Vivado Design Suite ユーザー ガイド

制約の使用

UG903 (v2012.3) 2012 年 10 月 16 日

XILINX®

本資料は英語版 (v2012.3) 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。

この資料に関するフィードバックおよびリンクなどの問題につきましては、jpn_trans_feedback@xilinx.com までお知らせください。いただきましたご意見を参考に早急に対応させていただきます。なお、このメール アドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	改訂内容

2012 年 10 月 16 日	2012.3	• マイナーなレイアウト、フォーマット、テキストの変更
		• 一部の図をアップデート
		  • 第 3 章の図 3-3
		  • 第 4 章の図 4-1
		• 新しい図を追加
		  • 第 4 章の図 4-4
		  • 第 4 章の図 4-5
		  • 第 4 章の図 4-6
		• 第 3 章の「ホールド チェック」を更新
		• 第 3 章の「ホールド パス要件の例」のホールド要件を更新
		• 第 4 章の「生成クロックについて」を更新
		• 第 4 章の「ユーザー定義の生成クロック」のコード例を更新
		• 第 4 章の「自動的に派生したクロック」のコード例を更新
		• 第 4 章の「クロック グループ」に「また、[Report Clock Interaction] コマンドを使用して、」で始まる文を追加
		• 第 4 章の「ユーザー定義の生成クロック」に「-source オプションではピンまたはポート ネットリスト オブジェクトのみを指定できます。クロック オブジェクトは指定できません」という重要な注記を追加

2012 年 9 月 4 日	2012.2	次を含むマイナーなアップデート • 文章のマイナーなアップデート • 図をアップデート • 30 ページの「セットアップ パス要件の例」を編集 • 32 ページの「ホールド パス要件の例」を編集

目次

第 1 章 : 概要

UCF 制約から XDC 制約への変換 6
XDC 制約について 6
XDC 制約の入力 7

第 2 章 : 制約の入力方法

制約の管理 8
制約の順序 11
制約の入力 14
合成制約の作成 21
インプリメンテーション制約の作成 26

第 3 章 : タイミング解析

タイミング パス 28
セットアップおよびホールド解析 30
リカバリおよびリムーバル解析 34

第 4 章 : クロックの定義

クロックについて 35
プライマリ クロック 36
仮想クロック 38
生成クロック 39
クロック グループ 43
クロック レイテンシ、ジッター、ばらつき 45
I/O 遅延 46

第 5 章 : タイミング例外

最小/最大遅延制約 50

第 6 章 : XDC の優先順位

XDC 制約の順序 52
例外の優先順位 52

第 7 章 : 物理制約

制約の適用 54
ネットリスト制約 55
I/O 制約 55
配置制約 57
配線制約 59
コンフィギュレーション制約 61

付録 A : その他のリソース

ザイリンクス リソース 62
ソリューション センター 62
リファレンス 62

XILINX

第 1 章

概要

Vivado™ 統合設計環境 (IDE) では、ザイリンクス デザイン制約 (XDC) を使用します。

UCF 制約から XDC 制約への変換

ザイリンクス デザイン制約 (XDC) とユーザー制約ファイル (UCF) の制約には、重要な違いがあります。XDC は、業界標準の Synopsys Design Constraints (SDC) フォーマットに基づいています。SDC は 20 年以上も使用され、向上してきており、デザイン制約を記述するのに最もよく使用される確立したフォーマットです。UCF に慣れていて XDC を初めて使用する場合は、付録 A「その他のリソース」のリンクから『Vivado Design Suite 移行手法ガイド』(UG911) の「UCF 制約を XDC に移行」の章にある「XDC と UCF 制約の違い」を参照してください。この章には、XDC 制約を作成する開始点として既存の UCF ファイルを XDC に変換する方法が説明されています。

重要 : デザインを適切に制約するには、XDC と UCF の基本的な違いを理解する必要があります。UCF から XDC への変換ユーティリティでは限界があり、XDC を正しく理解して作成するのが確実な方法です。このユーザー ガイドでは、各 XDC 制約について説明します。

XDC 制約について

XDC 制約は、次を組み合わせたものです。

- 業界標準の Synopsys Design Constraints (SDC)
- ザイリンクス独自の物理制約

XDC には、次の特徴があります。

- 単純な文字列ではなく、Tcl のフォーマットに従ったコマンドです。
- Vivado Tcl インタープリターにより、ほかの Tcl コマンドと同様に解釈されます。
- ほかの Tcl コマンドと同様に、順次読み込まれ、解析されます。

XDC 制約の入力

XDC 制約は、フローの異なる段階でいくつかの方法で入力できます。

- 制約を 1 つまたは複数のファイルに保存し、プロジェクトの制約セットに追加できます。
- ファイルを読み込むには、**read_xdc** コマンドを使用します。
- デザインがメモリに読み込まれたら、Tcl コンソールに制約を直接入力します。

  これは、新しい制約を個別に入力、検証、デバッグするのに有益な方法です。

XILINX

第 2 章

制約の入力方法

この章では、推奨される制約入力フローを示します。

デザイン制約は、デザインがボード上で正しく機能するようにするために、コンパイル フローで満たす必要のある要件を定義します。すべての制約がコンパイル フローのすべての段階で使用されるわけではありません。たとえば、物理制約はインプリメンテーション段階 (配置および配線) でのみ使用されます。

Vivado™ 統合設計環境 (IDE) の合成およびインプリメンテーション アルゴリズムはタイミング ドリブンなので、適切なタイミング制約を作成する必要があります。デザインの制約を厳しくしすぎたり緩くしすぎたりすると、タイミング クロージャを達成するのが困難になります。アプリケーションの要件に対応する適度な制約を使用する必要があります。

制約の管理

Vivado IDE では、1 つまたは多数の制約ファイルを使用できます。コンパイル フロー全体に 1 つの制約ファイルを使用する方が便利なように思えますが、デザインが複雑になればすべての制約を管理するのは簡単ではありません。これは、異なるチームが開発した複数の IP や大型のブロックを使用するデザインに特に言えます。

推奨 : タイミング制約と物理制約を別のファイルに保存することをお勧めします。また、特定のモジュール用の制約を別のファイルに保存することもできます。

図 2-1 「1 つの XDC ファイルまたは複数の XDC ファイル」に、プロジェクトに制約セットが 2 つ含まれている例を示します。

- 1 つ目の制約セットには、2 つの XDC ファイルが含まれます。
- 2 つ目の制約セットでは、すべての制約を含む XDC ファイルを 1 つ使用します。

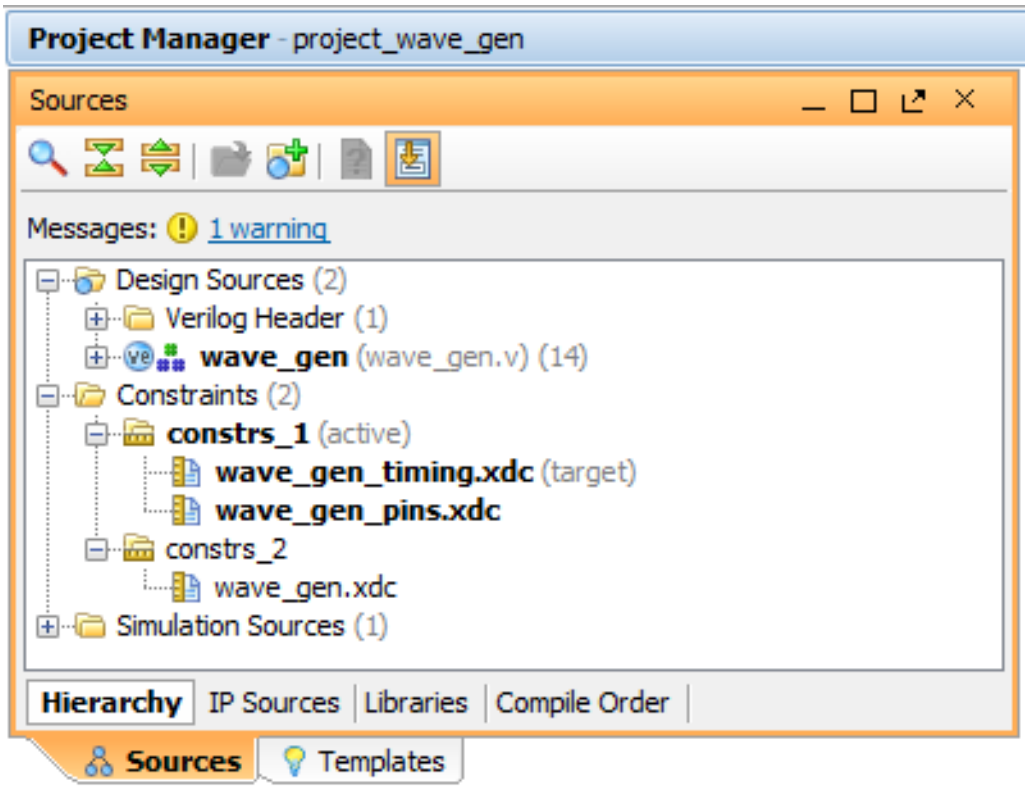

図 2-1 : 1 つの XDC ファイルまたは複数の XDC ファイル

非プロジェクト フロー

非プロジェクト フローで同じ結果を得るには、コンパイル コマンドを実行する前に各ファイルをそれぞれ読み込みます。

次に、合成およびインプリメンテーションで複数の XDC ファイルを使用するスクリプト例を示します。

スクリプト例

```
read_verilog [glob src/*.v]
read_xdc wave_gen_timing.xdc
read_xdc wave_gen_pins.xdc
synth_design -top wave_gen
opt_design
place_design
route_design
```

合成またはインプリメンテーションでの制約ファイルの使用

制約ファイルは、次のように使用できます。

- 合成のみ
- インプリメンテーションのみ
- 合成およびインプリメンテーションの両方

制約ファイルのプロパティを変更して、その制約ファイルを合成のみに使用するか、インプリメンテーションのみに使用するか、両方に使用するかを指定します。

制約ファイルをインプリメンテーションのみに使用する場合は、次の手順に従います。

1. [Sources] ビューで制約ファイルを選択します。
2. [Source File Properties] ビューで、オプションを次のように設定します。
   a. [Synthesis] をオフ
   b. [Implementation] をオン
3. [Apply] をクリックします。

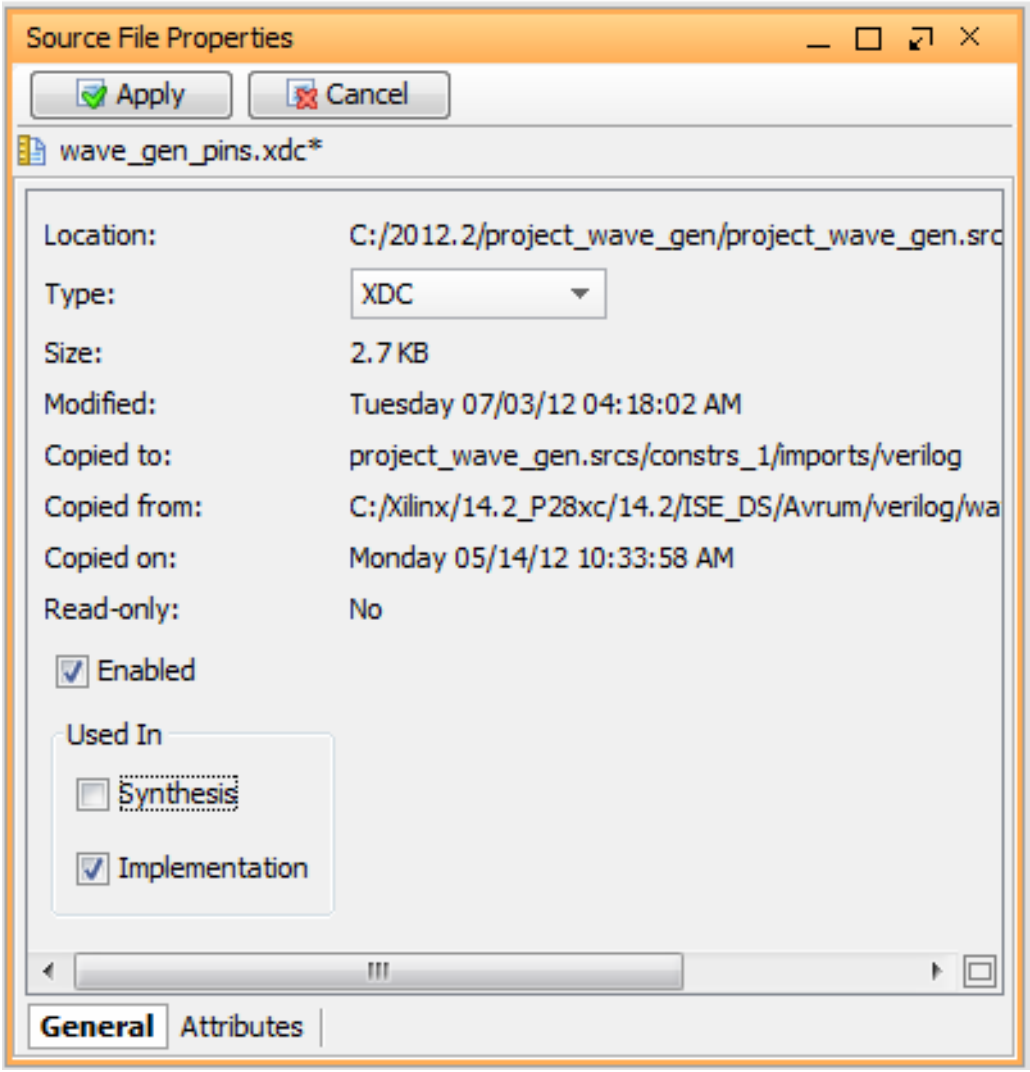

図 2-2 : [Source File Properties] ビュー

同等の Tcl コマンドは、次のとおりです。

```
set_property used_in_synthesis false [get_files wave_gen_pins.xdc]
set_property used_in_implementation true [get_files wave_gen_pins.xdc]
```

プロジェクトなしで Vivado IDE を実行する場合、フローのどの段階の間でも制約を直接読み込むことができます。

次に、2 つの XDC ファイルを読み込む Tcl スクリプト例を示します。

```
read_verilog [glob src/*.v]
read_xdc wave_gen_timing.xdc
synth_design -top wave_gen -part xc7k325tffg900-2
read_xdc wave_gen_pins.xdc
opt_design
place_design
route_design
```

表 2-1 : 2 つの XDC ファイルの読み込み

ファイル名	ファイルを読み込む段階	制約が使用される段階
wave_gen_timing.xdc	合成前	• 合成 • インプリメンテーション
wave_gen_pins.xdc	合成後	• インプリメンテーション

制約の順序

XDC 制約は順に適用され、規則に従って優先順位が付けられるので、制約の順序を注意深く確認する必要があります。詳細は、第 6 章「XDC の優先順位」を参照してください。

Vivado IDE では、デザインの内部を詳細に確認できます。制約を検証するには、次の手順に従います。

1. 適切なレポート コマンドを実行します。
2. [Tcl Console] または [Messages] ビューメッセージを確認します。

推奨される制約の順序

推奨 : デザインで XDC ファイルを 1 つまたは複数使用する場合のどちらでも、制約を次の順序で指定します。

```
## タイミング アサーション セクション
# プライマリ クロック
# 仮想クロック
# 生成クロック
# クロック グループ
# 入力および出力遅延制約

## タイミング例外セクション
# フォルス パス
# 最大遅延/最小遅延
# 複数サイクル パス
# ケース解析
# タイミングのディスエーブル

## 物理制約セクション
# ファイルのどこに配置しても可、タイミング解析の前か後が理想的
# または別の XDC ファイルに保存
```

まずクロック定義から開始します。クロックを作成しないと、ほかの制約で使用できません。クロック定義の前にクロックを参照すると、エラーが発生し、その制約は無視されます。これは、1 つの XDC ファイル内であっても、デザイン内の複数の XDC ファイルであっても同様です。

XDC ファイルの順序

XDC ファイルの順序は重要です。各ファイル内の制約が、ほかのファイル内の制約に依存しないようにする必要があります。あるファイル内の制約がほかのファイル内の制約に依存している場合は、依存する制約を含むファイルを最後に読み込む必要があります。2 つの制約ファイルにお互いのファイル内にある制約に依存する制約が含まれている場合は、次のいずれかを実行します。

- 2 つのファイルを 1 つにまとめ、制約を適切な順序に並べ替えます。
- ファイルを複数のファイルに分割し、正しい順序で読み込みます。

制約ファイルの順序

プロジェクト フローでは、制約は制約セットに含まれます。エラボレートされたネットリストまたは合成済みネットリストを開くと、制約ファイルは制約セットにリストされた順序で読み込まれます。これは、Vivado IDE の表示では上から下への順序になります。

たとえば、9 ページの図 2-1 「1 つの XDC ファイルまたは複数の XDC ファイル」で constr_1 に含まれる 2 つの XDC ファイルは、次の順序で読み込まれます。

表 2-2 : 制約ファイルの順序

ファイルの順序	ファイル名	読み込み順
1 番目	wave_gen_timing.xdc	1 番目
2 番目	wave_gen_pins.xdc	2 番目

読み込み順の変更

読み込み順を変更するには、次の手順に従います。

1. 移動する XDC ファイルを選択します。
2. XDC ファイルをドラッグして、適切な位置に移動します。

9 ページの図 2-1 「1 つの XDC ファイルまたは複数の XDC ファイル」に示す例では、Tcl コマンドは次のようになります。

```
reorder_files -fileset constrs_1 -before [get_files wave_gen_timing.xdc] \
  [get_files wave_gen_pins.xdc]
```

非プロジェクト フローでは、**read_xdc** コマンドで呼び出す順序により制約ファイルの順序が決定します。

制約ファイルを含むネイティブ IP を使用する場合は、IP の XDC ファイルはデザインのほかの XDC ファイルの後に、[IP Sources] タブにリストされる IP の順に読み込まれます。たとえば、図 2-3 「IP ソースの XDC ファイル」ではプロジェクト IP の 1 つに XDC ファイルが付属しています。

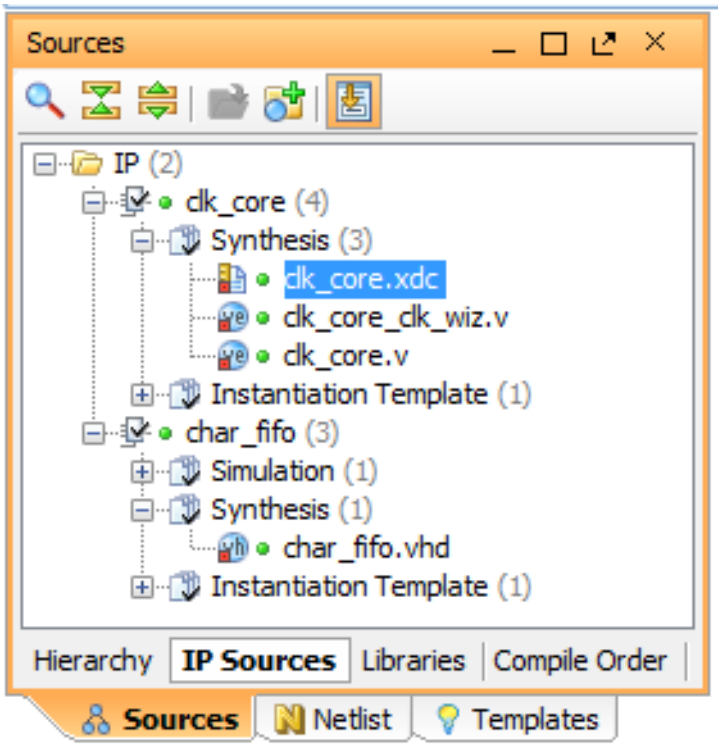

図 2-3 : IP ソースの XDC ファイル

デザインを開くと、ログ ファイルに IP の XDC ファイルが最後に読み込まれていることが示されます。

```
Parsing XDC File [C:/project_wave_gen.srcs/constrs_2/wave_gen_all.xdc]
INFO: [Timing 38-35] Done setting XDC timing
constraints.[C:/project_wave_gen.srcs/constrs_2/wave_gen_all.xdc:9]
INFO: [Timing 38-2] Deriving generated clocks
[C:/project_wave_gen.srcs/constrs_2/wave_gen_all.xdc:9]
Finished Parsing XDC File [C:/project_wave_gen.srcs/constrs_2/wave_gen_all.xdc]
Parsing XDC File [c:/project_wave_gen.srcs/sources_1/ip/clk_core/clk_core.xdc] for
cell 'clk_gen_i0/clk_core_i0/inst'
Finished Parsing XDC File
[c:/project_wave_gen.srcs/sources_1/ip/clk_core/clk_core.xdc] for cell
'clk_gen_i0/clk_core_i0/inst'
```

IP の XDC ファイルの順序を直接変更することはできません。変更が必要な場合は、次を実行します。

1. IP の XDC ファイルをディスエーブルにします ([Properties] ビューで [Enabled] をオフ)。
2. そのファイルの内容をコピーします。
3. 制約セットに含まれる XDC ファイルのいずれかにコピーした内容を貼り付けます。
4. コピーした IP の XDC 制約で名前を完全な階層パスを含むものに変更します。

制約の順序の編集

Vivado IDE では、編集した制約は XDC ファイルの元の位置に保存されます。新しい制約は、ターゲットとマークされた制約ファイルの最後に追加されます。制約セットに複数の XDC ファイルが含まれている場合、通常ターゲット制約ファイルはリストの最後のファイルではないので、デザインを開いたときまたは読み込み直したときに最後に読み込まれるわけではありません。そのため、ディスクに保存された制約の順序がメモリ内での順序と異なってしまう可能性があります。

重要 : 制約ファイルに保存されている最終的な順序が正しいことを確認する必要があります。順序を変更する必要がある場合は、XDC ファイルを直接編集します。これは、特にタイミング制約で重要です。

制約の入力

Vivado IDE では、いくつかの方法で制約を入力できます。テキスト エディターで直接 XDC ファイルを編集する場合以外は、Vivado IDE の制約入力機能にアクセスするにはデザイン データベース (エラボレート済み、合成済み、またはインプリメント済み) を開く必要があります。

メモリ内の制約の保存

編集中の制約を検証するには、メモリ内にデザインが読み込まれていることが必要です。Vivado IDE ユーザー インターフェイスを使用して制約を編集する場合は、Tcl コンソールで同等の XDC コマンドが実行され、メモリ内にも適用されます ([Timing Constraints] ビューを除く)。

合成またはインプリメンテーションを実行する前に、メモリ内の制約をプロジェクトの XDC ファイルに保存する必要があります。Vivado IDE では、制約の保存が必要な場合はメッセージが表示されます。

制約を手動で保存するには、次のいずれかを実行します。

- [Save Constraints] ボタンをクリックします。
- [File] → [Save Constraints] をクリックします。

制約編集フロー

図 2-4 「制約編集フロー」に、推奨される制約編集フローを示します。これらのフローを混合しないでください。混合すると、制約が失われる可能性があります。推奨されるフローは、次のとおりです。

- ユーザー インターフェイスを使用
- 手動で編集

ユーザー インターフェイスを使用

制約は Vivado IDE で管理されるので、XDC ファイルを同時に編集しないでください。Vivado IDE でメモリの内容を保存すると、制約は次のように保存されます。

- 変更された制約は、元のファイルの元の制約に上書きされます。
- 新しい制約は、ターゲットとマークされた制約ファイルの最後に追加されます。
- XDC ファイルに手動で変更を加えている場合、それらはすべて上書きされます。

手動で編集

制約を手動で編集する場合、ユーザーが XDC ファイルを変更および管理します。Tcl コンソールを使用して制約の構文を確認することはできますが、デザインを閉じたり読み込み直したりすると、メモリ内の変更は破棄されます。

制約を保存するときに競合がある場合は、次のいずれかを選択するようメッセージが表示されます。

- メモリ内の変更を破棄
- 変更を新しいファイルに保存
- XDC ファイルを上書き

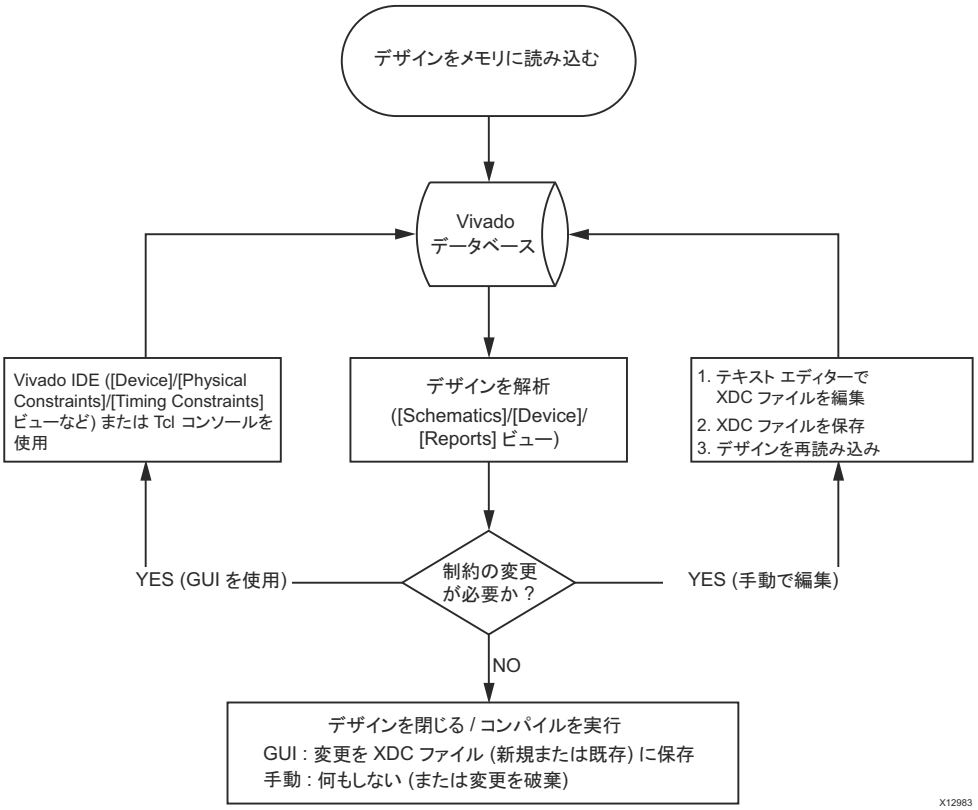

図 2-4 : 制約編集フロー

制約の作成は反復作業になります。一部にインターフェイス機能を使用し、ほかの部分は制約ファイルを手動で変更することも可能です。

図 2-4 「制約編集フロー」に示すフローを実行するときに、同時に両方のフローを使用しないでください。

2 つのフローを切り替えて使用する場合は、まず制約を保存するかデザインを読み込み直し、メモリ内の制約が XDC ファイルと一致するようにします。

ピン割り当て

最上位ポートを作成および既存の配置を変更するには、次の手順に従います。

1. [I/O Planning] レイアウトを選択します。
2. 次の表に示すビューを使用して、ポートを作成または変更します。

表 2-3 : 最上位ポートを作成および既存の配置を変更するのに使用するビュー

ビュー	機能
Device	デバイス フロアプランでポートの位置を表示および変更します。
Package	デバイス パッケージでポートの位置を表示および変更します。
I/O Ports	ポートを選択して [Device] または [Package] ビューにドラッグ アンド ドロップして配置したり、各ポートの現在の割り当てを表示します。
Package Pins	各 I/O バンクのリソースの使用状況を表示します。

ピン割り当ての詳細は、付録 A 「その他のリソース」のリストから『Vivado Design Suite ユーザー ガイド : I/O およびクロックの配置』(UG899) を参照してください。

クロック リソースの割り当て

クロック ツリーの配置を表示および変更するには、次の手順に従います。

1. [Clock Planning] レイアウトを選択します。
2. 次の表に示すビューを使用して、クロック リソースを表示または変更します。

表 2-4 : クロック ツリーの配置を表示および変更するのに使用するビュー

ビュー	機能
Clock Resources	• アーキテクチャでクロック リソース間の接続を表示します。 • クロック ツリー セルの現在の位置を表示します。
Netlist	• [Netlist] ビューでクロック リソースを選択し、[Clock Resources] または [Device] ビューにドラッグ アンド ドロップします。

クロック リソース割り当ての詳細は、付録 A 「その他のリソース」のリストから『Vivado Design Suite ユーザー ガイド : I/O およびクロックの配置』(UG899) を参照してください。

フロアプラン

Pblock を作成および変更するには、次の手順に従います。

1. [Floorplanning] レイアウトを選択します。
2. 次の表に示すビューを使用して、Pblock を作成または変更します。

表 2-5 : Pblock を作成および変更するのに使用するビュー

ビュー	機能
Netlist	Pblock に割り当てるセルを選択します。
Physical Constraints	既存の Pblock とそのプロパティを確認します。
Device	デバイス上の Pblock の形状と場所を作成および変更します。

特定の BEL またはサイトにセル配置制約を作成するには、次の手順に従います。

1. [Netlist] ビューでセルを選択します。
2. セルをドラッグして [Device] ビューの適切な位置に配置します。

フロアプランの詳細は、付録 A 「その他のリソース」のリストから『Vivado Design Suite ユーザー ガイド : デザイン解析およびクロージャ テクニック』(UG906) を参照してください。

タイミング制約

[Timing Constraints] ビューは、合成済みデザインおよびインプリメント済みデザインでのみ使用可能です。エラボレートされたデザインの制約については、「合成制約の作成」を参照してください。

[Timing Constraints] ビューを開くには、図 2-5 に示す 3 つのオプションのいずれかを使用します。

- [Window] → [Timing Constraints] をクリックします。
- Flow Navigator で [Synthesis] → [Synthesized Design] → [Edit Timing Constraints] をクリックします。
- Flow Navigator で [Implementation] → [Implemented Design] → [Edit Timing Constraints] をクリックします。

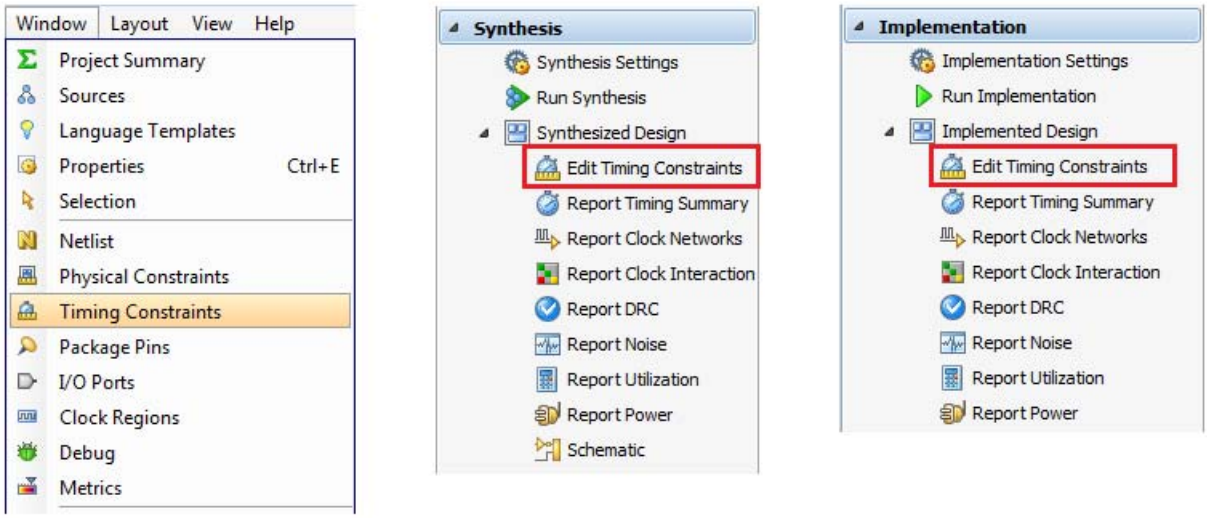

図 2-5 : [Timing Constraints] ビューを開く方法

[Timing Constraints] ビューには、メモリ内のタイミング制約が次のいずれかの順に表示されます。

- XDC ファイルで記述されているのと同じ順序
- [Tcl Console] ビューで入力したのと同じ順序

制約の表

[Timing Constraint] ビューには、既存の制約の詳細が表形式で示されます。この表を利用して、制約オプションを確認および変更します。

制約の作成にはダイアログ ボックスを使用することをお勧めしますが、表の右側にある [+] ボタンをクリックして、表に表示されている制約と同じタイプの制約を作成できます。表に新しい行が追加され、各オプションの値を入力できるようになります。

新しく追加した制約が制約の最後に追加されているかどうかは、[Position] 列の番号で確認できます。

新しく追加した制約は [Apply] をクリックすると検証され、メモリ内のデザインに適用されます。[Apply] をクリックすると、次のようになります。

- 適用する前にメモリ内のタイミング制約がリセットされます。
- 制約は XDC ファイルには保存されません。

制約の作成 (カテゴリ別)

[Timing Constraints] ビューの左上のペインで制約のカテゴリを選択すると、右側のペインにそのカテゴリの制約の表が表示されます。制約を作成するには、カテゴリ名をダブルクリックします。各オプションの値を指定するダイアログ ボックスが表示されます。ダイアログ ボックスで [OK] をクリックすると、次のようになります。

1. 構文が検証されます。
2. 制約がメモリに適用されます。
3. 新しい制約が表の最後に追加されます。
4. 新しい制約が全制約のリストの最後に追加されます。

すべての制約

[Timing Constraints] ビューの下部には、メモリに読み込まれている全制約のリストが適用されている順に表示されます。

- 制約を削除するには、その制約を選択して [X] をクリックします。
- 表で制約を変更し、再適用します。

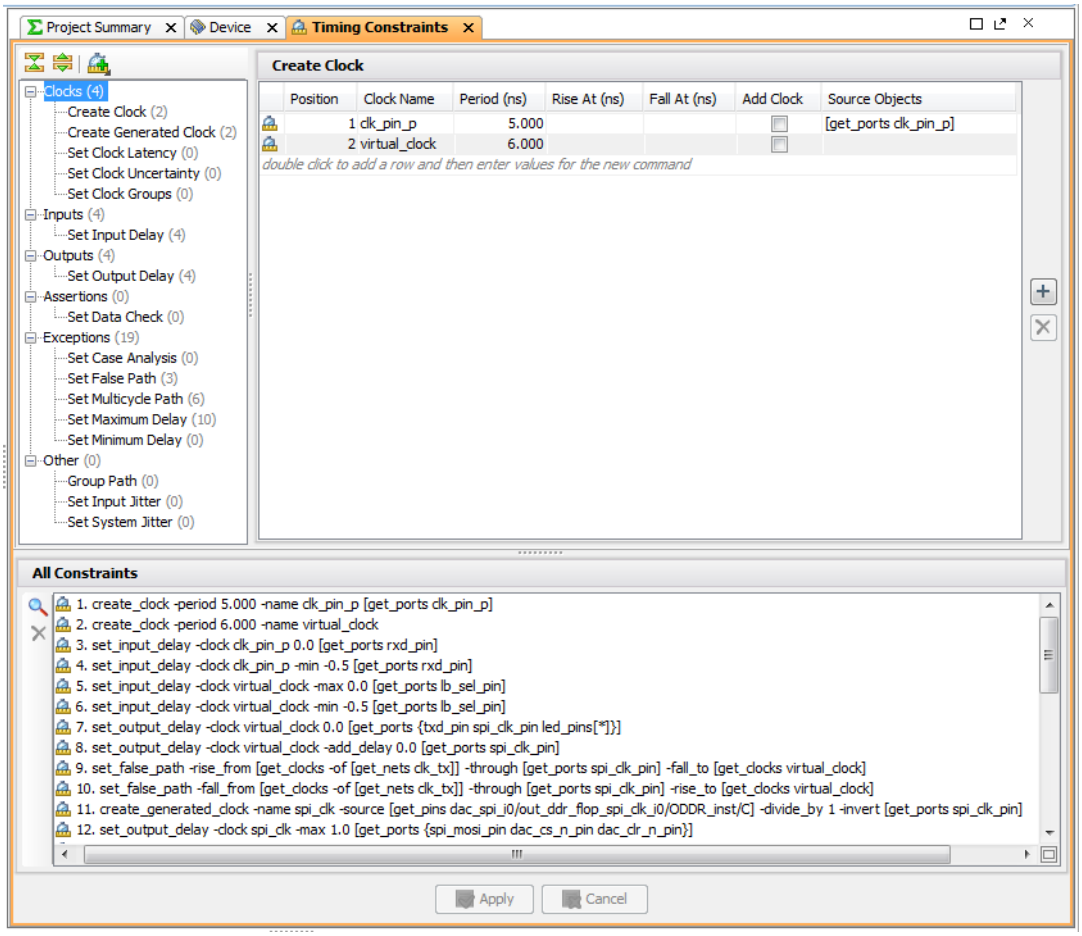

図 2-6 : [Timing Constraints] ビュー

[Tcl Console] ビューで有効なタイミング制約を入力すると、[Timing Constraints] ビューの全制約のリストの最後にすぐに追加されます。

[Timing Constraints] ビューの表で新しい制約を追加したり変更したりすると、その制約は [Apply] をクリックするまでメモリに適用されません。

注意 : [Timing Constraints] ビューの表に適用されていない制約があるときに、[Tcl Console] ビューで新しい制約を入力しないでください。[Timing Constraints] ビューのリストでの制約の順序とメモリでの制約の順序が異なるものになる可能性があります。混乱を避けるため、制約を追加したり変更したら、必ずすべての制約を適用し直すようにしてください。

制約は頻繁に保存してください。制約を保存するには、ツールバーの [Save Constraints] をクリックするか、または [File] → [Save Constraints] をクリックします。

XDC テンプレート

XDC テンプレートには、[Language Templates] ビューからアクセスできます。

XDC テンプレートの内容

XDC テンプレートには、次が含まれます。

- 次のようなよく使用されるタイミング制約
  - クロック定義
  - ジッター
  - 入力/出力遅延
  - 例外
- 物理制約
- コンフィギュレーション制約

XDC テンプレートの使用

XDC テンプレートを使用するには、次の手順に従います。

1. 使用するテンプレートを選択します。
2. [Preview] セクションに表示されるテキストをコピーします。
3. XDC ファイルにテキストを貼り付けます。
4. 汎用文字列をデザインの実際の名前または値に置き換えます。

アドバンス XDC テンプレート

システム同期およびソース同期 I/O 遅延制約などのアドバンス制約では、Tcl 変数を使用してデザイン要件を取得し、`set_input_delay` および `set_output_delay` 制約で使用します。

デフォルト値ではなく必要な値が代入されていることを確認してください。

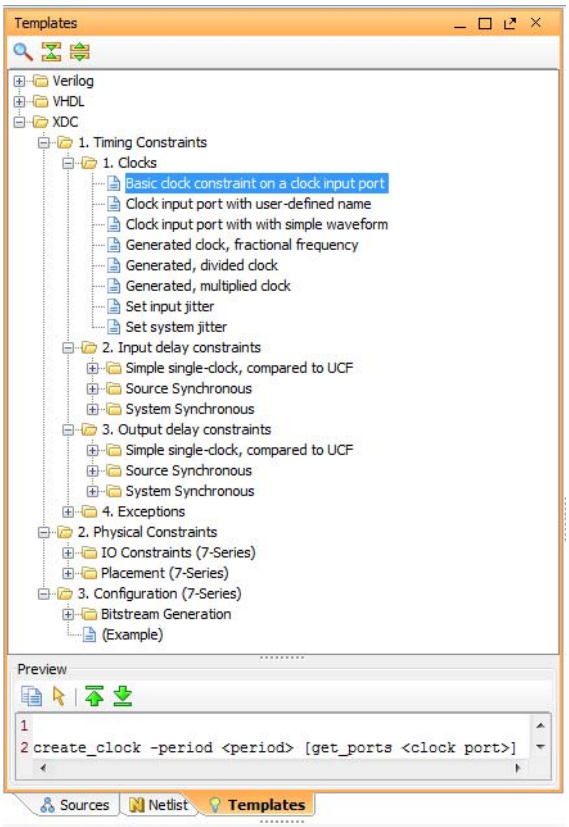

図 2-7 : XDC テンプレート

合成制約の作成

Vivado IDE 合成エンジンでは、デザインの RTL 記述がテクノロジにマップされたネットリストに変換されます。このプロセスは複数の段階で実行され、多数のタイミング ドリブン最適化が含まれます。

ザイリンクス FPGA デバイスには、さまざまな方法で使用可能なロジック機能が多数含まれています。インプリメンテーションの最後にすべてのデザイン要件が満たされるようにするため、制約を使用して合成エンジンに指示を与える必要があります。

Vivado IDE 合成の制約には、次の 3 種類があります。

- RTL 属性
- タイミング制約
- 物理制約およびコンフィギュレーション制約

RTL 属性

RTL 属性は RTL ファイルに記述する必要があります。RTL 属性では通常、ロジックの特定の部分のマップ スタイル、レジスタやネットなどの保持、最終的なネットリストのデザイン階層の制御などを指定します。

詳細は、付録 A 「その他のリソース」のリストから『Vivado Design Suite ユーザー ガイド : 合成』(UG901) を参照してください。

ネットリスト オブジェクトのプロパティとして XDC ファイルで設定できるのは、DONT_TOUCH 属性のみです。

DONT_TOUCH 属性の例

```
set_property DONT_TOUCH true [get_cells fsm_reg]
```

タイミング制約

タイミング制約は、XDC ファイルで合成エンジンに渡す必要があります。セットアップ解析に関する次の制約のみが、合成結果に影響します。

- create_clock
- create_generated_clock
- set_input_delay
- set_output_delay
- set_clock_groups
- set_false_path
- set_max_delay
- set_multicycle_path

物理制約およびコンフィギュレーション制約

物理制約およびコンフィギュレーション制約は、合成アルゴリズムでは無視されます。

RTL ベースの XDC

推奨 : 合成 XDC の最初のバージョンを作成するときは、高レベルのデザイン要件を記述する単純なタイミング制約を使用してください。

フローのこの段階では、ネット遅延のモデリングは正確ではありません。この時点での主な目的は、インプリメンテーションを開始する前に、タイミングを満たすか、タイミングが少しの差で満たされていない合成済みネットリストを得ることです。多くの場合、この状態を達成するには XDC および RTL を何回か修正する必要があります。

図 2-8 「エラボレートされたデザインでの制約の作成」に、RTL ベースの XDC の作成手順を示します。エラボレートされたデザインを使用して、合成用に制約するデザインのオブジェクト名を見つけます。

XDC ファイルを保存する前に、[Tcl Console] ビューを使用して XDC コマンドの構文を確認してください。エラボレートされたネットリストでは、タイミング レポートはサポートされていません。

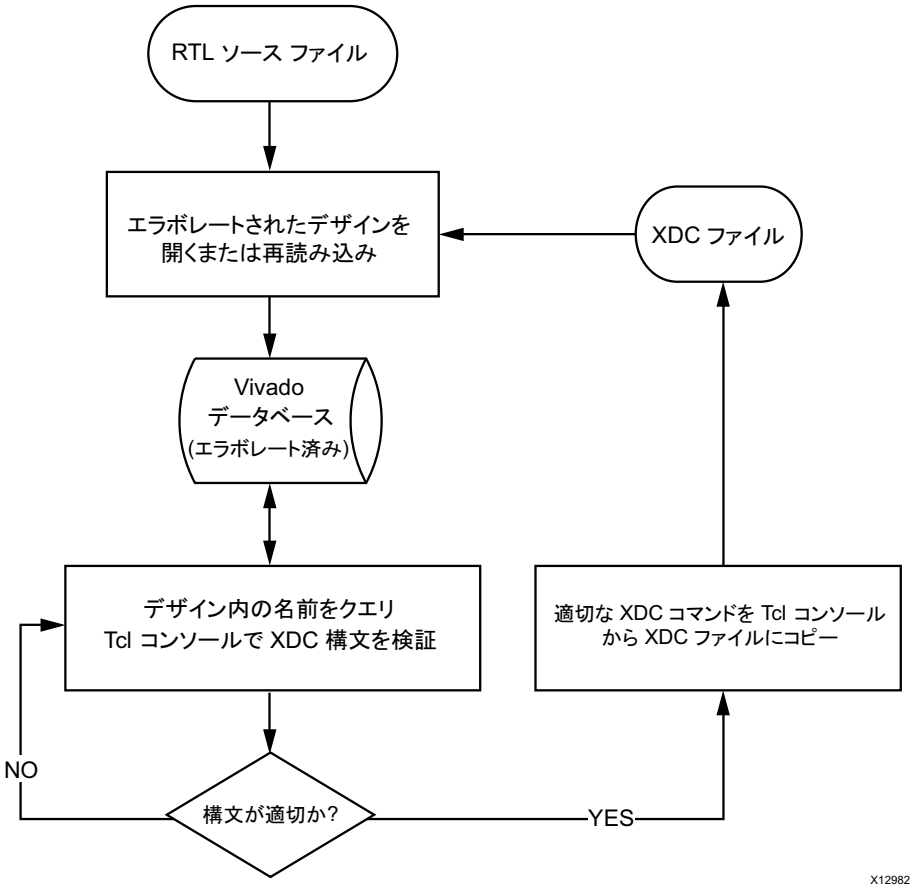

図 2-8 : エラボレートされたデザインでの制約の作成

合成用の制約を記述する際に確実に使用できるデザイン オブジェクトは、次のとおりです。

- 最上位ポート
- 手動でインスタンシエートされたプリミティブ (セルおよびピン)

エラボレートされたデザインの作成時に、一部の RTL 名が変更されたり失われたりします。これがよく発生するのは、次の名前です。

- 1 ビット レジスタの名前
- 複数ビット レジスタの名前
- 吸収されるレジスタおよびネット
- 階層名

1 ビット レジスタの名前

デフォルトでは、RTL 名の後に **_reg** が付いた名前になります。

VHDL での 1 ビット レジスタ名の例

```
signal wbDataForInputReg : std_logic;
```

Verilog での 1 ビット レジスタ名の例

```
reg wbDataForInputReg;
```

エラボレートされたデザインでの 1 ビット レジスタ名の例

```
wbDataForInputReg_reg
```

図 2-9 「エラボレートされたデザインでの 1 ビット レジスタ」に、ピンを含むレジスタの回路図を示します。必要であれば、XDC コマンドでレジスタのピンを参照することも可能です。

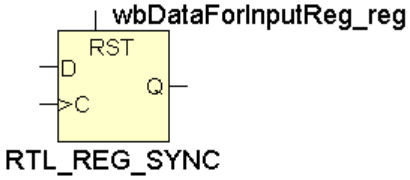

図 2-9 : エラボレートされたデザインでの 1 ビット レジスタ

複数ビット レジスタの名前

デフォルトでは、RTL 名の後に **_reg** が付いた名前になります。エラボレートされたデザインでクエリできない場合でも、XDC 制約で個々のビット参照できます。

VHDL での複数ビット レジスタ名の例

```
signal validForEgressFifo : std_logic_vector(13 downto 0);
```

Verilog での複数ビット レジスタ名の例

```
reg [13:0] validForEgressFifo;
```

エラボレートされたデザインでの複数ビット レジスタ名の例

```
validForEgressFifo_reg
```

図 2-10 「エラボレートされたデザインでの複数ビット レジスタ」に、レジスタの回路図を示します。ピンは、見やすくするためベクターとして表示されます。

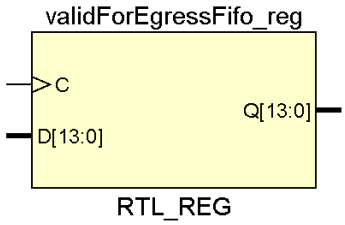

図 2-10 : エラボレートされたデザインでの複数ビット レジスタ

各レジスタを個別に制約するか、次の名前を使用してグループとして制約できます。

- レジスタ ビット 0 のみ

  `validForEgressFifo[0]`

- すべてのレジスタ ビット

  `validForEgressFifo[*]`

上記の名前は合成後のネットリストでの名前とも一致するので、これらに対して設定された制約は通常インプリメンテーションでも機能します。

吸収されるレジスタおよびネット

RTL ソースにあるレジスタやネットの一部が、さまざまな理由により RTL デザイン (または合成済みデザイン) からなくなってしまうことがあります。たとえば、メモリ ブロック、DSP、シフト レジスタの推論では、複数のデザイン オブジェクトを 1 つのリソースに配置することが必要となります。これらのオブジェクトを使用して制約を定義する場合は、接続されている別のレジスタやネットを使用できるか検討してみてください。

階層名

Vivado 合成でデザインの階層を完全に保持するよう指定しない場合、合成中に一部またはすべての階層がフラット化されます。詳細は、付録 A 「その他のリソース」のリストから『Vivado Design Suite ユーザー ガイド : 合成』(UG901) を参照してください。

推奨 : 合成制約には、完全に解決された階層名を使用してください。そのようにすると、階層の変換にかかわらず、最終的なネットリスト名と一致する可能性が高くなります。

たとえば、デザインのサブレベルに次のようなレジスタがあるとします。

RTL のデザイン例

```
inst_A/inst_B/control_reg
```

合成中、このレジスタに特別な最適化は実行されないとすると、ツール オプションでフラット ネットリストまたは階層ネットリストのどちらが指定されているかによって、フラット名または階層名が得られます。

フラット ネットリストの例

```
inst_A/inst_B/control_reg     (F)
```

階層ネットリストの例

```
inst_A/inst_B/control_reg     (H)
```

フラット化された階層レベルを示すのにもスラッシュ (/) が使用されるので、明らかな違いはありません。メモリ内のオブジェクトをクエリする際に、違いがはっきりします。次のコマンドでは、F のネットリスト オブジェクトが返され、H のネットリスト オブジェクトは返されません。

```
% get_cells -hierarchical *inst_B/control_reg
% get_cells inst_A*control_reg
```

階層名の問題を回避するため、次のようにすることをお勧めします。

- **get_*** コマンドを **-hierarchical** オプションなしで使用します。
- RTL デザインに表示されるすべての階層レベルを / を使用して示します。

階層オプションを使用しない例

次のコマンドは、フラット ネットリストと階層ネットリストのどちらでも機能します。

```
% get_cells inst_A/inst_B/*_reg
% get_cells inst_*/inst_B/control_reg
```

注意 : 階層セルでも同様に、合成を実行するときに階層ピンに制約を設定しないでください。また、組み合わせロジックの演算子を接続するネットに制約を設定しないでください。これらは LUT に結合され、ネットリストからなくなる可能性があります。

推奨 : 制約を変更したら XDC ファイルを保存し、エラボレートされたデザインを読み込み直して、メモリ内の制約と XDC ファイルの制約が一致するようにしてください。合成後に、メモリ内の同じ合成 XDC を使用して合成済みデザインを読み込み、[Report Timing Summary] を使用してタイミング解析を実行します。

詳細は、付録 A 「その他のリソース」のリストから『Vivado Design Suite ユーザー ガイド : デザイン解析およびクロージャ テクニック』(UG906) を参照してください。

合成によりデザインが変換されているため、合成前の制約の一部は正しく適用されない可能性があります。この問題を解決するには、次を実行します。

1. 合成済みネットリストに適用する新しい XDC 構文を見つけます。
2. インプリメンテーションのみに使用する制約を新しい XDC ファイルに保存します。
3. 合成のみに使用する合成制約を別の XDC ファイルに移動します。

インプリメンテーション制約の作成

合成済みネットリストが生成されたら、インプリメンテーションに適用する XDC ファイルと共にメモリに読み込みます。次の目的でタイミング解析を実行できます。

- タイミング制約をネットリストの名前で修正し、インプリメンテーションのみの XDC ファイルに保存
- 非同期および排他的なクロック グループなどの欠けている制約を追加
- 複数サイクル パスや最大遅延制約などのタイミングの例外を追加
- パスが長いために違反が大きくなっているものを特定し、RTL 記述を修正

合成に使用したのと同じ基本制約を使用し、インプリメンテーション特定の新しい制約を保存する 2 つ目の XDC ファイルを作成できます。物理制約およびコンフィギュレーション制約を別の XDC ファイルに保存することも可能です。

図 2-11 に、ネットリスト ベースの XDC の作成手順を示します。

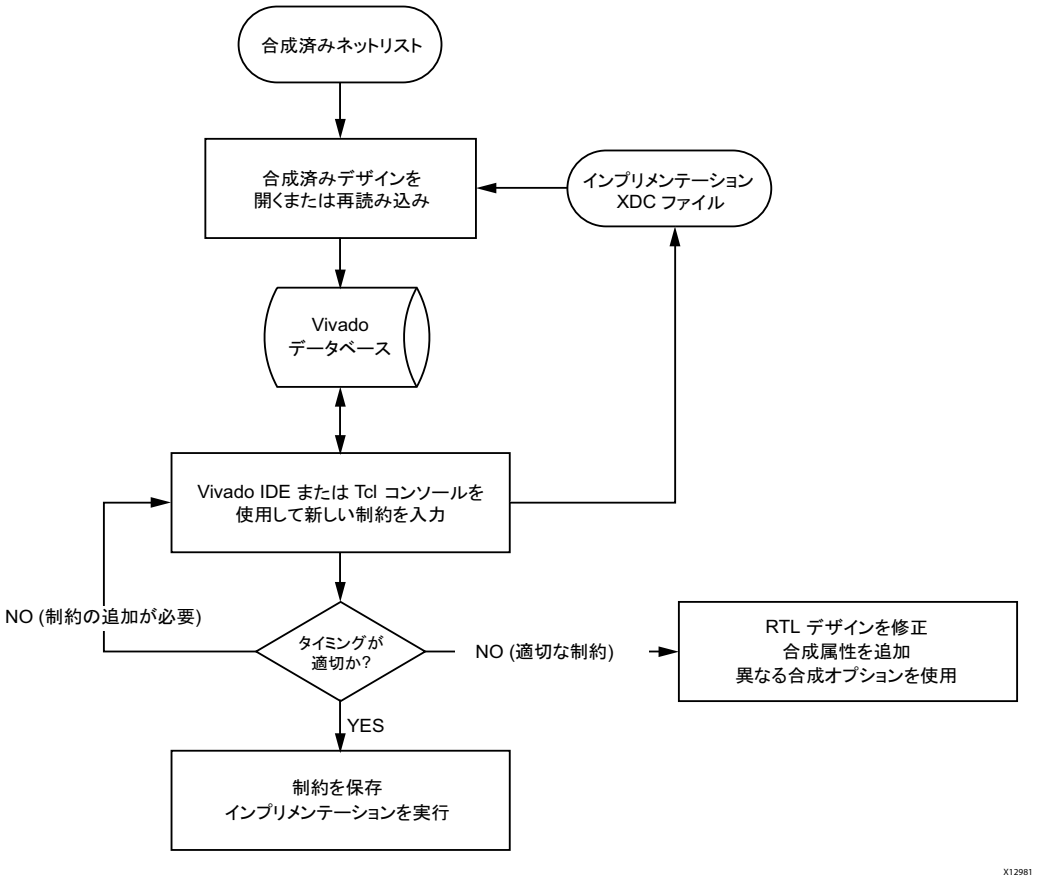

図 2-11 : 合成済みデザインでの制約の作成

インプリメンテーションに進む前に、大きなタイミング違反がないことを確認する必要があります。インプリメンテーションでは、最もクリティカルなパスのセルがお互いに近くに配置され、最速の配線リソースが使用されますが、大きい違反は解決できません。

推奨 : RTL を再確認して、違反のあるパスのロジック レベル数を削減したり、クロック ツリーを簡潔にして専用クロック リソースが使用されるようにして、関連するクロック間のスキューを最小限に抑えるようにしてください。合成属性を追加したり、異なる合成オプションを使用することも可能です。

詳細は、付録 A 「その他のリソース」のリストから『Vivado Design Suite ユーザー ガイド : 合成』(UG901) を参照してください

XILINX

第 3 章

タイミング解析

デザインにタイミング制約を追加する前に、タイミング解析の基本と用語を理解しておく必要があります。この章では、Vivado™ 統合設計環境 (IDE) のタイミング エンジンで使用される主要な概念を説明します。

タイミング パス

タイミング パスは、デザインのインスタンス間の接続に基づいて定義されます。デジタル デザインでは、タイミング パスは同じクロックまたは異なるクロックで制御される順次エレメントのペアで形成されます。

一般的なタイミング パス

デザインに含まれる最も一般的なパスは、次のとおりです。

- 入力ポートから内部シーケンシャル セルまでのパス
- シーケンシャル セル間の内部パス
- 内部シーケンシャル セルから出力ポートまでのパス
- 入力ポートから出力ポートまでのパス

入力ポートから内部シーケンシャル セルまでのパス

入力ポートから内部シーケンシャル セルまでのパスでは、データが次のように伝搬されます。

- デバイスの外部からポート クロックにより入力されます。
- 入力遅延 (SDC 定義) 後にデバイス ポートに到達します。
- デバイスの内部ロジックを介して、デスティネーション クロックが供給されるシーケンシャル セルに到達します。

シーケンシャル セル間の内部パス

シーケンシャル セル間の内部パスでは、データが次のように伝搬されます。

- デバイス内でソース クロックが供給されるシーケンシャル セルから駆動されます。
- 内部ロジックを介して、デスティネーション クロックが供給されるシーケンシャル セルに到達します。

内部シーケンシャル セルから出力ポートまでのパス

内部シーケンシャル セルから出力ポートまでのパスでは、データが次のように伝搬されます。

- デバイス内でソース クロックが供給されるシーケンシャル セルから駆動されます。
- 内部ロジックを介して出力ポートに到達します。
- 出力遅延 (SDC 定義) 後にポート クロックにより受信されます。

入力ポートから出力ポートまでのパス

入力ポートから出力ポートまでのパスでは、データが入力ポートからデバイス内でラッチされることなく出力ポートに伝搬されます。これらのパスは、通常 in-to-out データ パスと呼ばれます。ポート クロックとしては、仮想クロックまたはデザイン クロックを使用できます。

パスの例

図 3-1 に、この例では、デザイン クロック CLK0 を DIN および DOUT 遅延制約のポート クロックとして使用できます。

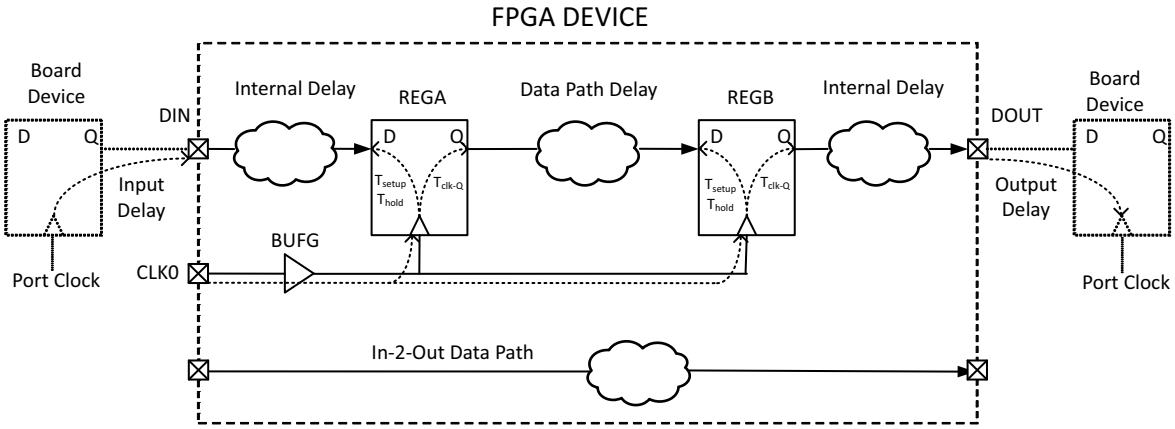

図 3-1 : パスの例

タイミング パスのセクション

各タイミング パスは、次の 3 つのセクションから構成されます。

- ソース クロック パス
- データ パス
- デスティネーション クロック パス

ソース クロック パス

ソース クロック パスは、ソース クロックがソース ポイント (通常は入力ポート) からソース シーケンシャル セルのクロック ピンまで伝搬されるパスです。入力ポートから開始するタイミング パスには、ソース クロック パスはありません。

データ パス

内部回路では、データ パスはソース シーケンシャル セルとデスティネーション シーケンシャル セルの間のパスです。

- ソース シーケンシャル セルのアクティブ クロック ピンは、パスの開始点と呼ばれます。
- デスティネーション シーケンシャル セルのデータ入力ピンは、パスの終点と呼ばれます。

入力ポート パスでは、データ パスは入力ポートから開始します。入力ポートがパスの開始点となります。

出力ポート パスでは、データ パスは出力ポートで終了します。出力ポートがパスの終点となります。

デスティネーション クロック パス

デスティネーション クロック パスは、デスティネーション クロックがソース ポイント (通常は入力ポート) からデスティネーション シーケンシャル セルのクロック ピンまで伝搬されるパスです。

出力ポートで終了するタイミング パスには、デスティネーション クロック パスはありません。

図 3-2 に、典型的なタイミング パスのこれら 3 つのセクションを示します。

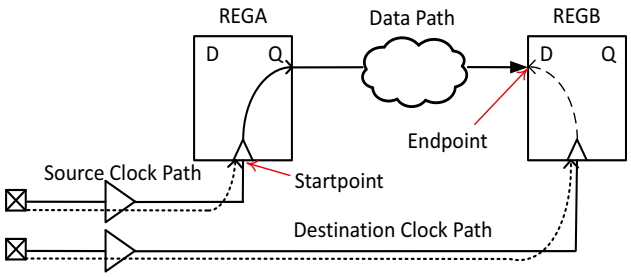

図 3-2 : 典型的なタイミング パス

セットアップおよびホールド解析

Vivado IDE では、タイミング パスの終点のスラックが解析およびレポートされます。スラックとは、データ所要時間とデータがパスの終点に到着する時間の差です。スラックが正の場合、パスはタイミングの面では正しく機能すると考えられます。

セットアップ チェック

セットアップ解析で使用するデータ所要時間を算出するため、タイミング エンジンは次を実行します。

1. ソース クロックとデスティネーション クロックの共通周期を検出します。共通周期が見つからない場合は、1,000 クロック サイクルまでが解析に使用されます。
2. 開始点クロックと終点クロックのすべての立ち上がりエッジと立ち下がりエッジを、共通周期の間調べます。
3. 2 つのアクティブ エッジの最小の正の差を特定します。この差が、セットアップ解析のタイミング パス要件になります。

セットアップ パス要件の例

異なるクロックの立ち上がりエッジで動作する 2 つのレジスタの間のパスを考えてみます。このパスのアクティブ クロック エッジは、立ち上がりエッジのみです。クロックは次のように定義されます。

- clk0 の周期は 6ns です。
- clk1 の周期は 4ns です。

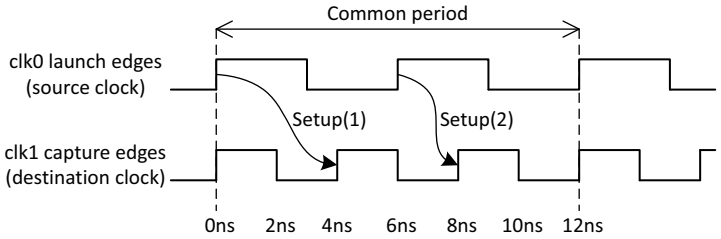

図 3-3 : セットアップ パス要件の例

図 3-3 では、セットアップ解析の基準となるソース クロック エッジおよびデスティネーション クロック エッジが 2 つあります (setup(1) および setup(2))。

clk0 から clk1 までの最小の正の差は 2ns で、これは setup(2) に対応します。

```
ソース クロック エッジ時間 : 0ns + 1 * T(clk0) = 6ns
デスティネーション クロック エッジ時間 : 0ns + 2 * T(clk1) = 8ns
セットアップ パス要件 = デスティネーション エッジ時間 - ソース エッジ時間 = 2ns
```

パス要件を算出する際、次の 2 つの重要な考慮事項があります。

1. クロック エッジは理想的なものであり、クロック ツリーの挿入遅延は考慮されていません。
2. デフォルトでは、位相シフトが設定されていなければ、クロックは時間 0 で位相が揃えられます。非同期クロックには、既知の位相関係がありません。非同期クロック間のパスの解析には、デフォルトの推定値が使用されます。非同期クロックの詳細は、次のセクションを参照してください。

セットアップ解析でのデータ所要時間

セットアップ解析でのデータ所要時間は、デスティネーション セルでデータを正しくキャプチャできるようにするために、データが安定した状態になっていなければならない時間です。この値は、次の要素に基づきます。

- デスティネーション クロック エッジ時間
- デスティネーション クロック遅延
- ソースおよびデスティネーション クロックのばらつき
- デスティネーション セルのセットアップ タイム

セットアップ解析でのデータ到着時間

セットアップ解析でのデータ到着時間は、データがソース クロックにより送信されてから、パスの終点で安定した状態になるまでにかかる時間です。この値は、次の要素に基づきます。

- ソース クロック エッジ時間
- ソース クロック遅延
- データパス遅延

データパス遅延には、開始点から終点までの、すべてのセルおよびネット遅延が含まれます。

Vivado IDE のタイミング レポートでは、セットアップ タイムがデータパスの一部として報告されます。そのため、データ到着時間およびデータ所要時間は、次の式で算出されます。

```
データ所要時間 (セットアップ) = デスティネーション クロック エッジ時間
                              + デスティネーション クロック パス遅延
                              - クロックのばらつき
データ到着時間 (セットアップ) = ソース クロック エッジ時間
                              + ソース クロック パス遅延
                              + データパス遅延
                              + セットアップ タイム
```

セットアップ スラックは、所要時間と到着時間の差です。

```
スラック (セットアップ) = データ所要時間 – データ到着時間
```

レジスタの入力データ ピンのセットアップ スラックが負の場合、レジスタに不明な値が取り込まれ、メタステーブル状態になる可能性があります。

ホールド チェック

ホールド スラックの算出は、セットアップ スラックの算出と直接関連しています。セットアップ解析は、最悪の条件でもデータが正しく受信されるかどうかを検証しますが、ホールド解析では次を検証します。

- 前のデスティネーション クロック エッジで同じデータが間違って受信されない。
- 次のソース クロック エッジで送信されたデータがセットアップ解析に使用されるデスティネーション クロック エッジで受信されない。

ホールド解析用のタイミング パス要件を算出するため、タイミング エンジンでセットアップ解析のソース クロック エッジおよびデスティネーション クロック エッジの可能な組み合わせすべてが考慮されます。

可能なセットアップ クロック エッジの組み合わせに対して、タイミング エンジンで対応するホールド解析エッジが検証されます。

- **delta(a)** : 前のデスティネーション クロック エッジから元のソース クロック エッジを引いた差
- **delta(b)** : 元のデスティネーション クロック エッジから次のソース クロック エッジを引いた差

すべての **delta(a)** および **delta(b)** 値のうち最大のものがホールド要件となり、対応するクロック エッジがホールド解析に使用されます。

ホールド パス要件の例

30 ページの「セットアップ パス要件の例」と同じクロックを考えます。セットアップ解析では、可能なエッジの組み合わせは 2 つのみです。

```
セットアップ パス要件 (S1) = 1*T(clk1) – 0*T(clk0) = 4ns
セットアップ パス要件 (S2) = 2*T(clk1) – 1*T(clk0) = 2ns
```

対応するホールド要件は次のとおりです。

```
セットアップ S1 :
  ホールド パス要件 (H1a) = (1-1)*T(clk1) - 0*T(clk0) = 0ns
  ホールド パス要件 (H1b) = 1*T(clk1) - (0+1)*T(clk0) = -2ns
セットアップ S2 :
  ホールド パス要件 (H2a) = (2-1)*T(clk1) - 1*T(clk0) = -2ns
  ホールド パス要件 (H2b) = 2*T(clk1) - (1+1)*T(clk0) = -4ns
```

最大ホールド要件は 0ns で、ソース クロックとデスティネーション クロック両方の最初の立ち上がりエッジに対応します。

図 3-4 に、セットアップ チェック エッジと関連するホールド チェックを示します。

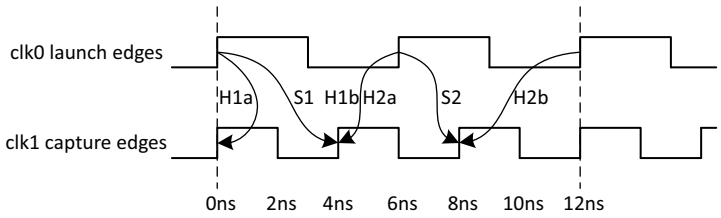

図 3-4 : ホールド パス要件の例

この例では、最終的なホールド要件は最も厳しいセットアップ要件から導かれたものではありません。これは、最も困難なホールド要件を見つけるために、すべての可能なセットアップ エッジが考慮されたからです。

セットアップ解析と同様、データ所要時間とデータ到着時間は、次の要素に基づいて算出されます。

- ソース クロック エッジ時間
- デスティネーション クロック エッジ時間
- ソースおよびデスティネーション クロック遅延
- クロックのばらつき
- データパス遅延
- デスティネーション レジスタのホールド タイム

データ所要時間 （ホールド） = デスティネーション クロック エッジ時間
+ デスティネーション クロック パス遅延
+ クロックのばらつき

データ到着時間 （ホールド） = ソース クロック エッジ時間
+ ソース クロック パス遅延
+ データパス遅延
- ホールド タイム

ホールド スラックは、所要時間と到着時間の差です。

スラック （ホールド） = データ到着時間 – データ所要時間

ホールド スラックが正の場合、最悪の条件でもデータが間違ったクロック エッジにより受信されることはありません。ホールド スラックが負の場合、間違ったデータが受信されたり、レジスタがメタステーブル状態になる可能性があります。

リカバリおよびリムーバル解析

リカバリおよびリムーバル タイミング チェックはセットアップおよびホールド チェックと似ていますが、セットやクリアなどの非同期データ ピンに適用されます。

非同期リセットを持つレジスタの場合、次のようになります。

- リカバリ時間は、新しいデータを安全に取り込むために必要な、非同期リセット信号が非アクティブ ステートにトグルされてから次のアクティブ クロック エッジまでの最小時間です。
- リムーバル時間は、非同期リセット信号を問題なく非アクティブ ステートにトグルできる、アクティブ クロック エッジからの最小時間です。

次に、これらのチェックでのスラックを算出する式を示します。

リカバリ チェック

次のものの算出方法を式で示します。

データ所要時間 (リカバリ) = デスティネーション クロック エッジ時間
+ デスティネーション クロック パス遅延
- クロックのばらつき

データ到着時間 (リカバリ) = ソース クロック エッジ時間
+ ソース クロック パス遅延
+ データパス遅延
+ リカバリ時間

スラック (リカバリ) = データ所要時間 – データ到着時間

リムーバル チェック

次のものの算出方法を式で示します。

データ所要時間 (リムーバル) = デスティネーション クロック エッジ時間
+ デスティネーション クロック パス遅延
+ クロックのばらつき

データ到着時間 (リムーバル) = ソース クロック エッジ時間
+ ソース クロック パス遅延
+ データパス遅延
- リムーバル時間

スラック (リムーバル) = データ到着時間 – データ所要時間

セットアップおよびホールド チェックと同様に、負のリカバリ スラックまたはリムーバル スラックは、レジスタがメタステーブル状態になり、デザインに不明な電気レベルが伝搬される可能性があることを示します。

XILINX

第 4 章

クロックの定義

クロックについて

デジタル デザインでは、クロックがレジスタからレジスタにデータを転送するための時間の基準となります。Vivado™ 統合設計環境 (IDE) では、タイミング エンジンはクロックの特性を使用して次を実行します。

- タイミング パス要件を算出
- スラックを算出してデザインのタイミング マージンをレポート

詳細は、第 3 章「タイミング解析」を参照してください。

タイミング パスを正確に最大限に網羅するため、クロックを正しく定義する必要があります。クロックは次の特性により定義されます。

- クロックは、そのツリー ルートのドライバー ピンまたはポート (ソース ポイント) で定義されます。
- クロック エッジは、周期と波形の特性で表現されます。
- 周期はナノ秒 (ns) で定義し、波形が繰り返す間隔に対応します。
- 波形は、クロック周期内の立ち上がりエッジおよび立ち下がりエッジの絶対時間 (ns) のリストです。波形のリストには偶数個の値を含める必要があり、最初の値は最初の立ち上がりエッジを示します。指定しない限り、デューティー サイクルはデフォルトで 50% になり、位相シフトは 0ns になります。

図 4-1 の例では、クロック Clk0 は周期が 10ns、デューティ サイクルが 50%、位相シフトが 0ns になり、クロック Clk1 は周期が 8ns、デューティ サイクルが 75%、位相シフトが 2ns になります。

```
Clk0: period = 10, waveform = {0 5}
Clk1: period = 8, waveform = {2 8}
```

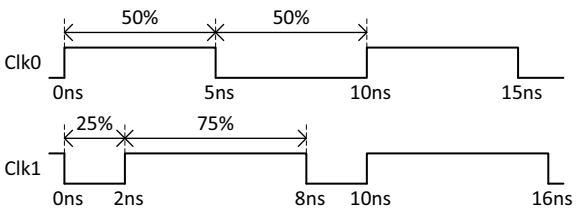

図 4-1 : クロック波形の例

伝搬されたクロック

周期および波形は、クロックの理想的な特性を表します。クロックが FPGA デバイスに入力され、クロック ツリーを介して伝搬されると、クロック エッジに遅延が発生し、ノイズおよびハードウェアの動作により変動する可能性があります。これらはクロック ネットワーク レイテンシおよびクロックのばらつきと呼ばれます。

クロックのばらつきには、次のものが含まれます。

- クロック ジッター
- 位相エラー
- その他指定したばらつき

Vivado IDE では、クロックはデフォルトでレイテンシおよびばらつきを含む伝搬されたクロックとして処理され、クロック ツリー挿入遅延およびばらつきを含む正確なスラック値が算出されます。

専用ハードウェア リソース

ザイリンクス FPGA デバイスの専用ハードウェア リソースにより、多数のデザイン クロックが効率的にサポートされます。これらのクロックは、通常ボード上の外部コンポーネントにより生成され、入力ポートからデバイスに供給されます。

クロックは、クロック調整ブロックと呼ばれる次のプリミティブでも生成できます。

- MMCM
- PLL
- BUFR

LUT やレジスタなどの通常のセルでクロックを変換することも可能です。

次のセクションに、クロックの生成元別にクロックを定義するのに最適な方法を説明します。

プライマリ クロック

プライマリ クロックは、次のいずれかから供給されるボード クロックです。

- 入力ポート
- ギガビット トランシーバーの出力ピン (再生されたクロックなど)

プライマリ クロックは、**`create_clock`** コマンドでのみ定義できます。

プライマリ クロックは、ネットリスト オブジェクトに接続する必要があります。このネットリスト オブジェクトは、すべてのクロック エッジの発信元となるポイントで、ここからクロックがクロック ツリーのダウンストリームに伝搬されます。Vivado IDE でスラック値の算出に使用されるクロック レイテンシおよびばらつきを求めるときに、プライマリ クロックのソース ポイントにより時間 0 が決まります。

重要 : Vivado IDE では、プライマリ クロックが定義されたポイントのアップストリームにあるセルからのクロック ツリー遅延は無視されます。デザインの中央にあるピンにプライマリ クロックを定義すると、タイミング解析では一部のレイテンシのみが使用されます。これは、このクロックがデザイン内のその他の関連クロックと通信している場合に、クロック間のスキュー値 (そして結果的にスラック値) が不正確となるので問題です。

まず、プライマリ クロックを定義する必要があります。プライマリ クロックは、多くのタイミング制約の基準となります。

プライマリ クロックの例

図 4-2 では、ボード クロックはポート sysclk からデバイスに入力され、入力バッファーおよびクロック バッファーを介してパス レジスタに到達します。

- 周期 : 10ns
- デューティ サイクル : 50%
- 位相シフト : なし

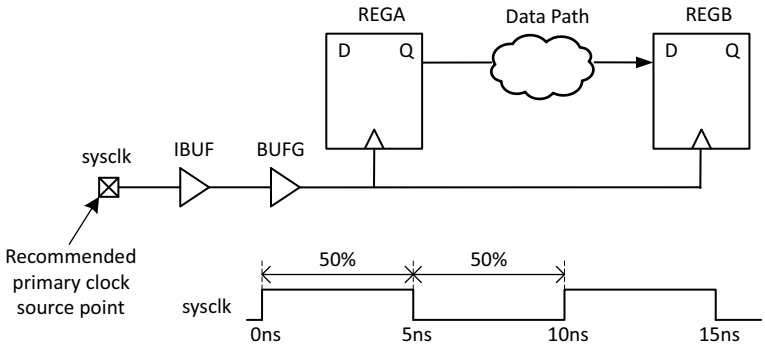

図 4-2 : プライマリ クロックの例

推奨 : ボード クロックは、クロック バッファーの出力ではなく入力ポートで定義します。

対応する XDC :

```
create_clock -period 10 [get_ports sysclk]
```

sysclk と同様に、ボード クロック devclk はポート ClkIn からデバイスに入力されます。

- 周期 : 10ns
- デューティ サイクル : 25%
- 位相シフト : 90 度

対応する XDC :

```
create_clock -name devclk -period 10 -waveform {2.5 5} [get_ports ClkIn]
```

図 4-3 では、トランシーバー gt0 はボード上の高速リンクからクロック rxclk を再生します。クロック rxclk は周期が 3.33ns、デューティ サイクルが 50% で、MMCM に入力されます。MMCM は、補正されたクロックを複数生成します。

GT0 の出力ドライバー ピン上に rxclk を定義する際、MMCM で駆動される生成されたクロックのソース ポイントはすべて gt0/RXOUTCLK となります。これらの間のパスのスラック算出では、適切なクロック レイテンシおよびばらつきの値が使用されます。

```
create_clock -name rxclk -period 3.33 [get_pins gt0/RXOUTCLK]
```

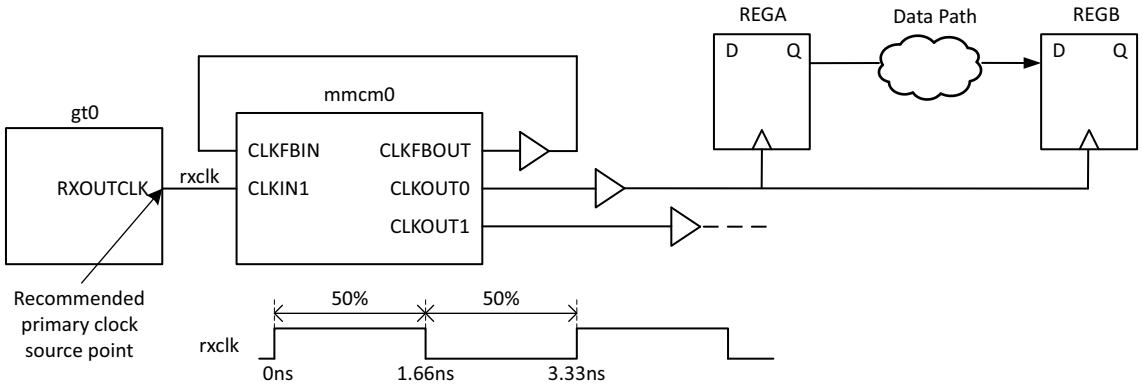

図 4-3 : GT プライマリ クロックの例

仮想クロック

仮想クロックは、デザインのどのネットリスト エレメントにも物理的に接続されていないクロックです。

仮想クロックを定義するには、**create_clock** コマンドをソースを指定せずに使用します。

一般的に、仮想クロックは次のような場合に入力遅延および出力遅延を定義するのに使用されます。

- 外部デバイス I/O の基準クロックがデザイン クロックのいずれでもない。
- FPGA I/O パスが内部で生成されたクロックに関連付けられており、そのクロックを派生したボード クロックを基準にして適切にタイミングを指定することができない。

  注記 : この状況は 2 つの周期の比が整数ではない場合に発生し、タイミング パス要件が非常に厳しくなったり、非現実的なものになったりすることがあります。
- 内部クロックの特性を変更せずに、I/O 遅延制約に関連するクロックに異なるジッターおよびレイテンシを変更する場合。

たとえば、クロック clk_virt は周期が 10ns で、ネットリスト オブジェクトに関連付けられていないとします。**[<objects>]** 引数は指定しません。この場合、**-name** の指定が必須になります。

```
create_clock -name clk_virt -period 10
```

入力および出力遅延制約で使用する前に、仮想クロックを定義する必要があります。

生成クロック

生成クロックには、次の 2 種類があります。

- ユーザー定義の生成クロック
- 自動的に派生したクロック

生成クロックについて

生成クロックは、MMCM などのクロック調整ブロックと呼ばれる特別なセルまたはユーザー ロジックにより駆動されます。

生成クロックは、マスター クロックに関連付けられています。次のクロックをマスター クロックとして使用できます。

- プライマリ クロック
- 別の生成クロック

生成クロックのプロパティは、マスター クロックから直接導出されます。周期または波形を指定する代わりに、マスター クロックが調整回路でどのように変換されるかを記述します。

マスター クロックと生成クロックの関係には、次のものを使用できます。

- 単純な周波数の分周
- 単純な周波数の逓倍
- 周波数の分周と逓倍の組み合わせて整数以外の比を生成 (通常 MMCM または PLL を使用)
- 位相シフトまたは波形の反転
- デューティ サイクルの変換
- 上記すべての組み合わせ

推奨 : まず、すべてのプライマリ クロックを定義してください。生成クロックを定義するには、プライマリ クロックが必要です。

ユーザー定義の生成クロック

ユーザー定義の生成クロックは、次のようなクロックです。

- **create_generated_clock** コマンドで定義されている。
- ネットリスト オブジェクト (理想的にはクロック ツリーのルート ピン) に接続されている。

-source オプションを使用してマスター クロックを指定してください。マスター クロックが伝搬されるピンまたはポートを指定します。一般的には、マスター クロックのソース ポイントまたは生成クロックのソース セルの入力クロック ピンが使用されます。

重要 : **-source** オプションでは、ピンまたはポート ネットリスト オブジェクトのみを指定できます。クロック オブジェクトは指定できません。

例 1 : 2 分周

プライマリ クロック clk1 の周期は 10ns です。このクロックはレジスタ REGA で 2 分周され、ほかのレジスタのクロック ピンを駆動します。このクロックを clkdiv2 と呼びます。

この生成クロックを指定する 2 つの例を次に示します。

```
create_clock -name clkin -period 10 [get_ports clkin]

# Option 1: master clock source is the primary clock source point
create_generated_clock -name clkdiv2 -source [get_ports clkin] -divide_by 2 \
  [get_pins REGA/Q]

# Option 2: master clock source is the REGA clock pin
create_generated_clock -name clkdiv2 -source [get_pins REGA/C] -divide_by 2 \
  [get_pins REGA/Q]
```

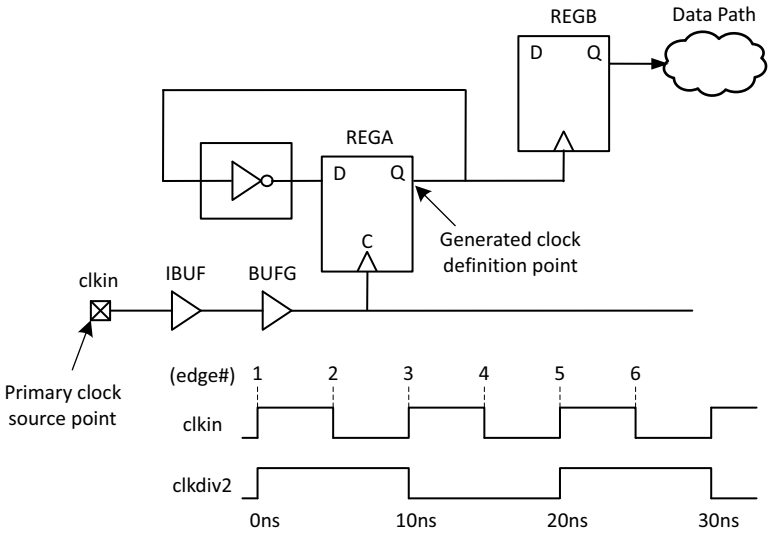

図 4-4 : 生成クロックの例 1

例 2 : –edges オプションを使用した 2 分周

-divide_by オプションの代わりに、**-edges** オプションを使用して、マスター クロックのエッジに基づいて生成クロックの波形を直接定義できます。引数はマスター クロック エッジのインデックスのリストで、生成クロックのエッジの時間位置を立ち上がりクロック エッジから開始して定義します。

次の例では、「例 1 : 2 分周」の生成クロックを **-edges** オプションを使用して定義しています。

```
# waveform specified with -edges instead of -divide_by
create_generated_clock -name clkdiv2 -source [get_pins REGA/C] -edges {1 3 5} \
  [get_pins REGA/Q]
```

例 3 : –edges および –edge_shift オプションを使用したデューティ サイクルおよび位相シフトの指定

生成クロック波形の各エッジは、**-edge_shift** オプションで正または負の値を指定して、個別にシフトできます。

-edge_shift オプションは、次のオプションと同時に使用することはできません。

- **-divide_by**
- **-multiply_by**
- **-invert**

マスター クロック clkin の周期が 10ns、デューティ サイクルが 50% だとします。このクロックはセル **mmcm0** に入力され、このセルによりデューティ サイクルが 25% で位相が 90 度シフトされたクロックが生成されます。生成クロックの定義には、マスター クロックのエッジ 1、2、および 3 が使用されています。これらのエッジは、それぞれ 0ns、5ns、および 10ns で発生します。適切な波形を得るには、1 番目と 3 番目のエッジを 2.5ns シフトします。

```
create_clock -name clkin -period 10 [get_ports clkin]
create_generated_clock -name clkshift -source [get_pins mmcm0/CLKIN] -edges {1 2 3} \
   -edge_shift {2.5 0 2.5} [get_pins mmcm0/CLKOUT]
# 最初の立ち上がりエッジ :     0ns + 2.5ns = 2.5ns
# 立ち下がりエッジ :           5ns + 0ns = 5ns
# 2 番目の立ち上がりエッジ : 10ns + 2.5ns = 12.5ns
```

注記 : **-edge_shift** の値は、正の場合と負の場合があります。

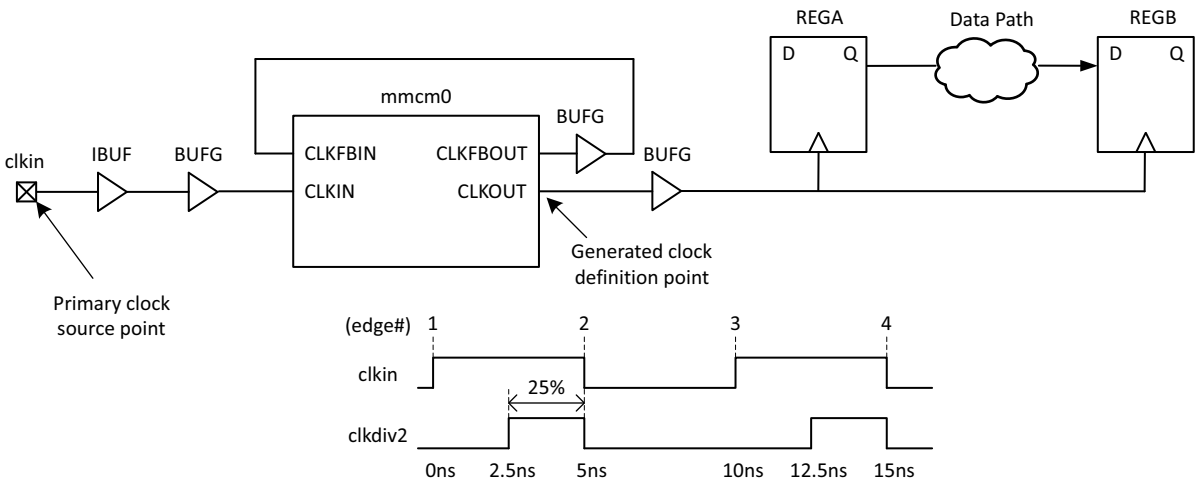

図 4-5 : 生成クロックの例 3

例 4 : –divide_by と –multiply_by オプションを同時に使用

Vivado IDE では、**-divide_by** と **-multiply_by** オプションを同時に指定できます。これは、標準 SDC のサポートを拡張したものです。これらのオプションを同時に指定すると、MMCM または PLL インスタンス で生成されたクロックを手動で定義する際に便利です。ただし、これらの制約はツールで自動的に作成されるようにすることをお勧めします。

詳細は、「自動的に派生したクロック」を参照してください。

たとえば、たとえば、「例 3 : –edges および –edge_shift オプションを使用したデューティ サイクルおよび位相シフトの指定」の **mmcm0** セルでマスター クロックの周波数が 4/3 で逓倍されるとすると、この生成クロックの定義は次のようになります。

```
create_generated_clock -name clk43 -source [get_pins mmcm0/CLKIN] -multiply_by 4 \
   -divide_by 3 [get_pins mmcm0/CLKOUT]
```

MMCM または PLL の出力に生成クロック制約を作成する場合は、波形の定義が MMCM または PLL のコンフィギュレーションに一致することを確認してください。

自動的に派生したクロック

自動的に派生したクロックは、自動生成クロックとも呼ばれます。関連のマスター クロックが既に定義されていれば、Vivado IDE によりクロック調整ブロック (CMB) の出力ピンに自動的に制約が作成されます。CMB には、MMCMx、PLLx、BUFR プリミティブがあります (MIG IP の PHASER_x を含む)。

同じ定義ポイントのネットリスト オブジェクト (ネットまたはピン) に既にユーザー定義クロック (プライマリまたは生成) が定義されている場合は、クロックは自動生成されません。自動生成クロックの名前は、定義ポイントに直接接続されているネットの名前に基づきます。

自動的に派生したクロックの例

次に、MMCM により生成されたクロックの例を示します。

マスター クロック clkin が MMCME2 インスタンス clkip/mmcm0 の入力 CLKIN を駆動するとすると、自動生成クロックの名前は cpuClk となり、その定義ポイントは clkip/mmcm0/CLKOUT となります。

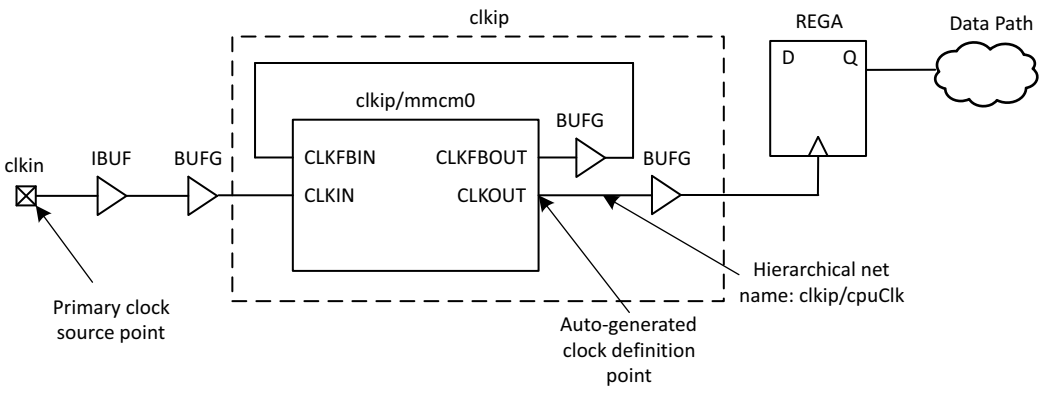

図 4-6 : 自動生成クロックの例

ローカル ネット名

CMB インスタンスがデザインの階層内に配置されている場合、生成クロックの名前にローカル ネット名 (親セル名を含まない名前) が使用されます。

たとえば、階層ネット名が clkip/cpuClk であるとすると、次のようになります。

- 親セル名は clkip です。
- 生成クロック名は cpuClk です。

名前の競合

2 つの生成クロックの名前が競合する場合、Vivado IDE で次のようにこれらを区別する接尾辞が付けられます。

- usrclk
- usrclk_1
- usrclk_2
- ...

生成クロックの名前を指定するには、次のいずれかの方法を使用します。

- RTL で固有のわかりやすいネット名を選択します。
- **`create_generated_clock`** を使用して生成クロック制約を定義します。

クロック グループ

Vivado IDE では、クロック グループ制約を使用して指定しない限り、すべてのクロックが関連していると想定されます。**`set_clock_groups`** コマンドを使用すると、クロック グループ間のタイミング解析がディスエーブルになります。

[Schematic] ビューまたは [Report Clock Networks] コマンドを使用してクロック ツリーのトポロジを表示し、どのクロックの関連性を保持する必要があるかを確認してください。また、[Report Clock Interaction] コマンドを使用して、2 つのクロック間の既存の制約を確認したり、同じプライマリ クロック (同じソース ポイント) を共有しているかを判断できます。

注意 : 2 つのクロック間のタイミング解析をディスエーブルにしても、それらのクロック間のパスがハードウェアで正しく機能するとは限りません。メタステーブル状態にならないようにするため、これらのパスに再同期化回路または非同期データ転送プロトコルがあることを確認してください。

同期クロック

2 つのクロックの位相関係が予測可能である場合、これらのクロックは同期しています。クロック ツリーがネットリストの同じルートから出発している場合、2 つのクロックは通常同期しています。

たとえば、生成クロックとそのマスター クロックは、生成クロックのソース ポイントまでは同じネットリスト リソースを伝搬されるので、同期しています。

非同期クロック グループ

2 つのクロックの位相関係を特定できない場合、これらのクロックは非同期です。たとえば、2 つのクロックがボード上の異なるオシレーターで生成され、異なる入力ポートから入力される場合、位相関係は不明なので、これらのクロックは非同期として扱う必要があります。2 つのクロックがボード上の同じオシレーターで生成されている場合は、非同期ではありません。

ほとんどの場合、プライマリ クロック同士は非同期として扱うことができます。プライマリ クロックとそれを基に生成されたクロックをまとめて、非同期クロック グループを構成できます。

非同期クロック グループの例

- プライマリ クロック clk0 が入力ポート上に定義され、MMCM に入力されてクロック usrclk および itfclk が生成されます。
- 2 番目のプライマリ クロック clk1 は GTP インスタンスの出力で定義される再生されたクロックで、2 番目の MMCM に入力されてクロック gtclkrx および gtclktx が生成されます。

非同期クロック グループの作成

非同期クロック グループを作成するには、**-asynchronous** オプションを使用します。

```
set_clock_groups -name async_clk0_clk1 -asynchronous -group {clk0 usrclk itfclk} \
  -group {clk1 gtclkrx gtclktx}
```

生成クロック名の取得

生成クロックの名前をあらかじめ予測できない場合は、**get_clocks -include_generated_clocks** を使用して取得します。**-include_generated_clocks** オプションは SDC からの拡張です。

上記の例は、次のようにも記述できます。

```
set_clock_groups -name async_clk0_clk1 -asynchronous \
  -group [get_clocks -include_generated_clocks clk0] \
  -group [get_clocks -include_generated_clocks clk1]
```

排他的なクロック グループ

デザインによっては、異なるクロックを使用するいくつかの動作モードがあります。この場合、通常クロックは次のエレメントを使用して選択されます。

- BUFGMUX および BUFGCTRL などのクロック マルチプレクサー
- LUT

推奨 : クロック ツリーでは LUT をできるだけ使用しないでください。

これらのセルは組み合わせセルであるため、すべての入力クロックは出力に伝搬されます。Vivado IDE では、1 つのクロック ツリーに複数のタイミング クロックを同時に存在させることができ、すべての動作モードに対するレポートを同時に生成できるので便利です。これはハードウェアでは不可能です。

このようなクロックは排他的クロックと呼ばれ、**set_clock_groups** で次のいずれかのオプションを使用して制約します。

- **-logically_exclusive**
- **-physically_exclusive**

排他的なクロック グループの例

MMCM インスタンスで clk0 および clk1 が生成され、BUFGMUX インスタンス clkmux に接続されています。clkmux の出力はデザインのクロック ツリーを駆動します。

clk0 と clk1 は同じクロック ツリーを共有していても同時に存在することはありませんが、デフォルトではこれらのクロック間のパスが解析されます。

これらのクロック間のパスの解析をディスエーブルにするには、次の制約を入力します。

```
set_clock_groups -name exclusive_clk0_clk1 -physically_exclusive \
  -group clk0 -group clk1
```

次のオプションは、ザイリンクス FPGA デバイスでは同等です。

- **-physically_exclusive**
- **-logically_exclusive**

physically および logically は、ASIC テクノロジでのさまざまなシグナル インテグリティ解析 (クロストーク) を表しており、ザイリンクス FPGA デバイスでは必要ありません。

クロック レイテンシ、ジッター、ばらつき

クロック波形の定義に加え、動作条件および環境による予測可能でランダムな変動を指定する必要があります。

クロック レイテンシ

クロック エッジは、ボードおよび FPGA デバイスを伝搬された後、ある遅延でデスティネーションに到達します。この遅延は、通常次のもので表されます。

- ソース レイテンシ (クロック ソース ポイント前、通常デバイス外部の遅延)
- ネットワーク レイテンシ

ネットワーク レイテンシによる遅延 (挿入遅延) の算出は、次のようになります。

- 自動的に予測 (配線前)
- 正確に算出 (配線後)

set_propagated_clock コマンドを使用すると、通常の SDC ツールでは伝搬遅延が算出されます。このコマンドは、Vivado IDE では次の理由から必要ありません。

- すべてのクロックが伝搬クロックとして処理されます。
- 生成クロックのレイテンシには、親プライマリ クロックの挿入遅延と、その生成クロックのネットワーク レイテンシが含まれます。

ザイリンクス FPGA では、**set_clock_latency** コマンドを使用してデバイス外部のクロック レイテンシを指定します。

set_clock_latency の例

```
# Minimum source latency value for clock sysClk (for both Slow and Fast corners)
set_clock_latency -source -early 0.2 [get_clocks sysClk]
# Maximum source latency value for clock sysClk (for both Slow and Fast corners)
set_clock_latency -source -late 0.5 [get_clocks sysClk]
```

クロック ジッターおよびクロックのばらつき

ASIC ではクロック ジッターは通常クロックのばらつき特性で表されますが、ザイリンクス FPGA デバイスではジッター特性は予測可能で、タイミング解析エンジンにより自動的に算出できるほか、別に指定することもできます。

入力ジッター

入力ジッターとは、標準または理想的なクロック到達時間と比較した連続クロック エッジ間のばらつきです。

各クロックの入力ジッターを個別に設定するには、**set_input_jitter** コマンドを使用します。入力ジッターは、マスター クロックからその派生クロックへは伝搬されません。このため、派生クロックにも入力ジッターを手動で指定する必要があります。

システム ジッター

システム ジッターは、次の要因による全体的なジッターです。

- 電源ノイズ
- ボード ノイズ
- システムのその他のジッター

set_system_jitter コマンドを使用して、デザイン全体 (すべてのクロック) に対して 1 つの値のみを設定します。

その他のクロックのばらつき

必要に応じて **set_clock_uncertainty** コマンドを使用し、異なるコーナー、遅延、特定のクロック関係に対するクロックのばらつきを定義します。これは、タイミングの観点から、デザインの部分にゆとりをもたせる効率的な方法です。

I/O 遅延

デザインの外部タイミングを正確に記述するには、入力ポートおよび出力ポートのタイミング情報を指定する必要があります。Vivado IDE では FPGA デバイス内のタイミング情報のみが認識されるので、デバイス外部に存在する遅延値は次のコマンドを使用して指定する必要があります。

- **set_input_delay**
- **set_output_delay**

入力遅延

set_input_delay コマンドを使用すると、入力ポートの入力パス遅延をデザインのインターフェイスでのクロック エッジに対して指定できます。アプリケーション ボードを考えた場合、この遅延は次のものの間の位相差を表します。

a. 外部チップからボードを介して FPGA デバイスの入力パッケージ ピンに伝搬されるデータ
b. 相対基準ボード クロック

このため、入力遅延は、デバイスのインターフェイスでのクロックおよびデータの相対位相によって、正または負の値になります。

入力遅延オプションの使用

-clock は、標準 SDC ではオプションですが、Vivado IDE では必須です。相対クロックは、デザイン クロックまたは仮想クロックのいずれかにできます。

推奨 : 仮想クロックを使用する場合は、デザイン内の入力ポートに接続されるデザイン クロックと同じ波形を使用してください。このようにすると、タイミング パス要件が現実的なものになります。仮想クロックを使用すると、デザイン クロックを変更せずに、さまざまなジッターまたはソース レイテンシのパターンを記述できます。

入力遅延コマンドには、次のオプションがあります。

- -min および -max オプション
- -clock_fall オプション
- -add_delay オプション

-min および -max オプション

-min および **-max** オプションは、次の解析用の値を指定します。

- 最小遅延解析 (ホールド/リムーバル)
- 最大遅延解析 (セットアップ/リカバリ)

どちらのオプションも使用しない場合、入力遅延値は最小値および最大値の両方に適用されます。

-clock_fall オプション

-clock_fall オプションを使用すると、相対クロックの立ち下がりクロック エッジで送信されるタイミング パスに入力遅延制約が適用されます。このオプションを使用しない場合、相対クロックの立ち上がりエッジのみが考慮されます。

-clock_fall オプションを **-rise** および **-fall** オプションと混同しないでください。**-rise** および **-fall** オプションは、クロック エッジではなくデータ エッジを参照します。

-add_delay オプション

-add_delay オプションは、次の両方の条件が満たされる場合に使用する必要があります。

- 最大 (または最小) 入力遅延制約が存在する。
- 2 番目の最大 (または最小) 入力遅延制約を指定する必要がある。

このオプションは通常、DDR インターフェイスのように、複数のクロック エッジに対して入力ポートに遅延制約を設定する場合に使用されます。

入力遅延制約は、入力ポートまたは双方向ポートに適用できます。ただし、クロック入力ポートには適用できず、これらは自動的に無視されます。内部ピンには適用できません。

入力遅延の例 1

次の例では、前に定義した sysClk に対して、最小遅延解析と最大遅延解析の両方に使用する入力遅延を定義します。

```
> create_clock -name sysClk -period 10 [get_ports CLK0]
> set_input_delay -clock sysClk 2 [get_ports DIN]
```

入力遅延の例 2

次の例では、前に定義した仮想クロックに対して入力遅延を定義します。

```
> create_clock -name clk_port_virt -period 10
> set_input_delay -clock clk_port_virt 2 [get_ports DIN]
```

入力遅延の例 3

次の例では、sysClk に対して最小遅延解析と最大遅延解析に異なる入力遅延値を定義します。

```
> create_clock -name sysClk -period 10 [get_ports CLK0]
> set_input_delay -clock sysClk -max 4 [get_ports DIN]
> set_input_delay -clock sysClk -min 1 [get_ports DIN]
```

入力遅延の例 4

次の例では、DDR クロックに対して入力遅延値を定義します。

```
> create_clock -name clk_ddr -period 6 [get_ports DDR_CLK_IN]
> set_input_delay -clock clk_ddr -max 2.1 [get_ports DDR_IN]
> set_input_delay -clock clk_ddr -max 1.9 [get_ports DDR_IN] -clock_fall -add_delay
> set_input_delay -clock clk_ddr -min 0.9 [get_ports DDR_IN]
> set_input_delay -clock clk_ddr -min 1.1 [get_ports DDR_IN] -clock_fall -add_delay
```

この例では、デバイス外部から clk_ddr クロックの立ち上がりエッジと立ち下がりエッジの両方で送信されるデータから、立ち上がりクロック エッジおよび立ち下がりクロック エッジの両方で動作する内部フリップフロップのデータ入力までの制約が作成されます。

出力遅延

set_output_delay コマンドを使用すると、出力ポートの出力パス遅延をデザインのインターフェイスでのクロック エッジに対して指定できます。

アプリケーション ボードを考えた場合、この遅延は次のものの間の位相差を表します。

a. FPGA デバイスの出力パッケージ ピンからボードを介して別のデバイスに伝搬されるデータ
b. 相対基準ボード クロック

出力遅延は、FPGA デバイス外部のクロックおよびデータの相対位相によって、正または負の値になります。

出力遅延オプションの使用

-clock は、標準 SDC ではオプションですが、Vivado IDE では必須です。

相対クロックは、デザイン クロックまたは仮想クロックのいずれかにできます。

推奨 : 仮想クロックを使用する場合は、デザイン内の出力ポートに接続されるデザイン クロックと同じ波形を使用してください。このようにすると、タイミング パス要件が現実的なものになります。仮想クロックを使用すると、デザイン クロックを変更せずに、さまざまなジッターまたはソース レイテンシのパターンを記述できます。

出力遅延コマンドには、次のオプションがあります。

- -min および -max オプション
- -clock_fall オプション
- -add_delay オプション

-min および -max オプション

-min および **-max** オプションを使用すると、最小遅延解析 (ホールド/リムーバル) および最大遅延解析 (セットアップ/リカバリ) に対してさまざまな値を指定できます。どちらのオプションも使用しない場合、入力遅延値は最小値および最大値の両方に適用されます。

-clock_fall オプション

-clock_fall オプションを使用すると、相対クロックの立ち下がりクロック エッジで受信されるタイミング パスに出力遅延制約が適用されます。このオプションを使用しない場合、デバイス外部の相対クロックの立ち上がりエッジのみが考慮されます。

-clock_fall オプションを **-rise** および **-fall** オプションと混同しないでください。**-rise** および **-fall** オプションは、クロック エッジではなくデータ エッジを参照します。

-add_delay オプション

-add_delay オプションは、次の両方の条件が満たされる場合に使用する必要があります。

1. 最大出力遅延制約が存在する。
2. 2 番目の最大出力遅延制約を指定する必要がある。

最小出力遅延制約の場合も同様に使用する必要があります。このオプションは通常、DDR インターフェイスの立ち上がりおよび立ち下がりエッジや、異なるクロックを使用する複数のデバイスに出力ポートが接続されている場合など、複数のクロック エッジに対して出力ポートを制約する場合に使用されます。

重要 : 出力遅延制約は、出力ポートまたは双方向ポートにのみ適用できます。内部ピンには適用できません。

出力遅延の例 1

次の例では、前に定義した sysClk に対して、最小遅延解析と最大遅延解析の両方に使用する出力遅延を定義します。

```
> create_clock -name sysClk -period 10 [get_ports CLK0]
> set_output_delay -clock sysClk 6 [get_ports DOUT]
```

出力遅延の例 2

次の例では、前に定義した仮想クロックに対して出力遅延を定義します。

```
> create_clock -name clk_port_virt -period 10
> set_output_delay -clock clk_port_virt 6 [get_ports DOUT]
```

出力遅延の例 3

次の例では、DDR クロックに対して、出力遅延を最小遅延 (ホールド) および最大遅延 (セットアップ) 解析で異なる値に指定します。

```
> create_clock -name clk_ddr -period 6 [get_ports DDR_CLK_IN]
> set_output_delay -clock clk_ddr -max 2.1 [get_ports DDR_OUT]
> set_output_delay -clock clk_ddr -max 1.9 [get_ports DDR_OUT] -clock_fall
-add_delay
> set_output_delay -clock clk_ddr -min 0.9 [get_ports DDR_OUT]
> set_output_delay -clock clk_ddr -min 1.1 [get_ports DDR_OUT] -clock_fall
-add_delay
```

この例では、デバイス外部から clk_ddr クロックの立ち上がりエッジと立ち下がりエッジの両方で送信されるデータから、立ち上がりクロック エッジおよび立ち下がりクロック エッジの両方で動作する内部フリップフロップのデータ入力までの制約が作成されます。

XILINX

第 5 章

タイミング例外

ロジックがデフォルトのままでは正しいタイミングで動作しない場合、タイミング例外を指定する必要があります。2 クロック サイクルごとに結果を受信するロジックなど、タイミングを別に処理する必要がある場合は、タイミング例外コマンドを使用する必要があります。

Vivado™ 統合設計環境 (IDE) では、表 5-1 「タイミング例外コマンド」に示すタイミング例外コマンドがサポートされています。

表 5-1 : タイミング例外コマンド

コマンド	機能
set_multicycle_path	パスの開始点から終点までデータを伝搬させるのに必要なクロック サイクル数を指定します。
set_false_path	デザインに含まれているロジック パスで、解析から除外すべきものを指定します。
set_max_delay **set_min_delay**	最小パス遅延または最大パス遅延の値を指定します。このコマンドを使用すると、デフォルトのセットアップおよびホールド制約ではなく、ユーザーが指定した最大/最小遅延値が使用されます。
set_case_analysis	ポートまたはピンのロジック定数またはロジック遷移を使用して伝搬される信号を制限し、タイミング解析を実行します。

最小/最大遅延制約

set_min_delay および **set_max_delay** コマンドは、次の目的で使用できます。

- in-to-out I/O パスなど、特別なパスを制約
- 通常クロックで定義される選択されたパスのデフォルト パス要件を変更

表 5-2 : 最小/最大遅延制約

制約	機能
最小遅延	ホールドおよびリムーバル解析に使用されるパス要件に対応
最小遅延	セットアップおよびリカバリ解析に使用されるパス要件に対応

スラックの算出には、次の場合を除き、デフォルトでパスのクロック スキューが含まれます。

- 制約の開始点が適切でないためにパスが分割され、デスティネーション クロックのみが考慮される場合。
- **set_max_delay** コマンドでのみサポートされる **-datapath_only** オプションが使用され、ソース クロックとデスティネーション クロックの両方が無視される場合。詳細は、次のセクションを参照してください。

コマンド オプション

-datapath_only オプションは、**set_max_delay** コマンドでのみサポートされます。

オプションは、通常表 5-3 「コマンド オプション」に示すように使用します。

表 5-3 : コマンド オプション

オプション	適用箇所
• **-from** • **-rise from** • **-fall_from**	入力または双方向ポート、シーケンシャル セルのクロック ピン、クロックなどの有効な開始点
• **-to** • **-rise_to** • **-fall_to**	出力または双方向ポート、シーケンシャル セルの入力データ ピン、クロックなどの有効な終点
• **-through** • **-rise_through** • **-fall_through**	任意のネットまたはピン

set_max_delay をパスに使用しても、同じパスの最小遅延解析に影響はありません。同様に、**set_min_delay** も最大遅延解析には影響しません。同じパスの最小遅延または最大遅延要件を削除するには、フォルス パス制約を使用する必要があります。

たとえば、次のコマンドは、最大遅延解析 (セットアップ チェック) 用にパス要件を 5ns に設定し、同じパスを最小遅延解析 (ホールド チェック) ではディスエーブルにしています。

```
> set_max_delay -from [get_pins FD1/C] -to [get_pins FD2/D] 5
> set_false_path -from [get_pins FD1/C] -to [get_pins FD2/D] -hold
```

-datapath_only オプション

-datapath_only オプションを使用すると、**set_max_delay** の使用をさらに制限できます。**-from** オプションを必ず使用する必要があります。

-datapath_only オプションの例 1

data1_0_reg/Q が data12_0_reg に直接接続されている場合を示します。この場合は、**-from**/**-to** オプションを使用できます。

```
> set_max_delay -from data1_0_reg/C -to data12_0_reg/D 5 -datapath_only
```

パス分割

set_max_delay または **set_min_delay** の **-from** オプションで有効な開始点以外のピンを指定した場合、そのピンを通過するタイミング パスが分割され、これらのピンが開始点として使用されます。

このコマンドで無効な終点を指定した場合も、指定したピンが終点となるので、それらのピンを通過するタイミング パスが分割されます。これは問題となります。パスが分割されると、分割されたパスのクロックの伝搬が中断され、これらのパスのスキューが大きくなってしまいます。これは、制約で指定した最小および最大遅延値に含める必要があります。**set_max_delay** または **set_min_delay** コマンドを不適切に使用してパスが分割された場合も、最小遅延解析および最大遅延解析の両方に影響します。

推奨 : パスが分割されないようにするため、有効な開始点と終点を注意して選択してください。

XILINX

第 6 章

XDC の優先順位

ザイリンクス デザイン制約 (XDC) の優先順位は、Synopsys Design Constraints (SDC) と同じです。この章では、制約の競合および重複がどのように解決されるかを説明します。

XDC 制約の順序

XDC 制約は順次解釈されるコマンドで、優先順位が同じ場合は、後のものが優先されます。

制約順序の例

```
> create_clock -name clk1 -period 10 [get_ports clk_in1]
> create_clock -name clk2 -period 11 [get_ports clk_in1]
```

この例では、次の理由で最初のクロック定義が 2 つ目のクロック定義で置き換えられます。

- どちらも同じ入力に設定されています。
- create_clock –add オプションは使用されていません。

例外の優先順位

複数のタイミング例外が同じパスに適用されているなど、制約が重複する場合は、優先順位は次のようになります。

1. クロック グループ (**set_clock_groups**)
2. フォルス パス (**set_false_path**)
3. 最大/最小遅延パス (**set_max_delay/set_min_delay**)
4. 複数サイクル パス (**set_multicycle_path**)

注記 : 同じ例外の場合は、制約が詳細に指定されているものほど優先されます。

例外の優先順位の例

```
> set_max_delay 12 -from [get_clocks clk1] -to [get_clocks clk2]
> set_max_delay 15 -from [get_clocks clk1]
```

この例では、clk1 から clk2 へのパスに対しては、2 つ目の制約ではなく 1 つ目の制約が適用されます。

タイミング例外を指定するのに使用されるオブジェクト タイプやフィルター オプションも優先順位に関係します。オプションの優先順位は、次のようになります。

1. **-from pin**
2. **-to pin**
3. **-through pin**
4. **-from clock**
5. **-to clock**

XILINX

第 7 章

物理制約

Vivado™ 統合設計環境 (IDE) では、デザイン オブジェクトの物理制約はオブジェクト プロパティを設定することにより指定します。次に例を示します。

- ロケーションおよび I/O 規格などの I/O 制約
- セル ロケーションなどの配置制約
- 固定配置などの配線制約
- コンフィギュレーション モードなどのコンフィギュレーション制約

制約の適用

制約は、次の方法で適用できます。

- XDC 制約ファイルからの制約の適用
- Tcl コマンドを使用した制約の適用

XDC 制約ファイルからの制約の適用

XDC 制約ファイルを使用すると、制約はネットリストが処理されるときにファイルに記述されている順序で適用されます。

Tcl コマンドを使用した制約の適用

Tcl コマンドまたは Tcl スクリプトを使用すると、制約は次のように適用されます。

- メモリ内のデザイン オブジェクトにすぐに適用されます。
- XDC **set_property** コマンドを使用して適用されます。

次の構文を使用します。

```
set_property <property> <value> <object list>
```

LOC プロパティを使用したロケーション制約の例

```
set_property LOC SLICE_X32Y49 [get_cells XCOUNTER/BU5]
```

クリティカル警告

デザインに存在しないオブジェクトに適用されている制約など、XDC ファイルの無効な制約に対しては、クリティカル警告が表示されます。

推奨 : デザインを適切に制約するため、クリティカル警告をすべて確認することをお勧めします。無効な制約をインタラクティブに適用すると、エラーが発生します。

制約の定義および使用方法は、付録 A 「その他のリソース」にリストされている『Vivado Design Suite 制約リファレンス ガイド』(UG912) を参照してください。

ネットリスト制約

ネットリスト制約は、ポート、ピン、ネット、セルなどのネットリスト オブジェクトに設定され、コンパイル ツールで特別な方法で処理される必要があります。

- CLOCK_DEDICATED_ROUTE

  ネットまたはピンに設定し、クロック信号をどのように配線するかを指定します。

- MARK_DEBUG

  RTL のネットに設定し、保持してネットリストに含まれるようにします。これにより、コンパイル フローの任意の時点で、ネットをロジック デバッグ ツールに接続できます。

- DONT_TOUCH

  セルまたは階層インスタンスに設定し、ネットリスト最適化中に保持されるようにします。

I/O 制約

I/O 制約は、次のものを設定します。

- ポート
- ポートに接続されているセル

一般的な制約は、次のとおりです。

- I/O 規格
- I/O ロケーション

Vivado 統合設計環境 (IDE) では、ISE® Design Suite の I/O 制約の多くがサポートされています。

- DRIVE

  出力バッファーの駆動電流を mA で指定します。一部の I/O 規格でのみ使用可能です。

- IOSTANDARD

  I/O バッファーに I/O 規格を設定します。

- SLEW

  デバイス出力のスルー レート (遷移レート) を設定します。

- IN_TERM

  入力ポートの入力終端抵抗のコンフィギュレーションを設定します。

- OUT_TERM

  出力ポートの出力終端抵抗のコンフィギュレーションを設定します。

- DIFF_TERM

  IBUFDS_DIFF_OUT などのプリミティブに対して 100 オームの差動終端のオン/オフを指定します。

- KEEPER

  トライステート出力または双方向ポートにウィーク ドライバーを適用し、駆動されていないときに値を保持します。

- PULLDOWN

  トライステート出力または双方向ポートにウィーク Low を適用し、フローティングしないようにします。

- PULLUP

  トライステート出力または双方向ポートにウィーク High を適用し、フローティングしないようにします。

- DCI_VALUE

  IBIS ファイルを生成するときに IOB エレメントに関連付けるバッファーのビヘイビアー モデルを指定します。

- DCI_CASCADE

  マスターおよびスレーブ バンクのセットを定義します。DCI 基準電圧は、マスター バンクからスレーブにチェーン接続されます。

- INTERNAL_VREF

  I/O バンクの Vref ピンを解放し、代わりに内部で生成された Vref を使用します。

- IODELAY_GROUP

  IDELAY および IODELAY セルのセットを IDELAYCTRL とグループにし、デザインの IDELAYCTRL が自動的に複製および配置されるようにします。

- IOBDELAY

  IDELAY または IODELAY 遅延ライン セルのタップ遅延値を設定します。

- IOB

  フリップフロップおよびラッチをスライス ファブリックではなく I/O ロジックに配置するよう試みます。

制約の詳細は、付録 A 「その他のリソース」のリストから『Vivado Design Suite プロパティ リファレンス ガイド』(UG912) を参照してください。

例

プロパティは、Tcl リストを使用して 1 つまたは複数のオブジェクトに設定できます。

次の例では、I/O 規格を mode0 および mode1 ポートに設定しています。

```
% set_property IOSTANDARD LVCMOS18 [get_ports {mode0 mode1}]
```

配置制約

配置制約はセルに適用し、デバイス内でのロケーションを制御します。Vivado IDE では、ISE Design Suite および PlanAhead™ ツールの配置制約の多くがサポートされています。

- LUTNM

  2 つの LUT に固有の名前を指定し、1 つの LUT サイトに配置します。

- HLUTNM

  同じ階層にある 2 つの LUT に固有の名前を指定し、1 つの LUT サイトに配置します。

- PROHIBIT

  サイトへの配置を禁止します。

- PBLOCK

  論理ブロックに設定し、FPGA の物理領域に制約します。

- PACKAGE_PIN

  ターゲット デバイス パッケージのピンのデザイン ポートのロケーションを指定します。

- LOC

  ネットリストの論理エレメントをデバイス上のサイトに配置します。

- BEL

  ネットリストの論理エレメントをデバイス上のスライス内の特定の BEL に配置します。

配置タイプ

次の 2 種類の配置があります。

- 固定配置
- 固定されていない配置

固定配置

固定配置は、次の方法でユーザーが指定する配置です。

- 手動で配置
- XDC 制約
- メモリに読み込まれているデザインのセル オブジェクトに次を使用して指定
  - IS_LOC_FIXED
  - IS_BEL_FIXED

固定されていない配置

固定されていない配置は、インプリメンテーション ツールで実行される配置です。配置が固定されていると、制約されてるセルはインプリメンテーションで移動できません。固定配置は、単純な LOC または BEL として XDC ファイルに保存されます。

- IS_LOC_FIXED

  LOC 制約を固定されていないものから固定されたものに変更します。

- IS_BEL_FIXED

  BEL 制約を固定されていないものから固定されたものに変更します。

配置制約の例 1

次の例では、ブロック RAM を RAMB18_X0Y10 に配置し、固定します。

```
% set_property LOC RAMB18_X0Y10 [get_cells u_ctrl0/ram0]
```

配置制約の例 2

次の例では、LUT をスライスの C5LUT BEL に配置し、その BEL 割り当てを固定します。

```
% set_property BEL C5LUT [get_cells u_ctrl0/lut0]
```

配置制約の例 3

次の例では、入力遅延を短くするため入力バス レジスタを ILOGIC セルに配置します。

```
% set_property IOB TRUE [get_cells mData_reg*]
```

配置制約の例 4

次の例では、2 つの小型の LUT を、O5 および O6 出力の両方を使用する 1 つの LUT6_2 に結合します。

```
% set_property LUTNM L0 [get_cells {u_ctrl0/dmux0 u_ctrl0/dmux1}]
```

配置制約の例 5

次の例では、ブロック RAM の最初の列が使用されないようにします。

```
% set_property PROHIBIT TRUE [get_sites {RAMB18_X0Y* RAMB36_X0Y*}]
```

配線制約

配線制約はネット オブジェクトに適用し、配線リソースを制御します。

ピン固定

LOCK_PINS はセル プロパティで、論理 LUT 入力 (I0、I1、I2、…) と LUT 物理入力ピン (A6、A5、A4、…) の間のマップを指定します。

通常、タイミング クリティカル LUT 入力を高速の A6 および A5 物理 LUT 入力にマップするために使用されます。

LOCK_PINS 制約の例 1

次の例では、I1 を A6 に、I0 を A5 にマップしています (デフォルトのマップをスワップ)。

```
% set myLUT2  [get_cell u0/u1/i_365]
% set_property LOCK_PINS {I0:A5 I1:A6} $myLUT2
# Which you can verify by typing the following line in the Tcl Console:
% get_property LOCK_PINS $myLUT2
```

LOCK_PINS 制約の例 2

次の例では、LUT6 の I0 を A6 にマップしています。I1 ～ I5 のマップは固定されません。

```
% set_property LOCK_PINS I0:A6 [get_cell u0/u1/i_768]
```

固定配線

固定配線は、ISE の指定配線と同様に、配線を固定します。ネットの配線リソースの固定には、3 つのネット プロパティが関係します。次の表を参照してください。

表 7-1 : ネット プロパティ

プロパティ	機能
ROUTE	読み取り専用のネット プロパティ
IS_ROUTE_FIXED	配線全体を固定するようマーク
FIXED_ROUTE	ネットの固定配線部分

ネットの配線が確実に固定されるようにするには、その配線のすべてのセルを固定しておく必要があります。

次に、配線を完全に固定する例を示します。この例では、図 7-1 「配線制約の例を示すためのシンプルなデザイン」のデザインのネット a (青色でハイライト) の配線を固定する制約を作成します。

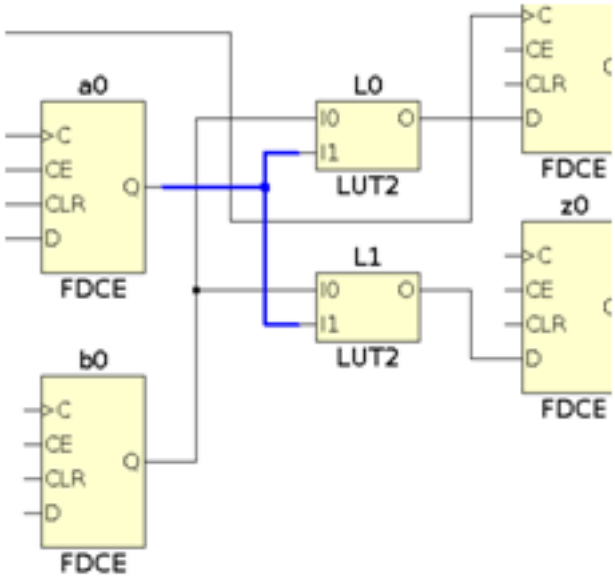

図 7-1 : 配線制約の例を示すためのシンプルなデザイン

インプリメント済みデザインをメモリに読み込むと、ネットの配線情報をクエリできるようになります。

```
% set net [get_nets a]
% get_property ROUTE $net
{ CLBLL_LL_CQ CLBLL_LOGIC_OUTS6 FAN_ALT5 FAN_BOUNCE5  { IMUX_L17 CLBLL_LL_B3 }  IMUX_L11
CLBLL_LL_A4 }
```

配線は、配線ノード名のリストで示され、ファンアウトは中かっこ ({ }) で示されます。配線を固定するには、次のコマンドを使用します。

```
% set_property IS_ROUTE_FIXED $net
```

制約を今後のために XDC ファイルにバックアノテートするには、固定されたネットに接続されているすべてのセルの配置も保持する必要があります。この情報は、[Schematic] または [Device] ビューでセルを選択し、[Properties] ビューの LOC/BEL プロパティ値で確認するか、Tcl コンソールでこれらの値をクエリします。

```
% get_property LOC [get_cells {a0 L0 L1}]
SLICE_X0Y47 SLICE_X0Y47 SLICE_X0Y47
% get_property BEL [get_cells {a0 L0 L1}]
SLICEL.CFF SLICEL.A6LUT SLICEL.B6LUT
```

固定された配線はタイミング クリティカルであることが多いので、LUT ピンのマップも LOCK_PINS 制約で指定し、配線時にピンがスワップされないようにする必要があります。スワップされると、固定された配線とネットのピンが揃わなくなります。

```
% get_site_pins -of [get_pins {L0/I1 L0/I0}]
SLICE_X0Y47/A4 SLICE_X0Y47/A2
% get_site_pins -of [get_pins {L1/I1 L1/I0}]
SLICE_X0Y47/B3 SLICE_X0Y47/B2
```

ネット a の配線を固定するために必要な XDC 制約は、次のようになります。

```
set_property LOC SLICE_X0Y47 [get_cells {a0 L0 L1}]
set_property BEL CFF [get_cells a0]
set_property BEL A6LUT [get_cells L0]
set_property BEL B6LUT [get_cells L1]
set_property LOCK_PINS {I1:A4 I0:A2} [get_cells L0]
set_property LOCK_PINS {I1:A3 I0:A2} [get_cells L1]
set_property FIXED_ROUTE { CLBLL_LL_CQ CLBLL_LOGIC_OUTS6 FAN_ALT5 FAN_BOUNCE5  { IMUX_L17
CLBLL_LL_B3 }  IMUX_L11 CLBLL_LL_A4 } [get_nets a]
```

XDC ではなく Tcl コマンドを使用する場合は、次の制約を 1 つの **place_cell** コマンドで設定できます。

- LOC
- BEL
- IS_LOC_FIXED
- IS_BEL_FIXED

```
place_cell a0 SLICE_X0Y47/CFF L0 SLICE_X0Y47/A6LUT L1 SLICE_X0Y47/B6LUT
```

place_cell コマンドの詳細は、付録 A「その他のリソース」のリストから『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835) を参照してください。

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、ビットストリーム生成用のグローバル制約で、現在のデザインに適用します。コンフィギュレーション モードなど制約が含まれます。

コンフィギュレーション制約の例 1

次の例では、CONFIG_MODE を M_SELECTMAP に設定しています。

```
% set_property CONFIG_MODE M_SELECTMAP [current_design]
```

コンフィギュレーション制約の例 2

次の例では、デバイス ピン E11 および F11 を電圧基準ピンに設定しています。

```
% set_property VREF {E11 F11} [current_design]
```

コンフィギュレーション制約の例 3

次の例では、CRC チェックをディスエーブルにしています。

```
% set_property BITSTREAM.GENERAL.CRC Disable [current_design]
```

ビットストリーム生成プロパティおよび定義のリストは、付録 A「その他のリソース」のリストから『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835) を参照してください。

XILINX

付録 A

その他のリソース

ザイリンクス リソース

アンサー、資料、ダウンロード、フォーラムなどのサポート リソースは、次のザイリンクス サポート サイトを参照してください。

http://japan.xilinx.com/support

ザイリンクス資料で使用される用語集は、次を参照してください。

http://japan.xilinx.com/company/terms.htm.

ソリューション センター

デバイス、ツール、IP のサポートについては、ザイリンクス ソリューション センターを参照してください。トピックには、デザイン アシスタンス、アドバイザリ、トラブルシュート ヒントなどが含まれます。

リファレンス

このガイドの補足情報は、次のサイトにリストされている資料を参照してください。

- Vivado Design Suite 2012.3 資料ページ
  http://japan.xilinx.com/support/documentation/dt_vivado_vivado2012-3.htm