

Synopsys Synplify Pro® for GoWin

Attribute Reference

September 2019



Copyright Notice and Proprietary Information

© 2019 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Free and Open-Source Licensing Notices

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

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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

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

Third-Party Links

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

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

September 2019

Contents

Chapter 1: Introduction

How Attributes and Directives are Specified	8
The SCOPE Attributes Tab	8
Summary of Attributes and Directives	11
Attribute and Directive Summary (Alphabetical)	11
Summary of Global Attributes	14

Chapter 2: Attributes and Directives

Attributes and Directives Summary	16
full_case	18
loop_limit	22
parallel_case	24
syn_allow_retiming	27
syn_area_group	30
syn_black_box	34
syn_direct_enable	37
syn_direct_reset	41
syn_direct_set	44
syn_dspstyle	49
syn_encoding	52
syn_hier	59
syn_insert_pad	70
syn_keep	73
syn_loc	78
syn_looplmit	81
syn_macro	83
syn_maxfan	86
syn_netlist_hierarchy	90
syn_noprune	94
syn_pad_type	97
syn_pipeline	101

syn_preserve	104
syn_probe	107
syn_pullup/syn_pulldown	111
syn_ramstyle	115
syn_replicate	121
syn_romstyle	125
syn_smhigheffort	129
syn_srlstyle	134
syn_tlvds_io/syn_elvds_io	136
syn_useenables	140
syn_use_set	144
translate_off/translate_on	146

CHAPTER 1

Introduction

This document describes the attributes and directives available in the synthesis tool. The attributes and directives let you direct the way a design is analyzed, optimized, and mapped during synthesis. Throughout the documentation, features and procedures described apply to all tools, unless specifically stated otherwise.

This chapter includes the following introductory information:

- [How Attributes and Directives are Specified](#), on page 8
- [Summary of Attributes and Directives](#), on page 11
- [Summary of Global Attributes](#), on page 14

How Attributes and Directives are Specified

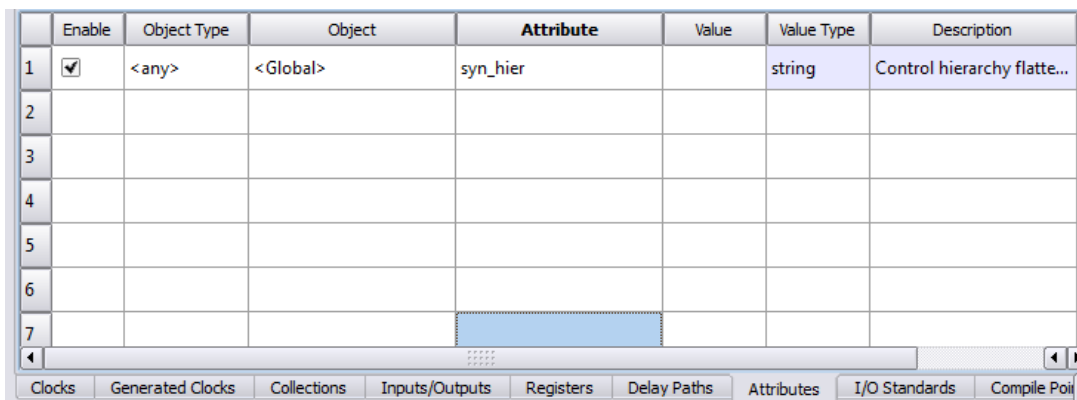
By definition, *attributes* control mapping optimizations and *directives* control compiler optimizations. Because of this difference, directives must be entered directly in the HDL source code or through a compiler design constraint file. Attributes can be entered either in the source code, in the SCOPE 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 90](#) 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 125](#).

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.



	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	<any>	<Global>	syn_hier		string	Control hierarchy flatte...
2							
3							
4							
5							
6							
7							

Navigation tabs at the bottom: Clocks, Generated Clocks, Collections, Inputs/Outputs, Registers, Delay Paths, **Attributes**, I/O Standards, Compile Poi...

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 93](#) 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 34 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 14](#) for more details on specifying global attributes in the synthesis environment.

Summary of Attributes and Directives

The following sections summarize the synthesis attributes and directives:

- [Attribute and Directive Summary \(Alphabetical\)](#), on page 11
- [Chapter 2, *Attributes and Directives*](#)

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

Attribute and Directive Summary (Alphabetical)

The following table summarizes the synthesis attributes and directives. For detailed descriptions of each one, you can find them listed in alphabetical order.

Attribute/Directive	Description
full_case	Specifies that a Verilog case statement has covered all possible cases.
loop_limit	Specifies a loop iteration limit for for loops.
parallel_case	Specifies a parallel multiplexed structure in a Verilog case statement, rather than a priority-encoded structure.
syn_allow_retiming	Determines whether registers may be moved across combinational logic to improve performance in devices that support retiming.
syn_area_group	Specifies the region where the instance should be placed.
syn_black_box	Defines a black box for synthesis.
syn_direct_enable	Assigns clock enable nets to dedicated flip-flop enable pins. It can also be used as a compiler directive that marks flip-flops with clock enables for inference.

Attribute/Directive	Description
<code>syn_direct_reset</code>	Controls the assignment of a net to the dedicated reset pin of a synchronous storage element (flip-flop). Using this attribute, you can direct the mapper to only use a particular net as the reset when the design has a conditional reset on multiple candidates.
<code>syn_direct_set</code>	Controls the assignment of a net to the dedicated set pin of a synchronous storage element (flip-flop). Using this attribute, you can direct the mapper to only use a particular net as the set when the design has a conditional set on multiple candidates.
<code>syn_dspstyle</code>	Determines whether the object is mapped to a technology-specific DSP component.
<code>syn_encoding</code>	Specifies the encoding style for state machines.
<code>syn_hier</code>	Determines hierarchical control across module or component boundaries.
<code>syn_insert_pad</code>	Removes an existing I/O buffer from a port or net when I/O buffer insertion is enabled.
<code>syn_keep</code>	Prevents the internal signal from being removed during synthesis.
<code>syn_loc</code>	Specifies the location (placement) of ports.
<code>syn_looplimit</code>	Specifies a loop iteration limit for while loops.
<code>syn_macro</code>	Determines whether instantiated macros are optimized and included in the final output netlist.
<code>syn_maxfan</code>	Overrides the default fanout guide for an individual input port, net, or register output.

Attribute/Directive	Description
syn_netlist_hierarchy	Controls hierarchy generation in VM output files.
syn_noprune	Controls the automatic removal of instances that have outputs that are not driven.
syn_pad_type	Specifies an I/O buffer standard.
syn_pipeline	Specifies that registers be moved into multipliers and ROMs in order to improve frequency.
syn_preserve	Preserves registers that can be optimized due to redundancy or constraint propagation.
syn_probe	Adds probe points for testing and debugging.
syn_pullup/syn_pulldown	Specifies if a port is a pullup or pulldown.
syn_ramstyle	Determines how RAMs are implemented.
syn_replicate	Controls replication, either globally or on registers.
syn_romstyle	Specifies how inferred ROMs are implemented.
syn_smhigh effort	Overrides the default behavior of seqshift implementation using RAM or registers.
syn_tlvs_io/syn_elvs_io	Controls differential I/O buffer inferencing.
syn_useenables	Generates clock enable pins for registers.
translate_off/translate_on	Specifies sections of code to exclude from synthesis, such as simulation-specific code.

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 [Summary of Attributes and Directives, on page 11](#).

Global Attribute	Can Also Be Set On Design Objects
syn_black_box	x
syn_netlist_hierarchy	
syn_ramstyle	x
syn_replicate	x
syn_romstyle	x

CHAPTER 2

Attributes and Directives

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

For details, see:

- [Attributes and Directives Summary](#), on page 16
- [Summary of Global Attributes](#), on page 14

Attributes and Directives Summary

The following attributes and directives are listed in alphabetical order:

- [full_case](#), on page 18
- [loop_limit](#), on page 22
- [parallel_case](#), on page 24
- [syn_allow_retiming](#), on page 27
- [syn_area_group](#), on page 30
- [syn_black_box](#), on page 34
- [syn_direct_enable](#), on page 37
- [syn_direct_reset](#), on page 41
- [syn_direct_set](#), on page 44
- [syn_dspstyle](#), on page 49
- [syn_encoding](#), on page 52
- [syn_hier](#), on page 59
- [syn_insert_pad](#), on page 70
- [syn_keep](#), on page 73
- [syn_loc](#), on page 78
- [syn_looplimit](#), on page 81
- [syn_macro](#), on page 83
- [syn_maxfan](#), on page 86
- [syn_netlist_hierarchy](#), on page 90
- [syn_noprune](#), on page 94
- [syn_pad_type](#), on page 97
- [syn_pipeline](#), on page 101
- [syn_preserve](#), on page 104
- [syn_probe](#), on page 107
- [syn_pullup/syn_pulldown](#), on page 111
- [syn_ramstyle](#), on page 115

- [syn_replicate](#), on page 121
- [syn_romstyle](#), on page 125
- [syn_smhigh effort](#), on page 129
- [syn_tlvds_io/syn_elvds_io](#), on page 136
- [syn_useenables](#), on page 140
- [syn_use_set](#), on page 144
- [translate_off/translate_on](#), on page 146

full_case

Directive

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

full_case Values

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

Description

For Verilog designs only. When used with a `case`, `casex`, or `casez` statement, this directive indicates that all possible values have been given, and that no additional hardware is needed to preserve signal values.

full_case Values Syntax

This table summarizes the syntax in the following file type:

Verilog	<code>/* synthesis full_case */;</code>	Verilog Example
---------	---	---------------------------------

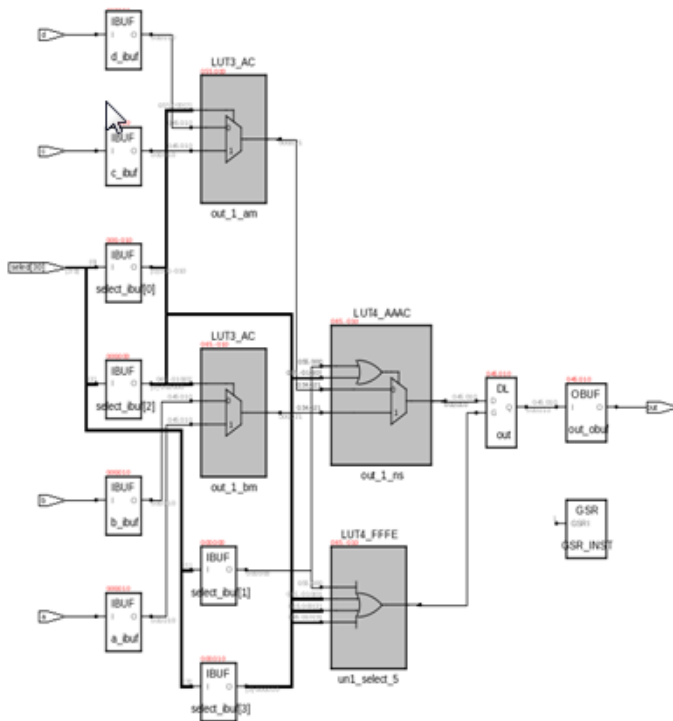
Verilog Example

```
module top (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
```

Effect of Using full_case

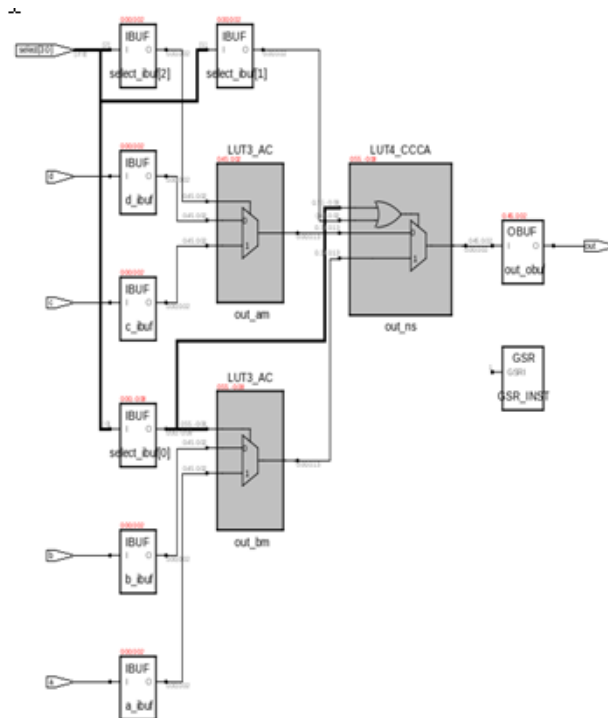
Before applying *full_case*:

The software uses additional hardware to preserve signal values.



After applying *full_case*:

The software does not require additional hardware to preserve signal values when *full_case* directive is used.



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 - N	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 [Verilog Panel, on page 317](#) in the *Command Reference*.

Note: VHDL applications use the `syn_looplimit` directive (see [syn_looplimit, on page 81](#)).

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 *object* /* **synthesis loop_limit** *value* */;

[Verilog Example](#)

Verilog Example

```
module test(din,dout,clk);

parameter WIDE=4000;

input[(WIDE-1) : 0] din;
input clk;
output[(WIDE-1): 0] dout;
reg[(WIDE-1) : 0] dout;
integer i;
always @(posedge clk)
begin
    /* synthesis loop_limit 4000 */
    for(i=0;i<WIDE;i=i+1)
    begin
        dout[i] <= din[i];
    end
end
endmodule
```

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

```
module test (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
```

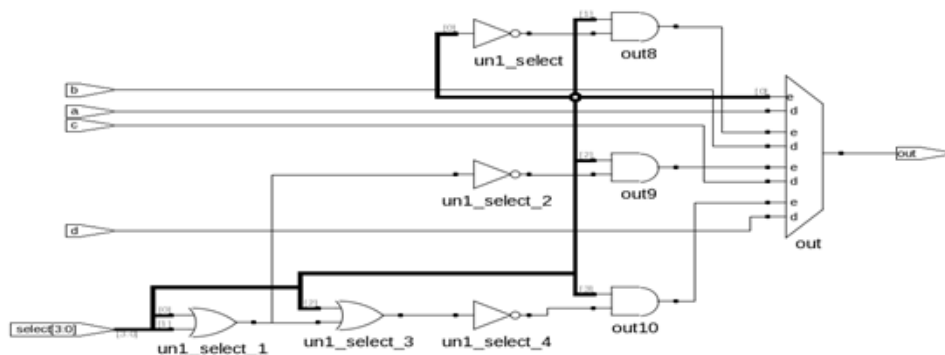
Effect of Using `parallel_case`

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.

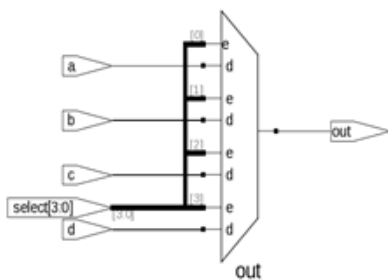
Before applying `parallel_case`:

Extra logic for priority encoding



After applying `parallel_case`:

No extra logic.



syn_allow_retiming

Attribute

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

syn_allow_retiming values

1	Allows registers to be moved during retiming.
0	Does not allow retimed registers to be moved.

Description

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

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

syn_allow_retiming Syntax

Global Object

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

You can specify the attribute in the following files:

FDC	define_attribute {register} syn_allow_retiming {value} define_global_attribute syn_allow_retiming {value}
Verilog	<i>object</i> /* synthesis syn_allow_retiming = value */ ;
VHDL	attribute syn_allow_retiming of object : objectType is value ;

FDC Example

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

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

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

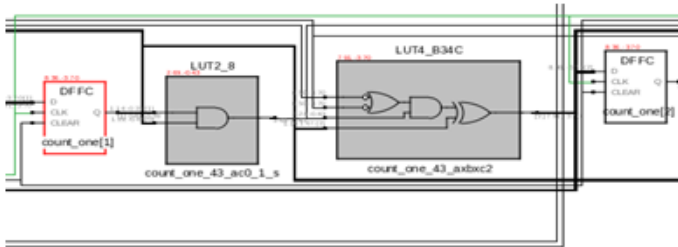
Verilog Example

```
module test (clk,rst,data,count_one);
input clk,rst;
input [20:0]data ;
output reg [3:0]count_one /* synthesis syn_allow_retiming=1*/;
integer i;
reg parity= 1'b1;
always @(posedge clk or posedge rst)
begin
    if (rst)
        count_one <=0;
    else begin
        for (i=0; i<21; i=i+1)
            if (data[i] == parity)
                count_one<=count_one+1;    end
    end
end
endmodule
```

Effect of using syn_allow_retiming

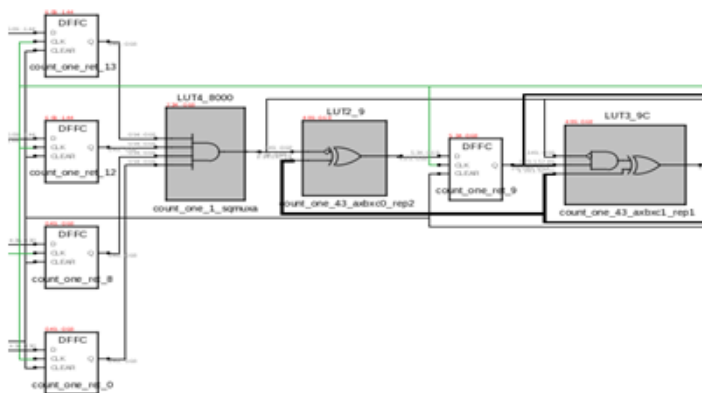
Before applying syn_allow_retiming:

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`:

The critical path and the worst slack for this scenario are shown along with the four `*_ret` retimed registers.



syn_area_group

Attribute/Directive

Specifies the region where the instance should be placed.

syn_area_group Values

Default	Global	Object
N/A	No	View

Description

`syn_area_group` is used to the region where the instance should be placed; by specifying its location and area.

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.

Syntax

FDC `define_attribute {v:module|architectureName}
syn_area_group {topleft:bottomright}` [SCOPE Example](#)

Verilog `/* synthesis syn_area_group =
"R10C29:R16C35" */;` [Verilog Example](#)

VHDL `attribute syn_area_group : string;
attribute syn_area_group of arch: architecture
is "R10C29:R16C35";` [VHDL Example](#)

SCOPE Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	view	v.mod10	syn_area_group	1	string	Specify region where instanc...
2	<input type="checkbox"/>						

```
define_attribute {v:work.mod10} {syn_area_group}
{R10C29:R16C35}
```

Verilog Example

```
module syn_area_group (a,b,c,clk,out1,out2)/* synthesis
syn_area_group = "R10C29: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 syn_area_group : string;
attribute syn_area_group of arch: architecture is
"R10C29: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 `syn_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} {syn_area_group} {R10C29:R16C3}
```

syn_black_box

Directive

Defines a module or component as a black box.

syn_black_box Value

Value	Description
<i>Module architecture Name</i>	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.

syn_black_box Syntax

Global	Object
No	Module architecture name

You can specify the attribute in the following files:

Verilog *object* /* **synthesis syn_black_box** */ ;

VHDL **attribute syn_black_box of** *object* : *objectType* **is 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

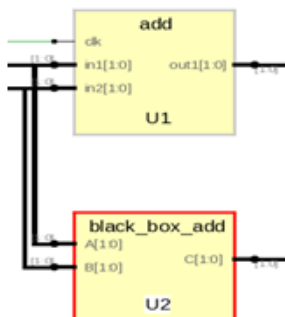
module add (clk, in1, in2, out1);
  input clk;
  input [1:0]in1;
  input [1:0]in2;
  output [1:0]out1;
  reg [1:0]out1;
  always@(posedge clk)
  begin
    out1 <= in1 + in2;
  end
endmodule

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
```

Effect of using syn_black_box

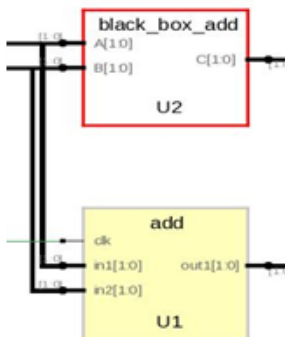
Before applying syn_black_box:

The contents of module black_box_add are visible as shown below.



After applying `syn_black_box`:

The contents of the black box are no longer visible as shown below.



syn_direct_enable

Attribute

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

syn_direct_enable values

1	Enables nets to be assigned to the clock enable pin.
0	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

Global	Object
No	Net Port

You can specify the attribute in the following files:

FDC	define_attribute {<i>object</i>} syn_direct_enable {<i>value</i>}
Verilog	<i>object</i> /* synthesis syn_direct_enable = <i>value</i> */ ;
VHDL	attribute syn_direct_enable of <i>object</i> : <i>objectType</i> is <i>value</i> ;

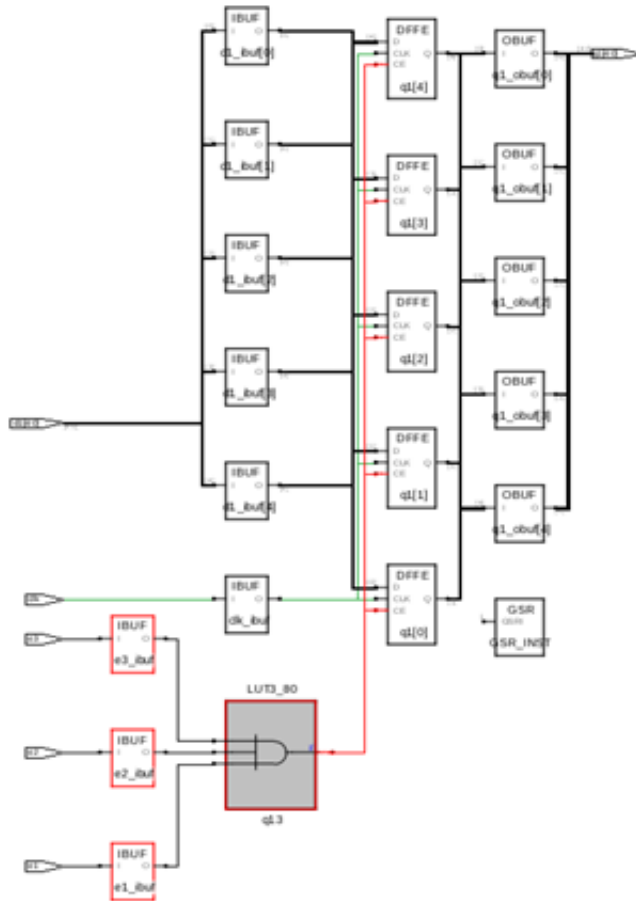
FDC Example

```
module direct_enable(q1, d1, clk, e1, e2, e3);  
  
    parameter size=5;  
  
    input [size-1:0] d1;  
    input clk;  
    input e1,e2;  
    input e3;  
  
    output reg [size-1:0] q1;  
  
    always@(posedge clk)  
        if (e1&e2&e3)  
            q1 = d1;  
  
endmodule  
  
FDC: define_attribute {p:e3} {syn_direct_enable} {1}
```

Effect of Using syn_direct_enable

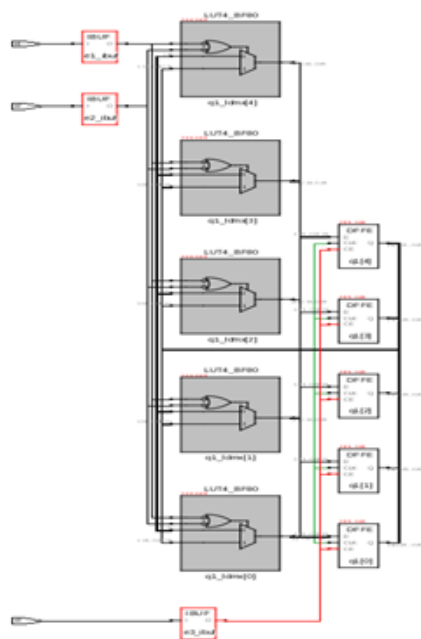
Before applying syn_direct_enable:

The clock enable nets e1, e2, and e3 are connected to the dedicated clock enable pin of the storage element (flip-flop).



After applying `syn_direct_enable`:

The clock enable net e3 is connected to the dedicated clock enable pin of the storage element (flip-flop). The enable nets e1 and e2 are moved to the data path.



syn_direct_reset

Attribute/Directive

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

syn_direct_reset Values

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

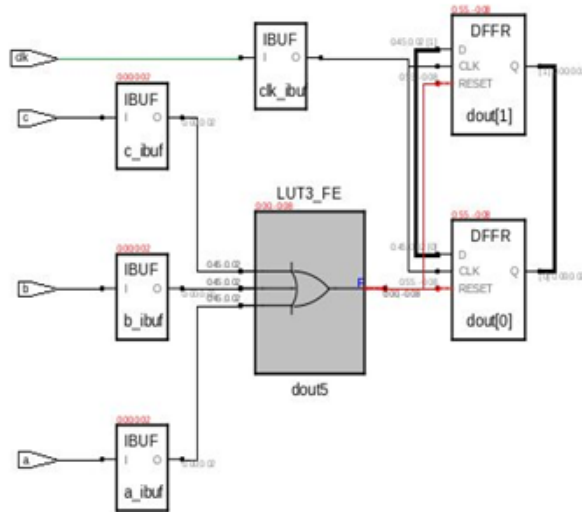
syn_direct_reset Syntax

Global	Object
No	Port

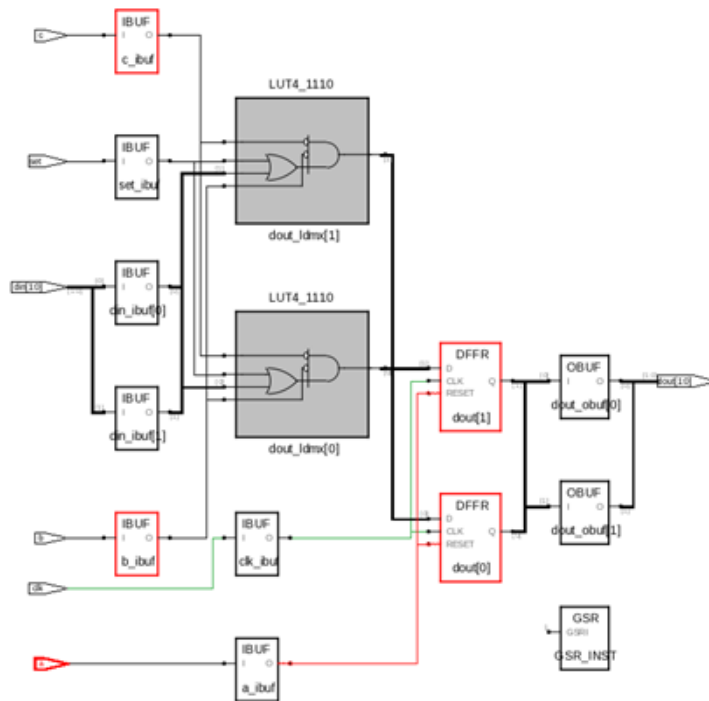
The following table summarizes the syntax in different files:

FDC	define_attribute {object} syn_direct_reset {value}
Verilog	<i>object</i> /* synthesis syn_direct_reset = value */ ;
VHDL	attribute syn_direct_reset of <i>object</i> : <i>objectType</i> is value ;

The software does not assign a net for the design to the dedicated reset pin of a synchronous storage element (flip-flop).



The software assigns a net for the design to the dedicated reset pin of a synchronous storage element (flip-flop). The red color in the schematic below shows the signal a assigned to a dedicated reset pin of the storage element and signals b and c moved to the data path.



syn_direct_set

Attribute/Directive

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

syn_direct_set Values

Value	Description
1	Enables the software to assign a net to the dedicated set pin of a synchronous storage element (flip-flop).
0	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	Port

The following table summarizes the syntax in different files:

FDC	define_attribute { <i>object</i> } syn_direct_set { <i>value</i> }
Verilog	<i>object</i> /* synthesis syn_direct_set = <i>value</i> */
VHDL	attribute syn_direct_set of <i>object</i> : <i>objectType</i> is <i>value</i> ;

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)
);

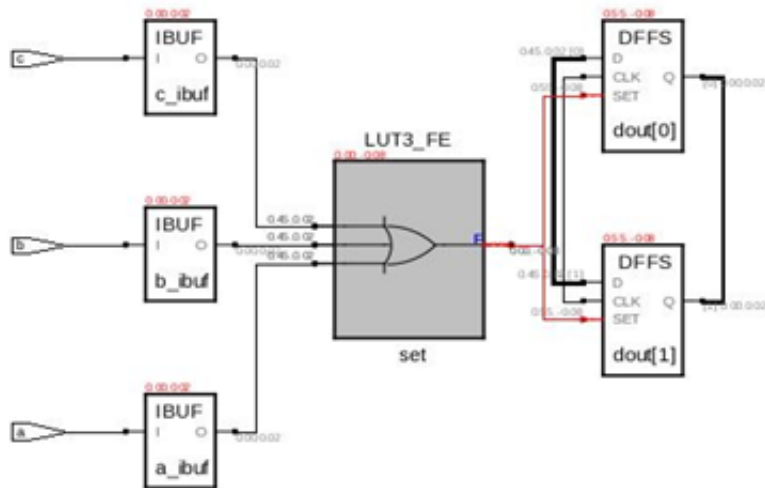
attribute syn_direct_set : boolean;
attribute syn_direct_set of a : signal is true;
end entity test;

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

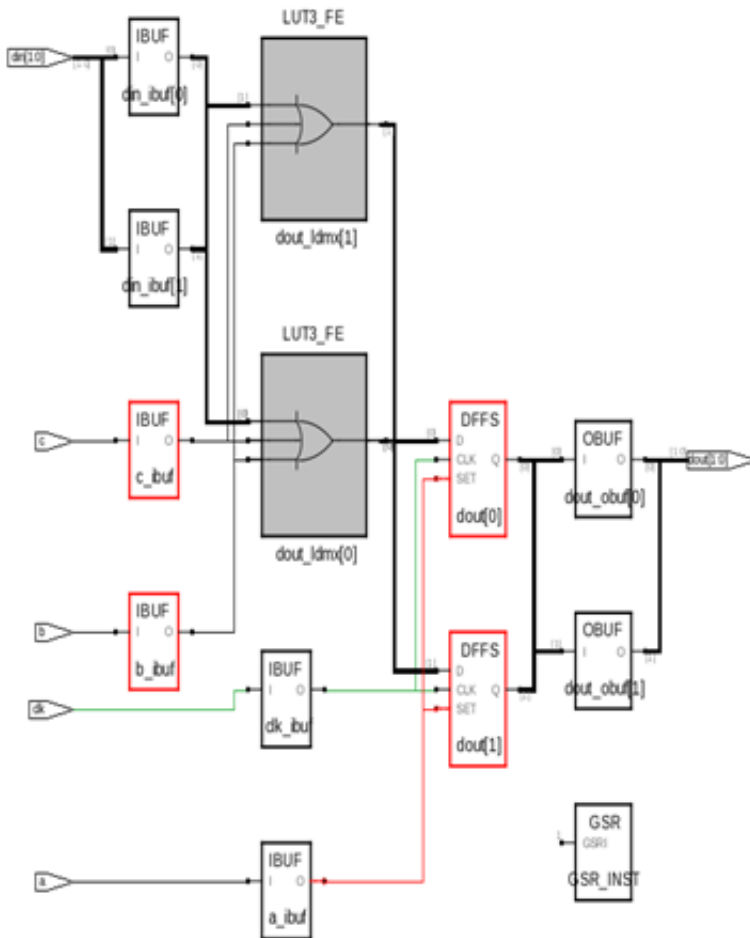
Before applying syn_direct_set:

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



After applying `syn_direct_set`:

The software assigns a net for the design to the dedicated set pin of a synchronous storage element (flip-flop).



syn_dspstyle

Attribute

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

syn_dspstyle Values

Value	Description
logic	Maps the object to logic instead of the DSP resources.
dsp	Maps the specified object to DSP.

Description

Specifies whether multipliers are implemented as dedicated hardware blocks or as logic.

syn_dspstyle Syntax

You can specify the attribute in the following files:

FDC	define_attribute {object} syn_dspstyle {value}
Verilog	object /* synthesis syn_dspstyle = value */ ;
VHDL	attribute syn_dspstyle of object : objectType is value;

Verilog Example

```
module test (clk, A, B, Q) ;  
  
  input clk;  
  input signed [8:0] A;  
  input signed [8:0] B;  
  output signed [17:0] Q;  
  
  reg signed [8:0] A_reg, B_reg;  
  reg signed [17:0] Q;
```

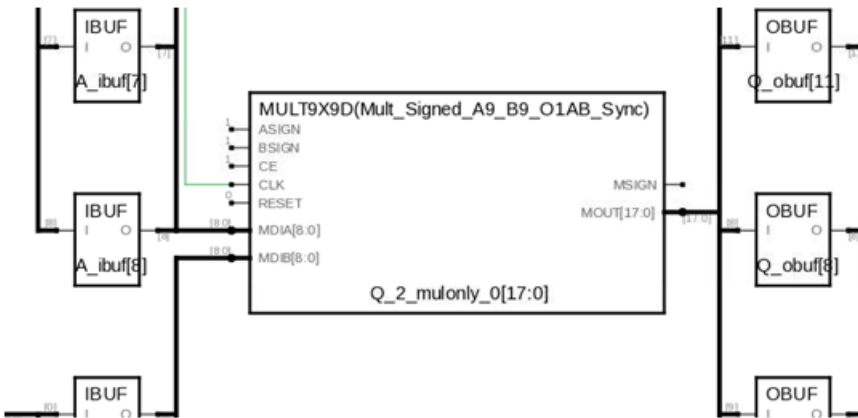
```

always @ (posedge clk)
begin
    A_reg <= A;
    B_reg <= B;
    Q <= A_reg * B_reg;
end
endmodule

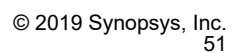
```

Effect of using `syn_dspstyle`

In this example, by default, the MULT9X9 multiplier is mapped to DSP.



To override the default behavior, apply `syn_dspstyle = "logic"`.



syn_encoding

Attribute

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

syn_encoding Values

The default is that the tool automatically picks an encoding style that results in the best performance. To ensure that a particular encoding style is used, explicitly specify that style, using the values below:

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>
gray	<p>More than one of the state registers can be hot. The synthesis tool <i>attempts</i> to have only one bit of the state registers change at a time, but it can allow more than one bit to change, depending upon certain conditions for optimization. For example:</p> <p>000, 001, 011, 010, 110</p> <p>Because 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>

Value	Description
sequential	<p>More than one bit of the state register can be hot. The synthesis tool makes no attempt at limiting the number of bits that can change at a time. For example:</p> <p>000, 001, 010, 011, 100</p> <p>This is one of the smallest encoding styles, so it is often used when area is a concern. Because more than one bit can be set (1), the value must be decoded to determine the state. This encoding style can be faster than a onehot style if you have a large output decoder following a state machine.</p>
safe	<p>This implements the state machine in the default encoding and adds reset logic to force the state machine to a known state if it reaches an invalid state. This value can be used in combination with any of the other encoding styles described above. You specify safe before the encoding style. The safe value is only valid for a state register, in conjunction with an encoding style specification.</p> <ul style="list-style-type: none"> • 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 <code>syn_shift_resetphase</code> attribute to remove it.
original	<p>This respects the encoding you set, but the software still does state machine and reachability analysis.</p>

You can specify multiple values. This snippet uses **safe,gray**. The encoding style for register `OUT` is set to **gray**, but if the state machine reaches an invalid state the synthesis tool will reset the values to a valid state.

```
module prep3 (CLK, RST, IN, OUT);
  input CLK, RST;
  input [7:0] IN;
  output [7:0] OUT;
  reg [7:0] OUT;
  reg [7:0] current_state /* synthesis syn_encoding="safe,gray" */;

  // Other code
```

Description

This attribute takes effect only when FSM Compiler is enabled. It overrides the default FSM Compiler encoding for a state machine. For the specified encoding to take effect, the design must contain state machines that have been inferred by the FSM Compiler. Setting this attribute when `syn_state_machine` is set to 0 will not have any effect.

The default encoding style automatically assigns encoding based on the number of states in the state machine. Use the `syn_encoding` attribute when you want to override these defaults.

syn_encoding Syntax

Global	Object
No	Instance Register

You can specify the attribute in the following files:

FDC	define_attribute {object} syn_encoding {value}
Verilog	<i>object</i> /* synthesis syn_encoding = "value" */ ;
VHDL	attribute syn_encoding of object : objectType is value

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.

FDC Example

```
module test (clk,rst,data,out);
input clk,rst,data;
output out;
reg out;

reg [2:0] ps,ns;

parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100,
S5=3'b101, S6=3'b110;
```

```
always@(posedge clk or posedge rst) begin
    if(rst)
        ps <= S0;
    else
        ps <= ns;    end

always@(ps or data)
begin

    ns = S0;
    case (ps)
        S0: if(data)
            ns = S1;
        else ns = S0;
        S1: if(data)
            ns = S1;
        else ns = S2;
        S2: if(data)
            ns = S1;
        else ns = S3;
        S3: if(data)
            ns = S4;
        else ns = S0;
        S4: if(data)
            ns = S5;
        else ns = S2;
        S5: if(data)
            ns = S1;
        else ns = S6;
        S6: if(data)
            ns = S1;
        else ns = S3;
        default: ns = S0;
    endcase
end

always@(ps or data) begin
    if((ps == S6) && (data == 1))
        out = 1;
    else
        out = 0;    end
endmodule

fdc: define_attribute {i:ps[6:0]} {syn_encoding} {onehot}
```

Effect of Using `syn_encoding`

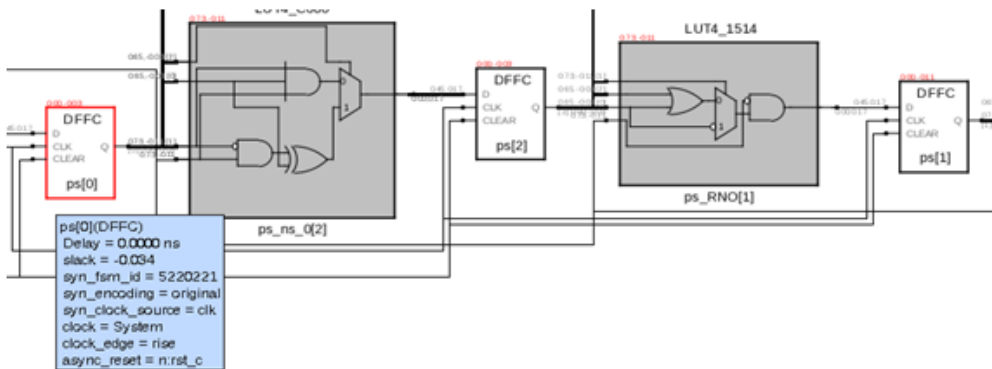
Before applying `syn_encoding`:

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

Encoding state machine `ps[6:0]` (view:work.test(verilog))

original code -> new code

```
000 -> 000
001 -> 001
010 -> 010
011 -> 011
100 -> 100
101 -> 101
110 -> 110
```



After applying `syn_encoding`:

The figure shows the state machine when the `syn_encoding` attribute is set to `onehot`, and the accompanying changes in the code.

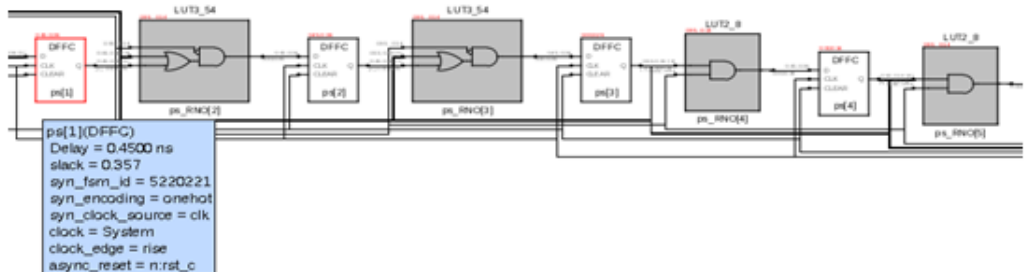
Encoding state machine `ps[6:0]` (view:work.test(verilog))

original code -> new code

```

000 -> 0000001
001 -> 0000010
010 -> 0000100
011 -> 0001000
100 -> 0010000
101 -> 0100000
110 -> 1000000

```



syn_hier

Attribute

Controls the amount of hierarchical transformation across boundaries on module or component instances during optimization.

syn_hier Values

Default	Global	Object
Soft	No	View

Value	Description
soft (default)	The synthesis tool determines the best optimization across hierarchical boundaries. This attribute affects only the design unit in which it is specified.
firm	Preserves the interface of the design unit. However, when there is cell packing across the boundary, it changes the interface and does not guarantee the exact RTL interface. This attribute affects only the design unit in which it is specified.
hard	Preserves the interface of the design unit and prevents most optimizations across the hierarchy. However, the boundary optimization for constant propagation is performed. Additionally, if all the clock logic is contained within the hard hierarchy, gated clock conversion can occur. This attribute affects only the specified design units.
fixed	Preserves the interface of the design unit with no exceptions. Fixed prevents all optimizations performed across hierarchical boundaries and retains the port interfaces as well. For more information, see Using syn_hier fixed, on page 62 .

remove	Removes the level of hierarchy for the design unit in which it is specified. The hierarchy at lower levels is unaffected. This only affects synthesis optimization. The hierarchy is reconstructed in the netlist and Technology view schematics.
macro	Preserves the interface and contents of the design with no exceptions. This value can only be set on structural netlists. (In the constraint file, or using the SCOPE editor, set <code>syn_hier</code> to <code>macro</code> on the view (the V: object type).
flatten	<p>Flattens the hierarchy of all levels below, but not the one where it is specified. This only affects synthesis optimization. The hierarchy is reconstructed in the netlist and Technology view schematics. To create a completely flattened netlist, use the <code>syn_netlist_hierarchy</code> attribute (syn_netlist_hierarchy, on page 90), set to false.</p> <p>You can use <code>flatten</code> in combination with other <code>syn_hier</code> values; the effects are described in Using syn_hier flatten with Other Values, on page 68.</p> <p>If you apply <code>syn_hier</code> to a compile point, <code>flatten</code> is the only valid attribute value. All other values only apply to the current level of hierarchy. The compile point hierarchy is determined by the type of compile point specified, so a <code>syn_hier</code> value other than <code>flatten</code> is redundant and is ignored.</p>

Description

During synthesis, the tool dissolves as much hierarchy as possible to allow efficient logic optimization across hierarchical boundaries while maintaining optimal run times. The tool then rebuilds the hierarchy as close as possible to the original source to preserve the topology of the design.

Use the `syn_hier` attribute to address specific needs to maintain the original design hierarchy during optimization. This attribute gives you manual control over flattening/preserving instances, modules, or architectures in the design.

It is advised that you avoid using `syn_hier="fixed"` with tri-states.

syn_hier Syntax

Global	Object
No	View

You can specify the attribute in the following files:

FDC file	define_attribute { <i>object</i> } syn_hier { <i>value</i> } define_global_attribute syn_hier { <i>value</i> }
Verilog	<i>object</i> /* synthesis syn_hier = " <i>value</i> " */ ;
VHDL	attribute syn_hier of <i>object</i> : architecture is " <i>value</i> " ;

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 62](#).

When Using Tri-states

It is advised that you avoid using syn_hier="fixed" with tri-states. However, if you do, here is how the software handles the following conditions:

- Tri-states driving output ports

If a module with syn_hier="fixed" includes tri-state gates that drive a primary output port, then the synthesis software retains a tri-state buffer so that the P&R tool can pack the tri-state into an output port.

- Tri-states driving internal logic

If a module with syn_hier="fixed" includes tri-state gates that drive internal logic, then the synthesis software converts the tri-state gate to a MUX and optimizes within the module accordingly.

In the following code example, myreg has syn_hier set to fixed.

```
module top(
    clk1,en1, data1,
    q1, q2
);
input clk1, en1;
input data1;
output q1, q2;
wire cwire, rwire;
wire clk_gt;
assign clk_gt = en1 & clk1;
```

```

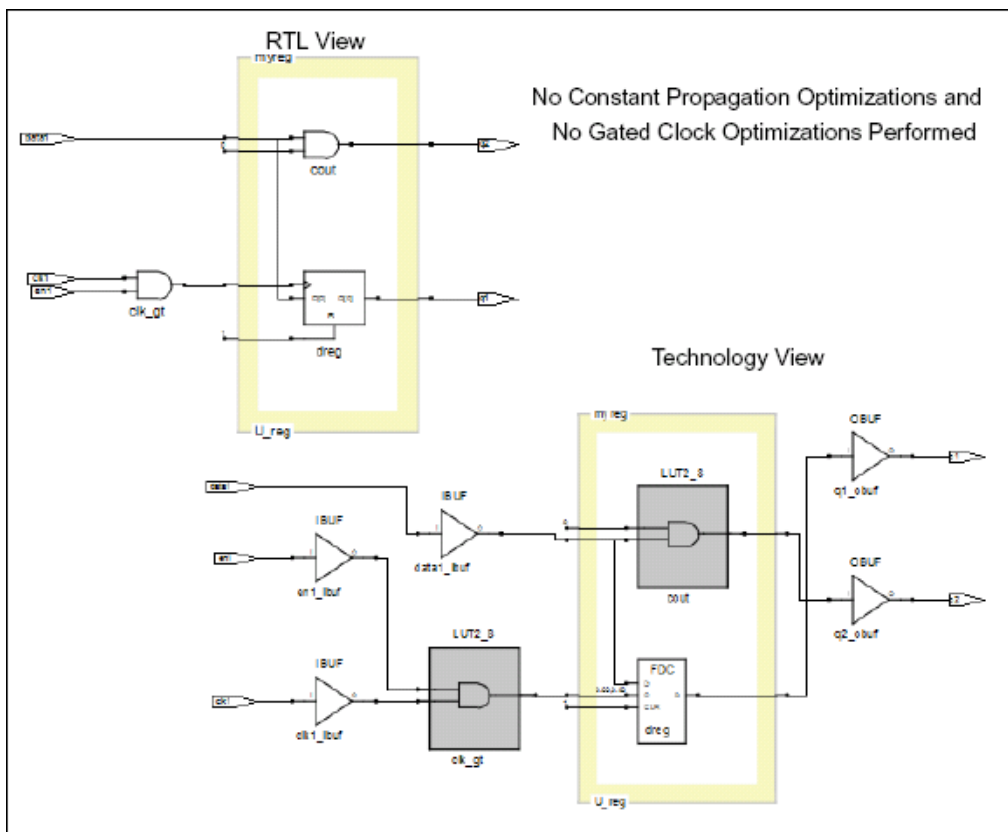
// Register module
myreg U_reg (
    .datain(data1),
    .rst(1'b1),
    .clk(clk_gt),
    .en(1'b0),
    .dout(rwire),
    .cout(cwire)
);
assign q1 = rwire;
assign q2 = cwire;
endmodule

module myreg (
    datain,
    rst,
    clk,
    en,
    dout,
    cout
) /* synthesis syn_hier = "fixed" */;
input clk, rst, datain, en;
output dout;
output cout;
reg dreg;
assign cout = en & datain;

always @(posedge clk or posedge rst)
begin
    if (rst)
        dreg <= 'b0;
    else
        dreg <= datain;
    end
assign dout = dreg;
endmodule

```

The HDL Analyst views show that myreg preserves its hierarchical boundaries without exceptions and prevents constant propagation and gated clock conversions optimizations



Verilog Example

```

module test (in1,in2,in3,in4,in5,in6,in7,in8,in9,in10,o1,o2);

input in1,in2,in3,in4,in5,in6,in7,in8,in9,in10;
output o1,o2;
wire o1,o2;

assign o2 = in9^in10;

syn_hier01 (in1,in2,in3,in4,in5,in6,in7,in8,o1);
endmodule

module syn_hier01 (a_i,b_i,c_i,d_i,e_i,f_i,g_i,h_i,y_o);

input a_i,b_i,c_i,d_i,e_i,f_i,g_i,h_i;
output y_o;
wire a_in,b_i,c_i,d_i,e_i,f_i,g_i,h_i,y_o;

assign a_in = a_i & 1'b1;

syn_hier02 ( a_in, b_i, c_i, d_i, e_i, f_i, g_i, h_i, y_o );
endmodule

module syn_hier02 ( a_i,b_i,c_i,d_i,e_i,f_i,g_i,h_i,y_o);

input a_i,b_i,c_i,d_i,e_i,f_i,g_i,h_i;
output y_o;
wire a_i,b_i,c_i,d_i,e_i,f_i,g_i,h_i, y_o;
wire ab,cd,ef,gh;

assign ab = a_i + b_i;
assign cd = c_i * d_i;
assign ef = e_i - f_i;
assign gh = g_i ^ h_i;

syn_hier03 ( ab, cd, ef, gh, y_o );

endmodule

module syn_hier03 ( ab,cd,ef,gh, y_o);

input ab,cd,ef,gh;
output y_o;
wire ab,cd,ef,gh;
wire y_o;
wire y_out;
assign y_o = !y_out;

```

:

```
syn_hier04 ( ab, cd, ef, gh, y_out );
endmodule

module syn_hier04 ( ab,cd,ef,gh, y_out);
input ab,cd,ef,gh;
output y_out;
wire ab,cd,ef,gh;
wire y_out;

syn_hier05 ( .a(ab),.b(cd),.c(ef),.d(gh),.y(y_out));
endmodule

module syn_hier05 ( a,b,c,d,y);
input a, b, c, d;
output y;
wire a, b, c, d;
wire y;
wire p;
wire q;

assign p = a & b;
assign q = c & d;

syn_hier06 ( .p_i(p),.q_i(q),.yn(y));
endmodule

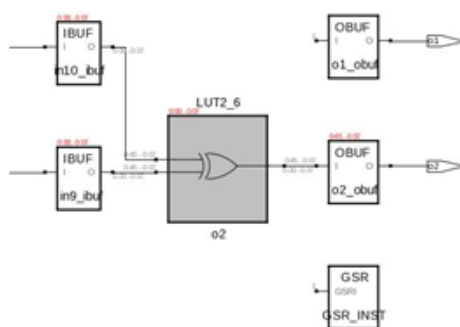
module syn_hier06 ( p_i,q_i,yn) /* synthesis syn_hier = fixed */;
input p_i,q_i;
output yn;
wire p_i,q_i;
wire yn;

assign yn = 1'b0;
endmodule
```

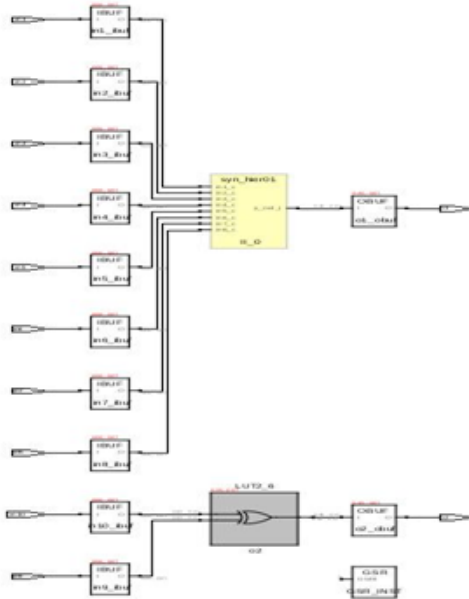
Effect of Using syn_hier

Before applying syn_hier:

The software dissolves the hierarchy due to the constant propagation.



The software does not dissolve the hierarchy and prevents optimization.



Using syn_hier flatten with Other Values

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

<code>flatten,soft</code>	Same as <code>flatten</code> .
<code>flatten,firm</code>	Flattens all lower levels of the design but preserves the interface of the design unit in which it is specified. This option also allows optimization of cell packing across the boundary.
<code>flatten,remove</code>	Flattens all lower levels of the design, including the one on which it is specified.

If you use `flatten` in combination with another option, the tool flattens as directed until encountering another `syn_hier` attribute at a lower level. The lower level `syn_hier` attribute then takes precedence over the higher level one.

These 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_insert_pad

Attribute

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

Description

The `syn_insert_pad` attribute is used when the Disable I/O Insertion option is not enabled (when buffers are automatically inserted) to allow users to selectively remove an individual buffer from a port or net or to replace a previously removed buffer.

- Setting the attribute to 0 on a port or net removes the I/O buffer (or prevents an I/O buffer from being automatically inserted).
- Setting the attribute to 1 on a port or net replaces a previously removed I/O buffer.

The `syn_insert_pad` attribute can only be applied through a constraint file.

syn_insert_pad Syntax

FDC `define_attribute {object} syn_insert_pad {1|0}`

[SCOPE Example](#)

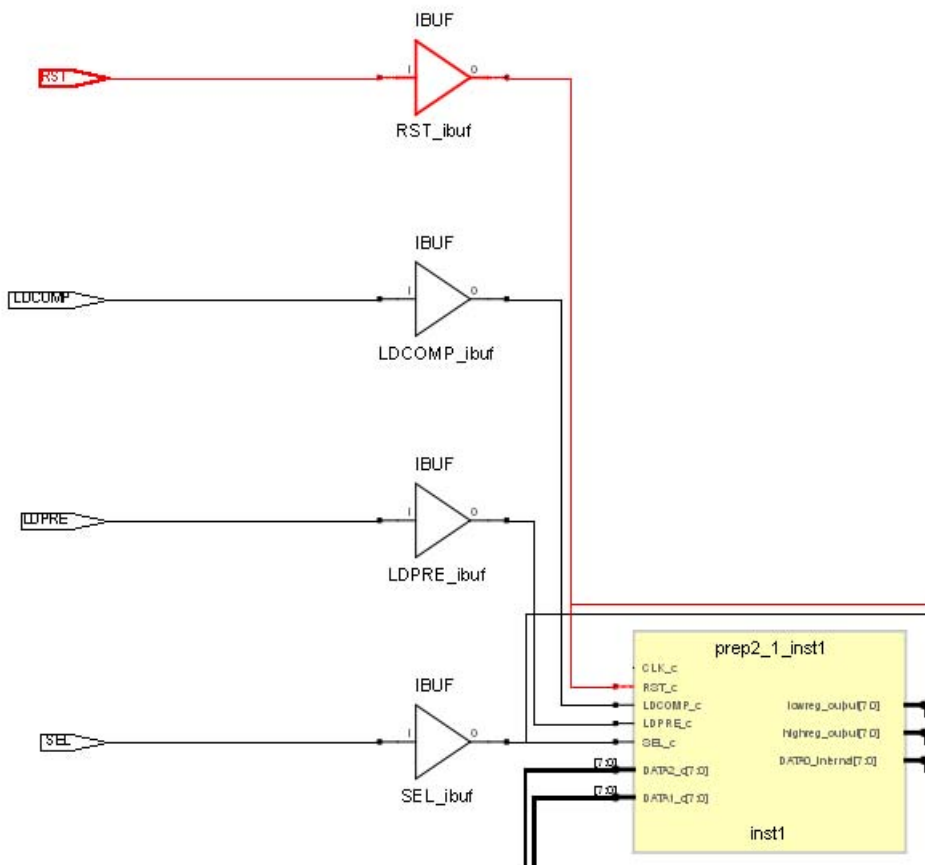
SCOPE Example

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

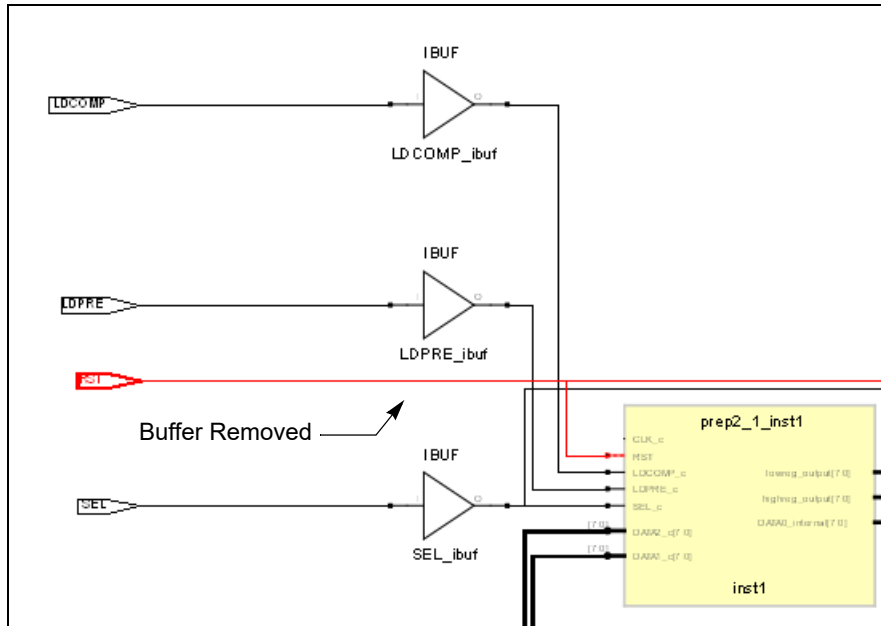
Enable	Object Type	Object	Attribute	Value	Value Type
<input checked="" type="checkbox"/>	<any>	p:RST	syn_insert_pad	0	

Effect of Using syn_insert_pad

Original design before applying `syn_insert_pad` (or after applying `syn_insert_pad` with a value of 1 to replace a previously removed buffer).



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



syn_keep

Directive

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

syn_keep Syntax

Global	Object
No	Net

syn_keep Values

1	Preserves the net without optimizing it away.
---	---

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 76](#) for an example). The buffer is not part of the final netlist, so no extra logic is generated. There are various situations where this directive is useful:

- To preserve a net that would otherwise be removed as a result of optimization. You might want to preserve the net for simulation results or to obtain a different synthesis implementation.
- To prevent duplicate cells from being merged during optimization. You apply the directive to the nets connected to the input of the cells you want to preserve.
- As a placeholder to apply the -through option of the set_multicycle_path or set_false_path timing constraint. This allows you to specify a unique path as a multiple-cycle or false path. Apply the constraint to the keep buffer.
- To prevent the absorption of a register into a macro. If you apply syn_keep to a reg or signal that will become a sequential object, the tool keeps the register and does not absorb it into a macro.

syn_keep with Multiple Nets in Verilog

In the following statement, `syn_keep` only applies to the last variable in the wire declaration, which is net `c`:

```
wire a,b,c /* synthesis syn_keep=1 */;
```

To apply `syn_keep` to all the nets, use one of the following methods:

- Declare each individual net separately as shown below.

```
wire a /* synthesis syn_keep=1 */;  
wire b /* synthesis syn_keep=1 */;  
wire c /* synthesis syn_keep=1 */;
```

- Use Verilog 2001 parenthetical comments, to declare the `syn_keep` directive as a single line statement.

```
(* syn_keep=1 *) wire a,b,c;
```

For more information, see [Attribute Examples Using Verilog 2001 Parenthetical Comments](#), on page 127.

syn_keep and SystemVerilog Data Types

The SystemVerilog data types behave like logic or reg, and SystemVerilog allows them to be assigned either inside or outside an `always` block. If you want to use `syn_keep` to preserve a net with a SystemVerilog data type, like `bit`, `byte`, `longint` or `shortint` for example, you must make sure that continuous assigns are made inside an `always` block, not outside.

The following table shows examples of SystemVerilog data type assignments:

Assignment in always block, syn_keep works	<pre> assign keep1_wireand_out; assign keep2_wireand_out; always @(*) begin keep1_bitand_out; keep2_bitand_out; keep1_byteand_out; keep2_byteand_out; keep1_longintand_out; keep2_longintand_out; keep1_shortintand_out; keep2_shortintand_out; </pre>
Assignment outside always block, syn_keep does not work	<pre> assign keep1_wireand_out; assign keep2_wireand_out; assign keep1_bitand_out; assign keep2_bitand_out; assign keep1_byteand_out; assign keep2_byteand_out; assign keep1_longintand_out; assign keep2_longintand_out; assign keep1_shortintand_out; assign keep2_shortintand_out; </pre>

For information about supported SystemVerilog data types, see [Data Types, on page 136](#).

You can specify the attribute in the following files:

Verilog	<code>object /* synthesis syn_keep="value" */ ;</code>
VHDL	<code>attribute syn_keep of object : objectType is value ;</code>

Verilog Example

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

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

Here is the source code used to produce the results shown in [Effect of Using syn_keep, on page 76](#).

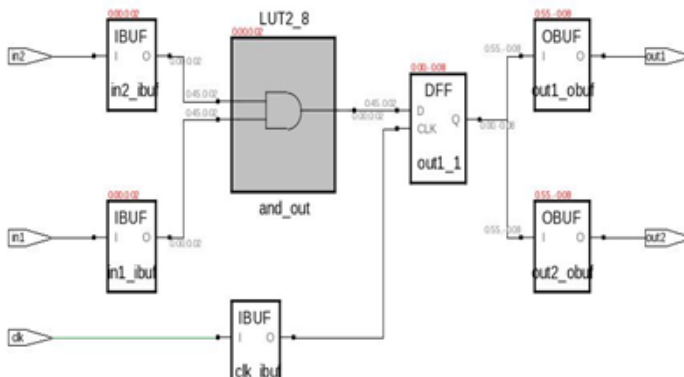
:

```
module test (out1, out2, clk, in1, in2);
  output out1, out2;
  input clk;
  input in1, in2;
  wire and_out;
  wire top1 /* synthesis syn_keep=1 */;
  wire top2 /* synthesis syn_keep=1 */;
  reg out1, out2;
  assign and_out=in1&in2;
  assign top1=and_out;
  assign top2=and_out;
  always @(posedge clk)begin;
    out1<=top1;
    out2<=top2;
  end
endmodule
```

Effect of Using syn_keep

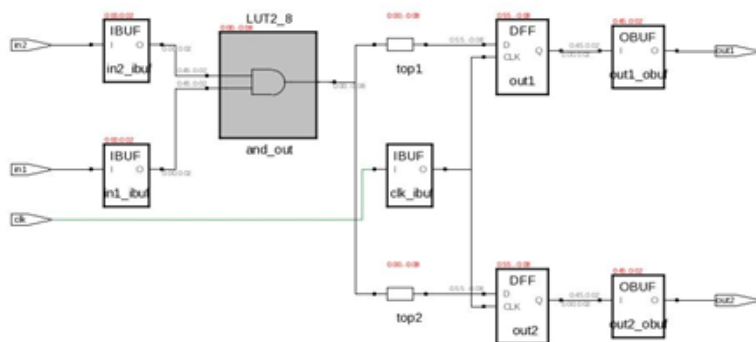
Before applying syn_keep:

The RTL has nets *top1* and *top2* driven by *and_out* and these nets in turn drive the two registers *out1* and *out2*. The software optimizes the *top1* and *top2* and keeps just one register as shown in the figure below.



After applying syn_keep:

To retain the two registers *out1* and *out2*, *syn_keep* is added to *top1* and *top2* nets and the software prevents it from getting optimized as shown in the figure below.



syn_loc

Attribute

The `syn_loc` attribute specifies the location (placement) of ports.

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.

syn_loc Syntax

Default	Global Attribute	Object
Not Applicable	No	Port

pinNumbers is a comma-separated list of pin or placement numbers. Refer to the vendor data book for valid values.

FDC	<code>define_attribute portDesignName {syn_loc} {pinNumbers}</code>	FDC Example
Verilog	<code>object /* synthesis syn_loc = "pinNumbers" */</code>	Verilog Example
VHDL	<code>attribute syn_loc of object : objectType is "pinNumbers";</code>	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	<input type="checkbox"/>						

Verilog Example

```

module test(a, b, clk, out1);
    input clk;
    input [2:0]a;
    input [2:0]b;
    output reg [2:0] out1/* synthesis syn_loc = "P14,P12,P11*/;

    always@(posedge clk)
    begin
        out1 <= a + b;
    end
endmodule

```

VHDL Example

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    generic (s : integer := 2);

    port (
        clk: in std_logic;
        in1: in std_logic_vector(s downto 0);
        in2: in std_logic_vector(s downto 0);
        d_out: out std_logic_vector(5 downto 0));
    attribute syn_loc : string;
    attribute syn_loc of d_out:signal is "P14,P12,P11,P5,P21,P13";
end test;

architecture beh of test is
begin
    process (clk)
    begin
        if rising_edge(clk) then
            d_out <= in1 & in2;
        end if;
    end process;
end beh;

```

Effect of Using syn_loc

This is the netlist output before the attribute is applied:

```
IBUF resetb_ibuf (  
    .o(resetb_c),  
    .I(resetb)  
);  
defparam resetb_ibuf.syn_loc="Z1";  
// @6:2  
IBUF clk_ibuf (  
    .o(clk_c),  
    .I(clk)  
);  
defparam clk_ibuf.syn_loc="P2";  
// @6:3  
IBUF dataA_ibuf (  
    .o(dataA_c),  
    .I(dataA)  
);  
defparam dataA_ibuf.syn_loc="L32";
```


syn_looplimit

Directive

VHDL

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

syn_looplimit Values

1-N	Overrides the default loop limit of 2000 in the RTL.
-----	--

Description

VHDL only. For Verilog applications use the `loop_limit` directive (see [loop_limit, on page 22](#)).

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.

The higher the value you set, the longer the runtime. To override the default limit of 2000 in the RTL, use the `loop_limit` directive (see [loop_limit, on page 22](#)).

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 315](#) in the *Command Reference*.

syn_looplimit Syntax

Global	Object
Yes	Specified at the beginning of the loop statement.

You can specify the attribute in the following file:

VHDL attribute syn_looplimit of loop: label is value

VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
entity sub is
    port ( din1,din2: in std_logic;
          clk : in std_logic;
          dout: out std_logic
    );
end sub;

architecture rtl of sub is
    signal temp : std_logic_vector(2555 downto 0);
    signal dout_temp: std_logic;
    attribute syn_looplimit : integer;
    attribute syn_looplimit of myloop1: label is 5000;
begin

    process(clk)
        variable il : integer := 0;
    begin
        if rising_edge(clk) then
            il:=0;
            myloop1:while il <= 2555 loop
                if (il mod 2 = 0) then
                    temp(il) <= din1;
                else
                    temp(il) <= not(din2);
                end if;
                dout_temp <= temp(il) xor dout_temp;
                il := il + 1;
            end loop myloop1;
        end if;
    end process;

    dout <= dout_temp;

end rtl;
```

syn_macro

Attribute

Determines whether instantiated macros are optimized and included in the final output netlist.

syn_macro Values

Value	Description	Default
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.	

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 also use this attribute with IP cores. 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 85](#).

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

You must set the attribute on a module:

```
fifo /* synthesis syn_macro = 1 */
```

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

The syn_macro setting affects how encrypted and unencrypted cores are handled in the software:

Core	syn_macro	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_maxfan

Attribute

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

syn_maxfan Value

value 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 set_option -fanout_limit command or -fanout_guide in the project file. Use the syn_maxfan attribute to specify a different (local) value for individual I/Os.

syn_maxfan Syntax

Global	Object Type
--------	-------------

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

FDC **define_attribute {object} syn_maxfan {integer}**

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

VHDL **attribute syn_maxfan of object : objectType is "value" ;**

VHDL Example

```

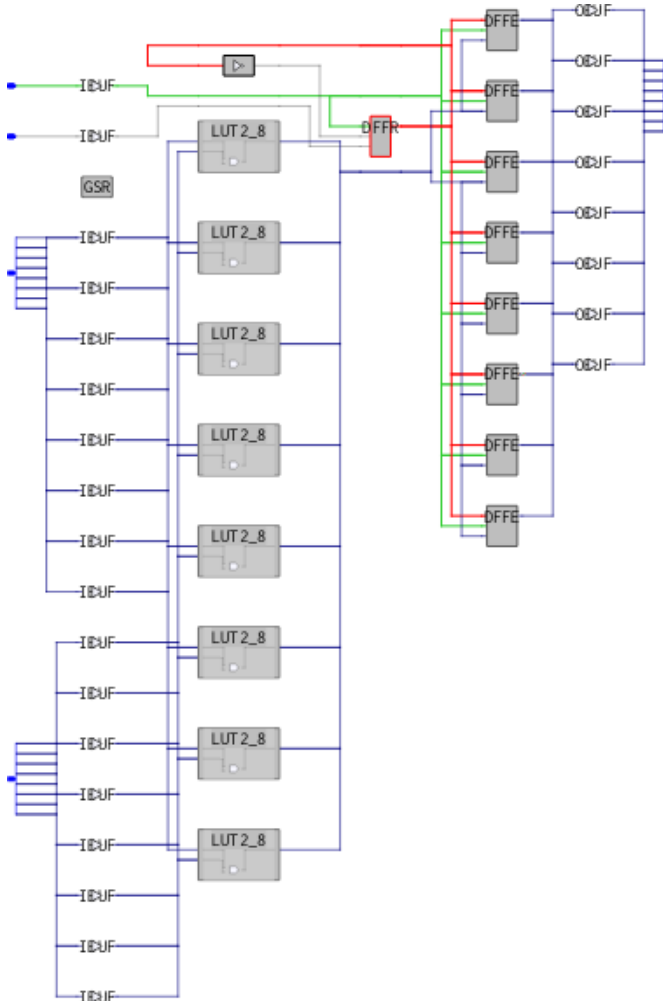
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity test is
    generic (n : integer := 10;
            m : integer := 7
    );
    port ( a : in std_logic_vector(7 downto 0);
          b : in std_logic_vector(7 downto 0);
          rst : in std_logic;
          clk : in std_logic;
          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 <= 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;

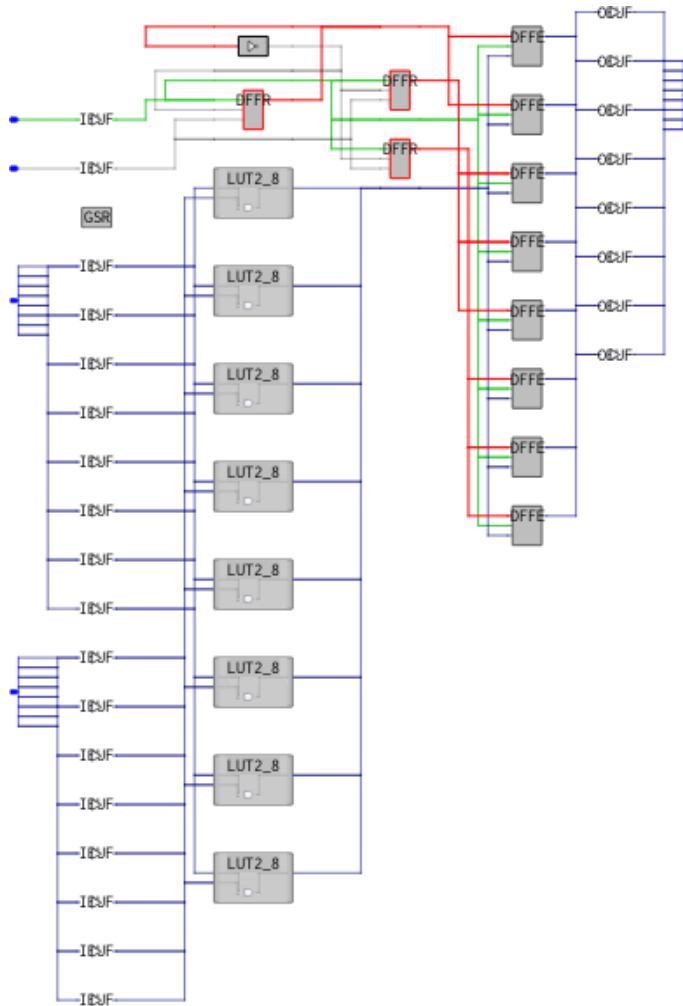
```

Effect of Using syn_maxfan

Before applying syn_maxfan:



After applying syn_maxfan:



syn_netlist_hierarchy

Attribute

Determines if the generated netlist is to be hierarchical or flat.

syn_netlist_hierarchy Values

Value	Description
1	Allows hierarchy generation
0	Flattens hierarchy in the netlist

Description

A global attribute that controls the generation of hierarchy in the VM 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 {value}
Verilog	<i>object</i> /* synthesis syn_netlist_hierarchy = "value" */ ;
VHDL	attribute syn_netlist_hierarchy of <i>object</i> : <i>objectType</i> is value ;

FDC Example

```

module fu_add  (input a,b,cin,output su,cy);

  assign su = a ^ b ^ cin;
  assign cy = (a & b) | ((a^b) & cin);
endmodule

module rca_adder #(parameter width =4)
  (input[width-1:0] A, input[width-1:0] 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, input[width-1:0] B1, input CIN1,
  output[width- 1:0] SUM, output COUT1 );
  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[7:4]));
  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

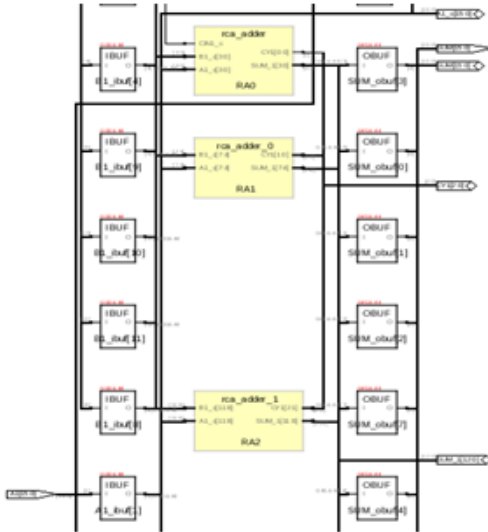
fdc: define_global_attribute  {syn_netlist_hierarchy} {0}

```

Effect of Using `syn_netlist_hierarchy`

Before applying `syn_netlist_hierarchy`:

Not applying the attribute (default is to allow hierarchy generation) or setting the attribute to 1 creates a hierarchical netlist.



After applying `syn_netlist_hierarchy`:

Applying the attribute with a value of 0 creates a flattened netlist.



syn_noprune

Directive

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

syn_noprune Values

0	Allows instances and black-box modules with unused output ports to be optimized away.
1	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 [Preserving Objects from Being Optimized Away, on page 396](#) in the *User Guide*.

syn_noprune Syntax

Global	Object
No	Module/architecture, instances, components

You can specify the attribute in the following files:

Verilog	<i>object</i> /* synthesis syn_noprune = "value" */;
VHDL	attribute syn_noprune of <i>object</i> : <i>objectType</i> is true;

VHDL Example

```

library ieee;
use ieee.std_logic_1164.all;
entity top is
    port (
        clk : in std_logic;
        a, b, c, d : in std_logic;
        out_a : out std_logic);
    end entity top;
architecture arch of top is
    component noprune_bb
        port(
            din : in std_logic;
            dout : out std_logic);
        end component noprune_bb;
    signal o1_noprunereg : std_logic;
    signal o2_reg : std_logic;

    attribute syn_noprune : integer;
    attribute syn_noprune of U1: label is 1;
    attribute syn_noprune of o1_noprunereg : signal is 1;

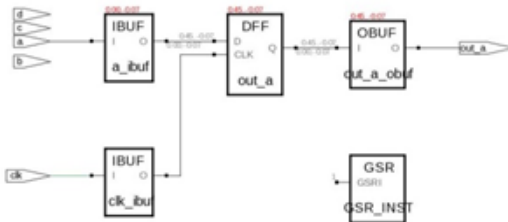
begin
    process(clk)
    begin
        if rising_edge(clk) then
            o1_noprunereg <= b and c;
            out_a <= a;
        end if;
    end process;
    U1: noprune_bb port map (d, o2_reg);
end architecture arch;

```

Effect of Using `syn_noprune`

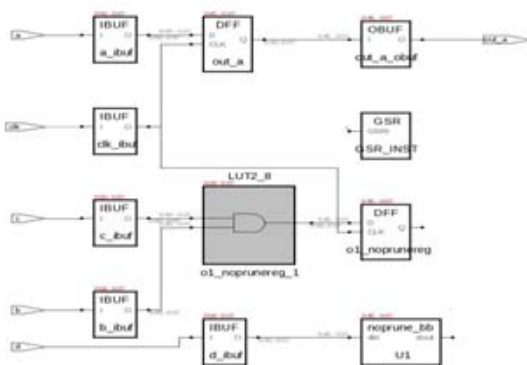
Before applying `syn_noprune`:

The black box instance `U1` and the instance `o1_noprunereg` are both optimized by the software as shown below.



After applying `syn_noprune`:

Both the instance and the black box are not optimized away by the software after applying `syn_noprune` directive as shown in figure below.



syn_pad_type

Attribute

Specifies an I/O buffer standard.

syn_pad_type Values

Value	Description
{standard} For example:LVCMOS_18	Specifies the port I/O standard.

Description

Specifies an I/O buffer standard. Refer to [Industry I/O Standards](#), on page 208 for a list of I/O buffer standards available.

syn_pad_type Syntax

FDC	define_io_standard -default portType {port} -delay_type portType syn_pad_type {io_standard} For example: define_io_standard {p} -delay_type output syn_pad_type {LVCMOS_18}	FDC Example
Verilog	object !* synthesis syn_pad_type = io_standard *!	Verilog Example
VHDL	attribute syn_pad_type of object : objectType is io_standard;	VHDL Example

FDC Example

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

-default_portType	<i>PortType</i> 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	<i>PortType</i> 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 LVTTL33 pad type	define_io_standard -default_input -delay_type input syn_pad_type {LVTTL33}
All output ports to the LVCMOS12 pad type	define_io_standard -default_output -delay_type output syn_pad_type {LVCMOS12}
All bidirectional ports to the SSTL15 pad type	define_io_standard -default_bidir -delay_type bidir syn_pad_type {SSTL15}

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 {LVCMOS15}

define_io_standard {out21} -delay_type output
    syn_pad_type {LVCMOS33}

define_io_standard {bidirbit} -delay_type bidir
    syn_pad_type {LVTTL33}
```

Verilog Example

```
module top (clk,A,B,PC,P);

input clk;
input A ;
input B,PC;
output reg P/* synthesis syn_pad_type = "LVCMOS18" */;

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 "LVCMOS18";
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;
```

syn_pipeline

Attribute

Permits registers to be moved to improve timing.

syn_pipeline Values

Value	Description
0	Disables pipelining.
1	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.

syn_pipeline Syntax

Global	Object
Yes	Registers

You can specify the attribute in the following files:

FDC	define_attribute syn_pipeline {value}
Verilog	<i>object</i> /* synthesis syn_pipeline = "value" */ ;
VHDL	attribute syn_pipeline of <i>object</i> : <i>objectType</i> is <i>value</i> ;

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 test
(a, b, clk, rst, q);
input [4:0] a, b;
input clk, rst;
output [9:0] q;

reg [9:0] q /* synthesis syn_pipeline=1 */;

always @(posedge clk)
begin
    if(rst)
        q <= 0;
    else
        q <= a * b;
end
endmodule

```

Effect of Using syn_pipeline

Before applying *syn_pipeline*:

The register *q* is not pipelined.



After applying *syn_pipeline*:

The register *q* is pipelined with register *q_2*.



syn_preserve

Directive

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

syn_preserve Values

Value	Description
1	Preserves register logic.
0	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.

syn_preserve Syntax

Global	Object
Yes	Registers, module/architecture

You can specify the attribute in the following files:

Verilog	<code>object /* synthesis syn_preserve = "value" */;</code>
VHDL	<code>attribute syn_preserve of object : objectType is value;</code>

Verilog Example

```
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;

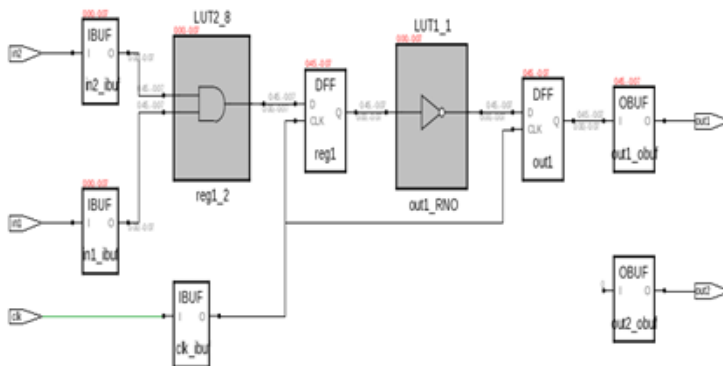
    always@ (posedge clk)begin
        reg1 <= in1 &in2;
        reg2 <= in1&in2;
        out1 <= !reg1;
        out2 <= !reg1 & reg2;
    end
endmodule
```

Effect of using syn_preserve

Before applying syn_preserve:

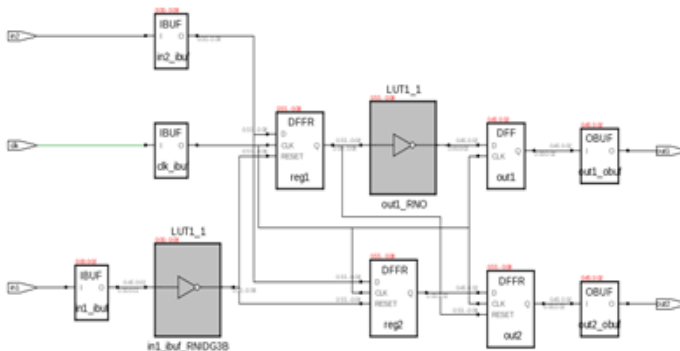
When syn_preserve is not set, reg1 and reg2 are shared because they are driven by the same source. out2 gets the result of the AND of reg2 and NOT reg1. This is equivalent to the AND of reg1 and NOT reg1, which is a 0. As this is a constant, the tool removes out2 and the output out2 is always 0.

The following figure shows reg1 and out2 are preserved during optimization with syn_preserve.



After applying `syn_preserve`:

The software preserves `reg1` and `out2` during optimization with `syn_preserve`.



syn_probe

Attribute

Inserts probe points for testing and debugging the internal signals of a design.

syn_probe Values

Value	Description
1	Inserts a probe, and automatically derives a name for the probe port from the net name.
0	Disables probe generation.
<i>portName</i>	Inserts a probe and generates a port with the specified 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. For string values on a bus slice, the port name is identical to the syn_probe value.

If empty square brackets [] are used, probe names are automatically indexed, according to the index of the bus being probed. If empty square brackets [] are not specified, the probe port names are adjusted.

syn_probe Syntax

Global	Object
No	Net

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

FDC	define_attribute {n:netName} syn_probe {value}
Verilog	object /* synthesis syn_probe = "value" */;
VHDL	attribute syn_probe of object : objectType is value ;

Verilog Example

```

`define plus      3'd0
`define minus     3'd1
`define band      3'd2
`define bor       3'd3
`define unegate   3'd4

module alu (out, opcode, a, b, sel);

output [7:0] out;
input [2:0] opcode;
input [7:0] a, b;
input sel;
reg [7:0] alu_out /* synthesis syn_probe="test_pt[]" */;
reg [7:0] out;

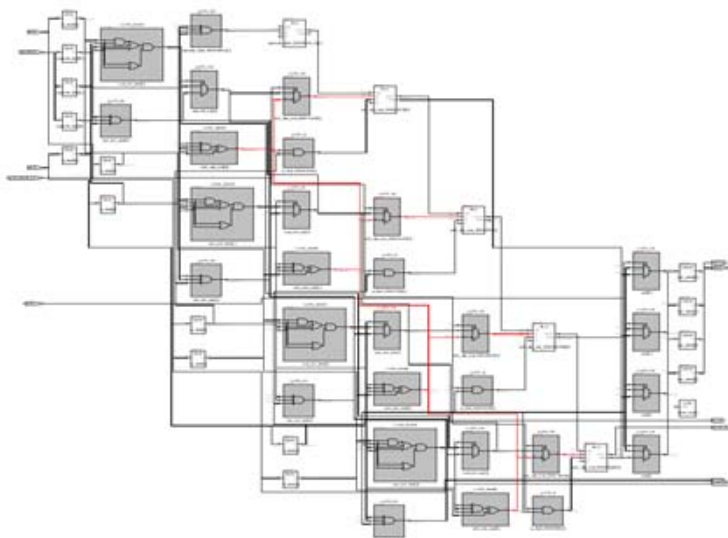
always @(opcode or a or b)
begin
    case (opcode)
        // arithmetic operations
        `plus:      alu_out =      a + b;
        `minus: alu_out =      a - b;
        // bitwise operations
        `band: alu_out =      a & b;
        `bor:      alu_out =      a | b;
        // unary operations
        `unegate:   alu_out =      ~a;
        default:    alu_out =      8'd0;
    endcase
end
always @(sel or alu_out or a)
    if (sel)
        out = alu_out;
    else
        out = a;
endmodule

```

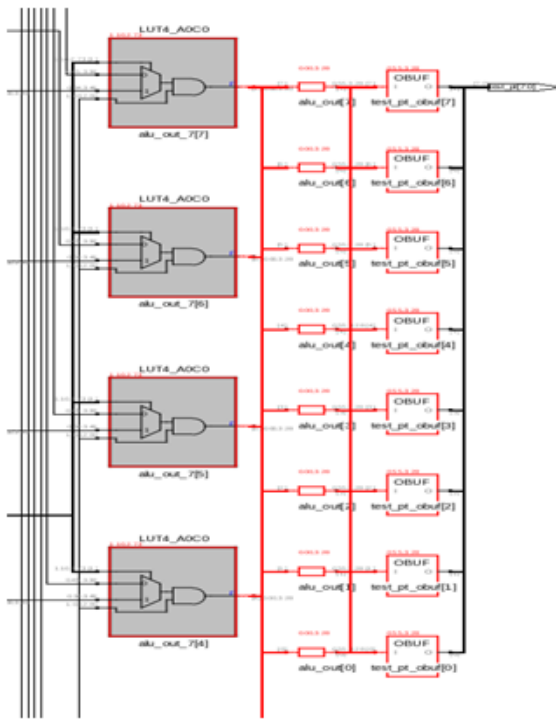
Effect of Using `syn_probe`

Before applying `syn_probe`:

The highlighted (in red color) signal `alu_out`, before applying `syn_probe` attribute, is not brought to the top-level of the design as shown in the figure below.



After applying `syn_probe`:



syn_pullup/syn_pulldown

Attribute

Specifies that a port is either a pullup or pulldown.

syn_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.

syn_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	define_attribute {portName} syn_pullup {1} define_attribute {portName} syn_pulldown {1}
Verilog	object /* synthesis syn_pulldown = 1 */; object /* synthesis syn_pullup = 1 */;
VHDL	attribute syn_pulldown of object : objectType is true; attribute syn_pullup of object : objectType is true;

SCOPE Example

	Enable	Object Type	Object	Attribute	Value
1	<input checked="" type="checkbox"/>	input_port	p:oen	syn_pullup	1
2	<input checked="" type="checkbox"/>	output_port	p:out1	syn_pullup	1

For example, you can apply the syn_pullup/syn_pulldown attribute as follows:

- On a multi-bit port such as RXD[3:0]


```
define_attribute {RXD[3:0]} syn_pullup {1}
```
- On a single-bit port such as RXA0


```
define_attribute {RXA0} syn_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} syn_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 syn_pullup/syn_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;
------	--

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))
        )
      )
    )
  )
```

syn_ramstyle

Attribute

Specifies the implementation for an inferred RAM.

syn_ramstyle Values

registers	Specifies that an inferred RAM be mapped to registers (flip-flops and logic), not technology-specific RAM resources.
block_ram	Specifies that the inferred RAM be mapped to the appropriate device-specific memory. It uses the dedicated memory resources in the FPGA.
distributed_ram	Specifies that the inferred RAM be mapped to the appropriate device-specific memory [distributed memory block].
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.
rw_check	<p>When enabled, the synthesis tool inserts bypass logic around the RAM to prevent a simulation mismatch between the RTL and post-synthesis simulations.</p> <p>You can use this option on its own or in conjunction with a RAM type value such as M512, or with the power value for supported technologies. You cannot use it with the no_rw_check option, as the two are mutually exclusive.</p> <p>There are other read-write check controls. See Read-Write Address Checks, on page 116 for details about the differences.</p>

no_rw_check_diff_clk When enabled, the synthesis tool prevents the insertion of bypass logic around the RAM. If you know your design has RAM that has a read clock and a write clock that are asynchronous, use `no_rw_check_diff_clk` to prevent the insertion of bypass logic. If this option is enabled, you should not set the asynchronous clock groups in your FDC file. For example, if you set the following, do not use this option:

```
create_clock {p:clkr} -period {10}
create_clock {p:clkw} -period {20}
set_clock_groups -derive -asynchronous -name
{async_clkgroup} -group { {c:clkw} }
```

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.

Description

The `syn_ramstyle` attribute specifies the implementation to use for an inferred RAM. You can apply the attribute globally, to a module, or a RAM instance. You can also use `syn_ramstyle` to prevent the inference of a RAM, by setting it to registers. If your RAM resources are limited, you can map additional RAMs to registers instead of RAM resources using this setting.

The `syn_ramstyle` values vary with the technology.

Read-Write Address Checks

When reads and writes are made to the same address, the output could be indeterminate, and this can cause simulation mismatches. By default, the synthesis tool inserts bypass logic around an inferred RAM to avoid these mismatches. The synthesis tool offers multiple ways to specify how to handle read-write address checking:

Read Write Control	Use when ...
<code>syn_ramstyle</code>	<p>You know your design does not read and write to the same address simultaneously and you want to specify the RAM implementation. The attribute has two mutually-exclusive read-write check options:</p> <ul style="list-style-type: none"> • Use <code>no_rw_check</code> to eliminate bypass logic. If you enable global RAM inference with the Read Write Check on RAM option, you can use <code>no_rw_check</code> to selectively disable glue logic insertion for individual RAMs. • Use <code>rw_check</code> to insert bypass logic. If you disable global RAM inference with the Read Write Check on RAM option, you can use <code>rw_check</code> to selectively enable glue logic insertion for individual RAMs.
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_rw_conflict_logic` attribute
- Read Write Check on RAM option on the Device panel of the Implementation Options dialog box.

syn_ramstyle Syntax

Global	Object
Yes	View, module, entity, RAM instance

You can specify the attribute in the following files:

FDC	define_attribute { <i>object</i> } syn_ramstyle { <i>value</i> } define_global_attribute syn_ramstyle { <i>value</i> }
Verilog	<i>object</i> /* synthesis syn_ramstyle = " <i>value</i> " */ ;
VHDL	attribute syn_ramstyle of <i>object</i> : <i>objectType</i> is value ;

FDC Example

```
module test (clk,we,waddr,raddr,din,q);

parameter ADDR_W = 4;
parameter DATA_W = 1;

input clk,we;

input [ADDR_W - 1 : 0] waddr,raddr;
input [DATA_W - 1 : 0] din;

output [DATA_W - 1 : 0] q;

reg [DATA_W - 1 : 0] q;

reg [DATA_W - 1 : 0] mem [(2**ADDR_W) - 1 : 0];

always @ (posedge clk)
    if(we) mem[waddr] <= din;

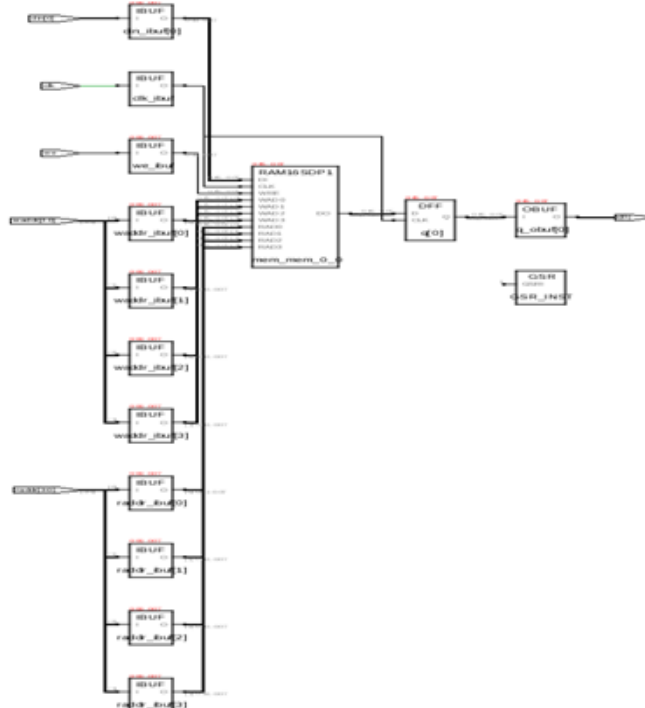
always @ (posedge clk )
    q <= mem[raddr];
endmodule

fdc: define_global_attribute {syn_ramstyle} {block_ram}
```

Effect of Using syn_ramstyle

Before applying syn_ramstyle:

Since the inferred RAM’s size is small, it is mapped to DRAM.



After applying `syn_ramstyle`:

With `syn_ramstyle="block_ram"`, the inferred RAM is mapped to Block RAM.

syn_replicate

Attribute

Controls replication of registers during optimization.

syn_replicate values

Value	Description
0	Disables duplication of registers
1	Allows duplication of registers

Description

The synthesis tool automatically replicates registers while optimizing the design and fixing fanouts, packing I/Os, or improving the quality of results.

If area is a concern, you can use this attribute to disable replication either globally or on a per-register basis. When you disable replication globally, it disables I/O packing and other QoR optimizations. When it is disabled, the synthesis tool uses only buffering to meet maximum fanout guidelines.

To disable I/O packing on specific registers, set the attribute to 0. Similarly, you can use it on a register between clock boundaries to prevent replication. Take an example where the tool replicates a register that is clocked by clk1 but whose fanin cone is driven by clk2, even though clk2 is an unrelated clock in another clock group. By setting the attribute for the register to 0, you can disable this replication.

syn_replicate Syntax

Global	Object
Yes	Register

You can specify the attribute in the following files:

FDC	define_global_attribute syn_replicate { <i>value</i> };
Verilog	<i>object</i> /* synthesis syn_replicate = " <i>value</i> " */;
VHDL	attribute syn_replicate of <i>object</i> : <i>objectType</i> is <i>value</i> ;

Verilog Example

```
module test (a, b, en1, en2, clk, q);
  input [31:0] a, b;
  input clk, en1, en2;
  output reg [31:0] q;

  reg en /* synthesis syn_replicate=0 */;

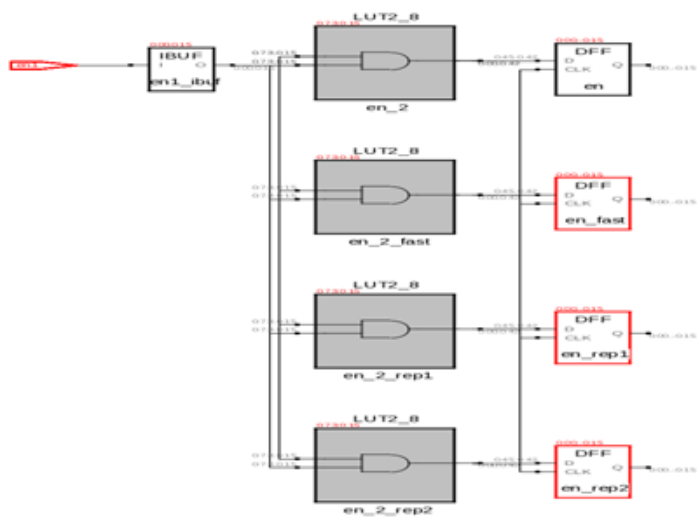
  always @(posedge clk)
  begin
    en <= en1 & en2;
    if (en)
      q <= a ^ b;
  end

endmodule
```

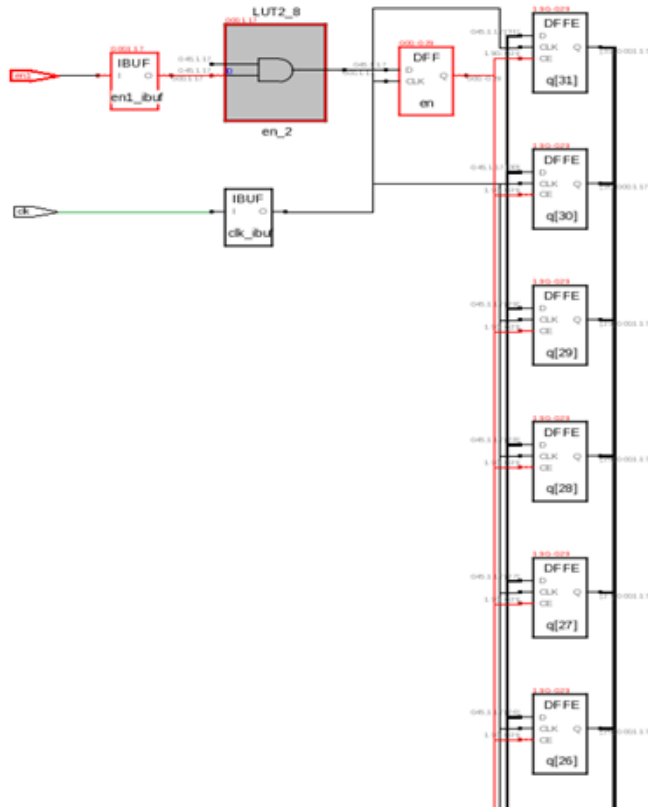
Effect of Using syn_replicate

Before applying *syn_replicate*:

The software replicates the instance *en* 3 times (shown in red), to improve timing, as shown in the figure below.



The software does not replicate the instance when `syn_replicate=0` attribute is applied.



syn_romstyle

Attribute

This attribute determines how ROM architectures are implemented.

syn_romstyle Values

Value	Description
logic	Specifies that an inferred RAM be mapped to logic resources (logic), not technology-specific RAM resources.
block_rom	Specifies that the inferred RAM be mapped to the appropriate device-specific memory [embedded memory block]. It uses the dedicated memory resources in the FPGA.
distributed	Specifies that the inferred RAM be mapped to the appropriate device-specific memory [distributed memory block].

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 Syntax

Global	Object
Yes	Module, ROM instance

This table summarizes the syntax in different files:

FDC	define_attribute { <i>object</i> } syn_romstyle { <i>value</i> } define_global_attribute syn_romstyle { <i>value</i> }
Verilog	<i>object</i> /* synthesis syn_romstyle = " <i>value</i> " */;
VHDL	attribute syn_romstyle of <i>object</i> : <i>objectType</i> is <i>value</i> ;

Verilog Example

```

module test (clock,addr,dataout) /* synthesis syn_romstyle =
"block_rom" */;
input clock;
input [4:0] addr;
output [7:0] dataout;
reg [7:0] dataout;
always @(posedge clock)
begin
    case (addr)
        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'b01001: dataout <= 8'b00011011;
        5'b01010: dataout <= 8'b10110001;
        5'b01011: dataout <= 8'b00110101;
        5'b01100: dataout <= 8'b01110010;
        5'b01101: dataout <= 8'b11100011;
        5'b01110: dataout <= 8'b00111111;
        5'b01111: dataout <= 8'b01010101;
        5'b10000: dataout <= 8'b10110000;
        5'b10001: dataout <= 8'b11111011;
        5'b10010: dataout <= 8'b00010001;
        5'b10011: dataout <= 8'b10110011;
        5'b10100: dataout <= 8'b00101011;
        5'b10101: dataout <= 8'b11101110;
        5'b10110: dataout <= 8'b01110111;
        5'b10111: dataout <= 8'b01110101;
        5'b11000: dataout <= 8'b01000011;
    endcase
end

```

```

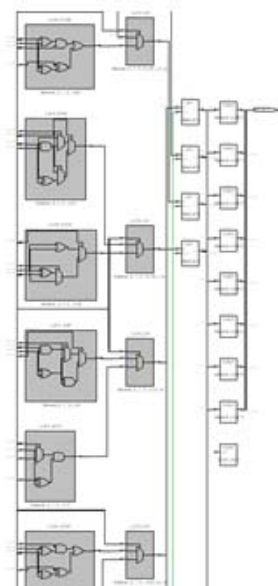
5'b11000: dataout <= 8'b01011100;
5'b11000: dataout <= 8'b11101011;
5'b11001: dataout <= 8'b00010100;
5'b11010: dataout <= 8'b00110011;
5'b11011: dataout <= 8'b00100101;
5'b11100: dataout <= 8'b01001110;
5'b11101: dataout <= 8'b01110100;
5'b11110: dataout <= 8'b11100101;
5'b11111: dataout <= 8'b01111110;
default: dataout <= 8'b00000000;
endcase
end
endmodule

```

Effect of Using `syn_romstyle`

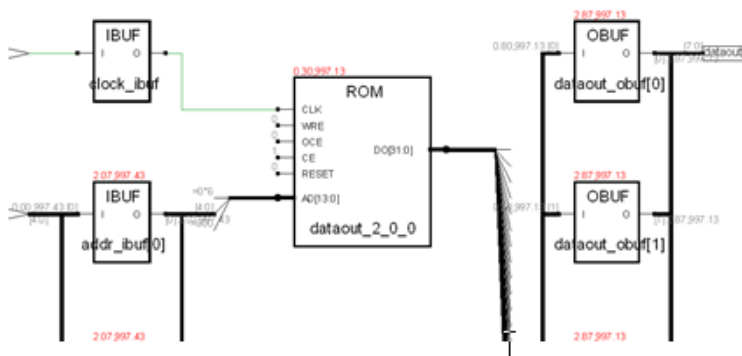
Before applying `syn_romstyle`:

The inferred ROM is mapped to logic as shown below in the figure.



After applying *syn_romstyle*:

The inferred ROM is mapped to the ROM resource after *syn_romstyle* attribute is applied.



syn_smhigheffort

Attribute

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

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

syn_smhigheffort Values

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

Description

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

syn_smhigheffort Syntax

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

For Verilog:

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

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

For VHDL:

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

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

```
module FSM1 (clk, in1, rst, out1);
input clk, rst, in1;
output [2:0] out1;
`define s0 3'b000
`define s1 3'b001
`define s2 3'b010
`define s3 3'bxxx
reg [2:0] out1;
reg [2:0] state /* synthesis syn_smhigheffort = 1 */;
reg [2:0] next_state;
always @(posedge clk or posedge rst)
if (rst) state <= `s0;
else state <= next_state;

// Combined Next State and Output Logic
always @(state or in1)
case (state)
`s0 : begin
out1 <= 3'b000;
if (in1) next_state <= `s1;
else next_state <= `s0;
end
`s1 : begin
out1 <= 3'b001;
if (in1) next_state <= `s2;
else next_state <= `s1;
end
`s2 : begin
out1 <= 3'b010;
if (in1) next_state <= `s3;
else next_state <= `s2;
end
default : begin
out1 <= 3'bxxx;
next_state <= `s0;
end
endcase
end
```

```

end
endcase
endmodule

```

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

```

library ieee;
use ieee.std_logic_1164.all;
entity FSM1 is
    port (clk,rst,in1 : in std_logic;
          out1 : out std_logic_vector (2 downto 0));
end FSM1;
architecture behave of FSM1 is
    type state_values is (s0, s1, s2,s3);
    signal state, next_state: state_values;
    attribute syn_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;
            when s2 =>
                out1 <= "010";
                if in1 = '1' then next_state <= s3;
                else next_state <= s2;
                end if;
            when others =>

```

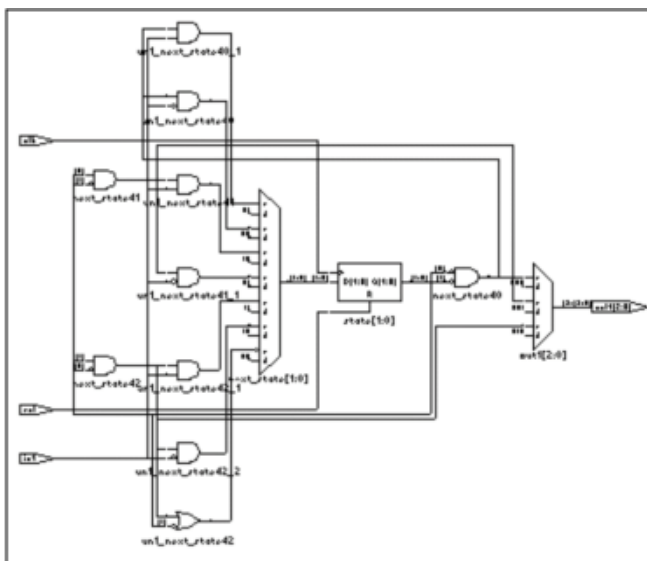
:

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

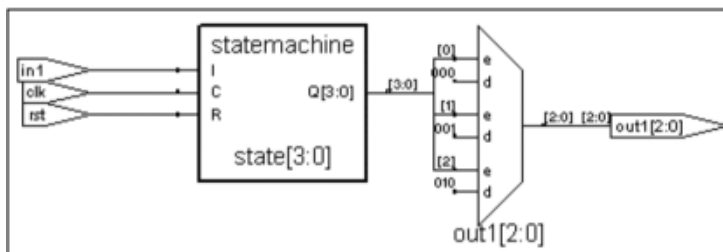
This is the VHDL source code used for the example in the following figure.

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

syn_srstyle

Attribute

Overrides the default behavior of seqshift implementation using RAM or registers.

Description

The default implementation of sequential shift registers is using RAM or registers depending on the thresholds. The `syn_srstyle` attribute is used to override the default behavior of seqshift implementation using RAM or registers.

syn_srstyle Values

Values	Description
registers	seqshifts are inferred as registers
block_ram	seqshifts are inferred as block RAM
distributed_ram	seqshifts are inferred as distributed RAM

The `syn_srstyle` attribute is used to determine the implementation of the sequential shift components as follows:

FDC	<code>define_attribute {object} syn_srstyle {registers distributed_ram block_ram }</code> <code>define_global_attribute syn_srstyle {registers distributed_ram block_ram }</code>
Verilog	<code>object /* synthesis syn_srstyle = " registers distributed_ram block_ram " */;</code>
VHDL	<code>attribute syn_srstyle : string;</code> <code>attribute syn_srstyle of object : signal is " registers distributed_ram block_ram ";</code>

The design below is implemented using registers instead of the default implementation of distributed RAM:

```
module seqshift (clk, din, dout) ;
parameter SRL_WIDTH = 3;
parameter SRL_DEPTH = 8;
input clk;
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 = " registers " */ ;
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
```

syn_tlvds_io/syn_elvds_io

Attribute

The `syn_tlvds_io/syn_elvds_io` attribute controls differential I/O buffer inferencing.

syn_tlvds_io/syn_elvds_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_tlvds_io/syn_elvds_io` attribute controls differential I/O buffer inferencing. 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 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.

syn_tlvds_io/syn_elvds_io Syntax

Global Attribute	Object	
Yes	Port	
FDC	<pre>define_attribute object {syn_tlvds_io} {1 0} define_global_attribute {syn_tlvds_io} {1 0}</pre>	FDC Example
Verilog	<pre>object /* synthesis syn_tlvds_io = 1 0 */</pre>	Verilog Example
VHDL	<pre>attribute syn_tlvds_io : true false; attribute syn_tlvds_io of object : objectType is true false;</pre>	VHDL Example

FDC Example

	Enable	Object Type	Object	Attribute	Value	Value Type	Description
1	<input checked="" type="checkbox"/>	port	p:in1_p	syn_diff_io	1	boolean	Allows inference of differential I/O pads
2	<input checked="" type="checkbox"/>	port	p:in1_n	syn_diff_io	1	boolean	Allows inference of differential I/O pads
3	<input checked="" type="checkbox"/>	port	p:out1	syn_diff_io	1	boolean	Allows inference of differential I/O pads
4	<input checked="" type="checkbox"/>	port	p:out2	syn_diff_io	1	boolean	Allows inference of differential I/O pads

Verilog Example

```

module test(in1_p, in1_n, clk, out1,out2);
input in1_p/* synthesis syn_tlvds_io = 1*/,
      in1_n/* synthesis syn_tlvds_io = 1*/;
input clk;
output out1/* synthesis syn_tlvds_io = 1*/,
       out2/* synthesis syn_tlvds_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_tlvds_io: boolean;
    attribute syn_tlvds_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

```

:

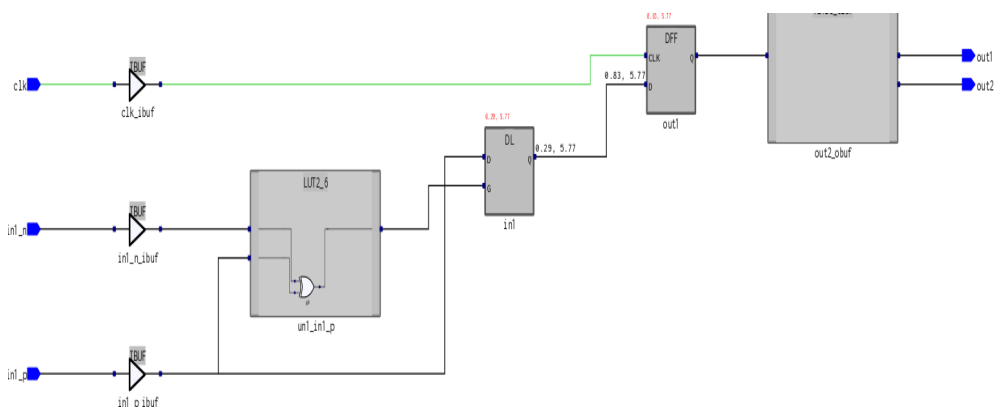
```
process(in1_p,in1_n)
begin
    if(in1_p /= in1_n)then
        in1 <= in1_p;
    end if;
end process;
process(clk)
begin
    if(clk'event and clk = '1')then
        out1 <= in1;
        tmp1 <= in1;
    end if;
end process;
out2 <= not tmp1;
end arch;
```

Effect of Using syn_tlvds_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_tlvds_io= 0*/;`
`in1_n /*synthesis syn_tlvds_io = 0*/;`
`output out1 /*synthesis syn_tlvds_io = 0*/;`
`out2 /*synthesis syn_tlvds_io = 0*/;`

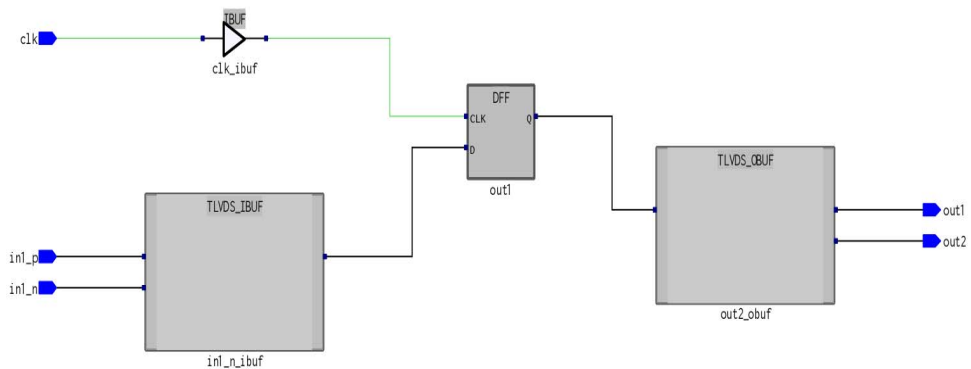
VHDL `attribute syn_tlvds_io of in1_p,in1_n,out1,out2:signal is`
`false;`



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

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

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



syn_useenables

Attribute

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

syn_useenables Values

Value	Description
1	Infers registers with clock-enable pins
0	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.

syn_useenables Syntax

Global	Object
No	Register

You can specify the attribute in the following files:

FDC	define attribute {<i>object</i>} syn_useenables {<i>value</i>}
Verilog	<i>object</i> /* synthesis syn_useenables = "<i>value</i>" */;
VHDL	attribute syn_useenables of <i>object</i> : <i>objectType</i> is <i>value</i>;

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

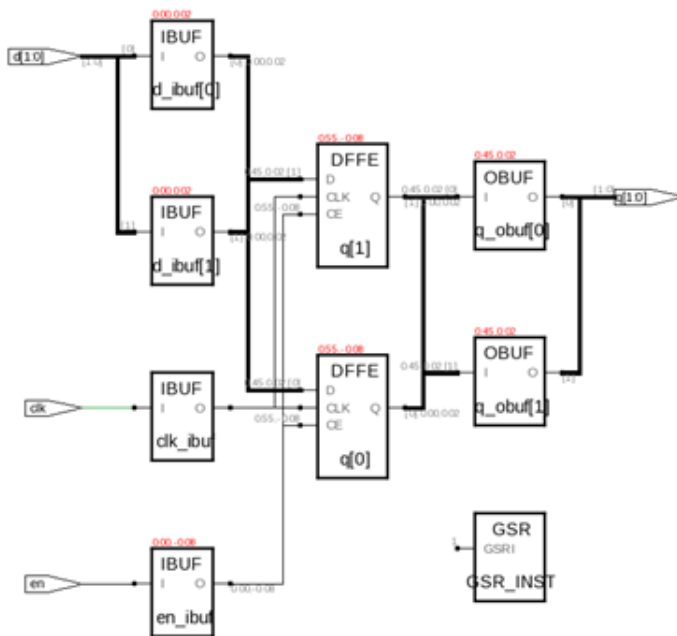
```

Effect of Using syn_useenables

Before applying *syn_useenables*:

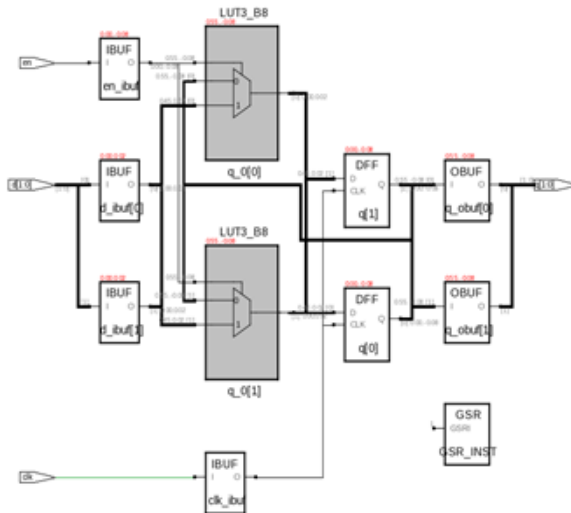
When attribute is not applied, the software uses registers with clock-enable pins (DFFEs in the below schematic).

:



After applying *syn_useenables*:

When the attribute is applied, the software uses registers without clock-enable pins. (DFFs in the below schematic).



syn_use_set

Attribute

Controls the use of set synchronous flip flops in the netlist, when they do not have any user-defined initial value.

syn_use_set Values

Value	Description
1 true (default value)	Enables the use of set synchronous flip flop primitives in a netlist, when the RTL does not have any user-defined initial value for the flip flop.
0 false	Disables the use of set synchronous flip flop primitives in a netlist, when the RTL does not have any user-defined initial value for the flip flop.

Description

This attribute is applied only when the set flip flops do not have any initial value in the RTL. The `syn_use_set` attribute can be specified in the RTL at the module/entity level or in a separate constraints file, as a global attribute.

syn_use_set Syntax

FDC `define_global_attribute {object} {syn_use_set} {value}`

Verilog `object /* synthesis syn_use_set = value*/`

VHDL `attribute syn_use_set of object: objectType is value;`

Example

The following code specifies `syn_use_set` correctly:

```
module test_RTL(clk,set,d,q)/*synthesis syn_use_set=0*/;
input clk,set;
input d;
output q;
reg q;
always @ (posedge clk)
```



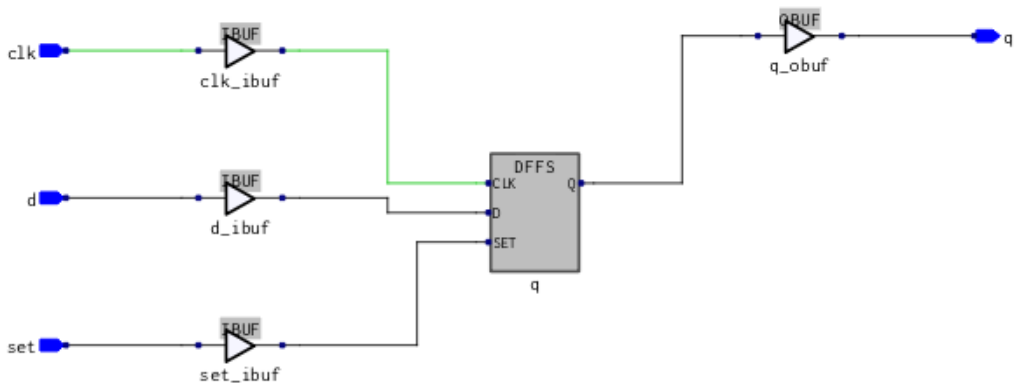
```

begin
    if(set) q <= 1;
    else q <= d;
end
endmodule

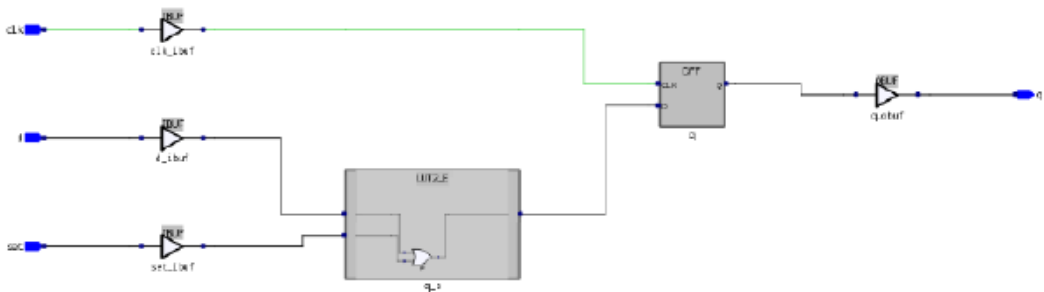
```

Effect of Using syn_use_set

Before applying syn_use_set:



After applying syn_use_set:



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.

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>
------	---

VHDL Example

```

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

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

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

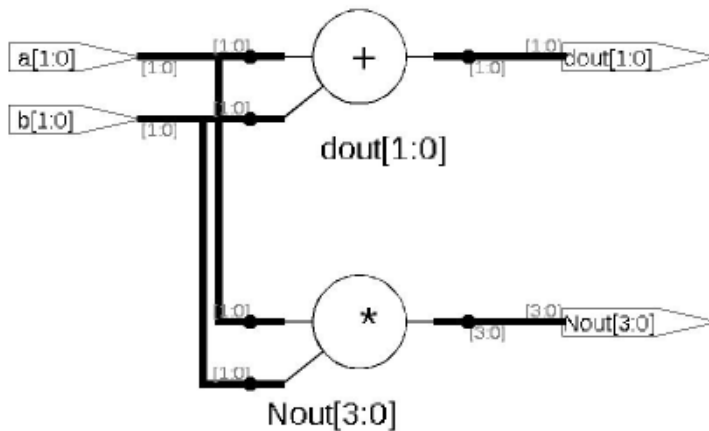
end rtl;

```

Effects of Using `translate_off`/`translate_on`

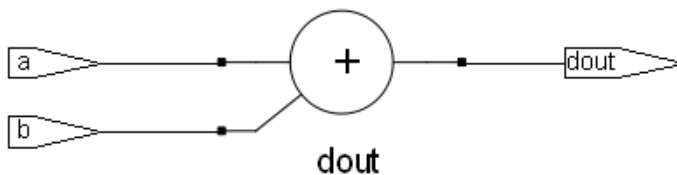
Before applying *translate_off*/*translate_on*:

The software maps the source code between the directives, [Nout logic, red color highlighted in the figure below]



After applying *translate_off/translate_on*:

The software ignores the source code between the directives mentioned, no logic for `Nout` output as shown in the graphic below.



Index

A

attributes

- alphabetical summary [11](#)
- global attribute summary [14](#)
- specifying in the SCOPE spreadsheet [8](#)
- specifying, overview of methods [8](#)
- syn_direct_set [44](#)

Attributes panel, SCOPE spreadsheet [9](#)

B

black box directives

- syn_black_box [34](#)

C

clock enables

- inferring with syn_direct_enable [37](#)
- net assignment [37](#)

compiler

- loop iteration, loop_limit [22](#)
- loop iteration, syn_looplevelimit [81](#)

D

define_attribute

- syntax [10](#)

define_false_path

- using with syn_keep [73](#)

define_global_attribute

- summary [14](#)
- syntax [10](#)

define_multicycle_path

- using with syn_keep [73](#)

F

fanout limits

- overriding default [86](#)
- syn_maxfan attribute [86](#)

FSMs

- syn_encoding attribute [52](#)

full_case directive [18](#)

G

global attributes summary [14](#)

H

hierarchy

- flattening with syn_hier [59](#)

I

I/O packing

- disabling with syn_replicate [121](#)

instances

- preserving with syn_noprune [94](#)

L

loop_limit directive [22](#)

M

macros

- preserving with syn_macro [12](#), [83](#)

N

nets

- preserving with syn_keep [73](#)

P

parallel_case directive [24](#)

pin locations

- forward annotating [78](#)

pipelining

- syn_pipeline attribute [101](#)

priority encoding [24](#)

probes

- inserting [107](#)

R

RAMs

- implementation styles [115](#)

registers
 preserving with `syn_preserve` [104](#)
replication
 disabling [121](#)
retiming
 `syn_allow_retiming` attribute [27](#)

S

SCOPE spreadsheet
 Attributes panel [9](#)
sequential optimization, preventing with `syn_
 _preserve` [104](#)
state machines
 extracting [129](#)
 `syn_allow_retiming` attribute [27](#)
 `syn_area_group` attribute [30](#)
 `syn_black_box` directive [34](#)
 `syn_direct_enable` attribute [37](#)
 `syn_direct_set` Attribute [44](#)
 `syn_dspstyle` attribute [49](#)
 `syn_elvds_io` attribute [136](#)
 `syn_encoding` attribute [52](#)
 `syn_hier` attribute [59](#)
 `syn_insert_pad` [70](#)
 `syn_keep` directive [73](#)
 `syn_loc` attribute [78](#)
 `syn_looplimit` directive [81](#)
 `syn_macro` directive [83](#)
 `syn_maxfan` attribute [86](#)
 `syn_netlist_hierarchy` attribute [90](#)
 `syn_noprune` directive [94](#)
 `syn_pad_type` attribute [97](#)
 `syn_pipeline` attribute [101](#)
 `syn_preserve` directive [104](#)
 `syn_probe` attribute [107](#)
 `syn_pulldown` attribute [111](#)
 `syn_pullup` attribute [111](#)
 `syn_ramstyle` attribute [115](#)
 `syn_reduce_controlset_size` attribute [121](#)
 `syn_replicate` attribute [121](#)
 `syn_romstyle` attribute [125](#)
 `syn_srlstyle` attribute [129](#)
 `syn_state_machine` attribute [129](#)
 `syn_tlvds_io` attribute [136](#)
 `syn_useenables` attribute [140](#)
SystemVerilog data types

 assignment for `syn_keep` [74](#)

T

`translate_off` directive [146](#)
`translate_on` directive [146](#)

V

Verilog
 ignoring code with `translate off/on` [146](#)
 `syn_keep` on multiple nets [74](#)

W

wires, preserving with `syn_keep` directive [73](#)