

タイミング クロージャ ユーザー ガイド

UG612 (v13.4) 2012 年 1 月 18 日



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.

© 2012 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

本資料は英語版 (v13.4) 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。

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	
2011 年 3 月 1 日	13.1	第 7 章「タイミング クロージャの達成」を追加
2011 年 7 月 6 日	13.2	<ul style="list-style-type: none">次に内容を追加<ul style="list-style-type: none">最小周期マルチコーナーおよびマルチノードのタイミング解析SYSTEM_JITTERReg_sr_r および Reg_sr_oSDC を .sdc へ変更タイムスペック数がランタイムに影響があることを示す警告を追加第 7 章「タイミング クロージャの達成」の複数箇所での情報を追加
2011 年 10 月 19 日	13.3	<ul style="list-style-type: none">タイトルを『タイミング制約ユーザー ガイド』から『タイミング クロージャ ユーザー ガイド』へ変更Timing Analyzer のサブタイトルを一部変更TIMEGRP DATA_IN OFFSET 例の構文エラーを修正グラフィックを一部修正一部の図の位置を変更一部の図のタイトルを変更非同期 SR および CLR リカバリ パスに関する情報を追加要件の制約が多すぎる場合の記述を変更transversing をすべて traversing に変更 (日本語版は変更なし)「マルチコーナーおよびマルチノードのタイミング解析」と「非同期リセットパス」セクションを移動新しい「タイミングが満たされているのにハードウェアでエラーになるデザイン」セクションを追加「デザイン エレメントの例は、複数のタイムグループで構成されています。」に文章を変更マルチコーナー解析に関する情報を追加以前の第 7 章を次の章に分割第 7 章「タイミング クロージャの達成」、第 8 章「タイミング エラーの回避」、第 9 章「クロスプローブ」
2012 年 1 月 18 日	13.4	<ul style="list-style-type: none">『制約ガイド』(UG625) からタイミング制約情報をこの文書へ移動

目次

改訂履歴.....	3
第 1 章: 概要	
第 2 章: タイミング制約手法	
基本的な制約手法	12
入力タイミング制約	14
register-to-register タイミング制約.....	21
出力タイミング制約	28
第 3 章: タイミング制約の基礎	
制約システム.....	37
制約の優先順位	58
タイミング制約	60
PERIOD 制約	61
OFFSET 制約	67
FROM:TO (マルチサイクル) 制約	73
グループ制約の構文	81
タイミング制約の作成.....	81
第 4 章: XST でのタイミング制約の指定	
XST タイミング制約の指定	83
タイミング モデル.....	83
XCF 制約の優先順.....	84
XST でのタイミング制約の指定方法	84
XST タイミング制約の構文例.....	86
ASYNC_REG (非同期レジスタ)	87
CLOCK_SIGNAL (クロック信号).....	88
MAXDELAY (最大遅延)	90
MAXSKEW (最大スキュー)	91
OFFSET (オフセット).....	93
PERIOD (周期)	94
SYSTEM_JITTER (システム ジッター)	97
NET/PIN/INST TIG (タイミング無視).....	99
TIMEGRP (タイミング グループ)	100
マルチサイクル パス	101
TIMESPEC (タイミング仕様).....	102
TNM (タイミング名).....	105
TNM_NET (タイミング名ネット)	106
第 5 章: Synplifyでのタイミング制約の指定	
制約のタイプ	108
HDL でのタイミング制約の指定	109
HDL タイミング制約の構文例	110
black_box_pad_pin	111
black_box_tri_pins	112

syn_force_seq_prim	114
syn_gatedclk_clock_en	116
syn_gatedclk_clock_en_polarity	118
syn_isclock	120
syn_tpdn	121
syn_tcon	123
syn_tsun	126
SDC ファイル (Tcl) でのタイミング制約の指定	129
define_clock	130
define_clock_delay	132
define_compile_point	133
define_current_design	134
define_false_path	135
define_input_delay	137
define_io_standard	138
define_multicycle_path	139
define_output_delay	141
define_path_delay	143
define_reg_input_delay	145
define_reg_output_delay	146
From/To/Through ポイントの指定	147
SCOPE スプレッドシートでのタイミング制約の指定	151
フォワード アノテーション	151

第 6 章: タイミング解析

マルチコーナーおよびマルチノードのタイミング解析	153
非同期リセット パス	154
Timing Analyzer	154
タイミング レポート	155
PERIOD 解析	156
クロック ドメイン	158
FROM:TO (マルチサイクル) 制約	166
OFFSET IN 制約の解析	170
OFFSET IN BEFORE 制約	174
OFFSET IN AFTER 制約	182
OFFSET OUT 制約の解析	182
OFFSET OUT 制約	185
OFFSET OUT AFTER 制約	186
OFFSET OUT BEFORE 制約	192
クロック スキュー	193
クロックのばらつき	196

第 7 章: タイミング クロージャの達成

タイミング クロージャの達成	199
タイミング クロージャを達成する手順	203
手順 1: ピン制約の指定	204
手順 2: HDL コード記述とデバイス アーキテクチャ リソースの使用	206
手順 3: 合成ツールの駆動	210
手順 4: インプリメンテーション ツールに対するグローバルまたはパス別のタイミング制約の設定	216
手順 5: インプリメンテーションの実行	219

手順 6 : SmartXplorer の実行	222
手順 7 : レポートの確認	224
手順 8 : TRCE の実行およびタイミング結果とレポートの解析	228
 第 8 章: タイミング エラーの修正	
タイミング結果の確認.....	229
クロック レポート	230
タイミング サマリ	230
便利なストラテジ	231
タイミング エラーの主な原因.....	233
タイミング エラーの例	234
 第 9 章: クロス プローブ	
FPGA Editor および Timing Analyzer 間のクロス プローブ.....	247
Timing Analyzer および Technology Viewer 間のクロス プローブ	248
PlanAhead から FPGA Editor へのクロス プローブ	248
デバッグ中のクロス プローブの使用	249
 付録 A: その他のリソース	
ザイリンクス リソース	253
ISE の資料.....	253
SmartXplorer の資料.....	253
PlanAhead の資料	254

概要

『タイミング クロージャ ユーザー ガイド』(UG612) では、パフォーマンスに優れたアプリケーションでのタイミング クロージャについて説明します。対象は、ビギナーからアドバンス ユーザーまで、すべての FPGA 設計者になります。

最新のザイリンクス デバイスでは、その他のテクノロジーや古いデバイスにあったスピード制限がなくなっています。ザイリンクスの FPGA デバイスは、ASIC デバイスにしかフィットしなかったり、ASIC デバイスでしか実行できなかったような高クロック周波数にも対応しています。ただし、設計者はパフォーマンス目標を達成する方法を知っておく必要があります。

このガイドに含まれる内容は、次のとおりです。

- 基本的なタイミング制約
- エレメントのグループ化および制約システム ソフトウェアの理解
- クロック スキューおよびクロックのばらつきなどを含む基本的な制約の解析情報
- XST (Xilinx Synthesis Technology) でのタイミング制約の指定
- Synplifyでのタイミング制約の指定

タイミング制約手法

パフォーマンス目標を達成するためには、確実な手法を使用する必要があります。この章では、次について説明します。

- デザイン要件の理解
- デザイン要件を満たすための制約の適用

デザインを開始する前に次を理解しておく必要があります。

- システムのパフォーマンス要件
- ターゲット デバイスの機能

これらを理解しておくで、デバイスの機能を使用して最高のパフォーマンスを達成するコードが記述できます。

FPGA デバイスの要件は、システムとアップストリーム/ダウストリーム デバイスによって異なります。FPGA デバイスへのインターフェイスがわかれば、内部要件の概要もわかります。これらのタイミング要件を満たす方法は、デバイスとその機能によって異なります。

ユーザーは次を理解している必要があります。

- デバイスのクロック構造
- RAM および DSP ブロック
- デバイスに含まれるハード IP すべて

詳細については、[付録 A「その他のリソース」](#)のデバイスのユーザー ガイドを参照してください。

タイミング制約を使用すると、すべてのデザイン要件をインプリメンテーション ツールに伝えることができます。これは、すべてのパスに適切な制約が使用されることを意味します。この章では、FPGA デバイスで最もよく使用されるタイミング パスを識別してそれらにできるだけ効率的に制約を付ける方法について、簡単に説明しています。

基本的な制約手法

デザインを有効にするには、すべてのパスのタイミング要件がインプリメンテーションに渡される必要があります。タイミング要件は、適用されるパスの種類に基づいて複数のグローバルなカテゴリに分類されます。最も一般的なパス カテゴリのタイプは次のとおりです。

- 入力パス
- レジスタ間のパス
- 出力パス
- パス特定の例外

これらのグローバル カテゴリの種類それぞれに関連付けられるのは、ザイリンクス タイミング制約です。これらの制約を指定する最も効率的な方法は、まずグローバル制約を付けて、パス特定の例外を必要に応じて追加していく方法です。多くの場合、グローバル制約だけで十分です。

FPGA デバイスのインプリメンテーション ソフトウェアは、指定されたタイミング要件で実行され、デバイス リソースを割り当て、タイミング要件を満たすのに必要なエフォートを使用しますが、要件の制約が多すぎる場合や、デザイン要件よりも大きな値が指定されている場合など、この制約を満たすためにツールで使われるエフォートがかなり増加します。このようにエフォートが増加すると、メモリ使用率が増加し、ツールのランタイムが長くなります。また、制約を付けすぎると、その特定の制約だけでなく、ほかの制約でもパフォーマンスが悪化することがあります。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ことをお勧めします。

本書で説明されている制約の適用方法では、UCF 制約構文例を使用しています。このフォーマットは、デザイン要件に従った制約構文をハイライトするために使用されています。次の理由から、デザイン制約を入力するには、**Constraints Editor** を使用するのが一番簡単な方法です。

- デザインに関連するすべてのタイミング制約すべてを同じディレクトリで管理できます。
- デザイン要件を基にタイミング制約を作成しやすくなります。

タイミング要件は、適用されるパスの種類によって複数のグローバル カテゴリに分類されます。

最も一般的なパス カテゴリのタイプは次のとおりです。

- 入力パス
- 同期エレメント間のパス
- パス特定の例外
- 出力パス

ザイリンクス タイミング制約は、これらのグローバル制約タイプにそれぞれ関連付けられます。これらの制約を指定するには、次の方法が効率的です。

- まずグローバル制約を付けます。
- 特定パスに例外を必要に応じて追加していきます。

ほとんどの場合、グローバル制約だけで十分です。

過剰な制約

FPGA デバイスのインプリメンテーション ツールは、指定されたタイミング要件で実行され、デバイス リソースを割り当て、タイミング要件を満たすのに必要なエフォートを実行します。

要件が次のいずれかの場合、この制約を満たすためにツールで使用するエフォートがかなり増加します。

- 制約が多すぎる場合
- デザイン要件よりも大きな値が指定されている場合

このようにエフォートが増加すると、次が発生します。

- メモリ使用率の増加
- ツールのランタイムの増加

制約を付けすぎること、次の両方の制約のパフォーマンスが落ちてしまうことがあります。

- 問題となっている制約
- ほかの制約

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ことをお勧めします。

INPUT_JITTER および **SYSTEM_JITTER** 要件を含め、デザインに正しく制約が付けられている場合、制約をそれ以上に付ける必要はありません。

デザイン ファイルへのコメント 添付

制約ファイルには常にコメントを付けるようにしてください。こうすることで、その制約が使用されている理由をほかの設計者に伝えることができます。

コメントには、次を含めます。

- 制約のソース
- **PERIOD** 制約が外部クロックに基づいているかどうか

Constraints Editor の使用

このガイドでは、**XCF (XST 制約ファイル)** の構文例を使用します。デザイン要件はこの形式でインプリメンテーション ツールに伝えられますが、デザイン制約を入力するには、**Constraints Editor** を使用するのが一番簡単な方法です。

- **Constraints Editor** を使用すると、デザインに関連するすべてのタイミング制約すべてを同じディレクトリで管理できます。
- **XCF** 構文を使用してデザイン要件に基づいたタイミング制約が作成しやすくなります。

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。入力タイミング制約は、FPGA デバイスのパッケージの外部ピンまたはパッドからデータをキャプチャする内部の同期エレメントまたはレジスタまでのデータパスに適用されます。入力タイミング要件を指定するには、OFFSET IN 制約を使用します。

入力タイミング要件は、そのインターフェイスのタイプ (ソース同期かシステム同期か) とデータレート (SDR か DDR か) によって異なります。これらのカテゴリには、次が含まれます。

- システム同期入力
- 理想的なシステム同期 SDR インターフェイス例のタイミング図
- ソース同期入力
- 理想的なソース同期 DDR インターフェイス例のタイミング図

OFFSET IN 制約では、データと、FPGA のピンまたはパッドでそのデータをキャプチャするために使用されるクロック エッジとの関係を定義します。OFFSET IN の解析には、クロック信号とデータ信号の遅延に影響する内部要因が含まれます。この要因には、次のようなものがあります。

- クロックの周波数と位相変換
- クロックのばらつき
- データ遅延の調整

入力クロックの誤差とクロック到着時間は、OFFSET IN 制約で参照されるインターフェイス クロックに付いた PERIOD 制約で決まります。PERIOD 制約と入力ジッタの追加方法については、第 3 章「タイミング制約の基礎」の「PERIOD 制約」を参照してください。

システム同期入力

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システム クロックのインターフェイスのことです。このインターフェイスでは、共通のシステム クロックが使用されます。ボード トレース遅延とクロック スキューによりインターフェイスの動作周波数が制限されます。

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。

関連する SDR タイミングを含むシステム同期インターフェイス

次の図のシステム同期 SDR アプリケーションの例では、データが次のようになります。

1. 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ます。
2. 次の立ち上がりエッジで FPGA デバイスにキャプチャされます。

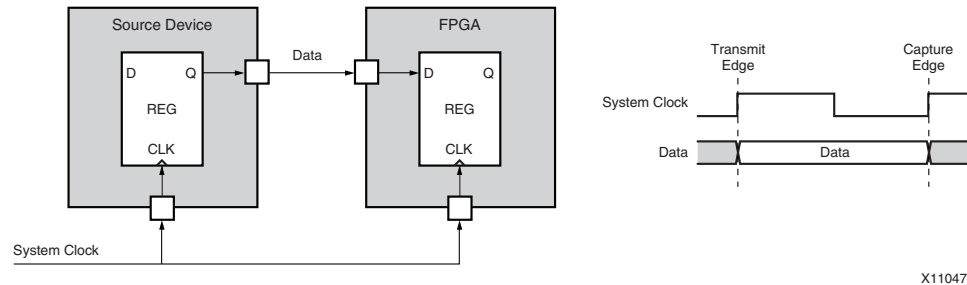


図 2-1：関連する SDR タイミングを含むシステム同期インターフェイス

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 つの制約が、同期エレメント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に適用されます。

入力タイミング

入力タイミングを指定するには、次の手順に従ってください。

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

理想的なシステム同期 SDR インターフェイス

次は、理想的なシステム同期 SDR インターフェイスのタイミング図です。

- このインターフェイスのクロック周期は **5ns** です。
- 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。

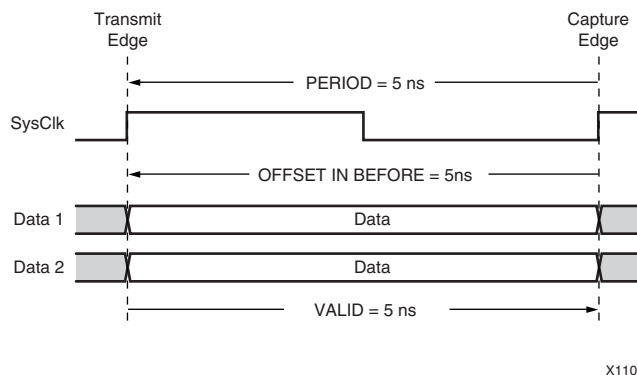


図 2-2：理想的なシステム同期 SDR インターフェイスのタイミング図

グローバル **OFFSET IN** 制約

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。

PERIOD 制約を使用した完全な OFFSET IN 指定は、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、次のバスのデータ ビット両方に適用されます。

- data1
- data2

ソース同期入力

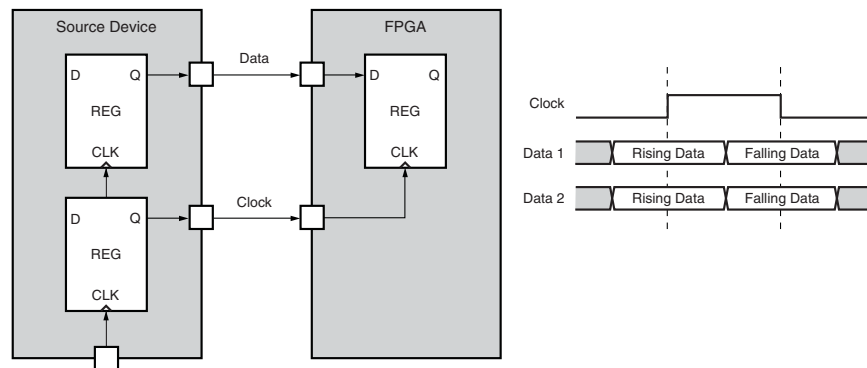
ソース同期入力インターフェイスでは、クロックがレジスタに入力され、ソース デバイスからデータと共に同様のボード トレースで送信されます。このクロックは、この後 FPGA デバイスでデータをキャプチャするために使用されます。

ボード トレース遅延やボード 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ることはなくなっています。また、周波数が高いと、ソース同期入力インターフェイスが通常はデュアルデータ レート (DDR) アプリケーションになります。

関連する DDR タイミングを含むソース同期インターフェイス

次の図のソース同期 DDR アプリケーションの例では、一意のデータが次のように処理されます。

1. クロックの立ち上がりエッジと立ち下りエッジでソース デバイスから送信されます。
2. 再生成されたクロックを使用して FPGA にキャプチャされます。



X11049

図 2-3：関連する DDR タイミングを含むソース同期インターフェイス

グローバル OFFSET IN 制約は、ソース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。DDR インターフェイスでは、1 つの OFFSET IN 制約を入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。これらの制約は、レジスタにキャプチャされた (指定入力クロック エッジでトリガされる) 入力データ ビットすべてのパスに適用されます。

入力タイミング

入力タイミングを指定するには、制約を次の表のように定義します。

表 2-1：入力タイミング

制約	定義対象
クロック周期	入力クロック
グローバル OFFSET IN	立ち上がりエッジ
グローバル OFFSET IN	立ち下がりエッジ

理想的なソース同期 DDR インターフェイス

次は、理想的なソース同期 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はクロック周期 5ns、50/50 デューティ サイクル
- バスの両方のビットのデータは全体の 1/2 周期間有効なまま

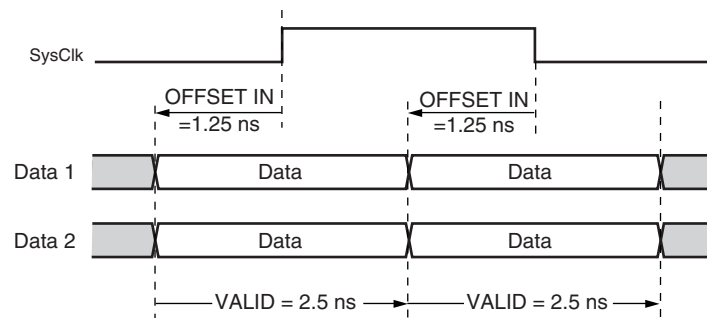


図 2-4：理想的なソース同期 DDR のタイミング図

グローバル OFFSET IN 制約

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 = 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。

このソース同期入力の例では、次のようになります。

- 立ち下がりデータも立ち下がりクロック エッジの1.25ns 前に有効になります。
- 立ち下がりデータも立ち下がりクロック エッジの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。

PERIOD 制約を使用した完全な OFFSET IN 指定は、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;
```

このグローバル制約は、次のバスのデータ ビット両方に適用されます。

- data1
- data2

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 (デュアル データ レート) 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。

波形の例

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 = CLK; 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 (デュアル データ レート) 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。

波形の例

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 = CLK; 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。

波形の例

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 = CLK; TIMESPEC TS_CLK = PERIOD CLK 5.0 ns HIGH 50%; OFFSET = IN 4.5 ns  
VALID 4 ns BEFORE clock;
```

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 **PERIOD** 指定方法について説明します。

PERIOD 制約には、次の特徴があります。

- クロック ドメインのタイミングを定義します。
- 内部レジスタ間の同期データ パスに適用されます。
- 1 つのクロック ドメイン内のパスを解析します。
- 関連するクロック ドメイン間のパスすべても解析します。
- 解析中のクロック ドメイン間のすべての周波数、位相、誤差を考慮します。

同期クロック ドメインに制約を付ける方法は、次のようになります。

- 「関連する同期 **DLL/DCM/PLL/MMCM** ドメイン (自動)」s
- 「関連する同期クロック 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。

詳細は、第 3 章「**タイミング制約の基礎**」の「**PERIOD 制約**」を参照してください。

関連する同期 **DLL/DCM/PLL/MMCM** ドメイン (自動)

よくあるクロック回路は、次のようなタイプです。

- 入力クロックが **DCM/PLL/MMCM** に読み込まれます。
- 出力がデバイスの同期パスのクロックに使用されます。

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。

入力クロックに **PERIOD** 制約を付けると、ツールで自動的に次が実行されます。

- 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ます。
- 出力クロック ドメイン間のクロック関係を決定します。
- これらの同期クロック ドメイン間のパスを自動的に解析します。

例

この例では、入力クロックが **DCM** に入力されています。次の図に、この例の回路図を示します。

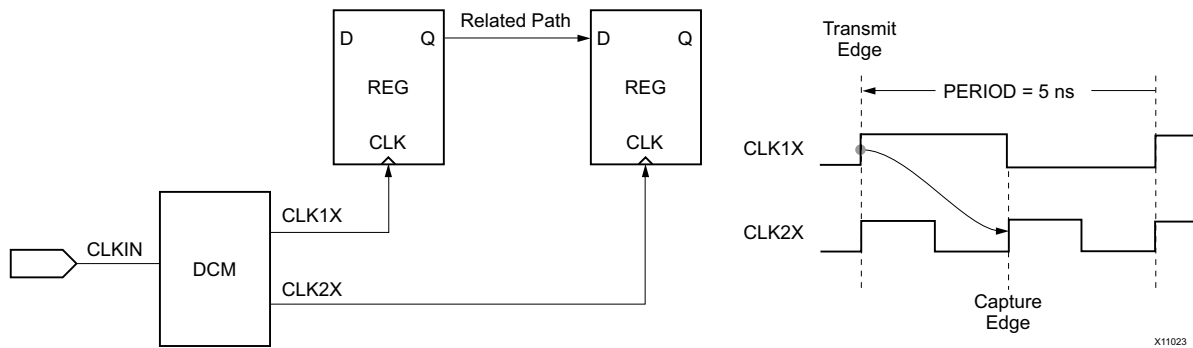


図 2-5 : DCM への入力クロック

この例の **PERIOD** 制約構文は次のようになります。

```
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。この例の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%;
```

DCM では、前述の入力クロックの **PERIOD** 制約に基づいて次が自動的に実行されます。

- DCM 出力の出力クロック制約が 2 つ作成されます。
- 2 つのドメイン間の解析が実行されます。

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**) がデスティネーション クロック

この例の回路図は、次の図のようになります。

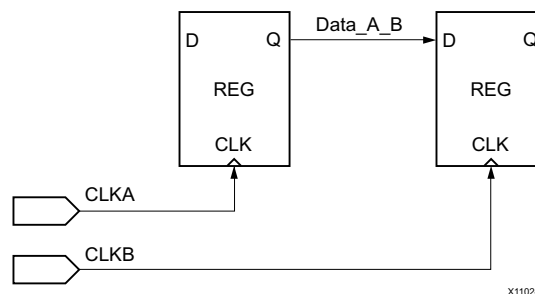


図 2-6 : 別々の外部ピンから FPGA デバイスに入力される 2 つの関連のないクロック

```
NET "CLKA" TNM_NET=FFS "GRP_A"; NET "CLKB" TNM_NET=FFS "GRP_B"; TIMESPECTS_Example=FROM "GRP_A" TO "GRP_B" 5ns DATAPATHONLY;
```

DCM への入力クロック

次の図は DCM を駆動する入力クロックの回路です。

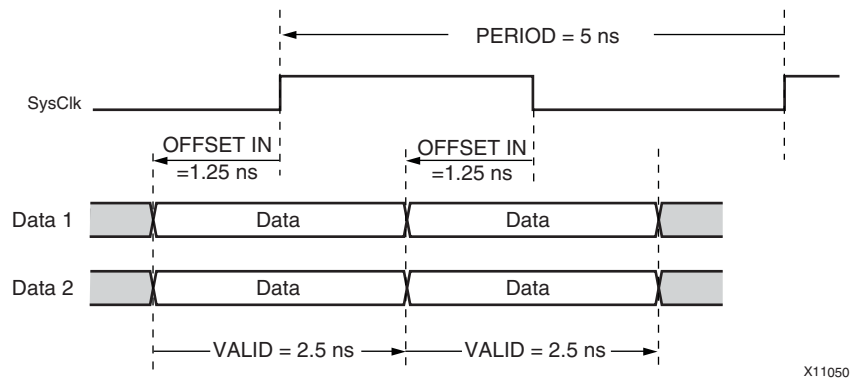


図 2-7 : DCM への入力クロックの例

PERIOD 制約構文

この例の PERIOD 制約構文は次のようになります。

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

表 2-2 : PERIOD 制約の設定

値	定義対象	この例の場合
PeriodValue	クロック周期の長さを定義	DCM への入力クロックの周期は 5ns
HighValue	High のクロック波形の割合をパーセントで定義	波形は 50/50 のデューティ サイクルで HighValue は 50%

この例の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%;
```

DCM では、前述の入力クロックの PERIOD 制約に基づいて次が自動的に実行されます。

- DCM 出力の出力クロック制約が 2 つ作成されます。
- 2 つのドメイン間の解析が実行されます。

関連する同期クロック ドメイン (手動)

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、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。この場合、次のように制約を付けることをお勧めします。

- 両方の入力クロックに PERIOD 制約を別々に作成します。
- クロック間に手動で関係を定義します。

手動で関係を定義したら、次が実行されます。

- この 2 つの同期ドメイン間のパスすべてが自動的に解析されます。

- すべての周波数、位相、誤差情報などが考慮されます。

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。

- プライマリ クロックの **PERIOD** 制約を定義します。
- 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。

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
- 2 つ目のクロック (**CLK2X180**) がその関連クロック

この例の回路図は、次の図のようになります。

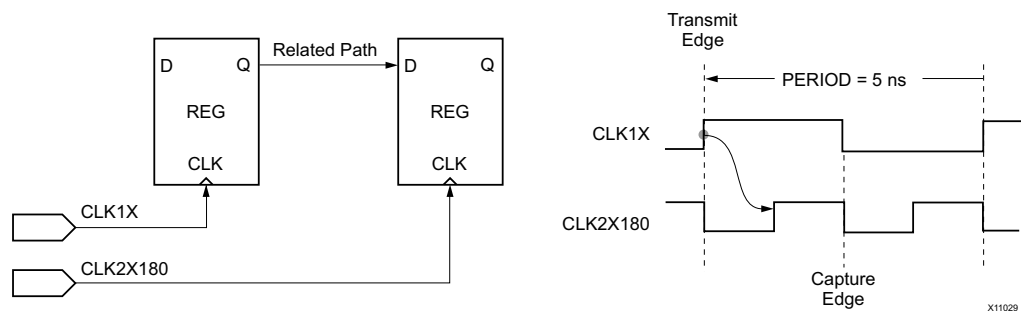


図 2-8：別々の外部ピンから **FPGA** デバイスに入力される 2 つの関連クロック

この例の **PERIOD** 構文は次のようになります。

```
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。

この例の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" 5ns;
TIMESPEC "TS_Clk2X180" = PERIOD "Clk2X180" "TS_Clk1X" / 2 PHAS2 +1.25ns;
```

非同期クロック ドメイン

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があります。

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。

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、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。この場合、次を実行することをお勧めします。

- 入力クロックごとに別の PERIOD 制約を定義します。
- クロック間に手動で関係を定義します。

パス解析

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がすべて自動的に解析されます。解析では、すべての周波数、位相、誤差などの情報が考慮されます。

複雑な手動関係の定義

ザイリンクスの制約システムでは、PERIOD 制約を使用して次のクロック ドメイン間の複雑な手動関係を定義できます。

- クロック周波数
- 位相変換

PERIOD 制約を使用してクロック ドメイン間の複雑な手動関係クロック ドメイン間を定義するには、次の PERIOD 制約を定義します。

- プライマリ クロックの PERIOD 制約
- 最初の PERIOD 制約をリファレンスとして使用した関連クロックの PERIOD 制約

クロック関係を定義するための PERIOD 制約の使用方法については、第 3 章「タイミング制約の基礎」の「PERIOD 制約」を参照してください。

別々の外部ピンから FPGA デバイスに入力される 2 つの関連クロック

別々の外部ピンから FPGA デバイスに入力される 2 つの関連クロック詳細は、次の図を参照してください。

- 最初のクロック (CLK1X) がプライマリ クロック
- 2 つ目のクロック (CLK2X180) がその関連クロック

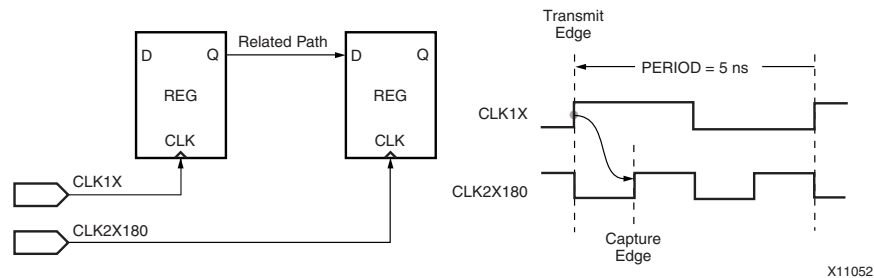


図 2-9：別々の外部ピンから FPGA デバイスに入力される 2 つの関連クロック

PERIOD 制約構文

この例の PERIOD 制約構文は次のようになります。

```
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;
```

PERIOD 値

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 2 倍の周波数で動作しているので、CLK2X180 の PERIOD 周期は CLK1X の 1/2 になります。

PHASE 値

関連する PERIOD 定義では、PHASE 値で次のクロックの時間差を定義します。

- ソース クロックの立ち上がりエッジ
- 関連クロック

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 クロックの立ち上がりエッジの 1.25ns 後に開始します。

PERIOD 制約の構文例

```
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;
```

非同期デザイン手法の例

FIFO エlementを使用して非同期クロック ドメイン間でデータをキャプチャして転送することができます。

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があります (必須ではありません)。

最大データ パス遅延への制約の使用

ザイリンクス制約システムでは、次に関係なく、最大データ パス遅延に制約を付けることができます。

- ソースおよびディスティネーション クロックの周波数
- 位相関係

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

ソースおよびディスティネーション クロックの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には、次を定義してください。

- ソース同期エレメントのタイム グループを定義
- ディスティネーション同期エレメントのタイム グループを定義
- 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データ パスの最大遅延を定義

FROM-TO 制約に DATAPATHONLY キーワードを使用する方法については、第 3 章「タイミング制約の基礎」の「FROM:TO (マルチサイクル) 制約」を参照してください。

別々の外部ピンから FPGA デバイスに入力される 2 つの関連しないクロック

次の図では、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

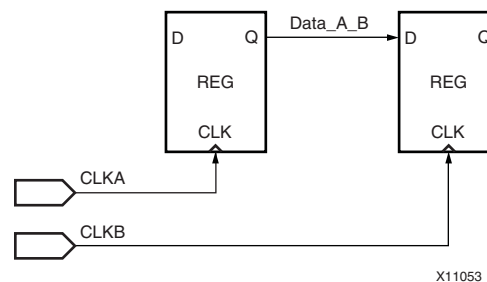


図 2-10 : 別々の外部ピンから FPGA デバイスに入力される 2 つの関連のないクロック

構文例

```
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、データをキャプチャする内部の同期エレメントまたはレジスタからFPGA デバイスのパッケージの外部ピンまたはパッドまでのデータパスに適用されます。

出力タイミング要件を指定するには、**OFFSET OUT** 制約を使用します。出力タイミング要件は、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 OUT 制約では、データと、FPGA のピンまたはパッドでそのデータを開始するために使用されるクロック エッジとの関係を定義します。**OFFSET OUT** の解析には、クロック信号とデータ信号の遅延に影響する内部要因が含まれます。この要因には、次のようなものがあります。

- クロックの周波数と位相変換
- クロックのばらつき
- データ遅延の調整

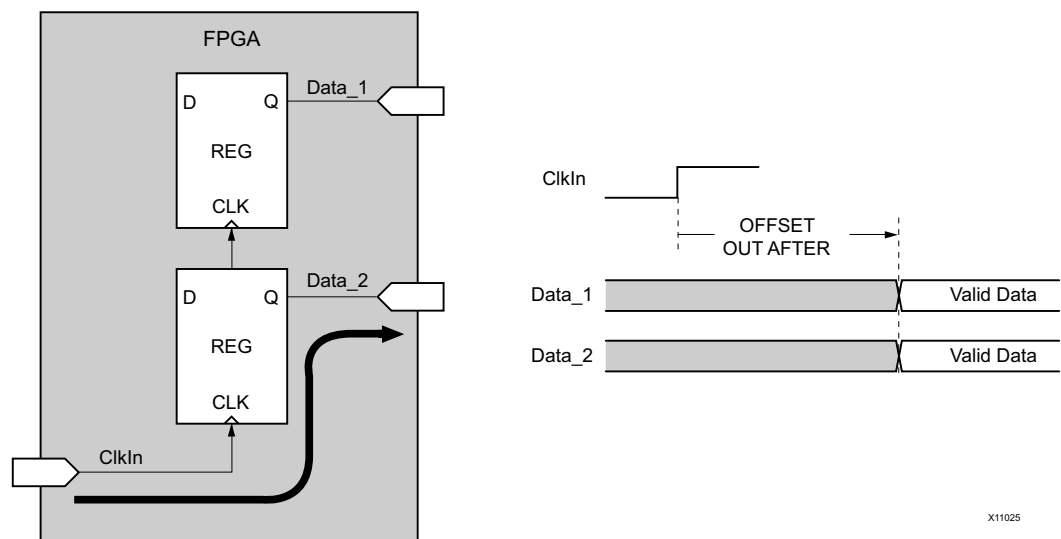


図 2-11：出力遅延パスの例

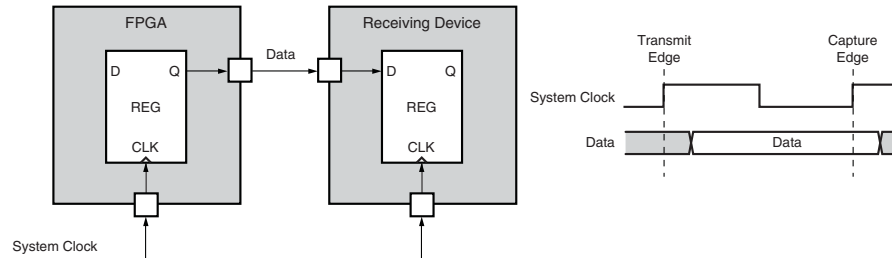
入力クロックの誤差とクロック到着時間は、**OFFSET OUT** 制約で参照されるインターフェイス クロックに付いた **PERIOD** 制約で決まります。

PERIOD 制約と入力ジッタの追加方法については、第 3 章「タイミング制約の基礎」の「**PERIOD 制約**」を参照してください。

システム同期出力

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システム クロックのインターフェイスのことです。このインターフェイスでは共通のシステム クロックが使用されているので、次の図のようにデータだけが FPGA から受信デバイスに送信されます。

詳細は、次の図を参照してください。



X11055

図 2-12：関連する SDR タイミングを含む単純化されたシステム同期出力インターフェイス

出力タイミングの指定

これらのパスに制約を付ける必要がある場合、グローバル **OFFSET OUT** 制約を使用すると、最も効率的に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できます。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 つの制約は、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に適用されます。

出力タイミングを指定するには、次を定義してください。

- 出力クロックのタイム名 (TNM) を定義し、入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。
- インターフェイスのグローバル **OFFSET OUT** 制約を定義します。

システム同期の SDR 出力インターフェイス

次は、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 デバイス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。

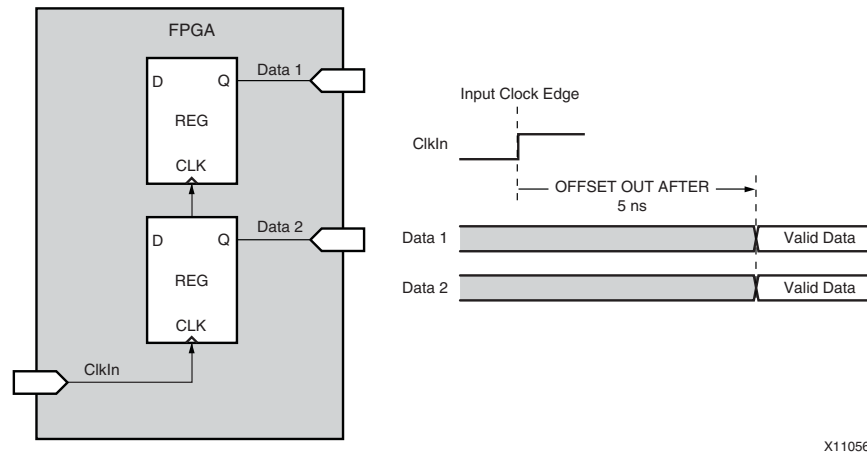


図 2-13：システム同期 SDR 出力インターフェイスのタイミング図

グローバル OFFSET IN 制約

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 AFTER clock;
```

OFFSET OUT 制約では、**OFFSET=OUT <value>** で定義します。

1. 入力クロック ポートの立ち上がりエッジから次までの最大時間
2. データは **FPGA** デバイスのデータ出力ポートでまず有効になります。

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、次のバスのデータ ビット両方に適用されます。

- data1
- data2

ソース同期出力

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。再生成されたクロックは、データと共に送信されます。

このインターフェイスのパフォーマンスは主に次によって制限されます。

- システム ノイズ
- 生成されたクロックとデータ ビット間のスキュー

詳細は、次の図を参照してください。

関連する DDR タイミングを含むソース同期インターフェイス

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。

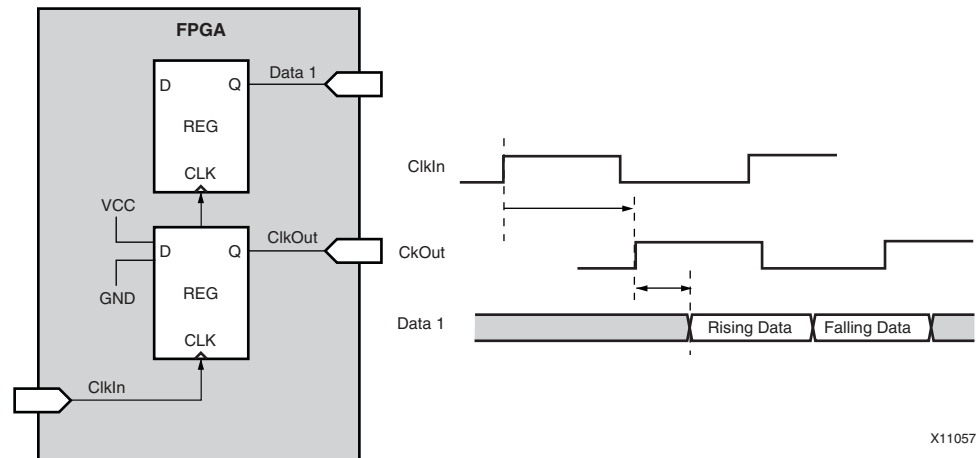


図 2-14 : 関連する DDR タイミングを含むソース同期インターフェイス

OFFSET OUT 制約

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に適用されます。

入力タイミング

入力タイミングを指定するには、次を定義してください。

- 出力クロックのタイム名 (**TNM**) 制約を定義し、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。
- インターフェイスの立ち上がりエッジ (**RISING**) 用にグローバル **OFFSET OUT** 制約を定義します。
- インターフェイスの立ち下がりエッジ (**FALLING**) 用にグローバル **OFFSET OUT** 制約を定義します。

理想的なソース同期 DDR インターフェイス

次は、理想的なソース同期 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はクロック周期 5ns、50/50 デューティ サイクル
- バスの両方のビットのデータは全体の 1/2 周期間有効なまま

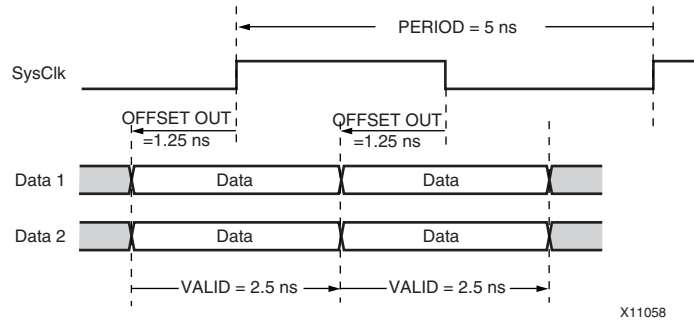


図 2-15：理想的なソース同期 DDR のタイミング図

OFFSET OUT 制約

OFFSET OUT 制約では、**OFFSET=OUT <value>** で定義します。

1. 入力クロック ポートの立ち上がりエッジから次までの最大時間
2. データは FPGA デバイスのデータ出力ポートでまず有効になります。

<value> が指定されない場合、この制約は出力バスのスキューをレポートする役割しか果たしません。

この制約に **REFERENCE_PIN** キーワードを使用すると、再生成した出力クロックを出力データ ピンのスキューをレポートするリファレンス ポイントとして定義できます。

この例の場合、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;
```

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、ほとんどのパスに正しく制約が指定されますが、グローバル制約の規則に当てはまらない例外を含むパスが少数含まれることもあります。

最もよくある例外は、次のとおりです。

- 「False パス (タイミングに影響しない 2 つのレジスタ間のパス)」
- 「マルチサイクル パス」

次の制約のグローバル定義を使用すると、ほとんどのパスに正しく制約が指定されます。

- 入力
- register-to-register
- 出力タイミング

ただし、グローバル制約の規則に当てはまらない例外を含むパスが少数含まれることもあります。

False パス (タイミングに影響しない 2 つのレジスタ間のパス)

タイミング パフォーマンスに影響のないパスのセットは、タイミング解析から削除できます。

タイミング解析から削除するパスのセットを指定するには、タイミング無視 (TIG) キーワードを使用して FROM-TO 制約を使用します。これにより、次を実行できます。

- ソース タイム グループのレジスタのセットを指定します。
- デスティネーション タイム グループのレジスタのセットを指定します。
- これらのタイム グループ間のパスをすべて解析から自動的に削除します。

この方法でタイミング無視 (TIG) 制約を指定するには、次を定義します。

- ソース タイム グループのレジスタのセット
- デスティネーション タイム グループのレジスタのセット
- グループ間のパスを削除するためには TIG キーワードを使用した FROM-TO 制約

次の図では、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しないので、解析から削除しています。

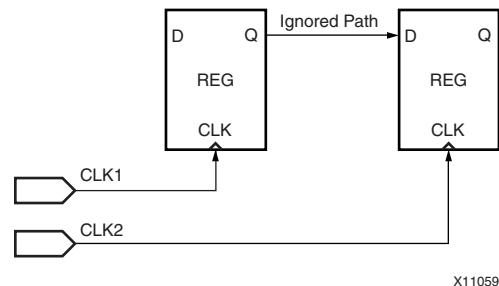


図 2-16: タイミングに影響しない 2 つのレジスタ間のパス

タイム グループ間の TIG (タイミング無視) の一般的な構文は、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、次が実行されています。

- SRC_GRP によりパス トレースの開始されるソース レジスタのセットが定義されています。
- DST_GRP によりパス トレースが終了するデスティネーション レジスタのセットが定義されています。
- SRC_GRP で始まり で DST_GRP 終わるパスはすべて無視されています。

構文例

```
NET "CLK1" TNM_NET = FFS "GRP_1";
NET "CLK2" TNM_NET = FFS "GRP_2";
TIMESPEC TS_Example = FROM "GRP_1" TO "GRP_2" TIG;
```

マルチサイクル パス

マルチサイクル パスとは、**PERIOD** で定義されたクロック周波数よりも低いレートで、データがソースからデスティネーションに転送されるパスのことです。

これは、同期エレメントに共通クロック イネーブル信号を付けてゲート処理される場合によく発生します。マルチサイクル パスを定義すると、これらの同期エレメントのタイミング制約をデフォルトの **PERIOD** 制約よりも緩められます。

マルチサイクル パス制約は、**PERIOD** 制約の識別子 (**TS_clk125**) と周期サイクルの乗算または数で定義できます (**TS_clk125 * 3**)。これで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の優先度が適切に決定されます。

マルチサイクル パスのセットの指定

マルチサイクル パスのセットを指定するのに最もよく使用されるのは、クロック イネーブル信号を使用してタイム グループを定義する方法です。これにより、次を実行できます。

- 共通のクロック イネーブル信号を使用して、ソースおよびデスティネーションの同期エレメント両方を含むタイム グループ 1 つを定義します。
- マルチサイクル制約をこれらの同期エレメント間のパスすべてに自動的に適用します。

この方法で **FROM:TO** (マルチサイクル) 制約を指定するには、次を定義します。

- 共通のクロック ドメインの **PERIOD** 制約
- 共通のクロック イネーブル信号に基づいたレジスタのセット
- 新しいタイミング要件を記述する **FROM-TO** (マルチサイクル) 制約

共通のクロック イネーブル信号でクロックが供給される 2 つのレジスタ間のパス

次の図では、2 つのレジスタ間のパスが共通のクロック イネーブル信号を使用していると仮定しています。

この例では、クロック イネーブルがリファレンス クロックの半分のレートでトグルしています。

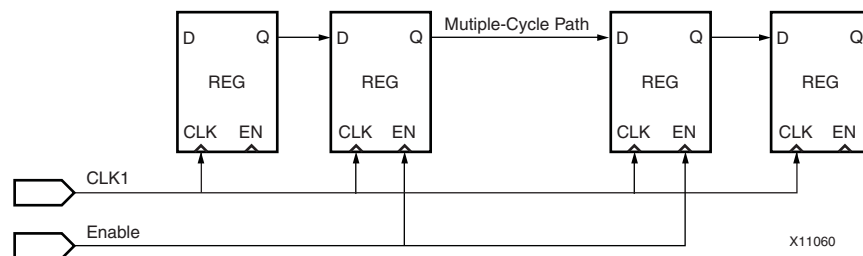


図 2-17：共通のクロック イネーブル信号でクロックが供給される 2 つのレジスタ間のパス

一般的な構文

タイム グループ間のマルチサイクル パスを定義する一般的な構文は、次のようになります。

```
TIMESPEC "TSid" = FROM "MC_GRP" TO "MC_GRP" <value>;
```

MC_GRP

FROM:TO (マルチサイクル) の例では、次が実行されます。

- MC_GRP により、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マルチサイクル タイミング要件が指定されます。
- MC_GRP への入力および出力パスが適切な PERIOD 制約を付けて解析されます。

構文例

```
NET "CLK1" TNM_NET = "CLK1";  
TIMESPEC "TS_CLK1" = PERIOD "CLK1" 5 ns HIGH 50%;  
NET "Enable" TNM_NET = FFS "MC_GRP";  
TIMESPEC TS_Example = FROM "MC_GRP" TO "MC_GRP" TS_CLK1*2;
```


タイミング制約の基礎

この章には、次の内容が含まれます。

- 「PERIOD 制約」
- 「OFFSET 制約」
- 「FROM:TO (マルチサイクル) 制約」

制約サブシステムをさらに理解するためのグループ エレメントの機能についても説明します。

制約システム

制約システムとは、デザインの物理制約およびタイミング制約を解析して理解するインプリメンテーション ツールの一部 (NGDBuild) のことです。

制約システムでは、次が実行されます。

- 次のファイルから制約を解析し、その他のインプリメンテーション ツールにこの情報を渡します。
 - NCF
 - XCF
 - EDN、EDF、EDIF
 - NGC
 - NGO
- 制約が正しく指定されているかどうかを確認します。
- 必要な属性を対応するエレメントに適用します。
- デザインと正しく関連付けられていない制約に対してエラーおよび警告メッセージを表示します。

DLL/DCM/PLL/BUFR/PMCD/MMCM コンポーネント

入力パッド クロック ネットの TIMESPEC PERIOD が DCM/DLL/PLL/BUFR/PMCD/MMCM コンポーネント (クロック調整ブロック) を介してトレースまたは変換されると、派生したクロックまたは出力クロックに新しい PERIOD 制約が付きます。

変換中にはクロック調整ブロックの各クロック出力ピンに次の制約が付けられ、デスティネーション エレメントのタイミング グループが生成されます。

- 新しい TIMESPEC PERIOD 制約
- 対応する TNM_NET 制約

新しい TIMESPEC PERIOD 制約はクロック調整ブロック コンポーネントの操作に基づいて作成されます。変換では、次が実行されます。

- クロック出力の位相関係の要因が考慮されます。
- PERIOD に必要な値の乗算または除算が実行されます。

変換条件

変換は、次の条件で発生します。

- TIMESPEC PERIOD 制約がクロック調整ブロック コンポーネントの CLKIN ピンにトレースされ、さらに
- PERIOD 制約に関連するグループが次の状態の場合：
 - 調度 1 つの PERIOD 制約で使用されている。
 - FROM:TO (マルチサイクル) または OFFSET 制約も含めたその他のタイミング制約で使用されていない。
 - 参照されない、またはその他のユーザー グループ定義とは関連していない。

DCM 出力の新規 PERIOD 制約の例

「変換条件」が満たされる場合、(1) の制約が (2) および (3) の制約に変換されます。

- (1) TIMESPEC "TS_clk20" = PERIOD "clk20_grp" 20 ns HIGH 50%;
- (2) CLK0:TS_clk20_0=PERIOD clk20_0 TS_clk20*1.000000 HIGH 50.000000%
- (3) CLK90:TS_clk20_90=PERIOD clk20_90 TS_clk20*1.000000 PHASE + 5.000000 nS HIGH 50.000000%

これらの制約は、次の図のクロック構造に基づいて変換されます。

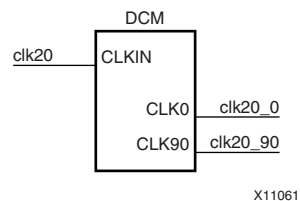


図 3-1 : DCM 出力の新規 PERIOD 制約

レポート メッセージ

次のメッセージが NGDBuild (design.bld) または MAP (design.mrp) レポートに表示されます。

```
INFO:XdmHelpers:851 - TNM " clk20_grp ", used in period specification "TS_clk20", was traced
into DCM instance "my_dcm".The following new TNM groups and period specifications were
generated at the DCM output(s):
```

```
clk0:TS_clk20_0=PERIOD clk20_0 TS_clk20*1.000000 HIGH 50.000000%
clk90:TS_clk20_90=PERIOD clk20_90 TS_clk20*1.000000 PHASE + 5.000000 nS HIGH 50.000000%
```

変換される PERIOD 制約の調整

CLKIN_DIVIDE_BY_2 属性が 図 3-1 「DCM 出力の新規 PERIOD 制約」の DCM で TRUE に設定されている場合は、変換される PERIOD 制約もそれに合わせて調整されます。次の制約は、この属性の結果です。

```
CLK0: TS_clk20_0=PERIOD clk20_0 TS_clk20*2.000000 HIGH 50.000000%
CLK90:TS_clk20_90=PERIOD clk20_90 TS_clk20*2.000000 PHASE + 5.000000 nS HIGH 50.000000%
```

変換条件が満たされない場合

「変換条件」が満たされない場合：

- PERIOD 制約はクロック調整ブロック コンポーネントの出力または派生クロックには配置されません。
- エラーまたは警告メッセージが NGDBuild レポートに表示されます。

エラー メッセージの例

```
"ERROR:NgdHelpers:702 - The TNM "PAD_CLK" drives the CLKIN pin of CLKDLL "$I1".This TNM cannot be traced through the CLKDLL because it is not used in exactly one PERIOD specification.This TNM is used in the following user groups and/or specifications:
```

```
TS_PAD_CLK=PERIOD PAD_CLK 20000.000000 pS HIGH 50.000000%
TS_01=FROM PAD_CLK TO PADS 20000.000000 pS"
```

元の TIMESPEC PERIOD は、タイミング レポートで「0 items analyzed」と表示されます。

新しく作成された TIMESPEC PERIOD 制約には、クロック調整ブロック コンポーネントに接続されるパスすべてが含まれます。

PERIOD 制約が変換されずにクロック調整ブロック コンポーネントにのみトレースされる場合は、次のように処理されます。

- タイミング レポートには「0 items analyzed」と表示されます。
- その他の PERIOD 制約はレポートされません。

PERIOD 制約がほかの同期エレメントをトレースする場合、解析にはこれらの同期エレメントのみが含まれます。

同期エレメント

同期エレメントには、次が含まれます。

- フリップフロップ
- ラッチ
- 分散 RAM
- ブロック RAM
- 分散 ROM
- ISERDES
- OSERDES
- PPC405
- PPC440
- MULT18X18

- DSP48
- MGT (GT、GT10、GT11、GTP、GTX、GTH)
- MCB
- SRL16
- EMAC
- FIFO (16、18、& 36)
- PCIE
- TEMAC

NET PERIOD を使用した解析

NET PERIOD 制約は入力クロック パッドまたはネットに適用される場合、クロック調整ブロック コンポーネントを介して変換されません。このため、これらの制約に対して解析されるアイテムやパス数が 0 になることがあります。

NET PERIOD は MAP、PAR、タイミング解析中にのみ解析されます。**MAP -timing** および **PAR** でタイミングが呼び出されると、タイミング ツールでは配置配線用にクロック調整ブロックが操作されますが、タイミング解析レポート用には操作されません。

TIMESPEC PERIOD 制約がクロック調整ブロックの入力ピンにトレースされると、NGDBuild または変換プロセスにより、元の TIMESPEC PERIOD 制約が派生した出力クロックに基づいて新しい TIMESPEC PERIOD 制約に変換されます。この変換情報は、NGDBuild レポート (design.bld) に記述されます。

MAP、PAR、および Timing Analyzer では、物理制約ファイル (PCF) に伝播されるこの新しい派生クロックの TIMESPEC PERIOD 制約が使用されます。

元の TIMESPEC PERIOD は、次のように処理されます。

- この変換中も変更されません。
- 新しい TIMESPEC PERIOD 制約のリファレンスとして使用されます。

Constraints Editor では、元の PERIOD 制約のみが表示されます。Constraints Editor では、新しく変換された PERIOD 制約は認識されません。

PHASE キーワード

PHASE キーワードは、関連するクロック同士の関係を定義するために使用されます。この関係は、タイミング解析ツールで OFFSET 制約とクロス クロック ドメイン パスが解析される際に使用されます。

PHASE キーワードは、次のいずれかで入力できます。

- UCF/NCF に入力
- NGDBuild の DCM/DLL/PLL コンポーネントの変換で入力

DCM/PLL/DLL コンポーネントの位相シフト値を FPGA Editor で変更すると、その変更は PCF ファイルには反映されません。

タイミング解析ツールは、PCF の PHASE 値を使用して DLL/DCM/PLL 位相シフト値をエミュレートします。FPGA Editor での変更を反映させるには、PCF にその変更を手動で入力する必要があります。

PHASE を使用した DLL/DCM/PLL 操作

次の表は、TIMESPEC PERIOD 制約の付いた新しい DCM/DLL/PLL コンポーネントの出力ネットを表示しています。これらの新しい制約は、元の PERIOD (TS_CLKIN) 制約に基づいています。TS_CLKIN は、時間値として表示されています。

TS_CLKIN が周波数として使用される場合は、乗算または除算が逆になります。DCM 属性の FIXED_PHASE_SHIFT または VARIABLE_PHASE_SHIFT が使用される場合、位相シフトの値が PHASE キーワードの値に含まれます。

DCM 属性の FIXED_PHASE_SHIFT または VARIABLE_PHASE_SHIFT の位相シフト値は、次の表には含まれていません。

表 3-1 : DCM を介した PERIOD 制約の変換クロージャ

出力ピン	PERIOD 値	位相シフト値
CLK0	$TS_CLKIN * 1$	なし
CLK90	$TS_CLKIN * 1$	$PHASE + (clk0_period * \frac{1}{4})$
CLK180	$TS_CLKIN * 1$	$PHASE + (clk0_period * \frac{1}{2})$
CLK270	$TS_CLKIN * 1$	$PHASE + (clk0_period * \frac{3}{4})$
CLK2x	$TS_CLKIN / 2$	なし
CLK2x180	$TS_CLKIN / 2$	$PHASE + (clk2x_period * \frac{1}{2})$
CLKDV	$TS_CLKIN * clkdv_divide$ (clkdv_divide = CLKDV_DIVIDE プロパティ (デフォルト = 2.0) の値)	なし
CLKFX	$TS_CLKIN / clkfx_factor$ (clkfx_factor = CLKFX_DIVIDE プロパティ (デフォルト = 1.0) の値で割った CLKFX_MULTIPLY プロパティ (デフォルト = 4.0) の値)	なし
CLKFX180	$TS_CLKIN / clkfx_factor$ (clkfx_factor = CLKFX_DIVIDE プロパティ (デフォルト = 1.0) の値で割った CLKFX_MULTIPLY プロパティ (デフォルト = 4.0) の値)	$PHASE + (clkfx_period * \frac{1}{2})$

TNM/TNM_NET 属性を使用したタイミング グループの作成

同じ TNM/TNM_NET で指定されたすべてのデザイン エLEMENT は 1 つのタイミング グループとみなされます。1 つのデザイン エLEMENT が複数のタイミング グループ (TNM/TNM_NET) に属することもあります。

TNM/TNM_NET 属性は、次のものに適用できます。

- ネット接続 (NET)
- インスタンス/モジュール (INST)
- インスタンス ピン (PIN)

各ELEMENT、ドライバピン、またはマクロ ドライバ ピンに設定する TNM/TNM_NET は 1 つだけにしないと、タイミング解析が正しく実行されないことがあります。

ネット接続 (NET)

ネット接続別にグループにすると、同期ELEMENTおよびパッドを駆動するネットまたは信号を指定することでELEMENTをグループ化できます。

この方法では、次のようなマルチサイクル パスのELEMENTが識別されます。

- クロック イネーブルで制御される
- FROM:TO (マルチサイクル パス) 制約を付けることが可能

この方法では、ネットに TNM_NET (タイミング ネット) または TNM (タイミング名) を使用します。TNM 制約は、通常直接パッドに接続される HDL ポート宣言で使用されます。

TNM 制約がネットまたは信号に設定される場合

TNM 制約がネットまたは信号に設定されると、制約解析ツールでその信号またはネットがダウンストリームの同期ELEMENTまでトレースされます。

TNM は、タイム グループを構成するELEMENTの識別に使用できる制約で、このタイム グループはタイミング制約で使用できます。

これらの同期ELEMENTには、同じ TNM 制約が付きます。TNM 制約名が TIMESPEC またはタイミング制約で使用されます。

クロック パッドまたはネットに TNM が設定される場合

次の図の回路図のクロック ネットは、2 つのフリップフロップ分順方向にトレースされています。

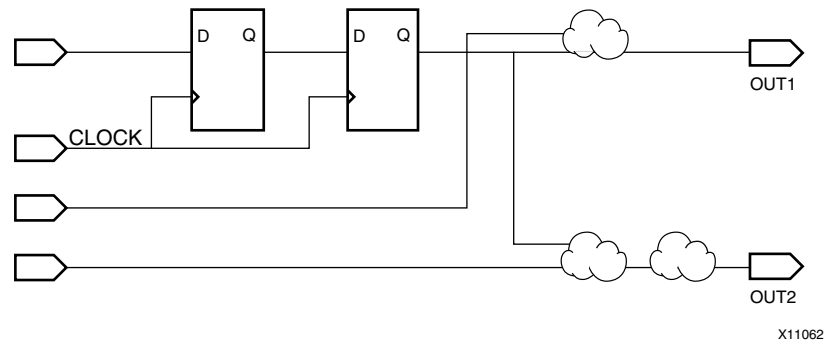


図 3-2: クロック パッドまたはネットの TNM がダウンストリームของフリップフロップまでトレースされる例

共通入力へのフラグ

共通入力は通常次のいずれかになります。

- クロック信号
- クロック イネーブル信号

次をまとめるために共通入力をフラグします。

- フリップフロップ
- ラッチ
- その他の同期エレメント

TNM は、パス上にあるゲート、バッファ、または組み合わせロジックを何個か介し、フリップフロップ、入力ラッチまたは同期エレメントに到達するまで順方向にトレースされます。これらのエレメントは指定された **TNM** またはタイム グループに追加されます。

CLK の TNM が組み合わせロジックを介して同期エレメント (フリップフロップ) までトレース される場合

次の図のように、順方向にトレースされるネットに TNM を使用すると、フリップフロップのグループを作成できます。

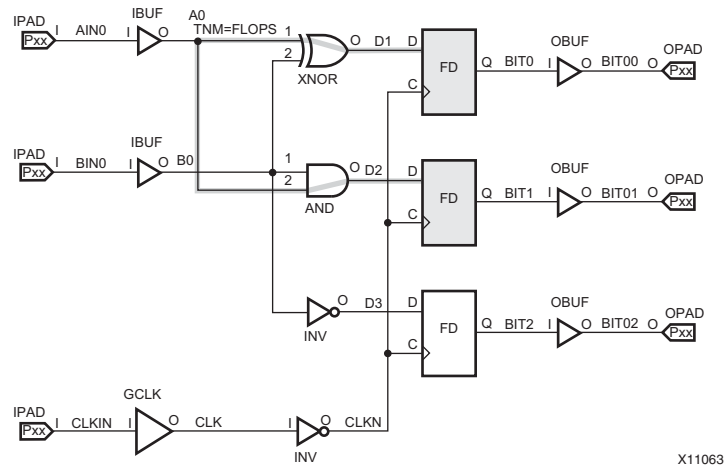


図 3-3 : CLK の TNM が組み合わせロジックを介して同期エレメント (フリップフロップ) までトレース される例

修飾子の使用

ネットに TNM 制約を設定する場合は、タイム グループ内のエレメントのリストを制限するために、修飾子を使用します。修飾子が付いた TNM は、その修飾子タイプと一致する最初の同期エレメントに到達するまで、順方向にトレースされます。修飾子タイプは定義済みのタイム グループです。

同期エレメントのタイプが修飾子と一致すると、同期エレメントにその TNM 制約が適用されます。一致しないにもかかわらず、TNM が同期エレメントを介してトレースされることはありません。

定義済みタイム グループ

次のキーワードは、定義済みのタイム グループです。

FFS

すべてのスライスおよび 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

PADS

すべての I/O パッド

DSPS

- Virtex クロージャ -4 デバイスのすべての DSP48
- Virtex-5 デバイスのすべての DSP48E
- Virtex-6 および 7 シリーズ FPGA デバイスのすべての DSP48E1
- Spartan-6 デバイスのすべての DSP48A1

RAMS

すべてのシングルポートおよびデュアルポートのスライス LUT RAM とブロック RAM

MULTS

Virtex-4 および Virtex-5 デバイスのすべての同期および非同期乗算器

HSIOS

- 次のデバイスのすべての GT および GT10
- Virtex-5 デバイスのすべての GTP
- Virtex-6 デバイスのすべての GTHE1 および GTXE1
- Spartan-6 デバイスのすべての GTPA1
- 7 シリーズ FPGA デバイスのすべての GTHE2 および GTXE2

CPU

- Virtex-4 デバイスのすべての PPC405
- Virtex-5 デバイスのすべての PPC450

LATCHES

すべてのスライス レベルで認識されるラッチ

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

ネットに設定される TNM_NET と TNM 制約の違い

TNM_NET (タイミング名ネット) 制約には、次の特徴があります。

- ネットの TNM と同じです。
- パッド ネットにさまざまな結果を出力します。

TNM 制約とは異なり、変換 (Translate) プロセスまたは NGDBuild コマンドで、TNM_NET 制約が設定されたネットから入力パッドに転送されることはありません。

TNM_NET はネットにのみ使用できます。ピンやインスタンスなどほかのオブジェクトに使用した場合は、次のようになります。

- 警告が表示されます。
- TNM_NET 定義は無視されます。

パッド ネットまたは IPAD と IBUF 間のネットに TNM 制約を設定すると、制約解析ツールは信号またはネットをアップストリームのパッド エレメントまでトレースします。詳細は、[図 3-4](#)、「TNM と TNM_NET の相違点」を参照してください。TNM_NET 制約は、バッファを介して同期エレメントまでトレースされます。

HDL デザインの場合、IBUF 出力信号が IPAD またはポート名と同じなので、TNM_NET と TNM 制約間に違いはありません。この場合、TNM 制約がダウンストリームの同期エレメントまでトレースされます。

TNM_NET 制約の適用ルール

TNM_NET の適用ルールは、次のとおりです。

TNM_NET をパッド ネットに設定した場合

パッド ネットに設定すると、IBUF およびその他の組み合わせロジックを介して同期エレメントまたはパッドに伝搬されます。

TNM_NET をクロック パッド ネットに設定した場合

クロック パッド ネットに設定すると、クロック バッファを介して同期エレメントまたはパッドに伝搬されます。

TNM_NET を一部の入力クロック ネットに設定した場合

DCM/DLL/PLL/PMCD/BUFR の入力クロック ネットに設定し、PERIOD 制約と関連付けると、クロック調整ブロックを介して同期エレメントまたはパッドに伝搬されます。

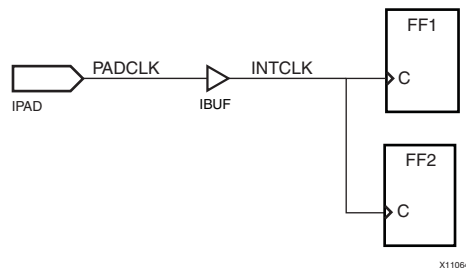


図 3-4 : TNM と TNM_NET の相違点

上記の図に示すデザインでは、IPAD 信号に付けられた TNM に、タイム グループには PAD シンボルのみが含まれます。IPAD 信号に TNM_NET を設定すると、タイム グループに IBUF より後の同期エレメントがすべて含まれます。

IPAD 信号を使用したタイム グループの作成

次の例は、IPAD 信号を使用してタイム グループを作成する方法を複数示しています。

NET PADCLK TNM = PAD_grp;

- タイム グループの PAD_grp を定義するのに padclk ネットを使用します。
- IPAD エレメントを含めます。

NET PADCLK TNM = FFS "FF_grp";

- タイム グループの FF_grp を定義するのに padclk ネットを使用します。
- フリップフロップ エレメントは含めません。

NET PADCLK TNM_NET = FFS FF2_grp;

- タイム グループの FF2_grp を定義するのに padclk ネットを使用します。
- このネットに関連するフリップフロップ エLEMENT すべてを含めます。

前の図に示すデザインでは、IBUF 出力信号に TNM を設定すると、タイム グループに IBUF 後の同期エレメントのみを含めることができます。

IBUF 出力信号のみを含むタイム グループ

次は、IBUF 出力信号のみを含めるタイム グループの例です。

NET INTCLK TNM = FFS FF1_grp;

- タイム グループの FF1_grp を定義するのに intelk ネットを使用します。
- このネットに関連するフリップフロップ エLEMENT すべてを含めます。

NET INTCLK TNM_NET = RAMS Ram1_grp;

- タイム グループの Ram1_grp を定義するのに intelk ネットを使用します。
- このネットに関連する分散 RAM およびブロック RAM エLEMENT すべてを含めます。

インスタンスまたは階層

TNM 制約がモジュールまたはマクロに設定される場合、制約解析ツールはマクロまたはモジュールを下位階層の同期エレメントおよびパッドまでトレースします。

制約は順方向のネットや信号ではなく、すべての階層レベルに適用されます。この機能については、次を参照してください。

- 図 3-2 「クロック パッドまたはネットの TNM がダウンストリームのフリップフロップまでトレースされる例」
- 図 3-3 「CLK の TNM が組み合わせロジックを介して同期エレメント (フリップフロップ) までトレース される例」

TNM 制約

これらの同期エレメントには、同じ TNM 制約が付きます。これで TNM 制約名が TIMESPEC またはタイミング制約で使用されます。この方法では、デザイン ブロックの TNM が使用されます。同じ TNM 制約が付いた複数のインスタンスが使用され、タイム グループが識別されます。

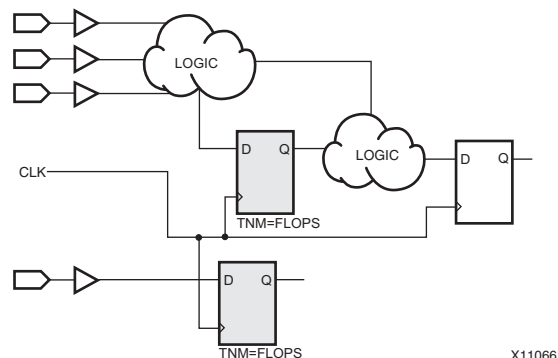


図 3-5：インスタンス別グループ

マクロおよびモジュール

マクロまたはモジュール

- 高度な汎用ファンクションを実行するエレメントです。
- 通常は次を含む下位レベル デザインが含まれます。
 - プリミティブまたはエレメント
 - ほかのマクロまたはモジュール

これらのコンポーネントが組み合わせられ、高度なファンクションがインプリメントされます。

TNM 制約をモジュールまたはマクロに設定すると、そのマクロまたはモジュール内にあるすべてのエレメント (そのマクロまたはモジュールの下位にある全階層レベル) が、指定したタイム グループの一部になります。

keep_hierarchy 制約を使用すると、デザイン階層が維持されます。この機能については、次の図を参照してください。

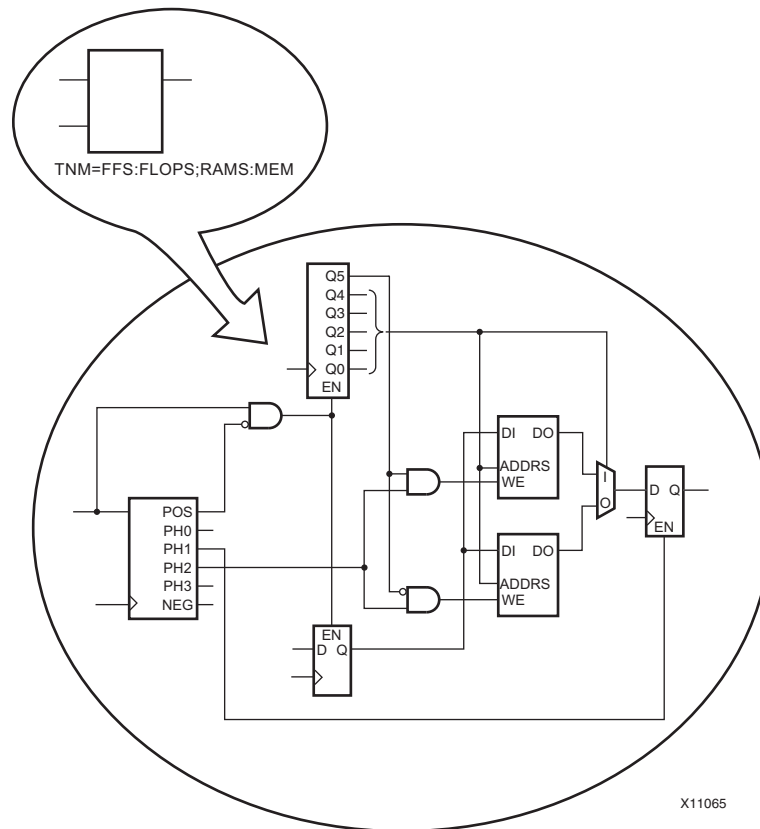


図 3-6：左上の階層の TNM が下位階層レベルのエレメントにトレースされる例

ワイルドカード文字の使用

ワイルドカード文字を使用して、デザイン階層全を表すことができます。

- 疑問符 (?) は 1 文字を表します。
- アスタリスク (*) は複数文字を表します。

次の例では、ワイルドカードを使用して **Level1** が最上位レベル モジュールの階層を示しています。

- **Level1/***

Level1 およびそれより下位レベルのすべてのブロックを表します。

- **Level1/*/***

Level1 のすべてのブロックを表します (下位レベルのブロックは含みません)。

インスタンスは次のいずれかになります。

- 回路図のシンボル、または
- EDIF ネットリストに記載されているシンボル名

デザイン階層全体のワイルドカード

図 3-7、「ワイルドカードを使用した階層の指定」では、次のインスタンスのデザイン階層を表すワイルドカードの例を示しています。

INST *

すべての同期エレメントがこのタイム グループに含まれます。

INST /*

すべての同期エレメントがこのタイム グループに含まれます。

INST /*/*

最上位レベルのエレメントまたはモジュールがこのタイム グループに含まれます。

- **A1**
- **B1**
- **C1**

INST A1/*

A1 階層より 1 レベルまたは複数レベル下位にあるすべてのエレメントがこのタイム グループに含まれます。

- **A21**
- **A22**
- **A3**
- **A4**

INST A1/*/

A1 階層より下位 1 レベルにあるすべてのエレメントがこのタイム グループに含まれます。

- A21
- A22

INST A1/*/*

A1 階層より 2 レベルまたはそれ以上のレベル下位にあるすべてのエレメントがこのタイム グループに含まれます。

- A3
- A4

INST A1/*/*/*

A1 階層より下位 2 レベルにあるすべてのエレメントがこのタイム グループに含まれます。

- A3

INST A1/*/*/*

A1 階層より 3 レベルまたはそれ以上のレベル下位にあるすべてのエレメントがこのタイム グループに含まれます。

- A4

INST A1/*/*/*/*

A1 階層より下位 3 レベルにあるすべてのエレメントがこのタイム グループに含まれます。

- A4

INST /*/*22/

インスタンス名 22 を含むエレメントすべてがこのタイム グループに含まれます。

- A22
- B22
- C22

INST /*/*22

インスタンス名 22 を含むエレメントすべてとそれより 1 レベル下の階層のエレメントがこのタイム グループに含まれます。

- A22
- A3
- A4
- B22
- B3
- C22
- C3

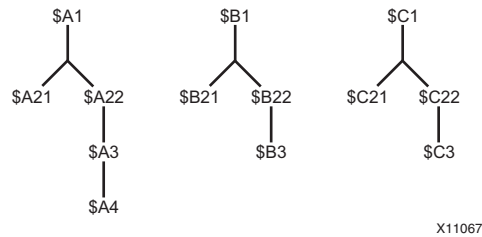


図 3-7: ワイルドカードを使用した階層の指定

インスタンス ピン

ピン接続でグループを識別すると、同期エレメントおよびパッドを駆動するピンを指定して、エレメントをグループ化できます。この方法では、デザインのピンの **TNM** (タイミング名) が使用されます。

TNM 制約がピンに設定される場合、制約解析ツールはそのピンをダウストリートの同期エレメントまでトレースします。**TNM** は、タイミング制約に後で使えるタイム グループを構成するエレメントを識別するための制約です。このタイム グループはタイミング制約で使用できます。詳細は、次の図を参照してください。詳細は、「[修飾子の使用](#)」を参照してください。

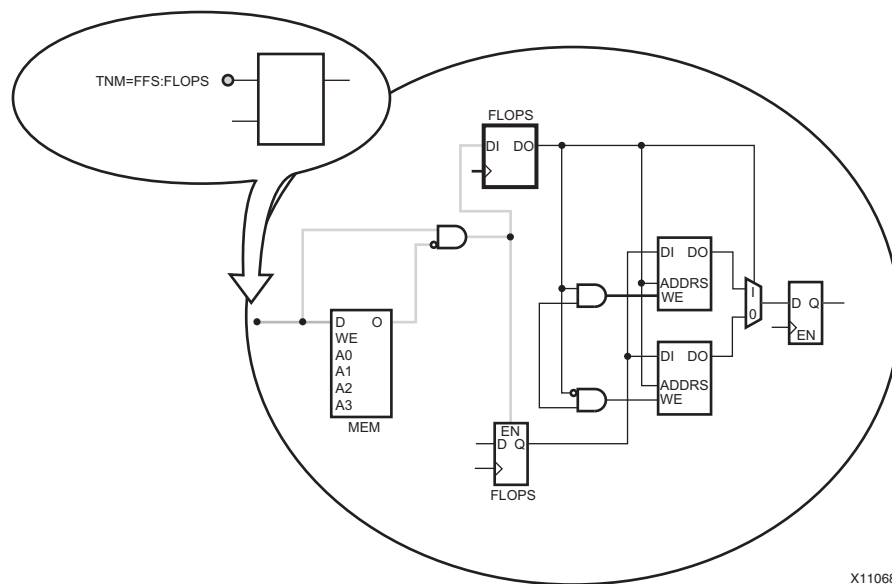


図 3-8: マクロ ピンに設定された TNM がダウストリートの同期エレメントまでトレースされる例

グループ制約

制約のグループを作成すると、同様のエレメントをまとめてタイミング解析に使用できます。制約のグループは、次のファイルで定義できます。

- UCF
- NGC
- EDN
- EDIF
- EDF

タイミング解析は、論理パスに適用されるタイミング制約で実行されます。ロジック パスはパッドと同期エレメントで開始および終了します。

グループ化されたエレメントは、タイミング解析の始点と終点を示します。これらの始点と終点は、定義済みグループ、ユーザー定義グループ、またはその両方に基づいて指定できます。

タイミング グループは、異なる速度で動作したり、異なるタイミング要件を持つロジック グループを識別するのに理想的です。

タイム グループ

タイム グループはタイミング解析で使用されます。ユーザー定義のグループおよび定義済みのタイム グループは、解析される各パスの始点と終点をタイミング解析ツールに伝えるためのものです。

タイム グループは、次の制約で使用されます。

- [Period]
- OFFSET IN
- OFFSET OUT
- FROM:TO (マルチサイクル)
- TIG (タイミング無視)

特定のネットまたはインスタンス名を使用する場合は、完全な階層パス名を使用する必要があります。これにより、インプリメンテーション ツールでネットまたはインスタンスを検出できるようになります。

定義済みタイム グループの修飾子を使用してタイム グループを作成する場合は、文字列一致ワイルドカードを使用して指定できます。これには、タイム グループ修飾子の後に括弧でパターンを記述する方法です。

定義済みタイム グループ

定義済みグループは、次を参照できます。

- フリップフロップ
- ラッチ
- パッド
- RAM
- CPU
- 乗算器
- 高速入力/出力

定義済みグループのキーワードは、グローバルに使用でき、ユーザー定義のサブグループを作成できます。定義済みタイム グループには、次のような特徴があります。

- 予約語です。
- FPGA デバイスの同期エレメントおよびパッドの種類を定義します。

ユーザー定義のタイム グループ

ユーザー定義のタイム グループには、次のような特徴があります。

- 大文字/小文字が区別されます。
- 次と重複することができます。
 - ほかのユーザー定義のタイム グループ
 - 定義済みタイム グループ

デザイン エレメントの例は、複数のタイムグループで構成されています。これらの場合、レジスタは次にあります。

- FFS (定義済みタイム グループ)
- PERIOD 制約の付いた clk タイム グループ

ユーザー定義のタイム グループ

次のキーワードを使用して、ユーザー定義のタイム グループを定義します。

- TNM
- TNM_NET
- TIMEGRP 制約

ユーザー定義のタイム グループ制約が付いたインスタンスまたはネットが内部予約語と同じ場合、そのタイミング グループまたは制約は処理されません。これは、ユーザー定義のタイム グループ名の場合も同様です。

二重引用符

NCF、UCF、PCF 制約ファイルで、インスタンスまたは内部予約語と一致する変数名は、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。

インスタンス名またはネット名が次のような場合は、二重引用符で名前を囲んでください。

- 内部予約語と同じ場合
- チルダ (~) やドル記号 (\$) などの特殊文字が含まれる場合

ザイリンクスでは、すべてのネットおよびインスタンスに二重引用符を使用することをお勧めしています。

TNM および TNM_NET 制約

同じ TNM または TNM_NET で指定されたすべてのエレメントは同じタイム グループとみなされます。

TNM および TNM_NET 制約に関する詳細は、第 3 章「タイミング制約の基礎」の「制約システム」を参照してください。

TIMEGRP 制約

TIMEGRP 制約は、次のために使用します。

- 既存のタイム グループ (定義済みまたはユーザー定義) をまとめます。
- 既存のタイム グループから共通エレメントを削除します。
- 文字列一致により、新しいタイム グループを作成します。

指定した文字列で始まる出力ネットを持つすべてのオブジェクト集合をグループ化します。

既存タイム グループのサブセットの作成

次のキーワードを使用すると、既存タイム グループのサブセットを作成できます。

- EXCEPT
共通エレメントの除外
- RISING
立ち上がりエッジの同期エレメント
- FALLING
立ち下がりエッジの同期エレメント

EXCEPT キーワード

TIMEGRP 制約に EXCEPT キーワードをつけると、既存のタイム グループからエレメントを除外できます。

元のタイム グループから除外する重複アイテムは、削除するか、EXCEPT タイム グループに含める必要があります。

除外タイム グループが元のタイム グループと重複していなければ、デザイン エレメントはなにも削除されません。この場合、新しいタイム グループに元のタイム グループと同じエレメントが含まれます。

RISING および FALLING キーワード

TIMEGRP 制約は、次のために使用できます。

- 複数タイム グループを含有
- 複数タイム グループを除外
- RISING および FALLING キーワードを使用してサブグループを作成

RISING および FALLING は、クロックの立ち上がりエッジまたは立ち下がりエッジでトリガされる同期エレメントに基づいてグループを作成する際に使用します。

文字列一致

ネットまたはインスタンス名の文字列一致方法を使用すると、ユーザー定義のタイム グループを定義できます。

ワイルドカードは次のために使用します。

- 関連するネット名またはインスタンス名が特定文字列に一致するシンボルのユーザー定義タイム グループを定義するため
- 同期エレメントのグループ選択の幅を広げるため
- 同期エレメントへの完全な階層パスを短縮するため

表 3-2：文字列一致

表示	シンボル	一致
アスタリスク	*	任意の数の文字列
疑問符	?	任意の 1 文字

表 3-3：文字列一致の例

文字列	内容	例
DATA*	DATA で始まるネットまたはインスタンス名	DATA1、DATA22、お よ び DATABASE
NUMBER?	NUMBER で始まり 1 文字で終わるネット名	NUMBER1 または NUMBERS (NUNMBER または NUMBER12 は除外)

ワイルドカード文字は、複数使用できます。たとえば、***AT?** は次のネット名を示します。

- AT が後に続く 1 文字または複数文字で始まる
- 任意の 1 文字で終了する

***AT?** には、次のネット名が含まれます。

- BAT2
- CAT4
- THAT9

タイム グループの例

次は、タイム グループの例 6 つを示します。

タイム グループの例 1 (定義済みの RAM グループ)

次は、検索文字列を使用して作成したタイム グループとマルチサイクル制約の定義済み RAM グループの例です。

- **INST my_core TNM = RAMS my_rams;**

このタイム グループ (my_rams) は、階層ブロック my_core の RAM コンポーネントです。

- **TIMESPEC TS01 = FROM FFS TO my_rams 14.24ns;**

- **NET clock_enable TNM_NET = RAMS(address*) fast_rams;**

このタイム グループは、出力ネット名 address* を使用したネット名 clock_enable で駆動される RAM コンポーネントです。

- **TIMESPEC TS01 = FROM FFS TO fast_rams 12.48ns;** または

- **TIMESPEC TS01 = FROM FFS TO RAMS(address*) 12.48ns;**

このデスティネーション タイム グループは、出力ネット名 address* を使用した RAM コンポーネントに基づいています。

タイム グループの例 2 (定義済みの FFS グループ)

次は、検索文字列を使用して作成したタイム グループとマルチサイクル制約の定義済み FFS グループの例です。

```
TIMESPEC TS01 = FROM RAMS TO FFS(macro_A/Qdata?)14.25ns;
```

このデスティネーション タイム グループは、出力ネット名 macro_A/Qdata? を使用したフリップフロップ コンポーネントに基づいています。

タイム グループの例 3 (階層インスタンスの定義済みグループ)

次は、階層インスタンスの定義済みグループを使用して作成されたタイム グループの例です。

- **INST macroA TNM = LATCHES latch_grp;**

このタイム グループ (latch_grp) は、階層インスタンス macroA のラッチ コンポーネントです。

- **INST macroB TNM = RAMS memory_grp;**

このタイム グループ (memory_grp) は、階層インスタンス macroB の RAM コンポーネントです。

- **INST "test_port" TNM = "TEST_GRP" ;**

このタイム グループ (overall_grp) は、階層インスタンス tester の同期コンポーネント (RAM、フリップフロップ、ラッチ、パッドなど) です。

タイム グループの例 4 (タイム グループの統合)

次の例は、ほかのタイム グループと統合して新しいタイム グループを定義する方法を示しています。

- **TIMEGRP "larger_grp" = "small_grp" "medium_grp";**
small_grp および medium_grp を統合して larger_grp というグループを作成します。
- **TIMEGRP memory_and_latch_grp = latch_grp memory_grp;**
latch_grp と memory_grp のエレメントが統合されます。

タイム グループの例 5 (タイム グループの削除)

次は、TIMEGRP 制約に EXCEPT キーワードを使用した例です。

- **TIMEGRP new_time_group = Original_time_group EXCEPT a_few_items_time_grp;**
Original_time_group から a_few_items_time_grp のエレメントを削除します。
- **TIMEGRP "medium_grp" = "small_grp" EXCEPT "smaller_grp";**
 - small_grp のエレメントから medium_grp タイム グループを作成し、
 - smaller_grp のエレメントを削除します。
- **TIMEGRP all_except_mem_and_latches_grp = overall_grp EXCEPT memory_and_latch_grp;**
memory_and_latch_grp と overall_grp 間に共通のエレメントを削除します。

タイム グループの例 6 (クロック エッジ)

次は、クロック エッジのトリガに基づいてサブグループを定義する例です。

- **TIMEGRP "rising_clk_grp" = RISING clk_grp;**
 - rising_clk_grp タイム グループを作成します。
 - clk_grp の立ち上がりエッジの同期エレメントすべてを含めます。
- **TIMEGRP "rising_clk_grp" = FALLING clk_grp;**
 - rising_clk_grp タイム グループを作成します。
 - clk_grp の立ち下がりエッジの同期エレメントすべてを含めます。

制約の優先順位

タイミング解析ツールでは、デザインの解析中にどの制約がどのパスを解析するのか決定されます。制約は、種類によって優先度が異なります。

優先順位

次は、制約の優先度を高い順から示しています。

- TIG (タイミング無視)
- 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
 - 特定のデータ IOB (NET OFFSET)
 - データ IOB コンポーネントのタイム グループ (グループ化された OFFSET)
 - すべてのデータ IOB コンポーネント (グローバル OFFSET)
- [Period]

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 ファイルでどの制約が後に表示されているかによって決まります。

- MAXSKEW および MAXDELAY

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。

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。詳細は、次の図を参照してください。

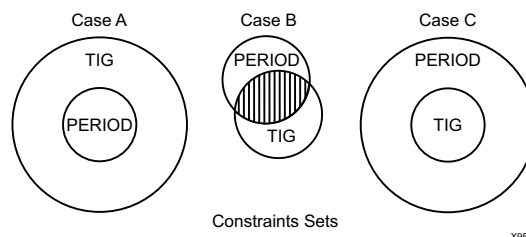


図 3-9：制約集合同士の相互関係

- ケース A の場合、タイミング無視 (TIG) 上位集合が 周期 (PERIOD) 設定と競合します。
- ケース B の場合、PERIOD と TIG 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 タイミング無視 (TIG) と判断されたり 周期 (PERIOD) と判断されたりします。

同じパスに適用される PERIOD 制約が 2 つある場合

同じパスに適用される PERIOD 制約が 2 つある場合は、次のように処理されます。

- 1 つ目の PERIOD 制約については、タイミング レポートで「0 paths analyzed」と表示されます。
- 2 つ目の PERIOD 制約ではパスが解析されます。

タイミング解析ツールで 2 つ目の PERIOD 制約ではなく 1 つ目の PERIOD 制約が使用されるようにするには、次のいずれかを実行します。

- PERIOD 制約に PRIORITY キーワードを使用します。
- マルチサイクルまたは FROM:TO 制約を使用します。

同じ制約タイプ内で優先順位を付けたり、同じパスに適用される 2 つのタイミング制約間の競合を避ける場合は、PRIORITY キーワードを値付きで使用する必要があります。

- PRIORITY の値は、-255 ～ +255 にできます。
- 値が小さいほど優先順位が高くなります。
- この値で、どのパスが最初に配置配線されるか決まるわけではありません。
- 同じ優先度の 2 つのタイミング制約が設定されている場合に、どの制約がそのパスに適用され、解析されるのかが決まるだけです。

PRIORITY キーワードの付いた制約が常に優先されます。

タイミング制約の優先順位構文

タイミング制約の優先順位を定義するには、次の構文を使用します。

- **TIMESPEC TS_01 = FROM A_grp TO B_grp 10 ns PRIORITY 5;**
TS_01 の優先度は TS_02 よりも低くなります。
- **TIMESPEC TS_02 = FROM A_grp TO B_grp 20 ns PRIORITY 1;**

PRIORITY キーワードの使用

PRIORITY キーワードには、次のような特徴があります。

- PRIORITY キーワードは、TS03 などの TSidentifiers の付いた TIMESPEC 制約にのみ設定できます。
- MAXDELAY、MAXSKEW、および OFFSET 制約には設定できません。

BUFGMUX コンポーネントの PRIORITY

これは、DCM からの 2 つのクロック信号が同じ BUFGMUX に駆動される場合に発生する可能性があります。詳細は、次の図を参照してください。

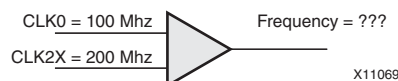


図 3-10 : BUFGMUX コンポーネントの PRIORITY

PRIORITY キーワードを使用した PERIOD 制約の例

次は、PRIORITY キーワードを使用した PERIOD 制約の例です。

```
TIMESPEC "TS_Clk0" = PERIOD "clk0_grp" 10 ns HIGH 50% PRIORITY 2;
TIMESPEC "TS_Clk2X" = PERIOD "clk2x_grp" TS_Clk0 / 2 PRIORITY 1;
```

タイミング制約

タイミング制約は、タイミング目標を達成するための基本要素です。グローバル タイミング制約を使用すると、制約を設定可能なパスすべてに適用されるタイミング条件を設定できます。

次を実行するには、グローバル制約を作成するのが一番簡単な方法です。

- 制約を付けることができる接続すべてに適用します。
- インプリメンテーション ツールですべてのパスのタイミング要件が満たされるようにします。

グローバル タイミング制約ではデザイン全体に制約が設定されます。

基本的なタイミング制約

次は、すべてのデザインに必要な基本的なタイミング制約です。

- 各クロックに PERIOD 制約を使用したクロック定義
同期エレメント パスへの同期エレメントの制約
- グローバル OFFSET IN 制約を使用した入力要件
同期エレメント パスへの入力のインターフェイスを制約
- グローバル OFFSET OUT 制約を使用した出力要件
出力からパスへの同期エレメントのインターフェイスを制約
- pad-to-pad 制約を使用した組み合わせパス要件

マルチサイクル パスまたはスタティック パスにはさらに特別なパス制約を使用できます。

- マルチサイクル パスとは、タイミング要件付きの 2 つのレジスタまたは同期エレメント間のパスのことです。この要件はレジスタまたは同期エレメントのクロックの PERIOD 制約の倍数となります。
- スタティック パスには、pad-to-pad パスなどクロックが送信されるエレメントは含まれません。

タイミング制約の例外

基本的なタイミング制約を設定したら、例外を指定する必要があります。

表 3-4：タイミング制約の例外

設定先	使用する制約
PERIOD	<ul style="list-style-type: none"> FROM:TO (マルチサイクル) 制約
グローバル OFFSET	<ul style="list-style-type: none"> パッド タイム グループ ベースの OFFSET 制約 ネット ベースの OFFSET 制約

タイミング制約要件の設定

次の事項に従うことをお勧めします。

- パスに必要となるタイミング要件値を正確に指定します。
- タイミング制約の要件を厳しくしすぎないでください。

制約条件を厳しくすると、次のような結果になることがあります。

- 配置配線 (PAR) またはインプリメンテーションのランタイムが長くなる
- メモリ使用率が増加する
- 結果の質 (QOR) が落ちる

PERIOD 制約

PERIOD (クロック周期仕様) 制約は、基本的なタイミングおよび合成制約です。

PERIOD 制約には、次のような特徴があります。

- デザイン内で各クロックを定義します。
- 各クロック ドメイン内のすべての同期パスに適用されます。
- 関連するクロック ドメイン間のクロック ドメイン パスをチェックします。
- クロックの期間を定義します。
- 異なるデューティ サイクルになるように設定できます。
- **PERIOD** 制約は、**FROM:TO** 制約よりもよく使用されます。
 - パスのほとんどに適用されます。
 - インプリメンテーション ツールのランタイムを削減できます。

クロックの PERIOD 制約

クロックの **PERIOD** 制約では、次が定義されます。

- レジスタを介したクロック ピンで終了する特定のクロック ネットからクロックを供給される同期エレメント (フリップフロップ、RAM、ラッチ、HSIOS、CPU、および DSP) 間のタイミングが定義されます。詳細は、次の図を参照してください。
- デスティーネーション クロック ドメインに基づいた関連するクロック ドメイン間のタイミング

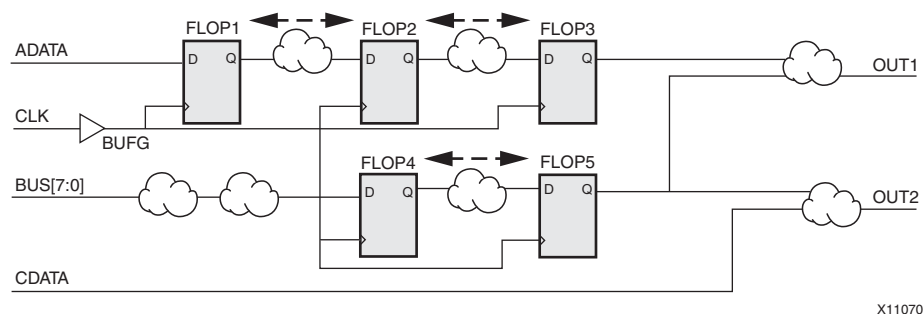


図 3-11 : register-to-register パスに適用される PERIOD 制約

クロック ネットの PERIOD 制約

PERIOD 制約をクロック ネットに設定すると、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解析が設定されたピンが終端となるすべてのパスで、遅延が解析されます。通常の解析には、次のデータ パスが含まれます。

- 同期エレメントの clock-to-out 遅延
- 配線およびロジック遅延
- 同期エレメントのセットアップ/ホールド遅延
- ソースおよびデスティネーションの同期エレメント間のクロック スキュー
- DCM 位相および負のエッジのクロックを含むクロック位相
- クロックのデューティ サイクル

PERIOD 制約に含有されるもの

PERIOD 制約には、次が含まれます。

- グローバル クロックおよびローカル クロックのクロック スキュー解析のクロック パス遅延
- ローカルクロック反転
- セットアップおよびホールド解析
- 関連するクロック間の位相関係

関連または派生するクロックは別のクロックのファンクションにできます (* および /)。

- クロックのばらつきとして、Virtex-4 の場合は DCM ジッタ、デューティ サイクルの歪み、DCM 位相エラー、Virtex-5 およびそれ以降のデバイスでは DCM ジッタ、PLL ジッタ、デューティ サイクルの歪み、DCM 位相エラー
- クロックのばらつきとしてユーザー定義のシステムおよびクロック入力ジッタ
- 等しくないクロック デューティ サイクル (50% ではない)
- DCM 位相および負のエッジのクロックを含むクロック位相

関連する TIMESPEC PERIOD 制約

ザイリンクスでは、すべてのクロックに PERIOD 制約を付けることをお勧めしています。PERIOD 制約の定義には、TIMESPEC PERIOD 制約がよく使用されます。TIMESPEC を使用すると、ほかの TIMESPEC PERIOD 制約との派生クロックの関係を定義できます。

この複雑な派生関係の例は、「[DLL/DCM/PLL/BUFR/PMCD/MMCM コンポーネント](#)」コンポーネント出力を介して実行されます。派生関係は、TIMESPEC PERIOD 同士の関係を使用して定義されます。データ パスが 1 つのクロック ドメインから別のクロック ドメインを通り、PERIOD 制約が関連付けられている場合、タイミング ツールではクロック ドメインを横切るパスの解析が実行されます。これは、「[DLL/DCM/PLL/BUFR/PMCD/MMCM コンポーネント](#)」からの出力でよく発生する状況です。

TNM および TNM_NET 制約に関する詳細は、第 3 章「タイミング制約の基礎」の「[制約システム](#)」を参照してください。

関連する PERIOD 制約のクロス クロック ドメインパスの解析中は、デスティネーション エレメントの PERIOD 制約がデータ パスに適用されます。

関連する PERIOD 制約

次の図では、TS_PERIOD#1 が TS_PERIOD#2 に関連付けられています。データパスは TS_PERIOD#2 で解析されます。

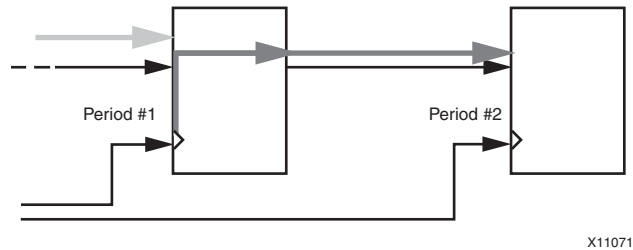


図 3-12 : 関連する PERIOD 制約

PERIOD 制約が互いに関連する場合、デザイン ツールで、内部クロック ドメイン パスの要件を決定できます。詳細は、次の図を参照してください。

PERIOD 制約構文

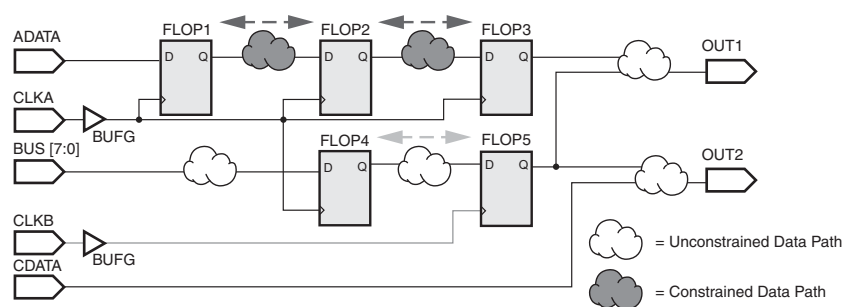
次は、PERIOD 制約構文の例です。TS_Period_2 制約値は、**TS_Period_1 TIMESPEC** を乗算した値になります。

```
TIMESPEC TS_Period_1 = PERIOD "clk1_in_grp" 20 ns HIGH 50%;
TIMESPEC TS_Period_2 = PERIOD "clk2_in_grp" TS_Period_1 * 2;
```

2 つの PERIOD 制約がこの方法で関連付けられていない場合、クロック ドメイン間を横切るデータパスには PERIOD 制約が適用されないか、パスが解析されません。

関連しないクロック ドメイン

次の図では、CLKA と CLKB が関連していないか、互いに非同期なので、レジスタ 4 とレジスタ 5 の間のデータパスがどちらの PERIOD 制約でも解析されません。



X11072

図 3-13 : 関連しないクロック ドメイン

PERIOD 制約が適用されるパス

PERIOD 制約は、同期エレメント間のパスにのみ適用されます。

次は、この解析には含まれません。

- パッド

PERIOD タイム グループにパッド エレメントが含まれる場合は、NGDBuild により警告メッセージが表示されます。

- 関連しない、または非同期のクロック ドメイン間の解析

PERIOD 制約解析には、同期エレメントのセットアップおよびホールド解析が含まれます。

セットアップ解析

セットアップ解析をすると、クロックの到着前のデスティネーション同期エレメントのデータ変更が確認できますこのデータは、ピンのアクティブ クロック エッジの到着より前に、入力ピンで少なくともセットアップ タイムで有効になる必要があります。

セットアップ解析の式

セットアップ解析の式は、次のようになります。

$$\text{セットアップ タイム} = \text{データ パス遅延} + \text{同期エレメントのセットアップ タイム} - \text{クロック パス スキュー}$$

セットアップ解析のタイミング レポート

タイミング レポート解析には、クロックのばらつき (Clock Uncertainty) が含まれ、セットアップ解析のスラック値が表示されます。データ パスには、データ パス遅延と同期エレメントのセットアップ タイムが含まれます。

$$\text{スラック} = \text{要件} - (\text{データ パス} - \text{クロック パス スキュー} + \text{クロックのばらつき})$$

セットアップ解析のクロックのばらつき

クロックのばらつきが増えると、セットアップ差が減ります。詳細は、次の図を参照してください。

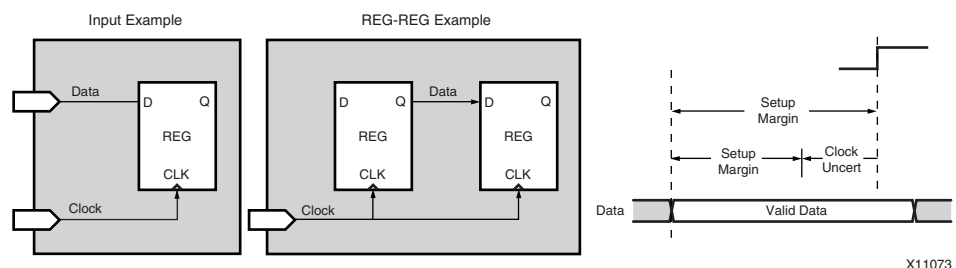


図 3-14：クロックのばらつき/ジッタによるセットアップ差の削減

ホールド解析

ホールド解析をすると、クロックの到着後にデスティネーションの同期エレメントのデータが変更できます。このデータは、ピンのアクティブ クロック エッジの到着後に、入力ピンで少なくともホールド タイムで有効なままである必要があります。

ホールド解析の式

ホールド解析の式は、次のようになります。

ホールド タイム = クロック スキュー + 同期エレメントのホールド タイム - データ パス遅延

ホールド タイム違反は、正のクロック スキューがデータ パス遅延よりも大きいと発生します。

ホールド解析のタイミング レポート

タイミング レポート解析には、次が表示されます。

- クロックのばらつき (Clock Uncertainty) が含まれます。
- ホールド解析のスラック値が決定されます。

データ パスには、次が含まれます。

- データ パス遅延
- 同期エレメントのホールド タイム

スラック = 要件 - (クロック パススキュー + クロックのばらつき - データ パス)

ホールド解析のクロックのばらつき

クロックのばらつきが増えると、ホールド差が減ります。詳細は、次の図を参照してください。

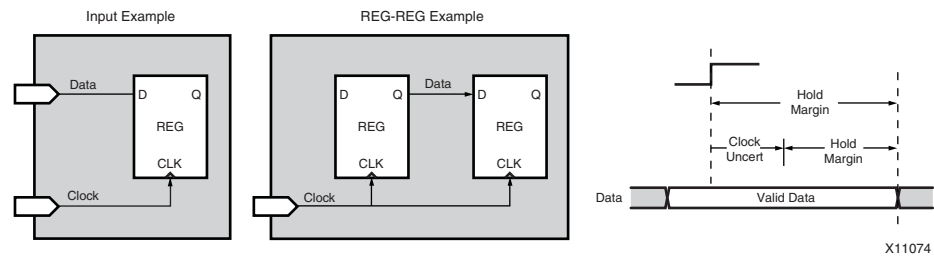


図 3-15 : クロックのばらつき/ジッタによるホールド差の削減

どちらの式にも、同期ソース エレメントの Clock-to-Out タイムがデータ パス遅延の一部として含まれています。

次の図では、正のクロック スキューがデータ パス遅延よりも大きいので、タイミング解析でホールド違反が表示されます。

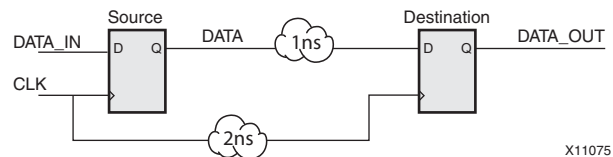
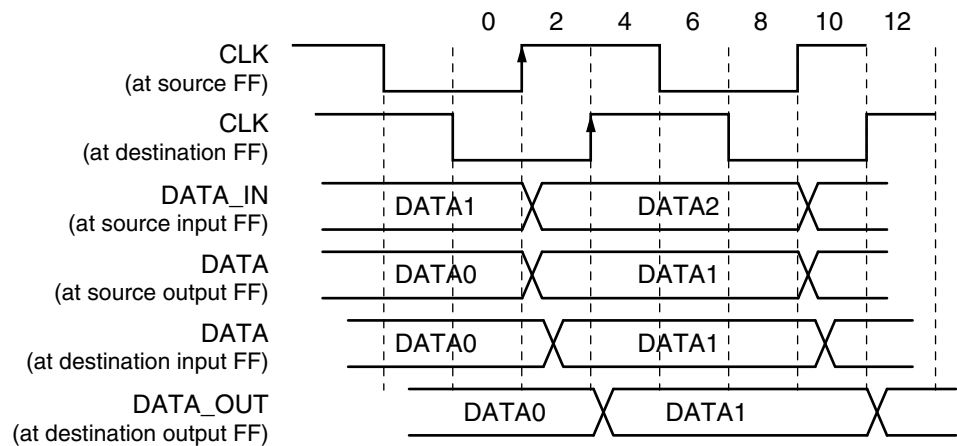


図 3-16 : ホールド違反 (クロック スキュー > データ パス)



X11076

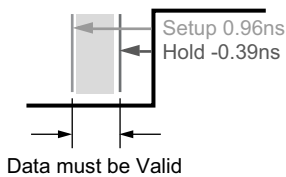
図 3-17：ホールド違反の波形

タイミング レポートには、パスがホールド違反の原因でない限り、ホールド パスをリストされません。

各制約のホールド パスをレポートするには、`trce` で `-fastpaths` オプションを使用するか、**Timing Analyzer** の **[Report Fast Paths]** をオンにします。次の図は、デバイスのデータ シートからのセットアップ タイムとホールド タイムの例を示しています。タイミング レポートのセットアップおよびホールド タイムは、通常デバイスのデータシートよりも少ない値になります。

データシートの値はすべてのピンおよび同期エレメントに適用されますが、タイミング レポートは特定ピンまたは同期エレメントなど、ユーザー デザインに特有のレポートになります。

OFFSET 制約



Setup and Hold Times With Respect to Clock at IOB input Register							
Pad, no delay	T_{iopick}/T_{ioickp}	All	0.88/-0.39	0.96/-0.39	1.11/0.45	ns, min	

X11077

図 3-18 : データシートのセットアップ/ホールド タイム

OFFSET 制約には、次のような特徴があります。

- 基本的なタイミング制約です。
- 次の 2 つのタイミング関係を定義するのに使用します。
 - 外部クロック パッド
 - 関連するデータ入力/データ出力パッド

この関係は、デバイスの pad-to-setup または clock-to-out パスへの制約でもあります。

外部コンポーネントを含むタイミング インターフェ이스の指定

次の制約は、外部コンポーネントを含むタイミング インターフェイスを指定する際に重要な制約です。

- pad-to-setup (OFFSET IN BEFORE) 制約
外部クロックと外部入力データが内部フリップフロップのセットアップ タイムを満たすことができます
- clock-to-out (OFFSET OUT AFTER) 制約
外部データ パッドと外部クロック パッドに関して、ダウンストリーム デバイスのセットアップ/ホールド要件をユーザーがさらに制御できます。
- OFFSET IN BEFORE および OFFSET OUT AFTER 制約
クロックに対する入力パッドからの内部データ遅延、または出力パッドへの内部データ遅延を指定できます。

外部データとクロックの関係の指定

また、OFFSET IN BEFORE および OFFSET OUT AFTER 制約では、ザイリンクス デバイスの入力パッドおよび出力パッドへのパスのタイミング用に、外部データとクロックの関係を指定することもできます。

タイミング ソフトウェアでは、次の制約のどちらかを設定しなくても、内部条件が決定されます。

- FROM PADS TO FFS
- FROM FFS TO PADS

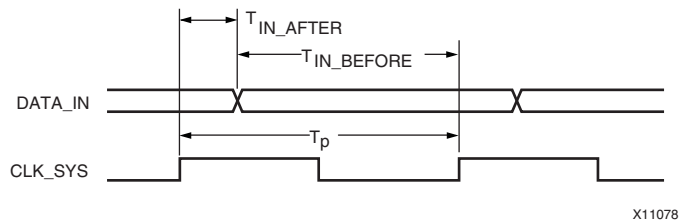


図 3-19 : OFFSET IN 制約のタイミング図

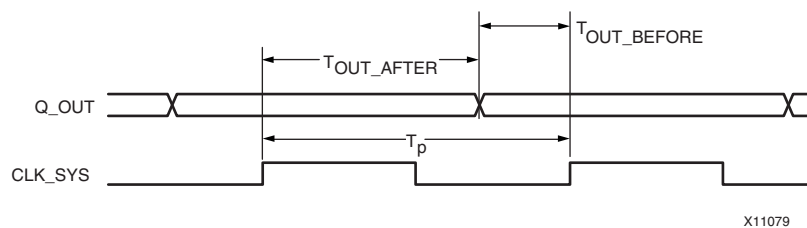


図 3-20 : OFFSET OUT 制約のタイミング図

OFFSET 制約に含有されるもの

OFFSET 制約には、次のような特徴があります。

- 解析で各同期エレメントに対するクロック パス遅延を含めます。
- 同期エレメントのすべてのタイプ (FFS、RAMS、LATCHES など) のパスを含めます。
- すべての入力または出力を 1 つの外部クロックに関連付けるグローバル構文を使用できます。
- 入力のセットアップおよびホールド タイム違反を解析します。

クロック パス遅延

OFFSET 制約では、PERIOD 制約を使用して定義および解析される場合、次のクロック パス遅延が自動的に考慮されます。

- 正確なタイミング情報を提供し、関連する PERIOD 制約に定義されたジッタを使用します。
- 同期エレメントに到着するように、入力信号の時間を増加します (クロック パスとデータ パスはパラレル)。
 - 入力のデータ パス遅延からクロック パス遅延を引きます。
- 出力ピンに到着するように、出力信号の時間を削減します (クロック パスとデータ パスはシリアル)。

- クロック パス遅延を出力のデータ パス遅延に足します。
- 関連する **PERIOD** 制約で定義された各同期エレメントに対して **DLL/DCM** コンポーネントを使ってクロック位相を含めます。
- 立ち上がりまたは立ち下がりクロック エッジによるクロック位相を含めます。

最初のクロック エッジ

OFFSET 制約解析用の最初のクロック エッジは、**PERIOD** 制約の **HIGH/LOW** キーワードを使用して定義します。

- **HIGH** キーワード => 最初のクロック エッジは立ち上がりエッジ
- **LOW** キーワード => 最初のクロック エッジは立ち下がりエッジ

OFFSET 制約解析用の最初のクロック エッジは、次の **OFFSET** 制約のキーワードを使用すると、**PERIOD** 制約デフォルトのクロック エッジよりも優先されます。

- **RISING** キーワード => 最初のクロック エッジは立ち上がりエッジ
- **FALLING** キーワード => 最初のクロック エッジは立ち下がりエッジ

外部クロック パッドおよび外部データ パッド

OFFSET 制約は、次のパッド同士の関係を定義します。

- 外部クロック パッド
- 外部データ パッド

同期エレメント

外部クロック パッドと外部データ パッド間の共通コンポーネントは、同期エレメントです。同期エレメントが内部クロック ネットで駆動される場合は、このデータ パスを解析するのに **FROM:TO** 制約が必要となります。

DCM/PLL/DLL/PMCD/BUFR で生成される内部クロックには、この規則は適用されません。

FROM:TO 制約

FROM:TO 制約を使用すると、次の状況の **OFFSET** 制約と同様の解析になります。

- 内部ネットからデータ入力とクロック入力が供給される同期エレメントでセットアップ タイムが違反していないかを計算します。
- 内部ネットからクロックが供給される内部同期エレメントの **Q** 出力から生成された外部出力ネットの遅延を指定します。

OFFSET 制約の適用されるパス

OFFSET 制約は、次のパスに適用されます。

- 入力パッドから同期エレメント (**OFFSET IN**)
- 同期エレメントから出力パッド (**OFFSET OUT**)

詳細は、次の図を参照してください。

同期エレメントにクロックを供給するクロック ネットが入力パッドを介していない場合 (ほかのクロックや同期エレメントから供給される場合など)、**OFFSET** 制約はタイミング解析中どのパスにも適用されません。

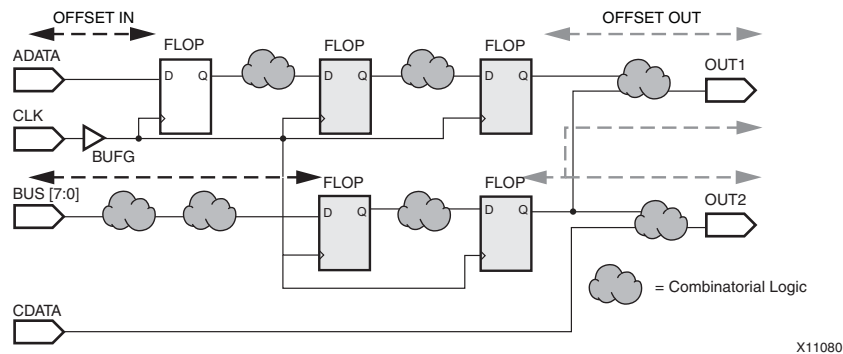


図 3-21 : OFFSET 制約の回路図

OFFSET 制約は、1 つのクロック エッジに対してのみ解析されます。**OFFSET** 制約でソース同期のデザインやデュアル データ レートのアプリケーションのように複数のクロック位相やクロック エッジを解析する必要がある場合、**OFFSET** 制約をクロック位相で手動で調整する必要があります。

OFFSET 制約は内部で生成されたクロックが供給されるパスは最適化しません。このようなパスには、クロック遅延を考慮に入れて **FROM:TO** またはマルチサイクル制約を使用します。

I/O タイミング解析

内部クロックまたは派生クロックの I/O タイミング解析を取得するには、次のオプションを使用します。

- これらのパスに **FROM:TO** またはマルチサイクル制約を作成します。
- 内部クロックが外部クロック信号と関連するかどうかを決定します。
- 2 つのクロック間の関係に基づいて条件を変更します。

次はその例です。

- 内部クロックが 2 つの外部クロック バージョンで分周
- 内部クロック付き **OFFSET OUT** の元の条件が 10 ns

外部クロック付き **OFFSET OUT** の条件は 20 ns になります。

適用範囲

OFFSET 制約の適用範囲は、次のように 3 種類あります。

- グローバル **OFFSET**
特定のクロックで、すべての入力または出力に対して全体的に指定する
- グループ **OFFSET** 制約
同じタイミング条件で、共通クロックが使用される入力または出力のグループを識別する
- ネット用 **OFFSET**
入力または出力ごとにタイミングを指定する

グループおよびグローバル OFFSET 制約

OFFSET 制約では、グローバルな対象よりも特定の対象が優先されます。

- 同じ I/O に指定する場合でも、グループ OFFSET 制約は、グローバル OFFSET 制約よりも優先されます。
- ネット用の OFFSET 制約は、グローバル OFFSET 制約やグループ OFFSET 制約よりも優先されます。

このような優先順位のルールがあるため、まずグローバル OFFSET 制約を作成してから、特定のタイミング要件がある I/O に対してグループまたはネット用の OFFSET 制約を作成します。

メモリ使用率およびランタイムの削減

グローバル OFFSET およびグループ OFFSET 制約を使用すると、メモリ使用率およびランタイムが削減できます。

ネット用 OFFSET 制約でワイルドカードを使用すると、グループの OFFSET 制約ではなくても、複数のネット用 OFFSET 制約を作成できます。

レジスタ グループおよびパッド グループ

グループ OFFSET 制約には、レジスタ グループとパッド グループの両方を含めることができます。パッドまたはレジスタ、もしくは両方ともを同じ条件を使用してグループにまとめることができます。

- レジスタ グループ

レジスタ グループは、クロック エッジの 1 つのパッドへの異なる条件を持つパス ソースまたはデスティネーションを識別するために使用できます。

- パッド グループ

パッド グループは、同じクロック エッジで異なる要件を持つパッドのグループへのパス ソースまたはデスティネーションを識別するために使用できます。

パッドとレジスタを 1 度に同じグループに含めることもできます。この方法は、クロックが入力および出力で立ち上がりエッジと立ち下りエッジに使用されている場合に便利です。

RISING および FALLING グループ

立ち上がりグループと立ち下りグループには、異なるグループ OFFSET 制約が必要です。

次の図の場合、レジスタ A、B、C が同じデータおよびクロック ソースを共有していますが、タイムグループは異なっています。

次は、この異なるタイム グループです。

- TIMEGRP AB = RISING FFS;
- TIMEGRP C = FALLING FFS;

これにより、これらのレジスタに対して 2 種類のタイミング解析を実行できます。

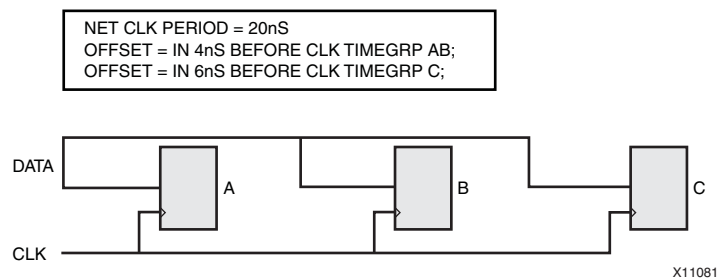


図 3-22：異なるタイムグループの OFFSET

CPLD デザイン

CPLD デザインの場合、OFFSET 制約で参照されるクロック入力、BUFG シンボルを使用するか、通常の入力に **BUFG=CLK** 制約を適用して、グローバル クロック ピンに明示的に割り当てる必要があります。これを実行しないと、タイミングドリブンで最適化するときに **OFFSET** が使用されません。

FROM:TO (マルチサイクル) 制約

マルチサイクルパスは、複数のクロック サイクルを使用できるパスです。

通常はデフォルトで **PERIOD** 制約が適用されますが、**PERIOD** 制約は単一クロック サイクルの制約なので、エラーになることがあります。

これらのエラーを回避するには、パスに特定のマルチサイクル制約を指定し、**PERIOD** 制約からのパスを削除します。

マルチサイクル制約

マルチサイクル制約には、次のような特徴があります。

- **FROM:TO** 制約を使用して適用します。
- 優先順位が **PERIOD** および **OFFSET** 制約よりも高い優先度の低い制約からパスを取り出し、パスをマルチサイクル制約で解析します。
- 優先度の低い制約よりも、厳しくしたり、緩くしたりできます。
- 特定のパスに制約を適用します。

特定のパスは、次のようにできます。

- 同じクロック ドメイン内にできます。
- ただし、**PERIOD** 制約とは条件が異なります。

また、クロック ドメイン間をまたがるデータ パスを含む特定パスには、マルチサイクル制約を使用します。

FROM:TO 制約

FROM:TO 制約には、次のような特徴があります。

- 優先順位が **PERIOD** 制約よりも高い
- **PERIOD** から **FROM:TO** 制約への特定パスを削除
- **FROM:TO** 制約は同期エレメントで開始し、同期エレメントで終了

たとえば、デザインの一部が **PERIOD** 制約の条件よりも遅く実行される必要がある場合、**FROM:TO** 制約を使用します。

マルチサイクルパスは、イネーブルされた各クロック エッジ間に複数のクロック サイクルがあることも意味します。

開始点と終了点の宣言

FROM:TO 制約を使用する場合は、開始点と終了点を宣言して、制約付きのパスを指定する必要があります。

この開始点と終了点は、次のいずれかにする必要があります。

- 指定済みのタイムグループ (**PADS**、**FFS**、**LATCHES**、**RAMS** など)
- ユーザー指定のタイムグループ
- ユーザー指定の同期ポイント (**TPSYNC** 参照)

FROM または TO (オプション)

特定パスに制約を付ける場合、**FROM** または **TO** はオプションです。

- **FROM** (マルチサイクル) 制約

ソースのタイムグループから次の同期エレメントまたはパッド エレメントまでに適用されます。

- **TO** (マルチサイクル) 制約

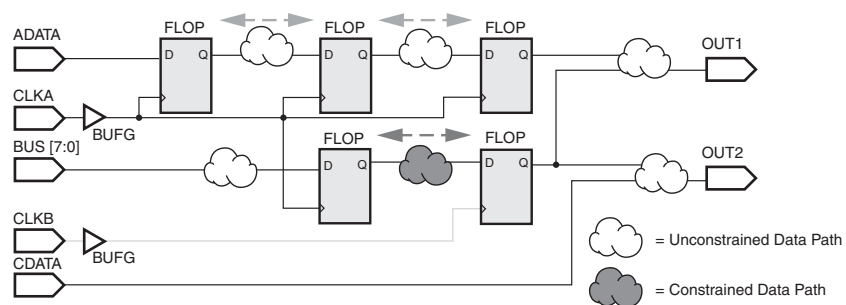
すべての前の同期エレメントまたはパッド エレメントからデスティネーション タイムグループまでに適用されます。

使用できる組み合わせは、次のとおりです。

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

FROM:TO 制約は、クロック ドメイン間のパスに適用されるマルチサイクルパスに適用できます。たとえば、次の図は、1 つのクロックがデザインの一部に適用され、もう 1 つのクロックが残りのデザインに適用されているのに、これらの 2 つのクロック ドメイン間に複数のパスがある例を示しています。

デザインの特性を明確に理解し、複数クロック ドメインを考慮するようにしてください。



X11082

図 3-23：クロス クロック ドメイン パスへ適用されるマルチサイクル制約

クロス クロック ドメイン パス

関連のない **PERIOD** 制約間のクロス クロック ドメイン パスは、制約なしのパス (**Unconstrained Paths**) レポートで解析されます。

これらのパスが間違っていて関連付けられている場合や、異なるタイミング条件が必要とされる場合は、マルチサイクルまたは **FROM:TO** 制約を作成します。

FROM:TO 制約には、別の TIMESPEC 識別子または TIG (タイミング無視) に関連する特定の値を指定できます。TIG を付けると、そのパスをタイミング解析で無視できます。

制約の定義によりクロックが関連付けられていなくても、クロック間に有効なパスがある場合は、FROM TO 制約を使用します。

2 つのクロック ドメイン間のパスへの制約の設定

2 つのクロック ドメイン間のパスに制約を付けるには、次を実行します。

1. 各クロック ドメインに基づいたタイムグループを作成
2. 2 つのクロック ドメイン間を通るパスの方向それぞれに対して FROM:TO 制約を作成

次は、FROM:TO 制約を使用したクロス クロック ドメインの例です。

```
TIMESPEC TS_clk1_to_clk2 = FROM clk1 TO clk2 8 ns;
```

タイムグループ `clkA` からタイミング グループ `clkB` までに 8 ns の制約を付けています。詳細は、次の図を参照してください。

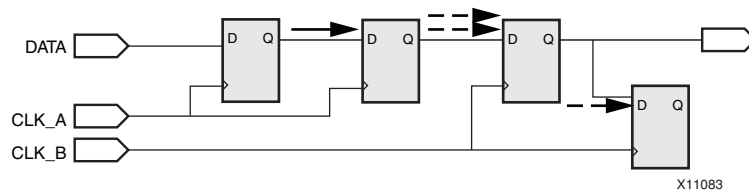


図 3-24 : CLK_A クロック ドメインと CLK_B クロック ドメイン間を横切るパス

pad-to-pads パス

基本的な FROM:TO 制約の 1 つは、pad-to-pads パス (またはデザインの非同期パス) です。FROM:PADS:TO:PADS 制約は、開始点と終了点と同様に、デザインのパッド インスタンスを使用した組み合わせパスに適用されます。これらのタイプのパスには、パスが非同期であるため、通常は制約を付けません。詳細は、次の図を参照してください。

次は、このタイプの制約の例です。

```
TIMESPEC TS_Pad2Pad = FROM PADS TO PADS 14.4 ns;
```

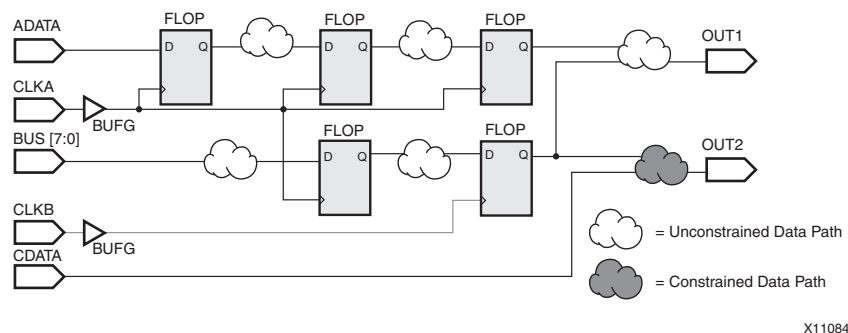


図 3-25 : pad-to-pad マルチサイクル制約の適用されるパス

Slow 例外

マルチサイクル制約は、pad-to-pad パスで使用するだけでなく、デザインの一部に例外を付けて遅くするため (Slow 例外) にも使用できます。これはデザインの大部分に適用される PERIOD 制約からの例外です。

PERIOD 制約と FROM:TO の Slow 例外の併用

次の図は、PERIOD 制約と FROM:TO の Slow 例外を併用した例です。ザイリンクスでは、Slow 例外に対してこの方法はお勧めしていません。

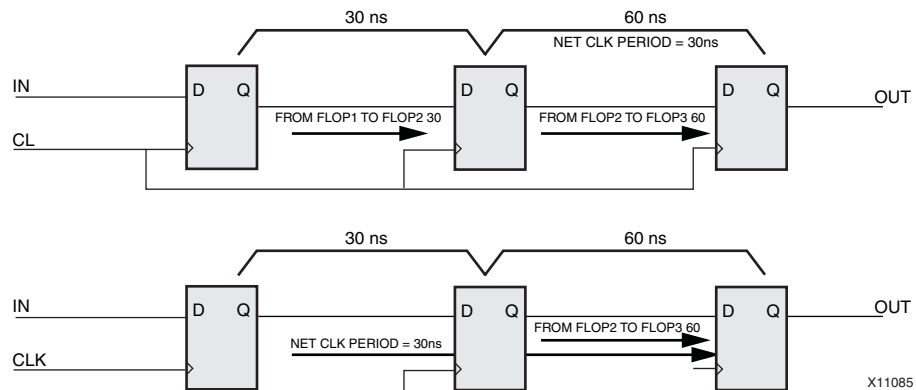


図 3-26 : PERIOD 制約と重複する Slow 例外のマルチサイクル制約

クロック イネーブル ネットを使用した Slow 例外の定義

クロック イネーブル ネットを使用すると、次の図に示すように Slow 例外を定義できます。タイミング グループに基づいているので、Slow 例外にはこの方法をお勧めします。

```
NET clk_en TNM = slow_exception;

TIMESPEC TS01 = PERIOD normal 8 ns;
TIMESPEC TS02 = FROM slow_exception TO slow_exception TS01*2;
```

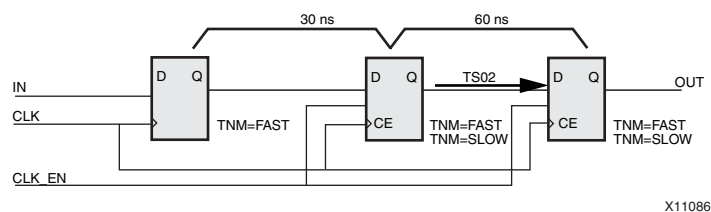


図 3-27 : FROM:TO 例外の Slow タイムグループと Fast タイムグループの重複

パスの無視

netand ネットを通る flopa と flopb 間のパスを無視するには、TIG 制約を使用します。詳細は、[図 3-28、「レジスタ間のパスの無視」](#)を参照してください。

FROM:TO:TIG 制約を使用してこれを作成するには、次の手順に従います。

1. **FFA_grp** タイムグループに flopa という名前を付けます。
2. **FFB_grp** タイムグループに flopb という名前を付けます。
3. 次の制約を作成します。

```
TIMESPEC TS_FFA_to_FFB = FROM FFA_grp TO FFB_grp TIG;
```

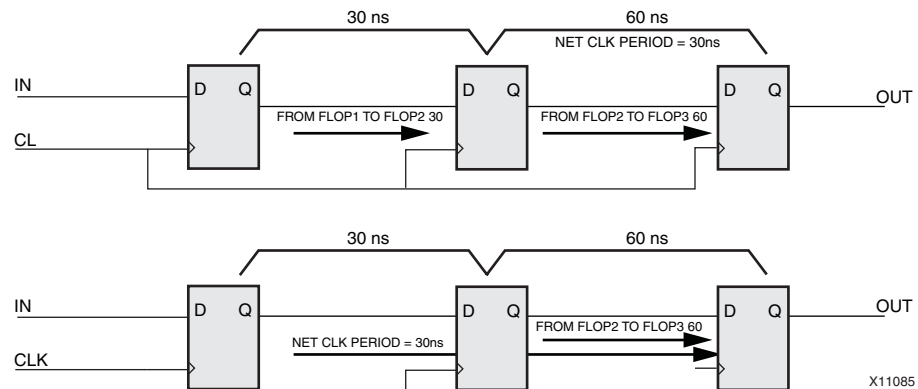


図 3-28 : レジスタ間のパスの無視

特定パスを **PERIOD** 制約よりも高速または低速にする必要がある場合は、そのパスに対して **FROM:TO** を作成します。ソースおよびデスティネーション同期エレメント間に複数のパスがある場合は、**FROM:THRU:TO** を作成して特定のパスを指定します。

この制約は、次のパスに適用されます。

1. ソース タイムグループで開始
2. 中間点を通過
3. デスティネーション グループで終了点

ソース タイムグループおよびデスティネーション タイムグループは、次のいずれかになります。

- ユーザー定義タイムグループ
- 定義済みタイムグループ

パスの中間点は、**TPTHRU** 制約で定義します。**FROM:TO** 制約の中間点の数に制限はありません。

FROM:THRU:TO 制約の例

次は、**FROM:THRU:TO** 制約の例です。

```
NET $3M17/On_the_Way TPTHU = abc;
TIMESPEC TS_mypath = FROM my_src_grp THRU abc TO my_dest_grp 9 ns;
```

タイムグループ **my_src_grp** から **abc** グループを通るタイムグループ **my_dest_grp** までに 9 ns の制約を付けています。

- **my_src_grp** は、次の図に示すように、**FIFO** に制約を付けています。
- **my_dest_grp** は次の図に示すように、レジスタに制約を付けています。

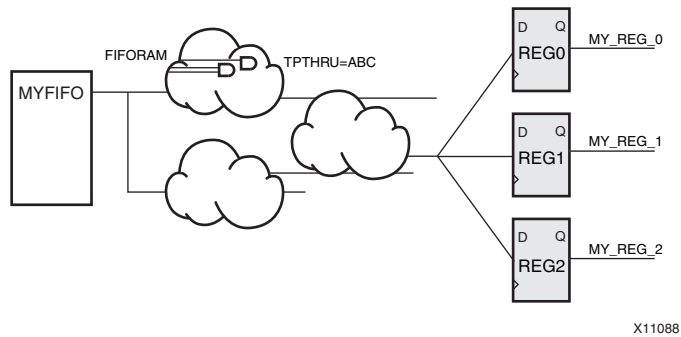


図 3-29：前の FROM:THRU:TO 制約例を使用した NET TPTHU の例

False パスまたはタイミング無視 (TIG) 制約

NET TIG には、次のような特徴があります。

- ネットに適用する制約です。
- タイミング解析で無視するネットを指定します。

FROM:TO TIG には、次のような特徴があります。

- 2 つの同期グループまたはパッド グループ間の複数のパスに指定する制約です。
- タイミング解析で無視する同期グループ間のネットすべてを指定します。

詳細は、次の図を参照してください。

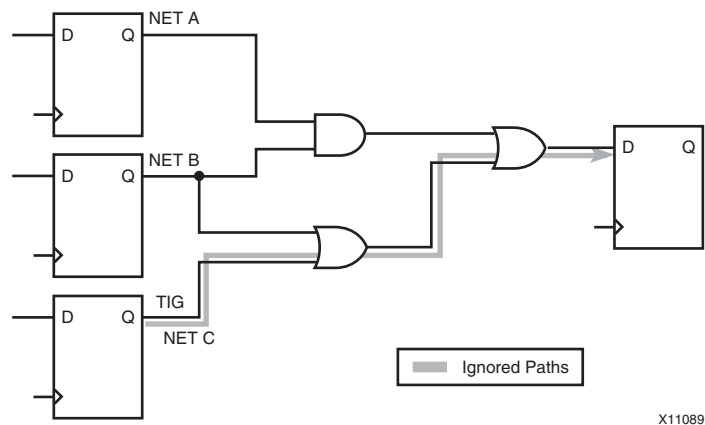


図 3-30：2 つのフリップフロップ間のパスのタイミング無視

同期パス以外の定義

FROM:THRU:TO 制約は、複数モジュールに共通のパスを使用するような同期パス以外のパスを定義するために使用できます。

モジュール同士が相互に関連していなくても、モジュール間に制約を付けることができます。

これらのモジュールは相互に関連していないので、TIG (タイミング無視) 制約を使用できます。要件が多い場合は FROM:TO 制約を設定できます。次の図に例を示します。

共通バスを通過する例

```
NET DATA_BUS* TPTHRU = DataBus;
TIMESPEC TS_TIG = FROM FFS THRU DataBus TO FFS TIG;
または
TIMESPEC TS_data_bus = FROM FFS THRU DataBus TO FFS 123ns;
```

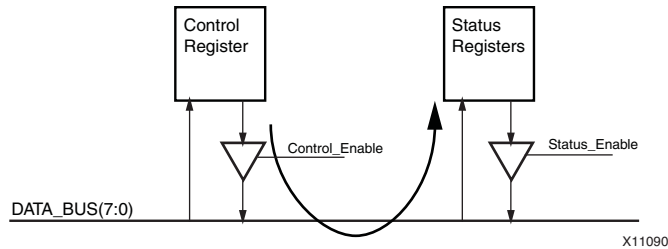


図 3-31 : 共通バスを通過する例

TPSYNC (タイミング ポイント同期) 制約の適用

TPTHRU (タイミング スルー ポイント) 制約を使用する以外にも、TPSYNC を特定ピンや組み合わせロジックに適用し、タイミング解析を非同期ポイントで停止または開始することができます。

FROM:TO を使用したトライステート バッファへの制約の適用

TPSYNC 制約では、マルチサイクル制約および解析で非同期ポイントを同期ポイントとして定義できます。たとえば、トライステート バッファへのパスには TPTSYNC 制約を使用できます。

次の図は、トライステート バッファへのパスに制約を適用する例を示しています。

```
NET $3M17/Blue TPTSYNC = Blue_S;
TIMESPEC TS_1A = FROM FFS TO Blue_S 15;
```

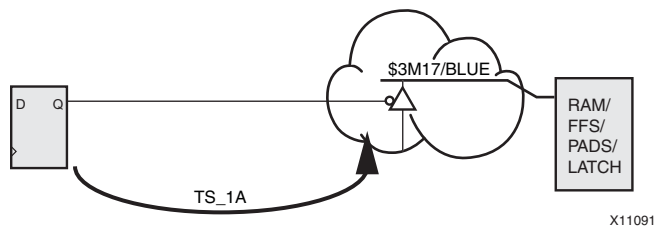


図 3-32 : FROM:TO を使用したトライステート バッファへの制約の適用

FROM:TO 制約の適用されるパス

FROM:TO 制約には、次のような特徴があります。

- 2つのグループ間のタイミング制約を定義します。
- PERIOD および OFFSET IN/OUT 制約と併用されます。
- Fast および Slow 例外を定義できます。

多用途に使用できます。次の図は、次のような単純なデザインの例です。

```
TIMESPEC TS_C2S = FROM FFS TO FFS 12 ns;
TIMESPEC TS_P2S = FROM PADS TO FFS 10 ns;
TIMESPEC TS_P2P = FROM PADS TO PADS 13 ns;
TIMESPEC TS_C2P = FROM FFS TO PADS 8 ns;
```

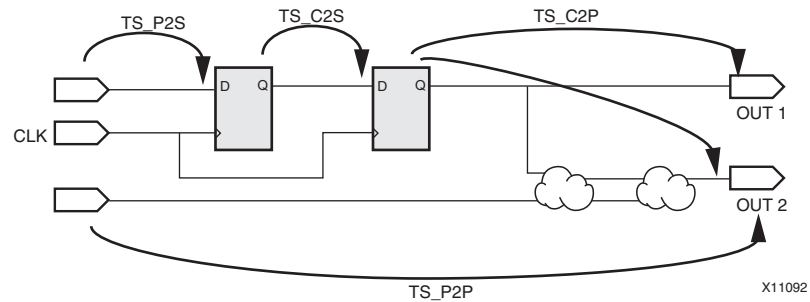


図 3-33：サンプル デザインの制約付きパスすべて

解析の PERIOD から FROM:TO への変更

解析を PERIOD から FROM:TO に変更すると、次のようになります。

- 解析されるパス数はパスに PERIOD 制約が適用された場合よりも多くなりますが
- 制約の付いていないパスの数は増加しません。

FROM:TO 制約のデスティネーションの TIMEGRP には、おそらく分散型デュアル ポート同期 RAM が含まれます。この RAM へのパスは、同期と非同期の両方になります。

- データ入力 (D) へのパスは同期です。
- 読み出しアドレス入力 (DPRA) へのパスは非同期です。

PERIOD 制約は同期パスにしか適用できません。FROM:TO 制約はこの RAM への同期と非同期の両方のパスに適用できます。

- フリップフロップからこの RAM の D 入力へのパスは同期パスです。このデータ パスには、PERIOD または FROM:TO 制約を付けることができます。
- フリップフロップからこの RAM の DPRA 入力へのパスは、読み出しアドレス入力 (DPRA) に対して非同期パスなので、FROM:TO 制約しか適用できません。

グループ制約の構文

グループ制約の構文の詳細は、付録 A「その他のリソース」に示す『制約ガイド』(UG625) の次の制約の項目を参照してください。

- TNM (タイミング名)
- TNM_NET (タイミング名ネット)
- TIMEGRP (タイミング グループ)
- PERIOD (周期)
- OFFSET IN
- OFFSET OUT
- FROM:TO (マルチサイクル)

タイミング制約の作成

タイミング制約は、次の 2 つの方法でデザインに追加できます。

- HDL デザイン詳細は、次を参照してください。
 - [「XST でのタイミング制約の指定」](#)
 - [「Synplifyでのタイミング制約の指定」](#)
- Constraints Editor (UCF) の使用

Constraints Editor (UCF) でのタイミング制約の作成

Constraints Editor では、NGD ファイルから情報を読み込んで、UCF ファイルの制約が作成されます。Constraints Editor で NGD ファイルが解析されると、各デザイン エLEMENT と制約が UCF 構文どおりにインプリメンテーション ツールで使用されるようになります。

Constraints Editor を使用すると、タイム グループとタイミング制約を作成できます。クロックおよび I/O があらかじめ供給されるため、名前のスペルがまったく同一である必要はありません。ユーザーが定義する必要があるのは、制約の構文ではなく、タイミング条件のみです。

特定のタイムグループを作成する場合、ELEMENT 名があらかじめ付けられており、グローバル制約に対する例外はこれらのグループを使用して設定できます。

タイム グループのサイズの削減

Constraints Editor ではワイルドカード付きのタイムグループや制約は作成されないため、手動で UCF ファイルを修正してタイムグループのサイズを小さくする必要があります。UCF でタイムグループのサイズを小さくするには、デザイン ELEMENT の特定部分と残りの共通の部分でワイルドカードを使用します。

タイム グループの削減例

```
INST my_bus* TNM = my_output_bus_grp;
```

アスタリスク (*) ワイルドカードにより、my_bus という名前で始まるインスタンスすべてに TNM 属性が適用されます。

XST でのタイミング制約の指定

この章では、ザイリンクスクロージャ合成ツール (XST) を使用してタイミング制約を指定する方法について説明します。

Synplify でのタイミング制約の指定方法については、[第 5 章「Synplifyでのタイミング制約の指定」](#)を参照してください。

「[XST タイミング制約の構文例](#)」では、ザイリンクス タイミング制約の VHDL、Verilog および XCF ファイルの構文例を示します。

詳細は、次を参照してください。

- 『合成/シミュレーション デザイン ガイド』(UG626) ([付録 A「その他のリソース」](#)にリンクをリスト)
- 『XST ユーザー ガイド (Virtex-6、Spartan-6、および 7 シリーズ デバイス用)』(UG687) ([付録 A「その他のリソース」](#)にリンクをリスト)

XST タイミング制約の指定

XST タイミング制約は、次のいずれかを使用して適用します。

- -glob_opt コマンド ライン オプション
- XST 制約ファイル (XCF)

ソース コードで指定したタイミング制約はネットリストに伝搬されないので、タイミング制約はすべて UCF で指定する必要があります。

タイミング モデル

XST でタイミング解析用に使用されるタイミング モデルでは、次が考慮されます。

- 論理遅延
論理遅延データは、配置配線後に TRACE (Timing Analyzer) でレポートされる遅延と同じです。
- ネット遅延
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。

これらの遅延には、次によって異なります。

- XST に指定可能なスピード グレード
- 選択したデバイス

XCF 制約の優先順

制約は次の順序で処理されます。

1. 信号の特定制約
2. 最上位モジュールの特定制約
3. 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。

XST でのタイミング制約の指定方法

XST でタイミング制約を指定する場合は、次のいずれかを使用します。

- ハードウェア記述言語 (HDL) コード
- XST 制約ファイル (XCF)
- -glob_opt コマンド ライン オプション

合成前にタイミング制約を指定するには、次の方法に従います。

- 次のいずれかでタイミング制約を指定します。
 - HDL
 - VHDL
 - Verilog
 - 回路図
- または
- XCF でタイミング制約を指定します。

HDL でのタイミング制約の指定

HDL コードでタイミング制約を指定する場合は、その制約の形式で記述します。

XCF でのタイミング制約の指定

XST では、合成およびタイミングの制約を指定するのにザイリンクス制約ファイル (XCF) の構文形式がサポートされます。制約ファイルを使用する場合は、ネイティブの XCF タイミング制約構文を使用できます。

XST でサポートされる XCF 制約

XCF 構文を使用する場合、XST では、次の制約がサポートされます。

- TNM_NET (タイミング名ネット)
- TIMEGRP (タイミング グループ)
- PERIOD (周期)
- TIG (タイミング無視)
- FROM-TO

これには、ワイルドカードおよび階層名が含まれます。

タイミング制約は、デフォルトのままでは NGC ファイルに書き込まれません。タイミング制約は、次のどちらかの設定をしている場合にのみ NGC ファイルに書き込まれます。

- ISE の [Process Properties] ダイアログ ボックスの [Synthesis Options] ページにある [Write Timing Constraints] をオン
- **-write_timing_constraints** コマンド ライン オプションを使用

XCF 構文の制限

XCF 構文には、次の制限があります。

- MODEL 文のネストはサポートされません。
- BEGIN MODEL と END 間にリストされたインスタンス名や信号名のみが、エンティティ内で有効です。
- 階層インスタンス名または信号名はサポートされません。

-glob_opt コマンド ライン オプションを使用したタイミング制約の指定

XST でサポートされるタイミング制約は、-glob_opt コマンド ライン オプションを使用しても適用できます。-glob_opt コマンド ライン オプションは、Project Navigator の次のオプションと同じです。

[Process] → [Process Properties] → [Synthesis Options] → [Global Optimization Goal]

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。この制約には、値を指定できません。最適なパフォーマンスを得るため、デザイン全体が最適化されます。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。

XST タイミング制約の構文例

このセクションでは、ザイリンクス タイミング制約の VHDL、Verilog および XCF ファイルの構文例を示します。制約によっては、これら 3 つすべての例が表示されないこともあります。

- 「ASYNC_REG (非同期レジスタ)」
- 「CLOCK_SIGNAL (クロック信号)」
- 「MAXDELAY (最大遅延)」
- 「MAXDELAY (最大遅延)」
- 「MAXSKEW (最大スキュー)」
- 「OFFSET (オフセット)」
- 「PERIOD (周期)」
- 「SYSTEM_JITTER (システム ジッター)」
- 「NET/PIN/INST TIG (タイミング無視)」
- 「TIMEGRP (タイミング グループ)」
- 「TIMESPEC (タイミング仕様)」
- 「TNM (タイミング名)」
- 「TNM_NET (タイミング名ネット)」

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。

ASYNC_REG (非同期レジスタ)

ASYNC_REG (非同期レジスタ) 制約は、非同期入力 (D 入力または CE 入力) 付きのレジスタまたはラッチにのみ設定できます。

詳細は、[付録 A「その他のリソース」](#)に記述されている『制約ガイド』(UG625) を参照してください。

VHDL 構文

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

VHDL 構文例

```
architecture behavioral of top_yann_mem_infrastructure is  
begin  
  signal sys_rst      : std_logic;  
  attribute ASYNC_REG : string;  
  attribute ASYNC_REG of sys_rst: signal is "TRUE";  
  --source code  
End behavioral;
```

Verilog 構文

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

Verilog 構文例

```
module mig_22  
( inout [7:0]  cntrl0_ddr2_dq,  
  output [14:0] cntrl0_ddr2_a,  
  input      sys_clk_p,  
  input      sys_clk_n,  
  input      clk200_p,  
  input      clk200_n,  
  input      sys_reset_in_n,  
  inout [0:0]  cntrl0_ddr2_dqs  
);  
  wire clk_0;  
  wire clk_90;  
  wire clk_200;  
  (* ASYNC_REG = "TRUE" *)  
  reg sys_rst;  
  // source code  
End module;
```

CLOCK_SIGNAL (クロック信号)

すべての FPGA に適用できます。CPLD には適用できません。

クロック信号がフリップフロップのクロック入力に接続される前に組み合わせロジックを通過する場合、クロック信号となる入力ピンまたは内部信号が認識されません。この制約を設定すると、クロック信号を定義できます。

VHDL 構文

```
attribute clock_signal : string;  
attribute clock_signal of signal_name : signal is "{yes|no}";
```

VHDL 構文例

```
entity top_yann_mem is  
port ( cntrl0_DDR2_DQ : inout std_logic_vector(71 downto 0);  
      SYS_CLK_P : in std_logic;  
      SYS_CLK_N : in std_logic;  
      CLK200_P : in std_logic;  
      CLK200_N : in std_logic  
      );  
attribute clock_signal : string;  
attribute clock_signal of clk200_p : signal is "yes";  
end entity;
```

Verilog 構文

```
(* clock_signal = "{yes|no}" *)
```

Verilog 構文例

```
module mig_22  
( inout [7:0] cntrl0_ddr2_dq,  
  output [14:0] cntrl0_ddr2_a,  
  input sys_clk_p,  
  input sys_clk_n,  
  input clk200_p,  
  input clk200_n,  
  input sys_reset_in_n,  
  inout [0:0] cntrl0_ddr2_dqs  
  );  
(* clock_signal = "yes" *)  
wire clk_0;  
wire clk_90;  
wire clk_200;  
reg sys_rst;  
// source code  
End module;
```

XCF の構文

```
BEGIN MODEL "entity_name"  
NET "primary_clock_signal" clock_signal={yes|no|true|false};  
END;
```

XCF の構文例

```
BEGIN MODEL "top_yann_mem"  
NET "CLK200_P" clock_signal = yes;  
END;
```

MAXDELAY (最大遅延)

MAXDELAY (最大遅延) には、次のような特徴があります。

- ネットの最大許容遅延を定義します。
- FPGA デバイスのネットにのみ使用できます。

詳細は、[付録 A「その他のリソース」](#)に記述されている『制約ガイド』(UG625) を参照してください。

VHDL 構文

```
attribute maxdelay of signal_name: signal is "value [units]";
```

説明

- value は、正の整数です。
- units は、ps、ns、us、ms、Hz、kHz、MHz などの単位です。デフォルトは ns です。

VHDL 構文例

```
entity top_yann_mem_data_path_iobs_0 is
port (
    CLK      : in std_logic;
    dqs_delayed : out std_logic_vector(31 downto 0);
    READ_EN_DELAYED_RISE : out std_logic_vector(31 downto 0);
    READ_EN_DELAYED_FALL : out std_logic_vector(31 downto 0);
);
attribute maxdelay: string;
attribute maxdelay of READ_EN_DELAYED_RISE: signal is "800 ps";
attribute maxdelay of READ_EN_DELAYED_FALL: signal is "800 ps";
end entity;
```

Verilog 構文

```
(*MAXDELAY = "value [units]" *)
```

Verilog 構文例

```
module mig_22
( inout [7:0]  cntrl0_ddr2_dq,
  output [14:0] cntrl0_ddr2_a,
  input        sys_clk_p,
  input        sys_clk_n,
  input        clk200_p,
  input        clk200_n,
  input        sys_reset_in_n,
  inout [0:0]  cntrl0_ddr2_dqs
);
wire clk_0;
wire clk_90;
wire clk_200;
(*MAXDELAY= " 800 ps" *)
wire read_en;
reg sys_rst;
// source code
End module;
```

MAXSKEW (最大スキュー)

ネット上のスキュー量を制御するために使用します。「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。

詳細は、[付録 A「その他のリソース」](#)に記述されている『制約ガイド』(UG625) を参照してください。

VHDL 構文

```
attribute maxskew: string;  
attribute maxskew of signal_name : signal is "allowable_skew [units]";
```

説明

- 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 構文例

```
entity top_yann_mem_infrastructure is  
port (  
    SYS_CLK_P: in std_logic;  
    SYS_CLK_N: in std_logic;  
    CLK200_P: in std_logic;  
    CLK200_N: in std_logic;  
    CLK      : out std_logic;  
    REFRESH_CLK : out std_logic;  
    sys_rst   : out std_logic;  
);  
attribute maxskew: string;  
attribute maxskew of sys_rst : signal is "3 ns";  
end entity;
```

Verilog 構文

```
(* MAXSKEW = "allowable_skew [units]" *)
```

説明

- 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 構文例

```
module mig_22
( inout [7:0]  cntrl0_ddr2_dq,
  output [14:0] cntrl0_ddr2_a,
  input        sys_clk_p,
  input        sys_clk_n,
  input        clk200_p,
  input        clk200_n,
  input        sys_reset_in_n,
  inout [0:0]  cntrl0_ddr2_dqs
);
wire clk_0;
wire clk_90;
wire clk_200;
(*MAXSKEW= " 3 ns" *)
wire read_en;
reg sys_rst;
// source code
End module;
```

OFFSET (オフセット)

OFFSET (オフセット) 制約には、次の特徴があります。

- 次の 2 つのタイミング関係を定義するのに使用します。
 - 外部クロック パッド
 - 関連するデータ入力/データ出力パッド
- パッド関連の信号にのみ使用されます。
- 内部信号への信号到着時間は指定できません。

詳細は、第 3 章「[タイミング制約の基礎](#)」を参照してください。

XCF 構文

```
OFFSET = {IN|OUT} offset_time [units] {BEFORE|AFTER}  
clk_name [TIMEGRP group_name];
```

説明

- **offset_time [units]**

クロック エッジのキャプチャ時とキャプチャされるデータの開始時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。units は ps、ns、micro、ms などの単位です。

- **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。

クロックとデータの間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、データはクロック エッジのキャプチャよりも前に開始されます。
- **BEFORE** に負の値を指定すると、データはクロック エッジのキャプチャよりも後に開始されます。

- **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。

- **VALID** キーワード

OFFSET 制約には指定できません。

XCF 構文例

```
OFFSET = IN 2 ns BEFORE "CLK200_N" ;  
OFFSET = IN 3.85 ns BEFORE "SYS_CLK_P" ;  
OFFSET = OUT 4 ns AFTER "CLK200_N" ;  
OFFSET = OUT 7 ns AFTER "SYS_CLK_P" ;  
NET "main_00/top_00/iobs_00/data_path_iobs_00/v4_dq_iob_0/DDR_DQ" TNM=  
DDR2_DQ_Grp ;  
OFFSET = OUT 6.7 ns AFTER "SYS_CLK_P" TIMEGRP DDR2_DQ_Grp ;  
OFFSET = IN 3.2 ns BEFORE "SYS_CLK_P" TIMEGRP DDR2_DQ_Grp ;
```

PERIOD (周期)

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ELEMENT間のタイミングが確認されます。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では、次が実行されます。

- レジスタのクロック ピンにおけるクロック ネットの反転やクロック位相が自動的に考慮されます。
- 同期ELEMENTのすべてのタイプが解析に含まれます。
- ホールド違反があるかどうかチェックされます。

詳細は、第 3 章「タイミング制約の基礎」を参照してください。

VHDL 構文

PERIOD 制約は、特定のクロック信号のみに適用されます。

```
attribute period: string;  
attribute period of signal_name : signal is "period [units]";
```

VHDL 構文例

```
entity top_yann_mem is  
port ( cntrl0_DDR2_DQ : inout std_logic_vector(71 downto 0);  
      SYS_CLK_P : in std_logic;  
      SYS_CLK_N : in std_logic;  
      CLK200_P : in std_logic;  
      CLK200_N : in std_logic  
      );  
attribute period: string;  
attribute period of SYS_CLK_P : signal is "5 ns";  
end entity;
```

Verilog 構文

PERIOD 制約は、特定のクロック信号のみに適用されます。

```
(* PERIOD = "period [units]" *)
```

説明

- 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 なども指定できます。

Verilog 構文例

```
module mig_22
( inout [7:0]  cntrl0_ddr2_dq,
  output [14:0] cntrl0_ddr2_a,
  input      sys_clk_p,
  input      sys_clk_n,
  input      clk200_p,
  input      clk200_n,
  input      sys_reset_in_n,
  inout [0:0]  cntrl0_ddr2_dqs
);
(*PERIOD = "5 ns"*)
wire clk_0; // The clk_0 is assigned with the period of 5 ns
wire clk_90;
wire clk_200;
wire read_en;
reg sys_rst;
// source code
End module;
```

XCF 構文

これは、PERIOD 制約を XCF 構文で指定する第一の方法です。ザイリンクスでは、この方法をお勧めしています。

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference period" [units]
[{HIGH |LOW} [high_or_low_time [hi_lo_units]]] INPUT_JITTER value [units];
```

NET PERIOD の XCF 構文

これは、PERIOD 制約を XCF 構文で指定する 2 つ目の方法です。この方法は、お勧めしません。

```
NET "net_name" PERIOD=period [units]
[{HIGH|LOW}[high_or_low_time[hi_lo_units]]];
```

説明

- **identifier** は、一意な名前を持つ参照識別名です。
- **TNM_reference** 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、または PLL 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 出力に指定できます。
- **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 および LOW は、タイミング概算および最適化中には考慮されません。これらは、**WRITE_TIMING_CONSTRAINTS = yes** の場合にのみ最終のネットリストに伝搬されます。
- **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。

次の文は、クロック周期 40ns を **CLOCK** という名前のネットに設定します。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;
```

次の文は、5 ナノ秒のクロック周期を **TIMESPEC** の形式で指定しています。

```
NET "infrastructure0/SYS_CLK_IN" TNM_NET = "SYS_CLK";  
TIMESPEC "TS_SYS_CLK" = PERIOD "SYS_CLK" 5 ns HIGH 50 %;
```

SYSTEM_JITTER (システム ジッター)

SYSTEM_JITTER 制約では、デザインのシステム ジッターを指定します。システム ジッターの値は、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適用されます。SYSTEM_JITTER を PERIOD 制約の INPUT_JITTER キーワードと組み合わせると、タイミング レポートのクロックのばらつき (Clock Uncertainty) の値を生成できます。

詳細は、第 3 章「タイミング制約の基礎」を参照してください。

SYSTEM_JITTER を使用すると、システムのジッターを特性化する方法がない場合に、追加のタイミング差を指定することもできます。この制約は、タイミング差が厳しい場合にデザインの制限をテストするのに便利です。SYSTEM_JITTER は、クロックを解析する制約のクロックのばらつき計算で使用されます。

デバイスによっては、スピード ファイルにデフォルトのシステム ジッターが含まれます。これは、SpeedPrint を使用すると確認できます。

特定の入力クロックの入力ジッターを修正しても、同じテストが実行できます。これは、システム全体ではなく、特定のクロック ドメインでのみ実行できます。

VHDL 構文

```
attribute SYSTEM_JITTER: string;
attribute SYSTEM_JITTER of
{component_name|signal_name|entity_name|label_name}:
{component|signal|entity|label} is "value ps";
```

説明

- value には、数値を指定します。デフォルトは ps です。

VHDL 構文例

```
entity top_yann_mem is
port ( cntrl0_DDR2_DQ : inout std_logic_vector(71 downto 0);
      SYS_CLK_P : in std_logic;
      SYS_CLK_N : in std_logic;
      CLK200_P : in std_logic;
      CLK200_N : in std_logic
      );
attribute SYSTEM_JITTER : string;
attribute SYSTEM_JITTER of top_yann_mem: entity is "10 ps";
end entity;
```

Verilog 構文

```
(* SYSTEM_JITTER = "value ps" *)
```

説明

- value には、数値を指定します。デフォルトは ps です。

Verilog 構文例

```
module mig_22
( inout [7:0]  cntrl0_ddr2_dq,
  output [14:0] cntrl0_ddr2_a,
  input        sys_clk_p,
  input        sys_clk_n,
  input        clk200_p,
  input        clk200_n,
  input        sys_reset_in_n,
  inout [0:0]  cntrl0_ddr2_dqs
);
(*SYSTEM_JITTER = "10 ps"*)
wire clk_0; // The clk_0 is assigned with system_jitter of 10 ps
wire clk_90;
wire clk_200;
wire read_en;
reg sys_rst;
// source code
End module;
```

XCF 構文

```
MODEL "entity_name" SYSTEM_JITTER = value ps;
```

XCF 構文例

```
MODEL "top_yann_mem" SYSTEM_JITTER = 10;
```

NET/PIN/INST TIG (タイミング無視)

TIG (タイミング無視) 制約には、次のような特徴があります。

- タイミング制約です。
- 合成制約です。
- FPGA デバイスにのみ適用できます。
- CPLD には適用できません。

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。

詳細は、第 3 章「[タイミング制約の基礎](#)」を参照してください。

XCF 構文

```
NET "net_name" TIG;  
PIN "ff_inst.RST" TIG=TS_1;  
INST "instance_name" TIG=TS_2;  
TIG=TSidentifier1,..., TSidentifiern
```

説明

- 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。ネットに設定されている場合、ネットの駆動ピンに適用されます。ピンに設定されている場合、ピンに適用されます。

XCF 構文例

```
NET "main_?0/top_?0/ddr2_controller_?0/load_mode_reg*" TIG;
```

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;
```

TIMEGRP (タイミング グループ)

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。この制約は、ザイリンクス制約ファイル (XCF) またはネットリスト制約ファイル (NCF) のような制約ファイルで設定できます。

詳細は、第 3 章「[タイミング制約の基礎](#)」を参照してください。

XCF 構文

```
TIMEGRP newgroup = existing_grp1 existing_grp2 [existing_grp3 ...];
```

説明

- **newgroup** は新規作成されるグループで、TNM 制約、定義済みグループ、ほかの TIMEGRP 制約で作成された既存のグループが含まれます。

XCF 構文例

```
TIMEGRP Top_Group = GroupA GroupB GroupC;
```

マルチサイクル パス

マルチサイクル パス制約は、2 つのグループ間のタイミング制約を指定する制約です。

詳細は、第 3 章「[タイミング制約の基礎](#)」を参照してください。

XCF 構文

```
TIMESPEC TSname =FROM "group1" TO "group2" value;
```

説明

- 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、ソースのタイミング グループです。
- group2 は、デスティネーションのタイミング グループです。
- 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。

FROM-TO 制約のサポートには、次の制限事項があります。

- FROM-THRU-TO はサポートされません。
- タイミング仕様の結合はサポートされません。
- 次のような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;
```

XCF 構文例

```
TIMESPEC TS_MY_PathA = FROM "my_src_grp" TO "my_dst_grp" 23.5 ns;  
TIMESPEC TS_ DQS_UNUSED = FROM FFS TO "control_unused_dqs" TIG;
```

TIMESPEC (タイミング仕様)

TIMESPEC (タイミング仕様) 制約には、次のような特徴があります。

- 基本的なタイミングに関連する制約です。
- TS 属性で定義するタイムスペックのプレースホルダの役割を果たします。

TS 属性には、次の特徴があります。

- パスの許容可能な遅延を定義します。
- TS という文字で始まります。
- 次を含むことのできる独自の識別子で終了します。
 - 文字
 - 数値
 - アンダースコア (_)

インプリメンテーションおよび解析ツールのランタイムおよびメモリ使用率は、デザインで使用する TIMESPEC 制約の数によって変わります。

XCF 構文

```
TIMESPEC "TSidentifier"=PERIOD "timegroup_name" value [units];  
TIMESPEC "TSidentifier"=FROM "source_group" TO "dest_group" value units;
```

説明

- **TSidentifier** は、TS 属性の名前です。
- **value** には、数値を指定します。**value** は、属性の最大遅延を定義します。TS 属性の遅延時間を指定するデフォルトの単位は、ナノ秒 (ns) です。遅延は、ピコ秒 (ps) やメガヘルツ (MHz) のようなほかの単位でも指定できます。
- **units** は、ms、micro、ps、ns などの時間の単位です。

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。大文字と小文字を組み合わせることはできません。

XCF 構文例

- 「最大許容遅延を定義する場合の XCF 構文例」
- 「クロック周期を定義する場合の XCF 構文例」
- 「派生クロックを定義する場合の XCF 構文例」
- 「XCF 構文例」

最大許容遅延を定義する場合の XCF 構文例

```
TIMESPEC "TSidentifier"=FROM "source_group" TO "dest_group" allowable_delay [units];
```

クロック周期を定義する場合の XCF 構文例

クロック周期を定義すると、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;
```

説明

- **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。**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。

派生クロックを定義する場合の XCF 構文例

```
TIMESPEC "TSidentifier"=PERIOD "TNM_reference" "another_PERIOD_identifier" [/ | *] number  
[{HIGH | LOW} [high_or_low_time [hi_lo_units]]] INPUT_JITTER value;
```

説明

- **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。**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。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。

XCF 構文例

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

説明

- **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。

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;
```

TNM (タイミング名)

タイミング名 (TNM) 制約には、次の特徴があります。

- 基本的なグループ制約です。
- タイミング仕様で使用するグループを構成するエレメントを指定します。
- 特定の定義済みグループにグループを構成するエレメントとして名前を付けて、グループへのタイミング仕様の適用を簡略化できます。
- RISING および FALLING キーワードのサポート

詳細は、第3章「タイミング制約の基礎」を参照してください。

XCF 構文

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

説明

- predefined_group は、グループを構成するすべてのエレメントまたはキーワード (FFS、RAMS、LATCHES、PADS、CPUS、HSIOS、BRAMS_PORTA、BRAMS_PORTB、DSPS および MULTS) を使用した定義済みグループのサブセットになります。
- 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。

XCF 構文例

```
NET clk TNM = FFS (my_flop) Grp1;  
INST clk TNM = FFS (my_macro) Grp2;
```

TNM_NET (タイミング名ネット)

タイミング仕様で使用されるグループを構成するエレメントを指定します。

タイミング名ネット (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。

詳細は、第 3 章「タイミング制約の基礎」を参照してください。

XCF 構文

```
{NET|INST} "net_name" TNM_NET= [predefined_group] identifier;
```

説明

- predefined_group は、キーワード (FFS、RAMS、PADS、MULTS、HSIOS、CPUS、DSPS、BRAMS_PORTA、BRAMS_PORTB または LATCHES) を使用した、定義済みグループのすべてのエレメントです。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)
- 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_NET が NET に設定される場合、name_qualifier はネット名になり、インスタンス (INST) に設定される場合、インスタンス名になります。

- 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、次の予約語は使用できません。FFS、RAMS、LATCHES、PADS、CPUS、HSIOS、MULTS、RISING、FALLING、TRANSHI、TRANSLO、または EXCEPT

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して 1 つのパターンしかサポートされません。

表 4-1 : TNM_NET のサポート制限

サポートあり	NET "PADCLK" TNM_NET=FFS "GRP1"; #
サポートなし	NET "PADCLK" TNM_NET = FFS(machine/*:xcounter/*) TG1; #

XCF 構文例

```
NET clk TNM_NET = FFS (my_flop) Grp1;
INST clk TNM_NET = FFS (my_macro) Grp2;
```

Synplifyでのタイミング制約の指定

この章では、Synplify でタイミング制約を指定する方法について説明します。ザイリンクスクロージャ合成ツール (XST) を使用してタイミング制約を指定する方法については、[第 4 章「XST でのタイミング制約の指定」](#)を参照してください。

このセクションでは、ザイリンクス タイミング制約の VHDL および Verilog の構文例を示します。詳細は、次を参照してください。

- 『合成/シミュレーション デザイン ガイド』(UG626) ([付録 A「その他のリソース」](#)にリンクをリスト)
- 『Synopsys FPGA Synthesis Reference Manual』

Synplify でのタイミング制約の指定方法は、次のとおりです。

- [「HDL でのタイミング制約の指定」](#)
- [「SDC ファイル \(Tcl\) でのタイミング制約の指定」](#)
- [「SCOPE スプレッドシートでのタイミング制約の指定」](#)

同じオブジェクトに複数のタイミング例外制約が設定されている場合、合成ツールでは『Synopsys FPGA Synthesis Reference Manual』の「Conflict Resolution for Timing Exceptions」のガイドラインにしたがって、制約の優先順位が決定されます。

制約のタイプ

表 5-1、「Synplify のタイミング制約の入力方法クロージャ」には、タイミング制約と関連コマンドが入力に使用される方法に従って、アルファベット順にリストされています。HDL のタイミング制約は、すべて指示子です。

表 5-1 : Synplify のタイミング制約の入力方法クロージャ

HDL	Tcl (.sdc ファイル)	SCOPE
「black_box_tri_pins」		
	「define_clock」	[Clocks] パネル
	「define_clock_delay」	[Clock to Clock] パネル
	「define_compile_point」	[Compile Points] パネル
	「define_current_design」	
	「define_false_path」	[False Paths] パネル
	「define_input_delay」	[Inputs/Outputs] パネル
	「define_io_standard」	[I/O Standard] パネル
	「define_multicycle_path」	[Multi-Cycle Paths] パネル
	「define_output_delay」	[Inputs/Outputs] パネル
	「define_path_delay」	[Max Delay Paths] パネル
	「define_reg_input_delay」	[Registers] パネル
	「define_reg_output_delay」	[Registers] パネル
「syn_force_seq_prim」 *		
「syn_gatedclk_clock_en」 *		
「syn_gatedclk_clock_en_polarity」 *		
「syn_isclock」		
「syn_tpdn」		
「syn_tcon」		
「syn_tsun」		

* この制約は、Synplify Pro および Synplify Premier でのみ使用できます。

HDL でのタイミング制約の指定

ソース コードの属性および指示子をハードウェア記述言語 (HDL) コードに書き込みます。

ソース コードには、ブラック ボックス タイミング指示子を入力する必要があります。ほかのタイミング制約は含めないでください。ソース コードのポータビリティは下がっているので、制約が適用されるにはデザインをコンパイルし直す必要があります。

属性は **SCOPE** スプレッドシートを使用しても入力できます。指示子にはソース コードを使用する必要があります。

HDL タイミング制約の構文例

次のセクションでは、HDL タイミング制約の構文例を示します。

- 「black_box_pad_pin」
- 「black_box_tri_pins」
- 「syn_force_seq_prim」
- 「syn_gatedclk_clock_en」
- 「syn_gatedclk_clock_en_polarity」
- 「syn_isclock」
- 「syn_tpdn」
- 「syn_tcon」
- 「syn_tsun」

black_box_pad_pin

black_box_pad_pin では、ユーザー定義のブラック ボックス コンポーネントのピンをブラック ボックスの外の環境から見える I/O パッドとして指定します。

複数のポートが I/O パッドの場合は、ポートを次のようにリストします。

- カンマ区切りの二重引用符 (" ") で囲む
- スペースなし

Verilog 構文

```
object /* synthesis syn_black_box black_box_pad_pin = "portList" */ ;
```

説明

- portList は I/O パッドのブラック ボックス ポート名のスペースのないカンマ区切りのリストです。

Verilog 構文例

```
module BBDLHS(D,E,GIN,GOUT,PAD,Q)
/* synthesis syn_black_box black_box_pad_pin="GIN[2:0],Q" */;
```

VHDL 構文

```
attribute black_box_pad_pin of object : objectType is "portList" ;
```

説明

- object は、ブラック ボックスのアーキテクチャまたはコンポーネント宣言です。データ型は、文字列です。
- portList は I/O パッドのブラック ボックス ポート名のスペースのないカンマ区切りのリストです。

VHDL 構文例

```
library ieee;
use ieee.std_logic_1164.all;
package my_components is
component BBDLHS
port (D: in std_logic;
E: in std_logic;
GIN : in std_logic_vector(2 downto 0);
Q : out std_logic );
end component;
attribute syn_black_box : boolean;
attribute syn_black_box of BBDLHS : component is true;
attribute black_box_pad_pin : string;
attribute black_box_pad_pin of BBDLHS : component is "GIN(2:0),Q";
end package my_components;
```

black_box_tri_pins

black_box_tri_pins では、ブラック ボックスとして定義されるコンポーネントの出力ポートがトライステートであることを指定します。**black_box_tri_pins** を使用すると、ブラック ボックスの出力に複数のドライバがある場合、複数ドライバ エラーを回避できます。複数ドライバ エラーは、**black_box_tri_pins** を使用して出力ピンがトライステートであることを指定しておかないと、発生します。

トライステート ポートが複数ある場合は、ポートを次のようにリストします。

- カンマ区切りの二重引用符 (" ") で囲む
- スペースなし

Verilog 構文

```
object /* synthesis syn_black_box black_box_tri_pins = "portList" */ ;
```

説明

- **portList** は複数ピンのスペースのないカンマ区切りのリストです。

Verilog 構文例

次は、1 つのポート名しかない場合の **black_box_tri_pins** の Verilog 構文例です。

```
module BBDLHS(D,E,GIN,GOUT,PAD,Q)
/* synthesis syn_black_box black_box_tri_pins="PAD" */;
Here is an example with a list of multiple pins:
module bbl(D,E,tri1,tri2,tri3,Q)
/* synthesis syn_black_box black_box_tri_pins="tri1,tri2,tri3" */;
For a bus, specify the port name followed by all the bits on the bus:
module bbl(D,bus1,E,GIN,GOUT,Q)
/* synthesis syn_black_box black_box_tri_pins="bus1[7:0]" */;
```

VHDL 構文

```
attribute black_box_tri_pins of object : objectType is "portList" ;
```

説明

- **object** は、アーキテクチャまたはコンポーネント宣言です。データ型は、文字列です。
- **portList** はトライステート出力ポート名のスペースのないカンマ区切りのリストです。

VHDL 構文例

```
library ieee;
use ieee.std_logic_1164.all;
package my_components is
component BBDLHS
port (D: in std_logic;
E: in std_logic;
GIN : in std_logic;
GOUT : in std_logic;
PAD : inout std_logic;
Q: out std_logic );
end component;
attribute syn_black_box : boolean;
```

```
attribute syn_black_box of BBDLHS : component is true;  
attribute black_box_tri_pins : string;  
attribute black_box_tri_pins of BBDLHS : component is "PAD";  
end package my_components;
```

同じコンポーネントの複数ピンは、次のようにリストして指定できます。

```
attribute black_box_tri_pins of bbl : component is "tri,tri2,tri3";
```

バスであるポートにこの属性を適用するには、バスのビットすべてを次のように指定します。

```
attribute black_box_tri_pins of bbl : component is "bus1[7:0]";
```

syn_force_seq_prim

syn_force_seq_prim は、ゲート付きクロックをこのブラック ボックス用に固定するかどうか指定します。固定されたゲート クロック アルゴリズムは、関連するプリミティブに適用できます。**syn_force_seq_prim** は、Synplify Pro および Synplify Premier でのみ使用できます。

ブラック ボックスと共に syn_force_seq_prim を使用するには、クロック信号に **syn_isclock** 制約を付け、イネーブル信号に **syn_gatedclk_clock_en** 制約性を付ける必要があります。データ型はブール型です。

Verilog 構文

```
object /* synthesis syn_force_seq_prim = 1 */ ;
```

説明

- object は、ブラック ボックスのモジュール名です。

Verilog 構文例

```
module bbe (ena, clk, data_in, data_out)
/* synthesis syn_black_box */
/* synthesis syn_force_seq_prim=1 */ ;
input clk /* synthesis syn_isclock = 1 */
/* synthesis syn_gatedclk_clock_en="ena" */;
input data_in,ena;
output data_out;
endmodule
```

VHDL 構文

```
attribute syn_force_seq_prim of object: objectType is true ;
```

説明

- object は、ブラック ボックスのエンティティ名です。

VHDL 構文例

```
library ieee;
use ieee.std_logic_1164.all;
entity bbam is
port (addr:IN std_logic_VECTOR(6 downto 0);
din:IN std_logic_VECTOR(7 downto 0);
dout :OUT std_logic_VECTOR(7 downto 0);
clk:IN std_logic;
en:IN std_logic;
we:IN std_logic);
attribute syn_black_box : boolean ;
attribute syn_black_box of bbam : entity is true ;
attribute syn_isclock : boolean;
attribute syn_isclock of clk: signal is true;
attribute syn_gatedclk_clock_en : string;
attribute syn_gatedclk_clock_en of clk : signal is "en";
end entity bbam;
architecture bb of bbam is
attribute syn_force_seq_prim : boolean;
```

```
attribute syn_force_seq_prim of bb : architecture is true;  
begin  
end architecture bb;
```

syn_gatedclk_clock_en

syn_gatedclk_clock_en では、ゲート付きクロックの固定に使用されるイネーブル ピンを指定します。ブラック ボックスと共に syn_gatedclk_clock_en を使用するには、次を実行しておく必要があります。

- クロック信号に syn_isclock 制約を付けます。
- syn_force_seq_prim 制約で固定されたゲート付きクロック アルゴリズムが適用されるように指定します。

データ型は、string (文字列) です。

Verilog 構文

```
object /* synthesis syn_gatedclk_clock_en = "value" */ ;
```

説明

- object は、モジュール名です。
- value はイネーブル ピンの名前です。

Verilog 構文例

```
module bbe (ena, clk, data_in, data_out)
/* synthesis syn_black_box */
/* synthesis syn_force_seq_prim=1 */;
input clk
/* synthesis syn_isclock = 1 */
/* synthesis syn_gatedclk_clock_en="ena" */;
input data_in,ena;
output data_out;
endmodule
```

VHDL 構文

```
attribute syn_gatedclk_clock_en of object: objectType is value ;
```

説明

- object は、ブラック ボックスのエンティティ名です。

VHDL 構文例

```
architecture top of top is component bbbram
port (myclk : in bit;
opcode : in bit_vector(2 downto 0);
a, b : in bit_vector(7 downto 0);
rambus : out bit_vector(7 downto 0) );
end component;
attribute syn_black_box : boolean;
attribute syn_black_box of bbbram: component is true;
attribute syn_force_seq_prim : boolean
attribute syn_force_seq_prim of bbbram: component is true;
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;
attribute syn_gatedclk_clock_en : string;
```

```
attribute syn_gatedclk_clock_en of bbam: signal is "ena  
//Other code
```

syn_gatedclk_clock_en_polarity

syn_gatedclk_clock_en_polarity では、ブラック ボックスのクロック イネーブル ポートの極性を指定します。この制約を設定すると、合成ツールでゲート付きクロックを固定するアルゴリズムが適用されます。この制約で極性を設定しない場合、合成ツールでデフォルトの正の極性に設定されます。

Verilog 構文

```
object /* synthesis syn_gatedclk_clock_en_polarity = 1 | 0 */ ;
```

説明

- object は、ブラック ボックスのモジュール名です。

値は 1 または 0 にできます。1 の場合はイネーブル信号の極性は正 (アクティブ **High**)、0 の場合は負 (アクティブ **Low**) を示します。この制約を設定しない場合、合成ツールでデフォルトの正の極性に設定されます。

Verilog 構文例

```
module bbel (ena, clk, data_in, data_out)
/* synthesis syn_black_box */
/* synthesis syn_force_seq_prim=1 */;
input clk /* synthesis syn_isclock = 1 */
/* synthesis syn_gatedclk_clock_en="ena" */
/* synthesis syn_gatedclk_clock_en_polarity = 0 */;
input data_in,ena;
output data_out;
endmodule
```

VHDL 構文

```
attribute syn_gatedclk_clock_en_polarity of object:objectType is true |
false;
```

VHDL 構文例

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity bbel is
port (ena : in std_logic;
clk : in std_logic;
data_in : in std_logic;
data_out : out std_logic );
attribute syn_black_box : boolean;
attribute syn_force_seq_prim : boolean;
attribute syn_gatedclk_clock_en_polarity : boolean;
attribute syn_gatedclk_clock_en_polarity of clk: signal is false;
attribute syn_gatedclk_clock_en : string;
attribute syn_isclock : boolean;
attribute syn_isclock of clk : signal is true;
attribute syn_gatedclk_clock_en of clk: signal is "ena";
attribute syn_force_seq_prim of clk: signal is true ;
```

```
end bbel;
architecture arch_bbel of bbel is
attribute syn_black_box : boolean;
attribute syn_black_box of arch_bbel: architecture is true;
attribute syn_force_seq_prim of arch_bbel: architecture is true;
begin
end arch_bbel;
```

syn_isclock

syn_isclock では、ブラック ボックスの入力ポートをクロックとして指定します。ブラック ボックスの入力ポートが認識された名前に対応していなくても、クロックとして指定されます。**syn_isclock** を使用すると、ブラック ボックスの入力ポートが適切な場合にクロック バッファに接続されます。データ型はブール型です。

Verilog 構文

```
object /* synthesis syn_isclock = 1 */ ;
```

説明

- object は、ブラック ボックスの入力ポートです。

Verilog 構文例

```
module ram4 (myclk,out,opcode,a,b) /* synthesis syn_black_box */;
output [7:0] out;
input myclk /* synthesis syn_isclock = 1 */;
input [2:0] opcode;
input [7:0] a, b;
//Other code
```

VHDL 構文

```
attribute syn_isclock of object:objectType is true ;
```

説明

- object は、ブラック ボックスの入力ポートです。

VHDL 構文例

```
library synplify;
entity ram4 is
port (myclk : in bit;
opcode : in bit_vector(2 downto 0);
a, b : in bit_vector(7 downto 0);
rambus : out bit_vector(7 downto 0) );
attribute syn_isclock : boolean;
attribute syn_isclock of myclk: signal is true;
// Other code
```

syn_tpdn

syn_tpdn は、ブラック ボックスを通る組み合わせ遅延のタイミング伝搬に関する情報を提供します。**syn_tpdn** は、SCOPE エディタの [Attribute] パネルを使用して属性として入力できます。オブジェクト、属性、値のフィールドへの情報は、手動で入力する必要があります。

Verilog 構文

```
object /* syn_tpdn = "bundle -> bundle = value" */ ;
```

説明

- **bundle** は、バスおよびスカラ信号のコレクションです。

bundle へ値を代入するには、次の構文を使用します。値の単位は ns です。

```
"bundle -> bundle = value"
```

bundle のオブジェクトは、スペースなしのカンマ区切りにする必要があります。たとえば、3 つの信号をリストする場合は、「A,B,C」になります。

Verilog 構文例

次の例では、ブラック ボックス タイミング制約と共に **syn_tpdn** を定義しています。

```
module ram32x4(z,d,addr,we,clk); /* synthesis syn_black_box
syn_tpd1="addr[3:0]->z[3:0]=8.0"
syn_tsu1="addr[3:0]->clk=2.0"
syn_tsu2="we->clk=3.0" */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;
endmodule
```

VHDL 構文

```
attribute syn_tpdn of object : objectType is "bundle -> bundle = value"
;
```

説明

- **bundle** は、バスおよびスカラ信号のコレクションです。

bundle へ値を代入するには、次の構文を使用します。値の単位は ns です。

```
"bundle -> bundle = value"
```

bundle のオブジェクトは、スペースなしのカンマ区切りにする必要があります。たとえば、3 つの信号をリストする場合は、「A,B,C」になります。

VHDL 構文例

VHDL の場合、Synplify ライブラリに指示子それぞれに対して定義済みインスタンスが次のように 10 個あります。次に例を示します。

```
syn_tpd1, syn_tpd2, syn_tpd3, ... syn_tpd10
```

ソース コードにタイミング制約を入力している場合に、制約のそれぞれに異なる 10 個以上のタイミング遅延値が必要な場合は、10 よりも大きい整数を付けて制約を宣言します。

```
attribute syn_tpd11 : string;
attribute syn_tpd11 of bitreg : component is "di0,di1 -> do0,do1 = 2.0";
attribute syn_tpd12 : string;
attribute syn_tpd12 of bitreg : component is "di2,di3 -> do2,do3 = 1.8";
```

次の例では、いくつかのブラック ボックス制約と共に `syn_tpdn` を定義しています。

```
-- A USE clause for the Synplify Attributes package was included
-- earlier to make the timing constraint definitions visible here.
architecture top of top is
component rcfl6x4z
port (ad0, ad1, ad2, ad3 : in std_logic;
di0, di1, di2, di3 : in std_logic;
clk, wren, wpe : in std_logic;
tri : in std_logic;
do0, do1, do2, do3 : out std_logic );
end component;
attribute syn_tpd1 of rcfl6x4z : component is
"ad0,ad1,ad2,ad3 -> do0,do1,do2,do3 = 2.1";
attribute syn_tpd2 of rcfl6x4z : component is "tri -> do0,do1,do2,do3 = 2.0";
attribute syn_tsu1 of rcfl6x4z : component is "ad0,ad1,ad2,ad3 -> clk = 1.2";
attribute syn_tsu2 of rcfl6x4z : component is "wren,wpe -> clk = 0.0";
// Other code
```

SDC ファイルの構文

```
define_attribute {v:blackboxModule} syn_tpdn { bundle -> bundle = value}
```

説明

- **v:** は制約がビューに適用されたことを示します。
- **blackboxModule** は、ブラック ボックス 7 のシンボル名です。
- **n** は、複数の信号/バンドルの異なる入力から出力へのタイミング遅延を指定できるアルファベット/数字の接尾語です。
- **bundle** は、バスおよびスカラ信号のコレクションです。**bundle** のオブジェクトは、スペースなしのカンマ区切りにする必要があります。たとえば、3 つの信号をリストする場合は、「A,B,C」になります。
- **value** は、出力遅延値への入力 (ns) です。

SDC ファイルの構文例

```
define_attribute {v:MEM} syn_tpd1 {MEM_RD->DATA_OUT[63:0]=20}
```

syn_tcon

syn_tcon では、ブラック ボックスを通るクロックから出力までのタイミング遅延を提供します。**syn_tcon** は、SCOPE エディタの [Attribute] パネルを使用して属性として入力できます。オブジェクト、属性、値のフィールドへの情報は、手動で入力する必要があります。

Verilog 構文

```
object /* syn_tcon = "[!]clock -> bundle = value" */ ;
```

説明

- **bundle** は、バスおよびスカラ信号のコレクションです。**bundle** へ値を代入するには、次の構文を使用します。値の単位は ns です。

```
"[!]clock -> bundle = value"
```

- **!** は、クロックの負のエッジを示すオプションの感嘆符です。**bundle** のオブジェクトは、スペースなしのカンマ区切りにする必要があります。たとえば、3 つの信号をリストする場合は、「A,B,C」になります。

Verilog 構文例

次の例では、ほかのブラック ボックス制約を使用して **syn_tcon** を定義しています。

```
module ram32x4(z,d,addr,we,clk);
/* synthesis syn_black_box syn_tco1="clk->z[3:0]=4.0"
syn_tpd1="addr[3:0]->z[3:0]=8.0"
syn_tsu1="addr[3:0]->clk=2.0"
syn_tsu2="we->clk=3.0" */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;
endmodule
```

VHDL 構文

```
attribute syn_tcon of object : objectType is "[!]clock -> bundle = value" ;
```

説明

- **bundle** は、バスおよびスカラ信号のコレクションです。**bundle** へ値を代入するには、次の構文を使用します。値の単位は ns です。

```
"[!]clock -> bundle = value"
```

- **!** クロックの負のエッジを示すオプションの感嘆符です。**bundle** のオブジェクトは、スペースなしのカンマ区切りにする必要があります。たとえば、3 つの信号をリストする場合は、「A,B,C」になります。

VHDL の場合、Synplify ライブラリに指示子それぞれに対して定義済みインスタンスが次のように 10 個あります。次に例を示します。

```
syn_tco1, syn_tco2, syn_tco3, ... syn_tco10
```

ソース コードにタイミング制約を入力している場合に、制約のそれぞれに異なる 10 個以上のタイミング遅延値が必要な場合は、10 よりも大きい整数を付けて制約を宣言します。

VHDL 構文例

```
attribute syn_tcoll1 : string;
attribute syn_tcoll1 of bitreg : component is "clk -> do0,do1 = 2.0";
attribute syn_tcoll2 : string;
attribute syn_tcoll2 of bitreg : component is "clk -> do2,do3 = 1.8";
```

次の例では、ブラック ボックス制約と共に `syn_tcon` を割り当てています。

```
-- A USE clause for the Synplify Attributes package
-- was included earlier to make the timing constraint
-- definitions visible here.
architecture top of top is
  component Dpram10240x8
  port (
    -- Port A
    ClkA, EnA, WeA: in std_logic;
    AddrA : in std_logic_vector(13 downto 0);
    DinA : in std_logic_vector(7 downto 0);
    DoutA : out std_logic_vector(7 downto 0);
    -- Port B
    ClkB, EnB: in std_logic;
    AddrB : in std_logic_vector(13 downto 0);
    DoutB : out std_logic_vector(7 downto 0) );
  end component;
  attribute syn_black_box : boolean;
  attribute syn_tsu1 : string;
  attribute syn_tsu2 : string;
  attribute syn_tco1 : string;
  attribute syn_tco2 : string;
  attribute syn_isclock : boolean;
  attribute syn_black_box of Dpram10240x8 : component is true;
  attribute syn_tsu1 of Dpram10240x8 : component is
    "EnA,WeA,AddrA,DinA -> ClkA = 3.0";
  attribute syn_tco1 of Dpram10240x8 : component is "ClkA -> DoutA[7:0] = 6.0";
  attribute syn_tsu2 of Dpram10240x8 : component is "EnB,AddrB -> ClkB = 3.0";
  attribute syn_tco2 of Dpram10240x8 : component is "ClkB -> DoutB[7:0] = 13.0";
  // Other code
```

SDC ファイルの構文

```
define_attribute {v:blackboxModule} syn_tcon { [!]clock -> bundle =
value}
```

説明

- **v**: は制約がビューに適用されたことを示します。
- **blackboxModule** は、ブラック ボックスのシンボル名です。
- **n** は、複数の信号/バンドルの異なるクロックから出力へのタイミング遅延を指定できる数字の接尾語です。
- **!** は、クロックが立ち下がり (負の) エッジでアクティブであることを示すオプションの感嘆符です。
- **clock** は、クロック信号の名前です。
- **bundle** は、バスおよびスカラ信号のコレクションです。

bundle のオブジェクトは、スペースなしのカンマ区切りにする必要があります。たとえば、3 つの信号をリストする場合は、「A,B,C」になります。

- value は、出力遅延値へのクロック (ns) です。

SDC ファイルの構文例

```
define_attribute {v:RCV_CORE} syn_tco1 {CLK-> R_DATA_OUT[63:0]=20}  
define_attribute {v:RCV_CORE} syn_tco2 {CLK-> DATA_VALID=30<n>}
```

syn_tsun

syn_tsun には、次のような特徴があります。

- ブラック ボックスの入力ピンに必要なタイミング セットアップ遅延に関する情報を提供します。
- SCOPE エディタの [Attribute] パネルを使用して属性として入力できます。

オブジェクト、属性、値のフィールドへの情報は、手動で入力する必要があります。

Verilog 構文

```
object /* syn_tsun = "bundle -> [! ]clock = value" */ ;
```

説明

- **bundle** は、バスおよびスカラ信号のコレクションです。

bundle へ値を代入するには、次の構文を使用します。値の単位は **ns** です。

```
"bundle -> [! ]clock = value"
```

- **!** は、クロックの負のエッジを示すオプションの感嘆符です。

bundle のオブジェクトは、スペースなしのカンマ区切りにする必要があります。たとえば、3 つの信号をリストする場合は、「A,B,C」になります。

Verilog 構文例

次の例では、ブラック ボックス制約と共に **syn_tsun** を定義しています。

```
module ram32x4(z,d,addr,we,clk);
/* synthesis syn_black_box syn_tpd1="addr[3:0]->z[3:0]=8.0"
syn_tsu1="addr[3:0]->clk=2.0" syn_tsu2="we->clk=3.0" */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;
endmodule
```

VHDL 構文

```
attribute syn_tsun of object : objectType is "bundle -> [! ]clock = value" ;
```

VHDL の場合、Synplify ライブラリに指示子それぞれに対して定義済みインスタンスが次のように 10 個あります。次に例を示します。

```
syn_tsu1, syn_tsu2, syn_tsu3, ... syn_tsu10
```

ソース コードにタイミング制約を入力している場合に、制約のそれぞれに異なる 10 個以上のタイミング遅延値が必要な場合は、10 よりも大きい整数を付けて制約を宣言します。

VHDL 構文例

```
attribute syn_tsul1 : string;
attribute syn_tsul1 of bitreg : component is "di0,di1 -> clk = 2.0";
attribute syn_tsul2 : string;
attribute syn_tsul2 of bitreg : component is "di2,di3 -> clk = 1.8";
```

説明

- **bundle** は、バスおよびスカラ信号のコレクションです。
bundle へ値を代入するには、次の構文を使用します。値の単位は **ns** です。
`"bundle -> [!]clock = value"`
- **!** は、クロックの負のエッジを示すオプションの感嘆符です。
bundle のオブジェクトは、スペースなしのカンマ区切りにする必要があります。たとえば、3 つの信号をリストする場合は、「**A,B,C**」になります。

上記のコードで使用される構文だけでなく、次の Verilog 構文を使用してもこの属性を指定できます。

```
attribute syn_tsul of inputfifo_coregen : component is "rd_clk->dout[48:0]=3.0";
```

次の例では、ほかのブラック ボックス制約と共に **syn_tsun** を割り当てています。

```
-- A USE clause for the Synplify Attributes package
-- was included earlier to make the timing constraint
-- definitions visible here.
architecture top of top is
  component rcfl6x4z
  port (ad0, ad1, ad2, ad3 : in std_logic;
        di0, di1, di2, di3 : in std_logic;
        clk, wren, wpe : in std_logic;
        tri : in std_logic;
        do0, do1, do2, do3 : out std_logic );
  end component;
  attribute syn_tcol of rcfl6x4z : component is
    "ad0,ad1,ad2,ad3 -> do0,do1,do2,do3 = 2.1";
  attribute syn_tpd2 of rcfl6x4z : component is "tri -> do0,do1,do2,do3 = 2.0";
  attribute syn_tsul of rcfl6x4z : component is "ad0,ad1,ad2,ad3 -> clk = 1.2";
  attribute syn_tsu2 of rcfl6x4z : component is "wren,wpe -> clk = 0.0";
  // Other code
```

SDC ファイルの構文

```
define_attribute {v:blackboxModule} syn_tsun { bundle -> [!]clock =
value}
```

説明

- **v:** は、制約がビューに適用されたことを示します。
- **blackboxModule** は、ブラック ボックスのシンボル名です。
- **nA** は、複数の信号/バンドルの異なるクロックから出力へのタイミング遅延を指定できる数字の接尾語です。
- **!** は、クロックが立ち下がり (負の) エッジでアクティブであることを示すオプションの感嘆符です。
- **clock** は、クロック信号の名前です。

- `bundle` は、バスおよびスカラ信号のコレクションです。
`bundle` のオブジェクトは、スペースなしのカンマ区切りにする必要があります。たとえば、3 つの信号をリストする場合は、「A,B,C」になります。
- `valueInput` は、クロックのセットアップ遅延値です。

SDC ファイルの構文例

```
define_attribute {v:RTRV_MOD} syn_tsu4 {RTRV_DATA[63:0]->!CLK=20}
```

SDC ファイル (Tcl) でのタイミング制約の指定

SDC (Tcl) ファイルに Tcl コマンドを書き込みます。

拡張子は .sdc です。このファイルには、タイミング制約、一般制約およびベンダー専用の制約が含まれます。

制約ファイルは Tcl コマンドを使用してテキスト エディタで手動で作成できますが、通常は SCOPE スプレッドシートを使用して自動的に生成します。

次のセクションは、Tcl タイミング制約の各タイプをリストしています。

- 「define_clock」
- 「define_clock_delay」
- 「define_compile_point」
- 「define_current_design」
- 「define_false_path」
- 「define_input_delay」
- 「define_io_standard」
- 「define_multicycle_path」
- 「define_output_delay」
- 「define_path_delay」
- 「define_reg_input_delay」
- 「define_reg_output_delay」

define_clock

define_clock は、特定のデューティ サイクルと周波数またはクロック周期目標でクロックを定義するための制約です。異なるクロック周波数で複数のクロックを含めることができます。

すべてのクロックにデフォルトの周波数を設定するには、プロジェクト ファイルで **Tcl** コマンドの **set_option -frequency** を使用します。グローバル周波数を指定しない場合は、デフォルトの周波数が使用されます。

define_clock timing 制約を使用すると、次が実行されます。

- デフォルトの設定が上書きされます。
- 特定のクロック信号に対してそれぞれ別のクロック周波数目標が指定されます。

define_clock を使用すると、クロック分周器ロジックのクロック信号出力のクロック周波数を設定することもできます。クロック名は、レジスタ インスタンスの出力信号名になります。

構文

```
define_clock [ -disable ] [ -virtual ] {clockObject} [ -freq MHz | -period ns ] [ -clockgroup domain ] [ -rise value -fall value ] [ -route ns ] [ -name clockName ] [ -comment textString ]
```

説明

- **disable** は、前のクロック制約をディスエーブルにします。
- **virtual** は、合成しているチップ (またはブロック) 外部のクロックでイネーブルになる最上位レベルのポートの到着時間と必要時間を指定します。

仮想ブロックに対して **-name** を指定すると、フィールドにデザインのポートまたはインスタンスと関係のない独自の名前を含めることができます。

- **clockObject** は、クロック オブジェクト名を指定するために必要なパラメータです。

クロックは、次に定義できます。

- 最上位レベルの入力ポート (p:)
- ネット (n:)
- 階層ポート (t:)
- インスタンス (i:)

ザイリンクス デバイスの場合は、インスタンスに **define_clock** 制約を指定してください。

- インスタンシエート済みセルの出力ピン (t:)
- インスタンシエート済みセルの内部ピン (t:)

クロックは、次のエレメントには定義できません。

- 最上位レベルの出力ポート
- インスタンシエート済みゲートの入力ピン
- 推論済みインスタンスのピン
- **name** は、クロック オブジェクト名以外の名前をクロック名に付ける場合に使用します。このエイリアス名はタイミング レポートにも表示されます。
- **freq** はクロック周波数 (MHz) を定義します。**freq** か **period** のどちらか一方を指定するようにしてください。

- **period** は、クロック周波数 (ns) を指定します。**freq** か **period** のどちらか一方を指定するようにしてください。

- **clockgroup** では、クロックの関係を指定できます。

関連 (同期) するクロック同士を同じクロック グループに割り当てたり、関連しないクロックを別々のクロック グループに割り当てることができます。合成ツールでは、同じクロック グループのクロック間の関係が計測され、それらの間のすべてのパスが解析されます。異なるグループのクロック間のパスは、無視されます (**False** パス)。

- **rise/fall** では、デフォルト以外のデューティ サイクルを指定します。

合成ツールでは、クロックは 0 で立ち上がり、周期/2 で立ち下がる 50% のデューティ サイクルになります (デフォルト)。これ以外のデューティ サイクルにするには、**Rise At** 値と **Fall At** 値を指定する必要があります。

- **route** は、このクロックで制御されるすべてのレジスタのパス遅延を改善するためのアドバンス ユーザー オプションです。

route の値は、合成タイミング レポートのパス遅延と配置配線タイミング レポートのパス遅延の値の差異になります。

route 制約は、クロック ドメインに対してグローバルに適用されるので、制約が必要のないレジスタにも制約が付いてしまうことがあります。

このオプションを使用する前に、最適化タイミング レポートで各レジスタのパス遅延を確認し、**define_reg_input_delay** および **define_reg_output_delay** 制約を必要とするレジスタにのみ適用して、遅延を改善するようにしてください。

define_clock の構文例

次の例では、クロックは **myInst1** および **myInst2** インスタンスの **Q** ピンに定義されています。

```
define_clock {CLK1} -period 10.0 -clockgroup default_clkgroup
define_clock {CLK3} -period 5.0 -clockgroup default_clkgroup
-uncertainty 0.2 -name INT_REF3
define_clock -virtual {CLK2} -period 20.0 -clockgroup g2
define_clock {CLK4} -period 20.000 -clockgroup g3 -rise 1.000 -fall
11.000 -ref_rise 0.000 -ref_fall 10.000
define_clock Pin-Level Constraint Examples
define_clock {i:myInst1.Q} -period 10.000 -clockgroup default -rise
0.200 -fall 5.200 -name myff1
define_clock {i:myInst2.Q} -period 12.000 -clockgroup default -rise
0.400 -fall 5.400 -name myff2
```

define_clock_delay

`define_clock_delay` は、クロック間の遅延を定義します。デフォルトでは、**define_clock** コマンドを使用して定義したクロック パラメータに基づいて、合成ツールでクロック遅延が計算されますが、**define_clock_delay** を使用して遅延値を指定すると、合成ツールの計算結果は上書きされます。タイミング レポートのクロック関係 (Clock Relationships) セクションに表示される結果は、この制約を使用して計算されています。

構文

```
define_clock_delay [-rise|fall ] {clockName1} [-rise|fall ] {clockName2} delayValue
```

説明

- `rise|fall` は、クロック エッジを指定します。
- `clockName` は制約を付けるクロックを指定します。
クロックは、**define_clock** であらかじめ定義されている必要があります。
- `delayValue` は 2 つのクロック間の遅延 (ns) を指定します。
`false` 値を指定すると、パスを **False** パスとして定義することもできます。

構文例

```
Define_clock_delay -rise {clk0} -rise {clk2x} 2
```

define_compile_point

define_compile_point コマンド :

- Synplify Pro および Synplify Premier でのみ使用できます。
- 最上位レベルの制約ファイルでのコンパイル ポイントを定義します。

定義するコンパイル ポイントごとに **define_compile_point** コマンドを 1 つずつ使用します。

構文

```
define_compile_point [ -disable ] { regionName | moduleName } -type { locked }  
[-cpfile { } ] [ -comment textString ]
```

説明

- **disable** は、前のコンパイル ポイントの定義ディスエーブルにします。
- **type** はコンパイル ポイントのタイプを指定します。これは必ずロックしてください。
- **cpfile** は、Synplicity 内部でのみ使用できます。

構文例

```
define_compile_point {v:work.prgm_cntr} -type {locked}
```

define_current_design

define_current_design コマンドでは、次が実行できます。

- Synplify Pro および Synplify Premier でのみ使用できます。
- コンパイル ポイント領域またはモジュールにその後続く制約を設定します。
- compile-point 制約ファイルの最初コマンドにする必要があります。

構文

```
define_current_design {regionName | libraryName.moduleName }
```

構文例

```
define_current_design {lib1.prgm_cntr}
```

このコマンドの後に続くすべての制約のオブジェクトは、prgm_cntr に関連付けられます。

define_false_path

`define_false_path` では、タイミング解析中に無視 (削除) し、最適化中に低い優先度 (または優先度なし) を割り当てるパスを定義します。この **False** パスはサポートされる配置配線ツールにも渡されます。

構文

```
define_false_path {-from startPoint | -to endPoint | -through throughPoint}  
[-comment textString]
```

説明

- **from** は、False パスの始点を指定します。

from ポイントは、タイミングの開始点を定義し、次のいずれかに指定します。

- クロック (c:)
- レジスタ (i:)
- 最上位レベルの入力ポートまたは双方向ポート (p:)
- ブラック ボックス出力 (i:)

詳細は、『Synopsys FPGA Synthesis Reference Manual』を参照してください。

- **to** は、False パスの終点を指定します。

to ポイントは、タイミングの終点を定義し、次のいずれかに指定します。

- クロック (c:)
- レジスタ (i:)
- 最上位レベルの出力ポートまたは双方向ポート (p:)
- ブラック ボックス入力 (i:)

- **through** は、タイミング例外の中間点を指定します。

中間点は、次のいずれかになります。

- 組み合わせネット (n:)
- 階層ポート (t:)
- インスタンシエート済みセルのピン (t:)

デフォルトでは、**through** ポイントは **OR** リストとして処理されます。パスがリストのポイントを通る場合に、制約が適用されます。

合成でも信号名を維持したい場合は、信号に **syn_keep directive** (Verilog または VHDL) を設定しておきます。

構文例

次の例は、レジスタ間に `define_false_path` を設定する構文を示しています。

```
define_false_path -from {i:myInst1_reg} -through {n:myInst2_net}  
-to {i:myInst3_reg}
```

この制約は、`myInst1_reg` の出力ピンから **myInst2_net** ネットを通過して **myInst3_reg** の入力まで指定されています。インスタンスがインスタンシエートされると、ピン レベルの制約は指定どお

りにピンに適用されますが、インスタンスが推論される場合は、ピン レベルの制約はインスタンスに適用されます。

ピンに **through** ポイントが指定される場合、制約は接続しているネットに適用されます。複数出力のあるインスタンスのピンには、**through** ポイントは定義できません。インスタンスのベクタにピンを指定する場合、そのベクタの 1 ビットを超えるビットは参照できません。

define_input_delay

define_input_delay 制約には、次のような機能があります。

- 最上位レベル ポートの外部入力遅延を指定します。これは、信号が入力ピンに到着する前のチップ外部の遅延です。
- FPGA デバイス入力インターフェイスを外部環境で記述します。合成ツールでは、タイミング制約で指定しておかない限り、入力遅延が検出されません。

構文

```
define_input_delay [ -disable ] { inputportName } | -default ns [ -route  
ns ]  
[ -ref clockName:edge ] [ -comment textString ]
```

説明

- **disable** は、該当ポートの前の遅延指定をディスエーブルにします。
- **inputportName** は、入力ポートの名前です。
- **default** は、すべての入力のデフォルトの入力遅延です。

このオプションを使用すると、すべての入力の入力遅延を設定できます。このデフォルトの制約は、各入力に **define_input_delay** を設定すると上書きされます。

次の例では、デフォルトの入力遅延 **3.0 (ns)** を設定しています。

```
define_input_delay -default 3.0
```

次の例では、**input_a** に遅延 **10.0 ns** を設定して、デフォルトの設定を上書きしています。

```
define_input_delay {input_a} 10.0
```

- **ref (推奨)** は、クロック名およびイベントを促すクロック エッジです。

値には、立ち上がりエッジであるか、立ち下がりエッジであるかを指定する必要があります。

- **r**
立ち上がりエッジ
- **f**
立ち下がりエッジ

次に例を示します。

```
define_input_delay {portb[7:0]} 10.00 -ref clock2:f
```

- **route** は、合成ツールでクロック周波数目標を達成する際の配線遅延を含めるアドバンス オプションです。

配置配線タイミング レポートに、入力ポートを介するロング パスがあるためにタイミング目標が達成されないことを示す記述がある場合は、入力ポートに **-route** オプションを使用します。

構文例

```
define_input_delay {porta[7:0]} 7.8 -ref clk1:r  
define_input_delay -default 8.0  
define_input_delay -disable {resetn}
```

define_io_standard

`define_io_standard` 制約は、特定の Actel、Altera およびザイリンクスのデバイス ファミリで標準の I/O パッド タイプを使用するために指定します。

構文

```
define_io_standard [-disable|-enable] {objectName} -delay_type  
input_delay|output_delay columnTclName{value}  
[columnTclName{value}...]
```

説明

- `delay_type` は、**input_delay** または **output_delay** のいずれかになります。

構文例

```
define_io_standard {DATA1[7:0]} -delay_type input_delay  
syn_pad_type{LVCMOS_33} syn_io_slew{high} syn_io_drive{12}  
syn_io_termination{pulldown}
```

define_multicycle_path

define_multicycle_path 制約には、次のような機能があります。

- 複数のクロック サイクルを使用するため、タイミング例外のパスを指定します。
- タイミング解析および最適化用に指定したパスへ余分なクロック サイクルを提供します。

構文

```
define_multicycle_path [ -start | -end ] { -from startPoint | -to endPoint |  
-through throughPoint } clockCycles [ -comment textString ]
```

説明

- **start|end** は、開始クロックと終了クロックの異なるクロック サイクルがパスで使用されるように指定します。

このオプションを使用すると、クロック周期がクロック距離計算の被乗数として使用されます。**start** または **end** オプションを使用しない場合は、デフォルトで **end** クロックになります。

- **from** はマルチサイクルのタイミング例外の始点を指定します。

from ポイントは、タイミングの開始ポイントを定義し、次のいずれかに指定します。

- クロック (c:)
- レジスタ (i:)
- 最上位レベルの入力ポートまたは双方向ポート (p:)
- ブラック ボックス出力 (i:)

- **to** はマルチサイクルのタイミング例外の終点を指定します。

to ポイントは、タイミングの終了ポイントを定義し、次のいずれかに指定します。

- クロック (c:)
- レジスタ (i:)
- 最上位レベルの入力ポートまたは双方向ポート (p:)
- ブラック ボックス出力 (i:)

- **through** は、タイミング例外の中間点を指定します。

中間点は、次のいずれかに指定できます。

- 組み合わせネット (n:)
- 階層ポート (t:)
- インスタンスシート済みセルのピン (t:)

デフォルトでは、中間点は OR リストとして処理されます。例外は、パスがリストのポイントを通る場合に適用されます。

詳細は、第 5 章「From/To/Through ポイントの指定」の「From/To/Through ポイントの指定」を参照してください。

このオプションを **-to** または **-from** と組み合わせて使用すると、特定のパスを指定できます。合成でも信号名を維持したい場合は、信号に **syn_keep** (Verilog または VHDL) を設定しておきます。

- **clockCycles** は、パス制約に使用するクロック サイクル数を指定します。

タイミング例外制約には、オブジェクト タイプを指定する必要があります。マルチサイクル パスおよび **False** パス制約などのタイミング例外では、インスタンス名のパラメータにそのオブジェクト タイプ (**n:** または **i:** など) をはっきりと指定する必要があります。次に例を示します。

```
define_multicycle_path -from {i:inst2.lowreg_output[7]} -to {i:inst1.DATA0[7]} 2
```

タイミング例外を指定するのに **SCOPE** を使用すると、オブジェクト タイプの修飾子がオブジェクト名に添付されます。

詳細は、『Synopsys FPGA Synthesis Reference Manual』を参照してください。

構文例

```
define_multicycle_path -from{i:regs.addr[4:0]} -to{i:special_regs.w[7:0]} 2
define_multicycle_path -to {i:special_regs.inst[11:0]} 2
define_multicycle_path -from {p:porta[7:0]} -through {n:prgmcntr.pc_sel44[0]} -to
{p:portc[7:0]} 2
define_multicycle_path -from {i:special_regs.trisc[7:0]} -through {t:uc_alu.aluz.Q}
-through {t:special_net.Q} 2
```

次の例は、レジスタ間にマルチサイクル パスを設定する構文を示しています。

```
define_multicycle_path -from {i:myInst1_reg} -through {n:myInst2_net} -to {i:myInst3_reg} 2
```

この制約は、**myInst1_reg** の出力から **myInst2_net** ネットを通過して **myInst3_reg** の入力まで指定されています。インスタンスがインスタンス化されると、ピン レベルの制約は指定どおりにピンに適用されますが、インスタンスが推論される場合は、ピン レベルの制約はインスタンスに適用されます。

ピンに **through** ポイントが指定される場合、制約は接続しているネットに適用されます。複数出力のあるインスタンスのピンには、**through** ポイントは定義できません。インスタンスのベクタにピンを指定する場合、そのベクタの 1 ビットを超えるビットは参照できません。

define_output_delay

define_output_delay 制約には、次のような機能があります。

- 最上位レベルの出力で駆動される **FPGA** 外部のロジックの遅延を指定します。
- **FPGA** デバイス外部のインターフェイスを外部環境で記述します。

FPGA デバイス外のデフォルト遅延は、**0.0 ns** です。出力信号は通常 **FPGA** デバイス外にあるロジックを駆動しますが、合成ツールではタイミング制約で指定しない限り、そのロジックの遅延は検出されません。

構文

```
define_output_delay [ -disable ] { outputportName } | -default ns [ -route ns ]  
[ -ref clockName:edge ] [ -comment textString ]
```

説明

- **disable** は、該当ポートの前の遅延指定をディスエーブルにします。
- **outputportName** は、出力ポートの名前です。
- **default** では、すべての出力のデフォルトの入力遅延を設定します。

このオプションを使用すると、すべての出力の遅延を設定できます。デフォルトの制約は、各入力に **define_output_delay** を設定すると上書きされます。次の例では、デフォルトの出力遅延 **8.0 (ns)** を設定しています。この遅延は、**FPGA** デバイスの外部で発生します。

構文例

```
define_output_delay -default 8.0
```

次の例では、**output_a** に出力遅延 **10.0 ns** を設定して、デフォルトの設定を上書きしています。この結果、**output_a** は関連するクロック エッジよりも前に **10 ns** の組み合わせロジックを駆動します。

```
define_output_delay {output_a} 10.0
```

説明

- **ref** では、クロック名およびイベントを促すクロック エッジを定義します。

値は、次のいずれかになります。

- **r**
立ち上がりエッジ
- **f**
立ち下がりエッジ

次に例を示します。

```
define_output_delay {portb[7:0]} 10.00 -ref clock2:f.
```

- **route** は、合成ツールでクロック周波数目標を達成する際の配線遅延を含めるアドバンス オプションです。

出力パッドのクロック ドメインのデフォルト

デフォルトでは、リファレンス クロックのない **define_output_delay** 制約は、ポートへのパスの開始クロックではなく、グローバル周波数に対して適用されます。合成ツールでは、レジスタとパッドが同じクロック ドメインにはないと仮定されます。これにより、タイミング レポートおよびレジスタとパッド間のすべてのロジックのタイミングドリブン最適化に影響が出ます。

出力遅延制約を設定した出力パッドすべてに対してクロック ドメインを指定する必要があります。クロックを指定しない場合、パッドでは **define_output_delay** 制約に **-ref** オプションを追加します。

```
define_output_delay {LDCOMP} 0.50 -improve 0.00 -route 0.25 -ref {CLK1:r}
```

define_path_delay

`define_path_delay` 制約では、最大遅延および最小遅延制約の **point-to-point** 遅延を **ns** で指定します。開始ポイント、終了ポイント、中間ポイントには次のオプションを使用します。

- **-from**
 - **-to**
 - **-through**
- または
- これらのオプションを組み合わせます。

同じパスに **from** と **to** の両方を指定すると、合成ツールでは 2 つの制約のより制限のある方が使用されます。

`define_path_delay` を指定し、入力または出力遅延も定義した場合、合成ツールではそのパス遅延に入力または出力遅延が追加されます。フォワード アノテートされるタイミング制約には、**I/O** 遅延とパス遅延が含まれます。これにより、ザイリンクスの配置配線ツールで矛盾が発生し、**I/O** 遅延が無視され、パス遅延のみがレポートされることがあります。

構文

```
define_path_delay [-disable] {-from {startPoint} | -to {endPoint} | -through {throughPoint}}  
-max delayValue [-comment textString ]
```

説明

- **disable** は、制約をディスエーブルにします。
- **from** は、パスの始点を指定します。
from ポイントは、タイミングの開始ポイントを定義し、次のいずれかに指定します。

- クロック (c:)
- レジスタ (i:)
- 最上位レベルの入力ポートまたは双方向ポート (p:)
- ブラック ボックス出力 (i:)

- **to** は、パスの終点を指定します。

to ポイントは、タイミングの終点を次のいずれかに指定します。

- クロック (c:)
- レジスタ (i:)
- 最上位レベルの出力ポートまたは双方向ポート (p:)
- ブラック ボックス入力 (i:)

このオプションを **-from** または **-through** と組み合わせて使用すると、特定のパスを指定できます。

- **through** は、タイミング例外の中間点を指定します。

中間点は、次のいずれかに指定できます。

- 組み合わせネット (n:)
- 階層ポート (t:)

- インスタンシエート済みセルのピン (t)

デフォルトでは、中間点は **OR** リストとして処理されます。例外は、パスがリストのポイントを通る場合に適用されます。このオプションを **-to** または **-from** と組み合わせて使用すると、特定のパスを指定できます。合成でも信号名を維持したい場合は、信号に **syn_keep** (Verilog または VHDL) を設定しておきます。

- **max** では、指定したパスの最大許容遅延を設定します。

これはナノ秒 (ns) の絶対値で、タイミング レポートには「max analysis」と表示されます。

構文例

```
define_path_delay -from {i:dmux.alu [5]} -to {i:regs.mem_regfile_15[0]} -max 0.800
```

次の例では、最大遅延 2 ns を **clk1** クロックを使用するフリップフロップの立ち下がりエッジまでのすべてのパスに設定しています。

```
define_path_delay -to {c:clk1:f} -max 2
```

次の例では、レジスタ間のピンにパス遅延制約を設定しています。

```
define_path_delay -from {i:myInst1_reg} -through {t:myInst2_net.Y}  
-to {i:myInst3_reg} -max 0.123
```

この制約は、**myInst1_reg** の出力ピンから **net myInst2_net** ネットの **Y** ピンを通して **myInst3_reg** の入力ピンまで指定されています。インスタンスがインスタンシエートされると、ピン レベルの制約は指定どおりにピンに適用されますが、インスタンスが推論される場合は、ピン レベルの制約はインスタンスに適用されます。

ピンに **through** ポイントが指定される場合、制約は接続しているネットに適用されます。複数出力のあるインスタンスのピンには、**through** ポイントは定義できません。

インスタンスのベクタにピンを指定する場合、そのベクタの 1 ビットを超えるビットは参照できません。

define_reg_input_delay

`define_reg_input_delay` 制約は、時間 (ナノ秒) を指定し、パスがレジスタに入力される時間をスピードアップします。合成ツールは、デザインのグローバル クロック周波数目標を達成しようとするだけでなく、**define_clock** で設定された個別のクロック周波数も達成しようとします。レジスタへのパスの速度を上げるには、この制約を使用します。

構文

```
define_reg_input_delay { registerName } [ -route ns ] [ -comment textString ]
```

説明

- **registerName** は、次のいずれかになります。
 - 単一ビット
 - バス全体、または
 - バスの 1 スライス
- **route** は、再合成中に制約を厳しくするアドバンス ユーザー オプションで、配置配線タイミング レポートに、レジスタへのロング パスがあるためにタイミング目標が達成されないことを示す記述がある場合に使用します。

define_reg_output_delay

`define_reg_output_delay` 制約は、時間 (ナノ秒) を指定し、レジスタからのパスの時間をスピードアップします。合成ツールは、デザインのグローバル クロック周波数目標を達成しようとするだけでなく、**define_clock** で設定された個別のクロック周波数も達成しようとします。レジスタからのパスの速度を上げるには、この制約を使用します。

構文

```
define_reg_output_delay { registerName } [ -route ns ] [ -comment textString ]
```

説明

- **registerName** は、次のいずれかになります。
 - 単一ビット
 - バス全体、または
 - バスの 1 スライス
- **route** は、再合成中に制約を厳しくするアドバンス ユーザー オプションで、配置配線タイミング レポートに、レジスタへのロング パスがあるためにタイミング目標が達成されないことを示す記述がある場合に使用します。

From/To/Through ポイントの指定

このセクションでは、次について説明します。

- 「From/To ポイント」
- 「Through ポイント」
- 「From/To ポイントとしてのクロックの指定」

From/To ポイント

from はタイミング例外の始点を指定します。to はタイミング例外の終点を指定します。次の表に、各項目の詳細を示します。

表 5-2：始点および終点になるオブジェクト

From ポイント	To ポイント
クロック	クロック
レジスタ	レジスタ
最上位レベルの入力ポートまたは 双方向ポート	最上位レベルの出力ポートまたは 双方向ポート
インスタンスシート済みライブラリ プリミティブ セル (ゲート セル)	
ブラック ボックスの出力	ブラック ボックスの入力

1 つの例外に複数の from ポイントを指定できます。これは、バスの全ビットに適用される例外を指定する場合によく使用されます。たとえば、制約を From A[0:15] to B と指定したとします。この場合、A のどのビットからでも開始でき、B で終了するバスに制約が適用されます。

同様に、次を指定することもできます。

- 1 つの例外に複数の from ポイントを指定
- From A[0:15] to B[0:15] のように開始点と終了点を両方とも複数で指定

Through ポイント

through ポイントはネットにしか適用できませんが、さまざまな方法で指定できます。

- 「単一の Through ポイント」
- 「Through ポイントの単一リスト」
- 「複数の Through ポイント」
- 「Through ポイントの複数リスト」

この制約は、次でも定義できます。

- 該当する SCOPE のパネル
- Sum of Products インターフェイス

ポートとネットが同じ名前の場合は、through ポイント名の前に次を付けます。

- n :

ネット

- **t:**
階層ポート
- **p:**
最上位レベルのポート

次に例を示します。

```
n:regs_mem[2] or t:dmux.bdpol
```

n: は、ネットを識別するために指定する必要があります。指定しないと、関連するタイミング制約が有効なネットに適用されません。

単一の Through ポイント

1 つの **through** ポイントは、次のように指定します。

```
define_false_path -through regs_mem[2]
```

この例では、制約が通過するパスすべてに適用されます。

- **regs_mem[2]:**

Through ポイントの単一リスト

through ポイントの単一リストを指定する場合は、次のようになります。

- **OR** ファンクションとして動作します。
- リストのポイントを通過するパスすべてに適用されます。

```
define_path_delay -through {regs_mem[2], prgcntr.pc[7], dmux.alub[0]}  
-max 5 -min 1
```

この例では、制約が次を通過するパスすべてに適用されます。

- **regs_mem[2]**
または
- **prgcntr.pc[7]**
または
- **dmux.alub[0]**

複数の Through ポイント

同じ制約に複数のポイントを指定するには、各ポイントの前に **-through** オプションを付けます。

```
define_path_delay -through regs_mem[2] -through prgcntr.pc[7] -through dmux.alub[0] -max 5  
-min 1
```

この例では、制約が **AND** ファンクションとして動作し、次を通過するパスに適用されます。

- **regs_mem[2]**
AND
- **prgcntr.pc[7]**
AND
- **dmux.alub[0]**

Through ポイントの複数リスト

複数の `-through` リストを指定すると、制約は次のようになります。

- AND/OR ファンクションとして動作します。
- リストのすべてのポイントを通るパスに適用されます。

Through ポイントの複数リスト - 例 1

```
define_false_path -through {A1 A2...An} -through {B1 B2 B3}
```

この例では、制約が通過するパスすべてに適用されます。

- {A1 or A2 or...An}
AND
- {B1 or B2 or B3}

Through ポイントの複数リスト - 例 2

```
define_multicycle_path -through {net1, net2} -through {net3, net4} 2
```

この例では、次のネットを通過するパスすべてに 2 クロック サイクルの制約が適用されます。

```
net1 AND net3
OR net1 AND net4
OR net2 AND net3
OR net2 AND net4
```

From/To ポイントとしてのクロックの指定

タイミング例外制約では、クロックを `from-to` ポイントとして指定できます。

構文

```
define_timing_exception -from | -to { c:clock_name [: edge] }
```

説明

- `timing_exception` は、次の制約タイプのいずれかになります。
 - `multicycle_path`
 - `false_path`
 - `path_delay`
- `c:clock_name:edge` は、クロック名とクロック エッジ (**r** または **f**) です。

クロック エッジを指定しない場合は、デフォルトで両方のエッジが使用されます。

マルチサイクル パスのクロック ポイント

クロックを `from` または `to` ポイントとして指定する場合、マルチサイクル パス制約が指定したクロックの供給されるレジスタすべてに適用されます。

次の例では、`clk1` のクロックが使用されるフリップフロップの立ち上がりエッジからのパスすべてに 2 クロック周期が使用されます。

```
define_multicycle_path -from {c:clk1:r} 2
```

クロックは `through` として指定できませんが、クロックに `from` または `to` 制約を設定し、次のオブジェクトに `through` を設定することはできます。

- ネット
- ピン
- 階層ポート

次の例では、階層ネットのビット 9 を通る、clk1 のクロックが使用されるフリップフロップの立ち下がりエッジまでのパスすべてに 2 クロック周期が使用されます。

```
define_multicycle_path -to {c:clk1:f} -through (n:MYINST.mybus2[9]) 2
```

False パスのクロック ポイント

クロックを from または to ポイントとして指定する場合、False 制約が指定したクロックの供給されるレジスタすべてに適用されます。Timing Analyzer ではすべての False パスが無視されます。

次の例では、clk1 のクロックが使用されるフリップフロップの立ち上がりエッジからのパスすべてをディスエーブルにしています。

```
define_false_path -from {c:clk1:r}
```

クロックは through として指定できませんが、クロックに from または to 制約を設定し、次のオブジェクトに through を設定することはできます。

- ネット
- ピン
- 階層ポート

次の例では、階層ネットのビット 9 を通る、clk1 のクロックが使用されるフリップフロップの立ち下がりエッジまでのパスすべてがディスエーブルされています。

```
define_false_path -to {c:clk1:f} -through (n:MYINST.mybus2[9])
```

パス遅延のクロック ポイント

パス遅延制約のクロックを from または to ポイントとして指定する場合、制約が指定したクロックの供給されるレジスタすべてに適用されます。

次の例では、最大遅延 2 ns を clk1 クロックを使用するフリップフロップの立ち下がりエッジまでのすべてのパスに設定しています。

```
define_path_delay -to {c:clk1:f} -max 2
```

クロックは through として指定できませんが、クロックに from または to 制約を設定し、次のオブジェクトに through を設定することはできます。

- ネット
- ピン
- 階層ポート

次の例では、clk1 のクロックが使用されるフリップフロップの立ち上がりエッジからの階層ネットのビット 9 を通るパスすべてに 0.2 の最大遅延を設定しています。

```
define_path_delay -from {c:clk1:r} -through (n:MYINST.mybus2[9]) -max .2
```

SCOPE スプレッドシートでのタイミング制約の指定

SCOPE (Synthesis Constraints Optimization Environment®) は、タイミング制約や合成属性を入力したり、管理するスプレッドシートのようなインターフェイスです。

SCOPE スプレッドシートを使用すると、制約ファイルが Tcl 形式で生成できます。できる限り、この方法で制約を指定してください。ソース コード 指示子以外のほとんどの制約がこの方法で指定できます。

新しい SCOPE ダイアログ ボックスを作成して開くには、次の手順に従います。

- プロジェクト ビューで [File] → [New] → [Constraint file (SCOPE)] をクリックします。

または

- ツールバーの SCOPE アイコンをクリックします。

Tcl タイミング制約のそれぞれのタイプに対して、同等の SCOPE スプレッドシート インターフェイスがあります。

詳細については、『Synopsys FPGA Synthesis Reference Manual』の「SCOPE and Timing Constraints」の「Scope Constraints」を参照してください。

フォワード アノテーション

合成ツールでは、配置配線ツールに転送してアノートできるベンダー別の制約ファイルが生成されます。制約ファイルは、デフォルトで生成されます。この機能をオフにするには、次をオフにします。

[Project] → [Implementation Option] → [Implementation Results] →
[Write Vendor Constraint File]

ザイリンクスの配置配線ツール用に生成される制約ファイルの拡張子は、.ncf です。

Tcl および SCOPE のセクションに記述されるタイミング制約は、このファイルでザイリンクスにフォワード アノートされます。合成ツールでは、これらの制約だけでなく、異なるクロック間の関係もフォワード アノートされます。

詳細は、次を参照してください。

- 「I/O タイミング制約」
- 「クロック グループ」
- 「フォワード アノテーションされた I/O 制約の緩和」
- 「デジタル クロック マネージャ / 遅延ロック ループ」

I/O タイミング制約

合成ツールは、デフォルトで `define_input_delay` および `define_output_delay` タイミング制約をザイリンクスの NCF ファイルにフォワード アノートします。フォワード アノテーションを制御するのは、`syn_forward_io_constraints` 属性です。

1 または `true` (デフォルト) の場合は、フォワード アノテーションがオンになり、0 または `false` の場合は、オフになります。

この属性は VHDL または Verilog のトップ レベルで使用するか、SCOPE スプレッドシートの [Attributes] パネルを使用して、属性をグローバル オブジェクトとして追加します。

クロック グループ

2 つのクロックが同じクロック グループに含まれる場合、合成ツールでは、フォワード アノテーション用に NCF ファイルが書き出されるので、1 つのクロックがもう 1 つの一部になります。

次の例では、clk1 が clk2 の一部として派生されているので、配置配線ツールに 2 つのクロックが同じクロック グループに属していることが伝わります。

```
NET "clk2" TNM_NET = "clk2";
TIMESPEC "TS_clk2" = PERIOD "clk2" 10.000 ns HIGH 50.00%;
NET "clk1" TNM_NET = "clk1";
TIMESPEC "TS_clk1" = PERIOD "clk1" "TS_clk2" * 2.000000 HIGH 50.00%;
```

次の例では、クロックが個別に宣言されているので、配置配線ツールではこれらのクロックが別々に考慮され、タイミングが計算されます。

```
NET "clk2" TNM_NET = "clk2";
TIMESPEC "TS_clk2" = PERIOD "clk2" 10.000 ns HIGH 50.00%;
NET "clk1" TNM_NET = "clk1";
TIMESPEC "TS_clk1" = PERIOD "clk1" 20.000 ns HIGH 50.00%;
```

フォワード アノテーションされた I/O 制約の緩和

xc_use_timespec_for_io がイネーブル (1) の場合、ザイリンクスの TIMESPEC FROM ... TO コマンドを使用して I/O 制約がフォワード アノテートされます。この場合、制約の条件が緩和されることはありません。

詳細は、『Synopsys FPGA Synthesis Reference Manual』を参照してください。

合成ツールでは、input-to-register、register-to-register および register-to-output パスが FREQUENCY 制約を使用して設定されますが、PERIOD 制約が input-to-register または register-to-output パスに対して厳しすぎる場合は、合成ツールはこれらのパスの制約を緩めようとします。

デジタル クロック マネージャ / 遅延ロック ループ

合成ツールでは、デジタル クロック マネージャ (DCM) と遅延ロック ループ (DLL) の周波数合成および位相シフト機能が使用できます。

オンチップ クロック生成に DLL または DCM を使用している場合、クロックを第一の入力に定義しておく必要があります。合成ツールでは、クロックが任意の数の DLL または DCM を通って伝搬されます。この際、位相シフトまたは周波数変更が考慮され、DLL または DCM の出力にクロックが必要なだけ生成されます。

位相シフトと周波数乗算のパラメータを指定するには、次のようなザイリンクスの標準プロパティを使用します。

- **duty_cycle_correction**
- **clkdv_divide**
- **clkfx_multiply**
- **clkfx_divide**

合成ツールでは、これらのクロックが相互に関係 (同期) していることも考慮され、同じクロック グループに配置されます。ただし、DLL/DCM 入力のクロックだけが NCF ファイルにフォワード アノテートされます。バックエンド ツールでは、この DLL および DCM が理解され、独自のクロック 伝搬が行われます。

タイミング解析

タイミング制約の解析には、**trce** コマンドを使用します。**trce** コマンドは、次のいずれかから実行できます。

- Timing Analyzer
- コマンド ライン

マルチコーナーおよびマルチノードのタイミング解析

マルチコーナーおよびマルチノードのタイミング解析を実行すると、さまざまな PVT (プロセス、電圧、温度) が保証されます。

- デザインは、高速プロセス コーナー (Fast Process Corner) と低速プロセス コーナー (Slow Process Corner) で解析されます。
- ワorstケースのタイミング解析は、タイミング レポートに表示されます。

スピード ファイルの値

高速プロセス コーナーと低速プロセス コーナーのスピード ファイル値は、キャラクタリゼーション データに基づきます。

- 各プロセス コーナーには最大および最小の測定遅延があります。
- 高速プロセス コーナーと低速プロセス コーナーは同時に解析されます。
- 最小値および最大値のプロセス コーナーのパターンは、ワorstケースとしてレポートされます。

この解析は、Virtex クロージャ -6、Spartan クロージャ -6、および 7 シリーズ デバイス ファミリの場合にのみ実行されます。

プロセス コーナー情報

プロセス コーナー情報には、遅延値を特性化するのにどのプロセス コーナーが使用されたかが示されます。

低速プロセス コーナー

低速プロセス コーナーは、次のように定義されます。

- 高温
- 低電圧

これは、典型的なワorst ケースまたは最大 PVT です。

高速プロセス コーナー

高速プロセス コーナーは、次のように定義されます。

- 低温
- 高電圧

これは典型的な絶対最小のスピード グレードです。

ワーストケース解析

ほとんどのデザインは、次のように処理されます。

- 高速プロセス コーナーはワーストケースのホールド解析用にレポートされます。
- 低速プロセス コーナーはワーストケースのセットアップ解析用にレポートされます。

クロックおよびデータ トポロジによっては、高速プロセス コーナーがワーストケースのセットアップ解析用にレポートされることもあります。

非同期リセット パス

リカバリ時間や出力時間までのリセットピンを含む非同期リセット パスの解析は、デフォルトでは **PERIOD** 制約解析に含まれません。

非同期リセット/セット パスを確認するには、パス トレーシング コントロール (PTC) を次のようにイネーブルにする必要があります。

- **ENABLE = REG_SR_R;**

リカバリ時間の場合はこのように設定します。

- **ENABLE = REG_SR_O;**

出力時間の場合はこのように設定します。

これらのパス トレーシング コントロールにより、非同期リセット ピンから同期エレメントのパスおよび同期エレメントのリセット リカバリ時間がイネーブルになります。

非同期 **SR** および **CLR** リカバリ パスは、**REG_SR_R** PTC で制御されます。非同期 **SR** および **CLR** の伝播パスおよび削除パスは、**REG_SR_O** PTC で制御されます。これらの PTC では、タイミング遅延名とタイミング アークが個別に制御されます。

Timing Analyzer

タイミング制約の解析は、**Timing Analyzer** または **trce** コマンドから実行できます。

タイミング解析を実行すると、次が実行されます。

- タイミング制約要件に合ったタイミング パスの詳細な解析が実行されます。
- 特定のタイミング制約がインプリメンテーション ツールに渡されます。

パス別の詳細な解析では、次が実行されます。

- 制約ごとにすべてのパスのタイミング要件が満たされているかどうかを確認します。
- 制約ごとにすべてのパスのセットアップおよびホールド要件が満たされているかどうかを確認します。
- デバイス コンポーネントが動作可能な周波数制限内で実行されているかどうかを確認します。
- 制約の付いていないパス (クリティカル パスが解析されない可能性あり) のリストを提供します。

タイミング レポート

このセクションでは、典型的なタイミング レポートについて説明します。

タイミング レポートの内容

通常のタイミング レポートには、次のセクションが含まれます。

- Constraint Details (制約の詳細) セクション
- Data Sheet (データシート) セクション
- Summary (サマリ) セクション

Constraint Details (制約の詳細) セクション

制約ごとのパスの詳細が表示されます。

Data Sheet (データシート) セクション

次が表示されます。

- 一般的なセットアップ タイム
- 一般的なホールド タイム
- clock-to-out タイム

Summary (サマリ) セクション

次が表示されます。

- タイミング エラー
- タイミング スコア
- 制約の適用範囲
- デザイン統計

パスの詳細

パスの詳細は、各タイミング制約の下に表示されます。含まれる内容は、次のとおりです。

- 次を含んだ制約ヘッダ
 - 解析されるパス
 - エンドポイント解析
 - エラー エンドポイント
 - 検出されたタイミング エラー
- 最小 PERIOD/OFFSET IN
- セットアップ パス
特定のスラック式を使用したセットアップ解析された個別パス
- ホールド パス
特定のスラック式を使用したホールド解析された個別パス
- 特定のスラック式を使用した PERIOD 制約のコンポーネント スイッチ制限

PERIOD 解析

同期エレメントから同期エレメントの解析は、PERIOD 解析で実行されます。PERIOD 制約では、クロック ドメインのタイミング関係が定義されます。

解析には、次が含まれます。

- 1 つのクロック ドメイン内のパス
- 関連するクロック ドメインと関連する PERIOD 制約間のパス
- 周波数/周期、位相、ソースおよびデスティネーションの同期エレメント間の誤差
- 1 つのクロック ドメインのパス
- クロス クロック ドメインのパス

ヘッダー サマリ

PERIOD 制約の解析には、ヘッダー サマリが含まれます。ヘッダー サマリには、次のような制約に関する情報がまとめられます。

- 解析されたパス数およびエンドポイント数
- セットアップ、ホールド、またはコンポーネント スイッチ制限などのエラー

この情報を使用すると、制約が予測通りのエンドポイントやパス数に適用されているかどうかや、この制約の全体的なワーストケース パフォーマンスが検証できます。

コンポーネント スイッチ制限の解析

コンポーネント スイッチ制限の解析では、次が実行されます。

- 通常のセットアップおよびホールド解析に加えて、実行されます。
- デバイス コンポーネントの動作周波数がデバイスの仕様を超えていないかどうかを確認されます。
- 次に対して実行されます。

- 大型コンポーネント (DSP、BRAM など)
- 小型のデバイス コンポーネント (ILOGIC、OLOGIC、SLICE など)
- 制約付きのクロック ドメインのクロック コンポーネント (DCM、PLL など)

よく使用されるコンポーネント スイッチ制限

よく使用されるコンポーネント スイッチの制限は、次のとおりです。

- MINPERIOD
- MINLOWPULSE
- MINHIGHPULSE

その他のコンポーネント スイッチ制限

次のような制限のあるコンポーネントもあります。

- MAXPERIOD
- MAXLOWPULSE
- MAXHIGHPULSE

パス解析の詳細

解析される各パスの詳細は、**PERIOD** 制約のヘッダ サマリの後に表示されます。各パスは、1 つの同期エレメントから別の同期エレメントまでのパスで、デスティネーションの同期エレメントにはセットアップまたはホールド タイミングが設定されます。

PERIOD 制約は、同期エレメント間のこれらのデータ パスに適用されます。最もよくあるクロック ドメインは、次のとおりです。

- 単一のクロック ドメイン
- 2 位相のクロック ドメイン
- 複数のクロック ドメイン

PERIOD 制約の適用されるこれらのパス タイプそれぞれに対して、タイミング レポートの例が提供されています。

最初の段落の内容

最初の段落には、次が表示されます。

- パスの全スラック
- 同期パスのパフォーマンス
- ソース デザイン同期エレメント
- デスティネーション デザイン同期エレメント
- ソース/デスティネーションのクロック信号 (クロック エッジ指定付き)
- データ パス遅延の合計
- クロックのばらつき
- スラック式
- クロックのばらつき式

2 段落目の内容

2 つ目の段落には、次の 2 つの間のデータパスの詳細が表示されます。

- ソース同期エレメント
- デスティネーション同期エレメント

使用されたデバイス リソースやデータパスのネット配線遅延など、データパスを構成する個々のエレメントも含まれます。

クロック ドメイン

PERIOD 制約は、同期エレメント間のこれらのデータパスに適用されます。最もよくあるクロック ドメインは、次のとおりです。

- 1 つのクロック ドメイン
- 2 位相のクロック ドメイン
- 複数のクロック ドメイン

PERIOD 制約の適用されるこれらのパス タイプそれぞれに対して、タイミング レポートの例が提供されています。

ゲート付きクロック

PERIOD 制約では、ゲート付きクロックまたは内部派生クロックが正しく解析されません。クロックがゲートを介したり、LUT (ルックアップ テーブル) を介す場合、タイミング解析では LUT の各入力から信号のソース (同期エレメントまたはパッド) までトレース バックされ、対応する「[クロック スキュー](#)」がレポートされます。

LUT から派生する「[クロック スキュー](#)」は、ロジック レベルや LUT の数によってかなり多くなります。

クロックが DCM でなく内部ロジックを使用して分周される場合、分周フリップフロップのクロック ピンの PERIOD 制約では、次の図のように、このフリップフロップを通る `clk_div` 信号までのパスがトレースされません。

タイミング解析には、新しいゲート付きクロック信号で駆動されるダウンストリームの同期エレメントは含まれません。

グローバル バッファを使用しない限り、分周フリップフロップから派生したこの新しいクロックはローカル配線になります。PERIOD 制約が分周フリップフロップの出力 (次の図の `clk_div` 信号) に設定され、元の PERIOD 制約に関連している場合は、タイミング解析にダウンストリームの同期エレメントが含まれます。

この関係とクロス クロック ドメイン解析が正しく実行されるようにするには、分周クロックと元のクロック間の違いを PHASE キーワードを付けて PERIOD 制約に含める必要があります。「[クロック スキュー](#)」は、2 つのクロック間の関係によって、大きくなることがあります。

PHASE キーワードでは2つのクロック間の違いを定義するので、これがクロス クロック ドメイン パス解析のタイミング制約要件になります。PHASE キーワード値が小さすぎる場合、クロス クロック ドメイン パス解析の要件を満たすことができません

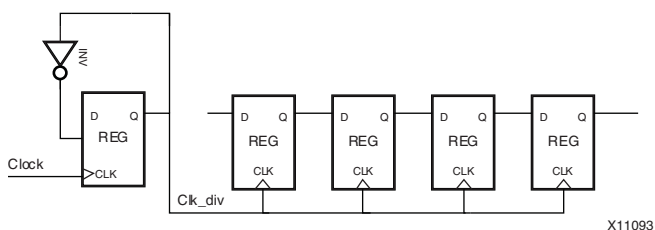


図 6-1：分周ダウンストリーム フリップフロップを使用したゲート付きクロック

単一のクロック ドメイン

クロック ドメインが1つの場合は、解析しやすくなっています。すべての同期エレメントが同じクロック ドメインにあり、クロックの立ち上がりエッジで解析されるか、すべてのエレメントがクロックの立ち下がりエッジで解析されます。

クロック ソースは同じクロック ソースから駆動され、クロック ソースは出力が1つだけの PAD または DCM/DLL/PLL/PMCD になります。

タイミング解析ツールでは、クロック ドライバのアクティブ エッジとソース エレメント間のデータ パスの対応時間がレポートされます。

次の図は、単純なデザイン例を示しています。PERIOD 制約は、ユーザー制約ファイル (UCF) から解析されます。

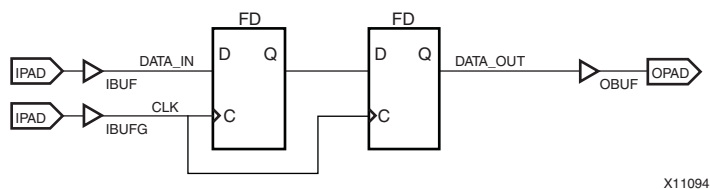


図 6-2：単一のクロック ドメインの回路図

単一のクロック ドメインのタイミング レポートの例

```
Slack (setup path):3.904ns (requirement - (data path - clock path skew + uncertainty))
Source:                IntA_1 (FF)
Destination:           XorA_1 (FF)
Requirement:           8.000ns
Data Path Delay:       4.036ns (Levels of Logic = 1)
Clock Path Skew:       0.000ns
Source Clock:          clk0 rising at 0.000ns
Destination Clock:     clk0 rising at 8.000ns
Clock Uncertainty:     0.060ns
```

2 位相のクロック ドメイン

次の図に示すようなクロックの両方のエッジを使用するデータ パスは、2 位相クロック ドメインまたは 2 位相データ パスとして知られています。

このクロックの出力は 1 つだけで、PAD、DCM/DLL/PLL/PMCD コンポーネントなどの同じクロック ソースで駆動されます。これらの同期エレメントは、DCM/DLL/PLL/PMCD の CLK0 と CLK180 や CLK90 と CLK270 などの 2 つの関連するクロックでも駆動できます。

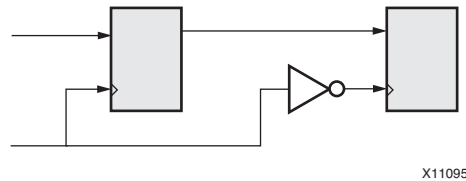


図 6-3：2 位相クロック

タイミング解析ツールでは、ソースおよびデスティネーションの同期エレメントのアクティブ クロック信号とアクティブ クロックの到着時間がレポートされます。ソースおよびデスティネーションの同期エレメントのクロック到着時間の違いによって、データ パスの要件が決まります。2 位相データ パスの場合、データ パス要件は次の図に示すように 1 位相データ パスの要件の一部になります。

タイミング解析ツールでは、データ パスの詳細がスラック値別にレポートされます。スラック値は、データ パス要件とデータ パス遅延間の関係を示すものです。データ パスは、スラック値の一番大きな負の値 (立ち下がり) から一番大きな正の値 (立ち上がり) の順に表示されます。

最小周期の値

一番大きなワースト/負のスラック値のデータ パスが最小周期 (Minimum Period) の値と違っている場合、通常原因は 2 位相データ パスのスラック値がデータ パスのリストの一番上にないことにあります。

ほとんどの場合、リストの一番上のデータ パスは最小周期値と対応しています。2 位相データ パスでも最小周期値に対応していることがあります。2 位相データ パスの場合、タイミング解析ツールで元の 1 位相または完全位相データ パスの要件と 2 位相データ パスの要件の分数関係が決定されます。

この分数値が使用され、2 位相データ パスのデータ パス遅延の合計が 1 位相または完全位相のデータ パス遅延に変換されます。この値が 1/2 の場合、2 位相データ パス遅延が 2 倍されて、完全データ パス遅延に変換されます。最小周期値は、分数のデータ パス遅延ではなく、完全位相データ パス遅延にのみ含まれます。

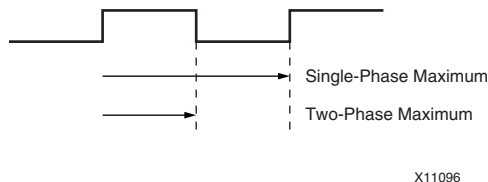


図 6-4：1 位相クロックと 2 位相クロックの関係

PERIOD 制約を使用した 2 位相デザインの例

PERIOD 制約または 6ns の完全位相データパス要件のデザイン例には、完全位相と 2 位相両方のデータパスが含まれます。

表 6-1 : PERIOD 制約を使用した例

データパス	データパス遅延の合計 (ns)	スラック (ns)
完全位相	8	-2
2 位相	4.036	-1.096

完全位相データパスはリストの一番上に表示され、その後に 2 位相データパスが表示されますが、最小周期値は 8.192ns です。最小周期値は、完全データパス遅延ではなく、2 位相データパス遅延に対応しています。

タイミングレポートの例 1

```
Slack (setup path):-1.096ns (requirement - (data path - clock path skew + uncertainty))
Source:IntA_1 (FF)
Destination:XorA_1 (FF)
Requirement:3.000ns
Data Path Delay:4.036ns (Levels of Logic = 1)
Clock Path Skew:0.000ns
Source Clock: clk0 rising at 0.000ns
Destination Clock: clk0 falling at 3.000ns
Clock Uncertainty:0.060ns
```

タイミングレポートの例 2

```
Timing constraint:TS_DRAM_CTRL_U_u_infrastructure_clk_pll = PERIOD TIMEGRP
"DRAM_CTRL_U_u_infrastructure_clk_pll" TS_clk_303 / 0.5 HIGH 50%;
```

```
56924 paths analyzed, 17458 endpoints analyzed, 366 failing endpoints
452 timing errors detected.(366 setup errors, 86 hold errors, 0 component switching limit
errors)
Minimum period is 24447.220ns.
```

```
-----
Paths for end point DRAM_CTRL_U/bank_conflict (SLICE_X39Y106.C3), 31 paths
```

```
-----
Slack (setup path):      -3.666ns (requirement - (data path - clock path skew + uncertainty))
Source:                  DRAM_CTRL_U/out_add[1].rd_addr_fifo/USE_SDPRAM_LUT.sdpram_lut_inst/
depth_le_5.gen_sdpram[0].sdpram32_RAMB (RAM)
Destination:            DRAM_CTRL_U/bank_conflict (FF)
Requirement:            0.002ns
Data Path Delay:        3.146ns (Levels of Logic = 3)(Component delays alone exceeds
constraint)
Clock Path Skew:        -0.250ns (2.637 - 2.887)
Source Clock:           clk_250 rising at 13312.000ns
Destination Clock:      DRAM_CTRL_U/clk rising at 13312.002ns
Clock Uncertainty:      0.272ns

Clock Uncertainty:      0.272ns ((TSJ^2 + DJ^2)^1/2) / 2 + PE
Total System Jitter (TSJ):0.070ns
Discrete Jitter (DJ):   0.157ns
Phase Error (PE):       0.185ns
```

Maximum Data Path at Slow Process Corner: DRAM_CTRL_U/out_add[1].rd_addr_fifo/
USE_SDPRAM_LUT.sdpram_lut_inst/depth_le_5.gen_sdpram[0].sdpram32_RAMB to DRAM_CTRL_U/
bank_conflict

Location	Delay type	Delay(ns)	Physical Resource Logical Resource(s)

SLICE_X42Y107.BMUX	Tshcko	1.492	DRAM_CTRL_U/rd_addr_pre<1><1> DRAM_CTRL_U/out_add[1].rd_addr_fifo/ USE_SDPRAM_LUT.sdpram_lut_inst/depth_le_5.gen_sdpram[0].sdpram32_RAMB
SLICE_X41Y107.D5	net (fanout=3)	0.331	DRAM_CTRL_U/rd_addr_pre<1><2>
SLICE_X41Y107.D	Tilo	0.068	DRAM_CTRL_U/rd_addr_1<2> DRAM_CTRL_U/
Mmux_last_bank[2].last_bank[2].MUX_834_o11			
SLICE_X41Y106.C4	net (fanout=1)	0.502	DRAM_CTRL_U/
Mmux_last_bank[2].last_bank[2].MUX_834_o1			
SLICE_X41Y106.C	Tilo	0.068	DRAM_CTRL_U/n0728<0> DRAM_CTRL_U/
Mmux_last_bank[2].last_bank[2].MUX_834_o12			
SLICE_X39Y106.C3	net (fanout=1)	0.612	DRAM_CTRL_U/
Mmux_last_bank[2].last_bank[2].MUX_834_o11			
SLICE_X39Y106.CLK	Tas	0.073	DRAM_CTRL_U/bank_conflict DRAM_CTRL_U/
Mmux_last_bank[2].last_bank[2].MUX_834_o19			

Total		3.146ns	(1.701ns logic, 1.445ns route) (54.1% logic, 45.9% route)

「タイミング レポートの例 2」の「Minimum period is 24447.220 ns」は、ソース クロックとデスティネーション クロック間のクロック到着時間の関係に基づいて算出されています。タイミング エンジンでは、次が実行されます。

1. これらのクロック ネットワークを解析
2. 2 つの時間の一番近いクロック エッジを決定

この 2 つのクロック エッジがクロック到着時間としてレポートされます。違いは、「Requirement」として定義されます。これは、フルサイクルの **PERIOD** 制約要件の一部です。フルサイクルの **PERIOD** 制約要件は 13.33ns なので、新しい要件と元のフルサイクル要件の関係は 1/6665 になります。

このセットアップ パス解析は、フルサイクルの 1/6665 です。最小周期の値はフルサイクル値です。セットアップ トータル (3.668ns) が 6665 で乗算されると、最小周期は 24,447.220ns になります。

複数のクロック ドメイン

クロス クロック ドメイン パスとは、ソースとデスティネーションの同期エレメントに 2 つの異なるクロックが含まれるパスのことです。1 つのクロックがソースを、もう 1 つのクロックがデスティネーションを駆動します。

ソースのクロックの **PERIOD** 制約がデスティネーションのクロックの **PERIOD** 制約に関連付けられている場合は、デスティネーションのクロック **PERIOD** 制約がクロス クロック ドメイン解析に使用されます。

ザイリンクスでは、**PERIOD** 制約でクロックを関連付ける方法をお勧めしています。この方法を使用すると、解析でクロス クロック ドメイン パスが正しく含まれます。

クロックが関連付けられていない場合は、クロス クロック ドメイン パスは解析されません。ザイリンクスでは **FROM:TO** またはマルチサイクル制約を使用してそのパスを **False** パスまたはマルチサイクル パスとして指定することをお勧めしています。

DCM 出力からのクロック

DCM/DLL/PLL/PMCD からのクロック信号は互いに関連しているため、**PERIOD** 制約同士も関連付ける必要があります。

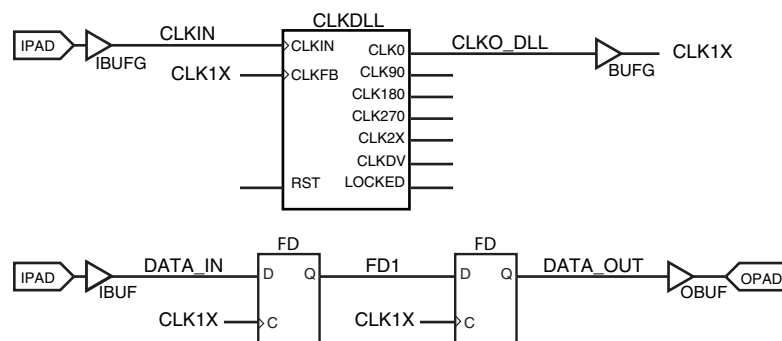
これは、次のいずれかの方法で実行できます。

- NGDBuild で入力クロック信号の **PERIOD** 制約に基づいて新しい **PERIOD** 制約を作成します。
または
- DCM/DLL/PLL/PMCD の出力クロック信号に基づいて手動で **PERIOD** 制約を作成し、その **PERIOD** 制約を手動で関連付けます。

Clk0 クロックドメイン

DCM/PLL/DLL/PMCD からのクロックは関連しているため、タイミング ツールでの解析中にこの関係が考慮されます。同期エレメントのクロック ピンは **DCM/DLL/PLL/PMCD** コンポーネント出力からと同じクロック ネットで駆動されます。タイミング解析ツールでは、クロックのアクティブエッジと同期エレメント間のデータ パスの対応時間がレポートされます。

次の図の例は、単純なデザインを使用して **CLK0** クロックの回路を示しています。このクロックドメインの要件と位相シフトは元の要件と同じです。



X11097

図 6-5 : Clk0 DCM 出力の回路図

Clk0 DCM 出力のタイミング レポートの例

```
Slack (setup path):3.904ns (requirement - (data path - clock path skew + uncertainty))
Source:                IntA_1 (FF)
Destination:           XorA_1 (FF)
Requirement:           8.000ns
Data Path Delay:        4.036ns (Levels of Logic = 1)
Clock Path Skew:        0.000ns
Source Clock:           clk0 rising at 0.000ns
Destination Clock:      clk0 rising at 8.000ns
Clock Uncertainty:      0.060ns
```

Clk90 クロック ドメイン

DCM/PLL/DLL/PMCD からのクロックは関連しているので、タイミング ツールでの解析中にこの関係が考慮されます。同期エレメントのクロック ピンは、DCM/DLL/PLL/PMCD コンポーネント出力と異なるクロック ネットで駆動されます。タイミング解析ツールでは、クロックのアクティブエッジと同期エレメント間のデータ パスの対応時間がレポートされます。

DCM 出力間の Clk90 クロック位相の回路図

次の図の例は、位相差 90 度の CLK0 と CLK90 信号を示しています。

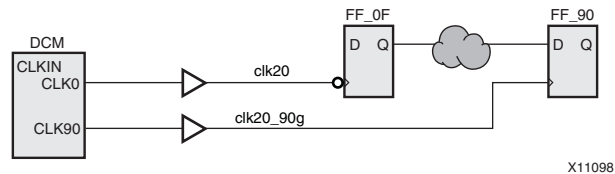


図 6-6：DCM 出力間の Clk90 クロック位相の回路図

位相シフトされたクロックのクロス クロック ドメイン解析でも、最小の PERIOD 値がタイミング レポートにリストされる最初のパスと異なることがあります。

2 つのクロック ドメイン間の位相差が 90 度の場合、データ遅延の合計に 4 を乗算して完全 PERIOD 値にします。

たとえば、この clock90 制約でデータ パス遅延が 1.5ns の場合、完全 PERIOD 値は 6ns になります。

また、この例の場合、次の図に示すように、データ パスは CLK0 クロック信号の立ち下がりエッジから CLK90 クロック信号の立ち上がりエッジまでで、タイミング解析には CLK0 からの 2 位相情報が含まれます。詳細は、次の図を参照してください。

前の図に示すように、元の PERIOD 制約は 20ns に設定されていましたが、このクロス クロック ドメイン解析には新しい要件の 15ns が指定されて、2 つのクロック間の位相差が補正されています。詳細は、次の図を参照してください。

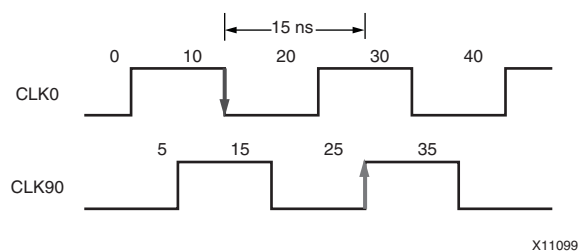


図 6-7：クロック エッジの関係

Clk90 のタイミング レポートの例

```
Slack (setup path):5.398ns (requirement - (data path - clock path skew + uncertainty))
Source:          IntB_2 (FF)
Destination:     XorB_2 (FF)
Requirement:     8.000ns
Data Path Delay:  2.542ns (Levels of Logic = 1)
Clock Path Skew:  0.000ns
Source Clock:     clk0 falling at 2.000ns
Destination Clock: clk90 rising at 10.000ns
```

...

```

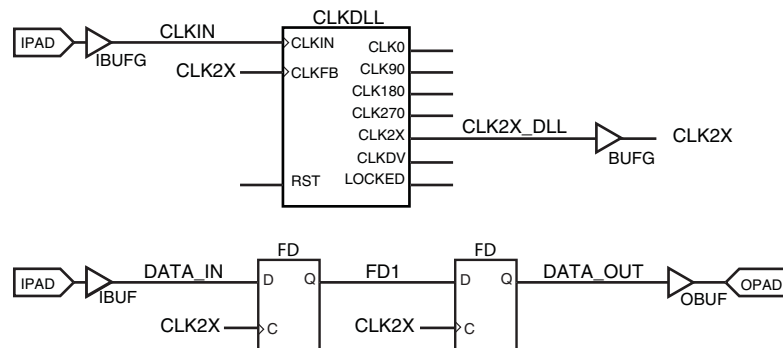
Slack (setup path):13.292ns (requirement - (data path - clock path skew + uncertainty))
Source:                IntC_2 (FF)
Destination:           XorB_2 (FF)
Requirement:           15.000ns
Data Path Delay:        2.594ns (Levels of Logic = 1)
Clock Path Skew:        -0.086ns
Source Clock:           clk0 falling at 10.000ns
Destination Clock:      clk90 rising at 25.000ns
Clock Uncertainty:      0.200ns

```

Clk2x クロックドメイン

DCM/PLL/DLL/PMCD からのクロックは関連しているので、タイミングツールでの解析中にこの関係が考慮されます。次の図は、CLK2X クロックドメインの単純なデザイン例を示しています。クロックは同じクロックソースで駆動されます。このクロックソースは DCM、DLL、PLL、PMCD のいずれかのコンポーネントの出力になります。

タイミング解析ツールでは、クロックのアクティブエッジと同期エレメント間のデータパスの対応時間がレポートされます。このクロックドメインの要件と位相シフトは元の要件と同じです。位相シフトは、元の要件の位相シフトと同じです。



X11100

図 6-8 : Clk2x DCM 出力の回路図

Clk2x のタイミングレポートの例

```

Slack (setup path):-1.663ns (requirement - (data path - clock path skew + uncertainty))
Source:                IntA_3 (FF)
Destination:           OutB_3 (FF)
Requirement:           2.000ns
Data Path Delay:        3.443ns (Levels of Logic = 0)
Clock Path Skew:        -0.020ns
Source Clock:           clk0 rising at 0.000ns
Destination Clock:      clk2x falling at 2.000ns
Clock Uncertainty:      0.200ns

```

CLKDV/CLKFX クロックドメイン

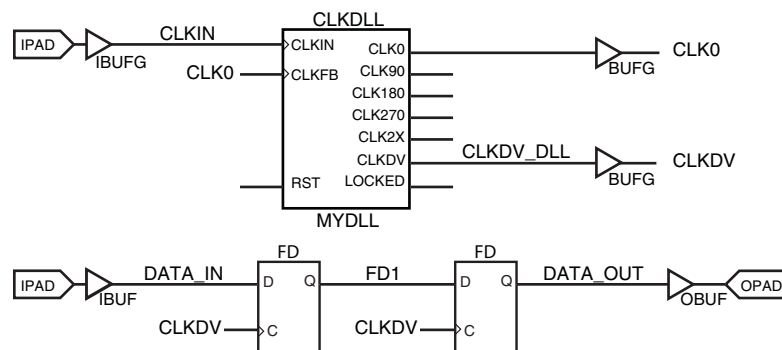
DCM/PLL/DLL/PMCD からのクロックは関連しているので、タイミングツールでの解析中にこの関係が考慮されます。CLKDV および CLKFX 出力は、元の入力クロック信号から派生したのよう

なクロック信号を作成するために使用できます。詳細は、表 3-1、「DCM を介した PERIOD 制約の変換クロージャ」を参照してください。

クロックは DCM/DLL/PLL/PMCD コンポーネントの 2 つの異なる出力で駆動されます。タイミング解析ツールでは、クロックのアクティブ エッジと同期エレメント間のデータ パスの対応時間がレポートされます。

次の図は、DIVIDE_BY が 2 に設定された CLKDV クロック ドメインの単純なデザイン例を示しています。

- このクロック ドメインの要件と位相シフトは元の要件と同じです。
- 位相シフトは、元の要件の位相シフトと同じです。



X11101

図 6-9 : ClkDV DCM 出力の回路図

ClkDV のタイミング レポートの例

```
Slack (setup path):1.909ns (requirement - (data path - clock path skew + uncertainty))
Source: XorC_7 (FF)
Destination: OutC_7 (FF)
Requirement: 4.000ns
Data Path Delay: 1.810ns (Levels of Logic = 0)
Clock Path Skew: 0.000ns
Source Clock: clk0 rising at 0.000ns
Destination Clock: clkdv rising at 4.000ns
Clock Uncertainty: 0.281ns
```

FROM:TO (マルチサイクル) 制約

multicycleanalysis

パス例外の解析は、FROM-TO の設定に従って実行されます。パス例外制約は、FROM:TO 制約で指定された特定のパスに適用され、その部分のグローバル制約は上書きされます。

この制約は、グローバル タイミング制約よりも高速または低速にする必要のある特定のパスにタイミング要件を指定するためのものです。この要件には、値またはタイミング無視 (TIG) 制約を指定します。

ヘッダー サマリ

例外制約の解析は、特定制約のサマリであるヘッダー サマリから開始されます。ヘッダー サマリーには、次が含まれます。

- 制約構文
- この制約が適用されるパス数およびエンドポイント数
- セットアップ エラー
- ホールド エラー

ヘッダー サマリの情報を使用すると、次が実行できます。

- 制約が予測通りパス数やエンドポイントに適用されているかどうか確認できます。
- この制約の全体的なワーストケース パフォーマンスが検証できます。

例外制約パスの解析

例外制約パスの解析には、クロック パスとデータ パスのパス詳細情報が含まれます。この解析には、1 つのパスに関する情報すべてが含まれます。

最初の段落の内容

最初の段落には、次が表示されます。

- 全体的なスラック値
- ソース/デスティネーション デザイン エレメント
- ソース クロック/デスティネーション クロック信号とクロック エッジ
- データ パス遅延の合計
- クロック スキュー
- クロックのばらつき
- スラック式
- クロックのばらつき式

2 段落目の内容

2 つ目の段落には、出力インターフェイスのクロック パスとデータ パスのパス詳細が表示されます。これには、使用されたデバイス リソースすべてとクロック パスとデータ パスの両方の配線遅延が含まれます。

FROM:TO (マルチサイクル) 制約の解析

FROM:TO (マルチサイクル) 制約の解析には、ソースとデスティネーション同期エレメント間のクロック スキューが含まれます。

クロック スキューは、デスティネーション同期エレメントへのクロック パスからソース同期エレメントへのクロック パスの値を引いて計算されます。

クロック スキュー解析は、制約の付けられたすべてのクロックに対して実行されます。解析には、次が含まれます。

- すべてのデバイス ファミリのセットアップ解析
- Virtex-5 以降のデバイスの場合はセットアップとホールド解析

DATAPATHONLY キーワード

DATAPATHONLY キーワードを使用すると、次が実行されます。

- FROM:TO 制約のクロック スキューが無視されます。
- 解析中に FROM:TO 制約でクロック スキューや位相情報が考慮されないように指定できます。
- グループ間のデータ パスのみが解析されます。

セットアップおよびホールド解析

セットアップ パスは次の式で計算されるスラック値別にリストされます。

$$\text{Tsu slack} = \text{constraint_requirement} - \text{Tclock_skew} - \text{Tdata_path} - \text{Tsu}$$

FROM:TO のセットアップ解析はデフォルトで実行されます。ホールド解析がレポートされるのは、Virtex-5 以降のデバイスのみです。

Virtex-5 よりも前のデバイスでホールド解析を実行するには、XIL_TIMING_HOLDCHECKING YES 環境変数を設定する必要があります。

ホールド解析は、データ パス (Tcko+Troute_total+Tlogic_total) からクロック スキュー (Tdest_clk - Tsrc_clk) とレジスタのホールド遅延 (Th) を引いて、register-to-register パスで実行されます。

ホールド チェックの確認

TWR レポートでは、スラック値でホールド違反を確認できます。

表 6-2 : ホールド違反

スラック	ホールド違反
負	あり
正	なし

ホールド スラックの計算

ホールド スラックの計算には、次の式が使用されます。

$$\text{Hold Slack} = \text{Tdata} - \text{Tskew} - \text{Th}$$

詳細パスのレポート

詳細パスは、そのデータ パスを含む制約の下にレポートされます。パスは、要件に対するスラックごとにリストされます。

ホールド パスの遅延タイプの識別子に -Th があります。この -Th は、ホールド遅延タイプの後に表示され、レース コンディションやホールド違反を識別しやすくします。

ホールド解析

ホールド解析は、すべてのグローバルおよびローカル クロック リソースで実行されます。データ パスが、シリコン全体のプロセス、電圧、温度 (PVT) のさまざまなパターンを表示するために調整されることはありません。

ホールド違反

ホールド違反はまれにしか発生しません。この問題が発生する前には、データ パス遅延が小さすぎ、クロック スキューが大きいはずです。

ホールド違反が発生する場合は、PAR で配線を変更して違反を修正できます。PAR とタイミングツールでは、次を実行されます。

- クロック スキューを削減します。
- 必要な場合は特定データ パスのクロック遅延を増加します。

ホールド スラック

ホールド スラックは制約要件には関連しません。これは、制約のスラックと最小遅延周期 (ns) を確認する場合はわかりにくいかもしれません。

ホールド スラックはクロック スキューとデータ パス遅延の関係に関連しています。最小遅延周期 (ns) に影響するのは、セットアップ パスからのスラックのみです。

既知の外部スキュー

FROM:TO 制約では、次の場合、クロック ソース間の既知の外部スキューが考慮されます。

- エンドポイント レジスタが共通クロックを共有しない場合、または
- クロックが相互に関連しない場合

レジスタ間で 1 つの共通クロック ソースが共有される場合、スキューは同期エレメントまでのクロック パスの特定部分に対してのみ計算されます。共通のクロック ソース ポイントが検出されない場合、スキューは最大クロック パスと最小クロック パス間の違いになります。

パス ヘッダーは、次のようになります。

- クロック スキューがレポートされます。
- ソース クロック ピンとデスティネーション クロック ピンまでの遅延の詳細は含まれません。

これらの遅延を指定するには、Timing Analyzer の [Analyze Against User Specified Paths ... by defining Endpoints] を使用します。

1. クロック パッド入力をソースとして指定します。
2. ホールド / セットアップ解析のレジスタまたは同期エレメントをデスティネーションとして指定します。

パッドから各レジスタのクロック ピンへのクロック遅延はレポートされます。この解析は、DLL/DCM/PLL のクロック パスでも実行されます。

クロック スキューを計算するには、ソース クロック遅延からデスティネーション クロック遅延を引きます。パスは、スラックではなく、パス遅延の合計別にリストされます。

例 1

IDDR から DQ CE ピンまでの DQS パスに約 2 分の 1 サイクルの制約を設定します。これにより、読み出しポストアンプルの終わりに DQS グリッチが IDDR への入力に到着する前に、DQ クロック イネーブルがディアサートされます。この値は、クロック周波数によって異なります。

```
INST */gen_dqs*.u_iob_dqs/u_iddr_dq_ce TNM = TNM_DQ_CE_IDDR;  
INST */gen_dq*.u_iob_dq/gen_stg2*.u_iddr_dq TNM = TNM_DQS_FLOPS;  
TIMESPEC TS_DQ_CE = FROM TNM_DQ_CE_IDDR TO TNM_DQS_FLOPS TS_SYS_CLK * 2;
```

この要件は、システム クロックに基づいています。

例 2

MUX のセレクト ピンからキャプチャされる同期エレメントの次のステージまでのパスに制約を設定します。この値は、クロック周波数によって異なります。

```
NET clk0 TNM = FFS TNM_CLK0;
NET clk90 TNM = FFS TNM_CLK90;
# MUX Select for either rising/falling CLK0 for 2nd stage read capture
INST */u_phy_calib_0/gen_rd_data_sel*.u_ff_rd_data_sel TNM = TNM_RD_DATA_SEL;
TIMESPEC TS_MC_RD_DATA_SEL = FROM TNM_RD_DATA_SEL TO TNM_CLK0 TS_SYS_CLK * 4;
```

この要件は、システム クロックに基づいています。

例 3

IDDR を駆動する DQS ゲートとクロック イネーブル入力間のパスに、その DQS グループの各 DQ キャプチャ IDDR への制約を設定します。この要件は周波数によって異なり、ユーザーは次の要件を設定する必要があります。

```
INST */gen_dqs[*].u_iob_dqs/u_iddr_dq_ce TNM = TNM_DQ_CE_IDDR;
INST */gen_dq[*].u_iob_dq/gen_stg2*.u_iddr_dq TNM = TNM_DQS_FLOPS;
TIMESPEC TS_DQ_CE = FROM TNM_DQ_CE_IDDR TO TNM_DQS_FLOPS 1.60 ns;
```

この要件は、システム クロック 333MHz に基づいています。

OFFSET IN 制約の解析

offsetinanalysis

入力タイミング解析には、次の制約が含まれます。

- OFFSET IN
- FROM:PADS:TO
- OFFSET IN と FROM:PADS:TO の両方

入力タイミング制約は、FPGA の外部ピンまたはパッドからデータをキャプチャする内部の同期エレメントまたはレジスタレジスタまでのデータ パスに適用されます。

このパスに対して従来からある制約は **OFFSET IN** で、次のように指定されます。

- デザインの入力タイミングを指定
- データと、デバイスのピンまたはパッド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

解析されるのは、データをキャプチャする同期エレメントのセットアップ パスとホールド パスです。クロック パスとデータ パスの内部配線および遅延は、タイミング解析ツールの **OFFSET IN** 解析に含まれます。これには、このクロックの周波数と位相変換、クロックのばらつき、I/O 規格およびその他のデータ遅延調整などが含まれます。

ワーストケース パス

タイミング オブジェクトの表 (Timing Object Table) には、選択した制約のワーストケース パスが表示されます。この表には、

次のエレメントを含む共通のタイミング解析詳細が行ごとに表示されます。

- スラック
- データ パス

- クロック パス
- ソース エレメント
- デスティネーション エレメント

制約サマリ

各タイミング レポート制約の詳細には、次を含む制約のサマリが表示されます。

- その制約の適用されるパス数とエンドポイント
- セットアップ エラー
- ホールド エラー

この解析情報を使用すると、次が可能になります。

- 制約が予測通りのパス数やエンドポイントに適用されているかどうか確認できます。
- この制約のパフォーマンスを大きな観点で確認できます。

バス ベースの解析

入力タイミング バスのバス ベースの解析には、1 つの入力クロックに接続された複数のデータ信号を含む入力タイミング インターフェイスが含まれます。このインターフェイスは、バス全体が正しく動作するかどうかには依存します。インターフェイスのバス ベースのタイミング解析とバスの各ビットの解析が含まれます。

バス ベースの解析を実行すると、バスの各ビットに対する詳細な解析が実行され、次が可能になります。

- エラーのよくある原因が判明します。
- バスのパフォーマンスを最適にするためにクロックおよびデータ遅延をどのように変更すればよいかがわかります。

バス ベースの解析のサマリ

バス ベースの解析中は、タイミング レポートのデータシート セクションにバス解析のサマリを含む次のようなセクションが含まれます。

高度なタイミング情報 (High Level Timing Detail)

このセクションには、インターフェイス バスの各ビットに対する高度なタイミング情報が表示されます。これらの詳細は、OFFSET IN 制約の下でのタイミング レポートの詳細セクションに基づいて記述されます。次が含まれます。

- セットアップおよびホールド 要件
- キャプチャ レジスタの各ビットまたはデバイス内の同期エレメントのセットアップ/ホールド スラック

バスの全体的なパフォーマンス (Overall Bus Performance)

このセクションにも、バスの全体的なパフォーマンスの詳細が表示されます。これには、ワースト ケース サマリの行や中心までのソース オフセット (Source Offset to Center) の列などが含まれます。

- [Source Offset to Center]

クロック エッジに対してインターフェイスのデータ ビットを中央にするために必要なデータパス遅延を調整でき、このインターフェイスの最大タイミング差が設定できます。

- [Ideal Clock Offset to Actual Clock]

バスに対してクロック エッジを中央にするために必要なクロック パス遅延を調整できます。このクロック パス遅延は、通常クロック調整ブロック (DCM、PLL、MMCM) を介してクロックを位相シフトすると追加されます。

- [Worst Case Data]

バス インターフェイスの全体的なワーストケースのセットアップ タイム + ホールド タイムの範囲を設定できます。

詳細なパス解析 (Detailed Path Analysis)

タイミング レポートの詳細なパス解析セクションには、入力インターフェイスのクロックとデータパスの詳細が記述されます。この解析には、入力インターフェイスのセットアップおよびホールド解析に必要な遅延すべてが含まれます。

詳細なパス解析セクションの内容

ここでは、詳細なパス解析セクションの内容について説明します。

サマリ ヘッダー

各 OFFSET IN 制約に対して、サマリ ヘッダーが表示され、制約構文、解析されたパスやエンドポイントに関する情報が含まれます。

- 制約構文
- 解析されたパス
- 解析されたエンドポイント

パス ヘッダー

解析されたパスごとに次の情報を含むパス ヘッダーが表示されます。

- スラック値の入力タイミング パスのサマリ
- タイミング チェック ボックスのスラック式
- ソース パッドとデスティネーション ソース エレメントの情報
- キャプチャされるクロック ネット名とクロック エッジ
- クロックとデータ パス遅延の合計
- クロックのばらつき

データおよびクロック パスの詳細 (Data and Clock Path Details)

データおよびクロック パスの詳細セクションには、入力インターフェイスのクロック パスとデータパスの両方に使用されるすべてのコンポーネントおよび配線ネットワーク遅延の詳細が表示されます。

OFFSET IN 制約

OFFSET IN 制約には、次の特徴があります。

- pad-to-setup タイミング要件が定義されます。
- 外部クロックとデータの間関係を指定します。

セットアップ要件

セットアップ要件は次のとおりです。

$(data_delay + setup - clock_delay - clock_arrival)$

セットアップ要件の解析時に、OFFSET IN 制約では次が考慮されます。

- クロック遅延
- クロック エッジ
- DLL/DCM によるクロック位相

クロック到着

クロックの到着では、DLL/DCM またはクロック エッジによって生成されたすべてのクロック位相が考慮されます。タイミング レポートに OFFSET 制約のクロック到着時間が表示されない場合は、タイミング解析ツールでその特定の同期エレメントの PERIOD 制約が解析されなかったことを意味します。

pad-to-setup 要件の作成

pad-to-setup 要件を作成する場合は、すべての位相または PERIOD 制約の変更点を OFFSET IN に設定した値に組み込むようにしてください。

次の例の場合、図 3-3 「CLK の TNM が組み合わせロジックを介して同期エレメント (フリップフロップ) までトレース される例」の回路図を参照してください。

DLL/DCM の CLK90 からのネットがレジスタにクロックを提供する場合、OFFSET 値は PERIOD 制約の 1/4 にする必要があります。

たとえば、PERIOD 制約値が 20ns で DCM の CLK90 から送信される場合、OFFSET IN 値は追加の 5ns にする必要があります。

- 変更前の制約

```
NET "PAD_IN" OFFSET = IN 10 BEFORE "PADCLKIN";
```

- 変更後の制約

```
NET "PAD_IN" OFFSET = IN 15 BEFORE "PADCLKIN"
```

OFFSET 制約に必要なクロック ネット名は、IPAD に適用されたクロック ネット名です。上記の例の場合、クロック パッドは CLK90 ではなく PADCLKIN です。

OFFSET IN BEFORE 制約

OFFSET IN BEFORE は、同期エレメントでデータがパッドからセットアップまで伝搬されるのに使用できる時間を定義します。詳細は、次の図を参照してください。この時間は、データがデバイスのエッジに到着してから次のクロック エッジがデバイスに到着するまでの時間と見なすことができます。

たとえば、「OFFSET = IN 2 ns BEFORE clock_pad」と指定すると、リファレンス クロック エッジがクロック パッドに到着する 2ns 周期前にデータが入力データ パッドで有効になることを示します。ツールでは、内部データとクロック遅延が自動的に計算されて、フリップフロップのセットアップ タイムを満たすように制御されます。

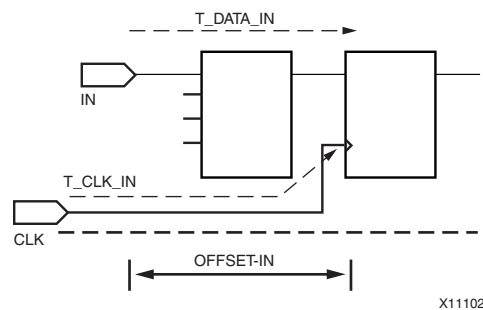


図 6-10 : OFFSET IN BEFORE 制約の計算変数を示す回路図

セットアップ関係の式

次の式は、セットアップ関係を定義しています。

$$T_{Data} + T_{Setup} - T_{Clock} \leq T_{offset_IN_BEFORE}$$

説明

T_{Setup} = フリップフロップのセットアップ タイム
 T_{Clock} = フリップフロップまでのクロック パス遅延の合計
 T_{Data} = フリップフロップまでのデータ パス遅延の合計
 $T_{offset_IN_BEFORE}$ = 全体的なセットアップ要件

OFFSET IN 要件の値

OFFSET IN 要件の値は、セットアップ タイム解析中に FPGA デバイスのセットアップ タイム要件として使用されます。

VALID キーワード

VALID キーワードを使用すると、次が実行されます。

- この要件とキーワードと一緒に使用すると、ホールド タイム解析中のホールド タイム要件を作成できます。
- 入力データの有効範囲の期間を指定します。ホールド タイム解析は、タイミング解析ツールで実行されます。

デフォルトでは VALID 値は OFFSET タイム要件と同じで、ホールド タイム要件は 0 に指定されます (次の図を参照)。

クロック パス遅延が長いほど、外部セットアップ タイムは短くなります。配分されたクロック遅延は、正確なセットアップ タイム解析のために使用されます。通常の配分率は、グローバル配線の場合は 85%、ローカル配線の場合は 80% です。

配分されたクロック パス遅延は、Virtex-II デバイス ファミリよりも古いファミリでは使用されません。

FPGA デバイスの OFFSET IN 解析に含まれる外部ホールドの式は、次のとおりです。

$$\text{外部ホールド} = \text{クロック パス遅延} + \text{フリップフロップのホールド タイム} - \text{データ遅延の配分バージョン}$$

データ遅延がクロック遅延よりも長い場合、ホールド タイムは短くなります。配分されたデータ遅延は、セットアップ解析の配分値と類似しています。

配分されたデータ遅延は、Virtex-II デバイス ファミリよりも古いファミリでは使用されません。

OFFSET IN 制約の単純な例

OFFSET IN 制約の単純な例では、PERIOD 制約に基づいて最初のクロック エッジが 0ns になっています。タイミング レポートには、この最初のクロック エッジがクロック到着時間として表示されます。

タイミング レポートには、次が表示されます。

- データ パス
- クロック パス
- クロック到着時間

タイミング レポートにクロック到着時間が表示されない場合は、タイミング解析ツールでその特定の同期エレメントの PERIOD 制約が認識されなかったことを意味します。

次の図では、OFFSET 要件が最初のクロック エッジの 3ns 前に設定されています。タイミング解析で使用される式は、次のとおりです。

$$\text{Slack} = (\text{Requirement} - (\text{Data Path} - \text{Clock Path} - \text{Clock Arrival}))$$

制約の構文例

```
TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET = IN 3 ns BEFORE clock;
```

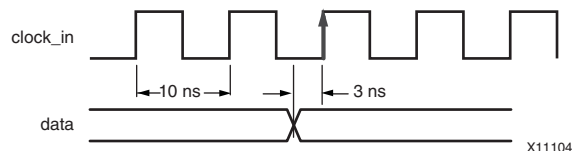


図 6-12：単純な OFFSET IN 制約のタイミング図

タイミング レポートの例

```
Slack:-0.191ns (requirement - (data path - clock path - clock arrival + uncertainty))
Source:                reset (PAD)
Destination:           my_oddrA_ODDR_inst/FF0 (FF)
Destination Clock:     clock0_ddr_bufg rising at 0.000ns
Requirement:           3.000ns
Data Path Delay:        2.784ns (Levels of Logic = 1)
Clock Path Delay:       -0.168ns (Levels of Logic = 3)
Clock Uncertainty:      0.239ns
```

2 位相の例

OFFSET IN 制約の 2 位相 (両クロック エッジ使用) の例では、最初のクロック エッジとクロックの 2 つのエッジが関連しています。

- 1 つ目のクロック エッジは PERIOD 制約に基づいて 0ns
- 2 つ目のクロック エッジは、PERIOD 制約の半分

タイミング レポートには、クロックの各エッジのクロック到着時間が表示されます。

タイミング レポートには、次が表示されます。

- データ パス
- クロック パス
- クロック到着時間

この例の場合、PERIOD 制約には FALLING キーワードに基づいた立ち下がりエッジのクロック到着時間が含まれます。このため、立ち下がりエッジの同期エレメントのクロック到着時間は 0 です。立ち上がりエッジの同期エレメントは PERIOD 制約の半分です。デュアル データ レートのようにどちらのエッジも使用される場合、クロック エッジごとに 1 つずつ、合計 2 つの OFFSET 制約が作成されます。

次の図では、OFFSET 要件が最初のクロック エッジの 3ns 前に設定されています。PERIOD 制約が HIGH に設定され、OFFSET IN 制約が FALLING に設定される場合、次の制約で同じレポート例が生成されます。

```
TIMESPEC TS_clock = PERIOD clock 10 ns HIGH 50%;
OFFSET = IN 3 ns BEFORE clock RISING;
OFFSET = IN 3 ns BEFORE clock FALLING;

TIMESPEC TS_clock=PERIOD clock 10 ns LOW 50%;
OFFSET=IN 3 ns BEFORE clock;
```

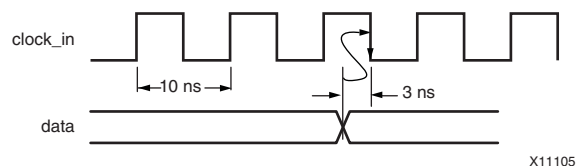


図 6-13 : 2 位相の OFFSET IN 制約のタイミング図

タイミング レポートの例

```
Slack:0.231ns (requirement - (data path - clock path - clock arrival + uncertainty))
Source: DataD<9> (PAD)
Destination: TmpAa_1 (FF)
Destination Clock: clock0_ddr_bufg falling at 0.000ns
Requirement: 3.000ns
Data Path Delay: 2.492ns (Levels of Logic = 2)
Clock Path Delay: -0.038ns (Levels of Logic = 3)
Clock Uncertainty: 0.239ns
```

位相シフトの例

OFFSET IN 制約の例の DCM 位相シフト クロック CLK90 の最初のクロック エッジは、PERIOD 制約に基づいて 0ns になっています。クロックは DCM で位相シフトされているので、タイミング

タイミング レポートには、次が表示されます。

- データ パス
- クロック パス
- クロック到着時間

次の図では、**OFFSET** 要件が最初のクロック エッジの **3ns** 前に設定されています。

```
TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET = IN 3 ns BEFORE clock;
```

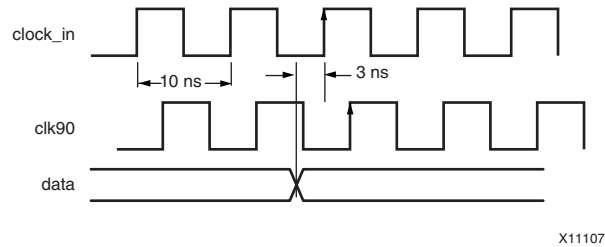


図 6-15 : OFFSET IN 制約の固定位相シフト クロックのタイミング図

タイミング レポートの例

```
Slack:4.731ns (requirement - (data path - clock path - clock arrival + uncertainty))
Source: DataD<9> (PAD)
Destination: TmpAa_1 (FF)
Destination Clock: clock1_fixed_bufg rising at 4.500ns
Requirement: 3.000ns
Data Path Delay: 2.492ns (Levels of Logic = 2)
Clock Path Delay: -0.038ns (Levels of Logic = 3)
Clock Uncertainty: 0.239ns
```

OFFSET IN 制約のデュアル データ レートの例

OFFSET IN 制約のデュアル データ レートの例では、2つのクロック エッジのうちの最初のクロック エッジが **0ns**、PERIOD 制約の半分になっています。タイミング レポートには、クロックの各エッジのクロック到着時間が表示されます。

タイミング解析ツールでは解析中にクロック位相が自動的に調整されないで、各クロック エッジごとに制約を手動で調整する必要があります。

タイミング解析ツールには、立ち下がりエッジのクロック到着時間を管理するために2つのオプションが含まれます。

タイミング レポートには、次が表示されます。

- データ パス
- クロック パス
- クロック到着時間

立ち下がりエッジのクロック到着時間の管理 (オプション 1)

立ち下がりエッジのクロック到着時間を管理する1つ目のオプションでは、

1. 次の2つのタイミンググループが作成されます。
 - 立ち上がりエッジの同期エレメント

Requirement: 3.000ns
Data Path Delay: 2.654ns (Levels of Logic = 2)
Clock Path Delay: -0.006ns (Levels of Logic = 3)
Clock Uncertainty: 0.239ns

クロック立ち下がり前 OFFSET = IN 3 ns のタイミング レポート例

Slack:0.101ns (requirement - (data path - clock path - clock arrival + uncertainty))
Source: DataA<3> (PAD)
Destination: TmpAa_3 (FF)
Destination Clock: clock0_ddr_bufg **falling at 0.000ns**
Requirement: 3.000ns
Data Path Delay: 2.654ns (Levels of Logic = 2)
Clock Path Delay: -0.006ns (Levels of Logic = 3)
Clock Uncertainty: 0.239ns

OFFSET IN AFTER 制約

OFFSET IN AFTER 制約は、FPGA 外部のデータで使用される時間を定義します。

この時間をクロックの **PERIOD** から引くことにより、データがパッドから伝搬されて同期エレメントでセットアップされるまでの最大時間が決定されます。

この時間は、現在のクロック エッジがデバイスに到着してからデータがデバイスに入力されるまでの時間と見なすことができます。

たとえば、「**OFFSET = IN 2 ns AFTER clock_pad**」と指定すると、リファレンス クロック エッジがアップストリーム デバイスに到着した 2ns 周期後に、FPGA デバイスにレジスタ入力されるデータが FPGA の入力パッドで使用可能になることを示します。OFFSET 制約の構文を示すため、チップ間の CLK にスキューはないものと仮定します。

次の式は、この関係を定義しています。

$$T_{Data} + T_{Setup} - T_{Clock} \leq T_{Period} - T_{offset_IN_AFTER}$$

説明

T_{Setup} = フリップフロップのセットアップ タイム
 T_{Clock} = フリップフロップまでのクロック パス遅延の合計
 T_{Data} = フリップフロップまでのデータ パス遅延の合計
 T_{Period} = 1 サイクルの PERIOD 要件
 $T_{offset_IN_AFTER}$ = 全体的なセットアップ要件

OFFSET IN 制約に **AFTER** キーワードを使用する場合は、**PERIOD** または **FREQUENCY** 制約が必要です。

OFFSET OUT 制約の解析

offsetoutanalysis

出力インターフェースの解析は、出力タイミング制約の **OFFSET OUT** に基づいて実行されます。出力タイミング制約は、外部クロック パッドからロジックを通して外部データ パッドに接続された同期エレメントまでのデータ パスに適用されます。この制約では、クロック エッジが外部パッドに到着してから最初のデータがその外部データ パッドに現れるまでの最大時間を定義します。

詳細なパス解析 (Detailed Path Analysis)

タイミング解析には、クロックおよびデータ パスに関連する遅延に影響を与える内部要因が自動的に含まれます。これらの要因には、次が含まれます。

- クロックの周波数と位相変換
- クロックのばらつき
- データ パス遅延調整

バス ベースの解析

タイミング レポートのデータシート セクションには新しい表が作成され、リファレンス ピンに関連する全体的なバス スキューやソース同期インターフェースの最高速のビットについてレポートされます。

バス ベースのタイミング解析

出力タイミング インターフェイスは、通常 1 つの入力クロックに接続された複数のデータ信号で構成されます。バス全体が正しく動作しているかどうかは、インターフェイスのバス ベースのタイミング解析でソース同期デザインのバス全体のワーストケース バス スキューをレポートさせると確認できます。

ビット解析

バス ベースのタイミング解析では、次を含めたバスの各ビットの解析がレポートされます。

- ソース同期エレメント
- パッド エレメント
- 全体遅延

全体遅延には、クロック入力から出力データ ビットまでの遅延が含まれます。

- バス スキュー

バス スキューとは、リファレンス ピンまたは最小データ ビット遅延に関連する各ビットのスキューです。

出力インターフェイスのバス解析の詳細には、出力インターフェイスのクロックおよびデータ パスの解析が含まれます。この解析には、1 つの出力データ パスに対する 1 つのデータ パスの情報が含まれます。

タイミング オブジェクト表

タイミング オブジェクト表 (Timing Object Table) には、Timing Analyzer のバス解析用のタイミング サマリが含まれます。これには、次が含まれます。

- そのバスの出力タイミングとなどが含まれます。
- バスのクロック コンポーネントおよびデータ コンポーネントの使用量

最初の段落の内容

最初の段落には、次の情報を含んだこの 1 つのバスに対するバス サマリが表示されます。

- スラック式を使用して算出されたスラック値での全体的なパフォーマンス サマリ
- ソース同期エレメント
- デスティネーションのパッド エレメント
- 送信クロック ネットワークの詳細
- クロックおよびデータ パス遅延の詳細
- クロックのばらつき値
- クロックのばらつき式

2 段落目の内容

2 つ目の段落には、次の情報を含んだ出力インターフェイスのクロック パスとデータ パスのバス詳細が表示されます。

- 使用されたデバイス リソースすべての詳細
- クロック パスとデータ パスの両方の配線遅延

ヘッダー サマリ セクション

OFFSET 制約のそれぞれの解析には、次に関する情報を含むヘッダ サマリー セクションが表示されます。

- 制約構文
- この制約で解析されるパスとエンドポイントの数
- タイミング エラー

ヘッダー サマリでは、次も確認できます。

- 制約が予測通りパス数やエンドポイントに適用されているかどうか
- この制約の全体的なワーストケース パフォーマンス

OFFSET OUT 制約

OFFSET OUT 制約には、次の特徴があります。

- clock-to-pad タイミング要件が定義されます。
- 外部 clock-to-data を指定します。
- clock-to-out 要件の解析時には次が考慮されます。
 - クロック遅延
 - クロック エッジ
 - DLL/DCM によるクロック位相

$$\text{Clock to Out} = \text{clock_delay} + \text{clock_to_out} + \text{data_delay} + \text{clock_arrival}$$

クロック到着時間

クロックの到着には、DLL/DCM またはクロック エッジによって生成されたすべてのクロック位相が考慮されます。

タイミング レポートにクロック到着時間が表示されない場合は、タイミング解析ツールでその特定の同期エレメントの PERIOD 制約が解析されなかったことを意味します。

clock-to-pad 要件

clock-to-pad 要件を作成する場合は、すべての位相または PERIOD 制約の調整要素を OFFSET OUT に設定した値に組み込むようにしてください。次の例の場合、[図 6-6 「DCM 出力間の CLK90 クロック位相の回路図」](#)を参照してください。

レジスタに PERIOD 制約 20ns の DCM の CLK90 ピンからのネットによりクロックが供給されている場合、OFFSET は元の制約よりも 5ns 少ない値に調整する必要があります。

- 変更前の制約

```
NET "PAD_OUT" OFFSET = OUT 15 AFTER "PADCLKIN";
```
- 変更後の制約

```
NET "PAD_OUT" OFFSET = OUT 10 AFTER "PADCLKIN";
```

OFFSET OUT AFTER 制約

OFFSET OUT AFTER 制約では、データが同期エレメントからパッドへ伝搬されるまでの最大時間を定義します。詳細は、次の図を参照してください。

この時間は、現在のクロック エッジがデバイスに到着してからデータがデバイスから出力されるまでの時間と見なすことができます。

OFFSET OUT AFTER 制約の例

OFFSET = OUT 2 ns AFTER clock_pad

たとえば、「OFFSET = OUT 2 ns AFTER clock_pad」と指定すると、リファレンス クロック パルスが FPGA デバイスに到着した 2ns 周期後に、ダウンストリーム デバイスにレジスタ入力されるデータが FPGA デバイスのデータ出力パッドで使用可能になることを示します

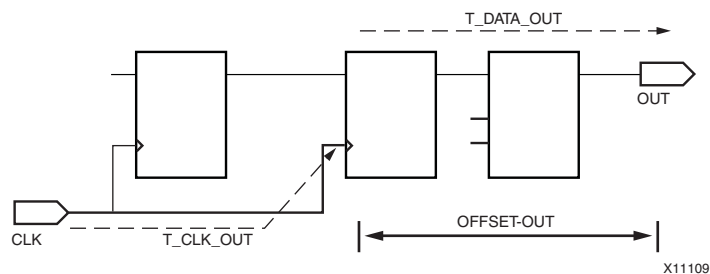


図 6-17 : OFFSET OUT AFTER 制約の計算変数を示す回路図

次の式は、この関係を定義しています。

$$Q + T_{Data2Out} + T_{Clock} \leq T_{offset_OUT_AFTER}$$

説明

T_Q = フリップフロップの clock-to-out
 T_{Clock} = フリップフロップまでのクロック パス遅延の合計
 $T_{Data2Out}$ = フリップフロップまでのデータ パス遅延の合計
 $T_{offset_OUT_AFTER}$ = 全体的な clock-to-out 要件

この制約の解析では、リファレンス パス (CLK_SYS から COMP) の最大遅延およびデータ パス (COMP から Q_OUT) の最大遅延が指定したオフセットを超えていないことを確認します。

OFFSET RISING/FALLING キーワード

OFFSET RISING/FALLING キーワードを使用すると、PERIOD 制約で定義された HIGH または LOW 設定が上書きできます。これは、信号がデータを立ち上がりおよび立ち下がりクロック エッジでキャプチャするか、データを立ち上がりおよび立ち下がりクロック エッジで出力する場合、50% のデューティ サイクルの DDR デザインで使用すると便利です。

HIGH に設定され、OFFSET 制約が FALLING に設定される場合、立ち下がりエッジの同期エレメントのクロック到着時間は 0 に設定されます。

RISING および FALLING に設定された OFFSET OUT 制約の例

```
TIMEGRP DATA_OUT OFFSET = OUT 10 AFTER CLK FALLING;
TIMEGRP DATA_OUT OFFSET = OUT 10 AFTER CLK RISING;
```

OFFSET OUT 制約の単純な例

OFFSET OUT 制約の単純な例では、PERIOD 制約に基づいて最初のクロック エッジが 0ns になっています。タイミング レポートには、この最初のクロック エッジがクロック到着時間として表示されます。

タイミング レポートには、次が表示されます。

- データ パス
- クロック パス
- クロック到着時間

タイミング レポートにクロック到着時間が表示されない場合は、タイミング解析ツールでその特定の同期エレメントの PERIOD 制約が認識されなかったことを意味します。

次の図では、OFFSET 要件は 3ns に設定されています。タイミング解析で使用される式は、次のとおりです。

$$\text{Slack} = (\text{Requirement} - (\text{Clock Arrival} + \text{Clock Path} + \text{Data Path}))$$

```
TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET = OUT 3 ns AFTER clock;
```

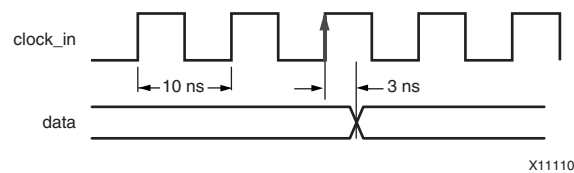


図 6-18 : 単純な OFFSET OUT 制約のタイミング図

タイミング レポートの例

```
Slack:-0 ns (requirement - (clock arrival + clock path + data path + uncertainty))
Source: OutD_7 (FF)
Destination: OutD<7> (PAD)
Source Clock: clock0_ddr_bufg rising at 0.000ns
Requirement: 3.000ns
Data Path Delay: 3.405ns (Levels of Logic = 1)
Clock Path Delay: 0.280ns (Levels of Logic = 3)
Clock Uncertainty: 0.180ns
```

2 位相の例

OFFSET OUT 制約の 2 位相 (両クロック エッジ使用) の例では、最初のクロック エッジが 2 つのクロック エッジに関連します。

- 1 つ目のクロック エッジは PERIOD 制約に基づいて 0ns
- 2 つ目のクロック エッジは、PERIOD 制約の半分

タイミング レポートには、クロックの各エッジのクロック到着時間が表示されます。この例では、PERIOD LOW 制約のクロックは立ち下がりエッジで到着します。このため、立ち下がりエッジの同期エレメントのクロック到着時間は 0 です。立ち上がりエッジの同期エレメントは PERIOD 制約の半分です。

タイミング レポートには、次が表示されます。

- データ パス
- クロック パス
- クロック到着時間

次の図では、OFFSET 要件は 3ns に設定されています。

```
TIMESPEC TS_clock=PERIOD clock 10 ns LOW 50%;
OFFSET = IN 3 ns AFTER clock;
```

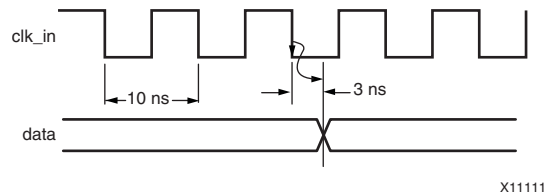


図 6-19 : OFFSET OUT 制約の 2 位相のタイミング図

タイミング レポートの例

```
Slack:-0 ns (requirement - (clock arrival + clock path + data path + uncertainty))
Source: OutD_7 (FF)
Destination: OutD<7> (PAD)
Source Clock: clock3_std_bufg falling at 0.000ns
Requirement: .3.000ns
Data Path Delay: 3.405ns (Levels of Logic = 1)
Clock Path Delay: 0.280ns (Levels of Logic = 3)
Clock Uncertainty: 0.180ns
```

位相シフトの例

OFFSET OUT 制約の例の DCM 位相シフト クロック CLK90 は、PERIOD 制約に基づいて最初のクロック エッジが 0ns になっています。クロックは DCM で位相シフトされているので、タイミング レポートにはクロック到着時間が位相シフト量として表示されます。CLK90 出力が使用される場合、位相シフトの量は PERIOD の 1/4 になります。クロック到着時間は位相シフト量と対応しています。この場合は、2.5ns です。

タイミング レポートには、次が表示されます。

- データ パス
- クロック パス
- クロック到着時間

次の図では、OFFSET 要件は 5ns に設定されています。

```
TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET = OUT 5 ns AFTER clock;
```

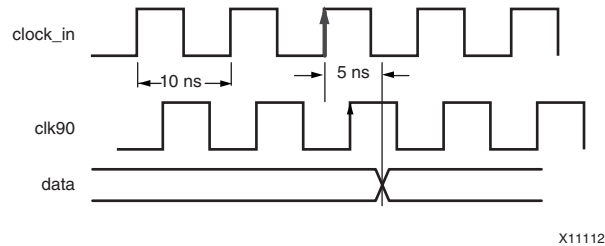


図 6-20 : OFFSET OUT 制約の位相シフト クロックのタイミング図

タイミング レポートの例

```
Slack:-1.365ns (requirement - (clock arrival + clock path + data path + uncertainty))
Source: OutD_7 (FF)
Destination: OutD<7> (PAD)
Source Clock: clock3_std_bufg rising at 2.500ns
Requirement: 5.000ns
Data Path Delay: 3.405ns (Levels of Logic = 1)
Clock Path Delay: 0.280ns (Levels of Logic = 3)
Clock Uncertainty: 0.180ns
```

固定位相シフトの例

OFFSET OUT 制約の固定位相シフトの例では、PERIOD 制約に基づいて最初のクロック エッジが 0ns になっています。

- クロックは DCM で位相シフトされているので、タイミング レポートにはクロック到着時間が位相シフト量として表示されます。
- CLK0 出力がユーザーの指定した量で位相シフトされる場合、位相シフトの量は PERIOD の % になります。

この例の場合、

- PERIOD 制約には立ち上がりエッジで最初のクロックが到着するように設定されています。
- クロック到着値は固定位相シフトの量になります。

クロック到着時間は位相シフト量と対応しています。

タイミング レポートには、次が表示されます。

- データ パス
- クロック パス
- クロック到着時間

次の図では、OFFSET 要件は 5ns に設定されています。

```
TIMESPEC TS_clock=PERIOD clock_grp 10 ns HIGH 50%;
OFFSET = OUT 5 ns AFTER clock;
```

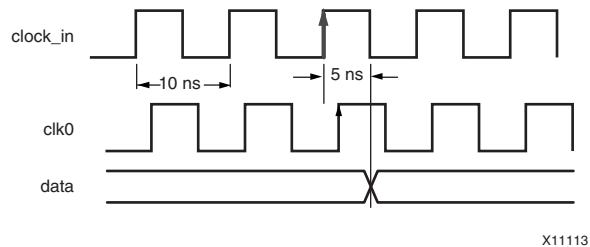


図 6-21：OFFSET OUT 制約の固定位相シフト クロックのタイミング図

タイミング レポートの例

Slack:0.535ns (requirement - (clock arrival + clock path + data path + uncertainty))

Source:	OutD_7 (FF)
Destination:	OutD<7> (PAD)
Source Clock:	clock3_std_bufg rising at 0.600ns
Requirement:	5.000ns
Data Path Delay:	3.405ns (Levels of Logic = 1)
Clock Path Delay:	0.280ns (Levels of Logic = 3)
Clock Uncertainty:	0.180ns

OFFSET OUT 制約のデュアル データ レートの例

OFFSET OUT 制約のデュアル データ レートの例では、2つのクロック エッジのうちの最初のクロック エッジが 0ns、PERIOD 制約の半分になっています。タイミング レポートには、クロックの各エッジのクロック到着時間が表示されます。

タイミング解析ツールでは解析中にクロック位相が自動的に調整されないで、各クロック エッジごとに制約を手動で調整する必要があります。

タイミング解析ツールには、立ち下がりエッジのクロック到着時間を管理するために 2つのオプションが含まれます。

タイミング レポートには、次が表示されます。

- データ パス
- クロック パス
- クロック到着時間

立ち下がりエッジのクロック到着時間の管理 (オプション 1)

立ち下がりエッジのクロック到着時間を管理する 1つ目のオプションでは、

1. 次の 2つのタイミンググループが作成されます。
 - 立ち上がりエッジの同期エレメント
 - 立ち下がりエッジの同期エレメント
2. 各タイムグループに OFFSET IN 制約が作成されます。

2つ目の OFFSET IN 制約の要件は異なります。

立ち下がりエッジの OFFSET IN 制約の要件は、

元の要件から PERIOD 制約の半分の引いたものと同じになります。

たとえば、元の要件が 3ns (PERIOD 制約 10ns) の場合、立ち下がり OFFSET IN 制約の要件は -2ns になります。

これにより、立ち下がりエッジの同期エレメントに関連してクロック到着時間が補正されます。この制約には、負の値を設定できます。

立ち下がりエッジのクロック到着時間の管理 (オプション 2)

立ち下がりエッジのクロック到着時間を管理する 2 つ目のオプションでは、

1. タイムグループ 1 つを作成し、対応する **OFFSET IN** 制約に各クロック エッジの元の制約要件を付けます。
2. 立ち下がりエッジの場合は **FALLING**、立ち上がりエッジの場合は **RISING** キーワードを追加します。

次の図では、**OFFSET** 要件は 3ns に設定されています。

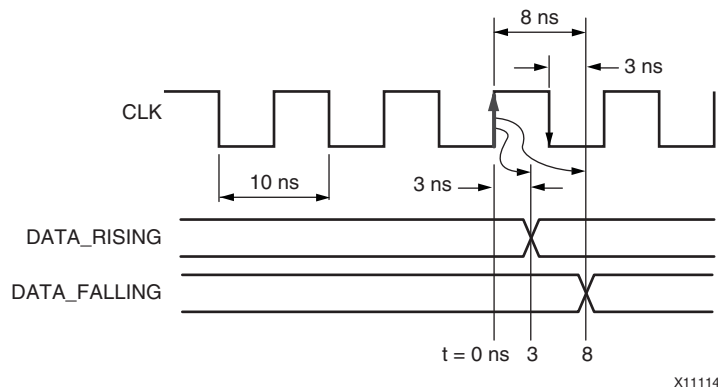


図 6-22 : OFFSET OUT 制約のデュアル データ レートのタイミング図

クロック立ち上がり後 OFFSET = OUT 3 のタイミング レポート例

```
Slack:-0.783ns (requirement - (clock arrival + clock path + data path + uncertainty))
Source:          OutA_4 (FF)
Destination:     OutA<4> (PAD)
Source Clock:     clock0_ddr_bufg rising at 0.000ns
Requirement:     3.000ns
Data Path Delay:  3.372ns (Levels of Logic = 1)
Clock Path Delay: 0.172ns (Levels of Logic = 3)
Clock Uncertainty: 0.239ns
```

クロック立ち下がり後 OFFSET = OUT 8 のタイミング レポート例

```
Slack:-0.783ns (requirement - (clock arrival + clock path + data path + uncertainty))
Source:          OutA_4 (FF)
Destination:     OutA<4> (PAD)
Source Clock:     clock0_ddr_bufg falling at 0.000ns
Requirement:     3.000ns
Data Path Delay:  3.372ns (Levels of Logic = 1)
Clock Path Delay: 0.172ns (Levels of Logic = 3)
Clock Uncertainty: 0.239ns
```

OFFSET OUT BEFORE 制約

OFFSET OUT BEFORE 制約は、FPGA 外部のデータで使用される時間を定義します。

この時間をクロックの **PERIOD** から引くことにより、データが同期エレメントからパッドまで伝搬されるまでの最大時間を決定します。

この時間は、現在のクロック エッジがデバイスに到着する前にデータがデバイスのエッジから出力されるまでの時間と見なすことができます。

たとえば、「**OFFSET = OUT 2 ns BEFORE clock_pad**」と指定すると、クロック パルスが出ダウンストリーム デバイスに到着する何周期分か前に、ダウンストリーム デバイスにレジスタ入力されるデータが **FPGA** の出力パッドで使用可能になることを示します。**OFFSET** 制約の構文を示すため、チップ間の **CLK** にスキューはないものと仮定します。

次の式は、この関係を定義しています。

$$TQ + TData2Out + TClock \leq TPeriod - Toffset_OUT_BEFORE$$

説明

TQ = フリップフロップの clock-to-out
TClock = フリップフロップまでのクロック パス遅延の合計
TData2Out = フリップフロップまでのデータ パス遅延の合計
TPeriod = 1 サイクルの **PERIOD** 要件
Toffset_OUT_BEFORE = 全体的な clock-to-out 要件

この制約の解析では、リファレンス パス (**CLK_SYS** から **COMP**) の最大遅延およびデータ パス (**COMP** から **Q_OUT**) の最大遅延が指定したオフセットを超えていないことを確認します。

OFFSET OUT 制約に **BEFORE** キーワードを使用する場合は、**PERIOD** または **FREQUENCY** 制約が必要です。

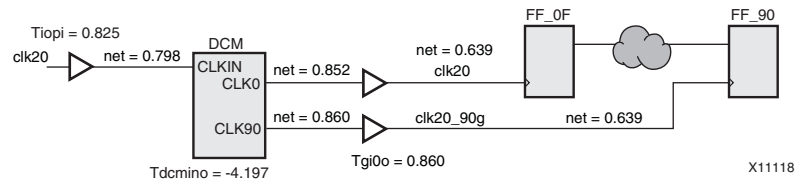


図 6-26 : クロック スキューの例

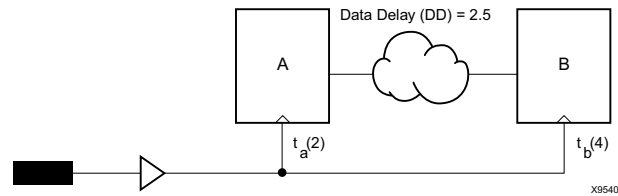


図 6-27 : MAXSKEW の例

この図の詳細は次のとおりです。

- **ta(2)** の場合、レジスタ **A** クロックの最大遅延は 2ns です。
- **ta(4)** の場合、レジスタ **B** クロックの最大遅延は 4ns です。
- **MAXSKEW** は、最大値 **tb** から最大値 **ta** 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すぎる値を設定すると、**PAR** のランタイムが長くなります。

クロックのばらつき

PERIOD 制約要件の差には、「クロック スキュー」だけでなく、クロックのばらつきも影響します。クロックのばらつきは、システム、ボード レベル、および DCM クロック ジッターによるタイミング精度を増加するために使用します。

次の図のように、PERIOD 制約に SYSTEM_JITTER 制約および INPUT_JITTER キーワードを使用すると、デザインのタイミングに影響する外部ジッターが含まれていることをタイミング解析ツールに伝えることができます。

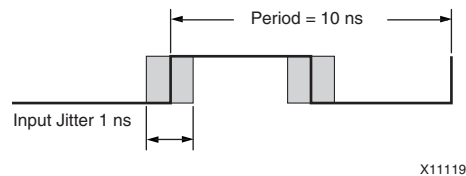


図 6-28：クロック信号の入カジッター

Virtex-4 以降のデバイス ファミリの解析には、クロックのばらつきだけでなく、次 も含まれます。

- DCM ジッター
- DCM 位相エラー
- DCM デューティ サイクルの歪み/ジッター

クロックのばらつきに含まれる個別コンポーネントについては、レポートされます。タイミング解析ツールでは、データ パスのソースとデスティネーションのクロックのばらつきが計算され、それらを合わせたクロックのばらつきの合計が算出されます。

DCM のクロックのばらつき式

次は、DCM のクロックのばらつきの式です。

$$\text{Clock Uncertainty} = [\sqrt{(\text{INPUT_JITTER}^2 + \text{SYSTEM_JITTER}^2)} + \text{DCM_Discrete_Jitter}] / 2 + \text{DCM_Phase_Error}$$

Virtex-4 以降のデバイスの場合、DCM 離散ジッターと DCM 位相エラーは、スピード ファイルに含まれます。これらは speedprint には含まれません。

クロックのばらつきの例

- INPUT_JITTER: $200\text{ps}^2 = 40000\text{ps}$
- SYSTEM_JITTER: $150\text{ps}^2 = 22500\text{ps}$
- DCM Discrete Jitter: 120ps
- DCM Phase Error: 0ps
- Clock Uncertainty: 185ps

INPUT_JITTER キーワードを使用した PERIOD 制約の例

次は、INPUT_JITTER キーワードを使用した PERIOD 制約の例です。

```
TIMESPEC "TS_Clk0" = PERIOD "clk0" 4 ns HIGH 60% INPUT_JITTER 200 ps PRIORITY 1;
```

SYSTEM_JITTER (システム ジッター) 制約には、次の特徴があります。

- システムに影響するジッターを定義
- ジッターは、次から表すことができます。
 - 電源ノイズ
 - ボード ノイズ
 - システム全体のその他のジッター

SYSTEM_JITTER 値は、次のようなデザイン条件によって異なります。

- 同時に変更する同期エレメントの数
- 同時に変更する入力および出力の数

SYSTEM_JITTER 値は、(1) 入力クロック エッジのノイズ (またはジッター) と (2) 電源ノイズ、の違いに基づいて指定できます。この違いは、(1) クロック エッジと (2) 電源プレーンおよびグランド プレーンの動き、の違いによってボードで測定できます。

ユーザー指定の SYSTEM_JITTER 制約は、該当するデバイス ファミリのデフォルトの SYSTEM_JITTER 値がある場合はそれを上書きします。

すべてのデバイス ファミリがデフォルトの SYSTEM_JITTER 値を持つわけではありません。この場合、ユーザーが値を指定する必要があります。

ザイリンクスでは、SYSTEM_JITTER 値に 300ps を使用することをお勧めしています。この値は、次のように処理されます。

- デザインのクロックすべてに適用されます。
- 該当するクロック ネットワーク トポロジの INPUT_JITTER 値と組み合わせられます。

UCF の SYSTEM_JITTER 制約の例

次は、UCF の SYSTEM_JITTER 制約の例です。

```
SYSTEM_JITTER = 300 ps;
```

クロック ジッターには、ランダムおよび離散ジッタ コンポーネントの両方が含まれます。INPUT_JITTER および SYSTEM_JITTER はランダム ジッター ソースで、通常はガウス分布になるので、この 2 つは二次方程式で追加されて、ワーストケースの組み合わせが示されます。

DCM ジッターは離散ジッター値なので、クロックのばらつきに直接追加されます。

クロックのばらつきの解析では、ランダムと離散の両方のジッタ コンポーネントすべてが peak-to-peak (最大振幅) 値として指定されます。peak-to-peak 値は +/- 範囲の合計で示され、クロック信号の到着時間はジッタの有無によって異なります。

ワーストケース解析では、タイミング スラックで離散の原因となる遅延変化のみが使用されます。このため、ピーク ジッタ値または peak-to-peak 値の半分のみが各セットアップおよびホールド タイミング チェックに使用されます。

クロックのばらつきの位相エラー コンポーネントは、2つのクロック信号間の位相変化を示します。これは離散値で、DCM クロック間の実際の位相差を示すので、クロックのばらつきの値に直接追加できます。

PLL のクロックのばらつき式

$$\text{Clock Uncertainty} = [\sqrt{(\text{INPUT_JITTER}^2 + \text{SYSTEM_JITTER}^2 + \text{PLL_Discrete_Jitter}^2)}] / 2 + \text{PLL Phase_Error}$$

Virtex-5 以降のデバイスの場合、PLL 離散ジッターと PLL 位相エラーは、スピード ファイルに含まれます。

クロックのばらつきの解析では、ランダムと離散の両方のジッター コンポーネントすべてが **peak-to-peak** (最大振幅) 値として指定されます。**peak-to-peak** 値は +/- 範囲の合計で示され、クロック信号の到着時間はジッターの有無によって異なります。ワーストケース解析では、タイミング スラックで離散の原因となる遅延変化のみが使用されます。

ピーク ジッター値または **peak-to-peak** 値の半分のみが各セットアップおよびホールド タイミング チェックに使用されます。

クロックのばらつきの位相エラー コンポーネントは、2 つのクロック信号間の位相変化を示します。これは離散値で、PLL クロック間の実際の位相差を示すので、クロックのばらつきの値に直接追加できます。

タイミング クロージャの達成

タイミング クロージャは、近年の主なデザイン課題の 1 つです。

- タイミング クロージャを達成しにくい原因は、デザインのパフォーマンス要件が高すぎるのに、ターゲット デバイスの容量が足りないことがあるからです。
- 以前は ASIC デバイスにフィットしていたデザインや、ASIC デバイスの高いクロック周波数で実行されていたデザインが、ザイリンクス FPGA デバイスに実装できるようになったことにより、この問題はさらに複雑化しています。

このため、パフォーマンス目標を達成するための確実な方法が必要となります。この章では、具体例を使用して推奨される方法を示すことで、タイミング クロージャ問題について説明します。

この章のガイド ラインは、パフォーマンスを改善してタイミング目標を達成するための手引きとして使用してください。

タイミング クロージャの達成

タイミング クロージャは、次のすべての動作状況が有効である場合に、デザインのタイミング制約がすべて満たされると達成されます。

- プロセス
- 電圧
- 温度

タイミング スコア

タイミング クロージャは、デザインに完全に制約が付いていて、タイミング スコアが 0 になると達成されます。タイミング スコアには、次の特徴があります。

- すべての制約のタイミング解析を示す総合的な値で、エラーになった制約の量を示します。
- 満たされなかったタイミング制約すべての合計 (ピコ秒) になります。
- すべてのタイミング制約に対するエラーの合計をピコ秒で示します。
- 次のように配線アルゴリズムの段階ごとに PAR レポートに表示されます。

- 『SmartXplorer for Command Line Users』 (UG688)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_3/ug688.pdf
- 『SmartXplorer for Project Navigator Users』 (UG689)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_3/ug689.pdf

PlanAhead の資料

- 『フロアプラン手法ガイド』 (UG633)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_4/Floorplanning_Methodolgy_Guide.pdf
- 『PlanAhead ユーザー ガイド』 (UG632)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx13_4/PlanAhead_UserGuide.pdf