

Vivado Design Suite Properties Reference Guide

UG912 (v2013.3) October 2, 2013



Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2012-2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/02/2013	2013.3	Added Chapter 1, Vivado First Class Objects . Added BUFFER_TYPE , CFGBVS , CONFIG_VOLTAGE , FSM_ENCODING , FSM_SAFE_STATE , REF_NAME , REF_PIN_NAME , and USED_IN . Replaced COMPATIBLE_CONFIG_MODES with CONFIG_MODE .
06/19/2013	2013.2	Added H_SET and HU_SET , IBUF_LOW_PWR , LOCK_PINS , PBLOCK , RLOC , RLOCS , RLOC_ORIGIN , ROUTE_STATUS , RPM , RPM_GRID , U_SET . Removed OUT_TERM .
03/20/2013	2013.1	Edited details of DCI_CASCADE , DIFF_TERM , and IOB . Added IOBDELAY , KEEPER , OUT_TERM , PULLUP , PULLDOWN , POST_CRC , POST_CRC_ACTION , POST_CRC_FREQ , POST_CRC_INIT_FLAG , and POST_CRC_SOURCE , properties.

Table of Contents

Chapter 1: Vivado First Class Objects

Introduction	5
First Class Objects	6
Copying Examples from this Document	8
BEL.....	9
CELL.....	12
NET	19
PIN	23
PORT	26
SITE	29

Chapter 2: Key Property Descriptions

Properties Information	33
ASYNC_REG	34
BEL.....	38
BUFFER_TYPE	41
CFGBVS	43
CLOCK_DEDICATED_ROUTE.....	45
CONFIG_MODE.....	47
CONFIG_VOLTAGE	49
DCI_CASCADE	51
DIFF_TERM	53
DONT_TOUCH.....	56
DRIVE	58
FSM_ENCODING.....	61
FSM_SAFE_STATE.....	63
H_SET and HU_SET.....	65
HIODELAY_GROUP.....	69
HLUTNM	72
IBUF_LOW_PWR	76
IN_TERM.....	78
INTERNAL_VREF.....	81
IOB.....	83

IOBDELAY	85
IODELAY_GROUP	87
IOSTANDARD	90
KEEP_HIERARCHY	93
KEEPER	95
LOC	97
LOCK_PINS	99
LUTNM	103
MARK_DEBUG	106
PACKAGE_PIN	108
PBLOCK	110
POST_CRC	112
POST_CRC_ACTION	114
POST_CRC_FREQ	116
POST_CRC_INIT_FLAG	118
POST_CRC_SOURCE	120
PROHIBIT	122
PULLDOWN	123
PULLUP	125
REF_NAME	127
REF_PIN_NAME	128
RLOC	129
RLOCS	133
RLOC_ORIGIN	135
ROUTE_STATUS	138
RPM	140
RPM_GRID	141
SLEW	143
U_SET	146
USED_IN	150
VCCAUX_IO	152

Appendix A: Additional Resources

Xilinx Resources	155
Solution Centers	155
References	155

Vivado First Class Objects

Introduction

This reference manual discusses the first class objects, and the properties available for those objects, in the Xilinx® Vivado® Design Suite. It consists of the following:

- **Chapter 1, Vivado First Class Objects:** Describes the various design and device objects used by the Vivado Design Suite to model the FPGA design database. A definition of the object, a list of related objects, and a list of properties attached to the object are provided.
- **Chapter 2, Key Property Descriptions:** For many Vivado Design Suite properties, a description, supported architectures, applicable elements, values, syntax examples (Verilog, VHDL, and XDC), and affected steps in the design flow are provided.
- **Appendix A, Additional Resources:** Resources and documents available on the Xilinx support website at www.xilinx.com/support are provided.

First Class Objects

Vivado Design Suite supports a number of first class objects in the in-memory design database. These objects represent the design, or the logical netlist, and the target Xilinx FPGA, or device. The relationship between the netlist objects and the device objects maps the design onto the device. [Figure 1-1](#) illustrates the relationships between some of the Vivado first class objects.

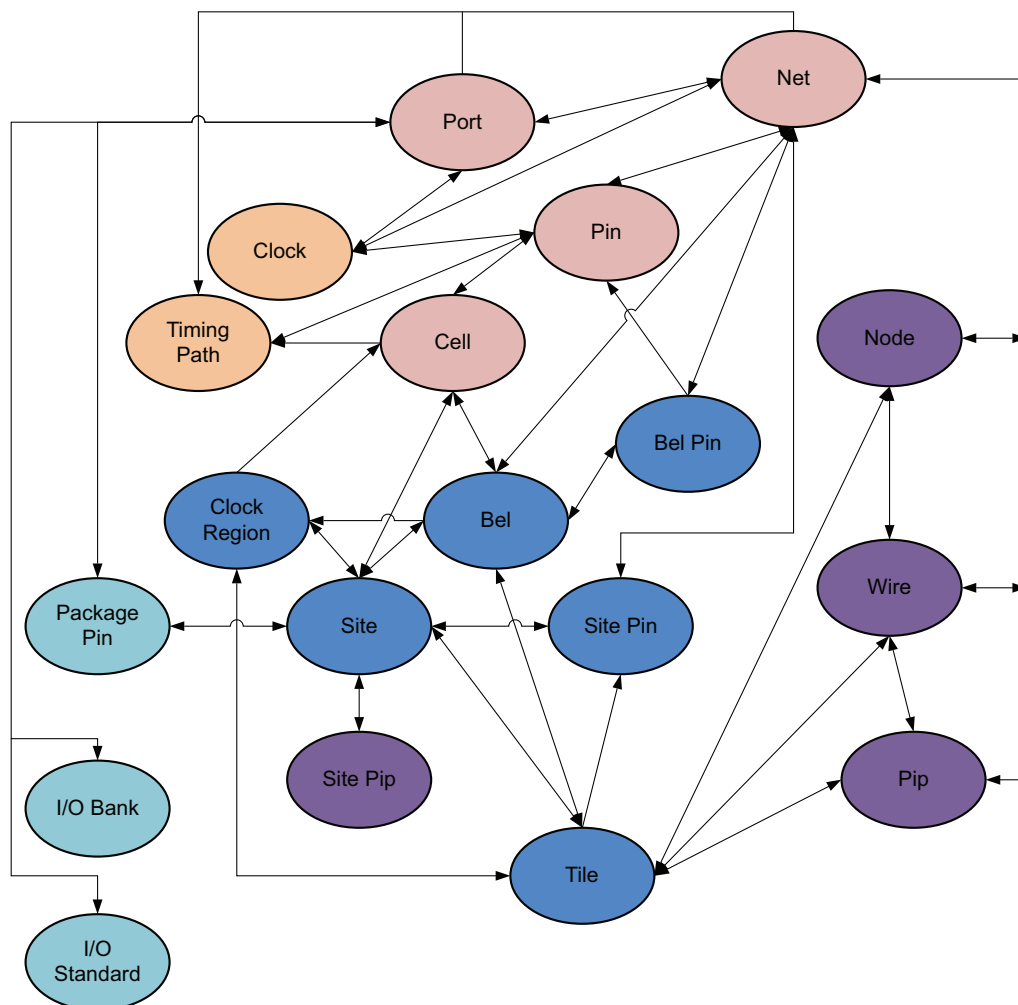


Figure 1-1: Vivado First Class Objects

The objects displayed in [Figure 1-1](#) are defined as netlist objects, or as device objects. Netlist objects, shown in pink above, include logic cells, pins, ports, and nets. Device objects include placement sites, such as clock regions, tiles, sites, and bels, shown in blue above. Device objects also include package pins and I/O banks, shown in green, and routing resources such as nodes, wires, and pips, shown in purple in [Figure 1-1](#).

The relationship between objects is shown by the arrows connecting two objects.

A double headed arrow indicates that the relationship can be queried from either direction. For instance, you can query the cells attached to specific nets (**get_cell -of_objects [get_nets]**), or query the nets connected to specific cells (**get_nets -of_objects [get_cells]**).

A single-ended arrow reflects a relationship that can only be queried in the direction of the arrow. For instance, in [Figure 1-1, page 6](#), you can see that you can query the bels located in specific clock regions (**get_bels -of_objects [get_clock_regions]**), but you cannot get clock regions associated with specific bels. You can get tiles associated with specific bels (**get_tiles -of_objects [get_bels]**), but not bels associated with tiles.

This figure is representative, and is not intended to depict all Vivado first class objects, or all their relationships. A description of first class objects, their relationships to other objects, and the properties defined on those objects follows in this chapter.

Additional categories of objects exist in the Vivado Design Suite, such as timing objects, which combine with the netlist design to create preliminary timing reports. Timing objects associated with the netlist and device objects, provide a complete timing analysis of the implemented design. Timing objects include clocks, timing paths, and delay objects.

Copying Examples from this Document



CAUTION! *Please read this section carefully before copying syntax or coding examples from this document into your code.*

This guide gives numerous syntax and coding examples to assist you in inserting properties into your code. Problems may arise if you copy those examples directly from this PDF document into your code.

- The dash character, '-', may be replaced with an en-dash or em-dash character when copying and pasting from the PDF into the Vivado Tcl console, or into a Tcl script or XDC file.
- PDF documents insert end of line markers into examples that wrap from line to line. These markers will cause errors in your Tcl scripts or XDC files.
- Copying examples that span more than one page in the PDF captures extraneous header and footer information along with the example. This extraneous information causes errors in your TCL scripts or XDC files.

To avoid these problems, edit the example in an ASCII text editor to remove any unnecessary markers or information, then paste it into your code, or the Vivado Design Suite Tcl shell or Tcl console.

BEL

Description

Typically a BEL, or Basic Element, corresponds to leaf-cell in the netlist view of the design. BELs are device objects on the target Xilinx FPGA on which to place, or map, basic netlist objects like flip-flops, LUTs, and carry logic.

BELs are grouped together on the device in [SITE](#) objects, such as SLICES and IO Blocks (IOBs). One or more BELs can be located in a single SITE, and you can use the BEL to assign logic from the design netlist into specific locations or device resources on the target device.

The different TYPEs of BELs are enumerated below:

Table 1-1: BELs by TYPE

BEL Types		
• BSCAN_BSCAN	• ILOGICE2_IFF	• MMCME2_ADV_MMCME2_ADV
• BUFFER	• ILOGICE3_IFF	• ODELAYE2_ODELAYE2
• BUFG_BUFG	• ILOGICE3_ZHOLD_DELAY	• OLOGICE2_MISR
• BUFHCE_BUFHCE	• INVERTER	• OLOGICE2_OUTFF
• BUFGIO_BUFGIO	• IN_FIFO_IN_FIFO	• OLOGICE2_TFF
• BUFMRCE_BUFMRCE	• IOB18M_INBUF_DCIEN	• OLOGICE3_MISR
• BUFR_BUFR	• IOB18M_OUTBUF_DCIEN	• OLOGICE3_OUTFF
• CAPTURE_CAPTURE	• IOB18M_TERM_OVERRIDE	• OLOGICE3_TFF
• CARRY4	• IOB18S_INBUF_DCIEN	• OUT_FIFO_OUT_FIFO
• DCIRESET_DCIRESET	• IOB18S_OUTBUF_DCIEN	• PAD
• DNA_PORT_DNA_PORT	• IOB18S_TERM_OVERRIDE	• PCIE_2_1_PCIE_2_1
• DSP48E1_DSP48E1	• IOB18_INBUF_DCIEN	• PHASER_IN_PHY_PHASER_IN_P HY
• EFUSE_USR_EFUSE_USR	• IOB18_OUTBUF_DCIEN	• PHASER_OUT_PHY_PHASER_OU T_PHY
• FF_INIT	• IOB18_TERM_OVERRIDE	• PHASER_REF_PHASER_REF
• FIFO18E1_FIFO18E1	• IOB33M_INBUF_EN	• PHY_CONTROL_PHY_CONTROL
• FRAME_ECC_FRAME_ECC	• IOB33M_OUTBUF	• PLLE2_ADV_PLLE2_ADV
• GTXE2_CHANNEL_GTXE2_CHA NNEL	• IOB33M_TERM_OVERRIDE	• PULL_OR_KEEP1
• GTXE2_COMMON_GTXE2_COM MON	• IOB33S_INBUF_EN	• RAMB18E1_RAMB18E1
• HARD0	• IOB33S_OUTBUF	• RAMBFIFO36E1_RAMBFIFO36E 1
• HARD1	• IOB33S_TERM_OVERRIDE	• REG_INIT
• IBUFDS_GTE2_IBUFDS_GTE2	• IOB33_INBUF_EN	• SELMUX2_1
• ICAP_ICAP	• IOB33_OUTBUF	• SLICEL_CARRY4_AMUX
• IDELAYCTRL_IDELAYCTRL	• IOB33_TERM_OVERRIDE	• SLICEL_CARRY4_AXOR

BEL Types

- IDELAYE2_FINEDELAY_IDELAYE2_FINEDELAY
- IDELAYE2_IDELAYE2
- LUT5
- LUT6
- LUT_OR_MEM5
- LUT_OR_MEM6
- STARTUP_STARTUP
- USR_ACCESS_USR_ACCESS
- XADC_XADC

Related Objects

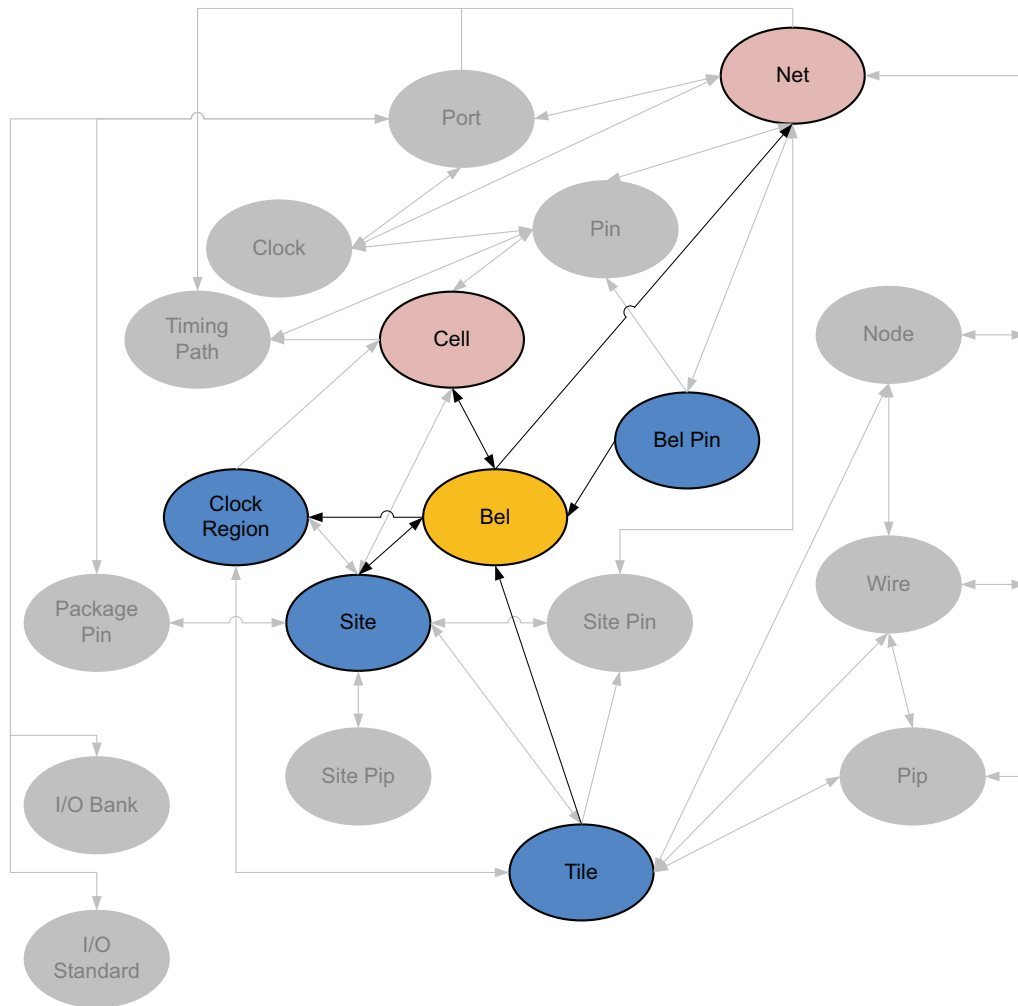


Figure 1-2: BEL Objects

As seen in Figure 1-2, leaf-level CELLS from the netlist design can be mapped onto BELs on the target part. BELs are grouped in SITES on the target Xilinx FPGA, and both BELs and SITES are grouped into CLOCK_REGIONS and TILES. Each BEL also has BEL_PINS that map to PINS on the cells, and are connection points to the NET netlist object.

Properties

Table 1-2 lists the properties found on BUFFER type BEL objects, as representative of the type of properties on BELs:

Table 1-2: BEL Properties: BUFFER

Name	Type	Property		Value
		Read Only	Visible	
CLASS	string	true	true	bel
IS_RESERVED	bool	true	true	0
IS_TEST	bool	true	true	0
IS_USED	bool	true	true	1
NAME	string	true	true	IPAD_XOY54/IPAD
NUM_BIDIR	int	true	true	0
NUM_CONFIGS	int	true	true	0
NUM_INPUTS	int	true	true	1
NUM_OUTPUTS	int	true	true	1
NUM_PINS	int	true	true	2
PROHIBIT	bool	true	true	0
TYPE	string	true	true	BUFFER

The properties assigned to BEL objects vary by TYPE. To report the properties for any of the TYPES of BEL listed above, you can use the **report_property** command:

```
report_property -all [lindex [get_bels -filter {TYPE == <BEL_TYPE>}] 0]
```

Where *<BEL_TYPE>* should be replaced by one of the listed BEL types. For example:

```
report_property -all [lindex [get_bels -filter {TYPE == SLICEM_CARRY4_AXOR}] 0]
report_property -all [lindex [get_bels -filter {TYPE == LUT5}] 0]
report_property -all [lindex [get_bels -filter {TYPE == IOB33S_OUTBUF}] 0]
```



TIP: The **report_property** command may return a warning that no objects were found if there are no related objects in the current design. Refer to the *Vivado Design Suite Tcl Command Reference (UG835)* [Ref 4] for more information on this command.

CELL

Description

A cell is an instance of a netlist logic object, which can either be a leaf-cell or a hierarchical cell. A leaf-cell is a primitive, or a primitive macro, with no further logic detail in the netlist. A hierarchical cell is a module or block that contains one or more additional levels of logic, and eventually concludes at leaf-cells.

There are different types of leaf-cell objects, defined by the `PRIMITIVE_GROUP`, `PRIMITIVE_SUBGROUP`, and `PRIMITIVE_TYPE` properties. All cells have a common set of properties, and each group or type may also have unique properties. The different groups, subgroups, and types of cells are enumerated below:

Table 1-3: Cells by PRIMITIVE_GROUP, PRIMITIVE_SUBGROUP, and PRIMITIVE_TYPE

Group	Subgroup	Type
BMEM	bram	RAMB18E1
		RAMB36E1
	fifo	FIFO36E1
CARRY	others	CARRY4
		MUXCY
CLK	gclk	BUFG
		MMCME2_ADV
	rclk	BUFHCE
DMEM	srl	SRL16E
FLOP_LATCH	flop	FDCE
		FDPE
		FDRE
		FDSE
IO	ddr	ODDR
	gt	GTXE2_CHANNEL
		ibuf
	obuf	IBUFDS
		IBUFDS_GTE2
		OBUF
LUT	others	LUT1
		LUT2
		LUT3

Table 1-3: Cells by PRIMITIVE_GROUP, PRIMITIVE_SUBGROUP, and PRIMITIVE_TYPE

Group	Subgroup	Type
		LUT4
		LUT5
		LUT6
MULT	dsp	DSP48E1
MUXFX	others	MUXF7
		MUXF8
OTHERS	others	GND
		VCC
		others
RTL_GATE	buf	RTL_INV
	logical	RTL_AND
		RTL_NAND
		RTL_NOR
		RTL_OR
		RTL_XNOR
		RTL_XOR
RTL_MEMORY	ram	RTL_RAM
	rom	RTL_ROM
RTL_MUX	mux	RTL_MUX
RTL_OPERATOR	arithmetic	RTL_ADD
		RTL_SUB
	equality	RTL_EQ
		RTL_NEQ
	relational	RTL_GEQ
		RTL_GT
		RTL_LEQ
		RTL_LT
	reduction	RTL_REDUCTION_OR
		RTL_REDUCTION_XOR
RTL_REGISTER	flop	RTL_REG
RTL_SPECIAL	others	RTL_BMERGE
		RTL_BSEL

Related Objects

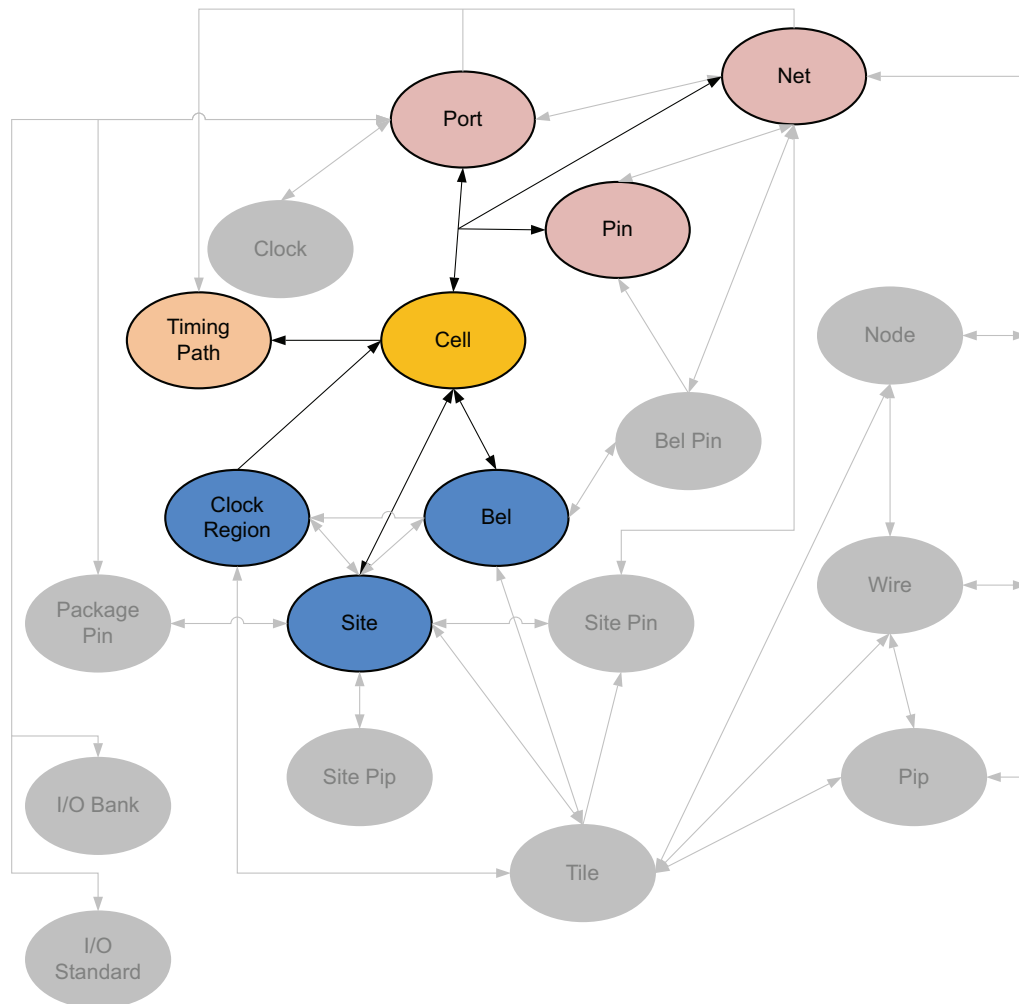


Figure 1-3: CELL Objects

CELLS can be hierarchical or leaf-cells, which are primitive. As seen in [Figure 1-3](#), leaf-cells have PINs which are connected to NETs to define the external netlist, and hierarchical cells contain PORTs that are associated with hierarchical PINs, and which connect internally to NETs to define the internal netlist.

CELLS are placed, or mapped, onto device resources on the target Xilinx FPGA. The CELL can be placed onto a BEL object in the case of basic logic such as flops, LUTs, and MUXes; or can be placed onto a SITE object in the case of larger logic cells such as BRAMs and DSPs. BELs are also collected into larger SITES, called SLICES, so a cell can be associated with a BEL and a SITE object. SITES are grouped into CLOCK_REGIONs and TILES.

CELLS are also associated with TIMING_PATHs in the design, and can be associated with DRC_VIOLATIONS to help you quickly locate and resolve design issues.

Properties:

The following table lists the properties that may be found on hierarchical CELL objects as representative of the properties found on all cells.

Table 1-4: CELL Properties

Hierarchical Cell Properties		
• ASYNC_REG	• IOB	• PRIMITIVE_SUBGROUP
• BEL	• IOBDELAY	• PWR_MODE
• BLKNM	• IODELAY_GROUP	• RAM_STYLE
• BOX_TYPE	• IS_BEL_FIXED	• REF_NAME
• bram_addr_begin	• IS_BLACKBOX	• REUSE_STATUS
• bram_addr_end	• IS_CLOCK_GATED	• RLOC
• bram_slice_begin	• IS_FIXED	• RLOC_ORIGIN
• bram_slice_end	• IS_IMPORTED	• RLOC_RANGE
• BUFG	• IS_LOC_FIXED	• ROM_STYLE
• CAPACITANCE	• IS_PRIMITIVE	• RPM
• CHECK_LICENSE_TYPE	• IS_REUSED	• RPM_GRID
• CLASS	• IS_SEQUENTIAL	• RTL_RAM_STYLE
• CONVERT_BRAM8	• keep	• SEL_VAL
• CORE_GENERATION_INFO	• KEEP_HIERARCHY	• SHREG_EXTRACT
• counter	• LIB_CELL	• SIM_COLLISION_CHECK
• DCI_VALUE	• LINE_NUMBER	• SITE
• DONT_TOUCH	• LOAD_VAL	• SOFT_HLUTNM
• ESSENTIAL_CLASSIFICATION_VALUE	• LOC	• srl_bus_name
• FILE_NAME	• LOCK_PINS	• srl_name
• FSM_ENCODING	• LUTNM	• TOOL_DERIVED_CLK_NAMES
• H_SET	• MACRO_NAME	• TOOL_INSERTED_BUFG
• HBLKNM	• MAP	• TRANSIENT_FILTER
• HD.ISOLATED	• MAX_FANOUT	• TYPE
• HD.PARTITION	• MEMDATA.SCOPE_BMM_FILE	• U_SET
• HD.RECONFIGURABLE	• METHODOLOGY_DRC_VIOS	• USE_DSP48
• HDPCBEL	• NAME	• USE_LUTNM
• HDPCLOC	• NODELAY	• USE_RLOC
• HIERARCHICALNAME	• ORIG_REF_NAME	• width
• HLUTNM	• PARENT	• XBLKNM
• HU_SET	• PBLOCK	• XILINX_LEGACY_PRIM
• IBUF_DELAY_VALUE	• POWER	• XILINX_TRANSFORM_PINMAP
• IMPORTED_FROM	• POWER_OPTED_CE	• XLNX_LINE_COL
• IMPORTED_TYPE	• PRIMITIVE_COUNT	• XLNX_LINE_FILE
• INIT_VAL	• PRIMITIVE_GROUP	• XSTLIB
• INV	• PRIMITIVE_LEVEL	

You can report the properties for specific types of CELL objects by filtering on the PRIMITIVE_GROUP and PRIMITIVE_TYPE property values. You can copy and paste the following commands into the command line of the Tcl console to extract the properties from the desired PRIMITIVE_TYPE.



TIP: The **report_property** command may return a warning that no objects were found if there are no related objects in the current design. Refer to the *Vivado Design Suite Tcl Command Reference (UG835)* [Ref 4] for more information on this command.

- HIERARCHICAL CELL

```
report_property -all [lindex [get_cells -hier -filter {!IS_PRIMITIVE}] 0]
```

- BMEM

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == BMEM.bram.RAMB18E1}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == BMEM.bram.RAMB36E1}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == BMEM.fifo.FIFO36E1}] 0]
```

- CARRY

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == CARRY.others.CARRY4}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == CARRY.others.MUXCY}] 0]
```

- CLK

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == CLK.gclk.BUFG}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == CLK.gclk.MMCME2_ADV}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == CLK.rclk.BUFHCE}] 0]
```

- DMEM

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == DMEM.srl.SRL16E}] 0]
```

- FLOP_LATCH

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == FLOP_LATCH.flop.FDCE}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == FLOP_LATCH.flop.FDPE}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == FLOP_LATCH.flop.FDRE}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE == FLOP_LATCH.flop.FDSE}] 0]
```

- IO

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
IO. DDR.ODDR}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
IO.gt.GTXE2_CHANNEL}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
IO.ibuf.IBUF}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
IO.ibuf.IBUFDS}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
IO.ibuf.IBUFDS_GTE2}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
IO.obuf.OBUF}] 0]
```

- LUT

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
LUT.others.LUT1}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
LUT.others.LUT2}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
LUT.others.LUT3}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
LUT.others.LUT4}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
LUT.others.LUT5}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
LUT.others.LUT6}] 0]
```

- MULT

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
MULT.dsp.DSP48E1}] 0]
```

- MUXFX

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
MUXFX.others.MUXF7}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
MUXFX.others.MUXF8}] 0]
```

- OTHERS

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
OTHERS.others.GND}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
OTHERS.others.VCC}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
others.others.others}] 0]
```

- RTL_GATE

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_GATE.buf.RTL_BUF}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_GATE.buf.RTL_INV}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_GATE.logical.RTL_AND}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_GATE.logical.RTL_NAND}] 0]
```

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_GATE.logical.RTL_NOR}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_GATE.logical.RTL_OR}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_GATE.logical.RTL_XNOR}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_GATE.logical.RTL_XOR}] 0]
```

- **RTL_MEMORY**

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_MEMORY.ram.RTL_RAM}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_MEMORY.rom.RTL_ROM}] 0]
```

- **RTL_MUX**

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_MUX.mux.RTL_MUX}] 0]
```

- **RTL_OPERATOR**

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_OPERATOR.arithmetic.RTL_ADD}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_OPERATOR.arithmetic.RTL_SUB}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_OPERATOR.equality.RTL_EQ}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_OPERATOR.equality.RTL_NEQ}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_OPERATOR.relational.RTL_GEQ}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_OPERATOR.relational.RTL_GT}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_OPERATOR.relational.RTL_LEQ}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_OPERATOR.relational.RTL_LT}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_OPERATOR.reduction.RTL_REDUCTION_OR}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_OPERATOR.reduction.RTL_REDUCTION_XOR}] 0]
```

- **RTL_REGISTER**

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_REGISTER.flop.RTL_REG}] 0]
```

- **RTL_SPECIAL**

```
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_SPECIAL.others.RTL_BMERGE}] 0]
report_property -all [lindex [get_cells -hier -filter {PRIMITIVE_TYPE ==
RTL_SPECIAL.others.RTL_BSEL}] 0]
```

NET

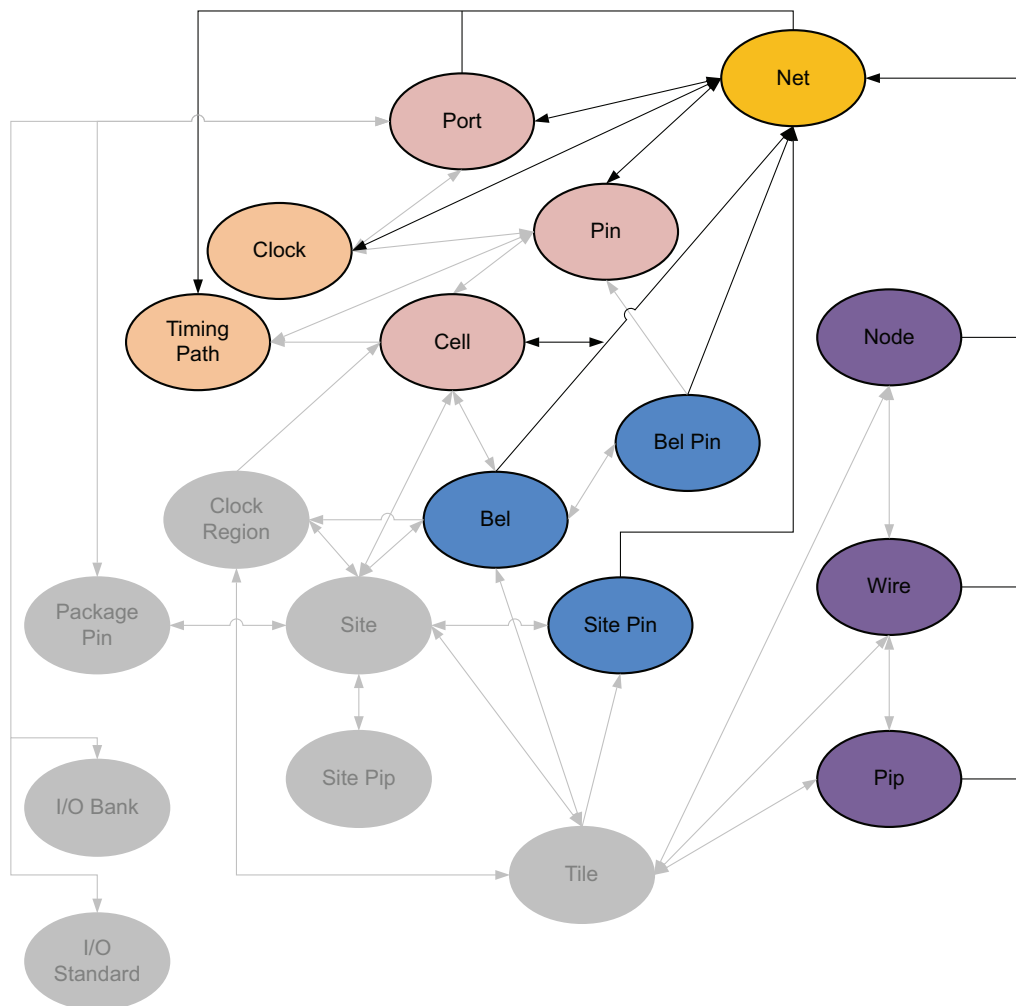


Figure 1-4: NET Objects

Description

A net is a set of interconnected pins, ports, and wires. Every wire has a net name, which identifies it. Two or more wires can have the same net name. All wires sharing a common net name are part of a single NET, and all pins or ports connected to these wires are electrically connected.

A default net name is assigned to the NET object as it is added to the netlist design during elaboration or compilation of the RTL source files into a netlist design. You can also manually assign names to nets.

Nets can either be scalar nets, with a single signal, or can be bus nets, which are groups of scalar nets with multiple signals. Buses are a convenient way to group related signals, allowing a less cluttered, more understandable schematics. It also clarifies the connection between the main circuit and a block symbol. Buses are especially useful for the following:

- Routing a number of signals from one side of the schematic to the other
- Connecting more than one signal to a block symbol
- Connecting more than one signal to pass between hierarchical levels by connecting to a single I/O marker

Related Objects:

In the design netlist, a NET can be connected to the PIN of a CELL, or to a PORT. As the design is mapped onto the target Xilinx FPGA, the NET is mapped to routing resources such as WIRES, NODEs, and PIPs on the device, and is connected to BELs through BEL_PINs, and to SITEs through SITE_PINs.

NETs are also associated with CLOCKS brought onto the design through PORTs, and to TIMING_PATHs in the design.

NETs can also be associated with DRC_VIOLATIONs to allow you to more quickly locate and resolve design issues.

Properties:

The properties assigned to a NET are as follows:

Table 1-5: Net Properties

Name	Type	Read-only	Visible	Value
AREA_GROUP	string	TRUE	TRUE	
BLKNM	string	TRUE	TRUE	
BUFFER_TYPE	enum	FALSE	TRUE	
BUFG	enum	TRUE	TRUE	
BUS_NAME	string	TRUE	TRUE	
BUS_START	int	TRUE	TRUE	
BUS_STOP	int	TRUE	TRUE	
BUS_WIDTH	int	TRUE	TRUE	
CLASS	string	TRUE	TRUE	net
CLOCK_BUFFER_TYPE	enum	FALSE	TRUE	
CLOCK_DEDICATED_ROUTE	enum	FALSE	TRUE	
CLOCK_REGION_ASSIGNMENT	string	FALSE	TRUE	
COLLAPSE	bool	TRUE	TRUE	
COOL_CLK	bool	TRUE	TRUE	
DATA_GATE	bool	TRUE	TRUE	

DCI_VALUE	int	FALSE	TRUE	
DIFF_TERM	bool	FALSE	TRUE	
DONT_TOUCH	bool	FALSE	TRUE	
DRIVE	int	TRUE	FALSE	
DRIVER_COUNT	int	TRUE	TRUE	1
ESSENTIAL_CLASSIFICATION_VALUE	int	FALSE	TRUE	
FILE_NAME	string	TRUE	TRUE	
FLAT_PIN_COUNT	int	TRUE	TRUE	2
FLOAT	bool	TRUE	TRUE	
GATED_CLOCK	bool	FALSE	TRUE	
HBLKNM	string	TRUE	TRUE	
HIERARCHICALNAME	string	TRUE	FALSE	wave_gen.dac_spi_i0.active0_out
HU_SET	string	TRUE	FALSE	
IBUF_DELAY_VALUE	double	TRUE	TRUE	
IBUF_LOW_PWR	bool	FALSE	TRUE	
IFD_DELAY_VALUE	double	TRUE	TRUE	
IN_TERM	enum	TRUE	TRUE	
IOB	enum	FALSE	TRUE	
IOBDELAY	enum	FALSE	TRUE	
IOSTANDARD	string	TRUE	FALSE	
IO_BUFFER_TYPE	enum	FALSE	TRUE	
IS_CONTAIN_ROUTING	bool	TRUE	TRUE	0
IS_REUSED	bool	TRUE	TRUE	0
KEEP	bool	TRUE	TRUE	
KEEPER	bool	TRUE	TRUE	
LINE_NUMBER	int	TRUE	TRUE	
LOC	string	TRUE	TRUE	
MARK_DEBUG	bool	FALSE	TRUE	0
MAXDELAY	double	TRUE	TRUE	
MAXSKEW	double	TRUE	TRUE	
MAX_FANOUT	string	FALSE	TRUE	
METHODOLOGY_DRC_VIOS	string	FALSE	TRUE	
NAME	string	TRUE	TRUE	dac_spi_i0/active0_out
NODELAY	bool	TRUE	TRUE	
NOREDUCE	bool	TRUE	TRUE	
OUT_TERM	enum	TRUE	TRUE	
PARENT	string	TRUE	TRUE	dac_spi_i0/active0_out
PARENT_CELL	cell	TRUE	TRUE	dac_spi_i0
PIN_COUNT	int	TRUE	TRUE	2
PULLDOWN	bool	TRUE	TRUE	
PULLUP	bool	TRUE	TRUE	

PWR_MODE	enum	TRUE	TRUE	
RAM_STYLE	enum	FALSE	TRUE	
REUSE_STATUS	enum	TRUE	TRUE	
RLOC	string	TRUE	TRUE	
RLOC_ORIGIN	string	TRUE	FALSE	
RLOC_RANGE	string	TRUE	FALSE	
ROM_STYLE	enum	FALSE	TRUE	
ROUTE_STATUS	enum	TRUE	TRUE	UNPLACED
RPM_GRID	enum	TRUE	TRUE	
RTL_KEEP	string	TRUE	FALSE	
RTL_MAX_FANOUT	string	TRUE	FALSE	
S	bool	TRUE	TRUE	
SCHMITT_TRIGGER	bool	TRUE	TRUE	
SLEW	string	TRUE	TRUE	
SUSPEND	string	TRUE	TRUE	
TYPE	enum	TRUE	TRUE	SIGNAL
USELOWSKEWLINES	bool	TRUE	TRUE	
USE_DSP48	enum	FALSE	TRUE	
U_SET	string	TRUE	FALSE	
WEIGHT	int	FALSE	TRUE	
WIREAND	bool	TRUE	TRUE	
XBLKNM	string	TRUE	TRUE	
XLNX_LINE_COL	int	FALSE	FALSE	

The properties of a NET object can be reported using the following command:

```
report_property -all [lindex [get_nets] 0]
```

PIN

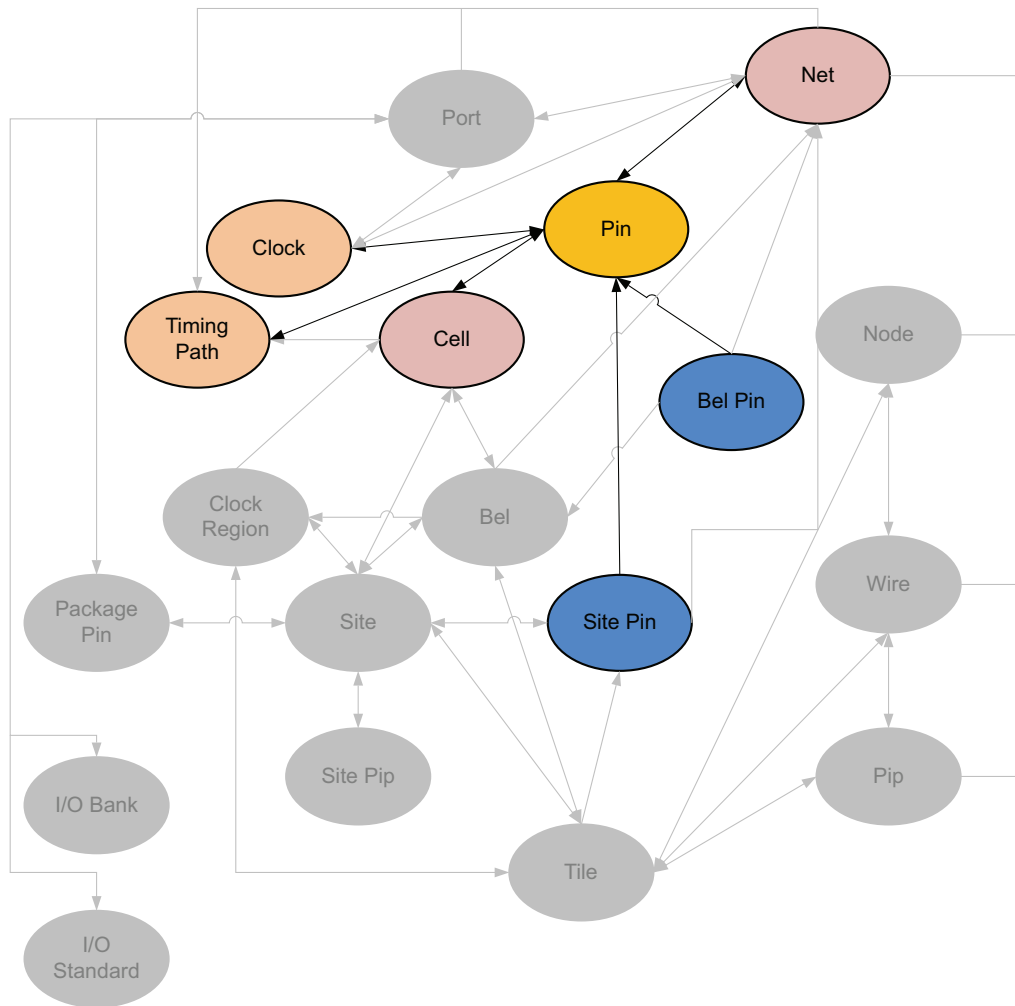


Figure 1-5: PIN Objects

Description

A pin is a point of logical connectivity on a primitive or hierarchical cell. A pin allows the contents of a cell to be abstracted away, and the logic simplified for ease-of-use. Pins can be scalar, containing a single connection, or can be defined as bus pins to group multiple signals together.

Related Objects

A pin is attached to a CELL and can be connected to PINs on other cells by a NET. The pins of cells are also related to the BEL PINs of the BEL object, or SITE PINS of a SITE that the cell is mapped to. Pins are associated with CLOCKS as part of the clock domain, and are part of TIMING PATHs when defined as the start point, end point, or through point of the path.

PINs can also be associated with DRC_VIOLATIONs to allow you to more quickly locate and resolve design issues.

Properties

The various properties found on PIN objects include the following:

Table 1-6: Pin Properties

Property	Type	Read-only	Visible	Value
BEL	string	FALSE	TRUE	
BUS_DIRECTION	enum	TRUE	TRUE	
BUS_NAME	string	TRUE	TRUE	
BUS_START	int	TRUE	TRUE	
BUS_STOP	int	TRUE	TRUE	
BUS_WIDTH	int	TRUE	TRUE	
CLASS	string	TRUE	TRUE	pin
CLOCK_DEDICATED_ROUTE	enum	FALSE	TRUE	
DCI_VALUE	int	FALSE	TRUE	
DIRECTION	enum	TRUE	TRUE	OUT
ESSENTIAL_CLASSIFICATION_VALUE	int	FALSE	TRUE	
FB_ACTIVE	bool	FALSE	TRUE	
HD.ASSIGNED_PPLOCS	string*	TRUE	TRUE	
HD.CLK_SRC	string	FALSE	TRUE	
HD.LOC_FIXED	bool	FALSE	FALSE	0
HD.PARTPIN_LOCS	string*	FALSE	TRUE	
HD.PARTPIN_RANGE	string*	FALSE	TRUE	
HIERARCHICALNAME	string	TRUE	FALSE	IBUF_Ib_sel_i0.O
HOLD_DETOUR	int	FALSE	TRUE	
HOLD_SLACK	double	TRUE	TRUE	1E+39
IS_CLEAR	bool	TRUE	TRUE	0
IS_CLOCK	bool	TRUE	TRUE	0
IS_CONNECTED	bool	TRUE	TRUE	1
IS_ENABLE	bool	TRUE	TRUE	0
IS_INVERTED	bool	FALSE	TRUE	0
IS_LEAF	bool	TRUE	TRUE	1
IS_PRESET	bool	TRUE	TRUE	0
IS_RESET	bool	TRUE	TRUE	0
IS_REUSED	bool	TRUE	TRUE	0

IS_SETRESET	bool	TRUE	TRUE	0
LOGIC_VALUE	string	TRUE	TRUE	unknown
NAME	string	TRUE	TRUE	IBUF_l0_sel_i0/O
PARENT_CELL	cell	TRUE	TRUE	IBUF_l0_sel_i0
REF_NAME	string	TRUE	TRUE	IBUF
REF_PIN_NAME	string	TRUE	TRUE	O
SETUP_SLACK	double	TRUE	TRUE	1E+39
TARGET_SITE_PINS	string*	FALSE	TRUE	
XLNX_LINE_COL	int	FALSE	FALSE	
XLNX_LINE_FILE	long	FALSE	FALSE	

The properties of pins can be listed with the following command:

```
report_property -all [lindex [get_pins] 0 ]
```

PORT

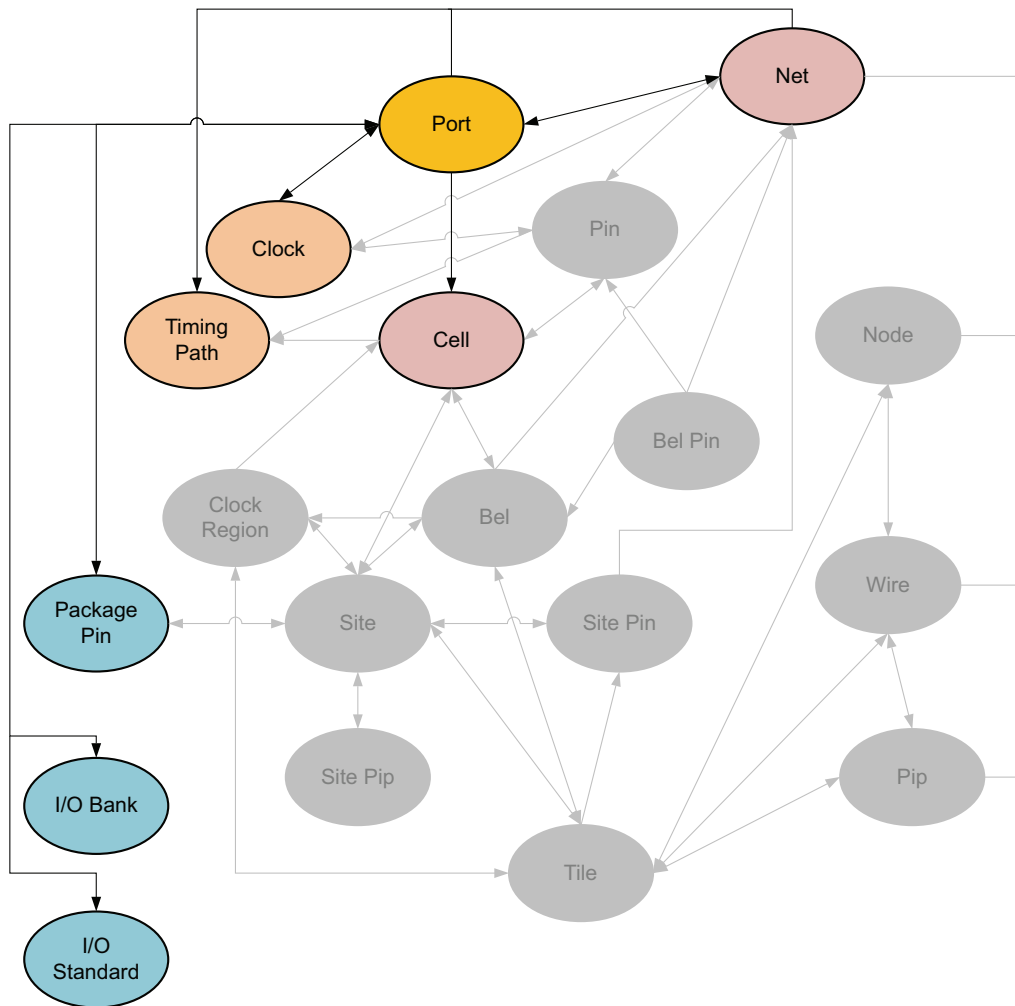


Figure 1-6: PORT Objects

Description

A PORT is a special type of hierarchical pin, providing an external connection point at the top-level of a hierarchical design, or an internal connection point in a hierarchical cell or block module to connect the internal logic to the pins on the hierarchical cell. Ports can be scalar, containing a single connection, or can be bus ports to group multiple signals together.

Related Objects

PORTS at the top level of the design make connection outside the FPGA through IOBANKs on the die, with assigned IOSTANDARDS, and through the PACKAGE_PINS of the device package.

PORTs can also carry CLOCK definitions onto the design from the system or BOARD, and should be assigned to TIMING_PATHs using the **set_input_delay** or **set_output_delay** constraint. Refer to the Vivado Design Suite User Guide: Using Constraints (UG903) for more information on these constraints.

Inside hierarchical cells, PORTs are assigned to CELLS, and connect to NETs inside the cell, the build the hierarchical netlist.

Properties

The properties found on PORT objects is as follows:

Table 1-7: Port Properties

Property	Type	Read-only	Visible	Value
BOARD_PIN	string	FALSE	TRUE	
BUFFER_TYPE	enum	FALSE	TRUE	
BUS_DIRECTION	enum	TRUE	TRUE	
BUS_NAME	string	TRUE	TRUE	
BUS_START	int	TRUE	TRUE	
BUS_STOP	int	TRUE	TRUE	
BUS_WIDTH	int	TRUE	TRUE	
CLASS	string	TRUE	TRUE	port
CLOCK_BUFFER_TYPE	enum	FALSE	TRUE	
DIFFTERMTYPE	bool	FALSE	FALSE	0
DIFF_PAIR_PORT	port	TRUE	TRUE	clk_pin_p
DIFF_PAIR_TYPE	enum	TRUE	TRUE	N
DIFF_TERM	bool	FALSE	TRUE	0
DIRECTION	enum	FALSE	TRUE	IN
DQS_BIAS	enum	FALSE	TRUE	
DRIVE	enum	FALSE	TRUE	0
DRIVE_STRENGTH	enum	FALSE	FALSE	0
ESSENTIAL_CLASSIFICATION_VALUE	int	FALSE	TRUE	
HD.ASSIGNED_PPLOCS	string*	TRUE	TRUE	
HD.CLK_SRC	string	FALSE	TRUE	
HD.LOC_FIXED	bool	FALSE	FALSE	0
HD.PARTPIN_LOCS	string*	FALSE	TRUE	
HD.PARTPIN_RANGE	string*	FALSE	TRUE	
HOLD_SLACK	double	TRUE	TRUE	1E+39
IBUF_LOW_PWR	bool	FALSE	TRUE	1

INTERFACE	string	FALSE	TRUE	
INTERMTYPE	enum	FALSE	FALSE	NONE
IN_TERM	enum	FALSE	TRUE	NONE
IOB	enum	FALSE	TRUE	
IOBANK	int	TRUE	TRUE	33
IOSTANDARD	enum	FALSE	TRUE	LVDS
IOSTD	enum	FALSE	FALSE	LVDS
IO_BUFFER_TYPE	enum	FALSE	TRUE	
IS_BEL_FIXED	bool	FALSE	FALSE	1
IS_FIXED	bool	FALSE	FALSE	1
IS_GT_TERM	bool	TRUE	TRUE	0
IS_LOC_FIXED	bool	FALSE	TRUE	1
IS_REUSED	bool	TRUE	TRUE	
KEEPER	bool	FALSE	TRUE	0
LOAD	double	FALSE	TRUE	
LOC	site	FALSE	TRUE	IOB_X1Y75
LOGIC_VALUE	string	TRUE	TRUE	unknown
NAME	string	FALSE	TRUE	clk_pin_n
OFFCHIP_TERM	string	FALSE	TRUE	NONE
OUT_TERM	enum	FALSE	TRUE	
PACKAGE_PIN	package_pin	FALSE	TRUE	AD11
PIN_TYPE	enum	TRUE	FALSE	
PIO_DIRECTION	enum	FALSE	TRUE	
PULLDOWN	bool	FALSE	TRUE	0
PULLTYPE	string	FALSE	FALSE	
PULLUP	bool	FALSE	TRUE	0
SETUP_SLACK	double	TRUE	TRUE	1E+39
SITE	site	FALSE	FALSE	IOB_X1Y75
SLEW	enum	FALSE	TRUE	
SLEWTYPE	enum	FALSE	FALSE	
UNCONNECTED	bool	TRUE	TRUE	0
USE_INTERNAL_VREF	enum	FALSE	TRUE	
VCCAUX_IO	enum	FALSE	TRUE	
XLNX_LINE_COL	int	FALSE	FALSE	
XLNX_LINE_FILE	long	FALSE	FALSE	131072
x_interface_info	string	FALSE	TRUE	

The properties of ports can be listed with the following command:

```
report_property -all [lindex [get_ports] 0 ]
```


SLICEMs can be configured to act as distributed RAM. Distributed Memory is a configuration feature of certain LUTs so it behaves as a small 64-bit memory. SLICEL LUTs can only function as logic and not memory

SITEs also include diverse objects such as block RAM, DSPs, I/O blocks, Clock resources, and GT blocks.

You utilize slice resources by inference from the HDL source by Vivado synthesis, or by instantiating a primitive or macro from the FPGA library, or an IP core from the Vivado IP catalog. The Libraries Guide describes the list of primitives that can be instantiated.

The different SITE_TYPES are as follows:

Table 1-8: Available SITE Types

SITE Types		
• BSCAN	• GTPE2_COMMON	• ODELAYE2_FINEDELAY
• BSCAN_JTAG_MONE2	• GTXE2_CHANNEL	• OLOGICE2
• BUFG	• GTXE2_COMMON	• OLOGICE3
• BUFGCTRL	• GTZE2_OCTAL	• OPAD
• BUFG_LB	• IBUFDS_GTE2	• OSERDESE2
• BUFGHCE	• ICAP	• OUT_FIFO
• BUFIO	• IDELAYCTRL	• PCIE_2_1
• BUFMRCE	• IDELAYE2	• PCIE_3_0
• BUFR	• IDELAYE2_FINEDELAY	• PHASER_IN
• CAPTURE	• ILOGICE2	• PHASER_IN_ADV
• CFG_IO_ACCESS	• ILOGICE3	• PHASER_IN_PHY
• DCI	• IN_FIFO	• PHASER_OUT
• DCIRESET	• IOB18	• PHASER_OUT_ADV
• DNA_PORT	• IOB18M	• PHASER_OUT_PHY
• DRP_AMS_ADC	• IOB18S	• PHASER_REF
• DRP_AMS_DAC	• IOB33	• PHY_CONTROL
• DSP48E1	• IOB33M	• PLLE2_ADV
• EFUSE_USR	• IOB33S	• RAMB18E1
• FIFO18E1	• IOBM	• RAMB36E1
• FIFO36E1	• IOBS	• RAMBFIFO36E1
• FRAME_ECC	• IPAD	• SLICEL
• GCLK_TEST_BUF	• ISERDESE2	• SLICEM
• GLOBALSIG	• KEY_CLEAR	• STARTUP
• GTHE2_CHANNEL	• MMCME2_ADV	• TIEOFF
• GTHE2_COMMON	• MTBF2	• USR_ACCESS
• GTPE2_CHANNEL	• ODELAYE2	• XADC

Related Objects

As seen in [Figure 1-7, page 29](#), SITES are related to many different netlist and device objects. Leaf-CELLs flops and latches are mapped to BELs which are in turn mapped to SITES like SLICES, or are mapped directly to SITES such as BRAMs and DSPs. BELs and SITES are grouped into TILES, and are assigned to CLOCK_REGIONs on the device.

PORTs, PINs, IO Banks, and Package Pins relate to IO blocks (IOBs) which are also SITES. Further, SITES have pins, or SITE PINs, that map to NODEs, PINs, and NETs.

Properties

There are over 80 different SITE types on Xilinx FPGA devices, but they all share the following properties:

Table 1-9: SITE Properties

Property	Type	Read-only	Visible	Value
ALTERNATE_SITE_TYPES	string	TRUE	TRUE	
CLASS	string	TRUE	TRUE	site
CLOCK_REGION	string	TRUE	TRUE	X0Y6
IS_BONDED	bool	TRUE	TRUE	0
IS_CLOCK_BUFFER	bool	TRUE	TRUE	0
IS_CLOCK_PAD	bool	TRUE	TRUE	0
IS_GLOBAL_CLOCK_BUFFER	bool	TRUE	TRUE	0
IS_GLOBAL_CLOCK_PAD	bool	TRUE	TRUE	0
IS_PAD	bool	TRUE	TRUE	0
IS_REGIONAL_CLOCK_BUFFER	bool	TRUE	TRUE	0
IS_REGIONAL_CLOCK_PAD	bool	TRUE	TRUE	0
IS_RESERVED	bool	TRUE	TRUE	0
IS_TEST	bool	TRUE	TRUE	0
IS_USED	bool	TRUE	TRUE	0
MANUAL_ROUTING	string	FALSE	TRUE	
NAME	string	TRUE	TRUE	SLICE_X2Y349
NUM_ARCS	int	TRUE	TRUE	153
NUM_BELS	int	TRUE	TRUE	32
NUM_INPUTS	int	TRUE	TRUE	37
NUM_OUTPUTS	int	TRUE	TRUE	13
NUM_PINS	int	TRUE	TRUE	50
PRIMITIVE_COUNT	int	TRUE	TRUE	0
PROHIBIT	bool	FALSE	TRUE	0
PROHIBIT_FROM_PERSIST	bool	TRUE	TRUE	0
RPM_X	int	TRUE	TRUE	21
RPM_Y	int	TRUE	TRUE	698
SITE_PINS	string	FALSE	TRUE	
SITE_TYPE	enum	TRUE	TRUE	SLICEM

The properties assigned to SITE objects are the same for all SITE_TYPES. To report the properties for any of the SITE_TYPES listed in [Table 1-8, page 30](#), you can use the **report_property** command:

```
report_property -all [lindex [get_sites -filter {SITE_TYPE == <SITE_TYPE>}] 0]
```

Where *<SITE_TYPE>* should be replaced by one of the listed SITE types. For example:

```
report_property -all [lindex [get_sites -filter {SITE_TYPE == DSP48E1}] 0]  
report_property -all [lindex [get_sites -filter {SITE_TYPE == RAMB36E1}] 0]  
report_property -all [lindex [get_sites -filter {SITE_TYPE == IBUFDS_GTE2}] 0]
```

Key Property Descriptions

Properties Information

This chapter provides information about Xilinx® Vivado™ Design Suite properties. The entry for each property contains the following information, where applicable:

- A **Description** of the property, including its primary uses.
- The Xilinx FPGA device **Architectures** supporting the property.
- The **Applicable Objects** or device resources supporting the property.
- Possible **Values** that can be assigned to the property.
- **Syntax** examples, including examples for **Verilog**, **VHDL**, and **XDC**.
- The **Affected Steps** in the design flow that are affected by the property.
- Cross references to related properties you should **See Also**.



IMPORTANT: *When a property is defined in both HDL code and as a constraint in the XDC file, the XDC property takes precedence and overrides the HDL property.*

For more information on the use of these properties within the Vivado Design Suite, refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 6].

ASYNC_REG

ASYNC_REG specifies that:

- A register can receive asynchronous data on the D input pin relative to its source clock.
or
- The register is a synchronizing register within a synchronization chain.

During simulation, when a timing violation occurs, the default behavior is for a register element to output an 'X', or unknown state (not a 1 or 0). When this happens, anything that element drives will see an 'X' on its input and in turn enters an unknown state. This condition can propagate through the design, in some cases causing large sections of the design to become unknown, and sometimes the simulator can not recover from this state. ASYNC_REG modifies the register to output the last known value even though a timing violation occurs.

Specifying ASYNC_REG also affects optimization, placement, and routing to improve Mean Time Between Failure (MTBF) for registers that may go metastable. If ASYNC_REG is applied, the placer will ensure the flip-flops on a synchronization chain are placed closely to maximize MTBF. Registers with ASYNC_REG that are directly connected will be grouped and placed together into a single SLICE, assuming they have a compatible control set and the number of registers does not exceed the available resources of the SLICE.

Note: ASYNC_REG also affects inference behavior in Vivado synthesis because the tool will not optimize registers or surrounding logic in ways which could reduce MTBF.

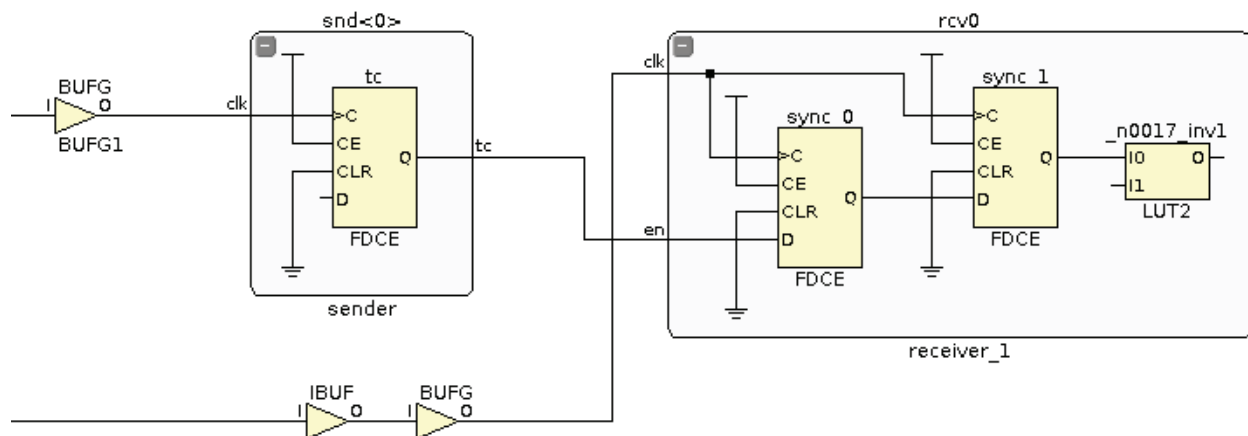


Figure 2-1: Synchronizing Clock Domains

The following is a Verilog example of a two FF, or one-stage synchronizer, as shown in [Figure 2-1, page 34](#). The registers synchronize a value from a separate clock domain. The ASYNC_REG property is attached to synchronizing stages with a value of TRUE:

```
(* ASYNC_REG = "TRUE" *) reg sync_0, sync_1;

always @(posedge clk) begin
  sync_1 <= sync_0;
  sync_0 <= en;
  . . .
end
```

With the ASYNC_REG property, the registers are grouped so that they are placed as close together as possible.:

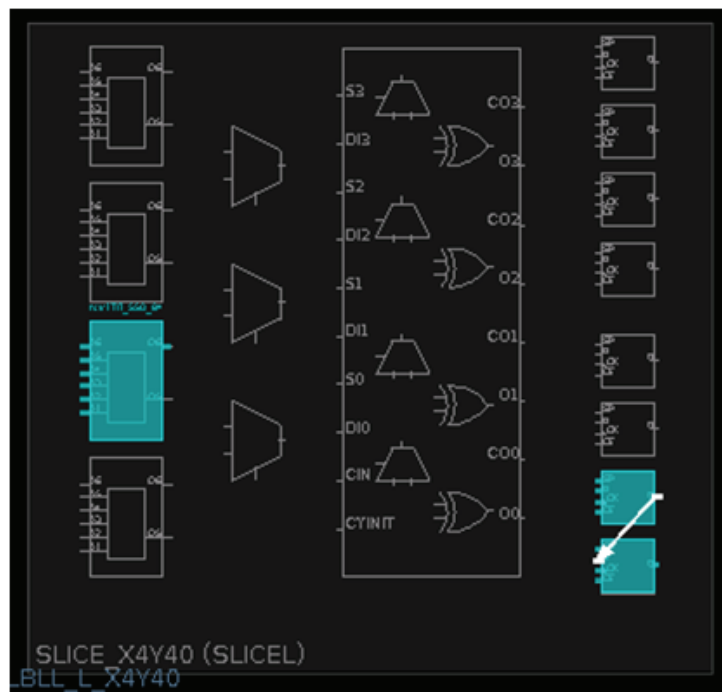


Figure 2-2: Grouping Registers

Architecture Support

All architectures

Applicable Objects

- Cells (`get_cells`)
 - Registers (FD, FDCE, FDPE, FDRE, FDSE)

Values

- FALSE (default)

The register can be optimized away, or absorbed into a block such as SRL, DSP, or RAMB. No special simulation, placement, or routing rules will be applied to it.

- TRUE

The register is part of a synchronization chain. It will be preserved through implementation, placed near the other registers in the chain and used for MTBF reporting.

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the instantiation or reg declaration of a register:

```
(* ASYNC_REG = "{TRUE|FALSE}" *)
```

Verilog Syntax Example

```
// Designates sync_regs as receiving asynchronous data  
(* ASYNC_REG = "TRUE" *) reg [2:0] sync_regs;
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute ASYNC_REG : string;
```

Specify the VHDL attribute as follows:

```
attribute ASYNC_REG of name: label is "{TRUE|FALSE}";
```

Where

- **name** is either:
 - The instance name of an instantiated register
 - or
 - The declared signal that will be inferred to a register

VHDL Syntax Example

```
attribute ASYNC_REG : string;
signal sync_regs : std_logic_vector(2 downto 1);
-- Designates sync_regs as receiving asynchronous data
attribute ASYNC_REG of sync_regs: label is "TRUE";
```

XDC Syntax

```
set_property ASYNC_REG value [get_cells instance_name]
```

Where

- **instance_name** is a register cell.

XDC Syntax Example

```
# Designates sync_regs as receiving asynchronous data
set_property ASYNC_REG TRUE [get_cells sync_regs*]
```

Affected Steps

- launch_xsim
- synth_design
- place_design
- route_design
- phys_opt_design
- power_opt_design
- report_drc
- write_verilog
- write_vhdl

BEL

BEL specifies a specific placement within a slice for a register or LUT. BEL is generally used with an associated LOC property to specify the exact placement of a register or LUT.

Architecture Support

All architectures

Applicable Objects

- Cells (`get_cells`)
 - Register (FD, FDCE, FDPE, FDRE, FDSE)
 - LUT (LUT1, LUT2, LUT3, LUT4, LUT5, LUT6, LUT6_2)
 - SRL (SRL16E, SRLC32E)
 - LUTRAM (RAM32X1S, RAM64X1S)

Values

- BEL = *<name>*

BEL names can take many different forms depending on the specific logic contents of the BEL. BEL names can also hierarchically include the SITE name for the BEL. For instance, some valid BEL names are BSCAN_X0Y0/BSCAN, IPAD_X0Y54/IPAD, BUFGCTRL_X0Y16/BUFG, and SLICE_X1Y199/A5FF.

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the instantiation of a LUT or register. The Verilog attribute can also be placed before the `reg` declaration of an inferred register, SRL, or LUTRAM.

```
(* BEL = "site_name" *)
```

Verilog Syntax Example

```
// Designates placed_reg to be placed in FF site A5FF  
(* BEL = "A5FF" *) reg placed_reg;
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute BEL : string;
```

For an instantiated instance, specify the VHDL attribute as follows:

```
attribute BEL of instance_name : label is "site_name";
```

Where

- **instance_name** is the instance name of an instantiated register, LUT, SRL, or LUTRAM.

VHDL Syntax Example

```
-- Designates instantiated register instance placed_reg to be placed in FF site A5FF  
attribute BEL of placed_reg : label is "A5FF";
```

For an inferred instance, specify the VHDL attribute as follows:

```
attribute BEL of signal_name : signal is "site_name";
```

Where

- **signal_name** is the signal name of an inferred register, LUT, SRL, or LUTRAM.

VHDL Syntax Example

```
-- Designates instantiated register instance placed_reg to be placed in FF site A5FF  
attribute BEL of placed_reg : signal is "A5FF";
```

XDC Syntax

```
set_property BEL site_name [get_cells instance_name]
```

Where

- **instance_name** is a register, LUT, SRL, or LUTRAM instance.

XDC Syntax Example

```
# Designates placed_reg to be placed in FF site A5FF  
set_property BEL A5FF [get_cells placed_reg]
```

Affected Steps

- Design Floorplanning
- place_design

See Also

[LOC, page 97](#)

BUFFER_TYPE

By default, Vivado synthesis infers an input buffer and global clock buffer (IBUF/BUFG) combination for clocks ports, and input buffers for input ports. However, you can manually specify the BUFFER_TYPE property to over ride the default behavior of the Vivado synthesis tool.

Architecture Support

All architectures

Applicable Objects

- The BUFFER_TYPE attribute can be placed on any top-level port (**all_inputs, get_ports**).

Values

- **ibuf**: Specify this value on clock ports where the default IBUF/BUFG pair is not wanted. In this case only, the IBUF is inferred for the clock.
- **none**: Indicates that no input or output buffers are used. A none value on a clock port results in no buffers.

Syntax

Verilog Syntax

```
(* buffer_type = "none" *) input in1; //this will result in no buffers
(* buffer_type = "ibuf" *) input clk1; //this will result in a clock with no bufg
```

VHDL Syntax

```
entity test is port(
in1 : std_logic_vector (8 downto 0);
clk : std_logic;
out1 : std_logic_vector(8 downto 0));
attribute buffer_type : string;
attribute buffer_type of in1 : signal is "none";
end test;
```

XDC Syntax

The BUFFER_TYPE property can also be applied to port objects in the XDC constraints file:

```
set_property BUFFER_TYPE <value> [get_ports <port_name>]
```

Where:

- *<value>* specifies one of the valid values for BUFFER_TYPE.
- *<port_name>* specifies the port or ports to apply the property to.

Affected Steps

- Synthesis

CFGBVS

The 7 series devices support configuration interfaces with 3.3V, 2.5V, or 1.8V I/O. The configuration interfaces include the JTAG pins in bank 0, the dedicated configuration pins in bank 0, and the pins related to specific configuration modes in bank 14 and bank 15.

To support the appropriate configuration interface voltage on bank 0, the Configuration Bank Voltage Select pin (CFGBVS) must be set to VCC0 or GND in order to configure I/O Bank 0 for either 3.3V/2.5V or 1.8V operation, respectively. The CFGBVS is a logic input pin referenced between VCCO_0 and GND. When the CFGBVS pin is connected to the VCCO_0 supply, the I/O on bank 0 support operation at 3.3V or 2.5V during configuration. When the CFGBVS pin is connected to GND, the I/O in bank 0 support operation at 1.8V during configuration.

The CFGBVS pin setting determines the I/O voltage support for bank 0 at all times. On devices in which bank 14 and bank 15 are the HR bank type, the [CONFIG_VOLTAGE](#) property determines the I/O voltage support.



IMPORTANT: *When CFGBVS is set to GND for 1.8V I/O operation, the VCCO_0 supply and I/O signals to Bank 0 must be 1.8V (or lower) to avoid damage to the Xilinx FPGA.*

Refer to the *7 Series FPGAs Configuration User Guide (UG470)* [\[Ref 1\]](#) for more information on Configuration Bank Voltage Select.

The Report DRC command checks the CFGBVS and CONFIG_VOLTAGE properties to determine the compatibility of CONFIG_MODE setting on the current design.

Architecture Support

All architectures

Applicable Objects

- Designs (**current_design**, **get_designs**)

Values

- VCC0: Configure I/O Bank 0 for 3.3V/2.5V operation.
- GND: Configure I/O Bank 0 for 1.8V operation.

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property CFGBVS [VCCO | GND] [current_design]
```

XDC Syntax Example

```
# Configure I/O Bank 0 for 3.3V/2.5V operation
set_property CFGBVS VCCO [get_designs impl_1]
```

Affected Steps

- I/O Planning
- Report DRC
- write_bitstream

See Also

[CONFIG_MODE](#), page 47

[CONFIG_VOLTAGE](#), page 49

CLOCK_DEDICATED_ROUTE

The CLOCK_DEDICATED_ROUTE property indicates whether the clock placement rules for the target device should be strictly followed.

External user clocks must be brought into the FPGA on differential clock pin pairs called clock-capable inputs (CCIOS). These CCIOs provide dedicated, high-speed routing to the internal global and regional clock resources to guarantee timing of various clocking features. Refer to the *7 Series FPGAs Clocking Resources User Guide (UG472)* [Ref 3] for more information on clock placement rules.

The CLOCK_DEDICATED_ROUTE property is generally used when it becomes necessary to place clock components in such a way as to take clock routing off of the dedicated clock trees in the target FPGA, and use standard routing channels. If the dedicated routes are not available, setting CLOCK_DEDICATED_ROUTE to FALSE demotes a clock placement DRC from an *ERROR* to a *WARNING* when a clock source is placed in a sub-optimal location compared to its load clock buffer.



CAUTION! Setting *CLOCK_DEDICATED_ROUTE* to *False* may result in sub-optimal clock delays, resulting in potential timing and other issues.

Architecture Support

All architectures

Applicable Objects

- Nets (`get_nets`) connected to the input of a global clock buffer (BUFG, BUFGCE, BUFGMUX, BUGCTRL)

Values

- TRUE: Clock placement DRC violations are reported as an ERROR (the default).
- FALSE: Clock placement DRC violations are downgraded to a WARNING. This should be used anytime a clock component (such as a BUFG, MMCM, or PLL) is placed so that the dedicated fast clock route cannot be used.
- BACKBONE: You may need to use this value if you assign location constraints that violate basic clock placement rules, but is not generally recommended. Use this value when an MMCM or PLL is placed far from the source CCIO pin. The extra wire length will add delay to the timing path from the CCIO to the MMCM, which may not be completely removed by the MMCM or PLL feedback. Use BACKBONE if the design meets timing with the added delay.

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property CLOCK_DEDICATED_ROUTE [TRUE | FALSE | BACKBONE] [get_nets net_name]
```

Where

- **net_name** is the signal name connected to the input of a global clock buffer.

XDC Syntax Example

```
# Designates clk_net to have relaxed clock placement rules  
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_net]
```

Affected Steps

- place_design
- report_drc

CONFIG_MODE

The CONFIG_MODE property defines which device configuration mode or modes to use for pin allocations, DRC reporting, and bitstream generation.



IMPORTANT: *COMPATIBLE_CONFIG_MODES* property has been deprecated in the 2013.3 release, and is replaced by the CONFIG_MODE property.

Xilinx 7 series FPGAs can be configured by loading application-specific configuration data, or a bitstream, into internal memory through special configuration pins. There are two general configuration datapaths: a serial datapath used to minimize the device pins required, and parallel datapaths for higher performance configuration. The CONFIG_MODES property defines which modes are used for the current design. Refer to the *7 Series FPGAs Configuration User Guide (UG470)* [Ref 1] for more information on device configuration modes.

Architecture Support

All architectures

Applicable Objects

- Design (`current_design`)

Values

- S_SERIAL
- M_SERIAL
- S_SELECTMAP
- M_SELECTMAP
- B_SCAN
- S_SELECTMAP+READBACK
- M_SELECTMAP+READBACK
- B_SCAN+READBACK
- S_SELECTMAP32
- S_SELECTMAP32+READBACK
- S_SELECTMAP16
- S_SELECTMAP16+READBACK

- SPIx1
- SPIx2
- SPIx4
- BPI8
- BPI16

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property CONFIG_MODE <value> [current_design]
```

Where *value* specifies the configuration mode.

XDC Syntax Example

```
# Specify using Configuration Mode Serial Peripheral Interface, 4-bit width  
set_property COMPATIBLE_CONFIG_MODES {{Master SPI x4}} [current_design]
```

Affected Steps

- I/O Planning
- place_design
- report_drc
- write_bitstream

CONFIG_VOLTAGE

The 7 series devices support configuration interfaces with 3.3V, 2.5V, or 1.8V I/O. The configuration interfaces include the JTAG pins in bank 0, the dedicated configuration pins in bank 0, and the pins related to specific configuration modes in bank 14 and bank 15. You can set the CONFIG_VOLTAGE property, or VCCO_0 voltage, to 3.3, 2.5, 1.8, or 1.5.

CONFIG_VOLTAGE must be set to the correct configuration voltage, in order to determine the I/O voltage support for the pins in bank 0. Refer to the *7 Series FPGAs Configuration User Guide (UG470)* [Ref 1] for more information on Configuration Voltage.

The CFGBVS pin setting determines the I/O voltage support for bank 0 at all times. For devices in which bank 14 and bank 15 are the HR bank type, the respective CONFIG_VOLTAGE property determines the I/O voltage support.

Report DRC checks are run on Bank 0, 14, and 15 to determine compatibility of CONFIG_MODE settings on the current design. DRCs are issued based on IOSTANDARD and CONFIG_VOLTAGE settings for the bank. The configuration voltages are also used when exporting IBIS models.

Architecture Support

All architectures

Applicable Objects

- Designs (**current_design**, **get_designs**)

Values

- 1.5, 1.8, 2.5, or 3.3

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property CONFIG_VOLTAGE {1.5 | 1.8 | 2.5 | 3.3} [current_design]
```

XDC Syntax Example

```
# Configure I/O Bank 0 for 3.3V/2.5V operation  
set_property CONFIG_VOLTAGE 1.8 [get_designs impl_1]
```

Affected Steps

- place_design
- report_drc
- write_bitstream

See Also

[CFGBVS](#), page 43

[CONFIG_MODE](#), page 47

DCI_CASCADE

DCI_CASCADE defines a master-slave relationship between a set of high-performance (HP) I/O banks. The digitally controlled impedance (DCI) reference voltage is chained from the master I/O bank to the slave I/O banks.

DCI_CASCADE specifies which adjacent banks use the DCI Cascade feature, thereby sharing reference resistors with a master bank. If several I/O banks in the same I/O bank column are using DCI, and all of those I/O banks use the same VRN/VRP resistor values, the internal VRN and VRP nodes can be cascaded so that only one pair of pins for all of the I/O banks in the entire I/O column is required to be connected to precision resistors. DCI_CASCADE identifies the master bank and all associated slave banks for this feature. For more information refer to the *7 Series FPGAs SelectIO Resources User Guide (UG471)* [Ref 2].

Architecture Support

- Kintex™-7 devices
- Virtex®-7 devices
- Larger Zynq™ devices

Applicable Objects

- I/O Bank (`get_iobanks`)
 - High Performance (HP) bank type

Values

Valid High Performance (HP) bank numbers. See

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property DCI_CASCADE {slave_banks} [get_iobanks master_bank]
```

Where

- **slave_banks** is a list of the bank numbers of the slave banks.
- **master_bank** is the bank number of the designated master bank.

XDC Syntax Example

```
# Designate Bank 14 as a master DCI Cascade bank and Banks 15 and 16 as its slaves  
set_property DCI_CASCADE {15 16} [get_iobanks 14]
```

Affected Steps

- I/O planning
- **place_design**
- DRC
- **write_bitstream**
- **report_power**

DIFF_TERM

The differential termination (DIFF_TERM) property supports the differential I/O standards for inputs and bidirectional ports. It is used to enable or disable the built-in, 100Ω, differential termination. Refer to the *7 Series FPGAs SelectIO Resources User Guide (UG471)* [Ref 2] from more information.

DIFF_TERM indicates a differential termination method should be used on differential input and bidirectional port buffers, and that the Vivado tool should add on-chip termination to the port.

Architecture Support

All architectures

Applicable Objects

- Ports (`get_ports`)
 - Input or bidirectional ports connected to a differential input buffer
- Cells (`get_cells`)
 - Differential input or bidirectional buffers (all IBUFDS and IOBUFDS variants)
- Applicable to elements using one of the following IOSTANDARDS:
 - LVDS
 - LVDS_25
 - MINI_LVDS_25
 - PPDS_25
 - RSDS_25

Values

- FALSE (default)

Differential termination is disabled.
- TRUE

Differential termination is enabled.

Syntax



RECOMMENDED: Use the instantiation template from the Language Templates or the Vivado Design Suite 7 Series FPGA Libraries Guide (UG953) [Ref 9] to specify the proper syntax.

Verilog Syntax

To set DIFF_TERM, assign the DIFF_TERM parameter on the instantiated differential buffer.

Verilog Syntax Example

The following example enables differential termination on the IBUFDS instance named `clk_ibufds`.

```
// IBUFDS: Differential Input Buffer
//      Virtex-7
// Xilinx HDL Language Template, version 2013.2
IBUFDS #(
    .DIFF_TERM("TRUE"),          // Differential Termination
    .IBUF_LOW_PWR("TRUE"),      // Low power="TRUE", Highest performance="FALSE" for
the specified IOSTANDARD
    .IOSTANDARD("DEFAULT")      // Specify the input I/O standard
) clk_ibufds (
    .O(clk),                    // Buffer output
    .I(CLK_p),                  // Diff_p buffer input (connect directly to top-level port)
    .IB(CLK_n)                  // Diff_n buffer input (connect directly to top-level port)
);
// End of clk_ibufds instantiation
```

VHDL Syntax

DIFF_TERM can be set by assigning the DIFF_TERM generic on the instantiated differential buffer.

VHDL Syntax Example

The following example enables differential termination on the IBUFDS instance named `clk_ibufds`.

```
-- IBUFDS: Differential Input Buffer
-- Xilinx HDL Language Template, version 2013.2
clk_ibufds : IBUFDS
generic map (
    DIFF_TERM => TRUE,          -- Differential Termination
    IBUF_LOW_PWR => TRUE,      -- Low power (TRUE) vs. performance (FALSE) setting
    IOSTANDARD => "DEFAULT")
port map (
    O => clk,                  -- Buffer output
    I => CLK_p,                -- Diff_p buffer input (connect directly to top-level port)
    IB => CLK_n                -- Diff_n buffer input (connect directly to top-level port)
);
-- End of clk_ibufds instantiation
```

XDC Syntax

```
set_property DIFF_TERM TRUE [get_ports port_name]
```

Where:

- **set_property DIFF_TERM** can be assigned to port objects.
- **port_name** is an input or bidirectional port connected to a differential buffer.

XDC Syntax Example

```
# Enables differential termination on port named CLK_p  
set_property DIFF_TERM TRUE [get_ports CLK_p]
```

Alternative XDC Syntax Example

This property can be applied to the buffer instance:

```
set_property DIFF_TERM TRUE [get_cells instance_name]
```

Where

- **instance_name** is an input or bidirectional differential buffer instance.

```
# Enables differential termination on buffer instance clk_ibufds  
set_property DIFF_TERM TRUE [get_ports clk_ibufds]
```

Affected Steps

- I/O Planning
- report_ssn
- report_power

See Also

- [IOSTANDARD](#), page 90

DONT_TOUCH

DONT_TOUCH directs the tool to not optimize a user hierarchy or instantiated component so that optimization does not occur across its boundary. While this can assist floorplanning, analysis, and debugging, it may inhibit optimization, resulting in a larger, slower design.



RECOMMENDED: Register all outputs of a module instance in which a DONT_TOUCH is attached. To be most effective, apply this attribute before synthesis.

Architecture Support

All architectures

Applicable Objects

- Cells (`get_cells`)
 - User defined instance

Values

- FALSE (default)

Allows optimization across the hierarchy.

- TRUE

Preserves the hierarchy by not allowing optimization across the hierarchy boundary.

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the user hierarchy instantiation:

```
(* DONT_TOUCH = "{TRUE|FALSE}" *)
```

Verilog Syntax Example

```
// Preserve the hierarchy of instance CLK1_rst_sync
(* DONT_TOUCH = "TRUE" *) reset_sync #(
    .STAGES(5)
) CLK1_rst_sync (
    .RST_IN(RST | ~LOCKED),
    .CLK(clk1_100mhz),
    .RST_OUT(rst_clk1)
);
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute DONT_TOUCH : string;
```

Specify the VHDL attribute as follows:

```
attribute DONT_TOUCH of name: label is "{TRUE|FALSE}";
```

Where

- **name** is the instance name of a user defined instance.

VHDL Syntax Example

```
attribute DONT_TOUCH : string;
-- Preserve the hierarchy of instance CLK1_rst_sync
attribute DONT_TOUCH of CLK1_rst_sync: label is "TRUE";
...
CLK1_rst_sync : reset_sync
  PORT MAP (
    RST_IN => RST_LOCKED,
    CLK => clk1_100mhz,
    RST_OUT => rst_clk1
  );
```

XDC Syntax

```
set_property DONT_TOUCH {TRUE|FALSE} [get_cells instance_name]
```

Where

- **instance_name** is a leaf cell or hierarchical cell.

XDC Syntax Example

```
# Preserve the hierarchy of instance CLK1_rst_sync
set_property DONT_TOUCH TRUE [get_cells CLK1_rst_sync]
```

Affected Steps

- synth_design
- opt_design
- phys_opt_design
- floorplanning

DRIVE

DRIVE specifies output buffer drive strength in mA for output buffers configured with I/O standards that support programmable output drive strengths.

Architecture Support

All architectures

Applicable Objects

- Ports (`get_ports`)
 - Output or bidirectional ports connected
- Cells (`get_cells`)
 - Output Buffers (all OBUF variants)

Values

Integer values:

- 2
- 4
- 6
- 8
- 12 (default)
- 16
- 24

Syntax

Verilog Syntax

To set this attribute when inferring I/O buffers, place the proper Verilog attribute syntax before the top-level output port declaration.

```
(* DRIVE = "{2|4|6|8|12|16|24}" *)
```

Verilog Syntax Example

```
// Sets the drive strength on the STATUS output port to 2 mA
(* DRIVE = "2" *) output STATUS,
```

Alternative Verilog Syntax Example

If the output or bidirectional buffer is instantiated, DRIVE can be set by assigning the DRIVE parameter on the instantiated output buffer.



RECOMMENDED: Use the instantiation template from the Language Templates or the Vivado Design Suite 7 Series FPGA Libraries Guide (UG953) [Ref 9] to specify the proper syntax.

The following example sets the drive strength on the OBUF instance named `status_obuf` to 2 mA:

```
// OBUF: Single-ended Output Buffer
//      Virtex-7
// Xilinx HDL Language Template, version 2013.2
OBUF #(
    .DRIVE(2), // Specify the output drive strength
    .IOSTANDARD("DEFAULT"), // Specify the output I/O standard
    .SLEW("SLOW") // Specify the output slew rate
) status_obuf (
    .O(STATUS), // Buffer output (connect directly to top-level port)
    .I(status_int) // Buffer input
);
// End of status_obuf instantiation
```

VHDL Syntax

To set this attribute when inferring I/O buffers, place the proper VHDL attribute syntax before the top-level output port declaration.

Declare the VHDL attribute as follows:

```
attribute DRIVE : integer;
```

Specify the VHDL attribute as follows:

```
attribute DRIVE of port_name : signal is value;
```

Where

- `port_name` is a top-level output port.

VHDL Syntax Example

```
STATUS : out std_logic;
attribute DRIVE : integer;
-- Sets the drive strength on the STATUS output port to 2 mA
attribute DRIVE of STATUS : signal is 2;
```

Alternative VHDL Syntax Example

If the output or bidirectional buffer is instantiated, DRIVE can be set by assigning the DRIVE generic on the instantiated output buffer.

The following example sets the drive strength on the OBUF instance named status_obuf to 2 mA.

```
-- OBUF: Single-ended Output Buffer
--       Virtex-7
-- Xilinx HDL Language Template, version 2013.2
status_obuf : OBUF
  generic map (
    DRIVE => 2,
    IOSTANDARD => "DEFAULT",
    SLEW => "SLOW")
  port map (
    O => STATUS,      -- Buffer output (connect directly to top-level port)
    I => status_int  -- Buffer input
  );
-- End of status_obuf instantiation
```

XDC Syntax

```
set_property DRIVE value [get_ports port_name]
```

Where

- `port_name` is an output or bidirectional port.

XDC Syntax Example

```
# Sets the drive strength of the port STATUS to 2 mA
set_property DRIVE 2 [get_ports STATUS]
```

Affected Steps

- I/O Planning
- Report Noise
- Report Power

See Also

Refer to the following design elements in the *Vivado Design Suite 7 Series FPGA Libraries Guide* (UG953) [Ref 9].

- OBUF
- OBUFT
- IOBUF

FSM_ENCODING

FSM_ENCODING controls how a state machine is encoded during synthesis.

As a default, the Vivado synthesis tool chooses an encoding protocol for state machines based on internal algorithms that determine the best solution for most designs. However, the FSM_ENCODING property lets you specify the state machine encoding of your choice.

Architecture Support

All architectures

Applicable Objects

- State machine registers

Values

- off - This disables state machine encoding within the Vivado synthesis tool. In this case the state machine is synthesized as logic.
- one_hot
- sequential
- johnson
- gray
- auto - This is the default behavior when FSM_ENCODING is not specified. It allows the Vivado synthesis tool to determine the best state machine encoding method. In this case, the tool may use different encoding styles for different FSMs in the same design.

Verilog Syntax

```
(* fsm_encoding = "one_hot" *) reg [7:0] my_state;
```

VHDL Syntax

```
type count_state is (zero, one, two, three, four, five, six, seven);  
signal my_state : count_state;  
attribute fsm_encoding : string;  
attribute fsm_encoding of my_state : signal is "sequential";
```

XDC Syntax

No applicable

Affected Steps

- Synthesis

See Also

- [FSM_SAFE_STATE](#), page 63

FSM_SAFE_STATE

The Vivado synthesis tool supports extraction of Finite State Machines in a variety of configurations as determined by the `FSM_ENCODING` property, or the `-fsm_extraction` command line option for Vivado synthesis. Refer to the *Vivado Design Suite User Guide: Synthesis (UG901)* [Ref 5] for more information.

However, a state machine can enter into an invalid, or “unreachable” state that causes the design to fail. If a Finite State Machine (FSM) enters an invalid state, the `FSM_SAFE_STATE` defines a recovery state for use when an FSM is synthesized in the Vivado synthesis tool.



TIP: *The unreachable state can also be managed in the HDL definition of the state machine, with a default case for instance. However, the `FSM_SAFE_STATE` property can be used as a fail safe for the design.*

Architecture Support

All architectures

Applicable Objects

- Design (`current_design`)

Values

- **reset_state** - Return the state machine to the RESET state, as determined by the Vivado synthesis tool.
- **power_on_state** - Return the state machine to the POWER_ON state, as determined by the Vivado synthesis tool.

Syntax

Verilog Syntax

```
(* fsm_safe_state = "reset_state" *) reg [2:0] state;
```

VHDL Syntax

```
attribute fsm_safe_state : string;  
attribute fsm_safe_state of state : signal is "power_on_state";
```

XDC Syntax

Not applicable

Affected Steps

- Synthesis

See Also

- [FSM_ENCODING](#), page 61

H_SET and HU_SET

Hierarchical sets are collections of logic elements based on the hierarchy of the design as defined by the HDL source files. H_SET, HU_SET, and U_SET are attributes within the HDL design source files, and do not appear in the synthesized or implemented design. They are used when defining Relatively Placed Macros, or RPMs in the RTL design. For more information on using these properties, and defining RPMs, refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 6].

H_SET is a property that is implied due to the presence of RLOC properties on logic cells in the hierarchy of a design. Logic elements inside of a hierarchical block, that have the RLOC property, are automatically assigned to the same Hierarchical Set, or H_SET.

Each hierarchical module is assigned an H_SET property based on the instance name of the module. Each hierarchical module may only have a single H_SET name, and all logic elements inside that hierarchy are elements of that H_SET.

Note: H_SET is only defined if there is no HU_SET or U_SET defined, but RLOC is defined.

You can also manually create a User-defined Hierarchical Set, or HU_SET, or a User-defined Set, or U_SET, that is not dependant on the hierarchy of the design.

You can define multiple HU_SET names for a single hierarchical module, and assign specific instances of that hierarchy to the HU_SET. This allows you to divide the logic elements of a single hierarchical module into multiple HU_SETs.



IMPORTANT: *When using H_SET or HU_SET, the KEEP_HIERARCHY property is also required for Vivado Synthesis to preserve the hierarchy for the RPM in the synthesized design.*

When RLOC is also present in the RTL source files, the H_SET, HU_SET, and U_SET properties get translated to a read-only RPM property on cells in the synthesized netlist. The HU_SET and U_SET are visible on the RTL source file in the Text editor in the Vivado Design Suite. However, in the Properties window of a cell object, the RPM property is displayed.

Architecture Support

All architectures.

Applicable Objects

The HU Set constraint may be used in one or more of the following design elements, or categories of design elements. Refer to the *Vivado Design Suite 7 Series FPGA Libraries Guide* (UG953) [Ref 9] for more information on the specific design elements:

- Registers
- LUT
- Macro Instance
- RAMS
- RAMD
- RAMB18/FIFO18
- RAMB36/FIFO36
- DSP48

Values

- NAME: A unique name for the HU_SET.

Syntax

Verilog Syntax

This is a Verilog attribute used in combination with the RLOC property to define the set content of a hierarchical block that will define an RPM in the synthesized netlist. Place the Verilog attribute immediately before the instantiation of a logic element.

```
(* RLOC = "X0Y0", HU_SET = "h0" *) FD sr0 (.C(clk), .D(sr_1n), .Q(sr_0));
```

Verilog Example

The following Verilog module defines RLOC and HU_SET properties for the shift register Flops in the module.

```
module ffs (  
    input  clk,  
    input  d,  
    output q  
);  
  
    wire  sr_0, sr_0n;  
    wire  sr_1, sr_1n;  
    wire  sr_2, sr_2n;  
    wire  sr_3, sr_3n;  
    wire  sr_4, sr_4n;  
    wire  sr_5, sr_5n;
```

```

wire    sr_6, sr_6n;
wire    sr_7, sr_7n;

wire    inr, inrn, outr;

inv i0 (sr_0, sr_0n);
inv i1 (sr_1, sr_1n);
inv i2 (sr_2, sr_2n);
inv i3 (sr_3, sr_3n);
inv i4 (sr_4, sr_4n);
inv i5 (sr_5, sr_5n);
inv i6 (sr_6, sr_6n);
inv i7 (sr_7, sr_7n);
inv i8 (inr, inrn);

(* RLOC = "X0Y0", HU_SET = "h0" *) FD sr0 (.C(clk), .D(sr_1n), .Q(sr_0));
(* RLOC = "X0Y0", HU_SET = "h0" *) FD sr1 (.C(clk), .D(sr_2n), .Q(sr_1));
(* RLOC = "X0Y1", HU_SET = "h0" *) FD sr2 (.C(clk), .D(sr_3n), .Q(sr_2));
(* RLOC = "X0Y1", HU_SET = "h0" *) FD sr3 (.C(clk), .D(sr_4n), .Q(sr_3));
(* RLOC = "X0Y0", HU_SET = "h1" *) FD sr4 (.C(clk), .D(sr_5n), .Q(sr_4));
(* RLOC = "X0Y0", HU_SET = "h1" *) FD sr5 (.C(clk), .D(sr_6n), .Q(sr_5));
(* RLOC = "X0Y1", HU_SET = "h1" *) FD sr6 (.C(clk), .D(sr_7n), .Q(sr_6));
(* RLOC = "X0Y1", HU_SET = "h1" *) FD sr7 (.C(clk), .D(inrn), .Q(sr_7));
(* LOC = "SLICE_X0Y0" *) FD inq (.C(clk), .D(d), .Q(inr));
FD outq (.C(clk), .D(sr_0n), .Q(outr));

assign q = outr;

endmodule // ffs

```

In the preceding example, you will need to specify the KEEP_HIERARCHY property to instances of the ffs module to preserve the hierarchy and define the RPM in the synthesized design:

```

module top (
    input  clk,
    input  d,
    output q
);

wire    c1, c2;

(* KEEP_HIERARCHY = "YES" *) ffs u0 (clk, d, c1);
(* KEEP_HIERARCHY = "YES" *) ffs u1 (clk, c1, c2);
(* KEEP_HIERARCHY = "YES" *) ffs u2 (clk, c2, q);

endmodule // top

```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute HU_SET : string;
```

Specify the VHDL constraint as follows:

```
attribute HU_SET of {component_name | entity_name | label_name} :  
{component|entity|label} is "NAME";
```

Where:

- {component_name | entity_name | label_name} is a choice of one design element.
- {component|entity|label} is the instance ID of the design element.
- "NAME" is the unique set name to give to the HU_SET.

XDC Syntax

The HU_SET property can not be defined using XDC constraints. The HU_SET property, when present on logic elements with the RLOC property, defines relatively placed macros (RPMs), and results in the read-only RPM property in the netlist of synthesized designs.



TIP: You can use the `create_macro` and `update_macro` commands to define macro objects in the Vivado Design Suite, that act like RPMs within the design. Refer to the Vivado Design Suite Tcl Command Reference (UG835) [Ref 4] for more information on these commands.

Affected Steps

- Design Floorplanning
- place_design
- synth_design

See Also

[KEEP_HIERARCHY](#), page 93

[RLOC](#), page 129

[RLOCS](#), page 133

[RLOC_ORIGIN](#), page 135

[RPM](#), page 140

[RPM_GRID](#), page 141

[U_SET](#), page 146

HIODELAY_GROUP

HIODELAY_GROUP groups IDELAYCTRL components to their associated IDELAY or ODELAY instances for proper placement and replication.

If you use HIODELAY_GROUP to assign a group name to an IDELAYCTRL, you need to also associate an IDELAY or ODELAY cell to the group using the same HIODELAY_GROUP property.



IMPORTANT: *While an HIODELAY_GROUP can contain multiple cells, a cell can only be assigned to one HIODELAY_GROUP.*

The following example uses `set_property` to group all the IDELAY/ODELAY elements associated with a specific IDELAYCTRL.

```
set_property HIODELAY_GROUP IO_DLY1 [get_cells MY_IDELAYCTRL_inst]
set_property HIODELAY_GROUP IO_DLY1 [get_cells MY_IDELAY_inst]
set_property HIODELAY_GROUP IO_DLY1 [get_cells MY_ODELAY_inst]
```

Difference Between HIODELAY_GROUP and IODELAY_GROUP

HIODELAY_GROUP is uniquified per hierarchy. Use HIODELAY_GROUP when:

- You expect to have multiple instances of a module that contains an IDELAYCTRL.
and
- You do not intend to group that instance with any IDELAY or ODELAY instances in other logical hierarchies.

Architecture Support

All architectures

Applicable Objects

- Cells (`get_cells`)
 - IDELAY, ODELAY, or IDELAYCTRL instances

Values

Any specified group name

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the instantiation of an IDELAY, ODELAY, or IDELAYCTRL.

```
(* HIODELAY_GROUP = "value" *)
```

Verilog Syntax Example

```
// Specifies a group name of DDR_INTERFACE to an instantiated IDELAYCTRL
// IDELAYCTRL: IDELAYE2/ODELAYE2 Tap Delay Value Control
//                               Virtex-7
// Xilinx HDL Language Template, version 2013.2
// Specifies DDR_INTERFACE group name for IDELAYs/ODELAYs and IDELAYCTRL
(* HIODELAY_GROUP = "DDR_INTERFACE" *)
IDELAYCTRL DDR_IDELAYCTRL_inst (
    .RDY(),           // 1-bit output: Ready output
    .REFCLK(REFCLK), // 1-bit input: Reference clock input
    .RST(1'b0)       // 1-bit input: Active high reset input
);
// End of DDR_IDELAYCTRL_inst instantiation
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute HIODELAY_GROUP : string;
```

For an instantiated instance, specify the VHDL attribute as follows:

```
attribute HIODELAY_GROUP of instance_name : label is "group_name";
```

Where

- **instance_name** is the instance name of an instantiated IDELAY, ODELAY, or IDELAYCTRL.

VHDL Syntax Example

```
// Specifies a group name of DDR_INTERFACE to an instantiated IDELAYCTRL
attribute HIODELAY_GROUP : STRING;
attribute HIODELAY_GROUP of DDR_IDELAYCTRL_inst: label is "DDR_INTERFACE";
begin
    -- IDELAYCTRL: IDELAYE2/ODELAYE2 Tap Delay Value Control
    --                               Virtex-7
    -- Xilinx HDL Language Template, version 2013.2
    DDR_IDELAYCTRL_inst : IDELAYCTRL
    port map (
        RDY => open,           -- 1-bit output: Ready output
        REFCLK => REFCLK,     -- 1-bit input: Reference clock input
        RST => '0'            -- 1-bit input: Active high reset input
    );
    -- End of DDR_IDELAYCTRL_inst instantiation
```

XDC Syntax

```
set_property HIODELAY_GROUP group_name [get_cells instance_name]
```

Where

- **instance_name** is the instance name of an IDELAY, ODELAY, or IDELAYCTRL.

XDC Syntax Example

```
# Specifies a group name of DDR_INTERFACE to an instantiated IDELAYCTRL
set_property HIODELAY_GROUP DDR_INTERFACE [get_cells DDR_IDELAYCTRL_inst]
```

Affected Steps

`place_design`

See Also

[IODELAY_GROUP](#), page 87

Refer to the following design elements in the *Vivado Design Suite 7 Series FPGA Libraries Guide* (UG953) [Ref 9].

- IDELAYCTRL
- IDELAYE2
- ODELAYE2

HLUTNM

HLUTNM instructs the tool to place two LUT5, SRL16, or LUTRAM components with compatible inputs into the same LUT6 site. Specify the HLUTNM in pairs per hierarchy, with two of these specified on compatible instance types with the same group name.

Difference Between HLUTNM and LUTNM

HLUTNM is uniquified per hierarchy.

- Use HLUTNM when you expect to have multiple instances of a module that contains LUT components to be grouped together.
- Use LUTNM to group two LUT components that exist in different hierarchies.

Architecture Support

All architectures

Applicable Objects

- Cells (`get_cells`)
 - LUT (LUT1, LUT2, LUT3, LUT4, LUT5)
 - SRL (SRL16E)
 - LUTRAM (RAM32X1S)

Values

A unique group name

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the instantiation of a LUT.

The Verilog attribute must be used in pairs in the same logical hierarchy.

```
(* HLUTNM = "group_name" *)
```

Verilog Syntax Example

```
// Designates state0_inst to be placed in same LUT6 as state1_inst
// LUT5: 5-input Look-Up Table with general output (Mapped to a LUT6)
//      Virtex-7
// Xilinx HDL Language Template, version 2013.2
(* HLUTNM = "LUT_group1" *) LUT5 #(
    .INIT(32'ha2a2aea2) // Specify LUT Contents
) state0_inst (
    .O(state_out[0]), // LUT general output
    .I0(state_in[0]), // LUT input
    .I1(state_in[1]), // LUT input
    .I2(state_in[2]), // LUT input
    .I3(state_in[3]), // LUT input
    .I4(state_in[4]) // LUT input
);
// End of state0_inst instantiation
// LUT5: 5-input Look-Up Table with general output (Mapped to a LUT6)
//      Virtex-7
// Xilinx HDL Language Template, version 2013.2
(* HLUTNM = "LUT_group1" *) LUT5 #(
    .INIT(32'h00330073) // Specify LUT Contents
) state1_inst (
    .O(state_out[1]), // LUT general output
    .I0(state_in[0]), // LUT input
    .I1(state_in[1]), // LUT input
    .I2(state_in[2]), // LUT input
    .I3(state_in[3]), // LUT input
    .I4(state_in[4]) // LUT input
);
// End of state1_inst instantiation
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute HLUTNM : string;
```

For an instantiated instance, specify the VHDL attribute as follows:

```
attribute HLUTNM of instance_name : label is "group_name";
```

Where

- **instance_name** is a LUT1, LUT2, LUT3, LUT4, LUT5, SRL16, or LUTRAM instance.

The VHDL attribute must be used in pairs in the same logical hierarchy.

VHDL Syntax Example

```
-- Designates state0_inst to be placed in same LUT6 as state1_inst
attribute HLUTNM : string;
attribute HLUTNM of state0_inst : label is "LUT_group1";
attribute HLUTNM of state1_inst : label is "LUT_group1";
begin
  -- LUT5: 5-input Look-Up Table with general output (Mapped to SliceM LUT6)
  --      Virtex-7
  -- Xilinx HDL Language Template, version 2013.2
  state0_inst : LUT5
  generic map (
    INIT => X"a2a2aea2") -- Specify LUT Contents
  port map (
    O => state_out(0), -- LUT general output
    I0 => state_in(0), -- LUT input
    I1 => state_in(1), -- LUT input
    I2 => state_in(2), -- LUT input
    I3 => state_in(3), -- LUT input
    I4 => state_in(4)  -- LUT input
  );
  -- End of state0_inst instantiation
  -- LUT5: 5-input Look-Up Table with general output (Mapped to SliceM LUT6)
  --      Virtex-7
  -- Xilinx HDL Language Template, version 2013.2
  state1_inst : LUT5
  generic map (
    INIT => X"00330073") -- Specify LUT Contents
  port map (
    O => state_out(1), -- LUT general output
    I0 => state_in(0), -- LUT input
    I1 => state_in(1), -- LUT input
    I2 => state_in(2), -- LUT input
    I3 => state_in(3), -- LUT input
    I4 => state_in(4)  -- LUT input
  );
  -- End of state1_inst instantiation
```

XDC Syntax

```
set_property HLUTNM group_name [get_cells instance_name]
```

Where

- **instance_name** is a LUT1, LUT2, LUT3, LUT4, LUT5, SRL16, or LUTRAM instance.

XDC Syntax Example

```
# Designates state0_inst LUT5 to be placed in same LUT6 as state1_inst
set_property HLUTNM LUT_group1 [get_cells state0_inst]
set_property HLUTNM LUT_group1 [get_cells state1_inst]
```

Affected Steps

- `place_design`

See Also

[LUTNM, page 103](#)

IBUF_LOW_PWR

The IBUF_LOW_PWR property allows an optional trade-off between performance and power.

The IBUF_LOW_PWR property is applied to the I/O buffer instance. This property is set to TRUE by default, which implements the input buffer in the lower-power mode rather than the higher-performance mode (FALSE).

The change in power can be estimated using the XPower Estimator (XPE) or the `report_power` command in the Vivado Design Suite.

Architecture Support

All architectures

Applicable Objects

- Input ports (`get_ports`) or Input buffers (`get_cells`) with a VREF-based I/O Standard such as SSTL or HSTL or a differential standard such as LVDS or DIFF_HSTL.

Values

- TRUE: Implements the input or bidirectional buffer in low power mode.
- FALSE: Implements the input or bidirectional buffer in high performance mode.

Syntax

Verilog Syntax

To set IBUF_LOW_PWR, assign the parameter on the buffer module definition, or the instantiated buffers.

VHDL Syntax

IBUF_LOW_PWR can be set by assigning the generic on the entity definition, or on the instance.

XDC Syntax

IBUF_LOW_PWR can be assigned as a property on port objects with a DIRECTION of IN or INOUT.

```
set_property IBUF_LOW_PWR TRUE [get_ports port_name]
```

Where:

- `set_property IBUF_LOW_PWR` can be assigned to port objects.
- `port_name` is an input or bidirectional port.

Affected Steps

- `report_power`
- `report_timing`

See Also

[IOSTANDARD](#), page 90

IN_TERM

IN_TERM specifies an un-calibrated input termination impedance value. IN_TERM is supported on High Range (HR) bank inputs only. For inputs in High Performance (HP) banks, specify a digitally controlled impedance (DCI) [IOSTANDARD](#) for on-chip termination.

The termination is present constantly on inputs, and on bidirectional pins whenever the output buffer is 3-stated. However, an important difference between this un-calibrated split-termination option and the 3-state split-termination DCI is that instead of calibrating to external reference resistors on the VRN and VRP pins when using DCI, this feature invokes internal resistors that have no calibration routine to compensate for temperature, process, or voltage variations. This option has target Thevenin equivalent resistance values of 40Ω, 50Ω, and 60Ω. For more information refer to the *7 Series FPGAs SelectIO Resources User Guide (UG471)* [\[Ref 2\]](#).

Architecture Support

All architectures on High Range (HR) bank inputs only.

Applicable Objects

- Ports (`get_ports`)
 - Input or bidirectional ports connected.
- Cells (`get_cells`)
 - Input Buffers (all IBUF variants).

Values

- NONE (default)
- UNTUNED_SPLIT_40
- UNTUNED_SPLIT_50
- UNTUNED_SPLIT_60

Syntax

Verilog Syntax

To set this attribute, place the proper Verilog attribute syntax before the top-level output port declaration.

```
(* IN_TERM = "{NONE|UNTUNED_SPLIT_40|UNTUNED_SPLIT_50|UNTUNED_SPLIT_60}" *)
```

Verilog Syntax Example

```
// Sets an on-chip input impedance of 50 Ohms to input ACT5  
(* IN_TERM = "UNTUNED_SPLIT_50" *) input ACT5,
```

VHDL Syntax

To set this attribute, place the proper VHDL attribute syntax before the top-level output port declaration.

Declare the VHDL attribute as follows:

```
attribute IN_TERM : string;
```

Specify the VHDL attribute as follows:

```
attribute IN_TERM of port_name : signal is value;
```

Where

- **port_name** is a top-level output port.

VHDL Syntax Example

```
ACT5 : in std_logic;  
attribute IN_TERM : string;  
-- Sets an on-chip input impedance of 50 Ohms to input ACT5  
attribute IN_TERM of ACT5 : signal is "UNTUNED_SPLIT_50";
```

XDC Syntax

```
set_property IN_TERM value [get_ports port_name]
```

Where:

- **IN_TERM** can be assigned to port objects, and nets connected to port objects.
- **port_name** is an output or bidirectional port.

XDC Syntax Example

```
# Sets an on-chip input impedance of 50 Ohms to input ACT5  
set_property IN_TERM UNTUNED_SPLIT_50 [get_ports ACT5]
```

Affected Steps

- I/O Planning
- Report Noise
- Report Power

See Also

[DCI_CASCADE](#), page 51

[DIFF_TERM](#), page 53

INTERNAL_VREF

INTERNAL_VREF specifies the use of an internal regulator on a bank to supply the voltage reference for standards requiring a reference voltage.

Architecture Support

All architectures

Applicable Objects

- I/O Bank (`get_iobanks`)

Values

- 0.60
- 0.675
- 0.75
- 0.90

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property INTERNAL_VREF {value} [get_iobanks bank]
```

Where

- `value` is the reference voltage value.

XDC Syntax Example

```
# Designate Bank 14 to have a reference voltage of 0.75 Volts  
set_property INTERNAL_VREF 0.75 [get_iobanks 14]
```

Affected Steps

- I/O planning
- `place_design`
- DRC
- `report_power`

IOB

IOB directs the tool to place a register in the input or output logic (I/O Block) to improve I/O timing.

Architecture Support

All architectures

Applicable Objects

- Ports (`get_ports`)
 - Any port connected to a register
- Cells (`get_cells`)
 - Registers connected directly to a top-level port

Values

- FALSE (default)
- TRUE

Syntax

Verilog Syntax

To set this attribute, place the proper Verilog attribute syntax before the top-level output port declaration.

```
(* IOB = "{TRUE|FALSE}" *)
```

Verilog Syntax Example

```
// Place the register connected to ACK in the input logic site  
(* IOB = "TRUE" *) input ACK,
```

Alternative Verilog Syntax Example

The IOB attribute can be placed on an instantiated or inferred register connected to a top-level port.

```
Place the register connected to ACK in the input logic site.  
input ACK;  
(* IOB = "TRUE" *) reg ack_reg = 1'b0;  
always @(posedge CLK)  
    ack_reg = 1'b0;
```

VHDL Syntax

To set this attribute, place the proper VHDL attribute syntax before the top-level output port declaration.

Declare the VHDL attribute as follows:

```
attribute IOB : string;
```

Specify the VHDL attribute as follows:

```
attribute IOB of <port_name>: signal is "{TRUE|FALSE}";
```

Where

- `port_name` is a top-level output port.

VHDL Syntax Example

```
ACK : in std_logic;  
attribute IOB : string;  
-- Place the register connected to ACK in the input logic site  
attribute IOB of ACK: signal is "TRUE";
```

Alternative VHDL Syntax Example

The IOB attribute can be placed on an instantiated or inferred register connected to a top-level port. Place the register connected to ACK in the input logic site.

XDC Syntax

```
set_property IOB value [get_ports port_name]
```

Where

- `value` is TRUE or FALSE.

XDC Syntax Example

```
# Place the register connected to ACK in the input logic site  
set_property IOB TRUE [get_ports ACK]
```

Affected Steps

- `place_design`

IOBDELAY

The Input Output Block Delay (IOBDELAY) property specifies whether to add or remove delay in the ILOGIC block in order to help mitigate input hold times for system-synchronous data input capture.

The ILOGIC block is located next to the I/O block (IOB), and contains the synchronous elements for capturing data as it comes into the FPGA through the IOB. The ILOGIC block in 7 series devices can be configured as ILOGICE2 in HP I/O banks, and as ILOGICE3 in HR I/O banks. ILOGICE2 and ILOGICE3 are functionally identical except that ILOGICE3 has a zero hold delay element (ZHOLD) which can be configured with IOBDELAY. Refer to the *7 Series FPGAs SelectIO Resources User Guide* (UG471) [Ref 2] for more information on the use of IOBDELAY.

Architecture Support

All architectures

Applicable Objects

- Input Buffers (**get_cells**)
- Nets (**get_nets**)

Values

- NONE: Sets the delay to OFF for both the IBUF and input flip-flop (IFD) paths.
- IBUF
 - Sets the delay to OFF for any register inside the I/O component.
 - Sets the delay to ON for the buffered path through the ILOGIC block.
- IFD
 - Sets the delay to ON for the IFF register inside the I/O component.
 - Sets the delay to OFF for the BUFFERED path through the ILOGIC.
- BOTH: Sets the delay to ON for both the IBUF and IFD paths.

Syntax

Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* IOBDELAY = {NONE|BOTH|IBUF|IFD} *)
```

VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute iobdelay: string;
```

Specify the VHDL constraint as follows:

```
attribute iobdelay of {component_name |label_name }: {component|label} is  
"{NONE|BOTH|IBUF|IFD}";
```

XDC Syntax

```
set_property IOBDELAY value [get_cells cell_name]
```

Where:

- **value** is one of NONE, IBUF, IFD, BOTH

XDC Syntax Example

```
set_property IOBDELAY "BOTH" [get_nets {data0_I}]
```

Affected Steps

- Timing
- Placement
- Routing

IODELAY_GROUP

IODELAY_GROUP groups IDELAYCTRL cells together with their associated IDELAY and ODELAY cells to allow proper placement and replication.

If you use IODELAY_GROUP to assign a group name to an IDELAYCTRL, you need to also associate an IDELAY or ODELAY cell to the group using the same IODELAY_GROUP property.



IMPORTANT: *While an IODELAY_GROUP can contain multiple cells, a cell can only be assigned to one IODELAY_GROUP.*

The following example uses `set_property` to group all the IDELAY/ODELAY elements associated with a specific IDELAYCTRL.

```
set_property IODELAY_GROUP IO_DLY1 [get_cells MY_IDELAYCTRL_inst]
set_property IODELAY_GROUP IO_DLY1 [get_cells MY_IDELAY_inst]
set_property IODELAY_GROUP IO_DLY1 [get_cells MY_ODELAY_inst]
```

Difference Between IODELAY_GROUP and HIODELAY_GROUP

IODELAY_GROUP can group elements across different hierarchies. Use IODELAY_GROUP to group I/O delay components in different hierarchies together.

HIODELAY_GROUP groups I/O delay components under the same hierarchical module.

Architecture Support

All architectures

Applicable Objects

- Cells (`get_cells`)
 - IDELAY, ODELAY, or IDELAYCTRL instances

Values

Any specified group name

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the instantiation of an IDELAY, ODELAY, or IDELAYCTRL.

```
(* IODELAY_GROUP = "value" *)
```

Verilog Syntax Example

```
// Specifies a group name of DDR_INTERFACE to an instantiated IDELAYCTRL
// IDELAYCTRL: IDELAYE2/ODELAYE2 Tap Delay Value Control
// Virtex-7
// Xilinx HDL Language Template, version 2013.2
// Specifies DDR_INTERFACE group name for IDELAYs/ODELAYs and IDELAYCTRL
(* IODELAY_GROUP = "DDR_INTERFACE" *)
IDELAYCTRL DDR_IDELAYCTRL_inst (
    .RDY(),          // 1-bit output: Ready output
    .REFCLK(REFCLK), // 1-bit input: Reference clock input
    .RST(1'b0)       // 1-bit input: Active high reset input
);
// End of DDR_IDELAYCTRL_inst instantiation
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute IODELAY_GROUP : string;
```

For an instantiated instance, specify the VHDL attribute as follows:

```
attribute IODELAY_GROUP of instance_name : label is "group_name";
```

Where

- **instance_name** is the instance name of an instantiated IDELAY, ODELAY, or IDELAYCTRL.

VHDL Syntax Example

```
// Specifies a group name of DDR_INTERFACE to an instantiated IDELAYCTRL
attribute IODELAY_GROUP : STRING;
attribute IODELAY_GROUP of DDR_IDELAYCTRL_inst: label is "DDR_INTERFACE";
begin
    -- IDELAYCTRL: IDELAYE2/ODELAYE2 Tap Delay Value Control
    -- Virtex-7
    -- Xilinx HDL Language Template, version 2013.2
    DDR_IDELAYCTRL_inst : IDELAYCTRL
    port map (
        RDY => open,          -- 1-bit output: Ready output
        REFCLK => REFCLK,    -- 1-bit input: Reference clock input
        RST => '0'           -- 1-bit input: Active high reset input
    );
    -- End of DDR_IDELAYCTRL_inst instantiation
```

XDC Syntax

```
set_property IODELAY_GROUP group_name [get_cells instance_name]
```

Where

- **group_name** is a user-specified name for the IODELAY_GROUP.
- **instance_name** is the instance name of an IDELAY, ODELAY, or IDELAYCTRL.

XDC Syntax Example

```
# Specifies a group name of DDR_INTERFACE to an instantiated IDELAYCTRL  
set_property IODELAY_GROUP DDR_INTERFACE [get_cells DDR_IDELAYCTRL_inst]
```

Affected Steps

- Placement

See Also

[HIODELAY_GROUP, page 69](#)

Refer to the following design elements in the *Vivado Design Suite 7 Series FPGA Libraries Guide* (UG953) [\[Ref 9\]](#).

- IDELAYCTRL
- IDELAYE2
- ODELAYE2

IOSTANDARD

IOSTANDARD specifies which programmable I/O Standard to use to configure input, output, or bidirectional ports. You must specify an IOSTANDARD on all ports in order to create a bitstream.

Architecture Support

All architectures

Applicable Objects

- Ports (`get_ports`)
 - Any port - Define the IOSTANDARD in the RTL source of I/O Ports, or as XDC constraints for port cells.
- Cells (`get_cells`)
 - I/O Buffers (all IBUF, OBUF, and IOBUF variants) - For instantiated buffers, IOSTANDARD must be defined in the RTL as an attribute of the cell. XDC Syntax is not supported.

Values

A valid I/O Standard for the target Xilinx FPGA. Refer to the *7 Series FPGAs SelectIO Resources User Guide (UG471)* [Ref 2].

Syntax

Verilog Syntax

To set this attribute when inferring I/O buffers, place the proper Verilog attribute syntax before the top-level output port declaration.

```
(* IOSTANDARD = "value" *)
```

Verilog Syntax Example

```
// Sets the I/O Standard on the STATUS output to LVCMOS12  
(* IOSTANDARD = "LVCMOS12" *) output STATUS,
```

Alternative Verilog Syntax Example

If the I/O buffer is instantiated, IOSTANDARD can be set by assigning the IOSTANDARD parameter on the instantiated I/O buffer.



RECOMMENDED: Use the instantiation template from the Language Templates or the Vivado Design Suite 7 Series FPGA Libraries Guide (UG953) [Ref 9] to specify the proper syntax.

The following example sets the I/O Standard on the STATUS output to LVCMOS12.

```
// OBUF: Single-ended Output Buffer
//      Virtex-7
// Xilinx HDL Language Template, version 2013.2
OBUF #(
    .DRIVE(12),      // Specify the output drive strength
    .IOSTANDARD("LVCMOS12"), // Specify the output I/O standard
    .SLEW("SLOW") // Specify the output slew rate
) status_obuf (
    .O(STATUS),     // Buffer output (connect directly to top-level port)
    .I(status_int) // Buffer input
);
// End of status_obuf instantiation
```

VHDL Syntax

To set this attribute when inferring I/O buffers, place the proper VHDL attribute syntax before the top-level output port declaration.

Declare the VHDL attribute as follows:

```
attribute IOSTANDARD : string;
```

Specify the VHDL attribute as follows:

```
attribute IOSTANDARD of <port_name>: signal is "<standard>";
```

Where

- **port_name** is a top-level output port.

VHDL Syntax Example

```
STATUS : out std_logic;
attribute IOSTANDARD : string;
-- Sets the I/O Standard on the STATUS output to LVCMOS12
attribute IOSTANDARD of STATUS: signal is "LVCMOS12";
```

Alternative VHDL Syntax Example

To set IOSTANDARD when the I/O buffer is instantiated, assign the IOSTANDARD generic on the instantiated I/O buffer. The following example sets the I/O Standard on the STATUS output to LVCMOS12.

```
-- OBUF: Single-ended Output Buffer
-- Xilinx HDL Language Template, version 2013.2
status_obuf : OBUF
  generic map (
    DRIVE => 12,
    IOSTANDARD => "LVCMOS12",
    SLEW => "SLOW")
  port map (
    O => STATUS,      -- Buffer output (connect directly to top-level port)
    I => status_int  -- Buffer input
  );
-- End of status_obuf instantiation
```

XDC Syntax

```
set_property IOSTANDARD value [get_ports port_name]
```

Where

- **port_name** is a top-level port.

XDC Syntax Example

```
# Sets the I/O Standard on the STATUS output to LVCMOS12
set_property IOSTANDARD LVCMOS12 [get_ports STATUS]
```

Affected Steps

- I/O Planning
- Report Noise
- Report Power
- Report DRC
- **place_design**

See Also

Refer to the following design elements in the *Vivado Design Suite 7 Series FPGA Libraries Guide* (UG953) [Ref 9].

- OBUF
- OBUFT
- IOBUF

KEEP_HIERARCHY

KEEP_HIERARCHY directs the tool to retain a user hierarchy so that optimization does not occur across its boundary. While this can assist floorplanning, analysis, and debugging, it may inhibit optimization, resulting in a larger, slower design.



RECOMMENDED: *To avoid these negative effects, register all outputs of a module instance in which a KEEP_HIERARCHY is attached. To be most effective, apply this attribute before synthesis.*

Architecture Support

All

Applicable Objects

- Cells (`get_cells`)
 - User defined instance

Values

- FALSE (default)

Allows optimization across the hierarchy.

- TRUE

Preserves the hierarchy by not allowing optimization across the hierarchy boundary.

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the user hierarchy instantiation:

```
(* KEEP_HIERARCHY = "{TRUE|FALSE}" *)
```

Verilog Syntax Example

```
// Preserve the hierarchy of instance CLK1_rst_sync
(* KEEP_HIERARCHY = "TRUE" *) reset_sync #(
    .STAGES(5)
) CLK1_rst_sync (
    .RST_IN(RST | ~LOCKED),
    .CLK(clk1_100mhz),
    .RST_OUT(rst_clk1)
);
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute KEEP_HIERARCHY : string;
```

Specify the VHDL attribute as follows:

```
attribute KEEP_HIERARCHY of name: label is "{TRUE|FALSE}";
```

Where

- **name** is the instance name of a user defined instance.

VHDL Syntax Example

```
attribute KEEP_HIERARCHY : string;
-- Preserve the hierarchy of instance CLK1_rst_sync
attribute KEEP_HIERARCHY of CLK1_rst_sync: label is "TRUE";
...
CLK1_rst_sync : reset_sync
  PORT MAP (
    RST_IN => RST_LOCKED,
    CLK => clk1_100mhz,
    RST_OUT => rst_clk1
  );
```

XDC Syntax

```
set_property KEEP_HIERARCHY {TRUE|FALSE} [get_cells instance_name]
```

Where

- **instance_name** is a register instance.

XDC Syntax Example

```
# Preserve the hierarchy of instance CLK1_rst_sync
set_property KEEP_HIERARCHY TRUE [get_cells CLK1_rst_sync]
```

Affected Steps

- Design Floorplanning
- opt_design
- phys_opt_design
- synth_design

KEEPER

KEEPER applies a weak driver on a tri-stateable output or bidirectional port to preserve its value when not being driven. The KEEPER property retains the value of the output net to which it is attached.

For example, if logic 1 is being driven onto a net, KEEPER drives a weak or resistive 1 onto the net. If the net driver is then tri-stated, KEEPER continues to drive a weak or resistive 1 onto the net to preserve that value.

Input buffers (e.g., IBUF), 3-state output buffers (e.g., OBUFT), and bidirectional buffers (e.g., IOBUF) can have a weak pull-up resistor, a weak pull-down resistor, or a weak “keeper” circuit. This feature can be invoked by adding the one of the following properties to the net object connected to the buffer:

- PULLUP
- PULLDOWN
- KEEPER

Architecture Support

All

Applicable Objects

- Nets connected to I/O Buffers (`get_nets`)

Values

- TRUE | YES: Use a keeper circuit to preserve the value on the net.
- FALSE | NO: Do not use a keeper circuit. Default.

Syntax

Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* KEEPER = " {YES|NO|TRUE|FALSE}" *)
```

VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute keeper: string;
```

Specify the VHDL constraint as follows:

```
attribute keeper of signal_name : signal is "{YES|NO|TRUE|FALSE}";
```

XDC Syntax

```
set_property KEEPER {TRUE|FALSE} [get_nets net_name]
```

Where

- **net_name** is the name of a net connected to an IBUF, OBUFT, or IOBUF cell.

XDC Syntax Example

```
# Use a keeper circuit to preserve the value on the specified net
set_property KEEPER true [get_nets n1]
```

Affected Steps

- Logical to Physical Mapping

See Also

[PULLDOWN](#), page 123

[PULLUP](#), page 125

LOC

LOC specifies the placement assignment of a logic cell to the device resources of the target Xilinx FPGA.



RECOMMENDED: *To assign I/O ports to physical pins on the device package, use the `PACKAGE_PINS` property rather than `LOC`.*

Architecture Support

All architectures

Applicable Objects

- Cells (`get_cells`)
 - Any primitive cell

Values

Site name (for example, `SLICE_X15Y14` or `RAMB18_X6Y9`)

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the instantiation of a component.

The Verilog attribute can also be placed before the `reg` declaration of an inferred register, SRL, or LUTRAM when that `reg` can be placed into a single device site:

```
(* LOC = "site_name" *)
// Designates placed_reg to be placed in Slice site SLICE_X0Y0
(* LOC = "SLICE_X0Y0" *) reg placed_reg;
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute LOC : string;
```

For an instantiated instance, specify the VHDL attribute as follows:

```
attribute LOC of instance_name : label is "site_name";
```

Where

- **instance_name** is the instance name of an instantiated primitive.

VHDL Syntax Example

```
-- Designates instantiated register instance placed_reg to be placed
-- in Slice site SLICE_X0Y0
attribute LOC of placed_reg : label is "SLICE_X0Y0";
```

For an inferred instance, specify the VHDL attribute as follows:

```
attribute LOC of signal_name : signal is "site_name";
```

Where

- **signal_name** is the signal name of an inferred primitive that can be placed into a single site.

VHDL Syntax Example

```
-- Designates inferred register placed_reg to be placed in Slice site SLICE_X0Y0
attribute LOC of placed_reg : signal is "SLICE_X0Y0";
```

XDC Syntax

```
set_property LOC site_name [get_cells instance_name]
```

Where

- **instance_name** is a primitive instance.

XDC Syntax Example

```
# Designates placed_reg to be placed in Slice site SLICE_X0Y0
set_property LOC SLICE_X0Y0 [get_cells placed_reg]
```

Affected Steps

- Design Floorplanning
- **place_design**

See Also

[BEL](#), page 38

[PACKAGE_PIN](#), page 108

[PBLOCK](#), page 110

LOCK_PINS

LOCK_PINS is a cell property used to specify the mapping of logical LUT inputs (I0, I1, I2, ...) to physical LUT inputs (A6, A5, A4, ...) on the Xilinx FPGA device resource. A common use is to force timing-critical LUT inputs to be mapped to the fastest A6 and A5 physical LUT inputs.

By default, LUT pins are mapped in order from highest to lowest. The highest logical pin is mapped to the highest physical pin.

- ALUT6 placed on an A6LUT bel, would have a default pin mapping of:

```
I5:A6 I4:A5 I3:A4 I2:A3 I1:A2 I0:A1
```

- A LUT5 placed on a D5LUT bel, would have a default pin mapping of:

```
I5:A5 I4:A4 I3:A3 I2:A2 I1:A1
```

- A LUT2 placed on an A6LUT bel, would have a default pin mapping of:

```
I1:A6 I0:A5
```

The LOCK_PINS property is used by the Vivado router, which will not modify pin mappings on locked LUTs even if it would result in improved timing. LOCK_PINS is also important for directed routing. If a pin that is connected by a directed route, is swapped with another pin, the directed route will no longer align with the LUT connection, resulting in an error. All LUT cells driven by a directed route net should have their pins locked using LOCK_PINS. Refer to the *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 7] for more information on directed routing.

Note: DONT_TOUCH does not imply LOCK_PINS.

When running the `phys_opt_design -critical_pin_opt` optimization, a cell with the LOCK_PINS property is not optimized, and the pin mapping specified by LOCK_PINS is retained. Refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4] for more information on the `phys_opt_design` command.

When the LOCK_PINS property is removed from a cell, the pin mapping is cleared and the pins are free to be swapped. However, there is no immediate change to the current pin assignments.

Architecture Support

All architectures

Applicable Objects

- LUT Cells (`get_cells`)

Values

- LOCK_PINS {I0:A6 I1:A5}: One or more pin mapping pairs, assigning LUT logical pins to LUT physical pins using logical-to-physical pin map pairs.
 - The LOCK_PINS value syntax is an unordered list of pin mappings, separated by commas in HDL, or by white space in XDC.
 - The list of possible instance pins ranges from I0 for a LUT1, to I0 through I5 for a LUT6. The physical pins range from A6 (fastest) to A1 for a 6LUT and A5 (fastest) to A1 for a 5LUT.



TIP: The ISE supported values of ALL, or no value to imply ALL, are not supported in the Vivado Design Suite. To lock ALL pins, each pin must be explicitly specified. Any unlisted logical pins are mapped to a physical pin using the default mapping.

Syntax

Verilog Syntax

LOCK_PINS values can be assigned as a Verilog attribute placed on instantiated LUT cells (e.g. LUT6, LUT5, etc).

The following example defines LOCK_PINS with pin mapping logical I1 to A5, and logical I2 to A6, on a LUT cell LUT_inst_0:

```
(* LOCK_PINS = "I1:A5, I2:A6" *) LUT6 #(.INIT(64'h1) ) LUT_inst_0 ( . . .
```

Verilog Example

```
module top (
    i0,
    i1,
    i2,
    i3,
    i4,
    i5,
    o0);
    input i0;
    input i1;
    input i2;
    input i3;
    input i4;
    input i5;
    output o0;

    (* LOCK_PINS = "I1:A5,I2:A6" *)
    LUT6 # (
        .INIT(64'h0000000000000001))
        LUT_inst_0
        (.I0(i0),
         .I1(i1),
```

```

        .I2(i2),
        .I3(i3),
        .I4(i4),
        .I5(i5),
        .O(o0));
endmodule

```

VHDL Syntax

LOCK_PINS values can be assigned as a VHDL attribute placed on instantiated LUT cells (e.g. LUT6, LUT5, etc).

The following example defines LOCK_PINS with pin mapping logical I1 to A5, and logical I2 to A6, on a LUT cell LUT_inst_0:

```

attribute LOCK_PINS : string;
attribute LOCK_PINS of LUT_inst_0 : label is "I1:A5, I2:A6";
. . .

```

VHDL Example:

```

entity top is port (
    i0, i1, i2, i3, i4, i5 : in std_logic;
    o0 : out std_logic
);
end entity top;

architecture struct of top is

attribute lock_pins : string;
attribute lock_pins of LUT_inst_0 : label is "I1:A5, I2:A6";

begin
    LUT_inst_0 : LUT6 generic map (
        INIT => "1"
    ) port map (
        I0 => i0,
        I1 => i1,
        I2 => i2,
        I3 => i3,
        I4 => i4,
        I5 => i5,
        O => o0
    );
end architecture struct;

```

XDC Syntax

The LOCK_PINS property can be set on LUT cells using the set_property Tcl command in the Vivado Design Suite:

```

set_property LOCK_PINS {pin pairs} [get_cells instance_name]

```

Where:

- `instance_name` is one or more LUT cells.



IMPORTANT: *XDC requires white space separation between pin pairs to satisfy the Tcl list syntax, while HDL syntax requires comma-separated values.*

XDC Syntax Example

```
% set myLUT2 [get_cell u0/u1/i_365]
% set_property LOCK_PINS {I0:A5 I1:A6} $myLUT2
% get_property LOCK_PINS $myLUT2
I0:A5 I1:A6
% reset_property LOCK_PINS $myLUT2
% set myLUT6 [get_cell u0/u1/i_768]
% set_property LOCK_PINS I0:A6 ; # mapping of I1 through I5 are dont-cares
```

Affected Steps

- `phys_opt_design`
- `route_design`

See Also

[BEL](#), page 38

[DONT_TOUCH](#), page 56

[LOC](#), page 97

LUTNM

LUTNM instructs the tool to place two LUT5, SRL16, or LUTRAM components with compatible inputs into the same LUT6 site. The LUTNM must be specified in pairs, with two of these specified on compatible instance types with the same group name.

Difference Between LUTNM and HLUTNM

HLUTNM can be used to combine two LUT components that exist in different user hierarchy. Use LUTNM to group two LUT components that exist in the same user hierarchy.

Architecture Support

All architectures

Applicable Objects

- Cells (`get_cells`)
 - LUT (LUT1, LUT2, LUT3, LUT4, LUT5)
 - SRL (SRL16E)
 - LUTRAM (RAM32X1S)

Values

A unique group name

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the instantiation of a LUT. The Verilog attribute must be used in pairs in the same logical hierarchy.

```
(* LUTNM = "group_name" *)
```

Verilog Syntax Example

```
// Designates state0_inst to be placed in same LUT6 as state1_inst
// LUT5: 5-input Look-Up Table with general output (Mapped to a LUT6)
(* LUTNM = "LUT_group1" *) LUT5 #(
    .INIT(32'ha2a2aea2) // Specify LUT Contents
) state0_inst (
    .O(state_out[0]), // LUT general output
    .I0(state_in[0]), // LUT input
    .I1(state_in[1]), // LUT input
    .I2(state_in[2]), // LUT input
    .I3(state_in[3]), // LUT input
    .I4(state_in[4]) // LUT input
);
// End of state0_inst instantiation
// LUT5: 5-input Look-Up Table with general output (Mapped to a LUT6)
// Virtex-7
// Xilinx HDL Language Template, version 2013.2
(* LUTNM = "LUT_group1" *) LUT5 #(
    .INIT(32'h00330073) // Specify LUT Contents
) state1_inst (
    .O(state_out[1]), // LUT general output
    .I0(state_in[0]), // LUT input
    .I1(state_in[1]), // LUT input
    .I2(state_in[2]), // LUT input
    .I3(state_in[3]), // LUT input
    .I4(state_in[4]) // LUT input
);
// End of state1_inst instantiation
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute LUTNM : string;
```

For an instantiated instance, specify the VHDL attribute as follows:

```
attribute LUTNM of instance_name : label is "group_name";
```

Where

- **instance_name** is a LUT1, LUT2, LUT3, LUT4, LUT5, SRL16, or LUTRAM instance.

The VHDL attribute must be used in pairs in the same logical hierarchy.

VHDL Syntax Example

```
-- Designates state0_inst to be placed in same LUT6 as state1_inst
attribute LUTNM : string;
attribute LUTNM of state0_inst : label is "LUT_group1";
attribute LUTNM of state1_inst : label is "LUT_group1";
begin
    -- LUT5: 5-input Look-Up Table with general output (Mapped to SliceM LUT6)
    state0_inst : LUT5
        generic map (
            INIT => X"a2a2aea2") -- Specify LUT Contents
```

```

port map (
    O => state_out(0), -- LUT general output
    I0 => state_in(0), -- LUT input
    I1 => state_in(1), -- LUT input
    I2 => state_in(2), -- LUT input
    I3 => state_in(3), -- LUT input
    I4 => state_in(4)  -- LUT input
);
-- End of state0_inst instantiation
-- LUT5: 5-input Look-Up Table with general output (Mapped to SliceM LUT6)
-- Virtex-7
-- Xilinx HDL Language Template, version 2013.2
State1_inst : LUT5
generic map (
    INIT => X"00330073") -- Specify LUT Contents
port map (
    O => state_out(1), -- LUT general output
    I0 => state_in(0), -- LUT input
    I1 => state_in(1), -- LUT input
    I2 => state_in(2), -- LUT input
    I3 => state_in(3), -- LUT input
    I4 => state_in(4)  -- LUT input
);
-- End of state1_inst instantiation

```

XDC Syntax

```
set_property LUTNM group_name [get_cells instance_name]
```

Where

- **instance_name** is a LUT1, LUT2, LUT3, LUT4, LUT5, SRL16, or LUTRAM instance.

XDC Syntax Example

```

# Designates state0_inst LUT5 to be placed in same LUT6 as state1_inst
set_property LUTNM LUT_group1 [get_cells U1/state0_inst]
set_property LUTNM LUT_group1 [get_cells U2/state1_inst]

```

Affected Steps

place_design

See Also

[HLUTNM](#)

MARK_DEBUG

Use MARK_DEBUG to specify that a net should be debugged using the ChipScope™ tool. This may prevent optimization that may have otherwise occurred to that signal. However, it provides an easy means to later observe the values on this signal during FPGA operation.

Architecture Support

All architectures

Applicable Objects

- Nets (`get_nets`)

- Any net accessible to the internal array.

Note: Some nets may have dedicated connectivity or other aspects that prohibit visibility for debug purposes.

Values

- TRUE
- FALSE

Syntax

Verilog Syntax

To set this attribute, place the proper Verilog attribute syntax before the top-level output port declaration:

```
(* MARK_DEBUG = "{TRUE|FALSE}" *)
```

Verilog Syntax Example

```
// Marks an internal wire for ChipScope debug
(* MARK_DEBUG = "TRUE" *) wire debug_wire,
```

VHDL Syntax

To set this attribute, place the proper VHDL attribute syntax before the top-level output port declaration.

Declare the VHDL attribute as follows:

```
attribute MARK_DEBUG : string;
```

Specify the VHDL attribute as follows:

```
attribute MARK_DEBUG of signal_name : signal is "{TRUE|FALSE}";
```

Where

- **signal_name** is an internal signal.

VHDL Syntax Example

```
signal debug_wire : std_logic;  
attribute MARK_DEBUG : string;  
-- Marks an internal wire for ChipScope debug  
attribute MARK_DEBUG of debug_wire : signal is "TRUE";
```

XDC Syntax

```
set_property MARK_DEBUG value [get_nets net_name]
```

Where

- **net_name** is a signal name.

XDC Syntax Example

```
# Marks an internal wire for ChipScope debug  
set_property MARK_DEBUG TRUE [get_nets debug_wire]
```

Affected Steps

- **place_design**
- ChipScope

See Also

[DONT_TOUCH](#)

PACKAGE_PIN

PACKAGE_PIN specifies a specific placement of a top-level port in the logical design to a physical package pin on the device.



RECOMMENDED: *To assign I/O ports to physical pins on the device package, use the PACKAGE_PINS property rather than LOCS. Use the LOC property to assign logic cells to device resources on the target Xilinx FPGA.*

Architecture Support

All architectures

Applicable Objects

- Ports (`get_ports`)
 - Any top-level port

Values

Package pin name

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the port declaration:

```
(* PACKAGE_PIN = "pin_name" *)
```

Verilog Syntax Example

```
// Designates port CLK to be placed on pin B26  
(* PACKAGE_PIN = "B26" *) input CLK;
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute PACKAGE_PIN : string;
```

Specify the VHDL attribute as follows:

```
attribute PACKAGE_PIN of port_name : signal is "pin_name";
```

VHDL Syntax Example

```
-- Designates CLK to be placed on pin B26
attribute PACKAGE_PIN of CLK : signal is "B26";
```

XDC Syntax

```
set_property PACKAGE_PIN pin_name [get_ports port_name]
```

XDC Syntax Example

```
# Designates CLK to be placed on pin B26
set_property PACKAGE_PIN B26 [get_ports CLK]
```

Affected Steps

- Pin planning
- `place_design`

See Also

[LOC](#)

PBLOCK

PBLOCK is a read-only property attached to cells that assigned to Pblocks in the Vivado Design Suite.

A Pblock is a collection of cells, and one or more rectangular areas or regions that specify the device resources contained by the Pblock. Pblocks are used during floorplanning placement to group related logic and assign it to a region of the target device. Refer to the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 8] for more information on the use of Pblocks in floorplanning your design.

Pblocks are created using the `create_pblock` Tcl command, and are populated with cells using the `add_cells_to_pblock` command. The following code defines a Pblock:

```
create_pblock Pblock_usbEngine
add_cells_to_pblock [get_pblocks Pblock_usbEngine] [get_cells -quiet [list
usbEngine1]]
resize_pblock [get_pblocks Pblock_usbEngine] -add {SLICE_X8Y105:SLICE_X23Y149}
resize_pblock [get_pblocks Pblock_usbEngine] -add {DSP48_X0Y42:DSP48_X1Y59}
resize_pblock [get_pblocks Pblock_usbEngine] -add {RAMB18_X0Y42:RAMB18_X1Y59}
resize_pblock [get_pblocks Pblock_usbEngine] -add {RAMB36_X0Y21:RAMB36_X1Y29}
```

The first line creates the Pblock, giving it a name.

The second line assigns logic cells to the Pblock. In this case, all of the cells in the specified hierarchical module are assigned to the Pblock. Cells that are assigned to a specific Pblock are assigned the PBLOCK property.

The subsequent commands, `resize_pblock`, define the size of the Pblock by specifying a range of device resources that are contained inside the Pblock. A pblock has a grid of four device resource types: SLICE, DSP48, RAMB18, RAMB36. Logic that does not match one of these device types can be placed anywhere in the device. To constrain just the Block RAMs in the level of hierarchy, disable (or simply do not define) the other Pblock grids.

Refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4] for details on the specific Tcl commands mentioned above.

Architecture Support

All architectures

Applicable Objects

- Cells (`get_cells`)

Values

- **NAME:** The property value is the name of the Pblock that the cell is assigned to. The Pblock name is defined when the Pblock is created with the `create_pblock` command.

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

The Pblock can be defined in the XDC file, or directly in the design, with the Tcl command:

```
create_pblock <pblock_name>
```

XDC Example

The following code defines a Pblock:

```
create_pblock Pblock_usbEngine
add_cells_to_pblock [get_pblocks Pblock_usbEngine] [get_cells -quiet [list
usbEngine1]]
resize_pblock [get_pblocks Pblock_usbEngine] -add {SLICE_X8Y105:SLICE_X23Y149}
resize_pblock [get_pblocks Pblock_usbEngine] -add {DSP48_X0Y42:DSP48_X1Y59}
resize_pblock [get_pblocks Pblock_usbEngine] -add {RAMB18_X0Y42:RAMB18_X1Y59}
resize_pblock [get_pblocks Pblock_usbEngine] -add {RAMB36_X0Y21:RAMB36_X1Y29}
```

Affected Steps

- Design Floorplanning
- `place_design`

See Also

[BEL, page 38](#)

[LOC, page 97](#)

POST_CRC

The Post CRC (POST_CRC) constraint enables or disables the Cyclic Redundancy Check (CRC) error detection feature for configuration logic, allowing for notification of any possible change to the configuration memory.

Enabling the POST_CRC property controls the generation of a pre-computed CRC value in the bitstream. As the configuration data frames are loaded, the device calculates a Cyclic Redundancy Check (CRC) value from the configuration data packets. After the configuration data frames are loaded, the configuration bitstream can issue a Check CRC instruction to the device, followed by the pre-computed CRC value. If the CRC value calculated by the device does not match the expected CRC value in the bitstream, the device pulls INIT_B Low and aborts configuration. For more information refer to the *7 Series FPGA Configuration User Guide (UG470)* [Ref 1].

When CRC is disabled a constant value is inserted in the bitstream in place of the CRC, and the device does not calculate a CRC.

Architecture Support

- Artix-7
- Virtex-7
- Kintex-7

Applicable Objects

- Design (`current_design`)
 - The current implemented design.

Values

- DISABLE: Disables the Post CRC checking feature (default).
- ENABLE: Enables the Post CRC checking feature.

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property POST_CRC ENABLE | DISABLE [current_design]
```

XDC Syntax Example

```
set_property POST_CRC Enable [current_design]
```

Affected Steps

- `write_bitstream`
- `launch_runs`

See Also

[POST_CRC_ACTION](#), page 114

[POST_CRC_FREQ](#), page 116

[POST_CRC_INIT_FLAG](#), page 118

[POST_CRC_SOURCE](#), page 120

POST_CRC_ACTION

The Post CRC Action property (POST_CRC_ACTION) applies to the configuration logic CRC error detection mode. This property determines the action that the device takes when a CRC mismatch is detected: correct the error, continue operation, or stop configuration.

During readback, the syndrome bits are calculated for every frame. If a single bit error is detected, the readback is stopped immediately. If correction is enabled using the POST_CRC_ACTION property, then the readback CRC logic performs correction on single bit errors. The frame in error is readback again, and using the syndrome information, the bit in error is fixed and written back to the frame. If the POST_CRC_ACTION is set to `Correct_And_Continue`, then the readback logic starts over from the first address. If the `Correct_And_Halt` option is set, the readback logic stops after correction. Refer to the *7 Series FPGA Configuration User Guide (UG470)* [Ref 1].

This property is only applicable when `POST_CRC` is set to `ENABLE`.

Architecture Support

- Artix-7
- Virtex-7
- Kintex-7

Applicable Objects

- Design (`current_design`)
 - The current implemented design.

Values

- **HALT:** If a CRC mismatch is detected, stop reading back the bitstream, stop computing the comparison CRC, and stop making the comparison against the pre-computed CRC.
- **CONTINUE:** If a CRC mismatch is detected by the CRC comparison, continue reading back the bitstream, computing the comparison CRC, and making the comparison against the pre-computed CRC.
- **CORRECT_AND_CONTINUE:** If a CRC mismatch is detected by the CRC comparison, it is corrected and continues reading back the bitstream, computing the comparison CRC, and making the comparison against the pre-computed CRC.
- **CORRECT_AND_HALT:** If a CRC mismatch is detected, it is corrected and stops reading back the bitstream, computing the comparison CRC, and making the comparison against the pre-computed CRC.

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property POST_CRC_ACTION <VALUE> [get_ports port_name]
```

Where:

- <VALUE> is one of the accepted values for the POST_CRC_ACTION property.

XDC Syntax Example

```
set_property POST_CRC_ACTION correct_and_continue [current_design]
```

Affected Steps

- `write_bitstream`
- `launch_runs`

See Also

[POST_CRC](#), page 112

[POST_CRC_FREQ](#), page 116

[POST_CRC_INIT_FLAG](#), page 118

[POST_CRC_SOURCE](#), page 120

POST_CRC_FREQ

The Post CRC Frequency property (POST_CRC_FREQ) controls the frequency with which the configuration CRC check is performed for the current design.

This property is only applicable when [POST_CRC](#) is set to ENABLE. Enabling the POST_CRC property controls the periodic comparison of a pre-computed CRC value in the bitstream with an internal CRC value computed by readback of the configuration memory cells.

The POST_CRC_FREQ defines the frequency in MHz of the readback function, with a default value of 1 MHz.

Architecture Support

- Artix-7
- Virtex-7
- Kintex-7

Applicable Objects

- Design (`current_design`)
 - The current implemented design.

Values

- Specify the frequency in MHz as an integer with one of the following accepted values:
 - 1 2 3 4 6 7 8 10 12 13 16 17 22 25 26 27 33 40 44 50 66 100
 - Default = 1 MHz

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property POST_CRC_FREQ <VALUE> [current_design]
```

Where:

- <VALUE> is one of the accepted values for the POST_CRC_FREQ property.

XDC Syntax Example

```
set_property POST_CRC_FREQ 50 [current_design]
```

Affected Steps

- `write_bitstream`
- `launch_runs`

See Also

[POST_CRC](#), page 112

[POST_CRC_ACTION](#), page 114

[POST_CRC_INIT_FLAG](#), page 118

[POST_CRC_SOURCE](#), page 120

POST_CRC_INIT_FLAG

The Post CRC INIT Flag property (POST_CRC_INIT_FLAG) determines whether the INIT_B pin is enabled as an output for the SEU (Single Event Upset) error signal.

The error condition is always available from the FRAME_ECC site. However, when the POST_CRC_INIT_FLAG is ENABLED, which is the default, the INIT_B pin also flags the CRC error condition when it occurs.

This property is only applicable when [POST_CRC](#) is set to ENABLE.

Architecture Support

- Artix-7
- Virtex-7
- Kintex-7

Applicable Objects

- Design (`current_design`)
 - The current implemented design.

Values

- DISABLE: Disables the use of the INIT_B pin, with the FRAME_ECC site as the sole source of the CRC error signal.
- ENABLE: Leaves the INIT_B pin enabled as a source of the CRC error signal. (Default)

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property POST_CRC_INIT_FLAG ENABLE | DISABLE [current_design]
```

XDC Syntax Example

```
set_property POST_CRC_INIT_FLAG Enable [current_design]
```

Affected Steps

- `write_bitstream`
- `launch_runs`

See Also

[POST_CRC](#), page 112

[POST_CRC_ACTION](#), page 114

[POST_CRC_FREQ](#), page 116

[POST_CRC_SOURCE](#), page 120

POST_CRC_SOURCE

The Post CRC Source (POST_CRC_SOURCE) constraint specifies the source of the CRC value when the configuration logic CRC error detection feature is used for notification of any possible change to the configuration memory.

This property is only applicable when [POST_CRC](#) is set to ENABLE.

Enabling the POST_CRC property controls the generation of a pre-computed CRC value in the bitstream. As the configuration data frames are loaded, the device calculates a Cyclic Redundancy Check (CRC) value from the configuration data packets. The POST_CRC_SOURCE property defines the expected CRC value as either coming from a pre-computed value, or as being taken from the configuration data in the first readback pass.

Architecture Support

- Artix-7
- Virtex-7
- Kintex-7

Applicable Objects

- Design (`current_design`)
 - The current implemented design.

Values

- PRE_COMPUTED: Determine an expected CRC value from the bitstream. (Default)
- FIRST_READBACK: Extract the actual CRC value from the first readback pass, to use for comparison with future readback iterations.

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property POST_CRC_SOURCE FIRST_READBACK | PRE_COMPUTED [current_design]
```

XDC Syntax Example

```
set_property POST_CRC_SOURCE PRE_COMPUTED [current_design]
```

Affected Steps

- `write_bitstream`
- `launch_runs`

See Also

[POST_CRC](#), page 112

[POST_CRC_ACTION](#), page 114

[POST_CRC_FREQ](#), page 116

[POST_CRC_INIT_FLAG](#), page 118

PROHIBIT

PROHIBIT specifies that a pin or site cannot be used for placement.

Architecture Support

All architectures

Applicable Objects

- Sites (`get_sites`)
- BELs (`get_bels`)

Values

1

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

```
set_property PROHIBIT 1 [get_sites site]
```

XDC Syntax Example

```
# Prohibit the use of package pin Y32
set_property prohibit 1 [get_sites Y32]
```

Affected Steps

- I/O planning
- `place_design`

PULLDOWN

PULLDOWN applies a weak logic low level on a tri-stateable output or bidirectional port to prevent it from floating. The PULLDOWN property guarantees a logic Low level to allow tri-stated nets to avoid floating when not being driven.

Input buffers (e.g., IBUF), 3-state output buffers (e.g., OBUFT), and bidirectional buffers (e.g., IOBUF) can have a weak pull-up resistor, a weak pull-down resistor, or a weak “keeper” circuit. This feature can be invoked by adding the one of the following properties to the net object connected to the buffer:

- PULLUP
- PULLDOWN
- KEEPER

For more information see *Vivado Design Suite 7 Series FPGA Libraries Guide (UG953)* [Ref 9].

Architecture Support

All

Applicable Objects

- Nets connected to I/O Buffers (`get_nets`)

Values

- TRUE | YES: Use a pulldown circuit to avoid signal floating when not being driven.
- FALSE | NO: Do not use a pulldown circuit. Default.

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the module or instantiation. Specify as follows:

```
(* PULLDOWN = " {YES|NO|TRUE|FALSE}" *)
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute pulldown: string;
```

Specify the VHDL attribute as follows:

```
attribute pulldown of signal_name : signal is "{YES|NO|TRUE|FALSE}";
```

XDC Syntax

```
set_property PULLDOWN {TRUE|FALSE} [get_nets net_name]
```

Where

- **net_name** is the name of a net connected to an IBUF, OBUFT, or IOBUF cell.

XDC Syntax Example

```
# Use a pulldown circuit  
set_property PULLDOWN true [get_nets n1]
```

Affected Steps

- Logical to Physical Mapping

See Also

[KEEPER](#), page 95

[PULLUP](#), page 125

PULLUP

PULLUP applies a weak logic High on a tri-stateable output or bidirectional port to prevent it from floating. The PULLUP property guarantees a logic High level to allow tri-stated nets to avoid floating when not being driven.

Input buffers (e.g., IBUF), 3-state output buffers (e.g., OBUFT), and bidirectional buffers (e.g., IOBUF) can have a weak pull-up resistor, a weak pull-down resistor, or a weak “keeper” circuit. This feature can be invoked by adding the one of the following properties to the net object connected to the buffer:

- PULLUP
- PULLDOWN
- KEEPER

For more information see *Vivado Design Suite 7 Series FPGA Libraries Guide (UG953)* [Ref 9].

Architecture Support

All

Applicable Objects

- Nets connected to I/O Buffers (`get_nets`)

Values

- TRUE | YES: Use a pullup circuit to avoid signal floating when not being driven.
- FALSE | NO: Do not use a pullup circuit. Default.

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the module or instantiation. Specify as follows:

```
(* PULLUP = " {YES|NO|TRUE|FALSE}" *)
```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute pullup: string;
```

Specify the VHDL attribute as follows:

```
attribute pullup of signal_name : signal is "{YES|NO|TRUE|FALSE}";
```

XDC Syntax

```
set_property PULLUP {TRUE|FALSE} [get_nets net_name]
```

Where

- **net_name** is the name of a net connected to an IBUF, OBUFT, or IOBUF cell.

XDC Syntax Example

```
set_property PULLUP true [get_nets n1]
```

Affected Steps

- Logical to Physical Mapping

See Also

[KEEPER](#), page 95

[PULLDOWN](#), page 123

REF_NAME

This is a read-only property on cells of the design indicating a logical cell name that uniquely identifies the cell.

The REF_NAME property is defined automatically by the Vivado Design Suite, and can not be modified by the user in either HDL or XDC. It is intended for reference only.

The property does not influence any steps but is very useful in defining filters and other Vivado Tcl command queries to identify specific cells or other objects.

For example, to select the clock pins on RAM cells, you can filter the pin objects based on the REF_NAME property of the cells:

```
get_pins -hier */*W*CLK -filter {REF_NAME =~ *RAM* && IS_PRIMITIVE}
```

Architecture Support

All architectures

Applicable Objects

- Cells (get_cells)

Values

Not applicable

Syntax

Not applicable

Affected Steps

None

REF_PIN_NAME

This is a read-only property on pins in the design indicating a logical name that uniquely identifies the pin.

The REF_PIN_NAME is automatically defined from the NAME or HIERARCHICAL NAME of the pin, and can not be modified by the user in either HDL or XDC. It is intended for reference only.

The property does not influence any steps but is very useful in defining filters and other Vivado Tcl command queries to identify specific cells or other objects.

Architecture Support

All architectures

Applicable Objects

- Pins (get_pins)

Values

Not applicable

Syntax

Not applicable

Affected Steps

None

RLOC

Relative Location (RLOC) constraints define the relative placement of logic elements assigned to a set, such as an H_SET, HU_SET, or U_SET.

When RLOC is present in the RTL source files, the H_SET, HU_SET, or U_SET properties get translated into a read-only RPM property on cells in the synthesized netlist. The RLOC property is preserved, but becomes a read-only property after synthesis. For more information on using these properties, and defining RPMs, refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 6].

You can define the placement of any element within the set relative to other elements in the set, regardless of the eventual placement of the entire group onto the target device. For example, if RLOC constraints are applied to a group of eight flip-flops organized in a column, the mapper maintains the column and moves the entire group of flip-flops as a single unit. In contrast, the LOC constraint specifies the absolute location of a design element on the target device, without reference to other design elements.

Architecture Support

All architectures

Applicable Objects

- Instances or Modules in the RTL source files.

Values

The Relative Location constraint is specified using a slice-based XY coordinate system.

RLOC=XmYn

Where:

- m is an integer representing the X coordinate value.
- n is an integer representing the Y coordinate value.



TIP: Because the X and Y numbers in Relative Location (RLOC) constraints define only the order and relationship between design elements, and not their absolute locations on the target device, their numbering can include negative integers.

Syntax

Verilog Syntax

The RLOC property is a Verilog attribute defining the relative placement of design elements within a set specified by H_SET, HU_SET, or U_SET in the RTL source files. Place the Verilog attribute immediately before the instantiation of a logic element.

```
(* RLOC = "XmYn", HU_SET = "h0" *) FD sr0 (.C(clk), .D(sr_1n), .Q(sr_0));
```

Verilog Example

The following Verilog module defines RLOC property for the shift register Flops in the ffs hierarchical module.

```
module inv (input a, output z);

    LUT1 #(.INIT(2'h1)) lut1 (.IO(a), .O(z));

endmodule // inv

module ffs
(
    input  clk,
    input  d,
    output q
);

    wire  sr_0, sr_0n;
    wire  sr_1, sr_1n;
    wire  sr_2, sr_2n;
    wire  sr_3, sr_3n;
    wire  sr_4, sr_4n;
    wire  sr_5, sr_5n;
    wire  sr_6, sr_6n;
    wire  sr_7, sr_7n;

    wire  inr, inrn, outr;

    inv i0 (sr_0, sr_0n);
    inv i1 (sr_1, sr_1n);
    inv i2 (sr_2, sr_2n);
    inv i3 (sr_3, sr_3n);
    inv i4 (sr_4, sr_4n);
    inv i5 (sr_5, sr_5n);
    inv i6 (sr_6, sr_6n);
    inv i7 (sr_7, sr_7n);
    inv i8 (inr, inrn);

    (* RLOC = "X0Y0" *) FD sr0 (.C(clk), .D(sr_1n), .Q(sr_0));
    (* RLOC = "X0Y1" *) FD sr1 (.C(clk), .D(sr_2n), .Q(sr_1));
    (* RLOC = "X0Y2" *) FD sr2 (.C(clk), .D(sr_3n), .Q(sr_2));
    (* RLOC = "X0Y3" *) FD sr3 (.C(clk), .D(sr_4n), .Q(sr_3));
    (* RLOC = "X0Y4" *) FD sr4 (.C(clk), .D(sr_5n), .Q(sr_4));
    (* RLOC = "X0Y5" *) FD sr5 (.C(clk), .D(sr_6n), .Q(sr_5));
```

```

(* RLOC = "X0Y6" *) FD sr6 (.C(clk), .D(sr_7n), .Q(sr_6));
(* RLOC = "X0Y7" *) FD sr7 (.C(clk), .D(inrn), .Q(sr_7));
(* LOC = "SLICE_X0Y0" *) FD inq (.C(clk), .D(d), .Q(inr));
FD outq (.C(clk), .D(sr_0n), .Q(outr));

assign q = outr;

endmodule // ffs

```



TIP: In the preceding example, the presence of the RLOC property implies the use of the H_SET property on the FD instances in the ffs hierarchical module.

When using the modules defined in the preceding example, you will need to specify the KEEP_HIERARCHY property to instances of the ffs module to preserve the hierarchy and define the RPM in the synthesized design:

```

module top
(
input  clk,
input  d,
output q
);

wire  c1, c2;

(* RLOC_ORIGIN = "X1Y1", KEEP_HIERARCHY = "YES" *) ffs u0 (clk, d, c1);
(* RLOC_ORIGIN = "X3Y3", KEEP_HIERARCHY = "YES" *) ffs u1 (clk, c1, c2);
(* RLOC_ORIGIN = "X5Y5", KEEP_HIERARCHY = "YES" *) ffs u2 (clk, c2, q);

endmodule // top

```

VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute RLOC: string;
```

Specify the VHDL constraint as follows:

```
attribute RLOC of {component_name | entity_name | label_name} :
{component|entity|label} is "XmYn";
```

Where:

- {component_name | entity_name | label_name} is a choice of one design element.
- {component|entity|label} is the instance ID of the design element.
- XmYn defines the RLOC value for the specified design element.

XDC Syntax

The RLOC property can not be defined using XDC constraints. The RLOC property defines the relative locations of objects in a relatively placed macro (RPM), and results in read-only RPM and RLOC properties in the netlist of synthesized designs.



TIP: You can use the `create_macro` and `update_macro` commands to define macro objects in the Vivado Design Suite, that act like RPMs within the design. Refer to the Vivado Design Suite Tcl Command Reference (UG835) [Ref 4] for more information on these commands.

Affected Steps

- Logical to Physical Mapping
- place_design
- synth_design

See Also

[H_SET and HU_SET, page 65](#)

[RLOC, page 129](#)

[RLOCS, page 133](#)

[RLOC_ORIGIN, page 135](#)

[RPM, page 140](#)

[RPM_GRID, page 141](#)

[U_SET, page 146](#)

RLOCS

RLOCS is a read-only property that is assigned to an XDC macro object that is created by the `create_macro` Tcl command in the Vivado Design Suite. The RLOCS property is assigned to the macro when it is updated with the `update_macro` command. Refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4] for more information on these commands.

Like relatively placed macros (RPMs), XDC macros enable relative placement of groups of cells. Macros are similar to RPMs in many ways, yet also have significant differences:

- RPMs are defined in the RTL source files by a combination of the RLOC property and the H_SET, HU_SET, or U_SET property.
- RPMs cannot be edited in the post-synthesis design.
- Macros are created from leaf cells that are grouped together with relative placement, after synthesis, and can be edited.
- RPMs cannot be automatically converted to macros.
- RPMs are not design objects, and the XDC macro commands cannot be used on RPMs.

The RLOCS property reflects the relative placement values specified by the `update_macro` command, as represented by the `rlocs` argument:

```
"cell0 rloc0 cell1 rloc1 ... cellN rlocN"
```

You can use `update_macro` command to change the RLOCS property assigned to an XDC macro object.

The RLOCS property is converted to an RLOC property on each of the individual cells that are part of the XDC macro. The RLOC property then functions in the same way it does for an RPM, by defining the relative placement of cells in the macro.

Architecture Support

All architectures

Applicable Objects

- Cells (`get_cells`)

Values

- Cell1 RLOC1 Cell2 RLOC2 Cell3 RLOC3...: The name of a cell in the macro paired with the relative location of the cell in the macro, defined for each cell in the macro.

Syntax

Verilog Syntax

Not applicable

VHDL Syntax

Not applicable

XDC Syntax

The RLOCS property is indirectly defined when an XDC macro is created and populated with cells and relative locations:

XDC Example

```
create_macro macro1
update_macro macro1 {u1/sr3 X0Y0 u1/sr4 X1Y0 u1/sr5 X0Y1}

report_property -all [get_macros macro1]
Property      Type      Read-only  Visible  Value
ABSOLUTE_GRID bool      true       true     0
CLASS         string   true       true     macro
NAME          string   true       true     macro1
RLOCS         string*  true       true     u1/sr3 X0Y0 u1/sr4 X1Y0 u1/sr5
```

Affected Steps

- Logical to Physical Mapping
- place_design
- synth_design

See Also

[H_SET and HU_SET, page 65](#)

[RLOC, page 129](#)

[RLOC_ORIGIN, page 135](#)

[RPM, page 140](#)

[RPM_GRID, page 141](#)

[U_SET, page 146](#)

RLOC_ORIGIN

The RLOC_ORIGIN property provides an absolute location, or LOC, for the relatively placed macro (RPM) in the RTL design. For more information on defining RPMs, and using the RLOC_ORIGIN property, refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 6].

RPMs are defined by assigning design elements to a set using the H_SET, HU_SET, or U_SET properties in the RTL design. The design elements are then assigned a relative placement to one another using the RLOC property. You can define the relative placement of any element within the set relative to other elements in the set, regardless of the eventual placement of the entire group onto the target device.

Having defined the elements of an RPM, and their relative placement, the RLOC_ORIGIN property lets you define the absolute placement of the RPM onto the target device. The RLOC_ORIGIN property is converted into LOC constraint during synthesis.

In the Vivado Design Suite, the RLOC_ORIGIN property defines the lower-left corner of the RPM. This is most often the design element whose RLOC property is X0Y0. Each remaining cell in the RPM set is placed on the target device using its relative location (RLOC) as an offset from the group origin (RLOC_ORIGIN).

Architecture Support

All architectures.

Applicable Objects

- Instances within the RTL source file.

Values

The Relative Location constraint is specified using a slice-based XY coordinate system.

```
RLOC_ORIGIN=XmYn
```

Where:

- m is an integer representing the absolute X coordinate on the target device of the lower-left corner of the RPM.
- n is an integer representing the absolute Y coordinate on the target device of the lower-left corner of the RPM.

Syntax

Verilog Syntax

The RLOC_ORIGIN property is a Verilog attribute defining the absolute placement of an RPM on the target device. Place the Verilog attribute immediately before the instantiation of a logic element.

```
(* RLOC_ORIGIN = "XmYn", HU_SET = "h0" *) FD sr0 (.C(clk), .D(sr_1n), .Q(sr_0));
```

Verilog Example

The following top-level Verilog module defines the RLOC_ORIGIN property for the ffs modules in the design.

```
module top
(
  input  clk,
  input  d,
  output q
);

  wire  c1, c2;

  (* RLOC_ORIGIN = "X1Y1", KEEP_HIERARCHY = "YES" *) ffs u0 (clk, d, c1);
  (* RLOC_ORIGIN = "X3Y3", KEEP_HIERARCHY = "YES" *) ffs u1 (clk, c1, c2);
  (* RLOC_ORIGIN = "X5Y5", KEEP_HIERARCHY = "YES" *) ffs u2 (clk, c2, q);

endmodule // top
```

The following example is very similar to the first, except that the RLOC_ORIGIN is only assigned to the first ffs module, u0, and the rest are defined with RLOC properties for relative placement:

```
module top
(
  input  clk,
  input  d,
  output q
);

  wire  c1, c2;

  // what would happen if the origin places the RPM outside
  // device?

  (* RLOC_ORIGIN = "X74Y15", RLOC = "X0Y0" *) ffs u0 (clk, d, c1);
  (* RLOC = "X1Y1" *) ffs u1 (clk, c1, c2);
  (* RLOC = "X2Y2" *) ffs u2 (clk, c2, q);

endmodule // top
```

VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute RLOC_ORIGIN: string;
```

Specify the VHDL constraint as follows:

```
attribute RLOC_ORIGIN of {component_name | entity_name | label_name} :  
{component|entity|label} is "XmYn";
```

Where:

- {component_name | entity_name | label_name} is a choice of one design element.
- {component|entity|label} is the instance ID of the design element.
- `XmYn` defines the RLOC_ORIGIN value for the specified design element.

XDC Syntax

The RLOC_ORIGIN property translates to the LOC property in the synthesized design. You can specify the LOC property of RPMs by placing one of the elements of the RPM onto the target device. The other elements of the RPM will be placed relative to that location, and assigned to LOC property.

Affected Steps

- Logical to Physical Mapping
- place_design
- synth_design

See Also

[H_SET and HU_SET, page 65](#)

[RLOC, page 129](#)

[RLOCS, page 133](#)

[RPM, page 140](#)

[RPM_GRID, page 141](#)

[U_SET, page 146](#)

ROUTE_STATUS

ROUTE_STATUS is a read-only property that is assigned to nets by the Vivado router to reflect the current state of the routing on the net.

The property can be queried by the individual net, or group of nets, using the `get_property` or `report_property` commands.

The property is used by the `report_route_status` command to return the ROUTE_STATUS of the whole design.

Architecture Support

All architectures.

Applicable Objects

- Nets (`get_nets`)

Values

- ROUTED: The net is fully placed and routed.
- PARTIAL: All pins and/or ports for the net are placed and some of the net is routed, but portions of the net are unrouted and `route_design` should be run.
- UNPLACED: The route has some unplaced pins or ports, and `place_design` should be run to complete the placement.
- UNROUTED: All pins and/or ports for the net are placed, but no route data exists on the net, and `route_design` should be run to complete the route.
- INTRASITE: The entire route is completed within the same Site on the target device, and no routing resources were required to complete the connection. This is not an error.
- NOLOADS: The route either has no logical loads, or has no routable load pins, and so needs no routing. This is not an error.
- NODRIVER: The route either has no logical driver, or has no routable driver, and so needs no routing. This is a design error.
- HIERPORT: The route is connected to a top-level hierarchical port that either has no routable loads or no routable drivers. This is not an error.
- ANTENNAS: The route has at least one antenna (a branch leaf that connects to a site pin, but that site pin does not show that it is connected to this logical net) or the route has at least one island (a section of routing that is not connected to any of the site pins associated with the logical net). This is a routing error.

- **CONFLICTS:** The router has one or more of the following routing errors:
 - **Routing conflict:** One or more of the nodes in this route are also used in some other route, or another branch of this route.
 - **Site pin conflict:** The logical net that is connected to the given site pin from inside the site is different from the logical net that is connected via the route to the outside of the site.
 - **Invalid site conflict:** The route connects to a site pin on a site where the programming of the site is in an invalid state, making it impossible to determine if the route is connected correctly within the site.
- **ERROR:** There was an internal error in determining the route status.
- **NONET:** The net object specified for route status does not exist, or could not be found as entered.
- **NOROUTE:** No routing object could be retrieved for the specified net due to an error.
- **NOROUTESTORAGE:** No route storage object is available for this device due to an error.
- **UNKNOWN:** The state of the route can not be calculated due to an error.

Syntax

The ROUTE_STATUS property is an enumerated property with one of the preceding property values. It is a read-only property assigned by the Vivado router and cannot be directly modified.

Affected Steps

- Route Design

RPM

The RPM property is a read-only property assigned to the logic elements of a set as defined by the H_SET, HU_SET, or U_SET property in the RTL source files.

When RLOC is also present in the RTL source files, the H_SET, HU_SET, and U_SET properties get translated to a read-only RPM property on cells in the synthesized netlist. The HU_SET and U_SET property are visible on the RTL source file in the Text editor in the Vivado Design Suite. However, in the Properties window of a cell object, the RPM property is displayed. For more information on using these properties, and defining RPMs, refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 6].

Architecture Support

All architectures.

Applicable Objects

- Cells in the synthesized design (`get_cells`)

Values

- NAME: The name of the RPM as it is derived from the set definition by the presence of the RLOC property together with the H_SET, HU_SET, or U_SET property in the RTL source files.

Syntax

The RPM property is a read-only property derived during synthesis of an RTL design with RLOC defined together with one of H_SET, HU_SET, or U_SET to define the RPM. The RPM property cannot be directly defined or edited.

See Also

[H_SET and HU_SET, page 65](#)

[RLOC, page 129](#)

[RLOCS, page 133](#)

[RLOC_ORIGIN, page 135](#)

[RPM_GRID, page 141](#)

[U_SET, page 146](#)

RPM_GRID

The RPM_GRID property defines the RLOC grids as absolute coordinates instead of relative coordinates. The RPM_GRID system is used for heterogeneous RPMs where the cells belong to different site types (such as a combination of slices, block RAM, and DSP). Because the cells may occupy sites of various sizes, the RPM_GRID system uses absolute RPM_GRID coordinates that are derived directly from the target device.

The RPM_GRID values are visible in the Site Properties window of the Vivado Integrated Design Environment (IDE) when a specific site is selected in the Device window. The coordinates can also be queried with Tcl commands using the RPM_X and RPM_Y site properties. For more information on using the RPM_GRID property, and defining RPMs with absolute coordinates, refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 6].

Architecture Support

All architectures.

Applicable Objects

- Cells (`get_cells`)

Values

- "GRID": The RPM_GRID property and GRID keyword combine to inform the Vivado Design Suite that the specified RLOCs are absolute grid coordinates from the target device, rather than the relative coordinates usually specified by RLOC.

Syntax

Verilog Syntax

Place the Verilog attribute immediately before the module or instantiation. Specify as follows:

```
(* RPM_GRID = "GRID" *)
```

Verilog Example

```
module iddr_regs
(
  input  clk, d,
  output y, z
);
```

```

(* RLOC = "X130Y195" *) IDDR ireg (.C(clk_i), .D(d), .Q1(q1), .Q2(q2));
defparam ireg.DDR_CLK_EDGE = "SAME_EDGE";
(* RLOC = "X147Y194" *) FD q1reg (.C(clk_i), .D(q1), .Q(y));
(* RLOC = "X147Y194", RPM_GRID = "GRID" *) FD q2reg (.C(clk_i), .D(q2), .Q(z));

endmodule // iddr_regs

```

VHDL Syntax

To use the RPM_GRID system, first define the attribute, then add the attribute to one of the design elements:

```
attribute RPM_GRID of ram0 : label is "GRID";
```

Declare the VHDL constraint as follows:

```
attribute RPM_GRID : string;
```

Specify the VHDL constraint as follows:

```
attribute RPM_GRID of {component_name | entity_name} :
{component|entity} is "GRID";
```

XDC Syntax

The RPM_GRID property is assigned in the RTL source file, and cannot be defined in XDC files or with Tcl commands. However, for XDC macros, the corresponding construct is the `-absolute_grid` option used with the `update_macros` command.

Affected Steps

- Logical to Physical Mapping
- place_design
- synth_design

See Also

[H_SET and HU_SET](#), page 65

[RLOC](#), page 129

[RLOCS](#), page 133

[RLOC_ORIGIN](#), page 135

[RPM](#), page 140

[U_SET](#), page 146

SLEW

SLEW specifies output buffer slew rate for output buffers configured with I/O standards that support programmable output slew rates.

Architecture Support

All architectures.

Applicable Objects

- Ports (`get_ports`)
 - Output or bidirectional ports connected
- Cells (`get_cells`)
 - Output Buffers (all OBUF variants)

Values

- SLOW (default)
- FAST

Syntax

Verilog Syntax

To set this attribute when inferring I/O buffers, place the proper Verilog attribute syntax before the top-level output port declaration.

```
(* DRIVE = "{SLOW|FAST}" *)
```

Verilog Syntax Example

```
// Sets the Slew rate to be FAST
(* SLEW = "FAST" *) output FAST_DATA,
```

Alternative Verilog Syntax Example

To set SLEW when the output or bidirectional buffer is instantiated, assign the SLEW parameter on the instantiated output buffer.



RECOMMENDED: Use the instantiation template from the Language Templates or the Vivado Design Suite 7 Series FPGA Libraries Guide (UG953) [Ref 9] to specify the proper syntax.

The following example sets the slew rate on the OBUF instance named `fast_data_obuf` to FAST:

```
// OBUF: Single-ended Output Buffer
//      Virtex-7
// Xilinx HDL Language Template, version 2013.2
OBUF #(
    .DRIVE(12),      // Specify the output drive strength
    .IOSTANDARD("DEFAULT"), // Specify the output I/O standard
    .SLEW("FAST") // Specify the output slew rate
) fast_data_obuf (
    .O(FAST_DATA), // Buffer output (connect directly to top-level port)
    .I(fast_data_int) // Buffer input
);
// End of fast_data_obuf instantiation
```

VHDL Syntax

To set this attribute when inferring I/O buffers, place the proper VHDL attribute syntax before the top-level output port declaration.

Declare the VHDL attribute as follows:

```
attribute SLEW : string;
```

Specify the VHDL attribute as follows:

```
attribute SLEW of port_name : signal is value;
```

Where

- `port_name` is a top-level output port.

VHDL Syntax Example

```
FAST_DATA : out std_logic;
attribute SLEW : string;
-- Sets the Slew rate to be FAST
attribute SLEW of STATUS : signal is "FAST";
```

Alternative VHDL Syntax Example

To set SLEW when the output or bidirectional buffer is instantiated, assign the SLEW generic on the instantiated output buffer.



RECOMMENDED: Use the instantiation template from the Language Templates or the Vivado Design Suite 7 Series FPGA Libraries Guide (UG953) [Ref 9] to specify the proper syntax.

The following example sets the slew rate on the OBUF instance named `fast_data_obuf` to FAST.

```
-- OBUF: Single-ended Output Buffer
--       Virtex-7
-- Xilinx HDL Language Template, version 2013.2
Fast_data_obuf : OBUF
  generic map (
    DRIVE => 12,
    IOSTANDARD => "DEFAULT",
    SLEW => "FAST")
  port map (
    O => FAST_DATA, -- Buffer output (connect directly to top-level port)
    I => fast_data_int -- Buffer input
  );
-- End of fast_data_obuf instantiation
```

XDC Syntax

```
set_property SLEW value [get_ports port_name]
```

Where

- `port_name` is an output or bidirectional port.

XDC Syntax Example

```
# Sets the Slew rate to be FAST
set_property SLEW FAST [get_ports FAST_DATA]
```

Affected Steps

- I/O Planning
- Report Noise
- Report Power

See Also

Refer to the following design elements in the *Vivado Design Suite 7 Series FPGA Libraries Guide* (UG953) [Ref 9].

- OBUF
- OBUFT
- IOBUF
- IOBUF_DCIEN
- IOBUF_INTERMDISABLE

U_SET

Groups design elements with attached Relative Location (RLOC) constraints that are distributed throughout the design hierarchy into a single set.

U_SET is an attribute within the HDL design source files, and does not appear in the synthesized or implemented design. U_SET is used when defining Relatively Placed Macros, or RPMs in the RTL design. For more information on using these properties, and defining RPMs, refer to the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 6].

While H_SET or HU_SET are used to define sets of logic elements based on the design hierarchy, you can manually create a User-defined set of logic elements, or U_SET, that is not dependant on the hierarchy of the design.

When RLOC is also present in the RTL source files, the H_SET, HU_SET, and U_SET properties get translated to a read-only RPM property on cells in the synthesized netlist. The HU_SET and U_SET property are visible on the RTL source file in the Text editor in the Vivado Design Suite. However, in the Properties window of a cell object, the RPM property is displayed.



IMPORTANT: *When attached to hierarchical modules, the U_SET constraint propagates downward through the hierarchy to any primitive symbols that are assigned RLOC constraints.*

Architecture Support

All architectures.

Applicable Objects

The U_Set constraint may be used in one or more of the following design elements, or categories of design elements. Refer to the *Vivado Design Suite 7 Series FPGA Libraries Guide* (UG953) [Ref 9] for more information on the specific design elements:

- Registers
- FMAP
- Macro Instance
- ROM
- RAMS
- RAMD
- MULT18X18S
- RAMB4_Sm_Sn

- RAMB4_Sn
- RAMB16_Sm_Sn
- RAMB16_Sn
- RAMB16
- DSP48

Values

- NAME: A unique name for the U_SET.

Syntax

Verilog Syntax

This is a Verilog attribute used in combination with the RLOC property to define the set content of a hierarchical block that will define an RPM in the synthesized netlist. Place the Verilog attribute immediately before the instantiation of a logic element.

```
(* RLOC = "X0Y0", HU_SET = "h0" *) FD sr0 (.C(clk), .D(sr_1n), .Q(sr_0));
```

Verilog Example

The following Verilog module defines RLOC and U_SET properties for the shift register flops in the module.

```
module ffs (
    input  clk,
    input  d,
    output q
);

    wire  sr_0, sr_0n;
    wire  sr_1, sr_1n;
    wire  sr_2, sr_2n;
    wire  sr_3, sr_3n;
    wire  sr_4, sr_4n;
    wire  sr_5, sr_5n;
    wire  sr_6, sr_6n;
    wire  sr_7, sr_7n;

    wire  inr, inrn, outr;

    inv i0 (sr_0, sr_0n);
    inv i1 (sr_1, sr_1n);
    inv i2 (sr_2, sr_2n);
    inv i3 (sr_3, sr_3n);
    inv i4 (sr_4, sr_4n);
    inv i5 (sr_5, sr_5n);
    inv i6 (sr_6, sr_6n);
    inv i7 (sr_7, sr_7n);
```

```

inv i8 (inr, inrn);

(* RLOC = "X0Y0", U_SET = "Uset0" *) FD sr0 (.C(clk), .D(sr_1n), .Q(sr_0));
(* RLOC = "X0Y0", U_SET = "Uset0" *) FD sr1 (.C(clk), .D(sr_2n), .Q(sr_1));
(* RLOC = "X0Y1", U_SET = "Uset0" *) FD sr2 (.C(clk), .D(sr_3n), .Q(sr_2));
(* RLOC = "X0Y1", U_SET = "Uset0" *) FD sr3 (.C(clk), .D(sr_4n), .Q(sr_3));
(* RLOC = "X0Y0", U_SET = "Uset1" *) FD sr4 (.C(clk), .D(sr_5n), .Q(sr_4));
(* RLOC = "X0Y0", U_SET = "Uset1" *) FD sr5 (.C(clk), .D(sr_6n), .Q(sr_5));
(* RLOC = "X0Y1", U_SET = "Uset1" *) FD sr6 (.C(clk), .D(sr_7n), .Q(sr_6));
(* RLOC = "X0Y1", U_SET = "Uset1" *) FD sr7 (.C(clk), .D(inrn), .Q(sr_7));
(* LOC = "SLICE_X0Y0" *) FD inq (.C(clk), .D(d), .Q(inr));
FD outq (.C(clk), .D(sr_0n), .Q(outr));

assign q = outr;

endmodule // ffs

```

Unlike the HU_SET property, which applies to the level of hierarchy it is defined in, the U_SET property transcends hierarchy. In this case, the following top-level module defines three instances of the ffs module, but results in only two U_SETS being created: Uset_0 and Uset_1, which contain Flops from all three ffs module instances defined below:

```

module top (
    input  clk,
    input  d,
    output q
);

    wire  c1, c2;

    ffs u0 (clk, d, c1);
    ffs u1 (clk, c1, c2);
    ffs u2 (clk, c2, q);

endmodule // top

```

VHDL Syntax

Declare the VHDL attribute as follows:

```
attribute U_SET : string;
```

Specify the VHDL constraint as follows:

```
attribute U_SET of {component_name | entity_name | label_name} :
{component|entity|label} is "NAME";
```

Where:

- {component_name | entity_name | label_name} is a choice of one design element.
- {component|entity|label} is the instance ID of the design element.
- "NAME" is the unique set name to give to the U_SET.

XDC Syntax

The U_SET property can not be defined using XDC constraints. The U_SET property, when present on logic elements with the RLOC property, defines relatively placed macros (RPMs), and results in the read-only RPM property in the netlist of synthesized designs.



TIP: You can use the `create_macro` and `update_macro` commands to define macro objects in the Vivado Design Suite, that act like RPMs within the design. Refer to the Vivado Design Suite Tcl Command Reference (UG835) [Ref 4] for more information on these commands.

Affected Steps

- Design Floorplanning
- place_design
- synth_design

See Also

[KEEP_HIERARCHY](#), page 93

[H_SET](#) and [HU_SET](#), page 65

[RLOC](#), page 129

USED_IN

The USED_IN property is assigned to design files (.v, .vhd, .xdc, .tcl) in the Vivado Design Suite to indicate what stage in the FPGA design flow the files are used.

For example, you could use the USED_IN property to specify an XDC file for use by the Vivado synthesis tool, but not for use in implementation. You could also specify HDL source files (.v or .vhd) as USED_IN simulation, but not for use in synthesis.



TIP: The `USED_IN_SYNTHESIS`, `USED_IN_SIMULATION`, and `USED_IN_IMPLEMENTATION` properties are related to the `USED_IN` property, and are automatically converted by the tool to `USED_IN` (`{synthesis, simulation, implementation}`) as appropriate.

You can also use the more granular values to specify an unmanaged Tcl file to be USED_IN `opt_design` or `place_design`, rather than simply used in implementation.

Architecture Support

All architectures

Applicable Objects

- Files

Values

- synthesis
- implementation
- simulation
- out_of_context
- opt_design
- power_opt_design
- place_design
- phys_opt_design
- route_design
- write_bitstream
- post_write_bitstream
- synth_blackbox_stub

- testbench
- board
- single_language

Syntax

Verilog Syntax

Not applicable.

VHDL Syntax

Not applicable.

XDC Syntax

```
set_property USED_IN {<value>} [get_files <files>]
```

Where

- *<value>* specifies one or more of the valid USED_IN values.
- *<files>* is the name or names of the files to set the USED_IN property.

XDC Syntax Example

```
# Designates the specified files as used in simulation  
set_property USED_IN {synthesis simulation} [get_files *.vhdl]
```

Affected Steps

- Synthesis
- Simulation
- Implementation
- Bitstream generation

VCCAUX_IO

VCCAUX_IO specifies the operating voltage of the VCCAUX_IO rail for a given I/O.

DRCs are available to ensure that VCCAUX_IO property assignments are correct:

- VCCAUXIOBT (warning): ensures that ports with VCCAUX_IO values of NORMAL or HIGH are only placed in HP banks.
- VCCAUXIOSTD (warning): ensures that ports with VCCAUX_IO values of NORMAL or HIGH do not use IOSTANDARDS that are only supported in HR banks.
- VCCAUXIO (error): ensures that ports with VCCAUX_IO values of NORMAL are not constrained/placed in the same bank as a port with a VCCAUX_IO value of HIGH.

Architecture Support

All architectures on High Performance (HP) bank I/O only.

Applicable Objects

- Ports (`get_ports`)
- Cells (`get_cells`)
 - I/O buffers

Values

- DONTCARE (default)
- NORMAL
- HIGH

Syntax

Verilog Syntax

To set this attribute, place the proper Verilog attribute syntax before the top-level output port declaration.

```
(* VCCAUXIO = "{DONTCARE|NORMAL|HIGH}" *)
```

Verilog Syntax Example

```
// Specifies a "HIGH" voltage for the VCCAUX_IO rail connected to this I/O  
(* VCCAUX_IO = "HIGH" *) input ACT3,
```

VHDL Syntax

To set this attribute, place the proper VHDL attribute syntax before the top-level output port declaration.

Declare the VHDL attribute as follows:

```
attribute VCCAUX_IO : string;
```

Specify the VHDL attribute as follows:

```
attribute VCCAUX_IO of port_name : signal is value;
```

Where

- `port_name` is a top-level port.

VHDL Syntax Example

```
ACT3 : in std_logic;  
attribute VCCAUX_IO : string;  
-- Specifies a "HIGH" voltage for the VCCAUX_IO rail connected to this I/O  
attribute VCCAUX_IO of ACT3 : signal is "HIGH";
```

XDC Syntax

```
set_property VCCAUX_IO value [get_ports port_name]
```

Where

- `port_name` is a top-level port.

XDC Syntax Example

```
# Specifies a "HIGH" voltage for the VCCAUX_IO rail connected to this I/O  
set_property VCCAUX_IO HIGH [get_ports ACT3]
```

Affected Steps

- I/O Planning
- place_design
- Report Power

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx® Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

Solution Centers

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

The following documents provide supplemental material to this guide:

1. *7 Series FPGA Configuration User Guide* ([UG470](#))
2. *7 Series FPGAs SelectIO Resources User Guide* ([UG471](#))
3. *7 Series FPGAs Clocking Resources User Guide* ([UG472](#))
4. *Vivado Design Suite Tcl Command Reference Guide* ([UG835](#))
5. *Vivado Design Suite User Guide: Synthesis* ([UG901](#))
6. *Vivado Design Suite User Guide: Using Constraints* ([UG903](#))
7. *Vivado Design Suite User Guide: Implementation* ([UG904](#))
8. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* ([UG906](#))



9. *Vivado Design Suite 7 Series FPGA Libraries Guide* ([UG953](#))
10. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* ([UG995](#))
11. *Vivado Design Suite Video Tutorials*
<http://www.xilinx.com/training/vivado/index.htm>
12. *Vivado Design Suite Documentation*
www.xilinx.com/support/documentation/dt_vivado2013-3.htm