

Attributes, Constraints, and Carry Logic

Overview

***Information for Mentor
Customers***

Schematic Syntax

UCF/NCF File Syntax

***Attributes/Logical
Constraints***

Placement Constraints

***Relative Location (RLOC)
Constraints***

Timing Constraints

Physical Constraints

***Relationally Placed Macros
(RPM)***

***Carry Logic in XC4000
FPGAs***

***Carry Logic in XC5200
FPGAs***

Attributes, Constraints, and Carry Logic



The Xilinx logo shown above is a registered trademark of Xilinx, Inc.

XILINX, XACT, XC2064, XC3090, XC4005, XC5210, XC-DS501, FPGA Architect, FPGA Foundry, NeoCAD, NeoCAD EPIC, NeoCAD PRISM, NeoROUTE, Timing Wizard, and TRACE are registered trademarks of Xilinx, Inc.



The shadow X shown above is a trademark of Xilinx, Inc.

All XC-prefix product designations, XACTstep, XACTstep Advanced, XACTstep Foundry, XACT-Floorplanner, XACT-Performance, XAPP, XAM, X-BLOX, X-BLOX plus, XChecker, XDM, XDS, XEPLD, XPP, XSI, BITA, Configurable Logic Cell, CLC, Dual Block, FastCLK, FastCONNECT, FastFLASH, FastMap, Foundation, HardWire, LCA, LogiBLOX, Logic Cell, LogiCORE, LogicProfessor, MicroVia, PLUSASM, Plus Logic, Plustran, P+, PowerGuide, PowerMaze, Select-RAM, SMARTswitch, TrueMap, UIM, VectorMaze, VersaBlock, VersaRing, WeblINX, XABEL, Xilinx Foundation Series, and ZERO+ are trademarks of Xilinx, Inc. The Programmable Logic Company and The Programmable Gate Array Company are service marks of Xilinx, Inc.

All other trademarks are the property of their respective owners.

Xilinx, Inc. does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx, Inc. reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx, Inc. will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx, Inc. devices and products are protected under one or more of the following U.S. Patents: 4,642,487; 4,695,740; 4,706,216; 4,713,557; 4,746,822; 4,750,155; 4,758,985; 4,820,937; 4,821,233; 4,835,418; 4,855,619; 4,855,669; 4,902,910; 4,940,909; 4,967,107; 5,012,135; 5,023,606; 5,028,821; 5,047,710; 5,068,603; 5,140,193; 5,148,390; 5,155,432; 5,166,858; 5,224,056; 5,243,238; 5,245,277; 5,267,187; 5,291,079; 5,295,090; 5,302,866; 5,319,252; 5,319,254; 5,321,704; 5,329,174; 5,329,181; 5,331,220; 5,331,226; 5,332,929; 5,337,255; 5,343,406; 5,349,248; 5,349,249; 5,349,250; 5,349,691; 5,357,153; 5,360,747; 5,361,229; 5,362,999; 5,365,125; 5,367,207; 5,386,154; 5,394,104; 5,399,924; 5,399,925; 5,410,189; 5,410,194; 5,414,377; 5,422,833; 5,426,378; 5,426,379; 5,430,687; 5,432,719; 5,448,181; 5,448,493; 5,450,021; 5,450,022; 5,453,706; 5,466,117; 5,469,003; 5,475,253; 5,477,414; 5,481,206; 5,483,478; 5,486,707; 5,486,776; 5,488,316; 5,489,858; 5,489,866; 5,491,353; 5,495,196; 5,498,979; 5,498,989; 5,499,192; 5,500,608; 5,500,609; 5,502,000; 5,502,440; 5,504,439; 5,506,518; 5,506,523; 5,506,878; 5,513,124; 5,517,135; 5,521,835; 5,521,837; 5,523,963; 5,523,971; 5,524,097; 5,526,322; 5,528,169; 5,528,176; 5,530,378; 5,530,384; 5,546,018; 5,550,839; 5,550,843; 5,552,722; 5,553,001; 5,559,751; 5,561,367; 5,561,629; 5,561,631; 5,563,527; 5,563,528; 5,563,529; 5,563,827; 5,565,792; 5,566,123; 5,570,051; 5,574,634; 5,574,655; 5,578,946; 5,581,198; 5,581,199; 5,581,738; 5,583,450; 5,583,452; 5,592,105; 5,594,367; 5,598,424; 5,600,263; 5,600,264; 5,600,271; 5,600,597; 5,608,342; 5,610,536; 5,610,790; 5,610,829; 5,612,633; 5,617,021; 5,617,041; 5,617,327; 5,617,573; 5,623,387; 5,627,480; 5,629,637; 5,629,886; 5,631,577; 5,631,583; 5,635,851; 5,636,368; 5,640,106; 5,642,058; 5,646,545; 5,646,547; 5,646,564; 5,646,903; 5,648,732; 5,648,913; 5,650,672; 5,650,946; 5,652,904; 5,654,631; 5,656,950; 5,657,290; 5,659,484; 5,661,660; 5,661,685; 5,670,897; 5,670,896; RE 34,363, RE 34,444, and RE 34,808. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx, Inc. assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx, Inc. will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

Copyright 1991-1997 Xilinx, Inc. All Rights Reserved.

Preface

About This Manual

This manual lists and describes all the attributes that you can use with your design entry software and the constraints that are contained in machine- and user-generated files.

Before using this manual, you should be familiar with the operations that are common to all Xilinx's software tools: how to bring up the system, select a tool for use, specify operations, and manage design data. These topics are covered in the *Development System Reference Guide*.

Conventions

Typographical

This manual uses the following conventions. An example illustrates each convention.

- `Courier font` indicates messages, prompts, and program files that the system displays.

```
speed grade: -100
```

- **Courier bold** indicates literal commands that you enter in a syntactical statement.

```
rpt_del_net=
```

Courier bold also indicates commands that you select from a menu.

File → **Open**

- *Italic font* denotes the following items.
 - Variables in a syntax statement for which you must supply values
`edif2ngd design_name`
 - References to other manuals
See the *Development System Reference Guide* for more information.
 - Emphasis in text

If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.

- Square brackets “[]” indicate an optional entry or parameter. However, in bus specifications, such as bus [7:0], they are required.

```
edif2ngd [option_name] design_name
```

Square brackets also enclose footnotes in tables that are printed out as hardcopy in DynaText[®].

- Braces “{ }” enclose a list of items from which you choose one or more.

```
lowpwr = {on|off}
```

- A vertical bar “|” separates items in a list of choices.

```
symbol editor_name [bus|pins]
```

- A vertical ellipsis indicates repetitive material that has been omitted.

```
IOB #1: Name = QOUT'  
IOB #2: Name = CLKIN'  
.  
.  
.
```

- A horizontal ellipsis “. . .” indicates that an item can be repeated one or more times.

```
allow block block_name loc1 loc2 . . . locn;
```

Contents

Preface

About This Manual	iii
-------------------------	-----

Conventions

Typographical.....	v
--------------------	---

Attributes, Constraints, and Carry Logic

Overview	21
Attributes.....	21
Constraints.....	21
Logical Constraints	22
Physical Constraints	22
Carry Logic.....	22
Information for Mentor Customers	23
Schematic Syntax	23
UCF/NCF File Syntax.....	24
Wildcards	26
File Name.....	26
Instances and Blocks	26
Attributes/Logical Constraints	27
Syntax Summary.....	27
Attributes/Constraints Applicability.....	34
Macro and Net Propagation Rules.....	36
Syntax Descriptions	38
BASE	38
Applicable Elements	38
Description.....	38
Syntax.....	39
Example.....	39
BLKNM.....	39
Applicable Elements	39

Description	40
Syntax	40
Example	40
BUFG	41
Applicable Elements	41
Description	41
Syntax	41
Example	41
COLLAPSE	41
Applicable Elements	41
Description	41
Syntax	41
Example	41
CONFIG	42
Applicable Elements	42
Description	42
Syntax	42
Example	43
DECODE	43
Applicable Elements	43
Description	43
Syntax	43
Example	43
DIVIDE1_BY and DIVIDE2_BY	44
Applicable Elements	44
Description	44
Syntax	44
Examples	44
DOUBLE	44
Applicable Elements	44
Description	44
Syntax	45
Example	45
DRIVE	45
Applicable Elements	45
Description	45
Syntax	45
Example	45
DROP_SPEC	46
Applicable Elements	46
Description	46
Syntax	46

Example	46
EQUATE_F and EQUATE_G	46
Applicable Elements	46
Description	46
Syntax	46
Example	47
FAST	47
Applicable Elements	47
Description	47
Syntax	47
Example	47
FILE	48
Applicable Elements	48
Description	48
Syntax	48
HBLKNM	48
Applicable Elements	48
Description	49
Syntax	49
Example	49
HU_SET	50
Applicable Elements	50
Description	50
Syntax	51
Example	51
INIT	51
Applicable Elements	51
Description	51
Syntax	51
Example	51
INREG	52
Applicable Elements	52
Description	52
Syntax	52
Example	52
KEEP	52
Applicable Elements	52
Description	53
Syntax	53
Example	53
LOC	53
Applicable Elements	53

Description for FPGAs	54
Description for CPLDs	54
Location Types	55
Syntax for FPGAs	56
Syntax for CPLDs	57
Examples	58
MAP	58
Applicable Elements	58
Description	59
Syntax	59
Example	59
MAXDELAY	59
Applicable Elements	59
Description	59
Syntax	59
Example	60
MAXSKEW	60
Applicable Elements	60
Description	60
Syntax	60
Example	60
MEDDELAY	60
Applicable Elements	60
Description	61
Syntax	61
Example	61
NODELAY	61
Applicable Elements	61
Description	61
Syntax	61
Example	62
NOREDUCE	62
Applicable Elements	62
Description	62
Syntax	62
Example	62
OFFSET	62
Applicable Elements	62
Description	63
Syntax	63
Example	63
OPT Effort	63

Applicable Elements	63
Description	63
Syntax	63
Example	64
OPTIMIZE	64
Applicable Elements	64
Description	64
Syntax	64
Example	64
OUTREG	64
Applicable Elements	64
Description	65
Syntax	65
Example	65
PART	65
Applicable Elements	65
Description	65
Syntax	65
Example	66
PERIOD	66
Applicable Elements	66
Description	66
Syntax	66
Example	67
PROHIBIT	67
Applicable Elements	67
Description	67
Location Types	67
Syntax	68
Example	69
PWR_MODE	69
Applicable Elements	69
Description	69
Syntax	69
Example	69
RLOC	70
Applicable Elements	70
Description	70
Syntax	70
Example	71
RLOC_ORIGIN	71
Applicable Elements	71

Description	71
Syntax	71
Example	72
RLOC_RANGE	72
Applicable Elements	72
Description	72
Syntax	72
Example	72
S(ave) - Net Flag Attribute	73
Applicable Elements	73
Description	73
Syntax	73
Example	73
SLOW	73
Applicable Elements	73
Description	73
Syntax	74
Example	74
TIG	74
Applicable Elements	74
Description	74
Syntax	74
Example	74
Time Group Attributes	75
Applicable Elements	75
Description	75
Syntax	75
Example 1	76
Example 2	76
TNM	76
Applicable Elements	76
Description	76
Syntax	77
Example	77
TPSYNC	77
Applicable Elements	77
Description	77
Syntax	78
Example	78
TPTHRU	78
Applicable Elements	78
Description	79

Syntax	79
Example	79
TSidentifier	79
Applicable Elements	79
Description	79
Syntax	79
Example	81
U_SET	82
Applicable Elements	82
Description	82
Syntax	82
Example	83
USE_RLOC	83
Applicable Elements	83
Description	83
Syntax	83
Example	83
WIREAND	83
Applicable Elements	83
Description	84
Syntax	84
Example	84
Placement Constraints	85
BUFT Constraint Examples	85
CLB Constraint Examples	87
Edge Decoder Constraint Examples (XC4000 Only)	89
Flip-Flop Constraint Examples	91
Global Buffer Constraint Examples (XC4000, XC5200)	92
I/O Constraint Examples	92
IOB Constraint Examples	93
Mapping Constraint Examples	93
CLBMAP (XC3000 Only)	94
FMAP and HMAP (XC4000, XC5200)	95
ROM and RAM Constraint Examples	97
Relative Location (RLOC) Constraints	99
Benefits and Limitations of RLOC Constraints	99
Guidelines for Specifying Relative Locations	99
RLOC Sets	101
U_SET	102
H_SET	102
HU_SET	102
RLOC Set Summary	104

Set Linkage	106
Set Modification	108
The Effect of the Hierarchy on Set Modification	108
Separating Elements from H_SET Sets	110
Set Modifiers	112
Using RLOCs with Xilinx Macros	113
LOC Propagation through Design Flattening	113
Fixing Members of a Set at Exact Die Locations	114
Specifying a Range or Area	114
Toggling the Status of RLOC Constraints	115
Choosing an RLOC Origin when Using Hierarchy Sets	117
Timing Constraints	120
TNM Attributes	120
TNMs on Nets	120
TNMs on Macro or Primitive Pins	120
TNMs on Primitives	120
TNMs on Macro Symbols	120
TIMEGRP Constraints	121
TIMESPEC Constraints	122
Basic Form	122
Defining Intermediate Points on a Path	123
Worst Case Allowable Delay (MAXDELAY)	123
Linked Specifications	123
Defining Priority for Equivalent Level Specifications	124
Ignoring Paths	124
Ignoring Paths Through Primitives	124
Defining a Clock Period	125
Specifying Derived Clocks	126
Controlling Net Skew	126
Controlling Net Delay	127
Physical Constraints	128
PCF File Syntax	128
Syntax Descriptions	128
COMPGRP	128
Description	128
Syntax	129
FREQUENCY	129
Description	129
Syntax	129
INREG	129
Description	129
Syntax	129

LOCATE.....	130
Description.....	130
Syntax.....	130
LOCK.....	130
Description.....	130
Syntax.....	130
MAXDELAY.....	131
Description.....	131
Syntax.....	131
MAXSKEW.....	131
Description.....	131
Syntax.....	132
OFFSET.....	132
Description.....	132
Syntax.....	132
OUTREG.....	132
Description.....	132
Syntax.....	132
PATH.....	133
Description.....	133
Syntax.....	133
PENALIZE TILDE.....	133
Description.....	133
Syntax.....	133
PERIOD.....	133
Description.....	133
Syntax.....	133
PIN.....	134
Description.....	134
Syntax.....	134
PRIORITIZE.....	134
Description.....	134
Syntax.....	135
PROHIBIT.....	135
Description.....	135
Syntax.....	135
SITEGRP.....	135
Description.....	135
Syntax.....	135
TIMEGRP (Timing Group).....	136
Description.....	136
Syntax.....	136

TIG (Timing Ignore).....	137
Description	137
Syntax	137
TSidentifier	137
Description	137
Syntax	137
Relationally Placed Macros (RPMs).....	139
Carry Logic in XC4000 FPGAs	140
Differences Between XC4000E and XC4000X Carry Logic.....	140
Carry Logic Primitives and Symbols	141
Carry Logic Handling	143
Carry Mode Configuration Mnemonics	143
Carry Logic Configurations	144
ADD-F-CI	144
ADD-FG-CI	144
ADD-G-F1	145
ADD-G-CI	145
ADD-G-F3-	145
SUB-F-CI	146
SUB-FG-CI	146
SUB-G-1	146
SUB-G-CI.....	147
SUB-G-F1	147
SUB-G-F3-	147
ADDSUB-F-CI.....	148
ADDSUB-FG-CI	148
ADDSUB-G-CI	148
ADDSUB-G-F1	149
ADDSUB-G-F3-	149
INC-F-CI	150
INC-FG-1	150
INC-FG-CI.....	150
INC-G-1	151
INC-G-F1	151
INC-G-CI	151
INC-G-F3-	152
DEC-F-CI	152
DEC-FG-0.....	152
DEC-FG-CI	153
DEC-G-0	153
DEC-G-CI	153
DEC-G-F1	154

DEC-G-F3-.....	154
INCDEC-F-CI.....	154
INCDEC-FG-1.....	155
INCDEC-FG-CI.....	155
INCDEC-G-0.....	155
INCDEC-G-CI.....	156
INCDEC-G-F1.....	156
FORCE-0.....	157
FORCE-1.....	157
FORCE-CI.....	157
FORCE-F1.....	157
FORCE-F3-.....	157
FORCE-G4.....	157
EXAMINE-CI.....	157
Carry Logic in XC5200 FPGAs.....	158
XC5200 Carry Logic Library Support.....	158
Cascade Function.....	159

Attributes, Constraints, and Carry Logic

This chapter lists and describes all the attributes that you can use with your design entry software and the constraints that are contained in machine- and user-generated files.

This chapter contains the following major sections.

- “Overview” section
- “Information for Mentor Customers” section
- “Schematic Syntax” section
- “UCF/NCF File Syntax” section
- “Attributes/Logical Constraints” section. This section contains alphabetical listings of the attributes and constraints as well as descriptions, syntax, and examples of each constraint.
- “Placement Constraints” section
- “Relative Location (RLOC) Constraints” section
- “Timing Constraints” section
- “Physical Constraints” section
- “Relationally Placed Macros (RPMs)” section
- “Carry Logic in XC4000 FPGAs” section
- “Carry Logic in XC5200 FPGAs” section

Overview

Attributes

Attributes are instructions placed on symbols or nets in an FPGA or CPLD schematic to indicate their placement, implementation, naming, directionality, and so forth. This information is used by the design implementation software during placement and routing of a design. All the attributes listed in this chapter are available in the schematic entry tools directly supported by Xilinx unless otherwise noted, but some may not be available in textual entry methods such as VHDL.

Attributes applicable only to a certain schematic entry tool are described in the documentation for that tool. For third-party interfaces, consult the interface user guides for information on which attributes are available and how they are used.

Refer to the “Schematic Syntax” section in this chapter for guidelines on placing attributes on symbols on a schematic.

Constraints

Constraints, which are a type, or subset, of attributes, indicate where an element should be placed.

Logical Constraints

Constraints that are attached to elements in the design prior to mapping technology are referred to as logical constraints. Applying logical constraints helps you to adapt your design's performance to expected worst-case conditions. Later, when you choose a specific Xilinx architecture and place and route your design, the logical constraints are converted into physical constraints.

You can attach logical constraints using attributes in the input design, which are written into the Net Constraints File (NCF), or with a User Constraints File (UCF). Refer to the “UCF/NCF File Syntax” section for the rules for entering constraints in a UCF or NCF file.

Three categories of logical constraints are described in detail in the “Attributes/Logical Constraints” section: placement constraints, relative location constraints, and timing constraints.

The “Placement Constraints” section gives examples showing how to place constraints on the various types of logic elements in FPGA designs.

For FPGAs, relative location constraints (RLOCs) group logic elements into discrete sets and 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. Refer to the “Relative Location (RLOC) Constraints” section for detailed information on RLOCs.

Timing constraints allow you to specify the maximum allowable delay or skew on any given set of paths or nets in your design. Refer to the “Timing Constraints” section for detailed information on using timing constraints in a UCF file.

Physical Constraints

Constraints can also be attached to the elements in the physical design, that is, the design after mapping has been performed. These constraints are referred to as physical constraints and are defined in the Physical Constraints File (PCF), which is created during mapping. (See the “Physical Constraints” section).

Note: It is preferable to place any user-generated constraint in the UCF file — *not* in an NCF or PCF file.

Carry Logic

Dedicated fast carry logic increases the efficiency and performance of adders, subtractors, accumulators, comparators, and counters. See the “Carry Logic in XC4000 FPGAs” section and “Carry Logic in XC5200 FPGAs” section at the end of this chapter.

Information for Mentor Customers

In some software programs, particularly Mentor Graphics, attributes are called properties, but their functionality is the same as that of attributes. In this document, they are referred to as attributes.

There are two types of Mentor Graphics properties. In one, a property is equal to a value, for example, LOC=AA. In the other, the property name and the value are the same, for example, DECODE. In the first type, “attribute” refers to the property. In the second, “attribute” refers to the property and the value.

The Mentor netlist writer program (ENWRITE) converts all property names to lowercase letters, and the Xilinx netlist reader EDIF2NGD then converts the property names to uppercase letters. Because property names are processed in this way, you must enter the variable text of certain constraints in uppercase letters only. The most salient examples are:

- a **TSidentifier** name should contain only uppercase letters on a Mentor Schematic (TSMMAIN, for example, but not TSmain or TSMmain). Also, if a **TSidentifier** name is referenced in a property value, it must be entered in uppercase letters. For example, the TSID1 in the second constraint below must be entered in uppercase letters to match the TSID1 name in the first constraint.

```
TSID1 = FROM: gr1: TO: gr2: 50;
TSMMAIN = FROM: here: TO: there: TSID1: /2;
```

- Group names should contain only uppercase letters on a Mentor schematic (MY_FLOPS, for example, but not my_flops or My_flops).
- If a group name appears in a property value, it must also be expressed in uppercase letters. For example, the GROUP3 in the first constraint below must be entered in uppercase letters to match the GROUP3 in the second constraint.

```
TIMEGRP GROUP1 = gr2: GROUP3;
TIMEGRP GROUP3 = FFS: except: grp5;
```

Schematic Syntax

This section describes how to place attributes on symbols on a schematic. The following guidelines are for specifying locations.

- To specify a single location, use the following syntax.

attribute=location

- To specify multiple locations, use the following syntax.

attribute=location,location,location

A comma separates locations in a list of locations. (Spaces are ignored if entered.)

When you specify a list of locations, PAR (Place and Route) chooses any one of the listed locations.

- To define a range by specifying the two corners of a bounding box, use the following syntax.

attribute=location: location [SOFT]

A colon is only used to separate the corners of a bounding box. The logic represented by the symbol is placed somewhere inside the bounding box. If SOFT is specified, PAR may place the attribute elsewhere to obtain better results.

Examples of location attributes:

LOC=CLB_R1C2	Locates the element in the CLB in the first row, second column.
LOC=CLBR1C2.LC3	Locates the element in the top-most slice of the four slices of the XC5200 CLB located in the first row, second column.
LOC=P12	Assigns the signal to the pin P12 (for CPLDs).
RLOC=R4C4	Assigns the location relative to other elements in the set to row 4, column 4.
RLOC=R0C1.FFX	Assigns the location relative to other elements in the set to the X flip-flop in row 0, column 1.
RLOC=R2C3.LC0	Assigns the location of the element to be one slice below another element in the set that has an RLOC=R2C3.LC1 attribute. (XC5200)
BLKNM=U1358	Assigns the LCA block to the CLB named U1358.
RLOC_ORIGIN=R4C4	Fixes the reference CLB of a designated set at row 4, column 4.
RLOC_RANGE=R4C4 : R10C10	Limits the members of a designated set to a range between row 4, column 4 and row 10, column 10.

A name can be composed of any ASCII character except for control characters and characters that have special meanings.

Control characters are : (colon) ; (semi-colon) , (comma) and " (double quotes).

Characters with special meaning are / (forward slash) used as the hierarchy separator and . (period) used as the pin separator and for extensions.

For additional propagation rules for each constraint and attribute, refer to the "Macro and Net Propagation Rules" table in the "Attributes/Logical Constraints" section.

UCF/NCF File Syntax

Logical constraints are found in a Netlist Constraint File (NCF), an ASCII file generated by synthesis programs, and in a User Constraint File (UCF), an ASCII file generated by the user. This section describes the rules for entering constraints in a UCF or NCF file.

Note: It is preferable to place any user-generated constraint in the UCF file — *not* in an NCF or PCF file.

The UCF and NCF files are case sensitive. Identifier names (names of objects in the design, 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 either all upper-case or all lower-case letters; mixed case is not allowed.

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.

You can add comments to the UCF/NCF file by beginning each comment line with a pound (#) sign. Following is an example of part of a UCF/NCF file containing comments.

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

Statements do not have to be placed in any particular order in the UCF/NCF file.

The constraints in the UCF/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 or synthesis file or in the UCF/NCF files. If the constraints overlap, however, UCF/NCF constraints override the schematic constraint.

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

The syntax for constraints in the UCF/NCF files is as follows.

```
{NET | INST | PIN} full_name constraint ;
or
SET set_name set_constraint ;
```

where

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 or FAST, and so forth.

set_name is the name of an RLOC set. Refer to the “RLOC Sets” section for more information.

set_constraint is an RLOC_ORIGIN or RLOC_RANGE constraint.

Note: To specify attributes for the CONFIG or TIMEGRP primitives (tables), you must prepend the attribute definitions in the constraints files with the keywords CONFIG or TIMEGRP, respectively, as shown in the following examples.

```
CONFIG PROHIBIT=CLB_R6C8 ;
TIMEGRP input_pads=pads EXCEPT output_pads ;
```

For the TIMESPEC primitive (table), the use of the keyword TIMESPEC in the constraints files is optional.

Note: In all of the constraints files (NCF, UCF, and PCF), instance or variable names that match internal reserved words will be 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 RLOC_RANGE=R4C2 : R5C5 ;
```

Following is the recommended way to enter the constraint.

```
NET "net" RLOC_RANGE=R4C2 : R5C5 ;
or
NET "$SIG_0" RLOC_RANGE=R4C2 : R5C5 ;
```

Wildcards

You can use the wildcard (*) character in constraint statements as follows.

In an instance name, a wildcard character by itself represents every symbol of the appropriate type. For example, the following constraint initializes an entire set of ROMs to a particular hexadecimal value, 5555.

```
INST $1I3*/ROM2 INIT=5555 ;
```

If the wildcard character is used as part of a longer instance name, the wildcard represents one or more characters at that position.

In a location, you can use a wildcard character for either the row number or the column number. For example, the following constraint specifies placement of any instance under the hierarchy of loads_of_logic in any CLB in column 8.

```
INST /loads_of_logic/* LOC=CLB_r*c8 ;
```

Wildcard characters can be used in dot extensions.

```
CLB_R1C3.*
```

Wildcard characters cannot be used for both the row number and the column number in a single constraint, since such a constraint is meaningless.

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 will automatically read in an NCF file that has the same name as any one of the input XNF/EDIF/NGO files.

Instances and Blocks

Because the statements in the constraints file concern instances and blocks, these entities are defined here.

An *instance* is a symbol on the schematic. An *instance name* is the symbol name as it appears in the EDIF or XNF netlist. A *block* is a CLB, an IOB, or a TBUF. You can specify the *block name* by using the BLKNM or the HBLKNM attribute; by default, the software assigns a block name on the basis of a signal name associated with the block.

Attributes/Logical Constraints

Syntax Summary

This section summarizes attribute and logical constraints syntax. This syntax conforms to the conventions given in the “Conventions” section. A checkmark (√) appearing in a column means that the attribute/constraint is supported for architectures that use the indicated library. (Refer to the “Applicable Architectures” section of the “Xilinx Unified Libraries” chapter for information on the specific device architectures supported in each library.) A blank column means that the attribute/constraint is not supported for architectures using that library.

BASE	BASE = {F FG FGM IO}				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√				

BLKNM	BLKNM = <i>block_name</i>				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

BUFG	BUFG = {CLK OE SR}				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
					√

COLLAPSE	COLLAPSE				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
					√

CONFIG*	CONFIG = <i>tag value</i> [<i>tag value</i>]				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√				

*The CONFIG attribute configures internal options of an XC3000 CLB or IOB. Do not confuse this attribute with the CONFIG primitive, which is a table containing PROHIBIT and PART attributes.

DECODE	DECODE				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
		√	√		

DIVIDE1_BY and DIVIDE2_BY	DIVIDE1_BY = {4 16 64 256} DIVIDE2_BY = {2 8 32 128 1024 4096 16384 65536}				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
				√	

DOUBLE	DOUBLE				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√		

DRIVE	DRIVE = {12 24}				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
			√*		

* supported on the XC4000XV only

DROP_SPEC	DROP_SPEC=TSidentifier				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	√

EQUATE_F and EQUATE_G	EQUATE_F = equation EQUATE_G = equation				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√				

FAST	FAST				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	√

FILE	FILE = file_name [.extension]				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	√

HBLKNM	HBLKNM = block_name				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

HU_SET	HU_SET = <i>set_name</i>				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
		√	√	√	

INIT	INIT = {S R <i>value</i> }				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√		√

INREG	INREG				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
				√	

KEEP	KEEP				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	√

LOC	FPGAs: LOC= <i>location</i> : <i>location</i> [SOFT] or: LOC= <i>location1</i> ... , <i>locationn</i> CPLDs: LOC = { <i>pin_name</i> / FB <i>nn</i> / FB <i>nn_mm</i> }				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	√

MAP	MAP = [PUC PUO PLC PLO]*				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

*Only PUC and PUO are observed. PLC and PLO are translated to PUC and PUO, respectively. The default is PUO.

MAXDELAY	MAXDELAY = <i>allowable_delay</i> [units]				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

MAXSKEW	MAXSKEW = <i>allowable_delay</i> [units]				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

MEDDELAY	MEDDELAY				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
			√		

NODELAY	NODELAY				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
		√	√		

NOREDUCE	NOREDUCE				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√				√

OFFSET	OFFSET {IN OUT} <i>offset_time</i> {BEFORE AFTER} [clk_net]				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	√

OPT Effort	OPT Effort= {NORMAL HIGH}				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

OPTIMIZE	OPTIMIZE = {AREA SPEED BALANCE OFF}				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

OUTREG	OUTREG				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
				√	

PART	PART = <i>part_type</i>				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	√

PERIOD	PERIOD = <i>period</i> [units] {HIGH LOW} [<i>high_or_low_time hi_lo_units</i>] or: TSidentifier=PERIOD TNM_reference period [units] {HIGH LOW} [<i>high_or_low_time hi_lo_units</i>]				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	√

PROHIBIT	PROHIBIT = <i>location : location</i> or: PROHIBIT = <i>location1... , locationn</i>				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

PWR_MODE	PWR_MODE = {LOW STD}				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
					√

RLOC	For XC4000: RLOC = <i>RmCn[.extension]</i> For XC5200: RLOC = <i>RmCn.extension</i>				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
		√	√	√	

RLOC_ORIGIN	RLOC_ORIGIN = <i>RmCn</i>				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
		√	√	√	

RLOC_RANGE	RLOC_RANGE = <i>Rm1Cn1:Rm2Cn2</i>				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
		√	√	√	

S(ave) - Net Flag Attribute	S				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

SLOW	SLOW				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
					√

TIG	TIG or: TIG= <i>TSidentifier1</i> [, <i>TSidentifier2</i> , ... <i>TSidentifiern</i>]				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

Time Group Attributes	<i>new_group_name</i> =[RISING FALLING] <i>group_name1</i> [EXCEPT <i>group_name2</i> ... <i>group_namen</i>] or: <i>new_group_name</i> =[TRANSHI TRANSLO] <i>group_name1</i> [EXCEPT <i>group_name2</i> ... <i>group_namen</i>]				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

TNM	TNM = { <i>name</i> RAMS PADS LATCHES FFS <i>name</i> }				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	√

TPSYNC	TPSYNC = <i>identifier</i>				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

TPTHRU	TPTHRU = <i>identifier</i>				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	

TSidentifier	<p>TSidentifier=[MAXDELAY] FROM source_group TO dest_group allowable_delay [units] or: TSidentifier=FROM source_group TO dest_group allowable_delay [units] or: TSidentifier=FROM source_group THRU thru_point [THRU thru_point1... thru_pointn] TO dest_group allowable_delay [units] or: TSidentifier=FROM source_group TO dest_group another_TSid[/ *] number or: TSidentifier=PERIOD TNM_reference period [units] [{HIGH LOW} high_or_low_time [hi_lo_units]] or: TSidentifier=PERIOD TNM_reference another_PERIOD_identifier [/ *] number [{HIGH LOW} high_or_low_time [hi_lo_units]] or: TSidentifier=FROM source_group TO dest_group TIG or: TSidentifier=FROM source_group THRU thru_point [THRU thru_point1... thru_pointn] TO dest_group TIG NOTE: The use of a colon (:) instead of a space as a separator is optional.</p>				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	√	√

U_SET	U_SET = name				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
		√	√	√	

USE_RLOC	USE_RLOC = {TRUE FALSE}				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
		√	√	√	

WIREAND	WIREAND				
	XC3000	XC4000E	XC4000X	XC5200	XC9000
					√

Attributes/Constraints Applicability

Certain constraints can only be defined at the design level, whereas other constraints can be defined in the various configuration files. The following table lists the constraints and their applicability to the design, and the NCF, UCF, and PCF files. A check mark (√) indicates that the constraint applies to the item for that column.

Table 0-1 Constraint Applicability Table

Attribute/Constraint	Design	NCF	UCF	PCF
BASE	√			
BLKNM	√	√	√	
BUFG	√	√	√	
COLLAPSE	√	√	√	
COMPGRP				√
CONFIG**	√			
DECODE	√	√	√	
DIVIDE1_BY	√	√		
DIVIDE2_BY	√	√		
DOUBLE	√			
DRIVE	√	√	√	
DROP_SPEC		√	√	√*
EQUATE_F	√			
EQUATE_G	√			
FAST	√	√	√	
FILE	√			
FREQUENCY				√
HBLKNM	√	√	√	
HU_SET	√	√	√	
INIT	√	√		
INREG	√	√	√	√
KEEP	√	√	√	
LOC	√	√	√	√*
LOCATE				√
LOCK				√
MAP	√	√	√	
MAXDELAY	√	√	√	√*
MAXSKEW	√	√	√	√*
MEDDELAY	√	√	√	
NODELAY	√	√	√	
NOREDUCE	√	√	√	
OFFSET		√	√	√*

Table 0-1 Constraint Applicability Table

Attribute/Constraint	Design	NCF	UCF	PCF
OPT_EFFORT	√	√	√	
OPTIMIZE	√	√	√	
OUTREG	√	√	√	√
PATH				√
PART	√	√	√	
PENALIZE TILDE				√
PERIOD	√	√	√	√*
PIN				√
PRIORITIZE				√
PROHIBIT	√	√	√	√*
PWR_MODE	√	√	√	
RLOC	√	√	√	
RLOC_ORIGIN	√	√	√	√
RLOC_RANGE	√	√	√	√
S(ave) - Net Flag attribute	√	√	√	
SITEGRP				√
SLOW	√	√	√	
TIG	√	√	√	√*
Time group attributes	√	√	√	√
TNM	√	√	√	
TPSYNC	√	√	√	
TPTHRU	√	√	√	
TSidentifier	√	√	√	√*
U_SET	√	√	√	
USE_RLOC	√	√	√	
WIREAND	√	√	√	

*Use cautiously — while constraint is available, there are differences between the UCF/NCF and PCF syntax.

**The CONFIG attribute configures internal options of an XC3000 CLB or IOB. Do not confuse this attribute with the CONFIG primitive, which is a table containing PROHIBIT and PART attributes.

Macro and Net Propagation Rules

Not all constraints can be attached to nets and macros. The following table lists the constraints and stipulates whether they can be attached to a net, a macro, or neither.

Table 0-2 Macro and Net Propagation Rules

Constraint/Attribute	Action when attached to a net	Action when attached to a macro
BASE	illegal	illegal
BLKNM	Note 1	Note 4
BUFG	Note 3	illegal
COLLAPSE	Note 3	illegal
CONFIG*	illegal	illegal
DECODE	Note 1	Note 4
DIVIDE1_BY and DIVIDE2_BY	Note 1	Note 4
DOUBLE	Note 1	Note 4
DRIVE	Note 6	Note 4
DROP_SPEC	illegal	illegal
EQUATE_F and EQUATE_G	illegal	illegal
FAST	Note 6	Note 4
FILE	illegal	Note 5
HBLKNM	Note 1	Note 4
HU_SET	Note 1	Note 4
INIT	FPGA: illegal CPLD: Note 1	Note 4
INREG	illegal	illegal
KEEP	Note 3	illegal
LOC	FPGA: Note 6 CPLD: Note 1	Note 4
MAP	illegal	illegal
MAXDELAY	Note 3	illegal
MAXSKEW	Note 3	illegal
MEDDELAY	Note 6	Note 4
NODELAY	Note 6	Note 4
NOREDUCE	Note 3	illegal
OFFSET	illegal	illegal
OPTIMIZE	illegal	Note 5
OPT_EFFORT	illegal	Note 5
OUTREG	illegal	illegal
PART	illegal	illegal
PERIOD	Note 3	illegal

Table 0-2 Macro and Net Propagation Rules

Constraint/Attribute	Action when attached to a net	Action when attached to a macro
PROHIBIT	illegal	illegal
PWR_MODE	Note 2	Note 4
RLOC	illegal	Note 4
RLOC_ORIGIN	illegal	Note 4
RLOC_RANGE	illegal	Note 4
S(ave) - Net Flag Attribute	Note 3	illegal
SLOW	Note 6	Note 4
TIG	Note 2	Note 4
Time Group Attributes	illegal	illegal
TNM	Note 2	Note 4
TPSYNC	Note 3	illegal
TSidentifier	illegal	illegal
U_SET	illegal	Note 4
USE_RLOC	illegal	Note 4
WIREAND	Note 3	illegal

Note 1: Attaches to all applicable elements that drive the net.

Note 2: The attribute has a net form and so no special propagation is required.

Note 3: Attribute is a net attribute and any attachment to a macro is illegal.

Note 4: Propagated to all applicable elements in the hierarchy below the module.

Note 5: Attribute is a macro attribute and any attachment to a net is illegal.

Note 6: Attribute is illegal when attached to a net except when the net is connected to a pad. In this case, the attribute is treated as attached to the pad instance.

*The CONFIG attribute configures internal options of an XC3000 CLB or IOB. Do not confuse this attribute with the CONFIG primitive, which is a table containing PROHIBIT and PART attributes.

Syntax Descriptions

The information that follows describes in alphabetical order the attributes and constraints. A checkmark (√) appearing in a column means that the attribute/constraint is supported for architectures that use the indicated library. (Refer to the “Applicable Architectures” section of the “Xilinx Unified Libraries” chapter for information on the specific device architectures supported in each library.) A blank column means that the attribute/constraint is not supported for architectures that use that library.

The description for each attribute contains a subsection entitled “Applicable Elements.” This section describes the base primitives and circuit elements to which the constraint or attribute can be attached. In many cases, constraints or attributes can be attached to macro elements, in which case some resolution of the user’s intent is required. Refer to the “Macro and Net Propagation Rules” section for a table describing the additional propagation rules for each constraint and attribute.

BASE

XC3000	XC4000E	XC4000X	XC5200	XC9000
√				

Applicable Elements

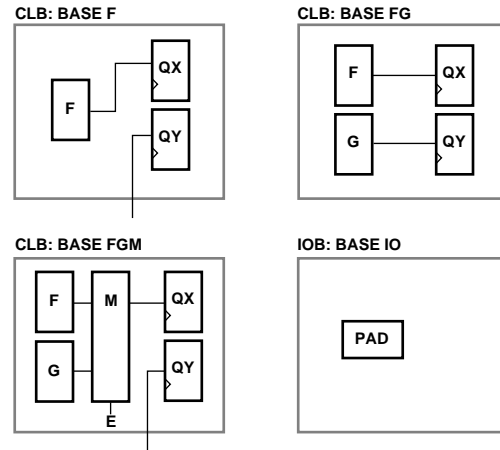
CLB or IOB primitives

Description

The BASE attribute defines the base configuration of a CLB or an IOB. For an IOB primitive, it should always be set to IO. For a CLB primitive, it can be one of three modes in which the CLB function generator operates.

- F mode allows the CLB to implement any one function of up to five variables.
- FG mode gives the CLB any two functions of up to four variables. Of the two sets of four variables, one input (A) must be common, two (B and C) can be either independent inputs or feedback from the Qx and Qy outputs of the flip-flops within the CLB, and the fourth can be either of the two other inputs to the CLB (D and E).
- FGM mode is similar to FG, but the fourth input must be the D input. The E input is then used to control a multiplexer between the two four-input functions, allowing some six- and seven-input functions to be implemented.

CLB and IOB base configurations of the XC3000 family are illustrated in the “IOB and CLB Primitives for Base Configurations XC3000” figure. In this drawing, BASE F, FG, and FGM are for CLBs; BASE IO is for IOBs.



X4708

Figure 0-1 IOB and CLB Primitives for Base Configurations XC3000

In a schematic, BASE can be attached to any valid instance. Not supported for UCF, NCF, or PCF files.

Syntax

BASE=mode

where *mode* can be F, FG, or FGM for a CLB and IO for an IOB.

Example

Schematic

Attached to a valid instance.

UCF/NCF file

N/A

BLKNM

XC3000	XC4000E	XC4000X	XC5200	XC9000
√ 1, 2, 3, 7, 8	√ 2, 3, 4, 5, 7, 8, 9, 10, 11	√ 2, 3, 4, 5, 7, 8, 9, 10, 11	√ 2, 3, 4, 6, 7	

Applicable Elements

1. IOB, CLB and CLBMAP (See Note below)
2. Flip-flop and latch primitives
3. Any I/O element or pad
4. FMAP
5. HMAP
6. F5MAP

7. BUFT
8. ROM primitive
9. RAM primitives
10. RAMS and RAMD primitives
11. Carry logic primitives

Note: You can also attach the BLKNM constraint to the net connected to the pad component in a 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
```

Description

Assigns LCA block names to qualifying primitives and logic elements. If the same BLKNM attribute is assigned to more than one instance, the software attempts to map them into the same LCA block. Conversely, two symbols with different BLKNM names are not mapped into the same block. Placing similar BLKNMs on instances that do not fit within one LCA block creates an error.

Specifying identical BLKNM attributes on FMAP and/or HMAP symbols tells the software to group the associated function generators into a single CLB. Using BLKNM, you can partition a complete CLB without constraining the CLB to a physical location on the device.

BLKNM attributes, like LOC constraints, are specified from the schematic. Hierarchical paths are not prefixed to BLKNM attributes, so BLKNM attributes for different CLBs must be unique throughout the entire design. See the section on the HBLKNM attribute for information on attaching hierarchy to block names.

For XC5200, a given BLKNM string can only be used to group a logic cell (LC), which contains one register, one LUT (FMAP), and one F5_MUX element. An error will occur if two or more registers, two or more FMAPs, or two or more F5_MUX elements have the same BLKNM attribute.

Syntax

```
BLKNM=block_name
```

where *block_name* is a valid LCA block name for that type of symbol. For a list of prohibited block names, see the “Naming Conventions” section of each user interface manual.

For information on assigning hierarchical block names, see the “HBLKNM” section elsewhere in this chapter.

Example

Schematic

Attached to a valid instance.

UCF/NCF file

This statement assigns an instantiation of an element named block1 to a block named U1358.

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

BUFG

XC3000	XC4000E	XC4000X	XC5200	XC9000
				√

Applicable Elements

Any input buffer (IBUF) that drives a CLK, OE, or SR pin

Description

Maps the tagged signal to a global net.

Syntax

```
BUFG={CLK | OE | SR}
```

where CLK, OE, and SR indicate clock, output enable, or set/reset, respectively.

Example

Schematic

Attached to a valid instance.

UCF/NCF file

This statement maps the signal named “fastclk” to a global net.

```
INST clkgen/fastclk BUFG;
```

COLLAPSE

XC3000	XC4000E	XC4000X	XC5200	XC9000
				√

Applicable Elements

Any internal net

Description

Forces a combinatorial node to be collapsed into all of its fanouts.

Syntax

```
COLLAPSE
```

Example

Schematic

Attached to a net.

UCF/NCF file

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

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

CONFIG

XC3000	XC4000E	XC4000X	XC5200	XC9000
√				

Applicable Elements

IOB and CLB primitives

Description

Defines the configuration of the internal options of a CLB or IOB.

Note: Do not confuse this attribute with the CONFIG primitive, which is a table containing PROHIBIT and PART attributes. Refer to the “PROHIBIT” section for information on disallowing the use of a site and to the “PART” section for information on defining the part type for the design.

Syntax

```
CONFIG=tag value [ tag value]
```

where *tag* and *value* are derived from the following tables.

Table 0-3 XC3000 CLB Configuration Options

Tag	BASE F	BASE FG	BASE FGM*
X	F, QX	F, QX	M, QX
Y	F, QY	G, QY	M, QY
DX	DI, F	DI, F, G	DI, M
DY	DI, F	DI, F, G	DI, M
CLK	K, NOT	K, NOT	K, NOT
RSTDIR	RD	RD	RD
ENCLK	EC	EC	EC
F	A,B,C,D,E,QX, QY	A,B,C,D,E,QX, QY	A,B,C,D,QX, QY
G	None	A,B,C,D,E,QX, QY	A,B,C,D,QX, QY

*For BASE FGM, M=F if E=0, and M=G if E=1

Table 0-4 XC3000 IOB Configuration Options

Tag	BASE IO
IN	I, IQ, IKNOT, FF, LATCH, PULLUP
OUT	O, OQ, NOT, OKNOT, FAST
TRI	T, NOT

Example**Schematic**

Attached to a valid instance.

Following is an example of a valid XC3000 CLB CONFIG attribute value.

```
X:QX Y:QY DX:F DY:G CLK:K ENCLK:EC
```

UCF/NCF file

N/A

DECODE

XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√		

Applicable Elements

WAND1

Description

Defines how a wired-AND (WAND) instance is created, either using a BUFT or an edge decoder. If the DECODE attribute is placed on a single-input WAND1 gate, the gate is implemented as an input to a wide-edge decoder in an XC4000 design.

Syntax

```
DECODE
```

DECODE attributes can only be attached to a WAND1 symbol.

Example**Schematic**

Attached to a WAND1 symbol.

UCF/NCF file

This statement implements an instantiation of a wired AND using the edge decoder \$COMP_1

```
INST address_decode/$COMP_1 DECODE;
```

DIVIDE1_BY and DIVIDE2_BY

XC3000	XC4000E	XC4000X	XC5200	XC9000
			√	

Applicable Elements

OSC5, CK_DIV

Description

Defines the division factor for the on-chip clock dividers.

Syntax

```
DIVIDE1_BY={4 | 16 | 64 | 256}
```

```
DIVIDE2_BY={2 | 8 | 32 | 128 | 1024 | 4096 | 16384 | 65536}
```

Examples

Schematic

Attached to a valid instance.

NCF file

This statement defines the division factor of 8 for the clock divider \$1I45678.

```
INST clk_gen/$1I45678 divide2_by=8;
```

Note: DIVIDE1_BY and DIVIDE2_BY are not supported in the UCF file.

DOUBLE

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√		

Applicable Elements

Pullups

Description

Specifies double pull-up resistors on the horizontal longline. On XC3000 parts, there are internal nets that can be set as 3-state with two programmable pull-up resistors available per line. If the DOUBLE attribute is placed on a PULLUP symbol, both pull-ups are used, enabling a fast, high-power line. If the DOUBLE attribute is not placed on a pull-up, only one pull-up is used, resulting in a slower, lower-power line.

When the DOUBLE attribute is present, the speed of the distributed logic is increased, as is the power consumption of the part. When only half of the longline is used, there is only one pull-up at each end of the longline.

While the DOUBLE attribute can be used for the XC4000 family, it is not recommended. The mapper activates both pull-up resistors if the entire longline (not a half-longline) is used. When DOUBLE is used, PAR will not add an additional pullup to achieve your timing constraints while routing an XC4000 design (refer to the “PAR—Place and Route” chapter of the *Development System Reference Guide* for information on PAR and timing optimization).

Syntax

```
DOUBLE
```

Example

Schematic

Attached to a pullup only.

UCF/NCF file

N/A

DRIVE

XC3000	XC4000E	XC4000X	XC5200	XC9000
		√*		

* supported on the XC4000XV only

Applicable Elements

IOB output components (OBUF, OFD, etc.)

Description

Programs the output drive current high (24 mA) or low (12 mA).

Syntax

```
DRIVE={12 | 24}
```

where 12 is the default.

Note: DRIVE need not be specified for the XC4000EX and XC4000XL since they support only the default DRIVE=12. Only the XC4000XV supports both DRIVE=12 and DRIVE=24 and if DRIVE is not specified defaults to 12.

Example

Schematic

Attached to a valid IOB output component.

UCF/NCF file

This statement specifies a high drive (XC4000XV only).

```
INST /top/my_design/ofd DRIVE=24 ;
```

DROP_SPEC

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	√

Applicable Elements

All timing specifications. Should be used only in UCF or PCF files.

Description

Allows you to eliminate a timing specification entered in the input design when changes are made in the constraint side files.

Syntax

```
DROP_SPEC=TSidentifier
```

where *TSidentifier* is the identifier name used for the timing specification that is to be removed.

Example

Schematic

N/A

UCF/NCF file

This statement cancels the input design specification TS67.

```
DROP_SPEC TS67;
```

EQUATE_F and EQUATE_G

XC3000	XC4000E	XC4000X	XC5200	XC9000
√				

Applicable Elements

CLB primitive

Description

These attributes set the logic equations describing the F and G function generators of a CLB, respectively.

Syntax

```
EQUATE_F=equation
```

```
EQUATE_G=equation
```

where *equation* is a logical equation of the function generator inputs (A, B, C, D, E, QX, QY) using the boolean operators listed in the following table.

Table 0-5 Valid XC3000 Boolean Operators

Symbol	Boolean Equivalent
~	NOT
*	AND
@	XOR
+	OR
()	Group expression

Example**Schematic**

Attached to a valid instance.

Here are two examples illustrating the use of the EQUATE_F attribute.

```
EQUATE_F=F=((~A*B)+D)@Q
F=A@B+(C*~D)
```

UCF/NCF file

N/A

FAST

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	√

Applicable Elements

Output primitives, output pads, bidirectional pads

Note: You can also attach the FAST attribute to the net connected to the pad component in a UCF file. NGDBUILD transfers the attribute 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
```

Description

Increases the speed of an IOB output.

Note: The FAST attribute produces a faster output but may increase noise and power consumption.

Syntax

```
FAST
```

Example**Schematic**

Attached to a valid instance.

UCF/NCF file

This statement increases the output speed of the element *y2*.

```
INST $1I87/y2 FAST;
```

This statement increases the output speed of the pad to which *net1* is connected.

```
NET net1 FAST;
```

FILE

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	√

Applicable Elements

Macros that refer to underlying non-schematic designs

Description

FILE is attached to a macro that does not have an underlying schematic. It identifies the file to be looked at for a logic definition. The type of file to be searched for is defined by the search order of the program compiling the design.

Syntax

```
FILE=file_name
```

where *file_name* is the name of a file that represents the underlying logic for the element carrying the attribute. Example file types include EDIF, XTF, NMC.

Schematic

Attached to a valid instance.

UCF/NCF file

N/A

HBLKNM

XC3000	XC4000E	XC4000X	XC5200	XC9000
√ 1, 2, 3, 7, 8, 9, 10,12	√ 2, 3, 4, 5, 7, 8, 10, 11, 12, 13, 14, 15	√ 2, 3, 4, 5, 7, 8, 10, 11, 12, 13, 14, 15	√ 2, 3, 4, 6, 7, 10, 15	

Applicable Elements

1. IOB, CLB and CLBMAP (See Note below)
2. Registers
3. I/O elements and pads
4. FMAP

5. HMAP
6. F5MAP
7. BUFT
8. Pullup
9. ACLK, GCLK
10. BUFG
11. BUFGS, BUFGP
12. ROM
13. RAM
14. RAMS and RAMD
15. Carry logic primitives

Note: You can also attach the HBLKNM constraint to the net connected to the pad component in a 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 HBLKNM=property_value
```

Description

Assigns hierarchical LCA 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, NGDBUILD prefixes a hierarchical path name to the HBLKNM value.

Like BLKNM, the HBLKNM attribute forces function generators and flip-flops into the same CLB. Symbols with the same HBLKNM attribute map into the same CLB, if possible. However, using HBLKNM instead of BLKNM has the advantage of adding hierarchy path names during translation, and therefore the same HBLKNM attribute and value can be used on elements within different instances of the same macro.

For XC5200, a given HBLKNM string can only be used to group a logic cell (LC), which contains one register, one LUT (FMAP), and one F5_MUX element. An error will occur if two or more registers, two or more FMAPs, or two or more F5_MUX elements have the same HBLKNM attribute.

Syntax

```
HBLKNM=block_name
```

where *block_name* is a valid LCA block name for that type of symbol.

Example

Schematic

Attached to a valid instance.

UCF/NCF file

This statement specifies that the element `this_map` will be put into the block named `group1`.

```
INST $I13245/this_hmap HBLKNM=group1;
```

This statement attaches the HBLKNM constraint to the pad connected to net1

```
NET net1 HBLKNM=$COMP_0;
```

Note: Elements with the same HBLKNM are placed in the same logic block if possible. Otherwise an error occurs. Conversely, elements with different block names will not be put into the same block.

HU_SET

XC3000	XC4000E	XC4000X	XC5200	XC9000
	√ 1, 2, 3, 5, 7, 8, 9, 10,11, 12	√ 1, 2, 3, 5, 7, 8, 9, 10, 12	√ 1, 2, 4, 6, 7, 8, 12	

Applicable Elements

1. Registers
2. FMAP
3. HMAP
4. F5MAP
5. CY4
6. CY_MUX
7. Macro Instance
8. EQN
9. ROM
10. RAM
11. RAMS, RAMD
12. BUFT

Description

The HU_SET constraint is defined by the design hierarchy. However, it also allows you to specify a set name. It is possible to have only one H_SET constraint within a given hierarchical element (macro) but by specifying set names, you can specify several HU_SET sets.

NGDBUILD hierarchically qualifies the name of the HU_SET as it flattens the design and attaches the hierarchical names as prefixes. The difference between an HU_SET constraint and an H_SET constraint is that an HU_SET has an explicit user-defined and hierarchically qualified name for the set, but an H_SET constraint has only an implicit hierarchically qualified name generated by the design-flattening program. An HU_SET set “starts” with the symbols that are assigned the HU_SET constraint, but an H_SET set “starts” with the instantiating macro one level above the symbols with the RLOC constraints.

For background information about using the various set attributes, refer to the “RLOC Sets” section.

Syntax

```
HU_SET=set_name
```

where *set_name* is the identifier for the set; it must be unique among all the sets in the design.

Example

Schematic

Attached to a valid instance.

UCF/NCF file

This statement assigns an instance of the register FF_1 to a set named heavy_set.

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

INIT

XC3000	XC4000E	XC4000X	XC5200	XC9000
	√ 1, 2, 3	√ 1, 2, 3		√ 3

Applicable Elements

1. ROM
2. RAM
3. Registers

Description

Initializes ROMs, RAMs, and registers. The least significant bit of the value corresponds to the value loaded into the lowest address of the memory element. For register initialization, S indicates Set and R indicates Reset. The INIT attribute can be used to specify the initial value directly on the symbol with the following limitation. INIT may only be used on a RAM or ROM that is 1 bit wide and not more than 32 bits deep.

Syntax

```
INIT={value | S | R}
```

where *value* is a 4-digit or 8-digit hexadecimal number that defines the initialization string for the memory element, depending on whether the element is 16-bit or 32-bit. For example, INIT=ABAC1234.

S indicates Set and R indicates Reset for registers.

Example

Schematic

Attached to a net, pin, or instance.

NCF file

This statement defines the initialization string for an instantiation of the memory element ROM2 to be the 16-bit hexadecimal string 5555.

```
INST $1I3245/ROM2 INIT = 5555;
```

Note: INIT is not supported in the UCF file.

INREG

XC3000	XC4000E	XC4000X	XC5200	XC9000
			√	

Applicable Elements

Flip-flops, latches

Description

Because XC5200 IOBs do not have flip-flops or latches, you can apply this attribute to meet fast setup timing requirements. If a flip-flop or latch is driven by an IOB, you can specify INREG to enable PAR (Place and Route) to place the flip-flop/latch close to the IOB so that the two elements can be connected using fast routes. See also the “OUTREG” section.

Syntax

```
INREG
```

Example

Schematic

Attached to a latch or flip-flop instance.

UCF/NCF file

This statement directs PAR to place the flip-flop \$I1 near the IOB driving it.

```
INST $I1 INREG;
```

KEEP

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	√

Applicable Elements

Nets

Description

When a design is mapped, some nets may be absorbed into logic blocks. When a net is absorbed into a block, it can no longer be seen in the physical design database. This may happen, for example, if the components connected to each side of a net are mapped into the same logic block. The net may then be absorbed into the block containing the components. The KEEP constraint prevents this from happening.

Note: The KEEP property is translated into an internal constraint known as NOMERGE when targeting an FPGA. Messaging from the implementation tools will therefore refer to the system property NOMERGE—not KEEP.

Syntax

```
KEEP
```

Example

Schematic

Attached to a net.

UCF/NCF file

This statement ensures that the net \$SIG_0 will remain visible.

```
NET $I13245/$SIG_0 KEEP;
```

LOC

XC3000	XC4000E	XC4000X	XC5200	XC9000
√ 1, 5, 6, 12	√ 1, 2, 3, 5, 7, 9, 10, 11, 12, 13, 14, 15	√ 1, 2, 3, 5, 7, 9, 10, 11, 12, 13, 14, 15	√ 1, 2, 4, 5, 8, 12, 14	√ 1, 5, 16

Applicable Elements

1. Registers
2. FMAP
3. HMAP
4. F5MAP
5. IO elements
6. CLB and IOB primitives, CLBMAP
7. CY4
8. CY_MUX
9. ROM
10. RAM
11. RAMS, RAMD
12. BUFT

13. WAND
14. Clock buffers
15. Edge decoders
16. Any instance

Description for FPGAs

Defines where a symbol can be placed within an FPGA. It 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. The LOC constraint can be specified from the schematic and statements in a constraints file can also be used to direct placement.

You can specify multiple locations for the same symbol by using a comma (,) to separate each location within the field. It specifies that the symbols be placed in any of the locations specified. Also, you can specify an area in which to place a symbol or group of symbols.

The legal names are a function of the target LCA part type. However, to find the correct syntax for specifying a target location, you can load an empty part into EPIC (the design editor). Place the cursor on any block and click to display its location in the EPIC history area. Do not include the pin name such as .I, .O, or .T as part of the location.

You can use the LOC constraint for logic that uses multiple CLBs, IOBs, soft macros, or other symbols. To do this, use the LOC attribute on a soft macro symbol, which passes the location information down to the logic on the lower level. The location restrictions are applied to all blocks on the lower level for which LOCs are legal.

The XC5200 CLB is divided into four physical site locations that each contain one flip-flop, one function generator, and one carry logic element. Therefore, for the XC5200, each LOC attribute can be used for only one register, one FMAP, one F5_MUX element, or one CY_MUX element. An error will occur if two or more registers, two or more FMAPs, two or more F5_MUX elements, or two or more CY_MUX elements have the same LOC attribute.

Description for CPLDs

For CPLDs, use the LOC=*pin_name* attribute on a PAD symbol to assign the signal to a specific pin. The PAD symbols are IPAD, OPAD, IOPAD, and UPAD. You can use the LOC=FBnn attribute on any instance to assign the logic or register to a specific function block or macrocell, provided the instance is not collapsed.

Assignments and function block assignments are unconditional; that is, the software does not attempt to relocate a pin if it cannot achieve the specified assignment. You can apply the LOC constraint to as many symbols in your design as you like. However, each assignment further constrains the software as it automatically allocates logic and I/O resources to internal nodes and I/O pins with no LOC constraints.

LOC attributes can be used to override pin assignments saved in a guide file from an earlier iteration.

The LOC=FBnn_*mm* attribute 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 will be ignored.

Note: Pin assignment using the LOC attribute is not supported for bus pad symbols such as OPAD8.

Location Types

Use the following location types to define the physical location of an element.

P12	IOB location (chip carrier)
A12	IOB location (pin grid)
B, L, T, R	Indicates edge locations (bottom, left, top, right) — applies to edge decoders only.
LB, RB, LT, RT, BR, TR, BL, TL	Indicates half edges (left bottom, right bottom, and so forth) — applies to edge decoders only.
TL, TR, BL, BR	Indicates a corner for global buffer placement
FB7	CPLD function block
FB7_6	CPLD function block, macrocell
AA	CLB location for XC3000
CLB_R4C3	CLB location for XC4000 or XC5200
CLB_R6C8.F (or.G)	Function generator, RAM, ROM, or RAMS location for XC4000
CLB_R6C8.LC0 (or .LC1, .LC2, .LC3)	Function generator or register location for XC5200
CLB_R6C8.LC0 (or .LC2)	F5_MUX location for XC5200
CLB_R6C8.FFX (or.FFY)	Flip-flop location for XC4000
TBUF_R6C7.1 (or.2)	TBUF location for XC4000
TBUF_R6C7.0 (or .1, .2, or.3)	TBUF location for XC5200

The wildcard character * can be used to replace a single location with a range. For example:

C*	Any CLB in row C of an XC3000 device
*D	Any CLB in column D of an XC3000 device
CLB_R*C5	Any CLB in column 5 of an XC4000 or XC5200 device

The following are *not* supported.

- Dot extensions on ranges. For example, LOC=CLB_R0C0:CLB_R5C5.G. However, for the XC5200, range locations will be expanded to include extensions, CLB_R0C0.*:CLB_R5C5.*, for example, when the mapper passes a range constraint to the PCF file.
- B, L, R, T used to indicate IO edge locations (bottom, left, top, right)
- LB, RB, LT, RT, BR, TR, BL, TL used to indicate IO half edges (left bottom, right bottom, etc.)

Syntax for FPGAs

Single location

`LOC=location`

where *location* is a legal LCA location for the LCA part type. Examples of the syntax for single LOC constraints are given in the “Single LOC Constraint Examples” table.

Table 0-6 Single LOC Constraint Examples

Attribute	Description
<code>LOC=P12</code>	Place I/O at location P12.
<code>LOC=B</code>	Place decode logic on the bottom edge.
<code>LOC=TL</code>	Place decode logic on the top left edge, or global buffer in the top left corner.
<code>LOC=AA</code> (XC3000)	Place logic in CLB AA.
<code>LOC=TBUF.AC.2</code> (XC3000)	Place BUFT in TBUF above and one column to the right of CLB AC.
<code>LOC=CLB_R3C5</code> (XC4000)	Place logic in the CLB in row 3, column 5.
<code>LOC=CLB_R4C4.LC0</code> (XC5200)	Place logic in the lowest slice of the CLB in row 4, column 4.
<code>LOC=CLB_R4C5.ffx</code> (XC4000)	Place CLB flip-flop in the X flip-flop of the CLB in row 4, column 5.
<code>LOC=CLB_R4C5.F</code> (XC4000)	Place CLB function generator in the F generator of row 4, column 5.
<code>LOC=TBUF_R2C1.1</code> (XC4000)	Place BUFT in row 2, column 1, along the top.
<code>LOC=TBUF_R4C4.3</code> (XC5200)	Place BUFT in the top buffer in row 4, column 4.
<code>LOC=TBUF_R*C0</code> (XC4000, XC5200)	Place BUFT in any row in column 0.

Multiple locations

`LOC=location1,location2...,locationn`

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. Examples of multiple LOC constraints are provided in the “Multiple LOC Constraint Examples” table.

Table 0-7 Multiple LOC Constraint Examples

Attribute	Description
LOC=T,B (XC4000)	Place decoder (XC4000) on the top or bottom edge.
LOC=clb_r2c4,clb_r7c9 (XC4000)	Place the flip-flop in either CLB R2C4 or CLB R7C9.

Range of locations

LOC=*location:location* [SOFT]

You can define a range by specifying the two corners of a bounding box. Specify the upper left and lower right corners of an area in which logic is to be placed. 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 an area (range) are given in the “Area LOC Constraint Examples” table.

Table 0-8 Area LOC Constraint Examples

Attribute	Description
LOC=AA:FF (XC3000)	Place CLB logic anywhere in the top left corner of the LCA bounded by row F and column F.
LOC=CLB_R1C1:CLB_R5C5 (XC4000)	Place logic in the top left corner of the LCA in a 5 x 5 area bounded by row 5 and column 5.
LOC=CLB_R1C1:CLB_R5C5 PROHIBIT=CLB_R5C5 (must be specified in one continuous line) (XC4000)	Place CLB logic in the top left corner of the LCA in a 5 x 5 area, but not in the CLB in row 5, column 5.
LOC=TBUF_R1C1:TBUF_R2C8 (XC4000, XC5200)	Place BUFT anywhere in the area bounded by row 1, column 1 and row 2, column 8.
LOC=CLB_R1C1.LC3:CLB_R4C4.LC0 (XC5200)	Place logic in any slice in the top left corner of the LCA bounded by row 4, column 4.

Note: For area constraints, LOC ranges can be supplemented by the user with the keyword SOFT.

Syntax for CPLDs

LOC=*pin_name*

or

LOC=**FBnn**

or

LOC=**FBnn_mm**

where

pin_name is *Pnn* for PC packages; *nn* is a pin number. The pin name is *nn* (row number and column number) for PG packages. See the appropriate data book for the pin package names, for example, p12, or unbonded pad names, for example, u16. Examples are LOC=P24 and LOC=G2. This form is valid only on pad instances.

nn is a function block number and *mm* is a macrocell within a function block number. This form is valid on any instances.

Examples

Refer to the “Placement Constraints” section for multiple examples of legal placement constraints for each type of logic element (flip-flops, ROMs and RAMs, FMAPs and HMAPs, CLBMAPs, BUFTs, CLBs, IOBs, I/Os, edge decoders, global buffers) in FPGA designs.

Schematic

Attached to an instance.

UCF/NCF file

This specifies that an instance of the element BUF1 be placed above the CLB in row 6, column 9. For XC4000 devices, you can place the TBUF above or below the CLB. For XC5200 devices, you can place the TBUF in one of four locations (.0-3).

```
INST /DESIGN1/GROUPS/BUF1 LOC=TBUF_R6C9.1 ;
```

This specifies that each instance found under “FLIP_FLOPS” is to be placed in any CLB in column 8.

```
INST /FLIP_FLOPS/* LOC=CLB_R*C8;
```

This specifies that an instantiation of MUXBUF_D0_OUT be placed in IOB location P110.

```
INST MUXBUF_D0_OUT LOC=P110 ;
```

This specifies that the net DATA<1> be connected to the pad from IOB location P111.

```
NET DATA<1> LOC=P111 ;
```

MAP

XC3000	XC4000E	XC4000X	XC5200	XC9000
√ 4	√ 1, 2	√ 1, 2	√ 1, 3	

Applicable Elements

1. FMAP
2. HMAP
3. F5MAP
4. CLBMAP

Description

Placed on an FMAP, F5MAP, HMAP, or CLBMAP 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.

Syntax

```
MAP=[PUC | PUO | PLC | PLO]
```

where

PUC means that the CLB pins are unlocked, and the CLB is closed.

PUO means that the CLB pins are unlocked, and the CLB is open.

PLC means that the CLB pins are locked, and the CLB is closed.

PLO means that the CLB pins are locked, and the CLB is open.

“Unlocked” in these definitions means that the software can swap signals among the pins on the CLB; “locked” means that it cannot. “Open” means that the software can add or remove logic from the CLB; conversely, “closed” indicates that the software cannot add or remove logic from the function specified by the MAP symbol.

The default is PUO.

Note: Currently, only PUC and PUO are observed. PLC and PLO are translated into PUC and PUO, respectively.

Example

Schematic

Attached to a map symbol instance.

UCF/NCF file

This statement allows pin swapping and ensures that no logic other than that defined by the original map will be mapped into the function generators.

```
INST $1I3245/map_of_the_world map=puc;
```

MAXDELAY

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	

Applicable Elements

Nets

Description

The MAXDELAY attribute defines the maximum allowable delay on a net.

Syntax

```
MAXDELAY=allowable_delay[ units]
```

where *units* may be ps, ns, us, ms, GHz, MHz, or KHz. The default is ns.

Example

Schematic

Attached to a net.

UCF/NCF file

This statement assigns a maximum delay of 1 us to the net \$SIG_4.

```
NET $1I3245/$SIG_4 MAXDELAY=1us;
```

MAXSKEW

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	

Applicable Elements

Nets

Description

Defines the allowable skew on a net.

Syntax

```
MAXSKEW=allowable_delay[ units]
```

where *units* may be ps, ns, us, ms, GHz, MHz, or KHz. The default is ns.

Example

Schematic

Attached to a net.

UCF/NCF file

This statement specifies a maximum skew of 3 ns on net \$SIG_6.

```
NET $1I3245/$SIG_6 MAXSKEW=3;
```

MEDDELAY

XC3000	XC4000E	XC4000X	XC5200	XC9000
		√		

Applicable Elements

Input register

Note: You can also attach the MEDDELAY constraint to a net that is connected to a pad component in a 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 MEDDELAY
```

Description

Specifies a medium sized delay for the IOB register.

Syntax

```
MEDDELAY
```

Example

Schematic

Attached to a valid instance.

UCF/NCF file

This statement specifies that the register in the IOB \$COMP_6 will have a medium sized delay.

```
INST $1I87/$COMP_6 MEDDELAY;
```

This statement assigns a medium sized delay to the pad to which net1 is connected.

```
NET Net1 MEDDELAY ;
```

NODELAY

XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√		

Applicable Elements

Input register

Note: You can also attach the NODELAY constraint to a net connected to a pad component in a 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 NODELAY
```

Description

The default configuration of IOB flip-flops in XC4000 designs includes an input delay that results in no external hold time on the input data path. However, this delay can be removed by placing the NODELAY attribute on input flip-flops or latches, resulting in a smaller setup time but a positive hold time.

The NODELAY attribute can be attached to the I/O symbols and the special function access symbols TDI, TMS, and TCK.

Syntax

```
NODELAY
```

Example

Schematic

Attached to a valid instance.

UCF/NCF file

This statement specifies that IOB register inreg67 not have an input delay.

```
INST $1I87/inreg67 NODELAY;
```

This statement specifies that there be no input delay to the pad that is attached to net1.

```
NET net1 NODELAY ;
```

NOREDUCE

XC3000	XC4000E	XC4000X	XC5200	XC9000
				√

Applicable Elements

Any net

Description

NOREDUCE is typically used to prevent reduction of redundant logic terms placed in a design to eliminate glitches or logic hazards.

Syntax

```
NOREDUCE
```

Example

Schematic

Attached to a net.

UCF/NCF file

This statement specifies that there be no Boolean logic reduction or logic collapse from the net named \$SIG_12 forward.

```
NET $SIG_12 NOREDUCE;
```

OFFSET

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	√

Applicable Elements

Global

Description

Supports the translation of Synopsys' SET_ARRIVAL, SET_INPUT_DELAY and SET_OUTPUT_DELAY constraints. Used only for pad-related signals and cannot be used to extend the arrival time specification method to the internal signals in a design.

Syntax

```
OFFSET {IN | OUT} offset_time [units] {BEFORE | AFTER} [clk_net]
```

where

offset_time is the external offset.

units is an optional field to indicate the units for the offset time. The default is nano-seconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to indicate the intended units.

clk_net is the fully hierarchical netname of the clock net between the pad and its input buffer.

Example

Schematic

N/A

UCF/NCF file

This statement specifies that the data will be present on input43 at least 20 ns before the triggering edge of the clock signal CLOCK.

```
NET input43 OFFSET IN 20 BEFORE CLOCK;
```

For a detailed description of OFFSET, please see the “Specialized Support for Synopsys” section of the “Using Timing Constraints” chapter in the *Development System Reference Guide*.

OPT_EFFORT

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	

Applicable Elements

Any macro or hierarchy level

Description

Defines an effort level to be used by the optimizer.

Syntax

```
OPT_EFFORT={NORMAL | HIGH}
```

Example

Schematic

Attached to a macro.

UCF/NCF file

This statement attaches a high effort of optimization to all of the logic contained within the module defined by instance \$1I678/adder.

```
INST $1I678/adder OPT_EFFORT=HIGH;
```

OPTIMIZE

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	

Applicable Elements

Any macro or hierarchy level

Description

Defines whether optimization is performed on the flagged hierarchical tree. The OPTIMIZE attribute has no effect on any symbol that contains no combinational logic, such as an input/output buffer.

Syntax

```
OPTIMIZE={AREA | SPEED | BALANCE | OFF}
```

Example

Schematic

Attached to a macro.

UCF/NCF file

This statement specifies that no optimization be performed on an instantiation of the macro CTR_MACRO.

```
INST /$1I678/CTR_MACRO OPTIMIZE=OFF;
```

OUTREG

XC3000	XC4000E	XC4000X	XC5200	XC9000
			√	

Applicable Elements

Flip-flops, latches

Description

Because XC5200 IOBs do not have flip-flops or latches, you can apply this attribute to meet fast setup requirements. If a flip-flop or latch is driving an IOB, you can specify OUTREG to enable PAR (Place and Route) to place the flip-flop/latch close to the IOB so that the two elements can be connected using fast routes. See also the “INREG” section.

Syntax

```
OUTREG
```

Example

Schematic

Attached to a latch or flip-flop instance.

UCF/NCF file

This statement directs PAR to place the flip-flop \$I1 near the IOB that it is driving.

```
INST $I1 OUTREG;
```

PART

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	√

Applicable Elements

1. Global
2. Attached to CONFIG symbol in schematics

Description

Defines the part type used for the design.

Syntax

```
PART=part_type
```

where *part_type* can be device-speed-package or device-package-speed. For example, 4028EX-PG299-3 or 4028EX-3-PG299

The package string must always begin with an alpha character — *never* with a number.

The speed string must always begin with a numeric character — *never* with an alpha character.

The text XC is an optional prefix to the whole *part_type* string.

In a constraints file, the PART specification must be prepended by the keyword CONFIG.

Example

Schematic

Global or attached to the CONFIG symbol.

UCF/NCF file

This statement specifies a 4005E device, a PQ160C package, with a speed of 5.

```
CONFIG PART=4005E-PQ160C-5;
```

PERIOD

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	√

Applicable Elements

Nets that feed forward to drive flip-flop clock pins.

Description

Provides a convenient way of defining a clock period for registers attached to a particular clock net.

PERIOD controls pad-to-setup and clock-to-setup paths but not clock-to-pad paths. Refer to the “Using Timing Constraints” chapter in the *Development System Reference Guide* for more information on clock period specifications.

Syntax

Simple method

```
PERIOD=period[ units] {HIGH | LOW} [high_or_low_time hi_lo_units]
```

where

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 us to indicate the intended units.

HIGH or LOW indicates whether the first pulse is to be high or low.

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, us, ms, or % if the high or low time is an actual time measurement.

Alternate method

```
TSidentifier=PERIOD TNM_reference period [ units] {HIGH | LOW} [high_or_low_time hi_lo_units]
```


where

identifier is a reference identifier that has a unique name.

TNM_reference is the identifier name that is attached to a clock net (or a net in the clock path) using the TNM attribute.

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, ms, us, or % to indicate the intended units.

HIGH or LOW indicates whether the first pulse is to be high or low.

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, us, ms, or % if the high or low time is an actual time measurement.

Example

The following examples are for the “simple method.”

Schematic

Attached to a net.

```
PERIOD=40 HIGH 25;
```

UCF/NCF file

This statement assigns a clock period of 40 ns to the net named \$SIG_24, with the first pulse being high and having a duration of 25 nanoseconds.

```
NET $SIG_24 PERIOD=40 HIGH 25;
```

PROHIBIT

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	

Applicable Elements

Attached to CONFIG symbol

Description

Disallows the use of a site within PAR and EPIC.

Location Types

Use the following location types to define the physical location of an element.

P12	IOB location (chip carrier)
A12	IOB location (pin grid)

B, L, R, T	Indicates edge locations (bottom, left, top, right) — applies to edge decoders only
LB, RB, LT, RT, BR, TR, BL, TL	Indicates half edges (left bottom, right bottom, and so forth) — applies to edge decoders only
TL, TR, BL, BR	Indicates a corner for global buffer placement
FB7	CPLD function block
FB7_6	CPLD function block, macrocell
AA	CLB location for XC3000
CLB_R4C3	CLB location for XC4000 or XC5200
CLB_R6C8.LC0 (or 1, 2, 3)	Function generator or register location for XC5200
CLB_R6C8.LC0 (or 2)	F5_MUX location for XC5200
TBUF_R6C7.1 (or.2)	TBUF location for XC4000
TBUF_R6C7.0 (or.1,.2, or.3)	TBUF location for XC5200

The wildcard character * can be used to replace a single location with a range. For example:

C*	Any CLB in row C of an XC3000 device
*D	Any CLB in column D of an XC3000 device
CLB_R*C5	Any CLB in column 5 of an XC4000 or XC5200 device

The following are *not* supported.

- Dot extensions on ranges. For example, LOC=CLB_R0C0:CLB_R5C5.G. However, for the XC5200, range locations will be expanded to include extensions, CLB_R0C0.*:CLB_R5C5.*, for example, when the mapper passes a range constraint to the PCF file.
- B, L, R, T used to indicate IO edge locations (bottom, left, top, right)
- LB, RB, LT, RT, BR, TR, BL, TL used to indicate IO half edges (left bottom, right bottom, etc.)
- .F or .G extension for function generator, RAM, ROM, or RAMS location for XC4000
- .FFX or .FFY extension for flip-flop location for XC4000

Syntax

Single location

PROHIBIT=*location*

Multiple single locations

PROHIBIT=*location1, location2... , locationn ;*

Range of locations

PROHIBIT=*location:location*

In a constraints file, the PROHIBIT specification must be prepended by the keyword CONFIG.

Example

Schematic

Unattached attribute or attached to a CONFIG symbol.

UCF/NCF file

This statement prohibits use of the site P45.

```
CONFIG PROHIBIT=P45;
```

This statement prohibits use of the CLB located in Row 6, Column 8.

```
CONFIG PROHIBIT=CLB_R6C8 ;
```

This statement prohibits use of the site TBUF_R5C2.2.

```
CONFIG PROHIBIT=TBUF_R5C2.2 ;
```

PWR_MODE

XC3000	XC4000E	XC4000X	XC5200	XC9000
				√

Applicable Elements

1. Nets
2. Any instance

Description

Defines the mode, low power or high performance (standard power) of the macrocell that implements the tagged element.

Note: If the tagged function is collapsed forward into its fanouts, the attribute is not applied.

Syntax

```
PWR_MODE={LOW | STD}
```

Example

Schematic

Attached to a net or an instance.

UCF/NCF file

This statement specifies that the macrocell that implements the net \$SIG_0 will be in low power mode.

```
NET $1187/$SIG_0 PWR_MODE=LOW;
```

RLOC

XC3000	XC4000E	XC4000X	XC5200	XC9000
	√ 1, 2, 3, 5, 7, 8, 9, 10, 11	√ 1, 2, 3, 5, 7, 8, 9, 10, 11	√ 1, 2, 4, 6, 10	

Applicable Elements

1. Registers
2. FMAP
3. HMAP
4. F5MAP
5. CY4
6. CY_MUX
7. ROM
8. RAM
9. RAMS, RAMD
10. BUFT
11. WAND primitives that do not have a DECODE attribute attached

Description

Relative location (RLOC) constraints group logic elements into discrete sets and 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. See the “Physical Constraints” section for detailed information about this type of constraint.

For XC5200, the RLOC attribute must include the extension that defines in which of the four slices of a CLB the element will be placed (.LC0, .LC1, .LC2, .LC3). This defines the relationship of the elements in the set and also specifies in which of the four slices the element will eventually be placed.

Syntax

For the XC4000:

```
RLOC=RmCn[ .extension]
```

For the XC5200:

```
RLOC=RmCn .extension
```

where

m and *n* are integers (positive, negative, or zero) representing relative row numbers and column numbers, respectively.

extension uses the LOC extension syntax as appropriate; it can take all the values that are available with the current absolute LOC syntax: FFX, FFY, F, G, H, 1, and 2. The 1 and 2 values are available for BUFT primitives, and the rest are available for primitives associated with CLBs. See the “LOC” section for more details.

For the XC5200, *extension* is required to define in which of the four slices of a CLB the element will be placed (.LC0, .LC1, .LC2, .LC3).

The RLOC value cannot specify a range or a list of several locations; it must specify a single location. See the “Guidelines for Specifying Relative Locations” section for more information.

Example

Schematic

Attached to an instance.

UCF/NCF file

This statement specifies that an instantiation of FF1 be placed in the CLB at row 4, column 4.

```
INST /4K/design/FF1 RLOC=R4C4;
```

This statement specifies that an instantiation of elemA be placed in the X flip-flop in the CLB at row 0, column 1.

```
INST /$1I87/elemA RLOC=r0c1.FFX;
```

RLOC_ORIGIN

XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	

Applicable Elements

Instances or macros that are members of sets

Description

An RLOC_ORIGIN constraint fixes the members of a set at exact die locations. This constraint must specify a single location, not a range or a list of several locations. For more information about this constraint, refer to the “Fixing Members of a Set at Exact Die Locations” section.

The RLOC_ORIGIN constraint is required for a set that includes BUFT symbols. The RLOC_ORIGIN constraint cannot be attached to a BUFT instance.

Syntax

```
RLOC_ORIGIN=RmCn
```

where *m* and *n* are positive integers (including zero) representing relative row and column numbers, respectively.

Example

Schematic

Attached to an instance that is a member of a set.

UCF/NCF file

This statement specifies that an instantiation of FF1, which is a member of a set, be placed in the CLB at R4C4 relative to FF1. For example, if RLOC=R0C2 for FF1, then the instantiation of FF1 is placed in the CLB that occupies row 4 ($R0 + R4$), column 6 ($C2 + C4$).

```
INST /archive/designs/FF1 RLOC_ORIGIN=R4C4;
```

RLOC_RANGE

XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	

Applicable Elements

Instances or macros that are members of sets

Description

The RLOC_RANGE constraint is similar to the 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 RLOCs, not just to the origin of the set.

Syntax

```
RLOC_RANGE=Rm1Cn1:Rm2Cn2
```

where the relative row numbers ($m1$ and $m2$) and column numbers ($n1$ and $n2$) can be positive integers (including zero) or the wildcard (*) character. This syntax allows three kinds of range specifications, which are defined in the “Fixing Members of a Set at Exact Die Locations” section.

Example

Schematic

Attached to an instance that is a member of a set.

UCF/NCF file

This statement specifies that an instantiation of the macro MACRO4 be placed within a region that is enclosed by the rows R4-R10 and the columns C4-C10.

```
INST /archive/designs/MACRO4 RLOC_RANGE=R4C4:R10C10;
```

S(ave) - Net Flag Attribute

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	

Applicable Elements

Nets

Description

Attaching the net flag attribute to nets affects the mapping, placement, and routing of the design.

Syntax

S

The S (save) net flag attribute prevents the removal of unconnected signals. If you do not have the S attribute on a net, any signal not connected to logic and/or an I/O primitive is removed.

Example

Schematic

Attached to a net.

UCF/NCF file

This statement specifies that the net named \$SIG_9 will not be removed.

```
NET $SIG_9 S;
```

SLOW

XC3000	XC4000E	XC4000X	XC5200	XC9000
				√

Applicable Elements

Output primitives, output pads, bidirectional pads

Note: You can also attach the SLOW constraint to the net connected to the pad component in a 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 SLOW
```

Description

Stipulates that the slew rate limited control should be enabled.

Syntax

```
SLOW
```

Example

Schematic

Attached to a valid instance.

UCF/NCF file

This statement establishes a slow slew rate for an instantiation of the element `y2`.

```
INST $1I87/y2 SLOW;
```

This statement establishes a slow slew rate for the pad to which `net1` is connected.

```
NET net1 SLOW;
```

TIG

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	

Applicable Elements

Nets, pins

Description

Paths that fan forward from the point of application are treated as if they do not exist (for the purposes of the timing model) during implementation.

A TIG may be applied relative to a specific timing specification.

Syntax

```
TIG
```

or

```
TIG=TSidentifier1... , TSidentifiern
```

where *identifier* refers to a timing specification that should be ignored.

Example

Schematic

Attached to a net or pin.

UCF/NCF file

This statement specifies that the timing specifications `TS_fast` and `TS_even_faster` will be ignored on all paths fanning forward from the net `$$Sig_5`.

```
NET $1I567/$Sig_5 TIG=TS_fast, TS_even_faster;
```

For more on TIG, see the “Ignoring Selected Paths” section of the “Using Timing Constraints” chapter in the *Development System Reference Guide*.

Time Group Attributes

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	√

Applicable Elements

1. Global in constraints file (preended by the keyword TIMEGRP)
2. Time group primitive

Description

Time group properties (attributes) are a set of grouping mechanisms that use existing TNMs (Timing Names) to create new groups or to define new groups based on the output net that the group sources. The timing group primitive (TIMEGRP) exists for the purpose of hosting these properties. In a constraints file, the specification of these properties must be prepended with the keyword TIMEGRP.

Note: When entering time group properties into a TIMEGRP symbol, some property names may conflict with the predefined property names of the TIMEGRP primitive.

The standard procedure for adding a property to a symbol is to use the following format:

```
PROPERTY=property_name VALUE=value
```

However, some property names are reserved, and should not be used because they cause a conflict. Hence, for *property_name* you must not use any of the system reserved names LIBVER, INST, COMP, MODEL, or any other names reserved by your schematic capture program. Please consult your schematic capture documentation to become familiar with reserved property names.

Note: For more on the TIMEGRP symbol, see the “TIMEGRP” section in the Design Elements chapter.

Syntax

```
new_group_name=[RISING | FALLING] group_name1 [EXCEPT group_name2...  
group_namen]
```

or

```
new_group_name=[TRANSHI | TRANSLO] group_name1 [EXCEPT group_name2...  
group_namen]
```

where

group_names can be

- the name assigned to a previously defined group.
- all of the members of a predefined group using the keywords FFS, RAMS, PADS or LATCHES. FFS refers to all flip-flops. RAMS refers to all RAMs. PADS refers to all I/O pads. LATCHES refers to all latches.
- a subset of elements in a group predefined by name matching using the following syntax.

```
predefined_group name qualifier1 . . . name_qualifiern
```

RISING or FALLING applies to the rising or falling edge sensitive elements of a group of flip-flops to be referred to as a subset.

TRANSHI or TRANSLO is the form of the constraint applied to latches.

EXCEPT excludes the object group.

Example 1

Schematic

The following attribute would be attached to a TIMEGRP primitive to combine the elements in two groups to form a new group.

```
big_group=little_group other_group
```

UCF/NCF file

The same constraint could appear in a User Constraints File (UCF) as follows.

```
TIMEGRP big_group=little_group other_group;
```

Example 2

Schematic

The following constraints would be attached to a TIMEGRP primitive to define new groups by exclusion.

```
input_pads=pads except output_pads
```

UCF/NCF file

The same constraint could appear in a UCF as follows.

```
TIMEGRP input_pads=pads EXCEPT output_pads;
```

For more on Time Group Attributes, see the “Timing Specifications” section of the “Using Timing Constraints” chapter in the *Development System Reference Guide*. See also the “Syntax Summary” section in the same chapter.

TNM

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	√

Applicable Elements

Nets, instances, macros

Note: You can attach the TNM constraint to the net connected to the pad component in a 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 TNM=property_value
```

Description

Tags specific flip-flops, RAMs, pads, and latches as members of a group to simplify the application of timing specifications to the group.

TNMs (Timing Names) applied to pad nets do not propagate forward through the IBUF/ OBUF. The TNM is applied to the external pad. This case includes the net attached to the D input of an IFD.

TNMs applied to the input pin of an IBUF/ OBUF will propagate the TNM to the next appropriate element.

TNMs applied to the output pin of an IBUF/OBUF will propagate the TNM to the next appropriate element.

TNMs applied to an IBUF or OBUF element stay attached to that element.

TNMs applied to a clock-pad-net will not propagate forward through the clock buffer.

When TNM is applied to a macro, all the elements in the macro will have that timing name.

See the “Timing Specifications” section of the “Using Timing Constraints” chapter in the *Development System Reference Guide* for detailed information about this attribute.

Syntax

TNM=identifier

where *identifier* can be any combination of letters, numbers, or underscores.

Do not use reserved words, such as FFS, LATCHES, RAMS, or PADS for TNM identifiers.

Example

Schematic

Attached to a net or a macro.

UCF/NCF file

This statement identifies the element `register_ce` as a member of the timing group `the_register`.

```
NET $1I87/register_ce TNM=the_register;
```

TPSYNC

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	

Applicable Elements

Nets

Description

Flags a particular point or a set of points with an identifier for reference in subsequent timing specifications. You can use the same identifier on several points, in which case timing analysis treats the points as a group. See the “Time Group Attributes” section.

Defining synchronous points

When the timing of a design must be designed from or to a point that is not a flip-flop, latch, RAM, or I/O pad, the following rules apply.

If a TPSYNC timing point is attached to:

- A net — the source of the net is identified as a potential source or destination for timing specifications.
- A macro pin — all of the sources inside the macro that drive the pin to which the attribute 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 primitive's output is flagged as a potential source or destination for timing specifications.
- The input pin of a primitive — the primitive's input 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.

Syntax

`TPSYNC=identifier`

where *identifier* is a name that is used in timing specifications in the same way that groups are used.

All flagged points are used as a source or destination or both for the specification where the TPSYNC identifier is used.

Note: The name for the identifier must be different from any identifier used for a TNM attribute.

Example

Schematic

Attached to a net.

UCF/NCF file

This statement identifies latch as a potential source or destination for timing specifications for the net `logic_latch`.

```
NET $1I87/logic_latch TPSYNC=latch;
```

TPTHRU

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	

Applicable Elements

Nets

Description

Flags a particular point or a set of points with an identifier for reference in subsequent timing specifications. You can use the same identifier on several points, in which case timing analysis treats the points as a group. See the “Time Group Attributes” section.

Defining through points

The TPTHRU attribute is used when it is necessary to define intermediate points on a path to which a specification applies. See the “TSidentifier” section.

Syntax

`TPTHRU=identifier`

where *identifier* is a name used in timing specifications for further qualifying timing paths within a design.

Note: The name for the identifier must be different from any identifier used for a TNM attribute.

Example

Schematic

Attached to a net.

UCF/NCF file

This statement identifies the net `on_the_way` as an intermediate point on a path to which the timing specification named “here” applies.

```
NET $1I87/on_the_way TPTHRU=here;
```

TSidentifier

XC3000	XC4000E	XC4000X	XC5200	XC9000
√	√	√	√	√

Applicable Elements

1. Global in constraints file
2. TIMESPEC primitive

Description

TSidentifier properties beginning with the letters “TS” are placed on the TIMESPEC symbol. In a constraints file, the specification of these properties can be prepended with the optional keyword TIMESPEC. The value of the *TSidentifier* attribute corresponds to a specific timing specification that can then be applied to paths in the design.

Syntax

Note: All the following syntax definitions use a space as a separator. The use of a colon (:) as a separator is optional.

Defining a maximum allowable delay

```
TSidentifier=[MAXDELAY] FROM source_group TO dest_group allowable_delay [units]
```

or

```
TSidentifier=FROM source_group TO dest_group allowable_delay [units]
```

Defining intermediate points

Note: This form is not supported for CPLDs.

```
TSidentifier=FROM source_group THRU thru_point [THRU thru_point1 . . . thru_pointn] TO
dest_group allowable_delay [units]
```

where

identifier is an ASCII string made up of the characters A-Z, a-z, 0-9, and _.

source_group and *dest_group* are user-defined or predefined groups.

thru_point is an intermediate point used to qualify the path, defined using a TPTHRU attribute.

allowable_delay is the timing requirement.

units is an optional field to indicate the units for the allowable delay. The default units are nanoseconds (ns), but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to indicate the intended units.

Defining a linked specification

This allows you to link the timing number used in one specification to another specification.

```
TSidentifier=FROM source_group TO dest_group another_TSid[ / | * ] number
```

where

identifier is an ASCII string made up of the characters A-Z, a-z, 0-9, and _.

source_group and *dest_group* are user-defined or predefined groups.

another_Tsid is the name of another timespec.

number is a floating point number.

Defining a clock period

This allows more complex derivative relationships to be defined as well as a simple clock period.

```
TSidentifier=PERIOD TNM_reference period [units] [ {HIGH | LOW} high_or_low_time
[ hi_lo_units ] ]
```

where

identifier is a reference identifier with a unique name.

TNM_reference is the identifier name attached to a clock net (or a net in the clock path) using a TNM attribute.

period is the required clock period.

units is an optional field to indicate the units for the allowable delay. The default units are nanoseconds (ns), but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to indicate the intended units.

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, us, ms, or % if the high or low time is an actual time measurement.

Specifying derived clocks

```
TSidentifier=PERIOD TNM_reference another_PERIOD_identifier [ / | * ] number
[ { HIGH | LOW } high_or_low_time [ hi_lo_units ] ]
```

where

TNM_reference is the identifier name attached to a clock net (or a net in the clock path) using a TNM attribute.

another_PERIOD_identifier is the name of the identifier used on another period specification.

number is a floating point number.

high_or_low_time is the optional high or low time (must be less than the period), depending on the preceding keyword. The default duty cycle is 50 percent.

Ignoring paths

Note: This form is not supported for CPLDs.

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.

```
TSidentifier=FROM source_group TO dest_group TIG
```

or

```
TSidentifier=FROM source_group THRU thru_point [ THRU thru_point1... thru_pointn ]
TO dest_group TIG
```

where

identifier is an ASCII string made up of the characters A-Z, a-z 0-9, and _.

source_group and *dest_group* are user-defined or predefined groups.

thru_point is an intermediate point used to qualify the path, defined using a TPTHRU attribute.

Example

Schematic

Attached to a TIMESPEC primitive.

UCF/NCF file

This statement says that the timing specification TS_35 calls for a maximum allowable delay of 50 ns between the groups “here” and “there”.

```
TIMESPEC TS_35=FROM here TO there 50;
```

This statement says 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.

```
TIMESPEC TS_70=PERIOD "clock_a" 25 high 15;
```

For more information, see the “Timing Constraints” section.

Note: In either example above, a colon can be used instead of a space as the separator. (Additional spaces entered before or after the colon are ignored.) The statements then become:

```
TIMESPEC TS_35=FROM:here:TO:there:50;
```

```
TIMESPEC TS_70=PERIOD:"clock_a":25:high:15;
```

U_SET

XC3000	XC4000E	XC4000X	XC5200	XC9000
	√ 1, 2, 3, 5, 7, 8, 9, 10, 11, 12	√ 1, 2, 3, 5, 7, 8, 9, 10, 11, 12	√ 1, 2, 4, 6, 7, 8, 12	

Applicable Elements

1. Registers
2. FMAP
3. HMAP
4. F5MAP
5. CY4
6. CY_MUX
7. Macro instance
8. EQN
9. ROM
10. RAM
11. RAMS, RAMD
12. BUFT

Description

The U_SET constraint groups design elements with attached 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; that is, you can arbitrarily select objects without regard to the design hierarchy and tag them as members of a U_SET. For detailed information about this attribute, refer to the “RLOC Sets” section.

Syntax

```
U_SET=name
```

where *name* is the identifier of the set. This name is absolute. It is not prefixed by a hierarchical qualifier.

Example**Schematic**

Attached to a valid instance.

UCF/NCF file

This statement specifies that the design element ELEM_1 be in a set called JET_SET.

```
INST $1I3245/ELEM_1 U_SET=JET_SET;
```

USE_RLOC

XC3000	XC4000E	XC4000X	XC5200	XC9000
	√	√	√	

Applicable Elements

Instances or macros that are members of sets

Description

Turns on or off the RLOC constraint for a specific element or section of a set. For detailed information about this constraint, refer to the “Toggling the Status of RLOC Constraints” section.

Syntax

```
USE_RLOC={TRUE | FALSE}
```

where TRUE turns on the RLOC attribute for a specific element, and FALSE turns it off. Default is TRUE.

Example**Schematic**

Attached to a member of a set.

UCF/NCF file

```
INST $1I87/big_macro USE_RLOC=FALSE;
```

WIREAND

XC3000	XC4000E	XC4000X	XC5200	XC9000
				√

Applicable Elements

Any net

Description

Forces a tagged node to be implemented as a wired AND function in the interconnect (UIM and Fastconnect).

Syntax

```
WIREAND
```

Example

Schematic

Attached to a net.

UCF/NCF file

This statement specifies that the net named SIG_11 be implemented as a wired AND when optimized.

```
NET $I16789/SIG_11 WIREAND;
```

Placement Constraints

This section describes the legal placement constraints for each type of logic element, such as flip-flops, ROMs and RAMs, FMAPs, F5MAPs, and HMAPs, CLBMAPs, BUFTs, CLBs, IOBs, I/Os, edge decoders, and global buffers in FPGA designs. Individual logic gates, such as AND or OR gates, are mapped into CLB function generators before the constraints are read and therefore cannot be constrained. However, if gates are represented by an FMAP, F5MAP, HMAP, or CLBMAP symbol, you can put a placement constraint on that symbol.

You can use the following constraints (described earlier in the “Attributes/Logical Constraints” section) to control mapping and placement of symbols in a netlist.

- BLKNM
- HBLKNM
- LOC
- PROHIBIT
- RLOC
- RLOC_ORIGIN
- RLOC_RANGE

Most constraints can be specified either in the schematic or in the UCF file.

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.

Note: The UCF and NCF files are case sensitive. Identifier names (names of objects in the design, 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, PROHIBIT, RLOC, BLKNM) may be entered in either all upper-case or all lower-case letters; mixed case is not allowed.

For each type of constraint described in the following sections, the method of determining the symbol name is explained and examples are given.

BUFT Constraint Examples

You can constrain internal 3-state buffers (BUFTs) to an individual BUFT location, a list of BUFT locations, or a rectangular block of BUFT locations. BUFT constraints all refer to locations with a prefix of TBUF, which is the name of the physical element on the device.

BUFT constraints can be assigned from the schematic or through the UCF file. From the schematic, LOC constraints are attached to the target BUFT. The constraints are then passed into the EDIF netlist file and after mapping are read by PAR. Alternatively, in a constraints file a BUFT is identified by a unique instance name.

In the XC3000, BUFT locations are not straightforward. View the device in EPIC to determine the exact BUFT names.

In the XC4000, BUFT locations are identified by the adjacent CLB. Thus, TBUF_R1C1.1 is just above CLB_R1C1, and TBUF_R1C1.2 is just below it. For an XC4000, use the following syntax to denote fixed locations.

```
TBUF_RowCol [ .1 | .2 ]
```

where *row* is the row location and *col* is the column location; they can be any number between 0 and 99, inclusive. They must be less than or equal to the number of CLB rows or columns in the target device. The suffixes have the following meanings.

- 1 indicates that the instance should be placed above the CLB.
- 2 indicates that the instance should be placed below the CLB.

In the XC5200, BUFT locations are identified by the adjacent slice. From bottom to top, they are number 0, 1, 2, and 3. Thus, TBUF_R1C1.0 is located toward the bottom of the row. TBUF_R1C1.3 is located toward the top of the row. For an XC5200, Use the following syntax to denote fixed locations.

```
TBUF_RrowCcol [.0 | .1 | .2 | .3]
```

where *row* is the row location and *col* is the column location; they can be any number between 0 and 99, inclusive. They must be less than or equal to the number of CLB rows or columns in the target device. The suffixes have the following meanings.

- 0 indicates that the instance should be placed in the bottom buffer.
- 1 indicates that the instance should be placed in the buffer that is second from bottom.
- 2 indicates that the instance should be placed in the buffer that is second from top.
- 3 indicates that the instance should be placed in the top buffer.

For either the XC4000 or XC52000, use the following syntax to denote a range of locations from the lowest to the highest.

```
TBUF_RrowCcol TBUF_RrowCcol
```

The following examples illustrate the format of BUFT LOC constraints. Specify LOC= and the BUFT location.

The following statements place the BUFT in the designated location.

```
LOC=TBUF.AA.1 (XC3000)
LOC=TBUF_R1C1.1 (XC4000)
LOC=TBUF_R1C1.3 (XC5200)
```

The next statements place BUFTs at any location in the first column of BUFTs. The asterisk (*) is a wildcard character.

```
LOC=TBUF.*A (XC3000)
LOC=TBUF_R*C0 (XC4000, XC5200)
```

The following statements place BUFTs within the rectangular block defined by the first specified BUFT in the upper left corner and the second specified BUFT in the lower right corner.

```
LOC=TBUF.AA:TBUF.BH (XC3000)
LOC=TBUF_R1C1:TBUF_R2C8 (XC4000, XC5200)
```

In the following examples, the instance names of two BUFTs are /top-72/rd0 and /top-79/ed7.

Example 1

This example specifies a BUFT adjacent to a specific CLB.

```
Schematic    LOC=TBUF_r1c5
UCF          INST /top-72/rd0 LOC=TBUF_r1c5 ;
```

Place the BUFT adjacent to CLB R1C5. In an XC4000, PAR uses either the longline above the row of CLBs or the longline below. In an XC5200, PAR places the BUFT in one of the four slices of the CLB at row 1, column 5.

Example 2

The following example places a BUFT in a specific location.

```
Schematic    LOC=TBUF_r1c5.1
UCF          INST /top-72/rd0 LOC=TBUF_r1c5.1 ;
```

Place the BUFT adjacent to CLB R1C5. In an XC4000 device, .1 tag specifies the longline above the row of CLBs; the .2 tag specifies the longline below it. In an XC5200 device, the .0 tag specifies the longline associated with the bottom-most slice in the CLB at the location; the .1, .2, .3 tags specify slices above the .0 slice for the specified row and column.

BUFTs that drive the same signal must carry consistent constraints. If you specify .1 or .2 for one of the BUFTs that drives a given signal, you must also specify .1 or .2 on the other BUFTs on that signal; otherwise, do not specify any constraints at all.

Example 3

The next example specifies a column of BUFTs.

```
Schematic    LOC=TBUF_r*c3
UCF          INST /top-72/rd0 /top-79/ed7 LOC=TBUF_r*c3 ;
```

Place BUFTs in column 3 on any row. This constraint might be used to align BUFTs with a common enable signal. You can use the wildcard (*) character in place of either the row or column number to specify an entire row or column of BUFTs.

Example 4

This example specifies a row of BUFTs .

```
Schematic    LOC=TBUF_r7c*
UCF          INST /top-79/ed7 LOC=TBUF_r7c* ;
```

Place the BUFT on one of the longlines in row 7 for any column. You can use the wildcard (*) character in place of either the row or column number to specify an entire row or column of BUFTs.

CLB Constraint Examples

You can assign soft macros and flip-flops to a single CLB location, a list of CLB locations, or a rectangular block of CLB locations. You can also specify the exact function generator or flip-flop within a CLB. CLB locations are identified as CLB_RowCol for an XC4000 and XC5200, or aa for an XC3000, where aa is a two-letter designator. The lower left CLB is CLB_R1C1 (for the XC4000 or XC5200) or AA (for the XC3000).

CLB locations can be a fixed location or a range of locations. Use the following syntax to denote fixed locations.

For XC4000:

```
CLB_R rowCcol { .F | .G | .FFX | .FFY }
```

For XC5200:

```
CLB_R rowCcol { .LC0 | .LC1 | .LC2 | .LC3 }
```

where

row is the row location and *col* is the column location; they can be any number between 0 and 99, inclusive, or *. They must be less than or equal to the number of CLB rows or columns in the target device. The suffixes have the following meanings.

.F means the CLB is mapped into the F function generator.

.G means the CLB is mapped into the G function generator.

.FFX indicates the X flip-flop in the CLB.

.FFY indicates the Y flip-flop in the CLB.

.LC0 means the bottom-most slice in the XC5200 CLB.

.LC1 means the slice above the .LC0 slice in the XC5200 CLB.

.LC2 means the slice above the .LC1 slice in the XC5200 CLB.

.LC3 means top-most slice in the XC5200 CLB.

Use the following syntax to denote a range of locations from the highest to the lowest.

```
CLB_Rrow1Ccol:CLB_Rrow2Ccol2
```

The following examples illustrate the format of CLB constraints. Enter LOC= and the pin or CLB 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.

The following statements place logic in the designated CLB.

```
LOC=AA (XC3000)
LOC=CLB_R1C1 (XC4000)
LOC=CLB_R1C1.LC0 (XC5200)
```

The following statements place logic within the first column of CLBs. The asterisk (*) is a wildcard character.

```
LOC=*A (XC3000)
LOC=CLB_R*C1 (XC4000)
```

The next two statements place logic in any of the three designated CLBs. There is no significance to the order of the LOC statements.

```
LOC=AA,AB,AC (XC3000)
LOC=CLB_R1C1,CLB_R1C2,CLB_R1C3 (XC4000,XC5200)
```

The following statements place logic within the rectangular block defined by the first specified CLB in the upper left corner and the second specified CLB towards the lower right corner.

```
LOC=AA:HE (XC3000)
LOC=CLB_R1C1:CLB_R8C5 (XC4000,XC5200)
```

The next statement places logic in the X flip-flop of CLB_R2C2. For the Y flip-flop, use the FFY tag.

```
LOC=CLB_R2C2.FFX (XC4000)
```

You can prohibit PAR from using a specific CLB, a range of CLBs, or a row or column of CLBs. Such prohibit constraints can be assigned only through the User Constraints File (UCF). CLBs are prohibited by specifying a PROHIBIT constraint at the design level, as shown in the following examples.

Example 1

```
Schematic      None
UCF            CONFIG PROHIBIT=clb_r1c5 ;
```

Do not place any logic in the CLB in row 1, column 5. CLB R1C1 is in the upper left corner of the device.

Example 2

```
Schematic      None
UCF            CONFIG PROHIBIT=clb_r1c1:clb_r5c7 ;
```

Do not place any logic in the rectangular area bounded by the CLB R1C1 in the upper left corner and CLB R5C7 in the lower right.

Example 3

```
Schematic      None
UCF            CONFIG PROHIBIT=clb_r*c3 ;
```

Do not place any logic in any row of column 3. You can use the wildcard (*) character in place of either the row or column number to specify an entire row or column of CLBs.

Example 4

```
Schematic      None
UCF            CONFIG PROHIBIT=clb_r2c4, clb_r7c9 ;
```

Do not place any logic in either CLB R2C4 or CLB R7C9.

Edge Decoder Constraint Examples (XC4000 Only)

In an XC4000 design, you can assign the decode logic to a specified die edge or half-edge. All elements of a single decode function must lie along the same edge; they cannot be split across two edges of the die. If you use decoder constraints, you must assign all decode inputs for a given function to the same edge. From the schematic, attach LOC constraints to the decode logic — either a DECODE macro or a WAND gate with the DECODE attribute. The constraints are then passed into the EDIF netlist and after mapping is read by PAR.

The format of decode constraints is LOC= and the decode logic symbol location. If the target symbol represents a soft macro containing only decode logic, for example, DECODE8, the LOC constraint is applied to all decode logic contained in that macro. If the indicated decode logic does not fit into the specified decoders, an error is generated.

To constrain decoders to precise positions within a side, constrain the associated pads. However, because PAR determines decoder edges before processing pad constraints, it is not enough to constrain the pads alone. To constrain decoders to a specific die side, use the following rule. For every output net that you want to constrain, specify the side for at least one of its input decoders (WAND gates), using one of the following.

```
LOC=L          LOC=T
LOC=R          LOC=B
```

The “Legal Edge Designations for Edge Decoders” table shows the legal edge designations.

Example 1

```
Schematic      LOC=T
UCF            INST dec1/$1I1 LOC=T ;
```

Place the decoder along the top edge of the die.

Example 2

```
Schematic      LOC=L
UCF            INST dec1/$1I1 LOC=L ;
```

Place the decoder logic along the left edge of the die.

Example 3

```
Schematic      LOC=LT
UCF            INST dec1/$1I1 LOC=LT ;
```

Place decoders along the top half of the left edge of the die. The first letter in this code represents the die edge, and the second letter represents the desired half of that edge.

Table 0-9 Legal Edge Designations for Edge Decoders

Edge Code	Edge Location
T	Top edge
B	Bottom edge
L	Left edge
R	Right edge
TL	Left half of top edge
TR	Right half of top edge
BL	Left half of bottom edge
BR	Right half of bottom edge
LT	Top half of left edge
LB	Bottom half of left edge
RT	Top half of right edge
RB	Bottom half of right edge

Note: The edges referred to in these constraints are die edges, which do not necessarily correspond to package edges. View the device in EPIC to determine which pins are on which die edge.

Flip-Flop Constraint Examples

Flip-flops can be constrained to a specific CLB, a range of CLBs, a row or column of CLBs, a specific half-CLB, or one of four specific slices of the XC5200 CLB. 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.

Example 1

```
Schematic    LOC=clb_r1c5
UCF          INST /top-12/fdrd LOC=clb_r1c5 ;
```

Place the flip-flop in the CLB in row 1, column 5. CLB R1C1 is in the upper left corner of the device.

Example 2

```
Schematic    LOC=clb_r1c1:clb_r5c7
UCF          INST /top-12/fdrd LOC=clb_r1c1:clb_r5c7 ;
```

Place the flip-flop in the rectangular area bounded by the CLB R1C1 in the upper left corner and CLB R5C7 in the lower right corner.

Example 3

```
Schematic    LOC=clb_r*c3
UCF          INST /top-12/fdrd/top-54/fdsd LOC=clb_r*c3 ;
```

Place the flip-flops in any row of column 3. You can use the wildcard (*) character in place of either the row or column number to specify an entire row or column of CLBs.

In the following example, 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.

Example 4

```
Schematic    LOC=clb_r2c4,clb_r7c9
UCF          INST /top-54/fdsd LOC=clb_r2c4,clb_r7c9 ;
```

Place the flip-flop in either CLB R2C4 or CLB R7C9.

Example 5

```
Schematic    LOC=clb_r3c5.ffx
UCF          INST /top-12/fdrd LOC=clb_r3c5.ffx ;
```

Place the flip-flop in CLB R3C5 and assign the flip-flop output to the XQ pin. (Note: Use the FFY tag to indicate the YQ pin of the CLB.) If either the FFX or FFY tags are specified, the wildcard (*) character cannot be used for the row or column numbers.

Example 6

```
Schematic    PROHIBIT=c1b_r5c*
UCF          CONFIG PROHIBIT=c1b_r5c* ;
```

Do not place the flip-flop in any column of row 5. You can use the wildcard (*) character in place of either the row or column number to specify an entire row or column of CLBs.

The XC5200 CLB is divided into four specific slices for every row and column location on the array. In order to place a flip-flop in a specific slice, use the .LC0, .LC1, .LC2, or .LC3 extension on the location constraint as shown in the following example.

Example 7

```
Schematic    LOC=c1b_r1c5.LC3
UCF          INST /top-12/fdrd LOC=c1b_r1c5.LC3 ;
```

Place the flip-flop in the top slice of the XC5200 CLB in row 1, column 5.

Global Buffer Constraint Examples (XC4000, XC5200)

You can constrain a global buffer — BUFG, BUFGP, BUFGS, BUFGLS, BUFGE, or BUFFCLK— to a corner of the die. From the schematic, attach LOC constraints to the global buffer symbols; specify LOC= and the global clock buffer location. The constraints are then passed into the EDIF netlist and after mapping are read by PAR.

Following is an example.

```
Schematic    LOC=TL
UCF          INST buf1 LOC=TL ;
```

Place the global buffer in the top left corner of the die. The following table shows the legal corner designations.

Table 0-10 Legal Corner Designations for Global Buffers

Corner Code	Corner Location
TL	Top left corner
TR	Top right corner
BL	Bottom left corner
BR	Bottom right corner

If a global buffer is sourced by an external signal, the dedicated IOB for that buffer must not be used by any other signal. For example, if a BUFGP is constrained to TL, the PGCK1 pin must be used to source it, and no other I/O can be assigned to that pin.

You cannot assign placement to the GCLK or ACLK buffers in the XC3000 family, since there is only one each, and their placements are fixed on the die.

I/O 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 Constraint Examples

You can assign I/O pads, buffers, and registers to an individual IOB location or to a specified die edge or half-edge. IOB locations are identified by the corresponding package pin designation or by the edge of the FPGA array.

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 statement places the I/O element in location P13. For PGA packages, the letter-number designation is used, for example, B3.

```
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.

IOBs are prohibited by specifying a PROHIBIT constraint prepended 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 IOBs at pins 36, 37, or 41. For pin grid arrays, pin names such as D14, C16, or H15 are used.

Mapping Constraint Examples

Mapping constraints control the mapping of logic into CLBs. They have two parts. The first part is a FMAP, HMAP, or CLBMAP component placed on the schematic. The second is a LOC constraint that can be placed on the schematic or in the constraints file.

CLBMAP (XC3000 Only)

With the CLBMAP symbol, you can specify logic mapping at the schematic level for all XC3000 designs. It is used in conjunction with standard logic elements, such as gates and flip-flops. It implicitly specifies the configuration of a CLB by defining the signals on its pins. Use the CLBMAP symbol to control mapping when the default mapping is not acceptable.

Enter the CLBMAP symbol on the schematic and assign signals to its pins. MAP processes this symbol and maps the appropriate logic, as defined by the input and output signals, into one CLB. The easiest way to define a CLBMAP is to connect a labeled wire segment to each pin, which connects that pin to the net carrying the same label.

If a CLBMAP specifies an illegal CLB configuration, MAP ignores the CLBMAP and issues a warning explaining why the CLBMAP is illegal.

A CLBMAP can be either closed or open. A closed CLBMAP must specify both the input and output signals for that CLB. MAP maps a closed CLBMAP exactly as specified, unless the indicated configuration is illegal. MAP does not add any logic to a CLB specified with a closed CLBMAP.

An open CLBMAP specifies only the output signals for the CLB, in contrast to the FMAP, which can define inputs as well. MAP assigns those signals to the CLB output pins and maps the source logic into the CLB as appropriate. Use an open CLBMAP to specify the function of a CLB without specifying the exact configuration.

Specify whether a CLBMAP is open or closed by attaching the appropriate MAP attribute to the symbol. See the “Map Attributes for CLBMAP Symbols” table for the exact conventions.

The pins on a CLBMAP can be either locked or unlocked. Specify whether a CLBMAP has locked or unlocked pins by attaching the appropriate attribute to the symbol. See the “Map Attributes for CLBMAP Symbols” table for the exact conventions. With an open CLBMAP, only the output pins are locked.

The default configuration for a CLBMAP is unlocked and closed.

Table 0-11 Map Attributes for CLBMAP Symbols

	Closed CLB	Open CLB
Pins locked	MAP=PLC	MAP=PLO
Pins unlocked	MAP=PUC	MAP=PUO (default)

Note: Currently, pin locking is not supported. PLC and PLO are translated into PUC and PUO, respectively.

Example 1

```
Schematic    LOC=CLB_R1C1
UCF          INST top/cntq7 LOC=CLB_R1C1 ;
```

Place the CLBMAP in CLB CLB_R1C1.

Example 2

```
Schematic    LOC=AA:EE
UCF          INST reg/bit7 LOC=AA:EE ;
```

Place the CLBMAP in the area bounded by CLB AA in the upper left corner and CLB EE in the lower right.

FMAP and HMAP (XC4000, XC5200)

The FMAP and HMAP symbols control mapping in an XC4000 design. They are similar to the XC3000 CLBMAP symbol. The FMAP may also be used to control mapping XC5200 designs.

FMAP and HMAP control the mapping of logic into function generators. These symbols do not define logic on the schematic; instead, they specify 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. The mapper assigns this function to an F or G function generator for the XC4000, so you are not required to specify whether it belongs in F or G. For the XC5200, the four-input function generator defined by the FMAP will be assigned to one of the four slices of the CLB.

The HMAP symbol defines mapping into a three-input (H) function generator for the XC4000. If the HMAP has two FMAP outputs and, optionally, one normal (non-FMAP) signal as its inputs, The mapper places all the logic related to these symbols into one CLB.

An example of how to use these symbols in your schematic appears in the “FMAP and HMAP Schematics” figure and the “Implementation of FMAP and HMAP” figure.

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.)

For the HMAP symbol, only MAP=PUC is supported.

Example 1

```
Schematic    LOC=clb_r7c3
UCF          INST $1I323 LOC=clb_r7c3;
```

Place the FMAP or HMAP symbol in the CLB at row 7, column 3.

Example 2

```
Schematic    LOC=clb_r2c4,clb_r3c4
UCF          INST top/dec0011 LOC=clb_r2c4,clb_r3c4;
```

Place the FMAP or HMAP symbol in either the CLB at row 2, column 4 or the CLB at row 3, column 4.

Example 3

```
Schematic    LOC=clb_r5c5:clb_r10c8
UCF          INST $3I27 LOC=clb_r5c5:clb_r10c8;
```

Place the FMAP or HMAP symbol in the area bounded by CLB R5C5 in the upper left corner and CLB R10C8 in the lower right.

Example 4

```
Schematic    LOC=clb_r10c11.f
```

```
UCF          INST top/done LOC=clb_r10i11.f ;
```

Place the FMAP in the F function generator of CLB R10C11. The .G extension specifies the G function generator. An HMAP can only go into the H function generator, so there is no need to specify this placement explicitly.

The XC5200 CLB is divided into four specific slices for every row and column location in the array. In order to place a function generator in a specific slice, use the .LC0, .LC1, .LC2., or LC3 extension on the location constraint on the FMAP as shown in the following example.

Example 5

```
Schematic    LOC=clb_r10c11.LC3
```

```
UCF          INST /top/done LOC=clb_r10c11.LC3 ;
```

Place the FMAP in the top slice of the XC5200 CLB in row 10, column 11.

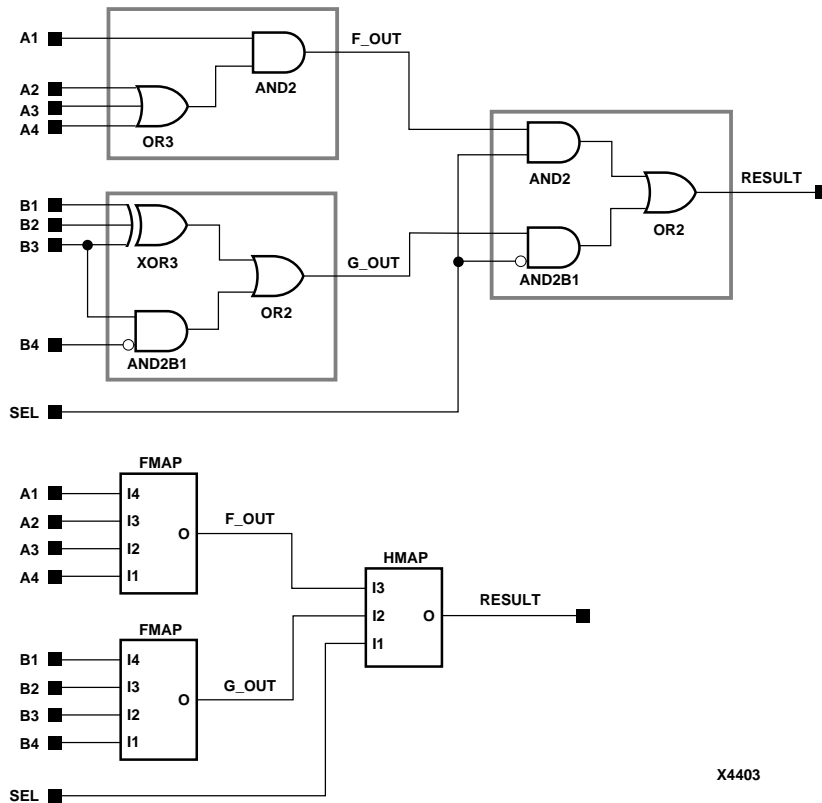


Figure 0-2 FMAP and HMAP Schematics

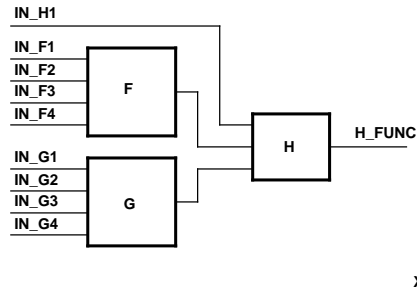


Figure 0-3 Implementation of FMAP and HMAP

ROM and RAM Constraint Examples

You can constrain a ROM or RAM to a specific CLB, a range of CLBs, or a row or column of CLBs. 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 appropriate interface user guide.

Alternatively, in the constraints file a memory is identified by a unique instance name. One or more memory instances of type ROM or RAM 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. The instance name of the RAM primitive, which is a piece of a RAM64X8 macro, is /top-11-ram64x8/ram-3.

Example 1

```
Schematic    LOC=clb_r1c5
UCF          INST /top-7/rq LOC=clb_r1c5 ;
```

Place the memory in the CLB in row 1, column 5. CLB R1C1 is in the upper left corner of the device. You can only apply a single-CLB constraint such as this to a 16 x 1 or 32 x 1 memory.

Example 2

```
Schematic    LOC=clb_r2c4, clb_r7c9
UCF          INST /top-7/rq LOC=clb_r2c4, clb_r7c9 ;
```

Place the memory in either CLB R2C4 or CLB R7C9.

Example 3

```
Schematic    LOC=clb_r1c1:clb_r5c7
UCF          INST /top-17/bigram/*
              LOC=clb_r1c1:clb_r5c7 ;
```

Place the LogiBlox module in the rectangular area bounded by the CLB R1C1 in the upper left corner and CLB R5C7 in the lower right.

From the schematic, attach the LOC constraint to the LogiBlox symbol for the bigram block.

In the UCF file, the /* is appended to the end of the LogiBlox symbol instance. The wildcard (*) character here specifies all instances that begin with the /top-17/bigram/ prefix, that is, all RAM elements within the LogiBlox block.

Example 4

```
Schematic    PROHIBIT clb_r5c*
UCF          CONFIG PROHIBIT=clb_r5c* ;
```

Do not place the memory in any column of row 5. You can use the wildcard (*) character in place of either the row or column number in the CLB name to specify an entire row or column of CLBs.

Relative Location (RLOC) Constraints

Note: This section applies only to FPGA families.

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 location (LOC) constraints constrain design elements to specific locations on the FPGA die with no relation to other design elements.

Benefits and Limitations of RLOC Constraints

RLOC constraints allow you to place logic blocks relative to each other to increase speed and use die resources efficiently. They provide an order and structure to related design elements without requiring you to specify their absolute placement on the FPGA die. They allow you to replace any existing hard macro with an equivalent that can be directly simulated.

In the Unified Libraries, you can use RLOC constraints with BUFT- and CLB-related primitives, that is, DFF, HMAP, FMAP, and CY4 primitives. You can also use them on non-primitive macro symbols. There are some restrictions on the use of RLOC constraints on BUFT symbols; for details, see the “Fixing Members of a Set at Exact Die Locations” section. You cannot use RLOC constraints with decoders, clocks, or I/O primitives. LOC constraints, on the other hand, can be used on all primitives: BUFTs, CLBs, IOBs, decoders, and clocks.

The following symbols (primitives) accept RLOCs.

1. Registers
2. FMAP
3. HMAP
4. F5MAP
5. CY4
6. CY_MUX
7. ROM
8. RAM
9. RAMS, RAMD
10. BUFT
11. WAND primitives that do not have a DECODE attribute attached

Guidelines for Specifying Relative Locations

General syntax for assigning elements to relative locations is

```
RLOC=RmCn [ .extension ]
```

where *m* and *n* are relative row numbers and column numbers, respectively.

The extension uses the LOC extension syntax as appropriate; for example .1 and .2 for TBUF location.

The extension is required for XC5200 designs in order to fully specify the order of the elements (.LC0, .LC1, .LC2, .LC3).

The row and column numbers can be any positive or negative integer including zero. Absolute die locations, in contrast, cannot have zero as a row or column number. Because row and column numbers in RLOC constraints define only the order and relationship between design elements and not their absolute die locations, their numbering can include zero or negative numbers. Even though you can use any integer in numbering rows and columns for RLOC constraints, it is recommended that you use small integers for clarity and ease of use.

It is not the absolute values of the row and column numbers that is important in RLOC specifications but their relative values or differences. For example, if design element A has an RLOC=R3C4 constraint and design element B has an RLOC=R6C7 constraint, the absolute values of the row numbers (3 and 6) are not important in themselves. However, the difference between them is important; in this case, 3 (6 -3) specifies that the location of design element B is three rows away from the location of design element A. To capture this information, a normalization process is used at some point in the design implementation. In the example just given, normalization would reduce the RLOC on design element A to R0C0, and the RLOC on design element B to R3C3.

In Xilinx programs, rows are numbered in increasing order from top to bottom, and columns are numbered in increasing order from left to right. RLOC constraints follow this numbering convention.

The “Different RLOC Specifications for Four Flip-flop Primitives for an XC4000 Design” figure demonstrates the use of RLOC constraints. Four flip-flop primitives named A, B, C, and D are assigned RLOC constraints as shown. These RLOC constraints require each flip-flop to be placed in a different CLB in the same column and stacked in the order shown — A above B, C, and D. Within a CLB, however, they can be placed either in the FFX or FFY position.

If you wish to place more than one of these flip-flop primitives per CLB, you can specify the RLOCs as shown in the “Different RLOC Specifications for Four Flip-flop Primitives for an XC4000 Design” figure. The arrangement in the figure requires that A and B be placed in a single CLB and that C and D be placed in another CLB immediately below the AB CLB. However, within a CLB, the flip-flops can be placed in either of the two flip-flop positions, FFX or FFY.

To control the ordering of these flip-flop primitives specifically, you can use the extension field, as shown in the “Different RLOC Specifications for Four Flip-flop Primitives for an XC4000 Design” figure. In this figure, the same four flip-flops are constrained to use specific resources in the CLBs. This specification always ensures that these elements are arranged exactly as shown— A must be placed in the FFX spot, B in the same CLB at the FFY spot, and so on.

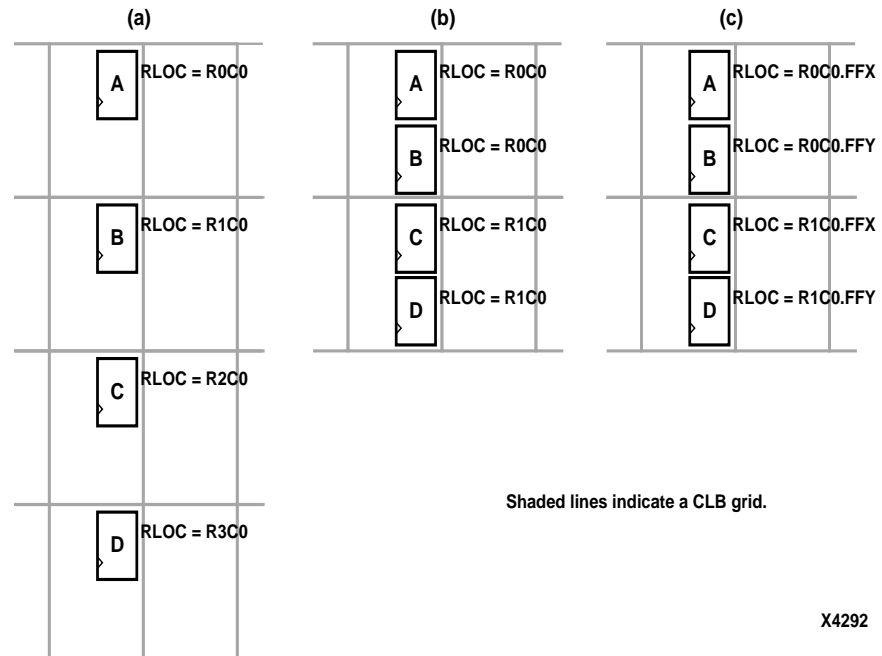


Figure 0-4 Different RLOC Specifications for Four Flip-flop Primitives for an XC4000 Design

RLOC Sets

RLOC constraints give order and structure to related design elements. This section describes RLOC sets, which are groups of related design elements to which RLOC constraints have been applied. For example, the four flip-flops in the “Different RLOC Specifications for Four Flip-flop Primitives for an XC4000 Design” figure are related by RLOC constraints and form a set. 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 one set only.

Sets can be defined explicitly through the use of a set parameter or implicitly through the structure of the design hierarchy.

Four distinct types of rules are associated with each 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.

These rules are discussed in the sections that follow.

The following sections discuss three different set constraints— U_SET, H_SET, and HU_SET. Elements must be tagged with both the RLOC constraint and one of these set constraints to belong to a set.

U_SET

U_SET constraints enable 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. U_SET constraints allow 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 RLOC constraints that are below it in the hierarchy.

The syntax of the U_SET constraint is the following.

```
U_SET=set_name
```

where *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 the sets in the design.

H_SET

In contrast to the U_SET constraint, which you explicitly define by tagging design elements, the H_SET (hierarchy set) is defined implicitly through the design hierarchy. The combination of the design hierarchy and the presence of RLOC constraints on elements defines a hierarchical set, or H_SET set. You do not use an HSET 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 RLOC_ORIGIN or RLOC_RANGE. If you explicitly tag any element with an RLOC_ORIGIN, RLOC_RANGE, U_SET, or HU_SET constraint, it is removed from an H_SET set. Most designs contain only H_SET constraints, since they are the underlying mechanism for relationally placed macros. The RLOC_ORIGIN or RLOC_RANGE constraints are discussed further in the “Fixing Members of a Set at Exact Die Locations” section.

NGDBUILD recognizes the implicit H_SET set, derives its name, or identifier, attaches the H_SET constraint to the correct members of the set, and writes them to the output file.

The syntax of the H_SET constraint as generated by NGDBUILD follows.

```
H_SET=set_name
```

set_name is the identifier of the set and is unique among all the sets in the design. The base name for any H_SET is “hset,” to which NGDBUILD adds a hierarchy path prefix to obtain unique names for different H_SET sets in the NGDBUILD output file.

HU_SET

The HU_SET constraint is a variation of the implicit H_SET (hierarchy 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 syntax of the HU_SET constraint is the following.

`HU_SET=set_name`

where *set_name* is the identifier of the set; it must be unique among all the sets in the design. You must define the base names to ensure unique hierarchically qualified names for the sets after the mapper resolves the design and attaches the hierarchical names as prefixes.

This user-defined name is the base name of the HU_SET set. Like the H_SET set, in which the base name of “hset” 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.

The HU_SET constraint 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 RLOCs 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.

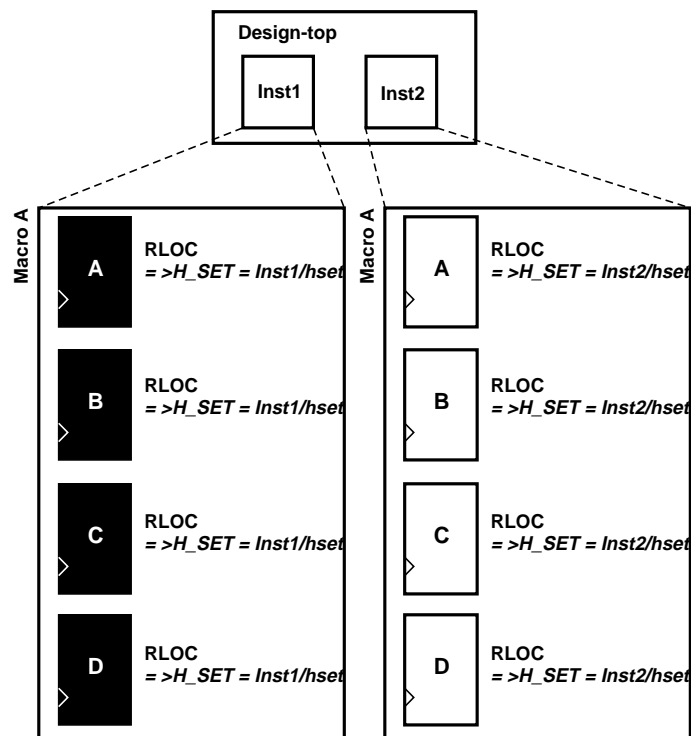


Figure 0-5 Macro A Instantiated Twice

Note: In the “Macro A Instantiated Twice” 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.

The “Macro A Instantiated Twice” figure demonstrates a typical use of the implicit H_SET (hierarchy 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=RmCn. In this example, macro A is originally designed with RLOC constraints on four flip-flops — A, B, C, and D. The macro is then instantiated twice in the design — Inst1 and Inst2. When the design is flattened, two different H_SET sets are recognized because two distinct levels of hierarchy contain elements with RLOC constraints. NGDBUILD creates and attaches the appropriate H_SET constraint to the set members: H_SET=Inst1/hset for the macro instantiated in Inst1, and H_SET=Inst2/hset 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. In the “Macro A Instantiated Twice” figure, Inst1 is the node (instantiating macro) that includes the four flip-flop elements with RLOCs shown on the left of the figure. Therefore, the name of this H_SET set is the hierarchically qualified name of “Inst1” followed by “hset.” 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 “Set Modifiers” section.

The “Macro A Instantiated Twice” 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.

Table 0-12 Summary of Set Types

Type	Definition	Naming	Linkage	Modification
Set	A set is a collection of elements to which relative location constraints are applied.			

Table 0-12 Summary of Set Types

Type	Definition	Naming	Linkage	Modification
U_SET= <i>name</i>	All elements with the same user-tagged U_SET 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 U_SET constraint.	U_SET is modified by applying RLOC_ORIGIN or RLOC_RANGE constraints on, at most, one of the U_SET constraint-tagged elements.
H_SET (implicit through hierarchy) is not available as a constraint that you can attach to symbols.	RLOC on the node. Any other constraint removes a node from the H_SET set.	The lowest common ancestor of the members defines the start of the set. The name is the hierarchically qualified name of the start followed by the base name, "hset."	H_SET links elements to other elements at the same node that do not have other constraints. It links down to all elements that have RLOC constraints and no other constraints. Similarly, it links to other elements up the hierarchy that have RLOC constraints but no other constraints.	H_SET is modified by applying RLOC_ORIGIN and RLOC_RANGE at the start of the set: the lowest common ancestor of all the elements of the set.

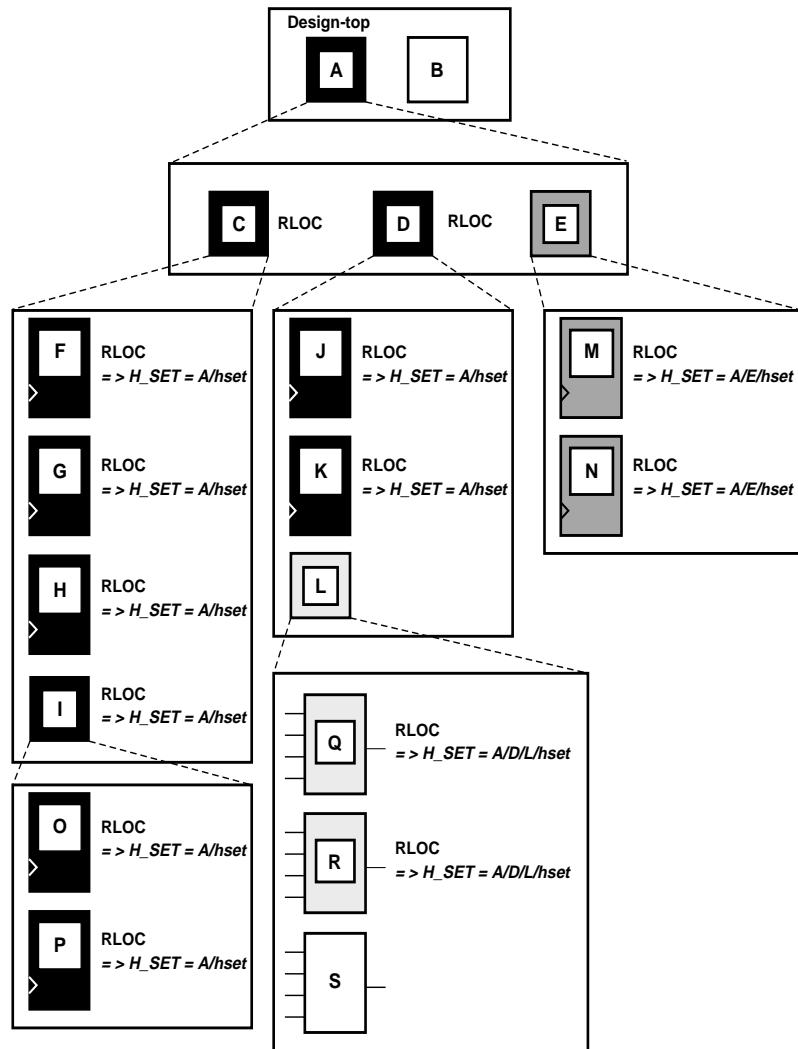
Table 0-12 Summary of Set Types

Type	Definition	Naming	Linkage	Modification
HU_SET= <i>name</i>	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.	HU_SET 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. An RLOC_ORIGIN or an RLOC_RANGE can be applied to, at most, one of these start elements of an HU_SET set.

Set Linkage

The example in the “Three H_SET Sets” figure explains and illustrates the process of linking together elements through the design hierarchy. Again, the complete RLOC specification, RLOC=RmCn, 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.



X4295

Figure 0-6 Three H_SET Sets

As noted previously, 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 assigned another type of set constraint, an RLOC_ORIGIN constraint, or an RLOC_RANGE constraint. In the “Three H_SET Sets” figure, RLOC 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 RLOC constraints were placed on B, E, L, or S. Macros C and D have an RLOC constraint at node A, so all the primitives below C and D that have RLOCs are members of a single H_SET set. Furthermore, the name of this H_SET set is “A/hset” because it is at node A that the set starts. The start of an H_SET set is the lowest common ancestor of all the RLOC-tagged constraints that constitute the elements of that H_SET set. Because element E does not have an RLOC constraint, it is not linked

to the A/hset set. The RLOC-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/hset.”

Similarly, the Q and R primitives are in their own H_SET 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/hset.” After the flattening, NGDBUILD attaches H_SET=A/hset to the F, G, H, O, P, J, and K primitives; H_SET=A/D/L/hset to the Q and R primitives; and H_SET=A/E/hset to the M and N primitives.

Consider a situation in which a set is created at the top of the design. In the “Three H_SET Sets” figure, there would be no lowest common ancestor if macro A also had an RLOC constraint, since A is at the top of the design and has no ancestor. In this case, the base name “hset” would have no hierarchically qualified prefix, and the name of the H_SET set would simply be “hset.”

Set Modification

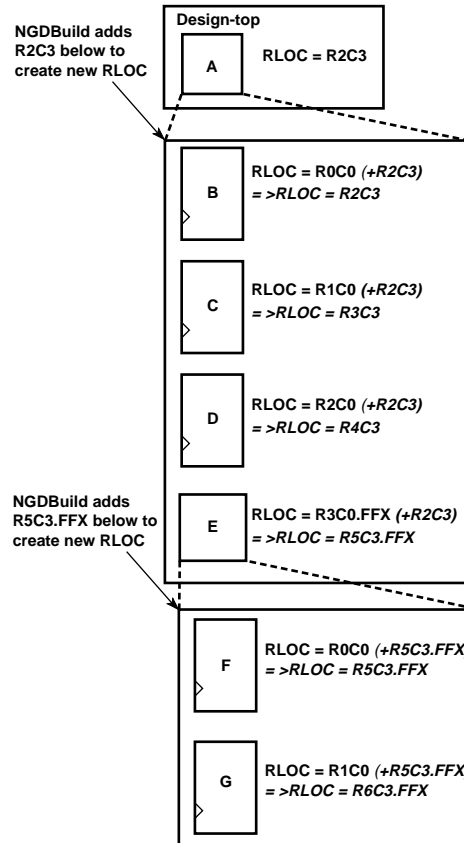
The RLOC constraint assigns a primitive an RLOC value (the row and column numbers with the optional extensions), specifies its membership in a set, and links together elements at different levels of the hierarchy. In the “Three H_SET Sets” figure, the RLOC constraint on macros C and D links together all the objects with RLOC constraints below them. An RLOC 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.

The Effect of the Hierarchy on Set Modification

When the design is flattened, the row and column numbers of an RLOC constraint on an element are added to the row and column numbers of the RLOC constraints of the set members below it in the hierarchy. This feature gives you the ability to modify existing RLOC values in submodules and macros without changing the previously assigned RLOC values on the primitive symbols. This modification process also applies to the optional extension field. However, when using extensions for modifications, you must ensure that inconsistent extensions are not attached to the RLOC value of a design element that may conflict with RLOC extensions placed on underlying elements. For example, if an element has an RLOC constraint with the FFX extension, all the underlying elements with RLOC constraints must either have the same extension, in this case FFX, or no extension at all; any underlying element with an RLOC constraint and an extension different from FFX, such as FFY or F, is flagged as an error.

After resolving all the RLOC constraints, extensions that are not valid on primitives are removed from those primitives. For example, if NGDBUILD generates an FFX extension to be applied on a primitive after propagating the RLOC constraints, it applies the extension if and only if the primitive is a flip-flop. If the primitive is an element other than a flip-flop, the extension is ignored. Only the extension is ignored in this case, not the entire RLOC constraint.

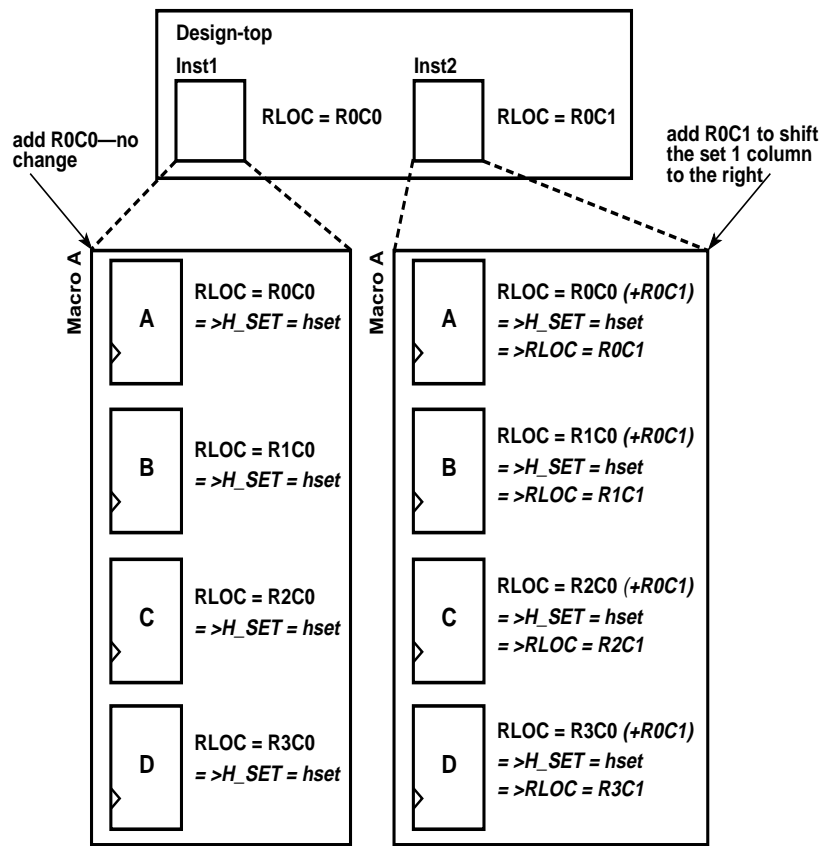
The “Adding RLOC Values Down the Hierarchy” figure illustrates the process of adding RLOC values down the hierarchy. The row and column values between the parentheses show the addition function performed by the mapper. The italicized text prefixed by => is added by MAP during the design resolution process and replaces the original RLOC constraint that you added.



X4296

Figure 0-7 Adding RLOC Values Down the Hierarchy

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 RLOC constraints that are modified when the macro is instantiated. The “Modifying RLOC Values of Same Macro and Linking Together as One Set” figure is a variation of the sample design in the “Macro A Instantiated Twice” figure. The RLOC constraint on Inst1 and Inst2 now link all the objects in one H_SET set. Because the RLOC=R0C0 modifier on the Inst1 macro does not affect the objects below it, the mapper only adds the H_SET tag to the objects and leaves the RLOC values as they are. However, the RLOC=R0R1 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.



X4297

Figure 0-8 Modifying RLOC Values of Same Macro and Linking Together as One Set

Separating Elements from H_SET Sets

The HU_SET constraint is a variation of the implicit H_SET (hierarchy set). The HU_SET constraint defines the start of a new 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 Constraint Linking and Separating Elements from H_SET Sets” figure demonstrates how HU_SET constraints designate elements as set members, break links between elements tagged with RLOC constraints in the hierarchy to separate them from H_SET sets, and generate names as identifiers of these sets.

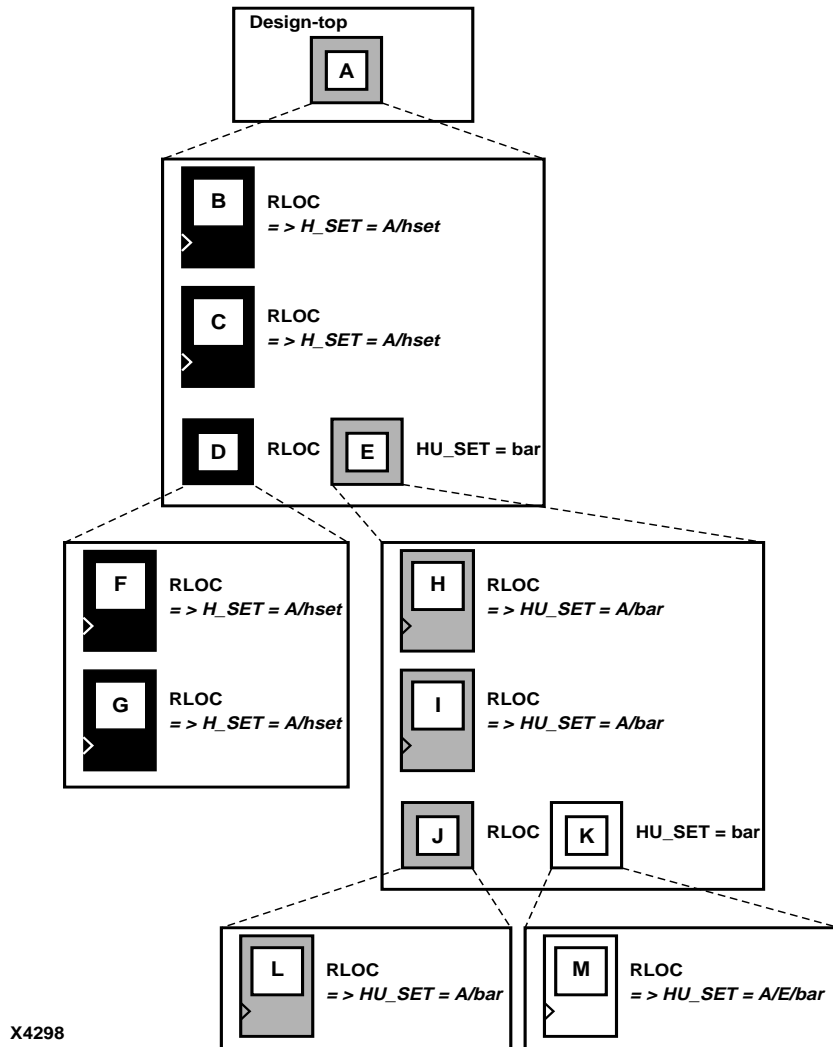
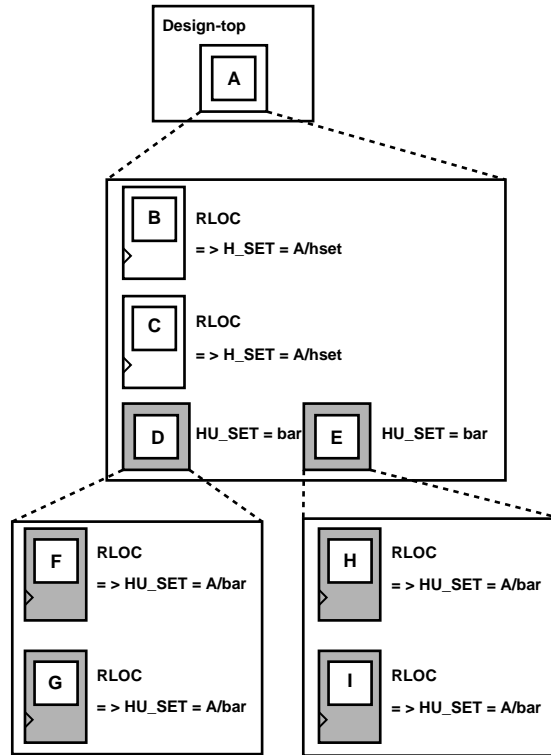


Figure 0-9 HU_SET Constraint Linking and Separating Elements from H_SET Sets

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/hset 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. In the “HU_SET Constraint Linking and Separating Elements from H_SET Sets” figure, 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.



X4299

Figure 0-10 Linking Two HU_SET Sets

The “Linking Two HU_SET Sets” figure 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 RLOC constraints below D and E become part of the same HU_SET.

Set Modifiers

A modifier, as its name suggests, modifies the RLOC constraints associated with design elements. Since it modifies the RLOC 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 easily and intuitively. For this reason, the RLOC modifiers of a set are placed at the start of that set. The following set modifiers apply to RLOC constraints.

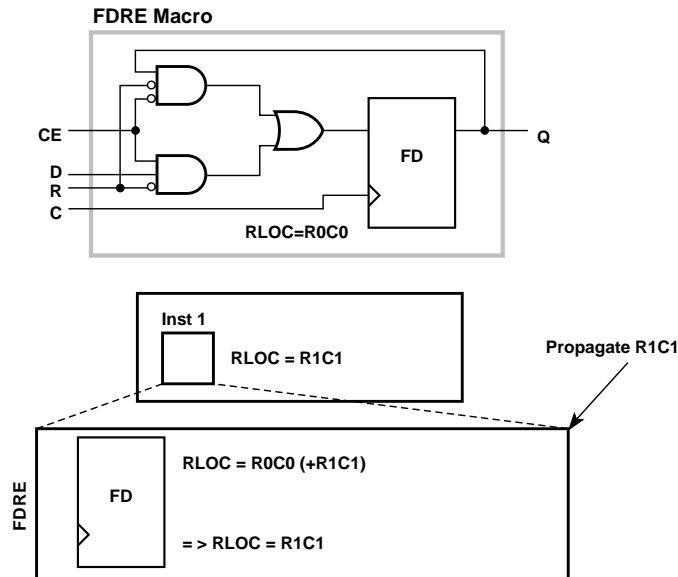
- RLOC

The RLOC constraint associated with a design element modifies the values of *other* RLOC constraints below the element in the hierarchy of the set. Regardless of the set type, RLOC row, column, and extension values on an element always propagate down the hierarchy and are added at lower levels of the hierarchy to RLOC constraints on elements in the same set.

- RLOC_ORIGIN (see the “RLOC_ORIGIN” section)
- RLOC_RANGE (see the “RLOC_RANGE” section)

Using RLOCs with Xilinx Macros

Xilinx-supplied flip-flop macros include an $RLOC=R0C0$ constraint on the underlying primitive, which allows you to attach an RLOC to the macro symbol. This symbol links the underlying primitive to the set that contains the macro symbol. Simply attach an appropriate RLOC constraint to the instantiation of the actual Xilinx flip-flop macro. The mapper adds the RLOC value that you specified to the underlying primitive so that it has the desired value.



X4304

Figure 0-11 Typical Use of a Xilinx Macro

For example, in the “Typical Use of a Xilinx Macro” figure, the $RLOC = R1C1$ constraint is attached to the instantiation (Inst1) of the FDRE macro. It is added to the $R0C0$ value of the RLOC constraint on the flip-flop within the macro to obtain the new RLOC values.

If you do not put an RLOC constraint on the flip-flop macro symbol, the underlying primitive symbol is the lone member of a set. The mapper removes RLOC constraints from a primitive that is the only member of a set or from a macro that has no RLOC objects below it.

LOC Propagation through Design Flattening

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 RLOC 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 RLOC constraint on an element are added to the row and column numbers of the RLOC constraints of the set members below it in the hierarchy. This feature gives you the ability to modify existing RLOC values in submodules and macros without changing the previously assigned RLOC values on the primitive symbols.

Specifying RLOC constraints to describe the spatial relationship of the set members to themselves allows the members of the set to float anywhere on the die as a unit. You can, however, fix the exact die location of the set members. The RLOC_ORIGIN constraint allows you to change the RLOC values into absolute LOC constraints that respect the structure of the set.

The design resolution program, NGDBUILD, translates the RLOC_ORIGIN constraint into LOC constraints. The row and column values of the RLOC_ORIGIN are added individually to the members of the set after all RLOC 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.

Fixing Members of a Set at Exact Die Locations

As noted in the previous section, you can fix the members of a set at exact die locations with the RLOC_ORIGIN constraint. You must use the RLOC_ORIGIN constraint with sets that include BUFT symbols. However, for sets that do not include BUFT symbols, you can limit 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.

The syntax of this constraint is the following:

```
RLOC_RANGE=Rm1Cn1:Rm2Cn2
```

where the relative row numbers ($m1$, $n2$) and column numbers ($n1$, $n2$) can be non-zero positive numbers, or the wildcard (*) character. This syntax allows for three kinds of range specifications:

- $Rr1Cc1:Rr2Cc2$ — A rectangular region enclosed by rows $r1$, $r2$, and columns $c1$, $c2$
- $R*Cc1:R*Cc2$ — A region enclosed by the columns $c1$ and $c2$ (any row number)
- $Rr1C*:Rr2C*$ — A region enclosed by the rows $r1$ and $r2$ (any column number)

For the second and third kinds of specifications with wildcards, applying the wildcard asterisk differently on either side of the separator colon creates an error. For example, specifying $R*C1:R2C*$ is an error since the wildcard asterisk is applied to rows on one side and to columns on the other side of the separator colon.

Specifying a Range or Area

To specify a range or area, use the following syntax, which is equivalent to placing an RLOC_RANGE constraint on the schematic.

```
set_name RLOC_RANGE=Rm1Cn1:Rm2Cn2
```

The range identifies a rectangular area. You can substitute a wildcard (*) character for either the row number or the column number of both corners of the range.

Note: The bounding rectangle applies to all elements in a relationally placed macro, not just to the origin of the set. See the “Relationally Placed Macros (RPMs)” section for more information.

The values of the RLOC_RANGE constraint are not simply added to the RLOC values of the elements. In fact, the RLOC_RANGE constraint does not change the values of the RLOC constraints on underlying elements. It is an additional constraint that is attached automatically by the mapper to every member of a set. The RLOC_RANGE constraint is attached to design elements in exactly the same way as the

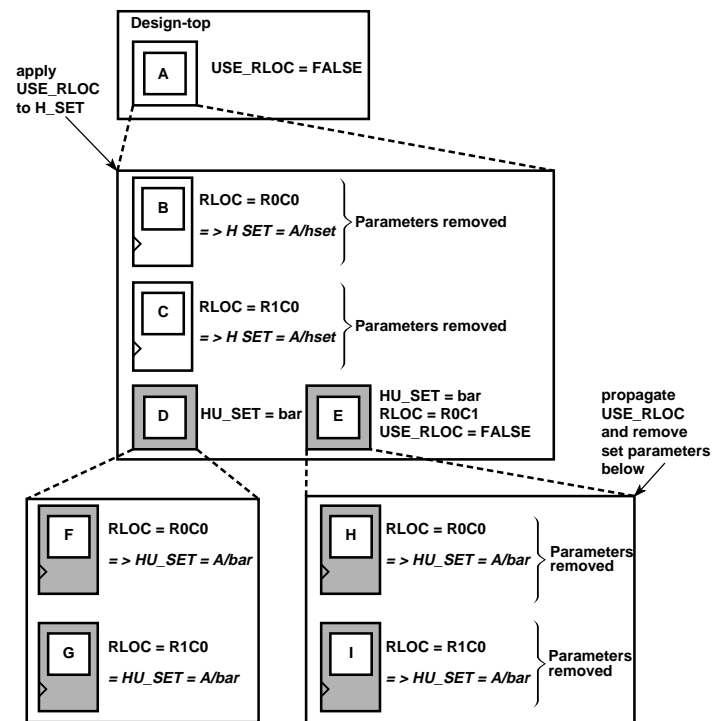
RLOC_ORIGIN constraint. The values of the RLOC_RANGE constraint, like RLOC_ORIGIN values, must be non-zero positive numbers since they directly correspond to die locations.

If a particular RLOC set is constrained by an RLOC_ORIGIN or an RLOC_RANGE constraint in the design netlist and is also constrained in the UCF file, the UCF file constraint overrides the netlist constraint.

Toggleing the Status of RLOC Constraints

Another important set modifier is the USE_RLOC constraint. It turns the RLOC constraints on and off for a specific element or section of a set. RLOC can be either TRUE or FALSE.

The application of the USE_RLOC constraint is strictly based on hierarchy. A USE_RLOC 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. However, if it is applied to an element below the start of the set (for example, E in the “Using the USE_RLOC Constraint to Control RLOC Application on H_SET and HU_SET Sets” figure), only the members of the set (H and I) below the specified element are affected. You can also attach the USE_RLOC constraint directly to a primitive symbol so that it affects only that symbol.



X4302

Figure 0-12 Using the USE_RLOC Constraint to Control RLOC Application on H_SET and HU_SET Sets

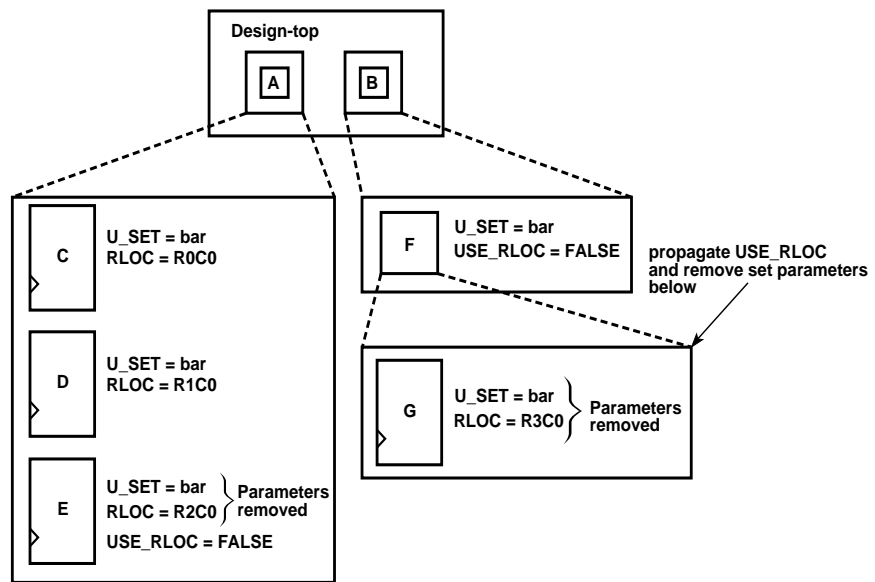
When the USE_RLOC=FALSE constraint is applied, the RLOC and set constraints are removed from the affected symbols in the NCD file. This process is different than that followed for the RLOC_ORIGIN constraint. For RLOC_ORIGIN, the mapper generates and outputs a LOC constraint in addition to all the set and RLOC constraints in the PCF file. The mapper does not retain the original constraints in the presence of a

USE_RLOC=FALSE constraint because these cannot be turned on again in later programs.

The “Using the USE_RLOC Constraint to Control RLOC Application on H_SET and HU_SET Sets” figure illustrates the use of the USE_RLOC constraint to mask an entire set as well as portions of a set.

Applying the USE_RLOC constraint on U_SET sets is a special case because of the lack of hierarchy in the U_SET set. Because the USE_RLOC 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 USE_RLOC constraints; no single USE_RLOC 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_RLOC 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 a 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.

The “Using the USE_LOc Constraint to Control RLOC Application on U_SET Sets” figure illustrates an example of the use of the USE_RLOC=FALSE constraint. The USE_RLOC=FALSE on primitive E removes it from the U_SET set, and USE_RLOC=FALSE on element F propagates to primitive G and removes it from the U_SET set.



X4303

Figure 0-13 Using the USE_LOc Constraint to Control RLOC Application on U_SET Sets

While propagating the USE_RLOC constraint, the mapper ignores underlying USE_RLOC constraints if it encounters elements higher in the hierarchy that already have USE_RLOC constraints. For example, if the mapper encounters an underlying element with a USE_RLOC=TRUE constraint during the propagation of a USE_RLOC=FALSE constraint, it ignores the newly encountered TRUE constraint.

Choosing an RLOC Origin when Using Hierarchy Sets

To specify a single origin for an RLOC set, use the following syntax, which is equivalent to placing an RLOC_ORIGIN attribute on the schematic.

```
set_name RLOC_ORIGIN=RmCn
```

The *set_name* can be the name of any type of RLOC set — a U_SET, an HU_SET, or a system-generated H_SET.

The origin itself is expressed as a row number and a column number representing the location of the elements at RLOC=R0C0.

When the RLOC_ORIGIN constraint is used in conjunction with an implicit H_SET (hierarchy 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 an RLOC_ORIGIN constraint 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 there could be several elements linked together with the HU_SET constraint at the same node, the RLOC_ORIGIN constraint can be applied to only one of these elements to prevent more than one RLOC_ORIGIN constraint from being applied to the HU_SET set.

Similarly, when used with a U_SET constraint, the RLOC_ORIGIN constraint can be placed on only one element with the U_SET constraint. If you attach the RLOC_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 RLOC_ORIGIN constraint attached to the newly created set.

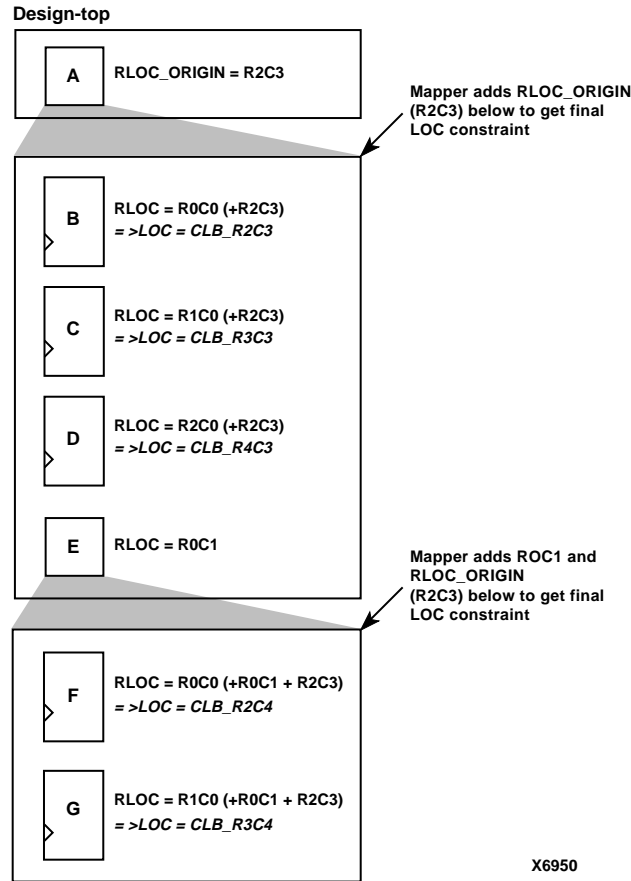
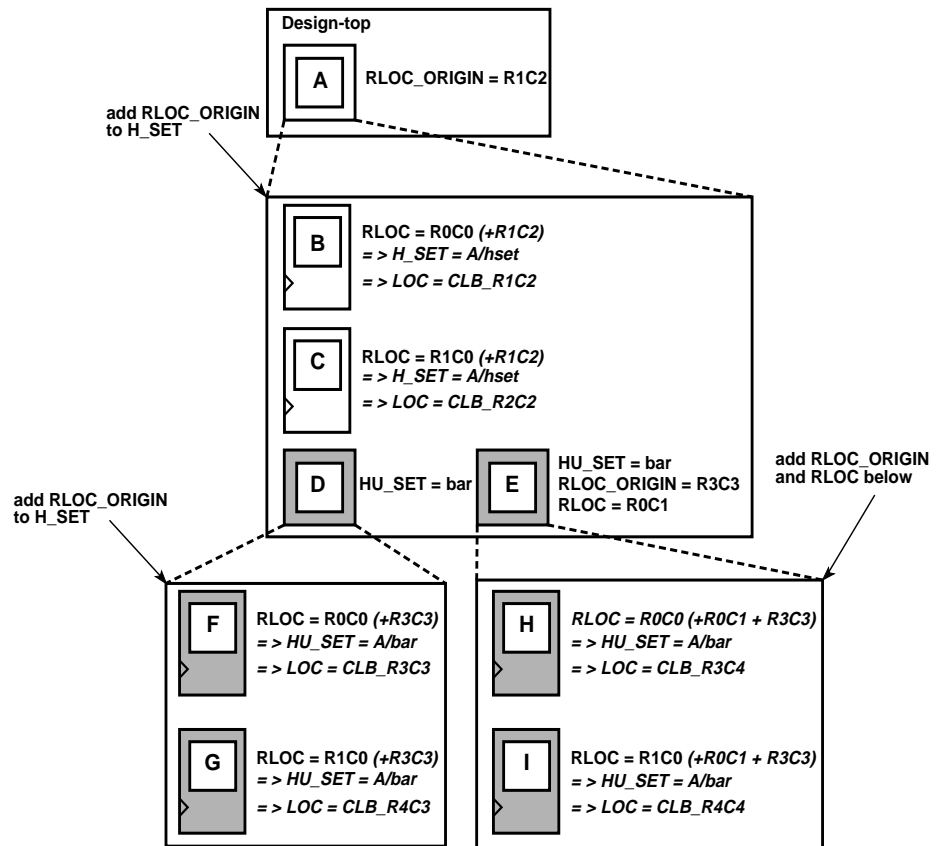


Figure 0-14 Using an RLOC_ORIGIN Constraint to Modify an H_SET Set

In the “Using an RLOC_ORIGIN Constraint to Modify an H_SET Set” figure, the elements B, C, D, F, and G are members of an H_SET set with the name A/hset. This figure is the same as the “Adding RLOC Values Down the Hierarchy” figure except for the presence of an RLOC_ORIGIN constraint at the start of the H_SET set (at A). The RLOC_ORIGIN values are added to the resultant RLOC values at each of the member elements to obtain the values that are then converted by the mapper to LOC constraints. For example, the RLOC value of F, given by adding the RLOC value at E (R0C1) and that at F (R0C0), is added to the RLOC_ORIGIN value (R2C3) to obtain the value of (R2C4), which is then converted to a LOC constraint, LOC = CLB_R2C4.



X4301

Figure 0-15 Using an RLOC_ORIGIN to Modify H_SET and HU_SET Sets

The “Using an RLOC_ORIGIN to Modify H_SET and HU_SET Sets” figure shows an example of an RLOC_ORIGIN constraint modifying an HU_SET constraint. The start of the HU_SET A/bar is given by element D or E. The RLOC_ORIGIN attached to E, therefore, applies to this HU_SET set. On the other hand, the RLOC_ORIGIN at A, which is the start of the H_SET set A/hset, applies to elements B and C, which are members of the H_SET set.

Timing Constraints

This section describes the syntax for using timing constraints in a UCF file. Timing constraints allow you to specify the maximum allowable delay or skew on any given set of paths or nets in your design.

There are three steps for applying timing specifications to a design.

1. Add TNM attributes to symbols on your schematic to group them into sets. This step is not necessary if you are using only predefined sets. This step can be performed in the schematic or in a constraints file. See the “Timing Specifications” section of the “Using Timing Constraints” chapter in the *Development Systems Reference Guide* for instructions.
2. Add a TIMEGRP symbol and add attributes to the symbol. These attributes can combine the sets defined in step 1 or by pattern matching into additional, more complex, sets, or they can match patterns. This step is optional. You can define these groups on the schematic or in the constraints file.
3. Add a TIMESPEC symbol and add attributes to the symbol, defining the timing requirements for the sets defined in steps 1 and 2. You can define the timing requirements on the schematic or in the constraints file.

TNM Attributes

Timing name (TNM) attributes can be used to identify the elements that make up a group and give them a name that can later be used in an actual timing specification. The value of the attribute can take several forms and there are several attachment mechanisms by which the attribute can identify the elements that make up a group.

TNM attributes can be attached to a net, an element pin, a primitive, or a macro.

TNMs on Nets

The TNM attribute can be placed on any net in the design. It is used to indicate that the TNM value should be attached to all valid elements fed by all paths that fan forward from the tagged net. Forward tracing stops at any flip-flop, latch, RAM or pad. TNMs do not propagate across IBUFs if they are attached to the input pad net.

TNMs on Macro or Primitive Pins

The TNM attribute can be placed on any macro or component pin in the design if the design entry package allows placement of attributes on macro or primitive pins. It is used to indicate that the TNM value should be attached to all valid elements fed by all paths that fan forward from the tagged pin. Forward tracing stops at any flip-flop, latch, RAM or pad.

TNMs on Primitives

Attaching a TNM attribute directly to a primitive defines that primitive as part of the named group.

TNMs on Macro Symbols

A TNM attribute attached to a macro indicates that all elements inside the macro (at all levels of hierarchy below the tagged macro) are part of the named group.

TIMEGRP Constraints

It is sometimes convenient to use existing TNMs to create new groups or to define a group based on the output nets that the group sources. A set of grouping mechanisms has been created to do this. The Timing Group primitive (TIMEGRP) serves as the host for these attributes. Because they contain no keyword, the attributes make no sense when used alone.

You can either attach a TIMEGRP constraint to the TIMEGRP schematic symbol or specify it with the TIMEGRP keyword in the UCF file. In the UCF file, the statement syntax is as follows.

```
TIMEGRP timegrp_name=timegrp_parameter
```

where *timegrp_parameter* is identical to the text you would attach to the TIMEGRP schematic symbol.

You can create groups using the following four methods.

1. Combine multiple groups into one; use the following syntax.

```
new_group=group1:group2:...groupn
```

where *new_group* is the group being defined; *group1*, *group2*, and so forth can be a valid TNM-defined group, predefined group (FFS, PADS, RAMS, LATCHES), or group defined with another TIMEGRP attribute. You can create a time group attribute that references another TIMEGRP attribute that appears after the initial definition. Do not use reserved words such as FFS, PADS, RISING, FALLING, or EXCEPT as group names.

Example

```
Schematic    NEWGRP=OLD1:OLD2
UCF          TIMEGRP NEWGRP=OLD1:OLD2 ;
```

2. Create groups by exclusion; use the following syntax.

```
new_group=group1:EXCEPT group2
```

where *new_group* is the group being defined; *group1* and *group2* can be a valid TNM-defined group, predefined group (FFS, PADS, RAMS, LATCHES), or group defined with another TIMEGRP attribute. Do not use reserved words such as FFS, PADS, RISING, FALLING, or EXCEPT as group names.

Example

```
Schematic    FFGRP2=FFS:EXCEPT FFGRP1
UCF          TIMEGRP FFGRP2=FFS:EXCEPT FFGRP1 ;
```

You can also specify multiple groups to include or exclude when creating the new group.

```
new_group=group1:group2:EXCEPT group3:...groupn
```

where *group1*, *group2*, *group3*, and *groupn* can be a valid TNM-defined group, predefined group (FFS, PADS, RAMS, LATCHES), or group defined with another TIMEGRP attribute. Do not use reserved words such as FFS, PADS, RISING, FALLING, or EXCEPT as group names.

3. Define groups of flip-flops triggered by rising and falling clock edges; use the following syntax.

```
new_group={RISING | FALLING group | ffs}
```

where *group* must be a group that includes only flip-flops. FFS is a predefined group.

Example

Defining a group of flip-flops that switch on the falling edge of the clock.

```
Schematic    newfall=FALLING ffs
UCF          TIMEGRP newfall=FALLING ffs ;
```

4. Use wildcard characters to define groups of symbols whose associated signal names match a specific pattern; use this syntax.

```
group=predefined_group pattern
```

where *predefined_group* can be one of the following predefined groups: FFS, PADS, RAMS, LATCHES.

pattern is the string characterizing the output net names of the blocks that you want to include in the new group. It can be any string of characters used with one or more wildcard characters, which can be either of the following.

An asterisk (*) matches any string of zero or more characters.

A question mark (?) matches one character.

Example

Group created by pattern matching.

```
Schematic    newfall=FALLING ffs(A*)
UCF          TIMEGRP newfall=FALLING ffs(A*) ;
```

TIMESPEC Constraints

Once you have defined appropriate groups by attaching TNM attributes to symbols and, optionally, by combining these groups using the TIMEGRP symbol, the next step is to add the timing specifications to the constraints file with the *TSidentifier* constraint. You can define these timing requirements by the following means.

The actual timing specifications take the form of attributes that are attached to a timing specification (TIMESPEC) primitive. The TIMESPEC primitive acts as a place to attach attributes and keeps the attributes together. More than one TIMESPEC primitive can be used in a design at any level of the hierarchy.

The sources and destinations can be any synchronous point in the design. The timing allowance specified is used explicitly by the timing analysis tools, i.e. there is no hidden accounting for any clock inversions between registers clocked by the same clock, etc.

If paths are not specified, they are ignored for the purposes of timing analysis. The forms described here require the definition of a source and a destination for a specification.

Basic Form

Syntax for defining a maximum allowable delay is as follows.

```
TSidentifier=FROM:source_group:TO:dest_group allowable_delay[ units]
```


where

identifier is an ASCII string made up of the characters A...Z, a...z, 0...9.

source_group and *dest_group* are user-defined or predefined groups.

allowable_delay is the timing requirement.

units is an optional field to indicate the units for the allowable delay. Default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to indicate the intended units.

In a schematic the timespec attribute is attached to the TIMESPEC symbol.

Defining Intermediate Points on a Path

It is sometimes convenient to define intermediate points on a path to which a specification applies. This defines the maximum allowable delay and has the following syntax.

```
TSidentifier=FROM:source_group THRU thru_point[THRU thru_point1 . . .  
thru_pointn]:TO:dest_group allowable_delay[units]
```

where

identifier is an ASCII string made up of the characters A...Z, a...z, 0...9.

source_group and *dest_group* are user-defined or predefined groups.

thru_point is an intermediate point used to qualify the path, defined using a TPTHRU attribute.

allowable_delay is the timing requirement.

units is an optional field to indicate the units for the allowable delay. Default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to indicate the intended units.

Worst Case Allowable Delay (MAXDELAY)

Syntax for maximum delay is as follows.

```
TSidentifier=MAXDELAY FROM:source_group:TO:dest_group allowable_delay[units]
```

Syntax for maximum delay using a through point is as follows.

```
TSidentifier=MAXDELAY FROM:source_group THRU thru_point [THRU thru_point1 . . .  
thru_pointn]:TO:dest_group allowable_delay[units]
```

where

identifier is an ASCII string made up of the characters A...Z, a...z, 0...9.

source_group and *dest_group* are user-defined or predefined groups.

thru_point is an intermediate point used to qualify the path, defined using a TPTHRU attribute.

allowable_delay is the timing requirement.

units is an optional field to indicate the units for the allowable delay.

Linked Specifications

This allows you to link the timing number used in one specification to another specification in terms of fractions or multiples.

Note: Circular links are not allowed.

Syntax is as follows.

```
TSidentifier=FROM:source_group:TO:dest_group another_Tsid [/ | *]number
```

where

identifier is an ASCII string made up of the characters A...Z, a...z, 0...9.

source_group and *dest_group* are user-defined or predefined groups.

another_Tsid is the name of another timespec.

number is a floating point number.

Defining Priority for Equivalent Level Specifications

A conflict between two specifications at the same level of priority can be resolved by defining their priority. You can do this by adding the following text to each of the conflicting specifications.

```
normal_timespec_syntax PRIORITY integer
```

where

normal_timespec_syntax is the timing specification.

integer represents the priority. The smaller the number, the higher the priority.

Ignoring Paths

Paths exercising a certain net can be ignored because from a timing specification point of view, all paths through a net, instance, or instance pin may not be important.

Syntax is as follows.

```
TIG=TSidentifier
```

where *identifier* is the timing specification name of the specific timespec for which any paths through the tagged object should be ignored. The attribute can be attached to a net, macro pin or primitive pin. Paths that fan forward from the attribute's point of application are treated as if they don't exist from the viewpoint of timing analysis against the timing specification.

Examples

The following attribute would be attached to a net to inform the timing analysis tools that it should ignore paths through the net for specification TS43.

```
TIG=TS43
```

The following attribute would be created in a UCF file to inform the timing analysis tools that it should ignore paths through the net \$1I567/sometimes_slow for specification TS_fast and TS_really_fast.

```
NET $1I567/sometimes_slow TIG=TS_fast , TS_really_fast;
```

Ignoring Paths Through Primitives

The tracing rules for how PAR's timing analysis handles the traversal of primitives are the same as those used for user driven timing analysis. If a user wishes to override the default behavior for an element the element can be tagged with an override attribute.

Defining a Clock Period

A clock period specification is used to define to the timing analysis tools the allowable time for paths between elements clocked by the flagged clock signal.

Note: The definition of a clock period is different from a FROM:TO style specification, because the timing analysis tools will automatically take into account any inversions of the clock signal at register clock pins.

There are two methods for specifying clock periods.

1. The quick, convenient way to define the clock period for registers attached to a particular clock net is to attach the following parameter directly to a net in the path that drives the register clock pin(s).

```
PERIOD=period [units] [HIGH | LOW] [high_or_low_time] [hi_lo_units]
```

where

period is the required clock period.

units is an optional field to indicate the units for the clock period. The default units are nanoseconds, but the timing number can be followed by ps, ns, us, or ms to indicate the intended units.

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%.

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, us, ms, or % if the high or low time is an actual time measurement.

The PERIOD constraint is forward-traced in exactly the same fashion as a TNM would be and attaches itself to all of the flip-flops that the forward tracing reaches. There are no rules about not tracing through certain elements. If you need a more complex form of tracing behavior, for example, where gated clocks are used in the design, you must place the PERIOD on a particular net, or use the preferred method as described in the following paragraphs.

2. The preferred method for defining a clock period allows more complex derivative relationships to be defined as well as a simple clock period. The following attribute is attached to a TIMESPEC symbol in conjunction with a TNM attribute attached to the relevant clock net.

```
TSidentifier=PERIOD TNM_reference period [units][HIGH | LOW] [high_or_low_time]  
[hi_lo_units]
```

where

identifier is a reference identifier that has a unique name.

TNM_reference is the identifier name that is attached to a clock net (or a net in the clock path) using a TNM attribute.

period is the required clock period.

units is an optional field to indicate the units for the clock period. Default units are nanoseconds, but the timing number can be followed by ps, ns, us, or ms to indicate the intended units.

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 %, but the high or low time number can be followed by % or ps, ns, us or ms if the high or low time is an actual time measurement.

Example

Clock net `sys_clk` has the attribute `tnm=master_clk` attached to it and the following attribute is attached to a TIMESPEC primitive.

```
TS_master=PERIOD master_clk 50 HIGH 30
```

Specifying Derived Clocks

The preferred method of defining a clock period uses an identifier, allowing another clock period specification to reference it. To define the relationship in the case of a derived clock, use the following syntax.

```
TSidentifier=PERIOD TNM_reference another_PERIOD_identifier [ / | * ] number [ HIGH  
| LOW ] [ high_or_low_time ] [ hi_lo_units ]
```

where

identifier is a reference identifier that has a unique name.

TNM_reference is the identifier name that is attached to a clock net or a net in the clock path using a TNM attribute.

another_PERIOD_identifier is the name of the identifier used on another period specification.

number is a floating point number.

high_or_low_time is the optional high or low time. This must be less than the period, depending on the preceding keyword. 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, us, ms, or % if the high or low time is an actual time measurement.

Example

Clock net `sub_clk` has the attribute `tnm=slave_clk` attached to it and the following attribute is attached to a TIMESPEC primitive.

```
ts_slave1=PERIOD slave_clk master_clk * 4
```

Controlling Net Skew

You can control the maximum allowable skew on a net by attaching the MAXSKEW attribute directly to the net. Syntax is as follows.

```
MAXSKEW=allowable_skew [ units ]
```

where

allowable_skew is the timing requirement.

units is an optional field to indicate the units for the allowable delay. Default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to indicate the intended units.

Controlling Net Delay

You can control the maximum allowable delay on a net by attaching the MAXDELAY attribute directly to the net. Syntax is as follows.

MAXDELAY=*allowable_delay* [*units*]

where

allowable_delay is the timing requirement.

units is an optional field to indicate the units for the allowable delay. Default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to indicate the intended units.

Physical Constraints

Note: The information in this section applies only to FPGA families.

When a design is mapped, the logical constraints found in the netlist and the UCF file are translated into physical constraints; that is, constraints that apply to a specific architecture. These constraints are found in a mapper-generated file called the Physical Constraints File (PCF). The file contains two sections, the schematic section and the user section. The schematic section contains the physical constraints based on the logical constraints found in the netlist and the UCF file. The user section can be used to add any physical constraints.

PCF File Syntax

The structure of the PCF file is as follows.

```
schematic start;
translated schematic and UCF or NCF constraints in PCF format
schematic end;
user-entered physical constraints
```

You should put all user-entered physical constraints after the “schematic end” statement.

Note: 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.

The end of each constraint statement must be indicated with a semi-colon.

Note: In all of the constraints files (NCF, UCF, and PCF), instance or variable names that match internal reserved words will be 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;
or
NET "$SIG_0" FAST ;
```

Syntax Descriptions

A description of each legal physical constraint follows.

Note: Although this section describes the constraint’s syntax for the PCF file, it is preferable to place any user-generated constraint in the UCF file — *not* in an NCF or PCF file.

COMPGRP

Description

Identifies a group of components.

Syntax

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

where

comp_item is one of the following,

- COMPONENT "*comp_name*"
- COMPGRP "*group_name*"

FREQUENCY

Description

Identifies the minimum operating frequency for all input pads and sequential output to sequential input pins clocked by the specified net. If no net name is given, the constraint applies to all clock nets in the design that do not have a specific clock frequency constraint.

Syntax

```
TSidentifier=FREQUENCY frequency_item frequency_value ;
```

```
frequency_item FREQUENCY=frequency_value;
```

where

frequency_item is one of the following,

- NET "*net_name*"
- TIMEGRP "*group_name*"
- ALLCLOCKNETS

frequency_value is one of the following,

- *frequency_number units*
 - *units* can be GHz, MHz or KHz (gigahertz, megahertz, or kilohertz)
- TSidentifier [{/ |*} *real_number*]

INREG

Description

Forces the placement of a flip-flop or latch close to the IOB so that the two elements can be connected using fast routes. Because XC5200 IOBs do not have flip-flops or latches, you can apply this attribute to meet fast setup timing requirements if a flip-flop or latch is driven by an IOB.

Syntax

```
NET "net_name" INREG ;
```

where *net_name* is the name of the net that connects the IOB to the INREG instance.

LOCATE

Description

Specifies a single location, multiple single locations, or a location range.

Syntax

Single or multiple single locations

```
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

```
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];
```

where

site_name is a component site (that is, a CLB or IOB location).

site_item is one of the following,

- SITE "*site_name*"
- SITEGRP "*site_group_name*"

n is 0, 1, 2, 3, or 4.

LOCK

Description

Locks a net that has been previously placed or routed (that is, cannot be unplaced, unrouted, moved, swapped, or deleted). Can also be used to lock all nets.

Syntax

A specific net

```
"net_name" LOCK;
```

All nets

```
ROUTING LOCK;
```


MAXDELAY

Description

Identifies a maximum total delay for a net or path in the design. If a net is specified, the maximum delay constraint applies to all driver-to-load connections on the net. If a path is specified, the delay value is the constraint for the path including net and component delays.

Syntax

```
TSidentifier=MAXDELAY path path_value [PRIORITY integer];
path MAXDELAY=path_value [PRIORITY integer];
net_delay_item MAXDELAY=delay_time [units] [PRIORITY integer];
```

where

path is one of the following,

- PATH “*path_name*”
- ALLPATHS
- FROM *group_item* THRU *group_item1*... *group_itemn*
- FROM *group_item* THRU *group_item1*... *group_itemn* TO *group_item*
- THRU *group_item1*... *group_itemn* TO *group_item*.

path_value is one of the following,

- *delay_time* [*units*]
 - *units* defaults to nanoseconds, but the delay time number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, microseconds, or milliseconds) to specify the units
- *frequency units*
 - *units* can be specified as GHz, MHz, or KHz (gigahertz, megahertz, or kilohertz)
- TSidentifier [{/ |*} *real_number*]

net_delay_item is one of the following:

- NET “*net_name*”
- TIMEGRP “*group_name*”
- ALLCLOCKNETS

MAXSKEW

Description

Specifies a maximum signal skew between a driver and loads on a specified clock signal. Skew is the difference between minimum and maximum load delays on a clock net. If no signal is specified, this constraint applies to all signals which have clock pins as loads and do not have a specified skew constraint.

Syntax

```
skew_item MAXSKEW=time [units];
```

where

skew_item is one of the following,

- NET "*net_name*"
- TIMEGRP "*group_name*"
- ALLCLOCKNETS

units defaults to nanoseconds, but the timing number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, microseconds, or milliseconds) to indicate the intended units.

OFFSET

Description

Specifies the timing relationship between an external clock and its associated data-in- or data-out-pin.

Can be used on a group of one or more data components or pads.

Syntax

```
[COMP "iob_name"] OFFSET={IN | OUT} offset_time [units] {BEFORE | AFTER}  
COMP "clk_iob_name";
```

where

offset_time is the external offset.

units defaults to nanoseconds, but the timing number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, microseconds, or milliseconds) to indicate the intended units.

clk_iob_name is the block name of the clock IOB.

OUTREG

Description

Forces the placement of a flip-flop or latch close to the IOB so that the two elements can be connected using fast routes. Because XC5200 IOBs do not have flip-flops or latches, you can apply this attribute to meet fast setup timing requirements if a flip-flop or latch is driven by an IOB.

Syntax

```
NET "net_name" OUTREG;
```

where *net_name* is the name of the net that connects the IOB to the OUTREG instance and *value* is an integer that is 50 or greater.

PATH

Description

Assigns a path specification to a path.

Syntax

```
PATH "path_name"=path_spec;
```

where

path_spec is one of the following,

- FROM *group_item* THRU *group_item1*... *group_itemn*
- FROM *group_item* THRU *group_item1*... *group_itemn* TO *group_item*
- THRU *group_item1*... *group_itemn* TO *group_item*.

group_item is one of the following,

- PIN "*pin_name*"
- NET "*net_name*"
- COMP "*comp_name*"
- MACRO "*macro_name*"
- TIMEGRP "*group_name*"
- BEL "*instance_name*"

BEL *instance_name* is the instance name of a basic element. Basic elements are the building blocks that make up a CLB— function generators, flip-flops, carry logic, and RAMs.

PENALIZE TILDE

Description

Penalizes those delays that are reported as only approximate (signified with a tilde (~) in delay reports) by a user-specified percentage. When the penalize tilde constraint is applied to an approximate delay, the delay will be penalized by the designated percentage in subsequent timing checks. Default for percent value is zero.

Syntax

```
PENALIZE TILDE=percent
```

PERIOD

Description

Assigns a timing period to a timing specification.

Syntax

```
TSidentifier=PERIOD period_item period_value [ {LOW | HIGH} {time [units] | percent} ];  
period_item PERIOD=period_value [ {LOW | HIGH} {time [units] | percent} ] ;
```

where

period_item is one of the following,

- NET "*net_name*"
- TIMEGRP "*group_name*"
- ALLCLOCKNETS

period_value is one of the following,

- *time* [*units*]
 - *units* defaults to nanoseconds, but the timing number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, microseconds, or milliseconds) to indicate the intended units.
- TS *identifier* [{/ | *} *real_number*]

PIN

Description

Identifies a specific pin.

Syntax

```
PIN "pin_name"=pin_spec;
```

where

pin_spec is one of the following,

- NET "*net_name*" BEL "*instance_name*"
- NET "*net_name*" COMP "*comp_name*"
- COMP "*comp_name*" NET "*net_name*"
- NET "*net_name*" MACRO "*macro_name*"
- MACRO "*macro_name*" NET "*net_name*"
- BEL "*instance_name*" NET "*net_name*"

BEL *instance_name* is the instance name of a basic element. Basic elements are the building blocks that make up a CLB— function generators, flip-flops, carry logic, and RAMs.

PRIORITIZE

Description

Assigns a weighted importance to a net or bus. Values range from 0–100, with 100 being the highest priority and 0 the lowest. The default is 3. Any net with a priority of 3 is not considered critical; no constraint will be generated. The prioritize constraint is used by PAR, which assigns longlines by net priority and routes higher-priority nets before routing lower-priority nets. The prioritize constraint is also used by BITGEN to determine which nets not to use for tiedown. A net with a priority greater than 3 will only be used for tiedown as a last resort.

Syntax

```
net "net_name" PRIORITIZE=integer;
```

PROHIBIT

Description

Disallows the use of a site or multiple sites within PAR and EPIC.

Syntax

Single or multiple single locations

```
PROHIBIT= site_group;  
PROHIBIT= site_group1... ,site_groupn;
```

Range of locations

```
PROHIBIT= site_group : site_group;
```

where

site_group is one of the following,

- SITE "*site_name*"
- SITEGRP "*site_group_name*"

site_name must be a valid site for the targeted device. (For example, CLB_R1C1.FFX is not a valid site for the XC4000X.)

SITEGRP

Description

Identifies a group of sites.

Syntax

```
SITEGRP site_group_name=site_group1... site_groupn ; [EXCEPT site_group];
```

where

site_group is one of the following,

- SITE "*site_name*"
- SITEGRP "*site_group_name*"

site_name must be a valid site for the targeted device. (For example, CLB_R1C1.FFX is not a valid site for the XC4000X.)

TIMEGRP (Timing Group)

Description

Defines objects that are to be treated as a group for timing considerations. You can refer to a group of flip-flops, input latches, pads, or RAMs by using the corresponding keywords.

Keyword	Description
FFS	CLB or IOB flip-flops only; not flip-flops built from function generators
LATCHES	CLB or IOB latches only; not latches built from function generators
PADS	Input/output pads
RAMS	For architectures with RAMS

Syntax

```
TIMEGRP "group_name"=[qualifier1] group_spec1... [qualifiern] group_specn [EXCEPT
group_spec1... group_specn];
```

where

qualifier is RISING or FALLING.

group_spec is one of the following,

- PIN "*pin_name*"
- NET "*net_name*"
- BEL "*instance_name*"
- COMP "*comp_name*"
- MACRO "*macro_name*"
- TIMEGRP "*group_name*"
- FFS [*pattern*]
- LATCHES [*pattern*]
- RAMS [*pattern*]
- PADS [*pattern*]

BEL *instance_name* is the instance name of a basic element. Basic elements are the building blocks that make up a CLB— function generators, flip-flops, carry logic, and RAMs.

This example shows you one way to use the TIMEGRP attribute. If you have some outputs that can be slower than others, you can create timespecs similar to those shown below for output signals `obc_data(7:0)` and `ingr_irq_n`.

First create the Timegroups.

```
TIMEGRP slow_outs=PADS(obc_data* : ingr_irq_n) ;
TIMEGRP fast_outs=PADS : EXCEPT : slow_outs ;
```

Then apply a timing spec to the Timegroups.

```
TIMESPEC TS08=FROM : FFS : TO : fast_outs : 22 ;
TIMESPEC TS09=FROM : FFS : TO : slow_outs : 75 ;
```

TIG (Timing Ignore)

Description

Identifies paths that can be ignored for timing purposes.

Syntax

```
ignore_item TIG [=TSidentifier1... TSidentifiern]
```

where

ignore_item is one of the following,

- PIN “*pin_name*”
- NET “*net_name*”
- COMP “*comp_name*”
- MACRO “*macro_name*”
- PATH “*path_name*”
- BEL “*instance_name*”
- FROM *group_item* THRU *group_item1*... *group_itemn*
- FROM *group_item* THRU *group_item1*... *group_itemn*TO *group_item*
- THRU *group_item*... *group_itemn* TO *group_item* }

BEL *instance_name* is the instance name of a basic element. Basic elements are the building blocks that make up a CLB— function generators, flip-flops, carry logic, and RAMs.

For a detailed description of TIG, see the “Timing Specifications” section of the “Using Timing Constraints” chapter in the *Development System Reference Guide*.

TSidentifier

Description

Assigns a timing period or frequency to a timing specification.

Syntax

Period

```
TSidentifier=PERIOD period_item period_value [ {LOW | HIGH} {time [units] | percent} ] ;
period_item PERIOD=period_value [ {LOW | HIGH} {time [units] | percent} ] ;
```

where

period_item is one of the following,

- NET “*net_name*”

- TIMEGRP “*group_name*”
- ALLCLOCKNETS

period_value is one of the following,

- *time* [*units*]
 - *units* defaults to nanoseconds, but the timing number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, microseconds, or milliseconds) to indicate the intended units.
- TS *identifier* [{/ | *} *real_number*]

Frequency

```
TSidentifier=FREQUENCY frequency_item frequency_value ;  
frequency_item FREQUENCY=frequency_value;
```

where

frequency_item is one of the following,

- NET “*net_name*”
- TIMEGRP “*group_name*”
- ALLCLOCKNETS

frequency_value is one of the following,

- *frequency_number* *units*
 - *units* can be GHz, MHz or KHz (gigahertz, megahertz, or kilohertz)
- TS*identifier* [{/ | *} *real_number*]

Relationally Placed Macros (RPMs)

The Xilinx libraries contain three types of elements.

- Primitives are basic logical elements such as AND2 and OR2 gates.
- Soft macros are schematics made by combining primitives and sometimes other soft macros.
- Relationally placed macros (RPMs) are soft macros that contain relative location constraint (RLOC) information, carry logic symbols, and FMAP/HMAP symbols, where appropriate.

The last item mentioned above, RPMs, applies only to FPGA families.

The relationally placed macro (RPM) library uses RLOC constraints to define the order and structure of the underlying design primitives. Because these macros are built upon standard schematic parts, they do not have to be translated before simulation. The components that are implemented as RPMs are listed in the “Relationally Placed Macros” section of the “Selection Guide” chapter.

Designs created with RPMs can be functionally simulated. RPMs can, but need not, include all the following elements:

- FMAPs, HMAPs, and CLB-grouping attributes to control mapping. FMAPs and HMAPs have pin-lock attributes, which allow better control over routing. FMAPs and HMAPs are described in the “Mapping Constraint Examples” section.
- Relative location (RLOC) constraints to provide placement structure. They allow positioning of elements relative to each other. They are discussed in the “Benefits and Limitations of RLOC Constraints” section.
- Carry logic primitive symbols. Carry logic is discussed in the “Carry Logic in XC4000 FPGAs” section.

The RPM library offers the functionality and precision of the hard macro library with added flexibility. You can optimize RPMs and merge other logic within them. The elements in the RPM library allow you to access carry logic easily and to control mapping and block placement. Because RPMs are a superset of ordinary macros, you can design them in the normal design entry environment. They can include any primitive logic. The macro logic is fully visible to you and can be easily back-annotated with timing information.

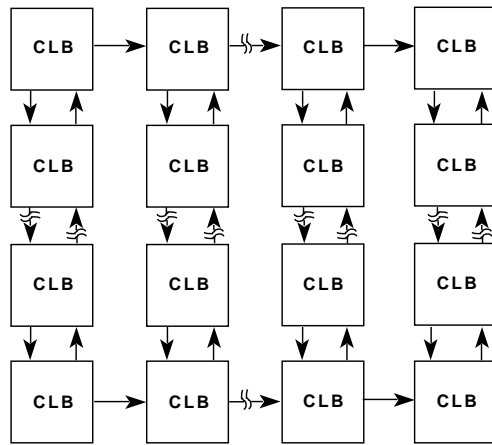
Carry Logic in XC4000 FPGAs

The XC4000 CLB contains a feature called dedicated carry logic. This carry logic is independent of the function generators, although it shares some of the same input pins. Dedicated interconnect propagates carry signals through a column of CLBs.

This section describes the use of carry logic in XC4000 CLBs and lists all the carry logic configuration mnemonics available.

Differences Between XC4000E and XC4000X Carry Logic

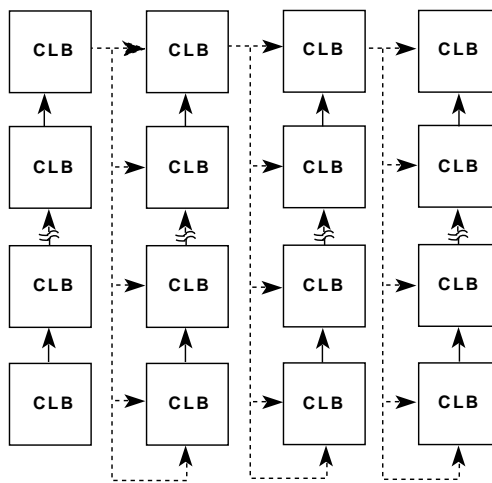
The carry chain in XC4000E devices can run either up or down. At the top and bottom of columns where there are no CLBs above and below, the carry is propagated to the right as shown in the figure below.



X8023

Figure 0-16 Available XC4000E Carry Propagation Paths

In XC4000X devices the carry chain travels upward only. Standard interconnect can be used to route a signal in the downward direction. See the figure below.

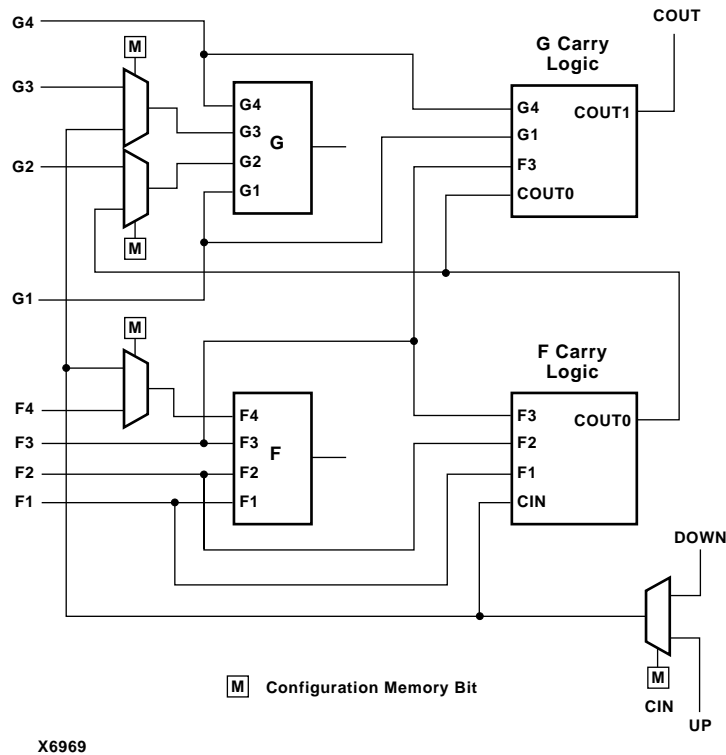


X8024

Figure 0-17 Available XC4000X Carry Propagation Paths (dotted lines use general interconnect)

The CY4_43 carry mode component (Force-G4) forces the signal on the G4 pin to pass through to the COUT pin. This component is available only for XC4000X devices.

Carry logic in each CLB can implement approximately 40 different functions, which you can use to build faster and more efficient adders, subtracters, counters, comparators, and so forth. The “XC4000 Carry Logic” figure shows the carry logic in an XC4000 CLB.



X6969
Figure 0-18 XC4000 Carry Logic

Carry Logic Primitives and Symbols

The schematic capture libraries that Xilinx supports contain one generic carry logic primitive and several specific carry mode primitive symbols. The generic carry logic primitive represents the complete carry logic in a single CLB and is shown in the “Representative Carry Logic Symbol” figure.

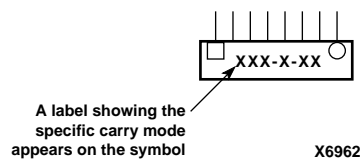


Figure 0-19 Representative Carry Logic Symbol

The carry mode primitive symbols represent unique carry modes, such as ADD-FG-CI. The “Carry Modes” table lists the carry mode names and symbols.

To specify the particular mode that you wish, connect a carry mode symbol to the C0-C7 mode pins of the carry logic symbol. It is the pair of symbols that defines the specific kind of carry logic desired.

A carry logic symbol requires you to place either a LOC or an RLOC constraint on it. If a LOC constraint is used, it must be a single LOC= constraint; it cannot be an area or prohibit LOC constraint or use wildcards in its syntax.

Table 0-13 Carry Modes

Carry Mode Name	Symbol
ADD-F-CI	cy4_01
ADD-FG-CI	cy4_02
ADD-G-F1	cy4_03
ADD-G-CI	cy4_04
ADD-G-F3-	cy4_05
ADDSUB-F-CI	cy4_12
ADDSUB-FG-CI	cy4_13
ADDSUB-G-CI	cy4_15
ADDSUB-G-F1	cy4_14
ADDSUB-G-F3-	cy4_16
FORCE-0	cy4_37
FORCE-1	cy4_38
FORCE-CI	cy4_40
FORCE-F1	cy4_39
FORCE-F3-	cy4_41
FORCE-G4	cy4_43*
EXAMINE-CI	cy4_42
DEC-F-CI	cy4_24
DEC-FG-0	cy4_26
DEC-FG-CI	cy4_25
DEC-G-0	cy4_27
DEC-G-CI	cy4_29
DEC-G-F1	cy4_28
DEC-G-F3-	cy4_30
INC-F-CI	cy4_17
INC-FG-1	cy4_19
INC-FG-CI	cy4_18
INC-G-1	cy4_20
INC-G-CI	cy4_22
INC-G-F1	cy4_21
INC-G-F3-	cy4_23
SUB-F-CI	cy4_06
SUB-FG-CI	cy4_07
SUB-G-1	cy4_08

Table 0-13 Carry Modes

Carry Mode Name	Symbol
SUB-G-CI	cy4_09
SUB-G-F1	cy4_10
SUB-G-F3-	cy4_11
INCDEC-F-CI	cy4_31
INCDEC-FG-1	cy4_33
INCDEC-FG-CI	cy4_32
INCDEC-G-0	cy4_34
INCDEC-G-CI	cy4_36
INCDEC-G-F1	cy4_35

*Available only for XC4000X devices

Carry Logic Handling

NGDBUILD and the mapper check for legal connections between carry logic symbols and also perform simple trimming on some carry modes. CY4 symbols might be trimmed as follows:

- If neither the COUT0 pin nor the COUT pin is used, the CY4 symbol is removed from the design. However, if the signal on the CIN pin connects to other logic, the mapper converts the CY4 to the EXAMINE-CI mode. An EXAMINE-CI mode CY4 is trimmed only if there is no other load on the signal on the CIN pin.
- If the specified mode does not require any of the A0, B0, A1, B1, and/or ADD CY4 inputs, signals are removed from these pins, which may save routing resources.

Carry Mode Configuration Mnemonics

The first step in configuring a CLB for carry logic is to choose the appropriate carry mode configuration mnemonic. Each of the 42 possible configurations of the carry logic has been assigned a three-part mnemonic code, for example:

ADD-FG-CI

- The first field (ADD) describes the operation performed in the CLB function generators, in this case, a binary addition. By implication, the carry logic in this CLB calculates the carry for this addition.
- The second field (FG) indicates which of the two function generators is used in the specified operation, in this case, both F and G.
- The last field (CI) specifies the source of the carry-in signal to the CLB, in this case, the CIN pin itself.

Consider another example:

INCDEC-G-F1

This mnemonic describes a CLB in which the G function generator performs an increment/decrement function. The carry-in to this CLB is sourced by the F1 pin.

All available carry mode configuration mnemonics are listed in the next section, the “Carry Logic Configurations” section.

To determine which carry mode primitive corresponds to which mnemonic, see the “Carry Modes” table.

Carry Logic Configurations

This section lists and describes all the available carry mode configuration mnemonics. The following information is given for each mnemonic:

- The name of the mode mnemonic.
- A brief description of the CLB function.
- The COUT0 and COUT equations performed by the carry logic.
- Default equations for the F and G function generators.
- Default assignments for the F4, G2, and G3 inputs.

The default F and G functions and default F4, G2, and G3 inputs are based on the generic CLB function described. You can change these defaults as required, allowing for features such as parallel enable or synchronous reset. However, if these defaults are changed, the CLB may no longer function as the mnemonic describes.

The COUT0 and COUT equations are absolutely determined by the carry mode configuration mnemonic. The only way to change these carry logic outputs is by selecting a different mnemonic.

ADD-F-CI

The ADD-F-CI configuration performs a 1-bit addition of A+B in the F function generator, with the A and B inputs on the F1 and F2 pins. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. This configuration can be used as the MSB of an adder, with the G function generator accessing the carry-out signal or calculating a two's-complement overflow.

$$F=(F1@F2)@F4$$

$$COUT0=(F1*F2) + CIN*(F1+F2)$$

$$G=$$

$$COUT=COUT0$$

$$F4=CIN$$

$$G2=G2I (COUT0 \text{ for overflow, } OFL=G2@G3, \text{ or for carry-out, } CO=G2)$$

$$G3=G3I (CIN \text{ for overflow, } OFL=G2@G3)$$

ADD-FG-CI

The ADD-FG-CI configuration performs a 2-bit addition of A+B in both the F and G function generators, with the lower-order A and B inputs on the F1 and F2 pins, and the higher-order A and B inputs on the G1 and G4 pins. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of an adder.

$$F=(F1@F2)@F4$$

$$COUT0=(F1*F2) + CIN*(F1+F2)$$

$$G=(G4@G1)@G2$$

$$COUT=(G4*G1) + COUT0*(G4+G1)$$

F4=CIN
 G2=COU0
 G3=G3I

ADD-G-F1

The ADD-G-F1 configuration performs a 1-bit addition of A+B in the G function generator, with the A and B inputs on the G1 and G4 pins. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COU0 pin. This configuration comprises the LSB of an adder, where the carry-in signal is routed to F1. The F function generator is not used.

F=
 COU0=F1
 $G=(G4@G1)@G2$
 $COU=(G4*G1) + COU0*(G4+G1)$
 F4=F4I
 G2=COU0
 G3=G3I

ADD-G-CI

The ADD-G-CI configuration performs a 1-bit addition of A+B in the G function generator, with the A and B inputs on the G1 and G4 pins. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COU0 pin. This configuration is for the middle bit of an adder, where the F function generator is reserved for another purpose.

F=
 COU0=CIN
 $G=(G4@G1)@G2$
 $COU=(G4*G1) + COU0*(G4+G1)$
 F4=F4I
 G2=COU0
 G3=G3I

ADD-G-F3-

The ADD-G-F3- configuration performs a 1-bit addition of A+B in the G function generator, with the A and B inputs on the G1 and G4 pins. The carry signal enters on the F3 pin, is inverted by the F carry logic, propagates through the G carry logic, and exits on the COU0 pin. This configuration comprises the LSB of an adder, where the inverted carry-in signal is routed to F3. The F function generator is not used.

F=
 COU0=~F3
 $G=(G4@G1)@G2$
 $COU=(G4*G1) + COU0*(G4+G1)$

$$F4=F4I$$

$$G2=COU0$$

$$G3=G3I$$
SUB-F-CI

The SUB-F-CI configuration performs a 1-bit twos-complement subtraction of A-B in the F function generator, with the A input on F1 and the B input on F2. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. This configuration can be used as the MSB of a subtracter, with the G function generator accessing the carry-out signal or calculating a twos-complement overflow.

$$F=(F1@F2)@\sim F4=\sim(F1@F2@F4)$$

$$COU0=(F1*\sim F2) + CIN*(F1+\sim F2)$$

$$G=$$

$$COUT=COU0$$

$$F4=CIN$$

$$G2=G2I \text{ (COU0 for overflow, OFL=G2@G3, or for carry-out, CO=G2)}$$

$$G3=G3I \text{ (CIN for overflow, OFL=G2@G3)}$$
SUB-FG-CI

The SUB-FG-CI configuration performs a 2-bit twos-complement subtraction of A-B in both the F and G function generators. For the lower bit, the A input is on F1 and the B input is on F2. For the upper bit, the A input is on G4 and the B input is on G1. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of a subtracter.

$$F=(F1@F2)@\sim F4=\sim(F1@F2@F4)$$

$$COU0=(F1*\sim F2) + CIN*(F1+\sim F2)$$

$$G=(G4@G1)@\sim G2=\sim(G4@G1@G2)$$

$$COUT=(G4*\sim G1) + COU0*(G4+\sim G1)$$

$$F4=CIN$$

$$G2=COU0$$

$$G3=G3I$$
SUB-G-1

The SUB-G-1 configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry-in is tied High (no borrow). The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of a subtracter with no carry-in. The F function generator is not used.

$$F=$$

$$COU0=1$$

$$G=(G4@G1)$$

$$\text{COUT}=(\text{G4}+\sim\text{G1})$$

$$\text{F4}=\text{F4I}$$

$$\text{G2}=\text{G2I}$$

$$\text{G3}=\text{G3I}$$

SUB-G-CI

The SUB-G-CI configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. This configuration is for the middle bit of a subtracter, where the F function generator is reserved for another purpose.

$$\text{F}=\text{F}$$

$$\text{COUT0}=\text{CIN}$$

$$\text{G}=(\text{G4}@\text{G1})@\sim\text{G2}=\sim(\text{G4}@\text{G1}@\text{G2})$$

$$\text{COUT}=(\text{G4}*\sim\text{G1}) + \text{COUT0}*(\text{G4}+\sim\text{G1})$$

$$\text{F4}=\text{F4I}$$

$$\text{G2}=\text{COUT0}$$

$$\text{G3}=\text{G3I}$$

SUB-G-F1

The SUB-G-F1 configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a subtracter, where the carry-in signal is routed to F1. The F function generator is not used.

$$\text{F}=\text{F}$$

$$\text{COUT0}=\text{F1}$$

$$\text{G}=(\text{G4}@\text{G1})@\sim\text{G2}=\sim(\text{G4}@\text{G1}@\text{G2})$$

$$\text{COUT}=(\text{G4}*\sim\text{G1}) + \text{COUT0}*(\text{G4}+\sim\text{G1})$$

$$\text{F4}=\text{F4I}$$

$$\text{G2}=\text{COUT0}$$

$$\text{G3}=\text{G3I}$$

SUB-G-F3-

The SUB-G-F3- configuration performs a 1-bit twos-complement subtraction of A-B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F3 pin, is inverted by the F carry logic, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a subtracter, where the inverted carry-in signal is routed to F3. The F function generator is not used.

$$\text{F}=\text{F}$$

$$\text{COUT0}=\sim\text{F3}$$

$$G=(G4@G1)@\sim G2=\sim(G4@G1@G2)$$

$$COUT=(G4*\sim G1) + COUT0*(G4+\sim G1)$$

$$F4=F4I$$

$$G2=COUT0$$

$$G3=G3I$$

ADDSUB-F-CI

The ADDSUB-F-CI configuration performs a 1-bit twos-complement add/subtract of A+B in the F function generator, with the A input on F1 and the B input on F2. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The F3 input indicates add (F3=1) or subtract (F3=0). This configuration can be used as the MSB of an adder/subtractor, with the G function generator accessing the carry-out signal or calculating a twos-complement overflow.

$$F=(F1@F2)@F4@\sim F3=\sim(F1@F2@F4@F3)$$

$$COUT0=F3*((F1*F2) + CIN*(F1+F2)) + \sim F3*((F1*\sim F2) + CIN*(F1+\sim F2))$$

$$G=$$

$$COUT=COUT0$$

$$F4=CIN$$

$$G2=G2I (COUT0 \text{ for overflow, } OFL=G2@G3, \text{ or for carry-out, } CO=G2)$$

$$G3=G3I (CIN \text{ for overflow, } OFL=G2@G3)$$

ADDSUB-FG-CI

The ADDSUB-FG-CI configuration performs a 2-bit twos-complement add/subtract of A+B in both the F and G function generators. For the lower bit, the A input is on F1 and the B input is on F2. For the upper bit, the A input is on G4 and the B input is on G1. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate add (F3=G3=1) or subtract (F3=G3=0): the add/subtract control signal must be routed to both the F3 and G3 pins. This configuration comprises the middle bits of an adder/subtractor.

$$F=(F1@F2)@F4@\sim F3=\sim(F1@F2@F4@F3)$$

$$COUT0=F3*((F1*F2) + CIN*(F1+F2)) + \sim F3*((F1*\sim F2) + CIN*(F1+\sim F2))$$

$$G=(G4@G1)@G2@\sim G3=\sim(G4@G1@G2@G3)$$

$$COUT=F3*((G4*G1)+COUT0*(G4+G1))+\sim F3*((G4*\sim G1)+COUT0*(G4+\sim G1))$$

$$F4=CIN$$

$$G2=COUT0$$

$$G3=G3I$$

ADDSUB-G-CI

The ADDSUB-G-CI configuration performs a 1-bit twos-complement add/subtract of A+B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate add (F3=G3=1) or subtract (F3=G3=0): the add/subtract control signal must be routed to both the F3 and G3 pins. This

configuration is for the middle bit of an adder/subtractor, where the F function generator is reserved for another purpose.

$$F =$$

$$\text{COUT0} = \text{CIN}$$

$$G = (G4 @ G1) @ G2 @ \sim G3 = \sim (G4 @ G1 @ G2 @ G3)$$

$$\text{COUT} = F3 * ((G4 * G1) + \text{COUT0} * (G4 + G1)) + \sim F3 * ((G4 * \sim G1) + \text{COUT0} * (G4 + \sim G1))$$

$$F4 = F4I$$

$$G2 = \text{COUT0}$$

$$G3 = G3I$$

ADDSUB-G-F1

The ADDSUB-G-F1 configuration performs a 1-bit twos-complement add/subtract of A+B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate add (F3=G3=1) or subtract (F3=G3=0): the add/subtract control signal must be routed to both the F3 and G3 pins. This configuration comprises the LSB of an adder/subtractor, where the carry-in signal is routed to F1. The F function generator is not used.

$$F =$$

$$\text{COUT0} = F1$$

$$G = (G4 @ G1) @ G2 @ \sim G3 = \sim (G4 @ G1 @ G2 @ G3)$$

$$\text{COUT} = F3 * ((G4 * G1) + \text{COUT0} * (G4 + G1)) + \sim F3 * ((G4 * \sim G1) + \text{COUT0} * (G4 + \sim G1))$$

$$F4 = F4I$$

$$G2 = \text{COUT0}$$

$$G3 = G3I$$

ADDSUB-G-F3-

The ADDSUB-G-F3- configuration performs a 1-bit twos-complement add/subtract of A+B in the G function generator, with the A input on G4 and the B input on G1. The carry signal enters on the F3 pin, is inverted by the F carry logic, propagates through the G carry logic, and exits on the COUT pin. Because the F3 input also indicates add (F3=1) or subtract (F3=0), the carry-in is always null (0 for add, 1 for subtract). This configuration comprises the LSB of an adder/subtractor with no carry-in. The F function generator is not used.

$$F =$$

$$\text{COUT0} = \sim F3$$

$$G = (G4 @ G1)$$

$$\text{COUT} = F3 * G4 * G1 + \sim F3 (G4 + \sim G1)$$

$$F4 = F4I$$

$$G2 = \text{COUT0}$$

$$G3 = G3I$$

INC-F-CI

The INC-F-CI configuration performs a 1-bit increment in the F function generator, with the input on the F1 pin. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The G function generator can be used to output the terminal count of a counter.

$$F=(F1@F4)$$

$$COUT0=CIN*F1$$

$$G=$$

$$COUT=COUT0$$

$$F4=CIN$$

$$G2=G2I \text{ (COUT0 for terminal count, TC=G2)}$$

$$G3=G3I$$
INC-FG-1

The INC-FG-1 configuration performs a 2-bit increment in both the F and G function generator, with the lower-order A input on the F1 pin and the higher-order A input on the G4 pin. The carry-in is tied High. The carry signal propagates through the F and G carry logic and exits on the COUT pin. This configuration comprises the two least significant bits of an incrementer that is always enabled.

$$F=\sim(F1)$$

$$COUT0=F1$$

$$G=G2@G4$$

$$COUT=COUT0*G4$$

$$F4=F4I \text{ or CIN}$$

$$G2=COUT0$$

$$G3=G3I \text{ or CIN}$$
INC-FG-CI

The INC-FG-CI configuration performs a 2-bit increment in both the F and G function generators, with the lower-order input on the F1 pin and the higher-order input on the G4 pin. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of an incrementer.

$$F=(F1@F4)$$

$$COUT0=CIN*F1$$

$$G=(G4@G2)$$

$$COUT=COUT0*G4$$

$$F4=CIN$$

$$G2=COUT0$$

$$G3=G3I$$

INC-G-1

The INC-G-1 configuration performs a 1-bit increment in the G function generator, with the input on the G4 pin. The carry-in is tied High. The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of an incrementer that is always enabled. The F function generator is not used. This configuration is identical to DEC-G-0, since the LSB of an incrementer is identical to the LSB of a decremter.

$$F=$$

$$COUT0=0$$

$$G=\sim(G4)$$

$$COUT=G4$$

$$F4=F4I$$

$$G2=G2I$$

$$G3=G3I$$
INC-G-F1

The INC-G-F1 configuration performs a 1-bit increment in the G function generator, with the input on the G4 pin. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an incrementer where F1 is an active-High enable. The F function generator is not used.

$$F=$$

$$COUT0=F1$$

$$G=(G4@G2)$$

$$COUT=COUT0*G4$$

$$F4=F4I$$

$$G2=COUT0$$

$$G3=G3I$$
INC-G-CI

The INC-G-CI configuration performs a 1-bit increment in the G function generator, with the input on the G4 pin. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. This configuration is for the middle bit of an incrementer where the F function generator is reserved for another purpose.

$$F=$$

$$COUT0=CIN$$

$$G=(G4@G2)$$

$$COUT=COUT0*G4$$

$$F4=F4I$$

$$G2=COUT0$$

$$G3=G3I$$

INC-G-F3-

The INC-G-F3- configuration performs a 1-bit increment in the G function generator, with the input on the G4 pin. The carry signal enters on the F3 pin, is inverted in the F carry logic, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an incrementer where F3 is an active-Low enable. The F function generator is not used.

$$F=$$

$$COUT0=\sim F3$$

$$G=(G4@G2)$$

$$COUT=COUT0*G4=\sim F3*G4$$

$$F4=F4I$$

$$G2=COUT0$$

$$G3=G3I$$

DEC-F-CI

The DEC-F-CI configuration performs a 1-bit decrement in the F function generator, with the input on the F1 pin. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The G function generator can be used to output the terminal count of a counter.

$$F=\sim(F1@F4)$$

$$COUT0=F1+CIN*\sim F1$$

$$G=$$

$$COUT=COUT0$$

$$F4=CIN$$

$$G2=G2I \text{ (COUT0 for terminal count, TC=G2)}$$

$$G3=G3I$$

DEC-FG-0

The DEC-FG-0 configuration performs a 2-bit decrement in both the F and G function generator, with the lower-order input on the F1 pin and the higher order input on the G4 pin. The carry-in is tied Low. The carry signal propagates through the F and G carry logic and exits on the COUT pin. This configuration comprises the two least significant bits of a decremter that is always enabled.

$$F=\sim(F1)$$

$$COUT0=F1$$

$$G=\sim(G4@G2)$$

$$COUT=COUT=(COUT0*\sim G4) + G4$$

$$F4=F4I$$

$$G2=COUT0$$

$$G3=G3I$$

DEC-FG-CI

The DEC-FG-CI configuration performs a 2-bit decrement in both the F and G function generators, with the lower-order input on the F1 pin and the higher-order input on the G4 pin. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. This configuration comprises the middle bits of a decremter.

$$F=\sim(F1@F4)$$

$$COUT0=F1+CIN*\sim F1$$

$$G=\sim(G4@G2)$$

$$COUT=G4+COUT0*\sim G4$$

$$F4=CIN$$

$$G2=COUT0$$

$$G3=G3I$$

DEC-G-0

The DEC-G-0 configuration performs a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry-in is tied High (no borrow). The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of a decremter that is always enabled. The F function generator is not used. This configuration is identical to INC-G-1, since the LSB of an incremter is identical to the LSB of a decremter.

$$F=$$

$$COUT0=0$$

$$G=\sim(G4)$$

$$COUT=G4$$

$$F4=F4I$$

$$G2=G2I$$

$$G3=G3I$$

DEC-G-CI

The DEC-G-CI configuration performs a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. This configuration is for the middle bit of a decremter, where the F function generator is reserved for another purpose.

$$F=$$

$$COUT0=CIN$$

$$G=\sim(G4@G2)$$

$$COUT=G4+COUT0*\sim G4$$

$$F4=F4I$$

$$G2=COUT0$$

$$G3=G3I$$

DEC-G-F1

The DEC-G-F1 configuration performs a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a decremter where F1 is an active-Low enable. The F function generator is not used.

$$F=$$

$$COUT0=F1$$

$$G=\sim(G4@G2)$$

$$COUT=COUT0 + G4$$

$$F4=F4I$$

$$G2=COUT0$$

$$G3=G3I$$

DEC-G-F3-

The DEC-G-F3- configuration performs a 1-bit decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the F3 pin, is inverted in the F carry logic, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of a decremter, where F3 is an active-High enable. The F function generator is not used.

$$F=$$

$$COUT0=\sim F3$$

$$G=\sim(G4@G2)$$

$$COUT=COUT0 + G4$$

$$F4=F4I$$

$$G2=COUT0$$

$$G3=G3I$$

INCDEC-F-CI

The INCDEC-F-CI configuration performs a 1-bit increment/decrement in the F function generator, with the input on the F1 pin. The carry signal enters on the CIN pin, propagates through the F carry logic, and exits on the COUT pin. The F3 input indicates increment (F3=1) or decrement (F3=0). The G function generator can be used to output the terminal count of a counter.

$$F=(F1@F4)@\sim F3$$

$$COUT0=\sim F3*(F1+ CIN) + F3*F1*CIN$$

$$G=$$

$$COUT=COUT0$$

$$F4=CIN$$

$$G2=G2I \text{ (COUT0 for terminal count, } TC=G2)$$

$$G3=G3I$$

INCDEC-FG-1

The INCDEC-FG-1 configuration performs a 2-bit increment/decrement in both the F and G function generator, with the lower-order input on the F1 pin and the higher-order input on the G4 pin. The F3 and G3 inputs indicate increment ($F3=G3=1$) or decrement ($F3=G3=0$): the increment/decrement control signal must be routed to both the F3 and G3 pins. The carry-in is always active (High in increment mode and Low in decrement mode). The carry signal propagates through the F and G carry logic and exits on the COUT pin. This configuration comprises the two least significant bits of an incrementer/decrementer that is always enabled.

$$F=\sim(F1)$$

$$COUT0=F1$$

$$G=(G2@G4)@\sim G3$$

$$COUT=COUT=\sim F3*((COUT0*\sim G4)+G4) + F3*(G4*COUT0)$$

$$F4=F4I$$

$$G2=COUT0$$

$$G3=G3I$$

INCDEC-FG-CI

The INCDEC-FG-CI configuration performs a 2-bit increment/decrement in both the F and G function generators, with the lower-order input on the F1 pin and the higher-order input on the G4 pin. The carry signal enters on the CIN pin, propagates through the F and G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate increment ($F3=G3=1$) or decrement ($F3=G3=0$): the increment/decrement control signal must be routed to both the F3 and G3 pins. This configuration comprises the middle bits of an incrementer/decrementer.

$$F=(F1@F4)@\sim F3$$

$$COUT0=\sim F3*(F1 + CIN) + F3*F1*CIN$$

$$G=(G4@G2)@\sim G3$$

$$COUT=\sim F3*(G4 + COUT0) + F3*G4*COUT0$$

$$F4=CIN$$

$$G2=COUT0$$

$$G3=G3I$$

INCDEC-G-0

The INCDEC-G-0 configuration performs a 1-bit increment/decrement in the G function generator, with the input on the G4 pin. The carry-in is tied High. The carry signal propagates through the G carry logic and exits on the COUT pin. This configuration comprises the LSB of an incrementer/decrementer that is always enabled. The F function generator is not used. F3 is not required for increment/decrement control, since the LSB of an incrementer is identical to the LSB of a decrementer; this configuration is identical to INC-G-1 and DEC-G-0.

$F=$
 $COUT0=0$
 $G=\sim(G4)$
 $COUT=G4$
 $F4=F4I$
 $G2=G2I$
 $G3=G3I$

INCDEC-G-CI

The INCDEC-G-CI configuration performs a 1-bit increment/decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the CIN pin, propagates through the G carry logic, and exits on the COUT pin. The F3 and G3 inputs indicate increment ($F3=G3=1$) or decrement ($F3=G3=0$): the increment/decrement control signal must be routed to both the F3 and G3 pins. This configuration is for the middle bit of an incrementer/decrementer, where the F function generator is reserved for another purpose, although the F3 pin is used by the carry logic.

$F=$
 $COUT0=CIN$
 $G=(G4@G2)@\sim G3$
 $COUT=\sim F3*(G4+ COUT0) + F3*G4*COUT0$
 $F4=F4I$
 $G2=COUT0$
 $G3=G3I$

INCDEC-G-F1

The INCDEC-G-F1 configuration performs a 1-bit increment/decrement in the G function generator, with the input on the G4 pin. The carry signal enters on the F1 pin, propagates through the G carry logic, and exits on the COUT pin. This configuration comprises the LSB of an incrementer/decrementer where the carry-in signal is routed to F1. The carry-in is active-High for an increment operation and active-Low for a decrement operation. The F function generator is not used. The F3 and G3 inputs indicate increment ($F3=G3=1$) or decrement ($F3=G3=0$): the increment/decrement control signal must be routed to both the F3 and G3 pins.

$F=$
 $COUT0=F1$
 $G=(G4@G2)@\sim G3$
 $COUT=F3*(G4*COUT0) + \sim F3*(G4+COUT0)$
 $F4=F4I$
 $G2=COUT0$
 $G3=G3I$

FORCE-0

The FORCE-0 configuration forces the carry-out signal on the COUT pin to be 0.

$$\text{COUT0}=0$$

$$\text{COUT}=0$$
FORCE-1

The FORCE-1 configuration forces the carry-out signal on the COUT pin to be 1.

$$\text{COUT0}=1$$

$$\text{COUT}=1$$
FORCE-CI

The FORCE-CI configuration forces the signal on the CIN pin to pass through to the COUT pin.

$$\text{COUT0}=\text{CIN}$$

$$\text{COUT}=\text{COUT0}=\text{CIN}$$
FORCE-F1

The FORCE-F1 configuration forces the signal on the F1 pin to pass through to the COUT pin.

$$\text{COUT0}=\text{F1}$$

$$\text{COUT}=\text{COUT0}=\text{F1}$$
FORCE-F3-

The FORCE-F3- configuration forces the signal on the F3 pin to pass inverted to the COUT pin.

$$\text{COUT0}=\sim\text{F3}$$

$$\text{COUT}=\text{COUT0}=\sim\text{F3}$$
FORCE-G4

The FORCE-G4 configuration forces the signal on the G4 pin to pass through to the COUT pin (XC4000X only).

$$\text{COUT0}=0$$

$$\text{COUT}=\text{G4}$$
EXAMINE-CI

The EXAMINE-CI configuration allows the carry signal on the CIN pin to be used in the F or G function generators. This configuration forces the signal on the CIN pin to pass through to the COUT pin and is equivalent to the FORCE-CI configuration. EXAMINE-CI is provided for CLBs in which the carry logic is unused but the CIN signal is required.

$$\text{COUT0}=\text{CIN}$$

$$\text{COUT}=\text{COUT0}=\text{CIN}$$

Carry Logic in XC5200 FPGAs

The XC5200 CLB contains a dedicated carry logic feature. This enhances the performance of arithmetic functions such as adders, subtractors, counters, comparators, and so forth. A carry multiplexer (CY_MUX) represents the dedicated 2:1 multiplexer in each logic cell. The multiplexer performs a 1-bit high speed carry propagate per logic cell (four bits per CLB).

In addition to providing a high speed carry propagate function, each CY_MUX can be connected to the CY_MUX in the adjacent logic cell to provide cascaded decode logic. The “XC5200 Carry Logic” figure illustrates how the four-input function generators can be configured to take advantage of the four cascaded CY_MUXes.

Note: AND and OR cascading are specific cases of a generic decode.

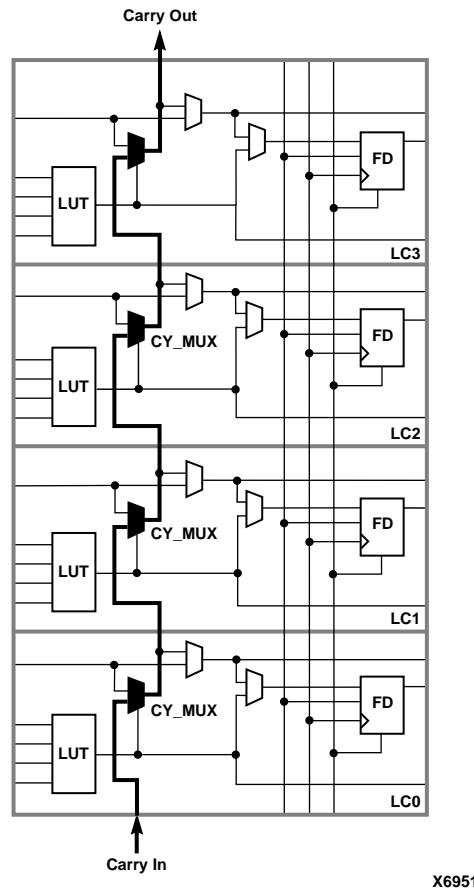
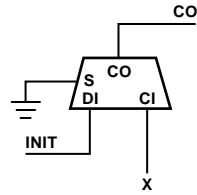


Figure 0-20 XC5200 Carry Logic

XC5200 Carry Logic Library Support

The design entry library contains one carry logic primitive and one carry logic macro. The carry multiplexer primitive (CY_MUX) represents the dedicated 2:1 multiplexer that performs the high speed carry propagate function. The carry initialize (CY_INIT) macro is used to initialize the carry chain for all arithmetic functions. The CY_INIT is implemented by forcing a zero onto the select line of the CY_MUX such that the DI pin of the CY_MUX is selected to drive the CO pin. See the “Carry Initialize Function XC5200” figure.



X6958

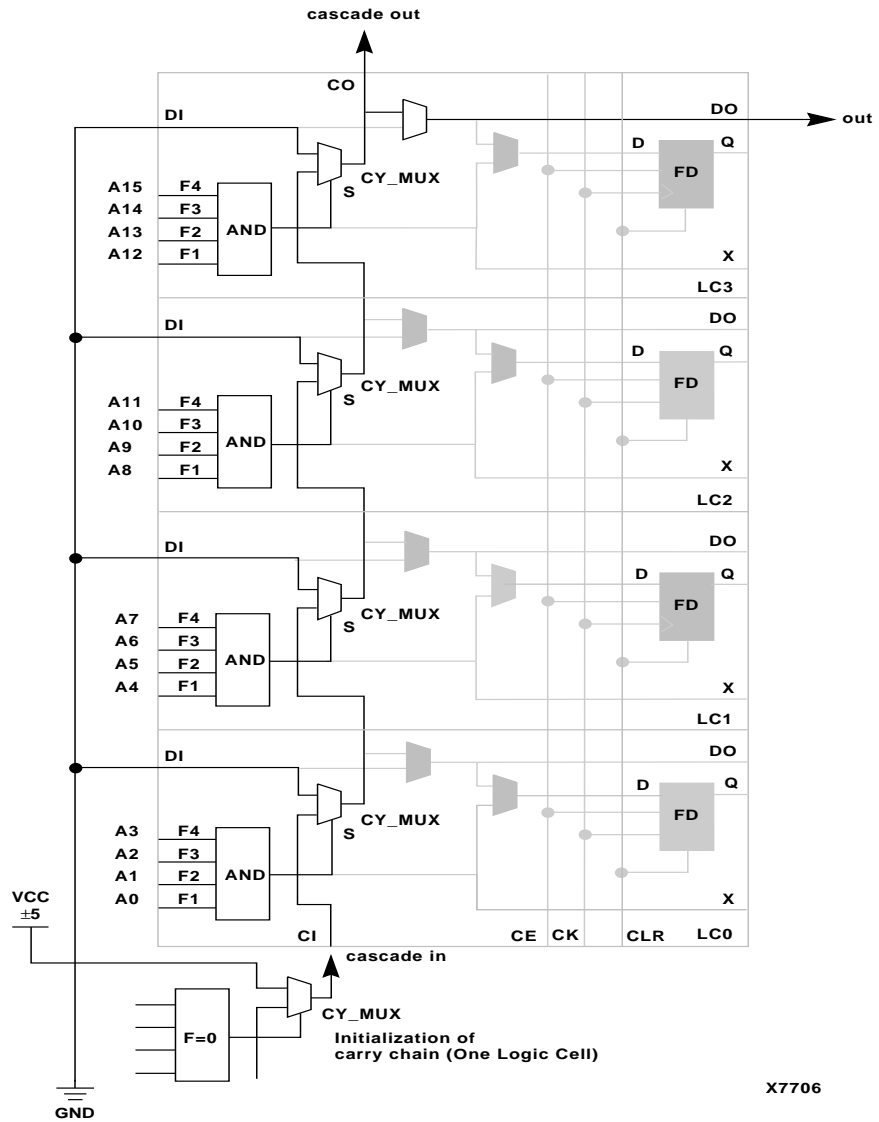
Figure 0-21 Carry Initialize Function XC5200

Note: The XC5200 library contains a set of RPMs designed to take advantage of the logic. Using the macros as they are or modifying them makes it much easier to take advantage of this feature.

Cascade Function

Each CY_MUX can be connected to the CY_MUX in the adjacent logic cell to provide cascadable decode logic. The “CY_MUX Used for Decoder Cascade Logic XC5200” figure illustrates how the 4-input function generators can be configured to take advantage of these four cascaded CY_MUXes.

Note: AND and OR cascading are specific cases of a general decode. In AND cascading, all bits are decoded equal to logic one. In OR cascading, all bits are decoded equal to logic zero. The flexibility of the LUT achieves this result.



X7706

Figure 0-22 CY_MUX Used for Decoder Cascade Logic XC5200

Index

A

ADD-F-CI, 142, 144
ADD-FG-CI, 142, 143, 144
ADD-G-CI, 142, 145
ADD-G-F1, 142, 145
ADD-G-F3-, 142, 145
ADDSUB-F-C1, 148
ADDSUB-F-CI, 142
ADDSUB-FG-CI, 142, 148
ADDSUB-G-CI, 142, 148
ADDSUB-G-F1, 142, 149
ADDSUB-G-F3, 149
ADDSUB-G-F3-, 142
ALLCLOCKNETS keyword in PCF
 used with FREQUENCY, 129, 138
 used with MAXDELAY, 131
 used with MAXSKEW, 132
 used with PERIOD, 134, 138
ALLPATHS keyword in PCF
 used with MAXDELAY, 131
attributes, 21

B

BASE attribute, 27, 34, 36, 38, 42
BEL, 133, 134, 136, 137
BLKNM constraint, 24, 26, 27, 34, 36, 40
block, 26
block name, 26, 40, 49
 in PCF, 132
Boolean operators
 XC3000, 47

BUFFCLK, 92
BUFG component, 92
BUFG constraint, 27, 34, 36, 41
BUFGE, 92
BUFGLS, 92
BUFGP, 92
BUFGS, 92
BUFT, 40, 49, 50, 53, 70, 82
 constraint examples, 56, 85
 creating wired-AND instance, 43
 identification of BUFT locations
 XC3000, 85
 XC4000, 85
 XC5200, 86
 with RLOC constraint, 71, 99
 with RLOC_ORIGIN constraint, 71, 114
bus pad symbols, 55

C

carry logic, 22, 139, 140
 carry mode, 141, 143
 handling in NDGDBUILD and the mapper, 143
 in XC5200 CLB, 54
 LOC and RLOC constraints, 142
 primitives, 40, 49, 133, 141
 symbols in relationally placed macros, 139
carry mode
 configuration mnemonics, 143

- names and symbols, 142
 - primitive symbols, 141
 - case sensitivity
 - in Mentor, 23
 - in UCF and NCF, 24, 85
 - CIN pin, 143
 - see also individual carry mode configuration mnemonics
 - CLBMAP
 - attaching mapping constraints to, 59
 - mapping constraint examples, 94
 - CLBs
 - base configuration modes, 38
 - block definition, 26
 - dedicated carry logic, 140, 143
 - flip-flop constraint examples, 91
 - mapping gates into function generators, 85
 - mapping with BLKNM attribute, 40
 - placement constraint examples, 24, 87
 - ROM and RAM constraints, 97
 - setting logic equations for function generators, 46
 - use with RLOC constraint, 100
 - XC3000 configuration options, 42
 - XC5200 physical site locations, 54
 - XC5200 slice extensions, 70
 - CLK, 41, 42
 - COLLAPSE constraint, 27, 34, 36, 41
 - colon
 - in location attributes, 23, 57
 - in TSidentifier syntax, 33, 79, 82
 - with wildcards on RLOCs, 114
 - COMPGRP physical constraint, 34, 128, 130
 - COMPONENT keyword, 129
 - CONFIG constraint (XC3000), 27, 34, 36, 42
 - CONFIG primitive
 - attaching PART to, 65
 - attaching PROHIBIT to, 67
 - examples
 - prohibiting CLBs, 89
 - prohibiting flip-flops, 92
 - prohibiting IOBs, 93
 - prohibiting memory placement, 98
 - configuration files see NCF, UCF, or PCF file
 - constraints
 - applicable files, 34
 - applied to macros and nets, 36
 - file name used by NGDBUILD, 26
 - logical, 22, 24
 - syntax list, 27
 - UCF/NCF file syntax, 25
 - physical, 22
 - PCF syntax, 128
 - used by Mentor Graphics, 23
 - user generated, 24
 - COUT pin, 143, 144
 - see also individual carry mode configuration mnemonics
 - COUT0 pin, 143, 144
 - CY4 symbols, 99, 142, 143
- D**
- DEC-F-CI, 142, 152
 - DEC-FG-0, 142, 152
 - DEC-FG-CI, 142, 153
 - DEC-G-0, 142, 153
 - DEC-G-CI, 142, 153
 - DEC-G-F1, 142, 154
 - DEC-G-F3-, 142, 154
 - DECODE constraint, 27, 34, 36, 43, 70, 99
 - on WAND gate, 89
 - decode logic, 89
 - DECODE macro, 89
 - decoders
 - cannot use with RLOC constraint, 99
 - dedicated carry logic, 140, 158
 - design hierarchy, 49, 50, 82, 101, 102, 106, 107, 108, 110, 113, 116

-
- DFF, 99
 - DIVIDE1_BY constraint, 28, 34, 36, 44
 - DIVIDE2_BY constraint, 28, 34, 36, 44
 - DOUBLE constraint, 28, 34, 36, 44
 - double quotes, 24, 25, 128
 - DRIVE constraint, 28, 34, 36, 45
 - DROP_SPEC constraint, 28, 34, 36, 46

 - E**
 - edge decoder
 - creating wired-AND instance, 43
 - edge decoders, 54, 55, 68
 - constraint examples, 89
 - edge designations, 90
 - edge indicators, 55
 - EDIF netlist
 - see netlist
 - EDIF2NGD, 23
 - EQUATE_F attribute, 28, 34, 36, 46
 - EQUATE_G attribute, 28, 34, 36, 46
 - EXAMINE-CI, 142, 143, 157

 - F**
 - F mode, 38, 42
 - F5MAP, 39, 49, 50, 53, 58, 59, 70, 82, 99
 - placement constraints, 85
 - FAST constraint, 25, 28, 34, 36, 47, 128
 - FFS group name, 75
 - FG mode, 38, 42
 - FGM mode, 38, 42
 - FILE attribute, 28, 34, 36, 48
 - flip-flop, 39, 66, 76, 94, 133
 - and PERIOD constraint, 125
 - and TNM attribute, 120
 - and XC5200, 52, 54, 64, 129, 132
 - CLB, 38, 49, 56, 71
 - CLB examples, 87, 88, 91
 - FFS group name, 75
 - FFX extension and NGDBUILD, 108
 - IOB, 61
 - macros with RLOCs, 113
 - mapper processing with RLOC constraints, 99
 - RLOC constraint examples, 100
 - timing groups in PCF, 136
 - triggering by rising/falling clock edges, 121
 - use with LOC constraint, 87, 91
 - use with RLOC constraint, 100, 104, 108
 - with RLOC sets, 101
 - X, 89
 - XC4000 location extensions, 55, 68
 - Xilinx-supplied macros, 113
 - Y, 89
 - FMAP, 39, 48, 50, 53, 58, 70, 82
 - grouping function generators into one CLB, 40
 - mapping constraint examples, 93, 95
 - placement constraints, 85
 - relationally placed macros, 139
 - schematic example, 96
 - using with RLOC constraints, 99
 - FORCE-0, 142, 157
 - FORCE-1, 142, 157
 - FORCE-CI, 142, 157
 - FORCE-F1, 142, 157
 - FORCE-F3-, 142, 157
 - FORCE-G4, 142
 - FREQUENCY physical constraint, 34, 129, 138
 - function generators, 143
 - base configuration modes, 38
 - carry logic, 140
 - carry mode configuration syntax, 143
 - grouping with BLKMN attribute, 40
 - grouping with HBLKMN attribute, 49
 - logic equations for F and G, 46, 144
 - mapping constraints, 95
 - mapping into F, 95
 - mapping into H, 95
 - placement constraints, 85

see also individual carry mode
configuration mnemonics
specifying with LOC constraint, 87

G

global buffer placement, 55, 56, 68, 92
corner designations, 92

H

H_SET constraint, 102, 117
and mapper processing, 109
compared to HU_SET, 50
implicit, 104
sets, 107
 generated name, 104
 modification of, 108
 separating elements, 110
 summary, 105
 with RLOC_ORIGIN, 117
half edge indicators, 55, 68
HBLKNM constraint, 26, 28, 34, 36, 49
hierarchical design *see* design hierarchy
HMAP, 39, 49, 50, 53, 58, 70, 82
 mapping constraints, 93, 95
 placement constraints, 85
 relationally placed macros, 139
 RLOC constraints, 99
 schematic example, 96
 use with BLKNM attribute, 40
 use with MAP attribute, 59
horizontal longline, 44
HU_SET constraint, 29, 34, 36, 50
 and removal from H_SET, 102
 purpose, 102, 110
 RLOC set hierarchy, 103, 110, 111, 112, 115
 summary, 106
 with RLOC_ORIGIN constraint, 117

I

I/O

 buffers, 64, 93
 constraint examples, 92
 edge indicators, 55
 element, 39, 48
 half edge indicators, 55
 pads, 75, 93
 pins, 54
 placement, 56
 primitives
 use with RLOC constraint, 99
 primitives and NODELAY attribute, 61
 primitives removed, 73
 registers, 93
IBUF, 41, 77
INCDEC-F-CI, 143, 154
INCDEC-FG-1, 143, 155
INCDEC-FG-CI, 143, 155
INCDEC-G-0, 143, 155
INCDEC-G-CI, 143, 156
INCDEC-G-F1, 143, 156
INC-F-CI, 142, 150
INC-FG-1, 142, 150
INC-FG-CI, 142, 150
INC-G-1, 142, 151
INC-G-CI, 142, 151
INC-G-F1, 142, 151
INC-G-F3-, 142, 152
INIT constraint, 26, 29, 34, 36, 51
INREG constraint, 29, 34, 36, 52
 in PCF, 129
instance, 26
instance name, 26
IOBs, 39, 42, 45, 47, 48, 52, 53, 61, 65, 93, 129, 132, 136
 base configuration, 38
 block definition, 26
 constraint examples, 93
 I/O constraints, 92
 increasing output speed with FAST attribute, 47
 location designations, 55, 58, 67

removing default delay, 61
 use with global buffers, 92
 XC3000 configuration options, 43

IOPAD, 54

IPAD, 54

K

KEEP constraint, 29, 34, 36, 53

L

latches, 52, 64, 129, 132, 136

 delay, 61

 time group, 76

LATCHES keyword, 32, 75, 77, 121, 136

LCA block names, 40, 49

LOC constraint, 29, 34, 36

 compared to RLOC constraints, 99

 CPLD syntax, 57

 dot extensions on ranges, 55

 examples

 BUFT placement, 85, 86

 CLB placement, 87, 88

 CLBMAP placement, 94

 decode logic placement, 89

 flip-flop placement, 91

 FMAP and HMAP placement, 95

 global buffer placement, 92

 I/O placement, 93

 IOB placement, 93

 memory placement, 97

 FPGA syntax for

 multiple locations, 56

 range of locations, 57

 single location, 56

 in PCF *see* LOCATE, 130

 on carry logic symbol, 142

 on CPLDs, 54

 on FPGAs, 54

propagation down design hierarchy,
 113

schematic syntax, 24

SOFT locations, 23, 57

LOCATE physical constraint, 34, 130

LOCK physical constraint, 34, 130

LogiBlox, 97

longline, 44, 45, 87, 134

LSB, 145, 146, 147, 149, 151, 152, 153, 154,

155, 156

M

macro

 attaching constraints to, 36

 attaching RLOCs to, 113

MAP

 processing of

 CLBMAP symbol, 94

MAP constraint, 29, 34, 36, 59

 examples

 on CLBMAP symbols, 94

 on FMAP symbols, 95

 on HMAP symbols, 95

mapper, 25, 45, 47

 addition function, 108, 109

 carry mode trimmng, 143

 hierarchy processing, 111, 117

 IOB constraints, 93

MAXDELAY constraint, 29, 34, 36, 59, 127

 in PCF, 131

 used with TSidentifier, 33, 80, 123

MAXSKEW constraint, 30, 34, 36, 60, 126

 in PCF, 131

MEDDELAY constraint, 30, 34, 36, 61

Mentor, 23

MSB, 144, 146, 148

N

NCD file

 set constraints removed, 115

NCF file, 22, 24, 34

- applicable constraints, 34
- case sensitivity, 24, 85
- rules, 24
- net
 - attaching constraints to, 36, 41
 - controlling delays, 127
 - controlling skew, 126
 - prioritizing in PCF, 134
 - S(ave) attribute, 73
- Net Constraints File *see* NCF file, 22
- net flag attribute, 31, 37, 73
- netlist, 85, 89, 91, 92, 93, 97
 - constraints overridden by UCF, 115
 - conversion of Mentor and Xilinx property names, 23
 - instance name, 26
 - Netlist Constraint File (NCF), 24
 - relationship of physical constraints to, 128
- NGDBUILD, 47, 49, 60, 61
 - carry mode trimming, 143
 - constraints file used, 26
 - hierarchy for HU_SET names, 50
- NODELAY constraint, 30, 34, 36, 61
- NOMERGE property, 53
- NOREDUCE constraint, 30, 34, 36, 62
- O**
- OE, 41
- OFFSET constraint, 30, 34, 36, 63
 - in PCF, 132
- OPAD, 54
- OPT_EFFORT constraint, 30, 35, 36, 63
- OPTIMIZE constraint, 30, 35, 36, 64
- OUTREG constraint, 30, 35, 36, 65
 - in PCF, 132
- P**
- PAD symbols, 54, 55, 76, 93
- PAR, 23, 45, 65
 - CLB constraints, 89
 - enabling for XC5200 fast routing, 52
 - FMAP mapping, 97
 - HMAP mapping, 97
 - placement constraints
 - schematic syntax, 23
 - PART constraint, 30, 35, 36, 65
 - PATH physical constraint, 35, 133
 - used with MAXDELAY, 131
 - with TIG, 137
 - PCF file, 22, 34, 128
 - PENALIZE TILDE physical constraint, 35, 133
 - PERIOD constraint, 31, 35, 36, 66, 125, 126
 - in PCF, 133, 137
 - used with TSidentifier, 33, 80, 81
 - physical constraints, 22, 128
 - Physical Constraints File *see* PCF file, 22
 - physical design, 22
 - applicable attributes, 34
 - physical location types, 55, 67
 - pin grid arrays, 93
 - PIN physical constraint, 35, 134
 - used with PATH, 133
 - used with TIG, 137
 - with TIMEGRP, 136
 - PLC, 59
 - PRIORITIZE physical constraint, 35, 134
 - PROHIBIT constraint, 31, 35, 37, 57, 67
 - examples
 - flip-flops in CLBs, 92
 - I/O in IOBs, 93
 - logic in CLBs, 89
 - memory in CLBs, 98
 - in PCF, 135
 - properties, 23
 - PUC, 59
 - pull-up resistors, 44, 45
 - PULLUP symbols, 44
 - PUO, 59, 95
 - PWR_MODE constraint, 31, 35, 37, 69

R

R(reset) a register, 51
RAM, 40, 49, 50, 51, 53, 70, 82, 120
 constraint examples, 55, 97
 initialization, 51
 location extensions not supported, 68
 timing groups, 76
 with RLOCs, 99
RAMD, 40, 49, 50, 53, 70, 82, 99
RAMS, 40, 49, 50, 53, 70, 82, 99, 136
 constraint examples, 55
 location extensions not supported, 68
register initialization, 51
relationally placed macros, 139
reserved words, 25, 75, 77, 121, 128
RLOC constraint, 24, 31, 35, 37, 70
 affects of mapper on, 108, 109, 116
 carry logic, 142
 general syntax guidelines, 99
 grouping logic elements, 99
 hierarchy sets (H_SET), 102
 LOC propagation, 113
 on carry logic symbols, 142
 set linkage, 106
 set modification, 108
 set modifiers, 112
 set types summary, 104
 sets, 101, 104
 symbols that accept RLOC, 99
 turn off/on for specific elements, 115
 turn on/off, 83
 use with Xilinx macros, 113
 user defined groups (U_SET), 102
 with HU_SET, 103, 110
 XC5200 slice placement, 70
RLOC constraint
 in relationally placed macros, 139
RLOC_ORIGIN constraint, 35, 37
 affects of mapper on, 115
 modifying H_SET, 102, 105, 107, 117
 modifying HU_SET, 106, 117
 modifying U_SET, 105

 priority in netlist vs UCF file, 115
 purpose, 71, 114
 syntax, 24, 25, 31, 71, 117
 use to change RLOC to LOC, 114
 with sets that include BUFT symbols, 114
RLOC_RANGE constraint, 31, 37
 affects of mapper on, 114
 applicability, 35
 modifying H_SET, 102
 modifying H_SET, 105, 107
 modifying HU_SET, 106
 modifying U_SET, 105
 priority in netlist vs UCF file, 115
 purpose, 72, 114
 syntax, 24, 25, 72, 114
ROM, 40, 49, 50, 53, 70, 82
 constraint examples, 55, 97
 initialization, 51
 location extensions not supported, 68
 with RLOCs, 99
RPMs, 139
 see relationally placed macros

S

S(ave) attribute, 31, 35, 37, 73
S(et) a register, 51
schematic
 FMAP and HMAP example, 96
 instance, 26
 placing attributes on symbols, 23
semicolon
 in UCF and NCF, 24
separator characters, 24, 33, 79, 82
 with wildcards, 114
SET_ARRIVAL, 63
SET_INPUT_DELAY, 63
SET_OUTPUT_DELAY, 63
SITEGRP physical constraint, 35, 130, 135
slice extension
 XC5200 CLBs, 70
SLOW constraint, 32, 35, 37, 73

SOFT location, 23, 57
soft macros, 54, 87, 139
 LOC applied to, 93
special function access symbols, 61
SR, 41
SUB-F-CI, 142, 146
SUB-FG-CI, 142, 146
SUB-G-1, 142, 146
SUB-G-CI, 143, 147
SUB-G-F1, 143, 147
SUB-G-F3-, 143, 147
symbols

 attributes on schematics, 23
 TIMEGRP, 120
 TIMESPEC, 122
synchronous reset, 144
syntax
 constraint summary list, 27
 conventions, 27
 PCF file, 128
 schematic, 23
 UCF/NCF files, 24
 wildcards, 26

T

TBUF
 block definition, 26
TCK, 61
TDI, 61
TIG constraint, 32, 33, 35, 37, 74, 81, 124
 in PCF, 137
time group attributes, 32, 35, 37, 75, 76
TIMEGRP keyword, 75
TIMEGRP physical constraint, 136
 with FREQUENCY, 138
 with FREQUENCY in PCF, 129
 with MAXDELAY, 131
 with MAXSKEW, 132
 with PATH, 133
 with PERIOD, 134
TIMEGRP primitive, 121
 constraints on, 75, 76, 120, 121, 122

TIMESPEC primitive
 use to define timing requirements, 120, 122
timing, 46
timing names, 77
timing specification, 74, 76, 77, 78, 79
 applying to a design, 120
TMS, 61
TNM constraint, 32, 35, 37, 67, 75, 78, 79, 80, 81, 120, 121, 122, 125, 126
TPSYNC constraint, 32, 35, 37, 77
TPTHRU constraint, 32, 35, 79, 80, 81, 123
TS *see* TSidentifier
TSidentifier constraint, 35, 37
 adding timing specifications to UCF, 122
 defining priority for timing specifications, 124
 ignoring, 124
 in PCF, 137
 with ALLCLOCKNETS, 129, 134, 138
 with FREQUENCY, 129, 138
 with MAXDELAY, 131
 with PERIOD, 133, 137
 with TIG, 137
 purpose, 79
 syntax, 33, 80
 used with MAXDELAY, 80
 used with PERIOD, 80, 81, 125, 126
 with DROP_SPEC, 28, 46
 with PERIOD, 31, 66
 with TIG, 32, 74
TSidentifier name
 considerations for Mentor Graphics, 23

U

U_SET constraint, 33, 35, 37, 117
 applying USE_RLOC to, 116
 purpose, 82, 102
 summary, 105

- use with USE_RLOC constraint, 116
- with RLOC_ORIGIN, 117
- UCF file, 22, 24, 34
 - applicable constraints, 34
 - case sensitivity, 24, 85
 - CLB constraints, 87
 - constraints on CLBMAP, 94
 - edge decoder constraints, 89
 - flip-flop constraints, 91
 - FMAP constraints, 95
 - global buffer constraints, 92
 - HMAP constraints, 95
 - I/O constraints, 92
 - IOB constraints, 93
 - prohibit constraints, 93
 - RAM constraints, 97
 - ROM constraints, 97
 - rules, 24
 - timing constraints, 120
- UPAD, 54
- USE_RLOC constraint, 33, 35, 37
 - purpose, 83, 115
 - to turn RLOCs on/off, 115
 - using with U_SET, 116
- User Constraints File *see* UCF file, 22

V

VHDL, 21

W

- WAND, 43, 54, 70, 89, 90, 99
- wide-edge decoders, 43
- wildcards, 72, 86, 87, 88, 89, 98, 114, 142
 - for CLB rows/columns, 91
- WIREAND constraint, 33, 35, 37, 84

