

# Constraints Guide

UG625 (v. 13.4) January 18, 2012

This document applies to the following software versions: ISE Design Suite 13.4 through

For information relating to ISE Design Suite timing constraints, see the Timing Closure User Guide (UG612).



Xilinx is disclosing this user guide, manual, release note, and/or specification (the “Documentation”) to you solely for use in the development of designs to operate with Xilinx hardware devices. You may not reproduce, distribute, republish, download, display, post, or transmit the Documentation in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Xilinx expressly disclaims any liability arising out of your use of the Documentation. Xilinx reserves the right, at its sole discretion, to change the Documentation without notice at any time. Xilinx assumes no obligation to correct any errors contained in the Documentation, or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information.

THE DOCUMENTATION IS DISCLOSED TO YOU “AS-IS” WITH NO WARRANTY OF ANY KIND. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DOCUMENTATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS. IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOSS OF DATA OR LOST PROFITS, ARISING FROM YOUR USE OF THE DOCUMENTATION.

© Copyright 2002-2012 Xilinx Inc. All Rights Reserved. XILINX, the Xilinx logo, the Brand Window and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners. The PowerPC name and logo are registered trademarks of IBM Corp., and used under license. All other trademarks are the property of their respective owners.

## Revision History

Date	Version	Revision
03/01/2011	13.1	Added VCCAUX_IO and MARK_DEBUG constraints.
06/22/2011	13.2	Added Spartan®-6 to list of supported devices where appropriate.  For PULLUP (Pullup) constraint, added information that NGDBuild ignores the following: • DEFAULT KEEPER = FALSE • DEFAULT PULLUP = FALSE • DEFAULT PULLDOWN = FALSE  For IODELAY_GROUP (IODELAY Group) constraint, added information under Limitations with LOC and Architecture Support  For Area Group (AREA_GROUP) constraint, added Note: All components can be constrained by the CLOCKREGION range except IOB and BUF.  For CONFIG_MODE (Configuration Mode) constraint, added new architecture support and new values.  For BEL (BEL) constraint, removed VHDL example.
10/19/2011	13.3	Removed sentence “The RISING and FALLING keywords may also be used with TNM.”  Added DIFF_TERM support for Virtex®-6 devices.  Changed default units for both INPUT_JITTER and SYSTEM_JITTER constraints from ps to ns.  Added information that OFFSET Constraints do not allow predefined groups.  Updated POST_CRC INIT Flag for Spartan-6 devices.  Added new Vcco Sense Mode (VCCOSENSEMODE) constraint.

Date	Version	Revision
01/18/2012	13.4	<ul style="list-style-type: none"> <li>Moved timing constraints material to the <i>Timing Closure User Guide (UG612)</i>.</li> <li>Removed sentence “TNM_NET is a property normally used in conjunction with an HDL design to tag a specific net.”</li> <li>Removed reference to FSM Style (FSM_STYLE) constraint. It is covered in <i>XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (UG687)</i>.</li> <li>Corrected syntax for Timing Group (TIMEGRP) constraint under <i>Pattern Matching UCF Syntax Example Two</i>.</li> </ul>

# Table of Contents

---

Revision History .....	2
<b>Chapter 1 Constraint Types.....</b>	<b>9</b>
Attributes and Constraints .....	9
CPLD Fitter.....	11
Logical Constraints .....	12
Physical Constraints.....	13
Mapping Directives .....	14
Placement Constraints .....	15
Routing Directives .....	17
Synthesis Constraints .....	18
Timing Constraints .....	19
Configuration Constraints .....	20
<b>Chapter 2 Entry Strategies for Xilinx Constraints .....</b>	<b>21</b>
Constraints Entry Methods.....	21
Constraints Entry Table .....	21
Schematic Design.....	24
VHDL Attributes.....	24
Verilog Attributes.....	25
User Constraints File (UCF).....	27
UCF and NCF File Syntax.....	28
Physical Constraints File (PCF) .....	31
Netlist Constraints File (NCF) .....	33
Constraints Editor .....	33
ISE Design Suite .....	35
PlanAhead .....	35
Setting Constraints in PACE.....	39
Partial Design Pin Preassignment.....	39
FPGA Editor .....	41
XST Constraint File (XCF) .....	43
Constraint Priority .....	43
<b>Chapter 3 Xilinx Constraints .....</b>	<b>45</b>
Constraint Information.....	45
Area Group .....	46
Asynchronous Register.....	55

BEL .....	57
Block Name .....	60
BUFG .....	62
Clock Dedicated Route .....	65
Collapse .....	67
Component Group .....	69
Configuration Mode.....	70
CoolCLOCK .....	73
Data Gate .....	75
DCI Cascade .....	77
DCI Value .....	79
Default.....	80
Diff Term .....	83
Directed Routing.....	85
Disable.....	87
Drive .....	89
Enable .....	92
Enable Suspend.....	94
Fast .....	95
Feedback.....	97
File.....	99
Float .....	101
From Thru To.....	103
From To .....	105
FSM Style .....	108
Hierarchical Block Name .....	109
HIODELAY Group .....	111
Hierarchical Lookup Table Name .....	112
H Set.....	114
HU Set.....	115
Input Buffer Delay Value .....	117
IFD Delay Value .....	119
In Term.....	121
Input Registers .....	123
Internal Vref Bank.....	124
IOB .....	125
Input Output Block Delay .....	128
IODELAY Group .....	130

Input Output Standard .....	132
Keep .....	135
Keep Hierarchy .....	137
Keeper.....	140
Location (LOC) .....	142
Locate .....	145
Lock Pins .....	158
Lookup Table Name.....	159
Map .....	162
Mark Debug .....	163
Max Fanout .....	165
Maximum Delay.....	168
Maximum Product Terms .....	170
Maximum Skew .....	171
MCB Performance .....	173
MIODELAY Group .....	175
No Delay .....	176
No Reduce .....	178
Offset In.....	180
Offset Out.....	184
Open Drain.....	188
Out Term.....	190
Period.....	192
Pin .....	199
Post CRC.....	200
Post CRC Action.....	201
Post CRC Frequency .....	203
Post CRC INIT Flag.....	204
Post CRC Signal .....	206
Post CRC Source.....	207
Priority .....	208
Prohibit .....	209
Pulldown .....	213
Pullup .....	215
Power Mode.....	217
Registers .....	219
Relative Location (RLOC).....	221
Relative Location Origin.....	239

Relative Location Range .....	242
Save Net Flag.....	245
Schmitt Trigger.....	247
SIM Collision Check.....	249
Slew .....	251
Slow .....	254
Stepping.....	256
Suspend .....	257
System Jitter .....	259
Temperature.....	261
Timing Ignore.....	263
Timing Group.....	266
Timing Specifications .....	272
Timing Name.....	275
Timing Name Net.....	281
Timing Point Synchronization .....	285
Timing Thru Points.....	288
Timing Specification Identifier .....	292
U Set.....	297
Use Internal VREF.....	299
Use LUTNM .....	301
Use Relative Location .....	303
Use Low Skew Lines.....	306
VCCAUX.....	308
VCCAUX_IO .....	309
Voltage .....	311
Vcco Sense Mode (VCCOSENSEMODE).....	313
VREF .....	314
Wire And .....	316
XBLKNM .....	317
<b>Appendix Additional Resources .....</b>	<b>319</b>





## Constraint Types

---

This chapter discusses the constraint types documented in this Guide.

**Note** For detailed information about using timing constraints to achieve timing closure, see the *Timing Closure User Guide (UG612)*.

### Attributes and Constraints

Some designers use the terms *attribute* and *constraint* interchangeably. Other designers give them different meanings. In addition, certain language constructs use the terms *attribute* and *directive* in similar, but not identical, senses. Xilinx® uses the terms *attributes* and *constraints* as defined below.

#### Attributes

An attribute is a property associated with a device architecture primitive component that generally affects an instantiated component functionality or implementation.

Attributes are passed by means:

- Generic maps (VHDL)
- Defparams or inline parameter passed while instantiating the primitive component (Verilog)

All attributes are described in the Xilinx *Libraries Guides* as a part of the primitive component description.

#### Attributes Examples

- INIT on a LUT4 component
- CLKFX\_DIVIDE on a DCM

### Implementation Constraints

The *Constraints Guide* documents implementation constraints.

An implementation constraint is an instruction given to the FPGA implementation tools to direct the mapping, placement, timing or other guidelines to follow while processing an FPGA design.

Implementation constraints are generally placed in the User Constraints File (UCF). They may also be placed in:

- The Hardware Description Language (HDL) code
- A synthesis constraints file.

## Implementation Constraints Examples

- [Location \(LOC\)](#) (placement)
- [PERIOD](#) (timing)

## CPLD Fitter

The following constraints apply to CPLD devices:

- BUFG (CPLD)
- Collapse (COLLAPSE)
- CoolCLOCK (COOL\_CLK)
- Data Gate (DATA\_GATE)
- Fast (FAST)
- Input Registers (INREG)
- Input Output Standard (IOSTANDARD)
- Keep (KEEP)
- Keeper (KEEPER)
- Location (LOC)
- Maximum Product Terms (MAXPT)
- No Reduce (NOREDUCE)
- Offset In (OFFSET IN)
- Offset Out (OFFSET OUT)
- Open Drain (OPEN\_DRAIN)
- Period (PERIOD)
- Prohibit (PROHIBIT)
- Pullup (PULLUP)
- Power Mode (PWR\_MODE)
- Registers (REG)
- Schmitt Trigger (SCHMITT\_TRIGGER)
- Slow (SLOW)
- Timing Group (TIMEGRP)
- Timing Specifications (TIMESPEC)
- Timing Name (TNM)
- Timing Specification Identifier (TSidentifier)
- VREF
- Wire And (WIREAND)

## Logical Constraints

Logical constraints are constraints that are attached to elements before mapping or fitting.

- Logical constraints help adapt design performance to expected worst-case conditions.
- Logical constraints are converted into physical constraints when you:
  1. Choose a specific Xilinx® architecture, and
  2. Place and Route, or fit, the design.
- You can attach logical constraints using attributes in the input design, which are written into the Netlist Constraints File (NCF) or NGC netlist, or with a User Constraints File (UCF).
- Three categories of logical constraints are:
  - [Placement Constraints](#)
  - [Relative Location Constraints](#)

For FPGA devices, Relative Location constraints:

- ◆ Group logic elements into discrete sets.
- ◆ Allow you to define the location of any element within the set relative to other elements in the set, regardless of eventual placement in the overall design.
- [Timing Constraints](#)

Timing constraints allow you to specify the maximum allowable delay or skew on any given set of paths or nets.

## Physical Constraints

**Note** This section applies to FPGA devices only.

Physical constraints are constraints attached to the elements in the physical design.

### Mapping

- The physical design is the design after it has been mapped.
- When a design is mapped, the *logical* constraints in (1) the netlist, and (2) the User Constraints File (UCF), are translated into *physical* constraints that apply to a specific architecture.
- Physical constraints are defined in the Physical Constraints File (PCF) created during mapping.

### Physical Constraints File (PCF)

The Physical Constraints File (PCF):

- Is a mapper-generated file.
- Contains two sections:
  - Schematic  
Contains the physical constraints based on the logical constraints found in the netlist and the UCF.
  - User
    - ◆ Can be used to add any physical constraints.
    - ◆ Xilinx® recommends that you place user-generated constraints in a UCF, not in an NCF or a PCF.

## Mapping Directives

Mapping directives instruct the mapper to perform specific operations.

### Mapping Directives

- [Area Group](#)
- [BEL](#)
- [Block Name](#)
- [DCI Value](#)
- [Drive](#)
- [Fast](#)
- [Hierarchical Block Name](#)
- [Hierarchical Lookup Table Name](#)
- [HU Set](#)
- [IOB](#)
- [Input Output Block Delay](#)
- [Input Output Standard](#)
- [Keep](#)
- [Keeper](#)
- [Lookup Table Name](#)
- [Map](#)
- [No Delay](#)
- [Pulldown](#)
- [Pullup](#)
- [Relative Location](#)
- [Relative Location Origin](#)
- [Relative Location Range](#)
- [Save Net Flag](#)
- [Slew](#)
- [U Set](#)
- [Use Relative Location](#)
- [XBLKNM](#)

## Placement Constraints

This section describes the placement constraints for each type of logic element in FPGA designs, including:

- Flip-Flop
- ROM
- RAM
- BUFT
- CLB
- IOB
- I/O
- Edge decoder
- Global buffer

Individual logic gates such as **AND** or **OR** gates:

- Are mapped into CLB function generators before the constraints are read.
- Cannot be constrained.

## Specifying Constraints

Most constraints can be specified in:

- HDL source code, or
- User Constraints File (UCF)

In a constraints file, each placement constraint acts upon one or more symbols. Every symbol in a design carries a unique name, which is defined in the input file. Use this name in a constraint statement to identify the symbol.

## Case Sensitivity

- The UCF and the NCF are case sensitive.
- Identifier names (names of objects, such as net names) must exactly match the case of the name as it exists in the source design netlist.
- Xilinx® keywords (such as [LOC](#), [PROHIBIT](#), [RLOC](#), and [BLKNM](#)) can be entered in all uppercase or all lowercase. Mixed case is not allowed.

## Netlist Mapping and Placement Constraints

The following constraints control mapping and placement of symbols in a netlist:

- [BLKNM](#)
- [HBLKNM](#)
- [HLUTNM](#)
- [LOC](#)
- [LUTNM](#)
- [PROHIBIT](#)
- [RLOC](#)
- [RLOC\\_ORIGIN](#)
- [RLOC\\_RANGE](#)
- [XBLKNM](#)

## Relative Location (RLOC) Constraints

The **RLOC** constraint groups logic elements into discrete sets.

- You can define the location of any element within the set relative to other elements in the set, regardless of eventual placement in the overall design.
- For example, if **RLOC** constraints are applied to a group of eight flip-flops organized in a column, the mapper maintains the columnar order and moves the entire group of flip-flops as a single unit.
- In contrast, absolute **LOC** constraints constrain design elements to specific locations on the FPGA die with no relation to other design elements.

## Placement Constraints

- **AREA\_GROUP**
- **BEL**
- **LOC**
- **LOCATE**
- **Prohibit**
- **RLOC**
- **RLOC\_ORIGIN**
- **RLOC\_RANGE**
- **USE\_RLOC**



## Routing Directives

Routing directives instruct PAR to perform specific operations.

- [AREA\\_GROUP](#)
- [CONFIG\\_MODE](#)
- [LOCK\\_PINS](#)

## Synthesis Constraints

Synthesis constraints direct the synthesis tool optimization technique for a particular design or piece of Hardware Description Language (HDL) code. The constraints are either embedded in the source code, or are included in a separate synthesis constraints file.

The following constraints are synthesis constraints:

- [FROM-TO](#)
- [IOB](#)
- [KEEP](#)
- [MAP](#)
- [MARK\\_DEBUG](#)
- [OFFSET IN](#)
- [OFFSET OUT](#)
- [PERIOD](#)
- [TIG](#)
- [TNM](#)
- [TNM\\_NET](#)

## Synthesis Constraint Documentation

XST synthesis constraints are documented in:

- *XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices (UG627)*
- *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (UG687)*

Other synthesis constraints are documented in the software vendor's documentation.

## Timing Constraints

The Xilinx® software enables you to specify precise timing requirements using either global or path-specific timing constraints.

The recommended methods for defining the constraints are discussed in the *Timing Closure User Guide (UG612)*.

The following are timing constraints and associated grouping constraints:

- [Asynchronous Register](#)
- [Disable](#)
- [Enable](#)
- [From Thru To](#)
- [From To](#)
- [Maximum Skew](#)
- [Offset In](#)
- [Offset Out](#)
- [Period](#)
- [Priority](#)
- [System Jitter](#)
- [Temperature](#)
- [Timing Ignore](#)
- [Timing Group](#)
- [Timing Specifications](#)
- [Timing Name](#)
- [Timing Name Net](#)
- [Timing Point Synchronization](#)
- [Timing Thru Points](#)
- [Timing Specification Identifier](#)
- [Voltage](#)

## Configuration Constraints

- [Configuration Mode](#)
- [DCI Cascade](#)
- [MCB Performance](#)
- [Stepping](#)
- [Post CRC](#)
- [Post CRC Action](#)
- [Post CRC Frequency](#)
- [Post CRC INIT Flag](#)
- [VCCAUX](#)
- [VREF](#)
- [Internal Vref Bank](#)

## Entry Strategies for Xilinx Constraints

This chapter discusses entry strategies for Xilinx® constraints, including how to use ISE® Design Suite to enter a given constraint type.

### Constraints Entry Methods

The following table shows which feature of ISE® Design Suite to use to enter a given constraint type.

**Constraints Entry Methods**

Constraint Type	Tool	Devices
Timing	Constraints Editor	All CPLD and FPGA device families
IO placement and area-group constraints	PlanAhead™ Software	All FPGA device families
IO placement	PACE	All CPLD device families
IO placement and other placement constraints	Schematic and Symbol Editors	All CPLD and FPGA device families

### Constraints Entry Table

The following table lists the constraints and their associated entry strategies. See the individual constraint for syntax examples.

**Constraints Entry Table**

Constraint	Schematic	VHDL Verilog	NCF	UCF	Constraints Editor	PCF	XCF	Plan-Ahead	PACE	FPGA Editor	ISE® Design Suite
<a href="#">AREA_GROUP</a>	Yes		Yes	Yes	Yes			Yes			
<a href="#">ASYNC_REG</a>		Yes	Yes	Yes	Yes						
<a href="#">BEL</a>		Yes	Yes	Yes				Yes			
<a href="#">BLKNM</a>	Yes	Yes	Yes	Yes			Yes				
<a href="#">BUFG (CPLD)</a>	Yes	Yes	Yes	Yes			Yes				
<a href="#">CLOCK_DEDICATED_ROUTE</a>			Yes	Yes							
<a href="#">COLLAPSE</a>	Yes	Yes	Yes	Yes							
<a href="#">COMPGRP</a>						Yes					

Constraint	Schematic	VHDL Verilog	NCF	UCF	Constraints Editor	PCF	XCF	Plan-Ahead	PACE	FPGA Editor	ISE® Design Suite
CONFIG_MODE				Yes							
COOL_CLK	Yes	Yes	Yes	Yes							
DATA_GATE	Yes	Yes	Yes	Yes							
DEFAULT	Yes	Yes	Yes	Yes			Yes	Yes	Yes		
DCI_CASCADE			Yes	Yes		Yes					
DCI_VALUE			Yes	Yes							
DIRECTED_ROUTING			Yes	Yes						Yes	
DISABLE			Yes	Yes		Yes					
DRIVE	Yes	Yes	Yes	Yes			Yes	Yes	Yes	Yes	
ENABLE			Yes	Yes		Yes					
ENABLE_SUSPEND			Yes	Yes							
FAST	Yes	Yes	Yes	Yes			Yes	Yes	Yes		
FEEDBACK				Yes	Yes	Yes	Yes	Yes			
FILE	Yes	Yes									
FLOAT	Yes	Yes	Yes	Yes			Yes				
FROM-THRU-TO			Yes	Yes	Yes	Yes		Yes			
FROM-TO			Yes	Yes	Yes	Yes	Yes	Yes			
HBLKNM	Yes	Yes	Yes	Yes							
HLUTNM	Yes	Yes	Yes	Yes	Yes		Yes				
HU_SET	Yes	Yes	Yes	Yes			Yes				
IBUF_DELAY_VALUE	Yes	Yes	Yes	Yes							
IFD_DELAY_VALUE	Yes	Yes	Yes	Yes							
INREG	Yes			Yes							
IOB	Yes	Yes	Yes	Yes			Yes				Yes
IOBDelay	Yes	Yes	Yes	Yes	Yes						
IODELAY_GROUP				Yes							
IOSTANDARD	Yes	Yes	Yes	Yes			Yes	Yes	Yes	Yes	
KEEP	Yes	Yes	Yes	Yes			Yes				
KEEPER	Yes	Yes	Yes	Yes	Yes		Yes			Yes	
KEEP_HIERARCHY	Yes	Yes	Yes	Yes			Yes				Yes
LOC	Yes	Yes	Yes	Yes		Yes	Yes	Yes	Yes		
LOCATE						Yes				Yes	
LOCK_PINS		Yes	Yes	Yes							
LUTNM	Yes	Yes	Yes	Yes							

Constraint	Schematic	VHDL Verilog	NCF	UCF	Constraints Editor	PCF	XCF	Plan-Ahead	PACE	FPGA Editor	ISE® Design Suite
MAP	Yes		Yes	Yes							
MARK_DEBUG		Yes					Yes	Yes			
MAXDELAY	Yes	Yes	Yes	Yes	Yes	Yes				Yes	
MAX_FANOUT		Yes					Yes				Yes
MAXPT		Yes	Yes	Yes							
MAXSKEW	Yes	Yes	Yes	Yes	Yes	Yes				Yes	
NODELAY	Yes	Yes	Yes	Yes			Yes				
IODELAY_GROUP				Yes							
NOREDUCE	Yes	Yes	Yes	Yes			Yes				
OFFSET IN	Yes		Yes	Yes	Yes	Yes	Yes	Yes			
OFFSET OUT	Yes		Yes	Yes	Yes	Yes	Yes	Yes			
OPEN_DRAIN	Yes	Yes	Yes	Yes			Yes				
PERIOD	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes	
PIN				Yes							
POST_CRC				Yes		Yes					
POST_CRC_ACTION				Yes		Yes					
POST_CRC_FREQ				Yes		Yes					
POST_CRC_INIT_FLAG				Yes		Yes					
PRIORITY			Yes	Yes		Yes					
PROHIBIT				Yes		Yes		Yes	Yes	Yes	
PULLDOWN	Yes	Yes	Yes	Yes			Yes	Yes	Yes	Yes	
PULLUP	Yes	Yes	Yes	Yes			Yes	Yes	Yes	Yes	
PWR_MODE	Yes	Yes	Yes	Yes			Yes				
REG	Yes	Yes	Yes	Yes			Yes				
RLOC	Yes	Yes	Yes	Yes			Yes	Yes			
RLOC_ORIGIN	Yes	Yes	Yes	Yes		Yes		Yes			
RLOC_RANGE	Yes	Yes	Yes	Yes		Yes	Yes				
SAVE NET FLAG	Yes	Yes	Yes	Yes			Yes				
SCHMITT_TRIGGER	Yes	Yes	Yes	Yes			Yes				
SLEW	Yes	Yes	Yes	Yes			Yes	Yes	Yes	Yes	
SLOW	Yes	Yes	Yes	Yes			Yes	Yes	Yes	Yes	
STEPPING				Yes							
SUSPEND	Yes	Yes	Yes	Yes					Yes		
SYSTEM_JITTER	Yes	Yes	Yes	Yes			Yes				
TEMPERATURE			Yes	Yes	Yes	Yes					
TIG	Yes		Yes	Yes	Yes	Yes	Yes	Yes			
TIMEGRP			Yes	Yes	Yes	Yes	Yes	Yes			

Constraint	Schematic	VHDL Verilog	NCF	UCF	Constraints Editor	PCF	XCF	Plan-Ahead	PACE	FPGA Editor	ISE® Design Suite
<a href="#">TIMESPEC</a>			Yes	Yes	Yes		Yes	Yes			
<a href="#">TNM</a>			Yes	Yes	Yes		Yes	Yes			
<a href="#">TNM_NET</a>	Yes		Yes	Yes	Yes		Yes	Yes			
<a href="#">TPSYNC</a>	Yes		Yes	Yes							
<a href="#">TPTHRU</a>	Yes		Yes	Yes	Yes						
<a href="#">TSidentifier</a>			Yes	Yes	Yes	Yes	Yes			Yes	
<a href="#">U_SET</a>	Yes	Yes	Yes	Yes			Yes				
<a href="#">USE_RLOC</a>	Yes	Yes	Yes	Yes			Yes	Yes			
<a href="#">USE_INTERNAL_VREF</a>		Yes	Yes	Yes			Yes				
<a href="#">VCCAUX</a>			Yes	Yes							
<a href="#">VCCAUX_IO</a>	Yes	Yes	Yes	Yes							
<a href="#">VCCOSENSEMODE</a>			Yes	Yes							
<a href="#">VOLTAGE</a>			Yes	Yes	Yes	Yes					
<a href="#">VREF</a>	Yes		Yes	Yes							
<a href="#">WIREAND</a>	Yes	Yes	Yes	Yes							
<a href="#">XBLKNM</a>	Yes	Yes	Yes	Yes			Yes				

## Schematic Design

Follow these rules to add Xilinx® constraints as attributes within a symbol or schematic drawing:

- If a constraint applies to a net, add it as an attribute to the net.
- If a constraint applies to an instance, add it as an attribute to the instance.
- You cannot add global constraints such as PART and [Prohibit](#).
- You cannot add any timing specifications that would be attached to a [Timing Specifications](#) or [Timing Group](#).
- Enter attribute names and values in either all uppercase or all lowercase. Mixed uppercase and lowercase is not allowed.

For more information about creating, modifying, and displaying attributes, see the Schematic and Symbol Editors Help.

The syntax for any constraint that can be entered in a schematic is described in the section for that constraint. For an example of correct schematic syntax, see the Schematic Syntax Example in [BEL](#).

## VHDL Attributes

In VHDL code, constraints can be specified with VHDL attributes. Before it can be used, a constraint must be declared with the following syntax:

```
attribute attribute_name : string;
```

Example

```
attribute RLOC : string;
```



An attribute can be declared in an entity or architecture.

- If the attribute is declared in the entity, it is visible both in the entity and the architecture body.
- If the attribute is declared in the architecture, it cannot be used in the entity declaration.

Once the attribute is declared, you can specify a VHDL attribute as follows:

```
attribute attribute_name of {component_name | label_name | entity_name | signal_name | variable_name | type_name } :
{component | label | entity | signal | variable | type} is attribute_value ;
```

Accepted *attribute\_values* depend on the attribute type.

Example One

```
attribute RLOC : string;
```

```
attribute RLOC of u123 : label is "R11C1.S0";
```

Example Two

```
attribute bufg : string;
```

```
attribute bufg of my_clock : signal is "clk";
```

For Xilinx® the most common objects are **signal**, **entity**, and **label**. A label describes an instance of a component.

**Note** The signal attribute must be used on the output port.

VHDL is case insensitive.

In some cases, existing Xilinx constraints cannot be used in attributes, since they are also VHDL keywords. To avoid this naming conflict, use a constraint alias. Each Xilinx constraint has its own alias. The alias is the original constraint name pre-pended with the prefix **XIL\_**. For example, the **RANGE** constraint cannot be used in an attribute directly. Use **XIL\_RANGE** instead.

## Verilog Attributes

Verilog attributes are bounded by asterisks (\*), and use the following syntax:

```
(* attribute_name = attribute_value *)
```

where

- *attribute* precedes the signal, module, or instance declaration to which it refers.
- *attribute\_value* is a string. No integer or scalar values are allowed.
- *attribute\_value* is between quotes.
- The default is 1. (\* **attribute\_name** \*) is the same as (\* **attribute\_name** = "1" \*).

### Verilog Attributes Syntax Example One

```
(* clock_buffer = "IBUFG" *) input CLK;
```

### Verilog Attributes Syntax Example Two

```
(* INIT = "0000" *) reg [3:0] d_out;
```

### Verilog Attributes Syntax Example Three

```
always@(current_state or reset)
  begin (* parallel_case *) (* full_case *)
    case (current_state)
```

### Verilog Attributes Syntax Example Four

```
(* mult_style = "pipe_lut" *) MULT my_mult (a, b, c);
```

## Verilog Limitations

Verilog attributes are not supported for:

- Signal declarations
- Statements
- Port connections
- Expression operators

## Verilog Meta Comments

Constraints can also be specified in Verilog code using meta comments. The Verilog format is the preferred syntax, but the meta comment style is still supported. Use the following syntax:

```
// synthesis attribute AttributeName [of] ObjectName [is] AttributeValue
```

### Verilog Meta Comments Examples

```
// synthesis attribute RLOC of u123 is R11C1.S0  
// synthesis attribute HU_SET u1 MY_SET  
// synthesis attribute bufg of my_clock is "clk"
```

## User Constraints File (UCF)

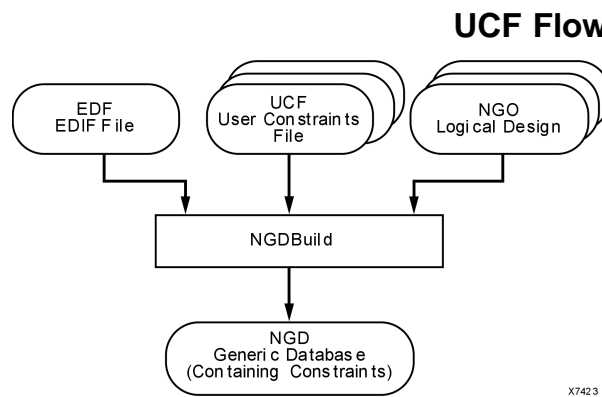
A User Constraints File (UCF) file is an ASCII file which specifies constraints on the logical design. You can create UCF files and enter constraints with:

- Any text editor
- [Constraints Editor](#)

These constraints affect how the logical design is implemented in the target device. You can use UCF files to override constraints specified during design entry.

### UCF Flow

The following figure illustrates the UCF flow.



UCF files are input to NGDBuild (see the preceding figure). The constraints in the UCF files become part of the information in the NGD file produced by NGDBuild. For FPGA devices, some of these constraints are used when the design is mapped by MAP, and some of the constraints are written into the Physical Constraints File (PCF) produced by MAP.

The constraints in the PCF file are used by each of the physical design tools (for example, PAR and the timing analysis tools), which are run after the design is mapped.

## Manual Entry of Timing Constraints

In addition to entering timing constraints through Constraints Editor, you can manually enter timing specifications as constraints in UCF files. When you run NGDBuild on the design, the timing constraints are added to the design database as part of the NGD file. You can also use the Constraints Editor to enter timing constraints in UCF files.

## Constraint Conflicts in Multiple UCF Files

The Xilinx® software still uses "last constraint wins" just as in HDL, NCF, UCF, and PCF processing. Currently, the UCF files are processed in the order in which they are added to the project (either in the ISE® Design Suite or by means of Tcl commands), and it has no bearing on timestamps or the order in which the files were modified.

## UCF and NCF File Syntax

Logical constraints are found in:

- The Netlist Constraints File (NCF), an ASCII file typically generated by synthesis programs
- The User Constraints File (UCF), an ASCII file generated by the user

Xilinx® recommends that you place user-generated constraints in the UCF or NCF files and not in the Physical Constraints File (PCF) file.

### General Rules for UCF and NCF

- UCF and NCF files are case sensitive. Identifier names (names of objects, such as net names) must exactly match the case of the name as it exists in the source design netlist. However, any Xilinx constraint keyword (for example, LOC, PERIOD, HIGH, LOW) may be entered in all uppercase, all lowercase, or mixed case.
- Each statement is terminated by a semicolon (;).
- No continuation characters are necessary if a statement exceeds one line, since a semicolon marks the end of the statement.
- Xilinx recommends that you group similar blocks, or components, as a single timing constraint, and not as separate timing constraints.
- To add comments to the UCF and NCF files, begin each comment line with a pound (#) sign, as in the following example.

```
# file TEST.UCF
# net constraints for TEST design
NET "$SIG_0 MAXDELAY" = 10;
NET "$SIG_1 MAXDELAY" = 12 ns;
```

- Statements need not be placed in any particular order in the UCF and NCF file.
- Enclose NET and INST names in double quotes (recommended but not mandatory).
- Enclose inverted signal names that contain a tilde (for example, ~OUTSIG1) in double quotes (mandatory).
- You can enter multiple constraints for a given instance. For more information, see Entering Multiple Constraints below.

### Conflict in Constraints

The constraints in the UCF and NCF files and the constraints in the schematic or synthesis file are applied equally. It does not matter whether a constraint is entered in the schematic, HDL, UCF or NCF files. If the constraints overlap, UCF overrides NCF and schematic/netlist constraints. NCF overrides schematic/netlist constraints.

If by mistake two or more elements are locked onto a single location, MAP detects the conflict, issues an error message, and stops processing so that you can correct the mistake.

## Syntax

The UCF file supports a basic syntax that can be expressed as:

`{NET|INST|PIN} "full_name" constraint;`

- *full\_name* is a full hierarchically qualified name of the object being referred to. When the name refers to a pin, the instance name of the element is also required.
- *constraint* is a constraint in the same form as it would be used if it were attached as an attribute on a schematic object. For example, LOC=P38 and FAST.

## Specifying Attributes for TIMEGRP and TIMESPEC

See the *Timing Closure User Guide* (UG612).

## Entering Multiple Constraints

You can cascade multiple constraints for a given instance in the UCF file:

```
INST instanceName constraintName = constraintValue | constraintName = constraintValue;
```

For example:

```
INST myInst LOC = P53 | IOSTANDARD = LVPECL33 | SLEW = FAST;
```

## File Name

By default, NGDBuild reads the constraints file that carries the same name as the input design with a .ucf extension. However, you can specify a different constraints file name with the **-uc** option when running NGDBuild. NGDBuild automatically reads in the NCF file if it has the same base name as the input EDIF file and is in the same directory as the EDIF file.

The implementation tools (for example, NGDBuild, MAP, and PAR) require file name extensions in all lowercase (for example, .ucf) in command lines.

## Instances and Blocks

The statements in the constraints file concern instances and blocks, which are defined as follows.

- An *instance* is a symbol on the schematic.
- An *instance name* is the symbol name as it appears in the EDIF netlist.
- A *block* is a CLB or an IOB.
- Specify the *block name* with the BLKNM, HBLKNM, or XBLKNM attributes. By default, the software assigns a block name on the basis of a signal name associated with the block.

## Physical Constraints File (PCF)

The Native Generic Database (NGD) file produced when a design netlist is read into the Xilinx® ISE® Design Suite may contain a number of logical constraints. These constraints originate in any of these sources.

- An attribute assigned within a schematic or Hardware Description Language (HDL) file
- A constraint entered in a User Constraints File (UCF)
- A constraint appearing in an Netlist Constraints File (NCF) produced by a CAE vendor toolset

Logical constraints in the NGD file are read by MAP. MAP uses some of the constraints to map the design and converts logical constraints to physical constraints. MAP then writes these physical constraints into a Physical Constraints File (PCF).

The PCF file is an ASCII file containing two separate sections:

- A section for those physical constraints created by the mapper
- A section for physical constraints entered by the user

The mapper section is rewritten every time you run the mapper.

Mapper-generated physical constraints appear first in the file, followed by user physical constraints. In the event of conflicts between mapper-generated and user constraints, user constraints are read last, and override mapper-generated constraints.

The beginning of the mapper-generated section is indicated by **SCHEMATIC START**. The end of this section is indicated by **SCHEMATIC END**. Enter user-generated constraints, such as timing constraints, after **SCHEMATIC END**.

You can write user constraints directly into the file, or you can write them indirectly (or undo them) from within the FPGA Editor. For more information on constraints in the FPGA Editor, see the FPGA Editor help.

**Note** Whenever possible, you should add design constraints to the HDL, schematic, or UCF, instead of PCF. This simplifies design archiving and improves design rule checking.

The PCF file is an optional input to PAR, FPGA Editor, TRACE, NetGen, and BitGen. The file may contain any number of constraints, and any number of comments, in any order. A comment consists of either a pound sign (#) or double slashes (//) followed by any number of other characters up to a new line. Each comment line must begin with a pound sign (#) or double slashes (//).

The structure of the PCF file is as follows.

**schematic start;**

**translated schematic and UCF and NCF constraints in PCF format**

**schematic end;**

**user-entered physical constraints**

**Caution!** Put all user-entered physical constraints after the schematic end statement. Any constraints preceding this section or within this section may be overwritten or ignored.

Do not edit the schematic constraints. They are overwritten every time the mapper generates a new PCF file.

Global constraints need not be attached to any object, but should be entered in a constraints file.

Indicate the end of each constraint statement with a semicolon.

In all of the constraints files (NCF, UCF, and PCF), instance or variable names that match internal reserved words are rejected unless the names are enclosed in double quotes. It is good practice to enclose all names in double quotes. For example, the following entry would not be accepted because the word *net* is a reserved word.

```
NET net FAST;
```

Following is the recommended way to enter the constraint.

```
NET "net" FAST;
```



## Netlist Constraints File (NCF)

The syntax rules for the Netlist Constraints File (NCF) are the same as those for the User Constraints File (UCF). For more information, see [User Constraints File \(UCF\)](#) and [Netlist Constraints File \(NCF\) File Syntax](#).

## Constraints Editor

The Constraints Editor:

- Is a graphical tool that simplifies the process of entering timing constraints.
- Guides you through the process of creating constraints without requiring you to understand User Constraints File (UCF) syntax.
- Is used in the implementation phase after the translation step (NGCBuild).
- Allows you to create and manipulate constraints without directly editing the UCF.

After you create or modify the constraints with Constraints Editor, you must run NGCBuild again. During this second run, you:

- Use the new UCF and design source netlist files as *input*.
- Generate a new NGD file as *output*.

For the constraints and devices with which Constraints Editor can be used, see [Constraints Entry Methods](#). For information on running Constraints Editor, see the ISE® Design Suite Help.

## Input/Output

Constraints Editor requires:

- A User Constraints File (UCF)
- A Native Generic Database (NGD) file

Constraints Editor uses the NGD file to provide names of logical elements for grouping. As output, it uses the UCF file.

After you start Constraints Editor, you must first open a UCF file. If the UCF and NGD root names are not the same, you must select the appropriate NGD file to open. For more information, see the Constraints Editor Help.

Upon successful completion, Constraints Editor writes out a UCF file. NGCBuild (translation) uses the UCF file, along with design source netlists, to produce an NGD file. The NGD file is read by the MAP program. MAP generates a physical design database in the form of an Native Circuit Description (NCD) file and also generates a Physical Constraints File (PCF). The implementation software uses these files to ultimately produce a bitstream.

**Note** Not all Xilinx® constraints are accessible through Constraints Editor.

## Starting Constraints Editor

Constraints Editor runs on personal computers and workstations. You can start Constraints Editor:

- From ISE Design Suite
- As a standalone tool
- From the command line

## Running Constraints Editor From ISE Design Suite

In ISE Design Suite, launch Constraints Editor from the Processes window.

1. Select a design file in the Sources window.
2. Double-click **Processes > Design Utilities > User Constraints > Create Timing Constraints**.

## Running Constraints Editor As a Standalone Tool

If you installed Constraints Editor as a standalone tool, either:

- Click the Constraints Editor icon on the Windows desktop, or
- Select **Start > Programs > Xilinx ISE > Accessories > Constraints Editor**

## Running Constraints Editor From the Command Line With No Data Loaded

To start Constraints Editor from the command line with no data loaded, enter:

```
constraints_editor
```

## Running Constraints Editor From the Command Line With the NGD File Loaded

To start Constraints Editor from the command line with the NGD file loaded, enter:

```
constraints_editor ngdfile_name
```

*ngdfile\_name* is the name of the NGD file

You must use the `.ngd` extension.

If a UCF file with the same base name as the NGD file exists, it is also loaded. Otherwise, you are prompted for a UCF file.

## Running Constraints Editor From the Command Line With the NGD File and the UCF File Loaded

To start Constraints Editor from the command line with the NGD file and the UCF file loaded, enter:

```
constraints_editor ngdfile_name -uc ucf_file_name
```

- *ngdfile\_name* is the name of the NGD file
- *ucf\_file\_name* is the name of the UCF file

You must use the `.ucf` extension.

## Running Constraints Editor From the Command Line As a Background Process

To run Constraints Editor as a background process on a workstation, enter:

```
constraints_editor &
```

## ISE Design Suite

To set implementation constraints in ISE® Design Suite:

- For FPGA devices, the implementation process properties specify how a design is translated, mapped, placed, and routed. You can set multiple properties to control the implementation processes.
- For CPLD devices, the implementation process properties specify how a design is translated and fit.

For more information, see the ISE Design Suite help for the Process Properties dialog box.

## PlanAhead

You can use the PlanAhead™ software either before or after synthesis. The PlanAhead software supports the following devices:

- Virtex®-4 devices and higher
- Spartan®-3 devices and higher

The PlanAhead software allows you to drag-and-drop placement constraints, including:

- Pinout
- Logic placement
- Area

For more information, see the *PlanAhead User Guide* (UG632).

## Assigning Placement Constraints

For FPGA devices, you can use the PlanAhead software to enter placement constraints that control:

- I/O pin and logic assignments
- Global logic placement
- Area group assignment

The PlanAhead software runs automatically at various stages of the design process to allow you to analyze the design and to apply placement constraints. A simplified version of the PlanAhead software is invoked from ISE® Design Suite to enable only the types of features required to perform the selected tasks. The standalone PlanAhead software has many more features available.

When the PlanAhead software is invoked from ISE Design Suite, it is a separate CPU process and does not communicate realtime with ISE Design Suite as some other tools do. In order to prevent data mismatch or out of sync issues, do not update ISE Design Suite source files while the PlanAhead software is running .

When the PlanAhead software is invoked, the appropriate source files are passed to the PlanAhead software to populate the PlanAhead Project. When the PlanAhead Project is saved, only the modified UCF files are passed back to ISE Design Suite to update the Project. These input source files vary depending on the process step invoked.

For more information on the types of files passed, see the *Pin Assignment* and *Floorplanning and Placement Constraints* sections later in this chapter. The following sections cover strategies for entering placement constraints using the PlanAhead software.

## Defining I/O Pin Configurations

This section discusses Defining I/O Pin Configurations and includes:

- Pin Assignment Overview
- Reviewing I/O Pin Data Information
- Pin Assignment
- I/O Planner Documentation

### Pin Assignment Overview

I/O Planner can be invoked either as a standalone tool or from within ISE Design Suite. Invoking I/O Planner standalone can be helpful early in the design process when HDL sources may not yet be complete. I/O ports can be defined manually within the tool, or by importing a CSV format spreadsheet or HDL sources. You can define an initial pinout and export a User Constraints File (UCF) file for use in the ISE Design Suite flow.

A UCF file is required when invoking I/O Planner from within ISE Design Suite. If a UCF file does not exist, an empty one is created. Creation of I/O ports manually or by importing a CSV spreadsheet is not enabled when invoking I/O Planner from ISE Design Suite.

I/O Planner is an I/O pin assignment environment containing many helpful views and capabilities. You can selectively drag and drop groups of I/O ports onto the device using a variety of methods. An automatic placement routine is also available. Comprehensive Design Rule Checks (DRCs) ensure legal pinout definition.

### Reviewing I/O Pin Data Information

[Data Sheets](#) provide device specifications, including I/O standards. To get device-specific I/O standard information, see the data sheet for the device you are targeting. A lot of the data contained in the data sheets is also available inside of the I/O Planner tool. The types of information available include I/O standards, clock capable pins, internal trace delays, differential pairs, clock region and I/O bank contents, etc. Information about I/O related device resources such as global and regional clock buffers, I/O delays and delay controllers, gigabit transceivers, etc. is also available.

## Pin Assignment

To invoke I/O Planner standalone either click the PlanAhead Windows Desktop icon or enter PlanAhead on the Linux command line. From ISE Design Suite, you can use any of the following methods to start your pin assignment process, which allows you to choose the method most convenient for you:

- Floorplanning I/O – Pre-Synthesis

When using this command or process step, the HDL source files are passed to the PlanAhead software in order to extract the top level I/O port information only. If a UCF files exists in the ISE Design Suite project, it is passed to the PlanAhead software for modification. If a UCF does not exist, you are prompted to create one. If multiple UCF files exist, you are prompted to select the desired file to add new constraints to. Existing constraints are modified in whichever file they are contained in.

Refer to the I/O Planner Documentation section for information on using the I/O Planner environment contained in the PlanAhead software.

Once the I/O pin assignment is made, you save the PlanAhead software project and exit the PlanAhead software. This updates the UCF files in the ISE Design Suite project and update the project status accordingly. Exiting the PlanAhead software without saving does not change the ISE Design Suite UCF source files or status.

- Floorplanning a Design – Post-Synthesis

When using this command or process step, the synthesized netlist source files are passed to the PlanAhead software. If a UCF files exists in the ISE Design Suite project, it is passed to the PlanAhead software for modification. If a UCF does not exist, you are prompted to create one. If multiple UCF files exist, you are prompted to select the desired file to add new constraints to. Existing constraints are modified in whichever file they are contained in.

Having a synthesized netlist as input enables more functionality in I/O Planner since the tool is now aware of the clocks and clock related logic in the design. Additional I/O planning capabilities and DRCs are provided to make more intelligent pin assignment decisions. The design connectivity can also be analyzed to ensure optimized use of device resources in relation to the I/Os.

Refer to the I/O Planner Documentation section for information on using the I/O Planner environment contained in the PlanAhead software.

Once the I/O pin assignment is made, you will then save the PlanAhead software project and exit the PlanAhead software. This will update the UCF files in the Project Navigator project and update the project status accordingly. Exiting PlanAhead without saving with not change the ISE Design Suite UCF source files or status.

## I/O Planning Documentation

The *PlanAhead User Guide* (UG632) contains a section on I/O planning for analyzing the device resources and I/O pin assignment.

The *PlanAhead Software Tutorial: I/O Pin Planning* (UG674), and the *Pin Planning Methodology Guide* (UG792) are also available.

## Floorplanning and Placement Constraints

The PlanAhead software provides a comprehensive environment for analyzing the design from a number of different aspects including connectivity, density, and timing. You can then apply placement constraints to help drive the implementation tools toward better or more consistent results. These constraints may include LOC constraints to lock specific logic objects into specific sites on the device or AREA\_GROUP constraints to constrain a group of logic within a specific area of the device.

## Placement LOC Constraint Assignment

The PlanAhead software enables you to lock down any logic to specific device sites. This often includes global logic objects such as the following: BUFG, BRAM, MULT, PPC405, GT, DLL, and DCM.

You can place logic objects by simply dragging the desired logic object from any of the appropriate PlanAhead software views and drop it in the Device View in the Workspace. Some types of logic such as I/O ports enable you to enter the desired location site in the object General Properties view.

For more information about assigning placement constraints, see “Using Placement Constraints” in the “Floorplanning the Design” chapter of the *PlanAhead User Guide* (UG632).

## Area Group Assignment

Area groups are the primary means of placing logic in specific regions of the device, for example, within a particular clock region. The PlanAhead software enables you to create area groups using a wide variety of methods. Assistance with connectivity, size logic types and ranges are all provided by the tool including DRCs to ensure proper [Area Group](#) property definition.

For more information about creating area group constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632).

## Setting Constraints in PACE

For CPLD devices, you can set constraints in the Pinout and Area Constraints Editor (PACE). The Pin Assignments Editor in PACE is used to:

- Assign location constraints to I/Os.
- Assign IO properties such as IO Standards.

For a list of the constraints and devices with which PACE can be used, see [Constraints Entry Methods](#). For more information about accessing and using PACE, see the ISE® Design Suite Help.

## Partial Design Pin Preassignment

This section deals with Pin Preassignment when a design is partially completed. For information on Pin Preassignment in which a Hardware Description Language (HDL) template is built by adding constraints to pins that are defined within PlanAhead™ or PACE, see the ISE® Design Suite Help. PACE is supported for CPLD devices. PlanAhead is supported for FPGA devices.

Designs that are not yet fully coded might still have layout requirements. Pin assignments, voltage standards, banking rules, and other board requirements might be in place long before the design has reached the point where these constraints can be applied. Pin Preassignment allows the design pinout rules to be determined before the design logic has been completed.

To use Pin Preassignment in PlanAhead or PACE:

1. Provide the complete list of ports in your top-level design
2. Assign I/O constraints to them

Even if the ports are not used by any logic (that is, no loads for input pins, no sources for output pins), they can still receive constraints and be kept through implementation.

Assign [Location \(LOC\)](#) or [Input Output Standard \(IOSTANDARD\)](#) constraints in the User Constraints File (UCF) just like for any I/O pin. These requirements are annotated in the database. PlanAhead and PACE can be used to assign pin locations, banking groups or voltage standards, and DRC checks can be run. The final PAD report contains any pins that have logic or constraints associated with them.

This implementation is incomplete and cannot be downloaded to the hardware. You should expect these errors during the DRC phase of bitstream generation (BitGen):

- ERROR: PhysDesignRules:368 - The signal <D\_OBUF> is incomplete. The signal is not driven by any source pin in the design.
- ERROR: PhysDesignRules:10 - The network <D\_OBUF> is completely unrouted.

To trim any unused ports from the design, remove the associated constraints. The Translate (NGDBuild) phase trims these unused pins.

In this example, there are six top-level ports. Only three (clk, A, C) are currently used. Of the remaining three ports:

- B is kept because it has a [Location \(LOC\)](#) constraint.
- D is kept because it has an [Input Output Standard \(IOSTANDARD\)](#) constraint.
- E is trimmed because it is completely unused and unconstrained.

### Verilog Example

```
-----  
module design_top(clk, A, B, C, D, E);  
input clk, A, B;  
output reg C, D, E;  
  
always@(posedge clk)  
C <= A;  
  
endmodule
```

### UCF Example

```
-----  
  
NET "A" LOC = "E2" ;  
NET "B" LOC = "E3" ;  
NET "C" LOC = "B15" ;  
NET "D" IOSTANDARD = SSTL2_II ;
```



## FPGA Editor

You can add or delete certain constraints in the Physical Constraints File (PCF) using FPGA Editor. FPGA Editor supports net, site, and component constraints as property fields in the individual nets and components. Properties are set with the **Setattr** command, and are read with the **Getattr** command.

All Boolean constraints, including BLOCK, **Locate**, LOCK, **Offset In**, **Offset Out**, and **Prohibit**, have values of **On** or **Off**. Offset direction has a value of either **In** or **Out**. Offset order has a value of either **Before** or **After**. All other constraints have a numeric value. They can also be set to **Off** in order to delete the constraint. All values are case-insensitive (for example, **On** and **on** are both accepted).

When you create a constraint in the FPGA Editor, the constraint is written to the PCF file whenever you save your design. When you use the FPGA Editor to delete a constraint, and then save your design file, the line on which the constraint appears in the PCF file remains in the file but is automatically commented out. Some of the constraints supported in the FPGA Editor are listed in the following table.

**Constraints Supported in FPGA Editor**

Constraint	Accessed Through
block paths	Component Properties and Path Properties property sheet
define path	Viewed with Path Properties property sheet
location range	Component Properties Constraints page
locate macro	Macro Properties Constraints page
lock placement	Component Properties Constraints page
lock routing of this net	Net Properties Constraints page
lock routing	Net Properties Constraints page
maxdelay allnets	Main Properties Constraints page
maxdelay allpaths	Main Properties Constraints page
maxdelay net	Net Properties Constraints page
maxdelay path	Path Properties property sheet
maxskew	Main Properties Constraints page
maxskew net	Net Properties Constraints page
offset comp	Component Properties Offset page
penalize tilde	Main Properties Constraints page
period	Main Properties Constraints page
period net	Net Properties Constraints page
prioritize net	Net Properties Constraints page
prohibit site	Site Properties property sheet

## Locked Nets and Components

If a net is locked, you cannot unroute any portion of the net, including the entire net, a net segment, a pin, or a wire. To unroute the net, you must first unlock it. You can add pins or routing to a locked net.

A net is displayed as locked in the FPGA Editor if the Lock Net [ *net\_name* ] constraint is enabled in the PCF file. You can use the Net Properties property sheet to remove the lock constraint.

When a component is locked, one of the following constraints is set in the PCF file.

**lock comp** [ *comp\_name* ]

**locate comp** [ *comp\_name* ]

**lock macro** [ *macro\_name* ]

**lock placement**

If a component is locked, you cannot unplace it, but you can unrout it. To unplace the component, you must first unlock it.

## Interaction Between Constraints

Schematic constraints are placed at the beginning of the PCF file by MAP. The start of this section is indicated with **SCHEMATIC START**. The end of this section is indicated with **SCHEMATIC END**. Because of a last-read order, all constraints that you enter in this file should come after **SCHEMATIC END**.

You are not prohibited from entering a user constraint before the schematic constraints section, but if you do, a conflicting constraint in the schematic-based section may override your entry.

Every time a design is remapped, the schematic section of the PCF file is overwritten by the mapper. The user constraints section is left intact, but certain constraints may be invalid because of the new mapping.

## XST Constraint File (XCF)

XST constraints can be specified in the XST Constraint File (XCF). The XCF has an extension of `.xcf`. For more information, see:

- ISE® Design Suite Help
- XST Constraint File (XCF) section in the *XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices*

## Constraint Priority

For more information, see the *Timing Closure User Guide (UG612)*.

In some cases more than one timing constraint covers the same path. The constraint conflict must be resolved, with the higher priority constraint taking precedence and being applied to the path, and the lower priority constraints being ignored for that path. The method of constraints resolution depends on both the order of constraint specification as well as the priority of the constraints specified. The rules of constraint priority resolution are described below. This determination is based upon the constraint prioritization or which constraint appears later in the Physical Constraints File (PCF), if there are overlapping constraints of the same priority. For example, if the design has two [Period](#) constraints that cover the same paths, the later Period constraint in the PCF file covers or analyzes these paths. The previous Period constraints have “0 items analyzed” in the timing report. In order to modify the default constraint resolution behavior, the constraint priority can be assigned using the **PRIORITY** keyword.

## File Priorities

When conflicting constraints have the same priority, the order of specification is used to determine the constraint that takes precedence. The resolution rule for identical priority constraints is the constraint that is specified last overwrites any previously defined constraints. This rule applies to constraints within a single User Constraints File (UCF) as well as constraints defined in multiple UCF files.

The following list defines the precedence order of identical priority constraints when these constraints are defined in different constraint files. The list is given in descending priority order with the highest priority constraint listed first.

- Constraints in a Physical Constraints File (PCF)
- Constraints in a User Constraints File (UCF)
- Constraints in a Netlist Constraints File (NCF)
- Attributes in a schematic or Constraints specified in HDL that are passed down in the netlist



## Xilinx Constraints

---

Each Xilinx® constraint includes the following, where applicable:

- Architecture Support
- Applicable Elements
- Description
- Propagation Rules
- Syntax
- Syntax Examples
- Additional information, if necessary

For more information about basic VHDL syntax, see [VHDL Attributes](#).

For more information about basic Verilog syntax, see [Verilog Attributes](#).

### Constraint Information

This chapter gives the following information for each constraint:

- Architecture Support  
Whether the constraint may be used with that device.
- Applicable Elements  
The elements to which the constraint may be applied.
- Description  
A brief description of the constraint, including its usage and behavior.
- Propagation Rules  
How the constraint is propagated.
- Syntax Examples  
Syntax examples for using the constraint with particular tools or methods. Not every tool or method is listed for every constraint. If a tool or method is not listed, the constraint may not be used with it.
- Additional Information  
Additional information is provided for certain constraints.

## Area Group

An Area Group (AREA\_GROUP) constraint:

- Is a design implementation constraint.
- Enables partitioning of the design into physical regions for mapping, packing, placement, and routing.
- Is attached to logical blocks.

The string value of the constraint identifies a named group of logical blocks that are to be 1) packed together by mapper, and 2) placed in the ranges if specified by PAR. If Area Group is attached to a hierarchical block, all sub-blocks in the block are assigned to the group.

Once defined, an Area Group can have a variety of additional constraints associated with it to control its implementation. For more information, see *Constraint Syntax* below.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

- Logic blocks
- Timing groups

For more information, see *Defining From Timing Groups* below.

## Propagation Rules

- When attached to a design element, Area Group is propagated to all applicable elements in the hierarchy below the component.
- It is illegal to attach Area Group to a net, signal, or pin.

## Area Group UCF Syntax

The User Constraints File (UCF) syntax for defining an Area Group is:

```
INST "X" AREA_GROUP=groupname ;
```

The UCF syntax for attaching constraints to an Area Group is any of the following:

```
AREA_GROUP "groupname" RANGE=range ;
```

```
AREA_GROUP "groupname" COMPRESSION=percent ;
```

```
AREA_GROUP "groupname" GROUP={OPEN|CLOSED} ;
```

```
AREA_GROUP "groupname" PLACE={OPEN|CLOSED} ;
```

*groupname* is the name assigned to an implementation partition to uniquely define the group.

Each of these additional Area Group constraints is described below.

## RANGE

RANGE defines the range of device resources that are available to place logic contained in the Area Group, in the same manner ranges are defined for the LOC constraint.

For all FPGA devices, the RANGE syntax is as follows:

**RANGE=SLICE\_X# Y#:SLICE\_X#Y#**

**RANGE=RAMB16\_X#Y#:RAMB16\_X#Y#**

**RANGE=MULT18X18\_X #Y#:MULT18X18\_X#Y#**

All FPGA devices SLICES are supported. If an Area Group contains both block RAM and SLICE elements, two separate Area Group RANGE components can be specified, one for block RAM elements and one for SLICE elements.

All locations in the FPGA device are specified in terms of X and Y coordinates. You can use the wildcard character for either the X coordinate or the Y coordinate.

The RANGE value can also be specified as a CLOCK REGION element or a set of CLOCK REGION elements. This syntax is supported for all INST types that can be used in Area Group constraints.

For all FPGA devices, Area Group is supported for various clock regions.

Single Region
<b>AREA_GROUP "groupname" RANGE=CLOCKREGION_X#Y#;</b>
Range of Clock Regions That Form a Rectangle
<b>AREA_GROUP "group_name" RANGE=CLOCKREGION_X#Y#:CLOCKREGION_X#Y#;</b>
List of Clock Regions
<b>AREA_GROUP "groupname" RANGE=CLOCKREGION_X#Y#,CLOCKREGION_X#Y#,...,;</b>

The valid X# and Y# values vary by device.

All components can be constrained by the CLOCKREGION range except IOB and BUF.

## Comma Separated RANGE Specifications

Comma separated RANGE specifications are not allowed on a single line. The second specification will be ignored. Each RANGE specification must be on its own line.

Following is an *invalid* syntax example.

```
INST "RM_data_control" AREA_GROUP = "RR_RM_data_control" ;
AREA_GROUP "RR_RM_data_control" RANGE = SLICE_X0Y44:SLICE_X27Y20, DSP48_X0Y25:DSP48_X0Y14;
```

Following is a *valid* syntax example.

```
INST "RM_data_control" AREA_GROUP = "RR_RM_data_control" ;
AREA_GROUP "RR_RM_data_control" RANGE = SLICE_X0Y44:SLICE_X27Y20;
AREA_GROUP "RR_RM_data_control" RANGE = DSP48_X0Y25:DSP48_X0Y14;
```

## Types of Logic Legal in the RANGE Constraint

Range Name	virtex4	virtex5	virtex6	spartan6
BSCAN_XnYn	X	X	X	X
BUFDS_XnYn		X		X
BUFGCTRL_XnYn	X	X	X	
BUFGMUX_XnYn				X
BUFHCE_XnYn			X	
BUFH_XnYn				X
BUFIO2FB_XnYn				X

Range Name	virtex4	virtex5	virtex6	spartan6
BUFIO2_XnYn				X
BUFIODQS_XnYn			X	
BUFIO_XnYn	X	X		
BUFO_XnYn			X	
BUFPLL_MCB_XnYn				X
BUFPLL_XnYn				X
BUFR_XnYn	X	X	X	
CAPTURE_XnYn			X	
CFG_IO_ACCESS_XnYn			X	
CRC32_XnYn		X		
CRC64_XnYn		X		
DCIRESET_XnYn			X	
DCI_XnYn	X	X	X	
DCM_ADV_XnYn	X	X		
DCM_XnYn				X
DNA_PORT_XnYn			X	
DPM_XnYn	X			
DSP48_XnYn	X	X	X	X
EFUSE_USR_XnYn			X	
EMAC_XnYn	X			
FIFO16_XnYn	X			
GLOBALSIG_XnYn	X	X	X	
GT11CLK_XnYn	X			
GT11_XnYn	X			
GTPA1_DUAL_XnYn				X
GTP_DUAL_XnYn		X		
GTXE1_XnYn			X	
GTX_DUAL_XnYn		X		
IBUFDS_GTXE1_XnYn			X	
ICAP_XnYn	X	X	X	X
IDELAYCTRL_XnYn	X	X	X	
ILOGIC_XnYn	X	X	X	X
IOB_XnYn	X	X	X	
IODELAY_XnYn		X	X	X
IPAD_XnYn	X	X	X	X
MCB_XnYn				X
MMCM_ADV_XnYn			X	
MONITOR_XnYn	X			



Range Name	virtex4	virtex5	virtex6	spartan6
OCT_CAL_XnYn				X
OLOGIC_XnYn	X	X	X	X
OPAD_XnYn	X	X	X	X
PCIE_XnYn		X	X	X
PCILOGIC_XnYn				X
PLL_ADV_XnYn		X		X
PMCD_XnYn	X			
PMVBRAM_XnYn		X	X	
PMVIOB_XnYn			X	
PMV_XnYn			X	
PPC405_ADV_XnYn	X			
PPC440_XnYn		X		
PPR_FRAME_XnYn			X	
RAMB16_XnYn	X			X
RAMB18_XnYn			X	
RAMB36_XnYn		X	X	
RAMB8_XnYn				X
SLICE_XnYn	X	X	X	X
STARTUP_XnYn			X	
SYSMON_XnYn		X	X	
TEMAC_XnYn		X	X	
TIEOFF_XnYn	X	X	X	X
USR_ACCESS_XnYn			X	

## Sites That Do Not Conform to the Normal X, Y Format

The following sites do not conform to the normal X, Y format and can be used in an Area Group range. The syntax is:

```
AREA_GROUP "group" RANGE=site1; AREA_GROUP "group" RANGE=site2;
```

Site Name	virtex4	virtex5	virtex6	spartan6
CAPTURE	X	X		
DCIRESET	X	X		
DNA_PORT				X
EFUSE_USR		X		
FRAME_ECC	X	X	X	
JTAGPPC	X	X		
KEY_CLEAR		X		
PAD				X
PMV	X	X		X
POST_CRC_INTERNAL				X
SLAVE_SPI				X
SPI_ACCESS				X
STARTUP	X	X		X
SUSPEND_SYNC				X
USR_ACCESS_SITE	X	X		

## COMPRESSION

COMPRESSION defines the compression factor for the Area Group constraints. The percent values can be from 0 to 100. If the Area Group does not have a RANGE, only 0 (no compression) and 1 (maximum compression) are meaningful. The mapper computes the number of CLB components *s* in the Area Group from the RANGE and attempts to compress the logic into the percentage specified. Compression does not apply to BRAM, or DSP block/multipliers.

The compression factor is similar to the **-c** option in MAP, except that it operates on the Area Group instead of the whole design. Area Group compression interacts with the **-c** map option as follows:

- Area Groups with a compression factor are not affected by the **-c** option. (Logic that is not part of an Area Group is not merged with grouped logic if the Area Group has its own compression factor.)
- Area Groups without a compression factor are affected by the **-c** option. The mapper may attempt to combine ungrouped logic with logic that is part of an Area Group without a compression factor.
- At no time is the logic from two separate Area Groups combined.
- The **-c** map option does not force compression among slices in the same Area Group.

The Map Report (MRP) includes a section that summarizes Area Group processing.

If a symbol that is part of an Area Group contains a **LOC** constraint, the mapper removes the symbol from the Area Group and processes the LOC constraint.

Logic that does not belong to any Area Group can be pulled into the region of logic belonging to an Area Group, as well as being packed or merged with such logic to form SLICES.

COMPRESSION on Area Group constraints does not apply when Timing Driven Packing and Placement is in MAP(-timing).

COMPRESSION on Area Group constraints is not supported for Virtex®-5 devices.

## GROUP

GROUP controls the packing of logic into physical components (slices) as follows.

- **CLOSED**  
Do not allow logic *outside* the Area Group to be combined with logic *inside* the Area Group.
- **OPEN**  
Allow logic *outside* the Area Group to be combined with logic *inside* the Area Group.  
The default value is GROUP=OPEN.

## PLACE

PLACE controls the allocation of resources in the Area Group's RANGE, as follows.

- **CLOSED**  
Do not allow comps that are not members of the Area Group to be placed within the RANGE defined for the Area Group.
- **OPEN**  
Allow comps that are not members of the Area Group to be placed within the RANGE defined for the Area Group.  
The default value is PLACE=OPEN.

## Area Group Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax Examples

- Attach **AREA\_GROUP**=*groupname* to a valid instance.
- Attach **RANGE** =*range* to a CONFIG symbol.
- Attach **COMPRESSION**=*percent* to a CONFIG symbol.
- Attach **GROUP**=**{ OPEN | CLOSED }** to a CONFIG symbol.
- Attach **PLACE**=**{ OPEN | CLOSED }** to a CONFIG symbol.
- Attach to a CONFIG symbol.  
For a value of TRUE, both PLACE and GROUP must be CLOSED.
- Attribute Names
  - **AREA\_GROUP**
  - **RANGE** *range*
  - **COMPRESSION** *percent*
  - **GROUP**=**{ OPEN | CLOSED }**
  - **PLACE**=**{ OPEN | CLOSED }**
- Attribute Values:
  - *groupname*
  - *range*
  - *percent*
  - **GROUP**=**{ OPEN | CLOSED }**
  - **PLACE**=**{ OPEN | CLOSED }**

### UCF and NCF Syntax Example

```
INST "state_machine_X" AREA_GROUP=group1;AREA_GROUP "group1"RANGE=SLICE_X1Y1:SLICE_X10Y10;
```

- Assigns all the logical blocks in **state\_machine\_X** to the area group **group1**.
- Places logic in the physical area between:
  - **SLICE row 1, column 1**
  - and
  - **SLICE row 10, column 10**

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

## Defining From Timing Groups

To create an area group based on a timing group, use the following User Constraints File (UCF) and Netlist Constraints File (NCF) syntax:

```
TIMEGRP timing_group_name AREA_GROUP = area_group_name ;
```

- *timing\_group\_name*  
The name of a previously defined timing group.
- *area\_group\_name*  
The name of a new area group to be defined from the [Timing Group](#) contents.

This is equivalent to manually assigning each member of the timing group to *area\_group\_name*. The area group name defined by this statement can be used in RANGE constraints, just like any other area group name.

## TNM\_NET Groups

In the AREA\_GROUP definition, the *timing\_group\_name* is generally a [Timing Name Net](#) group. This allows area groups to be formed based on the loads of clock or other control nets. Defining AREA\_GROUP constraints from [Timing Group](#) constraints can improve placement of designs with many different clock domains in devices that have more clocks than clock regions.

## TNM and TIMEGRP Groups

You can also specify

- A [Timing Name](#) group name.  
or
- The name of a user group defined by a [Timing Group](#) statement.

Edge qualifiers used in the TIMEGRP definition are ignored when determining area group membership. In all cases, the AREA\_GROUP members are determined after the TIMEGRP has been propagated to its target elements.

TIMEGRP constraints can contain only synchronous elements and pads. Area groups defined from timing groups also contain only these element types. If an AREA\_GROUP is defined by a TIMEGRP that contains only Flip-Flops or Latches, assigning a RANGE to that group is useful only if ungrouped logic is also allowed within the area. For this reason, do not define COMPRESSION for such groups.

## PERIOD Specifications

If a [Timing Name](#) is used by a [Period](#) specification, and is traced into any DCM, PLL, or MMCM, new TNM\_NET groups and PERIOD specifications are created at the DCM, PLL, or MMCM outputs. If the original TNM\_NET is used to define an area group, and if more than one clock tap is used on the DCM, PLL, or MMCM, the area group is split into separate groups at each clock tap.

For example, assume you have the following UCF constraints:

```
NET "clk" TNM_NET="clock";  
TIMESPEC "TS_clk" = PERIOD "clock" 10 MHz;  
TIMEGRP "clock" AREA_GROUP="clock_area";
```

If the net **clk** is traced into a DCM, PLL, or MMCM:

- A new group and PERIOD specification is created at each clock tap.
- A new area group is created at each clock tap.

A suffix indicates the clock tap name. If the CLK0 and CLK2X taps are used, the AREA\_GROUP clock\_area\_CLK0 and clock\_area\_CLK2X are defined automatically.

When AREA\_GROUP definitions are split in this manner, NGDBuild issues an informational message, showing the names of the new groups. Use these new group names, rather than the ones originally specified, in RANGE constraints.

## Asynchronous Register

The Asynchronous Register (ASYNC\_REG) constraint:

- Is a timing constraint.
- Improves the behavior of asynchronously clocked data for simulation.
- Disables X propagation during timing simulation.

In the event of a timing violation, the previous value is retained on the output instead of going unknown.

### Architecture Support

Applies to all FPGA devices and no CPLD devices.

### Applicable Elements

The Asynchronous Register constraint:

- Can be attached to registers and latches only.
- Should be used only on registers or latches with asynchronous inputs (D input or the CE input).

### Propagation Rules

The Asynchronous Register constraint applies to the register or latch to which it is attached.

### Constraint Values

- TRUE
- FALSE

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute ASYNC_REG : string;
```

Specify the VHDL constraint as follows:

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

#### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

#### UCF and NCF Syntax

```
INST "instance_name" ASYNC_REG = {TRUE|FALSE};
```

The default is FALSE.

If no Boolean value is supplied, it is considered TRUE.

### **Constraints Editor Syntax**

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.



## BEL

The BEL (BEL) constraint:

- Is an advanced placement constraint.
- Locks a logical symbol to a particular BEL site in:
  - A slice
  - or
  - An IOB
- Differs from the [Location \(LOC\)](#) constraint in that LOC allows specification to the component level. Examples of components include:
  - SLICE
  - BRAM
  - ILOGIC
  - OLOGIC
  - IOB
- Allows specification as to which particular BEL site of the component to be used. For example, specifying the specific LUT or FF to be used within a SLICE.
- Should always be used with an appropriate [Location \(LOC\)](#) or [Relative Location \(RLOC\)](#) attribute.

An IOB BEL constraint does not direct the mapper to pack the register into an IOB component. Some other feature (the **-pr** switch, for example) must cause the packing. Once the register is directed to an IOB, the BEL constraint causes the proper placement within the IOB.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

Registers	Latches
LUT LUTRAM RAMB18	SRL

## Propagation Rules

It is legal to place a BEL constraint only on an appropriate instance with a valid [Location \(LOC\)](#) or [Relative Location \(RLOC\)](#) attribute.

## Constraint Values

Value	Identify in a Slice
F, G	Specific LUT, SRL16, and distributed RAM components
A6LUT, B6LUT, C6LUT, D6LUT	
A5LUT, B5LUT, C5LUT, D5LUT	
FFA, FFB, FFC, FFD, FFX, FFY	Specific Flip-Flops, Latches, and other elements
XORF, XORG	XORCY elements

The following values are also valid for Virtex®-6 devices:

- AFF
- BFF
- CFF
- DFF
- A5FF
- B5FF
- C5FF
- D5FF

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
BEL

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* BEL = " {value}" *)
```

### UCF and NCF Syntax

```
INST "instance_name" BEL={value};
```

The syntax for the RAMB BEL instance is:

```
INST "upper_BRAM_instance_name" LOC = RAMB36_XnYn|BEL = UPPER;
```

```
INST "lower_BRAM_instance_name" LOC = RAMB36_XnYn|BEL = LOWER;
```

### UCF and NCF Syntax Example

```
INST "ramb18_inst0" LOC = RAMB36_X0Y2|BEL = UPPER; INST "ramb18_inst1" LOC = RAMB36_X0Y2|BEL = LOWER;
```

The following statement locks xyzzy to the FFX site on the slice.

```
INST "xyzzy" BEL=FFX;
```

## PlanAhead™ Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Block Name

The Block Name (BLKNM) constraint:

- Is an advanced mapping constraint
- Assigns block names to qualifying primitives and logic elements.

If the same Block Name constraint is assigned to more than one instance, the software attempts to map them into the same block. Conversely, two symbols with different Block Name names are not mapped into the same block. Placing similar Block Name constraints on instances that do not fit within one block causes an error.

Specifying identical Block Name constraints on FMAP tells the software to group the associated function generators into a single SLICE. By using the Block Name constraint, you can partition a complete SLICE without constraining the SLICE to a physical location on the device.

Block Name constraints, such as [Location \(LOC\)](#) constraints, are specified from the design. Since hierarchical paths are not prefixed to Block Name constraints, Block Name constraints for different SLICES must be unique throughout the entire design.

For information on attaching hierarchy to block names, see [Hierarchical Block Name](#).

Block Name allows any elements except those with a different Block Name to be mapped into the same physical component. Elements without a Block Name can be packed with those that have a Block Name.

For information on allowing only elements with the same [XBLKNM](#) to be mapped into the same physical component, see [XBLKNM](#).

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Block Name constraint may be used with an FPGA device in one or more of the following design elements, or categories of design elements. Not all devices support all elements. To see which design elements can be used with which devices, see the *Libraries Guides*. For more information, see the device data sheet.

- Flip-flop and latch primitives
- Any I/O element or pad
- FMAP
- ROM primitives
- RAMS and RAMD primitives
- Carry logic primitives
- Block RAM

You can also attach the Block Name constraint to the net connected to the pad component in a User Constraints File (UCF) file.

NGDBuild transfers the constraint from the net to the pad instance in the NGD file so that it can be processed by the mapper.

Use the following syntax:

```
NET "net_name" BLKNM=property_value ;
```

## Propagation Rules

When attached to a design element, the Block Name constraint is propagated to all applicable elements in the hierarchy within the design element.

## Constraint Values

*block\_name*

A valid block name for that type of symbol

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
BLKNM
- Attribute Value  
*block\_name*

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute blknm : string;
```

Specify the VHDL constraint as follows:

```
attribute blknm  
of {component_name | signal_name | entity_name | label_name}: {component | signal | entity | label}  
is "block_name";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
( * BLKNM = "blk_name" * )
```

### UCF and NCF Syntax

```
INST "instance_name" BLKNM=block_name ;
```

For information on assigning hierarchical block names, see [Hierarchical Block Name](#).

The following statement assigns an instantiation of an element named **block1** to a block named **U1358**.

```
INST "$1I87/block1" BLKNM=U1358;
```

### XCF Syntax

```
MODEL "entity_name" blknm = block_name ;
```

```
BEGIN MODEL "entity_name"
```

```
INST "instance_name" blknm = block_name ;
```

```
END;
```

## BUFG

The BUFG (BUFG) constraint:

- Is an advanced fitter constraint.
- Is a synthesis constraint.
- Maps the tagged signal to a global net when applied to:
  - An input buffer
  - or
  - An input pad net

When applied to an internal net, the tagged signal is either:

- Routed directly to a global net.
- or
- Brought out to a global control pin to drive the global net, as supported by the device architecture.

### Architecture Support.

Supports CPLD devices only. Does not support FPGA devices.

### Applicable Elements

The BUFG constraint applies to any:

- Input buffer (IBUF)
- Input pad net
- Internal net that drives a CLK, OE, SR, or DATA\_GATE pin.

### Propagation Rules

- When attached to a *net*, the BUFG constraint has a net or signal form. No special propagation is required.
- When attached to a *design element*, the BUFG constraint is propagated to all applicable elements in the hierarchy within that design element.

## Constraint Values

- CLK
  - Designates a global clock pin.
  - Supports all CPLD devices.
- OE
 

Designates either:

  - A global tristate control pin  
Supports all CPLD devices except CoolRunner™-II and CoolRunner XPLA3 devices.
  - An internal global tristate control line  
Supports CoolRunner-II devices only.
- SR
  - Designates a global set/reset pin.
  - Supports all CPLD devices except CoolRunner-II and CoolRunner XPLA3 devices.
- DATA\_GATE
 

Maps to the [Data Gate](#) latch enable control line

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to an IBUF instance of the input pad connected to an IBUF input
- Attribute Name  
BUFG

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute BUFG: string;
```

Specify the VHDL constraint as follows:

```
attribute BUFG of signal_name : signal is "{CLK|OE|SR|DATA_GATE} ";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
( * BUFG = "{CLK|OE|SR|DATA_GATE} " * )
```

### UCF and NCF Syntax

```
NET "net_name" BUFG={CLK|OE|SR|DATA_GATE};
```

```
INST "instance_name" BUFG={CLK|OE|SR|DATA_GATE};
```

The following statement maps the signal named **fastclk** to a global clock net.

```
NET "fastclk" BUFG=CLK;
```

### XCF Syntax

```
BEGIN MODEL "entity_name"  
    NET "signal_name" BUFG = {CLK|OE|SR|DATA_GATE} ;  
END;
```



## Clock Dedicated Route

The Clock Dedicated Route (CLOCK\_DEDICATED\_ROUTE) constraint:

- Is an advanced constraint.
- Directs the software whether to follow clock placement rules for a specific architecture.

## Clock Placement Rules

Clock placement rules must be followed when the Clock Dedicated Route constraint is:

- Not used, or
- Set to TRUE

Otherwise, placement errors.

When the Clock Dedicated Route constraint is set to FALSE, the software:

- Ignores the specific clock placement rule.
- Continues with Place and Route (PAR).

If possible, fix all clock placement rule violations. This ensures optimal clocking performance. Use the Clock Dedicated Route constraint only when absolutely necessary to violate a clock placement rule.

For more information about specific clock placement rules, see the *Hardware User's Guide*.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Clock Dedicated Route constraint applies to:

- Clock Buffers
- Clock Manager Blocks
- High Speed I/O Blocks

## Propagation Rules

The Clock Dedicated Route constraint applies to the NET or INSTANCE PIN.

## Constraint Values

- TRUE
- FALSE

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
CLOCK\_DEDICATED\_ROUTE
- Attribute Values  
See *Constraint Values* above.

### UCF and NCF Syntax

```
PIN "BEL_INSTANCE_NAME.PIN" CLOCK_DEDICATED_ROUTE = {TRUE|FALSE};
```

BEL\_INSTANCE\_NAME.PIN is the specific input or output pin of the instance you want to constrain, such as the CLKIN input pin of a DCM instance.

## Collapse

The Collapse (COLLAPSE) constraint:

- Is an advanced fitter constraint.
- Forces a combinatorial node to be collapsed into all of its fanouts.

### Architecture Support

Supports CPLD devices only. Does not support FPGA devices.

### Applicable Elements

The Collapse constraint applies to any internal net.

### Propagation Rules

This constraint is a net constraint. Any attachment to a design element is illegal.

### Constraint Values

- YES
- NO
- TRUE
- FALSE

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

- Attach to a logic symbol or its output net
- Attribute Name  
COLLAPSE
- Attribute Values
  - TRUE
  - FALSE

#### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute collapse: string;
```

Specify the VHDL constraint as follows:

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

#### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
NET "net_name" COLLAPSE;
```

The following statement forces net \$1N6745 to collapse into all its fanouts.

```
NET "$1I87/$1N6745" COLLAPSE;
```

## Component Group

The Component Group (COMPGRP) constraint:

- Is an advanced grouping constraint.
- Identifies a group of components.
- Is used in the Physical Constraints File (PCF) only.

### Architecture Support

Applies to all FPGA devices and no CPLD devices.

### Applicable Elements

The Component Group constraint applies to groups of components.

### Propagation Rules

Not applicable.

### Constraint Values

comp\_item is one of the following:

- **COMP** "comp\_name"
- **COMPGRP** "group\_name"

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### PCF Syntax

```
COMPGRP "group_name"=comp_item1... comp_itemn [EXCEPT comp_group];
```

## Configuration Mode

The Configuration Mode (CONFIG\_MODE) constraint communicates to PAR which dual purpose configuration pins can be used as general purpose I/O components.

PAR uses Configuration Mode to prohibit the use of the following Dual Purpose I/O components if they are required for Configuration Mode:

- S\_SELECTMAP+READBACK
- M\_SELECTMAP+READBACK

Use **bitgen -g Persist** to retain these I/O components for Readback and Reconfiguration.

In the following cases, PAR uses Dual Purpose I/O components as General Purpose I/O components only when necessary:

- CONFIG\_MODE: S\_SELECTMAP
- M\_SELECTMAP

## Architecture Support

The Configuration Mode constraint supports the following devices:

- Spartan®-3
- Virtex®-4
- Virtex-5
- Virtex-6
- 7 series

## Applicable Elements

The Configuration Mode constraint attaches to the CONFIG symbol.

## Propagation Rules

The Configuration Mode constraint applies to dual-purpose I/O components.

## Constraint Values

string

- M\_SERIAL  
Master Serial Mode (The default value)
- S\_SERIAL  
Slave Serial Mode
- B\_SCAN  
Boundary Scan Mode
- B\_SCAN+READBACK  
Boundary Scan Mode with Persist expected.
- M\_SELECTMAP  
Master SelectMAP Mode , 8-bit width
- M\_SELECTMAP+READBACK  
Master SelectMAP Mode, 8-bit width, with Persist expected.
- S\_SELECTMAP  
Slave SelectMAP Mode, 8-bit width
- S\_SELECTMAP+READBACK  
Slave SelectMAP Mode, 8-bit width, with Persist expected.
- S\_SELECTMAP16  
Slave SelectMAP Mode, 16-bit width
- S\_SELECTMAP16+READBACK  
Slave SelectMAP Mode, 16-bit width, with Persist expected.
- S\_SELECTMAP32  
Slave SelectMAP Mode, 32-bit width
- S\_SELECTMAP32+READBACK  
Slave SelectMAP Mode with Persist expected.

For S\_SELECTMAP32 and S\_SELECTMAP32+READBACK, you can select S\_SELECTMAP16 and S\_SELECTMAP16+READBACK for Virtex-5 devices to have the correct number of data pins needed persisting after configuration.

The following values apply to 7 series devices only:

- SPIx1  
Serial Peripheral Interface, 1-bit width
- SPIx2  
Serial Peripheral Interface, 2-bit width
- SPIx4  
Serial Peripheral Interface, 4-bit width
- BPI8  
Byte Peripheral Interface (Parallel NOR), 8-bit width
- BPI16  
Byte Peripheral Interface (Parallel NOR), 8-bit width

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF Syntax

```
CONFIG CONFIG_MODE=string;
```



## CoolCLOCK

The CoolCLOCK (COOL\_CLK) constraint:

- Supports CoolRunner™-II devices only.
- Reduces clocking power within a CoolRunner-II device by combining clock division circuitry with the DualEDGE circuitry.

### Reducing Clock Power

The clock net can be a significant power drain. To reduce the clock power:

1. Drive the net at half frequency.
2. Double the clock rate using DualEDGE triggered macrocells.

### Architecture Support

CoolRunner-II

### Applicable Elements

The CoolCLOCK constraint applies to any input pad or internal signal driving a register clock.

### Propagation Rules

- Applying the CoolCLOCK constraint to a clock net is equivalent to:
  - Passing the clock through a divide-by-two clock divider (CLK\_DIV2) then
  - Replacing all flip-flops controlled by that clock with DualEDGE flip-flops.
- Using the CoolCLOCK constraint does not alter your overall design functionality.
- You cannot use the CoolCLOCK constraint on a clock that triggers any flip-flop on the low-going edge. The CoolRunner™-II clock divider can be triggered only on the high-rising edge of the clock signal.
- If there are any DualEDGE flip-flops in the source, the clock that controls any of them cannot be specified as a CoolCLOCK constraint.
- If there is already a clock divider in the source, you cannot also use the CoolCLOCK constraint. CoolRunner-II devices contain only one clock divider.

### Constraint Values

- TRUE
- FALSE

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a input pad or internal signal driving a register clock
- Attribute Name  
COOL\_CLK
- Attribute Values  
See *Constraint Values* above.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute cool_clk: string;
```

Specify the VHDL constraint as follows:

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

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
NET "signal_name" COOL_CLK;
```

## Data Gate

The Data Gate (DATA\_GATE) constraint:

- Supports certain CoolRunner™-II devices only.
- Can reduce power consumption.

## Reducing Power Consumption

Each I/O pin input signal passes through a latch. This latch can block the propagation of incident transitions during periods when those transitions are not relevant to the design. Input transitions that do not affect the design still consume power, if not latched, since they are routed among the device function blocks.

By placing the Data Gate control I/O pin on the device:

- Selected I/O pin inputs become latched.
- The power dissipation associated with external transitions on those pins is eliminated.

## Architecture Support

The Data Gate constraint supports CoolRunner-II devices with 128 macrocells or more only.

## Applicable Elements

The Data Gate constraint applies to I/O pads and pins.

## Propagation Rules

Applying the Data Gate constraint to any I/O pad indicates that the pass-through latch on that device pin is to respond to the Data Gate control line.

- Any I/O pad (except the Data Gate control I/O pin itself), including clock input pads, can be configured to be latched by applying the Data Gate constraint.
- All other I/O pads that do not have a Data Gate constraint remain unlatched at all times.

The Data Gate control signal itself can be :

- Received from off-chip by means of the Data Gate I/O pin.
- Generated based on inputs that remain unlatched (pads without Data Gate attributes).

For more information on using Data Gate with Verilog and VHDL designs, see [BUFG](#).

## Constraint Values

- TRUE
- FALSE

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to I/O pads and pins
- Attribute Name  
DATA\_GATE
- Attribute Values  
See *Constraint Values* above.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute DATA_GATE : string;
```

Specify the VHDL constraint as follows:

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

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
NET "signal_name" DATA_GATE;
```

### XCF Syntax

```
BEGIN MODEL "entity_name"
```

```
    NET "signal_name" data_gate={TRUE|FALSE};
```

```
END ;
```

## DCI Cascade

The DCI Cascade (DCI\_CASCADE) constraint identifies a DCI master bank and its corresponding slave banks.

In Virtex®-5 and Virtex-6 devices, I/O banks that require DCI reference voltage can be cascaded with other DCI I/O banks. One set of VRN/VRP pins can be used to provide reference voltage to several I/O banks. This results in more usable pins and in reduced power usage because fewer VR pins and DCI controllers are used.

There can be multiple instances of DCI Cascade in order to specify multiple master-slave pairs. BitGen uses information from DCI Cascade to program DCI controllers for different banks and have them cascade up or down. The placer also uses this information to determine whether VR pins in slave banks can be used for other purposes.

Each instance of DCI Cascade must have one master bank and one or more slave banks.

- Enter the banks as a space-separated list.
- The first value in the list is the master bank.
- All subsequent values are slave banks.
- The slave banks draw DCI reference voltage from the master bank.

**Note** This restriction does not apply to Virtex-6 devices.

Cascaded banks must:

- Be in the same column (left, center or right).
- Have the same VCCO setting.

## Architecture Support

The DCI Cascade constraint applies to Virtex-5 and Virtex-6 devices only.

## Applicable Elements

A DCI Cascade attribute on the top level design block.

## Propagation Rules

The DCI Cascade constraint is:

- Placed as an attribute on the CONFIG block.
- Propagated to the physical design object.

## Constraint Values

The DCI Cascade constraint has the following values.

- *<master>* = [1...MAX\_NUM\_BANKS]
- *<slave1>* = [1...MAX\_NUM\_BANKS]
- *<slave2>* = [1...MAX\_NUM\_BANKS]

The constraint values follow these rules:

- All values in the list are legitimate I/O banks in the Virtex-5 device.
- The master bank must have an IOB with an I/O standard that requires DCI reference voltage.

**Note** This restriction does not apply to Virtex-6 devices.

- All slave banks must have the same VCCO setting as the master bank.
- If there are banks between the master and slave, they must be able to cascade in the required direction.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

```
CONFIG DCI_CASCADE = "<master> <slave1> <slave2> ...";
```

```
CONFIG DCI_CASCADE = "11 13 15 17";
```

### PCF Syntax

```
CONFIG DCI_CASCADE = "<master>, <slave1>, <slave2>, ..."
```

## DCI Value

The DCI Value (DCI\_VALUE) constraint determines which buffer behavioral models are associated with the IOB components in the generation of an IBIS file using IBISWriter.

### Architecture Support

- Virtex®-4
- Virtex-5
- Virtex-6
- Spartan®-3

### Applicable Elements

The DCI Value constraint applies to IOB components.

### Propagation Rules

Applies to the IOB to which it is attached.

### Constraint Values

integer

- Legal values are integers 25 through 100 with an implied units of ohms.
- The default value is 50 ohms.

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### UCF and NCF Syntax

```
INST pin_name DCI_VALUE = integer;
```

## Default

The Default (DEFAULT) constraint allows you to set a new default value for several constraints.

A specific constraint overrides the Default constraint value where applicable.

## Architecture Support

The Default constraint supports the following constraints and their architectures:

- KEEPER, PULLDOWN

Applies to all FPGA devices and the CoolRunner™-II CPLD device.

- PULLUP

Applies to all FPGA devices and the CoolRunner XPLA3 and CoolRunner-II CPLD devices.

## Applicable Elements

For the applicable elements for each of the constraints supported by the Default constraint, see:

- [Keeper](#)
- [Float](#)
- [Pulldown](#)
- [Pullup](#)

## Propagation Rules

For the propagation rules for each of the constraints supported by the Default constraint, see:

- [Keeper](#)
- [Float](#)
- [Pulldown](#)
- [Pullup](#)

## Constraint Values

- [Keeper](#)
- [Float](#)
- [Pulldown](#)
- [Pullup](#)



## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

The basic syntax for attaching a Default constraint to a schematic is:

- Attach to a net, instance, or pin
- Attribute Name  
DEFAULT *constraint\_name*
- Attribute Values  
Determined by the *constraint\_name*.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute attribute_name : string;  
attribute KEEPER: string;
```

Specify the VHDL constraint as follows:

```
attribute attribute_name of DEFAULT is attribute_value ;
```

For *attribute\_name*, see *Constraint Values* above.

Accepted *attribute\_value* depends on the attribute type as shown in the following example:

```
attribute of DEFAULT KEEPER is "TRUE";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
( * CONSTRAINT_NAME = "constraint_value" * ) DEFAULT
```

For *constraint\_name*, see *Constraint Values* above.

The *constraint\_value* is case sensitive.

Accepted *constraint\_values* depend on the *constraint\_name*, as shown in the following example.

```
( * KEEPER = "TRUE" * ) DEFAULT
```

### UCF Syntax

```
DEFAULT constraint_name ;
```

### UCF Syntax Example

```
DEFAULT KEEPER = TRUE;
```

### XCF Syntax

```
BEGIN MODEL "entity_name"  
DEFAULT constraint_name [attribute_value] ;  
END;
```

Accepted *attribute\_values* depend on the attribute type.

### XCF Syntax Example

```
BEGIN MODEL "my_design"  
  
    DEFAULT keeper = TRUE;  
  
END ;
```

### NCF Syntax

Same as UCF syntax.

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

### PACE Syntax

The Pinout and Area Constraints Editor (PACE) tool:

- Is supported for CPLD devices only
- Is mainly used to assign location constraints to I/Os.
- Can be used to assign certain I/O properties such as I/O Standards.
- Can be accessed from the **Processes** window in ISE® Design Suite.

For more information, see the PACE help, especially the topics in *Editing Pins and Areas* in the *Procedures* section.

## Diff Term

The Diff Term (DIFF\_TERM) constraint:

- Is a basic mapping constraint.
- Turns the built-in 100 ohm differential termination *on* or *off*.

## Architecture Support

Supports Virtex®-6 and Spartan®-6 devices.

## Applicable Elements

The Diff Term constraint applies to differential I/O blocks such as IBUFDS\_DIFF\_OUT.

## Propagation Rules

Not applicable.

### Constraint Values

- TRUE  
Turns the built-in 100 ohm differential termination *on*.
- FALSE  
Turns the built-in 100 ohm differential termination *off*.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attribute Name  
DIFF\_TERM
- Attribute Values  
See *Constraint Values* above.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
Attribute DIFF_TERM: string;
```

Specify the VHDL constraint as follows:

```
attribute DIFF_TERM of block_name: signal is "{TRUE|FALSE}";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

The following statement configures the I/O to use the built-in 100 ohm termination.

```
INST "IO block name" DIFF_TERM = "{TRUE|FALSE}" ;
```

### XCF Syntax

```
BEGIN MODEL "entity_name "  
    NET "block_name " DIFF_term=true;  
END;
```

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Directed Routing

The Directed Routing (DIRECTED\_ROUTING) constraint allows you to maintain routing and timing for a small number of loads and sources.

Directed routing requires that the relative position between the sources and loads be maintained exactly the same with the use of the following constraints:

- [Location \(LOC\)](#)
- [Relative Location \(RLOC\)](#)
- [BEL](#)

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

Applies to the net to which it is attached.

## Propagation Rules

Not applicable.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

The following examples are for illustration only. They are not valid executables. Formulation of a directed routing constraint requires the placement of the source and load components in a fixed location relative to each other.

## FPGA Editor Syntax

To generate directed routing constraints with FPGA Editor, select:

### Tools > Directed Routing Constraints

FPGA Editor provides the following settings for the type of placement constraint to be generated automatically on the sources and loads components.

- Do Not Generate Placement Constraint Setting
- Use Relative Location Constraint Setting
- Use Absolute Location Constraint Setting

### Do Not Generate Placement Constraint Setting

The Do Not Generate Placement Constraint Setting:

- Generates a constraint for the routing only.
- Is used with existing RPM components.

```
NET "net_name" ROUTE="{2;1;-4;-1;-53320;2920;14;90;200;30;13!0;- 2091;1480;24!0;16;-8!}";
```

### Use Relative Location Constraint Setting

The Use Relative Location Constraint Setting generates an RPM component for the source and load components along with the routing constraint.

The RPM component can be relocated around the device letting the Placer make the final decision on placement.

In this example:

- Each [Relative Location \(RLOC\)](#) reference signals the launch of a new instance.
- There are accordingly three instances encompassed in the example.

```
NET "net_name" ROUTE="{2;1;-4;-1;-53320;2920;14;90;200;30;13!0;- 2091;1480;24!0;16;-8!}";
INST "inst1" RLOC=X3Y0;
INST "inst1" RPM_GRID=GRID;
INST "inst1" U_SET=macro name;
INST "inst1" BEL="F";
INST "inst2" RLOC=X3Y0;
INST "inst2" U_SET=macro name;
INST "inst2" BEL="G";
```

### Use Absolute Location Constraint Setting

The Use Absolute Location Constraint Setting causes the source and load components attached to the target net to be locked in place by specifying [Relative Location \(RLOC\)](#) constraints as well as an [Relative Location Origin \(RLOC\\_ORIGIN\)](#) constraint.

You can also specify [Location \(LOC\)](#) constraints manually.

```
NET "net_name" ROUTE="{2;1;-4;-1;-53320;2920;14;90;200;30;13!0;- 2091;1480;24!0;16;-8!}";
INST "inst1" RLOC=X3Y0;
INST "inst1" RPM_GRID=GRID;
INST "inst1" RLOC_ORIGIN=X87Y200;
INST "inst1" U_SET=macro name;
INST "inst1" BEL="F";
INST "inst2" RLOC=X0Y1;
INST "inst2" U_SET=macro name;
INST "inst2" BEL="F";
INST "inst3" RLOC=X3Y0;
INST "inst3" U_SET=macro name;
INST "inst3" BEL="G";
```

## Disable

The Disable (DISABLE) constraint:

- Is a timing constraint.
- Turns off specific path tracing controls.

A path tracing control determines whether a common type of path is *enabled* or *disabled* for timing analysis.

All path tracing control statements from any source (netlist, UCF, or NCF) are passed forward to the Physical Constraints File (PCF).

You cannot override a Disable constraint in the netlist with an [Enable](#) constraint in the User Constraints File (UCF).

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Disable constraint applies globally in the constraints file.

## Propagation Rules

The Disable constraint disables timing analysis of specified block delay symbol.

## Constraint Values

`delay_symbol_name`

The name of one of the standard block delay symbols for path tracing or a specific delay name in the data sheet

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

**DISABLE**=*delay\_symbol\_name* ;

These symbols are listed in the following table. Component delay names are also supported in the Physical Constraints File (PCF).

### Standard Block Delay Symbols for Path Tracing

Delay Symbol Name	Path Type	Default
reg_sr_o	Asynchronous Set/Reset to output propagation delay	Disabled
reg_sr_r	Asynchronous Set/Reset to recovery path	Disabled for Virtex®-5 and earlier architectures Enabled for Virtex-6 and Spartan®-6 architectures
reg_sr_clk	Synchronous Set/Reset to clock setup and hold checks	Enabled
lat_d_q	Data to output transparent latch delay	Disabled
lat_ce_q	Clock Enable to output transparent latch delay	Disabled
ram_we_o	RAM write enable to output propagation delay	Enabled
io_pad_i	IO pad to input propagation delay	Enabled
io_t_pad	IO tristate to pad propagation delay	Enabled
io_o_i	IO output to input propagation delay. Disabled for tristated IOB components.	Enabled
io_o_pad	IO output to pad propagation delay.	Enabled

### PCF Syntax

Same as UCF and NCF syntax.



## Drive

The Drive (DRIVE) constraint:

- Is a basic mapping directive.
- Selects the output for drive strength for all supported FPGA architectures.
- Selects output drive strength (mA) for the SelectIO™ technology buffers that use any of the following interface I/O standards:
  - LVTTTL
  - LVCMOS12
  - LVCMOS15
  - LVCMOS18
  - LVCMOS25
  - LVCMOS33

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Drive constraint may be used with an FPGA device in one or more of the following design elements, or categories of design elements. Not all devices support all elements. To see which design elements can be used with which devices, see the *Libraries Guides*. For more information, see the device data sheet.

- IOB output components (such as OBUF and OFD)
- SelectIO technology output buffers with:
  - IOSTANDARD = LVTTTL
  - LVCMOS15
  - LVCMOS18
  - LVCMOS25, or
  - LVCMOS33
- Nets

## Propagation Rules

- The Drive constraint is illegal when attached to a net or signal, except when the net or signal is connected to a pad. In this case, Drive is treated as attached to the pad instance.
- When attached to a design element, the Drive constraint is propagated to all applicable elements in the hierarchy below the design element.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid IOB output component
- Attribute Name  
DRIVE
- Attribute Values

For a list of the constraint values, see the *UCF and NCF Syntax* section below.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute drive: string;
```

Specify the VHDL constraint as follows:

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

For a list of the constraint values, see the *UCF and NCF Syntax* section below.

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
( * DRIVE = "value" * )
```

For a list of the constraint values, see the *UCF and NCF Syntax* section below.

### UCF and NCF Syntax

This section gives UCF and NCF Syntax examples for the following:

- IOB Output Components (UCF)
- SelectIO Technology Output Components

#### IOB Output Components (UCF)

For Spartan®-3 and higher devices or Virtex®-4 and higher devices:

```
INST "instance_name" DRIVE={ 2 | 4 | 6 | 8 | 12 | 16 | 24 };
```

12 mA is the default

#### SelectIO Technology Output Components

This section applies to the following components:

- IOBUF\_SelectIO
- OBUF\_SelectIO
- OBUFT\_SelectIO

The following table shows syntax examples for the named standard with Spartan-3 devices and higher or Virtex-4 devices and higher. The default in each case is 12 mA.

Standard	Syntax
LVTTL	<code>INST "instance_name" DRIVE={2 4 6 8 12 16 24};</code>
LVC MOS12 LVC MOS15 LVC MOS18	<code>INST "instance_name" DRIVE={2 4 6 8 12 16};</code>
LVC MOS25 LVC MOS33	<code>INST "instance_name" DRIVE={2 4 6 8 12 16 24};</code>

### XCF Syntax

```
MODEL "entity_name" drive={2|4|6|8|12|16|24};

BEGIN MODEL "entity_name"

NET "signal_name" drive={2|4|6|8|12|16|24};

END;
```

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Enable

The Enable (ENABLE) constraint:

- Is a timing constraint.
- Turns on specific path tracing controls.

A path tracing control determines whether a common type of paths is *enabled* or *disabled* for timing analysis.

For more information, see the [Disable](#) constraint.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Enable constraint applies globally in the constraints file.

## Propagation Rules

The Enable constraint enables timing analysis for specified path delays.

## Constraint Values

delay\_symbol\_name

- The name of one of the standard block delay symbols for path tracing symbols shown in the following table.

or

- A specific delay name defined in the data sheet.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

The Enable constraint can be applied only to a global [Timing Specifications](#) constraint.

The User Constraints File (UCF) path tracing syntax is:

```
ENABLE= delay_symbol_name ;
```

### Standard Block Delay Symbols for Path Tracing

Delay Symbol Name	Path Type	Default
reg_sr_o	Asynchronous Set/Reset to output propagation delay	Disabled
reg_sr_r	Asynchronous Set/Reset to recovery path	Disabled for Virtex®-5 and earlier architectures Enabled for Virtex-6 and Spartan®-6 architectures
reg_sr_clk	Synchronous Set/Reset to clock setup and hold checks	Enabled
lat_d_q	Data to output transparent latch delay	Disabled
lat_ce_q	Clock Enable to output transparent latch delay	Disabled
ram_we_o	RAM write enable to output propagation delay	Enabled
io_pad_i	I/O pad to input propagation delay	Enabled
io_t_pad	I/O tristate to pad propagation delay	Enabled
io_o_1	I/O output to input propagation delay. Disabled for tristated IOB components	Enabled
io_o_pad	I/O output to pad propagation delay	Enabled

### PCF Syntax

**ENABLE**=*delay\_symbol\_name* ;

**TIMEGRP** *name* **ENABLE**=*delay\_symbol\_name* ;

## Enable Suspend

The Enable Suspend (ENABLE\_SUSPEND) constraint:

- Supports Spartan®-3A and Spartan-6 devices only.
- Can be used in the User Constraints File (UCF) only.
- Defines the behavior of the [Suspend](#) power-reduction mode.

### Architecture Support

- Spartan-3A
- Spartan-6

### Applicable Elements

The Enable Suspend constraint:

- Is a global attribute.
- Is not attached to any particular element.

### Propagation Rules

The Enable Suspend constraint:

- Is a global attribute.
- Is attached to the entire design.

### Constraint Values

- NO (default)  
Disables this feature.
- FILTERED
  - Activates SUSPEND.
  - Activates the glitch filter.
  - Requires longer pulse width to activate.
- UNFILTERED
  - Activates SUSPEND.
  - Bypasses the glitch filter.
  - Allows quicker activation of SUSPEND.

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### UCF Syntax

```
CONFIG ENABLE_SUSPEND="{NO|FILTERED|UNFILTERED}";
```

#### UCF Syntax Example

```
CONFIG ENABLE_SUSPEND="FILTERED";
```

## Fast

The Fast (FAST) constraint:

- Is a basic mapping constraint.
- Increases the speed of an IOB output.
- May increase noise and power consumption.

### Architecture Support

Applies to all FPGA devices and all CPLD devices.

### Applicable Elements

- Output primitives
- Output pads
- Bidirectional pads

You can also attach the Fast constraint to the net connected to the pad component in a User Constraints File (UCF). NGDBuild transfers the Fast constraint from the net to the pad instance in the NGD file so that it can be processed by the mapper.

Use the following syntax:

```
NET "net_name" FAST;
```

### Propagation Rules

- The Fast constraint is illegal when attached to a net, except when the net is connected to a pad. In this instance, Fast is treated as attached to the pad instance.
- When attached to a macro, module, or entity, the Fast constraint is propagated to all applicable elements in the hierarchy below the module.

### Constraint Values

- TRUE
- FALSE

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
FAST
- Attribute Values  
See *Constraint Values* above.

#### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute FAST: string;
```

Specify the VHDL constraint as follows:

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

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

- **INST** "\$1I87/y2" **FAST**;  
Increases the output speed of the element **y2**.
- **NET** "net1" **FAST**;  
Increases the output speed of the pad to which **net1** is connected.

### XCF Syntax

```
BEGIN MODEL "entity_name"  
  NET "signal_name" fast={TRUE|FALSE};  
END;
```

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints



## Feedback

The Feedback (FEEDBACK) constraint defines the external DCM feedback path delay when the DCM is used in board de-skew applications.

- The delay is the maximum external path delay of the board trace.
- The delay should not include any internal FPGA path delays.

The Feedback constraint allows the timing software to:

- Determine the DCM/PLL/MMCM phase shift.
- Analyze the associated synchronous paths.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

Not applicable.

## Propagation Rules

The following must correspond to pad nets:

- `input_feedback_clock_net`  
Must be an input pad.
- `output_clock_net`  
Must be an output pad.

If attached to any other net, an error results.

## Constraint Values

- `input_feedback_clock_net`  
The input pad net used as the feedback to the DCM
- `value`  
The board trace delay calculated or measured by you
- `units`
  - ns (default)
  - ps
- `output_clock_net`  
The output pad net driven by the DCM

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF Syntax

```
NET output_clock_net FEEDBACK = value units NET input_feedback_clock_net ;
```

### XCF Syntax

```
BEGIN MODEL "entity_name"

    NET output_clock_net FEEDBACK = value
    units NET input_feedback_clock_net ;

END;
```

### PCF Syntax

```
{BEL|COMP} output_clock_net FEEDBACK = value units {BEL|COMP} input_feedback_clock_net ;
```

### Constraints Editor Syntax

For information on Constraints Editor and Constraints Editor syntax in ISE® Design Suite, see the ISE Design Suite Help.

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## File

The File (FILE) constraint tells NGCBuild where to find a file in another netlist.

When you instantiate a module that resides in another netlist, NGCBuild finds this file by looking up its file name. This requires the netlist to have the same name as a module that is defined in the file. To name the netlist differently than the module name, attach the File constraint to an instance declaration. This tells NGCBuild to look for the module in the specified file.

Some Xilinx® constraints cannot be used in attributes, because they are also VHDL keywords. Use a constraint alias in such cases. Each constraint has its own alias. The alias name is based on the original constraint name with a XIL prefix. For example, FILE cannot be used in attributes directly. You must use XIL\_FILE instead. The existing XILFILE alias is still supported.

## Architecture Support

Applies to all FPGA devices and all CPLD devices.

## Applicable Elements

The File constraint applies to instance declaration in which the definition is defined in the specified file.

## Propagation Rules

The File constraint is applicable only on instances.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
FILE

- Attribute Values  
*file\_name.extension*

*file\_name* is the name of a file that represents the underlying logic for the element carrying the constraint.

Example file types include:

- EDIF
- EDN
- NGC
- NMC

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute xilfile: string;
```

Specify the VHDL constraint as follows:

```
attribute xilfile of {instance_name | component_name} : {label | component} is "file_name";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* XIL_FILE = "file_name" *)
```

### UCF and NCF Syntax

```
INST <instance definition> FILE= <filename definition is located in>;
```

There is no valid syntax for a User Constraints File (UCF).

## Float

The Float (FLOAT) constraint:

- Is a basic mapping constraint.
- Allows tristated pads to float when not being driven.

This functionality is useful when the default termination for applicable I/O components is set to any of the following in ISE® Design Suite:

- [Pullup](#)
- [Pulldown](#)
- [Keeper](#)

## Architecture Support

- CoolRunner™ XPLA3
- CoolRunner-II

## Applicable Elements

The Float constraint applies to nets or pins.

## Propagation Rules

The Float constraint applies to the net or pin to which it is attached.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
FLOAT
- Attribute Values
  - TRUE
  - FALSE
  - None required. If attached, TRUE is assumed.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute FLOAT: string;
```

Specify the VHDL constraint as follows:

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

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
NET "signal_name" FLOAT;
```

### XCF Syntax

```
BEGIN MODEL "entity_name"  
    NET "signal_name" FLOAT;  
END;
```

## From Thru To

The From Thru To (FROM-THRU-TO) constraint:

- Is an advanced timing constraint.
- Is associated with the [Period](#) constraint of the high or low time.

From synchronous paths, a From Thru To constraint controls only the setup path, not the hold path.

The From Thru To constraint applies to a specific path that:

1. Begins at a source group.
2. Passes through intermediate point.
3. Ends at a destination group.

The source and destination groups can be either user or predefined groups. You must define an intermediate path using TPTHU before using THRU.

**Note** OFFSET Constraints do not allow predefined groups.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The From Thru To constraint applies to predefined and user-defined groups

## Propagation Rules

The From Thru To constraint applies to the specified FROM-THRU-TO path only.

## Constraint Values

- identifier  
Can consist of characters or underbars
- source\_group and destination\_group  
User-defined or predefined groups  
**Note** OFFSET Constraints do not allow predefined groups.
- thru\_pt1 and thru\_pt2  
Intermediate points to define specific paths for timing analysis
- value  
The delay time
- units
  - ps
  - ms
  - ns
  - micro

You are not required to have a FROM, THRU, or TO. You can have almost any combination, such as:

- FROM-TO
- FROM-THRU-TO
- THRU-TO
- TO
- FROM
- FROM-THRU-THRU-THRU-TO
- FROM-THRU

There is no restriction on the number of THRU points. The source, THRU points, and destination can be a:

- net
- bel
- comp
- macro
- pin
- timegroup

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

```
TIMESPEC "TSidentifier" =FROM "source_group" THRU "thru_pt1"...[THRU "thru_pt2" ...]
TO "destination_group" value [Units] [DATAPATHONLY];
```

The **DATAPATHONLY** keyword:

- Indicates that the [From To](#) constraint does not take clock skew or phase information into consideration.
- Results in only the data path between the groups being constrained and analyzed.

```
TIMESPEC TS_MY_PathB = FROM "my_src_grp" THRU "my_thru_pt" TO "my_dst_grp" 13.5 ns DATAPATHONLY;
```

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

### PCF Syntax

```
TSname=MAXDELAY FROM TIMEGRP "source" THRU TIMEGRP "thru_pt1" ...THRU "thru_ptn" TO
TIMEGRP "destination" [DATAPATHONLY];
```



## From To

The From To (FROM-TO) constraint:

- Defines a timing constraint between two groups.
- Is associated with the [Period](#) constraint of the high or low time.

A group can be user-defined or predefined.

From synchronous paths, a From To constraint controls only the setup path, not the hold path.

For Virtex®-5 devices, the From To constraint controls both setup and hold paths.

## Architecture Support

Applies to all FPGA devices and all CPLD devices.

## Applicable Elements

The From To constraint applies to predefined and user-defined groups.

## Propagation Rules

The From To constraint applies to a path specified between two groups.

## Constraint Values

- *TSname*  
Must always begin with **TS**. Any alphanumeric character or underscore may follow.
- group1  
The origin path
- group2  
The destination path
- value
  - ns (default)
  - MHz
  - Another timing specification such as:
    - ◆  $TS\_C2S/2$
    - ◆  $TS\_C2S*2$

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

```
TIMESPEC TSname=FROM "group1" TO "group2" value [DATAPATHONLY];
```

The **DATAPATHONLY** keyword

- Indicates that the From To constraint does not take clock skew or phase information into consideration.
- Results in only the data path between the groups being constrained and analyzed.

```
TIMESPEC TS_MY_PathA = FROM "my_src_grp" TO "my_dst_grp" 23.5 ns DATAPATHONLY;
```

### XCF Syntax

XST supports From To, except that the following are not supported:

- FROM-THRU-TO
- Linked Specification
- Pattern matching for predefined groups:

```
TIMESPEC TS_1 = FROM FFS(machine/*) TO FFS 2 ns;
```

### PCF Syntax

```
TSname=MAXDELAY FROM TIMEGRP "group1" TO TIMEGRP "group2" value [DATAPATHONLY];
```

You are not required to have a FROM, THRU, and TO. You can have almost any combination, such as:

- FROM-TO
- FROM-THRU-TO
- THRU-TO
- TO
- FROM
- FROM-THRU-THRU-THRU-TO
- FROM-THRU

There is no restriction on the number of THRU points. The source, THRU points, and destination can be any of the following:

- net
- bel
- comp
- macro
- pin
- timegroup

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

## PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## FSM Style

For information about the FSM Style (FSM\_STYLE) constraint, see the *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices (UG687)*.

## Hierarchical Block Name

The Hierarchical Block Name (HBLKNM) constraint:

- Is an advanced mapping constraint.
- Assigns hierarchical block names to logic elements and controls grouping in a flattened hierarchical design.

When elements on different levels of a hierarchical design carry the same block name, and the design is flattened, NGCBuild prefixes a hierarchical path name to the Hierarchical Block Name value.

Like the [Block Name](#) constraint, the Hierarchical Block Name constraint forces function generators and flip-flops into the same CLB. Symbols with the same Hierarchical Block Name constraint map into the same CLB when possible.

Using the Hierarchical Block Name constraint instead of the Block Name constraint has the advantage of adding hierarchy path names during translation. The same Hierarchical Block Name constraint and value can be used on elements within different instances of the same design element.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Hierarchical Block Name constraint may be used with an FPGA device in one or more of the following design elements, or categories of design elements. Not all devices support all elements. To see which design elements can be used with which device families, see the Xilinx® *Libraries Guides* for details. For more information, see the device [data sheet](#).

- Registers
- I/O elements and pads
- FMAP
- PULLUP
- ACLK
- GCLK
- BUFG
- BUFGS
- BUFGP
- ROM
- RAMS
- RAMD
- Carry logic primitives

You can also attach HBLKNM to the net connected to the pad component in a User Constraints File (UCF). NGCBuild transfers the constraint from the net to the pad instance in the Native Generic Database (NGD) file so that it can be processed by the mapper.

Use the following syntax:

```
NET "net_name" HBLKNM=property_value ;
```

## Propagation Rules

- When attached to a design element, the Hierarchical Block Name constraint is propagated to all applicable elements in the hierarchy within the design element.
- When attached to a NET, the Hierarchical Block Name constraint is propagated to PAD components only.

## Constraint Values

`block_name`

A valid block name for that type of symbol.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
`HBLKNM`
- Attribute Values  
See *Constraint Values* above.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute hblknm: string;
```

Specify the VHDL constraint as follows:

```
attribute hblknm  
of {entity_name | component_name | signal_name | label_name}: {entity | component | signal | label}  
is "block_name";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* HBLKNM = "block_name" *)
```

### UCF and NCF Syntax

```
NET "net_name" HBLKNM=property_value ;
```

```
INST "instance_name" HBLKNM=block_name ;
```

- **INST "\$I13245/*this\_fmap*" HBLKNM=*group1*;**  
Specifies that the element **this\_fmap** is put into the block named **group1**.
- **NET "*net1*" HBLKNM=\$COMP\_0;**  
Attaches HBLKNM to the pad connected to **net1**.

Elements with the *same* block name *are* placed in the same logic block if possible. Otherwise an error occurs.

Elements with *different* block names *are not* placed in the same logic block.

## HIODELAY Group

The HIODELAY Group (HIODELAY\_GROUP) constraint:

- Is a design implementation constraint.
- Groups a hierarchical set of IDELAY and IODELAY constraints with an IDELAYCTRL to enable automatic replication and placement of IDELAYCTRL.

For more information, see the IDELAYCTRL section of the device user guide.

### Architecture Support

- The HIODELAY Group constraint supports Virtex®-4 and Virtex-5 devices only.
- The HIODELAY Group constraint supports Virtex-4 devices only when using the Timing Driven Pack and Placement Option in MAP.

### Applicable Elements

The HIODELAY Group constraint applies IDELAY, IODELAY, and IDELAYCTRL primitive instantiations

### Propagation Rules

- The HIODELAY Group constraint can be attached only to a design element.
- It is illegal to attach the HIODELAY Group constraint to a net, signal, or pin.
- To merge two or more embedded HIODELAY Group constraints, see [MIODELAY Group](#).

### Constraint Values

group\_name

The name assigned to a set of IDELAY or IODELAY constraints and an IDELAYCTRL to uniquely define the group.

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute HIODELAY_GROUP: string;
```

Specify the VHDL constraint as follows:

```
attribute HIODELAY_GROUP of {component_name | label_name}: {component | label} is "group_name";
```

#### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* HIODELAY_GROUP = "group_name" *)
```

#### UCF and NCF Syntax

```
INST "instance_name" HIODELAY_GROUP = group_name ;
```

## Hierarchical Lookup Table Name

The Hierarchical Lookup Table Name (HLUTNM) constraint:

- Supports Virtex®-5 devices only.
- Controls the grouping of logical symbols into the LUT sites of Virtex-5 devices.
- Is a string value property that is applied to two qualified symbols.
- Must be applied uniquely to two symbols within a given level of hierarchy. These two symbols are implemented in a shared LUT site within a SLICE component.
- Is functionally similar to the [Hierarchical Block Name](#) constraint.

### Architecture Support

The Hierarchical Lookup Table Name constraint supports Virtex-5 devices only.

### Applicable Elements

The Hierarchical Lookup Table Name constraint can be applied to:

- Two symbols that:
  - Share a common hierarchy, and
  - Are unique within their level of hierarchy
- Two 5-input or smaller function generator symbols (LUT, SRL16) if the total number of unique input pins required for both symbols does not exceed 5 pins.
- A 6-input read-only function generator symbol (LUT6) in conjunction with a 5-input read-only symbol (LUT5) if:
  - The total number of unique input pins required for both symbols does not exceed 6 inputs, and
  - The lower 32 bits of the 6-input symbol programming matches all 32 bits of the 5-input symbol programming.

### Propagation Rules

The Hierarchical Lookup Table Name constraint can be applied to two symbols that:

- Share a common hierarchy, and
- Are unique within their level of hierarchy.

### Constraint Values

- `instance_name`  
The instance name of an instantiated LUT or LUTRAM.
- `string_value`  
Applied uniquely to two symbols within a given level of hierarchy.
  - There is no default value.
  - If the value is blank, the constraint is ignored.



## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid element or symbol type
- Attribute Name  
HLUTNM
- Attribute Value  
<user\_defined>

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute hlutnm: string;
```

Specify the VHDL constraint as follows:

```
attribute hlutnm of instance_name : label is "string_value ";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* HLUTNM = "string_value" *)
```

### UCF and NCF Syntax

```
INST "symbol_name" HLUTNM=string_value ;
```

### XCF Syntax

```
MODEL "symbol_name" hlutnm = string_value ;
```

## H Set

For information about H Set (H\_SET), see [HU Set](#).

## HU Set

The HU Set (HU\_SET) constraint:

- Is an advanced mapping constraint.
- Is defined by the design hierarchy.
- Allows you to specify a set name.
  - You can have only one [H Set](#) within a given hierarchical element.
  - By specifying set names, you can specify several HU Set sets.

NGCBuild hierarchically qualifies the name of the HU Set as it flattens the design and attaches the hierarchical names as prefixes.

### Differences Between HU Set and H Set

HU Set	H Set
Has an explicit user-defined and hierarchically qualified name for the set	Has only an implicit hierarchically qualified name generated by the design-flattening program
Starts with the symbols that are assigned the HU_SET constraint	Starts with the instantiating macro one level above the symbols with the RLOC constraints

For more information about set attributes, see [Relative Location \(RLOC\)](#).

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The HU Set constraint may be used with an FPGA device in one or more of the following design elements, or categories of design elements. Not all devices support all elements. To see which design elements can be used with which devices, see the *Libraries Guides*. For more information, see the device data sheet.

- Registers
- FMAP
- Macro Instance
- ROM
- RAMS
- RAMD
- MULT18X18S
- RAMB4\_Sm\_Sn
- RAMB4\_Sn
- RAMB16\_Sm\_Sn
- RAMB16\_Sn
- RAMB16
- DSP48

## Propagation Rules

This constraint is a design element constraint. Any attachment to a net is illegal.

## Constraint Values

set\_name

- The identifier for the set.
- Must be unique among all sets.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
HU\_SET
- Attribute Values  
set\_name

### VHDL Syntax

Declare the VHDL constraint 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 "set_name";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
( * HU_SET = "set_name" * )
```

### UCF and NCF Syntax

```
INST "instance_name" HU_SET=set_name ;
```

### UCF and NCF Syntax Example

```
INST "$1I3245/FF_1" HU_SET=heavy_set;
```

Assigns an instance of the register FF\_1 to a set named heavy\_set.

### XCF Syntax

```
MODEL "entity_name" hu_set={yes | no};
```

```
BEGIN MODEL "entity_name"
```

```
INST "instance_name" hu_set=yes;
```

```
END;
```

## Input Buffer Delay Value

The Input Buffer Delay Value (IBUF\_DELAY\_VALUE) constraint:

- Is a mapping constraint.
- Adds additional static delay to the input path of the FPGA array.
- Can be applied to any input or bi-directional signal that *does not* directly drive a clock or IOB (Input Output Block) register.

For information regarding the constraint of signals that *do* drive clock and IOB registers, see [IFD Delay Value](#).

### Architecture Support

- Spartan®-3A
- Spartan-3E

### Applicable Elements

The Input Buffer Delay Value constraint applies to any top-level I/O port.

### Constraint Values

The Input Buffer Delay Value constraint can be set to an integer value from 0 (zero) to 16 (sixteen).

- 0
  - Zero (0) is the default.
  - Zero (0) applies no additional delay to the input path.
- 1-16
  - A value of 1-16 correlates to a larger delay added to input path.
  - These values do not directly correlate to a unit of time, but rather to additional buffer delay.

For more information, see the device [data sheets](#).

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

- Attach a new property to the top-level port of the schematic
- Attribute Name  
IBUF\_DELAY\_VALUE
- Attribute Values  
See *Constraint Values* above.

#### VHDL Syntax

Attach a VHDL attribute to the appropriate top-level port.

```
attribute IBUF_DELAY_VALUE : string;  
  
attribute IBUF_DELAY_VALUE of top_level_port_name :  
signal is "value";
```

The following statement assigns an IBUF\_DELAY\_VALUE increment of 5 to the net DataIn1:

```
attribute IBUF_DELAY_VALUE : string;  
attribute IBUF_DELAY_VALUE of DataIn1: label is "5";
```

### Verilog Syntax

Attach a Verilog attribute to the appropriate top-level port.

```
(* IBUF_DELAY_VALUE="value" *) input top_level_port_name ;
```

The following statement assigns an IBUF\_DELAY\_VALUE increment of 5 to the net DataIn1.

```
(* IBUF_DELAY_VALUE="5" *) input DataIn1;
```

### UCF and NCF Syntax

```
NET "top_level_port_name" IBUF_DELAY_VALUE = value;
```

The following statement assigns an IBUF\_DELAY\_VALUE increment of 5 to the net DataIn1.

```
NET "DataIn1" IBUF_DELAY_VALUE = 5;
```

## IFD Delay Value

The IFD Delay Value (IFD\_DELAY\_VALUE) constraint:

- Is a mapping constraint.
- Adds additional static delay to the input path of the FPGA array.
- Can be applied to any input or bi-directional signal that drives an IOB (Input Output Block) register.

For information on constraining signals that do *not* drive IOB registers, see [Input Buffer Delay Value](#).

### Architecture Support

- Spartan®-3A
- Spartan-3E

### Applicable Elements

The IFD Delay Value constraint applies to any top-level I/O port.

### Propagation Rules

Although the IFD Delay Value constraint is attached to an I/O symbol, it applies to the entire I/O component.

### Constraint Values

The IFD Delay Value constraint can be set to:

- An integer value from 0-8
  - When IFD\_DELAY\_VALUE is set to 0, the data path has no additional delay added.
  - The integers 1-8 correspond to increasing amounts of delay added to the data path.

These values do not directly correlate to a unit of time, but rather to additional buffer delay.

For more information, see the device [data sheets](#).

- AUTO (default)
  - Adds the appropriate amount of delay to the data path.
  - Ensures that the input hold time of the destination register is met.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a net
- Attribute Name  
IFD\_DELAY\_VALUE
- Attribute Values
  - 0-8
  - AUTO

### VHDL Syntax

Attach a VHDL attribute to the appropriate top-level port.

```
attribute IFD_DELAY_VALUE : string;
```

The following statement assigns an IFD\_DELAY\_VALUE increment of 5 to the net DataIn1:

```
attribute IFD_DELAY_VALUE : string;  
attribute IFD_DELAY_VALUE of DataIn1: label is "5";
```

### Verilog Syntax

Attach a Verilog attribute to the appropriate top-level port.

```
(* IFD_DELAY_VALUE="value" *) input top_level_port_name ;
```

The following statement assigns an IFD\_DELAY\_VALUE increment of 5 to the net DataIn1:

```
(* IFD_DELAY_VALUE="5" *) input DataIn1;
```

### UCF and NCF Syntax

```
NET "top_level_port_name" IFD_DELAY_VALUE = value;  
value
```

The numerical IBUF delay setting

The following statement assigns an IFD\_DELAY\_VALUE increment of 5 to the net DataIn1:

```
NET "DataIn1" IFD_DELAY_VALUE = 5;
```



## In Term

The In Term (IN\_TERM) constraint:

- Is a basic mapping constraint.
- Sets a configuration of input termination resistors.
- Is valid:
  - On an input pad NET
  - On an input pad INST
  - For the entire design

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The In Term constraint may be used with an FPGA device in one or more of the following design elements, or categories of design elements:

- IOB input components (such as IBUF)
- Input Pad Net

Not all devices support all elements. To see which design elements can be used with which devices, see the *Libraries Guides*. For more information, see the device [data sheet](#).

## Propagation Rules

The In Term constraint is illegal when attached to a net or signal, except when the net or signal is connected to a pad. In this case, In Term is treated as attached to the pad instance.

## Constraint Values

- NONE
- TUNED\_SPLIT
- UNTUNED\_SPLIT\_25
- UNTUNED\_SPLIT\_50
- UNTUNED\_SPLIT\_75

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a pad net
- Attribute Name  
IN\_TERM
- Attribute Values  
See *Constraint Values* above.

## VHDL Syntax

Declare the VHDL constraint as follows:

```
Attribute IN_TERM: string;
```

Specify the VHDL constraint as follows:

```
attribute IN_TERM of signal_name: signal is
"{NONE|TUNED_SPLIT|UNTUNED_SPLIT_25|UNTUNED_SPLIT_50|UNTUNED_SPLIT_75}";
```

## Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* IN_TERM = "{NONE|TUNED_SPLIT|UNTUNED_SPLIT_25|UNTUNED_SPLIT_50|UNTUNED_SPLIT_75}" *)
```

## UCF and NCF Syntax

```
NET "pad_net_name"IN_TERM = "{NONE|TUNED_SPLIT|UNTUNED_SPLIT_25|UNTUNED_SPLIT_50|UNTUNED_SPLIT_75}" ;
```

Configures the I/O to use a PULLUP.

```
DEFAULT IN_TERM = TUNED_SPLIT;
```

Configures In Term to be used globally.

## XCF Syntax

```
BEGIN MODEL "entity_name "
```

```
NET "signal_name " in_term=tuned_split;
```

```
END;
```

## PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Input Registers

The Input Registers (INREG) constraint:

- Supports CoolRunner™ XPLA3 and CoolRunner-II devices only.
- Applies to:
  - Registers and latches with their D-inputs driven by input pads, or
  - The Q-output nets of such registers and latches.

Registers and latches for devices that have their D-inputs driven by input pads are automatically implemented (where possible) using the device Fast Input path.

You can disable the ISE® Design Suite **Use Fast Input** property for INREG for the Fit (Implement Design) process. If you do so, only registers and latches with the INREG attribute are considered for Fast Input optimization.

## Architecture Support

Supports CoolRunner™ XPLA3 and CoolRunner-II devices only.

## Applicable Elements

The Input Registers constraint applies to:

- Registers and latches with their D-inputs driven by input pads, or
- The Q-output nets of such registers and latches.

## Propagation Rules

The Input Registers constraint applies to:

- The register or latch to which it is attached , or
- The Q-output nets of such registers and latches.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a register, latch, or net
- Attribute Name  
INREG
- Attribute Values  
None (TRUE by default)

### UCF Syntax

```
NET "signal_name" INREG;
```

```
INST "register_name" INREG;
```

## Internal Vref Bank

The Internal Vref Bank (INTERNAL\_VREF\_BANK) constraint:

- Assigns a voltage value to the internal Vref feature for a given I/O bank.
- Frees the Vref pins of I/O banks from providing a voltage reference.
- Allows the Vref pins to be used for other purposes.

## Architecture Support

The Internal Vref Bank constraint applies to Virtex®-6, Kintex™-7, and Virtex®-7 devices

## Applicable Elements

The Internal Vref Bank constraint:

- Is a global CONFIG constraint.
- Is not attached to any instance or signal name.

## Propagation Rules

The Internal Vref Bank constraint applies to I/O components in the specified bank for the entire design

## Constraint Values

- *n*  
The number of the bank
- *v*  
The target voltage value
  - 0.0
  - 0.6
  - 0.675
  - 0.75
  - 0.9
  - 1.1
  - 1.25

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

```
CONFIG INTERNAL_VREF_BANKn=v;
```

### UCF and NCF Syntax Example

```
CONFIG INTERNAL_VREF_BANK5=1.1;
```

## IOB

The IOB (IOB) constraint:

- Is a basic mapping and synthesis constraint.
- Indicates which flip-flops and latches can be moved into the IOB/ILOGIC/OLOGIC.
- Has precedence over the mapper **-pr** command line option.
- Does NOT have precedence over [Location \(LOC\)](#) constraints.

The mapper supports the following command line option:

**-pr i|o|b|off**

This option allows flip-flop or latch primitives to be pushed into the following on a global scale:

- Input IOB (i)
- Output IOB (o)
- Input/output IOB (b)

The IOB constraint, when associated with a flip-flop or latch, tells the mapper to pack that instance into an IOB type component if possible.

The Xilinx® Synthesis Technology (XST) software:

- Considers the IOB constraint to be an implementation constraint.
- Propagates it in the generated NGC file.
- Duplicates the flip-flops and latches driving the Enable pin of output buffers, allowing the corresponding flip-flops and latches to be packed in the IOB.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

- Non-INFF and non-OUTFF flip-flop and latch primitives
- Registers

## Propagation Rules

Applies to the design element to which it is attached.

## Constraint Values

- **TRUE**  
Allows the flip-flop or latch to be pulled into an IOB.
- **FALSE**  
Indicates not to pull the flip-flop or latch into an IOB.
- **AUTO (XST only)**  
XST takes timing constraints into account and automatically decides to push or not to push flip-flops into IOB components
- **FORCE**
  - Requires that the flip-flop or latch be pulled into an IOB.
  - Displays an error message if the register:
    - ◆ Has I/O connections, and
    - ◆ Cannot be packed in the IOB

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a flip-flop or latch instance or to a register
- Attribute Name  
IOB

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute iob: string;
```

Specify the VHDL constraint as follows:

```
attribute iob  
of {component_name | entity_name | label_name | signal_name} : {component | entity | label | signal} is  
" {TRUE | FALSE | AUTO | FORCE }";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
INST "instance_name" IOB= {TRUE | FALSE | FORCE};
```

- **INST "foo/bar" IOB=TRUE;**  
Instructs the mapper to place the **foo/bar** instance into an IOB component.
- **NET "foo/bar" IOB=TRUE;**  
This syntax is NOT supported in the User Constraints File (UCF).
- **INST "foo/bar" IOB=TRUE;**  
This syntax IS supported in the UCF.

### XCF Syntax

```
BEGIN MODEL "entity_name"  
  
  NET "signal_name" iob={true|false|auto|force};  
  
  INST " instance_name " iob={true|false|auto|force};  
  
END;
```

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

## Input Output Block Delay

The Input Output Block Delay (IOBDELAY) constraint:

- Is a basic mapping constraint.
- Specifies how the input path delay elements in all devices are to be programmed.

There are two possible destinations for input signals:

- The local IOB input FF
- A load external to the IOB

Xilinx® devices allow a delay element to delay the signal going to one or both of these destinations.

The Input Output Block Delay constraint cannot be used concurrently with the [No Delay](#) constraint.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Input Output Block Delay constraint applies to any I/O symbol:

- I/O pad
- I/O buffer
- Input pad net

## Propagation Rules

Although the Input Output Block Delay constraint is attached to an I/O symbol, it applies to the entire I/O component.

## Constraint Values

- NONE
 

Sets the delay to OFF for both the IBUF and IFD paths.

  - The following statement sets the delay to OFF for the IBUF and IFD paths.

```
INST "xyzzzy" IOBDELAY=NONE
```

  - For Spartan®-3 devices, the default is not set to NONE. This allows the device to achieve a zero hold time.
- BOTH
 

Sets the delay to ON for both the IBUF and IFD paths.
- IBUF
  - Sets the delay to OFF for any register inside the I/O component.
  - Sets the delay to ON for the registers outside of the component if the input buffer drives a register D pin outside of the I/O component.
- IFD
  - Sets the delay to ON for any register inside the I/O component.
  - Sets the delay to OFF for the registers outside the component if a register occupies the input side of the I/O component, regardless of whether the register has the IOB=TRUE constraint.



## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to an I/O symbol
- Attribute Name  
IOBDELAY

### 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}";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
INST "instance_name" IOBDELAY={NONE|BOTH|IBUF|IFD};
```

## IODELAY Group

The IODELAY Group (IODELAY\_GROUP) constraint:

- Is a design implementation constraint.
- Groups a set of IDELAY and IODELAY constraints with an IDELAYCTRL to enable automatic replication and placement of IDELAYCTRL in a design.

For more information, see the IDELAYCTRL section of the appropriate device user guide.

### Limitations with LOC

- Use IODELAY Group only when replicating a single IDELAYCTRL to multiple banks, without a [Location \(LOC\)](#) constraint.
- Do not use IODELAY Group in conjunction with IDELAYCTRL instances that have a LOC constraint.
- Instantiate only one IDELAYCTRL.
- Do not apply a LOC constraint.
- Group any IODELAY constraint that needs an IDELAYCTRL into an IODELAY Group.
- Create one group for each bank.

### Architecture Support

IODELAY Group supports Virtex®-4, Virtex-5, Virtex-6, and 7 series devices.

- For Virtex-4 devices, IODELAY Group is supported only when using the Timing Driven Pack and Placement Option in MAP.
- While IODELAY Group is supported on Virtex-4 and Virtex-5 devices, it is not the recommended method for replicating IDELAYCTRL. For the recommended method, see the device user guide.
- Xilinx® recommends using the IODELAY Group method for replicating IDELAYCTRL primitives on Virtex-6 and 7 series devices.

### Applicable Elements

The IODELAY Group constraint applies to the following elements:

- IDELAY
- IODELAY
- IODELAYE1
- IDELAYE2
- IODELAYE2
- IDELAYCTRL

### Propagation Rules

- IODELAY Group can be attached to a design element only.
- It is illegal to attach IODELAY Group to a net, signal, or pin.
- To merge two or more embedded IODELAY Group constraints, see [MIODELAY Group](#).

## Constraint Values

*group\_name*

The name assigned to a set of IDELAY or IODELAY constraints and an IDELAYCTRL to uniquely define the group.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute IODELAY_GROUP: string;
```

Specify the VHDL constraint as follows:

```
attribute IODELAY_GROUP of {component_name | label_name}: {component|label} is "group_name";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
( * IODELAY_GROUP = "group_name" * )
```

### UCF Syntax

```
INST "instance_name" IODELAY_GROUP = group_name ;
```

## Input Output Standard

The Input Output Standard (IOSTANDARD) constraint:

- Is a basic mapping constraint.
- Is a synthesis constraint.

### Input Output Standard for FPGA Devices

Use the Input Output Standard constraint to assign an I/O standard to an I/O primitive.

All components with the Input Output Standard constraint must follow the same placement rules (banking rules) as the SelectIO™ components.

- For information on the banking rules for each architecture, see the Xilinx® *Libraries Guides*.
- For descriptions of the supported I/O standards, see the device [data sheet](#).

For Spartan®-3, Spartan-3A, Spartan-3E, Virtex®-4, and Virtex-5 devices:

- Xilinx recommends attaching the Input Output Standard constraint to a buffer component instead of using the SelectIO variants of a component.

For example, use an IBUF with IOSTANDARD=HSTL\_III instead of the IBUF\_HSTL\_III component.

- Differential signaling standards apply to IBUFDS, IBUFGDS, OBUFDS, and OBUFTDS only (not IBUF or OBUF).

### Input Output Standard for CPLD Devices

You can apply the Input Output Standard constraint to I/O pads of CoolRunner™-II devices to specify both input threshold and output VCCIO voltage. For supported values, see the device [data sheet](#).

The CPLD fitter automatically groups outputs with compatible IOSTANDARD settings into the same bank when no location constraints are specified.

### Architecture Support

- All FPGA devices
- CoolRunner-II CPLD devices

### Applicable Elements

To see which design elements can be used with which device families, see the *Libraries Guides*. For more information, see the device [data sheet](#).

- IBUF, IBUFG, OBUF, OBUFT
- IBUFDS, IBUFGDS, OBUFDS, OBUFTDS
- Output Voltage Banks

## Propagation Rules

- It is illegal to attach the Input Output Standard constraint to a net or signal except when the signal or net is connected to a pad.
- In this case, the Input Output Standard constraint is treated as attached to an IOB instance:
  - IBUF
  - OBUF
  - IOB FF
- When attached to a design element, the Input Output Standard constraint propagates to all applicable elements in the hierarchy within the design element.

## Constraint Values

iostandard\_name

An I/O Standard name as specified in the device [data sheet](#)

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to an I/O primitive
- Attribute Name  
IOSTANDARD
- Attribute Values  
iostandard\_name

For more information, see *UCF and NCF Syntax* below.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute iostandard: string;
```

Specify the VHDL constraint as follows:

```
attribute iostandard of {component_name | label_name}: {component | label} is "iostandard_name";
```

For more information, see *UCF and NCF Syntax* below.

For CPLD devices you can also apply the Input Output Standard constraint to the pad signal.

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

For a description of *iostandard\_name*, see the UCF section.

For CoolRunner-II CPLD devices, you can also apply the Input Output Standard constraint to the pad signal.

### UCF and NCF Syntax

```
INST " instance_name " IOSTANDARD= iostandard_name ;  
NET "pad_net_name " IOSTANDARD=iostandard_name ;
```

### XCF Syntax

```
BEGIN MODEL "entity_name "  
    INST "instance_name " iostandard=string ;  
    NET "signal_name " iostandard=string ;  
END;
```

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

### Pinout and Area Constraints Editor (PACE) Syntax

Pinout and Area Constraints Editor (PACE) is supported for CPLD devices only. PACE is NOT supported for FPGA devices.

Access PACE from:

**ISE® Design Suite > Processes**

Use PACE to:

- Assign location constraints to I/O components.
- Assign certain I/O properties such as I/O Standards.

For more information, see the PACE Help, especially the topics in:

**Procedures > Editing Pins and Areas**

## Keep

The Keep (KEEP) constraint:

- Is an advanced mapping constraint.
- Is a synthesis constraint.
- Prevents a net from being absorbed into a logic block.
- Is translated into an internal constraint known as NOMERGE when targeting an FPGA device.

Messages from the implementation software refer to the system property as NOMERGE, not Keep.

### Net Absorption into a Logic Block

When a design is mapped, some nets may be absorbed into logic blocks. When a net is absorbed into a logic block, it can no longer be seen in the physical design database. This may happen, for example, when the components connected to each side of a net are mapped into the same logic block. The net may then be absorbed into the logic block containing the components. The Keep constraint prevents the net from being absorbed.

### Architecture Support

Applies to all FPGA devices and all CPLD devices.

### Applicable Elements

Applies to signals.

### Propagation Rules

Applies to the signal to which it is attached.

### Constraint Values

- TRUE (or YES XCF only)
- FALSE (or NO XCF only)
- SOFT (XST only)
  - Instructs XST to preserve the designated net.
  - Prevents XST from attaching a NOMERGE constraint to this net in the synthesized netlist.

As a result, the net is preserved during synthesis, but implementation tools are given all freedom to handle it. Conceptually, you are specifying a KEEP=TRUE for synthesis only, but a KEEP=FALSE for implementation tools.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a net
- Attribute Name  
KEEP
- Attribute Values
  - TRUE
  - FALSE
  - SOFT (XST only)

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute keep : string;
```

Specify the VHDL constraint as follows:

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

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* KEEP = "{TRUE|FALSE|SOFT}" *)
```

### UCF and NCF Syntax

```
INST "instance_name" KEEP={TRUE|FALSE};
```

The following statement ensures that the net \$SIG\_0 remains visible.

```
NET "$1I3245/$SIG_0" KEEP;
```

### XCF Syntax

```
BEGIN MODEL "entity_name"
```

```
  NET "signal_name" keep={yes|no|true|false};
```

```
END;
```

In an XST Constraint File (XCF), you may optionally enclose the value of the Keep constraint in double quotes. Double quotes are mandatory for the SOFT value.

```
BEGIN MODEL "entity_name"
```

```
  NET "signal_name" keep="soft";
```

```
END;
```



## Keep Hierarchy

The Keep Hierarchy (KEEP\_HIERARCHY) constraint is a synthesis and implementation constraint.

If hierarchy is maintained during synthesis, the implementation software uses Keep Hierarchy to:

- Preserve the hierarchy throughout the implementation process.
- Allow a simulation netlist to be created with the desired hierarchy.

XST may flatten the design to get better results by optimizing entity or module boundaries. If you set Keep Hierarchy to **true**, the generated netlist is hierarchical and respects the hierarchy and interface of any entity or module.

This option is related to the hierarchical blocks (VHDL entities, Verilog modules) specified in the Hardware Description Language (HDL) design and does not concern the macros inferred by the HDL synthesizer.

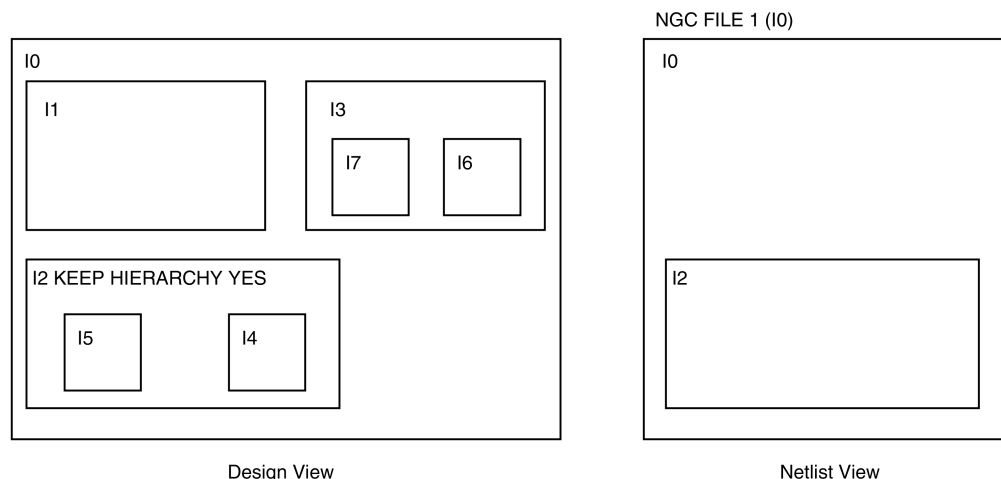
In general, an HDL design is a collection of hierarchical blocks. Preserving the hierarchy gives the advantage of fast processing because the optimization is done on separate pieces of reduced complexity. Nevertheless, very often, merging the hierarchy blocks improves the fitting results (fewer PTerms and device macrocells, better frequency) because the optimization processes (collapsing, factorization) are applied globally on the entire logic.

The Keep Hierarchy constraint enables or disables hierarchical flattening of user-defined design units. Allowed values are **true** and **false**. By default, the user hierarchy is preserved.

### Keep Hierarchy Constraint Example

In the following figure, if the Keep Hierarchy constraint is set to the entity or module **I2**:

- The hierarchy of I2 is in the final netlist.
- Its contents I4, I5 are flattened inside I2.
- I1, I3, I6, I7 are also flattened.



X9542

## Architecture Support

Applies to all FPGA devices and all CPLD devices.

## Applicable Elements

The Keep Hierarchy constraint is attached to logical blocks, including blocks of hierarchy or symbols.

## Propagation Rules

Applies to the entity, module, or signal to which it is attached.

## Constraint Values

- **true** (default for CPLD devices)  
Allows the preservation of the design hierarchy, as described in the HDL project. If this value is applied to synthesis, it is also propagated to implementation.
- **false** (default For FPGA devices)  
Hierarchical blocks are merged in the top level module.
- **soft**  
Allows the preservation of the design hierarchy in synthesis, but the Keep Hierarchy constraint is not propagated to implementation.

**Note** In XST, the Keep Hierarchy constraint can be set to the following values: **yes**, **true**, **no**, **false**, and **soft**. When used at the command line, only **yes**, **no**, and **soft** are accepted.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to the entity or module symbol
- Attribute Name  
KEEP\_HIERARCHY
- Attribute Values
  - TRUE
  - FALSE

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute keep_hierarchy : string;
```

Specify the VHDL constraint as follows:

```
attribute keep_hierarchy
of architecture_name : architecture is {TRUE|FALSE|SOFT};
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
INST "instance_name" KEEP_HIERARCHY={TRUE|FALSE};
```

### XCF Syntax

In XST, the Keep Hierarchy constraint accepts the following values:

- yes
- true
- no
- false
- soft

When the Keep Hierarchy constraint is used as a command-line switch, only **yes**, **no**, and **soft** are accepted.

```
MODEL "entity_name" keep_hierarchy={yes|no|soft};
```

### ISE Design Suite Syntax

Define globally with:

**ISE® Design Suite > Process > Properties > Synthesis Options > Keep Hierarchy**

With a design selected in the Sources window, select:

**Processes > Synthesize > Process Properties > Property display level > Advanced**

## Keeper

The Keeper (KEEPER) constraint:

- Is a basic mapping constraint.
- Retains the value of the output net to which it is attached.

For example, if logic 1 is being driven onto the net, Keeper drives a weak or resistive 1 onto the net. If the net driver is then tristated, Keeper continues to drive a weak or resistive 1 onto the net.

The Keeper constraint must follow the same banking rules as the Keeper component. For information on banking rules, see the Xilinx® *Libraries Guides*.

KEEPER, PULLUP, and PULLDOWN are valid only on pad NET components, not on INST components of any kind.

For CoolRunner™-II devices, KEEPER and PULLUP are mutually exclusive across the whole device.

## Architecture Support

- All FPGA devices
- CoolRunner-II CPLD devices.

## Applicable Elements

The Keeper constraint applies to tristate input and output pad nets.

## Propagation Rules

The Keeper constraint is illegal when attached to a net or signal, except when the net or signal is connected to a pad. In this case, the Keeper constraint is treated as attached to the pad instance.

## Constraint Values

- YES
- NO
- TRUE
- FALSE

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to an output pad net
- Attribute Name  
KEEPER
- Attribute Values
  - 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}";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

This statement configures the I/O to use KEEPER for a NET.

```
NET "pad_net_name" KEEPER;
```

This statement configures KEEPER to be used globally.

```
DEFAULT KEEPER = TRUE;
```

### XCF Syntax

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name" keeper={yes|no|true |false};
```

```
END;
```

## Location (LOC)

The Location (LOC) constraint:

- Is a basic placement constraint.
- Is a synthesis constraint.

For a more detailed discussion of the Location (LOC) constraint, see [Advanced Location \(LOC\) Description](#),

## Architecture Support

Applies to all FPGA devices and all CPLD devices.

## Applicable Elements

For information about which design elements can be used with which device families, see the *Libraries Guides*. For more information, see the device data sheet.

## Propagation Rules

- For all nets, the Location (LOC) constraint is illegal when attached to a net or signal except when the net or signal is connected to a pad. In this case, LOC is treated as attached to the pad instance.
- For CPLD nets, the Location (LOC) constraint attaches to all applicable elements that drive the net or signal.
- When attached to a design element, the Location (LOC) constraint propagates to all applicable elements in the hierarchy within the design element.

## Constraint Values

*location*

A legal location for the part type

## Syntax Examples

Following is the syntax for a single location:

```
INST "instance_name" LOC=location;
```

For examples of legal placement constraints for each type of logic element in FPGA designs, see Syntax for FPGA Devices for this constraint, and the [Relative Location \(RLOC\)](#) constraint. Logic elements include:

- Flip-Flops
- ROM
- RAM
- block RAM
- FMAP
- BUFT
- CLB
- IOB
- I/O
- Edge decoders
- Global buffers

For more examples, see [Advanced Location \(LOC\) Constraint Examples](#).

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to an instance
- Attribute Name  
LOC
- Attribute Values  
value

For valid values, see *Syntax for FPGA Devices* and *Syntax for CPLD Devices* for this constraint.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute loc: string;
```

Specify the VHDL constraint as follows:

```
attribute loc of {signal_name | label_name}: {signal | label} is "location";
```

Set the LOC constraint on a bus as follows:

```
attribute loc of bus_name : signal is " location_1  
location_2 location_3...";
```

To constrain only a portion of a bus (CPLD devices only), use the following syntax:

```
attribute loc of bus_name : signal is "* * location_1  
* location_2...";
```

For more information about *location*, see *Syntax for FPGA Devices* and *Syntax for CPLD Devices* for this constraint.

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* LOC = " location" *)
```

Set the LOC constraint on a bus as follows:

```
(* LOC = "location_1 location_2 location_3..." *)
```

To constrain only a portion of a bus (CPLD devices only), use the following syntax:

```
(* LOC = " * * location_1 location_2..." *)
```

For more information about *location*, see *Syntax for FPGA Devices* and *Syntax for CPLD Devices* for this constraint.

### UCF and NCF Syntax

- **INST** `"/FLIP_FLOPS/*" LOC=SLICE_X*Y8;`  
Place each instance found **FLIP\_FLOPS** in any CLB in column 8.
- **INST** `"MUXBUF_D0_OUT" LOC=P110;`  
Place an instantiation of **MUXBUF\_D0\_OUT** in IOB location **P110**.
- **NET** `"DATA<1>" LOC=P111;`  
Connect **NET DATA<1>** to the pad from IOB location **P111**.

### XCF Syntax

```
BEGIN MODEL " entity_name "

  PIN "signal_name" loc=string ;

  INST "instance_name" loc=string ;

END;
```

### PCF Syntax

LOC writes out a LOCATE constraint to the Physical Constraints File (PCF) file. For more information, see the [Locate \(LOCATE\)](#) constraint.

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

### PACE Syntax

PACE is mainly used to assign location constraints to I/O components. It can also be used to assign certain I/O properties such as I/O Standards. You can access PACE from the Processes window in the Project Navigator.

PACE is supported for CPLD devices only. It is not supported for FPGA devices.

For more information, see the PACE help, especially the topics within *Editing Pins and Areas* in the *Procedures* section.



## Locate

The Locate (LOCATE) constraint:

- Is a basic placement constraint.
- Specifies any one of the following:
  - A single location
  - Multiple single locations
  - A location range

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

- CLB
- IOB
- DCM
- Clock logic
- Macro

## Propagation Rules

- When attached to a *macro*, the Locate constraint propagates to all elements of the macro.
- When attached to a *primitive*, the Locate constraint applies to the entire primitive.

## Constraint Values

- `site_name`  
A component site:
  - A CLB location
  - or
  - An IOB location
- `site_item`
  - **SITE** "*site\_name*"
  - or
  - **SITEGRP** "*site\_group\_name*"
- *n* in LEVEL *n*:  
0, 1, 2, 3, or 4

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Single or Multiple Single Locations PCF Syntax

```
COMP "comp_name" LOCATE=[SOFT] "site_item1"... "site_itemn" [LEVEL n];  
COMPGRP "group_name" LOCATE=[SOFT] "site_item1"... "site_itemn" [LEVEL n];  
MACRO name LOCATE=[SOFT] "site_item1" "site_itemn" [LEVEL n];
```

### Range of Locations PCF Syntax

```
COMP "comp_name" LOCATE=[SOFT] SITE "site_name" : SITE "site_name" [LEVEL n];  
COMPGRP "group_name" LOCATE=[SOFT] SITE "site_name" : SITE "site_name" [LEVEL n];  
MACRO "macro_name" LOCATE=[SOFT] SITE "site_name" : SITE "site_name" [LEVEL n];
```

## Advanced Location (LOC) Description

The Location (LOC) constraint:

- Is a basic placement constraint.
- Is a synthesis constraint.

### LOC Description for FPGA Devices

The Location (LOC) constraint specifies the absolute placement of a design element on the FPGA die. It can be a single location, a range of locations, or a list of locations. You can specify LOC from the design file and also direct placement with statements in a constraints file.

To specify multiple locations for the same symbol, separate each location in the field with a comma. The comma specifies that the symbols can be placed in any of the specified locations. You can also specify an area in which to place a design element or group of design elements.

A convenient way to find legal site names is to use the PlanAhead™ software or FPGA Editor. The legal names are a function of the target part type. To find the correct syntax for specifying a target location, load an empty part into FPGA Editor. Place the cursor on any block, then click the block to display its location in the FPGA Editor history area. Do not include the pin name such as .I, .O, or .T as part of the location.

You can use LOC for logic that uses multiple CLB components, IOB components, soft macros, or other symbols. To do this, use LOC on a soft macro symbol, which passes the location information down to the logic on the lower level. The location restrictions are automatically applied to all blocks on the lower level for which LOC constraints are legal.

FPGA devices use a Cartesian-based XY designator at the slice level. The slice-based location specification uses the form:

**SLICE\_XmYn**

The XY slice grid starts as X0Y0 in the lower left CLB tile of the chip. The X values start at 0 and increase horizontally to the right in the CLB row, with two different X values per CLB. The Y values start at 0 and increase vertically up in the CLB column, with two different Y values per CLB.

For examples of how to specify the slices in the XY coordinate system, see Single LOC Constraint Examples below.

FPGA block RAM components and multipliers have their own specification different from the SLICE specifications. Therefore, the location value must start with SLICE, RAMB, or MULT.

- A block RAM located at RAMXB16\_X2Y3 is not located at the same site as a flip-flop located at SLICE\_X2Y3.
- A multiplier located at MULT18X18\_X2Y3 is not located at the same site as a flip-flop located at SLICE\_X2Y3 or at the same site as a block RAM located at RAMB16\_X2Y3.

The location values for global buffers (BUFG) and DCM elements is the specific physical site names for available locations.

Pin assignment using LOC is not supported for bus pad symbols such as OPAD8.

## Location Specification Types for FPGA Devices

Element Types	Location Examples	Meaning
IOB	P12	IOB location (chip carrier)
	A12	IOB location (pin grid)
	B, L, T, R	Applies to IOB components and indicates edge locations (bottom, left, top, right) for the following devices: Spartan®-3, Spartan-3A, Spartan-3E
	LB, RB, LT, RT, BR, TR, BL, TL	Applies to IOB components and indicates half edges (for example, left bottom, right bottom) for the following devices: Spartan-3, Spartan-3A, Spartan-3E
	Bank#	Applies to IOB components and indicates the bank for all FPGA devices
Slices	SLICE_X22Y3	SLICE_X22Y3 Slice location for all FPGA devices
Block RAM	RAMB16_X2Y56	Block RAM location for the following devices: Spartan®-3, Spartan-3A, Spartan-3Es
	RAMB36_X2Y56	Block RAM location for Virtex®-5 devices
Multipliers	MULT18X18_X#Y#	Multiplier location for Spartan-3 and Spartan-3A devices
	DSP48_X#Y#	Multiplier location for Virtex-4 and Virtex-5 devices
Digital Clock Manager	DCM_X#Y#	Digital Clock Manager for the following devices: Spartan-3, Spartan-3A, Spartan-3E
	DCM_ADV_X#Y#	Digital Clock Manager for Virtex-4 and Virtex-5 devices
Phase Lock Loop	PLL_ADV_X#Y#	Phase Lock Loop for all FPGA devices

The wildcard character (\*) can be used to replace a single location with a range as shown in the following example:

SLICE_X*Y5	Any slice of a FPGA device whose Y coordinate is 5
------------	--

The wildcard character for an FPGA global buffer, global pad, or DCM locations, is *not* supported.

## LOC Description for CPLD Devices

For CPLD devices, use the **LOC=pin\_name** constraint on a PAD symbol or pad net to assign the signal to a specific pin. The PAD symbols are:

- IPAD
- OPAD
- IOPAD
- UPAD

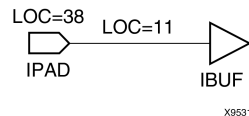
You can use the **LOC=FB $nn$**  constraint on any instance or its output net to assign the logic or register to a specific function block or macrocell, provided the instance is not collapsed.

The **LOC=FB  $nn\_mm$**  constraint on any internal instance or output pad assigns the corresponding logic to a specific function block or macrocell within the CPLD. If a LOC is placed on a symbol that does not get mapped to a macrocell or is otherwise removed through optimization, the LOC is ignored.

## LOC Priority

When specifying two adjacent LOC constraints on an input pad and its adjoining net, the LOC attached to the net has priority. In the following diagram, **LOC=11** takes priority over **LOC=38**.

### LOC Priority Example



## Advanced Location (LOC) Constraint Examples

This section includes the following advanced Location (LOC) constraint examples:

- Single LOC Constraint Examples
- Digital Clock Manager (DCM) LOC Constraint Examples
- Flip-Flop LOC Constraint Examples
- I/O LOC Constraint Examples
- IOB LOC Constraint Examples
- Mapping LOC Constraint Examples (FMAP)
- ROM LOC Constraint Examples
- Block RAM LOC Constraint Examples
- Slice LOC Constraint Examples

### Single LOC Constraint Examples

Constraint (UCF Syntax)	Device	Description
INST "instance_name" LOC=P12;		Place I/O at location P12.
INST "instance_name " LOC=SLICE_X3Y2;	Spartan-3, Spartan-3A, Spartan-3E, Virtex-4, and Virtex-5	Place logic in slice X3Y2 on the XY SLICE grid.
INST "instance_name " LOC=RAMB16_X0Y6;	Spartan-3, Spartan-3A, Spartan-3E, and Virtex-4	Place the logic in the block RAM located at RAMB16_X0Y6 on the XY RAMB grid.
INST "instance_name " LOC=MULT18X18_X0Y6;	Spartan-3 and Spartan-3A	Place the logic in the multiplier located at MULT18X18_X0Y6 on the XY MULT grid.
INST "instance_name " LOC=FIFO16_X0Y15;	Virtex-4	Place the logic in the FIFO located at FIFO16_X0Y15 on the XY FIFO grid.
INST "instance_name " LOC=IDELAYCTRL_X0Y3;	Virtex-4 and Virtex-5	Place the logic in the IDELAYCTRL located at the IDELAYCTRL_X0Y3 on the XY IDELAYCTRL grid.

Following is the syntax for multiple locations:

**LOC=** *location1,location2 ,...,locationn*

Separating each such constraint by a comma specifies multiple locations for an element. When you specify multiple locations, PAR can use any of the specified locations. Examples of multiple LOC constraints are provided in the following table.

### Multiple LOC Constraint Examples

Constraint	Device	Description
INST "instance_name " LOC=SLICE_X2Y10 , SLICE_X1Y10;	FPGA	Place the logic in SLICE_X2Y10 or in SLICE_X1Y10 on the XY SLICE grid.

Currently, using a single constraint there is no way to constrain multiple elements to a single location or multiple elements to multiple locations.

Following is the syntax for a range of locations:

**INST** " *instance\_name* " **LOC=***location* :*location* {**SOFT** };

You can define a range by specifying the two corners of a bounding box. Except for Spartan-3, Spartan-3A, Spartan-3E, Virtex-4, and Virtex-5 devices, specify the upper left and lower right corners of an area in which logic is to be placed. For FPGA devices, specify the lower left and upper right corners. Use a colon (:) to separate the two boundaries.

The logic represented by the symbol is placed somewhere inside the bounding box. The default is to interpret the constraint as a “hard” requirement and to place it within the box. If SOFT is specified, PAR may place the constraint elsewhere if better results can be obtained at a location outside the bounding box. Examples of LOC constraints used to specify a range are given in the following table.

### LOC Range Constraint Examples

Constraint	Device	Description
<pre>INST "instance_name " LOC=SLICE_X3Y5:SLICE_X5Y20;</pre>	FPGA	Place logic in any slice within the rectangular area bounded by SLICE_X3Y5 (the lower left corner) and SLICE_X5Y20 (the upper right corner) on the XY SLICE grid.

LOC ranges can be supplemented with the keyword SOFT. Unlike AREA\_GROUP, LOC ranges do not influence the packing of symbols. LOC range is strictly a placement constraint used by PAR.

Following is the LOC syntax for CPLD devices:

```
INST "instance_name " LOC=pin_name ;
```

or

```
INST " instance_name " LOC=FBff ;
```

or

```
INST "instance_name " LOC=FB ff_mm ;
```

- *pin\_name* is *Pnn* for numeric pin names or *rc* for row-column pin names
- *ff* is a function block number
- *mm* is a macrocell number within a function block

The two constraint formats for **FBff** and **FBff\_mm** are only applicable for outputs and bidirectional pins, not for inputs.

The first constraint format:

```
INST "instance_name" LOC=pin_name;
```

is applicable for all types of I/O.

### Digital Clock Manager (DCM) LOC Constraint Examples

This section applies to all FPGA devices.

You can lock the DCM in the UCF file. The syntax is as follows:

```
INST "instance_name " LOC = DCM_XYB;
```

All Spartan devices

```
INST "instance_name " LOC = DCM_ADV_XYB;
```

Virtex-4 and Virtex-5 devices

*A* is the X coordinate, starting with 0 at the left-hand bottom corner. *A* increases in value as you move across the device to the right.

*B* is the Y coordinate, starting with 0 at the left-hand bottom corner. *B* increases in value as you move up the device.

Example

```
INST "myinstance" LOC = DCM_X0Y0;
```

## Flip-Flop LOC Constraint Examples

Flip-flop constraints can be assigned from the schematic or through the UCF file.

From the schematic, attach LOC constraints to the target flip-flop. The constraints are then passed into the EDIF netlist and are read by PAR after the design is mapped.

The following examples show how the LOC constraint is applied to a schematic and to a UCF (User Constraints File). The instance names of two flip-flops, /top-12/fdrd and /top-54/fdsd, are used to show how you would enter the constraints in the UCF.

### Slice-Based XY Grid Designations

Spartan-3 devices and higher and Virtex-4 devices and higher are the only architectures that use slice-based XY grid designations.

Flip-flops can be constrained to a specific slice, a range of slices, a row or column of slices.

### Flip-Flop LOC Constraint Example One

Schematic	LOC=SLICE_X1Y5
UCF	INST "/top-12/fdrd" LOC=SLICE_X1Y5;

Places the flip-flop in SLICE\_X1Y5. SLICE\_X0Y0 is in the lower left corner of the device.

### Flip-Flop LOC Constraint Example Two

Place the flip-flop in the rectangular area bounded by the SLICE\_X1Y1 in the lower left corner and SLICE\_X5Y7 in the upper right corner.

Schematic	LOC=SLICE_R1C1:SLICE_R5C7
UCF	INST "/top-12/fdrd" LOC=SLICE_X1Y1:SLICE_X5Y7;

### Flip-Flop LOC Constraint Example Three

Place the flip-flops anywhere in the row of slices whose Y coordinate is 3. Use the wildcard (\*) character in place of either the X or Y value to specify an entire row (Y\*) or column (X\*) of slices.

Schematic	LOC=SLICE_X*Y3
UCF	INST "/top-12/fdrd/top-54/fdsd" LOC=SLICE_X*Y3;

### Flip-Flop LOC Constraint Example Four

Place the flip-flop in either SLICE\_X2Y4 or SLICE\_X7Y9.

Schematic	LOC=SLICE_X2Y4,SLICE_X7Y9
UCF	INST "/top-54/fdsd" LOC=SLICE_X2Y4, SLICE_X7Y9;



In Example Four, repeating the LOC constraint and separating each such constraint by a comma specifies multiple locations for an element. When you specify multiple locations, PAR can use any of the specified locations.

### Flip-Flop LOC Constraint Example Five

Do not place the flip-flop in the column of slices whose X coordinate is 5. Use the wildcard (\*) character in place of either the X or Y value to specify an entire row (Y\*) or column (X\*) of slices.

Schematic	PROHIBIT=SLICE_X5Y*
UCF	CONFIG PROHIBIT=SLICE_X5Y*;

## I/O LOC Constraint Examples

You can constrain I/Os to a specific IOB. You can assign I/O constraints from the schematic or through the UCF file.

From the schematic, attach LOC constraints to the target PAD symbol. The constraints are then passed into the netlist file and read by PAR after mapping.

Alternatively, in the UCF file a pad is identified by a unique instance name. The following example shows how the LOC constraint is applied to a schematic and to a UCF (User Constraints File). In the examples, the instance names of the I/Os are /top-102/data0\_pad and /top-117/q13\_pad. The example uses a pin number to lock to one pin.

Schematic	LOC=P17
UCF	INST "/top-102/data0_pad" LOC=P17;

Place the I/O in the IOB at pin 17. For pin grid arrays, a pin name such as B3 or T1 is used.

### IOB LOC Constraint Examples

You can assign I/O pads, buffers, and registers to an individual IOB location. IOB locations are identified by the corresponding package pin designation.

The following examples illustrate the format of IOB constraints. Specify LOC= and the pin location. If the target symbol represents a soft macro containing only I/O elements, for example, INFF8, the LOC constraint is applied to all I/O elements contained in that macro. If the indicated I/O elements do not fit into the specified locations, an error is generated.

The following UCF statement places the I/O element in location P13. For PGA packages, the letter-number designation is used, for example, B3.

**INST "instance\_name" LOC=P13;**

You can prohibit the mapper from using a specific IOB. You might take this step to keep user I/O signals away from semi-dedicated configuration pins. Such PROHIBIT constraints can be assigned only through the UCF file.

IOB components are prohibited by specifying a PROHIBIT constraint preceded by the CONFIG keyword, as shown in the following example.

Schematic	None
UCF	CONFIG PROHIBIT=p36, p37, p41;

Do not place user I/Os in the IOB components at pins 36, 37, or 41. For pin grid arrays, pin names such as D14, C16, or H15 are used.

## Mapping LOC Constraint Examples (FMAP)

Mapping constraints control the mapping of logic into CLB components. They have two parts. The first part is an FMAP component placed on the schematic. The second is a LOC constraint that can be placed on the schematic or in the constraints file.

FMAP controls the mapping of logic into function generators. This symbol does not define logic on the schematic; instead, it specifies how portions of logic shown elsewhere on the schematic should be mapped into a function generator.

The FMAP symbol defines mapping into a four-input (F) function generator.

For the FMAP symbol as with the CLBMAP primitive, MAP=PUC or PUO is supported, as well as the LOC constraint. (Currently, pin locking is not supported. MAP=PLC or PLO is translated into PUC and PUO, respectively.)

### Mapping Constraint (FMAP) Example One

Schematic	LOC=SLICE_X7Y3
UCF	INST "\$I323" LOC=SLICE_X2Y4, SLICE_X3Y4;

Places the FMAP symbol in the SLICE at row 7, column 3.

### Mapping Constraint (FMAP) Example Two

Schematic	LOC=SLICE_X2Y4, SLICE_X3Y4
UCF	INST "top/dec0011" LOC=CLB_R2C4,CLB_R3C4;

Places the FMAP symbol in either the SLICE at row 2, column 4 or the SLICE at row 3, column 4.

### Mapping Constraint (FMAP) Example Three

Schematic	LOC=SLICE_X5Y5:SLICE_X10Y8
UXCF	INST "\$I27" LOC=SLICE_X5Y5:SLICE_X10Y8;

Places the FMAP symbol in the area bounded by SLICE X5Y5 in the upper left corner and SLICE X10Y8 in the lower right

## Multiplier LOC Constraint Examples

This section applies to FPGA devices.

Multiplier constraints can be assigned from the schematic or through the UCF file. From the schematic, attach the LOC constraints to a multiplier symbol. The constraints are then passed into the netlist file and after mapping they are read by PAR. For more information on attaching LOC constraints, see the application user guide. Alternatively, in the constraints file a multiplier is identified by a unique instance name.

An FPGA multiplier has a different **XY** grid specification than slices and block RAM.

- Spartan-3, Spartan-3A, and Spartan-3E devices are specified using **MULT18X18\_X#Y#**
- Virtex-4 and Virtex-5 devices are specified using **DSP48\_X#Y#**, where the **X** and **Y** coordinate values correspond to the multiplier grid array.

A multiplier located at **MULT18X18\_X0Y1** is not located at the same site as a flip-flop located at **SLICE\_X0Y1** or a block RAM located at **RAMB16\_X0Y1**.

For example, assume you have a device with two columns of multipliers, each column containing two multipliers, where one column is on the right side of the chip and the other is on the left. The multiplier located in the lower left corner is **MULT18X18\_X0Y0**. Because there are only two columns of multipliers, the multiplier located in the upper right corner is **MULT18X18\_X1Y1**.

Schematic	LOC=MULT18X18_X0Y0
UCF	INST "/top-7/rq" LOC=MULT18X18_X0Y0;

### ROM LOC Constraint Examples

Memory constraints can be assigned from the schematic or through the UCF file.

From the schematic, attach the LOC constraints to the memory symbol. The constraints are then passed into the netlist file and after mapping they are read by PAR. For more information on attaching LOC constraints, see the application user guide.

Alternatively, in the constraints file memory is identified by a unique instance name. One or more memory instances of type ROM can be found in the input file. All memory macros larger than 16 x 1 or 32 x 1 are broken down into these basic elements in the netlist file.

In the following examples, the instance name of the ROM primitive is /top-7/rq.

#### Slice-Based XY Designations

Spartan-3 and higher and Virtex-4 and higher devices use slice-based XY grid designations. You can constrain a ROM to a specific slice, a range of slices, or a row or column of slices.

#### ROM LOC Constraint Example One

Schematic	LOC=SLICE_X1Y1
UCF	INST "/top-7/rq" LOC=SLICE_X1Y1;

Places the memory in the SLICE\_X1Y1. SLICE\_X1Y1 is in the lower left corner of the device. You can apply a single-SLICE constraint such as this only to a 16 x 1 or 32 x 1 memory.

#### ROM LOC Constraint Example Two

Schematic	LOC=SLICE_X2Y4, SLICE_X7Y9
UCF	INST "/top-7/rq" LOC=SLICE_X2Y4, SLICE_X7Y9;

Places the memory in either SLICE\_X2Y4 or SLICE\_X7Y9.

#### ROM LOC Constraint Example Three

Schematic	PROHIBIT SLICE_X5Y*
UCF	CONFIG PROHIBIT=SLICE_X5Y*;

Do not place the memory in column of slices whose X coordinate is 5. You can use the wildcard (\*) character in place of either the X or Y coordinate value in the SLICE name to specify an entire row (Y\*) or column (X\*) of slices.

### Block RAM LOC Constraint Examples

This section applies to FPGA devices

Block RAM constraints can be assigned from the schematic or through the UCF file. From the schematic, attach the LOC constraints to the block RAM symbol. The constraints are then passed into the netlist file. After mapping they are read by PAR. For more information on attaching LOC constraints, see the application user guide. Alternatively, in the constraints file a memory is identified by a unique instance name.

### Spartan-3 and Higher Devices

An FPGA block RAM has a different XY grid specification than a slice or multiplier. It is specified using RAMB16\_X $m$ Y $n$  where the X and Y coordinate values correspond to the block RAM grid array. A block RAM located at RAMB16\_X0Y1 is not located at the same site as a flip-flop located at SLICE\_X0Y1.

For example, assume you have a device with two columns of block RAM, each column containing two blocks, where one column is on the right side of the chip and the other is on the left. The block RAM located in the lower left corner is RAMB16\_X0Y0. Because there are only two columns of block RAM, the block located in the upper right corner is RAMB16\_X1Y1.

Schematic	LOC=RAMB16_X0Y0 (for all FPGA devices except Virtex-5 devices) LOC=RAMB36_X0Y0 (for Virtex-5 devices)
UCF	INST "/top-7/rq" LOC=RAMB16_X0Y0;

### Slice LOC Constraint Examples

This section applies to all FPGA devices. These are currently the only architectures that use the slice-based XY grid designations.

You can assign soft macros and flip-flops to a single slice location, a list of slice locations, or a rectangular block of slice locations.

Slice locations can be a fixed location or a range of locations. Use the following syntax to denote fixed locations.

**SLICE\_X $m$ Y $n$**

where

$m$  and  $n$  are the X and Y coordinate values, respectively

They must be less than or equal to the number of slices in the target device. Use the following syntax to denote a range of locations from the highest to the lowest.

**SLICE\_X  $m$ Y $n$ :SLICE\_X $m$ Y $n$**

### Format of Slice Constraints

The following examples illustrate the format of slice constraints: LOC= and the slice location. If the target symbol represents a soft macro, the LOC constraint is applied to all appropriate symbols (flip-flops, maps) contained in that macro. If the indicated logic does not fit into the specified blocks, an error is generated.

#### Slice Constraints Example One

The following UCF statement places logic in the designated slice.

**INST "instance\_name" LOC=SLICE\_X133Y10;**

#### Slice Constraints Example Two

The following UCF statement places logic within the first column of slices. The asterisk (\*) is a wildcard character

**INST "instance\_name" LOC=SLICE\_X0Y\*;**

#### Slice Constraints Example Three

The following UCF statement places logic in any of the three designated slices. There is no significance to the order of the LOC statements.

```
INST "instance_name" LOC=SLICE_X0Y3, SLICE_X67Y120, SLICE_X3Y0;
```

#### Slice Constraints Example Four

The following UCF statement places logic within the rectangular block defined by the first specified slice in the lower left corner and the second specified slice towards the upper right corner.

```
INST "instance_name" LOC=SLICE_X3Y22:SLICE_X10Y55;
```

## Slices Prohibited

You can prohibit PAR from using a specific slice, a range of slices, or a row or column of slices. Such prohibit constraints can be assigned only through the User Constraints File (UCF). Slices are prohibited by specifying a [Prohibit](#) constraint at the design level, as shown in the following examples.

#### Slices Prohibited Example One

Do not place any logic in the SLICE\_X0Y0. SLICE\_X0Y0 is at the lower left corner of the device.

Schematic	None
UCF	CONFIG PROHIBIT=SLICE_X0Y0;

#### Slices Prohibited Example Two

Do not place any logic in the rectangular area bounded by SLICE\_X2Y3 in the lower left corner and SLICE\_X10Y10 in the upper right.

Schematic	None
UCF	CONFIG PROHIBIT=SLICE_X2Y3:SLICE_X10Y10;

#### Slices Prohibited Example Three

Do not place any logic in a slice whose location has 3 as the X coordinate. This designates a column of prohibited slices. You can use the wildcard (\*) character in place of either the X or Y coordinate to specify an entire row (X\*) or column (Y\*) of slices.

Schematic	None
UCF	CONFIG PROHIBIT=SLICE_X3Y*;

#### Slices Prohibited Example Four

Do not place any logic in either SLICE\_X2Y4 or SLICE\_X7Y9.

Schematic	None
UCF	CONFIG PROHIBIT=SLICE_X2Y4, SLICE_X7Y9;

## Lock Pins

The Lock Pins (LOCK\_PINS) constraint:

- Instructs the implementation software to not swap the pins of the LUT symbol to which it is attached.
- Is distinct from the Lock Pins process in ISE® Design Suite, which is used to preserve the existing pinout of a CPLD design.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Lock Pins constraint applies only to specific instances of LUT symbols.

## Propagation Rules

The Lock Pins constraint applies only to a single LUT instance.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute lock_pins: string;
```

Specify the VHDL constraint as follows:

```
attribute lock_pins of {component_name/label_name} : {component|label} is "all";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* LOCK_PINS = "all" *)
```

### UCF and NCF Syntax

- Using No Designator

```
INST "XSYM1" LOCK_PINS;
```

- Using the ALL Attribute

```
INST "XSYM1" LOCK_PINS='ALL';
```

- Using a PIN Assignment List

```
INST I_589 LOCK_PINS=I0:A2;
```

```
INST I_894 LOCK_PINS=I3:A1,I2:A4;
```

```
INST tvAgy LOCK_PINS=I0:A4,I1:A3,I2:A2,I3:A1;
```

## Lookup Table Name

The Lookup Table Name (LUTNM) constraint:

- Allows you to control the grouping of logical symbols into the LUT sites of Virtex®-5 devices.
- Is a string value property that is applied to two qualified symbols.
- Must be applied uniquely to two symbols. These two symbols are implemented in a shared LUT site within a SLICE component.
- Is functionally similar to the [Block Name](#) constraint.

## Architecture Support

Virtex-5

## Applicable Elements

The Lookup Table Name constraint can be applied to:

- Two unique symbols.
- Two 5-input or smaller function generator symbols (LUT, ROM, or RAM) if the total number of unique input pins required for both symbols does not exceed 5 pins.
- A 6-input read-only function generator symbol (LUT6, ROM64) in conjunction with a 5-input read-only symbol (LUT5, ROM32) if:
  - The total number of unique input pins required for both symbols does not exceed 6 inputs, and
  - The lower 32 bits of the 6-input symbol programming matches all 32 bits of the 5-input symbol programming.

## Propagation Rules

The Lookup Table Name constraint can be applied to two unique symbols.

## Constraint Values

value

Any chosen name under which you want to group the two elements.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid element or symbol type
- Attribute Name  
LUTNM
- Attribute Value  
<user\_defined>

## VHDL Syntax

Before using the Lookup Table Name constraint, declare it with the following syntax placed after the architecture declaration, but before the begin statement in the top-level VHDL file:

```
attribute LUTNM: string;
```

Specify the VHDL constraint as follows:

```
attribute LUTNM of { LUT5_instance_name }: label is "value";
```

```
architecture MY_DESIGN of top is
attribute LUTNM: string;
    attribute LUTNM of LUT5_inst1: label is "logic_group1";
    attribute LUTNM of LUT5_inst2: label is "logic_group1";
begin
-- LUT5: 5-input Look-Up Table
-- Virtex-5
-- Xilinx HDL Libraries Guide version 8.2i
LUT5_inst1 : LUT5
    generic map (
        INIT => X"a49b44c1")
    port map (
        O => aout, -- LUT output (1-bit)
        I0 => d(0), -- LUT input (1-bit)
        I1 => d(1), -- LUT input (1-bit)
        I2 => d(2), -- LUT input (1-bit)
        I3 => d(3), -- LUT input (1-bit)
        I4 => d(4) -- LUT input (1-bit)
    );
-- End of LUT5_inst1 instantiation
-- LUT5: 5-input Look-Up Table
-- Virtex-5
-- Xilinx HDL Libraries Guide version 8.2i
LUT5_inst2 : LUT5
    generic map (
        INIT => X"649d610a")
    port map (
        O => bout, -- LUT output (1-bit)
        I0 => d(0), -- LUT input (1-bit)
        I1 => d(1), -- LUT input (1-bit)
        I2 => d(2), -- LUT input (1-bit)
        I3 => d(3), -- LUT input (1-bit)
        I4 => d(4) -- LUT input (1-bit)
    );
-- End of LUT5_inst2 instantiation
END MY_DESIGN;
```



## Verilog Syntax

Place the following attribute specification before the port declaration in the top-level Verilog code:

```
(* LUTNM = "value" *)

// LUT5: 5-input Look-Up Table
// Virtex-5
// Xilinx HDL Libraries Guide version 8.2i
(* LUTNM="logic_group1" *) LUT5 #(
  .INIT(32'ha49b44c1)
) LUT5_inst1 (
  .O(aout), // LUT output (1-bit)
  .I0(d[0]), // LUT input (1-bit)
  .I1(d[1]), // LUT input (1-bit)
  .I2(d[2]), // LUT input (1-bit)
  .I3(d[3]), // LUT input (1-bit)
  .I4(d[4]) // LUT input (1-bit)
);
// End of LUT5_inst1 instantiation
// LUT5: 5-input Look-Up Table
// Virtex-5
// Xilinx HDL Libraries Guide version 8.2i
(* LUTNM="logic_group1" *) LUT5 #(
  .INIT(32'h649d610a)
) LUT5_inst2 (
  .O(bout), // LUT output (1-bit)
  .I0(d[0]), // LUT input (1-bit)
  .I1(d[1]), // LUT input (1-bit)
  .I2(d[2]), // LUT input (1-bit)
  .I3(d[3]), // LUT input (1-bit)
  .I4(d[4]) // LUT input (1-bit)
);
// End of LUT5_inst2 instantiation
```

## UCF and NCF Syntax

Placed on the output, or bi-directional port:

```
INST "LUT5_instance_name" LUTNM="value";

INST "LUT5_inst1" LUTNM="logic_group1";

INST "LUT5_inst2" LUTNM="logic_group1";
```

## Map

The Map (MAP) constraint:

- Is an advanced mapping constraint.
- Is placed on an FMAP to specify whether pin swapping and the merging of other functions with the logic in the map are allowed.

If merging with other functions is allowed, other logic can also be placed within the CLB, if space allows.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Map constraint applies to FMAP.

## Propagation Rules

The Map constraint applies to the design element to which it is attached.

## Constraint Values

Value	CLB pins	CLB	Can the software can swap signals among the pins on the CLB?	Can the software add or remove logic from the CLB?
PUC	Unlocked (U)	Closed (C)	Yes	No
PUO (default)	Unlocked (U)	Open (O)	Yes	Yes
PLC	Locked (L)	Closed (C)	No	No
PLO	Locked (L)	Open (O)	No	Yes

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

```
INST " instance_name " MAP=[ PUC | PUO | PLC | PLO ] ;
```

Only PUC and PUO are observed. PLC and PLO are translated into PUC and PUO, respectively.

The following statement allows pin swapping, and ensures that no logic other than that defined by the original map is mapped into the function generators.

```
INST "$1I3245/map_of_the_world" map=puc;
```

## Mark Debug

The Mark Debug (MARK\_DEBUG) constraint:

- Is a synthesis constraint.
- Marks nets for debugging with the ChipScope™ software.

### Architecture Support

Applies to all FPGA devices and no CPLD devices.

### Applicable Elements

The Mark Debug constraint applies to the net to which it is attached.

### Propagation Rules

If the net is a bus, the Mark Debug constraint is propagated to the individual signals comprising the bus.

### Constraint Values

- true

The net is:

- Preserved from optimization.
- Marked for debugging with the ChipScope tool.

- false

The Mark Debug constraint is ignored.

- soft (XST only)

The net is marked for debugging *only if it is not optimized away during XST synthesis*.

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### VHDL Syntax Example

Declare the VHDL constraint as follows:

```
attribute mark_debug : string;
```

Specify the VHDL constraint as follows:

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

#### Verilog Syntax Example

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* mark_debug = "{TRUE|FALSE|SOFT}" *) wire wire_name;
```

### XCF Syntax Example

```
BEGIN MODEL "entity_name"  
  
    NET "signal_name" mark_debug = "{TRUE|FALSE|SOFT}" ;  
  
END;
```

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

Nets marked for debugging are automatically listed in the:

- ChipScope tool Unassigned Nets folder
- Set Up ChipScope Wizard

## Max Fanout

The Max Fanout (MAX\_FANOUT) constraint limits the fanout of nets or signals.

- Depending on the value of Max Fanout, both XST and MAP limit the fanout of a net when Max Fanout is applied.
- The value can either be an integer (XST only) or REDUCE (MAP only).

### Max Fanout for XST

Default integer values for XST are shown in the following table. Max Fanout is both a global and a local constraint in XST.

**Max Fanout Default Values**

Device	Default Value
Spartan®-3, Spartan-3E, Spartan-3A, Spartan-3A D	500
Virtex®-4	500
Virtex-5	100000 (One Hundred Thousand)

Large fanouts can cause routability problems. XST tries to limit fanout by duplicating gates or by inserting buffers. This limit is not a technology limit but a guide to XST. It may happen that this limit is not exactly respected, especially when this limit is small (less than 30).

In most cases, fanout control is performed by duplicating the gate driving the net with a large fanout. If the duplication cannot be performed, buffers are inserted. These buffers are protected against logic trimming at the implementation level by defining a [Keep](#) attribute in the NGC file. If the register replication option is set to **no**, only buffers are used to control fanout of flip-flops and latches.

Max Fanout is global for the design, but you can control maximum fanout independently for each entity or module or for given individual signals by using constraints.

If the actual net fanout is less than the Max Fanout value, XST behavior depends on how Max Fanout is specified.

- If the value of Max Fanout is set in ISE® Design Suite in the command line, or is attached to a specific hierarchical block, XST interprets its value as a guidance.
- If Max Fanout is attached to a specific net, XST does not perform logic replication. Putting Max Fanout on the net may prevent XST from having better timing optimization.

For example, suppose that the critical path goes through the net, which actual fanout is 80 and set Max Fanout value to 100. If Max Fanout is specified in ISE Design Suite, XST may replicate it, trying to improve timing. If Max Fanout is attached to the net itself, XST does not perform logic replication.

### Max Fanout for MAP

Max Fanout can drive MAP to limit fanout by duplicating registers, gates, or both. For this to occur:

- The MAP register duplication option must be enabled.
- Max Fanout constraints must be applied locally to nets.

When used during MAP, only the value of REDUCE is accepted.

- When **MAX\_FANOUT** = **"REDUCE"**, MAP limits fanout if it determines that it can provide an improvement in performance without causing problems in fitting the design.
- Review the MAP physical synthesis report (PSR) to see if **MAX\_FANOUT** = **"REDUCE"** actually reduced fanout.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

- When the value is an integer, Max Fanout applies globally, or to a VHDL entity, a Verilog module, or signal.
- When the value is REDUCE, Max Fanout applies only to a signal.

## Propagation Rules

Applies to the entity, module, or signal to which it is attached.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute max_fanout: string;
```

Specify the VHDL constraint as follows:

```
attribute max_fanout of {signal_name|entity_name}: {signal|entity} is "integer";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* max_fanout = "integer" *)
```

### XCF Syntax Example One

```
MODEL "entity_name" max_fanout=integer;
```

### XCF Syntax Example Two

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name" max_fanout=integer;
```

```
END;
```

### XST Command Line Syntax

Define globally with the **-max\_fanout** command line option of the **run** command:

```
-max_fanout integer
```

### ISE Design Suite Syntax

Define globally in ISE Design Suite in:

**Process > Properties > Xilinx-Specific Options > Max Fanout**

### UCF Syntax

When used with the MAP Register Duplication option, specify Max Fanout in the User Constraints File (UCF) as follows:

```
NET "signal_name" max_fanout=REDUCE;
```

## Maximum Delay

The Maximum Delay (MAXDELAY) constraint defines the maximum allowable delay on a net.

### Architecture Support

Applies to all FPGA devices and no CPLD devices.

### Applicable Elements

Applies to the net to which it is attached.

### Propagation Rules

The Maximum Delay constraint applies to the net to which it is attached.

### Constraint Values

- value  
Any positive integer
- units
  - ps
  - ns (default)
  - micro
  - ms
  - GHz
  - MHz
  - kHz

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

- Attach to a net
- Attribute Name: MAXDELAY
- Attribute Values
  - value  
Numerical time delay
  - units
    - ◆ micro
    - ◆ ms
    - ◆ ns
    - ◆ ps



### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute maxdelay: string;
```

Specify the VHDL constraint as follows:

```
attribute maxdelay of signal_name : signal is "value [units]";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(*MAXDELAY = "value [units]" *)
```

### UCF and NCF Syntax

```
NET "net_name" MAXDELAY=value units;
```

The following statement assigns a maximum delay of 10 nanoseconds to the net \$SIG\_4.

```
NET "$1I3245/$SIG_4" MAXDELAY=10 ns;
```

### PCF Syntax

```
item MAXDELAY = maxvalue [PRIORITY integer];
```

- item
  - ALLNETS
  - NET *name*
  - TIMEGRP *name*
  - ALLPATHS
  - PATH *name*
  - *path specification*
- maxvalue
  - Numerical time value with units of micro, ms, ps, or ns
  - Numerical frequency value with units of GHz, MHz, or KHz
  - TSidentifier

### Constraints Editor Syntax

To open Constraints Editor, select:

ISE® Design Suite > Processes > User Constraints > Exceptions > Timing Constraints > Nets

### FPGA Editor Syntax

To set Maximum Delay to all paths or nets, select:

File > Main Properties > Global Physical Constraints

To set Maximum Delay to a selected path or net, with a routed net selected, select:

Edit > Properties of Selected Items > Physical Constraints

## Maximum Product Terms

The Maximum Product Terms (MAXPT) constraint:

- Is an advanced constraint.
- Supports CPLD devices only.
- Specifies the maximum number of product terms the fitter is permitted to use when collapsing logic into the node to which Maximum Product Terms is applied.
- Overrides the Collapsing P-term Limit setting in ISE® Design Suite for the attached node.

## Architecture Support

Supports CPLD devices only. Does not support FPGA devices.

## Applicable Elements

Applies to signals.

## Propagation Rules

Applies to the signal to which it is attached.

## Constraint Values

integer

Any positive integer

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute maxpt: integer;
```

Specify the VHDL constraint as follows:

```
attribute maxpt of signal_name : signal is "integer";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* MAXPT = "integer" *)
```

### UCF and NCF Syntax

```
Net "signal_name" maxpt=integer;
```

## Maximum Skew

The Maximum Skew (MAXSKEW) constraint:

- Is a timing constraint.
- Controls the maximum amount of skew on a net.
- Controls the skew of:
  - Local clocks.
  - Clocks that are not on the global clock network.
- Is not required for global clock networks.

Xilinx® does not recommend using the Maximum Skew constraint for global clock networks.

## Skew

*Skew* is the difference between the delays of all loads driven by the net.

Because the Maximum Skew constraint identifies all loads driven by the net, skew may be reported between loads that have no logical connection.

To control the maximum allowable skew on a net, attach the Maximum Skew constraint directly to the net.

For a Maximum Skew Example, see the *Timing Closure User Guide (UG612)*

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Maximum Skew constraint applies to nets.

## Propagation Rules

Applies to the net to which it is attached.

## Constraint Values

- allowable\_skew  
The timing requirement
- units
  - ms
  - micro
  - ns (default)
  - ps

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a net
- Attribute Name  
MAXSKEW
- Attribute Values  
See *Constraint Values* above.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute maxskew: string;
```

Specify the VHDL constraint as follows:

```
attribute maxskew of signal_name : signal is  
"allowable_skew [units]";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* MAXSKEW = "allowable_skew [units] " *)
```

### UCF and NCF Syntax

```
NET " net_name " MAXSKEW=allowable_skew [units];
```

The following statement specifies a maximum skew of 3 ns on net \$SIG\_6.

```
NET "$1I3245/$SIG_6" MAXSKEW=3 ns;
```

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

### FPGA Editor Syntax

To set constraints in FPGA Editor, select:

**Edit > Properties of Selected Items**

With a routed net selected, you can set MAXSKEW from the Physical Constraints tab.

## MCB Performance

The MCB Performance (MCB\_PERFORMANCE) constraint:

- Applies to Spartan®-6 devices only.
- Is supported in the User Constraints File (UCF) only.
- Specifies the desired Memory Controller Block (MCB) performance level.

### Spartan-6 Memory Controller Block (MCB)

- The Memory Controller Block (MCB) in Spartan-6 devices supports two performance targets depending on voltage settings and conditions on the  $V_{CCINT}$  power supply.
- The performance targets are listed in the device [data sheet](#).
- To specify the MCB performance target in ISE® Design Suite, place the MCB Performance constraint in the User Constraints File (UCF).

### VCCINT Voltage Settings

- $V_{CCINT}$  voltage settings in the UCF and timing tools, and as reported by the ISE Design Suite tools, are independent of the MCB Performance setting.
- The software may report a different voltage for analysis than required by the MCB Performance setting.
- The Voltage requirements of this rail must be set based on both this attribute and the Voltage settings for timing analysis.

### Architecture Support

The MCB Performance constraint applies to Spartan-6 devices only.

### Propagation Rules

Not applicable.

### Constraint Values

- None  
If MCB Performance is not specified, the default is STANDARD.
- STANDARD  
To target the MCB to normal performance and the full voltage range on  $V_{CCINT}$  as shown in the device [data sheet](#), specify STANDARD in the UCF.  
**CONFIG MCB\_PERFORMANCE= STANDARD;**
- EXTENDED
  - To target the MCB to a faster performance, specify EXTENDED in the UCF.  
**CONFIG MCB\_PERFORMANCE=EXTENDED;**
  - There are explicit voltage requirements when using EXTENDED.
  - For more information, see the device [data sheet](#).

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF Syntax

```
CONFIG MCB_PERFORMANCE=[ STANDARD | EXTENDED ] ;
```

## MIODELAY Group

The MIODELAY Group (MIODELAY\_GROUP) constraint:

- Is a design implementation constraint.
- Combines two or more [IODELAY Group](#) constraints into a single Master IODELAY Group.
- Enables automatic replication and placement of IDELAYCTRL constraints.

### Architecture Support

- Supports Virtex®-4 and Virtex-5 devices only.
- Supports Virtex-4 devices only when using the Timing Driven Pack and Placement Option in MAP.

### Applicable Elements

The MIODELAY Group constraint is applied to two or more defined [IODELAY Group](#) constraints.

### Propagation Rules

- The MIODELAY Group constraint:
  - Is applied to an existing [IODELAY Group](#).
  - Is propagated to all design elements belonging to the original IODELAY\_GROUP.
- It is illegal to attach the MIODELAY Group constraint to a net, signal, or pin.

### Constraint Values

- master\_group\_name
  - Represents the master group being defined.
  - Contains all the elements in iodelay\_group1 and iodelay\_group2.
- iodelay\_group1 and iodelay\_group2  
Predefined IODELAY groups

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### UCF Syntax

```
MIODELAY_GROUP "master_group_name" = iodelay_group1 iodelay_group2 ... ;
```

## No Delay

The No Delay (NODELAY) constraint:

- Is an advanced mapping constraint.
- Removes input delay.
- Can be attached to:
  - I/O symbols
  - Special function access symbols:
    - ◆ TDI
    - ◆ TMS
    - ◆ TCK

## Removing Input Delay

The default configuration of IOB flip-flops includes an input delay that results in an external hold time on the input data path. To remove this delay, place the No Delay constraint on input flip-flops or latches. This results in a smaller setup time, but a positive hold time.

The input delay element is active in the default configuration for Spartan®-3, Spartan-3A, and Spartan-3E devices.

**IOBDELAY=NONE** is the preferred method of applying No Delay.

For more information see [Input Output Block Delay](#).

## Architecture Support

- Spartan-3
- Spartan-3A
- Spartan-3E

## Applicable Elements

- The No Delay constraint applies to input registers.
- You can also attach No Delay to a net connected to a pad component in a User Constraints File (UCF).

NGDBuild transfers the constraint from the net to the pad instance in the NGD file so that it can be processed by the mapper.

- Use the following UCF syntax:

```
NET " net_name " NODELAY;
```

## Propagation Rules

- The No Delay constraint is illegal when attached to a net or signal except when the net or signal is connected to a pad. In this case, No Delay is treated as attached to the pad instance.
- When attached to a design element, No Delay is propagated to all applicable elements in the hierarchy within the design element.



## Constraint Values

- TRUE
- FALSE

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
NODELAY
- Attribute Values  
See *Constraint Values* above.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute nodelay: string;
```

Specify the VHDL constraint as follows:

```
attribute nodelay of {component_name|signal_name|label_name} : {component|signal|label}  
is "{TRUE|FALSE}";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

- **INST "\$1I87/inreg67" NODELAY;**  
No input delay for IOB register **inreg67**.
- **NET "net1" NODELAY;**  
No input delay for the pad attached to **net1**.

### XCF Syntax

```
BEGIN MODEL "entity_name "  
  
  NET "signal_name" nodelay=true;  
  
  INST "instance_name" nodelay=true;  
  
END;
```

## No Reduce

The No Reduce (NOREDUCE) constraint:

- Is a fitter and synthesis constraint.
- Prevents minimization of redundant logic terms that are typically included to avoid logic hazards or race conditions.
- Identifies the output node of a combinatorial feedback loop to ensure correct mapping.

When constructing combinatorial feedback latches in a design:

- Always apply the No Reduce constraint to the latch output net.
- Include redundant logic terms when necessary to avoid race conditions.

## Architecture Support

Supports CPLD devices only. Does not support FPGA devices.

## Applicable Elements

Applies to the net to which it is attached.

## Propagation Rules

This constraint is a net constraint. Any attachment to a design element is illegal.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a net
- Attribute Name  
NOREDUCE
- Attribute Values
  - TRUE
  - FALSE

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute NOREDUCE: string;
```

Specify the VHDL constraint as follows:

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

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

The following statement specifies that there be no Boolean logic reduction or logic collapse from the net named \$SIG\_12 forward.

```
NET "$SIG_12" NOREDUCE;
```

### XCF Syntax

```
BEGIN MODEL entity_name
```

```
    NET signal_name noreduce={true|false};
```

```
END;
```

## Offset In

**Note** Offset Constraints do not allow predefined groups.

The Offset In (OFFSET IN) constraint:

- Specifies the timing requirements of an input interface to the FPGA device.
- Specifies the clock and data timing relationship at the external pads of the FPGA device.

An Offset In constraint specification checks the setup and hold timing requirements of all synchronous elements associated with the constraint.

The Offset In constraint is specified using a clock net name. The clock net associated with the Offset In constraint is the external clock pad. Because the constraint specifies the clock and data relationship at the external pads of the FPGA, the Offset In constraint cannot be specified using an internal clock net. However, the Offset In constraint automatically accounts for any phase or delay adjustments on the clock path due to components such as the DCM, PLL, MMCM, or IDELAY when analyzing the setup and hold timing requirements at the capturing synchronous element. In addition, the constraint propagates through the clock network and automatically applies to all clocks derived from the original external clock.

The Offset In constraint is global in scope by default. In the global Offset In constraint, all synchronous elements that are clocked by the specified clock net, and capture external data, are covered by the constraint. The scope of the synchronous elements covered by the constraint can be restricted by specifying time groups on a subset of input data pads, a subset of the capturing synchronous elements, or both.

For more information, see the *Timing Closure User Guide* (UG612).

## Architecture Support

Applies to all FPGA devices and all CPLD devices.

## Applicable Elements

- Global
- Net-Specific
- Pad Time Group

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

**Note** Although User Constraints File (UCF) examples are given below, Xilinx® recommends specifying the Offset In constraint using Constraints Editor.

### Global Method

The global method is the default Offset In constraint. The global Offset In constraint applies to all synchronous elements that capture incoming data and are triggered by the specified clock signal.

### Global Method UCF Syntax Example

```
OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]] {BEFORE|AFTER}
"clk_name" [{RISING|FALLING}];
```

## Global Method PCF Syntax Example

```
OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]] {BEFORE|AFTER}
COMP "clk_iob_name" [{RISING|FALLING}];
```

- offset\_time [units]
  - The difference in time between: 1) the capturing clock edge, and 2) the start of the data to be captured.
  - The time can be specified with or without explicitly declaring the units.
  - If no units are specified, the default value is nanoseconds (ns).
  - The valid values are:
    - ◆ ps
    - ◆ ns
    - ◆ micro
    - ◆ ms
- VALID <datavalid\_time> [UNITS]
  - The valid duration of the data to be captured.
  - This field is required for a hold time verification of the input interface.
  - The value can be specified with or without explicitly declaring the units.
  - If no units are specified, the default value is nanoseconds (ns).
  - The valid values are:
    - ◆ ps
    - ◆ ns
    - ◆ micro
    - ◆ ms
- BEFORE|AFTER
  - Defines the timing relationship of the start of data to the clock edge.
  - The best method of defining the clock and data relationship is to use the BEFORE option.
  - BEFORE describes the time the data begins to be valid relative to the capturing clock edge.
    - ◆ Positive values of BEFORE indicate the data begins *before* the capturing clock edge.
    - ◆ Negative values of BEFORE indicate the data begins *after* the capturing clock edge.
  - OFFSET = IN can be used with the AFTER option only if the **RISING** or **FALLING** qualifiers are not used.
- clk\_name
 

Defines the fully hierarchical name of the input clock pad net.
- RISING|FALLING
  - Optional keywords used to define the capturing clock edge in which the clock and data relationship is specified against.
  - The **RISING|FALLING** keywords automatically partition rising and falling edge registers in dual data rate (DDR) interfaces into separate groups for analysis.
  - The **RISING|FALLING** keywords can be used only with the BEFORE type of Offset In constraints.

## Input Group Method

When a group of inputs captured by the same clock have a shared timing requirement, the inputs can be grouped together to create a single timing constraint. The inputs can be grouped together by input signal names using pad groups, or by the synchronous elements using register groups. By grouping separate signals together into a single time group, the memory and runtime of the implementation tools is reduced. In addition, the timing report will contain bus-based skew and clock centering information.

## Input Group Method UCF Syntax Example

```
[TIMEGRP "pad_groupname"] OFFSET = IN "offset_time" [units]
[VALID <datavalid_time> [UNITS]] {AFTER "clk_name" [TIMEGRP "reg_groupname"] |
BEFORE "clk_name" [TIMEGRP "reg_groupname"] [{RISING|FALLING}]};
```

## Input Group Method PCF Syntax Example

```
[TIMEGRP "inputpad_grpname"] OFFSET = IN "offset_time" [units] [VALID <datavalid_time>
[UNITS]] {AFTER COMP "clk_iob_name" [TIMEGRP "reg_groupname"] | BEFORE
COMP "clk_iob_name" [TIMEGRP "reg_groupname"] [{RISING|FALLING}]};
```

- [TIMEGRP "pad\_groupname"]

The optional input pad time group. This time group can be used to limit the scope of the Offset In constraint to only the synchronous elements fed by the input pad nets contained in the timegroup.

- [TIMEGRP "reg\_groupname"]

The optional synchronous element time group. This time group can be used to limit the scope of the Offset In constraint to only the synchronous elements which capture input data with the specified clock and are contained in the time group.

## Net Specific Method

Offset In can also be used to specify an input constraint for a specific data net in a schematic, a specific input pad net in the UCF, or a specific input component in the PCF file.

### Schematic Syntax When Attached to a Net

```
OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]] {BEFORE|AFTER}
"clk_name" [TIMEGRP "reg_groupname"] [{RISING|FALLING}];
```

## Net Specific Method UCF Syntax Example

```
NET "pad_net_name" OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]]
{BEFORE|AFTER} "clk_name" [TIMEGRP "reg_groupname"] [{RISING|FALLING}];
```

## Net Specific Method PCF Syntax Example

```
COMP "pad_net_name" OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]] {BEFORE|AFTER}
COMP "clk_iob_name" [TIMEGRP "reg_groupname"] [{RISING|FALLING}];
```

- pad\_net\_name

The name of the input data net attached to the pad.

For the definition of the other variables and keywords, see *Global Method* above.

- The PCF specification uses I/O Blocks (COMP) instead of NET.

If the IOB COMP name is omitted in the PCF, or the NET name is omitted in the UCF, the Offset In specification is assumed to be global.

### Schematic Syntax

- Attach to a specific net
- Attribute Name  
OFFSET
- Attribute Values
  - IN|OUT
  - BEFORE|AFTER *clk\_pad\_netname*

### XCF Syntax

The XCF syntax is the same as the UCF syntax, except that the XCF syntax supports only the OFFSET IN BEFORE method.

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

## Offset Out

**Note** Offset Constraints do not allow predefined groups.

The Offset Out (OFFSET OUT) constraint:

- Specifies the timing requirements of an output interface from the FPGA device.
- Specifies the time from the clock edge at the input pin of the FPGA device until data becomes valid at the output pin of the FPGA device.
- Is specified using a clock net name.

The clock net associated with Offset Out is the external clock pad. Because the constraint specifies the time from the clock edge at the input pin of the FPGA device to the data at the output pin of the FPGA device, Offset Out cannot be specified using an internal clock net. However, Offset Out automatically accounts for any phase or delay adjustments on the clock path due to components such as the DCM, PLL, MMCM, or IDELAY when analyzing the output timing requirements. In addition, the constraint propagates through the clock network and automatically applies to all clocks derived from the original external clock.

Offset Out is global in scope by default. In the global Offset Out, all synchronous elements that are clocked by the specified clock net, and transmit external data, are covered by the constraint. The scope of the synchronous elements covered by the constraint can be restricted by specifying time groups on a subset of output data pads, a subset of the transmitting synchronous elements, or both.

For more information, see the *Timing Closure User Guide* (UG612).

## Architecture Support

Applies to all FPGA devices and all CPLD devices.

## Applicable Elements

- Global
- Nets
- Time groups

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

Although User Constraints File (UCF) examples are provided, Xilinx® recommends using the Constraints Editor to specify Offset Out.

### Global Method

The global method is the default Offset Out. The global Offset Out applies to all synchronous elements that transmit outgoing data and are triggered by the specified clock signal.

**Note** You can use **BEFORE** with the **RISEING** or **FALLING** keywords. However, if the **REFERENCE\_PIN** keyword is used, then you must use the **AFTER** keyword and cannot use the **BEFORE** keyword.

### UCF Syntax Example

```
OFFSET = OUT "offset_time" [units] {BEFORE "clk_name" | AFTER "clk_name" [REFERENCE_PIN  
"ref_pin"]} [{RISEING | FALLING}];
```



## PCF Syntax Example

```
OFFSET = OUT "offset_time" [units] {BEFORE COMP "clk_iob_name" | AFTER COMP  
"clk_iob_name" [REFERENCE_PIN "ref_pin"]} [{RISING|FALLING}];
```

- **offset\_time [units]**

An optional parameter that defines the time from the clock edge at the input pin of the FPGA device until data first becomes valid at the data output pin.

- If the *offset\_time* value is specified:
  - ◆ A timing constraint is applied to these paths
  - ◆ Errors against that constraint are reported
- If the *offset\_time* value is omitted:
  - ◆ A timing constraint is not generated
  - ◆ The output timing and bus skew of the interface are reported

The report-only option is best used in source synchronous interfaces where the clock to output time is of a lesser concern than the skew of the output bus.

- **BEFORE | AFTER**

Defines the timing relationship from the clock edge to the start of data.

The best method for defining the clock and data requirement is to use the **AFTER** option. **AFTER** describes the time the data begins to be valid after the clock edge at the pin of the FPGA device.

- *clk\_name* defines the fully hierarchical name of the input clock pad net.
- **REFERENCE\_PIN**
  - An optional keyword that is most commonly used in source synchronous output interfaces where the clock is regenerated and sent with the data
  - Allows a bus skew analysis of the output signals relative to the **ref\_pin** signal.

If **REFERENCE\_PIN** is not specified, the bus skew report is referenced to the signal with the minimum clock to output delay.

- **RISING | FALLING**

- Optional keywords that define the transmitting clock edge of the synchronous elements sending the data
- Automatically divide rising and falling edge registers in dual data rate (DDR) interfaces into separate groups for analysis

For more information about the **RISING** and **FALLING** keywords, see the *Timing Constraints User Guide*.

## Output Group Method

When a group of output transmitted by the same clock have a shared timing requirement, the outputs can be grouped together to create a single timing constraint. The outputs can be grouped together by output signal names using pad groups, or by synchronous elements using register groups. By grouping separate signals together into a single time group, the memory and runtime of the implementation tools is reduced. In addition, the timing report will contain bus-based skew and clock centering information.

## UCF Syntax Example

```
[TIMEGRP "pad_groupname"] OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} "clk_name" [REFERENCE_PIN  
"ref_pin"] [TIMEGRP "reg_groupname"] [{RISING|FALLING}];
```

## PCF Syntax Example

```
[TIMEGRP "pad_groupname"] OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} COMP
"clk_iob_name" [REFERENCE_PIN "ref_pin"] [TIMEGRP "reg_groupname"] [{RISING|FALLING}];
```

- The group specific method is identical to the general method with the additions noted below. For the definition of the other variables and keywords, see Global Method above.
- [TIMEGRP "pad\_groupname"] is the optional output pad time group. This time group can be used to limit the scope of the Offset Out constraint to only the synchronous elements feeding the output pad nets contained in the time group.
- [TIMEGRP "reg\_groupname"] is the optional synchronous element time group. This time group can be used to limit the scope of the Offset Out constraint to only the synchronous elements which transmit output data with the specified clock and are contained in the time group.

## Net Specific Method

Offset Out can also be used to specify an output constraint for a specific data net in a schematic, a specific output pad net in the UCF, or a specific output component in the PCF file.

## Schematic Syntax When Attached to a Net Example

```
OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} "clk_name" [TIMEGRP "reg_groupname"]
[REFERENCE_PIN "ref_pin"] [{RISING|FALLING}];
```

## UCF Syntax

```
NET "pad_net_name" OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} "clk_name" [TIMEGRP
"reg_groupname"] [REFERENCE_PIN "ref_pin"] [{RISING|FALLING}];
```

## PCF Syntax

```
COMP "pad_net_name" OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} "clk_name" [TIMEGRP
"reg_groupname"] [REFERENCE_PIN "ref_pin"] [{RISING|FALLING}];
```

- The group specific method is identical to the general method with the additions noted below. For the definition of the other variables and keywords, see *Global Method* above.
- "pad\_net\_name" is the name of the output data net attached to the pad.
- The PCF specification uses I/O Blocks (COMP) instead of NET.
- If the IOB COMP name is omitted in the PCF, or the NET name is omitted in the UCF, the Offset Out specification is assumed to be global.

## UCF Source Synchronous DDR Example

The Source Synchronous Dual Data Rate (DDR) case consists of an interface where the clock is regenerated inside the FPGA and sent with the data to the capturing device. In a DDR interface, data is transmitted with both the rising and falling clock edges. In the DDR case, separate Offset Out constraints must be defined for the rising and falling clock edge registers transmitting the data. The use of the **RISING** and **FALLING** keywords with the Offset Out constraint simplifies this task. Also, for a bus skew analysis relative to the regenerated clock, the **REFERENCE\_PIN** keyword is used.

In this example a clock signal called **clock** enters the FPGA. This clock signal triggers the data output synchronous elements. In addition, a regenerated clock called **TxClock** is created and sent along with the data. Because this is a source synchronous interface, the absolute clock to output time is not required, and the **OFFSET OUT AFTER** value is omitted to generate a report only constraint.

## UCF Syntax

```
NET "clock" TNM_NET = CLK;
TIMESPEC TS_CLK = PERIOD CLK 5.0 ns HIGH 50%;
OFFSET = OUT AFTER clock REFERENCE_PIN "TxClock" RISING;
OFFSET = OUT AFTER clock REFERENCE_PIN "TxClock" FALLING;
```

## UCF System Synchronous SDR Example

The System Synchronous Single Data Rate (SDR) case consists of an interface where the input clock is used to transmit the data to the receiving device. In the SDR interface, data is transmitted once per clock cycle. In this case a single Offset Out requirement is needed to constrain the interface.

In this example a clock signal called **clock** enters the FPGA. This clock signal trigger the data output synchronous elements. Because this is a system synchronous interface, the absolute clock to output time is required to constraint the interface. In this case, a regenerated clock is not present, and the **REFERENCE\_PIN** keyword is omitted to request the default skew reporting.

## UCF Syntax

```
NET "clock" TNM_NET = CLK;
TIMESPEC TS_CLK = PERIOD CLK 5.0 ns HIGH 50%;
OFFSET = OUT 5 ns AFTER "clock";
```

## Schematic Syntax

- Attach to a specific net
- Attribute Name  
OFFSET
- Attribute Values  
OUT *offset\_time* BEFORE|AFTER *clk\_pad\_netname*

## XCF Syntax

The XST Constraint File (XCF) syntax:

- Is the same as the UCF syntax.
- Supports only the OFFSET OUT AFTER method.

## Constraints Editor Syntax

For information on Constraints Editor and Constraints Editor syntax in ISE® Design Suite, see the ISE Design Suite Help.

## PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Open Drain

The Open Drain (OPEN\_DRAIN) constraint:

- Supports CoolRunner™-II devices only.
- Applies to non-tristate (always active) outputs.

### Configuring Outputs as Open Drain

- CoolRunner-II outputs can be configured to drive the primary macrocell output function as an open drain output signal on the pin.
- When the output structure is configured as open drain, a one-state on the output signal produces a high-Z on the device pin instead of a driven High voltage.
- The high-Z associated with Open Drain is not shown during functional simulation, but is represented accurately during post-fit timing simulation.

### Alternatives to Open Drain

- As an alternative to the Open Drain constraint, use the original output-pad signal as a tristate disable. This produces a constant-zero output data value.
- The CPLD Fitter automatically optimizes all tristate outputs with constant-zero data value to take advantage of the open drain capability of the device.

## Architecture Support

CoolRunner-II

### Applicable Elements

The Open Drain constraint applies to:

- Output pads
- Pad nets

### Propagation Rules

The Open Drain constraint is a net or signal constraint. Any attachment to a macro, entity, or module is illegal.

### Constraint Values

- TRUE
- FALSE

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to an output pad net
- Attribute Name  
OPEN\_DRAIN
- Attribute Values  
See *Constraint Values* above.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute OPEN_DRAIN: string;
```

Specify the VHDL constraint as follows:

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

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
NET "mysignal" OPEN_DRAIN;
```

### XCF Syntax

```
BEGIN MODEL "entity_name "
```

```
NET "signal_name " OPEN_DRAIN=true;
```

```
END;
```

## Out Term

The Out Term (OUT\_TERM) constraint:

- Is a basic mapping constraint.
- Sets a configuration of output termination resistors.
- Is valid:
  - On an output pad NET.
  - On an output pad INST.
  - For the entire design.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Out Term constraint may be used with an FPGA device in one or more of the following design elements, or categories of design elements:

- IOB input components (such as IBUF)
- Output Pad Net

Not all devices support all elements. To see which design elements can be used with which devices, see the *Libraries Guides*. For more information, see the device [data sheet](#).

## Propagation Rules

The Out Term constraint is illegal when attached to a net or signal, except when the net or signal is connected to a pad. In this case, Out Term is treated as attached to the pad instance.

## Constraint Values

- NONE
- TUNED
- UNTUNED\_25
- UNTUNED\_50
- UNTUNED\_75

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a pad net
- Attribute Name  
OUT\_TERM
- Attribute Values

See *Constraint Values* above.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
Attribute OUT_TERM: string;
```

Specify the VHDL constraint as follows:

```
attribute OUT_TERM of signal_name : signal is  
"{NONE|TUNED|UNTUNED_25|UNTUNED_50|UNTUNED_75}";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* OUT_TERM = "{NONE|TUNED|UNTUNED_25|UNTUNED_50|UNTUNED_75}" *)
```

### UCF and NCF Syntax

- **NET "pad\_net\_name" OUT\_TERM =**  
**"{NONE|TUNED|UNTUNED\_25|UNTUNED\_50|UNTUNED\_75}" ;**  
Configures the I/O to use a PULLUP.
- **DEFAULT OUT\_TERM = TUNED;**  
Configures the Out Term constraint to be used globally.

### XCF Syntax

```
BEGIN MODEL "entity_name "  
  
  NET "signal_name " out_term=tuned;  
  
END;
```

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide (UG632)*. See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Period

The Period (PERIOD) constraint is a basic timing constraint and synthesis constraint.

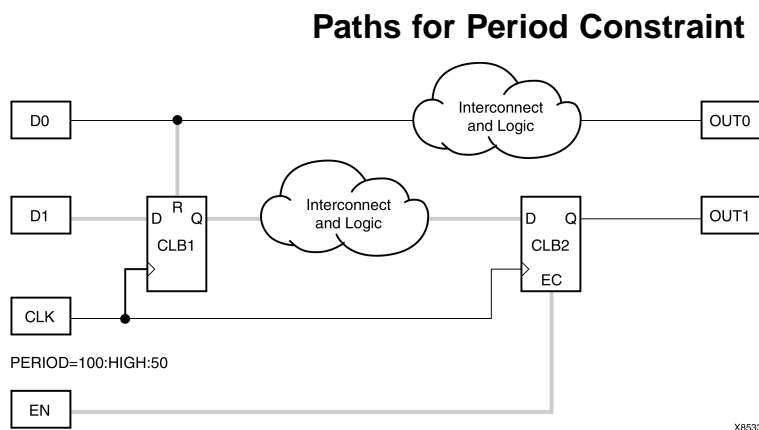
A clock period specification checks timing between all synchronous elements within the clock domain as defined in the destination element group. The group may contain paths that pass between clock domains if the clocks are defined as a function of one or the other.

Derived period constraints are defined in terms of the same units as their reference constraint.

The period specification is attached to the clock net. The definition of a clock period is unlike a FROM-TO style specification because the timing analysis tools automatically take into account any inversions of the clock net at register clock pins, lock phase, and includes all synchronous item types in the analysis. It also checks for hold violations.

A Period constraint on the clock net in the following figure would generate a check for delays on all paths that terminate at a pin that has a setup or hold timing constraint relative to the clock net. This could include the data paths **CLB1.Q** to **CLB2.D**, as well as the path **EN** to **CLB2.EC** (if the enable were synchronous with respect to the clock).

For more information, see the *Timing Closure User Guide* (UG612).



The timing tools do not check pad-to-register paths relative to setup requirements. For example, in the preceding figure, the path from **D1** to Pin **D** of **CLB1** is not included in the Period constraint. The same is true for **CLOCK\_TO\_OUT**.

Special rules that apply when using [Timing Name](#) and [Timing Name Net](#) with the Period constraint for **DLL**, **DCM**, **PLL**, and **MMCM** are discussed below.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Period constraint applies to nets that feed forward to drive flip-flop clock pins.

## Propagation Rules

The Period constraint applies to the signal to which it is attached.



## TIMESPEC Period Method

The primary, recommended method for defining a clock period allows more complex derivative relationships to be defined as well as a simple clock period.

### TIMESPEC Period Method UCF Syntax

The following constraint is defined using the **TIMESPEC** keyword in conjunction with a **TNM** constraint attached to the relevant clock net.

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference" period {HIGH | LOW}
[high_or_low_time] INPUT_JITTER value;
```

- *identifier*  
A reference identifier that has a unique name.
- *TNM\_reference*  
Identifies the group of elements to which the period constraint applies. This is typically the name of a TNM\_NET that was attached to a clock net, but it can be any TNM group or user group (TIMEGRP) that contains only synchronous elements.

The following rules apply:

- The variable name *period* is the required clock period.
- The default units for *period* are nanoseconds, but the number can be followed by ps, ns, micro, or ms. The *period* can also be specified as a frequency value, using units of MHz, GHz, or KHZ.
- Units may be entered with or without a leading space.
- Units are case-insensitive.
- The **HIGH|LOW** keyword indicates whether the first pulse in the period is high or low, and the optional *high\_or\_low\_time* is the polarity of the first pulse. This defines the initial clock edge and is used in the OFFSET constraint. HIGH is the default logic level if the logic level is not specified.
- If an actual time is specified, it must be less than the period.
- If no *high\_or\_low\_time* is specified the default duty cycle is 50%.
- The default units for *high\_or\_low\_time* is ns, but the number can be followed by % or by ps, ns, micro, or ms to specify an actual time measurement.
- INPUT\_JITTER is the random, peak-to-peak jitter on an input clock. The default units are ns.

### TIMESPEC Period Method UCF Syntax Examples

Clock net **sys\_clk** has the constraint *tnm=master\_clk* attached to it and the following constraint is attached to TIMESPEC.

```
TIMESPEC TS_master = PERIOD "master_clk" 50 HIGH 30 INPUT_JITTER
0.050;
```

This period constraint applies to the net **master\_clk**, and defines a clock period of 50 nanoseconds, with an initial 30 nanosecond high time, and INPUT\_JITTER at 50 ps.

```
TIMESPEC TS_clkinA = PERIOD "clkinA" 21 ns LOW 50% INPUT_JITTER
500 ps;
```

```
TIMESPEC TS_clkinB = PERIOD "clkinB" 21 ns HIGH 50% INPUT_JITTER
500 ps;
```

## NET Period Method

**Caution!** This is a secondary method, and is not recommended.

Another method of defining a clock period is to attach the following constraint directly to a net in the path that drives the register clock pins.

### NET Period Method Schematic Syntax

```
PERIOD = period {HIGH|LOW} [ high_or_low_time ] INPUT_JITTER value;
```

### NET Period Method UCF Syntax

```
NET "net_name" PERIOD = period {HIGH|LOW} [ high_or_low_time ] INPUT_JITTER value;
```

- *period* is the required clock period. The default units are nanoseconds, but the timing number can be followed by ps, ns, micro, or ms. The *period* can also be specified as a frequency value, using units of MHz, GHz, or kHz.
- Units may be entered with or without a leading space.
- Units are case-insensitive.
- The **HIGH|LOW** keyword indicates whether the first pulse in the period is high or low, and the optional *high\_or\_low\_time* is the duty cycle of the first pulse. HIGH is the default logic level if the logic level is not specified.
- If an actual time is specified, it must be less than the period.
- If no high or low time is specified the default duty cycle is 50%.
- The default unit for *high\_or\_low\_time* is ns, but the number can be followed by % or by ps, ns, micro or ms to specify an actual time measurement.

The Period constraint is forward traced in exactly the same way a [Timing Name](#) would be and attaches itself to all of the synchronous elements that the forward tracing reaches. If a more complex form of tracing behavior is required (for example, where gated clocks are used), you must place the Period constraint on a particular net or use the preferred method described in the next section.

## Specifying Derived Clocks

The preferred method of defining a clock period uses an identifier, allowing another clock period specification to reference it. Xilinx® recommends using the same **HIGH/LOW** keyword on the derived Period constraints as the master Period constraint. If the master Period constraint has the **HIGH** keyword or is the default, Xilinx recommends using the same **HIGH** keyword on the derived Period constraints. To define the relationship in the case of a derived clock, use the following syntax:

### Specifying Derived Clocks UCF Syntax

```
TIMESPEC "TSidentifier"=PERIOD "timegroup_name" "TSidentifier" [* or /]  
factor PHASE [+|-] phase_value [units];
```

where

- *identifier* is a reference identifier that has a unique name
- *factor* is a floating point number

**Note** You can omit the [\* or /] factor if the specification being defined has the same value as the one being referenced (that is, they differ only in phase); this is the same as using "\* 1".

- *phase\_value* is a floating point number
- *units* are ps, ms, micro, or ns (default)

The following rules apply:

- If an actual time is specified it must be less than the period.
- If no *high\_or\_low\_time* is specified, the default duty cycle is 50%.
- The default units for *high\_or\_low\_time* is *ns*, but the number can be followed by % or by *ps*, *ns*, *micro*, or *ms* to specify an actual time measurement.

### Examples of a Primary Clock with Derived Clocks

Period for primary clock:

```
TIMESPEC "TS01" = PERIOD "clk0" 10.0 ns;
```

Period for clock phase-shifted forward by 180 degrees:

```
TIMESPEC "TS02" = PERIOD "clk180" TS01 PHASE + 5.0 ns;
```

Period for clock phase-shifted backward by 90 degrees:

```
TIMESPEC "TS03" = PERIOD "clk90" TS01 PHASE - 2.5 ns;
```

Period for clock doubled and phase-shifted forward by 180 degrees (which is 90 degrees relative to TS01):

```
TIMESPEC "TS04" = PERIOD "clk180" TS01 / 2 PHASE + 2.5 ns;
```

### Schematic Syntax

- Attach to a net. Following is an example of the syntax format.
- Attribute Name: **PERIOD**
- Attribute Values: *period* [*units*] [{**HIGH**|**LOW**} [*high\_or\_low\_time* [*hi\_lo\_units*]]

### VHDL Syntax

For XST, Period applies only to a specific clock signal.

**Note** Period constraints in the source code (VHDL or Verilog) will not propagate to the netlist.

Declare the VHDL constraint as follows:

```
attribute period: string;
```

Specify the VHDL constraint as follows:

```
attribute period of signal_name : signal is "period [units]";
```

- *period* is the required clock period
- *units* is an optional field to indicate the units for a clock period. The default is nanoseconds (ns), but the timing number can be followed by *ps*, *ns*, or *micro* to indicate the intended units.

### Verilog Syntax

For XST, the Period constraint applies only to a specific clock signal.

**Note** Period constraints in the source code (VHDL or Verilog) do not propagate to the netlist.

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* PERIOD = "period [units]" *)
```

- **period**  
The required clock period
- **units**  
An optional field to indicate the units for a clock period. The default is nanoseconds (ns), but the timing number can be followed by ps, ns, or micro to indicate the intended units.

## UCF and NCF Syntax

Following are examples of UCF and NCF syntax.

- TIMESPEC Period Method, Recommended
- NET Period Method, Not Recommended

## TIMESPEC Period Method, Recommended

This is the primary, recommended method.

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference period" [units]
[{HIGH | LOW} [high_or_low_time [hi_lo_units]]]
INPUT_JITTER value [units];
```

- **identifier**  
A reference identifier that has a unique name.
- **TNM\_reference**  
The identifier name that is attached to a clock net (or a net in the clock path) using the TNM or TNM\_NET constraint.

When a TNM\_NET constraint is traced into the CLKIN input of a DLL, DCM, PLL, or MMCM component, new Period specifications may be created at the DLL/DCM/PLL/MMCM outputs. If new Period specifications are created, new TNM\_NET groups to use in those specifications are also created.

Each new TNM\_NET group is named the same as the corresponding DLL/DCM/PLL/MMCM output net (*outputnetname*). The new Period specification becomes "TS\_*outputnetname*=PERIOD *outputnetname* value units."

The new TNM\_NET groups are then traced forward from the DLL/DCM/PLL/MMCM output net to tag all synchronous elements controlled by that clock signal. The new groups and specifications are shown in the timing analysis reports.

The following rules apply:

- **period**  
Required clock period.
- **units**  
Optional field to indicate the units for a clock period. The default is nanoseconds (ns), but the timing number can be followed by ps, ms, micro, or % to indicate the intended units.
- **HIGH or LOW**  
Indicates whether the first pulse is to be High or Low.
- **high\_or\_low\_time**  
Optional High or Low time, depending on the preceding keyword. If an actual time is specified, it must be less than the period. If no *high\_or\_low\_time* is specified, the default duty cycle is 50 percent.
- **hi\_lo\_units**  
Optional field to indicate the units for the duty cycle. The default is nanoseconds (ns), but the *high\_or\_low\_time* number can be followed by ps, micro, ms, or % if the High or Low time is an actual time measurement.

The following statement assigns a clock period of 40 ns to the net named CLOCK, with the first pulse being High and having a duration of 25 nanoseconds.

```
NET "CLOCK" PERIOD=40 HIGH 25;
```

### NET Period Method, Not Recommended

**Caution!** This is a secondary method, and is not recommended.

```
NET "net_name" PERIOD=period [units] [{HIGH|LOW}
[high_or_low_time [hi_lo_units]]];
```

- **period**  
Required clock period
- **units**  
Optional field to indicate the units for a clock period. The default is nanoseconds (ns), but the timing number can be followed by ps, ns, or micro to indicate the intended units.
- **HIGH or LOW**  
Indicates whether the first pulse is to be High or Low.
- **hi\_lo\_units**  
Can be ns (default), ps, or micro

The following rules apply:

- *high\_or\_low\_time* is the optional High or Low time, depending on the preceding keyword.
- If an actual time is specified, it must be less than the period.
- If no *high\_or\_low\_time* is specified, the default duty cycle is 50 percent.
- *hi\_lo\_units* is an optional field to indicate the units for the duty cycle.
- The default is nanoseconds (ns), but the *high\_or\_low\_time* number can be followed by ps, micro, ms, or % if the High or Low time is an actual time measurement.

## Constraints Editor Syntax

To open Constraints Editor:

1. In the ISE® Design Suite Processes window, double-click **Create Timing Constraint**.
2. In the **Constraint Type** list box under **Timing Constraints**, double-click **Clock Domains**.

## XCF Syntax

XCF syntax is the same as UCF syntax

Both the simple and preferred are supported with the following limitation: HIGH/LOW values are not taken into account during timing estimation/optimization and only propagated to the final netlist if WRITE\_TIMING\_CONSTRAINTS = yes.

## PCF Syntax

**"TSidentifier"=PERIOD perioditem periodvalue INPUT\_JITTER value ;**

- *perioditem* can be:
  - NET *name*
  - TIMEGRP *name*
- *periodvalue* can be:
  - TSidentifier PHASE [+ | -] *time*
  - TSidentifier PHASE *time*
  - TSidentifier PHASE [+ | -] *time* [LOW | HIGH] *time*
  - TSidentifier PHASE *time* [LOW | HIGH] *time*
  - TSidentifier PHASE [+ | -] *time* [LOW | HIGH] *percent*
  - TSidentifier PHASE *time* [LOW | HIGH] *percent*

## PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## FPGA Editor Syntax

To set constraints, in the FPGA Editor main window, click Properties of Selected Items from the Edit menu. To set Period constraint, click Properties of Selected Items from the Edit menu with a net selected. You can set the constraint from the Physical Constraints tab.

## Period Specifications on CLKDLL, DCM, PLL, and MMCM

See the *Timing Closure User Guide* (UG612).

## Pin

The Pin (PIN) constraint:

- Is a User Constraints File (UCF) constraint.
- Defines a net location when used with the [Location \(LOC\)](#) constraint.
- Is used in creating design flows.
- Is translated into a COMP/LOCATE constraint in the Physical Constraints File (PCF) file.

### Architecture Support

Applies to all FPGA devices and no CPLD devices.

### Applicable Elements

Applies to nets.

### Propagation Rules

Not applicable.

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### UCF Syntax

```
PIN "module.pin" LOC=location ;
```

```
PINmod.pinTIG;
```

#### PCF Syntax

```
COMP "name" LOCATE = SITE "location";
```

The Pin constraint specifies that the pseudo component that is created for the pin on the module should be located in the site location. Pseudo logic is created only when a net connects from a pin on one module to a pin on another module.

## Post CRC

The Post CRC (POST\_CRC) constraint:

- Enables or disables the configuration logic CRC error detection feature allowing for notification of any possible change to the configuration memory.
- For Spartan®-3A devices, reserves the multi-use INIT pin for signaling of a configuration CRC failure.

This also allows the banking rules used by the PlanAhead™ software, PAR, and BitGen to refrain from using the IOB that drives the INIT pin. During configuration, the INIT pin operates as normal. After configuration, if Post CRC analysis is enabled, the INIT pin serves as a CRC status pin. If comparison of the real-time computed CRC differs from the pre-computed CRC, a configuration memory change has been detected and the INIT pin is driven low.

For more information, see the device [data sheet](#).

## Architecture Support

- Virtex®-5
- Virtex-6
- Spartan-3A
- Spartan-6

## Applicable Elements

Applies to the entire design.

## Propagation Rules

Applies to the entire design.

## Constraint Values

Value	Description
ENABLE	Enables the Post CRC checking feature
DISABLE	Disables the Post CRC checking features (default)

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF Syntax

```
CONFIG POST_CRC = {ENABLE|DISABLE|ONESHOT};
```

### PCF Syntax

```
CONFIG POST_CRC = {ENABLE|DISABLE|ONESHOT};
```



## Post CRC Action

The Post CRC Action (POST\_CRC\_ACTION) constraint:

- Is a configuration logic CRC error detection mode.
- Compares a pre-computed CRC for the configuration bitstream against a CRC computed by internal logic based on periodic readback of the configuration memory cells.
- Determines whether a CRC mismatch detection continues or whether the CRC operation is halted.
- Is applicable only when the [Post CRC](#) constraint is set to ENABLE.

## Architecture Support

- Spartan®-3A
- Spartan-6
- Virtex®-6

## Applicable Elements

- Applies to the entire device.
- Is not specified on any particular design element.

## Propagation Rules

Applies to the entire design or device.

### Constraint Values

Value	Description
HALT	If a CRC mismatch is detected, cease reading back the bitstream, computing the comparison CRC, and making the comparison against the pre-computed CRC (Default for Spartan-6 devices).
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 (Default for Virtex-6 devices).
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 ceases reading back the bitstream, computing the comparison CRC, and making the comparison against the pre-computed CRC.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF Syntax

```
CONFIG POST_CRC_ACTION = [HALT|CONTINUE];
```

### PCF Syntax

```
CONFIG POST_CRC_ACTION = [HALT|CONTINUE];
```

## Post CRC Frequency

The Post CRC Frequency (POST\_CRC\_FREQ) constraint:

- Is a configuration logic CRC error detection mode.
- Compares a pre-computed CRC for the configuration bitstream against a CRC computed by internal logic based on periodic readback of the configuration memory cells.
- Controls the frequency with which the configuration CRC check is performed for all devices that support this constraint.
- Is applicable only when Post CRC is set to ENABLE.

### Architecture Support

Supports Spartan®-3A, Spartan-6, and Virtex®-6 devices only.

### Applicable Elements

Applies to the entire device and is not specified on any particular design element.

### Propagation Rules

Applies to the entire design.

### Constraint Values

Device	Frequency Range (MHz)	Steps (MHz)	Default Value (MHz)
Spartan-3A	1 to 100	1, 3, 6, 7, 8, 10, 12, 13, 17, 22, 25, 27, 33, 44, 50, 100	1
Spartan-6	1 to 100	1, 2, 4, 6, 10, 12, 16, 22, 26, 33, 40, 50, 66	1
Virtex-6	1 to 50	1, 2, 3, 6, 13, 25, 50	1

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### UCF Syntax

```
CONFIG POST_CRC_FREQ =  
[ 1 | 3 | 6 | 7 | 8 | 10 | 12 | 13 | 17 | 22 | 25 | 27 | 33 | 44 | 50 | 100 ] ;
```

#### PCF Syntax

```
CONFIG POST_CRC_FREQ =  
[ 1 | 3 | 6 | 7 | 8 | 10 | 12 | 13 | 17 | 22 | 25 | 27 | 33 | 44 | 50 | 100 ] ;
```

## Post CRC INIT Flag

The Post CRC INIT Flag (POST\_CRC\_INIT\_FLAG) constraint:

- Is a logic CRC error detection mode.
- Replaces the [Post CRC Signal](#) constraint previously available for Virtex®-5 devices only.
- Applies only when the [Post CRC](#) constraint is set to **enable**.

## Logic CRC Error Detection Mode

In logic CRC error detection mode, a pre-computed CRC for the configuration bitstream is compared against a CRC computed by internal logic based on periodic readback of the configuration memory cells.

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

- Spartan®-6 devices have a POST\_CONFIG\_INTERNAL site where the error condition is also present. This is available regardless of the POST\_CRC\_INIT\_FLAG setting.
- For Virtex-5 devices and Virtex-6 devices, the error condition is always available from the FRAME\_ECC\_VIRTEX5 and FRAME\_ECC\_VIRTEX6 sites.

## Constraint Values

Value	Description
DISABLE	<ul style="list-style-type: none"> <li>• Virtex-5 devices and Virtex-6 devices Disables the use of the INIT_B pin, with the FRAME_ECC site as the sole source of the CRC error signal.</li> <li>• Spartan-6 devices Disables the use of the INIT_B pin as a status output, with the POST_CONFIG_INTERNAL site as the sole source of the CRC error signal. Also the INIT_B pin is reserved and cannot be used as User I/O.</li> </ul>
ENABLE	<ul style="list-style-type: none"> <li>• Leaves the INIT_B pin enabled as a source of the CRC error signal.</li> <li>• ENABLE is the default.</li> </ul>

## Architecture Support

The Post CRC INIT Flag constraint supports the following devices:

- Virtex-5
- Virtex-6
- Spartan-6

## Applicable Elements

Applies to the entire device and is not specified on any particular design element.

## Propagation Rules

Applies to the entire design or device.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF Syntax

```
CONFIG POST_CRC_INIT_FLAG = [DISABLE|ENABLE];
```

### PCF Syntax

```
CONFIG POST_CRC_INIT_FLAG = [DISABLE|ENABLE];
```

## Post CRC Signal

The Post CRC Signal (POST\_CRC\_SIGNAL) constraint:

- Supports Virtex®-5 devices only.
- Is a configuration logic CRC error detection mode.
- Compares a pre-computed CRC for the configuration bitstream against a CRC computed by internal logic based on periodic readback of the configuration memory cells.
- Determines whether the INIT\_B pin is enabled as an output for the Single Event Upset (SEU) error signal. The error condition is still available from the FRAME\_ECC\_VIRTEX5 site.
- Applies only when the [Post CRC](#) constraint is set to ENABLE.

## Architecture Support

Virtex-5

## Applicable Elements

- Applies to the entire device.
- Is not specified on any specific design element.

## Propagation Rules

The Post CRC Signal constraint applies to the entire design.

## Constraint Values

Value	Description
FRAME_ECC_ONLY	Disables the use of the INIT_B pin, with the FRAME_ECC site as the sole source of the CRC error signal
INIT_AND_FRAME_ECC	Leaves the INIT_B pin enabled as a source of the CRC error signal (default)

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF Syntax

```
CONFIG POST_CRC_SIGNAL = [ FRAME_ECC_ONLY | INIT_AND_FRAME_ECC ] ;
```

### PCF Syntax

```
CONFIG POST_CRC_SIGNAL = [ FRAME_ECC_ONLY | INIT_AND_FRAME_ECC ] ;
```

## 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.

### Architecture Support

The Post CRC Source constraint supports the following devices:

- Virtex®-5
- Virtex-6
- Spartan®-6

### Applicable Elements

The Post CRC Source constraint applies to the entire design.

### Propagation Rules

The Post CRC Source constraint applies to the entire design.

### Constraint Values

- PRE\_COMPUTED  
Uses a pre-computed bitstream CRC.
- FIRST\_READBACK  
Uses the first computed CRC.

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### UCF and NCF Syntax

```
CONFIG POST_CRC_SOURCE = {PRE_COMPUTED|FIRST_READBACK};
```

#### PCF Syntax

Same as UCF and NCF syntax.

## Priority

The Priority (PRIORITY) constraint:

- Is an advanced timing constraint.
- Is used when two timing constraints cover the same path.

The *lower* the value, the *higher* the priority.

The Priority value:

- Does not affect which paths are placed and routed first.
- Determines which constraint controls the path when two constraints of equal priority cover the same path.

For more information, see the *Timing Closure User Guide (UG612)*.

## Architecture Support

Applies to all FPGA devices and all CPLD devices.

## Applicable Elements

The Priority constraint applies to [Timing Specifications](#).

## Propagation Rules

Not applicable.

## Constraint Values

Every timing constraint has a priority of 0 as soon as it is written in the User Constraints File (UCF). If a timing constraint is to take precedence over every other constraint, a negative number is required behind the Priority keyword.

- `normal_timespec_syntax`  
A legal timing specification.
- `integer`
  - Represents the priority.
  - The number can be positive, negative, or zero.
  - The value has meaning only when compared with other Priority values.
  - The lower the value, the higher the priority.
  - The constraint with a Priority keyword always has a higher priority than the one without it.

```
TIMESPEC "TS01"=FROM "GROUPA" TO "GROUPB" 40 PRIORITY 4;
```

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

```
normal_timespec_syntax PRIORITY integer;
```

### PCF Syntax

Same as UCF and NCF syntax.



## Prohibit

The Prohibit (PROHIBIT) constraint:

- Is a basic placement constraint.
- Disallows the use of a site within:
  - PAR
  - FPGA Editor
  - CPLD fitter

## Location Types for FPGA Devices

For an FPGA device, use the following location types to define the physical location of an element.

Element Type	Location Specification	Meaning
IOB	P12	IOB location (chip carrier)
	A12	IOB location (pin grid)
	T, B, L, R	Applies to IOB components and indicates edge locations (bottom, left, top, right) for Spartan®-3, Spartan-3A, Spartan-3E, Virtex®-4 and Virtex-5 devices
	LB, RB, LT, RT, BR, TR, BL, TL	Applies to IOB components and indicates half edges (for example, left bottom, right bottom) for Spartan-3, Spartan-3A, Spartan-3E, Virtex-4 and Virtex-5 devices
	Bank 0, Bank 1, Bank 2, Bank 3, Bank 4, Bank 5, Bank 6, Bank 7	Applies to IOB components and indicates half edges (banks) for Spartan-3, Spartan-3A, Spartan-3E, Virtex-4 and Virtex-5 devices
Slice	SLICE_X22Y3	Slice location for Spartan-3, Spartan-3A, Spartan-3E, Virtex-4 and Virtex-5 devices
block RAM	RAMB16_X2Y56	Block RAM location for Spartan-3, Spartan-3A, Spartan-3E, Virtex-4 and Virtex-5 devices
Multiplier	MULT18X18_X55Y82	Multiplier location for Spartan-3, Spartan-3A, Spartan-3E, Virtex-4 and Virtex-5 devices
Global Clock	BUFGMUX0P	Global clock buffer location for Spartan-3, Spartan-3A, Spartan-3E, Virtex-4 and Virtex-5 devices
Digital Clock Manager (DCM)	DCM_X[A]Y[B]	Digital Clock Manager for Spartan-3, Spartan-3A, Spartan-3E, Virtex-4 and Virtex-5 devices
Phase Lock Loop (PLL)	PLL_X[A]Y[B]	Phase Lock Loop for Spartan-3, Spartan-3A, Spartan-3E, Virtex-4 and Virtex-5 devices
Mixed-Mode Clock Manager (MMCM)	MMCM_X[A]Y[B]	Mixed-Mode Clock Manager for Virtex-6 devices

You can use the wildcard character (\*) to replace a single location with a range as shown in the following example.

**SLICE\_X\*Y5**

Any slice of an FPGA device whose **Y-coordinate** is 5

The following are *not* supported:

- Dot extensions on ranges.

Example

**LOC=SLICE\_X3Y5:SLICE\_X5Y7.G**

- The wildcard character for Spartan-3, Spartan-3A, Spartan-3E, Virtex-4 and Virtex-5 global buffers or DLL locations.

## Location Types for CPLD Devices

CPLD devices support only the location type *pin\_name*.

*pin\_name* is:

- *Pnn* for numeric pin names
- *rc* for row-column pin names

## Architecture Support

Applies to all FPGA devices and all CPLD devices.

## Applicable Elements

The Prohibit constraint applies to sites.

## Propagation Rules

It is illegal to attach the Prohibit constraint to a net, signal, entity, module, or macro.

## Constraint Values

- location  
A legal location type for the part type
- site\_group
  - **SITE** "*site\_name*"
  - or
  - **SITEGRP** "*site\_group\_name*"
- site\_name
  - A component site
  - or
  - A CLB or IOB location

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF Syntax

In a User Constraints File (UCF), the Prohibit constraint must be preceded by the keyword **CONFIG**.

- For more information, see *Location Types for FPGA Devices* and *Location Types for CPLD Devices* below.
- For examples of using location types, see the [Location \(LOC\)](#) constraint.

Location	Example
Single Location	<b>CONFIG PROHIBIT=location;</b>
Multiple Single Locations	<b>CONFIG PROHIBIT=location1, location2, ... ,locationn;</b>
Range of Locations	<b>CONFIG PROHIBIT=location1:location2;</b>

CPLD devices do not support **Range of locations**

### UCF Syntax Examples

- **CONFIG PROHIBIT=P45;**  
Prohibits use of the site **P45**.
- **CONFIG PROHIBIT=SLICE\_X6Y8;**  
Prohibits use of the slice at the **SLICE\_X6Y8** site.

### PCF Syntax

Following are examples of Physical Constraints File (PCF) syntax for various locations.

Location	Example
Single Location or Multiple Single Locations	<b>COMP "comp_name" PROHIBIT = [SOFT] "site_group"... "site_group";</b>  <b>COMPGRP "group_name" PROHIBIT = [SOFT]</b> <b>"site_group"... "site_group";</b>  <b>MACRO "name" PROHIBIT = [SOFT] "site_group"... "site_group";</b>
Range of Locations	<b>COMP "comp_name" PROHIBIT = [SOFT]</b> <b>"site_group"... "site_group";</b>  <b>COMPGRP "group_name" PROHIBIT = [SOFT]</b> <b>"site_group"... "site_group";</b>  <b>MACRO "name" PROHIBIT = [SOFT] "site_group"... "site_group";</b>

### PlanAhead™ Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

### PACE Syntax

Pinout and Area Constraints Editor (PACE) supports the Prohibit constraint for CPLD devices only.

For more information, see *Prohibit Mode* in the PACE Help.

### FPGA Editor Syntax

FPGA Editor supports the Prohibit constraint.

FPGA Editor writes the Prohibit constraint to the Physical Constraints File (PCF).

For more information, see *Prohibit Constraint* in the FPGA Editor Help.

## Pulldown

The Pulldown (PULLDOWN) constraint:

- Is a basic mapping constraint.
- Guarantees a logic Low level to allow tri-stated nets to avoid floating when not being driven.

The [Keeper](#), [Pullup](#), and Pulldown constraints:

- Are valid only on a pad NET.
- Are not valid on an INST of any kind.

## Architecture Support

- All FPGA devices
- CoolRunner™-II

## Applicable Elements

- Input
- Tristate outputs
- Bidirectional pad nets

## Propagation Rules

This constraint is a net constraint. Any attachment to a design element is illegal.

## Constraint Values

- YES
- NO
- TRUE
- FALSE

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a pad net
- Attribute Name  
PULLDOWN
- Attribute Values
  - TRUE
  - FALSE

## VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute PULLDOWN: string;
```

Specify the VHDL constraint as follows:

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

## Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

## UCF and NCF Syntax

- **NET "pad\_net\_name" PULLDOWN;**  
Configures the I/O to use a Pulldown constraint.
- **DEFAULT PULLDOWN = TRUE;**  
Configures the Pulldown constraint to be used globally.

## XCF Syntax

```
BEGIN MODEL "entity_name "  
  
NET "signal_name " pulldown=true;  
  
END;
```

## PlanAhead™ Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide (UG632)*. See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Pullup

The Pullup (PULLUP) constraint:

- Is a basic mapping constraint.
- Guarantees a logic High level to allow tri-stated nets to avoid floating when not being driven.

The [Keeper](#), Pullup, and [Pulldown](#) constraints:

- Are valid only on a pad NET.
- Are not valid on an INST of any kind.

For CoolRunner™-II designs, Keeper and Pullup are mutually exclusive across the whole device.

NGDBUILD ignores the following:

- DEFAULT KEEPER = FALSE
- DEFAULT PULLUP = FALSE
- DEFAULT PULLDOWN = FALSE

## Architecture Support

- All FPGA devices
- CoolRunner XPLA3
- CoolRunner-II

## Applicable Elements

- Input
- Tristate outputs
- Bidirectional pad nets

## Propagation Rules

This constraint is a net constraint. Any attachment to a design element is illegal.

## Constraint Values

- YES
- NO
- TRUE
- FALSE

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a pad net
- Attribute Name  
PULLUP
- Attribute Values
  - TRUE
  - FALSE

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute PULLUP: string;
```

Specify the VHDL constraint as follows:

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

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

- **NET "pad\_net\_name" PULLUP;**  
Configures the I/O to use a Pullup constraint.
- **DEFAULT PULLUP = TRUE;**  
Configures the Pullup constraint to be used globally.

### XCF Syntax

```
BEGIN MODEL "entity_name "  
NET "signal_name " pullup=true;  
END;
```

### PlanAhead™ Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints



## Power Mode

The Power Mode (PWR\_MODE) constraint:

- Is an advanced fitter constraint.
- Defines the mode, Low power, or High performance (standard power), of the macrocell that implements the tagged element.
- Is not applied if the tagged function is collapsed forward into its fanouts.

## Architecture Support

Supports XC9500 devices only.

## Applicable Elements

- Nets
- Any instance

## Propagation Rules

- When attached to a *net*, the Power Mode constraint attaches to all applicable elements that drive the net.
- When attached to a *design element*, the Power Mode constraint propagates to all applicable elements in the hierarchy within the design element.

## Constraint Values

- LOW
- STD

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a net or an instance
- Attribute Name  
PWR\_MODE
- Attribute Values

See *Constraint Values* above.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute PWR_MODE: string;
```

Specify the VHDL constraint as follows:

```
attribute PWR_MODE of {signal_name | component_name | label_name}: {signal | component | label} is "{LOW|STD}";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
( * PWR_MODE = "{LOW|STD}" * )
```

### UCF and NCF Syntax

```
NET "$1187/$SIG_0" PWR_MODE=LOW;
```

The macrocell that implements the net \$SIG\_0 is in Low power mode.

### XCF Syntax

```
BEGIN MODEL "entity_name"
```

```
    NET "signal_name" PWR_MODE={LOW|STD};
```

```
    INST "instance_name" PWR_MODE={LOW|STD};
```

```
END;
```

## Registers

The Registers (REG) constraint:

- Is a basic fitter constraint.
- Specifies how a register is to be implemented in the CPLD macrocell.

### Architecture Support

Supports CPLD devices only. Does not support FPGA devices.

### Applicable Elements

Applies to registers.

### Propagation Rules

When attached to a design element, the Registers constraint propagates to all applicable elements in the hierarchy within the design element.

### Constraint Values

- CE
  - When the Registers constraint is applied to a flip-flop primitive with a CE input, CE forces the CE input to be implemented using a clock enable product term in the macrocell.
  - Normally the fitter uses the register CE input only if all logic on the CE input can be implemented using the single CE product term. Otherwise the fitter decomposes the CE input into the D (or T) logic expression unless REG=CE is applied.
  - CE product terms are not available in XC9500 devices (REG=CE is ignored). In XC9500XL devices, the CE product term is available only for registers that do not use both the CLR and PRE inputs.
- TFF
  - Indicates that the register is to be implemented as a T-type flip-flop in the CPLD macrocell.
  - If applied to a D-flip-flop primitive, the D-input expression is transformed to T-input form and implemented with a T-flip-flop.
  - Automatic transformation between D and T flip-flops is normally performed by the CPLD fitter.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a flip-flop instance or macro containing flip-flops
- Attribute Name  
REG

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute REG: string;
```

Specify the VHDL constraint as follows:

```
attribute REG of signal_name : signal is "{CE|TFF}";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* REG = {CE|TFF} *)
```

### UCF and NCF Syntax

```
INST "instance_name" REG = {CE|TFF};
```

The following statement implements the CE pin input using the clock enable product term of the XC9500XL macrocell.

```
INST "Q1" REG=CE;
```

### XCF Syntax

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name" REG={CE|TFF};
```

```
END;
```

## Relative Location (RLOC)

The Relative Location (RLOC) constraint:

- Is a basic mapping and placement constraint.
- Is a synthesis constraint.
- Groups logic elements into discrete sets.
- Allows you to define the location of any element within the set relative to other elements in the set, regardless of eventual placement in the overall design.
- Allows you to place logic blocks relative to each other to increase speed and use die resources efficiently.
- Provides an order and structure to related design elements without requiring you to specify their absolute placement on the FPGA die.
- Allows you to replace any existing hard macro with an equivalent that can be directly simulated.

## Grid Systems

Two coordinate systems can be used to define Relative Location constraints for all FPGA architectures:

- Original grid system  
Does NOT use a universal coordinate system for all component types.
- RPM grid system  
DOES use a universal coordinate system for all component types.

With the RPM grid system, you can create a relocatable RPM macro containing different types of components, such as Block RAM and slice components.

## Using RLOC in the Unified Libraries

In the Unified Libraries, you can use Relative Location constraints:

- With BUFT and CLB related primitives (FMAP).
- On non-primitive macro symbols.

There are some restrictions on the use of Relative Location constraints on BUFT symbols. For more information, see *Set Modifiers* below. You cannot use Relative Location constraints with decoders or clocks.

You can use LOC constraints on all primitives:

- BUFT
- CLB
- CLB
- decoder
- clock

Although Relative Location constraints control the relative placement of logic blocks, they do not guarantee that the same routing resources are used to connect the logic blocks from implementation to implementation. In order to control the routing used, see the [DIRECTED\\_ROUTING](#) constraint.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

For the design elements that can be used with particular device families, see the *Libraries Guides*. For more information, see the device [data sheet](#).

- Registers
- ROM
- RAMS, RAMD
- BUFT
  - Can be used only if the associated RPM has an RLOC\_ORIGIN that causes the Relative Location values in the RPM to be changed to LOC values.
- LUT, MUXF5, MUXF6, MUXCY, XORCY, MULT\_AND, SRL16, SRL16E, MUXF7
  - Spartan®-3, Spartan-3A, Spartan-3E devices only
- MUXF8
  - FPGA devices only
- Block RAM
- Multipliers
- DSP48

## Propagation Rules

Relative Location is a design element constraint. Any attachment to a net is illegal. When attached to a design element, Relative Location propagates to all applicable elements in the hierarchy within the design element.

NGDBuild continues to propagate LOC constraints down the design hierarchy. It adds this constraint to appropriate objects that are not members of a set. While Relative Location constraint propagation is limited to sets, the LOC constraint is applied from its start point all the way down the hierarchy.

When the design is flattened, the row and column numbers of an Relative Location constraint on an element are added to the row and column numbers of the Relative Location constraints of the set members below it in the hierarchy. This feature gives you the ability to modify existing Relative Location values in submodules and macros without changing the previously assigned Relative Location values on the primitive symbols.

## Constraint Syntax

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

**RLOC=XmYn**

where

- *m* is an integer representing the X coordinate
- *n* is an integer representing the Y coordinate

## Using RPM Grid

While Relative Location constraints are applied to symbols in the logical design in the same way as a standard RPM, the grid values are different. The RPM Grid coordinates are determined by selecting the site in question in FPGA Editor and reading the grid coordinates in the history window. For example, selecting the lower leftmost slice site results in the following:

site "SLICE\_X0Y0", type = SLICE (RPM grid X3Y4)

Slice X0Y0 in the original grid system is now shown as X3Y4 in the RPM Grid system. Apply the following constraint to any symbols intended for this slice:

**RLOC = X3Y4**

Use FPGA Editor to look up grid values for a specific device. In addition to the Relative Location constraints, you must apply the following constraint to one symbol in the macro:

**RPM\_GRID = GRID**

Not all synthesis tools support RPM\_GRID. You may need to assign RPM\_GRID using the User Constraints File (UCF) constraint.

**INST "*instance\_name*" RPM\_GRID = GRID**

*where*

*instance\_name* is the full hierarchical path to the symbol name.

## Set Modifiers

A modifier modifies the Relative Location constraints associated with design elements. Since it modifies the Relative Location constraints of all the members of a set, it must be applied in a way that propagates it to all the members of the set. For this reason, the Relative Location modifiers of a set are placed at the start of the set.

The following set modifiers apply to Relative Location constraints:

- **Relative Location**

Modifies the values of other Relative Location constraints below the element in the hierarchy of the set

Regardless of the set type, RLOC values (row, column, extension or XY values) on an element always propagate down the hierarchy and are added at lower levels of the hierarchy to Relative Location constraints on elements in the same set.

- **Relative Location Origin**

Sets the exact die location of the set members. This constraint lets you change the RLOC values into absolute LOC constraints that respect the structure of the set.

The design resolution program (NGCBuild) translates the Relative Location Origin constraint into **Location (LOC)** constraints. The row and column values of the Relative Location Origin are added individually to the members of the set after all Relative Location modifications have been made to their row and column values by addition through the hierarchy. The final values are then turned into LOC constraints on individual primitives.

- **Relative Location Range**

Limits the members of a set to a certain range on the die.

In this case, the set could float as a unit within the range until a final placement. Since every member of the set must fit within the range, it is important that you specify a range that defines an area large enough to respect the spatial structure of the set.

You cannot use Relative Location Range on sets that include BUFT symbols.

- **Use Relative Location**

Turns the Relative Location constraints on and off for a specific element or section of a set. Use Relative Location can be either TRUE or FALSE.

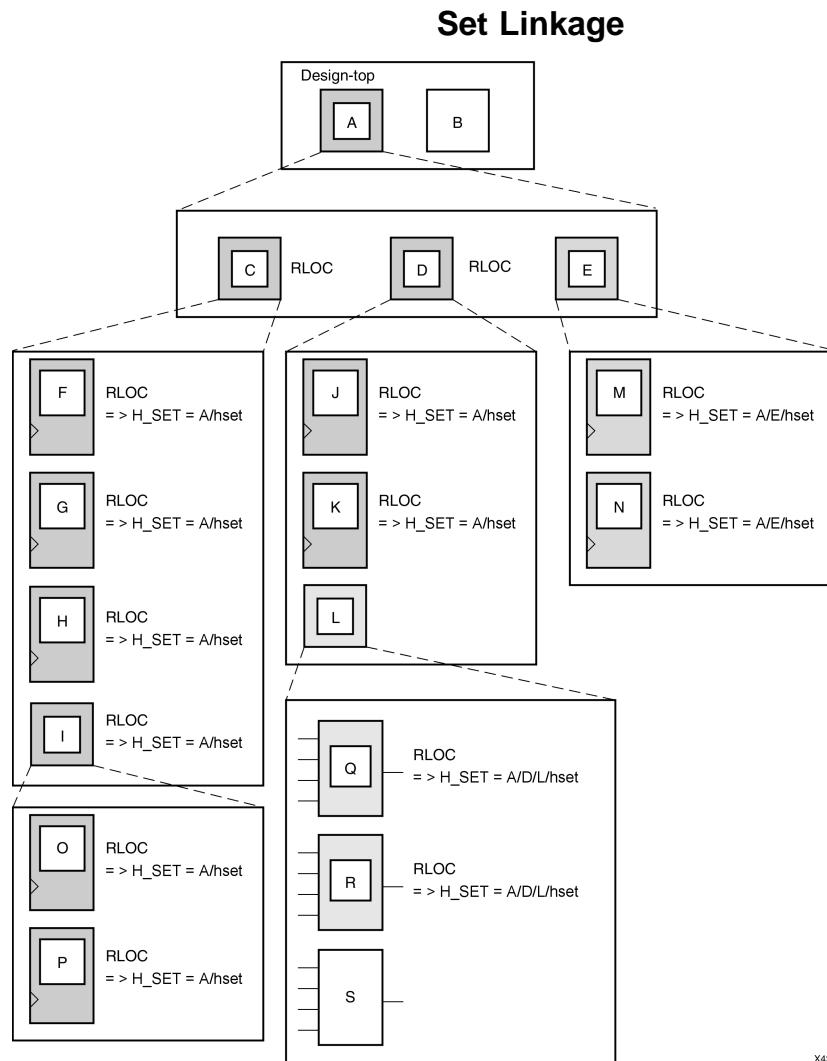
The application of Use Relative Location is strictly based on hierarchy. A Use Relative Location constraint attached to an element applies to all its underlying elements that are members of the same set. If it is attached to a symbol that defines the start of a set, the constraint is applied to all the underlying member elements, which represent the entire set.

When USE\_RLOC=FALSE is applied, the Relative Location and set constraints are removed from the affected symbols in the NCD file. This process is different than that followed for the **Relative Location Origin** constraint. For Relative Location Origin, the mapper generates and outputs a **Location (LOC)** constraint in addition to all the set and Relative Location constraints in the PCF file. The mapper does not retain the original constraints in the presence of USE\_RLOC=FALSE because these cannot be turned on again in later programs.

You can attach Use Relative Location directly to a primitive symbol so that it affects only that symbol.



## Linking Sets



X4295

This example shows the process of linking together elements through the design hierarchy. The complete Relative Location specification,  $RLOC=R\ mCn$  or  $RLOC=XmXn$ , is required for a real design.

**Note** In this and other illustrations in this section, the sets are shaded differently to distinguish one set from another.

All design elements with Relative Location constraints at a single node of the design hierarchy are considered to be in the same **H SET** set unless they are assigned another type of set constraint, an **Relative Location Origin** constraint, or an **Relative Location Range** constraint. In this figure, Relative Location constraints have been added on primitives and non-primitives C, D, F, G, H, I, J, K, M, N, O, P, Q, and R. No Relative Location constraints were placed on B, E, L, or S. Macros C and D have an Relative Location constraint at node A, so all the primitives below C and D that have RLOC values are members of a single H SET set.

The name of this **H SET** set is **A/h\_set** because the set starts at node A. The start of an H SET set is the lowest common ancestor of all the Relative Location-tagged constraints that constitute the elements of that H SET set.

Because element E does not have an Relative Location constraint, it is not linked to the **A/h\_set** set. The Relative Location-tagged elements M and N, which lie below element

E, are therefore in their own H SET set. The start of that H SET set is A/E, giving it the name **A/E/h\_set**.

Similarly, the Q and R primitives are in their own HSET set because they are not linked through element L to any other design elements. The lowest common ancestor for their H SET set is L, which gives it the name **A/D/L/h\_set**. After the flattening, NGDBuild attaches the sets to the primitives shown in the following tale.

Set	Primitives
H_SET=A/h_set	F, G, H, O, P, J, K
H_SET=A/D/L/h_set	Q, R
H_SET=A/E/h_set	N

Consider a situation in which a set is created at the top of the design. There is no lowest common ancestor if macro A also has an Relative Location constraint, since A is at the top of the design and has no ancestor. In this case, the base name **h\_set** has no hierarchically qualified prefix, and the name of the **H SET** set is simply **h\_set**.

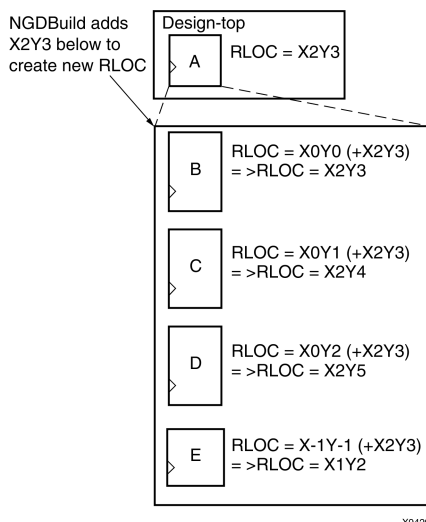
## Modifying Sets

The Relative Location constraint assigns a primitive an RLOC value (the row and column numbers with the optional extensions), specifies its membership in a set, and links together elements at different levels of the hierarchy. In the *Three H\_SET Sets* example, the Relative Location constraint on macros C and D links together all the objects with Relative Location constraints below them. An Relative Location constraint is also used to modify the RLOC values of constraints below it in the hierarchy. In other words, RLOC values of elements affect the RLOC values of all other member elements of the same **H SET** set that lie below the given element in the design hierarchy.

When the design is flattened, the XY values of an Relative Location constraint on an element are added to the XY values of the Relative Location constraints of the set members below it in the hierarchy. This feature allows you to modify existing RLOC values in submodules and macros without changing the previously assigned RLOC values on the primitive symbols.

The following sections describe the effect of the hierarchy on set modification.

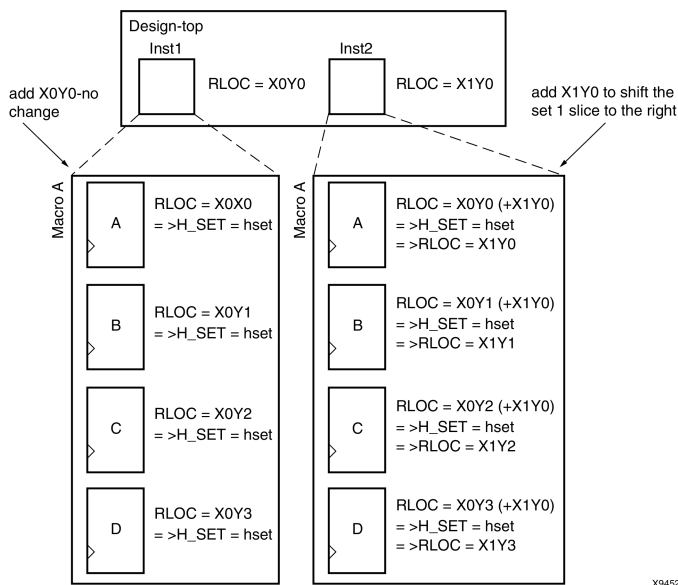
### Adding RLOC Values Down the Hierarchy Example (Slice-Based XY Designations)



This example illustrates the process of adding RLOC values down the hierarchy. The row and column values between the parentheses show the addition function performed by the mapper. The italicized text prefixed by `=>` is added by MAP during the design resolution process and replaces the original Relative Location constraint that you added.

### Modifying RLOC Values of Same Macro and Linking Together as One Set

The ability to modify RLOC values down the hierarchy is particularly valuable when instantiating the same macro more than once. Typically, macros are designed with Relative Location constraints that are modified when the macro is instantiated.

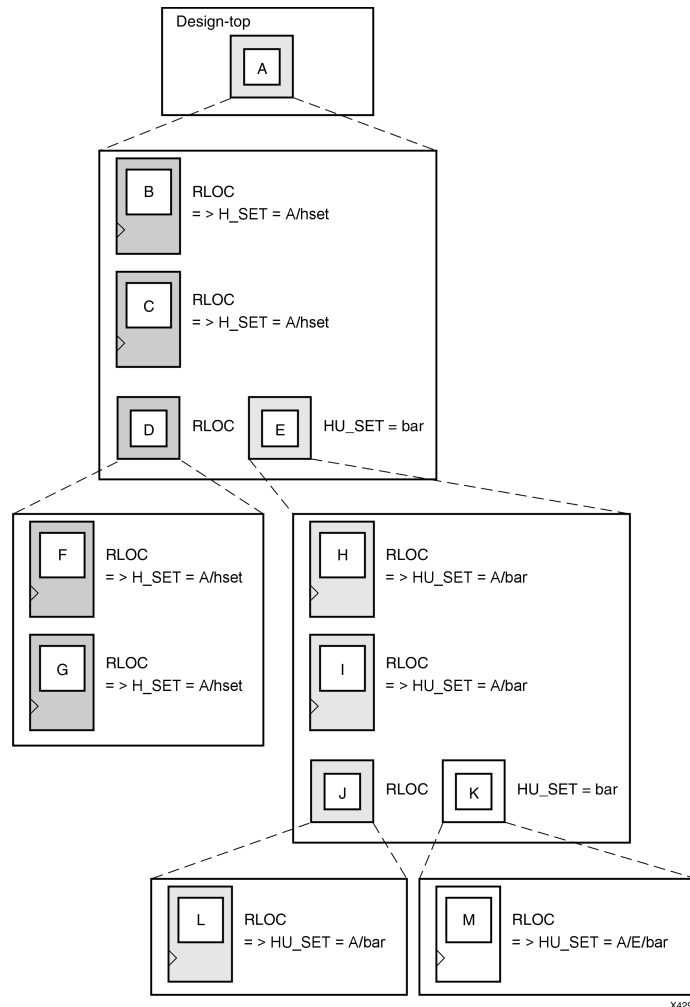


This example is a variation of the previous example. The Relative Location constraint on Inst1 and Inst2 now link all the objects in one **H SET** set.

Because the RLOC=X0Y0 modifier on the Inst1 macro does not affect the objects below it, the mapper adds only the H SET tag to the objects and leaves the RLOC values as they are. However, the RLOC=X1Y0 modifier on the Inst2 macro causes MAP to change the RLOC values on objects below it, as well as to add the H SET tag, as shown in the italicized text.

## Separating Elements from H\_SET Sets

The **HU Set** constraint is a variation of the implicit **H SET** (hierarchy set). HU SET defines the start of a new set. Like H SET, HU SET is defined by the design hierarchy. However, you can use HU SET to assign a user-defined name to the HU SET.



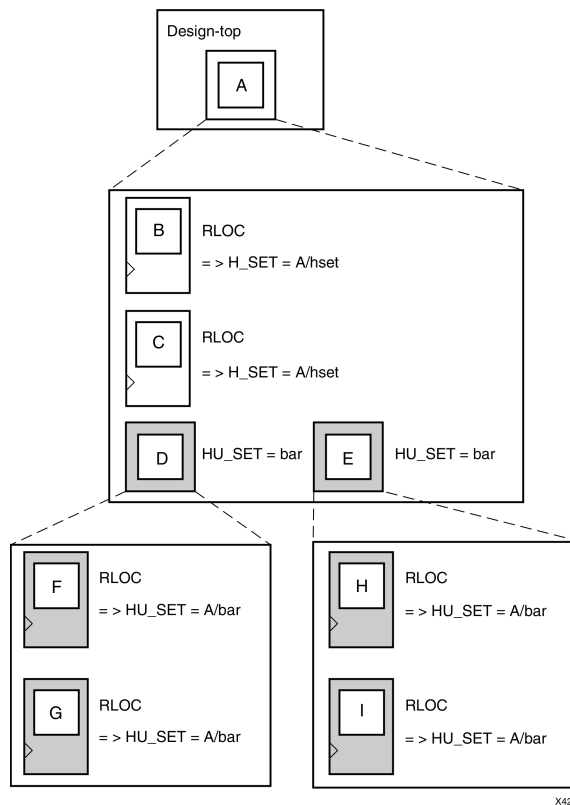
This example demonstrates how **HU SET** constraints designate elements as set members, break links between elements tagged with Relative Location constraints in the hierarchy to separate them from **H SET** sets, and generate names as identifiers of these sets.

The user-defined HU SET constraint on E separates its underlying design elements, namely H, I, J, K, L, and M from the implicit **H\_SET=A/h\_set** that contains primitive members B, C, F, and G. The HU SET set that is defined at E includes H, I, and L (through the element J).

The mapper hierarchically qualifies the name value *bar* on element E to be **A/bar**, since A is the lowest common ancestor for all the elements of the HU SET set, and attaches it to the set member primitives H, I, and L. An HU SET constraint on K starts another set that includes M, which receives the **HU\_SET=A/E/bar** constraint after processing by the mapper.

The same name field is used for the two HU SET constraints, but because they are attached to symbols at different levels of the hierarchy, they define two different sets.

## Linking Two HU\_SET Sets

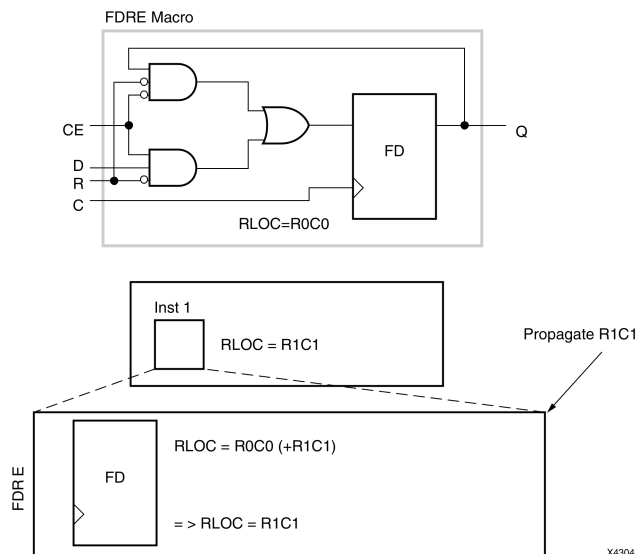


This example shows how HU SET constraints link elements in the same node together by naming them with the same identifier. Because of the same name (*bar*) on two elements, D and E, the elements tagged with Relative Location constraints below D and E become part of the same HU SET.

## Using Relative Location Constraints with Xilinx Macros

Xilinx®-supplied flip-flop macros include an `RLOC=R0C0` constraint on the underlying primitive, which allows you to attach a Relative Location constraint to the macro symbol. This symbol links the underlying primitive to the set that contains the macro symbol.

Simply attach an appropriate Relative Location constraint to the instantiation of the Xilinx flip-flop macro. The mapper adds the `RLOC` value that you specified to the underlying primitive so that it has the desired value.



In this example, the `RLOC = R1C1` constraint is attached to the instantiation (Inst1) of an example macro. It is added to the `R0C0` value of the Relative Location constraint on the flip-flop within the macro to obtain the new `RLOC` values.

If the `RLOC=X1Y1` constraint is attached to Inst1 of a macro, the `X0Y0` value of the Relative Location constraint on the flip-flop within the macro would be used to obtain the new `RLOC` values.

If you do not put an Relative Location constraint on the flip-flop macro symbol, the underlying primitive symbol is the lone member of a set. The mapper removes Relative Location constraints from a primitive that is the only member of a set or from a macro that has no Relative Location objects below it.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to an instance
- Attribute Name  
`RLOC`
- Attribute Values  
See *Constraint Syntax* above.

### 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 "[element]X mYn[.extension]" ;
```

For descriptions of valid values, see [Guidelines for Specifying Relative Locations](#).

The following code sample shows how to use Relative Location constraints with a VHDL generate statement. The code is a simple example showing how to auto-generate the Relative Location constraints for several instantiated FDE components. This methodology can be used with virtually any primitive.

**Note** The user must create the `itoa` function.

```
LEN:for i in 0 to bits-1 generate
  constant row :natural:=((width-1)/2)-(i/2);
  constant column:natural:=0;
  constant slice:natural:=0;
  constant rloc_str : string := "R" & itoa(row) & "C" & itoa(column) & ".S" & itoa(slice);
  attribute RLOC of U1: label is rloc_str;
begin
  U1: FDE port map (
    Q=> dd(j),
    D=> ff_d,
    C=> clk,
    CE => lcl_en(en_idx));
end generate LEN;
```

## Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
( * RLOC = "[element ] XmY n[.extension ] " * )
```

For descriptions of valid value, see [Guidelines for Specifying Relative Locations](#).

## UCF and NCF Syntax

For all FPGA devices, the following statement specifies that an instantiation of FF1 be placed in a slice that is +4 X coordinates and +4 Y coordinates relative to the origin slice.

```
INST "/V2/design/FF1" RLOC=X4Y4;
```

## XCF Syntax

For Virtex®-4 and Virtex-5 devices:

```
BEGIN MODEL "entity_name "
  INST "instance_name " rloc=[element]XmYn [.extension] ;
END;
```

## PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Guidelines for Specifying Relative Locations

The slice-based coordinate system for assigning elements to relative location uses the following general syntax.

**RLOC=X<sub>m</sub> Y<sub>n</sub>**

- m  
The relative X axis (left/right) value
- n  
The relative Y axis (up/down) value
- X and Y can be:
  - Zero
  - Any positive integer
  - Any negative integer

### Integers

Because the X and Y numbers in [Relative Location \(RLOC\)](#) constraints define only the order and relationship between design elements, and not their absolute die locations, their numbering can include negative integers.

Although you can use any integer for Relative Location constraints, Xilinx® recommends small integers for clarity and ease of use.

### Absolute and Relative Values

The *absolute* values of X and Y are not important in RLOC specifications, but rather their *relative* values.

For example, if design element A has an RLOC=X3Y4 constraint and design element B has an RLOC=X6Y7 constraint, the absolute values of X (3 and 6) are not important in themselves. However, the difference between them is significant. In this case, 3 (6-3) specifies that the location of design element B is three slices away from the location of design element A.

To capture this information, a normalization process is used and y coordinate-wise, element is 3 (7-4) slices above element A. In the example just given, normalization reduces the RLOC on design element A to X0Y0, and the RLOC on design element B to X3Y3.

### Slice Numbering in Higher Devices

In Spartan®-3 devices and higher and Virtex®-4 devices and higher, slices are numbered on an XY grid beginning in the lower left corner of the chip.

- X ascends in value horizontally to the right.
- Y ascends in value vertically up.

Relative Location constraints follow the Cartesian-based convention.



## Different RLOC Specifications for Four Flip-Flop Primitives

The following figure demonstrates the use of Relative Location constraints.



X9419

In diagram (a), four flip-flop primitives named A, B, C, and D are assigned Relative Location constraints. These Relative Location constraints require each flip-flop to be placed in a different slice with the slices stacked in the order shown: A below B, C, and D.

To place more than one of these flip-flop primitives per slice, specify the Relative Location constraints as shown in the diagram. The arrangement in the figure requires that A and B be placed in a single slice, and that C and D be placed in another slice immediately to the right of the AB slice.

## Relative Location (RLOC) Sets

Relative Location (RLOC) *constraints* give order and structure to related design elements.

Relative Location (RLOC) *sets* are groups of related design elements to which RLOC constraints have been applied.

- Elements in a set are related by RLOC constraints to other elements in the same set.
- Each member of a set must have an RLOC constraint, which relates it to other elements in the same set.
- You can create multiple sets, but a design element can belong to only one set.

For example, the four flip-flops in *Different RLOC Specifications for Four Flip-Flop Primitives* are related by RLOC constraints and form a set.

### Set Definition

RLOC sets can be defined:

- *Explicitly* through the use of a set parameter.
- or
- *Implicitly* through the structure of the design hierarchy.

### Set Rules

The following rules are associated with each RLOC set.

- **Definition Rules**  
Define the requirements for membership in a set.
- **Linkage Rules**  
Specify how elements can be linked to other elements to form a single set.
- **Modification Rules**  
Dictate how to specify parameters that modify RLOC values of all the members of the set.
- **Naming Rules**  
Specify the nomenclature of sets.

## Set Constraints

Elements must be tagged with both the [Relative Location \(RLOC\)](#) constraint and one of the following set constraints to belong to a set.

- [U Set](#)
- [H Set](#)
- [HU Set](#)

### U Set

The [U Set](#) constraint allows you to group into a single set design elements with attached RLOC constraints that are distributed throughout the design hierarchy. The letter U in the name U Set indicates that the set is user-defined.

The U Set constraint allows you to group elements, even though they are not directly related by the design hierarchy. By attaching a U Set constraint to design elements, you can explicitly define the members of a set.

The design elements tagged with a U Set constraint can exist anywhere in the design hierarchy. They can be primitive or non-primitive symbols. When attached to non-primitive symbols, the U Set constraint propagates to all the primitive symbols with [Relative Location \(RLOC\)](#) constraints that are below it in the hierarchy.

The U Set syntax is:

**U\_SET**=*set\_name*

*set\_name* is the user-specified identifier of the set.

All design elements with RLOC constraints tagged with the same U Set constraint name belong to the same set. Names therefore must be unique among all sets.

## H Set

In contrast to the [U Set](#) constraint, which is *explicitly* defined by tagging design elements, the [H Set](#) constraint is *implicitly* defined through the design hierarchy. A hierarchical set, or H Set set, is defined by the combination of:

- The design hierarchy and
- [Relative Location \(RLOC\)](#) constraints on elements

You can *not* use the H Set constraint to tag the design elements to indicate their set membership. The set is defined automatically by the design hierarchy.

All design elements with RLOC constraints at a single node of the design hierarchy are considered to be in the same H Set set unless they are tagged with another type of set constraint such as [Relative Location Origin \(RLOC\\_ORIGIN\)](#) or [Relative Location Range \(RLOC\\_RANGE\)](#).

If you explicitly tag an element with any of the following, it is removed from an H Set set:

- [Relative Location Origin \(RLOC\\_ORIGIN\)](#)
- [Relative Location Range \(RLOC\\_RANGE\)](#)
- [U Set](#)
- [HU Set](#)

Most designs contain only H Set constraints, since they are the underlying mechanism for relationally placed macros. The Relative Location Origin or Relative Location Range constraints are discussed further in *Set Modifiers*.

NGDBuild does the following:

1. Recognizes the implicit H Set set
2. Derives its name or identifier
3. Attaches the H Set constraint to the correct members of the set
4. Writes them to the output file

## HU Set

[HU Set](#) is a variation of the implicit [H Set](#). Like H Set, HU Set is defined by the design hierarchy. However, you can use the HU Set constraint to assign a user-defined name to the HU Set.

The HU Set syntax is:

**HU\_SET**=*set\_name*

- *set\_name* is the identifier of the set
- *set\_name* must be unique among all the sets in the design

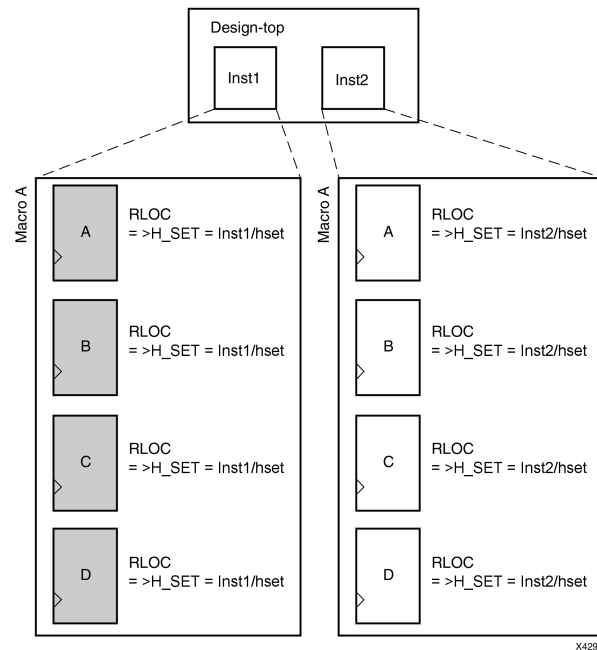
This user-defined name is the base name of the HU Set set. Like the H Set set, in which the base name of **h\_set** is prefixed by the hierarchical name of the lowest common ancestor of the set elements, the user-defined base name of an HU Set set is prefixed by the hierarchical name of the lowest common ancestor of the set elements.

You must define the base names to ensure unique hierarchically qualified names for the sets before the mapper resolves the design and attaches the hierarchical names as prefixes.

HU Set defines the start of a new set. All design elements at the same node that have the same user-defined value for the HU Set constraint are members of the same HU Set set. Along with the HU Set constraint, elements can also have an RLOC constraint.

The presence of an RLOC constraint in an H Set constraint links the element to all elements tagged with RLOC constraints above and below in the hierarchy. However, in the case of an HU Set constraint, the presence of an RLOC constraint along with the HU Set constraint on a design element does not automatically link the element to other elements with RLOC constraints at the same hierarchy level or above.

### Macro A Instantiated Twice



**Note** In this figure and the other related figures shown in the subsequent sections, the italicized text prefixed by => is added by NGDBuild during the design flattening process. You add all other text.

This figure demonstrates a typical use of the implicit H Set. The figure shows only the first RLOC portion of the constraint.

In a real design, the RLOC constraint must be specified completely with:

**RLOC=R** *mCn*

For Spartan®-3 devices and higher and Virtex®-4 devices and higher the RLOC constraint must be specified completely with:

**RLOC=X***mYn*

In this example, macro **A** is originally designed with RLOC constraints on four flip-flops:

- A
- B
- C
- D

The macro is then instantiated twice in the design:

- Inst1
- Inst2

When the design is flattened, two different H Set sets are recognized because two distinct levels of hierarchy contain elements with RLOC constraints. NGDDBuild creates and attaches the appropriate H Set constraint to the set members:

- **H\_SET=Inst1/h\_set** for the macro instantiated in **Inst1**
- **H\_SET=Inst2/h\_set** for the macro instantiated in **Inst2**

The design implementation programs place each of the two sets individually as a unit with relative ordering within each set specified by the RLOC constraints. However, the two sets are regarded to be completely independent of each other.

The name of the H Set set is derived from the symbol or node in the hierarchy that includes all the RLOC elements. **Inst1** is the node (instantiating macro) that includes the four flip-flop elements with RLOC constraints shown on the left of the figure. Therefore, the name of this H\_SET set is the hierarchically qualified name of **Inst1** followed by **h\_set**.

The **Inst1** symbol is considered the start of the H Set, which gives a convenient handle to the entire H Set and attaches constraints that modify the entire H Set. Constraints that modify sets are discussed in the [Save Net Flag](#) constraint.

This figure demonstrates the simplest use of a set that is defined and confined to a single level of hierarchy. Through linkage and modification, you can also create an H Set set that is linked through two or more levels of hierarchy.

Linkage allows you to link elements through the hierarchy into a single set. On the other hand, modification allows you to modify RLOC values of the members of a set through the hierarchy.

## RLOC Set Summary

The following table summarizes the RLOC set types and the constraints that identify members of these sets.

### Summary of Set Types

Type	Definition	Naming	Linkage	Modification
U_SET= name	All elements with the same user-tagged <b>U Set</b> constraint value are members of the same U Set set.	The name of the set is the same as the user-defined name without any hierarchical qualification.	U Set links elements to all other elements with the same value for the <b>U Set</b> constraint.	U Set is modified by applying <b>Relative Location Origin (RLOC_ORIGIN)</b> or <b>Relative Location Range (RLOC_RANGE)</b> constraints on, at most, one of the U Set constraint-tagged elements.
HU_SET= name	All elements with the same hierarchically qualified name are members of the same set.	The lowest common ancestor of the members is prefixed to the user-defined name to obtain the name of the set.	<b>HU Set</b> links to other elements at the same node with the same HU Set constraint value. It links to elements with RLOC constraints below.	The start of the set is made up of the elements on the same node that are tagged with the same HU Set constraint value. A RLOC_ORIGIN or a RLOC_RANGE constraint can be applied to, at most, one of these start elements of an HU Set set.

## Relative Location Origin

The Relative Location Origin (RLOC\_ORIGIN) constraint:

- Is a placement constraint.
- Fixes the members of a set at exact die locations.
- Must specify a single location, not a range or a list of several locations.  
For more information, see *Set Modifiers* in the [Relative Location \(RLOC\)](#) constraint.
- Is required for a set that includes BUFT symbols.
- Cannot be attached to a BUFT instance.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Relative Location Origin constraint applies to instances or macros that are members of sets.

## Propagation Rules

- The Relative Location Origin constraint is a macro constraint. Any attachment to a net is illegal.
- When Relative Location Origin is used in conjunction with an implicit [H SET](#), it must be placed on the element that is the start of the H\_SET set, that is, on the lowest common ancestor of all the members of the set.
- If you apply Relative Location Origin to an [HU SET](#) constraint, place it on the element at the start of the HU SET set, that is, on an element with the HU\_SET constraint.
- However, since several elements could be linked together with the HU SET constraint at the same node, the Relative Location Origin constraint can be applied to only one of these elements to prevent more than one Relative Location Origin constraint from being applied to the HU SET set.
- Similarly, when used with a U SET constraint, the Relative Location Origin constraint can be placed on only one element with the U SET constraint. If you attach the Relative Location Origin constraint to an element that has only an RLOC constraint, the membership of that element in any set is removed, and the element is considered the start of a new H SET set with the specified Relative Location Origin constraint attached to the newly created set.

## Constraint Syntax

To specify a single origin for an RLOC set, use the following syntax, which is equivalent to placing an Relative Location Origin constraint on the schematic.

```
set_name RLOC_ORIGIN=Xm Yn
```

- `set_name`  
Can be the name of any type of RLOC set:
  - U SET
  - HU SET
  - system-generated H SET
- The origin itself is expressed as an **X** and **Y** value representing the location of the elements at **RLOC=X0Y0**

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to an instance that is a member of a set
- Attribute Name  
`RLOC_ORIGIN`
- Attribute Values  
For a list of the constraint values, see the *UCF and NCF Syntax* section below.

### 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 "value";
```

For Spartan®-3, Spartan-3A, Spartan-3E, Virtex®-4 and Virtex-5 devices, *value* is **XmYn**.

For a list of the constraint values, see the *UCF and NCF Syntax* section below.

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* RLOC_ORIGIN = " value" *)
```

For Spartan-3, Spartan-3A, Spartan-3E, Virtex-4 and Virtex-5 devices, *value* is **XmYn**.

For a list of the constraint values, see the *UCF and NCF Syntax* section below.



## UCF and NCF Syntax for Architectures Using Slice-Based XY Coordinates

This section applies to Spartan-3, Spartan-3A, Spartan-3E, Virtex-4, and Virtex-5 devices.

**RLOC\_ORIGIN=X mYn**

- m

Any of the following representing relative **X** coordinates:

- Zero
- A positive integer
- A negative integer

- n

Any of the following representing relative **Y** coordinates:

- Zero
- A positive integer
- A negative integer

The following statement specifies that an instantiation of **FF1**, which is a member of a set, be placed in the slice at **X4Y4** relative to **FF1**.

**INST "/archive/designs/FF1" RLOC\_ORIGIN=X4Y4;**

For example, if **RLOC=X0Y2** for **FF1**, then the instantiation of **FF1** is placed in the slice that is:

- 0 rows to the right of **X4**
- 2 rows up from **Y4** (**X4Y6**)

## PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Relative Location Range

The Relative Location Range (RLOC\_RANGE) constraint:

- Is a placement constraint.
- Is similar to the [Relative Location Origin \(RLOC\\_ORIGIN\)](#), constraint except that it limits the members of a set to a certain range on the die.

The range or list of locations is meant to apply to all applicable elements with [Relative Location \(RLOC\)](#) constraints, not just to the origin of the set

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Relative Location Range constraint applies to instances or macros that are members of sets.

## Propagation Rules

- The Relative Location Range constraint is a macro constraint.
- Attachment to a net is illegal.
- The bounding rectangle applies to all elements in a relationally placed macro, not just to the origin of the set.
- The values of the Relative Location Range constraint are not added to the RLOC values of the elements.
- The Relative Location Range constraint:
  - Does not change the values of the RLOC constraints on underlying elements.
  - Is an additional constraint that is attached automatically by the mapper to every member of a set.
  - Is attached to design elements in the same way as the [Relative Location Origin](#) constraint.
  - Must have values (like Relative Location Origin values) which are non-zero positive numbers since they directly correspond to die locations.
- A User Constraints File (UCF) constraint overrides a netlist constraint if an RLOC set is constrained by either 1) a Relative Location Origin constraint, or 2) a Relative Location Range constraint in the design netlist.

## Constraint Syntax

**RLOC\_RANGE=Xm1 Yn1:X m2Yn2**

The relative X values ( $m1$  and  $m2$ ) and Y values ( $n1$  and  $n2$ ) can be:

- Non-zero positive numbers
- The wildcard (\*) character

This syntax allows for three kinds of range specifications:

- **$Xm1Yn1:Xm2Yn2$**   
A rectangular region bounded by the corners  $Xm1Yn1$  and  $Xm2Yn2$
- **$X*Yn1:X*Ym2$**   
The region on the **Y**-axis between  $n1$  and  $n2$  (any **X** value)
- **$Xm1Y*:Xm2Y$**   
A region on the **X**-axis between  $m1$  and  $m2$  (any **Y** value)

For the second and third kinds of specifications with wildcards, applying the wildcard character (\*) differently on either side of the separator colon creates an error. For example, specifying  $X*Y1:X2Y*$  is an error since the wildcard asterisk is applied to the **X** value on one side and to the **Y** value on the other side of the separator colon.

To specify a range, use the following syntax, which is equivalent to placing an Relative Location Range constraint on the schematic.

```
set_name RLOC_RANGE=X m1Yn1 :Xm2Y n2
```

The range identifies a rectangular area. You can substitute a wildcard (\*) character for either the **X** value or the **Y** value of both corners of the range.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to an instance that is a member of a set
- Attribute Name  
RLOC\_RANGE
- Attribute Values
  - Positive integers (including zero)
  - The wildcard (\*) character

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute rloc_range: string;
```

Specify the VHDL constraint as follows:

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

For Spartan®-3, Spartan-3A, Spartan-3E, Virtex®-4 and Virtex-5 devices *value* is:

```
Xm1Yn1:Xm2Yn2
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* RLOC_RANGE = "value" *)
```

For Spartan-3, Spartan-3A, Spartan-3E, Virtex-4 and Virtex-5 devices, *value* is:

**`Xm1Yn1:Xm2Yn2`**

### UCF and NCF Syntax

This section is applicable Spartan-3 devices and up, and Virtex-4 devices and up.

**`RLOC_RANGE=Xm1Yn1:Xm2Yn2`**

The relative X values (*m1* and *m2*) and Y values (*n1* and *n2*) can be:

- Positive integers (including zero)
- The wildcard (\*) character

The following statement specifies that an instantiation of the macro **MACRO4** be placed relative to other members of the set within a region that is bounded by:

- **X4Y4** in the lower left corner
- **X10Y10** in the upper right corner

**`INST "/archive/designs/MACRO4" RLOC_RANGE=X4Y4:X10Y10;`**

### XCF Syntax

**`MODEL "entity_name" rloc_range=value;`**

**`BEGIN MODEL "entity_name"`**

**`INST "instance_name" rloc_range=value;`**

**`END;`**

## Save Net Flag

The Save Net Flag (SAVE NET FLAG) constraint:

- Is a basic mapping constraint.
- When attached to nets or signals, affects mapping, placement, and routing by preventing the removal of unconnected signals.
- Prevents the removal of loadless or driverless signals.
  - For *loadless* signals, Save Net Flag acts as a dummy OBUF load connected to the signal.
  - For *driverless* signals, Save Net Flag acts as a dummy IBUF driver connected to the signal.
- Can be abbreviated as S NET FLAG.

If you do not have Save Net Flag on a net, any signal that cannot be observed or controlled via a path to an I/O primitive is removed.

Save Net Flag may prevent the trimming of logic connected to the signal.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

- Nets
- Signals

## Propagation Rules

- The Save Net Flag constraint is a net or signal constraint. Any attachment to a design element is illegal.
- The Save Net Flag constraint prevents the removal of unconnected signals. If you do not have the Save Net Flag constraint on a net, any signal not connected to logic or an I/O primitive is removed.

## Constraint Values

- YES
- NO
- TRUE
- FALSE

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a net or signal
- Attribute Name  
SAVE NET FLAG
- Attribute Values
  - TRUE
  - FALSE

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute S: string;
```

Specify the VHDL constraint as follows:

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

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
NET $SIG_9 S;
```

Do not remove the net or signal named \$SIG\_9.

### XCF Syntax

```
BEGIN MODEL entity_name
```

```
NET "signal_name" s=true;
```

```
END;
```

## Schmitt Trigger

The Schmitt Trigger (SCHMITT\_TRIGGER) constraint:

- Causes the attached input pad to be configured with Schmitt Trigger (hysteresis).
- Applies to any input pad.

### Architecture Support

Supports CoolRunner™-II devices only.

### Applicable Elements

The Schmitt Trigger constraint applies to all input pads and pad nets.

### Propagation Rules

The Schmitt Trigger constraint is a net or signal constraint. Any attachment to a macro, entity, or module is illegal.

### Constraint Values

- TRUE
- FALSE

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

- Attach to a net
- Attribute Name  
SCHMITT\_TRIGGER
- Attribute Values  
See *Constraint Values* above.

#### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute SCHMITT_TRIGGER: string;
```

Specify the VHDL constraint as follows:

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

#### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
NET "mysignal" SCHMITT_TRIGGER;
```

### XCF Syntax

```
BEGIN MODEL "entity_name "  
    NET "signal_name " SCHMITT_TRIGGER=true;  
END;
```



## SIM Collision Check

The SIM Collision Check (SIM\_COLLISION\_CHECK) constraint specifies simulation model behavior when a read/write collision occurs on a memory location of block RAM.

### Architecture Support

The SIM Collision Check constraint supports Virtex®-4 devices and higher only.

### Applicable Elements

The SIM Collision Check constraint applies to block RAM primitive elements.

### Propagation Rules

It is illegal to attach the SIM Collision Check constraint to a net or signal.

### Constraint Values

- ALL  
Generates both a WARNING message and X's on the output during simulation.
- NONE  
Ignores collisions leading to unpredictable results during simulation.
- WARNING\_ONLY  
Generates a WARNING message during simulation if there is a read/write collision on a memory location in the Virtex-4 block RAM memory,
- GENERATE\_X\_ONLY  
Generates X's on the outputs during simulation.

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

- Attached to a block RAM primitive
- Attribute Name  
SIM\_COLLISION\_CHECK
- Attribute Values  
See *Constraint Values* above.

#### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute sim_collision_check: string;
```

Specify the VHDL constraint as follows:

```
attribute sim_collision_check of {component_name | label_name}: {component | label}  
is "sim_collision_check_value";
```

## Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

```
// synthesis attribute sim_collision_check [of] {module_name | instance_name }  
[is] "sim_collision_check_value";
```

## UCF and NCF Syntax

The following statement sets the SIM Collision Check constraint for an instantiation of an I/O primitive element **y2**.

```
INST "$1187/y2 SIM_COLLISION_CHECK={WARNING_ONLY|GENERATE_X_ONLY|ALL|NONE};
```

## Slew

The Slew (SLEW) constraint:

- Defines the slew rate (rate of transition) behavior of each individual output to the device.
- May be placed on any output or bi-directional port to specify the port slew rate to be:
  - SLOW (default)
  - FAST
  - QUIETIO (Spartan®-3A devices only)

Use the slowest Slew attribute available to the device while still allowing applicable I/O timing to be met in order to minimize any possible signal integrity issues.

## Architecture Support

Applies to all FPGA devices and all CPLD devices.

## Applicable Elements

- Output primitives
- Output pads
- Bidirectional pads

You can also attach the Slew constraint to the net connected to the pad component in a User Constraints File (UCF).

NGCBuild transfers the Slew constraint from the net to the pad instance in the NGD file so that it can be processed by the mapper.

Use the following syntax:

```
NET "net_name" slew={FAST|SLOW};
```

## Propagation Rules

Place the Slew constraint only on a top-level output or bi-directional port.

## Constraint Values

- FAST
- SLOW
- QUIETIO (Spartan-3A devices only)

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

Specify a new attribute to an output port, or bi-directional port:

- Attribute Name  
SLEW
- Attribute Values

See *Constraint Values* above.

## VHDL Syntax

Before using the Slew constraint, declare it with the following syntax placed after the architecture declaration, but before the **begin** statement in the top-level VHDL file:

```
attribute SLEW: string;
```

Specify the VHDL constraint as follows:

```
attribute SLEW of {top_level_port_name }: signal is "value";
```

## VHDL Syntax Example

```
entity top is
port (FAST_OUT: out std_logic);
end top;

architecture MY_DESIGN of top is
attribute SLEW: string;
attribute SLEW of FAST_OUT: signal is "FAST";
begin
```

## Verilog Syntax

Place the following attribute specification before the port declaration in the top-level Verilog code:

```
(* SLEW="value" *)
```

## Verilog Syntax Example

```
module top (
(* SLEW="FAST" *) output FAST_OUT
);
```

## UCF and NCF Syntax

Placed on output or bi-directional port:

```
NET "top_level_port_name" SLEW="value";
```

## UCF and NCF Syntax Example

```
NET "FAST_OUT" SLEW="FAST";
```

## PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

### **PACE Syntax**

To set the Slew constraint in PACE, select the pin value in the **Design Objects** window. PACE supports CPLD devices only.

## Slow

The Slow (SLOW) constraint:

- Is a basic fitter constraint.
- Enables the slew rate limited control.

### Architecture Support

Applies to all FPGA devices and all CPLD devices.

### Applicable Elements

- Output primitives
- Output pads
- Bidirectional pads

You can attach the Slow constraint to the net connected to the pad component in a User Constraints File (UCF).

NGCBuild transfers the Slow constraint from the net to the pad instance in the Native Generic Database (NGD) file so that it can be processed by the mapper.

Use the following UCF syntax:

```
NET "net_name" SLOW;
```

### Propagation Rules

- The Slow constraint is illegal when attached to a net, except when the net is connected to a pad. In this case, the Slow constraint is treated as attached to the pad instance.
- When attached to a design element, the Slow constraint propagates to all applicable elements in the hierarchy within the design element.

### Constraint Values

- TRUE
- FALSE

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
SLOW
- Attribute Values  
See *Constraint Values* above.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute SLOW : string;
```

Specify the VHDL constraint as follows:

```
attribute SLOW of {signal_name|entity_name}: {signal|entity} is "{TRUE|FALSE}";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

- **INST "\$1I87/y2" SLOW;**  
Establishes a slow slew rate for an instantiation of the element **y2**.
- **NET "net1" SLOW;**  
Establishes a slow slew rate for the pad to which **net1** is connected.

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Stepping

The Stepping (STEPPING) constraint is assigned a value that matches the step level marking on the device. The step level identifies specific device capabilities.

Xilinx® recommends that you use Stepping to set the step level. Otherwise, the software uses a default target device.

For more information on the Stepping constraint, see Xilinx [Answer Record 20947](#), *Stepping FAQs*.

### Architecture Support

- CoolRunner™-II
- Spartan®-3A
- Spartan-3E
- Virtex®-4
- Virtex-5

### Applicable Elements

The Stepping constraint:

- Is a global CONFIG constraint.
- Is not attached to any instance or signal name.

### Propagation Rules

Applies to the entire design.

### Constraint Values

*n*

The target stepping level:

- ES
- SCD1
- 1, 2, 3 ...

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### UCF Syntax

```
CONFIG STEPPING="n";
```

```
CONFIG STEPPING="1";
```



## Suspend

The Suspend (SUSPEND) constraint:

- Supports Spartan®-3A and Spartan-6 devices only.
- Defines the behavior of each individual output when the FPGA device is placed in the SUSPEND power-reduction mode.
- May be placed on any output or bi-directional port to specify the port to be:
  - Tristated (3STATE)
  - Pulled high (3STATE\_PULLUP) or low (3STATE\_PULLDOWN)
  - Driven to the last value (3STATE\_KEEPER or DRIVE\_LAST\_VALUE)

### Architecture Support

- Spartan-3A
- Spartan-6

### Applicable Elements

Place the Suspend constraint only on a top-level output or bi-directional port targeting a Spartan-3A device or a Spartan-6 device.

### Propagation Rules

Place the Suspend constraint only on a top-level output or bi-directional port.

### Constraint Values

- DRIVE\_LAST\_VALUE
- 3STATE (default)
- 3STATE\_PULLUP
- 3STATE\_PULLDOWN
- 3STATE\_KEEPER

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

Specify a new attribute to an output port or bidirectional port:

- Attribute Name  
SUSPEND
- Attribute Values

See *Constraint Values* above.

## VHDL Syntax

Before using the Suspend constraint, declare it with the following syntax placed after the architecture declaration but before the **begin** statement in the top-level VHDL file:

```
attribute SUSPEND: string;
```

After the Suspend constraint has been declared, specify the VHDL constraint as follows:

```
attribute SUSPEND of {top_level_port_name} : signal is "value";
```

```
entity top is
port (STATUS: out std_logic);
end top;architecture MY_DESIGN of top is
attribute SUSPEND: string;
attribute SUSPEND of STATUS: signal is "DRIVE_LAST_VALUE";
begin
```

## Verilog Syntax

Place the following attribute specification before the port declaration in the top-level Verilog code:

```
( * SUSPEND="value" *)

module top ( ( * SUSPEND="DRIVE_LAST_VALUE" *) output STATUS );
```

## UCF and NCF Syntax

Placed on an output or bi-directional port:

```
NET "top_level_port_name" SUSPEND="value";

NET "STATUS" SUSPEND="DRIVE_LAST_VALUE";
```

## PACE Syntax

To set the Suspend constraint from the Pinout and Area Constraints Editor (PACE), select the appropriate pin value from the **Design Objects** window.

## System Jitter

The System Jitter (SYSTEM\_JITTER) constraint:

- Specifies the system jitter of the design.
- Depends on design conditions such as:
  - The number of flip-flops changing at one time.
  - The number of I/Os changing.
- Applies globally to all clocks in the design.
- Combines with the following to generate the Clock Uncertainty value shown in the timing report:
  - The **INPUT\_JITTER** keyword on the **PERIOD** constraint.
  - Any jitter or phase error in the clock network.

For more information, see the *Timing Closure User Guide* (UG612).

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

Applies to the entire design.

## Propagation Rules

Not applicable

## Constraint Values

*value*

- Is a numerical value.
- The default is ns.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
SYSTEM\_JITTER
- Attribute Values  
See *Constraint Values* above.

## VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute SYSTEM_JITTER: string;
```

Specify the VHDL constraint as follows:

```
attribute SYSTEM_JITTER  
of {component_name | signal_name | entity_name | label_name}: {component | signal | entity | label}  
is "value ns";
```

## Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* SYSTEM_JITTER = "value ns" *)
```

## UCF and NCF Syntax

```
SYSTEM_JITTER= value ns;
```

## XCF Syntax

```
MODEL "entity_name" SYSTEM_JITTER = value ns;
```

## Temperature

The Temperature (TEMPERATURE) constraint:

- Is a timing constraint.
- Allows you to specify the operating junction temperature.
- Provides a means of prorating device delay characteristics based on the specified temperature.

## Prorating

Prorating is a scaling operation on existing speed file delays and is applied globally to all delays. Newer devices may not support Temperature prorating until the timing information (speed files) are marked as production status.

## Range of Supported Temperatures

Each architecture has a specific range of supported temperatures. If the specified temperature does not fall within the supported range:

- The constraint is ignored.
- An architecture-specific default value is used instead.
- An error message is displayed during static timing

## Architecture Support

- Spartan®-3A
- Spartan-3E
- Virtex®-4
- Virtex-5

## Applicable Elements

Applies globally to the entire design.

## Propagation Rules

This constraint is a design element constraint. Any attachment to a net is illegal.

## Constraint Values

- value  
A real number specifying the temperature.
- C  
Degrees Celsius (default)
- K  
Degrees Kelvin
- F  
Degrees Fahrenheit

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

**TEMPERATURE=***value* [**C**|**F**|**K**];

The following statement specifies that the analysis for everything relating to speed file delays assumes a junction temperature of 25 degrees Celsius.

**TEMPERATURE=25 C;**

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

### PCF Syntax

Same as *UCF and NCF Syntax*.

## Timing Ignore

The Timing Ignore (TIG) constraint:

- Is a timing constraint and a synthesis constraint.
- Causes paths that fan forward from the point of application (of Timing Ignore) to be treated as if they do not exist (for the purposes of timing analysis) during implementation.
- Can be applied relative to a specific timing specification.
- Can have any of the following values:
  - Empty (global Timing Ignore that blocks all paths)
  - A single TSid to block
  - A comma separated list of TSid components to block, for example
- Is fully supported by Xilinx Synthesis Technology (XST).

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

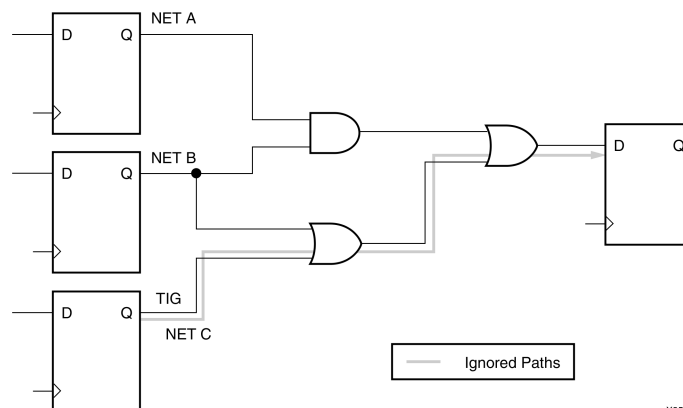
- Nets
- Pins
- Instances

## Propagation Rules

If the Timing Ignore constraint is attached to a net, primitive pin, or macro pin, all paths that fan forward from the point of application of the constraint are treated as if they do not exist for the purposes of timing analysis during implementation. In the following figure:

- NET C is ignored.
- The lower path of NET B that runs through the two OR gates is not ignored.

### Timing Ignore Example



X8529

The following constraint is attached to a net to inform the timing analysis software to ignore paths through the net for specification **TS43**.

Schematic Syntax	UCF Syntax
TIG = TS43	<b>NET</b> " <i>net_name</i> " <b>TIG</b> = <b>TS43</b> ;

You cannot perform path analysis in the presence of combinatorial loops. Therefore, the timing software ignores certain connections to break combinatorial loops. You can use the Timing Ignore constraint to direct the timing tools to ignore specified nets or load pins, consequently controlling how loops are broken.

## Constraint Values

- identifier  
A timing specification that should be ignored
- item  
One of the following:
  - PIN *name*
  - PATH *name*
  - *path specification*
  - NET *name*
  - TIMEGRP *name*
  - BEL *name*
  - COMP *name*
  - MACRO *name*

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

**Note** The Timing Ignore constraint does not affect the timing reported at the bottom of the XST report. The Timing Ignore constraint applies only to the timing reported by Timing Analyzer.

### Schematic Syntax

- Attach to a net or pin
- Attribute Name  
TIG
- Attribute Values  
*value*

### UCF and NCF Syntax

```
NET " net_name " TIG;

PIN "ff_inst.RST" TIG=TS_1;

INST "instance_name " TIG=TS_2;

TIG=TS identifier1 . . . TS identifiern
```



When Attached To ...	The Timing Ignore Constraint ...
instance	Is pushed to the output pins of that instance.
net	Pushes to the drive pin of the net.
pin	Applies to the pin.

The following statement specifies that the timing specifications *TS\_fast* and *TS\_even\_faster* is ignored on all paths fanning forward from the net RESET.

```
NET "RESET" TIG=TS_fast, TS_even_faster;
```

### XCF Syntax

The XST Constraint File (XCF) syntax is the same as the User Constraints File (UCF) syntax.

XST fully supports the Timing Ignore constraint. Timing Ignore can be applied to the nets, situated in the CORE files:

- Electronic Data Interchange Format (EDIF)
- Native Generic Database (NGD)

### Constraints Editor Syntax

For information on Constraints Editor and Constraints Editor syntax in ISE® Design Suite, see the ISE Design Suite Help.

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

### PCF Syntax

```
item TIG;
```

```
item TIG =;
```

```
item TIG = TSidentifier ;
```

## Timing Group

The Timing Group (TIMEGRP) constraint:

- Uses the [Timing Name](#) identifier to group design elements together for timing analysis.
- Allows you to:
  - Define groups in terms of other groups.
  - Create a group that is a combination of existing groups.
  - Place Timing Group constraints in a User Constraints File (UCF) or a Netlist Constraints File (NCF).

## Architecture Support

Applies to all FPGA devices and all CPLD devices.

## Applicable Elements

- Design elements
- Nets

## Propagation Rules

The Timing Group constraint applies to all elements or nets within the group.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Combining Multiple Groups into One

You can define a group by combining other groups.

### Multiple Groups UCF Syntax Example One

The following syntax example illustrates the simple combining of two groups.

```
TIMEGRP "big_group"="small_group" "medium_group";
```

In this syntax example, *small\_group* and *medium\_group* are existing groups defined using a Timing Name or Timing Group attribute.

### Multiple Groups UCF Syntax Example Two

A circular definition, as shown below, causes an error in NGCBuild:

```
TIMEGRP "many_ffs"="ffs1" "ffs2";
```

```
TIMEGRP "ffs1"="many_ffs" "ffs3";
```

### Creating Groups by Exclusion

You can define a group that includes all elements of one group except the elements that belong to another group, as illustrated by the following syntax examples.

### Groups by Exclusion UCF Syntax Example One

```
TIMEGRP "group1"="group2" EXCEPT "group3";
```

- group1  
Represents the group being defined. It contains all of the elements in *group2* except those that are also in *group3*.
- group2 and group3  
Can be a:
  - Valid TNM
  - Predefined group

**Note** OFFSET Constraints do not allow predefined groups.

  - TIMEGRP attribute

### Groups by Exclusion UCF Syntax Example Two

As illustrated by the following example, you can specify multiple groups to include or exclude when creating the new group.

```
TIMEGRP "group1"=" "group2" "group3" EXCEPT "group4" "group5";
```

The example defines a *group1* that includes the members of *group2* and *group3*, except for those members that are part of *group4* or *group5*. All of the groups before the keyword EXCEPT are included, and all of the groups after the keyword are excluded.

### Defining Flip-Flop Subgroups by Clock Sense

You can create subgroups using the **RISING** and **FALLING** keywords to group flip-flops triggered by rising and falling edges.

#### Clock Sense UCF Syntax Example One

```
TIMEGRP "group1"=RISING FFS;
TIMEGRP "group2"=RISING "ffs_group";
TIMEGRP "group3"=FALLING FFS;
TIMEGRP "group4"=FALLING "ffs_group";
```

- group1 to group4  
The new groups being defined.
- The ffs\_group  
Must be a group that includes only flip-flops.

Although keywords (such as **EXCEPT**, **RISING**, and **FALLING**) appear in the documentation in uppercase, you can enter them in lowercase or uppercase. Do not enter them in a combination of lowercase and uppercase.

#### Clock Sense UCF Syntax Example Two

The following example defines a group of flip-flops that switch on the falling edge of the clock.

```
TIMEGRP "falling_ffs"=FALLING FFS;
```

## Defining Latch Subgroups by Gate Sense

Groups of type LATCHES (no matter how these groups are defined) can be easily separated into transparent **high** and transparent **low** subgroups. The **TRANSHI** and **TRANSLO** keywords are provided for this purpose and are used in Timing Group statements like the **RISE** and **FALL** keywords for flip-flop groups.

## Gate Sense UCF Syntax Example One

```
TIMEGRP "lowgroup"=TRANSLO "latchgroup";
```

```
TIMEGRP "highgroup"=TRANSHI "latchgroup";
```

## Creating Groups by Pattern Matching

When creating groups, you can use wildcard characters to define groups of symbols whose associated net names match a specific pattern. This is typically used in schematic designs where net names are specified, not instance names. Synthesis plans typically use INST/TNM syntax. For more information, see [Timing Name \(TNM\)](#).

## Using Wildcards to Specify Net Names

The following wildcard characters enable you to select a group of symbols whose output net names match a specific string or pattern:

- Asterisk \*
- Represents any string of zero or more characters.
- Question mark ?
- Represents a single character.

For example:

- DATA\*
- Specifies any net name that begins with DATA, such as:
  - DATA
  - DATA1
  - DATA22
  - DATABASE
- NUMBER?
- Specifies any net name that begins with NUMBER and ends with one single character, such as:
  - NUMBER1
  - NUMBERS
  - but not
    - ◆ NUMBER
    - ◆ NUMBER12

You can also specify more than one wildcard character. For example:

- \*AT?

Specifies any net name that:

- Begins with any series of characters followed by AT
- Ends with any *one* character, such as
  - ◆ BAT1
  - ◆ CAT2
  - ◆ THAT5

- \*AT\*

Specifies any net name that:

- Begins with any series of characters followed by AT
- Ends with any series of characters, such as
  - ◆ BAT11
  - ◆ CAT26
  - ◆ THAT50

### Wildcards UCF Syntax Example One

The syntax for creating a group using pattern matching is:

**TIMEGRP** *"group\_name"* = *predefined\_group* ("pattern");

- predefined\_group

Can be one of the following predefined groups only:

- FF
- LATCH
- PAD
- RAM
- HSIO
- DSP
- BRAM\_PORTA
- BRAM\_PORTB
- MULT

**Note** OFFSET Constraints do not allow predefined groups.

For the definitions of these groups, see *UCF and NCF Syntax* in [Timing Name Net](#).

**Note** The use of the predefined type MULT would not be correct if multipliers are not available in the architecture.

- pattern

Any string of characters used in conjunction with one or more wildcard characters.

When specifying a net name, you must use its full hierarchical path name so PAR can find the net in the flattened design.

- For the following, specify the output net name:
  - FF
  - RAM
  - LATCH
  - PAD
  - CPU
  - DSP
  - HSIO
  - MULT
- For pads, specify the external net name.

### Wildcards UCF Syntax Example Two

The following example illustrates a group that includes the flip-flops that source nets whose names begin with \$1I3/FRED.

```
TIMGRP "group1"=FFS("$1I3/FRED*");
```

### Wildcards UCF Syntax Example Three

The following example illustrates a group that excludes certain flip-flops whose output net names match the specified pattern.

```
TIMGRP "this_group"=FFS EXCEPT FFS("a*");
```

*this\_group* includes all flip-flops except those whose output net names begin with the letter a

### Wildcards UCF Syntax Example Four

The following example defines a group named *some\_latches*.

```
TIMGRP "some_latches"=latches("$1I3/xyz*");
```

The group *some\_latches* contains all input latches whose output net names start with \$1I3/xyz.

### Additional Pattern Matching Information

In addition to using pattern matching when you create timing groups, you can specify a predefined group qualified by a pattern any place you specify a predefined group. The syntax below illustrates how pattern matching can be used within a timing specification.

### Pattern Matching UCF Syntax Example One

```
TIMSPEC "TSidentifier"=FROM predefined_group("pattern") TO predefined_group("pattern") value;
```

### Pattern Matching UCF Syntax Example Two

Instead of specifying one pattern, you can specify a list of patterns separated by a colon.

```
TIMGRP "some_ffs"=FFS("a*":"b*":"c*d");
```

The group *some\_ffs* contains flip-flops whose output net names adhere to one of the rules shown in the following table.

Pattern	Meaning
a*	Starts with a
b?	Contains two characters, the first of which is b
c*d	Starts with c and ends with d

## Defining Area Groups Using Timing Groups

For more information, see *Defining From Timing Groups* in the [Area Group](#) constraint.

## Timing Groups UCF Syntax Example One

```
TIMEGRP "newgroup"="existing_grp1 " "existing_grp2 " ["existing_grp3 " ...];
```

*newgroup* is a newly created group that consists of:

- Existing groups created via TNM
- Predefined groups
- Note** OFFSET Constraints do not allow predefined groups.
- Other TIMEGRP attributes

## Timing Groups UCF Syntax Example Two

```
TIMEGRP "GROUP1" = "gr2" "GROUP3";
```

```
TIMEGRP "GROUP3" = FFS except "grp5";
```

## XCF Syntax

XST supports TIMEGRP with the following limitations:

- Groups Creation by Exclusion is not supported
- When a group is defined on the basis of another user group with pattern matching:
  - TIMEGRP TG1 = FFS (**machine\***);  
Supported
  - TIMEGRP TG2 = TG1 (**machine\_clk1\***);  
Not supported

## Constraints Editor Syntax

For information on Constraints Editor and Constraints Editor syntax in ISE® Design Suite, see the ISE Design Suite Help.

## PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## PCF Syntax

```
TIMEGRP name ;
```

```
TIMEGRP name = list of elements ;
```

## Timing Specifications

The Timing Specifications (TIMESPEC) constraint:

- Is a basic timing related constraint.
- Is a placeholder for timing specifications (TS attribute definitions).

### TS Attributes

Every TS attribute:

- Begins with the letters TS.
- Ends with a unique identifier that can consist of:
  - Letters
  - Numbers
  - Underscore character

### Architecture Support

Applies to all FPGA devices and all CPLD devices.

### Applicable Elements

The Timing Specifications constraint applies to TS identifiers.

### Propagation Rules

Not applicable.

### Constraint Syntax

The *value* parameter defines the maximum delay for the attribute. Nanoseconds are the default units for specifying delay time in TS attributes. You can also specify delay using other units, such as picoseconds or megahertz.

Keywords, such as **FROM**, **TO**, and **TS**, appear in the documentation in uppercase. However, you can enter them in the TIMESPEC primitive in either uppercase or lowercase. The characters in the keywords must be *all* uppercase or *all* lowercase.

Examples of acceptable keywords are:

- FROM
- PERIOD
- TO
- from
- to

Examples of unacceptable keywords are:

- From
- To
- fRoM
- tO



## TSidentifier name

A *TSidentifier* name referenced in a property value must be in uppercase. For example, the **TSID1** in the second constraint below must be entered in uppercase to match the **TSID1** name in the first constraint.

```
TIMESPEC "TSID1" = FROM "gr1" TO "gr2" 50;
TIMESPEC "TSMIN" = FROM "here" TO "there" TSID1 /2;
```

## Separators

A colon may be used as a separator instead of a space in all timing specifications.

## FROM-TO Syntax

Use the following User Constraints File (UCF) syntax to specify timing requirements between specific end points.

```
TIMESPEC "TSidentifier"=FROM "source_group" TO
"dest_group" value units;
TIMESPEC "TSidentifier"=FROM "source_group" value units;
TIMESPEC "TSidentifier"=TO "dest_group" value units;
```

Unspecified **FROM** or **TO**, as in the second and third syntax statements, implies all points.

**Note** Although you can use a **FROM** or **TO** statement to imply all points, you cannot use an unspecified **THRU** statement by itself to imply all points.

The **From-To** statements are **TS** attributes that reside in the TIMESPEC primitive. The parameters *source\_group* and *dest\_group* must be one of the following:

- Predefined groups
  - Note** OFFSET Constraints do not allow predefined groups.
- Previously created TNM identifiers
- Groups defined in TIMEGRP symbols
- TPSYNC groups

Predefined groups consist of:

- FFS
- LATCHES
- RAMS
- PADS
- CPUS
- DSPS
- HSIOs
- BRAMS\_PORTA
- BRAM\_PORTB
- MULTS

These groups are defined in the UCF and NCF Syntax section in the [Timing Name Net](#) constraint.

Keywords, such as **FROM**, **TO**, and **TS** appear in the documentation in uppercase. However, you can use them in TIMESPEC in either uppercase or lowercase. You cannot enter them in a combination of lowercase and uppercase.

The *value* parameter defines the maximum delay for the attribute. Nanoseconds are the default units for specifying delay time in **TS** attributes. You can also specify delay using other units, such as picoseconds or megahertz.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### TIMESPEC Examples of FROM-TO TS Attributes

```
TIMESPEC "TS_master"=PERIOD "master_clk" 50 HIGH 30;
```

```
TIMESPEC "TS_THIS"=FROM FFS TO RAMS 35;
```

```
TIMESPEC "TS_THAT"=FROM PADS TO LATCHES 35;
```

### UCF Syntax Examples

A **TS** attribute defines the allowable delay for paths. The basic syntax for a **TS** attribute is:

```
TIMESPEC "TSidentifier"=PERIOD "timegroup_name" value [units];
```

- *TSidentifier*  
A unique name for the TS attribute
- *value*  
A numerical value
- *units*  
ms, micro, ps, ns

```
TIMESPEC "TSidentifier"=PERIOD "timegroup_name" "TSidentifier" [* or /]  
factor PHASE [+|-] phase_value [units];
```

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

### PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Timing Name

The Timing Name (TNM) constraint:

- Is a basic grouping constraint.
- Identifies the elements that make up a group which can then be used in a timing specification.
- Tags the following specific elements as members of a group to simplify the application of timing specifications to the group:
  - FF
  - RAM
  - LATCH
  - PAD
  - CPU
  - HSIO
  - MULT
- Can be used with the **RISING** and **FALLING** keywords.

## Timing Name and Partitions

- A Timing Name based upon a PAD name that is associated with a Partition is not supported.
- A Timing Name based upon a net name within a Partition is supported.

## Timing Name and Timing Name Net

- Placing Timing Name on a net groups together Flip-Flops, Latches, RAM, or pads driven by that net.
- Timing Name *does not* propagate through IBUF or BUFG components. The Timing Name is placed on the input pad.
- The [Timing Name Net](#) constraint *does* propagate through IBUF and global clock buffers.
- Xilinx® recommends:
  - Use Timing Name to group instances and macros (hierarchical blocks).
  - To group input pads, use a Timing Name on the net, driven by a pad.
  - Use Timing Name Net to group several (many) logic elements driven by a net, such as clocks, clock enables, chip enables, read/writes, and resets.

## Architecture Support

Applies to all FPGA devices and all CPLD devices.

## Applicable Elements

- You can attach Timing Name to a net, an element pin, a primitive, or a macro.
- You can attach Timing Name to the net connected to the pad component in a User Constraints File (UCF) file. NGCBuild transfers the constraint from the net to the pad instance in the NGDBuild file so that it can be processed by the mapper. Use the following UCF syntax:

```
NET "net_name" TNM="property_value";
```

## Propagation Rules

- When attached to a net or signal, Timing Name propagates to all synchronous elements and PAD components driven by that net. No special propagation is required.
- When attached to a design element, Timing Name propagates to all applicable elements in the hierarchy within the design element.
- Timing Name applied to pad nets does *not* propagate forward through IBUFs. The Timing Name is applied to the external pad. This case includes the net attached to the D input of an IFD. See [Timing Name Net](#) if you want the Timing Name to trace forward from an input pad net.
- Timing Name applied to an IBUF instance is illegal.
- Timing Name applied to the output pin of an IBUF propagates the Timing Name to the next appropriate element.
- Timing Name applied to an IBUF element stays attached to that element.
- Timing Name applied to a clock-pad-net does not propagate forward through the clock buffer.
- When Timing Name is applied to a macro, all the elements in the macro have that timing name.
- Timing Name does not propagate across IBUF components if they are attached to the input pad net.

## Placing Timing Name on Nets

You can place Timing Name on any net. The constraint indicates that the Timing Name value should be attached to all valid elements fed by all paths that fan forward from the tagged net.

Forward tracing stops at the following elements:

- FF
- RAM
- LATCH
- PAD
- CPU
- HSIOs
- MULT

## Placing Timing Name on Macro or Primitive Pins

You can place Timing Name on any component pin if the design entry package allows placement of constraints on primitive pins. The constraint indicates that the Timing Name value should be attached to all valid elements fed by all paths that fan forward from the tagged pin.

Forward tracing stops at the following elements:

- FF
- RAM
- LATCH
- PAD
- CPU
- HSIOs
- MULT

The UCF syntax is:

```
PIN "pin_name" TNM="FLOPS";
```

## Placing Timing Name on Primitive Symbols

You can group individual logic primitives explicitly by placing a constraint on each instance.

The flip-flops tagged with Timing Name form a group called FLOPS. The untagged flip-flops are not part of the group. See the UCF syntax example.

Place only one Timing Name on each symbol, driver pin, or macro driver pin.

## UCF Syntax

```
INST "instance_name" TNM=FLOPS;
```

## Placing Timing Name on Nets or Pins to Group Flip-Flops and Latches

You can easily group flip-flops, latches, or both by flagging a common input net, typically either a clock net or an enable net. If you attach a Timing Name to a net or driver pin, that Timing Name applies to all flip-flops and input latches that are reached through the net or pin. That is, that path is traced forward, through any number of gates or buffers, until it reaches a flip-flop or input latch. That element is added to the specified Timing Name group.

The Timing Name parameter on nets or pins is allowed to have a qualifier. For example, in UCF files:

```
{NET|PIN} "net_or_pin_name" TNM=FFS data;
```

```
{NET|PIN} "net_or_pin_name" TNM=RAMS fifo;
```

```
{NET|PIN} "net_or_pin_name" TNM=RAMS capture;
```

A qualified Timing Name is traced forward until it reaches the first storage element:

- FF
- RAM
- LATCH
- PAD
- CPU
- HSIOs
- MULT

If that type of storage element matches the qualifier, the storage element is given that Timing Name value. Whether or not there is a match, the Timing Name is *not* traced through that storage element.

Timing Name parameters on nets or pins are never traced through a storage element:

- FF
- RAM
- LATCH
- PAD
- CPU
- HSIO
- MULT

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

```
{NET|INST|PIN} "net_or_pin_or_inst_name" TNM= [predefined_group]
identifier;
```

- predefined\_group
  - Can be all of the members of a predefined group using the following keywords:
    - ◆ FF  
All CLB and IOB Flip-Flops, except flip-flops built from function generators.
    - ◆ RAM  
All RAM components, including LUT RAM and BLOCK RAM.
    - ◆ PAD  
All I/O pads.
    - ◆ LATCH  
All CLB or IOB latches, except Latches built from function generators.
    - ◆ MULT groups the Spartan®-3, Spartan-3A, and Spartan-3E registered multiplier.
  - Can be a subset of elements in a *predefined\_group*  
*predefined\_group (name\_qualifier1... name\_qualifiern)*  
*name\_qualifiern* can be any combination of letters, numbers, or underscores. The *name\_qualifier* type (net or instance) is based on the element type that Timing Name is placed on. If the Timing Name is on a NET, the *name\_qualifier* is a net name. If the Timing Name is an instance (INST), the *name\_qualifier* is an instance name.  
 Example  

```
NET clk TNM = FFS (my_flop) Grp1; INST clk TNM =
  FFS (my_macro) Grp2;
```
- identifier
  - Can be any combination of letters, numbers, or underscores.
  - Cannot be any the following reserved words:
    - ◆ FF
    - ◆ RAM
    - ◆ LATCH
    - ◆ PAD

- ◆ CPU
- ◆ HSIO
- ◆ MULT
- ◆ RISING
- ◆ FALLING
- ◆ TRANSHI
- ◆ TRANSLO
- ◆ EXCEPT

Do not use the words in the Reserved Words (Constraints) table below as *identifier*.

### Reserved Words (Constraints)

ADD	ALU	ASSIGN
BEL	BLKNM	CAP
CLKDV_DIVIDE	CLBNM	CMOS
CYMODE	DECODE	DEF
DIVIDE1_BY	DIVIDE2_BY	DOUBLE
DRIVE	DUTY_CYCLE_CORRECTION	FAST
FBKINV	FILE	F_SET
HBLKNM	HU_SET	H_SET
INIT	INIT OX	INTERNAL
IOB	IOSTANDARD	LIBVER
LOC	LOWPWR	MAP
MEDFAST	MEDSLOW	MINIM
NODELAY	OPT	OSC
RES	RLOC	RLOC_ORIGIN
RLOC_RANGE	SCHNM	SLOW
STARTUP_WAIT	SYSTEM	TNM
TRIM	TS	TTL
TYPE	USE_RLOC	U_SET

You can specify as many groups of end points as are necessary to describe your performance requirements. Xilinx® recommends that you use as few groups as possible in order to

- Simplify specification.
- Reduce the Place and Route (PAR) time.

### XCF Syntax

See *UCF and NCF Syntax* below.

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

## PlanAhead Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints



## Timing Name Net

The Timing Name Net (TNM\_NET) constraint:

- Is a basic grouping constraint.
- Identifies the elements that make up a group, which can then be used in a timing specification.
- Is essentially equivalent to [Timing Name \(TNM\)](#) on a net *except* for input pad nets.
- All downstream synchronous elements and pads tagged with the Timing Name Net identifier are considered a group.
- Tags specific synchronous elements, pads, and latches as members of a group to simplify the application of timing specifications to the group.

NGCBuild never transfers a Timing Name Net constraint from the attached net to an input pad, as it does with Timing Name.

## DLL, DCM, PLL, and MMCM Components

Special rules apply when using Timing Name Net with the [Period](#) constraint for the following components:

- DLL
- DCM
- PLL
- MMCM

For more information, see the *Timing Closure User Guide* (UG612).

## Timing Name and Timing Name Net

Placing Timing Name on a net groups together flip-flops, latches, RAM, or pads driven by that net.

Timing Name *does not* propagate through IBUF or BUFG components. The Timing Name will end up on the input pad.

Alternatively, the Timing Name Net attribute *does* propagate through IBUF and global clock buffers.

Xilinx® recommends:

- Use Timing Name to group instances and macros (hierarchical blocks)
- To group input pads, use a Timing Name on the net, driven by a pad.
- Use Timing Name Net to group several (many) logic elements driven by a net, such as clocks, clock enables, chip enables, read/writes, and resets.

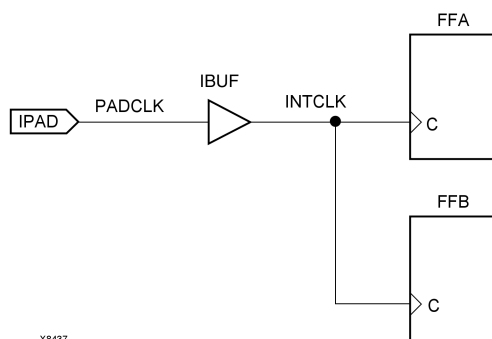
## Timing Name Net Rules

- Timing Name Net constraints applied to pad nets propagate forward through the IBUF or OBUF and any other combinatorial logic to synchronous logic or pads.
- Timing Name Net constraints applied to a clock-pad-net propagate forward through the clock buffer.
- Special rules apply when using Timing Name Net with [Period](#) for Virtex®-4 and Virtex-5 DLL, DCM, and PLL components.

Use Timing Name Net to define certain types of nets that cannot be adequately described by the [Timing Name](#) constraint.

For example, consider the following design

## Timing Name Associated with the IPAD



In the preceding design, a Timing Name constraint associated with the **IPAD** symbol includes only the **PAD** symbol as a member in a timing analysis group. For example, the following UCF file entry creates a time group that includes the **IPAD** symbol only.

```
NET "PADCLK" TNM= "PADGRP";
```

However, using Timing Name to define a time group for the net **PADCLK** creates an empty time group.

```
NET "PADCLK" TNM=FFS "FFGRP";
```

All properties that apply to a pad are transferred from the net to the **PAD** symbol. Since the Timing Name is transferred from the net to the **PAD** symbol, the qualifier **FFS** does not match the **PAD** symbol.

To overcome this obstacle for schematic designs using Timing Name, you can create a time group for the **INTCLK** net.

```
NET "INTCLK" TNM=FFS FFGRP;
```

However, for HDL designs, the only meaningful net names are the ones connected directly to pads. Then, use Timing Name Net to create the **FFGRP** time group.

```
NET PADCLK TNM_NET=FFS FFGRP;
```

NGDBuild does not transfer a Timing Name Net constraint from a net to an **IPAD** as it does with Timing Name.

You can use Timing Name Net in Netlist Constraints File (NCF) or User Constraints File (UCF) files as a property attached to a net in an input netlist (EDIF or NGC). Timing Name Net is not supported in PCF files.

You can use Timing Name Net with nets or instances. If Timing Name Net is used with any other object such as a pin or symbol, a warning is generated and the Timing Name Net definition is ignored.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Timing Name Net constraint applies to nets.

## Propagation Rules

It is illegal to attach the Timing Name Net constraint to a design element.

## Constraint Values

- predefined\_group

**Note** OFFSET Constraints do not allow predefined groups.

- All members of a predefined group using the following keywords:

- ♦ FF

All CLB and IOB flip-flops. Flip-flops built from function generators are not included.

- ♦ LATCH

All CLB or IOB latches. Latches built from function generators are not included.

- ♦ PAD

All I/O pads.

- ♦ RAM

All RAM components, including LUT RAM and block RAM.

- ♦ HSIOs

- ♦ DSP

DSP components groups DSP elements such as the Virtex-4 DSP48.

- ♦ BRAM\_PORTA

- ♦ BRAM\_PORTB

- ♦ MULT

MULT components group the Spartan®-3, Spartan-3A, and Spartan-3E registered multiplier.

- A subset of elements in a *predefined\_group* can be defined as follows:

*predefined\_group (name\_qualifier1... name\_qualifiern)*

*name\_qualifiern* can be any combination of letters, numbers, or underscores. The *name\_qualifier* type (net or instance) is based on the element type that Timing Name Net is placed on. If the TNM\_NET is on a NET, the *name\_qualifier* is a net name. If the Timing Name Net is an instance (INST), the *name\_qualifier* is an instance name.

Example

```
NET clk TNM_NET = FFS (my_flop) Grp1;
```

```
INST clk TNM_NET = FFS (my_macro) Grp2;
```

- identifier

- Can be any combination of letters, numbers, or underscores.
- Cannot be any of the following reserved words: FF, RAM, LATCH, PAD, CPU, HSIOs, MULT, RISING, FALLING, TRANSHI, TRANSLO, EXCEPT.
- Cannot be any of the reserved words in the [Timing Name](#) constraint Reserved Words table.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a net
- Attribute Name  
TNM\_NET
- Attribute Values  
identifier

See *Constraint Values* above.

### UCF and NCF Syntax

```
{NET | INST} "net_name" TNM_NET=[predefined_group:]identifier;
```

The following statement identifies all flip-flops fanning out from the **PADCLK** net as a member of the timing group **GRP1**.

```
NET "PADCLK" TNM_NET=FFS "GRP1";
```

### XCF Syntax

XST supports Timing Name Net with the limitation that only a single pattern is supported for predefined groups.

**Note** OFFSET Constraints do not allow predefined groups.

- XST supports the following command syntax:

```
NET "PADCLK" TNM_NET=FFS "GRP1";
```

- XST does *not* support the following command syntax:

```
NET "PADCLK" TNM_NET = FFS(machine/*:xcouter/*) TG1;
```

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

### PlanAhead™ Syntax

For information about using the PlanAhead™ software to create constraints, see *Floorplanning the Design* in the *PlanAhead User Guide* (UG632). See [PlanAhead](#) in this Guide for information about:

- Defining placement constraints
- Assigning placement constraints
- Defining I/O pin configurations
- Floorplanning and placement constraints

## Timing Point Synchronization

The Timing Point Synchronization (TPSYNC) constraint:

- Is a grouping constraint.
- Flags a particular point or a set of points with an identifier for use in subsequent timing specifications.

If you use the same identifier on several points, timing analysis treats the points as a group.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

- Nets
- Instances
- Pins

## Propagation Rules

When timing must be designed *from* or *to* a point that is not a synchronous element or I/O pad, the following rules apply if a Timing Point Synchronization timing point is attached to any of the following.

- Net

The source of the net is identified as a potential source or destination for timing specifications.

- Macro pin

All of the sources inside the macro that drive the pin to which the constraint is attached are identified as potential sources or destinations for timing specifications. If the macro pin is an input pin (that is, if there are no sources for the pin in the macro), then all of the load pins in the macro are flagged as synchronous points.

- The output pin of a primitive

The output is flagged as a potential source or destination for timing specifications.

- The input pin of a primitive

The input of the primitive is flagged as a potential source or destination for timing specifications.

- An instance

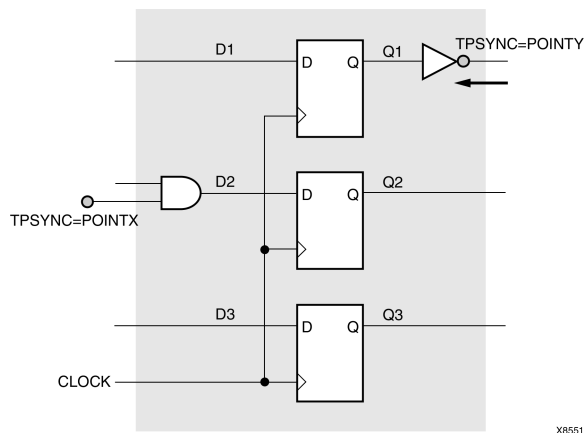
The output of that element is identified as a potential source or destination for timing specifications.

- A primitive symbol

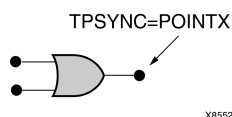
When attached to a primitive symbol, the Timing Point Synchronization constraint identifies the outputs of that element as a potential source or destination for timing specifications. See the following figure.

## TPSYNC Attached to Macro Pins

POINTY applies to the inverter.

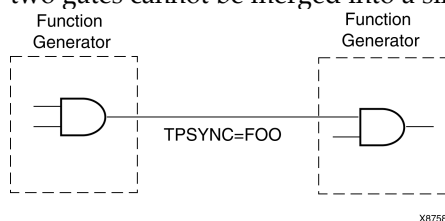


## TPSYNC Attached to a Primitive Symbol



## Working with Two Gates

Using a Timing Point Synchronization timing point to define a synchronous point implies that the flagged point cannot be merged into a function generator. For example, in the following diagram, because of the Timing Point Synchronization definition, the two gates cannot be merged into a single function generator.



## Constraint Values

identifier

A name that is used in timing specifications in the same way as groups.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attached to a net, instance, or pin
- Attribute Name  
TPSYNC
- Attribute Values  
See *Constraint Values* above.

### UCF and NCF Syntax

```
NET "net_name" TPSYNC=identifier;  
INST "instance_name" TPSYNC=identifier;  
PIN "pin_name" TPSYNC=identifier;
```

All flagged points are used as a source or destination or both for the specification where the TPSYNC identifier is used.

The name for the identifier must be unique to any identifier used for a [Timing Name](#) or [Timing Name Net](#) grouping constraint.

The following statement identifies **latch** as a potential source or destination for timing specifications for the net **logic\_latch**.

```
NET "logic_latch" TPSYNC=latch;
```

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

## Timing Thru Points

The Timing Thru Points (TPTHRU) constraint:

- Is a grouping constraint.
- Flags a particular *point* or *set of points* with an identifier for reference in subsequent timing specifications.
  - If you use the same identifier on several points, timing analysis treats the points as a group.
  - For more information, see [Timing Specifications](#).
- Defines intermediate points on a path to which a specification applies.

For more information, see [Timing Specification Identifier](#).

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

- Nets
- Pins
- Instances

## Propagation Rules

Not applicable.



## Constraint Values

- identifier  
ASCII string made up of the following characters:
    - A to Z
    - a to z
    - 0 to 9
    - Underscore \_
  - source\_group and dest\_group
    - User-defined group
    - or
    - Predefined group

**Note** OFFSET Constraints do not allow predefined groups.

or

  - TPSYNC
- thru\_point  
Intermediate point used to qualify the path, defined using a TPTHURU constraint
- allowable\_delay  
Timing requirement
- units  
Optional field to indicate the units for the allowable delay.
  - Default units are nanoseconds (ns).
  - The timing number can be followed by:
    - ◆ ps
    - ◆ ns
    - ◆ micro
    - ◆ ms
    - ◆ GHz
    - ◆ MHz
    - ◆ KHz

The identifier name must be different from any identifier used for a [Timing Name](#) constraint.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a net, instance, or pin
- Attribute Name  
TPTHURU
- Attribute Values  
identifier

For more information, see *UCF and NCF Syntax* below.

## UCF and NCF Syntax

```
NET "net_name" TPTHU=identifier;

INST "instance_name" TPTHU=identifier;

PIN "instance_name.pin_name" TPTHU="thru_group_name";
```

## Using TPTHU in a FROM TO Constraint

Defining intermediate points on a path to which a specification applies

- Defines the maximum allowable delay.
- Has the syntax shown in the following sections.

## UCF Syntax with TIMESPEC

```
TIMESPEC "TSidentifier"=FROM "source_group" THRU
"thru_point" [THRU "thru_point"] TO
"dest_group" allowable_delay [units];

TIMESPEC "TSidentifier"=FROM "source_group" THRU
"thru_point" [THRU "thru_point"] allowable_delay [units];
```

This example shows how to use the TPTHU constraint with the THRU constraint on a schematic.

The UCF syntax is as follows.

```
INST "FLOPA" TNM="A";

INST "FLOPB" TNM="B";

NET "MYNET" TPTHU="ABC";

TIMESPEC "Tspath1"=FROM "A" THRU "ABC" TO "B" 30;
```

- **NET "on\_the\_way" TPTHU="here";**  
Identifies the net **on\_the\_way** as an intermediate point on a path to which the timing specification named **here** applies.
- **TIMESPECT "TS\_1"=THRU "Thru\_grp" 30.0**  
Netlist Constraints File (NCF) construct is not supported.

## Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

## PCF Syntax

```
PATH "name"=FROM "source" THRU "thru_pt1" THRU "thru_ptn" TO
"destination";
```

You are not required to have a FROM, THRU, and TO.

You can have almost any combination, such as:

- FROM-TO
- FROM-THRU-TO
- THRU-TO
- TO
- FROM
- FROM-THRU-THRU-THRU-TO
- FROM-THRU

There is no restriction on the number of THRU points.

The source, THRU points, and destination can be a:

- Net
- Bel
- Comp
- Macro
- Pin
- Timegroup

## Timing Specification Identifier

The Timing Specification Identifier (*TSidentifier*) constraint is a basic timing constraint.

- *TSidentifier* properties beginning with the letters TS are used with the [Timing Specifications](#) constraint in a User Constraints File (UCF).
- The value of *TSidentifier* corresponds to a specific timing specification that can then be applied to paths.

### Architecture Support

Applies to all FPGA devices and all CPLD devices.

### Applicable Elements

The Timing Specification Identifier constraint applies to **TIMESPEC** keywords.

### Propagation Rules

It is illegal to attach the Timing Specification Identifier constraint to a net, signal, or design element.

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### UCF and NCF Syntax

The following syntax definitions use a space as a separator. Using a colon as a separator is optional.

#### Defining a Maximum Allowable Delay

```
TIMESPEC "TSidentifier"=FROM "source_group" TO "dest_group" allowable_delay [units];
```

## Defining Intermediate Points (UCF)

```
TIMESPEC "TSidentifier"=FROM "source_group" THRU "thru_point" [THRU "thru_point1"...
"thru_pointn"] TO "dest_group" allowable_delay [units];
```

- identifier  
ASCII string made up of the following characters:
  - A to Z
  - a to z
  - 0 to 9
  - Underscore \_
- source\_group and dest\_group  
User-defined or predefined groups  
**Note** OFFSET Constraints do not allow predefined groups.
- thru\_point  
An intermediate point used to qualify the path, defined using a TPTHURU constraint
- allowable\_delay  
Timing requirement value
- units
  - An optional field to indicate the units for the allowable delay.
  - The default units are nanoseconds (ns).
  - The timing number can be followed by ps, ns, micro, ms, GHz, MHz, or kHz to indicate the intended units.

## Defining a Linked Specification

You can link the timing number used in one specification to another specification.

```
TIMESPEC "TSidentifier"=FROM "source_group" TO "dest_group" another_TSid [/|*] number;
```

- identifier  
ASCII string made up of the following characters:
  - A to Z
  - a to z
  - 0 to 9
  - Underscore \_
- source\_group and dest\_group  
User-defined or predefined groups  
**Note** OFFSET Constraints do not allow predefined groups.
- another\_Tsid  
The name of another timespec
- number  
A floating point number

## Defining a Clock Period

This allows more complex derivative relationships to be defined as well as a simple clock period.

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference" value [units] [{HIGH | LOW}  
[high_or_low_time [hi_lo_units]]] INPUT_JITTER value;
```

- identifier  
Reference identifier with a unique name
- TNM\_reference  
Identifier name attached to a clock net (or a net in the clock path) using a TNM constraint
- value  
Required clock period
- units
  - Optional field to indicate the units for the allowable delay.
  - The default is nanoseconds (ns)
  - The timing number can be followed by micro, ms, ps, ns, GHz, MHz, or kHz to indicate the intended units
- HIGH or LOW  
Optionally specified to indicate whether the first pulse is to be High or Low
- high\_or\_low\_time
  - Optional High or Low time, depending on the preceding keyword
  - If an actual time is specified, it must be less than the period.
  - If no High or Low time is specified, the default duty cycle is 50 percent.
- hi\_lo\_units
  - Optional field to indicate the units for the duty cycle.
  - The default is nanoseconds (ns).
  - If the High or Low time is an actual time measurement, the High or Low time number can be followed by ps, micro, ms, ns or %.

## Specifying Derived Clocks

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference" "another_PERIOD_identifier" [/[*] number [{HIGH|LOW}  
[high_or_low_time [hi_lo_units]]] INPUT_JITTER value;
```

- TNM\_reference  
Identifier name attached to a clock net (or a net in the clock path) using a TNM constraint
- another\_PERIOD\_identifier  
Name of the identifier used on another period specification
- number  
A floating point number
- HIGH or LOW  
Optionally specified to indicate whether the first pulse is to be High or Low
- high\_or\_low\_time  
The optional High or Low time, depending on the preceding keyword. If an actual time is specified, it must be less than the period. If no High or Low time is specified, the default duty cycle is 50 percent.
- hi\_lo\_units

- An optional field to indicate the units for the duty cycle.
- The default is nanoseconds (ns).
- If the High or Low time is an actual time measurement, the High or Low time number can be followed by ps, micro, ms, or %.

### Ignoring Paths

**Note** This form is not supported for CPLD devices.

There are situations in which a path that exercises a certain net should be ignored because all paths through the net, instance, or instance pin are not important from a timing specification point of view.

```
TIMESPEC "TSidentifier"=FROM "source_group" TO "dest_group" TIG;
```

OR

```
TIMESPEC "TSidentifier"=FROM "source_group" THRU "thru_point" [THRU "thru_point1"...  
"thru_pointn"] TO "dest_group" TIG;
```

- identifier  
ASCII string made up of the following characters:
  - A to Z
  - a to z
  - 0 to 9
  - Underscore \_
- source\_group and dest\_group  
User-defined or predefined groups  
**Note** OFFSET Constraints do not allow predefined groups.
- thru\_point  
An intermediate point used to qualify the path, defined using a TPTHU constraint

### Ignoring Paths Examples

- **TIMESPEC "TS\_35"=FROM "here" TO "there" 50;**  
Specifies that the timing specification TS\_35 calls for a maximum allowable delay of 50 ns between the groups **here** and **there**.
- **TIMESPEC "TS\_70"=PERIOD "clock\_a" 25 high 15;**  
Specifies that the timing specification TS\_70 calls for a 25 ns clock period for **clock\_a**, with the first pulse being High for a duration of 15 ns.

For more information, see:

- [Logical Constraints](#)
- [Physical Constraints](#)

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

Item	Entry Method
Clock period constraints	Clock Domains entry
Input setup time	Inputs entry
Clock-to-output delay	Outputs entry
Pad-to-pad delays	Exceptions > Paths category

### PCF Syntax

The PCF syntax is the same as the UCF syntax without the **TIMESPEC** keyword.

### FPGA Editor Syntax

For information on setting constraints in FPGA Editor, including syntax, see the FPGA Editor Help.



## U Set

The U Set (U\_SET) constraint:

- Is an advanced mapping constraint.
- Groups design elements with attached [Relative Location \(RLOC\)](#) constraints that are distributed throughout the design hierarchy into a single set.

The elements that are members of a U Set can cross the design hierarchy. You can arbitrarily select objects without regard to the design hierarchy and tag them as members of a U Set.

For more information, see [Relative Location \(RLOC\) Sets](#).

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The U Set constraint may be used with an FPGA device in one or more of the following design elements, or categories of design elements. Not all devices support all elements. To see which design elements can be used with which devices, see the *Libraries Guides*. For more information, see the device data sheet.

- Registers
- Macro Instance
- FMAP
- ROM
- RAMS
- RAMD
- BUFT
- MULT18X18S
- RAMB4\_*Sm\_Sn*
- RAMB4\_*Sn*
- RAMB16\_*Sm\_Sn*
- RAMB16\_*Sn*
- RAMB16
- DSP48

## Propagation Rules

The U Set constraint is a macro constraint. Any attachment to a net is illegal.

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
U\_SET
- Attribute Values  
*name* is the identifier of the set

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute U_SET: string;
```

Specify the VHDL constraint as follows:

```
attribute U_SET of {component_name | label_name}:{component | label} is name;  
name
```

The identifier of the set.

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
( * U_SET = * ) name  
name
```

The identifier of the set.

### UCF and NCF Syntax

```
INST "instance_name" U_SET= name;  
name
```

- Is the identifier of the set.
- Is absolute.
- Is not prefixed by a hierarchical qualifier.

The following statement places the element ELEM\_1 in the set JET\_SET.

```
INST "$1I3245/ELEM_1" U_SET=JET_SET;
```

### XCF Syntax

```
BEGIN MODEL entity_name  
INST "instance_name" U_SET=uset_name ;  
END;
```

## Use Internal VREF

The Use Internal Vref (USE\_INTERNAL\_VREF) constraint:

- Assigns a voltage value to the internal Vref feature for a given I/O bank.
- Frees the Vref pins of I/O banks from their function of providing a voltage reference.
- Allows you to specify the Vref pins for:
  - Vref
  - An alternative use

### Architecture Support

The Use Internal Vref constraint applies to Virtex®-6 devices only.

### Applicable Elements

The Use Internal Vref constraint can be specified for an instance, comp or net.

### Propagation Rules

- The Use Internal Vref constraint is illegal when attached to a net except when the net is connected to a pad. In this case, Use Internal Vref is treated as attached to the pad instance.
- When attached to a design element, Use Internal Vref applies to the entity to which it is attached.

### Constraint Values

- TRUE  
Turns *on* Use Internal Vref for a specific element.
- FALSE  
Turns *off* Use Internal Vref for a specific element.
- DONT\_CARE  
Allows the tools to determine the use of the Vref pin.

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

- Attribute Name  
USE\_INTERNAL\_VREF
- Attribute Values  
See *Constraint Values* above.

#### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
(* USE_INTERNAL_VREF = "{TRUE|FALSE|DONT_CARE}" *)
```

The default is DONT\_CARE.

### UCF and NCF Syntax

```
INST "instance_name" USE_INTERNAL_VREF={TRUE|FALSE|DONT_CARE};
```

The default is TRUE.

### XCF Syntax

```
MODEL "entity_name" use_internal_vref ={true|false|dont_care}
```

The default is TRUE.

## Use LUTNM

The Use LUTNM (USE\_LUTNM) constraint:

- Is an advanced mapping and placement constraint.
- Turns the [Lookup Table Name \(LUTNM\)](#) constraint *on* or *off* for a specific element or section of a set.

### Architecture Support

Applies to all FPGA devices and no CPLD devices.

### Applicable Elements

The Use LUTNM constraint applies to instances or macros that are members of sets.

### Propagation Rules

It is illegal to attach the Use LUTNM constraint to a net.

### Constraint Values

- TRUE (default)  
Turns *on* the constraint for a specific element
- FALSE  
Turns *off* the constraint for a specific element

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

- Attach to a member of a set
- Attribute Name  
USE\_LUTNM
- Attribute Values  
See *Constraint Values* above.

#### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute USE_LUTNM: string;
```

Specify the VHDL constraint as follows:

```
attribute USE_LUTNM of entity_name: entity is "{TRUE|FALSE}";
```

#### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
INST "instance_name" USE_LUTNM={TRUE|FALSE};
```

### XCF Syntax

```
MODEL "entity_name" use_lutnm={true|false};
```

## Use Relative Location

The Use Relative Location (USE\_RLOC) constraint:

- Is an advanced mapping and placement constraint.
- Turns the [Relative Location \(RLOC\)](#) constraint *on* or *off* for a specific element or section of a set.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

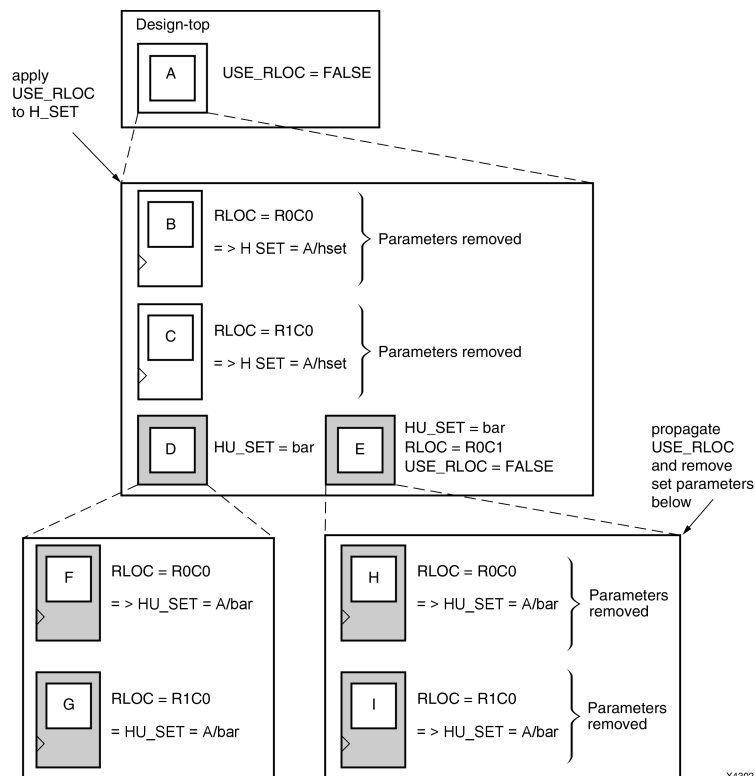
The Use Relative Location constraint applies to instances or macros that are members of sets.

## Propagation Rules

It is illegal to attach the Use Relative Location constraint to a net.

When attached to a design element, the [U Set](#) constraint propagates to all applicable elements in the hierarchy within the design element.

### Using USE\_RLOC to Control RLOC Application on H\_SET and HU\_SET Sets



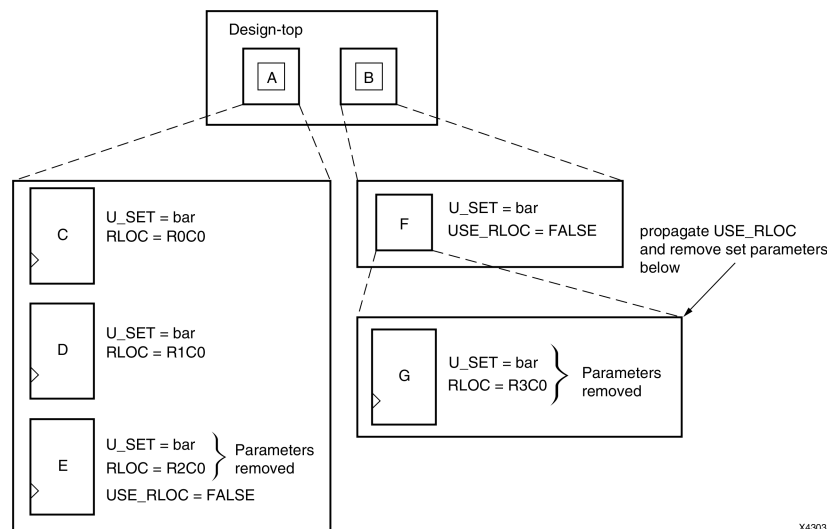
X4302

Applying the Use Relative Location constraint on U Set sets is a special case because of the lack of hierarchy in the U SET set. Because the Use Relative Location constraint propagates strictly in a hierarchical manner, the members of a U SET set that are in different parts of the design hierarchy must be tagged separately with the Use Relative Location constraint. No single Use Relative Location constraint is propagated to all the members of the set that lie in different parts of the hierarchy.

If you create a U SET set through an instantiating macro, you can attach the Use Relative Location constraint to the instantiating macro to allow it to propagate hierarchically to all the members of the set.

You can create this instantiating macro by placing the U SET constraint on a macro and letting the mapper propagate that constraint to every symbol with an RLOC constraint below it in the hierarchy.

### Using the USE\_RLOC Constraint to Control RLOC Application on U\_SET Sets



- `USE_RLOC=FALSE` on primitive E removes it from the U\_SET set.
- `USE_RLOC=FALSE` on element F propagates to primitive G and removes it from the U\_SET set.

While propagating the Use Relative Location constraint, the mapper ignores underlying Use Relative Location constraints if it encounters elements higher in the hierarchy that already have Use Relative Location constraints. For example, if the mapper encounters an underlying element with `USE_RLOC=TRUE` during the propagation of a `USE_RLOC=FALSE`, it ignores the newly encountered `TRUE` constraint.

## Constraint Values

- **TRUE (default)**  
Turns the **Relative Location (RLOC)** constraint *on* for a specific element.
- **FALSE**  
Turns the **Relative Location (RLOC)** constraint *off* for a specific element.



## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to a member of a set
- Attribute Name  
USE\_RLOC
- Attribute Values  
See *Constraint Values* above.

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute USE_RLOC: string;
```

Specify the VHDL constraint as follows:

```
attribute USE_RLOC of entity_name: entity is "{TRUE|FALSE}";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

```
INST "instance_name" USE_RLOC={TRUE|FALSE};
```

### XCF Syntax

```
MODEL "entity_name" use_rloc={true|false};
```

## Use Low Skew Lines

The Use Low Skew Lines (USELOWSKEWLINES) constraint:

- Is a PAR routing constraint.
- Specifies the use of low skew routing resources for any net.

You can use these resources for both internally-generated and externally-generated signals. Externally-generated signals are driven by IOB components.

The Use Low Skew Lines constraint on a net:

- Directs PAR to route the net on one of the low skew resources.
- Makes the timing tool automatically account for and report skew on register-to-register paths that utilize those low skew resources.

Specify the Use Low Skew Lines constraint only when all four primary global clocks have been used.

## Architecture Support

Applies to all FPGA devices and no CPLD devices.

## Applicable Elements

The Use Low Skew Lines constraint applies to nets.

## Propagation Rules

The Use Low Skew Lines constraint applies to the attached net.

## Constraint Values

- YES
- NO
- TRUE
- FALSE

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

- Attach to an output net
- Attribute Name  
USELOWSKEWLINES
- Attribute Values
  - TRUE
  - FALSE

### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute USELOWSKEWLINES: string;
```

Specify the VHDL constraint as follows:

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

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

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

### UCF and NCF Syntax

This statement forces net \$1I87/1N6745 to be routed on one of the device's low skew resources.

```
NET "$1I87/$1N6745" USELOWSKEWLINES;
```

### XCF Syntax

```
BEGIN MODEL "entity_name"
```

```
    NET "signal_name" uselowskewlines={yes|true};
```

```
END;
```

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

### PCF Syntax

Same as UCF and NCF syntax.

## VCCAUX

The VCCAUX (VCCAUX) constraint:

- Defines the voltage value of the VCCAUX pin for Spartan®-3A and Spartan-6 devices.
- Affects the banking rules for I/O placement in:
  - The automated placer
  - The PACE pin assignments software
- Affects the end-generated bitstream for the device.

### Architecture Support

The VCCAUX constraint supports the following devices:

- Spartan-3A
- Spartan-6

### Applicable Elements

- The VCCAUX constraint is a global attribute.
- The VCCAUX constraint is not attached to any particular element.

### Propagation Rules

Not applicable.

### Constraint Values

- 2.5
- 3.3

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### UCF and NCF Syntax

```
CONFIG VCCAUX="value";
```

```
CONFIG VCCAUX=3.3;
```

## VCCAUX\_IO

The auxiliary I/O (VCCAUX\_IO) supply rail:

- Is specific to the HP I/O banks only.
- Is used to power some of the I/O circuitry in the HP bank, including the single-ended and differential input buffer circuits.

The HP I/O banks contain:

- The VCCAUX\_IO pins.
- The regular VCCAUX pins that power the various internal block features.

In the Xilinx® 7 series FPGA device packages, the VCCAUX\_IO pins are connected together in groups of three to four I/O banks.

- The number of I/O banks that have their VCCAUX\_IO pins grouped together depends on the particular 7 series part and package combination.
- See the *7 Series Packaging and Pinout Guide* for banks that are grouped together for each part and package combination.

The [VCCAUX](#) and VCCAUX\_IO supplies must turn on before the VCCO supply. See the *7 Series FPGA Data Sheet* for more details regarding power supply requirements.

### Architecture Support

- Kintex™-7
- Virtex®-7

### Applicable Elements

For VCCAUX\_IO constraint applicable elements, see the *SelectIO™ User Guide*.

### Propagation Rules

For VCCAUX\_IO constraint propagation rules, see the *SelectIO User Guide*.

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

Attribute Name

VCCAUX\_IO

#### VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute VCCAUX_IO: string;
```

Specify the VHDL constraint as follows:

```
attribute VCCAUX_IO of {component_name | label_name} :  
{component | label} is "{NORMAL|HIGH|DONTCARE}";
```

### Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
( * VCCAUX_IO = {NORMAL|HIGH|DONTCARE} * )
```

### UCF and NCF Syntax

```
NET "net_name" VCCAUX_IO=( 0 | NORMAL | HIGH | DONTCARE ) ;  
INST "instance_name" VCCAUX_IO=( NORMAL | HIGH | DONTCARE ) ;
```

## Voltage

The Voltage (VOLTAGE) constraint:

- Is a timing constraint.
- Allows you to specify the operating voltage.

### Prorating

- Specifying the operating voltage allows the prorating of delay characteristics based on the specified voltage.
- Prorating is a scaling operation on existing speed file delays and is applied globally to all delays.
- Newer devices may not support Voltage prorating until the timing information (speed files) is marked as production status.

### Range of Supported Voltages

Each architecture has a specific range of supported voltages. If the specified voltage does not fall within the supported range:

- The constraint is ignored.
- An architecture-specific default value is used instead.
- An error message is displayed during static timing

### Architecture Support

- Spartan®-3A
- Spartan-3E
- Virtex®-4
- Virtex-5

### Applicable Elements

Applies globally to the entire design.

### Propagation Rules

This constraint is a design element constraint. Any attachment to a net is illegal.

### Constraint Values

- value  
A real number specifying the voltage
- V  
Volts (default voltage unit)

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### UCF and NCF Syntax

**VOLTAGE**=*value* [V];

The following statement specifies that the analysis for everything relating to speed file delays assumes an operating power of 5 volts.

**VOLTAGE**=5;

### Constraints Editor Syntax

For information on setting constraints in Constraints Editor, including syntax, see the Constraints Editor Help.

### PCF Syntax

Same as UCF and NCF syntax.



## Vcco Sense Mode (VCCOSENSEMODE)

The Vcco Sense Mode (VCCOSENSEMODE) constraint:

- Allows you to program special hardware in an HR (3.3V) I/O bank. This special hardware:
  - Detects when a VCCO voltage is applied to the bank that differs from the user-programmed voltage.
  - Can override the user-programmed voltage and change the bank to high voltage mode based upon the detected voltage.
- Allows you to program the HR bank independently of other banks.

### Architecture Support

- Artix™-7 devices
- Kintex™-7 devices

### Applicable Elements

This constraint is a global CONFIG constraint and is not attached to any instance or signal name.

### Propagation Rules

Applies to I/Os in the specified bank for the entire design.

### Constraint Values

- *n*  
The number of the bank.
- OFF  
The bank overvoltage detection circuit is disabled.
- ALWAYSACTIVE  
The bank overvoltage detection circuit is always enabled.
- FREEZE  
The bank overvoltage detection circuit is enabled only during the configuration sequence.

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### UCF and NCF Syntax

```
CONFIG VCCOSENSEMODE $n$ =[OFF|ALWAYSACTIVE|FREEZE];
```

#### UCF and NCF Syntax Example

```
CONFIG VCCOSENSEMODE5=OFF;
```

## VREF

The VREF (VREF) constraint:

- Applies as a global attribute.
- Does not apply directly to any element.
- Configures listed pins as VREF supply pins to be used with other I/O pins designated by an SSTL or HSTL I/O Standard.
- Is selectable on any I/O in designs for certain CoolRunner™-II devices.
- Allows you to select which pins are VREF pins.

**Note** Double-check pin assignments in the report (RPT) file.

The fitter automatically assigns sufficient VREF if:

- You do not specify any VREF pins for the differential I/O standards, HSTL and SSTL, or
- You do not specify sufficient VREF pins within the required proximity of differential I/O pins.

## Architecture Support

The VREF constraint applies only to CoolRunner-II devices with 128 macrocells and larger.

## Applicable Elements

Applies globally to the entire design.

## Propagation Rules

The VREF constraint configures listed pins as VREF supply pins to be used in conjunction with other I/O pins designated with one of the SSTL or HSTL I/O Standards.

## Constraint Values

- $Pnn$   
 $nn$   
numeric pin number
- $rc$ 
  - $r$   
alphabetic row
  - $c$   
numeric column

## Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

### Schematic Syntax

**VREF**=*value\_list*

on CONFIG symbol

### UCF and NCF Syntax

**CONFIG VREF**=*value\_list* ;

**CONFIG VREF**=P12,P13;

## Wire And

The Wire And (WIREAND) constraint:

- Is an advanced fitter constraint.
- Forces a tagged node to be implemented as a wired **AND** function in the interconnect (UIM and Fastconnect).

### Architecture Support

Supports XC9500 devices only.

### Applicable Elements

The Wire And constraint applies to any net.

### Propagation Rules

This constraint is a net constraint. Any attachment to a design element is illegal.

### Constraint Values

- YES
- NO
- TRUE
- FALSE

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

## XBLKNM

The XBLKNM (XBLKNM) constraint:

- Is an advanced mapping constraint.
- Assigns block names to qualifying primitives and logic elements.

If the same XBLKNM attribute is assigned to more than one instance, the software attempts to pack logic with the same block name into one or more slices. Conversely, two symbols with different XBLKNM names are not mapped into the same block. Placing the same XBLKNM constraints on instances that do not fit within one block creates an error.

Specifying identical XBLKNM attributes on FMAP symbols tells the software to group the associated function generators into a single slice. Using XBLKNM, you can partition a complete slice without constraining the slice to a physical location on the device.

Hierarchical paths are not prefixed to XBLKNM attributes, so XBLKNM attributes for different slices must be unique throughout the entire design.

The [Block Name](#) attribute allows any elements except those with a different BLKNM to be mapped into the same physical component. XBLKNM, however, allows only elements with the same XBLKNM to be mapped into the same physical component. Elements without an XBLKNM cannot be mapped into the same physical component as those with an XBLKNM.

XBLKNM can also be used with block RAM components.

### Architecture Support

Applies to all FPGA devices and no CPLD devices.

### Applicable Elements

For information about which design elements can be used with which device families, see the *Libraries Guides*. For more information, see the device [data sheet](#).

### Propagation Rules

Applies to the design element to which it is attached.

### Constraint Values

block\_name

A valid block name for that type of symbol

### Syntax Examples

The syntax examples in this section show how to use this constraint with particular tools or methods. If a tool or method is not listed, you cannot use this constraint with it.

#### Schematic Syntax

- Attach to a valid instance
- Attribute Name  
XBLKNM
- Attribute Values  
See *Constraint Values* above.

## VHDL Syntax

Declare the VHDL constraint as follows:

```
attribute XBLKNM: string;
```

Specify the VHDL constraint as follows:

```
attribute XBLKNM  
of {component_name | label_name}: {component | label} is block_name ;
```

## Verilog Syntax

Place the Verilog constraint immediately before the module or instantiation.

Specify the Verilog constraint as follows:

```
( * XBLKNM = "block_name" * )
```

## UCF and NCF Syntax

```
INST "instance_name" XBLKNM=block_name ;
```

The following statement assigns an instantiation of an element named **flip\_flop2** to a block named **U1358**.

```
INST "$1I87/flip_flop2" XBLKNM=U1358;
```

## XCF Syntax

```
BEGIN MODEL "entity_name"
```

```
    INST "instance_name" xblknm=xblknm_name ;
```

```
END;
```

## *Additional Resources*

---

- Xilinx Glossary - <http://www.xilinx.com/company/terms.htm>
- Xilinx Support and Documentation - <http://www.xilinx.com/support>