

Vendor	Devices
Achronix	Speedster7t devices
Intel FPGA	Arria and newer devices Cyclone and newer devices Stratix and newer devices
Lattice	iCE40 devices LatticeECP/EC and newer devices LatticeSC/SCM devices LatticeXP2/XP devices MachXO and newer devices ORCA and newer devices LIFCL devices
Microchip	Fusion, SmartFusion and newer devices IGLOO and newer devices ProASIC3 and newer devices RTG4 devices
Xilinx	Artix-7, Kintex-7 Spartan and newer devices UltraScale and newer devices Virtex and newer

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 231](#) in the *User Guide*.

<code>area</code>	<p><i>Xilinx</i></p> <p>When used with the <code>block_ram</code> or <code>no_rw_check</code> values, specifies that the inferred RAM be mapped to area-efficient Xilinx memory. The software uses wider block RAM configurations that are more area-efficient, instead of the default deeper RAM configurations. It also uses the parity bit for data. If you want the RAM implementations optimized for timing, use the <code>block_ram</code> or <code>no_rw_check</code> values.</p> <p>For Xilinx Virtex devices, the synthesis tool tries to map memory to a Virtex family Block SelectRAM, depending on the properties of the memory and the available resources. If this is not possible, the memory is mapped using <code>select_ram</code>.</p>
<code>block_ram</code>	<p>Specifies that the inferred RAM be mapped to the appropriate device-specific memory. It uses the 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.</p> <p>Alternatively, you can specify a <code>ramType</code> value. See RAM Type Values and Implementations, on page 513 for details of how memory is implemented for different devices.</p> <p><i>Xilinx:</i></p> <p>As the parity bit is not used for data by default, use <code>block_ram</code> with <code>power</code> for a more area-efficient RAM inferencing scheme. For Virtex designs, it implements block SelectRAM+ with additional glue logic to resolve read/write address conflicts.</p>
<code>ecc</code>	<p><i>Synplify Premier</i> <i>Intel FPGA, Microchip, Xilinx</i></p> <p>Generates an ECC (Error Correction Code) RAM primitive and implements its associated glue logic (address decoders/encoders or comparators) in triplicate, while keeping the RAM interface the same.</p> <p>Use this option to incorporate fault tolerance in high reliability designs. See Specifying ECC for RAMs, on page 944 for additional information about RAM in high reliability designs.</p>

large_ram

Lattice
LIFCL devices only

Specifies that the inferred RAM be mapped to the appropriate device-specific Large RAM memories of size 512Kb. It uses dedicated memory resources in the FPGA. Preference is given to the Large RAM resources available in the device part over the attribute.

logic_ram

Achronix

Specifies that the inferred RAM is mapped to the technology primitive LRAM2K_SDP.

no_rw_check

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. 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.

There are other read-write check controls. See [Read-Write Address Checks , on page 517](#) for details about the differences.

no_rw_check_diff_clk

When enabled, the synthesis tool prevents the insertion 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:

```
create_clock {p:clk_r} -period {10}
create_clock {p:clk_w} -period {20}
set_clock_groups -derive -asynchronous -name
{async_clkgroup} -group { {c:clk_w} }
```

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.

:

<code>no_uram</code>	<i>Xilinx</i> When the <code>no_uram</code> option is specified, the tool prevents the inference of UltraRAM (URAM) block. When <code>no_uram</code> is used with <code>block_ram</code> , the memory is mapped to the block RAM instead of the URAM block. Usage: <code>/* synthesis syn_ramstyle = "block_ram,no_uram" */</code>
<code>power</code>	<i>Synplify Premier, Xilinx</i> When used with the <code>block_ram</code> or <code>no_rw_check</code> values, specifies that the inferred RAM be mapped to power and area-efficient memory. The tool implements RAM using area-efficient wider block RAM configurations, instead of the default deeper RAM configurations.
<code>ramType</code>	Specifies a device-specific RAM implementation. Valid values vary from vendor to vendor as they are based on device architecture: <ul style="list-style-type: none"> • Achronix: <code>logic_ram</code> • Intel FPGA: M4K, M512, M9K, M144K, M20K, MLAB, M-RAM • Lattice: <code>distributed</code> • Microchip: <code>Isram, uram</code> • Xilinx: <code>select_ram, uram</code> See RAM Type Values and Implementations , on page 513 for details of how memory is implemented for different devices.
<code>registers</code>	Specifies that an inferred RAM be mapped to registers (flip-flops and logic), not technology-specific RAM resources.
<code>rw_check</code>	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 <code>no_rw_check</code> option, as the two are mutually exclusive. Do not enable this option for RAMs with asynchronous read/write clocks. If <code>rw_check</code> is enabled on block RAM with an asynchronous read clock (<code>rclk</code>) and write clock (<code>wclk</code>), the tool inserts extra logic and a timing path between <code>wclk</code> and <code>rclk</code> . If the clocks are asynchronous to each other, this path can produce glitches on hardware. There are other read-write check controls. See Read-Write Address Checks , on page 517 for details about the differences.

select_ram	<i>Xilinx</i>	Implements distributed RAM using the resources in the CLBs. For distributed RAMs, the write operation must be synchronous and the read operation asynchronous.
tmr	<i>Synplify Premier Intel FPGA, Xilinx</i>	Implements the inferred RAM primitive in triplicate for triple module redundancy (TMR). You must enable the advanced synthesis mode to ensure this implementation. Use this option to incorporate fault tolerance in high reliability designs. See Specifying Local TMR for RAMs, on page 943 for additional information about RAM in high reliability designs.
uram	<i>Xilinx</i>	When the uram option is specified, the tool forces the RAM to infer the UltraRAM (URAM) block.

RAM Type Values and Implementations

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

Vendor	Values	Implementation	Technology
Achronix	block_ram	Default: Device-specific RAM	Speedster7t family
	registers	Registers	
	no_rw_check	Without glue logic	
	rw_check	With glue logic	
	logic_ram	RAM is mapped to the technology primitive LRAM2K_SDP	

Vendor	Values	Implementation	Technology
Intel FPGA		Default: Device-specific memories (TriMatrix)	Stratix, Hardcopy II, Cyclone, Arria families
	registers	Registers	

:

Vendor	Values	Implementation	Technology
Intel FPGA	block_ram	Device-specific memory	
	block_ram, no_rw_check/ rw_check	Device-specific RAM without/with glue logic	Stratix Cyclone V Arria V families
	M4K	M4K RAMs	Stratix, Stratix GX Stratix II, Hardcopy II Stratix II GX Cyclone, Cyclone II families
	M512	M512 RAMs	Stratix, Stratix GX Stratix II, Hardcopy II
	M-RAM	M RAMs	Stratix II GX Arria GX, Arria II GX families
	M4K, no_rw_check/ rw_check	M4K RAM without/with glue logic	Stratix Stratix GX Stratix II
	M512, no_rw_check/ rw_check	M512 RAM without/with glue logic	Stratix II GX families
	M-RAM, no_rw_check/ rw_check	M-RAM without/with glue logic	
	M144K	M144K RAMs	Stratix III, Stratix IV Stratix IV GT families
	M9K	M9K RAMs	Stratix III, Stratix IV, Stratix IV GT Cyclone III, Cyclone IV E Cyclone IV GX families
MLAB	M9K, no_rw_check/ rw_check	M9K RAM implemented without/with glue logic	Stratix III Stratix IV families
	M144K, no_rw_check/ rw_check	M144K RAM implemented without/with glue logic	
	MLAB	Memory LAB resources configured as LUTRAM	Stratix III, Stratix IV Stratix IV GT, Stratix V families

Vendor	Values	Implementation	Technology
	M20K	M20K RAM	Stratix V, Stratix 10, Agilex
	M20K, no_rw_check/ rw_check	M20K RAM without/with glue logic	Cyclone V Arria V families
	ecc	ECC RAM and its associated glue logic in triplicate for high reliability	Arria V, Cyclone-III Cyclone-IV, Cyclone-V Stratix-III, Stratix-IV Stratix-V families
	tmr	TriPLICATE RAMs for high reliability	

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 ORCA families
	distributed	Distributed RAM or PFU resources	
		Default: block_ram	iCE40 family
	registers	Registers	
	block_ram	Device-specific RAMs	
	block_ram, no_rw_check/ rw_check	RAMs without/with glue logic	

Vendor	Values	Implementation	Technology
Microchip	block_ram	Default: block_ram	ProASIC3/ ProASIC3E/ ProASIC3L ProASICplus Axcelerator families
	registers	Registers	
	block_ram	Device-specific RAMs	
	block_ram, no_rw_check/ rw_check	RAMs without/with glue logic	
		Default: Registers	SmartFusion, Fusion IGLOO+, IGLOO IGLOOe families
	lsram	RAM1K18, RAM1K18_RT	RTG4, IGLOO2, SmartFusion2 families
	uram	RAM64X18, RAM64X18_RT	
	registers	Registers	
	no_rw_check/ rw_check	RAMs without/with glue logic	
	ecc, set	RAM1K18_RT, RAM64X18_RT	RTG4 family
low_power	low_power	RAMs with low power mode	PolarFire, RTG4, IGLOO2, SmartFusion2 families
	no_low_power	Turns off the low power mode for a device-specific RAM, if the global low power option is on.	PolarFire, RTG4, IGLOO2, SmartFusion2 families

Vendor	Values	Implementation	Technology
Xilinx		Default: block_ram. If not possible, select_ram.	Spartan-3, Spartan-6, and Virtex families
	registers	Registers and combinational logic	
	block_ram	Block RAM	

Vendor	Values	Implementation	Technology
	block_ram, no_rw_check/ rw_check	Block RAM with/without glue logic	
	block_ram, area	Same as block_ram, but with wider, more area-efficient block RAMs instead of the default deeper block RAMs.	
	block_ram, power	Power and area-efficient block RAM	
	select_ram	Distributed RAM	
	no_rw_check, power	Power and area-efficient RAM with no glue logic	
	ecc	ECC RAM and its associated glue logic in triplicate for high reliability	Artix-7, Kintex-7, Virtex-5 and newer families
	tmr	Triplicate RAMs for high reliability	
	uram, no_uram	UltraRAM block	Kintex, Virtex, Zynq UltraScale+ families

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 ...****syn_ramstyle**

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:

- Use `no_rw_check` to eliminate bypass logic. If you enable global RAM inference with the Read Write Check on RAM option, you can use `no_rw_check` to selectively disable glue logic insertion for individual RAMs.
- Use `rw_check` to insert bypass logic. If you disable global RAM inference with the Read Write Check on RAM option, you can use `rw_check` to selectively enable glue logic insertion for individual RAMs.

syn_rw_conflict_logic

You know your design does not read and write to the same address simultaneously and you want to let the tool select the RAM implementation. See [syn_rw_conflict_logic , on page 591](#) for details.

Read Write Check on RAM

You want to globally enable or disable glue logic insertion for all the RAMs in the design.

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

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

`syn_ramstyle` Syntax

FDC `define_attribute {signalname [bitRange]} syn_ramstyle value`
`define_global_attribute syn_ramstyle value`

[FDC Example](#)

Verilog `object /* synthesis syn_ramstyle = value */`

[Verilog Example](#)

VHDL `attribute syn_ramstyle of object : objectType is value ;`

[VHDL Example](#)

FDC Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	<any>	<global>	syn_ramstyle	select_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="select_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
  always @ (posedge CLK)
  if (EN && WE) mem[ADDR] = data_in;
endmodule
```

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.ALL;
library synplify;
```

```
entity RAMB4_S4 is
    port (ADDR: in std_logic_vector(7 downto 0);
          data_in : in std_logic_vector(3 downto 0);
          WE : in std_logic;
          CLK : in std_logic;
          RST : in std_logic;
          EN : in std_logic;
          data_out : out std_logic_vector(3 downto 0));
end RAMB4_S4;

architecture rtl of RAMB4_S4 is
type mem_type is array (255 downto 0) of std_logic_vector (3 downto 0);
signal mem : mem_type;
-- mem is the signal that defines the RAM
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "select_ram";

begin
    process (CLK)
    begin
        IF (CLK'event AND CLK = '1') THEN
            IF (EN = '1') THEN
                IF (RST = '1') THEN
                    data_out <= "0000";
                ELSE
                    IF (WE = '1') THEN
                        data_out <= data_in;
                    ELSE
                        data_out <= mem(to_integer(unsigned(ADDR)));
                    END IF;
                END IF;
            END IF;
        END IF;
    end process;

    process (CLK)
    begin
        IF (CLK'event AND CLK = '1') THEN
            IF (EN = '1' AND WE = '1') THEN
                mem(to_integer(unsigned(ADDR))) <= data_in;
            END IF;
        END IF;
    end process;
end rtl;
```

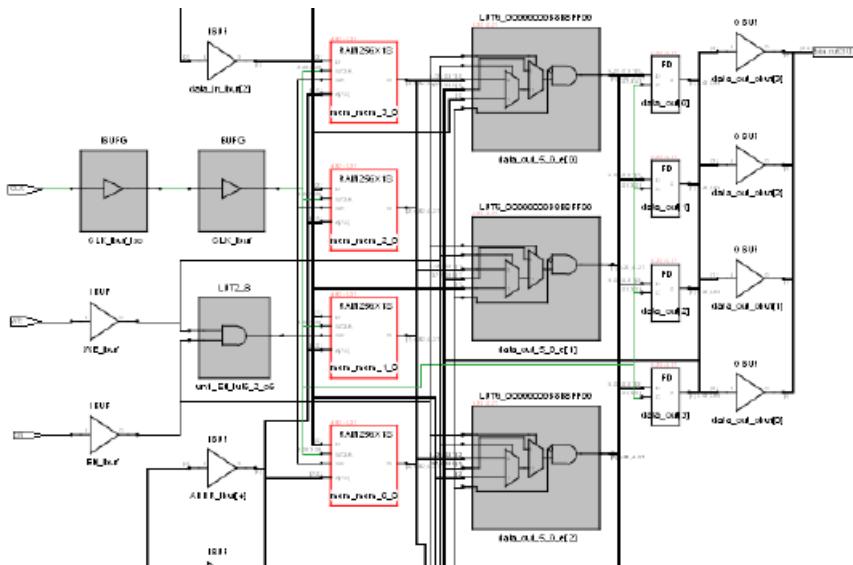
Effect of Setting syn_ramstyle in Xilinx Designs

The following are some examples of how the syn_ramstyle setting directs the implementation of RAM.

Distributed RAM (Select RAM) Example

Verilog reg [3:0] mem [255:0] /* synthesis syn_ramstyle="select_ram" */;

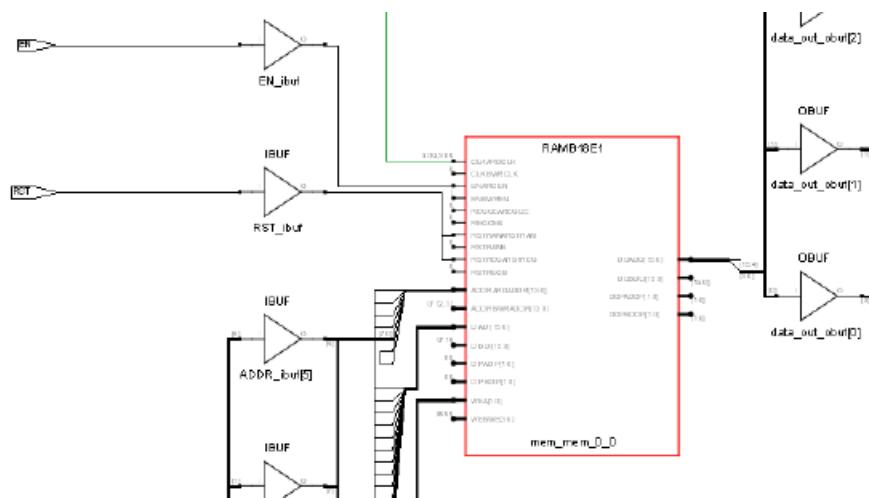
VHDL attribute syn_ramstyle of mem : signal is "select_ram";

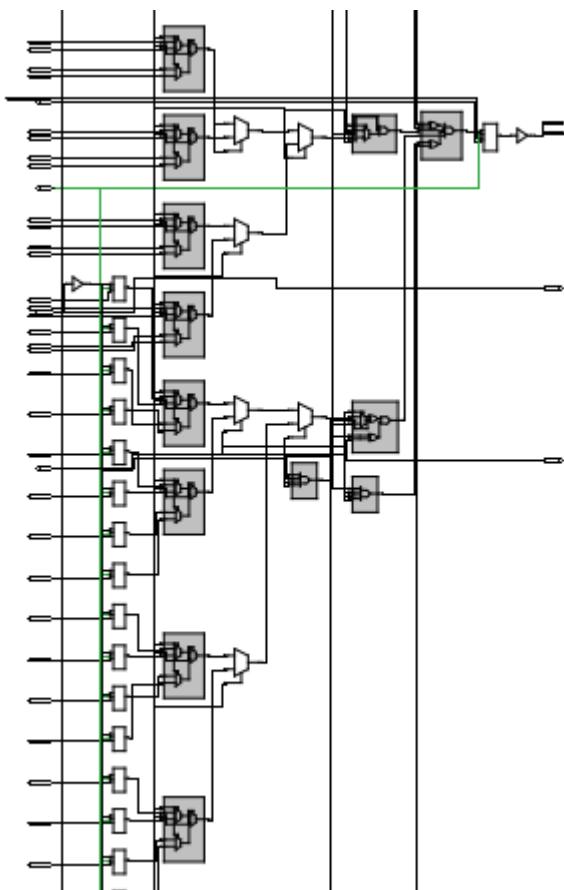


Block RAM Example

Verilog reg [3:0] mem [255:0] /* synthesis syn_ramstyle="select_ram" */;

VHDL attribute syn_ramstyle of mem : signal is "select_ram";





syn_reduce_controlset_size

Lattice, Xilinx

Attribute

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

Vendor	Devices
Lattice	iCE40
Xilinx	Spartan 6 Virtex-4, Virtex-5, Virtex-6, Virtex-7 families

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
Xilinx	Yes	Integer up to the maximum number of control sets. Default: Device-dependent	

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.

Xilinx The tool picks the default number of control-sets based on the target technology, QoR, and the resources available.

syn_reduce_controlset_size Syntax

FDC `define_attribute {v:object} syn_reduce_controlset_size value` [FDC Example](#)
`define_global_attribute syn_reduce_controlset_size value`

Verilog `object /* synthesis syn_reduce_controlset_size = value */` [Verilog Example](#)

VHDL `attribute syn_reduce_controlset_size of object : objectType is value;` [VHDL Example](#)

FDC Example

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

```
define_global_attribute syn_reduce_controlset_size 3
```

Verilog Example

```
//Example 1: Verilog syn_reduce_controlset_size 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

module test(i1, r1,s1, e1, clk, o1) /* synthesis
    syn_reduce_controlset_size = 3 */;
input [3:1] i1;
input clk;
input r1;
input s1;
input [2:1] e1;
output [3:1] o1;
reg reg1, reg2, reg3;

//reg1 FDRSE
always@(posedge clk)
if (r1)
    reg1 <= 1'b0;
else if (s1)
    reg1 <= 1'b1;
else if (e1[1])
    reg1 <= i1[1];
```

```

//reg2 FDRSE
always@(posedge clk)
if (r1)
    reg2 <= 1'b0;
else if (s1)
    reg2 <= 1'b1;
else if (e1[1])
    reg2 <= i1[2];

//reg3 FDRSE
always@(posedge clk)
if (r1)
    reg3 <= 1'b0;
else if (s1)
    reg3 <= 1'b1;
else if (e1[2])
    reg3 <= i1[3];

assign o1 = {reg3,reg2,reg1};
endmodule

```

VHDL Example

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
port (rst : in std_logic;
      set : in std_logic;
      en : in std_logic_vector(1 downto 0);
      clk : in std_logic;
      din : in std_logic_vector (2 downto 0);
      dout : out std_logic_vector (2 downto 0));
end entity test;

architecture test_arch of test is
attribute syn_reduce_controlset_size : integer;
attribute syn_reduce_controlset_size of test_arch :
  architecture is 3;

begin
process(clk, rst, set, en(0))
begin
  if (clk='1' and clk'event) then
    if (rst = '1') then
      dout(0) <= '0';
    else if (set = '1') then

```

```
dout(0) <= '1';
else if (en(0) = '1') then
    dout(0) <= din(0);
end if;
end if;
end if;
end if;
end process;

process(clk, rst, set, en(0))
begin
if (clk='1' and clk'event) then
    if (rst = '1') then
        dout(1) <= '0';
    else if (set = '1') then
        dout(1) <= '1';
    else if (en(0) = '1') then
        dout(1) <= din(1);
    end if;
end if;
end if;
end if;
end process;

process(clk, rst, set, en(1))
begin
if (clk='1' and clk'event) then
    if (rst = '1') then
        dout(2) <= '0';
    else if (set = '1') then
        dout(2) <= '1';
    else if (en(1) = '1') then
        dout(2) <= din(2);
    end if;
end if;
end if;
end 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);

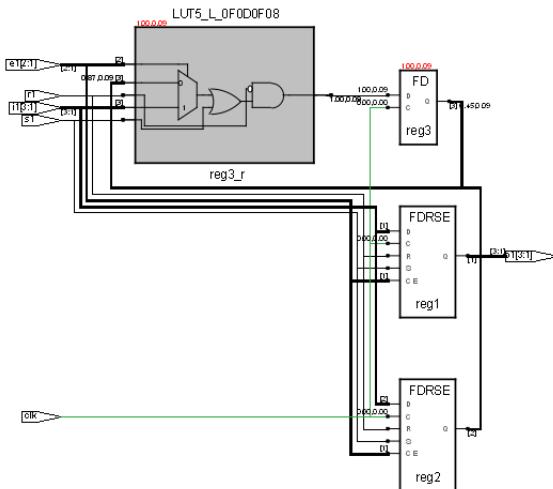
  end if;
end if;
```

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

Effect of Using syn_reduce_controlset_size in Xilinx Designs

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

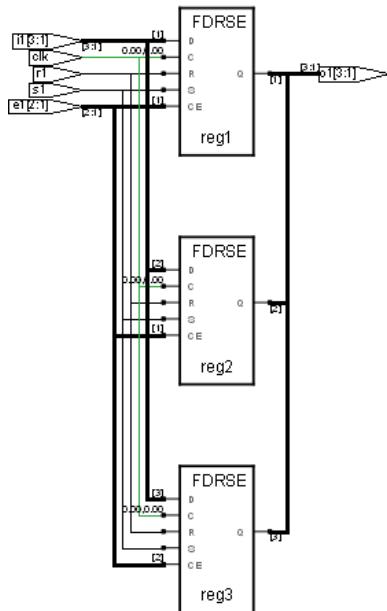
The following examples show how `syn_controlset_size` affects the implementation of control-sets in a Xilinx Virtex-5 design. This is the design without the attribute. The default value is 2.



Example 1: syn_reduce_controlset_size = 1

Verilog /* synthesis syn_reduce_controlset_size=1 */;

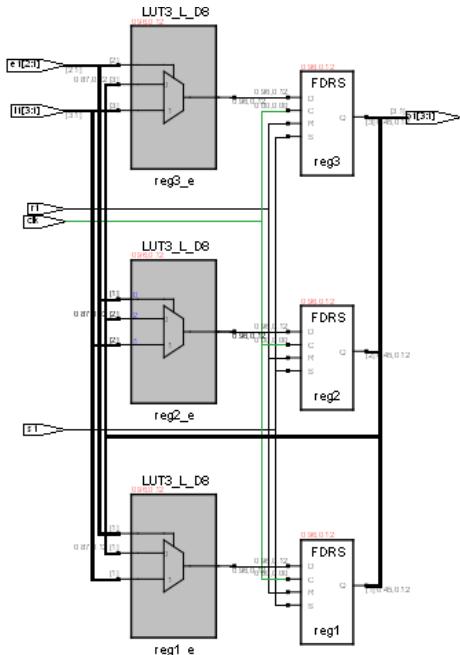
VHDL attribute syn_reduce_controlset_size of test_arch : architecture is 1;



Example 2: syn_reduce_controlset_size = 3

Verilog /* synthesis syn_reduce_controlset_size=3 */;

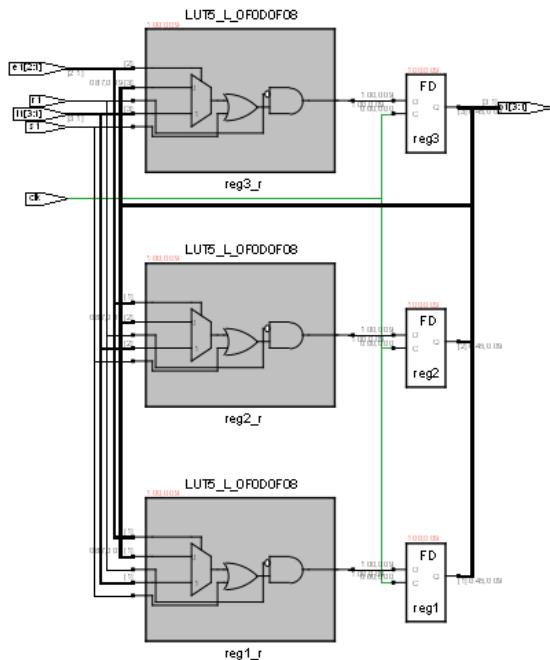
VHDL attribute syn_reduce_controlset_size of test_arch : architecture is 3;



Example 3: syn_reduce_controlset_size = 4

Verilog /* synthesis syn_reduce_controlset_size=4 */;

VHDL attribute syn_reduce_controlset_size of test_arch : architecture is 4;



syn_reduce_controlset_size (Achronix)

Attribute

Controls the minimum size of the unique control-set on which control-set optimizations can occur. Control sets are unique combinations of clock-enable, and synchronous reset/set signals.

Vendor	Technology
Achronix	Speedster7t and newer families

syn_reduce_controlset_size Values

Value	Description
1 (Default)	The minimum fanout threshold limit for control-set signals, above which the control-set optimizations occur. There is no upper limit.

Description

Reducing the unique control-set signals that drive registers can help improve placement QoR. By default, the minimum size of the unique control-set is 1.

For any other control-set size, if a signal has fanout less than the specified size driving the control net input for the register, then the control-set optimization does not connect the signal to the dedicated set/reset and enable pin of the register. The control signals are moved to the data input of the registers.

This attribute does not apply for asynchronous set/reset registers. Registers with datapaths that are timing critical are excluded from this optimization.

syn_reduce_controlset_size Syntax

Global Support Object

Yes	Top-level module or architecture
-----	----------------------------------

:

The following table summarizes the syntax in different files:

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

FDC Example

You can specify `syn_reduce_controlset_size` as a compiler directive (cdc) or as a constraint in the constraint file (fdc).

```
define_global_attribute syn_reduce_controlset_size {3}
define_attribute {v:work.test} syn_reduce_controlset_size {3}
```

Verilog Example

Suppose there are four registers (DFFC) with two registers driven by R1 reset and two registers driven by R2 reset. If `syn_reduce_controlset_size` is set to 2, then reset R1 and R2 are connected directly to the reset pin of DFFC.

However, when `syn_reduce_controlset_size` is set to 3 or greater, then all the registers are converted to DFFs with the reset logic moved to the datapath.

```
module test(i1, i2, i3, i4, r1, r2, clk, o1, o2, o3, o4)
/* synthesis 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;

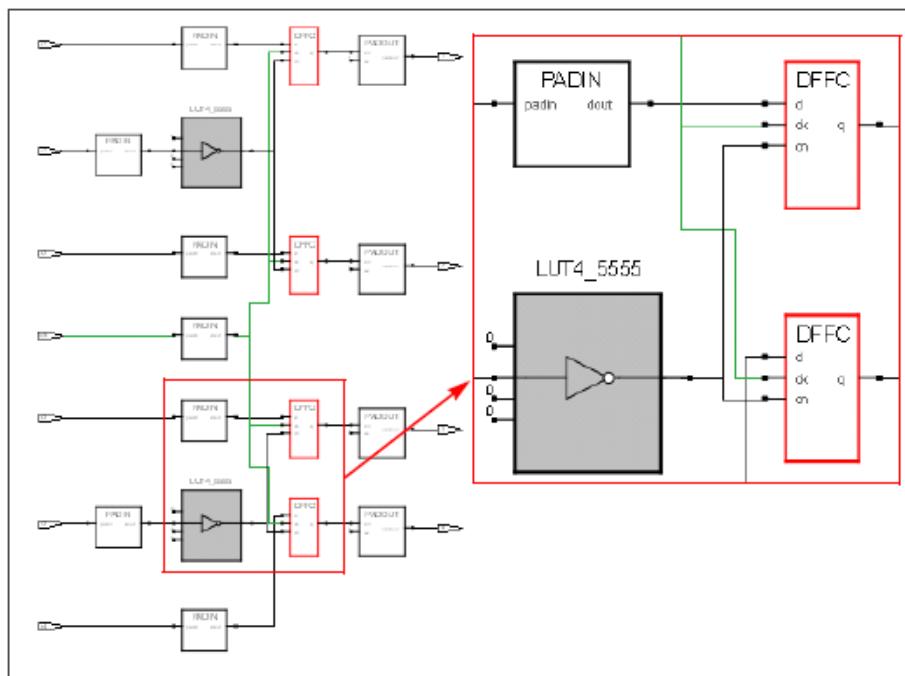
always@(posedge clk)
if (r1)
reg1 <= 1'b0;
else
reg1 <= i1;

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
```



VHDL Example

When `syn_reduce_controlset_size` is set to 7 for this example, all the registers are converted to DDFs with the reset logic moved to the datapath.

```
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_reduce_controlset_size : integer;
attribute syn_reduce_controlset_size of rtl : architecture is 7;

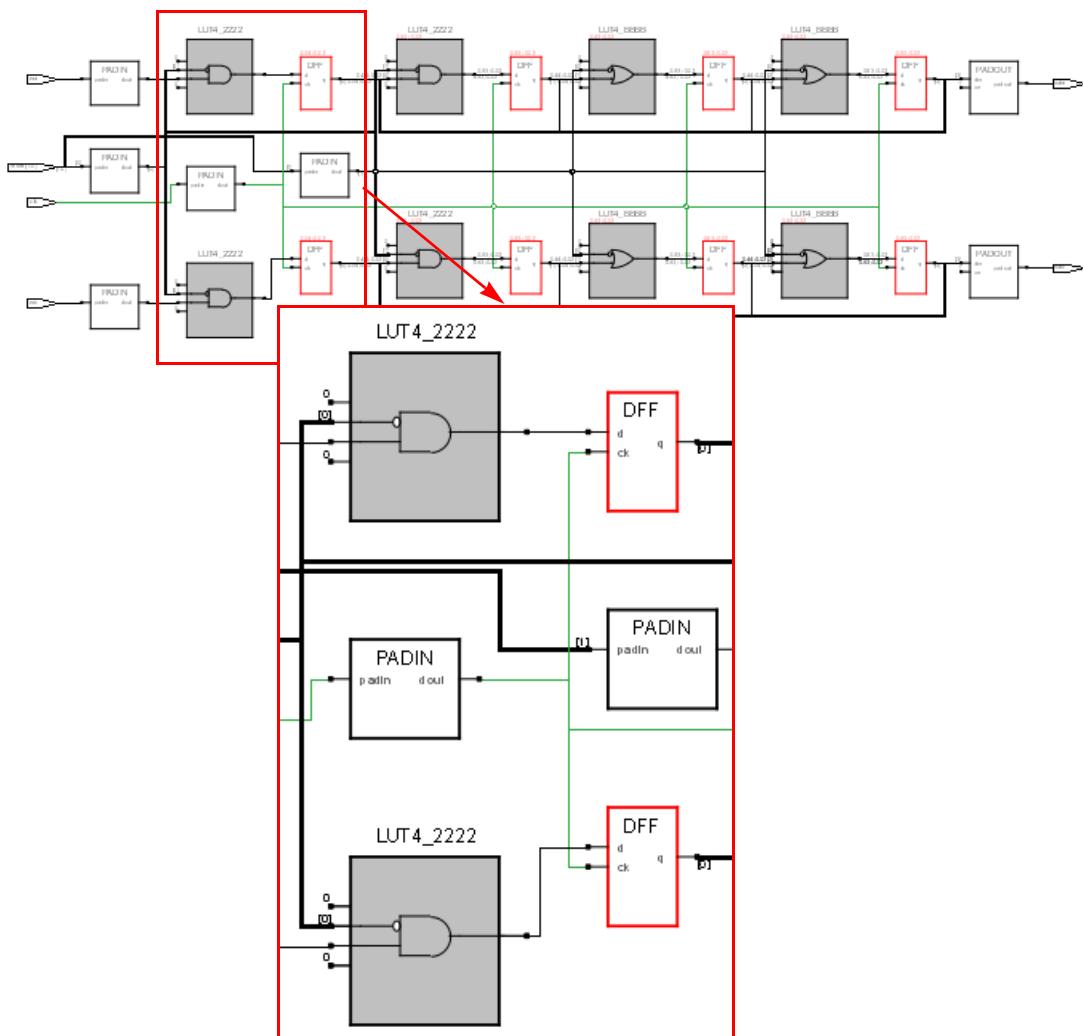
begin
process (clk,reset)
begin
if (clk'event and clk='1') then
if (reset(0)= '1') then
a_reg(0)<='0';
b_reg(0)<='0';
else
a_reg(0)<=ina;
b_reg(0)<=inb;
end if;
end if;
end process;

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
Achronix	Speedster families	-	-	Register
Intel FPGA	Stratix and newer families	-	-	Register
Lattice	iCE40, ECP4, ECP3 families	-	-	Register
Microchip	SmartFusion2, ProASIC3 families	-	-	Register
Xilinx	Virtex and newer 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 **define_attribute {register} syn_reference_clock {clockName}**

FDC Example

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							
Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type	Clock Group
clk	2.0 MHz	1609.5 MHz	500.000	0.621	499.379	declared	default_clkgroup_0
ref_clk	1.0 MHz	NA	1000.000	NA	NA	declared	default_clkgroup_1

This is the report after applying the attribute:

Performance Summary							

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type	Clock Group
clk	2.0 MHz	NA	500.000	NA	NA	declared	default_clkgroup_0
ref_clk	1.0 MHz	1609.5 MHz	1000.000	0.621	999.379	declared	default_clkgroup_1

syn_register_correction

Attribute

Implements single-bit error correction (SEC) and double bit error detection (DED) logic using Hamming 3 encoding.

syn_register_correction Syntax

Global	Object Type
No	FDC: Register instance HDL: Signal/reg

FDC define_attribute {i:u1.control_set[3:0]} } syn_register_correction
 {hamming3 | hamming3_ded}

[FDC Example](#)

Verilog reg [3:0] control_set /* synthesis
 syn_register_correction=hamming3 | hamming3_ded */;

[Verilog Example](#)

VHDL Attribute syn_register_correction : string
 Attribute syn_register_correction of control_set : signal is
 "hamming3 | hamming3_ded"

[VHDL Example](#)

FDC Example

	Enable	Object Type	Object	Attribute	Value
1	<input checked="" type="checkbox"/>	Instance	i:dff_u.control_set[7:0]	syn_register_correction	hamming3
2					

Verilog Example

```
module top (input clk, input reset, input a, output b)
reg [3:0] control_set /* synthesis
syn_register_correction=hamming3 */;
```

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
entity test is

    port (clk : in std_logic;
          a: in std_logic;
          b : out std_logic);

end test;
architecture rtl of test is
Signal control_set : std_logic_vector(3 downto 0);
attribute syn_register_correction : string;
attribute syn_register_correction of control_set : signal is
"hamming3";
```

Description

`syn_register_correction` implements SEC (hamming3) or single-bit error correction along with double-bit error detection (hamming3_ded) using Hamming 3 encoding.

When you specify the attribute value as hamming3 on register, SEC is implemented for register.

If hamming3 is applied on a register and distributed TMR (see [syn_radhardlevel](#)) is applied to a module containing this register, then hamming3 error correction is implemented for that register and also triplicated along with the rest of the module.

Note: This attribute on a register associated with multiplier or memory will affect the DSP and Block RAM inference.

syn_register_correction Values

hamming3 Builds single-bit error correction logic with Hamming 3 encoding.

hamming3_ded Builds single-bit error correction and double-bit error detection logic with Hamming 3 encoding.

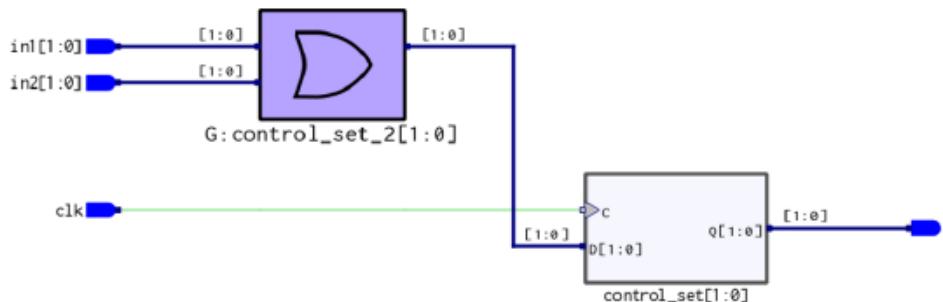
Limitations

`syn_register_correction` does not support:

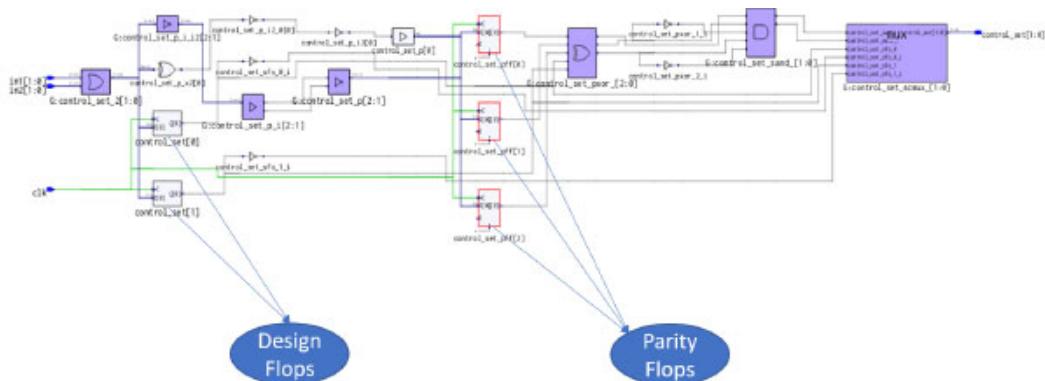
- Registers with initial value.
- Registers with both asynchronous reset and asynchronous set.
- Registers with both synchronous reset and synchronous set.

Effect of using `syn_register_correction`

The following figure shows without the `syn_register_correction=hamming3` error correction.



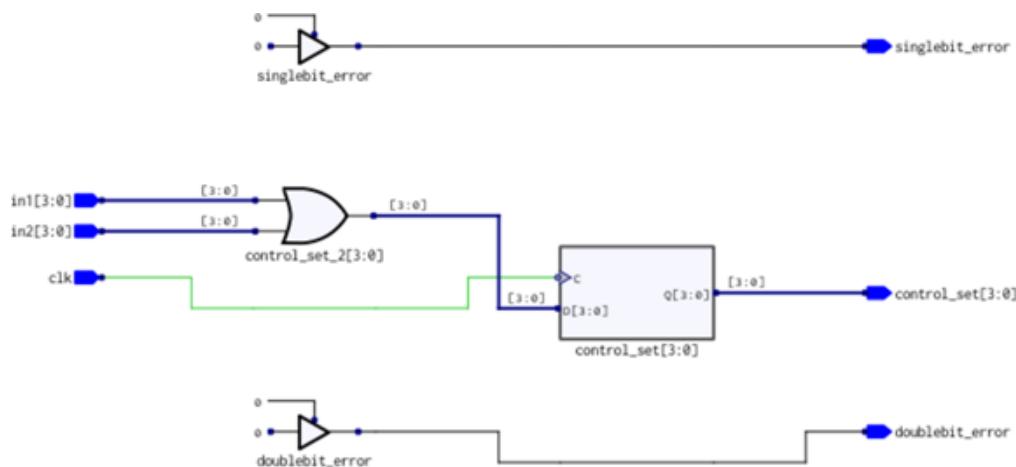
The following figure shows the register after Hamming 3 error correction is applied, with comparator circuits and parity flip-flops.



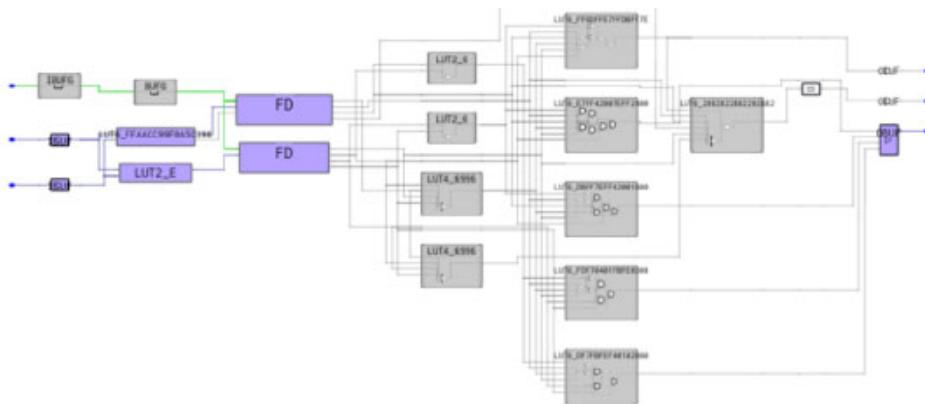
With hamming3_ded and error monitoring Tcl commands, EC comparator circuits and parity flip-flops along with double error flag, are added.

```
syn_create_err_net { -name {sbiterr} -inst {i:control_set[3:0]}  
-single_bit }  
syn_connect { -from {n:sbiterr} -to {p:singlebit_error} }  
syn_create_err_net { -name {dbiterr} -inst {i:control_set[3:0]}  
-double_bit }  
syn_connect { -from {n:dbiterr} -to {p:doublebit_error} }
```

The following figure shows without the `syn_register_correction=hamming3_ded` error correction.



The following figure shows the register after Hamming 3 error correction is applied, with double-bit error detection.



syn_rename_module

Directive

Renames a module.

syn_rename_module Values

Value	Description
<i>newModuleName</i>	Specifies a new name for the module.

Description

This directive renames the specified module. It is typically used to rename submodules to avoid possible naming conflicts between a generated name and the original module name. The `syn_rename_module` directive is specified through the cdc compiler directives file. The synthesis tool automatically replicates any necessary registers during optimization when fixing fanouts, packing I/Os, or improving the quality of results.

syn_rename_module Syntax Specification

CDC File `define_directive {moduleName} syn_rename_module
{newModuleName}`

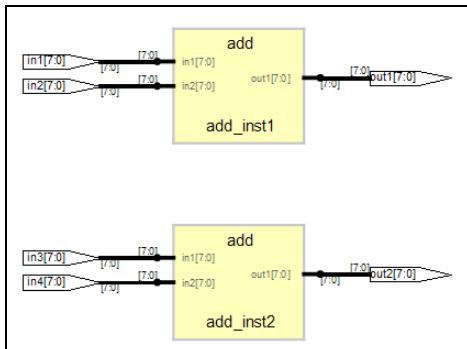
CDC File
Example

CDC File Example

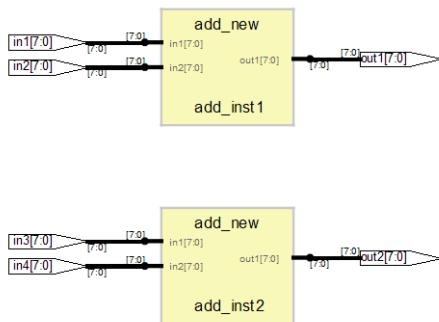
```
define_directive {v:work.add} syn_rename_module {add_new}
```

Effect of using syn_rename_module

The following figure shows the original design, before applying syn_rename_module:



The next figure shows the modules with new names after applying syn_rename_module:



syn_replicate

Attribute

Controls replication of registers during optimization.

Vendor	Technologies
Achronix	Speedster families
Lattice	iCE40, ECP4 families
Microchip	SmartFusion families
Xilinx	Virtex and newer 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.

Do not use the syn_replicate attribute with the fast synthesis option.

syn_replicate Syntax Specification

FDC	define_global_attribute syn_replicate {0 1};	FDC Example
Verilog	object /* synthesis syn_replicate = 1 0 */;	Verilog Example
VHDL	attribute syn_replicate : boolean; attribute syn_replicate of object : signal is true false;	VHDL Example

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;

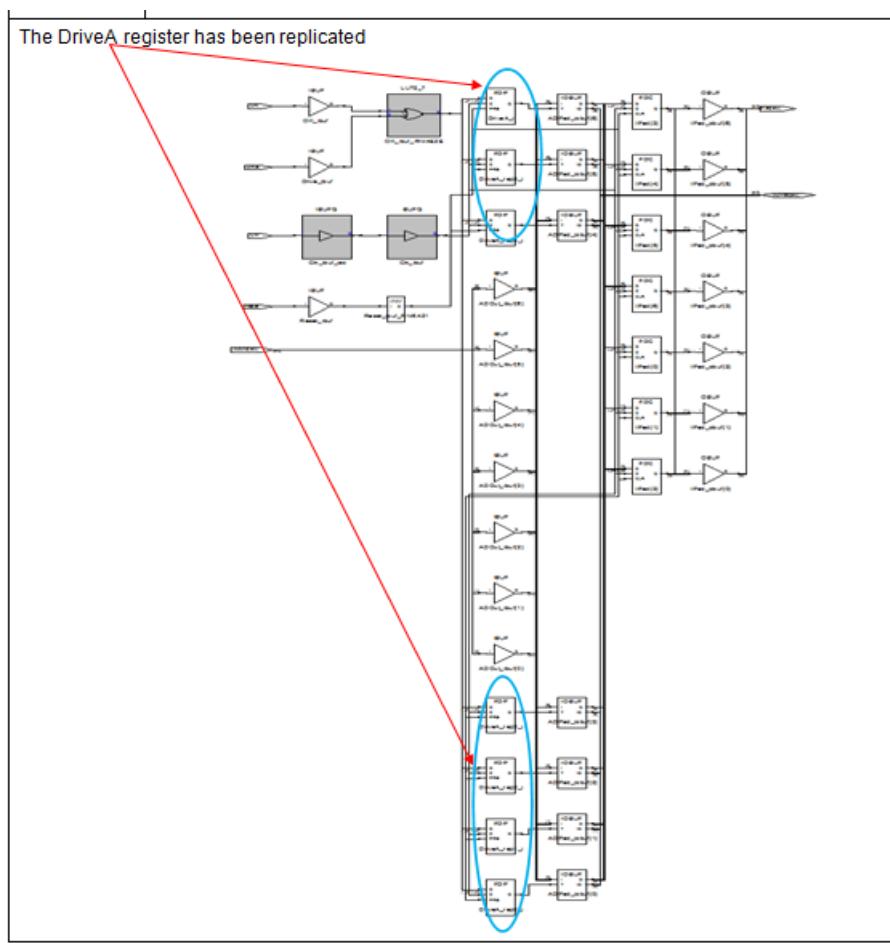
architecture archnorep of norep is
signal DriveA : std_logic;
attribute syn_replicate : boolean;
attribute syn_replicate of DriveA : signal is false;
begin
ADPad <= ADOut when DriveA='1' else (others => 'Z');
process (Clk, Reset)
begin
    if Reset='0' then
        DriveA <= '0';
        IPad <= (others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

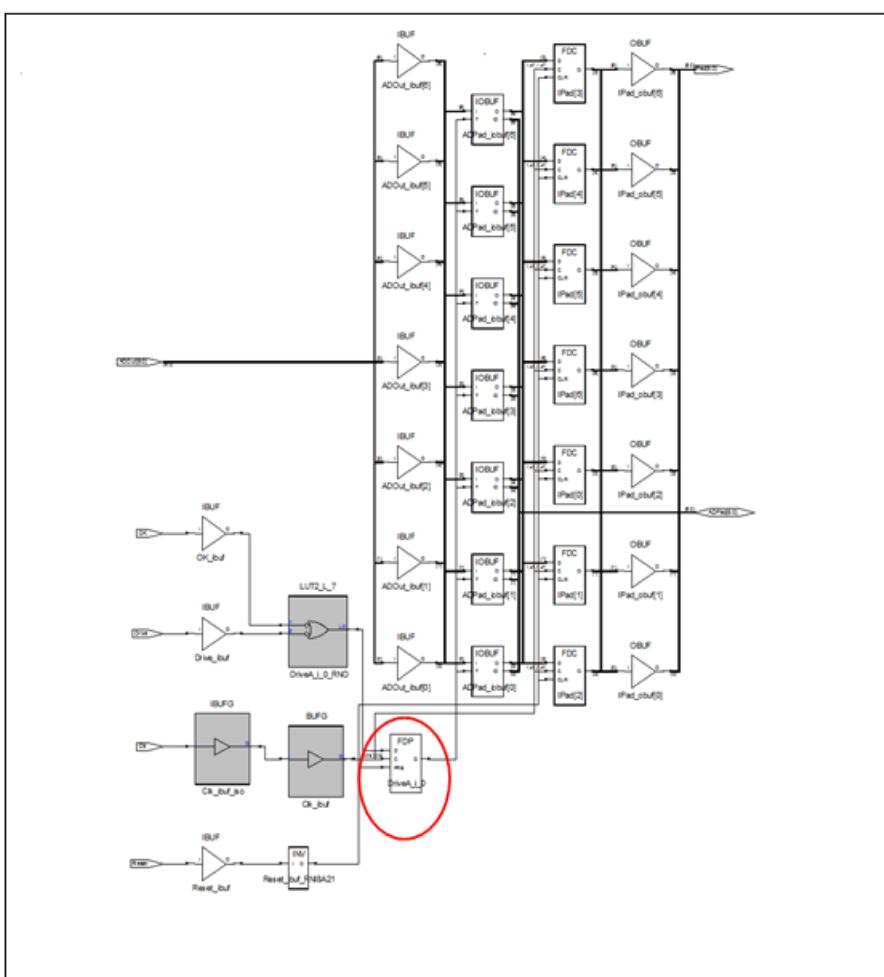
Verilog	<code>reg DriveA /*synthesis syn_replicate=1*/;</code>
VHDL	<code>attribute syn_replicate : boolean;</code> <code>attribute syn_replicate of DriveA : signal is true;</code>



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_resources

Attribute

Specifies the resources used inside a black box.

Vendor	Technology
Achronix	Speedster families
Intel FPGA	All families
Microchip	ProASIC3, IGLOO, Fusion, SmartFusion and newer families
Xilinx	All families

syn_resources Values

Global Support Object

No	Module or architecture
----	------------------------

The value for this attribute can be specified with any combination of the following:

Value	Description
luts=integer	Number of lookup tables
regs=integer	Number of registers
blockrams=integer	Number of RAM resources

Value	Description
blockmults=integer	Number of block multipliers resources
corecells=integer	<i>Microchip families only</i> Number of core cells
dsp_blocks=integer	<i>Intel FPGA and Xilinx families only</i> Number of DSP blocks

The value listed in the area usage report is the larger of the `luts` or `regs` value.

Vendor-specific usage model includes the following support:

- For the Intel FPGA Cyclone and Stratix families, the total number of logic elements (LEs) includes those that use only the LUT, those that use only the register, and those that use both. Use the total number of LEs for both the `luts` and `regs` values, even if this is larger than the actual number of LUTs or actual number of registers.
- For Intel FPGA designs, use this attribute instead of the obsolete `alter-a-area`.
- The Microchip families only support resource values of `blockrams` and `corecells`.
- The Xilinx families support DSP and DSP48 blocks.

Description

Specifies the resources used inside a black box. This attribute is applied to Verilog black-box modules and VHDL architectures or component definitions.

syn_resources Syntax

The following table summarizes the syntax in different files.

FDC	<pre>define_attribute {v:moduleName} syn_resources {luts=integer reg=integer blockrams=integer lrams=integer} dsp_blocks=integer blockmults=integer}</pre>	FDC Example
	<i>Microchip only</i>	
	<pre>define_attribute {v:moduleName} syn_resources {corecells=integer blockrams=integer}</pre>	
Verilog	<pre>object /* synthesis syn_resources = value */;</pre>	Example - Verilog syn_resources (Microchip)
VHDL	<pre>attribute syn_resources : string; attribute syn_resources of object : objectType is value;</pre>	Example - VHDL syn_resources (Microchip)

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	v:b	syn_resources	corecells=300, blockrams=5	string	Specifies the resources used inside a black box

You can apply the attribute to more than one kind of resource at a time by separating assignments with a comma (,), For example:

```
define_attribute {v:bb} syn_resources {corecells=300, blockrams=5}
define_attribute {v:bb} syn_resources {luts=500, regs=400,
    blockrams=10, lrams=2}
define_attribute {v:bb} syn_resources {luts=500, blockrams=10}
```

Example - Verilog syn resources (Xilinx)

//Example: Verilog syn resources (Xilinx)

```
module bb (o,i) /* synthesis syn_black_box syn_resources =
    "luts=500, regs=463, blockrams=10, blockmults=2" */;
    input i;
    output o;
```

```
:  
endmodule  
  
module top_bb (o,i);  
    input i;  
    output o;  
    bb u1 (o,i);  
endmodule
```

In Verilog, you can only attach this attribute to a module. Here is the example.

Example - Verilog syn_resources (Microchip)

```
//Example: Verilog syn_resources (Microsemi)  
  
module bb (o,i) /* synthesis syn_black_box syn_resources =  
    "corecells=300, blockrams=10" */;  
    input i;  
    output o;  
endmodule  
  
module top_bb (o,i);  
    input i;  
    output o;  
    bb u1 (o,i);  
endmodule
```

In Verilog, you can only attach this attribute to a module. Here is the example,

Example - VHDL syn_resources (Xilinx)

```
--Example: VHDL syn_resources (Xilinx)
```

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

entity top is
port (o : out std_logic;
      i : in std_logic
     );
end top;

architecture top_rtl of top is

component bb
port (o : out std_logic;
      i : in std_logic);
end component;

begin
U1: bb port map(o, i);
end top_rtl;

--black box entity
library ieee;
use ieee.std_logic_1164.all;

entity bb is
```

```
:  
  
port (o : out std_logic;  
      i : in std_logic  
    );  
  
end bb;  
  
architecture rtl of bb is  
  
attribute syn_resources : string;  
attribute syn_resources of rtl: architecture is "lutss=500,  
regs=463, blockrams=10, blockmults=2";  
  
begin  
end rtl;
```

In VHDL, this attribute can be placed on either an architecture or a component declaration.

Example - VHDL syn_resources (Microchip)

--Example: VHDL syn_resources (Microsemi)

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity top is  
port (o : out std_logic;  
      i : in std_logic
```

```

) ;

end top;

architecture top_rtl of top is

component bb
port (o : out std_logic;
      i : in std_logic);
end component;

begin
U1: bb port map(o, i);
end top_rtl;

--black box entity
library ieee;
use ieee.std_logic_1164.all;

entity bb is
port (o : out std_logic;
      i : in std_logic
) ;

end bb;

```

```
:  
  
architecture rtl of bb is  
  
attribute syn_resources : string;  
attribute syn_resources of rtl: architecture is "corecells=300,  
blockrams=10";  
  
begin  
end rtl;
```

In VHDL, this attribute can be placed on either an architecture or a component declaration.

Verilog Example (Achronix)

In Verilog, you can only attach this attribute to a module. Here is an example:

```
module bb (o,i) /* synthesis syn_black_box syn_resources =  
    "luts=500,regs=463,blockrams=10,lrams=2" */;  
    input i;  
    output o;  
endmodule  
  
module top_bb (o,i);  
    input i;  
    output o;  
    bb u1 (o,i);  
endmodule
```

VHDL Example (Achronix)

See [VHDL Attribute and Directive Syntax, on page 401](#) for different ways to specify VHDL attributes and directives. In VHDL, this attribute can be placed on either an architecture or a component declaration.

```
architecture top of top is  
component decoder  
    port (clk : in bit;  
          a, b : in bit;  
          qout : out bit_vector(7 downto 0) );  
end component;
```

```
attribute syn_resources : string;
attribute syn_resources of decoder: component is
    "luts=500,regs=463,blockrams=10,lrams=2";

-- Other code
```

Effect of Using `syn_resources` (Xilinx)

You can check the Resource Utilization report in the log file to verify how resources are actually mapped.

```
Resource Usage Report for top_bb

Mapping to part: xc7vx485tffg1157-1
Cell usage:
bb           1 use

I/O ports: 2
I/O primitives: 2
IBUF          1 use
OBUF          1 use

I/O Register bits:          0
Register bits not including I/Os: 463 (0%)

RAM/ROM usage summary
Occupied Block RAM sites (RAMB36) : 10 of 1030 (0%)

DSP48s: 2 of 2800 (0%)
Total load per clock:

Mapping Summary:
Total LUTs: 500 (0%)

Distribution of All Consumed LUTs = RAM
Distribution of All Consumed Luts 500 = 500
```

Effect of Using syn_resources (Microchip)

You can check the Resource Utilization report in the log file to verify how resources are actually mapped.

```
Target Part: A3P015_QFN68_-1
Report for cell top.top_rtl
Core Cell usage:
    cell count      area count*area
  corecells      300        1.0        300.0 (blackbox) :bb
      GND          1        0.0        0.0
      VCC          1        0.0        0.0

  -----
  TOTAL         2        300.0

IO Cell usage:
    cell count
  INBUF          1
  OUTBUF         1
  -----
  TOTAL         2

Core Cells      : 300 of 384 (78%)
IO Cells        : 2
Mapper successful!
```

syn_ret_lib_cell_type

UPF Directive

The custom retention model must include this directive with the same value as specified by the `-lib_cell_type` option of the `map_retention_cell` command.

Syntax

syn_ret_lib_cell_type = "asicLibCellType"

Where the type of ASIC library cell, *asicLibCellType* is the same as specified by the -lib cell type option of the map_retention_cell command.

Example

```
module ret_dffrse (Q, CLK, SAVE, RESTORE, D, S, R, E)
/* synthesis syn_ret_lib cell_type = "asicLibCellType" */;
```


syn_ret_type

UPF Directive

Specifies the type of sequential element, such as a flip-flop, latch, RAM, or seqshift, for which the custom retention model is defined.

Syntax

syn_ret_type = "type"

Where the type of sequential element, *type* can be specified as described in the following table:

Type of Sequential Element	Description
Flip-flops	
DFF	Standard flip-flop
DFFE	Flip-flop with enable
DFFR	Flip-flop with async reset
DFFRE	Flip-flop with enable and async reset
DFFS	Flip-flop with async set
DFFSE	Flip-flop with enable and async set
DFFRS	Flip-flop with async reset and set
DFFRSE	Flip-flop with enable and async set and reset
SDFFR	Flip-flop with sync reset
SDFFRE	Flip-flop with enable and sync reset
SDFFS	Flip-flop with sync set
SDFFSE	Flip-flop with enable and sync set

Type of Sequential Element	Description
SDFFRS	Flip-flop with sync reset and set
SDFFRSE	Flip-flop with enable and sync reset and set
Latches	
LAT	Latch
LATR	Latch with reset
LATS	Latch with set
LATRS	Latch with reset and set
RAM	
RAM primitives are decomposed and mapped into one of the DFF or SDFF types of cell, accordingly.	
SeqShift	
SEQSHIFT primitives are decomposed and mapped into one of the DFF or SDFF types of cell, accordingly.	

Example

```

module ret_dffrse (Q, CLK, SAVE, RESTORE, D, S, R, E)
    /* synthesis syn_implement = "1" syn_upf_ret_type = "DFFRSE" */;
    logic
endmodule

```

syn_romstyle

Attribute

This attribute determines how ROM architectures are implemented.

Vendor	Technology
Achronix	Speedster7t families
Intel FPGA	Stratix, Cyclone and newer families
Lattice	ECP4, ECP3, iCE40 and iCE40UP, LIFCL (Lattice Radiant software) families
Xilinx	Virtex, Spartan and newer families

syn_romstyle Values

Value	Description
logic	<p><i>All supported technologies</i></p> <p>Uses discrete logic primitives such as LUTs, registers, and muxes.</p>
block_rom	<p>Specifies the inferred ROM be mapped to the technology-specific block RAM.</p> <p><i>Achronix technologies</i></p> <p>Specifies the inferred ROM be mapped to the technology primitive BRAM72K_SDP.</p>
logic_rom	<p><i>Achronix</i></p> <p>Specifies the inferred ROM be mapped to the technology primitive LRAM2K_SDP.</p>
lpm_rom	<p><i>Intel FPGA technologies</i></p> <p>Uses the lpm_rom configuration.</p>

Value	Description
MLAB	<p><i>Intel FPGA technologies</i></p> <p>Maps asynchronous ROM to MLAB resources for Arria GX, Arria II GX, or Stratix and newer device families.</p>
distributed	<p><i>Lattice technologies</i></p> <p>Implements the ROM structure as distributed ROM.</p>
select_rom	<p><i>Xilinx technologies</i></p> <p>Implements ROMs using the distributed ROM resources in the CLBs.</p>

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 logic_rom lpm_rom MLAB distributed select_rom}</code>	SCOPE Example
Verilog	<code>object /* synthesis syn_romstyle = "logic block_ram logic_rom lpm_rom MLAB distributed select_rom" */;</code>	Verilog Example
VHDL	<code>attribute syn_romstyle of object: objectType is "logic block_rom logic_rom lpm_rom MLAB distributed select_rom";</code>	VHDL Example

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;
            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'b11100: dataout <= 8'b01001110;
      5'b11101: dataout <= 8'b01110100;
      5'b11110: dataout <= 8'b11100101;
      5'b11111: dataout <= 8'b01111110;
      default: dataout <= 8'b00000000;
    endcase
  end
endmodule

```

VHDL Example

The following VHDL code example applies the `syn_romstyle` value of `block_rom`.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity single_port_rom is
  generic
  (
    DATA_WIDTH : natural := 8;
    ADDR_WIDTH : natural := 8
  );
  port
  (
    clk : in std_logic;

```

```
addr : in natural range 0 to 2**ADDR_WIDTH - 1;
q : out std_logic_vector((DATA_WIDTH-1) downto 0)
);
attribute syn_romstyle : string;
attribute syn_romstyle of q : signal is "block_rom";

end entity;
architecture rtl of single_port_rom is
    subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
    type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

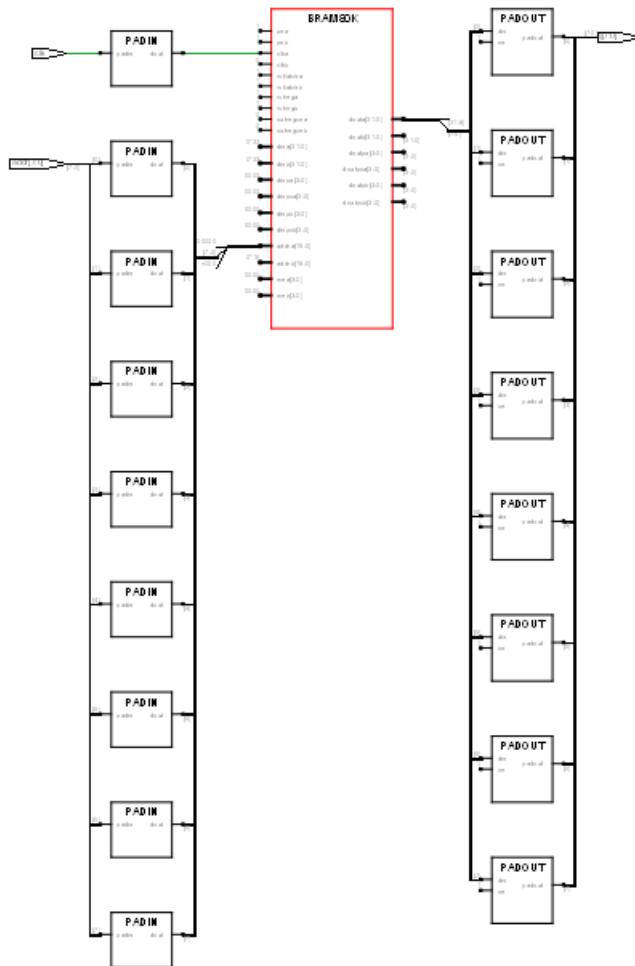
    function init_rom
        return memory_t is
        variable tmp : memory_t := (others => (others => '0'));
    begin
        for addr_pos in 0 to 2**ADDR_WIDTH - 1 loop
            tmp(addr_pos) := std_logic_vector(to_unsigned
                (addr_pos, DATA_WIDTH));
        end loop;
        return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

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

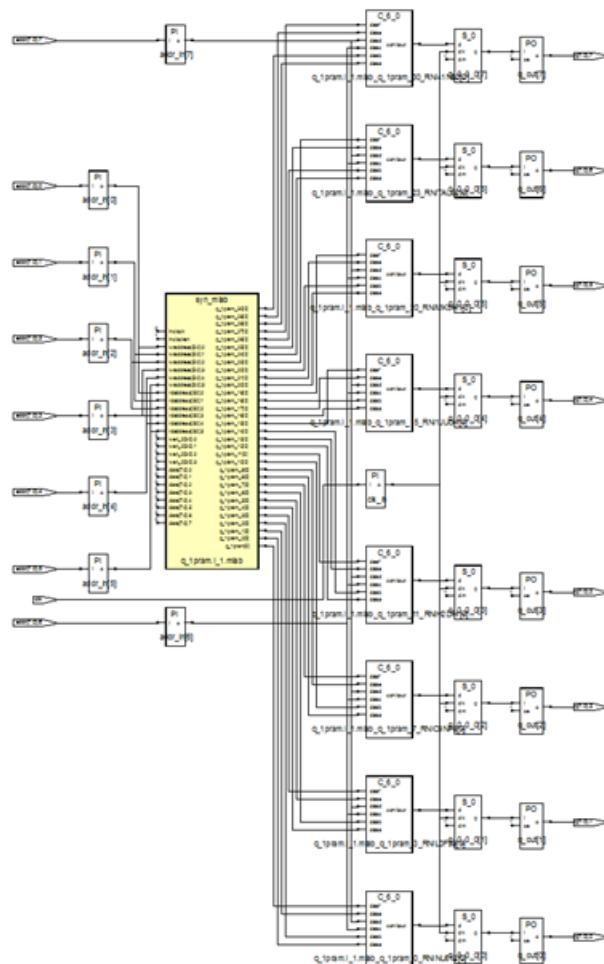
VHDL attribute syn_romstyle: string;
attribute syn_romstyle of q : signal is "block_rom";



Effect of Using syn_romstyle for Intel FPGA

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

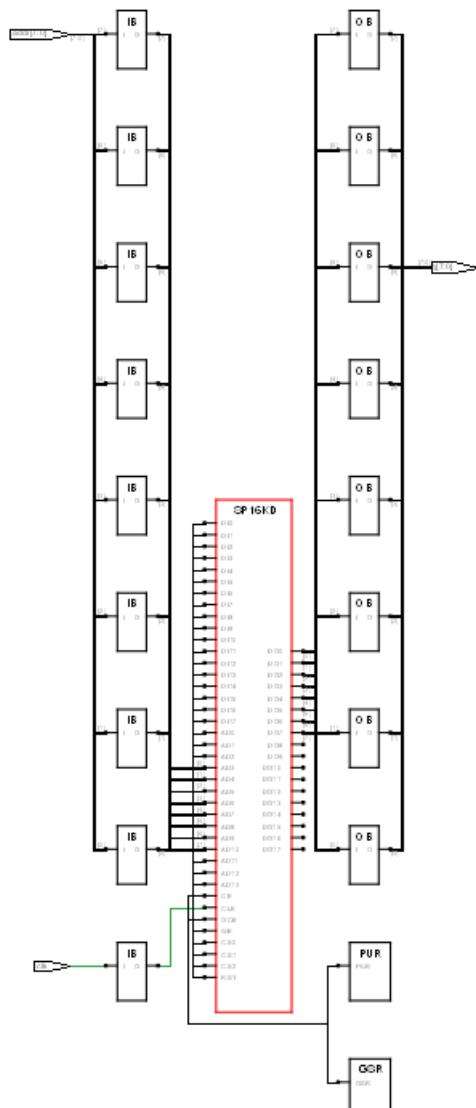
```
VHDL    attribute syn_romstyle: string;  
        attribute syn romstyle of q : signal is "MLAB";
```



Effect of Using syn_romstyle for Lattice

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

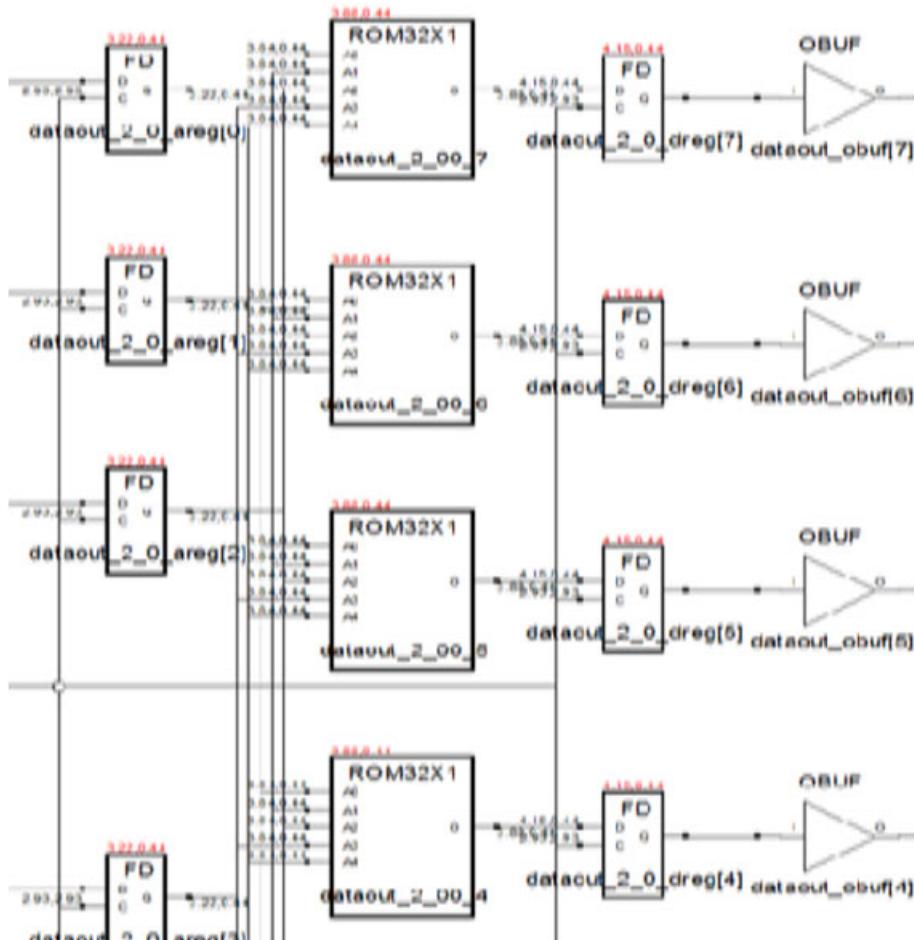
VHDL attribute syn_romstyle: string;
attribute syn_romstyle of q : signal is "block_rom";



Effect of Using syn_romstyle for Xilinx

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

VHDL attribute syn_romstyle: string;
attribute syn_romstyle of q : signal is "select_rom";



syn_romstyle (Microchip)

Attribute

This attribute determines how ROM architectures are implemented.

Vendor	Technology
Microchip	PolarFire

syn_romstyle Values

Value	Description
logic	ROM is inferred as registers or LUTs.
URAM sram	ROM is inferred as RAM1K20 or RAM64x12. Asynchronous ROM is mapped to RAM64x12 even if Isram attribute is applied.

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 RAM blocks. By default, small ROMs (less than twelve bits) are implemented as logic, and large ROMs (twelve or more 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	<pre>define_attribute {object} syn_romstyle {logic uram lsram} define_global_attribute syn_romstyle {logic uram lsram}</pre>	SCOPE Example
Verilog	<pre>object /* synthesis syn_romstyle = "logic uram lsram" */;</pre>	Verilog Example
VHDL	<pre>attribute syn_romstyle : string; attribute syn_romstyle of object : signal is "logic uram lsram";</pre>	VHDL Example

SCOPE Example

Enable	Object Type	Object	Attribute	Value	Value Type	Description
<input checked="" type="checkbox"/>	<any>	<Global>	syn_normstyle	isram	string	Inferred ROM implementation

Verilog Example

The following Verilog code example applies the `syn romstyle` value of block rom.

```
module test (clock,addr,dataout) /* synthesis syn_romstyle =
"lsram" */;
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;
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'b11110111;  
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 "uram";
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;
```


syn_rw_conflict_logic

Attribute/Directive

Infers an appropriate RAM implementation, and controls the insertion of glue logic if the inferred RAM is block RAM.

Vendor	Technologies
Achronix	Speedster7t
Intel FPGA	Stratix V and newer devices (Use <code>syn_ramstyle=blockram</code> for other Intel FPGA devices.)
Xilinx	Virtex-7, Spartan-6, and newer devices

syn_rw_conflict Values

- | | | |
|---|---------|--|
| 1 | Default | Infers an appropriate RAM implementation and generates glue logic if a block RAM is inferred. |
| 0 | | Infers an appropriate RAM implementation, but does not infer glue logic for any inferred block RAM. This setting could result in a simulation mismatch if you try to read and write to the same address. |

Description

The `syn_rw_conflict_logic` allows the synthesis tool to choose other RAM implementations as appropriate, instead of automatically inferring block RAM.

When you read and write to the same block RAM address, the value of the output is indeterminate. An indeterminate output can create a simulation mismatch between RTL and post-synthesis simulations. With the default setting for this attribute, if block RAM is inferred, the synthesis tools automatically generate bypass logic around the RAM to prevent mismatches.

You can specify that the bypass logic not be inserted if the tool implements block RAM, by setting the attribute value to 0. Glue logic insertion only applies to block RAM, not for any other RAM implementations. Use this setting only when you cannot simultaneously read and write to the same RAM location and you want to minimize overhead logic.

syn_rw_conflict_logic and Other Read-Write Checks

This attribute is similar in functionality to the `syn_ramstyle` attribute and its `no_rw_check` value. For a comparison of the different read-write address check options available, see [Read-Write Address Checks, on page 517](#).

syn_rw_conflict_logic Syntax

Name	Global Attribute	Object
<code>syn_rw_conflict_logic</code>	Yes	Module or RAM instance

You can specify `syn_rw_conflict_logic` as a compiler directive or as a constraint in the constraint file. The following table summarizes the syntax in different files:

FDC	<code>define_attribute {object} syn_rw_conflict_logic {0 1}</code> <code>define_global_attribute syn_rw_conflict_logic {0 1}</code>	SCOPE Example
Verilog	<code>object /* synthesis syn_rw_conflict_logic = 0 */</code>	Example — Verilog syn_rw_conflict_logic
VHDL	<code>attribute syn_rw_conflict_logic : boolean;</code> <code>attribute syn_rw_conflict_logic of Object : Object Type is value ;</code>	Example — VHDL syn_rw_conflict_logic

SCOPE Example

	Enabled	Object Type	Object	Attribute	Value
1	<input checked="" type="checkbox"/>		<global>	<code>syn_rw_conflict_logic</code>	0

Example — Verilog syn_rw_conflict_logic

```
//Example 1: Verilog syn_rw_conflict_logic example
module ram2Kx8 (data0, waddr0,raddr0, we0, clk0, dataout);
```

```

parameter d_width = 8;
parameter addr_width = 11;
parameter mem_depth = 2048;
input [d_width-1:0] data0;
input [addr_width-1:0] waddr0, raddr0;
input we0, clk0;
output [d_width-1:0] dataout;
reg [addr_width-1:0] reg_addr0;
reg [d_width-1:0] mem [mem_depth-1:0] /* synthesis
syn_rw_conflict_logic = 0 */;
assign dataout = mem[reg_addr0];

always @ (posedge clk0)
begin
    reg_addr0 <= raddr0;
    if (we0)
        mem[waddr0] <= data0;
end
endmodule

```

Example — VHDL syn_rw_conflict_logic

```
--Example 2: VHDL syn_rw_conflict_logic example

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

entity ram2Kx8 is port(
    data_out : out std logic vector(7 downto 0);
```

```
data_in : in std_logic_vector(7 downto 0);
raddr0 : in std_logic_vector(10 downto 0);
waddr0 : in std_logic_vector(10 downto 0);
clk, we : in std_logic);
end entity;

architecture beh of ram2Kx8 is
type mem_type is array (2047 downto 0) of std_logic_vector
(7 downto 0);
signal raddr0_in : std_logic_vector(10 downto 0);
signal mem : mem_type;
attribute syn_rw_conflict_logic : boolean;
attribute syn_rw_conflict_logic of mem : signal is false ;
begin
    data_out <= mem(conv_integer(raddr0_in));

process(clk)
begin
    if clk'event and clk='1' then
        if (we = '1') then
            mem(conv_integer(waddr0)) <= data_in;
        end if;
    end if;
end process;

process(clk)
begin
```

```
if clk'event and clk='1' then
    raddr0_in  <= raddr0;
end if;
end process;

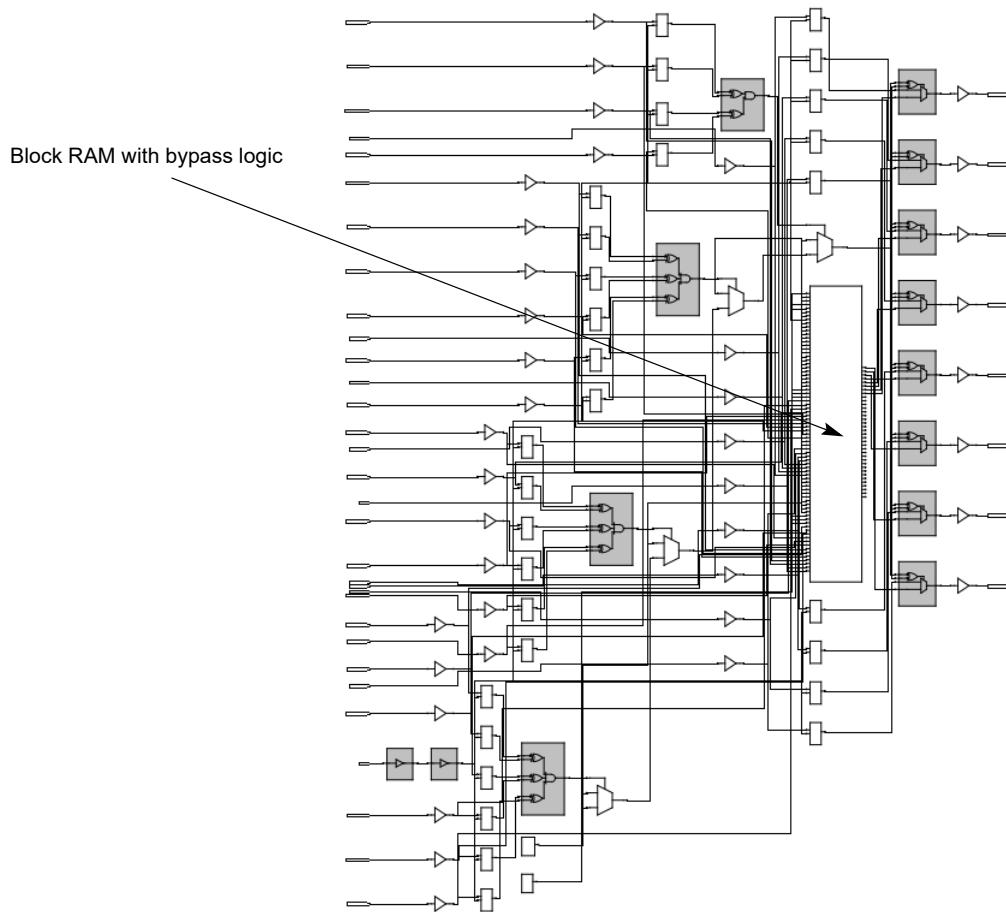
end beh;
```

Effect of Using syn_rw_conflict_logic

The following figure shows the attribute set to 1. The design shows a block RAM with bypass logic inserted:

Verilog `reg[d_width-1:0]mem[mem_dep-1:0]/*synthesis syn_rw_conflict_logic=1*/;`

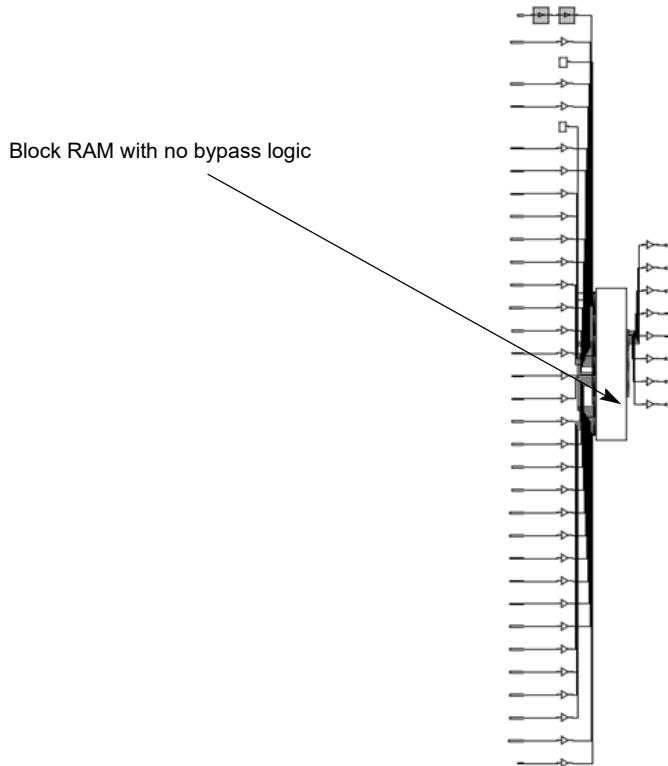
VHDL `attribute syn_rw_conflict_logic of mem : signal is true ;`



The next figure shows the attribute set to 0. The tool does not insert bypass logic with this value:

Verilog `reg[d_width-1:0]mem[mem_dep-1:0]/*synthesis syn_rw_conflict_logic=0*/;`

VHDL `attribute syn_rw_conflict_logic of mem : signal is false;`



syn_safe_case

Directive

Synplify Pro, Synplify Premier

This directive enables/disables the safe case option.

Vendor	Technologies
Intel FPGA	Arria V, Cyclone III, Cyclone IV, Cyclone V, Stratix III, Stratix IV, Stratix V, Stratix 10, and Agilex families
Lattice	ECP3 and MachXO2 families
Microchip	SmartFusion/2, RT ProASIC3, ProASIC3/3E/3L, IGLOO/+/E/2, 54SX, 54SX-A, eX, 40MX, Axcelerator families
Xilinx	Artix-7, Kintex-7, and Virtex families

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.

Note: 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.

For more information, see [Specifying Safe FSMs, on page 930](#).

syn_safe_case Syntax

Verilog	<code>module /* syn_safe_case = "1 0" */;</code>	Verilog Example
VHDL	<code>attribute syn_safe_case : boolean; attribute syn_safe_case of architectureName: architecture is "true false";</code>	VHDL Example
CDC	<code>define_directive {<view name>} {syn_safe_case} {1}</code>	CDC 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";
```

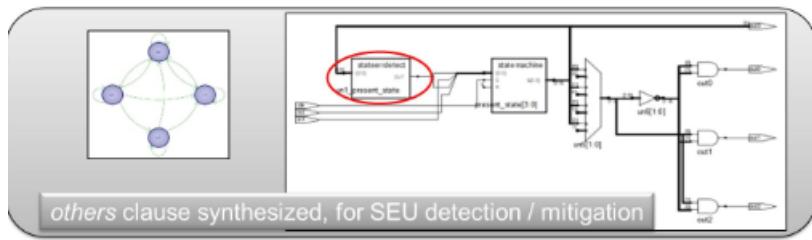
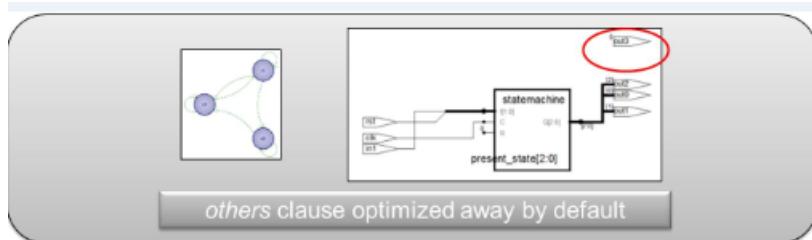
CDC Example

For example:

```
define_directive {v:state_test} {syn_safe_case} {1}
```

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_safefsm_pipe

Attribute

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

Vendor	Technologies
Intel FPGA	Arria V, Cyclone III, Cyclone IV, Cyclone V, Stratix III, Stratix IV, Stratix V, Stratix 10, and Agilex families
Lattice	ECP3 and MachXO2 families
Microchip	SmartFusion/2, RT ProASIC3, ProASIC3/3E/3L, IGLOO/+E/2, 54SX, 54SX-A, eX, 40MX, Axcelerator families
Xilinx	Artix-7, Kintex-7, and Virtex families

syn_safefsm_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_safefsm_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_safefsm_pipe Syntax

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

FDC Example

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

```
define_global_attribute syn_safefsm_pipe {0}
```

Verilog Example

The `syn_safefsm_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_safefsm_pipe = 0 */;
  parameter s1 = 2'b00; parameter s2 = 2'b01;
  parameter s3 = 2'b10; parameter s4 = 2'b11;
```

VHDL Example

The `syn_safefsm_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;
```

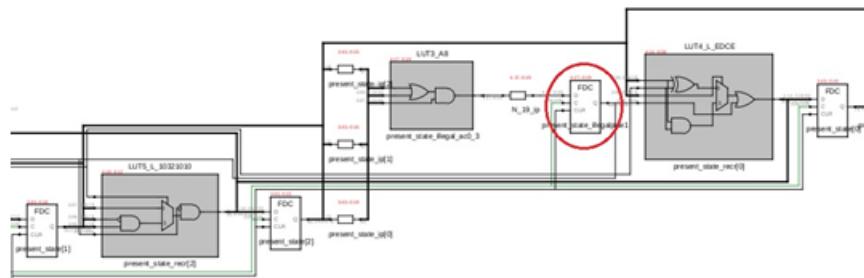
```

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_safefsm_pipe : string;
attribute syn_safefsm_pipe of state : signal is "false";

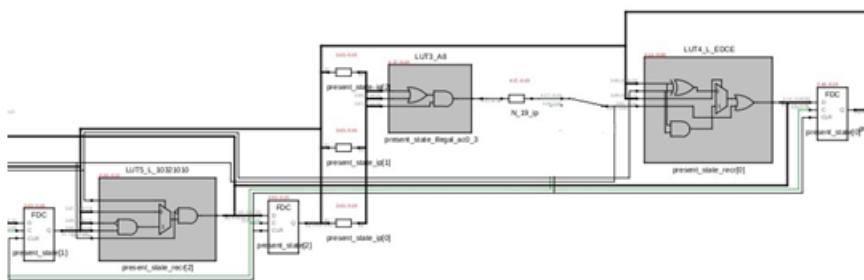
```

Effects of Using `syn_safefsm_pipe`

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



When the `syn_safefsm_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
0 off	Does not share resources during the compilation stage of synthesis.
1 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 581](#) in the *User Guide* for a detailed procedure.

:

syn_sharing Syntax

CDC File	define_directive {object} {syn_sharing} {on off}	CDC Example
Verilog	object /* synthesis syn_sharing="on off" */;	Verilog Example
VHDL	attribute syn_sharing of object : objectType is "on off";	VHDL Example

CDC Example

```
define_directive {v:module_add} {syn_sharing} {on}
```

Verilog Example

```
module add (a, b, x, y, out1, out2, sel, en, clk)
  /* synthesis syn_sharing=0 */;
  input a, b, x, y, sel, en, clk;
  output out1, out2;
  wire tmp1, tmp2;
  assign tmp1 = a * b;
  assign tmp2 = x * y;
  reg out1, out2;

  always@(posedge clk)
    if (en)
      begin
        out1 <= sel ? tmp1: tmp2;
      end
    else
      begin
        out2 <= sel ? tmp1: tmp2;
      end
endmodule
```

VHDL Example

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

entity add is
    port (a, b : in std_logic_vector(1 downto 0);
          x, y : in std_logic_vector(1 downto 0);
          clk, sel, en: in std_logic;
          out1 : out std_logic_vector(3 downto 0);
          out2 : out std_logic_vector(3 downto 0));
end add;

architecture rtl of add is
attribute syn_sharing : string;
attribute syn_sharing of rtl : architecture is "on";

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

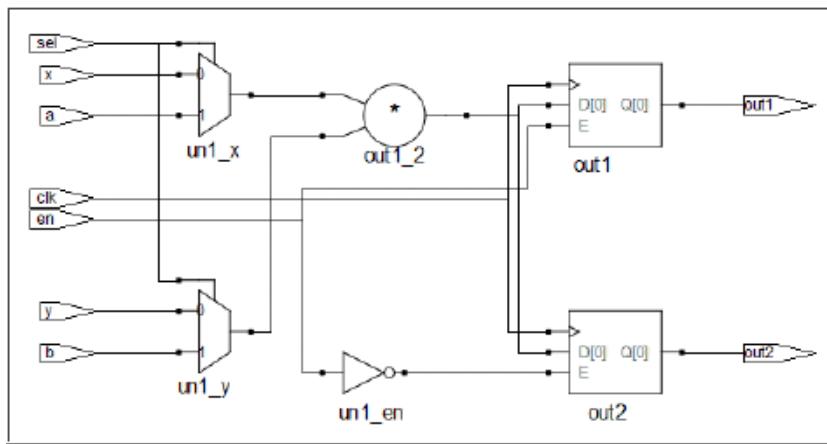
process(clk) begin
    if clk'event and clk='1' then
        if (en='1') then
            if (sel='1') then
                out1 <= tmp1;
            else
                out1 <= tmp2;
            end if;
        else
            if (sel='1') then
                out2 <= tmp1;
            else
                out2 <= tmp2;
            end if;
        end if;
    end if;
end process;
end rtl;
```

Effect of Using syn_sharing

The following example shows the default setting, where resource sharing in the compiler is on:

Verilog module add /* synthesis syn_sharing = "on" */;

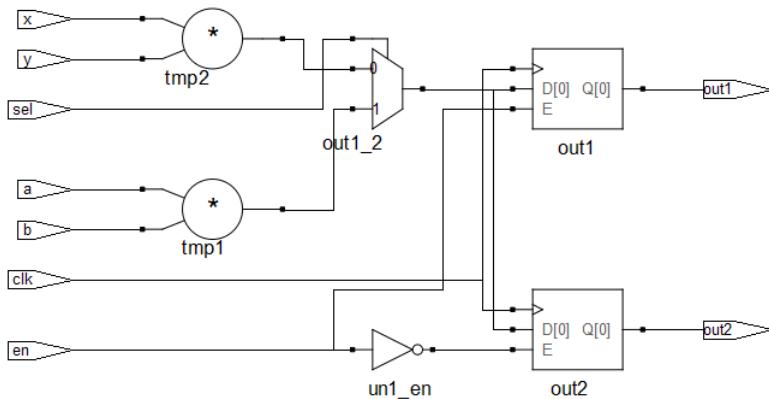
VHDL attribute syn_sharing of add : architecture is "on";



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

Synplify Pro, Synplify Premier

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
Achronix	Speedster7t
Intel FPGA	Cyclone V, Cyclone IV, Stratix V, Stratix 10, Agilex
Lattice	ECP3, MachXO2
Microchip	SmartFusion/2, RT ProASIC3, ProASIC3/3E/3L, IGLOO/+/-/E/2, Axcelerator, 54SX, 54SX-A, eX, 40MX
Xilinx	Virtex-7, Virtex-6, Virtex-5

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 259](#).

syn_shift_resetphase Syntax

Global Support	Object
Yes	FSM instance

The following table summarizes the syntax in different files:

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

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

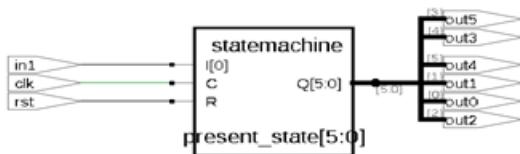
  attribute syn_shift_resetphase : boolean;
  attribute syn_shift_resetphase of rtl : architecture is false;

  -- Specifying on the state signal

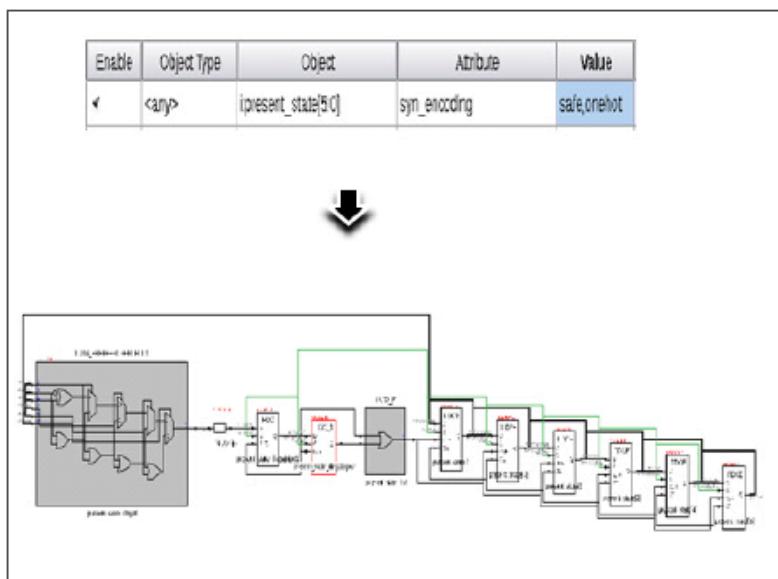
  attribute syn_shift_resetphase : boolean;
  attribute syn_shift_resetphase of present_state : signal is false;
begin
  ...
end rtl;
```

Effect of Using syn shift resetphase

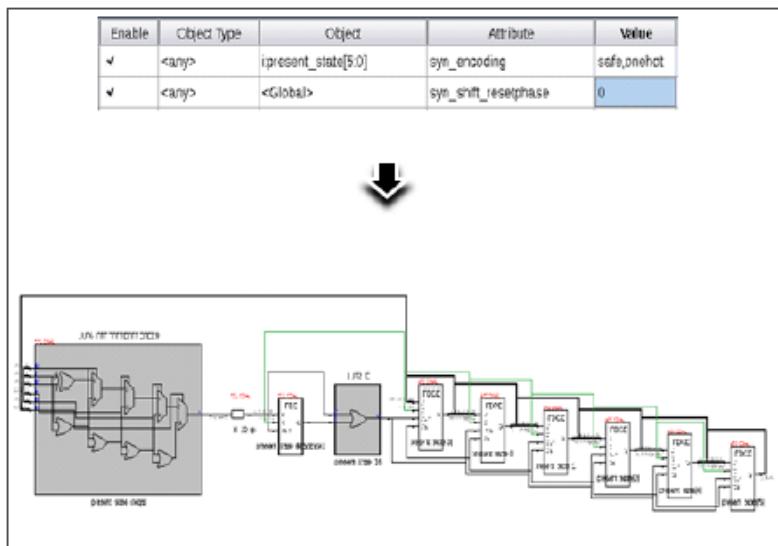
Safe encoding is implemented for the following state machine.



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



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



syn_slow

Attribute

Increases transition time of a specific output port or ports.

Vendor	Devices
Xilinx	Virtex6, Spartan6, older devices

syn_slow Values

Value	Description
1 True (Default)	Increases the transition time.

Description

The transition time of the output driver can be programmed as either fast or slow. The `syn_slow` attribute increases the transition time which reduces the noise level in the circuit.

You can speed up the transition time with the `syn_fast` attribute.

The synthesis tool provides attributes, passed into the XNF/EDIF netlist for Xilinx placement and routing, that affect the input setup times and output transition times for your I/Os. The `syn_slow` timing attribute gets passed to the Xilinx I/O Block parameter as SLOW in the XNF/EDIF netlist.

syn_slow Syntax

Global Support	Object
No	Output Port

:

The following table summarizes the syntax in different files:

FDC	<code>define_attribute {outputPort} syn_slow {1}</code>	SCOPE Example
Verilog	<code>outputPort /* synthesis syn_slow = 1 */;</code>	Verilog Example
VHDL	<code>attribute syn_slow of outputPort : objectType is true;</code>	VHDL Example

SCOPE Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description	Comment
1	<input checked="" type="checkbox"/>	<any>	p:portc[7:0]	syn_slow	1			

```
define_attribute {p:portc[7:0]} {syn_slow} {1}
```

Verilog Example

```
module counter (CLK, RST, DATA0);
  input CLK, RST;
  output [7:0] DATA0 /* synthesis syn_slow = 1 */;
  reg [7:0] DATA0;
  always @ (posedge CLK or posedge RST)
    begin
      if (RST)
        DATA0 = 0;
      else
        DATA0 = DATA0 + 1;
    end
endmodule
```

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity count is
  port(clk : in std_logic;
       rst : in std_logic;
       data0 : out std_logic_vector(7 downto 0)
     );
```

```

end count;

architecture behave of count is
    signal data0_i: std_logic_vector(7 downto 0);
begin
process (rst,clk)
begin
    if (clk'event and clk = '1') then
        if (rst = '1') then
            data0_i <= "00000000";
        else
            data0_i <= data0_i + "00000001";
        end if;
    end if;
end process;

data0 <= data0_i;
end behave;

```

Effect of Using syn_slow

The following table shows a design without the `syn_slow` attribute.

Verilog `output [7:0] DATA0 /* synthesis syn_slow = 0 */;`

VHDL `attribute syn_slow of data0 : signal is false;`

```

(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell counter (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port (array (rename DATA0 "DATA0[7:0]") 8) (direction OUTPUT))
        (port CLK (direction INPUT)
      )
      (port RST (direction INPUT)
    )
)

```

The following table shows a design with the `syn_slow` attribute.

Verilog `output [7:0] DATA0 /* synthesis syn_slow = 1 */;`

VHDL `attribute syn_slow of data0 : signal is true;`

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell counter (cellType GENERIC)
    (view verilog (viewType NELIST)
      (interface
        (port (array (rename DATA0 "DATA0[7:0]") 8) (direction OUTPUT)
          (property xc_slow (integer 1))
```

When place and route is run, a line is added for each output port bit attached to the `syn_slow` attribute in the following files:

```
<projectdirectory>\rev_1\pr_1\counter.pad(257):P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMS25*|2|12
|SLOW|||UNLOCATED|NO|NONE|
<projectdirectory>\rev_1\pr_1\counter_pad.txt(258):|P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMS25*|2|1
2|SLOW|||UNLOCATED|NO|NONE|
<projectdirectory>\rev_1\pr_1\syn_slow.edf(193):(property SLOW (string ""))
```

syn_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 *object* /* synthesis syn_smhigheffort = "0 | 1" */;

VHDL attribute syn_smhigheffort of <*object_name*>: *signal* is
"false | true";

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, rst, in1, out1);
    input clk, rst;
    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
endcase
endmodule
```

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

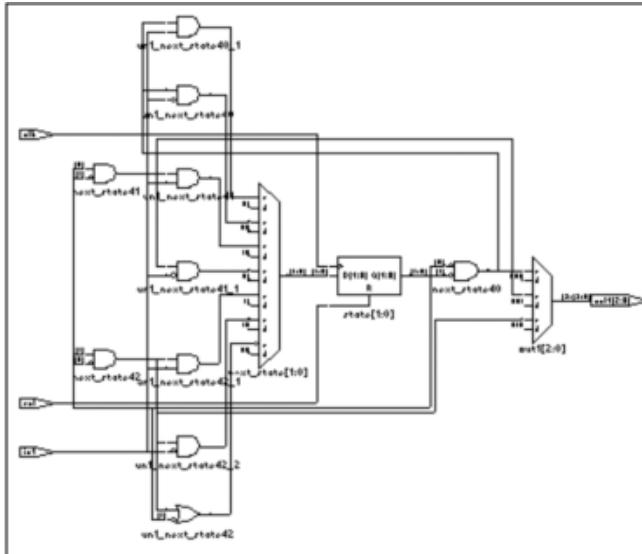
```
library ieee;
use ieee.std_logic_1164.all;
entity FSM1 is
    port (clk,rst,in1 : in std_logic;
          out1 : out std_logic_vector (2 downto 0));
end FSM1;
architecture behave of FSM1 is
type state_values is (s0, s1, s2,s3);
signal state, next_state: state_values;
attribute syn_smhigheffort : boolean;
attribute syn_smhigheffort 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;
        end case;
    end process;
end;
```

```
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;
```

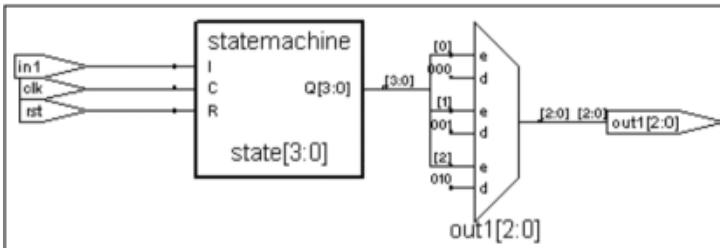
This is the VHDL source code used for the example in the following figure.

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 also:

- [syn_state_machine, on page 643](#) for information on enabling/disabling state-machine optimization on individual state registers.

syn_srl_mindepth

Attribute

The `syn_srl_mindepth` attribute lets you specify the threshold number of registers to pack into an SRL.

Vendor	Technology
Xilinx	Virtex-6 and newer families

syn_srl_mindepth Value

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

<code>NumOfRegisters</code>	Specifies the threshold number of registers to pack into an SRL.
-----------------------------	--

Description

This attribute lets you specify the threshold number of registers to pack into an SRL. This attribute helps guide the tool when it encounters a chain of registers.

Like `syn_srlstyle`, `syn_srl_mindepth` offers a shift register control. When `syn_srlstyle` is set globally to `select_srl`, the software infers an SRL for two-bit shift registers. With a global `syn_srl_mindepth` attribute, you can specify a register chain of three or more registers. Only Xilinx FD registers without set, reset, and enable logic can be used with this attribute.

Note that the synthesis software keeps the last register in a chain that feeds an output port unpacked, for better quality of results (QoR). You must adjust the `syn_srl_mindepth` value to account for how the tool implements this attribute. You can override this behavior by setting the `syn_srlstyle` attribute to `noextractff_srl`. This allows the final register of the chain to be packed into the SRL.

syn_srl_mindepth Syntax

Global Attribute Default Value

Yes 3 registers are packed into an SRL.

FDC define_global_attribute {syn_srl_mindepth} {7} [SCOPE Example](#)

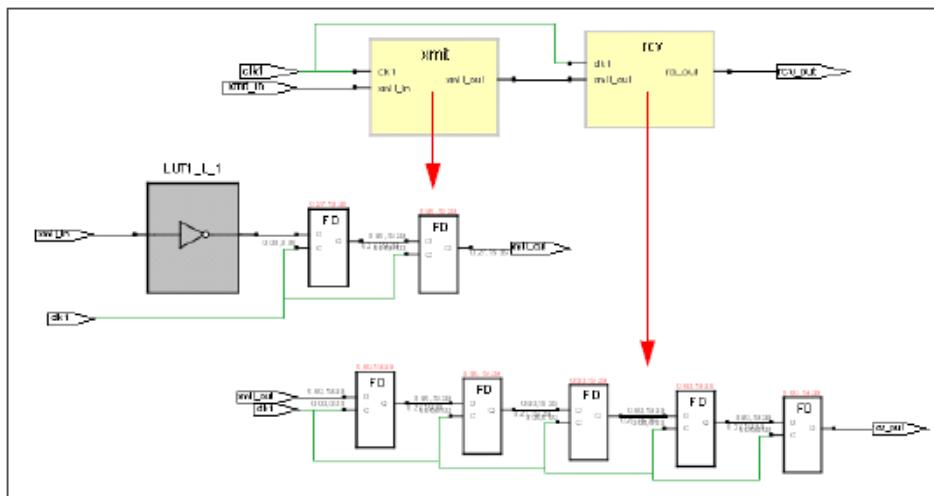
SCOPE Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	<Global>	syn_srl_mindepth	7		
2							

Effect of Using syn_srl_mindepth

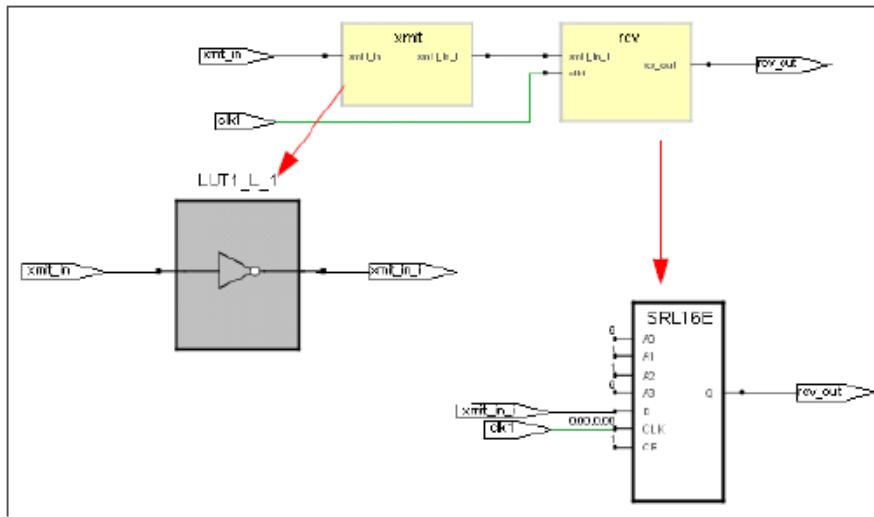
When you push down into the xmit and rcv blocks in the Technology view for the following example, you see they have two FD registers and five FD registers respectively. All seven registers are not packed into an SRL, despite the global attribute you set in the FDC file:

```
define_global_attribute {syn_srl_mindepth} {7}
```



To ensure that all seven registers get packed into the SRL16E primitive, set the attributes as shown below. The `syn_srlstyle` attribute overrides the default behavior of the tool and allows the last register to be packed into the SRL:

```
define_global_attribute {syn_srl_mindepth} {7}
define_global_attribute {syn_srlstyle} {noextractff_srl}
```



syn_srlstyle

Attribute

Determines how to implement the sequential shift components.

Vendor	Technology
Achronix	Speedster7t and newer families
Intel FPGA	Cyclone and newer families Stratix and newer families
Lattice	LatticeEC/ECP/ECP2 families LatticeSC/SCM families LatticeXP/XP2 families MachXO family ORCA families
Microchip	PolarFire
Xilinx	Virtex and newer families

syn_srlstyle Values

Technology	Value	Implements ...
Achronix		
Speedster7t families	registers	Specifies that seqshift be mapped to registers.
	logic_ram	Specifies that seqshift be mapped to LRAM640 irrespective of the depth or width threshold for the seqshift. For more information, see Achronix Shift Register Inference, on page 523 in the <i>Reference</i> manual.

Technology	Value	Implements ...
Intel FPGA		
Stratix families	altshift_tap <i>groupName</i>	Shift registers using the Intel Stratix altshift_tap (AST) megafunction. If you specify an optional <i>groupName</i> , the software ensures that shift registers with different group names are not packed together. You can also use other attributes to control packing, as described in Inferring Shift Registers, on page 537 in the <i>User Guide</i> .
Stratix and Cyclone families	block_ram	Maps shift registers to M9K, M20K, or M144K memories.
Stratix III and newer devices	MLAB	Maps shift registers to MLAB resources.
All supported families	registers	Maps seqShift or altshift_tap register components to registers.

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.

Technology	Value	Implements ...
Microchip		
PolarFire	URAM	<ul style="list-style-type: none"> • Infers seqShift register components as RAM64X12.

Technology	Value	Implements ...
Xilinx		
All supported families	registers	Maps seqShift register components to registers.
	select_srl	Maps seqShift components using 16-bit shift register lookup table primitives (SRL16) with flip-flops inserted ahead of the output buffers for improved timing.
	noextractff_srl	Maps seqShift components using 16-bit shift register lookup table primitives (SRL16) without output flip-flops.

If you do not specify `syn_srlstyle`, the default behavior depends on the timing at the SRL output. If the output has very positive slack, the tool uses `syn_srlstyle=noextractff_srl`. In all other cases, the software implements `syn_srlstyle=select_srl` behavior. For more information, see [Inferring Shift Registers, on page 537](#) in the *User Guide*.

Description

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

The `syn_srlstyle` attribute can be set globally, either on a module or a register instance. The global attribute can be overridden by the attribute set on the module or instances.

syn_srlstyle Syntax

SCOPE	define_attribute {object} syn_srlstyle {register logic_ram altshift_tap groupName URAM block_ram MLAB distributed select_srl noextractff_srl} define_global_attribute syn_srlstyle {register logic_ram altshift_tap groupName URAM block_ram MLAB distributed select_srl noextractff_srl}}	SCOPE Example
Verilog	object /* synthesis syn_srlstyle = "register logic_ram altshift_tap groupName URAM block_ram MLAB distributed select_srl noextractff_srl" */;	See Vendor-specific Verilog Examples
VHDL	attribute syn_srlstyle: string; attribute syn_srlstyle of object : signal is "register logic_ram altshift_tap groupName URAM block_ram MLAB distributed select_srl noextractff_srl";	See Vendor-specific VHDL Examples

SCOPE Example

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

Here is a Tcl command that applies for Achronix devices.

```
define_attribute {i:regBank[13:0]} {syn_srlstyle} {logic_ram}
```

Here is a Tcl command that applies for Intel FPGA devices.

```
define_attribute {object} syn_srlstyle
{registers|block_ram|MLAB|altshift_tap,groupName}
```

Here is a Tcl command that applies for Xilinx devices.

```
define_global_attribute syn_srlstyle
{registers|select_srl|noextractff_srl}
```

This Tcl command applies to all devices:

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

HDL Example

In the HDL file, you must apply the `syn_srlstyle` attribute on the final stage of the shift register. In the following example, apply the `syn_srlstyle` attribute on register `pll_status_ck245_s`. The constraint is not honored if it is placed on other registers in the 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_srlstyle : string;
attribute syn_srlstyle of pll_status_ck245_s : signal is
"registers";
end test;

architecture behave of test is
signal pll_status_ck245_r : std_logic;
signal pll_status_ck245_r1 : std_logic;
begin
resynchro_ck245_reg: process(lbdr_clk)
BEGIN
if_clk: IF lbdr_clk'EVENT AND lbdr_clk = '1' THEN
    pll_status_ck245_r <= pll_status;
    pll_status_ck245_r1 <= pll_status_ck245_r;
    pll_status_ck245_s <= pll_status_ck245_r1;
END IF if_clk;
END PROCESS resynchro_ck245_reg;
end behave;

```

Verilog Example (Achronix)

For the Verilog RTL code below, synthesis implements the shift register using LRAM640.

```

module p_seqshift (clk, din, dout)
    /* synthesis syn_srlstyle = "logic_ram" */;
parameter SRL_WIDTH = 9;
parameter SRL_DEPTH = 4;
input clk;
input [SRL_WIDTH-1:0] din;
output [SRL_WIDTH-1:0] dout;
reg [SRL_WIDTH-1:0] regBank[SRL_DEPTH-1:0];

```

```
integer i;
always @ (posedge clk) begin
    for (i=SRL_DEPTH-1; i>0; i=i-1) begin
        regBank[i] <= regBank[i-1];
    end
    regBank[0] <= din;
end
assign dout = regBank[SRL_DEPTH-1];
endmodule
```

VHDL Example (Achronix)

In this example, synthesis implements the shift register using registers.

```
library ieee;
use ieee.std_logic_1164.all;
entity p_seqshift is
    generic (SRL_WIDTH : natural := 9;
             SRL_DEPTH : natural := 8);
    port(clk : in std_logic;
         din : in std_logic_vector((SRL_WIDTH-1) downto 0);
         dout : out std_logic_vector((SRL_WIDTH-1) downto 0));
end p_seqshift;
architecture archi of p_seqshift is
type shift is array ((SRL_DEPTH-1) downto 0) of
std_logic_vector((SRL_WIDTH-1) downto 0);
    signal tmp: shift;
attribute syn_srlstyle : string;
attribute syn_srlstyle of tmp : signal is "registers";
begin
    process (clk)
    begin
        if rising_edge(clk) then
            for i in (SRL_DEPTH-1) downto 1 loop
                tmp(i) <= tmp(i-1);
            end loop;
            tmp(0) <= din;
        end if;
    end process;
    dout <= tmp(SRL_DEPTH-1);
end archi;
```

Verilog Syntax and Example (Intel FPGA)

```
object /* synthesis syn_srlstyle =
    "registers |altshift_tap | block_ram | MLAB" */;
```

In the above syntax, *object* is a register declaration.

The following example implements a seqShift for 16x256 bits wide and serial in and serial out register using *syn_srlstyle* set to MLAB.

```
// shift left register with 16X256 bits width and serial in and
// serial out
module test(clock, arst, sr_en, shiftin, shiftout);

parameter sh_len=16;
parameter sh_width=256;
parameter ARESET_VALUE = {(sh_width){1'b0}};

input clock,arst,sr_en;
input [sh_width-1:0] shiftin;
output [sh_width-1:0] shiftout;
integer i;

reg [sh_width-1:0] sreg [sh_len-1:0] /* synthesis
syn_srlstyle="MLAB" */;

always @ (posedge clock or posedge arst)
begin /* synthesis loop_limit 17000 */
    if(arst)
        begin
            for(i = 0;i <= sh_len-1;i = i+1)
                sreg[i] <= ARESET_VALUE;
        end
    else
        begin /* synthesis loop_limit 17000 */
            if(sr_en)
                begin
                    sreg[0] <= shiftin;
                    for(i=sh_len-1;i>0;i=i-1)
                        sreg[i] <= sreg[i-1];
                end
        end
    end
end

assign shiftout = sreg[sh_len-1];
endmodule
```

Resource report in the srr file contains:

Memory ALUTs: 256

Effect of Using syn_srlstyle in Microchip Designs

PolarFire Technologies

Microchip devices support URAM inferencing with sequential shift registers. By default, seqshift is implemented using registers. You can override this default behavior using the uram option of the syn_srlstyle attribute.

The attribute can be applied on the top-level module or seqshift instances in the RTL. If the attribute is applied

- On the top-level module, then the tool infers URAM for the seqshift in the design using the threshold values:
Depth >= 8 and Depth * Width > 84
- On the seqshift instance, then the tool infers URAM regardless of the threshold values.

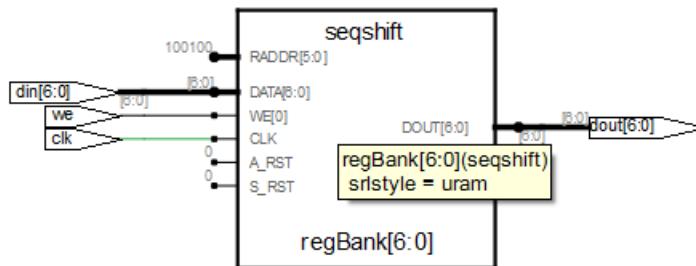
For this example, the software infers a seqshift primitive.

```
module p_seqshift(clk, we, din, dout);
parameter SRL_WIDTH = 7;
parameter SRL_DEPTH = 37;

input clk, we;
input [SRL_WIDTH-1:0] din;
output [SRL_WIDTH-1:0] dout;
reg [SRL_WIDTH-1:0]
    regBank[SRL_DEPTH-1:0]/*synthesis syn_srlstyle = "uram"*/;
integer i;

always @ (posedge clk) begin
    if (we) begin
        for (i=SRL_DEPTH-1; i>0; i=i-1) begin
            regBank[i] <= regBank[i-1];
        end
        regBank[0] <= din;
    end
end

assign dout = regBank[SRL_DEPTH-1];
endmodule
```



Verilog Syntax and Example (Xilinx)

```
object /* synthesis syn_srlstyle =
"select_srl | registers | noextractff_srl" */;
```

In the above syntax, *object* is a register declaration.

This example implements seqShift components as SRL16 without output flip-flops.

```
module test_srl(clk, enable, dataIn, result, addr);
    input clk, enable;
    input [3:0] dataIn;
    input [3:0] addr;
    output [3:0] result;
    reg [3:0] regBank[15:0]
        /* synthesis syn_srlstyle="noextractff_srl" */;
    integer i;

    always @ (posedge clk) begin
        if (enable == 1) begin
            for (i=15; i>0; i=i-1) begin
                regBank[i] <= regBank[i-1];
            end
            regBank[0] <= dataIn;
        end
    end
    assign result = regBank[addr];
endmodule
```

VHDL Syntax and Example (Xilinx)

```
attribute syn_srlstyle of object : signal is
    "select_srl | registers | noextractff_srl";
```

In the above syntax, *object* is a register. See [VHDL Attribute and Directive Syntax, on page 401](#) for different ways to specify VHDL attributes and directives.

The example below implements seqShift components as SRL16 primitives without inserting timing improvement flip-flops between the SRL outputs and the output buffers.

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

entity d_p is
    port (clk : in std_logic;
          data_out : out std_logic_vector(127 downto 0));
end d_p;

architecture rtl of d_p is
type dataAryType is array(3 downto 0) of
    std_logic_vector(127 downto 0);
signal h_data_pip_i : dataAryType;
attribute syn_srlstyle : string;
attribute syn_srlstyle of h_data_pip_i : signal
is "noextractff_srl";
begin
    process (Clk)
    begin
        if (Clk'Event And Clk = '1') then
            h_data_pip_i <= (h_data_pip_i(2 DOWNTO 0)) &
                h_data_pip_i(3);
        end if;
    end process;
    data_out <= h_data_pip_i(0);
end rtl;
```

syn_state_machine

Directive

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

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

syn_state_machine Values

Value	Description
0 false	Does not extract state machines automatically.
1 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

CDC File	CDC Example for a Verilog Design define_directive {n:libName.moduleName 1netName} {syn_state_machine p} {0 1} CDC Example for a VHDL Design define_directive {n:libName.moduleName.[architectureName22 netName]} {syn_state_machine} {0 1}	CDC Example
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

1. A pipe (|) separator character is required with no spaces between the net name and the module name
2. Optionally, specifies the name of the library for VHDL designs. Use for single entity or multiple architecture-based designs.

CDC Example

```
define_directive {n:work.FSM1|state} {syn_state_machine} {1}
```

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;

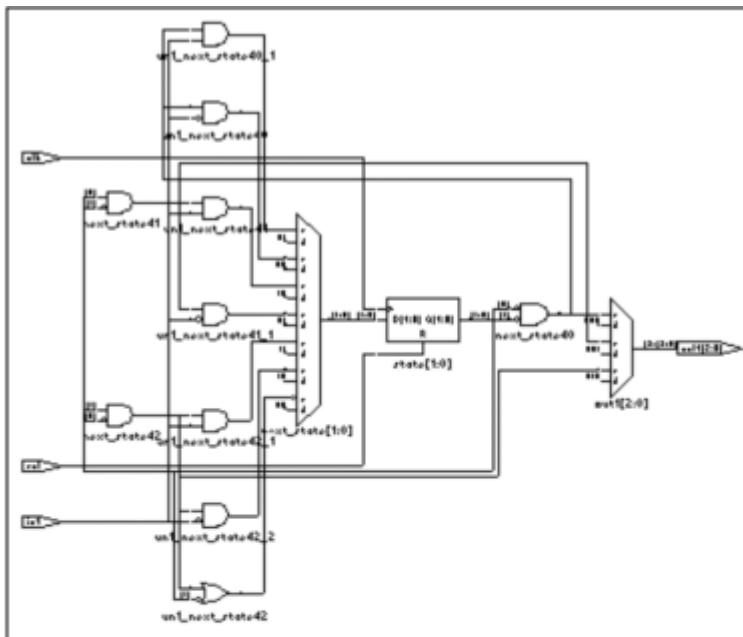
process (state, in1) begin
    case state is
        when s0 =>
            out1 <= "000";
            if in1 = '1' then next_state <= s1;
            else next_state <= s0;
        end if;
        when s1 =>
            out1 <= "001";
            if in1 = '1' then next_state <= s2;
            else 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 case;  
end process;  
end behave;
```

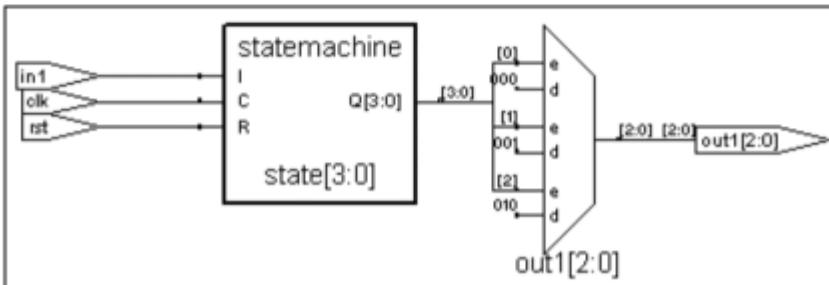
This is the VHDL source code used for the example in the following figure.

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.



`syn_state_machine = 0`



`syn_state_machine = 1`

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

- [syn_encoding, on page 259](#) for information on overriding default encoding styles for state machines.
- For VHDL designs, [syn_encoding, on page 259](#) 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 149](#) 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:

v:	Constraint file syntax that indicates the directive is attached to the view.
blackboxModule	The symbol name of the black-box.
n	A numerical suffix that lets you specify different clock to output timing delays for multiple signals/bundles.

! 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>bundle</i>	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
<i>value</i>	The values are in ns. 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 651](#) 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_tpdl="b->tout=8.0" syn_tsul="a->myclk=2.0" */;
    input myclk;
    input a, b;
    output tout;
endmodule

//Top Level
module top (input clk, input a, b, output fout);
    test U1 (clk, a, b, fout);
endmodule
```

VHDL Example

In VHDL, there are ten predefined instances of each of these directives in the `synplify` library: `syn_tco1`, `syn_tco2`, `syn_tco3`, ... `syn_tco10`. If you are entering the timing directives in the source code and you require more than 10 different timing delay values for any one of the directives, declare the additional directives with an integer greater than 10. For example:

```
attribute syn_tco11 : string;
attribute syn_tco12 : string;
```

See [syn_tco<n> Syntax, on page 651](#) for other syntax explanations.

See [VHDL Attribute and Directive Syntax, on page 401](#) 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_tco1 : string;
attribute syn_tco1 of test : component is
"clk->tout = 1.0";
attribute syn_tpdl : string;
attribute syn_tpdl of test : component is
"b->tout= 2.0";
attribute syn_tsul : string;
attribute syn_tsul of test : component is
"a-> myclk = 1.2";
begin
U1 : test port map(fout, a, b, clk);
end;
```

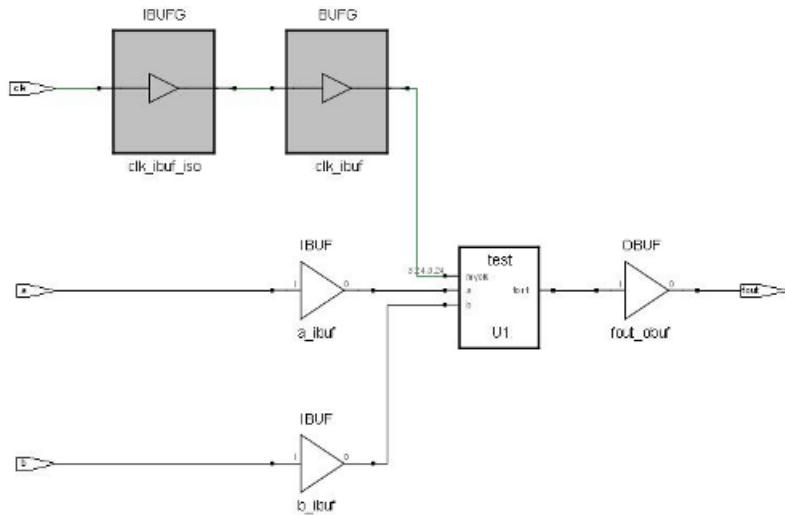
Verilog-Style Syntax in VHDL for Black Box Timing

In addition to the syntax used in the code above, you can also use the following Verilog-style syntax to specify black-box timing constraints:

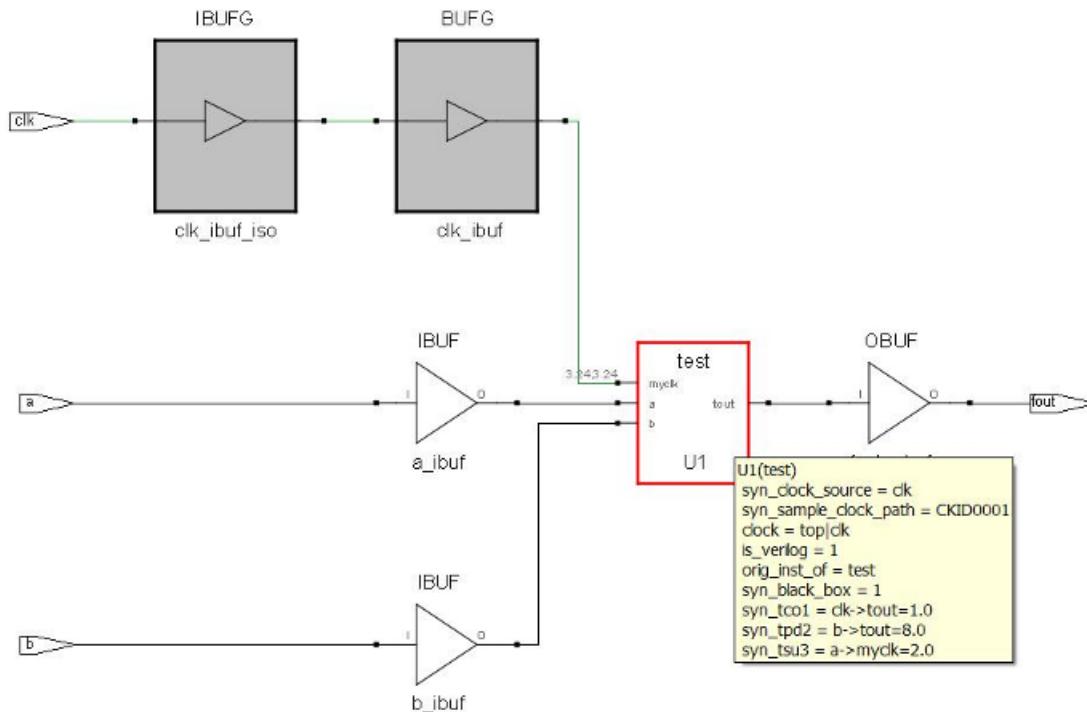
```
attribute syn_tco1 of inputfifo_coregen : component is
"rd_clk->dout[48:0]=3.0";
```

Effect of using syn_tco

This figure shows the 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 149](#) for a list of the associated directives.

syn_tpd<n> Syntax

Verilog `object /* syn_tpdn = "bundle -> bundle = value" */;`

VHDL `attribute syn_tpdn 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_tpdn {bundle->bundle=value}`

For details about the syntax, see the following table:

<code>v:</code>	Constraint file syntax that indicates the directive is attached to the view.
<code>blackboxModule</code>	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>	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. "<i>bundle->bundle=value</i>" The values are in ns.
<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 657](#) 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 657 for an explanation of the syntax.

See [VHDL Attribute and Directive Syntax](#), on page 401 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 654](#) 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;
```

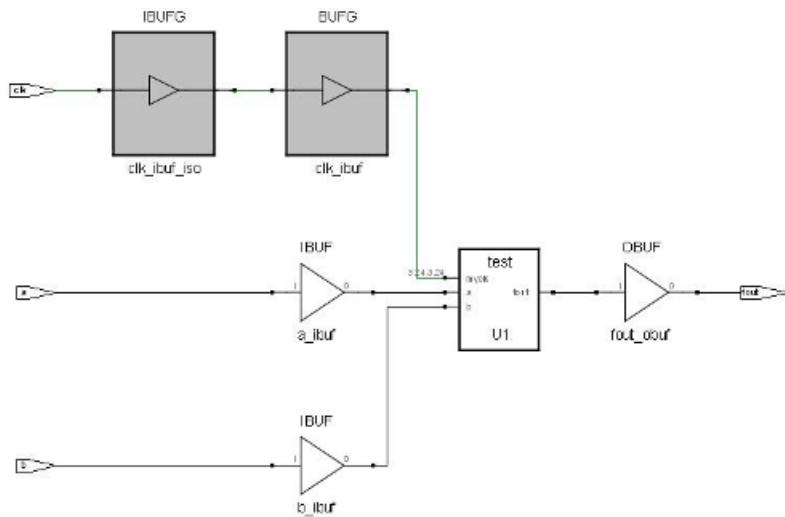
```
-- TOP Level--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity top is
generic (size: integer := 8);
port (fout : out std_logic_vector(size- 1 downto 0);
      a : in std_logic_vector (size- 1 downto 0);
      b : in std_logic_vector (size- 1 downto 0);
      clk : 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_tco1 : string;
attribute syn_tco1 of test : component is
  "clk->tout = 1.0";
attribute syn_tpdl : string;
attribute syn_tpdl of test : component is
  "b->tout= 2.0";
attribute syn_tsul : string;
attribute syn_tsul of test : component is
  "a-> myclk = 1.2";
begin
U1 : test port map(fout, a, b, clk);
end;
```

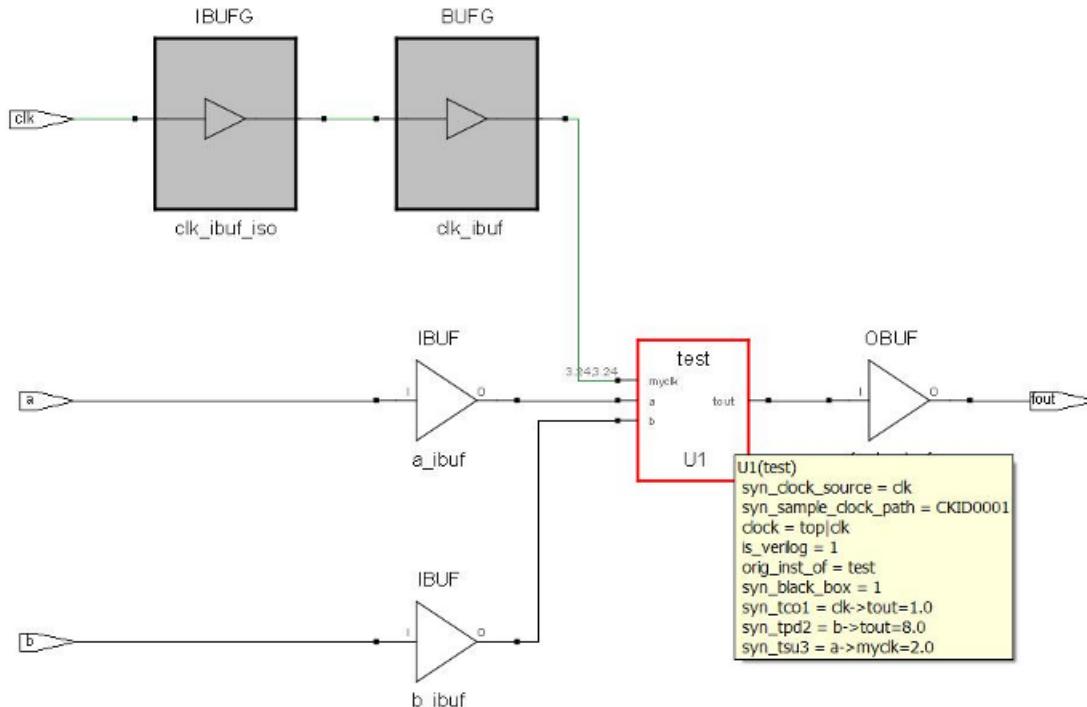
Effect of using syn_tpd

This figure shows the 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 /* synthesis syn_tristate = 1 */;</i>
VHDL	attribute syn_tristate : boolean; attribute syn_tristate of tout: signal is true;

Verilog Example

```
module test(myclk, a, b, tout) /* synthesis syn_black_box */;
  input myclk;
  input a, b;
  output tout/* synthesis syn_tristate = 1 */;
endmodule

//Top Level
```

```
module top(input [1:0]en, input clk, input a, b, output reg fout);
  wire tmp;
  assign tmp = en[0] ? (a & b) : 1'bz;
  assign tmp = en[1] ? (a | b) : 1'bz;
  always@(posedge clk)
  begin
    fout <= tmp;
  end
  test U1 (clk, a, b, tmp);
endmodule
```

VHDL Example

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

entity test is
  port (tout : out std_logic;
        a : in std_logic;
        b : in std_logic;
        myclk : in std_logic);

attribute syn_tristate : boolean;
attribute syn_tristate of tout: signal is true;
end;

architecture rtl of test is
attribute syn_black_box : boolean;
attribute syn_black_box of rtl: architecture is true;
begin
end;

-- TOP Level--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity top is
  port (fout : out std_logic;
        a : in std_logic;
        b : in std_logic;
        en: in std_logic_vector(1 downto 0);
        clk : in std_logic
      );
end;
```

```

architecture rtl of top is
signal tmp : std_logic;
component test
port (tout : out std_logic;
      a : in std_logic;
      b : in std_logic;
      myclk : in std_logic
    );
end component;

begin
tmp <= (a and b)when en(0) = '1' else 'Z';
tmp <= (a or b) when en(1) = '1' else 'Z';
process (clk)
begin
  if (clk = '1' and clk'event) then
    fout <= tmp;
  end if;
end process;

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

```


syn_tristatetomux

Attribute

Analyzes each net in the design that is driven by tristate drivers and converts the tristate drivers to multiplexers.

Vendor	Technology
Xilinx	Virtex, Virtex-E Spartan II, Spartan-IIIE

syn_tristatetomux Values

Value	Description	Global
<i>integer</i>	Converts only those tristate-driven nets that meet the threshold criteria you specified. However, the software can determine how it converts tristates depending on the specific technology you are synthesizing. For details, see syn_tristatetomux Conversion Table , on page 668 .	Yes
1 (Default)	Converts only those tristate-driven nets that meet this threshold criteria of 1.	

syn_tristatetomux Conversion Table

The conversion of tristates to muxes varies with the technology, as shown in the following table:

Technology¹	syn_tristatetomux Behavior
Virtex-4 and newer devices	All tristates are converted to logic, regardless of the attribute value.
Spartan-3/3E	
Spartan-6	
Virtex-II	If you do not specify the attribute, the software converts all tristates to logic.
Virtex-II Pro	If you specify the attribute, the software only converts those tristate-driven nets that meet the threshold criteria. If the threshold is set to 0, the software does not convert any tristates.
Virtex	
Virtex-E	
Spartan-II	If you do not specify the attribute, the software only converts tristates on the critical path to logic. It does not change tristates on non-critical paths.
Spartan-IIIE	If you specify the attribute, the software only converts those tristate-driven nets that meet the threshold criteria. If the threshold is set to 0, the software does not convert any tristates.

1. For Xilinx designs, all the internal tristates along a critical path are automatically converted to muxes. This is done to avoid probable routing problems and to improve performance. You cannot turn off the conversion, but you can use this attribute to relax the constraints.

Description

This attribute analyzes each net in the design that is driven by tristate drivers and converts the tristate drivers to multiplexers. The software converts the tristates to muxes when the number of tristate drivers on the net is less than the value of the integer argument. All drivers to be converted to multiplexers must be tristate drivers, otherwise the conversion fails and the software issues a warning.

The synthesis process cannot precisely emulate a tristate condition with a mux. Therefore, during tristate-to-mux conversion, the default behavior of the output when all the enables are disabled is zero. The user has no control over this.

syn_tristatetomux Syntax

The following table summarizes the syntax in FDC file:

FDC	<code>define_attribute {object} syn_tristatetomux {integer}</code>	SCOPE Example
Verilog	<i>Virtex-2 and older</i> <code>object /* synthesis syn_tristatetomux = integer */;</code>	Verilog Example
VHDL	<i>Virtex-2 and older</i> <code>attribute syn_tristatetomux: integer;</code> <code>attribute syn_tristatetomux of object : objectType is integer;</code>	VHDL Example

SCOPE Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	view	v:work.tristate2	syn_tristatetomux	4	integer	Threshold for converting tristates to mux
2							

The Tcl equivalent is:

```
define_attribute {v:work.tristate2} syn_tristatetomux {4}
```

Verilog Example

object / synthesis syn_tristatetomux = integer */;*

Virtex-2 and older

For example:

```
module tristate2 (input0, input1, input2, input3, enable, qout)
    /* synthesis syn_tristatetomux = 4 */;
    input [7:0] input0, input1, input2, input3;
    input [3:0] enable;
    output [7:0] qout;
    wire [7:0] temp;
    assign temp = (enable[0]) ? input0 : 8'hZZ;
    assign temp = (enable[1]) ? input1 : 8'hZZ;
    assign temp = (enable[2]) ? input2 : 8'hZZ;
    assign temp = (enable[3]) ? input3 : 8'hZZ;
    assign qout = temp & input0;
endmodule
```

VHDL Example

attribute syn_tristatetomux of object : objectType is integer;

Virtex-2 and older

Here is a VHDL example of multiple tristate drivers on a bus.

```

library ieee;
library synplify;

use ieee.std_logic_1164.all;
entity tristate2 is
    port (input3,
          input2,
          input1,
          input0: in std_logic_vector (7 downto 0);
          enable : in std_logic_vector (3 downto 0);
          qout : out std_logic_vector (7 downto 0));
end tristate2;

architecture multiple_drivers of tristate2 is
attribute syn_tristatetomux : integer;
attribute syn_tristatetomux of multiple_drivers :
    architecture is 4;
signal temp : std_logic_vector (7 downto 0);

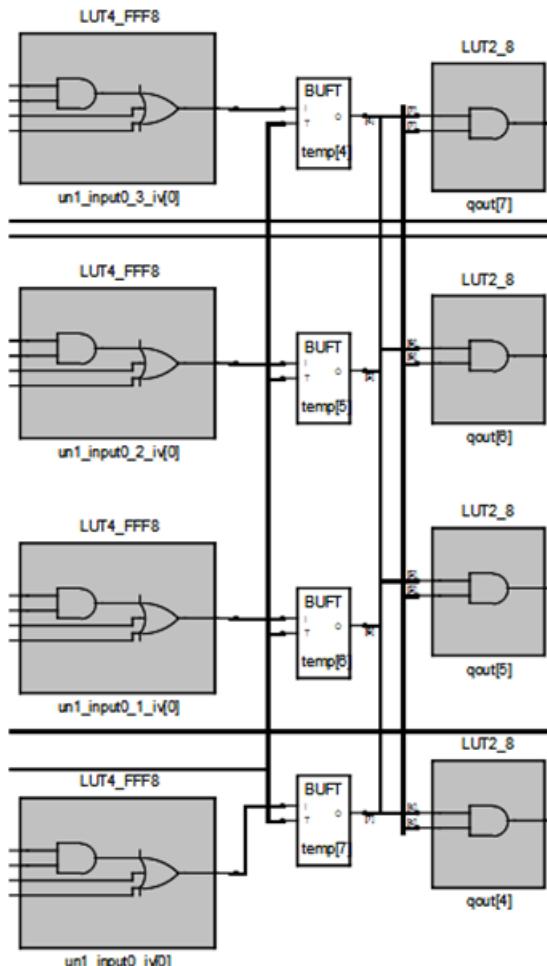
begin
    temp <= input3 when enable(3) = '1'
        else "ZZZZZZZZ";
    temp <= input2 when enable(2) = '1'
        else "ZZZZZZZZ";
    temp <= input1 when enable(1) = '1'
        else "ZZZZZZZZ";
    temp <= input0 when enable(0) = '1'
        else "ZZZZZZZZ";
    qout <= temp and input0;
end multiple_drivers;

```

Effect of Using syn_tristatetomux

The value of syn_tristatetomux is 0, which is synthesized with the Spartan2e device. The table below shows the corresponding values in the FDC file:

FDC	define attribute {v:work.tristate2} syn_tristatetomux {0}
Verilog	<i>Virtex-2 and older</i> <pre>module tristate2 (input0, input1, input2, input3, enable, qout) /* synthesis syn_tristatetomux = 0 */;</pre>
VHDL	<i>Virtex-2 and older</i> <pre>attribute syn_tristatetomux : integer; attribute syn_tristatetomux of multiple_drivers : architecture is 0;</pre>



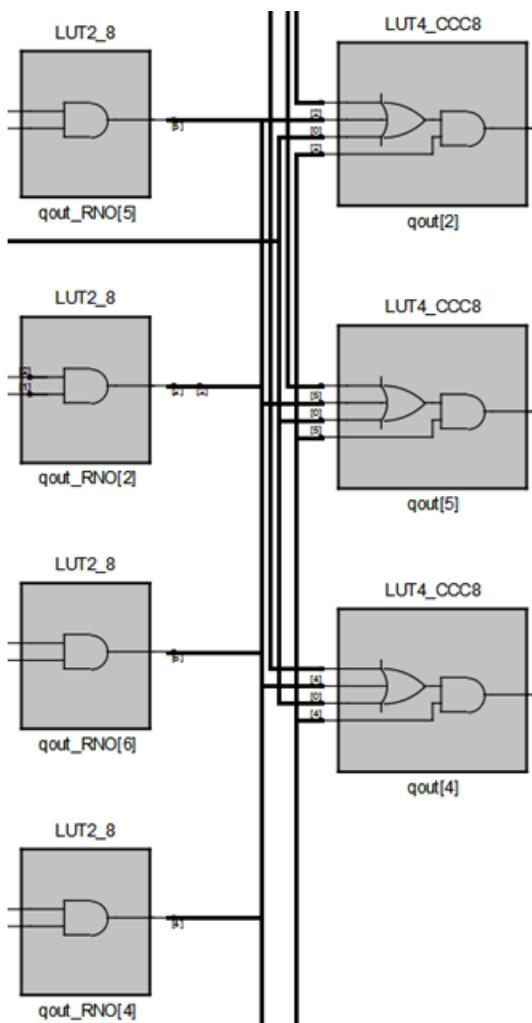
:

The value of `syn_tristatetomux` is now set to 4, which is re-synthesized with the Spartan2e device. The table below shows the corresponding values in the FDC file:

FDC	define attribute {v:work.tristate2} <code>syn_tristatetomux</code> {4}
-----	--

Verilog	<i>Virtex-2 and older</i> module tristate2 (input0, input1, input2, input3, enable, qout) /* synthesis <code>syn_tristatetomux</code> = 4 */;
---------	---

VHDL	<i>Virtex-2 and older</i> attribute <code>syn_tristatetomux</code> : integer; attribute <code>syn_tristatetomux</code> of multiple_drivers l: architecture is 4;
------	--



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 149](#) 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:

v:	Constraint file syntax that indicates the directive is attached to the view.
blackboxModule	The symbol name of the black box.
n	A numerical suffix that lets you specify different clock to output timing delays for multiple signals/bundles.

bundle A collection of buses and scalar signals. The objects of a bundle must be separated by commas with no intervening spaces. A valid bundle is A,B,C, which lists three signals. The values are in ns. This is the syntax to define a bundle:

*bundle->[!]*clock*=*value**

! The optional exclamation mark indicates that the clock is active on its falling (negative) edge.

clock The name of the clock signal.

value 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 675](#).

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_tco1="clk->tout=1.0" syn_tpdl="b->tout=8.0"
syn_tsu1="a->myclk=2.0" */;
  input myclk;
  input a, b;
  output tout;
endmodule

//Top Level
module top (input clk, input a, b, output fout);
  test U1 (clk, a, b, fout);
endmodule
```

VHDL Examples

In VHDL, there are 10 predefined instances of each of these directives in the `synplify` library, for example: `syn_tsu1`, `syn_tsu2`, `syn_tsu3`, ... `syn_tsu10`. If you are entering the timing directives in the source code and you require more than 10 different timing delay values for any one of the directives, declare the additional directives with an integer greater than 10:

```
attribute syn_tsu11 : string;
attribute syn_tsu11 of bitreg : component is
    "di0,di1 -> clk = 2.0";
attribute syn_tsu12 : string;
attribute syn_tsu12 of bitreg : component is
    "di2,di3 -> clk = 1.8";
```

For other syntax explanations, see [syn_tsu<n> Syntax, on page 675](#).

See [VHDL Attribute and Directive Syntax, on page 401](#) 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 654](#).

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);
```

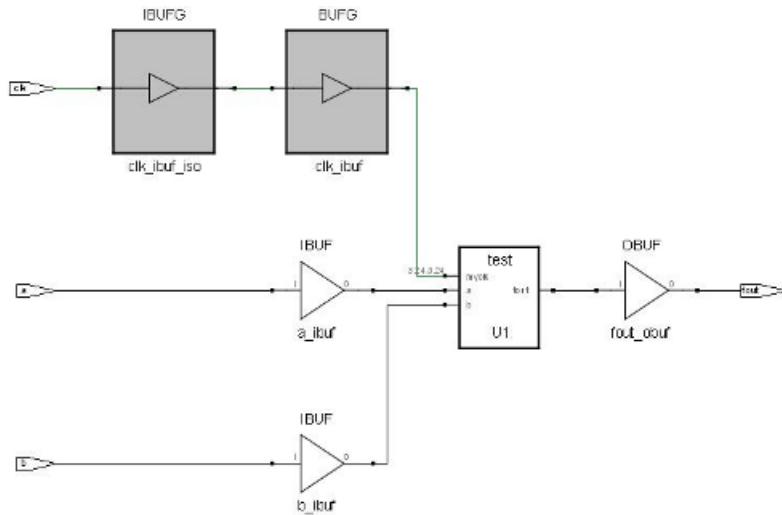
```
b :    in std_logic_vector (size- 1 downto 0);
clk : in std_logic
);
end;

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;

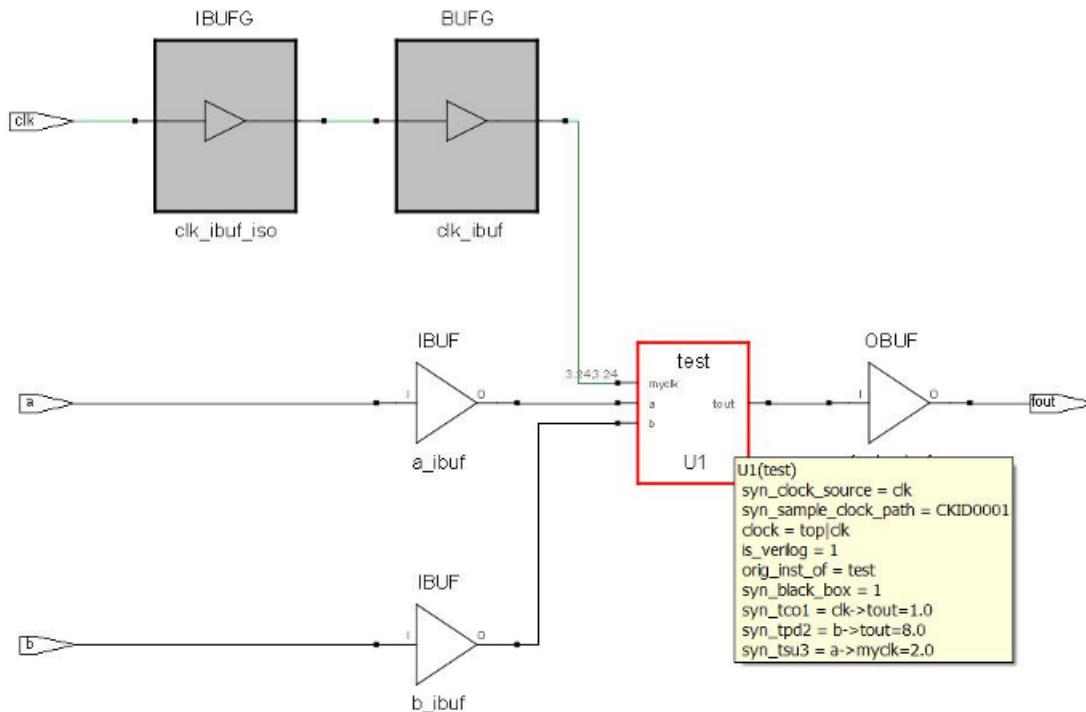
attribute syn_tco1 : string;
attribute syn_tco1 of test : component is
  "clk->tout = 1.0";
attribute syn_tpdl : string;
attribute syn_tpdl of test : component is
  "b->tout= 2.0";
attribute syn_tsul : string;
attribute syn_tsul of test : component is
  "a-> myclk = 1.2";
begin
U1 : test port map (fout, a, b, clk);
end;
```

Effect of using syn_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_unconnected_inputs

Attribute

You can specify that input pins of an instance be left floating (unassigned in the RTL).

Vendor	Technologies
Achronix	Speedster7t and newer families

syn_unconnected_inputs Values

Value	Description	Default	Global
<i>instanceName</i> and <i>stringValue</i>	Specifies input pins of an instance with undriven wires be left floating.	None	No

Description

You can specify that input pins of an instance be left floating (unassigned in the RTL).

If an instance has a floating input pin, the synthesis software does not tie it to GND in the output netlist. However when the input pin is driven by a wire that subsequently is unassigned, then the synthesis tool ties it to GND. Connecting floating input pins to GND causes routing problems in the place-and-route tool. Use the syn_unconnected_inputs attribute to ensure that input pins with undriven wires are left floating. This attribute can only be specified in the constraint file.

syn_unconnected_inputs Syntax

FDC	define_attribute {instanceName} syn_unconnected_inputs {string}	FDC Example
Verilog	object /*synthesis syn_unconnected_inputs={string}*;	Verilog Example
VHDL	attribute syn_unconnected_inputs: string; attribute syn_unconnected_inputs of {component architectureName}: object is {string};	VHDL Example

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	Instance	tdecode.decode_in130[0]	syn_unconnected_inputs	aluz	string	Leave a pin unconnected on an instance
2							

For example:

```
define_attribute {i:decode.decode_in130[0]}
    syn_unconnected_inputs {aluz}
```

Verilog Example

In this Verilog example, the syn_unconnected_inputs attribute is applied on the ipA input of the mac module.

Verilog Example: syn_unconnected_inputs (MAC Module)

```
//Example: Verilog syn_unconnected_inputs using MAC Module

module mac (
    ipA, ipB, ipC,
    op
) /* synthesis syn_unconnected_inputs = ipA */;

parameter WIDTH = 8;
```

```
input [WIDTH-1:0] ipA, ipB, ipC;  
output [WIDTH-1:0] op;  
  
assign op = ipA * ipB + ipC;  
  
endmodule
```

Verilog Example: syn_unconnected_inputs (Top-Level Module)

```
//Example -- Verilog syn_unconnected_inputs using Top-Level Module

module top (
    clk,
    rst,
    dinA, dinB, dinC,
    dinD, dinE, dinF,
    dout1, dout2
);

    input clk;
    input rst;

    input [7:0] dinA, dinB, dinC;
    input [7:0] dinD, dinE, dinF;

    output reg [7:0] dout1;
    output reg [7:0] dout2;
```

```
reg [7:0] dinA_reg, dinB_reg, dinC_reg;
reg [7:0] dinD_reg, dinE_reg, dinF_reg;

wire [7:0] dout1_reg;
wire [7:0] dout2_reg;

mac INS1 (
    .ipA(dinA_reg),
    .ipB(dinB_reg),
    .ipC(dinC_reg),
    .op(dout1_reg)
);

mac INS2 (
    .ipA(dinD_reg),
    .ipB(dinE_reg),
    .ipC(dinF_reg),
    .op(dout2_reg)
);

always @ (posedge clk or posedge rst)
begin
if (rst)
begin
    dinA_reg <= 0;
    dinB_reg <= 0;
```

```

    dinC_reg <= 0;
    dinD_reg <= 0;
    dinE_reg <= 0;
    dinF_reg <= 0;
    dout1 <= 0;
    dout2 <= 0;
  end
else
begin
  dinA_reg <= dinA;
  dinB_reg <= dinB;
  dinC_reg <= dinC;
  dinD_reg <= dinD;
  dinE_reg <= dinE;
  dinF_reg <= dinF;
  dout1 <= dout1_reg;
  dout2 <= dout2_reg;
end
end
endmodule

```

VHDL Example

In this VHDL example, the `syn_unconnected_inputs` attribute is applied to the `dodatain` pin of the component `mac`.

VHDL Example: syn_unconnected_inputs (MAC Module)

--Example -- VHDL syn unconnected inputs using MAC Module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mac is
    Port (
        datain1, datain2, datain3 : in STD_LOGIC_VECTOR (7 downto 0);
        dataout : out STD_LOGIC_VECTOR (15 downto 0)
    );
end mac;

architecture Behavioral of mac is

begin
    dataout <= (datain1 * datain2) + datain3;

end Behavioral;
```

VHDL Example: syn_unconnected_inputs (Top-Level Module)

```
--Example -- VHDL syn_unconnected_inputs (Top-Level Module)

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top is
    Port (
        clk : in std_logic;
        rst : in std_logic;
        datain1, datain2, datain3 : in STD_LOGIC_VECTOR (7 downto 0);
        datain4, datain5, datain6 : in STD_LOGIC_VECTOR (7 downto 0);
        dataout1 : out STD_LOGIC_VECTOR (15 downto 0);
        dataout2 : out STD_LOGIC_VECTOR (15 downto 0)
    );
end top;

architecture Behavioral of top is

component mac is
    Port (
        datain1, datain2, datain3 : in STD_LOGIC_VECTOR (7 downto 0);
        dataout : out STD_LOGIC_VECTOR (15 downto 0)
    );
end component;

signal datain1_reg, datain2_reg, datain3_reg : STD_LOGIC_VECTOR (7
downto 0);
signal datain4_reg, datain5_reg, datain6_reg : STD_LOGIC_VECTOR (7
downto 0);

signal dataout1_reg : STD_LOGIC_VECTOR (15 downto 0);
```

```
:  
  
signal dataout2_reg : STD_LOGIC_VECTOR (15 downto 0);  
  
attribute syn_unconnected_inputs : string ;  
attribute syn_unconnected_inputs of mac : component is "datain1";  
  
begin  
  
INS1: mac port map (datain1 => datain1_reg, datain2 => datain2_reg,  
datain3 => datain3_reg, dataout => dataout1_reg);  
INS2: mac port map (datain1 => datain4_reg, datain2 => datain5_reg,  
datain3 => datain6_reg, dataout => dataout2_reg);  
  
process (clk, rst)  
begin  
if (clk='1' and clk'event) then  
if (rst='1') then  
    datain1_reg <= (others => '0');  
    datain2_reg <= (others => '0');  
    datain3_reg <= (others => '0');  
    datain4_reg <= (others => '0');  
    datain5_reg <= (others => '0');  
    datain6_reg <= (others => '0');  
    dataout1 <= (others => '0');  
    dataout2 <= (others => '0');  
else  
    datain1_reg <= datain1;  
    datain2_reg <= datain2;  
    datain3_reg <= datain3;
```

```

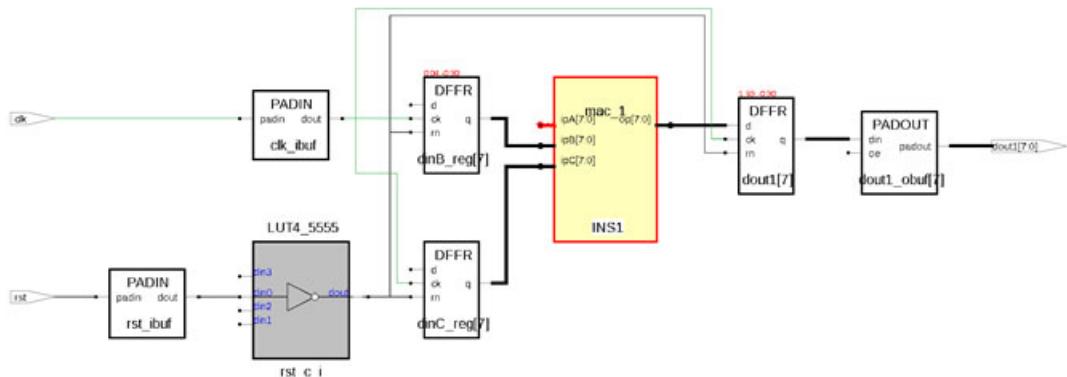
    datain4_reg <= datain4;
    datain5_reg <= datain5;
    datain6_reg <= datain6;
    dataout1 <= dataout1_reg;
    dataout2 <= dataout2_reg;
end if;
end if;
end process;

end Behavioral;

```

Effect of Using syn_unconnected_inputs

When the `syn_unconnected_inputs` attribute is applied on the input pin `ipA` of the `mac` module, the corresponding input is floating and not connected to GND as shown below.



syn_unique_inst_module

Directive

Renames the module for a particular instance, leaving other instances and the original module unchanged.

syn_unique_inst_module Value

Value	Description
<i>newModuleName</i>	Specifies a new name for the module of the instance

Description

The syn_unique_inst_module directive, which is applied through a compiler directives file (cdc), renames the module for a particular instance without affecting the other instances of the original module or the original module.

Use this directive in a hierarchical design flow where subprojects are created for instance-based export. This flow requires that the instance uniquely link to a particular module. You can achieve this by only renaming the module for the instance, leaving the other instances of the module as is, pointing to the original module.

syn_unique_inst_module Syntax Specification

CDC File define_directive {*instanceName*} syn_unique_inst_module
 {*newModuleName*}

[CDC Example](#)

CDC Example

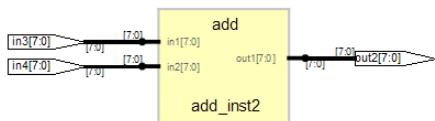
```
define_directive {instanceName}  
                  syn_unique_inst_module{newModuleName}
```

A design has multiple instantiations of a module. Preserve the hierarchy of a single instance, leaving the other instances of the module as is.

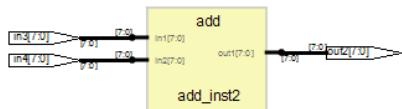
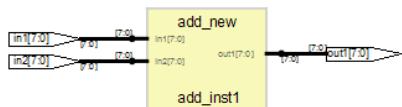
```
CDC: define_directive
{i:add_inst1}
{syn_unique_inst_module}
{add_new}
FDC: define_attribute
{v:work.add_new}
{syn_hier}
{hard}
Verilog:
module test (input clk, input din, output dout);
add add_inst1 (clk, A, B, dout) ;
endmodule
module add (input clk, input A, B, output logic D);
always@ (posedge clk)
begin
D<= A+B;
end
endmodule
```

Effect of Using syn_unique_inst_module

The following figure shows a design before syn_unique_inst_module is applied.



This figure shows the results after applying syn_unique_inst_module:



syn_upf_ret_port_type

UPF Directive

Identifies the port type for the sequential element.

Syntax

syn_upf_ret_port_type = "portType"

Where the port for the sequential element, *portType* can be specified as described in the following table:

Argument 1	Argument 2	Description
CLK		Clock
RESET	SYNC ASYNC	Synchronous or Asynchronous Reset
SET	SYNC ASYNC	Synchronous or Asynchronous Set
EN		Enable
SAVE ¹		Retention Save Control
RESTORE ¹		Retention Restore Control
D		Input Data
Q		Output Data

1. The custom retention model interface requires that you define ports for both single and dual retention, but use:
 - Either save or restore for single-signal retention
 - Both save and restore for dual-signal retention

:

Example

```
input clk /* synthesis syn_upf_ret_port_type = "CLK" */;
input reset /* synthesis syn_upf_ret_port_type =
    "RESET, ASYNC" */;
input SAVE /* synthesis syn_upf_ret_port_type = "SAVE" */;
input RESTORE /* synthesis syn_upf_ret_port_type = "RESTORE" */;
```

syn_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, 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	<code>define_attribute {instanceName} syn_use_carry_chain {integerValue} define_global_attribute syn_use_carry_chain {integerValue}</code>	FDC Example
Verilog	<code>object /* synthesis syn_use_carry_chain = integerValue */;</code>	Verilog Examples
VHDL	<code>attribute syn_use_carry_chain : string; attribute syn_use_carry_chain of object : objectType is integerValue;</code>	VHDL Examples

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:un1_state[1:0]	syn_use_carry_chain	10	integer	inference of carry chains

```
define_attribute {i:un1_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[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  
  
always@(c or d)  
begin  
if(c >= d)  
begin  
gr2 = 1'b1;  
ngr2 = 1'b0;  
end  
else  
begin  
gr2 = 1'b0;  
ngr2 = 1'b1;  
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)

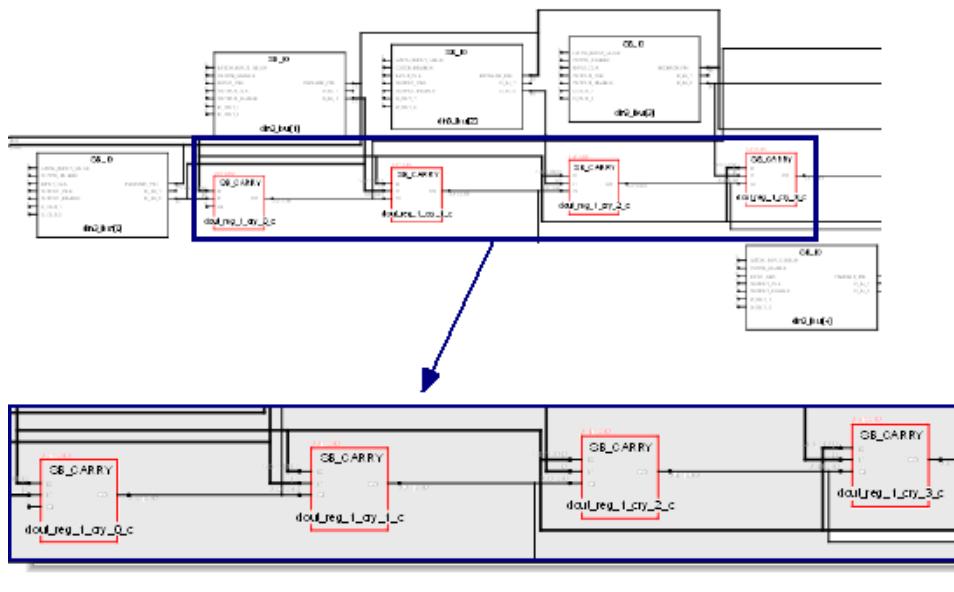
```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;
```

```
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 in the Technology view below:



syn_useenables

Attribute

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

Vendor	Technology
Achronix	Speedster family
Intel FPGA	Agilex, Stratix 10, Stratix V, Cyclone V, Arria V, and newer families
Lattice	EC, SC, XP, ORCA, isp, and newer families
Microchip	30, 40, and 50 series, ACT, Axcelerator, ex, SmartFusion2, Igloo2, RTG4, PolarFire
Xilinx	Virtex-7 and new families Spartan-3 and Spartan-6 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	<code>define attribute {register signal} syn_useenables {0 1}</code>	SCOPE Example
Verilog	<code>object /* synthesis syn_useenables = "0 1" */;</code>	Verilog Example
VHDL	<code>attribute syn_useenables of object : objectType is "true false";</code>	VHDL Example

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

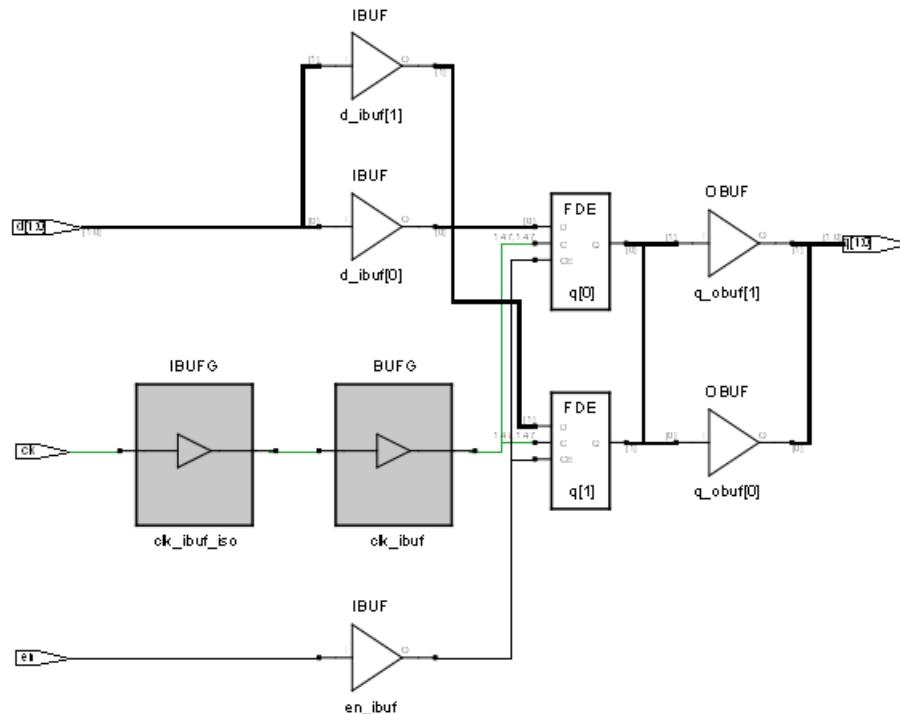
```
library ieee;
use ieee.std_logic_1164.all;

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;

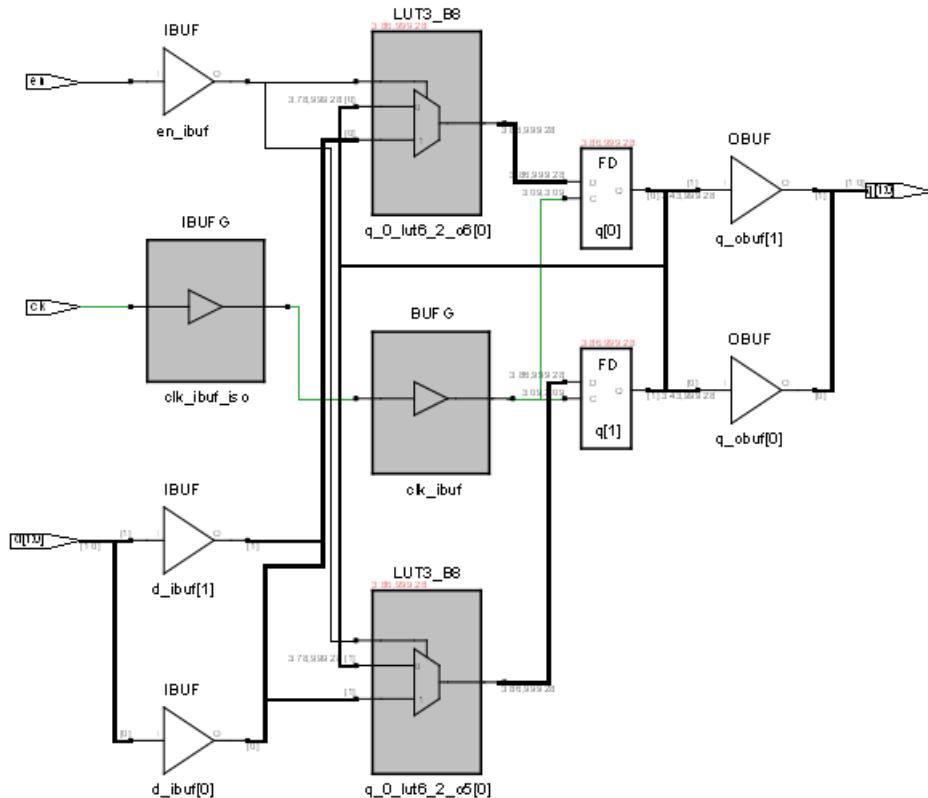
architecture syn_ue of syn_useenables is
begin
    process (clk) begin
        if (clk = '1' and clk'event) then
            if (en='1') then
                q <= d;
            end if;
        end if;
    end process;
end architecture;
```

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

The tool supports forward annotation of the property `syn_useioff=0` or `1` in the netlist (.vm file).

Vendor	Technologies
Intel FPGA	Agilex, Stratix 10, Stratix V, Cyclone V, Arria V, and newer families
Lattice	iCE40
Xilinx	Virtex-7 and New families Spartan-3 and Spartan-6 families

syn_useioff Values

Value	Description
<code>1/true</code>	Packs the registers into I/O pad cells.
<code>0/false</code>	Do not pack the registers into I/O pad cells.
<code>force</code>	<p><i>Xilinx only</i></p> <p>Packs the registers into I/O pad cells.</p> <p>Xilinx place and route requires that registers be packed into I/O pad cells with this option. The results of packing registers are not different if you use force or true. However, when a register is ineligible to be packed into an IOB, specifying <code>IOB=force</code> generates an error message in the Vivado tool. If <code>IOB=true</code> is used, a warning message is generated instead. For more information about how IOB constraints control register packing, see the Xilinx documentation.</p>

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
<code>syn_useioff = force</code>	Packs register into the I/O pad cell regardless of the port or global specification

2. Ports

<code>syn_useioff = 1/true</code>	Packs registers into the I/O pad cell regardless of global specification
<code>syn_useioff = 0/false</code>	Does not pack registers into I/O pad cell regardless of global specification
<code>syn_useioff = force</code>	Packs registers into the I/O pad cell regardless of global specification

3. Global (lowest priority)

<code>syn_useioff = 1/true</code>	Packs registers into the I/O pad cells
<code>syn_useioff = 0/false</code>	Does not pack registers into I/O pad cells
<code>syn_useioff = force</code>	Packs registers into the I/O pad cells

Vendor Considerations

Intel FPGA	Arria and Stratix families: When this attribute is enabled, registers are not packed into Multiply/Accumulate (MAC) blocks.
Xilinx	When specified, IOB properties are added to the output register in the Technology view to reflect the state of the <code>syn_useioff</code> attribute.

syn_useioff Syntax Specification

The following table summarizes the syntax in different files. See [Object Considerations, on page 718 Examples of syn_useioff on Different Ports](#), on [page 722](#) for descriptions of the ports where you can set the attribute.

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

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")))
      )
    )
  )
)
```

Setting the `syn_useioff` attribute to force ensures that the software attempts to pack registers into an IOB. When you set the attribute on a top-level register or port, the synthesis software writes out the `syn_useioff` attribute to the output netlist file as an `IOB=FORCE` statement to be used by Xilinx place and route.

Example of an edif netlist when `syn_useioff` is set to force:

```
(library work
  (edifLevel 0)
  (technology (numberDefinition))
  (cell VLOG_INR_OUTR_FD_1 (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port io (direction INOUT))
        (port clk (direction INPUT))
        (port tst (direction INPUT)))
      )
      (contents
        (instance clk_ibuf (viewRef PRIM (cellRef BUFG
          (libraryRef VIRTEX)))
        )
        (instance clk_ibuf_iso (viewRef PRIM (cellRef IBUFG
          (libraryRef VIRTEX)))
        )
        (instance io_iobuf_RNO (viewRef PRIM (cellRef INV
          (libraryRef UNILIB)))
        )
        (instance inff (viewRef PRIM (cellRef FD_1 (libraryRef UNILIB)))
          (property IOB (string "FORCE")))
      )
    )
  )
)
```

Also, the IOB FORCE property is forward annotated in the XDC constraint file, specifically when a VM netlist is generated. For example:

```
1 # 1000 : define_clock [get_ports {clk}] -name {VHDL_INR_OUTR_FDCE|clk} -ref_rise
2
3 create_clock -name {VHDL_INR_OUTR_FDCE|clk} [get_ports {clk}] -period {1.000}
4
5
6 set_property IOB force [get_cells {inff_Z}]
7 set_property IOB force [get_cells {outff_Z}]
8
9 #Constraints which are not forward annotated in XDC and intentionally commented out
10
```



```
1 # 1000 : define_clock [get_ports {clk}] -name {VHDL_INR_OUTR_FDCE|clk} -ref_rise
2
3 create_clock -name {VHDL_INR_OUTR_FDCE|clk} [get_ports {clk}] -period {1.000}
4
5
6 set_property IOB force [get_cells {inff_Z}]
7 set_property IOB force [get_cells {outff_Z}]
8
9 #Constraints which are not forward annotated in XDC and intentionally commented out
10
```

syn_useioff and ace_useioff

Attribute

The tool supports forward annotation of the property syn_useioff=0 or 1 or ace_useioff = 0 or 1 in the netlist (.vm file).

Vendor	Technologies
Achronix	Speedster7t

Attribute Support

When you apply the attribute syn_useioff= 1|0 or ace_useioff= 1|0 on the registers, instances or ports, then the tool forward annotates the property on that instance in the netlist (.vm file). Additionally, the attribute ace_useioff= 1|0 can be applied on the nets to leverage the retiming on the corresponding register.

- If the attribute syn_useioff/ace_useioff is applied globally, then the property will be forward annotated to the top level.
- The syn_useioff attribute cannot be applied on the nets, use the ace_useioff attribute instead. The tool will issue a warning message incase of violation of this rule.
- The tool does not retime the registers when the syn_useioff or ace_useioff attributes are set to 1.
- The tool does not retime the registers driving or driven by the nets when the syn_useioff or ace_useioff attributes are set to 1.
- Retiming on the registers is allowed only if both the attributes are set to '0'.
- If the attributes syn_useioff or ace_useioff set to 1 is applied globally, then the tool will not disable retiming. Retiming prevents the tool to achieve the better performance /timing QOR benefits. Only registers which are connected directly or indirectly (separated by hierarchy) at top level port will be prevented from retiming.
- If the attribute is applied at the sub-module level, then the tool will ignore the property and issues a warning message.

Syntax Specification

The syn_useioff and ace_useioff attributes are applied in the RTL or constraint file (.fdc file).

Verilog syntax:

The following are the valid syntax for applying the attribute through Verilog RTL.

- Attribute applied on input /output registers:

output reg	<i>d</i> /*synthesis syn_useioff=1 0*/ OR /*ace_useioff=1 or 0*/;
------------	---

input reg	<i>din</i> /*synthesis syn_useioff=1 0*/ OR /*ace_useioff=1 or 0*/;
-----------	---

- Attribute applied on input/output port and net.

output	<i>d</i> /*synthesis syn_useioff=1 0*/ OR /*ace_useioff=1 or 0*/;
--------	---

input	<i>din</i> /*synthesis syn_useioff=1 0*/ OR /*ace_useioff=1 or 0*/;
-------	---

wire	<i>tmp</i> /*ace_useioff=1*/
------	------------------------------

VHDL syntax

In VHDL the attribute can be applied through RTL as follows:

```
attribute syn_useioff of data_out : signal is "true|false";
--similarly ace_useioff can be declared and applied.
```

SDC file syntax

Attribute syn_useioff/ace_useioff applied on the input/output port through the constraint file is shown below.

output	define_attribute { <i>p:dout</i> } {syn_useioff} {0 1}
--------	--

input	define_attribute { <i>p:din[7:0]</i> } {ace_useioff} {0 1}
-------	--

register	define_attribute { <i>i:U1.areg[7:0]</i> } {syn_useioff} {1}
----------	--

Global	define_global_attribute {syn_useioff} {0 1}
--------	---

Example 1

In the following example, the attribute `syn_useioff` is set on the top level output port `q`. The top-level output port `q` is marked with the property `syn_useioff = "1"` in the `.vm`-netlist. Top level output port is not driven within the sequential block hence the attribute will only be forward annotated on the port. In this case top level output port can be connected to a register inside lower level submodules.

```
module test_top (
    input clk,
    input d,
    output q/* synthesis syn_useioff=1 */;
    test U (
        .clk(clk),
        .d(d),
        .q(q));
endmodule
```

For the above example, the property `syn_useioff` is forward annotated to the output port '`q`' in the `.vm` netlist.

```
(*syn_useioff = 1*) output q;
```

Example 2

The following is a Verilog example for `syn_useioff` applied on a register of the sub-module.

```
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=1 */;
    assign q = qreg;
    always @ (posedge clk)
```

```

begin
  temp <= d;
  qreg <= temp;
end
endmodule

```

For the above example, the property `syn_useioff = 1` is forward annotated on the register instance in the .vm netlist.

```
(*syn_useioff = 1*) DFF qreg;
```

Example 3

The following is a Verilog example for `syn_useioff` applied on an output port in the lower-level module. The output port is driven inside the sequential block hence attribute will be forward annotated on the output register.

```

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 reg q/* synthesis syn_useioff=1 */);
  reg temp;

//reg qreg;

//assign q = qreg;
  always @ (posedge clk)
  begin
    temp <= d;
    q <= temp;
  end
endmodule

```

For the above example, the property `syn_useioff = 1` is forward annotated to register instance `q` in the netlist.

```
(*syn_useioff = 1*) DDF q;
```

Example 4

The `syn_useioff` attribute is applied in the VHDL RTL.

```
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;
```

In the above example, the `syn_useioff = 0` property is forward annotated to the register `data_out` in the netlist.

```
(*syn_useioff = 0*) dff data_out;
```

Example 5

The `syn_useioff` and `ace_useioff` attributes are applied globally through the top level.

```
module test_top (
    input clk,
    input d,
    output q) /* synthesis syn_useioff=1 */;
    test U (
        .clk(clk),
        .d(d),
        .q(q));

```

```
endmodule
module test(
    input clk,
    input d,
    output q);
    reg temp;
    reg qreg;
    assign q = qreg;
    always @ (posedge clk)
        begin
            temp <= d;
            qreg <= temp;
        end
endmodule
```

In the above example, the properties are forward annotated at the top level in the netlist.

```
(*syn_useioff = 1/0*) module test_top(...
```


syn_user_instance

Synplify Premier

Attribute

Prevents optimizations on user-instantiated primitives, but does not apply to inferred logic.

Vendor	Devices
Xilinx	All

syn_user_instance Values

Value	Description	Default
0 false	The tool can optimize user-instantiated primitives as needed, as long as advanced synthesis mode is enabled. See syn_macro , syn_user_instance , Advanced Synthesis, and Cores , on page 378 for details about how this attribute works with other attributes.	Yes
1 true	Prevents optimizations to user-instantiated primitives. If you specify this attribute globally, it prevents optimizations to all user-instantiated primitives in the whole design. If it is set on a module or instance, it prevents optimization to primitives within the module or instance at all hierarchical levels. If the module or instance contains RTL in addition to user-instantiated primitives, the tool can still optimize the RTL portion.	

Description

The `syn_user_instance` attribute determines whether a user-instantiated primitive, such as LUT6 or FDCE, is optimized. It does not affect inferred logic. When this attribute is set to prevent optimizations on user-instantiated

primitives, the logic around the primitive can still be optimized. For example, if the design has an instantiated input buffer primitive that drives an instantiated flip-flop primitive, the tool might insert a clock buffer between the two.

The `syn_user_instance` attribute is similar to the `syn_macro` attribute, but there are differences in how the objects are handled. In addition, you can set `syn_user_instance` on an instance, but you must set `syn_macro` on a view or module. `syn_macro` is also available in the Synplify Pro tool; `syn_user_instance` is specific to the Synplify Premier software. You can use the `syn_user_instance` attribute to selectively disable optimizations for instantiated user-instantiated primitives when the global advanced synthesis mode is enabled.

If both attributes are applied to the same object, `syn_user_instance` takes precedence. See [syn_macro, syn_user_instance, Advanced Synthesis, and Cores, on page 378](#) for further details about how the attributes affect optimization when used together.

syn_user_instance Syntax

Default Value	Global	Object
0	Yes	View or module User-instantiated primitives like LUTs, flip-flops, and block RAM

This table shows the syntax for setting this attribute in different file formats:

FDC	<code>define_attribute {i: instName} syn_user_instance {1 0}</code> <code>define_attribute {v: moduleName} syn_user_instance {1 0}</code> <code>define_global_attribute syn_user_instance {1 0}</code>	Constraint Examples
Verilog	<code>module moduleName () /* synthesis syn_user_instance = 1 0 */</code> <code>instanceName /* synthesis syn_user_instance = 1 0 */</code>	Verilog Examples
VHDL	<code>attribute syn_user_instance of inst:: label is true false;</code>	VHDL Examples

Constraint Examples

Global setting	<code>define_global_attribute syn_user_instance {1}</code>
Macro view	<code>define_attribute {v:work.prep2_2} syn_user_instance {1}</code>
User-instantiated primitive	<code>define_attribute {i:LUTinst0} syn_user_instance {1}</code>

Verilog Examples

Verilog Example 1: syn_user_instance on a View

```

module LutRestructure(
    input din,
    input in_lut0,in_lut1,in_lut2,in_lut3,in_lut4,in_lut5,
    input in2_lut0,in2_lut1,in2_lut2,in2_lut3,
    output out);

LutRestructure_mod inst0(
    .din(din),.in_lut0(in_lut0),.in_lut1(in_lut1),
    .in_lut2(in_lut2),.in_lut3(in_lut3),.in_lut4(in_lut4),
    .in_lut5(in_lut5),.in2_lut0(in2_lut0),.in2_lut1(in2_lut1),
    .in2_lut2(in2_lut2),.in2_ut3(in2_lut3),.out(out));
endmodule

module LutRestructure_mod(
    input din,
    input in_lut0,in_lut1,in_lut2,in_lut3,in_lut4,in_lut5,
    input in2_lut0,in2_lut1,in2_lut2,in2_lut3,
    output out) /*synthesis syn_user_instance = 1*/;
wire out_1;

LUT6 LUTinst0(
    .O(out_1),.I5(in_lut5),.I2(in_lut4),.I0(in_lut3),
    .I4(in_lut2),.I3(in_lut1),.I1(in_lut0));
defparam LUTinst0.INIT = 64'h8000000000000000;

LUT5 LUTinst1(
    .O(out),.I2(in2_lut0),.I0(din),.I4(in2_lut1),
    .I1(in2_lut2),.I3(out_1));
defparam LUTinst1.INIT = 32'h80000000;
endmodule

```

Verilog Example 2: syn_user_instance on an Instance

```

module LutRestructure(
    input din,
    input in_lut0,in_lut1,in_lut2,in_lut3,in_lut4,in_lut5,
    input in2_lut0,in2_lut1,in2_lut2,in2_lut3,
    output out);

```

```

LutRestructure_mod inst0(
    .din(din), .in_lut0(in_lut0), .in_lut1(in_lut1),
    .in_lut2(in_lut2), .in_lut3(in_lut3), .in_lut4(in_lut4),
    .in_lut5(in_lut5), .in2_lut0(in2_lut0), .in2_lut1(in2_lut1),
    .in2_lut2(in2_lut2), .in2_lut3(in2_lut3), .out(out));
endmodule

module LutRestructure_mod(
    input din,
    input in_lut0, in_lut1, in_lut2, in_lut3, in_lut4, in_lut5,
    input in2_lut0, in2_lut1, in2_lut2, in2_lut3,
    output out);
    wire out_1;

    LUT6 LUTinst0(
        .O(out_1), .I5(in_lut5), .I2(in_lut4), .I0(in_lut3), .I4(in_lut2),
        .I3(in_lut1), .I1(in_lut0))
        /*synthesis syn_user_instance = 1*/;
    defparam LUTinst0.INIT = 64'h8000000000000000;

    LUT5 LUTinst1(
        .O(out), .I2(in2_lut0), .I0(din), .I4(in2_lut1), .I1(in2_lut2),
        .I3(out_1)) /*synthesis syn_user_instance = 1*/;
    defparam LUTinst1.INIT = 32'h80000000;
endmodule

```

VHDL Examples

VHDL Example 1: syn_user_instance on a View

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library synplify;
use synplify.components.all;
library UNISIM;
use UNISIM.VCOMPONENTS.all;

entity LutRestructure is
    port (out_c : out std_logic;
          in2_lut1_c : in std_logic;
          in2_lut0_c : in std_logic;
          in2_lut2_c : in std_logic;
          din_c : in std_logic;
          in_lut5_c : in std_logic;
          in_lut2_c : in std_logic;

```

```
in_lut1_c : in std_logic;
in_lut4_c : in std_logic;
in_lut0_c : in std_logic;
in_lut3_c : in std_logic);
end LutRestructure;

architecture beh of LutRestructure is
signal OUT_L : std_logic;
signal GND : std_logic;
signal VCC : std_logic;
attribute syn_user_instance : boolean;
attribute syn_user_instance of beh : architecture is true;

begin
LUTINST0: LUT6
generic map (INIT => X"8000000000000000")
port map (
    I0 => in_lut3_c,
    I1 => in_lut0_c,
    I2 => in_lut4_c,
    I3 => in_lut1_c,
    I4 => in_lut2_c,
    I5 => in_lut5_c,
    O => OUT_L);
LUTINST1: LUT5
generic map (INIT => X"80000000")
port map (
    I0 => din_c,
    I1 => in2_lut2_c,
    I2 => in2_lut0_c,
    I3 => OUT_L,
    I4 => in2_lut1_c,
    O => out_c);
GND <= '0';
VCC <= '1';
end beh;
```

VHDL Example 2: syn_user_instance on an Instance

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library synplify;
use synplify.components.all;
library UNISIM;
use UNISIM.VCOMPONENTS.all;
```

```
entity LutRestructure is
    port (out_c : out std_logic;
          in2_lut1_c : in std_logic;
          in2_lut0_c : in std_logic;
          in2_lut2_c : in std_logic;
          din_c : in std_logic;
          in_lut5_c : in std_logic;
          in_lut2_c : in std_logic;
          in_lut1_c : in std_logic;
          in_lut4_c : in std_logic;
          in_lut0_c : in std_logic;
          in_lut3_c : in std_logic);
end LutRestructure;

architecture beh of LutRestructure is
signal OUT_L : std_logic;
signal GND : std_logic;
signal VCC : std_logic;
attribute syn_user_instance : boolean;
attribute syn_user_instance of LUTINST0 : label is true;
attribute syn_user_instance of LUTINST1 : label is true;

begin
LUTINST0: LUT6
generic map (INIT => X"8000000000000000")
port map (
    I0 => in_lut3_c,
    I1 => in_lut0_c,
    I2 => in_lut4_c,
    I3 => in_lut1_c,
    I4 => in_lut2_c,
    I5 => in_lut5_c,
    O => OUT_L);
LUTINST1: LUT5
generic map (INIT => X"80000000")
port map (
    I0 => din_c,
    I1 => in2_lut2_c,
    I2 => in2_lut0_c,
    I3 => OUT_L,
    I4 => in2_lut1_c,
    O => out_c);
GND <= '0';
VCC <= '1';
end beh;
```

syn_vote_loops

Attribute

Synplify Premier

Use this attribute when a loop exists in the module for which distributed TMR is applied. By default, to restore the state after effects of an SEU fault, the loop in each of the three instances must insert a voter. If you have used a mitigation mechanism to avoid accumulation of faults in a sequential feedback path and do not want the overhead of voter logic, you can turn off adding voters for your design by using this attribute.

Vendor	Technologies
Intel FPGA	Arria V, Cyclone III, Cyclone IV, Cyclone V, Stratix III, Stratix IV, Stratix V, Stratix 10, and Agilex
Lattice	ECP3, MachXO2
Microchip	IGLOOE, IGLOO+, IGLOO, ProASIC3L, ProASIC3E, ProASIC3, SmartFusion
Xilinx	Artix-7, Kintex-7, and Virtex families

syn_vote_loops Values

Value	Description	Default	Global
false	Turns off adding voters for the design.	true	No
true	Voter logic is inserted in the design for sequential feedback paths.		

Description

Use this attribute when a loop exists in the module for which distributed TMR is applied. By default, to restore the state after effects of an SEU fault, the loop in each of the three instances must insert a voter. If you have used a

mitigation mechanism to avoid accumulation of faults in a sequential feedback path and do not want the overhead of voter logic, you can turn off adding voters for your design by setting the `syn_vote_loops` attribute to false. Doing so can provide the following results:

- Reduces the area for the design because less voter logic is inserted.
- Improves timing or quality of results (QoR) for the design. The logic in the critical path is reduced through the synchronous voters.
- Does not change the interface for the TMR module.

If this attribute is set to false, the message “Found sequential loop” will not appear in the log file reducing its size and runtime because messages are not included and is less than when voter logic is inserted for sequential feedback paths. You must use this attribute with the `syn_radhardlevel` attribute. This attribute can only be applied on a view with the `syn_radhardlevel` attribute. Do not use this attribute on any of its sub-hierarchies or an instance. For more information, see [syn_radhardlevel, on page 495](#).

syn_vote_loops Syntax

SCOPE	<code>define_attribute {object} syn_vote_loops {true false}</code>	FDC Example
Verilog	<code>object /* syn_vote_loops = "true false" */;</code>	Verilog Example
VHDL	<code>attribute syn_vote_loops : string; attribute syn_vote_loops of rtl : architecture is "true false";</code>	VHDL Example

FDC Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	✓	view	v:work.top	syn_radhardlevel	distributed_tmr	string	Radiation-hardened implementation style
2	✓	view	v:work.top	syn_vote_loops	False	string	Radiation-hardened Fault retention paths

For example:

```
define_attribute {v:work.top} syn_vote_loops {false}
```

Verilog Example

```
module ram (addra,addrb,clka,dinla,acc_out,rst,we,re)
/* synthesis syn_vote_loops = "true" */;
```

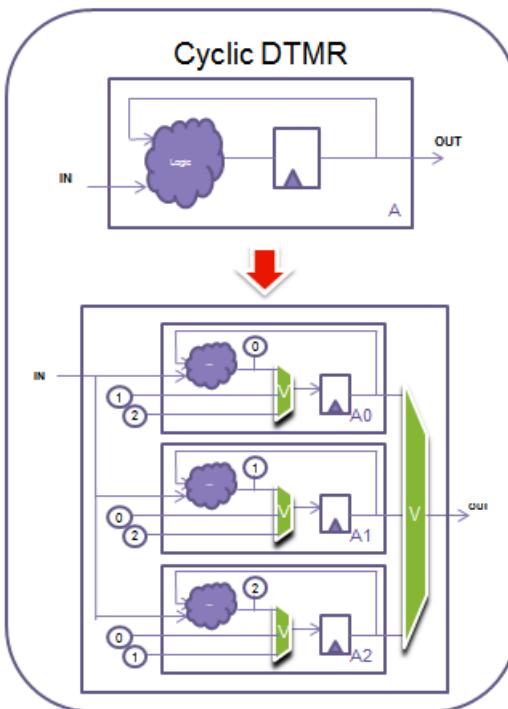
VHDL Example

```
architecture RTL of ram is
attribute syn_vote_loops : string;
attribute syn_vote_loops of RTL : architecture is "false";
```

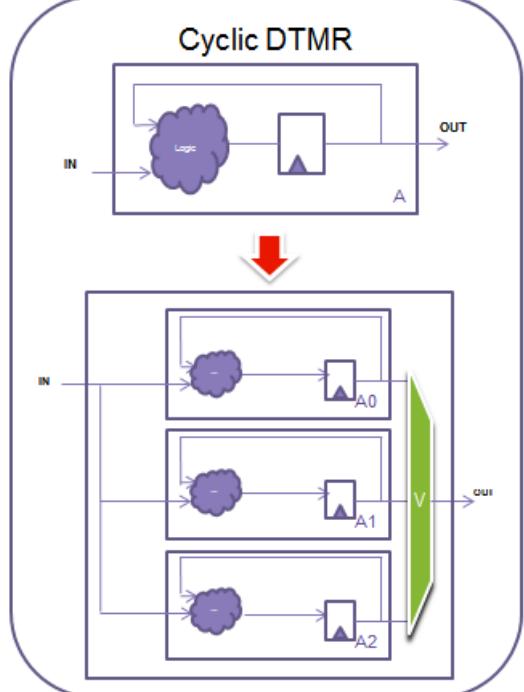
Effect of Using syn_vote_loops

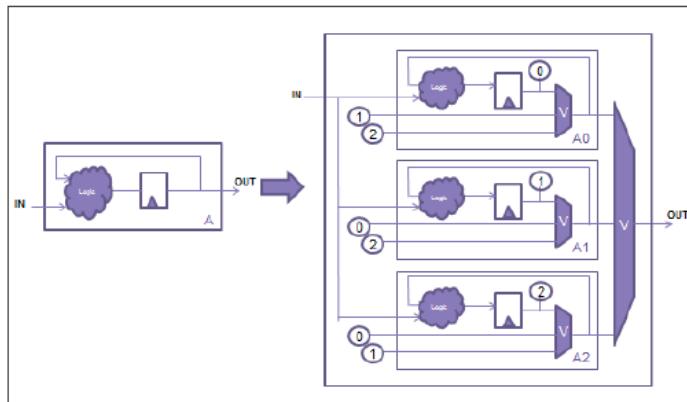
The following examples show the effects of using the `syn_vote_loops` attribute and turning it off on a distributed TMR module with sequential loops.

```
define_attribute {v:work.A} {syn_vote_loops} {true} (Default)
```



```
define_attribute {v:work.A} {syn_vote_loops} {false}
```





syn_vote_register

Attribute

Synplify Premier

This attribute enables voter logic to be inserted after every register of the module specified for distributed TMR.

Vendor	Technologies
Intel FPGA	Arria V, Cyclone III, Cyclone IV, Cyclone V, Stratix III, Stratix IV, Stratix V, Stratix 10, and Agilex families
Lattice	ECP3 and MachXO2 families
Microchip	IGLOOE, IGLOO+, IGLOO, ProASIC3L, ProASIC3E, ProASIC3, SmartFusion
Xilinx	Artix-7, Kintex-7, and Virtex families

syn_vote_register Values

Value	Description	Scope
all	Enables voter logic to be inserted after every register of the distributed TMR module.	Global
none	Disables voter logic insertion on the specified flushable register within the distributed TMR module. This is the default.	Register Instance

Description

When you specify this attribute globally with the value `all`, voter logic is inserted on the output ports and after every register (flushable circuits) of the module specified for distributed TMR. When disabled, voter logic is inserted only on the output ports, sequential loops, and clock enabled flip-flop loops

within the module specified for distributed TMR. Additionally, you can disable voter logic insertion for individual flushable registers by specifying the attribute with the value none.

syn_vote_register Syntax

SCOPE	define_global_attribute {syn_vote_register} {all none} define_attribute <i>registerInstanceName</i> {syn_vote_register} {all none}	FDC Example
Verilog	<i>module</i> /* syn_vote_register = "all none" */;	Verilog Example
VHDL	attribute syn_vote_register : boolean; attribute syn_vote_register of <i>architectureName</i> : architecture is "all" none";	VHDL Example

FDC Example

```
define_global_attribute {syn_vote_register} {all}
define_attribute registerInstanceName {syn_vote_register} {none}
```

Verilog Example

```
module top (clk, rst, set, en, d, q1)
    /* synthesis syn_vote_register = "all" */;
    // Input ports
    input clk;
    input rst;
    input set;
    input en;
    input d;

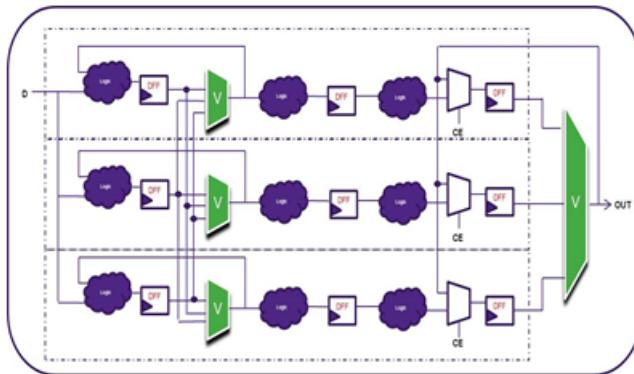
    // Output ports
    output q1;
    dff inst_dff (clk, rst, set, en, d, q1);
endmodule
```

VHDL Example

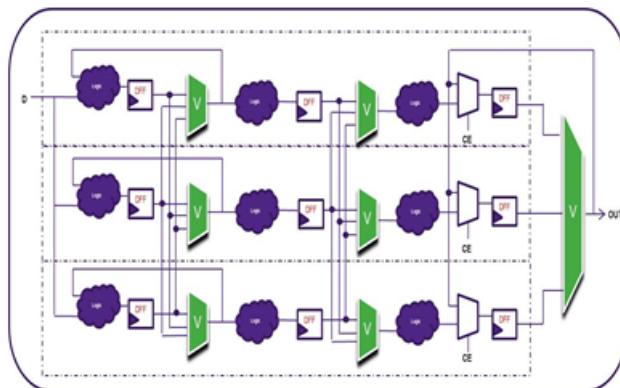
```
library synplify;
architecture rtl of top is
attribute syn_vote_register : string;
attribute syn_vote_register of rtl: architecture is "all";
-- Other code
```

Effect of Using `syn_vote_register`

This example shows `syn_vote_register` disabled. Voter logic is inserted on the output ports, sequential loops, and clock enabled flip-flop loops within the module specified for distributed TMR.



The following example shows `syn_vote_register="all"`. Voter logic is inserted on the output ports and after every register within the module specified for distributed TMR.



Using syn_vote_register on Lower-Level Instances Within Block TMR

You can specify whether or not voter logic is inserted after every register for instances within the hierarchy of the block TMR module. The `syn_vote_register` attribute can be specified as follows:

Attribute	Value	Scope
<code>syn_vote_register</code>	<code>all</code>	Global/View/Instance
	<code>none</code>	Instance

The following block TMR design hierarchy is used for the examples below:



Example 1

Specify the following syntax that enables voter logic to be inserted after every register stage inside instance U1 within the block TMR module Mod.

```
define_attribute {v:work.Mod} {syn_radhardlevel} {block_tmr}  
or  
define_global_attribute {syn_radhardlevel} {block_tmr}  
define_attribute {i:U1} {syn_vote_register} {all}
```

Example 2

Specify the following syntax that enables voter logic to be inserted after every register stage, except for one register instance inside instance U1 within the block TMR module Mod.

```
define_attribute {v:work.Mod} {syn_radhardlevel} {block_tmr}  
or  
define_global_attribute {syn_radhardlevel} {block_tmr}  
define_attribute {i:U1} {syn_vote_register} {all}  
define_attribute {i:U1.U11.reg1} {syn_vote_register} {none}
```


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 95. 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 128](#) 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 401](#) 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;

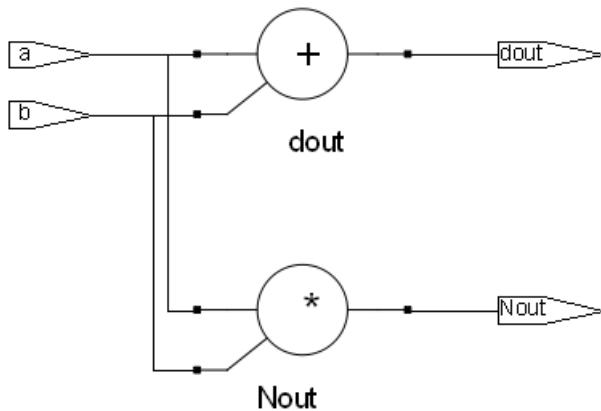
architecture rtl of test is
begin
    dout <= a + b;

--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;
```

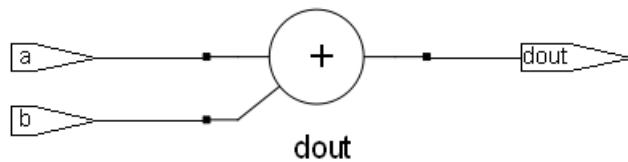
Effects of Using translate_off/translate_on

Here is the RTL view before applying the attribute.



:

This is the RTL view after applying the attribute.



xc_area_group

Attribute/Directive

Synplify Pro, Synplify Premier

Specifies the region where the instance should be placed.

Vendor	Technology
Xilinx	Virtex and newer families
	Spartan and newer families

xc_area_group Values

Default	Global	Object
N/A	No	View

Description

xc_area_group is used to assign a compile point to a Xilinx area group; by specifying its location and area. Physical regions you define with the Design Planner option in the Synplify Premier tool are automatically assigned to corresponding Xilinx area groups. During fitting, the Xilinx place-and-route tool places all of the compile-point logic within the assigned area group.

Apply this attribute to a compile point. The attribute assigns the compile point to a Xilinx area group, specifying its location and area. During fitting, the Xilinx place-and-route tool places all of the compile point logic within the assigned area group. You must not use this attribute on a module that has been instantiated more than once.

The attribute value specifies the location and area of the area group, by locating its top-left and bottom-right corners. N1 and N2 are integers that correspond to X and Y coordinates of the corners, respectively.

For Spartan families, specify: CLB_R N1 C N2

For Virtex families, specify: SLICE_X N1 Y N2

Syntax

FDC	define_attribute {v:module architectureName} xc_area_group {topleft:bottomright}	SCOPE Example
Verilog	/* synthesis xc_area_group = "CLB_R10C29:CLB_R16C35" */;	Verilog Example
VHDL	attribute xc_area_group : string; attribute xc_area_group of arch: architecture is "CLB_R10C29:CLB_R16C35";	VHDL Example

SCOPE Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	instance	v:work.mod10	xc_area_group	CLB_R10C29:CLB_R16C35	string	Specifies the region where instance should be placed

```
define_attribute {v:work.mod10} {xc_area_group}
    {CLB_R10C29:CLB_R16C35}
```

Verilog Example

```
module xc_area_group (a,b,c,clk,out1,out2) /* synthesis
  xc_area_group = "CLB_R10C29:CLB_R16C35" */;
  input a,b,c,clk;
  output reg out1,out2;
  mod10 inst1 (.clk(clk),.c(c),.d(out2));
  always @ (posedge clk)
  out1 <= a + b;
endmodule

module mod10 (clk,c,d);
  input c,clk;
  output reg d;
  always @ (posedge clk)
  d <= c;
endmodule
```

VHDL Example

```
library IEEE;
use ieee.std_logic_1164.all;
entity test is
port (a : in std_logic;
      b : in std_logic;
      c : in std_logic;
      clk : in std_logic;
      out1 : out std_logic;
      out2 : out std_logic);
end test;
architecture arch of test is
attribute xc_area_group : string;
attribute xc_area_group of arch: architecture is
"CLB_R10C29:CLB_R16C35";

component test1
port (c : in std_logic;
      clk : in std_logic;
      d : out std_logic);
end component;

begin
inst1 : test1 port map (c =>c, d=>out2,clk=>clk);

process (clk)
begin
if (clk'event and clk ='1')then
out1 <= a and b;
end if;
end process;
end arch;

library IEEE;
use ieee.std_logic_1164.all;

entity test1 is
port (c : in std_logic;
      clk : in std_logic;
      d : out std_logic);
end test1;
```

```
:  
  
architecture arch of test1 is  
begin  
process (clk)  
begin  
if (clk'event and clk ='1')then  
d <= c;  
end if;  
end process;  
end;
```

Effect of Using xc_area_group

The following netlist is generated, before applying the attribute:

```
library work  
(edifLevel 0)  
(technology (numberDefinition))  
(cell mod10 (cellType GENERIC)  
(view verilog (viewType NETLIST)  
(interface  
(port clk (direction INPUT))  
(port c (direction INPUT))  
(port d (direction OUTPUT))  
)  
(contents  
(instance (rename dZ0 "d") (viewRef PRIM (cellRef FD  
(libraryRef UNILIB)))  
)  
(net clk (joined  
(portRef clk)  
(portRef C (instanceRef dZ0))  
)  
(net c (joined  
(portRef c)  
(portRef D (instanceRef dZ0))  
)  
(net d (joined  
(portRef Q (instanceRef dZ0))  
(portRef d)
```

The following netlist is generated, after applying the attribute:

FDC define_attribute {v:work.mod10} {xc_area_group} {CLB_R10C29:CLB_R16C3}

Spartan family:

```
(library work
  (edifLevel 0)
  (technology (numberDefinition))
  (cell mod10 (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port clk (direction INPUT))
        (port c (direction INPUT))
        (port d (direction OUTPUT))
        (instance (rename dZ0 "d") (viewRef PRIM (cellRef FD
          (libraryRef UNILIB)))
          (net clk (joined
            (portRef clk)
            (portRef C (instanceRef dZ0)))
          (net c (joined
            (portRef c)
            (portRef D (instanceRef dZ0)))
          (net d (joined
            (portRef Q (instanceRef dZ0))
            (portRef d)
          )
          (property xc_area_group (string "CLB_R10C29:CLB_R16C35"))
        )
      )
    )
  )
)
```

:
Virtex family:

```
(library work
  (edifLevel 0)
  (technology (numberDefinition))
  (cell mod10 (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port clk (direction INPUT))
        (port c (direction INPUT))
        (port d (direction OUTPUT)))
      )
      (contents
        (instance (rename dZ0 "d") (viewRef PRIM (cellRef FD
          (libraryRef UNILIB)))
        )
        (net clk (joined
          (portRef clk)
          (portRef C (instanceRef dZ0)))
        ))
        (net c (joined
          (portRef c)
          (portRef D (instanceRef dZ0)))
        ))
        (net d (joined
          (portRef Q (instanceRef dZ0))
          (portRef d)))
        )
      )
      (property xc_area_group (string "SLICE_X10Y29:SLICE_X16Y35"))
    )
  )
)
```

xc_clockbuftype

Xilinx

Attribute/Directive

Controls the use of the BUFGDLL buffer for a clock port.

Vendor	Technologies
Xilinx	Virtex-6 and older

xc_clockbufftype Values

Default	Global	Object	Value	Description
None	No	Clock port	BUFGDLL	Infers the buffer BUFGDLL

Description

Uses the Clock Delay Locked Loop primitive (CLKDLL) for a clock port. This is inferred as a buffer called BUFGDLL, which includes the CLKDLL primitive. BUFGDLL consists of an IBUFG, followed by a CLKDLL, followed by a BUFG. It is a special purpose clock delay locked loop buffer for clock skew management.

xc_clockbufftype Syntax Specification

FDC	define_attribute { <i>port</i> } xc_clockbuftype {BUFGDLL}	FDC Example
Verilog	<i>object</i> /* synthesis xc_clockbuftype = "BUFGDLL" */;	Verilog Example
VHDL	attribute xc_clockbuftype of <i>object</i> : <i>objectType</i> is "BUFGDLL"	VHDL Example

:

FDC Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description	Comment
1	<input checked="" type="checkbox"/>	input_port	pcik	xc_clockbuftype	BUFGDLL	string	Use the Xilinx BUFGDLL clock buffer	

Verilog Example

```
module xc_clock (d,rst,clk,q);
    input [3:0] d;
    input rst;
    input clk /* synthesis syn_clockbuftype = "BUFGDLL" */;
    output reg[3:0] q;
    always@(posedge clk)
        begin
            if(rst)
                q<=0;
            else
                q<=d;
        end
    endmodule
```

VHDL Example

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

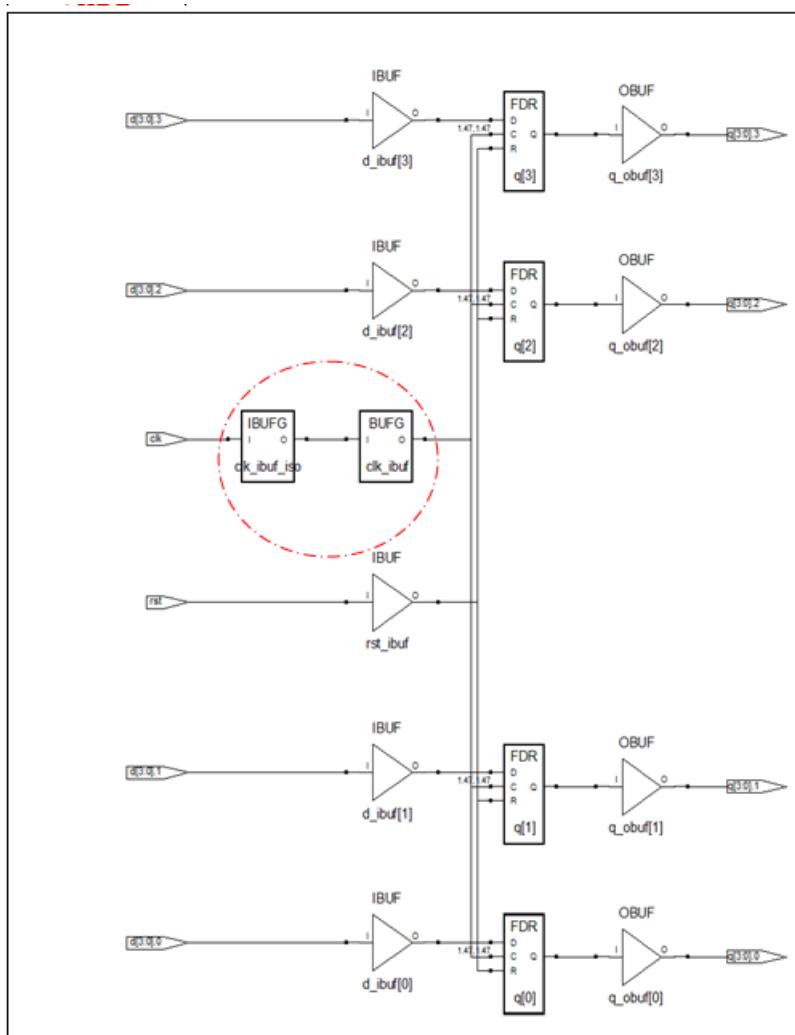
entity dff is
port(data_in:    in std_logic_vector (3 downto 0);
      clock,rst:       in std_logic;
      data_out:    out std_logic_vector (3 downto 0));
attribute syn_clockbuftype : string;
attribute syn_clockbuftype of clock : signal is "BUFGDLL";
end dff;
architecture behv of dff is
begin

process(data_in, clock)
begin
if (clock'event and clock='1') then
if rst='1' then
data_out <= "0000";
else
```

```
    data_out <= data_in;  
end if;  
end if;  
end process;  
end behv;
```

Effect of Using xc_clockbufftype

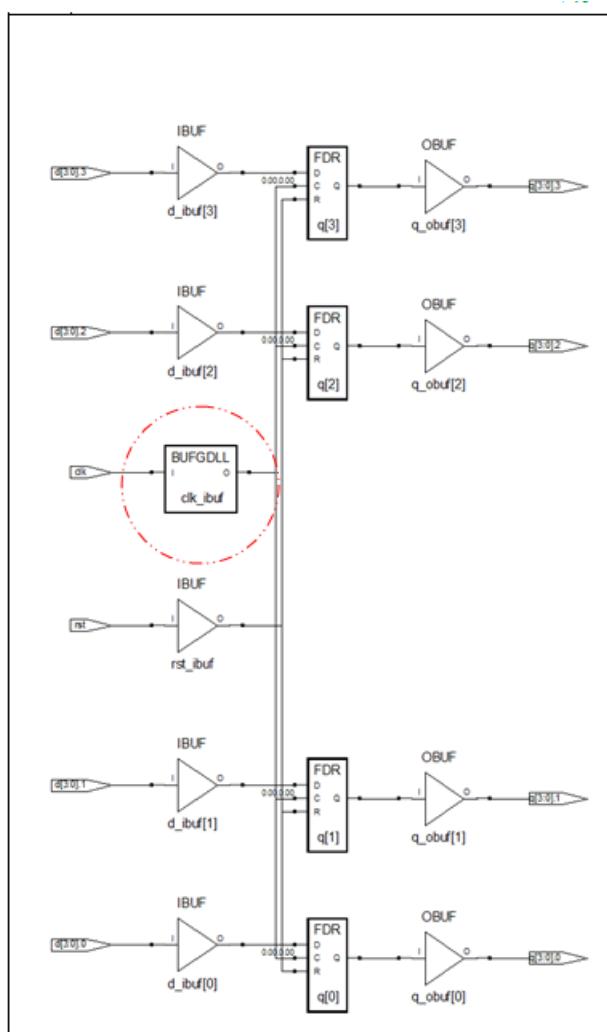
The following example shows a design without the `xc_clockbufftype` attribute:



The next figure shows the same design with a BUFGDLL after the xc_clockbuftype attribute is applied:

Verilog input clk /* synthesis xc_clockbuftype = "BUFGDLL" */;

VHDL attribute xc_clockbuftype : string;
 attribute xc_clockbuftype of clock : signal is "BUFGDLL"



xc_fast

Attribute

Synplify Pro, Synplify Premier

Speeds up transition time of the output driver.

Vendor	Technology
Xilinx	All

xc_fast Values

Value	Description
0 false (Default)	Keeps the transition time slow.
1 true	Speeds up the transition time.

Description

Use this attribute to make the transition time of the output driver fast. The default transition time is slow (see [xc_slow, on page 817](#)).

The synthesis tool provides attributes that are passed into the XNF or EDIF netlist for Xilinx placement and routing. These attributes affect the input setup times and output transition times for your I/Os. The transition time of the output driver can be programmed as fast or slow for Xilinx devices. Use the `xc_fast` attribute to decrease the transition time for the output driver. The `xc_fast` attribute gets passed to the Xilinx I/O Block Parameter as `FAST` in the XNF or EDIF netlist.

xc_fast Syntax

Global Support	Object
No	Output port

This is how you specify the attribute in different files:

FDC	define_attribute {object} xc_fast {value}	Constraint Examples
Verilog	Object /* synthesis xc_fast = "value" */;	Verilog Example
VHDL	attribute xc_fast of object: objectType is "value";	VHDL Example

Constraint Examples

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>		p:DATA0[7:0]	xc_fast	1		

```
define_attribute {DATA0[5]} xc_fast {1}
```

Verilog Example

```
module counter (CLK, RST, DATA0);
    input CLK, RST;
    output [7:0] DATA0 /* synthesis xc_fast = 1 */;
    reg [7:0] DATA0;

    always @ (posedge CLK or posedge RST)
    begin
        if (RST)
            DATA0 = 0;
        else
            DATA0 = DATA0 + 1;
    end
endmodule
```

VHDL Example

```

entity count is
port(clk : in bit;
      rst : in bit;
      data0 : out std_logic_vector(7 downto 0)
      );
attribute xc_fast : Boolean;
attribute xc_fast of data0 : signal is true;
end count;

architecture behave of count is
    signal data0_i: std_logic_vector(7 downto 0);
begin

    counter: process (rst,clk)
    begin
        if  rst = '1' then
            data0_i <= "00000000";
        elsif clk = '1' and clk'event then
            data0_i  <= data0_i + "00000001";
            end if;
    end process counter;

    data0 <= data0_i;
end behave;

```

Effect of Using xc_fast

The following table shows a design without the `xc_fast` attribute.

```
Verilog    output [7:0] DATA0 /* synthesis xc fast = 0 */;
```

```
VHDL      attribute xc fast of data0 : signal is false;
```

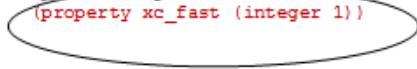
```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell counter (cellType GENERIC)
    (view verilog (viewType NELIST)
      (interface
        (port (array (rename DATA0 "DATA0[7:0]" ) 8) (direction OUTPUT))
        (port CLK (direction INPUT)
      )
      (port RST (direction INPUT)
    )
  )
)
```

The following table shows a design with the `xc_fast` attribute.

Verilog `output [7:0] DATA0 /* synthesis xc_fast = 1 */;`

VHDL `attribute xc_fast of data0 : signal is true;`

```
(library work
  (editLevel 0)
  (technology (numberDefinition ))
  (cell counter (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port (array (rename DATA0 "DATA0[7:0]" ) 8) (direction OUTPUT)
          (property xc_fast (integer 1))
      )
    )
  )
)
```



When place and route is run, a line is added for each output port bit with an `xc_fast` attribute in the following files:

```
E:\srm\xc_fast\rev_1\pr_1\counter.pad(257): P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMS25*|2|12
|FAST|||UNLOCATED|NO|NONE|
E:\srm\xc_fast\rev_1\pr_1\counter_map.mrp(172): | DATA0[0] | IOB| OUTPUT | LVCMS25|| 12| FAST ||
E:\srm\xc_fast\rev_1\pr_1\counter_pad.txt(258):|P2|DATA0[0]|IOB|IO_L2P_A23_2|OUTPUT|LVCMS25*|2|12|
FAST|||UNLOCATED|NO|NONE |
E:\srm\xc_fast\rev_1\pr_1\proj_3.edf(193): <property FAST (string "")>
E:\srm\xc_fast\rev_1\pr_1\counter_map.xrpt(239): <item label="Slew&#xA;Rate" stringID="SLEW_RATE"
value="FAST"/>
E:\srm\xc_fast\rev_1\pr_1\counter_par.xrpt(1450): <item label="Slew&#xA;Rate" stringID="Slew_Rate"
value="FAST"/>
E:\srm\xc_fast\rev_1\pr_1\proj_3.xise(223): <property xil_pn:name="Output Slew Rate" xil_pn:value=
"Fast" il_pn:valueState="default"/>
```

xc_fast_auto

Attribute

By default (`xc_fast_auto` value of 1), the synthesis tool infers the fast output buffers `OBUF_F_24`, `OBUFT_F_24`, and `IOBUF_F_24`. Use this global attribute to force inference of slow buffers.

Vendor	Technology
Xilinx	Virtex-5 and older families

Description

By default (`xc_fast_auto` value of 1), the synthesis tool infers the fast output buffers `OBUF_F_24`, `OBUFT_F_24`, and `IOBUF_F_24`. Use this global attribute to force inference of slow buffers. In HDL source code you specify `xc_fast_auto` for the top-level module or architecture. You can override this attribute on an individual basis by using the `xc_padtype` attribute.

The `xc_fast_auto` attribute has no affect on output ports specified with the `xc_padtype` attribute.

xc_fast_auto Syntax

Global Support	Object
Yes	Module or architecture

The following table summarizes the syntax in different files.

SCOPE	<code>define_global_attribute xc_fast_auto {0 1}</code>	FDC Example
Verilog	<code>object /* synthesis xc_fast_auto = 0 1 */;</code>	Verilog Example xc_fast_auto
VHDL	<code>attribute xc_fast_auto : boolean; attribute xc_fast_auto of object : objectType is false true;</code>	VHDL Example xc_fast_auto

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description	Comment
1	<input checked="" type="checkbox"/>	<any>	<Global>	xc_fast_auto	0	boolean	Enable automatic fast output buffer...	

```
define_global_attribute xc_fast_auto {0}
```

Verilog Example xc_fast_auto

```
//Example: Verilog xc_fast_auto

module top( input a, b, clk, input gsrin, output fout)
/*synthesis xc_fast_auto=1 */;

test T1(a, b, clk, gsrin, fout);

endmodule

module test(a, b, clk, gsrin, out)/* synthesis syn_black_box */;
input a, b, clk;
input gsrin;
output out;
endmodule
```

VHDL Example xc_fast_auto

```
--Example: VHDL xc_fast_auto

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

entity top is
    port (a :    in std_logic;
          b :    in std_logic;
          clk :   in std_logic;
          gsrin :  in std_logic;
          dout :   out std_logic
        );
end top;

architecture rtl of top is
    component test is
        port (a :    in std_logic;
              b :    in std_logic;
              clk :   in std_logic;
              gsrin :  in std_logic;
              o :    out std_logic);
    end component;
begin
    U1 : test port map(a, b, clk, gsrin, dout);
end rtl;

--Black Box module
```

```
:  
  
library ieee;  
  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
  
  
entity test is  
port (a : in std_logic;  
      b : in std_logic;  
      clk : in std_logic;  
      gsrin : in std_logic;  
      o :    out std_logic  
      );  
end;  
  
architecture rtl of test is  
  
attribute syn_black_box : boolean;  
attribute syn_black_box of rtl : architecture is true;  
  
  
attribute xc_fast_auto : boolean;  
attribute xc_fast_auto of rtl : architecture is false;  
begin  
  
  
end;
```

Effect of Using `xc_fast_auto`

The tool infers fast output buffers by default, however, you can control this behavior using the `xc_fast_auto` attribute. If you set `xc_fast_auto=0`, fast buffer inferencing is turned off and this value is forward annotated to the output .edf netlist as follows:

```
(property xc_fast_auto (integer 0))
```


xc_global_buffers

Attribute

Controls the number of global buffers used in a device.

Vendor	Technology
Xilinx	All families

Description

Controls the number of global buffers used in a device. 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 added. Also, if there is a black box in the design that has global buffers, you can use `xc_global_buffers` to prevent the synthesis tool from inferring clock buffers or exceeding the number of global resources. You can only specify this attribute through a constraint file (cannot be specified in HDL source code).

The `xc_global_buffers` attribute has the same effect as the `syn_global_buffers` attribute. If both attributes are specified in the same Xilinx design, `syn_global_buffers` overwrites `xc_global_buffers`. See [syn_global_buffers, on page 311](#) for details about this attribute.

It is recommended that you use the Xilinx global buffers attribute `syn_global_buffers` instead; see [syn_global_buffers, on page 311](#).

xc_global_buffers Syntax

Global Support Object

Yes	Maximum number of global buffers
-----	----------------------------------

The following table summarizes the syntax in different files.

SCOPE define_global_attribute xc_global_buffers {maximum} [FDC Example](#)

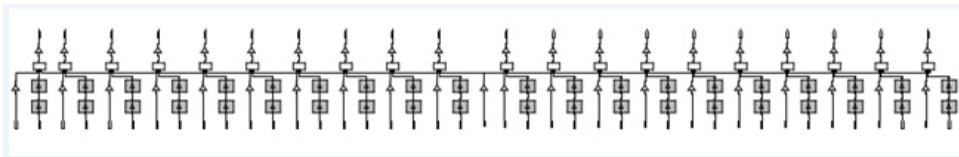
FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description	Comment
1	<input checked="" type="checkbox"/>	<any>	<Global>	xc_global_buffers	3	integer	Number of global buffers	

```
define_global_attribute xc_global_buffers {3}
```

Effect of Using xc_global_buffers

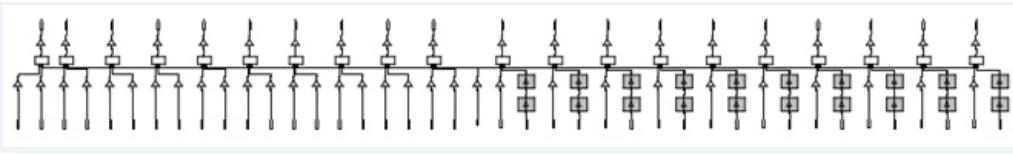
Before applying attribute:



A message like the one below is generated:

```
@W:FX726: | Ignoring out-of-range global buffer count of 33 for  
chip view:work.top (behave)
```

After applying attribute:



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
```


xc_isgsr

Attribute

Specifies that a port on a black box is connected to an internal STARTUP block.

Vendor	Technology
Xilinx	XC4000 and Virtex-4 and older families

Description

Specifies that a port on a black box is connected to an internal STARTUP block. Use this directive with designs targeting Xilinx-specific families where the synthesis tool infers a STARTUP block.

xc_isgsr Syntax

Global Support	Object
No	Black box port

The following table summarizes the syntax in different files.

SCOPE	define_attribute {instance.resetPort} xc_isgsr {0 1}	FDC Example
Verilog	object /* synthesis xc_isgsr = 0 1 */;	Verilog Example xc_isgsr
VHDL	attribute xc_isgsr : string; attribute xc_isgsr of object : objectType is false true;	VHDL Example xc_isgsr

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description	Comment
1	<input checked="" type="checkbox"/>	<any>	bbgsr.gsrin	xc_isgsr	1			

```
define_attribute {bbgsr.gsrin} xc_isgsr {1}
```

Verilog Example xc_isgsr

```
//Example: Verilog xc_isgsr
```

```
module top( input a, b, clk, input gsrin, output fout);

test T1(a, b, clk, gsrin, fout);

endmodule

module test(a, b, clk, gsrin, out)/* synthesis syn_black_box */;
input a, b, clk;
input gsrin /* synthesis xc_isgsr = 1 */;
output out;
endmodule
```

VHDL Example xc_isgsr

```
--Example: VHDL xc_isgsr
```

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

```

entity top is

    port (a :    in std_logic;
          b :    in std_logic;
          clk :   in std_logic;
          gsrin :   in std_logic;
          dout :   out std_logic
        );
end top;

architecture rtl of top is

component test is

    port (a :    in std_logic;
          b :    in std_logic;
          clk :   in std_logic;
          gsrin :   in std_logic;
          o :    out std_logic);
end component;

begin

    U1 : test port map(a, b, clk, gsrin, dout);
end rtl;

--Black Box module

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

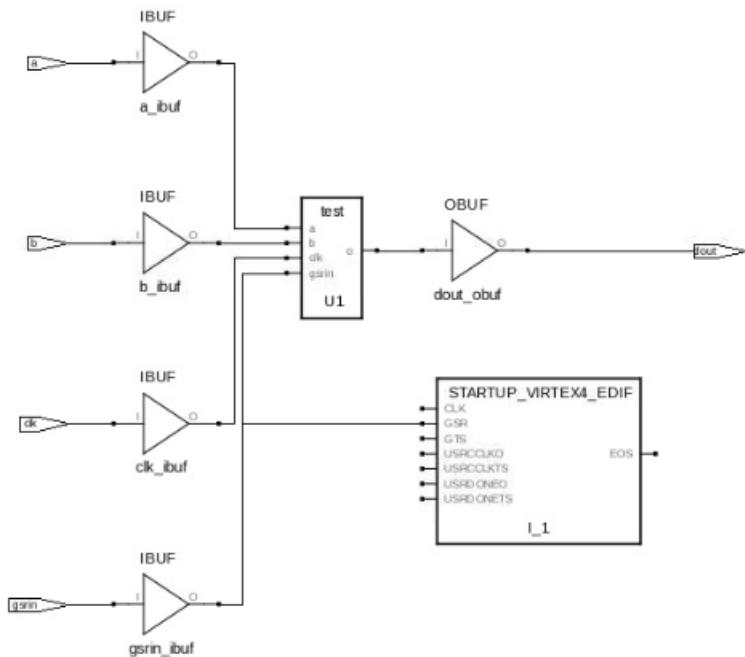
```

```
entity test is
    port (a : in std_logic;
          b : in std_logic;
          clk : in std_logic;
          gsrin : in std_logic;
          o :     out std_logic
        );
attribute xc_isgsr : boolean;
attribute xc_isgsr of gsrin : signal is true;
end;
architecture rtl of test is
attribute syn_black_box : boolean;
attribute syn_black_box of rtl : architecture is true;

begin
end;
```

Effect of Using xc_isgsr

The reset input pin of the black box is connected to a GSR block as shown in the Technology view below:



xc_loc

Attribute

Specifies the location (placement) of ports.

Vendor	Technology
Xilinx	All

Description

Specifies the location (placement) of ports. Refer to the Xilinx databook for valid placement locations. This attribute can be specified in a top-level source file or the constraint file.

It is recommended that you use the Xilinx location attribute `syn_loc` instead; see [syn_loc, on page 367](#).

xc_loc Syntax

Global Support	Object
No	Port

The following table summarizes the syntax in different files:

SCOPE	<code>define_attribute {portDesignName} xc_loc {placements}</code>	FDC Example
Verilog	<code>object /* synthesis xc_loc = "placements" */;</code>	Verilog Example
VHDL	<code>attribute xc_loc of object : objectType is "placements";</code>	VHDL Example

FDC Example

The following example is a constraint file assignment of a pad location to all bits of a 5-bit bus.

```
define_attribute {DATA0[4:0]} xc_loc {P5,P11,P12,P14,P21}
```

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	p:data0[4:0]	xc_loc	P5, P11, P12, P14, P21	string	Port placement

Verilog Example

```
module test(a, b, clk, out1);
    input clk;
    input [2:0]a;
    input [2:0]b;
    output [2:0] out1;
    reg [2:0] out1/* synthesis xc_loc = "P14,P12,P11" */;

    always@(posedge clk)
    begin
        out1 <= a + b;
    end
endmodule
```

VHDL Example

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

entity test is
    generic (s : integer := 4);

    port (
        clk: in std_logic;
        in1: in std_logic_vector(s downto 0);
        in2: in std_logic_vector(s downto 0);
        d_out: out std_logic_vector(9 downto 0));
        attribute xc_loc : string;
        attribute xc_loc of d_out : signal is "P14,P12,P11,P5,P21";
    end test;

    architecture beh of test is
```

```

begin
  process (clk)
  begin
    if rising_edge(clk) then
      d_out <= in1 & in2;
    end if;
  end process;
end beh;

```

Effect of Using xc_loc

The specified ports are locked to particular pins of the FPGA. This information, along with the output edf netlist, is forward-annotated to the P&R tool through the ncf file for Virtex-6 and xdc file for Virtex-7.

After applying the attribute on Virtex-7:

```

create_clock [get_ports {clk}] -period {5}
set_property LOC P11 [get_cells {out1obuf[0]}]
set_property LOC P12 [get_cells {out1obuf[1]}]
set_property LOC P14 [get_cells {out1obuf[2]}]

set_property LOC P11 [get_ports {out1[0]}]
set_property LOC P12 [get_ports {out1[1]}]
set_property LOC P14 [get_ports {out1[2]}]
set_property IOB true [get_cells {out1[2]}]
set_property IOB true [get_cells {out1[1]}]
set_property IOB true [get_cells {out1[0]}]

```

After applying the attribute on Virtex-6:

```

NET "out1[0]" LOC="P11";
NET "out1[1]" LOC="P12";
NET "out1[2]" LOC="P14";

```

Examples: xc_loc on Individual Bus Bits

You can use xc_loc to specify locations of individual bits of the bus in one of two ways. This example specifies the bit:

```

define_attribute {valid[0]} xc_loc {R6}
define_attribute {valid[1]} xc_loc {T2}
define_attribute {valid[2]} xc_loc {R5}
define_attribute {valid[3]} xc_loc {R1}

```

:

This example uses the b: prefix as well as the bit slice:

```
define_attribute {b:valid[0]} xc_loc {R6}
define_attribute {b:valid[1]} xc_loc {T2}
define_attribute {b:valid[2]} xc_loc {R5}
define_attribute {b:valid[3]} xc_loc {R1}
```

xc_map

Attribute

Specifies RLOCs. RLOCs are relative location constraints.

Vendor	Technology
Xilinx	Virtex families

Description

RLOCs are relative location constraints. They let you control placement in critical sections, thus improving performance.

Specify RLOCs using `xc_map` along with `xc_rloc` and `xc_uset`. As with other attributes, you can define them in the source code, or in the SCOPE window.

See [Specifying RLOCs, on page 914](#) in the *User Guide* for further information.

xc_map Syntax

Global Support	Object
No	Primitive

The following table summarizes the syntax in different files:

SCOPE	<code>define_attribute {v:<i>primitiveName</i>} xc_map {fmap hmap lut}</code>	FDC Example
Verilog	<code>object /* synthesis xc_map = "fmap hmap lut" */;</code>	Verilog Example
VHDL	<code>attribute xc_map of <i>object</i> : <i>objectType</i> is "fmap hmap lut";</code>	VHDL Example

:

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	v:hmap_xor4	xc_map	fmap	string	Map entity to fmap/hmap/lut

```
define_attribute {v:hmap_xor4} xc_map {fmap}
```

Verilog Example

```
module fmap_xor4 (z, a, b, c, d) /* synthesis xc_map=fmap */;
  output z;
  input a, b, c, d;
  assign z = a ^ b ^ c ^ d;
endmodule

module hmap_xor3 (z, a, b, c) /* synthesis xc_map=hmap */;
  output z;
  input a, b, c;
  assign z = a ^ b ^ c;
endmodule

module clb_xor9 (z, a);
  output z;
  input [8:0] a;
  wire z03, z47;
  fmap_xor4 x03 /* synthesis xc_uset="SET1" xc_rloc="R0C0.f" */
    (z03, a[0], a[1], a[2], a[3]);
  hmap_xor3 zz /* synthesis xc_uset="SET1" xc_rloc="R0C0.h" */
    (z, z47, a[4], a[5]);
  fmap_xor4 x47 /* synthesis xc_uset="SET1" xc_rloc="R0C0.g" */
    (z47, z03, a[5], a[6], a[7]);
endmodule

module xor9top (z, a);
  output z;
  input [8:0] a;
  clb_xor9 x (z, a);
endmodule
```

VHDL Example

```
--hmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity hmap_xor4 is
    port (a, b, c, d : in std_logic;
          z : out std_logic);
end hmap_xor4;
architecture rtl of hmap_xor4 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "hmap";
begin
    z <= a xor b xor c xor d;
end rtl;

-- fmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity fmap_xor3 is
port (a, b, c : in std_logic;
      z : out std_logic);
end fmap_xor3;

architecture rtl of fmap_xor3 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "fmap";
begin
    z <= (a and b) xor c;
end rtl;

-- hmap & fmap instantiations
library IEEE;
use IEEE.std_logic_1164.all;

entity clb_xor9 is
port (a : in std_logic_vector(8 downto 0);
      z : out std_logic
     );
end clb_xor9;

architecture rtl of clb_xor9 is
signal z03, z47 : std_logic;
```

```
component fmap_xor3
port (a : in std_logic;
      b : in std_logic;
      c : in std_logic;
      z : out std_logic
    );
end component;

component hmap_xor4
port (a : in std_logic;
      b : in std_logic;
      c : in std_logic;
      d : in std_logic;
      z : out std_logic
    );
end component;
attribute xc_use : string;
attribute xc_rloc : string;
attribute xc_use of x03 : label is "SET1";
attribute xc_rloc of x03 : label is "R0C0.f";
attribute xc_use of x47 : label is "SET1";
attribute xc_rloc of x47 : label is "R0C0.g";
attribute xc_use of zz : label is "SET1";
attribute xc_rloc of zz : label is "R0C0.h";
begin
  x03 : hmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
  x47 : hmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
  zz : fmap_xor3 port map(z03, z47, a(8), z);
end rtl;

--Top level entity definition
library IEEE;
use IEEE.std_logic_1164.all;

entity top is
port (a : in std_logic_vector(8 downto 0);
      z : out std_logic
    );
end top;

architecture rtl of top is
component clb_xor9
port (a : in std_logic_vector (8 downto 0);
```

```
z : out std_logic);
end component;
begin
    U1: clb_xor9 port map (a, z);
end rtl;
```

Effect of Using xc_map

After applying the attribute:

The following example shows forward-annotated regional constraints in the output edif netlist.

```
(property xc_map (string "hmap"))
(property orig_inst_of (string "nmap_xor4"))
)
)
(cell clb_xor9 (cellType GENERIC)
(view netlist (viewType NETLIST)
(interface
    (port (array (rename a_c "a_c(8:0)") 9) (direction INPUT))
    (port z_c (direction OUTPUT))
)
(contents
    (instance zz (viewRef PRIM (cellRef LUT3 (libraryRef
VIRTEX)))
        (property RLOC (string "R0C0.h"))
        (property HU_SET (string "U1$SET1"))
        (property INIT (string "8'h78"))
)
        (instance x03 (viewRef netlist (cellRef hmap_xor4))
        (property RLOC (string "R0C0.f"))
        (property HU_SET (string "SET1"))
)
        (instance x47 (viewRef netlist (cellRef hmap_xor4_0))
        (property RLOC (string "R0C0.g"))
        (property HU_SET (string "SET1"))
)
```


xc_padtype

Attribute

Specifies an I/O buffer standard.

Vendor	Technology
Xilinx	Virtex families

Description

It is recommended that you use the Xilinx I/O buffer standard attribute `syn_pad_type` instead; see [syn_pad_type, on page 459](#).

xc_props

Attribute

Was used to specify the Xilinx attributes to forward-annotate to the gate-level netlist.

Vendor	Technology
Xilinx	All

Description

This is an obsolete attribute that was used to specify the Xilinx attributes to forward-annotate to the gate-level netlist. These attributes are now forward-annotated directly with synthesis attribute statements as shown below. For step-by-step procedures of different ways to specify INIT values, see [Initializing RAMs, on page 528](#) in the *User Guide*.

xc_props Syntax

Global Support	Object
No	Instance

The following table summarizes the syntax in different files:

SCOPE	FDC Syntax
Verilog	Verilog Syntax
VHDL	VHDL Syntax

FDC Syntax

Old xc_props	<pre>define_attribute {FOO_0} xc_props {"HU_SET=FOOBAR, RLOC_ORIGIN=X5Y2, RLOC=X0Y0"} define_attribute {FOO_1} xc_props {"HU_SET=FOOBAR, RLOC=X0Y1"}</pre>
New synthesis attribute	<pre>define_attribute {i:FOO_0} HU_SET {FOOBAR} define_attribute {i:FOO_0} RLOC_ORIGIN {X5Y2} define_attribute {i:FOO_0} RLOC {X0Y0} define_attribute {i:FOO_1} HU_SET {FOOBAR} define_attribute {i:FOO_1} RLOC {X0Y1} define_attribute {i:FOO_0} syn_useioff {0} define_attribute {i:FOO_1} syn_useioff {0}</pre>

Verilog Syntax

Old xc_props	<pre>FD FOO_0 (.D(D0),.Q(Q0),.C(CLK)) /* synthesis xc_props="HU_SET=FOOBAR, RLOC_ORIGIN=X5Y2, RLOC=X0Y0" */; FD FOO_1 (.D(D1),.Q(Q1), .C(CLK)) /* synthesis xc_props="HU_SET=FOOBAR, RLOC=X0Y1" */;</pre>
New synthesis attribute	<pre>FD FOO_0 (.D(D0),.Q(Q0),.C(CLK)) /* synthesis HU_SET="FOOBAR" RLOC_ORIGIN="X5Y2" RLOC="X0Y0" */; FD FOO_1 (.D(D1),.Q(Q1), .C(CLK)) /* synthesis HU_SET="FOOBAR" RLOC="X0Y1" */;</pre>

VHDL Syntax

Old xc_props

```
architecture struct of attr is
attribute xc_props : string;
attribute xc_props of FOO_0: label is "HU_SET=FOOBAR,
RLOC_ORIGIN=X5Y2, RLOC=X0Y0";
attribute xc_props of FOO_1: label is "HU_SET=FOOBAR,
RLOC=X0Y1";
begin
FOO_0 : FD port map(D => D0, Q => Q0, C => CLK);
FOO_1 : FD port map(D => D1, Q => Q1, C => CLK);
```

New synthesis attribute

```
architecture struct of attr is
attribute HU_SET : string;
attribute RLOC_ORIGIN : string;
attribute RLOC : string;
attribute HU_SET of FOO_0: label is "FOOBAR";
attribute RLOC_ORIGIN of FOO_0: label is "X5Y2";
attribute RLOC of FOO_0: label is "X0Y0";
attribute HU_SET of FOO_1: label is "FOOBAR";
attribute RLOC of FOO_1: label is "X0Y1";
begin
FOO_0 : FD port map(D => D0, Q => Q0, C => CLK);
FOO_1 : FD port map(D => D1, Q => Q1, C => CLK);
```

If you want to specify different synthesis attributes values for each register bit on a bus, currently you must specify the values separately in the constraint file; you cannot do it in the source code. See [INIT Values, on page 802](#) and [RLOC Constraints, on page 803](#) for details.

Register Preservation

The `xc_props` attribute also worked as a `syn_preserve` directive, and ensured that the registers on which it was set were not optimized away:

```
module init_attr (output [3:0] rst_cntr_out, input clk);
//Original xc_props attribute
reg [3:0] rst_cntr /* synthesis xc_props="INIT=1" */;
initial rst_cntr = 4'hf;

always @ (posedge clk)
if (!rst_cntr)
    rst_cntr <= rst_cntr - 1'b1;
    assign rst_cntr_out = rst_cntr;
endmodule
```

With the new synthesis syntax, registers can be optimized. This is because the synthesis tool does not use the `INIT="1"` attribute, but merely forwards the attribute in the EDIF netlist. In the following example, the register `rst_cntr[3:0]` is optimized away by the synthesis tool because the tool assumes that `rst_cntr[3:0]` initializes to all zeroes, so the expression conditional-if (`!rst_cntr`) is never executed:

```
module init_attr (output [3:0] rst_cntr_out, input clk);
//New synthesis attribute
reg [3:0] rst_cntr /* synthesis INIT="1" */;
always @ (posedge clk)
if (!rst_cntr)
    rst_cntr <= rst_cntr - 1'b1;
    assign rst_cntr_out = rst_cntr;
endmodule
```

Verilog INIT Example with syn_preserve

To ensure that you keep the register, you must explicitly set the `syn_preserve` directive, as shown in this Verilog example:

```
module init_attr (output [3:0] rst_cntr_out, input clk);
//New synthesis attribute
reg [3:0] rst_cntr /* synthesis INIT="1" syn_preserve=1 */;
always @ (posedge clk)
if (!rst_cntr)
    rst_cntr <= rst_cntr - 1'b1;
    assign rst_cntr_out = rst_cntr;
endmodule
```

VHDL INIT Example with syn_preserve

This is a similar VHDL example with an explicit `syn_preserve` directive:

```

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

entity init_attr is
    port (clk:in std_logic;
          rst_cntr_out: out std_logic_vector(3 downto 0));
end init_attr;

architecture behave of init_attr is
signal rst_cntr: std_logic_vector(3 downto 0);
attribute INIT: string;
attribute INIT of rst_cntr : signal is "1";
attribute syn_preserve : boolean;
attribute syn_preserve of rst_cntr : signal is true;

begin
    process (clk)
    begin
        if rising_edge(clk) then
            if (rst_cntr /= 0) then
                rst_cntr <= rst_cntr-1;
            end if;
        end if;
    end process;
    rst_cntr_out <= rst_cntr;
end behave;
```

Constraint File and syn_preserve

Even if you specify the register `INIT` values in the constraint file as shown below (`INIT="1"`), you must still include the `syn_preserve` directive in the source code to ensure that the register is retained during optimization.

```
define_attribute {rst_cntr[3:0]} INIT {"1"}
```

This value is then embedded in the EDIF netlist:

```
(instance (rename rst_cntr_0 "rst_cntr[0]") (viewRef PRIM
    (cellRef FD (libraryRef UNILIB)))
    (property INIT (string "1"))
)
```

INIT Values

You can use the synthesis syntax to pass INIT values. However, Xilinx ISE 8.2sp3 and later versions require that the INIT value be a string rather than an integer. This means that you must specify the INIT value in quotes:

Constraint define_attribute {i:rst_cntr} INIT {"1"}

Verilog reg [3:0] rst_cntr /* synthesis INIT="1" */;

VHDL attribute INIT: string;
 attribute INIT of rst_cntr : signal is "1";

Currently, if you want to specify different synthesis attributes (such as INIT values) for each register bit on a bus, you must use the constraint file. You cannot do it in the source code. Note that for register INIT, if you want to retain the register during optimization, you must still specify syn_preserve in the source code, because syn_preserve is a directive. So, you set syn_preserve only in the source code and specify different INIT values for each register bit on a bus in the constraint file. The following Verilog example shows syn_preserve set in the source code:

```
module init_attr (output [3:0] rst_cntr_out, input clk);
    reg [3:0] rst_cntr /* synthesis syn_preserve=1 */;
    always @(posedge clk)
        if (!rst_cntr)
            rst_cntr <= rst_cntr - 1'b1;
    assign rst_cntr_out = rst_cntr;
endmodule

module top(input clk, output [3:0] rst_cntr_outver, output [3:0]
            rst_cntr_outvhdl);
    init_attr init_attrver (rst_cntr_outver, clk);
endmodule
```

To specify different bit values for this register, you must include constraints in the constraint file, as shown below. The i: prefix in the example indicates that it is an instance, and the period is the hierarchical separator. The module init_attrver is instantiated within the top-level module that contains the rst_cntr[3:0] register. You need not name the top-level module in the hierarchical path. If the instance is in the top-level module, you do not have to specify a hierarchical path. The example specifies different bit values for the bus. Each value is enclosed in quotes so that it is passed to the Xilinx ISE tool as a string value, as required by that tool.

```
define_attribute {i:init_attrver.rst_cntr[0]} INIT {"0"}  
define_attribute {i:init_attrver.rst_cntr[1]} INIT {"1"}  
define_attribute {i:init_attrver.rst_cntr[2]} INIT {"0"}  
define_attribute {i:init_attrver.rst_cntr[3]} INIT {"1"}
```

The tool forward-annotates INIT values as is, so it is up to you to ensure that the values are in the right binary format for place-and-route.

RLOC Constraints

This section describes how to pass RLOC and RLOC_ORIGIN attributes to the Xilinx P&R tool.

VHDL RLOC Example

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity attr_vhdl is  
    port (clk, d: in std_logic;  
          q_out: out std_logic);  
end attr_vhdl;  
  
architecture behave of attr_vhdl is  
signal q: std_logic;  
attribute RLOC_ORIGIN : string;  
attribute RLOC_ORIGIN of behave : architecture is "X0Y2";  
attribute RLOC : string;  
attribute RLOC of q : signal is "X0Y0";  
attribute syn_useioff : boolean;  
attribute syn_useioff of q : signal is false;  
  
begin  
    process (clk)  
    begin  
        if rising_edge(clk) then  
            q <= d;  
        end if;  
    end process;  
  
    q_out <= q;  
end behave;
```

Verilog RLOC Example

```
module attr (output q, input clk, input d);
  reg FOO_0 /* synthesis RLOC_ORIGIN="X0Y2" RLOC="X0Y0"
    syn_useioff = 0 */;

  always @ (posedge clk)
    FOO_0 <= d;
  assign q = FOO_0;
endmodule
```

As with the INIT values described in [INIT Values, on page 802](#), you cannot apply the individual RLOC values to a bus using a single RLOC synthesis attribute in the source code:

```
reg [3:0] tmp /* synthesis RLOC="X3Y0 X2Y0 X1Y0 X0Y0" */;
```

This causes the following error in the ISE tool:

```
ERROR:Map:6 - Bad format for RLOC constraint "X3Y0 X2Y0 X1Y0
X0Y0" on FD
```

Instead, specify RLOC_ORIGIN for the Verilog module or VHDL architecture, and then specify individual RLOCs for the registers in the constraint file. The following Verilog example and the subsequent constraints illustrate this technique.

```
module attr (clk, di, do) /* synthesis RLOC_ORIGIN="X5Y0" */;
  input clk;
  input [3:0] di;
  output [3:0] do;
  reg [3:0] tmp /* synthesis syn_useioff = 0 */;

  always @ (posedge clk)
    begin
      tmp <= di;
    end
  assign do = tmp;
endmodule
```

You then define the RLOCs for the individual registers in the constraint file as follows:

```
define_attribute {i:tmp[0]} RLOC {"X3Y0"}
define_attribute {i:tmp[1]} RLOC {"X2Y0"}
define_attribute {i:tmp[2]} RLOC {"X1Y0"}
define_attribute {i:tmp[3]} RLOC {"X0Y0"}
```

xc_pullup/xc_pulldown

Attribute

Specifies that a port is either a pullup or pulldown.

Vendor	Devices
Xilinx	Virtex-7, Virtex-6, Virtex-5, and older families

xc_pullup Values

Default	Global	Object
1	No	Port

Value	Description
0	Specifies that a port is not a pullup.
1	Specifies that a port is a pullup.

xc_pulldown Values

Default	Global	Object
1	No	Port

Value	Description
0	Specifies that a port is not a pulldown.
1	Specifies that a port is a pulldown.

Description

Use this attribute to specify that a port is either a pullup or pulldown.

Syntax Specification

SCOPE	<code>define_attribute {portName} xc_pullup {1}</code> <code>define_attribute {portName} xc_pulldown {1}</code>
Verilog	<code>object /* synthesis xc_pulldown = 1 */;</code> <code>object /* synthesis xc_pullup = 1 */;</code>
VHDL	<code>attribute xc_pulldown of object : objectType is true;</code> <code>attribute xc_pullup of object : objectType is true;</code>

SCOPE Example

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description
1	<input checked="" type="checkbox"/>	port	p:A[7:0]	xc_pullup	1	boolean	Add a pullup
2	<input checked="" type="checkbox"/>	port	p:P[16:0]	xc_pulldown	1	boolean	Add a pulldown

For example, you can apply the `xc_pullup`/`xc_pulldown` attribute as follows:

- On a multi-bit port such as RXD[3:0]

```
define_attribute {RXD[3:0]} xc_pullup {1}
```

- On a single-bit port such as RXA0

```
define_attribute {RXA0} xc_pullup {1}
```

- If you want to define this attribute on the n^{th} bit of the multi-bit port RXD[3:0], then you must separate out the output port RXD[n]. For example, apply the attribute on the single-bit port RXD_ n .

```
define_attribute {RXD_1} xc_pullup {1}
```

Otherwise, you cannot define this attribute on a single bit of a multi-bit port.

Verilog Example

```
module pullup_down (clk, A,B,PC,P);
  input clk;
  input [7:0] A /* synthesis syn_pullup = 1 */;
  input [7:0] B, PC;
  output reg [16:0] P /* synthesis syn_pulldown = 1 */;
  reg [7:0] a_d, b_d;
  reg [16:0] m;
  always @ (posedge clk) begin
    a_d <= A;
    b_d <= B;
    m   <= a_d * b_d;
    P   <= m + PC;
  end
endmodule
```

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity dsp_style is
port (clk : in std_logic;
      A : in std_logic_vector(7 downto 0);
      B : in std_logic_vector(7 downto 0);
      PC : in std_logic_vector(7 downto 0);
      P : out std_logic_vector(15 downto 0));
end dsp_style;

architecture rtl of dsp_style is
signal m : std_logic_vector(15 downto 0);
attribute syn_pullup : boolean;
attribute syn_pullup of A : signal is true;
attribute syn_pulldown : boolean;
attribute syn_pulldown of P : signal is true;
begin
process(clk)
begin
  if (clk'event and clk = '1') then
    m <= A * B;
    P <= m + PC;
```

```
:  
    end if;  
end process;  
end rtl;
```

Effect of Using xc_pullup/xc_pulldown

Before applying the attribute:

Verilog	input [7:0] A /* synthesis syn_pullup = 0 */; output reg [16:0] P /* synthesis syn_pulldown = 0 */;
---------	--

VHDL	attribute syn_pullup of A : signal is false; attribute syn_pulldown of P : signal is false;
------	--

Log File

```

Mapper Initialization Complete (Time elapsed 0h:00m:00s; Memory used current: 49MB peak: 49MB)
-----+
Adding property xc_pullup, value 0, to port A[7:0]
Adding property xc_pulldown, value 0, to port P[16:0]
-----+
Start loading timing files (Time elapsed 0h:00m:00s; Memory used current: 49MB peak: 50MB)

```

Net List

```

(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell pullup_down (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port (array (rename A "A[7:0]") 8) (direction INPUT)
          (property xc_pullup (integer 0))
        )
        (port (array (rename B "B[7:0]") 8) (direction INPUT))
        (port (array (rename PC "PC[7:0]") 8) (direction INPUT))
        (port (array (rename P "P[16:0]") 17) (direction OUTPUT)
          (property xc_pulldown (integer 0))
        )
        (port clk (direction INPUT)
      )
    )
  )
)
```

P&R Files

When P&R is run, there is no syn_pullup and syn_pulldown attributes in the following files:

```

<projectdirectory>\rev_1\pr_1\pullup_down.pad(262):P7|A[0]|IOB|IO_L9N_CC_GC_4|INPUT|LVCMS25*|4|||
NONE||UNLOCATED|NO|NONE|
(318): U9|P[4]|IOB|IO_L2N_GC_D10_4|OUTPUT|LVCMS25*|4|12|SLOW||||UNLOCATED|NO|NONE|
<projectdirectory>\rev_1\pr_1\pullup_down_pad.txt(263):|P7 |A[0] |IOB |IO_L9N_CC_GC_4 | INPUT |
LVCMS25*|4 || ||NONE| UNLOCATED |NO |NONE |
(288):R14 |P[0]|IOB|IO_L14P_17_|OUTPUT_|LVCMS25*...|17_|12_|SLOW_|_|_|_|UNLOCATED |NO |NONE

```

After applying the attribute:

Verilog input [7:0] A /* synthesis syn_pullup = 1 */;
 output reg [16:0] P /* synthesis syn_pulldown = 1 */;

VHDL attribute syn_pullup of A : signal is true;
 attribute syn_pulldown of P : signal is true;

xc_rloc

Attribute

Specifies the relative locations for component instances.

Vendor	Technology
Xilinx	Virtex

Specifies the relative locations for component instances. This attribute is used in conjunction with [xc_uset](#) and [xc_map](#). Use this attribute to specify locations for components grouped together using xc_uset. See [Xilinx I/O Support, on page 868](#) for information.

xc_rloc Syntax

Global Support	Object
No	Instance

The following table summarizes the syntax in different files:

FDC	define_attribute {designName} xc_rloc {instanceName}	FDC Example
Verilog	object /* synthesis xc_rloc = "instanceName" */;	Verilog Example
VHDL	attribute xc_rloc of object : objectType is "instanceName";	VHDL Example

:

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	i:x03	xc_uset	SET1	string	Assign group name for placement, use with xc_rloc
2	<input checked="" type="checkbox"/>	<any>	i:x03	xc_rloc	R0C0.f	string	Relative placement specification, use with xc_uset

```
define_attribute {x03} xc_uset {SET1}
define_attribute {x03} xc_rloc {R0C0.f}

define_attribute {x47} xc_uset {SET1}
define_attribute {x47} xc_rloc {R0C0.g}

define_attribute {zz} xc_uset {SET1}
define_attribute {zz} xc_rloc {R0C0.h}
```

Verilog Example

```
module fmap_xor4 (z, a, b, c, d) /* synthesis xc_map=fmap */;
  output z;
  input a, b, c, d;
  assign z = a ^ b ^ c ^ d;
endmodule

module hmap_xor3 (z, a, b, c) /* synthesis xc_map=hmap */;
  output z;
  input a, b, c;
  assign z = a ^ b ^ c;
endmodule

module clb_xor9 (z, a);
  output z;
  input [8:0] a;
  wire z03, z47;
  fmap_xor4 x03 /* synthesis xc_uset="SET1" xc_rloc="R0C0.f" */
    (z03, a[0], a[1], a[2], a[3]);
  hmap_xor3 zz /* synthesis xc_uset="SET1" xc_rloc="R0C0.h" */
    (z, z47, a[4], a[5]);
  fmap_xor4 x47 /* synthesis xc_uset="SET1" xc_rloc="R0C0.g" */
    (z47, z03, a[5], a[6], a[7]);
endmodule
```

```
module xor9top (z, a);
output z;
input [8:0] a;
clb_xor9 x (z, a);
endmodule
```

VHDL Example

```
--hmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity hmap_xor4 is
port (a, b, c, d : in std_logic;
      z : out std_logic);
end hmap_xor4;
architecture rtl of hmap_xor4 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "hmap";
begin
  z <= a xor b xor c xor d;
end rtl;

-- fmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity fmap_xor3 is
port (a, b, c : in std_logic;
      z : out std_logic);
end fmap_xor3;

architecture rtl of fmap_xor3 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "fmap";
begin
  z <= (a and b) xor c;
end rtl;

-- hmap & fmap instantiations
library IEEE;
use IEEE.std_logic_1164.all;

entity clb_xor9 is
port (a : in std_logic_vector(8 downto 0);
```

```
z : out std_logic
);
end clb_xor9;

architecture rtl of clb_xor9 is
    signal z03, z47 : std_logic;
component fmap_xor3
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          z : out std_logic
        );
end component;
component hmap_xor4
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          d : in std_logic;
          z : out std_logic
        );
end component;
attribute xc_uset : string;
attribute xc_rloc : string;
attribute xc_uset of x03 : label is "SET1";
attribute xc_rloc of x03 : label is "R0C0.f";
attribute xc_uset of x47 : label is "SET1";
attribute xc_rloc of x47 : label is "R0C0.g";
attribute xc_uset of zz : label is "SET1";
attribute xc_rloc of zz : label is "R0C0.h";
begin
    x03 : hmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
    x47 : hmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
    zz : fmap_xor3 port map(z03, z47, a(8), z);
end rtl;

--Top level entity definition
library IEEE;
use IEEE.std_logic_1164.all;

entity top is
port (a : in std_logic_vector(8 downto 0);
      z : out std_logic
    );
end top;

architecture rtl of top is
component clb_xor9
```

```
port (a : in std_logic_vector (8 downto 0);
      z : out std_logic);
end component;
begin
  U1: clb_xor9 port map (a, z);
end rtl;
```

Effect of Using xc_rloc

After applying the attribute, the output constraint is forward-annotated to the output `edif` netlist:

```
(instance x03 (viewRef PRIM (cellRef LUT4 (libraryRef VIRTEX)))
  (property RLOC (string "R0C0.f"))
  (property HU_SET (string "x$SET1"))
  (property INIT (string "16'h6996"))
```


xc_slow

Attribute

Specifies that the transition time of the driver of an output port is slow (the default).

This attribute is the same as the `syn_slow` attribute. See [syn_slow](#), on page 619.

xc_use_keep_hierarchy

Attribute

Synplify Pro, Synplify Premier

Preserves the hierarchy of the marked module for diagnostic purposes and timing simulation.

Vendor	Technology
Xilinx	Virtex-7 and older families

xc_use_keep_hierarchy Values

Value	Description
0 (Default)	The software does not add the Xilinx attribute (KEEP_HIERARCHY (string "TRUE")) in the EDIF file for all marked compile points and hierarchical blocks.
1	The software adds the Xilinx attribute (KEEP_HIERARCHY (string "TRUE")) in the EDIF file for all marked compile points and hierarchical blocks.

Description

This attribute lets you preserve the hierarchy of a marked module for diagnostic purposes and timing simulation. The marked modules are not flattened in the place-and-route tool, and their hierarchy is preserved.

When active, the software adds the Xilinx attribute `KEEP_HIERARCHY=TRUE` in the EDIF file for all marked compile points and hierarchical blocks.

xc_use_keep_hierarchy Implementation Details

- Make sure that all blocks with this attribute are registered.

- You must use `xc_use_keep_hierarchy` together with the `syn_hier` attribute and a value of "hard" ([syn_hier, on page 319](#)) to preserve the hierarchy of the module.
- Use this attribute in conjunction with the `xc_area_group` compile point attribute ([xc_area_group, on page 753](#)).
- When this attribute is applied, the synthesis tool does not do boundary optimizations, so use the attribute with caution, because it can compromise design performance.
- When specified as a global attribute, all compile points in the design are marked (KEEP_HIERARCHY=TRUE) as independent hierarchical modules.

xc_use_keep_hierarchy Syntax

Global Support Object

Yes	View
-----	------

The following table summarizes the syntax in different files:

FDC	<code>define_global_attribute {xc_use_keep_hierarchy} {1 0}</code>	SCOPE Example
Verilog	<code>object /* synthesis xc_use_keep_hierarchy = 1 0 */</code>	Verilog Example
VHDL	<code>attribute xc_use_keep_hierarchy of all : architecture is true false;</code> <code>attribute <syn_hier of all : architecture is "hard";</code>	VHDL Example

SCOPE Example

The following SCOPE example shows that the `xc_use_keep_hierarchy` attribute must be used in conjunction with the `syn_hier` attribute.

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	view	<Global>	<code>xc_use_keep_hierarchy</code>	1	boolean	Preserves the hierarchy of the marked
2	<input checked="" type="checkbox"/>	view	vwork.mod10	<code>syn_hier</code>	hard	string	Control hierarchy flattening

The Tcl equivalent commands for these attributes are shown below:

```
define_global_attribute {xc_use_keep_hierarchy} {1}
```

```
define_attribute {v:work.mod10} {syn_hier} {hard}
```

Verilog Example

```
module keep_hier (a,b,c,clk,out1,out2)
    /* xc_use_keep_hierarchy= 1 */;
    input a,b,c,clk;
    output reg out1,out2;
    mod10 inst1 (.clk(clk),.c(c),.d(out2));

    always @ (posedge clk)
        out1 <= a + b;
    endmodule

module mod10 (clk,c,d) /* synthesis syn_hier = "hard" */;
    input c,clk;
    output reg d;

    always @ (posedge clk)
        d <= c;

endmodule
```

VHDL Example

```
library IEEE;
use ieee.std_logic_1164.all;
entity test is
port (a : in std_logic;
      b : in std_logic;
      c : in std_logic;
      clk : in std_logic;
      out1 : out std_logic;
      out2 : out std_logic);
end test;
architecture arch of test is

attribute syn_use_keep_hierarchy : boolean;
attribute syn_use_keep_hierarchy of all : architecture is true;

component test1
port (c : in std_logic;
      clk : in std_logic;
      d : out std_logic);
end component;
```

```

begin
  inst1 : test1 port map (c =>c, d=>out2,clk=>clk);
  process (clk)
  begin
    if (clk'event and clk ='1')then
      out1 <= a and b;
    end if;
  end process;
end arch;

library IEEE;
use ieee.std_logic_1164.all;

entity test1 is
port (c : in std_logic;
      clk : in std_logic;
      d : out std_logic);
end test1;

architecture asrch of test1 is

attribute syn_hier : string;
attribute syn_hier of all : architecture is "hard";

begin
  process (clk)
  begin
    if (clk'event and clk ='1')then
      d <= c;
    end if;
  end process;
end arch;

```

Effects of Using xc_use_keep_hierarchy

This example shows the contents of the edf file before applying the xc_use_keep_hierarchy attribute.

Verilog module keep_hier (a, b, c, clk, out1,out2)
/* synthesis xc_use_keep_hierarchy = 0 */;
module mod10 (clk, c, d) /*synthesis syn_hier = "hard";

VHDL attribute xc_use_keep_hierarchy of all : architecture is false;
attribute syn_hier of all : architecture is "hard";

```

(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell hier (cellType GENERIC)
    (view verilog (viewType NELIST)
      (interface
        (port clk (direction INPUT))
        (port c (direction INPUT))
        (port d (direction OUTPUT)))
      )
      (contents
        (instance d (viewRef PRIM (cellRef FD (libraryRef UNILIB)))
          )
        (net clk (joined
          (portRef clk)
          (portRef C (instanceRef d))
        )))
        (net c (joined
          (portRef c)
          (portRef D (instanceRef d))
        )))
        (net (rename dZ0 "d") (joined
          (portRef Q (instanceRef d))
          (portRef d)
        )))
      )
    )
  )
)

```

This example shows the contents of the edf file after applying the `xc_use_hierarchy` attribute.

```
Verilog      module keep_hier (a, b, c, clk, out1, out2)
/* synthesis xc_use_keep_hierarchy = 1 */;
module mod10 (clk, c, d) /*synthesis syn_hier = "hard",
```

```
VHDL      attribute xc_use_keep_hierarchy of all : architecture is true;  
          attribute syn_hier of all : architecture is "hard";
```

```
(library work
  (edifLevel 0)
  (technology (numberDefinition ))
  (cell hier (cellType GENERIC)
    (view verilog (viewType NETLIST)
      (interface
        (port clk (direction INPUT))
        (port c (direction INPUT))
        (port d (direction OUTPUT))
      )
      (contents
        (instance d (viewRef PRIM (cellRef FD (libraryRef UNILIB)))
        )
        (net clk (joined
          (portRef clk)
          (portRef C (instanceRef d))
        )))
        (net c (joined
          (portRef c)
          (portRef D (instanceRef d))
        )))
        (net (rename dZ0 "d") (joined
          (portRef Q (instanceRef d))
          (portRef d)
        )))
      )
      (property KEEP_HIERARCHY (string "TRUE"))
    )
  )
)
```

xc_use_timespec_for_io

Attribute

Uses the TIMESPEC FROM ...TO command for writing flop-to-out and in-to-flop I/O constraints to the Xilinx constraint (`ncf`) file.

Vendor	Technology
Xilinx	Virtex family Spartan family

Xilinx	Virtex family Spartan family
--------	---------------------------------

Description

By default, the Xilinx OFFSET keyword is used for writing flop-to-out and in-to-flop I/O constraints to the Xilinx constraint (`ncf`) file for forward annotation to placement and routing. To use the TIMESPEC FROM ... TO command instead, you globally assign the `xc_use_timespec_for_io` attribute a value of 1. The default value of the attribute is 0 (use OFFSET).

When TIMESPEC FROM ... TO is used, only the data delay is taken into account by the Xilinx place-and-route tool; when OFFSET is used, the clock arrival time and clock delay are also taken into account. For more information on the behavior of these keywords, refer to the appropriate Xilinx documentation.

Note: When the `xc_use_timespec_for_io` attribute is enabled (1), there is no relaxation of constraints that are forward-annotated to the `ncf` file.

xc_use_timespec_for_io Syntax

Global Support Object

No	Module or architecture
----	------------------------

The following table summarizes the syntax in different files:

FDC	define_global_attribute xc_use_timestep_for_io {0 1}	FDC Example
Verilog	object /* synthesis xc_use_timestep_for_io = 0 1 */;	Verilog Example
VHDL	attribute xc_use_timestep_for_io of object : objectType is false true;	VHDL Example

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	<Global>	xc_use_timestep_for_io	1	boolean	Enable use of from-to timestep instead of offset for I/O constraint

```
define_global_attribute xc_use_timestep_for_io {1}
```

Verilog Example

```
module fmap_xor4 (z, a, b, c, d) /* synthesis xc_map=fmap */;
output z;
input a, b, c, d;
assign z = a ^ b ^ c ^ d;
endmodule

module hmap_xor3 (z, a, b, c) /* synthesis xc_map=hmap */;
output z;
input a, b, c;
assign z = a ^ b ^ c;
endmodule

module clb_xor9 (z, a);
output z;
input [8:0] a;
wire z03, z47;
fmap_xor4 x03 /* synthesis xc_uset="SET1" xc_rloc="R0C0.f" */
(z03, a[0], a[1], a[2], a[3]);
hmap_xor3 zz /* synthesis xc_uset="SET1" xc_rloc="R0C0.h" */
(z, z47, a[4], a[5]);
fmap_xor4 x47 /* synthesis xc_uset="SET1" xc_rloc="R0C0.g" */
(z47, z03, a[5], a[6], a[7]);
endmodule
```

```
module xor9top (z, a);
output z;
input [8:0] a;
  clb_xor9 x (z, a);
endmodule
```

VHDL Example

```
--hmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity hmap_xor4 is
port (a, b, c, d : in std_logic;
z : out std_logic);
end hmap_xor4;
architecture rtl of hmap_xor4 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "hmap";
begin
  z <= a xor b xor c xor d;
end rtl;

-- fmap entity definition
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity fmap_xor3 is
port (a, b, c : in std_logic;
z : out std_logic);
end fmap_xor3;

architecture rtl of fmap_xor3 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "fmap";
begin
  z <= (a and b) xor c;
end rtl;

-- hmap & fmap instantiations
library IEEE;
use IEEE.std_logic_1164.all;

entity clb_xor9 is
```

```
port (a : in std_logic_vector(8 downto 0);
      z : out std_logic
    );
end clb_xor9;

architecture rtl of clb_xor9 is
begin
  signal z03, z47 : std_logic;
  component fmap_xor3
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          z : out std_logic
        );
  end component;

  component hmap_xor4
    port (a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          d : in std_logic;
          z : out std_logic
        );
  end component;
  attribute xc_uset : string;
  attribute xc_rloc : string;
  attribute xc_uset of x03 : label is "SET1";
  attribute xc_rloc of x03 : label is "R0C0.f";
  attribute xc_uset of x47 : label is "SET1";
  attribute xc_rloc of x47 : label is "R0C0.g";
  attribute xc_uset of zz : label is "SET1";
  attribute xc_rloc of zz : label is "R0C0.h";
begin
  x03 : hmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
  x47 : hmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
  zz : fmap_xor3 port map(z03, z47, a(8), z);
end rtl;

--Top level entity definition
library IEEE;
use IEEE.std_logic_1164.all;

entity top is
port (a : in std_logic_vector(8 downto 0);
      z : out std_logic
    );
end top;
```

```
architecture rtl of top is
component clb_xor9
    port (a : in std_logic_vector (8 downto 0);
          z : out std_logic);
end component;
begin
    U1: clb_xor9 port map (a, z);
end rtl;
```

xc_uset

Attribute

Assigns a group name to component instances.

Vendor	Technology
Xilinx	Virtex families

Description

Assigns a group name to component instances. This attribute is used with the [xc_rloc](#) and [xc_map](#) attributes to specify a relative location to a group of components instances. See [Xilinx I/O Support, on page 868](#), for more information.

xc_uset Syntax

Global Support Object

No	Instance
----	----------

The following table summarizes the syntax in different files:

FDC	define_attribute {instanceName} xc_uset {groupName}	FDC Example
Verilog	object /* synthesis xc_uset = "groupName" */;	Verilog Example
VHDL	attribute xc_uset of object : objectType is "groupName";	VHDL Example

:

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	i:x03	xc_uset	SET1	string	Assign group name for placement, use with xc_rloc
2	<input checked="" type="checkbox"/>	<any>	i:x03	xc_rloc	ROCO.f	string	Relative placement specification, use with xc_uset

```
define_attribute {x03} xc_uset {SET1};
define_attribute {x03} xc_rloc {ROCO.f};

define_attribute {x44} xc_uset {SET1};
define_attribute {x44} xc_rloc {ROCO.g};

define_attribute {zz} xc_uset {SET1};
define_attribute {zz} xc_rloc {ROCO.h};
```

Verilog Example

```
module clb_xor9(z, a);
  output z;
  input [8:0] a;
  wire z03, z47;
  fmap_xor4 x03 /* synthesis xc_uset = "SET1" xc_rloc = "ROCO.f" */
    (z03, a[0], a[1], a[2], a[3]);
  fmap_xor4 x47 /* synthesis xc_uset = "SET1" xc_rloc = "ROCO.g" */
    (z47, a[4], a[5], a[6], a[7]);
  hmap_xor3 zz /* synthesis xc_uset = "SET1" xc_rloc = "ROCO.h" */
    (z, z03, z47, a[8]);
endmodule
```

VHDL Example

```
library synplify;
architecture rtl of clb_xor9 is
  signal z03, z47 : std_logic;
  component hmap_xor3
    port (a, b, c : in std_logic;
          z : out std_logic);
  end component;

  component fmap_xor4
    port (a, b, c : in std_logic;
          z : out std_logic);
  end component;
```

```
attribute xc_uset : string;
attribute xc_rloc : string;
attribute xc_uset : string;
attribute xc_uset of x03 : label is "SET1";
attribute xc_rloc of x03 : label is "R0C0.f";
attribute xc_uset of x47 : label is "SET1";
attribute xc_rloc of x47 : label is "R0C0.g";
attribute xc_uset of zz : label is "SET1";
attribute xc_rloc of zz : label is "R0C0.h";

begin
x03 : fmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
x47 : fmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
zz : hmap_xor3 port map(z03, z47, a(8), z);
end rtl;
```


Index

Symbols

.cdc file
examples 12

A

advanced_synthesis
effect on syn_macro and
syn_user_instance 379
alsloc 31
alspin 35
alspreserve 39
altera_io_powerup attribute 43
altera_logicclock_location attribute 47
altera_logicclock_size attribute 51
attributes
custom 270
global attribute summary 22
specifying in the SCOPE spreadsheet 10
specifying, overview of methods 10
syn_assign_to_region 131
syn_clock_priority 169
syn_direct_reset 195
syn_highrel_iocnector 329
Attributes panel, SCOPE spreadsheet 10

B

black_box directives
black_box_pad_pin 55
black_box_tri_pins 61
syn_black_box 149
syn_force_seq_prim 287
syn_isclock 355
syn_resources 564
syn_tco 651
syn_tpd 657
syn_tristate 663
syn_tsu 675

black boxes

directives. See black box directives
internal startup box (Xilinx) 779
source code directives 150
syn_gatedclk_clock_en directive 301
timing directives 657

black_box_pad_pin directive 55

black_box_tri_pins directive 61

buffers

clock. See clock buffers
global 775
global. See global buffers
output 769

C

case statement
default 75
clock buffers
assigning resources 429
clock enables
inferring with syn_direct_enable 190
net assignment 189
clock priority
syn_clock_priority 169
clocks
on black boxes 355
code
ignoring with pragma translate off/on 95
compile points
allowed resources 105
preserving hierarchy for Xilinx Place
and Route 820
compiler
loop iteration, loop_limit 79
loop iteration, syn_looplmit 373
custom attributes 270

D

DCM clock priority 169
define_attribute
 syntax 11
define_false_path
 using with syn_keep 360
define_global_attribute
 summary 22
 syntax 11
define_input_delay
 forward-annotation with
 syn_forward_io_constraints 291
define_multicycle_path
 using with syn_keep 360
define_output_delay
 forward-annotation with
 syn_forward_io_constraints 291
delays
 forward annotating 291
 forward-annotating with
 syn_forward_io_constraints 291
diff_term attribute 69
DSP (Xilinx) 219, 227
DSP48 (Xilinx)
 converting asynchronous reset
 registers 161, 169, 505

E

edf file. *See* edif file
edif file
 bus name styles 255
 character case 243
 port name length 239
 scalar and array ports 425
 syn_noarrayports attribute 425
enable
 in std_logic_vector (3 downto 0) 667
entity tristate2 is port (input3, input2,
 input1, input0
 in std_logic_vector (7 downto 0) 667
enumerated types
 syn_enum_encoding directive 269

F

fanout limits
 overriding default 393
 syn_maxfan attribute 393
files
 .edf. *See* edif file
fix gated clocks
 syn_gatedclk_clock_en directive 301
forward-annotation
 RLOC 803
 RLOC_ORIGIN 803
 syn_forward_io_constraints attribute
 291
 xc_props attribute 797
FSMs
 syn_encoding attribute 259
full_case directive 73

G

gated clocks
 syn_force_seq_prim directive 287
global attributes summary 22
global buffers 775
 defining 311
 xc_global_buffers attribute 775

H

hierarchy
 flattening with syn_hier 319
 preserving for Xilinx Place and Route
 819
high reliability
 syn_radhardlevel 495

I

I/O buffers
 inserting 351
 specifying I/O standards 459
I/O packing
 disabling with syn_replicate 557
I/O registers, power-up mode (Intel) 43
I/Os

inferring Xilinx buffers with syn_diff_io
183

INIT values
syntax for forward-annotation 802

instances
controlling optimization 733
preserving with syn_noprune 441

Intel
I/O registers power-up mode 43
syn_useioff attribute 717

L

Lattice
loc attribute 77
orca_padtype attribute 83
orca_props attribute 87
pad types (ORCA) 83

library ieee 667

library synplify 667

loc attribute (Lattice) 77

loop_limit directive 79

M

macros
preserving with syn_macro 375

map primitive 789

Microsemi
alsloc attribute 31
alspin attribute 35
alspreserve attribute 39
assigning I/O ports 35
preserving relative placement 31
syn_radhardlevel attribute 495

modules
renaming 129, 555, 691

multicycle paths
syn_reference_clock 546

N

nets
preserving with syn_keep 359

O

OFFSET I/O constraints (Xilinx) 825

orca_padtype attribute (Lattice) 83

orca_props attribute 87

output buffers
selection 769

output drivers
slowing transition times 817
transition time 765

output files
.edf. *See* edif file

P

pad locations
See also pin locations

pad types
Lattice ORCA 83
Xilinx 795

parallel_case directive 91

pin locations
forward annotating 367
Lattice 77
Microsemi 35

pipelining
syn_pipeline attribute 465

port placement 785

pragma translate_off directive 95

pragma translate_on directive 95

priority encoding 91

probes
inserting 487

Q

qout
out std_logic_vector (7 667)

R

radiation effects. *See* high reliability

RAM MMI file
syn_ram_write_mem attribute 505

RAMs
implementation styles 509

- technology support 513
- register packing
 syn_useioff attribute, Intel 717
- registers
 preserving with syn_preserve 471
- relative location
 alsloc (Microsemi) 31
 xc_map (Xilinx) 789
 xc_rloc 811
 xc_uset 831
- replication
 disabling 557
- resource sharing
 syn_sharing directive 607
- retiming
 syn_allow_retim ing attribute 99
- S**
- SCOPE spreadsheet
 Attributes panel 10
- SECDED 295
- sequential optimization, preventing with
 syn_preserve 471
- set/reset priority (Microsemi) 481
- SIMD mode
 using syn_DSPstyle attribute 233
- simulation mismatches
 full_case directive 76
- startup blocks
 Xilinx 779
- state machines
 enumerated types 269
 extracting 623, 643
- syn_allow_retim ing attribute 99
- syn_append_submodules directive 129
- syn_assign_to_region attribute 131
- syn_assign_to_slr attribute 133
- syn_async_reg attribute 137
- syn_auto_insert_bufg attribute 141
- syn_black_box directive 149
- syn_clean_reset attribute 161
- syn_clock_gmux_proxy attribute 165
- syn_clock_priority attribute 169
- syn_connect_hrefs directive 173
- syn_cp_use_fast_synthesis attribute 181
- syn_diff_io attribute 183
- syn_direct_enable attribute 189
- syn_direct_reset Attribute 195
- syn_direct_set attribute 201
- syn_disable_purifyclock attribute 207
- syn_donot_infer_iddr attribute 215
- syn_donot_infer_odd r attribute 211, 215
- syn_DSPstyle 219
- syn_DSPstyle attribute 219
 using SIMD mode 233
- syn_edif_bit_format attribute 255
- syn_edif_name_length attribute 239
- syn_edif_scalar_format attribute 243
- syn_encoding
 compared with syn_enum_encoding
 directive 270
 using with enum_encoding 271
- syn_encoding attribute 259
- syn_enum_encoding
 using with enum_encoding 271
- syn_enum_encoding directive 269
 compared with syn_encoding attribute 270
- syn_fast_auto attribute 275
- syn_force_pads attribute 283
- syn_force_seq_prim directive 287
- syn_forward_io_constraints attribute 291
- syn_fsm_correction 295
- syn_fsm_correction directive 295
- syn_gatedclk_clock_en directive 301
- syn_gatedclk_clock_en_polarity directive 305
- syn_global_buffers
 and xc_global_buffers 312
- syn_global_buffers attribute 311
- syn_hier attribute 319
 and xc_use_keep_hierarchy 820
- syn_highrel_ioconnector attribute 329
- syn_insert_buffer attribute 339
- syn_insert_pad attribute 351

syn_isclock directive 355
syn_keep
 compared with **syn_preserve** and
 syn_noprune directives 361
syn_keep directive 359
syn_loc attribute 367
syn_looplimit directive 373
syn_macro directive 375
syn_map_dffrs attribute 381
syn_max_memsize_reg attribute 387
syn_maxfan attribute 393
syn_modify_ram_contents attribute 505
syn_multstyle attribute 401
syn_netlist_hierarchy attribute 413
syn_no_compile_point attribute 421
syn_noarrayports attribute 425
syn_noclockbuf attribute 429
 using with fanout guides 394
syn_noclockpad attribute 435
syn_noprune directive 441
syn_pad_type attribute 459
syn_preserve
 compared with **syn_keep** and
 syn_noprune 472
syn_preserve directive 471
syn_preserve_sr_priority attribute
 (Microsemi) 481
syn_probe attribute 487
syn_radhardlevel 495
 Intel and Xilinx options 496
 Intel and Xilinx syntax 497
 TMR. *See* TMR, distributed TMR
syn_radhardlevel attribute
 TMR 495
syn_ram_write_mem attribute 505
syn_ramstyle attribute 509
syn_reduce_controlset_size (Achronix)
 539
syn_reduce_controlset_size attribute
 (Lattice, Xilinx) 525
syn_reference_clock attribute 545
syn_register_correction attribute 549
syn_rename_module directive 555
syn_replicate
 using with fanout guides 394
syn_replicate attribute 557
syn_resources attribute 563
syn_romstyle attribute 577, 586
syn_rw_conflict_logic attribute 591
syn_safe_case directive 599
syn_safefsm_pipe directive 603
syn_sharing directive 607
syn_shift_resetphase 613
syn_slow attribute 619
syn_smhigheffort attribute 623
syn_srl_mindepth attribute 629
syn_srlstyle attribute 633
syn_state_machine attribute 623
syn_state_machine directive 643
syn_tco directive 651
syn_tpd directive 657
 black-box timing 657, 675
syn_tristate directive 663
syn_tristatetomux attribute 667
syn_tsu directive 675
 black-box timing 675
syn_unconnected_inputs attribute 681
syn_unique_inst_module directive 691
syn_use_carry_chain attribute 697
syn_useenables attribute 711
syn_useioff attribute
 Intel 717
syn_user_instance
 comparison with **syn_macro** 378
syn_user_instance attribute 733
syn_vote_loops Attribute 739
syn_vote_register Attribute 743
syn_vote_register attribute 743
synthesis_off directive 750
synthesis_on directive 750
SystemVerilog
 ignoring code with **synthesis_off/on** 750

T

TIMESPEC I/O constraint (Xilinx)
 `xc_use_timespec_for_io` attribute [825](#)

timing

`syn_tco` directive [651](#)
 `syn_tpd` directive [657](#)
 `syn_tsu` directive [675](#)

TMR

 Microsemi `syn_radhardlevel` [497](#)
 `syn_radhardlevel` attribute [495](#)

translate_off directive [749](#)

translate_on directive [749](#)

tristates

`black_box_tri_pins` directive [61](#)
 converting drivers to muxes [667](#)
 `syn_tristate` directive [663](#)

U

UCF

 clock priority with `syn_clock_priority`
 [169](#)

use `ieee.std_logic_1164.all` [667](#)

Use OFFSET for I/O constraints option
 [825](#)

use `synplify.attributes.all` [667](#)

V

Verilog

 ignoring code with translate off/on [749](#)
 `syn_keep` on multiple nets [360](#)

W

wires, preserving with `syn_keep` directive
 [359](#)

X

`xc_area_group` (Xilinx) [753](#)
`xc_clockbuftype` attribute [759](#)
`xc_fast` attribute [765](#)
`xc_fast_auto` attribute [769](#)
`xc_global_buffers`
 and `syn_global_buffers` [775](#)

`xc_global_buffers` attribute [775](#)
`xc_isgsr` directive [779](#)
`xc_loc` attribute [785](#)
`xc_map` attribute [789](#)
`xc_padtype` attribute [795](#)
`xc_props` attribute [797](#)
`xc_pulldown` attribute [805](#)
`xc_pullup` attribute [805](#)
`xc_rloc` attribute [811](#)
`xc_slow` attribute [817](#)
`xc_use_keep_hierarchy` attribute [819](#)
`xc_use_timespec_for_io` attribute [825](#)
`xc_uset` attribute [831](#)

Xilinx

 DSP component [219, 227](#)
 forward-annotating properties [797](#)
 inferring I/O buffers with `syn_diff_io`
 [183](#)

 OFFSET for I/O constraints [825](#)

 pad types [795](#)

 TIMESPEC for I/O constraints [825](#)

Xilinx DSP48 register conversion
 [161, 169, 505](#)