

ザイリンクス FPGA および SoC 用 UltraFast 設計手法 ガイド

UG949 (v2020.2) 2021 年 2 月 18 日

この資料は表記のバージョンの英語版を翻訳したもので、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。

改訂履歴

次の表に、この文書の改訂履歴を示します。

セクション	改訂内容
2021 年 2 月 18 日 バージョン 2020.2	
タイトル	『ザイリンクス FPGA および SoC 用 UltraFast 設計手法ガイド』に変更。
同期リセット vs 非同期リセット	セクションをアップデート。
自動パイプライン処理に関する注意事項	セクションをアップデート。
Vitis 環境の制約の指定	新しいセクションを追加。
ブロック デザインの合成	グローバル合成モードとアウト オブ コンテキスト合成モードに関する情報を追加。
インクリメンタル合成	アウト オブ コンテキスト合成モードからインクリメンタル合成への変換に関する情報を追加。
配線ログでの密集のレポート	情報メッセージの例をアップデート。
UltraScale および UltraScale+ デバイスでのクロック遅延の削減	MAX_PROG_DELAY プロパティに関する情報およびクロック使用率レポートでの行プログラム可能タップ遅延設定の確認に関する情報を追加。
レジスタの複製の使用	MAX_FANOUT_MODE に関する情報を追加。
密集の解消	report_qor_suggestions の実行に関する情報を追加。
配置結果の再利用	コード例をアップデート。
推奨される消費電力制約	新しいセクションを追加。
正確な消費電力解析のためのベスト プラクティス	report_power の信頼性レベルに関する情報を追加。
配線後の ECO を使用したネットの外部ピンへの接続	新しいセクションを追加。
2020 年 8 月 14 日 バージョン 2020.1	
第 3 章: RTL を使用したデザインの作成	個別の章に移動。
モジュール レベルの属性の適用	合成属性伝搬規則へのリンクを追加。
CLOCK_DEDICATED_ROUTE 制約の使用	SAME_CMT_COLUMN の例をアップデート。
第 4 章: デザイン制約	個別の章に移動。
合成フロー	新しいセクションを追加。
合成最適化	新しいセクションを追加。
合成後の QoR の評価	新しいセクションを追加。
タイミング違反の解析および解決	QoR 推奨項目レポートに関する情報を追加。
MMCM 設定を使用したクロックのばらつき削減	推奨事項をアップデートし、式を追加。
タイミング レポートを使用して消費電力最適化の影響を確認	Tcl の例をアップデート。

目次

改訂履歴.....	2
第 1 章: 概要.....	5
UltraFast 設計手法について.....	5
UltraFast 設計手法のコンセプト.....	8
Vivado Design Suite の使用.....	12
その他の資料およびトレーニングへのアクセス.....	12
第 2 章: ボードおよびデバイス プランニング.....	14
PCB レイアウトに関する推薦事項.....	14
デバイスの消費電力およびシステムの依存性.....	19
クロック リソースのプランニングおよび割り当て.....	21
I/O プランニング デザイン フロー.....	22
SSI デバイスでの設計.....	26
HBM デバイスでの設計.....	32
コンフィギュレーション.....	36
第 3 章: RTL を使用したデザインの作成.....	38
適切なデザイン階層の定義.....	38
IP (Intellectual Property) の使用.....	41
RTL コーディング ガイドライン.....	44
クロッキング ガイドライン.....	77
クロック乗せ換え.....	123
第 4 章: デザイン制約.....	128
デザイン制約の分類.....	128
タイミング制約定義の 4 つの段階.....	132
クロック制約の定義.....	134
入力ポートおよび出力ポートの制約.....	141
クロック グループおよび CDC 制約の定義.....	149
タイミング例外の指定.....	154
マルチサイクル パス制約の追加.....	158
その他のアドバンス タイミング制約.....	160
物理制約の定義.....	161
第 5 章: デザイン インプリメンテーション.....	162
合成の実行.....	162
合成後の処理.....	169
デザインのインプリメンテーション.....	173

第 6 章: デザイン クロージャ	180
タイミング クロージャ	180
消費電力クロージャ	253
コンフィギュレーションおよびデバッグ	257
付録 A: その他のリソースおよび法的通知	267
ザイリンクス リソース	267
ソリューション センター	267
Documentation Navigator およびデザイン ハブ	267
参考資料	268
トレーニング リソース	270
お読みください: 重要な法的通知	271

概要

UltraFast 設計手法について

ザイリンクス UltraFast™ 設計手法は、デバイスの設計プロセスを効率的にするためのベスト プラクティスをまとめたものです。これらのデザインは大型で複雑性が高いため、各設計段階を正しく実行していくためには、特定の手順と設計タスクが必要です。これらの手順およびベスト プラクティスに従うと、デザイン目標を短期間で効率的に達成できます。

- この資料: さまざまなデザイン タスク、解析とレポート機能、デザインの作成とクロージャのベスト プラクティスを説明します。
- 『UltraFast 設計手法クイック リファレンス ガイド』 (UG1231): 主要な設計手法手順を使いやすい両面印刷カード用の形式にまとめた早見表です。
- 『UltraFast 設計手法タイミング クロージャクイック リファレンス ガイド』 (UG1292): 初期デザイン チェック、デザインのベースライン制約の作成、タイミング違反の解決など、タイミング クロージャに関する推奨事項を示します。
- 『UltraFast 設計手法チェックリスト』 (XTP301): ザイリンクス Documentation Navigator から使用可能なほか、スタンドアロンのスプレッドシートとしても提供されています。ただし、ザイリンクス Documentation Navigator からは日本語版のチェックリストは使用できません。日本語版のスプレッドシートをご利用ください。このチェックリストを使用すると、設計プロセス中のよくあるミスや決定事項を特定できます。
- UltraFast 設計手法のシステム レベルのデザイン フロー図: Vivado® Design Suite デザイン フロー全体を示す図で、ザイリンクス Documentation Navigator から使用できます。図の設計段階をクリックすると、それに関連する資料、ファイル、FAQ などを開くことができます。



推奨: ザイリンクスでは、これらのリソースに加え、エンベデッド デザインを設計する場合は『UltraFast エンベデッド デザイン設計手法ガイド』 (UG1046: [英語版](#)、[日本語版](#))、Vivado IP インテグレーターの C ベース IP を使用する複雑なシステムを設計する場合は『UltraFast Vivado HLS 設計手法ガイド』 (UG1197: [英語版](#)、[日本語版](#)) を参照することをお勧めします。

ザイリンクスが提供する次のリソースを使用すると、UltraFast 設計手法の利点を活かすことができます。



ヒント: ザイリンクスでは、Vivado Design Suite の `report_methodology Td` コマンドで使用可能な、各デザイン段階の設計手法関連のデザイン ルール チェック (DRC) も提供しています。

このユーザー ガイドについて

このガイドでは、システム統合およびデザイン インプリメンテーションの両方で生産性を最大限にするためのベスト プラクティスを示します。次のトピックの概要、設計ガイドライン、デザインの決定事項のトレードオフを示します。

- **第 2 章: ボードおよびデバイス プランニング:** デザイン作成前に達成しておくことをザイリンクスが推奨する決定事項とデザイン タスクについて説明します。I/O およびクロック プランニング、PCB レイアウトの考慮事項、デバイス容量、スループット評価、代替デバイスの定義、消費電力の見積もり、デバッグなどが含まれます。

- **第 3 章: RTL を使用したデザインの作成:** RTL の定義、IP の設定と管理、制約の割り当てなどのベスト プラクティスを説明します。
- **第 4 章: デザイン制約:** 適切なタイミング、消費電力、および物理制約を作成するための推奨事項と、合成およびインプリメンテーションで使用される追加の制約、属性、およびその他の要素を指定するための推奨事項を示します。
- **第 5 章: デザイン インプリメンテーション:** デザインを合成およびインプリメントするために使用可能なオプションとベスト プラクティスを説明します。
- **第 6 章: デザイン クロージャ:** デザインのタイミング クロージャや消費電力の削減に使用するさまざまなデザイン解析およびインプリメンテーション手法について説明します。ハードウェア検証用のデバッグ ロジックを追加する際の注意事項についても説明します。

このガイドには、Vivado Design Suite ユーザー ガイド、Vivado Design Suite チュートリアル、および QuickTake ビデオ チュートリアルなどのその他の資料へのリファレンスも含まれます。このガイドはそれらの資料に置き換わるものではありません。サイリンクスでは、ツールの使用方法および設計手法を含む詳細および最新情報は、これらの資料を参照することをお勧めします。

この情報は Vivado Design Suite 用に記述されていますが、ほとんどの概念的な情報は ISE® Design Suite にも応用できます。

関連情報

[その他のリソースおよび法的通知](#)

UltraFast 設計手法チェックリストの使用

UltraFast 設計手法を最大限に活用するため、このガイドを『UltraFast 設計手法チェックリスト』(XTP301)と合わせて使用してください。チェックリストは、サイリンクス Documentation Navigator から使用可能なほか、スタンドアロンのスプレッドシートとしても提供されています。

UltraFast 設計手法チェックリストの質問は、後の段階に悪影響を及ぼす可能性のあるデザインの決定事項に関するもので、見逃されたり、無視されてしまいがちな問題が含まれます。チェックリストの各タブには、次のような特徴があります。

- 典型的なデザイン チーム内の特定の役割をターゲットにしています。
- 各デザイン フロー段階(プロジェクト プランニング、ボードおよびデバイス プランニング、IP およびサブモジュールの作成、最上位デザインのクロージャ)における、よくある質問とで推奨される操作が含まれます。
- デザイン フロー段階に関連するリソースをリストする資料およびトレーニング セクションが含まれます。
- この資料の内容やその他のサイリンクス資料へのリンクが含まれます。これらの参考資料は、質問に関連したデザインの考慮事項に対処するためのガイダンスとしてご使用ください。



ビデオ: チェックリストの詳細は、[Vivado Design Suite QuickTake ビデオ: UltraFast 設計手法チェックリストの説明](#)を参照してください。

UltraFast 設計手法 DRC の使用

Vivado Design Suite には、`report_methodology Tcl` コマンドを使用して実行可能な設計手法関連の DRC が複数含まれます。このコマンドは、次の各設計段階で実行できます。

- エラボレート済み RTL デザインで合成前に実行して、RTL のコンストラクトを検証。
- 合成後に実行して、ネットリストおよび制約を検証。

- インプリメンテーション後に実行して、制約およびタイミング関連の問題を検証。



推奨: 各設計段階で設計手法 DRC を実行し、次の段階に進む前に問題を解決しておくのが最も効果的です。

設計手法 DRC の詳細は、『Vivado Design Suite Tcl コマンド リファレンス ガイド』 (UG835) の [report_methodology](#) Tcl コマンドを参照してください。

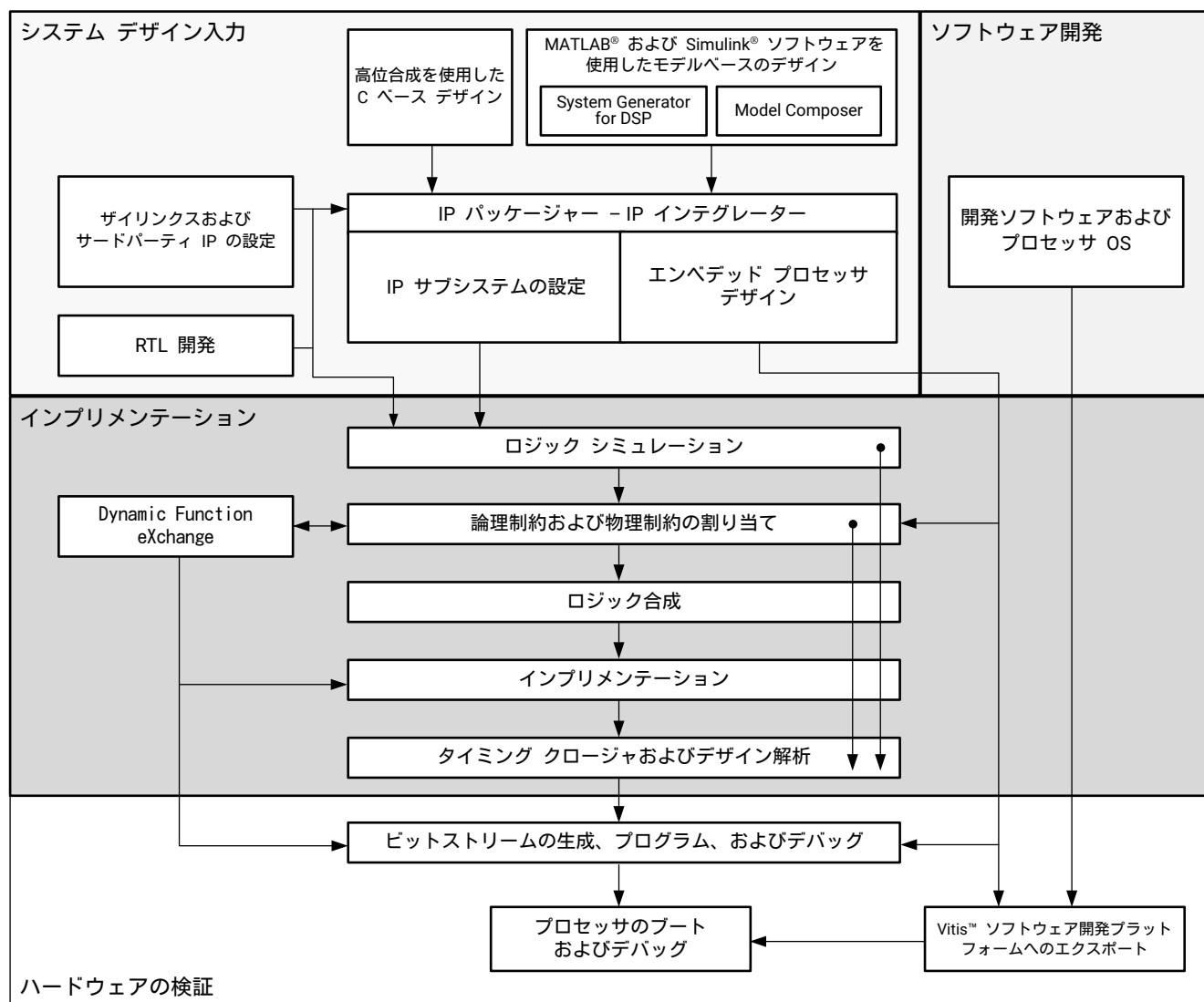
関連情報

[設計手法レポートの生成](#)

UltraFast 設計手法のシステム レベルのデザイン フロー図の使用

次の図に、Vivado Design Suite に含まれるさまざまな設計段階と機能を示します。サイリンクス Documentation Navigator の [Design Hub View] タブから、この図のインタラクティブなバージョンにアクセスでき、各段階のリンクをクリックすることにより関連リソースを参照できます。

図 1: UltraFast 設計手法のシステム レベルのデザイン フロー



X15150-120319

UltraFast 設計手法のコンセプト

最初から正しい手法を使用し、初期段階から RTL、クロック、ピン、および PCB プランニングなどのデザイン目標に注意することが重要です。各設計段階でデザインを正しく定義および検証すると、その後のインプリメンテーション段階でタイミング クロージャ、配線クロージャ、および消費電力の問題を軽減できます。

ハードウェア デザインの作成およびインプリメンテーション

デバイスの I/O プランニングを実行し、PCB のレイアウトをプランニングし、Vivado® Design Suite の使用モデルを決定したら、デザインの作成を開始できます。デザインの作成には、次が含まれます。

- デザイン階層をプランニング
- デザイン内で使用およびカスタマイズする IP コアを特定
- IP カタログにない特別なインターコネクトまたは機能のために必要な RTL モジュールをインスタンスエート
- タイミング制約、消費電力制約、および物理制約を作成
- 合成およびインプリメンテーションで使用される追加の制約、属性、およびその他の要素を指定

デザインを作成する際の主な考慮点は、次のとおりです。

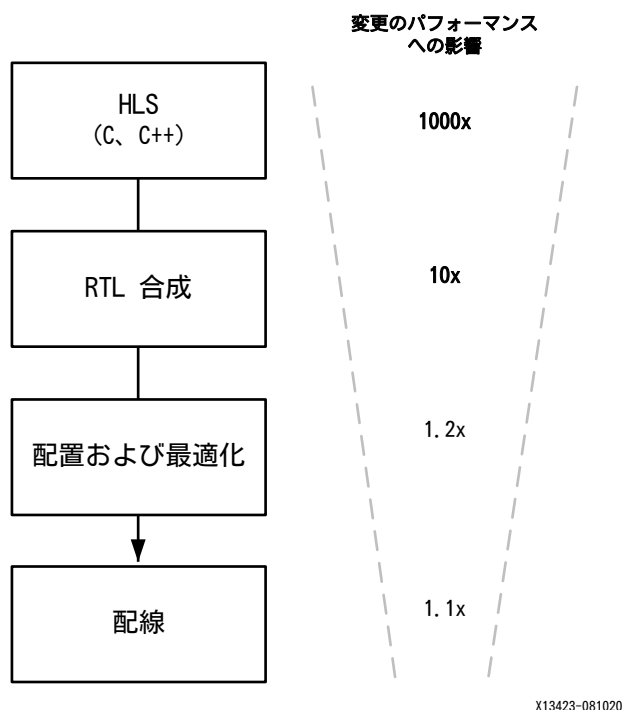
- 必要な機能を達成する
- 必要な周波数で動作する
- 必要な信頼度で動作する
- シリコン リソースおよび消費電力を要件内に収める

この段階での決定事項が、最終的な製品に影響します。この段階で不適切な決定を下すと、後の段階で問題となり、デザイン サイクル全体で問題が発生する可能性があります。プロセスの初期段階で時間をかけてデザインを注意深くプランニングすると、デザイン要件を満たし、ラボでのデバッグ時間を最小限に抑えることができます。

開発サイクルの初期段階の影響

次の図に示すように、デザイン フローの初期段階 (C、C++、RTL 合成) の方が、後のインプリメンテーション段階よりもデザイン パフォーマンス、集積度、消費電力への影響が大きくなります。このため、デザインのタイミング目標が満たされない場合は、サイリンクスではインプリメンテーション段階のみを繰り返してソリューションを見つけようとするのではなく、合成段階 (HDL および制約を含む) を見直すことをお勧めします。

図 2: フローの各段階におけるデザイン変更の影響



各設計段階での検証

UltraFast 設計手法では、初期段階からデザイン バジレット (エリア、消費電力、タイミングなど) を監視してデザインを修正することが重要です。

- ザイリンクス テンプレートを使用して最適な RTL コンストラクトを作成し、エラボレーション後、合成前に設計手法 DRC を使用して RTL を検証します。

Vivado ツールではフロー全体でタイミング ドリブン アルゴリズムが使用されるので、デザインにはデザイン フローの最初から制約を正しく設定する必要があります。

- 合成後にタイミング解析を実行します。

正しいタイミングを指定するには、まず各マスター クロックとそれらに関連する生成クロック間の関係を解析する必要があります。Vivado ツールでは、非同期またはフォルス パスとして明確に定義されていない限り、すべてのクロック間のタイミングが解析されます。

- 次の設計段階に進む前に正しい制約を使用してタイミングが満たされるようにします。

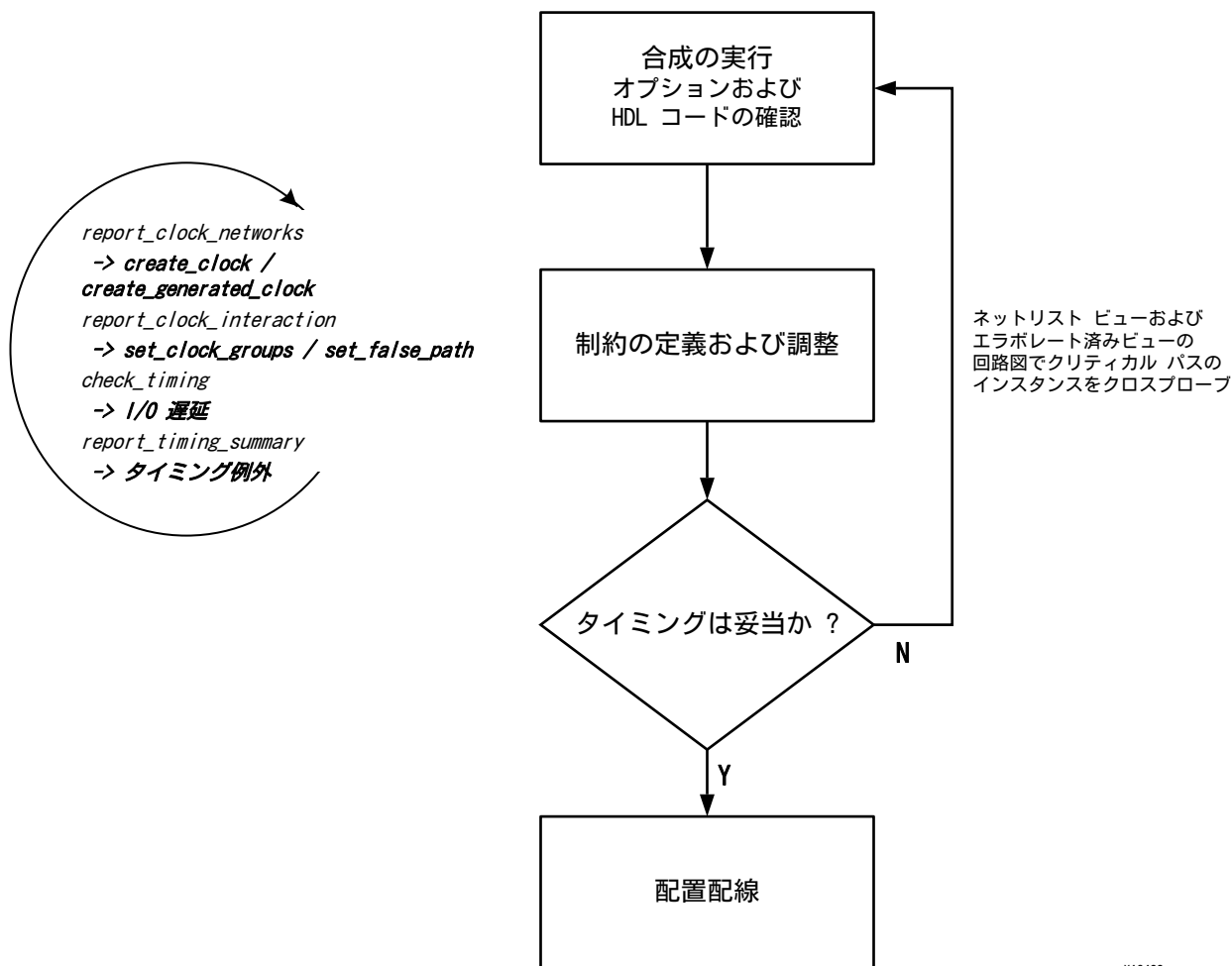
この推奨事項に従って Vivado Design Suite のインタラクティブな解析環境を使用することにより、全体的なタイミングおよびインプリメンテーションの収束期間が短縮されます。



ヒント: これらの推奨事項とこのガイドの HDL 設計ガイドラインを合わせて使用することで、設計期間をさらに短縮できます。

次の図に、この推奨される設計手法を示します。

図 3: デザインを短期間で収束するための RTL 設計手法



X13422

合成段階は、デザイン目標が正のマージン (または比較的小さい負のタイミング マージン) で満たされれば、完了したと考えることができます。合成後のタイミングが満たされない場合、配置配線結果でタイミングが満たされない可能性が高くなります。ただし、タイミングが満たされない場合でも、残りのフローを実行することは可能です。インプリメンテーション ツールでエラーのあったパスに最適なリソースが割り当てられ、タイミングが満たされる可能性があります。また、フローを進めていくと、負のスラックの大きさをより正確に理解できるようになり、合成後の WNS (ワースト ネガティブ スラック) をどれくらい改善する必要があるのかを判断しやすくなります。これらの情報は、HDL および制約を改善するために合成に戻ったときに使用できます。

短時間検証

この資料では、次のようなシステム アーキテクチャおよびマイクロ アーキテクチャの選択に関する短時間検証の概念も紹介しています。

- システム デザインでは、デザイン全体をインプリメントする前に、I/O 帯域幅がシステムで検証されます。I/O 帯域幅の検証により、I/O を最終決定する前に、システム アーキテクチャとインターフェイスの選択を見直す必要があるかがわかります。

- デザイン インプリメンテーションでは、最も単純な制約セット (ベースライン制約) を作成し、内部デバイス タイミングの課題を特定します。ベースライン制約の作成は、インプリメンテーション段階に進む前に、RTL のマイクログラフィクスを変更する必要があるかどうかを特定するプロセスです。

関連情報

[インターフェイスの帯域幅の検証](#)
[デザインのベースライン制約の作成](#)

Vivado Design Suite の使用

Vivado Design Suite には柔軟な使用モデルがあり、さまざまな開発フローやデザイン タイプに対応できます。Vivado Design Suite でのこれらの機能の使用方法については、『Vivado Design Suite ユーザー ガイド: デザイン フローの概要』(UG892) およびその他の Vivado Design Suite 資料を参照してください。

リビジョン管理システムを使用した Vivado Design Suite ソースの管理

ほとんどの設計チームがデザイン ソースと結果を市販のリビジョン管理システムを使用して管理しています。Vivado Design Suite では、デザインと IP データを管理するためのさまざまな使用モデルに対応しています。Vivado ツールでリビジョン管理システムを使用する方法については、『Vivado Design Suite ユーザー ガイド: デザイン フローの概要』(UG892) の[このセクション](#)を参照してください。

Vivado Design Suite の新規リリースへのアップグレード

Vivado Design Suite の新規バージョンには、サイリンクス IP のアップデートが含まれることがほとんどです。IP をアップグレードをするとデザインが変更されるので、アップグレードするかは十分に考慮して決める必要があります。前のリリースで設計した IP を今後も引き続き使用する場合は、特定の規則に従う必要があります。詳細は、『Vivado Design Suite ユーザー ガイド: IP を使用した設計』(UG896) の[このセクション](#)を参照してください。

その他の資料およびトレーニングへのアクセス


この資料は、ユーザー ガイド、リファレンス ガイド、チュートリアル、および QuickTake ビデオなどの Vivado Design Suite 資料の情報を補足するものです。サイリンクス Documentation Navigator (DocNav) では、Vivado Design Suite の資料、ビデオ、サポート リソースにアクセスでき、特定の情報を取得するためにフィルター機能や検索機能を利用できます。サイリンクス Documentation Navigator (DocNav) を開くには、次のいずれかを実行します。

- Vivado IDE で [Help] → [Documentation and Tutorials] をクリックします。
- Windows で [Start] → [All Programs] → [Xilinx Design Tools] → [DocNav] をクリックします。
- Linux コマンド プロンプトに「docnav」と入力します。

サイリンクス デザイン ハブには、資料やビデオへのリンクがデザイン タスクおよびトピックごとにまとめられており、これらを参照することでキー コンセプトを学び、よくある質問 (FAQ) を参考に問題を解決できます。デザイン ハブにアクセスするには、次のいずれかを実行します。

- ザイリンクス Documentation Navigator (DocNav) で [Design Hubs View] タブをクリックします。
- ザイリンクス ウェブサイトで[デザイン ハブ](#) ページを参照します。



ヒント: Vivado IDE の各ダイアログ ボックスにある [Quick Help] ボタン  をクリックすると、そのダイアログ ボックスに関する情報にアクセスできます。Tcl コマンドの詳細を取得するには、[Tcl Console] ウィンドウでそのコマンドに `-help` を付けて入力します。

ボードおよびデバイス プランニング

ボード上にデバイスを正しく配置し、信号を特定のピンに割り当てると、全体的なシステム パフォーマンス、消費電力、熱パフォーマンス、およびデザイン サイクル時間を大幅に改善できます。デバイスとプリント回路基板 (PCB) 間の物理的および論理的な通信を把握することにより、デバイス上でのデータフローを効率的なものにすることができます。

I/O コンフィギュレーションが適切にプランニングされていないと、システム パフォーマンスが低下し、デザイン クロージャに時間がかかります。ザイリンクスでは、I/O プランニングをボード プランニングと共に考慮することを強くお勧めします。

詳細は、次を参照してください。

- 『Vivado Design Suite ユーザー ガイド: I/O およびクロック プランニング』 (UG899)
- [Vivado Design Suite QuickTake ビデオ: I/O プランニングの概要](#)

PCB レイアウトに関する推薦事項

ボード上のデバイスとそれと通信するコンポーネントのレイアウトは、I/O プランニングに大きく影響します。

PCB 上の物理コンポーネントに対する配置

まず、PCB 上のデバイスの配置を決定する必要があります。固定されている PCB コンポーネントおよび内部デバイス リソース両方の位置を考慮します。たとえば、デバイス パッケージ上の GT インターフェイスを、それと通信する PCB 上のコンポーネントの近くに配置すると、PCB トレース長が短くなり、PCB ビア数を削減できます。

重要なインターフェイスを含む PCB の図を使用すると、PCB 上でのデバイスの向きおよび PCB コンポーネントの配置を決定するのに役立ちます。配置を決定したら、デバイスの I/O インターフェイスをプランニングできます。

メモリなどの高速インターフェイスは、それと通信する PCB コンポーネントと短距離で直接接続できると有益です。これらの PCB トレースは通常一致した長さにする必要があります。可能な限り PCB ビアを使用しないようにします。この場合、接続を短くし、BGA ピンの大きなマトリックス内から配線を取り出すのを回避するため、デバイスのエッジに最も近いパッケージ ピンが適しています。

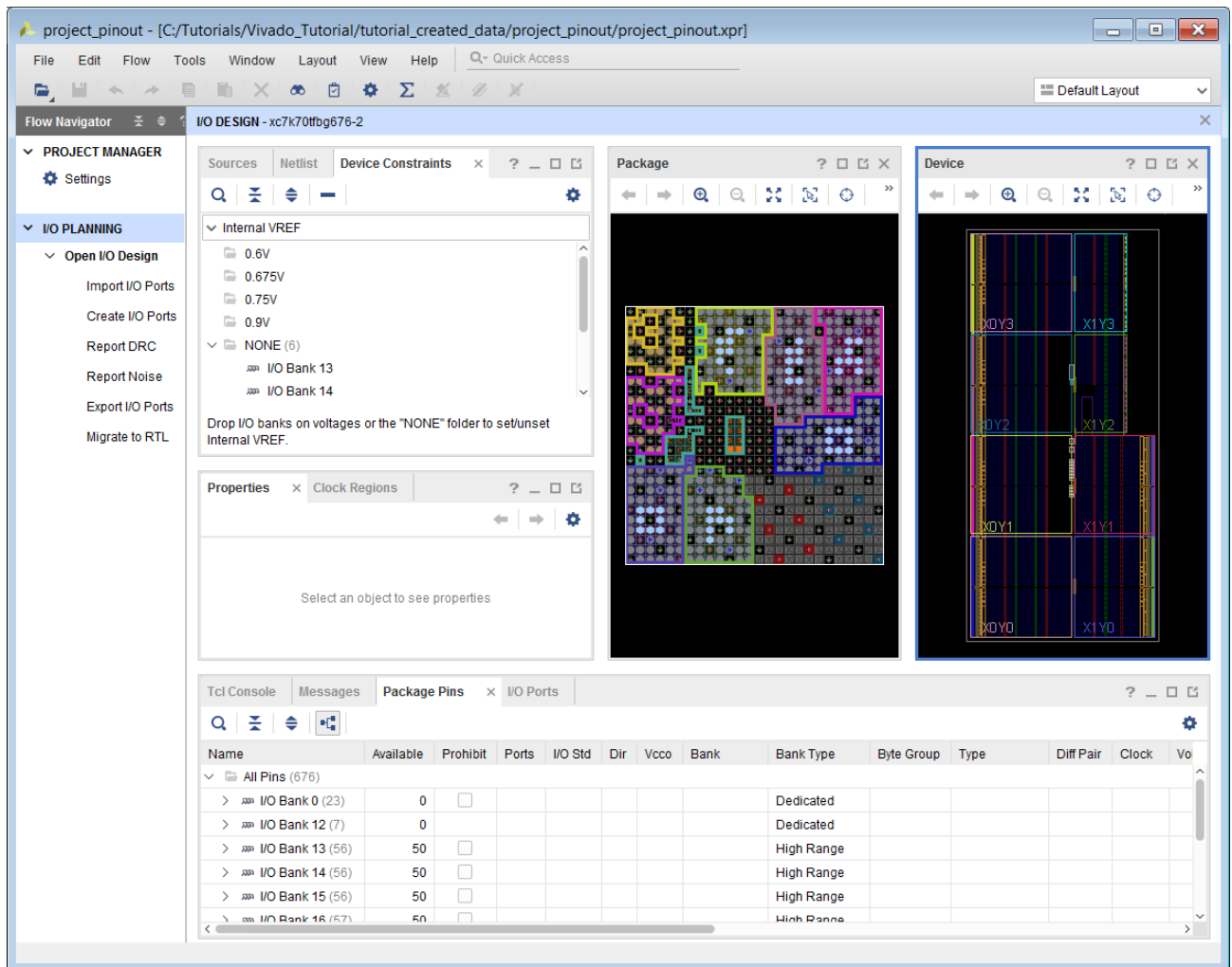
この段階で Vivado® IDE の [I/O Planning] レイアウトを使用すると、デバイスの上面と下面の両方が表示され、物理的なデバイスの寸法に対して I/O 接続を視覚化できるので便利です。



熱ヒント: 熱が問題となるデザインでは、ほかの消費電力が大きいコンポーネントに対するデバイスの配置に注意して、熱結合を最小限に抑え、エアフローが最大限になるようにしてください。デバイスを消費電力が大きい別のコンポーネントの排気口に配置したり、ボードの温度上昇が動作温度に悪影響を与えるような場所に配置しないようにしてください。ザイリンクスでは、熱シミュレーションを実行し、デバイスの配置および環境条件がデバイスのジャンクション温度にどのように影響するかを調べることをお勧めします。

次の図に、[I/O Planning] レイアウトを示します。

図 4: [I/O Planning] レイアウト



電源分配システム

ザイリンクス デバイスの電源分配システム (PDS) を設計する際は、独特のタスクが必要となります。大型マイクロプロセッサなど、ほかの大型で集積度の高い回路のほとんどには、特定のバイパス キャパシタ要件があります。これらのデバイスはハード化されたシリコン アーキテクチャで特定のタスクをインプリメントするために設計されているので、それらの電源要件は固定しており、変動は通常ある一定の範囲内です。

ザイリンクス デバイスには、このような特性はありません。デバイスには、複数のクロック ドメインを使用したユーザー指定の周波数で動作するアプリケーションをほぼ無制限にインプリメントできます。

そのため、デザインの消費電力要件を理解しておくことが重要です。デザインの消費電力要件は、Xilinx Power Estimator (XPE) で消費電力を見積もることにより評価できます。また、該当するデバイスの PCB デザイン ガイドを参照して、消費電力見積もりの前に PDS の配置および一般的なデカップリング要件を理解しておくようにしてください。

PDS の設計では、主に次の事項を考慮する必要があります。

- 消費電力見積もりに基づき、ノイズおよび電流の要件を満たすために適切な電圧レギュレータを選択します。

注記: 電源デザインを可能にし、単純化するため、ザイリンクス パートナーと主要な電源ベンダーにより、すべての消費電力要件を満たすリファレンス デザインが設計、ビルド、テストされています。詳細は、ザイリンクス ウェブサイトの[消費電力削減](#)ページの [電力管理ソリューション] タブを参照してください。

- 電源を統合します。UltraScale™ デバイスでの統合オプションの詳細は、『UltraScale アーキテクチャ PCB デザイン ユーザー ガイド』 (UG583: [英語版](#)、[日本語版](#)) の[このセクション](#)を参照してください。



電源/消費電力ヒント: ザイリンクスでは、各レールの電源を監視するため、分路抵抗を追加することをお勧めします。または、PMBus がイネーブルのレギュレータまたは電流監視集積回路 (IC) を使用できます。

- XADC 電源 (Vrefp および Vrefn ピン) を設定します。
- 電源分配ネットワーク (PDN) シミュレーションを実行します。

UltraScale デバイスでは、『UltraScale アーキテクチャ PCB デザイン ユーザー ガイド』 (UG583: [英語版](#)、[日本語版](#)) にリストされている推奨数のデカップリング キャパシタを使用してください。これは、このガイドの前提に基づいています。デザインの前提がこのガイドのものと異なる場合は、デザインのシミュレーションを実行して、必要なデカップリングの数を判断してください。PDN シミュレーションを実行すると、電源を確実に推奨される動作範囲内にするために必要なデカップリング キャパシタの適切な数を確認するのに役立ちます。

注記: ご使用のデバイスの詳細は、『7 シリーズ FPGA PCB デザイン ガイド』 (UG483: [英語版](#)、[日本語版](#))、『UltraScale アーキテクチャ PCB デザイン ユーザー ガイド』 (UG583: [英語版](#)、[日本語版](#))、または『Zynq-7000 SoC PCB デザイン ガイド』 (UG933: [英語版](#)、[日本語版](#)) を参照してください。

PDN シミュレーションの詳細は、『S パラメーター モデルを使用した FPGA パワー インテグリティのシミュレーション』 ([WP411](#)) を参照してください。



電源/消費電力ヒント: ザイリンクスでは、SIMetrix/SIMPLIS で SIMPLIS シミュレータを使用して電源デザインをシミュレーションし、デザインの動作条件がザイリンクスの推奨する範囲内であることを確認することをお勧めします。ほとんどの電源ベンダーは、このシミュレーションを実行するのに必要な限定バージョンの SIMPLIS とモデルを提供しています。SIMPLIS は、電圧レギュレータの過渡および AC 解析に使用されるサードパーティ ソフトウェアです。



電源/消費電力ヒント: 各レールに必要な電流が使用する電源供給システムの制限を超えないようにするため、Vivado ツールの `report_power` コマンドを使用して、レギュレータまたは電圧レギュレータ モジュール (VRM) ごとに消費電力を解析できます。

関連情報

[消費電力クロージャ](#)

熱ソリューションに関する考慮事項

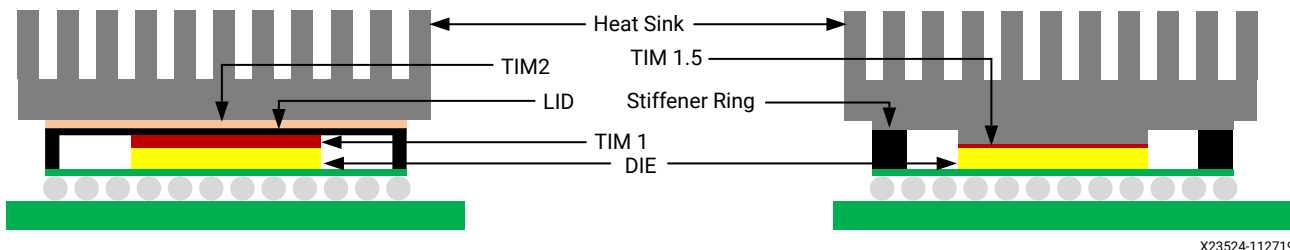
デザインの消費電力見積もりでは、熱ソリューションの効率を理解することが重要です。ジャンクション温度が低いほど、デザインのスタティック消費電力が低くなります。

ザイリンクスでは、ご使用のデバイスにリッドレス パッケージがある場合は、それを使用することをお勧めします。リッドレス パッケージでは、より効率的な熱ソリューションが提供され、熱源に直接接触できるので熱伝導材料 (TIM) 層を削除できます。ザイリンクスのリッド付きパーツとリッドレス パーツの取り扱いおよび製造要件は同じです。次の図に、リッド付きデバイスおよびリッドレス デバイスのヒートシンクの取り付けを示します。



熱ヒント: ザイリンクスでは、ヒートシンクは最小ボンド ラインの厚さ (BLT) を確実にする 20 ~ 50 重量ボン ド毎平方インチ (PSI) をお勧めします。また、リッド付きおよびリッドレス デバイスの両方で、4 つの穴を使用 して取り付け、圧力が均等になるようにしてください。リッドレス パッケージでのガイドラインは、『リッ ドレス フリップチップ パッケージの機械/熱設計ガイドライン』(XAPP1301: [英語版](#)、[日本語版](#)) を参照してく ださい。

図 5: ヒートシンクの例



X23524-112719

ザイリンクスでは、適当なマージンがあることを確認し、正確な消費電力見積もりを得るため、熱シミュレーション を実行することもお勧めします。Xilinx Power Estimator (XPE) では、次の熱設定を指定できます。

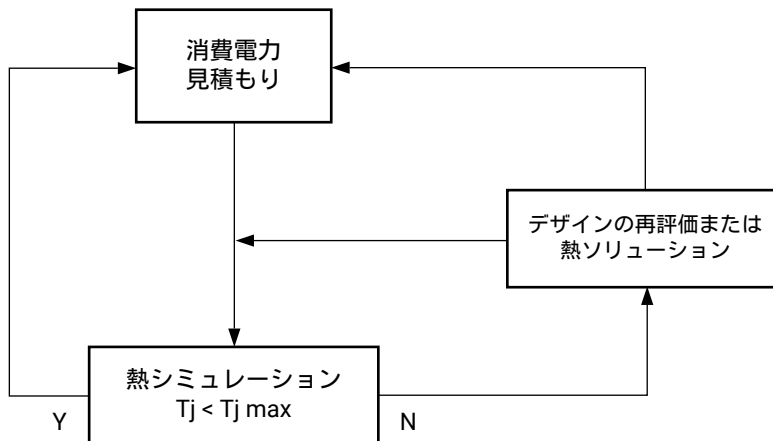
- [Junction Temperature] (ジャンクション温度 T_j): この設定を熱シミュレーションに一致するジャンクション温度 に変更できます。熱シミュレーションを実行しない場合は、ジャンクション温度をワースト ケースに設定します。
- [Effective Θ_{JA}]: 熱ソリューションの熱効率をワット毎セルシウス度 ($^{\circ}\text{C}/\text{W}$) で示します。たとえば、 Θ_{JA} が $2.1^{\circ}\text{C}/\text{W}$ の場合、デバイスで 1 ワット消費されるたびにジャンクション温度が 2.1°C 上昇することを意味します。 10W デザインでは、周囲温度からの温度上昇は 21°C です。

注記: Θ_{JA} は、熱シミュレーションから次の式を使用して取得できます。

$$\Theta_{JA} = (T_j - T_a) / \text{消費電力}$$

次の図に、熱検証に推奨されるフローを示します。

図 6: 熱検証に推奨されるフロー



X23525-111319

ジャンクション温度が仕様の範囲内で、十分なマージンが考慮されていれば、熱ソリューションは有効であるとみなすことができます。



熱ヒント: 消費電力見積もりおよび熱シミュレーションの結果を Vivado デザイン制約に追加します。次の XDC 制約を使用できます。これらの制約は、『Xilinx Power Estimator ユーザー ガイド』(UG440) に説明されているように、XPE のエクスポート機能を使用してエクスポートできます。

```
# Standard Constraints:
set_operating_conditions -process Maximum
set_operating_conditions -design_power_budget <value>
#If thermal simulation completed
set_operating_conditions -ambient_temp <value>
set_operating_conditions -thetaja <value>
#Else if no thermal simulation completed
set_operating_conditions -junction_temp <value>
```

PCB の設計に関する考慮事項

PCB は、デバイスとの信号インターフェイスが最速となるように設計する必要があります。高速信号は、トレースの形状、ビア、損失、およびクロストークに大きく影響されます。これは、特に多層 PCB で顕著です。高速インターフェイスに対しては、シグナル インテグリティ シミュレーションを実行します。必要なパフォーマンスを得るため、よりよい PCB 材料を使用したり、トレースの形状を変更するなど、ボードの再設計が必要な場合もあります。

ザイリンクスでは、PCB を設計する際は次の手順に従うことをお勧めします。

1. 次のデバイス資料を確認します。
 - 該当するデバイスの PCB デザイン ガイド。
 - 該当するデバイスのトランシーバー ユーザー ガイドのボード設計ガイドライン。
2. IP の製品ガイドでメモリ IP および PCIe® デザインのガイドラインを確認します。
3. Vivado ツールを使用して I/O プランニングを検証します。
 - 同時スイッチ ノイズ (SSN) 解析を実行します。
 - ビルトイン DRC を実行します。
 - IBIS (I/O Buffer Information Specification) モデルをエクスポートします。
4. 次のようにシグナル インテグリティ 解析を実行します。
 - ギガビット トランシーバー (GT) に対し、チャネル パラメーターを使用して SPICE または IBIS-AMI シミュレーションを実行します。
 - パフォーマンスの低いインターフェイスに対し、IBIS シミュレーションを実行してオーバーシュートやアンダーシュートの問題がないかどうかをチェックします。
5. XPE で [Process] を [Maximum] に設定し、デザインの消費電力の初期見積もりを生成します。
6. 終了してデバイスの回路図チェックリストに従います。

注記: 『7 シリーズ回路図レビュー推奨事項』(XMP277)、『Kintex UltraScale および Virtex UltraScale FPGA 回路図レビュー チェックリスト』(XTP344)、または『UltraScale+ FPGA および Zynq UltraScale+ MPSoC 回路図レビュー チェックリスト』(XTP427) を参照してください。
7. XPE を使用してザイリンクス デザイン制約 (XDC) ファイルを生成し、このファイルに対応する Vivado プロジェクトにインポートします。XPE 環境設定は、XDC 制約に変換されます。合計オンチップ消費電力の見積もりが、Vivado 消費電力解析で使用するデザインの消費電力バジェットになります。詳細は、『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』(UG907) を参照してください。

関連情報

[その他のザイリンクス資料](#)

デバイスの消費電力およびシステムの依存性

PCB をプランニングする際は、消費電力を考慮する必要があります。

- デバイスとユーザー デザインにより、システム電源と放熱の要件が決定します。
- デバイスの動作中、電源が最大消費電力要件を満たすようにし、推奨される電圧および温度動作条件内に収まるようにする必要があります。デバイスがこれらの制限内に収まるようにするため、消費電力の見積もりと熱モデルが必要です。
- 電源レールを統合することをプランニングし、それが電源ドメイン切り替えにどのように影響するかを把握するようにします。
- 統合は可能ですが、柔軟性を最大限にするため、ザイリンクスでは可能な限り完全なパワー マネージメントを使用することをお勧めします。

これらの理由から、デバイスの消費電力要件および冷却要件を理解し、ボード設計に組み込む必要があります。



電源/消費電力ヒント: ザイリンクス パートナーおよびザイリンクスが承認している電源供給リファレンス デザインのリストは、ザイリンクス ウェブサイトの[消費電力削減](#)ページを参照してください。

デバイスの電源パス

デバイスに電源を供給するには複数の電源が必要であり、特定の順序で投入する必要があります。デバイスおよびその他のアクティブ コンポーネントに正しい順序で電源を投入するため、電源モニターまたはシーケンス回路を使用することを考慮してください。より複雑な環境では、マイクロコントローラーを使用するか、MBUS や PMBUS などのパワー マネージメント バスを使用して、電源とリセット プロセスを制御すると有益な場合があります。電源のオン/オフ シーケンスの詳細は、デバイスのデータシートを参照してください。電源の統合およびトポロジについては、ご使用のデバイスによって『7 シリーズ FPGA PCB デザイン ガイド』(UG483: [英語版](#)、[日本語版](#))、『UltraScale アーキテクチャ PCB デザイン ユーザー ガイド』(UG583: [英語版](#)、[日本語版](#))、または『Zynq-7000 SoC PCB デザイン ガイド』(UG933: [英語版](#)、[日本語版](#)) を参照してください。

異なるデバイス リソースに、個別の電源から必要な電力が供給されます。これにより、さまざまなリソースを異なる電圧レベルで動作させることができるので、ノイズや寄生効果に対して高い耐性を保ちながら、パフォーマンスおよび信号強度を向上できます。

電力タイプ

デバイスは、電源を投入してから切るまでの間に、電力要件の異なるいくつかの電力フェーズを経過します。

電源投入

電源投入電力は、デバイスに最初に電力を投入したときに発生する過渡スパイク電流です。この電流は、各電圧電源で異なり、デバイスの構造、電源ソースが公称電圧に上昇する能力、温度や電源シーケンスなどのデバイスの動作条件に依存します。

現代のデバイス アーキテクチャでは、電源投入シーケンスのガイドラインに従えば、スパイク電流は発生しません。

スタートアップ電力

スタートアップ電力は、デバイスの初期の立ち上げおよびコンフィギュレーションに必要な電力です。この電力は通常短期間に発生するので放熱を考慮する必要はありませんが、電流要件は満たす必要があります。ほとんどの場合、動作中のデザインのアクティブ電流の方が高いので、変更は必要ありません。ただし、アクティブ電流が低い低消費電力デザインでは、この期間電流要件を高くすることが必要な場合があります。XPE を使用して、この要件を理解できます。[Process] を [Maximum] に設定した場合、各電圧レールの電流要件が動作電流かスタートアップ電流の高い方に指定されます。スタートアップ電流の方が高い場合、XPE に電流値が青で表示されます。

スタティック消費電力

デザインのスタティック消費電力 (スタンバイ電力とも呼ばれる) は、デバイスがデザインでコンフィギュレーションされたときに供給される電力で、外部から適用されるアクティビティや内部で生成されるアクティビティはありません。スタティック消費電力は、デザインの動作中に供給する必要がある最小継続電力を示します。

スタティック消費電力は、ジャンクション温度の関数です。そのため、周囲温度および熱ソリューション パラメータを正しくモデリングすることが、消費電力見積もりツールでスタティック消費電力が正しくレポートされるようにするために重要です。

関連情報

[推奨される消費電力制約](#)

ダイナミック消費電力

ダイナミック消費電力は、デバイスでアプリケーションが実行され、スイッチング アクティビティ (クロックとデータパスが High と Low の間で遷移) が発生しているときに必要な電力です。ダイナミック消費電力は、一定期間内のデバイス回路の平均スイッチング アクティビティに基づいて計算されます。総消費電力には、スタティック消費電力とダイナミック消費電力の両方が含まれます。

消費電力に影響する環境要因

デザイン自体だけでなく、環境要因も消費電力に影響します。これらの要因は、デバイスの電圧およびジャンクション温度に影響するので、消費電力に影響します。詳細は、『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』 (UG907) の[このセクション](#)を参照してください。



熱ヒント: -2LI および -2LE UltraScale+™ デバイスでは、定義された時間内であれば 110°C までの温度逸脱が許容されるので、熱ソリューションのコストを削減できます。詳細は、『逸脱温度を利用した熱ソリューションの拡張』 (WP517: [英語版](#)、[日本語版](#)) を参照してください。

電源レール統合の消費電力への影響

消費電力管理のため電源ドメインの切り替えを活用するには、一部の電源レールを個別にしておく必要があります。そうすると、電源ドメイン切り替えロジックを使用してレールの電源を個別に切断することが可能になりますが、電圧レギュレータまたはレギュレータ出力が追加されます。詳細は、『UltraScale アーキテクチャ PCB デザイン ユーザー ガイド』 (UG583: [英語版](#)、[日本語版](#)) の[このセクション](#)を参照してください。

消費電力モデルの精度

ツールに組み込まれている特性データの精度は、デバイスの入手可能性や製造プロセスの成熟度を反映してしだいに進みます。詳細は、『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』(UG907)の[このセクション](#)を参照してください。



電源/消費電力ヒント: 消費電力見積もりの精度は、入力するデータの精度で決まります。ザイリンクスでは、詳しい見積もりを実行し、その結果と熱評価をデザイン制約として使用することをお勧めします。

デバイスの消費電力と全般的なシステム設計プロセス

プロジェクト考案から完成まで、さまざまなデザイン プロセスの側面が消費電力に影響します。詳細は、『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』(UG907)の[このセクション](#)を参照してください。



電源/消費電力ヒント: 設計プロセス中、`set_operating_conditions -design_power_budget <Power in Watts>` XDC 制約を使用してデザインの総消費電力を消費電力バジェットと比較できます。消費電力バジェットを超えている場合にデザインの消費電力を修正するには、早期に処置を取るのが最も簡単です。

Xilinx Power Estimator (XPE) を使用したワースト ケース消費電力解析

ザイリンクスでは、ワースト ケースの消費電力のボードを設計することをお勧めします。詳細は、『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』(UG907)の[このセクション](#)を参照してください。

クロック リソースのプランニングおよび割り当て

ザイリンクスでは、デザインの最初の段階の 1 つとして、ピン配置を選択する前にクロック リソースを選択することをお勧めします。特にスタックド シリコン インターコネクト (SSI) テクノロジ デバイスの場合、クロックの選択により、特定のピン配置が必要となったり、そのロジックの配置が決定される場合があります。クロックを適切に選択することにより、優れた結果を得ることができます。次を考慮します。

- 特にリソース使用率の高い大型デバイスでは、クロック プランニングと共に制約を作成します。
- デザイン クロージャに必要な場合は、クロック リソースを手動で配置します。
- デバイス特有の機能には、問題を回避してデバイス機能を利用するために、前もってプランニングしておく必要があるものもあります。7 シリーズの機能については、『7 シリーズ FPGA クロッキング リソース ユーザー ガイド』(UG472: [英語版](#)、[日本語版](#))の[このセクション](#)および[このセクション](#)を参照してください。UltraScale デバイスの機能については、『UltraScale アーキテクチャ クロッキング リソース ユーザー ガイド』(UG572: [英語版](#)、[日本語版](#))の[このセクション](#)を参照してください。

関連情報

[クロッキング ガイドライン](#)

[自動パイプライン処理に関する注意事項](#)

[幅の広いバスが SLR 間をまたぐ場合](#)

I/O プランニング デザイン フロー

Vivado IDE では、デザインの I/O ポートおよびクロック ロジックをインタラクティブに確認、表示、割り当て、および検証できます。この環境では、I/O を割り当てたときにそれが正しいかが検証されます。外部パッケージ ピンと内部ダイ パッドの関係も表示されます。

デバイス内のデータフローを表示でき、外部および内部の視点から I/O を適切にプランニングできます。Vivado IDE で I/O を割り当ておよび設定すると、インプリメンテーション用に制約が自動的に作成されます。

Vivado Design Suite の I/O およびクロック プランニング機能の詳細は、次の資料を参照してください。

- 『Vivado Design Suite ユーザー ガイド: I/O およびクロック プランニング』 ([UG899](#))
- [Vivado Design Suite QuickTake ビデオ: I/O プランニングの概要](#)

I/O プランニングの実行可能な Vivado Design Suite プロジェクト タイプ

I/O プランニングは、次のいずれかのプロジェクト タイプで実行できます。

- I/O プランニング プロジェクト: I/O プランニング プロジェクトは簡単なエントリ ポイントで、一部の I/O 制約を指定して、定義したピンから最上位 RTL ファイルを生成できます。
- RTL プロジェクト: RTL プロジェクトでは、合成およびインプリメンテーションが可能であり、より包括的なデザイン ルール チェック (DRC) を実行できます。また、IP コアも生成できます。これは、メモリ インターフェイスのピン配置プランニングおよび GT を使用するコアで重要です。



ヒント: I/O プランニング プロジェクトから開始して、後で RTL プロジェクトに変換することもできます。

合成後のネットリストでは、さらに包括的な DRC を実行できます。インプリメンテーションおよびビットストリーム生成後も同様です。このため、ザイリンクスでは、クロック コンポーネントと一部の基本的なロジックを含むスケルトン デザインを使用して、DRC を実行することをお勧めします。これにより、後でボードで問題が発生しないピン定義を作成できます。

推奨されるサインオフ プロセスは、RTL プロジェクトをビットストリーム生成まで実行し、すべての DRC を実行する方法です。ただし、デザイン サイクルによってはそれほど時間がなく、合成可能な RTL が作成される前に I/O コンフィギュレーションを定義することが必要な場合もあります。Vivado ツールでは RTL 作成前に I/O プランニングを実行できますが、この時点で実行可能な DRC チェックは限られます。また、I/O 規格とピン割り当てを含むダミーの最上位デザインを作成すると、バンク規則に関連する DRC を実行するのに役立ちます。

RTL 作成前の I/O プランニング

合成済みネットリストを生成する前に I/O コンフィギュレーションを定義する必要がある場合は、関連するすべての規則に従っていることを確認してください。Vivado ツールにはピン プランニング プロジェクト環境があり、CSV または XDC フォーマットの I/O 定義ファイルをインポートできます。ポート宣言のみを定義したダミー RTL を作成することもできます。入力信号と出力信号の同時スイッチ ノイズ (SSN) への影響は異なるので、ポートの方向が指定されていると SSN 解析がより正確なものになります。

I/O ポートはインタラクティブに作成および設定することも可能です。基本的な I/O バンク DRC ルールも提供されています。

ご使用のデバイスに適切な I/O 設定は、『7 シリーズ FPGA PCB デザイン ガイド』(UG483: [英語版](#)、[日本語版](#))、『UltraScale アーキテクチャ PCB デザイン ユーザー ガイド』(UG583: [英語版](#)、[日本語版](#))、または『Zynq-7000 SoC PCB デザイン ガイド』(UG933: [英語版](#)、[日本語版](#))を参照してください。詳細は、『Vivado Design Suite ユーザー ガイド: I/O およびクロック プランニング』(UG899)の[このセクション](#)を参照してください。

ネットリスト ベースの I/O プランニング

I/O およびクロック ロジック制約を設定するのに推奨されるデザイン段階は、デザインの合成後です。ネットリストでは、制約を設定するためにクロック ロジック パスが作成されています。I/O およびクロック ロジックの DRC も、より包括的です。

ご使用のデバイスに適切な I/O 設定は、『7 シリーズ FPGA PCB デザイン ガイド』(UG483: [英語版](#)、[日本語版](#))、『UltraScale アーキテクチャ PCB デザイン ユーザー ガイド』(UG583: [英語版](#)、[日本語版](#))、または『Zynq-7000 SoC PCB デザイン ガイド』(UG933: [英語版](#)、[日本語版](#))を参照してください。詳細は、『Vivado Design Suite ユーザー ガイド: I/O およびクロック プランニング』(UG899)の[このセクション](#)を参照してください。

ピン互換デバイスの指定

デザインの初期プランニングでは、最終的なデバイスのサイズを予測するのは困難です。デザイン サイクルの過程でロジックが追加または削除され、デバイス サイズの変更が必要となる場合があります。Vivado ツールでは代替デバイスを指定して、パッケージが同じであれば、I/O ピン設定が選択されたすべてのデバイスで互換したものになるよう定義できます。詳細は、『Vivado Design Suite ユーザー ガイド: I/O およびクロック プランニング』(UG899)の[このセクション](#)を参照してください。



重要: デバイスは、同じパッケージに含まれるものである必要があります。



ヒント: デザインを移行する際のリスクを軽減するには、デザイン プロセスの初期段階でデバイスの選択、ピン配置の選択、およびデザインの基準を注意深く計画します。同じパッケージのサイズの異なるデバイスに移行する際は、ピン配置、クロック、およびリソース管理を考慮します。

ピン割り当て

ピン配置を適切に選択することは、デザイン ロジックの適切な配置、短い配線、消費電力の削減、およびパフォーマンスの向上につながります。大型のデバイスでは、ピン配置が分散していると関連の信号の距離が長くなるので、ピン配置を適切に選択することは特に重要です。詳細は、『Vivado Design Suite ユーザー ガイド: I/O およびクロック プランニング』(UG899)の[このセクション](#)を参照してください。

ピン配置にザイリンクス ツールを使用

ザイリンクス ツールでは、インタラクティブにデザイン プランニングおよびピン選択を実行できます。ツールがどれだけ効率的であるかは、供給する情報によって決まります。Vivado I/O プランナーなどのツールを使用すると、ピン配置の作業が簡略化されます。これらの機能では、I/O 配置をグラフィカルに表示したり、クロックと I/O コンポーネントの関係を表示でき、ピンの選択を解析するデザイン ルール チェック (DRC) も提供されています。

デザイン バージョンを使用できる場合は、最上位フロアプランを作成してデバイス内のデータフローをすばやく解析できます。詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906)を参照してください。

必要な情報

ツールが効果的に機能するためには、I/O の特性およびトポロジに関する情報をできるだけ多く供給する必要があります。I/O 規格、駆動電流、スルー レート、I/O の方向などの電気特性を指定する必要があります。

また、クロックのトポロジやタイミング制約などの関連情報も考慮する必要があります。クロッキングはピン配置に、ピン配置はクロッキングに大きく影響します。

トランシーバー、PCIe、メモリ インターフェイスなどの I/O 要件を持つ IP の場合は、I/O ピン割り当てを終了する前に IP を設定しておく必要があります。I/O の電気特性の指定方法は、『Vivado Design Suite ユーザー ガイド: I/O およびクロック プランニング』(UG899)の[このセクション](#)を参照してください。

関連情報

[クロッキング ガイドライン](#)

ピン配置の選択

ザイリンクスでは、次に説明するように、一部の信号ピン配置を注意して選択することをお勧めします。

ピン配置選択の一般的なガイドライン

次に、一般的なガイドラインを示します。

- 同じインターフェイスのデータ ピン、アドレス ピン、制御ピンを同じバンクにグループ化します。これらのコンポーネントを同じバンクにグループ化できない場合、隣接するバンクにグループ化します。

注記: SSI テクノロジ デバイスでは、隣接するバンクも同じ SLR (Super Logic Region) 内に配置する必要があります。

- インターフェイス制御信号(クロック、イネーブル、リセット、ストロブ)を、制御の対象となるデータ バスの中央に配置します。
- デザイン全体で使用されるファンアウトの大きい制御信号を、デバイス中央に配置します。

注記: SSI テクノロジ デバイスでは、これらの信号は、駆動先の SLR コンポーネントの中央にある SLR に配置します。

コンフィギュレーション ピン

効率の良いシステムを設計するには、システム要件に最適なデバイス コンフィギュレーション モードを選択する必要があります。次の事項を考慮してください。

- 専用または多目的コンフィギュレーション ピン。

どのコンフィギュレーション モードでも、一部のデバイス ピンを専用ピンとして使用し、多目的ピンをコンフィギュレーション中に一時的に使用できます。多目的ピンは、コンフィギュレーションが完了すると、汎用ピンになります。

- コンフィギュレーション モードにより、一部のデバイス I/O バンクの電圧が制限されることがあります。
- 異なるコンフィギュレーション ピンに適した終端を選択します。
- コンフィギュレーション ピンのプルアップまたはプルダウン抵抗に推奨される値を使用します。



推奨: コンフィギュレーション クロックは低速ですが、ボード上でシグナル インテグリティ 解析を実行して信号にノイズがないことを確認してください。

コンフィギュレーション オプションは複数あり、柔軟性がありますが、各システムに最適なソリューションがあるのが一般的です。コンフィギュレーション オプションを選択する際は、次を考慮します。

- セットアップ
- スピード
- コスト
- 複雑さ

デバイス コンフィギュレーション オプションの詳細は、『Vivado Design Suite ユーザー ガイド: プログラムおよびデバッグ』 (UG908) を参照してください。

関連情報

[コンフィギュレーション](#)

メモリ インターフェイス

ザイリンクス メモリ IP を使用する場合は、追加の I/O ピン プランニング手順が必要です。IP をカスタマイズした後、Vivado IDE でエラボレート済みデザインまたは合成済みデザインを開いて最上位 IP ポートを物理的なパッケージ ピンに割り当てます。各メモリ IP に関連するポートはすべて I/O ポート インターフェイスとしてグループ化されており、簡単に特定および割り当てできます。メモリ バンク/バイト プランナーが提供されており、メモリ I/O ピン グループを物理的なデバイス ピンのバイト レーンに割り当てることができます。詳細は、『Vivado Design Suite ユーザー ガイド: I/O およびクロック プランニング』 (UG899) の[このセクション](#)を参照してください。

中央に I/O 列があるデバイスの場合は特に、メモリ インターフェイスを割り当てる際に注意して、できるだけ密集が起こらないようにしてください。メモリ インターフェイスをまとめると、デバイス中で配線ボトルネックが発生してしまう可能性もあります。デザインおよびピン配置のガイドラインについては、『Zynq-7000 SoC および 7 シリーズ デバイス メモリ インターフェイス ソリューション ユーザー ガイド』 (UG586: [英語版](#)、[日本語版](#)) および『UltraScale アーキテクチャ FPGA メモリ IP LogiCORE IP 製品ガイド』 (PG150: [英語版](#)、[日本語版](#)) を参照してください。これらのガイドで推奨されるトレース長を使用し、正しい終端が使用されていることを確認して、メモリ IP I/O 割り当ての後 DRC を実行してピン配置を検証してください。メモリ インターフェイス信号の終端および配線のガイドラインは、『UltraScale アーキテクチャ PCB デザイン ユーザー ガイド』 (UG583: [英語版](#)、[日本語版](#)) を参照してください。

ギガビット トランシーバー (GT)

ギガビット トランシーバー (GT) には、特定のピン配置要件がありますので、次に注意する必要があります。

- 基準クロックの共有
- 同じクワッド内での PLL の共有
- PCIe などのハード ブロックの配置と、それらのトランシーバーへの距離
- SSI テクノロジ デバイスの場合は、SLR 境界のクロッシング

ザイリンクスでは、コアの生成に GT ウィザードを使用することをお勧めします。または、このプロトコルにはザイリンクス IP コアを使用できます。推奨されるピン配置は、該当する製品ガイドを参照してください。

クロック リソースをバランスよく使用するため、Vivado 配置で GT 出力クロック (TXOUTCLK または RXOUTCLK) が供給されるロードをそのクロックをソースとする GT の横に制約するよう試みられます。SSI テクノロジ デバイスでは、GT が別の SLR に隣接するクロック領域に配置されている場合、SLL に入力される信号または SLL から出力される信号と GT 出力クロック ロードの間で配線リソースを取り合うことになります。そのため、SLR 間をまたぐ部分の横にあるクロック領域に GT が配置されていると、これらのクロック領域内にある SLL への配線接続および SLL からの配線接続が削減される可能性があります。

高速 I/O

HP (High Performance) バンクと HR (High Range) バンクは、信号を送受信する速度が異なります。I/O の速度によって、HP バンクまたは HR バンクを選択します。

内部 VREF および DCI カスケード制約

DCI カスケードおよび内部 VREF の設定に基づいて、ピンを通常の I/O として使用するよう解放できます。これらの設定により、関連の DRC チェックが実行され、制約が有効であることも確認されます。詳細は、ご使用のデバイスによって『7 シリーズ FPGA SelectIO リソース ユーザー ガイド』(UG471: [英語版](#)、[日本語版](#)) または『UltraScale アークテクチャ SelectIO リソース ユーザー ガイド』(UG571: [英語版](#)、[日本語版](#)) を参照してください。

インターフェイスの帯域幅の検証

小型のコネクティビティ デザインを作成し、デザイン上の各インターフェイスを検証します。このような小型のデザインでは、特定のハードウェア インターフェイスのみが使用され、次が有効になります。

- ピン配置、クロッキング、およびタイミングのフル DRC チェック
- ボードが戻された場合のハードウェア テスト デザイン
- Vivado ツールを使用した短時間インプリメンテーション (最速のインターフェイス デバッグ方法)

これらのインターフェイス用にテスト データを生成するには、複数のオプションがあります。一部のインターフェイス IP コアでは、Vivado ツールで次のテスト デザインを生成できます。

- SerDes の IBERT
- IP コア内のサンプル デザイン



ヒント: テスト デザインがない場合は、AXI トラフィック ジェネレーターの使用を考慮してください。

プロダクション環境では、システム レベルのテスト用に別のデザインを作成する必要があることもあります。これは通常、テスト済みインターフェイスとオプションでプロセッサを含む 1 つのデザインです。このデザインは、小さなコネクティビティ デザインを使用してデザインの再利用を活用することにより構築できます。このデザインはフローの初期段階では必要ありませんが、Vivado IP インテグレーターを使用してすばやく作成でき、使用するとより多くの DRC チェックが有効になり、ソフトウェアを早期開発できるようになります。

SSI デバイスでの設計

SSI のピン配置に関する注意事項

特定の SLR に配置されているコンポーネントのピンは、同じ SLR 内に配置します。たとえば、外部インターフェイスの一部としてデバイスの DNA 情報を使用する場合は、そのインターフェイスのピンを DNA_PORT が存在するマスター SLR に配置します。そのほかに、次の事項を考慮する必要があります。

- 1 つのインターフェイスに含まれるすべてのピンを同じ SLR にグループ化します。
- 複数の SLR のコンポーネントを駆動する信号は、中央の SLR に配置します。
- SLR 間で CCIO または CMT コンポーネントをバランスよく使用します。

- SLR 間をまたぐパスを削減します。

SLR (Super Logic Region)

SLR (Super Logic Region) は、SSI テクノロジ デバイスに含まれる 1 つのデバイス ダイ スライスです。各 SLR は、CLB、ブロック RAM、DSP タイル、GT などのデバイス リソースのサブセットを含み、SSI ではないデバイスと同様の構造になっています。

複数の SLR コンポーネントが垂直方向に積み重ねられ、インターポーザーを介して接続されて、1 つの SSI テクノロジ デバイスを構成しています。一番下の SLR は SLR0 で、上に行くほど番号が増加します。たとえば、XC7V2000T デバイスには 4 つの SLR コンポーネントがあります。一番下の SLR は SLR0 で、その上が SLR1、さらにその上が SLR2 で、一番上の SLR は SLR3 になります。

注記: ザイリンクス ツールではグラフィカル ユーザー インターフェイス (GUI) およびレポートで SLR コンポーネントが明確に識別されます。

SLR の名前

ターゲット デバイスで SLR の名前がどのように付けられているかを理解することは、次の作業で重要となります。

- ピンの選択
- フロアプラン
- タイミングおよびその他のレポートの解析
- ロジックの位置、そのロジックのソースまたはデスティネーションの特定

Vivado Tcl コマンド `get_slrs` を使用すると、特定のデバイスの SLR に関する特定の情報を取得できます。たとえば、次のコマンドを使用します。

- デバイスの SLR 数を取得するには、`llength [get_slrs]` を使用します。
- `my_cell` が配置されている SLR を取得するには、`get_slrs -of_objects [get_cells my_cell]` を使用します。

マスター SLR

各 SSI テクノロジ デバイスには、マスター SLR が 1 つあります。マスター SLR には、デバイスのコンフィギュレーションを開始するプライマリ コンフィギュレーション ロジックと、すべての SLR コンポーネントが含まれています。マスター SLR には、コンフィギュレーションに使用される DNA_PORT および EFUSE_USE が含まれます。これらのコンポーネントを使用すると、配置配線により関連のピンおよびロジックが適切な SLR に割り当てられます。通常、追加の操作は必要ありません。



ヒント: Vivado Design Suite でどの SLR がマスター SLR であるかを確認するには、`get_slrs -filter IS_MASTER` Tcl コマンドを使用します。

シリコン インターポーザー

シリコン インターポーザーは、SSI テクノロジ デバイス内の不動態層で、次の SLR コンポーネント間が配線されます。

- コンフィギュレーション

- グローバル クロック
- 汎用インターコネクト

SLL (Super Long Line) 配線

SLL (Super Long Line) 配線では、1 つの SLR からそのデバイス内の別の SLR までの信号が接続されます。



ヒント: SLR 間で使用可能な SLL の数を判断するには、SLR プロパティを確認します。次に例を示します。

```
get_property NUM_TOP_SLLS [get_slrs SLR0]  
get_property NUM_BOT_SLLS [get_slrs SLR1]
```

伝搬の制限



ヒント: SLR 間的高速伝搬では、SLR 境界をまたぐ信号にレジスタを付けてください。

SLR コンポーネント間では、SLL 信号が唯一のデータ接続です。

次のものは SLR コンポーネント間では伝搬されません。

- キャリー チェーン
- DSP カスケード
- ブロック RAM アドレスのカスケード
- DCI カスケードやブロック RAM カスケードなどのその他の専用接続

ツールでは通常、伝搬に関するこの制限が考慮されます。デザインが適切に配線され、デザイン要件が満たされるようにするには、次を実行するときにこの制限を考慮する必要があります。

- 非常に長い DSP カスケードを作成し、そのようなロジックを SLR 境界の近くに手動で配置する
- デザインのピン配置を指定する

SLR の使用に関する考慮事項

Vivado インプリメンテーション ツールでは、ロジックを複数の SLR に分割するための特別なアルゴリズムが使用されます。SSI テクノロジ デバイスをターゲットにするデザインのタイミング クロージャが困難な場合は、次のガイドラインを使用することで改善できます。

タイミング クロージャおよびコンパイル時間を改善するには、Pblock を使用してロジックを各 SLR に割り当てて、その各 SLR 内でファブリック リソース タイプすべての中から過度に使用率が高いものがないようにします。たとえば、ブロック RAM の使用率が 70% のデザインで、ブロック RAM リソースが SLR 間でバランスよく配置されておらず、1 つの SLR でブロック RAM の 85% 以上が使用されている場合、タイミング クロージャ問題が発生する可能性があります。



ヒント: SLR Pblock は、完全な SLR を指定することにより定義できます (`resize_pblock pblock_SLR0 - add SLR0` など)。

次の vu160 の使用率レポートの例では、全体のブロック RAM の使用率は 56% で、SLR0 では 59%、SLR1 では 40%、SLR2 では 58% です。ブロック RAM の使用率は SLR 間に均等に分散されており、各 SLR の使用率は妥当なものであるため、タイミングを満たすために Vivado インプリメンテーションがより柔軟に実行されます。

図 7: 使用率レポートのブロック RAM セクション

3. BLOCKRAM

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	1843	0	3276	56.26
RAMB36/FIFO*	1820	2	3276	55.56
FIFO36E2 only	88			
RAMB36E2 only	1732			
RAMB18	46	8	6552	0.70
RAMB18E2 only	46			

図 8: 使用率レポートの SLR セクション

14. SLR CLB Logic and Dedicated Block Utilization

Site Type	SLR0	SLR1	SLR2	SLR0 %	SLR1 %	SLR2 %
CLB	22361	40858	42493	61.70	91.28	94.94
CLBL	17061	31936	33311	60.76	90.99	94.90
CLBM	5300	8922	9182	64.95	92.36	95.05
CLB LUTs	104506	196677	236523	36.05	54.93	66.05
LUT as Logic	104482	194364	235584	36.04	54.28	65.79
using O5 output only	268	913	789	0.09	0.25	0.22
using O6 output only	101383	180318	219746	34.97	50.36	61.37
using O5 and O6	2831	13133	15049	0.98	3.67	4.20
LUT as Memory	24	2313	939	0.04	2.99	1.22
LUT as Distributed RAM	24	1316	136	0.04	1.70	0.18
using O5 output only	0	0	0	0.00	0.00	0.00
using O6 output only	8	48	64	0.01	0.06	0.08
using O5 and O6	16	1268	72	0.02	1.64	0.09
LUT as Shift Register	0	997	803	0.00	1.29	1.04
CLB Registers	191575	330568	473777	33.04	46.16	66.16
CARRY8	901	1715	3816	2.49	3.83	8.53
F7 Muxes	1725	10631	4762	1.19	5.94	2.66
F8 Muxes	216	2891	366	0.30	3.23	0.41
F9 Muxes	0	0	0	0.00	0.00	0.00
Block RAM Tile	599	512	732	59.42	40.63	58.10
RAMB36/FIFO	598	501	721	59.33	39.76	57.22
RAMB36E2 only	598	472	662	59.33	37.46	52.54
RAMB18	2	22	22	0.10	0.87	0.87
RAMB18E2 only	2	22	22	0.10	0.87	0.87
URAM	0	0	0	0.00	0.00	0.00
DSPs	1	2	12	0.21	0.33	2.00
PLL	0	0	0	0.00	0.00	0.00
MMCM	0	0	0	0.00	0.00	0.00
Unique Control Sets	1125	3482	8567	1.55	3.89	9.57

ザイリンクスでは、ブロック RAM および DSP グループを SLR Pblock に割り当て、共有信号が SLR 間をまたぐバスを最小限に抑えることをお勧めします。たとえば、複数の SLR に分散しているブロック RAM グループにファンアウトするアドレスバスがあると、SLR 間をまたぐことによりタイミングクリティカルな信号に遅延が追加されるため、タイミングクロージャが達成しにくくなることがあります。

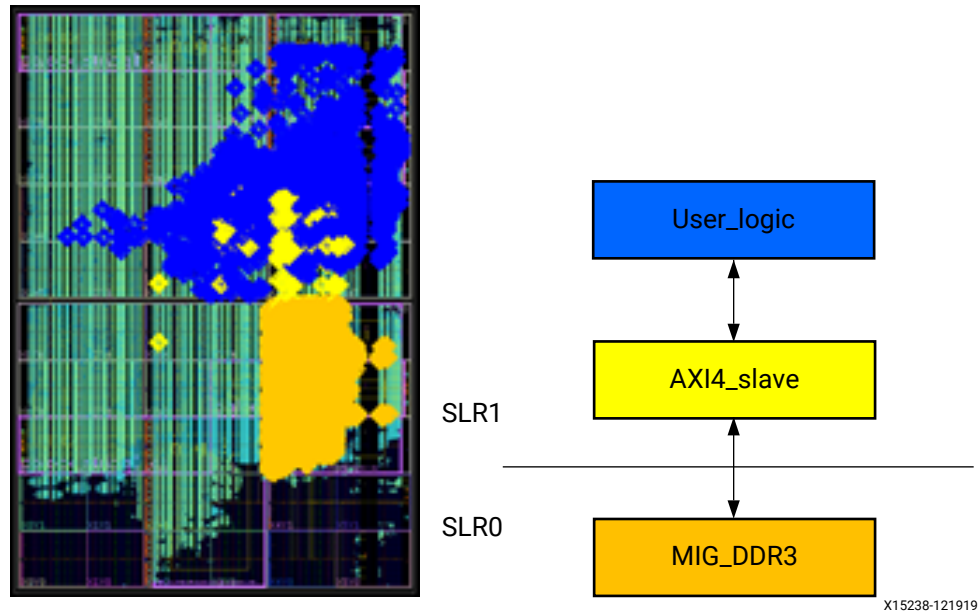
IP は、デバイスリソースの位置やユーザー I/O の選択により SLR に固定されます (例: GT、ILKN、PCIe、および CMAC 専用ブロックまたはメモリインターフェイスコントローラー)。ザイリンクスでは、次を推奨します。

- データフローが SLR の境界を何度もまたぐことがないように、専用ブロックの位置とピン配置の選択には特に注意してください。

- 密に相互接続されたモジュールと IP は、同じ SLR 内に配置します。これが不可能な場合は、パイプライン レジスタを追加してより柔軟な配置が実行されるようにし、ロジック グループ間の接続が SLR 間をまたぐ場合でも適切なソリューションが見つけれられるようにしてください。
- クリティカル ロジックは同じ SLR 内に配置します。主なモジュールがそれらのインターフェイスで適切にパイプライン処理されるようにすると、配置で SLR 間をまたぐ部分がフリップフロップからフリップフロップの接続になる SLR を見つけることができるようになります。

次の図では、SLR0 に制約されたメモリ インターフェイスが SLR1 のユーザー ロジックを駆動する必要があります。AXI4-Lite スレーブ インターフェイスはメモリ IP バックエンドに接続されており、メモリ IP と AXI4-Lite スレーブ インターフェイス間の境界が適切に定義されているため、SLR0 から SLR1 への遷移が問題なく実行されます。

図 9: SLR0 のメモリ インターフェイスが SLR1 のユーザー ロジックを駆動

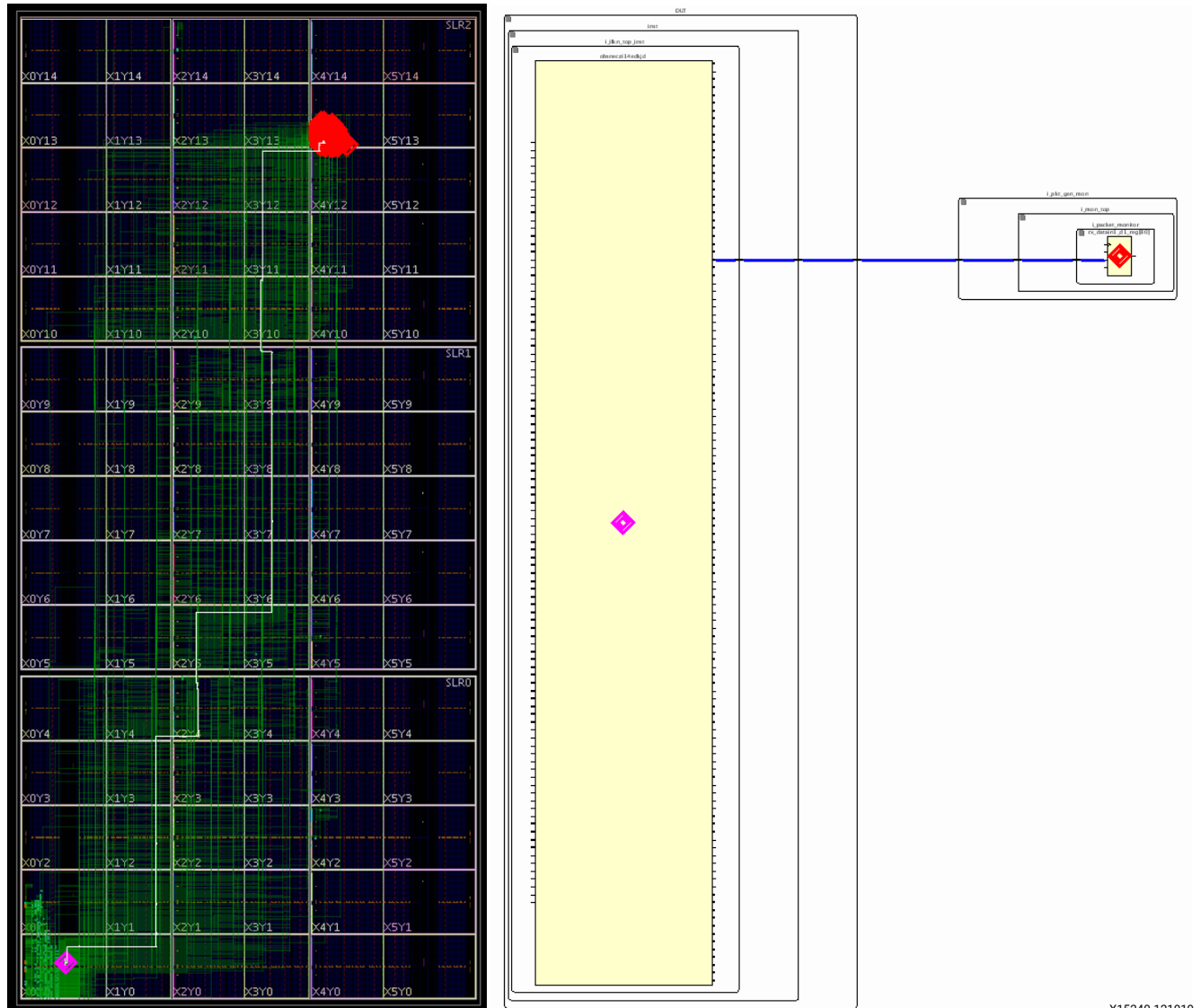


幅の広いバスが SLR 間をまたぐ場合

幅の広いバスが SLR 間をまたぐデータフロー要件がある場合は、パイプライン ストラテジを使用してタイミング クロージャを改善し、長いリソースの配線密集を軽減します。ザイリンクスでは、250 MHz を超える周波数で動作する幅の広いバスには、少なくとも 3 段のパイプライン (SLR の一番上に 1 段、一番下に 1 段、真ん中に 1 段) を使用して SLR 間をまたぐようにすることをお勧めします。非常にパフォーマンスの高いバスの場合や、水平方向に横切ったり、垂直方向に縦断したりする場合は、さらに多くのパイプライン段が必要になることもあります。

次の図は、vu190-2 デバイスで SLR 間をまたぐワースト ケースの状況を示しています。この例では、SLR0 の左下の Interlaken 専用ブロックから、SLR2 の右上に割り当てられたパケット モニター ブロックに到達しています。パケット モニターを行き来するデータ バスに対してパイプライン レジスタがないと、デザインはタイミング要件である 300 MHz を大差で満たすことができません。

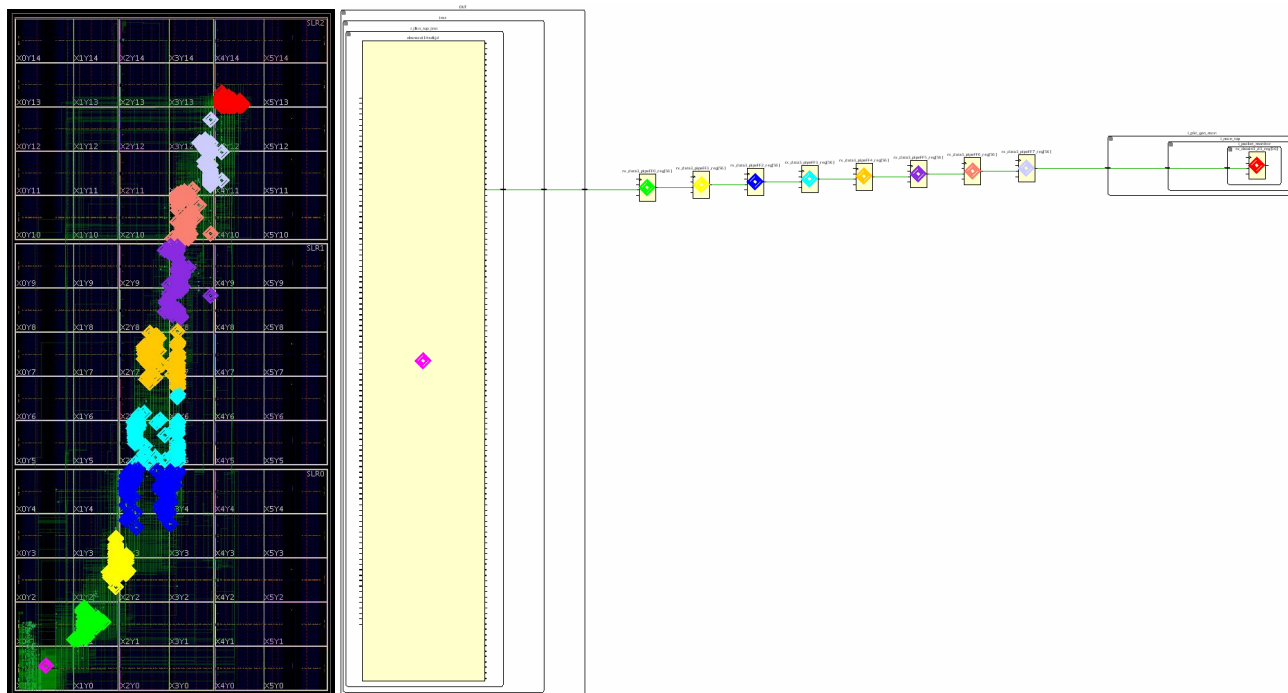
図 10: SLR 間をまたぐデータパス (パイプライン フリップフロップなし)



X15240-121919

SLR0 から SLR2 に遷移しやすいようにするため 7 段のパイプラインを追加すると、デザインのタイミング要件が満たされるようになります。これにより、次の図に示すように、垂直および水平方向の長い配線リソースの使用も減ります。

図 11: SLR 間をまたぐデータパス (パイプライン フリップフロップを追加)



X15239-110415



ヒント: SLR をまたぐ幅の広いバスのタイミング クロージャを達成するには、AXI Register Slice IP またはカスタム自動パイプライン IP を使用します。

関連情報

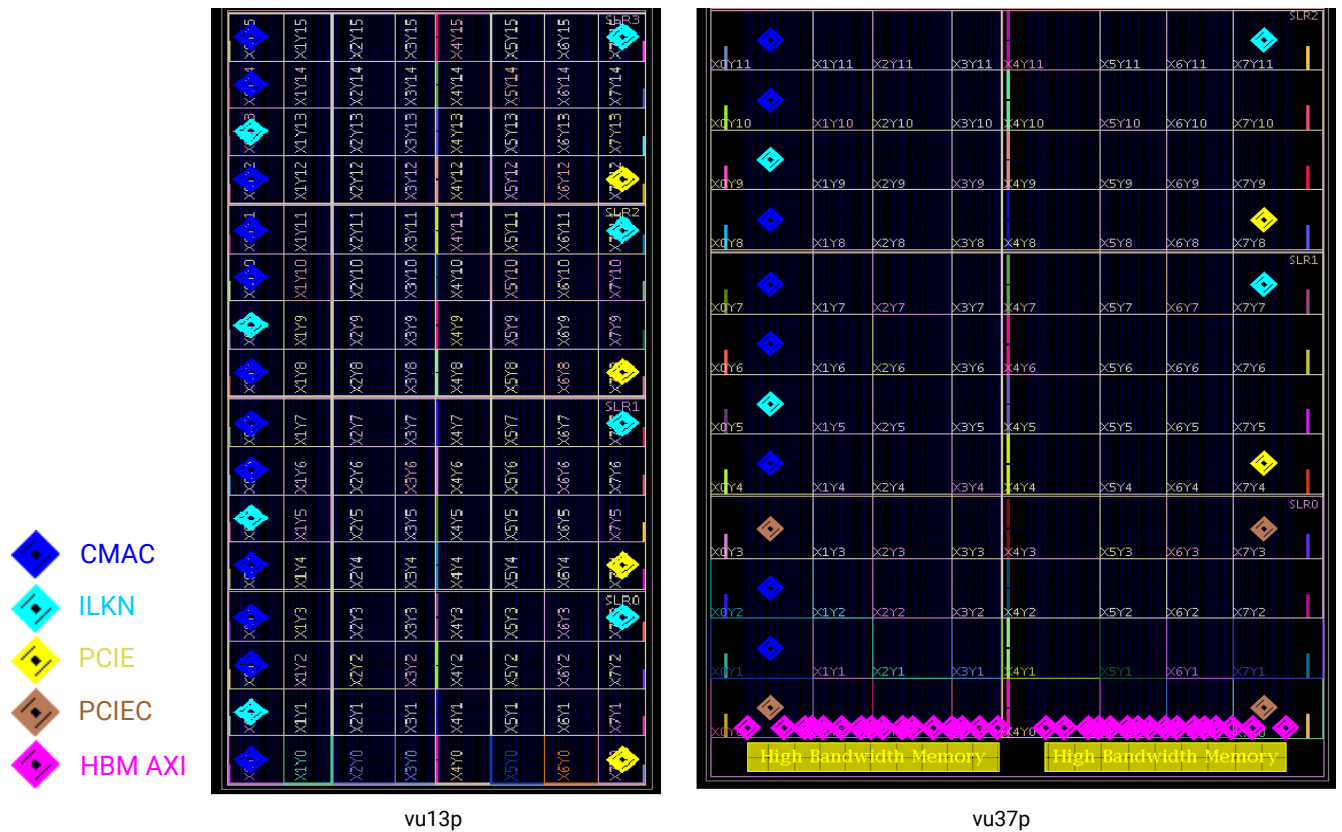
自動パイプライン処理に関する注意事項

HBM デバイスでの設計

Virtex® UltraScale+™ HBM デバイスには、デバイス ダイの横に 4 GB 高帯域幅メモリ (HBM) スタックが組み込まれています。SSI テクノロジを使用するデバイスは、デバイスの下部にあるシリコン インターポーザーを介して接続されているメモリコントローラーを使用して HBM スタックと通信します。各 Virtex UltraScale+ HBM デバイスには 1 つまたは 2 つの 4 GB HBM スタックが含まれているので、デバイスごとに 8 GB までの HBM があります。デバイスには、HBM との通信に使用される 32 個の HBM AXI インターフェイスが含まれます。ビルトイン スイッチにより柔軟なアドレス指定が可能で、32 個の HBM AXI インターフェイスどれでも、HBM スタックのいずれかまたは両方にあるどのメモリ アドレスにもアクセスできます。このデバイスと HBM スタック間の柔軟な接続は、フロアプランおよびタイミング クロージャで有益です。

次の図に、Virtex UltraScale+ vu13p デバイスと Virtex UltraScale+ HBM vu37p デバイスを示します。VU37P デバイスでは、VU13P デバイスの下の 2 つの SLR が HBM スタック (vu13p デバイスの SLR0) と 32 個の HBM AXI インターフェイスを含む SLR (vu13p デバイスの SLR1) に置き換えられています。上の 2 つの SLR は、vu13p と vu37p デバイスで同一です。

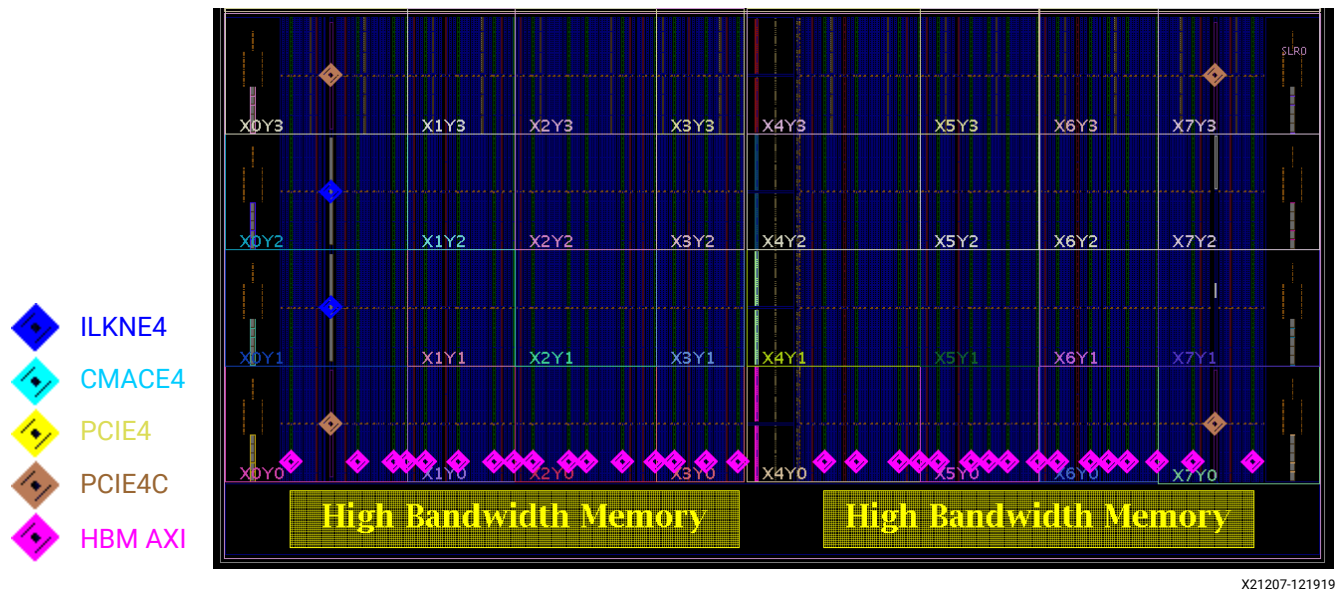
図 12: vu13p および vu37p デバイスの [Device] ビュー



X21195-121919

vu37p デバイスでは、SLR0 に 4 つの PCIE4C サイト、2 つの ILKNE4 サイト、および 32 個の HBM AXI インターフェイスが含まれます。Virtex UltraScale+ HBM SLR0 の 4 つの PCIE4C サイトは、 V_{CCINT} が 0.72V のときに PCIe Gen3 x 16 を使用した CCIX (Cache Coherent Interconnect for Accelerators) プロトコルが可能です。

図 13: Virtex UltraScale+ HBM vu37p デバイスの SLR0



X21207-121919

HBM デバイスを使用する場合の配置における考慮事項

SLR 間をまたぐパスのパイプライン処理における考慮事項

Virtex UltraScale+ HBM デバイスでの SLR 間をまたぐパスのパイプライン処理における考慮事項は、ほかの UltraScale および Virtex UltraScale+ SSI テクノロジ デバイスと同じです。

SLR2 のファブリック ロジックから SLR0 の HBM AXI インターフェイスへのパスには、タイミングを満たすため 5 段以上のパイプラインが必要です。Virtex UltraScale+ HBM デバイスのデザインを注意深くプランニングすることにより、追加のパイプライン段の必要性を低減し、配線密集を緩和できます。次の図に、SLR2 から HBM AXI インターフェイスへの SLR 間をまたぐパスの例を示します。

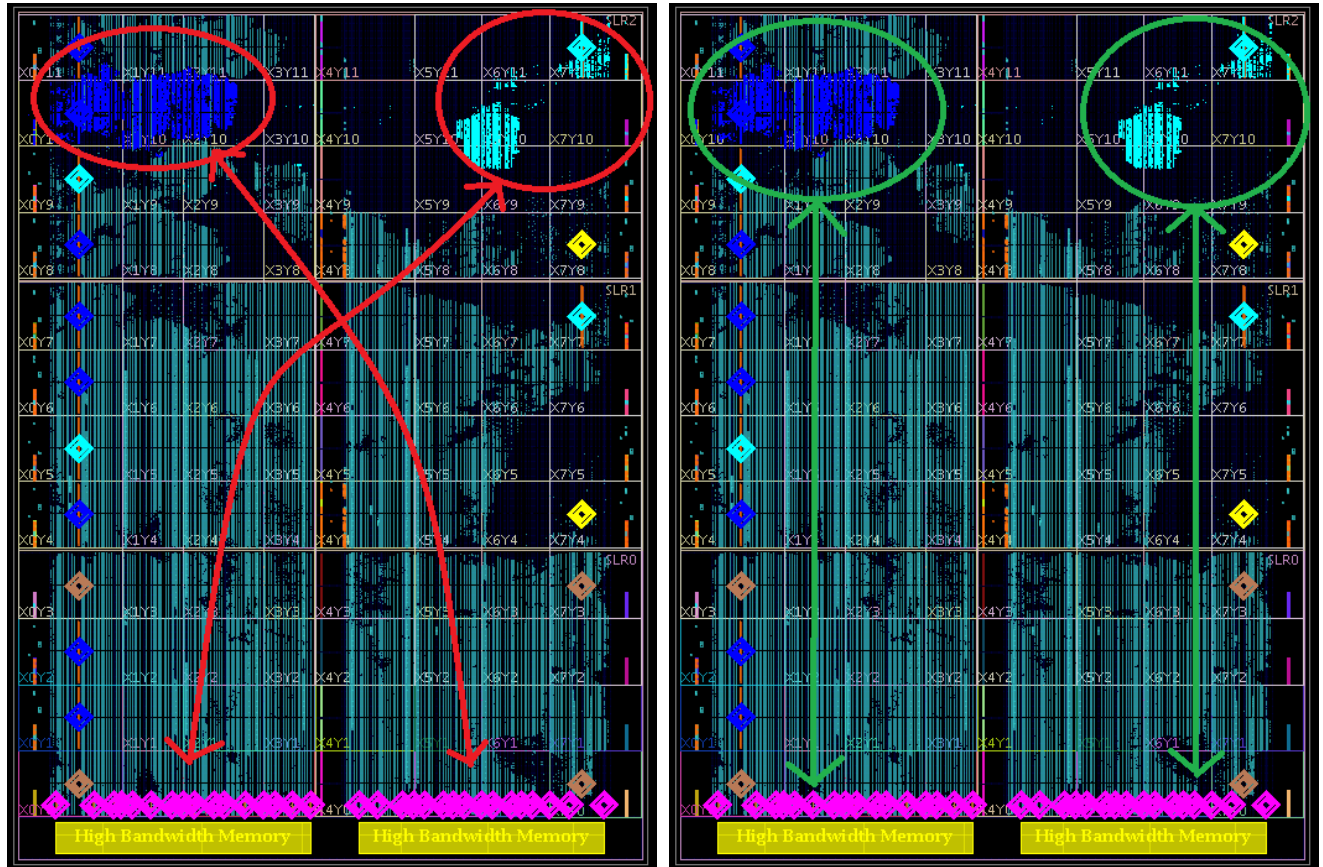


推奨: ザイリンクスでは、SLR2 および SLR1 から HBM AXI インターフェイスへのパスが垂直に揃うようにし、デバイスを斜めに横切らないようにすることをお勧めします。



ヒント: HBM インターフェイスと任意の SLR の間のタイミング クロージャを 450 MHz で達成するには、自動パイプライン (AXI Register Slice IP など) を使用します。

図 14: HBM の最適でないデザイン プランニング (左) と最適なデザイン プランニング (右)



X21196-121919

関連情報

自動パイプライン処理に関する注意事項
幅の広いバスが SLR 間をまたぐ場合

SLR0 内でのリソース プランニング

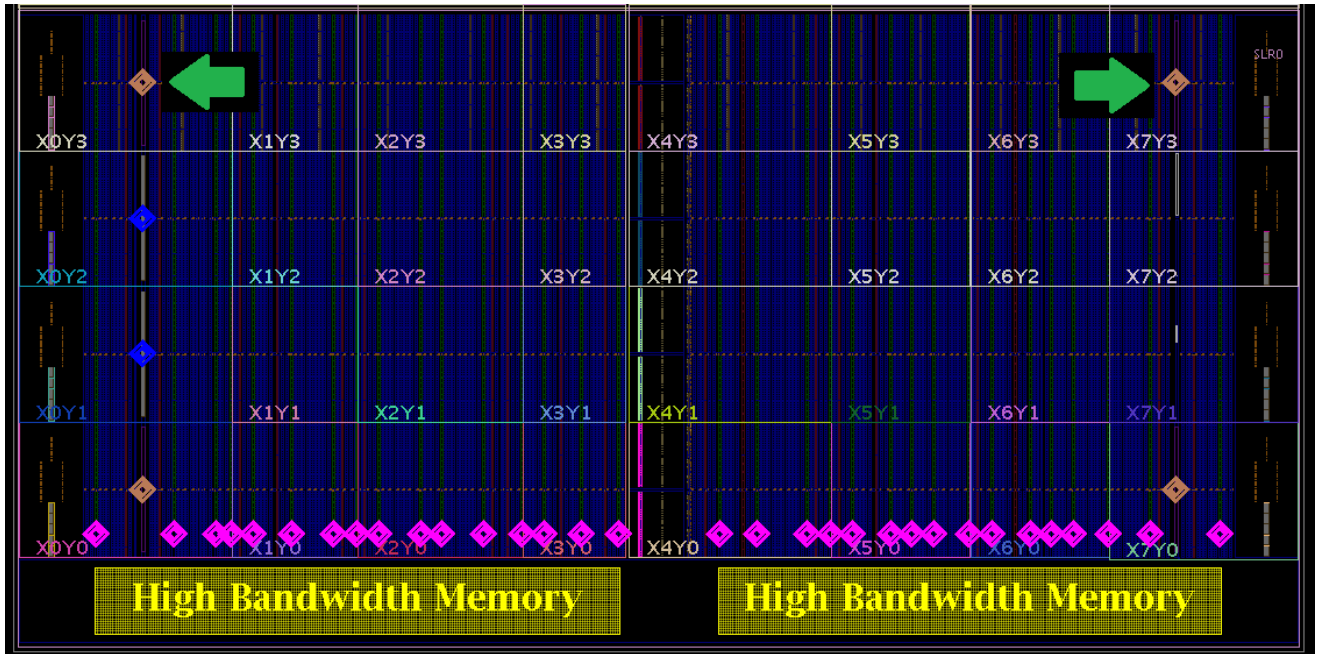
SLR0 内の HBM AXI インターフェイスおよびその他のロジックを適切に制御することにより、QoR (結果の品質) が最適なものとなり、配線の密集を最小限に抑えることができます。次に、HBM デバイスの SLR0 のデザイン プランニングにおける一般的な考慮事項を示します。

- HBM AXI インターフェイスを頻繁に使用するデザインでは HBM AXI インターフェイスに必要なリソースを確保するため、SLR0 の HBM 以外のロジックの全体的なファブリック使用率を低くします。
- SLR0 で MIG IP を使用すると、デバイスの I/O 列の近くに配置されている HBM AXI インターフェイスでタイミング クロージャを達成するのが困難になります。MIG IP を使用する場合は、SLR2 または SLR1 の I/O 列を使用することを検討します。
- アドレス範囲と HBM AXI インターフェイスの物理的な位置はデザインのレイテンシおよび帯域幅に影響することがあるので、注意してください。HBM のパフォーマンスを最適化するには、アドレス指定される HBM スタックとデバイスの同じ側にある HBM AXI インターフェイスを使用します。

SLR0 内の PCIE4C から HBM AXI へのパス

HBM および PCIE4C を使用する設計で最適なタイミング QoR を達成し、配線の密集を最小限に抑えるには、ザイリンクスでは SLR0 の 32 個の HBM AXI インターフェイスから最遠の PCIE4C サイトを使用することをお勧めします。次の図に、これらのサイト PCIE4CE4_X0Y1 および PCIE4CE4_X1Y1 を緑色の矢印で示します。

図 15: Virtex UltraScale+ HBM vu37p デバイスの SLR0 で推奨される PCIE4C サイト



コンフィギュレーション

コンフィギュレーションは、アプリケーション特定のデータをデバイスの内部メモリに読み込むプロセスです。ザイリンクス デバイスのコンフィギュレーション データは CMOS コンフィギュレーション ラッチ (CCL) に格納されるので、コンフィギュレーション データには揮発性があり、デバイスに電源を投入するたびに読み込み直す必要があります。

ザイリンクス デバイスは、外部不揮発性メモリ デバイスからコンフィギュレーション ピンを介して読み込むことができます。次のような外部スマート ソースからコンフィギュレーションすることも可能です。

- マイクロプロセッサ
- マイクロコントローラー
- DSP プロセッサ
- パソコン (PC)
- ボード テスター

ボード プランニングでは、初期段階でコンフィギュレーションを考慮する必要があります。これにより、コンフィギュレーションだけでなくデバッグしやすくなります。サポートされるコンフィギュレーション モードおよびそのピン数、パフォーマンス、コストにおけるトレードオフの詳細は、各デバイス ファミリのコンフィギュレーション ユーザー ガイドを参照してください。

関連情報

[その他のザイリンクス資料](#)

ボード設計でのヒント

ボード設計では、コンフィギュレーションだけでなく、どのインターフェイスおよびピンがデバッグに役立つかを考慮することが重要です。たとえば、ザイリンクスでは、JTAG が主なコンフィギュレーション モードでない場合でも、ボードの JTAG インターフェイスにアクセスできるようにしておくことをお勧めします。JTAG インターフェイスを使用すると、デバイス ID およびデバイス DNA 情報を確認でき、プロトタイプ時にこのインターフェイスを使用して間接プログラム ソリューションをイネーブルにできます。

また、INIT_B や DONE などの信号は、デバイス コンフィギュレーションのデバッグに重要です。INIT_B 信号には複数の機能があります。電源投入時の初期化の完了を示し、CRC エラーが発生したことを示すこともできます。ザイリンクスでは、INIT_B および DONE 信号を LED ドライバーおよびプルアップを使用して LED に接続することをお勧めします。

推奨されるプルアップ値は、該当するデバイスのコンフィギュレーション ユーザー ガイドを参照してください。

- 『7 シリーズ FPGA コンフィギュレーション ユーザー ガイド』 (UG470: [英語版](#)、[日本語版](#))
- 『UltraScale アーキテクチャ コンフィギュレーション ユーザー ガイド』 (UG570: [英語版](#)、[日本語版](#))

ボード レベルのピン接続を特定および確認するには、次の回路図チェックリストを使用してください。

- 『7 シリーズ回路図レビュー推奨事項』 ([XMP277](#))
- 『Kintex UltraScale および Virtex UltraScale FPGA 回路図レビュー チェックリスト』 ([XTP344](#))
- 『UltraScale+ FPGA および Zynq UltraScale+ MPSoC 回路図レビュー チェックリスト』 ([XTP427](#))

RTL を使用したデザインの作成

デバイスの I/O プランニングを実行し、PCB のレイアウトをプランニングし、Vivado® Design Suite の使用モデルを決定したら、デザインの作成を開始できます。デザインの作成には、次が含まれます。

- デザイン階層をプランニング
- デザイン内で使用およびカスタマイズする IP コアを特定
- 適切な IP がないインターコネクト ロジックおよび機能のカスタム RTL を作成
- タイミング制約、消費電力制約、および物理制約を作成
- 合成およびインプリメンテーションで使用される追加の制約、属性、およびその他の要素を指定

デザインを作成する際の主な考慮点は、次のとおりです。

- 必要な機能を達成する
- 必要な周波数で動作する
- 必要な信頼度で動作する
- シリコン リソースおよび消費電力を要件内に収める

この段階での決定事項が、最終的な製品に影響します。この段階で不適切な決定を下すと、後の段階で問題となり、デザイン サイクル全体で問題が発生する可能性があります。プロセスの初期段階で時間をかけてデザインを注意深くプランニングすると、デザイン要件を満たし、ラボでのデバッグ時間を最小限に抑えることができます。

適切なデザイン階層の定義

デザインの作成では、まずデザインをどのように論理的に分割するかを決定します。階層を考慮する際は、特定のファンクションを含むデザインの部分を分割することを考えます。これにより、特定の設計者が IP を個別に設計できるほか、コードを別にして再利用することもできます。

ただし、機能のみに基づいて階層を定義する場合、タイミング クロージャ、実行時間、およびデバッグのためにどのように最適化するかは考慮されません。階層プランニングの際に次についても考慮すると、タイミング クロージャを達成しやすくなります。

I/O コンポーネントを最上位付近に追加

可能であれば、I/O コンポーネントを最上位付近に追加し、デザインを解釈しやすくします。コンポーネントを推論する際は、必要な機能を記述します。合成ツールにより HDL コードが解釈され、その機能を実行するのに使用するハードウェア コンポーネントが決定されます。推論可能なコンポーネントは、単純なシングルエンド I/O (IBUF、OBUF、OBUFT、および IOBUF) および I/O のシングル データ レートのレジスタです。

ツールを使用して IOBUF または OBUFT コンポーネントを推論する場合は、イネーブル ロジックおよび入力/出力ロジックをすべて同じ階層に含めます。ロジックが異なる階層にあり、階層間に KEEP_HIERARCHY または DONT_TOUCH 属性がある場合は、ツールでこれらのバッファを推論することはできません。

差動 I/O (IBUFDS、OBUFDS) やダブル データ レート レジスタ (IDDR、ODDR、ISERDES、OSERDES) などの I/O コンポーネントをインスタンス化する場合、最上位付近でインスタンス化する必要があります。コンポーネントをインスタンス化する場合、このコンポーネントのインスタンスを HDL ファイルに追加します。インスタンス化すると、コンポーネントの使用方法を完全に制御できるので、ロジックの使用方法が完全に既知になります。

クロック エLEMENT を最上位付近に挿入

クロック エLEMENT をできるだけ最上位の近くに挿入すると、モジュール間でクロックを共有しやすくなります。クロックの共有により必要なクロック リソースが削減され、リソース使用量、パフォーマンス、および消費電力が改善する可能性があります。

クロックが作成されるモジュール以外では、クロック パスはモジュールのみを駆動する必要があります。最上位からダウンストリームのモジュールを介して最上位にまたがるようなパスは、VHDL シミュレーションでデバッグが困難で時間がかかるデルタ サイクル問題の原因となります。

データパスの論理境界にレジスタを付ける

階層境界の出力にレジスタを付けて、クリティカル パスが 1 つのモジュールまたは境界内に含まれるようにします。階層境界の入力にもレジスタを付けることを考慮してください。タイミング パスが 1 つのモジュール内にあると、複数のモジュール間をまたいでいるよりも解析および修正が簡単です。階層境界でレジスタが付いていないパスはすべて階層を再構築 (rebuild) またはフラット化 (flat) するオプションを使用して合成し、階層間で最適化が実行されるようにする必要があります。データパスの論理境界にレジスタを付けると、階層間の最適化が最小限に抑えられ、ロジックがモジュール間で移動しないので、デザイン プロセス全体でデバッグの際にトレースしやすくなります。

フロアプランに関する考慮事項

フロアプランでは、デザイン ネットリストの特定部分に属するセルをデバイスの特定箇所に配置できます。手動フロアプランを使用すると、次のようになります。

- SSI テクノロジ デバイスを使用する場合に、ロジックを分割して特定の SLR に配置。
- 標準フローではタイミングが満たされないデザインのタイミング クロージャを達成。

セルが 1 つの階層レベルに含まれない場合は、すべてのオブジェクトを個別にフロアプラン制約に含める必要があります。合成によりこれらのオブジェクト名が変更される場合は、制約をアップデートする必要があります。フロアプランを 1 つの階層レベル内に収めると、必要な制約は 1 行のみになるので、推奨されます。

フロアプランは常に必要なわけではありません。必要な場合にのみフロアプランしてください。

フロアプランの詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906) のこのセクションを参照してください。



推奨: Vivado ツールでは階層間をまたぐフロアプランが可能です。さらにメンテナンスが必要となります。可能な限り、階層間をまたぐフロアプランは避けてください。

ファンクションおよびタイミング デバッグ用の階層の最適化

このセクションの前半で説明したように、クリティカル パスが同じ階層境界内に含まれるようにしておくと、タイミングをデバッグし、タイミング要件を満たしやすくなります。同様に、ファンクション デバッグ (および修正) のためにも、関連する信号は同じ階層に含めておく必要があります。関連の信号を 1 つの階層レベルに含めると、合成で最適化される信号名を見つけやすくなり、関連信号を比較的簡単にプローブおよび修正できるようになります。

モジュール レベルの属性の適用

モジュール レベルで属性を適用すると、コードがシンプルになり、拡張性も増します。属性を信号レベルではなくモジュール レベルで適用すると、現在の階層で宣言された信号すべてにその属性が伝搬されます。属性をモジュール レベルで適用すると、グローバル合成オプションを無効にできます。



注意: ほかの属性とは異なり、DONT_TOUCH はモジュールからモジュール内のすべての信号には伝搬されません。詳細は、『Vivado Design Suite ユーザー ガイド: 合成』(UG901) の[このセクション](#)を参照してください。

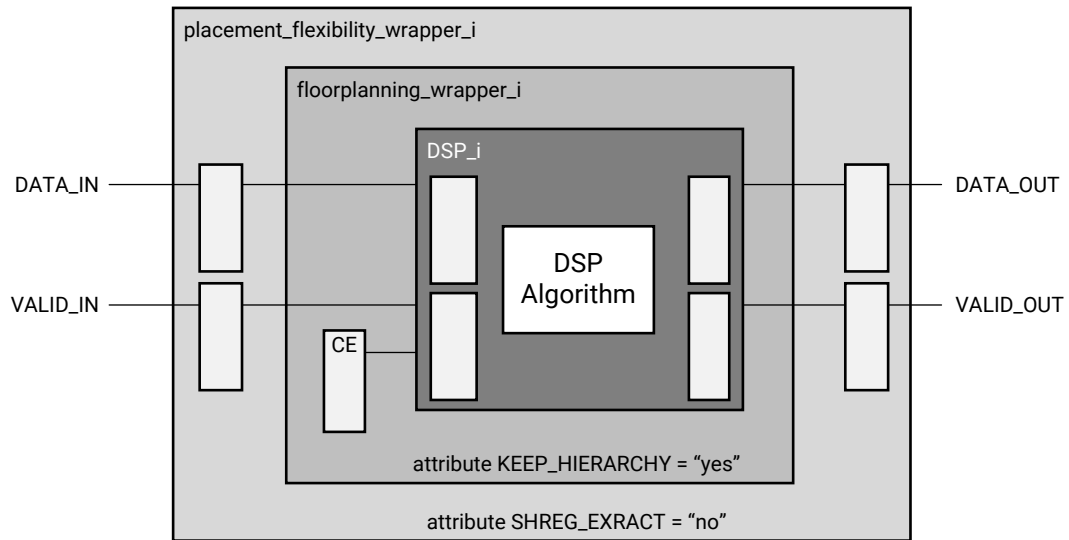
アドバンス設計手法のための階層の最適化

ボトムアップ合成、Dynamic Function eXchange (DFX)、アウト オブ コンテキスト (OOC) デザインなどのアドバンス設計手法を使用する場合、階層レベルでのプランニングが必要となります。デザインには、使用される手法に適した階層レベルを選択する必要があります。これらの手法については、この資料では説明されていません。詳細は、『Vivado Design Suite ユーザー ガイド: 階層デザイン』(UG905) の[このセクション](#)を参照してください。

高速 DSP デザインでの階層プランニング例

次の例はすべてのデザインに適用できるわけではありませんが、階層を使用して何が達成できるかを示しています。DSP デザインでは、通常デザインにレイテンシを追加できるので、レジスタを追加してパフォーマンスを最適化できます。レジスタは、配置の柔軟性を上げるためにも使用できます。高クロック周波数では 1 クロック サイクルでダイを通過することは不可能なので、これは重要です。レジスタを追加することで、到達しにくいエリアが使用されるようになります。次の図に、効果的な階層プランニングにより、タイミング クロージャにかかる時間を短縮できることを示します。

図 16: 効果的な階層プランニングの例



X13500-122019

デザインのこの部分には、次の 3 つのレベルがあります。

- DSP_i

DSP_i アルゴリズム ブロックには、入力と出力の両方にレジスタが付いています。デバイスにはレジスタが多数含まれるので、この方法を使用してタイミング バジエツトを改善することをお勧めします。

- floorplanning_wrapper_i

floorplanning_wrapper_i には、CE 信号が含まれます。CE 信号は通常負荷が高く、タイミングを満たすのが困難なので、フロアプランに含めておく必要があります。フロアプラン ラッパーを作成しておく、このモジュールを後で必要に応じて手動でフロアプランできます。

また、モジュール レベルに KEEP_HIERARCHY を追加しておく、その他のグローバル合成オプションに関係なく、階層がフロアプラン用に保持されます。

- placement_flexibility_wrapper_i

placement_flexibility_wrapper_i では、DATA_IN、VALID_IN、DATA_OUT、および VALID_OUT 信号にレジスタが付いています。これらの信号はフロアプランの一部になるようには意図されていないので、floorplanning_wrapper_i 外にあります。フロアプランに含めると、配置の柔軟性に必要な条件が満たされません。

また、DATA_IN と VALID_IN または DATA_OUT と VALID_OUT がペアとして処理されていれば、後でさらにレジスタを追加できます。さらにレジスタを追加すると、合成ツールでシフト レジスタ LUT (SRL) が推論され、すべてのレジスタが強制的に 1 つのコンポーネントに配置され、配置に柔軟性がなくなる可能性があります。これを回避するには、モジュール レベルに SHREG_EXTRACT を追加して NO に設定します。

IP (Intellectual Property) の使用

検証済みの IP (Intellectual Property) コアを使用すると設計および検証にかかる時間を大幅に削減できるので、タイムトゥ マーケットが短縮されます。IP を使用した設計に関する詳細は、次を参照してください。

- 『Vivado Design Suite ユーザー ガイド: IP を使用した設計』 (UG896)

- 『Vivado Design Suite ユーザー ガイド: IP インテグレーターを使用した IP サブシステムの設計』 (UG994)
- [Vivado Design Suite QuickTake ビデオ: Vivado での再利用可能な IP のコンフィギュレーションおよび管理](#)

IP 要件のプランニング

IP 要件のプランニングは、どの新規プロジェクトでも最も重要な段階の 1 つです。

- 必要な機能およびその他の設計目標に対して、サイリンクスまたはサードパーティ パートナーから提供されている IP オプションを評価してください。次に例を示します。
 - 提供されている IP コアよりもカスタム ロジックの方が適しているか。
 - 複数プロジェクトで再利用するために、カスタム デザインを業界標準フォーマットでパッケージ化する価値はあるか。
- メモリ、ネットワーク、およびペリフェラルなどの必要なインターフェイスを考慮してください。



重要: ツールで IP 特定の制約が適切に処理されるようにするため、プロジェクトに .xci または .xcix IP ソース ファイルを追加してください。IP を使用する場合は、IP で生成された出力 DCP ファイルを使用しないでください。

AMBA AXI

サイリンクスでは、オープン AMBA® 4 AXI4 インターコネクト プロトコルに基づいて規格化された IP インターフェイスを提供しています。この規格化により、サイリンクスおよびサードパーティ プロバイダーから提供されている IP を簡単に統合でき、最高のシステム パフォーマンスを達成できます。サイリンクスは、デバイス アーキテクチャに効率的にマップできるように、Arm® と協力して AXI4、AXI4-Lite、および AXI4-Stream を定義しました。

AXI4 は、高パフォーマンス、高クロック周波数システムのデザインをターゲットにしており、高速インターコネクトに適しています。AXI4-Lite は、AXI4 の軽量バージョンで、主に制御レジスタおよびステータス レジスタへのアクセスに使用されます。

AXI4-Stream は、マスターからスレーブへの一方向のデータ ストリーミングに使用されます。通常は、DSP、ビデオおよび通信アプリケーションに使用されます。

Vivado Design Suite の IP カタログ

IP カタログは、サイリンクスが提供する IP にアクセスするためのリポジトリで、エンベデッド システム、DSP、通信、インターフェイスなどの IP コアが含まれます。

IP カタログからは、使用可能な IP コアを検索し、その製品ガイド、変更ログ、製品ウェブ ページ、IP のアンサーなどを確認できます。

IP カタログのコアは、GUI または Tcl シェルを使用してアクセスおよびカスタマイズできます。Tcl スクリプトを使用すると、IP コアのカスタマイズを自動化できます。

カスタム IP

サイリンクスでは、業界標準の IP-XACT 形式を使用して IP を提供しており、カスタム IP をパッケージ化する IP パッケージャーというツールを提供しています。また、カスタマイズした IP をカタログに追加して、チームまたは企業内で共有可能な IP リポジトリを作成できます。サードパーティ プロバイダーからの IP も、既に IP-XACT フォーマットであっても、IP パッケージャーによりパッケージ化されれば、このカタログに追加できます。

IP カタログでの IP の選択

ザイリンクスおよびサードパーティ ベンダーの IP は、通信およびネットワーク、ビデオおよび画像処理、オートモーティブおよびインダストリアルなど、用途に基づくカテゴリに分類されています。このカテゴリを使用して、該当する分野で使用可能な IP を探すことができます。

IP カタログの IP のほとんどは無償ですが、有償でライセンスが必要なものもあります。IP を購入する必要があるかどうか、およびライセンスのステータスは、IP カタログに表示されます。IP カタログから IP を選択する際は、デザイン要件および IP が提供する機能に基づいて、次の事項を考慮してください。

- この IP に必要なシリコン リソース (該当する IP の製品ガイドを参照)
- 考慮しているデバイスおよびスピード グレードでこの IP がサポートされているかどうか (選択した IP によって指定するスピード グレードが異なる)、サポートされる場合、達成可能な最大スループットおよび Fmax はどれくらいか。
- デザインがボード上のコンパニオン チップと通信するために必要な外部インターフェイス規格
 - イーサネット、PCIe® インターフェイスなどの業界標準インターフェイス。
 - メモリ インターフェイス: サイズおよびパフォーマンスを含むメモリ インターフェイスの数。
 - Aurora などのザイリンクスのインターフェイス。

注記: 独自のカスタム インターフェイスを設計することもできます。
- IP でサポートされるオンチップ バス プロトコル
- 残りのデザインとの通信に必要なオンチップ バス プロトコル。例:
 - AXI4
 - AXI4-Lite
 - AXI4-Stream
- 複数のプロトコルを使用する場合は、IP カタログからのインフラストラクチャ IP を使用して、ブリッジ IP コアを選択することが必要となる可能性があります。次に例を示します。
 - AXI-AHB ブリッジ
 - AXI-AXI インターコネクト
 - AXI-PCIe ブリッジ
 - AXI-PLB ブリッジ

IP のカスタマイズ

IP は、GUI または Tcl スクリプトを使用してカスタマイズできます。

関連情報

[カスタマイズ GUI の使用](#)

[Tcl スクリプトの使用](#)

カスタマイズ GUI の使用

IP の検出、調査、カスタマイズには、グラフィック インターフェイスを使用すると簡単です。各 IP は、一連のタブまたはページを使用してカスタマイズできるようになっており、関連する設定オプションがグループ化されています。次の図に、カスタマイズ ウィンドウの例を示します。IP をカスタマイズすると、その内容が XCI ファイルに含まれます。ここから、IP のさまざまな出力ファイルを作成できます。

Tcl スクリプトの使用

ほとんどすべての GUI 操作に対して Tcl コマンドが実行されます。カスタマイズ オプションすべての設定を含めた IP の作成を、ユーザーの操作なしで Tcl スクリプトで実行できます。

ただし、設定オプションの名前と設定可能な値を知っておく必要があります。通常は、まず GUI を使用してカスタマイズして、そこからスクリプトを作成します。結果の Tcl スクリプトができたなら、データ サイズの変更など、必要に応じてスクリプトを簡単に修正できます。

Tcl ベースでの IP の作成は、バージョン管理システムを使用する場合など、自動化する際に便利です。ソース管理およびリビジョン制御の詳細は、『Vivado Design Suite ユーザー ガイド: デザイン フローの概要』(UG892)の[このセクション](#)を参照してください。

IP のバージョンおよびリビジョン管理

IP をカスタマイズすると、選択したパラメーター値をすべて含む XCI ファイルが作成されます。Vivado IDE の各バージョンでサポートされる IP のバージョンはそれぞれ 1 つのみです。ザイリンクスでは、使用している Vivado IDE のバージョンで最新の IP バージョンを使用することをお勧めします。以前のバージョンの IP を使用する場合は、そのバージョン用の出力ファイルをすべて保存しておく必要があります。ソース管理およびリビジョン制御の詳細は、『Vivado Design Suite ユーザー ガイド: デザイン フローの概要』(UG892)の[このセクション](#)を参照してください。



重要: 7 シリーズ デバイスのメモリ IP では、XCI ファイルに加えて PRJ ファイルが作成されます。7 シリーズ メモリ IP にリビジョン管理を使用する場合は、PRJ ファイルを XCI ファイルと同じディレクトリに保持してください。

RTL コーディング ガイドライン

カスタム RTL を作成することにより、グルー ロジック ファンクションや、適切な IP がない機能をインプリメントできます。最適な結果を得るには、このセクションのコーディング ガイドラインに従ってください。追加のガイドラインは、『Vivado Design Suite ユーザー ガイド: 合成』(UG901)の[このセクション](#)を参照してください。

Vivado Design Suite HDL テンプレートの使用

RTL を作成またはザイリンクス プリミティブをインスタンス化するには、Vivado Design Suite 言語テンプレートを使用してください。言語テンプレートには、ザイリンクス デバイス アーキテクチャに適切な推論が実行されるようにするために推奨されるコード構文が含まれます。言語テンプレートを使用すると、設計プロセスが簡単になり、結果が向上します。Vivado IDE で言語テンプレートを開くには、Flow Navigator で [Language Templates] をクリックし、必要なテンプレートを選択します。

制御信号および制御セット

制御セットとは、SRL、LUTRAM、またはレジスタを駆動する制御信号 (セット/リセット、クロック イネーブル、クロック) をまとめたもので、制御信号の固有の組み合わせに対して固有の制御セットが作成されます。7 シリーズ スライス内ではすべてのレジスタで制御信号が共有され、同じスライスにパックできるのは共通の制御セットを使用するレジスタのみなので、この概念は重要です。たとえば、ある制御セットを持つレジスタのロードが 1 つのレジスタのみの場合、そのレジスタが含まれるスライスにある残り 7 個のレジスタが使用できなくなります。

デザインに含まれる固有の制御セットの数が多すぎると、リソースが無駄に使用されたり、配置オプションが少なくなったりすることがあり、消費電力が増加してパフォーマンスが低下します。制御セットを少なくすると、配置のオプションが多くなり、柔軟性も向上するので、通常結果は改善します。

UltraScale™ デバイスでは、CLB 内での制御セットのマップの柔軟性は高くなります。駆動されないリセットはスライス内でローカルに接続されるので、制御セットには含まれませんが、ロジックのグループを最大限に柔軟に配置できるようにするため、固有の制御セットの数を制限することをお勧めします。

リセット

リセットは、最もよく使用される重要な制御信号の 1 つです。リセットは、デザインのパフォーマンス、エリア、および消費電力に大きく影響します。

推論された同期コードには、次のようなリソースが使用される可能性があります。

- LUT
- レジスタ
- SRL
- ブロックまたは LUT メモリ
- DSP48 レジスタ

リセットの選択および使用がこれらのコンポーネントの選択にも影響し、デザインに対して最適化リソースが使用されないことがあります。配列にリセットを誤って配置すると、1 つのブロック RAM が推論される代わりに、何千個ものレジスタが推論されてしまう可能性があります。

乗算器の入力または出力に非同期リセットを記述すると、レジスタが DSP ブロックではなく、スライスに配置されることがあります。このような状況では追加のリソースが使用され、消費電力とデザイン パフォーマンスに悪影響を与える可能性があります。

リセットを使用する状況と使用する場所

ザイリンクス デバイスには、専用のグローバル セット/リセット信号 (GSR) が含まれます。この信号は、デバイス コンフィギュレーションの最後にハードウェアに含まれるすべてのシーケンシャル セルを初期値に設定します。

初期ステートが指定されていない場合は、シーケンシャル プリミティブはデフォルト値になります。ほとんどの場合、デフォルト値は 0 です。ただし、FDSE および FDPE プリミティブは例外で、デフォルト値は 1 です。各レジスタは、コンフィギュレーション後に既知のステートになります。そのため、電源投入時にデバイスを初期化する目的のためだけにグローバル リセットのコードを記述する必要はありません。

ザイリンクスでは、デザインがリセットを必要とする状況と必要としない状況を判断する際は、十分に注意することをお勧めします。適切に動作させるために制御バス ロジックにリセットが必要な場合がありますが、データバス ロジックでは通常リセットはそれほど必要ではありません。リセットの使用を抑えると、次のようになります。

- リセット ネットの全体的なファンアウトが制限される。

- リセットの配線に必要なインターコネクトの量が削減される。
- リセット パスのタイミングが簡潔になる。
- パフォーマンス、エリア、消費電力が全体的に改善することが多い。



推奨: 各同期ブロックを評価して、最適な動作にリセットが必要かどうかを判断してください。リセットが実際に必要かどうかを確認せずにデフォルトで記述しないでください。

リセットが必要かどうかは、論理シミュレーションで簡単に判断できます。

リセットを記述しないロジックでは、ロジックをマップするデバイス リソースを選択する際の柔軟性が増します。

たとえば次を考慮することにより、合成ツールでそのコードに最適なリソースが選択され、結果が改善することがあります。

- 必要な機能
- パフォーマンス要件
- 使用可能なデバイス リソース
- 消費電力

同期リセット vs 非同期リセット

ザイリンクスでは、リセットが必要な場合は、同期リセットを使用することをお勧めします。同期リセットには、非同期リセットと比較して次の利点があります。

- 同期リセットを使用すると、デバイス アーキテクチャのより多くのリソース エLEMENTに直接マップできます。
- 非同期リセットは、一般的なロジック構造のパフォーマンスに影響します。ザイリンクス デバイスの汎用レジスタでは、セット/リセットを非同期または同期のいずれかにプログラムできるので、非同期リセットを使用しても問題ないと思われがちですが、グローバル非同期リセットを使用すると、制御セットの数は増加しませんが、このリセット信号をすべてのレジスタ エLEMENTに配線する必要があるため、配線はより複雑になります。
- 非同期リセットを使用すると、リセットをアサートしたときに、ブロック RAM、LUTRAM、および SRL のメモリの内容が破損する可能性が高くなります。これは特に、ブロック RAM、LUTRAM、および SRL の入力ピンを駆動する非同期リセットを持つレジスタに当てはまります。
- 集積度の高い配置や精度の高い配置が必要な場合は、同期リセットを使用した方が、制御セットを柔軟にマップし直すことができます。最適に配置されたスライス内で互換性のないリセットが検出された場合、同期リセットであればレジスタのデータパスにマップし直すことができます。これにより、配線リソースの使用率が下がり、配置の集積度が増加して、適切なフィットおよびパフォーマンスの向上が可能になります。
- DSP48 およびブロック RAM のような一部のリソースには、ブロック内のレジスタ エLEMENTに同期リセットしかありません。これらのエLEMENTに関連するレジスタ エLEMENTに非同期リセットが使用されると、これらのレジスタがこれらのブロックに直接推論されない可能性があります。

また、次の点にも注意してください。

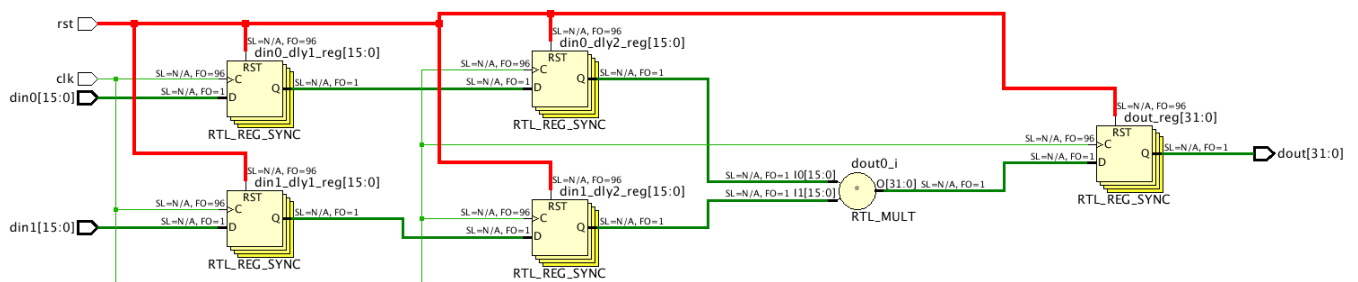
- クロックは、同期リセットの小さなリセット グリッチのフィルターとして機能しますが、これらのグリッチがアクティブ クロック エッジ付近で発生すると、フリップフロップがメタステーブル状態になることがあります。
- 同期リセットは、リセット信号のパルスがクロックのアクティブ エッジ中に存在するようにするため、パルス幅を十分に広くする必要がある場合があります。

- 非同期リセットを使用する場合は、非同期リセットのディASSERTを同期するようにしてください。クロックとリセットの相対的なタイミングは、リセットのアサート時には無視できますが、リセットの解放はクロックに同期させる必要があります。リセットの解放を同期させないと、メタステーブル状態になる可能性があります。リセットの解放時は、レジスタのクロックピンに対するリセットピンのセットアップおよびホールド条件を満たす必要があります。非同期リセット (リセット リカバリおよびリムーバルのタイミングなど) のセットアップおよびホールド条件に違反すると、フリップフロップがメタステーブル状態になり、不明な状態になったことが原因でデザイン エラーが発生する可能性があります。この状況は、フリップフロップ データ ピンのセットアップおよびホールド条件の違反に類似しています。

リセットのコード例: 同期リセットを持つ乗算器

既存の DSP プリミティブの機能を活用する方法を示すため、次に同期リセットを持つ乗算器の例を示します。

図 17: 同期リセットを使用するパイプライン レジスタを含む乗算器



この回路では、DSP48 プリミティブが推論され、すべてのパイプライン レジスタが DSP48 プリミティブ内にパックされます (AREG/BREG=1、MREG=1、PREG=1)。

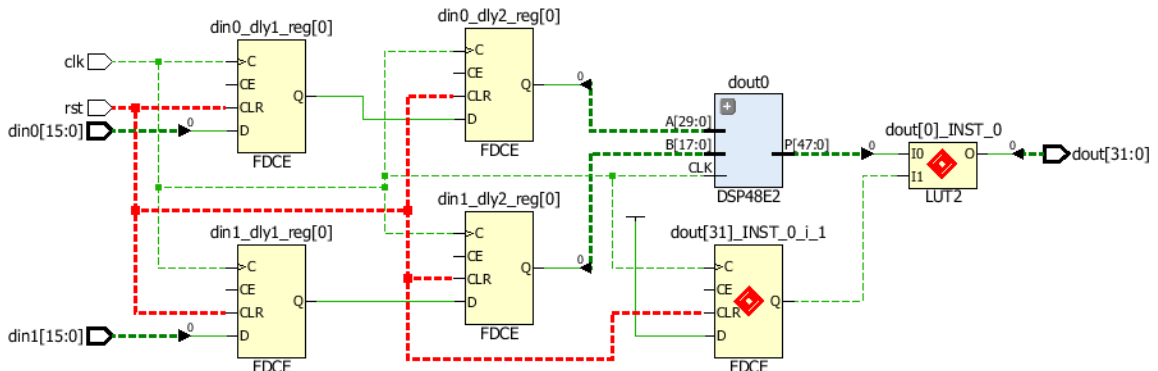
このコード例には、次の利点があります。

- リソース使用量が最適
- パフォーマンスが高く、消費電力が少ない
- 終点の数が少ない

リセットのコード例: 非同期リセットを持つ乗算器

次の例では、専用 DSP リソースをターゲットとするロジックに同期リセットを使用するレジスタを使用することが重要であることを示します。次の図に、非同期リセットを使用したパイプライン レジスタを含む 16x16 ビットの DSP48 ベース乗算器を示します。この場合、入力段と、外部レジスタおよび 32 個の LUT2 (赤色のマーカーで示す) に通常のファブリック レジスタを使用し、DSP 出力の非同期リセットをエミュレートする必要があります (DSP48 の P レジスタはイネーブルだがリセットには接続されていない)。これにより 65 個のレジスタと 32 個の LUT が余分に使用され、DSP48 の設定が AREG/BREG=0、MREG=0、PREG=1 になります。

図 18: 非同期リセットを使用するパイプライン レジスタを含む乗算器



次の図に示すようにリセット定義を変更するだけで、乗算器のパイプライン レジスタに同期リセットが使用され、DSP48 の内部レジスタが使用されます (AREG/BREG=1、MREG=1、PREG=1)。

図 19: 乗算器の非同期リセットを同期リセットに変更

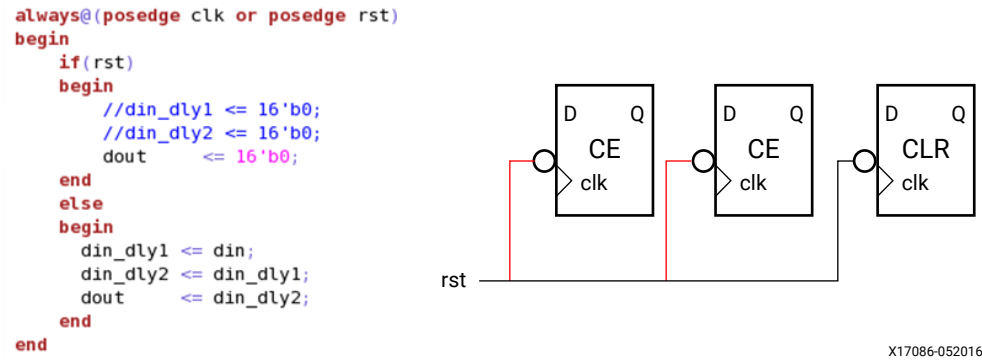
<pre> always @ (posedge clk or posedge rst) begin if (rst) begin din0_dly1 <= 16'h0; din0_dly2 <= 16'h0; din1_dly1 <= 16'h0; din1_dly2 <= 16'h0; dout <= 32'h0; end else begin din0_dly1 <= din0; din0_dly2 <= din0_dly1; din1_dly1 <= din1; din1_dly2 <= din1_dly1; dout <= din0_dly2 * din1_dly2; end end </pre>		<pre> always @ (posedge clk) begin if (rst) begin din0_dly1 <= 16'h0; din0_dly2 <= 16'h0; din1_dly1 <= 16'h0; din1_dly2 <= 16'h0; dout <= 32'h0; end else begin din0_dly1 <= din0; din0_dly2 <= din0_dly1; din1_dly1 <= din1; din1_dly2 <= din1_dly1; dout <= din0_dly2 * din1_dly2; end end </pre>
--	--	---

ファブリック リソースが節約され、すべての DSP48 の内部レジスタが利用されるので、デザイン パフォーマンスおよび電力効率が最適なものになります。

HDL コードでリセットを削除する際の問題

コードを最適化してリセットを削除する場合、リセット宣言で条件をコメントアウトするだけでは適切な構造は作成されず、問題が発生します。たとえば、次の図に示すように、3 段のパイプラインでそれぞれに非同期リセットが使用されているとします。リセット条件のコードをコメントアウトしてパイプラインの 2 段でリセット条件を削除しようとする、非同期リセットがイネーブルになります (rst の反転ロジック)。

図 20: リセット条件を含むコードをコメントアウト



リセットを削除するには、次の図に示すように、リセット条件用の順序ロジック プロシージャとリセット以外の条件用の順序ロジック プロシージャを作成するのが最適な方法です。

図 21: リセットを含むレジスタとリセットを含まないレジスタのプロシージャ文

```

always@(posedge clk)
begin
  din_dly1 <= din;
  din_dly2 <= din_dly1;
end

always@(posedge clk or posedge rst)
begin
  if(rst)
    dout <= 16'd0;
  else
    dout <= din_dly2;
end

```



ヒント: リセットを使用する場合は、プロシージャ文のすべてのレジスタがリセットであることを確認します。

クロック イネーブル

クロック イネーブルは、適切に使用すると、エリアやパフォーマンスにほとんど影響を与えることなくデザインの消費電力を削減できますが、間違って使用すると、次のような問題が発生することがあります。

- リソース使用率の増加
- 配置集積度の減少
- 消費電力の増加
- パフォーマンスの悪化

ほとんどの場合、制御セット数が増える主な原因はファンアウトの小さいクロック イネーブルです。

クロック イネーブルの作成

クロック イネーブルは、不完全な条件文が同期ブロックに記述されると作成されます。クロック イネーブルは、前の条件が満たされない場合に、最後の値を保持するように推論されます。これが必要な機能であればこのようにコードを記述しても問題ありませんが、前の条件値が満たされない場合に出力がドントケアとなることがあります。このような場合、サイリックスでは、定義された定数 (信号に 1 か 0 を代入) を使用して、条件を閉じる (`else` 文を使用) ことをお勧めします。

ほとんどのインプリメンテーションでは、これによってロジックが追加されることはなく、クロック イネーブルも不要ですが、幅の広いバスでは値が保持されるクロック イネーブルを推論すると、消費電力が削減されることがあります。基本的には、推論されるレジスタの数が少ないときにはクロック イネーブルを使用すると制御セット数が増加するので有害となる可能性があります。レジスタ数が多い場合は利点が多いので、クロック イネーブルの使用をお勧めします。

リセットとクロック イネーブルの優先順位

サイリックス デバイスでは、非同期または同期セット/リセットのどちらが記述されていても、セット/リセットがクロック イネーブルよりも優先されるように全レジスタが構築されています。サイリックスでは、最適な結果を得るため、同期ブロック内の `if/else` 文で常にクロック イネーブル (必要と判断した場合) よりも前にセット/リセットを記述することをお勧めします。クロック イネーブルを最初に記述すると、リセットがデータパスに接続され、追加ロジックが作成されます。

関連情報

[クロッキング ガイドライン](#)

合成属性を使用したイネーブル/リセットの抽出の制御

`DIRECT_RESET`、`DIRECT_ENABLE`、`EXTRACT_RESET`、`EXTRACT_ENABLE` 属性を使用すると、指定の構造における制御セットのマッピングを制御できます。

デザインに同期リセット/イネーブルが含まれていると、ロード数が `-control_set_opt_threshold` 合成オプションで設定されたしきい値以上である場合は CE/R/S ピンを介してマッピングされたロジック コーンが作成され、しきい値未満である場合は D ピンを介してマッピングされたロジック コーンが作成されます。デフォルトのしきい値は次のとおりです。

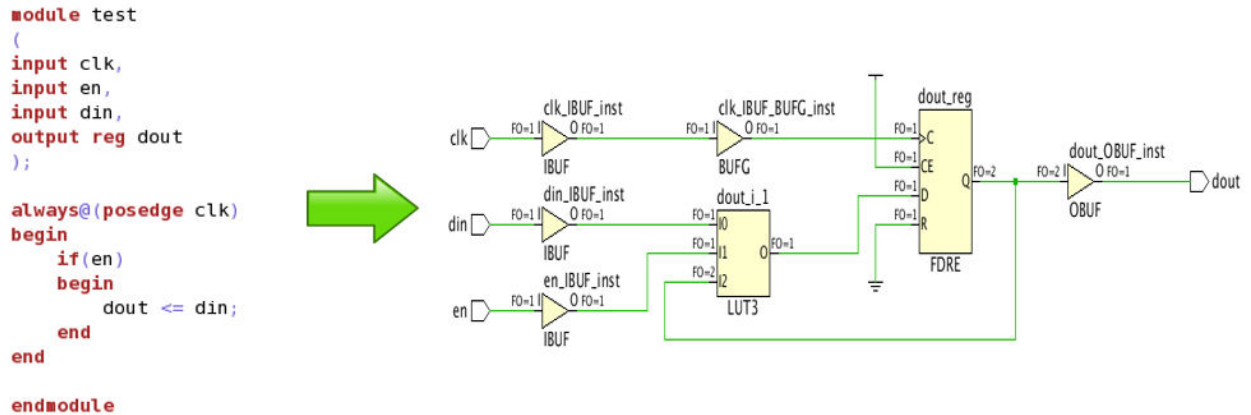
- 7 シリーズ デバイス: 4
- UltraScale デバイス: 2

`DIRECT_ENABLE` および `DIRECT_RESET` の使用

制御セットのマッピングを使用するには、イネーブル/リセット信号に接続されているネットに属性を適用できます。これにより、CE/R ピンが使用されます。

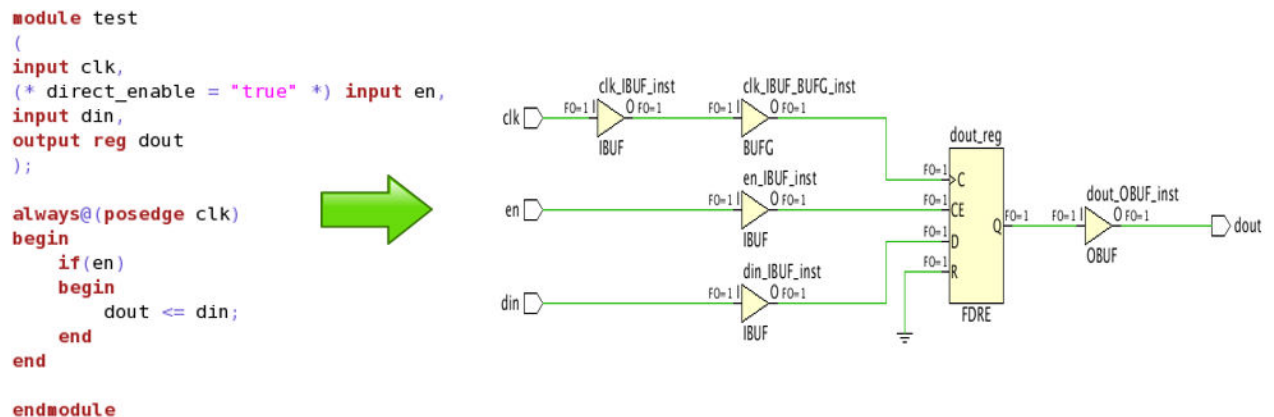
次の図では、イネーブル信号 (`en`) は 1 つのフリップフロップのみに接続されているので、合成エンジンにより `en` 信号がロジックの `FDRE/D` ピン コーンに接続されています。CE ピンは 1 に接続されます。

図 22: データパス ロジックを使用したクロック イネーブルのインプリメンテーション



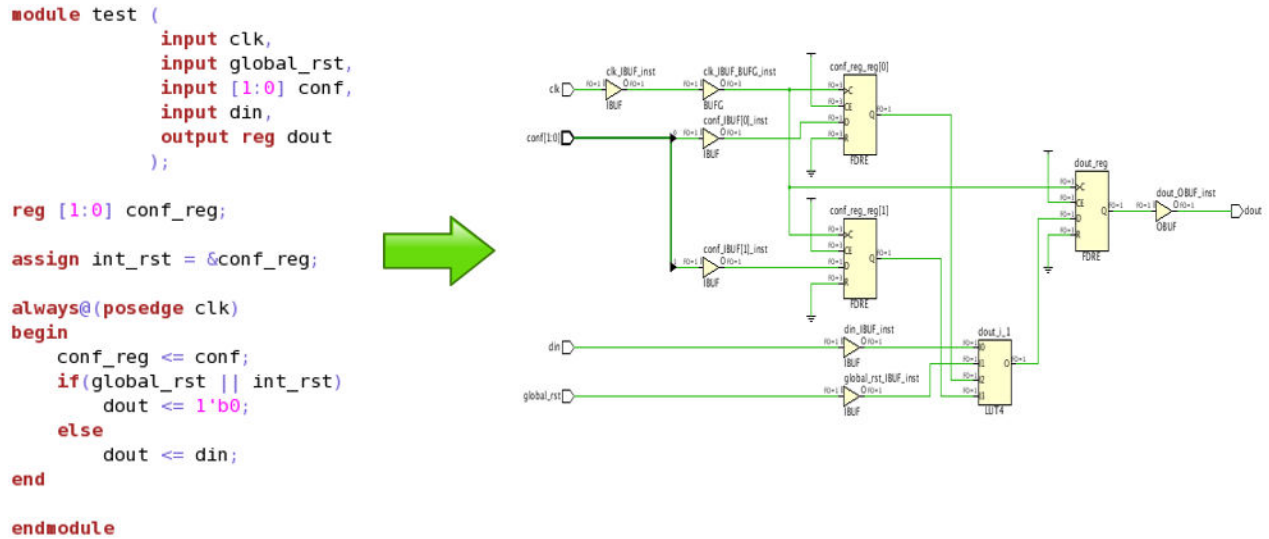
このデフォルトの動作を変更するには、DIRECT_ENABLE 属性を使用します。たとえば次の図では、DIRECT_ENABLE 属性をポート/信号に追加することにより、イネーブル信号 (en) をレジスタの CE ピンに接続する方法を示しています。

図 23: direct_enable を使用した専用クロック イネーブルのインプリメンテーション



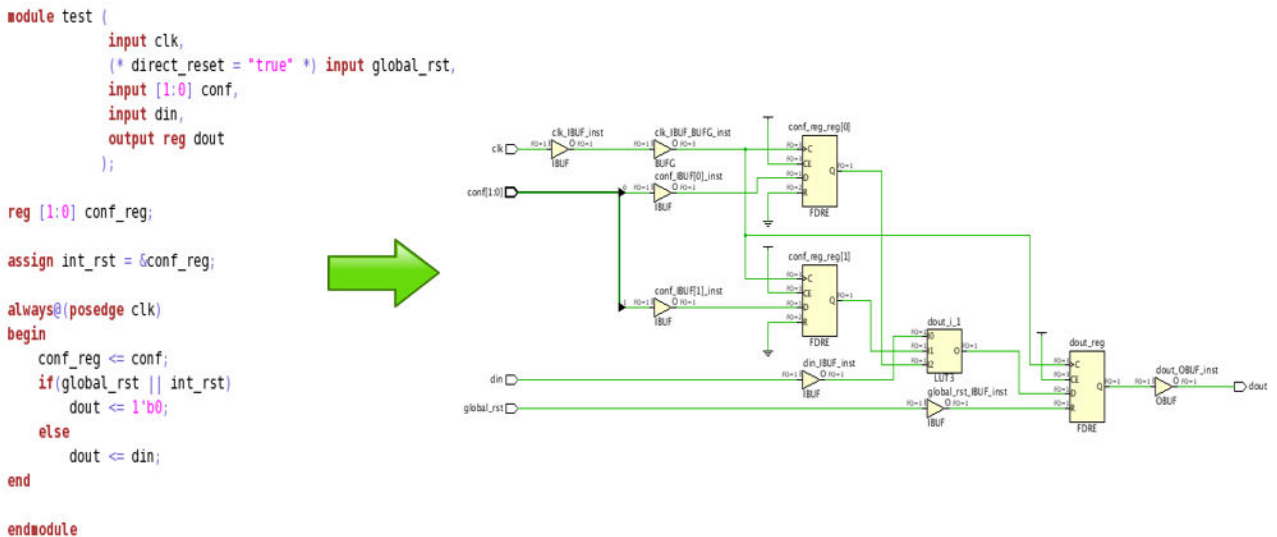
次の図に、レジスタを global_rst または int_rst のどちらでもリセット可能な RTL コードを示します。デフォルトでは、両方ともロジックのリセット ピンのコーンにマップされます。

図 24: データパス ロジックを介してマップされた複数のリセット条件



DIRECT_RESET 属性を使用すると、どのリセット信号をレジスタのリセット ピンに接続するかを指定できます。たとえば次の図では、DIRECT_RESET 属性を使用して global_rst 信号のみをレジスタ FDRE/R ピンに接続し、int_rst 信号はロジックの FDRE/D コーンに接続しています。

図 25: DIRECT_RESET 属性を使用した専用リセット ピンへの接続



ロジックを制御ピンからデータ ピンに移動

クリティカル パスの解析で、複数のパスが制御ピンで終了していることが検出される場合があります。これらのパスを解析し、ロジック段数の増加などのペナルティを発生させずに、ロジックをデータパスに移動できるかを調べる必要があります。最後の LUT の出力からフリップフロップの D 入力に直接接続があるので、ロジック段数が同じであれば、D ピンへのパスの遅延は CE/R/S ピンへのパスより小さくなります。次に、ロジックを制御ピンからレジスタのデータ ピンに移動する方法を示します。

次の例では、dout_reg[0] のイネーブル ピンのロジック段数は 2 であり、データ ピンのロジック段数は 0 です。この場合、RTL ファイルの dout レジスタ定義で EXTRACT_ENABLE 属性を "no" に設定してイネーブル ロジックを D ピンに移動することにより、タイミングを改善できます。

図 26: レジスタの制御ピン (イネーブル) で終了するクリティカル パス

```

module test
(
  input clk,
  input [9:0] en,
  input [7:0] din,
  output reg [7:0] dout
);

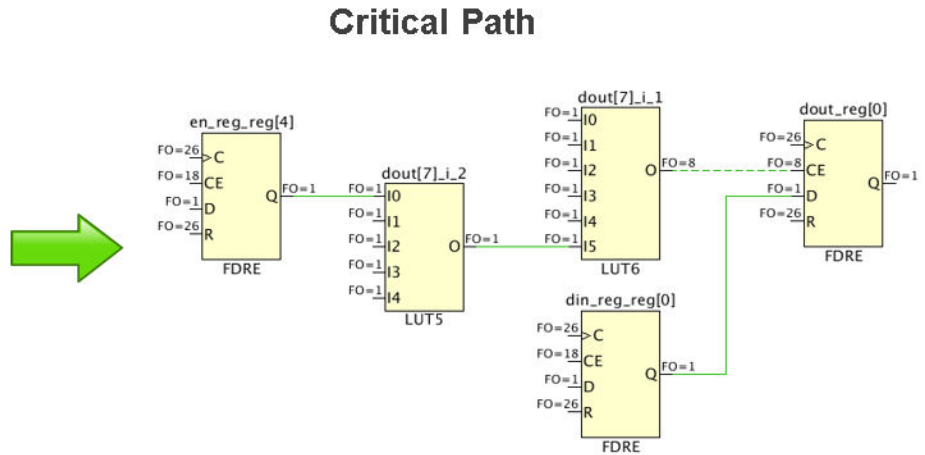
wire en_tmp;
reg [7:0] din_reg;
reg [9:0] en_reg;

assign en_tmp = &en_reg;

always@(posedge clk)
begin
  en_reg <= en;
  din_reg <= din;
  if(en_tmp)
    dout <= din_reg;
end

endmodule

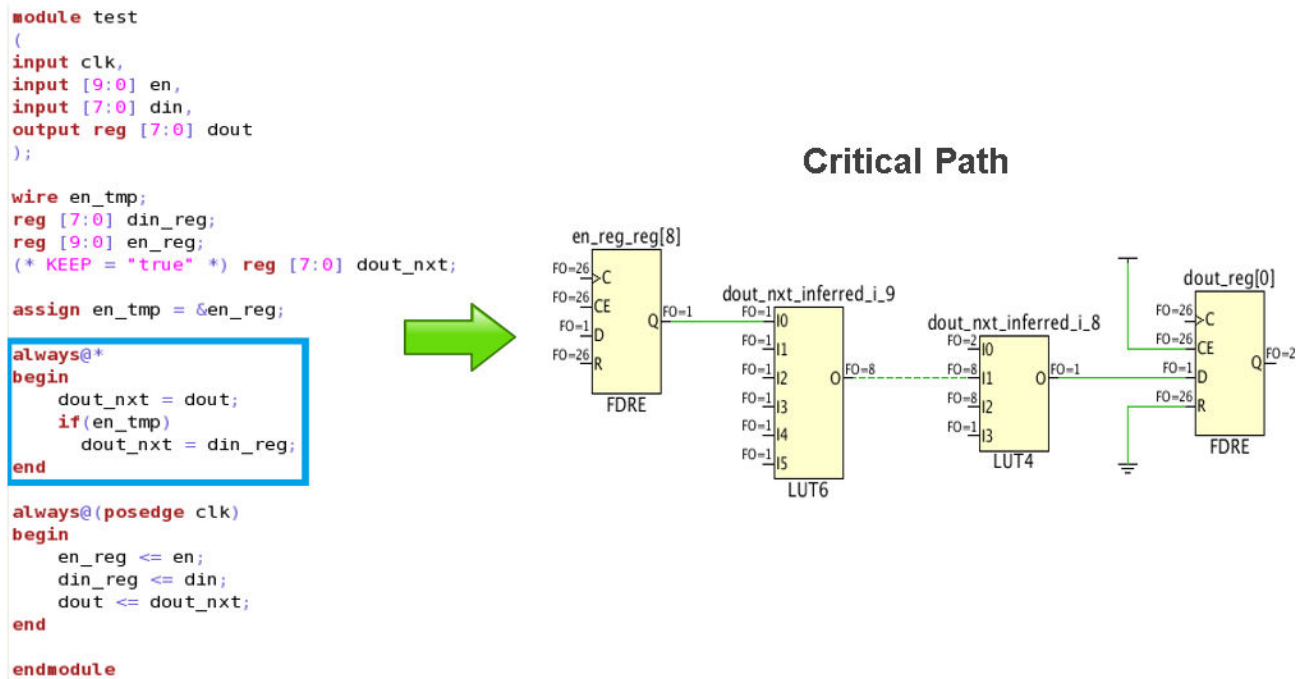
```



次の例では、組み合わせロジックと順序ロジックを分離し、ロジック全体をデータパスにマップしています。これによりロジックが D ピンに挿入され、ロジック段数は 2 のままになります。

同じ構造は、EXTRACT_ENABLE 属性を "no" に設定しても達成できます。EXTRACT_ENABLE 属性の詳細は、『Vivado Design Suite ユーザー ガイド: 合成』(UG901) を参照してください。

図 27: レジスタの制御ピンで終了するクリティカル パス (イネーブルの抽出をディスエーブル)



制御信号に関するヒント

- グローバル リセットが本当に必要かどうかを確認。
- 非同期制御信号は避ける。
- クロック、イネーブル、リセットの極性を統一する。
- セットおよびリセットを同じレジスタ エLEMENT に記述しない。
- 非同期リセットが絶対に必要な場合は、そのディアサート同期する。

推論の理解

コードは、最終的にデバイスに存在するリソースにマップされます。ターゲット アーキテクチャの主な演算、ストレージ、ロジック エレメントについて理解し、デザインの機能を記述しながらコードがマップされるハードウェア リソースを予測するようにします。コードがどのハードウェア リソースにマップされるかを理解しておくと、発生する可能性のある問題を早期に特定できます。

次の例に、ハードウェア リソースとマップを理解することが、どのようにデザインに関する決定を下すのに役立つかを示します。

- 4 ビットを超える加算、減算、加減算には、通常キャリー チェーンと 2 ビットの加算ごとに 1 つの LUT が使用されます (8 ビット X 8 ビット加算には 8 個の LUT とキャリー チェーンを使用)。三項加算または加算の結果がレジスタを介さず別の値と加算される場合、3 ビット加算ごとに 1 つの LUT が使用されます (8 ビット X 8 ビット X 8 ビット加算にも 8 個の LUT とキャリー チェーンを使用)。

複数の加算が必要な場合、加算 2 段ごとにレジスタを指定すると、3 項のインプリメンテーションを生成できるようになり、デバイス使用量が半分にになります。

- 通常、乗算は DSP ブロックをターゲットにします。18x25 未満 (UltraScale デバイスでは 18x27 未満) の符号付きビット幅は、1 つの DSP ブロックにマップされます。大きな積を必要とする乗算は、複数の DSP ブロックにマップされる可能性があります。DSP ブロックには、内部にパイプライン リソースが含まれます。

DSP ブロックに推論されるロジックが正しくパイプライン処理されていると、パフォーマンスおよび消費電力が大幅に改善されます。乗算を記述する際には、その周囲に 3 段のパイプラインを使用すると、最適なセットアップ、clock-to-out、および消費電力特性が生成されます。パイプライン段数が少ないと (1 段またはなし)、タイミング問題が発生し、これらのブロックの消費電力が増加する可能性があり、DSP 内のパイプライン レジスタは未使用のままになります。

- 深さが 16 ビット以下の 2 つの SRL は 1 つの LUT に、32 ビットまでの 1 つの SRL も 1 つの LUT にマップできます。
- 条件付きコードは、次のような標準マルチプレクサー コンポーネントになります。
 - 4:1 マルチプレクサーは 1 つの LUT にインプリメントでき、ロジック段数は 1 になります。
 - 8:1 マルチプレクサーは 2 つの LUT と MUXF7 コンポーネントにインプリメントでき、ロジック (LUT) 段数は 1 になります。
 - 16:1 マルチプレクサーは 4 つの LUT と MUXF7 および MUXF8 リソースを組み合わせたものにインプリメントでき、ロジック (LUT) 段数は 1 になります。

同じ CLB/スライス構造内の LUT、MUXF7、および MUXF8 を組み合わせて使用した場合、組み合わせ遅延はかなり小さくなり、ロジック段数 1 と同等と考えられます。これを理解しておく、リソースをより良く管理でき、データパスのロジック段数を理解および制御するのに役立ちます。

汎用ロジックでは、レジスタへの入力数を考慮します。その数から LUT とロジックの段数を見積もることができます。一般的には、入力が 6 個以下の場合、ロジック段数は 1 になります。理論上はロジック段数 2 で最大 36 個の入力を処理できますが、実際には最大約 20 個までの入力しか処理できないと考えてください。通常、入力の数が多いと、論理式が複雑になり、必要な LUT およびロジック段数が増加します。



重要: 使用可能なハードウェア リソースとそれらが効率的に使用されているかどうかをデザイン サイクルの早期に確認してください。その方が変更が簡単であり、デザイン サイクルの後の方でタイミング クロージャ中に変更しようとするよりも良い結果が得られます。

RAM および ROM の推論

RAM および ROM は複数の方法で指定できます。それぞれに利点と欠点があります。

推論

利点:

- 移植性が高い
- 読みやすく理解しやすい
- 自己文書化
- シミュレーションが高速

欠点:

- 可能な RAM 設定すべてを使用できない可能性あり
- 最適な結果にならない可能性あり

推論は通常良い結果が得られるので推奨される方法ですが、使用方法がサポートされない場合や、パフォーマンス、エリア、または消費電力が最適にならない場合などは例外です。このような場合は、別の方法を使用してください。

ザイリンクスでは、RAM を推論する場合は Vivado ツールに含まれる HDL テンプレートを使用することをお勧めします。前述したように、非同期リセットを使用すると RAM の推論に影響が出るので、使用しないようにしてください。

- ザイリンクス パラメーター指定マクロ (XPM)

利点:

- 。 ザイリンクス デバイス ファミリ間で移植可能
- 。 シミュレーションが高速
- 。 対称幅をサポート
- 。 予測可能な QoR

欠点:

- 。 サポートされる XPM オプションに制限される

XPM は、ユーザーが変更できない固定のテンプレートを使用することにより推論されます。そのため、QoR を確実にすることができ、標準の推論ではサポートできない機能をサポートできます。ザイリンクスでは、標準の推論で必要な機能がサポートされない場合は、XPM を使用することをお勧めします。

注記: `compile_simlib` を使用してシミュレーション ライブラリをコンパイルすると、XPM が自動的にコンパイルされます。詳細は、『Vivado Design Suite ユーザー ガイド: ロジック シミュレーション』 ([UG900](#)) を参照してください。

- RAM プリミティブの直接インスタンス化

利点:

- 。 インプリメンテーションを詳細に制御可能
- 。 ブロックのすべての機能にアクセス可能

欠点:

- 。 コードの移植性が低い
- 。 文字数が多く、機能および意図を理解しにくい

- IP カタログからのコア

利点:

- 。 複数コンポーネントを使用する場合により最適な結果が得られる
- 。 指定および設定がシンプル

欠点:

- 。 コードの移植性が低い
- 。 コアを管理する必要がある

関連情報

[Vivado Design Suite HDL テンプレートの使用](#)

RAM をインプリメントする際のパフォーマンスに関する考慮事項

メモリ エLEMENT を効率的に推論するには、パフォーマンスに影響する次の要素について考慮する必要があります。

- 専用ブロック RAM または分散 RAM の使用

RAM は、専用ブロック RAM 内、または分散 RAM を使用して LUT 内にインプリメントできます。この選択は、リソース選択に影響するだけでなく、パフォーマンスおよび消費電力にも大きく影響することがあります。

通常、RAM に必要なワード数が最初の基準となり、64 ビットのワード数までのメモリ配列は、通常 LUT RAM にインプリメントされます (32 ビット以下のワード数は LUT ごとに 2 ビットをマップ、64 ビットまでのワード数は LUT ごとに 1 ビットをマップ可能)。使用可能なリソースおよび合成ツールの割り当てによって、よりワード数の多い RAM も LUT RAM にインプリメントできます。

256 ビットよりもワード数の多いメモリ配列は、通常ブロック RAM メモリにインプリメントされます。ザイリンクス デバイスには柔軟性があるので、このような構造をさまざまな幅およびワード数の組み合わせでマップできます。コードの大容量メモリ配列宣言に使用されるブロック RAM の数および構造を理解するため、これらの構成を知っておく必要があります。

- 出力パイプライン レジスタの使用

出力レジスタの使用は優れたパフォーマンスのデザインには必須で、すべてのデザインに推奨されます。これにより、ブロック RAM の clock-to-out タイミングが改善されます。また、スライスの出力レジスタの clock-to-out タイミングはブロック RAM レジスタよりも高速なので、2 つ目の出力レジスタがあると有益です。両方のレジスタを使用すると、読み出しレイテンシの合計は 3 になります。これらのレジスタを推論する場合は、レジスタが RAM 配列と同じ階層レベルにある必要があります。これにより、ツールでブロック RAM 出力レジスタがプリミティブに統合されるようになります。

- 入力パイプライン レジスタの使用

RAM 配列が大きく、多数のプリミティブにマップされる場合、ダイのかなりのエリアに広がる可能性があり、これがアドレスおよび制御ラインでのパフォーマンス問題につながる可能性があります。これらの信号の生成後、RAM の前にレジスタを追加することを考慮してください。タイミングをさらに向上させるため、フローの後の方で `phys_opt_design` を使用してこのレジスタを複製します。入力にロジックのないレジスタの方が簡単に複製できます。

ブロック RAM 出力レジスタが推論されない状況

ザイリンクスでは、メモリと出力レジスタをすべて 1 つの階層レベルに推論することをお勧めします。これが、意図したとおりに推論されるようにする最も簡単な方法です。ブロック RAM 出力レジスタが推論されない状況が 2 つあります。1 つ目は出力に余分なレジスタが存在する状況、2 つ目は読み出しアドレス レジスタがメモリ配列の周囲でリタイミングされる状況です。これは、シングル ポート RAM を使用している場合にのみ発生します。これを次の図に示します。

図 28: ブロック RAM 出力レジスタの推論を妨げる余分な読み出しレジスタを含む RAM

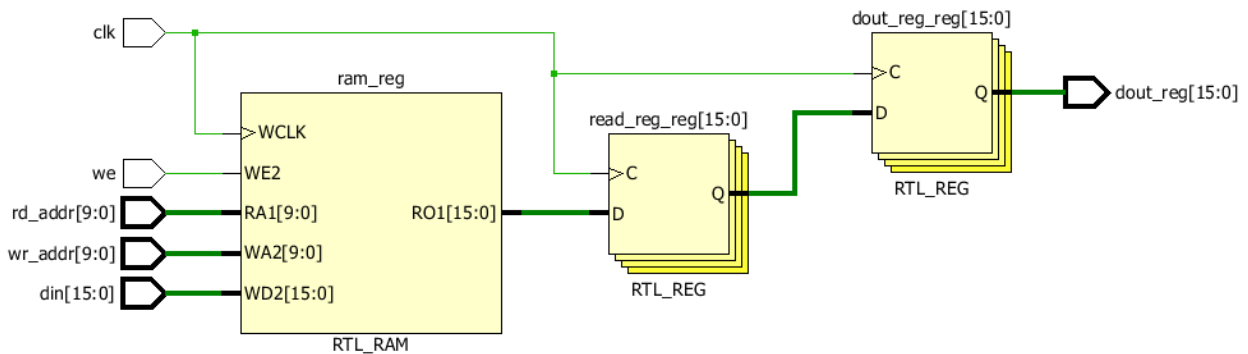
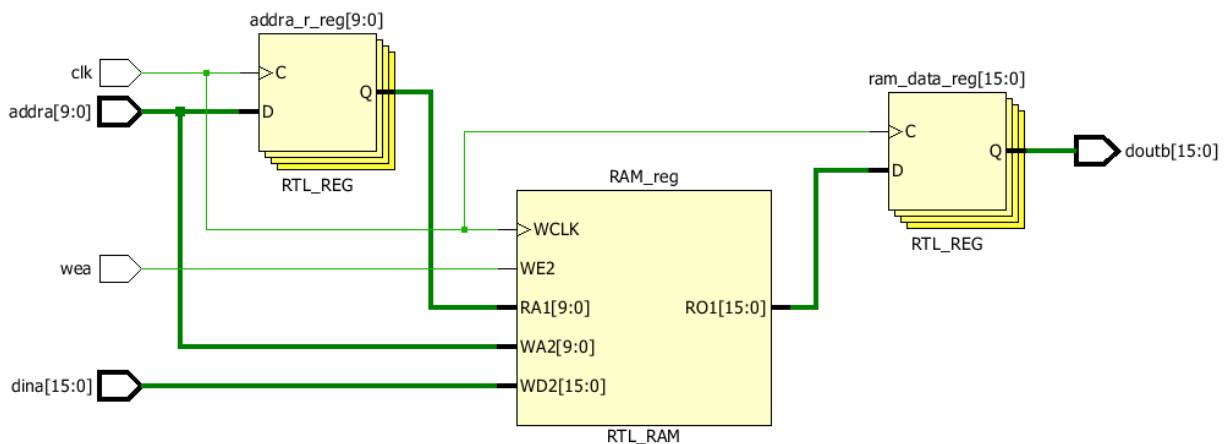


図 29: アドレスレジスタのリタイミング前の RAM

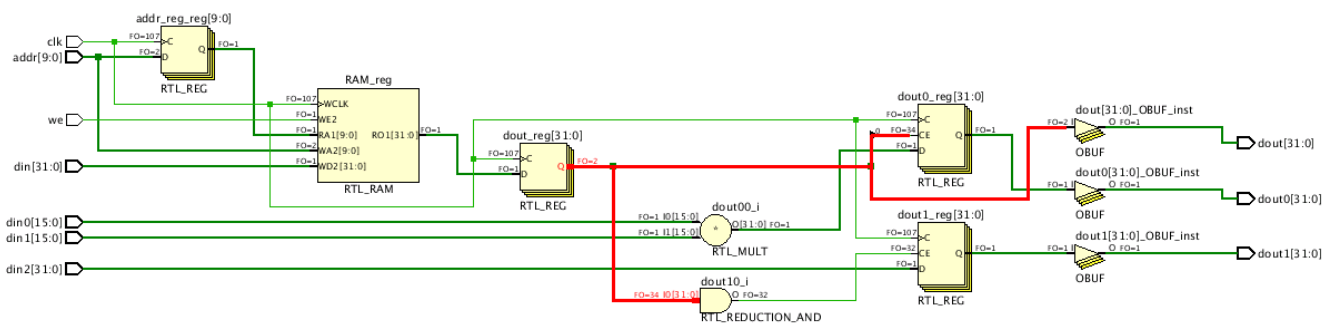


これらの例のバリエーションでも、出力レジスタは推論されません。

読み出しデータ レジスタの出力のファンアウトを確認

2 つ目のレジスタが RAM プリミティブに含まれるようにするには、メモリ配列からのデータ出力ビットのファンアウトを 1 にする必要があります。これを次の図に示します。

図 30: 複数のファンアウトによりブロック RAM 出力レジスタが推論されない状況

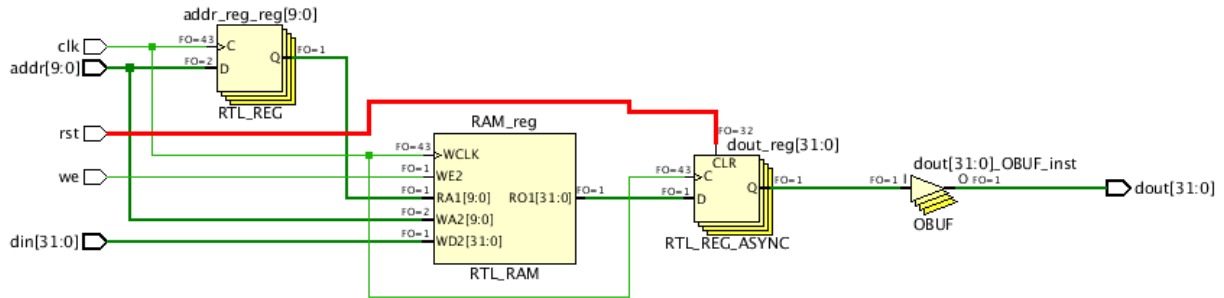


アドレス/読み出しデータ レジスタのリセット信号を確認

メモリ配列をリセットすることはできません。RAM の出力にのみリセットを使用できます。出力レジスタを RAM プリミティブ内に推論するには、リセットは同期である必要があります。非同期リセットを使用すると、レジスタが RAM プリミティブ内に推論されなくなります。また、出力信号は 0 にのみリセットできます。

次の図に、RAM および出力レジスタが正しく推論されるようにするために回避する必要がある状況を示します。

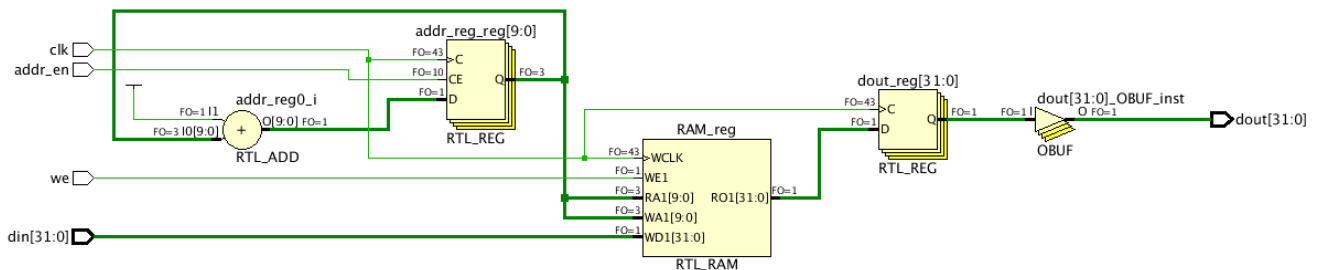
図 31: アドレス/読み出しデータ レジスタのリセットを確認



レジスタのフィードバック構造を確認

レジスタに次の図に示すようにフィードバックが含まれないようにします。加算器にレジスタの現在の値が必要なので、このロジックはリタイミングできず、ブロック RAM に配置できません。生成される回路は、出力レジスタのないブロック RAM です (DOA_REG および DOB_REG が 0)。

図 32: RAM ブロックの周辺にフィードバックがあるかどうかを確認



メモリを UltraRAM ブロックにマップ

UltraRAM は、1 つのクロックを使用する 2 つのポートを持つ 4Kx72 メモリ ブロックです。このプリミティブは、一部の UltraScale+™ デバイスでのみ使用可能です。これらのデバイスでは、ブロック RAM リソースに加えて UltraRAM が含まれています。

デザインで UltraRAM を使用するには、次のいずれかの方法を使用します。

- HDL のメモリ宣言に `ram_style = "ultra"` 属性を設定して合成で UltraRAM が推論されるようにする。
- ザイリンクス XPM_MEMORY プリミティブをインスタンス化。
- UltraRAM UNISIM プリミティブをインスタンス化。

下に示すコード例は、HDL 言語テンプレートに含まれる XPM メモリのインスタンス化です。ハイライトされている `MEMORY_PRIMITIVE` および `READ_LATENCY` が、メモリを UltraRAM に推論して高パフォーマンスを達成するための主要なパラメーターです。

- `MEMORY_PRIMITIVE = "ultra"` は、メモリを UltraRAM として推論するよう指定します。
- `READ_LATENCY` は、メモリの出力に存在するパイプライン レジスタの数を指定します。

大型メモリは、複数の UltraRAM セルを行 x 列の構造に構成した UltraRAM マトリックスにマップされます。

マトリックスは、ワード数によって、1 列または複数列で作成できます。UltraRAM 列の高さのデフォルトしきい値は 8 ですが、`CASCADE_HEIGHT` 属性を使用して制御できます。

1 列と複数列の UltraRAM マトリックスの違いは、次のとおりです。

- 1 列 UltraRAM マトリックスでは、ファブリック ロジックを使用せずにビルトインのハードウェア カスケードが使用されます。
- 複数列 UltraRAM マトリックスでは、各列にビルトインのハードウェア カスケードが使用され、列どうしを接続するためにファブリック ロジックが使用されます。パフォーマンスを保持するため、パイプラインを追加する必要がある場合があります。これは、読み出しレイテンシを増加することにより推論されます。Vivado ツールでは、必要に応じてこれらのレジスタが UltraRAM に自動的に挿入されます。

図 33: RTL コードでの UltraRAM の指定 (XPM を使用)

```
xpm_memory_spram # (
    // Common module parameters
    .MEMORY_SIZE      (8*(4096*72)), //positive integer
    .MEMORY_PRIMITIVE ("ultra"),      //string; "auto", "distributed", "block" or "ultra";
    .MEMORY_INIT_FILE ("none"),       //string; "none" or "<filename>.mem"
    .MEMORY_INIT_PARAM (""),          //string;
    .USE_MEM_INIT     (0),             //integer; 0,1
    .WAKEUP_TIME       ("disable_sleep"), //string; "disable_sleep" or "use_sleep_pin"
    .MESSAGE_CONTROL   (0),            //integer; 0,1

    // Port A module parameters
    .WRITE_DATA_WIDTH_A (72),           //positive integer
    .READ_DATA_WIDTH_A  (72),           //positive integer
    .BYTE_WRITE_WIDTH_A (72),           //integer; 8, 9, or WRITE_DATA_WIDTH_A value
    .ADDR_WIDTH_A       (16),           //positive integer
    .READ_RESET_VALUE_A ("0"),          //string
    .READ_LATENCY_A     (9),            //non-negative integer
    .WRITE_MODE_A       ("read_first")  //string; "write_first", "read_first", "no_change"

) xpm_memory_spram_inst (

    // Common module ports
    .sleep      (1'b0),

    // Port A module ports
    .clk_a      (clk_a),
    .rst_a      (rst_a),
    .ena        (ena),
    .regcea     (regcea),
    .wea        (wea),
    .addra      (addra),
    .dina       (dina),
    .injectsbiterra (1'b0), //do not change
    .injectdbiterra (1'b0), //do not change
    .douta      (douta),
    .sbiterra   (), //do not change
    .dbiterra   () //do not change

);
```


上記の例では、8 つの UltraRAM を使用する 32K x 72 メモリ構造が使用されています。UltraRAM のパフォーマンスを向上するには、カスケード チェーンにさらにパイプライン レジスタを追加する必要があります。これは、読み出しレイテンシを増加することにより達成されます。

Vivado 合成での UltraRAM の推論に関する詳細は、『Vivado Design Suite ユーザー ガイド: 合成』(UG901) の[このセクション](#)を参照してください。

DSP および演算ブロックを最適に推論するためのコード記述

ザイリンクス デバイスの DSP ブロックでは、次のようなさまざまなファンクションを実行できます。

- 乗算
- 加算および除算
- コンパレータ
- カウンター
- 汎用ロジック

DSP ブロックは複数のレジスタ段を使用してパイプライン処理されており、全体的なリソースの消費電力を削減しながら高速動作が可能です。ザイリンクスでは、コードを完全にパイプライン処理して DSP48 にマップされるようにし、すべてのパイプライン段が使用されるようにすることをお勧めします。この追加リソースが柔軟に使用されるようにするには、セット条件をファンクション内に含めないようにして、DSP48 に正しくマップされるようにします。

ザイリンクス デバイス内の DSP48 スライス レジスタには、リセットのみが含まれ、セットは含まれません。そのため、必要でない限り、DSP48 スライス内にインプリメント可能な乗算器、加算器、カウンター、またはその他のロジックの付近にセット (信号適用時の値が 1) は記述しないようにしてください。また、DSP スライスでは同期リセット動作しかサポートされないため、非同期リセットは使用しないでください。セットまたは非同期リセットを生成するコードでは、エリア、パフォーマンス、または消費電力の面で最適な結果が得られない可能性があります。

多くの DSP デザインは、ザイリンクス アーキテクチャに適しています。アーキテクチャが最適に使用されるようにするには、アーキテクチャの機能について理解し、デザイン入力コードでこれらのリソースが活用されるようにする必要があります。

DSP48 ブロックでは、符号付き演算のインプリメンテーションが使用されます。ザイリンクスでは、リソース機能に適した最も効率的なマップを達成できるように、HDL ソースに符号付きの値を記述することをお勧めします。符号なしのバス値がコードで使用された場合でも、合成ツールでこのリソースを使用できる可能性はありますが、符号なしから符号付きへの変換があるため、そのコンポーネントの完全なビット精度を得られない可能性があります。

ザイリンクスでは、ターゲット デザインに多数の加算器が含まれる場合は、デザインを評価して、DSP48 スライスの前置加算器と後置加算器がより多く使用されるようにすることをお勧めします。たとえば、FIR フィルターを使用する場合は、複数の加算器を連続して使用する (加算器ツリー) よりも、加算器カスケードを使用してシストリック (対称) フィルターを構築できます。フィルターが対称の場合は、専用の前置加算器を使用して、ファンクションをより少ない数の LUT とフリップフロップに、また、より少ない数の DSP スライス (ほとんどの場合リソース量は半分) に統合できるかどうかを確認します。

加算器ツリーが必要な場合は、6 入力 LUT アーキテクチャで、単純な 2 入力加算と同じリソース量を使用するだけで三項加算 ($A + B + C = D$) を効率的に作成できます。これにより、キャリア ロジック リソースを節約できます。多くの場合、これらの方法を使用する必要はありません。

これらの機能を知っておくと、トレードオフを前もって認識できるので、インプリメンテーションが最初からよりスムーズで効率的に実行されるように RTL コードを記述できます。ザイリンクスでは、ほとんどの場合、DSP リソースを推論することをお勧めします。

DSP48 スライスの機能に関する詳細、およびデザインの要件に合わせたこのリソースの利用方法は、『7 シリーズ FPGA DSP48E1 スライス ユーザー ガイド』(UG479: [英語版](#)、[日本語版](#)) および『UltraScale アーキテクチャ DSP スライス ユーザー ガイド』(UG579: [英語版](#)、[日本語版](#)) を参照してください。

シフト レジスタおよび遅延ラインコード記述

シフト レジスタには、通常次の制御信号およびデータ信号の一部またはすべてが含まれます。

- クロック
- シリアル入力
- 非同期セット/リセット
- 同期セット/リセット
- 同期/非同期パラレル ロード
- クロック イネーブル
- シリアル/パラレル出力

ザイリンクス デバイスには専用の SRL16 および SRL32 リソース (LUT に組み込まれている) が含まれており、フリップフロップ リソースを使用せずにシフト レジスタを効率的に生成できます。ただし、これらのエレメントではレフト シフトしかサポートされておらず、使用可能な次の I/O 信号の数も制限されます。

- クロック
- クロック イネーブル
- シリアル データ入力
- シリアルデータ出力

SRL には、上記の信号に加え、シフト レジスタの長さを決定するアドレス入力 (SRL16 では A3、A2、A1、A0 入力) があります。シフト レジスタの長さは、固定することも変動させることもできます。

可変長モードでは、新しいアドレスが 4 ビットの入力アドレス ピンに読み込まれると、LUT にアクセスする時間分遅れて、Q に新しいビット位置の値が出力されます。SLR プリミティブでは、同期および非同期セット/リセット制御信号は使用できません。ただし、RTL コードにリセットが含まれる場合は、ザイリンクス合成ツールで SRL の周りに追加のロジックが推論され、リセット機能が提供されます。

SRL を使用する際に最適なパフォーマンスを達成するには、ザイリンクスではシフト レジスタの最後の段を専用スライス レジスタにインプリメントすることをお勧めします。スライス レジスタを使用した方が SRL よりも clock-to-out 時間が短くなり、シフト レジスタ ロジックからのパスにスラックが追加されます。このレジスタがインスタンス化されているか、属性または階層境界を越える最適化の制限によりそのようなレジスタが推論できない場合を除き、このレジスタは合成で自動的に推論されます。

ザイリンクスでは、Vivado Design Suite の HDL テンプレートに記述されている HDL コーディング スタイルを使用することをお勧めします。

チップの配置を柔軟にするためにレジスタを使用する場合は、次の属性を指定して SRL の推論をオフにします。

```
SHREG_EXTRACT = "no"
```

合成属性に関する詳細とこれらの属性の HDL コードでの指定方法は、『Vivado Design Suite ユーザー ガイド: 合成』([UG901](#)) を参照してください。

推論されたレジスタ、SRL、およびメモリの初期化

GSR ネットは、すべてのレジスタを HDL コードで指定された初期値に初期化します。初期値が指定されていない場合、合成ツールにより初期状態が 0 か 1 のいずれかに割り当てられます。Vivado 合成では通常デフォルトで 0 になりますが、ワンホット ステート マシン エンコードのような例外もあります。

推論された SRL メモリまたはその他の同期エレメントにも定義された初期状態があり、コンフィギュレーション時に関連するエレメントにプログラムされます。

ザイリンクスでは、同期エレメントはすべて適切に初期化されるようにすることを強くお勧めします。レジスタの初期化は、主なデバイス合成ツールすべてで完全に推論できます。これにより、初期化目的のみにリセットを追加する必要がなくなり、デバイスでコンフィギュレーション後にすべての同期エレメントが既知の値で開始するので、RTL コードが論理シミュレーションのインプリメント済みデザインとさらに一致するようになります。

レジスタおよびラッチの初期状態を指定する VHDL コード例 1:

```
signal reg1 : std_logic := '0'; -- specifying register1 to start as a zero
signal reg2 : std_logic := '1'; -- specifying register2 to start as a one
signal reg3 : std_logic_vector(3 downto 0):="1011"; -- specifying INIT
value for
4-bit register
```

レジスタおよびラッチの初期状態を指定する Verilog コード例 2:

```
reg register1 = 1'b0; // specifying register1 to start as a zero
reg register2 = 1'b1; // specifying register2 to start as a one
reg [3:0] register3 = 4'b1011; //specifying INIT value for 4-bit register
```

Verilog では、initial 文も使用できます。

```
reg [3:0] register3;
initial begin
    register3= 4'b1011;
end
```

インスタンスエートするか推論するかの判断

ザイリンクスでは、デザインを RTL で記述し、合成ツールでコードがデバイスで使用可能なリソースにマップされるようにすることをお勧めします。コードが移植しやすくなるだけでなく、合成ツールで推論されたロジックすべてが認識され、ファンクション間の最適化が実行されるようになります。これらの最適化には、ロジック複製、再構築と統合、レジスタ間のロジック遅延のバランスを取るためのリタイミングなどが含まれます。

合成ツールの最適化

デバイスのライブラリ セルがインスタンスエートされる場合、合成ツールではそれらはデフォルトでは最適化されません。デバイスのライブラリ セルを最適化するように指定した場合でも、合成ツールで RTL を使用した場合と同じレベルの最適化は実行できません。通常はこれらのセルに入力されるパスおよびセルから出力されるパスの最適化のみが実行され、セルを通過するパスは最適化されません。

たとえば、SRL がインスタンスエートされていて、それが長いパスの一部である場合、このパスがボトルネックとなる可能性があります。SRL の clock-to-out 遅延は通常のレジスタの遅延よりも大きくなります。SRL によるエリア削減を維持しながら clock-to-out パフォーマンスを改善するため、実際に必要な遅延よりも遅延が 1 つ少ない SRL が作成され、最後の段は標準フリップフロップにインプリメントされます。

インスタンスーションが必要な場合

合成ツールのマップでタイミング、消費電力、エリア制約が満たされない場合、またはデバイス内の特定機能を推論できない場合、インスタンスーションを使用した方がよいことがあります。

インスタンスーションすると、ユーザーが合成ツールを完全に制御できます。たとえば、パフォーマンスを改善するために、通常合成ツールで選択される LUT とキャリー チェーン エLEMENT の組み合わせではなく、LUT のみを使用するコンパレータをインプリメントできます。

デバイスで使用可能な複雑なリソースを使用できるようにするには、インスタンスーションが唯一の方法であることもあります。これは、次が原因です。

- HDL 言語の制限

たとえば、VHDL でダブル データ レート (DDR) 出力を記述することはできません。これは、2 つの別のプロセスで同じ信号を駆動する必要があるからです。

- ハードウェアの複雑さ

合成可能なコードを作成するよりも、I/O SerDes ELEMENT をインスタンスエートの方が簡単です。

- 合成ツールの推論の制限

たとえば、合成ツールには現在のところ RTL 記述からハード FIFO を推論する機能はないので、インスタンスエートする必要があります。

ザイリンクス プリミティブをインスタンスエートする場合は、ターゲット アーキテクチャのユーザー ガイドおよびライブラリ ガイドを参照し、コンポーネントの機能、設定、接続を完全に理解してください。

推論とインスタンスエートのどちらでも、ザイリンクスでは Vivado Design Suite 言語テンプレートからテンプレートを使用することをお勧めします。

次にヒントを示します。

- 可能な限り機能を推論します。
- 合成可能な RTL コードで要件が満たされない場合は、コードをデバイス ライブラリ コンポーネントのインスタンスーションと置き換える前に、要件を見直します。
- 一般的な Verilog および VHDL ビヘイビア構文を記述する場合や、プリミティブをインスタンスエートする必要がある場合は、Vivado Design Suite 言語テンプレートを使用することを考慮します。

パフォーマンスを向上するためのコーディング スタイル

高パフォーマンスのデザインでは、このセクションで説明されるコーディング手法を使用することにより、発生する可能性のあるタイミング問題を軽減できます。

クリティカル パスでのファンアウトの大きいネット

ファンアウトの大きいネットは、デザイン プロセスの早期に対処の方が簡単です。ファンアウトが大きすぎる原因は、パフォーマンス要件およびパスの構造にあることが多いです。ファンアウトの大きいネットの問題を解決するには、次の手法を使用できます。



推奨: ファンアウトの大きいネットを特定するには、合成後に `report_high_fanout_nets` Tcl コマンドを使用します。インプリメンテーション プロセスの進行中、これらのネットがデザイン パフォーマンスに与える影響を監視します。

ネットを必要としないデザイン部分でのロードの削減

ファンアウトの大きい制御信号の場合、デザインのコード記述されたすべての部分でそのネットが必要かどうかを判断します。ロード数を減らすと、タイミング問題を大幅に改善できることがあります。

ファンアウトの大きいネット ドライバーの複製

レジスタの複製を使用すると、レジスタのコピーを作成して信号のファンアウトを削減することにより、クリティカルパスを高速化できます。これにより、インプリメンテーション ツールでさまざまなロードおよび関連ロジックの配置配線がより柔軟に実行されるようになります。合成ツールでは、この手法が広範に使用されます。

ほとんどの合成ツールでは、ファンアウトしきい値制限を使用して、レジスタを複製するかどうか自動的に決定されます。このグローバルしきい値を下げると、ファンアウトの大きいネットが自動的に複製されるようになりますが、どのレジスタを複製するか、そのロードをどのようにグループ化するかなどの詳細な制御はできません。また、グローバル複製メカニズムではタイミング スラックが正確に評価されないため、セルが不必要に複製されたり、ロジックの使用率が増加したり、消費電力が大きくなったりする可能性があります。

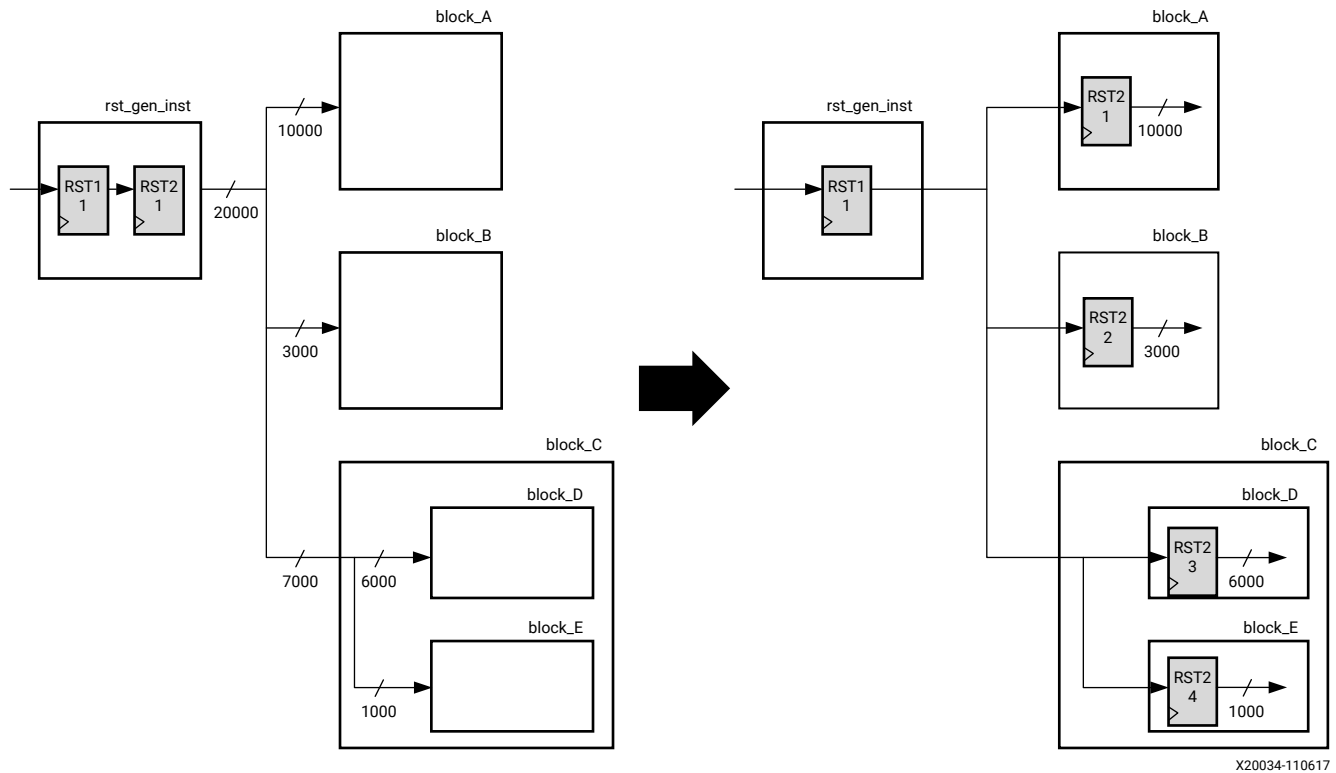
ファンアウトを削減するには、多くの場合、ファンアウトの大きい信号に対してバランスの取れたツリーを使用するのがより良い方法です。1 つの階層に含まれるセルは通常一緒に配置されるので、デザイン階層に基づいてレジスタを手動で複製することを考慮してください。たとえば、次の図に示すバランスの取れたリセット ツリーでは、ファンアウトの大きいリセット FF RST2 が RTL で複製され、異なるモジュール間でファンアウトのバランスが取られます。必要であれば、物理合成でさらに複製を実行し、配置情報に基づいて WNS を向上できます。



ヒント: 合成で複製されたレジスタを保持するには、DONT_TOUCH ではなく KEEP 属性を使用してください。DONT_TOUCH 属性を使用すると、インプリメンテーション フローの物理最適化で最適化が実行されなくなります。

注記: レジスタではなく LUT1 が複製された場合は、属性または制約が正しく適用されていません。

図 34: ファンアウトの大きいリセットをバランスの取れたリセット ツリーに変換



X20034-110617



推奨: ファンアウトの大きいグローバル信号に MAX_FANOUT 属性を使用すると、合成でグローバル ファンアウト制限を下げた場合と同様に、最適な複製が実行されません。そのためザイリンクスでは、MAX_FANOUT は階層内のファンアウトが中から小のローカル信号にのみ設定することをお勧めします。

クロック ドメイン間をまたぐ信号を同期するのに使用されたレジスタは複製しないようにしてください。これらのレジスタに ASYNC_REG 属性を適用すると、複製されなくなります。同期チェーンのファンアウトが大きく、タイミングを満たすために複製が必要な場合は、同期チェーンの後に ASYNC_REG 属性を設定しないレジスタを追加します。

次の表に、デザインで許容されるファンアウト数のガイドラインを示します。

表 1: 中程度パフォーマンスの 7 シリーズ デバイスでのファンアウト ガイドライン

条件	ファンアウト > 5000	ファンアウト > 200	ファンアウト > 100
低周波数 1 ~ 125 MHz	最大周波数での同期ロジック間のロジック段数は 13 未満	なし	なし
中周波数 125 ~ 250 MHz	デザインでタイミングが満たされない場合、ファンアウトまたはロジック段数を削減する必要がある場合あり。	最大周波数でのロジック段数は 6 未満 (ドライバーおよびロードタイプがパフォーマンスに影響)。	なし
高周波数 > 250 MHz	ほとんどのデザインで推奨されない。	高速の場合、通常ロジック段数を少なくすることが必要。	アドバンス パイプライン手法が必要。注意してロジック複製。コンパクトなファンクション。ロジック段数を少なくすることが必要 (ドライバーおよびロードタイプがパフォーマンスに影響)。



ヒント: タイミング レポートにファンアウトの大きい信号が原因でデザイン パフォーマンスが制限されていることが示されている場合は、`opt_design -hier_fanout_limit`、`place_design`、`phys_opt_design` などのインプリメンテーション ツール オプションを使用して信号を複製することを考慮してください。



ヒント: レジスタを複製する際は、`<original_name>_a`、`<original_name>_b` のような名前を付けて複製されたものであることがわかるようにし、RTL コードが管理しやすくなるようにしてください。

パイプラインに関する注意事項

パフォーマンスを向上するには、ロジック段数が数段の長いデータパスを再構築し、複数のクロック サイクルに分散するという方法もあります。この方法を使用すると、クロック サイクルが速くなり、データ スループットが増加しますが、レイテンシが増加し、パイプライン オーバーヘッド ロジックの管理が必要になります。

デバイスにはレジスタが多く含まれるので、レジスタの追加およびロジックのオーバーヘッドは通常問題とはなりません。ただし、データパスは複数サイクルにまたがるので、残りのデザインで追加されたパス レイテンシを考慮する必要があります。

SSI デバイスのパイプラインを考慮

SLR の境界をまたぐレジスタ間の接続を高パフォーマンスにするには、HDL コードに適切なパイプライン処理を記述して、合成で制御する必要があります。これにより、シフト レジスタ LUT (SRL) の推論およびその他の最適化が、SLR 境界をまたぐ必要のあるロジック パスでは実行されなくなります。この方法でコードを変更し、Pblock を適切に使用することにより、SLR の境界を超える箇所が定義されます。

パイプラインを前もって考慮

パイプラインについて後で考慮するのではなく、前もって考慮しておくことで、タイミング クロージャを改善しやすくなります。特定のパスに後の段階でパイプラインを追加すると、通常はレイテンシの差が回路全体に伝搬されます。そのため、少しと思われた変更により、コードの一部を大きく設計し直す必要が出てくる場合があります。

パイプラインを使用するかどうかをデザインの初期段階で判断しておく、タイミング クロージャ、インプリメンテーションの実行時間 (タイミング問題が解決しやすいため)、およびデバイス消費電力 (ロジックの切り替えが削減される) が大幅に改善されることがよくあります。

推論ロジックをチェック

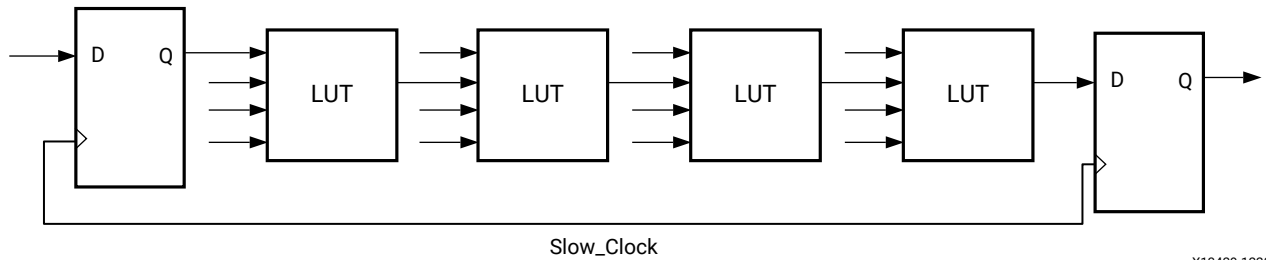
デザインをコード記述する際には、推論されるロジックに注意してください。次の状況で、パイプラインをさらに考慮する必要があるかどうかを調べます。

- ファンインの大きいロジック コーン
たとえば、出力の算出に大きなバスや複数の組み合わせ信号を必要とするコードです。
- 配置に制限がある、clock-to-out が遅い、またはセットアップ要件が大きいブロック
たとえば、出力レジスタのないブロック RAM または正しくパイプラインされていない演算コードです。
- 長い配線の原因となる指定配線
たとえば、ピン配置により配線がチップを横断する必要がある場合、高速動作を可能にするためにパイプラインが必要です。
- 大型の XOR 関数で構成されるロジック
大型の XOR 関数は、スイッチング レートが高く、ダイナミック消費電力が大きくなる可能性があります。これらの関数をパイプライン処理してスイッチングを削減すると、回路の消費電力を削減できる可能性があります。

次の図では、クロック速度が次によって制限されます。

- ソース フリップフロップの clock-to-out タイム
- 4 段のロジックを介するロジック遅延
- 4 個のファンクション ジェネレーターに関連した配線
- デスティネーション レジスタのセット アップ タイム

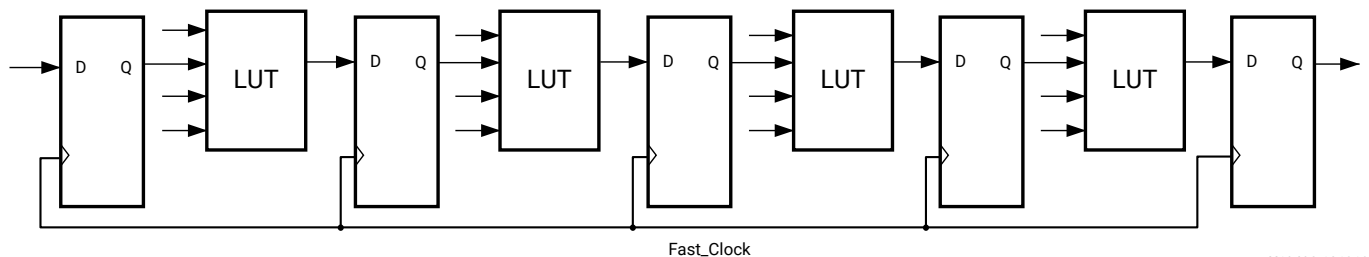
図 35: パイプライン処理前



X13429-122019

次の図は、パイプライン処理前と同じデータパスの例です。フリップフロップがファンクション ジェネレーターと同じスライスに含まれているので、クロック速度はソース フリップフロップの clock-to-out 時間、1 段のロジックを介するロジック遅延、配線遅延、デスティネーション レジスタのセットアップ タイムによって制限されます。この例では、パイプライン処理前よりシステム クロックが高速になります。

図 36: パイプライン処理後



X13430-121919

パイプライン処理が必要かどうかの決定

大きな組み合わせロジック パスを特定し、より小さなパスに分割し、これらのパスの間にレジスタ段を挿入して、各パイプライン段をバランス調整するのが、よく使用されるパイプライン処理手法です。

デザインにパイプライン処理が必要かどうかを決定するには、クロックの周波数と各クロック グループに分散されるロジック量を確認します。report_design_analysis Tcl コマンドを -logic_level_distribution オプションを指定して使用すると、各クロック グループのロジック段の分布を判断できます。



ヒント: デザイン解析レポートには、ロジック段数 0 のパス数も示されるので、コードのどの部分を修正すればいいかを判断できます。

レイテンシのバランス

レイテンシのバランスを調整するには、パイプライン段をデータパスではなく制御パスに追加します。データパスにはより幅の広いバスが含まれるので、使用されるフリップフロップとレジスタ リソースの数が増えます。

たとえば、128 ビットのデータパスと 2 段のレジスタがあり、レイテンシ要件が 5 サイクルの場合は、レジスタを 3 段追加すると $3 \times 128 = 384$ 個のフリップフロップが追加されます。または、レジスタを使用してロジックを制御し、データパスをイネーブルにできます。5 段の 1 ビットレジスタを使用して、データパス フリップフロップのイネーブル信号を制御し、それに合わせてマルチサイクル パスのタイミング例外を調整します。

注記: この例は、特定のデザインの場合にのみ可能です。たとえば、中間データパス フリップフロップからのファンアウトがある場合、2 段だけでは動作しません。



推奨: デバイス最適な LUT:FF 比は 1:1 です。デザインのフリップフロップの比率が大幅に高い場合、スライスにパックされる関係のないロジックの量が増え、配線の複雑性が増し、QoR (結果の品質) が低下します。

パイプライン段数と SRL の使用量のバランス

レジスタ パイプラインの段数が多い場合、できるだけ多くのレジスタを SRL にマップし、レジスタの使用率が大幅に増加するのを回避します。たとえば、データ幅 32 の 9 段のパイプラインには、各ビットに 9 個のレジスタ、合計 $32 \times 9 = 288$ 個のレジスタが使用されます。同じ構造を SRL にマップすると、32 個の SRL が使用されます。各 SRL には、5'b01000 に接続されたアドレス ピン A4 ~ A0 があり、9 段のパイプラインをインプリメントできます。

合成で SRL を推論する方法は、次のように複数あります。

- SRL
- REG -> SRL
- SRL -> REG
- REG -> SRL -> REG

これらの構造は、次のように RTL コードで `srl_style` 属性を使用すると作成できます。

- `(* srl_style = "srl" *)`
- `(* srl_style = "reg_srl" *)`
- `(* srl_style = "srl_reg" *)`
- `(* srl_style = "reg_srl_reg" *)`

より深いパイプライン段で異なるイネーブル/リセット制御信号を使用してしまうというのが、よくある間違いです。次は、深さ 9 のパイプライン段で使われるリセット (3、4、8 つ目のパイプライン段に接続) の例です。この構造では、SRL プリミティブにリセット ピンがあるので、パイプライン段はレジスタのみにマップされます。

```
FF->FF->FF(reset) -> FF->FF(reset)->FF->FF->FF(reset)->FF
```

SRL 推論の利点を活かすには、次のようにします。

- パイプライン段用のリセットがないようにします。
- リセットが本当に必要かどうかを解析します。
- フリップフロップの 1 つ (たとえばパイプラインの最初または最後の段) でリセットを使用します。

不要なパイプラインを回避

使用率の高いデザインでは、パイプライン処理が多すぎると最適な結果が得られないことがあります。たとえば、不要なパイプライン段があると、フリップフロップと配線リソースの数が増加し、使用率が高い場合に配置配線が制限されることがあります。

注記: 0/1 レベルのロジックを含むパスが多くある場合は、それが意図したものであることを確認してください。

パイプライン マクロ プリミティブを考慮

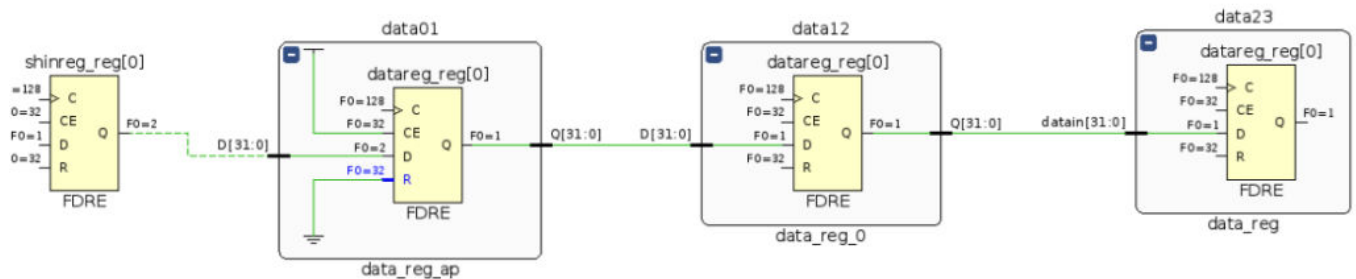
ターゲット アーキテクチャに基づいて、パイプライン処理が十分使用されている場合は、ブロック RAM や DSP などの専用プリミティブは 500 MHz 以上で動作できます。サイリンクスでは、周波数が高いデザインには、これらのブロック内のすべてのパイプラインを使用することをお勧めします。

自動パイプライン処理に関する注意事項

自動パイプライン処理機能を使用すると、配置に必要なパイプライン段数とその最適な場所が判断され、インターフェースの境界をまたぐ部分のタイミング クロージャを達成するのに役立ちます。この機能をイネーブルにするには、AXI レジスタ スライス コアの自動パイプライン処理モード設定するか、データ バスに自動パイプライン処理 HDL 属性または XDC 制約を適用します。挿入はタイミング ドリブンなので、ターゲット パスに適切なタイミング制約を適用するようにしてください。詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 (UG904) のこのセクションを参照してください。

次に、モジュール data01 と data12 の間のインターフェースに自動パイプライン処理を適用した例を示します。data01 からの出力は、制御セットのないレジスタで構成されています。

図 37: モジュール間のシンプルなデータフロー接続



この例の RTL コードは、次のとおりです。階層モジュール data01 に autopipeline_module 属性を適用し、レジスタの Q ピンで直接駆動されるネットに autopipeline_group、autopipeline_limit、autopipeline_include 属性を適用する必要があります。

```
data_reg_ap # ( .C_DATA_WIDTH(C_DATA_WIDTH)) data01 (
    .clk (clk),
    .datain (shinreg),
    .datareg (d1)
);

data_reg # ( .C_DATA_WIDTH(C_DATA_WIDTH)) data12 (
    .clk (clk),
    .datain (d1),
    .datareg (d2)
);

(* autopipeline_module="yes" *)
module data_reg_ap # (
    parameter integer C_DATA_WIDTH = 32
)
(
    input wire clk,
    input wire [C_DATA_WIDTH-1:0] datain,
    (* autopipeline_group="fwd", autopipeline_limit=24 *)
    output reg [C_DATA_WIDTH-1:0] datareg
);
```

```
always @(posedge clk) begin
    datareg <= datain;
end
endmodule
```

この例の XDC 制約は、次のとおりです。これは、RTL コードで属性を設定する代わりに使用します。

```
# It's suggested to add the USER_SLR_ASSIGNMENT property at the module
level to ensure better logic clustering with its driver and load, see UG912
for more details on this property
set_property USER_SLR_ASSIGNMENT APSRC [get_cells data01]
set_property USER_SLR_ASSIGNMENT APDST [get_cells data12]

set_property AUTOPIPELINE_MODULE TRUE [get_cells data01]
set_property AUTOPIPELINE_GROUP WBUS [get_nets -of [get_pins -filter
REF_PIN_NAME==Q -of [get_cells data01/*]]]
set_property AUTOPIPELINE_LIMIT 10 [get_nets -of [get_pins -filter
REF_PIN_NAME==Q -of [get_cells data01/*]]]
```

消費電力を向上するためのコーディング スタイル

クロックまたはデータパスにゲートを付ける

クロックまたはデータパスにゲートを付けるのは、これらのパスの結果が使用されない場合に、遷移を停止するためによく使用される方法です。クロックにゲートを付けると、駆動されているすべての同期ロードが停止し、データパス信号が切り替わらなくなり、グリッチが伝搬されなくなります。

消費電力の最適化 (power_opt_design) を使用すると、信号ゲーティング ロジックを自動的に生成し、スイッチング アクティビティを削減できます。ただし、ツールでは認識されず、設計者のみが指定可能なアプリケーション、データフロー、および依存性に関する情報もあります。

ゲートで制御するエレメントの数を最大にする

ゲーティング信号により制御するエレメントの数を最大限にします。たとえば、各ロードをクロック イネーブル信号でゲーティングするよりも、クロック ドメインの駆動ソースをゲーティングする方が消費電力の節約量が大きくなります。

専用クロック バッファのクロック イネーブル ピンを使用する

アクティビティまたはクロック ツリーの使用を最小限に抑えるためにクロックをゲーティングしたりクロックにマルチプレクサーを付ける場合は、専用クロック バッファのクロック イネーブル ポートを使用します。LUT を挿入したり、その他の方法でクロック信号をゲーティングすると、消費電力とタイミングの面で効率的ではありません。

プライオリティ エンコーダーが不要な場合は case ブロックを使用する

プライオリティ エンコーディングが不要な場合は、if-then-else ブロックや三項演算の代わりに case ブロックを使用します。

効率の悪いコード例:

```
if (reg1)
    val = reg_in1;
else if (reg2)
    val = reg_in2;
else if (reg3)
    val = reg_in3;
else val = reg_in4;
```

正しいコード例:

```
(* parallel_case *) casex ({reg1, reg2, reg3})
1xx: val = reg_in1 ;
01x: val = reg_in2 ;
001: val = reg_in3 ;
default: val = reg_in4 ;
endcase
```

ブロック RAM でのパフォーマンスと消費電力のトレードオフ

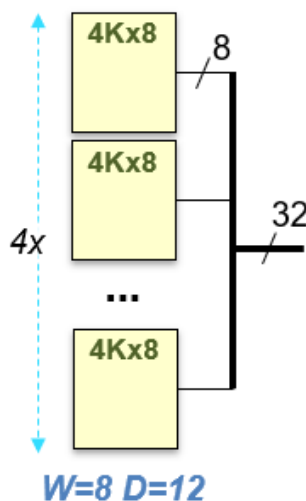
特定の要件に対応するため、メモリ構造を分割する方法が複数あります。デザインの要件は、パフォーマンス、消費電力、またはその両方の組み合わせです。

次の例では、要件を達成するために生成可能な異なる構造を示します。パフォーマンスと消費電力のトレードオフのため、CASCADE_HEIGHT 属性を使用してブロック RAM のカスケードを制限できます。この属性の使用方法および引数は、『Vivado Design Suite ユーザー ガイド: 合成』(UG901) を参照してください。

次の図に、高パフォーマンスを達成するための 8Kx32 メモリ構造の例を示します。

注記: この例は、UltraScale および UltraScale+ デバイスの場合のみ使用できます。

図 38: 4Kx8 と CASCADE_HEIGHT=1 を使用した 4Kx32 の RTL 記述



```
module test(
    input clk,
    input we,
    input [31:0] din,
    input [11:0] addr,
    output reg [31:0] dout
);

(* ram_style = "block", cascade_height = 1 *)
reg [31:0] mem [(2**12)-1:0];
reg [11:0] addr_reg;

always @(posedge clk)
begin
    addr_reg <= addr;
    dout <= mem[addr_reg];
    if (we)
        mem[addr_reg] <= din;
end

endmodule
```


このインプリメンテーションでは、各読み出しまたは書き込みですべてのブロック RAM が常にイネーブルになっているので、消費電力が増加します。

次の図に、低消費電力を達成するためにすべてのブロック RAM をカスケード接続した例を示します。

図 39: 1Kx32 と CASCADE_HEIGHT=4 を使用した 4Kx32 の RTL 記述

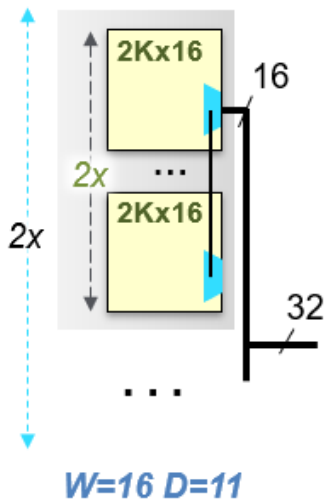


このインプリメンテーションでは、各ユニットから一度に選択されるブロック RAM は 1 つなので、ダイナミックな消費電力はほぼ半分になります。ブロック RAM には専用のカスケード MUX と配線構造があり、複数のブロック RAM プリミティブを必要とする幅が広くワード数の多いメモリを、電力効率の高い構造で作成できます。

次の図に、カスケード接続を制限し、パフォーマンスをトレードオフせずに、消費電力とパフォーマンスの両方を向上する例を示します。

注記: この例は、UltraScale および UltraScale+ デバイスの場合のみ使用できます。

図 40: 2Kx16 と CASCADE_HEIGHT=2 を使用した 4Kx32 の RTL 記述



```

module test(
input clk,
input we,
input [31:0] din,
input [11:0] addr,
output reg [31:0] dout
);

(* ram_style = "block", cascade height = 2 *)
reg [31:0] mem [(2**12)-1:0];
reg [11:0] addr_reg;

always @(posedge clk)
begin
addr_reg <= addr;
dout <= mem[addr_reg];
if (we)
mem[addr_reg] <= din;
end

endmodule

```

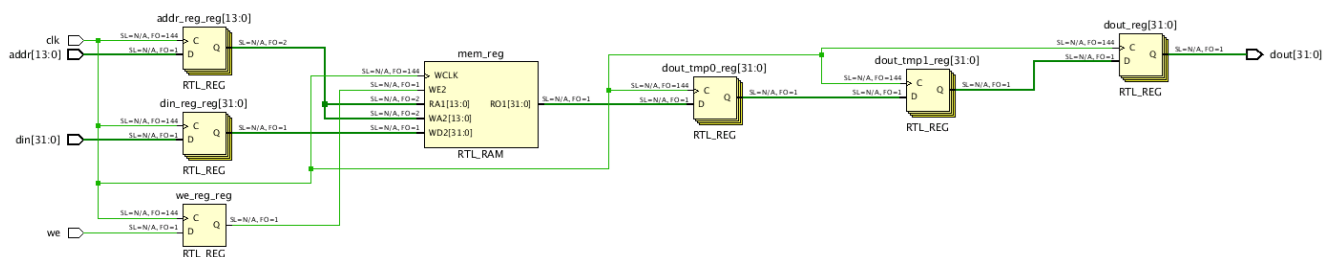
このインプリメンテーションでは、一度に 2 個のブロック RAM が選択されるので、ダイナミック消費電力は高パフォーマンス構造よりも低くなりますが、低消費電力構造よりは高くなります。この構造の利点は、クリティカルパスに 4 個のブロック RAM が含まれている低消費電力構造と比較して、カスケードパスに使用されるブロック RAM は 2 つのみであり、ターゲット周波数が高くなることです。

消費電力とパフォーマンスのバランスを取るためのワード数の多いメモリの分解

ワード数の多いメモリでは、RTL で RAM_DECOMP 合成属性を使用してメモリの構成を向上することにより、消費電力を削減できます。メモリ配列に RAM_DECOMP 属性を適用すると、メモリ ロジックは幅の広い配列のブロック RAM プリミティブにマップされます。消費電力とパフォーマンスのバランスを取るため、CASCADE_HEIGHT 属性と RAM_DECOMP 属性を組み合わせて使用してカスケードを制御できます。この方法ではアドレスデコードロジックがより多く必要になりますが、各読み出し操作でイネーブルになるブロック RAM の数が削減されるので、消費電力が削減されます。

次の図に、32x16K メモリの例を示します。

図 41: 32x16K メモリ



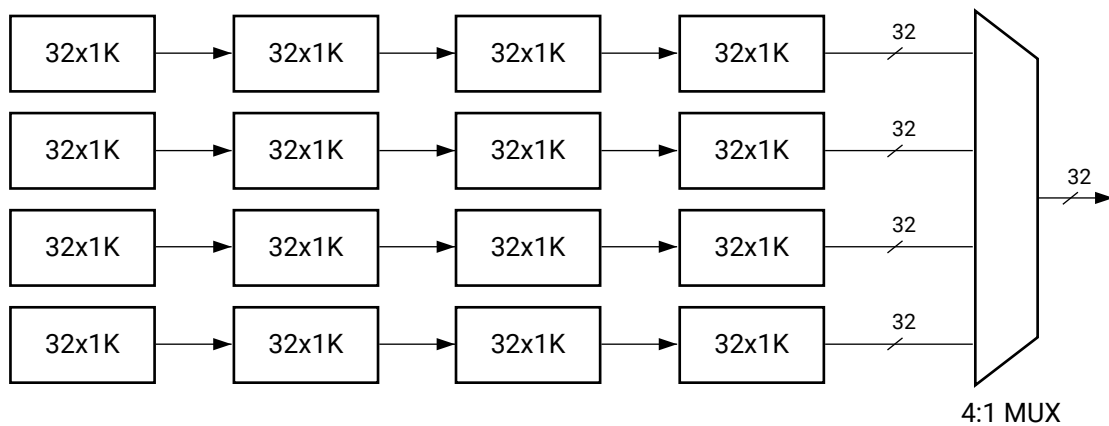
次の属性を適用するとします。

```
ram_decomp = "power"
cascade_height = 4
```

16 個の RAMB36E2 が推論され、メモリは次のように分解されます。

- 基本のプリミティブは 32x1K です。
- 4 つのブロック RAM がカスケード接続され、32x4K が作成されます。
- これを 4 つ並列に構成することにより、ワード数が 16K のメモリが作成されます。
- これらの出力がマルチプレクサーで結合され、出力データが作成されます。

図 42: CASCADE_HEIGHT および RAM_DECOMP 属性を使用して生成された 32x16K メモリの構造例



X19283-121919

次の RTL コードに、CASCADE_HEIGHT および RAM_DECOMP 属性の使用例を示します。

図 43: CASCADE_HEIGHT および RAM_DECOMP 属性を使用した 32x16K メモリの RTL コード例

```
module test
(
  input  clk,
  input  we,
  input  [13:0] addr,
  input  [31:0] din,
  output reg [31:0] dout
);

(* ram_style = "block", ram_decomp = "power", cascade_height = 4 *) reg [31:0] mem [(16*1024)-1:0];
reg [13:0] addr_reg;
reg [31:0] dout_tmp0;
reg [31:0] dout_tmp1;
reg [31:0] din_reg;
reg [31:0] we_reg;

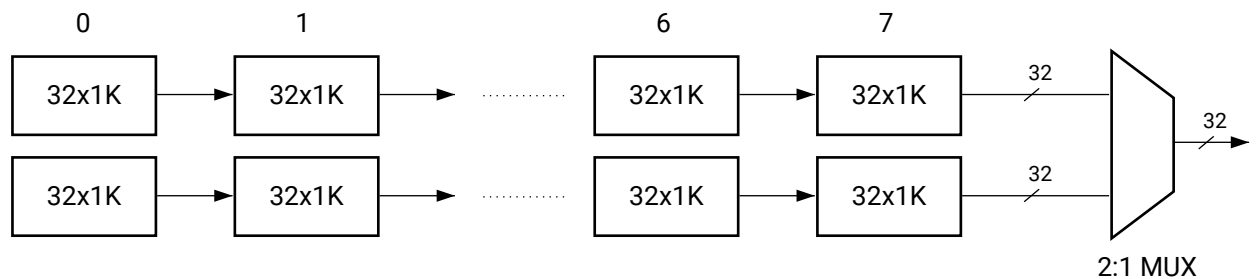
always @(posedge clk)
begin
  addr_reg <= addr;
  din_reg  <= din;
  we_reg   <= we;
  dout_tmp0 <= mem[addr_reg];
  dout_tmp1 <= dout_tmp0;
  dout <= dout_tmp1;
  if (we_reg)
    mem[addr_reg] <= din_reg;
end

endmodule
```

ram_decomp = "power" 属性のみを適用すると、16 個の RAMB36E2 が推論され、メモリは次のように分解されます。

- 基本のプリミティブは 32x1K です。
- 8 つのブロック RAM がカスケード接続され、32x8K が作成されます。
- これを 2 つ並列に構成することにより、ワード数が 16K のメモリが作成されます。
- これらの出力が 2:1 マルチプレクサーで結合され、出力データが作成されます。

図 44: RAM_DECOMP 属性を使用して生成された 32x16K メモリの構造例



X19284-050517

次の RTL コードに、RAM_DECOMP 属性の使用例を示します。

図 45: RAM_DECOMP 属性を使用した 32x16K メモリの RTL コード例

```

module test
(
    input clk,
    input we,
    input [13:0] addr,
    input [31:0] din,
    output reg [31:0] dout
);

(* ram_style = "block", ram_decomp = "power" *) reg [31:0] mem [(16*1024)-1:0];
reg [13:0] addr_reg;
reg [31:0] dout_tmp0;
reg [31:0] dout_tmp1;
reg [31:0] din_reg;
reg we_reg;

always @(posedge clk)
begin
    addr_reg <= addr;
    din_reg <= din;
    we_reg <= we;
    dout_tmp0 <= mem[addr_reg];
    dout_tmp1 <= dout_tmp0;
    dout <= dout_tmp1;
    if (we_reg)
        mem[addr_reg] <= din_reg;
end
endmodule

```

RAM_DECOMP 属性のみを使用した場合、全体的な消費電力の削減量は RAM_DECOMP および CASCADE_HEIGHT 属性の両方を使用した場合に近いです。一度にアクティブになるブロック RAM は 1 つのみです。4 ワード数のカスケード ブロック RAM チェーンの方が 8 ワード数のカスケード ブロック RAM チェーンよりもパフォーマンスは良くなります。

詳細は、『Vivado Design Suite ユーザー ガイド: 合成』(UG901) のこのセクションを参照してください。

RTL DRC の実行

HDL で発生する可能性のあるコードの問題を特定するための RTL DRC ルールがあります。これらのチェックは、Flow Navigator の [Open Elaborated Design] をクリックして開いたエラボレート済みデザインで実行できます。これらの DRC チェックを実行するには、Flow Navigator で [RTL Analysis] → [Report Methodology] をクリックするか、Tcl コマンド プロンプトで `report_methodology` を実行します。

クロッキング ガイドライン

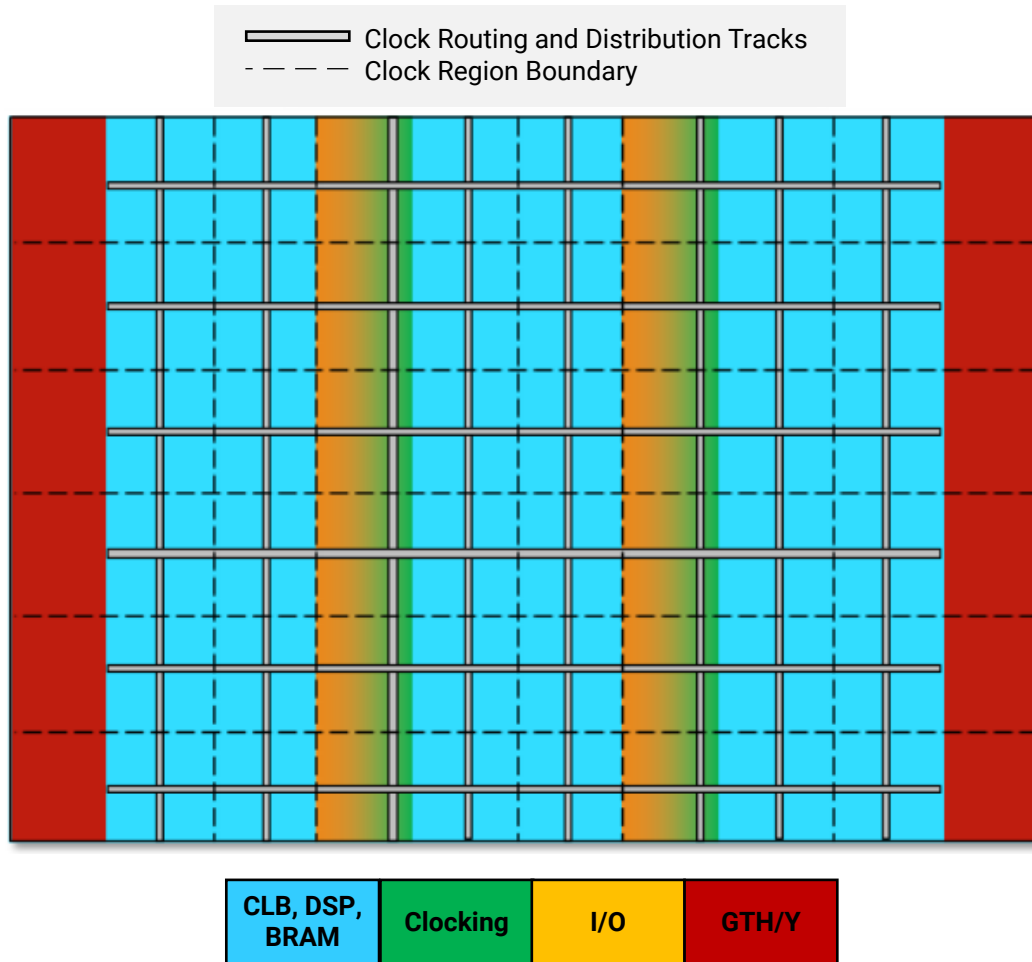
各デバイス アーキテクチャには、クロック専用リソースがいくつか含まれています。使用しているデバイス アーキテクチャのクロック リソースを理解しておくと、これらのリソースを最適に活用するようクロッキングをプランニングできます。ほとんどのデザインではこれらの詳細に注意する必要はありませんが、ユーザーが配置を制御でき、各クロック ドメインのファンアウトについて理解している場合は、クロッキングの次の詳細に基づいて、さまざまな方法を試すことができます。これらのクロック リソースのいずれかを使用する場合は、対応するクロック エLEMENT を明示的にインスタンスエートする必要があります。

UltraScale デバイスのクロッキング

UltraScale デバイスには、以前のデバイス アーキテクチャとは異なるクロッキング構造が含まれており、グローバルクロックとリージョナルクロックの違いがあいまいになっています。UltraScale デバイスには 7 シリーズのようにリージョナルクロックバッファはなく、ロードがローカル/リージョナルかグローバルかにかかわらず、共通のバッファおよびクロック配線構造が使用されます。

UltraScale デバイスには、より小さい固定サイズのクロック領域が含まれ、クロック領域の幅はデバイス幅の半分ではありません。各行のクロック領域の数は、UltraScale デバイスごとに異なります。各クロック領域には、24 本の垂直方向/水平方向の配線トラックと 24 本の垂直方向/水平方向の分配トラックに分割されるクロック ネットワーク配線が含まれます。次の図に、36 個のクロック領域 (6 列 x 6 行) を含むデバイスを表示します。同等の 7 シリーズ デバイスには、12 個のクロック領域 (2 列 x 6 行) が含まれます。

図 46: UltraScale デバイスのクロック領域タイル



X15241-122019

クロッキング アーキテクチャは、特定の配置のクロック バッファとロードを接続するのに必要なクロック リソースのみが使用されるように設計されるので、ロードのないクロック領域でリソースが無駄に使用されることはありません。クロック リソースが効率的に使用されると、アーキテクチャでより多くのデザインクロックをサポートできるようになるほか、パフォーマンスおよび消費電力のためにクロック特性も改善されます。クロック タイプおよび関連のクロック構造は、ドライバーおよび使用法によって次の主なカテゴリに分類されます。

- 高速 I/O クロック

これらのクロックは高速 SelectIO™ インターフェイスのビット スライス ロジックに関連付けられており、PLL により生成され、専用の低ジッターのリソースを介して高速 I/O インターフェイスのビット スライス ロジックに配線されます。通常、このクロッキング構造はメモリ IP や High Speed SelectIO Wizard などのザイリンクス IP で作成されて制御され、ユーザーは指定しません。

- 汎用クロック

これらのクロックは、ほとんどのクロック ツリー構造で使用され、GCIO パッケージ ピン、MMCM/PLL、またはファブリック ロジック セル (通常は推奨されない) により供給されます。汎用クロッキング ネットワークは、I/O 列を含むどのクロック領域でも使用可能な BUFGCE/BUFGCE_DIV/BUFGCTRL バッファで駆動する必要があります。各クロック領域で 24 個までの固有のクロックをサポートでき、ほとんどの UltraScale デバイスでクロック トポロジ、ファンアウト、ロード配置によって 100 個以上のクロック ツリーをサポート可能です。

- ギガビット トランシーバー (GT) クロック

ギガビット トランシーバー (GTH または GTY) の送信クロック、受信クロック、および基準クロックでは、GT を含むクロック領域の専用クロッキングが使用されます。GT クロックを使用すると、次を達成できます。

- BUFG_GT バッファを使用して汎用クロッキング ネットワークを駆動し、ファブリック内のロードを接続
- 同じまたは異なるクワッドの複数トランシーバー間でクロックを共有

クロック プリミティブ

ほとんどのクロックは、グローバル クロック 兼用 I/O (GCIO) ピンを介してデバイスに入力されます。これらのクロックは、クロック バッファを介してクロック ネットワークを直接駆動するか、I/O 列に隣接するクロック マネージメント タイル (CMT) にある PLL または MMCM により変換されます。

CMT には、次のクロック リソースが含まれます。

- クロック生成ブロック
 - 2 つの PLL
 - 1 つの MMCM
- グローバル クロック バッファ
 - 24 個の BUFGCE
 - 8 個の BUFGCTRL
 - 4 個の BUFGCE_DIV

注記: ボンディングされていない I/O を含む I/O 列に隣接する CMT 内のクロック リソースは、使用可能です。

GT ユーザー クロックは、BUFG_GT バッファを介してグローバル クロック ネットワークを駆動します。GTH/GTY 列に隣接するクロック領域ごとに 24 個の BUFG_GT バッファがあります。

次に、UltraScale デバイスの各クロック バッファについて簡単に説明します。

- BUFGCE

最もよく使用されるバッファです。クロック イネーブル/ディスエーブル 機能を持つ汎用クロック バッファで、7 シリーズの BUFGCE と同等のバッファです。
- BUFGCE_DIV

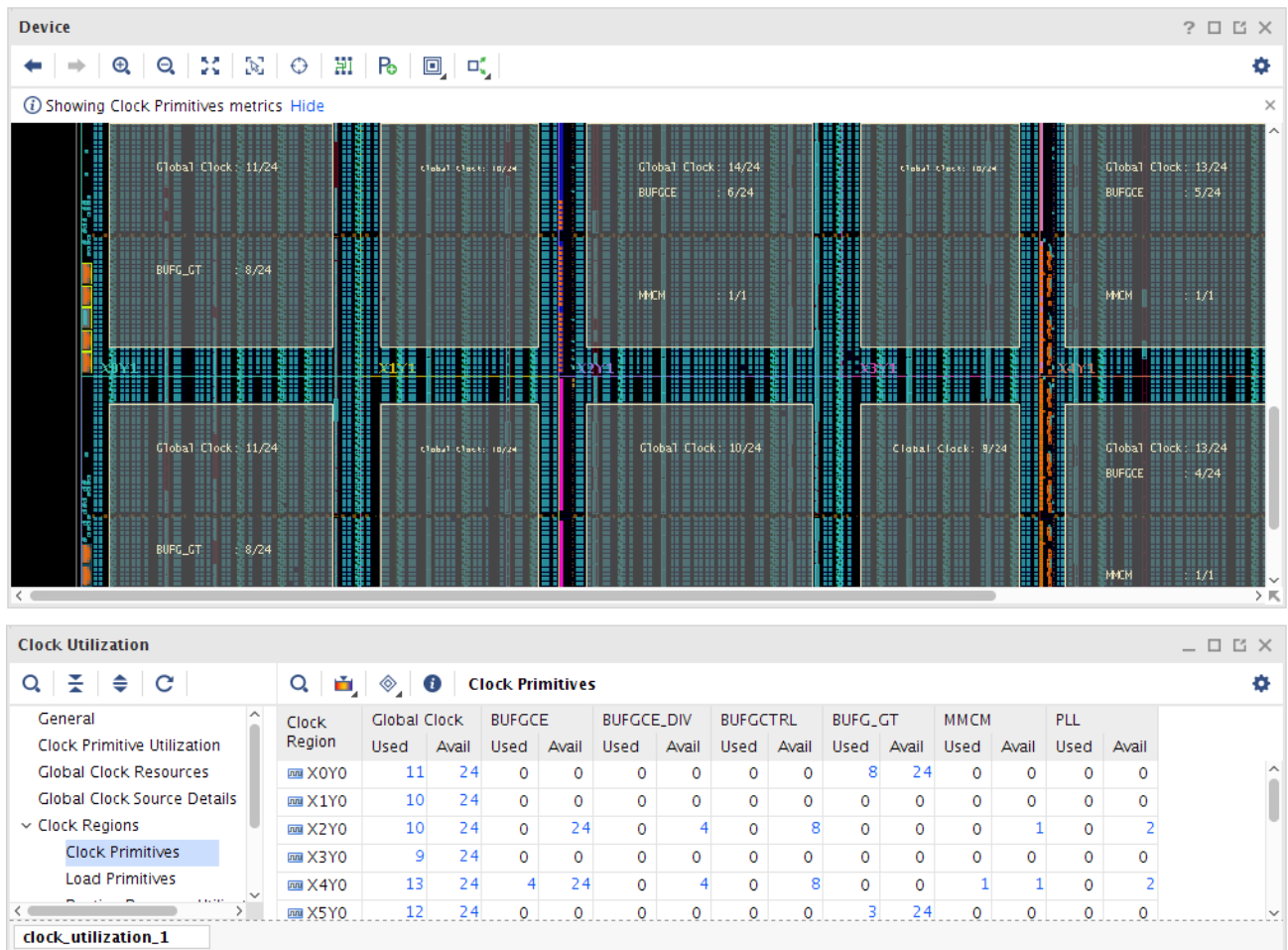
クロックの単純な分周が必要な場合に便利です。単純なクロック分周には、MMCM または PLL を使用するよりも BUFGCE_DIV を使用した方が簡単で消費電力も低くなります。また、正しく使用すれば、クロック乗せ換えにおいて、MMCM または PLL を使用するよりもクロック ドメイン間のスキューが小さくなることがあります。BUFGCE_DIV は、7 シリーズ デバイスの BUFR ファンクションの代わりによく使用されますが、グローバル クロック ネットワークを駆動できるので、BUFR コンポーネントよりも多くの機能があります。
- BUFGCTRL (および BUFGMUX)

BUFGCTRL は BUFGMUX としてインスタンス化でき、通常は複数のクロック ソースを 1 つのクロック ネットワークに多重化する際に使用されます。BUFGCE および BUFGCE_DIV と同様に、リージョナル クロッキングまたはグローバル クロッキングのいずれかのクロック ネットワークを駆動できます。
- BUFG_GT

GT で生成されたクロックを使用する場合、BUFG_GT クロック バッファを使用するとクロック ネットワークへの接続が可能になります。ほとんどの場合、BUFG_GT は 1 つまたは 2 つの隣接クロック領域にあるロードを駆動するリージョン バッファとして使用されます。BUFG_GT にはビルトインのダイナミック クロック分周機能があり、MMCM の代わりにクロック レートを変更するために使用できます。

Vivado IDE のクロック使用率レポートを使用して、クロック リソースの使用率およびクロック配置を視覚的に解析できます。次の図に、[Device] ウィンドウにクロック領域ごとのクロック リソース使用率を表示した例を示します。このレポートの詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906)を参照してください。

図 47: クロック使用率レポート



BUFGCE、BUFGCE_DIV、および BUFGCTRL の詳細は、『UltraScale アーキテクチャ クロッキング リソース ユーザー ガイド』(UG572: [英語版](#)、[日本語版](#))を参照してください。BUFG_GT バッファの接続および使用の詳細は、該当する UltraScale アーキテクチャ トランシーバー ユーザー ガイドを参照してください。

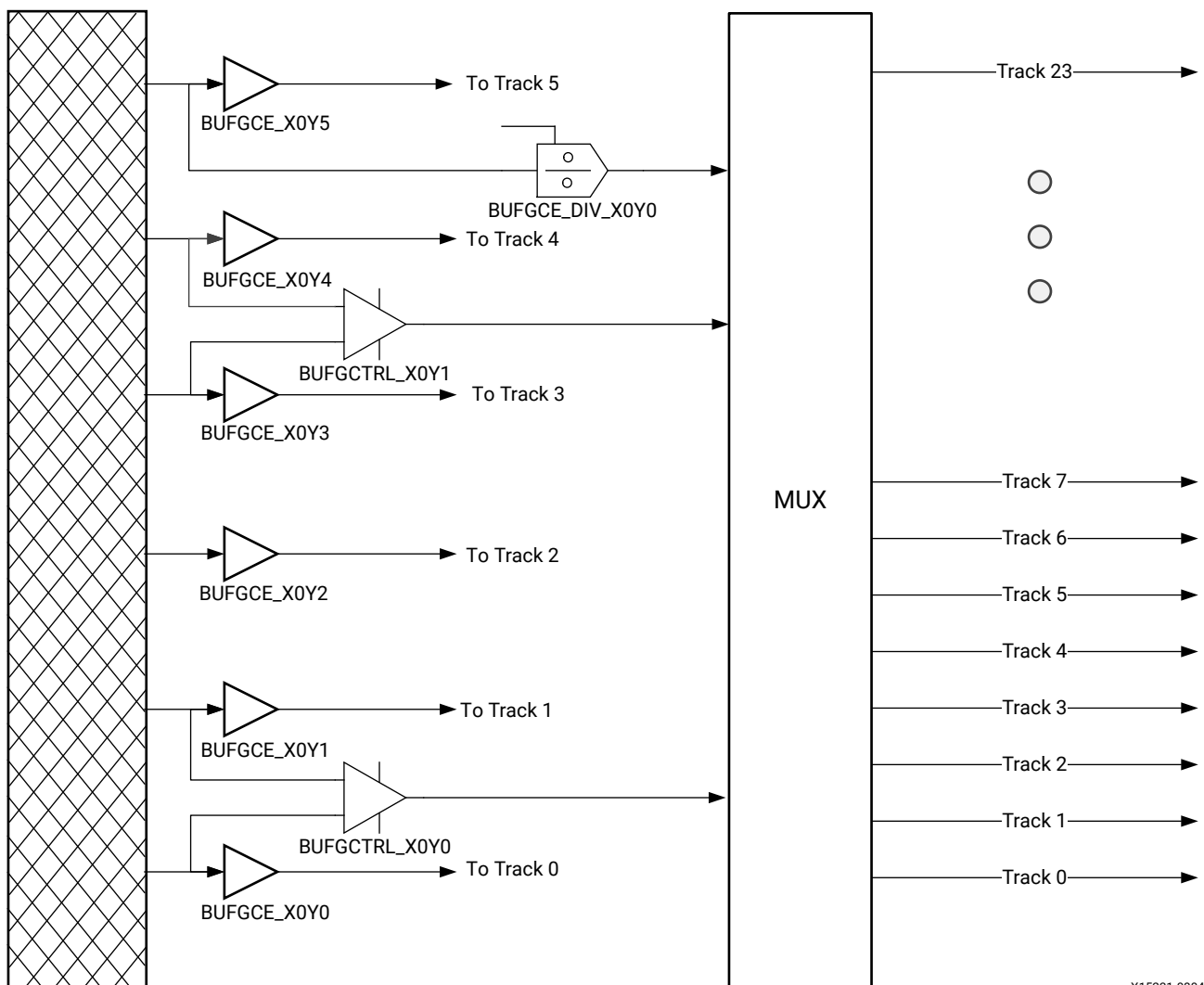
- 『UltraScale アーキテクチャ GTH トランシーバー ユーザー ガイド』(UG576: [英語版](#)、[日本語版](#))
- 『UltraScale アーキテクチャ GTY トランシーバー ユーザー ガイド』(UG578: [英語版](#)、[日本語版](#))

グローバル クロック バッファの接続および配線トラック

クロック領域の 24 個の BUFGCE バッファは、それぞれ特定のクロック配線トラックのみを駆動できますが、BUFGCTRL および BUFGCE_DIV 出力は MUX 構造を介して 24 本のトラックのどれでも使用できます。各 BUFGCE_DIV は特定の BUFGCE サイトと入力接続を共有し、各 BUFGCTRL は 2 つの特定の BUFGCE サイトと入力接続を共有します。このため、クロック領域で BUFGCE_DIV または BUFGCTRL バッファを使用する場合、BUFGCE バッファの使用が制限されます。次の図は、クロック領域の一番下の 6 つの BUFGCE がクロック領域内で 4 回複製されることを示しています。

注記: グローバル クロック ネットは、クロックが使用するすべての垂直方向配線、水平方向配線、および分配リソースに対して、デバイス内の特定のトラック ID に割り当てられます。クロックは、クロックが別のクロック バッファを介さない限り、トラック ID を変更できません。

図 48: BUFGCE、BUFGCE_DIV、BUFGCTRL の共有入力と MUX を介した出力



X15231-080420

クロックの配線、ルート、および分配

UltraScale デバイスのクロッキング機能とデザインのクロッキングの使用を正しく理解するには、クロック配線で専用配線リソースがどのように使用されるかを知っておくことが重要です。

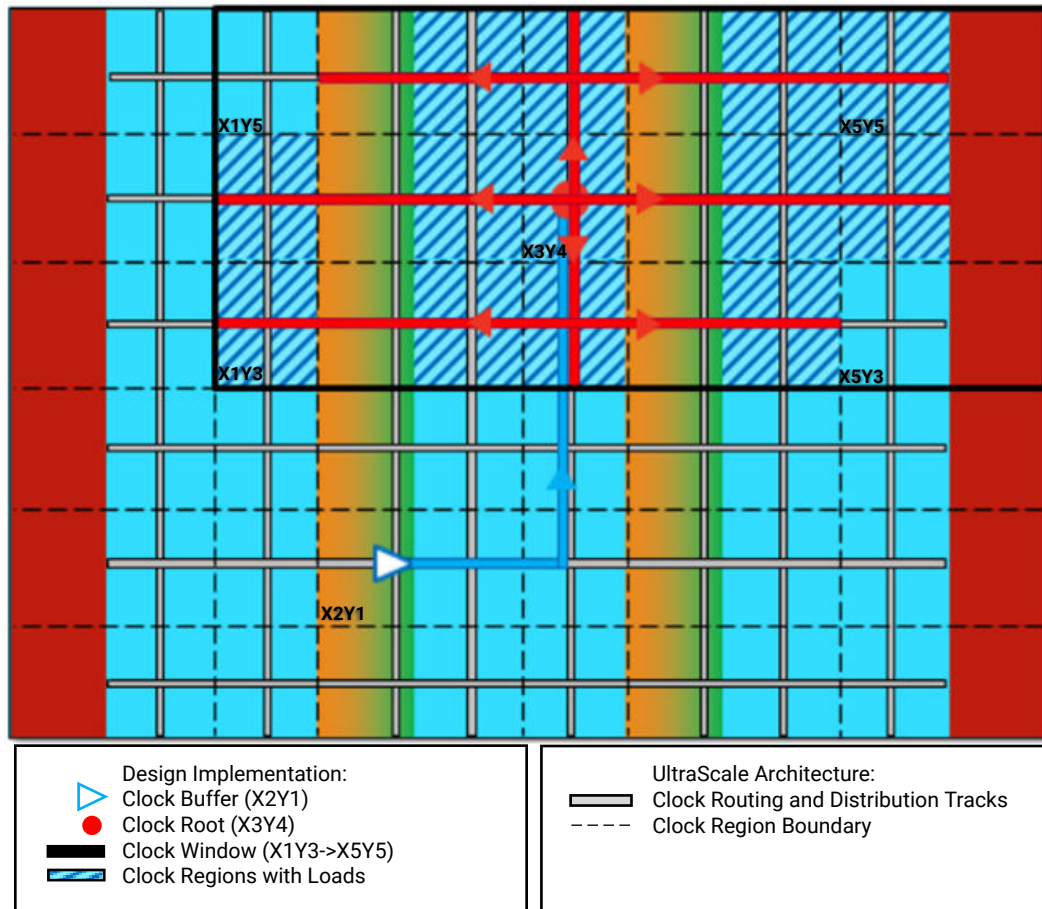
- クロック バッファからクロック ルートまでは、クロック信号は垂直方向配線および水平方向配線の 1 つまたは複数セグメントを通過します。各セグメントは、同じトラック ID (0 ~ 23) を使用する必要があります。
- クロック ルートでは、クロック信号が配線トラックから同じトラック ID の分配トラックに遷移します。クロック ルートは通常、スキューを削減するため、クロック ウィンドウの中央に配置されているクロック領域にあります。クロック ウィンドウは、クロック ネットのロードが配置されるクロック領域すべてを含む長方形のエリアです。スキュー最適化の理由から、Vivado IDE によりクロック ルートが中心からずらされることがあります。
- クロック信号は、クロック ルートからロードが配置されている CLB 列に向かって、まず垂直方向 (必要に応じてデバイスの上下両方の方向) に分配された後、水平方向 (必要に応じてデバイスの左右両方の方向) に分配されます。
- CLB 列は、水平方向分配リソースの上下にある 2 つの半分に分割されます。CLB 列の半分にはそれぞれ、水平方向の分配トラックのいずれかで到達可能な複数の最下位クロック配線リソースが含まれます。

場合によっては、クロック バッファでクロック分配トラックを直接駆動できます。これは通常、クロック ルートがクロック バッファと同じクロック領域にある場合、またはクロック バッファがクロック以外のピン (ファンアウトの大きいネットなど) のみを駆動する場合に発生します。

クロック配線リソースはセグメントに分割されるので、クロック領域を横切るため、またはクロック領域のロードに到達するために使用される配線セグメントおよび分配セグメントのみが消費されます。

次の図は、クロック領域 X2Y1 にあるクロック バッファがクロック ウィンドウ (長方形で囲まれた X1Y3 から X5Y5 のクロック領域) 内にあるロードに到達するところを示しています。

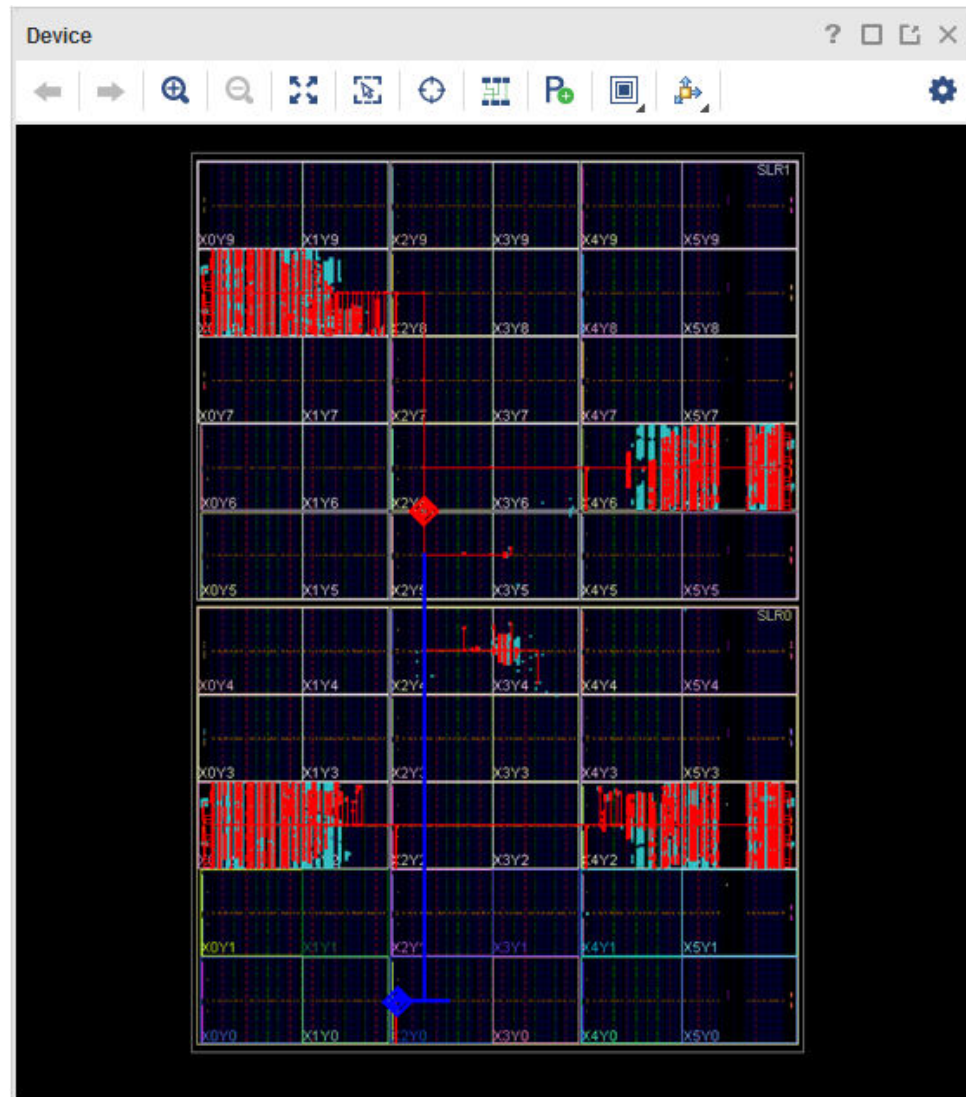
図 49: UltraScale デバイスでのドライバーからロードまでのクロック配線



X15389-120619

次の図の配線済みデバイス表示は、デバイスのほぼ全体に分配されるグローバル クロックの例を示しています。ネットワークを駆動するクロック バッファは、クロック領域 X2Y0 に青色で示されており、そのクロック領域内の横方向配線を駆動しています。この後ネットは水平方向配線からクロック領域 X2Y0 の垂直方向配線に遷移し、クロック領域 X2Y5 のクロック ルートに到達します。すべてのクロック配線は青色で示されています。クロック ルートは、クロック領域 X2Y5 に赤色で示されています。X2Y5 のクロック ルートからは、ネットが垂直分配に遷移した後、水平方向分配に遷移し、クロックの最下位ピンに到達します。分配レイヤーと CLB 列の最下位クロック配線リソースは、赤色で示されています。

図 50: 配線済みクロック ネットワークを示す [Device] ウィンドウ



クロック ツリーの配置配線

次の段階では、Vivado 配置で物理的な XDC 制約を保持しながら、MMCM/PLL、グローバル クロック バッファ、およびクロック ルートの配置が決定されます。

1. I/O およびクロック配置

配置ツールで接続規則とユーザー制約に基づいて I/O バッファと MMCM/PLL が配置されます。配置ツールでは、クロック バッファがクロック領域に割り当てられますが、LOC プロパティを使用して制約していない場合、個別のサイトには割り当てられません。クロック以外のロードのみを駆動するクロック バッファのみが、フローの後の段階で、そのドライバーとロードの配置に基づいて別のクロック領域に移動可能です。

この段階の配置ツール エラーは、接続規則とユーザー制約のいずれかまたは両方が競合するために発生します。ログ ファイルにエラーの根本的な原因と考えられる要因に関する情報が表示されるので、詳細に確認してデザインまたは制約を適切に変更する必要があります。

2. SLR 分割 (SSI テクノロジ デバイスのみ) およびグローバル配置

配置ツールでは、早期のドライバーおよびロードの配置に基づいて、初期クロック ツリー インプリメンテーションが実行されます。各クロック ネットはクロック ウィンドウに関連付けられます。クロック ウィンドウが過剰に重なっている場合、クロック配線の競合のため、配置ツール エラーが発生する可能性があります。

クロック分割エラーが発生した場合、ログ ファイルに各クロック ネットの最後のクロック バジレット ソリューションと、各クロック領域に存在する固有のクロック ネットの数が表示されます。ログ ファイルを詳細に確認し、過剰に使用されているクロック領域からどのクロックを削除するかを判断します。クロックを削除するには、次の手法を使用できます。

- 同じ同期クロックどうしを組み合わせるか、不要な MMCM フィードバック クロックを削除するか、ファンアウトの小さいクロックをファンアウトの大きいクロックにまとめて、デザインのクロック数を削除します。
- クロック プリミティブを別のクロック領域、特に接続ベースの配置規則のない領域に移動します。
- クロック ロードにフロアプラン制約を追加して、ファンアウトの小さいクロックをそれらのドライバー近くに配置するか、使用率の高いクロック領域から離します。

配置ツールによりクロック ツリー インプリメンテーションが複数回実行されて調整され、タイミングの QoR (結果の品質) が向上されます。たとえば、配置最適化の後の方の段階では、困難なクロックが解析され、より適したクロック ルートの位置が決定されます。

3. クロック ツリーの事前配線

配置ツールで後続のインプリメンテーション段階がガイドされ、配置後のタイミング解析用に正確な遅延が見積もられます。

配置後は、Vivado ツールでクロック ツリー インプリメンテーションを次のように変更できます。

- Vivado 物理最適化では、セルを複製して、関連クロックのないクロック領域に移動できます。
- Vivado 配線では、タイミング QoR (結果の品質) を改善してクロック配線を有効にするための調整を実行できます。

次の表に、主なクロック トポロジの配置規則と、制約がこれらの規則に及ぼす影響を示します。

表 2: 配置規則のある場合とない場合のトポロジ

制約付きソース	制約なしのデスティネーション	結果
GCIO	BUFGCE、BUFGCTRL、BUFGCE_DIV、PLL/MMCM	同じクロック領域に自動的に配置されます。
PLL/MMCM	BUFGCE、BUFGCTRL、BUFGCE_DIV	同じクロック領域に自動的に配置されます。
GT*_CHANNEL	BUFG_GT	同じクロック領域に自動的に配置されます。
BUFGCTRL	BUFGCTRL	同じクロック領域に自動的に配置されます。 注記: 同じクロック領域内の配置は、CLOCK_REGION 制約を使用して無効にできます。
BUFG*	BUFG*	制約が設定されていないデスティネーション BUFG は予測不可能な配置になります。 デスティネーション BUFG* に CLOCK_REGION 制約を設定することを推奨します。 注記: BUFGCTRL > BUFGCTRL は例外です。

表 2: 配置規則のある場合とない場合のトポロジ (続き)

制約付きソース	制約なしのデスティネーション	結果
BUFG*	MMCM/PLL	<p>制約が設定されていないデスティネーション MMCM/PLL は予測不可能な配置になります。</p> <p>MMCM/PLL に LOC 制約を設定することを推奨します。</p> <p>配線が隣接するクロック領域または複数のクロック領域にまたがる場合は、CLOCK_DEDICATED_ROUTE 制約を推奨します。</p>

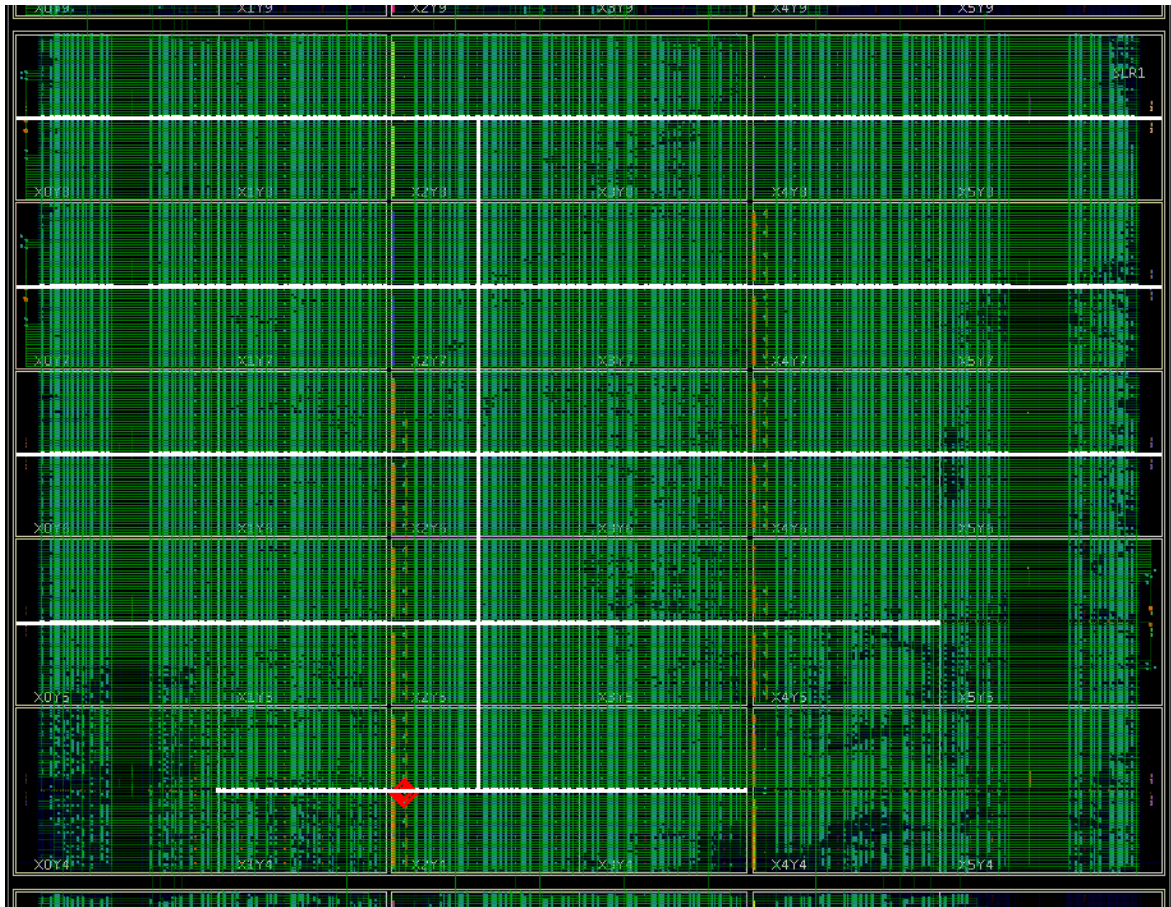
クロッキング機能

クロック プランニングは、ターゲット デバイスのファンアウトの大きいクロックとファンアウトの小さいクロックの総数に基づいて実行する必要があります。

ファンアウトの大きいクロック

ファンアウトの大きいクロックは、SSI テクノロジ デバイスの SLR のほとんど全体に分配されるか、モノリシック デバイスのほとんどすべてのクロック 領域に分配されます。次の図に、SLR のほぼ全体に分配されるファンアウトの大きいクロックを示します。BUFGCE ドライバーは赤色で示しています。

図 51: SLR 全体に分配されるファンアウトの大きいクロック



注記: デザインで使用されるクロックが 24 個を超える場合は、特別なデザイン考慮事項やその他の事前プランニングが必要となるような問題が発生することがあります。



重要: ZHOLD および BUF_IN 補正モードでは、MMCM フィードバック クロック パスは、配線トラック、クロック ルートの位置、分配トラックなどが CLKOUT0 クロック パスと同じになります。そのため、クロック バッファとクロック ルートが離れている場合、フィードバック クロックはファンアウトの大きいクロックと考えられます。

関連情報

[MMCM ZHOLD/BUF_IN 補正を使用した I/O タイミング](#)

ファンアウトの小さいクロック

ファンアウトの小さいクロックは、ほとんどの場合、水平方向に隣接する 3 つ以下のクロック領域に配置された 5,000 個未満のクロック ピンに接続されるクロック ネットです。クロック配線、クロック ルート、およびクロック分配は、すべて限られたエリア内に含まれます。

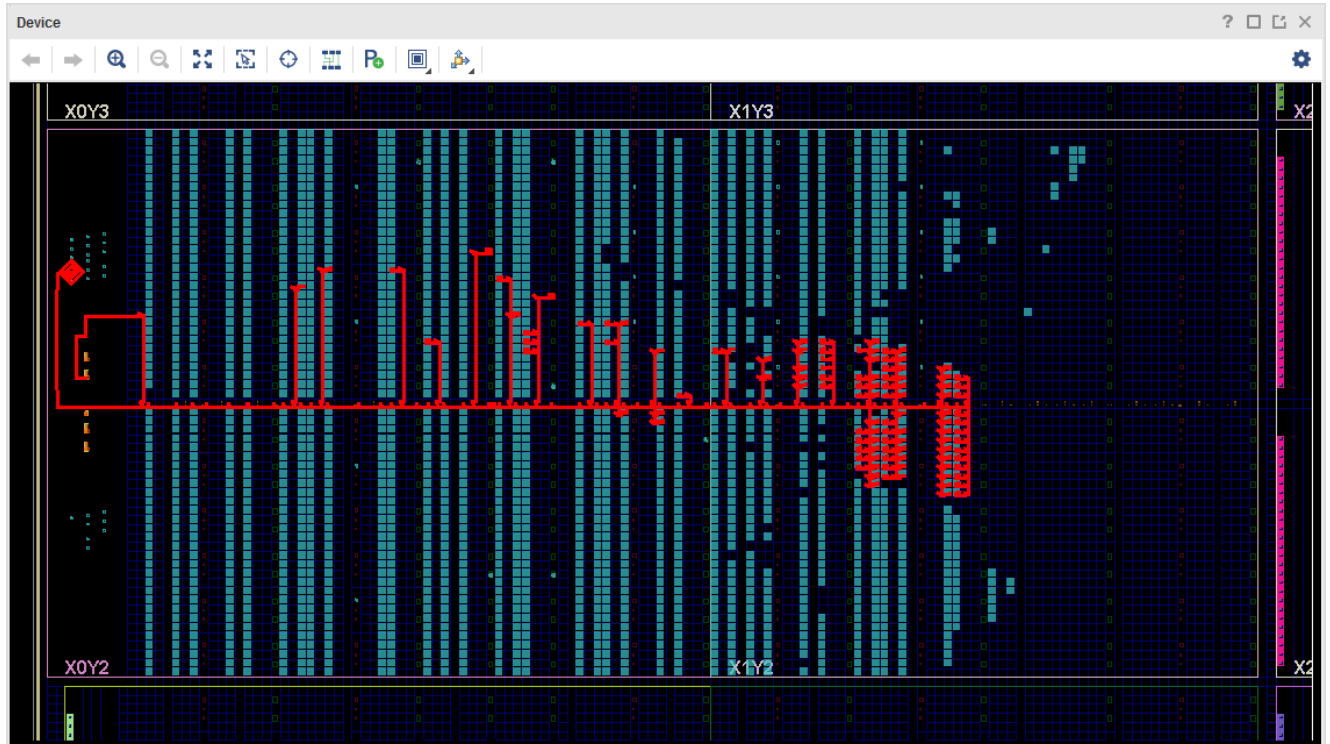
配置ツールでは、ファンアウトの小さいクロックが特定されるはずですが、できないこともあります。これは、デザイン サイズ、デバイス サイズ、または LOC 制約や Pblock などの物理的な XDC 制約が原因で、配置ツールでロードをローカル エリアに配置できないことがあるからです。この問題を修正するには、手動で Pblock を作成して、既存の物理制約を変更する必要があることがあります。

BUFG_GT で駆動されるクロックは、ファンアウトの小さいクロックの例です。これらのクロック ネットは Vivado 配置ツールで自動的に認識され、ロードが GT インターフェイスに隣接するクロック領域に含まれます。次の図に、2つのクロック領域に含まれるファンアウトの小さいクロックを示します。BUFG_GT ドライバーは赤色で示されています。



ヒント: ファンアウトの小さいクロックを 1つのクロック領域に含めるには、CLOCK_LOW_FANOUT XDC 制約を使用できます。

図 52: 2つのクロック領域に含まれるファンアウトの小さいクロック



関連情報

[CLOCK_LOW_FANOUT 制約の使用](#)

ファンアウトの大きいクロックと小さいクロックのバランス

UltraScale デバイスでは、それ以前のザイリンクス デバイス ファミリよりも多くのクロックがサポートされており、次のようなさまざまなクロッキング手法を使用できます。

- 24 個以下のクロック

競合するユーザー制約がない場合、配置または配線が競合するリスクなしに、すべてのクロックをファンアウトの大きいクロックとして扱うことができます。

- ほぼ 300 個のクロック

6つのクロック領域を含むデバイスをターゲットにしており、各クロックが3つまでのクロック領域に含まれるファンアウトの小さいクロックしか含まれないデザインでは、6行 x 各行 2 クロック ウィンドウ x 各領域 24 クロック = 288 個のクロックが必要となります。

ファンアウトの小さいクロックのウィンドウには決まったサイズはありませんが、通常は 1 ～ 3 クロック領域です。ファンアウトの大きいクロックがデバイス全体や SLR 全体に分配されることはまれです。

次の手法は、ファンアウトの大きいクロックと小さいクロックのバランスを取る方法を示しています。ファンアウトの小さいクロックのうち数個は I/O インターフェイスから、ほとんどは GT インターフェイスからのものであると想定しています。SSI テクノロジ デバイスの各 SLR にも同じ手法を使用できます。

- ファンアウトの大きいクロック
 - モノリシック デバイスの場合は最大 12 個
 - SSI テクノロジ デバイスの場合は最大 24 個 (ファンアウトの大きいクロックの一部は 1 つの SLR 内にのみ存在すると想定した場合)
- ファンアウトの小さいクロック
 - 最大 12 個 + GT を使用するクワッドごとに 8 個
 - または最大 12 個 + GT インターフェイス (RXUSRCLK および TXUSRCLK を共有する GT チャンネルのグループ) ごとに 6 個

クロック制約

物理 XDC 制約は、クロック ツリーのインプリメンテーションを指示し、ファンアウトの大きいクロック リソースの使用を制御します。UltraScale デバイスのクロッキングはそれ以前のアーキテクチャのクロッキングよりも柔軟性があり、より多くのアーキテクチャ制約が含まれるので、インプリメンテーションのためにクロックを正しく制約する方法を理解しておくことが重要です。

IO/MMCM/PLL/GT の LOC 制約

クロックを制約するには、配置制約を次のように設定できます。

- I/O ポートのクロック入力に設定

GCIO のクロックに PACKAGE_PIN 制約を割り当てたり、IOB に LOC を割り当てたりすると、クロック ネットワークに影響します。入力ポートに直接接続される MMCM/PLL およびクロック バッファは、同じクロック領域に配置する必要があります。
- MMCM または PLL に設定

MMCM または PLL 出力に直接接続されるクロック バッファと MMCM または PLL 入力に接続される入力クロック ポートは、同じクロック領域に自動的に配置されます。入力クロック ポートと MMCM または PLL が直接接続されており、別のクロック領域に制約されている場合、クロック バッファを手動で挿入し、MMCM または PLL に接続されたネットに CLOCK_DEDICATED_ROUTE 制約を設定する必要があります。
- GT*_CHANNEL または IBUFDS_GT* セルに設定

これらのセルで駆動される BUFG_GT は、同じクロック領域に配置されます。



注意: ザイリンクスでは、クロック バッファ セルに LOC 制約を使用することはお勧めしません。クロックが特定のトラック ID に指定され、配線できない配置になることがあります。UltraScale デバイスでは、デザインのクロック ツリー全体を理解していて、配置がデザインで一貫している場合に、ファンアウトの大きいバッファを配置するためにのみ LOC 制約を使用してください。このように注意を払っても、デザインや制約の変更により、インプリメンテーションで競合が発生することがあります。

クロック バッファーでの CLOCK_REGION プロパティの使用

CLOCK_REGION 制約を使用すると、クロック バッファーをサイトを指定せずにクロック領域に割り当てることができます。この制約を使用すると、すべてのクロック ツリーを最適化する際、およびすべてのクロックを配線できるようにするために適切なバッファー サイトを決定する際に、配置がより柔軟に実行されるようになります。

CLOCK_REGION 制約は、カスケード クロック バッファーや、ファブリック ロジックなどのクロック以外のプリミティブで駆動されるクロック バッファーの配置にガイドラインを提供するために使用することもできます。

次のコマンド例では、XDC 制約により clkgen/clkout2_buf クロック バッファーが CLOCK_REGION X2Y2 に割り当てられます。

```
set_property CLOCK_REGION X2Y2 [get_cells clkgen/clkout2_buf]
```

注記: ほとんどの場合、クロック バッファーは、既にクロック領域に制約された入力クロック ポート、MMCM、PLL、または GT*_CHANNEL で直接駆動されます。この場合、クロック バッファーが自動的に同じクロック領域に配置されるので、CLOCK_REGION 制約を使用する必要はありません。

Pblock を使用してクロック バッファーの配置を制限

クロック バッファーを特定のクロック領域に配置する必要がない場合は、Pblock を使用してクロック領域の範囲を指定できます。たとえば、別のエリアにある 2 つのクロックを多重化するために BUFGCTRL が必要な場合に Pblock を使用します。BUFGCTRL を 2 つのクロック ドライバー間のクロック領域を含む Pblock に割り当て、有効な配置が見つけられるようにできます。

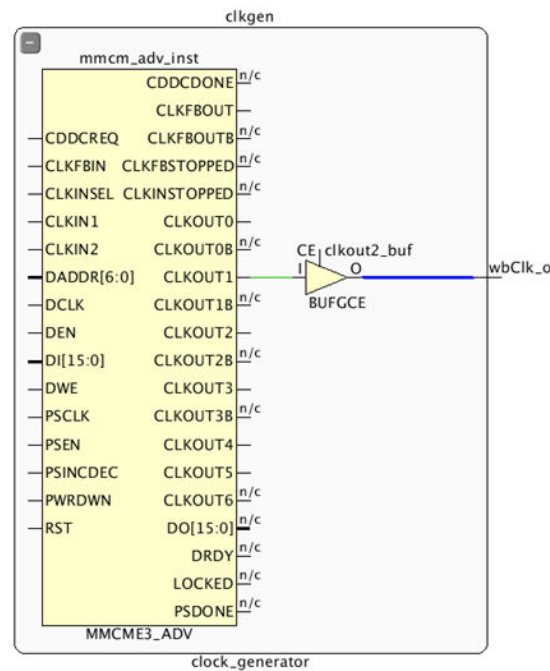
注記: ザイリンクスでは、1 つのクロック領域に Pblock を使用することはお勧めしません。

クロック ネットへの USER_CLOCK_ROOT プロパティの使用

USER_CLOCK_ROOT プロパティを使用すると、クロック バッファーで駆動されるクロックのクロック ルートの位置を指定できます。USER_CLOCK_ROOT プロパティを指定すると、クロック配線が変更されて挿入遅延とスキューの両方に影響するため、デザインの配置に影響します。USER_CLOCK_ROOT 値はクロック領域に対応するので、このプロパティはファンアウトの大きいクロック バッファーで直接駆動されるネット セグメントに設定する必要があります。次に例を示します。

```
set_property USER_CLOCK_ROOT X2Y3 [get_nets clkgen/wbClk_o]
```

図 53: クロック バッファで駆動されるネット セグメントに適用された USER_CLOCK_ROOT



配置後は、次の例に示すように、CLOCK_ROOT を使用して実際のクロック ルートをクエリできます。CLOCK_ROOT に対しては、割り当てられたルートがユーザーによって割り当てられたものであるか、Vivado ツールで自動的に割り当てられたものであるかがレポートされます。

```
get_property CLOCK_ROOT [get_nets clkgen/wbClk_o]
=> X2Y3
```

インプリメントしたデザインのクロック ルート割り当てを確認するには、Tcl コマンドの report_clock_utilization を使用する方法もあります。次に例を示します。

```
report_clock_utilization [-clock_roots_only]
```

次の図に、このレポートを示します。

図 54: report_clock_utilization によるクロック ルート割り当ての確認

Index	Clock Net	Root Clock Region
1	clkgen/clkfbout_buf	X4Y1
2	clkgen/cpuClk_o	X4Y1
3	clkgen/fftClk_o	X3Y2
4	clkgen/phyClk0_o	X3Y3
5	clkgen/phyClk1_o	X3Y2
6	clkgen/usbClk_o	X3Y3

複数のクロック ネットでの CLOCK_DELAY_GROUP 制約の使用

CLOCK_DELAY_GROUP 制約を使用すると、異なるクロック バッファで駆動される複数の関連するクロック ネットワークの挿入遅延を同じにできます。この制約は、同じ MMCM、PLL、または GT ソースからのクロック間の同期 CDC タイミング パスのスキューを最小限に抑えるためによく使用されます。CLOCK_DELAY_GROUP 制約は、クロック バッファに直接接続されたネット セグメントに設定する必要があります。次に、クロック バッファで直接駆動されるクロック ネット、clk1_net および clk2_net に設定する例を示します。

```
set_property CLOCK_DELAY_GROUP grp12 [get_nets {clk1_net clk2_net}]
```

関連情報

[同期 CDC](#)

CLOCK_DEDICATED_ROUTE 制約の使用

CLOCK_DEDICATED_ROUTE 制約は、あるクロック領域のクロック バッファで別のクロック領域の MMCM または PLL を駆動する場合に通常使用されます。デフォルトでは、CLOCK_DEDICATED_ROUTE 制約は TRUE に設定され、バッファと MMCM または PLL のペアは同じクロック領域に配置されます。

注記: UltraScale デバイスで BACKBONE に 7 シリーズの次の CLOCK_DEDICATED_ROUTE 値を使用すると、SAME_CMT_COLUMN と同じ動作になります。

次の表に、CLOCK_DEDICATED_ROUTE 制約の値、使用法、および動作を示します。

表 3: UltraScale デバイス用 CLOCK_DEDICATED_ROUTE 制約のサマリ

値	使用法	結果
TRUE	クロック ネットのデフォルト値	グローバル クロック バッファおよび MMCM/PLL を同じクロック領域に配置する必要があります。 ネットがグローバル クロック リソースのみを使用して配線されます。
SAME_CMT_COLUMN (BACKBONE)	グローバル クロック バッファで駆動されるネット 例: <pre>set_property CLOCK_DEDICATED_ROUTE SAME_CMT_COLUMN \ [get_nets -of [get_pins BUFGCE_inst/O]]</pre>	MMCM/PLL を同じ垂直列のクロック領域に配置する必要があります。 ネットがグローバル クロック リソースのみを使用して配線されます。 最適な結果を得るには、サイリンクスでは MMCM/PLL に LOC 制約を使用して MMCM/PLL を同じ垂直列に配置することをお勧めします。

表 3: UltraScale デバイス用 CLOCK_DEDICATED_ROUTE 制約のサマリ (続き)

値	使用法	結果
ANY_CMT_COLUMN	<p>グローバル クロック バッファで駆動されるネット 例:</p> <pre>set_property CLOCK_DEDICATED_ROUTE ANY_CMT_COLUMN \ [get_nets -of [get_pins BUFGCE_inst/O]] set_property CLOCK_DEDICATED_ROUTE ANY_CMT_COLUMN \ [get_nets -of [get_pins BUFGCE_DIV_inst/O]] set_property CLOCK_DEDICATED_ROUTE ANY_CMT_COLUMN \ [get_nets -of [get_pins BUFGCTRL_inst/O]]</pre>	<p>MMCM/PLL は、使用可能なリソースのあるどのクロック領域にでも配置できます。</p> <p>ネットがグローバル クロック リソースのみを使用して配線されます。</p> <p>最適な結果を得るには、ザイリンクスでは MMCM/PLL に LOC 制約を使用して MMCM/PLL をデバイス内に配置することをお勧めします。</p>
FALSE	<p>グローバル クロック バッファでは駆動されないがクロック ネットワークの一部であるクロック ネット (IBUF で駆動されるネット、MMCM の出力クロック ピンに直接接続されているネットなど) 例:</p> <pre>set_property CLOCK_DEDICATED_ROUTE FALSE \ [get_nets -of [get_pins MMCME4_ADV_inst/CLKOUT0]] set_property CLOCK_DEDICATED_ROUTE FALSE \ [get_nets -of [get_pins IBUF_inst/O]]</pre>	<p>ネットがファブリックおよびグローバル クロック リソースを使用して配線されます。</p> <p>クロック ネットワークのタイミングおよびパフォーマンスに悪影響を与える可能性があります。</p> <hr/> <p>重要: UltraScale デバイスでは、特別なデザイン要件によりグローバル クロック リソースをファブリック リソースに配置する必要がある場合以外は、FALSE に設定しないでください。</p>

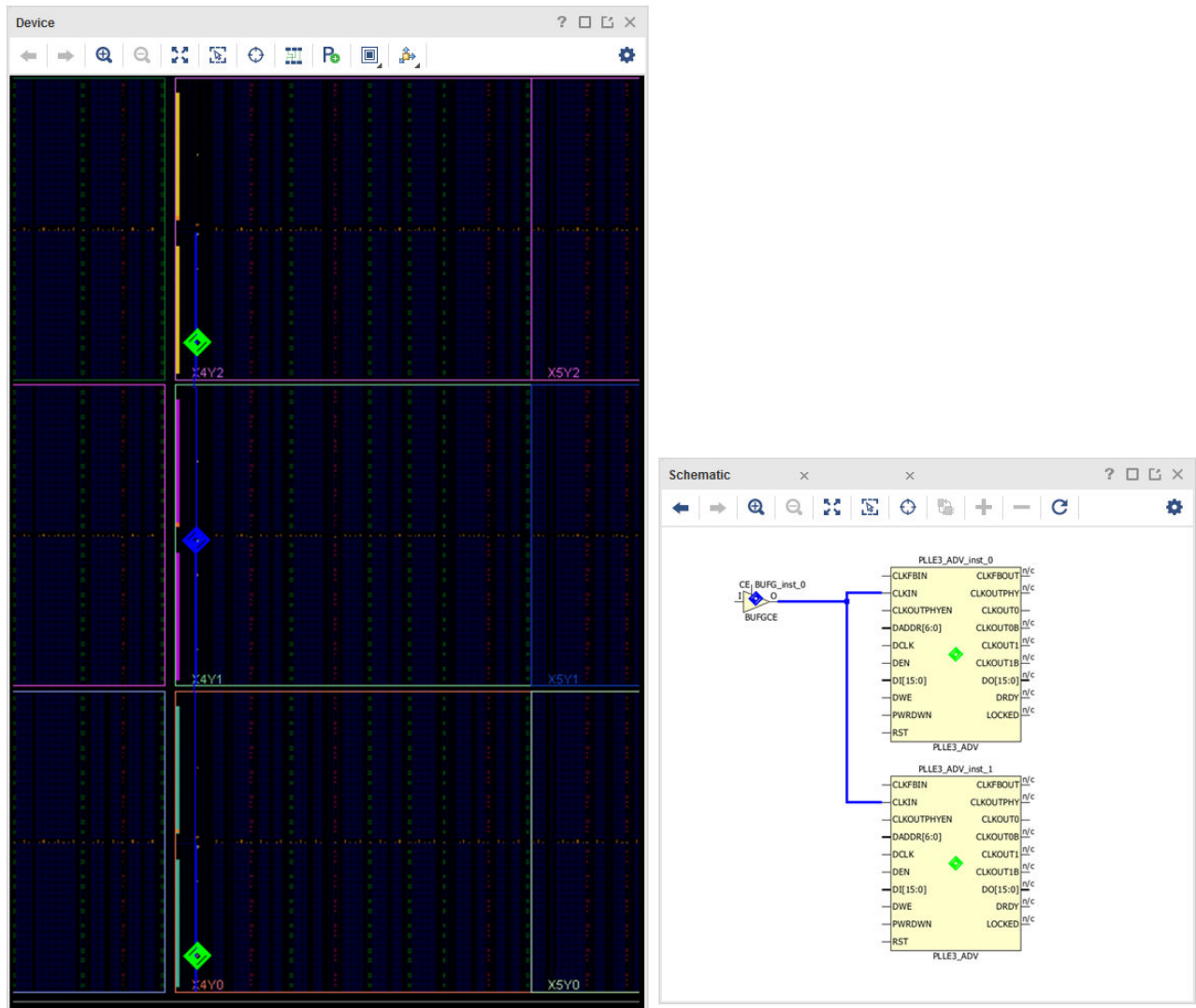
注記: UltraScale デバイスでは、直接ポートで駆動されるネットには CLOCK_DEDICATED_ROUTE プロパティは適用しないでください。その代わりに、IBUF の出力に CLOCK_DEDICATED_ROUTE プロパティを適用してください。

上下に隣接するクロック領域の制約例

1 つのクロック領域のクロック バッファで上下に隣接するクロック領域の MMCM または PLL を駆動する場合は、CLOCK_DEDICATED_ROUTE を 7 シリーズ デバイスでは BACKBONE に、UltraScale デバイスでは SAME_CMT_COLUMN に設定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 エラーが発生しなくなり、クロックがグローバル クロック リソースのみを使用して配線されるようになります。次の例および図に、垂直方向に隣接したクロック領域にある 2 つの PLL を駆動するクロック バッファを示します。

```
set_property CLOCK_DEDICATED_ROUTE SAME_CMT_COLUMN [get_nets -of [get_pins BUFG_inst_0/O]]
set_property LOC PLLE3_ADV_X0Y0 [get_cells PLLE3_ADV_inst_0]
set_property LOC PLLE3_ADV_X0Y4 [get_cells PLLE3_ADV_inst_1]
```

図 55: CLOCK_DEDICATED_ROUTE 制約を SAME_CMT_COLUMN に設定

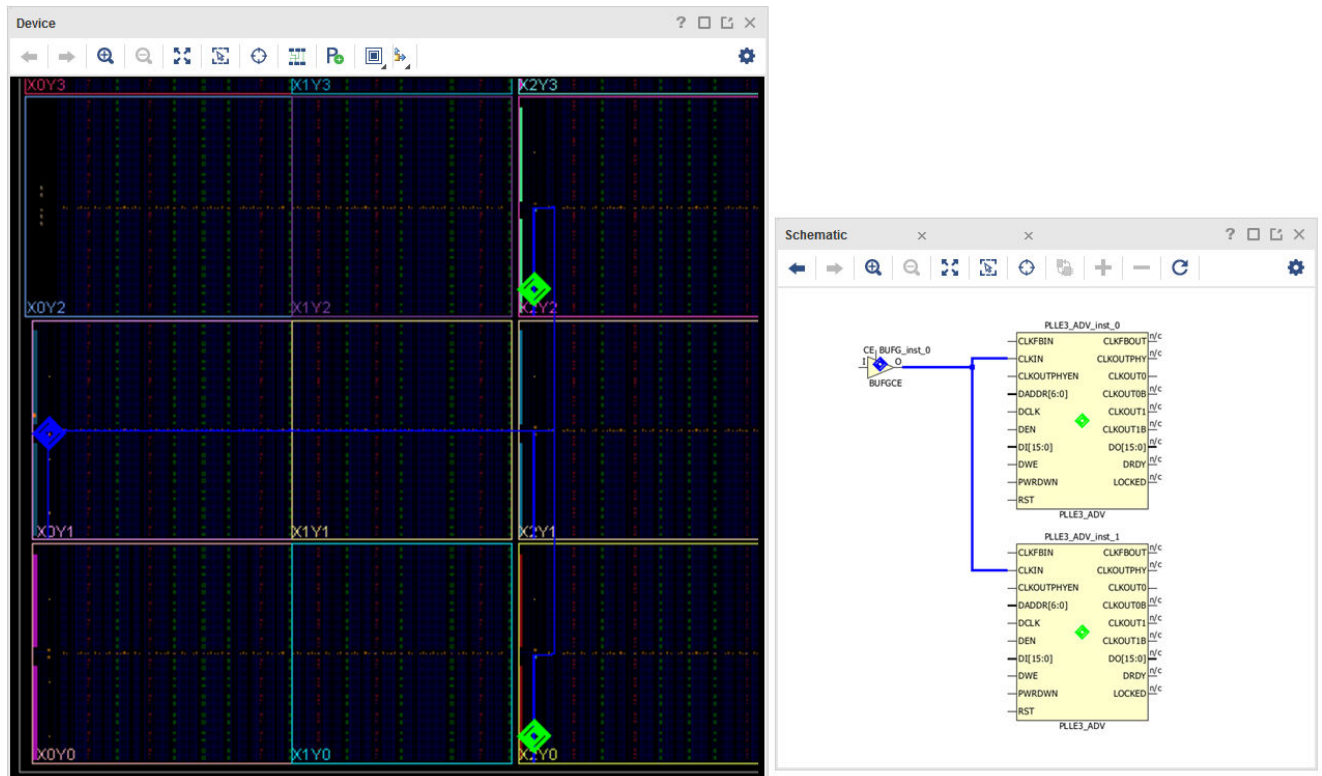


上下に隣接していないクロック ネットの制約例

1つのクロック領域のクロック バッファで上下に隣接していないクロック領域を駆動する場合は、CLOCK_DEDICATED_ROUTE を 7 シリーズ デバイスでは FALSE に、UltraScale デバイスでは ANY_CMT_COLUMN に設定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 エラーが発生しなくなり、クロックがグローバル クロック リソースのみを使用して配線されるようになります。次の例および図に、BUFGCE が入力バッファと同じ列にないクロック領域に配置されている 2 つの PLL を駆動しているところを示します。

```
set_property CLOCK_DEDICATED_ROUTE ANY_CMT_COLUMN [get-nets -of [get-pins BUFG_inst_0/O]]
set_property LOC PLLE3-ADV-X1Y0 [get-cells PLLE3-ADV_inst_0]
set_property LOC PLLE3-ADV-X1Y4 [get-cells PLLE3-ADV_inst_1]
```

図 56: CLOCK_DEDICATED_ROUTE 制約を ANY_CMT_COLUMN に設定



CLOCK_LOW_FANOUT 制約の使用

クロック バッファのロードを 1 つのクロック領域に含めるには、CLOCK_LOW_FANOUT 制約を使用できます。CLOCK_LOW_FANOUT 制約は、グローバル クロック バッファで直接駆動されるクロック ネット セグメントまたはフリップフロップに設定できます。

注記: CLOCK_LOW_FANOUT 制約の優先順位は、ほかのクロック制約よりも低くなります。CLOCK_LOW_FANOUT が USER_CLOCK_ROOT、CLOCK_DELAY_GROUP、CLOCK_DEDICATED_ROUTE などのクロック制約と競合する場合、CLOCK_LOW_FANOUT は無視されます。

フリップフロップの制約例

CLOCK_LOW_FANOUT 制約をグローバル クロック バッファで直接駆動されるフリップフロップのリストに設定すると、フリップフロップを隔離するため opt_design により新しい並列グローバル クロック バッファが作成されます。place_design の実行中、新しく作成された並列グローバル クロック バッファで駆動される隔離されたフリップフロップは、1 つのクロック領域に含まれます。

次の例では、クロック ゲーティング同期化回路の一部として使用されるフリップフロップのリストに CLOCK_LOW_FANOUT 制約を設定し、グローバル クロック バッファのクロック イネーブルを制御しています。

```
set_property CLOCK_LOW_FANOUT TRUE [get_cells safeClockStartup-reg[*]]
```

このデザインでは最初、alwaysOn クロック ネットワークが、ほかのロジックのクロック ゲーティングに使用するクロック ゲーティング同期化回路の一部であるフリップフロップを含め、2000 個を超えるロード (フリップフロップを含む) を駆動していました。次の回路図に、opt_design を使用して新しい並列グローバル クロック バッファを作成してクロック ゲーティング同期化回路を隔離する前と後の、alwaysOn クロック ネットワークに接続されたクロック ゲーティング同期化回路と追加のロジックを示します。

図 57: CLOCK_LOW_FANOUT をフリップフロップに適用する opt_design 変換前の回路図

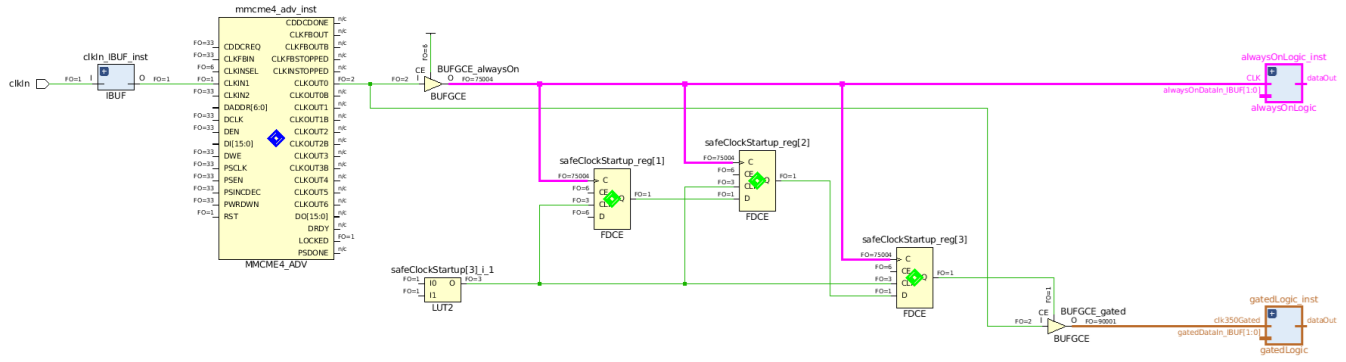
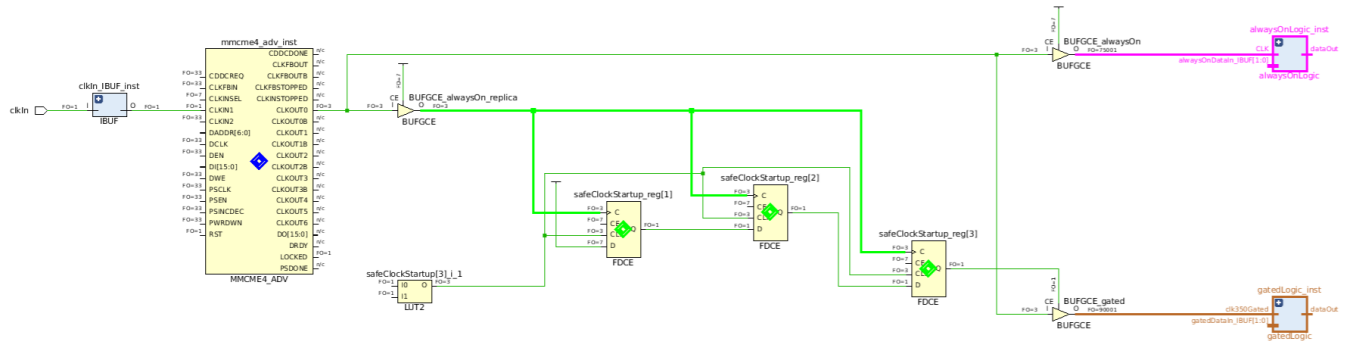
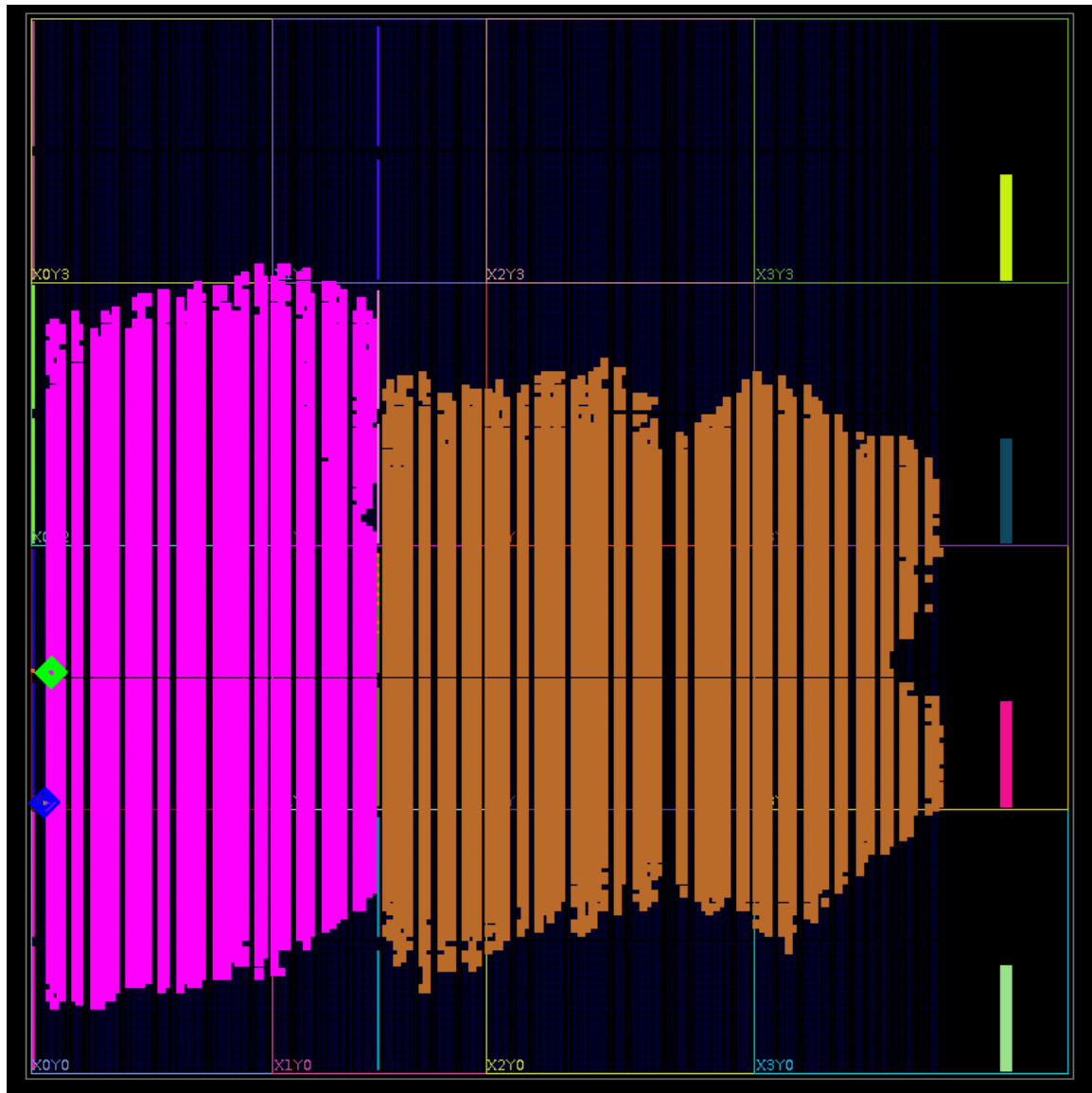


図 58: CLOCK_LOW_FANOUT をフリップフロップに適用する opt_design 変換後の回路図



完全にインプリメントされたデザインの [Device] ウィンドウに、緑色で示されたクロック ゲーティング同期化回路と、alwaysOn ロジックおよびクロック ゲーティングされたロジックが示されます。クロック ゲーティング同期化回路は、MMCM と同じクロック領域の、グローバル クロック バッファの近くに配置されます。

図 59: クロック ゲーティング同期化回路の配置を示した完全にインプリメントされたデザイン



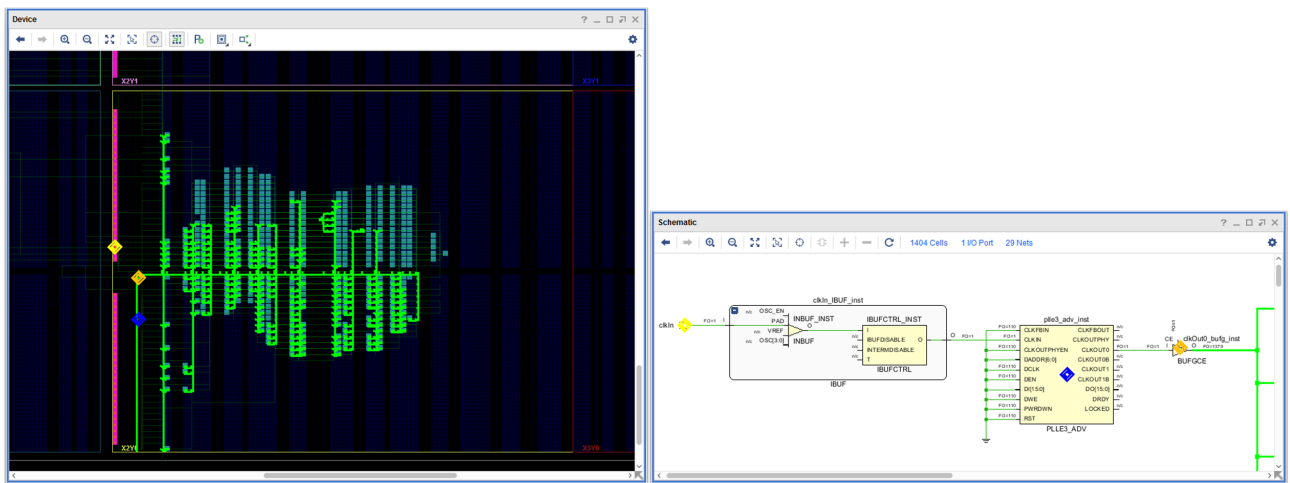
クロック ネットの制約例

CLOCK_LOW_FANOUT プロパティをグローバル クロック バッファで直接駆動されるクロック ネット セグメントに設定し、グローバル クロック バッファのファンアウトが 2,000 ロード未満の場合、ロードの配置は 1 つのクロック領域に含まれます。

次に、グローバル クロック バッファで直接駆動されるクロック ネットに CLOCK_LOW_FANOUT 制約を設定する例を示します。クロック ネットワークで駆動されるロードは 2000 個未満で、1 つのクロック領域に含まれます。入力クロック ポート clkIn は、PACKAGE_PIN が CLOCK_REGION X2Y0 内の GCIO に設定されており、PLLE3_ADV を駆動します。PLLE3_ADV はグローバル クロック バッファを駆動し、そのグローバル クロック バッファはロード数が 1379 個のクロック ネットワークを駆動します。グローバル クロック バッファのロードは、すべて CLOCK_REGION X2Y0 内に配置されます。

```
# PACKAGE_PIN AF9 - IOBank 64 - CLOCK_REGION X2Y0
set_property PACKAGE_PIN AF9 [get_ports clkIn]
set_property IOSTANDARD LVCMOS18 [get_ports clkIn]
set_property CLOCK_LOW_FANOUT TRUE [get_nets -of [get_pins
clkOut0_bufg_inst/O]]
```

図 60: CLOCK_LOW_FANOUT 例の [Device] ウィンドウおよび [Schematic] ウィンドウ



クロッキング トポロジの推奨事項

ザイリンクスでは、デザインに最低限必要な数のクロック バッファを含む単純なクロック ツリー トポロジを使用することをお勧めします。余分なクロック バッファを使用すると、必要となる配線トラックが増えるので、クロック領域で配置エラーまたは配線競合が発生する可能性があり、クロック配線要件も厳しくなり、最大容量に近くなってしまう。

次に、BUFGCE/BUFGCTRL/BUFGCE_DIV 接続に関するクロッキング トポロジの推奨事項を示します。

並列クロック バッファ

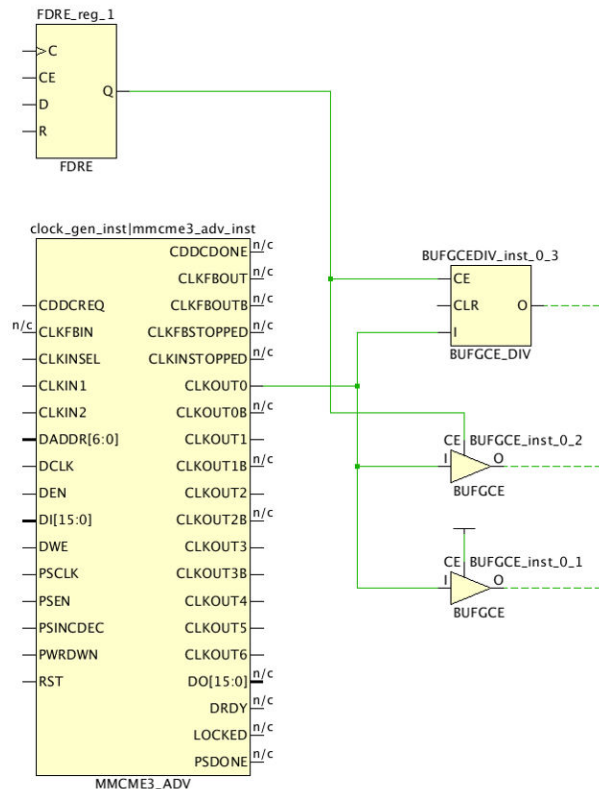
並列クロック バッファを使用すると、次を達成できます。

- インプリメンテーション run 間で予測可能な配置
並列クロック バッファを同じ入力クロック ポート、MMCM、PLL、または GT*_CHANNEL で直接駆動すると、ネットリストの変更またはロジック配置のバリエーションに関係なく、バッファは常にそのドライバーと同じクロック領域に配置されます。
- クロック ツリーの並列分岐間の挿入遅延を一致させる

ザイリンクスでは、分岐間に同期パスがある場合は、カスケード クロック バッファよりも並列バッファをお勧めします。カスケードされたバッファを使用すると、CLOCK_DELAY_GROUP または USER_CLOCK_ROOT 制約を使用する場合でも、クロック ツリーの分岐間でクロック挿入遅延が一致しません。この結果、クロック スキューが大きくなり、タイミング クロージャが困難になります。

次の図に、MMCM CLKOUT0 ポートで駆動される 3 つの並列 BUFGCE バッファーを示します。

図 61: MMCM 出力の並列 BUFGCE



カスケード クロック バッファ

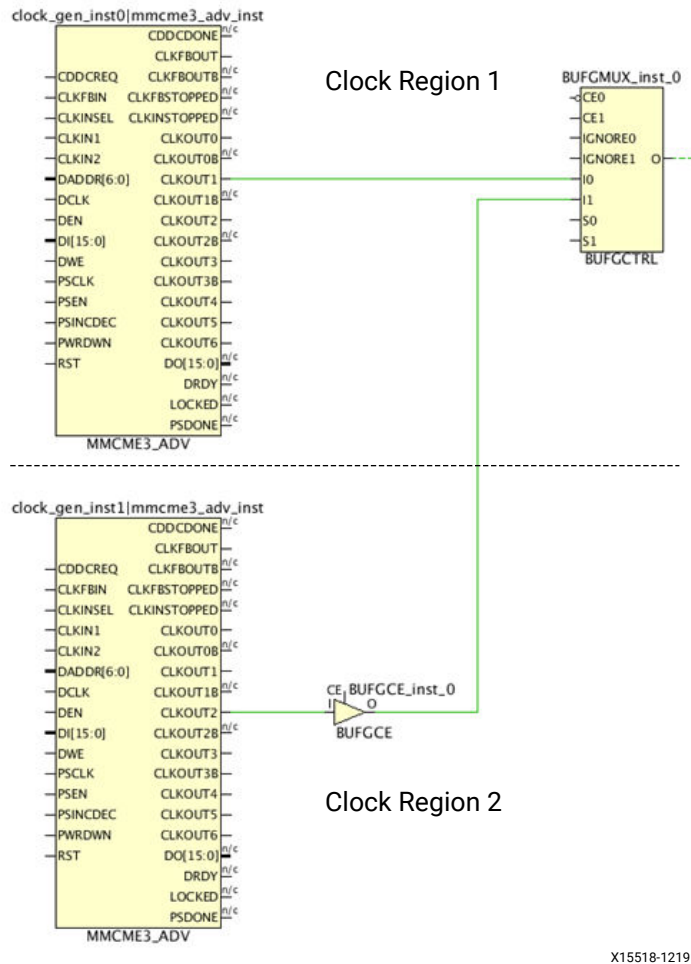
サイリクスでは、カスケード バッファを使用して遅延を人工的に増加し、関連しないクロック ツリー分岐間のスキューを削減することは通常お勧めしません。BUFGCTRL 間の接続とは異なり、ほかのクロック バッファの接続にはアーキテクチャ内に専用パスがありません。このため、クロック バッファの相対配置は予測できず、すべての配置規則が制約されていないカスケード バッファの配置よりも優先されます。

ただし、次を達成するためにはカスケード クロック バッファを使用できます。

- クロックを異なるクロック領域にある別のクロック バッファに配線する

この方法は、異なるクロック領域にある MMCM で生成されたクロックに対してクロック マルチプレクサーを使用する場合に通常使用します。MMCM の 1 つは BUFGCTRL (BUFGMUX) を直接駆動できますが、もう一方の MMCM はクロック信号をそれ以外の領域に配線するために中間クロックを必要とします。次の図に例を示します。

図 62: クロックを別のクロック領域に配線



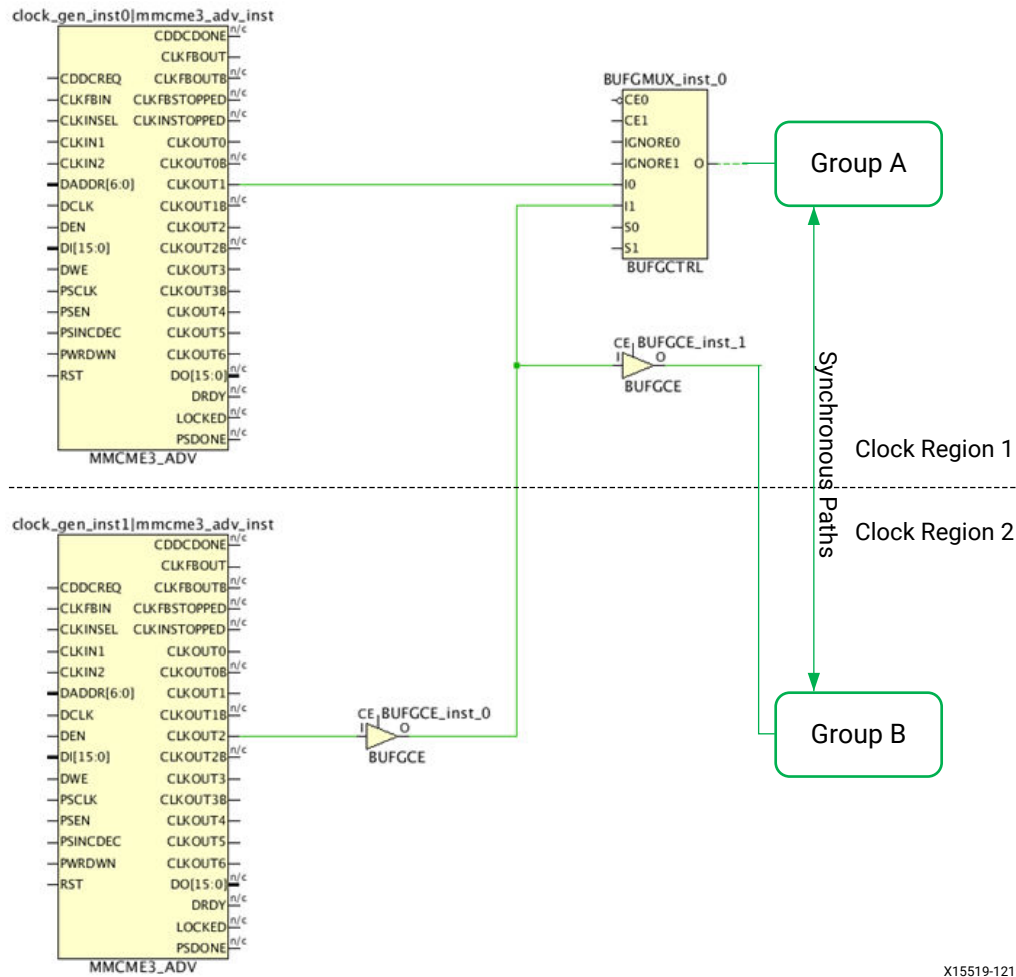
X15518-121919

- 分岐間に同期パスがある場合は、クロック ツリー分岐間でクロック バッファのレベル数のバランスを調整します。

たとえば、グループ A (異なるクロック領域にある BUFGCTRL を介して駆動されるシーケンシャル セル) とグループ B (シーケンシャル セル) の両方を駆動する clk0 という MMCM クロックがあるとします。分岐間の遅延をより一致させるには、グループ B に BUFGCE を挿入し、BUFGCTRL と同じクロック領域に配置します。これにより、グループ A とグループ B 間の同期パスのスキューの量が制御されます。次の図に例を示します。

注記: Vivado ロジック最適化コマンド `opt_design` では、タイミング クロックとクロック ネットワーク分岐のタイミング関係は認識されません。そのため、`opt_design` でできるだけ多くのカスケード接続されたクロック バッファまたは余分なクロック バッファが削除されます。この例では、`opt_design` プロパティが設定されていない場合は、`DONT_TOUCH="TRUE"` により `BUFGCE_inst_1` が削除されます。クロック ツリー分岐間に非同期パスしかない場合、受信クロック ドメインに適切な同期回路があれば、これらの分岐間でバランスを調整する必要はありません。

図 63: クロック領域間の同期パスのクロック ツリーのバランス調整



X15519-121919

- クロック マルチプレクサーに示すようにクロック マルチプレクサーを構築する。

カスケード クロック バッファを使用する場合は、サイリンクスでは挿入遅延とスキューの変動を削減するため次を実行することをお勧めします。

- カスケード バッファを同じクロック領域または隣接するクロック領域に配置する。
- クロック ツリー分岐のバランスが調整されている場合は、同じレベルのクロック バッファすべてを同じクロック領域に割り当てる。

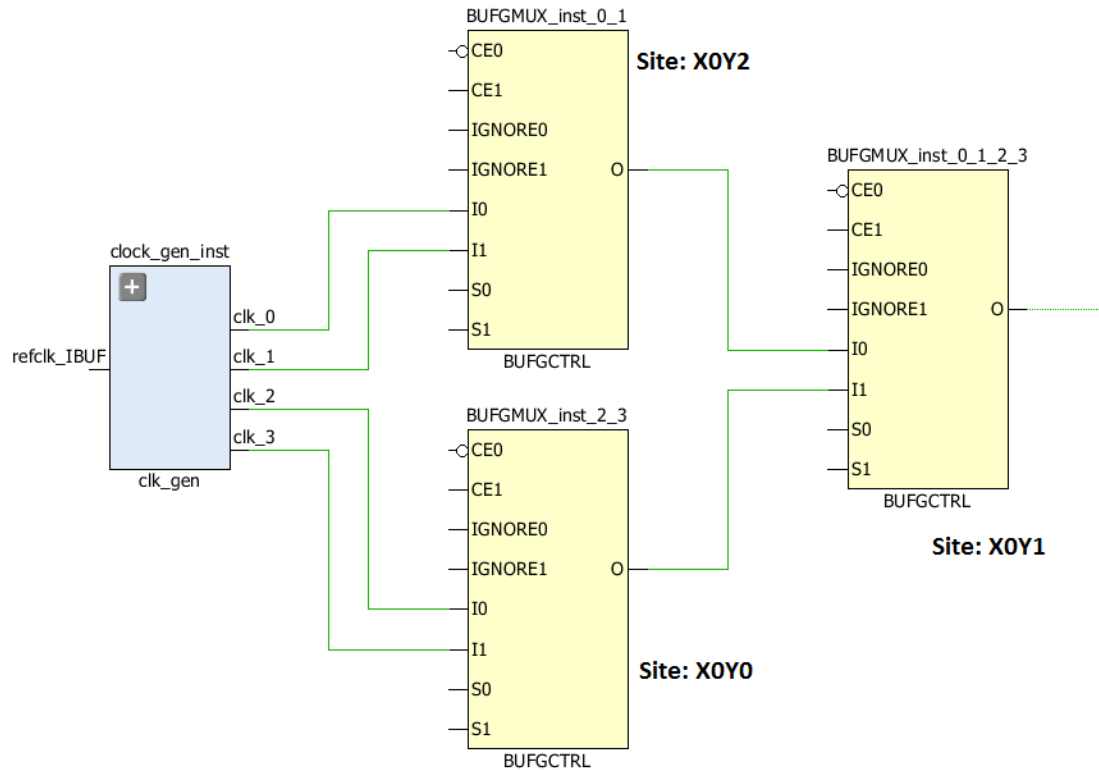
注記: 絶対に必要な場合は、サイリンクスではカスケード BUFGCE の代わりに 2 つのカスケード BUFGCTRL を使用することをお勧めします。専用配線を使用すると、両方の BUFGCTRL が同じクロック領域内に配置される場合に、最小限の遅延で 2 つの隣接する BUFGCTRL をカスケードできます。

クロック マルチプレクサー

並列 BUFGCTRL とカスケード BUFGCTRL を組み合わせて使用すると、クロック マルチプレクサーを構築できます。配置ツールでは、使用可能なクロック バッファ サイトに基づいて、最適な配置が検出されます。可能であれば、BUFGCTRL が隣接するサイトに配置され、専用カスケード パスが活用されます。可能でない場合は、隣接するクロック領域の同じレベルからの BUFGCTRL を配置することが試みられます。

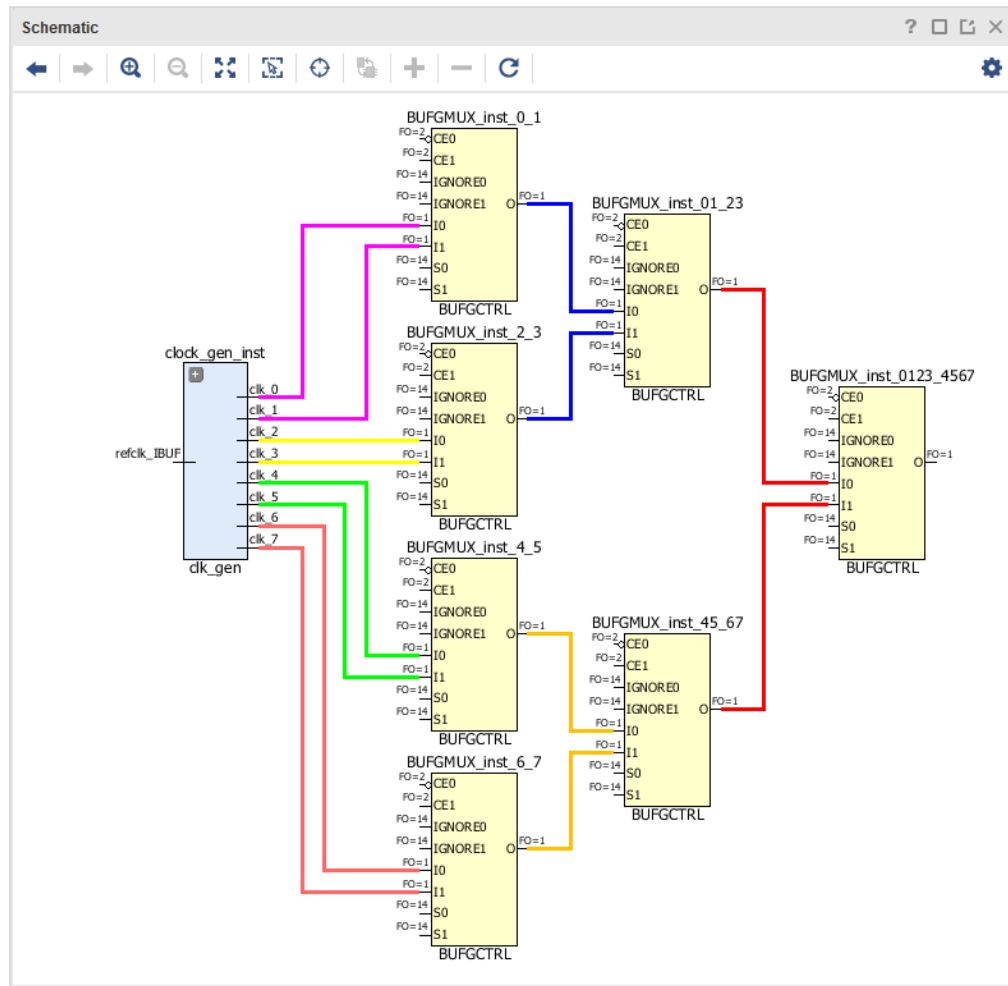
次の図に、バランス調整されてカスケードされた 4:1 MUX を示します。BUFGCTRL バッファの最初のレベルは、両方とも最後の BUFGCTRL (X0Y1) に直接隣接するサイト (X0Y2、X0Y0) に配置されます。この構成により、最後の BUFGCTRL に到達するすべてのクロックに同じ挿入遅延が使用されます。同等の構造は 3:1 MUX にも使用できます。

図 64: 並列 BUFGCTRL を使用した 4:1 MUX



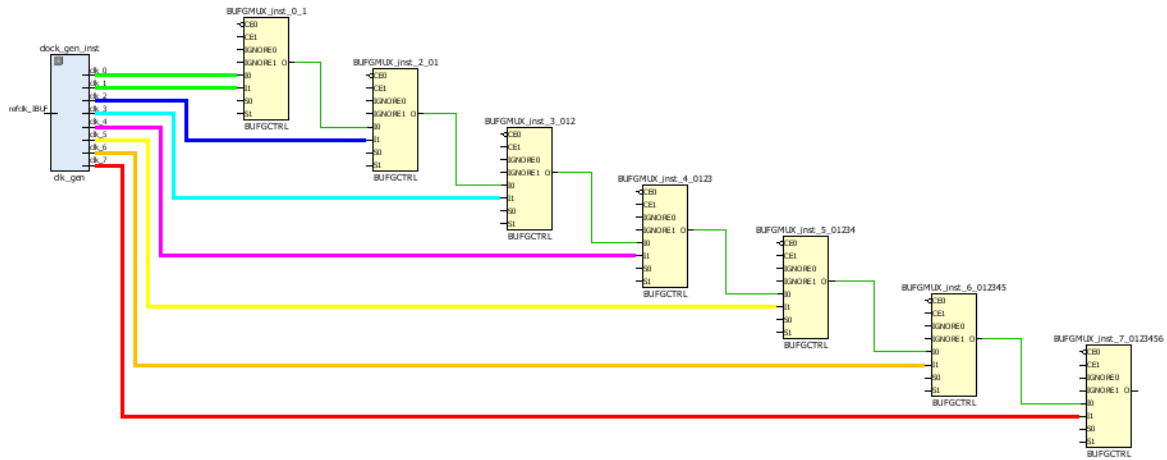
5:1 またはそれ以上のクロック MUX 構造を作成する場合は、次の図に示すような対称的なクロック構造を作成する方法がよく使用されますが、これは最適なソリューションではありません。各 BUFGCTRL には 2 つの隣接する BUFGCTRL へのカスケードパスが 1 つしかなく、BUFGCTRL 間のすべての接続に最低限の遅延が提供されないからです。

図 65: 推奨されないバランス調整済み 8:1 クロック MUX 構造



大型のクロック マルチプレクサー (5:1 ~ 8:1 MUX) をサポートするため、ザイリンクスでは次の図に示すようなカスケード BUFGCTRL を使用することをお勧めします。この図は、7 つの BUFGCTRL を使用する最適な 8:1 MUX を示しています。

図 66: カスケード BUFGCTRL を使用した 8:1 MUX



注記: 幅の広い BUFGCTRL ベースのクロック マルチプレクサーを使用する場合、一部のパスがハードウェアのほかのパスよりも長いので、クロック挿入遅延のバランスを調整できません。このため、この方法は非同期クロックを多重化する場合にのみ推奨されます。

PLL/MMCM フィードバック パスと補正モード

PLL では遅延補正はサポートされず、常に INTERNAL 補正モードで動作するので、フィードバック パスは必要ありません。同様に、MMCM を INTERNAL 補正モードに設定する場合もフィードバック パスは必要ありません。どちらの場合も、Vivado ツールで不要なフィードバック クロック バッファが常に自動的に削除されるわけではありません。ファンアウトの大きいクロック リソースの使用量を減らすには、クロック バッファを手動で削除する必要があります。これは、特にクロック競合が発生する可能性のあるクロック リソースの使用率が高いデザインで重要です。

MMCM 補正を ZHOLD または BUF_IN に設定すると、フィードバック バッファで駆動されるネットおよび CLKOUT0 ピンに直接接続されているすべてのバッファで駆動されるネットに同じクロック ルートが割り当てられます。これにより挿入遅延が一致するようになり、CLKOUT0 に接続されている I/O ポートとシーケンシャルセルの位相が揃い、ホールド タイムがデバイス インターフェイスで満たされるようになります。Vivado ツールでは、クロック ルートを最適に定義するため、これらのツールのロードがすべて考慮されます。

Vivado ツールでは、挿入遅延とその他の MMCM 出力を一致させる処理は自動的に実行されません。その他の MMCM 出力バッファで駆動されるネットの挿入遅延を一致させるには、次のプロパティを使用します。

- **CLOCK_DELAY_GROUP**

同じ CLOCK_DELAY_GROUP プロパティ 値をフィードバック クロック バッファ、CLKOUT0 バッファ、およびその他の MMCM 出力バッファで直接駆動されるネットに適用します。これが推奨される方法です。

- **USER_CLOCK_ROOT**

特定のクロック ルートを指定する必要がある場合は、フィードバック クロック バッファ、CLKOUT0 バッファ、その他の MMCM 出力バッファで直接駆動されるネットに同じ USER_CLOCK_ROOT プロパティ 値を使用します。

BUFG_GT 分周器

BUFG_GT バッファはファブリックのどのロードでも駆動でき、GT*_CHANNEL からクロックを分周するために使用できるオプションの分周器を含みます。これにより、クロックを分周するために余分な MMCM または BUFG_DIV を使用する必要はなくなります。

SelectIO クロッキング

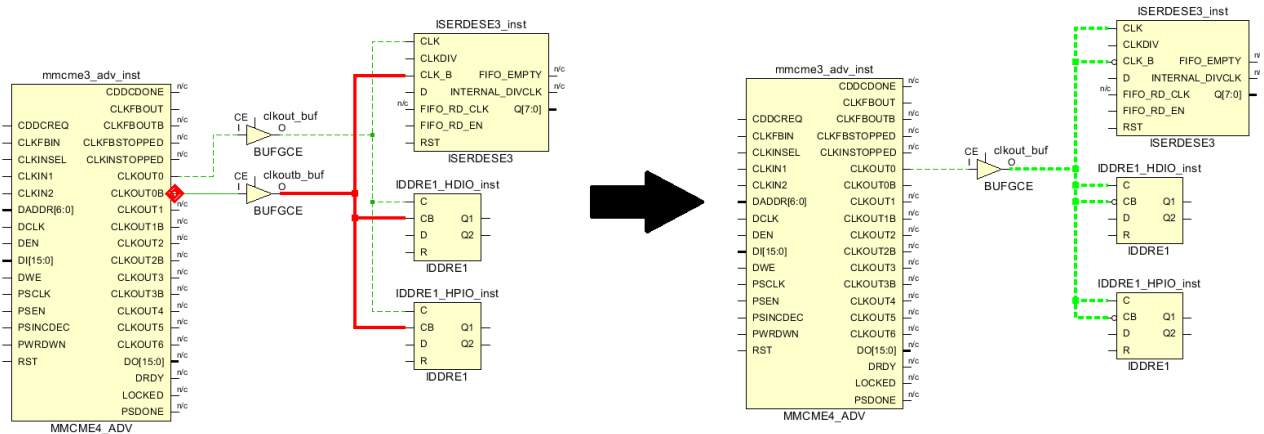
UltraScale デバイスの SelectIO プリミティブには、クロック ピン間に最大スキュー要件があります。SelectIO プリミティブに最適なクロッキング トポロジを使用すると、最大スキュー違反を回避でき、UltraScale デバイスとファブリック ロジックの間のタイミングが向上し、使用されるクロック リソースの量が少なくなります。

ISERDESE3 および IDRE1 のクロッキング

UltraScale および UltraScale+ デバイスの ISERDESE3 と IDRE1 のクロッキングには、クロック ピンと反転クロック ピンの間に最大スキュー要件があります。サイリンクスでは、ローカル反転を使用する場合は、最大スキュー要件を満たすため、クロック ピンと反転クロック ピンに 1 つのネットを使用することをお勧めします。

次の図では、左側に MMCM の CLKOUT0B 出力を使用した最適ではない構成を示し、右側に ISERDESE3 と IDRE1 の CLK_B ピンと CB ピンにローカル反転を使用した最適な構成を示します。最適な構成を使用することにより、最大スキュー要件が満たされ、使用されるグローバル クロック リソースの量が少なくなります。

図 67: ISERDESE3 および IDRE1 の最適ではないクロッキング トポロジと最適なクロッキング トポロジ



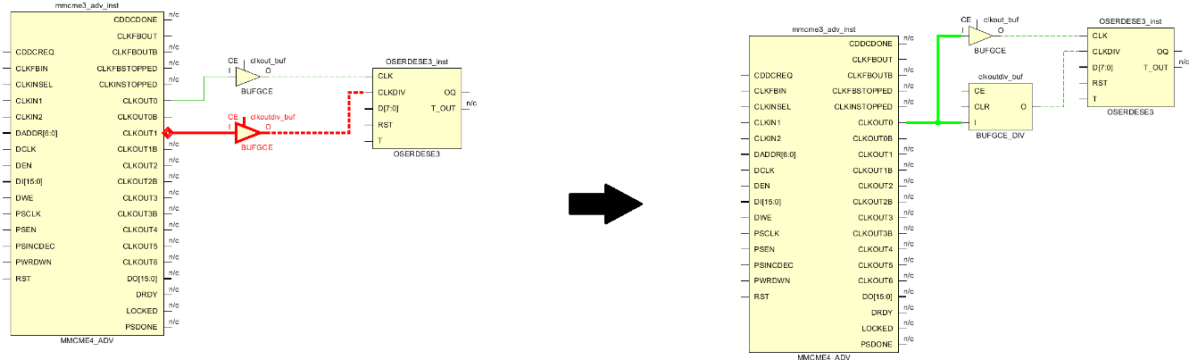
OSERDESE3 クロッキング

UltraScale および UltraScale+ デバイスの OSERDESE3 のクロッキングには、高速クロック ピンと分周クロック ピンの間に最大スキュー要件があります。サイリンクスでは、最大スキュー要件を満たすため、グローバル クロック バッファを並列に使用し、そのうち 1 つのグローバル クロック バッファを BUFCE_DIV にすることをお勧めします。これにより、MMCM の 2 つの出力の間に発生する追加のクロックのばらつきが除去されます。

次の図では、左側に MMCM の 2 つの出力を使用した最適ではない構成を示し、右側に 1 つの MMCM 出力と BUFCE_DIVIDE プロパティを使用して分周クロックを供給する BUFCE_DIV セルを使用した最適な構成を示します。

注記: 高速クロックは BUFCE で駆動する必要はありません。または、BUFCE_DIVIDE プロパティを 1 に設定し、BUFCE_DIV と共に使用することもできます。

図 68: OSERDESE3 の最適ではないクロッキング トポロジと最適なクロッキング トポロジ



MMCM ZHOLD/BUF_IN 補正を使用した I/O タイミング

ZHOLD 補正モードでは、I/O 列の I/O レジスタすべてのホールド タイムが負になるように MMCM が設定されます。クロック兼用 I/O (CCIO) が ZHOLD 補正モードに設定された MMCM 1 つを駆動する場合、MMCM を CCIO と同じクロック領域に配置することが試みられます。この場合、CCIO で BUFG を介さずに直接 MMCM を駆動できるので、MMCM の ZHOLD 補正は有効なままになります。

ただし、CCIO が ZHOLD モードの MMCM とさらにもう 1 つ別の MMCM を駆動する場合は、ロジック最適化で MMCM へのクロック配線を有効なものにするため CCIO の後に BUFG が挿入されます。ZHOLD 補正モードの MMCM が CCIO により直接駆動されなくなるので、補正モードは BUF_IN に変更されます。これを回避するには、CCIO で ZHOLD モードの MMCM を直接駆動し、もう 1 つの MMCM は BUFG を介して駆動するようにします。また、BUFG で駆動されるネットの CLOCK_DEDICATED_ROUTE を ANY_CMT_COLUMN に設定してください。

クロック挿入遅延は、クロック ルートの位置によって異なり、クロック ルート配置はロードの配置によって異なるので、run 間でばらつきがあります。このばらつきにより、デバイス内のタイミングと I/O タイミングも変わります。

高周波数 I/O を使用する場合は、I/O タイミングを詳細に制御し、run 間のばらつきを減らすことが必要な場合があります。これを達成する方法の 1 つに、クロック ルート配置を指定する方法があります。ツールを自動モードで実行して、クロック ルート領域を確認します。I/O タイミングに問題なければ、I/O タイミングに関連するバッファ ネットにクロック ルートを強制的に配置できます。クロック ルートの配置を確認するには、`report_clock_utilization [-clock-roots-only] Tcl` コマンドを使用します。

次の例では、I/O ポートは X0Y0 領域にあります。Vivado 配置により、I/O 配置およびその他のロードの配置に基づいて、クロック ルートが X1Y2 に配置されています。

図 69: クロック ルートに制約が設定されていない場合のクロック使用量のサマリ

Index	Clock Net	Root Clock Region	Clock Root Node
1	mmcm3_inst/clk0	X1Y2	RCLK_BRAM_L_X30Y209/CLK_VDISTR_B0T0
2	mmcm3_inst/clkfbout_buf_mmcm_zhold	X1Y2	RCLK_CLEL_R_L_X25Y209/CLK_VDISTR_B0T

次のサマリは、クロック ルートに制約が設定されていない場合の I/O タイミングを示しています。

図 70: クロック ルートに制約が設定されていない場合のタイミング サマリ

Setup	Hold
Worst Negative Slack (WNS): -0.279 ns	Worst Hold Slack (WHS): 0.036 ns
Total Negative Slack (TNS): -5.394 ns	Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 47	Number of Failing Endpoints: 0

次の例では、クロック ルートが XOY0 の I/O レジスタの隣に移動されているので、クロック挿入遅延とタイミングの不必要に悪い見積もり部分が削減され、I/O タイミングが改善しています。

図 71: クロック ルートにユーザー制約が設定されている場合のクロック使用量のサマリ

Index	Clock Net	Root Clock Region	Clock Root Node
1	mmcn/inst/clk0	XOY0	XIPHY_L_XOY60/CLK_VDISTR_B0T20
2	mmcn/inst/clkfbout_buf_mmcn_zhold	XOY0	XIPHY_L_XOY60/CLK_VDISTR_B0T14

次のサマリは、クロック ルートが移動された場合の I/O タイミングを示しています。

図 72: クロック ルートにユーザー制約が設定されている場合のタイミング サマリ

Setup	Hold
Worst Negative Slack (WNS): -0.217 ns	Worst Hold Slack (WHS): 0.060 ns
Total Negative Slack (TNS): -1.488 ns	Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 16	Number of Failing Endpoints: 0

同期 CDC

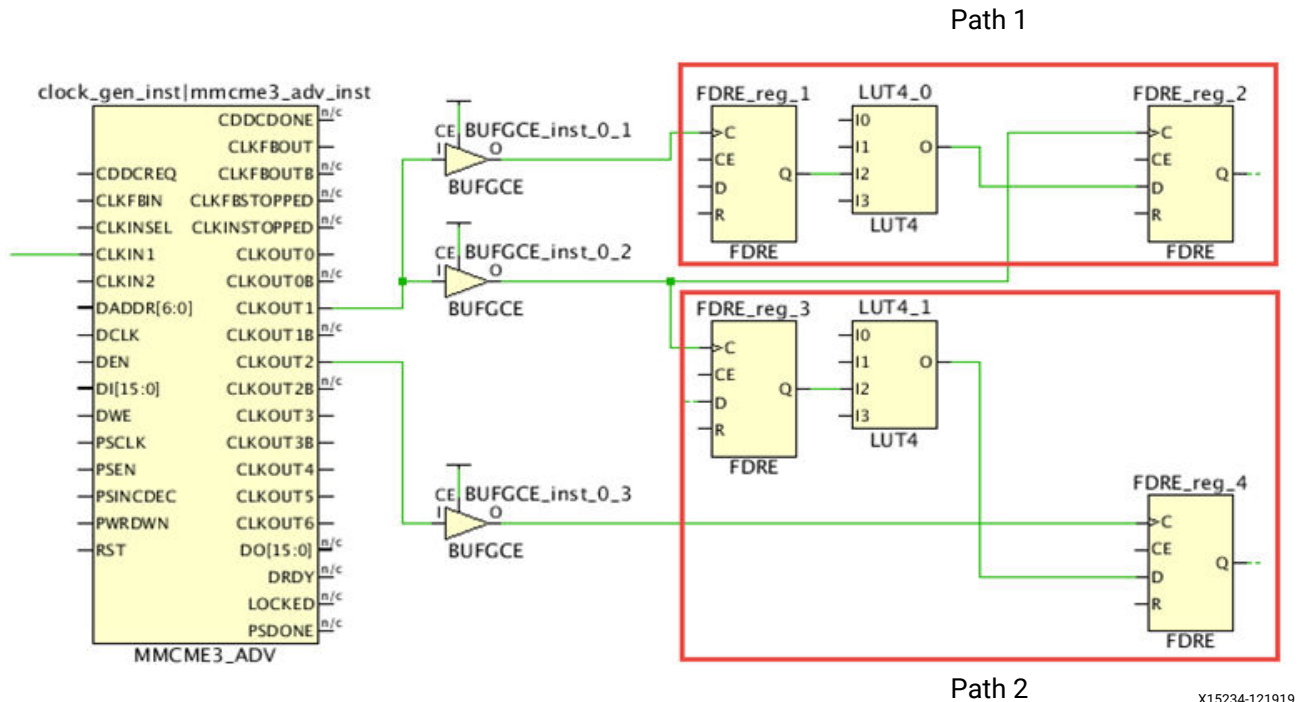
デザインに同じ MMCM/PLL からのクロック間の同期 CDC パスが含まれる場合、次の方法を使用して、クロック挿入遅延とスキューを制御することにより、これらのパスのスラックを制御できます。

★ **重要:** 異なる MMCM/PLL からのクロック間にある CDC パスでは、MMCM/PLL のクロック挿入遅延は制御が困難です。この場合、ザイリンクスではこれらのクロック乗せ換えを非同期として扱い、それに合わせてデザインを変更することをお勧めします。

同じ MMCM/PLL の異なる出力ピンからの 2 つのクロック間のパスに対してタイミング解析が実行されると、MMCM/PLL 位相エラーによりそのパスにクロックのばらつきが追加されます。高クロック周波数を使用するデザインでは、位相エラーによりセットアップとホールドの両方でタイミング クロージャ問題が発生することがあります。

次の図に、位相エラーのある場合とない場合のパスの例を示します。パス 1 は、同じ MMCM 出力に接続された 2 つのバッファからクロックが供給されている CDC パスで、位相エラーはありません。パス 2 には、2 つの異なる MMCM 出力から 2 つのクロックが供給されており、位相エラーがあります。

図 73: MMCM および位相エラー



同じ MMCM/PLL からの 2 つの同期クロックがシンプルな周期比 (1/2 /4 /8) である場合は、2 つの BUFGCE_DIV バッファに接続される 1 つの MMCM/PLL 出力を使用すると、2 つのクロック ドメイン間で位相エラーが発生しないようにできます。BUFGCE_DIV バッファでは、クロック分周 (1/1、1/2、1/4、1/8) が実行されます。その他の周期比 (1/3、1/5、1/6、1/7) も使用できますが、クロックのデューティ サイクルを変更する必要があり、混合エッジのタイミングパスがさらに困難になります。

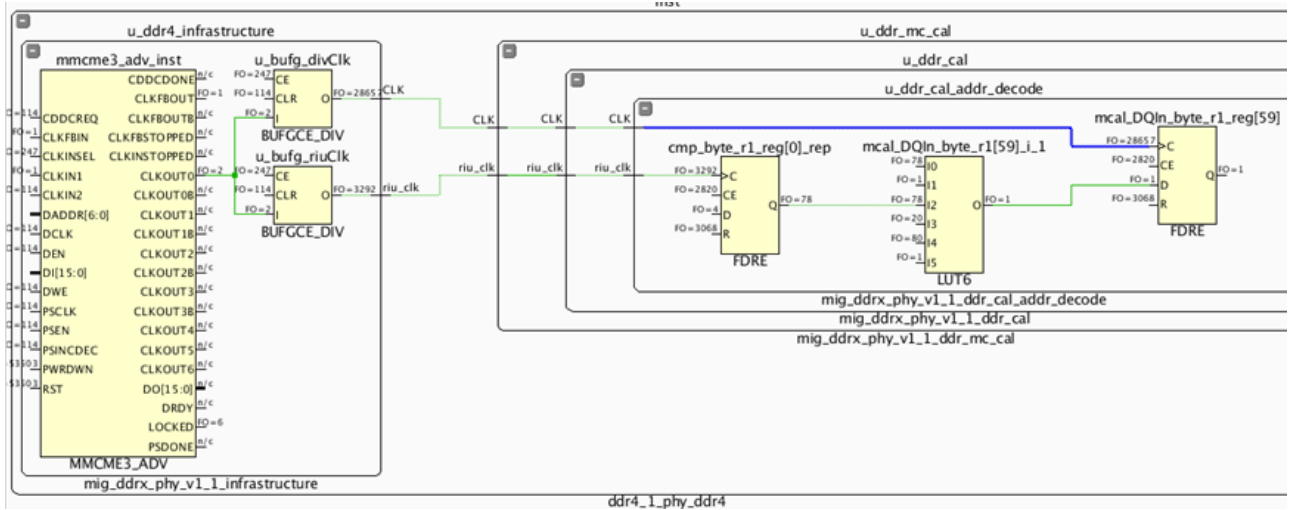
注記: BUFGCE と BUFGCE_DIV のセル遅延は異なるので、サイリンクスでは両方の同期クロックに同じクロック バッファ (2 つの BUFGCE または 2 つの BUFGCE_DIV バッファ) を使用することをお勧めします。

次の図に、CLKOUT0 クロックを 1 と 2 でそれぞれ分周する 2 つの BUFGCE_DIV を示します。



重要: BUFGCE_DIVIDE プロパティが 1 より大きい値に設定されている並列の BUFGCE_DIV セル間のタイミングが安全なものになるように、両方のバッファで同じイネーブル信号 (CE) および同じリセット信号 (RST) を使用する必要があります。そうしないと、ハードウェアで分周クロックがお互いに位相シフトしたものになる可能性があり、これは Vivado ツールによりレポートされません。

図 74: 1 つの MMCM 出力に接続された BUFGCE_DIV を使用する MMCM 同期 CDC



同じ MMCM または PLL からの複数のクロック間のバランスが自動的に調整されるようにするには、バランス調整する必要のあるクロック バッファで駆動されるネットに同じ CLOCK_DELAY_GROUP プロパティ値を設定します。次はその他の推奨事項です。

- CLOCK_DELAY_GROUP を設定するクロックの数が多すぎないようにします。設定するクロックの数が多すぎると、クロック配置ツールに負荷がかかり、ソリューションが最適でなくなるか、エラーが発生します。
- タイミング サマリ レポートでクリティカル同期 CDC パスを確認し、タイミングを満たすためにどのクロックの遅延を一致させる必要があるかを判断します。
- 要件が厳しく、クロック トポロジが同一の同期クロックのグループに対しては、CLOCK_DELAY_GROUP の使用は制限してください。



重要: ザイリンクスでは、最適なクロッキング構造を作成するため、Clocking Wizard を使用することをお勧めします。Clocking Wizard では、BUFGCE と BUFGCE_DIV が組み合わせて使用され、関連のクロック グループ制約が設定されます。

GT インターフェイスのクロッキング

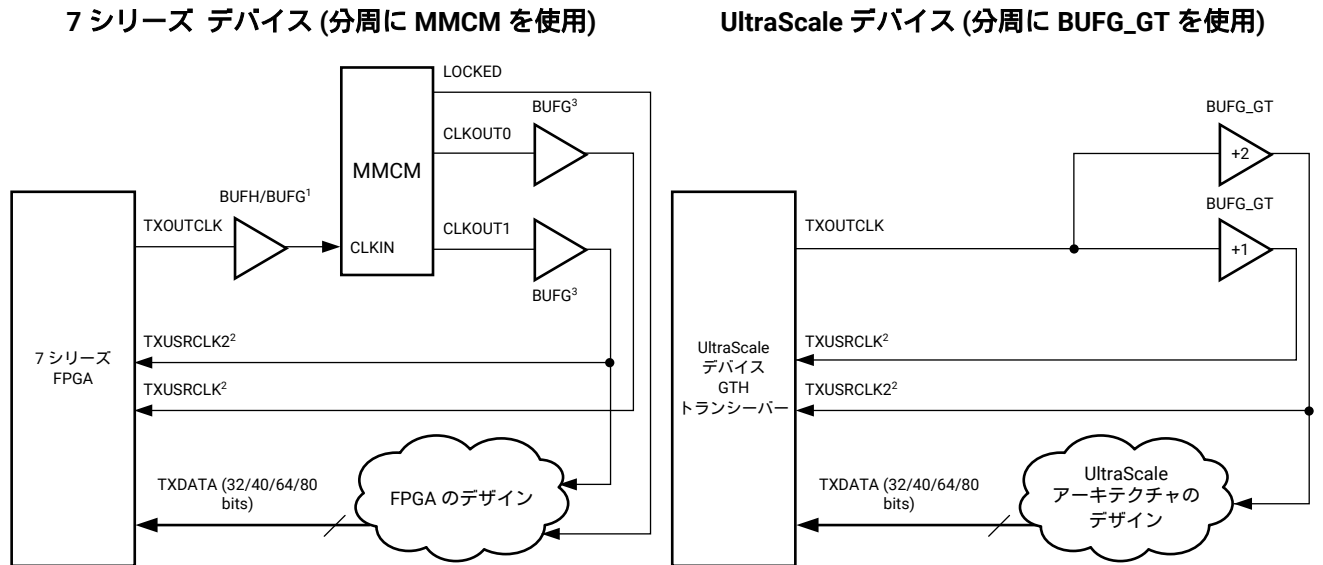
各 GT インターフェイスには複数のクロックが必要です。その一部は、1 つまたは複数の GT クワッド内のボンディングされた GT*_CHANNEL セル間で共有されます。UltraScale デバイスには、最大 128 個の GT*_CHANNEL サイトがあるので、デザインに数百個のクロックが使用されることがあります。ほとんどの GT クロックには、関連する GT*_CHANNEL の隣のクロック領域内に配置されたロードが使用され、ファンアウトは小さくなります。一部の GT クロックは、デバイス全体のロードを駆動し、多くのクロック領域のクロック配線リソースを使用する必要があります。UltraScale アーキテクチャでは、必要な多くの GT クロックを効率的にサポートするため、次のような機能が提供されています。

ダイナミック ドライバー付き BUFG_GT

UltraScale デバイスでは、BUFG_GT バッファにより GT クロッキングが単純化されています。BUFG_GT にはダイナミック分周機能が含まれるので、MMCM で GT 出力クロックの単純な整数分周を実行する必要はありません。これにより、クロック リソースを節約でき、分周された GT*_CHANNEL 出力クロックとフルレートのクロックの両方が必要な場合に、クロック パスのスキューを小さくできます。

ユーザー ロジックが内部 PCS ロジックの半分のクロック周期で動作する GT インターフェイス、または GT*_CHANNEL で user_clk、sys_clk、pipe_clk 用に複数のクロック周波数を生成する必要がある PCIe インターフェイスには、BUFG_GT グローバル クロック バッファを使用できます。次の図では、TXUSRCLK2 の周波数が TXUSRCLK の周波数の半分である単一レーンの GT インターフェイスにおける、7 シリーズと UltraScale デバイスのクロッキング要件を比較しています。

図 75: クロッキング要件の比較



X15237-121919

クワッド内の GT*_CHANNEL の出力クロックまたはクワッド内の IBUFDS_GTE3/ODIV2 ピンで生成される基準クロックのどれを使用しても、同じクロック領域内の 24 個の BUFG_GT バッファのどれでも駆動できます。BUFG_GT_SYNC は、共有クロック ソースで駆動される BUFG_GT のリセットとクリアを同期するために常に必要です。

注記: BUFG_GT_SYNC プリミティブがデザインに含まれない場合、Vivado ツールで自動的に追加されます。

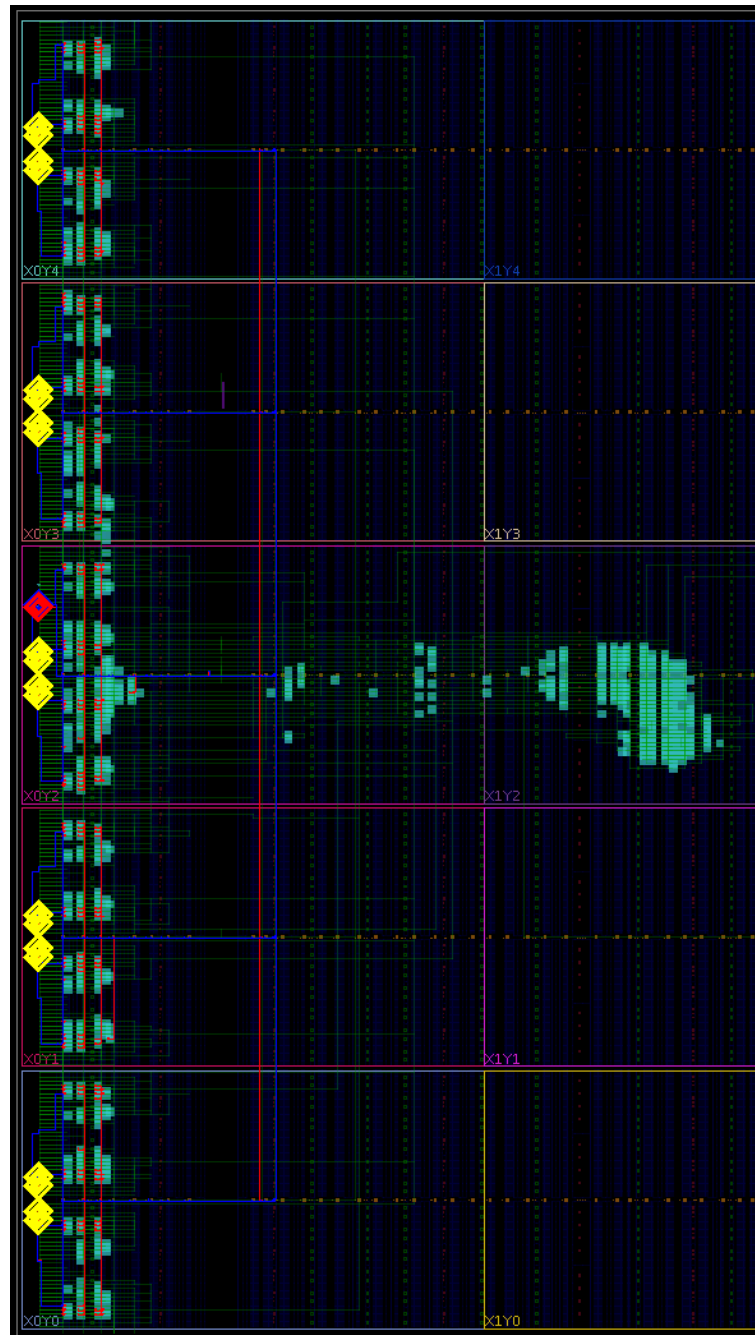
アプリケーションによっては、GT 出力クロックまたは IBUFDS_GTE3/ODIV2 基準クロックの複雑な非整数クロック分周を生成するために MMCM を使用する必要がある場合があります。この場合、BUFG_GT が MMCM を直接駆動する必要があります。デフォルトでは、MMCM は BUFG_GT と同じクロック領域の行に配置されます。ほかの MMCM が同じ MMCM サイトを使用しようとする、自動配置された MMCM が BUFG_GT のできるだけ近くに配置されているかを確認し、長い配線によりクロッキング リソースが無駄に使用されないようにする必要があります。

シングル クワッド vs. マルチ クワッド インターフェイス

マルチチャネル インターフェイスの場合、マスター チャネルでそのインターフェイスの GT*_CHANNE すべてに対して [RT]XUSRCLK[2] を生成できます。マルチチャネル インターフェイスが複数のクワッドにまたがる場合、基準クロック ソースからの GT*_CHANNEL の最大許容距離は上下 2 クロック領域です。

次の図に、マルチ クワッド インターフェイスを示します。GT*_CHANNEL は黄色でマークされ、TXUSRCLK は青、TXUSRCLK2 は赤でハイライトされています。TXUSRCLK と TXUSRCLK2 の両方を駆動している BUFG_GT は中央のクワッドにあり、青と赤でマークされています。

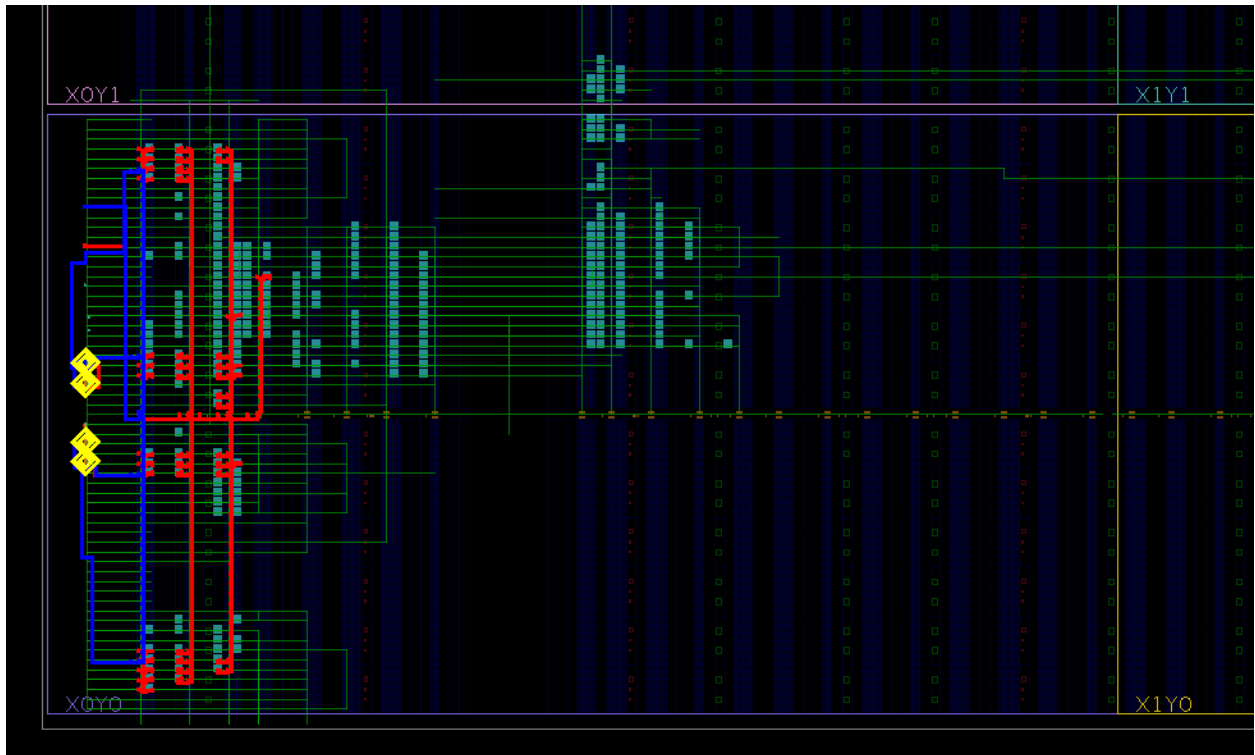
図 76: マルチ クワッド インターフェイスの TXUSRCLK/TXUSRCLK2 クロック配線



GT インターフェイスが 1 つのクワッド内に含まれる場合は、配置で BUFG_GT クロックがローカル クロックとして扱われます。この場合、BUFG_GT の水平方向に隣接したクロック領域に BUFG_GT クロック ロードが配置されるよう試みられます (BUFG_GT を含むクロック領域から開始し、デバイスの半分の幅まで使用される可能性あり)。

配置のリージョナル クロック制約を無効にするには、BUFG_GT クロック ロードのいずれかを Pblock に割り当てます。次の図に、シングル クワッド インターフェイスを示します。GT*CHANNEL は黄色でマークされ、TXUSRCLK は青、TXUSRCLK2 は赤でハイライトされています。TXUSRCLK2 のロードはすべて、GT*CHANNEL と同じクロック領域に配置されます。

図 77: シングル クワッド インターフェイスの TXUSRCLK/TXUSRCLK2 クロック配線



[RT]XUSRCLK/[RT]XUSRCLK2 スキューの一致

[RT]XUSRCLK2 が [RT]XUSRCLK の半分の周波数で動作する場合 (1 で分周した BUFG_GT と 2 で分周した BUFG_GT)、GT インターフェイスの各 GT*CHANNEL の [RT]XUSRCLK/[RT]XUSRCLK2 ペア間のスキュー要件は厳しくなります。このスキュー要件を満たすには、[RT]XUSRCLK/[RT]XUSRCLK2 ペアを生成するマスター チャネルの最大 2 クロック領域上または下までに GT*CHANNEL を配置します。また、配置ツールでスキューが次のように厳しく制御されます。

- BUFG_GT ペアをクワッドの上部または下部 12 個の BUFG_GT に割り当てます。
- 両方のクロックのクロック ルートを BUFG_GT を含むクロック領域の近くに割り当てます。



推奨: ザイリンクスでは、スキュー違反を避けるため、[RT]XUSRCLK2 が [RT]XUSRCLK の半分の周波数で動作する場合はこのクロッキング トポロジに従うことをお勧めします。

Integrated Block for PCI Express の CORECLK/PIPECLK/USERCLK スキューの一致

UltraScale Integrated Block for PCI Express® には、CORECLK、USERCLK、および PIPECLK の 3 つのクロックが必要です。この 3 つのクロックは、物理インターフェイスの 1 つの GT*CHANNEL の TXOUTCLK ピンで駆動される BUFG_GT から供給されます。CORECLK ピンと PIPECLK ピン間、および CORECLK ピンと USERCLK ピン間には、厳しいスキュー要件があります。このスキュー要件を満たすため、配置ツールで次のようにスキューが厳しく制御されます。

- グループ内の 3 つの PCIe クロックを駆動する BUFG_GT を、クワッドの上部または下部 12 個の BUFG_GT に割り当てます。
- 3 つすべてのクロックのクロック ルートを同じクロック領域に割り当てます。

注記: PCIe のクロッキング要件の詳細は、『UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP 製品ガイド』 (PG156: [英語版](#)、[日本語版](#)) を参照してください。

7 シリーズ デバイスのクロッキング

注記: このセクションでは、Virtex®-7 のクロック リソースを例として使用します。Virtex-6 デバイスのクロック リソースも同様です。別のアーキテクチャを使用する場合は、『7 シリーズ FPGA クロッキング リソース ユーザー ガイド』 (UG472: [英語版](#)、[日本語版](#)) または『UltraScale アーキテクチャ クロッキング リソース ユーザー ガイド』 (UG572: [英語版](#)、[日本語版](#)) を参照してください。

Virtex-6 および Virtex-7 デバイスには、BUFG という 32 個のグローバル クロック バッファが含まれます。BUFG は、クロック数、デザイン パフォーマンス、およびクロック制御の要件がそれほど厳しくないデザインの、ほとんどのクロッキングのニーズを満たすことができます。グローバル クロック リソースには BUFG、BUFGCE、BUFGMUX、BUFGCTRL プリミティブなどがあり、それぞれ独自の機能があります。これらのグローバル クロック コンポーネントの機能の詳細は、該当するデバイスのクロッキング リソース ユーザー ガイド (『7 シリーズ FPGA クロッキング リソース ユーザー ガイド』 (UG472: [英語版](#)、[日本語版](#)) または『UltraScale アーキテクチャ クロッキング リソース ユーザー ガイド』 (UG572: [英語版](#)、[日本語版](#))) およびライブラリ ガイド (『Vivado Design Suite 7 シリーズ FPGA および Zynq-7000 SoC ライブラリ ガイド』 ([UG953](#)) または『UltraScale アーキテクチャ ライブラリ ガイド』 ([UG974](#))) を参照してください。



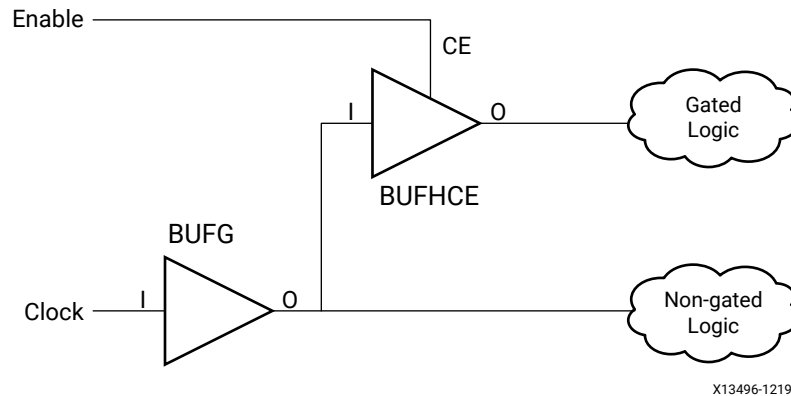
推奨: 使用可能な BUFG の数より多くの BUFG が必要な場合、または全体的なクロック特性を改善する必要がある場合は、使用可能なクロック リソースに対してクロックの要件を解析し、タスクに最適なリソースを選択してください。

グローバル クロック リソースに加えてリージョナル クロック リソースもあり、クロック ネットワークのより厳密な制御に使用できます。リージョナル クロック リソースには、水平クロック領域バッファ (BUFH、BUFHCE)、リージョナル クロック バッファ (BUFR)、I/O クロック バッファ (BUFIO)、およびマルチ リージョナル クロック バッファ (BUFMR) があります。これらのリージョナル クロック コンポーネントの機能の詳細は、該当するデバイスのクロッキング リソース ユーザー ガイド (『7 シリーズ FPGA クロッキング リソース ユーザー ガイド』 (UG472: [英語版](#)、[日本語版](#)) または『UltraScale アーキテクチャ クロッキング リソース ユーザー ガイド』 (UG572: [英語版](#)、[日本語版](#))) およびライブラリ ガイド (『Vivado Design Suite 7 シリーズ FPGA および Zynq-7000 SoC ライブラリ ガイド』 ([UG953](#)) または『UltraScale アーキテクチャ ライブラリ ガイド』 ([UG974](#))) を参照してください。

水平クロック領域バッファをクロック ゲーティングに使用

水平クロック領域バッファ (BUFHCE) を BUFG と使用して、中粒度のクロック ゲーティングを実行できます。クロック ドメインの数百から数千個のロードを含む部分でクロックを断続的に停止する必要がある場合は、BUFHCE を効率的なクロック リソースとして使用できることがあります。BUFG は同じまたは異なるクロック領域内の複数の BUFHCE を駆動できるので、クロック スキューの小さい複数のドメインのクロッキングを個別に制御できます。

図 78: 水平クロック領域バッファ



BUFH を単独で使用する場合は、BUFH に接続されているロードはすべて同じクロック領域に含まれる必要があります。これは、高速で細粒度の (ロードが少ない) クロックが必要な場合に適しています。BUFHCE は、特定クロック領域内で中粒度のクロック ゲーティングを達成するために使用できます。BUFH で駆動されるリソースがクロック領域で使用可能なリソースを超えないようにし、また、その他の競合が発生しないようにしてください。

BUFH と、BUFG、ほかの BUFH、またはほかのクロック リソースで駆動されるクロック ドメイン間で、位相関係が異なる可能性があります。ただし、2 つの BUFH が水平方向に隣接した領域を駆動する場合は例外です。この場合、両方の BUFH が同じクロック ソースで駆動されると、左のクロック領域と右のクロック領域の間のスキューは厳密に制御された位相関係になり、データを安全に 2 つの BUFH クロック ドメイン間で転送できます。BUFH は、クロック入力または GT の反対側の領域の MMCM または PLL にアクセスするために使用できますが、その場合は MMCM または PLL が使用可能であることを確認してください。

SSI デバイスのクロックに関する追加の考慮事項

一般に、前述のクロックに関する考慮事項はすべて SSI テクノロジ デバイスにも当てはまりますが、これらのデバイスをターゲットにする場合は、その構造のため追加の考慮事項があります。BUFMR を使用する場合は、SLR の境界をまたぐクロック リソースは駆動できません。そのためザイリンクスでは、BUFMR を駆動するクロックを、SLR 内の中央クロック領域にあるバンクまたはクロック領域に配置することをお勧めします。これにより、SLR の左側または右側にある 3 つすべてのクロック領域にアクセスできるようになります。

グローバル クロッキングに関しては、デザインに必要なグローバル クロック (BUFG) が 16 個以下の場合は、追加の考慮事項はありません。ツールにより競合が発生しないように BUFG が自動的に割り当てられます。必要な BUFG が 17 個以上 32 個未満の場合は、ピンの選択および配置に注意を払い、グローバル クロック ラインの競合やクロック ロードの配置によってリソースの競合が発生しないようにする必要があります。

その他すべての ザイリンクス 7 シリーズ デバイスと同様、CCIO (クロック兼用 I/O) およびそれに関連する CMT (クロック マネージメント タイプ) は、SLR 内で駆動できる BUFG に制限があります。SLR の上半分または下半分にある CCIO は、その SLR の上半分または下半分にある BUFG のみを駆動できます。このため、ピンおよび関連する CMT は、すべての SLR の上半分または下半分で必要な BUFG が合計 16 個以下になるように選択する必要があります。このようにすると、すべてのクロックが競合なしですべての SLR を駆動できるように、ツールですべての BUFG が自動的に割り当てられます。

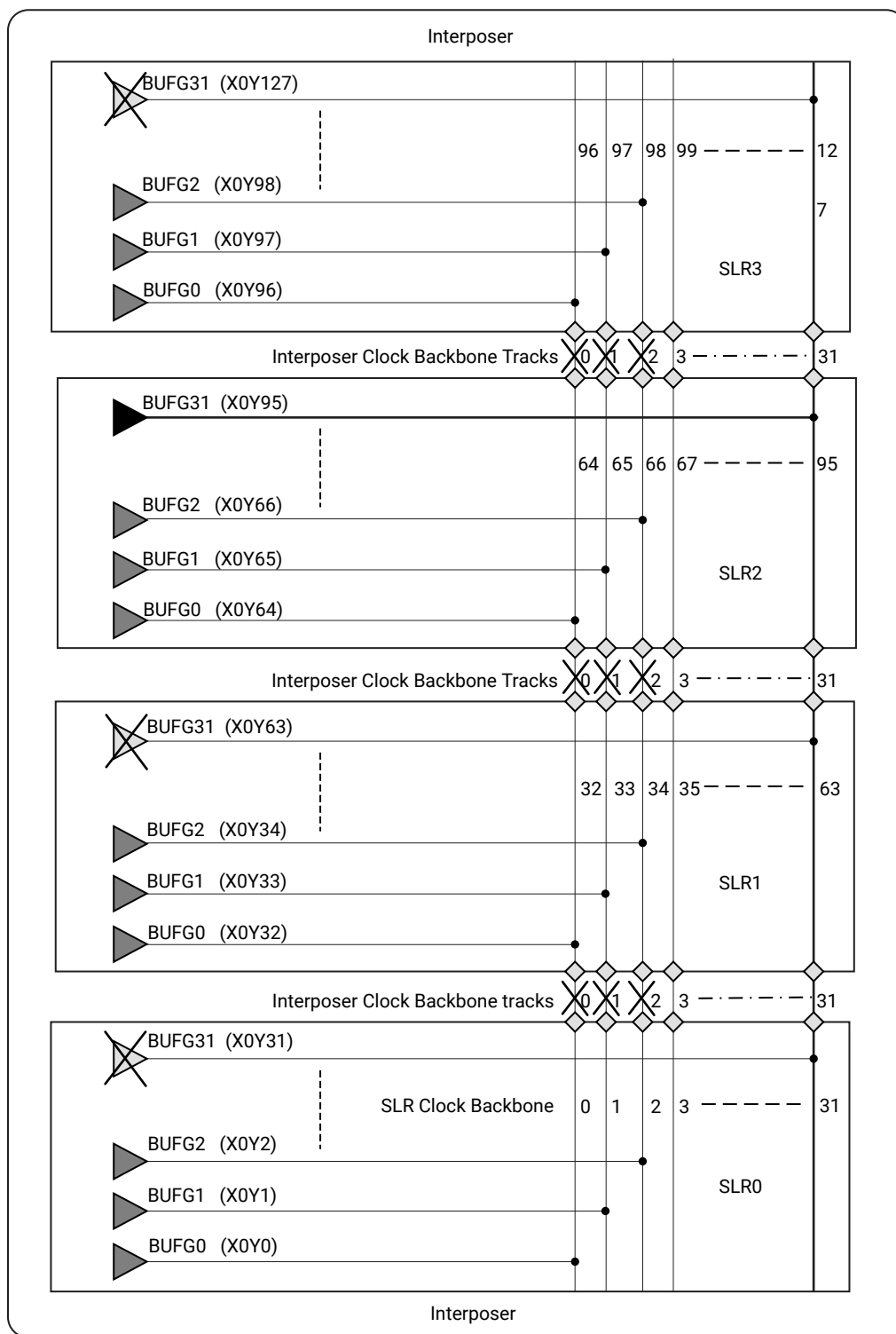
33 個以上のグローバル クロックが必要なデザインでは、ザイリンクスでは小さいクロック ドメインには BUFR および BUFH を使用して、必要なグローバル クロック ドメインの数を減らせるかどうかを試してみることをお勧めします。BUFR と BUFMR を一緒に使用すると、SLR の半分 (Virtex-7 クラスの SLR の場合、約 25 万個のロジック セル) を占める 3 つのクロック領域内のリソースを駆動できます。垂直方向に隣接したクロック領域では、左側と右側の BUFH 両方を小さいスキューで駆動できるので、SLR の 1/3 (約 16 万 7 千個のロジック セル) のクロック ドメインがイネーブルになります。

これらのリソースをできるだけ使用すると、クロック リソースの競合に関する考慮事項が減るだけでなく、全体的な配置も改善されることが多いので、パフォーマンスおよび消費電力も改善されます。

SLR 半分以上または複数の SLR を駆動するクロックが 33 個以上必要な場合は、BUFG グローバル クロック スパインを分割できます。SLR 周辺の垂直グローバル クロック ラインには絶縁バッファが存在するので、異なる SLR の同じ垂直グローバル クロック トラックにある 2 つの BUFG を競合なく使用できます。この機能を使用するには、ユーザーの制御および操作がさらに必要となります。次の図では、3 つの SLR にある BUFG0 から BUFG2 が絶縁されているので、それぞれの SLR 内に独立したクロックが含まれます。BUFG31 ラインは絶縁されていないので、同じ BUFG31 (図の SLR2 内) が 3 つの SLR すべてのクロック ラインを駆動するようにし、ほかの SLR の BUFG31 はディスエーブルにする必要があります。

BUFG を注意深く選択し、手動配置 (LOC) する必要があります。各クロック ドメインのすべてのロードを手動でグループ化し、適切な SLR に配置してクロック競合が発生しないようにする必要があります。すべてのグローバル クロックが配置され、すべてのロードがクロック競合がないように管理され、クロックがすべてのロードに到達できるようになっていると、32 個を超えるグローバル リソースを使用できるようになります。

図 79: SSI デバイスのクロック ラインの絶縁 (オプション)



X14051_122019

SSI テクノロジ デバイスのグローバル クロック リソースのクロック スキュー

高集積度デバイスでは、クロック スキューがパスの全体的なタイミング バジレットのかなりの部分を占めることがあります。クロック スキューが大きすぎると、最大クロック速度で問題が発生するだけでなく、ホールド タイム要件も厳しくなります。デバイスに複数のダイがあると、PVT 式のプロセス部分が悪化しますが、ザイリンクスのアセンブリ プロセスで管理されており、同程度の速度のダイのみがパッケージされます。

このような追加操作はありますが、ザイリンクス タイミング ツールではこれらの差異がタイミング レポートの一部として考慮されます。パス解析中には、これらの面がセットアップおよびホールド算出の一部として解析され、指定した要件に対するパス遅延の一部としてレポートされます。SSI テクノロジ デバイスでも、これらはタイミング解析 ツールで考慮されるので、別に算出したり考慮したりする必要はありません。

一番上または一番下の SLR を使用すると、距離が遠いほど遅延差も大きくなるので、スキューが増加します。このため、ザイリンクスでは複数の SLR を駆動するグローバル クロックは中央の SLR に配置することをお勧めします。これにより、パーツ全体にクロック ネットワークがより均一に分配され、全体的なクロック スキューが削減されます。

UltraScale デバイスをターゲットとするとクロック配置への影響は小さくなりますが、クロック挿入遅延およびクロックの消費電力を削減するため、クロック ソースをクロック ロードの中央のできるだけ近くに配置することを強くお勧めします。

クロック構造の設計

ここまででクロッキングに関する決定を下す際の主な考慮事項について説明したので、次にデザインに必要なクロッキングを達成する方法について説明します。

推論

Vivado 合成では、ユーザーが指定しなくても、すべてのクロック構造に対してアーキテクチャで許容される最大数までのグローバル バッファ (BUFG) が自動的に指定されます (特に指定した場合または合成ツールで制御される場合を除く)。前述のように、BUFG を使用すると、ほとんどのクロッキングのニーズに合った、詳細に制御されたスキューの小さいネットワークが提供されます。デザイン クロックがそのパーツの BUFG の数または機能を超過していなければ、これ以上の操作は必要ありません。

ただし、クロック構造をさらに制御すると、ジッター、スキュー、配置、消費電力、パフォーマンス、またはその他の特性などを改善することができます。

合成制約および属性

クロック リソースを制御する簡単な方法の 1 つは、CLOCK_BUFFER_TYPE 合成制約または属性を使用することです。合成制約は、次の目的で使用できます。

- BUFG の推論を回避。
- BUFG を代替クロック構造と置換。
- 存在しない場合にクロック バッファを指定。

合成制約を使用すると、コードを変更せずに、これらの制御を実行できます。

属性は、次のいずれかで指定できます。

- HDL コードに直接 (コード内に存続可能)

- XDC ファイルの制約として (HDL ソース コードに変更を加える必要なし)

IP の使用

IP の中には、クロック構造を作成できるものもあります。特に Clocking Wizard および IO Wizard を使用すると、次を含むクロック リソースおよび構造を簡単に選択および作成できます。

- BUFG
- BUFGCE
- BUFGCE_DIV (UltraScale デバイス)
- BUFGCTRL
- BUFIO (7 シリーズ デバイス)
- BUFR (7 シリーズ デバイス)
- 次のようなクロック調整ブロック:
 - MMCM (Mixed Mode Clocking Manager)
 - 位相ロック ループ (PLL) コンポーネント

PCIe または Transceiver Wizard などのさらに複雑な IP には、IP の一部としてクロック構造が含まれていることもあります。正しく考慮すれば、これによりクロック リソースが追加されます。正しく考慮しない場合、デザインの残りの部分でクロック オプションの一部が制限される可能性があります。

ザイリンクスでは、すべてのインスタンスシート済み IP のクロック要件、機能、リソースをよく理解し、できるだけデザインのほかの部分で利用できるようにしておくことをお勧めします。

関連情報

[IP \(Intellectual Property\) の使用](#)

インスタンスエーション

クロッキング構造を簡単に直接的な方法で制御するには、必要なクロック リソースを HDL デザインにインスタンスエートします。これにより、デバイスで使用可能な機能すべてにアクセスでき、それらを完全に制御できます。BUFGCE、BUFGMUX、BUFGCE、または追加のロジックおよび制御を必要とするその他のクロック構造を使用する場合、通常インスタンスエーションが唯一のオプションです。単純なバッファでも、デザインに直接インスタンスエートするのが必要な結果を得るのに最も迅速な方法であることがあります。

クロック リソースを管理する効率的な方法は、特にインスタンスエートする場合は、クロック リソースを別のエンティティか、コードの最上位またはその付近にインスタンスエートされたモジュールに含めることです。コードの最上位に記述すると、デザイン内の複数モジュールに分配しやすくなります。

クロック リソースが共有できる箇所および共有すべき箇所に注意してください。不必要なクロック リソースを作成すると、リソースが無駄になるだけでなく、通常消費電力が増加し、競合が発生する可能性が高くなるため全体的なインプリメンテーション ツールのコンパイル時間が長くなり、タイミング条件がさらに複雑になります。これが、クロック リソースを最上位モジュールの近くに含めるのが重要であるもう 1 つの理由です。



ヒント: 特定のクロック プリミティブをインスタンスエートするには、Vivado HDL テンプレートを使用できます。

関連情報

[Vivado Design Suite HDL テンプレートの使用](#)

クロックの位相、周波数、デューティ サイクル、およびジッターの制御

このセクションでは、クロック特性を詳細に調整する手法について説明します。

クロック調整ブロック (MMCM および PLL) の使用

MMCM または PLL を使用すると、入力クロックの全体的な特性を変更できます。MMCM は、クロックの挿入遅延を削除したり (入力システム同期データにクロックの位相を揃える)、次のようなクロックの特性を条件付けおよび制御するためによく使用されます。

- 位相をさらに厳しく制御
- クロックのジッターをフィルター
- クロック周波数を変更
- クロックのデューティ サイクルを修正または変更

MMCM または PLL を使用するには、MMCM が仕様範囲内で動作し、出力に必要なクロック特性を提供できるように、複数の属性を調整する必要があります。このため、ザイリンクスでは Clocking Wizard を使用してこのリソースを正しく設定することを強くお勧めします。

MMCM または PLL を直接インスタンス化すると、より詳細な制御が可能です。ただし、次のような問題を回避するため、適切な設定を使用してください。

- ジッターの増加によりクロックのばらつきが増加する
- 正しくない位相関係が構築される
- タイミング クロージャが困難になる



重要: MMCM または PLL を設定するのに Clocking Wizard を使用すると、デフォルトで Clocking Wizard により妥当な消費電力特性が使用され、出力ジッターが小さくなるよう MMCM が設定されます。

目的によって Clocking Wizard の設定を変更してジッターを最小限に抑えると、消費電力は大きくなりますがタイミングは向上します。消費電力を削減すると、出力ジッターが増加します。

MMCM または PLL を使用する場合は、次のガイドラインに従ってください。

- 入力を未接続のままにしないでください。合成ツールまたはその他の最適化ツールで未接続の入力が接続されるようにすると、接続される値が必要な値と異なる可能性があるため、推奨されません。
- RST はユーザー ロジックに接続し、信頼性のあるクロック ソースで制御されるロジックでアサートできるようにする必要があります。RST をグランド接続すると、クロックが中断した場合に問題となることがあります。
- リセットのインプリメンテーションに LOCKED 出力を使用します。たとえば、PLL からクロックが供給される同期ロジックは、LOCKED がアサートされるまでリセットのままにします。LOCKED 信号は、デザインの同期部分で使用する前に同期する必要があります。ザイリンクスでは、LOCKED 信号をプロセッサ マップに追加し、デバッグするときに表示されるようにすることをお勧めします。
- CLKFBIN および CLKFBOUT 間の接続を確認してください。たとえば ZHOLD 補正モードを使用する際、PLL/MMCM 出力クロックの位相を入力基準クロックと揃える必要がある場合は、BUFG はフィードバック パスにのみ含める必要があります。

- UltraScale デバイスで同期クロック乗せ換えパスでの MMCM または PLL 位相エラーによるタイミングへの悪影響を回避するには、BUFGCE ではなく BUFGCE_DIV を参照してください。



推奨: Clocking Wizard 内でさまざまな設定を試して、全体的なデザイン目標を達成するために最も適切な設定が作成されるようにしてください。

関連情報

同期 CDC

クロックに IDELAY を使用した位相制御

7 シリーズ デバイスでは、位相を少しだけ調整する必要がある場合は、MMCM または PLL の代わりに IDELAY または ODELY を使用して遅延を追加できます。これにより、関連データに対するクロックの位相オフセットが増加します。UltraScale デバイスでは、入力クロック ソースに IDELAY を使用することはできません。ザイリンクスでは、位相の調整が必要な場合は MMCM を使用することをお勧めします。

ゲーテッド クロックの使用

ザイリンクス デバイスには、ファンアウトが大きく、スキューの小さいクロック リソースを提供可能な専用クロック ネットワークが含まれています。HDL コードに細粒度クロック ゲーティング手法が含まれていると、専用クロック リソースの機能が妨害され、効率的に使用されない可能性があります。このため、ザイリンクスでは、特定のデバイスをターゲットとする HDL を記述する場合はクロック ゲーティングの構文をクロック パスに記述することはお勧めしません。デザインの特定の部分の機能または消費電力を停止するには、クロック イネーブルを推論するコードを使用してクロック供給を制御するようにしてください。

クロック ゲーティングをクロック イネーブルに変換

ザイリンクスでは、コードに既にクロック ゲーティング構文が含まれる場合、またはそのようなコーディング スタイルを必要とする別のテクノロジー用である場合は、合成ツールを使用してクロック パス内に配置されたゲートをそのデータパスのクロック イネーブルにマップし直すことをお勧めします。これにより、クロック リソースへのマップが改善され、ゲーティングされているドメインに入力されるデータおよびそのドメインから出力されるデータの回路のタイミング解析を単純化できます。たとえば、Vivado 合成で `-gated_clock_conversion auto` オプションを使用し、自動的にレジスタ クロック イネーブル ロジックに変換されるようにします。複雑なゲーテッド クロック構造では、RTL コードで GATED_CLOCK 属性を使用して Vivado 合成での処理を制御します。

クロック バッファのゲーティング

クロック ネットワークの大部分を一定の期間シャットダウン可能な場合、BUFGCE または BUFGCTRL を使用してクロック ネットワークをイネーブルまたはディスエーブルにできます。また、UltraScale デバイスを使用する場合は、BUFGCE_DIV および BUFG_GT をゲーティングできます。7 シリーズ デバイスでは、クロックのゲーティングに BUFHCE、BUFR、および BUFMRCE も使用できます。

クロックを一定期間低速にできる場合は、これらのバッファを追加ロジックと共に使用して定期的にクロック ネットをイネーブルにできます。または、BUFGMUX または BUFGCTRL を使用して、クロック ソースをより高速のクロック信号からより低速のクロックに切り替えることもできます。

これらの手法のいずれを使用しても、ダイナミック消費電力を効果的に削減できますが、要件およびクロック トポロジによって、この中の 1 つの手法がほかの手法よりも効果的である可能性があります。次に、7 シリーズ デバイスでの例を示します。

- 3 つまでのクロック領域に供給される外部生成クロック (450 MHz 未満) には、BUFR が最適である可能性があります。
- Virtex-7 デバイスの場合、複数のクロック領域 (垂直に隣接した 3 つまでの領域) でこの手法を使用するには、BUFMRCE も必要となることがあります。
- BUFGCE は、1 つのクロック領域に制限できる高速クロックに適しています。BUFGCE はデバイス全体を駆動でき、最も柔軟ですが、消費電力の面では最適な選択とは言えません。

デバイス スタートアップの制御と同期化

デバイスのコンフィギュレーションが完了すると、デバイスがコンフィギュレーション ステートを完了し、通常動作に移行するためにイベントのシーケンスが実行されます。ほとんどのコンフィギュレーション シーケンスでは、最後の手順の 1 つとしてグローバル セット/リセット (GSR) がディアサートされ、その後グローバル イネーブル (GWE) 信号がディアサートされます。これでデザインが既知の初期ステートになり、動作を開始できるようになります。

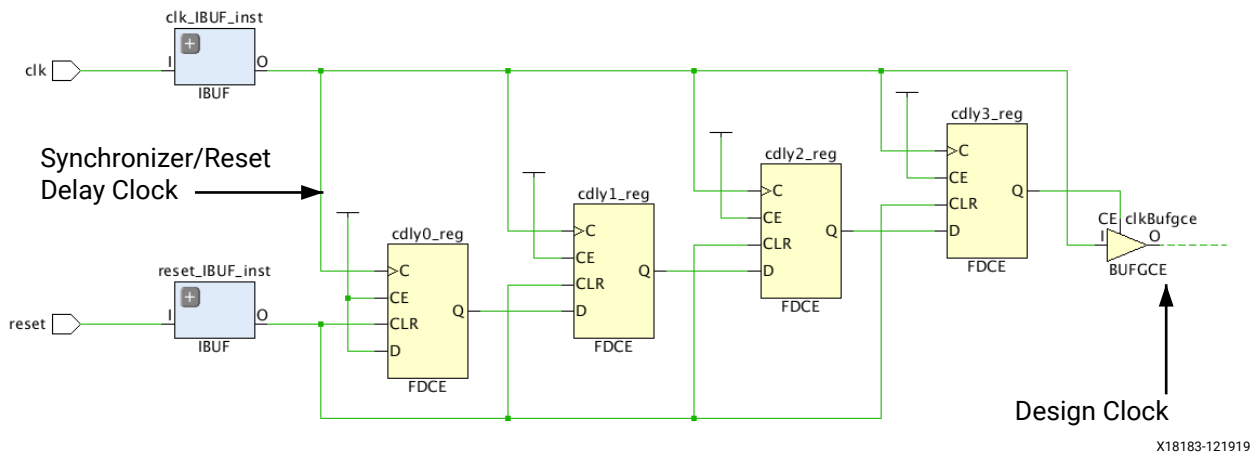
このディアサートが指定のクロック ドメインに同期していないか、クロックが高速で動作しているために GWE を安全にディアサートできない場合、デザインの一部分が不明なステートになる可能性があります。これが問題とならないデザインもありますが、デザインによってはこれが原因で不安定な状態になったり、初期データ セットが正しく処理されなかったりする可能性があります。

デザインを既知のステートで開始させる必要がある場合は、サイリンクスでは次のいずれかの方法を使用してスタートアップ時の同期プロセスを制御することをお勧めします。

- ステート マシンなどのデザインのクリティカルな部分にクロック イネーブル、ローカル リセット (同期)、またはその両方を使用して、これらの部分のスタートアップを制御し、既知のものになるようにします。
- クロック イネーブル機能を持つクロック バッファ コンポーネントをインスタンス化します。

デザイン クロックをイネーブルにする前に、必要なクロック数だけリセットのディアサートを遅らせます。次に、UltraScale デバイスでリセットをディアサートした後の最初のデザイン クロック エッジを遅延させる方法の例を示します。シンクロナイザー レジスタに `ASYNC_REG=TRUE` を設定することにより、すべてのレジスタが 1 つのスライスに配置されるので、グローバル クロック リソースで駆動する必要はありません。シンクロナイザー クロックにクロック バッファが挿入されないようにするには、入力クロック ポートに `CLOCK_BUFFER_TYPE=NONE` プロパティを使用します。

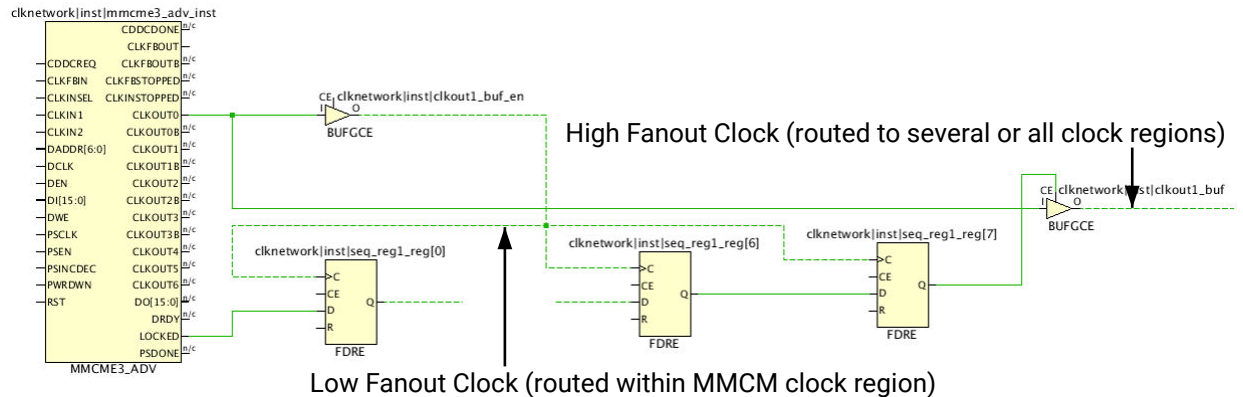
図 80: 安全にクロックを開始するためのリセットの同期化と遅延の例



- MMCM を使用する場合は、Clocking Wizard で [Safe Clock Startup] オプションをオンにして、デザイン クロックが安定した後のみにイネーブルになるようにします。

次の例に、UltraScale デバイスでユーザー ロジックを駆動する BUFGCE の CE ピンに接続された MMCM LOCKED 信号の同期化段を示します。2 番目の BUFGCE はファンアウトの大きい BUFGCE (ユーザー クロック) に並列に接続され、BUFGCE/CE ピンを制御するロジック専用で使用されます。このトポロジは、シンクロナイザーと BUFGCE ピンの間のクロック スキューを最小限に抑えることにより、UltraScale デバイスの BUFGCE/CE で タイミング クロージャを達成するのに有益です。

図 81: MMCM での安全なクロック開始例



X18185-020321



ヒント: MMCM または PLL の補正モードが ZHOLD または BUF_IN に設定されている場合、CLKOUT0 からのすべてのクロックがフィードバック クロックと共にグループ化され、同じ CLOCK_ROOT が使用されます。これにより BUFGCE/CE でタイミング違反が発生する場合は、ファンアウトの大きいクロックとフィードバッククロックの間にのみ CLOCK_DELAY_GROUP 制約を作成します。オプションで、ファンアウトの小さいクロック ネットに USER_CLOCK_ROOT 制約を設定し、MMCM と同じクロック領域へのロードに制約を設定することもできます。7 シリーズ デバイスでは、クロッキング アーキテクチャが異なるので、タイミング クロージャを達成するために 2 番目のクロック バッファは通常必要ありません。

ローカル クロックを避ける

ローカル クロックは、専用グローバル クロック リソースではなく通常のファブリック リソースを使用して配線されるクロック ネットです。ほとんどの場合、Vivado 合成および Vivado ロジック最適化により、アーキテクチャで必要な位置またはクロック ロードが 30 を超えるクロック ネットにクロック バッファが挿入されます。ローカル クロックは、次のような場合に使用されます。

- グローバル クロックがファブリック ロジックにインプリメントされたカウンタにより分周される
- クロック ゲーティング変換でクロック パスからすべての LUT を削除できない
- 7 シリーズ デバイスで使用されているクロック バッファが多すぎる

注記: UltraScale デバイスには 7 シリーズ デバイスよりも多くのクロック バッファがあり、ファンアウトの小さいクロック バッファの使用は通常問題となりません。

通常は、ローカル クロックの使用は避けてください。ローカル クロックを使用すると、インプリメンテーションで次のような問題が発生する可能性があります。

- クロック スキューが予測不可能になり、タイミング クロージャを達成するのが困難になる
- 配線で特別な配慮が必要なファンアウトが小から中のネットが増加し、配線性の問題が発生する可能性がある



ヒント: ローカル クロックによりタイミング QoR の問題が発生する場合は、Pblock を使用してクロック ドライバーとロードをフロアプランしてみてください。report_clock_utilization を使用してローカル クロックの位置を特定し、クロック配置を確認して、その数を削減するか影響を低減する方法を決定します。

出力クロックの作成

デバイス外部のデバイスにクロックを供給するためにデバイスからクロックをフォワードするには、ODDR コンポーネントを使用するのが効率的です。入力の 1 つを High、それ以外を Low に接続すると、位相関係およびデューティサイクルが適切に制御されたクロックを簡単に作成できます。たとえば、D1 ピンを 0 に、D2 ピンを 1 に保持すると、180 度の位相シフトを達成できます。セット/リセットおよびクロック イネーブルを使用すると、クロックを停止したり、一定期間極性を変更したりできます。

外部クロックにさらに詳細な位相制御が必要な場合は、MMCM または PLL を外部フィードバック補正、粗粒度または細粒度の固定または可変の位相補正と共に使用できます。これにより、ほかのデバイスへのクロック位相および伝搬時間を詳細に制御して、そのデバイスからの外部タイミング要件を単純化できます。

クロック乗せ換え

デザインのクロック乗せ換え (CDC) 回路は、デザインの信頼性に直接影響します。独自の回路を設計することも可能ですが、Vivado Design Suite でその回路が認識されるようにし、ASYNC_REG 属性を正しく設定する必要があります。ザイリンクスでは、次のように正しい回路デザインを確実にするため、XPM を提供しています。

- place_design で同期回路の平均故障間隔 (MTBF) を削減する特定の機能を実行する。
- report_synchronizer_mtbtf で確実に認識されるようにする。
- report_cdc のエラーおよび警告を回避する。これらのエラーおよび警告は、イテレーションが長いデザイン サイクルの後の方でよく発生します。



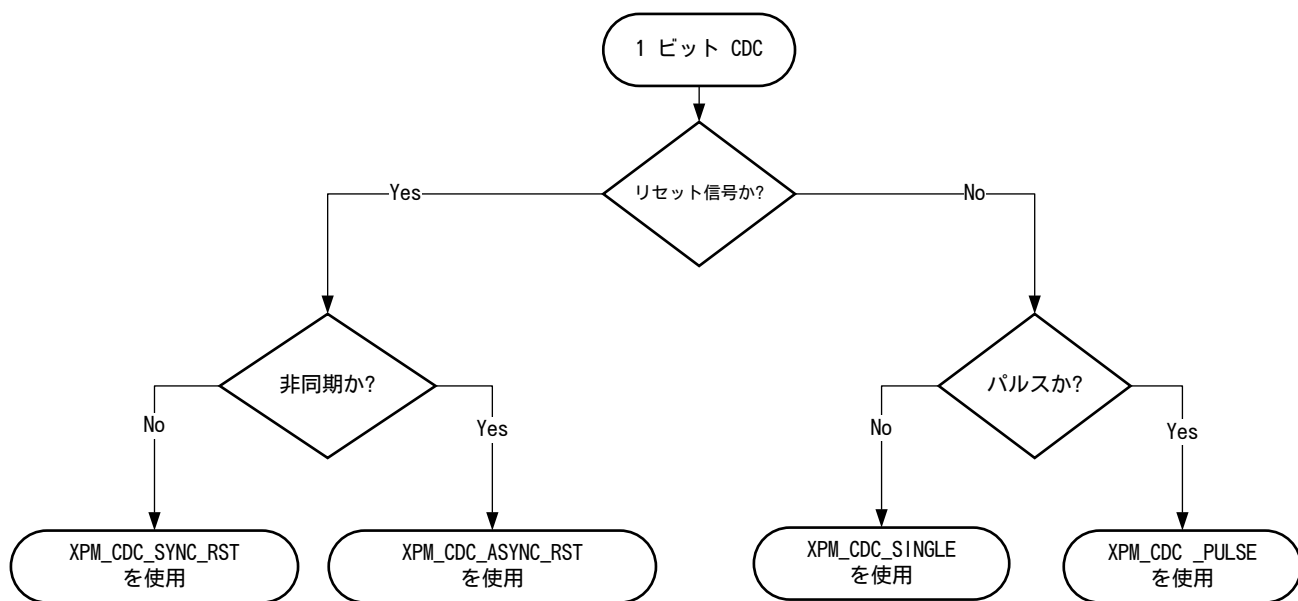
ヒント: 安全に無視できる CDC 違反は、除外してレポートされないようにすることができます。詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906) のこのセクションを参照してください。

非同期クロック乗せ換えの場合、または 2 つの同期クロック間のタイミングを緩和するためにフォルス パス制約を追加する場合は、CDC 回路が必要です。XPM を使用すると、クロック乗せ換えに 1 ビットまたは複数ビット バスを選択できます。

1 ビット CDC

次の図に、1 ビット CDC を使用する場合に必要な決定事項を示します。

図 82: 1 ビット CDC を使用する際の決定事項



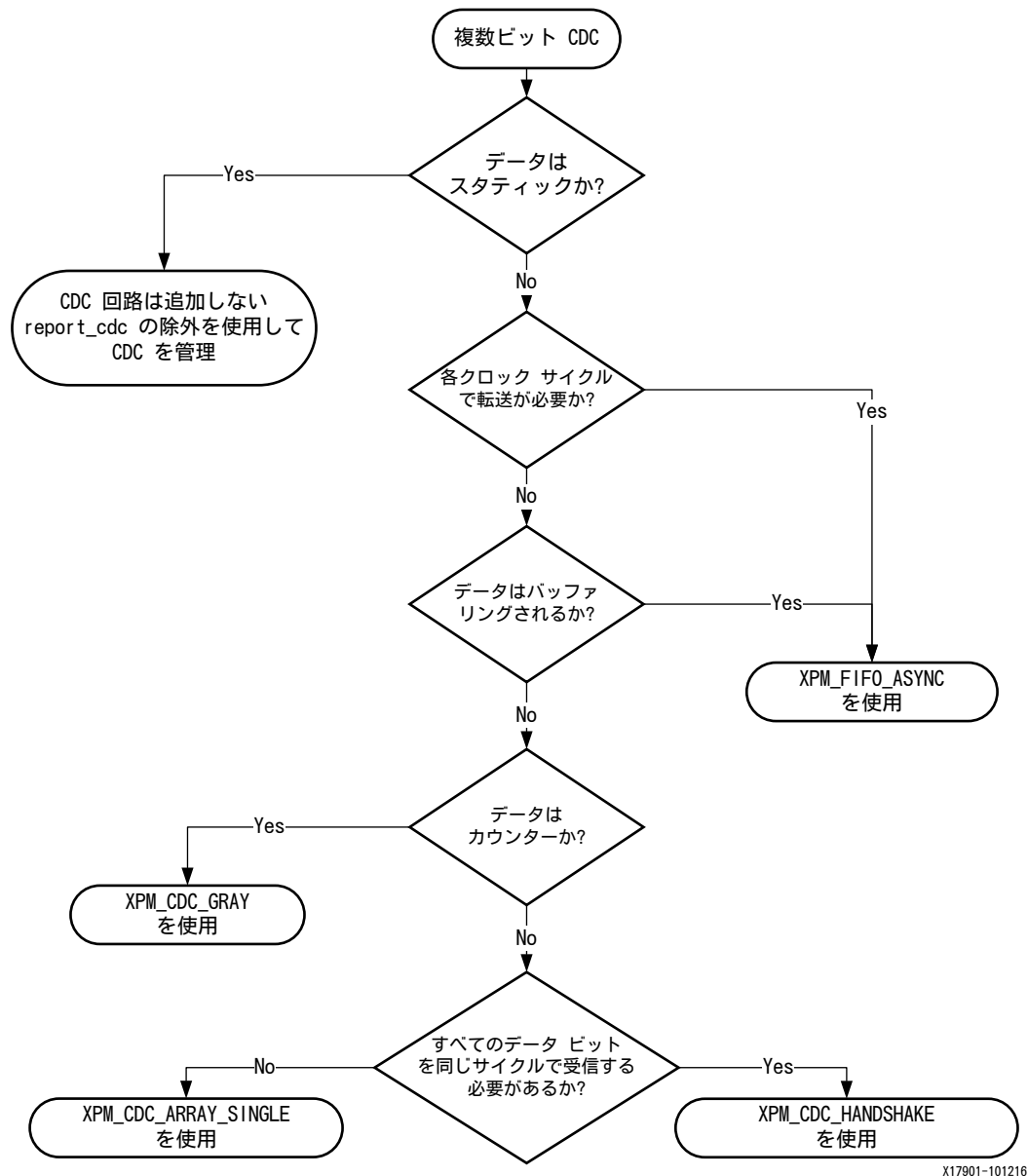
X17900-101016

注記: 異なる 1 ビット シンクロナイザーの詳細は、ご使用のデバイスのライブラリ ガイドを参照してください。

複数ビット CDC

次の図に、複数ビット CDC を使用する場合に必要な決定事項を示します。

図 83: 複数ビット CDC を使用する際の決定事項



注記: 異なる複数ビット シンクロナイザーの詳細は、ご使用のデバイスのライブラリ ガイドを参照してください。

MTBF のための最適化

デザインの MTBF は、次のものの関数です。

- シンクロナイザー MTBF
- シングル イベント アップセット (SEU) によるデバイスの FIT (Failure In Time) 率

注記: SEU によるデバイスの FIT 率は、主にプロセスおよびデバイスのサイズによります。

シンクロナイザー MTBF は、デザインおよび次の要因によって異なります。

- 非同期 CDC 点の数
- 各クロック乗せ換え点のシンクロナイザーの段数
- デスティネーション FF の周波数
- ソースのトグル レート

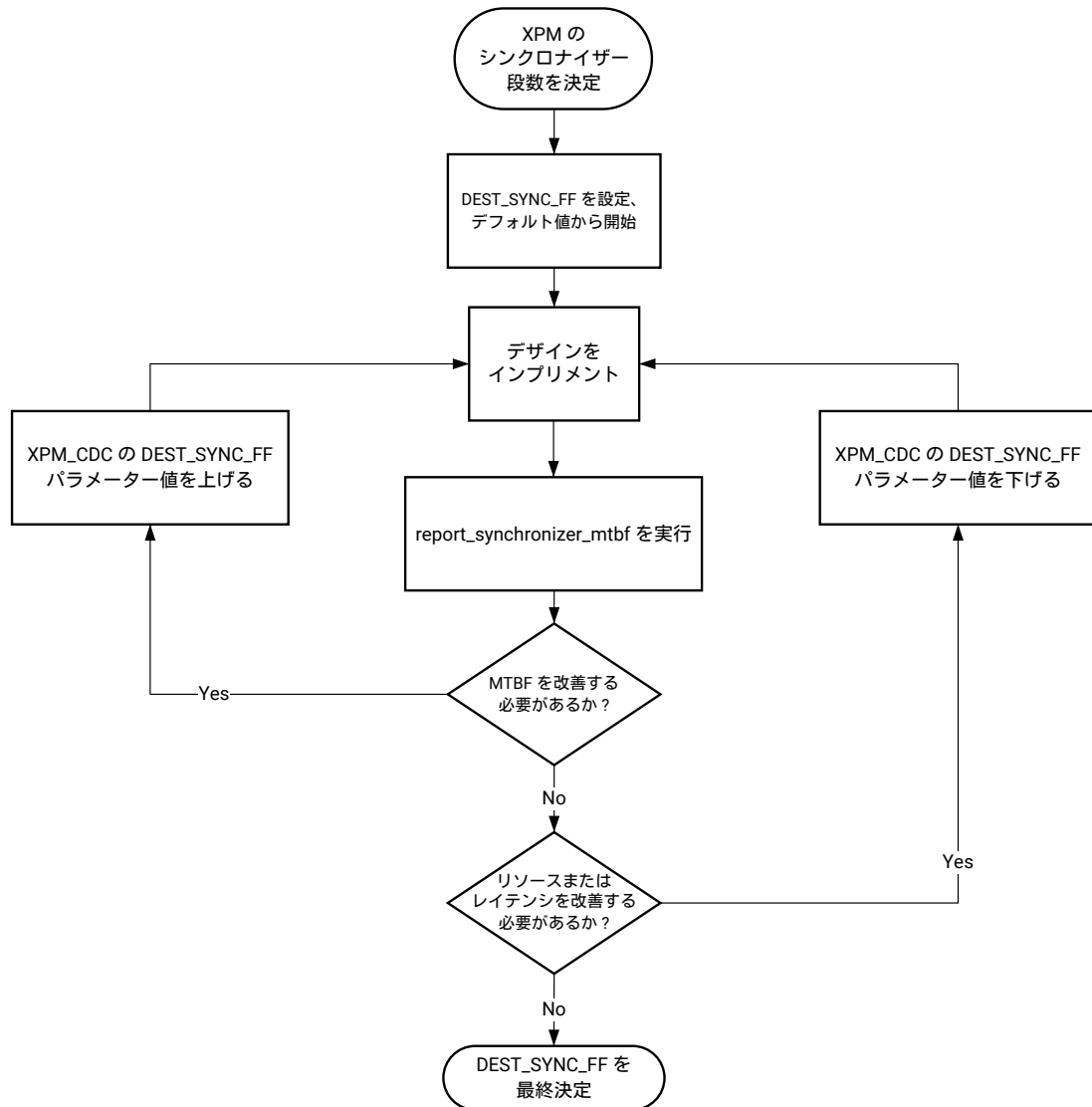
DEST_SYNC_FF パラメーターに正しい値を選択

XPM CDC モジュールを使用する場合、DEST_SYNC_FF パラメーターはメタスタビリティ 保護レジスタの数を設定します。このレジスタの値は、MTBF、デザイン サイズ、およびクロック乗せ換え点でのレイテンシに影響します。このレジスタに正しい値を選択することは、反復プロセスであり、次の手順が必要です。

1. デザインで Vivado Design Suite インプリメンテーション フローを実行します。
2. ターゲット デバイスによって、次のいずれかを実行します。
 - 7 シリーズ デバイスでは、DEST_SYNC_FF のデフォルト値を選択します。これは、典型的な信頼性要件を満たすための控えめな方法です。重要なデザインでは、さらに解析を実行します。
 - UltraScale デバイスでは、デザイン全体の MTBF をレポートする `report_synchronizer_mtbfs` コマンドを実行します。次の図に示すようにフローを繰り返すことにより、MTBF、レイテンシ、およびリソース間の適切なトレードオフを見つけることができます。

注記: この反復プロセスは、すべての同期レジスタに ASYNC_REG 属性が正しく適用されているユーザー CDC 回路にも使用できます。

図 84: UltraScale デバイスのシンクロナイザー MTBF 最適化フロー



X17899-122019

デザインを正しく制約

XPM CDC には、独自の `set_max_delay -datapath-only` 制約があります。`set_clock_groups` 制約は優先度が高く、XPM の制約が無効になってしまうので、XPM CDC と共に使用することはできません。

関連情報

[クロック グループおよび CDC 制約の定義](#)

デザイン制約

デザイン制約は、デザインがハードウェアで正しく機能するようにするために、コンパイル フローで満たす必要のある要件を定義します。複雑なデザインの場合は、ツールに対してガイダンスを定義してタイミング クロージャを達成しやすくすることもできます。すべての制約がコンパイル フローのすべての段階で使用されるわけではありません。たとえば、物理制約はインプリメンテーション段階 (最適化、配置、および配線) でのみ使用されます。

合成およびインプリメンテーション アルゴリズムはタイミングドリブンなので、適切なタイミング制約を作成することが重要になります。デザインの制約を厳しくしすぎたり緩くしすぎたりすると、タイミング クロージャを達成するのが困難になります。アプリケーションの要件に合った適度な制約を使用する必要があります。制約に関する詳細は、次の資料を参照してください。

- 『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』 ([UG906](#))
- ザイリンクス ウェブサイトの [Vivado Design Suite ビデオ チュートリアル](#) ページにあるデザイン制約の適用に関するビデオ チュートリアル

デザイン制約の分類

制約は通常、1 つまたは多数のファイルに、カテゴリ別、デザイン モジュール別、またはその両方で分類されます。分類方法に関係なく、制約の全体的な依存性を理解して、メモリに読み込まれた後の最終的なシーケンスを確認する必要があります。たとえば、タイミング クロックはほかの制約で使用される前に定義する必要があるため、その定義が制約ファイルの最初 (メモリに読み込まれる最初の制約セット) に含まれるようにする必要があります。

推奨される制約ファイル

プロジェクトの大きさや複雑さによって、制約方法はさまざまです。次に、推奨事項の一部を示します。

単純なデザイン

少数の設計者チームによる単純なデザインの場合:

- すべての制約に対して 1 ファイル
- 物理制約用に 1 ファイル + タイミング制約用に 1 ファイル
- 物理制約用に 1 ファイル + タイミング (合成) 制約用に 1 ファイル + タイミング (インプリメンテーション) 制約用に 1 ファイル

複雑なデザイン

IP コアまたは複数設計者のチームによる複雑なデザインの場合:

- 最上位タイミング制約用に 1 ファイル + 最上位物理制約用に 1 ファイル + IP/主なブロックごとに 1 ファイル

読み出しシーケンスの検証

プロジェクト制約ファイルの分類方法が決まったら、ファイルの内容に基づいて、ファイルの読み出しシーケンスを確認する必要があります。プロジェクト モードでは、Vivado® IDE または `reorder_files` Tcl コマンドを使用して制約ファイル シーケンスを変更できます。非プロジェクト モードでは、コンパイル フローの Tcl スクリプトで `read_xdc` コマンド (XDC ファイルの場合) および `source` コマンド (Tcl スクリプトで生成された制約の場合) を使用してシーケンスを定義します。

推奨される制約の順序

制約言語 (XDC) は Tcl 構文および解釈規則に基づいています。Tcl と同様、XDC は逐次言語です。

- 変数は使用される前に定義する必要があります。タイミング クロックも、ほかの制約で使用するまえに定義します。
- 優先度が同じ制約が同じパスに適用されている場合、後に指定された制約が適用されます。

上記の優先規則を考慮すると、タイミング制約には次の順序が使用されます。

```
## Timing Assertions Section
# Primary clocks
# Virtual clocks
# Generated clocks
# Delay for external MMCM/PLL feedback loop
# Clock Uncertainty and Jitter
# Input and output delay constraints
# Clock Groups and Clock False Paths
## Timing Exceptions Section
# False Paths
# Max Delay / Min Delay
# Multicycle Paths
# Case Analysis
# Disable Timing
```

複数の XDC ファイルを使用する場合は、クロック定義に特に注意して、依存関係が正しい順序になっているかどうかを確認する必要があります。

物理制約は、制約ファイルのどこにでも記述できます。

合成制約の作成

合成では、タイミング ドリブン アルゴリズムが使用され、デザインの RTL 記述が、デバイスにマップされた最適化済みネットリストに変換されます。QoR (結果の品質) は、RTL コードのクオリティおよび指定された制約に影響されます。コンパイル フローのこの段階では、ネット遅延は概算であり、配置制約や密集などの複雑な状況は反映されません。主な目的は、タイミングを満たすか、タイミングが少しの差で満たされないようなネットリスト (現実的でシミュレーション可能な制約を含む) を得ることです。

合成エンジンではすべての XDC コマンドが認識されますが、実際に影響するのは一部です。

- セットアップ/リカバリ解析に関するタイミング制約は、QoR (結果の品質) に影響します。
 - `create_clock/create_generated_clock`
 - `set_input_delay/set_output_delay`
 - `set_clock_groups/set_false_path/set_max_delay/ set_multicycle_path`

- 次のホールドおよびリムーバブル解析に関するタイミング制約は合成中は無視されます。

- `set_min_delay / set_false_path -hold/set_multicycle_path -hold`

- RTL 属性を指定すると、マップおよび最適化アルゴリズムによる決定が強制されます。次に、その例をいくつか示します。

- `DONT_TOUCH/KEEP/KEEP_HIERARCHY/MARK_DEBUG`
- `MAX_FANOUT`
- `RAM_STYLE / ROM_STYLE / USE_DSP / SHREG_EXTRACT`
- `FULL_CASE/PARALLEL_CASE` (Verilog RTL のみ)

注記: 同じ属性を XDC からのプロパティとして設定することもできます。XDC ベースの制約を使用すると、場合によっては RTL を変更せずに合成結果に影響を与えることができるので便利です。

- 物理制約 (LOC、BEL、Pblock) は無視されます。

合成制約はエラボレート済みネットリストからの名前 (できればポートおよびシーケンシャル セル) を使用する必要があります。RTL 信号の中にはエラボレーション中に削除されるものもあり、それらには XDC 制約は適用できません。また、エラボレーション後のさまざまな最適化のために、ネットまたは論理セルが LUT や DSP ブロックなどのさまざまなデバイスのプリミティブに統合されます。デザイン オブジェクトのエラボレート済みの名前を確認するには、Flow Navigator で [Open Elaborated Design] をクリックし、該当する階層を参照します。

一部のレジスタは RAM ブロックに吸収され、境界をまたぐ最適化を実行できるように階層レベルが削除されることもあります。

エラボレート済みネットリスト オブジェクトまたは階層レベルは、`DONT_TOUCH`、`KEEP`、`KEEP_HIERARCHY`、または `MARK_DEBUG` 制約を使用して保持することもできますが、タイミングやエリアの QoR (結果の品質) は悪化する可能性があります。

最後に、制約の中には、競合するために合成で適用されないものもあります。たとえば、`MAX_FANOUT` 属性が複数の階層レベルにまたがるネットに設定され、階層の一部が `DONT_TOUCH` で保持される場合、ファンアウト最適化は制限されるか、完全に回避されます。



重要: インプリメンテーション中と異なり、タイミング制約を定義するのに使用される RTL ネットリスト オブジェクトが、エリア QoR 改善のため合成の最適化で削除されることがあります。これは、制約がアップデートされインプリメンテーション用に検証されていれば、通常問題とはなりません。必要であれば、`KEEP` 制約を使用して、合成とインプリメンテーションの両方でオブジェクトを保持できます。

ザイリンクスでは、合成が終了したら、タイミングおよび使用量レポートで、ネットリストがアプリケーション要件を満たしているか、インプリメンテーションに使用できるかを確認することをお勧めします。

インプリメンテーション制約の作成

インプリメンテーション制約は、最終的なアプリケーションの要件を正確に反映させる必要があります。I/O ロケーションおよび I/O 規格などの物理制約は、ボード トレース遅延を含め、ボード デザインで検出されます。全体的なシステム要件からのデザイン内部要件も含まれます。ザイリンクスでは、インプリメンテーションに進む前に、すべての制約が正しいかどうかを確認しておくことをお勧めします。制約が正しく設定されていないと、インプリメンテーションの QoR が悪化し、サインオフ用のタイミングの信頼性レベルも下がる可能性があります。

多くの場合、合成とインプリメンテーションで同じ制約を使用できますが、合成中にデザイン オブジェクトが削除されたり、名前が変更される可能性があるため、インプリメンテーション ネットリストですべての合成制約が正しく適用されているかを検証する必要があります。適用されていない場合は、インプリメンテーションのみで有効な制約を含む XDC ファイルを作成する必要があります。

ブロック レベルの制約の作成

チーム プロジェクトの場合、最上位デザインの主なブロックそれぞれに個別の制約ファイルを作成すると便利です。これらのブロックは個別に開発および検証された後、最終的に 1 つまたは多くの最上位デザインに統合されます。

ブロック レベル制約は、最上位制約とは別に作成する必要がある、さまざまなコンテキストで使用できるように、できるだけ一般的に記述する必要があります。また、これらの制約がブロックの境界を超えるロジックには影響しないようにする必要があります。

サブブロックをインプリメントする場合は、クロッキング ネットワーク全体をタイミング解析に含め、スキューおよびクロック乗せ換え解析が正確に実行されるようにすることをお勧めします。これには、クロック コンポーネントを含む HDL ラッパーと、最上位クロック制約を複製する追加の制約ファイルが必要な場合があります。これは、サブモジュールのタイミング検証でのみ使用されます。

制約の適用範囲および規則、ブロック レベル制約を最上位デザインに読み込むメカニズムの詳細は、『Vivado Design Suite ユーザー ガイド: 制約の使用』(UG903)の[このセクション](#)を参照してください。

Vitis 環境の制約の指定

Vitis™ 環境では、ハードウェア カーネルを C/C++ カーネルまたは RTL カーネルとして指定できます。

- C/C++ カーネルを使用する場合は、Vitis HLS を使用して、合成またはインプリメンテーション用の追加のユーザー制約を指定する必要があります。Vitis HLS からの出力は、IP パッケージャーを使用してパッケージする必要があります。このパッケージされた IP には、ユーザー制約とツールで生成された制約の両方が含まれます。詳細は、『Vitis 統合ソフトウェア プラットフォームの資料』(UG1416)のアプリケーション アクセラレーション開発フローの [Vitis HLS 資料](#) を参照してください。
- RTL カーネルを使用する場合、IP パッケージで追加の合成制約およびインプリメンテーション制約を指定する必要があります。詳細は、『Vivado Design Suite ユーザー ガイド: カスタム IP の作成とパッケージ』([UG1118](#))を参照してください。

Vitis 環境では、合成およびインプリメンテーションのデザイン制約はすべて、IP と共にパッケージする必要があります。IP のパッケージ後に合成用に追加の制約を追加する必要がある場合は、それらの制約を含めて IP をパッケージし直す必要があります。

インプリメンテーションで使用する追加の XDC 制約は、IP をパッケージした後に指定できます。Vitis 環境では、Vivado ツールによりプログラマブル ロジック領域をインプリメントするプロセスは抽象化されますが、Vitis 環境から Vivado ツール フローを制御するアドバンス オプションも提供されています。これらのアドバンス制御を使用すると、`init_design`、`opt_design`、`place_design`、`phys_opt_design`、`route_design`、`write_bitstream`などの各インプリメンテーション段階の前後に実行する Tcl スクリプトを指定できます。Tcl スクリプト機能の詳細は、『Vivado Design Suite ユーザー ガイド: Tcl スクリプト機能の使用』([UG894](#))を参照してください。プリ/ポスト Tcl スクリプトを利用すると、`read_xdc` または `source` Tcl コマンドを使用して追加の XDC 制約を適用するなど、特定の Vivado ツール コマンドを実行できます。

プリ/ポスト Tcl スクリプトは、Vitis 環境設定ファイルを使用して指定するか、または `v++` コンパイラのコマンド ラインで直接指定できます。

Vitis 環境設定ファイル内でプリ/ポスト Tcl スクリプトを指定するには、`[vivado]` セクション内で `prop=run.impl_1.STEP.<PHASE>.TCL.<PRE|POST>` パラメーターを使用します。

説明:

- `<PHASE>`: インプリメンテーションの段階を指定します。有効な値は、`INIT_DESIGN`、`OPT_DESIGN`、`PLACE_DESIGN`、`PHYS_OPT_DESIGN`、`ROUTE_DESIGN`、または `WRITE_BITSTREAM` です。
- `PRE`: スクリプトを指定したインプリメンテーション段階の前に実行します。

- POST: スクリプトを指定したインプリメンテーション段階の後に実行します。

次に例を示します。

```
[vivado]
prop=run.impl_1.STEPS.OPT_DESIGN.TCL.PRE=<pathToTclScript>
prop=run.impl_1.STEPS.OPT_DESIGN.TCL.POST=<pathToTclScript>
prop=run.impl_1.STEPS.PLACE_DESIGN.TCL.PRE=<pathToTclScript>
prop=run.impl_1.STEPS.PLACE_DESIGN.TCL.POST=<pathToTclScript>
prop=run.impl_1.STEPS.PHYS_OPT_DESIGN.TCL.PRE=<pathToTclScript>
prop=run.impl_1.STEPS.PHYS_OPT_DESIGN.TCL.POST=<pathToTclScript>
prop=run.impl_1.STEPS.ROUTE_DESIGN.TCL.PRE=<pathToTclScript>
prop=run.impl_1.STEPS.ROUTE_DESIGN.TCL.POST=<pathToTclScript>
```

プリ/ポスト Tcl スクリプトを v++ パラメーターとして指定するには、`--vivado.prop run.impl_1.STEP.<PHASE>.TCL.<PRE|POST>=<pathToTclScript>` コマンド ライン オプションを使用します。たとえば、Tcl スクリプトを `opt_design` の前に実行するには、次のように指定します。

```
--vivado.prop run.impl_1.STEP.OPT_DESIGN.TCL.PRE=<pathToTclScript>
```

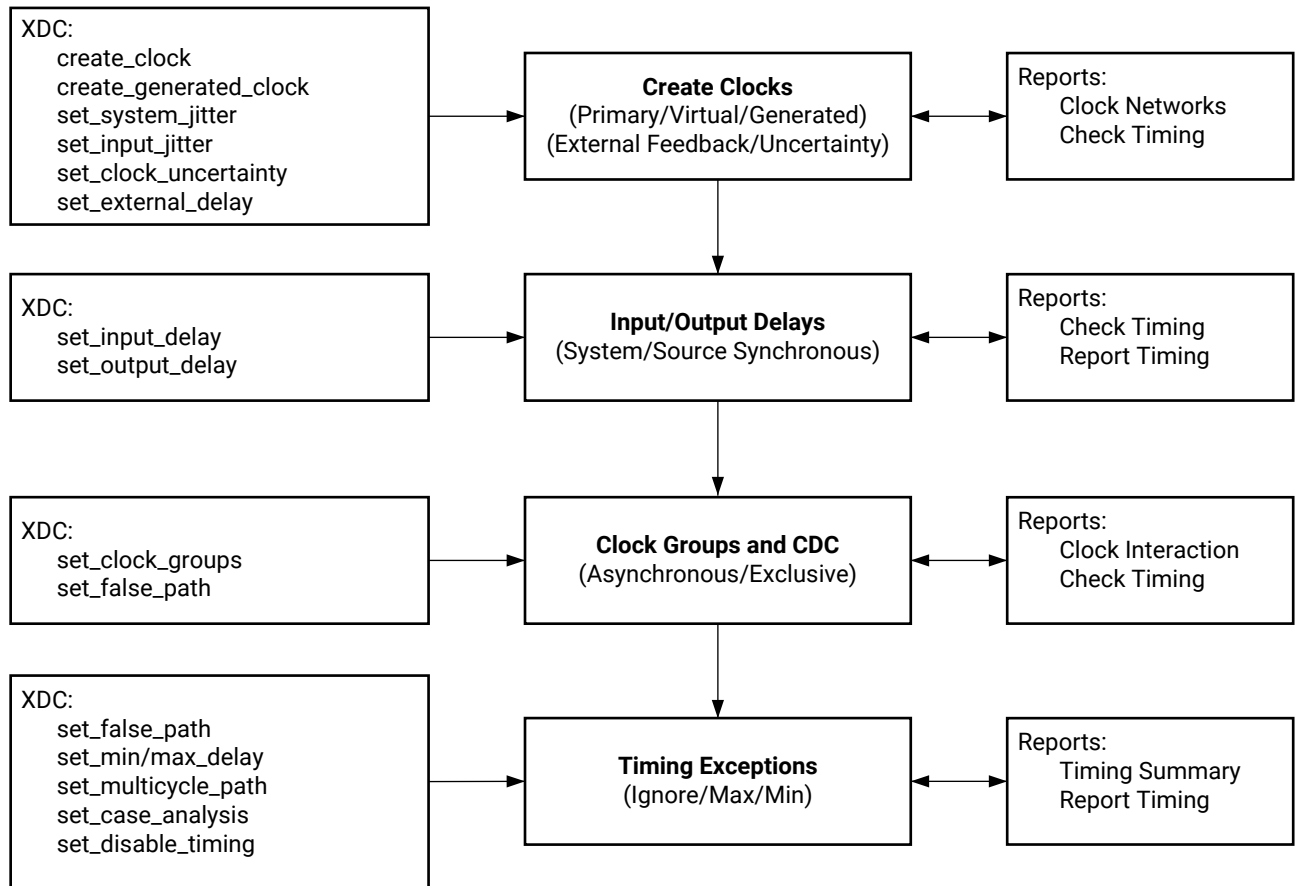
説明:

- `--vivado`: Vivado ツールの指示子を指定する v++ コマンド ライン オプションです。
- `prop`: プロパティ設定を示します。
- `run.`: `run` プロパティを示します。
- `impl_1.`: `run` の名前を示します。
- `STEP.OPT_DESIGN.TCL.PRE`: 指定する `run` プロパティです。
- `<pathToTclScript>`: プロパティ値を指定します。

タイミング制約定義の 4 つの段階

適切な制約を定義する手順には、次の図に示す 4 つの段階があります。これらの段階は、タイミング制約の優先順位および依存規則に従っており、解析を実行するためにタイミング エンジンに情報を供給する論理的方法です。

図 85: タイミング制約の開発手順



X13445-122019

- 最初の 2 つの段階は、クロック波形および I/O 遅延制約からデフォルトのタイミング パス要件を派生させるタイミング アサーションです。
- 3 つ目の段階では、少なくとも 1 つの論理パスを共有する非同期/排他的クロック ドメイン間の関係を確認します。この関係に基づいて、これらのパスのタイミング解析を無視するクロック グループまたはフォルス パス制約を入力します。
- 最後の段階はタイミング例外で、特定の制約を使用して、デフォルトのタイミング パス要件を無視、緩和、または厳しくして変更します。

制約の作成では、制約を特定し、タイミング エンジンで生成されるさまざまなレポートを使用して制約を検証します。タイミング エンジンは、合成後などの完全にマップされたネットリストに対してのみ機能します。エラーレポート済みネットリストで制約を入力することはできませんが、制約の解析およびレポートをインタラクティブに実行できるように、最初の制約セットは合成後のネットリストで作成することをお勧めします。

新規デザインのタイミング制約を作成する際、または既存の制約を完成させる際は、ザイリンクスでは Timing Constraints ウィザードを使用して、最初の 3 つの段階で不足している制約を特定することをお勧めします。Timing Constraints ウィザードは、このセクションで説明されている設計手法に従って、タイミング クロージャを達成するためにデザイン制約が安全で信頼できるものであることを確認します。Timing Constraints ウィザードの詳細は、『Vivado Design Suite ユーザー ガイド: 制約の使用』 (UG903) を参照してください。

次のセクションでは、上記の 4 つの段階の詳細を説明します。

- クロック制約の定義

- 入力ポートおよび出力ポートの制約
- クロック グループおよび CDC 制約の定義
- タイミング例外の指定

制約作成プロセスの各段階での詳細な手法およびユース ケースは、各セクションを参照してください。

クロック制約の定義

クロックは、ほかの制約でできるように、最初に定義する必要があります。タイミング制約作成フローでは、まずクロックを定義する必要のある箇所を特定し、プライマリ クロックまたは生成クロックのどちらとして定義する必要があるかを判断します。



重要: クロックに特定の名前を指定する場合 (-name オプション)、そのクロック名がほかのクロック制約または既存の自動生成クロックで使用されていないことを確認する必要があります。同じクロック名が複数のクロック制約で使用されていると、Vivado Design Suite のタイミング エンジンから最初のクロック定義が上書きされることを警告するメッセージが表示されます。同じクロック名が 2 回使用された場合、最初のクロック定義と、2 つのクロック定義の間に指定されたその名前を参照する制約は失われます。ザイリンクスでは、ほかの制約に影響を与えずにすべてのタイミング パスに制約が適用された状態を維持できる場合以外は、クロック定義が上書きされる状況を避けることをお勧めします。

クロック ソースの特定

制約が設定されていないクロック ソースは、クロック ネットワーク レポートおよびチェック タイミング レポートで特定できます。

クロック ネットワーク レポート

制約が設定されているクロックの起点と設定されていないクロックの起点は、それぞれ別のカテゴリにリストされます。制約が設定されていない起点に対しては、それがプライマリ クロックなのか生成クロックなのかを判断する必要があります。

```
% report_clock_networks
Unconstrained Clocks
Clock sysClk (endpoints: 15633 clock, 0 nonclock)
Port sysClk
Clock TXOUTCLK (endpoints: 148 clock, 0 nonclock)
GTXE2_CHANNEL/TXOUTCLK
(mgtEngine/ROCKETIO_WRAPPER_TILE_i/gt0_ROCKETIO_WRAPPER_TILE_i/gtxe2_i)
Clock Q (endpoints: 8 clock, 0 nonclock)
FDRE/Q (usbClkDiv2_reg)
```

チェック タイミング レポート

no_clock チェックでは、クロック定義のないアクティブな最下位クロック ピンのグループがレポートされます。各グループは、問題を修正するためにクロックを定義する必要のあるクロックの起点に関連付けられています。

```
% check_timing -override_defaults no_clock
1. checking no_clock
-----
There are 15633 register/latch pins with no clock driven by root clock pin: sysClk
(HIGH)
```

```
There are 148 register/latch pins with no clock driven by root clock pin:
mgtEngine/ROCKETIO_WRAPPER_TILE_i/gt0-ROCKETIO_WRAPPER_TILE_i/gtxe2_i/TXOUTCLK
(HIGH)
There are 8 register/latch pins with no clock driven by root clock pin:
usbClkDiv2-reg/C (HIGH)
```

check_timing では、クロック ツリー全体のトポロジに基づいて、同じクロック ソース ピンまたはポートが複数のグループに含まれることがあります。このような場合、推奨されるソース ピンまたはポートにクロックを作成すると、それに関連するグループすべてでクロック 定義がなかった問題が修正されます。

関連情報

デザインが適切に制約されているかを確認

プライマリ クロックの作成

プライマリ クロックは、デザインのタイミング基準を定義するクロックで、タイミング エンジンでタイミング バス要件とその他のクロックとの位相関係を算出するために使用されます。プライマリ クロックの挿入遅延は、クロックの起点(クロックが定義されるドライバー ピン/ポート) から、ファンアウトするシーケンシャル セルのクロック ピンまでで算出されます。

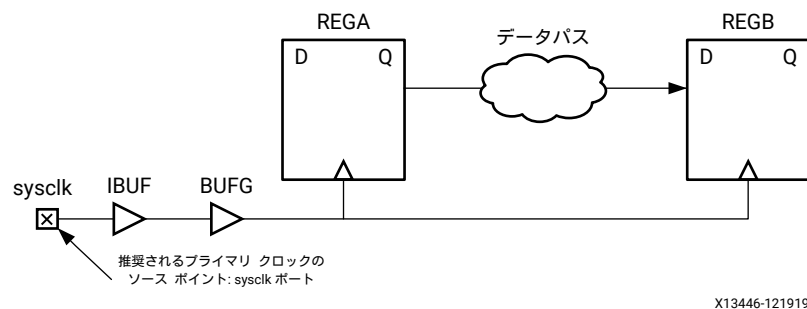
このため、遅延および間接的なスキューが正確に算出されるように、デザインの境界となるオブジェクトにプライマリ クロックを定義することが重要です。

次のセクションでは、典型的なプライマリ クロック ルートについて説明します。

入力ポート

次の図に示すように、入力ポートをプライマリ クロック ルートとして使用できます。

図 86: 入力ポートの create_clock



制約の例:

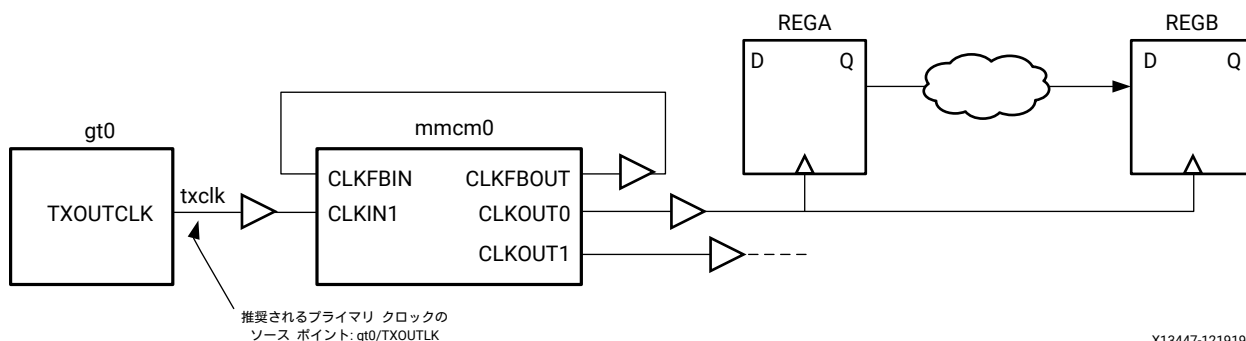
```
create_clock -name SysClk -period 10 -waveform {0 5} [get_ports sysclk]
```

この例では、波形はデューティ サイクルが 50% になるように定義されています。上記の -waveform オプションは使用法を表すために含めているだけで、50% 以外のデューティ サイクルでクロックを定義する場合にのみ必要です。差動クロック入力バッファの場合、プライマリ クロックはペアの P 側でのみ定義する必要があります。

7 シリーズ デバイスのギガビット トランシーバーの出力ピン

次の図に示すように、ギガビット トランシーバー出力ピン (リカバリ クロックなど) をプライマリ クロック ルートとして使用できます。

図 87: プリミティブ ピンの create_clock



X13447-121919

制約の例:

```
create_clock -name txclk -period 6.667 [get_pins gt0/TXOUTCLK]
```



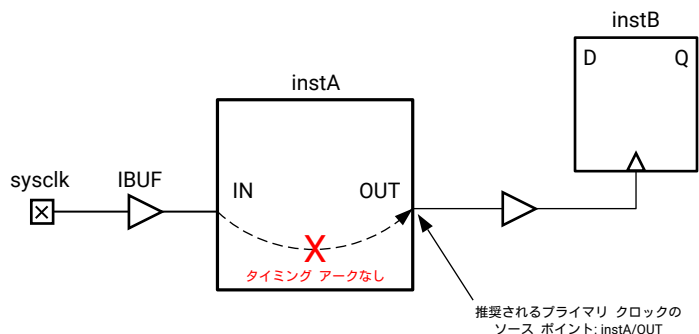
推奨: 7 シリーズ デバイスをターゲットとするデザインでは、GT 出力ピンに必要なクロックも Vivado ツールで算出され、それらのクロックがユーザー作成クロックと比較されるので、ザイリンクスでは GT 入力クロックも定義することをお勧めします。クロックが異なる場合、または GT への入力クロックがない場合、設計手法チェック警告が表示されます。

注記: UltraScale™ デバイスをターゲットとするデザインでは、関連のボード入力クロックが定義されたときに GT クロックが自動的に派生されるので、ザイリンクスでは GT の出力にプライマリ クロックを定義しないことをお勧めします。

一部のハードウェア プリミティブの出力ピン

次の図に示すような同じプリミティブの入力ピンからのタイミング アークがない出力ピンなど、一部のハードウェア プリミティブの出力ピンはプライマリ クロック ルートとして使用できます。

図 88: タイミング アークがないためにクロック パスが切断される



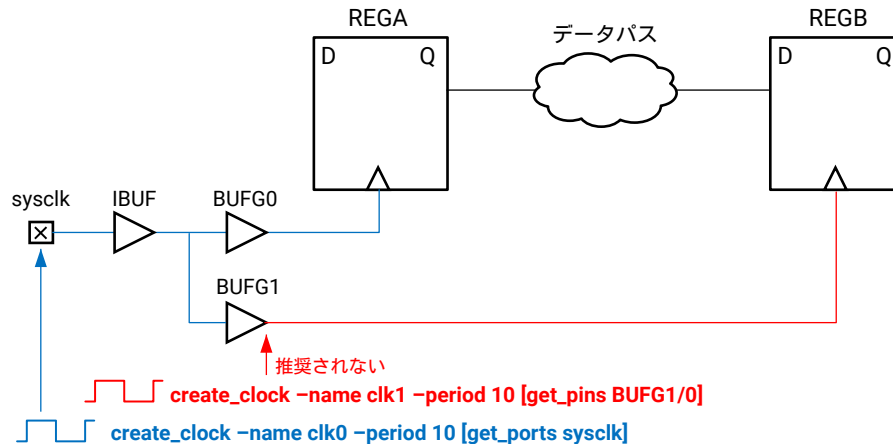
X13448-121919



重要: プライマリ クロックを別のプライマリ クロックのファンアウトで定義しないようにしてください。このような状況は、実際のハードウェアでは発生しません。また、このような状況では完全なクロック挿入遅延を算出できず、タイミング解析は正しく実行されません。この状況が発生したら、制約を見直して修正する必要があります。

次の図に、`clk1` クロックが `clk0` クロックのファンアウトで定義される例を示します。`clk1` は BUFG1 の出力に定義されており、BUFG1 の出力から `clk0` の代わりに使用されます。そのため、`clk0` と `clk1` 間のスキュー算出が無効になるので、REGA および REGB 間のタイミング解析が不正確になります。

図 89: 別のクロックのファンアウトで `create_clock` を設定 (推奨されない)



X13449-121919

生成クロックの作成

生成クロックとは、マスター クロックと呼ばれる別の既存クロックから派生したもので、通常マスター クロックに対してロジック ブロックにより実行される波形変換を記述します。生成クロックの定義はマスター クロックの特性に依存するので、マスター クロックを最初に定義する必要があります。生成クロックを明示的に定義するには、`create_generated_clock` コマンドを使用する必要があります。

自動派生クロック

Vivado タイミング エンジンでは、クロック調整ブロック (CMB) とそれによりマスター クロックに対して実行される変換が認識されるので、生成クロックのほとんどは自動的に生成されます。

ザイリックス 7 シリーズ デバイス ファミリの CMB は次のとおりです。

- MMCM*/PLL*
- BUFR
- PHASER*

ザイリックス UltraScale デバイス ファミリの CMB は次のとおりです。

- MMCM*/PLL*
- BUFG_GT/BUFGCE_DIV
- GT*_COMMON/GT*_CHANNEL/IBUFDS_GTE3
- BITSlice_CONTROL/RX*_BITSlice

- ISERDESE3

クロック ツリーにあるその他の組み合わせセルでは、波形がセルで変換されなければ、タイミング クロックがそれらを介して伝搬されるので、出力で定義し直す必要はありません。この自動派生機能は、実際のハードウェア動作に一致する生成クロックを定義する最も安全な方法なので、できるだけ利用してください。

Vivado Design Suite タイミング エンジンで選択された自動派生クロックの名前が不適切な場合は、`create_generated_clock` コマンドを使用すると、波形を変換せずに名前を指定できます。この制約は、制約ファイルのマスター クロックを定義している制約の直後に記述する必要があります。たとえば、MMCM インスタンスで生成されたクロックのデフォルト名が `net0` の場合、次の制約を追加して別の名前 (この例の場合は `fftClk`) を指定できます。

```
create_generated_clock -name fftClk [get_pins mmcm_i/CLKOUT0]
```

あいまいさを避けるため、制約はクロックのソース ピンに設定する必要があります。詳細は、『Vivado Design Suite ユーザー ガイド: 制約の使用』 ([UG903](#)) を参照してください。

ユーザー定義の生成クロック

プライマリ クロックをすべて定義したら、クロック ネットワークまたはチェック タイミング (`no_clock`) レポートを使用し、タイミング クロックを含まないクロック ツリー部分を見つけて、生成クロックを定義できます。

マスター クロックのロジック コーンで実行される変換は理解しにくいこともあります。この場合、最も控えめな制約を使用する必要があります。たとえば、ソース ピンがシーケンシャル セルの出力で、マスター クロックが少なくとも 2 で分周される場合、適切な制約は次のようになります。

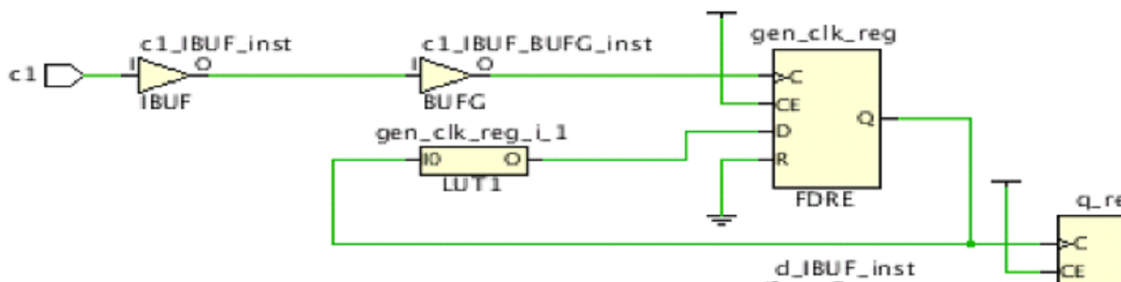
```
create_generated_clock -name clkDiv2 -divide_by 2 \
-source [get_pins fd/C] [get_pins fd/Q]
```

デザインにラッチが含まれる場合は、タイミング クロックがラッチ ゲート ピンにも到達する必要があります。制約が存在しない場合はチェック タイミング レポート (`no_clock`) に表示されます。上記の例に従って、これらのクロックを定義します。

マスター クロックと生成クロック間のパス

プライマリ クロックとは異なり、生成クロックはマスター クロックのファンアウトに定義して、タイミング エンジンでその挿入遅延が正確に算出されるようにする必要があります。この規則に従っていないと、タイミング解析が不正確になり、スラックの算出が無効となる可能性が高くなります。たとえば次の図では、`gen_clk_reg/Q` は次のフリップフロップ (`q_reg`) のクロックとして使用され、プライマリ クロック `c1` のファンアウト コーンに含まれます。このため、`gen_clk_reg/Q` には、`create_generated_clock` ではなく、`create_clock` を使用する必要があります。

図 90: マスター クロックのファンアウトの生成クロック



```
create_generated_clock -name GC1 -source [get_pins gen_clk_reg/C] -divide_by 2
[get_pins gen_clk_reg/Q]
```

クロック定義および範囲の検証

すべてのデザイン クロックを定義し、インメモリ デザインに適用したら、report_clocks コマンドを使用して各クロックの波形、マスター クロックと生成クロックの関係を検証できます。

```
Clock Period Waveform Attributes Sources
sysClk 10.00000 {0.00000 5.00000} P {sysClk}
clkfbout 10.00000 {0.00000 5.00000} P,G {clkgen/mmcm_adv_inst/CLKFBOUT}
cpuClk 20.00000 {0.00000 10.00000} P,G {clkgen/mmcm_adv_inst/CLKOUT0}
...
=====
Generated Clocks
=====
Generated Clock : cpuClk
Master Source : clkgen/mmcm_adv_inst/CLKIN1
Master Clock : sysClk
Edges : {1 2 3}
Edge Shifts : {0.000 5.000 10.000}
Generated Sources : {clkgen/mmcm_adv_inst/CLKOUT0}
```

また、すべての内部タイミング パスに少なくとも 1 つのクロックが適用されているかどうかを検証できます。タイミング チェック レポートでは、次の 2 種類のチェックが提供されています。

- no_clock: 定義したクロックが到達しないすべてのアクティブ クロック ピンをレポートします。
- unconstrained_internal_endpoint: クロックに対するタイミング チェックがあるのに、クロックが定義されていないシーケンシャル セルのすべてのデータ入力ピンをレポートします。

両方のチェックで 0 が返された場合、タイミング解析の適用範囲が高いことを示します。

または、XDC およびタイミング設計手法チェックを実行して、制約の競合または不正確なタイミング解析が実行される状況が発生せずに、推奨されるネットリスト オブジェクトにクロックが定義されていることを確認できます。

次のコマンドを使用して、これらのチェックを実行します。

```
report_methodology -checks [get_methodology_checks {TIMING-* XDC*}]
```


関連情報

設計手法レポートの生成

クロック特性の調整

クロックとその波形を定義したら、次はノイズおよびばらつきに関する情報を入力します。XDC 言語では、ジッターに関するばらつきと位相エラーの記述がスキューと遅延の記述とは別になっています。

ジッター

ジッターには、Vivado Design Suite で使用されるデフォルト値を使用するのが最適です。デフォルト値は次のように変更できます。

- デバイスに供給されるプライマリ クロックに 0 を超えるランダムなジッターが含まれる場合、`set_input_jitter` コマンドを使用してピーク トゥ ピーク ジッター値をナノ秒 (ns) で指定します。
- デバイスの電源にノイズがある場合にグローバル ジッターを調節する場合は、`set_system_jitter` コマンドを使用します。ザイリンクスでは、デフォルト システム ジッター値を増加することはお勧めしません。

生成クロックの場合、ジッターはマスター クロックとそのクロック調整ブロックの特性から算出されます。これらの値を変更する必要はありません。

その他のばらつき

1 つのクロックまたは 2 つのクロック間のタイミング パスにマージンを追加する必要がある場合は、`set_clock_uncertainty` コマンドを使用します。これは、実際のクロック エッジおよび全体的なクロック関係を変更せずに、デザインの一部の制約を厳しくする場合に最適で最も安全な方法でもあります。ユーザーが定義したクロックのばらつきは、Vivado ツールで算出されたジッターに追加され、セットアップおよびホールド解析用にも個別に指定できます。

たとえば、デザインをセットアップおよびホールド両方のノイズに耐性の高いものにするには、次のようにデザイン クロック `clk0` のすべてのクロック パス内のマージンを 500 ps 分縮める必要があります。

```
set_clock_uncertainty -from clk0 -to clk0 0.500
```

注記: デザインのホールド マージンを縮めると、専用サイト内およびカスケード パスで、配線によりサイト内ネットを迂回することにより修正できないホールド違反が発生する可能性があります。

2 つのクロック間のばらつきを追加で指定する場合、制約は両方向に適用する必要があります (データフローが双方向である場合)。次に、`clk0` と `clk1` 間のセットアップのみでばらつきを 250 ps 増加する例を示します。

```
set_clock_uncertainty -from clk0 -to clk1 0.250 -setup  
set_clock_uncertainty -from clk1 -to clk0 0.250 -setup
```

ソースのクロック レイテンシ

`set_clock_latency` コマンドに `-source` オプションを指定すると、クロック レイテンシをそのソースで指定できます。これは、次のような場合に便利です。

- 入力および出力遅延制約とは別に、デバイスの外部でクロック遅延の伝搬を指定するため。

- アウト オブ コンテキスト コンパイル中にブロックで使用されるクロックの内部伝搬レイテンシを記述するため。このようなコンパイル フローでは、完全なクロック ツリーは必要ではないので、ブロック外部の最小および最大動作条件間の変動は自動的に算出されず、手動で記述する必要があります。

有効なレイテンシ値を指定するのは困難なので、この制約はアドバンス ユーザー以外は使用しないでください。

MMCM または PLL の外部フィードバック ループ遅延

内部クロック遅延ではなくボード遅延を補正するために MMCM または PLL フィードバック ループを接続する場合、`set_external_delay` コマンドを使用してデバイス外のベスト ケースとワースト ケースの遅延を指定する必要があります。この遅延を指定しないと、MMCM または PLL に関する I/O タイミング解析が不適切なものとなり、タイミング クロージャが不可能になる可能性があります。また、外部補正を使用している場合は、通常どおりにボードでのクロック トレース遅延を考慮するだけでなく、入力および出力の遅延制約値を適切に変更する必要があります。

入力ポートおよび出力ポートの制約

デザインの各ポートのロケーションおよび I/O 規格を指定するだけでなく、入力および出力遅延制約を指定して、デバイスのインターフェイスに入出力される外部パスのタイミングを記述する必要があります。これらの遅延は、通常ボードで生成されてデバイスに入力されるクロックに対して定義されています。I/O パスがボード クロックとは異なる波形のクロックと関連している場合、遅延を仮想クロックに対して定義する必要があることがあります。

システム レベルの視点

I/O パスは、Vivado Design Suite タイミング エンジンでレジスタ間のパスと同様にモデリングされますが、デバイス外部にあるパス遅延のモデリングには制約を定義する必要があります。内部パスを解析する場合は、セットアップおよびホールド解析の両方に対して最小遅延と最大遅延が考慮されます。これは、I/O パスでも同様です。このため、最小遅延と最大遅延の条件を記述しておくことが重要です。I/O タイミング パスは、デフォルトでシングルサイクルパスとして解析されます。これは、次を意味します。

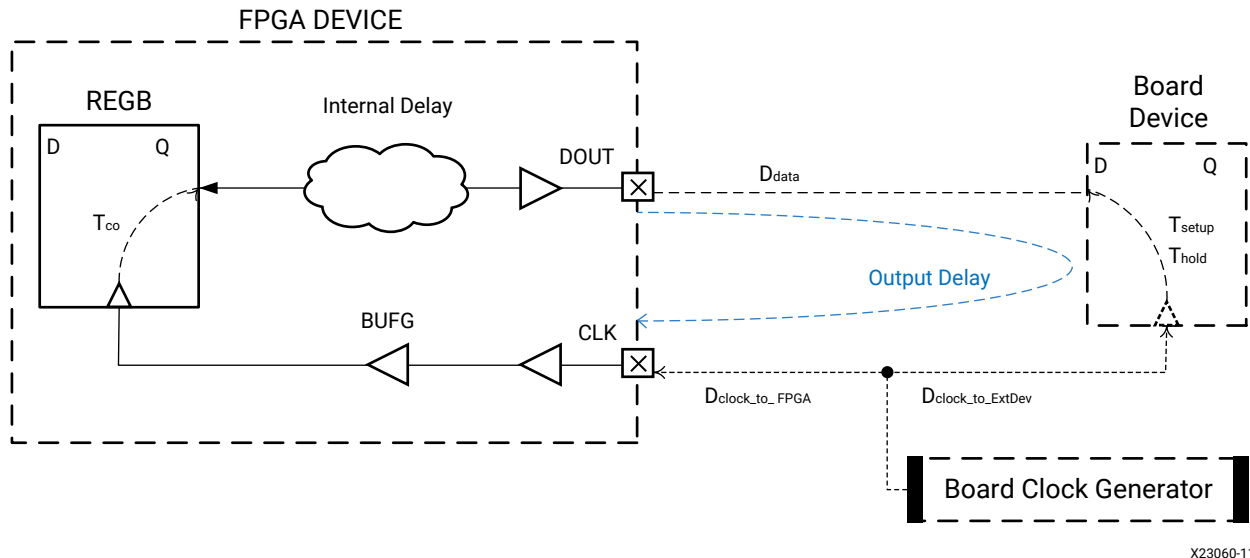
- 最大遅延解析 (セットアップ) では、シングル データ レート インターフェイスではソース エッジの 1 クロック サイクル後にデータがキャプチャされ、ダブル データ レート インターフェイスではソース エッジの半クロック サイクル後にデータがキャプチャされます。
- 最小遅延解析 (ホールド) の場合、データは同じクロック エッジで開始されてキャプチャされます。

ソース同期インターフェイスなど、クロックと I/O データ間の関係に異なるタイミングを設定する必要がある場合は、異なる I/O 遅延を指定し、タイミング例外を設定します。これは、高度な I/O タイミング制約になります。

入力遅延の定義

入力遅延はデバイスのインターフェイスのクロックに対して定義されます。基準クロックのソース ピンに `set_clock_latency` が指定されていない場合、入力遅延はソース エッジからクロック トレースを介して外部デバイスおよびデータ トレースに到達するまでの絶対時間に相当します。クロック レイテンシが既に別に指定されている場合、クロック トレース遅延は無視できます。

図 93: 出力遅延の算出



X23060-111419

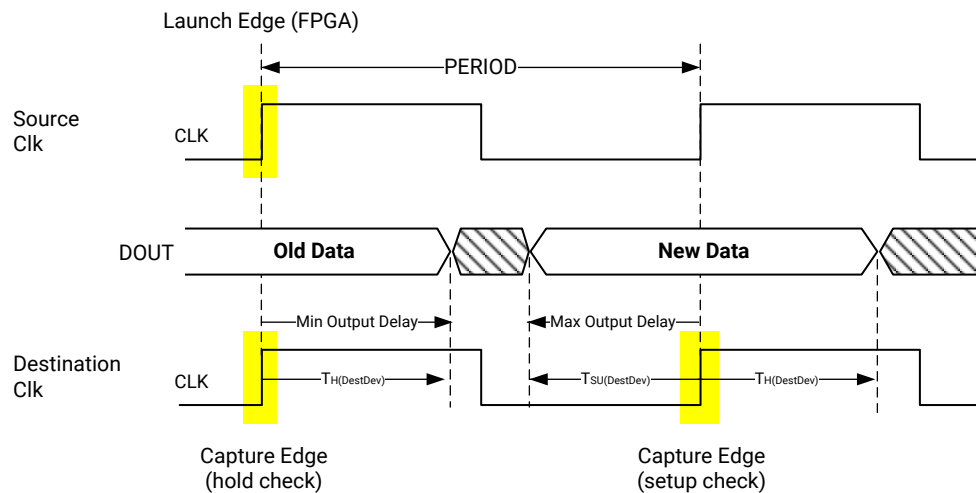
両方の解析タイプの出力遅延値は、次のように算出されます。

```
Output Delay(max) = Tsetup + Ddata(max) + Dclock_to_FPGA(max) - Dclock_to_ExtDev(min)
Output Delay(min) = Ddata(min) - Thold + Dclock_to_FPGA(min) - Dclock_to_ExtDev(max)
```

次の図に、sysClk クロックが既に CLK ポートに定義されている場合に、セットアップ (最大) およびホールド (最小) 解析の出力遅延制約を設定する単純な例を示します。

```
set_output_delay -max -clock sysClk 2.4 [get_ports DOUT]
set_output_delay -min -clock sysClk -1.1 [get_ports DOUT]
```

図 94: 最小および最大出力遅延



X13453-121919

出力遅延はデスティネーション クロック エッジ前のボードの遅延に相当します。クロックとデータのボード トレースのバランスが取られた標準的なシステム同期インターフェイスの場合、デスティネーション デバイスのセットアップ時間により、最大解析の出力遅延値が定義されます。また、デスティネーション デバイスのホールド タイムにより、最小解析の出力遅延が定義されます。指定された最小出力遅延は、信号がデザインから出力されてから、デスティネーション デバイス インターフェイスでホールド 解析に使用されるまでの最小遅延を示します。このため、クロック内の遅延はそれだけ小さくなります。最小出力遅延が正の値の場合は、信号にデザイン内で負の遅延が含まれることを意味します。最小出力遅延が負の値であることがよくあるのはこのためです。たとえば次のコード例は、ホールド タイム要件を満たすためには、DOUT までのデザイン内の遅延が少なくとも +0.5 ns である必要があることを示しています。

```
set_output_delay -min -0.5 -clock CLK [get_ports DOUT]
```

基準クロックの選択

入力または出力ポートに関するシーケンシャル セルを制御するクロック ツリートポロジによって、入力または出力遅延制約を定義するのに最適なクロックを選択する必要があります。I/O パス レジスタのクロックが生成クロックである場合は通常、遅延制約を生成クロックのアップストリームで定義されているプライマリ クロックを基準に定義する必要があります。このセクションでは、この規則の例外を説明します。

各ポートに関連するクロックの特定

I/O 遅延制約を定義する前に、各ポートにどのクロックが関連付けられているかを特定する必要があります。クロックを特定するには、次のセクションで説明する方法を使用できます。

ボード回路図の確認

ボード上の別のデバイス インターフェイスに接続されているポートのグループに対しては、ザイリンクス デバイスと外部デバイス インターフェイスの両方に接続されているボード クロックを入力または出力遅延制約の基準クロックとして使用できます。関連するポートのグループのタイミングを制御するには、ボード クロックが I/O ポートのタイミングのために内部で変換され、デザインでザイリンクス デバイス内の同じクロックが生成されるようになっていることを外部デバイスのデータシートで確認する必要があります。

デザイン回路図の確認

各ポートで、パスの回路図をシーケンシャル セルの最初のレベルまで展開し、これらのセルのクロック ピンからクロック ソースまでたどります。この方法は、ファンアウトの大きいネットに接続されたポートでは困難です。

ポートの入出力のタイミングのレポート

`report_timing` コマンドを使用すると、ポートに既に制約が設定されているかどうかに関係なく、デザイン内の関連するクロックを特定できます。すべてのタイミング クロックを定義したら、I/O ポートに入出力されるワースト パスをレポートし、レポートされたクロックに対して I/O 遅延制約を作成して、デザインのほかのクロックに対して同じタイミング レポートを作成します。ポートが複数のクロックに関連付けられている場合は、対応する制約を作成してプロセスを繰り返します。

たとえば、`din` 入力ポートがデザイン内の `clk1` および `clk2` クロックに関連しているとします。

```
report_timing -from [get_ports din] -sort_by group
```

レポートには、din ポートが clk1 に関連していることが示されます。入力遅延制約 (この例では最小遅延および最大遅延の両方) は、次のように設定します。

```
set_input_delay -clock clk1 5 [get_ports din]
```

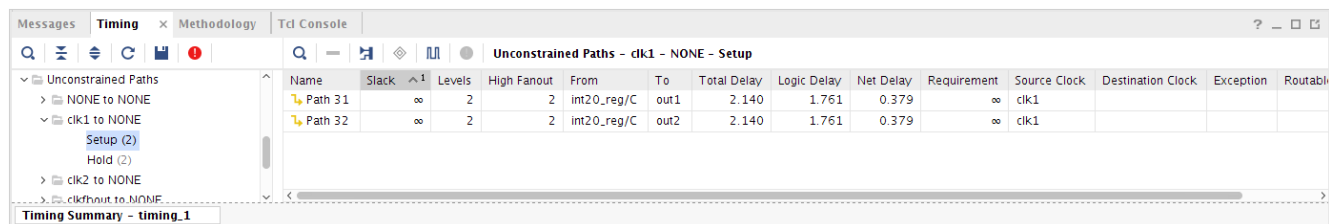
前と同じコマンドでタイミング解析を再び実行して、din が clk2 にも関連していることを確認します。-sort_by group オプションを使用すると、終点クロックごとに N 個のパスがレポートされます。対応する遅延制約を追加し、レポート コマンドを再実行して、din ポートが別のクロックに関連していないことを確認します。

タイミング サマリ レポートで -report_unconstrained オプションを使用しても同じ解析を実行できます。デザインにクロック制約しか含まれない場合、[Unconstrained Paths] セクションは次のように表示されます。

```
-----
| Unconstrained Path Table
-----
Path Group      From Clock    To Clock
-----
(none)
(none)          clk1
(none)          clk2
(none)          clk1
(none)          clk2
-----
```

クロック名のないフィールド (Vivado IDE では <NONE>) は、始点 (From Clock) または終点 (To Clock) がクロックに関連付けられていないパス グループを意味します。制約が設定されていない I/O ポートは、このカテゴリになります。残りのレポートを確認すると、これらの名前を取得できます。たとえば、Vivado IDE で [clk1 to NONE] カテゴリのセットアップ パスを選択すると、[To] 列に clk1 で駆動されるポートが表示されます。

図 95: 制約の付いていない出力ポートのリスト



Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Routable
Path 31	∞	2	2	int20_reg/C	out1	2.140	1.761	0.379	∞	clk1			
Path 32	∞	2	2	int20_reg/C	out2	2.140	1.761	0.379	∞	clk1			

新しい制約を追加してメモリでそれらを適用したら、レポートを再作成して、制約が設定されていないポートがまだあるかどうかを確認します。ほとんどのデザインで、レポートされるパスの数を増加して、すべての I/O パスがレポートに含まれるようにする必要があります。

自動的に認識されたサンプリング クロックの使用

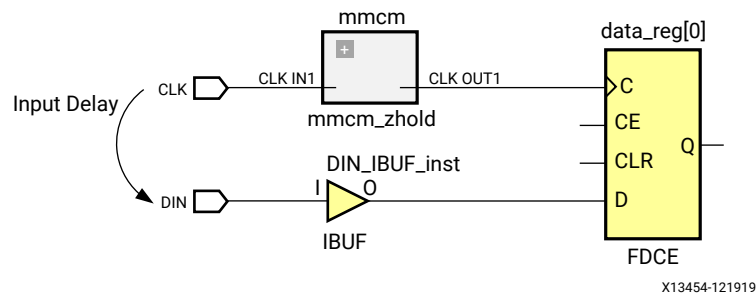
set_input_delay および set_output_delay 制約は、関連するクロックを指定せずに使用できます。Vivado Design Suite タイミング エンジンは、デザインを解析し、各ポートをすべてのサンプリング クロックに自動的に関連付けます。I/O パスのタイミング レポートを生成すると、各 I/O ポートがどのように制約されたかを確認できます。これによりデザインをすばやく制約できますが、このような汎用制約では一般的すぎて実際のハードウェアに適さない場合は、問題となることがあります。

プライマリ クロックの使用

プライマリ クロック (入力されるボード クロック) は、クロック調整ブロックを介さず、I/O パスのシーケンシャル セルを直接制御する際に使用する必要があります。I/O 遅延ラインは、クロック挿入遅延にのみ影響し、波形には影響しないので、クロック調整ブロックとしては考慮されません。これは、[入力遅延の定義](#) および [出力遅延の定義](#) の 2 つの例に示されています。ほとんどの場合、外部デバイスにも同じボード クロックに対して定義される独自のインターフェイス特性があります。

プライマリ クロックが、ゼロ ホールド違反モード (ZHOLD) を使用してデバイス内の PLL または MMCM で補正されると、I/O パスのシーケンシャル セルがそのプライマリ クロックの内部コピー (たとえば生成クロック) に接続されます。両方のクロックの波形は同じなので、ザイリンクスでは入力/出力遅延制約の基準クロックとしてプライマリ クロックを使用することをお勧めします。

図 96: クロック パスに ZHOLD MMCM が含まれる場合の入力遅延



ZHOLD MMCM は補正量に相当する負の挿入遅延を含むクロック バッファのように動作するので、制約は [入力遅延の定義](#) の例と同じになります。

仮想クロックの使用

ボード クロックが挿入遅延を補正するだけでなく実際に波形を変換するクロック調整ブロックを通過する場合、入力または出力遅延の基準クロックとしてボード クロックの代わりに仮想クロックを使用することをお勧めします。仮想クロックを使用するには、主に次の 3 つの場合があります。

- 内部クロックおよびボード クロックの周期が異なる: 仮想クロックを内部クロックの周期および波形と同じになるよう定義する必要があります。これにより、I/O パスのシングルサイクル パス要件が標準的なものになります。
- 入力パスで、内部クロックの波形がボード クロックから正の方向にシフトされている: 仮想クロックはボード クロックと同様に定義し、仮想クロックから内部クロックにセットアップ対して 2 サイクルのマルチサイクル パス制約を定義します。これらの制約により、セットアップ タイミング解析が 1 クロック サイクル + 位相シフト量の要件で実行されます。
- 出力パスで、内部クロックの波形がボード クロックから負の方向にシフトされている: 仮想クロックはボード クロックと同様に定義し、内部クロックから仮想クロックにセットアップ対して 2 サイクルのマルチサイクル パス制約を定義します。これらの制約により、セットアップ タイミング解析が 1 クロック サイクル + 位相シフト量の要件で実行されます。

仮想クロックを使用すると、デフォルトのタイミング解析を調整して、I/O パスが要件の厳しいクロック乗せ換えパスとして処理されるのを回避できます。



重要: 位相シフトされたクロックを含む I/O パスにマルチサイクル パスを使用する必要があるのは、位相シフトによりクロック波形が変更される場合のみです。クロック調整ブロックの挿入遅延に位相シフトが追加され、クロック波形が保持される場合は、マルチサイクル パスは必要ありません。詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906) のこのセクションを参照してください。

たとえば、100 MHz で動作する sysClk ボード クロックを MMCM で逡倍し、266 MHz で動作する clk266 を生成するとします。clk266 で生成される出力では、clk266 を基準クロックとして使用する必要があります。set_output_delay で sysClk を基準クロックとして使用しようとする、非同期クロックとして認識され、パスに単一サイクルのタイミング制約が適用されなくなります。

生成クロックの使用

出力ソース同期インターフェイスの場合、デザインで内部クロックのコピーが生成されて、データと共にボードに転送されます。このクロックは、転送されたクロックとデータ間の位相関係 (スキュー) を制御してレポートする場合は、通常出力データ遅延制約の基準クロックとして使用されます。システム同期インターフェイスでは、入力および出力遅延制約にフォワード クロックを使用することもできます。

基準クロックの立ち上がり/立ち下がりエッジ

I/O 制約で使用するクロック エッジは、デバイスに接続された外部デバイスのデータシートに従う必要があります。デフォルトでは、set_input_delay および set_output_delay コマンドで基準クロックの立ち上がりエッジに対する遅延制約が定義されます。立ち下がりエッジに対する遅延を指定する場合は、clock_fall オプションを使用する必要があります。add_delay オプションでポートに 2 つ目の制約を指定して、立ち上がりクロック エッジと立ち下がりクロック エッジに異なる遅延制約を指定することもできます。

ほとんどの場合、I/O 基準クロックのエッジはデバイス内部の I/O データをラッチまたは送信するために使用されるクロック エッジです。I/O タイミング パスを解析すると、どちらのクロック エッジが使用されるか確認でき、それらが実際のハードウェア動作でどのようになるか検証できます。クロックの立ち上がりエッジが I/O パスの基準クロック (内部ではクロックの立ち下がりエッジにのみ関連) として間違えて使用されると、パス要件は 1/2 の周期になり、タイミング クロージャを達成しにくくなります。

遅延制約の確認

I/O タイミング制約を入力したら、I/O パスのタイミングの解析方法と、セットアップおよびホールド チェックの両方でのスラック違反の量を確認することが重要になります。すべてのポートに関するタイミング レポートを使用してセットアップおよびホールド解析の両方を実行すると (delay type = min_max)、次を確認できます。

- 正しいクロックとクロック エッジが遅延制約の基準として使用されているかどうか。
- デバイス内部の I/O データが予測どおりクロックで送信およびキャプチャされているかどうか。
- 違反が配置の変更または適切な遅延ラインのタップ設定で合理的に修正可能かどうか。修正できない場合は、制約に入力された I/O 遅延値を確認し、それらが現実的かどうか判断し、タイミングを満たすためにデザインを修正する必要があるかどうかを確認する必要があります。

I/O パス レポートのコマンド ラインの例

```
report_timing -from [all_inputs] -nworst 1000 -sort_by group \
-delay_type min_max
```

```
report_timing -to [all_outputs] -nworst 1000 -sort_by group \
-delay_type min_max
```

I/O 遅延制約に問題があると、タイミング クロージャを達成できないことがあります。インプリメンテーション ツールはタイミング ドリブンで、タイミングを満たすために配置配線を最適化します。I/O パス要件を満たすことができず、I/O パスがワースト違反になる場合、デザイン全体の QoR (結果の品質) に影響します。

入力から出力までのフィードスルー パス

入力ポートから出力ポートまでの組み合わせパスを制約するには、複数の方法があります。

例 1

フィードスルー パスのターゲット最大遅延以上の周期の仮想クロックを使用し、入力および出力遅延制約を次のように適用します。

```
create_clock -name vclk -period 10
set_input_delay -clock vclk <input_delay_val> [get_ports din] -max
set_output_delay -clock vclk <output_delay_val> [get_ports dout] -max
```

説明:

```
input_delay_val(max) + feedthrough path delay (max) + output_delay_val(max)
<= vclk period.
```

この例では、最大遅延のみが制約されています。

例 2

フィードスルー ポート間に最小および最大遅延制約を組み合わせて使用します。例:

```
set_max_delay -from [get_ports din] -to [get_ports dout] 10
set_min_delay -from [get_ports din] -to [get_ports dout] 2
```

これは、パスに最小遅延と最大遅延の両方の遅延を設定するシンプルな方法で、タイミング解析では、同じポートの既存の入力および出力遅延制約も使用されます。このため、このスタイルはそれほど使用されません。

最大遅延は通常最適化され、スロー タイミング コーナーに対してレポートされますが、最小遅延はファースト タイミング コーナーに対してレポートされます。特にポートが互いに遠くに配置されている場合は、フィードスルー パスの遅延制約を 2、3 回試して、それらが妥当で、インプリメンテーション ツールで満たすことができるかを検証してください。

XDC テンプレートの使用: ソース同期インターフェイス

ザイリンクスでは、ソース同期インターフェイスに I/O 制約テンプレートを使用することをお勧めします。ソース同期制約は、複数の方法で記述できます。Vivado Design Suite に含まれるテンプレートは、デフォルトのタイミング パス要件に基づいています。構文はシンプルですが、セットアップ解析が同じエッジ (0 サイクル) ではなく異なるソース エッジおよびデスティネーション エッジ (1 サイクルまたは 1/2 サイクル) で実行されることを考慮して、遅延値を調整する必要があります。タイミング レポートでは、クロック エッジが直接ハードウェアの実際のクロック エッジに対応しないので、解読しにくい場合があります。Vivado IDE でこれらのテンプレートを開くには、[Tools] → [Language Templates] → [XDC] → [Timing Constraints] → [Input Delay Constraints] → [Source Synchronous] をクリックします。

クロック グループおよび CDC 制約の定義

Vivado IDE では、デフォルトでデザインのすべてのクロック間のパスでタイミング解析が実行されます。次の制約を使用すると、このデフォルト動作を変更できます。

- `set_clock_groups`: 指定したクロック グループ間のタイミング解析をディスエーブルにします。同じグループ内のクロック間のタイミング解析はディスエーブルになりません。
- `set_false_path`: `-from` および `-to` オプションで指定された方向のみのクロック間のタイミングをディスエーブルにします。

場合によっては、クロック乗せ換え (CDC) の 1 つまたは複数のパスに次の制約を使用して、レイテンシまたはバス スキューを制限すると有益なこともあります。

- `set_max_delay -datapath_only`: 非同期 CDC パスに最大遅延制約を設定し、レイテンシを制限します。

注記: クロック グループまたはフォルス パス制約が既にクロック間または同じ CDC パスに設定されている場合、最大遅延制約は無視されます。そのため、1 つの CDC タイミング制約を選択する前にすべてのクロック ペア間のパスを注意して確認し、制約が競合しないようにすることが重要です。



推奨: ザイリンクスでは、`report_methodology` を実行して、`set_max_delay -datapath_only` 制約が `set_clock_groups` または `set_false_path` 制約により無効になっていないかどうかを確認することをお勧めします。

- `set_bus_skew`: 非同期 CDC パス間の信号のセットを、レイテンシではなくバス スキューで制約します。



ヒント: Vivado IDE からバス スキュー制約を設定することもできます。[Timing Constraints] ウィンドウで [Assertions] を展開し、[Set Bus Skew] をダブルクリックします。

関連情報

[設計手法レポートの生成](#)

クロックの関連性の確認

クロック間に論理パスがある場合、タイミング解析が実行されます。クロック関係には、同期、非同期、排他があります。

同期

2 つのクロックに固定の位相関係がある場合、クロック関係は同期になります。これは、2 つのクロックが次を共有している場合です。

- 共通の回路 (共通のノード)
- プライマリ クロック (同じ初期位相)

非同期

2 つのクロックに固定の位相関係がない場合、クロック関係は非同期になります。これは、2 つのクロックに次のいずれかの状況がある場合に発生します。

- デザイン内で共有される共通の回路がなく、共通のプライマリ クロックもない。
- 1000 サイクル以内に共通周期を見つけることができず、タイミング エンジンで適切にタイミング解析を実行できない。
- 共通クロックはあるが共通ノードがない。
- クロックの自動派生プロセスにより、既知の位相関係が確実にないトポロジの一部である。

2 つのクロックが同期していて共通周期が非常に小さい場合、セットアップ パス要件がタイミングを満たすには厳しすぎるものになります。この場合、ザイリンクスでは 2 つのクロックを非同期として処理し、安全な非同期 CDC 回路をインプリメントすることをお勧めします。

排他的

同じクロック ツリーを伝搬し、同じシーケンシャル セルのクロック ピンに到達するが、物理的に同時にアクティブにならない場合、クロック関係は排他的になります。

クロック ペアの分類

クロック ペアは、クロック関連性レポートおよびチェック タイミング レポートを使用して分類できます。

クロック関連性レポート

クロック関連性レポートは、2 つのクロック間でどのようにタイミング解析が実行されるかを示します。

- 2 つのクロックに共通のプライマリ クロックがあるか。クロックが正しく定義されている場合、デザインの同じソースからのクロックはすべて同じプライマリ クロックを共有します。
- 2 つのクロックに共通周期があるか。タイミング エンジンでワースト ケースのセットアップまたはホールド 関係を決定できない場合、これがセットアップまたはホールド パス要件の列に表示されます。
- 2 つのクロック間の一部またはすべてのパスにクロック グループまたはタイミング例外制約が適用されているか。
- 2 つのクロック間のセットアップ パス要件が非常に厳しいか。この状況は、2 つのクロックが同期しているのに、丸めなどのためにそれらの周期が正確な通倍として指定されない場合に発生することがあります。クロック サイクルが繰り返されるにつれてエッジがずれていき、ワースト ケース要件が非常に厳しくなることがあります。

チェック タイミング レポート

チェック タイミング (Check Timing) レポート (`multiple_clock`) では、複数のクロックが到達するクロック ピンで、これらのクロック間に `set_clock_groups` または `set_false_path` 制約が設定されていないものが特定されます。

排他的なクロック グループの制約

通常のタイミングまたはクロック ネットワークのレポートを使用すると、クロック パスを確認でき、2 つのクロックが同じクロック ツリーに伝搬され、始点と終点のクロック ピンが同じクロック ツリーに接続されるタイミング パスで同時に使用されている状況を特定できます。この解析には時間がかかることがあるので、代わりにチェック タイミング レポートの `multiple_clock` セクションを確認します。このセクションには、クロック ピンとそれに関連したタイミング クロックがリストされます。

次のセクションで説明するように、クロック ツリー トポロジに基づいて、異なる制約を適用する必要があります。

同じクロック ソースで定義されたクロックの重複

これは、2つのクロックが `create_clock -add` コマンドを使用して同じネットリスト オブジェクトに定義され、アプリケーションの複数のモードを示す場合に発生します。この場合、クロック間にクロック グループ制約を適用するのが安全です。次に例を示します。

```
create_clock -name clk_mode0 -period 10 [get_ports clk_in]
create_clock -name clk_mode1 -period 13.334 -add [get_ports clk_in]
set_clock_groups -physically_exclusive -group clk_mode0 -group clk_mode1
```

clk_mode0 および clk_mode1 クロックがほかのクロックを生成する場合、同じ制約をその生成クロックにも適用する必要があります。これには、次を実行します。

```
set_clock_groups -physically-exclusive \
-group [get_clocks -include_generated_clock clk_mode0] \
-group [get_clocks -include_generated_clock clk_mode1]
```

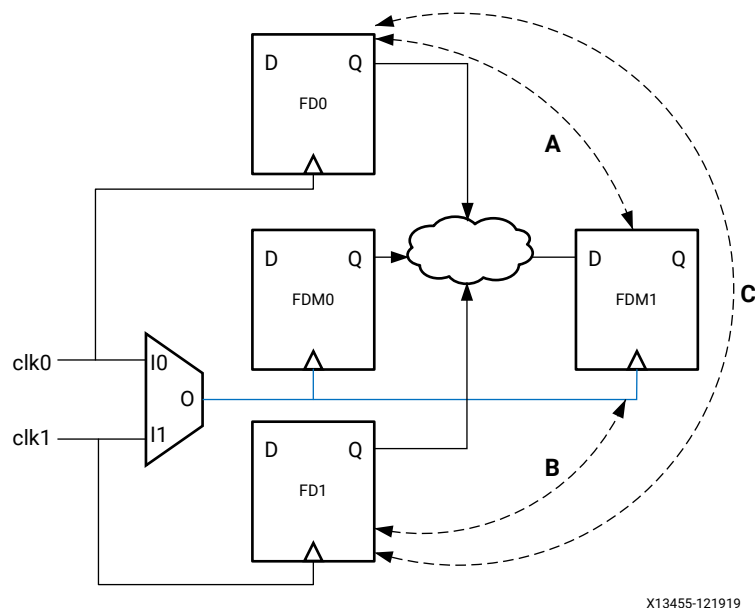
クロック マルチプレクサーで駆動されるクロックの重複

2つ以上のクロックがマルチプレクサー (または組み合わせセル) に入力されると、すべてが伝搬され、セルのファンアウトで重複します。現実的には、一度に伝搬できるのは1つのクロックのみですが、タイミング解析では同時に複数のタイミング モードがレポートされます。

このため、CDC (クロック乗せ換え) パスを確認し、新しい制約を追加して、一部のクロック関係を無視する必要があります。正しい制約は、クロックがデザインのどこでどのように動作するかによって異なります。

次の図の例に、マルチプレクサーを駆動する 2 つのクロックと、マルチプレクサーの前後でそれらのクロックに可能な関係を示します。

図 97: マルチプレクサーを介するクロック



- パス A、B、C が存在しない場合

clk0 および clk1 は、マルチプレクサーのファンアウト (FDM0 および FDM1) でのみ関連性があります。clk0 および clk1 に直接クロック グループ制約を安全に適用できます。

```
set_clock_groups -logically-exclusive -group clk0 -group clk1
```

- パス A、B、C のいずれかしか存在しない場合

clk0 および clk1 の両方またはいずれかは、マルチプレクサーを介するクロックと直接関連性があります。タイミング パス A、B、C を保持するために、制約を clk0 および clk1 に直接適用することはできません。マルチプレクサーのファンアウトのクロック部分に制約を適用する必要があります。これには、クロック定義を追加する必要があります。

```
create_generated_clock -name clk0mux -divide_by 1 \
-source [get_pins mux/I0] [get_pins mux/O]

create_generated_clock -name clk1mux -divide_by 1 \
-add -master_clock clk1 \
-source [get_pins mux/I1] [get_pins mux/O]

set_clock_groups -physically-exclusive -group clk0mux -group clk1mux
```

非同期クロック グループおよびクロック乗せ換えの制約

共通のプライマリ クロックまたは共通周期がないクロック ペアの非同期関係は、クロック関連性レポートですばやく検出できます。クロック周期が同じ場合でも、異なるソースから生成されていれば、クロックどうしは非同期のままになります。非同期 CDC (クロック乗せ換え) パスは注意して確認し、タイミングの正確さに依存しない、メタステーブル状態が発生する可能性を最小限に抑えた適切な同期回路が使用されるようにする必要があります。非同期 CDC パスには通常、スキューが大きく、非現実的なパス要件があります。これらのパスは、ハードウェアで機能することを判断できないので、デフォルトのタイミング解析では解析しないようにする必要があります。

CDC レポート

[Report CDC] (report_cdc) コマンドは、デザインのクロック乗せ換えの構造解析を実行します。この情報を使用して、メタステーブル状態またはデータ コヒーレンスの問題が発生する可能性のある危険な CDC を特定できます。CDC レポートはクロック関連性レポートと似ていますが、構造および関連するタイミング制約に焦点が置かれます。非同期クロック ドメイン間をまたぐパスではタイミング スラックは意味がないので、CDC レポートにはタイミング情報は示されません。

CDC レポートでは、次のような最も一般的な CDC トポロジが特定されます。

- 1 ビット シンクロナイザー
- バス用の複数ビット シンクロナイザー
- 非同期リセット シンクロナイザー
- MUX および CE で制御される回路
- シンクロナイザーの前の組み合わせロジック
- 複数クロックからのシンクロナイザーへのファンイン
- デスティネーション クロック ドメインへのファンアウト

report_cdc コマンドの詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906) の [このセクション](#) を参照してください。また、『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835) の [report_cdc](#) も参照してください。

非同期クロック乗せ換えでデフォルトのタイミング解析が実行されるのを回避するため、特定の制約を適用する必要があります。

関連情報

[クロック間の双方向のグローバル制約](#)

[各 CDC パスの制約](#)

クロック間の双方向のグローバル制約

最大レイテンシを制限する必要がない場合は、クロック グループを使用できます。次に、clkA および clkB 間のパスを無視する例を示します。

```
set_clock_groups -asynchronous -group clkA -group clkB
```

2 つのマスター クロックとそれぞれの生成クロックにより 2 つの非同期ドメインが形成されており、それらのドメイン間のすべてのパスが適切に同期化されている場合は、クロック グループ制約を一度に複数のクロックに適用できます。

```
set_clock_groups -asynchronous \
-group {clkA clkA_gen0 clkA_gen1 } \
-group {clkB clkB_gen0 clkB_gen1 }
```

または、単純に次を使用します。

```
set_clock_groups -asynchronous \
-group [get_clocks -include_generated_clock clkA] \
-group [get_clocks -include_generated_clock clkB]
```

各 CDC パスの制約

CDC パスがグレイ コードを使用する場合 (FIFO など) や、1 つまたは複数の信号で 2 つの非同期クロック間のレイテンシを制限する必要がある場合、set_max_delay 制約を -datapath_only オプションを使用して指定してこれらのパスのクロック スキューおよびジッターが無視されるようにし、さらにデフォルト パス要件をレイテンシ要件で置き換える必要があります。通常は最大遅延値にソース クロック周期を使用するので十分ですが、任意の時間で CDC パスに複数のデータが存在することがないように注意してください。

クロック周期間の比率が大きい場合、最小のソースとデスティネーション クロック周期を選択して転送レイテンシを削減することもできます。クリーンな非同期 CDC パスにはソースおよびデスティネーション シーケンシャル セル間にロジックが含まれないので、通常はインプリメンテーション ツールで set_max_delay -datapath_only 制約を簡単に満たすことができます。

一部の非同期 CDC パスでは、バス レイテンシの制約の代わりに、バスのビット間のスキュー制御が必要です。バス スキュー制約を使用すると、受信クロック ドメインで同じクロック エッジでバスの複数のステートがラッチされるのが回避されます。バスにバス スキュー制約を設定するには、set_bus_skew コマンドを使用します。たとえば、グレイ コードを使用する CDC パスに、set_max_delay -datapath_only 制約の代わりに set_bus_skew を適用できます。詳細は、『Vivado Design Suite ユーザー ガイド: 制約の使用』 (UG903) のこのセクションを参照してください。

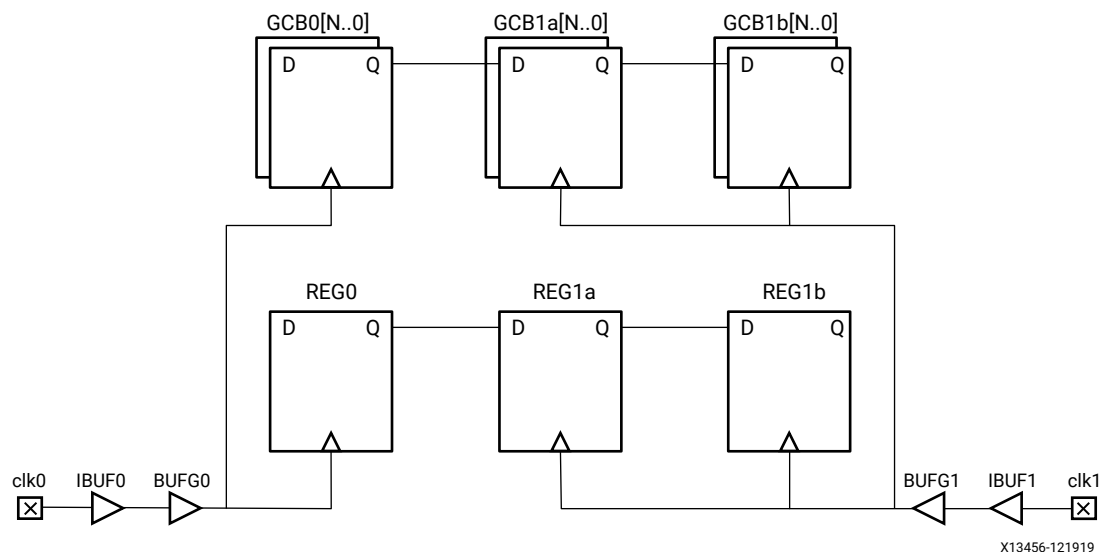
レイテンシ制御を必要としないパスの場合は、ポイント トゥ ポイントのフォルス パス制約を定義できます。

set_max_delay より優先されるクロック例外

CDC 制約を記述する場合は、優先度を考慮する必要があります。set_max_delay -datapath_only を 2 つのクロック間の少なくとも 1 つのパスに使用すると、同じクロック間に set_clock_groups 制約は使用できず、2 つのクロック間のほかのパスに使用できるのは set_false_path 制約のみです。

次の図では、clk0 クロックの周期は 5 ns で、clk1 とは非同期です。clk0 ドメインから clk1 ドメインまでには、2 つのパスがあります。最初のパスは 1 ビット データ同期で、2 つ目のパスは複数ビットのグレイ コードのバス転送です。

図 98: 2 つの非同期クロック間の複数の関連性



グレイ コードのバス転送でビット内の遅延パターンを制限するために set_max_delay -datapath_only 制約が必要かどうかは設計者が判断するので、クロック間に直接クロック グループまたはフォルス パス制約を使用することはできません。代わりに、2 つの制約を定義する必要があります。

```
set_max_delay -from [get_cells GCB0[*]] -to [get_cells [GCB1a[*]]] \
-datapath_only 5
set_false_path -from [get_cells REG0] -to [get_cells REG1a]
```

この例にはパスがないので、clk1 から clk0 までのフォルス パスを設定する必要はありません。

タイミング例外の指定

タイミング例外は、特定のパスでのタイミング解析の実行方法を変更するために使用します。デフォルトでは、最悪のクロック状況を網羅するため、タイミング エンジンによりセットアップ解析ですべてのパスにシングル サイクル要件のタイミング制約が適用されますが、パスによってはこれは当てはまりません。次に、その例をいくつか示します。

- 非同期 CDC パスにクロック間に固定された位相関係がないので、安全にタイミング解析できない。これらのパスは無視するか(クロック グループ、フォルス パス制約)、データパス遅延制約 (set_max_delay -datapath_only) を設定する必要があります。

- シーケンシャル セルのソース クロック エッジおよびデスティネーション クロック エッジがすべてのクロック サイクルでアクティブになるわけではない。この場合、パス要件を緩和できます (マルチサイクル パス)。
- ハードウェアでのデザイン マージンを増加するため、パス遅延要件を厳しくする必要がある (最大遅延)。
- 組み合わせセルを介するパスがスタティックで、タイミングを適用する必要がない (フォルス パス、ケース解析)。
- マルチプレクサーで駆動される特定のクロックでのみ解析を実行する必要がある (ケース解析)。

どの場合でも、タイミング例外の使用には注意が必要で、実際のタイミング問題を隠すために追加するべきではありません。

タイミング例外のガイドライン

タイミング例外は、使用する数を制限し、できる限り簡潔にします。そうしないと、次の問題が発生します。

- 例外が多用されると (特に多数のネットリスト オブジェクトに使用される場合)、インプリメンテーション コンパイル時間が大幅に長くなります。
- 複数の例外が同じパスに適用されると、制約のデバッグが複雑になります。
- 信号の制約により、その信号の最適化が妨害されることがあります。不要な例外や例外コマンドに不要なポイントを含めると、最適化が妨害される可能性があります。

次に、実行時間に悪影響を及ぼすタイミング例外の例を示します。

```
set_false_path -from [get_ports din] -to [all_registers]
```

- `din` ポートに入力遅延がない場合は、制約は設定されないの、フォルス パスを追加する必要はありません。
- `din` ポートが順次エレメントにのみ接続される場合、シーケンシャル セルに対してフォルス パスを明示的に指定する必要はありません。制約は、次のように記述した方が効率的です。

```
set_false_path -from [get_ports din]
```

- フォルス パスが必要だが `din` ポートからシーケンシャル セルへのパスが 2、3 個しかない場合は、制約をより特定の指定します (`all_registers` を使用すると、デザインで使用されるレジスタの数によって何千ものセルが返される可能性あり)。

```
set_false_path -from [get_ports din] -to [get_cells blockA/config_reg[*]]
```

タイミング例外の優先順位と規則

タイミング例外には、厳しい優先順位と規則が適用されます。最も重要な規則は、次のとおりです。

- 制約が具体的になるほど、優先順位が高くなります。次に例を示します。

```
set_max_delay -from [get_clocks clkA] -to [get_pins inst0/D] 12
set_max_delay -from [get_clocks clkA] -to [get_clocks clkB] 10
```

最初の `set_max_delay` の方が、`-to` オプションでピンが使用されていて、クロックよりも具体的なので優先順位は高くなります。

- 例外の優先順位は次のとおりです。

1. `set_false_path`
2. `set_max_delay` または `set_min_delay`
3. `set_multicycle_path`

`set_clock_groups` コマンドは、2つのクロック間に2つの `set_false_path` コマンドを使用するのと同じですが、タイミング例外とは考慮されません。`set_clock_groups` はタイミング例外よりも優先順位が高くなります。

`set_case_analysis` および `set_disable_timing` コマンドは、デザインの特定部分でのタイミング解析をディスエーブルにします。これらのコマンドはタイミング例外よりも優先順位が高くなります。

XDC の優先順位の詳細は、『Vivado Design Suite ユーザー ガイド: 制約の使用』(UG903) のこのセクションを参照してください。

フォルス パス制約の追加

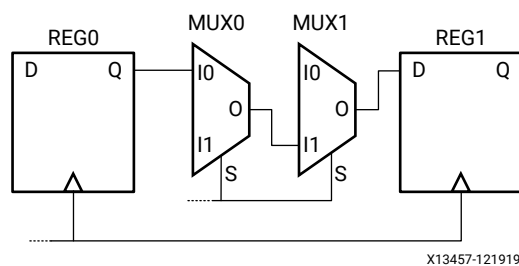
フォルス パス例外をタイミング パスに追加すると、これらのパスのスラック算出が無視されます。通常は、シミュレーション ツールを使用しても、パスが機能するためにタイミングが必要でないということを証明するのは困難です。ザイリンクスでは、フォルス パスを使用する場合のリスクを理解せずに、フォルス パスを使用することはお勧めしません。

ユース ケース

フォルス パス制約を使用する典型的な状況は、次のとおりです。

- アクティブにならないパスのタイミングを無視する場合。たとえば、セレクト ピンが接続されているため、同じクロック サイクルではデータを伝搬することのできない2つのマルチプレクサーを介するパスなどです。

図 99: 認識できないパス



```
set_false_path -through [get_pins MUX0/I0] -through [get_pins MUX1/I1]
```

- 非同期 CDC パスのタイミングを無視する場合。
- デザインのスタティック パスを無視する場合。レジスタの中には、アプリケーションの初期化段階で値を取り込んだ後、二度とトグルしないものがあります。これらがデザインのクリティカル パスにある場合、そのタイミングを無視してインプリメンテーションで制約を緩和すると、タイミング クロージャを達成しやすくなります。明示的にパスの終点を指定しなくても、スタティック レジスタからのフォルス パス制約を定義するだけで十分です。たとえば、32 ビット コンフィギュレーション レジスタ `config_reg[31..0]` からデザインの残りの部分へのパスは、次のフォルス パス制約を適用して無視できます。

```
set_false_path -from [get_cells config_reg[*]]
```

合成への影響

フォルス パス制約は合成ツールでサポートされており、最大遅延(セットアップ/リカバリ)パスの最適化にのみ影響します。通常は、CDC パスを無視する場合を除き、合成中にフォルス パス例外を使用する必要はありません。

インプリメンテーションへの影響

すべてのインプリメンテーション段階で、フォルス パス タイミング例外が認識されます。

最小および最大遅延制約の追加

最小および最大遅延例外は、それぞれホールド/リムーバルおよびセットアップ/リカバリのチェックのデフォルトパス要件のソース クロック エッジ タイムおよびデスティネーション クロック エッジ タイムを制約からの遅延値に置き換えるために使用されます。

ユース ケース

最小または最大遅延制約を使用する主な理由は、次のとおりです。

- セットアップ/リカバリ パス要件を厳しくすることで、デザインの 2、3 のパスの制約を過剰にする。
これにより、ロジック最適化または配置が一部のクリティカル パスのセルに集中的に実行されるようにすることができ、配線でタイミングを満たすために柔軟な処理が可能になります。

- マルチサイクル制約を置き換える。

これは、ソース クロック エッジおよびデスティネーション クロック エッジが N クロック サイクルごとにアクティブになるパスのセットアップ要件を緩和するのには有効ですが、推奨される方法ではありません。ただし、これが配線中にタイミング クロージャを達成しやすくするためにマルチサイクル パスをクロック周期のごく小さい割合だけ厳しくすることができる唯一のオプションです。たとえば、マルチサイクル制約が 3 に設定されたパスが配線後に違反が最も大きいワースト パスになり、数百 ps タイミングが満たされていないとします。

この場合、元のマルチサイクル パス制約を、最適化および配置段階で次の制約に置き換えることができます。14.5 は 3 クロック周期に相当し (それぞれ 5 ns)、-500 ps は必要な外部マージンの量に相当します。

```
set_max_delay -from [get_pins <startpointCell>/C] \  
-to [get_pins <endpointCell>/D] 14.5
```

- 非同期 CDC パスに最大データパス遅延制約を付ける。

この手法の詳細は、[クロック グループおよび CDC 制約の定義](#) を参照してください。

set_min_delay 制約を使用してパスに遅延を挿入する方法はあまり使用されず、推奨もされません。通常は、ホールドまたはリムーバルに対するデフォルトの最小遅延要件だけで十分で、スラックが正の値の場合はハードウェアが問題なく機能します。

合成への影響

set_max_delay 制約は、-datapath_only オプションも含め、合成でサポートされます。set_min_delay 制約は無視されます。

インプリメンテーションへの影響

set_max_delay 制約は、セットアップ パス要件に置き換わるもので、インプリメンテーション フロー全体に影響します。set_min_delay 制約はホールド パス要件に置き換わるもので、ホールドを修正する必要がある場合にのみ配線動作に影響します。

パス分割の回避

パス分割は、`-from` および `-to` コマンドの `set_max_delay` または `set_min_delay` オプションに無効な始点または終点が指定されると発生します。`set_max_delay` によりパスでパス分割が発生すると、デフォルトのホールド解析は実行されなくなります。ホールド解析も制約する必要がある場合は、同じパスに `set_min_delay` を設定する必要があります。セットアップ解析でも、同じパスに `set_min_delay` を設定する必要があります。

パス分割は、次のようにタイミング解析の基盤を変更してしまうので、通常は使用しないでください。

- パス分割により、分割されたパスのクロック スキュー算出が分割される。
- パス分割では、`set_max_delay` または `set_min_delay` コマンドで設定されたパスよりも多くのパスが分割される。

有効な始点および終点

パス分割は、制約が適用されたときにツールによりログ ファイルでレポートされます。有効な始点および終点を使用すると、パス分割を回避できます。

- 始点: クロック、クロック ピン、シーケンシャル セル (セルの有効な始点ピン)、入力または入出力ポート。
- 終点: クロック、シーケンシャル セルの入力データ ピン、シーケンシャル セル (セルの有効な終点ピン)、出力または入出力ポート。

パス分割の詳細は、『Vivado Design Suite ユーザー ガイド: 制約の使用』(UG903) のこのセクションを参照してください。

マルチサイクル パス制約の追加

マルチサイクル パス例外は、デザインの機能を反映する必要があり、アクティブ クロック エッジがクロック サイクルごとでないパスに対して、ソース クロック、デスティネーション クロック、または両方のクロックに適用する必要があります。パスの乗数は、`-start` オプションを使用する場合はソース クロック、`-end` オプションを使用する場合はデスティネーション クロックに基づいてクロック サイクル数で指定します。これは、クロック周期値とは関係なく、始点および終点間のセットアップおよびホールド関係を修正する場合に特に便利です。

ホールドは、常にセットアップと関連しています。そのため、ほとんどの場合、セットアップ関係を変更した後にホールド関係も別に調整する必要があります。2 つ目の制約に `-hold` オプションが必要なのはこのためです。この規則に対する主な例外は、位相シフト クロック間の同期 CDC パスで、この場合はセットアップのみを変更する必要があります。

ホールドを変更しないままセットアップ要件を緩和

これは、N サイクルごとにクロックをアクティブにするクロック イネーブル信号でソースおよびデスティネーション シーケンシャル セルが制御される場合に発生します。次の例では、クロック イネーブルは 3 サイクルごとにアクティブになり、始点と終点のクロックは同じです。

図 100: 同じクロック信号でフリップフロップをイネーブル

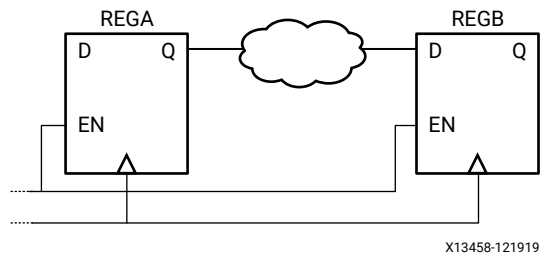
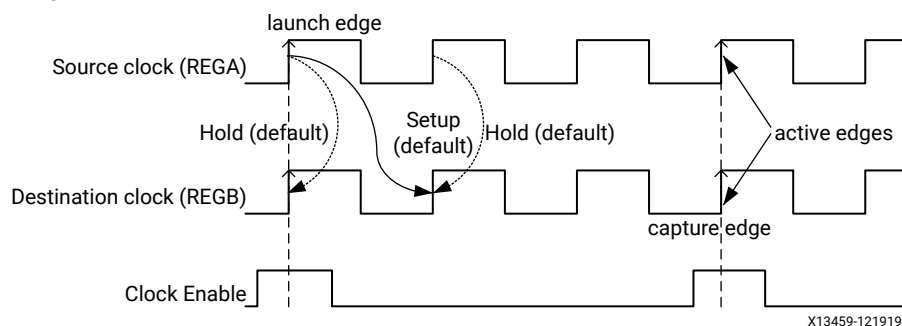


図 101: セットアップ/ホールド チェックのタイミング図

BEFORE

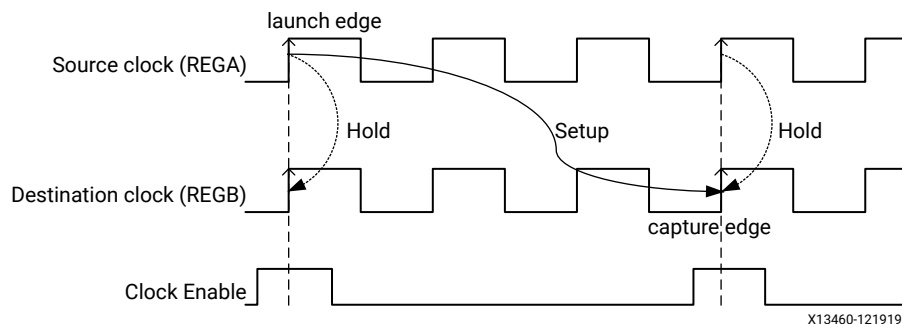


制約:

```
set_multicycle_path -from [get_pins REGA/C] -to [get_pins REGB/D] -setup 3
set_multicycle_path -from [get_pins REGA/C] -to [get_pins REGB/D] -hold 2
```

図 102: マルチサイクルの指定により変更されたセットアップ/ホールド チェック

AFTER



注記: 1 つ目のコマンドでは、セットアップ デスティネーション エッジが 3 つ目のエッジに (デフォルトの位置から 2 サイクル分) 移動されるのに合わせて、ホールド エッジも 2 サイクル分移動されます。2 つ目のコマンドでは、2 サイクル分逆方向に移動して、ホールド エッジが元の位置に戻されます。

同期クロック間の位相シフトおよびマルチサイクル パスなど、マルチサイクル パスのその他の状況に関する詳細は、『Vivado Design Suite ユーザー ガイド: 制約の使用』 (UG903) のこのセクションを参照してください。



重要: クロック位相シフトによりクロック波形は変更されず、クロック調整ブロックの挿入遅延に含まれる場合は、このクロックからのパスまたはこのクロックへのパスに対してタイミング解析を正しく解析するためにセットアップのみのマルチサイクル パスを追加する必要はありません。詳細は、『Vivado Design Suite ユーザーガイド: デザイン解析およびクロージャ テクニク』 (UG906) のこのセクションを参照してください。

合成およびインプリメンテーションへの影響

`set_multicycle_path` 制約は合成でサポートされており、各クロック サイクルでアクティブにならない長いパスの制約を緩和することで、セットアップのみのタイミング QoR (結果の品質) をかなり改善できます。

合成に関しては、マルチサイクル パス例外を使用すると、インプリメンテーションのタイミング ドリブン アルゴリズムが実際のクリティカル パスに集中できます。ホールド要件は配線でのみ重要です。セットアップ関係は `set_multicycle_path` 制約で調整できますが、ホールド関係には対応しておらず、ワースト ホールド要件は 2、3 ns 超えると満たしにくくなることがあります。この状況では、ホールド違反は修正できますが、配線により遅延が追加されるので、セットアップ スラックに悪影響があります。

典型的なミス

次に、回避する必要のある一般的なミスを 2 つ示します。

- マルチサイクルが各クロック サイクルでアクティブにならない場合に、ホールドを同じソース クロック エッジおよびデスティネーション クロック エッジに戻さずにセットアップを緩和する。

ホールド要件が大きくなりすぎ (ほとんどの場合少なくとも 1 クロック周期)、満たすことができなくなります。

- デザイン内の間違ったポイント間にマルチサイクル パス例外を設定する。

これは、始点セルから終点セルまでにパスが 1 つしかない想定した場合に発生します。そうではない場合もあります。終点セルには、クロック イネーブルおよびリセット ピンを含め、複数のデータ入力ピンがあり、少なくとも 2 つの連続したクロック エッジでアクティブになります。

このためザイリンクスでは、セル (またはクロック) だけでなく、終点ピンを指定することをお勧めします。たとえば、終点セル REGB には C、EN、D の 3 つの入力ピンがありますが、マルチサイクル パス例外の制約を設定する必要があるのは REGB/D ピンだけです。EN ピンには、各クロック サイクルで変更される可能性があるため、制約は設定しません。制約がピンではなくセルに適用される場合は、EN (クロック イネーブル) ピンを含むすべての有効な終点ピンがその制約に対して考慮されます。

確実にするため、ザイリンクスでは常に次の構文を使用することをお勧めします。

```
set_multicycle_path -from [get_pins REGA/C] \  
-to [get_pins REGB/D] -setup 3  
set_multicycle_path -from [get_pins REGA/C] \  
-to [get_pins REGB/D] -hold 2
```

その他のアドバンス タイミング制約

次のセクションで説明するその他のタイミング制約を設定すると、デフォルトのタイミング解析を無視および変更できます。

ケース解析

ケース解析コマンドは、デザインの機能モードを記述するのによく使用され、コンフィギュレーション レジスタのようにロジックに定数を設定することにより指定します。この制約は、入力ポート、ネット、階層ピン、または最下位セル入力ピンに適用できます。定数値がロジックを介して伝搬されると、アクティブになることがないパスの解析がオフになります。フォルス パス例外と同様の効果があります。

最もよく使用されるのは、マルチプレクサーのセレクト ピンを 0 か 1 に設定して、2 つのマルチプレクサー入力のうちの 1 つだけが伝搬されるようにする場合です。次の例では、mux/S および mux/I1 ピンを介するパスの解析がオフになります。

```
set_case_analysis 0 [get_pins mux/S]
```

タイミングのディスエーブル

ディスエーブル タイミング コマンドでは、タイミング データベースのタイミング アークがオフになり、そのアークを介した解析が完全に回避されます。ディスエーブルにされたタイミング アークは、report_disable_timing コマンドでレポートできます。



注意: ディスエーブル タイミング コマンドは注意して使用しないと、パスが意図した以上に分割されてしまう可能性があります。

データ チェック

set_data_check コマンドでは、2 つのピン間のセットアップまたはホールド タイミング チェックと同等のチェックが設定され、たとえば、非同期インターフェイスのタイミングをレポートするのにこの制約を使用できます。このコマンドは、インプリメンテーション ツールでは無視されるので、タイミングをレポートするためにのみ上級ユーザーが使用してください。

最大タイム ボローイング

set_max_time_borrow コマンドでは、ラッチが次の段 (ラッチ後のロジック) から借用可能な最大時間が設定され、前の段 (ラッチ前のロジック) に渡されます。ラッチはハードウェアではテストおよび検証が困難なので、通常推奨されません。このコマンドは、アドバンス ユーザー用です。

物理制約の定義

物理制約は、フロアプラン、特定配置、I/O 割り当て、配線、および同様のファンクションを制御するために使用されます。各ピンに対して I/O ロケーションと規格が指定される必要があります。物理制約の詳細は、次のユーザー ガイドを参照してください。

- マクロの相対的な配置を含む配置配線のロックについては、『Vivado Design Suite ユーザー ガイド: 制約の使用』(UG903) を参照してください。
- フロアプランについては、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906) を参照してください。
- コンフィギュレーションについては、『Vivado Design Suite ユーザー ガイド: プログラムおよびデバッグ』(UG908) を参照してください。

デザイン インプリメンテーション

デバイスを選択し、IP を選択および設定して、RTL および制約を記述したら、インプリメンテーションを実行します。インプリメンテーションでは、合成と配置配線を介してデザインがコンパイルされ、デバイスのプログラムに使用するファイルが生成されます。インプリメンテーション プロセスでは、反復作業が必要な場合があります。この章では、インプリメンテーションの各段階を説明し、特別な注意が必要な点を指摘し、問題を特定して回避するためのヒントおよび手法を示します。

注記: インプリメンテーション段階は、Vitis™ 環境フローの一部として自動的に実行されます。この章で説明されている手法を Vitis コマンド ライン オプションおよびコンフィギュレーション ファイルを使用して適用することで、パフォーマンスを向上できます。詳細は、『Vitis 統合ソフトウェア プラットフォームの資料』 ([UG1416](#)) を参照してください。

合成の実行

合成では、RTL およびタイミング制約を入力し、RTL と機能的に同等な最適化されたネットリストを生成します。通常、合成ツールでは、任意の有効な RTL を使用して、そのロジックを作成できます。合成では、現実的なタイミング制約が必要です。

合成の詳細は、次を参照してください。

- 『Vivado Design Suite ユーザー ガイド: 合成』 ([UG901](#))
- [Vivado Design Suite QuickTake ビデオ: デザイン フローの概要](#)

関連情報

[デザイン制約](#)

[デザインのベースライン制約の作成](#)

合成フロー

Vivado® Design Suite では、次のセクションで説明する合成フローを実行できます。各フローには、それぞれ利点とトレードオフがあります。

グローバル合成

グローバル合成フローでは、1 回の実行でデザイン全体が合成されます。このフローには、次の利点があります。

- 合成ツールで最大限の最適化を実行可能。合成ツールでデザイン全体が認識されるので、ほかのフローでは不可能な階層間での最適化を実行できます。
- 合成後の解析が簡単。

このフローの欠点は、コンパイル時間が長いことです。合成を実行するたびに、デザイン全体が再合成されます。この欠点は、インクリメンタル合成を使用して回避できます。

注記: デザインに XDC 制約が含まれる場合は、最上位デザインを基準にオブジェクトを参照する必要があります。

関連情報

インクリメンタル合成

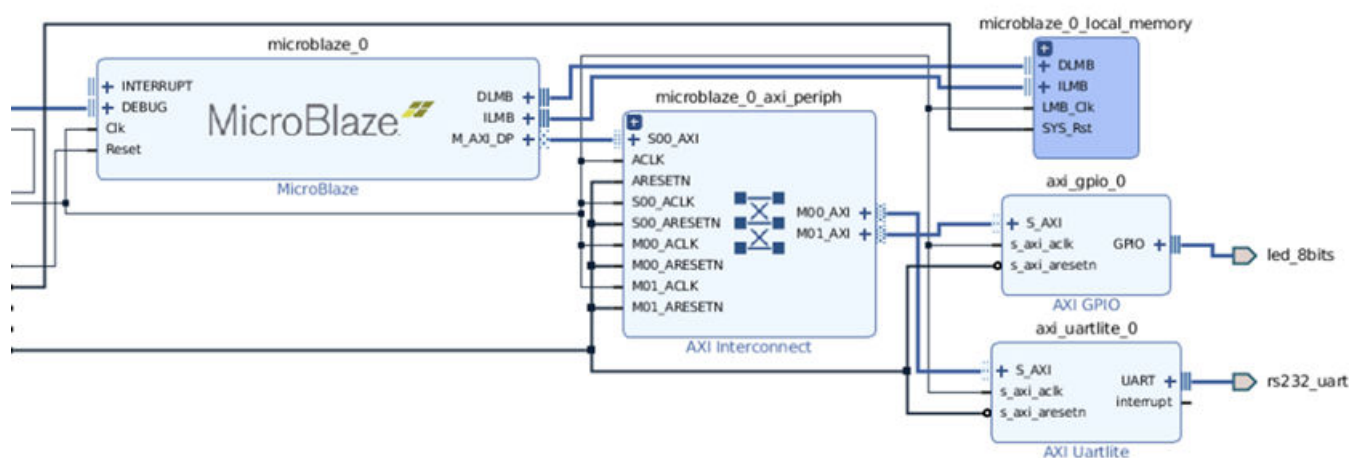
ブロック デザインの合成

ブロック デザインの合成フローでは、カスタム IP およびザイリンクス IP を使用して、複雑なシステムを作成できます。このフローでは、Vivado IP インテグレーターを使用してブロック デザインを作成します。ザイリンクス IP またはカスタム IP を .bd ファイルに追加し、システムとして接続します。このフローには、次の利点があります。

- 1 つのデザインにさまざまな機能を組み込むことができます。
- システムの個別の部分ではなく、システムに焦点を置くことができます。
- デザインの設定および合成を簡単に高速に実行できます。

次の図に、ブロック デザインの例を示します。

図 103: ブロック デザインの例



ブロック デザインを作成する際は、アウト オブ コンテキスト (OOC) 合成モードまたはグローバル合成モードで合成を実行できます。アウト オブ コンテキスト合成モードを使用すると、ブロック デザインは残りのデザインから独立させて合成されます。これにより、BD 外の階層が変更された場合に、再合成を高速に実行できます。グローバル合成モードを使用すると、毎回デザイン全体がコンパイルおよび合成されます。グローバル合成モードは、制約がグローバル レベルで設定されるので、セットアップは簡単です。ただし、このモードを使用すると、再合成の際に実行時間が長くなります。インクリメンタル合成を使用すると、実行時間を短縮できます。

アウト オブ コンテキスト合成

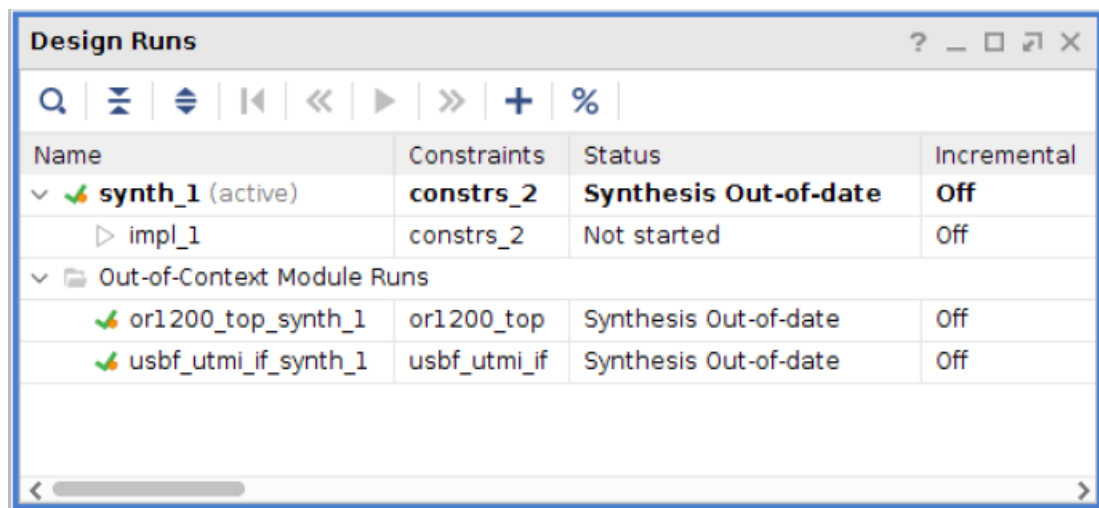
アウト オブ コンテキスト (OOC) 合成フローでは、階層の特定のレベルが最上位から独立させて合成されます。アウト オブ コンテキスト階層が最初に合成されます。その後最上位合成が実行され、各アウト オブ コンテキスト run はブラック ボックスとして扱われます。ザイリンクス IP はアウト オブ コンテキスト合成モードで実行されるのがほとんどです。すべてのアウト オブ コンテキスト合成および最上位合成が完了すると、最上位合成済みデザインを開いたときに、Vivado ツールですべての合成 run からデザインがアSEMBルされます。このフローには、次の利点があります。




- 次回からの合成 run のコンパイル時間が短縮されます。指定した run のみが再合成され、ほかの run はそのままになります。
- デザインを変更した場合に安定した結果を取得できます。変更を含む run のみが再合成されます。

このフローの欠点は、追加の設定が必要なことです。どのモジュールをアウト オブ コンテキスト合成モジュールとして設定するかを選択する際は、注意が必要です。追加の XDC 制約を個別に定義する必要があり、アウト オブ コンテキスト合成 run でのみ使用します。

次の図に、最上位合成 run (synth_1) と 2 つの下位アウト オブ コンテキスト合成 run を含むデザインを示します。

図 104: 2 つのアウト オブ コンテキスト合成 run を含む最上位合成



Name	Constraints	Status	Incremental
▼  synth_1 (active)	constrs_2	Synthesis Out-of-date	Off
▶ impl_1	constrs_2	Not started	Off
▼ Out-of-Context Module Runs			
 or1200_top_synth_1	or1200_top	Synthesis Out-of-date	Off
 usbf_utmi_if_synth_1	usbf_utmi_if	Synthesis Out-of-date	Off

アウト オブ コンテキスト合成 run の設定に関する詳細は、『Vivado Design Suite ユーザー ガイド: 合成』(UG901) のこのセクションを参照してください。

インクリメンタル合成

インクリメンタル合成を使用すると、既存の合成結果を再利用できます。これには、次のような利点があります。

- 通常合成コンパイル時間が 50% 短縮されます。
- インクリメンタル インプリメンテーション フローと共に使用すると、全体的なコンパイル時間およびタイミングクローズジャの一貫性が向上します。

インクリメンタル合成は、最上位デザインが RTL で、デザインの大部分が RTL である場合に最も有益です。このモードでは、合成のコンパイル時間が最適化され、結果が再利用されるので、ブロック デザインまたは IP、あるいはその両方が多く含まれるデザインでは、Vivado ツールでこれらのブロックの合成が自動的にアウト オブ コンテキスト (OOC) モードで個別に実行されます。このようなデザインでは、インクリメンタル合成を実行する価値はそれほどありません。

デザインのほとんどが RTL で、実行時間を短縮または run 間の変動を低減するために OOC 合成フローを使用している場合は、OOC run をインクリメンタル合成フローに変換すると有益場合があります。インクリメンタル合成フローは、実行時間を短縮し、run 間の変動を低減しますが、合成で QoR を高めるための最適化をより多く実行できます。インクリメンタル合成フローに変換する場合は、タイミング制約を OOC run から最上位に移動する必要があります。

インクリメンタル合成は、QoR に悪影響を与えずにイネーブルにできます。合成後のデザイン チェックポイントのサイズは約 2 倍になり、合成データが読み込まれて再利用されない場合は実行時間がわずかに増加します。

インクリメンタル合成を使用すると、変更されていない階層が基準合成 run から再利用されるので、実行時間が短縮されます。インクリメンタル合成を効果的に実行するには、デザインに 10000 個以上のインスタンスを含むパーティションが 5 個以上含まれていることが必要です。また、デザインの変更を加えるパーティションの数はできるだけ少なくし、デザインの最上位には変更を加えないようにします。

注記: 変更によっては境界をまたぐ最適化に影響し、再合成が必要なパーティションが増えることがあります。

変更される階層がわかっている場合は、次のプロパティを使用して階層を保持します。これにより、その後の合成 run でインクリメンタル合成で保持された階層のみを再度最適化できるようになります。ただし、このプロパティを使用すると境界をまたぐ最適化は実行されないため、QoR に影響することがあります。

```
set_property BLOCK_SYNTH.PRESERVE_BOUNDARY [get_cells <cellName>]
```



重要: 基準 run とインクリメンタル run で合成オプションを変更すると、Vivado ツールで自動的にデザイン全体が再合成されます。

詳細は、『Vivado Design Suite ユーザー ガイド: 合成』 ([UG901](#)) を参照してください。

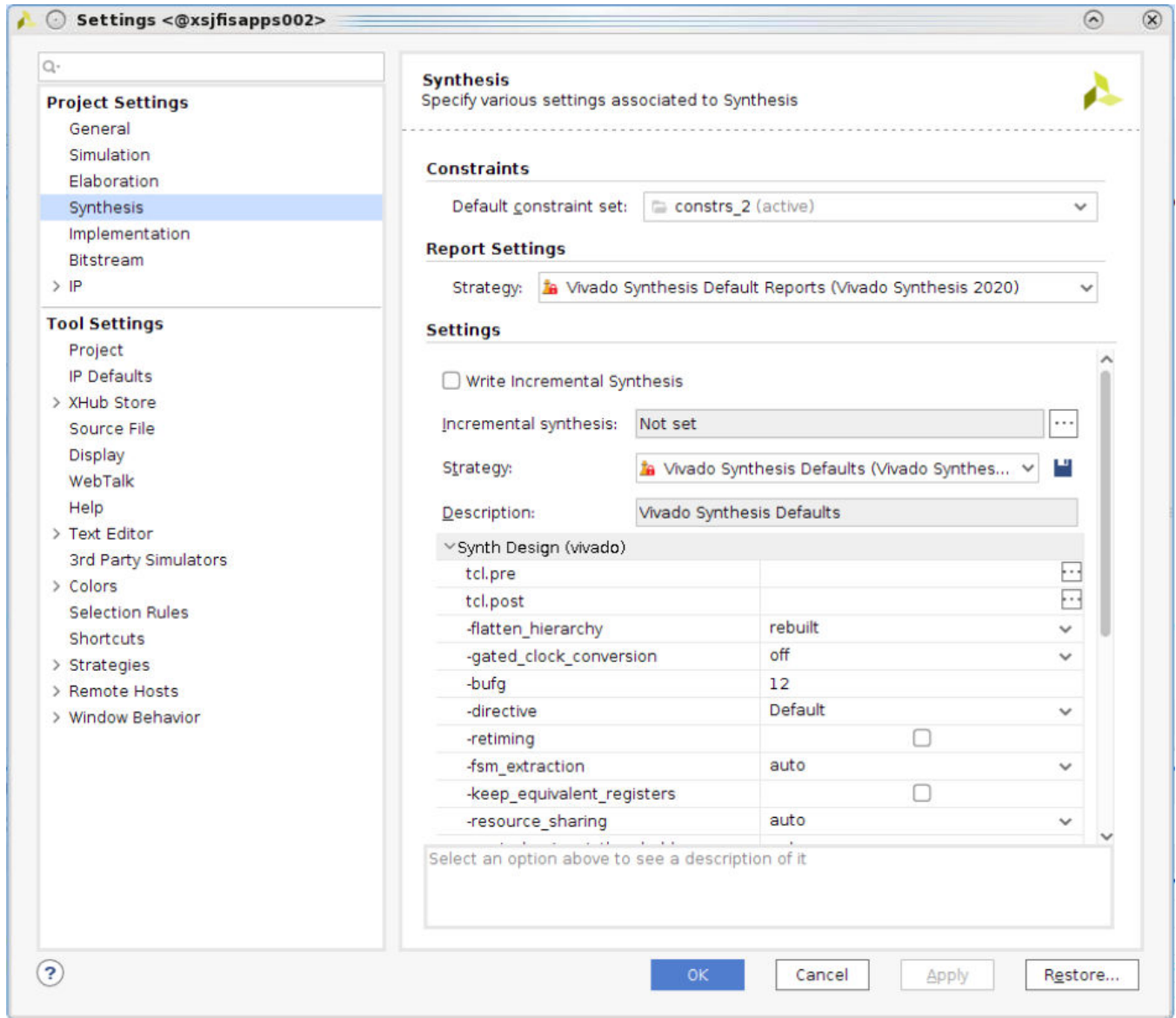
合成最適化

Vivado 合成では、大多数のデザインで最適な結果が得られる最適化がデフォルトで適用されます。これらのデフォルトの合成最適化は、次のセクションで説明するように変更できます。

合成設定

Vivado Design Suite の [Settings] ダイアログ ボックスの [Synthesis] ページでは、デザイン全体に影響するグローバル設定を指定できます。これらの設定は、ロジックの推論方法、インクリメンタル合成の実行方法を最適化します。ザイリンクスでは、まずデザインをデフォルト オプションで実行し、デザイン特定のニーズに応じてオプションを変更することをお勧めします。詳細は、『Vivado Design Suite ユーザー ガイド: 合成』 ([UG901](#)) の [このセクション](#) を参照してください。

図 105: Vivado の [Settings] ダイアログ ボックスの [Synthesis] ページ



合成属性

合成属性を使用すると、ロジックの推論を制御できます。合成アルゴリズムはほとんどのデザインで最適な結果が得られるように設定されていますが、要件の異なるデザインもあります。この場合、属性を使用してデザインを変更すると、QoR (結果の品質) を改善できます。合成でサポートされる属性については、『Vivado Design Suite ユーザー ガイド: 合成』(UG901) を参照してください。



電源/消費電力ヒント: 合成設定はデザインの消費電力に大きな影響を与える可能性があるため、合成設定を注意深く評価してください。たとえば、制御セットのしきい値を低くすると、レジスタのクロック イネーブルの使用率が上がり、集積度は下がります。合成後に `report_power` コマンドを実行して、合成設定が消費電力に与える影響を評価します。

注記: サイリンクスでは、デザインのターゲットを新しいデバイスに変更する前に、古いデバイスをターゲットにした前のデザイン run の合成属性を確認することをお勧めします。

KEEP、DONT_TOUCH および MAX_FANOUT 属性を使用する場合、次のセクションで説明する特定の考慮事項に注意してください。

KEEP および DONT_TOUCH

KEEP および DONT_TOUCH は、デザインをデバッグする際に有益な属性です。これらの属性が配置されたオブジェクトは、最適化で削除されません。

- KEEP 属性は合成ツールで使用され、ネットリストでプロパティとしては渡されません。KEEP 属性は特定の信号を保持するためのもので、たとえば、合成中に特定の信号に対する特定の最適化をオフにします。
- DONT_TOUCH 属性は、合成ツールで使用された後、配置配線ツールにも渡されるので、オブジェクトは最適化されません。

これらの属性の設定には注意が必要です。

- RAM の出力を受信するレジスタに KEEP 属性を設定すると、レジスタが RAM に統合されなくなり、ブロック RAM が推論されなくなります。
- 上の階層のトライステート出力または双方向信号を駆動する階層に、これらの属性を設定しないでください。駆動信号とトライステート状態がこの階層レベルにある場合、IOBUF を作成するには階層を変更する必要があるため、IOBUF が推論されません。
- 最適化をディスエーブルにする属性を使用すると、多くの場合、大型の消費電力の大きい回路になります。サイリンクスでは、これらの制御はできるだけ使用しないようにし、不要になったら削除することをお勧めします。

また、DONT_TOUCH 属性を信号に設定するか階層レベルに設定するかで違いがあります。

- この属性が信号に設定されている場合、その信号が保持されます。
- この属性が階層レベルに設定されている場合、階層の境界は最適化されず、階層を介する定数の伝搬は実行されませんが、その階層レベル内での最適化は実行されます。

MAX_FANOUT

MAX_FANOUT 属性は、ファンアウトの制限を満たすため、強制的にロジックを複製します。ツールでロジックは複製できますが、入力またはブラックボックスは複製できません。デザインへの直接入力で駆動される信号に MAX_FANOUT 属性が設定されていると、ツールでこの制約を処理できません。

MAX_FANOUT 属性が設定されている信号を注意して解析してください。DONT_TOUCH 属性が設定されているレジスタで駆動される信号、または DONT_TOUCH 属性が設定されている異なる階層にある信号を駆動する信号に MAX_FANOUT 属性を設定しても、MAX_FANOUT 属性は適用されません。

合成では、最初に複製されるセルには `_rep` が、その後複製されるセルには `_rep__0`、`_rep__1` という接尾辞が付いていきます。これらのセルは、[Edit]→[Find] をクリックすると、合成後のネットリストに表示されます。



重要: 合成では、MAX_FANOUT 属性は多用しないでください。Vivado® ツールの `place_design` および `phys_opt_design` コマンドは配置に基づく複製を実行するので、合成での論理的複製よりも効果的です。特定のファンアウトが必要な場合は、通常コードに手動でレジスタを追加の方が適切な結果が得られます。

ブロック レベル合成ストラテジ

Vivado 合成では、さまざまなストラテジおよびグローバル設定を使用してデザインの合成方法をカスタマイズできます。ほとんどの場合、これらのオプションはグローバルに適用され、デザイン全体に影響します。ブロック レベル合成ストラテジを使用すると、異なる階層レベルに異なるグローバル オプションを使用してトップダウン フローで合成できます。このフローは、ボトムアップ コンパイルよりも高速で簡単に実行できます。下位レベルに制約を設定して最上位で再設定するのではなく、デザイン全体に制約を設定できます。

ブロック レベル合成ストラテジを設定するには、XDC ファイルで次の構文を使用します。

```
set_property BLOCK_SYNTH.<option_name> <value> [get_cells <instance_name>]
```

説明:

- <option_name>: 設定するオプションを指定します。
- <value>: オプションに割り当てる値を指定します。
- <instance_name>: オプションを設定する階層インスタンスを指定します。

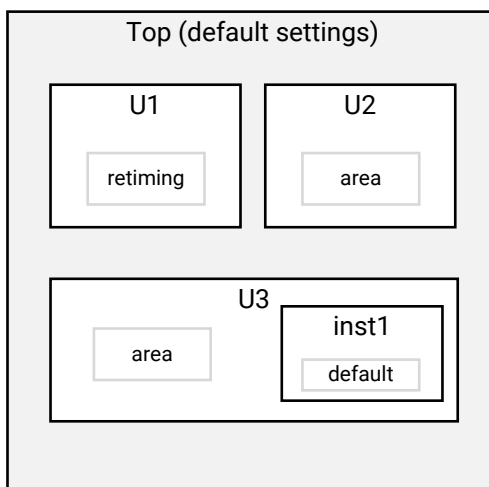
注記: これらのプロパティは、常に階層インスタンスに設定されます。これにより、複数回インスタンス化されているモジュールまたはエンティティを異なるオプションを使用して合成できます。

たとえば、XDC ファイルで次のストラテジを設定できます。

```
set_property BLOCK_SYNTH.RETIMING 1 [get_cells U1]
set_property BLOCK_SYNTH.STRATEGY {AREA_OPTIMIZED} [get_cells U2]
set_property BLOCK_SYNTH.STRATEGY {AREA_OPTIMIZED} [get_cells U3]
set_property BLOCK_SYNTH.STRATEGY {DEFAULT} [get_cells U3/inst1]
```

次の図に示すように Vivado 合成が実行されます。

図 106: ブロック レベル合成ストラテジの例



X19285-121919

同じインスタンスに複数の BLOCK_SYNTH プロパティを設定して、異なるオプションを試すことができます。次に例を示します。

```
set_property BLOCK_SYNTH.STRATEGY {ALTERNATE_ROUTABILITY} [get_cells inst]
set_property BLOCK_SYNTH.FSM_EXTRACTION {OFF} [get_cells inst]
```

IP を使用する場合、ブロック レベル合成ストラテジを次のように使用できます。

- IP がグローバルにコンパイルされる場合、IP の最上位にこのストラテジを使用できます。
- IP がアウト オブ コンテキストである場合、IP はブラック ボックスとして示されるので、このストラテジは使用できません。IP をコンパイルする際は、グローバル設定を使用してください。

注記: この機能の詳細とサポートされるストラテジおよびオプションは、『Vivado Design Suite ユーザー ガイド: 合成』(UG901) を参照してください。

合成後の処理

合成で生成されるネットリストが質の良いもので、後続のフローで問題を引き起こさないことを確認してください。次のセクションでは、残りのインプリメンテーション フローに進む前に確認する必要がある重要項目を示します。

DRC の確認および修正

report_drc コマンドを使用すると、一般的なデザインの問題およびエラーを確認するデザイン ルール チェック (DRC) が実行されます。複数のルール デックがあります。デフォルトのルール デックでは、次がチェックされます。

- 合成後のネットリスト
- I/O、BUFG、およびその他の配置特定の要件
- MGT、IODELAY、MMCM、PLL などのプリミティブの属性および配線



推奨: インプリメンテーション フローの後の方でのタイミングまたはロジック関連の問題を回避するため、DRC 違反をデザイン プロセスのできるだけ早い段階で確認して修正してください。



ヒント: 安全に無視できる DRC 違反は、除外してレポートされないようにすることができます。詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906) のこのセクションを参照してください。

設計手法レポートの生成

Vivado ツールには、設計手法に従っているかどうかをチェックするための設計手法レポート機能が含まれています。デザイン プロセスの段階によって、異なるチェックが実行されます。

- RTL デザイン: RTL リント スタイル チェック
- 合成済みおよびインプリメンテーション済みデザイン: ネットリスト、制約、およびタイミング チェック

プロジェクト モードでは、インプリメンテーション (opt_design または route_design) 中にデフォルトで自動的に設計手法レポート ([Report Methodology] コマンド) が実行されます。これらのチェックを手動で実行するには、次のいずれかの方法を使用します。

- Tcl プロンプトで、検証するデザインを開き、`report_methodology` Tcl コマンドを入力します。
- これらのチェックを Vivado IDE から実行するには、検証するデザインを開き、[Reports]→[Report Methodology] をクリックします。



推奨: よくあるデザイン問題を見つけるには、デザインを最初に合成したときにこのレポートを実行します。そして、モジュールの大幅な変更、制約の変更、またはクロッキング回路の変更後にこのレポートを再実行します。

注記: ザイリンクスが提供する IP コアでは、違反は既に確認されています。

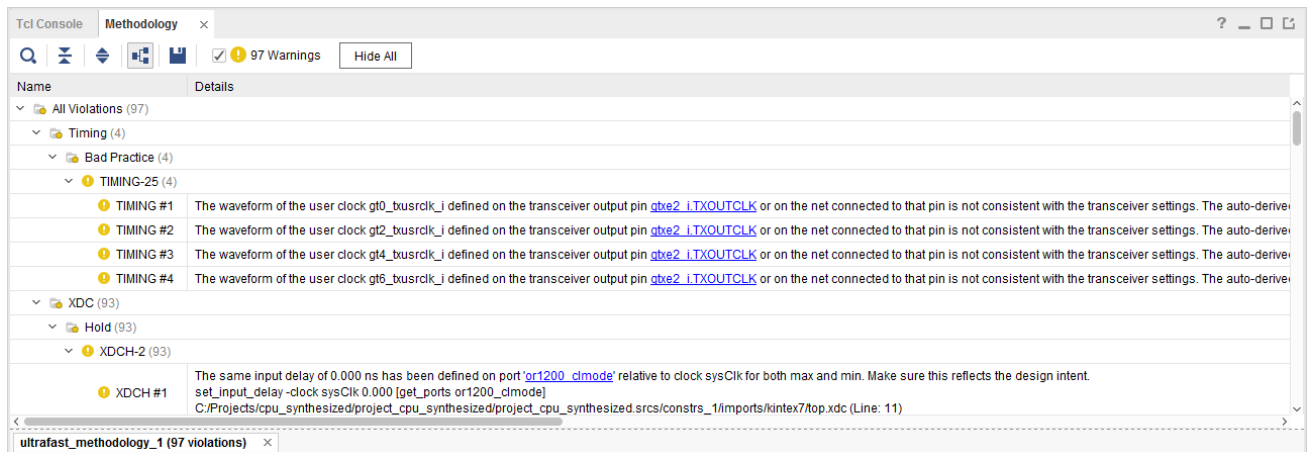
違反が検出された場合は、次の図に示す [Methodology] ウィンドウに表示されます。デザインで特定の手法違反を修正する必要がない場合は、違反とその影響を理解し、なぜその違反がデザインに悪影響を与えないかを把握しておく必要があります。



ヒント: 良い QoR を達成し、タイミング解析が正確なものになるようにするため、すべてのクリティカル 警告およびほとんどの警告を解決する必要があります。詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』 (UG906) のこのセクションを参照してください。安全に無視できる設計手法チェック違反は、除外してレポートされないようにすることができます。詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』 (UG906) のこのセクションを参照してください。

注記: RAMB および DSP プリミティブのオプションのパイプライン処理に関する設計手法チェック (SYNTH-6、SYNTH-11、SYNTH-12、および SYNTH-13) は、プリミティブのすべての入力パスまたは出力パスのセットアップ タイミングが 1 ns を超える場合はレポートされません。

図 107: [Methodology] ウィンドウ



設計手法レポート生成の詳細は、『Vivado Design Suite ユーザー ガイド: システム レベル デザイン入力』 (UG895) を参照してください。また、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』 (UG906) のこのセクションも参照してください。

合成ログの確認

合成ログ ファイルを参照し、ツールで生成されたメッセージがデザインで意図したものと一致しているかどうかを確認する必要があります。クリティカル警告および警告を特に注意して確認してください。ほとんどの場合、信頼性の高い合成結果を得るため、クリティカル警告を修正する必要があります。



注意: あるメッセージが 100 回以上表示される場合、合成ログ ファイルには最初の 100 回のみが記述されます。この制限は、Tcl コマンド `set_param messaging.defaultLimit` を使用して変更できます。

タイミング制約の確認

該当する場合は、クリーンなタイミング制約およびタイミング例外を提供する必要があります。不適切な制約を使用すると、コンパイル時間が長くなり、パフォーマンスの問題やハードウェアでのエラーの原因となります。



推奨: タイミング制約に関するクリティカル警告および警告をすべて確認してください。これらの警告は、制約が読み込まれていないことや適切に適用されなかったことを示します。

関連情報

[デザイン制約の分類](#)

合成後の QoR の評価

QoR 評価レポートは、ロジック段数チェック、使用率チェック、および最も一般的なクロッキング トポロジ チェックを 1 つのサマリ レポートにまとめ、デザインの全体的な評価を示します。このレポートは、タイミング クロージャの問題の重大さを理解するのに役立ちます。ザイリンクスでは、正しいタイミング制約を適用したデザインに大きなネットリスト アップデートを加えた後、合成済みデザインにこのレポートを実行することをお勧めします。

QoR 評価レポートは、デザインがタイミング クロージャを達成する可能性を 1 ～ 5 のスコアで示します。次の表に、スコアの定義を示します。スコア 1 および 2 の場合はタイミング クロージャを達成する可能性はなく、スコア 3 の場合はタイミング クロージャを達成する可能性は低くなります。つまり、スコアが低い場合、タイミング クロージャを達成するのににより多くの作業が必要となります。

表 4: QoR 評価レポートのスコア

スコア	説明
1	デザインがインプリメンテーション フローを完了する可能性は低くなります。
2	デザインはインプリメンテーション フローを完了しますが、タイミング制約は満たしません。
3	デザインがタイミングを満たす可能性は低くなります。
4	デザインがタイミングを満たす可能性はあります。
5	デザインはタイミングを簡単に満たします。

レポートでは、スコアに基づく情報が詳細な表で示されます。詳細な表に示されるしきい値は、デバイスの絶対的な制限ではなく、タイミング クロージャを達成するのがさらに困難になる可能性があることを示します。これらの項目のしきい値を超えると、タイミング クロージャを達成するのが指数関数的に困難になります。

QoR 評価レポートにレビューが必要と示されるすべての項目を修正するようにしてください。QoR 推奨項目レポートを使用すると、多くの項目は自動的に解決されます。

ガイドラインに従って残りの違反を解決



重要: 合成後のタイミングを解析して、フローを続行する前に解決する必要のあるデザインの問題を特定してください。

HDL の変更は、QoR に大きく影響する傾向があるので、タイミングを短期間で満たすことができるようにするため、インプリメンテーションの前に問題を解決しておくのが適切です。タイミング パスを解析する際は、次の事項に特に注意してください。

- タイミングが満たされていない最悪のパスに最多数表示されるセルまたはネット

- レジスタが付けられていないブロック RAM をソースとするパス
- SRL をソースとするパス
- レジスタが付けられていないカスケード接続された DSP ブロック
- ロジック段数の多いパス
- ファンアウトの大きいパス

関連情報

[タイミング クロージャ](#)

ロジック段数が多い場合の処理

長いロジック パスを特定すると、困難な QoR の問題を調べるのに役立ちます。合成後のネット遅延見積もりは、最適な配置に近いものです。ロジック段遅延の大きいパスがタイミングを満たしているかを評価するには、ネット遅延なしでタイミング レポートを生成します。ネット遅延なしでタイミング違反が発生しているパスでは、タイミング クロージャを達成することはできません。

関連情報

[タイミング クロージャ](#)

使用率の確認

LUT、フリップフロップ、ブロック RAM、および DSP コンポーネントの使用率をそれぞれ確認することが重要です。LUT/フリップフロップの使用率が低いデザインでも、ブロック RAM の使用率が高いと、配置が困難になることがあります。report_utilization コマンドを使用すると、すべてのデザイン オブジェクトに対してそれぞれ個別のセクションを含む包括的な使用率レポートが生成されます。

注記: 合成後は、デザイン フロー後半の最適化により使用率の数値が異なることもあります。

クロック ツリーの確認

このセクションでは、クロック バッファの使用率およびクロック ツリー トポロジなど、クロック ツリーの確認方法について説明します。

クロック バッファの使用率

report_clock_utilization コマンドを使用すると、クロック プリミティブの使用率の詳細が表示されます。アーキテクチャのクロッキング規則を確認し、配置の問題が発生しないようにします。配置制約が無効な場合や、リージョナル クロック バッファのファンアウトが大きすぎる場合、配置で問題が発生する可能性があります。クロック バッファの使用率が非常に高いデザインでは、クロック ジェネレーターおよび一部のリージョナル クロック バッファの配置を固定することが必要な場合があります。

ソース同期インターフェイスなど、厳しいタイミング関係が必要なインターフェイスでは、厳しいタイミング関係が必要な信号の特定のリソースを固定した方がよい場合もあります。通常は、上記のような特別な理由がない場合は、初めのうちは I/O のみを固定します。

関連情報

[タイミング クロージャ](#)

クロック ツリーのトポロジ

クロック ツリーを使用する際は、次の推奨事項に従ってください。

- `report_clock_networks` コマンドを実行すると、クロック ネットワークが詳細なツリー ビューで表示されません。
- クロック ツリーをスキューを最小限に抑えるように使用します。
- PLL および MMCM の出力には、スキューを最小限に抑えるため同じクロック バッファ タイプを使用します。
- 意図しないカスケード接続された BUFG エLEMENT がないかどうかを確認します。カスケード接続された BUFG エLEMENT は、遅延やスキューの原因となります。

デザインのインプリメンテーション

Vivado Design Suite インプリメンテーションは、デザインの論理制約、物理制約、タイミング制約を満たしながらネットリストをデバイス リソースに配置配線するためのすべての段階を含みます。インプリメンテーションの詳細は、次の資料を参照してください。

- 『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 ([UG904](#))
- [Vivado Design Suite QuickTake ビデオ: デザイン フローの概要](#)

プロジェクト モードまたは非プロジェクト モードの使用

インプリメンテーションは、プロジェクト モードおよび非プロジェクト モード両方で実行できます。プロジェクト モードでは、run の管理、ファイル セットの管理、レポートの生成、クロスプローブなどのプロジェクト 構造を使用できます。非プロジェクト モードは統合が簡単で、Tcl スクリプトを使用して実行するので、必要なレポートを明示的に作成する必要があります。これらのモードの詳細は、『Vivado Design Suite ユーザー ガイド: デザイン フローの概要』 ([UG892](#)) の [このセクション](#) を参照してください。

ストラテジ

ストラテジは、Vivado Design Suite のプロジェクト モードで、ツール オプションと合成およびインプリメンテーション run で生成されるレポートを制御します。ストラテジを使用すると、インプリメンテーションの目標を変更したり、生成するレポートを指定するために使用できます。ストラテジの詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 ([UG904](#)) を参照してください。



推奨: まずデフォルト ストラテジである Vivado Implementation Defaults を試してみてください。コンパイル 時間とデザイン パフォーマンスのバランスを取ってインプリメンテーションが実行されます。

注記: ストラテジは、ツールおよびバージョン別になっています。ストラテジによっては、コンパイル時間が長くなることがあります。

指示子

-directive オプションを使用すると、次のインプリメンテーション コマンドを異なる指示子を使用して実行できます。

- `opt_design`
- `place_design`

- `phys_opt_design`
- `route_design`

最初は、デフォルト設定を使用してください。デザインが完了に近づいてきたら、ほかの指示子を使用してソリューションの可能性を探ります。一度に指定できるのは、1 つの指示子のみです。指示子の詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 ([UG904](#)) を参照してください。

反復フロー

非プロジェクト モードでは、さまざまな最適化コマンドを異なるオプションを使用して反復できます。たとえば、`phys_opt_design -directive AggressiveFanoutOpt` の後 `phys_opt_design -directive AlternateFlowWithRetiming` を実行すると、タイミングが満たされていない配置済みデザインに対して異なる物理合成最適化を実行できます。

`phys_opt_design` コマンドを繰り返して実行すると、タイミングを向上できる可能性があります。`phys_opt_design` コマンドでは、最も問題のあるパスが最適化されます。`phys_opt_design` コマンドを反復実行すると、より多くのクリティカル パスが最適化により改善する可能性があります。配線後に `phys_opt_design` を実行すると、未接続だったネットが配線し直されます。このため、配線後に `phys_opt_design` を実行したら、`route_design` を明示的に実行する必要はありません。

チェックポイントを使用したさまざまな段階でのデザイン解析

Vivado Design Suite では、物理デザイン データベースに配置配線情報が保存されます。デザイン チェックポイント ファイル (`.dcp`) を使用すると、デザイン フローの主要な段階でこの物理データベースを保存 (`write_checkpoint` コマンド) および復元 (`read_checkpoint` コマンド) できます。チェックポイントは、フローの特定の時点におけるデザインのスナップショットです。プロジェクト モードでは、Vivado ツールでデザイン チェックポイント ファイルが自動的に生成され、インプリメンテーション `run` ディレクトリに格納されます。これらは、Vivado ツールの別のインスタンスとして開くことができます。

デザイン チェックポイント ファイルには、次のものが含まれます。

- インプリメンテーション中に適用された最適化を含む現在のネットリスト
- デザイン制約
- インプリメンテーション結果

チェックポイント デザインに対しては、`Tcl` コマンドを使用してデザイン フローの残りの段階を実行できます。新しいデザイン ソースを使用して変更することはできません。

チェックポイントで使用するコマンドの例は、次のとおりです。

- フローのある部分に戻り、さらに解析するため、結果を保存します。
- `-directive` オプションの異なる指示子を使用して `place_design` コマンドを実行し、それぞれのチェックポイントを保存します。このようにすると、次のインプリメンテーション段階でタイミング結果が最高の配置済みチェックポイントを選択できます。

チェックポイントの詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 ([UG904](#)) を参照してください。

インタラクティブ レポート ファイルの使用

チェックポイントを開くと、Vivado IDE で生成されたレポートを読み込んで解析できるようになります。レポートを生成するには、次のコマンドを `-rpx <filename.rpx>` オプションを指定して実行します。

```
report_timing_summary  
report_timing  
report_power  
report_methodology  
report_drc
```

チェックポイントを開いた後にインタラクティブ レポート ファイルを開くには、[Reports]→[Open Interactive Report] をクリックします。

注記: プロジェクト モードでは、自動的にインタラクティブ レポートが生成されて開きます。



推奨: レポートを生成する際、RPX ファイルのサイズに制限があるので、ザイリンクスでは、フローを停止するエラーが発生するのを回避するため、`catch` コマンドを使用することをお勧めします。たとえば、`catch {report_timing_summary -rpx timing_summary.rpx -file timing_summary.rpt}` のようになります。

インクリメンタル インプリメンテーション フローの使用

Vivado Design Suite では、インクリメンタル インプリメンテーションを使用して既存の配置配線を再利用することにより、インプリメンテーションのコンパイル時間を短縮し、予測可能な結果が得られるようにできます。95% 以上再利用されるデザインでは、インクリメンタル配置配線のコンパイル速度が通常の配置配線の 2 倍以上になり、基準 run の WNS も保持できます。詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』(UG904) のこの [セクション](#) を参照してください。



推奨: インクリメンタル インプリメンテーションは、フロー スクリプトに変更を加えるのが困難なデザイン サイクルのクリティカルな段階で最も有益です。デザイン サイクルの早期にフロー スクリプトにインクリメンタル インプリメンテーションを含め、クリティカルな段階でインクリメンタル インプリメンテーションをイネーブルにできるようにしてください。

注記: さらにコンパイル時間を短縮して QoR を向上するには、インクリメンタル合成も使用します。

関連情報

[インクリメンタル合成](#)

インクリメンタル インプリメンテーション フローのモード

自動インクリメンタル インプリメンテーション モード

自動インクリメンタル インプリメンテーション モードを使用すると、インクリメンタル インプリメンテーション フローがアクティブになりますが、Vivado ツールが基準チェックポイントと現在のデザインについて情報を取得するまでインクリメンタル インプリメンテーションは実行されません。`read_checkpoint` コマンドを実行すると、Vivado ツールにより、インプリメンテーション フローをデフォルト フロー アルゴリズムで実行するか、インクリメンタル フロー アルゴリズムで実行するかが判断されます。自動モードでは、インクリメンタル インプリメンテーションの基準デザイン データがツールで管理されるので、簡単に実行できます。

注記: 自動インクリメンタル インプリメンテーション モードは、デフォルトのインクリメンタル インプリメンテーション フローよりも控えめに実行され、インクリメンタル インプリメンテーション フローを実行した場合に QoR がより良く維持されます。

プロジェクト モード

プロジェクト モードでは、Vivado ツールにより使用するアルゴリズムとチェックポイントのアップデートが管理されます。プロジェクト モードで自動インクリメンタル インプリメンテーション モードをイネーブルにするには、[Design Runs] ウィンドウでインプリメンテーション run を右クリックし、[Set incremental Compile]→[Automatically use the checkpoint from the previous run] をクリックします。

これと同等の Tcl コマンドは、次のとおりです。

```
set_property AUTO_INCREMENTAL_CHECKPOINT 1 [get_runs <runName>]
```

非プロジェクト モード

非プロジェクト モードでは、Vivado ツールにより使用するアルゴリズムは管理されますが、チェックポイントをアップデートするかどうかはユーザーが判断する必要があります。非プロジェクト モードで自動インクリメンタル インプリメンテーション モードをイネーブルにするには、`-auto_incremental` オプションを使用します。次にコマンド例を示します。

```
read_checkpoint -incremental -auto_incremental <reference>.dcp
```

チェックポイントをアップデートする場合は、インプリメンテーション フローの最後に次のコマンドを使用して、WNS が許容される範囲を下回らないようにします。

```
if {[get_property SLACK [get_timing_path -setup]] > -0.250} {
    file copy -force <postroute>.dcp <reference>.dcp
}
```

高再利用モードと低再利用モード

自動インクリメンタル インプリメンテーション モードをイネーブルにせずにインクリメンタル インプリメンテーション フローを使用すると、次のいずれかの再利用モードが使用されます。

- 高再利用モード: 高再利用モードは、セルの再利用率が 75% 以上の場合にイネーブルになります。これがインクリメンタル インプリメンテーションの標準モードであり、インクリメンタル アルゴリズムが実行されます。
- 低再利用モード: 低再利用モードは、セルの再利用率が 75% 未満の場合にイネーブルになります。このモードではセルの配置は再利用されますが、デフォルト アルゴリズムが実行されます。このモードは、DSP、BRAM、または階層セルのブロック配置を再利用したい場合に有益です。

インクリメンタル指示子およびターゲット WNS

インクリメンタル指示子を使用して、インプリメンテーション フローのターゲット WNS を指定できます。ターゲット WNS は、インプリメンテーション ツールでタイミング クロージャを達成するよう試みるか、基準チェックポイントと同程度のタイミング クロージャを達成するかを決定します。インプリメンテーション フローでデフォルト アルゴリズムを使用する場合は、インクリメンタル指示子は無視され、`place_design` および `route_design` の指示子が使用されます。

次の表に、インクリメンタル指示子とそのターゲット WNS を示します。

表 5: インクリメンタル指示子とターゲット WNS

インクリメンタル指示子	ターゲット WNS
RuntimeOptimized:	基準チェックポイントと同じ
TimingClosure	0.000

表 5: インクリメンタル指示子とターゲット WNS (続き)

インクリメンタル指示子	ターゲット WNS
Quick	フローはタイミング ドリブンではなく、配置は関連ロジック ドリブン

注記: インクリメンタル指示子が以前のリリースの指示子マップに置き換わります。

並列実行

デフォルト フローでは、タイミングを満たす確率を上げるのに異なる配置指示子を使用した多数の run を並列実行するのが一般的です。インクリメンタル フローでは、指示子はタイミング クロージャを達成するように実行するか、タイミングを保持するように実行するかを指定します。さまざまな結果を得るには、必要なインクリメンタル指示子を異なるチェックポイントで使用してください。

コンパイル時間に関する考慮事項

自動インクリメンタル インプリメンテーション モードまたは高再利用モードで RuntimeOptimized 指示子を使用すると、デザインの再利用率が 95% 以上である場合はコンパイル時間が 1/2 になります。再利用率が下がれば、コンパイル時間の短縮量も下がります。変更がクリティカル パスに影響するものでなければ、コンパイル時間の削減は通常予測可能です。

自動インクリメンタル インプリメンテーション モードまたは高再利用モードで TimingClosure 指示子を使用すると、タイミング クロージャを達成するために追加のアルゴリズムが実行されます。このモードを使用すると、特にタイミング クロージャを達成するのが困難な場合、または密集エリアにタイミング エラーがある場合に、コンパイル時間が長くなることがあります。基準チェックポイントのタイミングが満たされている場合、コンパイル時間の短縮量は RuntimeOptimized 指示子を使用した場合に近くなります。

低再利用モードでは、コンパイル時間は予測できません。配置配線 run がタイミングを満たしそうになると、タイミングを満たすために Vivado ツールでのコンパイル時間が長くなることがあります。既存の配置配線が効率的に再利用されれば、Vivado ツールでのコンパイル時間は短縮されます。

合成済みデザインを開く

合成後は、まず最初に合成済みデザインからのネットリストをメモリに読み込み、デザイン制約を適用します。合成済みのデザインは、使用されるデザイン フローによってさまざまな方法で開くことができます。詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』(UG904)のこのセクションを参照してください。

ロジック最適化 (opt_design)

Vivado Design Suite ロジック最適化では、現在メモリに読み込まれているネットリストを最適化します。これが統合されたデザイン (RTL および IP ブロック) の最初の表示なので、デザインは通常さらに最適化できます。opt_design コマンドでは、デフォルトでロジックの自動削除、ロードのないセルの削除、定数入力の伝搬、およびブロック RAM の消費電力最適化が実行されます。また、複数の LUT を少ない数の LUT に統合してパスのロジック段数を削減する再マップなど、ほかの最適化も実行できます。

最適化の解析

`opt_design` コマンドを実行すると、各最適化フェーズの結果を詳細に示すメッセージが生成されます。最適化を実行したら、`report_utilization` コマンドを実行して使用率が向上したかを調べることができます。最適化の結果を解析しやすくするため、`opt_design` を `-verbose` および `-debug_log` オプションを使用して再実行し、各最適化がロジックにどのように影響したか、ユーザー制約により一部の最適化が実行されなかったかなどの詳細を表示します。詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』(UG904) のこのセクションおよびこのセクションを参照してください。

配置 (place_design)

Vivado Design Suite 配置エンジンによりネットリストのセルがターゲット ザイリンクス デバイスの特定サイトに配置されます。

配置解析

配置後にタイミング サマリ レポートを使用し、クリティカル パスをチェックします。

- ネガティブ セットアップ タイム スラックが大きいパスは、タイミング クロージャを達成するため、制約が適切で完全なものであるかをチェックしたり、ロジックの再構築が必要な場合があります。
- ネガティブ ホールド タイム スラックが大きいパスは、ほとんどの場合は正しくない制約またはクロッキング トポロジが原因であり、デザインを配線する前に修正する必要があります。
- ネガティブ ホールド タイム スラックが小さいパスは、たいていは配線で修正されます。
`report_clock_utilization` 後に `place_design` を実行して、クロック領域ごとのクロック リソースおよびロード数をレポートすることもできます。

配置の詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』(UG904) のこのセクションを参照してください。

関連情報

[タイミング クロージャ](#)

物理最適化 (phys_opt_design)

物理最適化は、フローのオプションの段階で、デザインの負のスラック パスに対してタイミング ドリブンの最適化を実行します。複製、リタイミング、ホールドの修正、および配置の向上が実行されます。物理最適化では、ネットリストおよび配置の必要な変更が自動的に実行されるので、`place_design` の後に `phys_opt_design` を実行する必要はありません。

物理合成の必要性

デザインに物理合成が有益であるかを判断するには、配置後のタイミングを評価します。タイミングが満たされていないパスでファンアウトを解析します。ファンアウトの大きいクリティカル パスは、ファンアウト最適化で向上する可能性があります。また、複数のブロック RAM を含む大型 RAM ブロックのファンアウトの大きいデータ、アドレス、および制御ネットで `route_design` 後にタイミングが満たされない場合、ネットの複製を強制すると有益であることがあります。物理合成の詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』(UG904) のこのセクションを参照してください。

配線 (route_design)

Vivado Design Suite 配線では、配置済みデザインに配線を実行し、ホールド タイム違反を解決するように配線済みデザインが最適化されます。デフォルトでは、密集を緩和しながら、実行時間とデザイン パフォーマンスのバランスを取るように配線の最適化が実行されます。配線の指示子によっては、デザイン パフォーマンスを向上し、より積極的に密集を削減するため、実行時間が長くなるものもあります。配線の詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 (UG904) のこのセクションを参照してください。

配線解析

最適に配線されていないネットがある場合、問題は多くの場合間違ったタイミング制約です。配線設定を変更して、前に、制約が正しく設定されていることを確認してください。配線の前の配置済みデザインのタイミング レポートを参照して、タイミングと制約を確認します。

不適切なタイミング制約の例として、ホールド タイミングで配線遅延が追加されるクロック乗せ換えパスや正しくないマルチサイクル パスなどがあります。密集したエリアは、RTL 合成でのファンアウトの最適化または物理最適化により解決できます。すべてまたは一部のデザイン階層を保持し、境界をまたぐ最適化が実行されないようにし、ネットリストの集積度を削減できます。または、フロアプラン制約を使用して密集を緩和できます。

関連情報

[タイミング クロージャ](#)

配線コンパイル時間

`route_design -ultrathreads` オプションを使用すると、結果の再現性は低下しますが、配線のコンパイル時間を短縮できます。このオプションにより、配線に複数のスレッドを実行する自由度が追加され、配線は高速になりますが、結果は毎回多少異なります。同一の run のスラックの差は 1% 未満ですが、コンパイル時間は大幅に削減されます。結果が厳密に再現可能である必要がない場合に、配線コンパイル時間を短縮するためこのオプションを使用することを考慮してください。

デザイン クロージャ

デザイン クロージャは、タイミング、システム パフォーマンス、および消費電力要件を満たすことと、ハードウェアで機能を検証する作業を含みます。デザイン クロージャでは通常、結果の解析、デザインの変更、および制約の変更を複数回繰り返すことが必要です。デザイン クロージャは、タイミングと消費電力最適化のトレードオフとみなされることがよくありますが、タイミングを向上する最適化には消費電力にも効果的であるものも多数あります。たとえば、セル間の距離を短縮する配置は、伝搬遅延とインターコネクト消費電力の両方を削減します。

典型的なミスは、まずタイミング クロージャだけに焦点を置き、タイミング クロージャが達成されてから消費電力最適化を始めることです。ほとんどの消費電力最適化はタイミングに悪影響を及ぼしませんが、論理ネットリストにバリエーションが発生します。タイミングがぎりぎりでも満たされている場合、消費電力最適化により変更が加えられたときにタイミング クロージャを保てない可能性があります。ザイリンクスでは、消費電力最適化をデザイン サイクルの初期段階から組み込み、タイミング クロージャと消費電力クロージャの間の反復を回避することをお勧めします。

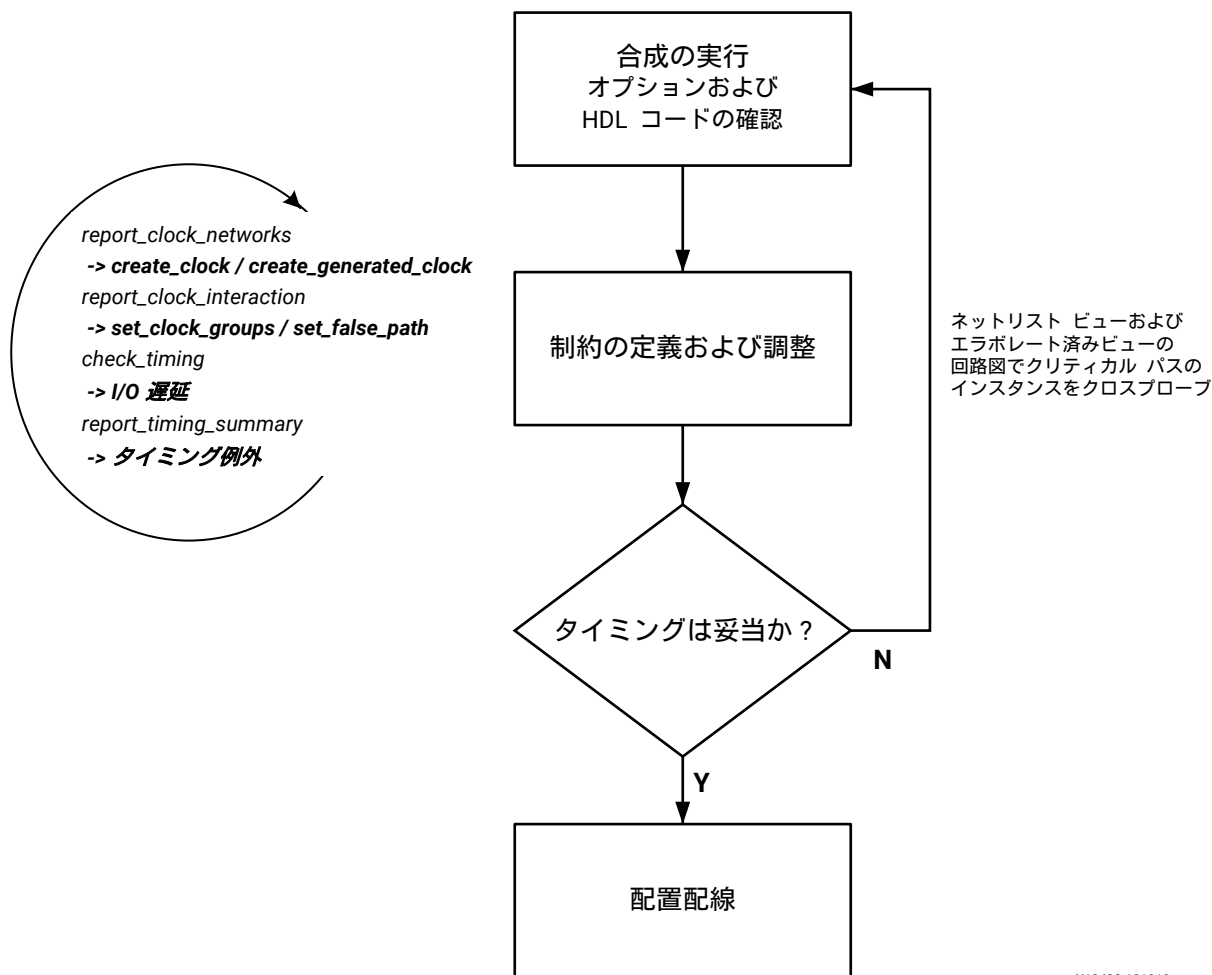


ヒント: 初期デザイン チェック、デザインのベースライン制約の作成、タイミング違反の解決など、この章で説明されている手法がコンパクトにまとめられている『UltraFast 設計手法タイミング クロージャクイック リファレンス ガイド』([UG1292](#))を参照してください。

タイミング クロージャ

タイミング クロージャでは、デザインがすべてのタイミング要件を満たすことが必要です。合成で使用する HDL および制約が適切なものであれば、タイミング クロージャを達成しやすくなります。さらに、次の図に示すように、改善した HDL、制約、合成オプションを使用して合成を再実行することが重要です。

図 108: タイミング クロージャを短期間で達成するための設計手法



X13422-121919

タイミング クロージャを達成するには、次の一般的なガイドラインに従います。

- 初期段階でタイミングが満たされない場合、フローを通してタイミングを評価します。
- トータル ネガティブ スラック (TNS) を向上する主な方法として、各クロックのワースト ネガティブ スラック (WNS) に焦点を置きます。
- ワースト ホールド スラック (WHS) 違反が大きいもの (< -1 ns) を見直し、不足している制約または不適切な制約を特定します。
- デザインでの選択、制約、およびターゲット アーキテクチャでのトレードオフを再度考慮します。
- ツール オプションおよびザイリンクス デザイン制約 (XDC) の使用方法を理解します。
- タイミングが満たされると、ツールではそれ以上のタイミングの向上 (マージンの増加) は試みられないということに注意してください。

次のセクションに、設計手法デザイン ルール チェック (DRC)、ベースライン制約の作成、タイミング違反の根本的な原因の特定、一般的な手法を使用した違反の解決により、タイミング制約が完全で正しいものであることを確認するための推奨事項を示します。

タイミング クロージャの理解

タイミング クロージャは、デザインがハードウェアでどのように動作するかを表す有効な制約を記述するところから始まります。次のセクションで説明されているように、タイミング サマリ レポートを確認します。

制約が有効であることの確認



電源/消費電力ヒント: 適切なタイミング制約が指定されたデザイン run がある場合は、Vivado® IDE で [Create Runs] コマンドを使用して複数のストラテジを作成して試みます。各デザインに正確なスイッチング アクティビティ XDC 制約を使用して `report_power` コマンドを実行し、タイミングおよび消費電力の両方がベストな run を見つけます。

タイミング サマリ レポートの [Check Timing] セクションで、次を含むタイミング制約の適用範囲を確認します。

- すべてのアクティブなクロック ピンにクロック定義が適用されている。
- すべてのアクティブ パスの終点に、定義済みクロックに対する要件 (セットアップ/ホールド/リカバリ/リムーバル) がある。
- すべてのアクティブな 入力ポートに 入力遅延制約が設定されている。
- すべてのアクティブな 出力ポートに 出力遅延制約が設定されている。
- タイミング例外が正しく指定されている。



注意: 制約にワイルドカードを過剰に使用すると、実際に適用される制約が意図したものと異なるものになることがあります。 `report_exceptions` コマンドを使用して、タイミング例外の競合を特定し、各タイミング例外が適用されるネットリスト オブジェクト、タイミング クロック、タイミング パスを確認します。

`check_timing` に加え、設計手法レポート (TIMING および XDC チェック) で不正確なタイミング解析およびハードウェアの誤動作の原因となり得るタイミング制約が特定されます。レポートされたすべての問題を注意深く調べ、解決する必要があります。

注記: デザインのベースライン制約を作成する際、すべてのザイリンクス IP 制約を使用する必要があります。ユーザー I/O 制約を指定してユーザー I/O 制約がないために `check_timing` および `report_methodology` で生成される違反を無視しないでください。

正のタイミング スラックの確認

次のタイミング メトリクスはタイミング違反を示します。タイミングを満たすためには、これらの数値が正である必要があります。

- セットアップ/リカバリ (最大遅延解析): $WNS > 0 \text{ ns}$ および $TNS = 0 \text{ ns}$
- ホールド/リムーバル (最小遅延解析): $WHS > 0 \text{ ns}$ および $THS = 0 \text{ ns}$
- パルス幅: $WPWS > 0 \text{ ns}$ および $TPWS = 0 \text{ ns}$

タイミング レポートの理解

タイミング サマリ レポートには、設定されている制約と比較したデザインのタイミング特性に関する情報が示されます。デザインのサインオフでは、タイミング サマリ の次の値を確認します。

- TNS (トータル ネガティブ スラック): デザイン全体または特定のクロック ドメインに含まれる各終点のセットアップ/リカバリ違反の合計。ワースト セットアップ/リカバリ スラックは、WNS (ワースト ネガティブ スラック) です。

- THS (トータル ホールド スラック): デザイン全体または特定のクロック ドメインに含まれる各終点のホールド/リムーバル違反の合計。ワースト ホールド/リムーバル スラックは、WHS (ワースト ホールド スラック) です。
- TPWS (トータル パルス幅スラック): デザイン全体または特定のクロック ドメインに含まれる各クロック ピンの、次のチェックにおける違反の合計。
 - 最小 Low パルス幅
 - 最小 High パルス幅
 - 最小周期
 - 最大周期
 - 最大スキュー (同じ最下位セルの 2 つのクロック ピン間)
- WPWS (ワースト パルス幅スラック): クロック ピンのパルス幅、周期、またはスキュー チェックすべてのワースト スラック。

トータル スラック (TNS、THS、または TPWS) は、デザインの違反のみを反映します。すべてのタイミング チェックが満たされると、トータル スラックはヌルになります。

タイミング パス レポートには、各タイミング チェックで論理パスのスラックがどのように算出されたかの詳細情報も示されます。完全に制約されたデザインでは、各パスに 1 つまたは複数の要件があり、関連のロジックが正しく機能するためにはそれらの要件がすべて満たされる必要があります。

WNS、TNS、WHS、および THS で示される主なチェックは、シーケンシャル セルの機能要件から求められます。

- セットアップ タイム: 新しいデータが正しく取り込まれるために、次のアクティブ クロック エッジの前に新しいデータが安定していなければならない時間
- ホールド要件: 無効な値が取り込まれないように、アクティブ クロック エッジ後にデータが安定したままでいなければならない時間。
- リカバリ タイム: 非同期リセット信号が非アクティブ ステートにトグルされてから次のアクティブ クロック エッジまでに必要な最小時間。
- リムーバル タイム: 非同期リセット信号を問題なく非アクティブ ステートにトグルできる、アクティブ クロック エッジからの最小時間。

同じクロック ネットに接続された 2 つのフリップフロップ間のパスが、その単純な例です。

クロック ネットにタイミング クロックを定義すると、タイミング解析により、デスティネーション フリップフロップのデータ ピンで最悪だが妥当な動作条件におけるセットアップ チェックとホールド チェックのが実行されます。セットアップおよびホールド スラックの両方が正の場合、ソース フリップフロップからデスティネーション フリップフロップへのデータ転送が正しく実行されます。

タイミング解析の詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906) の[タイミング解析の基礎的理解](#)を参照してください。

デザインが適切に制約されているかを確認

タイミング結果を見て違反がないかどうかを確認する前に、デザインのすべての同期終点が適切に制約されていることを確認してください。

`check_timing` を実行して制約が適用されていないパスを特定します。このコマンドはスタンドアロンで実行できますが、`report_timing_summary` の一部としても実行されます。また、`report_timing_summary` には [Unconstrained Paths] セクションが含まれ、タイミング要件のない N 個論理パスが定義済みのソースまたはデステーション タイミング クロックごとにリストされます。N は `-max_path` オプションで指定されます。

デザインが完全に制約されたら、`report_methodology` コマンドを実行し、TIMING および XDC チェックで最適でない制約を特定します。最適でない制約により、タイミング解析が正確なものではなく、ハードウェアでのタイミング マージンが変動する可能性があります。

check_timing で検出された問題の修正

`check_timing Tcl` コマンドは、タイミング定義に不足しているものがあるか、間違っているものがあるかをレポートします。`check_timing` コマンドでレポートされた問題を確認および修正する際は、最も重要なチェックから始めます。次に、チェックを重要なものから順にリストします。

no_clock および unconstrained_internal_endpoints

これらのチェックでは、デザインの内部パスが完全に制約されているかを確認できます。スタティック タイミング解析のサインオフのクオリティ確認においては、`unconstrained_internal_endpoints` が 0 である必要があります。

これは、すべての内部パスにタイミング解析用の制約が適用されていることを示しているだけで、制約の値が正しいことを示しているわけではありません。

generated_clocks

デザインで生成クロックが使用されるのは通常のことですが、生成クロックが同じクロック ツリーにないマスタークロック ソースを基準としている場合、重大な問題となる可能性があります。タイミング エンジンで生成クロック ツリーの遅延を適切に算出できないことがあり、スラックの算出が不正確になります。ワースト ケースの状況では、レポートにタイミングが満たされていると示されていても、デザインがハードウェアで機能しません。

loops および latch_loops

タイミング ループはタイミング エンジンにより分割されるので、理想的なデザインには組み合わせループは含まれません。分割されたパスは、タイミング解析ではレポートされず、インプリメンテーション中に評価されません。これにより、全体的なタイミング要件は満たされていても、ハードウェアでの動作が正しいものではない可能性があります。

no_input_delay、no_output_delay、partial_input_delay、partial_output_delay

すべての I/O ポートが正しく制約されている必要があります。



推奨: ベースライン制約を検証してから I/O タイミングの制約を確認します。

multiple_clock

複数のクロックは、通常は許容されます。ザイリンクスでは、これらのクロックが同じクロック ツリーに伝搬されることを確認することをお勧めします。また、これらのクロック間のパス要件により、デザインがハードウェアで機能するために必要な要件より厳しい要件が適用されていないかも確認する必要があります。

この場合、これらのパスのクロック間に `set_clock_groups` または `set_false_path` を使用します。タイミング例外を使用する場合は、指定のパスのみに適用されていることを確認してください。



重要: XDC は Tcl 構文およびセマンティクス規則に従うので、制約の順序が関係します。詳細は、『Vivado Design Suite ユーザー ガイド: 制約の使用』(UG903)を参照してください。

report_methodology で検出された問題の修正

report_methodology コマンドは、追加の制約およびタイミング解析問題をレポートします。これらの問題は、配置配線の実行前後に注意深く確認する必要があります。このセクションでは、主な XDC および TIMING カテゴリと、それらのタイミング クロージャおよびハードウェアの安定性への相対的な影響について説明します。まず、タイミング クロージャに影響するチェックを確認して解決する必要があります。

次のチェックの詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906)の[タイミング設計手法チェック](#)を参照してください。

タイミング クロージャに影響する設計手法 DRC

次の表に示す DRC は、インプリメンテーション ツールへの負荷を増加し、タイミング クロージャが不可能になったり、一貫しなくなる可能性のあるデザインとタイミング制約の組み合わせをレポートします。これらの DRC は通常、不足しているクロック乗せ換え (CDC) 制約、不適切なクロック ツリー、またはロジックの複製によりタイミング例外の適用範囲が一貫しない状況を示します。これらの問題は、優先的に解決する必要があります。



重要: タイミング チェックのクリティカル警告を注意深く検証してください。

表 6: タイミング クロージャ設計手法 DRC

チェック	重要度	説明
TIMING-6	クリティカル警告	関連クロック間に共通クロックがない
TIMING-7	クリティカル警告	関連クロック間に共通ノードがない
TIMING-8	クリティカル警告	関連クロック間に共通周期がない
TIMING-14	クリティカル警告	クロック ツリーに LUT がある
TIMING-15	警告	クロック間のパスに大きなホールド違反がある
TIMING-16	警告	大きいセットアップ違反がある
TIMING-30	警告	生成クロックに対して最適でないマスター ソース ピンが選択されている
TIMING-31	クリティカル警告	位相シフトされたクロック間に不適切なマルチサイクル パスがある
TIMING-32、TIMING-33、 TIMING-34、TIMING-37、 TIMING-38、TIMING-39	警告	推奨されないバス スキュー制約がある
TIMING-36	クリティカル警告	生成クロックに対してマスター クロック エッジが伝搬されない
TIMING-42	警告	パス分割によりクロックが伝搬されない
TIMING-44 TIMING-45	警告	ユーザーのクロック内およびクロック間のばらつきが妥当でない
TIMING-48	アドバイザリ	ラッチ入力に set_max_delay -datapath_only 制約が適用されている
TIMING-49	クリティカル警告	並列 BUFCE_DIV から危険なイネーブルまたはリセット トポロジ
TIMING-50	警告	同レベルのラッチ間のパス要件が妥当でない
XDCB-3	警告	同じ set_clock_groups コマンドで複数のグループに同じクロックが使用されている
XDCH-1	警告	マルチサイクル パス制約にホールド オプションがない

表 6: タイミング クロージャ 設計手法 DRC (続き)

チェック	重要度	説明
XDCV-1	警告	複製で使用する元のオブジェクトがないために制約の適用範囲が不完全
XDCV-2	警告	複製されたオブジェクトがないために制約の適用範囲が不完全

サインオフのクオリティに影響する設計手法 DRC

次の表に示す DRC は通常、タイミング クロージャの達成しやすさに影響する問題をレポートするのではなく、推奨されない制約のためにタイミング解析の正確さに影響する問題をレポートします。セットアップ スラックおよびホールド スラックが正であっても、ハードウェアがすべての動作条件下で適切に機能するとはかぎりません。ほとんどのチェックは、デザインの境界で定義されていないクロック、予期しない波形を持つクロック、不足しているタイミング要件、または不適切な CDC 回路に関するものです。この最後のカテゴリでは、`report_cdc` コマンドを使用し、より包括的な解析を実行します。



重要: タイミング チェックのクリティカル警告を注意深く検証してください。

表 7: サインオフ クオリティ 設計手法 DRC

チェック	重要度	説明
TIMING-1、TIMING-2、TIMING-3、TIMING-4、TIMING-27	クリティカル警告	推奨されないクロック起点が定義されている
TIMING-5、TIMING-25、TIMING-19	クリティカル警告	予期しないクロック波形
TIMING-9、TIMING-10	警告	不明な CDC ロジック
TIMING-11	警告	不適切な <code>set_max_delay -datapath_only</code> コマンド
TIMING-12	警告	CRPR (Clock Reconvergence Pessimism Removal) がディスエーブル
TIMING-13、TIMING-23	警告	分割されたパスのためタイミング解析が不完全
TIMING-17	クリティカル警告	シーケンシャル セルにクロックが供給されていない
TIMING-18、TIMING-20、TIMING-26	警告	クロックまたは入力/出力遅延制約が不足している
TIMING-21、TIMING-22	警告	MMCM 補正に関する問題
TIMING-24	警告	無効になった <code>set_max_delay -datapath_only</code> コマンド
TIMING-29	警告	マルチサイクル パスのペアに一貫性がない
TIMING-35	クリティカル警告	同じクロックのパスに共通ノードなし
TIMING-40、TIMING-43	警告	クロッキング トポロジまたは要件が不適切な
TIMING-41	警告	内部ピンに無効なフォワード クロックが定義されている
TIMING-46	警告	マルチサイクル パスの CE ピンが接続されている
TIMING-47	警告	同期クロック間にフォルス パスまたは非同期クロック グループがある
TIMING-51	クリティカル警告	並列 MMCM または PLL からの関連クロック間に共通位相がない
TIMING-52	クリティカル警告	スペクトラム 拡散 MMCM からの関連クロック間に共通位相がない

その他のタイミング設計手法 DRC

その他の TIMING および XDC チェックは、実行時間が長くなる制約、既存の制約を無効にする制約、またはネットリスト名の変更の影響を受けやすい制約を特定します。対応する情報は、制約の競合をデバッグする際に有益です。TIMING-28 チェック (自動派生クロックがタイミング制約で参照されている) に特に注意してください。自動派生クロックの名前は、デザイン ソース コードを変更して再合成すると変更される可能性があります。この場合、以前に定義された制約が機能しなくなるか、間違ったタイミング パスに適用される可能性があります。

デザインの最大周波数の評価

最大周波数 (FMAX) の定義および評価には、次の方法を使用できます。

- 指定のインプリメンテーションでデザインがハードウェア上で実行可能な $F_{MAX} \text{ (MHz)} = 1000 / (T - WNS)$ (WNS は正または負)。
- 指定のアーキテクチャでデザインが実行可能な $F_{MAX} \text{ (MHz)} = 1000 / (T - WNS)$ (WNS は 0 未満の場合のみ)。WNS が 0 未満になるまで、T を削減して合成またはインプリメンテーションを再実行する必要があります。達成可能な最高の FMAX を得るには、合成およびインプリメンテーションの異なる戦略が必要です。

説明:

- T: ターゲット クロック周期 (ns)。
- WNS: ターゲット クロックのワースト ネガティブ スラック (ns)。

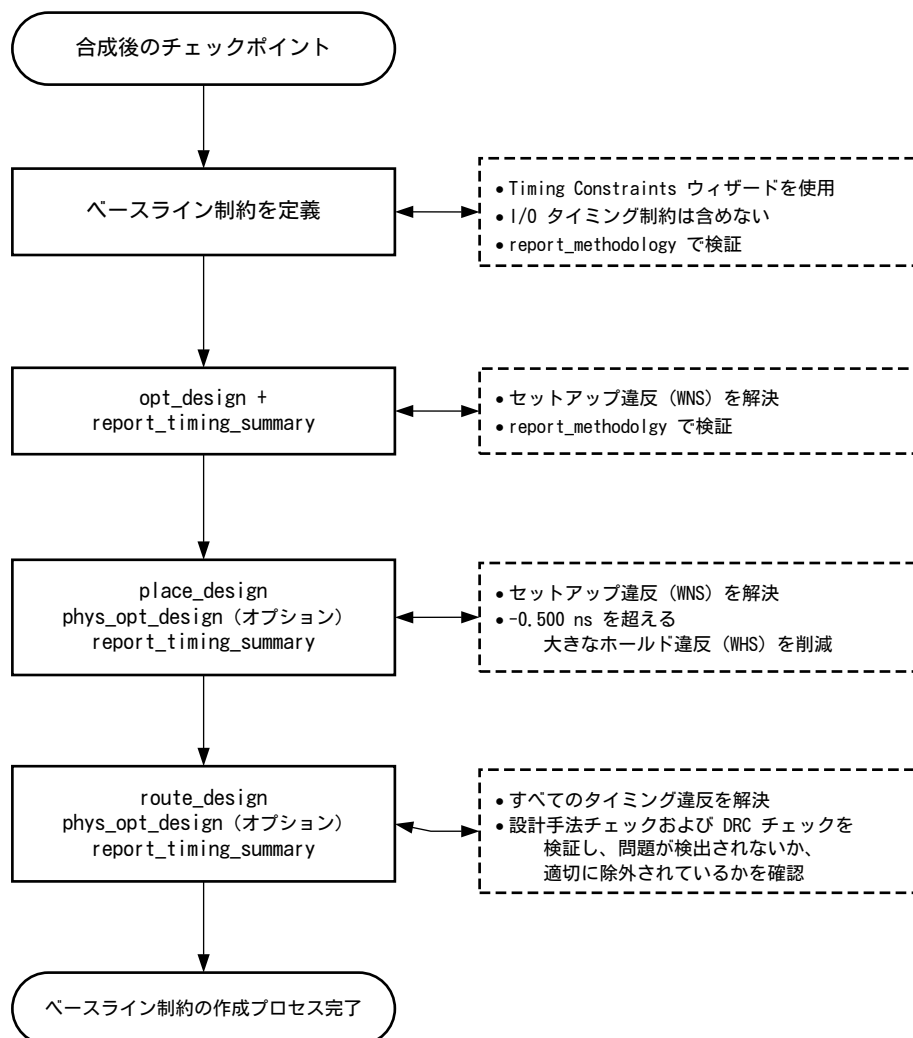
注記: FMAX 値は、`report_timing` または `report_timing_summary` レポートには明示的には示されません。

デザインのベースライン制約の作成

ベースライン制約の作成では、最も単純なタイミング制約のセットを作成し、I/O タイミングは無視します。すべてのクロックを完全に制約すると、始点と終点がデザイン内に含まれるすべてのパスは自動的に制約されます。これにより、デザインが変更中でも、内部デバイスのタイミングの課題を特定できます。デザインにはクロック乗せ換えがある可能性があるため、ベースライン制約には生成クロックを含む指定のクロック間の関係を含める必要があります。

デザインのベースライン制約を作成する際は、フローを通してタイミング問題を解析して解決し、各インプリメンテーション段階でタイミングが満たされるようにする必要があります。まず、Vivado® インプリメンテーション ツールでの現実的なタイミングを取得するため、単純で有効な制約を作成します。そして、さまざまなインプリメンテーション段階を実行していくときには、次の段階に進む前にタイミング違反を解決するようにします。次の図に、ベースライン制約の作成プロセスを示します。

図 109: デザインのベースライン制約の作成



X20037-021821

ベースライン制約を作成したら、次を実行できます。

- 小さいタイミング違反を除去します。
- 制約がデザイン全体に適用されるようにします。
- 新しいモジュールを最上位デザインに追加する前に、個別のベースライン制約を作成します。



推奨: サイリンクスでは、デザイン プロセスの初期にベースライン制約を作成し、デザイン HDL への主要な変更をこれらのベースライン制約に対してプランニングすることをお勧めします。

ベースライン制約の定義

最も単純な制約のセットを作成するには、ユーザー タイミング制約を含まない有効な合成後の Vivado チェックポイントを使用します。チェックポイントを開き、Timing Constraints ウィザードを使用して制約を定義します。ウィザードを使用すると、体系的に制約を作成していくことができます。

この段階では、すべての制約を定義する必要はありません。デフォルトでは、Vivado ツールでは I/O タイミングは制約が設定されていなければ無視されます。そのため、ベースライン制約にはこの段階で I/O タイミング制約を定義する必要はありません。ベースライン プロセスが完了した後に、フローの後の方で I/O タイミング制約を定義します。



ヒント: Timing Constraints ウィザードを使用する場合は、推奨される I/O タイミング制約をオフにしてください。

デバイスの内部タイミングを正確に把握するには、次の制約を定義します。

- すべてのクロック制約
- クロック乗せ換え (CDC) 制約

同期クロック間の CDC パスはデフォルトで安全にタイミング解析が実行されますが、非同期クロック間には安全な CDC 回路を使用して、タイミング例外を指定する必要があります。

制約を作成した後、タイミングを満たすことができないパスを特定します。対応する RTL を記述し直すか、クロック周期を緩和します。



重要: ザイリンクス IP およびパートナー IP には、ザイリンクス制約手法に従った XDC 制約が提供されます。IP 制約は、合成とインプリメンテーションに自動的に含まれます。ベースライン制約を作成する際に、IP 制約を変更しないでください。

制約を定義するのに Timing Constraints ウィザードを使用しない場合は、次のセクション説明するベースライン制約を手動で定義するために必要な手順を参照してください。

作成する必要があるクロックの特定

合成後のネットリストまたはチェックポイントを Vivado IDE に読み込みます。Tcl コンソールで `reset_timing` コマンドを使用し、すべてのタイミング制約を削除します。

`report_clock_networks` Tcl コマンドを使用して、デザインで定義する必要があるプライマリ クロックをすべてリストします。このクロック ネットワークのリストにより、作成する必要があるクロック制約がわかります。

[Timing Constraints Editor] ウィンドウを使用して、各クロックのパラメーターを適切に指定します。

不足しているクロックがないことの確認

クロック ネットワーク レポートにすべてのクロック ネットワークに制約が設定されたことが示されたら、生成クロックが正しいことを確認します。Vivado ツールでは、クロック制約はクロック調整ブロックを介して自動的に伝搬されるので、生成された制約を確認することが重要です。`report_clocks` を使用して、どのクロックが `create_clock` 制約で生成され、どのクロックが自動生成されたかを確認します。

注記: MMCM、PLL、およびクロック バッファはクロック調整ブロックです。UltraScale™ デバイスでは、GT もクロック調整ブロックです。

`report_clocks` の結果にすべてのクロックが伝搬されていることが示されます。`create_clock` で作成されるプライマリ クロックと生成クロックの違いは、[Attributes] 列に示されます。

- 伝搬 (P) とのみ示されるクロックは、プライマリ クロックです。
- ほかのクロックから派生されたクロックは、伝搬 (P) および生成 (G) の両方として示されます。
- クロック調整ブロックにより生成されたクロックは、自動派生 (A) と示されます。
- その他の属性は、自動派生クロックの名前が変更されている (R)、生成クロックの波形が入力マスター クロックから反転されている (I)、プライマリ クロックが仮想クロックである (V) ことを示します。

生成クロックは、`create_generated_clock` 制約を使用しても作成できます。詳細は、『Vivado Design Suite ユーザー ガイド: 制約の使用』 (UG903) を参照してください。

図 110: `report_clocks` により示されるプライマリ クロックから生成されたクロック

Attributes				
P: Propagated				
G: Generated				
A: Auto-derived				
R: Renamed				
V: Virtual				
I: Inverted				
Clock	Period(ns)	Waveform(ns)	Attributes	Sources
sysClk	10.000	{0.000 5.000}	P	{sysClk}
clkfbout	10.000	{0.000 5.000}	P,G,A	{clkgen/mmcm_adv_inst/CLKFBOUT}
cpuClk	20.000	{0.000 10.000}	P,G,A,R	{clkgen/mmcm_adv_inst/CLKOUT0}
wbClk_4	20.000	{0.000 10.000}	P,G,A	{clkgen/mmcm_adv_inst/CLKOUT1}
usbClk_3	10.000	{0.000 5.000}	P,G,A	{clkgen/mmcm_adv_inst/CLKOUT2}
phyClk0_2	10.000	{0.000 5.000}	P,G,A	{clkgen/mmcm_adv_inst/CLKOUT3}
phyClk1_1	10.000	{0.000 5.000}	P,G,A	{clkgen/mmcm_adv_inst/CLKOUT4}
fftClk_0	10.000	{0.000 5.000}	P,G,A	{clkgen/mmcm_adv_inst/CLKOUT5}



ヒント: デザインに制約が適用されていない終点がないことを調べるには、チェック タイミング レポート (`no_clock` カテゴリ) を確認します。このレポートは、タイミング サマリ レポートから、または `check_timing` Tcl コマンドを使用して取得できます。

クロック乗せ換えの制約

クロック制約を確認したら、非同期および制約が厳しすぎるクロック乗せ換えパスを特定する必要があります。

注記: このセクションではクロック領域の境界を適切にまたぐ方法は説明しませんが、存在するクロック乗せ換えパスを特定し、制約する方法を示します。

クロック関連性の確認

クロック間の関係は、`report_clock_interaction` Tcl コマンドを使用して確認できます。このレポートには、ソース クロックとデスティネーション クロックのマトリックスが示されます。各セルの色は、クロック間の関連性を既存の制約を含めて示します。次の図に、クロック関連性レポートの例を示します。

図 111: クロック関連性レポートの例



次の表に、このレポートの各色の説明を示します。

表 8: report_clock_interaction の色の説明

色	ラベル	説明	次の操作
黒	No path	クロック ドメイン間に関連性はありません。	これらのクロック ドメインに関連性があるはずの場合以外は、特に操作は必要ありません。
緑	Timed	クロック ドメイン間に関連性があり、パスにタイミング制約が適用されています。	これらのクロック ドメインに関連性がないはずの場合以外は、特に操作は必要ありません。
シアン	Partial False Path	関連性のあるクロック ドメイン間の一部にユーザー例外が設定されており、タイミングが適用されていません。	タイミング例外が適切なものであることを確認します。
赤	Timed (unsafe)	クロック ドメイン間に関連性があり、パスにタイミング制約が適用されていますが、クロックが独立 (非同期) であるように見えます。	これらのクロックを非同期として定義する必要があるか、共通のプライマリソースを共有する必要があるかを確認します。
オレンジ	Partial False Path (unsafe)	クロック ドメイン間に関連性があります。クロックは独立 (非同期) であるように見えますが、タイミング例外が設定されているためにタイミングが適用されないのは一部のパスのみです。	一部のパスにタイミング例外が設定されないのはなぜかを確認します。
青	User Ignored Paths	クロック ドメイン間に関連性がありますが、クロックグループまたはフォールス パス タイミング例外が設定されているためにタイミング解析は実行されません。	これらのクロックが非同期であるかを確認します。また、対応する HDL コードが正しく記述されているかを確認し、クロック ドメイン間が正しく同期され、データが適切に転送されるようにします。
水色	Max Delay Datapath Only	クロック ドメイン間に関連性があり、set_max_delay -datapath_only によりパスにタイミング制約が適用されています。	クロックが非同期であり、指定された遅延が正しいことを確認します。

フォルス パス制約またはクロック グループ制約を作成する前に、マトリックスに表示される色が黒、赤、緑のみにするようにしてください。すべてのクロックにはデフォルトでタイミング関係が適用されるので、非同期クロックを分離することは重要です。非同期クロックを分離しないと、デザインが過剰に制約される可能性があります。

共通のプライマリ クロックを持たないクロック ペアの特定

クロック関連性レポートは、関連性のあるクロックのプライマリ クロック ソースが共通であることを示します。プライマリ クロックが共通でないクロック ペアは、多くの場合非同期です。そのため、レポートで [Common Primary Clock] 列で並べ替えて、これらのペアを特定すると有益です。このレポートでは、クロック ドメイン間をまたぐパスが適切に設計されているかどうかは判断できません。

`report_cdc` Tcl コマンドを使用して、非同期クロック間のクロック乗せ換え回路の包括的な解析を実行します。`report_cdc` コマンドの詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906) の [クロック乗せ換えレポート](#) を参照してください。また、『Vivado Design Suite Tcl コマンド リファレンス ガイド』 (UG835) の `report_cdc` も参照してください。

厳しいタイミング要件の特定

クロック関連性レポートには、各クロック ペアに対して、ワースト パスのセットアップ要件も示されます。表を [Path Req (WNS)] で並べ替え、デザインで最も厳しい要件を確認します。これらの要件を見直して、無効な厳しい要件が存在していないことを確認します。

Vivado ツールは、各クロックを 1000 サイクル調べ、最も少ないサイクルでエッジが揃う箇所を特定します。最も厳しい要件を判断するのに 1000 サイクルでは十分でない場合は、レポートに「Not Expanded」と示され、2 つのクロックを非同期として処理する必要があります。

たとえば、250 MHz クロックから 200 MHz クロックに切り替わるタイミング パがあるとしてします。

- 200 MHz クロックの立ち上がりエッジは {0, 5, 10, 15, 20} です。
- 250 MHz クロックの立ち上がりエッジは {0, 4, 8, 12, 16, 20} です。

このクロック ペアで要件が最も厳しいのは、次の 2 つの条件が満たされるときです。

- 250 MHz クロックの立ち上がりエッジ 4 ns。
- 200 MHz クロックの次の立ち上がりエッジが 5 ns。

これにより、250 MHz クロック ドメインから 200 MHz クロック ドメインに切り替わるすべてのパスに 1 ns のタイミングが適用されます。

注記: この例では、デスティネーション エッジとソース エッジが同じであることではないので、エッジが揃う 20 ns は最も厳しい要件とはなりません。

これは厳しいタイミング要件なので、追加の手順が必要となります。デザインによって、次のいずれかの制約を使用すると、これらのタイミング ドメイン間をまたぐ状況を適切に処理できます。

- `set_clock_groups/set_false_path/set_max_delay -datapath_only`

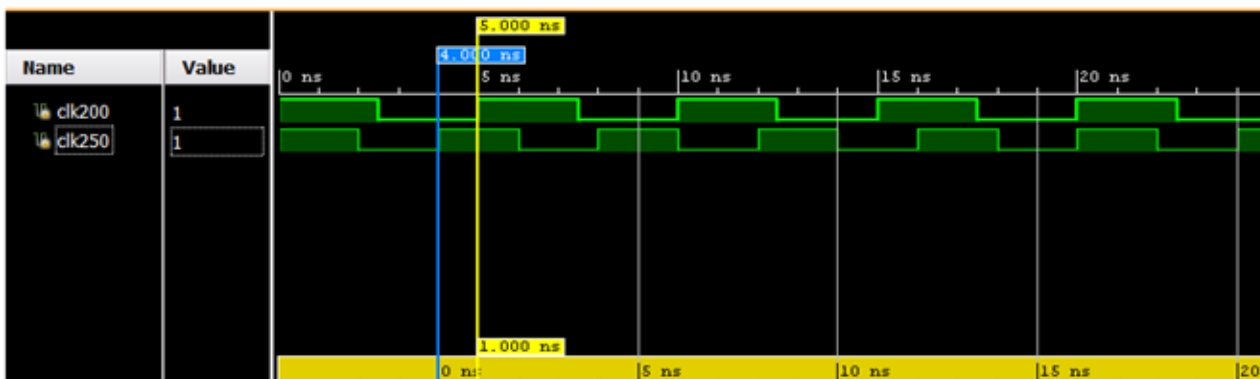
クロック ペアを非同期とする場合は、上記のいずれかの制約を使用します。`report_cdc` Tcl コマンドを使用して、クロック乗せ換え回路が安全であるかどうかを検証します。

- `set_multicycle_path`

この制約は、適切なクロック回路によりソース クロック エッジとデスティネーション クロック エッジが制御されている場合に、タイミング要件を緩和する場合に使用します。

何もしない場合、これらのクロック ドメイン間をまたぐパスでタイミング違反が発生したり、最適化および配置配線が、デザインの実際のクリティカル パスではなく、これらのパスに焦点を置いて実行される可能性があります。このようなパスをタイミング ドリブンのインプリメンテーションの前に特定することが重要です。

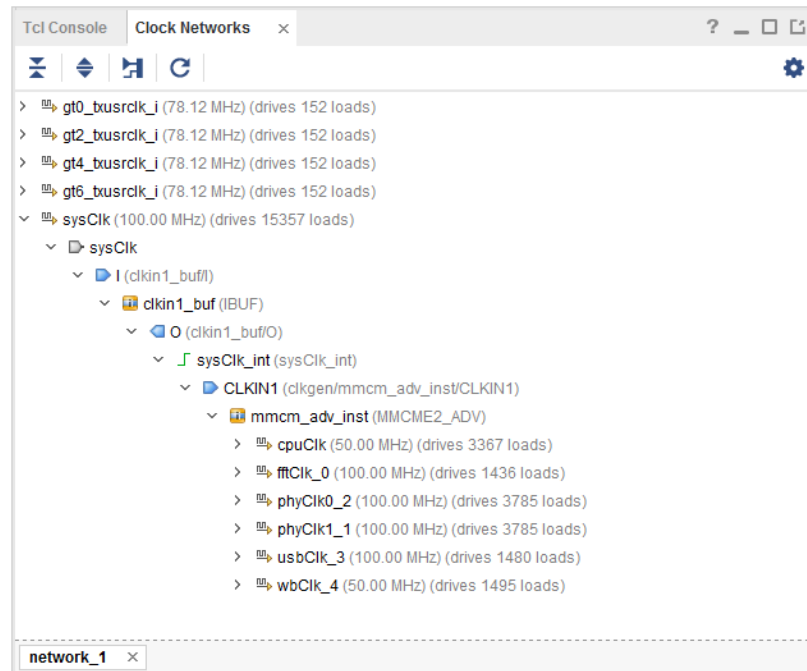
図 112: クロック ドメインが 250 MHz から 200 MHz に切り替わる



プライマリ クロックと生成クロックを同時に制約

タイミング例外を作成する前に、`report_clock_networks` でデザインに含まれるプライマリ クロックを特定しておくことが有益です。すべてのプライマリ クロックがお互いに非同期である場合、1 つの制約を使用してプライマリ クロックどうしと、それらから生成されたクロックどうしを分離できます。`report_clock_networks` で生成されたレポートのプライマリ クロックをガイドとして使用し、次の図に示すように各クロック グループと関連のクロックを分離できます。

図 113: クロック ネットワーク レポート



```
### Decouple asynchronous clocks
set_clock_groups -asynchronous \
-group [get_clocks sysClk -include_generated_clocks] \
-group [get_clocks gt0_txusrclk_i -include_generated_clocks] \
-group [get_clocks gt2_txusrclk_i -include_generated_clocks] \
-group [get_clocks gt4_txusrclk_i -include_generated_clocks] \
-group [get_clocks gt6_txusrclk_i -include_generated_clocks]
```

I/O 制約およびタイミング例外の制限

ほとんどのタイミング違反は内部パスで発生します。I/O 制約は、特にソース レジスタとデスティネーション レジスタが I/O バンク内にある場合は、最初のベースライン制約の作成時には必要ありません。I/O タイミング制約は、デザインおよびほかの制約が安定し、タイミング クロージャがほぼ達成されてから追加します。



ヒント: `config_timing_analysis -ignore_io_paths yes` Tcd コマンドを使用すると、インプリメンテーション中およびタイミング情報を使用するレポートですべての I/O パスのタイミングを無視できます。このコマンドは、デザインをメモリに開く前または開いた直後に手動で入力する必要があります。

RTL 設計者の推奨事項に基づき、タイミング例外は制限して使用し、実際のタイミング問題を隠すために使用しないでください。この段階の前に、クロック間のフォルス パスまたはクロック グループを確認し、確定しておく必要があります。

IP 制約は、すべて保持する必要があります。IP タイミング制約がないと、既知のフォルス パスがタイミング違反としてレポートされる可能性があります。

各段階後のデザイン WNS の評価

合成およびインプリメンテーションの各段階の後に、デザインの WNS を評価する必要があります。Tcl コマンド ライン フローを使用している場合は、ビルド スクリプトのインプリメンテーションの各段階後に `report_timing_summary` を含めることができます。Vivado IDE を使用している場合は、`tcl.post` スクリプトを使用して、各段階の後に `report_timing_summary` を実行できます。どちらの場合も、WNS が大幅に悪化した場合は、その段階のすぐ前のチェックポイントを解析する必要があります。

各インプリメンテーション段階の後にデザイン全体のタイミングを評価するのに加え、フローの各段階がタイミングに与える影響を評価するため、個別のパスを対象とした解析を実行できます。たとえば、タイミング パスのネット遅延の見積もり値が、配置後と最適化後で大幅に異なるとします。各段階後のクリティカル パスのタイミングを比較すると、クリティカル パスのタイミングがどの時点で悪化したのかを調べることができます。

合成後およびロジック最適化後

ネット遅延の見積もり値は、すべてのパスの最適な配置に近いものです。違反の発生しているパスを修正するには、次のいずれかを試してみてください。

- RTL を変更します。
- 異なる合成オプションを使用します。
- マルチサイクル パスなどのタイミング例外を追加します (適切でハードウェアでの機能に問題が発生しない場合)。

合成前および配置後

配置後のネット遅延の見積もり値は、ファンアウトが中程度から大きいネットを除き、最適な配線に近いもので、大きめの遅延が使用されます。また、この時点のネット遅延では密集やホールドの修正が考慮されておらず、タイミング結果が実際より見積もりの良いものとなることがあります。

クロック スキューは正確に見積もられ、バランスの悪いクロック ツリーのスラックへの影響を確認するのに使用できます。

ホールドの修正は、最小遅延解析を実行することにより見積もることができます。スライス、ブロック RAM、または DSP 間の WHS が -0.500 ns 以上である大きなホールド違反は、修正する必要があります。小さい違反は、たいていは配線で修正されます。

注記: PCIe® ブロックなどの専用ブロックに入出力されるパスでは、ホールド タイムの見積もり値が -0.500 ns 以上であっても、配線で自動的に修正されます。このような場合は、配線後に `report_timing_summary` を実行し、対応するすべてのホールド違反が修正されることを確認してください。

物理最適化の前後

次に関するタイミングの問題を修正するために、物理最適化を実行する必要があるかを評価します。

- ファンアウトの大きいネット (`report_high_fanout_nets` でファンアウトが最大のクロック以外のネットを表示)
- ドライバーとロードが遠いネット
- パイプライン レジスタの使用が最適でないデジタル信号処理 (DSP) およびブロック RAM

配線の前後

ネットが完全に配線されていない場合を除き、スラックは実際の配線ネット遅延でレポートされます。スラックには、セットアップに対するホールドの修正の影響と、密集の影響が反映されます。

配線後には、ワースト セットアップ スラック (WNS) 値にかかわらず、ホールド違反が残っていないようにする必要があります。デザインでホールド要件が満たされていない場合、さらに解析が必要です。これは通常、密集度が高く、配線でそれ以上のタイミングの最適化が実行されないことが原因です。ホールド違反が大きい (4 ns 以上) 場合にも発生することがあります。大きいホールド違反は、配線でデフォルトでは修正されません。大きいホールド違反は通常、不適切なクロック制約、大きいクロック スキュー、不適切な I/O 制約が原因ですが、これらは配置後または合成後に特定できます。

ホールド要件が満たされている (WHS > 0) 場合は、[タイミング違反の解析および解決](#)の解析手順に従ってください。

ベースライン制約の作成とタイミング制約の検証

次の手順は、タイミング クロージャ達成に向けての進行状況を確認し、ボトルネックとなる可能性のある問題を特定するのに役立ちます。

1. 合成済みデザインを開きます。
2. `report_timing_summary -delay_type min_max` コマンドを実行し、表示される情報を次の表に記録します。

	WNS	TNS	タイミングが満たされていない 終点 (Failing Endpoints) の 数	WHS	THS	タイミングが満たされていない 終点 (Failing Endpoints) の 数
合成						

3. 合成後の `report_timing_summary` テキスト レポートを開き、`check_timing` の `no_clock` セクションを記録します。

デザインで満たされていないクロック要件の数: _____

4. `report_clock_networks` コマンドを実行し、デザインのプライマリ クロックのソース ピン/ポートを特定します。QPLLOUTCLK および QPLLOUTREFCLK は、パルス幅のみのチェックなので無視します。

デザインで制約されていないクロックの数: _____

5. `report_clock_interaction -delay_type min_max` コマンドを実行し、WNS パス要件の結果を並べ替えます。

デザインで最小の WNS パス要件: _____

6. `report_clock_interaction` の結果を [WHS] で並べ替え、合成後に大きなホールド違反 (> 500 ps) があるかどうかを調べます。

デザインで最大の負の WHS: _____

7. `report_clock_interaction` の結果を [Inter-Clock Constraints] で並べ替え、「unsafe」と示されるクロックペアをすべてリストします。

8. 合成済みデザインを開くとき、クリティカル警告はいくつ表示されますか。

合成済みデザインのクリティカル警告の数: _____

9. どのタイプのクリティカル警告がありますか。

各クリティカル警告タイプの例:

10. `report_high_fanout_nets -timing -load_types -max_nets 25` を実行します。

フリップフロップで駆動されていないファンアウトの大きいネットの数: _____

フリップフロップで駆動されていないファンアウトが最大のネット上にあるロードの数: _____

ファンアウトの大きいネットにスラックが負のものがある場合、その WNS: _____

11. デザインをインプリメントします。各段階の後に `report_timing_summary` コマンドを実行し、表示される情報を次の表に記録します。

	WNS	TNS	タイミングが満たされていない 終点 (Failing Endpoints) の 数	WHS	THS	タイミングが満たされていない 終点 (Failing Endpoints) の 数
最適化						
配置						
物理最適化						
配線						

12. `report_exceptions -ignored` コマンドを実行し、デザインで重複している制約がないかを確認します。結果を記録します。

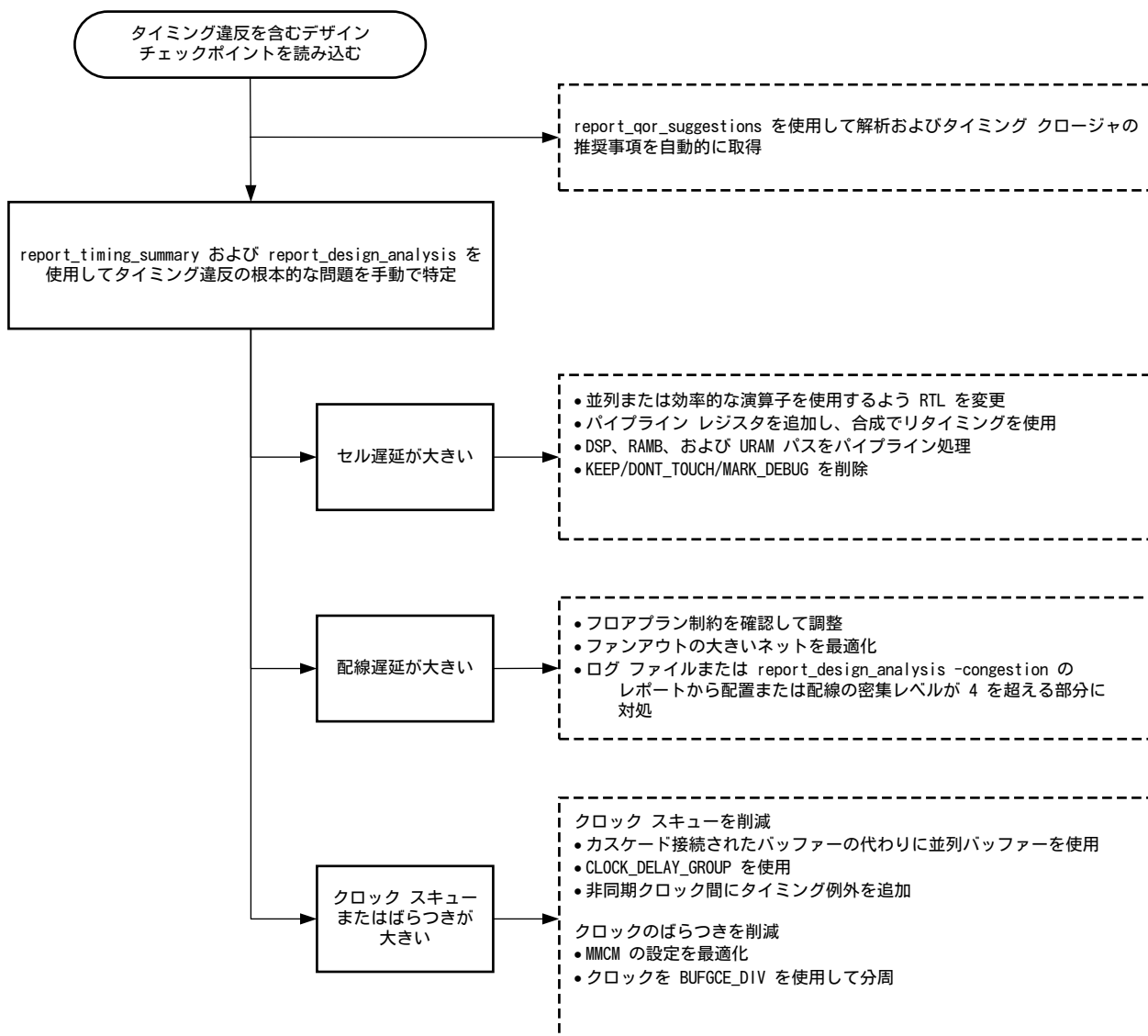
タイミング違反の解析および解決

タイミングドリブン アルゴリズムは、各クロック ドメインの最大の違反に焦点を置いて処理を実行します。最大の違反に関連する問題を理解して修正すると、インプリメンテーション フローを再実行したときに、ほとんどの小さい違反も修正されます。多数のパスを改善する解決策に焦点を置くと有益です。たとえば、最適でないクロッキングを修正すると、通常多数のパスに影響するので、サイリンクスでは、パス特定の解決策を試す前にこれらの問題に焦点を置くことをお勧めします。

QoR 推奨項目レポート コマンドを使用すると、問題が自動的に特定され、推奨項目が重要度順に表示されます。推奨項目の適用前後に QoR 評価レポートを実行すると、タイミング クロージャにどれだけ近づいたかがわかります。QoR 評価のスコアが上がり、詳細な表に含まれる確認が必要とマークされた項目が減少していれば、改善しているということです。

次の図に、タイミング違反を解析して解決する基本的なプロセスを示します。

図 114: タイミング違反の解析および解決



X20036-110617

タイミング違反の根本的な原因の特定

セットアップでは、まず各クロック グループの最大の違反を解析します。クロック グループとは、あるクロックで取り込まれるクロック内、クロック間、および非同期のパスすべてを指します。

ホールドでは、すべての違反を次のように解析する必要があります。

- 配線前は、0.5 ns を超える違反のみを確認します。
- 配線後は、最大の違反から開始します。

タイミング スラックの確認

セットアップおよびホールド スラックには、複数の要素が影響します。次の簡潔なセットアップおよびホールド スラックの式から、各要素を特定できます。

- スラック (セットアップ/リカバリ) = セットアップ パス要件:

- データパス遅延 (最大)
- + クロック スキュー
- クロックのばらつき
- セットアップ/リカバリ タイム

- スラック (ホールド/リムーバル) = ホールド パス要件:

- + データパス遅延 (最小)
- クロック スキュー
- クロックのばらつき
- ホールド/リムーバル タイム

タイミング解析では、クロック スキューは常に次のように算出されます。

- クロック スキュー = デスティネーション クロック遅延 - ソース クロック遅延 (存在する場合は共通ノードの後)

違反が発生しているタイミング パスの解析中、各変数の相対的な影響を調べ、どの変数が違反に最も影響しているかを判断する必要があります。その後主要な要因を解析して、パスのどの特性がその値に最も影響しているかを理解し、その影響を軽減するためのデザインおよび制約の変更を特定します。デザインまたは制約の変更が実質的でない場合は、最悪のものから開始してほかのすべての要因を同様に解析する必要があります。次に、典型的な要因を最悪のものから順にリストします。

セットアップ/リカバリの場合:

- データパス遅延: データパス遅延からタイミング パス要件を減算します。差が負のスラック値と同等である場合は、パス要件が厳しすぎるか、データパス遅延が大きすぎます。
- データパス遅延 - セットアップ/リカバリ タイム: データパス遅延からタイミング パス要件を減算し、セットアップ/リカバリ タイムを加算します。差が負のスラック値と同等である場合は、パス要件が厳しすぎるか、セットアップ/リカバリ タイムが通常より大きいことが違反の原因となっています。
- クロック スキュー: クロック スキューおよびスラックが似たような負の値であり、スキューの絶対値が数百ピコ秒以上である場合、スキューが主要な要因であり、クロック トポロジを見直す必要があります。
- クロックのばらつき: クロックのばらつきが数百ピコ秒を超える場合、クロック トポロジとジッター値を見直してばらつきが大きい原因を理解する必要があります。

ホールド/リムーバルの場合:

- クロック スキュー: クロック スキューが 300 ps を超える場合、クロック トポロジを見直す必要があります。
- クロックのばらつき: クロックのばらつきが 200 ps を超える場合、クロック トポロジとジッター値を見直してばらつきが大きい原因を理解する必要があります。
- ホールド/リムーバル タイム: ホールド/リムーバル タイムが数百ピコ秒以上である場合、プリミティブのデータシートを見直してこれが予測される結果であるかどうかを確認できます。

- ホールド パス要件: 要件は通常 0 です。0 でない場合、タイミング制約が正しいかどうかを確認する必要があります。

すべてのタイミング制約が正しく妥当である場合は、タイミング違反の最も一般的な要因はセットアップ/リカバリ タイミング パスのデータパス遅延およびホールド/リムーバル タイミング パスのスキューです。デザイン サイクルの初期段階では、これら 2 つの要因を解析することによりほとんどのタイミング問題を修正できます。デザインおよび制約を改善および調整した後は、残りの違反は複数の要素が組み合わさって発生しており、すべての要素を同時に調べてどれを改善するかを特定する必要があります。

タイミング解析の概念の詳細は『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906)の[タイミング解析の実行](#)、タイミング レポート (report_timing_summary/report_timing) の詳細は同じガイドの[タイミング解析機能](#)を参照してください。

デザイン解析レポートの使用

タイミング クロージャを達成するのが困難な場合、またはアプリケーションの全体的なパフォーマンスを向上しようとしている場合は、合成の実行後およびインプリメンテーション フローの各段階の後に、デザインの特性を確認する必要があります。QoR 解析では、グローバルおよびローカルの複数の特性を同時に見て、デザインおよび制約に最適でないものがあるか、ターゲット デバイスおよびインプリメンテーション ツールに適切でないロジック構造があるかを確認します。report_design_analysis コマンドを実行すると、論理特性、タイミング特性、物理特性が表示され、QoR の根本的な原因を簡単に解析できます。

注記: report_design_analysis コマンドでは、タイミング制約が完全であるか正確であるかはレポートされません。



ヒント: Vivado IDE でデザイン解析レポートを実行すると、表示が向上し、自動フィルターおよびクロスプロープが提供されます。

次のセクションでは、タイミング パス特性解析についてのみ説明します。デザイン解析レポートには、密集およびデザインの複雑性に関する有益な情報も含まれます。

関連情報

[デザインが適切に制約されているかを確認](#)

パス特性の解析

50 個のワースト セットアップ タイミング パスをレポートするには、Vivado IDE の [Report Design Analysis] ダイアログ ボックスを使用するか、次のコマンドを使用します。

```
report_design_analysis -max_paths 50 -setup -name design_analysis_postRoute
```

次の図に、このコマンドで生成される [Setup Path Characteristics] (セットアップ パス特性) の表の例を示します。追加の列を表示するには、水平方向にスクロールします。

図 115: 配線後のデザイン解析レポートのタイミング パス特性

Design Analysis												
Setup Path Characteristics												
General Information	Name	Requirement	Path Delay	Logic Delay	Net Delay	Clock Skew	Slack	Clock Relationship	Logic Levels	Routes	Logical Path	Start Point (
Setup Path Characteristics	Path 1	2.034	1.833	0.755	1.078	-0.292	-0.116	Safely Timed	1	2	URAM288_BASE LUT5 FDCE	clk_out5_s1
Logic Level Distribution	Path 2	2.034	2.08	0.92	1.16	-0.008	-0.079	Safely Timed	6	5	FDCE CARRY8 CARRY8 LUT4 LUT6 LUT3 CARRY8 FDCE	clk_out5_s1
	Path 3	2.034	1.463	0.449	1.014	-0.234	-0.055	Safely Timed	4	3	FDCE LUT2 CARRY8 CARRY8 LUT2 RAMB18E2	clk_out5_s1
	Path 4	2.034	1.747	0.761	0.986	-0.302	-0.040	Safely Timed	1	1	URAM288_BASE LUT4 FDCE	clk_out5_s1
	Path 5	2.034	1.441	0.449	0.992	-0.234	-0.033	Safely Timed	4	3	FDCE LUT2 CARRY8 CARRY8 LUT2 RAMB18E2	clk_out5_s1
	Path 6	2.034	1.444	0.449	0.995	-0.231	-0.033	Safely Timed	4	3	FDCE LUT2 CARRY8 CARRY8 LUT2 RAMB18E2	clk_out5_s1
	Path 7	2.034	2.036	0.945	1.091	0	-0.028	Safely Timed	7	5	FDCE CARRY8 CARRY8 LUT4 LUT6 LUT3 CARRY8 CARRY8 FDCE	clk_out5_s1
	Path 8	2.034	1.963	0.949	1.014	-0.054	-0.008	Safely Timed	7	6	FDCE CARRY8 CARRY8 LUT4 LUT4 LUT6 LUT3 CARRY8 FDCE	clk_out5_s1

次に、この表の操作に関するヒントを示します。

- ツールバーの [Show Percentage] ([%]) ボタンをクリックし、数値と割合を切り替えます。これは、セル遅延とネット遅延の割合を確認するのに便利です。
- デフォルトでは、値がヌルまたは空の列のみが非表示になっています。ツールバーの [Hide Unused] (未使用を表示) ボタンをクリックすると、すべての列が表示されます。または、表のヘッダーを右クリックして、列の表示/非表示を切り替えることができます。

この表から、どの特性が各パスにタイミング違反を発生させているのかを特定できます。

- ロジック遅延の割合 ([Logic Delay]) が大きい
 - ロジック段数は多いですか ([Logic Levels])。
 - ロジック最適化を妨げる制約または属性が設定されていますか ([Dont Touch]、[Mark Debug])。
 - パスにブロック RAM や DSP などのロジック遅延の大きいセルが含まれていますか ([Logical Path]、[Start Point Pin Primitive]、[End Point Pin Primitive])。
 - パスの要件が現在のパス トポロジには厳しすぎませんか ([Requirement])。
- ネット遅延の割合 ([Net Delay]) が大きい
 - パスのファンアウトの大きいネットがありますか ([High Fanout]、[Cumulative Fanout])。
 - セルが離して配置可能な複数の Pblock に割り当てられていますか ([Pblocks])。
 - セルが離れて配置されていますか ([Bounding Box Size]、[Clock Region Distance])。
 - SSI デバイスで SLR の境界をまたぐネットがありますか ([SLR Crossings])。
 - 配置は正しいようなものにもかかわらず、予測よりもかなり大きいネット遅延がありますか。パスを選択し、その配置と配線を [Device] ウィンドウに表示します。
 - ブロック RAM または DSP セルに不足しているパイプライン レジスタはありますか ([Comb DSP]、[MREG]、[PREG]、[DOA_REG]、[DOA_REG])。
- スキューが大きい (セットアップで <-0.5 ns、ホールドで >0.5 ns) ([Clock Skew])。
 - そのパスはクロック乗せ換えパスですか ([Start Point Clock]、[End Point Clock])。
 - クロックは同期ですか、非同期ですか ([Clock Relationship])。
 - パスは I/O 列をまたいでいますか ([IO Crossings])。



ヒント: Vivado IDE でタイミング パスの詳細を確認するには、表でパスを選択し、[Properties] ウィンドウを見ます。

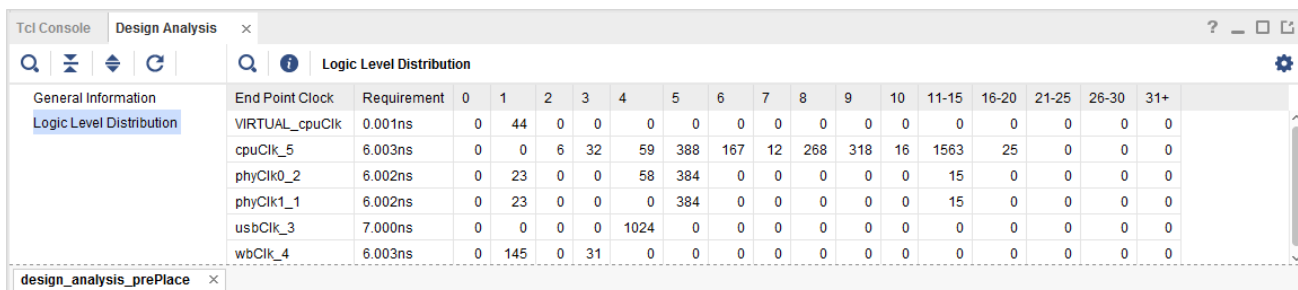
ロジック段の分布の確認

report_design_analysis コマンドでは、ワースト パス 1000 個 (デフォルト) の Logic Level Distribution (ロジック段の分布) の表も生成され、デザインの長いパスを特定するのに使用できます。通常、配置の段階でタイミングを満たすために最長のパスが最初に最適化されるので、短いパスの配置の質が低下する可能性があります。全体的なタイミング QoR を向上するため、長いパスをなるべくなくすようにしてください。このため、配置の前に最長のパスを確認することをお勧めします。

次の図に、デザインのロジック段の分布の例を示します。この例では、クロック周期は 7.5 ns で、ロジック段数が 17 のパスを含む最悪のものから 5000 個のパスがレポートされています。このレポートを生成するには、次のコマンドを実行します。

```
report_design_analysis -logic_level_distribution -logic_level_dist_paths
5000 -name design_analysis_prePlace
```

図 116: 配置前のデザイン解析レポートのタイミング パス特性



General Information	End Point Clock	Requirement	0	1	2	3	4	5	6	7	8	9	10	11-15	16-20	21-25	26-30	31+
Logic Level Distribution	VIRTUAL_cpuClk	0.001ns	0	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	cpuClk_5	6.003ns	0	0	6	32	59	388	167	12	268	318	16	1563	25	0	0	0
	phyClk0_2	6.002ns	0	23	0	0	58	384	0	0	0	0	0	15	0	0	0	0
	phyClk1_1	6.002ns	0	23	0	0	0	384	0	0	0	0	0	15	0	0	0	0
	usbClk_3	7.000ns	0	0	0	0	1024	0	0	0	0	0	0	0	0	0	0	0
	wbClk_4	6.003ns	0	145	0	31	0	0	0	0	0	0	0	0	0	0	0	0

ロジック段数が 10 を超える場合、-min_level および -max_level オプションを使用して、指定した最小レベルと最大レベル間のパスのより詳細な分布情報を指定できます。次に例を示します。

```
report_design_analysis -logic_level_distribution -min_level 16 -max_level 20
-logic_level_dist_paths 5000 -name design_analysis_1
```

最長のパスのタイミング レポートを生成するには、次のコマンドを実行します。

```
report_timing -name longPaths -of-objects [get_timing_paths -setup -to [get_clocks
cpuClk_5] -max_paths 5000 -filter {LOGIC_LEVELS>=16 && LOGIC_LEVELS<=20}]
```

解析結果に基づいて、RTL を変更、異なる合成オプションを使用、またはタイミング制約および物理制約を変更してネットリストを向上できます。

データパス遅延およびロジック段数

通常、パスに含まれる LUT およびその他のプリミティブの数が遅延に影響する最も重要な要素です。LUT 遅延はデバイスによってレポート方法が異なるので、個別のセル遅延と配線遅延の範囲を考慮する必要があります。

パス遅延が次の場合を考えます。

- 7 シリーズ デバイスでセル遅延が > 25%、UltraScale デバイスで > 50%。

パスを変更して短くするか、より高速のロジック セルを使用できますか。 [ロジック遅延の削減](#) を参照してください。

- 7 シリーズ デバイスでセル遅延が > 75%、UltraScale デバイスで > 50%。

このパスはホールド違反の修正の影響を受けていますか。これを判断するには、report_design_analysis -show_all を実行し、[Hold Detour] 列を確認します。対応する解析手法を使用します。

- 。 はい: 影響を受けているネットは CDC パスの一部ですか。
 - はい: CDC パスに制約は設定されていますか。
 - いいえ: ホールド違反が修正されたパスの始点および終点にバランスの取れたクロック ツリーが使用されていますか。スキュー値を確認します。
- 。 いいえ: 次の密集に関する項目を参照します。

このパスは密集の影響を受けていますか。各ネット遅延およびファンアウトを確認し、[Device] ウィンドウで配線リソースの表示をオンにして配線を表示します (配線後の解析のみ)。密集メトリクスをオンにして、パスが密集エリアまたはその近くに配置されているかも確認できます。簡単な評価には次の解析手順を使用し、包括的な解析には[密集によるネット遅延の削減](#)を参照してください。

- 。 はい: 遅延値が最も大きいネットで、ファンアウトは小さいですか (< 10)。
 - はい: 配線が最適であるように見える (直線) がドライバーとロードが離れている場合、最適でない配置は密集に関連しています。最適な解決手法を判断するには、[密集の解消](#)を参照してください。
 - いいえ: 物理ロジック最適化を使用してネットのドライバーを複製してみます。複製すると、各ドライバーが自動的にロードの近くに配置されるので、全体的なデータパス遅延が削減されます。詳細およびほかの手法を学ぶには、[ファンアウトの大きいネットを最適化](#)を参照してください。
- 。 いいえ: デザインが分散しすぎています。配置を向上するには、次のいずれかの方法を試してみてください。
 - [制御セットの削減](#)
 - [コンパイル フローの調整](#)
 - [フロアプラン](#)

クロックのスキューとばらつき

ザイリンクス デバイスでは、さまざまなタイプの配線リソースを使用して、一般的なクロッキング方法と、ファンアウトの大きいクロック、小さい伝搬遅延、非常に小さいスキューなどの要件をサポートします。クロック スキューは、組み合わせロジックまたはインターコネクトのどちらを含む場合でも、レジスタ間のパスに影響します。



推奨: デザイン解析レポート (report_design_analysis) を実行してタイミング レポートを作成します。このレポートに、クロック スキュー データに関する情報が含まれます。クロック ネットに可能なクロック スキューがないことを確認してください。

高パフォーマンスのクロック ドメイン (300 MHz 以上) のクロック スキューは、パフォーマンスに影響します。通常、クロック スキューは 500 ps 以下である必要があります。たとえば、500 ps は 300 MHz クロック周期の 15% であり、ロジック段数 1 または 2 のタイミング バジェットに対応します。クロック乗せ換えパスでは、クロックで異なるリソースが使用され、共通ノードがクロック ツリーのさらに上の方に配置されているので、スキューが大きくなる場合があります。SDC ベースのツールでは、set_clock_groups、set_false_path、または set_max_delay - datapath_only を使用してクロック間のタイミング解析を実行しないよう指定しない場合、すべてのクロック間のパスのタイミングが解析されます。

関連情報

[クロック スキューの削減](#)
[クロックのばらつきの削減](#)

ロジック遅延の削減

Vivado インプリメンテーションでは、まず最もクリティカルなパスに焦点が置かれますが、これにより配置後または配線後にそれほど困難でなかったパスがクリティカルになることがあります。ザイリンクスでは、合成後または `opt_design` 後に最長のパスを特定して向上することをお勧めします。これはタイミングおよび消費電力 QoR (結果の品質) に最も大きく影響し、タイミング クロージャを達成するまでの配置配線の実行回数を大幅に削減できます。

配置前のタイミング解析では、理想的な配置および典型的なクロック スキューに対応する見積もり遅延が使用されます。 `report_timing`、 `report_timing_summary`、または `report_design_analysis` を使用すると、ロジック段数が多いパス、セル遅延が大きいパスをすばやく特定できます。これらのパスでは通常、配置前にはタイミングが満たされないか、かろうじてしか満たされません。 [タイミング違反の根本的な原因の特定](#) に示されている設計手法を使用して、デザインをインプリメントする前に向上する必要がある長いパスを見つけます。

関連情報

[タイミング違反の根本的な原因の特定](#)

通常ファブリックパスを最適化

通常ファブリックパスは、ファブリックレジスタまたはシフトレジスタ間のパスで、LUT などのさまざまなリソースを通過します。デザイン解析レポートのタイミングパス特性の表にはロジックパス トポロジのサマリが示され、次の問題を特定できます。

- 数個の小型 LUT がカスケード接続されている

LUT へのマップは、階層、KEEP_HIERARCHY、DONT_TOUCH、または MARK_DEBUG 属性が存在するか、10 以上のファンアウトを持つ中間信号の影響を受けます。 `opt_design -remap` オプションを使用するか、 `-directive` オプションの `AddRemap` または `ExploreWithRemap` 指示子を使用して、小型の LUT をまとめてロジック段数を削減してください。小型の LUT 間のネット ファンアウトが 1 より大きいために `opt_design` で最長のパスを最適化できない場合は、LUT に `LUT_REMAP` プロパティを設定して最適化を強制できます。

- パスに CARRY セルが 1 つ存在する

CARRY プリミティブは、カスケード接続した場合にタイミング QoR に最も有益です。CARRY セルは LUT よりも配置が困難であり、合成で 1 つの CARRY セルではなく複数の LUT が使用されるようにした方が、多くの場合に LUT の構造がよくなり、より柔軟に配置を実行できます。 `synth_design` で `-directive FewerCarryChains` オプションを使用するか、 `PerfThresholdCarry` ストラテジ (プロジェクト モードのみ) を使用して、ほとんどの 1 つの CARRY セルを削除してみてください。または、 `CARRY_REMAP` プロパティを使用して、 `opt_design` でタグが付いた CARRY セルを LUT にリマップされるようにします。

注記: この最適化手法は、 `report_qor_suggestions` Tcl コマンドにより自動的に適用されます。

- パスがシフトレジスタ (SRL) で終了する

RTL で `SRL_STYLE` 属性を使用して、シフトレジスタから最初のレジスタを取り出します。詳細は、『Vivado Design Suite ユーザーガイド: 合成』 (UG901) の [このセクション](#) を参照してください。または、 `opt_design` で同じ最適化をインプリメントする前に、 `SRL_STAGES_TO_REG_INPUT` プロパティを適用します。詳細は、『Vivado Design Suite ユーザーガイド: インプリメンテーション』 (UG904) の [このセクション](#) を参照してください。

注記: この最適化手法は、 `report_qor_suggestions` Tcl コマンドにより自動的に適用されます。

- パスがファブリックレジスタ (FD) のクロックイネーブルまたは同期セット/リセットで終了する

データピン (D) で終了するパスのマージンが大きく、ロジック段数が少ない場合は、RTL で信号に `EXTRACT_ENABLE` または `EXTRACT_RESET` 属性を `no` に設定します。または、最適化するレジスタに `CONTROL_SET_REMAP` プロパティを設定して `opt_design` で同じ最適化が実行されるようにします。

注記: この最適化手法は、 `report_qor_suggestions` Tcl コマンドにより自動的に適用されます。



ヒント: 合成後のパスから対応する RTL ソース コードにクロスプローブするには、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』 (UG906) の[このセクション](#)を参照してください。

関連情報

[ロジックを制御ピンからデータ ピンに移動](#)

専用ブロックおよびマクロ プリミティブを含むパスを最適化

専用ブロックおよびマクロ プリミティブ (DSP、ブロック RAM、UltraRAM など) で開始または終了するパス、あるいはその間のパスは、これらのプリミティブには次のようなタイミング特性があるので、特別な注意が必要です。

- 一部のピンでセットアップ/ホールド/clock-to-output タイミング アークの値が大きい。たとえばブロック RAM の clock-to-output 遅延は、オプションの出力レジスタなしで 1.5 ns、オプションの出力レジスタありで 0.4 ns です。詳細は、ターゲット デバイス シリーズのデータシートを参照してください。
- 通常の FD/LUT 接続よりも配線遅延が大きい。
- 通常の FD-FD パスよりもクロック スキューの変動が大きい。

また、これらのプリミティブの使用可能性とサイト ロケーションも CLB スライスより制限されるので、その配置はより困難であり、QoR 低下の原因となることがよくあります。

これらの理由から、ザイリンクスでは次を推奨します。

- 専用ブロックおよびマクロ プリミティブで開始または終了するパスをできるだけパイプライン処理します。
- これらのセルに接続されている組み合わせロジックの構造を変更し、ロジック段数を最低でも 1 セル、またはパイプライン処理により追加されるレイテンシが懸念される場合は 2 セル削減します。
- 配置前にこれらのパスのセットアップ タイミングが最低でも 500 ps で満たされるようにします。
- 専用ブロックまたはマクロ プリミティブを遠くに配置する必要がある場合は、それら多数に接続されているロジック コーンを複製します。
- デザインに含まれる DSP ブロックに入出力されるパスまたは DSP ブロック内のタイミング要件が厳しい場合は、`opt_design -dsp_register_opt` を実行してレジスタをタイミングがより適切になる位置に移動します。

注記: `opt_design` の段階ではタイミングは見積もりであるため、`phys_opt_design -dsp_register_opt` を実行して、配置前にはタイミングが正確でなかった部分の移動を修正する必要がある場合もあります。

物理制約によるネット遅延の削減

すべてのデザインには、特に I/O ロケーション、場合によってクロックおよびロジック配置など、最小限の物理制約が含まれます。デザインでタイミング クロージャを実行する準備ができた時点では I/O ロケーションを変更することはできませんが、Pblock や LOC などの物理制約は解析する必要があります。 `report_design_analysis` のタイミング パス特性の表を使用して、各クリティカル パス上に Pblock 制約があるかどうかを判断します。

タイミング パス特性の表でパスを選択すると、Vivado IDE の [Properties] ウィンドウでパス上のセルを制約する Pblockを確認できます。Pblock によりロジックが分散する場合は、1 つまたは複数の Pblock 制約を削除することを検討します。

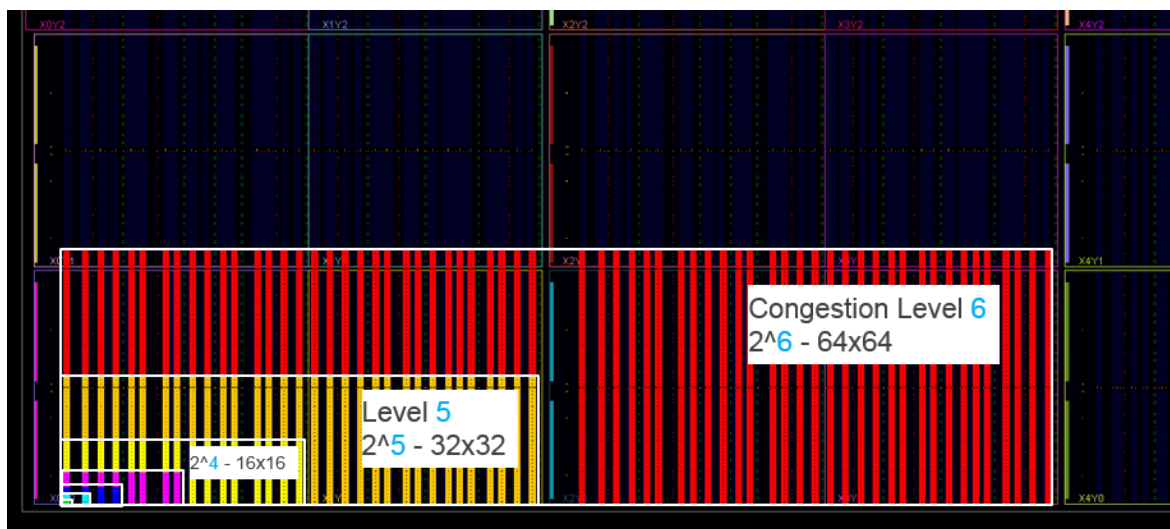
密集によるネット遅延の削減

デバイスの密集は、クリティカル パスが密集したエリアの中または近くにある場合、またはデバイスの使用率が高く配置されたデザインの配線が困難な場合、タイミング クロージャの達成が困難になる原因となる可能性があります。多くの場合、密集により配線の実行時間が大幅に増加します。パスの配線遅延が見積もりよりも大きい場合は、デザインの密集度を解析して、密集を緩和するのに最適な手法を特定します。

密集エリアおよび密集レベルの定義

ザイリンクス デバイスの配線アーキテクチャは、左右上下の各方向へのさまざまな長さのインターコネクト リソースで構成されます。密集エリアは、特定の方向へのインターコネクト リソース使用率が 100% に近いか 100% を超えている、隣接したインターコネクト タイル (INT_XnYm) または CLB タイル (CLE_M_XnYm) を含む最小の矩形としてレポートされます。密集レベルは、矩形の 1 辺の長さに対応する正の整数です。次の図に、ザイリンクス デバイスでの密集エリアのクロック領域に対する相対サイズを示します。

図 117: [Device] ビューでの密集レベルとエリア



密集レベルの範囲

ツールでレポートされる密集レベルは、次の表に示すように定義されます。

注記: 密集レベルが 5 以上の場合、QoR に影響することがほとんどで、配線の実行時間が常に長くなります。

表 9: 密集レベルの範囲

レベル	エリア	密集度	QoR への影響
1、2	2x2、4x4	なし	なし
3、4	8x8、16x16	低	QoR が低下する可能性あり
5	32x32	中	QoR が低下する可能性大
6	64x64	高	配線が困難
7、8	128x128、256x256	不可能	配線不可能である可能性大

[Device] ウィンドウでのインターコネクトの密集度

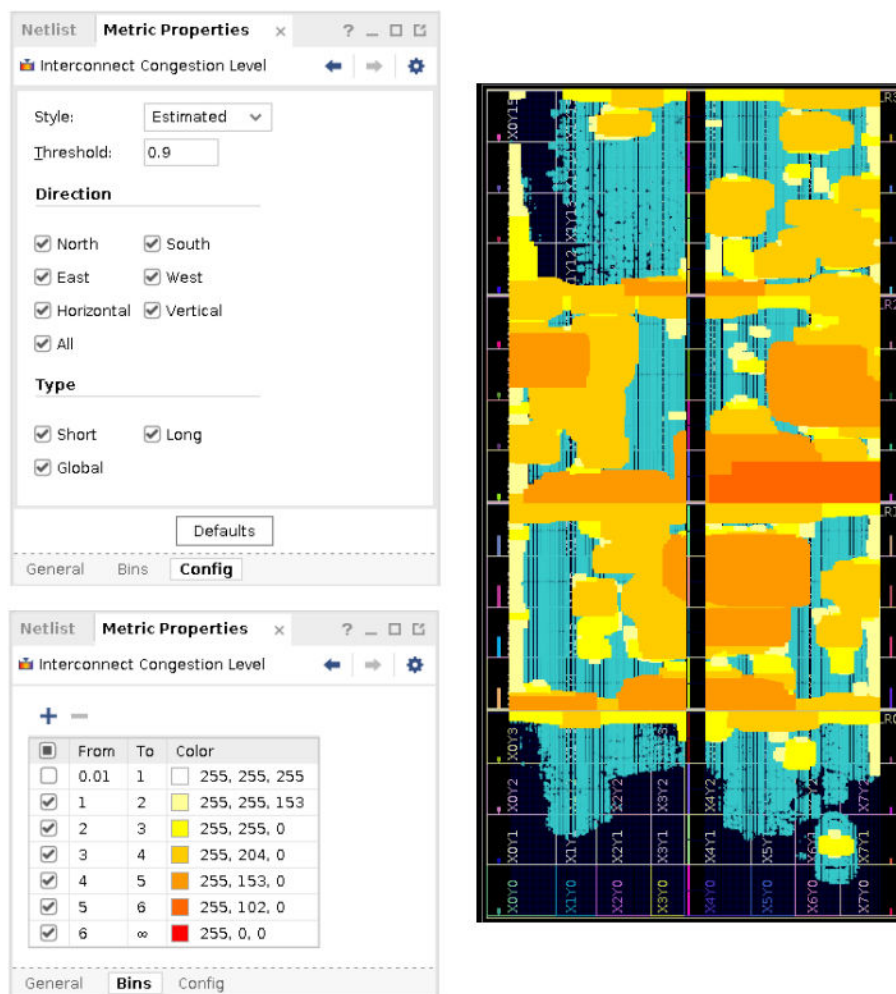
インターコネクトの密集度メトリクスは、配線リソースが過剰に使用されている最大の連続エリアをハイライトします。デフォルトでは、初期配置後の密集度と同様、このメトリクスは見積もりに基づきます。配線が存在する場合は、実際の配線も表示できます。配置後または配線後に [Device] ウィンドウを右クリックして [Metric]→[Interconnect Congestion Level] をクリックすると、この密集メトリクスを表示できます。

このメトリクスを使用すると、デバイス上の密集している部分を全体的にすばやく目で確認できます。次の図に、いくつかの密集エリアを含む配置済みデザインを示します。このメトリクスは、現在のインターコネクトの需要と供給に基づいており、しきい値は 0.9 (配線使用率 90%) に設定されています。有効な範囲は 0.1 ~ 0.9 です。

密集は、次の項目に基づいて表示できます。

- [Direction] (方向): [North] (上)、[South] (下)、[East] (右)、[West] (左)、[Vertical] (垂直)、[Horizontal] (水平)
- [Type] (タイプ): [Short] (短い)、[Long] (長い)、[Global] (グローバル)
- [Style] (スタイル): [Estimated] (見積もり)、[Routed] (配線済み)、[Mixed] (混合)

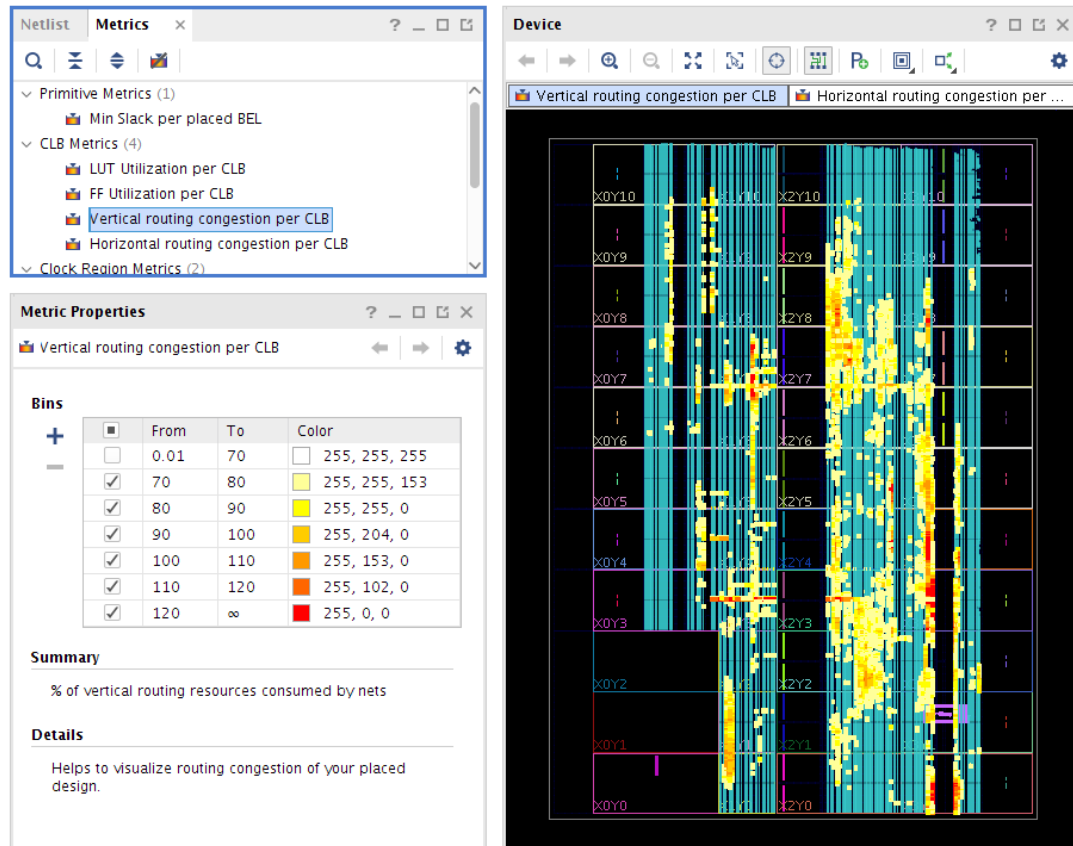
図 118: [Device] ウィンドウでのインターコネクトの密集度の例



実際の配線ではなく見積もりに基づく CLB ごとの配線の密集度を使用します。配置後または配線後に [Device] ウィンドウを右クリックして [Metric] → [Vertical and Horizontal Routing Congestion per CLB] をクリックすると、この密集メトリクスを表示できます。これにより、デバイス上の密集している部分を全体的にすばやく目で確認できます。次の図に、使用率が高く、ネットリストが複雑なために密集が発生している配置済みデザインを示します。

注記: この方法は、7 シリーズおよび UltraScale デバイスでのみ使用できます。

図 119: [Device] ウィンドウでの CLB ごとの密集度の例



配置ログでの密集のレポート

配置では、異なるフェーズを通して密集が見積もられ、密集したエリアのロジックが分散されます。これにより、インターコネクトの使用率が削減されて配線性が向上し、配線遅延の見積もり値と実際の値の相関性も向上します。ただし、使用率が高いなどの理由で密集を緩和できない場合は、配置で密集の詳細は表示されず、次のような警告メッセージが表示されます。

WARNING: [Place 46-14] The placer has determined that this design is highly congested and may have difficulty routing. Run report_design_analysis -congestion for a detailed report.

これは QoR に影響する可能性が高く、配線に進む前に密集の原因となっている問題を解決しておくことをお勧めします。メッセージに記載されているように、report_design_analysis コマンドを使用して実際の密集レベルをレポートし、密集エリアの場所と同じエリアに配置されているロジックを特定してください。

配線ログでの密集のレポート

配線では、密集レベルおよび特定リソースの配線の困難度に応じて追加のメッセージが表示されます。配線では、中間タイミング サマリも複数回生成されます。最初の間タイミング サマリはすべてのクロックが配線された後に生成され、配置後のタイミング解析に類似した WNS/TNS/WHST/TNS の値が示されます。次の中間タイミング サマリは、初期配線の後にレポートされます。タイミングが大幅に低下した場合は、タイミング QoR がホールド違反の修正または密集の影響を受けています。

密集レベルが 4 以上の場合は、密集の初期見積もりの表が生成され、密集の特性の詳細が示されます。

- [Global Congestion] は、配置の密集がどのように見積もられるかに似ており、すべてのインターコネクト タイプに基づいています。
- [Long Congestion] は、特定方向の長いインターコネクトの使用率のみを考慮しています。
- [Short Congestion] は、特定方向のその他すべてのインターコネクトの使用率を考慮しています。

32x32 (レベル 5) 以上の密集エリア (次の表で黄色でハイライト) は、QoR および配線性に影響する可能性が高くなります。長いインターコネクトが密集していると短いインターコネクトの使用量が多くなり、配線遅延が大きくなります。短いインターコネクトの密集は通常、実行時間が長くなる原因となり、タイルの割合が 5% を超える場合、QoR が低下する可能性が高くなります (次の表で赤でハイライト)。

図 120: 密集の初期見積もりの表

INFO: [Route 35-449] Initial Estimated Congestion

Direction	Global Congestion		Long Congestion		Short Congestion	
	Size	% Tiles	Size	% Tiles	Size	% Tiles
NORTH	16x16	1.95	32x32	1.68	32x32	11.58
SOUTH	8x8	1.90	16x16	2.00	32x32	9.23
EAST	8x8	0.93	2x2	0.20	32x32	9.14
WEST	8x8	1.37	2x2	0.15	32x32	14.50

配線の Global Iteration フェーズ中、オーバーラップがなく、セットアップおよびホールドの両方のタイミングを満たす (ホールド違反の修正が優先) 有効なソリューションが検索されます。有効なソリューションが見つからない場合は、次の例に示すように、有効な配線ソリューションが見つかるまでタイミングの最適化は停止されます。

```
Phase 4.1 Global Iteration 0
Number of Nodes with overlaps = 1157522
Number of Nodes with overlaps = 131697
Number of Nodes with overlaps = 28118
Number of Nodes with overlaps = 10971
Number of Nodes with overlaps = 7324
WARNING: [Route 35-447] Congestion is preventing the router from routing all nets.
The router will prioritize the successful completion of routing all nets over timing
optimizations.
```

有効な配線ソリューションが見つかったら、タイミング最適化が再び有効になります。

CLB の配線密集もレポートされ、密集度が高い CLB の名前が示されます。情報メッセージが表示され、メッセージ本文に密集度の高い CLB およびネットが記述されたテキスト ファイルの名前が示されます。このテキスト ファイルで CLB ピン フィードの密集に関係する CLB タイルとネットのリストを確認し、「密集の解消」セクションに記載されている密集緩和手法を使用して、デザインを配線する前に CLB の密集を解決できます。

INFO: [Route 35-443] CLB routing congestion detected. Several CLBs have high routing utilization, which can impact timing closure. Congested CLBs and Nets are dumped in: iter_200_CongestedCLBsAndNets.txt



ヒント: [Global] (グローバル)、[Long] (長い)、または [Short] (短い) 密集にレポートされる密集レベルが許容範囲内 (5 未満) であっても、CLB 配線が局地的に密集していると、配線エラーが発生することがあります。上記のメッセージおよび生成されたテキスト ファイルで、局地的な密集ホットスポットがないかどうかを確認してください。

有効な配線ソリューションが見つからない場合、次に示すようなクリティカル警告メッセージが複数表示され、完全に配線されていないネットの数およびオーバーラップしているインターコネクトの数が表示されます。

```
CRITICAL WARNING: [Route 35-162] 44084 signals failed to route due to routing
congestion. Please run report_route_status to get a full summary of the design's
routing.
```

```

...
CRITICAL WARNING: [Route 35-2] Design is not legally routed. There are 91566 node
overlaps.

```



ヒント: 配線中はネットが密集エリアの周辺に分散され、デザインが正常に配線された場合にログ ファイルにレポートされる最終的な密集レベルが削減されます。

デザイン解析レポートの密集レポート

密集があるかどうかを調べるには、[Report Design Analysis] コマンドを使用して密集レポートを生成すると、デバイスの密集したエリアとそれらのエリアにあるデザイン モジュールの名前を確認できます。このレポートには、配置および配線アルゴリズムにより特定された密集エリアを示す複数の密集に関する表が含まれます。次の図に、密集の表の例を示します。

図 121: 密集の表

Tcl Console	Messages	Metric Results	Design Analysis	Timing														?	—	□	□	
Q		≡	⚙	🔍	Placed Maximum													⚙				
General Information																						
Congestion		Window	Direction	Congestion Level	Congestion	Cell Names									Combined LUTs	LUT6	LUT5	Flop	MUXF	RAMB		
						Top Cell 1	Top Cell 2	Top Cell 3														
Placed Maximum		Window 1	North	4	120	inst_1022144/inst_1022144/inst_1021022144/inst_1022134/inst_1018559/inst_990436 (10%)								26%	37%	22%	43%	0%	NA			
Placed Tile Based (V)		Window 2	East	4	107	inst_1022144/inst_cv_33 (17%)								26%	29%	7%	60%	1%	50%			
Placed Tile Based (H)		Window 3	South	4	131	inst_1022144/inst_1022144/inst_1021022144/inst_1022134/inst_1018559/inst_990436/inst_979691 (6%)								32%	38%	18%	54%	0%	50%			
		Window 4	West	2	143	inst_1022144/inst_1022144/inst_1021022144/inst_1022134/inst_1018559/inst_887992 (9%)								84%	40%	1%	60%	0%	NA			
design_analysis_1																						

[Placed Maximum]、[Initial Estimated Router Maximum]、および [Router Maximum] セクションには、上下左右の各方向において最も密集したエリアに関する情報が示されます。表の密集ウィンドウの 1 つを選択すると、対応する密集エリアが [Device] ウィンドウでハイライトされます。

これらの表は、デザインフローの異なる段階での密集を示します。

- [Placed Maximum]: セルの位置および配線のモデルに基づく密集を示します。
- [Initial Estimated Router Congestion]: クイック配線後の密集を示します。この段階では配置による密集の正確な状況が示されるので、密集の解析には最も有益な段階です。
- [Router Maximum]: 配線プログラムで密集を削減するために労力が費やされた後の密集を示します。

密集の表での密集の割合は、その密集ウィンドウでの配線リソースの使用率を示します。密集ウィンドウに含まれる階層セルの上位 3 つがリストされ、選択すると [Device] または [Schematic] ウィンドウでハイライトされます。密集ウィンドウでのセルの使用率もパーセントで示されます。

密集エリアに存在する階層セルを特定したら、このガイドの後の方で説明する密集を緩和する手法を使用して、デザインの全体的な密集を削減します。

デザイン解析レポートの密集レポートの生成および解析の詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』 (UG906) のこのセクションを参照してください。

デザイン解析レポートの複雑性レポート

複雑性レポートには、Rent 指数、平均ファンアウト、最上位デザインまたは階層セルでの最下位セル タイプの分布が示されます。Rent 指数は、最小カット アルゴリズムを使用してデザインを再帰的に分割した場合のネットリストパーティションのポート数とセル数の関係を示します。グローバル配置中に配置プログラムにより使用されるアルゴリズムと同様のアルゴリズムにより算出されます。そのため、デザインの階層がグローバル配置中に検出される物理パーティションとよく一致している場合に特に、配置の課題を示す指標として使用できます。

Rent 指数が大きいデザインは、密に接続されたロジックのグループがほかのグループとも密に接続されていることを示します。これは、グローバル配線リソースの使用率が高く、配線の複雑性が高いことを意味します。このレポートに示される Rent 指数は、未配置および未配線のネットリストに対して算出されます。配置後の Rent 指数は論理パーティションではなく物理パーティションに基づくので、同じデザインでも配置後の Rent 指数は異なる場合があります。

次のオプションを使用すると、デザイン解析レポートは複雑性モードで生成されます。

- [Report Design Analysis] ダイアログ ボックスの [Options] タブで、[Complexity] オプションを確認します。
- `report_design_analysis Tcl` コマンドを `-complexity` オプションを使用して実行する。

次の図に、複雑性レポートの例を示します。

図 122: 複雑性レポート

General Information		Instance	Module	Rent	Average Fanout	Total Instances	LUT1	LUT2	LUT3	LUT4	LUT5	LUT6	Memory LUT	DSP	RAMB	MUXF	URAM
√	N	cv_33	cv_33	0.41	2.91	1131310	0.7%	11.9%	18.4%	15.7%	17.2%	36.1%	22141	125	913	23685	82
>		Inst_1022144 (cv_71)	cv_71	0.42	2.86	1011347	0.6%	11.7%	18.6%	15.8%	17.2%	36.1%	17807	122	810	21452	82
>		Inst_1029467 (cv_13905)	cv_13905	0.37	3.44	7236	0.7%	11.9%	10.2%	24.6%	16.2%	36.4%	1472	0	0	3	0
>		Inst_1036789 (cv_13934)	cv_13934	0.41	3.44	7236	0.7%	11.9%	10.2%	24.6%	16.2%	36.4%	1472	0	0	3	0
>		Inst_1051863 (cv_13963)	cv_13963	0.47	3.01	61384	1.6%	9.1%	19.6%	13.4%	17.7%	38.6%	22	0	68	1892	0
>		Inst_1052499 (cv_13973)	cv_13973	0.63	3.16	1366	0.7%	13.3%	12.6%	9.6%	27.5%	36.3%	8	1	4	9	0
>		Inst_1055086 (cv_13982)	cv_13982	0.42	2.64	2525	2.3%	25.4%	12.7%	21.7%	11.4%	26.4%	4	0	6	0	0
>		Inst_1059242 (cv_13998)	cv_13998	0.25	2.32	4076	2.7%	39.1%	18.6%	16.2%	9.7%	13.6%	0	0	12	0	0
>		Inst_1071723 (cv_14030)	cv_14030	0.5	3.3	10914	0.4%	12.2%	15.4%	11.7%	16.4%	43.8%	912	2	8	204	0
>		Inst_1075799 (cv_14081)	cv_14081	0.41	3.1	4001	0.2%	18.3%	10.7%	14.6%	21.2%	35.0%	128	0	0	72	0
>		Inst_1077925 (cv_14087)	cv_14087	0.67	3.43	2067	0.2%	12.5%	15.1%	10.1%	14.1%	48.1%	256	0	0	18	0
>		Inst_130 (cv_39)	cv_39	0.16	4.2	17216	2.6%	26.4%	22.0%	13.6%	13.1%	22.4%	60	0	0	4	0

次の表に、Rent 指数の典型的な範囲を示します。

表 10: Rent 指数の範囲

範囲	説明
0.0 ~ 0.65	低から標準。
0.65 ~ 0.85	高 (特に合計インスタンス数が 15,000 個を超える場合)。

表 10: Rent 指数の範囲 (続き)

範囲	説明
0.85 以上	非常に高い。インスタンス数も多い場合はインプリメンテーションでエラーが発生する可能性があります。

次の表に、平均ファンアウトの典型的な範囲を示します。

表 11: 平均ファンアウトの範囲

範囲	説明
4 未満	標準。
4 ~ 5	高。密集を発生させずにデザインを配置することが困難である可能性があります。 SSI テクノロジ デバイスを使用する場合、インスタンスの総数が 100,000 を超える場合は、1 つの SLR に収まる配置ソリューションまたは 2 つの SLR にまたがる配置ソリューションを見つけるのが困難になる可能性があります。
5 以上	非常に高い。インプリメンテーションでエラーが発生する可能性があります。

大型で重要度の高いモジュールで Rent 指数または平均ファンアウトが大きい場合は、対処が必要です。小型のモジュール、特にインスタンスの総数が 15,000 未満のモジュールでは、Rent 指数および平均ファンアウトが大きくても問題なく配置配線できることがあります。そのため、Rent 指数および平均ファンアウトと共に [Total Instances] 列も確認する必要があります。



ヒント: 一部の低位モジュールの Rent 指数が高く、平均ファンアウトが大きくても、最上位モジュールの複雑性メトリクスが高いとは限りません。-hierarchical_depth オプションを使用して低位モジュールを解析に含めてください。

デザイン解析レポートの複雑性レポートの生成および解析の詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニク』(UG906)の**複雑性レポート**を参照してください。

クロック スキューの削減

ザイリンクス デバイスでは、ファンアウトの大きいクロック、小さい伝搬遅延、小さいクロック スキューなどの要件を満たすため、専用配線リソースを使用した一般的なクロッキング方法がサポートされます。クロック スキューは、高周波数クロックのタイミング バジエットを大幅に削減し、またデバイスの使用率が高い場合にセットアップとホールドの両方を満たすためにインプリメンテーション ツールに過剰な負荷がかかります。

典型的なクロック スキューは、ソース クロックとデスティネーション クロックが同じタイミング パスで 300 ps 未満、バランスが取られた同期クロック間のタイミング パスで 500 ps 未満です。リソース列をまたぐと、クロック スキューの変動は大きくなり、それがタイミング スラックに反映され、インプリメンテーション ツールで最適化されます。バランスが取られていないクロック ツリー間または共通ノードのないクロック間のタイミング パスでは、クロック スキューは数ナノ秒になることがあり、タイミング クロージャを達成するのがほぼ不可能になります。

クロック スキューを削減するには、次を実行します。

1. すべてのクロックの関連性を同期クロック パスのみがタイミング解析されて最適化されるようにします。
2. 次のセクションに説明するように、クロック ツリー トポロジと、見積もりよりも大きいクロック スキューの影響を受けるタイミング パスの配置を確認します。
3. 次のセクションで説明されているように、クロック スキューを削減する手法を特定します。

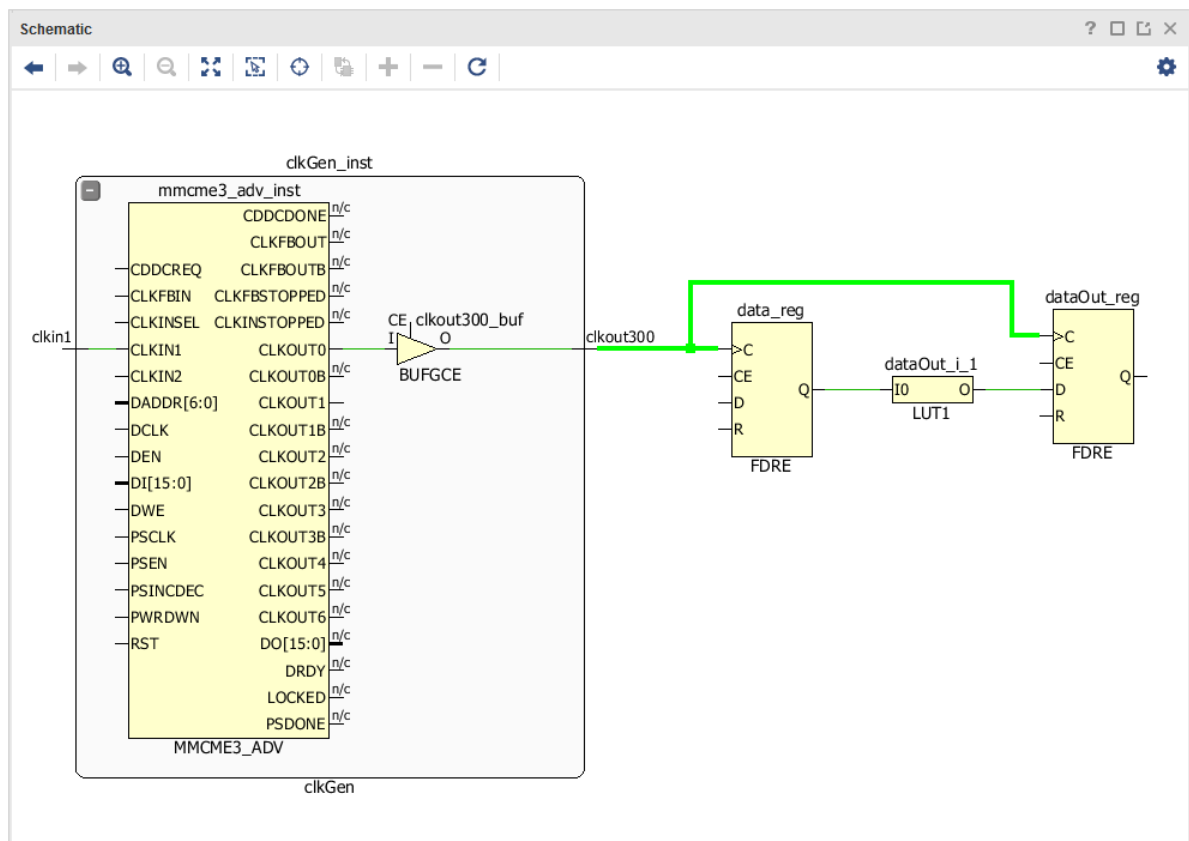
関連情報

クロック グループおよび CDC 制約の定義

ソース クロックとデスティネーション クロックが同じタイミング パスの使用

ソース クロックとデスティネーション クロックが同じで同じクロック バッファにより駆動されるタイミング パスでは、通常スキューは非常に小さくなります。これは、次の図に示すように、共通ノードが最下位クロック ピンの近くの専用クロック ネットワークに配置されているからです。

図 123: 共通ノードが緑色のネットに配置された典型的な同期クロッキング トポロジ



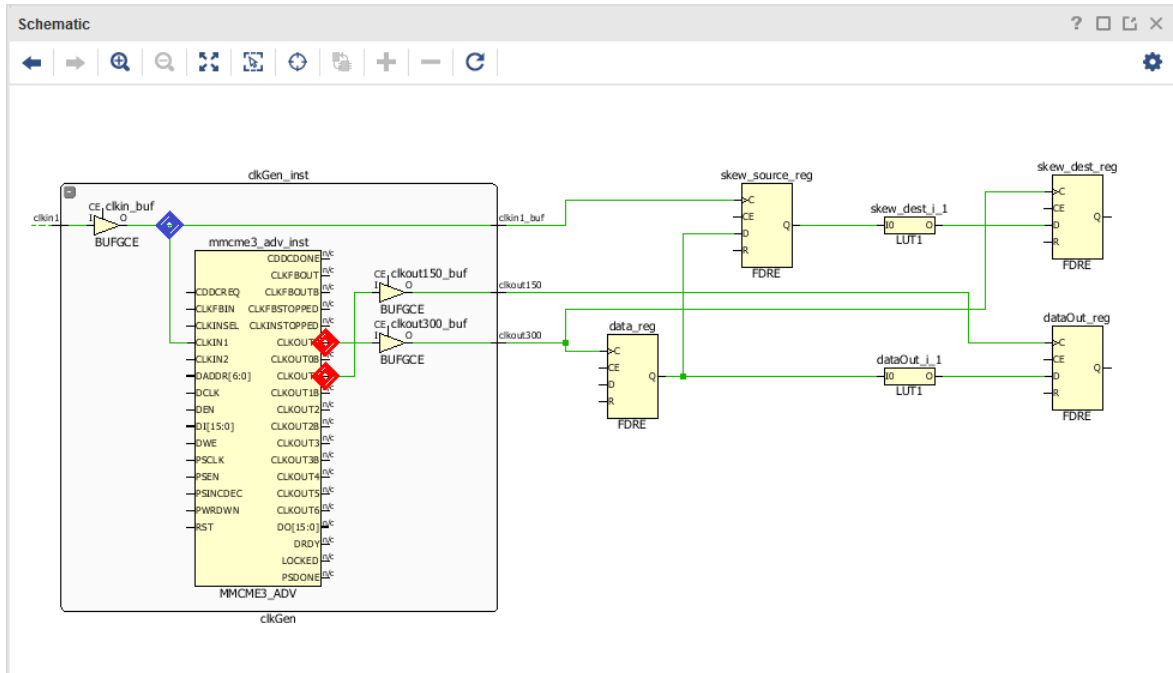
共通ノードはデザインの物理データベースにのみ含まれ、論理ビューには含まれないので、タイミング レポートのクロック パスを解析しても、共通ノードの前後の遅延は個別には示されません。そのため、共通ノードは Vivado IDE の [Device] ウィンドウで [Routing Resources] をオンにすると表示されますが、[Schematic] ウィンドウには表示されません。タイミング レポートでは、スキュー算出のサマリと、ソース クロック遅延、デスティネーション クロック遅延、共通ノードを調整する Clock Pessimism Removal (CPR) のみが示されます。

同期クロック乗せ換えパスの制限

別のクロック バッファで駆動される同期クロック間のタイミング パスでは、共通ノードがクロック バッファの前にあるので、スキューが大きくなります。つまり、共通ノードが最下位クロック ピンから離れており、タイミング解析で不必要に悪い見積もり部分が大きくなります。バランスの取られていないクロック ツリー間のタイミング パスでは、ソース クロック パスとデスティネーション クロック パス間の遅延差により、クロック スキューがさらに大きくなります。正のスキューは、セットアップ タイムを満たすのに役立つこともありますが、ホールド タイムのクロージャには悪影響を与えることがあります (逆の場合もあり)。

次の図では、3つのクロックが使用されており、クロック内のパスとクロック間のパスが複数あります。MMCMで駆動される2つのクロックの共通ノードは、MMCMの外にあります(赤色のひし形)。MMCM入力クロックとMMCM出力クロック間のパスの共通ノードは、MMCMの前のネット上にあります(青色のひし形)。MMCM入力クロックとMMCM出力クロック間のパスでは、clk_in_buf BUFGEの場所とMMCMの補正モードによって、クロックスキューが非常に大きくなることがあります。

図 124: MMCM の入力および出力上に共通ノードを持つ同期 CDC パス



ザイリンクスでは、クロック スキューが許容範囲内であっても、同期クロック乗せ換えの数を制限することをお勧めします。また、スキューが異常に大きく、削減することが不可能な場合は、ザイリンクスではこれらのパスを非同期として扱い、非同期クロック乗せ換え回路をインプリメントしてタイミング例外を追加することをお勧めします。

非同期クロック間にタイミング例外を追加

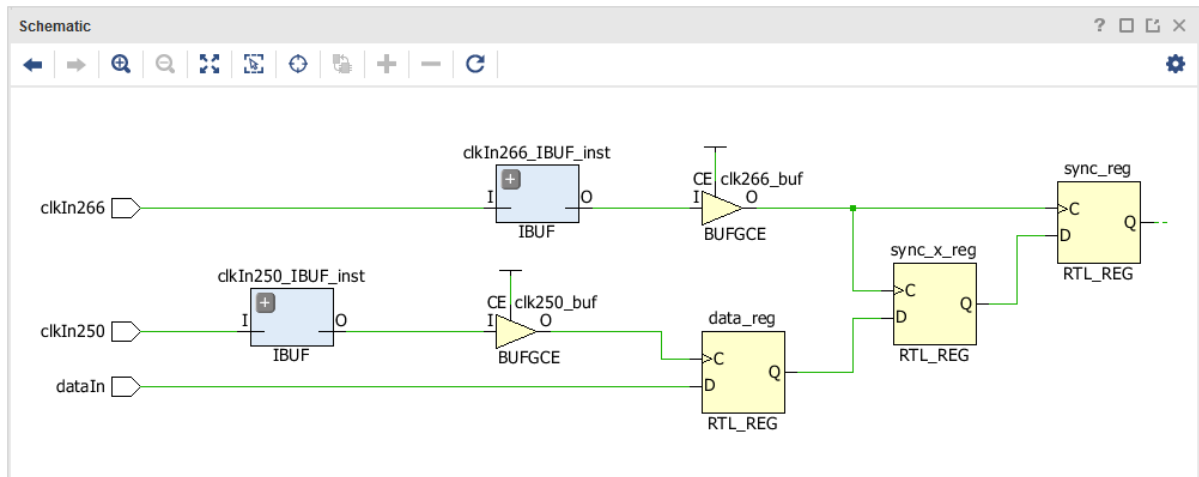
ソース クロックとデスティネーション クロックが異なるプライマリ クロックから供給されているタイミング パス、あるいは共通ノード、共通位相、または共通周期のないタイミング パス、は、非同期クロックとして扱う必要があります。この場合、スキューが極端に大きくなり、タイミング クロージャを達成するのが不可能になります。

非同期クロック間のすべてのタイミング パスを調べ、次を確認する必要があります。

- 適切な非同期クロック乗せ換え回路 (report_cdc)
- タイミング解析を無視するタイミング例外定義 (set_clock_groups、set_false_path) またはスキューを無視するタイミング例外定義 (set_max_delay -datapath-only)

クロック関連性レポート (report_clock_interaction) を使用すると、非同期で適切なタイミング例外が設定されていないクロックを特定するのに役立ちます。

図 125: 適切な CDC 回路が設定された共通ノードのない非同期 CDC パス



関連情報

クロック グループおよび CDC 制約の定義

クロック スキューを削減するための一般的な手法

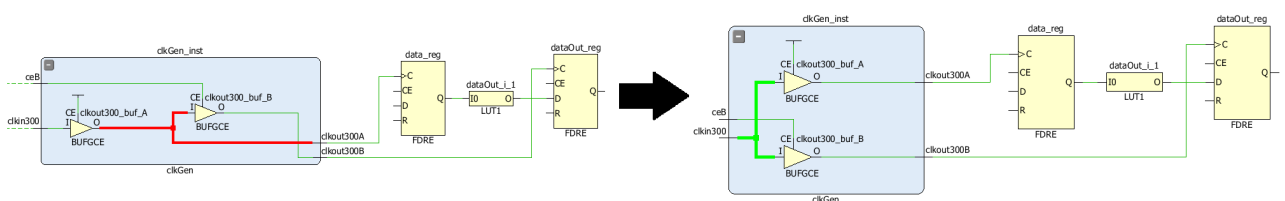


ヒント: UltraScale デバイスのクロッキング アーキテクチャの柔軟性を考慮し、`report_methodology` コマンドには最適なクロッキング トポロジを作成するのに役立つチェックが含まれています。

次の手法は、一般的な状況で使用できます。

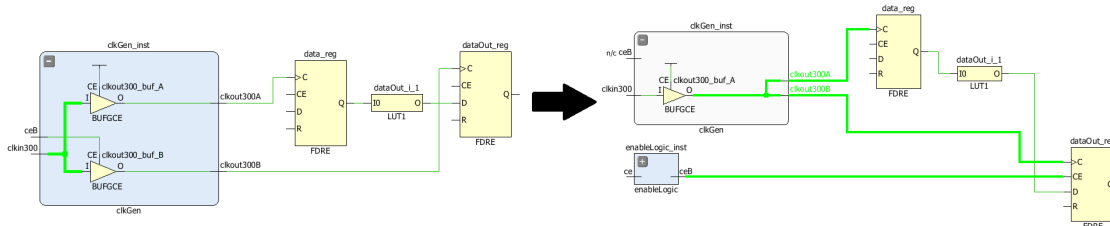
- 不要なバッファを削除するか、次の図に示すようにバッファを並列に接続することにより、カスケード接続されたクロック バッファ間のタイミング パスを回避します。

図 126: カスケード接続された BUFG を並列に接続し直した同期クロッキング トポロジ



- 次の図に示すように、並列クロック バッファを 1 つのクロック バッファに統合し、クロック バッファのクロック イネーブル ロジックを対応するシーケンシャル セルのイネーブル ピンに接続します。一部のクロックがバッファのビルトイン分周器により分周されている場合は、クロック イネーブル ロジックを使用して同等の分周をインプリメントし、必要に応じてマルチサイクル パス タイミング例外を適用します。ダウストリームのロジックで立ち上がりクロック エッジと立ち下がりクロック エッジの両方が使用される場合、または消費電力が重要な要素である場合は、この手法は適切でないことがあります。

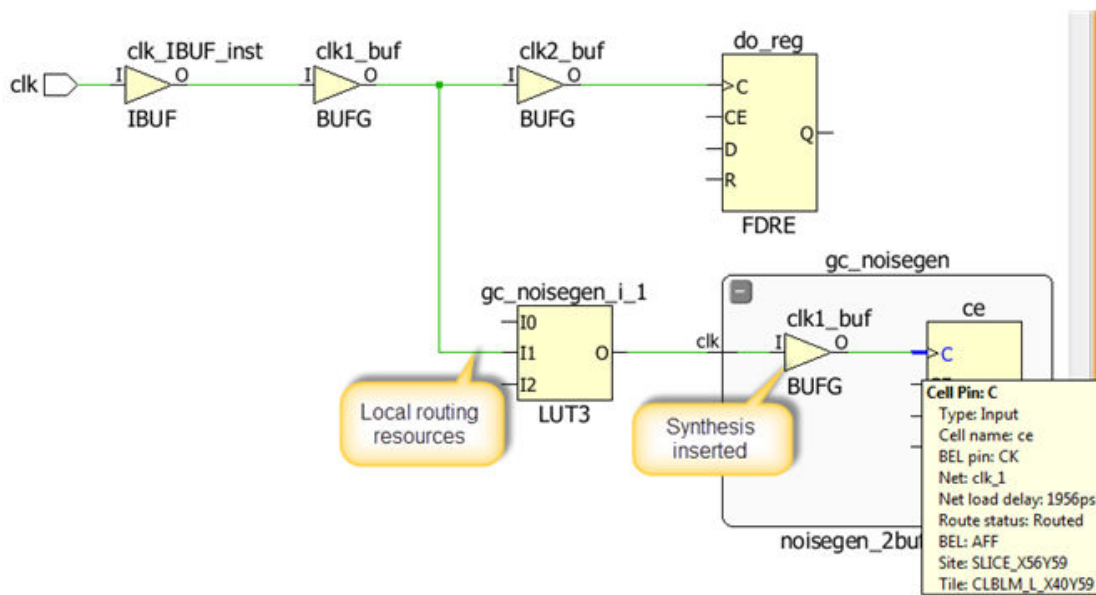
図 127: 並列クロック バッファを 1 つのバッファに統合した同期クロッキング トポロジ



- クロックパスにある LUT または組み合わせロジックは削除します。クロックパスに LUT または組み合わせロジックがあると、配置中のクロック遅延とクロック スキューが予測不可能なものになり、QoR が低下します。また、クロックパスの一部が汎用インターコネクトを使用して配線されていると、グローバル クロック リソースを使用している場合よりもノイズの影響を受けやすくなります。組み合わせロジックは通常、最適でないクロックゲーティングの変換により作成されるので、クロック バッファまたはシーケンシャルセルに接続されているクロック イネーブル ロジックに移動できます。

次の図では、最初の BUFG (clk1_buf) が LUT3 で使用され、ゲーテッド クロックが作成されています。

図 128: クロック ネットワークでローカル配線が使用されるために発生するスキュー



★ **重要:** 7 シリーズと UltraScale デバイスのクロッキング アーキテクチャは異なります。ターゲット アーキテクチャのクロッキング ガイドラインに従い、デザインがコンパイルできるかどうかを確認する必要があります。

関連情報

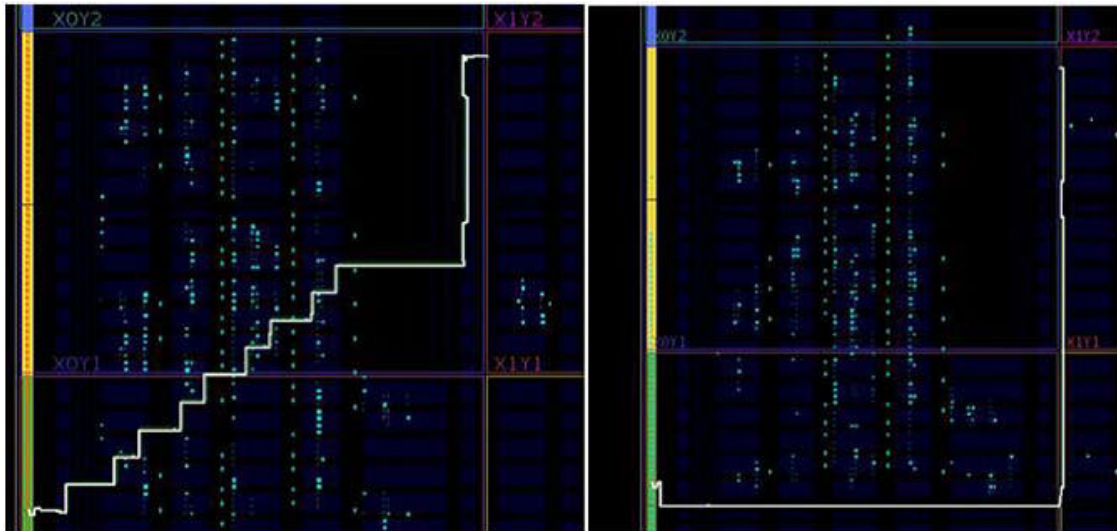
クロッキング ガイドライン

7 シリーズ デバイスでスキューを向上する手法

7 シリーズと UltraScale デバイスのクロック アーキテクチャは異なりますが、両方のファミリーに共通するクロックに関する考慮事項があります。

- プロダクション7シリーズ デザインでは、CLOCK_DEDICATED_ROUTE=FALSE 制約は使用しないでください。CLOCK_DEDICATED_ROUTE=FALSE は、デバッグ用にクロック トポロジを表示するためにデザインで配置配線を完了するためにクロック エラーを一時的に回避する場合にのみ使用してください。ファブリック インターコネクトを使用して配線されたクロック パスでは、クロック スキューが大きくなり、スイッチ ノイズの影響を受けることがあるので、パフォーマンスが悪くなったり、デザインが機能しなくなる可能性があります。次の図の右側には専用クロック配線があり、左側では専用配線はクロックで使用できないようになっています。

図 129: ファブリック クロック配線と専用クロック配線の比較

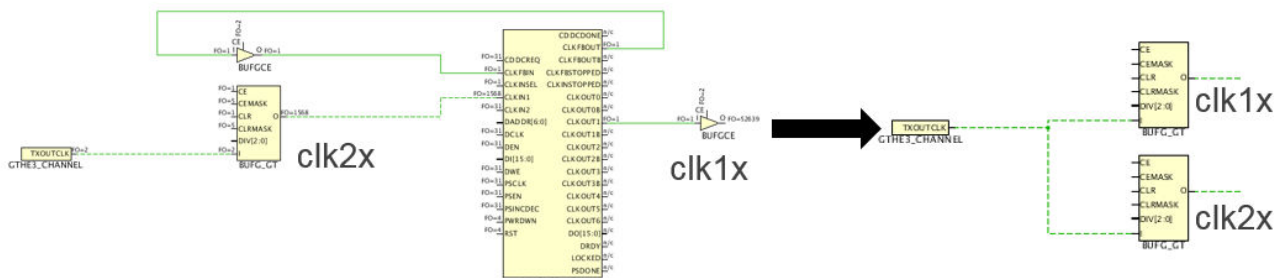


- リージョンル クロック バッファ (BUFR、BUFIO、BUFH) で複数のクロック領域にあるロジックを駆動しないでください。各クロック領域のクロック ツリー分岐間でのスキューが非常に大きくなります。不適切な LOC または Pblock 制約を削除して、この状況を解消してください。

UltraScale および UltraScale+ デバイスでのスキューの向上

- MMCM または PLL を使用して BUFG_GT クロックの単純な分周を実行しないようにします。BUFG_GT セルには入力クロックを分周する機能があります。次の図に、MMCM リソースを節約し、GTHE3_CHANNEL セルからの 2 つのクロックに対してバランスの取れたクロック ツリーをインプリメントする方法を示します。

図 130: UltraScale BUFG_GT を使用してバランスの取れたクロック ツリーをインプリメント



- 配置配線中の CLOCK_ROOT と配線が一致するようにするため、クリティカル同期クロックのドライバー ネットに CLOCK_DELAY_GROUP を設定します。この制約が適用されるようにするためには、クロック バッファを同じセルで駆動する必要があります。

注記: この最適化手法は、report_qor_suggestions Tcl コマンドにより自動的に適用されます。

- タイミング パスでタイミングを満たすことが困難で、スキューが見積もりよりも大きい場合は、タイミング パスが SLR または I/O 列をまたいでいる可能性があります。その場合は、Pblock などの物理制約を使用してソースとデスティネーションを 1 つの SLR に配置するか、I/O 列をまたがないようにします。
- 高速同期クロック乗せ換えタイミング パスでは、MMCM、PLL などのクロック調整ブロックの位置を制約してクロック ロードの中央に配置すると、タイミングを満たしやすくなります。クロック ネットワークの遅延を削減すると、クロック乗せ換えパスの不必要に悪い見積もり部分が小さくなります。
- CLOCK_DEDICATED_ROUTE=FALSE 制約が設定されたクロック ネットがグローバル クロック リソースを使用して配線されていることを確認します。FALSE の代わりに ANY_CMT_COLUMN を使用して、配線除外が設定されたクロック ネットが専用クロック リソースを使用して配線されるようにします。クロック ネットがファブリック インターコネクトを使用して配線されている場合は、この状況を解決するのに必要なデザインの変更またはクロック配置制約を特定し、インプリメンテーション ツールでグローバル クロック リソースが使用されるようにします。ファブリック インターコネクトを使用して配線されたクロック パスでは、クロック スキューが大きくなったりスイッチ ノイズの影響を受けたりすることがあるので、パフォーマンスが悪くなったり、デザインが機能しなくなる可能性があります。

関連情報

[同期 CDC](#)

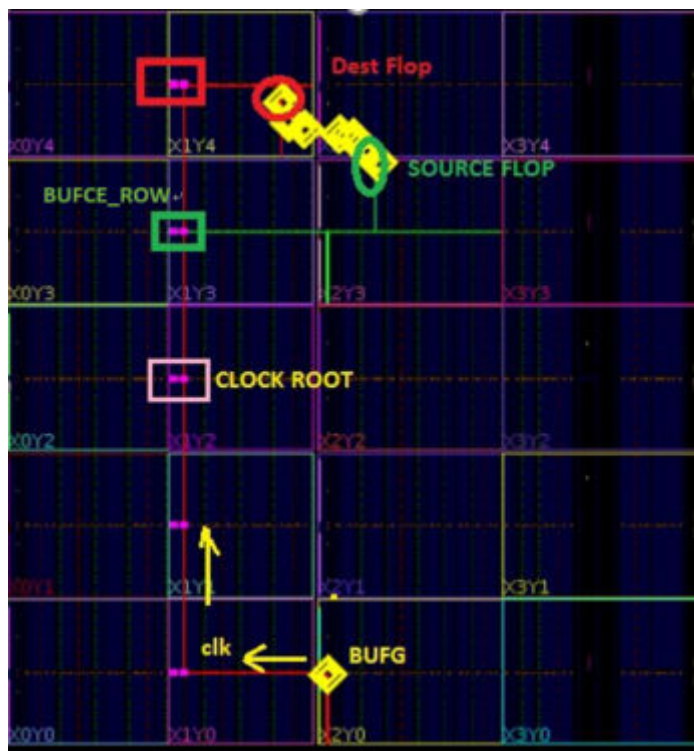
[クロック制約](#)

UltraScale および UltraScale+ デバイスでのクロック遅延の削減

UltraScale および UltraScale+™ でのグローバル クロックの配線では、まずグローバル クロック バッファから水平配線トラックと垂直配線トラックを介してクロック ルートと呼ばれる中央位置へのクロック ネットが配線されます。クロック ネットは、クロック ルートから垂直分配トラックを介して各クロック領域のクロック行に分配されます。各行には、BUFCE_ROW ルートスルー サイトのクロック ネットワークにプログラム可能な遅延があり、クロック ルートからさらに遠くに伝搬されるクロックの大まかなスキュー調整が実行されます。

次の図に、グローバル クロック バッファ (BUFG) からクロック ルートへのクロック パスを示します。クロック配線は、垂直分配トラックへの配線から、各クロック領域行の BUFCE_ROW で水平分散トラックに切り替わり、その後最下位レベルに駆動されます。ソースは緑、デスティネーションは赤で示されています。

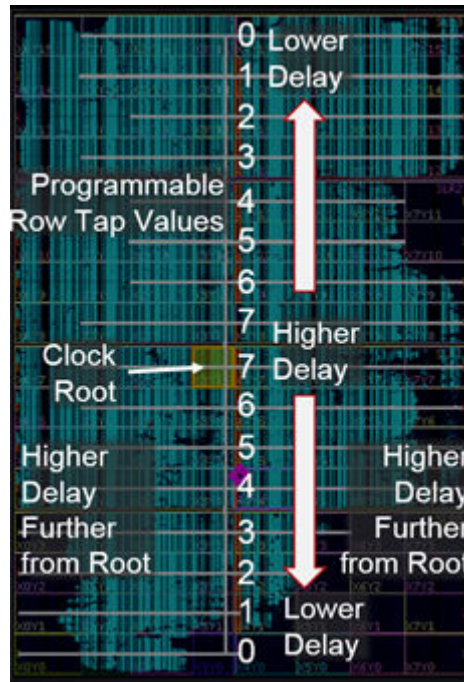
図 131: BUFG から BUFCE_ROW を介して最下位レベルに到達するクロック パス



行プログラム可能タップ遅延は、クロック ルートの近くで最大です。この遅延は、ルートから垂直方向に離れるにつれてクロック領域ごとに 1 タップ減少し、最終的に 0 になります。

次の図に、プログラム可能な行タップ値がルートから遠ざかるにつれ減少するトポロジを示します。タップ値が大きいと、製造プロセスの違いによる最小/最大遅延の変動によりタイミングのばらつきが追加されるので、遅延が大きくなり、SLR をまたぐクロック スキューも大きくなります。そのため、プログラム可能なタップ遅延値の大きいルート近くのタイミングを満たすのが困難になります。ルートからの垂直距離が長くなると、ばらつきが少なくなり、通常は SLR パスをまたぐ際のホールド違反を修正しやすくなります。SLR をまたぐバスのルートから水平距離が長くなると、クロック行遅延が大きくなります。この追加の遅延により、最小/最大遅延の変動が大きくなり、SLR をまたぐ際のパフォーマンスが低下します。

図 132: UltraScale+ SSI テクノロジ デバイスの行プログラム可能タップ遅延設定



UltraScale+ SSI テクノロジ デバイスでは、次のいずれかの方法を使用すると、SLR をまたぐスピードを向上できます。

- クロック ルートの SLR をまたぐ部分からの水平距離を近くする。
- 最大行プログラム可能タップ遅延値を制限してばらつきを削減する。

注記: ルートからの垂直距離が長いタイミング パスは、ホールド修正配線迂回からの遅延が増加するため、多少遅くなる可能性があります。これらの方法を使用すると全体的なパフォーマンスが向上します。

クロック使用率レポートの [Device Cell Placement Summary for Global Clock] セクションで、デザインの各グローバル クロックに対して Vivado ツールで選択された行プログラム可能タップ遅延設定を確認できます。次の例に、[HORIZONTAL PROG DELAY] 列に表示される g13 グローバル クロックの行プログラム可能タップ設定を、黄色でハイライトして示します。

図 133: クロック使用率レポートのグローバル クロックの行プログラム可能タップ遅延設定

22. Device Cell Placement Summary for Global Clock g13

Global Id	Driver Type/Pin	Driver Region (D)	Clock	Period (ns)	Waveform (ns)	Root (R)	Slice Loads
g13	BUFGCE/O	X4Y10	Multiple	4.926	{0.000 2.463}	X3Y8	12511

* Slice Loads column represents load cell count of all cell types other than IO, GT and clock resources

** IO Loads column represents load cell count of IO types

*** Clocking Loads column represents load cell count that are clock resources (global clock buffer, MMCM, PLL, etc)

**** GT Loads column represents load cell count of GT types

	X0	X1	X2	X3	X4	X5	X6	X7	HORIZONTAL PROG DELAY
Y15	2820	4086	308	0	0	0	0	0	0
Y14	713	392	0	0	3	0	0	0	0
Y13	0	0	0	0	0	0	0	0	0
Y12	0	0	0	0	0	0	0	0	0
Y11	0	0	0	0	0	62	94	3	3
Y10	0	0	801	3	{D} 45	224	216	0	4
Y9	0	0	252	91	361	185	10	0	5
Y8	0	0	66	{R} 261	241	2	0	0	5
Y7	0	17	365	117	16	0	0	0	4
Y6	0	165	275	39	2	0	0	0	3
Y5	0	120	66	0	0	0	0	0	2
Y4	0	27	7	0	0	0	0	0	1
Y3	21	21	0	3	0	0	0	0	0
Y2	11	1	0	0	0	0	0	0	0
Y1	0	0	0	0	0	0	0	0	0
Y0	0	0	0	0	0	0	0	0	0

UltraScale+ SSI テクノロジ デバイスでは、最小/最大遅延の変動を低減してクロック ルート付近の SLR をまたぐクロック スキューを低減するため、配置で最大行プログラム可能タップ遅延値が制限されると同時に、ルートから遠くにある SLR をまたぐパスのクロック スキューのバランスを取るため、SLR 境界の両側にあるクロック領域が増加または減少タップ遅延を持つようになります。配置で使用される最大行プログラム可能タップ遅延値を調べるには、クロック ネットの MAX_PROG_DELAY プロパティをクエリします。

行プログラム可能タップ遅延を制限するには、USER_MAX_PROG_DELAY プロパティを使用します。次に例を示します。USER_MAX_PROG_DELAY プロパティを設定するには、グローバル クロック バッファで直接駆動されるネット セグメントに値を適用する必要があります。USER_MAX_PROG_DELAY プロパティを設定しない場合、配置で可能な最大タップ設定 7 が使用される可能性があります。

```
set_property USER_MAX_PROG_DELAY <0-7> [get_nets -of [get_pins BUFG/O]]
```

次に、USER_MAX_PROG_DELAY プロパティを使用する際のヒントを示します。

- UltraScale+ SSI テクノロジ デバイスの大部分に分配されるクロックに推奨される USER_MAX_PROG_DELAY タップ値は 3 または 4 です。デバイスの中央でない GT、PCIe[®]、または CMAC ブロックの近くにクロック ルートがある場合、ソース クロックとデスティネーション クロックの共通ノードが SLR をまたぐ部分から遠くなるので、デバイスの反対側の SLR をまたぐパフォーマンスが大きく影響を受けます。
- クロック ネットワーク一致のため CLOCK_DELAY_GROUP を使用するクロック グループでは、クロック グループ内のすべてのクロックに同じ USER_MAX_PROG_DELAY 値を使用してください。

クロックのばらつきの削減

クロックのばらつきは、理想的なクロックに対するばらつきの量です。ユーザー指定の外部クロックのばらつき (set_clock_uncertainty)、システム ジッター、またはデューティ サイクルの歪みが原因で発生します。MMCM や PLL などのクロック調整ブロックでも、ディスクリート ジッター、複数の関連クロックが使用される場合の位相エラーの形でクロックのばらつきが生成されます。

Clocking Wizard では、指定のデバイスの正確なばらつきデータが提供され、異なるトポロジを比較するためにさまざまな MMCM クロッキング構成を生成できます。ザイリンクスでは、ターゲット アーキテクチャで最適な結果を得るには、以前のアーキテクチャからのレガシ クロック生成ロジックを使用するのではなく、Clocking Wizard を使用してクロック生成ロジックを再生成することをお勧めします。

MMCM 設定を使用したクロックのばらつきの削減



ヒント: この問題は、report_qor_suggestions Td コマンドでレポートされます。

MMCM を周波数合成用に設定する際は、ザイリンクスではクロックのジッターが最小になるように設定することをお勧めします。MMCM を、電圧制御オシレーター (VCO) のデバイスの動作範囲を満たす最大周波数で動作するよう最適化します。次に、VCO 周波数、M (通倍値)、D (分周値)、および O (出力分周値) と、入力および出力クロック周波数との関係式を示します。

$$F_{VCO} = F_{CLKIN} \times \frac{M}{D}$$

$$F_{OUT} = F_{CLKIN} \times \frac{M}{D \times O}$$



ヒント: VCO 周波数を上げるには、M を大きくするか D を小さくし (またはその両方)、周波数の変更を O を大きくすることにより補正します。VCO 周波数を上げると、MMCM または PLL からの消費電力が増加します。また、BUFG を使用する複数の MMCM クロック出力から BUFGCE_DIV を使用する 1 つのクロック出力に切り替え、VCO 周波数を少しだけ増加することもできます。これにより、分数分周を使用するクロックの数を増やすことができます。MMCM または PLL のどちらにするかを選択する場合、MMCM の方が高い VCO 周波数で動作でき、M と D の値をより細かく選択でき、分数分周 (CLKOUT0) があるので、MMCM の方が好まれます。

アーキテクチャによって最大 VCO 周波数は異なります。そのため、ザイリンクスではターゲット アーキテクチャ用にクロッキング コンポーネントが最適なものになるよう再生成することをお勧めします。ザイリンクスでは、MMCM をターゲット デバイス用に正しく設定するため、Clocking Wizard で VCO 周波数と共に M と D の値も自動計算されるようにすることをお勧めします。



ヒント: IP カタログの Clocking Wizard を使用する場合は、[Jitter Optimization] で [Minimize Output Jitter] をオンにしてください。この設定により、高い方の VCO 周波数が選択されます。また、出力クロック周波数要件を多少変更することにより、VCO 周波数が上がってクロックのばらつきが削減されることがあります。

次の MMCM 周波数合成の例では、62.5 MHz の入力クロックを使用して約 40 MHz の出力クロックを生成しています。2 つのソリューションがありますが、VCO 周波数が高いほうの MMCM_2 の方がジッターと位相エラーが削減されるので、クロックのばらつきが小さくなります。

表 12: MMCM 周波数合成の例

	MMCM_1	MMCM_2
入力クロック	62.5 MHz	62.5 MHz
出力クロック	40.0 MHz	39.991 MHz

表 12: MMCM 周波数合成の例 (続き)

	MMCM_1	MMCM_2
CLKFBOUT_MULT_F(M)	16	22.875
DIVCLK_DIVIDE(D)	1	1
VCO 周波数	1000.000 MHz	1429.688
CLKOUT0_DIVIDE_F(O)	25	35.750
ジッター (ps)	167.542	128.632
位相エラー (ps)	384.432	123.641

BUFGCE_DIV を使用したクロックのばらつきの削減

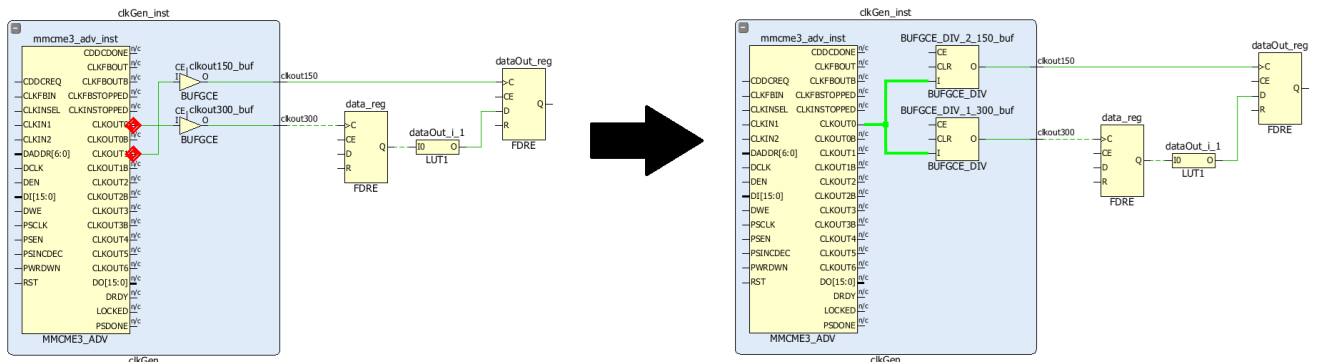


ヒント: この問題は、report_qor_suggestions Tcl コマンドでレポートされます。

UltraScale デバイスでは、BUFGCE_DIV セルを使用して、MMCM 位相エラーを取り除くことにより同期クロック乗せ換えのクロックのばらつきを削減できます。たとえば、300 MHz および 150 MHz クロック ドメイン間のパスがあり、両方のクロックが同じ MMCM で生成されているとします。

この場合、セットアップおよびホールド解析の両方で、クロックのばらつきに 120 ps の位相エラーが含まれます。150 MHz クロックを MMCM を使用して生成する代わりに、MMCM の 300 MHz 出力に BUFGE_DIV を接続し、クロックを 2 で分周できます。最適な結果を得るには、次の図に示すように、300 MHz クロックにも BUFGE_DIV を BUFGE_DIVIDE を 1 に設定して使用し、150 MHz クロックの遅延に正確に一致させる必要があります。

図 134: UltraScale の同期 CDC タイミング パスのクロック トポロジの向上



新しいトポロジには、次の特徴があります。

- セットアップ解析では、クロックのばらつきに MMCM 位相エラーは含まれず、120 ps 削減されています。
- ホールド解析では、クロックのばらつきはありません (同じエッジのホールド解析の場合のみ)。
- 共通ノードがバッファの近くに移動し、不必要に悪い見積もり部分が削減されます。

2 つのクロック ネットに CLOCK_DELAY_GROUP 制約を適用することにより、クロック パスの配線が一致します。

注記: これらの制約は、report_qor_suggestions Tcl コマンドにより供給されます。

次の表に、UltraScale の同期 CDC タイミング パスのセットアップおよびホールド解析でのクロックのばらつきの比較を示します。

表 13: UltraScale の同期 CDC タイミング パスのセットアップ解析におけるクロックのばらつきの比較

セットアップ解析	MMCM で生成された 150 MHz クロック		BUFGCE_DIV 150 MHz クロック
	トータル システム ジッター (TSJ)	0.071 ns	0.071 ns
	ディスクリート ジッター (DJ)	0.115 ns	0.115 ns
	位相エラー (PE)	0.120 ns	0.000 ns
	クロックのばらつき	0.188 ns	0.068 ns

表 14: UltraScale の同期 CDC タイミング パスのホールド解析におけるクロックのばらつきの比較

ホールド解析	MMCM で生成された 150 MHz クロック		BUFGCE_DIV 150 MHz クロック
	トータル システム ジッター (TSJ)	0.071 ns	0.000 ns
	ディスクリート ジッター (DJ)	0.115 ns	0.000 ns
	位相エラー (PE)	0.120 ns	0.000 ns
	クロックのばらつき	0.188 ns	0.000 ns

関連情報

[同期 CDC](#)

一般的なタイミング クロージャ手法

次の手法は、困難なデザインでデザイン クロージャを達成するのに役立ちます。これらの手法を試す前に、デザインが適切に制約されており、違反が大きいパスに影響する主な問題を特定する必要があります。



推奨: ザイリンクスでは、`report_qor_suggestions` Tcl コマンドを実行してこれらの多くの手法を自動的に特定して適用することをお勧めします。詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906)の[このセクション](#)を参照してください。

ブロック レベル合成ストラテジを使用したネットリストの向上

ほとんどのデザインではデフォルトの Vivado 合成設定でタイミング要件を満たすことができますが、大型で複雑なデザインでは通常、タイミング クロージャを達成するために異なる階層に合成ストラテジを組み合わせ需要使用する必要があります。

たとえば、1つのモジュールではタイミング クリティカルなファンクションをインプリメントするのに MUXF* リソースの使用が必要であり、デザインのその他の部分ではロジックを MUXF* にインプリメントするよりも LUT にインプリメントした方が密集を削減できる場合などです。この場合、タイミング クリティカルなモジュールに PERFORMANCE_OPTIMIZED ストラテジを使用し、デザインのその他の部分は密集を削減するために Flow_AlternateRoutability ストラテジを使用して合成します。

関連情報

[ブロック レベル合成ストラテジ](#)

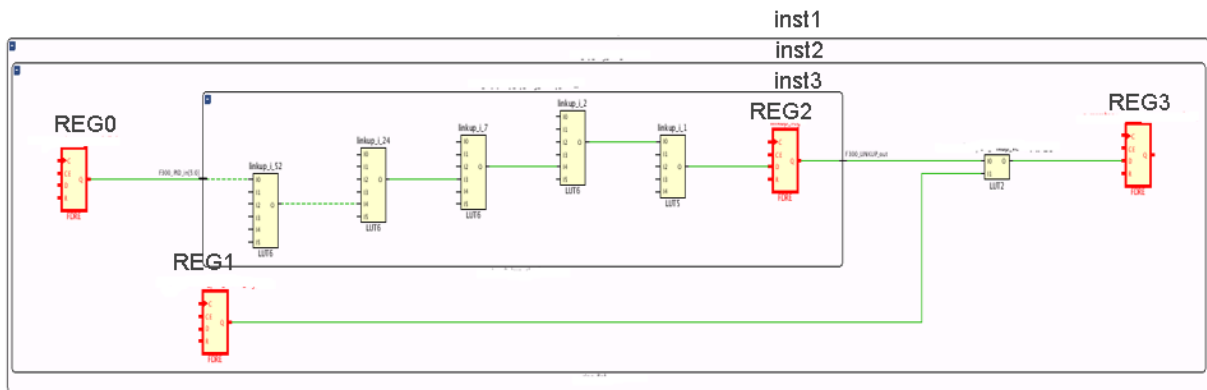
ロジック段数の削減

デザイン サイクルを通して、ロジック段の分布がターゲットのザイリンクス デバイス ファミリとデバイス スピード グレードのクロック周波数要件を達成するのに適切であるかを確認する必要があります。ロジック段数の多いパスの数が限られる場合は必ずしもタイミング クロージャの問題が発生するわけではありませんが、Vivado 合成のリタイミング オプションを使用してデザインの最長のパスを最適化すると、タイミング QoR (結果の品質) を向上できます。

リタイミング オプションをグローバルに適用すると、通常実行時間が長くなり、消費電力が増加することがあります。そのため、ザイリンクスでは合成後または配置後にロジック段数の多いパス上に違反のある階層を特定することをお勧めします。最長のパスのファンインまたはファンアウトにあるパスのロジック段数が少なく、小型または中型の階層モジュールに含まれる場合は、BLOCK_SYNTH.RETIMING ブロック レベル合成ストラテジを使用できます。

次の図に、600 MHz クロックで制約されている 5 つの LUT を含むクリティカル パスを示します。REG2 デスティネーション フリップフロップは、REG2 から 1 つ上の階層に含まれる 1 つの LUT を含むタイミング パスを駆動しています。

図 135: ロジック段数が 5 のクリティカル パスを示す回路図



Vivado IDE の [Schematic] ウィンドウを使用するのに加え、`report_design_analysis -logic_level_distribution` コマンドを使用して特定のパスのロジック段の分布を確認できます。これにより、タイミング QoR を向上するためにバランスを取り直す必要のあるパスの数を判断できます。

Vivado 合成で `retiming_forward` および `retiming_backward` 属性を使用し、特定のレジスタまたはパスのリタイミング最適化を制御できます。これらの属性を使用すると、最上位モジュールまたはサブモジュールではなく、特定のパスにリタイミング最適化が適用されます。これらの属性は、RTL または XDC ファイルで設定できます。これらの属性の使用方法および制限などの詳細は、『Vivado Design Suite ユーザー ガイド: 合成』 (UG901) を参照してください。

次の図は、inst1/inst2 階層に含まれる 600 MHz クロックで制約されたロジック段数が 5 のパスが 58 個あり、ロジック段数が 1 のパスが 32 個あることを示しています。

図 136: デフォルトの合成最適化によるロジック段の分布

```
current_instance inst1/inst2
report_design_analysis -timing -logic_level_distribution -of_timing_paths [get_timing_paths -max_paths 100 -group core_clk_600]
```

End Point Clock	Requirement	0	1	2	3	4	5	6	7	8	9	10	11-15	16-20	21-25	26-30	31+
core_clk_600	1.667ns	0	32	10	0	0	58	0	0	0	0	0	0	0	0	0	0

Vivado 合成では、ロジック段数の少ないパスからロジック段数の多いパスにレジスタを移動することにより、ロジック段数のバランスを取ることができます。この例では、合成 XDC ファイルに次の制約を追加すると、inst1/inst2 階層でリタイミングを実行できます。

```
set_property BLOCK_SYNTH.RETIMING 1 [get_cells inst1/inst2]
```

同じグローバル設定とアップデートした XDC ファイルで合成を再実行した後、inst1/inst2 タイミング パスに対して通常のタイミング解析を実行するか `report_design_analysis` コマンドを再実行して、次の図に示すように最長パスのロジック段数が減少しているかどうかを確認できます。クリティカル パスは REG0 > 3 つの LUT > REG2 になっており (バックワード リタイミング)、REG2 から REG4 へのパスのロジック段数は 3 です。

図 137: 合成最適化でリタイミングをイネーブルにした後のロジック段の分布

```
current_instance inst1/inst2
report_design_analysis -timing -logic_level_distribution -of_timing_paths [get_timing_paths -max_paths 100 -group core_clk_600]
```

End Point Clock	Requirement	0	1	2	3	4	5	6	7	8	9	10	11-15	16-20	21-25	26-30	31+
core_clk_600	1.667ns	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0

制御セットの削減



ヒント: この最適化手法は、`report_qor_suggestions` Tcl コマンドにより自動的に適用されます。

通常、リセットやクロック イネーブルなどの制御信号についてはそれほど考慮されません。リセットが必要かどうかを判断せずに、HDL コードを `if reset` 文で開始する例がよく見られます。リセットおよびクロック イネーブルはすべてのレジスタでサポートされますが、これらの使用が最終的なインプリメンテーションのパフォーマンス、使用率、および消費電力に大きく影響します。

考慮すべき第一の要素は、制御セットです。制御セットは、1 つのシーケンシャル セルで使用されるクロック、イネーブル、およびセット/リセットのグループです。たとえば、同じクロックに接続されている 2 つのセルであっても、1 つのセルのみがリセットまたはクロック イネーブルに接続されている場合は、2 つのセルの制御セットは異なるものになります。定数または未使用のイネーブルおよびセット/リセット レジスタも、制御セットを形成します。

考慮すべき 2 つ目の要素は、ターゲット アーキテクチャです。一緒にパックできる制御セットの数は、アーキテクチャによって異なります。

- 7 シリーズ デバイスのスライス (CLB の半分) は 8 つのレジスタで構成され、これらすべてのレジスタで 1 つのクロック、1 つのセット/リセット、および 1 つのクロック イネーブルが共有されます。8 つのレジスタのグループごとに使用できる制御セットは 1 つのみです。
- UltraScale デバイスの CLB の半分は 4 つのレジスタのグループ 2 つで構成され、これらすべてのレジスタで 1 つのクロック、1 つのセット/リセットが共有されます。4 つのレジスタの各グループには 1 つのクロック イネーブルがあり、セット/リセットは無視できます。定数セット/リセット信号は配線されず、無視できます。定数イネーブル信号はダイナミック イネーブル信号と同様に扱われ、配線される必要があります。最適な状況では、8 つのレジスタのグループごとに 2 つの制御セットを使用できます。

制御セットにより CLB へのパックが制限される場合、一部のレジスタ (入力 LUT を含む) が移動されます。レジスタが最適でない場所に移動されることもあります。距離が長くなると、ロジックの分散 (長いネット遅延) および高いインターコネクト リソース使用率のため、使用率だけでなく配置 QoR にも悪影響が出ます。これは主に、クロック イネーブルが 1 つのレジスタに供給されるなど、ファンアウトが小さい制御信号が多数あるデザインで考慮する必要があります。

UltraScale デバイスでは CLB で使用可能な制御セットの数は多くなりますが、一般的なデザインでは制御セットの使用量は 7 シリーズ デザインと同様です。そのため、ザイリンクスの推奨事項は両方のアーキテクチャで同じです。

制御セットのガイドライン

次の表に、7 シリーズと UltraScale デバイスで、ターゲット デバイスのサイズ別に推奨される制御セット数のガイドラインを示します。

表 15: 制御セットのガイドライン

ガイドライン	制御セットの割合
可	デバイスの制御セットの総数の 7.5% 未満
削減することを推奨	デバイスの制御セットの総数の 7.5% ~ 15%
削減が必要	デバイスの制御セットの総数の 15% を超える

これらのガイドラインは、次を前提としています。

- 一般的に使用可能な制御セット数: 8 個の CLB レジスタごとに 1 つ
- デバイスの制御セットの総数: CLB レジスタ数/8

デザインの制御セットの数を調べるには、次のコマンドを使用します。

- 配置前: `report_control_sets -verbose`
- 配置後: `report_utilization` (テキスト モードのみ)



ヒント: デザインの小さな領域に多数の固有制御セットがあると、デバイスの対応するエリアでネット遅延または密集が発生する原因となることがあります。固有の制御セットが密集しているエリアを特定するには、Vivado IDE の [Device] ウィンドウで制御信号を異なる色でハイライトするなどして、配置を詳細に解析する必要があります。

制御セット数の削減

制御セット数が多い場合は、次のいずれかのストラテジを使用して削減します。

- HDL ソース ファイルまたは制約ファイルで制御信号に設定されている `MAX_FANOUT` 属性を削除します。制御信号を複製すると、固有の制御セットの数が大幅に増加します。ザイリンクスでは、`place_design` で実行される大まかな複製を利用し、配置後の詳細な複製には `phys_opt_design -directive Explore` を使用することをお勧めします。これにより、配線の密集の原因となる不要な複製や等価制御セットが相互にまたがるのを回避できます。
- Vivado 合成 (またはその他の合成ツール) で制御セットのしきい値を増加します。`report_control_sets -verbose` で生成された制御セットのファンアウトの分布の表を確認し、合成で使用するのにより適切な制御セット数のしきい値を判断します。`control_set_opt` を増加すると、アクティブに消費電力を削減できるクロックイネーブルが削除されるので、消費電力に悪影響を与える可能性があることに注意してください。次に例を示します。

```
synth_design -control_set_opt_threshold 16
```



ヒント: 配置の分散または密集の影響を最も受けているモジュールの制御セット数のしきい値を変更するには、`BLOCK_SYNTH` 合成制約を使用します。

- 合成後に `opt_design -control_set_merge` または `opt_design -merge_equivalent_drivers` を使用して、等価の制御セットを統合します。

- CONTROL_SET_REMAP プロパティを使用して、レジスタの同期セット/リセットまたは CE ピンを駆動するファンアウトの小さい制御信号を D 入力にマップします。詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』(UG904) のこのセクションを参照してください。
- ファンアウトの小さい非同期セット/リセット (プリセット/クリア) は、専用非同期ピンのみに接続可能であり、合成でデータパスに移動することはできないので、使用しないようにします。そのため、合成の制御セットの小さい値は非同期セット/リセットには適用されません。
- 異なるシーケンシャル セルに対して、アクティブ High およびアクティブ Low の両方の制御信号を使用しないようにします。
- クロック イネーブルおよびセット/リセットは必要な場合にのみ使用します。通常データパスには多数のレジスタが含まれ、これらのレジスタの初期化されていない値は自動的にフラッシュされるので、セット/リセットまたはイネーブル信号は最初と最後の段にのみ必要です。

関連情報

制御信号および制御セット

ファンアウトの大きいネットを最適化

ファンアウトの大きいネットは、インプリメンテーションの問題の原因となることがよくあります。デバイス ファミリのダイ サイズが大きくなると、ファンアウトの問題も増加します。終点が数千個あるネットでは、特にパス上に追加のロジックがあったり、LUT や分散 RAM などのシーケンシャルでないセルで駆動される場合、タイミングを満たすことは困難です。

レジスタの複製の使用

ほとんどのツールでは、レジスタを複製してクリティカル パス上のファンアウトの大きいネットを削減できます。または、特定のレジスタまたは階層レベルに属性を適用して、複製可能なレジスタと不可能なレジスタを指定することもできます。たとえば、複製されたネットに LUT1 があると、属性または制約が最適化の妨げになります。合成中、最適化されたネット上にある階層セルの KEEP_HIERARCHY 属性、または異なる階層のネット セグメントにある KEEP 属性により、複製最適化が変更されることがあります。合成およびインプリメンテーション中は、DONT_TOUCH 制約によっても有益な最適化が妨げられます。

ファンアウトの大きいネットは、特定のネットに MAX_FANOUT 属性を使用して RTL または合成で対処されることがあります。これは、特に MAX_FANOUT 属性の値が小さすぎる場合や複数の主要な階層に接続されているネットに設定されている場合、最適な配線リソースが使用されるとは限りません。さらに、ファンアウトの大きい信号がレジスタの制御信号であり、必要以上に複製されると、タイミング クロージャには不要なレジスタが追加され、制御セットの数が多くなってデザインの消費電力が増加することがあります。

ファンアウトを削減するには、多くの場合、ファンアウトの大きい信号に対してバランスの取れたツリーを使用するのがより良い方法です。1 つの階層に含まれるセルは通常一緒に配置されるので、デザイン階層に基づいてレジスタを手動で複製することを考慮してください。

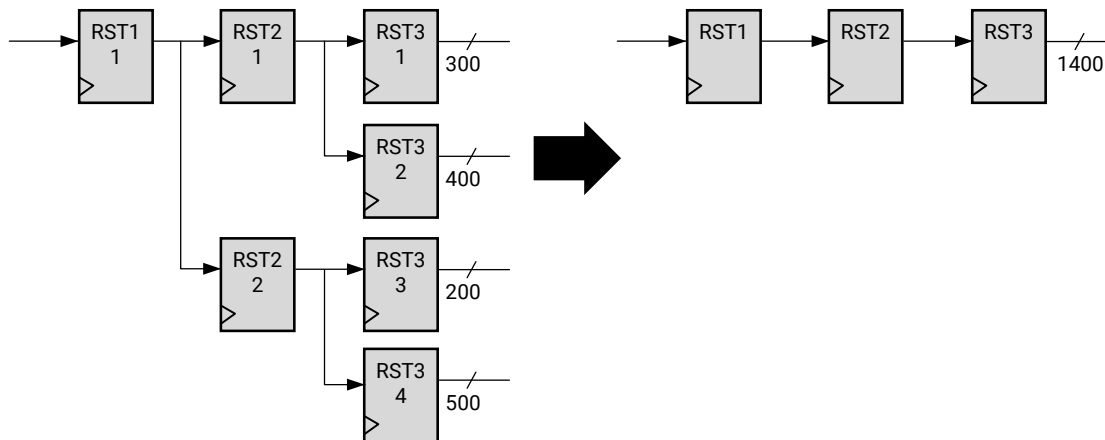
再構成して制御セットおよびファンアウトの大きいネットを削減するには、opt_design Tcl コマンドの次のいずれかのオプションを指定して使用できます。

- -control_set_merge: 論理的に等価の制御信号の複数のドライバーを積極的に 1 つのドライバーに統合します。
- -merge_equivalent_drivers: 制御信号を含む論理的に等価の信号のドライバーを 1 つのドライバーに統合します。

注記: このオプションを使用すると、ツールで主な階層と Pblock 制約が認識されるので、まずこのオプションを試してみてください。

これらのオプションはファンアウト複製の逆であり、ネットがモジュールベースの複製に適したものになります。この統合は、次の図に示すように複数段のリセット ツリーでも機能します。

図 138: `opt_design -control_set_merge` を使用した制御セットの統合



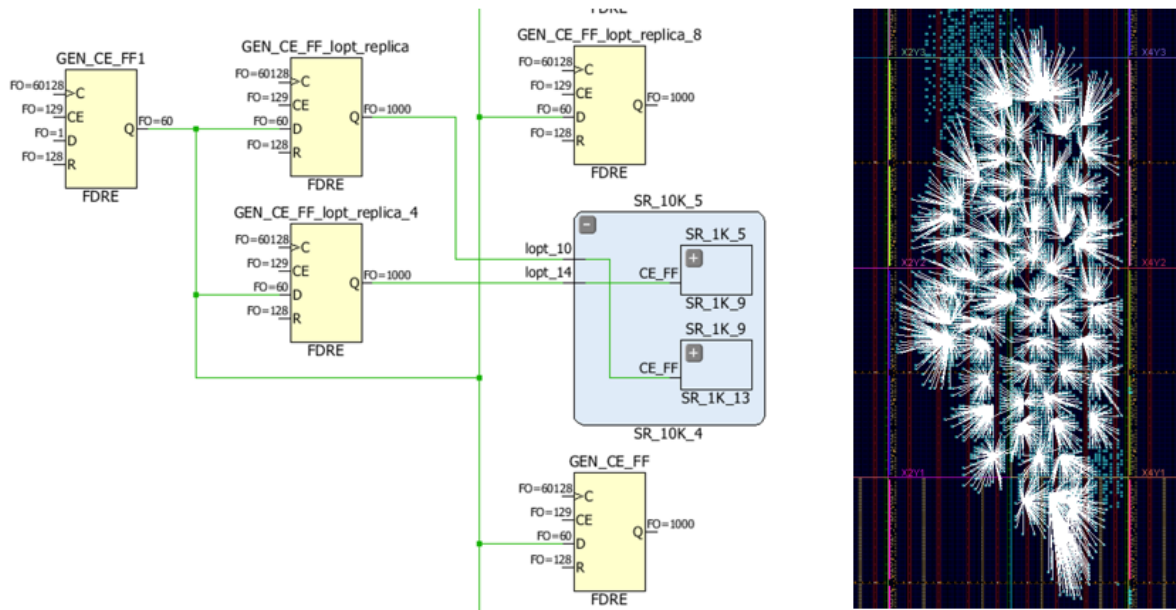
X20035-122019

複製されたオブジェクトの数を削減したら、`opt_design Tcl` コマンドで次のオプションを使用して、階層の特性に基づいて制限的な複製を実行できます。

- `-hier_fanout_limit <arg>`: 階層に応じてレジスタを複製します。`<arg>` は論理階層に応じた複製のファンアウトの制限を指定します。ファンアウトの大きいネットで駆動される階層インスタンスで、階層内のファンアウトが指定の値以内である場合、階層内のネットはファンアウトの大きいネットの複製されたドライバーにより駆動されます。複製されたドライバーは元のドライバーと同じ階層レベルに配置され、複製は制御セット レジスタに制限されません。

次の図に、`opt_design -hier_fanout_limit 1000` を使用して、ファンアウトが 60000 のクロック イネーブル ネットを複製したところを示します。各モジュール `SR_1K` には 1000 個のロードが含まれるので、ドライバーは 59 回複製されます。

図 139: ファンアウトの大きいクロック イネーブル ネットのモジュール ベースの複製



ファンアウト最適化は、place_design でデフォルトでイネーブルになっています。複製は配置フローの初期に実行され、配置情報に基づきます。1000 個を超えるロードを駆動するレジスタや、DSP、ブロック RAM、および UltraRAM を駆動するレジスタは複製の対象として考慮され、複製が実行された場合はロードと一緒に配置されます。FORCE_MAX_FANOUT プロパティをネットに追加すると、ネットを駆動するレジスタまたは LUT を強制的に複製できます。FORCE_MAX_FANOUT の値は、複製最適化後のネットに必要な最大物理ファンアウトを指定します。

MAX_FANOUT_MODE プロパティを使用すると、物理デバイス属性に基づいて複製を強制できます。MAX_FANOUT_MODE プロパティにサポートされる値は、CLOCK_REGION、SLR、MACRO です。たとえば、MAX_FANOUT_MODE プロパティに CLOCK_REGION 値を指定すると、物理クロック領域に基づいてドライバーが複製され、同じクロック領域に配置されたロードがまとめられます。詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 (UG904) のこのセクションを参照してください。

SSI テクノロジ デバイスでは、ファンアウトの大きいドライバーを各 SLR で複製し、Pblock を使用してそのロードと共に同じ SLR 内に割り当てます。この手法を使用すると、SLR 間をまたぐ遅延による影響を削減し、複製されたファンアウトの大きいネットを各 SLR 内で個別により柔軟に配置できます。

関連情報

ファンアウトの大きいネット ドライバーの複製

ファンアウトの大きいネットをグローバル配線に移動



ヒント: この最適化手法は、report_qor_suggestions Tcl コマンドにより自動的に適用されます。

ファンアウトの大きいネットのパフォーマンスが低い場合は、ドライバーとロードの間にクロック バッファを挿入することにより、グローバル配線に移動できます。この最適化は、使用されているクロック バッファの数が少なく、ネットで駆動されるロジックのクロック周期がターゲット デバイスおよびスピード グレードの制限を超える場合にのみ、ファンアウトが 25000 を超えるネットに対して opt_design で自動的に実行されます。

RTL ファイルまたは制約ファイル (XDC) で `CLOCK_BUFFER_TYPE` 属性を設定すると、`synth_design` および `opt_design` でクロック バッファの挿入を強制的に実行できます。次に例を示します。

```
set_property CLOCK_BUFFER_TYPE BUFG [get_nets netName]
```

グローバル クロック配線を使用すると、ネット遅延は大きくなりますが、配線が最適なものになります。最高のパフォーマンスを得るには、クロック バッファで中間組み合わせロジックを介さずにシーケンシャル ロードを直接駆動します。ほとんどの場合、`opt_design` でシーケンシャルでないロードが並列にクロック バッファに再接続されます。必要に応じてクロック バッファに `DONT_TOUCH` を適用すると、この最適化が実行されるのを防ぐことができます。ファンアウトの大きいネットが制御信号である場合、ロードが専用クロック イネーブルまたはセット/リセット ピンでない理由を調べる必要があります。

配置では、ファンアウトの大きいネット (ファンアウト > 10000) もクロックの配線が実行された後に使用可能なグローバル配線に自動的に配線されます。この最適化は配置フローの後の方で実行され、タイミングが悪化しない場合のみ実行されます。この機能は、`-no_bufg_opt` オプションを使用するとディスエーブルになります。

関連情報

[制御信号および制御セット](#)

物理最適化の使用

物理最適化 (`phys_opt_design`) を使用すると、スラックおよび配置情報に基づいてファンアウトの大きいネットドライバが自動的に複製され、通常はタイミングが大幅に改善されます。ザイリンクスでは、ファンアウトの大きいネットはファブリック レジスタで駆動することをお勧めします。そうすると、物理最適化中に複製および移動しやすくなります。

場合によっては、デフォルトの `phys_opt_design` コマンドでファンアウトの大きいすべてのクリティカル ネットが複製されないことがあります。`-directive` オプションの `Explore`、`AggressiveExplore`、`AggressiveFanoutOpt` 指示子などを使用して、コマンドの-effort を増加してみます。また、ファンアウトの大きいネットが配線中にクリティカルになる場合は、配置配線を再実行する前に、`phys_opt_design` を特定のネットを複製するオプションを使用しもう 1 回実行できます。次に例を示します。

```
phys_opt_design -force_replication_on_nets [get_nets [list netA netB netC]]
```

group_path コマンドを使用したクリティカル ロジックの優先度の指定

`group_path` コマンドを `-weight` オプションを指定して使用すると、クロック グループに定義されているパス終点の優先度を高くすることができます。たとえば、特定のクロックが供給されるロジックのグループの優先度を高くするには、次のコマンドを使用します。

```
group_path -name [get_clocks clock] -weight 2
```

この例では、重み 2 が指定されているクロック グループ `clock` に属するパスが、デザインのほかのパスよりも優先されます。

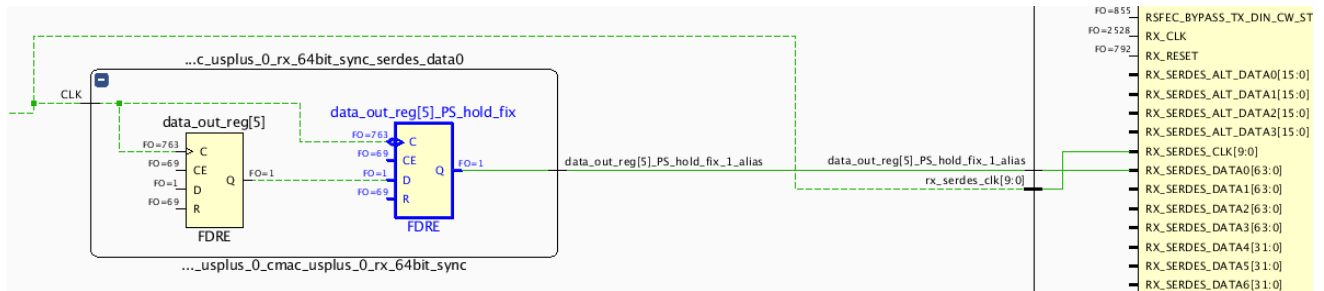
配線の前に大きなホールド違反を修正

ホールド違反が大きい (> 0.4 ns) パスでは、デザインを配線する前にホールド違反を削減しておく、配線ツールで配線迂回を使用して残りの小さなホールド違反を修正しやすくなるので、有益です。ホールド違反の修正が配線密集の原因である場合は、配線の前にホールド違反を削減すると有益であることがあります。phys_opt_design のホールド違反の修正オプションは、それぞれ異なるリソースを使用し、特定のターゲットがあります。デバイスの使用率および必要な効果によって、適切なオプションを使用することが重要です。ホールド違反の修正のために phys_opt_design を実行する前に、デザインのクロックツリーがスキューを最小限に抑えるために適切に制約され手入ることを確認してください。

順次エレメントの間に立ち下がりエッジでトリガーされるレジスタを挿入すると、タイミングパスが 2 つの $1/2$ 周期のパスに分割され、ホールド違反を大幅に削減できます。立ち下がりエッジでトリガーされるレジスタを挿入するには、-insert_negative_edge_ffs インプリメンテーション段階で phys_opt_design オプションを使用します。この最適化は、フリップフロップドライバーを含み、順次エレメント間の LUT 数が 1 つまでのパスのみに適用されます。パスの最適化後のセットアップ スラックが十分に大きな正の値になる必要があり、そうならない場合は最適化は破棄されます。

次の図に、CMAC ブロックを駆動するフリップフロップの後に挿入された立ち下がりエッジでトリガーされるレジスタを示します。最適化の前は、フリップフロップとドライバーの間のホールド スラックは -0.492 ns でした。立ち下がりエッジでトリガーされるレジスタ (青でハイライト) を挿入すると、セットアップ スラックおよびホールド スラックの両方が正になります。

図 140: 立ち下がりエッジでトリガーされるレジスタを挿入してホールド違反を修正



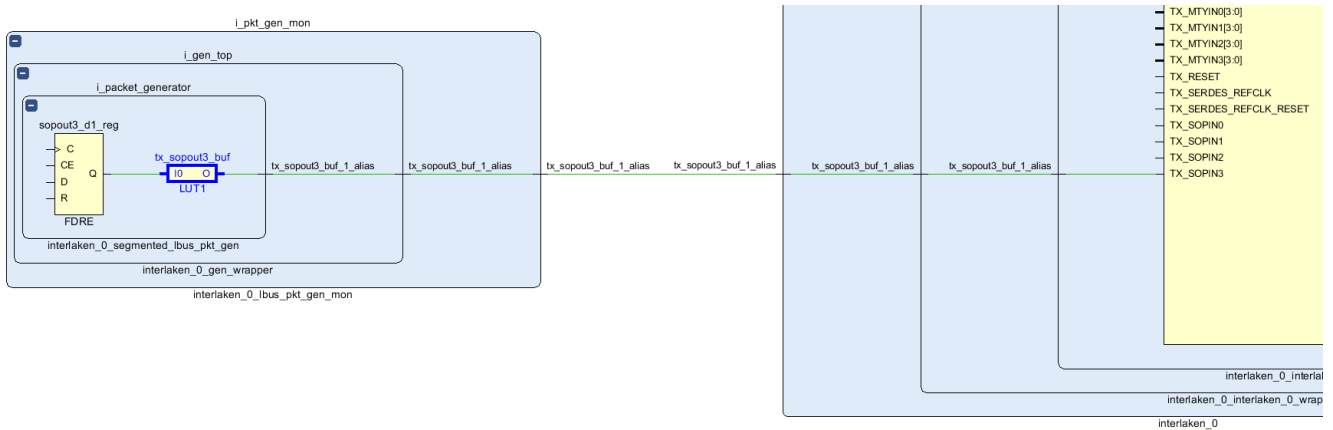
データパスに LUT1 遅延を挿入してホールド違反を削減することもできます。LUT1 遅延を挿入するには、phys_opt_design インプリメンテーション段階で次のオプションを使用します。

- hold_fix: LUT1 の挿入を実行し、十分な正のセットアップ スラックがある最大の WHS 違反のパスのみを考慮します。
- aggressive_hold_fix: LUT1 の挿入を -hold_fix オプションよりも積極的に実行します。-aggressive_hold_fix オプションは多くのホールド違反パスに LUT1 を挿入することを考慮し、デザインの THS は大幅に削減しますが、LUT の使用率は増加します。

注記: phys_opt_design -directive ExploreWithAggressiveHoldFix 指示子は、Explore 指示子と -aggressive_hold_fix オプションを 1 つの最適化で実行します。

次の図では、ILKN ブロックを駆動するフリップフロップの後に LUT1 遅延が挿入されています。最適化の前は、フリップフロップから ILKN のパスがデザインの WHS パスで、ホールド スラックは -0.277 ns でした。LUT1 遅延 (青色) を挿入すると、ホールド スラックが正になり、セットアップ スラックは正のままになります。

図 141: LUT1 遅延を挿入してホールド違反を修正



密集の解消

密集はさまざまな要因で発生する複雑な問題であり、簡潔なソリューションが見つかるとは限りません。
report_design_analysis の密集レポートを使用すると、密集領域と密集ウィンドウ内に含まれるモジュールを特定できます。密集領域に配置されているモジュールを最適化するには、さまざまな手法があります。
report_qor_suggestions を使用すると、密集の原因となる多くの項目を自動的に解決できます。



ヒント: 次に説明する手法を使用して密集の解消を試みる前に、制約が適切なものであり、ザイリンクスが推奨するクロッキング ガイドラインに従っていることを確認してください。過剰なホールド タイム エラー (または負のホールド スラック) およびクロックのばらつきがあると、配線を迂回させることが必要となり、密集が発生することがあります。複数の Pblock が重なっている部分があると、密集の原因となる可能性があるため、Pblock が重ならないようにします。

デバイス使用率の低減

複数のファブリック リソースの使用率が高い (平均で > 75%) 場合、ネットリストの複雑性も高い (最上位の接続性が高い、Rent 指数が大きい、平均ファンアウトが大きい) と配置が困難になります。高パフォーマンス デザインでは、配置に関する追加の課題もあります。その場合、デザインの機能を見直し、使用率の高いファブリック リソースが 1 つか 2 つになるまで必須ではないモジュールを削除することを検討します。ロジックを削減できない場合は、この章に示すほかの密集緩和手法を試します。



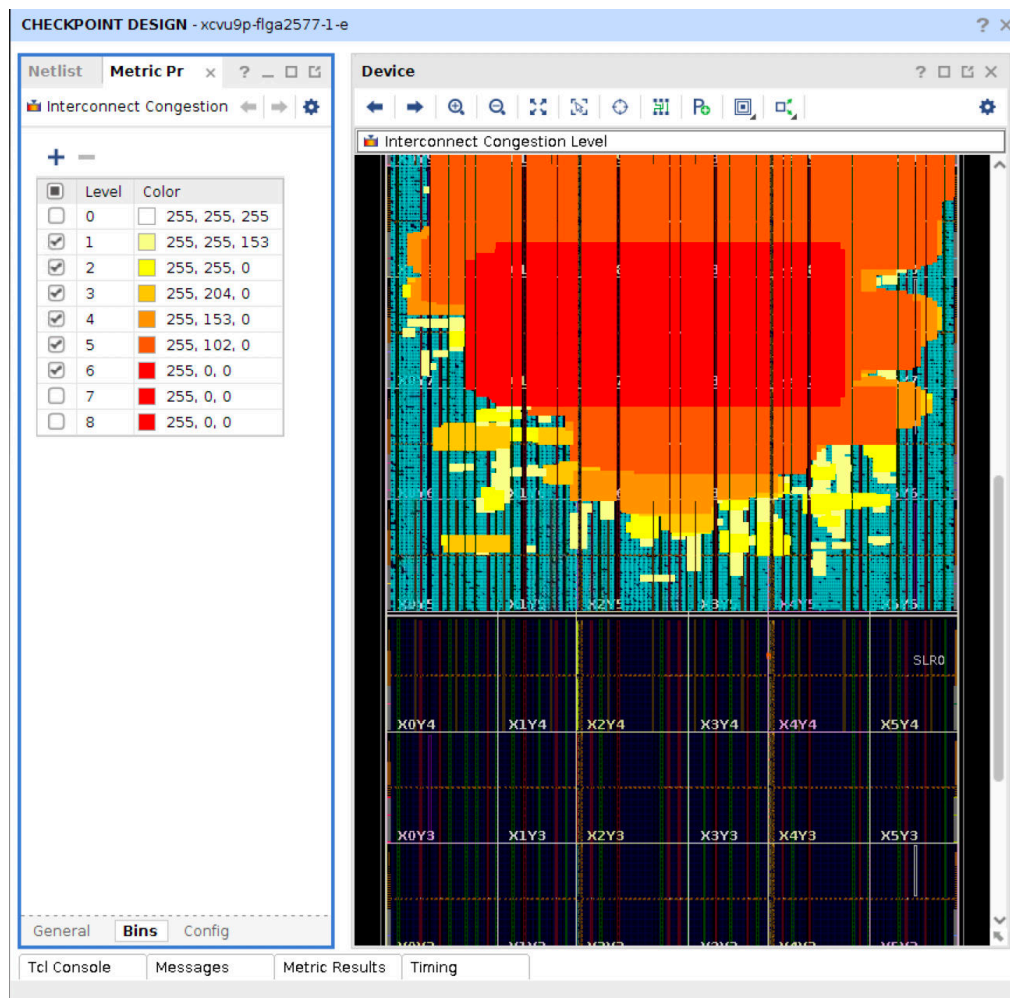
ヒント: 未使用のロジックを削除したら、正確な値を得るため、合成後ではなく opt_design 後のリソース使用率を確認してください。

SSI デバイスでの SLR 使用率のバランス

SSI テクノロジ デバイスをターゲットとする場合、SLR 領域ごとの使用率を解析することが重要です。全体的な使用率が低い場合でも、1 つの SLR で使用率が高いと、密集が発生する可能性があります。

次の図では、全体的な使用率が低いですが、SLR2 の使用率が高く、SLR2 のロジックにはほかの SLR のロジックよりも多くの配線リソースが必要です。このエリアのロジックは幅の広いバス マルチプレクサーであり、配線リソースがフルに使用されます。

図 142: SLR 領域ごとの使用率の解析



使用率のバランスを取るには、次のいずれかを試してみてください。

- 配置の -directive オプションの異なる指示子を使用して、デザインを分散します。
- Pblock などのフロアプラン制約を使用して、一部のモジュールが使用率が高く密集した SLR に配置されないようにします。

配置および配置の異なる指示子を使用

配置は通常デザインの全体的なデザイン パフォーマンスに最も影響するので、密集を緩和するには、-directive オプションの異なる指示子を試すことが最初に試す手法の 1 つです。Pblock 制約なしで -directive オプションの別の指示子を実行し、配置で必要に応じてロジックを分散する柔軟性が増すようにすることを考慮してください。

ロジックをデバイス全体に分散して密集領域を回避することにより、密集を緩和する -directive オプションの指示子がいくつかあります。ロジックを分散する配置の -directive オプションの指示子は、次のとおりです。

- AltSpreadLogic_high
- AltSpreadLogic_medium
- AltSpreadLogic_low

- SSI_SpreadLogic_high
- SSI_SpreadLogic_low

SLR 間をまたぐデザインで密集が検出された場合は、次を考慮します。

- SSI_BalanceSLLs 指示子を使用すると、SLR 間で SLL のバランスが取られるようにデザインが SLR 間に分割されます。
- SSI_SpreadSLLs 指示子を使用すると、デザインを SLR 間に分割する際に、接続の多い領域に追加のエリアが割り当てられます。

配置の -directive のほかの指示子またはインプリメンテーション ストラテジも密集の緩和に役立つことがあるので、上記の指示子の後に試してみてください。

配置の -directive オプションの異なるモード指示子における密集度を比較するには、place_design の実行後にデザイン解析レポートの密集レポートを生成するか、配線ログ ファイルでの密集の初期見積もりを確認します。

配置の -directive オプションの方が配線よりも密集への影響は大きくなりますが、配線の -directive オプションの異なる指示子を試すと有益な場合もあります。次の指示子を使用すると、配線でより多くの配線にアクセスするよう努力が費やされ、インターコネクト タイルの密集が緩和されます。

- AlternateCLBRouting

注記: Short 密集がある場合、または Short と Long の両方の密集がある場合に最も有益です。この指示子は、UltraScale デバイスにのみ使用可能です。

詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 (UG904) のこのセクションを参照してください。

関連情報

密集レベルの範囲

境界をまたぐ最適化をオフにする

合成で境界をまたぐ最適化をオフにすると、追加のロジックがモジュール内の配置されるのを防ぐことができます。これによりモジュールの複雑性は削減しますが、全体的な使用率が高くなる場合があります。これをグローバルに実行するには、-flatten_hierarchy none の synth_design オプションを使用します。この手法は、RTL で KEEP_HIERARCHY 属性が設定されたモジュールにも適用されます。

MUXF のマップの削減



ヒント: この最適化手法は、report_qor_suggestions Tcl コマンドにより自動的に適用されます。

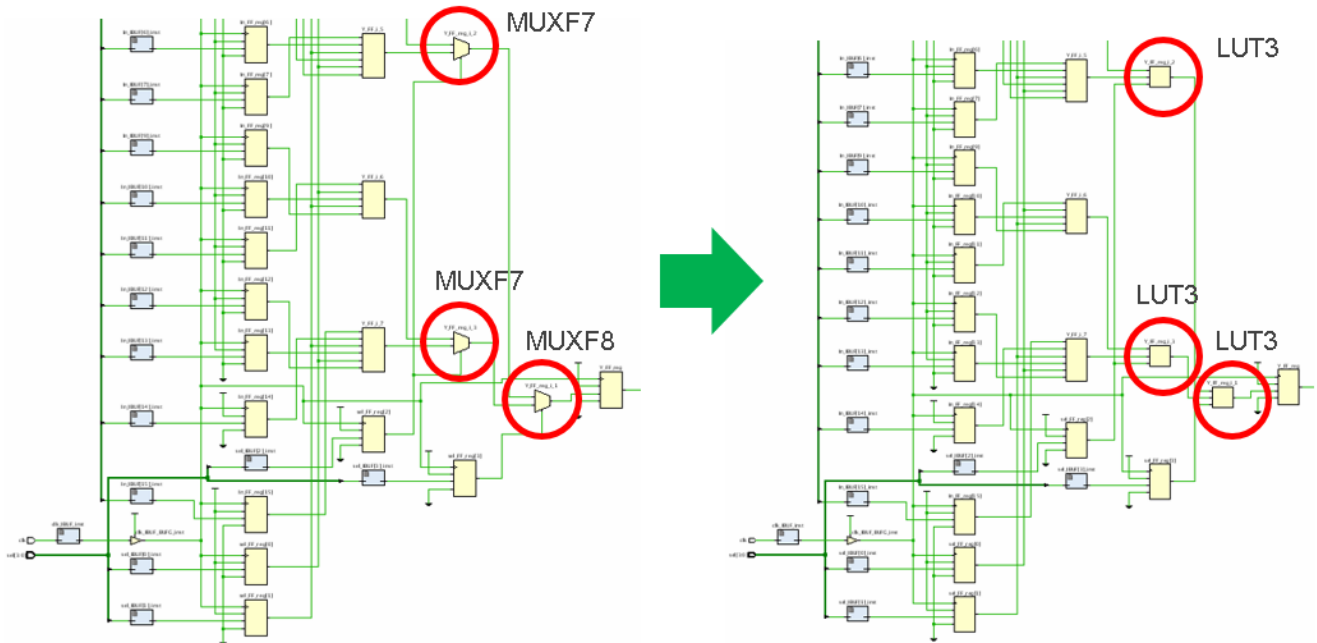
MUXF* プリミティブを使用すると、ロジック段数が多いクリティカル パスやクロック要件の厳しいクリティカル パスで有益であり、消費電力も削減されます。MUXF* は CLB 内に配置された専用のマルチプレクサー リソースで、MUXF7、MUXF8、MUXF9 があります。これらのリソースは、配置中に 8 個までの LUT にグループ化されます。このグループ化により CLB 入力の使用率および配線要求が高くなり、ネットリストの接続性が複雑な場合に配置の柔軟性が制限されることがあり、配線の密集度が高くなってタイミングが悪化することがあります。

さらに、opt_design コマンドでは、配線性を向上するために MUXF* 構造を LUT3 プリミティブにリマップするオプションのマルチプレクサー最適化フェーズを実行できます。-muxf_remap オプションを使用すると、すべての MUXF* セルをリマップできます。または、密集した領域の一部のセルで MUXF_REMAP プロパティを TRUE に設定し、マルチプレクサーのリマップの適用範囲を制限します。MUXF_REMAP プロパティが TRUE に設定されたすべての MUXF* セルに対して、opt_design の実行中に MUX 最適化が実行され、LUT3 にリマップされます。

注記: これらのリソースをディスエーブルにすると、消費電力が増加することがあります。この方法は、タイミング クロージャを達成するために必要な場合にのみ使用してください。

次の図に、MUXF* 最適化の前後の 16:1 マルチプレクサーを示します。

図 143: MUX 最適化前後のネットリスト



MUX 最適化を実行した後にネットリストをさらに最適化するには、`-remap` オプションを `-muxf_remap` オプションと共に使用します。これにより、可能な場合に、MUXF* 最適化で生成された LUT3 プリミティブが接続されたロジックと結合されます。

配置または配線後にログ ファイルまたはデザイン解析レポート (`report_design_analysis -congestion`) の [Router Initial Estimated Congestion] 表を確認することにより、配線の密集がタイミング クロージャに影響しているかどうかを判断できます。

次の図のデザイン解析/レポートは、デバイスの 7% が South 方向の Short 密集レベル 5 (32x32 CLB) による影響を受けており、対応する密集エリアで 26% の MUXF が使用されていることを示しています。

図 144: `report_design_analysis` の密集表の例

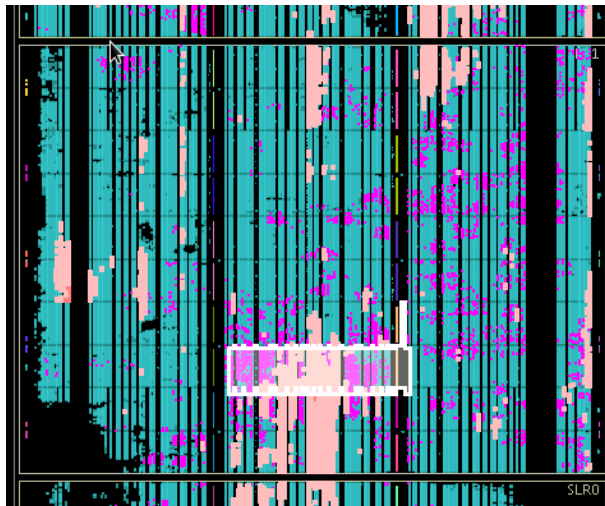
Direction	Type	Congestion Level	Percentage Tiles	Congestion Window	Cell Names	Combined LUTs	LUT6	...	MUXF
North	Short	4	6.983%	(CLEL_L_X48Y73,CLEL_R_X63Y88)	inst_name1 (92%)	0%	22%	...	4%
South	Short	5	7.136%	(CLEL_L_X32Y297,CLEL_R_X63Y328)	inst_name2 (99%)	2%	49%	...	26%
East	Short	4	6.005%	(CLEL_L_X48Y109,CLE_M_X63Y125)	inst_name3 (94%)	0%	38%	...	10%
West	Short	4	6.541%	(CLEL_L_X32Y273,CLE_M_X47Y320)	inst_name4 (92%)	1%	48%	...	23%

Vivado IDE では、デザイン解析レポートの表で行を選択すると、対応する密集エリアが [Device] ウィンドウでハイライトされます。次の図では、MUXF の使用率が高いエリアで密集がオーバーラップしています。Vivado IDE の [Tcl Console] ウィンドウで次のコマンドを使用して MUXF セルをマゼンタ色にハイライトしています。

```
highlight_objects -color magenta [get_cells -hier -filter REF_NAME=~MUXF*]
```

MUXF* は CLB 内に配置された専用のマルチプレクサー リソースで、MUXF7、MUXF8、MUXF9 が含まれます。これらのリソースは配置中に 8 個までの LUT とグループ化され、CLB 入力の使用率および配線要求が高くなり、配置の柔軟性が低下します。CLB ごとの密集の見積もりは、Vivado IDE メトリクスを使用して表示されます。

図 145: Vivado IDE の [Device] ウィンドウでハイライトされた MUXF の密集



MUXF* の使用率が高いエリアと密集度が高いエリアがオーバーラップする場合は、ザイリンクスでは MUXF* に対応する機能を LUT にマップして MUXF* の数を削減することをお勧めします。LUT はより柔軟に配置配線できます。XDC 合成制約で次のコマンドを使用して、ネットリストを変更します。

```
set_property BLOCK_SYNTH.MUXF_MAPPING 0 [get_cells inst_name4]
```

合成、配置、および配線を再実行すると、デザイン解析レポートの密集の表がアップデートされ、South 方向の Short 密集のレベルが下がり (レベル 4)、これにより通常タイミング結果が向上します。

図 146: モジュールの MUXF の使用を削減した後の [Initial Router Congestion] の表

Direction	Type	Congestion Level	Percentage Tiles	Congestion Window	Cell Names	Combined LUTs	LUT6	MUXF
North	Short	4	6.983%	(CLEL_L_X48Y73,CLEL_R_X63Y88)	inst_name1(92%)	0%	22%	4%
South	Short	4	9.040%	(CLEL_L_X34Y297,CLEL_R_X49Y312)	inst_name2(99%)	2%	54%	4%
East	Short	4	6.005%	(CLEL_L_X48Y109,CLE_M_X63Y125)	inst_name3(94%)	0%	38%	10%
West	Short	4	6.541%	(CLEL_L_X32Y273,CLE_M_X47Y320)	inst_name4(92%)	1%	48%	23%

LUT の組み合わせをディスエーブル



ヒント: この最適化手法は、report_qor_suggestions Tcl コマンドにより自動的に適用されます。

LUT の組み合わせでは、入力を共有する LUT のペアを O5 と O6 の両方の出力を使用する 1 つの LUT に統合することにより、ロジックの使用率を削減します。ただし、LUT の組み合わせはスライスの入力/出力の接続を増加する傾向があるので、密集度が増加する可能性があります。密集エリアで LUT の組み合わせ率が高い場合は (> 40%)、密集を緩和するため、LUT の組み合わせを削除する合成ストラテジを使用してみてください。Flow_AlternateRoutability 合成ストラテジおよび -directive オプションの AlternateRoutability 指示子を使用すると、合成で追加の LUT の組み合わせは生成されなくなります。

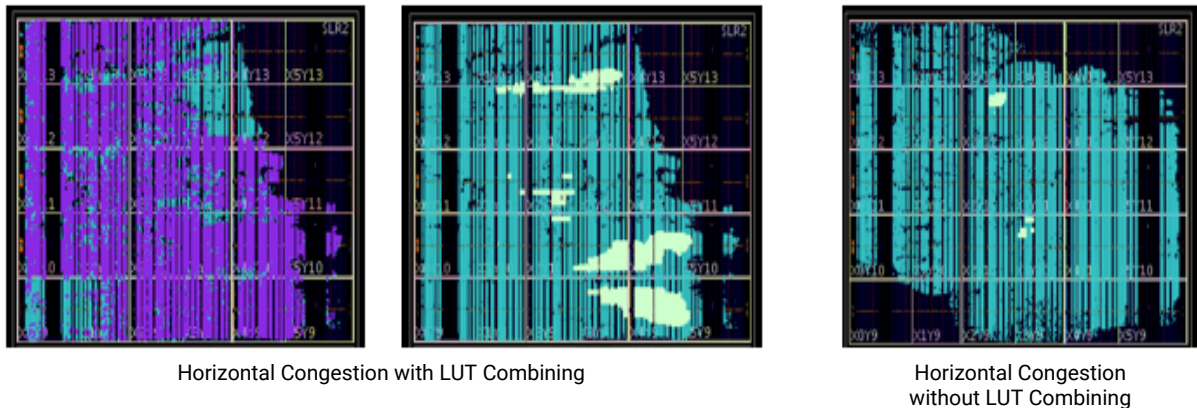
注記: 合成に Synplify Pro を使用している場合は、[Device] タブの [Implementation Options] にある [Enable Advanced LUT Combining] オプションを使用できます。このオプションはデフォルトでオンになっています。Synplify Pro プロジェクト ファイル (*.prj) を変更する場合は、`set_option -enable_prepacking 1` を指定します。

次のコマンドを使用すると、デザインで LUT の組み合わせをイネーブルにするセルを選択できます。

```
select_objects [get_cells -hier -filter {SOFT_HLUTNM != "" || HLUTNM != ""}]
```

次の図に、LUT の組み合わせがある場合とない場合におけるデザインの水平方向の密集を示します。LUT の組み合わせを使用するセルは、紫色でハイライトされています。

図 147: LUT の組み合わせが水平方向の密集に与える影響



X18040-120519

密集度の高いエリアとオーバーラップするモジュールで LUT の組み合わせをディスエーブルにするには、次の Tcl コマンドを使用します。

```
reset_property SOFT_HLUTNM [get_cells -hierarchical -filter {NAME =~ <module name> && SOFT_HLUTNM != ""}]
```

密集エリアのファンアウトの大きいネットを制限



ヒント: この最適化手法は、`report_qor_suggestions` Tcl コマンドにより自動的に適用されます。

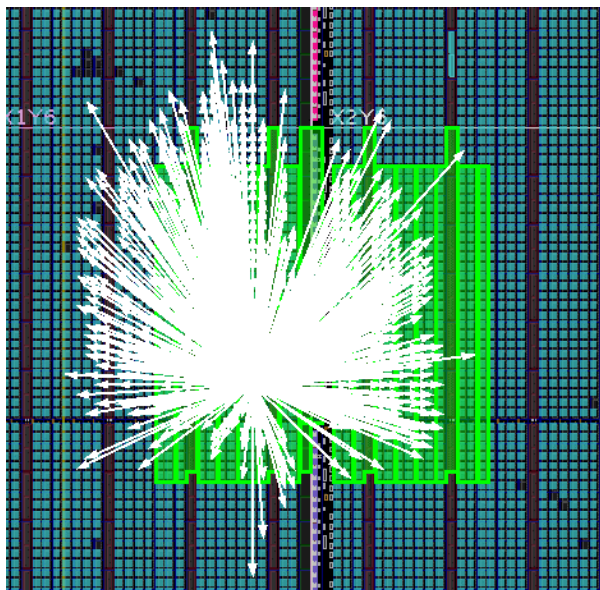
タイミング制約が厳しいファンアウトの大きいネットは、タイミングを満たすため、厳密にクラスター化された配置が必要です。これにより、次の図に示すように局所的な密集が発生する可能性があります。ファンアウトの大きいネットは、配線リソースを消費し、それらの配線リソースを密集ウィンドウのほかのネットで使用できなくすることによって密集に影響します。

ファンアウトの大きいグローバルでないネットの密集ウィンドウにおける配線性への影響を解析するには、次を実行できます。

- 密集ウィンドウで最上位モジュールの最下位セルを選択します。
- [Edit]→[Find] をクリックし、選択したセル オブジェクトのすべてのネットを選択します ([GLOBAL_CLOCK]、[Power]、[GROUND] ネットを検索)。
- ネットを [Flat Pin Count] の降順に並べ替えます。
- ファンアウトの大きいものを上位から選択し、密集ウィンドウとの関係を表示します。

これにより、密集に影響しているファンアウトの大きいネットをすばやく特定できます。

図 148: 密集ウィンドウ内のファンアウトの大きいネット



密集ウィンドウ内にあるタイミング制約の厳しいファンアウトの大きいネットに対しては、ドライバーを複製すると、配置制約が緩和されて密集が削減します。

タイミング スラックが正のファンアウトの大きいネット (ファンアウト > 5000) は、ファブリック リソースの代わりにグローバル クロック リソースに配線できます。配置の最後の方でグローバル配線リソースが使用可能である場合、ファンアウトが 1000 を超えるネットが配置段階で自動的に配線されます。この最適化は、タイミングが悪化しない場合にのみ実行されます。

また、ネットに `CLOCK_BUFFER_TYPE=BUFG` プロパティを設定して、配置の前に合成またはロジック最適化でバッファが自動的に挿入されるようにすることもできます。`place_design` の後に挿入されたバッファの配置とそのドライバーおよびロードの配置を調べ、最適なものであるかを確認します。配置が最適でない場合は、クロック バッファに UltraScale デバイスでは `CLOCK_REGION` 制約、7 シリーズ デバイスでは `LOC` 制約を設定して配置を制御します。

セル膨張の使用

セル膨張を使用すると、`place_design` 段階で空白を挿入 (セル スペースを増加) できます。これによりダイの指定のエリアにおけるセル集積度が下がり、密集が緩和されて使用可能な配線が増加します。この手法は、比較的高パフォーマンスのロジックの小さな密集したエリアに使用すると特に効果的です。

セル膨張を使用するには、階層セルに `CELL_BLOAT_FACTOR` プロパティを適用し、値を `LOW`、`MEDIUM`、または `HIGH` に設定します。数百個のセルを含む小型のモジュールでは、`HIGH` に設定することをお勧めします。



注意: デバイスで使用されている配線リソースが既に多すぎる場合は、セル膨張の使用はお勧めしません。また、セル膨張を大型のセルに使用すると、セルの配置が離れすぎてしまう可能性があります。

コンパイル フローの調整

デフォルトのコンパイル フローでは、すばやくベースライン制約を取得し、タイミングが満たされているかどうかを解析し始めることができます。初期インプリメンテーションの後、タイミング クロージャを達成するため、コンパイル フローを調整する必要がある場合があります。

ストラテジおよび指示子の使用

ストラテジおよび指示子を使用して、デザインの最適なソリューションを見つけることができます。ストラテジは、プロジェクトのインプリメンテーション run にグローバルに適用されます。指示子は、プロジェクト モードおよび非プロジェクト モードの両方で、インプリメンテーション フローの各段階で個別に設定できます。

ML ストラテジ

機械学習 (ML) ストラテジを使用すると、デザイン用の最適化されたストラテジをすばやく取得できます。配線済みデザインのストラテジ推奨事項オブジェクトを生成するには、`report_qor_suggestions` コマンドを実行します。ML ストラテジを使用するには、次のようにインプリメンテーション フローを実行する必要があります。

- プロジェクト モードでは、Default または PerformanceExplore ストラテジを使用します。
- 非プロジェクト モードでは、`opt_design` および `phys_opt_design` をイネーブルにし、すべての指示子を Default または Explore に設定します。Default と Explore を混合することはできません。

ストラテジ オブジェクトをアクティブにするには、`opt_design` を実行する前にストラテジ推奨事項を含む RQS ファイルを読み込み、すべてのコマンドの指示子を RQS に設定します。ストラテジ推奨事項フローの詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906) の「ストラテジ推奨事項」セクションを参照してください。

最高の結果を得るには、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』(UG906) のこのセクションに説明されているように、`report_qor_assessment` を実行してデザインで ML ストラテジを使用できるようにします。デザインで ML ストラテジを使用できるようにするには、設計手法チェックを解決し、QoR 評価レポート (RQA) のスコアを 3 以上にする必要があります。デザインがこの条件を満たしているのに ML ストラテジを使用できない場合は、`report_qor_suggestions` を使用して RQA スコアを改善してください。

定義済みストラテジ

ザイリンクスでは、大部分のデザインで効果的なソリューションとなるよう調整された定義済みストラテジを提供しています。

注記: ザイリンクスでは、SSI テクノロジ ストラテジを SSI テクノロジ デバイス以外のデバイスに適用することはお勧めしません。

カスタム ストラテジ

定義済みのストラテジでタイミングが満たされない場合は、`-directive` オプションを手動で組み合わせて試すことができます。配置は通常デザインの全体的なパフォーマンスに最も影響するので、I/O ロケーション制約のみを使用して配置制約なしで配置の `-directive` オプションのさまざまな指示子を試すと有益な場合があります。各配置の実行で WNS および TNS の両方を配置ログで確認することにより、最高のタイミング結果が得られた `-directive` オプションの 2 つまたは 3 つの指示子を、ダウストリームのインプリメンテーション フローの基準として選択できます。



ヒント: 指示子のリストとその機能の説明を表示するには、インプリメンテーション コマンドを `-help` オプションで指定して実行します (`place_design -help` など)。ストラテジの詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』(UG904) のこのセクションを参照してください。

これらの各チェックポイントに対して、`phys_opt_design` および `route_design` で `-directive` オプションのいくつかの指示子を試し、見積もりまたは最終 WNS/TNS が最適なもののみを保持できます。非プロジェクト モードでは、Tcl スクリプトでフローを明示的に指定し、最高のチェックポイントを保存する必要があります。プロジェクト モードでは、配置の `-directive` オプションの各指示子に対して個別にインプリメンテーション run を作成し、配置までを実行します。配置後は、最高の結果が得られた run でインプリメンテーションを続行します (Tcl ポスト スクリプトで判断)。

物理制約 (Pblock、DSP および RAM マクロ制約) を使用すると、配置で最適なソリューションが見つけれなくなることがあります。ザイリンクスでは、配置の `-directive` オプションを指定する場合は Pblock 制約を設定しないことをお勧めします。`-directive` オプションを使用して配置を実行する前に Pblock を削除するには、次の Tcl コマンドを使用できます。

```
delete_pblock [get_pblocks *]
```

`place_design -directive <directive>` を実行して最高の結果の配置を解析すると、デザインをフロアプランまたはブロック RAM マクロまたは DSP マクロの配置を再利用するためのテンプレートを取得することもできます。これにより、各実行で安定した結果が得られるようになります。

最適化済みイテレーションの使用

コマンドを複数回実行すると、最適な結果を得るのに有益である場合があります。たとえば、`phys_opt_design` をまず `force_replication_on_nets` オプションを使用して実行し、配線中に WNS に影響すると思われるクリティカル ネットの一部を最適化すると有益な場合があります。

その後、`phys_opt_design` を `-directive` オプションで指示子を指定して実行し、デザインの全体的な WNS を向上します。

非プロジェクト モードでは、次のコマンドを使用します。

```
phys_opt_design -force_replication_on_nets [get_nets -hier *phy_reset*]  
phys_opt_design -directive <directive name>
```

プロジェクト モードでは、`phys_opt_design` オプションを使用して実行する `phys_opt_design` の Tcl プリスク립トの一部として最初の `-directive` コマンドを実行します。

デザインの制約を厳しくする

配線後にデザインのタイミングが少しの差で満たされていない場合、これは通常配置後のタイミング マージンが小さいためです。配置および物理最適化でタイミング要件を厳しくすると、配線のタイミング マージンを増加できる可能性があります。これを達成するには、次の理由からザイリンクスでは `set_clock_uncertainty` 制約を使用することをお勧めします。

- クロックの関連性は変更されない (クロック波形は同じ)。
- ツールで算出されるクロックのばらつき (ジッター、位相エラー) に追加される。
- `-from` および `-to` オプションで指定されたクロック ドメインまたはクロック乗せ換えに特定。
- ヌル値を適用することにより、前に設定したクロックのばらつき制約を簡単にリセット可能。

どの場合でも、ザイリンクスでは次のようにすることをお勧めします。

- セットアップ タイミングが満たされていないクロックまたはクロック乗せ換えのみの制約を厳しくします。
- `-setup` オプションを使用してセットアップ要件のみを厳しくします。

注記: このオプションを指定しない場合、セットアップおよびホールド要件の両方が厳しくなります。

- 配線を実行する前に追加したばらつきをリセットします。

制約が厳しすぎる例

配線の前後で、デザインの `clk1` クロック ドメインのパスでタイミングが -0.2 ns の差で満たされておらず、`clk2` から `clk3` へのパスでタイミングが -0.3 ns の差で満たされていないとします。

1. ネットリスト デザインを読み込み、通常の制約を適用します。
2. 一部のクロックにクロックのばらつきを追加して制約を厳しくします。
 - a. 値は違反の量以上にします。
 - b. 制約はセットアップ パスのみに適用します。

```
set_clock_uncertainty -from clk0 -to clk1 0.3 -setup
set_clock_uncertainty -from clk2 -to clk3 0.4 -setup
```

3. 配線まで実行します。配線前にタイミングが満たされるのが理想的です。
4. 追加したばらつきを削除します。

```
set_clock_uncertainty -from clk0 -to clk1 0 -setup
set_clock_uncertainty -from clk2 -to clk3 0 -setup
```

5. 配線を実行します。

配線後にタイミングを解析し、制約を厳しくすることでタイミングが向上したかを確認します。配置後にタイミングが満たされており、配線後に少しの差でタイミングが満たされない場合は、ばらつきを増加してもう一度実行します。



推奨: 制約を 0.5 ns 以上厳しくしないでください。デザインの制約を厳しくしすぎると、インプリメンテーションの消費電力が増加するだけでなく、実行時間が長くなる可能性があります。



ヒント: デザインの制約を厳しくする代わりに、各パス グループの相対的な優先度を変更する方法もあります。デフォルトでは、インプリメンテーション中に、各クロックおよびユーザー定義のパス グループが同じ優先度で個別に解析されます。group_path -weight 2 -name <ClockName> オプションを使用すると、クロック ベースのパス グループに優先度を設定できます。ユーザー定義のパス グループの優先度は変更できません。

フロアプラン

フロアプランを使用すると、高レベルの階層レイアウトや詳細な配置を指定できます。これにより、QoR が向上し、予測可能な結果が得られるようになります。最悪の問題または最も一般的な問題を修正すると、最も大きく改善します。たとえば、スラックが大幅に悪いパス、またはロジック段数が多いパスがある場合、Pblock を使用してそれらのパスをデバイスの同じ領域にグループ化するなどして修正します。フロアプランは、デザイン全体に対して実行するのではなく、必要な部分のみに制限します。

I/O に接続されるロジックを I/O の近くにフロアプランすると、結果が向上する可能性があり、コンパイル反復実行において一定した結果が得られるようになります。通常は、Pblock のサイズをクロック領域内に収めるのが最適です。このようにすると、配置が最も柔軟に実行されます。複数の Pblock が重なっている部分があると、共有エリアの密度が高くなる可能性があるため、Pblock が重ならないようにします。2 つの Pblock 間にそれらを接続する信号が多数ある場合は、これらの Pblock を 1 つの Pblock にまとめることを考慮します。また、Pblock 間をまたぐネットの数を最小限に抑えます。



ヒント: Vivado Design Suite を新しいバージョンにアップグレードする際は、まず Pblock なしまたは最小限の Pblock (SLR レベルの Pblock のみなど) でコンパイルし、タイミング クロージャを達成するのに問題があるかどうかを確認してください。以前のバージョンでは QoR の向上に効果的であった Pblock が、新しいバージョンのツールでは最適なインプリメンテーションを見つけるのを妨げる可能性があります。

SSI テクノロジ デバイスでは、SLR Pblock またはソフト フロアプラン制約 (USER_SLR_ASSIGNMENT) を使用することも考慮してみてください。

関連情報

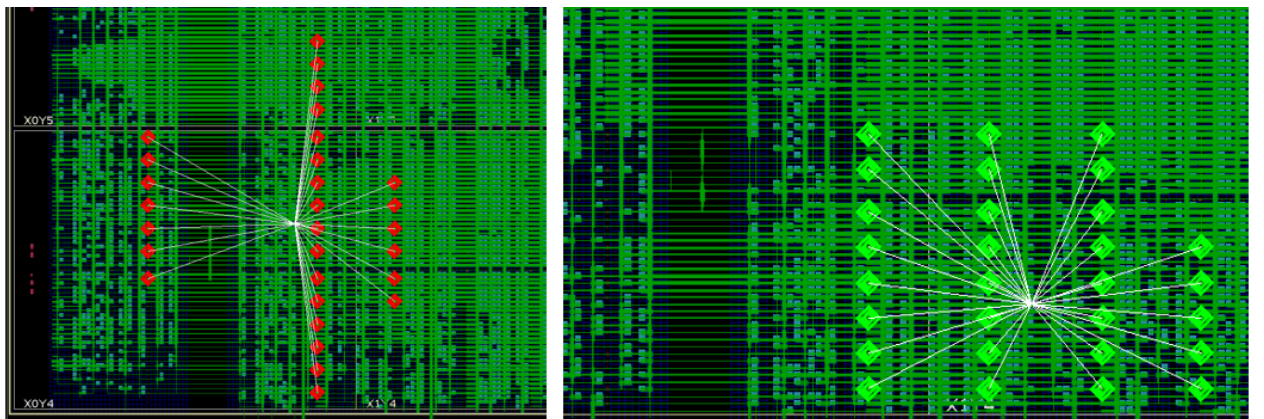
SSI テクノロジに関する考慮事項

クリティカル ロジックのグループ化

クリティカル ロジックをグループ化して SLR または I/O 列をまたぐ状況を回避すると、デザインのクリティカル パスが向上します。次の図に、大型の FIFO を 29 個の FIFO36E2 プリミティブを使用してインプリメントする 2 つの例を示します。グループの各 FIFO36E2 のクリティカル パスは WRRSTBUSY ピンから 5 つの LUT を介してグループの各 FIFO36E2 の WREN ピンに到達するパスです。

- 左側の例では、ブロック RAM の使用率が高いため、ツールでパスに最適な配置を見つけることができません。FIFO36E2 プリミティブは赤でマークされています。
- 右側の例では、FIFO36E2 ブロックがグループ化され、コンフィギュレーション列をまたいでいないので、タイミングが満たされています。FIFO36E2 プリミティブは緑色でマークされています。

図 149: Config 列を回避する場所



Locations Not Avoiding the Configuration Column

Preassigned Locations Avoiding the Configuration Column

X18041-120219

配置結果の再利用

ブロック RAM マクロおよび DSP マクロの配置を再利用するのは比較的簡単です。配置を再利用すると、ネットリストのリビジョン間での結果がより一貫したものになります。これらのプリミティブの名前は通常変化しないので、配置を保持するのは簡単です。配置の -directive オプションの指示子には、ほかの指示子よりもブロック RAM および DSP マクロの配置が良くなるものがあります。1 つの配置 run からのこの向上したマクロの配置を別の run に適用して、-directive オプションの異なる指示子を使用して実行できます。次に、UltraScale および UltraScale+ デバイス デザインでブロック RAM の配置を XDC ファイルに保存する Tcl スクリプトを示します。

```
set_property IS_LOC_FIXED 1 \
  [get_cells -hier -filter {PRIMITIVE_TYPE =~ BLOCKRAM.*.*}]
write_xdc bram_loc.xdc -exclude_timing
```

bram_loc.xdc ファイルをブロック RAM のロケーション制約のみが保持されるように変更し、これを今後の run で使用できます。



重要: 汎用スライス ロジックの配置を再利用しないでください。デザインの変更する予定の部分は、配置を再利用しないでください。デザインに小さな変更を加え、以前の配置を再利用してより予測可能な結果を取得し、コンパイル時間を短縮するには、インクリメンタル コンパイル フローを使用します。

インクリメンタル インプリメンテーションの使用

インクリメンタル インプリメンテーションを使用すると、インプリメンテーションのコンパイル時間を短縮し、予測可能な結果を生成できます。サイリンクスでは、インクリメンタル インプリメンテーションをタイミング クロージャの標準的なストラテジの一部として使用することをお勧めします。詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 (UG904) の [インクリメンタル インプリメンテーション](#) または 『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』 (UG906) の [QoR 推奨項目レポート](#) を参照してください。

このセクションでは、高再利用モードおよび低再利用モードでの自動インクリメンタル インプリメンテーションの推奨事項を示します。

質の高い基準チェックポイントを選択する

インクリメンタル インプリメンテーション フローは再利用に依存するので、最も重要な入力基準チェックポイントです。プロジェクト モードで自動インクリメンタル インプリメンテーションを使用する場合、Vivado ツール基準チェックポイントのアップデートが管理されます。これにより、再利用率が高くなり、タイミング クロージャがほぼ達成されるようになります。

インクリメンタル インプリメンテーションのその他すべてのユース ケースでは、ユーザーが基準チェックポイントを選択する必要があります。基準チェックポイント選択のガイドラインは、次のとおりです。

- タイミングを満たしているか、タイミングをほぼ満たしている基準チェックポイントを使用します。基準チェックポイントがタイミングをほぼ満たしている場合、インクリメンタル インプリメンテーション フローを実行する前に、次のようにタイミングを向上しておくことが有益な場合があります。

注記: 自動インクリメンタル インプリメンテーションでは、WNS が -0.250 ns 以上のチェックポイントは拒否されます。

- `route_design -tns_cleanup` を実行してワースト ケース パスでないパスを最適化します。
- 配線後に `phys_opt_design` コマンドを実行し、タイミングを向上します。このコマンドの実行時間は長くなる場合がありますが、インクリメンタル インプリメンテーション実行ではこれらの再激化は高速に再実行されます。
- `report_qor_suggestions` コマンドを使用して、デザインを向上するための推奨事項を取得します。インクリメンタル インプリメンテーション フローに適用する新しい推奨事項は、インクリメンタル インプリメンテーションに適したものである必要があります。基準チェックポイントに既に適用されている推奨事項は、インクリメンタル インプリメンテーションに適したものである必要はありません。推奨事項がインクリメンタル インプリメンテーションに適していない場合は、デフォルト フローを使用して推奨事項を適用し、チェックポイントをアップデートすることを考慮してください。
- 密集度の最も低いチェックポイントを選択します。密集度の低いチェックポイントの方が、密集したチェックポイントよりも変更に対応できます。
- 基準デザインとインクリメンタル チェックポイントの一致を最大限にします。

注記: 自動インクリメンタル インプリメンテーションでは、セル一致が 94% 未満、ネット一致が 90% 未満のチェックポイントは拒否されます。

- インクリメンタル合成を使用して、RTL の変更によるネットリストの変更を削減します。インクリメンタル インプリメンテーションを使用する準備ができるまで待つのではなく、デザイン クロージャ サイクルの早期にインクリメンタル合成をイネーブルにします。
- 基準チェックポイント `run` とインクリメンタル インプリメンテーション `run` で、`synth_design` および `opt_design` のオプションを同じにします。
- 同じツール バージョンを使用します。これは必須要件ではありませんが、しきい値の変更および新しい最適化が追加されているので、一致が下がる可能性があります。

- `opt_design` `AddRemap` および `ExploreWithRemap` 指示子は、これらがタイミング クロージャを達成するための唯一の指示子である場合以外は使用しないようにします。これらの指示子を使用すると、コードベースを変更した場合に名前が一致しなくなる可能性があります。
- `report_qor_assessment` を使用して、デザインがインクリメンタル インプリメンテーション フローに適したものであるか、デフォルト フローから切り替えるのが望ましいかどうかを判断します。



ヒント: インクリメンタル インプリメンテーションしきい値の調整に関する情報は、`config_implementation -help` を実行します。基準デザインとインクリメンタル チェックポイントの違いをレポートするには、`report_incremental_reuse` を使用します。

高再利用モードではインクリメンタル インプリメンテーションの指示子を選択する

インクリメンタル インプリメンテーション フローの動作は、`-directive` オプションの指示子を使用して制御できます。インプリメンテーション `run` にインクリメンタル インプリメンテーション アルゴリズムを使用すると、ツールはこれらの指示子に従います。フローをデフォルトのアルゴリズムに戻すと、ツールは `place_design`、`phys_opt_design`、および `route_design` コマンドで指定された指示子に従います。

インクリメンタル インプリメンテーション フローで使用可能な指示子は、次のとおりです。

- **RuntimeOptimized:** 基準チェックポイントの WNS をターゲットとします。これにより、基準チェックポイントと一貫したものになり、配置配線の実行速度が最低でも 2 倍になります。基準チェックポイントでタイミング クロージャが達成されていない場合は、タイミング クロージャの達成は試みられません。これがデフォルトの指示子です。
- **TimingClosure:** WNS = 0.000 ns をターゲットとします。基準 `run` がタイミングをほぼ満たしており、結果と実行時間の一貫性よりも、タイミングを満たすことが優先される場合に使用します。このモードを使用すると、困難なデザインで WNS を最大 250 ps 向上できます。この指示子を QoR 推奨項目と共に使用すると、タイミング クロージャを達成できる可能性が最大になります。この指示子を使用すると、実行時間は長くなります。
- **Quick:** 99% 以上再利用され、タイミングを簡単に満たすことのできるデザインで使用するためのオプションです。このオプションは通常、タイミングに影響しない小さな変更を含む ASIC エミュレーションおよびプロトタイプデザインで使用します。

次に、プロジェクト モードのコマンドの例を示します。

```
set_property -name INCREMENTAL_CHECKPOINT.MORE_OPTIONS -value {-directive
TimingClosure} -object [get_runs <runName>]
```

次に、非プロジェクト モードのコマンドの例を示します。

```
read_checkpoint -incremental -directive TimingClosure <reference>.dcp
```

注記: `RuntimeOptimized` 指示子は、以前の Vivado Design Suite リリースの `Default` マップ指示子に置き換わり、`TimingClosure` 指示子は `Explore` マップ指示子に置き換わります。

低再利用モードで QoR の変動を低減する

低再利用モードでは、特定のセル (デザインの階層セルなど) またはセル タイプ (DSP、ブロック RAM など) を再利用できます。これは、次の両方の条件が満たされている場合に効果的です。

- 一部のデザイン `run` ではタイミングが満たされるが、多くの `run` では満たされない。
- デザイン フローの早期であるか、または大きな変更がまだ加えられている。

特定のセルの配置が WNS に大きく影響する場合、階層セルを再利用すると効果的です。DSP、ブロック RAM、またはその両方を再利用すると、これらのブロックの使用率が高いデザインで有益です。

特定のセルまたはセル タイプを再利用するには、次を実行します。

- タイミングが満たされていないチェックポイントなど基準 run を解析して、結果が良い run と悪い run の違いを特定します。
 - WNS が良く、密集度が低い run を特定します。
 - フロアプランを使用して SLR の配置を定義します。
- ターゲット エリアを特定したら、低再利用モードを使用した run とデフォルト フローを使用したベースライン run を比較して効果を評価します。
 - 異なる place_design 指示子を使用して、それらの結果を比較します。

注記: 低再利用モードでは、インクリメンタル インプリメンテーション指示子は無視され、ターゲット WNS は常に 0.000 ns です。

ブロック メモリの配置のみを再利用するには、次の Tcl スクリプトを使用します。

```
read_checkpoint -incremental routed.dcp \
-reuse_objects [all_rams] -fix_objects [all_rams]
```

DSP の配置のみを再利用するには、次の Tcl スクリプトを使用します。

```
read_checkpoint -incremental routed.dcp \
-reuse_objects [all_dsps] -fix_objects [all_dsps]
```

ブロック メモリと DSP の両方の配置を再利用するには、次の Tcl スクリプトを使用します。

```
read_checkpoint -incremental routed.dcp \
-reuse_objects [all_rams] -reuse_objects [all_dsps] -fix_objects
[current_design]
```

特定の階層セルの階層とその下の階層すべてを再利用するには、次の Tcl スクリプトを使用します。

```
read_checkpoint -incremental routed.dcp \
-only_reuse [get_cells <cell_name>] -fix_objects [get_cells <cell_name>]
```



推奨: ザイリンクスでは、階層モジュールを再利用する際は、PRESERVE_BOUNDARY 制約を設定したアウトオブ コンテキスト合成またはインクリメンタル合成を使用して、セル一致が 100% になるようにすることをお勧めします。

フロアプランおよび過剰な制約の回避

インクリメンタル インプリメンテーション フローを使用する場合は、次を回避してください。

- インクリメンタル インプリメンテーション run にはフロアプランを使用しないようにします。

Pblock の配置は基準チェックポイントの配置に置き換えられます。
- 配置で制約を厳しくしすぎないようにします。

インクリメンタル インプリメンテーション run でデザインの制約を厳しくしすぎると、ツールが不適切に変更されたターゲット WNS を満たそうとするため、再利用に大きく影響します。

関連情報

[デザインの制約を厳しくする](#)

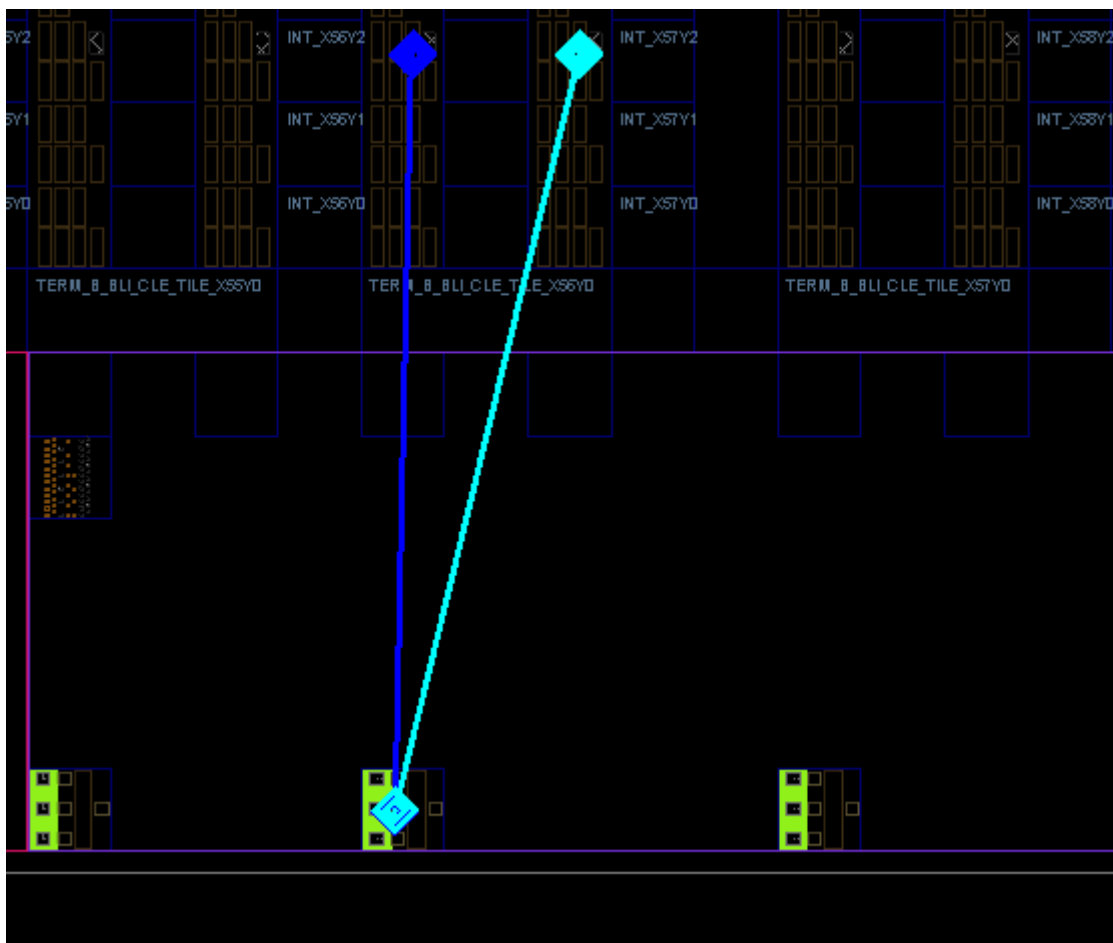
XPIO と PL のインターフェイスのタイミング手法

ハードウェアの XPIO プログラマブル ロジック (PL) インターフェイスの間に境界ロジック インターフェイス フリップフロップがあり、これをタイミングを向上するために使用できます。XPHY ロジック、I/O ロジック、およびクロック調整ブロックなどの XPIO の専用ブロックには、境界ロジック インターフェイス フリップフロップがあります。デザインのフリップフロップに境界ロジックインターフェイス (BLI) 制約を適用すると、デザインの配置時にこのハードウェア機能を自動的に活用できます。この例では、XPIO の I/O ロジックセル ODDRE1 および IDORE1 との間のデータパスに BLI FFS を利用しています。

```
set_property BLI TRUE [get_cells {oddr_D1-BLI-reg oddr_D2-BLI-reg}]
set_property BLI TRUE [get_cells {iddr_Q1-BLI-reg iddr_Q2-BLI-reg}]
```

次の図に、BLI プロパティを TRUE に設定した場合の配置と接続の結果を示します。

図 150: ODDRE1 および IDORE1 に対する XPIO と PL のインターフェイス BLI フリップフロップの配置



SSI テクノロジに関する考慮事項

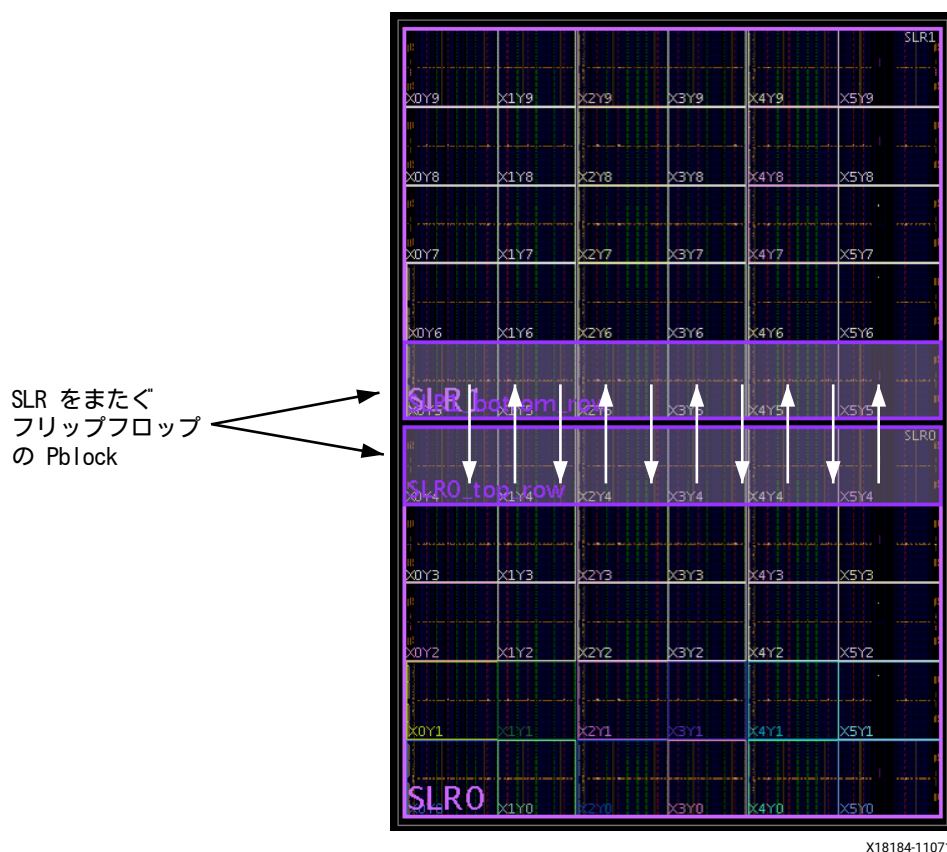
スタックド シリコン インターコネクト (SSI) テクノロジ デバイスは複数の SLR (Super Logic Region) で構成されており、インターポーザーにより結合されています。インターポーザーの接続は、SLL (Super Long Line) と呼ばれます。1 つの SLR から別の SLR への移動には多少の遅延があります。デザインでの SLL 遅延の影響を最小限に抑えるには、SLR 間をまたぐパスがクリティカル パスの一部とならないようにします。フロアプランで要件を満たすのが困難なモジュールを 1 つの SLR 内に配置し、SLR 間をまたぐパスを最小限に抑えると、SSI テクノロジ デバイスをターゲットとするデザインのタイミングおよび配線性を向上できます。

ハード SLR フロアプラン制約の使用

高パフォーマンス デザインでは、グローバル配置および SLR 分割を容易にするため、主な階層間に十分なパイプラインが必要です。困難なデザインでは、SLR 間をまたぐ点が run によって変わることがあります。SLR Pblock の定義に加え、クロック領域に揃い、SLR 境界沿いに配置された追加の Pblock を作成することにより、SLR 間をまたぐ部分にあるフリップフロップを制約できます。次の図に、次の Pblock を含む UltraScale ku115 SSI デバイスを示します。

- 2 つの SLR Pblock: SLR0 および SLR1
- 2 つの SLR 間をまたぐ Pblock: SLR0_top_row および SLR1_bottom_row

図 151: SLR 間をまたぐ Pblock の例



重要: ザイリンクスでは、SLR 間をまたぐ Pblock には、LAGUNA ではなく CLOCKREGION 範囲を使用することをお勧めします。



ヒント: SLR Pblock は、完全な SLR を指定することにより定義できます。たとえば、`resize_pblock pblock_SLR0 -add SLR0` のように指定します。

詳細は、『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』 (UG906) のこのセクションを参照してください。



ビデオ: フロアプランを使用してデザイン パフォーマンスの問題に対処する方法は、[Vivado Design Suite QuickTake ビデオ: Vivado でのデザイン解析およびフロアプラン](#)を参照してください。

ソフト SLR フロアプラン制約の使用

大型デザインでは、ほとんどの主要なブロックのロジックは 1 つの SLR に収まるので、デザインを数回イテレーションするだけでタイミングを満たすことができます。ただし、特に主要なブロック間および SLR 間の接続などのロジックの小さな部分は、全体的なデザイン配置によって QoR が変化します。そのような場合、配置の問題に対処してタイミングを満たすためにロジックの一部を別の SLR に複製したり移動したりするのに、配置および物理最適化アルゴリズムに追加の柔軟性が必要です。

インプリメンテーション ツールでより柔軟な処理が実行できるようにするため、大型のデザイン ブロックを SLR に割り当ててフロアプランするために `USER_SLR_ASSIGNMENT` プロパティを使用できます。このプロパティは文字列値に設定し、階層セルに適用します。最下位セルでは無視されます。このプロパティに設定する値は、ロジックの分割に次のように影響します。

- SLR 名: 階層セルに SLR の名前 (SLR0、SLR1、SLR2 など) を割り当てると、そのセル全体が指定の SLR に配置されます。
- 文字列値: 階層セルに任意の文字列値を割り当てると、SLR は配置により選択されます。この設定により、セルが複数の SLR に分割されるのを防ぐことができます。

注記: 複数のセルに同じ `USER_SLR_ASSIGNMENT` 値を設定すると、それらのセルが同じ SLR にグループ化されます。

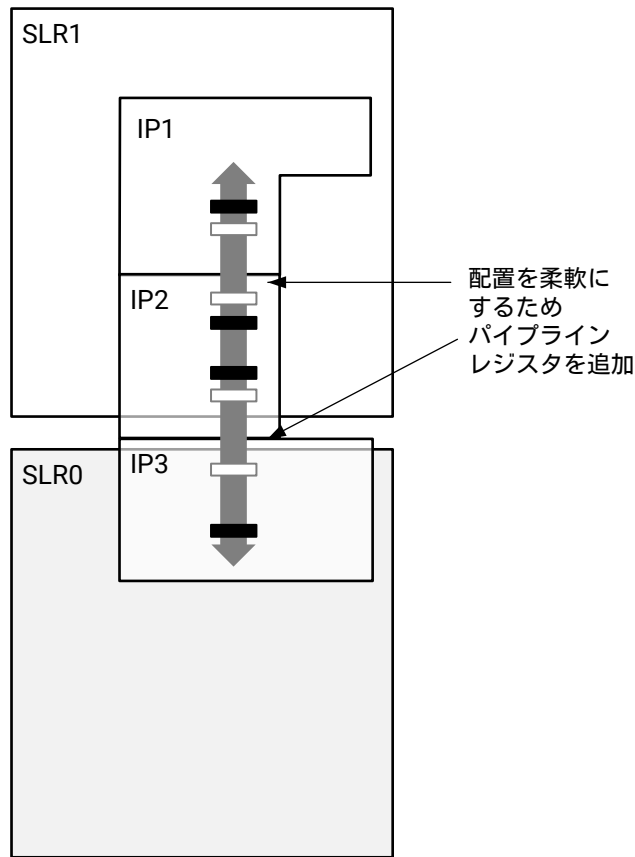
`USER_SLR_ASSIGNMENT` プロパティは SLR 分割中はソフト制約ですが、Pblock は SLR 分割中およびグローバル配置中はハード制約です。Pblock とは異なり、`USER_SLR_ASSIGNMENT` は配置により有効な分割方法を見つけるために無視されることがあります。`USER_SLR_ASSIGNMENT` および Pblock は両方とも、タイミングを向上するため、配置および物理最適化で SLR 境界付近の最下位セルの配置を詳細に調整することが可能です。これらの調整には、パイプライン レジスタを SLR 境界の反対側に移動することなどが含まれます。このようなレジスタの移動は、Pblock 境界では不可能です。

次の例では、デザインに 3 つのタイミング クリティカルな階層ブロック (IP1、IP2、および IP3) が含まれており、SLR が 2 つあるデバイスをターゲットとしています。IP1 と IP2 を SLR1 に、IP3 を SLR0 に配置するには、次の XDC 制約を適用します。

```
set_property USER_SLR_ASSIGNMENT SLR1 [get_cells {IP1 IP2}]
set_property USER_SLR_ASSIGNMENT SLR0 [get_cells IP3]
```

次の図に、この配置結果を示します。パフォーマンスを向上するため、デバイス内の長距離移動にパイプライン段を追加できます。これは特に、IP2 と IP3 間をまたぐ部分で有益です。詳細な配置および `phys_opt_design` では、タイミングが向上するのであれば、IP2 と IP3 からのパイプライン レジスタが SLR 境界を越えて自動的に移動されます。

図 152: USER_SLR_ASSIGNMENT プロパティの配置例



X21199-121919

USER_SLR_ASSIGNMENT を設定できない場合、または配置でクリティカルなパスが SLR の境界をまたぐ場合は、USER_CROSSING_SLR プロパティを設定して、SLR 間をまたぐ位置またはまたいではいけない位置を指定できます。このプロパティは通常、ネット ドライバーと同じ SLR に配置する必要のあるピンのネットまたは最下位ピンに、あるいはレジスタ チェーンの場合は SLR 間をまたぐ位置に適用します。このプロパティはブール値に設定し、SLR 間をまたぐ個々の部分を制約するためネットおよびピンに適用します。

- TRUE: ターゲット ネット オブジェクトが SLR 間をまたぐようにするか、ターゲット ピン オブジェクトが SLR 間をまたいで接続されるようにします。レジスタ間の接続でその間のファンアウトが 1 のものだけにのみ TRUE 値を適用できます。

注記: 任意のロジックに TRUE 値を使用することはできません。このオプションは、複数のインプリメンテーションストラテジを試すときに、レジスタのチェーンが常に特定のレジスタで SLR の境界をまたぐようにするのに便利です。

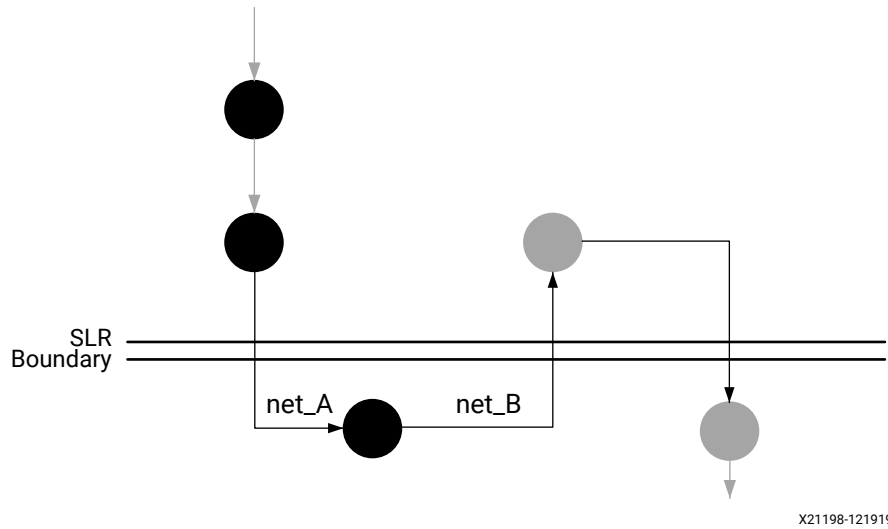
- FALSE: ターゲット ネット オブジェクトが SLR 間をまたがないようにするか、ターゲット ピン オブジェクトが SLR 間をまたいで接続されないようにします。FALSE 値は任意のネットまたはピンに適用できます。

注記: マクロ プリミティブ内にあるピンには適用できません。これらのピンは内部用であり、制約できません。

次の例では、パイプライン レジスタ チェーンが SLR 間を 2 回またいでおり、非効率なジグザグなパスになっています。

注記: 次の 2 つの図で、丸はレジスタ段を示します。

図 153: USER_CROSSING_SLR プロパティ 設定前の最適でない SLR 間をまたぐパス

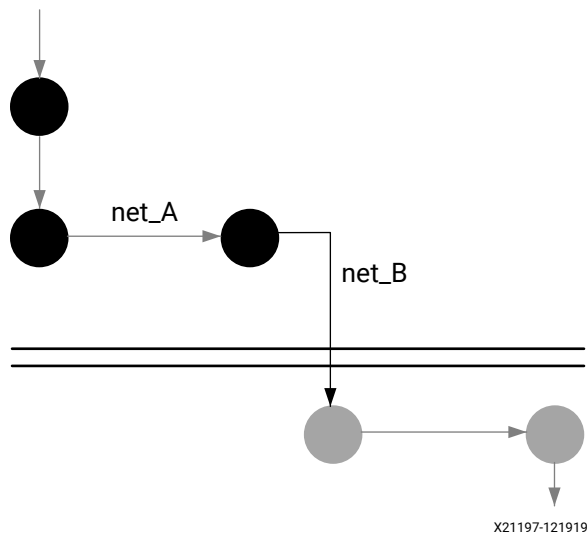


net_B のみが SLR 間をまたぐ最適な配置を達成するには、次の XDC 制約を適用します。

```
set_property USER_CROSSING_SLR FALSE [get_pins -leaf -of [get_nets net_A]]
set_property USER_CROSSING_SLR TRUE [get_pins -leaf -of [get_nets net_B]]
```

これにより、次の図に示すように、net_B でのみ SLR 間をまたぐ配置が得られます。

図 154: USER_CROSSING_SLR プロパティ 設定後の最適な SLR 間をまたぐパス



SLR をまたぐレジスタの使用

UltraScale+ SSI テクノロジ デバイスをターゲットとする場合は、SLR をまたぐレジスタからレジスタへの接続を Laguna RX_REG を駆動する Laguna TX_REG に直接マップできます。このタイプの接続は、Vivado 配線でローカル プログラマブル クロック遅延を設定することによりホールド タイム違反を修正可能な UltraScale+ デバイス ファミリでのみ可能です。パイプライン レジスタ間の接続に TX_REG から RX_REG への SLR をまたぐトポロジを使用すると、次のようなパフォーマンスの利点があります。

- SLR をまたぐ配置が垂直方向に分散され、SLR 境界での配線の密集が削減します。
- レジスタを Laguna サイトに配置すると、遅延見積もりの精度が向上し、タイミング QoR が高くなります。
- SLR をまたぐ部分のパフォーマンスが高速でより一貫したものになります。

注記: UltraScale SSI テクノロジ デバイスをターゲットとする場合は、SLR をまたぐネットには Laguna TX_REG または RX_REG のいずれかのみを使用可能であり、両方を同時に使用することはできません。パフォーマンスの利点は先ほどリストしたものと同様です。

SLR の境界に配置されると予測されるレジスタに USER_SLL_REG プロパティを設定して、Laguna レジスタ サイトに配置されるようにできます。レジスタの D および Q ピンが SLR の境界をまたがないネットまたは複数の SLR に配置されるロードを駆動するネットに接続されている場合は、USER_SLL_REG 制約は `place_design` で無視されます。次に例を示します。

```
set_property USER_SLL_REG TRUE [get_cells {reg_A reg_B}]
```

SLR をまたぐレジスタを Laguna にマップするのに確実な方法は、BEL および LOC 制約の両方をレジスタに適用し、配置を固定する方法です。LOC 値は Laguna サイトを指定し、BEL 値はそのサイト内の特定の Laguna レジスタ (6 個の TX_REG レジスタまたは 6 個の RX_REG レジスタのいずれか) を選択します。Laguna の SLR をまたぐレジスタは一定の距離で配置されており、直接接続のため各 TX_REG レジスタは RX_REG レジスタとペアになっています。

次の例では、レジスタ間の接続を TX_REG から RX_REG への接続に手動で配置します。パイプライン レジスタ `reg_A` は、ロードがレジスタ `reg_B` のみの 1 つのファンアウトを駆動します。ターゲット デバイスが `vu9p` の場合、次の XDC 制約を適用し、TX_REG から RX_REG への直接接続を使用して SLR2 の `reg_A` が SLR1 の `reg_B` を駆動するようにしています。

```
set_property BEL TX_REG3 [get_cells reg_A]
set_property BEL RX_REG3 [get_cells reg_B]
set_property LOC LAGUNA_X2Y480 [get_cells reg_A]
set_property LOC LAGUNA_X2Y360 [get_cells reg_B]
```

まず BEL を割り当て、レジスタ位置 (0、1、... 5) が TX_REG と RX_REG の間 (この例では 3) に一致するようにします。ペアとなっている Laguna サイトの間の距離は 120 行です。レジスタ `reg_A` は、SLR2 Laguna 列の 1 番下の行から SLR1 Laguna 列の 1 番下の行を駆動します。Laguna の BEL および LOC 制約を作成する際は、同じクロック、クロック イネーブル、リセットを使用するレジスタをグループ化し、制御セットの互換性の問題が発生しないようにしてください。

SLR をまたぐパスでの自動パイプラインの使用

ソフト SLR フロアプラン制約またはハード SLR フロアプラン制約を使用する場合、あるいはフロアプラン制約を使用しない場合のいずれでも、異なる SLR に配置されているデザインの主要部分の間のタイミングを満たすために必要なパイプライン段数は次の条件によって異なります。

- ターゲット周波数
- デバイスのフロアプラン
- デバイスのスピードグレード

自動パイプライン機能を使用して、配置アルゴリズムで必要な段数と最適な場所が判断されるようにすると、SLR の境界をまたぐ部分のタイミング クロージャを達成するのに有益です。この機能を使用すると、ほかに何も設定しなくても、Vivado 配置で自動的に Laguna レジスタが使用されます。

自動パイプラインをイネーブルにするには、RTL でバスおよびハンドシェイク ロジックに AUTOPIPELINING_* 属性を設定しますが、追加されるレイテンシによりデザインの機能に悪影響が出ないことを確認してください。または、ザイリンクス AXI Register Slice Memory Mapped または Streaming IP を SLR をまたぐパスに設定して使用します。

関連情報

[自動パイプライン処理に関する注意事項](#)

消費電力クロージャ

消費電力の重要性を考慮し、Vivado ツールでは正確な消費電力見積もりを取得する方法がサポートされており、消費電力最適化の機能が提供されています。詳細は、『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』(UG907)を参照してください。



推奨: UltraScale および UltraScale+™ デバイスをターゲットとして -directive の Explore 指示子または Explore ベースのストラテジを使用する場合は、opt_design を実行した後に power_opt_design コマンドを実行するか opt_design -bram_power_opt を使用して、ブロック RAM の消費電力最適化を手動でイネーブルにする必要があります。ザイリンクスでは、消費電力を削減するため、ブロック RAM をターゲットとすることをお勧めします。

フローを通した消費電力見積もり

デザイン フローを合成からインプリメンテーションに進行していく際、消費電力を定期的に調べ、消費電力が要件内であり、ボード電源レギュレータが現在の動作範囲内で動作し、デザインがシステムの電源範囲内であることを確認する必要があります。そうすることにより、消費電力が要件に近づきすぎた場合にすばやく対処できます。

XDC 制約を使用して消費電力マージンをレポートするために消費電力バジェットを指定するには、次のコマンドを使用します。

```
set_operating_conditions -design_power_budget <value in watts>
```

この値は、report_power コマンドで使用されます。計算されたオンチップ消費電力と消費電力バジェットの差が消費電力マージンで、計算された消費電力が消費電力バジェットを超えている場合は Vivado IDE で赤で表示されます。これにより、フローを通して消費電力を監視しやすくなります。



ヒント: UltraScale+ デバイスでは、環境設定を含む XPE から XDC ファイルをエクスポートできます。この XDC ファイルには、消費電力バジェット制約として使用可能な XPE 見積もりが含まれます。XPE または XDC を使用して消費電力バジェットを変更できます。消費電力マージンのレポートに XDC 制約を追加します。

消費電力見積もりの精度は、デザインの段階によって異なります。合成後からインプリメンテーションでは、消費電力の見積もりに report_power コマンドを使用するか、Vivado IDE で消費電力レポートを開きます。

- 合成後: ネットリストがターゲット デバイスで使用可能な実際のリソースにマップされます。
- 配置後: ネットリスト コンポーネントが実際のデバイス リソースに配置されます。このパック情報を基に最終的なロジック リソース数およびコンフィギュレーションが判明します。Xilinx® Power Estimator スプレッドシートにこのデータをエクスポートできます。このコマンドを使用すると、次が可能になります。

- XPE で What-if 解析を実行します。
- 同様の特性を持つ今後のデザインでスプレッドシートを入力する際の基準を提供します。
- 配線後: 配線が完了すると、使用される配線リソースに関するすべての詳細およびデザインに含まれる各パスの正確なタイミング情報が定義されます。

シミュレータでは、インプリメントされた回路の機能をベスト ケースおよびワースト ケースのロジックおよび配線遅延で検証することに加え、グリッチを含む内部ノードの的確なアクティビティがレポートされます。このレベルの消費電力解析では、プロトタイプボードで消費電力を実際に計測する前に、最も正確な消費電力見積もりが得られます。

消費電力制約アドバイザーの使用

消費電力制約アドバイザーは、デザイン内の制御信号すべてのツールで算出されたスイッチング アクティビティを、ファンアウトの大きい順にレポートします。このリストで信頼性レベルの低いものを確認します。これは、スイッチング アクティビティの高いリセットと、スイッチング アクティビティが 0 または 0 に近いイネーブルを示します。どちらも、消費電力が誤ってよく見積もられる要因となります。詳細は、『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』 (UG907) の消費電力制約アドバイザーを参照してください。

推奨される消費電力制約

デザイン クロージャを達成するには、正しい消費電力制約を適用することが重要です。Vivado ツールの `report power` コマンドは、適用されたバジェットおよび追加の制約に基づいて、消費電力マージンをレポートします。次に、最小限推奨される制約をリストします。詳細は、『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』 (UG907) を参照してください。

推奨される最小限の制約

次の制約の例は、消費電力見積もりで消費電力バジェットをチェックし、スタティック消費電力解析にワースト ケースの最大プロセスを使用します。

```
set_operating_conditions -design_power_budget <Power in Watts>
set_operating_conditions -process maximum
```

推奨される追加の制約

次の制約を使用して熱ソリューションを定義すると、`report_power` コマンドを使用してジャンクション温度を見積もりことができ、より正確なスタティック消費電力を得ることができます。

```
set_operating_conditions -ambient_temp <max Ambient requested for
application is Celsius>
set_operating_conditions -thetaja <the rise in junction temperature for
every watt dissipated, obtained from thermal simulation, C/W>
```

Vivado ツールの `report_power` コマンドでは、次の制約を使用することにより、レギュレータまたは電圧レギュレータ モジュール (VRM) ベースの消費電力をレポートすることもできます。

新規電源レールの作成

```
create_power_rail <power rail name> -power_sources {supply1, supply2 ,...}
```

既存の電源レールに電源ソースを追加

```
add_to_power_rail <power rail name> -power_sources {supply1, supply2, ..}
```

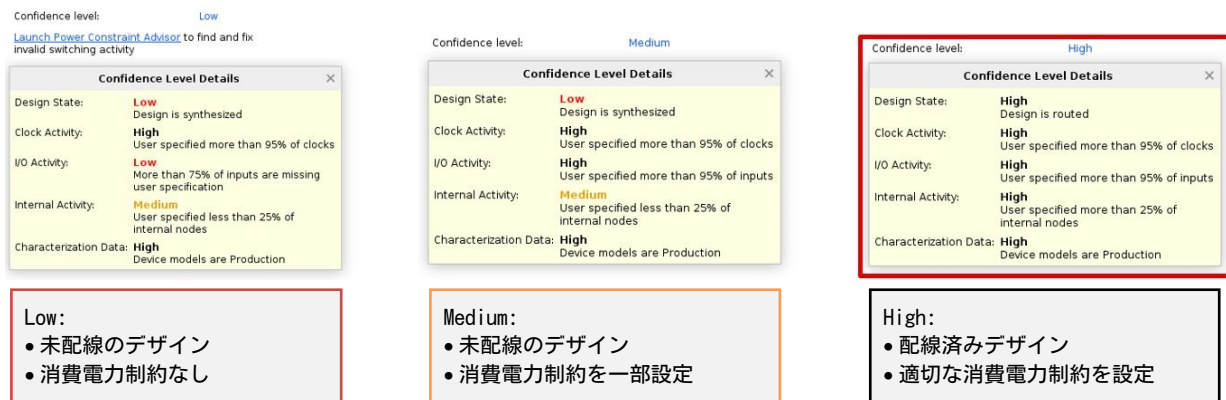
電流バジェットの定義

```
set_operating_conditions -supply_current_budget {<supply rail name>  
<current budget in Amp>} -voltage {<supply rail name> <voltage>}
```

正確な消費電力解析のためのベスト プラクティス

正確な消費電力解析のためには、タイミング制約、I/O 制約、およびスイッチング アクティビティが正しいことを確認します。report_power コマンドにより、次の図に示すように信頼性レベルが表示されます。正確な消費電力解析のため、信頼性レベルが High になるようにしてください。詳細は、『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』 (UG907) のこのセクションを参照してください。

図 155: 消費電力解析の信頼性レベル



X25127-021621

消費電力解析を実行した後にデザインの電力分配を確認

オンチップの合計消費電力、温度特性、リソース レベルでの消費電力の詳細を確認して、全消費電力に最も影響するデザインの部分を特定できます。詳細は、『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』 (UG907) のこのセクションを参照してください。



電源/消費電力ヒント: 現在の回路図/PCB に対する完成した Vivado デザインのデカップリング要件を確認および検証します。次の Tcl コマンドを使用して、Vivado ツールの `report_power` により .xpe ファイルを生成します。

```
set_operating_conditions -process maximum
```

```
set_operating_conditions -ambient_temp <max Ambient requested for  
application is Celsius>
```

```
set_operating_conditions -thetaja <the rise in junction temperature for  
every watt dissipated, obtained from thermal simulation, C/W>
```

```
report_power -xpe {C:/Design_Runs/Vivado_export.xpe} -name {Any_Name}
```

この .xpe ファイルを XPE にインポートします。XPE の [Power Delivery] シートに、消費電力見積もりおよび電力供給オプションに基づくデカップリング要件が示されます。

消費電力解析を実行した後に制御信号のアクティビティを調整

正確な消費電力解析用に SAIF ベースのアノテーションを使用していない場合は、初期段階の解析を実行した後に消費電力解析を調整できます。詳細は、『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』(UG907) の [制御信号のアクティビティの調整](#) を参照してください。

消費電力の最適化

消費電力見積もりがバジェットを超えている場合は、次の手順を使用して消費電力を削減する必要があります。

消費電力見積もりおよび最適化結果の解析

`report_power` を使用して消費電力見積もりレポートを生成したら、サイリンクスでは次を実行することをお勧めします。

- [Summary] セクションで総消費電力を確認します。総消費電力およびジャンクション温度はデザインの温度要件および消費電力要件を満たしていますか。
- 結果が要件を大幅に超えている場合は、ブロック タイプごとおよび電源レールごとの電力分配を確認します。これにより、消費電力が最も大きいブロックを特定できます。
- [Utilization Details] の下の [Hierarchical] ページを確認します。階層ごとに情報が示されているので、電力が最も消費されているモジュールを特定できます。特定のモジュールを展開表示すると、各ブロックの消費電力を確認できます。クロスプローブを使用して、モジュールの特定のセクションがどのようにコード記述されているか、消費電力を削減するようコードを記述し直せるかを判断できます。

注記: デザインにタイミング マージンがある場合は、複数の run を実行して総消費電力が低いものがあるかどうかを評価します。たとえば、マージンが 2 ps のデザインとマージンが 15 ps のデザインのパフォーマンスが同様でも、マージンが 2 ps のデザインの方が消費電力が低い場合があります。

消費電力最適化の実行

消費電力最適化は、消費電力を最小にするため、デザイン全体またはデザインの一部 (`set_power_opt` を使用する場合) に実行できます。

消費電力最適化は、デザイン フローの配置前または配置後のいずれかで実行できますが、両方では実行できません。配置前の消費電力最適化では、消費電力の削減量を最大にすることに焦点が置かれ、まれにタイミングが悪化することもあります。タイミングの保持が主な目標の場合は、ザイリンクスでは消費電力最適化を配置後に実行することをお勧めします。この場合、タイミングを保持する消費電力最適化のみが実行されます。

レガシ (IP) またはタイミングの面からデザインの一部を保持する必要がある場合は、`set_power_opt` コマンドを使用して特定の階層、クロック ドメイン、またはセル タイプなどを除外してから、消費電力最適化を実行します。

関連情報

[消費電力を向上するためのコーディング スタイル](#)

消費電力最適化レポートの使用

消費電力最適化の影響を判断するため、Tcl コンソールで次のコマンドを実行し、消費電力最適化レポートを生成します。

```
report_power_opt -file myopt.rep
```

タイミング レポートを使用して消費電力最適化の影響を確認

消費電力最適化では、消費電力を最大限に削減しながら、タイミングへの影響は最小限に抑えます。消費電力最適化によりタイミングが悪化した場合は、この影響を補正するための手法を使用できます。

可能な場合、タイミング クリティカルでないクロック ドメインまたはモジュールを特定し、`set_power_opt` XDC コマンドを使用してそれらだけに消費電力最適化を適用します。最もクリティカルなクロック ドメインがデザインの大部分を占めている場合や、電力の大部分を消費している場合は、クリティカル パス上のセルで `IS_CLOCK_GATED` が `TRUE` に設定され、このパスが消費電力最適化の結果であることが示されていないかを確認します。次のインプリメンテーションで消費電力が大きくなってもタイミングが向上するようにするには、`set_power_opt` XDC 制約を使用してクリティカル パスの消費電力が最適化されたセルの消費電力最適化をディスエーブルにし、その `set_power_opt` XDC 制約を含めてインプリメンテーションを再実行するか、Tcl コマンドを使用してインプリメンテーションを再実行します。

次の Tcl 例では、タイミングが満たされていない上位 100 個のパスにあるセルの消費電力最適化をディスエーブルにしています。

```
set pwr_critical_cells [get_cells -of [get_timing_paths -slack_lesser_than  
0 -max_paths 100] -filter {IS_CLOCK_GATED}]  
set_power_opt -exclude_cells $pwr_critical_cells
```

コンフィギュレーションおよびデバッグ

デザイン インプリメンテーションを完了したら、デバイスにデザインを読み込み、ハードウェア上で実行します。コンフィギュレーションは、アプリケーション特定のデータをデバイスの内部メモリに読み込むプロセスです。デザインがハードウェア上の要件を満たさない場合はデバッグが必要です。

コンフィギュレーションおよびデバッグのソフトウェア フローおよびコマンドについては、次のユーザー ガイドを参照してください。

- 『Vivado Design Suite ユーザー ガイド: プログラムおよびデバッグ』 ([UG908](#))

- 『Vivado Design Suite Tcl コマンド リファレンス ガイド』 (UG835)
- 『7 シリーズ FPGA コンフィギュレーション ユーザー ガイド』 (UG470: [英語版](#)、[日本語版](#))
- 『UltraScale アーキテクチャ コンフィギュレーション ユーザー ガイド』 (UG570: [英語版](#)、[日本語版](#))
- [Vivado Design Suite QuickTake ビデオ: Vivado で write_bitstream コマンドを使用する方法](#)

コンフィギュレーション

ビットストリーム イメージを作成するには、まずデザインの合成およびインプリメンテーションを正しく完了させる必要があります。ビットストリームを生成してすべての DRC を解析して修正したら、次のいずれかの方法を使用してビットストリームをデバイスに読み込むことができます。

- 直接プログラム:

ケーブル、プロセッサ、またはカスタム ソリューションを使用して、デバイスに直接ビットストリームを読み込みます。

- 間接プログラム: ビットストリームを外部フラッシュ メモリに読み込みます。この後、フラッシュ メモリからビットストリームをデバイスに読み込みます。

Vivado ツールを使用すると、次を実行できます。

- ビットストリーム (.bit または .rbit) を作成。
- [Tools]→[Edit Device] をクリックし、ビットストリーム生成用のコンフィギュレーション設定を確認。
- ビットストリームをフラッシュ プログラム ファイル (.mcs) にフォーマット。
- 次のいずれかの方法を使用してデバイスをプログラム:

- デバイスを直接プログラム。
- 接続されているコンフィギュレーション フラッシュ デバイスを間接的にプログラム。

フラッシュ デバイスは不揮発性デバイスで、プログラム前に消去する必要があります。チップ全体の消去が指定されない限り、割り当てられている MCS で指定されているアドレス範囲のみが消去されます。



重要: Vivado Design Suite デバイス プログラマでは、JTAG を使用してサイリンクス デバイスのステータス レジスタ データを読み出すことができます。コンフィギュレーション エラーが発生した場合、ステータス レジスタに問題の原因を特定するのに役立つエラー状態がキャプチャされます。また、ステータス レジスタでモード ピン設定 M[2:0] およびバス幅検出を確認することもできます。ステータス レジスタの詳細は、該当するデバイスのコンフィギュレーション ユーザー ガイドを参照してください。



ヒント: コンフィギュレーションで問題が発生した場合、デバイスに対して JTAG リードバック/検証操作を実行し、コンフィギュレーション データが正しくデバイスに読み込まれたかどうかを確認できます。

デバッグ

インシステム デバッグでは、ターゲット デバイス上でリアルタイムにデザインをデバッグできます。この手順は、シミュレータで再現しづらい状況に遭遇した場合に必要です。

デバッグでは、デザインを監視および制御できるようにするための特別なデバッグ IP をデザインに追加します。デバッグが完了したら、パフォーマンスを高め、ロジック使用率を低減するため、このデバッグ IP を取り除くことができます。

デザインのデバッグは複数の段階を含む反復作業です。複雑な問題を処理する場合と同様に、デザインのデバッグプロセスも、一度にデザイン全体を処理するのではなく、細分化してセクションごとに集中して作業するのがベストです。

実際のデバッグ手順はデザインのインプリメンテーションが問題なく終了した後になりますが、ザイリンクスではデザインサイクルの早期にデバッグをどのように実行するか、どこでデバッグするかをプランニングしておくことをお勧めします。デバイスのプログラムおよびインシステム デバッグの実行に必要なコマンドはすべて、Vivado (IDE) の Flow Navigator の [Program and Debug] から実行できます。

デバッグの手順は次のとおりです。

1. プロブ: デザインでプロブする信号を特定し、プロブ方法を指定します。
2. インプリメント: プロブするネットに追加されたデバッグ IP を含むデザインをインプリメントします。
3. 解析: デザインに含まれるデバッグ IP にアクセスし、機能的な問題をデバッグおよび検証します。
4. 修正: バグがあれば修正し、必要に応じて繰り返します。

詳細は、『Vivado Design Suite ユーザー ガイド: プログラムおよびデバッグ』 (UG908) を参照してください。

PL のデバッグ

PL のロジック シミュレーションで再現しづらい状況に遭遇した場合、プログラマブル ロジック (PL) のデバッグが必要となることがあります。このセクションでは、PL ドメイン内を可視化できるようにするデバッグ ツールについて説明します。

ILA コアの使用

Integrated Logic Analyzer (ILA) コアを使用すると、デバイス上でインプリメント後のデザインのインシステム デバッグを実行できます。このコアは、デザインの信号を監視する必要がある場合に使用できるほか、ハードウェア イベントをトリガーし、データをシステム速度でキャプチャするためにも使用できます。

デザインのプローブ

Vivado ツールでは、デザインにデバッグ プロブを追加するのに複数の方法があります。次の表に、それらの方法と、それぞれの長所と短所を示します。

表 16: デバッグ フロー

デバッグ フロー名	フローの段階	長所/短所
HDL インスタンスエーション プロブ フロー	HDL ソースまたは IP インテグレーター キャンバスで、ILA デバッグ コア インスタンスに信号を明示的に接続します。	<ul style="list-style-type: none">• デバッグ ネットおよび IP はデザインから手動で追加/削除する必要があるため、HDL ソースはユーザーが修正する必要があります。• HDL デザイン レベルでプロブするオプションあり。• インターフェイス レベルで AXI または AXI4-Stream など特定のプロトコルをプロブ可能。• デバッグ コアの生成、インスタンスエート、接続の際に、間違いやすい。

表 16: デバッグ フロー (続き)

デバッグ フロー名	フローの段階	長所/短所
ネットリスト挿入プローブ フロー	<p>デバッグする信号を選択するには、次のいずれかの方法を使用します。</p> <ul style="list-style-type: none"> ソース RTL コードで MARK_DEBUG 属性を使用してデバッグする信号をマーク。 合成済みデザイン ネットリストでネットを右クリックして [Mark Debug] をクリックし、デバッグするネットを選択。 <p>デバッグする信号をマークしたら、[Set Up Debug] ウィザードを使用してネットリスト挿入プローブ フローを実行します。</p>	<ul style="list-style-type: none"> 柔軟性が最も高く、予測しやすい。 さまざまなデザイン レベル (HDL、合成済みデザイン、システム デザイン) でのプローブが可能。 HDL ソースを修正する必要なし。
Tcl ベースのネットリスト挿入プローブ フロー	<p>set_property Tcl コマンドを使用してデバッグ ネットに MARK_DEBUG プロパティを設定し、ネットリスト挿入プローブ Tcl コマンドを使用してデバッグ コアを作成し、それらをデバッグ ネットに接続します。</p>	<ul style="list-style-type: none"> ネットリスト挿入を完全に自動実行。 デバッグのオン/オフを Tcl コマンドで切り替え可能。 HDL ソースを修正する必要なし。

関連情報

[インプリメント済みネットリストを変更して既存のデバッグ プローブを置換](#)

デバッグ ネットの選択

ザイリンクスでは、デバッグ ネットを選択する際に次の推奨事項に従うことをお勧めします。

- 特定の階層の境界 (入力または出力) でネットをプローブすると、問題のある箇所をすばやく特定できます。その後、必要に応じてその階層をさらにプローブできます。
- 組み合わせロジック パス間のネットはプローブしないでください。組み合わせロジック パスにあるネットに MARK_DEBUG を追加すると、インプリメンテーション段階で適用可能であった最適化が適用されなくなり、タイミング QoR (結果の品質) が低下します。
- サイクル精度のデータを取得するため同期ネットをプローブします。

MARK_DEBUG を使用したデバッグ プローブ ネットの名前の保持

デバッグする信号は、RTL の段階または合成後にマークできます。ネットに MARK_DEBUG 属性を設定すると、ネットの複製、リタイミング、名前の変更、削除などの最適化は実行されなくなります。MARK_DEBUG は、最上位ポート、ネット、階層モジュール ポート、階層モジュール内部のネットに適用できます。この方法では、ほとんどの場合 HDL 信号が合成後に保持されます。デバッグ用にマークされたネットは、合成後に [Debug] ウィンドウの [Unassigned Debug Nets folder] に表示されます。

HDL ファイルに mark_debug 属性を追加するには、次のように指定します。

VHDL:

```
attribute mark_debug : string;
attribute keep : string;
attribute mark_debug of sine : signal is "true";
```

Verilog:

```
(* mark_debug = "true" *) wire sine;
```

デバッグ用のネットは、合成後のネットリストにも追加できます。この場合、HDL ソースを変更する必要はありません。ただし、ネットリストの最適化によりデザイン構造が吸収または統合されたために、合成で元の RTL 信号が保持されない場合があります。合成後にデバッグ用にネットを追加するには、次のいずれかの方法を使用できます。

- [Netlist]、[Schematic] などのデザイン ウィンドウでネットを右クリックし、[Mark Debug] をクリックします。
- デザイン ウィンドウのいずれかでネットを選択し、[Unassigned Debug Nets] フォルダーにドラッグ アンド ドロップします。
- [Set Up Debug] ウィザードのネット セレクターを使用します。
- [Properties] ウィンドウまたは Tcl コンソールで MARK_DEBUG プロパティを設定します。

```
set_property mark_debug true [get_nets -hier [list {sine[*]}]]
```

このコマンドにより、現在開いているネットリストに mark_debug プロパティが追加されます。この方法は、Tcl コマンドで MARK_DEBUG のオン/オフを切り替えることができるので柔軟です。

ILA コアとタイミングに関する考慮事項

ILA コアを設定すると、デザイン全体のタイミング目標の達成に影響します。タイミングへの影響を最小限に抑えるため、次の推奨事項に従うことをお勧めします。

- プローブ幅を注意して選択します。プローブ幅が大きいほど、リソース使用量およびタイミングへの影響も大きくなります。
- ILA コアのデータの深さを注意して選択します。データの深さが大きいほど、ブロック RAM リソース使用量およびタイミングへの影響も大きくなります。
- ILA に選択するクロックはフリーランニング クロックにします。そうでないと、デザインがデバイスに読み込まれたときに、デバッグ コアと通信できなくなる可能性があります。
- dbg_hub へのクロックはフリーランニング クロックにします。そうでないと、デザインがデバイスに読み込まれたときに、デバッグ コアと通信できなくなる可能性があります。Tcl コマンドの connect_debug_port を使用すると、デバッグ ハブの clk ピンをフリーランニング クロックに接続できます。
- デバッグ コアを追加する前にデザインのタイミング クロージャを達成しておきます。ザイリンクスでは、デバッグ コアをタイミング関連の問題をデバッグするために使用することはお勧めしません。
- ILA デバッグ コアを追加したためにタイミングが悪化し、クリティカル パスが dbg_hub にある場合は、次を実行してください。
 1. 合成済みデザインを開きます。
 2. ネットリストで dbg_hub セルを見つけます。
 3. dbg_hub の [Properties] ウィンドウに移動します。
 4. C_CLK_INPUT_FREQ_HZ プロパティを見つけます。
 5. dbg_hub に接続されるクロックの周波数 (Hz) をそれに設定します。
 6. C_ENABLE_CLK_DIVIDER プロパティを見つけ、オンにします。
 7. デザインをインプリメントし直します。
- ILA コアへのクロック入力がプローブする信号と同期していることを確認します。そうでないと、デザインをデバイスにプログラムしたときに、タイミング問題が発生したり、デバッグ コアと通信できなくなったりする可能性があります。
- ハードウェアでの実行前に、デザインのタイミングが満たされていることを確認します。そうでないと、プローブされた波形の信頼性が低くなります。

次の表に、特定の ILA 機能を使用した場合のデザイン タイミングおよびリソースへの影響を示します。

注記: この表は 1 つの ILA を含むデザインに関するものであり、すべてのデザインに当てはまるとは限りません。

表 17: ILA 機能のデザイン タイミングおよびリソースへの影響

ILA 機能	使用する状況	タイミング	エリア
キャプチャ制御/ストレージ必要条件	関連データをキャプチャする データ キャプチャ ストレージ (ブロック RAM) を効率的に使用 する	影響: 中～大	<ul style="list-style-type: none"> 追加のブロック RAM なし LUT/FF 数が多少増加
アドバンス トリガー	BASIC トリガー条件が不十分な 場合 問題のエリアに焦点を置くため に複雑なトリガーを使用する場 合	影響: 大	<ul style="list-style-type: none"> 追加のブロック RAM なし LUT/FF 数が中程度増加
プローブ ポートごとの コンパレータ数 注記: 最大値は 4 です。	複数の条件文でプローブを使用 する場合 <ul style="list-style-type: none"> 基本トリガー: 1 ～ 2 アドバンス トリガー: 1 ～ 4 キャプチャ制御: 1 以上 	影響: 中～大	<ul style="list-style-type: none"> 追加のブロック RAM なし LUT/FF 数が多少から中程度 増加
データの深さ	より多くのデータ サンプルをキャ プチャする	影響: 大	<ul style="list-style-type: none"> ILA コアごとに追加のブロッ ク RAM LUT/FF 数が多少増加
ILA プローブ ポート幅	スカラーでなく大型バスをデバ ッグする	影響: 中	<ul style="list-style-type: none"> ILA コアごとに追加のブロッ ク RAM LUT/FF 数が多少増加
プローブ ポート数	多数のネットをプローブするた め	影響: 小	<ul style="list-style-type: none"> ILA コアごとに追加のブロッ ク RAM LUT/FF 数が多少増加



ヒント: デザインの初期段階では通常、デバイス上にデバッグに使用可能なリソースが多数あります。

高速クロックを使用する ILA コア デザイン

高速クロックを使用するデザインでは、次を考慮してください。

- デバッグする信号の数および幅を制限します。
- ILA への入力プローブをパイプライン処理し (C_INPUT_PIPE_STAGES)、追加のパイプライン段をイネーブルにします。

注記: 使用可能な MMCM/BUFG が制限されるデザインでは、デバッグ ハブ内のクロック分周期を使用するのではなく、デバッグ ハブをデザインの最低のクロック周波数を供給することを考慮してください。

VIO コアの使用

VIO (Virtual Input/Output) コアは、内部デバイス信号をリアルタイムに監視および駆動できるようにします。このコアは、リセットやステータス信号などの低速信号を駆動または監視する必要がある場合に使用します。VIO デバッグ コアはデザインにインスタンスエートする必要があり、Vivado IP インテグレーター ブロック デザインおよび RTL の両方で使用できます。VIO コアは、RTL ベース デザインでは IP カタログから、および IP インテグレーターで使用できます。

VIO コアのカスタマイズの詳細は、『Virtual Input/Output LogiCORE IP 製品ガイド』 (PG159) を参照してください。
VIO コアを使用した計測方法の詳細は、『Vivado Design Suite ユーザー ガイド: プログラムおよびデバッグ』 (UG908) のこのセクションを参照してください。

VIO コアに関する考慮事項

VIO コアを使用する場合は、次を考慮します。

- VIO 入力プローブに接続されている信号は、VIO コアの VIO `clk` ポートに接続されているクロックと同期している必要があります。`clk` ポートに同期していない信号を接続すると、VIO 入力プローブ ポートでクロック乗せ換えが発生します。
- VIO 出力プローブに接続されている信号は、VIO コアの VIO `clk` ポートに接続されているクロックと同期している必要があります。
- VIO コアは、プッシュボタンや LED などの低速ボード I/O に置き換わるものなので、更新レートは比較的低くなっています。高速信号をキャプチャするには、ILA コアを使用することを考慮してください。

Vivado IP インテグレーターを使用したデザインのデバッグ

Vivado IP インテグレーターでは、デザインをデバッグ用に設定する方法が複数あります。次のいずれかのフローを使用して、デバッグ コアを IP インテグレーター デザインに追加できます。どのフローを選択するかは、ユーザーの好みと、デバッグするネットおよび信号のタイプによります。

- System ILA コアを使用してブロック デザインのインターフェイス、ネット、またはその両方をデバッグします。
このフローは、次を実行する場合に使用します。
 - MicroBlaze™ デバイス、Zynq®-7000 SoC、または Zynq UltraScale+ MPSoC のクロス トリガー機能を使用したハードウェア/ソフトウェアの協調検証。
 - インターフェイス レベルの接続性を検証。
- ネットリスト挿入フロー
このフローは、合成後のデザインで I/O ポートおよび内部ネットを解析する場合に使用します。

注記: これらのフローを組み合わせ使用してデザインをデバッグすることも可能です。

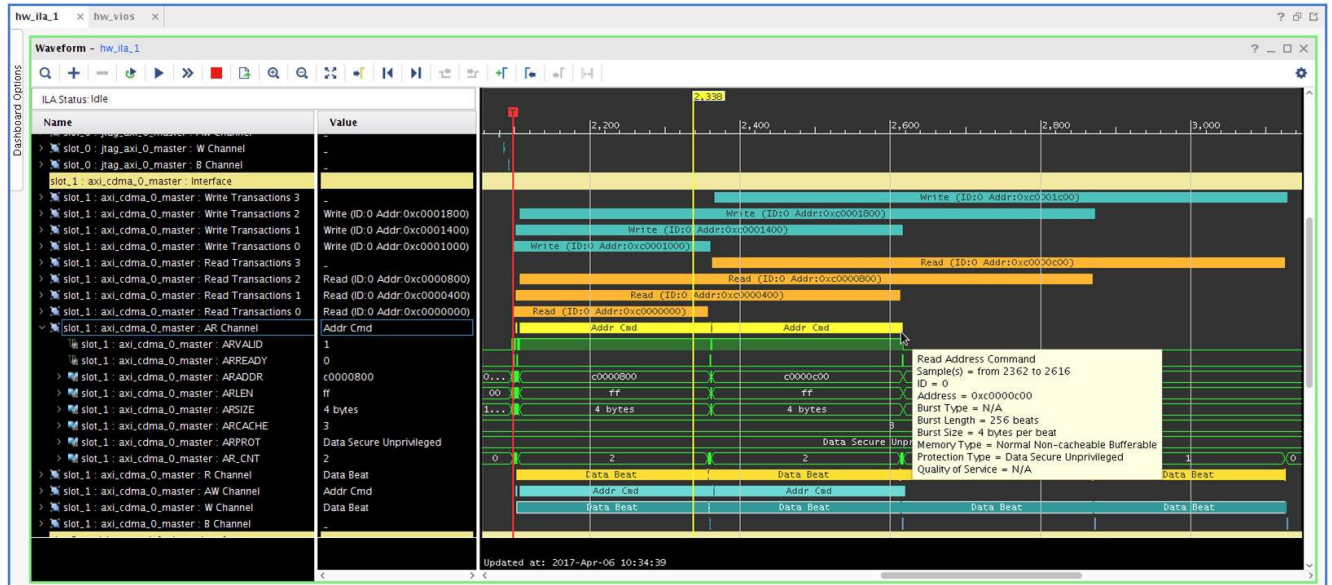
IP インテグレーター デザインでの System ILA の使用に関する詳細は、『Vivado Design Suite ユーザー ガイド: IP インテグレーターを使用した IP サブシステムの設計』 (UG994) を参照してください。

Vivado ハードウェア マネージャーを使用した AXI インターフェイスのデバッグ

ILA を使用すると、ザイリンクス デバイス上でインプリメント後のデザインのインシステム デバッグを実行できます。この機能は、デザインでインターフェイスおよび信号を監視する必要がある場合に使用します。

ILA モードを [Interface] に変更すると、次の図に示すように、[Waveform] ウィンドウで AXI トランザクションと読み出しおよび書き込みイベントをデバッグおよび監視できます。[Waveform] ウィンドウには、インターフェイス スロット、トランザクション、イベント、ILA のインターフェイス スロットでプローブされるインターフェイスに対応する信号グループが表示されます。

図 156: [Waveform] ウィンドウ



System ILA の詳細および Vivado ハードウェア マネージャーでの AXI インターフェースのデバッグについては、『Vivado Design Suite ユーザー ガイド: プログラムおよびデバッグ』 (UG908) の[このセクション](#)および[このセクション](#)を参照してください。

In-System IBERT の使用

In-System IBERT コアを使用すると、UltraScale および UltraScale+ デバイスのトランシーバーの RX データに対するアイ スキャン プロットを使用した RX マージン解析が可能になります。GTH/GTY トランシーバーを設定および調整でき、トランシーバーのダイナミック リコンフィギュレーション ポート (DRP) と通信するロジックを介してアクセス可能です。このコアを使用して、属性設定と、rxrate、rxlpmen、txdiffctrl、txpostcursor、および txprecursor ポートの値を制御するレジスタを変更できます。

デザインをデバイス上にプログラムすると、ハードウェア マネージャーの Vivado シリアル I/O 解析ツールは JTAG を介してコアと通信します。デザインごとに、In-System IBERT のインスタンスが 1 つあります。In-System IBERT は、デザインで使用されているすべての GT で使用できます。ただし、異なる GT タイプ (GTH、GTY など) には、別の In-System IBERT コアを生成する必要があります。

In-System IBERT デザインを内部システム クロックを使用して作成すると、スキャンが実行されないようにすることができます。アイ スキャンを作成すると、ステータスが [In Progress] から [Incomplete] に変わります。内部システム クロック (MGTRFCLK) を In-System IBERT IP の clk/drpcik_i 入力に接続すると、アイ スキャンのステータスは Incomplete になります。

注記: 必要な場合、外部クロックを使用するとこの動作は発生しません。または、Vivado シリアル I/O 解析で使用可能なリンクをクリックします。[Properties] ウィンドウで [LOGIC] フィールドの下に [MB_RESET reg] を見つけ、1 に設定した後 0 に戻します。アイ スキャンまたはスイープを再実行します。

このコアの詳細は、『In-System IBERT LogiCORE IP 製品ガイド』 (PG246) を参照してください。

デバッグ関連の DRC の実行

Vivado Design Suite ではデバッグ関連の DRC が提供されており、report_drc を実行する際のデフォルト ルール デックの一部として選択されています。これらの DRC では、次がチェックされます。

- デバッグ コアの現在の要件のためにデバイスのブロック RAM リソースが超過していないか。
- クロック以外のネットがデバッグ コアのカロック ポートに接続されていないか。
- デバッグ コアのポートが未接続になっていないか。

インプリメント済みネットリストを変更して既存のデバッグ プローブを置換

配置配線済みのデザイン チェックポイントで、ILA コアに接続されているデバッグ ネットを置き換えることができます。これには、通常エンジニアリング チェンジ オーダー (ECO) を使用します。これは、完成に近づいており、既存の ILA プローブ ポートに接続されているネットを交換する必要があるデザインに使用されるアドバンス デザイン フローです。ECO フローを使用して既存の ILA コアのネットを変更する方法の詳細は、『Vivado Design Suite ユーザーガイド: インプリメンテーション』 (UG904) のこのセクションを参照してください。

インプリメント済みネットリストでの ILA コアの挿入、削除、および変更

ILA コアを追加、削除、または変更する場合は (プローブの幅の変更、データ深さの変更など)、ザイリンクスではインクリメンタル コンパイル フローを使用することをお勧めします。デバッグ コア用のインクリメンタル コンパイル フローは、合成済みデザインまたはチェックポイント (DCP) に対して実行し、基準のインプリメント済みチェックポイント (理想的には前回のインプリメンテーション run からのチェックポイント) を使用します。この方法を使用すると、デザインを完全に再インプリメントするよりも時間を短縮できます。

インクリメンタル コンパイル フローを使用して ILA コアを挿入、削除、または変更する方法の詳細は、『Vivado Design Suite ユーザー ガイド: プログラムおよびデバッグ』 (UG908) のデバッグ コア (ILA) を変更した場合のインクリメンタル コンパイルを参照してください。

配線後の ECO を使用したネットの外部ピンへの接続

外部テスト装置を使用したデバッグのため、ネットを空いたデバイス ピンに接続することが必要な場合があります。これは、デバイスにこの目的に使用可能な空いたピンが 1 つ以上あることが必要です。

1. Vivado ツールを起動して配線後のデザイン チェックポイントを開きます。
2. [Layout]→[ECO] をクリックして ECO モードに切り替えます。
3. [ECO Navigator] で [Create Port] をクリックします。適切な [IOSTANDARD] を選択し、[Name] に名前を入力します。[Location] ダイアログ ボックスで、デバッグに使用する空いたパッケージ ピンを入力します。[OK] をクリックします。新しい I/O ポートが [Schematic] ウィンドウに表示されます。
4. [ECO Navigator] で [Create Cell] をクリックし、セル タイプとして [OBUF] を選択します。[Name] にセルの名前を入力し、[OK] をクリックします。
5. [Ctrl] キーを押しながら、前の手順の [OBUF] の [O] ポートと、手順 3 で作成したポートを選択します。[ECO Navigator] で [Create Net] をクリックします。[Name] に名前を入力し、[OK] をクリックします。[OBUF] が I/O ポートに接続されます。
6. [Schematic] ウィンドウで、外部ポートに接続するデザインのネットを見つけます。[Ctrl] キーを押しながら、デザインのネットと、前の手順で作成した [OBUF] の [I] ポートを選択します。
7. [ECO Navigator] で [Connect Net] をクリックし、ネットを [OBUF] に接続します。

これらの手順を繰り返し、必要なネットをすべて接続します。すべてのネットを適切なピンに接続したら、次の手順に従ってインクリメンタル配線を実行し、前の手順で加えた変更を含む新しいビットストリームを生成します。

1. [ECO Navigator] で [Route Design] をクリックします。[Incremental Route] (インターネット プロトコル バージョン 4 (TCP/IPv4)) を選択し、[OK] (プロパティ) をクリックします。
2. 配線が完了したら、[ECO Navigator] で [Generate Bitstream] をクリックして新しいビットストリームを生成します。デザインに ILA や VIO などの Vivado デバッグ コアが含まれている場合は、[Write Debug Probes] を選択して新しいデバッグ プローブ ファイルも生成する必要があります。

注記: 必要に応じて、[Programing the ECO Navigator] の下の [Save Checkpoint As] をクリックしてこの配線済みチェックポイントを保存できます。

3. 新しいビットストリームには、I/O ポートへの接続が含まれます。

リモート デバッグの使用

ザイリンクスでは、デザインをリモートでデバッグまたはアップグレードするための次のソリューションを提供しています。

- ザイリンクス ハードウェア サーバー製品を使用して、ラボのリモート コンピューターに接続します。

その他のリソースおよび法的通知

ザイリンクス リソース

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

ソリューション センター

デバイス、ツール、IP のサポートについては、[ザイリンクス ソリューション センター](#)を参照してください。デザイン アシスタント、デザイン アドバイザリ、トラブルシューティングのヒントなどが含まれます。

Documentation Navigator およびデザイン ハブ

ザイリンクス Documentation Navigator (DocNav) では、ザイリンクスの資料、ビデオ、サポート リソースにアクセスでき、特定の情報を取得するためにフィルター機能や検索機能を利用できます。DocNav を開くには、次のいずれかを実行します。

- Vivado® IDE で [Help] → [Documentation and Tutorials] をクリックします。
- Windows で [スタート] → [すべてのプログラム] → [Xilinx Design Tools] → [DocNav] をクリックします。
- Linux コマンド プロンプトに「docnav」と入力します。

ザイリンクス デザイン ハブには、資料やビデオへのリンクがデザイン タスクおよびトピックごとにまとめられており、これらを参照することでキー コンセプトを学び、よくある質問 (FAQ) を参考に問題を解決できます。デザイン ハブにアクセスするには、次のいずれかを実行します。

- DocNav で [Design Hub View] タブをクリックします。
- ザイリンクス ウェブサイトで[デザイン ハブ](#) ページを参照します。

注記: DocNav の詳細は、ザイリンクス ウェブサイトの [Documentation Navigator](#) ページを参照してください。DocNav からは、日本語版は参照できません。ウェブサイトのデザイン ハブ ページをご利用ください。

参考資料

このガイドの補足情報は、次の資料を参照してください。

1. [Vivado® Design Suite の資料](#)
2. 『UltraFast 設計手法クイック リファレンス ガイド』 ([UG1231](#))
3. 『UltraFast 設計手法タイミング クロージャクイック リファレンス ガイド』 ([UG1292](#))
4. 『UltraFast 設計手法チェックリスト』 ([XTP301](#))

Vivado Design Suite ユーザー ガイドおよびリファレンス ガイド

1. 『Xilinx Power Estimator ユーザー ガイド』 ([UG440](#))
2. 『UltraFast エンベデッド デザイン設計手法ガイド』 ([UG1046](#): [英語版](#)、[日本語版](#))
3. 『UltraFast Vivado HLS 設計手法ガイド』 ([UG1197](#): [英語版](#)、[日本語版](#))
4. 『Vivado Design Suite Tcl コマンド リファレンス ガイド』 ([UG835](#))
5. 『Vivado Design Suite ユーザー ガイド: デザイン フローの概要』 ([UG892](#))
6. 『Vivado Design Suite ユーザー ガイド: Vivado IDE の使用』 ([UG893](#))
7. 『Vivado Design Suite ユーザー ガイド: Tcl スクリプト機能の使用』 ([UG894](#))
8. 『Vivado Design Suite ユーザー ガイド: システム レベル デザイン入力』 ([UG895](#))
9. 『Vivado Design Suite ユーザー ガイド: IP を使用した設計』 ([UG896](#))
10. 『Vivado Design Suite ユーザー ガイド: エンベデッド プロセッサ ハードウェア デザイン』 ([UG898](#))
11. 『Vivado Design Suite ユーザー ガイド: I/O およびクロック プランニング』 ([UG899](#))
12. 『Vivado Design Suite ユーザー ガイド: ロジック シミュレーション』 ([UG900](#))
13. 『Vivado Design Suite ユーザー ガイド: 合成』 ([UG901](#))
14. 『Vivado Design Suite ユーザー ガイド: 高位合成』 ([UG902](#))
15. 『Vivado Design Suite ユーザー ガイド: 制約の使用』 ([UG903](#))
16. 『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 ([UG904](#))
17. 『Vivado Design Suite ユーザー ガイド: 階層デザイン』 ([UG905](#))
18. 『Vivado Design Suite ユーザー ガイド: デザイン解析およびクロージャ テクニック』 ([UG906](#))
19. 『Vivado Design Suite ユーザー ガイド: 消費電力解析および最適化』 ([UG907](#))
20. 『Vivado Design Suite ユーザー ガイド: プログラムおよびデバッグ』 ([UG908](#))
21. 『Vivado Design Suite ユーザー ガイド: Dynamic Function eXchange』 ([UG909](#))
22. 『Vivado Design Suite ユーザー ガイド: 入門』 ([UG910](#))
23. 『Vivado Design Suite プロパティ リファレンス ガイド』 ([UG912](#))
24. 『Vivado Design Suite ユーザー ガイド: リリース ノート、インストール、およびライセンス』 ([UG973](#))
25. 『Vivado Design Suite ユーザー ガイド: IP インテグレーターを使用した IP サブシステムの設計』 ([UG994](#))
26. 『Vivado Design Suite ユーザー ガイド: カスタム IP の作成とパッケージ』 ([UG1118](#))

27. a. 『Vivado Design Suite 7 シリーズ FPGA および Zynq-7000 SoC ライブラリ ガイド』 ([UG953](#))
b. 『UltraScale アーキテクチャ ライブラリ ガイド』 ([UG974](#))
28. 『UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP 製品ガイド』 (PG213: [英語版](#)、[日本語版](#))

Vivado Design Suite チュートリアル

1. 『Vivado Design Suite チュートリアル: 高位合成』 ([UG871](#))
2. 『Vivado Design Suite チュートリアル: デザイン フローの概要』 ([UG888](#))
3. 『Vivado Design Suite チュートリアル: ロジック シミュレーション』 ([UG937](#))
4. 『Vivado Design Suite チュートリアル: エンベデッド プロセッサ ハードウェア デザイン』 ([UG940](#))
5. 『Vivado Design Suite チュートリアル: Dynamic Function eXchange』 ([UG947](#))

その他のザイリンクス資料

1. a. 『7 シリーズ FPGA PCB デザイン ガイド』 (UG483: [英語版](#)、[日本語版](#))
b. 『UltraScale アーキテクチャ PCB デザイン ユーザー ガイド』 (UG583: [英語版](#)、[日本語版](#))
c. 『Zynq-7000 SoC PCB デザイン ガイド』 (UG933: [英語版](#)、[日本語版](#))
2. a. 『7 シリーズ FPGA SelectIO リソース ユーザー ガイド』 (UG471: [英語版](#)、[日本語版](#))
b. 『UltraScale アーキテクチャ SelectIO リソース ユーザー ガイド』 (UG571: [英語版](#)、[日本語版](#))
3. a. 『7 シリーズ FPGA クロッキング リソース ユーザー ガイド』 (UG472: [英語版](#)、[日本語版](#))
b. 『UltraScale アーキテクチャ クロッキング リソース ユーザー ガイド』 (UG572: [英語版](#)、[日本語版](#))
4. a. 『UltraScale アーキテクチャ GTH トランシーバー ユーザー ガイド』 (UG576: [英語版](#)、[日本語版](#))
b. 『UltraScale アーキテクチャ GTY トランシーバー ユーザー ガイド』 (UG578: [英語版](#)、[日本語版](#))
5. 『UltraScale Devices Gen3 Integrated Block for PCI Express LogiCORE IP 製品ガイド』 (PG156: [英語版](#)、[日本語版](#))
6. 『Virtual Input/Output LogiCORE IP 製品ガイド』 ([PG159](#))
7. 『In-System IBERT LogiCORE IP 製品ガイド』 ([PG246](#))
8. 『7 シリーズ FPGA メモリ リソース ユーザー ガイド』 (UG473: [英語版](#)、[日本語版](#))
9. 『7 シリーズ FPGA DSP48E1 スライス ユーザー ガイド』 (UG479: [英語版](#)、[日本語版](#))
10. 『UltraScale アーキテクチャ DSP スライス ユーザー ガイド』 (UG579: [英語版](#)、[日本語版](#))
11. 『7 シリーズ FPGA および Zynq-7000 SoC XADC デュアル 12 ビット 1MSPS アナログ-デジタル コンバーター ユーザー ガイド』 (UG480: [英語版](#)、[日本語版](#))
12. 『リファレンス システム: IP インテグレーターを用いた Kintex-7 MicroBlaze システム シミュレーション』 (XAPP1180: [英語版](#)、[英語版](#))
13. 『Zynq-7000 SoC および 7 シリーズ デバイス メモリ インターフェイス ソリューション ユーザー ガイド』 (UG586: [英語版](#)、[日本語版](#))
14. 『UltraScale アーキテクチャ FPGA メモリ IP LogiCORE IP 製品ガイド』 (PG150: [英語版](#)、[日本語版](#))
15. 『S パラメーター モデルを使用した FPGA パワー インテグリティのシミュレーション』 ([WP411](#))
16. 『逸脱温度を利用した熱ソリューションの拡張』 (WP517: [英語版](#)、[日本語版](#))
17. a. 『7 シリーズ回路図レビュー推奨事項』 ([XMP277](#))

- b. 『Kintex UltraScale および Virtex UltraScale FPGA 回路図レビュー チェックリスト』 (XTP344)
- c. 『UltraScale+ FPGA および Zynq UltraScale+ MPSoC 回路図レビュー チェックリスト』 (XTP427)
- 18. 『UltraScale FPGA の BPI コンフィギュレーションおよびフラッシュ プログラム』 (XAPP1220: [英語版](#)、[日本語版](#))
- 19. 『7 シリーズ FPGA の BPI 高速コンフィギュレーションおよび iMPACT フラッシュ プログラム』 (XAPP587: [英語版](#)、[日本語版](#))
- 20. 『SPI フラッシュを使用した 7 シリーズ FPGA のコンフィギュレーション』 (XAPP586: [英語版](#)、[日本語版](#))
- 21. 『UltraScale FPGA での SPI コンフィギュレーションおよびフラッシュ プログラミング』 (XAPP1233: [英語版](#)、[日本語版](#))
- 22. 『暗号化を使用して 7 シリーズ FPGA のビットストリームを保護』 (XAPP1239: [英語版](#)、[日本語版](#))
- 23. 『リッドレス フリップチップ パッケージの機械/熱設計ガイドライン』 (XAPP1301: [英語版](#)、[日本語版](#))
- 24. 『Vitis 統合ソフトウェア プラットフォームの資料』 (UG1416) のアプリケーション アクセラレーション開発フローの [Vitis HLS 資料](#)

トレーニング リソース

- 1. [トレーニング コース: UltraFast 設計手法](#)
- 2. [Vivado Design Suite QuickTake ビデオ: UltraFast Vivado 設計手法](#)
- 3. [Vivado Design Suite QuickTake ビデオ: Vivado デザイン フローの概要](#)
- 4. [Vivado Design Suite QuickTake ビデオ: Vivado IP インテグレーターを使用した Zynq デバイスの設計](#)
- 5. [Vivado Design Suite QuickTake ビデオ: Vivado Design Suite でパースシャル リコンフィギュレーションを実行](#)
- 6. [Vivado Design Suite QuickTake ビデオ: さまざまなタイプのプロジェクトの作成](#)
- 7. [Vivado Design Suite QuickTake ビデオ: プロジェクトでソース ファイルを管理](#)
- 8. [Vivado Design Suite QuickTake ビデオ: Vivado Design Suite でのリビジョン管理の使用](#)
- 9. [Vivado Design Suite QuickTake ビデオ: Vivado IP のバージョン アップグレードの管理](#)
- 10. [Vivado Design Suite QuickTake ビデオ: I/O プランニングの概要](#)
- 11. [Vivado Design Suite QuickTake ビデオ: Vivado での再利用可能な IP のコンフィギュレーションおよび管理](#)
- 12. [Vivado Design Suite QuickTake ビデオ: Vivado で write_bitstream コマンドを使用する方法](#)
- 13. [Vivado Design Suite QuickTake ビデオ: Vivado でのデザイン解析およびフロアプラン](#)
- 14. [Vivado Design Suite QuickTake ビデオ: UltraFast 設計手法チェックリストの説明](#)
- 15. [Vivado Design Suite ビデオ チュートリアル](#)

お読みください: 重要な法的通知

本通知に基づいて貴殿または貴社 (本通知の被通知者が個人の場合には「貴殿」、法人その他の団体の場合には「貴社」。以下同じ) に開示される情報 (以下「本情報」といいます) は、ザイリンクスの製品を選択および使用することのためにのみ提供されます。適用される法律が許容する最大限の範囲で、(1) 本情報は「現状有姿」、およびすべて受領者の責任で (with all faults) という状態で提供され、ザイリンクスは、本通知をもって、明示、黙示、法定を問わず (商品性、非侵害、特定目的適合性の保証を含みますがこれらに限られません)、すべての保証および条件を負わない (否認する) ものとし、また、(2) ザイリンクスは、本情報 (貴殿または貴社による本情報の使用を含む) に関係し、起因し、関連する、いかなる種類・性質の損失または損害についても、責任を負わない (契約上、不法行為上 (過失の場合を含む)、その他のいかなる責任の法理によるかを問わない) ものとし、当該損失または損害には、直接、間接、特別、付随的、結果的な損失または損害 (第三者が起こした行為の結果被った、データ、利益、業務上の信用の損失、その他あらゆる種類の損失や損害を含みます) が含まれるものとし、それは、たとえば当該損害や損失が合理的に予見可能であったり、ザイリンクスがそれらの可能性について助言を受けていた場合であったとしても同様です。ザイリンクスは、本情報に含まれるいかなる誤りも訂正する義務を負わず、本情報または製品仕様のアップデートを貴殿または貴社に知らせる義務も負いません。事前の書面による同意のない限り、貴殿または貴社は本情報を再生産、変更、頒布、または公に展示してはなりません。一定の製品は、ザイリンクスの限定的保証の諸条件に従うこととなるので、<https://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。IP コアは、ザイリンクスが貴殿または貴社に付与したライセンスに含まれる保証と補助的条件に従うことになります。ザイリンクスの製品は、フェイルセーフとして、または、フェイルセーフの動作を要求するアプリケーションに使用するために、設計されたり意図されたりしていません。そのような重大なアプリケーションにザイリンクスの製品を使用する場合のリスクと責任は、貴殿または貴社が単独で負うものです。<https://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。

自動車用のアプリケーションの免責条項

オートモーティブ製品 (製品番号に「XA」が含まれる) は、ISO 26262 自動車用機能安全規格に従った安全コンセプトまたは余剰性の機能 (「セーフティ 設計」) がない限り、エアバッグの展開における使用または車両の制御に影響するアプリケーション (「セーフティ アプリケーション」) における使用は保証されていません。顧客は、製品を組み込むすべてのシステムについて、その使用前または提供前に安全を目的として十分なテストを行うものとし、セーフティ設計なしにセーフティ アプリケーションで製品を使用するリスクはすべて顧客が負い、製品の責任の制限を規定する適用法令および規則にのみ従うものとし、また、

商標

© Copyright 2013-2021 Xilinx, Inc. Xilinx、Xilinx のロゴ、Alveo、Artix、Kintex、Spartan、Versal、Virtex、Vivado、Zynq、およびこの文書に含まれるその他の指定されたブランドは、米国およびその他の各国のザイリンクス社の商標です。AMBA、AMBA Designer、Arm、ARM1176JZ-S、CoreSight、Cortex、PrimeCell、Mali、および MPCore は、EU およびその他の各国の Arm Limited の商標です。OpenCL および OpenCL のロゴは Apple Inc. の商標であり、Khronos による許可を受けて使用されています。PCI、PCIe、および PCI Express は PCI-SIG の商標であり、ライセンスに基づいて使用されています。MATLAB および Simulink は、MathWorks, Inc. の登録商標です。

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、または各ページの右下にある [フィードバック送信] ボタンをクリックすると表示されるフォームからお知らせください。フィードバックは日本語で入力可能です。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。