

# 制約ガイド

UG625 (v11.4) 2009 年 12 月 2 日

## ザイリンクス商標および著作権情報



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

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

© Copyright 2002–2009 Xilinx Inc. All Rights Reserved. XILINX, the Xilinx logo, the Brand Window and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners. The PowerPC name and logo are registered trademarks of IBM Corp., and used under license. All other trademarks are the property of their respective owners.

本資料は英語版 (v.11.4) を翻訳したもので、内容に相違が生じる場合には原文を優先します。  
資料によっては英語版の更新に対応していないものがあります。  
日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

# 目次

ザイリンクス商標および著作権情報 .....	2
1 : このマニュアルについて .....	13
マニュアルの内容 .....	13
このリリースの新機能 .....	13
その他のリソース .....	14
表記規則 .....	14
書体 .....	14
オンライン マニュアル .....	15
2 : アーキテクチャ サポート .....	17
制約が使用されるアーキテクチャ .....	17
3 : 制約のタイプ .....	19
属性および制約 .....	19
属性 .....	19
インプリメンテーション制約 .....	20
CPLD フィッタ .....	20
タイミング用グループ制約 .....	20
定義済みグループの使用 .....	20
定義済みグループの例 .....	21
BRAMS_PORTA および BRAMS_PORTB の例 .....	21
論理制約 .....	23
物理制約 .....	23
マップ制約 .....	23
配置制約 .....	24
相対ロケーション制約 (RLOC 制約) .....	25
配置制約 .....	25
配線制約 .....	25
合成制約 .....	25
タイミング制約 .....	26
XSTタイミング制約 .....	26
コマンド ライン オプション .....	26
制約ファイル .....	26
UCF タイミング制約 .....	27
FROM-TO 制約 .....	27
OFFSET IN .....	27
OFFSET OUT .....	27
TIG .....	27
TIMEGRP .....	27
TNM .....	27
TNM_NET .....	28
タイミング モデル .....	28
優先順位 .....	28
タイミング制約およびグループ制約 .....	28
コンフィギュレーション制約 .....	29
4 : ザイリンクス制約の入力 .....	31
制約の入力方法 .....	31
制約の入力表 .....	32
回路図デザイン .....	35
VHDL .....	35
Verilog .....	36
Verilog の制限 .....	36
Verilog のメタ コメント .....	36
UCF .....	36
UCF フロー .....	37
手動でのタイミング制約の入力 .....	37
UCF ファイルが複数ある場合の制約の競合 .....	37

UCF および NCF の構文 .....	37
UCF および NCF の一般規則 .....	38
制約の競合 .....	38
構文 .....	38
TIMEGRP および TIMESPEC 属性の指定 .....	38
予約語の使用 .....	39
ワイルドカード .....	39
階層の指定 .....	40
複数制約の入力 .....	40
ファイル名 .....	40
インスタンスおよびブロック .....	41
PCF .....	41
NCF .....	42
Constraints Editor .....	42
入力/出力 .....	42
Constraints Editor の起動 .....	43
Project Navigator からの起動 .....	43
スタンドアロンとして起動 .....	43
コマンド ラインからの起動 .....	43
ファイルを読み込まずに起動 .....	43
NGD ファイルを読み込んで起動 .....	43
NGD および UCF を読み込んで起動 .....	43
バックグラウンドとして起動 .....	44
Project Navigator .....	44
PlanAhead .....	44
配置制約の割り当て .....	44
I/O ピン コンフィギュレーションの定義 .....	44
フロアプランおよび配置制約 .....	46
PACE .....	46
未完成のデザインにおけるピンの事前割り当て .....	46
FPGA Editor .....	47
固定されたネットとコンポーネント .....	48
制約の相互関係 .....	49
XCF .....	49
制約の優先順位 .....	49
ファイルの優先順位 .....	49
タイミング制約の優先順位 .....	49
OFFSET の優先度 .....	50
MAXSKEW および MAXDELAY の優先度 .....	50
優先順位の例外 .....	50
制約集合の相互関係 .....	50
5 : タイミング制約の設定方法 .....	53
基本的な制約設定方法 .....	53
入力タイミング制約 .....	53
概要 .....	54
システム同期入力 .....	54
ソース同期入力 .....	55
Register-to-Register タイミング制約 .....	57
はじめに .....	57
関連する DCM/PLL/MMCM ドメイン (自動) .....	58
関連するクロック ドメイン (手動) .....	58
非同期クロック ドメイン .....	59
出力タイミング制約 .....	60
はじめに .....	60
システム同期出力 .....	61
ソース同期出力 .....	62
例外のタイミング制約 .....	64
概要 .....	64
False パス .....	64

複数サイクル パス.....	65
6 : ザイリンクス制約 .....	67
制約に関する情報 .....	67
ザイリンクス制約のリスト (アルファベット順) .....	68
AREA_GROUP .....	70
アーキテクチャ サポート .....	70
適用可能エレメント .....	70
適用ルール .....	71
構文 .....	71
タイミング グループによる定義 .....	74
エリア グループからの定義 .....	75
ASYNC_REG .....	75
アーキテクチャ サポート .....	75
適用可能エレメント .....	75
適用ルール .....	76
BEL .....	76
アーキテクチャ サポート .....	76
適用可能エレメント .....	77
適用ルール .....	77
BLKNM .....	78
アーキテクチャ サポート .....	78
適用可能エレメント .....	79
適用ルール .....	79
BUFG .....	80
アーキテクチャ サポート .....	80
適用可能エレメント .....	80
適用ルール .....	80
CLOCK_DEDICATED_ROUTE .....	82
アーキテクチャ サポート .....	82
CLOCK_DEDICATED_ROUTE 適用可能エレメント .....	82
適用ルール .....	82
COLLAPSE .....	83
アーキテクチャ サポート .....	83
適用可能エレメント .....	83
適用ルール .....	83
COMPGRP .....	84
アーキテクチャ サポート .....	84
適用可能エレメント .....	84
CONFIG_MODE .....	84
アーキテクチャ サポート .....	85
適用可能エレメント .....	85
適用ルール .....	85
COOL_CLK (CoolCLOCK) .....	86
アーキテクチャ サポート .....	86
適用可能エレメント .....	86
適用ルール .....	86
DATA_GATE .....	87
アーキテクチャ サポート .....	87
適用可能エレメント .....	87
適用ルール .....	87
DEFAULT .....	88
アーキテクチャ サポート .....	88
適用可能エレメント .....	88
適用ルール .....	88
DCI_CASCADE .....	90
アーキテクチャ サポート .....	91
適用可能エレメント .....	91
適用ルール .....	91
DCI_VALUE .....	91

アーキテクチャ サポート .....	92
適用可能エレメント .....	92
適用ルール .....	92
DIRECTED_ROUTING .....	92
アーキテクチャ サポート .....	92
適用可能エレメント .....	92
適用ルール .....	92
DISABLE .....	93
アーキテクチャ サポート .....	93
適用可能エレメント .....	94
適用ルール .....	94
DRIVE .....	95
アーキテクチャ サポート .....	95
適用可能エレメント .....	95
適用ルール .....	95
DROP_SPEC .....	97
アーキテクチャ サポート .....	97
適用可能エレメント .....	97
適用ルール .....	97
ENABLE .....	97
アーキテクチャ サポート .....	97
適用可能エレメント .....	97
適用ルール .....	98
ENABLE_SUSPEND .....	99
アーキテクチャ サポート .....	99
適用可能エレメント .....	99
適用ルール .....	99
FAST .....	99
アーキテクチャ サポート .....	99
適用可能エレメント .....	99
適用ルール .....	100
FEEDBACK .....	101
アーキテクチャ サポート .....	101
適用可能エレメント .....	101
適用ルール .....	101
FILE .....	102
アーキテクチャ サポート .....	102
適用可能エレメント .....	102
適用ルール .....	102
FLOAT .....	103
アーキテクチャ サポート .....	103
適用可能エレメント .....	103
適用ルール .....	103
FROM-THRU-TO .....	104
アーキテクチャ サポート .....	104
適用可能エレメント .....	104
適用ルール .....	105
FROM-TO .....	106
アーキテクチャ サポート .....	106
適用可能エレメント .....	106
適用ルール .....	106
HBLKNM .....	107
アーキテクチャ サポート .....	107
適用可能エレメント .....	108
適用ルール .....	108
HLUTNM .....	109
アーキテクチャ サポート .....	109
適用可能エレメント .....	109
適用ルール .....	109

HU_SET.....	110
アーキテクチャ サポート .....	111
適用可能エレメント .....	111
適用ルール.....	111
IBUF_DELAY_VALUE.....	112
アーキテクチャ サポート .....	112
適用エレメント.....	112
IFD_DELAY_VALUE.....	113
アーキテクチャ サポート .....	114
適用エレメント.....	114
適用ルール.....	114
INREG.....	115
アーキテクチャ サポート .....	115
適用可能エレメント .....	115
適用ルール.....	115
IOB .....	115
アーキテクチャ サポート .....	115
適用可能エレメント .....	116
適用ルール.....	116
IOBDELAY .....	117
アーキテクチャ サポート .....	117
適用可能エレメント .....	117
適用ルール.....	117
IODELAY_GROUP .....	118
アーキテクチャ サポート .....	118
適用可能エレメント .....	119
適用ルール.....	119
構文.....	119
VHDL 構文 .....	119
Verilog 構文 .....	119
UCF 構文 .....	119
IOSTANDARD.....	119
アーキテクチャ サポート .....	120
適用可能エレメント .....	120
適用ルール.....	120
KEEP.....	122
アーキテクチャ サポート .....	122
適用可能エレメント .....	122
適用ルール.....	122
KEEP_HIERARCHY.....	123
アーキテクチャ サポート .....	124
適用可能エレメント .....	124
適用ルール.....	124
KEEPER.....	125
アーキテクチャ サポート .....	125
適用可能エレメント .....	126
適用ルール.....	126
LOC .....	127
FPGA デバイスの場合 .....	127
CPLD デバイスの場合 .....	128
LOC の優先順位 .....	129
アーキテクチャ サポート .....	129
適用可能エレメント .....	129
適用ルール.....	129
構文.....	129
DCM 制約の例 .....	133
フリップフロップ制約の例.....	133
I/O 制約の例 .....	134
IOB 制約の例.....	135

マップ制約の例 (FMAP).....	135
乗算器制約の例 .....	136
ROM 制約の例 .....	136
ブロック RAM (RAMB) 制約の例 .....	137
スライス制約の例 .....	137
スライス禁止制約 .....	138
LOCATE .....	139
アーキテクチャ サポート .....	139
適用可能エレメント .....	139
適用ルール .....	139
LOCK_PINS .....	140
アーキテクチャ サポート .....	140
適用可能エレメント .....	140
適用ルール .....	140
LUTNM.....	141
アーキテクチャ サポート .....	141
適用可能エレメント .....	141
適用ルール .....	141
MAP .....	143
アーキテクチャ サポート .....	143
適用可能エレメント .....	143
適用ルール .....	143
MAX_FANOUT.....	144
アーキテクチャ サポート .....	145
適用可能エレメント .....	145
適用ルール .....	145
MAXDELAY.....	146
アーキテクチャ サポート .....	146
適用可能エレメント .....	146
適用ルール .....	146
MAXPT .....	148
アーキテクチャ サポート .....	148
適用可能エレメント .....	148
適用ルール .....	148
MAXSKEW .....	149
アーキテクチャ サポート .....	149
適用可能エレメント .....	150
適用ルール .....	150
MIODELAY_GROUP.....	151
アーキテクチャ サポート .....	151
適用可能エレメント .....	151
適用ルール .....	151
構文.....	151
NODELAY .....	151
アーキテクチャ サポート .....	152
適用可能エレメント .....	152
適用ルール .....	152
NOREDUCE.....	153
アーキテクチャ サポート .....	153
適用可能エレメント .....	153
適用ルール .....	153
OFFSET IN.....	154
アーキテクチャ サポート .....	154
適用可能エレメント .....	154
OFFSET OUT .....	159
アーキテクチャ サポート .....	159
適用可能エレメント .....	159
OPEN_DRAIN.....	162
アーキテクチャ サポート .....	162



適用可能エレメント .....	163
適用ルール .....	163
OPT_EFFORT .....	163
アーキテクチャ サポート .....	164
適用可能エレメント .....	164
適用ルール .....	164
OPTIMIZE .....	164
アーキテクチャ サポート .....	164
適用可能エレメント .....	164
適用ルール .....	164
PERIOD .....	165
アーキテクチャ サポート .....	166
適用可能エレメント .....	166
適用ルール .....	166
CLKDLL、DCM、PLL、MMCM での PERIOD 仕様 .....	172
PIN .....	174
アーキテクチャ サポート .....	174
適用可能エレメント .....	174
適用ルール .....	174
POST_CRC .....	174
アーキテクチャ サポート .....	175
適用可能エレメント .....	175
適用ルール .....	175
POST_CRC_ACTION .....	175
アーキテクチャ サポート .....	176
適用可能エレメント .....	176
適用ルール .....	176
POST_CRC_FREQ .....	176
アーキテクチャ サポート .....	176
適用可能エレメント .....	176
適用ルール .....	176
POST_CRC_SIGNAL .....	177
アーキテクチャ サポート .....	177
適用可能エレメント .....	177
適用ルール .....	177
PRIORITY .....	178
アーキテクチャ サポート .....	178
適用可能エレメント .....	178
適用ルール .....	178
PROHIBIT .....	178
アーキテクチャ サポート .....	180
適用可能エレメント .....	180
適用ルール .....	180
PULLDOWN .....	181
アーキテクチャ サポート .....	181
適用可能エレメント .....	181
適用ルール .....	182
PULLUP .....	183
アーキテクチャ サポート .....	183
適用可能エレメント .....	183
適用ルール .....	183
PWR_MODE .....	184
アーキテクチャ サポート .....	184
適用可能エレメント .....	184
適用ルール .....	184
REG .....	185
アーキテクチャ サポート .....	185
適用可能エレメント .....	186
適用ルール .....	186

RLOC .....	187
アーキテクチャ サポート .....	187
適用可能エレメント .....	188
適用ルール .....	188
構文 .....	188
集合の修正子 .....	189
集合のリンク .....	190
集合の編集 .....	191
ザイリンクス マクロでの RLOC の使用 .....	194
相対位置を指定するためのガイドライン .....	196
RLOC 集合 .....	198
U_SET .....	198
H_SET .....	198
HU_SET .....	199
RLOC 集合の要約 .....	200
RLOC_ORIGIN .....	201
アーキテクチャ サポート .....	201
適用可能エレメント .....	201
適用ルール .....	201
構文 .....	202
RLOC_RANGE .....	203
アーキテクチャ サポート .....	203
適用可能エレメント .....	203
適用ルール .....	203
構文 .....	203
SAVE NET FLAG .....	205
アーキテクチャ サポート .....	205
適用可能エレメント .....	205
適用ルール .....	205
SCHMITT_TRIGGER .....	206
アーキテクチャ サポート .....	206
適用可能エレメント .....	206
適用ルール .....	207
SLEW .....	207
アーキテクチャ サポート .....	207
適用可能エレメント .....	208
適用ルール .....	208
SLOW .....	209
アーキテクチャ サポート .....	209
適用可能エレメント .....	209
適用ルール .....	210
STEPPING .....	211
アーキテクチャ サポート .....	211
適用可能エレメント .....	211
適用ルール .....	211
SUSPEND .....	211
アーキテクチャ サポート .....	211
適用可能エレメント .....	211
適用ルール .....	211
SYSTEM_JITTER .....	213
アーキテクチャ サポート .....	213
適用可能エレメント .....	213
適用ルール .....	213
TEMPERATURE .....	214
アーキテクチャ サポート .....	214
適用可能エレメント .....	214
適用ルール .....	214
TIG (タイミング無視) .....	215
アーキテクチャ サポート .....	215

適用可能エレメント .....	215
適用ルール .....	215
TIMEGRP .....	217
アーキテクチャ サポート .....	218
適用可能エレメント .....	218
適用ルール .....	218
TIMESPEC .....	222
アーキテクチャ サポート .....	222
適用可能エレメント .....	222
適用ルール .....	222
構文 .....	222
TNM (タイミング名) .....	224
アーキテクチャ サポート .....	224
適用可能エレメント .....	224
適用ルール .....	225
TNM_NET .....	228
アーキテクチャ サポート .....	228
適用可能エレメント .....	228
適用ルール .....	228
適用ルール .....	229
TPSYNC .....	231
アーキテクチャ サポート .....	231
適用可能エレメント .....	231
適用ルール .....	231
TPTHRU .....	233
アーキテクチャ サポート .....	233
適用可能エレメント .....	233
適用ルール .....	233
TSidentifier .....	235
アーキテクチャ サポート .....	235
適用可能エレメント .....	236
適用ルール .....	236
U_SET .....	238
アーキテクチャ サポート .....	238
適用可能エレメント .....	239
適用ルール .....	239
USE_RLOC .....	240
アーキテクチャ サポート .....	240
適用可能エレメント .....	240
適用ルール .....	240
VCCAUX .....	243
アーキテクチャ サポート .....	243
適用可能エレメント .....	243
VOLTAGE .....	244
アーキテクチャ サポート .....	244
適用可能エレメント .....	244
適用ルール .....	244
VREF .....	245
アーキテクチャ サポート .....	245
適用可能エレメント .....	245
適用ルール .....	245
WIREAND .....	246
アーキテクチャ サポート .....	246
適用可能エレメント .....	246
適用ルール .....	246
XBLKNM .....	247
アーキテクチャ サポート .....	247
適用可能エレメント .....	247
適用ルール .....	247



## このマニュアルについて

---

本書では、ザイリンクスの FPGA および CPLD デバイスのデザインに適用可能な制約および属性について説明します。この章には、次のセクションが含まれています。

- ・ [マニュアルの内容](#)
- ・ [このリリースの新機能](#)
- ・ [その他のリソース](#)
- ・ [表記規則](#)

### マニュアルの内容

このマニュアルは、次の章から構成されています。

- ・ 第 1 章「[このマニュアルについて](#)」: このガイドの ISE® Design Suite に関する新情報およびその他の概要などを記述します。
- ・ 第 2 章「[制約タイプ](#)」: CPLD フィッタ、グループ制約、論理制約、物理制約、マップ制約、配置制約、配線制約、合成制約、タイミング制約など、さまざまな制約タイプについて説明します。
- ・ 第 3 章「[ザイリンクス制約の入力方法](#)」: さまざまな制約の入力方法と、入力に使用する ISE ソフトウェアの機能について説明します。
- ・ 第 4 章「[タイミング制約の設定](#)」: インプリメンテーション ツールを使用して FPGA デザインにタイミング制約を設定する方法について説明します。
- ・ 第 5 章「[ザイリンクス制約](#)」: FPGA および CPLD デバイスで利用できる制約について、そのアーキテクチャ サポート、適用可能エレメント、説明、適用ルール、合成コード例などを制約別に説明します。

### このリリースの新機能

このセクションでは、今回のリリースで変更された点について説明します。

- ・ 次の制約を新しく追加しました。
  - [DEFAULT](#)
  - [MAX\\_FANOUT](#)
  - [IODELAY\\_GROUP](#)
  - [MODELAY\\_GROUP](#)
- ・ 廃止されたアーキテクチャをすべて削除しました。詳細は、「[アーキテクチャ別制約](#)」を参照してください。
- ・ LOC 制約ターゲットのリストに ICAP を追加しました。
- ・ AREA\_GROUP 制約に CLOCK REGION の説明を追加しました。

- ・ SET コマンドを削除しました。
- ・ lat\_ce\_q に ENABLE および DISABLE 制約を追加しました。
- ・ TNM 制約は回路図デザインではサポートされなくなりました。
- ・ IOB 制約に FORCE 値を追加しました。
- ・ TEMPERATURE 制約を Spartan-3A DSP アーキテクチャがサポートされるようにアップデートしました。
- ・ PlanAhead™ ソフトウェアに関する情報を追加しました。
- ・ PACE は、CPLD でのみサポートされ、FPGA ではサポートされません。
- ・ USELOWSKEWLINES 制約を削除しました。

## その他のリソース

その他の資料については、次の Web サイトから参照してください。

<http://japan.xilinx.com/literature>

シリコン、ソフトウェア、IP に関する質問および解答をアンサー データベースで検索したり、テクニカル サポートのウェブ ケースを開くには、次のザイリンクス Web サイトにアクセスしてください。

<http://japan.xilinx.com/support>

## 表記規則

このマニュアルでは、次の表記規則を使用しています。各規則について、例を挙げて説明します。

## 書体

次の規則は、すべてのマニュアルで使用されています。

表記規則	使用箇所	例
Courier フォント	システムが表示するメッセージ、プロンプト、プログラム ファイルを表示します。	speed grade: - 100
<b>Courier</b> フォント (太字)	構文内で入力するコマンドを示します。	<b>ngdbuild</b> design_name
イタリック フォント	ユーザーが値を入力する必要のある構文内の変数に使用します。	<i>ngdbuild design_name</i>
二重/一重かぎカッコ『』、『』	『』はマニュアル名を、『』はセクション名を示します。	詳細については、『コマンドライン ツール ユーザー ガイド』の「PAR」を参照してください。
角カッコ [ ]	オプションの入力またはパラメータを示しますが、 <b>bus[7:0]</b> のようなバス仕様では必ず使用します。また、GUI 表記にも使用します。	ngdbuild [option_name] design_name [File] → [Open] をクリックします。
中カッコ { }	1 つ以上の項目を選択するためのリストを示します。	<b>lowpwr = {on off}</b>

表記規則	使用箇所	例
縦棒	選択するリストの項目を分離します。	<b>lowpwr</b> = {on off}
縦の省略記号	繰り返し項目が省略されていることを示します。	IOB #1: Name = QOUT IOB #2: Name = CLKIN . . .
横の省略記号. . .	繰り返し項目が省略されていることを示します。	<b>allow block</b> . . . <i>block_name</i> <i>loc1 loc2 ... locn</i> ;

## オンライン マニュアル

このマニュアルでは、次の規則が使用されています。

表記規則	使用箇所	例
青色の文字	マニュアル内の相互参照、その他の文書へのリンクを示します。	詳細については、「 <a href="#">その他のリソース</a> 」を参照してください。  詳細については、第 1 章「 <a href="#">タイトル フォーマット</a> 」を参照してください。  詳細は、『 <a href="#">Virtex-6 ハンドブック</a> 』の図 25 を参照してください。





## 第 2 章

# アーキテクチャ サポート

---

各アーキテクチャでサポートされる制約に関する情報については、「[アーキテクチャ別制約](#)」を参照してください。

### 制約が使用されるアーキテクチャ

このガイドでは、次のアーキテクチャについて説明します。

- ・ CoolRunner™ XPLA3、CoolRunner-II、XC9500、XC9500XL CPLD
- ・ Spartan®-3、Spartan-3A、Spartan-3E、Spartan-3L、Spartan-6、Virtex®-4、Virtex-5、Virtex-6



## 制約のタイプ

---

この章では、さまざまな制約のタイプについて説明します。この章は、次のセクションから構成されています。

- ・ 属性および制約
- ・ CPLD フィッタ
- ・ グループ制約
- ・ 論理制約
- ・ 物理制約
- ・ マップ制約
- ・ 配置制約
- ・ 配線制約
- ・ 合成制約
- ・ タイミング制約
- ・ コンフィギュレーション制約

### 属性および制約

「属性」と「制約」は、同義で使用されるときと、異なる意味で使用されるときがあります。また、構文で属性と指示子ที่ใช้される場合も、意味は類似していますが異なります。本書における「属性」と「制約」の定義を次に示します。

### 属性

属性は、通常インスタンス化されるコンポーネントのファンクションおよびインプリメンテーションに影響するデバイスアーキテクチャのプリミティブ コンポーネントに関連付けられるプロパティです。属性は、次の方法で設定します。

- ・ VHDL : ジェネリック マップ
- ・ Verilog : プリミティブ コンポーネントをインスタンス化し、defparams またはインライン パラメータで属性を渡します。

属性の例 :

- ・ LUT4 コンポーネント上の INIT プロパティ
- ・ DCM 上の CLKFX\_DIVIDE プロパティ

属性はすべて、『ライブラリ ガイド』のプリミティブ コンポーネントの記述に含まれています。

## インプリメンテーション制約

インプリメンテーション制約を使用すると、FPGA インプリメンテーション ツールでマップ、配置、タイミング、またはその他のガイドラインに従って FPGA デザインが処理されます。インプリメンテーション制約は、通常 UCF ファイルに含められますが、HDL コードまたは合成制約ファイルにも含めることができます。次に、インプリメンテーション制約の例を示します。

- ・ LOC (配置) 制約
- ・ PERIOD (タイミング) 制約

『制約ガイド』は、インプリメンテーション制約について記述しています。

## CPLD フィッタ

CPLD デバイスには、次の制約が適用されます。

BUFG (CPLD の場合)	COLLAPSE	COOL_CLK (CoolCLOCK)
DATA_GATE	FAST	INREG
IOSTANDARD	KEEP	KEEPER
LOC	MAXPT	NOREDUCE
OFFSET IN OFFSET OUT	OPEN_DRAIN	PERIOD
PROHIBIT	PULLUP	PWR_MODE
REG	SCHMITT_TRIGGER	SLOW
TIMEGRP	TIMESPEC	TNM
TSIdentifier	VREF	WIREAND

## タイミング用グループ制約

TIMESPEC 制約では、次のいずれかの方法でソースとターゲットをグループにまとめて、解析するパスの集合を指定します。

- ・ 対応するキーワード CPUS、DSPS、FFS、HSIOS、LATCHES、MULTS、PADS、RAMS、BRAMS\_PORTA、BRAMS\_PORTB を指定し、定義済みのグループを参照します。
- ・ **TNM** および **TNM\_NET** 制約をシンボルに設定し、定義済みグループ内に独自のグループを作成します。
- ・ **TIMEGRP** シンボルを使用した既存のグループを組み合わせ、グループを作成します。
- ・ ネット名の文字列を指定して、グループを作成します。詳細は、「**TIMEGRP**」の「文字列の指定によるグループの作成」を参照してください。

## 定義済みグループの使用

定義済みのグループを使用すると、対応するキーワードでフリップフロップ、入力ラッチ、パッド、RAM のグループを参照できます。

## 定義済みグループのキーワード

キーワード	説明
CPUS	Virtex®-4 FX デバイスの PPC405
DSPS	DSP48 および DSP48 派生のエレメントすべて (Virtex-4、Virtex-5 および Spartan®-3A Extended デバイス)  Virtex-5 FXT デバイスの CPUS
FFS	CLB および IOB のエッジで動作するフリップフロップ、およびシフトレジスタ LUT (すべてのデバイスにシフトレジスタ LUT あり)
HSIOS	GT11 (Virtex-4)、GTP_DUAL (Virtex-5)、および GTX_DUAL (Virtex-5 FXT)
LATCHES	CLB および IOB のレベル センシティブ ラッチすべて
MULTS	同期および非同期乗算器
PADS	I/O パッド すべて (通常、最上位の HDL ポートから推論される)
RAMS	<ul style="list-style-type: none"> <li>- シングルおよびデュアル ポート CLB LUT RAM すべて (デュアル ポートの両方のポートを含む)</li> <li>- シングルおよびデュアル ポート ブロック RAM すべて (デュアル ポートの両方のポートを含む)</li> <li>- FIFOS - すべての FIFO (First In, First Out) ブロック RAM メモリ</li> </ul>
BRAMS_PORTA	すべてのデュアル ポート ブロック RAM のポート A
BRAMS_PORTB	すべてのデュアル ポート ブロック RAM のポート B

FROM-TO 文を使用すると、定義済みグループ間のパスに対するタイミング仕様を定義できます。UCF に入力する TIMESPEC 制約の例は、次のとおりです。この方法を使用すると、簡単にデザインのタイミング仕様をデフォルト設定できます。

## 定義済みグループの例

次は、UCF 構文例です。

```
TIMESPEC "TS01"=FROM FFS TO FFS 30;
TIMESPEC "TS02"=FROM LATCHES TO LATCHES 25;
TIMESPEC "TS03"=FROM PADS TO RAMS 70;
TIMESPEC "TS04"=FROM FFS TO PADS 55;
TIMESPEC "TS01" = FROM BRAMS_PORTA TO BRAMS_PORTB(gork*);
```

BRAMS\_PORTA および BRAMS\_PORTB の場合、TS01 は A ポートからパターンが gork\* の信号を駆動する B ポートまでのパスを制御します。

## BRAMS\_PORTA および BRAMS\_PORTB の例

次に示すのは、BRAMS\_PORTA および BRAMS\_PORTB を使用した例です。

```
NET "X" TNM_NET = BRAMS_PORTA groupA;
```

TNM グループ groupA には、ネット X で駆動されるすべての A ポートが含まれます。ネット X が B ポートの入力にトレースされる場合、シングル ポート ブロック RAM のエレメントまたは SelectRAM のエレメントは groupA には含まれません。

```
NET "X" TNM_NET = BRAMS_PORTB( dob* ) groupB;
```

B ポートの出力のうち最低 1 つでもパターンが dob\* の信号を駆動している場合、TNM グループ groupB にネット X で駆動される B ポートが含まれます。

```
INST "Y" TNM = BRAMS_PORTB groupC;
```

TNM グループ groupC には、インスタンス Y 内のすべての B ポートが含まれます。インスタンス Y がデュアル ポート ブロック RAM のプリミティブである場合、groupC にはそのインスタンスの B ポートが含まれます。

```
INST "Y" TNM = BRAMS_PORTA( doa* ) groupD;
```

A ポートの出力のうち最低 1 つでもパターンが doa\* の信号を駆動している場合、TNM グループ groupD にはインスタンス Y 内の A ポートが含まれます。

```
TIMEGRP "groupE" = BRAMS_PORTA;
```

ユーザー グループ groupE には、デザイン内にあるすべてのデュアル ポート ブロック RAM エレメントの A ポートが含まれます。この設定は BRAMS\_PORTA( \* ) と同等です。

```
TIMEGRP "groupF" = BRAMS_PORTB( mem/dob* );
```

ユーザー グループ groupF には、デザイン内でパターンが mem/dob\* の信号を駆動するすべての B ポートが含まれます。

定義済みグループには名前修飾子を付けることもできます。名前修飾子は、定義済みのグループが使用される場所に表示され、参照されるエレメントの数に制限を付けます。構文は次のとおりです。

```
predefined group ( name_qualifier [ name_qualifier ] )
```

*name\_qualifier* は、指定したプリミティブを基に付けられるネットの完全階層名です。

名前修飾子には、次のようなワイルドカードを含めることができます。

- ・ アスタリスク (\*) : 任意の数の文字
- ・ クエスチョン マーク (?) : 任意の 1 文字

ワイルドカードを使用すると、次のようなことが可能です。

- ・ 複数のネットを指定
- ・ 完全階層名を省略

たとえば、group FFS (MACRO\_A/Q?) と指定すると、ネット Q0、Q1、Q2、Q3 を駆動するフリップフロップのみを選択できます。

## グループ制約

COMPGRP	PIN	TIMEGRP
TNM	TNM_NET	TPSYNC
TPTHRU		

## 論理制約

論理制約とは、マップまたはフィットが実行される前にデザインのエレメントに設定する制約です。論理制約を適用すると、予測されるワーストケース条件にデザインのパフォーマンスを適応させることができます。論理制約は、後でザイリックスのアーキテクチャを選択し、デザインを配置配線またはフィットする際に、物理制約に変換されます。

論理制約を設定するには、ネットリスト制約ファイル (NCF) またはユーザー制約ファイル (UCF) に書き込まれた入力デザインの属性を使用します。

論理制約には、次の 3 種類があります。

- ・ [配置制約](#)
- ・ [相対ロケーション制約 \(RLOC 制約\)](#)
- ・ [タイミング制約](#)

FPGA デバイスの場合、相対位置制約 (RLOC) は、ロジック エレメントをグループ化します。デザイン全体での最終的な配置に関係なく、グループ内でのエレメント同士の位置を相対的に定義できます。詳細は、「[配置制約](#)」を参照してください。

タイミング制約は、デザイン内にある特定のパスまたはネットの集合に、最大許容遅延またはスキューを指定できます。

## 物理制約

制約は、マップが実行された後の物理的なデザイン内のエレメントにも設定できます。この制約のことを「物理制約」と呼びます。これらは、Physical Constraints File (PCF) で定義されます。このファイルは、マップ中に作成されます。

ユーザー制約を設定する場合は、NCFまたは PCFではなく、UCFに入力することをお勧めします。

**メモ：** このセクションは FPGA デバイス ファミリのみに適用されます。

デザインをマップするときに、ネットリストと UCF ファイルにある論理制約は、物理制約 (特定のアーキテクチャに適用される制約) に変換されます。これらの制約は、マップで生成された物理制約ファイル (PCF) に書き込まれます。

このファイルには、2 つのセクションがあります。

- ・ [回路図セクション](#) ネットリストと UCF ファイルに記述された論理制約に基づく物理制約が含まれます。
- ・ [ユーザー セクション](#) 物理制約を追加するために使用します。

## マップ制約

マップ制約は、特定の動作を実行するように MAP に指示を与えます。マップ制約は次のとおりです。

AREA_GROUP	BEL	BLKNM
DCI_VALUE	DRIVE	FAST
HBLKNM	HLUTNM	HU_SET
IOB	IOBDELAY	IOSTANDARD
KEEP	KEEPER	LUTNM
MAP	NODELAY	OPTIMIZE
PULLDOWN	PULLUP	RLOC
RLOC_ORIGIN	RLOC_RANGE	SAVE NET FLAG
SLEW	U_SET	USE_RLOC
XBLKNM		

## 配置制約

ここでは、FPGA デザインの各種エレメントに設定する配置制約について説明します。エレメントには次のような種類があります。

- ・ フリップフロップ
- ・ ROM および RAM
- ・ BUFT
- ・ CLB 数
- ・ IOB
- ・ I/O
- ・ エッジ デコーダ
- ・ グローバル バッファ

AND ゲートや OR ゲートなどの論理ゲートは、制約が読み込まれる前に CLB ファンクション ジェネレータにマップされてしまうため、制約を設定できません。

次の制約は、ネットリストのシンボルの配置やマップを制御するために使用します。

- ・ BLKNM
- ・ HBLKNM
- ・ HLUTNM
- ・ LOC
- ・ LUTNM
- ・ PROHIBIT
- ・ RLOC
- ・ RLOC\_ORIGIN
- ・ RLOC\_RANGE
- ・ XBLKNM

ほとんどの制約は、HDL または UCF のいずれかで設定できます。制約ファイルでは、配置制約は 1 つまたは複数のシンボルに適用されます。デザイン内のシンボルには個々に名前が付けられ、名前は入力ファイルで定義されます。シンボルに制約を設定するには、この名前を制約文で使用します。



UCF ファイルおよび NCF ファイルでは、大文字と小文字が区別されます。識別子名（ネット名などのデザイン内のオブジェクトの名前）は、ソース デザインのネットリストに記述されている名前と大文字/小文字が正確に一致しなければなりません。ただし、LOC、PROHIBIT、RLOC、BLKNM などの制約キーワードは、大文字か小文字のいずれかを使用できます。大文字と小文字は混用できません。

## 相対ロケーション制約 (RLOC 制約)

RLOC 制約は、ロジック エLEMENTを独立した集合にグループ化します。ELEMENTの位置は、デザイン全体の最終的な配置に関係なく、同じ集合内のほかのELEMENTに相対的に定義できます。たとえば、1 列に並んだ 8 つのフリップフロップのグループに RLOC 制約を適用すると、マップは列の順番を保ったまま、フリップフロップのグループ全体を 1 つの単位として移動します。これに対して、絶対ロケーション (LOC) 制約は、ほかのデザイン ELEMENTに関係なく、FPGA チップ上の特定の位置にデザイン ELEMENTを指定します。

## 配置制約

配置制約は次のとおりです。

AREA_GROUP	BEL	LOC
LOCATE	OPT_EFFORT	PROHIBIT
RLOC	RLOC_ORIGIN	RLOC_RANGE
USE_RLOC		

## 配線制約

配線制約は、特定の動作を実行するように PAR に指示を与えます。配線制約は次のとおりです。

- ・ AREA\_GROUP
- ・ CONFIG\_MODE
- ・ LOCK\_PINS
- ・ OPT\_EFFORT

## 合成制約

合成制約を使用すると、合成ツールによる特定のデザインまたは HDL コードの一部に対する最適化手法を制御できます。合成制約は、VHDL または Verilog コードに組み込むか、または別の合成制約ファイルに含めます。次に、合成制約の例を示します。

- ・ USE\_DSP48 (XST ツール)
- ・ RAM\_STYLE (XST ツール)

合成制約の詳細は、次を参照してください。

- ・ 『XST ユーザー ガイド』は、XST 制約について記述しています。
- ・ その他の合成ツールの合成制約については、ベンダーのマニュアルを参照してください。詳細は、合成ツール ベンダーにお問い合わせください。

次の制約が合成制約です。

FROM-TO	IOB	KEEP
MAP	OFFSET IN OFFSET OUT	PERIOD
TIG	TNM	TNM_NET

## タイミング制約

ザイリンクスのソフトウェアを使用すると、デザインに厳密なタイミング要件を指定できます。タイミング要件は、グローバルまたはパスごとにタイミング制約を割り当てることで指定できます。よくあるタイミング要件の制約の定義方法については、第 5 章を参照してください。

タイミング制約を指定する方法としては、制約を UCF ファイルに入力するのが一般的です。タイミング制約は HDL デザインおよび回路図デザインのソース ファイルに入力することもできます。各制約の詳細については、後述します。

タイミング仕様を定義しデザインをマップすると、その要件を基に PAR によってデザインが配置配線されます。

タイミング仕様の結果を解析するには、コマンド ツールの TRACE (TRCE) または ISE® Timing Analyzer を使用します。

## XSTタイミング制約

XST では、合成制約およびタイミング制約を定義するのに XCF (XST 制約ファイル) の構文形式がサポートされています。以前の XST 構文は、ISE 7.1i 以降、使用できなくなりました。

XST でサポートされるタイミング制約は、次のいずれかで設定できます。

- ・ -glob\_opt コマンド ライン オプション
- ・ 制約ファイル

## コマンド ライン オプション

-glob\_opt コマンド ライン オプションは、Project Navigator の [Process Properties] ダイアログ ボックスで [Synthesis Options] をクリックして表示される [Global Optimization Goal] オプションと同じです。このオプションを使用すると、グローバル タイミング制約を設定できます。これらの制約に対して値を指定できない場合、XST で最適なパフォーマンスを得るための値が最適化されます。これらの制約は、制約ファイルで指定された制約で上書きされます。

## 制約ファイル

制約ファイルを使用すると、ネイティブの UCF 構文を使用してタイミング制約を設定できます。XCF 構文を使用すると、ワイルドカードや階層名を含む TNM\_NET、TIMEGRP、PERIOD、TIG、FROM-TO などの XST でサポートされる制約がサポートされます。

タイミング制約を設定するには、Project Navigator の [Process Properties] ダイアログ ボックスで [Synthesis Options] をクリックし、[Write Timing Constraints] オプションをオンにします。または、-write\_timing\_constraints オプションを使用した場合にのみ、NGC ファイルに記述されます。デフォルトでは NGC ファイルに書き込まれません。

タイミング制約の指定とは別に、CLOCK\_SIGNAL 制約はタイミング制約の処理に影響を与えます。クロック信号が通過する入力ピンが実際のクロック ピンの場合、CLOCK\_SIGNAL 制約を使用すると、クロック ピンを定義できます。詳細は、『XST ユーザー ガイド』を参照してください。

## UCF タイミング制約

**注意:** XCF ファイルでタイミング制約を指定する場合、階層の区切り記号にはアンダースコア ( `_` ) ではなく、スラッシュ ( `/` ) を使用してください。

XCF (XST Constraints File) でサポートされるタイミング制約は次のとおりです。

### FROM-TO 制約

FROM-TO は、2 つのグループ間のタイミング制約を設定する制約で、この場合のグループは、ユーザー定義のグループまたは定義済みのグループ (FFS、PADS、RAMS) です。詳細は、「[FROM-TO](#)」を参照してください。XCF 構文は次のとおりです。

```
TIMESPEC "TS name"=FROM group1 TO "group2" value ;
```

### OFFSET IN

OFFSET IN 制約は、FPGA への入力インターフェースのタイミング要件を指定するために使用します。この制約で指定できるのは、FPGA の外部パッドにおけるクロックとデータのタイミング関係です。OFFSET IN 制約を指定すると、制約の付いた同期エレメントすべてのセットアップ タイムとホールド タイムの要件が確認されます。次の図は、OFFSET IN 制約でカバーされるパスを示しています。詳細は、「[OFFSET IN](#)」を参照してください。

### OFFSET OUT

OFFSET OUT 制約は、FPGA からの出力インターフェースのタイミング要件を指定するために使用します。この制約で指定できるのは、FPGA の入力ピンのクロック エッジから FPGA の出力ピンでデータが有効になるまでの時間です。詳細は、「[OFFSET OUT](#)」を参照してください。

### TIG

[タイミング無視 \(TIG\)](#) 制約を設定すると、タイミング解析および最適化の際に、制約を設定したネットを通過するパスが無視されます。この制約は、その影響を受ける信号名に適用されます。

XCF 構文 :

```
NET "netname" TIG;
```

### TIMEGRP

[TIMEGRP](#) は、基本的なグループ制約です。TNM 識別子を使用したグループの作成に加え、ほかのグループを基にしてグループを定義できます。TIMEGRP 制約を使用すると、既存グループを組み合わせてグループを作成できます。

この制約は、制約ファイル (XCF または NCF) で設定できます。TIMEGRP を使用したグループの作成は、次のいずれかの方法で行います。

- ・ 複数のグループを 1 つのグループにまとめる
- ・ クロックの立ち上がり/立ち下がりによってフリップフロップのサブグループを指定する

### TNM

[TNM](#) はグループを構成するエレメントを指定する基本的なグループ制約で、作成したグループにタイミング仕様を設定できます。この制約を使用すると、特定の FFS、RAMS、LATCHES、PADS、BRAMS\_PORTA、BRAMS\_PORTB、CPUS、HSIOS、MULTS をグループのエレメントとして指定し、タイミング仕様の適用を簡略化できます。

この制約は、キーワード RISING および FALLING と一緒に使用することもできます。

XCF 構文 :

```
{NET|INST|PIN} "net_or_pin_or_inst_name" TNM=[predefined_group] identifier ;
```

## TNM\_NET

**TNM\_NET** は、入力パッド ネットに設定する場合を除き、ネットに設定した TNM と基本的に同等です。すべての FPGA デバイスの DLL/DCM/PLL/MMCM に、**PERIOD** 制約を使用した **TNM\_NET** を設定する場合は、特別なルールが適用されます。

詳細は、「**PERIOD**」の「**CLKDLL、DCM、PLL、および MMCM での PERIOD 指定**」を参照してください。

**TNM\_NET** は通常、特定ネットを指定するため HDL デザインで使用するプロパティです。**TNM\_NET** で指定されたダウンストリームの同期エレメントおよびパッドは、すべてグループと見なされます。詳細は、「**TNM**」を参照してください。

XCF 構文 :

```
NET "netname" TNM_NET=[predefined_group] identifier;
```

## タイミング モデル

タイミング解析用に XST で使用されるタイミング モデルは、ロジック遅延およびネット遅延の両方を考慮に入れます。これらの遅延は、XST に対して指定されたスピード グレードや選択されたアーキテクチャによって変化します。ロジック遅延のデータは、TRACE (配置配線後に Timing Analyzer で実行) でレポートされる遅延と類似しています。ネット遅延のモデルは、ファンアウト ロードを基に予測されます。

## 優先順位

制約は次の順序で処理されます。

- ・ 信号の特定制約
- ・ 最上位モジュールの特定制約
- ・ 最上位モジュールのグローバル制約

2 つの異なるドメインまたは信号に制約を設定した場合、優先順位は同じになります。たとえば、**PERIOD clk1** は **PERIOD clk2** と共に適用されます。

## タイミング制約およびグループ制約

タイミング制約および関連のグループ制約は次のとおりです。

ASYNC_REG	DISABLE	DROP_SPEC
ENABLE	FROM-THRU-TO	FROM-TO
MAXSKEW	OFFSET IN OFFSET OUT	PERIOD
PRIORITY	SYSTEM_JITTER	TEMPERATURE
TIG	TIMEGRP	TIMESPEC
TNM	TNM_NET	TPSYNC
TPTHRU	TSidentifier	VOLTAGE

## コンフィギュレーション制約

コンフィギュレーション制約は次のとおりです。

CONFIG_MODE	DCI_CASCADE	STEPPING
POST_CRC	POST_CRC_ACTION	POST_CRC_FREQ
POST_CRC_SIGNAL	VCCAUX	VREF



# ザイリンクス制約の入力

ここでは、さまざまなザイリンクス制約の入力方法と、入力に使用する ISE® ソフトウェアの機能について説明します。この章は、次のセクションから構成されています。

- ・ [制約の入力方法](#)
- ・ [制約の入力表](#)
- ・ [回路図デザイン](#)
- ・ [VHDL](#)
- ・ [Verilog](#)
- ・ [UCF](#)
- ・ [PCF](#)
- ・ [NCF](#)
- ・ [Constraints Editor](#)
- ・ [Project Navigator](#)
- ・ [PACE](#)
- ・ [未完成のデザインにおけるピンの事前割り当て](#)
- ・ [PlanAhead™](#)
- ・ [FPGA Editor](#)
- ・ [制約の優先順位](#)

## 制約の入力方法

次の表は、制約の入力に使用する ISE® ソフトウェアを示したものです。

### 制約の入力方法

ISE ツール	制約のタイプ	デバイス
Constraints Editor	タイミング	すべての CPLD および FPGA
PlanAhead™	IO 配置制約およびエリア グループ制約	すべての FPGA
PACE	I/O の配置	すべての CPLD
Schematic and Symbol Editors	I/O 配置制約およびその他の配置制約	すべての CPLD および FPGA

## 制約の入力表

次の表は、制約とその入力方法を示します。制約の詳細を参照するには、制約名をクリックしてください。

制約の入力表

制約	回路図	VHDL Verilog	NCF	UCF	Constraints Editor	PCF	XCF	PlanAhead™	PACE	FPGA Editor	Project Navigator
<a href="#">AREA_GROUP</a>	○		○	○	○			○			
<a href="#">ASYNC_REG</a>		○	○	○	○						
<a href="#">BEL</a>		○	○	○				○			
<a href="#">BLKNM</a>	○	○	○	○			○				
<a href="#">BUFG (CPLD)</a>	○	○	○	○			○				
<a href="#">CLOCK_DEDICATED_ROUTE</a>			○	○							
<a href="#">COLLAPSE</a>	○	○	○	○							
<a href="#">COMPGRP</a>						○					
<a href="#">CONFIG_MODE</a>				○							
<a href="#">COOL_CLK</a>	○	○	○	○							
<a href="#">DATA_GATE</a>	○	○	○	○							
<a href="#">DEFAULT</a>	○	○	○	○			○	○	○		
<a href="#">DCI_CASCADE</a>			○	○		○					
<a href="#">DCI_VALUE</a>			○	○							
<a href="#">DIRECTED_ROUTING</a>			○	○						○	
<a href="#">DISABLE</a>			○	○		○					
<a href="#">DRIVE</a>	○	○	○	○			○	○	○	○	
<a href="#">DROP_SPEC</a>	○		○	○		○					
<a href="#">ENABLE</a>			○	○		○					
<a href="#">ENABLE_SUSPEND</a>			○	○							
<a href="#">FAST</a>	○	○	○	○			○	○	○		
<a href="#">FEEDBACK</a>				○	○	○	○	○			
<a href="#">FILE</a>	○	○									
<a href="#">FLOAT</a>	○	○	○	○			○				
<a href="#">FROM-THRU-TO</a>			○	○	○	○		○			
<a href="#">FROM-TO</a>			○	○	○	○	○	○			
<a href="#">HBLKNM</a>	○	○	○	○							
<a href="#">HLUTNM</a>	○	○	○	○	○		○				
<a href="#">HU_SET</a>	○	○	○	○			○				



制約	回路図	VHDL Verilog	NCF	UCF	Constraints Editor	PCF	XCF	PlanAhead™	PACE	FPGA Editor	Project Navigator
IBUF_DELAY_VALUE	○	○	○	○							
IFD_DELAY_VALUE	○	○	○	○							
INREG	○			○							
IOB	○	○	○	○			○				○
IOBDELAY	○	○	○	○	○						
IODELAY_GROUP				○							
IOSTANDARD	○	○	○	○			○	○	○	○	
KEEP	○	○	○	○			○				
KEEPER	○	○	○	○	○		○			○	
KEEP_HIERARCHY	○	○	○	○			○				○
LOC	○	○	○	○		○	○	○	○		
LOCATE						○				○	
LOCK_PINS		○	○	○							
LUTNM	○	○	○	○							
MAP	○		○	○							
MAXDELAY	○	○	○	○	○	○				○	
MAX_FANOUT		○					○				○
MAXPT		○	○	○							
MAXSKEW	○	○	○	○	○	○				○	
NODELAY	○	○	○	○			○				
IODELAY_GROUP				○							
NOREDUCE	○	○	○	○			○				
OFFSET IN	○		○	○	○	○	○	○			
OFFSET OUT	○		○	○	○	○	○	○			
OPEN_DRAIN	○	○	○	○			○				
OPT_EFFORT	○		○	○							○
OPTIMIZE	○	○	○	○							○
PERIOD	○	○	○	○	○	○	○	○		○	
PIN				○							
POST_CRC				○		○					
POST_CRC_ACTION				○		○					
POST_CRC_FREQ				○		○					
POST_CRC_SIGNAL				○		○					
PRIORITY			○	○		○					

制約	回路図	VHDL Verilog	NCF	UCF	Constraints Editor	PCF	XCF	PlanAhead™	PACE	FPGA Editor	Project Navigator
PROHIBIT				○		○		○	○	○	
PULLDOWN	○	○	○	○			○	○	○	○	
PULLUP	○	○	○	○			○	○	○	○	
PWR_MODE	○	○	○	○			○				
REG	○	○	○	○			○				
RLOC	○	○	○	○			○	○			
RLOC_ORIGIN	○	○	○	○		○		○			
RLOC_RANGE	○	○	○	○		○	○				
SAVE NET FLAG	○	○	○	○			○				
SCHMITT_TRIGGER	○	○	○	○			○				
SLEW	○	○	○	○			○	○	○	○	
SLOW	○	○	○	○			○	○	○	○	
STEPPING				○							
SUSPEND	○	○	○	○					○		
SYSTEM_JITTER	○	○	○	○			○				
TEMPERATURE			○	○	○	○					
TIG	○		○	○	○	○	○	○			
TIMEGRP			○	○	○	○	○	○			
TIMESPEC			○	○	○		○	○			
TNM			○	○	○		○	○			
TNM_NET	○		○	○	○		○	○			
TPSYNC	○		○	○							
TPTHRU	○		○	○	○						
TSidentifier			○	○	○	○	○			○	
U_SET	○	○	○	○			○				
USE_RLOC	○	○	○	○			○	○			
VCCAUX			○	○							
VOLTAGE			○	○	○	○					
VREF	○		○	○							
WIREAND	○	○	○	○							
XBLKNM	○	○	○	○			○				

## 回路図デザイン

シンボルまたは回路図でザイリンクス制約を属性として設定するには、次のルールに従ってください。

- ・ ネットに適用する場合、属性としてネットに設定します。
- ・ インスタンスに適用する場合、属性としてインスタンスに設定します。
- ・ PART や PROHIBIT のようなグローバル制約は設定できない
- ・ TIMESPEC や TIMEGRP などに設定されるタイミング要件も追加できない
- ・ 属性名および属性値は、大文字のみまたは小文字のみで入力します。大文字と小文字は一緒に使用できません。

属性の作成、変更、表示については、Schematic Editor/Symbol Editor ヘルプで手順を確認してください。

このマニュアルには、回路図に入力できる制約の構文例が記載されています。たとえば、BEL 制約の回路図構文は、「BEL」の「回路図からの設定」を参照してください。

## VHDL

VHDL コードの場合、制約は VHDL 属性で記述します。次のような構文で制約を宣言してから、制約を使用する必要があります。

```
attribute attribute_name : string;
```

例

```
attribute RLOC : string;
```

属性は、エンティティまたはアーキテクチャで宣言できます。

- ・ エンティティで宣言すると、エンティティとアーキテクチャ本体の両方で属性を使用できます。
- ・ アーキテクチャで宣言した場合は、その属性はエンティティ宣言では使用できません。

VHDL 属性を宣言した後、次のように指定します。

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

使用できる属性値 *attribute\_values* は、属性のタイプによって異なります。

例 1 :

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

例 2 :

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

ザイリンクスで最もよく使用されるオブジェクトは、signal、entity、label です。label はコンポーネントのインスタンスを表します。

メモ : signal 属性は、出力ポートに使用される必要があります。

VHDL では大文字と小文字の区別をする必要はありません。

ザイリンクス制約が VHDL キーワードである場合、VHDL 属性でその制約を使用できません。この問題を回避するには、制約エイリアスを使用します。各制約には、固有のエイリアスがあります。エイリアス名は、接頭辞 XIL\_ に制約名を付けたものです。たとえば、RANGE 制約は VHDL 属性には使用できないので、XIL\_RANGE を使用します。

## Verilog

属性は、(\*) で区切り、次の構文に従って記述する必要があります。

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

この場合、次を表します。

- ・ attribute は、参照する信号、モジュール、またはインスタンスの宣言の前に記述する必要があります。
- ・ attribute\_value には、文字列を指定する必要があります。整数値やスカラー値は使用できません。
- ・ attribute\_value は、二重引用符 (") で囲む必要があります。
- ・ デフォルト値は 1 です。( \* attribute\_name \*) is the same as ( \* attribute\_name = "1" \*) と同じです。

### Verilog 属性の構文例 1

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

### Verilog 属性の構文例 2

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

### Verilog 属性の構文例 3

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

### Verilog 属性の構文例 4

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

## Verilog の制限

次の Verilog 属性は、サポートされていません。

- ・ 信号の宣言
- ・ ステートメント
- ・ ポートの接続
- ・ 論理演算子

## Verilog のメタ コメント

メタコメントを使用すると、制約を Verilog コードで指定することもできます。Verilog 形式がよく使用されますが、Verilog でもメタコメントがサポートされています。次の構文を使用してください。

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

例

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

## UCF

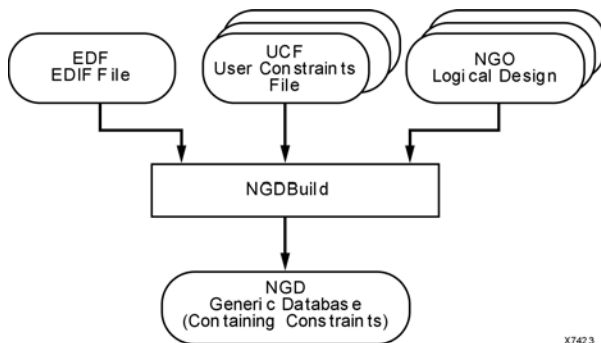
UCF ファイルは、論理デザインに制約を指定する ASCII 形式のファイルです。UCF ファイルは、ユーザーがテキストエディタで作成して、制約を入力できます。Constraints Editor を使用しても UCF ファイル内に制約を作成できます。LFSR の使用方法については、「[Constraints Editor](#)」を参照してください。

このファイルで指定する制約によって、論理デザインがターゲット デバイスでどのようにインプリメントされるかが決定されます。UCF ファイルは、デザイン入力中に指定した制約を上書きする際に使用されます。

## UCF フロー

次の図に、UCF フローを示します。

### UCF ファイルのフロー



UCF ファイルは NGDBuild の入力ファイルです (上図を参照)。UCF ファイルに記述された制約は、NGDBuild で生成される NGD ファイルの情報の一部となります。FPGA デバイスの場合、これらの制約はデザインのマップで使用されたり、MAP で生成される PCF (物理制約ファイル) に書き込まれます。

PCF ファイルの制約は、デザインのマップ後に実行する PAR やタイミング解析ツールなどのツールで使用されます。

## 手動でのタイミング制約の入力

Constraints Editor でタイミング制約を入力する方法以外に、タイミング仕様を制約として UCF ファイルに手動で入力することもできます。その後、NGDBuild を実行する際に、タイミング制約は NGD の一部としてデザインのデータベースに追加されます。Constraints Editor を使用しても UCF ファイルに制約を作成できます。

## UCF ファイルが複数ある場合の制約の競合

ザイリンクスでは、HDL/NCF/UCF/PCF プロセスと同様、最後に入力した制約が優先される方式をとっています。現在のところ、UCF ファイルはプロジェクトに追加した順序で処理されます (Project Navigator または Tcl コマンド)。タイムスタンプ、またはファイルが修正された順序は記録されません。

## UCF および NCF の構文

論理制約は、次のファイルに記述されています。

- ・ NCF (ネットリスト制約ファイル) : 合成プログラムにより生成される ASCII 形式のファイル
- ・ UCF (ユーザー制約ファイル) : ユーザーが作成する ASCII 形式のファイル

ユーザー制約を設定する場合は、PCFではなく、UCF または NCF に入力することをお勧めします。

## UCF および NCF の一般規則

- ・ UCF ファイルおよび NCF ファイルでは、大文字と小文字が区別されます。識別子名（ネット名などのデザイン内のオブジェクトの名前）は、ソース デザインのネットリストに記述されている名前と大文字/小文字が正確に一致しなければなりません。ただし、LOC、PERIOD、HIGH、LOW などの制約キーワードは、大文字か小文字のいずれか、またはその両方を使用できます。
- ・ 各制約文の終わりには、セミコロン（;）を付けます。
- ・ 制約文の最後にセミコロンを付けるため、文が 2 行にまたがる場合でも行継続文字を使用する必要はありません。
- ・ 似たようなブロックまたはコンポーネントのタイミング制約は個別に設定せず、まとめて 1 つのタイミング制約を設定します。
- ・ UCF および NCF にコメントを追加するには、各コメント行の先頭にシャープ記号（#）を付けます。例は次のとおりです。

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

C および C++ 形式のコメント（/\* ... \*/ など）もサポートされます。

- ・ UCF および NCF では、文を特定の順序に並べる必要はありません。
- ・ ネット名 (NET) およびインスタンス名 (INST) は、ダブル クォーテーションで囲むことをお勧めします。ただし、これはエラーを防ぐためであり、必須条件ではありません。
- ・ ~OUTSIG1 のようなチルダを含む反転信号名は、必ずダブル クォーテーションで囲んでください。
- ・ 指定したインスタンスに対して複数の制約を設定できます。詳細は、下記の「複数の制約の入力」を参照してください。

## 制約の競合

UCF/NCF ファイル、回路図、合成ファイルにそれぞれ記述された制約は、同等に適用されます。制約がどのファイルに入力されているかは問題ではありません。制約が重複した場合、UCF の制約が NCF や回路図/ネットリストの制約よりも優先されます。NCF の制約は回路図/ネットリストの制約よりも優先されます。

1 つの位置に複数のエレメントを誤ってロックしてしまった場合、マップにより競合が検知されてエラー メッセージが表示されるので、エラーを修正します。

## 構文

UCF ファイルでは、次のような基本的な構文がサポートされます。

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

- ・ *full\_name* は、階層名が付いたオブジェクト名です。名前がピンを表す場合は、そのエレメントのインスタンス名も必要です。
- ・ *constraint* は、回路図オブジェクトに属性を設定する場合に使用されるのと同じ形式の制約です。LOC=P38 や FAST などがその例です。</fc>

## TIMEGRP および TIMESPEC 属性の指定

TIMEGRP 属性を指定するには、制約ファイルで属性を定義する前にキーワード TIMEGRP を付けます。

```
TIMEGRP "input_pads"=PADS EXCEPT output_pads;
```

## 予約語の使用

すべての制約ファイル (NCF、UCF、PCF) で、内部予約語と一致するインスタンス名または変数名は、二重引用符で囲んでおかないと処理されないことがあります。したがって、すべての名前を二重引用符で囲むことをお勧めします。

たとえば、net は予約語なので、次の構文は使用できません。

```
NET "net" OFFSET=IN 20 BEFORE CLOCK;
```

この場合は、次のように入力することをお勧めします。

```
NET "net" OFFSET=IN 20 BEFORE CLOCK;
```

または

```
NET "$SIG_0" OFFSET=IN 20 BEFORE CLOCK;
```

~OUTSIG1 のようなチルダを含む反転信号名は、次のようにダブル クォーテーションで囲む必要があります。

```
NET "~OUTSIG1" OFFSET=IN 20 BEFORE CLOCK;
```

## ワイルドカード

制約文では、次のようなワイルドカード文字を使用できます。

- ・ アスタリスク (\*) : 任意の数の文字列
- ・ 疑問符 (?) は任意の 1 文字を表します。

ネット名にワイルドカード文字を使用すると、出力ネット名が特定の文字列またはパターンに一致するシンボルのグループを選択できます。たとえば、次のような形式に沿った名前を指定すると、その名前のネットが接続されているパッドの出力速度が速くなります。

- ・ アスタリスク (\*) で表された文字列
- ・ 最初の文字列の後に AT を付ける
- ・ クエスチョン マーク (?) で表された 1 文字

```
NET "**AT?" FAST;
```

インスタンス名では、ワイルドカード文字は該当する種類のすべてのシンボルを表します。たとえば、次の制約は ROM の集合全体を特定の 16 進数の値 5555 に初期化します。

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

ワイルドカード文字を長いインスタンス名の一部として使用すると、ワイルドカード文字はその位置の 1 文字または複数の文字を表します。

ロケーションを指定する場合は、ワイルドカード文字を行番号または列番号の代わりに使用できます。たとえば、次の制約は、loads\_of\_logic 階層以下のインスタンスの配置を列 8 の SLICE に指定します。

```
INST "/loads_of_logic/*" LOC=SLICE_X*Y8;
```

1 つの制約内で、ワイルドカード文字を行番号と列番号の両方には使用できません。

## 階層の指定

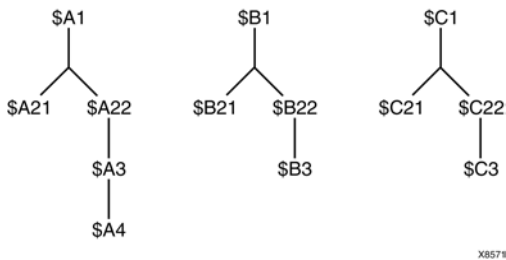
インスタンス名を検索するとき、最上位レベルのブロック名 (デザイン名) は無視されます。ワイルドカード (\*) を使用すると、UCF または NCF 内で処理の対象となるデザインの階層を指定できます。これには、次の構文が適用されます (level1 は階層レベル名の例です)。

### UCF デザイン階層

*	階層の全レベルを対象とします。
level1/*	level1 に含まれるブロックおよび下位レベルに含まれるブロック全体を対象とします。
level1/*/	level1 に含まれるブロックを対象とし、それ以下のレベルに含まれるブロックは対象からはずします。

次のようなデザイン階層があるとします。

### UCF デザイン階層



このデザイン階層例の場合、ワイルドカードの範囲は次のようになります。

```

INST *           => <everything>
INST /*          => <everything>
INST /*/         => <$A1,$B1,$C1>
INST $A1/*       => <$A21,$A22,$A3,$A4>
INST $A1/*/*     => <$A21,$A22>
INST $A1/*/*/*   => <$A3,$A4>
INST $A1/*/*/*/* => <$A3>
INST $A1/*/*/*/* => <$A4>
INST $A1/*/*/*/* => <$A4>
INST /*/*22/     => <$A22,$B22,$C22>
INST /*/*22      => <$A22,$A3,$A4,$B22,$B3,$C3>
  
```

## 複数制約の入力

指定したインスタンスに対して複数の制約を UCF ファイルに設定することができます。

**INST** *instanceName* *constraintName* = *constraintValue* | *constraintName* = *constraintValue*;

例 :

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

## ファイル名

デフォルトで制約ファイルは NGDBuild によって読み出されます。制約ファイル名は入力デザイン名と同じで、拡張子は .ucf です。異なる制約ファイル名を指定するには、NGDBuild を実行する際に -uc オプションを使用します。EDIF 入力ファイルと同じベース名を持つ NCF が EDIF と同じディレクトリに含まれている場合、NCF が自動的に読み込まれます。

NGDBuild、MAP、PAR などのインプリメンテーション ツールを実行するには、コマンドラインでファイル名の拡張子を .ucf などとすべて小文字で入力する必要があります。



## インスタンスおよびブロック

次に、制約ファイルの構文で使用されるインスタンスとブロックを定義します。

- ・ *instance* とは、回路図上のシンボルです。
- ・ *instance name* : EDIF ネットリストに含まれるシンボル名です。
- ・ *block* とは、CLB または IOB のことです。
- ・ *block name* : BLKNM、HBLKNM、または XBLKNM 属性を使用して指定できます。デフォルトでは、ブロックに関連する信号名に基づいてブロック名が割り当てられます。

## PCF

デザイン ネットリストがザイリンクス開発システムに読み出される際に生成される NGD ファイルには、多数の論理制約が含まれます。これらの制約は次のいずれかをソースとしています。

- ・ 回路図または HDL ファイルに記述された属性
- ・ UCF (ユーザー制約ファイル) に入力された制約
- ・ CAE ベンダー ツールにより生成された NCF (ネットリスト制約ファイル) の制約

NGD に記述された論理制約は、MAP により読み込まれます。一部の論理制約は、デザインのマップに使用され、物理制約に変換されます。物理制約はその後、物理制約ファイル (PCF) に書き込まれます。

PCF は、次の 2 つのセクションに分割された ASCII 形式のファイルです。

- ・ マップにより作成された物理制約のセクション
- ・ ユーザーが入力した物理制約のセクション

マップで生成された制約のセクションは、マップを実行するたびに書き込まれます。

ファイルでは、マップで生成された制約の後にユーザーが入力した制約が記述されています。制約の競合が発生しても、常にこの順番が維持され、ユーザーが入力した制約が最後に読み込まれてマップで生成された制約に上書きされます。

マップで生成された制約のセクションは、SCHEMATIC START 行で開始し、SCHEMATIC END で終了します。タイミング制約などのユーザー制約を入力する場合は、常に SCHEMATIC END の後に記述してください。

ユーザー制約はファイルに直接書き込むか、FPGA Editor を使用して入力します。FPGA Editor の制約情報については、FPGA Editor ヘルプを参照してください。

**メモ :** デザイン制約は、できる限り HDL、回路図、または UCF ファイルに書き込んでください。PCF ファイル以外のファイルに書き込むと、デザインを簡単に保存でき、デザイン ルール チェックが向上します。

PCF ファイルは、PAR、FPGA Editor、TRACE、NetGen および BitGen の入力ファイル (オプション) として使用されます。

このファイルには制約およびコメントをいくつでも含めることができます。コメントは、シャープ記号 (#) またはダブル スラッシュ (//) で始めます。文字数に制限はありませんが、1 行以内におさめる必要があります。

PCF ファイルの構文は次のとおりです。

```
schematic start;  
  
translated schematic and UCF and NCF constraints in PCF format  
  
schematic end;  
  
user-entered physical constraints
```

**注意：** ユーザー制約は、schematic end 文の後に記述する必要があります。このセクションの前またはセクション内に記述された制約は、上書きされるか、無視される場合があります。

回路図制約には変更を加えないでください。この制約は、マップで新しい PCF ファイルが作成されるたびに上書きされます。

グローバル制約は、オブジェクトに設定せず、制約ファイルに入力してください。

各制約文の最後に、セミコロン (;) を付ける必要があります。

NCF、UCF、および PCF のすべての制約ファイルで、内部予約語と一致するインスタンス名または変数名をダブルクォーテーションで囲んでおかないと処理されません。したがって、すべての名前を二重引用符で囲むことをお勧めします。たとえば、net は予約語なので、次の構文は使用できません。

**NET net FAST;**

この場合は、次のように入力することをお勧めします。

**NET "net" FAST;**

## NCF

NCF ファイルの構文ルールは UCF のルールと同じです。詳細は、「[UCF および NCF 構文](#)」を参照してください。

## Constraints Editor

Constraints Editor は、タイミング制約の入力プロセスを単純化するグラフィカル ツールで、UCF 構文を理解していなくても制約を簡単に作成できます。Constraints Editor を使用できる制約およびデバイスについては、「[制約の入力方法](#)」を参照してください。Constraints Editor の実行に関する詳細は、ISE® ヘルプを参照してください。

NGDBuild による変換後、デザインのインプリメンテーションに NGCBuild を使用すると、UCF を直接変更せずに制約の作成および変更ができます。Constraints Editor で制約を作成、変更した後は、新しい UCF とデザイン ソース ネットリストを入力ファイルとして使用して NGCBuild をもう一度実行し、新しい NGD を出力する必要があります。

## 入力/出力

Constraints Editor には、次の必要です。

- ・ User Constraints File (UCF)
- ・ Native Generic Database (NGD) ファイル

NGD には、グループ化する論理エレメントの名前が記述されています。出力には UCF が使用されます。

Constraints Editor を起動して、UCF ファイルを開きます。ここで UCF および NGD のベース名が異なる場合、正しい NGD ファイルを選択する必要があります。詳細は、Constraints Editor ヘルプを参照してください。

制約を設定すると、Constraints Editor により制約が UCF に出力されます。UCF ファイルは、NGCBuild によりデザイン ソースのネットリストと共に使用され、NGD ファイルが生成されます。NGD ファイルは MAP プログラムで読み出されます。MAP は、物理的デザインのデータベースを NCD (Native Circuit Description) ファイル形式で生成し、PCF (物理制約ファイル) も生成します。これらのファイルはインプリメンテーション ツールによって使用され、最終的にビットストリームが生成されます。

**メモ：** すべてのザイリンクス制約が Constraints Editor で設定できるわけではありません。

## Constraints Editor の起動

Constraints Editor は PC および UNIX で実行でき、次のいずれかの方法で起動できます。

- ・ [開始位置 Project Navigator](#)
- ・ [スタンドアロンとして起動](#)
- ・ [コマンド ラインからの起動](#)

### Project Navigator からの起動

Project Navigator の [Processes] ウィンドウから Constraints Editor を起動します。

1. [Sources] ウィンドウでデザイン ファイルを選択します。
2. [Processes] ウィンドウで [Design Utilities] → [User Constraints] の下の [Create Timing Constraints] をダブルクリックします。

### スタンドアロンとして起動

Constraints Editor をスタンドアロン型ツールとしてインストールしている場合は、次のいずれかに従って起動します。

- ・ Windows デスクトップ上の [Constraints Editor] アイコンをクリックします。
- ・ [スタート] → [プログラム] → [アクセサリ] → [Constraints Editor] をクリックします。

### コマンド ラインからの起動

コマンド ラインから Constraints Editor を起動する方法はいくつかあります。

### ファイルを読み込まずに起動

ファイルを読み込まずに Constraints Editor を起動するには、次のように入力します。

```
constraints_editor
```

### NGD ファイルを読み込んで起動

NGD を読み込んで Constraints Editor を起動するには、次のように入力します。

```
constraints_editor ngdfile_name
```

*ngdfile\_name* は、NGD ファイル名です。

拡張子は .ngd とする必要があります。

NGD と同じベース名を持つ UCF がある場合、UCF も一緒に読み込まれます。同じベース名のファイルがない場合は、UCF を指定するようメッセージが表示されます。

### NGD および UCF を読み込んで起動

NGD および UCF を読み込んで Constraints Editor を起動するには、次のように入力します。

```
constraints_editor ngdfile_name -uc ucf_file_name
```

- ・ *ngdfile\_name* は、NGD ファイル名です。
- ・ *ucf\_file\_name* は、UCF ファイル名です。

拡張子は .ucf にする必要があります。

## バックグラウンドとして起動

UNIX でバックグラウンドとして Constraints Editor を起動するには、次のように入力します。

```
constraints_editor &
```

## Project Navigator

Project Navigator でインプリメンテーション制約を設定するには

- ・ FPGA デバイスを使用している場合、インプリメンテーションのプロセス プロパティで、デザインの変換、マップ、配置、および配線の方法を指定できます。複数のプロパティを設定して、インプリメンテーション プロセスを制御できます。
- ・ CPLD デバイスを使用している場合、インプリメンテーションのプロセス プロパティで、デザインの変換やフィットの方法を指定できます。

詳細は、Project Navigator ヘルプの [Process Properties] ダイアログ ボックスに関するセクションを参照してください。

## PlanAhead

PlanAhead™ ソフトウェアは、合成前または合成後に使用できます。PlanAhead は、Virtex®-4 および Spartan®-3 以降のデバイスすべてで起動します。ドラッグ アンド ドロップ機能を使用することで、ピン配置制約、ロジック配置、エリア制約などの配置制約を簡単に入力できます。詳細およびデバイス サポートについては、『PlanAhead ユーザー ガイド』を参照してください。

## 配置制約の割り当て

FPGA をターゲットにする場合は、PlanAhead を使用して I/O ピンやロジック割り当て、グローバル ロジック配置、エリア グループ割り当てなどを制御する配置制約を入力できます。PlanAhead は、デザイン プロセスのさまざまな段階で自動的に起動され、デザインを解析したり、配置制約を適用するために使用されます。PlanAhead の簡易版は Project Navigator から起動され、選択したタスクを実行するのに必要な機能だけが使用可能になります。スタンドアロンの PlanAhead には、さらに多くの機能が含まれます。

PlanAhead を Project Navigator から起動する際には、次に気を付けてください。この場合、CPU プロセスは別で、その他一部のツールと同様、Project Navigator とリアルタイムで連動していません。PlanAhead を起動中に Project Navigator ソース ファイルをアップデートすると、データが一致しなかったり、同期エラーになったりすることがあります。

PlanAhead を起動中は、ソース ファイルをまず PlanAhead に渡してから、PlanAhead プロジェクトを作成してください。PlanAhead プロジェクトを保存すると、修正された UCF ファイルが Project Navigator に戻されて、Project Navigator プロジェクトがアップデートされます。入力ソース ファイルは、起動するプロセスによって異なります。

どの形式のファイルが渡されるかは、この後の「ピン配置」および「フロアプランおよび配置制約」セクションを参照してください。次のセクションでは、PlanAhead を使用した配置制約の入力について説明します。

## I/O ピン コンフィギュレーションの定義

ピンの割り当ての概要

PinAhead は、スタンドアロン ツールとして起動できるほか、Project Navigator から起動できます。スタンドアロンの PinAhead は、HDL ソースがまだ完成していないようなデザイン プロセスの早期段階で使用すると便利です。I/O ポートはこのツール内で手動で定義できるほか、CSV 形式のスプレッドシートや HDL ソースにもインポートできます。初期のピン配置を定義してから UCF ファイルをエクスポートして、Project Navigator フローで使用することもできます。

Project Navigator から PinAhead を起動する場合は、UCF ファイルが必要です。UCF ファイルがないと、空のファイルが作成されます。Project Navigator から PinAhead を起動する場合は、I/O ポートの手動作成や CSV スプレッドシートのインポートはできません。

PinAhead は、さまざまな便利なビューや機能を含む I/O ピン割り当てのための環境です。I/O ポートのグループをさまざまな方法で別々にデバイスにドラッグ アンドドロップできます。自動配置ルーチンも使用できます。デザイン ルール チェック (DRC) を使用することで、有効なピン配置の定義を確認することもできます。

#### I/O ピン データ情報の確認

I/O 規格を含むデバイス仕様については、[データシート](#)を参照してください。デバイス特有の I/O 規格情報については、ターゲットにするデバイスのデータシートを参照してください。データシートに含まれるデータの多くは、PinAhead ツールからも入手できます。入手可能な情報には、I/O 規格、クロック対応のピン、内部トレース遅延、差動ペア、クロック領域、I/O バンクの内容などがあります。グローバル/リージョナルクロックバッファ、I/O 遅延、遅延コントローラ、ギガビットトランシーバなどの I/O 関連のデバイスリソースについての情報も含まれます。

#### ピンの割り当て

PinAhead をスタンドアロンで起動するには、Windows の PlanAhead のデスクトップ アイコンをクリックするか、Linux コマンドラインで planAhead と入力します。Project Navigator から起動する場合は、次のいずれかの方法でピン割り当てプロセスを実行します。一番適切な方法を使用してください。

- [Floorplanning I/O - Pre-Synthesis]

このコマンドまたはプロセスを使用すると、HDL ソースファイルが PlanAhead に渡され、最上位レベルの I/O ポート情報のみが抽出されます。UCF が Project Navigator プロジェクトに既に存在する場合は、それが PlanAhead に渡されます。UCF ファイルがない場合は、作成することを促すメッセージが表示されます。UCF ファイルが複数存在する場合、新しい制約を追加するために使用するファイルを選択するように促すメッセージが表示されます。既存の制約は、それが含まれるファイルで修正されます。

PlanAhead に含まれる PinAhead の使用方法については、「PinAhead のマニュアル」セクションを参照してください。

I/O ピンの割り当てができれば、PlanAhead プロジェクトを保存して PlanAhead を終了します。これで Project Navigator プロジェクトの UCF ファイルがアップデートされ、プロジェクトステータスもアップデートされます。PlanAhead を保存せずに閉じると、Project Navigator の UCF ソースファイルまたはステータスが変更されません。

- [Floorplanning a Design - Post-Synthesis]

このコマンドまたはプロセスを使用すると、合成済みのネットリストソースファイルが PlanAhead に渡されます。UCF が Project Navigator プロジェクトに既に存在する場合は、それが PlanAhead に渡されます。UCF ファイルがない場合は、作成することを促すメッセージが表示されます。UCF ファイルが複数存在する場合、新しい制約を追加するために使用するファイルを選択するように促すメッセージが表示されます。既存の制約は、それが含まれるファイルで修正されます。

合成済みネットリストを入力ファイルとして使用すると、PinAhead でクロックおよびクロック関連のロジックが認識されるようになるので、使える PinAhead の機能がさらに増えます。I/O プラン機能および DRC が使用できるようになり、さらに知的なピン割り当てが可能になります。デザインの接続も解析できるようになり、I/O に関するデバイスリソースを最小限に抑えることができます。

PlanAhead に含まれる PinAhead の使用方法については、「PinAhead のマニュアル」セクションを参照してください。

I/O ピンの割り当てができれば、PlanAhead プロジェクトを保存して PlanAhead を終了します。これで Project Navigator プロジェクトの UCF ファイルがアップデートされ、プロジェクトステータスもアップデートされます。PlanAhead を保存せずに閉じると、Project Navigator の UCF ソースファイルまたはステータスが変更されません。

#### PinAhead 資料

『PlanAhead ユーザーガイド』の「I/O ピンの配置」には、デバイスリソースと I/O ピン割り当ての解析をする PinAhead に関する情報が含まれます。



『PlanAhead チュートリアル』の「PinAhead を使用した I/O ピンの割り当て」には、PinAhead を使用したピン割り当て方法例が具体的に示されています。

PinAhead のビデオ デモについては、<http://japan.xilinx.com/design> を参照してください。

## フロアプランおよび配置制約

PlanAhead は、接続、集積度、タイミングなどを含むさまざまな側面からデザインを解析する総合的な環境を提供するツールです。PlanAhead で配置制約を設定すると、結果がより良く、一貫性のあるものになるように、インプリメンテーション ツールを導くことができます。この配置制約には、特定のロジックをデバイスの特定のサイトに固定する LOC 制約やロジックのグループをデバイスの特定エリアに制約する AREA\_GROUP 制約も含まれます。

配置制約の種類については、「制約の概要」を参照してください。

### 配置制約 (LOC) の割り当て

PlanAhead を使用すると、どのロジックでも特定のデバイス サイトに固定できます。これは、BUFG、BRAM、MULT、PPC405、GT、DLL、DCM のようなグローバル ロジック オブジェクトに設定することもできます。

ロジック オブジェクトは、そのロジック オブジェクトを該当する PlanAhead のビューからドラッグして、ワークスペースの [Device] ビューにドロップするだけで配置できます。I/O ポートのような、そのオブジェクトの [General Properties] ビューに該当ロケーション サイトを入力することができるロジックもあります。

配置制約の割り当ての詳細は、『PlanAhead ユーザー ガイド』の「デザインのフロアプラン」の「配置制約の使用」のセクションを参照してください。

### エリア グループの割り当て

エリアグループは、たとえば特定のクロック 領域内などのデバイスの特定の領域にロジックを配置する主な手段です。PlanAhead を使用するとさまざまな手段でエリア グループを作成できます。DRC を含め、接続、サイズ ロジック タイプ、範囲などが提供されるので、最適な AREA\_GROUP プロパティの設定が可能です。

エリア グループ制約の作成方法は、『PlanAhead ユーザー ガイド』の「デザインのフロアプラン」セクションを参照してください。

## PACE

CPLD の場合、制約は PACE (Pinout Area Constraints Editor) で設定できます。PACE の Pin Assignment Editor は、ロケーション制約をデザインの I/O に割り当てる場合に使用します。さらに、I/O 規格など特定の I/O プロパティの指定にも使用します。

PACE を使用できる制約およびデバイスについては、「[制約の入力方法](#)」を参照してください。PACE の起動および使用方法の詳細については、PACE ヘルプを参照してください。

## 未完成のデザインにおけるピンの事前割り当て

**メモ：** このセクションでは、未完成のデザインにおけるピンの事前割り当てについて説明します。定義したピンに制約を追加すると HDL テンプレートが生成される、「ピンの事前割り当て」は PlanAhead™ または PACE で実行できます。詳細は、ISE® ヘルプを参照してください。PACE は、CPLD 用、PlanAhead は FPGA 用です。

デザインが未完成でも、ピンの割り当て、電圧標準、バンク規則、その他のボードの条件などのレイアウト要件が整っていることがあります。ピンの事前割り当ての機能を使用すると、デザインのロジックが完成していなくてもデザインのピン配置ルールを決定できます。

PlanAhead または PACE でピンの事前割り当て機能を使用するには、次の手順に従います。

1. 最上位デザインのポートをすべて特定します。
2. I/O 制約を割り当てます。

入力ピンにロードがない、出力ピンにソースがないなどのように、デザイン内のどのロジックでも使用されていないポートも制約の適用を受け、インプリメントされます。

ほかの I/O ピンと同じように、LOC または IOSTANDARD 制約を UCF で割り当てます。これらの要件はデータベースでアノテートされます。PlanAhead および PACE を使用して、ピンのロケーション、バンクグループ、電圧標準を割り当て、DRC チェックを実行します。最終的な PAD レポートには、ロジック、または制約が関連付けられたピンが含まれます。

このデザイン インプリメンテーションは完全なものではなく、ハードウェアにはダウンロードできません。BitGen (ビットストリーム生成) の デザイン ルール チェックで次のようなエラーが発生します。

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

デザインから使用されていないポートを取り除くには、関連付けられた制約を削除します。変換プロセス (NGDBuild) で使用されていないピンが削除されます。

以下の例では、最上位レベルに 6 つのポートがあり、clk、A、C の 3 つだけがデザインで使用されています。残りの 3 つのポートは、次のとおりです。

- ・ B は LOC 制約があるので保持
- ・ D は IOSTANDARD 制約があるので保持
- ・ E は使用されておらず、制約もないのでデザインから削除

## Verilog コード例

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

## UCF の例

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

## FPGA Editor

FPGA Editor を使用すると、制約を PCF ファイルに追加したり、ファイルから制約を削除できます。FPGA Editor では、ネットおよびコンポーネントのプロパティフィールドとしてネット、サイト、コンポーネントに設定する制約がサポートされています。プロパティは Setattr コマンドで設定でき、Getattr コマンドで読み込まれます。

ブール属性 (BLOCK、LOCATE、LOCK、OFFSET IN、OFFSET OUT、PROHIBIT) の値は On または Off です。オフセットの方向には In または Out を、オフセットの順番には Before または After という値を使用します。それ以外の制約には数値を指定します。制約を削除するには Off を指定します。値の指定には大文字と小文字を区別する必要はありません (たとえば、On でも on でも同じです)。

FPGA Editor で制約を作成する場合、デザインを保存するたびに PCF ファイルに制約が書き込まれます。FPGA Editor を使用して制約を削除し、デザイン ファイルに保存した場合、PCF ファイル内で制約が記述された行は自動的にコメントとして保持されます。次の表に、FPGA Editor でサポートされる制約を示します。

### FPGA Editor でサポートされる制約

制約	制約を指定する場所
block paths	[Component Properties] および [Path Properties] プロパティシート
define path	[Viewed with Path Properties] プロパティシート
location range	[Component Properties Constraints] ページ
locate macro	[Macro Properties Constraints] ページ
lock placement	[Component Properties Constraints] ページ
lock routing of this net	[Net Properties Constraints] ページ
lock routing	[Net Properties Constraints] ページ
maxdelay allnets	[Main Properties Constraints] ページ
maxdelay allpaths	[Main Properties Constraints] ページ
maxdelay net	[Net Properties Constraints] ページ
maxdelay path	[Path Properties] プロパティシート
maxskew	[Main Properties Constraints] ページ
maxskew net	[Net Properties Constraints] ページ
offset comp	[Component Properties Offset] ページ
penalize tilde	[Main Properties Constraints] ページ
period	[Main Properties Constraints] ページ
period net	[Net Properties Constraints] ページ
prioritize net	[Net Properties Constraints] ページ
prohibit site	[Site Properties] プロパティシート

### 固定されたネットとコンポーネント

ネットが固定されている場合、ネット全体、ネットの一部、ピン、またはワイヤなどの配線を解除できません。ネットの配線を解除するには、まず固定を解除する必要があります。固定されたネットに、ピンや配線を追加できます。

PCF ファイルで Lock Net [ *net\_name* ] 制約が有効になっている場合、FPGA Editor でネットは固定された状態で表示されます。固定制約を削除するには、[ネットのプロパティ] ダイアログ ボックスを使用します。

コンポーネントが固定されている場合、PCF ファイルで次のように設定します。

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

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

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

**lock placement**



コンポーネントが固定されている場合、配置の解除はできませんが、配線は解除できます。コンポーネントの配置を解除するには、まず固定を解除する必要があります。

## 制約の相互関係

回路図制約は、MAP により PCF ファイルの最初に記述されます。このセクションの最初は **SCHEMATIC START**、最後は **SCHEMATIC END** と記述されます。制約は後ろから使用されるので、すべての制約は **SCHEMATIC END** の後に入力する必要があります。

ユーザー制約は、回路図制約セクションの前に記述してもかまいませんが、競合する制約があった場合、回路図制約が優先されることがあります。

デザインのマップが実行されるたびに、PCF ファイルの回路図セクションが上書きされます。原則としてユーザー制約セクションはそのまま保持されますが、制約によっては無効になるものがあります。

## XCF

XST の制約は、ザイリンクス制約ファイル (XCF) で指定できます。制約ファイルの拡張子は .xcf です。ISE® Design Suite で XCF を指定する方法については、ISE ヘルプを参照してください。詳細は、『XST ユーザー ガイド』の「XST 制約ファイル」を参照してください。

## 制約の優先順位

場合によっては、2 つ以上のタイミング制約がデザイン内の同じパスに設定されることがあります。このような場合、そのパスに対して優先度の高い制約が適用され、優先度の低い制約はそのパスでは無視されるようにして、制約同士の競合が起こらないようにする必要があります。優先順序は、制約を指定した順序と制約の優先度の両方の条件によって異なります。優先順序の規則は、次のようになります。制約の優先度によって決まるほか、同じ優先度の制約が重なっている場合は、PCF ファイルでのどの制約が後に表示されているかによって決まります。たとえば、同じパスをカバーする PERIOD 制約が 2 つある場合、PCF ファイルで後の方に記述されている PERIOD 制約が使用され、これらのパスが解析されます。前の方の PERIOD 制約については、タイミング レポートで「0 items analyzed」と表示されます。このデフォルトの優先順序を変更する場合は、PRIORITY キーワードを使用して優先度を設定する必要があります。

## ファイルの優先順位

同じ優先度の制約により競合が発生する場合、指定する順序によってどちらの制約が優先されるかが決まります。同じ優先度の制約の場合は、最後に記述された制約が前に記述された制約を上書きします。この規則は、1 つの UCF ファイルだけでなく、複数の UCF ファイルで定義された制約にも適用されます。

次のリストは、同じ優先度の制約が別の制約ファイルで定義されている場合に、どのファイルの制約が優先されるかを表示しています。優先度の高いファイルの制約から順にリストしています。

- ・ Physical Constraints File (PCF) の制約
- ・ User Constraints File (UCF) の制約
- ・ Netlist Constraints File (NCF) の制約
- ・ 回路図の属性、またはネットリストに渡される HDL で指定された制約

## タイミング制約の優先順位

デザイン内の同じパスに対して 2 つの異なる制約が設定されている場合、優先度の高い制約が適用され、優先度の低い制約は無視されます。別の制約の優先度を定義する規則は、次のようになります。優先度の高い制約から順にリストしています。

- ・ タイミング無視 (TIG)
- ・ FROM-THRU-TO
  - ソースおよびデスティネーションがユーザー定義のグループ
  - ソースまたはデスティネーションがユーザー定義のグループ
  - ソースおよびデスティネーションが定義済みグループ
- ・ FROM TO
  - ソースおよびデスティネーションがユーザー定義のグループ
  - ソースまたはデスティネーションがユーザー定義のグループ
  - ソースおよびデスティネーションが定義済みグループ
- ・ OFFSET 制約
  - 特定のデータ IOB (NET OFFSET) 制約
  - データ IOB のタイム グループ (グループ化された OFFSET) 制約
  - すべてのデータ IOB (グローバル OFFSET) 制約
- ・ PERIOD

## OFFSET の優先度

同じパスに指定されている 2 つの OFFSET 制約が同じ優先順位の場合、レジスタ修飾子の付いた方が優先されます。修飾子で判断できない場合は、制約ファイルで後に記述された方が優先されます。

## MAXSKEW および MAXDELAY の優先度

ネット遅延およびネット スキューの仕様はパス遅延解析とは別に解析され、相互に干渉されることはありません。NET TIG は NET 制約と関係しており、優先されます。

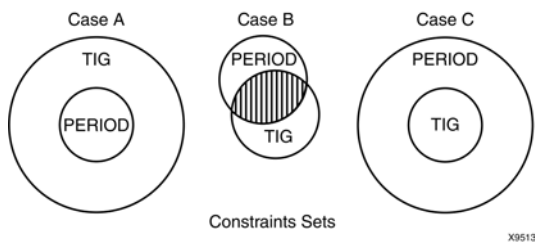
## 優先順位の例外

PRIORITY キーワードはデフォルトの優先順序を上書きするために使用します。PRIORITY キーワードには -255 から +255 までの値を使用して、制約に優先度の手動で割り当てます。値が小さいほど、優先順位が高くなります。このキーワードは制約の優先順序のみに適用されるもので、インプリメンテーション ツールが制約の適用されるリソースを配置配線する順序には影響しません。PRIORITY 制約の詳細については、第 6 章を参照してください。

## 制約集合の相互関係

制約がルールどおりに優先されない場合があります。これは図のように上位集合や下位集合が含まれていたり、制約の集合が重なり合っている場合です。

## 制約集合同士の相互関係



- ・ ケース A の場合、TIG グループが PERIOD グループの上位グループになっていると問題が発生します。
- ・ ケース B の場合、TIG グループと PERIOD グループが部分的に重なり合っており、この部分の制約が場合によって TIG と判断されたり PERIOD と判断されたりします。



# タイミング制約の設定方法

タイミング制約は、すべてのデザイン要件がインプリメンテーション ツールに渡されるのを確実にするために使用します。このためには、デザインのすべてのパスに適切な制約が付いている必要があります。この章では、できるだけ効率的なやり方で FPGA デバイスによくあるタイミング パスを識別して制約を付ける方法を説明します。この章は、次のセクションから構成されています。

- ・ [基本的な制約設定方法](#)
- ・ [入力タイミング制約](#)
- ・ [Register-to-Register タイミング制約](#)
- ・ [出力タイミング制約](#)
- ・ [例外のタイミング制約](#)

## 基本的な制約設定方法

デザインに含まれるすべてのパスのタイミング要件は、インプリメンテーション ツールに渡される必要があります。タイミング要件はそのパスのタイプによって複数のグローバル カテゴリに分割できます。よくあるパスのタイプには、入力パス、レジスタからレジスタへのパス、出力パス、パス特有の例外などがあります。これらのグローバル カテゴリのタイプが、それぞれザイリンクスのタイミング制約になります。これらの制約を指定するには、まずグローバル制約から始めて、パス特有の例外を必要に応じて追加していく方法が最も効率的です。ほとんどの場合は、グローバル制約のみが必要とされます。

FPGA インプリメンテーション ツールでは、特定のタイミング要件にしたって、デバイスリソースを割り当て、できる限りタイミング要件を満たそうとしますが、制約が付きすぎたり、デザイン要件以上の値が指定されていたりすると、メモリ使用率が上がったり、ツールのランタイムが増加したりします。また、制約を付けすぎたために、問題となっている制約だけでなく、ほかの制約のパフォーマンスも低下してしまうことがあります。このため、制約値は実際のデザイン要件を使用して指定するようにしてください。

このマニュアルで説明する制約の付け方は、UCF 制約構文例を使用しています。このフォーマットはデザイン要件を含んだ制約構文を目立たせるために使用していますが、一番簡単なのは、Constraints Editor ツールを使用してデザインに制約を入力する方法です。このツールでは、デザインに関するすべてのタイミング制約を管理する統一ディレクトリが提供されます。また、デザイン要件からタイミング制約を作成するための援助ツールでもあります。

## 入力タイミング制約

このセクションでは、入力タイミング制約の指定方法について説明します。

## 概要

入力タイミング制約は、FPGA の外部ピンからデータをキャプチャする内部レジスタまでのデータパスに適用されます。入力タイミングを指定するための制約は、OFFSET IN 制約です。入力タイミング要件を指定する最適な方法は、そのインターフェイスのタイプ (ソース/システム同期) とデータレート (SDR/DDR) によって異なります。

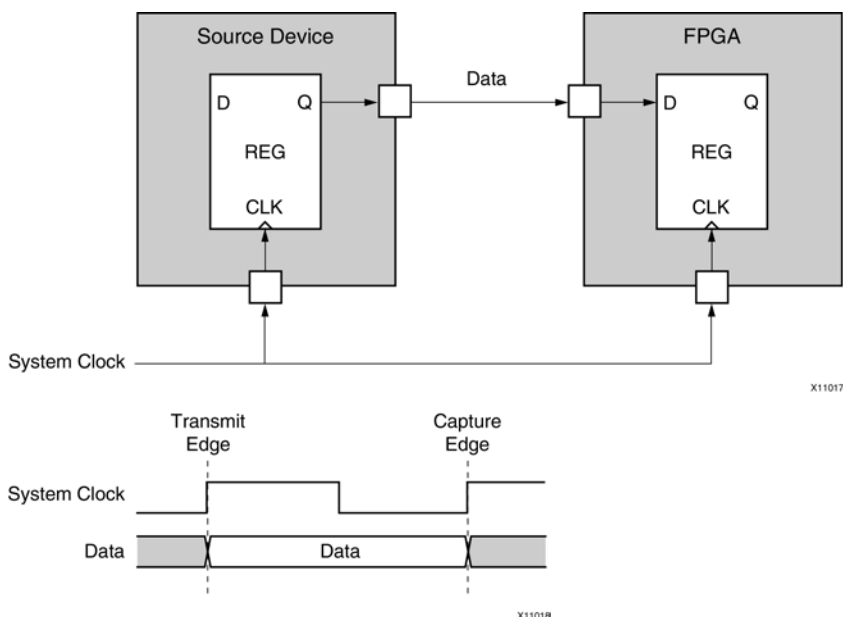
OFFSET IN 制約では、データと、FPGA のピンでそのデータをキャプチャするクロック エッジの関係を定義します。タイミング解析ツールでは、OFFSET IN 制約を解析する際に、クロックとデータの遅延に影響する内部要因がすべて自動的に考慮されます。これらの要因には、クロックの周波数と位相変換、クロック誤差、データ遅延調整などが含まれます。この自動調整に加え、設計者によって、インターフェイス クロックに設定された PERIOD 制約にさらに入力クロック誤差が追加されることもあります。PERIOD 制約の詳細と INPUT\_JITTER の追加については、「[PERIOD](#)」を参照してください。

OFFSET IN 制約は、1 つの入力クロックに付けられます。デフォルトでは、FPGA の入力パッドからそのデータをキャプチャする内部レジスタまでのパスすべてに適用され、指定した OFFSET IN クロックでトリガされます。この OFFSET IN 制約の適用方法は、「グローバル」な適用方法と呼ばれ、入力タイミングを指定する最適な方法です。

## システム同期入力

### 概要

システム同期インターフェイスとは、データの送信および受信のどちらにも使用される共通のシステム クロックのインターフェイスのことです。次の図は、SDR タイミングと関連付けられたシステム同期インターフェイスを単純化したものです。



このインターフェイスではよくあるシステム クロックが使用されているので、ボードトレース遅延やスキューによってインターフェイスの動作周波数が制限されます。また、周波数が低いと、システム同期入力インターフェイスが通常はシングル データ レート (SDR) アプリケーションになります。このシステム同期 SDR アプリケーションの例では、データがクロックの立ち上がりエッジでソース デバイスから送信され、次の立ち上がりエッジで FPGA にキャプチャされます。

### 手法

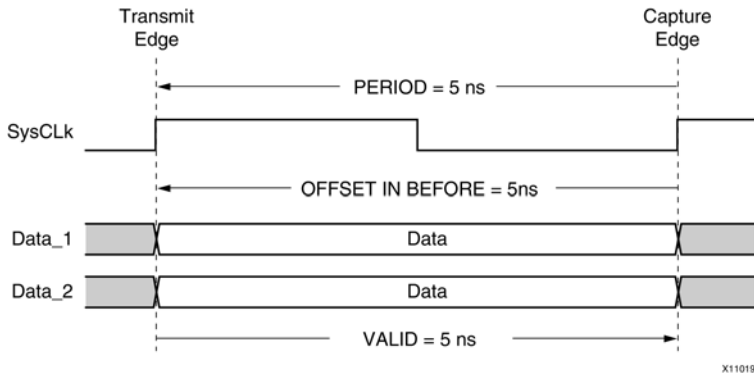
グローバル OFFSET IN 制約は、システム同期インターフェイスの入力タイミングを指定するために使用すると最も効率的です。この手法では、1 つの OFFSET IN 制約を各システム同期の入力インターフェイス クロックに対して定義します。この 1 つの制約で、レジスタにキャプチャされた (指定入力クロックでトリガされる) 入力データビットすべてのパスがカバーされます。

入力タイミングを指定するプロセスは、次のとおりです。

1. このインターフェイスに接続される入力クロックの PERIOD 制約を指定します。
2. インターフェイスのグローバル OFFSET IN 制約を定義します。

### 例

次の例は、理想的なシステム同期の SDR インターフェイスのタイミング図を示しています。このインターフェイスのクロック周期は 5ns で、バスの両方のビットのデータは全周期で有効なままになります。



グローバル OFFSET IN 制約は、次のように定義します。

**OFFSET = IN value VALID value BEFORE clock;**

OFFSET IN 制約では、OFFSET=IN value でデータが最初に有効になってからクロック エッジがキャプチャされるまでの時間を定義します。このシステム同期の例では、データはクロック エッジがキャプチャされる 5ns 前に有効になります。また、OFFSET IN 制約では、VALID value でデータが有効な状態に保たれる時間を定義します。この例では、データは 5ns 間有効のままになります。完全な OFFSET IN 指定 (PERIOD 制約も使用) は、次のようになります。

**NET "SysClk" TNM\_NET = "SysClk";**

**TIMESPEC "TS\_SysClk" = PERIOD "SysClk" 5 ns HIGH 50%;**

**OFFSET = IN 5 ns VALID 5 ns BEFORE "SysClk";**

このグローバル制約は、data1 と data2 バスのデータビット両方に適用されます。

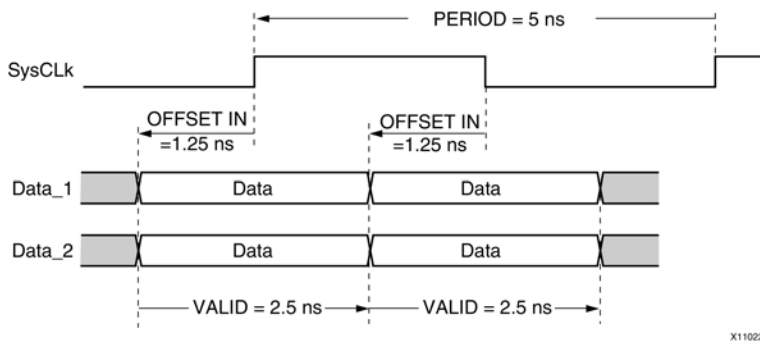
## ソース同期入力

### 概要

ソース同期インターフェイスとは、クロックが再生成されて、ソース デバイスからデータと共に転送されるインターフェイスのことです。このクロックは、この後 FPGA でデータをキャプチャするために使用されます。次の図は、DDR タイミングと関連付けられたソース同期インターフェイスを単純化したものです。







DDR の場合、グローバル OFFSET IN 制約は、次のように定義します。

**OFFSET = IN value VALID value BEFORE clock RISING;**

**OFFSET = IN value VALID value BEFORE clock FALLING;**

OFFSET IN 制約では、OFFSET=IN value でデータが最初に有効になってからクロック エッジがキャプチャされるまでの時間を定義します。このソース同期の例では、立ち上がりデータはクロックの立ち上がりエッジがキャプチャされる 1.25ns 前に有効になり、立ち下がりデータはクロックの立ち下がりエッジがキャプチャされる 1.25ns 前に有効になります。また、OFFSET IN 制約では、VALID value でデータが有効な状態に保たれる時間を定義します。この例では、立ち上がりデータも立ち下がりデータも 2.5ns 間有効のままになります。完全な OFFSET IN 指定 (PERIOD 制約も使用) は、次のようになります。

**NET "SysClk" TNM\_NET = "SysClk";**

**TIMESPEC "TS\_SysClk" = PERIOD "SysClk" 5 ns HIGH 50%;**

**OFFSET = IN 1.25 ns VALID 2.5 ns BEFORE "SysClk" RISING;**

**OFFSET = IN 1.25 ns VALID 2.5 ns BEFORE "SysClk" FALLING;**

これらのグローバル制約は、data1 と data2 バスのデータ ビット両方に適用されます。

## Register-to-Register タイミング制約

このセクションでは、レジスタ間の同期パス タイミング要件の指定方法について説明します。register-to-register の制約は、内部レジスタ間の同期データ パスに適用されます。

### はじめに

PERIOD 制約は、クロックドメインのタイミングを定義するために使用します。この制約は 1 つのクロックドメイン内のパスを解析するだけでなく、関連するクロックドメイン間のすべてのパスも解析します。また、PERIOD 制約を使用すると、自動的にすべての周波数、位相、ドメイン間の誤差が解析中に考慮されます。PERIOD 制約の詳細については、[「PERIOD」](#)を参照してください。

同期クロックドメインに制約を付ける方法は、次のカテゴリ別に分類されます。

- ・ 関連する DCM/PLL/MMCM ドメイン (自動)
- ・ 関連するクロックドメイン (手動)
- ・ 非同期クロックドメイン

ツールで自動的に DCM/PLL/MMCM 出力クロックの関係が作成されるようにし、外部の関連クロックの関連を手動で定義すると、すべてのクロスクロックドメインパスに最適な制約が付き、解析されます。この方法に従って PERIOD 制約を付けると、クロスクロックドメイン制約を追加する必要がなくなります。

それぞれの方法は、次に説明します。

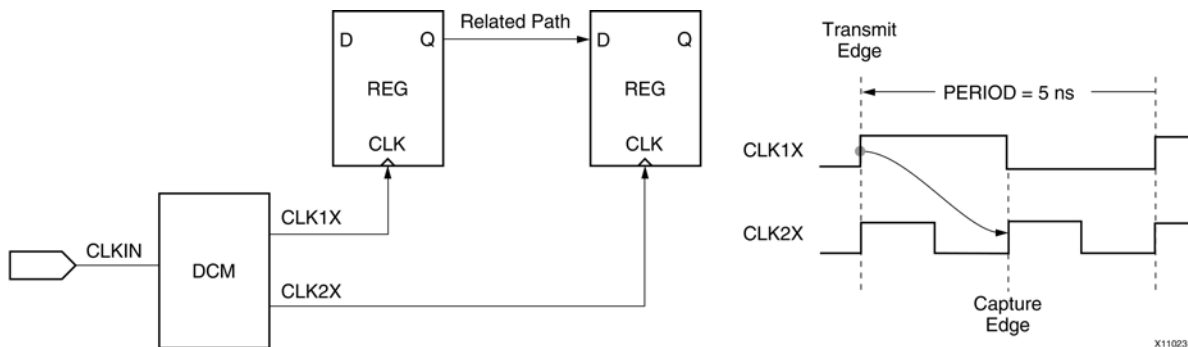
## 関連する DCM/PLL/MMCM ドメイン (自動)

### 概要

最もよくあるクロック回路では、入力クロックが DCM/PLL/MMCM に使用され、出力がデバイスの同期パスのクロックに使用されています。この場合、DCM/PLL/MMCM への入力クロックに PERIOD 制約を定義することをお勧めします。この入力クロックに PERIOD 制約を付けると、サイリンクス ツールは各 DCM/PLL/MMCM 出力クロックに対して新しい PERIOD 制約を自動的に作成し、出力クロックドメイン間のクロック関係を決定し、これらの同期ドメイン間のパスをすべて解析します。

### 例

この例では、入力クロックが DCM に入力されています。この例の回路図は、次の図のようになります。



この例の PERIOD 制約は、次のように定義されます。

```
NET "ClockName" TNM_NET = "TNM_NET_Name";
```

```
TIMESPEC "TS_name" = PERIOD "TNM_NET_Name" PeriodValue HIGH HighValue%;
```

PERIOD 制約では、PeriodValue でクロック周期の継続時間を定義します。この例の場合、DCM への入力クロックの周期は 5ns です。PERIOD 制約の HighValue は、High のクロック波形の割合をパーセントで定義します。この例の場合、波形は 50/50 のデューティサイクルで HighValue は 50% です。構文は次のようになります。

```
NET "ClkIn" TNM_NET = "ClkIn";
```

```
TIMESPEC "TS_ClkIn" = PERIOD "ClkIn" 5 ns HIGH 50%;
```

上記の入力クロックの PERIOD 制約に基づいて、DCM はその出力に対して自動的に 2 つの出力クロック制約を作成し、その 2 つのドメイン間の解析を実行します。

## 関連するクロックドメイン (手動)

### 概要

同期クロックドメイン間の関係がツールで自動的に決定されないこともあります。たとえば、関連するクロックが別々のピンで FPGA デバイスに入力される場合などがそうです。この場合、両方の入力クロックに対して別々の PERIOD 制約を作成し、クロック間の関係を手動で定義することをお勧めします。手動で関係を定義したら、この 2 つの同期ドメイン間のパスすべてが自動的にすべての周波数、位相、誤差情報などが考慮された上で解析されます。

### 手法

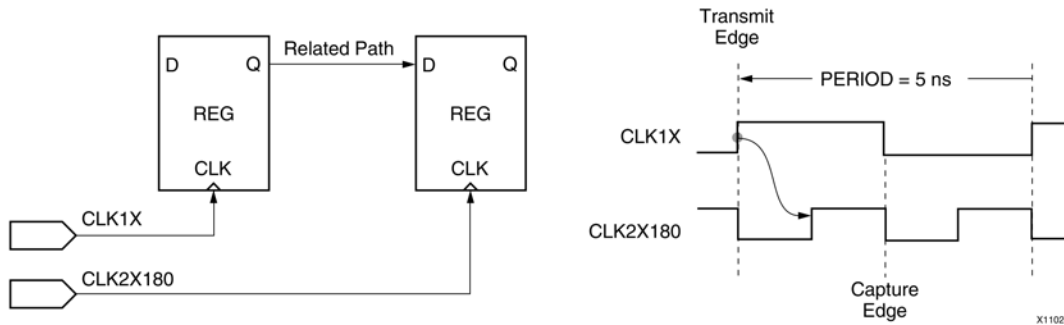
ザイリンクスの制約システムでは、複雑な手動関係を PERIOD 制約を使用してクロックドメイン間を定義することで設定できます。この手動関係には、クロック周波数と位相変換を含めることができます。このプロセスは、次のように実行します。

1. プライマリ クロックの PERIOD 制約を定義します。
2. 最初の PERIOD 制約を使用して関連クロックの PERIOD 制約をリファレンスとして定義します。

PERIOD 制約の使用方法については、「[PERIOD](#)」を参照してください。

### 例

この例では、2 つの関連クロックが別々の外部ピンから FPGA デバイスに入力されています。最初のクロック CLK1X がプライマリ クロックで、CLK2X180 が関連クロックです。この例の回路図は、次の図のようになります。



この例の PERIOD 制約は、次のように定義されます。

```
NET "PrimaryClock" TNM_NET = "TNM_Primary";
```

```
NET "RelatedClock" TNM_NET = "TNM_Related";
```

```
TIMESPEC "TS_primary" = PERIOD "TNM_Primary" PeriodValue HIGH HighValue%;
```

```
TIMESPEC "TS_related" = PERIOD "TNM_Related" TS_Primary_relation PHASE value;
```

関連する PERIOD 定義では、PERIOD の値がプライマリ クロックに関連した時間単位 (周期) として定義されます。この関係は、プライマリ クロックの TIMESPEC で記述されます。この例では、CLK2X180 が CLK1X の周波数の 2 倍で動作するため、PERIOD 関係が 1/2 になります。関連する PERIOD 定義では、PHASE 値はソース クロックの立ち上がりエッジと関連クロック間の時間差を定義します。この例では、CLK2X180 クロックが 180 度シフトされるので、その立ち上がりエッジはプライマリ エッジの立ち上がりエッジの 1.25ns 後に開始します。構文は次のようになります。

```
NET "Clk1X" TNM_NET = "Clk1X";
```

```
NET "Clk2X180" TNM_NET = "Clk2X180";
```

```
TIMESPEC "TS_Clk1X" = PERIOD "Clk1X" 5 ns;
```

```
TIMESPEC "TS_Clk2X180" = PERIOD "Clk2X180" TS_Clk1X/2 PHASE + 1.25 ns ;
```

## 非同期クロックドメイン

### 概要

非同期クロックドメインは、周波数または位相関係のないクロックの送信と受信をするドメインです。クロックが関連していないので、セットアップおよびホールド タイム解析用に最終的な関係を決定することはできません。このため、データが問題なくキャプチャされるためには、最適な非同期デザイン手法を使用することをお勧めします。ただし、必須ではありませんが、クロック パス周波数または位相関係を考慮せず、最大データ パス遅延を個別に制約する場合もあります。

### 手法

ザイリンクス制約システムでは、ソースおよびディスティネーション クロック周波数および位相関係に関係なく、最大データパス遅延に制約を付けることができます。

これは、FROM-TO 制約に DATAPATHONLY キーワードを使用して指定します。

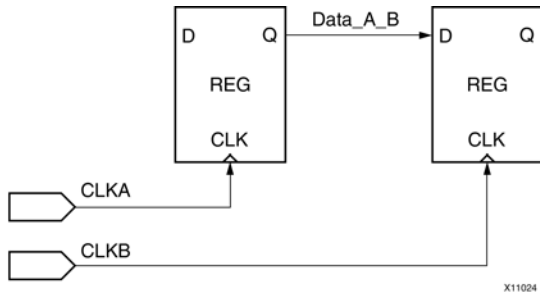
このプロセスは、次のように実行します。

1. ソースレジスタのタイム グループを定義します。
2. ディスティネーションレジスタのタイム グループを定義します。
3. 2 つのタイム グループ間に FROM-TO 制約を DATAPATHONLY キーワードを付けて使用して、ネットの最大遅延を定義します。

FROM-TO 制約に DATAPATHONLY キーワードを使用する方法については、「[FROM-TO](#)」を参照してください。

### 例

この例では、2 つの関連しないクロックが別々の外部ピンから FPGA デバイスに入力されています。最初のクロック CLKA がソースクロック、2 つ目のクロック CLKB がディスティネーションクロックです。この例の回路図は、次の図のようになります。



```
NET "CLKA" TNM_NET = FFS "GRP_A";
```

```
NET "CLKB" TNM_NET = FFS "GRP_B";
```

```
TIMESPEC TS_Example = FROM "GRP_A" TO "GRP_B" 5 ns DATAPATHONLY;
```

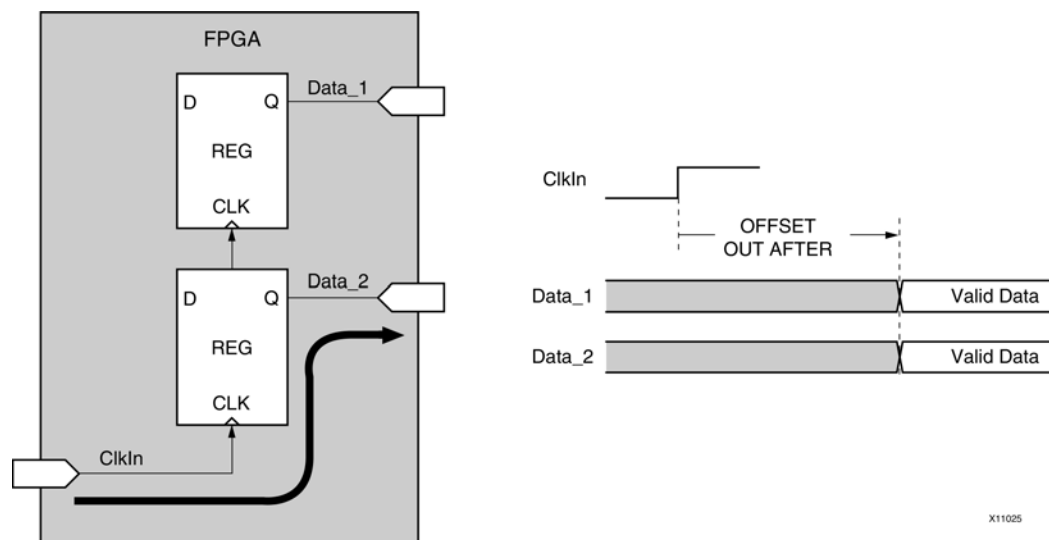
## 出力タイミング制約

このセクションでは、出力タイミング制約の指定方法について説明します。出力タイミング制約は、FPGA 内部のレジスタから FPGA の外部ピンまでのデータパスに適用されます。

### はじめに

出力タイミングを指定するための制約は、OFFSET OUT 制約です。出力タイミング要件を指定する最適な方法は、そのインターフェイスのタイプ (ソース/システム同期) とデータレート (SDR/DDR) によって異なります。

この制約は、FPGA からデータが送信される最大時間を定義します。出力遅延パスは、FPGA の入力クロックピンから、出力レジスタを通して、FPGA のデータピンまでになります。次の図は、このパスを示しています。

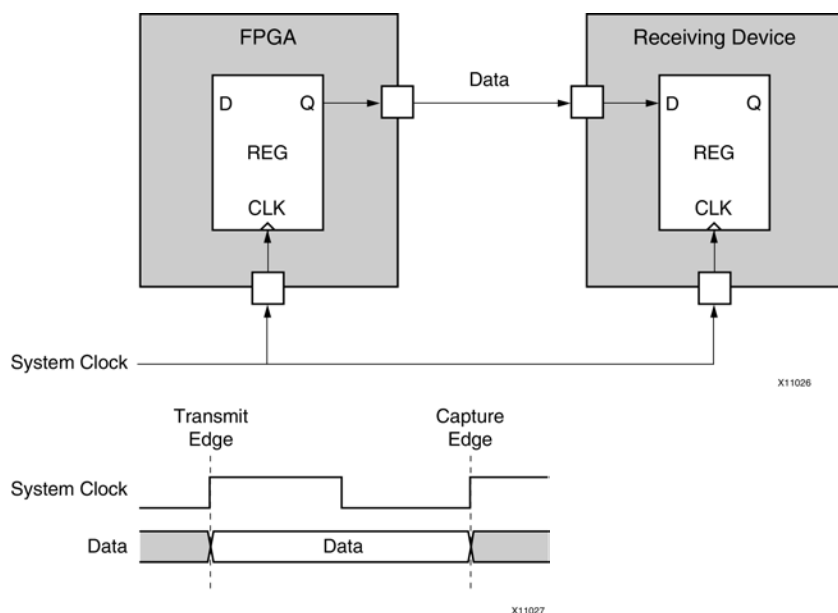


タイミング解析ツールでは、OFFSET OUT 制約を解析する際に、クロックとデータの遅延に影響する内部要因がすべて自動的に考慮されます。これらの要因には、クロックの周波数と位相変換、クロック誤差、データパス遅延調整などが含まれます。OFFSET OUT 制約の詳細については、「[OFFSET OUT](#)」を参照してください。

## システム同期出力

### 概要

システム同期出力インターフェイスとは、データの送信および受信のどちらにも使用される共通のシステムクロックのインターフェイスのことです。次の図は、SDR (シングル データ レート) タイミングと関連付けられたシステム同期出力インターフェイスを単純化したものです。



このインターフェイスではよくあるシステムクロックが使用されているので、データだけが FPGA から受信デバイスに送信されます。

### 手法

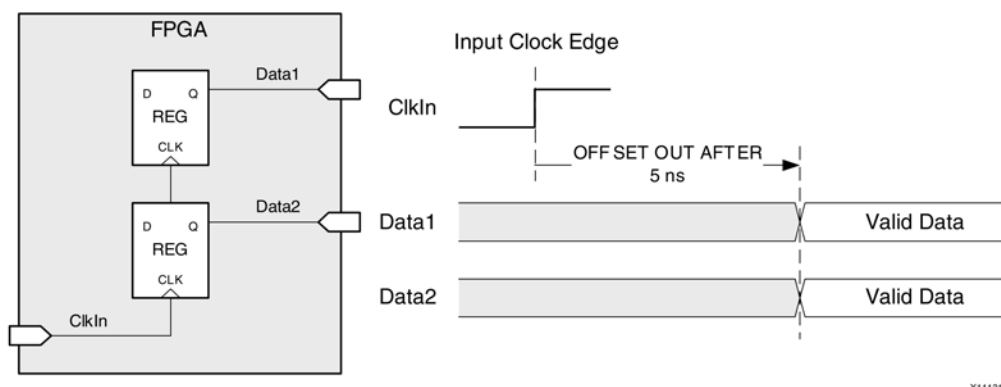
グローバル OFFSET OUT 制約は、システム同期インターフェイスの出力タイミングを指定するために使用すると最も効率的です。このグローバル手法では、1 つの OFFSET OUT 制約を各システム同期の出力インターフェイス クロックに対して定義します。この 1 つの制約で、レジスタから送信される (指定出力クロックでトリガされる) 出力データ ビットすべてのパスがカバーされます。

出力タイミングを指定するプロセスは、次のとおりです。

1. 出力クロックのタイム名 (TNM) を定義し、出力クロックでトリガされる出力レジスタすべてを含むタイムグループを作成します。
2. インターフェイスのグローバル OFFSET OUT 制約を定義します。

## 例

次の例は、システム同期の SDR 出力インターフェイスのタイミング図を示しています。この例のデータは、FPGA のピンの入力クロック エッジの後、最大 5ns で出力ピンで有効になるはずです。



システム同期インターフェイスのグローバル OFFSET OUT 制約は、次のように定義します。

**OFFSET = OUT value VALID value AFTER clock;**

OFFSET=OUT 制約では、OFFSET=OUT value で入力クロック ポートの立ち上がりエッジから FPGA のデータ出力ポートでデータが最初に有効になるまでの最大時間を定義します。このシステム同期の例では、出力データは入力クロック エッジ後少なくとも 5ns で有効になります。完全な OFFSET OUT 指定は、次のようになります。

**NET "ClkIn" TNM\_NET = "ClkIn";**

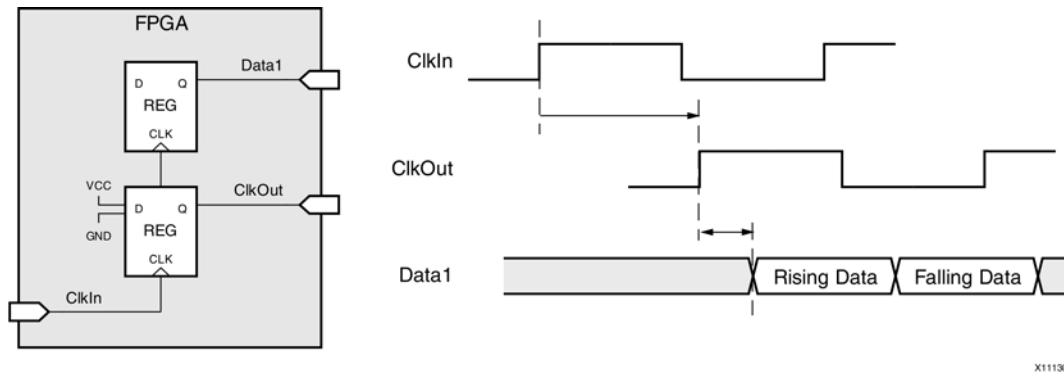
**OFFSET = OUT 5 ns AFTER "ClkIn";**

このグローバル制約は、data1 と data2 バスの出力データ ビット両方に適用されます。

## ソース同期出力

### 概要

ソース同期出力インターフェイスとは、クロックが再生成されて、FPGA からデータと共に転送されるインターフェイスのことです。次の図は、DDR タイミングと関連付けられたソース同期出力インターフェイスを単純化したものです。



再生成されたクロックはデータと共に送信されるので、このインターフェイスのパフォーマンスはシステム ノイズや生成されたクロックとデータ ビット間のスキューによって制限されます。このインターフェイスでは、入力クロック エッジから出力データが有効になるまでの時間は出力データ ビット間のスキューほど重要な問題ではないので、ほとんどの場合制約を付けなくても問題ありません。

### 手法

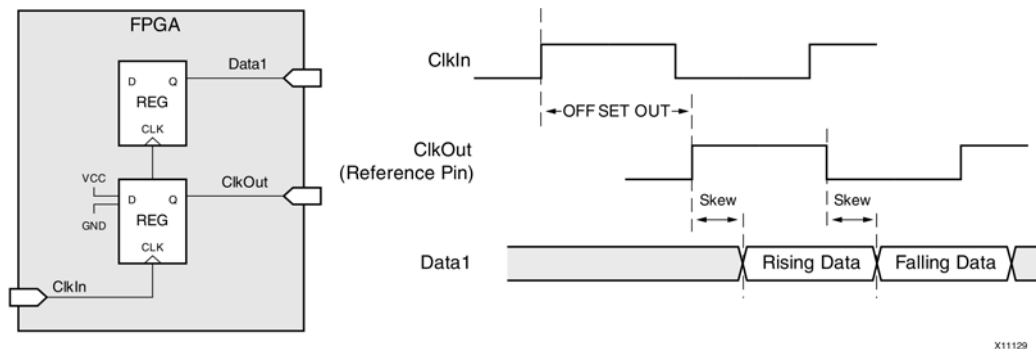
グローバル OFFSET OUT 制約は、ソース同期インターフェイスの出力タイミングを指定するために使用すると最も効率的です。DDR インターフェイスでは、1 つの OFFSET OUT 制約を出力インターフェイス クロックの各エッジに対して定義します。これらの制約では、レジスタから送信される (指定出力クロック エッジでトリガされる) 出力データ ビットすべてのパスがカバーされます。

出力タイミングを指定するプロセスは、次のとおりです。

1. 出力クロックのタイム名 (TNM) を定義し、出力クロックでトリガされる出力レジスタすべてを含むタイムグループを作成します。
2. インターフェイスの立ち上がりエッジ用にグローバル OFFSET OUT 制約を定義します。
3. インターフェイスの立ち下がりエッジ用にグローバル OFFSET OUT 制約を定義します。

### 例

次の例は、理想的なソース同期の DDR インターフェイスのタイミング図を示しています。このインターフェイス例では、クロックから出力への絶対時間は重要ではなく、再生成されたクロックと出力データ ビット間のスキューだけが必要となります。



DDR の場合、グローバル OFFSET OUT 制約は、次のように定義します。

```
OFFSET = OUT AFTERclock REFERENCE_PIN "REF_CLK" RISING;
```

```
OFFSET = OUT AFTERclock REFERENCE_PIN "REF_CLK" FALLING;
```



OFFSET=OUT 制約では、OFFSET=OUT *value* で入力クロックポートの立ち上がりエッジから FPGA のデータ出力ポートでデータが最初に有効になるまでの最大時間を定義します。値が OFFSET OUT 制約に設定されていない場合、この制約は出力パスのスキューをレポートする役割しか果たしません。この制約の REFERENCE\_PIN キーワードは、出力データピンのスキューをレポートするリファレンスポイントとして再生成した出力クロックを定義するためのものです。

立ち上がりおよび立ち下がりクロックエッジ両方の完全な OFFSET OUT 指定は、次のようになります。

```
NET "ClkIn" TNM_NET = "ClkIn";  
  
OFFSET = OUT AFTER "ClkIn" REFERENCE_PIN "ClkOut" RISING;  
OFFSET = OUT AFTER "ClkIn" REFERENCE_PIN "ClkOut" FALLING;
```

## 例外のタイミング制約

### 概要

入力、register-to-register、出力タイミング制約のグローバル定義を使用すると、デザインのほとんどのパスに問題なく制約が付きませんが、一部のパスにグローバル制約規則にあてはまらないことがあります。こういった例外は、次のパスでよくあります。

1. False パス
2. 複数サイクル パス

## False パス

### 概要

パスがデザインのタイミングパフォーマンスに影響がないとわかっている場合、解析からそれらのパスを削除した方がいいことがあります。

### 方法

タイミング解析から一部のパスを指定して削除するには、タイミング無視 (TIG) キーワードを付けて FROM-TO 制約を使用します。この方法を使用すると、ソース タイム グループのレジスタのセット、ディスティネーション タイム グループのレジスタのセットを指定し、これらのタイム グループ間のパスを解析からすべて削除するといったことが柔軟にできるようになります。

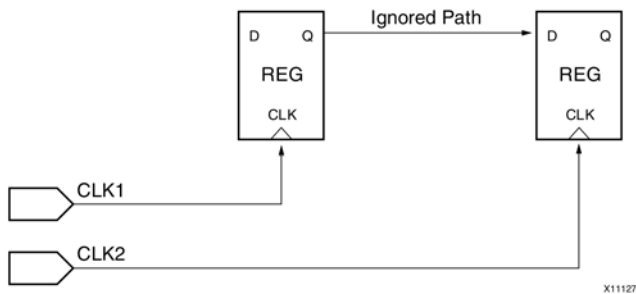
タイミング無視 (TIG) 制約は、次の方法で指定します。

1. ソース タイム グループのレジスタのセットを定義します。
2. ディスティネーション タイム グループのレジスタのセットを定義します。
3. TIG キーワードを付けて FROM-TO 制約を定義して、2 つのグループ間のパスを削除します。

### 例

次の例では、2 つのレジスタ間のパスがデザインのタイミングに影響せず、解析から削除する必要があるとします。この例の回路図は、次のようになります。





タイム グループ間にタイミング無視 (TIG) を定義するジェネリック構文は、次のようになります。

**TIMESPEC "Tsid" = FROM "SRC\_GRP" TO "DST\_GRP" TIG;**

この FROM-TO TIG 例では、SRC\_GRP でパストレースが始まるソース レジスタのセットを定義し、DST\_GRP でパストレースが終了するディスティネーション レジスタのセットが定義されています。SRC\_GRP で始まり DST\_GRP で終了するパスはすべて無視されます。

この例の構文は、次のようになります。

**NET "CLK1" TNM\_NET = FFS "GRP\_1";**

**NET "CLK2" TNM\_NET = FFS "GRP\_2";**

**TIMESPEC TS\_Example = FROM "GRP\_1" TO "GRP\_2" TIG;**

## 複数サイクル パス

### 概要

複数サイクル パスでは、ソースからディスティネーション レジスタへのデータが PERIOD で定義されたクロック周波数よりも小さいレートで転送されます。これは、レジスタが共通するクロック イネーブル信号と一緒にゲートを介している場合によくあります。複数サイクル パスを定義すると、これらのレジスタのタイミング制約はデフォルトの PERIOD 制約のまま、インプリメンテーション ツールでこれらのパスのインプリメンテーションに適切な優先順位が付けられるようになります。

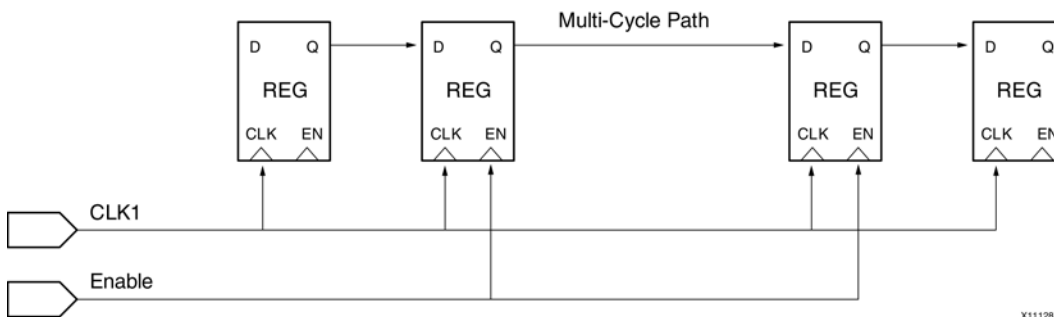
### 方法

FROM-TO 複数サイクル制約は、次の方法で指定します。

1. 共通のクロックドメインに PERIOD 制約を定義します。
2. 共通のクロック イネーブル信号に基づいてレジスタのセットを定義します。
3. 新しいタイミング要件を記述した FROM-TO 複数サイクル制約を定義します。

### 例

次の例では、2 つのレジスタ間のパスに共通のクロック イネーブル信号からのクロックが使用されているとします。クロック イネーブル信号は、リファレンス クロックの半分のレートでトグルされます。この例の回路図は、次のようになります。



タイム グループ間に複数サイクル パスを定義するジェネリック構文は、次のようになります。

```
TIMESPEC "Tsid" = FROM "MC_GRP" TO "MC_GRP" value;
```

この FROM-TO 複数サイクルの例では、MC\_GRP で共通のクロック イネーブル信号で駆動されるレジスタのセットが定義されます。MC\_GRP で始まり MC\_GRP で終わるパスにはすべて複数サイクル タイミング要件が設定されており、MC\_GRP に入力し出力するパスには適切な PERIOD 制約が付けられて解析されます。

この例の構文は、次のようになります。

```
NET "CLK1" TNM_NET = "CLK1";
```

```
TIMESPEC "TS_CLK1" = PERIOD "CLK1" 5 ns HIGH 50%;
```

```
ET "Enable" TNM_NET = FFS "MC_GRP";
```

```
TIMESPEC TS_Example = FROM "MC_GRP" TO "MC_GRP" TS_CLK1*2;
```

# ザイリンクス制約

この章では、FPGA および CPLD デバイスで利用できる制約について、それぞれのアーキテクチャ サポート、適用可能エレメント、説明、適用ルール、構文例、必要であれば特定の制約情報などを説明します。この章に含まれるセクションは、次のとおりです。

- ・ [制約に関する情報](#)
- ・ [ザイリンクス制約のリスト \(アルファベット順\)](#)

## 制約に関する情報

各制約のセクションには、次の情報が記載されています。

- ・ アーキテクチャ サポート  
どのデバイスで制約を使用できるかを示します。
- ・ 適用可能エレメント  
制約を適用するエレメントを示します。
- ・ 説明  
使用法およびビヘイビアなど制約について説明します。
- ・ 適用ルール  
どのように制約が適用されるかを示します。
- ・ 構文例  
構文例をツールまたは手法ごとに示します。例で網羅されていないツールや手法には、制約に使用できないものがあります。次は使用可能なツールと手法を示しています。
- ・ その他の情報  
制約によっては、追加の情報が記載されています。

回路図	Project Navigator
Verilog	VHDL
NCF	UCF
XCF	Constraints Editor
PCF	PinAhead
PACE	FPGA Editor

## ザイリンクス制約のリスト (アルファベット順)

ザイリンクス制約は次のとおりです。

- ・ AREA\_GROUP
- ・ ASYNC\_REG
- ・ BEL
- ・ BLKNM
- ・ BUFG (CPLD の場合)
- ・ CLOCK\_DEDICATED\_ROUTE
- ・ COLLAPSE
- ・ COMPGRP
- ・ CONFIG\_MODE
- ・ COOL\_CLK (CoolCLOCK)
- ・ DATA\_GATE
- ・ DEFAULT
- ・ DCI\_CASCADE
- ・ DCI\_VALUE
- ・ DIRECTED\_ROUTING
- ・ DISABLE
- ・ DRIVE
- ・ DROP\_SPEC
- ・ ENABLE
- ・ ENABLE\_SUSPEND
- ・ FAST
- ・ FEEDBACK
- ・ FILE
- ・ FLOAT
- ・ FROM-THRU-TO
- ・ FROM-TO
- ・ HBLKNM
- ・ HLUTNM
- ・ HU\_SET
- ・ IBUF\_DELAY\_VALUE
- ・ IFD\_DELAY\_VALUE
- ・ INREG
- ・ IOB
- ・ IOBDELAY
- ・ IODELAY\_GROUP
- ・ IOSTANDARD

- ・ KEEP
- ・ KEEPER
- ・ KEEP\_HIERARCHY
- ・ LOC
- ・ LOCATE
- ・ LOCK\_PINS
- ・ LUTNM
- ・ MAP
- ・ MIODELAY\_GROUP
- ・ MAXDELAY
- ・ MAX\_FANOUT
- ・ MAXPT
- ・ MAXSKEW
- ・ NODELAY
- ・ NOREDUCE
- ・ OFFSET IN
- ・ OFFSET OUT
- ・ OPEN\_DRAIN
- ・ OPT\_EFFORT
- ・ OPTIMIZE
- ・ PERIOD
- ・ PIN
- ・ POST\_CRC
- ・ POST\_CRC\_ACTION
- ・ POST\_CRC\_FREQ
- ・ POST\_CRC\_SIGNAL
- ・ PRIORITY
- ・ PROHIBIT
- ・ PULLDOWN
- ・ PULLUP
- ・ PWR\_MODE
- ・ REG
- ・ RLOC
- ・ RLOC\_ORIGIN
- ・ RLOC\_RANGE
- ・ SAVE NET FLAG
- ・ SCHMITT\_TRIGGER
- ・ SLEW

- ・ SLOW
- ・ STEPPING
- ・ SUSPEND
- ・ SYSTEM\_JITTER
- ・ TEMPERATURE
- ・ TIG
- ・ TIMEGRP
- ・ TIMESPEC
- ・ TNM
- ・ TNM\_NET
- ・ TPSYNC
- ・ TPTHU
- ・ TSidentifier
- ・ U\_SET
- ・ USE\_RLOC
- ・ VCCAUX
- ・ VOLTAGE
- ・ VREF
- ・ WIREAND
- ・ XBLKNM

## AREA\_GROUP

AREA\_GROUP は、マップ、パック、配置配線のためにデザインを物理的な領域に分割できるようにするインプリメンテーション制約で、

属性値には、MAP によりパックされる論理ブロックのグループや、PAR により指定範囲に配置される論理ブロックのグループを指定します。階層ブロックに設定した場合、その階層ブロック内のすべての下位ブロックが同じグループに割り当てられます。

AREA\_GROUP 制約を設定すると、この制約に関連したさまざまな制約を追加で使用して、インプリメンテーションを制御できます。詳細については、この制約の「構文」セクションを参照してください。

## アーキテクチャ サポート

AREA\_GROUP 制約は、FPGA デバイスにのみ適用されます。

## 適用可能エレメント

- ・ ロジック ブロック
- ・ タイミング グループ

詳細は、次の「タイミング グループによる定義」を参照してください。

## 適用ルール

次のルールが AREA\_GROUP に適用されます。

- ・ デザイン エLEMENT に設定すると、そのデザイン エLEMENT の階層にあるすべての適用可能ELEMENT に適用されます。
- ・ ネット、信号、またはピンには設定できません。

## 構文

エリア グループを定義する基本的な UCF 構文は次のとおりです。 **INST "X" AREA\_GROUP=groupname ;**

エリア グループに属性を指定した構文は次のとおりです。

**AREA\_GROUP "groupname" RANGE=range;**

or

**AREA\_GROUP "groupname" COMPRESSION=percent;**

or

**AREA\_GROUP "groupname" GROUP={OPEN|CLOSED};**

or

**AREA\_GROUP "groupname" PLACE={OPEN|CLOSED};**

この場合、それぞれ次を表します。

*groupname* は、グループを定義するため区画に割り当てられた名前です。

次に、AREA\_GROUP に指定する属性について説明します。

## RANGE

LOC 制約を使用した範囲の定義と同様で、AREA\_GROUP 内のロジックを配置するデバイスリソースの範囲を定義します。

FPGA デバイスの場合、*RANGE* は次のようになります。

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

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

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

すべての FPGA デバイスで、SLICE がサポートされます。AREA\_GROUP にブロック RAM および SLICE の両方が含まれている場合、一方を BRAM に、もう一方を SLICE というふうに、2 つの異なる AREA\_GROUP RANGE を指定できます。

FPGA の位置はすべて X 座標または Y 座標で指定されます。ワイルドカード文字は X 座標または Y 座標のいずれかに対して使用できます。

RANGE の値は、CLOCK REGION または複数の CLOCK REGION のセットとしても指定できます。この構文は、AREA\_GROUP 制約で使用されるすべての INIT タイプでサポートされます。

FPGA デバイスの場合、AREA\_GROUP は次のようなさまざまなクロック領域でサポートされます。

1 つの領域 :

**AREA\_GROUP "groupname" RANGE=CLOCKREGION\_X#Y#;**

長方形を形成するクロック領域 :

**AREA\_GROUP "group\_name" RANGE=CLOCKREGION\_X#Y#:CLOCKREGION\_X#Y#;**

クロック領域のリスト :

**AREA\_GROUP "groupname" RANGE=CLOCKREGION\_X#Y#,CLOCKREGION\_X#Y#,...,;**

X# および Y# の値は、デバイスによって異なります。



## COMPRESSION

COMPRESSION は、AREA\_GROUP の圧縮係数を定義します。原則として、0 から 100 までの値を使用できます。グループに範囲を指定しない場合、使用できる値は 0 (圧縮なし) および 1 (最大圧縮) のみです。マップは RANGE を使用して AREA\_GROUP 内の CLB 数を計算し、指定されたパーセンテージでロジックを圧縮します。BRAM または DSP ブロック/乗算器には、圧縮は適用されません。

圧縮係数はマップの **-c** オプションに似ていますが、この係数はデザイン全体に有効ではなく、AREA\_GROUP に有効です。AREA\_GROUP の圧縮とマップの **-c** オプションの関係は次のとおりです。

- ・ 圧縮係数が定義されているエリア グループは、**-c** オプションの影響を受けません (圧縮係数が定義されたエリア グループの場合、その AREA\_GROUP に含まれないロジックはグループ化されたロジックにマージされません)。
- ・ 圧縮係数が定義されていないエリア グループは、**-c** オプションの影響を受けます。マップは、圧縮係数が定義されていないエリア グループの一部であるロジックを、グループ化されていないロジックにマージしようとします。
- ・ 2 つの異なるエリア グループのロジックをマージすることはありません。
- ・ **-c** マップ オプションを使用しても、エリア グループ内のスライスの圧縮を強制的に実行できません。

マップ レポート (MRP) には、処理された AREA\_GROUP の要約が記載されます。

エリア グループの一部となっているシンボルに LOC 制約を設定すると、そのシンボルはマップでエリア グループから削除され、LOC 制約が処理されます。

AREA\_GROUP に含まれないロジックは、エリア グループに含まれるロジック (スライスを形成するロジックにパックまたはマージされるロジック) の領域に移動する場合があります。

AREA\_GROUP の COMPRESSION は、マップでタイミングドリブン パックと配置 (**-timing** オプション) を使用している場合は使用できません。

AREA\_GROUP の COMPRESSION は、Virtex®-5 ではサポートされません。

## GROUP

スライスなどロジックの物理コンポーネントへのパックを制御します。

CLOSED

エリア グループ外のロジックがエリア グループ内のロジックと共にパックされません。

OPEN

エリア グループ外のロジックがエリア グループ内のロジックと共にパックされます。

デフォルト値は、GROUP=OPEN です。

## PLACE

エリア グループの範囲内でのリソースの割り当てを制御します。

CLOSED

エリア グループ以外のロジックがエリア グループの範囲内に配置されません。

OPEN

エリア グループ以外のロジックがエリア グループの範囲内に配置されます。

デフォルト値は、PLACE=OPEN です。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ AREA\_GROUP=*groupname* を有効なインスタンスに設定します。
- ・ RANGE=*range* を CONFIG シンボルに設定します。
- ・ COMPRESSION=*percent* を CONFIG シンボルに設定します。
- ・ GROUP={**OPEN**|**CLOSED**} を CONFIG シンボルに設定します。
- ・ PLACE={**OPEN**|**CLOSED**} を CONFIG シンボルに設定します。
- ・ CONFIG シンボルを設定します。TRUE、PLACE、GROUP の値は、CLOSED に指定する必要があります。
- ・ 属性名 : AREA\_GROUP、RANGE *range*、COMPRESSION *percent*、GROUP={**OPEN** |**CLOSED**}、PLACE={**OPEN** |**CLOSED**}
- ・ 属性値 : *groupname*, *range* , *percent*, GROUP={**OPEN** |**CLOSED**}, PLACE={**OPEN** |**CLOSED**}

## UCF および NCF 構文

次の例では、state\_machine\_X のすべての論理ブロックをエリア グループ group1 に割り当て、ロジックを SLICE の行 1、列 1 と SLICE の行 10、列 10 の間の物理的なエリアに配置します。

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

## PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## Constraints Editor の構文

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Miscellaneous] の下の [Time Group Based Area Group] をダブルクリックし、ダイアログ ボックスにアクセスします。

## タイミング グループによる定義

タイミング グループを基にしたエリア グループを作成するには、次の UCF/NCF 構文を使用します。

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

- ・ *timing\_group\_name* は、定義済みのタイミング グループ名です。
- ・ *area\_group\_name* は、そのタイミング グループを基にして定義するエリア グループ名です。

これは、タイミング グループの各メンバーに *area\_group\_name* を手動で割り当てることと同じです。この構文で定義されたエリア グループ名は、ほかのエリア グループ名と同様、RANGE 制約で使用できます。

エリア グループを定義する場合、通常はタイミング グループ (*timing\_group\_name*) 名を TNM\_NET グループとして使用し、クロックのロードやその他の制御ネットを基にしてエリア グループを作成します。タイミング グループを基にしてエリア グループを定義すると、クロック領域よりも多くのクロックを使用するデバイスで、異なるクロックドメインを使用したデザインを効率的に配置できます。

TNM グループ名または TIMEGRP で定義されたユーザー グループ名も指定できます。ただし、TIMEGRP で使用されるエッジ識別子は、エリア グループのメンバーを識別する際に無視されます。いずれの場合にも、エリア グループのメンバーは、タイミング グループがターゲット エlement へ伝搬された後で識別されます。

タイミング グループには同期 Element およびパッドのみが含まれているので、タイミング グループを基にして定義されたエリア グループにもこれらの Element のみが含まれます。フリップフロップまたはラッチだけを含むタイミング グループを基にエリア グループを定義した場合、そのエリア内でグループにまとめられていないロジックも使用できる場合にのみ、RANGE を設定できます。したがって、COMPRESSION はそのようなグループに対して設定できません。

TNM\_NET が PERIOD 仕様により使用され、DCM、PLL または MMCM にトレースされる場合、新しい TNM\_NET グループと PERIOD 仕様は DCM、PLL または MMCM 出力で作成されます。エリア グループの定義にオリジナルの TNM\_NET を使用し、DCM、PLL、または MMCM で複数のクロック タップを使用した場合、エリア グループはクロック タップごとに異なるグループに分割されます。

たとえば、次のような UCF 構文があるとします。

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

ネット clk が DCM、PLL、MMCM へトレースされた場合、新しいグループと PERIOD 仕様が各クロック タップで作成されます。同様に、各クロック タップで新しいエリア グループが作成され、クロック タップ名を示す接尾辞が付けられます。CLK0 または CLK2X タップが使用された場合、エリア グループ clock\_area\_CLK0 および clock\_area\_CLK2X が自動的に定義されます。

このようにしてエリア グループの定義が分割されると、NGDBuild により新しいグループ名を示すメッセージが表示されます。RANGE で使用するのには、以前に指定されていたグループ名ではなく、この新しいグループ名です。

## エリア グループからの定義

エリア グループを基にしたエリア グループを作成するには、次の UCF/NCF 構文を使用します。

```
AREAGRP timing_group_name AREA_GROUP = area_group_name ;
```

この場合、それぞれ次を表します。

- ・ *area\_group\_name* は、定義済みのタイミング グループ名です。
- ・ *area\_group\_name* は、そのタイミング グループを基にして定義するエリア グループ名です。

## ASYNCR\_REG

ASYNCR\_REG タイミング制約は、非同期クロックをソースとしたデータのビヘイビアをシミュレーション用に向上します。つまり、タイミング シミュレーション中に X が適用されないようにします。タイミング違反があった場合、不定値とはならず、以前の値が出力で保持されます。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能 Element

ASYNCR\_REG 制約は、レジスタおよびラッチにのみ適用できます。非同期入力 (D 入力または CE 入力) 付きのレジスタおよびラッチのみに設定できます。

## 適用ルール

設定されたレジスタまたはラッチに適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

#### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute ASYNC_REG : string;
```

Specify the VHDL constraint as follows:

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

#### Verilog 構文

この制約はインスタンシエーションの直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

#### UCF および NCF 構文

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

デフォルトは FALSE です (制約が適用されません)。ブール属性値を使用しない場合、TRUE と見なされます。

### Constraints Editor からの設定

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Miscellaneous] の下の [Asynchronous Registers] をダブルクリックし、ダイアログ ボックスにアクセスします。

## BEL

BEL は、高度な配置制約で、論理シンボルを IOB またはスライス 内の BEL サイトにロックします。BEL は、コンポーネントレベルに指定する [ロケーション \(LOC\)](#) 制約とは異なります。コンポーネントには、SLICE、BRAM、ILOGIC、OLOGIC、IOB があります。BEL 制約では、使用されるコンポーネントの特定の BEL サイトまで指定できます。たとえば、これは特定の LUT または FF を SLICE 内で使用されるように指定するために使用できます。BEL 制約には、常に LOC または RLOC 属性を使用する必要があります。

BEL 制約を IOB に設定すると、MAP によりレジスタが IOB コンポーネントにパックされないため、**-pr** オプションなどほかの機能を使用してパックする必要があります。レジスタが IOB にパックされると、BEL 制約により IOB 内で適正に配置されます。

## アーキテクチャ サポート

BEL 制約は、サポートされる FPGA アーキテクチャすべてに適用されます。

## 適用可能エレメント

レジスタ	ラッチ
LUT	SRL
LUTRAM	
RAMB18	

## 適用ルール

BEL 制約は有効な LOC または RLOC の付いたインスタンスにのみ付けることができます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : BEL
- ・ 属性値 : F、G、FFA、FFB、FFC、FFD、FFX、FFY、XORF、XORG、A6LUT、B6LUT、C6LUT、D6LUT、A5LUT、B5LUT、C5LUT、D5LUT

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute bel : string;
```

VHDL 制約を次のように指定します。

```
attribute bel of {component_name} label_name :  
{component} label is  
"F|G|FFA|FFB|FFC|FFD|FFX|FFY|XORF|XORG|A6LUT|B6LUT|C6LUT|D6LUT|A5LUT|B5LUT|C5LUT|D5LUT";
```

BEL 値の詳細は、この制約の「UCF および NCF 構文」を参照してください。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(* BEL = "F|G|FFA|FFB|FFC|FFD|FFX|FFY|XORF|XORG|A6LUT|B6LUT|C6LUT|D6LUT|A5LUT|B5LUT|C5LUT|D5LUT" *)
```

BEL 値の詳細は、この制約の「UCF および NCF 構文」を参照してください。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

### UCF および NCF 構文

構文は次のとおりです。

```
INST "instance_name" BEL={F | G | FFA | FFB | FFC | FFD | FFX | FFY | XORF | XORG | A6LUT | B6LUT | C6LUT |
D6LUT | A5LUT | B5LUT | C5LUT | D5LUT};
```

この場合、次を表します。

- ・ F、G、A6LUT、B6LUT、C6LUT、D6LUT、A5LUT、B5LUT、C5LUT、D5LUT は、スライスにある特定の LUT、SRL16、分散 RAM コンポーネントを指します。
- ・ FFX、FFY、FFA、FFB、FFC、FFD : スライス内のフリップフロップやラッチなどのエレメントです。
- ・ XORF および XORG は、スライスにある XORCY エレメントを指します。

RAMB BEL インスタンスの構文は、次のようになります。

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

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

例 :

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

次の文は、スライス内の FFX サイトに xyzyz をロックするよう指定します。

```
INST "xyzyz" BEL=FFX;
```

## PlanAhead からの設定

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## BLKNM

BLKNM は、高度なマップ制約です。BLKNM は、ブロック名を適切なプリミティブおよびロジック エレメントに割り当てます。同じ BLKNM を複数のインスタンスに割り当てた場合、ソフトウェアはそれらを同じブロックにマップしようと試みます。逆に、異なる BLKNM 名を持つ 2 つのシンボルが同じブロックにマップされることはありません。複数の類似した BLKNM 制約を、1 つのブロック内に収まらない複数のインスタンスに設定すると、エラーが発生します。

FMAP に同じ BLKNM を複数設定すると、関連するファンクション ジェネレータが 1 つのスライスにまとめられます。BLKNM を使用すると、スライスをデバイス上の物理的な位置に制約することなく、スライスを分割できます。LOC 制約と同様、BLKNM はデザイン内で指定します。階層パスが接頭辞として割り当てられないため、デザイン内でそれぞれの SLICE に固有の BLKNM を設定する必要があります。階層ブロック名の割り当てについては、「[階層ブロック名 \(HBLKNM\)](#)」を参照してください。

BLKNM を使用すると、別の BLKNM が指定されているエレメントを除き、すべてのエレメントを同じ物理的コンポーネントにマップできます。BLKNM が指定されていないエレメントは、BLKNM が指定されているエレメントとともにパックできます。同じ XBLKNM を設定したエレメントだけを 1 つの物理コンポーネントにマップする場合は、「[XBLKNM](#)」を参照してください。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

BLKNM 制約は、次のエレメントを 1 つまたは複数使用して設定できます。すべてのデバイス ファミリーでこれらのエレメントがすべてサポートされているわけではありません。どのデバイス ファミリーにどのエレメントを使用できるかは、ライブラリ ガイドを参照してください。詳細は、[データシート](#)を参照してください。

- ・ フリップフロップおよびラッチ プリミティブ
- ・ すべての I/O エレメントまたはパッド
- ・ FMAP
- ・ ROM プリミティブ
- ・ RAMS および RAMD プリミティブ
- ・ キャリー ロジック プリミティブ
- ・ ブロック RAM

BLKNM は、UCF ファイル内のパッド コンポーネントに接続されているネットにも設定できます。ネットに設定した制約は、NGDBuild により NGD ファイル内のパッド インスタンスに挿入され、マップで処理されます。次の構文を使用してください。

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

## 適用ルール

デザイン エレメントに設定すると、そのデザイン エレメントの階層にあるすべての適用可能エレメントに適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : BLKNM
- ・ 属性値 : *block\_name*

### VHDL 構文

次の構文を使って VHDL 制約を宣言します。

```
attribute blknm : string;
```

Specify the VHDL constraint as follows:

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

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、[「Verilog」](#)を参照してください。



## UCF および NCF 構文

UCF 構文は次のとおりです。

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

この場合、それぞれ次を表します。

*block\_name* は、その種類のシンボルで有効なブロック名です。

階層ブロック名の割り当てについては、「[階層ブロック名 \(HBLKNM\)](#)」を参照してください。

次の文は、エレメント block1 のインスタンスをブロック U1358 に割り当てます。

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

## XCF 構文

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

```
BEGIN MODEL "entity_name"
```

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

```
END;
```

## BUFG

BUFG は、高度なフィッタ制約かつ合成制約です。入力バッファまたは入力パッド ネットに BUFG 属性を適用すると、指定した信号がグローバル ネットにマップされます。内部ネットに設定した場合、指定された信号は直接グローバル ネットに配線されるか、グローバル ネットを駆動するためグローバル制御ピンに接続されます。

## アーキテクチャ サポート

この制約は、CPLD デバイスにのみ適用できます。

## 適用可能エレメント

CLK、OE、SR、DATA\_GATE ピンを駆動する入力バッファ (IBUF)、入力パッド ネット、または内部ネット

## 適用ルール

ネットや信号に設定されている場合、特別な適用ルールはありません。デザイン エレメントに設定すると、そのデザイン エレメントの階層にあるすべての適用可能エレメントに適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。



## 回路図

- ・ IBUF 入力に接続されている入力パッドの IBUF インスタンスに設定します。
- ・ 属性名 : BUFG
- ・ 属性値 : CLK、OE、SR、DATA\_GATE
- ・ BUFG=CLK : グローバル クロック (GCK) に割り当てます。
- ・ BUFG=OE : グローバル トライステート制御 (GTS) に割り当てます。
- ・ BUFG=SR : グローバル セット/リセット制御 (GSR) に割り当てます。
- ・ BUFG=DATA\_GATE : DataGate ラッチ イネーブル制御に割り当てます。

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute BUFG: string;
```

VHDL 制約を次のように指定します。

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

BUFG=CLK : グローバル クロック (GCK) に割り当てます。

BUFG=OE : グローバル トライステート制御 (GTS) に割り当てます。

BUFG=SR : グローバル セット/リセット制御 (GSR) に割り当てます。

BUFG=DATA\_GATE : DataGate ラッチ イネーブル制御に割り当てます。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約はインスタンス化の直前に入力します。

BUFG を次のように指定します。

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

- ・ BUFG=CLK : グローバル クロック (GCK) に割り当てます。
- ・ BUFG=OE : グローバル トライステート制御 (GTS) に割り当てます。
- ・ BUFG=SR : グローバル セット/リセット制御 (GSR) に割り当てます。
- ・ BUFG=DATA\_GATE : DataGate ラッチ イネーブル制御に割り当てます。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

UCF 構文は次のとおりです。

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

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

- ・ CLK : グローバル クロック ピンを指定します (すべての CPLD ファミリ)。
- ・ OE は、グローバル トライステート制御ピン (CoolRunner™-II と CoolRunner XPLA3 デバイスを除く CPLD) または内部グローバル トライステート コントロール ライン (CoolRunner-II デバイス) を指定します。
- ・ SR : CoolRunner-II および CoolRunner XPLA3 デバイスを除く CPLD デバイスでグローバル セット/リセット ピンを指定します。
- ・ DATA\_GATE : DataGate ラッチ イネーブル制御線に指定します。

次の文により、信号 **fastclk** がグローバル クロック ネットに割り当てられます。

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

### XCF 構文

```
BEGIN MODEL "entity_name "
```

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

```
END;
```

## CLOCK\_DEDICATED\_ROUTE

CLOCK\_DEDICATED\_ROUTE 制約は、アーキテクチャ別にクロック配置規則に従うかどうかを指定する高度な制約です。この制約が使用されていない場合、または TRUE に設定されている場合は、クロック配置規則に従いますが、従わない場合は、配置でエラーになります。この制約が FALSE に設定されていると、特定のクロック配置規則は無視されたまま、配置配線が続行されます。可能であれば、デザインに含まれるすべてのクロック配置規則違反を修正し、最適なクロック パフォーマンスを達成できるようにします。この制約は、クロック配置規則に違反することが絶対に必要な場合にのみ使用してください。特定のクロック配置規則の詳細は、『Hardware User Guide』を参照してください。

## アーキテクチャ サポート

この制約は、サポートされる FPGA アーキテクチャすべてに適用されます。

## CLOCK\_DEDICATED\_ROUTE 適用可能エレメント

- ・ ネット
- ・ 次のプリミティブの入力ピンと出力ピン
  - BUFG
  - BUFR
  - DCM
  - PLLPMCD
  - GT11
  - GT11 DUAL
  - GT11 CLK
  - GTP DUAL

## 適用ルール

ネットまたはインスタンス ピンに設定できます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : CLOCK\_DEDICATED\_ROUTE
- ・ TRUE、FALSE

### UCF および NCF 構文

構文は次のとおりです。

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

*BEL\_INSTANCE\_NAME.PIN* は制約を付ける特定のインスタンスの入力/出力ピン (例 : DCM インスタンスの CLKIN 入力ピン) です。

## COLLAPSE

COLLAPSE は、高度なフィッタ制約です。COLLAPSE を指定すると、組み合わせノードがすべてのファンアウトにコラプスします。

## アーキテクチャ サポート

この制約は、CPLD デバイスにのみ適用できます。

## 適用可能エレメント

すべての内部ネット

## 適用ルール

COLLAPSE は、ネット制約です。デザイン エレメントには適用できません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ ロジック シンボルまたはその出力ネットに設定します。
- ・ 属性名 : COLLAPSE
- ・ 属性値 : TRUE、FALSE

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute collapse: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

### Verilog 構文

この制約はインスタンスエーションの直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

### UCF および NCF 構文

UCF 構文は次のとおりです。

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

次の文は、ネット \$1N6745 をすべてのファンアウトにコラプスします。

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

## COMPGRP

COMPGRP は、高度なグループ制約で、コンポーネント グループを定義できます。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

コンポーネントのグループ

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 構文

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

この場合、次を表します。

*comp\_item* は、次のいずれかです。

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

## CONFIG\_MODE

この制約は、どちらの兼用コンフィギュレーション ピンを汎用 I/O として使用するかを PAR に指示します。

CONFIG\_MODE が S\_SELECTMAP+READBACK または M\_SELECTMAP+READBACK の場合、兼用 I/O が使用されないように指示します。

CONFIG\_MODE が S\_SELECTMAP または M\_SELECTMAP の場合、兼用 I/O は必要に応じて汎用 I/O として使用されます。

## アーキテクチャ サポート

この制約は、Spartan®-3、Virtex-4、Virtex-5 デバイスに適用できます。

## 適用可能エレメント

CONFIG シンボルの後に付加します。

## 適用ルール

兼用 I/O に適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF 構文

UCF 構文は次のとおりです。

**CONFIG CONFIG\_MODE=*string*;**

*string* は、次のいずれかです。

- S\_SERIAL : スレーブ シリアル モード
- M\_SERIAL : マスタ シリアル モード (デフォルト)
- S\_SELECTMAP : スレーブ SelectMAP モード
- M\_SELECTMAP : マスタ SelectMAP モード
- B\_SCAN : バウンダリ スキャン モード
- S\_SELECTMAP+READBACK : リードバックと再コンフィギュレーションをサポートするために設定された Persist 付きのスレーブ SelectMAP モード
- M\_SELECTMAP+READBACK : リードバックと再コンフィギュレーションをサポートするために設定された Persist 付きのマスタ SelectMAP モード
- B\_SCAN+READBACK : Persist 付きのバウンダリ スキャン モード (リードバックとリコンフィギュレーションをサポート)
- S\_SELECTMAP32+READBACK : リードバックと再コンフィギュレーションをサポートするために設定された Persist 付きのスレーブ SelectMAP モード
- S\_SELECTMAP32 : スレーブ SelectMAP32 モード

### メモ :

S\_SELECTMAP32 および S\_SELECTMAP32+READBACK については、Virtex-5 の場合に S\_SELECTMAP16 および S\_SELECTMAP16+READBACK を選択すると、コンフィギュレーション後も維持する必要のあるデータピンの数が正しく設定できます。

## COOL\_CLK (CoolCLOCK)

クロック分周器と DualEDGE 回路を組み合わせると消費電力を低減できます。この機能は、CoolCLOCK と呼ばれ、CPLD 内でクロック消費電力を低減するために開発されました。クロック ネットではかなりの量の電力が消費されるため、半分の周波数でネットを駆動し、DualEDGE で動作するマクロセルを使用してクロック レートを倍増し、クロック消費電力を低減できます。

### アーキテクチャ サポート

この制約は、CoolRunner™-II デバイスにのみ適用できます。

### 適用可能エレメント

入力パッドまたはレジスタ クロックを駆動する内部信号に適用できます。

### 適用ルール

クロック ネットに COOL\_CLK を設定すると、2 分周のクロック分周器 (CLK\_DIV2) を介してクロックを渡し、そのクロックで制御されるフリップフロップをすべて DualEDGE フリップフロップに置き換えることになります。COOL\_CLK 属性を使用しても、デザイン全体の機能は変更できません。

COOL\_CLK を使用する場合、次のような制限が適用されます。

- ・ 立ち下がりエッジでフリップフロップを動作させるクロックに COOL\_CLK は使用できません。CoolRunner-II のクロック分周器は、クロック信号の立ち上がりエッジのみで動作します。
- ・ デザイン ソースに DualEDGE フリップフロップがある場合、そのフリップフロップを制御するクロックは COOL\_CLK として指定できません。
- ・ デザイン ソースにクロック分周器が既にある場合は、COOL\_CLK を使用できません。CoolRunner-II デバイスにはクロック分周器 1 つのみが含まれます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

#### 回路図

- ・ 入力パッドまたはレジスタ クロックを駆動する内部信号に適用できます。
- ・ 属性名 : COOL\_CLK
- ・ 属性値 : TRUE、FALSE

#### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute cool_clk: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

#### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

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

## DATA\_GATE

CoolRunner™-II の DataGate 機能により、デザインの消費電力を削減できます。入力信号の遷移が CPLD デザインのファンクションに関係ない場合に、信号の遷移が伝搬されないようブロックするラッチが使用できるようになります。ラッチを通過しない場合、入力信号は CPLD デバイスのファンクション ブロック内で配線されているため、信号遷移が CPLD デザインの機能に影響を与えることはないものの電力が消費されます。デバイスに DATA\_GATE 制御の I/O ピンをアサートすると、選択した I/O ピンの入力にラッチが付けられ、それらのピンの外部信号遷移に伴う電力の消費が抑えられます。

## アーキテクチャ サポート

この制約は、128 個以上のマクロセルを使用した CoolRunner-II デバイスにのみ適用できます。

## 適用可能エレメント

I/O パッドおよびピン

## 適用ルール

I/O パッドに DATA\_GATE 属性を設定すると、そのデバイス ピンに付けられた通過ラッチが DataGate 制御ピンに反応します。クロック入力パッドを含む I/O パッド (DATA\_GATE 制御の I/O ピンを除く) は、DATA\_GATE 属性を使用してラッチを付けるようにコンフィギュレーションできます。DATA\_GATE 属性が設定されていないその他の I/O パッドは常に、ラッチが付いていない状態になります。DATA\_GATE 制御信号は、DATA\_GATE 制御の I/O ピンを介してオフチップから送信されます。または、ラッチが付いていない入力 (DATA\_GATE 属性が設定されていないパッド) を含むデザインで生成することができます。

VHDL および Verilog デザインでの DATA\_GATE の使用については「[BUFG](#)」を参照してください。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ I/O パッドおよびピンに設定します。
- ・ 属性名 : DATA\_GATE
- ・ 属性値 : TRUE、FALSE

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute DATA_GATE : string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

### Verilog 構文

この制約はインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

### NCF 構文

UCF と同じです。

### UCF 構文

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

### XCF 構文

```
BEGIN MODEL "entity_name"
```

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

```
END ;
```

## DEFAULT

この制約を使用すると、複数の制約に新しい DEFAULT 制約値を設定できます。特定の制約は、該当箇所で DEFAULT 制約値を上書きします。

制約値には、KEEPER、FLOAT、PULLDOWN、PULLUP のいずれかを使用します。

## アーキテクチャ サポート

DEFAULT 制約は、次の制約およびアーキテクチャに適用されます。

- ・ KEEPER、PULLDOWN – すべての FPGA および CPLD デバイスの CoolRunner™-II に適用されます。
- ・ PULLUP – すべての FPGA および CPLD デバイスの CoolRunner XPLA3 および CoolRunner-II に適用されます。

## 適用可能エレメント

DEFAULT でサポートされる制約のエレメントについては、上記の説明の各制約のリンクをクリックしてください。

## 適用ルール

DEFAULT でサポートされる制約の適用ルールについては、上記の説明の各制約のリンクをクリックしてください。

### 構文

DEFAULT でサポートされる制約の構文については、上記の説明の各制約のリンクをクリックしてください。



次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

DEFAULT 制約を回路図に適用する場合は、次の手順に従います。

- ・ ネット、インスタンス、またはピンに設定します。
- ・ 属性名 : **DEFAULT** *constraint\_name* *constraint\_name* は、KEEPER、FLOAT、PULLDOWN、PULLUP のいずれかになります。
- ・ 属性値 : これらは、*constraint\_name* によって決まります。

## VHDL 構文

VHDL コードの場合、制約は VHDL 属性で記述します。DEFAULT 制約は、次のような構文で宣言してから使用してください。

```
attribute attribute_name : string;
```

例 :

```
attribute KEEPER : string;
```

VHDL 属性を宣言した後、次のように指定します。

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

DEFAULT に使用可能な *attribute\_names* は、KEEPER、FLOAT、PULLDOWN、PULLUP です。

使用できる属性値 *attribute\_values* は、属性のタイプによって異なります。

例 :

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

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog コードの場合、次のように DEFAULT 制約を指定します。

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

*constraint\_value* は、大文字/小文字が区別されます。

DEFAULT に使用可能な *CONSTRAINT\_NAMES* は、KEEPER、FLOAT、PULLDOWN、PULLUP です。

使用できる *constraint\_values* は、*constraint\_name* によって異なります。

例

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF 構文

UCF ファイルでは、DEFAULT 制約は次のように記述できます。

```
DEFAULT constraint_name;
```

DEFAULT に使用可能な *constraint\_names* は、KEEPER、FLOAT、PULLDOWN、PULLUP です。

例 :

```
DEFAULT KEEPER = TRUE;
```

### XCF 構文

次に、DEFAULT 制約の基本構文を示します。

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

DEFAULT に使用可能な *constraint\_names* は、KEEPER、FLOAT、PULLDOWN、PULLUP です。

使用できる属性値 *attribute\_values* は、属性のタイプによって異なります。

例 :

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

### NCF 構文

UCF と同じです。

### PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

### PACE の構文

**メモ** : PACE は、CPLD でのみサポートされます。

PACE は、主としてロケーション制約を I/O に設定する際に使用しますが、I/O 規格など特定の I/O プロパティの設定にも使用できます。PACE は、Project Navigator の [Processes] ウィンドウからアクセスできます。

詳細は、PACE ヘルプでエリア制約およびピンの編集に関するセクションを参照してください。

## DCI\_CASCADE

Virtex®-5 デバイス ファミリでは、DCI 参照電圧を必要とする IO バンクを、ほかの DCI IO バンクとカスケード接続でき、1 組の VRN/VRP ピンで、複数の IO バンクに参照電圧を供給できます。この方法では使用される VR ピン数および DCI コントローラ数が少なくて済むため、使用可能なピン数が増え、消費電力量が減少します。DCI\_CASCADE 制約では、DCI マスタ バンクと対応するスレーブ バンクを指定します。複数のマスタ/スレーブの組み合わせを指定する場合は、デザイン内にこの制約のインスタンスが複数含まれます。この制約で指定した情報を基に BitGen で DCI コントローラがプログラムされ、バンクがカスケード接続されます。配置の際にもこの情報が使用され、スレーブ バンクの VR ピンをほかの用途に使用できるかを調べます。

DCI\_CASCADE 制約の各インスタンスには、マスタ バンクが 1 つと、少なくとも 1 つのスレーブ バンクが含まれている必要があります。バンク間は、スペースで区切ります。最初に指定される値はマスタ バンクで、それ以降の値はスレーブ バンクです。スレーブ バンクには、マスタ バンクから DCI 参照電圧が供給されます。カスケード接続されたバンクは、同じ列にあり (左、中央、または右) VCCO 設定が同じである必要があります。詳細は、「UCF および NCF 構文」を参照してください。

## アーキテクチャ サポート

この制約は、Virtex-5 デバイスにのみ適用できます。

## 適用可能エレメント

最上位デザイン ブロックの DCI\_CASCADE 属性。

## 適用ルール

CONFIG ブロックに属性として配置され、物理デザイン オブジェクトに伝播します。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF および NCF 構文

UCF/NCF の構文は次のとおりです。

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

- ・ <master> = [1...MAX\_NUM\_BANKS]
- ・ <slave1> = [1...MAX\_NUM\_BANKS]
- ・ <slave2> = [1...MAX\_NUM\_BANKS]
- ・ 値はすべて、Virtex-5 デバイスで有効な IO バンクです。
- ・ マスタ バンクには、DCI 参照電圧を必要とする IO 規格が適用された IOB が必要です。
- ・ すべてのスレーブ バンクの VCCO 設定は、マスタ バンクと同じである必要があります。
- ・ マスタとスレーブ間にその他のバンクが存在する場合は、正しい方向にカスケード接続されている必要があります。

例 :

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

### PCF 構文

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

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

## DCI\_VALUE

DCI\_VALUE は、IBISWriter を使用して IBS ファイルを作成する際に、どのバッファのビヘイビア モデルがデザインの IOB に関連するかを指定します。

## アーキテクチャ サポート

Spartan®-3、Virtex®-4 および Virtex-5 デバイスがサポートされます。

## 適用可能エレメント

IOB

## 適用ルール

設定された IOB に適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF および NCF 構文

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

*integer* の値は、25 ~ 100 (単位は  $\Omega$ ) です。デフォルトでは 50 に設定されています。

## DIRECTED\_ROUTING

DIRECTED\_ROUTING は、少数のロードおよびソースの配線とタイミングを維持するために使用する制約です。この制約を使用する場合は、LOC、RLOC、または BEL 制約を使用してロードとソースの相対位置をまったく同じに維持しておく必要があります。

## アーキテクチャ サポート

この制約は、すべての FPGA アーキテクチャでサポートされています。

## 適用可能エレメント

ネットのみに設定できます。

## 適用ルール

ありません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF および NCF 構文

次の構文例は、説明のために示しており、実際には使用できません。指定配線制約を使用する場合は、ロードとソース コンポーネントを相対的に配置する必要があります。

## FPGA Editor からの設定

FPGA Editor で指定配線制約を設定するには、[Tools] → [Directed Routing Constraints] をクリックします。ロードとソース コンポーネントに自動設定される配置制約のタイプは、次の 3 つのいずれかになります。

- ・ 配置制約を生成しない場合
- ・ 相対ロケーション制約を使用しない場合
- ・ 絶対ロケーション制約を使用しない場合

### 配置制約を生成しない場合

[Do not generate Placement Constraint] オプションは、既存の RPM を使用して配線制約のみを生成します。この場合、配置制約は生成されません。

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

### 相対ロケーション制約を使用しない場合

[Use Relative Location Constraint] オプションは、ロードとソース コンポーネントに対して RPM を生成し、配線制約を設定します。RPM は、デバイス上の相対的な位置に配置し直されます。

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

上記の例では、RLOC 制約ごとにインスタンスが呼び出されます。この例では、インスタンスが 3 つ含まれています。

### 絶対ロケーション制約を使用しない場合

[Use Absolute Location Constraint] を使用すると、RLOC 制約と RLOC\_ORIGIN 制約を指定することで、ロードとソース コンポーネントをネットに接続して、特定位置にロックできます。ロケーション制約 (LOC) は、ユーザーが手動で指定することもできます。

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

## DISABLE

DISABLE は、特定パスのトレース制御をオフにするタイミング制約です。パストレーサ制御は、よくあるタイプのパスがタイミング パス解析でイネーブルになるかディスエーブルになるかを決定するために使用されます。パストレーサ制御文は、ソースがネットリスト、UCF、NCF のいずれの場合でも、PCF に出力されます。ただし、ネットリストの DISABLE を UCF で **ENABLE** に置き換えることはできません。</fc>

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

制約ファイル内全般

## 適用ルール

指定したブロックの遅延シンボルに対してタイミング解析が実行されないようにします。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF および NCF 構文

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

この場合、それぞれ次を表します。

*delay\_symbol\_name* は、パストレースの標準ブロックの遅延シンボル名、またはデータシートに表示される遅延名です。

これらのシンボルについては、次の表を参照してください。PCF ではコンポーネント遅延名も使用できます。

### パストレースの標準ブロック遅延シンボル

遅延シンボル名	パスのタイプ	デフォルト
reg_sr_o	非同期セット/リセットから出力への伝搬遅延	ディスエーブル
reg_sr_r	非同期セット/リセットからリカバリ パス	Virtex®-5 およびそれ以前のアーキテクチャではディスエーブル Virtex-6 および Spartan®-6 アーキテクチャではイネーブル
reg_sr_clk	同期セット/リセットからクロックへの設定 およびホールド チェック	イネーブル
lat_d_q	データから出力へのトランスペアレント ラッチ遅延	ディスエーブル
lat_ce_q	クロック イネーブルから出力へのトランス ペアレント ラッチ遅延	ディスエーブル
ram_we_o	RAM 書き込みイネーブルから出力への 伝搬遅延	イネーブル
io_pad_i	I/O パッドから入力への伝搬遅延	イネーブル
io_t_pad	I/O トライステートからパッドへの伝搬遅延	イネーブル
io_o_i	I/O 出力から入力への伝搬遅延。トライ ステート IOB ではオフ。	イネーブル
io_o_pad	I/O 出力からパッドへの伝搬遅延	イネーブル

### PCF 構文

UCF と同じです。

## DRIVE

DRIVE は、サポートされる FPGA デバイスすべての駆動電流の出力を選択する基本的なマップ制約です。

DRIVE 制約では、I/O 規格に LVTTTL、LVCMOS12、LVCMOS15、LVCMOS18、LVCMOS25 または LVCMOS33 を使用する SelectIO™ バッファの駆動電流 (mA) が選択されます。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

上記の表でサポートありと示されたデバイスに次のエレメントを 1 つまたは複数使用して制約を設定できます。すべてのデバイス ファミリーでこれらのエレメントがすべてサポートされているわけではありません。どのデバイス ファミリーにどのエレメントを使用できるかは、ライブラリ ガイドを参照してください。詳細は、[データシート](#)を参照してください。

- ・ IOB 出力コンポーネント (OBUF および OFD など)
- ・ LVTTTL、LVCMOS15、LVCMOS18、LVCMOS25、LVCMOS33 を I/O 規格に使用する SelectIO 出力バッファ
- ・ ネット

## 適用ルール

ネットまたは信号がパッドに接続されている場合を除いて、ネットや信号に設定できません。この場合、DRIVE はパッドインスタンスに設定されているものと見なされます。デザイン エレメントに設定すると、そのデザイン エレメントの階層にあるすべての適用可能エレメントに適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効な IOB 出力コンポーネントに設定します。
- ・ 属性名 : DRIVE
- ・ 属性値 : 詳細は、「UCF および NCF 構文」を参照してください。

### VHDL 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

VHDL 制約を次のように宣言します。

```
attribute drive: string;
```

Specify the VHDL constraint as follows:

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

有効な *values* については、「UCF および NCF 構文」を参照してください。VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

### Verilog 構文

Verilog 制約を次のように指定します

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

有効な *values* については、「UCF および NCF 構文」を参照してください。Verilog 構文の詳細は、「Verilog」を参照してください。

## UCF および NCF 構文

### IOB 出力コンポーネント (UCF)

Spartan®-3 および Virtex®-4 以降のデバイスでは、次のように記述します。

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

デフォルトでは 12mA に設定されています。

### SelectIO 出力コンポーネント (IOBUF\_SelectIO、OBUF\_SelectIO、OBUFT\_SelectIO)

- ・ Spartan-3 および Virtex-4 以降のデバイスの LVTTTL 規格では、次のように記述します。

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

デフォルトでは 12mA に設定されています。

- ・ Spartan-3 および Virtex-4 以降のデバイスの LVCMOS12、LVCMOS15、LVCMOS18 規格では、次のように記述します。

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

デフォルトでは 12mA に設定されています。

- ・ Spartan-3 および Virtex-4 以降のデバイスの LVCMOS25 および LVCMOS33 規格では、次のように記述します。

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

デフォルトでは 12mA に設定されています。

## XCF 構文

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

```
BEGIN MODEL "entity_name "
```

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

```
END;
```

## PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約



## DROP\_SPEC

DROP\_SPEC は、高度のタイミング制約で、入力デザインで定義されたタイミング制約を解析の対象から外すように指定できます。この制約は、入力デザインで定義したタイミング制約を新たに制約ファイルで定義した制約に置き換えられない場合や、入力デザインの一部の制約を無視する場合に使用します。このタイミング制約を入力ネットリスト (または NCF ファイル) で頻繁に使用することは望ましくありませんが、不正ではありません。入力デザインで定義した場合、TIMESPEC の後に DROP\_SPEC を付ける必要があります。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイ스에適用できます。

## 適用可能エレメント

タイミング制約

## 適用ルール

ネットまたはマクロには設定できません。DROP\_SPEC は、指定されたタイムスペックを削除します。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF および NCF 構文

**TIMESPEC "TSidentifier"=DROP\_SPEC;**

TSidentifier は、削除するタイムスペックの識別名です。

次の文は、入力デザイン仕様 TS67 をキャンセルするように指定します。

**TIMESPEC "TS67"=DROP\_SPEC;**

## 構文

**"TSidentifier" DROP\_SPEC;**

## ENABLE

ENABLE は、特定パスのトレース制御をオンにするタイミング制約です。パストレース制御は、よくあるタイプのパスがタイミング パス解析でイネーブルになるかディスエーブルになるかを決定するために使用されます。「DISABLE」も参照してください。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

制約ファイル内全般

## 適用ルール

指定したパス遅延に対してタイミング解析が実行されます。<:cnmk 6>

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF および NCF 構文

グローバルなタイミング仕様にのみ適用できます。UCF パストレース構文は、次のとおりです。

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

*delay\_symbol\_name* は、パストレースの標準ブロックの遅延シンボル名 (次表)、またはデータシートに表示される遅延名です

### パストレースの標準ブロック遅延シンボル

遅延シンボル名	パスのタイプ	デフォルト
reg_sr_o	非同期セット/リセットから出力への伝搬遅延	ディスエーブル
reg_sr_r	非同期セット/リセットからリカバリ パス	Virtex®-5 およびそれ以前のアーキテクチャではディスエーブル Virtex-6 および Spartan®-6 アーキテクチャではイネーブル
reg_sr_clk	同期セット/リセットからクロックへの設定およびホールド チェック	イネーブル
lat_d_q	データから出力へのトランスペアレントラッチ遅延	ディスエーブル
lat_ce_q	クロック イネーブルから出力へのトランスペアレントラッチ遅延	ディスエーブル
ram_we_o	RAM 書き込みイネーブルから出力への伝搬遅延	イネーブル
io_pad_i	I/O パッドから入力への伝搬遅延	イネーブル
io_t_pad	I/O トライステートからパッドへの伝搬遅延	イネーブル
io_o_1	I/O 出力から入力への伝搬遅延。トライステート IOB には無効です。	イネーブル
io_o_pad	I/O 出力からパッドへの伝搬遅延	イネーブル

### PCF 構文

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

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

## ENABLE\_SUSPEND

Spartan®-3A デバイス ファミリで、電力削減モードの SUSPEND のビヘイビアを定義します。使用可能な値は、NO、FILTERED、UNFILTERED で、NO を指定すると、この機能はオフになります。オンにする場合、FILTERED を指定するとグリッチ フィルタが使用されます。この場合は、長いパルス幅が必要です。UNFILTERED を指定すると、フィルタがバイパスされ、SUSPEND がすぐにオンになります。デフォルト値は、NO です。

### アーキテクチャ サポート

この制約は、Spartan-3A デバイスにのみ適用できます。

### 適用可能エレメント

Spartan-3A デバイスのグローバル属性で、特定のエレメントには設定されません。

### 適用ルール

グローバル属性で、デザイン全体に設定します。

#### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

#### UCF 構文

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

Example:

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

## FAST

FAST は、基本的なマップ制約です。この制約を設定すると、IOB 出力の速度が上がりますが、ノイズおよび消費電力が増加する場合があります。

### アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイ스에適用できます。

### 適用可能エレメント

- ・ 出力プリミティブ
- ・ 出力パッド
- ・ 双方向パッド

FAST 制約は、UCF ファイル内で、パッド コンポーネントに接続されているネットにも設定できます。ネットに設定した制約は、NGDBuild により NGD ファイル内のパッド インスタンスに挿入され、マップで処理されます。次の構文を使用してください。

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

## 適用ルール

基本的に FAST はネットに設定できませんが、ネットがパッドに接続されている場合は例外です。この場合、FAST はパッド インスタンスに設定されているものと見なされます。マクロ、エンティティ、またはモジュールに設定されている場合、FAST は、そのモジュール以下の階層にあるすべての適用可能エレメントに適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : FAST
- ・ 属性値 : TRUE、FALSE

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute FAST: string;
```

Specify the VHDL constraint as follows:

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

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

### Verilog 構文

この制約はインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、[「Verilog」](#)を参照してください。

### UCF および NCF 構文

次の文は、エレメント y2 の出力速度を向上します。

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

次の文は、net1 が接続されているパッドの出力速度を向上します。

```
NET "net1" FAST;
```

### XCF 構文

```
BEGIN MODEL "entity_name"
```

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

```
END;
```

### PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## FEEDBACK

FEEDBACK 制約は、DCM がボード スキュー アプリケーションで使用される場合に、外部の DCM フィードバック パスの遅延を定義するために使用します。この遅延は、ボードトレースの最大外部パス遅延として定義されます。内部 FPGA パス遅延は含まれません。この制約は、タイミング ツールで DCM 位相シフトが正しく指定され、関連する同期パスが解析されるために必要です。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

ありません。

## 適用ルール

*input\_feedback\_clock\_net* および *output\_clock\_net* は、パッド ネットに対応している必要があります。それ以外のネットに設定した場合、エラーが発生します。*input\_feedback\_clock\_net* には入力パッド、*output\_clock\_net* には出力パッドを指定してください。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### Constraints Editor からの設定

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Miscellaneous] の下の [DCM Feedback] をダブルクリックし、ダイアログ ボックスにアクセスします。

### PlanAhead からの設定

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## UCF 構文

UCF 構文は次のとおりです。

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

- ・ *input\_feedback\_clock\_net* は、DCM へのフィードバックとして使用される入力パッド ネットの名前です。
- ・ *value* は、ユーザーによって算出されたボードトレース遅延です。
- ・ *units* は、ns または ps です。デフォルトは ns です。
- ・ *output\_clock\_net* は、DCM により駆動される出力パッド ネットの名前です。

### XCF 構文

```
BEGIN MODEL "entity_name"
```

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

```
END;
```

- ・ *input\_feedback\_clock\_net* は、DCM へのフィードバックとして使用される入力パッド ネットの名前です。
- ・ *value* は、ユーザーによって算出されたボードトレース遅延です。
- ・ *units* は、ns または ps です。デフォルトは ns です。
- ・ *output\_clock\_net* は、DCM により駆動される出力パッド ネットの名前です。

### PCF 構文

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

## FILE

ネットリストに含まれたモジュールをインスタンス化すると、このファイル名が NGCBuild によって検索されます。このため、ネットリストの名前はファイル内で定義されたモジュールの名前と同じにする必要があります。モジュール名とは異なる名前を付ける場合は、インスタンス宣言に FILE 制約を指定しておく、NGCBuild で指定したファイル内でモジュールが検索されます。

ザイリンクス制約が VHDL のキーワードとなっている場合、VHDL 属性でザイリンクス制約を使用できません。この問題を回避するには、制約エイリアスを使用します。ISE® 7.1 以降は、各制約に固有のエイリアスがあるようになりました。エイリアス名には、制約名に XIL という接頭辞を付けます。たとえば、FILE 制約は VHDL で使用できないので、XIL\_FILE という名前を使用します。現在のところ、既存の XILFILE エイリアスはサポートされています。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイ스에適用できます。

## 適用可能エレメント

指定したファイル内で定義されるインスタンス宣言

## 適用ルール

インスタンスのみに適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : FILE
- ・ 属性値 : *file\_name.extension*

*file\_name* は、制約が設定されたエレメントの基になるロジックを表すファイルの名前です。

ファイル タイプには EDIF、EDN、NGC、NMC があります。

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute xilfile: string;
```

Specify the VHDL constraint as follows:

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンシエーションの直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

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

この構文は、UCF には適用できません。

## FLOAT

FLOAT は、基本的なマップ制約で、トライステートパッドが駆動されていないときにフロートするよう設定します。FLOAT 制約は、適用される I/O のデフォルトの終端が、Project Navigator で PULLUP、PULLDOWN、または KEEPER に設定されている場合に有効です。

## アーキテクチャ サポート

この制約は、CoolRunner™ XPLA3 および CoolRunner-II デバイスに適用できます。

## 適用可能エレメント

ネットまたはピンに設定できます。

## 適用ルール

設定されたネットまたはピンに適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : 浮動小数点
- ・ 属性値 : 必要ありません。TRUE および FALSE が使用できます。デフォルトで TRUE が設定されています。

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute FLOAT: string;
```

Specify the VHDL constraint as follows:

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

## Verilog 構文

この制約はインスタンシエーションの直前に入力します。

Verilog 制約を次のように指定します

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

## UCF および NCF 構文

UCF 構文は次のとおりです。

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

## XCF Syntax

```
BEGIN MODEL "entity_name"
```

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

```
END;
```

## FROM-THRU-TO

FROM-THRU-TO は、高度なタイミング制約で、High または Low の時間を使用した PERIOD 制約に関連しています。同期パスの場合はセットアップ パスのみを設定できます。この制約は、ソース グループを始点として、中間点を通過し、デスティネーション グループを終点とするパスに適用されます。ソース グループおよびデスティネーション グループは、ユーザー グループまたは定義済みグループです。THRU を使用する前に TPTHU を使用して、パスの中間点を定義する必要があります。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

定義済みのグループおよびユーザー定義のグループに適用されます。



## 適用ルール

特定の FROM-THRU-TO パスのみに適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF および NCF 構文

**TIMESPEC** "*TSidentifier*"=**FROM** "*source\_group*" **THRU** "*thru\_pt1*"...[**THRU** "*thru\_pt2*" ...] **TO** "*destination\_group*" *value* [*Units*] [**DATAPATHONLY**];

- ・ *identifier* には、文字とアンダースコアを組み合わせで入力します。
- ・ *source\_group* および *destination\_group* は、ユーザー定義のグループまたは定義済みのグループです。
- ・ *thru\_pt1* および *thru\_pt2* は、タイミング解析用のパスを定義する中間点です。
- ・ *value* は、遅延時間です。
- ・ *units* は、ps、ms、ns、micro などの時間の単位です。

DATAPATHONLY キーワードは、FROM-TO 制約でクロック スキューまたは位相情報が考慮されないことを示します。このキーワードを使用すると、制約が設定されているグループとタイミング解析されるグループのデータ パスのみが指定されます。

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

FROM または TO はオプションです。FROM のみ、TO のみを使用することも可能です。

FROM、THRU、TO すべてを必ずしも指定する必要はありません。FROM-TO、FROM-THRU-TO、THRU-TO、TO、FROM、FROM-THRU などさまざまな組み合わせで指定できます。THRU ポイントの数に制限はありません。ソース、THRU ポイント、デスティネーションには、ネット、BEL、コンポーネント、マクロ、ピン、タイムグループを指定できます。

### Constraints Editor からの設定

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。

1. [Timing Constraints] の下の [Exceptions] で [Paths in the Constraint Type] をクリックし、[Path Exceptions] ダイアログ ボックスを表示します。
2. [Through points] のプラス記号 (+) をクリックして展開すると、使用可能なスルーアウト ポイントが識別できます。
3. [Path Exceptions] ダイアログ ボックスを設定します。

### PlanAhead からの設定

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## PCF 構文

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

FROM、THRU、TO すべてを必ずしも指定する必要はありません。FROM-TO、FROM-THRU-TO、THRU-TO、TO、FROM、FROM-THRU-THRU-THRU-TO、FROM-THRU などさまざまな組み合わせで指定できます。THRU ポイントの数に制限はありません。ソース、THRU ポイント、デスティネーションには、ネット、BEL、コンポーネント、マクロ、ピン、タイムグループを指定できます。

## FROM-TO

FROM-TO は、2 つのグループ間のタイミング制約を設定する制約で、High または Low の時間を使用した PERIOD 制約に関連しています。この場合のグループは、ユーザー定義のグループまたは定義済みのグループです。同期パスの場合、この制約はセットアップ パスのみを設定できます。

Virtex®-5 の場合、FROM-TO 制約でセットアップ パスとホールド パスの両方が制御されます。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイスに適用できます。

## 適用可能エレメント

定義済みのグループおよびユーザー定義のグループに適用されます。

## 適用ルール

2 つのグループ間で指定されたパスに適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## UCF および NCF 構文

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

- ・ TSname の冒頭は常に「TS」で、その後に英数字またはアンダースコアを付けます。
- ・ group1 は、オリジナル パスです。
- ・ group2 は、デスティネーション パスです。
- ・ value は、デフォルトでは ns に設定されています。その他の値としては、MHz や、TS\_C2S/2、TS\_C2S\*2 などのタイムスペックがあります。

DATAPATHONLY キーワードは、FROM-TO 制約でクロック スキューまたは位相情報が考慮されないことを示します。このキーワードを使用すると、制約が設定されているグループとタイミング解析されるグループのデータ パスのみが指定されます。

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

## XCF 構文

XST では、FROM-TO 制約が次の制限付きでサポートされます。

- ・ FROM-THRU-TO はサポートされません。
- ・ 仕様の結合はサポートされません。
- ・ 定義済みグループの文字列の一致はサポートされません。

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

## Constraints Editor からの設定

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Timing Constraints] の下の [Exceptions] でパスをダブルクリックし、ダイアログ ボックスにアクセスします。

## PlanAhead からの設定

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## 構文

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

FROM、THRU、TO すべてを必ずしも指定する必要はありません。FROM-TO、FROM-THRU-TO、THRU-TO、TO、FROM、FROM-THRU-THRU-THRU-TO、FROM-THRU などさまざまな組み合わせで指定できます。THRU ポイントの数に制限はありません。ソース、THRU ポイント、デスティネーションには、ネット、BEL、コンポーネント、マクロ、ピン、タイムグループを指定できます。

## HBLKNM

HBLKNM は、高度なマップ制約で、階層的なブロック名をロジック エlement に割り当て、フラット化された階層デザインでのグループ化を行います。階層デザインの異なるレベルにある Element に同じブロック名が付いていて、そのデザインがフラットな場合、NGCBuild により階層パス名が接頭辞として HBLKNM 値に追加されます。

BLKNM と同様、HBLKNM はファンクション ジェネレータとフリップフロップを同じ CLB にマップします。同じ HBLKNM を持つシンボルは、可能であれば、同じ CLB にマップされます。

BLKNM の代わりに HBLKNM を使用した場合、変換中に階層パス名が追加され、同じデザイン Element の異なるインスタンス内の Element に同じ HBLKNM や値を使用できるという利点があります。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

上記の表でサポートありと示されたデバイスに次のエレメントを 1 つまたは複数使用して制約を設定できます。すべてのデバイス ファミリでこれらのエレメントがすべてサポートされているわけではありません。どのデバイス ファミリにどのエレメントを使用できるかは、ライブラリ ガイドを参照してください。詳細は、[データシート](#)を参照してください。

1. レジスタ
2. I/O エレメントおよびパッド
3. FMAP
4. PULLUP
5. ACLK、GCLK
6. BUFG
7. BUFGS、BUFGP
8. ROM
9. RAMS および RAMD
10. キャリー ロジック プリミティブ

HBLKNM 制約は、UCF ファイル内で、パッド コンポーネントに接続されているネットにも設定できます。ネットに設定した制約は、NGCBuild により NGD ファイル内のパッド インスタンスに挿入され、マップで処理されます。次の構文を使用してください。

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

## 適用ルール

デザイン エレメントに設定すると、そのデザイン エレメントの階層にあるすべての適用可能エレメントに適用されます。ただし、NET に適用される場合、HBLKNM は PAD にのみ適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : HBLKNM
- ・ 属性値 : *block\_name*

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute hblknm: string;
```

VHDL 制約を次のように指定します。

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

*block\_name* は、その種類のシンボルで有効なブロック名です。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

*block\_name* は、その種類のシンボルで有効なブロック名です。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

UCF 構文は次のとおりです。

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

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

*block\_name* は、その種類のシンボルで有効なブロック名です。

次の構文で、エレメント *this\_fmap* がブロック *group1* に配置されます。

```
INST "$I13245/this_fmap" HBLKNM=group1;
```

次の文を使って、HBLKNM 制約を *net1* に接続されているパッドに設定します。

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

同じ HBLKNM があるエレメントは、可能であれば同じロジックブロックに配置されます。配置できない場合は、エラーが発生します。複数のエレメントに異なるブロック名を付けた場合、1 つのブロックにまとめて配置されません。

## HLUTNM

論理シンボルのグループ化の制御に使用されます。グループ化されたシンボルは、Virtex®-5 FPGA アーキテクチャの LUT サイトにまとめられます。値は文字列で、条件を満たす 2 つのシンボルに設定され、シンボルの階層内ではこれらのシンボルのみに使用されます。シンボルは、SLICE コンポーネント内で共有された LUT サイト内でインプリメントされます。

この制約の機能は、[HBLKNM](#) 制約に似ています。

## アーキテクチャ サポート

この制約は、Virtex-5 デバイスにのみ適用できます。

## 適用可能エレメント

同じ階層に存在し、それぞれの階層のレベル内で重複したシンボルがない 2 つのシンボルに設定します。両方のシンボルの、重複しない入力ピン数の合計が 5 を超えない場合は、5 入力以下のファンクション ジェネレータ シンボル (LUT、SRL16) 2 つに設定できます。両方のシンボルの、重複しない入力ピン数の合計が 6 入力を超えない場合は、6 入力の読み込み専用ファンクション ジェネレータ シンボル (LUT6) と 5 入力の読み込み専用シンボル (LUT5) に設定できます。ただし、6 入力シンボル プログラムの下位 32 ビットは 5 入力シンボル プログラムの 32 ビットすべてと一致している必要があります。

## 適用ルール

同じ階層に存在し、それぞれの階層のレベル内で重複したシンボルがない 2 つのシンボルに設定します。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ 有効なエレメントまたはシンボル タイプに設定します。
- ・ 属性名 : HLUTNM
- ・ 属性値 : <user\_defined>

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute hlutnm: string;
```

VHDL 制約を次のように指定します。

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

この場合、次を表します。

- ・ *instance\_name* は、インスタンス化された LUT、または LUTRAM のインスタンス名です。
- ・ *string\_value* は、シンボルの階層内で固有に使用される値です。デフォルト値はありません。空白の場合は、制約は無視されます。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

*string\_value* は、シンボルの階層内で固有に使用される値です。デフォルト値はありません。空白の場合は、制約は無視されます。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

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

*string\_value* は、シンボルの階層内で固有に使用される値です。デフォルト値はありません。空白の場合は、制約は無視されます。

## XCF 構文

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

## HU\_SET

HU\_SET は、高度なマップ制約で、デザイン階層によって定義されます。ただし、この制約では集合の名前を指定することもできます。H\_SET の場合は、1 つの階層エレメントに 1 つの H\_SET 制約しか指定できませんが、HU\_SET の場合は、集合として名前を指定すると複数の HU\_SET を指定できるようになります。

NGCBuild は、デザインをフラットにし、HU\_SET に階層名を接頭語として付けます。

HU\_SET と H\_SET の違いは次のとおりです。

HU_SET	H_SET
HU_SET 集合には、ユーザー定義のベース名の前に階層名が付けられる	デザインのフラット化によって自動的に階層名がベース名の前に付けられる
HU_SET が設定されたシンボルが冒頭にくる	RLOC 制約が設定されたシンボルの 1 レベル上にあるマクロのインスタンスが冒頭にくる

各種の集合制約については、「[RLOC](#)」を参照してください。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

上記の表でサポートありと示されたデバイスに次のエレメントを 1 つまたは複数使用して制約を設定できます。すべてのデバイス ファミリでこれらのエレメントがすべてサポートされているわけではありません。どのデバイス ファミリにどのエレメントを使用できるかは、ライブラリ ガイドを参照してください。詳細は、[データシート](#)を参照してください。

1. レジスタ
2. FMAP
3. マクロ インスタンス
4. ROM
5. RAMS、RAMD
6. MULT18X18S
7. RAMB4\_Sm\_Sn、RAMB4\_Sn
8. RAMB16\_Sm\_Sn, RAMB16\_Sn
9. RAMB16
10. DSP48

## 適用ルール

HU\_SET はデザイン エレメント制約です。ネットには適用できません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : HU\_SET
- ・ 属性値 : *set\_name*

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute HU_SET: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

### Verilog 構文

この制約はインスタンスエーションの直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

### UCF および NCF 構文

UCF 構文は次のとおりです。

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

*set\_name* は、集合の識別名です。

*set\_name* には、デザイン内のほかの集合名とは異なる名前を指定する必要があります。

次の文は、レジスタ FF\_1 のインスタンスを集合 heavy\_set に割り当てます。

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

### XCF 構文

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

```
BEGIN MODEL "entity_name"
```

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

```
END;
```

## IBUF\_DELAY\_VALUE

IBUF\_DELAY\_VALUE は、追加のスタティック遅延を FPGA アレイの入力パスに追加するマップ制約です。この制約は、クロックまたは IOB (入出力ブロック) レジスタを直接駆動しない入力信号または双方向信号に適用できます。クロックおよび IOB レジスタを駆動する信号に設定する制約については、「[IFD\\_DELAY\\_VALUE](#)」を参照してください。IBUF\_DELAY\_VALUE には 0 ~ 16 の整数を指定できます。デフォルトでは 0 に設定されており、入力パスに遅延が追加されません。この制約に大きい値を設定すると、入力パスに大きい遅延が追加されます。この値は、バッファの追加遅延と関連しています。詳細は、[データシート](#)を参照してください。

## アーキテクチャ サポート

この制約は、Spartan®-3A および Spartan-3E デバイスに適用できます。

## 適用エレメント

最上位の I/O ポートに設定できます。



## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ 最上位のポートに設定します。
- ・ 属性名 : IBUF\_DELAY\_VALUE
- ・ 属性値 : 0-16

## VHDL 構文

最上位のポートに、VHDL 属性を設定します。

```
attribute IBUF_DELAY_VALUE : string;
```

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

*value* は 0 ～ 16 です。

次の構文は、ネット DataIn1 に値 5 の IBUF\_DELAY\_VALUE 制約を設定します。

```
attribute IBUF_DELAY_VALUE : string;
```

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

## Verilog 構文

最上位のポートに Verilog 属性を設定します。

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

*value* は 0 ～ 16 です。

次の構文は、ネット DataIn1 に値 5 の IBUF\_DELAY\_VALUE 制約を設定します。

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

## UCF および NCF 構文

UCF 構文は次のとおりです。

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

*value* は、IBUF の遅延時間です。 *value* は 0 ～ 16 です。

次の構文は、ネット DataIn1 に値 5 の IBUF\_DELAY\_VALUE 制約を設定します。

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

## IFD\_DELAY\_VALUE

IFD\_DELAY\_VALUE は、追加のスタティック遅延を FPGA アレイの入力パスに追加するマップ制約です。この制約は、IOB (入出力ブロック) レジスタを駆動する入力信号または双方向信号に適用できます。IOB レジスタを駆動しない信号に設定する制約については、「IBUF\_DELAY\_VALUE」を参照してください。

IFD\_DELAY\_VALUE には 0 ～ 8 の整数と AUTO を指定できます。デフォルトには AUTO が設定されており、デステーションレジスタの入力ホールドタイムが満たされるよう適切な遅延が自動的にデータパスに追加されます。

IFD\_DELAY\_VALUE を 0 に設定すると、データパスに遅延が追加されません。1 ~ 8 の整数を設定すると、データパスのその値の分だけ遅延が追加されます。この値は、バッファの追加遅延と関連しています。詳細は、[データシート](#)を参照してください。

## アーキテクチャ サポート

Spartan®-3A および Spartan-3E デバイスがサポートされます。

## 適用エレメント

最上位の I/O ポートに設定できます。

## 適用ルール

この制約は I/O シンボルに設定しますが、適用されるのは I/O コンポーネント全体です。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ ネットに設定します。
- ・ 属性名 : IFD\_DELAY\_VALUE
- ・ 属性値 : 0-8、AUTO

### VHDL 構文

最上位のポートに、VHDL 属性を設定します。

```
attribute IFD_DELAY_VALUE : string;
```

次の構文は、ネット DataIn1 に値 5 の IFD\_DELAY\_VALUE 制約を設定します。

```
attribute IFD_DELAY_VALUE : string;
```

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

### Verilog 構文

最上位のポートに Verilog 属性を設定します。

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

次の構文は、ネット DataIn1 に値 5 の IFD\_DELAY\_VALUE 制約を設定します。

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

### UCF および NCF 構文

UCF 構文は次のとおりです。

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

value は、IBUF の遅延時間です。

次の構文は、ネット DataIn1 に値 5 の IFD\_DELAY\_VALUE 制約を設定します。

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

## INREG

この制約は、入力パッドにより駆動される D 入力付きのレジスタおよびラッチのインスタンスか、そのレジスタおよびラッチの Q 出力ネットに設定します。デフォルトでは、CoolRunner™ XPLA3 または CoolRunner-II デザインに含まれる入力パッドにより駆動される D 入力付きのレジスタおよびラッチが、デバイスの高速入力パスを使用して自動的にインプリメントされます。Project Navigator で [Use Fast Input for INREG for the Fit (Implement Design)] プロセスを使用しない場合、INREG 属性が設定されたレジスタおよびラッチのみが高速入力で最適化されます。

## アーキテクチャ サポート

CoolRunner XPLA3 および CoolRunner-II デバイスがサポートされます。

## 適用可能エレメント

入力パッドにより駆動される D 入力付きのレジスタおよびラッチのインスタンスか、そのレジスタおよびラッチの Q 出力ネットに設定します。

## 適用ルール

制約が設定されたレジスタ/ラッチか、そのレジスタ/ラッチの Q 出力ネットに適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ レジスタ、ラッチ、またはネットに設定します。
- ・ 属性名 : INREG
- ・ 属性値 : なし (デフォルトは TRUE)

### UCF 構文

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

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

## IOB

IOB は、基本的なマップ制約および合成制約で、IOB/ILOGIC/OLOGIC に移動可能なフリップフロップとラッチを指定します。マップ処理では、コマンドライン オプション (-pr i | o | b | off) を使用して、フリップフロップまたはラッチのプリミティブを入力 IOB (i)、出力 IOB (o)、または入出力 IOB (b) に移動します。フリップフロップまたはラッチで指定する IOB 制約は、マップに対して、可能であればそのインスタンスを IOB タイプのコンポーネントにパックするよう指示します。IOB 制約は、マップの -pr コマンドライン オプションよりも優先されますが、LOC 制約よりは優先されません。

IOB 制約は XST によりインプリメンテーション制約と見なされ、生成された NGC ファイルに伝搬されます。また、出力バッファのイネーブル ピンを駆動するフリップフロップおよびラッチのコピーを作成して、対応するフリップフロップおよびラッチが IOB にパックされるようにします。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

- ・ INFF/OUTFF 以外のフリップフロップ、ラッチ プリミティブ
- ・ レジスタ

## 適用ルール

設定したデザイン エレメントに適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ フリップフロップまたはラッチのインスタンス、またはレジスタに設定します。
- ・ 属性名 : IOB
- ・ 属性値 : TRUE、FALSE、AUTO、FORCE

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute iob:  string;
```

VHDL 制約を次のように指定します

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

この場合、それぞれ次を表します。

- ・ TRUE の場合は、フリップフロップまたはラッチが IOB に移動します。
- ・ FALSE の場合は、IOB に移動しません。
- ・ AUTO は、XST でのみ使用されます。XST では、タイミング制約を考慮して、フリップフロップを IOB に移動するかどうか自動的に決定されます。
- ・ FORCE の場合は、フリップフロップまたはラッチが IOB に移動し、それ以外の場合はエラー メッセージが表示されます。FORCE に設定すると、レジスタに I/O 接続があり、IOB にパックできないと、エラー メッセージが表示されます。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します。

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

値の定義については、上記の「VHDL 構文」を参照してください。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

### UCF および NCF 構文

構文は次のとおりです。

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

値の定義については、上記の「VHDL 構文」を参照してください。

次の文は、マップがインスタンス foo/bar を IOB コンポーネントに配置するように指定します。

```
INST "foo/bar" IOB=TRUE;
```

### XCF 構文

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name" iob={true|false|auto|force};
```

```
INST " instance_name" iob={true |false |auto|force};
```

```
END;
```

**メモ:** AUTO を指定すると、XST ではタイミング制約を考慮して、フリップフロップを IOB に移動するかどうか自動的に決定されます。

### Constraints Editor からの設定

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リストボックスの [Miscellaneous] の下の [Registers to be Placed on IOBs] をダブルクリックし、ダイアログ ボックスにアクセスします。

## IOBDELAY

IOBDELAY は、基本的なマップ制約で、デバイス内の入力パス遅延エレメントのプログラム方法を指定できます。

入力信号には、ローカルの IOB 入力 FF または IOB への外部読み込みという 2 種類のデスティネーションが考えられます。ザイリンクス デバイスを使用すると、遅延エレメントはこれらデスティネーションのうち 1 つか、または両方への信号送信を遅らせることができます。

IOBDELAY は [NODELAY](#) と併用できません。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

すべての I/O シンボル (I/O パッド、I/O バッファ、または入力パッド ネット)

## 適用ルール

この制約は I/O シンボルに設定されますが、I/O コンポーネント全体に適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ I/O シンボルに設定します。
- ・ 属性名 : IOBDELAY
- ・ 属性値 : NONE、BOTH、IBUF、IFD

**メモ** : Spartan®-3 ファミリの場合、デフォルトは NONE に設定されているので、ホールド タイムを 0 にすることができます。

## VHDL 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

VHDL 制約を次のように宣言します。

```
attribute iobdelay: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

UCF 構文は次のとおりです。

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

NONE に指定すると、IBUF パスと IFD パスの遅延が OFF に設定されます。

**メモ** : Spartan-3 ファミリの場合、デフォルトは NONE に設定されているので、ホールド タイムを 0 にすることができます。

- ・ BOTH : IBUF パスと IFD パスの遅延が ON に設定されます。
- ・ IBUF : 入力バッファが I/O コンポーネント外部にあるレジスタの D ピンを駆動する場合、I/O コンポーネント内部のレジスタで遅延をオフに、I/O コンポーネント外部のレジスタで遅延をオンにします。
- ・ IFD : レジスタが I/O コンポーネントの入力側に配置されている場合、IOB=TRUE 制約がレジスタに設定されているかどうかにかかわらず、I/O コンポーネント内部のレジスタで遅延をオンに、I/O コンポーネント外部のレジスタで遅延をオフに設定します。

この文は、IBUF パスと IFD パスの遅延を OFF に設定します。

```
INST "xyzyzy" IOBDELAY=NONE
```

## IODELAY\_GROUP

IODELAY\_GROUP は、IDELAY/IODELAY のセットと IDELAYCTRL を組み合わせてデザイン インプリメンテーション制約で、IDELAYCTRL を自動的に複製して配置させることができます。この制約を使用する具体的な例は、該当するアーキテクチャのユーザー ガイドの IDELAYCTRL のセクションを参照してください。

## アーキテクチャ サポート

IODELAY\_GROUP 制約は、Virtex®-4 および Virtex-5 アーキテクチャで使用できます。Virtex-4 の場合、この制約は MAP の [Timing Driven Pack and Placement] オプションを使用した場合にのみサポートされます。

## 適用可能エレメント

IDELAY、IODELAY、IDELAYCTRL プリミティブ インスタンスレーション

## 適用ルール

IODELAY\_GROUP は、デザイン エレメントにのみ使用できる制約です。IODELAY\_GROUP はネット、信号、またはピンには適用できません。2 つ以上の IODELAY\_GROUP 制約をマージする場合は、MIODELAY\_GROUP を参照してください。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute IDELAY_GROUP: string;
```

VHDL 制約を次のように指定します。

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

*group\_name* の詳細は、「UCF 構文」を参照してください。

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンスレーションの直前に入力します。

Verilog 制約を次のように指定します

```
(* IDELAY_GROUP = "group_name" *)
```

*group\_name* の詳細は、「UCF 構文」を参照してください。

Verilog 構文の詳細は、[「Verilog」](#)を参照してください。

## UCF 構文

IODELAY\_GROUP を定義する基本的な UCF 構文は次のとおりです。

```
INST "instance_name" IDELAY_GROUP = group_name;
```

*group\_name* は、IDELAY/IODELAY と IDELAYCTRL のセットをグループとして定義するために割り当てる名前です。

## IOSTANDARD

IOB は、基本的なマップ制約および合成制約です。

FPGA デバイスの場合

IOSTANDARD を使用して、I/O 規格を I/O プリミティブに割り当てます。

IOSTANDARD が設定されたコンポーネントは、SelectIO™ コンポーネントと同じ配置規則（バンク規則）に従わなければなりません。各アーキテクチャのバンク規則については、該当するライブラリ ガイドを参照してください。サポートされている I/O 規格の詳細は、該当するデバイスの [データシート](#) を参照してください。

Spartan®-3、Spartan-3A、Spartan-3E、Virtex®-4、Virtex-5 の場合、コンポーネントの SelectIO を使用するよりも、バッファ コンポーネントに IOSTANDARD を設定することをお勧めします。たとえば、IBUF\_HSTL\_III の代わりに、IOSTANDARD=HSTL\_III が設定された IBUF を使用します。

Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 の場合、差動信号標準は IBUF および OBUF には適用されず、IBUFDS、IBUFGDS、OBUFDS、OBUFTDS のみに適用されます。

CPLD デバイスの場合

CoolRunner™-II の I/O パッドに IOSTANDARD 制約を設定し、入力しきい値および出力 VCCIO 電圧を指定できます。サポートされる値については、該当するデバイスの [データシート](#) を参照してください。

ロケーション制約が指定されていない場合は、IOSTANDARD が設定された出力が CPLDFitter により自動的に同じバンクにまとめられます。

## アーキテクチャ サポート

IOSTANDARD は、すべての FPGA に適用されます。CPLD の場合は、CoolRunner-II デバイスのみに適用されます。

## 適用可能エレメント

どのデバイス ファミリーにどのエレメントを使用できるかは、ライブラリ ガイドを参照してください。詳細は、[データシート](#) を参照してください。

- ・ IBUF、IBUFG、OBUF、OBUFT
- ・ IBUFDS、IBUFGDS、OBUFDS、OBUFTDS
- ・ 出力電圧バンク

## 適用ルール

原則として信号またはネットには設定できませんが、信号またはネットがパッドに接続されている場合は例外です。パッドに接続されている場合、IOSTANDARD 制約は IOB インスタンス (IBUF、OBUF、IOB FF) に設定されているものと見なされます。デザイン エレメントに設定すると、そのデザイン エレメントの階層にあるすべての適用可能エレメントに適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ I/O プリミティブに設定します。
- ・ 属性名 : IOSTANDARD
- ・ 属性値 : *iostandard\_name*

詳細については、この制約の「UCF および NCF 構文」セクションを参照してください。

## VHDL 構文

VHDL 制約を次のように宣言します。



```
attribute iostandard: string;
```

VHDL 制約を次のように指定します。

```
attribute iostandard of {component_name|label_name}: {component| label} is "iostandard_name";
```

*iostandard\_name* の詳細については、この制約の「UCF および NCF 構文」セクションを参照してください。

CPLD デバイスの場合、パッド信号にも適用できます。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(* IOSTANDARD = "iostandard_name" *)
```

*iostandard\_name* については、UCF セクションを参照してください。

CPLD デバイスの場合、パッド信号にも適用できます。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

構文は次のとおりです。

```
INST "instance_name" IOSTANDARD= iostandard_name;
```

```
NET "pad_net_name" IOSTANDARD=iostandard_name;
```

*iostandard\_name* は、デバイスの[データシート](#)で指定されている I/O 規格の名前になります。

## XCF 構文

```
BEGIN MODEL "entity_name "
```

```
INST "instance_name" iostandard=string ;
```

```
NET "signal_name" iostandard=string ;
```

```
END;
```

## PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## PACE の構文

PACE は、CPLD の場合にのみ、主としてロケーション制約を I/O に設定する際に使用しますが、I/O 規格など特定の I/O プロパティの設定にも使用できます。PACE は、Project Navigator の [Processes] ウィンドウからアクセスできます。

詳細は、PACE ヘルプでエリア制約およびピンの編集に関するセクションを参照してください。PACE は、CPLD でのみサポートされており、FPGA ではサポートされていません。

## KEEP

KEEP は、高度なマップ制約かつ合成制約です。デザインのマップ時に、一部のネットが論理ブロックに含まれることがあります。ブロックに含まれたネットは、物理的なデザイン データベースには存在しなくなります。この問題は、ネットの各サイドに接続されているコンポーネントが同じロジック ブロックにマップされる場合などに発生する可能性があります。この後、ネットはそのコンポーネントを含むブロックに含まれます。KEEP を使用すると、これが回避できます。

FPGA の場合、KEEP は内部制約 NOMERGE に変換されます。そのため、インプリメンテーション ツールのメッセージには、KEEP ではなく、システム プロパティ NOMERGE が表示されます。TRUE および FALSE のほかに、合成 (XST) では SOFT も使用できます。この値を使用すると、指定したネットが保持されるだけでなく、合成後のネットリストでこのネットに NOMERGE 制約が付かなくなります。この結果、ネットが合成で保持されても、インプリメンテーション ツールではどのようにでも処理できるようになります。つまり、合成の場合にのみ KEEP=TRUE に設定され、インプリメンテーション ツールでは KEEP=FALSE に設定されるようなものです。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイ스에適用できます。

## 適用可能エレメント

信号

## 適用ルール

設定した信号に適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ ネットに設定します。
- ・ 属性名 KEEP
- ・ 属性値 : TRUE、FALSE、SOFT (XST のみ)

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute keep : string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

### Verilog 構文

この制約はインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(* KEEP = "{TRUE|FALSE |SOFT}" *)
```

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

次の構文で、ネット \$SIG\_0 が常に認識されるように指定します。

```
NET "$1I3245/$SIG_0" KEEP;
```

## XCF 構文

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name" keep={yes|no|true|false};
```

```
END;
```

メモ :

XCF ファイルでは、KEEP 制約の値はオプションで二重引用符 ( " ") で囲むことができます。値が SOFT の場合は、次のように必ず二重引用符 ( " ") で囲む必要があります。

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name"keep="soft";
```

```
END;
```

## KEEP\_HIERARCHY

KEEP\_HIERARCHY は、合成制約およびインプリメンテーション制約です。デザイン階層が合成で維持された場合、インプリメンテーションでもこの制約を使用して階層が維持され、この階層を含むシミュレーション ネットリストが生成されます。

XST では、よい結果を得るために、エンティティおよびモジュールすべてを最適化してデザインをフラットにすることがあります。この制約を true に設定すると、生成されたネットリストが階層構造であるため、デザインのエンティティとモジュールの階層およびインターフェイスを維持できます。

この制約は、HDL 合成で推論されたマクロではなく、HDL デザインで指定された階層ブロック (VHDL のエンティティ、Verilog のモジュール) に適用されます。このオプションでは、次の 3 つの値を使用できます。

- ・ **true**

HDL プロジェクトで記述されたデザイン階層を保持します。この値を合成に適用した場合、インプリメンテーションにも適用されます。

- ・ **false**

階層ブロックが 最上位モジュールにマージされます。

- ・ **soft**

デザイン階層は合成で維持されますが、インプリメンテーションで制約は適用されません。

CPLD デバイスの場合、true がデフォルトです。FPGA デバイスの場合、false がデフォルトです。

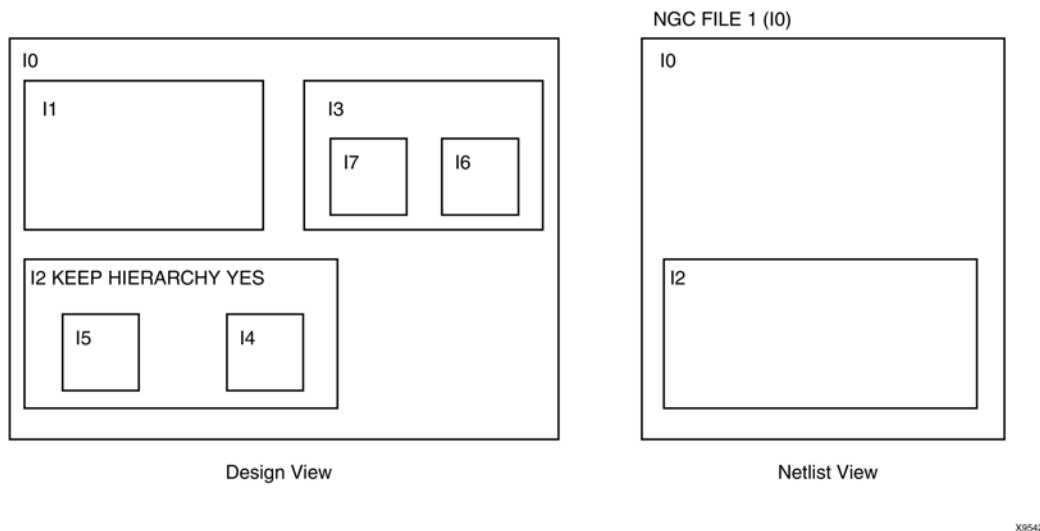
メモ : この制約は、XST では yes、true、no、false、および soft に設定できます。コマンドラインで使用する場合は、yes、no、および soft のみが使用できます。

一般的に、HDL デザインは階層ブロックの集合です。単純化された階層で別々に最適化が行われるため、デザインの階層を維持することで処理速度が向上します。また、コラプスや因数分解などの最適化のプロセスはロジック全体にグローバルに適用されるため、階層ブロックのマージによりフィットの結果が向上します（積項およびデバイス マクロセルの少量化、周波数の向上など）。

KEEP\_HIERARCHY を設定すると、ユーザー定義のデザイン ユニットの階層をフラットに設定できます。属性値は、true または false です。デフォルトで、ユーザーの階層は維持されます。

たとえば、エンティティまたはモジュール I2 に制約を設定した場合、I2 の階層は維持されたままで最後のネットリストに含まれますが、I2 の下にある I4 および I5 はフラットになります。また、I1、I3、I6、I7 も同様にフラットになります。

## 例



## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイスに適用できます。

## 適用可能エレメント

階層ブロックまたはシンボル ブロックを含む論理ブロックに適用します。

## 適用ルール

設定したエンティティまたはモジュールに適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ エンティティまたはモジュール シンボルに設定します。
- ・ 属性名 : KEEP\_HIERARCHY
- ・ 属性値 : TRUE、FALSE

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute keep_hierarchy : string;
```

VHDL 制約を次のように指定します。

```
attribute keep_hierarchy of architecture_name: architecture is {TRUE|FALSE|SOFT};
```

デフォルトは、FPGA デバイスの場合は false で、CPLD デバイスの場合は true です。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

インスタンスには、次のように指定します。

```
INST "instance_name" KEEP_HIERARCHY={TRUE|FALSE|SOFT};
```

## XCF 構文

この制約は、XST では yes、true、no、false、および soft に設定できます。コマンドラインで使用する場合は、yes、no、および soft のみが使用できます。

```
MODEL "entity_name" keep_hierarchy={yes|no|soft};
```

## Project Navigator からの設定

Project Navigator の [Process Properties] ダイアログ ボックスで [Synthesis Options] をクリックし、[Keep Hierarchy] でグローバルに設定できます。[Process Properties] ダイアログ ボックスを表示するには、[Sources] タブでデザインを選択し、[Processes] タブで [Synthesize - XST] を右クリックして [Process Properties] をクリックします。[Process Properties] ダイアログ ボックスで [Property display level] → [Advanced] をクリックします。

## KEEPER

KEEPER は、基本的なマップ制約です。出力ネットに対して KEEPER を指定すると、その出力ネットの値が保持されます。たとえば、ネットに対してロジック 1 を駆動すると、KEEPER はそのネットに対してウィーク/抵抗 1 を駆動します。その後、ネットドライバがトライステートになっても、KEEPER はウィーク/抵抗値 1 を駆動し続けます。

KEEPER 制約は、KEEPER コンポーネントと同じバンク規則に従う必要があります。バンク規則の詳細については、[ライブラリ ガイド](#)を参照してください。

KEEPER、PULLUP および PULLDOWN は、パッド ネットにのみ使用でき、INST には使用できません。

CoolRunner™-II デバイスでは、KEEPER と PULLUP を共に使用できません。

## アーキテクチャ サポート

KEEPER 制約は、すべての FPGA および CPLD の CoolRunner-II にのみ適用されます。

## 適用可能エレメント

トライステート入力/出力パッド ネット

## 適用ルール

ネットまたは信号には設定できませんが、ネットまたは信号がパッドに接続されている場合は例外です。この場合、KEEPER はパッド インスタンスに設定されているものと見なされます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 出力パッド ネットに設定します。
- ・ 属性名 : KEEPER
- ・ 属性値 : TRUE、FALSE

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute keeper: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

### Verilog 構文

この制約はインスタンス化の直前に入力します。

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

Verilog 構文の詳細は、[「Verilog」](#)を参照してください。

### UCF および NCF 構文

次の文は、ネットに対して KEEPER を使用するように I/O を設定しています。

```
NET "pad_net_name" KEEPER;
```

次の文は、KEEPER をグローバルに設定しています。

```
DEFAULT KEEPER = TRUE;
```

### XCF 構文

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name" keeper={yes|no|true|false};
```

```
END;
```

## LOC

LOC は、基本的な配置制約および合成制約です。

### FPGA デバイスの場合

LOC は、FPGA 内でデザイン エレメントを配置する位置 (ロケーション) を定義します。具体的には、FPGA チップ上のデザイン エレメントの絶対配置を指定します。絶対位置には、単一の位置、位置の範囲、または複数の位置のリストを指定できます。LOC の指定にはデザイン ファイルを使用しますが、制約ファイルの構文でも指定できます。

同じシンボルに対して複数の位置を指定するには、各位置をカンマで区切ります。これにより、指定された位置のいずれにもシンボルを配置できるようになります。また、デザイン エレメントまたはデザイン エレメントのグループを配置する領域も指定できます。

PlanAhead または FPGA Editor を使用すれば、有効なサイト名を簡単に識別できます。有効な名前は、ターゲットとなるデバイス タイプのファンクションです。ターゲット ロケーションを指定する構文を検索するには、FPGA Editor に空のデバイスをロードします。いずれかのブロックにカーソルを合わせてクリックすると、FPGA Editor のヒストリ エリア内で特定ブロックの位置を表示できます。位置に .I、.O、.T などのピン名を含めてはいけません。

LOC 制約は、複数の CLB、IOB、ソフト マクロなどのシンボルを使用するロジックに適用できます。LOC 制約をソフト マクロ シンボルに使用すると、位置の情報が下位レベルのロジックに渡されます。ロケーション制約は、LOC が有効な下位レベルのすべてのブロックに自動的に適用されます。

FPGA デバイスでは、スライスレベルでデカルト座標をベースとした XY 指定を使用します。スライスで位置を指定するには、SLICE\_XmYnを使用します。チップの左下にある CLB を XY 座標の基点 X0Y0 とします。X 値および Y 値は、CLB ごとに 2 つずつあります。X 値は 0 で始まり、CLB の行を右方向に向かって増えていきます。Y 値も 0 で始まり、CLB の列を上方向に向かって増えていきます。

XY 座標でスライスを指定する構文例を次に示します。

#### 単一の位置を指定する LOC 制約の例

SLICE_X0Y0	チップの左下にある CLB の最初 (一番下) のスライス
SLICE_X0Y1	チップの左下にある CLB の 2 番目のスライス
SLICE_X1Y0	チップの左下にある CLB の 3 番目のスライス
SLICE_X1Y1	チップの左下にある CLB の 4 番目 (一番上) のスライス
SLICE_X0Y2	左下の CLB の上にある CLB の最初のスライス
SLICE_X2Y0	左から 2 番目にある CLB の最初 (一番下) のスライス
SLICE_X2Y1	左から 2 つ目の CLB にある 2 番目のスライス
SLICE_X50Y125	SLICE_X0Y0 から上方向に 125 番目、右方向に 50 番目のスライス

FPGA のブロック RAM、乗算器を指定する場合にも SLICE とは異なる独自の仕様になります。したがって、位置の値は SLICE、RAMB、または MULT で始める必要があります。RAMB16\_X2Y3 にあるブロック RAM は、SLICE\_X2Y3 にあるフリップフロップと同じサイトには配置されません。また、MULT18X18\_X2Y3 にある乗算器は、SLICE\_X2Y3 にあるフリップフロップや RAMB16\_X2Y3 にあるブロック RAM と同じサイトには配置されません。

グローバル バッファ (BUFG) と DCM エレメントの位置の値は、配置できる位置の特定の物理的なサイト名となります。

LOC 制約を使用したピンの割り当ては、OPAD8 などのバス パッド シンボルではサポートされていません。

## FPGA デバイスのロケーション指定の種類

エレメントのタイプ	位置の例	意味
IOB	P12	IOB の位置 (チップ キャリア)
	A12	IOB の位置 (ピン グリッド)
	B、L、T、R	Spartan®-3、Spartan-3A、Spartan-3E の場合、IOB に適用され、エッジの位置 (下、左、上、右) を示します。
	LB、RB、LT、RT、BR、TR、BL、TL	Spartan-3、Spartan-3A、Spartan-3E の場合、IOB に適用され、ハーフ エッジの位置 (左下、右下など) を示します。
	Bank0、Bank1、Bank2、Bank3、Bank4、Bank5、Bank6、Bank7	すべての FPGA で IOB に適用され、ハーフ エッジ (バンク) を示します。
スライス	SLICE_X22Y3	すべての FPGA の SLICE_X22Y3 スライスの位置
ブロック RAM	RAMB16_X2Y56	Spartan-3、Spartan-3A、Spartan-3E、Virtex®-4 デバイスのブロック RAM の位置
	RAMB36_X2Y56	Virtex-5 デバイスのブロック RAM の位置
乗算器	MULT18X18_X#Y#	Spartan-3 および Spartan-3A デバイスの乗算器の位置
	DSP48a_X#Y#	Spartan-3A DSP の乗算器の位置
	DSP48_X#Y#	Virtex-4 および Virtex-5 デバイスの乗算器の位置
デジタル クロック マネージャ	DCM_X#Y#	Spartan-3、Spartan-3A、Spartan-3E デバイスのデジタル クロック マネージャ
	DCM_ADV_X#Y#	Virtex-4 および Virtex-5 デバイスのデジタル クロック マネージャ
位相ロック ループ (PLL)	PLL_ADV_X#Y#	すべての FPGA の PLL

次のように、ワイルドカード文字を使用すると、位置の範囲を指定できます。

SLICE_X*Y5	Y 座標が 5 にある FPGA デバイスのすべてのスライス
------------	--------------------------------

次の文字は、サポートされていません。

FPGA のグローバル バッファ、グローバル パッド、DCM の位置を表すワイルドカード文字

## CPLD デバイスの場合

CPLD デバイスでは、LOC=*pin\_name* 制約を PAD シンボルまたはパッド ネットに設定し、信号を特定ピンに割り当てます。PAD シンボルは、IPAD、OPAD、IOPAD、UPAD です。インスタンスがコラプスされていない場合、そのインスタンスまたはその出力ネットに対して LOC=FBnn 制約を使用すると、特定のファンクション ブロックまたはマクロセルにロジックまたはレジスタを割り当てることができます。

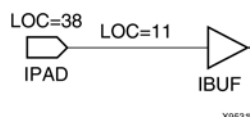


すべての内部インスタンスまたは出力パッドの LOC=FBnn<sub>mm</sub> 制約は、対応するロジックを CPLD 内の特定のファンクション ブロックまたはマクロセルに割り当てます。LOC が、マクロセルにマップされていないシンボルに設定されていたり、最適化により削除された場合、LOC は無視されます。

## LOC の優先順位

隣接する 2 つの LOC を入力パッドとそれに隣接するネットに設定する場合、ネットに設定された LOC が優先されます。たとえば、LOC=11 は LOC=38 よりも優先されます。

### LOC の優先順位の例



## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイ스에適用できます。

## 適用可能エレメント

どのデバイス ファミ리에どのエレメントを使用できるかは、ライブラリ ガイドを参照してください。詳細は、[データシート](#)を参照してください。

## 適用ルール

ネットまたは信号には設定できませんが、ネットまたは信号がパッドに接続されている場合は例外です。この場合、LOC はパッド インスタンスに設定されていると見なされます。

CPLD の場合、ネットまたは信号を駆動するすべての適用可能なエレメントに設定できます。

デザイン エレメントに設定すると、そのデザイン エレメントの階層にあるすべての適用可能なエレメントに適用されます。

## 構文

単一の位置

UCF 構文は次のとおりです。

```
INST "instance_name" LOC=location;
```

この場合、それぞれ次を表します。

location は、パーツ タイプに対して有効な位置になります。

単一の LOC 制約の構文例は、次のとおりです。

制約 (UCF 構文)	デバイス	説明
INST "instance_name" LOC=P12;		I/O を位置 P12 に配置します。
INST "instance_name" LOC=SLICE_X3Y2;	Spartan-3、Spartan-3A、Spartan-3E、Virtex®-4、Virtex-5 デバイス	スライスの XY 座標上にある SLICE_X3Y2 にロジックを配置します。
INST "instance_name" LOC=RAMB16_X0Y6;	Spartan-3、Spartan-3A、Spartan-3E、Virtex-4 デバイス	RAMB の XY 座標上にある RAMB16_X0Y6 にあるブロック RAM にロジックを配置します。
INST "instance_name" LOC=MULT18X18_X0Y6;	Spartan-3、Spartan-3A デバイス	MULT の XY 座標上にある MULT18X18_X0Y6 にある乗算器にロジックを配置します。
INST "instance_name" LOC=FIFO16_X0Y15;	Virtex-4	FIFO の XY 座標上の FIFO16_X0Y15 にある FIFO にロジックを配置します。
INST "instance_name" LOC>IDELAYCTRL_X0Y3;	Virtex-4、Virtex-5 デバイス	IDELAYCTRL_X0Y3 にある IDELAYCTRL にロジックを配置します。

複数の位置

**LOC=** *location1, location2 ,... , locationx*

個々の制約をカンマで区切ることで、1 つの要素に複数の位置を指定できます。複数の位置を指定した場合、PAR は指定された位置のいずれかを使用することもできます。複数の LOC 制約の例は、次のとおりです。

### 複数の位置を指定する LOC 制約の例

制約	デバイス	説明
INST "instance_name" LOC=SLICE_X2Y10, SLICE_X1Y10;	FPGA	スライスの XY 座標上にある SLICE_X2Y10 または SLICE_X1Y10 にロジックを配置します。

現在のところ、単一の制約を使用して、複数の要素を単一の位置または複数の位置に配置できません。

位置の範囲

UCF 構文は次のとおりです。

**INST "instance\_name" LOC=location : location {SOFT };**

バウンディング ボックスの 2 隅を指定することで、範囲を定義できます。Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 以外のアーキテクチャの場合、ロジックが配置される領域の左上隅および右下隅を指定します。FPGA デバイスの場合、左下隅と右上隅を指定します。指定した 2 隅は、コロン (:) で区切ります。

シンボルで表したロジックは、ボックス内のいずれかの位置に配置されます。デフォルトでは、制約は「ハード」条件と解釈され、ボックス内に配置されます。SOFT を指定すると、バウンディング ボックス外の位置の方がより良い結果が得られるような場合、PAR は制約を別の位置に配置します。範囲を指定する LOC 制約の例は、次のとおりです。

## 位置の範囲を指定する LOC 制約の例

制約	デバイス	説明
INST " <i>instance_name</i> " LOC=SLICE_X3Y5:SLICE_X5Y20;	FPGA	スライスの XY 座標上の SLICE_X3Y5 (左下隅) または SLICE_X5Y20 (右上隅) で囲まれた矩形領域にあるいずれかのスライスにロジックを配置します。

LOC の範囲には、SOFT も使用できます。AREA\_GROUP とは異なり、LOC はシンボルのパックに影響を与えません。厳密に言えば、LOC は PAR で使用される配置制約です。

LOC の構文 (CPLD デバイスの場合)

UCF 構文は次のとおりです。

```
INST "instance_name" LOC=pin_name;
```

または

```
INST "instance_name" LOC=FBff;
```

または

```
INST "instance_name" LOC=FB ffmm;
```

この場合、それぞれ次を表します。

- ・ *pin\_name* では、ピン名に *Pnn*、または行/列のピン名に *rc* を使用します。
- ・ *ff* は、ファンクション ブロックの番号です。
- ・ *mm* は、ファンクション ブロック内のマクロセルの番号です。

**FB*ff*** および **FB*ff**mm*** の 2 つの制約フォーマットは、出力および双方向ピンにのみ適用でき、入力ピンには適用できません。

1 つ目の制約フォーマットは次のようになります。

```
INST "instance_name" LOC=pin_name;
```

はすべての IO タイプに適用できます。

## 構文

FPGA デザインで各種ロジック エLEMENT に設定できる配置制約の例は、この制約の FPGA デバイス用の構文か、RLOC 制約の項目を参照してください。ロジック エLEMENT には、フリップフロップ、ROM、RAM、ブロック RAM、FMAP、BUFT、CLB、IOB、I/O、エッジ デコーダ、グローバル バッファが含まれます。

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ インスタンスに設定します。
- ・ 属性名 : LOC
- ・ 属性値 : *value*

有効な値についての詳細は、この制約の FPGA デバイス用および CPLD デバイス用の構文を参照してください。

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute loc: string;
```

VHDL 制約を次のように指定します。

```
attribute loc of {signal_name| label_name}: {signal |label} is "location";
```

バスに LOC を設定するには、次のように指定します。

```
attribute loc of bus_name: signal is " location_1 location_2 location_3...";
```

CPLD デバイスの場合、バスの一部のみに LOC を設定するには、次のように指定します。

```
attribute loc of bus_name: signal is "*" * location_1 * location_2...";
```

*location* の詳細は、この制約の FPGA デバイス用および CPLD デバイス用の構文を参照してください。

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

## Verilog 構文

この制約はインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(* LOC = " location" *)
```

バスに LOC を設定するには、次のように指定します。

```
(* LOC = "location_1 location_2 location_3... " *)
```

CPLD デバイスの場合、バスの一部のみに LOC を設定するには、次のように指定します。

```
(* LOC = "*" *location_1 location_2..." *)
```

*location* の詳細は、この制約の FPGA デバイス用および CPLD デバイス用の構文を参照してください。

Verilog 構文の詳細は、[「Verilog」](#)を参照してください。

## UCF および NCF 構文

次の文は、FLIP\_FLOPS の下にある各インスタンスを列 8 のいずれかの CLB に配置するように指定します。

```
INST "/FLIP_FLOPS/*" LOC=SLICE_X*Y8;
```

次の文は、MUXBUF\_D0\_OUT のインスタンスを IOB の位置 P110 に配置するように指定します。

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

次の文は、ネット DATA<1> を IOB の位置 P111 からパッドに接続するように指定します。

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

## XCF 構文

```
BEGIN MODEL " entity_name"
```

```
PIN "signal_name" loc=string ;
```

```
INST "instance_name" loc=string ;
```

```
END;
```

## PCF 構文

LOC は、LOCATE 制約を PCF ファイルに書き込みます。詳細は、「[LOCATE](#)」を参照してください。

## PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## PACE の構文

PACE は、主としてロケーション制約を I/O に設定する際に使用しますが、I/O 規格など特定の I/O プロパティの設定にも使用できます。PACE は、Project Navigator の [Processes] ウィンドウからアクセスできます。

詳細は、PACE ヘルプでエリア制約およびピンの編集に関するセクションを参照してください。PACE は、CPLD でのみサポートされており、FPGA ではサポートされていません。

## DCM 制約の例

このセクションは、FPGA のみに適用されます。

DCM は、UCF ファイルで LOC 制約を使用して設定できます。構文は次のようになります。

```
INST "instance_name" LOC = DCM_X AYB; (for all Spartan devices)
```

```
INST "instance_name" LOC = DCM_ADV_X AYB; (for Virtex-4, and Virtex-5 devices)
```

A は X 軸で、左下隅の値を 0 として、デバイスの右方向に向かって値が大きくなります。

B は Y 軸で、左下隅の値を 0 として、デバイスの上方向に向かって値が大きくなります。

例 :

```
INST "myinstance" LOC = DCM_X0Y0;
```

## フリップフロップ制約の例

フリップフロップ制約は、回路図または UCF ファイルで割り当てることができます。

回路図の場合は、フリップフロップに LOC 制約を設定します。制約は EDIF ネットリストに渡され、デザインのマップ後 PAR によって読み出されます。

次に、LOC 制約を回路図および UCF に適用する方法を示します。UCF の例にある /top-12/fdrd および /top-54/fdsd は、フリップフロップのインスタンス名です。

スライスをベースとした XY 座標で指定する場合

スライスをベースとした XY 座標で指定できるのは Spartan-3 および Virtex-4 以降のアーキテクチャのみです。

フリップフロップ制約は、特定のスライス、スライスの範囲、スライスの行および列に設定されます。

例 1

フリップフロップを SLICE\_X1Y5 に配置します。SLICE\_X0Y0 はデバイスの左下隅になります。

回路図	LOC=SLICE_X1Y5
UCF	INST “/top-12/fdrd” LOC=SLICE_X1Y5;

## 例 2

左下隅の SLICE\_X1Y1 と右上隅の SLICE\_X5Y7 で囲まれる矩形領域にフリップフロップを配置します。

回路図	LOC=SLICE_R1C1:SLICE_R5C7
UCF	INST “/top-12/fdrd” LOC=SLICE_X1Y1:SLICE_X5Y7;

## 例 3

Y 座標が 3 のスライスのすべての行にフリップフロップを配置します。X 値または Y 値の代わりにワイルドカード文字を使用すると、スライスの行全体または列全体を指定できます。

回路図	LOC=SLICE_X*Y3
UCF	INST “/top-12/fdrd/top-54/fdsd” LOC=SLICE_X*Y3;

## 例 4

SLICE\_X2Y4 または SLICE\_X7Y9 にフリップフロップを配置します。

回路図	LOC=SLICE_X2Y4,SLICE_X7Y9
UCF	INST “/top-54/fdsd” LOC=SLICE_X2Y4, SLICE_X7Y9;

例 4 の場合、各 LOC 制約をカンマで区切りながら繰り返して使用することで、1 つのエLEMENT に複数の位置を指定できます。複数の位置を指定した場合、PAR は指定された位置のいずれかを使用することもできます。

## 例 5

X 座標が 5 のスライスの列にフリップフロップを配置しません。X 値または Y 値の代わりにワイルドカード文字を使用すると、スライスの行全体または列全体を指定できます。

回路図	PROHIBIT=SLICE_X5Y*
UCF	CONFIG PROHIBIT=SLICE_X5Y*;

## I/O 制約の例

I/O 制約は、特定の IOB に設定できます。I/O 制約は、回路図から設定するか、または UCF ファイルを介して設定できます。

回路図の場合は、ターゲットのパッド シンボルに LOC 制約を設定します。制約はネットリスト ファイルに渡され、マップ後に PAR によって読み出されます。

UCF ファイルの場合は、一意なインスタンス名でパッドを識別します。次に、回路図と UCF (ユーザー制約ファイル) に LOC 制約を設定する方法を示します。この例では、I/O のインスタンス名は /top-102/data0\_pad と /top-117/q13\_pad です。さらに、ピン番号を使用して 1 つのピンにロックしています。

回路図	LOC=P17
UCF	INST “/top-102/data0_pad” LOC=P17;

ピン 17 の IOB に I/O を配置します。ピン グリッド アレイの場合は、B3 や T1 などのピン名を使用します。

## IOB 制約の例

I/O パッド、バッファ、レジスタは、個々の IOB の位置に割り当てることができます。IOB の位置は、対応するパッケージピンの指定によって識別されます。

IOB 制約は、次のような形式で使用できます。LOC= にピンの位置を指定してください。INFF8 などのように、シンボルがソフト マクロを表し、そのマクロに I/O エLEMENT のみが含まれる場合、マクロに含まれるすべての I/O エLEMENT に LOC 制約が適用されます。指定した I/O エLEMENT が指定した位置に収まらない場合は、エラーが発生します。

次の文は、I/O エLEMENT を位置 P13 に配置するように指定します。PGA パッケージの場合は、B3 などのように文字と数字で指定します。

**INST " instance\_name" LOC=P13;**

マップで特定の IOB を使用しないように設定できます。また、半専用のコンフィギュレーション ピンからユーザー I/O 信号を分離させることができます。このような PROHIBIT 制約は、UCF ファイルでのみ割り当てることができます。

次の例に示すように、IOB を禁止するには、CONFIG キーワードに続けて PROHIBIT 制約を指定します。

回路図	なし
UCF	CONFIG PROHIBIT=p36, p37, p41;

ピン 36、37、41 にある IOB にはユーザー I/O を配置しません。ピングリッドアレイの場合は、D14、C16、H15 などのピン名を使用します。

## マップ制約の例 (FMAP)

マップ制約は、CLB へのロジックのマップを制御します。この制約は 2 種類に分類されます。回路図で設定できる FMAP と回路図または制約ファイルで設定できる LOC 制約があります。

FMAP は、ファンクション ジェネレータへのロジックのマップを制御します。このシンボルは回路図上のロジックを定義しませんが、回路図上の別の場所にあるロジック部分をファンクション ジェネレータにマップする方法を指定します。

FMAP シンボルは、4 入力の F ファンクション ジェネレータへのマップを定義します。

FMAP シンボルの場合、CLBMAP プリミティブと同じように、LOC 制約のほかに MAP=PUC および MAP=PUO がサポートされています (現在、ピンのロックはサポートされていません。MAP=PLC と MAP=PLO は、それぞれ MAP=PUC と MAP=PUO に変換されます)。

例 1

FMAP シンボルを行 7、列 3 の SLICE に配置します。

回路図	LOC=SLICE_X7Y3
UCF	INST "\$I1323" LOC=SLICE_X2Y4, SLICE_X3Y4;

例 2

FMAP シンボルを行 2、列 4 または行 3、列 4 のいずれかにある SLICE に配置します。

回路図	LOC=SLICE_X2Y4, SLICE_X3Y4
UCF	INST "top/dec0011" LOC=CLB_R2C4, CLB_R3C4;

例 3

FMAP シンボルを左上隅の SLICE X5Y5 と右上隅の SLICE X10Y8 で囲まれた領域に配置します。



回路図	LOC=SLICE_X5Y5:SLICE_X10Y8
UCF	INST "\$3I27" LOC=SLICE_X5Y5:SLICE_X10Y8;

## 乗算器制約の例

このセクションは、FPGA に適用されます。

乗算器の制約は、回路図から設定するか、または UCF ファイルを介して設定できます。回路図の場合は、LOC 制約を乗算器シンボルに設定します。制約はネットリスト ファイルに渡され、マップ後に PAR により読み込まれます。LOC 制約の設定については、該当するアプリケーションのユーザー ガイドを参照してください。制約ファイルの場合、乗算器は一意的なインスタンス名によって識別されます。

FPGA の乗算器は、スライス、ブロック RAM の場合とは異なる XY 座標仕様で指定します。Spartan-3、Spartan-3A、Spartan-3E の場合は、MULT18X18\_X#Y# を、Spartan-3A DSP の場合は DSP48a\_X#Y# を、Virtex-4 および Virtex-5 の場合は DSP48\_X#Y# を使用します。ここでは、XY 座標の値が乗算器のグリッド アレイに相当します。MULT18X18\_X0Y1 にある乗算器は、SLICE\_X0Y1 にあるフリップフロップや RAMB16\_X0Y1 にあるブロック RAM と同じサイトには配置されません。

たとえば、2 列の乗算器があるデバイスがあるとします。各列には 2 つの乗算器が含まれ、一方の列はチップの右側に、もう一方の列は左側にあります。左下隅にある乗算器は MULT18X18\_X0Y0 です。乗算器は 2 列しかないため、右上隅にある乗算器は MULT18X18\_X1Y1 です。

回路図	LOC=MULT18X18_X0Y0
UCF	INST "/top-7/rq" LOC=MULT18X18_X0Y0;

## ROM 制約の例

メモリの制約は、回路図から設定するか、または UCF ファイルを介して設定できます。

回路図の場合は、LOC 制約をメモリシンボルに設定します。制約はネットリスト ファイルに渡され、マップ後に PAR によって読み出されます。LOC 制約の適用方法の詳細は、そのアプリケーションのユーザー ガイドを参照してください。

制約ファイルの場合、メモリは一意的なインスタンス名によって識別されます。タイプが ROM のメモリ インスタンスを入力ファイルで 1 つ以上指定できます。16 X 1 または 32 X 1 より大きいすべてのメモリ マクロは、ネットリスト ファイル内でこれらの基本エレメントに分割されます。

次の例では、ROM プリミティブのインスタンス名は /top-7/rq です。

スライスをベースとした XY 座標で指定する場合

スライスをベースとした XY 座標で指定できるのは Spartan-3 および Virtex-4 以降のアーキテクチャです。ROM 制約は、特定のスライス、スライスの範囲、スライスの行または列に設定できます。

例 1

SLICE\_X1Y1 にメモリを配置します。SLICE\_X1Y1 は、デバイスの左下隅になります。16 X 1 または 32 X 1 メモリには、SLICE 制約を 1 つだけ設定できます。

回路図	LOC=SLICE_X1Y1
UCF	INST "/top-7/rq" LOC=SLICE_X1Y1;

例 2

SLICE\_X2Y4 または SLICE\_X7Y9 のいずれかにメモリを配置します。



回路図	LOC=SLICE_X2Y4, SLICE_X7Y9
UCF	INST “/top-7/rq” LOC=SLICE_X2Y4, SLICE_X7Y9;

## 例 3

X 座標が 5 のスライスの列にメモリを配置しません。X 座標または Y 座標にワイルドカード文字を使用すると、スライスの行全体または列全体を指定できます。

回路図	PROHIBIT SLICE_X5Y*
UCF	CONFIG PROHIBIT=SLICE_X5Y*;

## ブロック RAM (RAMB) 制約の例

このセクションは、FPGA に適用されます。

ブロック RAM の制約は、回路図から設定するか、または UCF ファイルを介して設定できます。回路図の場合は、LOC 制約をブロック RAM シンボルに設定します。制約は、ネットリスト ファイルに渡され、マップ後に PAR によって読み込まれます。LOC 制約の設定については、該当するアプリケーションのユーザー ガイドを参照してください。制約ファイルの場合、メモリは一意なインスタンス名によって識別されます。

## Spartan-3 以降のデバイス

FPGA のブロック RAM は、スライス、乗算器の場合とは異なる XY 座標仕様で指定します。具体的には、RAMB16\_XmYn を使用します。ここでは、XY 座標の値がブロック RAM のグリッド アレイに相当します。RAMB16\_X0Y1 にあるブロック RAM は、SLICE\_X0Y1 にあるフリップフロップと同じサイトには配置されません。

たとえば、2 列のブロック RAM があるデバイスがあるとします。各列には 2 つのブロックが含まれ、一方の列はチップの右側に、もう一方の列は左側にあります。左下隅にあるブロック RAM は、RAMB16\_X0Y0 です。ブロック RAM は 2 列しかないため、右上隅にあるブロックは RAMB16\_X1Y1 になります。

回路図	LOC=RAMB16_X0Y0 (Virtex-5 以外のすべての FPGA) LOC=RAMB36_X0Y0 (Virtex-5)
UCF	INST “/top-7/rq” LOC=RAMB16_X0Y0;

## スライス制約の例

このセクションは、すべての FPGA に適用されます。現在のところ、スライスをベースとした XY 座標で指定できるのはこれらのみです。

単一のスライス位置、スライス位置のリスト、または スライス位置の矩形ブロックにソフト マクロとフリップフロップを割り当てることができます。

スライスの位置は、固定位置または位置の範囲で表すことができます。固定位置を表すには、次の構文を使用します。

**SLICE\_XmY n**

この場合、それぞれ次を表します。

*m* および *n* は、それぞれ X 座標と Y 座標の値です。

また、ターゲットとなるデバイスで、値はスライス番号以下でなければなりません。一番上から一番下までの位置の範囲を表すには、次の構文を使用します。

**SLICE\_X mYn:SLICE\_XmY n**

スライス制約の形式

スライス制約は、次のような形式で指定します。LOC= にスライスの位置を指定してください。ターゲットとなるシンボルがソフト マクロを表す場合、そのマクロに含まれるすべての該当シンボル (フリップフロップ、マップ) に対して LOC 制約が適用されます。指定したロジックが指定したブロックに収まらない場合は、エラーが発生します。

#### スライス制約の例 1

次の UCF 構文は、指定されたスライスにロジックを配置します。

```
INST "instance_name" LOC=SLICE_X133Y10;
```

#### スライス制約の例 2

次の UCF 構文は、スライスの最初の列にロジックを配置するように指定します。アスタリスク (\*) は、ワイルドカードです。

```
INST "instance_name" LOC=SLICE_X0Y*;
```

#### スライス制約の例 3

次の UCF 構文は、指定された 3 つのスライスのすべてにロジックを配置するように指定します。LOC 制約の構文の順番に意味はありません。

```
INST "instance_name" LOC=SLICE_X0Y3, SLICE_X67Y120, SLICE_X3Y0;
```

#### スライス制約の例 4

次の UCF 構文は、最初に指定される左下隅のスライスと 2 番目に指定される右上隅のスライスによって定義される矩形ブロック内にロジックを配置します。

```
INST "instance_name" LOC=SLICE_X3Y22:SLICE_X10Y55;
```

## スライス禁止制約

特定のスライスや、スライスの範囲、あるいはスライスの行または列を PAR が使用しないように指定できます。このような PROHIBIT 制約は、UCF ファイルでのみ割り当てることができます。次の例に示すように、スライスを禁止するにはデザイン レベルで PROHIBIT 制約を指定します。

#### スライス禁止制約の例 1

SLICE\_X0Y0 にロジックを配置しません。SLICE\_X0Y0 は、デバイスの左下隅になります。

回路図	なし
UCF	CONFIG PROHIBIT=SLICE_X0Y0;

#### スライス禁止制約の例 2

左下隅の SLICE\_X2Y3 と右上隅の SLICE\_X10Y10 で囲まれた矩形領域にロジックを配置しません。

回路図	なし
UCF	CONFIG PROHIBIT=SLICE_X2Y3:SLICE_X10Y10;

#### スライス禁止制約の例 3

X 座標の 3 にあるスライスにロジックを配置しません。これは、禁止されたスライスの列を指定しています。X 座標または Y 座標にワイルドカード文字を使用すると、スライスの行全体または列全体を指定できます。

回路図	なし
UCF	CONFIG PROHIBIT=SLICE_X3Y*;

#### スライス禁止制約の例 4

SLICE\_X2Y4 または SLICE\_X7Y9 のいずれにもロジックを配置しません。

回路図	なし
UCF	CONFIG PROHIBIT=SLICE_X2Y4, SLICE_X7Y9;

## LOCATE

LOCATE は、基本的な配置制約で、単一の位置、複数の位置、または位置の範囲を指定できます。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

- ・ CLB 数
- ・ IOB
- ・ DCM 数
- ・ クロック ロジック
- ・ マクロ

## 適用ルール

マクロに設定されている場合、そのマクロのすべてのエレメントに適用されます。プリミティブに設定されている場合は、プリミティブ全体に適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

#### PCF 構文

##### 単一の位置または複数の位置

```
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];
```

##### ロケーションの範囲

```
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];
```

この場合、次を表します。

- ・ *site\_name* は、コンポーネント サイト (CLB または IOB の位置) です。
- ・ *site\_item* は、次のいずれかです。
  - **SITE** "*site\_name*"
  - **SITEGRP** "*site\_group\_name*"
- ・ LEVEL *n* の *n* は、0、1、2、3、または 4 です。

## LOCK\_PINS

LOCK\_PINS は、インプリメンテーション ツールで LUT シンボルに付いているピンがスワップしないように指定します。この制約は、CPLD デザインで既存のピン配置を維持するために使用する Project Navigator のピン固定機能とは異なります。混同しないように注意してください。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

LUT シンボルの特定インスタンスのみに設定できます。

## 適用ルール

単一の LUT インスタンスのみに適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute lock_pins:  string;
```

VHDL 制約を次のように指定します。

```
attribute lock_pins of {component_name|label_name} : {component|label} is "all";
```

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

```
(* LOCK_PINS = "all" *)
```

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

### UCF および NCF 構文

#### 識別子を使用しない場合

```
INST "XSYM1" LOCK_PINS;
```

すべての属性を使用する場合

```
INST "XSYM1" LOCK_PINS='ALL';
```

ピン割り当てリストを使用する場合

```
INST I_589 LOCK_PINS=I0:A2;
```

```
INST I_894 LOCK_PINS=I3:A1,I2:A4;
```

```
INST tvAgy LOCK_PINS=I0:A4,I1:A3,I2:A2,I3:A1;
```

## LUTNM

LUTNM 制約は、論理シンボルのグループ化の制御に使用されます。グループ化されたシンボルは、Virtex®-5 FPGA アーキテクチャの LUT サイトにまとめられます。値は文字列で、条件を満たす 2 つのシンボルに設定され、デザイン内ではこれらのシンボルのみに使用されます。シンボルは、SLICE コンポーネント内で共有された LUT サイト内でインプリメントされます。

この制約の機能は、[BLKNM](#) 制約に似ています。

## アーキテクチャ サポート

この制約は、Virtex-5 デバイスにのみ適用できます。

## 適用可能エレメント

LUTNM 制約はデザイン内で特定の 2 つのシンボルのみに適用されます。両方のシンボルの、重複しない入力ピン数の合計が 5 を超えない場合は、5 入力以下のファンクション ジェネレータ シンボル (LUT、ROM、または RAM) 2 つに設定できます。両方のシンボルの、重複しない入力ピン数の合計が 6 入力を超えない場合は、6 入力の読み込み専用ファンクション ジェネレータ シンボル (LUT6、ROM64) と 5 入力の読み込み専用シンボル (LUT5、ROM32) に設定できます。ただし、6 入力シンボル プログラムの下位 32 ビットは 5 入力シンボル プログラムの 32 ビットすべてと一致している必要があります。

## 適用ルール

LUTNM 制約はデザイン内で特定の 2 つのシンボルのみに適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効なエレメントまたはシンボル タイプに設定します。
- ・ 属性名 : LUTNM
- ・ 属性値 : <user\_defined>

### VHDL 構文

LUTNM は、アーキテクチャ宣言と最上位 VHDL ファイルの begin 文の間で次のように宣言します。

```
attribute LUTNM: string;
```

宣言後、次のように指定します。

```
attribute LUTNM of {LUT5_instance_name}: label is "value";
```

*value* 2 つのエレメントのグループに選択した任意の名前になります。

例 :

```
architecture MY_DESIGN of top is
attribute LUTNM: string;
    attribute LUTNM of LUT5_inst1: label is "logic_group1";
    attribute LUTNM of LUT5_inst2: label is "logic_group1";
begin
-- LUT5: 5-input Look-Up Table
-- Virtex-5
-- Xilinx HDL Libraries Guide version 8.2i
LUT5_inst1 : LUT5
generic map (
    INIT => X"a49b44c1")
port map (
    O => aout, -- LUT output (1-bit)
    I0 => d(0), -- LUT input (1-bit)
    I1 => d(1), -- LUT input (1-bit)
    I2 => d(2), -- LUT input (1-bit)
    I3 => d(3), -- LUT input (1-bit)
    I4 => d(4) -- LUT input (1-bit)
);
-- End of LUT5_inst1 instantiation
-- LUT5: 5-input Look-Up Table
-- Virtex-5
-- Xilinx HDL Libraries Guide version 8.2i
LUT5_inst2 : LUT5
generic map (
    INIT => X"649d610a")
port map (
    O => bout, -- LUT output (1-bit)
    I0 => d(0), -- LUT input (1-bit)
    I1 => d(1), -- LUT input (1-bit)
    I2 => d(2), -- LUT input (1-bit)
    I3 => d(3), -- LUT input (1-bit)
    I4 => d(4) -- LUT input (1-bit)
);
-- End of LUT5_inst2 instantiation
END MY_DESIGN;
```

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

最上位の Verilog コードの port 宣言の前に、次のように記述します。

**(\* LUTNM = "value" \*)**

*value* 2 つのエレメントのグループに選択した任意の名前になります。

例 :

```
// LUT5: 5-input Look-Up Table
// Virtex-5
// Xilinx HDL Libraries Guide version 8.2i
(* LUTNM="logic_group1" *) LUT5 #(
  .INIT(32'h49b44c1)
) LUT5_inst1 (
  .O(aout), // LUT output (1-bit)
  .I0(d[0]), // LUT input (1-bit)
  .I1(d[1]), // LUT input (1-bit)
  .I2(d[2]), // LUT input (1-bit)
  .I3(d[3]), // LUT input (1-bit)
  .I4(d[4]) // LUT input (1-bit)
);
// End of LUT5_inst1 instantiation
// LUT5: 5-input Look-Up Table
// Virtex-5
// Xilinx HDL Libraries Guide version 8.2i
(* LUTNM="logic_group1" *) LUT5 #(
  .INIT(32'h649d610a)
) LUT5_inst2 (
  .O(bout), // LUT output (1-bit)
  .I0(d[0]), // LUT input (1-bit)
  .I1(d[1]), // LUT input (1-bit)
  .I2(d[2]), // LUT input (1-bit)
  .I3(d[3]), // LUT input (1-bit)
  .I4(d[4]) // LUT input (1-bit)
);
// End of LUT5_inst2 instantiation
```

Verilog 構文の詳細については、「[Verilog](#)」を参照してください。

## UCF および NCF の構文

出力ポートまたは双方向ポートに設定します。

**INST "LUT5\_instance\_name" LUTNM="value";**

*value* は 2 つのエレメントのグループ化に使用する任意の名前です。

例 :

**INST "LUT5\_inst1" LUTNM="logic\_group1";**

**INST "LUT5\_inst2" LUTNM="logic\_group1";**

## MAP

MAP は、高度なマップ制約です。FMAP に適用すると、マップでロジックにほかのファンクションのマージを許可するか、ピンの交換を許可するかを指定できます。ほかのファンクションのマージが許可されると、スペースが許す限り、CLB 内にほかのロジックも配置できます。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

FMAP

## 適用ルール

設定したデザイン エレメントに適用されます。

## MAP の構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF および NCF 構文

UCF 構文は次のとおりです。

```
INST "instance_name" MAP=[PUC | PUO | PLC | PLO] ;
```

値にはそれぞれ、次のような意味があります。

#### PUC

CLB ピンがアンロック (U) で、CLB がクローズ (C) です。ソフトウェアが CLB のピン間で信号を交換できますが、CLB からロジックを追加または削除することはできません。

#### PUO

CLB ピンがアンロック (U) で、CLB がオープン (O) です。ソフトウェアが CLB のピン間で信号を交換でき、CLB からロジックを追加または削除することもできます。

#### PLC

CLB ピンがロック (L) で、CLB がクローズ (C) です。ソフトウェアが CLB のピン間で信号を交換できず、CLB からロジックを追加または削除することもできません。

#### PLO

CLB ピンがロック (L) で、CLB がオープン (O) です。ソフトウェアが CLB のピン間で信号を交換できませんが、CLB からロジックを追加または削除することはできます。

デフォルトは PUO です。現時点では、PUC と PUO のみが有効です。PLC は PUC に、PLO は PUO に変換されます。

次の構文で、ピンのスワップを許可し、オリジナルのマップで定義されたロジック以外はファンクション ジェネレータにマップされないように設定します。

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

## MAX\_FANOUT

MAX\_FANOUT を使用すると、ネットまたは信号のファンアウト数を制限できます。制約の値により、XST および MAP でネットのファンアウトが制限されます。この値は整数 (XST のみ) または REDUCE (MAP のみ) のいずれかにできます。

### XST の MAX\_FANOUT

XST のデフォルト整数値は、次の表を参照してください。MAX\_FANOUT は、XST でグローバルにもローカルにも設定できます。

### 最大ファンアウトのデフォルト値

デバイス	デフォルト値
Spartan®-3, Spartan-3E, Spartan-3A, Spartan-3A D	500
Virtex®-4	500
Virtex-5	100000 (10 万)

ファンアウトが大きいと配線で問題を生じることがあるため、XST ではゲートを複製したり、バッファを挿入することでファンアウト数が制限されます。これは技術面での限界ではなく、XST の基準です。この制限は、特に 30 未満に設定されている場合などは、厳密に適用されるとは限りません。



ほとんどの場合、ファンアウトが大きいネットを駆動するゲートを複製することでファンアウト数が制限されます。ゲートを複製できない場合は、バッファが挿入されます。これらのバッファには、NGC ファイルで **キープ (KEEP)** 属性が設定され、インプリメンテーションでの最適化により削除されることはありません。レジスタの複製オプションを no に設定している場合は、バッファのみを使用してフリップフロップおよびラッチのファンアウト数が制限されます。

MAX\_FANOUT は、グローバルに設定できますが、エンティティやモジュール、指定した信号ごとに設定して、最大ファンアウトを制御できます。

実際のネット ファンアウトが MAX\_FANOUT 値よりも小さい場合は、MAX\_FANOUT の設定方法によって XST のビヘイビアが異なります。

- MAX\_FANOUT の値を ISE® Design Suite またはコマンド ラインを使用して設定するか、特定の階層ブロックに適用した場合、XST ではこの値が基準として解釈されます。
- MAX\_FANOUT を特定のネットに設定した場合は、ロジックは複製されません。ネットに設定した場合は、XST で最適なタイミング最適化が行われなかったことがあります。

たとえば、実際のファンアウトが 80 で、MAX\_FANOUT 値が 100 に設定されたネットをクリティカル パスが通過しているとします。MAX\_FANOUT を ISE Design Suite で設定している場合は、XST がタイミングを向上しようとしてネットを複製する場合があります。MAX\_FANOUT をネットに設定している場合は、ロジックは複製されません。

MAP の MAX\_FANOUT

MAX\_FANOUT 制約を使用すると、レジスタまたはゲートの両方ともまたはいずれかを複製することにより、マップでファンアウトを制限させることもできます。このためには、マップのレジスタ複製オプション (-register\_duplication) をイネーブルにし、MAX\_FANOUT 制約をネットに (ローカルに) 適用する必要があります。マップ中に使用する場合、値には REDUCE しか使用できません。MAX\_FANOUT = "REDUCE" の場合、フィットで問題を引き起こすことなく、パフォーマンスに改良があると判断される場合に、マップでファンアウトが制限されます。MAX\_FANOUT = "REDUCE" でファンアウトの削減が実際に行われたかどうかは、マップで生成される物理合成レポート (\*PSR) を確認するとわかります。

## アーキテクチャ サポート

すべての FPGA に適用できます。CPLD には適用できません。

## 適用可能エレメント

値が整数の場合は、グローバルに適用するか、VHDL エンティティ、Verilog モジュール、または信号に適用します。

値が REDUCE の場合は、信号にのみ適用します。

## 適用ルール

設定したエンティティ、モジュール、または信号に適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### VHDL 構文

使用する前に、次の構文を使って宣言します。

```
attribute max_fanout: string;
```

After declaring Max Fanout, specify the VHDL constraint:

```
attribute max_fanout of {signal_name | entity_name}: {signal | entity} is "integer";
```

## Verilog 構文

次をモジュール宣言または信号宣言の直前に入力します。

```
(* max_fanout = "integer" *)
```

## XCF 構文 1

```
MODEL "entity_name" max_fanout=integer;
```

## XCF 構文 2

```
BEGIN MODEL "entity_name"  
NET "signal_name" max_fanout=integer;  
END;
```

## XST コマンド ライン構文

run コマンドの **-max\_fanout** オプションでグローバルに設定します。

```
-max_fanout integer
```

## ISE からの設定

ISE の [Process Properties] ダイアログ ボックスの [Xilinx Specific Options] ページにある [Max Fanout] でグローバルに設定します。

## UCF 構文

マップのレジスタ複製オプションと共に使用される場合、MAX\_FANOUT 制約は UCF ファイルで次のように指定します。

```
NET "signal_name" max_fanout=REDUCE;
```

## MAXDELAY

MAXDELAY 制約は、ネットの最大許容遅延を定義します。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

ネット

## 適用ルール

設定したネットに適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ ネットに設定します。
- ・ 属性名 : MAXDELAY
- ・ 属性値 : *value units*  
*value* 遅延時間です。  
*units* micro、ms、ns、ps などの単位です。

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute maxdelay: string;
```

VHDL 制約を次のように指定します。

```
attribute maxdelay of signal_name: signal is "value [units]";
```

*value* は、正の整数です。

*units* は、ps、ns、micro、ms、GHz、MHz、kHz などの単位です。デフォルトは ns です。

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

## Verilog 構文

次をモジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(*MAXDELAY = "value [units]" *)
```

*value* は正の整数です。

*units* は、ps、ns、micro、ms、GHz、MHz、kHz などの単位です。デフォルトは ns です。

Verilog 構文の詳細は、[「Verilog」](#)を参照してください。

## UCF および NCF 構文

```
NET "net_name" MAXDELAY=value units;
```

*value* は遅延時間です。

*units* は micro、ns、ms、ps などの単位です。

次の文は、10 ナノ秒の最大遅延をネット \$SIG\_4 に指定します。

```
NET "$1I3245/$SIG_4" MAXDELAY=10 ns;
```

## PCF 構文

```
item MAXDELAY = maxvalue [PRIORITY integer];
```

*item* は次のいずれかになります。

- ・ **ALLNETS**
- ・ **NET** *name*
- ・ **TIMEGRP** *name*
- ・ **ALLPATHS**
- ・ **PATH** *name*
- ・ *path specification*

*maxvalue* は次のいずれかになります。

- ・ 時間を表す数字 (単位: micro、ms、ps、ns)
- ・ 周波数を表す数字 (単位: GHz、MHz、KHz)
- ・ TSidentifier

### Constraints Editor からの設定

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Timing Constraints] の下の [Exceptions] で [Nets] をダブルクリックし、ダイアログ ボックスにアクセスします。

### FPGA Editor からの設定

すべてのパスまたはネットに制約を設定するには、[File] → [Main Properties] → [Global Physical Constraints] タブをクリックします。

選択したパスまたはネットに制約を設定するには、配線済みネットを選択して [Edit] → [Properties of Selected Items] → [Physical Constraints] タブをクリックします。

## MAXPT

MAXPT は、CPLD のみに適用される高度な制約です。MAXPT を使用すると、制約を設定したノード内でロジックをコラプスする際にフィルタで利用できる積項の最大数を指定できます。この値は、Project Navigator の [Collapsing Pterm Limit] で設定されている値より優先されます。

## アーキテクチャ サポート

この制約は、CPLD デバイスにのみ適用できます。

## 適用可能エレメント

信号

## 適用ルール

設定した信号に適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute maxpt: integer;
```

VHDL 制約を次のように指定します。

```
attribute maxpt of signal_name: signal is "integer";
```

*integer* 正の整数です。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(* MAXPT = "integer" *)
```

*integer* 正の整数です。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

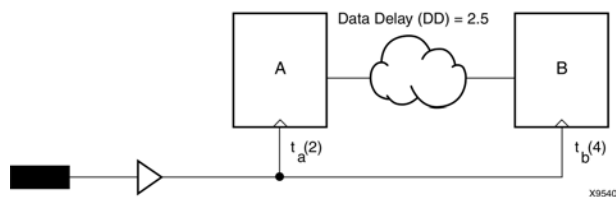
## UCF および NCF 構文

```
Net "signal_name" maxpt=integer;
```

## MAXSKEW

MAXSKEW は、ネット上の最大スキューを制御するために使用するタイミング制約です。この制約はローカル クロックまたはグローバル クロック ネットワークに含まれないクロックのスキューを制御するためによく使用されます。この制約は、グローバル クロック ネットワークには必要ないので、使用しないでください。<:imk 2>スキューとは、ネットで駆動されるすべてのロードの遅延の差です。この制約では、ネットで駆動されるすべてのロードが識別されるので、論理的な接続のないロード間のスキューがレポートされることもあります。ネットで許容される最大スキューを制御するには、MAXSKEW をネットに直接適用します。MAXSKEW が定義する内容を正確に把握しておくことが大切です。次に、例を示します。

### MAXSKEW



この図で、 $t_a(2)$  はレジスタ A クロックに対する最大遅延が 2ns であることを示し、 $t_b(4)$  はレジスタ B クロックに対する最大遅延が 4ns であることを示します。MAXSKEW は、最大値  $t_b$  から最大値  $t_a$  を引いた値を定義します。この場合は、 $4 - 2 = 2$  となります。

セットアップおよびホールド タイムの解析に、ネットの相対最小遅延が使用されることがあります。相対最小遅延を使用するネットワーク リソースに MAXSKEW 制約を適用すると、スキューを計算する際に相対最小遅延が考慮されます。

制約に大き過ぎる値や厳密過ぎる値を設定すると、PAR のランタイムが長くなります。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

ネット

## 適用ルール

設定したネットに適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ ネットに設定します。
- ・ 属性名 : MAXSKEW
- ・ 属性値 : *allowable\_skew* [units]  
*allowable\_skew* はタイミング要件です。  
*units* は ms、micro、ns、ps などの単位です。デフォルトは ns です。

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute maxskew: string;
```

VHDL 制約を次のように指定します。

```
attribute maxskew of signal_name: signal is "allowable_skew[units]";
```

*allowable\_skew* はタイミング要件です。

*units* は ms、micro、ns、ps などの単位です。デフォルトは ns です。

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(* MAXSKEW = "allowable_skew[units]" *)
```

*allowable\_skew* はタイミング要件です。

*units* は ms、micro、ns、ps などの単位です。デフォルトは ns です。

Verilog 構文の詳細は、[「Verilog」](#)を参照してください。

### UCF および NCF 構文

```
NET "net_name" MAXSKEW=allowable_skew[units];
```

*allowable\_skew* はタイミング要件です。

*units* は ms、micro、ns、ps などの単位です。デフォルトは ns です。

次の文は、3ns の最大スキューをネット \$SIG\_6 に指定します。

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

## Constraints Editor の構文

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リストボックスの [Timing Constraints] の下の [Exceptions] で [Nets] をダブルクリックし、ダイアログ ボックスにアクセスします。

## FPGA Editor の構文

制約を設定するには、FPGA Editor のメイン ウィンドウで [Edit] → [Properties of Selected Items] をクリックします。配線済みのネットを選択し、[Physical Constraints] タブから MAXSKEW を設定します。

## MIODELAY\_GROUP

MIODELAY\_GROUP は、2 つ以上の IODELAY\_GROUP を 1 つのマスタ IODELAY グループにまとめるデザイン インプリメンテーション制約で、IDELAYCTRL を自動的に複製して配置させることができます。

## アーキテクチャ サポート

MIODELAY\_GROUP 制約は、Virtex®-4 および Virtex-5 アーキテクチャで使用できます。Virtex-4 の場合、この制約は MAP の [Timing Driven Pack and Placement] オプションを使用した場合にのみサポートされます。

## 適用可能エレメント

MIODELAY\_GROUP は、定義済みの複数の IODELAY\_GROUP に適用できます。

## 適用ルール

MIODELAY\_GROUP は、既存の IODELAY\_GROUP に適用する制約で、元の IODELAY\_GROUP に属するデザイン エレメントすべてに適用されます。ネット、信号、またはピンには設定できません。

## 構文

IODELAY\_GROUP を定義する基本的な UCF 構文は次のとおりです。

**MIODELAY\_GROUP** "*master\_group\_name*" = *iodelay\_group1* *iodelay\_group2* ... ;

この場合、次を表します。

- ・ *master\_group\_name* は、定義されたマスタ グループを表しています。これには、*iodelay\_group1* と *iodelay\_group2* のエレメントがすべて含まれます。
- ・ *iodelay\_group1* と *iodelay\_group2* は、既に定義済みの IODELAY グループ (IODELAY\_GROUP) です。

## NODELAY

NODELAY は、高度なマップ制約です。IOB フリップフロップのデフォルトコンフィギュレーションには入力遅延が含まれているため、入力データパスに外部ホールド タイムは必要ありません。入力フリップフロップまたはラッチに NODELAY 属性を設定すると、この遅延を削除できます。この場合、セットアップ タイムは小さくなりますが、ホールド タイムが必要になります。

Spartan®-3、Spartan-3A、Spartan-3E では、デフォルトのコンフィギュレーションに入力遅延エレメントが含まれています。

NODELAY は、I/O シンボルや特殊ファンクションのアクセス シンボル TDI、TMS、TCK に割り当てることができます。

## アーキテクチャ サポート

Spartan-3、Spartan-3A および Spartan-3E デバイスがサポートされます。

すべての FPGA デバイスでこの制約を適用するに場合は、IOBDELAY=NONE を指定しておくことをお勧めします。詳細は、「IOBDELAY」を参照してください。

## 適用可能エレメント

入力レジスタ

NODELAY 制約は、UCF ファイル内で、パッド コンポーネントに接続されているネットにも設定できます。制約は、NGDBuild により NGD ファイル内のパッド インスタンスに挿入され、マップで処理されます。これには、次の UCF 構文を使用します。

```
NET " net_name" NODELAY;
```

## 適用ルール

ネットまたは信号には設定できませんが、ネットまたは信号がパッドに接続されている場合は例外です。この場合、NODELAY はパッド インスタンスに設定されているものと見なされます。

デザイン エLEMENT に設定すると、そのデザイン エLEMENT の階層にあるすべての適用可能エレメントに適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : NODELAY
- ・ 属性値 : TRUE、FALSE

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute nodelay: string;
```

VHDL 制約を次のように指定します。

```
attribute nodelay of {component_name | signal_name | label_name} : {component | signal | label} is  
"TRUE|FALSE";
```

VHDL 構文の詳細は、「VHDL」を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(* NODELAY = "TRUE|FALSE" *)
```

Verilog 構文の詳細は、「Verilog」を参照してください。



## UCF および NCF 構文

次の文は、IOB レジスタ inreg67 に入力遅延を含めないように指定します。

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

次の文は、net1 に接続されているパッドに入力遅延を含めないように指定します。

```
NET "net1" NODELAY;
```

## XCF 構文

```
BEGIN MODEL "entity_name "
```

```
NET "signal_name" nodelay=true;
```

```
INST "instance_name" nodelay=true;
```

```
END;
```

## NOREDUCE

NOREDUCE は、フィッタ制約および合成制約です。この制約を指定すると、通常はロジック ハザードやレース コンディションを避けるためにデザインに含まれている冗長な論理記述が最小化されません。また、組み合わせフィードバックループの出力ノードを識別して、正確なマップ処理をします。デザイン内に組み合わせフィードバック ラッチを作成するときは、必ず NOREDUCE をラッチの出力ネットに適用して、レース コンディションの回避に必要となる冗長な論理記述を含めます。

## アーキテクチャ サポート

この制約は、CPLD デバイスにのみ適用できます。

## 適用可能エレメント

すべてのネット

## 適用ルール

この制約はネットまたは信号の制約なので、デザイン エレメントには適用できません。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ ネットに設定します。
- ・ 属性名 : NOREDUCE
- ・ 属性値 : TRUE、FALSE

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute NOREDUCE: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します。

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

### UCF および NCF 構文

次の文は、ネット \$SIG\_12 以降にブール ロジックの減少やロジックのコラプスがないように指定します。

```
NET "$SIG_12" NOREDUCE;
```

### XCF 構文

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name" noreduce={true|false};
```

```
END;
```

## OFFSET IN

OFFSET IN 制約は、FPGA への入力インターフェイスのタイミング要件を指定するために使用します。この制約で指定できるのは、FPGA の外部パッドにおけるクロックとデータのタイミング関係です。OFFSET IN 制約を指定すると、制約の付いた同期エレメントすべてのセットアップ タイムとホールド タイムの要件が確認されます。次の図は、OFFSET IN 制約でカバーされるパスを示しています。

OFFSET IN 制約は、クロック ネット名を使用して指定します。OFFSET IN 制約の付いたクロック ネットは、外部クロックパッドになります。この制約で指定されるのは、FPGA の外部パッドのクロックとデータのタイミング関係であるため、内部クロック ネットを使用して制約を指定することはできません。ただし、同期エレメントがキャプチャされてセットアップ タイムとホールド タイムの要件が解析される際に、DCM、PLL、MMCM、IDELAY などのコンポーネントがあると、クロックパスの位相または遅延が自動的に考慮されます。また、この制約はクロック ネットワークを介して伝搬され、元の外部クロックから派生したすべてのクロックに自動的に適用されます。

OFFSET IN 制約は、デフォルトではグローバルに適用されます。この場合、特定のクロック ネットからのクロックが使用され、外部データをキャプチャする同期エレメントすべてにこの制約が適用されます。この制約が適用される同期エレメントは、入力データパッドのサブセットのタイム グループや、同期エレメントをキャプチャするサブセットのタイム グループ、またはその両方のグループを指定することで制限できます。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイスに適用できます。

## 適用可能エレメント

- ・ グローバル
- ・ 特定のネット
- ・ パッド タイム グループ

## OFFSET IN の構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

ここでは、UCF の構文例も提供していますが、OFFSET IN 制約の指定には Constraints Editor の使用をお勧めします。

### グローバルに指定する方法

OFFSET IN 制約は、デフォルトでグローバルに設定されます。この場合、入力データをキャプチャし、指定したクロック信号でトリガされる同期エレメントすべてに適用されます。

#### UCF 構文 :

```
OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]] {BEFORE|AFTER} "clk_name" [{RISING | FALLING}];
```

#### PCF 構文 :

```
OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]] {BEFORE|AFTER} COMP "clk_job_name" [{RISING | FALLING}];
```

#### 説明 :

- ・ "offset\_time" [units] : キャプチャクロックエッジとデータの開始時点のタイミングの違いです。この時間の単位は、設定してもしなくてもかまいません。単位を指定しない場合、デフォルトは ns (ナノ秒) になります。指定できる値は、ps、ns、micro、ms のいずれかです。
- ・ [VALID <datavalid\_time> [UNITS]] : データがキャプチャされるまでの時間です。この値は、入力インターフェイスの正確なホールドタイム違反を確認するために必要です。この時間の単位は、設定してもしなくてもかまいません。単位を指定しない場合、デフォルトは ns (ナノ秒) になります。指定できる値は、ps、ns、micro、ms のいずれかです。
- ・ BEFORE|AFTER : データの開始時点とクロックエッジのタイミング関係を定義します。クロックとデータ関係を定義するには、BEFORE を使用するのが最適な方法です。BEFORE は、クロックエッジに対して、データが有効になる時間を示します。BEFORE に正の値を指定すると、データはキャプチャクロックエッジの前に、負の値を指定すると、後に開始されます。
- ・ "clk\_name" : 入力クロックパッドネットの階層名すべてを定義します。
- ・ [{RISING | FALLING}] : クロックエッジを定義するオプションのキーワードで、データがクロックの立ち上がりエッジまたは立ち下がりエッジのどちらでキャプチャされるかが定義できます。また、これらのキーワードを使用すると、DDR (デュアルデータレート) インターフェイスの立ち上がりエッジレジスタと立ち下がりエッジレジスタが自動的に別のグループに分けられて、解析されるようになります。

### 入力グループを使用する方法

同じクロックでキャプチャされる入力グループが同じタイミング要件を持つ場合、入力同士がグループになり、1 つのタイミング制約が作成されます。入力は、パッドグループを使用して入力信号名別、またはレジスタグループを使用して同期エレメント別にグループ化できます。別々の信号を 1 つのタイムグループにすると、インプリメンテーションツールのメモリやランタイムが削減され、タイミングレポートにバスベースのスキューやクロックのセンタリング情報などが含まれるようになります。

#### UCF 構文 :

```
[TIMEGRP "pad_groupname"] OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]] {BEFORE|AFTER} "clk_name" [TIMEGRP "reg_groupname"] [{RISING | FALLING}];
```

#### PCF 構文 :

```
[TIMEGRP "inputpad_grpname"] OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]] {BEFORE|AFTER} COMP "clk_job_name" [TIMEGRP "reg_groupname"] [{RISING | FALLING}];
```

## 説明 :

- ・ [TIMEGRP “*pad\_groupname*”]: オプションの入力パッドのタイム グループです。このタイム グループを使用すると、OFFSET IN 制約の適用範囲をタイム グループに含まれる入力パッド ネットから接続された同期エレメントのみに制限することができます。
- ・ [TIMEGRP “*reg\_groupname*”]: オプションの同期エレメントのタイム グループです。このタイム グループを使用すると、OFFSET IN 制約の適用範囲を、指定クロック付き入力データをキャプチャする同期エレメントのみに制限することができます。

## ネットにのみ適用する方法

OFFSET IN 制約は、回路図のデータ ネットや UCF ファイルの入力パッド ネットや PCF ファイルの入力コンポーネントにも使用できます。

## ネットに設定した場合の回路図構文

```
OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]] {BEFORE|AFTER} "clk_name" [TIMEGRP "reg_groupname"] [{RISING | FALLING}];
```

## UCF 構文 :

```
NET "pad_net_name" OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]] {BEFORE|AFTER} "clk_name" [TIMEGRP "reg_groupname"] [{RISING | FALLING}];
```

## PCF 構文 :

```
COMP "pad_net_name" OFFSET = IN "offset_time" [units] [VALID <datavalid_time> [UNITS]] {BEFORE|AFTER} "clk_job_name" [TIMEGRP "reg_groupname"] [{RISING | FALLING}];
```

## 説明 :

- ・ “*pad\_net\_name*” : パッドに接続された入力データ ネットの名前です。
- ・ その他の変数やキーワードの定義は、前述の「グローバルに指定する方法」を参照してください。
- ・ PCF ファイルでは、ネット (NET) ではなく、I/O ブロック (COMP) を使用します。
- ・ PCF ファイルで IOB COMP 名が、UCF ファイルで NET 名が指定されていない場合、OFFSET IN 制約はグローバルに指定されているものと見なされます。

## ソース同期 DDR のエッジで揃えられた UCF 例

この例に含まれるインターフェイスでは、FPGA へのデータに対してエッジが揃えられたデバイスからクロックが送信されます。DDR インターフェイスでは、データは立ち上がりエッジと立ち下がりエッジの両方のクロック エッジでキャプチャされます。データをキャプチャする立ち上がりおよび立ち下がりクロック エッジのレジスタ別に OFFSET IN 制約を定義してください。OFFSET IN 制約に RISING および FALLING キーワードを使用すると、このタスクが簡単になります。

## 波形例 :

この例では、DDR インターフェイスが 5ns および 50/50 デューティ サイクルのクロック周期で記述されています。立ち上がり/立ち下がりデータは、2ns 間有効で、クロック波形の High と Low の真ん中にセンタリングされます。この結果、データの有効範囲の前後に 250 ps の差が出ます。

## 立ち上がりエッジ制約 :

立ち上がりエッジを指定した OFFSET IN 制約では、データをキャプチャする立ち上がりクロック エッジよりも前に、データが有効になる時間を定義します。この例では、データは立ち上がりエッジの後 250ps 間有効になります。データはクロック エッジの後に有効になるため、OFFSET IN BEFORE の値は -250ps と負の値になります。データの転送が開始されると、2ns 間有効のままになります。このため、VALID の値は 2ns になります。この制約に使用されている RISING キーワードは、この制約が立ち上がりエッジの同期エレメントにのみ適用され、OFFSET IN BEFORE 値が立ち上がりクロック エッジに指定されていることを示しています。

#### 立ち下がりエッジ制約 :

立ち下がりエッジを指定した OFFSET IN 制約では、データをキャプチャする立ち下がりクロック エッジよりも前に、データが有効になる時間を定義します。この例では、データは立ち下がりエッジの後 250ps 間有効になります。データはクロック エッジの後に有効になるため、OFFSET IN BEFORE の値は -250ps と負の値になります。データの転送が開始されると、2ns 間有効のままになります。このため、VALID の値は 2ns になります。この制約に使用されている FALLING キーワードは、この制約が立ち下がりエッジの同期エレメントにのみ適用され、OFFSET IN BEFORE 値が立ち下がりクロック エッジに指定されていることを示しています。

#### UCF 構文 :

次は、この例のクロックの PERIOD および OFFSET IN 制約の UCF 構文を示しています。

```
NET "clock" TNM<_NET = clock;  
TIMESPEC TS_CLK = PERIOD CLK 5.0 ns HIGH 50%;  
  
OFFSET = IN -250 ps VALID 2 ns BEFORE clock RISING;  
OFFSET = IN -250 ps VALID 2 ns BEFORE clock FALLING
```

#### ソース同期 DDR の真ん中で揃えられた UCF 例

この例に含まれるインターフェイスでは、データの中央に揃えられたデバイスからクロックが送信されます。DDR インターフェイスでは、データは立ち上がりエッジと立ち下がりエッジの両方のクロック エッジでキャプチャされます。データをキャプチャする立ち上がりおよび立ち下がりクロック エッジのレジスタ別に OFFSET IN 制約を定義してください。OFFSET IN 制約に RISING および FALLING キーワードを使用すると、このタスクが簡単になります。

#### 波形例 :

この例では、DDR インターフェイスが 5ns および 50/50 デューティ サイクルのクロック周期で記述されています。立ち上がり/立ち下がりデータは、2ns 間有効で、クロック波形の High と Low の真ん中にセンタリングされます。この結果、データの有効範囲の前後に 250 ps の差が出ます。

#### 立ち上がりエッジ制約 :

立ち上がりエッジを指定した OFFSET IN 制約では、データをキャプチャする立ち上がりクロック エッジよりも前に、データが有効になる時間を定義します。この例では、データは立ち上がりエッジの前に 1ns 間有効になります。データはクロック エッジの前に有効になるため、OFFSET IN BEFORE の値は 1ns と正の値になります。データの転送が開始されると、2ns 間有効のままになります。このため、VALID の値は 2ns になります。この制約に使用されている RISING キーワードは、この制約が立ち上がりエッジの同期エレメントにのみ適用され、OFFSET IN BEFORE 値が立ち上がりクロック エッジに指定されていることを示しています。

#### 立ち下がりエッジ制約 :

立ち下がりエッジを指定した OFFSET IN 制約では、データをキャプチャする立ち下がりクロック エッジよりも前に、データが有効になる時間を定義します。この例では、データは立ち下がりエッジの前に 1ns 間有効になります。データはクロック エッジの前に有効になるため、OFFSET IN BEFORE の値は 1ns と正の値になります。データの転送が開始されると、2ns 間有効のままになります。このため、VALID の値は 2ns になります。この制約に使用されている FALLING キーワードは、この制約が立ち下がりエッジの同期エレメントにのみ適用され、OFFSET IN BEFORE 値が立ち下がりクロック エッジに指定されていることを示しています。

#### UCF 構文 :

次は、この例のクロックの PERIOD および OFFSET IN 制約の UCF 構文を示しています。

```
NET "clock" TNM<_NET = clock;  
TIMESPEC TS_CLK = PERIOD CLK 5.0 ns HIGH 50%;  
  
OFFSET = IN 1 ns VALID 2 ns BEFORE clock RISING;  
OFFSET = IN 1 ns VALID 2 ns BEFORE clock FALLING;
```

### システム同期 SDR の UCF 例

この例に含まれるインターフェイスでは、クロックがクロック エッジ 1 つでデバイスから送信され、次のクロック エッジで FPGA にキャプチャされます。データはクロック サイクルごとに送信されます。必要な OFFSET IN 制約は 1 つのみです。

#### 波形例 :

この例では、SDR インターフェイスが 5ns および 50/50 デューティ サイクルのクロック周期で記述されています。データは 4ns 間有効で、送信クロック エッジの 500ps 後に開始されます。

#### 入力制約 :

OFFSET IN 制約では、データをキャプチャする立ち上がりクロック エッジよりも前に、データが有効になる時間を定義します。この例では、データは送信クロック エッジの後 500ps 有効になるか、データをキャプチャするクロック エッジの前に 4.5ns 間有効になります。データはクロック エッジの前に有効になるため、OFFSET IN BEFORE の値は 4.5ns と正の値になります。データの転送が開始されると、4ns 間有効のままになります。このため、VALID の値は 4ns になります。

#### UCF 構文 :

次は、この例のクロックの PERIOD および OFFSET IN 制約の UCF 構文を示しています。

```
NET "clock" TNM<_NET = clock;  
TIMESPEC TS_CLK = PERIOD CLK 5.0 ns HIGH 50%;  
OFFSET = IN 4.5 ns VALID 4 ns BEFORE clock;
```

### 回路図

- ・ 特定のネットに設定します。
- ・ 属性名 : OFFSET
- ・ 属性値 : IN|OUT *offset\_time* BEFORE|AFTER *clk\_pad\_netname*

### XCF 構文

UCF と同じ構文を使用しますが、XCF 構文でサポートされるのは OFFSET IN BEFORE を使用した方法のみです。

### PlanAhead からの設定

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

### Constraints Editor からの設定

ISE® の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Timing Constraints] の下の [Inputs] をダブルクリックし、ダイアログ ボックスにアクセスします。



## OFFSET OUT

OFFSET OUT 制約は、FPGA からの出力インターフェイスのタイミング要件を指定するために使用します。この制約で指定できるのは、FPGA の入力ピンのクロック エッジから FPGA の出力ピンでデータが有効になるまでの時間です。

OFFSET OUT 制約は、クロック ネット名を使用して指定します。OFFSET OUT 制約の付いたクロック ネットは、外部クロック パッドになります。この制約で指定されるのは、FPGA の入力ピンのクロック エッジから FPGA の出力ピンのデータまでなので、内部クロック ネットを使用して制約を指定することはできません。ただし、出力のタイミング要件が解析される際に、DCM、PLL、MMCM、IDELAY などのコンポーネントがあると、クロック パスの位相または遅延が自動的に考慮されます。また、この制約はクロック ネットワークを介して伝搬され、元の外部クロックから派生したすべてのクロックに自動的に適用されます。

OFFSET OUT 制約は、デフォルトではグローバルに適用されます。この場合、特定のクロック ネットからのクロックが使用され、外部データを送信する同期エレメントすべてにこの制約が適用されます。この制約が適用される同期エレメントは、出力データ パッドのサブセットのタイム グループや、同期エレメントを送信するサブセットのタイム グループ、またはその両方のグループを指定することで制限できます。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイ스에適用できます。

## 適用可能エレメント

- ・ グローバル
- ・ ネット
- ・ タイム グループ

## OFFSET OUT の構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

ここでは、UCF の構文例も提供していますが、OFFSET OUT 制約の指定には Constraints Editor の使用をお勧めします。

### グローバルに指定する方法

OFFSET OUT 制約は、デフォルトでグローバルに設定されます。この場合、出力データを送信し、指定したクロック信号でトリガされる同期エレメントすべてに適用されます。

#### UCF 構文 :

```
OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} "clk_name" [REFERENCE_PIN "ref_pin"] [{RISING | FALLING}];
```

#### PCF 構文 :

```
OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} COMP "clk_job_name" [REFERENCE_PIN "ref_pin"] [{RISING | FALLING}];
```

**説明 :**

- ・ `["offset_time" [units]]`: FPGA の入力ピンのクロック エッジから FPGA の出力ピンでデータが有効になるまでの時間を定義するオプションのパラメータです。offset\_time の値を指定すると、これらのパスにタイミング制約が適用され、制約に対するエラーがレポートされます。offset\_time を指定しない場合、タイミング制約は生成されませんが、そのインターフェイスの出力タイミングとバス スキューはレポートされます。このレポートのみを表示するオプションは、出力バスのスキューの方がクロックから出力への時間より重視されるソース同期インターフェイスで使用すると便利です。
- ・ BEFORE|AFTER : クロック エッジからデータの開始時点までのタイミング関係を定義します。クロックとデータの要件を定義するには、AFTER を使用するのが最適な方法です。AFTER は、FPGA のピンのクロック エッジ後にデータが有効になるまでの時間を示します。
- ・ `"clk_name"`: 入力クロック パッド ネットの階層名すべてを定義します。
- ・ [REFERENCE\_PIN "ref\_pin"] : クロックが再生成され、データと共に送信されるようなソース同期出力インターフェイスでよく使用されるオプションのキーワードです。REFERENCE\_PIN キーワードを使用すると、ref\_pin 信号に接続された出力信号のバス スキューを解析できるようになります。REFERENCE\_PIN キーワードを指定しない場合、バス スキューのレポートには出力遅延への最小クロック付きの信号が記述されます。
- ・ [{RISING | FALLING}] : データを送信する同期エレメントの送信クロック エッジを定義するためのオプションのキーワードです。また、これらのキーワードを使用すると、DDR (デュアル データレート) インターフェイスの立ち上がりエッジレジスタと立ち下がりエッジレジスタが自動的に別のグループに分けられて、解析されるようになります。

**出力グループを使用する方法**

同じクロックで送信される出力グループが同じタイミング要件を持つ場合、出力同士がグループになり、1 つのタイミング制約が作成されます。出力は、パッドグループを使用して出力信号名別、またはレジスタグループを使用して同期エレメント別にグループ化できます。別々の信号を 1 つのタイムグループにすると、インプリメンテーション ツールのメモリやランタイムが削減され、タイミング レポートにバス ベースのスキューやクロックのセンタリング情報などが含まれるようになります。

**UCF 構文 :**

```
[TIMEGRP "pad_groupname"] OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} "clk_name" [TIMEGRP "reg_groupname"] [REFERENCE_PIN "ref_pin"] [{RISING | FALLING}];
```

**PCF 構文 :**

```
[TIMEGRP "pad_groupname"] OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} COMP "clk_job_name" [TIMEGRP "reg_groupname"] [REFERENCE_PIN "ref_pin"] [{RISING | FALLING}];
```

**説明 :**

- ・ グループ別の方法は、次に示す一般的な方法と同じです。その他の変数やキーワードの定義は、前述の「グローバルに指定する方法」を参照してください。
- ・ [TIMEGRP "pad\_groupname"] : オプションの出力パッドのタイムグループです。このタイムグループを使用すると、OFFSET OUT 制約の適用範囲をタイムグループに含まれる出力パッド ネットから接続された同期エレメントのみに制限することができます。
- ・ [TIMEGRP "reg\_groupname"] : オプションの同期エレメントのタイムグループです。このタイムグループを使用すると、OFFSET OUT 制約の適用範囲を、指定クロック付き出力データを送信する同期エレメントのみに制限することができます。

**ネットにのみ適用する方法**

OFFSET OUT 制約は、回路図のデータ ネットや UCF ファイルの出力パッド ネットや PCF ファイルの出力コンポーネントにも使用できます。



### ネットに設定した場合の回路図構文

```
OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} "clk_name" [TIMEGRP "reg_groupname"]
[REFERENCE_PIN "ref_pin"] [{RISING | FALLING}];
```

#### UCF 構文 :

```
NET "pad_net_name" OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} "clk_name" [TIMEGRP
"reg_groupname"] [REFERENCE_PIN "ref_pin"] [{RISING | FALLING}];
```

#### PCF 構文 :

```
COMP "pad_net_name" OFFSET = OUT "offset_time" [units] {BEFORE|AFTER} "clk_name" [TIMEGRP
"reg_groupname"] [REFERENCE_PIN "ref_pin"] [{RISING | FALLING}];
```

#### 説明 :

- ・ グループ別の方法は、次に示す一般的な方法と同じです。その他の変数やキーワードの定義は、前述の「グローバルに指定する方法」を参照してください。
- ・ “pad\_net\_name” : パッドに接続された出力データ ネットの名前です。
- ・ PCF ファイルでは、ネット (NET) ではなく、I/O ブロック (COMP) を使用します。
- ・ PCF ファイルで IOB COMP 名が、UCF ファイルで NET 名が指定されていない場合、OFFSET OUT 制約はグローバルに指定されているものと見なされます。

### ソース同期 DDR の UCF 例

この例に含まれるインターフェイスでは、クロックが FPGA 内で再生成され、データと共に送信され、デバイスがキャプチャされます。DDR インターフェイスでは、データは立ち上がりエッジと立ち下がりエッジの両方のクロック エッジで送信されます。データを送信する立ち上がりおよび立ち下がりクロック エッジのレジスタ別に OFFSET OUT 制約を定義してください。OFFSET OUT 制約に RISING および FALLING キーワードを使用すると、このタスクが簡単になります。また、再生成されたクロックに対するバス スキューを解析するために、REFERENCE\_PIN キーワードが使用されます。

#### インターフェイス情報 :

この例では、clock というクロック信号が FPGA に入力され、データ出力の同期エレメントをトリガするために使用されます。また、TxClock という再生成されたクロックが作成されて、データと共に送信されます。これはソース同期インターフェイスなので、クロックから出力までの絶対時間は必要ないので、OFFSET OUT AFTER 値を指定しないで、レポートのみの制約を生成します。

#### UCF 構文 :

次は、この例のクロックの PERIOD および OFFSET OUT 制約の UCF 構文を示しています。

```
NET "clock" TNM_NET = clock;
TIMESPEC TS_CLK = PERIOD CLK 5.0 ns HIGH 50%;

OFFSET = OUT AFTER clock REFERENCE_PIN "TxClock" RISING;
OFFSET = OUT AFTER clock REFERENCE_PIN "TxClock" FALLING;
```

### システム同期 SDR の UCF 例

この例に含まれるインターフェイスでは、入力クロックが受信デバイスへデータを送信するために使用されます。SDR インターフェイスでは、データはクロック サイクルごとに送信されます。この場合、インターフェイスに制約を付けるため、OFFSET OUT が 1 つ必要になります。

#### インターフェイス情報 :

この例では、clock というクロック信号が FPGA に入力され、データ出力の同期エレメントをトリガするために使用されます。これはシステム同期インターフェイスなので、クロックから出力までの絶対時間は必要ありません。この場合、再生成されたクロックは存在しないので、REFERENCE\_PIN キーワードを指定しないで、デフォルトのスキュー レポートを出力させます。

## UCF 構文

次は、この例のクロックの PERIOD および OFFSET OUT 制約の UCF 構文を示しています。

```
NET "clock" TNM_NET = clock;
TIMESPEC TS_CLK = PERIOD CLK 5.0 ns HIGH 50%;

OFFSET = OUT 5 ns AFTER "clock";
```

## 回路図

- ・ 特定のネットに設定します。
- ・ 属性名 : OFFSET
- ・ 属性値 : OUT *offset\_time* BEFORE|AFTER *clk\_pad\_netname*

## XCF 構文

UCF と同じ構文を使用しますが、XCF 構文でサポートされるのは OFFSET OUT AFTER を使用した方法のみです。

## Constraints Editor からの設定

ISE® の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Timing Constraints] の下の [Outputs] をダブルクリックし、ダイアログ ボックスにアクセスします。

## PlanAhead からの設定

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## OPEN\_DRAIN

CoolRunner™-II デバイスの出力は、オープンドレイン出力信号としてプライマリ マクロセル出力を駆動するよう設定できます。OPEN\_DRAIN 制約は、トライステート以外の出力（常にアクティブ）に適用されると、オープンドレインに変換され、出力信号のステートはデバイスのピンでハイ インピーダンスになります。

このハイ インピーダンス状態は、論理シミュレーションでは発生しませんが、フィット後のタイミング シミュレーションで発生します。

OPEN\_DRAIN 制約を使用する代わりに、元の出力パッド信号を定数 0 に対して無効となるトライステートとして使用する方法もあります。CPLD フィッタは、定数 0 ですべてのトライステート出力を自動的に最適化し、デバイスのオープンドレイン機能を利用します。

## アーキテクチャ サポート

この制約は、CoolRunner-II デバイスにのみ適用できます。

## 適用可能エレメント

- ・ 出力パッド
- ・ パッド ネット

## 適用ルール

この制約は、ネットまたは信号の制約なので、マクロ、エンティティ、またはモジュールには設定できません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 出力パッド ネットに設定します。
- ・ 属性名 : OPEN\_DRAIN
- ・ 属性値 : TRUE、FALSE

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute OPEN_DRAIN: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、[「Verilog」](#)を参照してください。

### UCF および NCF 構文

```
NET "mysignal" OPEN_DRAIN;
```

### XCF 構文

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name" OPEN_DRAIN=true;
```

```
END;
```

## OPT\_EFFORT

OPT\_EFFORT は、基本的な配置配線制約で、オブティマイザで使用するエフォートレベルを定義します。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

すべてのマクロまたは階層レベル

## 適用ルール

OPT\_EFFORT は、マクロ、エンティティ、モジュールの制約なので、ネットまたは信号には適用できません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ マクロに設定します。
- ・ 属性名 : OPT\_EFFORT
- ・ 属性値 : Low (デフォルト)、Lowest、Normal、High、Highest

### UCF および NCF 構文

次の文は、インスタンス \$1I678/adder で定義されるモジュール内に含まれるすべてのロジックに、高度の最適化エフォートを設定します。

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

### Project Navigator からの設定

Project Navigator の [Process Properties] ダイアログ ボックスを表示し、[Place & Route Properties] タブの [Place & Route effort Level (Overall)] でグローバルに指定できます。デフォルトは [Standard] です。

[Process Properties] ダイアログ ボックスを表示するには、[Sources] タブでデザインを選択し、[Processes] タブで [Implement Design] を右クリックして [Process Properties] をクリックします。

## OPTIMIZE

OPTIMIZE は、基本的なマップ制約で、指定した階層ツリーに最適化を実行するかを定義します。OPTIMIZE は、I/O バッファなどの組み合わせロジックが含まれていないシンボルには効果がありません。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

マクロ、エンティティ、モジュール、または階層レベル

## 適用ルール

設定されたマクロ、エンティティ、モジュールに適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ デザイン エlement に設定します。
- ・ 属性名 : OPTIMIZE
- ・ 属性値 : AREA、SPEED、BALANCE、OFF

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute optimize string;
```

VHDL 制約を次のように指定します。

```
attribute optimize of entity_name:entity is "{AREA|SPEED|BALANCE|OFF}"
```

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

次のように指定します。

```
(* OPTIMIZE = "{AREA|SPEED|BALANCE|OFF}" *)
```

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

次の文は、マクロ CTR\_MACRO のインスタンスに最適化が実行されないように指定します。

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

## Project Navigator からの設定

Project Navigator の [Process Properties] ダイアログ ボックスの [Map Properties] タブの [Optimization Strategy (Cover Mode)] でグローバルに指定できます。デフォルトは [Area] です。

[Process Properties] ダイアログ ボックスを表示するには、[Sources] タブでデザインを選択し、[Processes] タブで [Implement Design] を右クリックして [Process Properties] をクリックします。

## PERIOD

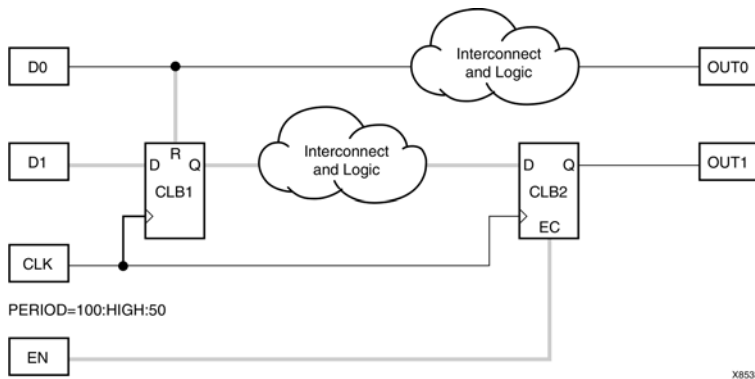
PERIOD は、基本的なタイミング制約および合成制約です。PERIOD 制約を指定すると、デスティネーション エlement のグループで定義されているクロックドメイン内で、すべての同期エlement間のタイミングが確認されます。クロックが別のクロックの関数として定義されている場合、グループには複数のクロックドメインを通過するパスが含まれます。

派生周期制約は、それらの参照制約と同じ単位で定義されます。

PERIOD 制約は、クロック ネットに設定されます。タイミング解析ツールは、レジスタのクロック ピンにおけるクロック ネットの反転や固定位相を自動的に考慮し、同期エlementのすべてのタイプを解析の対象とするため、クロック周期の定義は FROM-TO とは異なります。また、ホールド タイム違反があるかどうかともチェックされます。

クロック ネットに設定された PERIOD 制約は、クロック ネットに対応するセットアップまたはホールドのタイミング制約が設定されたピンが終端となるすべてのパスで、遅延を確認します。イネーブルがクロックと同期化している場合、このパスには CLB1.Q から CLB2.D までのパス、EN から CLB2.EC までのパスが含まれることがあります。

## PERIOD 制約のパス



設定要件によって必然的に決定される pad-to-register パスは、タイミング ツールでチェックされません。たとえば、D1 から CLB1 のピン D までのパスは PERIOD 制約には含まれません。また、clock-to-out パスも同様にチェックされません。

DLL、DCM、PLL、MMCM で、PERIOD 制約と共に TNM または TNM\_NET を使用する場合は、特別なルールが適用されます。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

フリップフロップのクロック ピンを駆動するためのネット

## 適用ルール

設定した信号に適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### TIMESPEC PERIOD の使用 (方法 1、推奨)

この方法では、単純なクロック周期だけでなく、より複雑な派生関係も定義できます。関連するクロック ネットに TNM 制約が設定されるとともに、TIMESPEC キーワードを使用して次のように設定されます。

## UCF 構文

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference" period {HIGH | LOW} [high_or_low_time] INPUT_JITTER value;
```

この場合、それぞれ次を表します。

- ・ *identifier* は、一意な名前を持つ参照識別名です。
- ・ *TNM\_reference* は、PERIOD 制約を適用するエレメントのグループを識別します。通常では、TNM\_NET という名前がクロック ネットに付けられていますが、それ以外にも同期エレメントのみを含む TNM グループまたはユーザー グループ (TIMEGRP) も使用できます。

次の規則が適用されます。

- ・ *period* は、必要なクロック周期です。
- ・ デフォルトで *period* の単位は ns ですが、ps、ns、micro、ms など指定できます。MHz、GHz、kHz などの単位を使用して周波数も指定できます。
- ・ 単位の前に空白を入れても入れなくてもかまいません。
- ・ 単位で大文字/小文字の区別をする必要はありません。
- ・ HIGH|LOW キーワードは、周期の最初のパルスの高低を表し、オプションの *high\_or\_low\_time* は最初のパルスの極性を表します。これらのキーワードは、最初のクロック エッジを定義し、OFFSET 制約で使用されます。ロジックレベルを指定しない場合は、デフォルトの High になります。
- ・ 実際の時間を指定する場合は、周期より小さい値にする必要があります。
- ・ *high\_or\_low\_time* を指定しない場合、デフォルトのデューティ サイクルは 50% です。
- ・ デフォルトで *high\_or\_low\_time* の単位は ns ですが、% も使用できます。単位には ps、ns、micro、ms など選択できます。
- ・ INPUT\_JITTER は、入力クロックの peak-to-peak ジッタです。デフォルトで、単位は ps に設定されています。

## 例

クロック ネット sys\_clk には、制約 *tnm=master\_clk* が設定され、次のように TIMESPEC に設定されます。

UCF 構文

```
TIMESPEC TS_master = PERIOD "master_clk" 50 HIGH 30 INPUT_JITTER 50;
```

PERIOD 制約はネット master\_clk に適用され、最初の High 状態の時間 30ns でクロック周期 50ns を、入力ジッタ 50ps を定義します。

```
TIMESPEC TS_clkInA = PERIOD "clkinA" 21 ns LOW 50% INPUT_JITTER 500 ps; TIMESPEC TS_clkInB = PERIOD "clkinB" 21 ns HIGH 50% INPUT_JITTER 500 ps;
```

## NET PERIOD の使用 (方法 2、推奨されない方法)

この方法では、レジスタのクロック ピンを駆動するパスで、制約をネットに直接設定します。

## 回路図

```
PERIOD = period {HIGH | LOW} [ high_or_low_time] INPUT_JITTER value;
```



## UCF 構文

```
NET "net_name" PERIOD = period {HIGH|LOW} [ high_or_low_time] INPUT_JITTER value;
```

次の規則が適用されます。

- ・ *period* は、必要なクロック周期です。デフォルトの単位は ns ですが、ps、micro、ms など指定できます。*period* には、MHz、GHz、kHz などの単位を使用して周波数を指定できます。
- ・ 単位の前に空白を入れても入れなくてもかまいません。
- ・ 単位で大文字/小文字の区別をする必要はありません。
- ・ HIGH|LOW キーワードは、周期の最初のパルスの高低を表し、オプションの *high\_or\_low\_time* は最初のパルスのデューティサイクルを表します。ロジックレベルを指定しない場合は、デフォルトの High になります。
- ・ 実際の時間を指定する場合は、周期より小さい値にする必要があります。
- ・ *high\_or\_low\_time* を指定しない場合、デフォルトのデューティサイクルは 50% になります。
- ・ デフォルトで *high\_or\_low\_time* の単位は ns ですが、% も使用できます。単位には ps、ns、micro、ms などを選択できます。

PERIOD 制約は TNM の場合とまったく同じ方法で順方向にトレースされ、到達したすべての同期エレメントに設定されます。ゲートを介したクロックをデザイン内で使用する場合など、複雑な形式のトレースが必要な場合は、特定ネットに PERIOD を設定するか、次の「推奨する方法」を使用する必要があります。

## 派生クロックの指定

「クロック周期を定義する推奨方法」では、識別名を使用して、別のクロック周期指定がそれを参照できるようにしています。ザイリンクスでは、派生の PERIOD 制約に同じ HIGH/LOW キーワードをマスタ PERIOD 制約として使用することをお勧めしています。マスタ PERIOD 制約に HIGH キーワードが付いているか、マスタ PERIOD 制約がデフォルトの場合、同じ HIGH キーワードを派生の PERIOD 制約に使用してください。派生クロックの場合に関係を定義するには、次の構文を使用します。

## UCF 構文

```
TIMESPEC "TSidentifier"=PERIOD "timegroup_name" "TSidentifier" [* or /] factor PHASE [+|-] phase_value [units];
```

この場合、それぞれ次を表します。

- ・ *identifier* は、一意な名前を持つ参照識別名です。
- ・ *factor* は、浮動小数点数です。

**メモ：** 定義される値が参照される値と同じで位相だけが異なる場合、(\*) または (/) の引数を省略できます。これは (\* 1) を使用するのと同じことになります。

- ・ *phase\_value* は、浮動小数点数です。
- ・ *units* は、ps、ms、micro、ns などの単位です。デフォルトは ns です。

次の規則が適用されます。

- ・ 実際の時間を指定する場合は、周期より小さい値にする必要があります。
- ・ *high\_or\_low\_time* を指定しない場合、デフォルトのデューティサイクルは 50% です。
- ・ デフォルトで *high\_or\_low\_time* の単位は ns ですが、% も使用できます。単位には ps、ns、micro、ms などを選択できます。



## 派生クロックを使用したプライマリ クロックの例

プライマリ クロックの周期

```
TIMESPEC "TS01" = PERIOD "clk0" 10.0 ns;
```

180° 順方向へ位相シフトしたクロックの周期

```
TIMESPEC "TS02" = PERIOD "clk180" TS01 PHASE + 5.0 ns;
```

90° 逆方向へ位相シフトしたクロックの周期

```
TIMESPEC "TS03" = PERIOD "clk90" TS01 PHASE - 2.5 ns;
```

180° 順方向 (TS01 に対して 90° ) に位相シフトしたり、倍増したクロックの周期

```
TIMESPEC "TS04" = PERIOD "clk180" TS01 / 2 PHASE + 2.5 ns;
```

## 回路図

- ・ ネットに設定します。構文は、次のようになります。
- ・ 属性名 : PERIOD
- ・ 属性値 : *period* [*units*] [{**HIGH**|**LOW**} [*high\_or\_low\_time* [*hi\_lo\_units*]]

## VHDL 構文

XST の場合、PERIOD は特定のクロック信号のみに適用されます。

VHDL 制約を次のように宣言します。

```
attribute period: string;
```

VHDL 制約を次のように指定します。

```
attribute period of signal_name: signal is "period[units]";
```

- ・ *period* は、必要なクロック周期です。
- ・ *units* は、オプションでクロック周期の単位を指定します。デフォルトの単位はナノ秒 (ns) ですが、ps、ns、micro など指定できます。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

XST の場合、PERIOD は特定のクロック信号のみに適用されます。

Verilog 制約を次のように指定します

```
(* PERIOD = "period[units]" *)
```

- ・ *period* は、必要なクロック周期です。
- ・ *units* は、オプションでクロック周期の単位を指定します。デフォルトの単位はナノ秒 (ns) ですが、ps、ns、micro など指定できます。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

## TIMESPEC PERIOD の使用 (方法 1、推奨)

UCF 構文

```
TIMESPEC "TS $_{identifier}$ "=PERIOD "TNM $_{reference\ period}$ " [units] [{HIGH | LOW} [high_or_low_time [hi_lo_units]]]
INPUT_JITTER value [units];
```

この場合、それぞれ次を表します。

- ・ *identifier* は、一意な名前を持つ参照識別名です。
- ・ *TNM<sub>reference</sub>* は、TNM 制約や TNM\_NET 制約を使用してクロック ネットまたはクロック パスのネットに適用するときの識別名です。

DLL、DCM、PLL、MMCM コンポーネントの CLKIN 入力に TNM\_NET 制約がトレースされる場合、新たに PERIOD を DLL/DCM/PLL/MMCM 出力に指定できます。この場合、その指定で使用する TNM\_NET グループも新たに生成されます。

新しい TNM\_NET グループには、対応する DLL/DCM/PLL/MMCM の出力ネットと同じ名前 (*outputnetname*) が付けられます。新しい PERIOD 指定は、TS\_*outputnetname*=PERIOD *outputnetname* value units になります。

新しい TNM\_NET グループは、DLL/DCM/PLL/MMCM の出力ネットからトレースされ、クロック信号で制御されるすべての同期エレメントを指定します。新しいグループと指定は、タイミング解析レポートに出力されます。

次の規則が適用されます。

- ・ *period* は、必要なクロック周期です。
- ・ *units* は、オプションでクロック周期の単位を指定します。デフォルトはナノ秒 (ns) ですが、ps、ms、micro、% なども指定できます。
- ・ HIGH または LOW は、最初のパルスを High にするか、Low にするかを指定します。
- ・ *high\_or\_low\_time* は、High または Low になっている時間を指定します (オプション)。High か Low かは、この前のキーワードによって指定します。実際の時間を指定する場合は、周期より小さい値にする必要があります。*high\_or\_low\_time* を指定しない場合、デフォルトのデューティサイクルは 50% になります。
- ・ *hi\_lo\_units* は、デューティサイクルの単位を指定します (オプション)。デフォルトはナノ秒 (ns) です。*high\_or\_low\_time* が実際の計測値である場合、High または Low の時間を示す値の後に ps、micro、ms、% を付けて単位を指定できます。

次の文は、クロック周期 40ns をネット \$SIG\_24 に設定します。最初のパルスは High で、その継続時間は 25ns です。

```
NET "CLOCK" PERIOD=40 HIGH 25;
```

## NET PERIOD の使用 (方法 2、推奨されない方法)

```
NET "net_name" PERIOD=period [units] [{HIGH|LOW} [high_or_low_time [hi_lo_units]]];
```

この場合、それぞれ次を表します。

- ・ *period* は、必要なクロック周期です。
- ・ *units* は、オプションでクロック周期の単位を指定します。デフォルトの単位はナノ秒 (ns) ですが、ps、ns、micro など指定できます。
- ・ オプションで HIGH または LOW を使用すると、最初のパルスを High または Low に指定できます。
- ・ *hi\_lo\_units* は、ns、ps、または micro です。デフォルトは ns です。

次の規則が適用されます。

- ・ *high\_or\_low\_time* は、High または Low になっている時間を指定します (オプション)。High か Low かは、この前のキーワードによって指定します。
- ・ 実際の時間を指定する場合は、周期より小さい値にする必要があります。
- ・ *high\_or\_low\_time* を指定しない場合、デフォルトのデューティ サイクルは 50% になります。
- ・ *hi\_lo\_units* は、デューティ サイクルの単位を指定します (オプション)。
- ・ デフォルトはナノ秒 (ns) です。*high\_or\_low\_time* が実際の計測値である場合、High または Low の時間を示す値の後に ps、micro、ms、% を付けて単位を指定できます。

## Constraints Editor の構文

ISE® の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リストボックスの [Timing Constraints] の下の [Clock Domains] をダブルクリックし、ダイアログ ボックスにアクセスします。

## XCF 構文

UCF と同じ構文を使用します。

単純な方法および推奨する方法の両方がサポートされていますが、HIGH/LOW の値はタイミング概算および最適化では使用されず、WRITE\_TIMING\_CONSTRAINTS=yes の場合に最終のネットリストに含められるのみです。

## PCF 構文

```
"TSidentifier"=PERIOD perioditem periodvalue INPUT_JITTER value;
```

*perioditem* は、次のいずれかです。

- ・ NET *name*
- ・ TIMEGRP *name*

*periodvalue* は、次のいずれかです。

- ・ TSidentifier PHASE [+ | -] *time*
- ・ TSidentifier PHASE *time*
- ・ TSidentifier PHASE [+ | -] *time* [LOW | HIGH] *time*
- ・ TSidentifier PHASE *time* [LOW | HIGH] *time*
- ・ TSidentifier PHASE [+ | -] *time* [LOW | HIGH] *percent*
- ・ TSidentifier PHASE *time* [LOW | HIGH] *percent*

## PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## FPGA Editor の構文

制約を設定するには、FPGA Editor のメイン ウィンドウで [Edit] → [Properties of Selected Items] をクリックします。PERIOD 制約を設定するには、ネットを選択した状態で [Edit] → [Properties of Selected Items] をクリックします。制約は、[Physical Constraints] タブから設定できます。

## CLKDLL、DCM、PLL、MMCM での PERIOD 仕様

DLL、DCM、PLL、および MMCM の入力ピンへトレースされる場合、TNM または TNM\_NET プロパティは次のように処理されます。

確認および変換は、NGDBuild の実行中に論理的なタイムスペック処理コードで行われます。この時、タイミング要件の確認状態メッセージは、論理的なタイムスペック処理が実行中であることを表示します。変更データは、後ほどマップで使用される NGD ファイルに保存され、PCF ファイルを介して PAR および TRACE に渡されます。

ただし、NGD ファイルに保存されたデータは、ユーザーが適用したオリジナルのタイムスペック プロパティとは異なります。オリジナルのプロパティは、このプロセスでは変更されません。したがって、Constraints Editor では、新しく作成されたグループや仕様は識別されませんが、オリジナルのプロパティは識別され、変更されます。

### 変換の条件

DLL、DCM、PLL、または MMCM の CLKIN ピンに TNM\_NET プロパティがトレースされる場合、TNM グループとその使用が確認されます。次の条件に適合した場合にのみ、TNM は CLKDLL、DCM、PLL、MMCM に適用されます。

- ・ TNM グループが単一の PERIOD 制約で使用される。
- ・ TNM グループが FROM-TO 指定または OFFSET 指定で使用されない。
- ・ TNM グループがユーザー グループ定義では参照されない。

上記のいずれの条件にも適合しない場合、TNM は CLKDLL/DCM/PLL/MMCM には適用されず、警告メッセージが表示されます。これによって、ほかのエレメントに TNM がトレースされなくなるわけではありません。ただし、どこにもトレースせずに仕様に使用された場合はエラーになります。

### 新しい PERIOD 仕様の定義

CLKDLL、DCM、PLL、または MMCM の CLK0 出力のみが使用されている場合 (CLKIN\_DIVIDE\_BY\_2 および CLKOUT\_PHASE\_SHIFT=FIXED のいずれも使用されていない場合)、オリジナルの PERIOD 仕様がそのクロック出力に反映されます。上記以外の場合、CLKDLL、DCM、PLL、または MMCM で使用される各クロック出力ピンに対して、接続されたネットで新しい TNM グループが作成され、このグループに対して新たに PERIOD 仕様が作成されます。次に、PERIOD 仕様の定義方法を示します。ここでは、オリジナルの PERIOD 仕様を TS\_CLKIN とします。

## 新しい PERIOD 仕様

出力ピン	周期値	位相シフト	DUTY_CYCLE 制約
CLK0	$TS\_CLKIN * 1$	なし	DUTY_CYCLE_CORRECTION が FALSE の場合は TS_CLKIN からコピーされます。それ以外は 50% です。
CLK90	$TS\_CLKIN * 1$	$PHASE + (clk0\_period * 1/4)$	DUTY_CYCLE_CORRECTION が FALSE の場合は TS_CLKIN からコピーされます。それ以外は 50% です。
CLK180	$TS\_CLKIN * 1$	$PHASE + (clk0\_period * 1/2)$	DUTY_CYCLE_CORRECTION が FALSE の場合は TS_CLKIN からコピーされます。それ以外は 50% です。
CLK270	$TS\_CLKIN * 1$	$PHASE + (clk0\_period * 3/4)$	DUTY_CYCLE_CORRECTION が FALSE の場合は TS_CLKIN からコピーされます。それ以外は 50% です。
CLK2X	$TS\_CLKIN / 2$	なし	50%
CLK2X180	$TS\_CLKIN / 2$	$PHASE + (clk2X\_period * 1/2)$	50%
CLKDV	$TS\_CLKIN * clkdv\_divide$  $clkdv\_divide$ は、CLKDV_DIVIDE プロパティ値 (デフォルト 2.0) です。	なし	高周波数モードで除算される非整数は除きます (CLKDLLHF、または DLL_FREQUENCY_MODE が HIGH に設定された DCM の場合)  CLKDV_DIVIDE  1.5 33.33% HIGH  2.5 40.00% HIGH  3.5 42.86% HIGH  4.5 44.44% HIGH  5.5 45.45% HIGH  6.5 46.15% HIGH  7.5 46.67% HIGH
CLKFX		なし	
CLKFX180	$TS\_CLKIN / clkfx\_factor$  $clkfx\_factor$ は、CLKFX_MULTIPLY プロパティの値で (デフォルトは 4.0)、CLKFX_DIVIDE プロパティ (デフォルト 1.0) の値で除算されます。	$PHASE + (clkfx\_period * 1/2)$	50%

周期値では、オリジナルの指定 TS\_CLKIN が時間として表されていることを前提とします。TS\_CLKIN が周波数として表された場合、乗算または除算が元に戻されます。

DCM の属性 FIXED\_PHASE\_SHIFT または VARIABLE\_PHASE\_SHIFT を使用する場合、指定した位相の値が PHASE の値に含まれている必要があります。

## PIN

PIN 制約は、ネットの位置 (ロケーション) を定義するのに LOC 制約と一緒に使用されます。

PIN/LOC の UCF 構文は、次のとおりです。

```
PIN " module.pin " LOC=" location " ;
```

この構文は、モジュール デザインのフローのみで使用します。この制約は PCF ファイルで COMP/LOCATE 制約に変換されます。PCF ファイルでは、次の構文が使用されます。

```
COMP "name" LOCATE = SITE "location " ;
```

この制約は、モジュールでピンに対して作成された疑似コンポーネントがサイトの位置に配置されるように指定します。疑似ロジックは、ネットがあるモジュールのピンから別のモジュールのピンに接続される場合にのみ、作成されます。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

ネット

## 適用ルール

ありません。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF 構文

```
PIN " module.pin " LOC=location ;
```

```
PIN mod.pin TIG;
```

## POST\_CRC

コンフィギュレーション ロジック CRC エラー検出機能をオンまたはオフにします。コンフィギュレーション メモリの変更は、この機能により知ることができます。Spartan-3A の場合は、コンフィギュレーション CRC エラーの信号送信用に 多用途 INIT ピンが予約される際にもこの機能が影響します、また、この機能により、PlanAhead™、PAR、および BitGen によって使用されるバンク規則で、INIT ピンを駆動する IOB が使用されないように指定することもできます。INIT ピンは、コンフィギュレーション中は通常通りに作動し、POST\_CRC 解析がオンの場合は、コンフィギュレーション後、CRC ステータス ピンとして動作します。リアルタイムに計算された CRC が事前に計算された CRC と異なると、コンフィギュレーション メモリの変更が検出され、INIT ピンは Low に駆動されます。

次の表に POST\_CRC の値を示します。

値	説明
ENABLE	Post CRC の検出機能をオンにします。
DISABLE	Post CRC の検出機能をオフにします。(デフォルト)。

## アーキテクチャ サポート

この制約は、Virtex®-5、Virtex-6、Spartan®-3A および Spartan-6 デバイスに適用できます。

## 適用可能エレメント

この制約は特定のデザイン エレメントではなく、デバイス全体に適用されます。

## 適用ルール

この制約はデザイン/デバイス全体に適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

#### UCF 構文

```
CONFIG POST_CRC = {ENABLE|DISABLE|ONESHOT};
```

#### PCF 構文

```
CONFIG POST_CRC = {ENABLE|DISABLE|ONESHOT};
```

## POST\_CRC\_ACTION

Spartan®-3A、Spartan-6、Virtex®-6 デバイスは、POST\_CRC (コンフィギュレーション ロジック CRC エラー検出モード) をサポートします。POST\_CRC では、事前に計算された、コンフィギュレーション ビットストリームの CRC を、コンフィギュレーション メモリ セルの周期的なリードバックに基づいて内部ロジックで計算した CRC と比較します。POST\_CRC\_ACTION では、CRC の不一致が検出されたときに検出を続けるか、処理を停止するかを指定します。この制約は、POST\_CRC が ENABLE に設定されているときにのみ有効です。

次の表に POST\_CRC\_ACTION の値を示します。

値	説明
HALT	CRC の不一致が検出されると、ビットストリームのリードバック、比較 CRC の計算、および事前に計算された CRC との比較が停止されます (Spartan-6 のデフォルト)。
CONTINUE	CRC の不一致が検出されても、ビットストリームのリードバック、比較 CRC の計算、事前に計算された CRC との比較が続行されます (Virtex-6 のデフォルト)。
CORRECT_AND_CONTINUE	CRC の不一致が検出されると、それが修正され、ビットストリームのリードバック、CRC の計算、事前に計算された CRC との比較が続行されます。
CORRECT_AND_HALT	CRC の不一致が検出されると、それが修正され、ビットストリームのリードバック、CRC の計算、事前に計算された CRC との比較が停止されます。



## アーキテクチャ サポート

この制約は、Spartan-3A、Spartan-6、Virtex-6 デバイスに適用できます。

## 適用可能エレメント

この制約は特定のデザイン エレメントではなく、デバイス全体に適用されます。

## 適用ルール

この制約はデザイン/デバイス全体に適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

#### UCF 構文

```
CONFIG POST_CRC_ACTION = [HALT|CONTINUE];
```

#### PCF 構文

```
CONFIG POST_CRC_ACTION = [HALT|CONTINUE];
```

## POST\_CRC\_FREQ

Spartan®-3A、Spartan-6、Virtex®-6 デバイスは、POST\_CRC (コンフィギュレーション ロジック CRC エラー検出モード) をサポートします。POST\_CRC では、事前に計算された、コンフィギュレーション ビットストリームの CRC を、コンフィギュレーション メモリ セルの周期的なリードバックに基づいて内部ロジックで計算した CRC と比較します。POST\_CRC\_FREQ はコンフィギュレーション CRC チェックが Spartan-3A デバイス内で実行される頻度を制御します。この制約は、POST\_CRC が ENABLE に設定されているときにのみ有効です。

Spartan-3A の場合、頻度の範囲は 10 ビットで表され、1 ~ 100MHz の間です。指定できる値は 1、3、6、7、8、10、12、13、17、22、25、27、33、44、50、および 100MHz です。デフォルトは 1MHz です。

Spartan-6 の場合、頻度の範囲は 10 ビットで表され、1 ~ 100MHz の間です。指定できる値は 1、2、4、6、10、12、16、22、26、33、40、50、および 66MHz です。デフォルトは 1MHz です。

Virtex-6 の場合、頻度の範囲は 10 ビットで表され、1 ~ 50MHz の間です。指定できる値は 1、2、3、6、13、25、および 50MHz です。デフォルトは 1MHz です。

## アーキテクチャ サポート

この制約は、Spartan-3A、Spartan-6、Virtex-6 デバイスに適用できます。

## 適用可能エレメント

この制約は特定のデザイン エレメントではなく、デバイス全体に適用されます。

## 適用ルール

この制約はデザイン/デバイス全体に適用されます。



## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF 構文

```
CONFIG POST_CRC_FREQ = [1|3|6|7|8|10|12|13|17|22|25|27|33|44|50|100];
```

### PCF 構文

```
CONFIG POST_CRC_FREQ = [1|3|6|7|8|10|12|13|17|22|25|27|33|44|50|100];
```

## POST\_CRC\_SIGNAL

Virtex®-5 デバイスは、POST\_CRC (コンフィギュレーション ロジック CRC エラー検出モード) をサポートします。POST\_CRC では、事前に計算された、コンフィギュレーション ビットストリームの CRC を、コンフィギュレーション メモリ セルの周期的なリードバックに基づいて内部ロジックで計算した CRC と比較します。POST\_CRC\_SIGNAL では、INIT\_B ピンを SEU (Single Event Upset) エラー信号の出力としてイネーブルにするかどうか指定されます。エラーが発生した場合は FRAME\_ECC\_VIRTEX5 サイトより知ることができます。この制約は、POST\_CRC が ENABLE に設定されているときにのみ有効です。

次の表に POST\_CRC\_SIGNAL の値を示します。

値	説明
FRAME_ECC_ONLY	INIT_B ピンの使用をオフにします。FRAME_ECC サイトが CRC エラー信号の唯一のソースとなります。
INIT_AND_FRAME_ECC	INIT_B ピンを CRC エラー信号のソースとして使用します (デフォルト)。

## アーキテクチャ サポート

この制約は、Virtex-5 デバイスにのみ適用できます。

## 適用可能エレメント

この制約は特定のデザイン エレメントではなく、デバイス全体に適用されます。

## 適用ルール

この制約はデザイン/デバイス全体に適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF 構文

```
CONFIG POST_CRC_SIGNAL = [FRAME_ECC_ONLY|INIT_AND_FRAME_ECC];
```

### PCF 構文

```
CONFIG POST_CRC_SIGNAL = [FRAME_ECC_ONLY|INIT_AND_FRAME_ECC];
```

## PRIORITY

PRIORITY は、高度なタイミング制約キーワードです。同じパスを持つ 2 つのタイミング制約間で競合が発生する場合があります。PRIORITY の値は、小さい数字から優先順位が高くなります。この値は、配置配線が実行されるパスの順番には影響しませんが、優先順位が同じ 2 つの制約を同じパスに設定した場合、パスに適用される制約の選択に影響します。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイ스에適用できます。

## 適用可能エレメント

TIMESPEC

## 適用ルール

ありません。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF および NCF 構文

次の構文を使用して、タイミング制約の優先順位を定義します。

```
normal_timespec_syntax PRIORITY integer;
```

*normal\_timespec\_syntax* 有効なタイミング仕様を示します。

*integer* 優先順位を表します。数字が小さい方が優先順位が高くなります。

数字には、正の数、負の数、またはゼロを使用できます。この数字は、ほかの PRIORITY 値と比較する場合にのみ有効です。小さい数字から優先順位が高くなります。

```
TIMESPEC "TS01"=FROM "GROUPA" TO "GROUPB" 40 PRIORITY 4;
```

### PCF 構文

UCF と同じです。

## PROHIBIT

PROHIBIT は、基本的な配置制約です。PAR、FPGA Editor、CPLD fitter でサイトが使用されないよう指定します。

FPGA デバイスのロケーション指定の種類

エレメントの物理的な位置を定義するには、次のように指定します。

## FPGA デバイスのロケーション指定の種類

エレメント のタイプ	位置の指定	意味
IOB	P12	IOB の位置 (チップ キャリア)
	A12	IOB の位置 (ピン グリッド)
	T、B、L、R	Spartan®-3、Spartan-3A、Spartan-3E、Virtex®-4、Virtex-5 の場合、IOB に適用され、エッジの位置 (下、左、上、右) を示します。
	LB、RB、LT、RT、BR、TR、BL、TL	Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 の場合、IOB に適用され、ハーフ エッジの位置 (左下、右下など) を示します。
	Bank 0、Bank 1、Bank 2、Bank 3、Bank 4、Bank 5、Bank 6、Bank 7	Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 の場合、IOB に適用され、ハーフ エッジ (バンク) を示します。
スライス	SLICE_X2Y3	Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 デバイスのサイトの位置
ブロック RAM	RAMB16_X2Y56	Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 デバイスのブロック RAM の位置
乗算器	MULT18X18_X55Y82	Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 デバイスの乗算器の位置
グローバル クロック	BUFGMUX0P	Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 デバイスのグローバル クロック バッファの位置
デジタル クロック マネージャ (DCM)	DCM_X[A]Y[B]	Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 デバイスのデジタル クロック マネージャ
位相ロック ループ (PLL)	PLL_X[A]Y[B]	Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 の位相ロック ループ
混合モード クロック マネージャ (MMCM)	MMCM_X[A]Y[B]	Virtex-6 用の混合モード クロック マネージャ (MMCM)

次の例のように、ワイルドカード文字 (\*) を使用すると、単一の位置を範囲に置き換えることができます。

SLICE_X*Y5	Y 座標が 5 にある FPGA デバイスのすべてのスライス
------------	--------------------------------

次の文字は、サポートされていません。

- ・ 範囲を指定する拡張子 (LOC=SLICE\_X3Y5:SLICE\_X5Y7.G など)
- ・ Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 のグローバル バッファ、グローバル パッド、DLL の位置を表すワイルドカード文字

## CPLD デバイスのロケーション指定の種類

CPLD デバイスでは、位置の種類 *pin\_name* しかサポートされません。

この場合、それぞれ次を表します。

*pin\_name* では、ピン名に *Pnn*、または行/列のピン名に *rc* を使用します。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイスに適用できます。

## 適用可能エレメント

サイト

## 適用ルール

ネット、信号、エンティティ、モジュール、マクロには設定できません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF 構文

UCF ファイルでは、PROHIBIT の前に CONFIG キーワードを付ける必要があります。

単一の位置

```
CONFIG PROHIBIT=location;
```

複数の位置

```
CONFIG PROHIBIT=location1, location2, ... ,locationnn;
```

位置の範囲

```
CONFIG PROHIBIT=location1:location2;
```

この場合、それぞれ次を表します。

*location* は、デバイス タイプに有効な位置です。

詳細は、FPGA デバイスおよび CPLD デバイスの位置指定の制約の箇所を参照してください。位置指定の例は「[LOC](#)」を参照してください。CPLD デバイスでは、「位置の範囲」形式の PROHIBIT はサポートされていません。

次の文は、サイト P45 の使用を禁止します。

```
CONFIG PROHIBIT=P45;
```

次の文は、SLICE\_X6Y8 のサイトにあるスライスの使用を禁止します。

```
CONFIG PROHIBIT=SLICE_X6Y8;
```

### PCF 構文

単一または複数の位置の場合：

```
COMP "comp_name" PROHIBIT = [SOFT] "site_group"..."site_group";
```

```
COMPGRP "group_name" PROHIBIT = [SOFT] "site_group"..."site_group";
```

```
MACRO "name" PROHIBIT = [SOFT] "site_group"..."site_group";
```

位置の範囲の場合 :

```
COMP "comp_name" PROHIBIT = [SOFT] "site_group"... "site_group";
```

```
COMPGRP "group_name" PROHIBIT = [SOFT] "site_group"... "site_group";
```

```
MACRO "name" PROHIBIT = [SOFT] "site_group"... "site_group";
```

この場合、それぞれ次を表します。

- ・ *site\_group* は、次のいずれかです。
  - **SITE** "*site\_name*"
  - **SITEGRP** "*site\_group\_name*"
- ・ *site\_name* は、コンポーネント サイト (CLB または IOB の位置) です。

### PlanAhead の構文

PlanAhead™ を使用したエリア グループ制約の作成方法は、『PlanAhead ユーザー ガイド』の「Floorplanning the Design (デザインのフロアプラン)」を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

### PACE の構文

PACE を使用して指定します。詳細は、PACE ヘルプの [Prohibit Mode] コマンドに関するセクションを参照してください。

**メモ** : PACE は、CPLD でのみサポートされます。

### FPGA Editor の構文

FPGA Editor では PROHIBIT がサポートされます。詳細は、FPGA Editor ヘルプの「禁止制約」の項目を参照してください。制約は PCF ファイルに書き込まれます。

## PULLDOWN

PULLDOWN は、基本的なマップ制約です。この制約を設定すると、ロジック レベルが Low になるため、トライステート状態のネットが駆動されていない場合でもフロートしません。

KEEPER、PULLUP および PULLDOWN は、パッド ネットにのみ使用でき、INST には使用できません。

## アーキテクチャ サポート

PULLDOWN 制約は、すべての FPGA および CPLD の CoolRunner™-II にのみ適用されます。

## 適用可能エレメント

- ・ 入力
- ・ トライステート出力
- ・ 双方向のパッド ネット

## 適用ルール

PULLDOWN は、ネット制約なので、デザイン エLEMENT には適用できません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ パッド ネットに設定します。
- ・ 属性名 : PULLDOWN
- ・ 属性値 : TRUE、FALSE

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute PULLDOWN: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、[「Verilog」](#)を参照してください。

### UCF および NCF 構文

次の文は、I/O で PULLDOWN を使用するように指定します。

```
NET "pad_net_name" PULLDOWN;
```

次の文は、PULLDOWN をグローバルに設定しています。

```
DEFAULT PULLDOWN = TRUE;
```

### XCF 構文

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name" pulldown=true;
```

```
END;
```

### PlanAhead からの設定

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## PULLUP

PULLUP は、基本的なマップ制約です。この制約を使用すると、ロジックレベルが High になるので、トライステートで駆動されるネットが駆動していない状態のときでもフロートしません。

KEEPER、PULLUP および PULLDOWN は、パッド ネットにのみ使用でき、INST には使用できません。

CoolRunner™-II デザインでは、KEEPER と PULLUP を共に使用できません。

## アーキテクチャ サポート

PULLUP 制約は、すべての FPGA および CPLD デバイスの CoolRunner XPLA3 および CoolRunner-II に適用されます。

## 適用可能エレメント

- ・ 入力
- ・ トライステート出力
- ・ 双方向のパッド ネット

## 適用ルール

PULLUP は、ネット制約なので、デザイン エレメントには適用できません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ パッド ネットに設定します。
- ・ 属性名 : PULLUP
- ・ 属性値 : TRUE、FALSE

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute PULLUP: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

次の文は、I/O で PULLUP を使用するよう指定します。

```
NET "pad_net_name" PULLUP;
```

次の文は、PULLUP をグローバルに設定しています。

```
DEFAULT PULLUP = TRUE;
```

## XCF 構文

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name" pullup=true;
```

```
END;
```

## PlanAhead からの設定

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## PWR\_MODE

PWR\_MODE は、高度なフィッタ制約で、指定したエレメントをインプリメントするマクロセルのモードを Low または High (標準) に定義します。

指定したファンクションがファンアウト分割されている場合、PWR\_MODE は適用されません。

## アーキテクチャ サポート

この制約は、XC9500 デバイスにのみ適用できます。

## 適用可能エレメント

- ・ ネット
- ・ すべてのインスタンス

## 適用ルール

ネットに設定されている場合、ネットを駆動するすべての適用可能エレメントに適用されます。



デザイン エlementに設定すると、そのデザイン エlementの階層にあるすべての適用可能Elementに適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ ネットまたはインスタンスに設定します。
- ・ 属性名 : PWR\_MODE
- ・ 属性値 : LOW、STD

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute PWR_MODE: string;
```

VHDL 制約を次のように指定します。

```
attribute PWR_MODE of {signal_name|component_name|label_name}: {signal|component|label}  
is \"LOW|STD\";
```

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(* PWR_MODE = \"LOW|STD\" *)
```

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

次の文は、ネット \$SIG\_0 をインプリメントするマクロセルを Low モードに指定します。

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

## XCF 構文

```
BEGIN MODEL \"entity_name\"
```

```
NET \"signal_name\" PWR_MODE={LOW|STD};
```

```
INST \"instance_name\" PWR_MODE={LOW|STD};
```

```
END;
```

## REG

REG は基本的なフィタ制約で、CPLD のマクロセルにレジスタをインプリメントする方法を指定します。

## アーキテクチャ サポート

この制約は、CPLD デバイスにのみ適用できます。

## 適用可能エレメント

レジスタ

## 適用ルール

デザイン エレメントに設定すると、そのデザイン エレメントの階層にあるすべての適用可能エレメントに適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ フリップフロップのインスタンスまたはフリップフロップを含むマクロに設定します。
- ・ 属性名 : REG
- ・ 属性値 : CE、TFF

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute REG: string;
```

VHDL 制約を次のように指定します。

```
attribute REG of signal_name: signal is "{CE|TFF}";
```

CE および TFF の詳細については、この制約の「UCF および NCF 構文」セクションを参照してください。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(* REG = {CE|TFF} *)
```

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

### UCF および NCF 構文

UCF 構文は次のとおりです。

```
INST "instance_name" REG = {CE | TFF};
```

- ・ CE を CE 入力付きのフリップフロップ プリミティブに適用すると、CE 入力はマクロセルのクロック イネーブル積項を使用してインプリメントされます。CE 入力のすべてのロジックが 1 つのクロック イネーブル積項でインプリメントできる場合、通常レジスタの CE 入力を使用されます。それ以外の場合は、REG=CE が指定されない限り、CE 入力は D または T ロジック表現に分解されます。XC9500 デバイスではクロック イネーブル積項は使用できず、REG=CE は無視されます。XC9500XL デバイスでは、クロック イネーブル積項は CLR 入力と PRE 入力のいずれも使用しないレジスタでのみ使用できます。
- ・ TFF を指定すると、レジスタは CPLD のマクロセルに T フリップフロップとしてインプリメントされます。D フリップフロップ プリミティブに適用すると、D 入力が T 入力に変換され、T フリップフロップとしてインプリメントされます。通常、D フリップフロップと T フリップフロップの自動変換は、CPLD フィッタで実行されます。

次の文は、XC9500XL のマクロセルのクロック イネーブル積項を使用して CE ピン入力がインプリメントされるように指定します。

```
INST "Q1" REG=CE;
```

### XCF 構文

```
BEGIN MODEL "entity_name"  
NET "signal_name" REG={CE|TFF};  
END;
```

## RLOC

RLOC (相対ロケーション) 制約は、基本的なマップおよび配置制約で、合成制約でもあります。この制約を使用すると、複数のロジック エレメントを 1 つのグループにまとめて、デザイン全体での最終的な配置に関係なく、グループ内でのエレメント同士の位置を相対的に定義できます。

Spartan®-3A、Spartan-3E、Virtex®-4、Virtex-5 では、スライスをベースとした XY 座標を使用して RLOC 制約を指定します。

RLOC 制約を使用すると、ロジック ブロックを相対的に配置するように指定できるため、速度が向上し、チップ リソースを効率的に使用できます。FPGA チップ上で絶対的な配置を指定しなくても、この制約によって関連するデザイン エレメントに順序と構造が指定されます。RLOC 制約を使用すれば、既存のハード マクロに相当する、直接シミュレーション可能なマクロを作成できます。

ユニファイド ライブラリでは、RLOC 制約を BUFT および CLB 関連のプリミティブ (FMAP) に使用できます。また、非プリミティブのマクロ シンボルにも使用できます。BUFT シンボルで RLOC 制約を使用するには、いくつかの制限があります。詳細は、次の「集合の修正子」セクションを参照してください。RLOC 制約は、デコーダ、クロックには使用できませんが、LOC 制約はすべてのプリミティブ (BUFT、CLB、IOB、デコーダ、クロック) に使用できます。

- ・ [相対位置を指定するためのガイドライン](#)
- ・ [RLOC 集合](#)

RLOC 制約は、ロジック ブロックの相対的配置の制御に使用されますが、この制約を設定しても、同じ配線リソースがロジック ブロックを接続するためにインプリメンテーション間で使用されるとは限りません。使用される配線を制御するには、DIRECTED\_ROUTING 制約を使用してください。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

どのデバイス ファミリーにどのエレメントを使用できるかは、ライブラリ ガイドを参照してください。詳細は、[データシート](#)を参照してください。

- ・ レジスタ
- ・ ROM
- ・ RAMS、RAMD
- ・ BUFT – 関連付けられている RPM に RLOC\_ORIGIN が設定されており、RPM の RLOC 値が LOC 値に変換される場合のみ使用可能
- ・ LUT、MUXF5、MUXF6、MUXCY、XORCY、MULT\_AND、SRL16、SRL16E、MUXF7 (Spartan-3、Spartan-3A、Spartan-3E デバイスのみ)
- ・ MUXF8 (FPGA ファミリーすべて)
- ・ ブロック RAM
- ・ 乗算器
- ・ DSP48

## 適用ルール

この制約は、デザイン エレメントの制約なので、ネットには設定できません。デザイン エレメントに設定すると、そのデザイン エレメントの階層にあるすべての適用可能エレメントに適用されます。

NGDBuild は、デザイン階層の下位に LOC 制約を伝搬し、集合の要素ではない適切なオブジェクトに LOC 制約を追加します。RLOC 制約の伝搬は集合内に制限されますが、LOC 制約は開始点から下位階層にあるすべてのエレメントに適用されます。

デザインをフラット化すると、エレメントに設定された RLOC 制約の行番号と列番号は、そのエレメントより下位にある集合エレメントの RLOC 制約の行番号と列番号に追加されます。この機能を使用すると、プリミティブ シンボルに割り当てられている RLOC 値を変更せずに、サブモジュールやマクロの RLOC 値を修正できます。

## 構文

スライスをベースとした XY 座標を使用して RLOC 制約を指定します。

**RLOC=XmY n**

*m* X 軸を表す整数です。

*n* Y 軸を表す整数です。

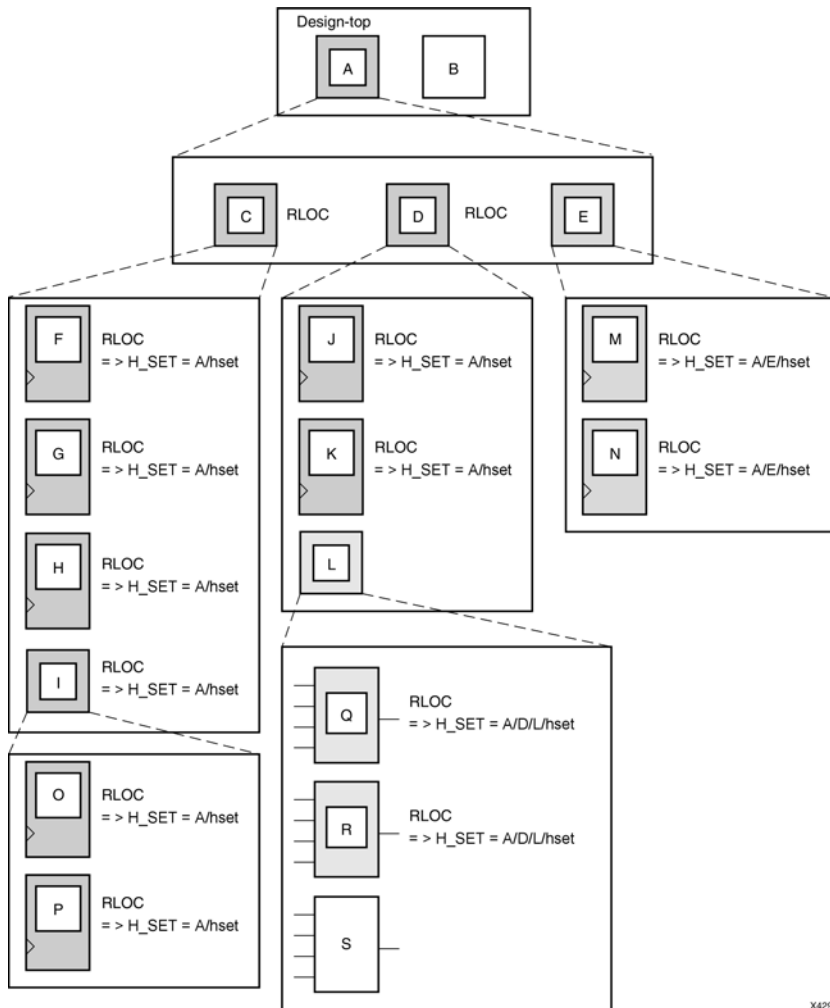
## 集合の修正子

修正子は、その名前が示唆するように、デザイン エLEMENTに関連した RLOC 制約を修正します。修正子は集合に属するすべてのELEMENTの RLOC 制約を修正するので、すべてのELEMENTに簡単かつ直接的に伝搬されるように適用する必要があります。このため、集合の RLOC 修正子は、その集合の開始点に配置します。RLOC 制約には、次のような集合修正子が適用されます。

- ・ RLOC 制約は、集合の階層内でそのELEMENTの下位にあるほかの RLOC 制約の値を修正します。  
集合の種類に関係なく、ELEMENTの RLOC 値 (行、列、拡張子、XY 値) は、常に階層の下位に伝搬され、下位の階層レベルで同じ集合に含まれるELEMENTの RLOC 制約に追加されます。
- ・ **RLOC\_ORIGIN** は、集合のELEMENTをチップ上の特定の位置に固定します。RLOC\_ORIGIN 制約を使用すると、RLOC 値を集合の構造を保った絶対的な LOC 制約に変更できます。  
RLOC\_ORIGIN 制約から LOC 制約への変換は、NGCBuild により行われます。RLOC\_ORIGIN の行と列の値は、下位階層のELEMENTに RLOC 値を加えて行と列の値を変更した後、集合の各ELEMENTに追加されます。最終的な値が、各プリミティブの LOC 制約になります。
- ・ **RLOC\_RANGE** は、集合の配置位置をチップ上のある範囲内に制限できます。  
この場合、集合は最終的に配置されるまで、1 つの単位としてその範囲内を移動できます。集合のすべてのELEMENTが範囲内に収まらなければならないので、集合の空間的な構造を収めることができる範囲を指定することが重要です。  
この制約は、BUFT シンボルが含まれる集合には使用できません。
- ・ **USE\_RLOC** は、集合の特定のELEMENTまたは一部に対して、RLOC 制約を適用するかどうかを指定します。使用可能な値は、TRUE または FALSE です。  
USE\_RLOC 制約は、厳密に階層に基づいて適用されます。あるELEMENTに設定された USE\_RLOC 制約は、そのELEMENTの下位にあり、同じ集合の要素であるすべてのELEMENTに適用されます。集合の開始点を定義するシンボルに USE\_RLOC を設定すると、そのシンボルの下位にあるすべてのELEMENT、すなわち集合全体に制約が適用されます。  
USE\_RLOC=FALSE を適用すると、NCD ファイル内のシンボルから RLOC と集合の制約が削除されます。このプロセスは、RLOC\_ORIGIN 制約の適用プロセスとは異なります。RLOC\_ORIGIN の場合は、すべての集合の制約および RLOC 制約に加えて LOC 制約が生成され、PCF ファイルに出力されます。USE\_RLOC=FALSE 制約を設定すると、元の制約は保持されず、後の段階で再びオンにすることはできません。  
USE\_RLOC 制約をプリミティブ シンボルに直接設定すると、そのシンボルのみに制約が適用されます。

## 集合のリンク

### 集合のリンク



次の例は、複数のデザイン階層に存在する元素同士をリンクする方法を示しています。実際のデザインでは  $RLOC=RmCn$  または  $RLOC=XmX\ n$  のように RLOC を完全に指定する必要があります。

**メモ：** このセクションの図では、集合を区別するため、それぞれの集合に異なる影を付けています。

デザイン階層の 1 つのノードにある RLOC 制約が設定されたデザイン エlement はすべて、RLOC\_ORIGIN 制約や RLOC\_RANGE 制約などのほかの集合制約を設定しない限り、同じ H\_SET 集合に属すると見なされます。この図では、プリミティブおよび非プリミティブ C、D、F、G、H、I、J、K、M、N、O、P、Q、R に RLOC 制約が設定され、B、E、L、S には RLOC 制約は設定されていません。マクロ C と D にはノード A で RLOC 制約が設定されているので、C と D の下位にある RLOC 制約が設定されたすべてのプリミティブは、1 つの H\_SET 集合の Element になります。

この H\_SET 集合はノード A から始まるので、名前は「A/h\_set」になります。H\_SET 集合の開始点は、H\_SET 集合の Element を構成する RLOC 制約が指定されたすべての Element の共通の親で、最も下位にあるものです。

h\_set Element E には RLOC 制約が設定されていないため、A/h\_set 集合にはリンクされません。RLOC 制約が設定されている Element M と N は、Element E の下位にあるので、別の H\_SET 集合の要素になります。この H\_SET 集合の開始点は A/E で、名前は「A/E/h\_set」です。

同様に、プリミティブ Q と R は、Element L がほかのデザイン Element にリンクされていないため、別の H\_SET 集合になります。この H\_SET 集合の最も下位にある共通の親は L で、名前は「A/D/L/h\_set」です。フラット化した後、

NGDBuild はプリミティブ F、G、H、O、P、J、K に  $H\_SET=A/h\_set$  を、プリミティブ Q と R に  $H\_SET=A/D/L/h\_set$  を、プリミティブ M と N に  $H\_SET=A/E/h\_set$  を設定します。

デザインの最上位に集合を作成する状況を考えてみましょう。マクロ A にも RLOC 制約を設定した場合、A はデザインの最上位にあるため親がありません。このような場合、ベース名「h\_set」には階層で修飾した接頭辞は付けられず、 $H\_SET$  集合の名前は「h\_set」になります。

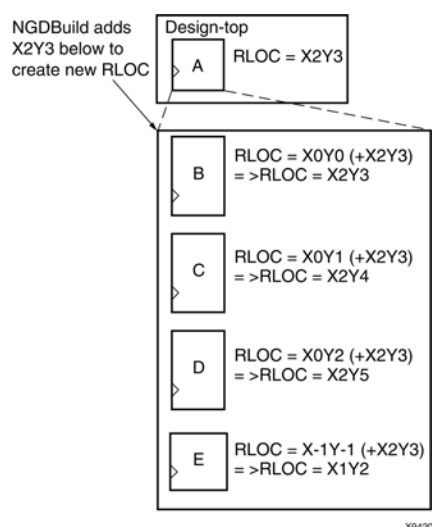
## 集合の編集

RLOC 制約は、プリミティブに RLOC 値 (行番号と列番号にオプションで拡張子を付けたもの) を割り当て、集合の帰属関係を指定して、異なる階層レベルにあるエレメントを接続します。「3 つの  $H\_SET$  集合」の例で、マクロ C と D に設定された RLOC 制約は、これらマクロの下位にある RLOC 制約が設定されたすべてのオブジェクトをリンクします。また、RLOC 制約を使用して、下位階層にある制約の RLOC 値を修正できます。つまり、エレメントの RLOC 値は、デザイン階層で特定のエレメントより下位にある同じ  $H\_SET$  集合のほかのエレメントの RLOC 値に影響を与えます。

デザインをフラット化すると、エレメントの RLOC 制約の XY 値は、そのエレメントより下位の階層にある集合エレメントの RLOC 制約の XY 値に加えられます。この機能を使用すると、プリミティブ シンボルに割り当てられている RLOC 値を変更せずに、サブモジュールやマクロの RLOC 値を修正できます。

次のセクションでは、集合の修正における階層の影響について説明します。

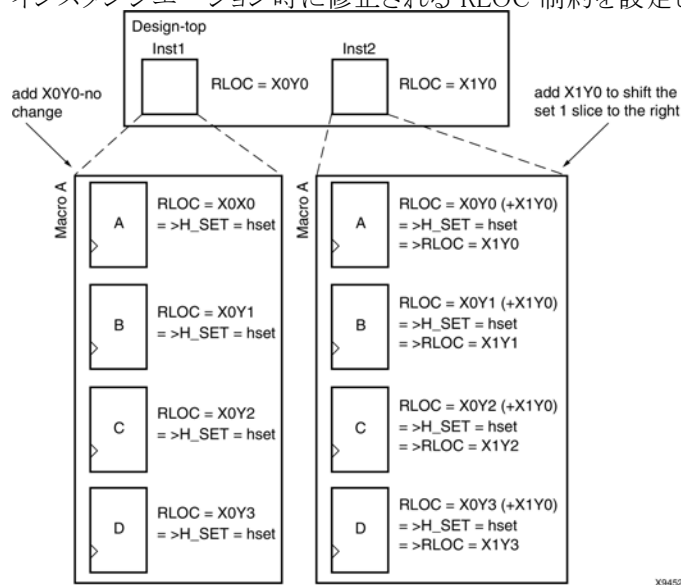
## 下位階層に RLOC 値を加算する方法 (スライスをベースとした XY 座標)



この例では、下位階層のエレメントに RLOC 値を加算するプロセスを示しています。カッコ内の行と列の値は、MAP により実行される加算を示します。先頭に => が付いている斜体文字は、デザイン決定プロセスのときに自動的に追加され、ユーザーが追加したオリジナルの RLOC 制約の代わりに使用されます。

## 同じマクロの RLOC 値の修正と 1 つの集合としてのリンク

下位階層の RLOC 値を修正できると、同じマクロを複数回インスタンスエートする場合に便利です。一般に、マクロは、インスタンスエーション時に修正される RLOC 制約を設定してデザインされます。



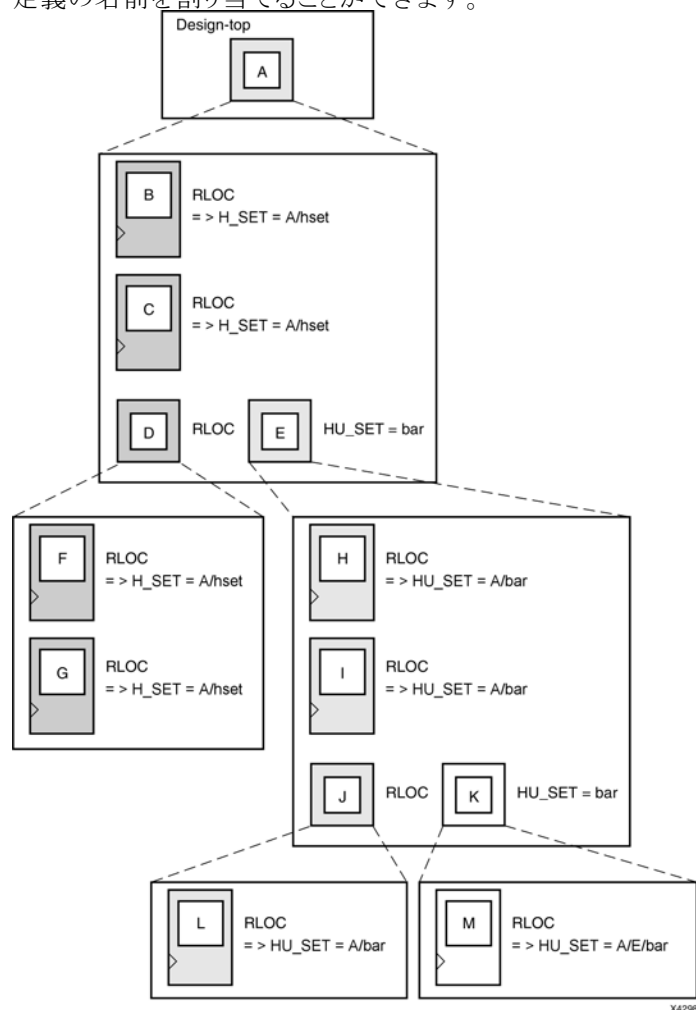
この例は、前の例のバリエーションの 1 つです。この例では、Inst1 と Inst2 の RLOC 制約が、1 つの H\_SET 集合に含まれるすべてのオブジェクトをリンクしています。

Inst1 マクロの RLOC=X0Y0 修正子はそれより下位のオブジェクトに影響しないので、H\_SET がオブジェクトに追加されるだけで、RLOC 値はそのままです。Inst2 マクロの RLOC=X0Y1 修正子の場合は、斜体文字で示すように、H\_SET が追加されるだけでなく、それより下位にあるオブジェクトの RLOC 値が変更されます。



## H\_SET 集合からのエレメントの分離

HU\_SET 制約は、H\_SET (階層集合) のバリエーションです。HU\_SET 制約は、新しい集合の開始点を定義します。H\_SET と同様に、HU\_SET はデザイン階層によって定義されますが、HU\_SET 制約を使用すると、HU\_SET にユーザー定義の名前を割り当てることができます。



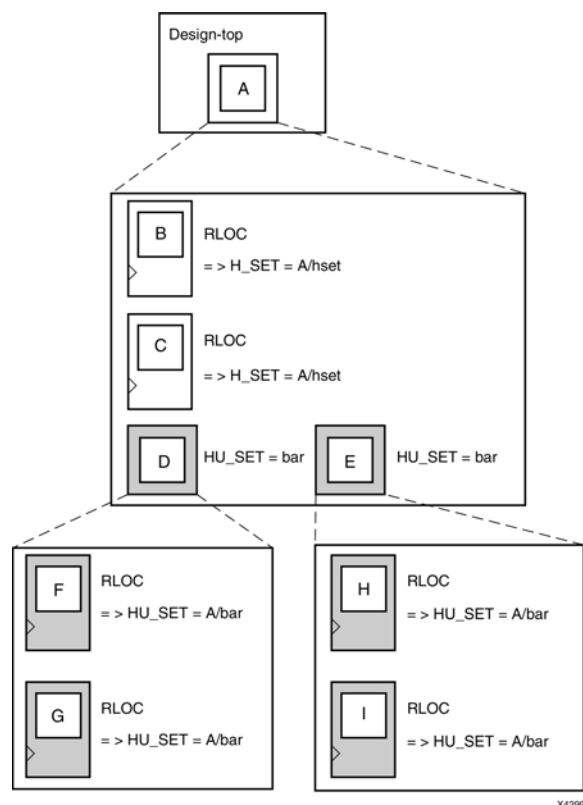
この例は、HU\_SET 制約を使用してエレメントを集合の要素として指定し、階層内で RLOC 制約が設定されたエレメントを H\_SET 集合から分離し、集合の識別名を生成する様子を示しています。

E に設定されたユーザー定義の HU\_SET 制約は、その下位にあるデザイン エレメント H、I、J、K、L、M を、プリミティブ要素 B、C、F、G を含む H\_SET=A/h\_set から分離します。E で定義された HU\_SET 集合には、エレメント H、I および J を介した L が含まれます。A は HU\_SET 集合に含まれるすべてのエレメントの共通の親です。

HU\_SET 集合のすべてのエレメントに共通の親で最も下位にあるのが A であるため、MAP はエレメント E の名前「bar」に階層修飾子を付けて「A/bar」とし、これをプリミティブ H、I、L に追加します。K の HU\_SET 制約は M を含む別の集合を開始し、この集合にはマップ処理後に HU\_SET=A/E/bar が追加されます。

2 つの HU\_SET 制約で同じ名前が使用されていますが、これらの制約はそれぞれ異なる階層レベルにあるシンボルに追加されるので、2 つの異なる集合が定義されます。

## 2 つの HU\_SET 集合のリンク

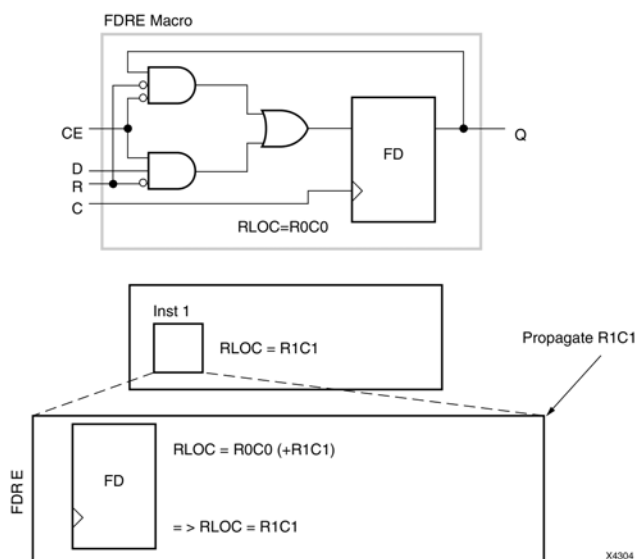


この例では、同じノードにあるエレメントに HU\_SET 制約を使用して同じ識別名を付けることにより、それらがリンクされる様子を示しています。2 つのエレメント D と E には「bar」という同じ名前が付いているため、D と E の下位にある RLOC 制約が設定されたエレメントは同じ HU\_SET の一部になります。

## ザイリンクス マクロでの RLOC の使用

ザイリンクスが提供するフリップフロップ マクロには、下位のプリミティブに RLOC=R0C0 制約が含まれるため、マクロ シンボルに RLOC を設定できます。このシンボルは、マクロ シンボルが含まれる集合に下位のプリミティブをリンクします。

このため、ザイリンクスのフリップフロップ マクロのインスタンスに、適切な RLOC 制約を設定するだけですみます。MAP により、目的の値になるように、指定された RLOC 値が下位のプリミティブに加えられます。



この例では、マクロのインスタンス (Inst1) に RLOC = R1C1 制約が設定されています。これはマクロ内のフリップフロップに設定された RLOC 制約の R0C0 値に追加され、新しい RLOC 値 (R1C1) が得られます。

RLOC=X1Y1 制約をマクロの Inst1 に設定した場合、これがマクロ内のフリップフロップに適用された RLOC 制約の X0Y0 値に追加され、新しい RLOC 値 (X1Y1) が得られます。

RLOC 制約をフリップフロップ マクロ シンボルに設定しない場合、下位のプリミティブ シンボルは集合の唯一の要素になります。集合の唯一の要素であるプリミティブ、または下位に RLOC オブジェクトがないマクロからは、RLOC 制約が削除されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ インスタンスに設定します。
- ・ 属性名 : RLOC
- ・ 属性値 : 「[構文](#)」を参照してください。

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute rloc: string;
```

FPGA デバイスの場合は、次のように指定します。

```
attribute rloc of {component_name|entity_name|label_name}: {component|entity|label} is "[element]XmYn[.extension]";
```

有効な値については、「[相対位置を指定するためのガイドライン](#)」を参照してください。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

次に、VHDL の generate 文で RLOC を使用する場合のコード例を示します。これは、インスタンス化された複数の FDE で RLOC を自動的に生成する簡単な方法です。この方法は、ほぼすべてのプリミティブで使用できます。

メモ: ユーザーが itoa 関数を作成する必要があります。

```
LEN:for i in 0 to bits-1 generate
  constant row :natural:=((width-1)/2)-(i/2);
  constant column:natural:=0;
  constant slice:natural:=0;
  constant rloc_str : string := "R" & itoa(row) & "C" & itoa(column) & ".S" & itoa(slice);
  attribute RLOC of U1: label is rloc_str;
begin
  U1: FDE port map (
    Q=> dd(j),
    D=> ff_d,
    C=> clk,
    CE => lcl_en(en_idx));
end generate LEN;
```

## Verilog 構文

この制約はインスタンス化の直前に入力します。

FPGA デバイスの場合は、次のように指定します。

```
(* RLOC = "[element]XmYn[.extension]" *)
```

有効な値については、「[相対位置を指定するためのガイドライン](#)」を参照してください。Verilog 構文については、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

FPGA デバイスの場合、次の文は、X 座標の +4 と Y 座標の +4 にあるスライスに FF1 のインスタンスが配置されるように指定します。

```
INST "/V2/design/FF1" RLOC=X4Y4;
```

## XCF 構文

Virtex-4 および Virtex-5 デバイスの場合:

```
BEGIN MODEL "entity_name "
INST "instance_name " rloc=[element]XmYn[.extension] ;
END;
```

## PlanAhead の構文

PlanAhead™ を使用したエリア グループ制約の作成方法は、『PlanAhead ユーザー ガイド』の「Floorplanning the Design (デザインのフロアプラン)」を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## 相対位置を指定するためのガイドライン

相対位置にエレメントを割り当てるスライス ベースの座標システムでは、一般的な構文が使用されます。

```
RLOC=Xm Yn
```

この場合、次を表します。

- ・  $m$  は X 軸 (左/右) の相対値、 $n$  は Y 軸 (上/下) の相対値です。
- ・ X 値と Y 値には、0 を含む正または負の整数を指定します。

RLOC 制約の X 値と Y 値はデザイン エLEMENT 間の順序と関係を指定するだけで、絶対的なチップ位置を定義するわけではないので、番号に負数を使用できます。RLOC 制約には任意の整数を使用できますが、簡単でわかりやすいように、小さい数の使用をお勧めします。

RLOC の指定で重要なのは、X 座標と Y 座標の絶対値ではなく、相対値 (差) です。たとえば、デザイン エLEMENT A に RLOC=X3Y4 という制約を、デザイン エLEMENT B に RLOC=X6Y7 という制約を指定した場合、X 座標の絶対値 (3 と 6) は重要ではありません。重要なのはこれらの値の差であり、この場合は 6 から 3 を引いた値 3 で、デザイン エLEMENT B がデザイン エLEMENT A から 3 スライス離れるように指定されます。

この情報を得るため、デザインのインプリメンテーション中に値が正規化されます。デザイン エLEMENT B は、デザイン エLEMENT A から上方向に 3 列 (7 から 4 を引いた値) 離れた位置に指定されます。上記の例では、正規化によってデザイン エLEMENT A の RLOC が X0Y0 に、デザイン エLEMENT B の RLOC が X3Y3 になります。

Spartan®-3 および Virtex®-4 以降のデバイスの場合、XY 軸を使用してチップの左下隅からスライスに番号が付けられています。X 軸では右に移動するに従って増加し、Y 軸では上に移動するに従って増加します。RLOC 制約は、このデフォルト座標をベースとした規則に従います。

#### 4 つのフリップ フロップ プリミティブの RLOC 指定



X9419

次の図に、RLOC 制約の使用例を示します。(a) では、A、B、C、D という 4 つのフリップフロップ プリミティブに RLOC 制約を割り当てています。これらの RLOC 制約では、各フリップフロップは異なるスライスに配置され、図のように下から A、B、C、D という順に並びます。

2 つ以上のフリップフロップ プリミティブを 1 つのスライスに配置するには、の (b) に示すように RLOC を指定します。この場合、A と B は 1 つのスライスに配置され、そのすぐ右側のスライスに C と D が配置されます。

## RLOC 集合

RLOC 制約は、関連するデザイン エLEMENT の順序と構造を指定します。このセクションでは、RLOC 制約が適用された関連デザイン エLEMENT のグループである RLOC 集合について説明します。たとえば、「[4 つのフリップフロップ プリミティブの RLOC 指定](#)」の 4 つのフリップフロップは、RLOC 制約によって関連付けられ、集合を形成しています。集合内のELEMENTは、RLOC 制約によって同じ集合内のほかのELEMENTに関連付けられます。集合内のELEMENT同士を関連付けるため、集合内の各ELEMENTに RLOC 制約を設定する必要があります。複数の集合を作成できますが、1 つのデザイン ELEMENT は 1 つの集合にしか含めることができません。

集合は、集合パラメータを使用して明示的に定義するか、デザイン階層の構造によって間接的に定義できます。

次に示す 4 種類の規則が各集合に適用されます。

- ・ 定義規則は、集合内の帰属関係の条件を定義します。
- ・ リンク規則は、1 つの集合を形成するためにELEMENT同士をリンクする方法を指定します。
- ・ 修正規則は、集合内のすべてのELEMENTの RLOC 値を修正するパラメータを指定します。
- ・ 命名規則は、集合を命名する方法を指定します。

これらの規則について、次のセクションで説明します。

次のセクションでは、U\_SET、H\_SET、HU\_SET という 3 つの集合制約について説明します。ELEMENTを集合に属するように指定するには、これらの集合制約のいずれかと RLOC 制約の両方を設定する必要があります。

## U\_SET

U\_SET 制約は、デザイン階層全体に分散されているデザイン ELEMENT に RLOC 制約を設定し、1 つの集合にグループ化できるようにします。U\_SET の U という文字は、集合がユーザー定義であることを表します。

U\_SET 制約を使用すると、デザイン階層で直接関連していない場合でも、ELEMENTをグループ化できます。U\_SET 制約をデザイン ELEMENT に設定すると、集合のELEMENTを明示的に定義できます。

U\_SET 制約は、プリミティブ シンボル、非プリミティブ シンボルのどちらにも設定でき、デザイン ELEMENT はデザイン階層のどのレベルに位置していてもかまいません。非プリミティブ シンボルに設定した場合、U\_SET 制約は、そのシンボルの下位階層にあり、RLOC 制約が設定されているプリミティブ シンボルにも適用されます。

U\_SET 制約の構文は、次のとおりです。

**U\_SET**=*set\_name*

この場合、それぞれ次を表します。

*set\_name* は、ユーザーが指定する集合の識別名です。

RLOC 制約が指定されているデザイン ELEMENT で、同じ U\_SET 制約を設定したものはすべて、同じ集合に属します。そのため、名前がデザイン内のほかの集合名と重複しないようにしてください。

## H\_SET

U\_SET 制約はデザイン ELEMENT に設定して明示的に定義できますが、H\_SET (階層集合) はデザイン階層によって暗示的に定義されます。デザイン階層とELEMENTに設定された RLOC 制約の組み合わせによって、H\_SET が定義されます。

帰属関係を表すために、H\_SET 制約をデザイン エLEMENT に設定することはできません。集合はデザイン階層によって自動的に定義されます。

デザイン階層の 1 つのノードにある RLOC 制約が設定されたデザイン エLEMENT はすべて、RLOC\_ORIGIN や RLOC\_RANGE などのほかの種類の集合制約を設定しない限り、同じ H\_SET 集合に属すると見なされます。RLOC\_ORIGIN、RLOC\_RANGE、U\_SET、HU\_SET のいずれかの制約をELEMENT に設定すると、そのELEMENT は H\_SET 集合から除外されます。

H\_SET 制約は相対配置マクロの基本的なメカニズムであるため、ほとんどのデザインには H\_SET 制約のみが含まれます。RLOC\_ORIGIN 制約と RLOC\_RANGE 制約については、「[集合の修正子](#)」を参照してください。

NGDBuild は H\_SET 集合を認識し、その名前 (識別名) を導き出し、集合のELEMENT に H\_SET 制約を設定して、出力ファイルに書き込みます。

## HU\_SET

HU\_SET 制約は、H\_SET (階層集合) のバリエーションです。H\_SET と同様に、HU\_SET はデザイン階層によって定義されますが、HU\_SET 制約を使用すると、HU\_SET にユーザー定義の名前を割り当てることができます。

HU\_SET 制約の構文は、次のとおりです。

**HU\_SET=***set\_name*

この場合、それぞれ次を表します。

*set\_name* は、集合の識別名です。デザイン内のほかの集合名とは異なる名前を指定する必要があります。

このユーザー定義名が、HU\_SET 集合のベース名になります。H\_SET では、集合ELEMENT の共通の親で最も下位にあるものの階層名がベース名「h\_set」の前に付けられます。HU\_SET 集合の場合も同様で、集合ELEMENT 共通の親で最も下位にあるものの階層名がユーザー定義のベース名の前に付けられます。

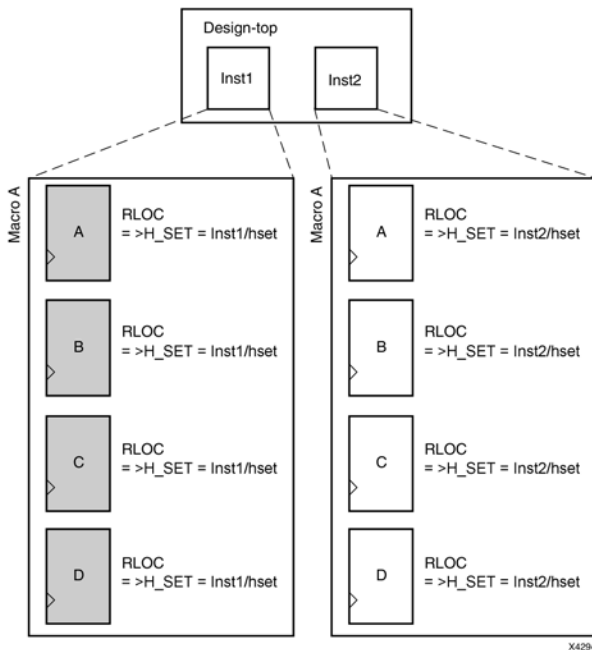
集合に階層で識別された固有の名前が付けられるようにするため、デザインがマップされ階層名が接頭辞として付けられる前に、ベース名を定義する必要があります。

HU\_SET 制約は、新しい集合の開始点を定義します。同じノードにあるデザイン エLEMENT で、HU\_SET 制約のユーザー定義の値が同じであるものは、すべて同じ HU\_SET 集合のELEMENT になります。また、HU\_SET 制約と共に RLOC 制約もELEMENT に設定できます。

H\_SET 制約と RLOC 制約が設定されたELEMENT は、階層の上位と下位にあるELEMENT で RLOC が設定されているものにリンクされます。HU\_SET 制約の場合は、RLOC 制約と一緒に設定しても、デザイン エLEMENT は、同じ階層レベルまたはそれより上位にある RLOC 制約が設定されたほかのELEMENT に自動的にリンクされません。



## 2 回インスタンス化されるマクロ A



**メモ:** この図と次のセクションで示す図では、=> の後の斜体文字は、デザインフラット化の際に NGDBuild によって追加されます。ほかのすべての文字は、ユーザーが入力します。

この図に、H\_SET (階層集合) の使用例を示します。この図では、RLOC 制約の最初の「RLOC」部分のみが示されています。実際のデザインでは、RLOC 制約を RLOC=R *mCn* (Spartan®-3)、RLOC=X*mYn* (Virtex®-4 以降のデバイス) を使用して完全に指定する必要があります。この例では、まず 4 つのフリップフロップ A、B、C、D に RLOC 制約を設定してマクロ A を作成し、このマクロを Inst1 および Inst2 としてデザインで 2 回インスタンス化しています。

RLOC 制約が設定されたエレメントが 2 つの異なる階層レベルに含まれているため、デザインのフラット化の際に 2 つの異なる H\_SET 集合が認識されます。NGDBuild は、Inst1 でインスタンス化されるマクロには H\_SET=Inst1/h\_set が、Inst2 でインスタンス化されるマクロには H\_SET=Inst2/h\_set、のように最適な H\_SET 制約を作成して設定します。デザインインプリメンテーションプログラムは、RLOC 制約で指定された集合内の相対的な順序を使用して、2 つの集合をそれぞれ 1 つの単位として別々に配置します。ただし、2 つの集合は、互いに完全に独立していると考えられます。

H\_SET 集合の名前は、すべての RLOC エレメントが含まれる階層のノードまたはシンボルから導き出されます。Inst1 は図の左側に示した RLOC が設定された 4 つのフリップフロップエレメントを含むノード (インスタンス化されるマクロ) です。したがって、この H\_SET 集合の名前には、階層名「Inst1」の後に「h\_set」を付けます。

Inst1 シンボルは H\_SET の開始点と考えられ、これを H\_SET 全体の名前として H\_SET 全体を修正する制約を設定します。集合を修正する制約については、「[ネットフラグの保存 \(SAVE NET FLAG\)](#)」を参照してください。

この図は、階層の 1 つのレベルに制限される集合について、簡単な使用例を示しています。リンクと修正を使用すると、複数の階層レベルにわたってリンクされた H\_SET 集合も作成できます。

リンクを使用すると、異なる階層レベルにあるエレメントをリンクし、集合を形成できます。修正を使用すると、複数の階層にまたがる集合のエレメントの RLOC 値を修正できます。

## RLOC 集合の要約

RLOC 集合のタイプと集合のエレメントを識別する制約を、次の表に要約します。



## 集合のタイプ

タイプ	定義	命名	リンケージ	修正
U_SET= name	ユーザーが指定する U_SET 値が同じエレメントは、すべて同じ U_SET 集合のエレメントになります。	集合名は、階層の修飾子を除いたユーザー定義の名前と同じです。	U_SET 値が同じエレメントをすべてリンクします。	U_SET を指定したエレメントのいずれかに RLOC_ORIGIN または RLOC_RANGE を適用すると、U_SET が修正されます。
HU_SET= name	階層の修飾子が付いた名前が同じエレメントは、すべて同じ集合の要素となります。	エレメントの共通の親をユーザー定義名の前に付け、集合名とします。	同じノードにあり、HU_SET 値が同じエレメントをリンクします。また、階層の下位にある RLOC が設定されたエレメントもリンクします。	集合の開始点は、同じノードにあり HU_SET 値が同じエレメントで構成されます。HU_SET 集合の開始エレメントのいずれかに、RLOC_ORIGIN または RLOC_RANGE を適用できます。

## RLOC\_ORIGIN

RLOC\_ORIGIN 制約は、集合のエレメントをチップ上の特定の位置に固定する配置制約です。この制約では単一の位置を指定する必要があり、位置の範囲や複数の位置のリストは指定できません。詳細は、「[RLOC](#)」の「集合の修正子」を参照してください。

RLOC\_ORIGIN は、BUFT シンボルを含む集合には設定する必要がありますが、BUFT インスタンスには設定できません。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

集合の要素であるインスタンスまたはマクロ

## 適用ルール

この制約はマクロの制約なので、ネットには設定できません。

RLOC\_ORIGIN 制約を H\_SET (階層集合) と共に使用する場合は、H\_SET 集合の開始点となるエレメント、すなわち集合の全エレメントに共通の親に設定する必要があります。

RLOC\_ORIGIN 制約を HU\_SET 制約に適用する場合は、HU\_SET 集合の開始点となるエレメント、すなわち HU\_SET 制約が設定されているエレメントに配置します。

同じノードで HU\_SET 制約を使用して複数のエレメントをリンクする場合は、複数の RLOC\_ORIGIN 制約が HU\_SET 集合に適用されないようにするため、RLOC\_ORIGIN 制約は 1 つのエレメントのみに設定してください。

同様に、RLOC\_ORIGIN 制約を U\_SET 制約と共に使用する場合、U\_SET 制約が設定されているエレメントの 1 つにのみ設定します。RLOC 制約のみが設定されたエレメントに RLOC\_ORIGIN 制約を設定すると、そのエレメントの帰属関係はすべての集合で削除され、そのエレメントが、RLOC\_ORIGIN 制約が設定された H\_SET 集合の開始点と見なされます。

## 構文

RLOC 集合に 1 つの基点を指定するには、次の構文を使用します。これは、回路図で RLOC\_ORIGIN 制約を配置する場合と同じです。

*set\_name* **RLOC\_ORIGIN=X***m* **Y***n*

- ・ *set\_name* は、任意のタイプの RLOC 集合 (U\_SET、HU\_SET、または生成される H\_SET) の名前です。
- ・ 基点は、RLOC=X0Y0 にあるエレメントの位置を示す X 値と Y 値で表されます。

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ 集合の要素であるインスタンスに設定します。
- ・ 属性名 : RLOC\_ORIGIN
- ・ 属性値 : 詳細は、「UCF および NCF 構文」を参照してください。

## VHDL 構文

VHDL 制約を次のように宣言します。

**attribute rloc\_origin: string;**

VHDL 制約を次のように指定します。

**attribute rloc\_origin of** {*component\_name* | *entity\_name* | *label\_name*} **:** {*component* | *entity* | *label*} **is** " *value* " ;

For Spartan®-3, Spartan-3A, Spartan-3E, Virtex®-4 and Virtex-5 devices, *value* is **X** *mYn*.

有効な値については、下記の「UCF および NCF 構文」を参照してください。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

**( \* RLOC\_ORIGIN = " value" \* )**

Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 デバイスの場合、*value* は **X** *mYn* となります。

有効な値については、下記の「UCF および NCF 構文」を参照してください。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## スライスをベースとして XY 座標を使用する UCF および NCF 構文

この制約は、Spartan-3、Spartan-3A、Spartan-3E、Virtex-4 および Virtex-5 デバイ스에適用できます。

**RLOC\_ORIGIN=X** *mYn*

*m* および *n* は、相対的な X 座標および Y 座標を表す正または負の整数 (0 を含む) です。

次の文は、集合の要素である FF1 のインスタンスが、X4Y4 にあるスライスを基準として配置されるように指定します。たとえば、FF1 に RLOC=X0Y2 を設定した場合、FF1 のインスタンスは、X4 から右に 0 列、Y4 から上に 2 行 (X4Y6) の位置にあるスライスに配置されます。

```
INST "/archive/designs/FF1" RLOC_ORIGIN=X4Y4;
```

## PlanAhead からの設定

PlanAhead™ を使用したエリア グループ制約の作成方法は、『PlanAhead ユーザー ガイド』の「Floorplanning the Design (デザインのフロアプラン)」を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## RLOC\_RANGE

RLOC\_RANGE 制約は、基本的なマップおよび配置制約で、RLOC\_ORIGIN 制約と似ていますが、集合の要素をチップ上の特定の範囲に制限するという点で異なります。位置の範囲や複数の位置のリストは、集合の基点だけでなく、RLOC が設定されている適用可能なすべてのエレメントに適用されます。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

集合の要素であるインスタンスまたはマクロ

## 適用ルール

この制約はマクロの制約なので、ネットには設定できません。

矩形領域は、集合の基点だけでなく、相対配置マクロのすべてのエレメントに適用されます。

RLOC\_RANGE 制約の値は、単にエレメントの RLOC 値に加えられるわけではなく、下位エレメントの RLOC 制約の値は変更しません。この制約は、MAP により集合の全エレメントに自動的に設定される追加の制約です。

RLOC\_RANGE 制約は、RLOC\_ORIGIN 制約とまったく同じ方法でデザイン エレメントに設定されます。RLOC\_RANGE 制約の値は、RLOC\_ORIGIN の値と同様にチップ上の位置に対応するので、0 以外の正の数を指定する必要があります。

デザイン ネットリストと UCF ファイルの両方で RLOC\_ORIGIN 制約または RLOC\_RANGE 制約が RLOC 集合に設定されている場合、UCF ファイルの制約がネットリストの制約よりも優先されます。

## 構文

```
RLOC_RANGE=Xm1 Yn1:X m2Yn2
```

この場合、次を表します。

相対的な X 値 ( $m1$ ,  $m2$ ) と Y 値 ( $n1$ ,  $n2$ ) には、次を指定できます。

- 0 以外の正の整数
- ワイルドカード (\*)

この構文では、次のような 3 種類の範囲指定が可能です。

- ・  $XmYn1:Xm2Yn2 - XmYn1$  and  $Xm2Yn2$  で囲まれた矩形領域
- ・  $X*Yn1:X*Ym2 - Y$  軸の  $n1$  から  $n2$  までの領域 ( $X$  の値は問わない)
- ・  $XmY*:Xm2Y* - X$  軸の  $m1$  から  $m2$  までの領域 ( $Y$  の値は問わない)

ワイルドカードを使用する 2 番目と 3 番目の指定の場合、区切り記号のコロンの前後でワイルドカード文字 (\*) を異なる位置に適用すると、エラーが発生します。たとえば、 $X*Y1:X2Y*$  と指定すると、コロンの左側では  $X$  値に、コロンの右側では  $Y$  値にワイルドカードが適用されているため、エラーになります。

#### 範囲の指定

範囲を指定するには、次の構文を使用します。これは、回路図で RLOC\_RANGE 制約を配置するのと同じです。

*set\_name* **RLOC\_RANGE=X m1Yn1 :Xm2Y n2**

範囲は、長方形のエリアが識別されます。範囲を指定する  $X$  値または  $Y$  値の代わりに、ワイルドカード文字 (\*) を使用できます。

#### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

#### 回路図

- ・ 集合の要素であるインスタンスに設定します。
- ・ 属性名 : RLOC\_RANGE
- ・ 属性値 : 「UCF および NCF 構文」を参照してください。

#### VHDL 構文

VHDL 制約を次のように宣言します。

**attribute rloc\_range: string;**

VHDL 制約を次のように指定します。

**attribute rloc\_range of {component\_name|entity\_name|label\_name}: {component|entity|label} is "value";**

Spartan®-3、Spartan-3A、Spartan-3E、Virtex®-4、Virtex-5 デバイスの場合、 **$XmYn1:Xm2Yn2$**  です。

有効な値については、下記の「UCF および NCF 構文」を参照してください。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

#### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

**(\* RLOC\_RANGE = "value" \*)**

Spartan-3、Spartan-3A、Spartan-3E、Virtex-4、Virtex-5 デバイスの場合、 **$XmYn1:Xm2Yn2$**  です。

有効な値については、下記の「UCF および NCF 構文」を参照してください。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

このセクションは、Spartan-3 および Virtex-4 以降のデバイスにのみ適用されます。

**RLOC\_RANGE=Xm/Yn1:Xm2Yn2**

相対的な X 値 ( $m1$ ,  $m2$ ) と Y 値 ( $n1$ ,  $n2$ ) には、次を指定できます。

- 正の整数 (0 を含む)
- ワイルドカード (\*)

次の文は、マクロ MACRO4 のインスタンスが、集合内のほかの要素に相対して、左下角が X4Y4、右上角が X10Y10 で囲まれた矩形領域に配置されるように指定します。

**INST "/archive/designs/MACRO4" RLOC\_RANGE=X4Y4:X10Y10;**

## XCF 構文

**MODEL "entity\_name" rloc\_range=value;**

**BEGIN MODEL "entity\_name"**

**INST "instance\_name" rloc\_range=value;**

**END;**

## SAVE NET FLAG

SAVE NET FLAG 制約は、基本的なマップ制約です。この制約をネットまたは信号に設定すると、未接続の信号が削除されなくなり、デザインのマップや配置配線に影響します。

この制約は、ロードのない信号やドライバのない信号が削除されないようにします。ロードのない信号の場合、S 制約がその信号に接続された疑似 OBUF ロードの働きをします。ドライバのない信号では、S 制約はその信号に接続された疑似 IBUF ドライバの働きをします。

ネットに S 制約が設定されていない場合、I/O プリミティブへのパスを介して確認または制御されない信号はすべて削除されます。

S 制約により、信号に接続されたロジックがトリムされなくなる場合もあります。SAVE NET FLAG は、「S NET FLAG」とも呼ばれます。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能要素

- ・ ネット
- ・ 信号

## 適用ルール

この制約は、ネットまたは信号の制約なので、デザイン エlement には適用できません。

S 制約は、未接続の信号が削除されないようにします。ネットに S 制約が設定されていない場合、ロジックや I/O プリミティブに接続されていない信号はすべて削除されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ ネットまたは信号に設定します。
- ・ 属性名 : S
- ・ 属性値 : TRUE、FALSE

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute S: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

次の文は、ネットまたは信号 \$SIG\_9 が削除されないように指定します。

```
NET $SIG_9 S;
```

## XCF 構文

```
BEGIN MODEL entity_name
```

```
NET "signal_name" s=true;
```

```
END;
```

## SCHMITT\_TRIGGER

この制約は、入力パッドがシュミットトリガ (ヒステリシス) でコンフィギュレーションされるよう設定します。デザインの入力パッドに適用されます。

## アーキテクチャ サポート

この制約は、CoolRunner™-II デバイスにのみ適用できます。

## 適用可能エレメント

すべての入力パッドおよびパッド ネット

## 適用ルール

この制約は、ネットまたは信号の制約なので、マクロ、エンティティ、またはモジュールには設定できません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ ネットに設定します。
- ・ 属性名 : SCHMITT\_TRIGGER
- ・ 属性値 : TRUE、FALSE

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute SCHMITT_TRIGGER: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、[「Verilog」](#)を参照してください。

### UCF および NCF 構文

```
NET "mysignal" SCHMITT_TRIGGER;
```

### XCF 構文

```
BEGIN MODEL "entity_name"
```

```
NET "signal_name" SCHMITT_TRIGGER=true;
```

```
END;
```

## SLEW

各出力のスルー レート (遷移レート) ビヘイビアをデバイスに定義します。出力ポート、または双方向ポートに設定可能で、SLOW (デフォルト)、FAST または QUIETIO (Spartan®-3A および Spartan-3A DSP) に指定できます。一番遅い SLEW 属性を使用しますが、シグナル インテグリティの問題を最小限に抑えるために I/O タイミング要件が満たされるように設定します。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイスに適用できます。

## 適用可能エレメント

出力プリミティブ、出力パッド、双方向パッド

SLEW 制約は、UCF ファイル内で、パッド コンポーネントに接続されているネットにも設定できます。ネットに設定した SLEW 制約は、NGCBuild により NGD ファイル内のパッド インスタンスに挿入され、マップで処理されます。次の構文を使用してください。

```
NET "net_name" slew={FAST|SLOW};
```

## 適用ルール

SLEW 属性は最上位の、出力ポートまたは双方向ポートのみに設定します。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

出力ポートまたは双方向ポートに設定します。

- ・ 属性名 : SLEW
- ・ 属性値 : FAST、SLOW、QUIETIO (Spartan-3A のみ)

### VHDL 構文

SLEW は、アーキテクチャ宣言と最上位 VHDL ファイルの begin 文の間で次のように宣言します。

```
attribute SLEW: string;
```

After SLEW has been declared, specify the VHDL constraint as follows:

```
attribute SLEW of {top_level_port_name}: signal is "value";
```

value には、SLOW、FAST、または QUIETIO (Spartan-3A のみ) を指定します。

例 :

```
entity top is
port (FAST_OUT: out std_logic);
end top;

architecture MY_DESIGN of top is
attribute SLEW: string;
attribute SLEW of FAST_OUT: signal is "FAST";
begin
```

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

### Verilog 構文

最上位の Verilog コードの port 宣言の前に、次のように記述します。

```
(* SLEW="value" *)
```

value には、SLOW、FAST、または QUIETIO (Spartan-3A のみ) を指定します。



例 :

```
module top (  
  (* SLEW="FAST" *) output FAST_OUT  
);
```

Verilog 構文の詳細については、「[Verilog](#)」を参照してください。

### UCF および NCF 構文

出力ポートまたは双方向ポートに設定します。

```
NET "top_level_port_name" SLEW="value";  
value FAST、SLOW、QUIETIO (Spartan-3A のみ)
```

例 :

```
NET "FAST_OUT" SLEW="FAST";
```

### PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

### PACE の構文

SLEW 属性を PACE で指定できます。[Design Objects] ウィンドウでピンに値を設定します。

**メモ :** PACE は、CPLD でのみサポートされます。

## SLOW

SLOW は基本的なフィッタ制約で、スルー レートを制限します。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイスに適用できます。

## 適用可能エレメント

- ・ 出力プリミティブ
- ・ 出力パッド
- ・ 双方向パッド

SLOW 制約は、UCF ファイル内で、パッド コンポーネントに接続されているネットにも設定できます。ネットに設定した SLOW 制約は、NGCBuild により NGD ファイル内のパッド インスタンスに挿入され、マップで処理されます。これには、次の UCF 構文を使用します。

```
NET "net_name" SLOW;
```

## 適用ルール

基本的にはネットには設定できませんが、ネットがパッドに接続されている場合は例外です。この場合、SLOW はパッド インスタンスに設定されていると見なされます。

デザイン エLEMENT に設定すると、そのデザイン エLEMENT の階層にあるすべての適用可能ELEMENT に適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : SLOW
- ・ 属性値 : TRUE、FALSE

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute SLOW : string;
```

VHDL 制約を次のように指定します。

```
attribute SLOW of {signal_name|entity_name}: {signal|entity} is "{TRUE|FALSE}";
```

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

### UCF および NCF 構文

次の文は、ELEMENT y2 のインスタンスに低速スルー レートを設定します。

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

次の文は、net1 が接続されているパッドに低速スルー レートを設定します。

```
NET "net1" SLOW;
```

### PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## STEPPING

STEPPING 制約を使用すると、シリコン上に記載されたステップ レベルに一致する値が割り当てられます。ステップ レベルよりデバイスの性能が判断できます。ザイリンクスでは、この制約を使用してデザインにステップ レベルを設定することを推奨しています。設定がない場合は、デフォルトのターゲット デバイスが使用されます。

STEPPING の詳細は、ザイリンクスの[アンサー #20947](#) の「ステッピングに関するよくある質問」を参照してください。

## アーキテクチャ サポート

- ・ CoolRunner™-II CPLD アーキテクチャ
- ・ Spartan®-3A、Spartan-3E、Virtex®-4、Virtex-5 FPGA アーキテクチャ

## 適用可能エレメント

グローバル CONFIG 制約です。特定のインスタンスや信号名には設定されません。

## 適用ルール

CONFIG STEPPING は、デザイン全体に適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF 構文

**CONFIG STEPPING="n";**

*n* はターゲットのステップ レベル (ES、SCD1、1、2、3 など) です。

例 :

**CONFIG STEPPING="1";**

## SUSPEND

FPGA が電力削減モードの SUSPEND に 設定されている場合に、各出力のビヘイビアを定義します。出力ポートまたは双方向ポートに設定可能で、トライステート (3STATE)、High (3STATE\_PULLUP)、または Low (3STATE\_PULLDOWN) に指定できます。あるいは 3STATE\_KEEPER または DRIVE\_LAST\_VALUE と指定して、前回使用した値にすることもできます。デフォルト値は 3STATE です。

## アーキテクチャサポート

この制約は、Spartan®-3A デバイスにのみ適用できます。

## 適用可能エレメント

Spartan-3A で、最上位の出力ポートまたは双方向ポートのみに設定します。

## 適用ルール

SUSPEND 属性は、最上位の出力ポートまたは双方向ポートのみに設定します。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

出力ポートまたは双方向ポートに設定します。

- ・ 属性名 : SUSPEND
- ・ 属性値 : DRIVE\_LAST\_VALUE、3STATE、3STATE\_PULLUP、3STATE\_PULLDOWN、3STATE\_KEEPER

## VHDL 構文

SUSPEND は、アーキテクチャ宣言と最上位 VHDL ファイルの begin 文の間に次のように宣言します。

```
attribute SUSPEND: string;
```

SUSPEND を宣言後、次のように指定します。

```
attribute SUSPEND of {top_level_port_name} : signal is "value";
```

*value* は DRIVE\_LAST\_VALUE、3STATE、3STATE\_PULLUP、3STATE\_PULLDOWN、3STATE\_KEEPER になります。

例 :

```
entity top is  
port (STATUS: out std_logic);  
end top;  
architecture MY_DESIGN of top is  
attribute SUSPEND: string;  
attribute SUSPEND of STATUS: signal is "DRIVE_LAST_VALUE";  
begin
```

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

最上位の Verilog コードの port 宣言の前に、次のように記述します。

```
(* SUSPEND="value" *)
```

*value* は DRIVE\_LAST\_VALUE、3STATE、3STATE\_PULLUP、3STATE\_PULLDOWN、3STATE\_KEEPER になります。

例 :

```
module top ( (* SUSPEND="DRIVE_LAST_VALUE" *) output STATUS );
```

Verilog 構文の詳細については、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

出力ポートまたは双方向ポートに設定します。

```
NET "top_level_port_name" SUSPEND="value";
```

*value* は DRIVE\_LAST\_VALUE、3STATE、3STATE\_PULLUP、3STATE\_PULLDOWN、3STATE\_KEEPER になります。

例 :

```
NET "STATUS" SUSPEND="DRIVE_LAST_VALUE";
```

### PACE からの設定

SUSPEND 属性は PACE で指定できます。[Design Objects] ウィンドウでピンに値を設定します。

## SYSTEM\_JITTER

SYSTEM\_JITTER 制約は、デザインのシステム ジッタを指定します。システム ジッタの値は、一度に切り替わるフリップフロップや I/O の数などの条件によって決まります。この制約は、デザイン内のすべてのクロックにグローバルに適用されます。この制約は、INPUT\_JITTER キーワードを指定した PERIOD 制約と共に使用すると、タイミングレポートでクロック誤差の値、クロック ネットワークのすべてのジッタまたは位相エラーが表示されるようになります。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

グローバルに適用できます。

## 適用ルール

ありません。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : SYSTEM\_JITTER

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute SYSTEM_JITTER: string;
```

VHDL 制約を次のように指定します。

```
attribute SYSTEM_JITTER of {component_name | signal_name|entity_name| label_name}: {component | signal|entity|label} is "value ps";
```

value は数値になります。デフォルトは ps です。

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

### Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(* SYSTEM_JITTER = "value ps" *)
```

*value* は数値になります。デフォルトは ps です。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

### UCF および NCF 構文

UCF 構文は次のとおりです。

```
SYSTEM_JITTER= value ps;
```

*value* は数値になります。デフォルトは ps です。

### XCF 構文

```
MODEL "entity_name " SYSTEM_JITTER = value ps;
```

## TEMPERATURE

TEMPERATURE はタイミング制約で、動作ジャンクション温度を指定します。また、指定した温度に基づいてデバイスの遅延特性を比例配分できます。比例配分は、既存のスピード ファイルの遅延に対して行われ、すべての遅延に対してグローバルに適用されます。

**メモ：** 新しいデバイスでは、タイミング情報 (スピード ファイル) が Production ステータスになるまで温度の比例配分がサポートされない場合があります。

アーキテクチャにはそれぞれ、有効な動作温度範囲があります。入力した温度がこの範囲外の場合、TEMPERATURE 制約は無視され、そのアーキテクチャのワーストケースの値が使用されます。これに関するエラー メッセージは、スタティック タイミングで出力されます。

## アーキテクチャ サポート

次の FPGA は、サポートされていません。

- ・ Spartan®-3A
- ・ Spartan-3AN
- ・ Spartan-3A DSP
- ・ Spartan-3E
- ・ Virtex®-4
- ・ Virtex-5

## 適用可能エレメント

グローバル

## 適用ルール

ネットには設定できません。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## UCF および NCF 構文

**TEMPERATURE=***value* [**C** | **F** | **K**];

*value* は温度を指定する実数です。

C、K、および F は温度の単位です。

- ・ F : 華氏
- ・ K : ケルビン
- ・ C : 摂氏 (デフォルト)

次のように指定すると、スピード ファイル遅延に関連した解析でのジャンクション温度は 25°C になります。

**TEMPERATURE=25 C;**

## Constraints Editor からの設定

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Timing Constraints] の下の [Operating Conditions] をダブルクリックし、ダイアログ ボックスにアクセスします。

## TIG (タイミング無視)

TIG (タイミング無視) は、タイミングおよび合成制約です。この制約を指定すると、TIG を適用したポイント (点) 以降のパスは、インプリメンテーション中、タイミング解析において存在しないものとして処理されます。

TIG は、特定のタイムスペックに関連して設定できます。

TIG は、次のいずれかの値に設定します。

- ・ 値なし (すべてのパスに適用されるグローバル TIG)
- ・ 1 つの TSid からブロック
- ・ カンマで区切った TSid からブロックのリスト (次の例を参照)

**NET "RESET" TIG=TS\_fast, TS\_even\_faster;**

TIG は、XST で完全にサポートされています。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

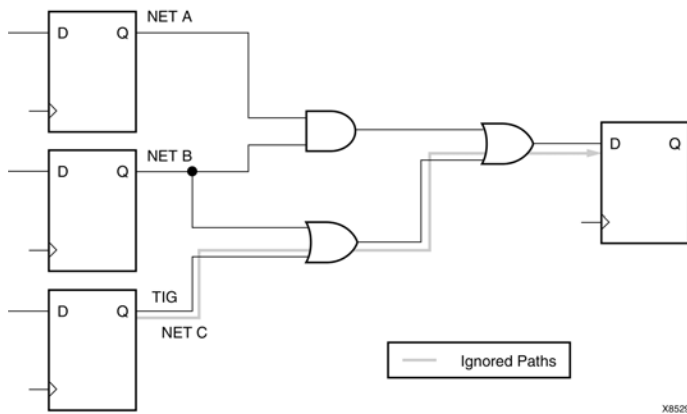
## 適用可能エレメント

- ・ ネット
- ・ ピン
- ・ インスタンス

## 適用ルール

TIG をネット、プリミティブ ピン、またはマクロ ピンに設定した場合、インプリメンテーション中、タイミング解析においては、制約が適用されるポイント以降のパスはすべて存在しないものとして処理されます。次の図では、ネット C は無視されますが、2 つの OR ゲートを通過するネット B 以前のパスは無視されません。

## TIG の例



次の制約をネットに設定すると、タイミング解析において、タイムスペック TS43 でそのネットを通るパスが無視されます。

回路図の構文

**TIG = TS43**

UCF 構文

**NET "net\_name" TIG = TS43;**

組み合わせループがある場合、パスの解析はできません。そのため、タイミング解析では特定の接続を無視し、組み合わせループを切断します。TIG 制約を使用すると、特定ネットまたはロード ピンを無視するようにタイミング ツールに命令でき、ループの接続を制御できます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

**メモ:** TIG 制約を使用しても、XST レポートの一番下のタイミング レポートには影響ありません。TIG 制約が適用されるのは、Timing Analyzer でレポートされるタイミングのみです。

## 回路図

- ・ ネットまたはピンに設定します。
- ・ 属性名 : TIG
- ・ 属性値 : *value*

## UCF および NCF 構文

UCF 構文は次のとおりです。

**NET "net\_name" TIG;**

**PIN "f\_inst.RST" TIG=TS\_1;**

**INST "instance\_name" TIG=TS\_2;**

**TIG=TS identifier1 . . . TS identifiern**

*identifier* は無視するタイムスペックを表します。

インスタンスに設定すると、インスタンスの出力ピンに適用されます。ネットに設定するとネットの駆動ピンに、ピンに設定するとピンに適用されます。



次の文は、タイムスペック TS\_fast および TS\_even\_faster が、ネット RESET 以降のすべてのパスで無視されるよう指定します。

```
NET "RESET" TIG=TS_fast, TS_even_faster;
```

## XCF 構文

UCF と同じ構文を使用します。

TIG は、XST で完全にサポートされています。TIG は、CORE Generator ファイル (EDIF、NGC) に含まれるネットに適用することも可能です。

## Constraints Editor の構文

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Timing Constraints] の下の [Exceptions] で [Nets] をダブルクリックし、ダイアログ ボックスにアクセスします。

## PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## PCF 構文

*item* **TIG**;

*item* **TIG** =;

*item* **TIG** = **TSidentifier** ;

*item* は次のいずれかになります。

- ・ **PIN** *name*
- ・ **PATH** *name*
- ・ *path specification*
- ・ **NET** *name*
- ・ **TIMEGRP** *name*
- ・ **BEL** *name*
- ・ **COMP** *name*
- ・ **MACRO** *name*

## TIMEGRP

TIMEGRP は、タイミング解析用にデザイン エレメントをグループ化するための制約です。TNM 識別子を使用してデザイン エレメントのグループを定義するだけでなく、グループをほかのグループと関連させて定義できます。TIMEGRP 制約を使用すると、既存グループを組み合わせることでグループを作成できます。

TIMEGRP 制約は、制約ファイル (UCF または NCF) で設定できます。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイスに適用できます。

## 適用可能エレメント

- ・ デザイン エレメント
- ・ ネット

## 適用ルール

グループ内のすべてのエレメントまたはネットに適用されます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 複数のグループを 1 つのグループにまとめる

複数のグループをまとめて 1 つのグループを定義します。次に、2 つのグループを 1 つにまとめる構文例を示します。

UCF 構文 1

```
TIMEGRP "big_group"="small_group" "medium_group";
```

*small\_group* および *medium\_group* は、TNM または TIMEGRP で定義された既存のグループです。

UCF 構文 2

次のように定義が循環していると、NGCBuild でデザインを実行する際にエラーが発生します。

```
TIMEGRP "many_ffs"="ffs1" "ffs2";
```

```
TIMEGRP "ffs1"="many_ffs" "ffs3";
```

### 除外によって新しいグループを作成する方法

あるグループから別のグループに属するエレメントを除外することにより、グループを定義できます。次に、その構文例を示します。

UCF 構文 1

```
TIMEGRP "group1"="group2" EXCEPT "group3";
```

この場合、それぞれ次を表します。

- ・ *group1* は、定義されたグループを表しています。これには、*group2* のエレメントすべてが含まれますが、*group3* にも属するエレメントは除外されています。
- ・ *group2* および *group3* は、次のいずれかです。
  - 有効な TNM
  - 定義済みのグループ
  - TIMEGRP 属性

UCF 構文 2

次に示すように、新しいグループを作成する際に、含めるグループまたは除外するグループを複数指定できます。

```
TIMEGRP "group1"=" "group2" "group3" EXCEPT "group4" "group5";
```

この例では、*group1* は *group2* および *group3* のエレメントを含みますが、*group4* または *group5* に属するエレメントは除外されます。キーワード EXCEPT の前のグループはすべて含まれ、キーワードの後のグループは除外されます。

### クロック エッジによるフリップフロップのサブグループの定義

キーワード RISING および FALLING を使用して、立ち上がりエッジおよび立ち下がりエッジでトリガされるフリップフロップをグループ化し、サブグループを作成できます。

UCF 構文 1

```
TIMEGRP "group1"=RISING FFS;  
TIMEGRP "group2"=RISING "ffs_group";  
TIMEGRP "group3"=FALLING FFS;  
TIMEGRP "group4"=FALLING "ffs_group";
```

この場合、それぞれ次を表します。

*group1* ~ *group4* は、新しく定義されるグループです。*ffs\_group* は、フリップフロップのみを含むグループである必要があります。

EXCEPT、RISING、FALLING などのキーワードには、大文字/小文字のいずれを使用してもかまいません。ただし、大文字と小文字を組み合わせて入力することはできません。

UCF 構文 2

次に、クロックの立ち下がりエッジでオンになるフリップフロップのグループを定義します。

```
TIMEGRP "falling_ffs"=FALLING FFS;
```

### ゲートによるラッチのサブグループの定義

LATCHES のグループは、どのように定義されていても、透過 High および透過 Low のサブグループに分類できます。これにはキーワード TRANSHI および TRANSLO を使用します。これらのキーワードは、フリップフロップ グループのキーワード RISING および FALLING と同様、TIMEGRP 文で使用します。

UCF 構文

```
TIMEGRP "lowgroup"=TRANSLO "latchgroup";  
TIMEGRP "highgroup"=TRANSHI "latchgroup";
```

### 文字列の指定によるグループの作成

特定の文字列に一致するネット名を持つシンボルをグループ化できます。文字列の指定には、ワイルドカード文字を使用します。通常、この方法は、ネット名を指定した回路図デザインで使用します。合成には、INST/TNM 構文を使用します。詳細は、「TNM」を参照してください。

ワイルドカードを使用したネット名の指定

アスタリスク(\*)やクエスチョン マーク(?)などのワイルドカード文字を使用すると、出力ネット名が指定の文字列に一致するシンボルのグループを定義できます。アスタリスク(\*): 任意の数の文字列 疑問符(?) は任意の 1 文字を表します。

たとえば、DATA\* は DATA、DATA1、DATA22、DATABASE など「DATA」で始まるネット名を表します。文字列 NUMBER? は、「NUMBER」の後に 1 文字が付くネット名を表します。NUMBER1、NUMBERS などがその例です。NUMBER や NUMBER12 は該当しません。

ワイルドカード文字は、複数使用できます。たとえば、\*AT? は、BAT1、CAT2、THAT5 など、「AT」の前に任意の数の文字、後に 1 文字が付く文字列を指定します。\*AT\* と指定すると、BAT11、CAT26、THAT50 などが該当します。

## UCF 構文 1

次に、文字列の指定によってグループを作成するための構文を示します。

```
TIMEGRP "group_name"=predefined_group("pattern");
```

この場合、それぞれ次を表します。

- ・ *predefined\_group* は、定義済みグループ FFS、LATCHES、PADS、RAMS、CPUS、HSIOS、DSPS、BRAMS\_PORTA、BRAMS\_PORTB、および MULTS のいずれかになります。これらのグループの定義の詳細は、「TNM\_NET」の「UCF および NCF 構文」を参照してください。
- ・ *pattern* は、ワイルドカード文字を使用した文字列です。

ネット名を指定する場合、PAR がフラット化されたデザインでネットを検索できるように、完全な階層パス名を使用する必要があります。

FFS、LATCHES、PADS、RAMS、BRAMS\_PORTA、BRAMS\_PORTB、CPUS、DSPS、HSIOS、および MULTS の場合は出力ネット名を指定し、パッドの場合は外部ネット名を指定します。

## UCF 構文 2

次の例では、名前が \$I13/FRED で始まるネットを持つフリップフロップを含むグループを作成します。

```
TIMEGRP "group1"=FFS("$I13/FRED*");
```

## UCF 構文 3

次の例では、出力ネット名が特定の文字列に一致するフリップフロップを除くグループを作成します。

```
TIMEGRP "this_group"=FFS EXCEPT FFS("a*");
```

この場合、それぞれ次を表します。

*this\_group* には、名前が a で始まる出力ネットを持つフリップフロップを除く、すべてのフリップフロップが含まれます。

## UCF 構文 4

次の例では、some\_latches という名前のグループを定義します。

```
TIMEGRP "some_latches"=latches("$I13/xyz*");
```

この場合、それぞれ次を表します。

グループ *some\_latches* には、名前が「\$I13/xyz」で始まる出力ネットを持つ入力ラッチがすべて含まれます。

## その他の文字列指定

文字列の指定は、タイミング グループ作成時だけではなく、定義済みグループを指定する場所であればどこでも使用できます。次の構文では、タイムスペックで文字列を指定する方法を示します。

## UCF 構文 1

```
TIMESPEC "TSidentifier"=FROM predefined_group("pattern") TO predefined_group("pattern") value;
```

文字列を 1 つ指定する代わりに、コロンで区切った文字列のリストを指定できます。

## UCF 構文 2

```
TIMEGRP "some_ffs"=FFS("a*:b?:c*d");
```

この場合、それぞれ次を表します。

グループ *some\_ffs* には、名前が次のいずれかのルールに一致する出力ネットを持つフリップフロップが含まれます。

- 最初の文字が「a」
- 最初の文字が「b」で 2 文字の名前
- 最初の文字が「c」で、最後の文字が「d」

タイムグループを使用したエリア グループの定義

詳細は、「[AREA\\_GROUP](#)」の「タイミング グループによる定義」を参照してください。

## UCF 構文

次は TIMEGRP の UCF 構文です。

例 1

```
TIMEGRP "newgroup" = "existing_grp1" "existing_grp2" ["existing_grp3" ...];
```

この場合、それぞれ次を表します。

*newgroup* は、新しく作成するグループです。このグループは、次のいずれかです。

- TNM で作成された既存のグループ
- 定義済みグループ
- ほかの TIMEGRP 属性

例 2

```
TIMEGRP "GROUP1" = "gr2" "GROUP3";
```

```
TIMEGRP "GROUP3" = FFS except "grp5";
```

## XCF 構文

XST での TIMEGRP の使用には、次のような制限があります。

- ・ 除外によるグループの作成はサポートされていません。
- ・ 文字列の一致を使用して別のユーザー グループを基にグループを定義する場合

TIMEGRP TG1 = FFS (machine\*); # Supported

TIMEGRP TG2 = TG1 (machine\_clk1\*); # Not supported

## Constraints Editor の構文

ISE® の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Group Constraints] の下の [Upstream Elements by Nets] または [By Clock Edge] をダブルクリックし、ダイアログ ボックスにアクセスします。

## PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## PCF 構文

**TIMEGRP** *name*;

**TIMEGRP** *name* = *list of elements*;

## TIMESPEC

TIMESPEC は、基本的なタイミング制約で、タイムスペックのプレースホルダの役割を果たします。タイムスペックは、TS 属性を定義することです。TS 属性は、文字列「TS」で開始し、その後に英数字またはアンダースコア ( ) からなる固有の識別名を付けます。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイ스에適用できます。

## 適用可能エレメント

TS 識別子

## 適用ルール

ありません。

## 構文

**Value** パラメータ

*value* は、属性の最大遅延を定義します。デフォルトの単位はナノ秒 (ns) ですが、ピコ秒やメガヘルツなど、ほかの単位も使用できます。

### キーワード

このマニュアルでは FROM、TO、TS などのキーワードは大文字で表記されていますが、TIMESPEC には、大文字または小文字のどちらでも使用できます。ただし、キーワードの文字は大文字または小文字で統一する必要があります。次の例は、キーワードとして使用できます。

- ・ FROM
- ・ PERIOD
- ・ TO
- ・ from
- ・ to

次の例は、キーワードとして使用できません。

- ・ From
- ・ To
- ・ fRoM
- ・ tO

## TSidentifier 名

TSidentifier 名がプロパティ値で参照されている場合は、大文字で入力する必要があります。たとえば次の例で示すように、2 番目の制約名 TSID1 は、1 番目の制約名 TSID1 と一致させるため大文字で入力する必要があります。

```
TIMESPEC "TSID1" = FROM "gr1" TO "gr2" 50;
```

```
TIMESPEC "TSMIN" = FROM "here" TO "there" TSID1 /2;
```

## 区切り文字

タイムスペックでは、区切り文字として、スペースの代わりにコロンを使用できます。

## 構文

TIMESPEC で、特定の終了地点間のタイムスペックを指定するには、次の UCF 構文を使用します。

```
TIMESPEC "TSidentifier"=FROM "source_group" TO "dest_group" value units;
```

```
TIMESPEC "TSidentifier"=FROM "source_group" value units;
```

```
TIMESPEC "TSidentifier"=TO "dest_group" value units;
```

2 番目と 3 番目の構文のように、FROM または TO を指定しない場合は、すべての点であることを示します。

**メモ：** FROM または TO を指定せずにすべての点を示すことはできますが、THRU を指定せずにすべての点を示すことはできません。

FROM TO 文は、TIMESPEC プリミティブ内に存在する TS 属性です。パラメータ *source\_group* および *dest\_group* は、次のいずれかです。

- ・ 定義済みグループ
- ・ 作成済みの TNM 識別子
- ・ TIMEGRP シンボルで定義されたグループ
- ・ TPSYNC グループ

定義済みグループには、FFS、PADS、LATCHES、RAMS、BRAMS\_PORTA、BRAMS\_PORTB、CPUS、DSPS、HSIOS、または MULTS が含まれます。これらのグループの定義については TNM\_NET の「UCF および NCF 構文」を参照してください。詳細は、「制約のタイプ」の章の「[グループ制約](#)」を参照してください。

このマニュアルでは FROM、TO、TS などのキーワードは大文字で表記されていますが、TIMESPEC には、大文字または小文字のどちらでも使用できます。ただし、大文字と小文字を組み合わせることはできません。ただし、大文字と小文字を組み合わせることはできません。

*value* は、属性の最大遅延を定義します。デフォルトの単位はナノ秒 (ns) ですが、ピコ秒やメガヘルツなど、ほかの単位も使用できます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### FROM TO を使用した TIMESPEC の構文例

UCF および NCF 構文：

```
TIMESPEC "TS_master"=PERIOD "master_clk" 50 HIGH 30;
```

```
TIMESPEC "TS_THIS"=FROM FFS TO RAMS 35;
```

```
TIMESPEC "TS_THAT"=FROM PADS TO LATCHES 35;
```

## UCF 構文

TS 属性は、デザインのパスに対して許容される遅延を定義します。次に、TS 属性の基本的な構文を示します。

```
TIMESPEC "TSidentifier"=PERIOD "timegroup_name" value [units];
```

この場合、次を表します。

- ・ TSidentifier は、TS 属性の名前です。
- ・ value は、数値です。
- ・ units は、ms、micro、ps、ns などの時間の単位です。

```
TIMESPEC "TSidentifier"=PERIOD "timegroup_name" "TSidentifier" [* or /] factor PHASE [+|-] phase_value [units];
```

## Constraints Editor からの設定

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。制約の設定方法については、Constraints Editor ヘルプを参照してください。

## PlanAhead からの設定

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## TNM (タイミング名)

TNM はグループを構成するエレメントを指定する基本的なグループ制約で、タイミング仕様を設定できます。

TNM は、特定の FFS、RAMS、LATCHES、PADS、BRAMS\_PORTA、BRAMS\_PORTB、CPUS、DSPS、HSIOS、MULTS をグループのエレメントとして指定し、タイミング仕様の適用を簡略化できます。

この制約は、キーワード RISING および FALLING と一緒に使用することもできます。

**メモ：**パーティションに関連する PAD 名に基づいた TNM は、サポートされません。パーティション内のネット名に基づいた TNM は、サポートされます。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイ스에適用できます。

## 適用可能エレメント

ネット、エレメントピン、プリミティブ、またはマクロに設定できます。

TNM 制約は、UCF ファイル内で、パッド コンポーネントに接続されているネットにも設定できます。ネットに設定した制約は、NGCBuild により NGD ファイル内のパッド インスタンスに挿入され、マップで処理されます。これには、次の UCF 構文を使用します。

```
NET "net_name" TNM="property_value";
```



## 適用ルール

ネットまたは信号に TNM を設定すると、そのネットにより駆動されるすべての同期エレメントに伝搬されます。特別な伝搬はありません。

デザイン エレメントに設定すると、そのデザイン エレメントの階層にあるすべての適用可能エレメントに適用されます。

TNM を使用する場合、次のルールが適用されます。

- ・ TNM をパッド ネットに適用すると、IBUF を介して順方向には伝搬されず、外部パッドに適用されます。これには、IFD の D 入力に接続されているネットも含まれます。TNM を入力パッド ネットから順方向に伝搬させる方法については、「[TNM\\_NET](#)」を参照してください。
- ・ IBUF インスタンスには設定できません。
- ・ IBUF の出力ピンに設定すると、その次の適切なエレメントに適用されます。
- ・ IBUF エレメントに適用した場合、TNM はそのエレメントに設定されます。
- ・ クロック パッド ネットに設定すると、クロック バッファ以降には適用されません。
- ・ マクロに設定すると、マクロ内のすべてのエレメントにそのタイミング名が適用されます。

### ネットに設定する場合

TNM 制約は、デザイン内のどのネットにも設定でき、指定されたネット以降のパスにより信号が供給されるすべての有効なエレメントに TNM 値が付加されます。制約は、FFS、RAMS、LATCHES、PADS、BRAMS\_PORTA、BRAMS\_PORTB、CPUS、DSPS、HSIOS、MULTS まで適用されます。TNM を入力パッド ネットに設定した場合、IBUF を介しては伝搬されません。

### マクロまたはプリミティブ ピンに設定する場合

プリミティブ ピンに TNM を設定する場合

デザイン入力パッケージでプリミティブ ピンに制約を設定できる場合、デザイン内のコンポーネントピンに TNM 制約を設定できます。指定されたピン以降のパスにより信号が供給されるすべての有効なエレメントに、TNM 値が付加されます。制約は、FFS、RAMS、LATCHES、PADS、BRAMS\_PORTA、BRAMS\_PORTB、CPUS、DSPS、HSIOS、MULTS まで適用されます。

次に、UCF ファイルの構文を示します。

```
PIN "pin_name" TNM="FLOPS";
```

### プリミティブ シンボルに設定する場合

各インスタンスに制約を設定すると、ロジック プリミティブをグループ化できます。

TNM が設定されたフリップフロップからグループ「FLOPS」が作成されます。設定されていないフリップフロップは、このグループには含まれません。UCF の構文例を参照してください。

TNM は、各シンボル、ドライバ ピン、またはマクロドライバ ピンに 1 つずつしか設定できません。

### UCF 構文

```
INST "instance_name" TNM=FLOPS;
```

### ネット/ピンに設定してフリップフロップ/ラッチをグループ化する場合

クロック ネットやイネーブル ネットなど共通の入力ネットに TNM を設定すると、フリップフロップ、ラッチを簡単にグループ化できます。TNM をネットまたはドライバ ピンに設定した場合、TNM はそのネットまたはピン以降のパス上にあるすべてのフリップフロップおよび入力ラッチに適用されます。つまり、TNM 制約は、パス上にあるゲートまたはバッファを介して、フリップフロップまたは入力ラッチに到達するまで順方向に伝搬され、そのフリップフロップまたはラッチが、指定された TNM グループに追加されます。

ネットまたはピンに設定する TNM パラメータには、修飾子を含めることができます。たとえば、UCF ファイルは次のように記述します。

```
{NET|PIN} "net_or_pin_name" TNM=FFS data;
{NET|PIN} "net_or_pin_name" TNM=RAMS fifo;
{NET|PIN} "net_or_pin_name" TNM=RAMS capture;
```

修飾子が付いた TNM は、最初のメモリ エLEMENT (FFS、RAMS、LATCHES、PADS、BRAMS\_PORTA、BRAMS\_PORTB、CPUS、DSPS、HSIOS、MULTS) まで伝搬されます。記憶エレメントの種類が修飾子と一致すると、そのエレメントに TNM 値が適用されます。一致するしないにかかわらず、TNM は記憶エレメントを介して伝搬されることはありません。

ネットまたはピンに設定された TNM パラメータは、メモリ エLEMENT (FFS、RAMS、LATCHES、PADS、BRAMS\_PORTA、BRAMS\_PORTB、CPUS、DSPS、HSIOS、MULTS) 以降には適用されません。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## UCF および NCF 構文

```
{NET|INST|PIN} "net_or_pin_or_inst_name" TNM= [predefined_group] identifier;
```

この場合、それぞれ次を表します。

- ・ *predefined\_group* には、次のエレメントを指定できます。
  - FFS、RAMS、LATCHES、PADS、BRAMS\_PORTA、BRAMS\_PORTB、CPUS、DSPS、HSIOS、MULTS キーワードを使用した、定義済みグループのすべてのエレメント
    - ◆ FFS : CLB と IOB のすべてのフリップフロップを表します。ファンクション ジェネレータから作成されたフリップフロップを除きます。
    - ◆ RAMS : RAMS を含むアーキテクチャのすべての RAM を表します。LUT RAMS およびブロック RAMS を含みます。
    - ◆ PADS : すべての I/O パッドを表します。
    - ◆ LATCHES : CLB または IOB のすべてのラッチを表します。ファンクション ジェネレータから作成されたラッチは除きます。
    - ◆ MULTS : Spartan®-3、Spartan-3A、Spartan-3E のレジスタ付き乗算器のグループです。
  - *predefined\_group* のエレメントのサブセットを指定するには、次の構文を使用します。

```
predefined_group (name_qualifier1... name_qualifiern)
```

この場合、それぞれ次を表します。

*name\_qualifiern* は、英数字、アンダースコアを自由に組み合わせて指定できます。*name\_qualifier* のタイプ (ネットまたはインスタンス) は、TNM が配置されているエレメントのタイプによって決まります。TNM が NET に配置されている場合はネット名で、インスタンス (INST) に配置されている場合はインスタンス名です。

例 :

```
NET clk TNM = FFS (my_flop) Grp1;
INST clk TNM = FFS (my_macro) Grp2;
```

- ・ *identifier* は、英数字、アンダースコアを自由に組み合わせて指定できます。

*identifier* には、FFS、RAMS、LATCHES、PADS、CPUS、HSIOS、MULTS、RISING、FALLING、TRANSHI、TRANSLO、EXCEPT という予約語は使用できません。

また、次にリストされている予約語も *identifier* には使用できません。

### 予約語 (制約)

ADD	ALU	ASSIGN
BEL	BLKNM	CAP
CLKDV_DIVIDE	CLBNM	CMOS
CYMODE	DECODE	DEF
DIVIDE1_BY	DIVIDE2_BY	DOUBLE
DRIVE	DUTY_CYCLE_CORRECTION	FAST
FBKINV	FILE	F_SET
HBLKNM	HU_SET	H_SET
INIT	INIT OX	INTERNAL
IOB	IOSTANDARD	LIBVER
LOC	LOWPWR	MAP
MEDFAST	MEDSLOW	MINIM
NODELAY	OPT	OSC
RES	RLOC	RLOC_ORIGIN
RLOC_RANGE	SCHNM	SLOW
STARTUP_WAIT	SYSTEM	TNM
TRIM	TS	TTL
TYPE	USE_RLOC	U_SET

デザインのパフォーマンス要件を記述するのに必要なだけ、終端地点のグループを指定できます。ただし、指定プロセスを単純化し、配置配線時間を削減するため、グループの数を最小限に抑えてください。

### XCF 構文

詳細は、「UCF および NCF 構文」を参照してください。

### Constraints Editor の構文

ISE® の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リストボックスの [Group Constraints] の下の [By Instance/Hierarchy] または [By DCM Outputs] をダブルクリックし、ダイアログ ボックスにアクセスします。

### PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## TNM\_NET

TNM\_NET はグループを構成するエレメントを指定する基本的なグループ制約で、タイミング仕様の設定に使用できます。TNM\_NET 制約は、入力パッド ネットに設定する場合を除き、TNM と基本的に同等です。

DLL/DCM/PLL/MMCM に、PERIOD 制約を使用した TNM\_NET を設定する場合は、特別なルールが適用されます。詳細は、「[PERIOD](#)」の「[CLKDLL、DCM、PLL、および MMCM での PERIOD 指定](#)」を参照してください。

TNM\_NET は通常、特定ネットを指定するため HDL デザインで使用するプロパティです。TNM\_NET で指定されたダウンストリームの同期エレメントおよびパッドは、すべてグループと見なされます。

TNM\_NET は、特定の同期エレメント、パッド、ラッチをグループ化することにより、タイミング仕様の適用を簡略化するために使用できます。TNM 制約とは異なり、NGCBuild で、制約が設定されたネットから入力パッドに TNM\_NET が伝搬されることはありません。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

ネット

## 適用ルール

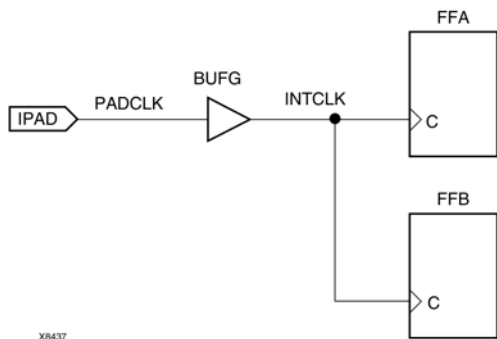
TNM\_NET を使用する場合、次のルールが適用されます。

- ・ パッド ネットに設定すると、IBUF、OBUF、または組み合わせロジックを介して同期ロジックまたはパッドに伝搬されます。
- ・ クロック パッド ネットに設定すると、クロック バッファを介して伝搬されます。
- ・ Virtex®-4 および Virtex-5 の DLL、DCM、および PLL で、PERIOD 制約と共に TNM\_NET を使用する場合は、特別なルールが適用されます。

TNM 制約では適切に記述できないタイプのネットを定義する場合は、TNM\_NET を使用します。

たとえば、次のようなデザインがあるとします。<:fc 2><:iaf 1><:/fc>

## IPAD に設定した TNM



このデザインで、IPAD シンボルに TNM を設定すると、グループには PAD シンボルのみが含まれます。たとえば、次の UCF 構文は、IPAD シンボルのみを含むタイムグループを作成します。

**NET "PADCLK" TNM= "PADGRP";** (UCF ファイル例)

一方、ネット PADCLK のタイムグループを定義するために TNM を使用すると、空のタイムグループが作成されます。

**NET "PADCLK" TNM=FFS "FFGRP";**(UCF ファイル例)

パッドに適用されるプロパティはすべて、ネットから PAD シンボルに転送されます。TNM はネットから PAD シンボルに転送されるので、修飾子「FFS」は PAD シンボルと一致しません。

回路図デザインで TNM を使用する際にこの問題を回避するには、INTCLK ネットのタイムグループを作成します。

**NET "INTCLK" TNM=FFS FFGRP;**(UCF ファイル例)

HDL デザインの場合は、意味を持つネット名のみがパッドに直接接続されます。この場合は、TNM\_NET を使用して FFGRP タイムグループを作成します。

**NET PADCLK TNM\_NET=FFS FFGRP;**(UCF ファイル例)

TNM の場合とは異なり、TNM\_NET 制約はネットから IPAD には転送されません。

TNM\_NET は、入力ネットリスト (EDIF または NGC) 内のネットに設定されたプロパティとして、NCF または UCF ファイル内で使用できます。TNM\_NET は、PCF ファイルではサポートされていません。

TNM\_NET は、ネットまたはインスタンスに使用できます。ピンやシンボルなどほかのオブジェクトに使用した場合は、警告が表示され、TNM\_NET 定義は無視されます。

## 適用ルール

デザイン エレメントには設定できません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ ネットに設定します。
- ・ 属性名 : TNM\_NET
- ・ 属性値 : *identifier*

詳細は、「UCF および NCF 構文」を参照してください。

## UCF および NCF 構文

```
{NET|INST} "net_name" TNM_NET=[predefined_group:]identifier;
```

- ・ *predefined\_group* には、次のエレメントを指定できます。
  - FFS、RAMS、BRAM\_PORTA、BRAM\_PORTB、PADS、LATCHES、MULTS、HSIOS、CPUS、DSPS キーワードを使用した、定義済みグループのすべてのエレメント
    - ◆ FFS : CLB と IOB のすべてのフリップフロップを表します。ファンクション ジェネレータから作成されたフリップフロップを除きます。
    - ◆ RAMS : RAMS を含むアーキテクチャのすべての RAM を表します。LUT RAMS およびブロック RAMS を含みます。
    - ◆ PADS : すべての I/O パッドを表します。
    - ◆ MULTS : Spartan®-3、Spartan-3A、Spartan-3E のレジスタ付き乗算器のグループです。
    - ◆ DSPS : Virtex-4 DSP48 のような DSP エレメントをグループにするために使用されます。
    - ◆ LATCHES : CLB または IOB のすべてのラッチを表します。ファンクション ジェネレータから作成されたラッチは除きます。

- *predefined\_group* のエレメントのサブセットを指定するには、次の構文を使用します。

```
predefined_group (name_qualifier1... name_qualifiern)
```

この場合、それぞれ次を表します。

*name\_qualifiern* は、英数字、アンダースコアを自由に組み合わせて指定できます。*name\_qualifier* のタイプ (ネットまたはインスタンス) は、TNM\_NET が配置されているエレメントのタイプによって決まります。TNM\_NET が NET に配置されている場合はネット名で、インスタンス (INST) に配置されている場合はインスタンス名です。

例 :

```
NET clk TNM_NET = FFS (my_flop) Grp1;
```

```
INST clk TNM_NET = FFS (my_macro) Grp2;
```

- ・ *identifier* は、英数字、アンダースコアを自由に組み合わせて指定できます。

*identifier* には、FFS、RAMS、LATCHES、PADS、CPUS、HSIOS、MULTS、RISING、FALLING、TRANSHI、TRANSLO、EXCEPT という予約語は使用できません。

また、「TNM」に表示されている予約語も *identifier* には使用できません。

次の文は、PADCLK ネット以降のパス上にあるすべてのフリップフロップをタイミング グループ GRP1 として指定します。

```
NET "PADCLK" TNM_NET=FFS "GRP1";
```

## XCF 構文

XST で TIME\_NET を使用する場合、あらかじめ定義されているグループに対する 1 つのパターンしかサポートされていません。

次のコマンド ライン構文がサポートされます。

```
NET "PADCLK" TNM_NET=FFS "GRP1";
```

次のコマンド ライン構文はサポートされません。

```
NET "PADCLK" TNM_NET = FFS(machine/*:xcounter/*) TG1;
```

## Constraints Editor の構文

ISE® の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Group Constraints] の下の [Downstream Elements by Nets] をダブルクリックし、ダイアログ ボックスにアクセスします。

## PlanAhead の構文

PlanAhead™ を使用した制約の作成方法は、『PlanAhead ユーザー ガイド』を参照してください。

また、このマニュアルの「PlanAhead」でも次を説明しています。

- ・ 配置制約の定義
- ・ 配置の割り当て
- ・ I/O ピン コンフィギュレーションの定義
- ・ フロアプランおよび配置制約

## TPSYNC

TPSYNC はグループ制約で、特定のポイントまたはポイントの集合に識別名を指定し、タイミング仕様で 사용할 ことができます。同じ識別名を複数のポイントに指定すると、それらのポイントはタイミング解析でグループとして扱われます。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

- ・ ネット
- ・ インスタンス
- ・ ピン

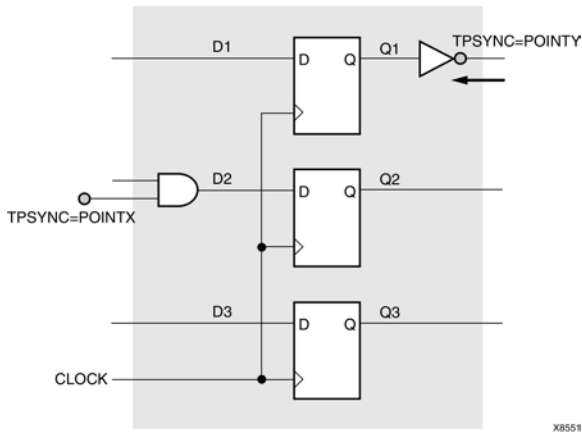
## 適用ルール

同期エレメントまたは I/O パッド以外のポイントに関してデザインのタイミングを指定する場合、TPSYNC タイミング ポイントをネット、マクロ ピン、プリミティブの入出力ピン、インスタンスのいずれに設定するかで、次のような規則が適用されます。

- ・ ネット：ネットのソースがタイミング仕様のソースまたはデスティネーションとして指定されます。
- ・ マクロ ピン：制約が設定されたピンを駆動するマクロ内のすべてのソースが、タイミング仕様のソースまたはデスティネーションとして指定されます。マクロ ピンが入力ピンの場合 (ピンのソースがマクロ内にない場合) は、マクロ内のすべてのロード ピンが同期点として指定されます。

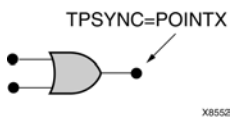
次の図では、POINTY がインバータに設定されます。

## マクロ ピンに設定した TPSYNC



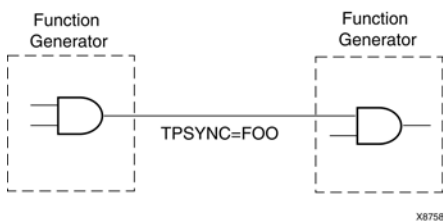
- ・ プリミティブの出力ピン: プリミティブの出力がタイミング仕様のソースまたはデスティネーションとして指定されます。
- ・ プリミティブの入力ピン: プリミティブの入力がタイミング仕様のソースまたはデスティネーションとして指定されます。
- ・ インスタンス: そのエレメントの出力がタイミング仕様のソースまたはデスティネーションとして指定されます。
- ・ プリミティブ シンボル: 設定されたエレメントの出力が、タイミング仕様のソースまたはデスティネーションとして指定されます。次の図を参照してください。

## プリミティブシンボルに指定した場合



デザインでの同期点の定義に TPSYNC タイミング ポイントを使用すると、指定したポイントをファンクション ジェネレータに結合できません。次のようなデザインがあるとして。

## 2 つのゲートに指定した場合



この例では、TPSYNC が定義されているため、2 つのゲートは 1 つのファンクション ジェネレータにインプリメントされません。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。



## 回路図

- ・ ネット、インスタンス、またはピンに設定します。
- ・ 属性名 : TPSYNC
- ・ 属性値 : *identifier*

*identifier* グループと同様にタイミング仕様で使用する名前です。

## UCF および NCF 構文

```
NET "net_name" TPSYNC=identifier;
```

```
INST "instance_name" TPSYNC=identifier;
```

```
PIN "pin_name" TPSYNC=identifier;
```

*identifier* グループと同様にタイミング仕様で使用する名前です。

指定されたすべてのポイントは、TPSYNC 識別名が使用される仕様のソースまたはデスティネーションのいずれか、あるいは両方に使用されます。

識別名には、ほかの TNM または TNM\_NET の識別名とは異なる名前を指定する必要があります。

次の文は、ネット `logic_latch` のタイミング仕様のソースまたはデスティネーションとしてラッチを指定します。

```
NET "logic_latch" TPSYNC=latch;
```

## Constraints Editor からの設定

ISE® の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リストボックスの [Group Constraints] の下の [By Combinatorial Pinsto] をダブルクリックし、ダイアログ ボックスにアクセスします。

## TPTHRU

TPTHRU はグループ制約で、特定のポイントまたはポイントの集合に識別名を指定し、タイミング仕様で使用するようにします。同じ識別名を複数のポイントに指定すると、それらのポイントはタイミング解析でグループとして扱われます。詳細は、「TIMESPEC」を参照してください。

TPTHRU 制約は、仕様を適用するパス上に中間点を定義する必要がある場合に使用します。詳細は、「TSidentifier」を参照してください。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

- ・ ネット
- ・ ピン
- ・ インスタンス

## 適用ルール

ありません。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ ネット、インスタンス、またはピンに設定します。
  - ・ 属性名 TPTHURU
  - ・ 属性値 : *identifier*
- 詳細は、「UCF および NCF 構文」を参照してください。

## UCF および NCF 構文

UCF の構文は次のとおりです。

```
NET "net_name" TPTHURU=identifier;  
INST "instance_name" TPTHURU=identifier;  
PIN "instance_name.pin_name" TPTHURU="thru_group_name";
```

この場合、次を表します。

*identifier* は、デザイン内のタイミング パスを指定するために、タイミング仕様で使用する名前です。

識別名には、ほかの TNM 制約で使用されている識別名とは異なる名前にする必要があります。

## FROM-TO 制約との併用

仕様を適用するパス上に中間点を定義すると便利な場合があります。次のような構文で、最大許容遅延を定義します。

## TIMESPEC を併用した UCF 構文

```
TIMESPEC "TSidentifier"=FROM "source_group" THRU "thru_point" [THRU "thru_point"] TO "dest_group"  
allowable_delay [units];
```

```
TIMESPEC "TSidentifier"=FROM "source_group" THRU "thru_point" [THRU "thru_point"] allowable_delay  
[units];
```

この場合、次を表します。

- ・ *identifier* は、英数字 (A ~ Z, a ~ z, 0 ~ 9) およびアンダースコアから構成される ASCII 形式の文字列です。
- ・ *source\_group* および *dest\_group* は、ユーザー定義のグループ、定義済みのグループ、または TPSYNC です。
- ・ *thru\_point* は、パスを指定するために使用する中間点で、TPTHRU 制約で定義します。
- ・ *allowable\_delay* は、タイミング要件です。
- ・ *units* は、オプションで許容遅延の単位を指定します。デフォルトの単位はナノ秒ですが、ps、ns、micro、ms、GHz、MHz、または kHz のいずれかを指定できます。

次の例は、回路図で TPTHRU 制約と THRU 制約を併用する方法を示します。UCF の構文は次のとおりです。

```
INST "FLOPA" TNM="A";  
INST "FLOPB" TNM="B";  
NET "MYNET" TPTHRU="ABC";  
TIMESPEC "TSpath1"=FROM "A" THRU "ABC" TO "B" 30;
```

次の文は、ネット *on\_the\_way* を、タイミング仕様 *here* が適用されるパス上の中間点として指定します。

```
NET "on_the_way" TPTHRU="here";
```

メモ：次のような NCF 制約はサポートされません。

```
TIMESPECT "TS_1"=THRU "Thru_grp" 30.0
```

## Constraints Editor からの設定

ISE® の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リストボックスの [Group Constraints] の下の [By Through Points] をダブルクリックし、ダイアログボックスにアクセスします。

## PCF 構文

```
PATH "name"=FROM "source" THRU "thru_pt1" THRU "thru_ptn" TO "destination";
```

FROM、THRU、TO すべてを必ずしも指定する必要はありません。FROM-TO、FROM-THRU-TO、THRU-TO、TO、FROM、FROM-THRU-THRU-THRU-TO、FROM-THRU などさまざまな組み合わせで指定できます。THRU ポイントの数に制限はありません。ソース、THRU ポイント、デスティネーションには、ネット、BEL、コンポーネント、マクロ、ピン、タイムグループを指定できます。

## TSidentifier

*TSidentifier* は、基本的なタイミング制約です。この「TS」という文字で始まる *TSidentifier* プロパティは、TIMESPEC キーワードと共に使用します。*TSidentifier* の値は特定のタイミング仕様に対応し、これをデザイン内のパスに適用できます。

## アーキテクチャ サポート

この制約は、すべての FPGA および CPLD デバイスに適用できます。

## 適用可能エレメント

TIMESPEC キーワード

## 適用ルール

ネット、信号、デザイン エレメントには設定できません。

次の構文では、区切り文字としてスペースを使用していますが、コロン (:) も使用できます。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF および NCF 構文

#### 最大許容遅延の定義

```
TIMESPEC "TSidentifier"=FROM "source_group" TO "dest_group" allowable_delay [units];
```

#### 中間点の定義 (UCF)

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

この場合、次を表します。

- ・ *identifier* は、英数字 (A ~ Z, a ~ z, 0 ~ 9) およびアンダースコア ( ) によって構成される ASCII 文字列です。
- ・ *source\_group* および *dest\_group* は、ユーザー定義のグループ、または定義済みのグループです。
- ・ *thru\_point* は、パスを指定するために使用する中間点で、TPTHRU 制約で定義します。
- ・ *allowable\_delay* は、タイミング要件です。
- ・ *units* は、オプションで許容遅延の単位を指定します。デフォルトの単位はナノ秒ですが、ps、ns、micro、ms、GHz、MHz、または kHz のいずれかを指定できます。

### 仕様の結合

ある仕様で使用されているタイミングの値を別の仕様に結合できます。

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

この場合、次を表します。

- ・ *identifier* は、英数字 (A ~ Z, a ~ z, 0 ~ 9) およびアンダースコア ( ) によって構成される ASCII 文字列です。
- ・ *source\_group* および *dest\_group* は、ユーザー定義のグループ、または定義済みのグループです。
- ・ *another\_Tsid* は、別のタイムスペックの名前です。
- ・ *number* は、浮動小数点数です。

### クロック周期の定義

単純なクロック周期だけでなく、より複雑な派生関係も定義できます。

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference" value [units] [(HIGH | LOW) [high_or_low_time [hi_lo_units]]] INPUT_JITTER value;
```

この場合、次を表します。

- ・ *identifier* は参照識別名です。
- ・ *TNM\_reference* は、TNM 制約を使用してクロック ネット (またはクロック パスのネット) に設定した識別名です。
- ・ *value* は、必要なクロック周期です。
- ・ *units* は、オプションで許容遅延の単位を指定します。デフォルトの単位はナノ秒 (ns) ですが、micro、ms、ps、ns、GHz、MHz、または kHz のいずれかを指定できます。
- ・ オプションの HIGH または LOW を使用すると、最初のパルスを High または Low に指定できます。
- ・ *high\_or\_low\_time* は、High または Low になっている時間を指定します (オプション)。High か Low かは、この前のキーワードによって指定します。実際の時間を指定する場合は、周期より小さい値にする必要があります。<:cs *high\_or\_low\_time*>: /cs> を指定しない場合、デフォルトのデューティ サイクルは 50% になります。
- ・ *hi\_lo\_units* は、デューティ サイクルの単位を指定します (オプション)。デフォルトはナノ秒 (ns) ですが、High または Low の時間が実際の時間である場合は、ps、micro、ms、ns、% のいずれかを指定できます。

### 派生クロックの指定

**TIMESPEC "TS*identifier*"=PERIOD "TNM\_reference" "another\_PERIOD\_identifier" [/ | \*] number [(HIGH | LOW) *high\_or\_low\_time* [*hi\_lo\_units*]] INPUT\_JITTER value;**

この場合、次を表します。

- ・ *TNM\_reference* は、TNM 制約を使用してクロック ネット (またはクロック パスのネット) に設定した識別名です。
- ・ *another\_PERIOD\_identifier* は、別の周期の仕様で使用されている識別名です。
- ・ *number* は、浮動小数点数です。
- ・ オプションの HIGH または LOW を使用すると、最初のパルスを High または Low に指定できます。
- ・ *high\_or\_low\_time* は、High または Low になっている時間を指定します (オプション)。High か Low かは、この前のキーワードによって指定します。実際の時間を指定する場合は、周期より小さい値にする必要があります。<:cs *high\_or\_low\_time*>: /cs> を指定しない場合、デフォルトのデューティ サイクルは 50% になります。
- ・ *hi\_lo\_units* は、デューティ サイクルの単位を指定します (オプション)。デフォルトはナノ秒 (ns) です。ただし、*high\_or\_low\_time* が実際の計測値である場合、High または Low の時間を示す値の後に ps、micro、ms、% を続けて、単位を指定できます。

### パスの無視

**メモ:** この形式は、CPLD デバイスではサポートされません。

ネット、インスタンス、インスタンス ピンを通るすべてのパスが、タイミング仕様の観点からは重要でない場合、特定のネットを通るパスを無視するよう指定できます。

**TIMESPEC "TS*identifier*"=FROM "source\_group" TO "dest\_group" TIG;**

または

**TIMESPEC "TS*identifier*"=FROM "source\_group" THRU "thru\_point" [THRU "thru\_point1"... "thru\_pointn"] TO "dest\_group" TIG;**

この場合、次を表します。

- ・ *identifier* は、英数字 (A ~ Z, a ~ z, 0 ~ 9) およびアンダースコア ( ) によって構成される ASCII 文字列です。
- ・ *source\_group* および *dest\_group* は、ユーザー定義のグループ、または定義済みのグループです。
- ・ *thru\_point* は、パスを指定するために使用する中間点で、TPTHRU 制約で定義します。

次の文は、タイミング仕様 TS\_35 で、グループ here と there の間の最大許容遅延が 50ns に指定しています。

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

次の文は、タイミング仕様 TS\_70 で、clock\_a のクロック周期が 25ns で、最初のパルスが High、その継続時間が 15ns になるよう指定しています。

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

詳細は、第 2 章「制約のタイプ」の「[論理制約](#)」および「[物理制約](#)」を参照してください。

## Constraints Editor からの設定

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。

クロック周期制約は、[Clock Domains] に入力します。入力セットアップ タイムは、[Inputs] に入力します。clock-to-output 遅延は、[Outputs] に入力します。pad-to-pad 遅延は、[Exceptions] → [Paths] から入力します。

## PCF 構文

TIMESPEC キーワードを使用しない UCF 構文と同じです。

## FPGA Editor からの設定

FPGA Editor のメイン ウィンドウでコンポーネント、ネット、パス、またはピンを選択して [Edit] → [Properties of Selected Items] をクリックし、[Physical Constraints] タブで TSid 制約を設定します。

## U\_SET

U\_SET は高度なマップ制約で、デザイン全体に分散している RLOC 制約が設定されたデザイン エlement を、1 つの集合にグループ化します。U\_SET 集合のエlement は、デザイン階層のどこにあってもかまいません。デザイン階層に関係なく、任意のオブジェクトを選択して、それらを U\_SET 集合のエlement として指定できます。詳細は、「[RLOC の集合](#)」を参照してください。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能エレメント

どのデバイス ファミリーにどのエレメントを使用できるかは、ライブラリ ガイドを参照してください。詳細は、[データシート](#)を参照してください。

- ・ レジスタ
- ・ FMAP
- ・ マクロ インスタンス
- ・ ROM
- ・ RAMS、RAMD
- ・ <:crmk 4>BUFT
- ・ MULT18X18S
- ・ RAMB4\_Sm\_Sn、RAMB4\_Sn
- ・ RAMB16\_Sm\_Sn、RAMB16\_Sn
- ・ RAMB16
- ・ DSP48

## 適用ルール

この制約は、マクロの制約なので、ネットには設定できません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : U\_SET
- ・ 属性値 : *name*

*name* は集合の識別名です。

### VHDL 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

VHDL 制約を次のように宣言します。

```
attribute U_SET: string;
```

VHDL 制約を次のように指定します。

```
attribute U_SET of {component_name | label_name}: {component|label} is name;
```

*name* は集合の識別名です。

VHDL 構文の詳細は、[「VHDL」](#)を参照してください。

### Verilog 構文

Verilog 制約を次のように指定します

```
(* U_SET = name *)
```

*name* は集合の識別名です。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

### UCF および NCF 構文

UCF 構文は次のとおりです。

```
INST "instance_name" U_SET= name;
```

*name* は集合の識別名です。

この名前は絶対名です。名前の前に階層修飾子は付けません。

次の文は、デザイン エLEMENT ELEM\_1 が集合 JET\_SET に属するように指定します。

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

### XCF 構文

```
BEGIN MODEL entity_name
```

```
INST "instance_name" U_SET=uset_name;
```

```
END;
```

## USE\_RLOC

USE\_RLOC は、高度なマップ制約および配置制約で、特定のELEMENTまたは集合の一部に対して RLOC 制約を適用するかどうかを指定します。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能ELEMENT

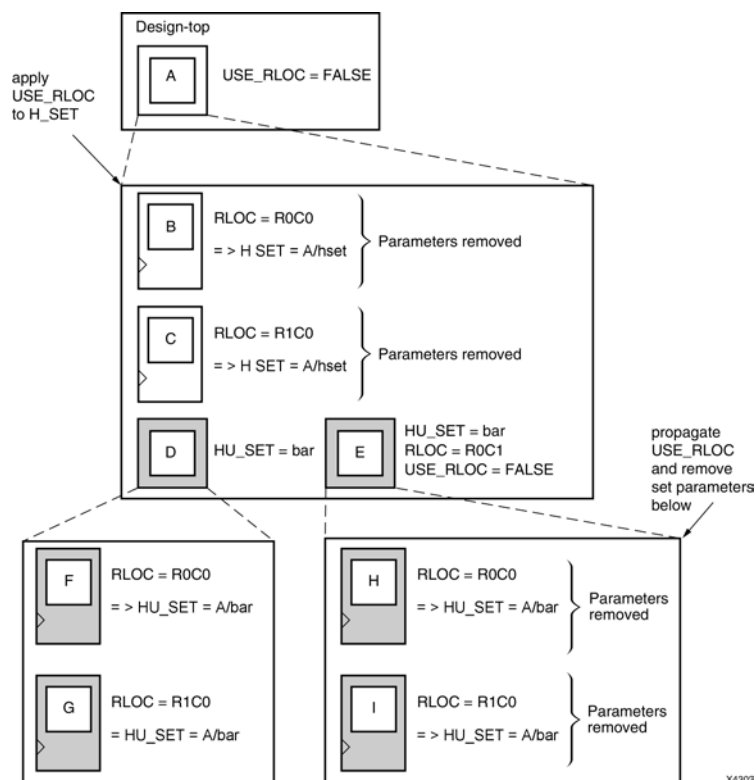
集合の要素であるインスタンスまたはマクロ

## 適用ルール

ネットには設定できません。デザイン ELEMENTに設定すると、そのデザイン ELEMENTの階層にあるすべての適用可能ELEMENTに適用されます。



## USE\_RLOC 制約を使用した H\_SET および HU\_SET 集合の RLOC の制御

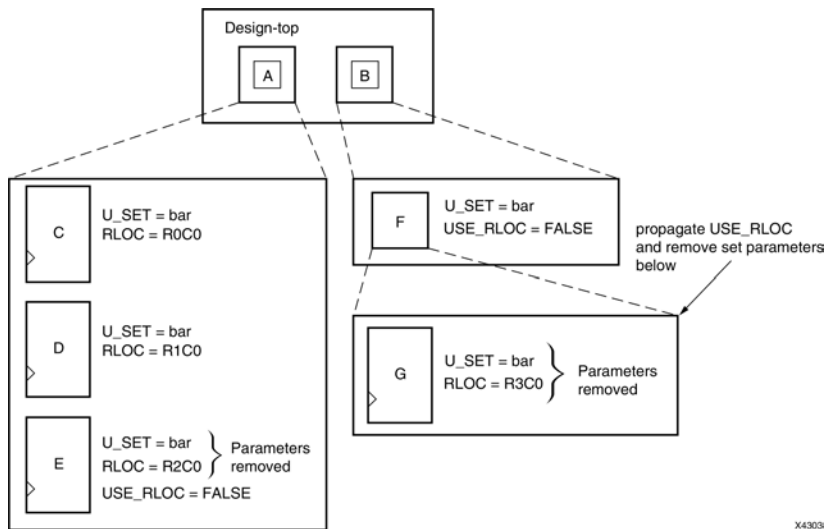


U\_SET 集合には階層がないので、U\_SET 集合に `USE_RLOC` 制約を適用するのは特殊なケースです。`USE_RLOC` 制約は階層を基に厳密に伝搬されるため、デザインの異なる階層レベルにある U\_SET 集合の要素には、別々に `USE_RLOC` 制約を設定する必要があります。特定の `USE_RLOC` 制約は、異なる階層レベルにある集合の要素には伝搬されません。

インスタンスエーション マクロを使用して U\_SET 集合を作成する場合は、そのマクロに `USE_RLOC` 制約を設定して、集合の全要素に階層を通して伝搬できます。

このようなマクロを作成するには、マクロに U\_SET 制約を設定し、そのマクロの下位にある RLOC 制約が設定されているすべてのシンボルに対して、その制約が伝搬されるようにします。

## USE\_RLOC 制約を使用した U\_SET 集合の RLOC の制御



この図は、USE\_RLOC=FALSE の使用例を示しています。プリミティブ E の USE\_RLOC=FALSE によって U\_SET 集合から RLOC 制約が削除されます。また、エレメント F の USE\_RLOC=FALSE はプリミティブ G に伝搬され、U\_SET 集合から RLOC 制約が削除されます。

USE\_RLOC 制約を伝搬するときに、上位階層に既に USE\_RLOC 制約が設定されているエレメントがあると、それより下位の USE\_RLOC 制約が無視されます。たとえば、USE\_RLOC=FALSE 制約を伝搬する場合、USE\_RLOC=TRUE 制約が下位のエレメントに設定されていても、その TRUE 制約は無視されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ 集合のエレメントに設定します。
- ・ 属性名 : USE\_RLOC
- ・ 属性値 : TRUE、FALSE

## VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute USE_RLOC: string;
```

VHDL 制約を次のように指定します。

```
attribute USE_RLOC of entity_name: entity is "{TRUE|FALSE}";
```

- ・ **TRUE** : 指定したエレメントに RLOC 制約が適用されます。
- ・ **FALSE** : RLOC 制約は適用されません。

デフォルトは TRUE です。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

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

- ・ **TRUE** : 指定したエレメントに RLOC 制約が適用されます。
- ・ **FALSE** : RLOC 制約は適用されません。

デフォルトは TRUE です。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

UCF 構文は次のとおりです。

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

- ・ **TRUE** : 指定したエレメントに RLOC 制約が適用されます。
- ・ **FALSE** : RLOC 制約は適用されません。

デフォルトは TRUE です。

## XCF 構文

```
MODEL "entity_name" use_rloc={true|false};
```

## VCCAUX

デバイスの VCCAUX ピンの電圧値を定義します。有効な値は 2.5 または 3.3 で、デフォルト値は 2.5 です。この設定は、PACE だけでなく、自動配置プログラム内の I/O 配置のバンク規則にも影響します。また、デバイスのビットストリームにも影響を与えます。

## アーキテクチャ サポート

この制約は、Spartan®-3A デバイスにのみ適用できます。

## 適用可能エレメント

Spartan-3A デバイスのグローバル属性で、特定のエレメントには設定されません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## UCF および NCF 構文

次のように、出力または双方向ポートに配置します。

```
CONFIG VCCAUX="value";
```

value は 2.5 または 3.3 です。

例 :

```
CONFIG VCCAUX=3.3;
```

## VOLTAGE

VOLTAGE はタイミング制約で、指定した電圧に基づいてデバイスの遅延特性を比例配分できるように動作電圧を指定します。比例配分は、既存のスピード ファイルの遅延に対して行われ、すべての遅延に対してグローバルに適用されます。

**メモ:** 新しいデバイスでは、タイミング情報 (スピード ファイル) が Production ステータスになるまで電圧の比例配分がサポートされない場合があります。

アーキテクチャにはそれぞれ、サポートされる電圧の範囲があります。入力した電圧がサポートされる範囲外の場合、この制約は無視され、アーキテクチャのデフォルト値が使用されます。これに関するエラー メッセージは、スタティック タイミングで出力されます。

## アーキテクチャ サポート

次の FPGA がサポートされています。

- ・ Spartan®-3A
- ・ Spartan-3AN
- ・ Spartan-3A DSP
- ・ Spartan-3E
- ・ Virtex®-4
- ・ Virtex-5

## 適用可能エレメント

グローバル

## 適用ルール

ネット、信号、デザイン エレメントには設定できません。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### UCF および NCF 構文

**VOLTAGE=** *value* [V];

*value* 電圧を指定する実数です。

V デフォルトの電圧の単位であるボルトを表します。

次の文は、スピード ファイル遅延に関するすべての解析で、動作電圧が 5 ボルトになるように指定します。

**VOLTAGE=5;**

### Constraints Editor の構文

Project Navigator の [Processes] ウィンドウで [User Constraints] の下の [Create Timing Constraints] をダブルクリックすると、Constraints Editor が起動します。[Constraint Type] リスト ボックスの [Timing Constraints] の下の [Operating Conditions] をダブルクリックし、ダイアログ ボックスにアクセスします。

## PCF 構文

UCF と同じです。

## VREF

VREF 制約は、デザインのエLEMENTには直接適用されず、グローバル属性としてデザインに適用されます。この制約を指定すると、リストされたピンが VREF の供給ピンとして、SSTL または HSTL I/O 標準のいずれかで指定された I/O ピンと共に使用されます。

CoolRunner™-II デザインでは、どの I/O も VREF として使用できるので、この制約を使用することにより、VREF ピンとして選択するピンを指定できます。レポートファイル (RPT) でピン割り当てを確認してください。差動 I/O 規格の HSTL および SSTL に VREF ピンを指定しない場合や、差動 I/O ピンの近くに必要 VREF ピンが指定されていない場合は、フィッタにより自動的に必要な VREF が割り当てられます。

## アーキテクチャ サポート

この制約は、128 個以上のマクロセルを使用した CoolRunner-II デバイスにのみ適用できます。

## 適用可能エレメント

グローバル

## 適用ルール

リストされたピンが VREF の供給ピンとして、SSTL または HSTL I/O 標準のいずれかで指定された I/O ピンと共に使用されます。

## 構文例

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

VREF=*value\_list* (CONFIG シンボル上)

有効な値は、次のとおり。

- ・ *Pnn*  
この場合、次を表します。  
*nn* は、ピン番号です。
- ・ *rc*  
この場合、次を表します。
  - *r* は、行を表すアルファベットです。
  - *c* は、列を表す数値です。

## UCF および NCF 構文

**CONFIG** VREF=*value\_list*;

有効な値は、次のとおりです。

- ・  $P_{nn}$

この場合、次を表します。

$nn$  は、ピン番号です。

- ・  $rc$

この場合、次を表します。

$r$  は、行を表すアルファベットです。

$c$  は、列を表す数値です。

```
CONFIG VREF=P12,P13;
```

## WIREAND

WIREAND は高度なフィッタ制約で、指定したノードをワイヤード AND ファンクションとしてインターコネクト (UIM および Fastconnect) にインプリメントします。

## アーキテクチャ サポート

この制約は、XC9500 デバイスにのみ適用できます。

## 適用可能エレメント

すべてのネット

## 適用ルール

WIREAND はネット制約なので、デザイン エレメントには適用できません。

### 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

### 回路図

- ・ ネットに設定します。
- ・ 属性名 : WIREAND
- ・ 属性値 : TRUE、FALSE

### VHDL 構文

VHDL 制約を次のように宣言します。

```
attribute WIREAND: string;
```

VHDL 制約を次のように指定します。

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

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

この制約はインスタンス化の直前に入力します。

Verilog 制約を次のように指定します

```
(* WIREAND = "{YESE|NO|TRUE|FALSE}" *)
```

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

次の文は、最適化の際にネット SIG\_11 がワイヤード AND としてインプリメントされるよう指定します。

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

## XBLKNM

XBLKNM 制約は、高度なマップ制約で、ブロック名をプリミティブおよびロジック エlement に割り当てます。つの XBLKNM 制約を複数のインスタンスに設定した場合、同じブロック名が付けられたロジックが 1 つまたは複数のスライスにマップされます。逆に、2 つのシンボルの XBLKNM 名が異なる場合は、同じブロックにはマップされません。1 つのブロックに収まらない複数のインスタンスに同じ XBLKNM を設定すると、エラーが発生します。

FMAP シンボルに同じ XBLKNM を指定すると、関連するファンクション ジェネレータが 1 つのスライスにグループ化されます。XBLKNM を使用すると、スライスをデバイス上の物理的な位置に制約することなく、スライスを分割できます。

XBLKNM には階層パスは接頭辞として追加されないため、各スライスには固有の XBLKNM 名を割り当てる必要があります。

BLKNM を使用すると、別の BLKNM が指定されている Element を除き、すべての Element を同じ物理的コンポーネントにマップできます。XBLKNM 属性の場合は、同じ XBLKNM が設定されている Element のみが、同じ物理コンポーネントにマップされます。XBLKNM が設定されていない Element は、XBLKNM が設定されている Element と同じ物理コンポーネントにはマップできません。

**メモ** : この制約は、ブロック RAM とも使用できます。

## アーキテクチャ サポート

この制約は、FPGA デバイスにのみ適用できます。

## 適用可能 Element

どのデバイス ファミリーにどの Element を使用できるかは、[ライブラリ ガイド](#)を参照してください。詳細は、[データシート](#)を参照してください。

## 適用ルール

設定したデザイン Element に適用されます。

## 構文

次の例では、この制約を特定のツールまたは手法でどのように設定するかを示しています。ツールや手法が記述されていない場合は、その方法では設定できないことを示しています。

## 回路図

- ・ 有効なインスタンスに設定します。
- ・ 属性名 : XBLKNM
- ・ 属性値 : *block\_name*

*block\_name* その種類のシンボルで有効なブロック名です。

## VHDL 構文

この制約は、モジュール宣言またはインスタンス化の直前に入力します。

VHDL 制約を次のように宣言します。

```
attribute XBLKNM: string;
```

VHDL 制約を次のように指定します。

```
attribute XBLKNM of {component_name|label_name}: {component|label} is block_name;
```

*block\_name* その種類のシンボルで有効なブロック名です。

VHDL 構文の詳細は、「[VHDL](#)」を参照してください。

## Verilog 構文

Verilog 制約を次のように指定します

```
(* XBLKNM = "block_name" *)
```

*block\_name* その種類のシンボルで有効なブロック名です。

Verilog 構文の詳細は、「[Verilog](#)」を参照してください。

## UCF および NCF 構文

UCF 構文は次のとおりです。

```
INST "instance_name" XBLKNM=block_name;
```

*block\_name* その種類のシンボルで有効なブロック名です。

次の文は、エレメント flip\_flop2 のインスタンスをブロック U1358 に割り当てます。

```
INST "$1I87/flip_flop2" XBLKNM=U1358;
```

## XCF 構文

```
BEGIN MODEL "entity_name"
```

```
INST "instance_name" xblknm=xblknm_name;
```

```
END;
```