

Vivado Design Suite ユーザー ガイド

階層デザイン

UG905 (v2015.3) 2015 年 9 月 30 日

本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。

日付	バージョン	改訂内容
2015年9月30日	2015.3	リリース番号をアップデート 「コンポーネント間の専用接続」に双方向ポートを回避するための推奨事項を追加 関連するチュートリアルへの参照リンクを削除。チュートリアルはザイリンクス サポートに連絡して入手可能になりました。
2015年4月1日	2015.1	リリース番号をアップデート 15 ページの「合成」で BUFFER_TYPE プロパティ名を IO_BUFFER_TYPE に変更。

目次

改訂履歴	2
Vivado 階層デザイン	
概要	4
設計に関する考慮事項	5
チェックポイント	8
アウト オブ コンテキスト コマンドおよび制約	8
最上位再利用コマンドおよび制約	15
Tcl スクリプト	17
既知の問題	17
付録 A: 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	18
ソリューション センター	18
参考資料	18
トレーニング リソース	18
お読みください: 重要な法的通知	19

Vivado 階層デザイン

概要

階層デザイン (HD) フローを使用すると、デザインを小さい管理可能なブロックにパーティションして個別に処理することができます。Vivado® Design Suite では、パーティションされたモジュールを残りのデザインから独立した箇所 (アウト オブ コンテキスト (OOC)) にインプリメントできます。次は、Vivado Design Suite での手法です。

- **モジュール解析**：このフローでは、モジュールを残りのデザインとは別に解析して、リソース使用量を決定し、タイミング解析を実行します。ラッパーまたはダミー ロジックは必要ありません。単にモジュールだけを合成、最適化、配置配線します。フルデザインの場合と同様に、リソース使用量解析を実行し、タイミング レポートを検証して、配置結果を確認します。

このフローでは、パーティションされたモジュールまたは IP コアがデザインの最上位から独立して (OOC で) インプリメントされます。モジュールは特定のパーツ/パッケージの組み合わせでデバイスの決まった位置にインプリメントされます。I/O バッファ、グローバル クロックおよびその他のチップ レベルのリソースは挿入されませんが、モジュール内にインスタンスすることはできます。この OOC インプリメンテーションの結果は、デザイン チェックポイント (DCP) ファイルとして保存できます。

- **モジュール再利用**：このフローでは、最上位デザイン内のモジュール解析フローからの配置配線されたモジュールを使用して、検証結果をロックします。デザインの特定のセクションを繰り返すことでタイミング クロージャーおよびその他の特定の目標を達成した後、結果をそのまま再利用できます。

この OOC モジュールを再利用する場合、モジュール ピンとインターフェイス ロジックが配置された箇所を把握しておかないと、接続ロジックが正しくフロアプランできません。インポートされた OOC モジュールの保持レベルは選択できるので、必要であれば配置配線を少し変更することができます。このフローでは、デバイスのほかのエリアや別のデバイスへの OOC インプリメンテーション結果の移動または複製はまだサポートされていません。

モジュール再利用フローには、2 つのパターンがあります。この 2 つの違いはモジュール制約を構築するメカニズムにあります。最上位デザインを 1 つまたは複数の再利用モジュールと問題なく組み合わせるには、コンテキスト制約 (フル デザインでモジュールをどのように接続するか定義) およびタイミング制約が重要となります。

モジュール再利用には、次のような方法があります。

- **ボトムアップ再利用**：この方法を使用すると、最上位デザインについて何も知らなくても OOC インプリメンテーションが実行され、再利用できるようになります。この OOC 結果を使用して、最上位インプリメンテーションが実行されます。配置配線を介して IP の一部のような検証済みのモジュールを構築でき、1 つまたは複数の最上位デザインで再利用できます。このフローの場合、最上位デザインの詳細はわからないので、コンテキスト制約はユーザーが指定する必要があります。これらは、モジュールの物理的な位置、モジュール I/O の配置の詳細、クロック ソースの定義、モジュールを出入りするパスのタイミング要件、および未使用 I/O に関する情報などを定義します。
- **トップダウン再利用**：この方法を使用すると、最上位デザインおよびフロアプランが使用され、OOC インプリメンテーション制約が作成されます。OOC インプリメンテーションは最上位デザインにより駆動されます。これにより、チーム デザインで、デザイン内の 1 つまたは複数モジュールの合成およびインプリメンテーションを並行して実行できます。チーム メンバーは、アセンブルされたデザインでその結果を再利用して、それぞれの担当部分をインプリメントできます。このフローの場合、最上位デザインの詳細 (ピン配置、フロアプラン、タイミング要件) は既知の状態、OOC インプリメンテーションをガイドするために使用さ

れます。これにより、OOC モジュールのピン制約、最上位の入力/出力タイミング要件、バウンダリ最適化制約すべてが最上位デザインから作成できるようになります。

これらどのフローを使用しても、デザイン全体ではなく、デザインに含まれるモジュールの 1 つのみをインプリメントすることで、インプリメンテーション実行時間を削減できます。これにより、より多くコンパイルできるので、設計時間を削減でき、モジュールベースでタイミングを検証して満たすことができます。また、残りのデザインが完成していなかったり、使用可能でない場合でも、モジュールの作業を進めることができます。

設計に関する考慮事項

階層デザイン手法で最適な結果にするためには、特別な考慮が必要です。次のセクションでは、Vivado 階層デザインフローのアーキテクチャ プランニング、設計、制約に関して考慮すべき点について説明します。

パフォーマンス目的のデザイン

デザインの残りの部分から独立した (OOC) モジュールをインプリメントすると、通常トップダウンフローで実行されるモジュール間の最適化は実行されません。こういった制限のためのパフォーマンス劣化を防ぐには、次のガイドラインに従ってください。

- インプリメントされる OOC モジュールを注意して選択します。デザインのほかのモジュールから論理的に独立し、デバイスの連続したエリアに物理的に制約が付けられたブロックを選択してください。
- 選択したモジュールを使用して効率的な階層を構築します。各インプリメンテーション用に階層の構造を作成します。デザイン階層は、重要な考慮事項です。デザインがパーティションされる箇所によっては、結果の質に多大な影響があることがあります。OOC インプリメンテーション用に適切なモジュールをグループ分けするために、階層を追加または変更する必要のあることもあります。
- モジュール内に完全に含まれるクリティカルパスは下位モジュールまたは最上位モジュールのいずれかに保持します。
- モジュール間の入力および出力にレジスタを付けて、モジュール内の最適化が最大限に実行されるようにし、最も柔軟な配置配線が実行できるようにします。
- コンテキスト制約を定義して、モジュールの使用方法に関する情報を提供します。コンテキスト制約では、最上位でモジュールをどのように接続されるかが定義されるので、さらに多くの最適化および正確なタイミング解析が実行できます。詳細は、「コマンドおよび制約」セクションの 10 ページの「[アウト オブ コンテキスト デザイン制約](#)」を参照してください。
- 専用接続は、常に OOC モジュールバウンダリをまたがって適切に処理されるわけではありません。関連するデザイン エレメントはすべて一緒にパーティションする必要があります。たとえば、トランシーバーまたは IOLOGIC コンポーネントなどの専用接続が付いた I/O コンポーネントがその例です。詳細は、「[コンポーネント間の専用接続](#)」を参照してください。

効率的なフロアプランの構築

OOC モジュールをインプリメントするには、次の要件に従う必要があります。

- 各モジュールのインプリメンテーションに Pblock 制約を含めて、配置を制御する必要があります。Pblock が使用されない場合、アセンブリ段階で配置が競合する可能性があります。
- CONTAIN_ROUTING プロパティをすべての OOC Pblock に追加します。このプロパティを付けないと、配線に競合が発生する可能性があるため、lock_design でインポートされたモジュールの配線をロックできなくなります。
- 各 OOC モジュールの Pblock 範囲は重複しないようにします。最上位デザインに複数の OOC モジュールをインポートする場合は、モジュールがそれぞれデバイスの別の領域を使用している必要があります。

- 入れ子状態になった Pblock (子 Pblock) の場合は、その Pblock の範囲が親 Pblock の範囲に完全に含まれていて、PARENT プロパティが正しく設定されている限り、OOC インプリメンテーションでサポートされます。詳細は、表 5 の「Pblock 制約」を参照してください。
- すべてのクロック バッファ (最上位と OOC モジュールの両方) はロックされる必要があります。OOC モジュールの中のバッファには LOC 制約を付ける必要があります。最上位のバッファの位置は HD.CLK_SRC 制約で識別される必要があります。HD.CLK_SRC 制約の詳細については、10 ページの「アウト オブ コンテキスト デザイン 制約」を参照してください。
- OOC インプリメンテーション結果が使用される場合は、HD.PARTPIN_RANGE または HD.PARTPIN_LOCS 制約を使用して OOC インプリメンテーション中に OOC モジュール ピンをロックしておく必要があります。

OOC インプリメンテーションでは、パーティション ピンに主に 2 つの役割があります。

- 1 つ目は、関連するインターフェイス ロジックの配置をガイドするための物理ロケーションを作成する役割です。配置に影響を与えるには、PartPin へ向かってまたは PartPin からのタイミング制約 (set_max_delay) が必要です。
- 2 つ目のパーティション ピンの役割は、配線ツールに対してインターフェイス サブネットを配線するポイントを示すことにあります。PartPin がいない場合、インターフェイス ネットは配線できません。作成した OOC 結果は、再利用中にインターフェイス ネットが必要とされる内部からモジュール ブロック配線リソースへ配線できます。これにより、配線不可能なデザインにならずに、OOC ネットすべてが保持されなくなります。

HD.PARTPIN_RANGE および HD.PARTPIN_LOCS 制約の詳細は表 6 の「コンテキスト制約」、set_max_delay 詳細については、表 4 の「タイミング制約」を参照してください。

コンポーネント間の専用接続

専用接続のあるコンポーネントをデザインと同じパーティションに保持することが推奨されたり、必要とされることがあります。OOC モジュールのバウンダリにまたがる専用接続があると、パフォーマンスが落ちたり、インプリメンテーションエラーが発生することがあります。次は、専用接続を含むコンポーネントのリストです。

- **IOLGIC および IOBUF** : ILOGIC または OLOGIC、IDDR、ODDR、ISERDES、および OSERDES に配置されたレジスタから IBUF、OBUF、IBUFDS、OBUFDS、IOBUF、および IOBUFDS を含む I/O コンポーネントへの接続が含まれます。
- **GT トランシーバー コンポーネント** : GTX および GTP トランシーバーおよびそれらの専用 I/O 接続です。
- 双方向ポートは、できるだけ使用しないようにしてください。双方向ポートは PP_LOCS を受信できないので、双方向ポートの PP_RANGE または PP_LOCS 制約は OOC モジュールで自動的に削除されます。双方向ポートが必要な場合は、関連するインターフェイス ロジックをフロアプランして、再利用フロー中に OOC モジュールがインポートされる際のタイミング問題を回避するようにしてください。

デザインの異なるパーティションに相互接続する I/O コンポーネントは配置しないようにしてください。

IDELAYCTRL グループの使用

OOC モジュール内の IDELAYCTRL グループの使用はサポートされています。OOC インプリメンテーションにより、IDELAYCTRL が挿入されて、その結果が Top にインポートされます。IDELAYCTRL グループを使用する場合は、次の規則が適用されます。

- 複数の OOC モジュール (それぞれに独自の IDELAYCTRL が含まれる場合) は、同じクロック領域を共有できません。
- IDELAYCTRL の付いた OOC モジュールは、100% 保持されません。これは、このバージョンの Vivado における既知の制限であり、今後の Vivado Design Suite リリースで修正される予定です。

I/O およびクロック バッファ

I/O およびクロック バッファは OOC モジュール内でサポートされますが、使用方法によっては特別な注意が必要なこともあります。

- I/O バッファ**：OOC ポートが最上位の I/O バッファに直接接続される場合は、このバッファを OOC モジュール内に移動した方が結果が改善することがあります。ただし、すべての状況においてこれが実行できるわけではありません (たとえば、OOC ポートが最上位の IBUF に直接接続されているのに、IBUF が OOC モジュール以外のその他のロジックも駆動する場合)。このような場合、OOC モジュール内のロジックは HD.PARTPIN_LOCS 制約で制御する必要があります。詳細は、「[アウト オブ コンテキスト コマンドおよび制約](#)」を参照してください。
- リージョナルクロック バッファ**：BUFR または BUFHCE が OOC モジュール内にある場合、特定の位置にロックする必要があります。これにより、バッファで駆動されるロジックが適切に配置されるようになりますが、BUFR または BUFHCE が最上位デザインに含まれる場合、OOC Pblock がバッファへのアクセスよりも多くのクロック領域にまたがるので、さらに多くの情報を提供する必要があります。入れ子になった Pblock は、OOC Pblock で定義された範囲のサブセットである範囲で作成する必要があります。このような Pblock には、BUFR または BUFHCE で駆動されるすべてのセルが含まれます。この Pblock は次のコマンドで作成できます。

```
create_pblock -parent <parent_pblock_name> <nested_pblock_name>
add_cells_to_pblock <nested_pblock_name> -cells [get_cells -of [get_nets
-segments -of [get_ports [list <clock_port> <clock_port>]]] -filter
"(IS_PRIMITIVE)"]
resize_pblock <nested_pblock_name> -add {SLICE_Xx1Yy1:SLICE_Xx2Yy2}
```

これは、最上位の BUFR または BUFHCE で駆動される各モジュール ポートに対して実行する必要があります。複数の OOC クロック ポートに同じクロック領域のロードが含まれる場合、すべての適用可能なポートは上記の add_cells_to_pblock コマンドでリストできます。入れ子になった Pblock の範囲は、最上位インプリメンテーションの BUFR または BUFHCE 位置に対応する必要があります。最上位のバッファ位置と対応する OOC モジュールの Pblock 範囲間が一致しないと、最上位インプリメンテーション中に配線不可能な状態になることがあります。

- グローバル クロック バッファ**：グローバル バッファは OOC モジュール内でサポートされます。BUFR が OOC インスタンス内にあると、クロック ネットは OOC インプリメンテーションでグローバル配線に配線されます。OOC ポートが最上位のクロック ネットで駆動される場合、クロック ネットは OOC インプリメンテーション中には配線されません。クロック遅延/スキューはタイミング概算により判断されます。この場合、HD.CLK_SRC 制約を使用するとタイミング概算が改善できます。この制約により、ドライバーの位置およびタイプがツールに伝わり、CPR (Clock Pessimism Removal) が計算されてタイミング概算が改善します。CPR 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) [\[参照 1\]](#) を参照してください。

アウト オブ コンテキスト デザインのデザイン ルール

表 1 は、OOC デザインをインプリメントする際に実行されるデザイン ルール チェック (DRC) とその DRC をテストするためのデザイン ルールについて示しています。

表 1: 階層デザインのデザイン ルール

DRC 番号	重要度	デザイン ルール
HDOOC-1	エラー	リコンフィギャブル モジュールには、定義済みの Pblock が含まれる必要があります。
HDOOC-2	エラー	HD モジュールでは、一部の Pblock プロパティが定義されている必要があります。
HDOOC-3	エラー	OOC モジュールの場合、ビットストリームは生成できません。
HDOOC-4	エラー	セルに対する Pblock 範囲または LOC がありません。

チェックポイント

階層デザインフローでは、モジュールインプリメンテーションの結果をエクスポートおよびインポートするためにチェックポイントが使用されます。チェックポイントは、論理デザイン、物理デザイン、モジュール制約をアーカイブしたもので、デザインを完全に復元するのに必要な唯一のファイルです。

保存されたチェックポイントは、元々生成されていたのと同じパーツ/パッケージ/スピードグレードの組み合わせにのみ読み込むことができます。



推奨: 階層デザインフローで読み込まれるすべてのデータがモジュールインターフェイスで完全に一致するようにするには、`read_checkpoint` コマンドに `-strict` オプションを使用することをお勧めします。チェックポイントの詳細は、『Vivado Design Suite ユーザーガイド：合成』(UG901) [\[参照 2\]](#) の「[デザインチェックポイントを使用したデザインのスナップショットの保存および復元](#)」を参照してください。

アウトオブコンテキストコマンドおよび制約

階層デザインフローは、現在のところ、非プロジェクトモードのバッチ/Tcl インターフェイス (Vivado IDE (GUI) またはプロジェクトベースのコマンドなし) でのみサポートされています。サンプルデザインおよび関連スクリプトもご要望に応じて用意しております。詳細は、ザイリンクスサポートにお問い合わせください。

次のセクションでは、階層デザインフローで使用される特定のアウトオブコンテキストコマンドおよび制約について説明します。階層デザインフローを実行するためのこれらのコマンドの使用例を示します。各コマンドの詳細は、『Vivado Design Suite Tcl コマンドリファレンスガイド』(UG835) [\[参照 3\]](#) を参照してください。

アウトオブコンテキストコマンド

残りのデザインから独立したアウトオブコンテキスト (OOC) モジュールを合成またはインプリメントするには、ツールがアウトオブコンテキストモードで実行される必要があります。これ以外の場合、アウトオブコンテキストフローを実行するために使用されるコマンドはほかのフローと同じになります。現在のところ、合成、最適化、またはインプリメンテーションでサポートされないコマンドはありません。

合成

このフローでは、複数の合成ツールおよび手法がサポートされます。次は、サポートされるツールのリストです。

- [XST]: パーティション (PMXL ファイル) を使用したボトムアップ合成またはインクリメンタル合成。これは 7 シリーズのみでサポートされています。
注記: XST は新しい Vivado デザインや、7 シリーズよりも新しいアーキテクチャをターゲットにしたデザインには推奨されません。
- [Synplify]: ボトムアップ合成またはコンパイルポイント (階層プロジェクトを使用して各ネットリストを生成)
- [Vivado synthesis]: アウトオブコンテキスト (OOC) 合成のみ。



重要: OOC 合成 (ボトムアップ合成) は、各モジュールにそれぞれ合成 run がある合成フローです。通常は、下位モジュールの自動 I/O バッファ挿入をオフにします。

この文書では、Vivado 合成フローのみについて説明します。Synplify フローに関する詳細は、Synopsys 社の Synplify の資料を参照してください。

このフローの Vivado 合成は、`synth_design` コマンドを使用してバッチ モードで実行されます。

```
synth_design -mode out_of_context -flatten_hierarchy rebuilt -top <top_module_name>
-part <part>
```

表 2: `synth_design` オプション

コマンド オプション	説明
<code>-mode out_of_context</code>	合成およびダウンストリーム ツールの I/O 挿入がオフになります。このモードは、 <code>write_checkpoint</code> が使用されるとチェックポイントに保存されます。
<code>-flatten_hierarchy rebuilt</code>	<code>-flatten_hierarchy</code> に使用できる値は複数ありますが、階層デザイン フローの設定には <code>rebuilt</code> が推奨されます。
<code>-top</code>	合成されるモジュールのモジュール/エンティティ名を指定します。 <code>synth_design</code> より前に <code>set_property top <top_module_name> [current_fileset]</code> が実行されている場合、このオプションを使用する必要はありません。
<code>-part</code>	ターゲットにするザイリンクス パーツ (例: <code>xc7k325tffg900-3</code>) を指定します。

`synth_design` コマンドは、デザインを合成して、その結果をメモリに格納します。結果をファイルに書き出すには、`write_checkpoint` コマンドを実行する必要があります。

```
write_checkpoint <file_name>.dcp
```

上記のコマンドを実行すると、合成結果が DCP ファイルに保存され、インメモリ デザインを閉じることができます。`open_checkpoint` を使用すると、合成結果は後で読み出すことができます。これにより、合成をそのたびに実行し直さなくても、合成結果に対して 1 つまたは複数のインプリメンテーション `run` を実行できます。

タイミング制約および物理制約は、OOC 合成に渡されます。物理制約は、合成では無視され、最終的な DCP に渡されます。タイミング制約の場合、モジュール レベル コンテキスト制約はすべて正しく `USED_IN out_of_context` とマークされていることが重要です。制約が OOC にのみ使用されることを指定する方法については、[10 ページの「OOC 専用制約」](#)を参照してください。

インプリメンテーション

このセクションでは、OOC フローでモジュール インスタンスをインプリメントするのに必要なコマンドについて説明します。最上位デザインにインスタンス化されたインスタンスが複数このモジュールに含まれ、アセンブリ フローが使用される場合、必要なインプリメンテーション結果を生成するために、それぞれ独自の Pblock 制約の付いた OOC インプリメンテーションが複数必要になります。

`synth_design -mode out_of_context` が以前に実行され、結果がまだメモリに含まれる場合は、インプリメンテーションを直接実行できます。たとえば、次のインプリメンテーション コマンドを使用できます。

- `read_xdc`: すべての制約がまだ読み込まれていない場合に使用します。一部のモジュール レベルの XDC 制約を OOC インプリメンテーションにのみ適用する必要のあることもあります。OOC のみの制約の詳細については、[10 ページの「OOC 専用制約」](#)を参照してください。
- `set_property HD.PARTITION 1 [current_design]`: セルをインプリメント済み OOC として識別し、再利用フロー中に DRC と必要なソフトウェア機能を有効にします。OOC インプリメンテーションに必須なわけではありませんが、再利用フロー中に最終的な DRC が正しくインポートされるように設定されているかどうかを確認する良い方法です。HD.PARTITION の再利用フローでの役割については、[15 ページの「最上位再利用コマンド」](#)を参照してください。
- `opt_design` (オプションですが、推奨)
- `place_design`
- `phys_opt_design` (オプションですが、推奨)
- `route_design`

メモリにデザインが読み込まれていない場合は、デザインを読み込む必要があります。これは、次の方法のいずれかで実行できます。

- 方法 1: ネットリスト デザインの読み込み

```
read_edif <file_name>.edif/edn/ngc
link_design -mode out_of_context -top <top_module_name> -part <part>
```

表 3: link_design オプション

コマンド オプション	説明
-mode out_of_context	ネットリスト デザインを OOC モードで読み込みます。ダウンストリーム ツールの特定チェックおよび最適化をオンにします。
-part	ターゲットにするザイリンクス パーツ (例: xc7k325tffg900-3) を指定します。
-top	インプリメントされるモジュールのモジュール/エンティティ名を指定します。link_design の前に set_property -top <top_module_name> [current_fileset] が実行されている場合、このオプションを使用する必要はありません。

デザイン ネットリストを読み込んだ後に link_design に -mode out_of_context オプションを付けない場合、その後のインプリメンテーション段階でデザインが完全デザインとして処理され、ソースのない信号やロードのない信号がすべて削除されます。synth_design または link_design の実行中には、OOC モジュールを定義して、モジュール解析フローを実行する必要があります。

- 方法 2: チェックポイントを開く

```
open_checkpoint <file_name>.dcp
```



重要: open_checkpoint コマンドには、-mode out_of_context オプションは使用できません。このモジュールはチェックポイントの一部として保存されるので、チェックポイントを書き出す際にはツールが正しいモードであるかどうかを必ず確認してください。

- 方法 3: さまざまなファイル タイプを追加

add_files コマンドを link_design と一緒に使用すると、複数のファイルおよびさまざまなファイル タイプを含むモジュールに読み込むことができます。

```
add_files <file_name>.dcp
add_files <file_name>.edf
add_files <file_name>.xdc
link_design -mode out_of_context -top <top_module_name> -part <part>
```

アウト オブ コンテキスト デザイン制約

モジュール解析フローを使用したデザインの場合、次の制約のいずれも絶対に必要というわけではありません。より正確なタイミング解析には、HD.CLK_SRC および create_clock を使用してください。その他すべての制約はオプションです。

モジュール再利用フローを使用する場合は、これらのコンテキスト制約が重要になってきます。OOC モジュールを含むデザインを問題なくアセンブルするには、これらの制約を使用して、物理リソースが適切に分配され、クロック関係が理解され、モジュール インターフェイスに関する情報が正確に設定されるようにします。各モジュールの制約を設定しておかないと、アセンブリがより困難になります。

OOC 専用制約

OOC インプリメンテーションに必要な制約の中には、最上位デザインにインポートされると問題となるものがあります。このような状況を避けるため、別の XDC ファイルでこれらの制約を指定して、OOC 使用目的にのみ設定する

必要があります。XDC ファイルを OOC フローにのみ使用されるように指定する方法は、2 つあります。特定の XDC ファイルの制約が OOC フローでのみ使用されるように指定すると、それらにマーカーが追加され、OOC デザイン以外に読み込まれる際に制約が無視されるようになります。

- **方法 1:** read_xdc の使用

read_xdc コマンドで XDC ファイルを読み込む際に -mode out_of_context オプションを使用します。

```
read_xdc -mode out_of_context <file>.xdc
```



ヒント: read_xdc コマンドはデザインを link_design で読み込む前またはあとに実行できます。

- **方法 2:** USED_IN プロパティ

ファイルが add_files コマンドで追加される際に、ファイルにプロパティを設定して OOC でのみ使用されるように指定します。XDC ファイルが使用されるフローすべて (合成やインプリメンテーションなど) を指定する必要があります。

```
add_files <file>.xdc
set_property USED_IN {synthesis implementation out_of_context} [get_files <file>]
```



重要: add_files コマンドは link_design でデザインを読み込むよりも前に実行する必要があります。既に読み込まれたデザインに対して add_files を使用しても、ファイルを追加はできません。

制約の構文

次の表は、アウト オブ コンテキスト インプリメンテーションで使用すべきタイミング制約、配置制約、コンテキスト制約をリストしています。これらの制約の多くは、どのデザインフローでも使用できます。詳細は、『Vivado Design Suite ユーザー ガイド: 制約の使用』(UG903) [参照 4] を参照してください。



重要: トップダウン再利用フローを使用すると、このセクションの制約はすべて自動的に生成されます。これらの制約を生成するスクリプトおよび方法については、『Vivado Design Suite チュートリアル: 階層デザイン』(UG946) を参照してください。詳細は、ザイリンクス サポートにお問い合わせください。

表 4: タイミング制約

制約名	説明
set_max_delay	入力および出力遅延を定義して、OOC モジュールで許容される時間を概算します。OOC インプリメンテーションの配置を制御し、最上位のタイミングクロージャを達成しやすくします。
create_clock	OOC モジュール ポートのクロックを定義します。create_clock 制約は、クロック バッファが最上位にインスタンス化されていても、OOC モジュールにインスタンス化されていても、各クロック ポートごとに必要です。
set_clock_uncertainty	OOC モジュールへの入力になるクロックのばらつきを定義します。OC モジュールのすべてのクロックに対して定義して、タイミング解析が正確になるようにします。定義しないと、モジュールがインポートされたときに、パスのタイミングが満たされなくなることがあります。
set_system_jitter	システム ジッター値を定義します。最上位デザインに基づいてユーザー クロックのばらつき (set_clock_uncertainty) を定義する場合は 0 に設定する必要があります。0 に設定しておかないと、システム ジッターが OOC インプリメンテーションのばらつき計算の要素に入れられ、最終値がユーザーの定義した値とは異なってしまいます。

表 4: タイミング制約

制約名	説明
set_clock_latency	OOC モジュールへの入力になるクロックのレイテンシを定義します。この制約は、クロック パス全体が既知ではない場合に、クロック遅延を正しくモデリングするために必要です。
set_clock_groups	非同期クロック (-asynchronous) または同じグローバル バッファで駆動されるクロック (-physically_exclusive) を定義します。

タイミング制約の例

- `create_clock -period 8.000 -name clk -waveform {0.000 4.000} [get_ports clk]`
- `set_max_delay -from [get_ports <ports>] -to [get_pins <synchronous pin>] -datapath_only <delay>`
- `set_max_delay -from [get_pins <synchronous pin>] -to [get_ports <ports>] -datapath_only <delay>`
- `set_system_jitter 0.0`
- `set_clock_latency -source -min 0.10`
- `set_clock_latency -source -max 0.20`
- `set_clock_groups -physically_exclusive -group [clk1] -group [clk2]`
- `set_clock_groups -asynchronous -group [clk1] -group [clk2]`

これらのタイミング制約の範囲は、その OOC モジュール自体に適用されます。OOC インプリメンテーションには、インスタンスへのタイミング、インスタンスからのタイミング、インスタンス内部のタイミングなどすべてが含まれます。これには、フォルス パスおよびマルチサイクルパスなどの特殊なパスも含まれます。

表 5: Pblock 制約

コマンド/プロパティ名	説明
create_pblock	各 OOC インスタンスの最初の Pblock を作成するコマンドです。
resize_pblock	Pblock のサイト タイプ (SLICE、RAMB36 など) とサイト位置を定義します。
add_cells_to_pblock	Pblock に含まれるインスタンスを指定します。通常、個別インスタンスに対する階層レベルを指定します。OOC インプリメンテーションの場合は、セル名を指定する代わりに、-top を使用して、OOC モジュールの下のセルすべてを指定します。
CONTAIN_ROUTING	Pblock に含まれない配線リソースが使用されないようにする Pblock プロパティです。デフォルトの値は FALSE です。Pblock 範囲に完全に含まれるパスのみが含まれます (例: BUFGMUX の範囲がない場合、BUFGMUX を出入りするパスは含まれません。これは BUFGMUX のような多くのコンポーネントが必要とされるビヘイビアです)。
EXCLUDE_PLACEMENT	定義された Pblock 範囲内で Pblock に含まれないロジックが配置されないようにするため使用する Pblock プロパティです。デフォルトは false です。このプロパティは OOC インプリメンテーションには影響しませんが、アセンブリ中の最上位ロジックの配置に影響します。アセンブリ中に最適な結果となるので、値を false のままにしておくことをお勧めします。
PARENT	Pblock 階層を識別する Pblock プロパティで、値は子 Pblock が完全に含まれる親 Pblock の名前になります。OOC インプリメンテーションを含むネストされた Pblock はすべて PARENT キーワードを使用してモジュール再利用中の動作を正しくしておく必要があります。

Pblock コマンド および プロパティ の例

- `create_pblock <pblock_name>`
- `add_cells_to_pblock [get_pblocks <pblock_name>] -top`
- `resize_pblock [get_pblocks <pblock_name>] -add {SLICE_X0Y0:SLICE_X100Y100}`
- `resize_pblock [get_pblocks <pblock_name>] -add {RAMB18_X0Y0:RAMB18_X2Y20}`
- `set_property CONTAIN_ROUTING true [get_pblocks <pblock_name>]`

Pblock に追加されるセルが `-top` を使用して指定されていることに注意してください。これは、OOC インプリメンテーションでは、OOC インスタンスが最上位になり、OOC インスタンス全体が Pblock に含まれる必要があるからです。`-top` を使用することで、OOC モジュールが最上位デザインにインポートされる際に、Pblock が正しい階層レベルに適切に変換されるようになります。

OOC モジュール内のロジックをフロアプランする場合は、入れ子になった Pblock を使用できます。子 Pblock は完全に親 Pblock 内に含まれる必要があります。Pblock 間の親子関係は、次に示すように PARENT プロパティを使用して宣言します。

```
set_property PARENT <parent_pblock_name> [get_pblocks <child_pblock_name>]
```

Pblock プロパティの PARENT は別の Pblock を参照するので、制約が処理される順序が重要です。親 Pblock (`-top` を使用) は、それを参照する子 Pblock よりも前に定義する必要があります。

前述のタイミング制約および物理制約だけでなく、OOC インプリメンテーションのコンテキストを定義する制約もあります。コンテキスト制約では、OOC インプリメンテーションがインポートされる最上位の環境を定義します。

表 6: コンテキスト制約

コマンド/プロパティ名	説明
HD.CLK_SRC	OOC インプリメンテーションで、クロック バッファが OOC モジュール外で使用される場合に、それをインプリメンテーション ツールに伝えるため使用します。値は、クロック バッファ インスタンスの位置になります。これはクロック ポートに適用され、ポートにはこの制約が適用されるよりも前に定義済みクロック (<code>create_clock</code>) が含まれる必要があります。
HD.PARTPIN_LOCS	配線される指定ポートにインターコネクト タイル (INT) を指定します。HD.PARTPIN_RANGE よりも優先されます。内部 OOC ロジックの配置配線に影響します。 クロック ポートに使用すると、クロックにローカル配線が使用されるので、クロック ポートには使用しないでください。 専用接続には使用しないでください。
HD.PARTPIN_RANGE	指定したピン/ポートを配線するために使用可能なコンポーネント サイト (SLICE、DSP、BRAM) またはインターコネクト タイル (INT) の範囲を定義します。 この制約は専用接続を持たないピンまたはポートに対してのみ有効です (例: クロックまたは最上位 I/O パッドへの直接接続など)。これらのピンまたはポートに適用される場合、制約は無視されます。
set_logic_unconnected	最上位で未接続のままになる指定した出力ポートに対して、追加で最適化が実行できるようになります。
set_logic_one	最上位の VCC で駆動される指定した入力ポートに対して、追加で最適化が実行できるようになります。
set_logic_zero	最上位で GND で駆動される指定した入力ポートに対して、追加で最適化が実行できるようになります。



重要: `set_logic` バウンダリ最適化制約を間違っ指定すると、不正なビヘイビアおよびツールエラーが発生する可能性があります。たとえば、出力ポートを OCC モジュールで未接続として定義したのに、実際にはそれが最上位で使用される場合、次のようなエラー メッセージが表示されます。

```
ERROR:[Opt 31-67] Problem:A LUT2 cell in the design is missing a connection on input pin I0, which is used by the LUT equation.
```

コンテキスト制約の例

- `set_property HD.CLK_SRC BUFGCTRL_X0Y16 [get_ports <port_name>]`
- `set_property HD.PARTPIN_LOCS INT_R_X0Y0 [get_ports <port_name>]`
- `set_property HD.PARTPIN_RANGE SLICE_X0Y1:SLICE_X1Y3 [get_ports <port_name>]`
- `set_logic_unconnected [get_ports <port_name>]`
- `set_logic_one [get_ports <port_name>]`
- `set_logic_zero [get_ports <port_name>]`

モジュール解析フローの場合、デフォルトではインターフェイス ネット (OCC モジュール ポートに接続されるモジュール内のネット) は配線されません。これらのインターフェイス ネットが配線されるようにするには、`HD.PARTPIN` 制約を使用してモジュールをロックする必要があります。配置配線モジュールポート (またはパーティションピン) の配置を素早く取得するには、`HD.PARTPIN_RANGE` に OCC モジュールの Pblock SLICE 範囲の値を付けます。これらのピンのさらに詳細な配置を取得するには、さらに厳密な `HD.PARTPIN_RANGE` 値を使用するか、明示的な `HD.PARTPIN_LOCS` 値を指定します。最適なサイトや範囲を決定するには、Vivado IDE で [Device] ビューを開いて、次のボタンをクリックして配線リソースをオンにします。



拡大すると、[図 1](#) のように INT の位置が表示されます (この図では見やすくするために配線リソースを非表示にしています)。



図 1: INT タイルの位置

最上位再利用コマンドおよび制約

次のセクションでは、アウト オブ コンテキスト インプリメンテーションをインポートする際に最上位で使用されるコマンドおよび制約について説明します。サンプル デザインおよび関連スクリプトもご要望に応じて用意しております。詳細は、ザイリンクス サポートにお問い合わせください。

各コマンドの詳細は、『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835) [参照 3] を参照してください。

最上位再利用コマンド

アセンブリでは、前にインプリメントしたモジュールをモジュール解析フローから読み込む必要があります。パーティション分割された各インスタンスのチェックポイントは、必ず存在している必要があります。また、それぞれのチェックポイントは最上位デザインに読み込まれ、OOC モジュールごとにブラック ボックスが含まれます。OOC インプリメンテーション結果はブラック ボックスではないセルには読み込むことができません。これで標準的なインプリメンテーションコマンドを使用して、まだ配置配線されていないデザイン部分 (Top) をインプリメントできるようになります。

合成

パーティション分割されたインスタンスそれぞれに対するブラック ボックスを含む最上位ネットリストが必ず必要です。これには、最上位合成に、パーティション分割済みのインスタンスのモジュール/エンティティ宣言が含まれ、ロジックは含まれないようにする必要があります。

最上位合成は、通常すべての最上位ポートで I/O バッファを推論しますが、I/O バッファが OOC モジュールに特にインスタンス化される場合は、ポートごとに最上位合成で I/O バッファ挿入をオフにする必要があります。Vivado 合成でこれを実行する属性は `IO_BUFFER_TYPE = "none"` です。IO_BUFFER_TYPE およびその他の合成属性の詳細は、『Vivado Design Suite ユーザー ガイド : 合成』(UG901) [参照 5] の「[合成属性](#)」を参照してください。

インプリメンテーション

最上位インプリメンテーションは、標準デザインと同じように実行できますが、次をさらに実行する必要があります。

1. OOC インプリメンテーション結果で回避されるようにするには、ブラック ボックス セルそれぞれに `HD.PARTITION` プロパティを設定します。

注記: `HD.PARTITION` プロパティは、OOC インプリメンテーションで設定されていれば、OOC モジュールと共にインポートされますが、最上位ブラック ボックス インスタンスに設定すると、セルが正しくマークされているかどうか確認しやすくなります。

2. パーティションされたインスタンスごとに OOC チェックポイントを読み込みます。
3. 保持レベル (logical、placement、routing) を選択します。

セルをパーティションとしてマーク

セルをパーティションとしてマークするには、`HD.PARTITION` セルプロパティが必要です。`HD.PARTITION` を設定すると、次が実行されます。

- 特定のセルに `DONT_TOUCH` を設定すると、HD バウンダリ中または HD バウンダリ上で不正な最適化がされないようになります。
- HD 特有の DRC がトリガーされます。
- `read_checkpoint -cell` で特定のコードをイネーブルにし、`PartPin` を一掃して余分な Pblock を削除します。
`set_property HD.PARTITION 1 [get_cells <cell_name>]`

OOO チェックポイントの読み込み

OOO チェックポイントは、`read_checkpoint` コマンドに `-cell` オプションを付けて読み込みます。最上位デザインは既に開いており、パーティション分割されたインスタンスにそれぞれブラックボックスが含まれる必要があります。

```
read_checkpoint -cell <cell_name> <file> [-strict]
```

表 7: `read_checkpoint` オプション

オプション名	説明
<code>-cell</code>	OOO モジュールの完全な階層名を指定するために使用します。
<code>-strict</code>	セルを置き換えるのにポートが完全に一致している必要があり、パーツ、パッケージ、スピード グレード値が同一であることを確認します。
<code><file></code>	読み込む OOO チェックポイントを指定します。

OOO モジュール インプリメンテーションの結果をインポートしてロックする際、インターフェイス ネットは保持されません。また、最初のトップダウンまたは OOO インプリメンテーション中に生成された PartPin ロケーションはすべて理想的なロケーションではない可能性があり、配線ツールで不必要な制限が課せられることがあります。このため、`read_checkpoint -cell` コマンドでは自動的にインターフェイス ネットの配線が解除され、セルのチェックポイントからの PartPin がすべて削除されます。

保持レベルの設定

OOO チェックポイントを読み込んだら、このモジュールの保持レベルを定義する必要があります。

インポートした OOO チェックポイントの配置および配線をロックするには、`lock_design` コマンドを使用します。

```
lock_design [-level <value>] [-unlock] [<cell>]
```

このコマンドを使用する場合、`-level` オプションで次の値を指定すると、保持レベルが指定できます。

- logical**: 論理デザインを保持します。配置または配線情報も使用されますが、ツールで結果を改善できる可能性がある場合は変更可能です。
- placement** (デフォルト): 論理および配置デザインが保持されます。配置情報も使用されますが、ツールで結果を改善できる可能性がある場合は変更可能です。
- routing**: 論理、配置、および配線デザインが保持されます。内部配線は保持されますが、インターフェイス ネットは保持されません。配線を保持するには、OOO インプリメンテーション中に `CONTAIN_ROUTING` プロパティが `Pblock` に使用される必要があります。これにより、OOO インプリメンテーションが再利用される際に配線が競合することはありません。

必要な保持レベルに関係なく、物理データベース全体は配線も含めてまだ読み込まれますが、ツールで結果が改善できると判断されない限りは、変更されません。

表 8: `lock_design` の引数

引数名	説明
<code>-level</code>	保持レベルを指定します。使用できる値は、 <code>logical</code> 、 <code>placement</code> 、または <code>routing</code> で、デフォルト値は <code>placement</code> です。
<code>-unlock</code>	セルのロックを解除します。セルを指定しない場合は、デザイン全体の固定が解除されます。ロック ビヘイビアと正反対の動作になるロック解除には、 <code>-level</code> パラメーターを指定する必要があります。 <code>routing</code> を指定すると、配線のロックを解除するのみですが、 <code>placement</code> を指定すると配置配線のロックが解除されます。
<code><cell></code>	ロックされる階層セル名です。セルが指定されない場合は、全デザインがロックされます。

最上位再利用制約

最上位デザインで OOC モジュールを再利用する場合は、すべての標準的なデザイン制約を適用できます。OOC インプリメンテーションに使用される OOC 制約は、チェックポイントに保存され、デザインに適用可能なときに適用されます。

Tcl スクリプト

このフローの Tcl スクリプトは、ご要望に応じて用意しております。詳細は、ザイリンクス サポートにご連絡ください。

既知の問題

現在のリリースの階層デザイン フローに関する既知の問題について説明します。

UltraScale サポート

UltraScale デバイスは、現時点ではデザイン解析しかサポートされていません。再利用フローは、このリリースではベータ版になります。既知の制限などはありませんが、フローはまだ完全には保証されていません。

グローバル クロック配線の制限

最上位のバッファで駆動されるクロックは、OOC インプリメンテーション中は配線されません。配線の概算が使用されます。HD.CLK_SRC を使用すると配線の概算は改善されます。OOC モジュール内のクロック バッファは OOC インプリメンテーション中に配線されます。OOC クロックに最適な制約を付けると、このタイミング概算にタイミングの正確な結果が表示されるようになります。

IDELAYCTRL の付いた OOC モジュールの制限

IDELAYCTRL の付いた OOC モジュールはインポートはできますが、ロックできませんので、OOC モジュールは 100% 保持されません。

非プロジェクト モードのサポート

Vivado Design Suite の階層デザインは、現時点ではプロジェクト モードではサポートされません。

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、次の[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。トピックには、デザインアシスタント、アドバイザー、トラブルシューティング ヒントなどが含まれます。

参考資料

1. 『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)
 2. 『Vivado Design Suite ユーザー ガイド : インプリメンテーション』(UG904)
 3. 『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835)
 4. 『Vivado Design Suite ユーザー ガイド : 制約の使用』(UG903)
 5. 『Vivado Design Suite ユーザー ガイド : 合成』(UG901)
 6. [Vivado Design Suite の資料](#)
-

トレーニング リソース

ザイリンクスでは、本書に含まれるコンセプトを説明するさまざまなトレーニング コースおよびオンライン ビデオを提供しています。次のリンクから関連するトレーニング リソースを参照してください。

1. [Vivado Design Suite ビデオ チュートリアル](#)
2. [Vivado でのアドバンスド FPGA 設計トレーニング コース](#)

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、<http://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となります。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。<http://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。

© Copyright 2015 Xilinx, Inc. Xilinx, Xilinx のロゴ、Artix、ISE、Kintex、Spartan、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。すべてのその他の商標は、それぞれの保有者に帰属します。

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。

