Vivado Design Suite ユーザー ガイド

階層デザイン : デザインの再利用

UG905 (2012.3) 2012 年 10 月 16 日

XILINX®

本資料は英語版 (v2012.3) を翻訳したもので、内容に相違が生じる場合には原文を優先します。
資料によっては英語版の更新に対応していないものがあります。
日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

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

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	改訂内容
2012 年 10 月 16 日	2012.3	初版

目次

デザインの再利用

概要

階層デザイン (HD) 手法を使用すると、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分割することができます。Vivado Design Suite では、次のフローで残りのデザインとは関連のない (out-of context (OOC)) ブロックを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 インプリメンテーション** : このフローでは、デザインの最上位レベルとは関係なくパーティション分割されたブロックまたは IP コア (OOC) が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。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) ファイルに保存できます。このフローでは、次が実行できます。
  - デザインの残りの部分とは別にブロックをインプリメントし、フットプリントおよびタイミング解析を実行します。I/O バッファーは挿入されませんが、モジュール内にインスタンシエートできます。
  - モジュール　インプリメンテーションのチェックポイントを作成し、結果が完全なデザイン　インプリメンテーションに読み込まれるようにします。結果を再利用するには、モジュール ピンおよびインターフェイス ロジックの配置を制御する制約を使用して、完全なデザインに読み込まれたときに高品質な結果になるようにする必要があります。
- **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 インプリメンテーション フローから作成されたデザイン チェックポイントを使用します。この OOC モジュールを使用する場合、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を選択することはできますが、配置配線は少ししか変更できません。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。
- **トップ ダウン再利用** : このフローは、1 つの相違点を除いて、モジュール インプリメンテーション フローとモジュール再利用フローを統合したフローに類似しています。このフローでは、最上位フロアプラン (Pblock、ピン配置、グローバル ロジックなど) は既知で、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 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。OOC モジュールはこのフローの場合、必ず必要ではありません。最上位のみ合成して、OOC モジュールに必要なコンテキスト制約を準備するには、OOC モジュールのポート定義を含むブラック ボックスのみが必要になります。

  このフローでは、次が実行できます。

  - 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パラレルに実行できます。チーム メンバーは、アセンブルされたデザインで対応する結果を再利用して、それぞれの担当部分をインプリメントできます。
  - 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、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。

パーシャル リコンフィギュレーション、単独デザイン フローなどのその他の階層デザイン フローは、今後の Vivado Design Suite バージョンでサポートされる予定です。

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、**`-strict`** オプションの使用をお勧めします。チェックポイントの詳細は、『Vivado Design Suite ユーザー ガイド : インプリメンテーション』(UG904) を参照してください。

注意事項

デザイン再利用手法で最適な結果にするためには、特別な注意が必要です。次のセクションでは、Vivado 階層デザイン フローでのアーキテクチャ、設計、制約に関して注意すべき点について説明します。

最適化の制限

デザインの残りの部分と関連しない (OOC) ブロックをインプリメントすると、通常のトップ ダウン フロー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 ブロックを注意して選択します。デザインのほかのブロックからは論理的に孤立し、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。
- ブロック内に完全に含まれるクリティカル パスを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。
- ブロック間のバウンダリをレジスタに入力して、モジュールのバウンダリ最適化制限による損失を最小限に抑えます。
- コンテキスト制約を定義して、ブロックの使用方法に関する情報を提供します。コンテキスト制約の中には、どのポートが定数で駆動されるのか、どのポートを未接続にするのかなどをツールに伝えるために、別の最適化ができるものもあります。詳細は、「コマンドおよび制約」セクションの「OOC 制約」を参照してください。
- モジュール バウンダリ間は最適化されませんので、パーティション分割された関連するデザイン エレメントはすべて一緒に維持する必要があります。

フロアプラン要件

関連のない個別 (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。
- 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。
- ネスト化された Pblock (子 Pblock) は、そのネスト化された Pblock の範囲が親 Pblock の範囲に完全に含まれている限り、OOC インプリメンテーションでサポートされます。

- すべてのクロック バッファー (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ーには LOC 制約を付ける必要があり、最上位のバッファーの位置は **HD.CLK_SRC** 制約で識別される必要があります。
- 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。

階層に関する注意事項

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分割される箇所によっては、結果の質に多大な影響の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で分類するために、階層を追加または変更する必要のあることもあります。

専用接続

専用接続のあるコンポーネントをデザインの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。

- **IOLOGIC 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 IO コンポーネントへの接続が含まれます。
- **GT コンポーネント** : GTX、GTP、およびそれらの専用 I/O 接続。

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

IO およびクロック バッファー

IO およびクロック バッファーは 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。

- **IO バッファー** : OOC ポートが最上位の IO バッファーに直接接続される場合は、このバッファーを 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。インプリメンテーション ツールを使用すれば、IO コンポーネントが完全に表示されるので、最も効率的に配置できます。ただし、すべての状況の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ても、IBUF が OOC モジュール以外のその他のロジックを駆動する場合)。また、この場合、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、「コマンドおよび制約」セクションを参照してください。
- **リージョン クロック バッファー** : BUFR は 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、ツールで BUFR 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 が最上位デザインに含まれる場合、OOC Pblock が BUFR 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。ネスト化された 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される必要があります。ネスト化された Pblock には、BUFR 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> [get_cells -of [get_nets -of [get_ports <bufr_clock_port>]]]`**
  - **`resize_pblock <nested_pblock_name> -add {SLICE_Xx1Yy1:SLICE_Xx2Yy2}`**

これは、最上位の BUFR 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。ネスト化された Pblock の範囲は、最上位インプリメンテーションの BUFR 位置に対応する必要があります。最上位の BUFR 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 バッファー** : グローバル バッファーは 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 ネットが 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。

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、プロジェクトのないバッチ モード/Tcl インターフェイス (ユーザー インターフェイス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。

次のセクションでは、階層デザイン フローに必要な特定のコマンドおよびオプション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。各コマンドの詳細については、『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835) を参照してください。

OOC コマンド

関連のない孤立 (out of context) ブロックを合成またはインプリメントするには、ツールが out-of-context モードで実行される必要があります。これ以外の場合、out-of-context 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
- 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 合成 : ボトムアップ合成のみ

重要 : ボトムアップ合成は、各ブロックにそれぞれ合成プロジェクトがあるような合成フローのことです。通常は、下位ブロックの自動 IO バッファー挿入はオフになります。

この文書では、Vivado 合成フローのみについて説明します。その他のフローについては、『XST ユーザー ガイド』または Synopsys 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 1 : synth_design のオプション

コマンド オプション	説明
-mode out_of_context	合成およびダウンストリーム ツールの IO 挿入が阻止されます。このモードは、**write_checkpoint** が使用されるとチェックポイントに保存されます。
-flatten_hierarchy rebuilt	**-flatten_hierarchy** 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 **rebuilt** が推奨されます。
-top	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。**synth_design** より前に **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。
-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。

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、コマンドを実行する必要があります。書き出す形式には、EDIF または Vivado チェックポイントのいずれかが推奨されます。次のいずれかのコマンドを使用してください。

```
write_edif <file_name>.edf

write_checkpoint <file_name>.dcp
```

プロジェクトなしのデザインを閉じて、合成を再実行せずにインプリメンテーションを実行するには、上記のコマンドのいずれかを実行して、合成結果をファイルに書き出す必要があります。

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シエート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)
- **opt_design** (オプション)
- **place_design**
- **phys_opt_design** (オプション)
- **route_design**

メモリにデザインが含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 2 つの方法のいずれかで実行できます。

方法 1 : ネットリスト デザインの読み出し

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

表 2 : link_design のオプション

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** より前に **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** のを実行中には、OOC モジュールを定義して、モジュール 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。

方法 2 : チェックポイントの読み出し

```
read_checkpoint <file_name>.dcp
```

重要 : `read_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。

OOC 制約

OOC (out-of-context) モジュールを問題なくインプリメントするには、完全デザインにインスタンシエートされる OOC モジュールのコンテキストを記述する必要があります。これには、クロック ポートの定義、定数および未接続ピン、物理制約、タイミング要件などが含まれます。

次の表は、OOC 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) を参照してください。

表 3 : タイミング制約

制約名	説明
`set_input_delay`	入力遅延を定義するのに使用して、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。
`set_output_delay`	出力遅延を定義するのに使用して、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するのに使用します。`create_clock` 制約は、クロック バッファーが最上位にインスタンシエートされていても、OOC モジュールにインスタンシエートされていても、各クロック ポートごとに必要です。

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_input_delay -clock <clock_name> 3.0 [get_ports <ports>]`
- `set_output_delay 5.0 -clock [get_clocks <clock_name>] [get_ports]`

これらのタイミング制約の範囲は、その OOC モジュール自体までです。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 4 : Pblock コマンドとプロパティ

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 を作成するために使用するコマンドです。
`resize_pblock`	サイト タイプ (SLICE、RAMB36 など) と Pblock のサイト位置を定義するために使用するコマンドです。
`add_cells_to_pblock`	Pblock に含まれるインスタンスを指定するために使用するコマンドです。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。

表 4 : Pblock コマンドとプロパティ

コマンド/プロパティ名	説明
CONTAIN_ROUTING	配線リソースが Pblock に含まれないように配線を制御する Pblock プロパティです。デフォルトの値は false です。Pblock RANGE 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で必要とされるビヘイビアです)。
EXCLUDE_PLACEMENT	定義された Pblock RANGE 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。

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もなります。

重要 : 現在のバージョンのソフトウェアでは、1 つのリソース タイプに 1 つの長方形 (Pblock) しか使用できません。

OOC モジュール内のロジックをフロアプランする場合は、ネスト化された 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 オプションを使用して宣言します。

```
create_pblock -parent <parent_pblock_name> <child_pblock_name>
```

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 5 : コンテキスト コマンドとプロパティ

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 バッファーが 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 バッファー インスタンスの位置になります。
HD.PARTPIN_LOCS	配線される指定ポートの特定のインターコネクト位置を定義するために使用し、**HD.PARTPIN_RANGE** の値を上書きします。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、クロックのローカル配線が推測され、専用接続で使用されなくなるので、使用しないでください。

表 5 : コンテキスト コマンドとプロパティ

コマンド/プロパティ名	説明
HD.PARTPIN_RANGE	指定した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) の範囲を定義するために使用します。クロック ポートに使用すると、クロックのローカル配線が推測されるので、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。
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て指定すると、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_Xx1Yy1 [get_ports <*port_name*>]**
- **set_property HD.PARTPIN_RANGE SLICE_Xx1Yy1 :SLICE_Xx2Yy2 [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 インプリメンテーション 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 OOC 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、次のアイコ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)。

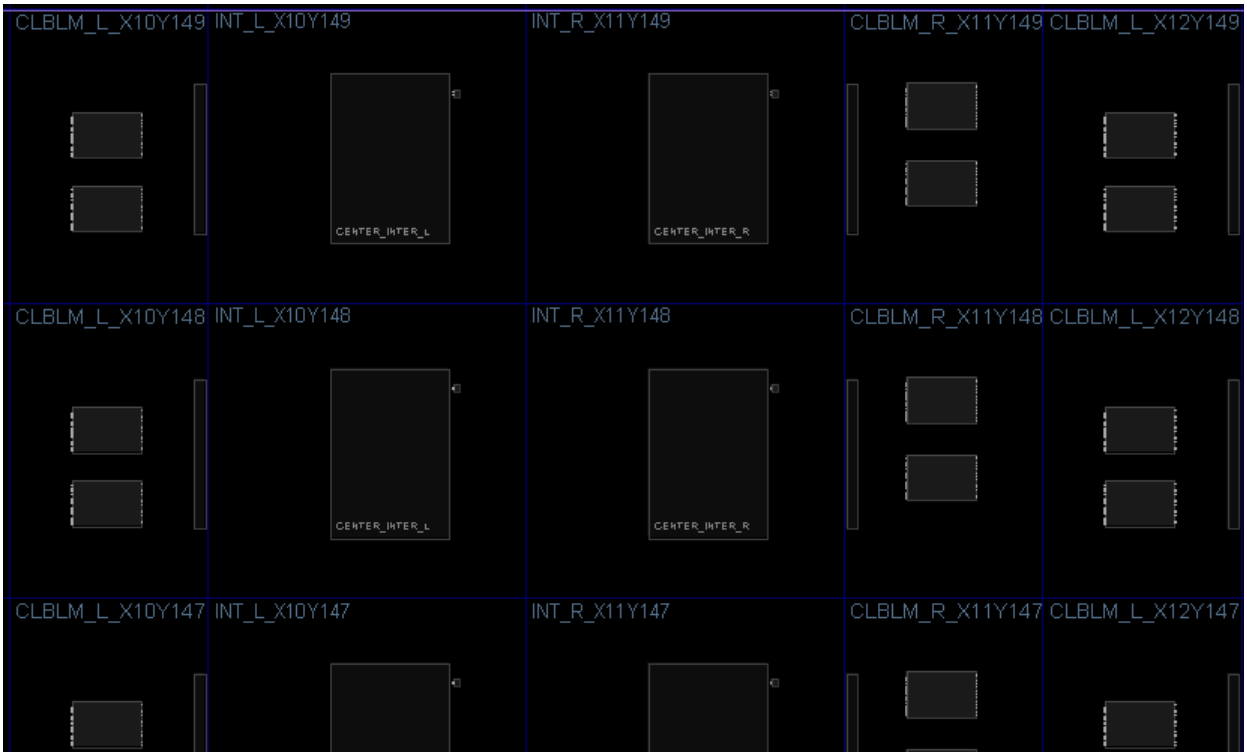

図 1 : INT タイルの位置

最上位アセンブリ (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 インプリメンテーション 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、インプリメント済み OOC モジュールを読み込む各インスタンスのブラックボックスと一緒に最上位デザインに読み込まれます。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 IO バッファーを推論しますが、IO バッファーが OOC モジュールに特にインスタンシエートされる場合は、ポートごとに最上位合成で IO バッファー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 **`BUFFER_TYPE`** です。**`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) 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、次の 2 つの手順を除いて、標準デザインと同じように実行できます。

1. パーティション分割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。
2. 保持レベル (logical、placement、routing) を選択します。

OOC チェックポイントの読み込み

OOC チェックポイントは、**read_checkpoint** コマンドを使用して読み込みます。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 6 : read_checkpoint のオプション

オプション名	説明
-cell	OOC 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。
-strict	セルを置き換えるのにポートが完全に一致する必要があるので、そのパーツ、パッケージ、スピードグレード値が同じかどうかがチェックされます。
<*file*>	読み込む OOC チェックポイントを指定します。

保持レベルの設定

OOC 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。

インポートされた OOC チェックポイントの配置配線をロックするには、**lock_design** という Tcl プロシージャーが存在する必要があります。

```
lock_design <cell_name> <preservation> <lock_value>
```

このプロシージャー内で使用できる保持レベルは、次のとおりです。

- **logical** : 論理デザインを保持します。配置または配線情報は使用されますが、ツールが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。配置情報は使用されますが、ツールが結果を改善できる可能性がある場合は変更可能です。
- **routing** : 論理および配置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。OOC インプリメンテーション中は、**CONTAIN_ROUTING** 制約を使用する必要があります。

これらのプロパティ以外の組み合わせはサポートされません。また、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。物理データベースは、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 7 : Tcl プロシージャーの **lock_design** 引数

引数名	説明
<cell_name>	ロックされる階層セル名です。
<preservation>	保持レベルを指定します。使用できる値は、**logical**、**placement**、または **routing**、デフォルト値は **placement** です。
<lock_value>	**1** でロック、**0** でロック解除になります。

最上位アセンブリ (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

このセクションでは、提供されているスクリプトの詳細について説明します。ユーザーの必要性に合わせてスクリプトを修正するための参照にしてください。

design.tcl

design.tcl スクリプトは、提供されているスクリプトを実行するために知っておく必要のある重要なファイルです。後のセクションでは、提供されるその他のさまざまなスクリプトについて説明しますが、design.tcl を編集するだけで、これらのスクリプトを使用したデザインおよびフローを設定することができます。

セクション 1 : Tcl 変数

このセクションのコマンドでは、その他の Tcl ファイルの存在するディレクトリだけが設定されます。この部分は編集する必要はありません。1 つまたは複数の Tcl の param を設定するコマンドはありますが、コメントアウトされています。これは通常必要ありませんが、必要であればスクリプトでサポートされます。

```
set tclParams [list <param1> <value> <param2> <value> ...<paramN> <value>]
```

セクション 2 : 変数の設定

このセクションで定義される変数は、フローのどの部分が実行されるかを定義します。これにより、デザインの 1 段階に集中でき、その他の手順は必要な場合にのみ実行できます。次の表では、これらの変数の詳細を説明しています。

表 8 : フロー制御変数

変数名	説明
synthBlock	1 に設定されると、$oocModules で定義されるすべてのモジュールに対して **synth_design -mode out_of_context** が実行されます。結果は $synthDir/$module に出力され、$netlistDir/$module にもコピーされます。
implBlock	1 に設定されると、$oocModules で定義されるすべての $module#_instances に対してインプリメンテーション (**opt_design**、**place_design**、**phys_opt_design**、**route_design**) が実行されます。インプリメンテーションのどの部分を実行するか制御するには、run.tcl スクリプトを参照してください。 ネットリストは $netlistDir/$module に含まれている必要があります。サードパーティ フローの場合はこのフォルダーにネットリストを手動で追加するか、この手順の前に **synthBlock** を実行します。
synthTop	1 に設定されると、最上位モジュールに対して synth_design が実行されます。結果は $synthDir/$top に出力され、$netlistDir/$top にもコピーされます。
implTop	1 に設定されると、最上位ネットリストが読み込まれ、各 OOC インスタンスの保持レベルがロックされ、最上位のインプリメンテーション (**opt_design**、**place_design**、**phys_opt_design**、**route_design**) が実行されます。インプリメンテーションのどの部分を実行するか制御するには、run.tcl スクリプトを参照してください。 各 OOC インスタンスには、$implDir/$instance にチェックポイントが必要です。チェックポイントがない場合は、この手順より前に **implBlock** を実行します。

このセクションには、フロー制御変数に加えて、入力および出力ディレクトリも含まれます。次の表では、これらのディレクトリについて説明しています。

表 9 : ディレクトリ変数

変数名	説明
srcDir	rtl、prj、xdc および netlist の下位ディレクトリを含む主要なソース ディレクトリです。
rtlDir	$srcDir の下位ディレクトリと仮定され、合成用の RTL ソース ファイルすべてが含まれます。このディレクトリには各モジュールの下位フォルダー (例 : $top、$module1、$module2) を含めることをお勧めします。
prjDir	$srcDir の下位ディレクトリと仮定され、XST の prj ファイルが保存されます。これにより、XST から VDS へデザインを変換しやすくなりますが、XST ファイルが存在しなくても PRJ は簡単に作成できます。prj ファイルが解析されると、合成で読み込まれるファイルがツールに伝えられます。
xdcDir	$srcDir の下位ディレクトリと仮定され、最上位と OOC インスタンスの XDC ファイルが保存されます。
netlistDir	$srcDir の下位ディレクトリと仮定され、サードパーティ合成のネットリスト ファイルと VDS フローを使用した場合の **synth_design** の結果が保存されます。
synthDir	RTL フローを実行した場合の **synth_design** の出力ディレクトリです。最上位および各 OOC モジュールの下位ディレクトリが作成されます。これらの結果は $netlistDir にコピーされ、合成フローの使用の有無に関係なく、互換性のあるインプリメンテーション スクリプトができるようになります。
implDir	すべてのインプリメンテーションの出力ディレクトリです。最上位および各 OOC インスタンスの下位ディレクトリが作成されます。

セクション 3 : 最上位の定義

このセクションでは、合成およびインプリメンテーションの最上位モジュールに関するすべてが定義されます。

表 10 : 最上位モジュールの変数

変数名	説明
top	最上位モジュール名
top_prj	合成の PRJ ファイルへのパス。ここで定義されると、**top_vlog**、**top_sysvlog**、**top_vhdl** の値が上書きされます。
top_vlog_headers	Verilog ヘッダー ファイルの識別に使用
top_vlog_defines	Verilog 定義に使用
top_vlog	Verilog ファイルと関連するコンパイル ライブラリの定義に使用
top_sysvlog	SystemVerilog ファイルと関連するコンパイル ライブラリの定義に使用
top_vhdl	VHDL ファイルと関連するコンパイル ライブラリの定義に使用
top_cores	IP コアのネットリストの定義に使用
top_xdc	合成で使用される XDC ファイル
top_impl_xdc	インプリメンテーションで使用される XDC ファイル
top_impl_preservation	編集不可。現時点ではサポートなし。
top_impl	編集不可。最上位インプリメンテーションのすべての変数のリスト
topModule	編集不可。最上位に関するすべての変数のリスト

サードパーティ合成フローの場合、すべての Verilog および VHDL 変数は空のままにできます。最上位のネットリストは、$netlistDir/$top/$top.{edf,edn, ngc} に保存されます。

セクション 4 : OOC モジュールの定義

このセクションでは、最上位デザインのインスタンスすべてを含む OOC モジュールに関するすべてが定義されます。このセクションは、デザインのすべての OOC モジュールに対して複製する必要があります。また、**module#_inst#_*** という名前の変数はモジュールのインスタンスごとに複製する必要があります。

表 11 : OOC モジュールの変数

変数名	説明
module#	モジュール名
module#_prj	最上位合成の PRJ ファイルへのパス。ここで定義されると、**top_vlog**、**top_sysvlog**、**top_vhdl** の値が上書きされます。
module#_vlog_headers	Verilog ヘッダー ファイルの識別に使用
module#_vlog_defines	Verilog 定義に使用
module#_vlog	Verilog ファイルと関連するコンパイル ライブラリの定義に使用
module#_sysvlog	SystemVerilog ファイルと関連するコンパイル ライブラリの定義に使用
module#_vhdl	VHDL ファイルと関連するコンパイル ライブラリの定義に使用
module#_cores	IP コアのネットリストの定義に使用
module#_xdc	合成で使用される XDC ファイル
module#_inst#_name	OOC モジュールのローカル インスタンス名 (例 : u1)
module#_inst#_hierName	OOC モジュールの完全階層インスタンス名 (例 : a/b/u1)。アセンブリ中に OOC インスタンスを最上位へ正しくインポートするために使用
module#_inst#_xdc	OOC インプリメンテーションで使用される XDC ファイル
module#_inst#_preservation	アセンブリ中にインポートされた OOC インスタンスの保持レベル (logical、placement、routing) を定義するために使用され、lock_design プロシージャへ渡されます。
module#_instance#	編集不可。OOC インスタンスに関するすべての変数のリスト
module#_instances	モジュールに対して定義された **$module#_instance#** 変数すべてのリスト

ヒント : サードパーティ合成フローまたはネットリストのみの IP の場合、すべての Verilog および VHDL 変数は空のままにできます。モジュールのネットリストは、`$netlistDir/$module#/$module#.{edf, edn, ngc}` に保存されます。モジュールの下位レベルの IP コア ネットリストは、**$module#_cores** で定義できます。

セクション 5 : デザイン サマリ

このセクションには、スクリプトで合成/インプリメントされるすべてのモジュールのリストである数個のリストのみが含まれます。**$Modules** 変数には、**$oocModules** および **$topModule** にリストされる OOC モジュールがすべてリストされるはずです。

また、このセクションは、合成およびインプリメンテーション ツールを呼び出す `run.tcl` スクリプトも source で読み込みます。

既知の問題

現在の 2012.3 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。

OOC モジュール内の IP ロジックの制限

現在のところ、OOC モジュール内に IO バッファーを挿入することはお勧めできません。IO を含む IP コアがある場合、またはこれが必要なその他の状況では、すべての専用タイプの接続を同じパーティションに置きます。OOC モジュールに I/OLOGIC ブロックが含まれる場合は、それに関連するバッファーもモジュールに含まれるはずです。

配線保持の制限

現在のところ、OOC モジュールをインポートする際に、配線保持に関する次のような制限があります。

- インターフェイス ネットが保持されません。
- OOC インプリメンテーションで **CONTAIN_ROUTING** Pblock プロパティが使用されている必要があります。これは、配線がロックされると回避することのできないような配線競合が発生しないようにするためです。
- インターフェイス配線が存在する必要があります。これらは保持はされませんが、インターフェイス配線は存在する必要があり、OOC インプリメンテーション中に配線される必要があります。これには、OOC インプリメンテーションで **HD.PARTPIN** 制約を使用する必要があります。

グローバル クロックの制限

最上位のバッファー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 バッファーは OOC インプリメンテーション中に配線されます。

OOC モジュールより上位で定義されるクロック間の関係は、OOC モジュールがコンパイルされた時点ではわかりません。たとえば、同じ MMCM で生成されても位相シフト値の異なる 2 つのクロックは、OOC モジュール内に両方のクロックが読み込まれる場合には、同じであると判断されます。

タイミング制約処理に関する既知の問題

物理およびタイミング制約はすべて、**write_checkpoint** または **write_xdc** で作成されるデザイン XDC ファイルに格納されます。タイミング制約の複製、または再利用フローが使用される際のその他の問題が発生しないようにするには、これらの OOC モジュールのチェックポイントの作成時にバウンダリ タイミング制約を削除することをお勧めします。これらの制約には、最上位で作成されるクロックに対する **create_clock**、**set_input_delay**、**set_output_delay** などがあります。これらの制約は、モジュール チェックポイントを書き出す前に **reset_timing** コマンドを実行すると削除できます。これにより、内部タイミング制約と **set_logic_*** 制約が削除されるので、最上位デザインのインプリメンテーションが終了するには、これらの制約を再構築する必要があります。これには、これらの制約のみを含むモジュール XDC ファイルを読み込み直すか、最上位 XDC ファイルにこれらの制約を含めます。

XILINX

付録 A

その他のリソース

ザイリンクス リソース

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

http://japan.xilinx.com/support

ザイリンクス資料で使用される用語集は、次を参照してください。

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

ソリューション センター

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

リファレンス

- Vivado™ Design Suite 2012.3 資料ページ : http://japan.xilinx.com/support/documentation/dt_vivado_vivado2012-3.htm
- 『階層デザイン手法ガイド』(UG748)