

# Vivado Design Suite チュートリアル

## Dynamic Function eXchange

UG947 (v2020.1) 2020 年 9 月 22 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
2020 年 9 月 22 日 v2020.1	
<a href="#">演習 8: Nested DFX</a>	新しい演習を追加。

# 目次

改訂履歴.....	2
概要.....	6
ハードウェアおよびソフトウェア要件.....	6
チュートリアル デザインの概要.....	6
演習 1: 7 シリーズの基本 DFX フロー.....	9
手順 1: チュートリアル デザイン ファイルの抽出.....	9
手順 2: スクリプトの確認.....	9
手順 3: デザインの合成.....	10
手順 4: デザインのまとめおよびインプリメンテーション.....	11
手順 5: デザイン フロアプランの構築.....	12
手順 6: 初回コンフィギュレーションのインプリメント.....	18
手順 7: 2 回目のコンフィギュレーションのインプリメント.....	22
手順 8: スクリプトをハイライトして結果を確認.....	25
手順 9: ビットストリームの生成.....	27
手順 10: FPGA のパーシャル リコンフィギュレーション.....	29
まとめ.....	29
演習 2: UltraScale の基本 DFX フロー.....	31
手順 1: チュートリアル デザイン ファイルの抽出.....	31
手順 2: スクリプトの確認.....	31
手順 3: デザインの合成.....	32
手順 4: デザインのまとめおよびインプリメンテーション.....	33
手順 5: デザイン フロアプランの構築.....	34
手順 6: 初回コンフィギュレーションのインプリメント.....	36
手順 7: 2 回目のコンフィギュレーションのインプリメント.....	40
手順 8: スクリプトをハイライトして結果を確認.....	42
手順 9: ビットストリームの生成.....	44
手順 10: FPGA のパーシャル リコンフィギュレーション.....	46
まとめ.....	47
演習 3: DFX プロジェクト フロー.....	48
手順 1: チュートリアル デザイン ファイルの抽出.....	48
手順 2: 初回デザイン ソースの読み込み.....	48
手順 3: Dynamic Function eXchange ウィザードを使用しデザインを完了.....	52
手順 4: 現在のデザインの合成およびインプリメンテーション.....	56
手順 5: 追加リコンフィギュラブル モデルおよびコンフィギュレーションの追加.....	59
手順 6: グレー ボックス モジュールの作成およびインプリメンテーション.....	63

手順 7: デザイン ソースまたはオプションの変更.....	65
まとめ.....	66
<b>演習 4: Vivado デバッグおよび DFX プロジェクト フロー.....</b>	<b>67</b>
手順 1: チュートリアル デザイン ファイルの抽出.....	67
手順 2: 初回デザイン ソースの読み込み.....	67
手順 3: DFX 用のデザイン設定.....	70
手順 4: DFX ウィザードを使用してデザインの残りの部分を完成.....	73
手順 5: リCONFIGYラブル モジュール内での IP の追加.....	76
手順 6: デザインの合成およびフロアプランの作成.....	78
手順 7: DFX コンフィギュレーション解析レポートの実行.....	82
手順 8: デザインのインプリメント.....	83
手順 9: 追加リCONFIGYラブル モジュールおよびコンフィギュレーションの追加.....	86
手順 10: ビットストリームの生成.....	90
手順 11: ボードへの接続および FPGA のプログラム.....	91
まとめ.....	98
<b>演習 5: 7 シリーズ デバイスの DFX Controller IP.....</b>	<b>99</b>
手順 1: チュートリアル デザイン ファイルの抽出.....	99
手順 2: Dynamic Function eXchange (DFX) Controller IP のカスタマイズ.....	99
手順 3: デザインのコンパイル.....	105
手順 4: ボードの設定.....	106
手順 5: サンプル デザインの操作.....	107
手順 6: FPGA での DFX Controller のクエリ.....	109
手順 7: FPGA での DFX Controller の変更.....	111
まとめ.....	112
<b>演習 6: UltraScale デバイスの DFX Controller .....</b>	<b>114</b>
手順 1: チュートリアル デザイン ファイルの抽出.....	114
手順 2: Dynamic Function eXchange (DFX) Controller IP のカスタマイズ.....	114
手順 3: デザインのコンパイル.....	120
手順 4: ボードの設定.....	121
手順 5: サンプル デザインの操作.....	122
手順 6: FPGA での DFX Controller のクエリ.....	123
手順 7: FPGA での DFX Controller の変更.....	124
まとめ.....	126
<b>演習 7: UltraScale+ デバイスの DFX Controller IP.....</b>	<b>127</b>
手順 1: チュートリアル デザイン ファイルの抽出.....	127
手順 2: チュートリアル デザイン ファイルの処理.....	127
手順 3: チュートリアル デザインの実行.....	132
まとめ.....	139
<b>演習 8: Nested DFX.....</b>	<b>140</b>
概要.....	140
手順 1: チュートリアル デザイン ファイルの抽出.....	140



手順 2: スクリプトの確認.....	140
手順 3: デザインの合成.....	142
手順 4: デザインのまとめおよびインプリメンテーション.....	142
手順 5: ハードウェアでのデザインのテスト.....	152
まとめ.....	154
 付録 A: その他のリソースおよび法的通知.....	 156
ザイリンクス リソース.....	156
Documentation Navigator およびデザイン ハブ.....	156
その他のリソース.....	156
Please Read: Important Legal Notices.....	157

# 概要

このチュートリアルでは、Vivado® Design Suite 2020.1 での Dynamic Function eXchange (DFX) のソフトウェア サポートについて説明します。

**演習 1: 7 シリーズの基本 DFX フロー** および **演習 2: UltraScale の基本 DFX フロー** では、現在の DFX デザイン フロー、サンプルの Tcl スクリプトの基本、Vivado 統合設計環境 (IDE) 内で結果を説明します。演習の一部でスクリプトを実行し、ほかの部分はインタラクティブに設計していきます。また、フロー全体をスクリプト化することも可能で、その場合はスクリプトにデザイン ファイルが含まれます。これらの演習は、RTL からビットストリームまでのソフトウェア フローに焦点をあてており、DFX の処理方法を実際に学びます。演習 2 では、UltraScale+™ デバイスも対象になっています。

**演習 3: DFX プロジェクト フロー** では、Vivado® IDE 内でプロジェクト フローを順番に進めていきます。DFX ウィザードを使用してデザインを確立するところから、合成、繰り返し実行、設計作業の繰り返しまでを説明します。**演習 4: Vivado デバッグおよび DFX プロジェクト フロー** でもプロジェクト フローを説明しますが、ここでは IP デバッグ コアの追加、Vivado® ハードウェア マネージャーを介したデバッグ作業を説明します。

**演習 5: 7 シリーズ デバイスの DFX Controller IP**、**演習 6: UltraScale デバイスの DFX Controller**、および **演習 7: UltraScale+ デバイスの DFX Controller IP** は、Vivado Design Suite での DFX Controller IP の基本および機能の説明を目的にしています。パーシャル ビットストリームの管理は DFX で新しく導入されたデザイン要件の 1 つです。パーシャル ビットストリームがいつ必要になるのか、それをどこに保存するのか、コンフィギュレーション エンジンにそれをどう渡すのか、また、新しいパーシャル ビットストリームを渡す前後およびその最中に、スタティック デザインはどう動作するのかを計画しておく必要があります。DFX Controller IP はこうした課題を克服しやすくするために設計されています。

Nested DFX は、リコンフィギュラブルなパーティションをリコンフィギュラブルなパーティションの中に入れ子にするためのフローと手法を示しています。DFX ソリューションのこの拡張機能により、UltraScale または UltraScale+ デバイスでのダイナミック リコンフィギュレーションの柔軟性がさらに高まります。

---

## ハードウェアおよびソフトウェア要件

このチュートリアルを実行するには、Vivado Design Suite 2020.1 以降のバージョンをインストールしておく必要があります。

このチュートリアルの演習では、8 つの異なるザイリンクス開発プラットフォームをターゲットにしています。特筆されていない限り、各演習の手順に一致させるには、プロダクション シリコンおよびプロダクション ボードが必要です。OS サポートについては、『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) を参照してください。

---

## チュートリアル デザインの概要

チュートリアルの演習のデザインは ZIP ファイルになっていて、ザイリンクスのウェブサイトからダウンロードできます。この ZIP ファイル内に、演習ごとのフォルダーがあります。チュートリアル デザイン ファイルにアクセスするには、次の手順に従ってください。

1. ザイリンクス ウェブサイトから[リファレンス デザイン ファイル](#)をダウンロードします。
2. ZIP ファイルの内容を書き込み可能なディレクトリに抽出します。

### 演習 1: 7 シリーズ 基本 DFX のフロー

このチュートリアルで使用するサンプル デザインは、`led_shift_count_7s` という名前です。このデザインは、次のザイリンクス開発プラットフォームをターゲットにしています。

- KC705 (xc7k325t)
- VC707 (xc7vx485t)
- VC709 (xc7vx690t)
- AC701 (xc7a200t)

このデザインは非常にコンパクトなので、データ サイズを最小限に抑えやすく、最小限のハードウェア要件で、チュートリアルをすばやくこなせます。

### 演習 2: UltraScale™ 基本 DFX

このチュートリアルで使用するサンプル デザインは、`led_shift_count_us` という名前です。このデザインは、次のザイリンクス開発プラットフォームをターゲットにしています。

- KCU105 (xcku040)
- VCU108 (xcvu095)
- KCU116 (xcku5p)
- VCU118 (xcvu9p)

### 演習 3: DFX プロジェクト フロー

このチュートリアルで使用するサンプル デザインは、`dfx_project` という名前です。これは、演習 1 で使用した `led_shift_count` を変更したもので、カウンター とシフトのインスタンスが 1 つずつではなく、2 つのシフト インスタンスが含まれています。これで、パーティション タイプが同じであるインスタンスすべてに、1 つのパーティション定義が適用されることを学べるようになっています。このデザインは、次のザイリンクス開発プラットフォームをターゲットにしています。

- KC705 (xc7k325t)
- VC707 (xc7vx485t)
- VC709 (xc7vx690t)
- KCU105 (xcku040)
- KCU116 (xcku5p)
- VCU108 (xcvu095)
- VCU118 (xcvu9p)

### 演習 4: Vivado デバッグおよび DFX プロジェクト フロー

ここで使用されるサンプル デザインは、`dfx_project_debug` という名前です。このデザインは、次のザイリンクス開発プラットフォームをターゲットにしています。

- KCU105 (xcku040)

- VCU108 (xcvu095)
- KCU116 (xcku5p)
- VCU118 (xcvu9p)

#### 演習 5: 7 シリーズ デバイスの DFX Controller IP

このチュートリアルで使用するサンプル デザインは、`dfxc_7s` という名前で、演習 1 で使用したデザインをベースにしています。このデザインは、次のザイリンクス開発プラットフォームをターゲットにしています。

- KC705 (xc7k325t)
- VC707 (xc7vx485t)
- VC709 (xc7vx690t)

#### 演習 6: UltraScale デバイスの DFX Controller IP

このチュートリアルで使用するサンプル デザインは、`dfxc_us` という名前です。このデザインは、VCU108 デモ ボード Rev 1.0 で使用するため xcvu095 デバイスをターゲットにし、演習 2 で使用したデザインをベースにしています。

#### 演習 7: UltraScale+ デバイスの DFX Controller IP チュートリアル デザイン

このチュートリアルで使用するサンプル デザインは、`dfxc_usp` という名前で、[演習 6: UltraScale デバイスの DFX Controller](#) で使用したデザインをベースにしています。このデザインは、KCU116 および VCU118 デモ ボードをターゲットにしています。

#### 演習 8: Nested Dynamic Function eXchange

このチュートリアルのサンプル デザインは、シフト カウント デザインのバリエーションの 1 つで、8 つの LED すべてに対してシフターやカウンターをコンフィギュレーションしたり、8 つある LED のうち 4 つのみを変更して、より低い粒度でリコンフィギュレーションしたりできます。このデザインは、演習 4: KCU105、VCU108、KCU116、および CCU118 と同じ UltraScale および UltraScale+ 開発プラットフォームをターゲットにしています。

## 演習 1

# 7 シリーズの基本 DFX フロー

この演習では、7 シリーズ デバイスの基本的な Dynamic Function eXchange (DFX) フローについて説明します。まずスクリプトを使用して、スタティック モジュールおよび各リコンフィギャラブル モジュール (RM) のバリエーションを個別に合成します。それから IDE で、RM の位置を Pblock を使用して制約し、デザインの初回コンフィギュレーションをインプリメントします。次に、デザインのスタティック部分をロックし、RM を少しずつ変えながらインプリメンテーションを再実行して、さまざまなコンフィギュレーションをインプリメントします。最後に、各インプリメント済みの RM がデザインのスタティック部分と互換性があることを検証し、互換性があればビットストリームを生成します。

## 手順 1: チュートリアル デザイン ファイルの抽出

1. ザイリンクス ウェブサイトから[リファレンス デザイン ファイル](#)をダウンロードします。
2. ZIP ファイルの内容を書き込み可能なディレクトリに抽出します。
3. `\led_shift_count_7s` に移動します。

## 手順 2: スクリプトの確認

まずは、デザイン アーカイブにあるスクリプトを確認します。`run_dfx.tcl` および `advanced_settings.tcl` ファイルは、ルート ディレクトリにあります。`run_dfx.tcl` スクリプトには、Dynamic Function eXchange の実行に必要な最小限の設定が含まれています。`advanced_settings.tcl` には、デフォルト フローの設定が含まれているので、上級ユーザーでない限り、このファイルは変更しないでください。

### メインスクリプト

`\led_shift_count_7s` にある `run_dfx.tcl` をテキスト エディターで開きます。これは、デザイン パラメーター、デザイン ソース、デザイン 構造を定義するマスター スクリプトです。Dynamic Function eXchange デザイン全体をコンパイルするには、このファイルのみを変更します。`run_dfx.tcl`、`advanced_settings.tcl` などの基本スクリプトの詳細は、`Tcl_HD` サブディレクトリにある `README.txt` を参照してください。

`run_dfx.tcl` の詳細は次のとおりです。

- [Define target demo board] では、このデザインでサポートされているデモ ボードの中から 1 つボードを選択できます。
- [flow control] では、実行する合成およびインプリメンテーションのフェーズを制御できます。このチュートリアルでは、合成のみスクリプトで実行できます。インプリメンテーション、検証、ビットストリーム生成はインタラクティブに実行されます。これらの追加ステップをスクリプトから実行するには、フロー変数 (`run.prImpl` など) を 1 に変更します。

- [Output Directories] および [Input Directories] では、デザイン ソースおよび結果ファイルに必要なファイル構造が設定されます。ファイル構造を変更する場合は、その変更をここに反映させる必要があります。
- [Top Definition] および [RP Module Definitions] セクションでは、デザインの各部分のソース ファイルをすべて参照できます。[Top Definition] は、スタティック デザインに必要なソース (制約および IP を含む) がすべて対象になっています。[RP Module Definitions] では、リコンフィギャラブル パーティション (RP) のソースが対象になっています。各 RP を特定し、各 RP のリコンフィギャラブル モジュール (RM) のバリエーションをすべてリストします。
  - このデザインには 2 つの RP (`inst_shift` および `inst_count`) があり、各 RP にモジュールのバリエーションが 2 つあります。
- 「Configuration Definition」 セクションでは、1 コンフィギュレーションを構成するスタティック モジュールおよび RM のセットを定義します。
  - このデザインには、`config_shift_right_count_up_implement` および `config_shift_left_count_down_import` のマスター スクリプト内で定義された 2 つのコンフィギュレーションがあります。
  - さらにコンフィギュレーションを作成する場合は、RM を追加するか、既存の RM をまとめます。

### サポート スクリプト

`Tcl_HD` サブディレクトリの下には、サポート Tcl スクリプトがいくつかあります。これらのスクリプトは `run_dfx.tcl` に呼び出され、Dynamic Function eXchange フローの詳細を管理します。主な DFX スクリプトについていくつか詳細を次に挙げます。



**注意:** サポート Tcl スクリプトは変更しないでください。

- `step.tcl`: チェックポイントを監視して、デザインの現在のステータスを管理します。
- `synthesize.tcl`: 合成フェーズの詳細をすべて管理します。
- `implement.tcl`: モジュール インプリメンテーション フェーズの詳細をすべて管理します。
- `dfx_utils.tcl`: DFX デザインの差上位インプリメンテーション フェーズの詳細をすべて管理します。
- `run.tcl`: 合成およびインプリメンテーションの実際の run を起動します。
- `log_utils.tcl`: フロー中のキー ポイントでレポート ファイルを生成します。

残りのスクリプトは、これらのスクリプト内の詳細を提供したり (ほかの `*_utils.tcl` スクリプトなど)、ほかの階層デザインフローを管理したりします (`hd_utils.tcl` など)。

## 手順 3: デザインの合成

`run_dfx.tcl` スクリプトは、このチュートリアル of 合成フェーズを自動化します。最上位のスタティック デザインに 1 回、4 つの RM それぞれに 1 回ずつ、合計 5 回、合成の繰り返し実行が呼び出されます。

### 1. Vivado Tcl シェルを開きます。

- Windows で、ザイリンクス Vivado デスクトップ アイコンをクリックするか、または [Start] → [All Programs] → [Xilinx Design Tools] → [Vivado 2020.1] → [Vivado 2020.1 Tcl Shell] をクリックします。
- Linux の場合は、「`vivado -mode tcl`」と入力するだけです。

2. シェルで `\led_shift_count_7s` に移動します。
3. KC705 以外のデモ ボードをターゲットにする場合は、`run_dfx.tcl` で `xboard` 変数を変更します。  
ほかに使用できるボードは、VC707、VC709、および AC701 です。
4. 次のコマンドを入力して、`run_dfx.tcl` スクリプトを実行します。

```
source run_dfx.tcl -notrace
```

Vivado 合成で 5 回の繰り返し実行をすべて完了しても、Vivado Tcl シェルは開いたままです。各モジュールのログやレポート ファイル、最終チェックポイントは、Synth サブディレクトリに各モジュール名の付いたフォルダがあって、その中に生成されています。



**ヒント:** `\led_shift_count_7s` ディレクトリには、複数のログ ファイルが作成されています。

- `run.log` には、Tcl シェルのウィンドウに表示されたサマリが書き込まれています。
- `command.log` には、スクリプトにより実行された個々のステップすべてが書き込まれています。
- `critical.log` には、実行中に出力されたクリティカル警告メッセージがすべて書き込まれています。

## 手順 4: デザインのまとめおよびインプリメンテーション

各モジュールおよび最上位の合成済みチェックポイントが生成されたので、デザインをまとめる準備が整いました。

すべてのフロー ステップは Tcl コンソールから実行されますが、インタラクティブなイベントに対しては IDE 内の機能 (フロアプランニング ツールなど) を使用できます。



**ヒント:** Vivado IDE でのミス タイプを避けるため、このチュートリアルから直接コマンドをコピーして貼り付けます。コマンドは 1 つずつコピーして貼り付けてください。中には、複数行にわたる長いコマンドもあるので注意してください。

1. Vivado® IDE を開きます。開いている Tcl シェルに「`start_gui`」と入力して Vivado® IDE を開くか、Vivado® の「`-mode gui`」コマンドを実行して Vivado を起動します。
2. 現在のディレクトリが `\led_shift_count_7s` でない場合は、そこへ移動します。pwd コマンドを使用すると、現在のディレクトリを確認できます。
3. 変数を設定します。このチュートリアルからコマンドをコピーして Tcl コンソールに貼り付けるときに役立ちます。この演習でターゲットにしているパーツおよびボードを選択し、Vivado でそれを適用します。

```
set part "xc7k325t-ffg900-2"
set board "kc705"

set part "xc7vx485t-ffg1761-2"
set board "vc707"

set part "xc7vx690t-ffg1761-2"
set board "vc709"

set part "xc7a200t-fbg676-2"
set board "ac701"
```

4. Tcl コンソールで次のコマンドを実行し、インメモリ デザインを作成します。

```
create_project -in_memory -part $part
```

5. 次のコマンドを実行し、スタティック デザインを読み込みます。

```
add_files ./Synth/Static/top-synth.dcp
```

6. 次のコマンドを実行し、最上位のデザイン制約を読み込みます。

```
add_files ./Sources/xdc/top_io_$board.xdc
set_property USED_IN {implementation} [get_files ./Sources/xdc/top_io_$board.xdc]
```

使用可能な XDC ファイルの top\_io\_\$board バージョンを選択すると、ピンの位置およびクロック制約が読み込まれますが、フロアプラン情報は読み込まれません。top\_\$board バージョンを読み込むと、ピンの位置、クロック制約、およびフロアプラン制約が読み込まれます。

7. 次のコマンドを実行し、シフトおよびカウント ファンクションの最初の合成チェックポイントを 2 つ読み込みます。

```
add_files ./Synth/shift_right/shift-synth.dcp
set_property SCOPED_TO_CELLS {inst-shift} [get_files ./Synth/shift_right/shift-synth.dcp]
add_files ./Synth/count_up/count-synth.dcp
set_property SCOPED_TO_CELLS {inst-count} [get_files ./Synth/count_up/count-synth.dcp]
```

SCOPED\_TO\_CELLS プロパティを使用すると、ターゲット セルに正しく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: 制約の使用』 (UG903) のこのセクションを参照してください。

8. link\_design コマンドを使用してデザイン全体をリンクします。

```
link_design -mode default -reconfig_partitions {inst-shift inst-count} -part $part -top top
```

この時点で、スタティック部分とリコンフィギュラブル ロジックを含んだフル コンフィギュレーションが読み込まれます。非プロジェクト モードで作業している間は、Flow Navigator は表示されないので注意してください。



**ヒント:** [Layout] → [Floorplanning] をクリックして、IDE をフロアプランニング モードにします。[Device] ビューが表示されていることを確認します。

9. この初回コンフィギュレーションでまとめられたデザインを保存します。

```
write_checkpoint -force ./Checkpoint/top-link-right-up.dcp
```

## 手順 5: デザイン フロアプランの構築

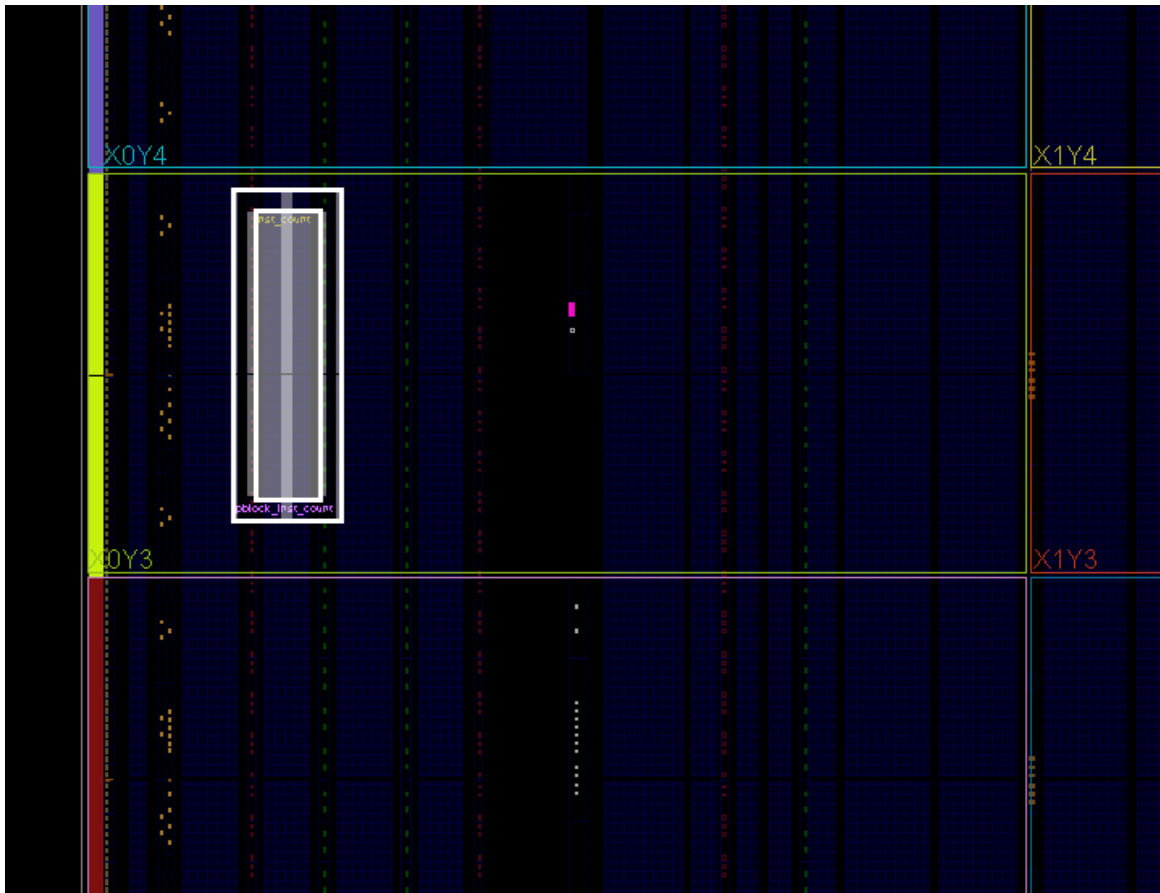
次に、部分的にリコンフィギュレーションされる領域を定義するため、フロアプランを作成します。

1. [Netlist] ウィンドウで [inst\_count] を選択します。右クリックして、[Floorplanning] → [Draw Pblock] をクリックするか、[Draw Pblock] ツールバー ボタンをクリックして、X0Y3 クロック領域の左側に細長い長方形を描きます。この段階では、長方形のサイズや形状は重要ではありませんが、クロック領域内に配置してください。

[Device] ビューで Pblock が選択されていることを確認し、作業を続けます。



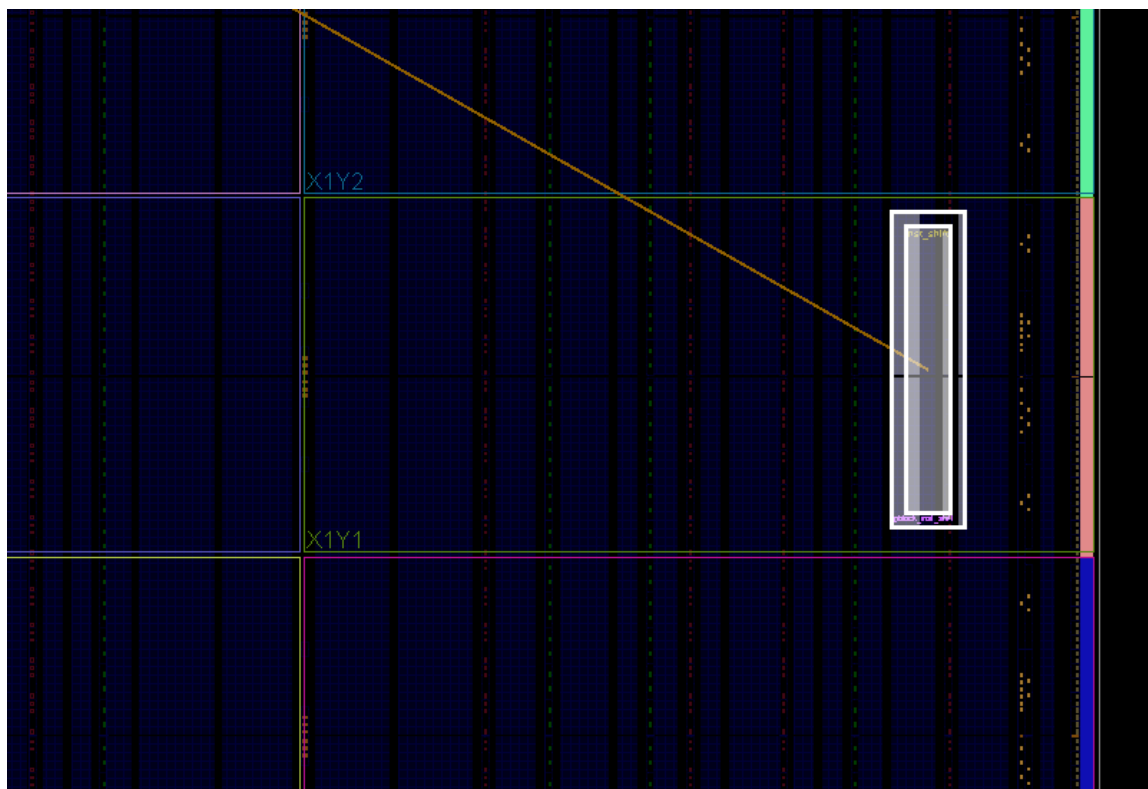
図 1: inst\_count リコンフィギャラブル パーティションの Pblock



この RM には CLB リソースのみが必要ですが、この長方形に RAMB16、RAMB32、DSP48 のブロック タイプが含まれる場合は、そのリソースも含まれます。これで、これらのブロック タイプの配線リソースが、リコンフィギャラブル領域に含まれます。[Pblock Properties] の [General] タブは、必要な場合にこれらのブロック タイプを追加するために使用できます。[Statistics] タブには、現在読み込まれている RM のリソース要件が表示されます。

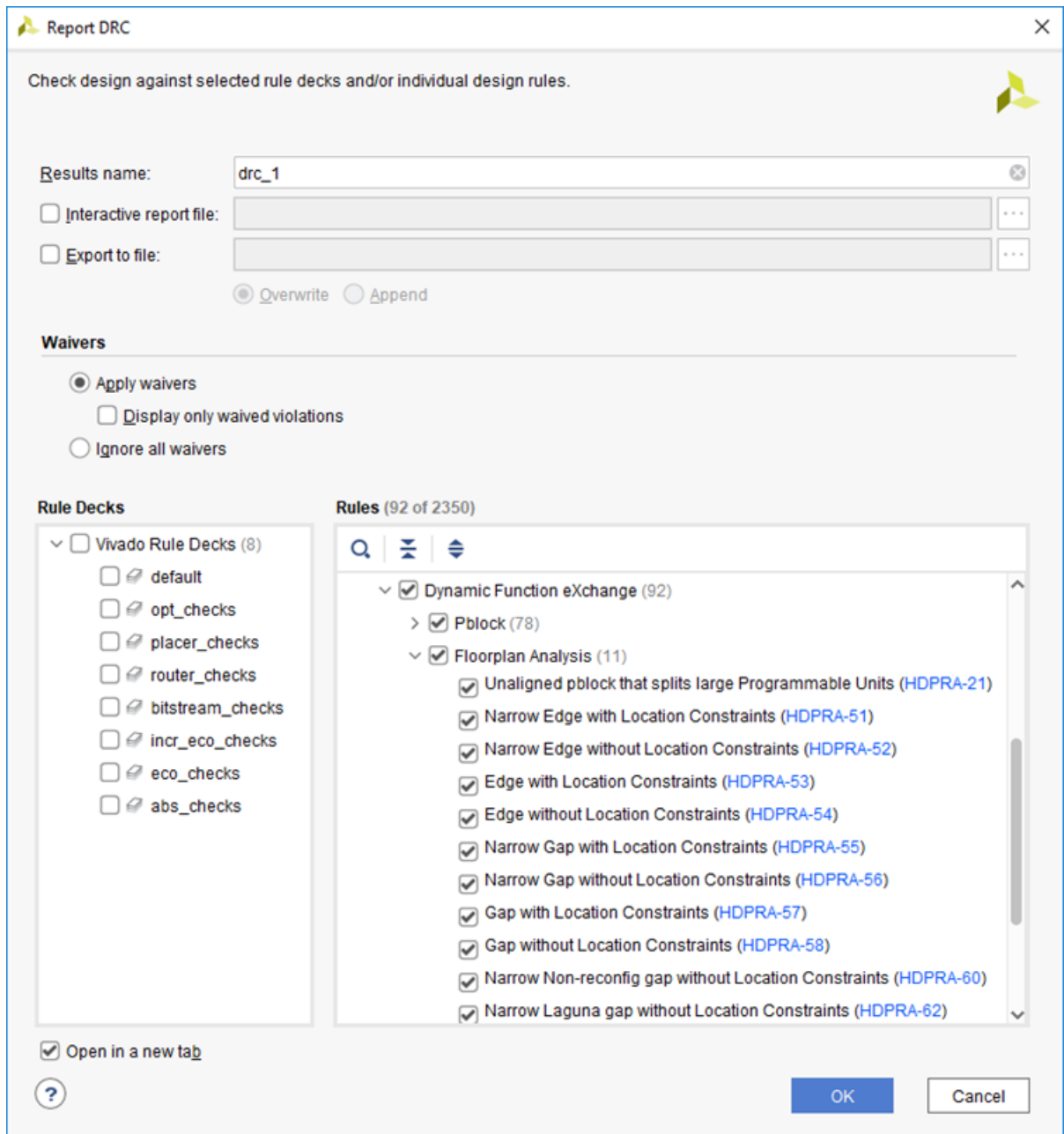
2. [Properties] で、[RESET\_AFTER\_RECONFIG] のチェック ボックスをオンにして、リコンフィギュレーションが完了した後に、このモジュールのロジックのみを初期化します。
3. inst\_shift インスタンスに手順 1 および 2 を繰り返しますが、今回は、クロック領域 X1Y1 の右側をターゲットにします。この RM にはブロック RAM インスタンスが含まれているので、そのリソース タイプを含める必要があります。そのリソース タイプが含まれていない場合は、[Statistics] タブで RAMB の詳細が赤く表示されます。

図 2: inst\_shift リコンフィギャラブルパーティションの Pblock



4. [Reports] → [Report DRC] をクリックして DFX のデザイン ルール チェックを実行します。[All Rules] をオフにしてから、[Dynamic Function eXchange] をオンにして、DFX DRC のみのレポートを作成します。

図 3: DFX デザイン ルール チェック (DRC)

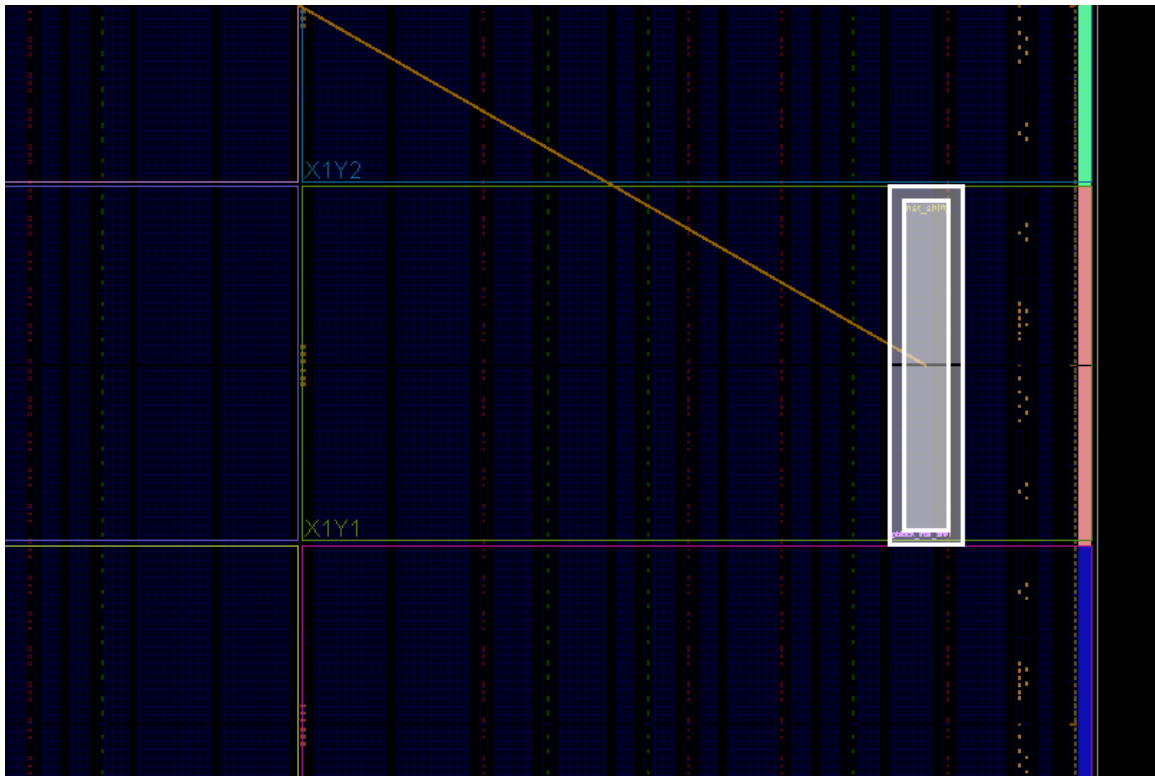


この時点で、DRC が 1 つまたは 2 つレポートされますが、その解決方法は 2 つあります。この演習では、inst\_shift に 1 つの方法、inst\_count にもう 1 つの方法を利用します。

最初の DRC は HDPRA-10 というエラーで、RESET\_AFTER\_RECONFIG に Pblock のフレーム アライメントが必要であることを知らせています。

5. この最初の DRC エラーを解決するには、クロック領域の境界に Pblock の高さを揃えるようにします。次の図のように、inst\_shift の Pblock を引き伸ばして、上下の辺を X1Y1 のクロック領域の境界線に揃えます。Pblock の中の塗りつぶしがこれで均等になりました。

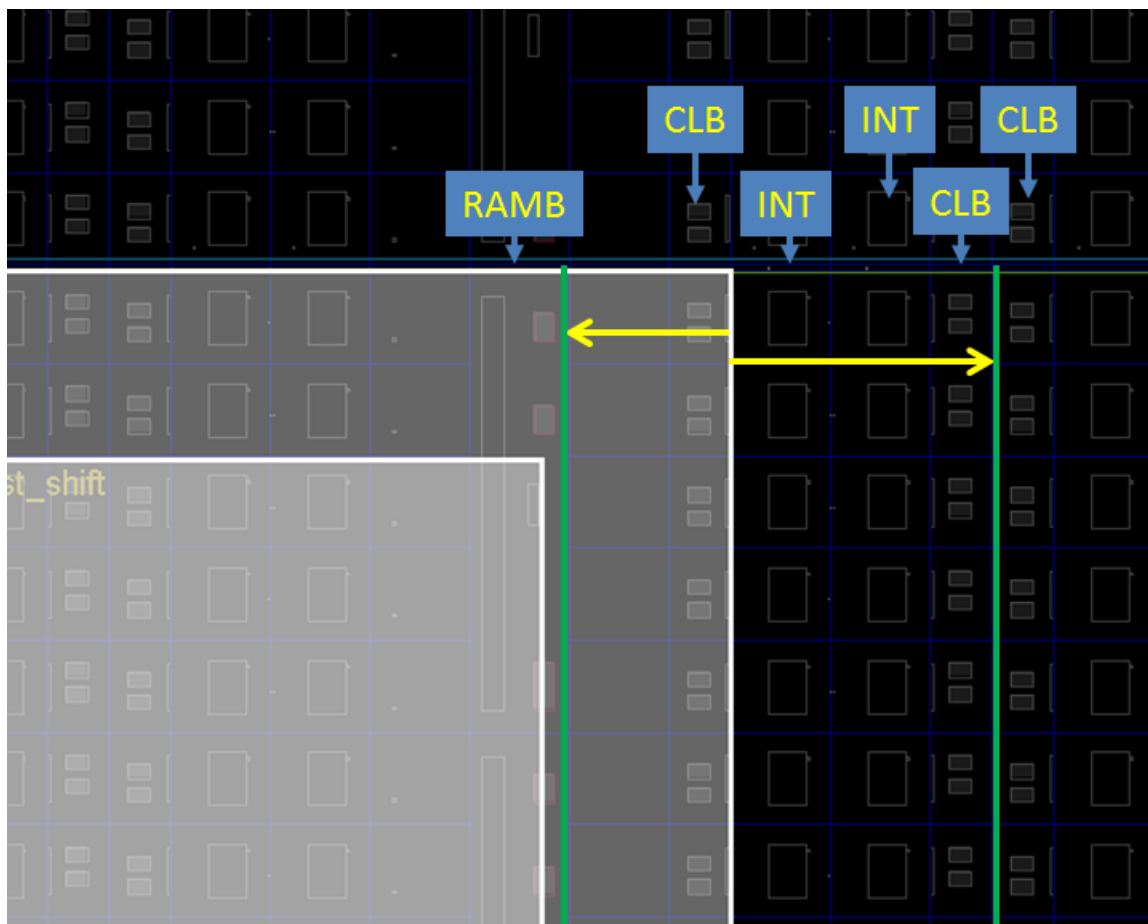
図 4: 揃えられた inst\_shift リコンフィギュラブル パーティションの Pblock



もう 1 つの DRC は HDPR-26 という警告で、リコンフィギュラブル Pblock の左辺または右辺が間違った境界線で終わっていることを知らせています。左辺または右辺が、インターコネクト (INT) 列を分割するようなことがあってはいけません。この要件の詳細は、『Vivado Design Suite ユーザー ガイド: Dynamic Function eXchange』(UG909) のリコンフィギュラブル パーティションの Pblock のサイズおよび形状に関するセクションを参照してください。

6. この DRC 警告が表示されないようにするには、inst\_shift (または inst\_shift で問題がレポートされていない場合は inst\_count) でレポートされている上辺または下辺の角を拡大表示し、違反が発生していないか確認します。図 5 の黄色い矢印で示すように、この辺を 1 列左へまたは右へ移動させ、CLB と INT の間または BRAM と INT の間ではなく、2 つのリソース タイプ (例: CLB と CLB、または CLB と RAMB) の間に配置します。

図 5: リコンフィギャラブル Pblock の辺の調整



7. PR DRC をもう一度実行し、対処したエラーおよび警告が inst\_shift インスタンスで解決されていることを確認します。

リコンフィギャラブル Pblock のサイズおよび形状を手動で調節しない場合は、SNAPPING\_MODE 機能を使用します。この機能は、有効な境界線に辺を自動的に揃えます。RESET\_AFTER\_RECONFIG 機能が選択されている場合は、クロック領域の境界線に揃えるので、Pblock は縦に長細くなります。必要に応じて左辺または右辺を調整すると、Pblock の幅が狭くなります。SNAPPING\_MODE により Pblock が変更されると、使用可能なリソースの数やタイプが変わります。

8. [Device] ウィンドウで inst\_count の Pblock を選択し、[Pblock Properties] の [Properties] で SNAPPING\_MODE の値を OFF から ROUTING (または ON) に変更します。

もとの Pblock は変更されませんが、その背後の塗りつぶしは変わります。DFX ルールに準拠するよう、ソースの制約を変更せずに Pblock は自動的に調整されます。

9. もう一度 DFX DRC を実行して、エラーがすべて解決されていることを確認します。Pblock がデバイスの端のほうに配置されている場合は特に、アドバイザリ メッセージが表示される可能性があります。
10. Pblock および関連付けられているプロパティを保存します。

```
write_xdc ./Sources/xdc/top_all.xdc
```

これで、top\_io\_\$board.xdc から先にインポートしてあった制約も含め、デザインの現在の制約がすべてエクスポートされます。これらの制約は、XDC ファイルで管理するか、または run スクリプトで管理できます (通常は HD.RECONFIGURABLE を使用)。

または、Pblock 制約そのものを抽出して別に管理できます。このタスクを実行しやすくするための Tcl プロシージャがあります。

- a. まず、Tcl ユーティリティ ファイルの 1 つにある、次のプロシージャを実行します。

```
source ./Tcl_HD/hd_utils.tcl
```

- b. 次に、`export_pblocks proc` を使用して、この制約情報を出力します。

```
export_pblocks -file ./Sources/xdc/pblocks.xdc
```

これで、デザインの両方の Pblock の制約情報が出力されます。いずれか一方のみを選択する場合は、`-pblocks` オプションを使用します。

これでフロアプランが確立しました。次は、デザインをインプリメントします。

## 手順 6: 初回コンフィギュレーションのインプリメント

この手順では、デザインを配置配線し、新しい RM で再利用するため、デザインのスタティック部分を準備します。

### デザインのインプリメンテーション

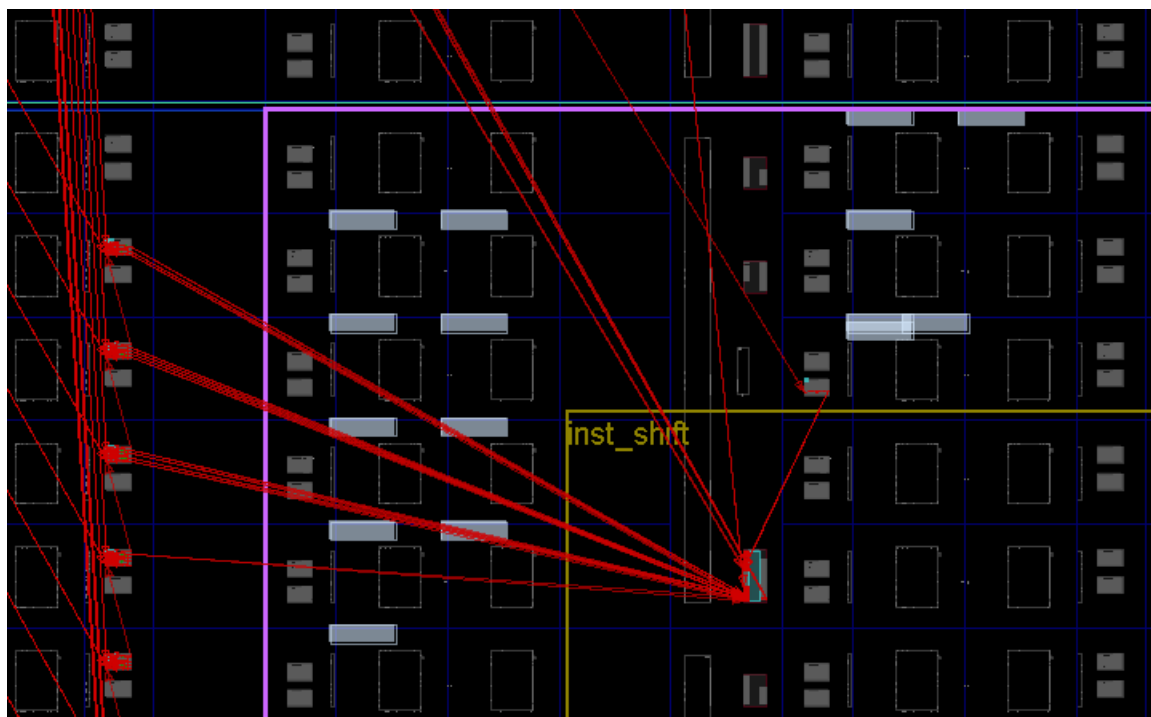
1. 次のコマンドを実行し、デザインを最適化し、配置配線します。

```
opt_design
place_design
route_design
```

`place_design` および `route_design` の両方が完了したら、次の図のように [Device] ビューでデザインのステートを確認します。`place_design` 実行後に注意すべき点は、パーティション ピンの導入です。これらのピンは、スタティック ロジックとリコンフィギュラブル ロジックとの間をつなぐ物理的なインターフェイス ポイントで、RM の各 I/O が配線されるインターコネクト タイル内のアンカーポイントでもあります。これらのピンは、配置済みデザインのビューで白いボックスとして表示されます。

`pblock_shift` の場合、スタティック部分への接続がデバイスの該当 Pblock のすぐ外側にあるので、パーティション ピンは Pblock の上辺近くに表示されます。`pblock_count` の場合は、`SNAPPING_MODE` が RP に追加すべきフレームを垂直方向に集めるので、パーティション ピンはユーザー定義領域の外側に表示されます。

図 6: 配置済みデザイン内のパーティション ピン



2. GUI で、これらのパーティション ピンを簡単に検索するには、次の作業を実行します。
  - a. RM (たとえば inst\_shift) を [Netlist] で選択します。
  - b. [Cell Properties] で [Cell Pins] タブをクリックします。
3. 任意のピンを選択してハイライトさせます。または、[Ctrl+A] キーを押して、すべてのピンを選択します。Tcl ですべてのピンを選択する場合は、次のコマンドを使用します。

```
select_objects [get_pins inst_shift/*]
```


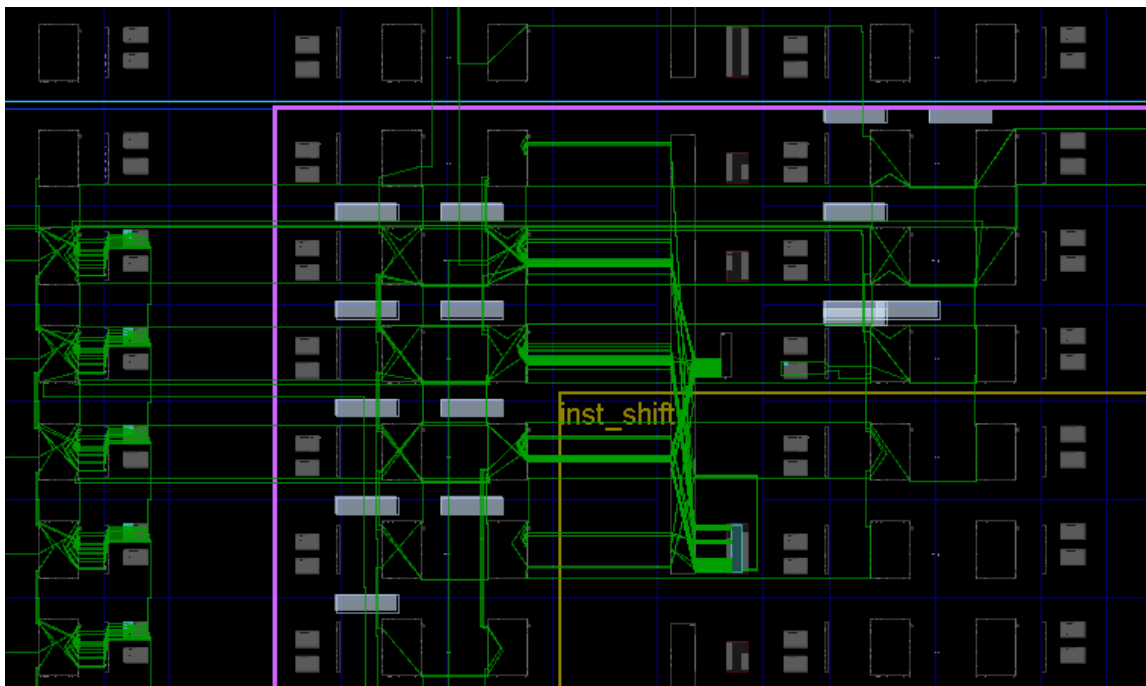
4. 抽象化された配線情報と実際の配線を切り替え、また、配線リソース自体の表示を変更するには、ツールバーの [Routing Resources] ボタン  を使用します。デザインのすべてのネットがこの時点で完全に配線されています。

図 7: 配線された初回コンフィギュレーションの拡大表示



### 結果の保存

1. 次のコマンドを使用して、フル デザイン チェックポイントを保存し、レポート ファイルを生成します。

```
write_checkpoint -force Implement/Config_shift_right_count_up_implement/
top_route_design.dcp

report_utilization -file Implement/Config_shift_right_count_up_implement/
top_utilization.rpt

report_timing_summary -file Implement/
Config_shift_right_count_up_implement/top_timing_summary.rpt
```

2. [オプション] 次の 2 つのコマンドを使用し、各 RM のチェックポイントを保存します。

```
write_checkpoint -force -cell inst_shift Checkpoint/
shift_right_route_design.dcp

write_checkpoint -force -cell inst_count Checkpoint/
count_up_route_design.dcp
```



**ヒント:** パッチ モードでデザイン全体を処理するため `run_dfx.tcl` を実行する場合、デザイン チェックポイント、ログ ファイル、レポート ファイルはフローの各ステップで作成されます。

この時点で、完全にインプリメントされた Dynamic Function eXchange デザインを作成したので、ここからフルおよびパッチのビットストリームを生成できます。このコンフィギュレーションのスタティック部分は、この後に作成するコンフィギュレーションすべてに使用されます。このスタティック部分を隔離するには、現在の RM を削除します。配線リソースがイネーブルになっていることを確認し、パーティション ピンのあるインターコネクト タイルを拡大表示します。



3. 次のコマンドを使用して、RM ロジックをクリアにします。

```
update_design -cell inst_shift -black_box
update_design -cell inst_count -black_box
```

これらのコマンドを使用すると、次の図のようにデザインで多数の変更が発生します。

- 完全配線されたネット (緑色) の数が減少します。
- inst\_shift および inst\_count は、[Netlist] ビューで空の状態が表示されます。

図 8: update\_design -black\_box 実行前 (上) の inst\_shift モジュール

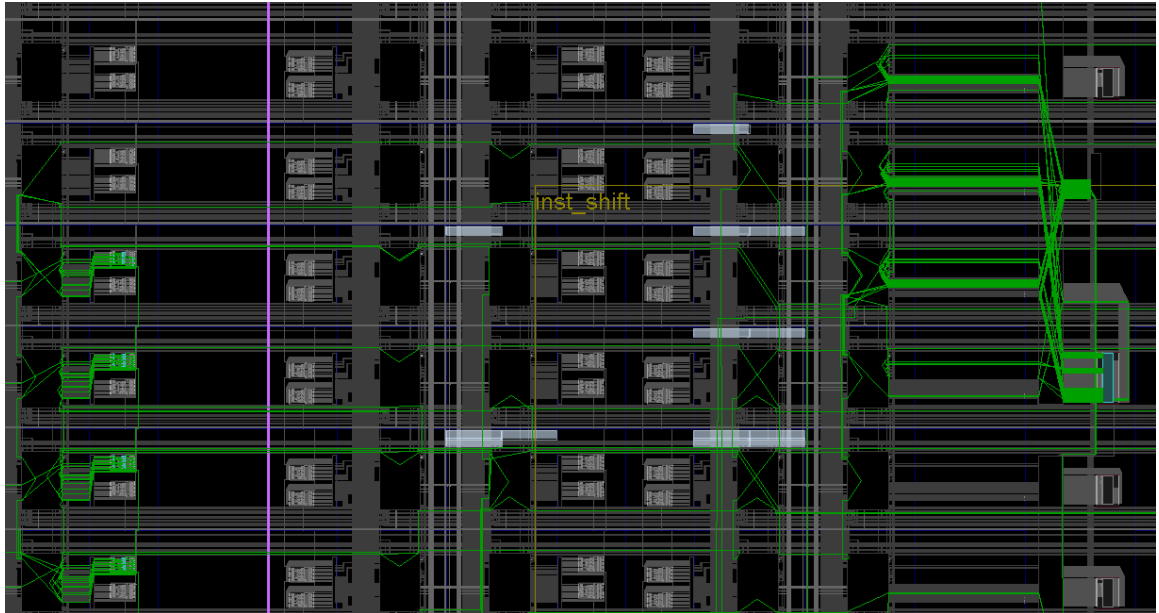
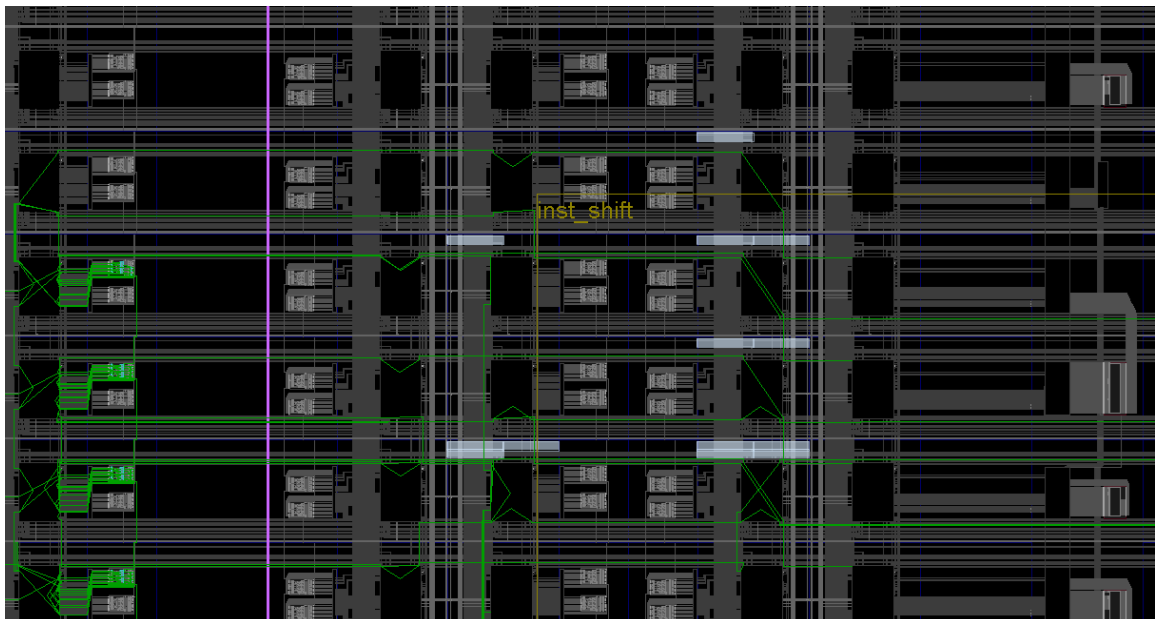


図 9: update\_design -black\_box 実行後 (下) の inst\_shift モジュール

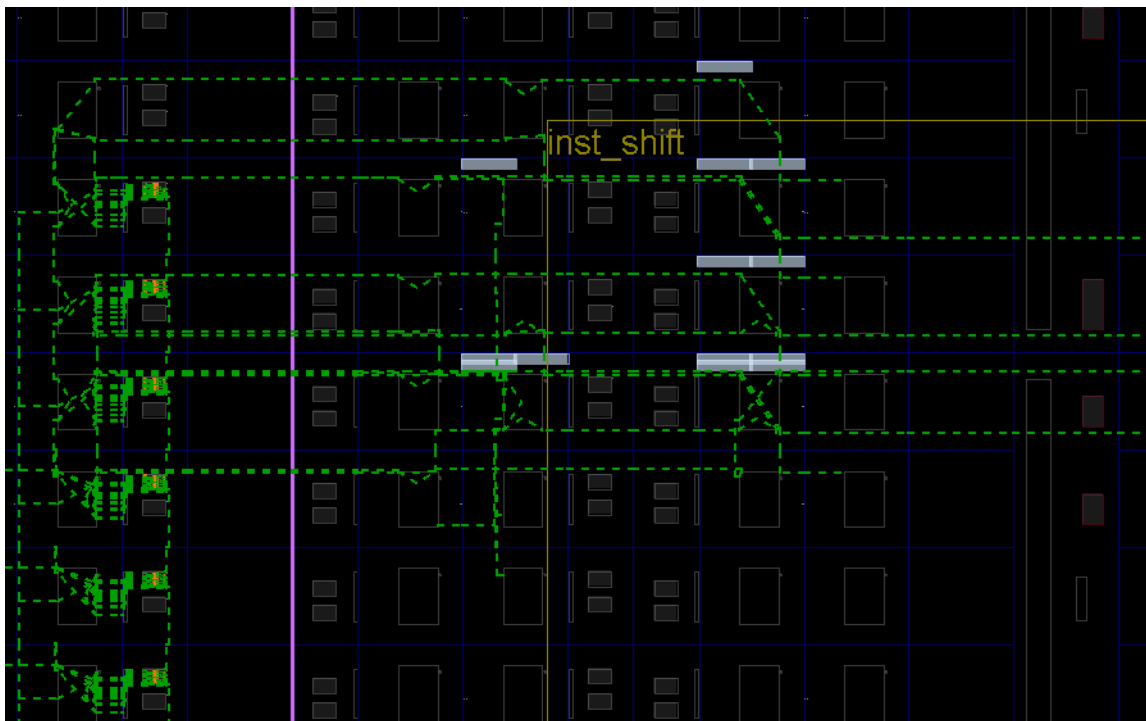


- すべての配置配線をロックするには、次のコマンドを使用します。

```
lock_design -level routing
```

lock\_design コマンドでセルが特定されていないので、メモリのデザイン全体 (現在はブラック ボックスになっているスタティック デザインで構成) に影響します。配線済みネットはすべてロックされた状態で表示されます (次の図の点線部分)。配置済みコンポーネントもロックされていることを示すため、すべて 青色からオレンジ色 に表示が変わります。

図 10: ロックされた配線を示すスタティック専用デザインの拡大表示



- 残りのスタティック専用チェックポイントを出力するには、次のコマンドを使用します。

```
write_checkpoint -force Checkpoint/static_route_design.dcp
```

このスタティック専用チェックポイントはこれ以降のコンフィギュレーションに使用されます。

- 次のコンフィギュレーションに進む前にこのデザインを閉じます。

```
close_project
```

## 手順 7: 2 回目のコンフィギュレーションのインプリメント

スタティック デザインの結果が確立されロックされたので、これを参考にしながら、さらに RM をインプリメントできます。

## デザインのインプリメンテーション

1. Tcl コンソールで次のコマンドを実行し、新しくインメモリ デザインを作成します。

```
create_project -in_memory -part $part
```

2. 次のコマンドを実行し、スタティック デザインを読み込みます。

```
add_files ./Checkpoint/static_route_design.dcp
```

3. 次のコマンドを実行し、シフトおよびカウント ファンクションの 2 回目の合成チェックポイントを 2 つ読み込みます。

```
add_files ./Synth/shift_left/shift_synth.dcp
```

```
set_property SCOPED_TO_CELLS {inst_shift} [get_files ./Synth/shift_left/shift_synth.dcp]
```

```
add_files ./Synth/count_down/count_synth.dcp
```

```
set_property SCOPED_TO_CELLS {inst_count} [get_files ./Synth/count_down/count_synth.dcp]
```

4. link\_design コマンドを使用してデザイン全体をリンクします。

```
link_design -mode default -reconfig_partitions {inst_shift inst_count} -part $part -top top
```

この時点で、フル コンフィギュレーションが読み込まれます。ただし今回は、スタティック デザインが配線されロックされているので、リコンフィギュラブル ロジックはまだネットリストの状態でしかありません。ここからの配置配線はこのロジックにのみ適用されます。

5. 次のコマンドを実行し、スタティック デザインの初回コンフィギュレーションを参考に新しい RM を配置配線します。

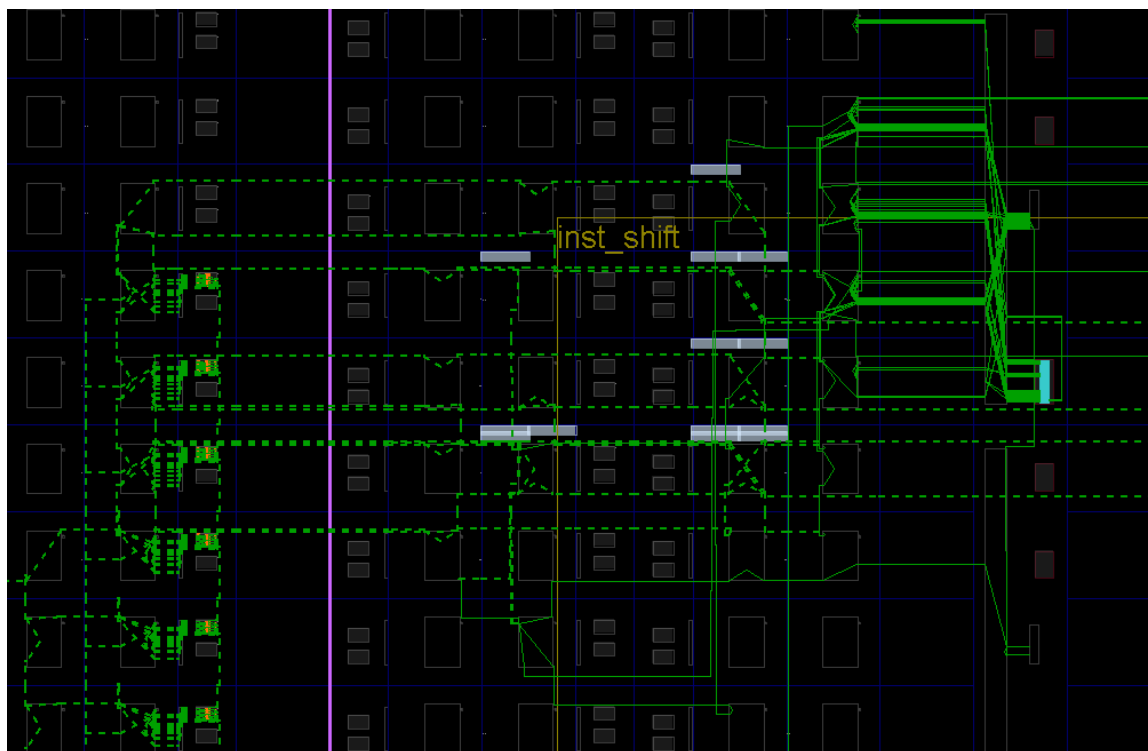
```
opt_design
```

```
place_design
```

```
route_design
```

デザインは再びフルにインプリメントされますが、今回は、この新しい RM のバリエーションが使用されます。次の図のように、点線 (ロックされた配線) および実線 (新しい配線) の配線セグメントが入り混じっています。

図 11: 2 回目のコンフィギュレーション (ロックされた配線および新しい配線の両方表示)



### 結果の保存

1. 次のコマンドを使用して、フル デザイン チェックポイントおよびレポート ファイルを保存します。

```
write_checkpoint -force Implement/Config_shift_left_count_down_import/
top_route_design.dcp

report_utilization -file Implement/Config_shift_left_count_down_import/
top_utilization.rpt

report_timing_summary -file Implement/
Config_shift_left_count_down_import/top_timing_summary.rpt
```

2. [オプション] 次の 2 つのコマンドを使用し、各 RM のチェックポイントを保存します。

```
write_checkpoint -force -cell inst_shift Checkpoint/
shift_left_route_design.dcp

write_checkpoint -force -cell inst_count Checkpoint/
count_down_route_design.dcp
```

これで、スタティック デザインおよびすべての RM のバリエーションのインプリメンテーションを完了しました。RP に RM が 3 つ以上あるデザインの場合は、このプロセスを繰り返します。

## 手順 8: スクリプトをハイライトして結果を確認

配線済みのコンフィギュレーションを IDE で開いた状態で、タイルやネットをハイライトするため可視化スクリプトをいくつか実行します。これらのスクリプトは、Dynamic Function eXchange 用に割り当てられているリソースを特定し、また自動的に生成されます。

1. Tcl コンソールで、<Extract\_Dir> ディレクトリから次のコマンドを実行します。

```
source hd_visual/pblock_inst_shift_AllTiles.tcl

highlight_objects -color blue [get_selected_objects]
```

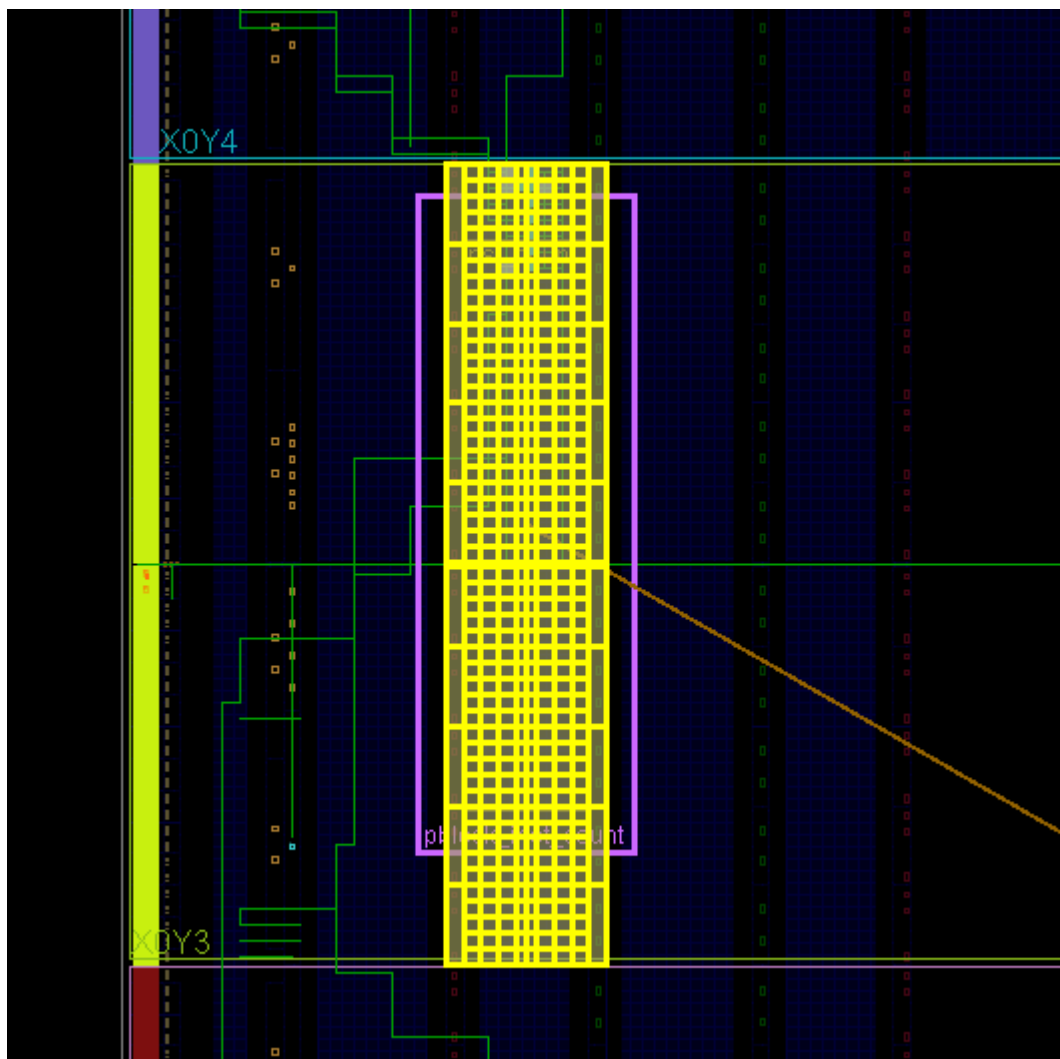
2. フレームの選択を解除するため、[Device] ビューのどこかでクリックし (または 「unselect\_objects」と入力)、次のコマンドを実行します。

```
source hd_visual/pblock_inst_count_AllTiles.tcl

highlight_objects -color yellow [get_selected_objects]
```

次の図のように、[Device] ビューでパーティション フレームがハイライトされた状態で表示されます。

図 12: ハイライトされた RP フレーム



ここでハイライトされているタイルは、パースナル ビットストリームを作成するため、ビットストリーム生成に送信されるコンフィギュレーション フレームを表しています。上に示すように、SNAPPING\_MODE 機能により、RESET\_AFTER\_RECONFIG および有効な RP 幅用に、pblock\_count の 4 つのエッジすべてが調整されます。

これ以外の「タイル」スクリプトもこのバリエーションです。クロック領域の境界線に垂直に揃えた Pblock を作成していない場合は、FrameTiles スクリプトにより明示的な Pblock タイルがハイライトされ、また AllTiles スクリプトにより、リコンフィギュラブル フレームの高さいっぱいこれらのタイルが引き伸ばされます。選択されていないフレーム タイプ (グローバル クロックなど) がある場合は、そこにギャップが残るので注意してください。

GlitchTiles スクリプトはフレーム サイトのサブセットなので、専用シリコン リソースを避けます。それ以外のスクリプトからは、さらに多くの情報が提供されます。

3. 現在のデザインを閉じます。

```
close_project
```

## 手順 9: ビットストリームの生成

### コンフィギュレーションの検証



**推奨:** ビットストリームを生成する前に、すべてのコンフィギュレーションを検証し、各コンフィギュレーションのスタティック部分が同じであり、生成したビットストリームがシリコンで使用しても安全であることを確認します。PR の検証機能は、パーティション ピンを含め、それらが同じであることを確認するところまで、スタティック デザインを検証します。RM 内の配置配線は、ここでは異なるモジュール結果が予想されるため、オンになっていません。

1. Tcl コンソールで `pr_verify` コマンドを実行します。

```
pr_verify Implement/Config_shift_right_count_up_implement/  
top_route_design.dcp Implement/Config_shift_left_count_down_import/  
top_route_design.dcp
```

上記のコマンドで問題が発生しなければ、次のようなメッセージが表示されます。

```
INFO: [Vivado 12-3253] PR_VERIFY: check points Implement/  
Config_shift_right_count_up/  
top_route_design.dcp and Implement/Config_shift_left_count_down/  
top_route_design.dcp are compatible
```

デフォルトでは、(不一致がある場合は) 最初の不一致のみがレポートされます。すべての不一致を表示させるには、`-full_check` オプションを使用します。

2. プロジェクトを閉じます。

```
close_project
```

### ビットストリームの生成

コンフィギュレーションを検証できたので、ビットストリームを生成し、それを使用して選択したデモ ボードをターゲットにします。

**注記:** 1 回目のコンフィギュレーションでは `shift_right` および `count_up` がインプリメントされます。2 回目のコンフィギュレーションでは `shift_left` および `count_down` がインプリメントされます。

1. まず、初回コンフィギュレーションをメモリに読み込みます。

```
open_checkpoint Implement/Config_shift_right_count_up_implement/  
top_route_design.dcp
```

2. このデザインのフルおよびパースシャル ビットストリームを生成します。ビットストリームが生成されたフル デザイン チェックポイントに関連したディレクトリ (重複しない場所) にビット ファイルを保存してください。

```
write_bitstream -force -file Bitstreams/Config_RightUp.bit
```

```
close_project
```

3 つのビットストリームが生成された点に注目してください。

- `Config_RightUp.bit`

パワーアップ、フル デザイン ビットストリームです。右の 4 つのシフト LED は右にシフトし、左の 4 つのカウント LED はカウントアップします。

- `Config_RightUp_Pblock_inst_shift_partial.bit`

shift\_right モジュールのパーシャル ビット ファイルです。

- Config\_RightUp\_Pblock\_inst\_count\_partial.bit

カウント LED にカウントアップさせる count\_up モジュールのパーシャル ビット ファイルです。



**重要:** write\_bitstream を 1 回呼び出してビット ファイルを生成する場合、ファイルの名前に RM のバリエーションの名前が反映されず、どのイメージが読み込まれるのかをはっきりさせることができません。この解決策として、-file オプションを使用して、リコンフィギャラブル セルの Pblock 名に基本名を付けます。リコンフィギャラブル ビット ファイルをはっきりと識別できるよう、わかりやすい基本名を付けることが大切です。すべてのパーシャル ビット ファイルには \_partial という接尾辞が付きます。

ビットストリーム生成までデザイン全体の処理に run\_dfx.tcl を使用する場合は、異なるビットストリーム生成テクニックが使用されます。配線済みのデザイン チェックポイントを開くと、write\_bitstream が複数回呼び出されるので、ビットストリームに名前が付けやすくなり、また、フル/パーシャルのビットストリームに異なるオプション(ビットストリームの圧縮など)を適用できます。たとえば、advanced\_settings.tcl スクリプトで設定される名前は次のようになります。

- Config\_shift\_right\_count\_up\_implement\_full.bit

パワーアップ、フル デザイン ビットストリームです。

- pblock\_shift\_shift\_right\_partial.bit

shift\_right モジュールのパーシャル ビット ファイルです。

- pblock\_count\_count\_up\_partial.bit

count\_up モジュールのパーシャル ビット ファイルです。

- 2 回目のコンフィギュレーションのフルおよびパーシャル ビットストリームを生成します。今回も生成されたビット ファイルを該当フォルダーに保存します。

```
open_checkpoint Implement/Config_shift_left_count_down_import/
top_route_design.dcp
write_bitstream -force -file Bitstreams/Config_LeftDown.bit
close_project
```

同様に、3 つのビットストリームが生成されますが、今回は基本名が異なります。

- グレー ボックスを使用してフル ビットストリームを生成し、さらに RM 用にブランキング ビットストリームも生成します。ブランキング ビットストリームは、消費電力を低減する目的で、既存のコンフィギュレーションを「消去」するために使用できます。

```
open_checkpoint Checkpoint/static_route_design.dcp
update_design -cell inst_count -buffer_ports
update_design -cell inst_shift -buffer_ports
place_design
route_design
write_checkpoint -force Checkpoint/Config_greybox.dcp
write_bitstream -force -file Bitstreams/config_greybox.bit
close_project
```

基本になっているコンフィギュレーション ビットストリームには、どちらの RP のロジックもありません。

update\_design コマンドにより、RP のすべての出力の定数ドライバー (グラウンド) が挿入されるので、これらの出力はフロートしません。グレー ボックス (grey box) という用語は、これらの LUT が挿入されているので完全には空ではないことを示していて、この領域を出入りするネットが未接続になっているブラック ボックスとは異なります。place\_design および route\_design コマンドは、完全にインプリメントされるようにします。



## 手順 10: FPGA のパーシャル リコンフィギュレーション

`count_shift_led` デザインは、4 つのデモ ボードの 1 つをターゲットにします。現在のデザインでは、KC705、VC707、VC709 および AC701 ボード、Rev 1.0 および Rev 1.1 がサポートされています。

### フル イメージを使用したデバイスのコンフィギュレーション

1. プラットフォーム ケーブル USB を使用してコンピューターにボードを接続し、電源を投入します。
2. メインの Vivado® IDE で、[Flow] → [Open Hardware Manage] をクリックします。
3. 緑色のバナー上の [Open a new hardware target] をクリックします。ボードとの通信を確立するため、ウィザードの手順に従います。
4. 緑色のバナー上の [Program device] をクリックし、ターゲット デバイスを選択します。Bitstreams フォルダを参照し、[Config\_RightUp.bit] を選択してから、[OK] をクリックしてデバイスをプログラムします。

これで 2 つのタスクを実行している GPIO LED のバンクを確認できるはずです。4 つの LED がカウントアップを実行し (MSB は左側)、別の 4 つの LED が右方向ヘシフトしています。フル デバイスのコンフィギュレーションにかかる時間に注目してください。

**注記:** AC701 デモ ボードには、4 ビットの LED バンクしかありません。したがって、このデザインにはシフトまたはカウントの一方の機能しか表示されません。一度に両方を表示することはできません。シフトとカウントを切り替えるには、GPIO DIP スイッチ (SW2) でスイッチ 1 をトグルします。

### デバイスのパーシャル リコンフィギュレーション

これまでに作成したパーシャル ビットストリームのいずれかを使用して、デバイスをパーシャル リコンフィギュレーションする準備が整いました。

1. 緑色のバナー上の [Program device] をもう一度クリックします。Bitstreams フォルダを参照し、[Config\_LeftDown\_pblock\_inst\_shift\_partial.bit] を選択してから、[OK] をクリックしてデバイスをプログラムします。

LED のシフト部分の方向が変わりますが、カウンタはリコンフィギュレーションの影響を受けず、引き続きカウントアップします。今回のコンフィギュレーションには前回ほど時間がかからなかった点に注目してください。

2. 緑色のバナー上の [Program device] をもう一度クリックします。Bitstreams フォルダを参照し、[Config\_LeftDown\_pblock\_inst\_count\_partial.bit] を選択してから、[OK] をクリックしてデバイスをプログラムします。

カウンタはカウントダウンし始めますが、シフト用の LED はリコンフィギュレーションの影響を受けません。このプロセスは、元のコンフィギュレーションに戻るまで `Config_RightUp` パーシャル ビット ファイルを使用して繰り返すことができます。または、ブランキング (グレー ボックス) のパーシャル ビット ファイルを使用して、(点灯したままの) LED のアクティビティを 停止するまで繰り返すことができます。

## まとめ

これで演習 1 は終了です。この演習では、次の作業を実行しました。

- Dynamic Function eXchange のインプリメンテーションを準備するため、ボトムアップでデザインを合成しました。

- Dynamic Function eXchange デザインの有効なフロアプランを作成しました。
- 共通のスタティック結果を使用し、2 つのコンフィギュレーションを作成しました。
- この 2 つのコンフィギュレーションをインプリメントし、それぞれでスタティック デザインを使用できるよう保存しました。
- 後で再利用するため、スタティック モジュールおよび RM のチェックポイントを作成しました。
- フレームセットを確認し、この 2 つのコンフィギュレーションを検証しました。
- フルおよびパーシャル ビットストリームを作成しました。
- FPGA をコンフィギュレーションしてから、パーシャル リコンフィギュレーションしました。

## 演習 2

# UltraScale の基本 DFX フロー

この演習では、UltraScale™ および UltraScale+™ デバイスの基本的な Dynamic Function eXchange (DFX) フローについて説明します。まずスクリプトを使用して、スタティック モジュールおよび各リコンフィギュラブル モジュール (RM) のバリエーションを個別に合成します。それから IDE で、RM の位置を Pblock を使用して制約し、デザインの初回コンフィギュレーションをインプリメントします。次に、デザインのスタティック部分をロックし、RM を少しずつ変えながらインプリメンテーションを再実行して、さまざまなコンフィギュレーションをインプリメントします。最後に、各インプリメント済みの RM がデザインのスタティック部分と互換性があることを検証し、互換性があればビットストリームを生成します。

## 手順 1: チュートリアル デザイン ファイルの抽出

1. ザイリンクス ウェブサイトから[リファレンス デザイン ファイル](#)をダウンロードします。
2. ZIP ファイルの内容を書き込み可能なディレクトリに抽出します。
3. `\led_shift_count_us` に移動します。

## 手順 2: スクリプトの確認

まずは、デザイン アーカイブにあるスクリプトを確認します。`run_dfx.tcl` および `advanced_settings.tcl` ファイルは、ルート ディレクトリにあります。`run_dfx.tcl` スクリプトには、Dynamic Function eXchange の実行に必要な最小限の設定が含まれています。`advanced_settings.tcl` には、デフォルト フローの設定が含まれているので、上級ユーザーでない限り、このファイルは変更しないでください。

### メイン スクリプト

`\led_shift_count_us` にある `run_dfx.tcl` をテキスト エディターで開きます。これは、デザイン パラメーター、デザイン ソース、デザイン 構造を定義するマスター スクリプトです。Dynamic Function eXchange デザイン全体をコンパイルするには、このファイルのみを変更します。`run_dfx.tcl`、`advanced_settings.tcl` などの基本スクリプトの詳細は、`Tcl_HD` サブディレクトリにある `README.txt` を参照してください。

`run_dfx.tcl` の詳細は次のとおりです。

- [Define target demo board] では、このデザインでサポートされているデモ ボードの中から 1 つボードを選択できます。
- [flow control] では、実行する合成およびインプリメンテーションのフェーズを制御できます。このチュートリアルでは、合成のみスクリプトで実行できます。インプリメンテーション、検証、ビットストリーム生成はインタラクティブに実行されます。これらの追加ステップをスクリプトから実行するには、フロー変数 (`run.prImpl` など) を 1 に変更します。

- [Output Directories] および [Input Directories] では、デザイン ソースおよび結果ファイルに必要なファイル構造が設定されます。ファイル構造を変更する場合は、その変更をここに反映させる必要があります。
- [Top Definition] および [RP Module Definitions] セクションでは、デザインの各部分のソース ファイルをすべて参照できます。[Top Definition] は、スタティック デザインに必要なソース (制約および IP を含む) がすべて対象になっています。[RP Module Definitions] では、リコンフィギャラブル パーティション (RP) のソースが対象になっています。各 RP を特定し、各 RP のリコンフィギャラブル モジュール (RM) のバリエーションをすべてリストします。
  - このデザインには 2 つの RP (`inst_shift` および `inst_count`) があり、各 RP にモジュールのバリエーションが 2 つあります。
- 「Configuration Definition」 セクションでは、1 コンフィギュレーションを構成するスタティック モジュールおよび RM のセットを定義します。
  - このデザインには、`config_shift_right_count_up_implement` および `config_shift_left_count_down_import` のマスター スクリプト内で定義された 2 つのコンフィギュレーションがあります。
  - さらにコンフィギュレーションを作成する場合は、RM を追加するか、既存の RM をまとめます。

### サポート スクリプト

`Tcl_HD` サブディレクトリの下には、サポート Tcl スクリプトがいくつかあります。これらのスクリプトは `run_dfx.tcl` に呼び出され、Dynamic Function eXchange フローの詳細を管理します。主な DFX スクリプトについていくつか詳細を次に挙げます。



**注意:** サポート Tcl スクリプトは変更しないでください。

- `step.tcl`: チェックポイントを監視して、デザインの現在のステータスを管理します。
- `synthesize.tcl`: 合成フェーズの詳細をすべて管理します。
- `implement.tcl`: モジュール インプリメンテーション フェーズの詳細をすべて管理します。
- `dfx_utils.tcl`: DFX デザインの差上位インプリメンテーション フェーズの詳細をすべて管理します。
- `run.tcl`: 合成およびインプリメンテーションの実際の run を起動します。
- `log_utils.tcl`: フロー中のキー ポイントでレポート ファイルを生成します。

残りのスクリプトは、これらのスクリプト内の詳細を提供したり (ほかの `*_utils.tcl` スクリプトなど)、ほかの階層デザインフローを管理したりします (`hd_utils.tcl` など)。

## 手順 3: デザインの合成

`run_dfx.tcl` スクリプトは、このチュートリアル of 合成フェーズを自動化します。最上位のスタティック デザインに 1 回、4 つの RM それぞれに 1 回ずつ、合計 5 回、合成の繰り返し実行が呼び出されます。

1. Vivado Tcl シェルを開きます。
  - Windows で、ザイリンクス Vivado デスクトップ アイコンをクリックするか、または [Start] → [All Programs] → [Xilinx Design Tools] → [Vivado 2020.1] → [Vivado 2020.1 Tcl Shell] をクリックします。
  - Linux の場合は、「`vivado -mode tcl`」と入力するだけです。

2. シェルで `\led_shift_count_us` に移動します。
3. KC705 以外のデモ ボードをターゲットにする場合は、`run_dfx.tcl` で `xboard` 変数を変更します。  
この演習でほかに使用可能なボードは、VCU108、KCU116、VCU118 です。
4. 次のコマンドを入力して、`run_dfx.tcl` スクリプトを実行します。

```
source run_dfx.tcl -notrace
```

Vivado 合成で 5 回の繰り返し実行をすべて完了しても、Vivado Tcl シェルは開いたままです。各モジュールのログやレポート ファイル、最終チェックポイントは、Synth サブディレクトリに各モジュール名の付いたフォルダがあって、その中に生成されています。



**ヒント:** `\led_shift_count_us` ディレクトリには、複数のログ ファイルが作成されています。

- `run.log` には、Tcl シェルのウィンドウに表示されたサマリが書き込まれています。
- `command.log` には、スクリプトにより実行された個々のステップすべてが書き込まれています。
- `critical.log` には、実行中に出力されたクリティカル警告メッセージがすべて書き込まれています。

## 手順 4: デザインのまとめおよびインプリメンテーション

各モジュールおよび最上位の合成済みチェックポイントが生成されたので、デザインをまとめる準備が整いました。

すべてのフロー ステップは Tcl コンソールから実行されますが、インタラクティブなイベントに対しては IDE 内の機能 (フロアプランニング ツールなど) を使用できます。



**ヒント:** Vivado IDE でのミス タイプを避けるため、このチュートリアルから直接コマンドをコピーして貼り付けます。コマンドは 1 つずつコピーして貼り付けてください。中には、複数行にわたる長いコマンドもあるので注意してください。

1. Vivado® IDE を開きます。開いている Tcl シェルに「`start_gui`」と入力して Vivado® IDE を開くか、Vivado® の「`-mode gui`」コマンドを実行して Vivado を起動します。
2. 現在のディレクトリが `\led_shift_count_us` でない場合は、そこへ移動します。pwd コマンドを使用すると、現在のディレクトリを確認できます。
3. 変数を設定します。このチュートリアルからコマンドをコピーして Tcl コンソールに貼り付けるときに役立ちます。この演習でターゲットにしているパーツおよびボードを選択し、Vivado でそれを適用します。

```
set part "xc7k325t-ffg900-2"
set board "kc705"

set part "xc7vx485t-ffg1761-2"
set board "vc707"

set part "xc7vx690t-ffg1761-2"
set board "vc709"

set part "xc7a200t-fbg676-2"
set board "ac701"
```

4. Tcl コンソールで次のコマンドを実行し、インメモリ デザインを作成します。

```
create_project -in_memory -part $part
```

5. 次のコマンドを実行し、スタティック デザインを読み込みます。

```
add_files ./Synth/Static/top-synth.dcp
```

6. 次のコマンドを実行し、最上位のデザイン制約を読み込みます。

```
add_files ./Sources/xdc/top_io_$board.xdc
set_property USED_IN {implementation} [get_files ./Sources/xdc/top_io_$board.xdc]
```

使用可能な XDC ファイルの top\_io\_\$board バージョンを選択すると、ピンの位置およびクロック制約が読み込まれますが、フロアプラン情報は読み込まれません。top\_\$board バージョンを読み込むと、ピンの位置、クロック制約、およびフロアプラン制約が読み込まれます。

7. 次のコマンドを実行し、シフトおよびカウント ファンクションの最初の合成チェックポイントを 2 つ読み込みます。

```
add_files ./Synth/shift_right/shift-synth.dcp
set_property SCOPED_TO_CELLS {inst-shift} [get_files ./Synth/shift_right/shift-synth.dcp]
add_files ./Synth/count_up/count-synth.dcp
set_property SCOPED_TO_CELLS {inst-count} [get_files ./Synth/count_up/count-synth.dcp]
```

SCOPED\_TO\_CELLS プロパティを使用すると、ターゲット セルに正しく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: 制約の使用』 (UG903) のこのセクションを参照してください。

8. link\_design コマンドを使用してデザイン全体をリンクします。

```
link_design -mode default -reconfig_partitions {inst-shift inst-count} -part $part -top top
```

この時点で、スタティック部分とリコンフィギュラブル ロジックを含んだフル コンフィギュレーションが読み込まれます。非プロジェクト モードで作業している間は、Flow Navigator は表示されないので注意してください。



**ヒント:** [Layout] → [Floorplanning] をクリックして、IDE をフロアプランニング モードにします。[Device] ビューが表示されていることを確認します。

9. この初回コンフィギュレーションでまとめられたデザインを保存します。

```
write_checkpoint ./Checkpoint/top-link-right-up.dcp
```

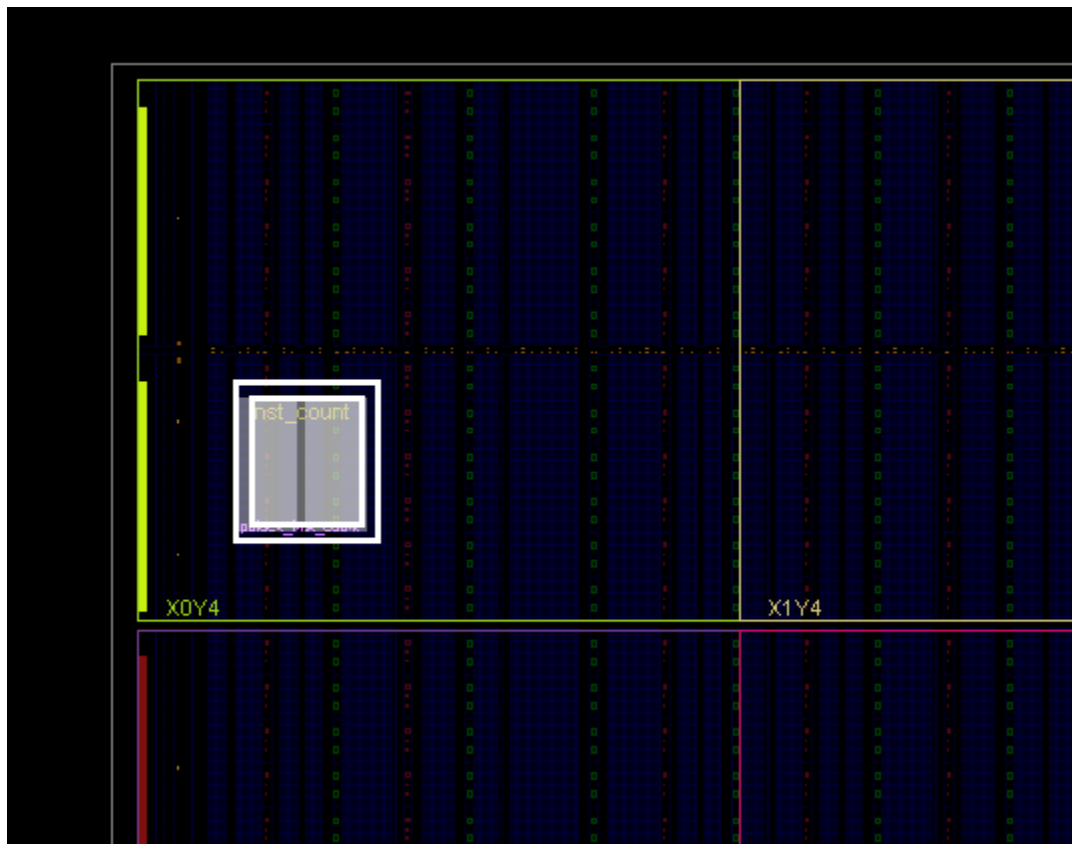
## 手順 5: デザイン フロアプランの構築

次に、Dynamic Function eXchange 用の領域を定義するため、フロアプランを作成します。

1. [Netlist] ウィンドウで [inst\_count] を選択します。右クリックして、[Floorplanning] → [Draw Pblock] をクリックし、デバイスの左上に細長い長方形を描画します。この段階では、長方形のサイズや形状は重要ではありませんが、クロック領域内に配置してください。

[Device] ビューで Pblock が選択されていることを確認し、作業を続けます。

図 13: inst\_count RP の Pblock



RM には CLB リソースのみが必要ですが、この長方形に RAMB16、RAMB32、DSP48 のブロック タイプが含まれる場合は、そのリソースも含まれます。これで、これらのブロック タイプの配線リソースが、リコンフィギュラブル領域に含まれます。[Pblock Properties] の [General] タブは、必要な場合にこれらのブロック タイプを追加するために使用できます。[Statistics] タブには、現在読み込まれている RM のリソース要件が表示されます。

2. inst\_shift インスタンスに対しても同じ手順を繰り返しますが、今回は最初にターゲットにしたクロック領域の下にある、別のクロック領域をターゲットにします。この RM にはブロック RAM インスタンスが含まれているので、そのリソース タイプを含める必要があります。そのリソース タイプが含まれていない場合は、[Statistics] タブで RAMB の詳細が赤く表示されます。
3. [Reports] → [Report DRC] をクリックして、Dynamic Function eXchange のデザイン ルール チェックを実行します。[All Rules] をオフにしてから、[Dynamic Function eXchange] をオンにして、DFX DRC のみのレポートを作成します。

inst\_shift Pblock に RAMB18 および RAMB36 リソースが含まれている限り、DRC はレポートされないはずです。Pblock がデバイスの端のほうに配置されている場合は特に、アドバイザリ メッセージが表示される可能性があります。両方の Pblock に対し、[Pblock Properties] の [Properties] タブでレポートされているように、SNAPPING\_MODE が ON に設定されている点に注目してください。このアーキテクチャのプログラマブル ユニットの精度を考慮し、すべての UltraScale および UltraScale+ デバイスで、この SNAPPING\_MODE は常に有効になっています。

4. Pblock および関連付けられているプロパティを保存します。

```
write_xdc ./Sources/xdc/top_all.xdc
```



これで、`top_io_$board.xdc` から先にインポートしてあった制約も含め、デザインの現在の制約がすべてエクスポートされます。これらの制約は、XDC ファイルで管理するか、または run スクリプトで管理できます (通常は HD.RECONFIGURABLE を使用)。

または、Pblock 制約そのものを抽出して別に管理できます。このタスクを実行しやすくするための Tcl プロシージャがあります。

- a. まず、Tcl ユーティリティ ファイルの 1 つにある、次のプロシージャを実行します。

```
source ./Tcl_HD/hd_utils.tcl
```

- b. 次に、`export_pblocks` プロシージャを使用して、この制約情報を出力します。

```
export_pblocks -file ./Sources/xdc/pblocks.xdc
```

これで、デザインの両方の Pblock の制約情報が出力されます。いずれか一方のみを選択する場合は、`-pblocks` オプションを使用します。

## 手順 6: 初回コンフィギュレーションのインプリメント

この手順では、デザインを配置配線し、新しい RM で再利用するため、デザインのスタティック部分を準備します。

### デザインのインプリメンテーション

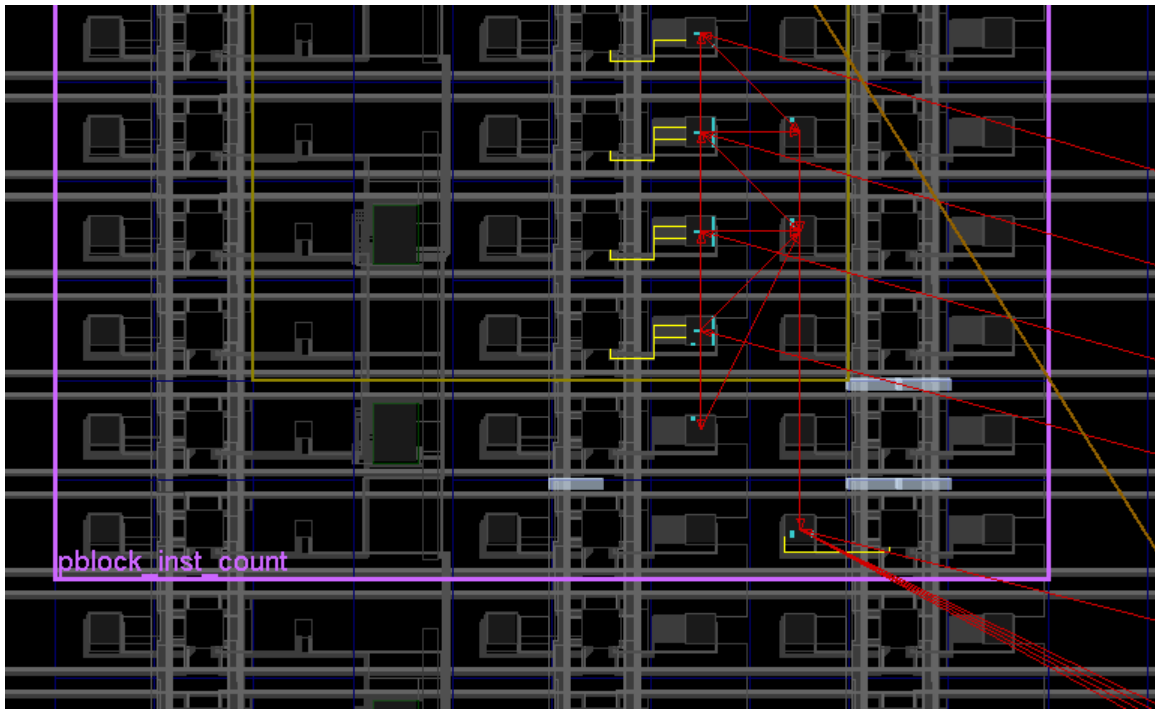
1. 次のコマンドを実行し、デザインを最適化し、配置配線します。

```
opt_design
place_design
route_design
```

`place_design` および `route_design` の両方が完了したら、次の図のように [Device] ビューでデザインのステータスを確認します。`place_design` 実行後に注意すべき点は、パーティション ピンの導入です。これらのピンは、スタティック ロジックとリコンフィギュラブル ロジックとの間をつなぐ物理的なインターフェイス ポイントで、RM の各 I/O が配線されるインターコネクト タイル内のアンカーポイントでもあります。これらのピンは、配置済みデザインのビューで白いボックスとして表示されます。`pblock_shift` の場合、スタティック部分への接続がデバイスの該当 Pblock のすぐ外側にあるので、パーティション ピンは Pblock の上辺近くに表示されます。`Pblock_count` の場合は、`SNAPPING_MODE` が RP に追加すべきフレームを垂直方向に集めるので、パーティション ピンはユーザー定義領域の外側に表示されます。

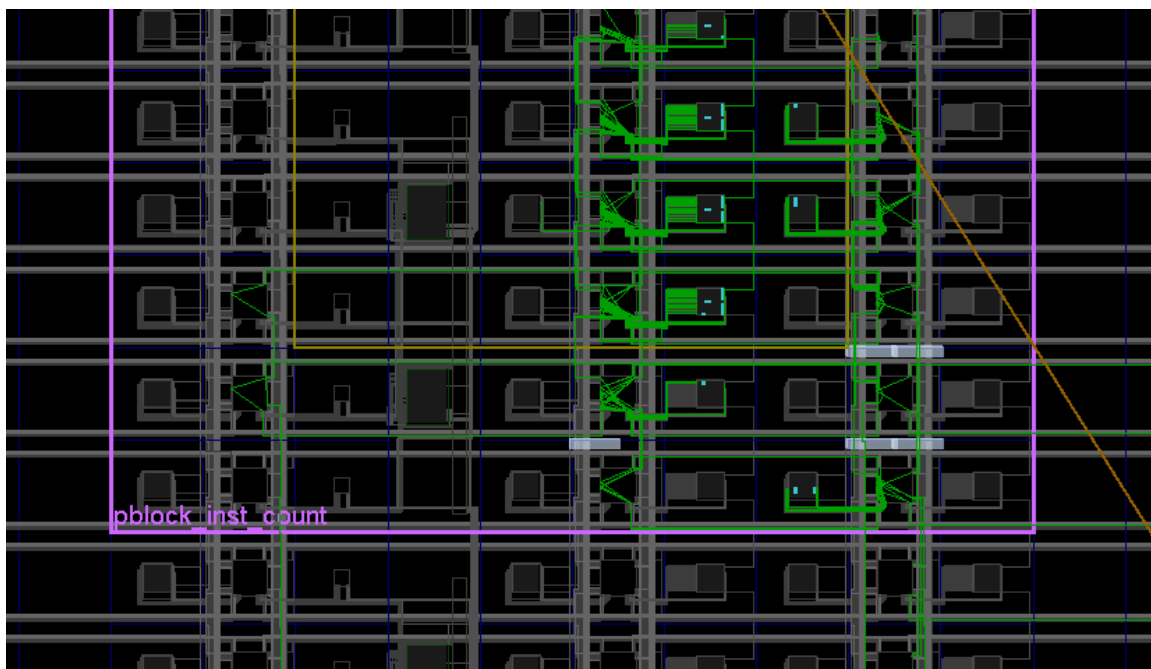


図 14: 配置済みデザイン内のパーティション ピン



2. GUI で、これらのパーティション ピンを簡単に検索するには、次の作業を実行します。
  - a. RM (たとえば `inst_shift`) を [Netlist] で選択します。
  - b. [Cell Properties] で [Cell Pins] タブをクリックします。
3. 任意のピンを選択してハイライトさせます。または、Ctrl+A キーを押して、すべてのピンを選択します。Tcl ですべてのピンを選択する場合は、`select_objects [get_pins inst_shift/*]` コマンドを使用します。
4. 抽象化された配線情報と実際の配線を切り替え、また、配線リソース自体の表示を変更するには、ツールバーの [Routing Resources] ボタンを使用します。デザインのすべてのネットがこの時点で完全に配線されています。

図 15: 配線された初回コンフィギュレーションの拡大表示



### 結果の保存

1. 次のコマンドを使用して、フル デザイン チェックポイントを保存し、レポート ファイルを生成します。

```
write_checkpoint -force Implement/Config_shift_right_count_up_implement/
top_route_design.dcp
report_utilization -file Implement/Config_shift_right_count_up_implement/
top_utilization.rpt
report_timing_summary -file Implement/
Config_shift_right_count_up_implement/top_timing_summary.rpt
```

2. [オプション] 次の 2 つのコマンドを使用し、各 RM のチェックポイントを保存します。

```
write_checkpoint -force -cell inst_shift Checkpoint/
shift_right_route_design.dcp
write_checkpoint -force -cell inst_count Checkpoint/
count_up_route_design.dcp
```



**ヒント:** バッチ モードでデザイン全体を処理するため `run_dfx.tcl` を実行する場合、デザイン チェックポイント、ログ ファイル、レポート ファイルはフローの各ステップで作成されます。

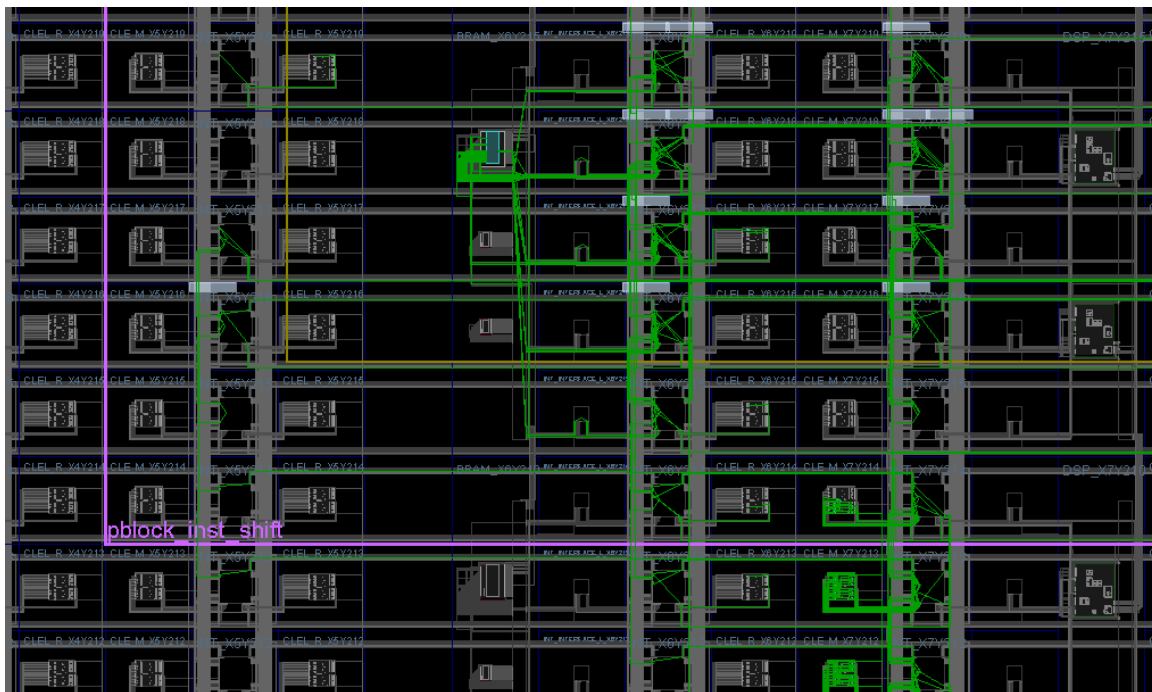
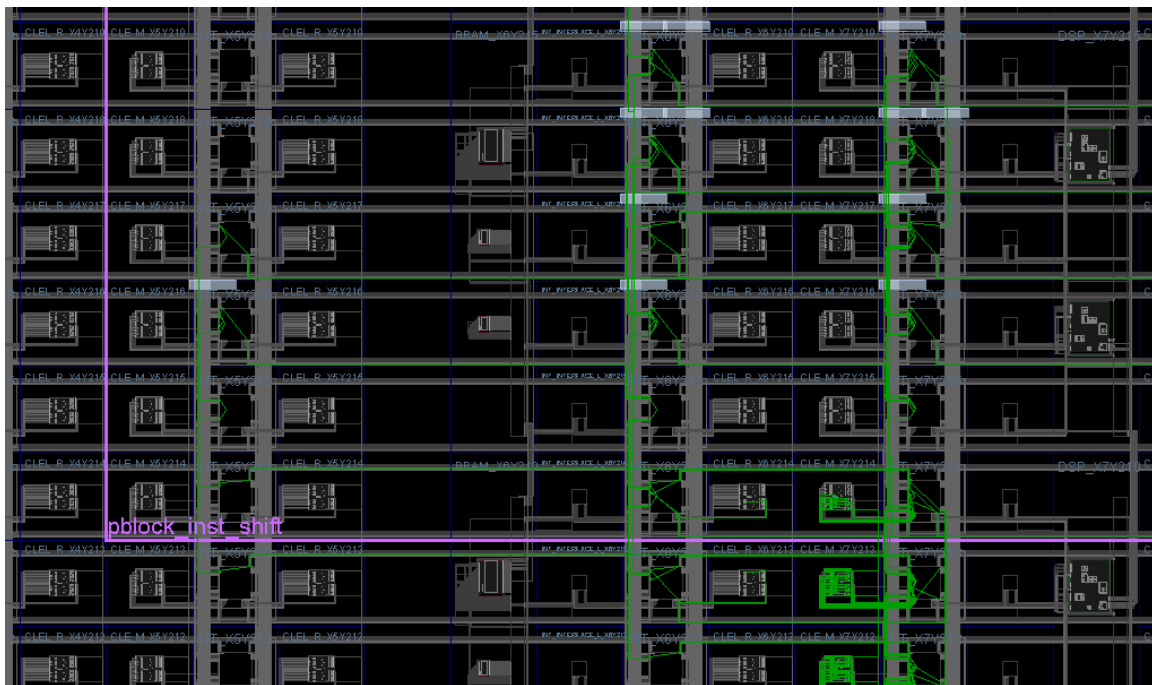
この時点で、完全にインプリメントされた Dynamic Function eXchange デザインを作成したので、ここからフルおよびパースシャルのビットストリームを生成できます。このコンフィギュレーションのスタティック部分は、この後に作成するコンフィギュレーションすべてに使用されます。このスタティック部分を隔離するには、現在の RM を削除します。配線リソースがイネーブルになっていることを確認し、パーティション ピンのあるインターコネクト タイルを拡大表示します。

3. 次のコマンドを使用して、RM ロジックをクリアにします。

```
update_design -cell inst_shift -black_box
update_design -cell inst_count -black_box
```

これらのコマンドを使用すると、次の図のようにデザインで多数の変更が発生します。

- 完全配線されたネット (緑色) の数は減少します。
- `inst_shift` および `inst_count` は、[Netlist] ビューで空の状態が表示されます。

 図 16: `update_design -black_box` 実行前 (上) の `inst_shift` モジュール

 図 17: `update_design -black_box` 実行後 (下) の `inst_shift` モジュール


1. すべての配置配線をロックするには、次のコマンドを使用します。

```
lock_design -level routing
```

lock\_design コマンドでセルが特定されていないので、メモリのデザイン全体 (現在はブラック ボックスになっているスタティック デザインで構成) に影響します。配線済みネットはすべてロックされた状態で表示されます (図 18 の点線部分)。配置済みコンポーネントもロックされていることを示すため、すべて青色からオレンジ色に表示が変わります。

2. 残りのスタティック専用チェックポイントを出力するには、次のコマンドを使用します。

```
write_checkpoint -force Checkpoint/static_route_design.dcp
```

このスタティック専用チェックポイントはこれ以降のコンフィギュレーションに使用されます。

3. 次のコンフィギュレーションに進む前に close\_project コマンドを実行してデザインを閉じます。

## 手順 7: 2 回目のコンフィギュレーションのインプリメント

スタティック デザインの結果が確立されロックされたので、これを参考にしながら、さらに RM をインプリメントできます。

### デザインのインプリメンテーション

1. Tcl コンソールで次のコマンドを実行し、新しくインメモリ デザインを作成します。

```
create_project -in_memory -part $part
```

2. 次のコマンドを実行し、スタティック デザインを読み込みます。

```
add_files ./Checkpoint/static_route_design.dcp
```

3. 次のコマンドを実行し、シフトおよびカウンタ ファンクションの 2 回目の合成チェックポイントを 2 つ読み込みます。

```
add_file ./Synth/shift_left/shift_synth.dcp
set_property SCOPED_TO_CELLS {inst_shift} [get_files ./Synth/shift_left/shift_synth.dcp]
add_file ./Synth/count_down/count_synth.dcp
set_property SCOPED_TO_CELLS {inst_count} [get_files ./Synth/count_down/count_synth.dcp]
```

4. link\_design コマンドを使用してデザイン全体をリンクします。

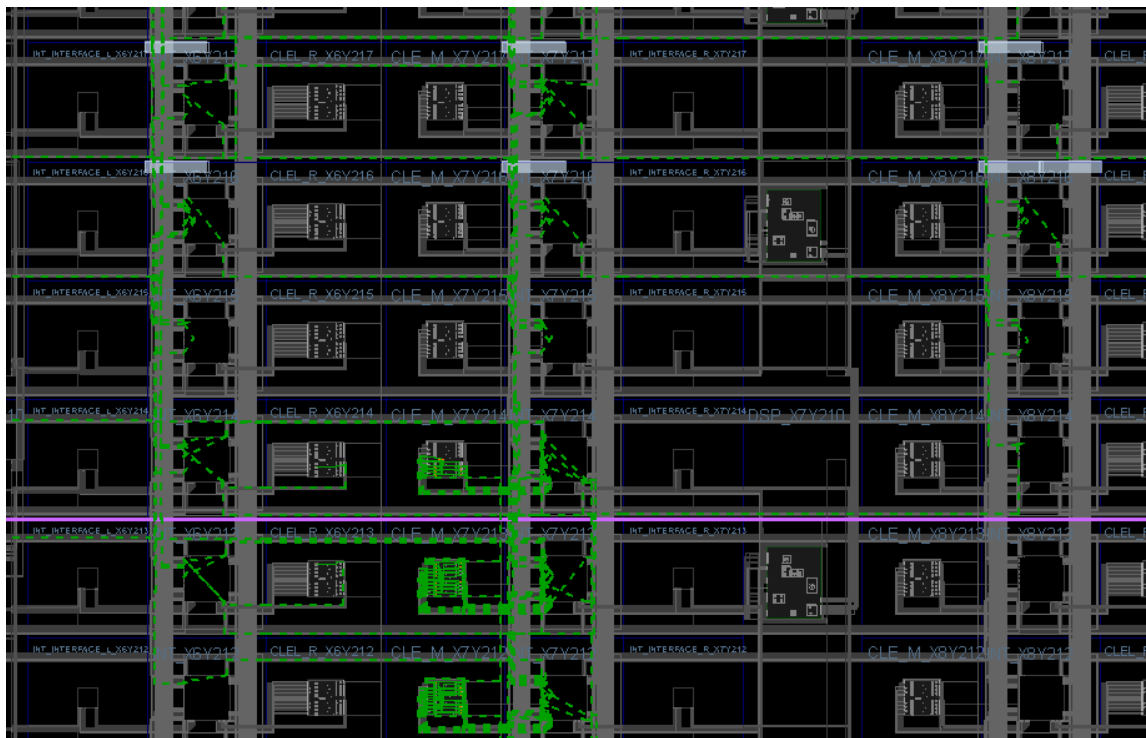
```
link_design -mode default -reconfig_partitions {inst_shift inst_count} -part $part -top top
```

この時点で、フル コンフィギュレーションが読み込まれます。ただし今回は、スタティック デザインが配線されロックされているので、リコンフィギュラブル ロジックはまだネットリストの状態でしかありません。ここからの配置配線はこのロジックにのみ適用されます。

5. opt\_design、place\_design、route\_design コマンドを実行し、スタティック デザインの初回コンフィギュレーションを参考に新しい RM を配置配線します。

デザインは再びフルにインプリメントされますが、今回は、この新しい RM のバリエーションが使用されます。次の図のように、点線(ロックされた配線)および実線(新しい配線)の配線セグメントが入り混じっています。

図 18: 2 回目のコンフィギュレーション (ロックされた配線および新しい配線の両方表示)



### 結果の保存

1. 次のコマンドを使用して、フル デザイン チェックポイントおよびレポート ファイルを保存します。

```
write_checkpoint -force Implement/Config_shift_left_count_down_import/
top_route_design.dcp
report_utilization -file Implement/Config_shift_left_count_down_import/
top_utilization.rpt
report_timing_summary -file Implement/
Config_shift_left_count_down_import/top_timing_summary.rpt
```

2. [オプション] 次の 2 つのコマンドを使用し、各 RM のチェックポイントを保存します。

```
write_checkpoint -force -cell inst_shift Checkpoint/
shift_left_route_design.dcp
write_checkpoint -force -cell inst_count Checkpoint/
count_down_route_design.dcp
```

これで、スタティック デザインおよびすべての RM のバリエーションのインプリメンテーションを完了しました。RP に RM が 3 つ以上あるデザインの場合は、このプロセスを繰り返します。

## 手順 8: スクリプトをハイライトして結果を確認

配線済みのコンフィギュレーションを IDE で開いた状態で、タイルやネットをハイライトするため可視化スクリプトをいくつか実行します。これらのスクリプトは、Dynamic Function eXchange 用に割り当てられているリソースを特定し、また自動的に生成されます。

1. Tcl コンソールで、\led\_shift\_count\_us ディレクトリから次のコマンドを実行します。

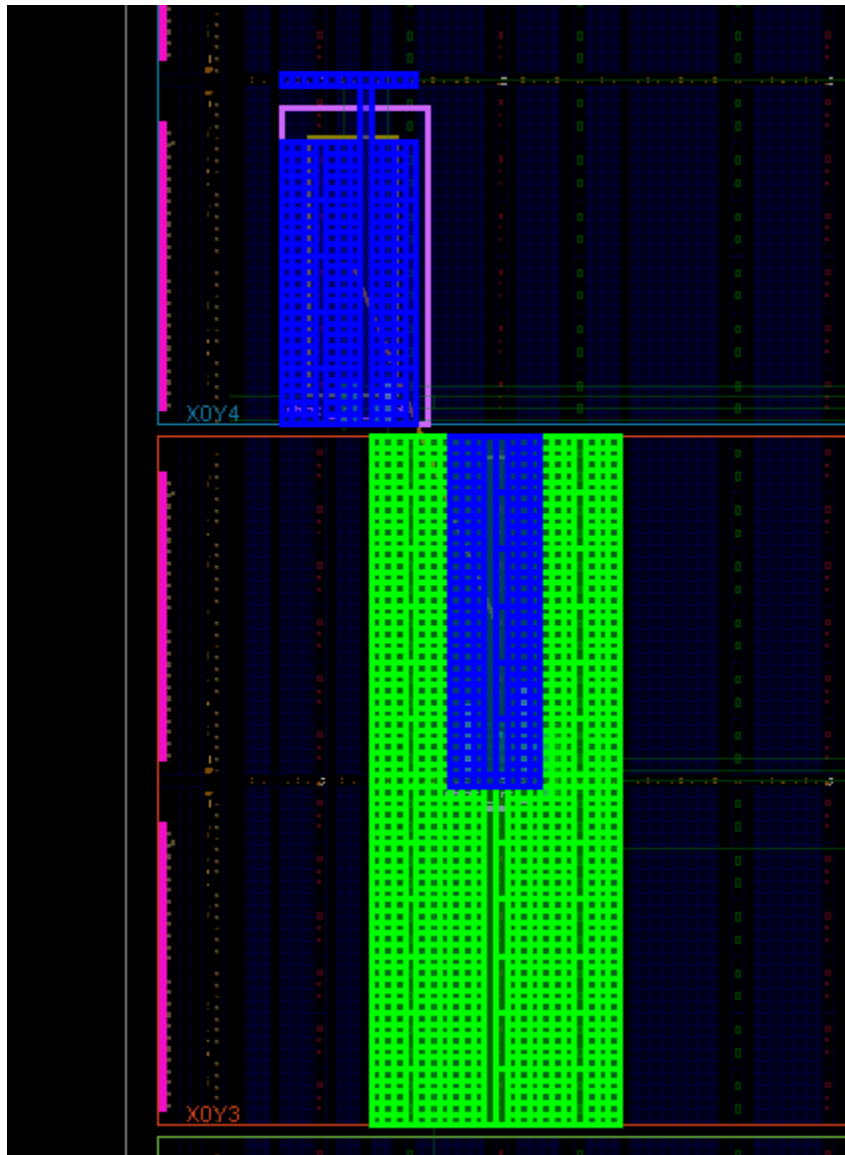
```
source hd_visual/pblock_inst_shift_Routing_AllTiles.tcl  
  
highlight_objects -color green [get_selected_objects]
```

2. フレームの選択を解除するため、[Device] ビューのどこかでクリックし (または 「unselect\_objects」と入力)、次のコマンドを実行します。

```
source hd_visual/pblock_inst_shift_Placement_AllTiles.tcl  
  
source hd_visual/pblock_inst_count_Placement_AllTiles.tcl  
  
highlight_objects -color blue [get_selected_objects]
```

次の図のように、[Device] ビューでパーティション フレームがハイライトされた状態で表示されます。

図 19: ハイライトされた RP フレーム



これらのハイライトされたタイルは、各 RM の配置 (青色) および配線 (緑色) に使用されたコンフィギュレーションフレームを表しています。緑色のタイルは、パーシャル ビットストリームを生成するため、ビットストリーム生成に送られます (inst\_shift)。SNAPPING\_MODE 機能は、プログラマブル ユニットの境界線に揃える目的で、pblock\_shift の 4 辺のうち 3 辺を調整します。これまでの手順で確認したように、スタティック ロジックが RP の内側に配置されているように見えているのは、このスナッピング動作が理由です。実際には、有効境界線はユーザー定義の Pblock の境界線よりも CLB 行で 1 行上なので、このスタティック ロジックは正しく配置されています。[手順 5: デザイン フロアプランの構築](#)にあるように、この有効境界線は作成中の Pblock の塗りつぶしでも確認できます。

**注記:** Pblock の幅の分の RCLK 行は含まれています。これらの RP のロジックを駆動するグローバル クロックは、RP の行全体に接続されていて、Dynamic Function eXchange 中にイネーブルまたはディスエーブルになります。

これ以外の「タイル」スクリプトはこのバリエーションです。クロック領域の境界線に垂直に揃えた Pblock を作成していない場合は、FrameTiles スクリプトにより明示的な Pblock タイルがハイライトされ、また AllTiles スクリプトにより、リコンフィギュラブル フレームの高さいっぱいこれらのタイルが引き伸ばされます。選択されていないフレーム タイプ (グローバル クロックなど) がある場合は、そこにギャップが残るので注意してください。

GlitchTiles スクリプトはフレーム サイトのサブセットなので、専用シリコン リソースを避けます。それ以外のスクリプトからは、さらに多くの情報が得られます。

3. 現在のデザインを閉じます。

```
close_project
```

## 手順 9: ビットストリームの生成

### コンフィギュレーションの検証



**推奨:** ビットストリームを生成する前に、すべてのコンフィギュレーションを検証し、各コンフィギュレーションのスタティック部分が同じであり、生成したビットストリームがシリコンで使用しても安全であることを確認します。PR の検証機能は、パーティション ピンを含め、それらが同じであることを確認するところまで、スタティック デザインを検証します。RM 内の配置配線は、ここでは異なるモジュール結果が予想されるため、オンになっていません。

1. Tcl コンソールで `pr_verify` コマンドを実行します。

```
pr_verify Implement/Config_shift_right_count_up-implement/
top_route_design.dcp
Implement/Config_shift_left_count_down-import/top_route_design.dcp
```

上記のコマンドで問題が発生しなければ、次のようなメッセージが表示されます。

```
INFO: [Vivado 12-3253] PR_VERIFY: check points
Implement/Config_shift_right_count_up/top_route_design.dcp and
Implement/Config_shift_left_count_down/top_route_design.dcp are
compatible
```

デフォルトでは、(不一致がある場合は) 最初の不一致のみがレポートされます。すべての不一致を表示させるには、`-full_check` オプションを使用します。

2. `close_project` を実行して、プロジェクトを閉じます。

### ビットストリームの生成

コンフィギュレーションを検証できたので、ビットストリームを生成し、それを使用して選択したデモ ボードをターゲットにします。

**注記:** 1 回目のコンフィギュレーションでは `shift_right` および `count_up` がインプリメントされます。2 回目のコンフィギュレーションでは `shift_left` および `count_down` がインプリメントされます。

1. まず、初回コンフィギュレーションをメモリに読み込みます。

```
open_checkpoint Implement/Config_shift_right_count_up-implement/
top_route_design.dcp
```



- このデザインのフルおよびパーシャル ビットストリームを生成します。ビットストリームが生成されたフル デザイン チェックポイントに関連したディレクトリ (重複しない場所) にビット ファイルを保存してください。

```
write_bitstream -force -file Bitstreams/Config_RightUp.bit close_project
```

5 つのビットストリームが生成されていることに注目してください (UltraScale+ を使用している場合は 3 つ)。

- Config\_RightUp.bit: パワーアップ、フル デザイン ビットストリームです。右の 4 つのシフト LED は右にシフトし、左の 4 つのカウント LED はカウントアップします。
- Config\_RightUp\_pblock\_inst\_shift\_partial.bit: シフト LED に右方向へシフトさせる shift\_right モジュールのパーシャル ビット ファイルです。
- Config\_RightUp\_pblock\_inst\_count\_partial.bit: カウント LED にカウントアップさせる count\_up モジュールのパーシャル ビット ファイルです。
- Config\_RightUp\_pblock\_inst\_shift\_partial\_clear.bit: UltraScale デバイス専用の shift\_right モジュールのクリア ビット ファイルです。シフト モジュールがリコンフィギュレーションできるよう、右シフトを安全にクリアにします。
- Config\_RightUp\_pblock\_inst\_count\_partial\_clear.bit: UltraScale デバイス専用の count\_up モジュールのクリア ビット ファイルです。カウント モジュールがリコンフィギュレーションできるよう、アップ カウントを安全にクリアにします。



**重要:** write\_bitstream を 1 回呼び出してビット ファイルを生成する場合、ファイルの名前に RM のバリエーションの名前が反映されず、どのイメージが読み込まれるのかをはっきりさせることができません。この解決策として、- file オプションを使用して、リコンフィギュラブル セルの Pblock 名に基本名を付けます。リコンフィギュラブル ビット ファイルをはっきりと識別できるよう、わかりやすい基本名を付けることが大切です。パーシャル ビットにはすべて \_partial という接尾辞が付き、クリア ビット ファイルにはすべて \_partial\_clear という接尾辞が付きます。

ビットストリーム生成までデザイン全体の処理に run\_dfx.tcl を使用する場合は、異なるビットストリーム生成テクニックが使用されます。配線済みのデザイン チェックポイントを開くと、write\_bitstream が複数回呼び出されるので、ビットストリームに名前が付けやすくなり、また、フル/パーシャルのビットストリームに異なるオプション (ビットストリームの圧縮など) を適用できます。たとえば、run\_dfx.tcl スクリプトで設定される名前は次のようになります。

- Config\_shift\_right\_count\_up\_implement\_full.bit: パワーアップ、フル デザイン ビットストリームです。
  - pblock\_shift\_shift\_right\_partial.bit: shift\_right モジュールのパーシャル ビット ファイルです。
  - pblock\_count\_count\_up\_partial.bit: count\_up モジュールのパーシャル ビット ファイルです。
  - pblock\_shift\_shift\_right\_partial\_clear.bit: UltraScale デバイス専用の shift\_right モジュールのクリア ビット ファイルです。
  - pblock\_count\_count\_up\_partial\_clear.bit: UltraScale デバイス専用の count\_up モジュールのクリア ビット ファイルです。
- 2 回目のコンフィギュレーションのフルおよびパーシャル ビットストリームを生成します。今回も生成されたビット ファイルを該当フォルダーに保存します。

```
open_checkpoint Implement/Config_shift_left_count_down_import/
top_route_design.dcp
write_bitstream -force -file Bitstreams/Config_LeftDown.bit
close_project
```

同様に、5 つ (または 3 つ) のビットストリームが生成されますが、今回は基本名が異なります。

2. グレー ボックスを使用してフル ビットストリームを生成し、さらに RM 用にブランキング ビットストリームも生成します。ブランキング ビットストリームは、消費電力を低減する目的で、既存のコンフィギュレーションを「消去」するために使用できます。

**注記:** グレー ボックスのブランキング ビットストリームはクリア ビットストリーム生成と同じではありません。クリア ビットストリームは、次のパーシャル ビットストリーム用のグローバル信号マスクの準備に必要で、GSR イベントが正しく起きるようにします。

```
open_checkpoint Checkpoint/static_route_design.dcp
update_design -cell inst_count -buffer_ports
update_design -cell inst_shift -buffer_ports
place_design
route_design
write_checkpoint -force Checkpoint/config_grey_box.dcp
write_bitstream -force -file Bitstreams/config_grey_box.bit
close_project
```

基本になっているコンフィギュレーション ビットストリームには、どちらの RP のロジックもありません。update\_design コマンドにより、RP のすべての出力の定数ドライバー (グラウンド) が挿入されるので、これらの出力はフロートしません。グレー ボックス (grey box) という用語は、これらの LUT が挿入されているので完全には空ではないことを示していて、この領域を出入りするネットが未接続になっているブラック ボックスとは異なります。place\_design および route\_design コマンドは、完全にインプリメントされるようにします。有効な RM として、これらのインスタンスには UltraScale デバイス専用のクリア ビットストリームもあります。

## 手順 10: FPGA のパーシャル リコンフィギュレーション

count\_shift\_led デザインは、4 つのデモ ボードの 1 つをターゲットにします。現在のデザインでは、KCU105、VCU108、KCU116、VCU118 ボード、リビジョンは 1.0 およびそれ以降のものがサポートされています。

### フル イメージを使用したデバイスのコンフィギュレーション

1. プラットフォーム ケーブル USB を使用してコンピューターにボードを接続し、ボードに電源を投入します。
2. メインの Vivado IDE で、[Flow] → [Open Hardware Manager] をクリックします。
3. 緑色のバナー上の [Open target] をクリックします。ボードとの通信を確立するため、ウィザードの手順に従います。
4. 緑色のバナー上の [Program device] をクリックし、xcvu040\_0 などのターゲット デバイスを選択します。
5. [bitstreams] フォルダを参照し、[Config\_RightUp.bit] を選択してから、[OK] をクリックしてデバイスをプログラムします。

これで 2 つのタスクを実行している GPIO LED のバンクを確認できるはずです。4 つの LED がカウントアップを実行し (MSB は左側)、別の 4 つの LED が右方向ヘシフトしています。フル デバイスのコンフィギュレーションにかかる時間に注目してください。

### デバイスのパーシャル リコンフィギュレーション

これまでに作成したパーシャル ビットストリームのいずれかを使用して、デバイスをパーシャル リコンフィギュレーションする準備が整ったので、まずは該当するクリア ビットストリームから始めます。

1. UltraScale をターゲットにしている場合のみ: 緑色のバナー上の [Program device] をもう一度クリックします。Bitstreams フォルダを参照し、[Config\_RightUp\_pblock\_inst\_shift\_partial\_clear.bit] を選択してから、[OK] をクリックしてデバイスをプログラムします。

LED のシフト部分が停止しますが、カウンタはリコンフィギュレーションの影響を受けず、引き続きカウントアップします。今回のコンフィギュレーションには前回ほど時間がかからず、DONE LED がオフになった点に注目してください。

2. UltraScale をターゲットにしている場合のみ: 緑色のバナー上の [Program device] をもう一度クリックします。Bitstreams フォルダを参照し、[Config\_LeftDown\_pblock\_inst\_shift\_partial.bit] を選択してから、[OK] をクリックしてデバイスをプログラムします。

LED のシフト部分が逆方向で再開し、DONE LED が再び点灯します。

3. 緑色のバナー上の [Program device] をもう一度クリックします。Bitstreams フォルダを参照し、[Config\_RightUp\_pblock\_inst\_count\_partial\_clear.bit] を選択してから、[OK] をクリックしてデバイスをプログラムします。

LED のカウント部分が停止しますが、シフターはリコンフィギュレーションの影響を受けず、シフトし続けます。

4. 緑色のバナー上の [Program device] をもう一度クリックします。Bitstreams フォルダを参照し、[Config\_LeftDown\_pblock\_inst\_count\_partial.bit] を選択してから、[OK] をクリックしてデバイスをプログラムします。

カウンタはカウントダウンし始めますが、シフト用の LED はリコンフィギュレーションの影響を受けません。このプロセスは、元のコンフィギュレーションに戻るまで Config\_RightUp パーシャル ビット ファイルを使用して繰り返すことができます。または、ブランキング パーシャル ビット ファイルを使用して、(点灯したままの) LED のアクティビティを停止するまで繰り返すことができます。次のパーシャル ビットストリームの前に正しいクリア ビットストリームが読み込まれるよう、各パーティションに現在読み込まれているモジュールを確認します。

## まとめ

これで演習 2 は終了です。この演習では、次の作業を実行しました。

- Dynamic Function eXchange のインプリメンテーションを準備するため、ボトムアップでデザインを合成しました。
- Dynamic Function eXchange デザインの有効なフロアプランを作成しました。
- 共通のスタティック結果を使用し、2 つのコンフィギュレーションを作成しました。
- この 2 つのコンフィギュレーションをインプリメントし、それぞれでスタティック デザインを使用できるよう保存しました。
- 後で再利用するため、スタティック モジュールおよび RM のチェックポイントを作成しました。
- フレームセットを確認し、この 2 つのコンフィギュレーションを検証しました。
- フルおよびパーシャル ビットストリームを作成しました。
- FPGA をコンフィギュレーションしてから、パーシャル リコンフィギュレーションしました。

## 演習 3

# DFX プロジェクト フロー

この演習では、プロジェクトを新規作成し、DFX デザインの構造を定義するソースおよび run をすべて設定します。この演習で使用するデザインは、演習 1 および 2 の単純なデザインをベースにしていますが、変更して、シフトとカウントを 1 ずつではなく、シフト モジュールのインスタンスを 2 つにしています。これでプロジェクト フローでパーティション定義がどうなるかを確認できます。

この演習では、次のザイリンクス開発プラットフォームをターゲットにしています。

- KCU116 (Kintex® UltraScale+™)
- VCU118 (Virtex® UltraScale+™)
- KCU105 (Kintex UltraScale)
- VCU108 (Virtex UltraScale)
- KC705 (Kintex-7)
- VC707 (Virtex-7)
- VC709 (Virtex-7)

## 手順 1: チュートリアル デザイン ファイルの抽出

1. ザイリンクス ウェブサイトから[リファレンス デザイン ファイル](#)をダウンロードします。
2. ZIP ファイルの内容を書き込み可能なディレクトリに抽出します。
3. `\dfx_project` に移動します。

## 手順 2: 初回デザイン ソースの読み込み

DFX デザイン フロー (プロジェクト ベース) ではまず、デザインのリコンフィギャラブル部分を定義する作業から始めます。これには、プロジェクト モードのコンテキスト メニュー [Hierarchical Source View] を使用して定義します。ここでは、最初にプロジェクトを作成し、単純デザインでパーティションを定義するところまでを実行します。

1. アーカイブ ファイルからデザインを抽出します。dfx\_project データ ディレクトリは、この演習では `<Extract_Dir>` と表記されます。
2. Vivado IDE を開き、[Create Project] を選択して、[Next] をクリックします。
3. [Project location] に `<Extract_Dir>` を選択します。プロジェクト名は `project_1` のままに、[Create project subdirectory] をオンのままにしておきます。[Next] をクリックします。
4. RTL Project を選択し、Next チェック ボックスがオフになっているのを確認して、[Next] をクリックします。

5. [Add Directories] ボタンをクリックし、デザインに追加するソース ディレクトリを選択します。

- <Extract\_Dir>\Sources\hdl\top
- <Extract\_Dir>\Sources\hdl\shift\_right

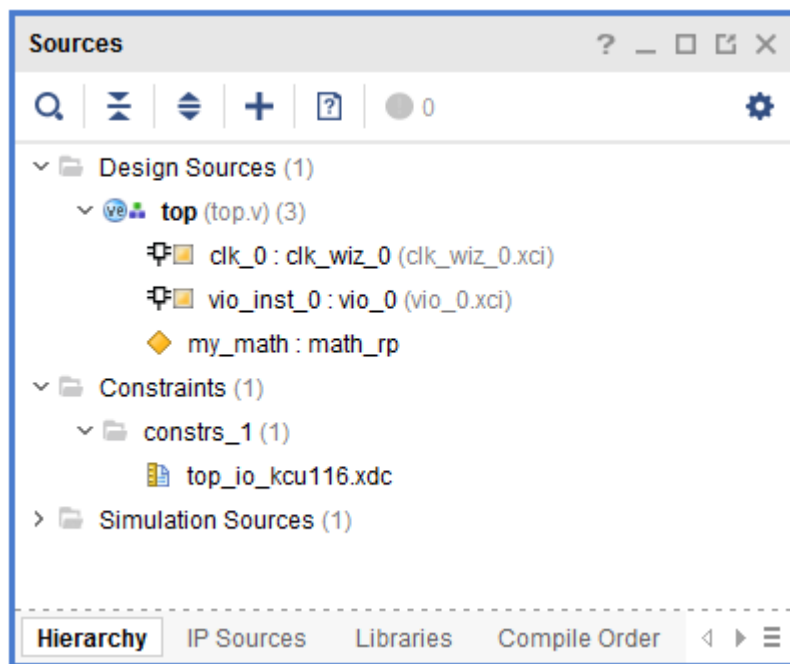
6. [Next] をクリックして [Add Constraints] に進み、次のファイルを追加します。

- <Extract\_Dir>\Sources\xdc\top\_io\_<board>.xdc
- <Extract\_Dir>\Sources\xdc\pblocks\_<board>.xdc

これらの制約ファイルはフル デザイン制約で、最上位デザイン用です。フロアプランを実行する必要がある場合は、top\_io xdc のみを選択し、pblocks\_<board>.xdc は割愛します。ユーザー自身のフロアプランを作成するには、合成後に次の手順に従います。

7. パーツを選択するには、[Next] をクリックします。パーツを選択するメニューで [Boards] をクリックし、選択した制約ファイルと一致するターゲット ボードを選択します。それから [Next] をクリックし、[Finish] をクリックしてプロジェクト作成を完了させます。[Sources] ウィンドウにはデザインの標準階層ビューが表示されます。

図 20: プロジェクト作成後の [Sources] ウィンドウ

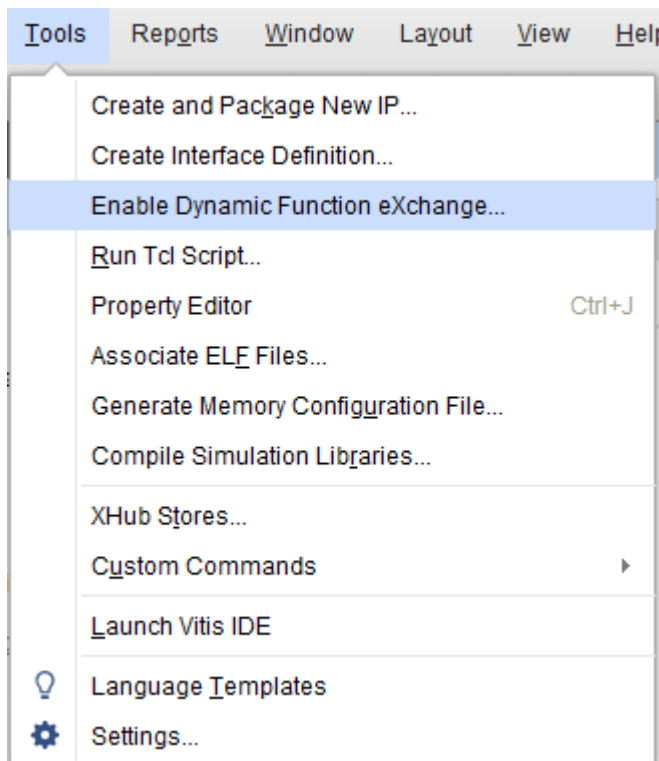


この時点で、標準プロジェクトが開いています。Dynamic Function eXchange に特化した作業は実行されていません。

8. [Tools] → [Enable Dynamic Function eXchange] をクリックします。

これでプロジェクトが DFX デザイン フロー用になります。これをいったん設定すると元に戻すことはできません。このオプションを選択する前に、プロジェクトをアーカイブすることをお勧めします。

図 21: Dynamic Function eXchange のイネーブル



確認のダイアログ ボックスで [Convert] をクリックしてこのプロジェクトを DFX プロジェクトに変換します。

9. 2 つあるシフト インスタンスの 1 つを右クリックし、[Create Partition Definition...] オプションを選択します。

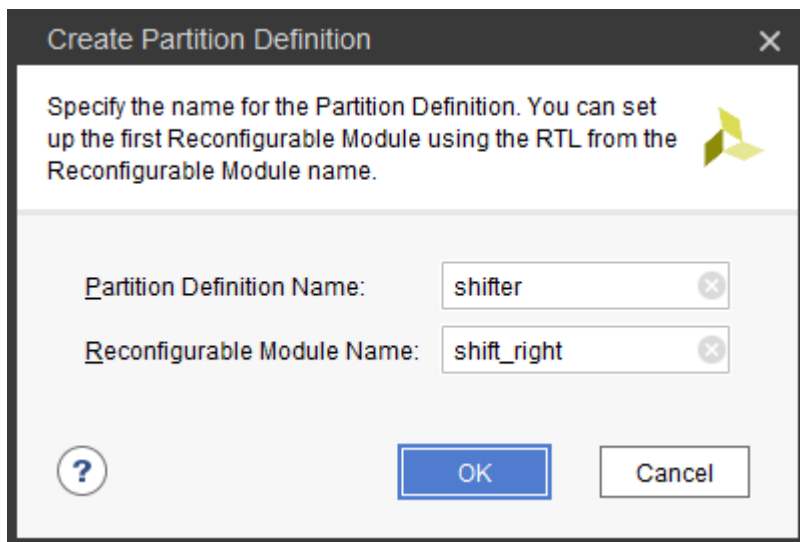
この作業で、デザインの両方のシフト インスタンスがリコンフィギャラブル モジュール (RM) として定義されます。どちらのインスタンスも RTL ソースは同じなので、論理的には同じです。最上位とこのモジュールを分けておくため、アウト オブ コンテキスト合成が実行され、1 つの合成後のチェックポイントが両方のシフト インスタンスに使用されます。



**ヒント:** デザイン内に同じモジュールのインスタンスが複数あって、リコンフィギャラブルである必要はない場合、これらのモジュール インスタンスをそれぞれ独立させるため手動で変更する必要があります。そうしておく、必要なインスタンスをリコンフィギャラブル パーティション (RP) として個別にタグ付けできます。

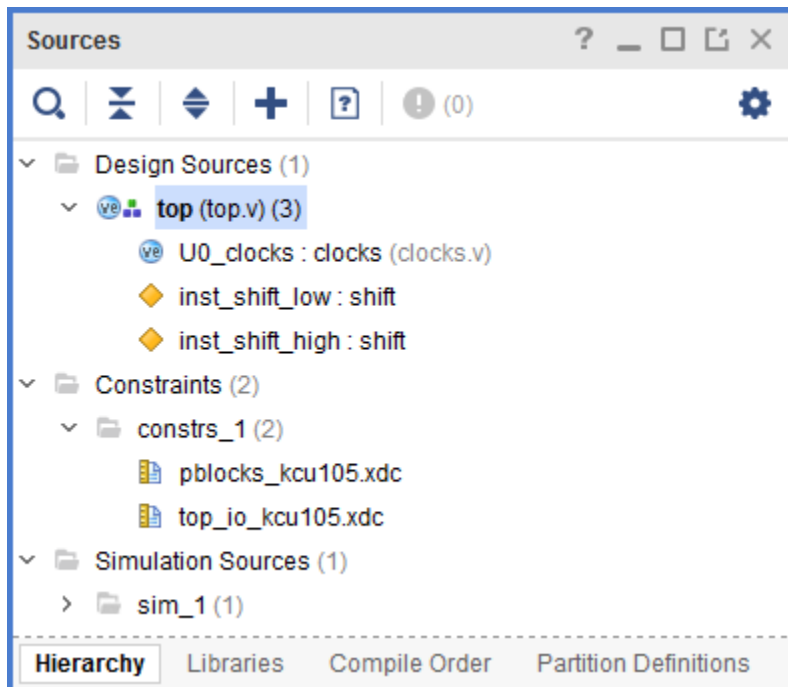
10. 表示されるダイアログ ボックスで、[Partition Definition Name] および [Reconfigurable Module Name] の両方に名前を指定します。このパーティション定義は、すべての RM が挿入されるワークスペースを一般的に参照するためのものなので、「shifter」など適切な名前を付けます。一方、RM は、この特定の RTL インスタンスを参照するので、「shift\_right」など機能がわかるような名前を付けて、[OK] をクリックします。

図 22: shifter パーティション定義の作成



[Sources] ウィンドウがこれで少し変更され、両方のシフト インスタンスは、この 2 つがパーティションであることがわかるように、その横に黄色いひし形が表示されます。また、このウィンドウの [Partition Definitions] タブに、デザインのすべてのパーティション定義のリストおよび内容が表示されているのも確認できます (この時点では 1 つ)。さらに、shift\_right モジュールを合成するため、アウト オブ コンテキスト モジュール run が作成されています。

図 23: シフトのパーティション定義後の [Sources] ウィンドウ



この時点で、さらに RM ソースを追加できます。これは、Dynamic Function eXchange ウィザードを使用して追加できます。





**重要:** パーティションを定義した後に追加する RM はすべて、DFX ウィザードを使用して追加する必要があります。RM ソース、コンフィギュレーション、run の管理にも、このウィザードを使用する必要があります。

## 手順 3: Dynamic Function eXchange ウィザードを使用しデザインを完了

1. Flow Navigator または [Tools] メニューから [Dynamic Function eXchange Wizard] をクリックし、Partial Reconfiguration ウィザードを開きます。
2. [Next] をクリックして [Edit Reconfigurable Modules] ページに進みます。shift\_right RM が既にあるのがわかります。この RM の上のほう、ウィンドウの左側に追加、削除、編集ボタンがあります。青い [+] アイコンをクリックし、新しく RM を追加します。
3. [Add Directories] ボタンをクリックして [shift\_left] フォルダを選択します。

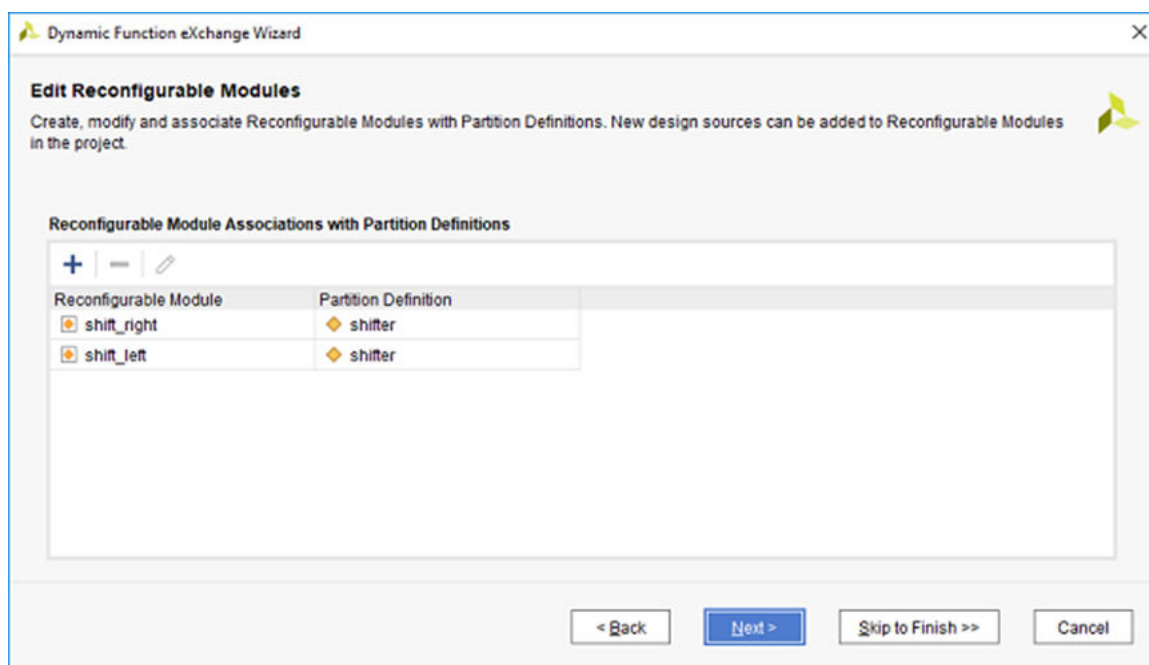
```
<Extract_Dir>\Sources\hdl\shift_left
```

または、[Add Files] ボタンをクリックし、このディレクトリにある shift\_left.v ファイルを選択します。モジュールレベルの制約が必要な場合は、ここで追加します。その場合、制約をこのパーティションの階層に合わせる必要があります。

[Reconfigurable Module] フィールドに「shift\_left」と入力します。[Partition Definition] を shifter に設定し、[Top Module] フィールドを空白のままにし、[Sources are already synthesized] チェック ボックスをオフにします。[OK] をクリックし、この新しいモジュールを作成します。

これで、シフター RP に対し、RM が 2 つ使用できるようになりました。

図 24: RM を 2 つ定義した Dynamic Function eXchange ウィザード

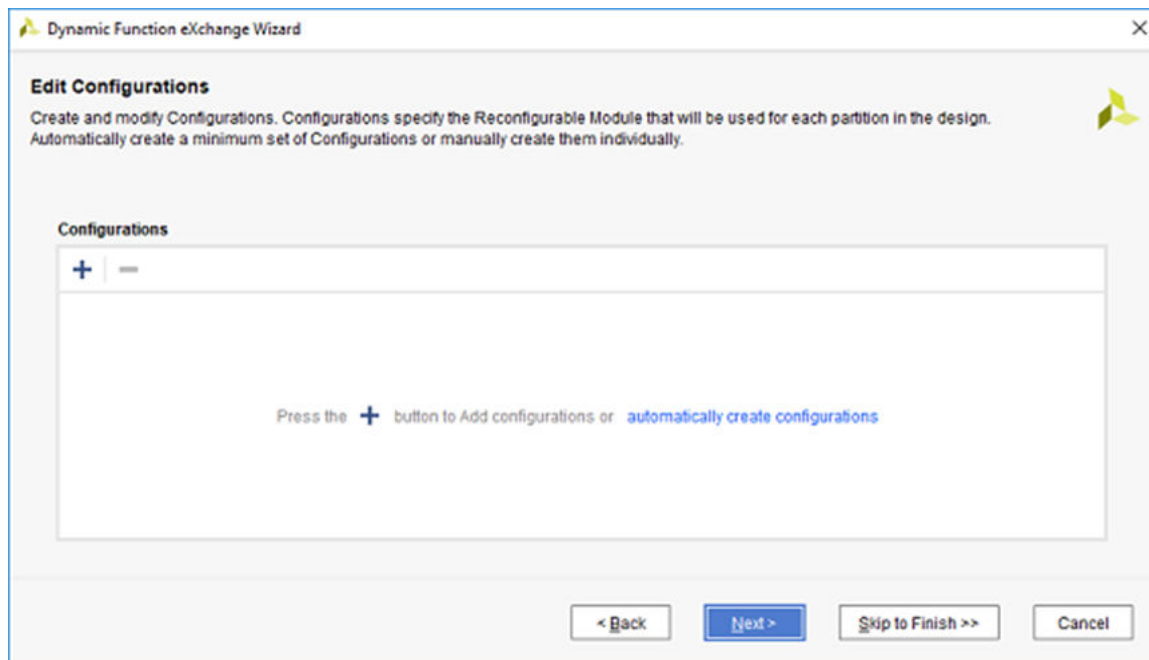




次のページでは [Configurations] を定義します。コンフィギュレーションはフル デザインのイメージで、ステティック デザイン、および RP ごとに 1 つの RM から構成されています。必要な数のコンフィギュレーションを作成することもできますし、単にウィザードに任せてコンフィギュレーションを自動選択することもできます。

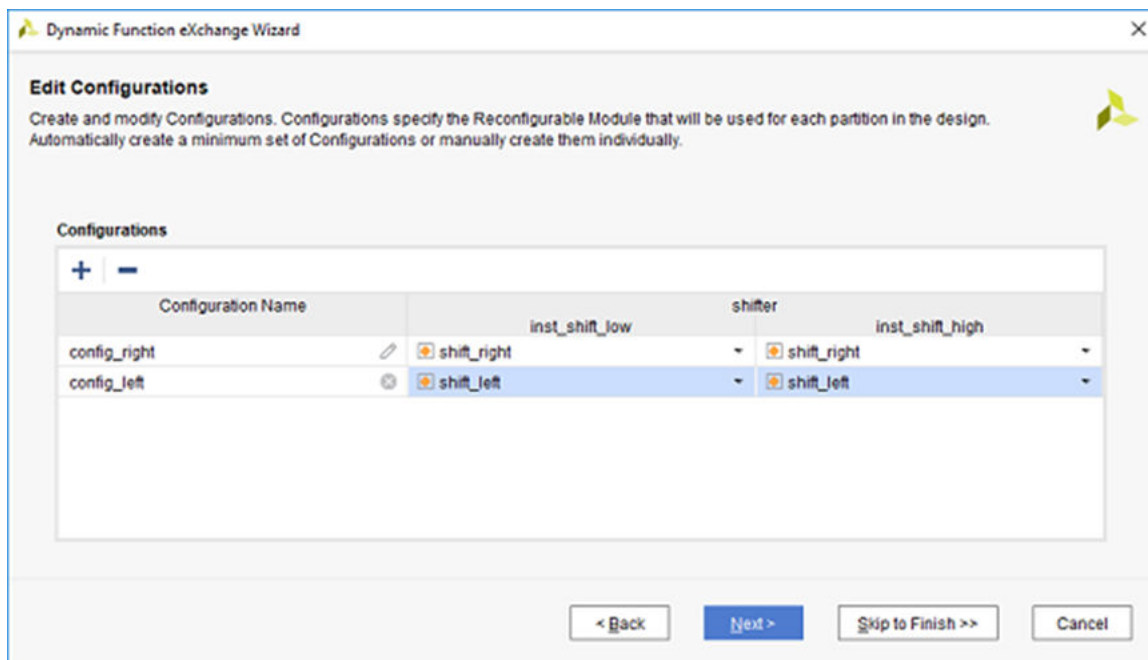
4. ここでは、automatically create configurations リンクを選択して、コンフィギュレーションをウィザードで自動作成します。

図 25: Dynamic Function eXchange ウィザードの [Configurations] ページ



このオプションを選択した後は、最低 2 つのコンフィギュレーションがセットになって作成されます。2 つあるシフト インスタンスのうち、shift\_right を 1 番目のコンフィギュレーションに、shift\_left を 2 番目のコンフィギュレーションに割り当てます。[Configuration Name] は編集可能なフィールドですが、次の図に示すように、それぞれに含まれる RM を反映し、名前が [config\_right] および [config\_left] に更新されています。

図 26: 自動生成された最小セットのコンフィギュレーション



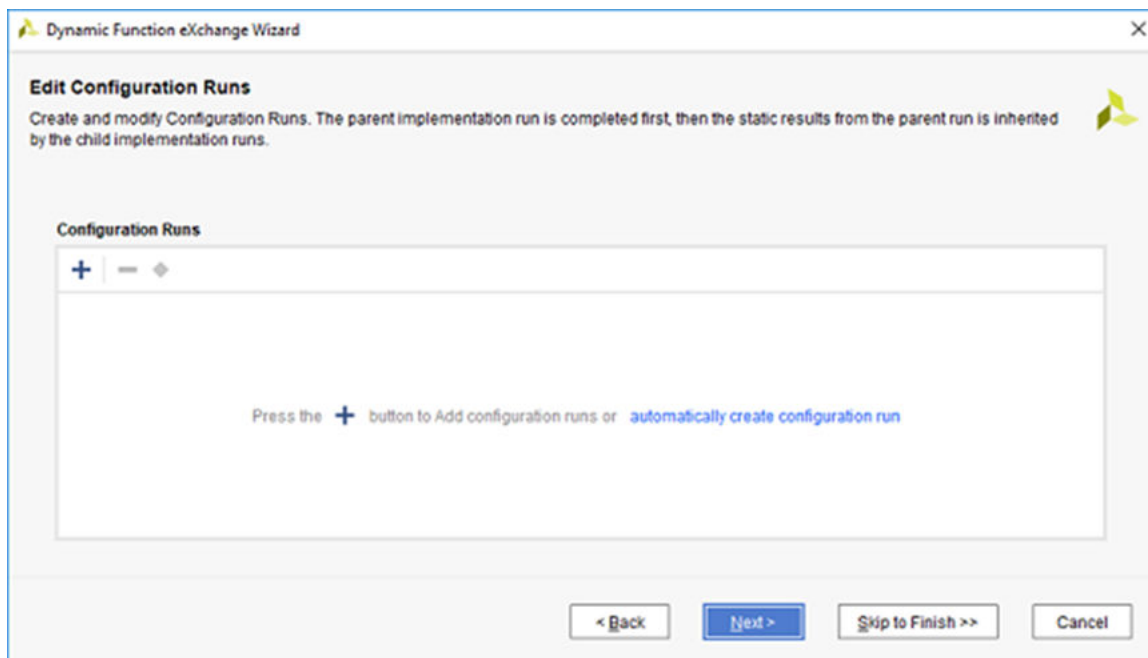
追加コンフィギュレーションは、この 2 つの RM を使用して作成できますが、どの RP でも最大 RM 数は 2 なので、このバージョンのデザインに必要なパーシャル ビットストリームをすべて作成するのに必要なコンフィギュレーションは 2 つだけです。

5. [Next] をクリックして [Edit Reconfigurable Runs] ページに進みます。

コンフィギュレーションの場合と同じように、各コンフィギュレーションをインプリメントするのに使用される run が手動または自動で作成できます。階層の親子関係で run の関わり合い方が定義されます。まず親 (上位) run で、スタティック デザイン、およびそのコンフィギュレーション内のすべての RM がインプリメントされます。それから子 run で、ロックされたスタティック デザインが再利用され、同時に、その確立されたコンテキストでそのコンフィギュレーション内で RM がインプリメントされます。

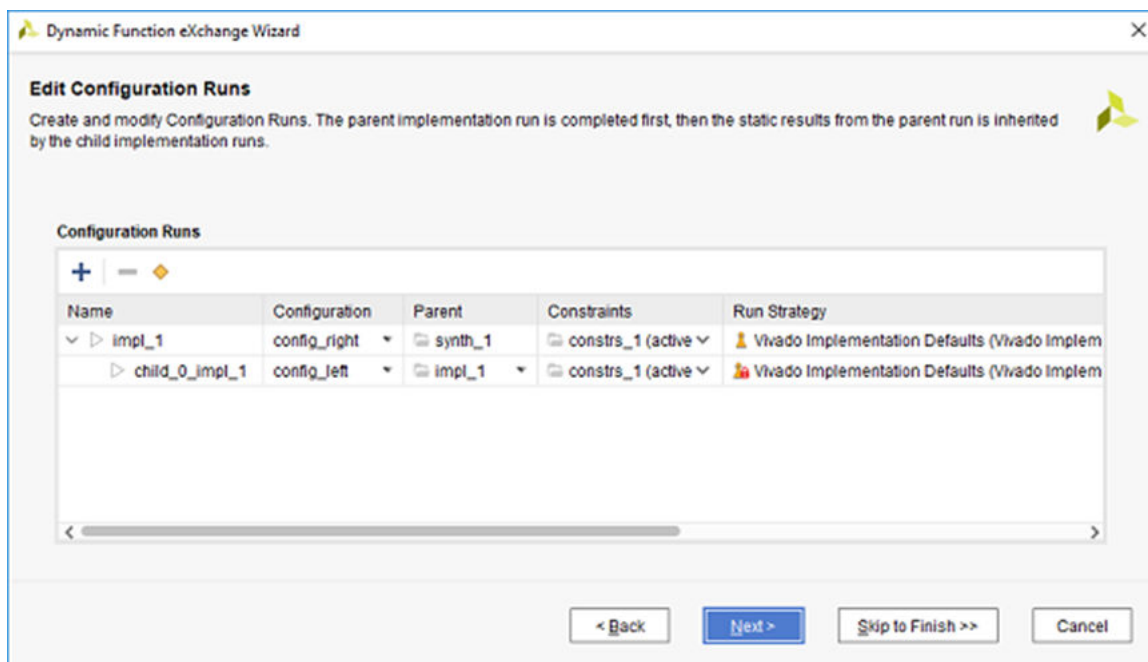
6. [automatically create configuration run] リンクをクリックすると、最低数の run で [Configuration Runs] ページに情報が自動入力されます。

図 27: Dynamic Function eXchange ウィザード



これで、親コンフィギュレーション (config\_right) 1つ、および子コンフィギュレーション (config\_left) 1つの run が2つ作成されます。独立した run でも関連している run でも、このウィザード内で、異なるストラテジや制約セットを使用し、必要な数だけ run を作成できます。ここでは、run はこの2つだけにとどめておきます。これらの run の名前は編集できないので注意してください。

図 28: コンフィギュレーション run の最小セットを自動生成



7. [Next] をクリックして、[Summary] ページを確認し、[Finish] をクリックして、デザイン設定を完了させ、ウィザードを終了します。



**重要:** [Finish] をクリックして DFX ウィザードを終了するまでは、何も作成されず、変更も反映されません。[Finish] ボタンをクリックするまで、それまでの作業内容はすべてキューに残っているので、変更部分をインプリメントしないで、ウィザードの前のページに戻って、納得がいくまで設定を変更することが可能です。

Vivado IDE に戻って、[Design Runs] ウィンドウがアップデートされているのを確認します。shift\_left RM に 2 番目のアウト オブ コンテキスト合成 run が追加され、子インプリメンテーション run (child\_0\_impl\_1) が親 (impl\_1) の下に作成されます。これでデザインを処理する準備が整いました。

図 29: 合成およびインプリメンテーションがすべて起動できる状態になった [Design Runs] ウィンドウ

Tcl ConsoleMessagesLogReportsDesign Runs × Configurations

🔍

⌵

⬆

⏮

⏪

⏩

⏭

+

%

Name	Configuration	Constraints	Status	WNS
▼ ▶ synth_1 (active)		constrs_1	Not started	
▼ ▶ impl_1 (active)	config_right	constrs_1	Not started	
▶ child_0_impl_1	config_left	constrs_1	Not started	
▼ Out-of-Context Module Runs				
▶ shift_right_synth_1		shift_right	Not started	
▶ shift_left_synth_1		shift_left	Not started	

## 手順 4: 現在のデザインの合成およびインプリメンテーション

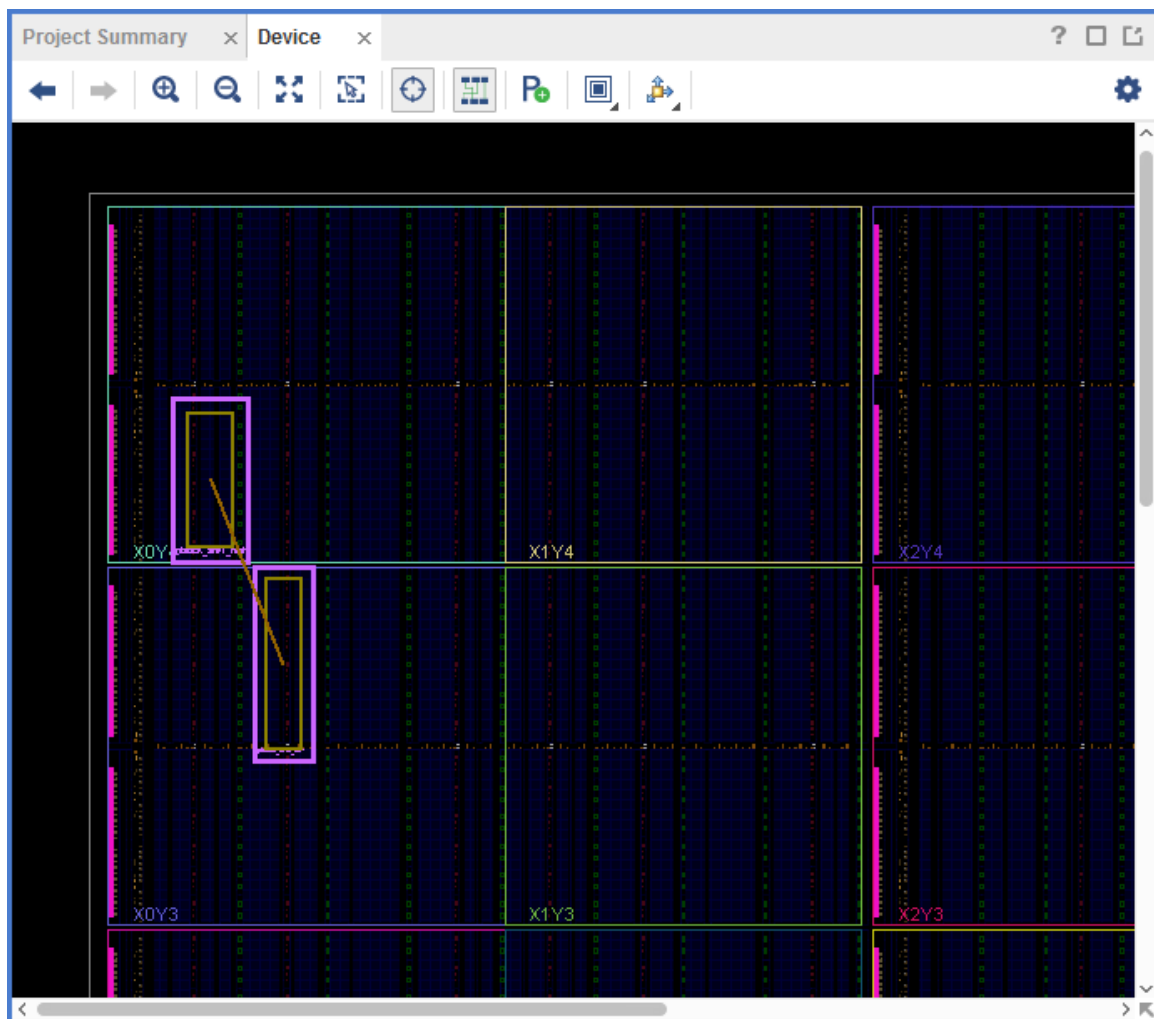
前段階で用意したデザインを Vivado IDE で開き、[Design Runs] ウィンドウを確認します。最上位デザイン合成 run (synth\_1) および親インプリメンテーション run (impl\_1) は「アクティブ」になっています。Flow Navigator での作業内容は、これらのアクティブな run およびその子 run に適用されるので、[Run Synthesis] または [Run Implementation] をクリックすると、これらの run のみが実行され、また、その完了に必要な OOC 合成 run まで実行されます。特定の親または子インプリメンテーション run を選択し、右クリックして [Launch Runs] を選択すれば、最終ターゲットのフローを最後まで実行できます。

1. Flow Navigator で [Run Synthesis] → [Open Synthesized Design] をクリックします。

これで、すべての OOC モジュールが合成され、それから最上位の合成が実行されます。OOC モジュール (IP であってもなくても) を含んだデザインであればすべて、この同じように実行されます。

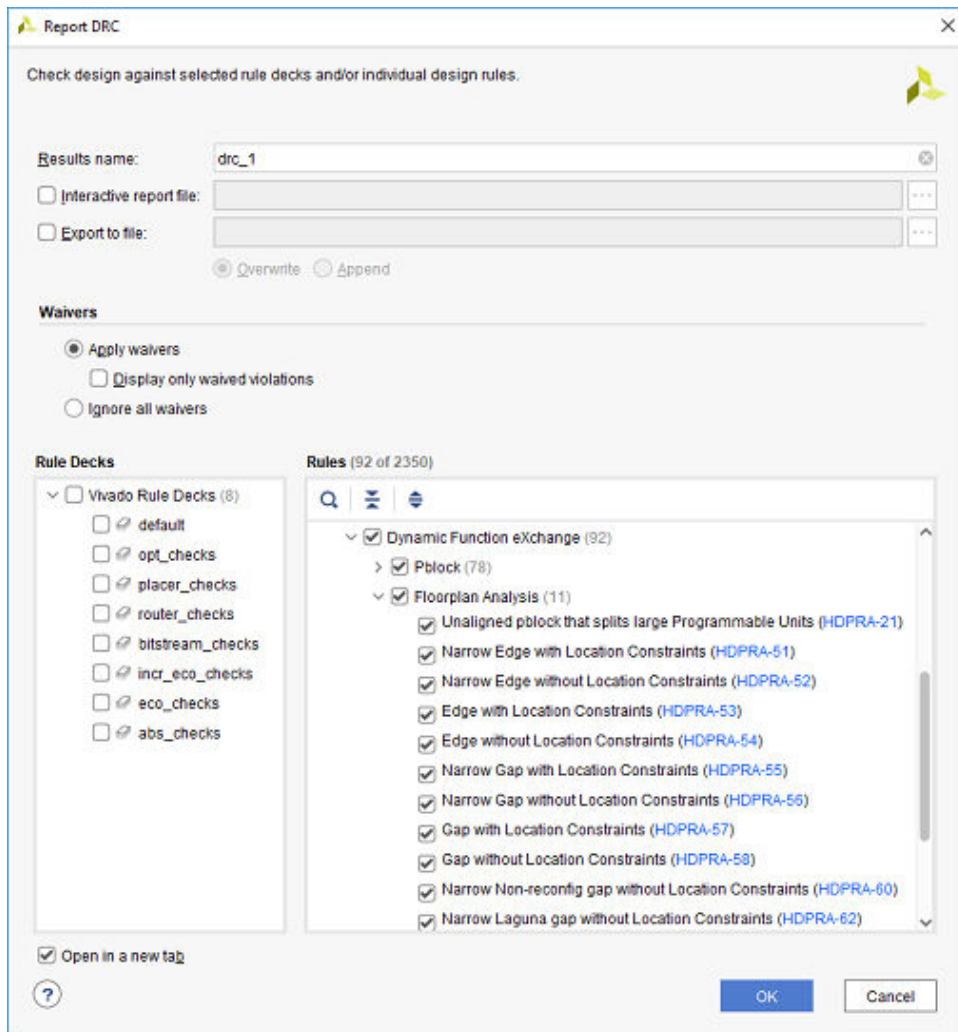
合成後に開いたデザインには、2 つの Pblock が既に定義されている点に注目してください。この 2 つの Pblock は pblocks\_<board>.xdc にあり、最上位の 2 つのシフト インスタンスにマップされます。デザイン ソースに Pblock がない場合は、フローのこのステップで作成できます。デザイン階層で inst\_shift インスタンスで右クリックし、[Floorplanning] → [Draw Pblock] をクリックすると作成できます。インスタンスごとに Pblock が必要になります。

図 30: 2 つの RP を使用したフロアプラン (KC105)



2. フロアプランにある 2 つの Pblock のいずれかを選択し、そのプロパティを確認します。最後にリストされている 2 つのプロパティは RESET\_AFTER\_RECONFIG (7 シリーズのみ) および SNAPPING\_MODE で、DFX に特化したプロパティです。これらのオプションはどちらも Pblock の XDC でイネーブルになっている点に注目してください。
3. [Reports] → [Report DRC] をクリックして、DFX に特化したデザイン ルール チェックを実行します。時間を節約するには、Dynamic Function eXchange のチェック ボックスだけでなく、すべてのチェック ボックスをすべてオフにできます。

図 31: DFX DRC の確認



提供されているソースおよび制約に DRC を実行すると、エラーは発生しません。デバイスによっては、Pblock の質を改善する方法を提案するアドバイザー メッセージが表示される可能性があります。この演習のデザインは単純なので、これらのメッセージが表示されても無視しても問題ありません。

ユーザー独自のフロアプランを作成してあって、それに対して DRC エラーがレポートされている場合は、作業を進める前に問題を修正します。どちらのモジュールにも BRAM リソースが必要です。水平方向または垂直方向のアライメントに関するエラーは、SNAPPING\_MODE では修正されることを思い出してください。



**ヒント:** DFX のデザイン ルール チェックを早い段階で、頻繁に実行します。

4. Flow Navigator で [Run Implementation] を選択し、すべてのコンフィギュレーションで 配置配線を実行します。  
まず impl\_1 のインプリメンテーション、次に child\_0\_impl\_1 のインプリメンテーションが実行されます。詳細はすべて、バックグラウンドで Vivado で処理されます。すべての DFX 要件が揃った状態でこの 2 つの run で配置配線が実行されるだけでなく、DFX に特化したタスクがさらに 2、3 実行されます。impl\_1 が完了したら、Vivado により次の作業が自動的に実行されます。

- 配線済みの shift\_right RM それぞれに、モジュール レベル (OOC) のチェックポイントが生成されます。



- 最上位用にスタティックのみのデザイン イメージを作成するため、各 RP のロジックが抜き出されます。そのため、各インスタンスに `update_design -black_box` が呼び出されます。
- デザインのスタティック部分のみの配置配線がすべてロックされます。そのため、`lock_design -level routing` が呼び出されます。
- すべての子 run に再利用できるよう、ロックされたスタティックの親イメージが保存されます。

さらに、子 run が完了すると、モジュール レベルのチェックポイントが配線済みの `shift_left RM` 用に作成されます。ロックされたスタティック デザインのイメージは、親と同じなので、このステップは不要です。










特定のコンフィギュレーション run のみが必要な場合は、[Design Runs] ウィンドウ内でそれらを個別に選択できます。子 run は親 run からのロックされたスタティック デザインを使用して開始するため、子 run を起動する前に親 run を完了させておく必要があります。

- インプリメンテーションが完了したらダイアログ ボックスが開くので、そこで [Cancel] をクリックします。



**注意:** デザインが子インプリメンテーション run まで処理されていても、[Open Implemented Design] を選択すると、デフォルトで親 run が開きます。プルダウン メニューを使用して、目的のインプリメンテーション run を選択します。

図 32: すべてのコンフィギュレーションが配線された状態

Tcl Console Messages Log Reports Design Runs × I/O Ports							
        							
Name	Configuration	Constraints	Status	WNS	TNS	WHS	THS
✓ synth_1 (active)		constrs_1	synth_design Complete!				
✓ impl_1 (active)	config_right	constrs_1	route_design Complete!	8.254	0.000	0.079	0.000
✓ child_0_impl_1	config_left	constrs_1	route_design Complete!	8.254	0.000	0.079	0.000
Out-of-Context Module Runs							
✓ shift_right_synth_1		shift_right	synth_design Complete!				
✓ shift_left_synth_1		shift_left	synth_design Complete!				

この時点で、あと 2 つのステップが残っています。1 つ目のステップは、デザイン イメージのスタティック部分が一貫していることを確認するため、2 つのコンフィギュレーションを比較して、PR 検証を実行します。このステップの実行は強く推奨され、Vivado プロジェクト内で自動的に実行されます。2 つ目のステップはビットストリーム自体の生成です。

- Flow Navigator で [Generate Bitstream] をクリックします。これで、アクティブになっている 親 run に対しビットストリーム生成が起動し、インプリメント済みの子 run すべてに対し、PR 検証、それからビットストリーム生成が起動します。

各コンフィギュレーション run に対し、フルおよびパースシャルのビットストリームがデフォルトで生成されます。

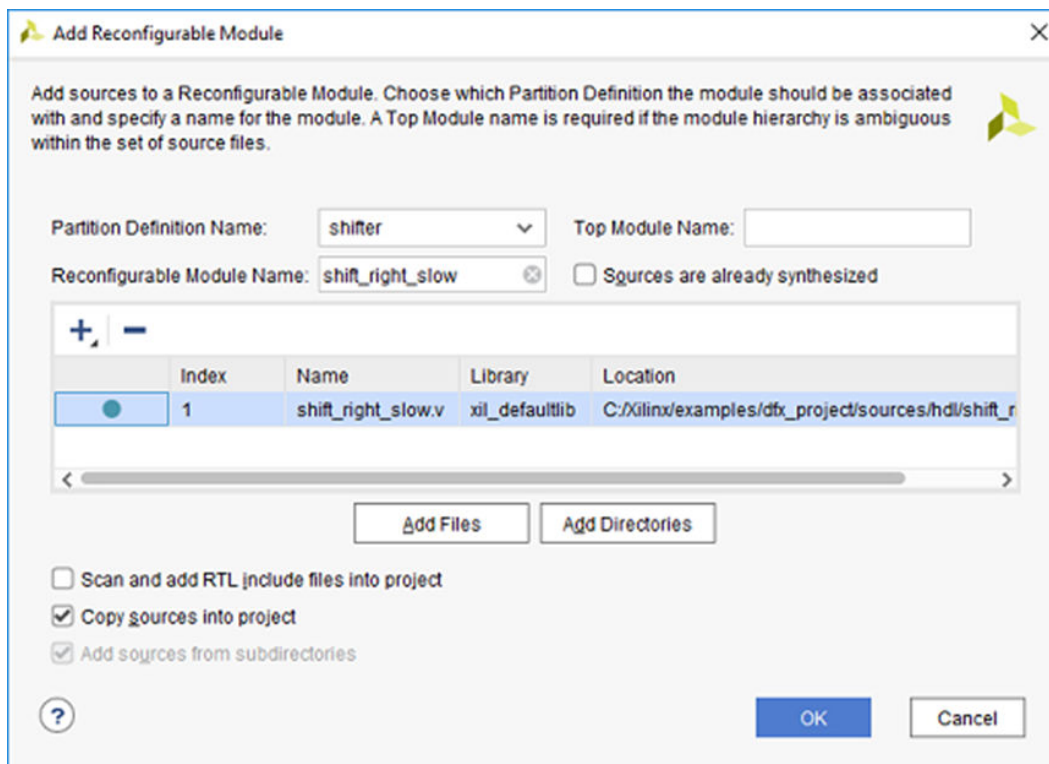
これで、Dynamic Function eXchange フロー全体をプロジェクト環境で実行できるようになりました。モジュール レベルの合成からビットストリーム生成まで、すべてのステップを GUI で実行できます。

## 手順 5: 追加リコンフィギャラブル モデルおよびコンフィギュレーションの追加

- Vivado IDE でデザインを開いた状態で、Dynamic Function eXchange ウィザードを開きます。

2. [Edit Reconfigurable Modules] ページで [+] ボタンをクリックし、新しい RM を追加します。
3. <Extract\_Dir>\Sources\hdl\shift\_right\_slow で shift\_right\_slow.v ファイルを選択し、[OK] をクリックします。
4. [Reconfigurable Module Name] に「shift\_right\_slow」と入力し、[OK] をクリックしてから、[Next] をクリックします。

図 33: 新規 RM 「shift\_right\_slow」の追加

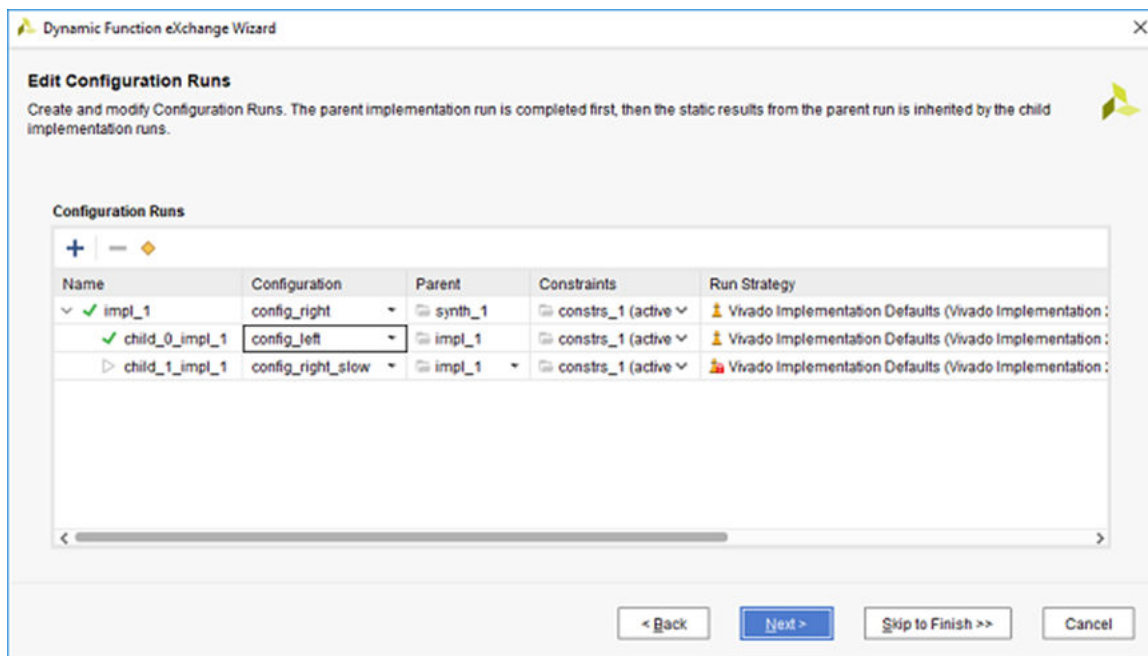


[Edit Configurations] ページにコンフィギュレーションを自動作成するオプションがなくなっています。これは、既にコンフィギュレーションが 2 つ存在しているからです。既存のコンフィギュレーションをすべて削除するとこのオプションが再び表示されますが、そうすると、すべてのコンフィギュレーションがまた作成され、既に生成されている結果ファイルがすべて削除されます。

5. [+] ボタンをクリックして新しいコンフィギュレーションを作成し、「config\_right\_slow」と名前を付け、ENTER キーを押します。各 RP のインスタンスに shift\_right\_slow を選択します。

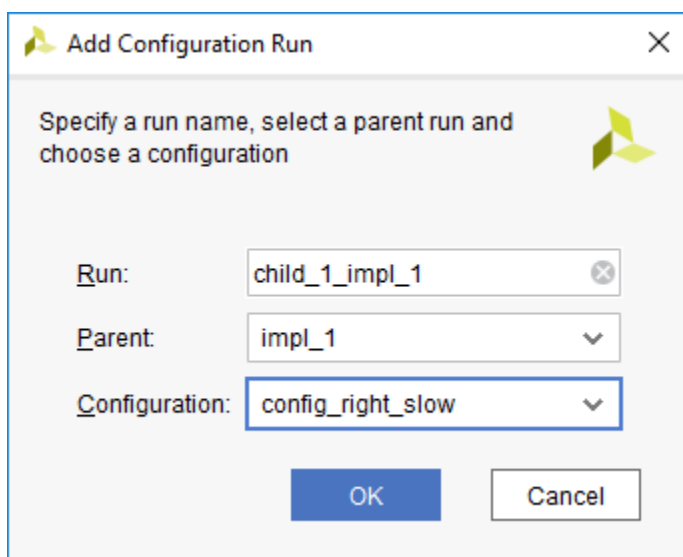


図 34: config\_right\_slow コンフィギュレーションの作成



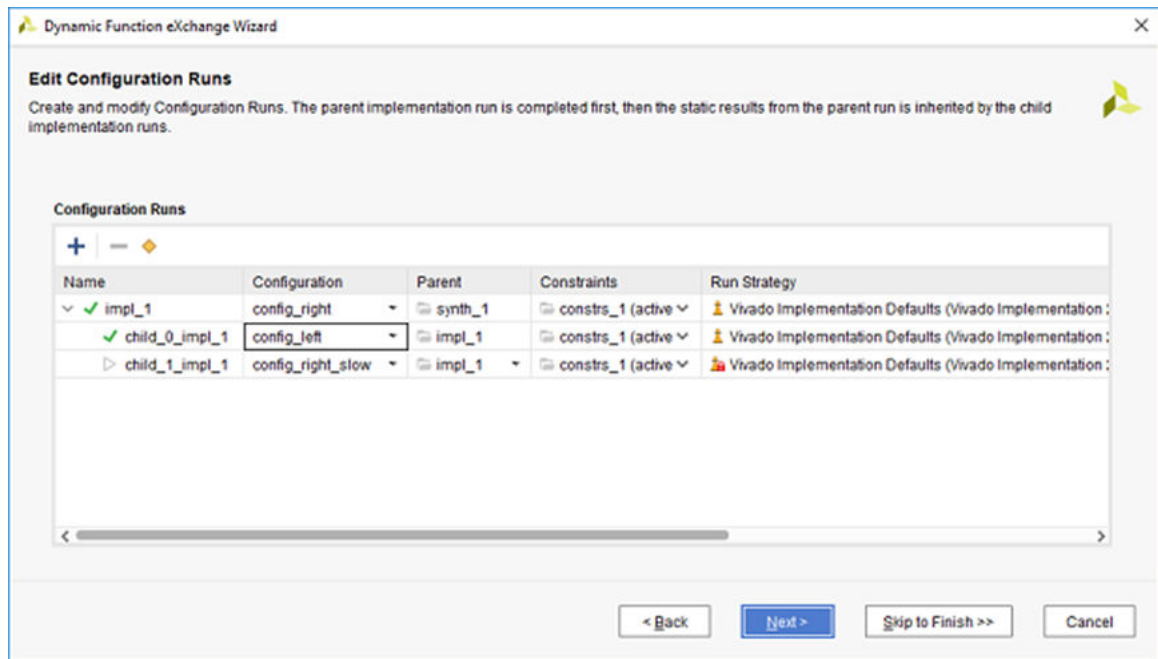
6. [Next] をクリックして [Configuration Runs] に進みます。[+] ボタンをクリックして、次のプロパティを設定し、コンフィギュレーションを新規作成します。
  - [Run]: child\_1\_impl\_1 - 既存のルールに合わせた名前の run です。
  - [Parent]: impl\_1 - このコンフィギュレーションは、既存の親 run の子 run になります。
  - [Configuration]: config\_right\_slow - 先ほど定義した新規 RM のコンフィギュレーションです。
7. [OK] をクリックして、この新しいコンフィギュレーション run を追加します。

図 35: 新しいコンフィギュレーション run の作成



この新しいコンフィギュレーションは、既存の impl\_1 の子で、config\_left の場合と同じように、スタティック デザイン インプリメンテーションの結果を再利用します。この時点で 3 つの run があり、そのうち 2 つが最初の親の子です。緑色のチェック マークは、これらの run のうち 2 つが現在完了していることを示しています。

図 36: 新しく子 run として追加された config\_right\_slow コンフィギュレーション



8. [Next] をクリックし、さらに [Finish] をクリックして、この新規コンフィギュレーション run を構築します。

図 37: 新しく追加された OOC 合成 run およびコンフィギュレーション run

Tcl Console Messages Log Reports Design Runs × Configurations								
<div> <div>🔍</div> <div>⌵</div> <div>⬅</div> <div>⏪</div> <div>⏩</div> <div>➡</div> <div>+</div> <div>%</div> </div>								
Name	Configuration	Constraints	Status	WNS	TNS	WHS	THS	
✓ synth_1 (active)		constrs_1	synth_design Complete!					
✓ impl_1 (active)	config_right	constrs_1	route_design Complete!	8.254	0.000	0.079	0.000	
✓ child_0_impl_1	config_left	constrs_1	route_design Complete!	8.254	0.000	0.079	0.000	
▶ child_1_impl_1	config_right_slow	constrs_1	Not started					
Out-of-Context Module Runs								
✓ shift_right_synth_1		shift_right	synth_design Complete!					
✓ shift_left_synth_1		shift_left	synth_design Complete!					
▶ shift_right_slow_synth_1		shift_right_slow	Not started					

9. この新しく追加した子インプリメンテーション run を選択して右クリックし、[Launch Runs] をクリックします。これで、shift\_right\_slow モジュールに対し OOC 合成が実行され、それからロックされたスタティック デザインのコンテキスト内でこのモジュールがインプリメントされます。

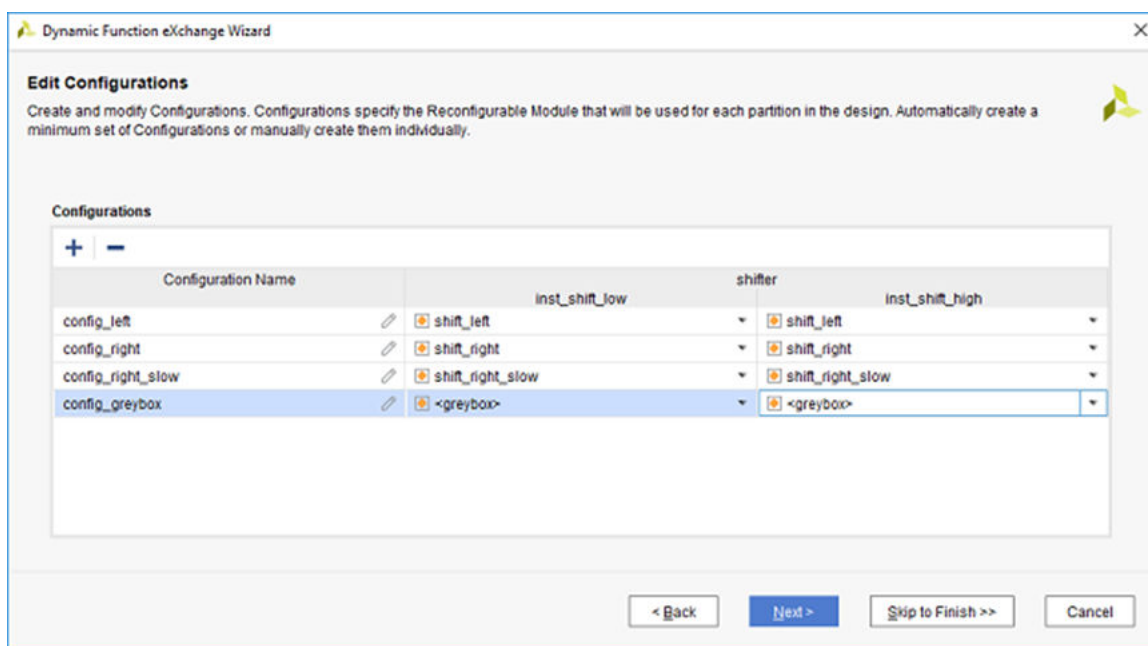
## 手順 6: グレー ボックス モジュールの作成およびインプリメンテーション

デザインによっては、デバイスの目的の初期コンフィギュレーションは、リコンフィギャラブル パーティションに存在する機能がないイメージである可能性があります。または、インプリメントできる RM がない場合もあります。そういうケースでは、グレー ボックス コンフィギュレーションを使用して、実際の RM ネットリストを使用せずに、スタティック デザインのみをインプリメントできます。

グレー ボックスは、初めはブラック ボックス モジュールなのですが、後ですべてのポートに LUT が自動挿入されます。出力ポートは、フロートしないように、ロジック 0 に駆動されます (デフォルトでは 1 で、プロパティで選択可能)。このグレー ボックス モジュールを利用すれば、RM が使用できない場合でも、デザインを処理できます。このグレー ボックス イメージ用のタイミング バジレットを作成するトレーニング スクリプトがあり、スタティック デザインのインプリメンテーション結果を最適化できます。グレー ボックス RM を使用したコンフィギュレーションは親 run にできますが、ほかに RM がなく、RP インターフェイスの配置を最適化するための制約が使用されている場合にのみ推奨します。

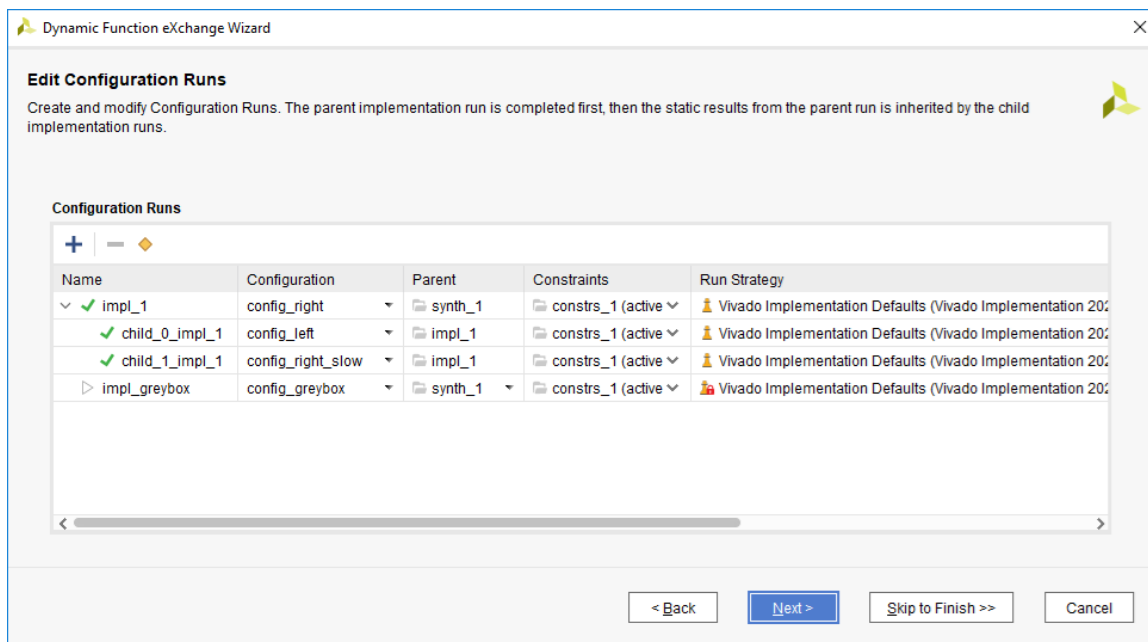
1. Dynamic Function eXchange (DFX) ウィザードを開き、[Configurations] ページに移動します。この場合は専用機能なので、新しく RM を定義する必要はありません。config\_greybox という名前のコンフィギュレーションを新規作成し、各 RP のインスタンスに「<greybox>」と入力します。

図 38: 新規 config\_greybox コンフィギュレーションの追加



2. [Next] をクリックして [Configuration Runs] ページに移動し、もう 1 つコンフィギュレーションを新規作成しますが、今回はグレー ボックス コンフィギュレーション用に作成します。
  - [Parent]: synth\_1 - このコンフィギュレーションを新しい親にし、合成済みの最上位デザインから開始します。
  - [Configuration]: config\_greybox - RM は LUT の接続のみで構成されています。
  - [Run]: impl\_greybox

図 39: 独立したグレー ボックス コンフィギュレーション run の作成



- [Next] をクリックし、さらに [Finish] をクリックして、この run を新規作成します。

[Design Runs] ウィンドウには、インプリメンテーション run が 4 つ、アウト オブ コンテキスト run が 3 つ表示されています。合成は DFX ソリューションのエンベデッド機能なので、グレー ボックス モジュールには合成は不要です。

図 40: 実行準備が整ったグレー ボックス インプリメンテーション

Tcl Console Messages Log Reports Design Runs × Configurations								
Name	Configuration	Constraints	Status	WNS	TNS	WHS	THS	
synth_1 (active)		constrs_1	synth_design Complete!					
impl_1 (active)	config_right	constrs_1	route_design Complete!	8.254	0.000	0.079	0.000	
child_0_impl_1	config_left	constrs_1	route_design Complete!	8.254	0.000	0.079	0.000	
child_1_impl_1	config_right_slow	constrs_1	route_design Complete!	8.254	0.000	0.079	0.000	
impl_greybox	config_greybox	constrs_1	Not started					
Out-of-Context Module Runs								
shift_right_synth_1		shift_right	synth_design Complete!					
shift_left_synth_1		shift_left	synth_design Complete!					
shift_right_slow_synth_1		shift_right_slow	synth_design Complete!					

グレー ボックス コンフィギュレーションは、この時点でインプリメントできます。

- [impl\_greybox] デザイン run を選択して右クリックし、[Launch Runs] をクリックします。この run はアクティブな親ではないので、Flow Navigator ではこの run は起動されません。



**重要:** impl\_1 および impl\_greybox はどちらも親であるため、これらのスタティック デザイン結果は異なります。したがって、生成されるビットストリームの互換性はハードウェアではありません。1 つの親から生成されたビットストリーム (および DFX 検証を使用して後で確認したもの) のみを Dynamic Function eXchange (DFX) を介してデバイスにダウンロードしてください。

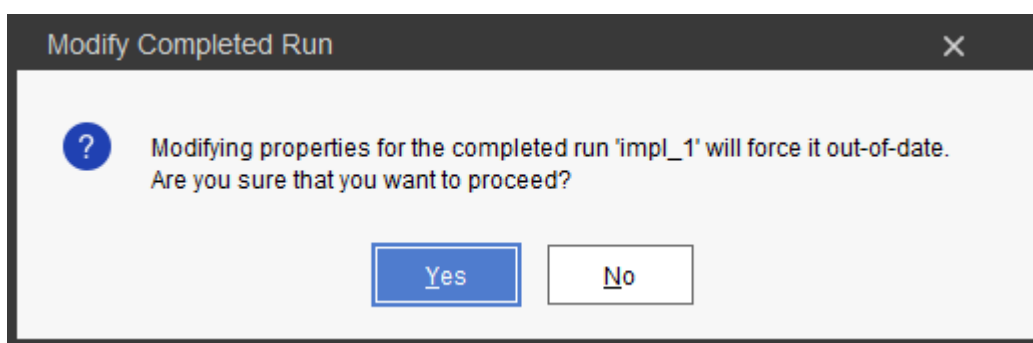
## 手順 7: デザイン ソースまたはオプションの変更

Vivado IDE では、デザイン run 間の依存性が管理されます。コンフィギュレーション間には依存性があるため、この管理機能は Dynamic Function eXchange の重要な機能です。親コンフィギュレーションまたはインプリメンテーションの結果が少しでも変更になると、その子コンフィギュレーションをすべて再コンパイルする必要があります。

1. [impl\_1 design] の run を選択します。
2. [Run Properties] ウィンドウの [Options] タブで、[Strategy] を [Performance\_Explore] に変更します。

このまま作業を続けると jmp 1 のタイムスタンプが古くなる旨を知らせるダイアログボックスが表示されます。

図 41: 完成した run の変更



3. [Yes] をクリックします。

複数の run (impl\_1 およびそれに依存する複数の子 run) のタイムスタンプが古くなっています。結果ファイルはそれぞれのフォルダーにそのまま残っていますが、親 run が起動するとすぐに削除されます。一方、impl\_greybox デザイン run は、impl\_1 に依存していないため、完了した状態のままです。

子 run は親 run のオプションを継承しないので、各子 run の [Strategy] オプションは [Vivado Implementation Defaults] のままです。ただし、スタティック デザインは既に配線およびロックされているので、子 run のストラテジまたはオプションは、RM のインプリメンテーションにのみ影響します。

図 42: 親を変更した後にリセットされるデザイン run

Tcl Console

Messages

Log

Reports

Design Runs

Configurations

🔍

⏮

⏪

⏩

⏭

⏴

⏵

⏶

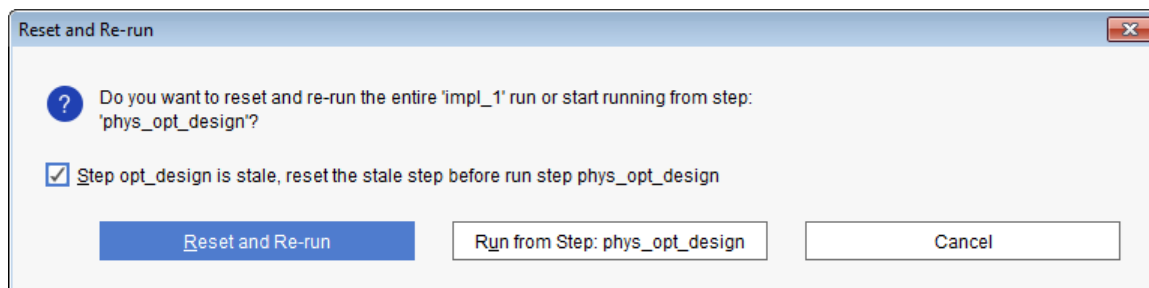
⏷

Name	Configuration	Constraints	Status	WNS	TNS	WHS	THS
✓ synth_1 (active)		constrs_1	synth_design Complete!				
⌵ impl_1 (active)	config_right	constrs_1	Implementation Out-of-date				
✓ child_0_impl_1	config_left	constrs_1	Implementation Out-of-date	8.254	0.0...	0.079	0.0...
✓ child_1_impl_1	config_right_slow	constrs_1	Implementation Out-of-date	8.254	0.0...	0.079	0.0...
✓ impl_greybox	config_greybox	constrs_1	route_design Complete!	8.741	0.0...	0.077	0.0...
📁 Out-of-Context Module Runs							
✓ shift_right_synth_1		shift_right	synth_design Complete!				
✓ shift_left_synth_1		shift_left	synth_design Complete!				
✓ shift_right_slow_synth_1		shift_right_slow	synth_design Complete!				

4. Flow Navigator で、[Run Implementation] をクリックします。

作業を続ける前に run をリセットする必要があるかどうかを確認するダイアログ ボックスが表示されます。親 run の 1 番目のステップが古いステップなので、親 run およびすべての子 run がインプリメンテーションの最初のフェーズに完全にリセットされます。[Reset and Re-run] または [Run from Step: phys\_opt\_design] をクリックして作業を続けます。

図 43: [Reset and Re-run] ダイアログ ボックス



これで 3 つの run がインプリメントされます。まず、parent impl\_1 run が完了し、それから 2 つの子 run が並行処理されます。

## まとめ

Dynamic Function eXchange プロジェクト フローは柔軟性に富み、ユーザーがデザイン環境を管理し、さまざまなオプションを検討できる環境です。1 つのロックされたスタティック イメージから構築された互換性のあるビットストリームのみをターゲット デバイスにダウンロードするため、ユーザーはインプリメンテーション結果およびビットストリームを管理する必要があります。

## 演習 4

# Vivado デバッグおよび DFX プロジェクト フロー

## 概要

この演習では、Dynamic Function eXchange (DFX) ソリューションのプロジェクトベースの機能をさらに学びます。この演習では、次の点について説明します。

- Vivado® IDE での PR のプロジェクト フロー
- リコンフィギャラブル モジュール (RM) 内の現在の IP サポート
- RM 内での Vivado デバッグ コアの挿入
- DFX に特化したレポート機能への改善
- Vivado ハードウェア マネージャー内でのデバッグ

このプロジェクト フローではグレー ボックス インプリメンテーションなどの機能は説明されないで、その点においては演習 3 の DFX フローとは異なりますが、ここでは IP およびデバッグについて説明します。この演習では、次の開発プラットフォームをサポートしています。

- KCU116 (Kintex® UltraScale+™)
- VCU118 (Virtex® UltraScale+™)
- KCU105 (Kintex UltraScale)
- VCU108 (Virtex UltraScale)

---

## 手順 1: チュートリアル デザイン ファイルの抽出

1. ザイリンクス ウェブサイトから [リファレンス デザイン ファイル](#) をダウンロードします。
2. ZIP ファイルの内容を書き込み可能なディレクトリに抽出します。
3. `\dfx_project_debug` に移動します。

---

## 手順 2: 初回デザイン ソースの読み込み

DFX デザイン フロー (プロジェクト ベース) ではまず、デザインのどの部分をリコンフィギャラブルに指定および定義する作業から始めるのが、ほかのフローとは異なります。これには、プロジェクト モードのコンテキスト メニュー [Hierarchical Source View] を使用して定義します。



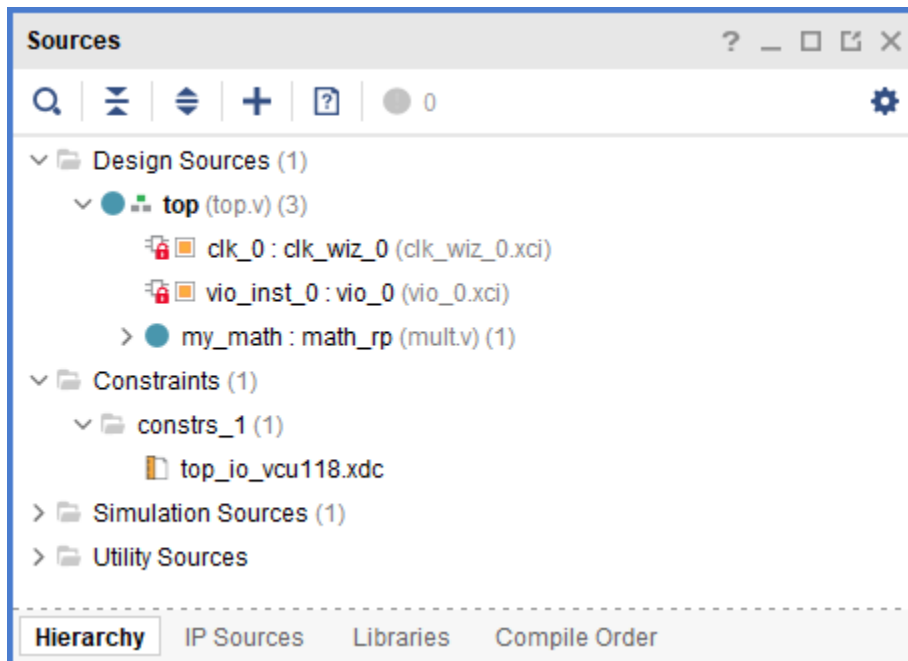
1. TSC アーカイブからデザインを抽出します。dfx\_project\_debug データ ディレクトリは、この演習では `<Extract_Dir>.` と表記されます。
2. Vivado IDE を開き、[Create Project] を選択して、[Next] をクリックします。
3. [Project location] に `<Extract_Dir>` を選択します。プロジェクト名は `project_1` のままに、[Create project subdirectory] をオンのままにしておきます。[Next] をクリックします。
4. [RTL Project] を選択し、[Do not specify sources] チェック ボックスがオフになっているのを確認してから、[Next] をクリックします。
5. [Add Files] ボタンをクリックし、デザインに追加する次のソースを選択します。
  - `<Extract_Dir>\Sources\hdl\top.v`
  - `<Extract_Dir>\Sources\hdl\multiplier\mult.v`
  - `<Extract_Dir>\Sources\ip\<board>\clk_wiz\clk_wiz_0.xci`
  - `<Extract_Dir>\Sources\ip\<board>\vio\vio_0.xci`

`add.v` または `mult_no_ila.v` (それぞれ `adder` および `multiplier_without_ila` フォルダにある) は選択しないでください。これらは RM のソースで、後で追加されます。
6. [Copy sources into project] チェック ボックスを右クリックします。
7. [Next] をクリックして [Add Constraints] に進み、[Add Files] ボタンをクリックし、`<Extract_Dir>\Sources\xdc\top_io-<board>.xdc` ファイルを選択します。
8. [Copy constraints files into project] チェック ボックスをオンにします。
 

**注記:** これらの制約ファイルは、フル デザイン制約で、最上位デザイン用です。この制約ファイルにはフロアプランは含まれていません。
9. パーツを選択するには、[Next] をクリックします。パーツを選択するメニューで [Boards] をクリックし、適切なターゲット プラットフォームを選択します (必要ならフィルターを使用)。
  - Kintex UltraScale KCU105 評価プラットフォーム
  - Virtex UltraScale VCU108 評価プラットフォーム
  - Kintex UltraScale+ KCU116 評価プラットフォーム
  - Virtex UltraScale+ VCU118 評価プラットフォーム
10. それから [Next] をクリックし、[Finish] をクリックしてプロジェクト作成を完了させます。[Sources] ウィンドウにはデザインの標準階層ビューが表示されます。



図 44: プロジェクト作成後の [Sources] ウィンドウ

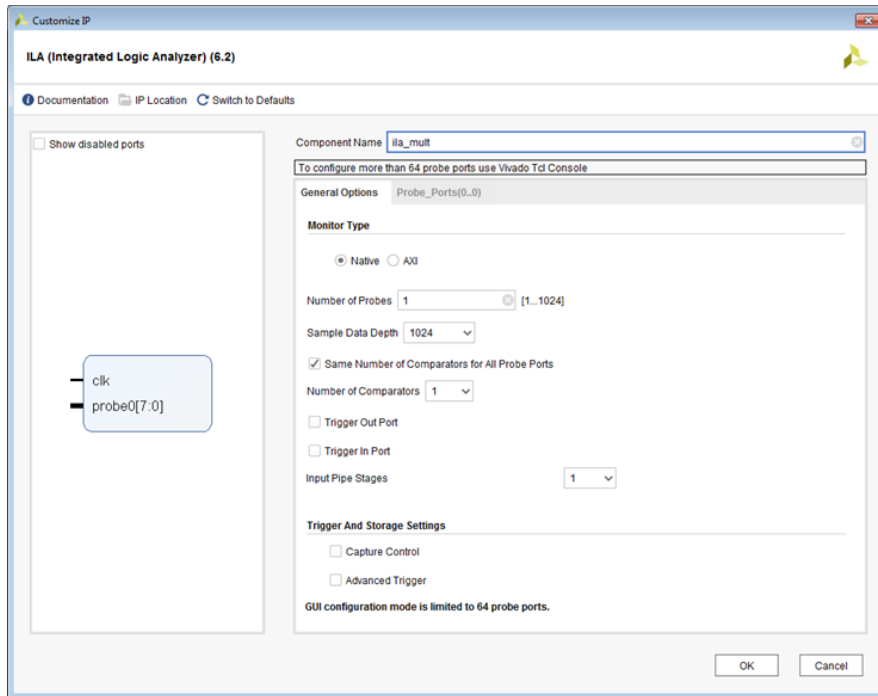


上の図のようにいずれかの IP に赤い鍵型アイコンが表示される場合は、[Reports]→[Report IP Status] をクリックし、IP がアップグレード可能かどうかを確認してください。日付が古くなっている IP ないかを確認したら、[Upgrade Selected] をクリックして、使用可能な最新バージョンのものにアップグレードします。コア コンテナをディスエーブルのままにし、出力ファイルを生成するかどうかを尋ねるメッセージが表示されたら、[Skip] をクリックします。

この時点で、標準プロジェクトが開いています。Dynamic Function eXchange に特化した作業はまだ実行していません。次に ILA コアを追加します。

11. Flow Navigator で、[Project Manager] の下にある [IP Catalog] を開き、[Debug & Verification]→[Debug] をクリックします。
12. [ILA (Integrated Logic Analyzer)] を右クリックし、[Customize IP] をクリックします。[General Options] および [Probe\_Ports(0.0)] タブで次のようにデフォルト設定を変更して、IP をカスタマイズします。
  - [Component Name]: [ila\_mult]
  - [Input Pipe Stages]: [1]
  - [Probe Width of PROBE0]: [8]

図 45: 乗算器用にカスタマイズされた ILA



13. [OK] をクリックしてから、[Skip] をクリックし、IP を作成します。

[Generate] はクリックしないでください。[Synthesis Options] を [Out of context per IP] の設定のままにします。

この IP は `my_math` 階層の下に表示されます。この ILA コアは乗算ファンクションを監視します。これでフルデザイン階層は完了です。

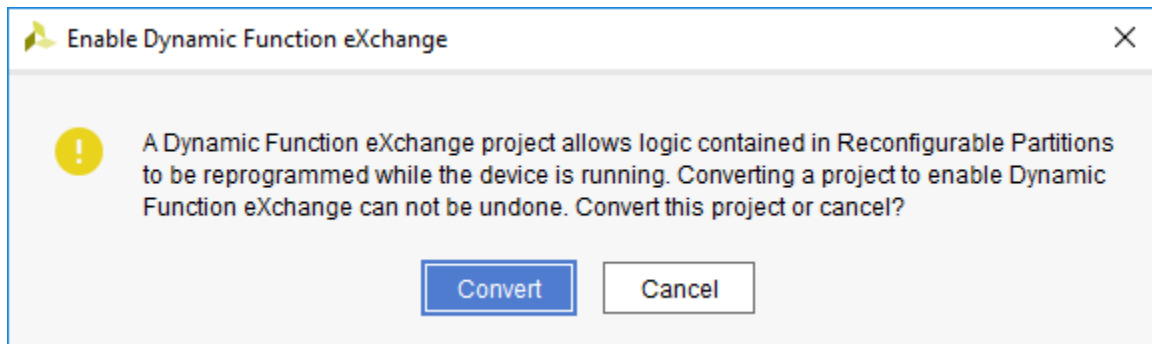
## 手順 3: DFX 用のデザイン設定

1. [Tools] → [Enable Dynamic Function eXchange] をクリックします。

これでプロジェクトが DFX デザイン フロー用になります。この設定は、いったん定義すると元に戻すことはできません。ザイリンクスでは、このオプションを選択する前にプロジェクトのアーカイブをお勧めします。

確認のダイアログ ボックスで [Convert] をクリックしてこのプロジェクトを DFX プロジェクトに変換します。

図 46: Dynamic Function eXchange のイネーブル



2. [Sources] ウィンドウで [my\_math] を右クリックし、[Create Partition Definition...] オプションを選択します。  
これで、このインスタンスがデザインの RM として定義されます。最上位とこのモジュールを分けておくため、アウト オブ コンテキスト合成が実行され、合成後のチェックポイントが math\_rp インスタンスに使用されます。

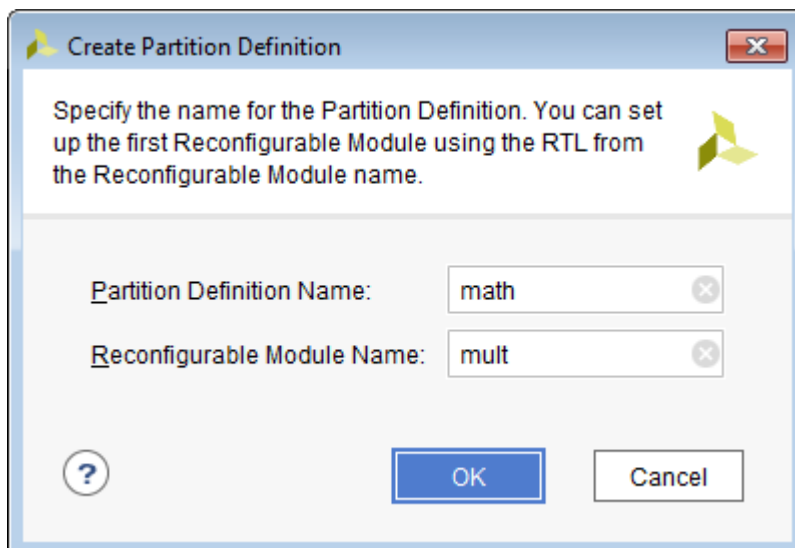


**ヒント:** デザイン内に 1 つのモジュールに複数のインスタンスがある場合、それぞれをリコンフィギャラブルに指定します。どれもリコンフィギャラブルにする必要がない場合は、モジュールが重複しないように手動で変更する必要があります。そうしておく、必要なインスタンスをリコンフィギャラブル パーティション (RP) として個別にタグ付けできます。

**注記:** リコンフィギャラブル モジュール内の IP は、グローバルまたはアウト オブ コンテキストで合成できます。この演習では、ILA IP をデフォルトのアウト オブ コンテキストのままにしておきます。

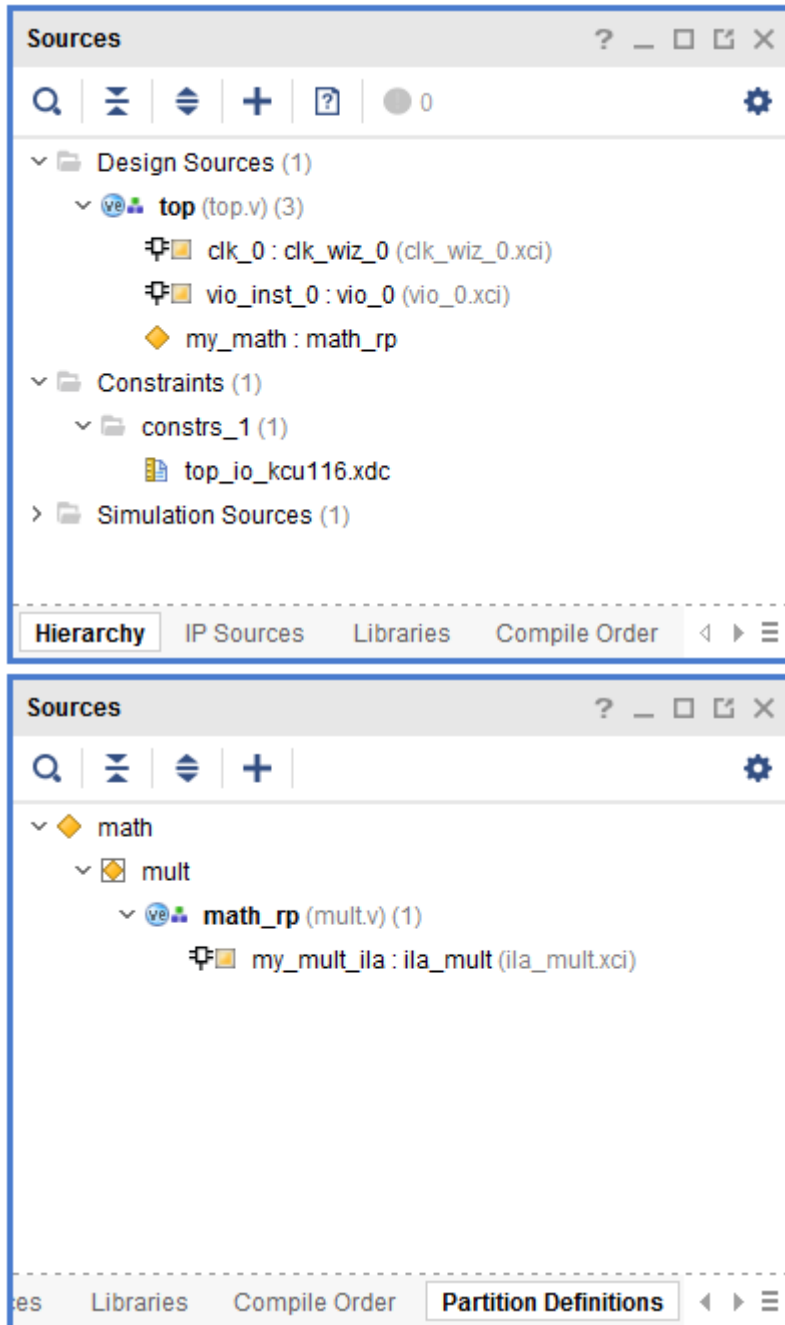
3. 表示されるダイアログ ボックスで、[Partition Definition Name] および [Reconfigurable Module Name] の両方に名前を指定します。  
  
このパーティション定義は、すべての RM が挿入されるワークスペースを一般的に参照するためのものなので、「math」など適切な名前を付けます。  
  
一方 RM は、この特定の RTL インスタンスを参照するので、「mult」など機能がわかるような名前を付けて、[OK] をクリックします。
4. [OK] をクリックします。

図 47: math パーティション定義の作成



[Sources] ウィンドウがこれで少し変更され、my\_math インスタンスはパーティションであることがわかるように、その横に黄色いひし形が表示されます。また、このウィンドウの [Partition Definitions] タブに、デザインのすべてのパーティション定義のリストおよび内容が表示されているのも確認できます (この場合は 1 つだけ)。さらに、mult モジュールを合成するため、アウト オブ コンテキスト モジュール run が作成されています。

図 48: math パーティション定義後の [Sources] ウィンドウ



この時点で、Dynamic Function eXchange ウィザードを使用して新しく RM を追加できるようになります。



**重要:** パーティションを定義した後に追加する RM はすべて、DFX ウィザードを使用して追加する必要があります。RM ソース、コンフィギュレーション、run の管理にも、このウィザードを使用する必要があります。

## 手順 4: DFX ウィザードを使用してデザインの残りの部分を完成

1. Flow Navigator または [Tools] メニューから Dynamic Function eXchange Wizard をクリックし、DFX ウィザードを開きます。
2. [Next] をクリックして [Edit Reconfigurable Modules] ページに進みます。mult RM が既にあり、ウィンドウの左側に追加、削除、編集ボタンがあります。[+] アイコンをクリックし、新しく RM を追加します。
3. [Add Files] ボタンをクリックして、最上位の加算器を選択します。

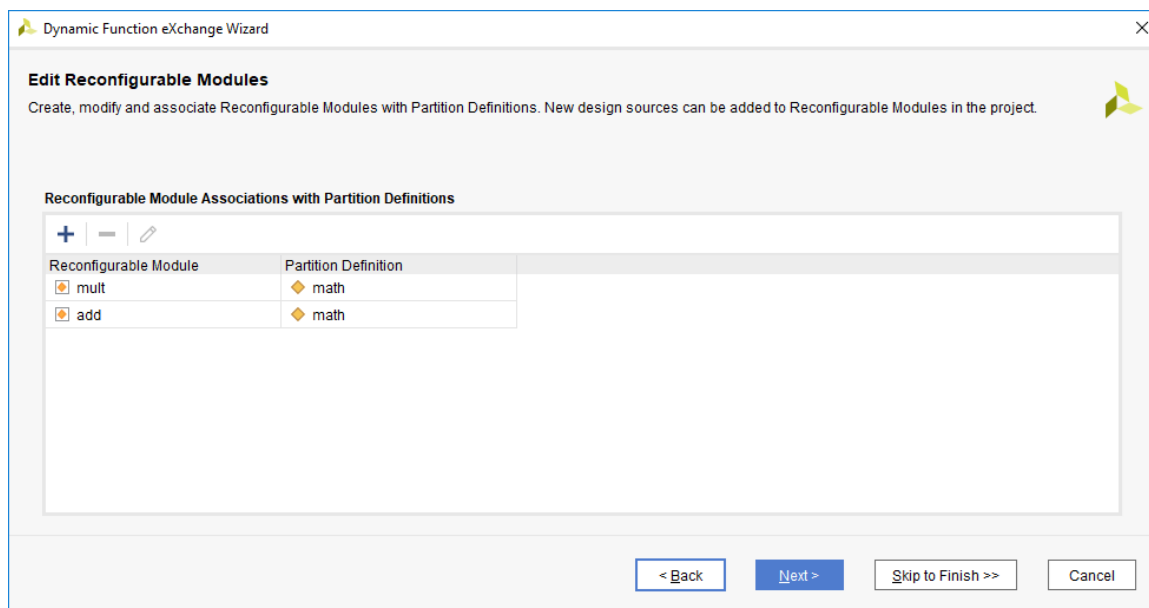
```
<Extract_Dir>\Sources\hdl\adder\add.v
```

モジュール レベルの制約が必要な場合は、ここで追加します。その場合、制約をこのパーティションの階層に合わせる必要があります。

4. [add.v] をダブルクリックするか、1 回クリックをしてから [OK] をクリックします。
5. [Reconfigurable Module] フィールドに「[add]」と入力します。[Partition Definition] を [math] に設定し、[Top Module] を空白のままにし、[Sources are already synthesized] チェック ボックスをオフにします。[Copy sources into project] チェック ボックスを右クリックします。[OK] をクリックし、この新しいモジュールを作成します。

これで、math RP に対し、RM が 2 つ使用できるようになりました。

図 49: RM を 2 つ定義した DFX ウィザード



6. [Next] をクリックしてコンフィギュレーションを定義します。

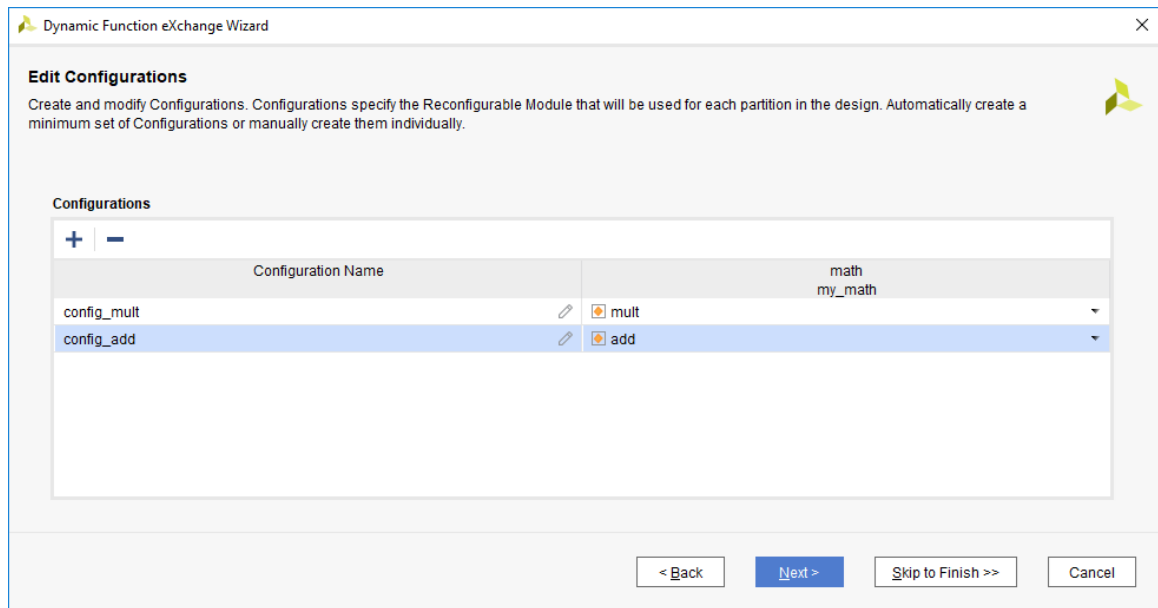
7. ここでは、[automatically create configurations] リンクを選択して、コンフィギュレーションをウィザードで自動作成します。

コンフィギュレーションはフル デザインのイメージで、スタティック デザイン、および RP ごとに 1 つの RM から構成されています。前段階でやったように、必要な数のコンフィギュレーションを作成することもできますし、単にウィザードに任せてコンフィギュレーションを自動選択することもできます。

最低 2 つのコンフィギュレーションがセットになって作成されます。math インスタンスは 1 つ目のコンフィギュレーションで mult に、2 つ目のコンフィギュレーションで add に渡されます。

**注記:** [Configuration Name] は編集できるようになっており、各コンフィギュレーションに含まれる RM を反映させるため、コンフィギュレーション名が config\_mult および config\_add にアップデートされています。

図 50: 自動生成された最小限のコンフィギュレーション セット (名前を変更)



追加コンフィギュレーションは、必要であれば、これら 2 つの RM を使用して作成できます。グレー ボックス (LUT が接続されているブラック ボックス) コンフィギュレーションも選択可能ですが、この機能はこの演習では使用しません。

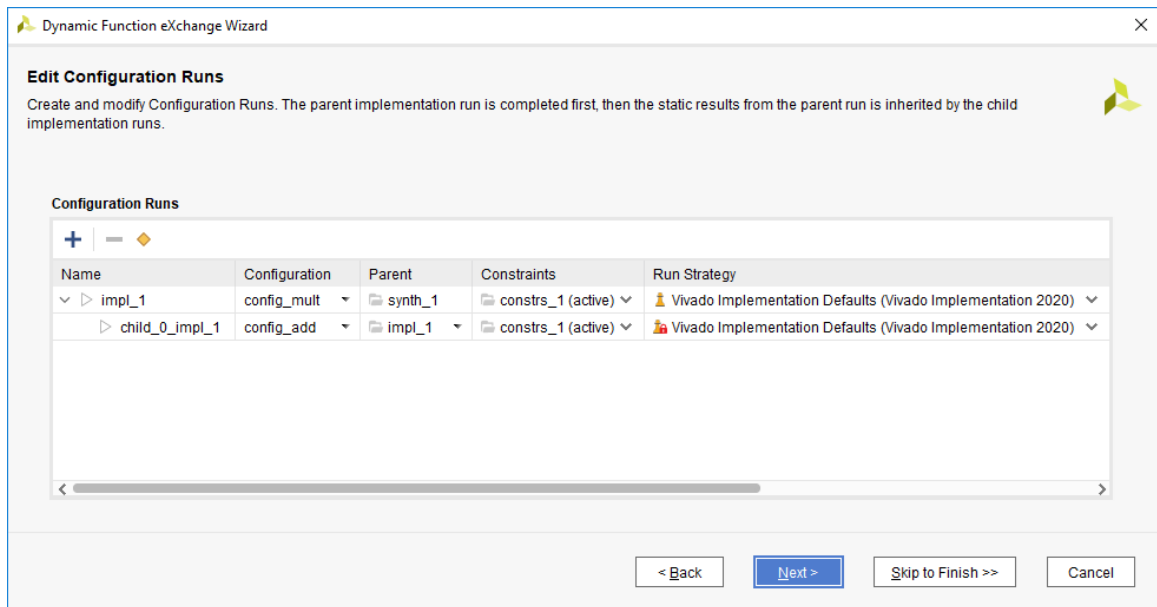
8. [Next] をクリックして [Edit Reconfigurable Runs] ページに進みます。

コンフィギュレーションの場合と同じように、各コンフィギュレーションをインプリメントするのに使用される run が手動または自動で作成できます。階層の親子関係で run の関わり合い方が定義されます。まず親 (上位) run で、スタティック デザイン、およびそのコンフィギュレーション内のすべての RM がインプリメントされます。それから子 run で、ロックされたスタティック デザインが再利用され、同時に、その確立されたコンテキストでそのコンフィギュレーション内で RM がインプリメントされます。

9. [automatically create configuration run] リンクをクリックすると、[Configuration Runs] ページに最低数の run が自動入力されます。

これで、親コンフィギュレーション (config\_mult) 1 つ、および子コンフィギュレーション (config\_add) 1 つの run が 2 つ作成されます。独立した run でも関連している run でも、このウィザード内で、異なるストラテジーや制約セットを使用し、必要な数だけ run を作成できます。ここでは、run はこの 2 つだけにとどめておきます。これらの run の名前は編集できないので注意してください。

図 51: コンフィギュレーション run の最小セットを自動生成



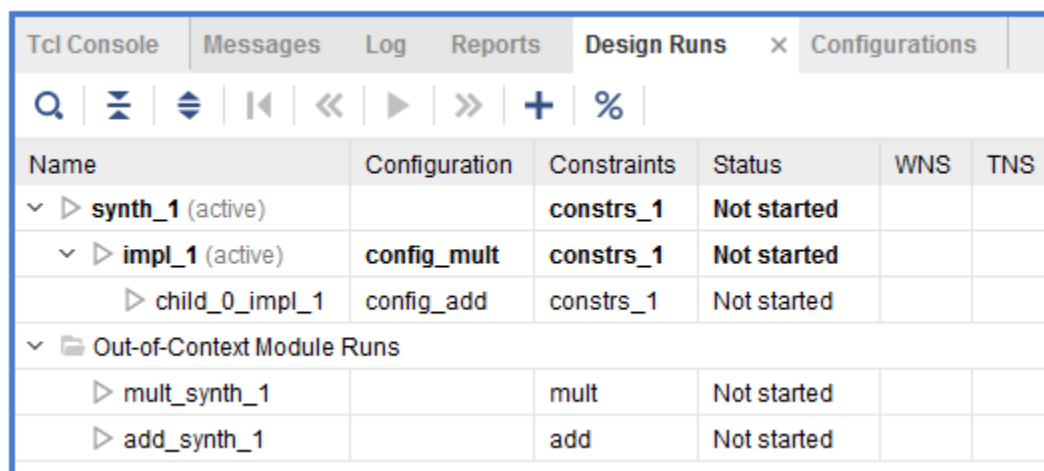
10. [Next] をクリックして、[Summary] ページを確認し、[Finish] をクリックして、デザイン設定を完了させ、ウィザードを終了します。



**重要:** [Finish] をクリックして DFX ウィザードを終了するまでは、何も作成されず、変更も反映されません。[Finish] ボタンをクリックするまで、それまでの作業内容はすべてキューに残っているため、変更部分をインプリメントしないで、ウィザードの前のページに戻って、納得がいくまで設定を変更することが可能です。

Vivado IDE に戻って、[Design Runs] ウィンドウがアップデートされています。math RM に対し 2 番目のアウトオブ コンテキスト合成 run が追加され、子インプリメンテーション run (child\_0\_impl\_1) が親 (impl\_1) の下に作成されます。

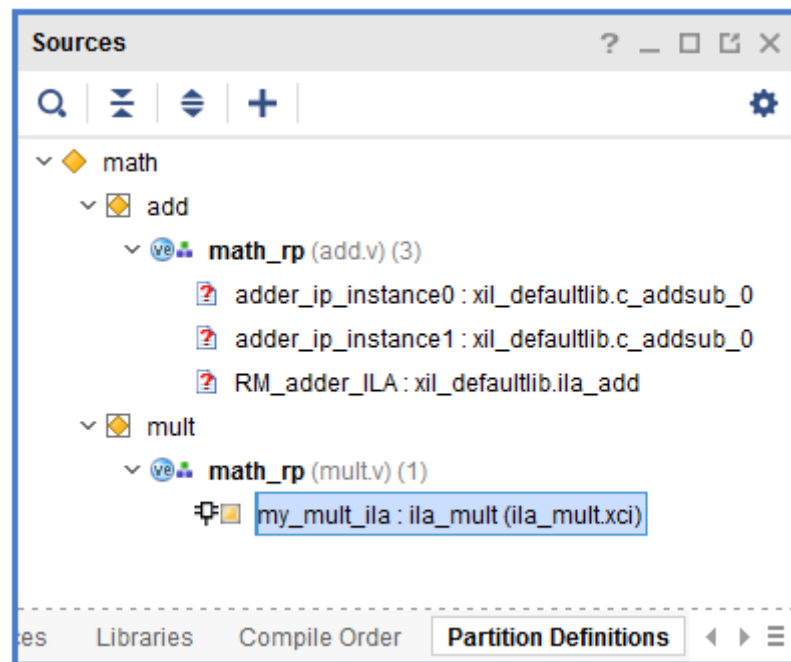
図 52: 合成およびインプリメンテーションがすべて起動できる状態になった [Design Runs] ウィンドウ



## 手順 5: リコンフィギャラブル モジュール内での IP の追加

[Partition Definitions] タブに戻り、追加された RM math\_rp を展開して、3 のサブモジュールにクエスチョン マークのアイコンが付いていることを確認します。このアイコンが表示されているのは、機能を完了させるためにこれら 3 つのサブモジュールを追加する必要があるからです。

図 53: 追加されていないソースが表示されている [Partition Definitions] タブ

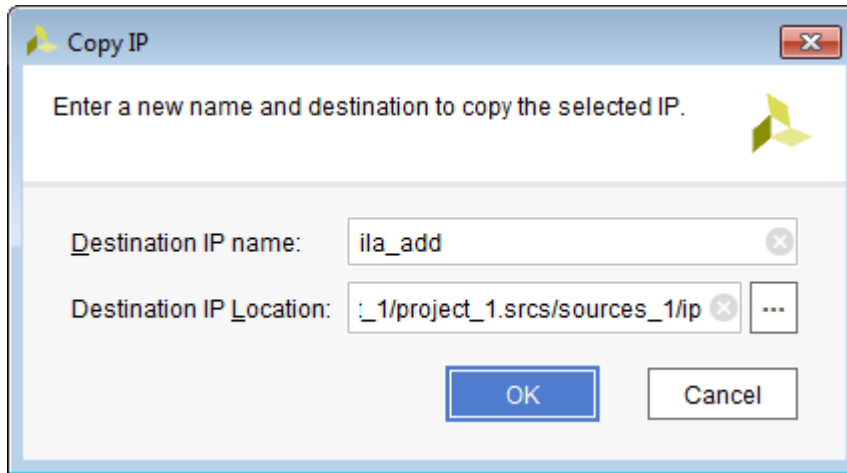


この 3 つのサブモジュールはすべて IP です。各 RM 内で IP インスタンスを重複させることはできないので、同じ ILA コアのインスタンスはスタティックのものでも、別の RM のものでも使用できません。

1. ila\_mult RM 内にある [[ila\_mult]] インスタンスを右クリックし、[Copy IP] をクリックします。
2. [Destination IP Name] を「ila\_add」に設定し、[Destination IP Location] はそのままにして、[OK] をクリックします。



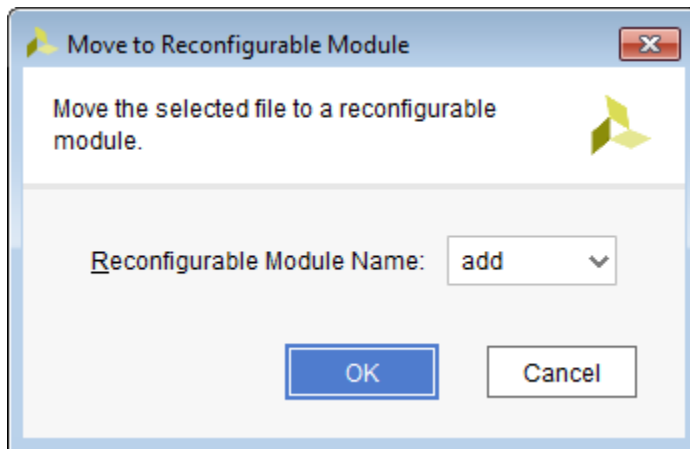
図 54: math から ILA をコピーして追加



コピーされた IP はメインのデザイン ファイルセットとして [Sources] ウィンドウの階層に配置されるので、add RM ブロックセットに移動させる必要があります。

3. [Hierarchy] タブで [ila\_add] インスタンスを右クリックし、[Move to Reconfigurable Module] をクリックします。[add] RM を選択して、[OK] をクリックします。

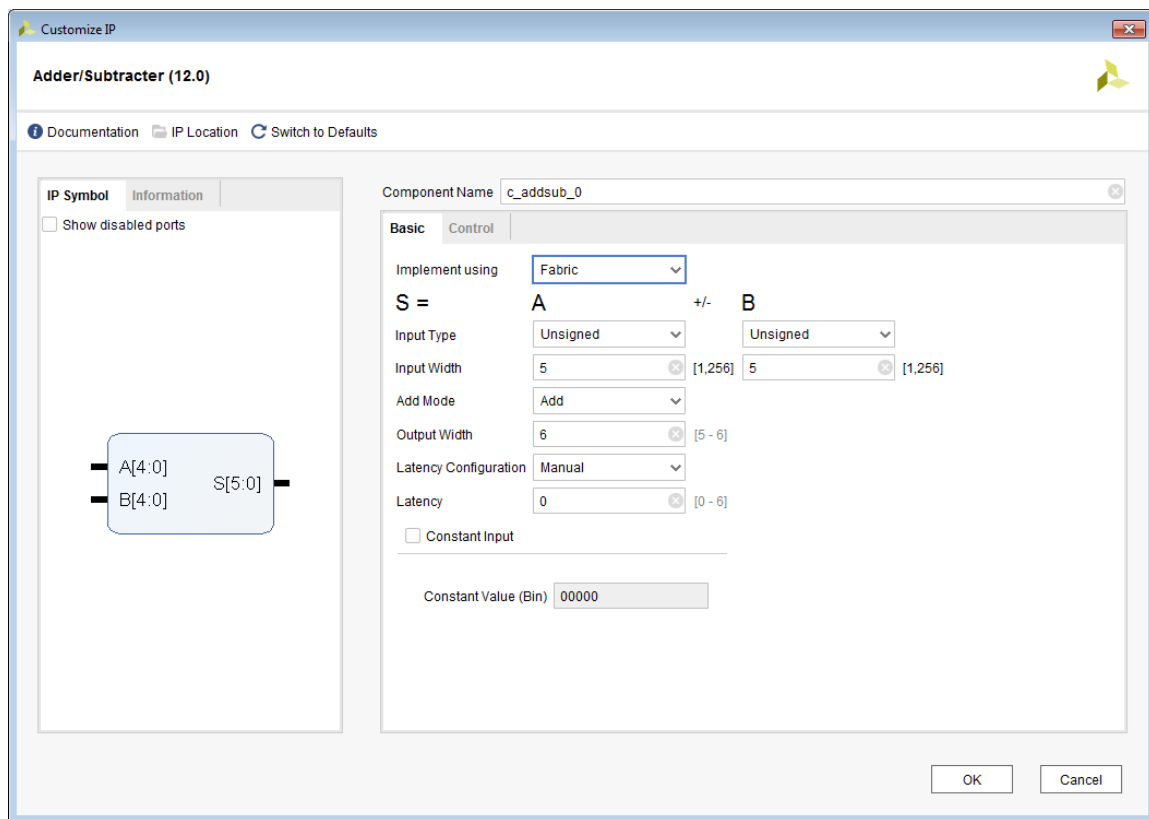
図 55: IP を RM へ移動



[Partitions Definitions] タブに戻ると、ILA IP インスタンスがちゃんと add RM の下に移動しているのが確認できます。

4. IP カタログを開き、「add」入力して Adder/Subtractor IP を検索します。この IP を開き、名前は c\_addsub\_0 のままで、デフォルト以外のオプションを指定して IP をカスタマイズします。
  - [Input Type]: [Unsigned] (A および B の両方)
  - [Input Width]: 5 (A および B の両方)
  - [Output Width]: 6
  - [Latency]: 0
  - [Control] タブで [Clock Enable] をオフ
5. [OK] をクリックします。

図 56: Adder/Subtractor IP のカスタマイズ



6. [Skip] をクリックして IP の生成を完了します。

ILA IP の場合と同様に、これもメインのソース セットに追加されているので、同じ手順に沿って add RM に移動させます。

7. [Sources] ウィンドウの [Hierarchy] タブで [c\_addsub\_0] インスタンスを右クリックし、[Move to Reconfigurable Module] をクリックします。

[add] RM を選択して、[OK] をクリックします。

この IP は add RM 内の両方の加算器ファンクション インスタンスに対して使用されます。この時点で、デザイン全体が読み込まれているので、インプリメンテーションに進む準備が整いました。

## 手順 6: デザインの合成およびフロアプランの作成

合成を起動する前に、Vivado デバッグ ソリューションのデバッグ ハブを挿入する命名規則を確認します。

1. mult.v を開き、その中のポート リストを確認します。

このポート リストには S\_BSCAN\_ から始まって、12 個のポートが含まれています。これらのポートは、デザインのスタティック部分とリコンフィギャラブル部分に挿入されているデバッグ ハブの接続に使用されます。これらのハブは自動的に挿入されます。ポート リストがこの命名規則に一致している限りは、接続は自動的に実行されます。



**注意:** 正しく推論するためには、正確なポート名を使用する必要があります。ポート名が少しでも違う場合、新しい信号名をデバッグ プロパティに割り当てるため、RTL ファイルのコメントにある属性を使用する必要があります。

前段階で用意したデザインを Vivado IDE を開き、[Design Runs] ウィンドウを確認します。最上位デザイン合成 run (synth\_1) および親インプリメンテーション run (impl\_1) は「アクティブ」になっています。Flow Navigator での作業内容は、これらのアクティブな run に適用されるので、[Run Synthesis] または [Run Implementation] をクリックすると、これらの run だけが実行され、また、その完了に必要な OOC 合成 run まで実行されます。子インプリメンテーション run を選択し、右クリックして [Launch Runs] を選択すれば、フロー全体を実行できますが、ここでは合成を別に実行します。

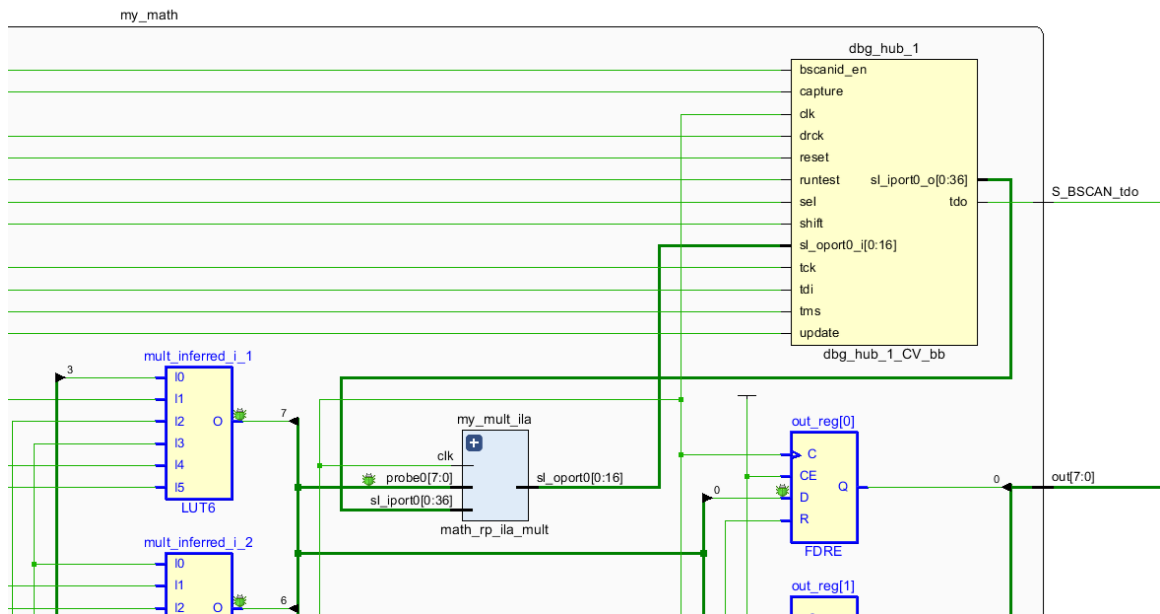
2. Flow Navigator で [Run Synthesis] をクリックします。合成が完了したら、[Open Synthesized Design] を選択します。

これで、すべての OOC モジュールが合成され、それから最上位の合成が実行されます。OOC モジュール (IP であってもなくても) を含んだデザインであればすべて、この同じように実行されます。

3. 合成中に挿入されたインスタンスを確認するため、回路図を開きます。

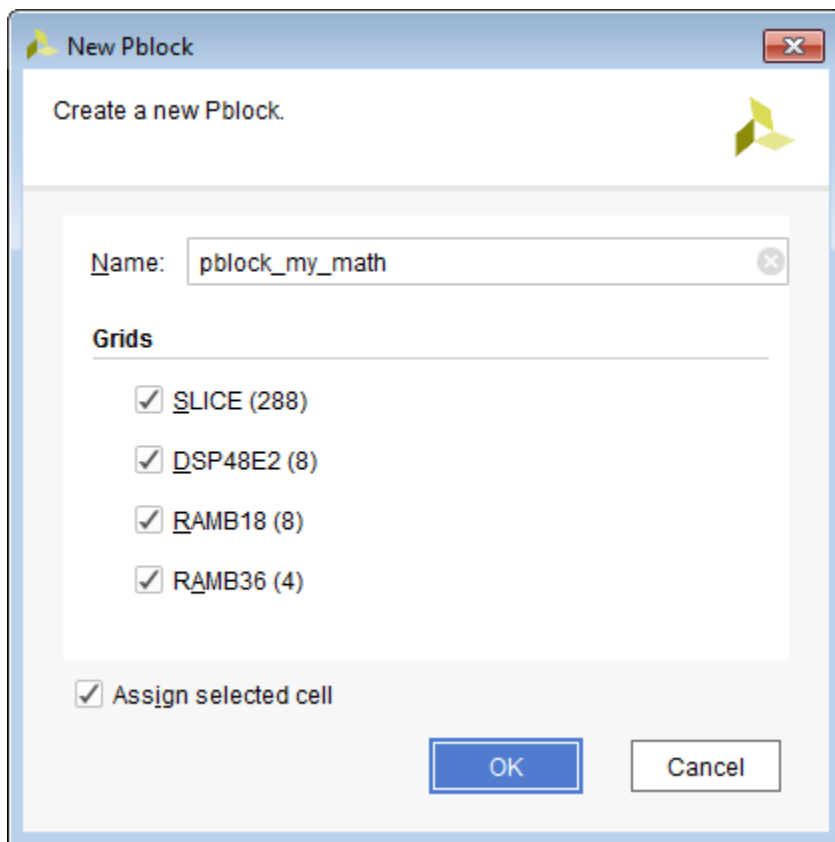
最上位デザインに dbg\_hub インスタンスが挿入されているのがわかります。その sl\_\* ポートが VIO デバッグ コアに最上位で接続されています。次に、その下にある my\_math の階層に移動すると、別の dbg\_hub インスタンスが挿入されていて、sl\_\* ポートがそのモジュールの ILA デバッグ コアに接続されています。この RM は乗算器で、これはアクティブな親 run の回路図ビューであることに注意してください。

図 57: mult RM 内に挿入された dbg\_hub\_1



4. [Layout]→[Floorplanning] をクリックして、Vivado をフロアプランニング モードにします。[Netlist] ウィンドウで [my\_math] インスタンスを右クリックし、[Floorplanning]→[Draw Pblock] をクリックします。適当なときに Pblock を作成します。ダイアログ ボックスが表示されるので、「pblock\_my\_math」という名前をそのまま使用し、リソース タイプには、SLICE、DSP およびブロック RAM のみをオンにしておきます。

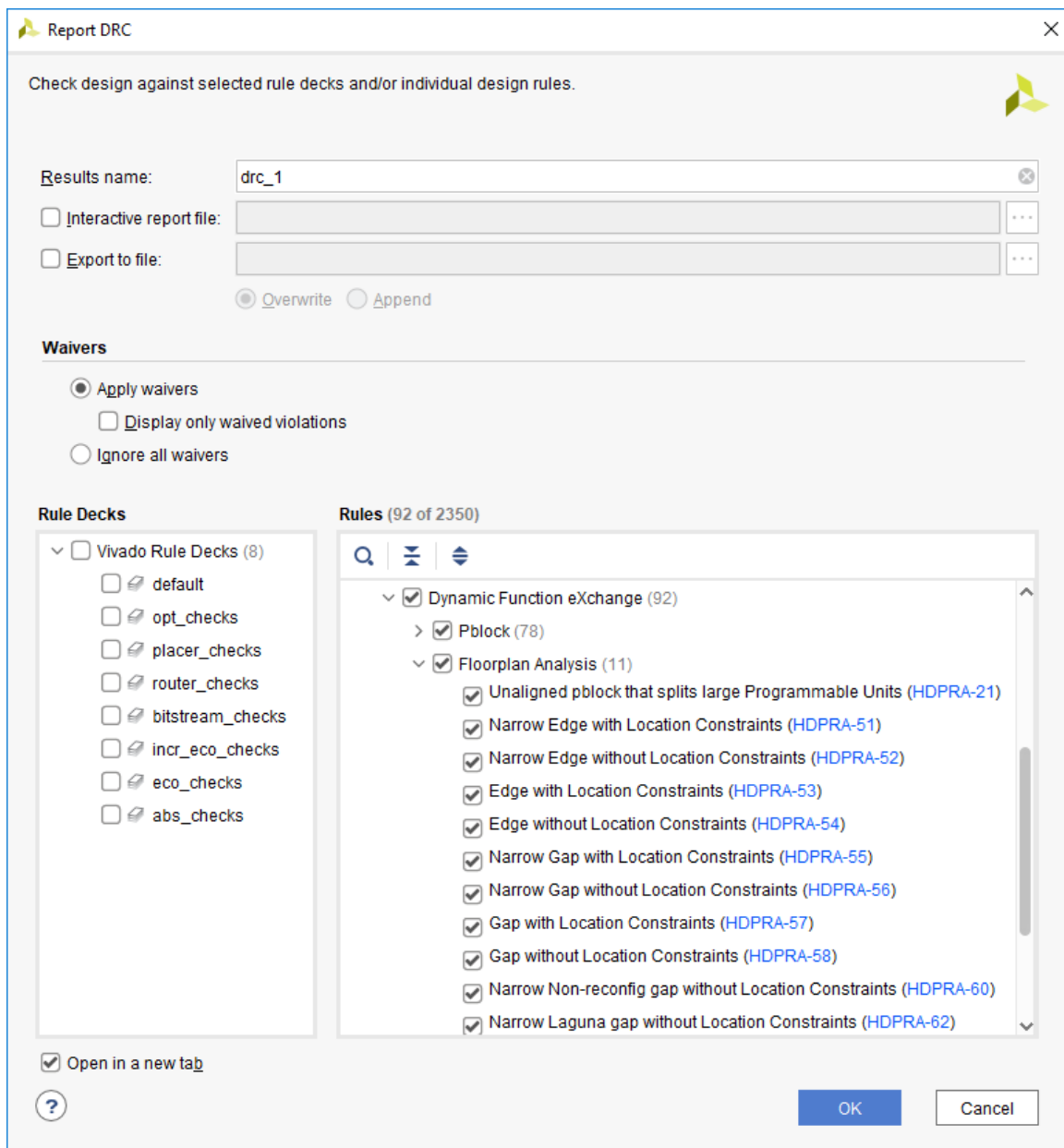
図 58: math RP の Pblock を描画



選択した領域内に指定したタイプのリソースが足りない場合は、[Pblock Properties] の [Statics] タブに不足しているリソース タイプが赤く表示されます。適宜調整してから、フロアプランを保存します。各 RM には、ブロック RAM を必要とする ILA コアが 1 つ含まれていることを思い出してください。また、このデザインには制御セット数が多いので、必要な領域が予想以上に大きくなる可能性があります。エリアには CLB が最低 3000 個、ブロック RAM の列が 1 つあるぐらいがよいでしょう。

5. [Reports] → [Report DRC] をクリックして、DFX に特化したデザイン ルール チェックを実行します。時間を節約するには、Dynamic Function eXchange のチェック ボックスだけでなく、すべてのチェック ボックスをすべてオフにできます。[OK] をクリックして、実行します。

図 59: DFX DRC の確認



エラーが表示された場合は、それを修正します。デバイスによっては、Pblock の質を改善する方法を提案するアドバイザー メッセージが表示される可能性があります。



**ヒント:** DFX のデザイン ルール チェックを早い段階で、頻繁に実行します。

6. ウィンドウ上部のツールバーで [Save Constraints] をクリックして、制約を保存します。

## 手順 7: DFX コンフィギュレーション解析レポートの実行

DFX コンフィギュレーション解析ツールは、選択した各 RM を比較し、DFX デザインに関する情報を提供します。リソース使用量、フロアプラン、クロッキング、タイミング メトリクスが解析されるので、DFX デザイン全体を管理するのに役立ちます。DFX コンフィギュレーション解析ツールは、[Tcl Console] ウィンドウまで実行されます。

1. Tcl コンソールで、cd コマンドを使用して、プロジェクト ディレクトリに移動します。次のコマンドを入力して、このデザインで使用可能な 2 つの RM でレポートを生成します。

```
report_pr_configuration_analysis -cells my_math -dcps
{./project_1.runs/add_synth_1/math_rp.dcp ./project_1.runs/mult_synth_1/
math_rp.dcp}
```

**注記:** プロジェクト名が「project\_1」でない場合は、Tcl コマンドで正しいプロジェクト名を指定する必要があります。

これで、デフォルト設定で解析が実行され、次にリストする上位 3 点に関するデータが収集されます。3 点に絞ることが可能であることを確認するには、-help オプションを使用します。

- -complexity オプション: RP に必要なリソース タイプの最大数など、リソース使用量をレポートします。
- -clocking オプション: 各 RM のクロック使用率およびロードに関する情報をレポートします。
- -timing オプション: 境界インターフェイスのタイミングの詳細をレポートします。
- -rent オプション: レポートにレント メトリクスを追加しますが、実行に時間がかかる可能性があります。
- -file オプション: レポートをファイルに保存します。

Tcl コンソールでレポートを確認すると、セクション 2 に複雑性に関するまとめがあります。現在の RM (乗算器)、RM 1 および 2 (加算器および乗算器)、および最大数を示す列が表示されています。この表で、各モジュールのリソース使用率およびその最大数を確認できるので、それを確認しながら、適切な Pblock を作成できます。

RM 1 および RM 2 のリソース数が少なそうである点に注目してください。ログのレポートには、次のようなクリティカル警告がいくつか表示されています。

```
CRITICAL WARNING: [Project 1-486] Could not resolve non-primitive
black box cell 'math_rp_c_addsub_0' instantiated as
'adder_ip_instance0'
```

合成後のチェックポイントはアウト オブ コンテキストで生成されているので、サブモジュール IP は含まれていません。各 RM の全体像を確認するには、これらの下位チェックポイントをリンクさせるか、または IP をグローバルに設定して合成する必要があります。

2. [Partition Definitions] タブで、階層を展開させます。[my\_mult\_ila] IP を右クリックして、[Generate Output Products] をクリックします。
3. [Synthesis Options] を [Global] に設定し、[Apply] をクリックしてから [Cancel] をクリックします。
4. 加算器モジュールの下にある my\_add\_ila および adder\_ip の両方の IP に対してこの作業を繰り返します。adder\_ip にはインスタンスが 2 つありますが、どちらも同じインスタンスなので、この作業は一度だけで十分です。
5. これらのモジュールの合成のタイムスタンプがこれで古くなります。Flow Navigator で [Run Synthesis] をクリックします。すべての run をリセットし再合成するかどうかを尋ねるダイアログ ボックスが表示されるので、[OK] をクリックします。
6. 合成が完了したら、report\_pr\_configuration\_analysis コマンドを手順 1 から再実行し、ログおよび結果値を確認します。

## 手順 8: デザインのインプリメント

1. Flow Navigator で [Run Implementation] を選択し、すべてのコンフィギュレーションで 配置配線を実行します。

まず impl\_1 のインプリメンテーション、次に child\_0\_impl\_1 のインプリメンテーションが実行されます。すべての DFX 要件が揃った状態でこの 2 つの run で配置配線が実行されるだけでなく、DFX に特化したタスクがさらに 2、3 実行されます。impl\_1 が完了したら、Vivado により次の作業が自動的に実行されます。

- 配線済みの乗算器 RM のモジュール レベル (OOC) のチェックポイントが生成されます。
- スタティックのみのデザイン イメージを作成するため、RP のロジックが抜き出されます。この RP インスタンスに update\_design -black\_box が呼び出されます。
- デザインのスタティック部分のみの配置配線がすべてロックされます。そのため、lock\_design -level routing が呼び出されます。
- すべての子 run に再利用できるよう、ロックされたスタティックの親イメージが保存されます。

さらに、子 run が完了すると、モジュール レベルのチェックポイントが配線済みの加算器 RM 用に作成されます。ロックされたスタティック デザインのイメージは、親と同じなので、このステップは不要です。

Vivado プロジェクトの場合、依存関係は Vivado IDE で管理されます。ソースが変更されると、対象となる run のタイムスタンプが古くなります。つまり、親子関係がある場合は、これらのチェックでその依存関係が理解される必要があります。たとえば、add.v が変更された場合、OOC 合成 run およびその子インプリメンテーション run のみのタイムスタンプが古くなります。

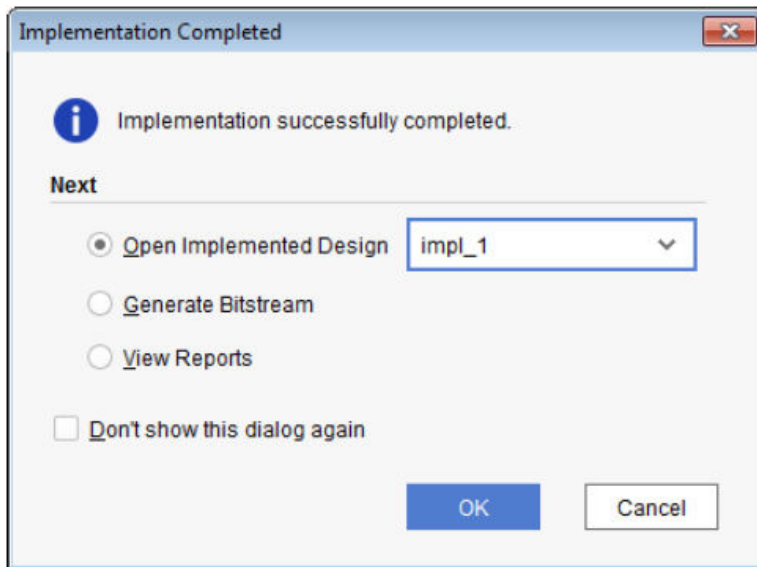
特定のコンフィギュレーション run のみが必要な場合は、[Design Runs] ウィンドウ内でそれらを個別に選択できます。子 run は親 run からのロックされたスタティック デザインをインポートして開始するため、子 run を起動する前に親 run を完了させておく必要があります。

図 60: 両方のコンフィギュレーションが配線された状態

Tcl Console	Messages	Log	Reports	Design Runs	I/O Ports	Timing	Power	Methodology
<div> <span>🔍</span> <span>🔧</span> <span>🔄</span> <span>⏪</span> <span>⏩</span> <span>⏴</span> <span>⏵</span> <span>+</span> <span>%</span> </div>								
Name	Configuration	Constraints	Status	WNS	TNS	WHS		
✓ synth_1 (active)		constrs_1	synth_design Complete!					
✓ impl_1 (active)	config_mult	constrs_1	route_design Complete!	6.052	0.0...	0.013		
✓ child_0_impl_1	config_add	constrs_1	route_design Complete!	5.771	0.0...	0.013		
Out-of-Context Module Runs								
✓ mult_synth_1		mult	synth_design Complete!					
✓ add_synth_1		add	synth_design Complete!					

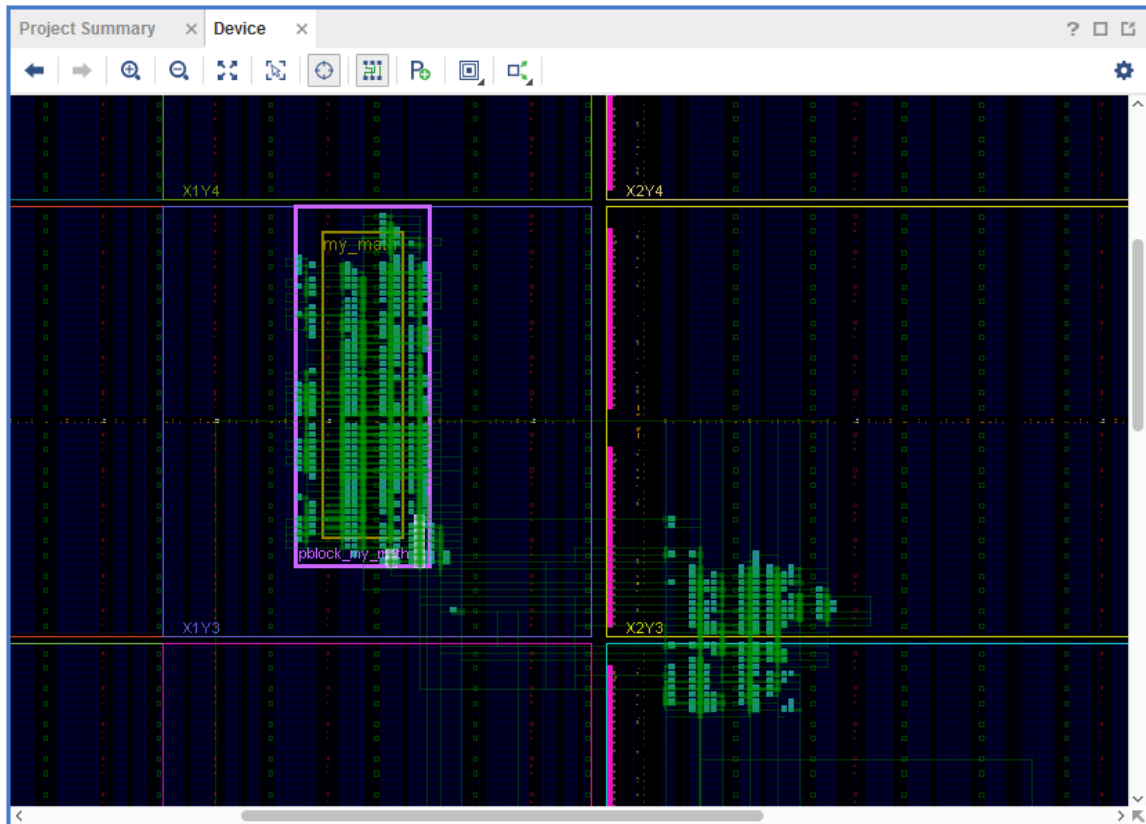
2. インプリメンテーション run が完了したら、[Open Implemented Design] をクリックし、[OK] をクリックします。

図 61: impl\_1 コンフィギュレーションを開く



**注意:** デザインが子インプリメンテーション run まで処理されていても、[Open Implemented Design] を選択すると、デフォルトで親 run が開きます。プルダウン メニューを使用して、目的のインプリメンテーション run を選択します。

図 62: 配線済み親デザインの [Device] ビュー





これは乗算器コンフィギュレーションの配線済みデザインです。次に、配置配線エリアのフレームセットを確認します。

3. Tcl コンソールで、cd コマンドを使用し、現在のプロジェクト ディレクトリに移動します (既にそのディレクトリで作業している場合は次のステップに進みます)。次のコマンドを実行して、可視化スクリプトを実行します。

```
source project_1.runs/impl_1/hd_visual/  
pblock_my_math_Routing_AllTiles.tcl  
highlight_objects -color yellow [get_selected_objects]
```

最初の Tcl スクリプトで、デザインのリコンフィギュラブル部分を配線するのに有効なフレームが特定されます。これは、垂直方向に Pblock が配置されているクロック領域の高さまで、水平方向ではプログラマブル ユニット 2 つ分、左右に広がられます (プログラマブル ユニットはリソース列のペア)。

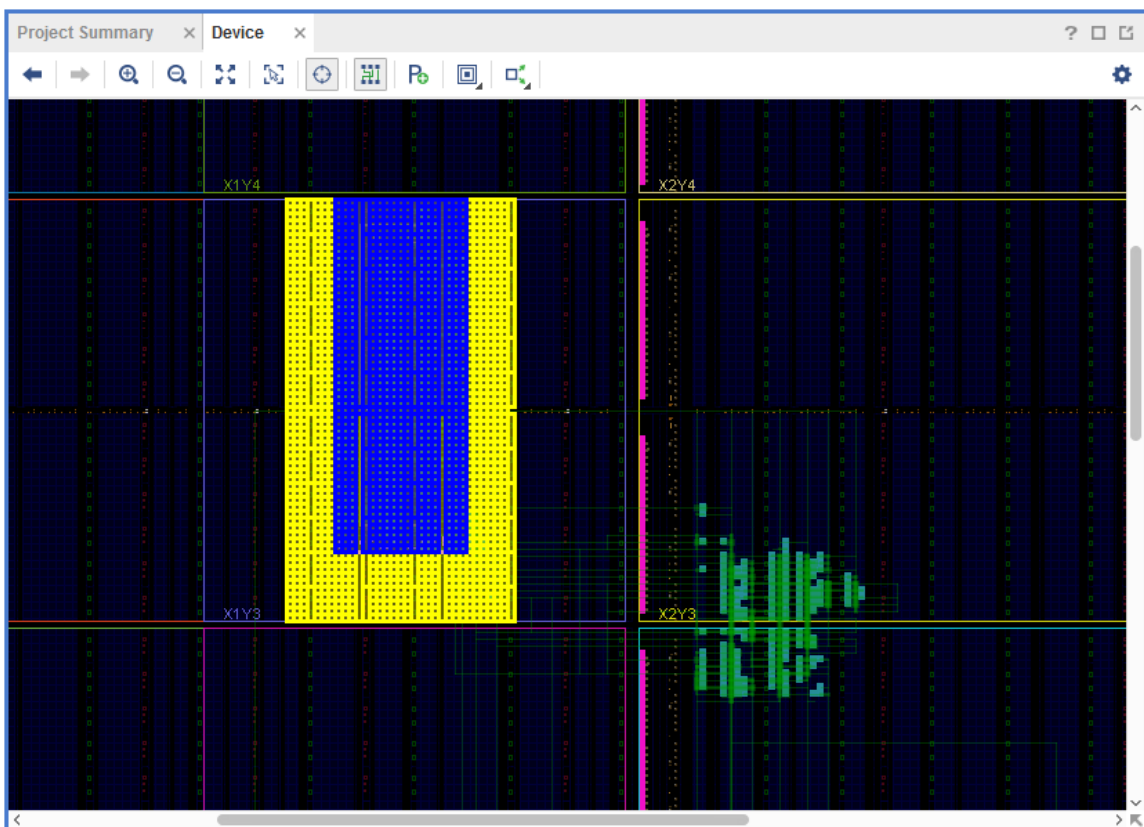
4. 次のコマンドを実行して、デザインのリコンフィギュラブル部分を配線するのに有効なフレームセットを特定します。

```
source project_1.runs/impl_1/hd_visual/  
pblock_my_math_Placement_AllTiles.tcl  
highlight_objects -color blue [get_selected_objects]
```

ハイライトされた領域は Pblock エリアそのものか、または、Pblock がプログラマブル ユニットの境界線に揃えられていない場合は、Pblock よりもやや小さなエリアになります。

デバイスのビューは次のようになるはずです。

図 63: 配置 (青) および配線 (黄) の境界を示すためハイライトされた math RP



スタティック ロジックは、拡張された配線領域 (今は黄色で残っている領域) に配置可能です。スタティックな配線には、デバイスのどのリソースでも使用できます。

- Flow Navigator で [Report Timing Summary] を選択し、[OK] をクリックしてデザインのタイミングを解析します。
- [Timing] タブで [Design Timing Summary] を選択し、[Worst Negative Slack (WNS)] の値をクリックして、ワーストパスを上位 10 個表示します。最初のパスをダブルクリックして、そのパスのタイミング サマリを開きます。

このタイミング レポートの [Clock Paths] および [Data Path] に、[Partition] という新しい列が追加されています。ここには、パスの特定部分がどのパーティション (または境界) にあるのかが示されています。



**注意:** 場合によっては、表のヘッダーを右クリックし、[Partition] をチェックして、[Partition] 列の表示を切り替える必要があります。それから、タイミング レポートのウィンドウを拡大するか、列幅を調整して、最終行の右側に [Partition] 列が表示されるようにします。

図 64: タイミング レポートでパーティションが表示された状態のパス

Delay Type	Intr (ns)	Path (ns)	Location	PBlock	Netlist Resource(s)	Partition
(clock clk_out1_clk_wiz_0 rise edge)	(t) 9.999	9.999				
	(t) 0.000	9.999	Site: AK17		clk_in1_p	static
net (to=)	0.000	9.999			clk_in1stckn1_outdnt	static
DEFINBU (Pps: DEFINBU_HPOSDEFINBU_DIFF_IN_P_O)	(t) 0.318	10.317	Site: HPOSDEFINBU_X0Y35		clk_in1stckn1_outdnt/DEFINBU_INSTIO	static
net (to=1, routed)	0.051	10.368			clk_in1stckn1_outdnt/OUT	static
BUFCCTRL (Pps: BUFCCTRL_HPOSCTRL_I_O)	(t) 0.000	10.368	Site: AK17		clk_in1stckn1_outdnt/BUFCCTRL_INSTIO	static
net (to=1, routed)	0.778	11.146			clk_in1stckn1_clk_wiz_0	static
MMCM3_ADV (Pps: MMCM3_ADV_CLKIN_CLKOUT0)	(t) -4.849	6.197	Site: MMCM3_ADV_X0Y1		clk_in1stckn1_clk_wiz_0/instCLKOUT0	static
net (to=1, routed)	0.345	6.542			clk_in1stckn1_out1_clk_wiz_0	static
BUFGCE (Pps: BUFGCE_BUFGCE_I_O)	(t) 0.075	6.617	Site: BUFGCE_X0Y38		clk_in1stckn1_out1_clk_wiz_0/instBUFGCE	static
net (to=1940, routed)	2.173	8.790	Clock region: (CLOCK_ROOT)		my_mathmy_mult_flat_TL_EG1U_CTLs_dc	boundary
FDRE			Site: SLICE_X35Y214	pblock_my_math	my_mathmy_mult_flatn_CTLsdc_reg1...	reconfigurable
clock pessimism	0.200	8.990				
clock uncertainty	-0.074	8.916				
FDRE (Setup_HFF_SLICEM_C_CE)	-0.047	8.869	Site: SLICE_X35Y214	pblock_my_math	my_mathmy_mult_flatn_CTLsdc_reg1...	reconfigurable
Required Time		8.869				

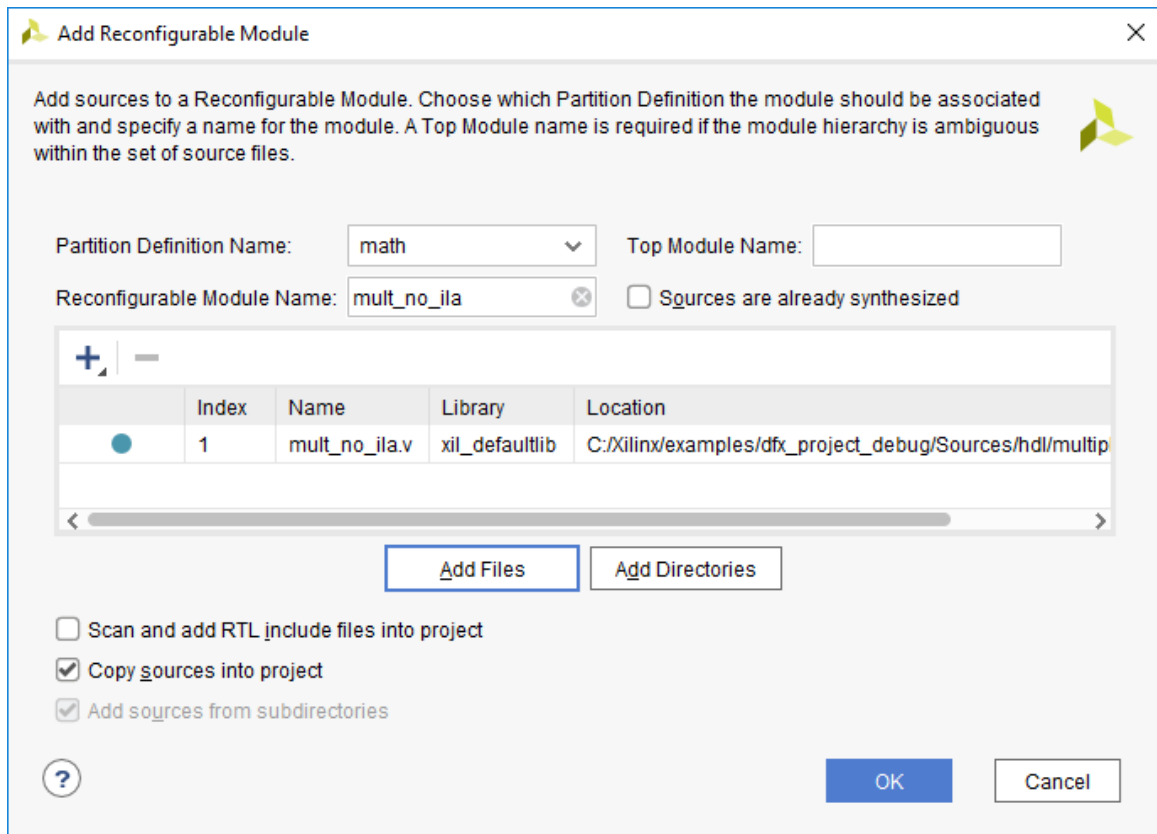
- [File] → [Close Implemented Design] をクリックして、impl\_1 のインプリメント済みデザインを閉じます。

## 手順 9: 追加リコンフィギュラブル モジュールおよびコンフィギュレーションの追加

この手順では、3 つ目の RM を追加し、そのコンフィギュレーションをインプリメントします。この RM は同じ乗算器ファンクションですが、ILA インスタンス化がコメントアウトされています。このモジュールにはデバッグ コアはありませんが、すべての RM に対して一貫性を持たせるため、デバッグに特化したポート名 (および使用されている場合はその属性) がやはり必要です。これらのポートは、グレー ボックスの場合と同じように LUT を介して接続されています。

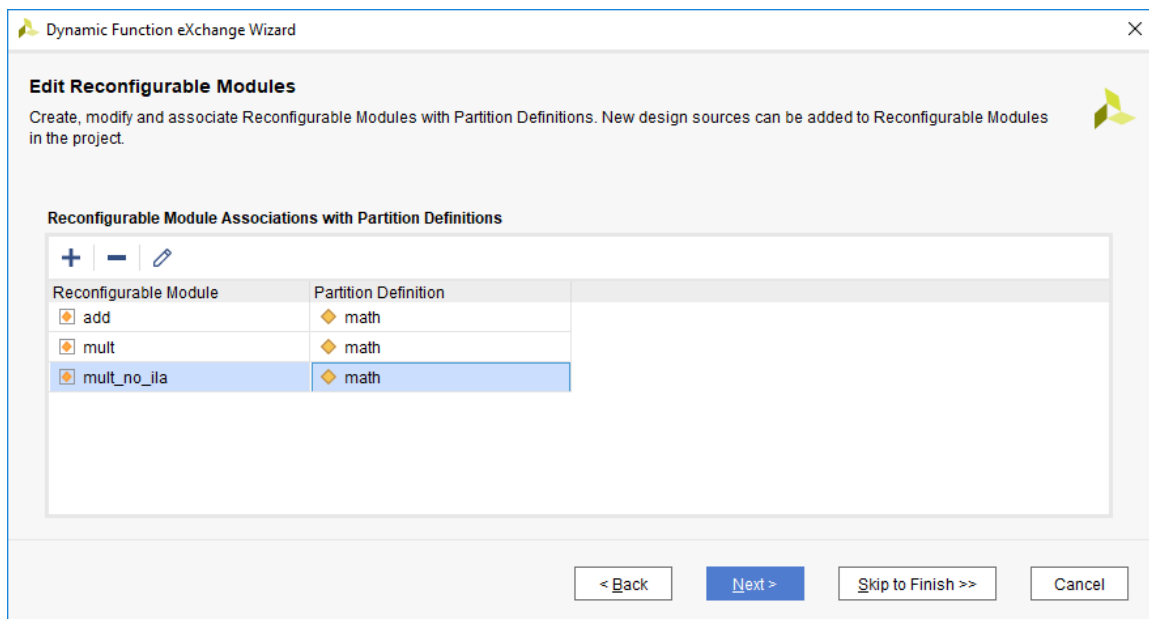
- Dynamic Function eXchange ウィザードを開きます。
- Edit Reconfigurable Modules ページで [+] ボタンをクリックし、新しい RM を追加します。

図 65: 新規 RM 「mult\_no\_ila」 の追加



3. <Extract\_Dir>\Sources\hdl\multiplier\_without\_ila にある mult\_no\_ila.v ファイルを選択し、RM を「mult\_no\_ila」と名前を付けてから [OK] をクリックし、[Next] をクリックします。

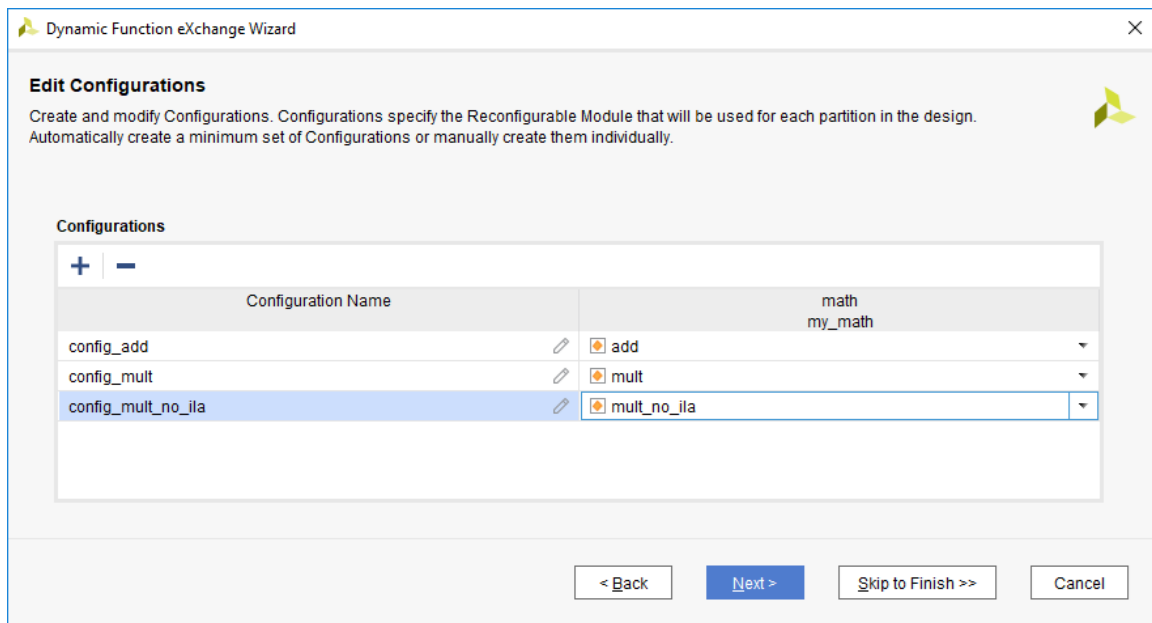
図 66: mult\_no\_ila コンフィギュレーションの作成



[Edit Configurations] ページにコンフィギュレーションを自動作成するオプションがなくなっています。これは、既にコンフィギュレーションが 2 つ存在しているからです。既存のコンフィギュレーションをすべて削除するとこのオプションが再び表示されますが、そうすると、すべてのコンフィギュレーションがまた作成され、既に生成されている結果ファイルがすべて削除されます。

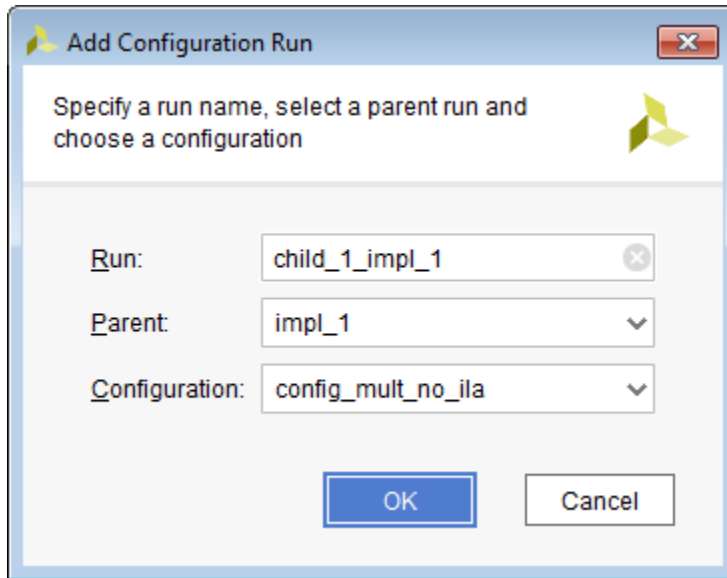
4. [+] ボタンをクリックして新しいコンフィギュレーションを作成し、「config\_mult\_no\_ila」と名前を付け、[OK] をクリックします。RM に [mult\_no\_ila] を選択します。

図 67: 新しいコンフィギュレーション run の作成



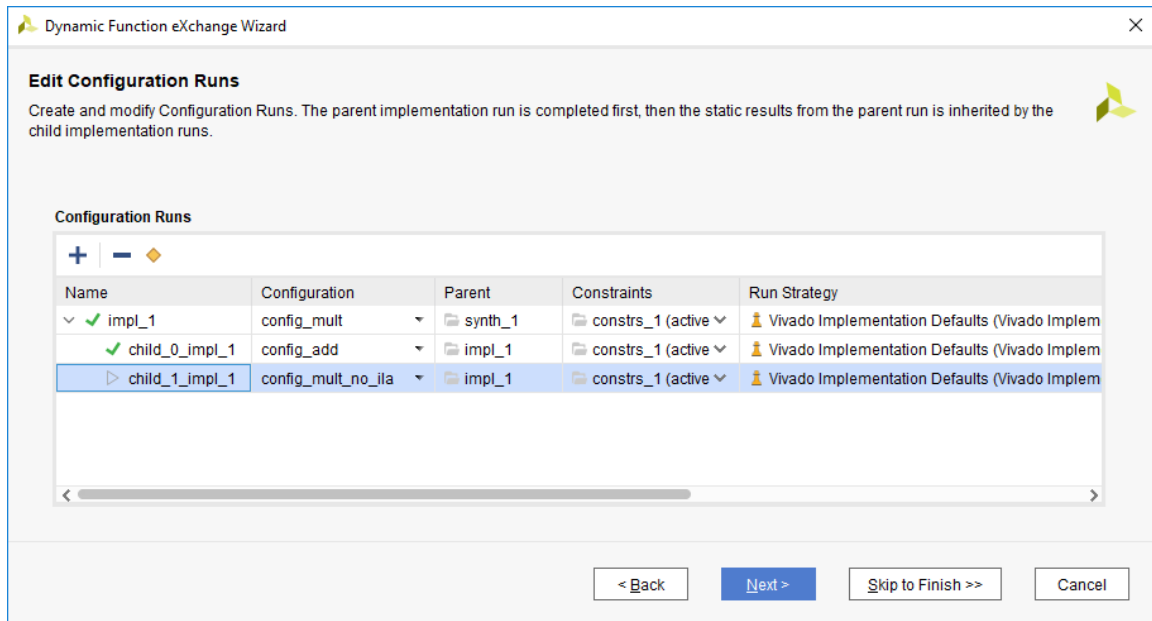
5. [Next] をクリックして [Configuration Runs] に進みます。[+] ボタンをクリックして、次のプロパティを設定し、コンフィギュレーションを新規作成します。
  - [Run]: child\_1\_impl\_1 - 既存のルールに従った名前を付けますが、どのような名前を付けてもかまいません。
  - [Parent]: impl\_1 - このコンフィギュレーションは、既存の親 run の子 run になります。
  - [Configuration]: config\_mult\_no\_ila - 先ほど定義した新規 RM のコンフィギュレーションです。
 [OK] をクリックして、この新しいコンフィギュレーションを受け入れます。

図 68: 新しいコンフィギュレーション run の作成



この新しいコンフィギュレーションは、既存の impl\_1 の子で、config\_add の場合と同じように、スタティック デザイン インプリメンテーションの結果を再利用します。この時点で 3 つの run があり、そのうち 2 つが最初の親の子です。緑色のチェック マークは、これらの run のうち 2 つが現在完了していることを示しています。

図 69: 新しく子 run として追加された config\_mult\_no\_ila コンフィギュレーション



- [Next] をクリックし、さらに [Finish] をクリックして、この新規コンフィギュレーション run を構築します。

図 70: 新しく追加された OOC 合成 run およびコンフィギュレーション run

Tcl Console	Messages	Log	Reports	Design Runs	×	Configurations
<div> <div>🔍</div> <div>⌵</div> <div>⌶</div> <div>⏪</div> <div>⏩</div> <div>⏴</div> <div>⏵</div> <div>+</div> <div>%</div> </div>						
Name	Configuration	Constraints	Status	WNS	TNS	
✓ synth_1 (active)		constrs_1	synth_design Complete!			
✓ impl_1 (active)	config_mult	constrs_1	route_design Complete!	6.052	0.000	
✓ child_0_impl_1	config_add	constrs_1	route_design Complete!	5.606	0.000	
▶ child_1_impl_1	config_mult_no_ila	constrs_1	Not started			
Out-of-Context Module Runs						
✓ mult_synth_1		mult	synth_design Complete!			
✓ add_synth_1		add	synth_design Complete!			
▶ mult_no_ila_synth_1		mult_no_ila	Not started			

- この新しく追加した子インプリメンテーション run を選択して右クリックし、[Launch Runs] をクリックします。これで、mult\_no\_ila モジュールに対し OOC 合成が実行され、それからロックされたスタティック デザインのコンテキスト内でこのモジュールがインプリメントされます。



**注意:** Flow Navigator で [Run Implementation] を選択しないでください。完了済みのインプリメンテーション run であっても、すべて再実行されます。

- インプリメンテーションが完了した後に開くダイアログ ボックスで [Cancel] をクリックします。  
child\_1\_impl\_1 を右クリックし、[Open Run] をクリックします。デバイスのビューでは次の 2 点に注意してください。
  - スタティック ロジックはロックされているので、オレンジ色で表示されます。
  - [Design Runs] タブで、RP Pblock のロジック数がほかのコンフィギュレーションと比較してかなり小さい点に注目してください。
- [Tools] → [Schematic] をクリックして (または F4 キーを押して)、回路図ビューを開きます。下位にある math\_rp インスタンスに移動し、すべての BSCAN ポートが LUT に接続され、ILA または Dbg\_Hub コアが挿入されていないことを確認します。

## 手順 10: ビットストリームの生成

この時点で、あと 2 つの手順が残っています。1 つ目のステップは、デザイン イメージのスタティック部分が一貫していることを確認するため、2 つのコンフィギュレーションを比較して、PR 検証を実行します。このステップの実行は強く推奨され、Vivado プロジェクト内で自動的に実行されます。2 つ目のステップはビットストリーム自体の生成です。

- Flow Navigator で [Generate Bitstream] をクリックします。これで、アクティブになっている 親 run に対しビットストリーム生成が起動し、インプリメント済みの子 run に対し、PR 検証、それからビットストリーム生成が起動します。
- 各コンフィギュレーション run のフルおよびパーシャル ビットストリームが生成されます。
- ビットストリーム生成が完了したら、[Open Hardware Manager] を選択します。

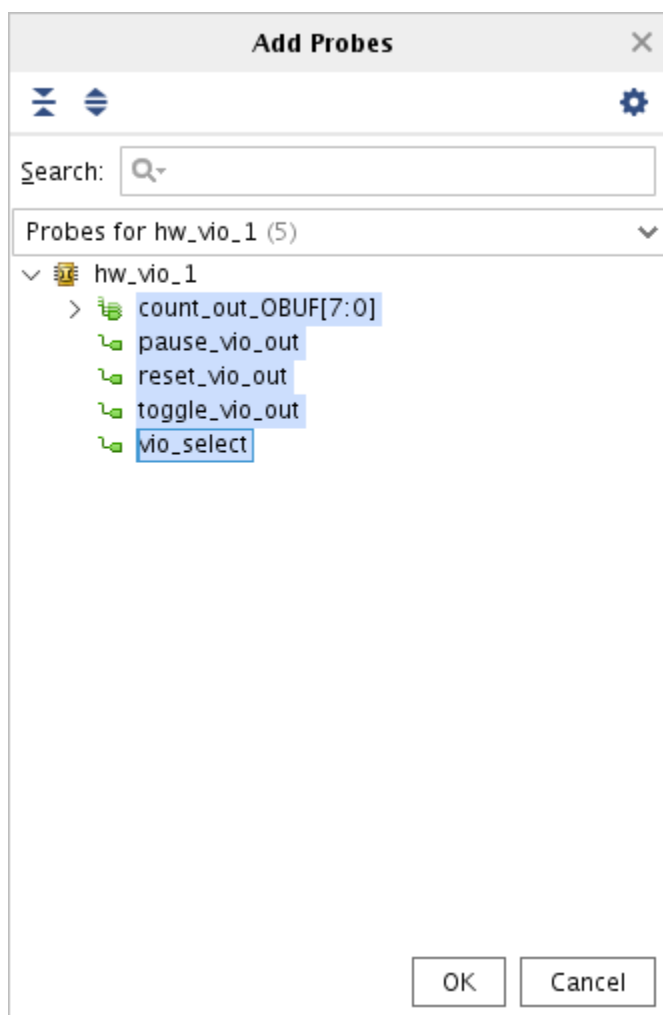
## 手順 11: ボードへの接続および FPGA のプログラム

1. ハードウェア マネージャーを開き、ターゲット ボードに接続します。

ターゲット ボードは、ローカルのもので、リモート サーバー上のものでかまいません。この接続方法の詳細は、ユーザーのセットアップによって異なります。デザインにリモートにアクセスする場合は、VIO または ILA デバッグ コアを紹介します。

2. ボードに接続できたら、次は FPGA インスタンスを右クリックし、[Program Device] をクリックします。デフォルトで、top.bit ファイルが project\_1.runs/impl\_1 ディレクトリから選択されているはずです。選択されていない場合は、impl\_1 プロジェクト run ディレクトリから top.bit を選択します。top.ltx プローブ ファイルは自動的に選択されます。これは、乗算器 RM が含まれている完全なデバイス ビットストリームです。
3. [hw\_vio\_1] ダッシュボード タブをクリックします。これが表示されていない場合は、[Dashboard Options] を開いて [hw\_vio\_1] チェック ボックスをオンにします。
4. [+] ボタンをクリックし、[Add Probes] ダイアログ ボックスからすべてのプローブを選択し、[OK] をクリックします。

図 71: デバッグ用プローブの選択



5. プローブを右クリックし、次のように設定します。
  - count\_out\_OBUF[7:0] のバス - 基数 (符号なし 10 進数)
  - count\_out\_OBUF[7:0] の個々のビット - LED (Low の場合は赤、High の場合は緑)
  - pause\_vio\_out - アクティブ High のボタン
  - reset\_vio\_out - アクティブ High のボタン
  - toggle\_vio\_out - アクティブ High のボタン
  - vio\_select - トグル ボタン

設定後のダッシュボードは次のようになります。

図 72: デバッグ用の初期 VIO ダッシュボード

hw\_ila\_1 x hw\_vios x

hw\_vios\_1

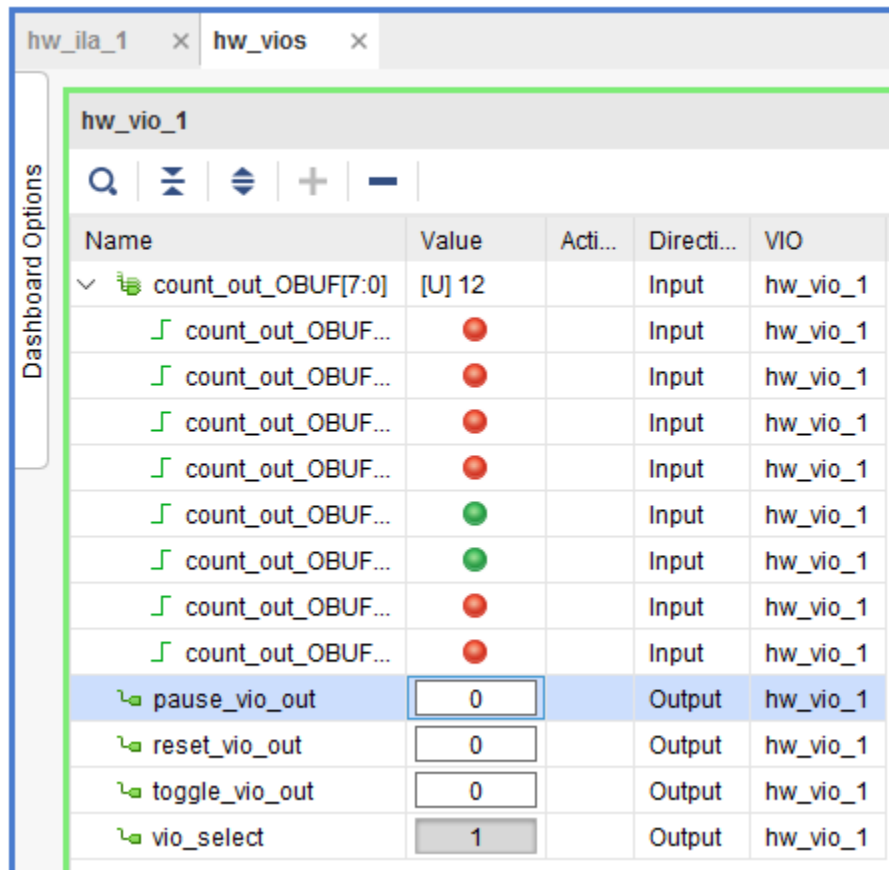
Search, Expand, Collapse, Add, Subtract

Name	Value	Acti...	Directi...	VIO
✓  count_out_OBUF[7:0]	[U] 2		Input	hw_vios_1
count_out_OBUF...			Input	hw_vios_1
count_out_OBUF...			Input	hw_vios_1
count_out_OBUF...			Input	hw_vios_1
count_out_OBUF...			Input	hw_vios_1
count_out_OBUF...			Input	hw_vios_1
count_out_OBUF...			Input	hw_vios_1
count_out_OBUF...			Input	hw_vios_1
count_out_OBUF...			Input	hw_vios_1
pause_vio_out	<input type="text" value="0"/>		Output	hw_vios_1
reset_vio_out	<input type="text" value="0"/>		Output	hw_vios_1
toggle_vio_out	<input type="text" value="0"/>		Output	hw_vios_1
vio_select	<input type="text" value="0"/>		Output	hw_vios_1

6. `vio_select` の値を 1 に変更します。これで物理的なボード上のボタンがディスエーブルになるので、VIO を介し、一時停止、リセット、およびトグル ボタンをイネーブルにします。
7. `[Pause_vio_out]` の `[Value]` をクリックして、`[pause]` ボタンを選択します。特定の値で LED のカウンタが停止します。`[count_out_OBUF]` の値は符号なしの 2 進数である点に注意してください。このスクリーンショットでは値は 12 です。



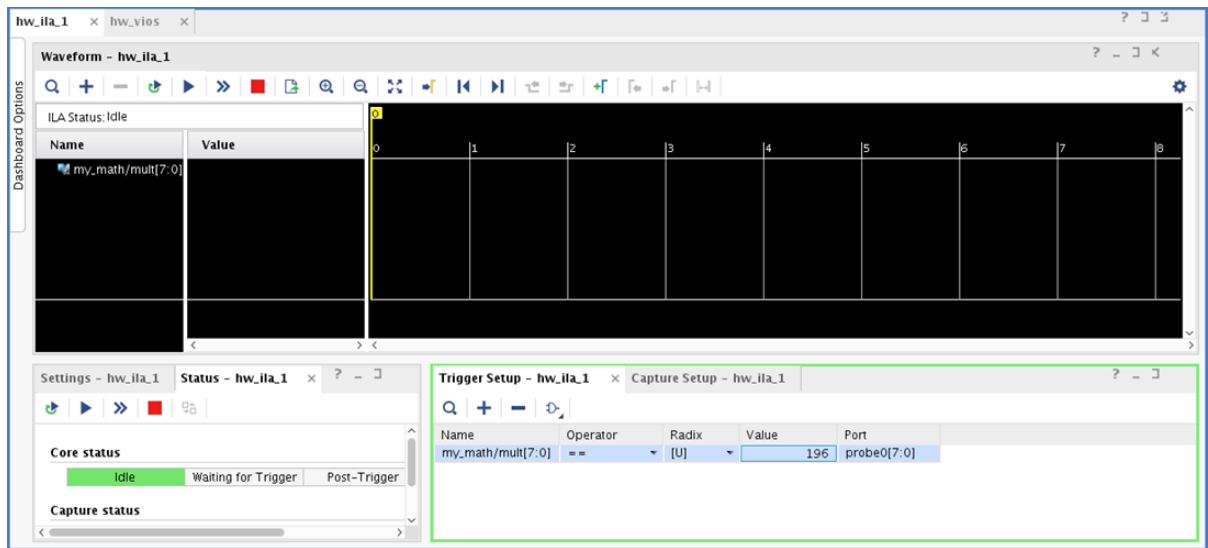
図 73: VIO を介したカウンターの監視



Name	Value	Acti...	Directi...	VIO
count_out_OBUF[7:0]	[U] 12		Input	hw_vio_1
count_out_OBUF...			Input	hw_vio_1
count_out_OBUF...			Input	hw_vio_1
count_out_OBUF...			Input	hw_vio_1
count_out_OBUF...			Input	hw_vio_1
count_out_OBUF...			Input	hw_vio_1
count_out_OBUF...			Input	hw_vio_1
count_out_OBUF...			Input	hw_vio_1
count_out_OBUF...			Input	hw_vio_1
pause_vio_out	0		Output	hw_vio_1
reset_vio_out	0		Output	hw_vio_1
toggle_vio_out	0		Output	hw_vio_1
vio_select	1		Output	hw_vio_1

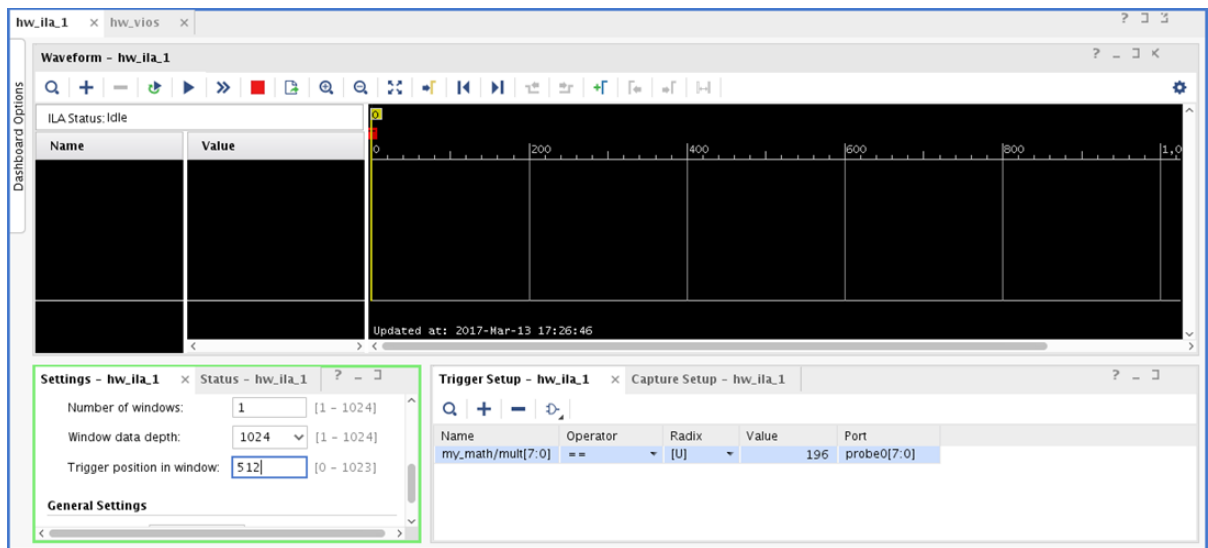
8. [toggle\_vio\_out] ボタンをクリックします。現在の RM は乗算器なので、count\_out\_OBUF バスの値は二乗になっています。この場合は 144 です。
9. [pause] ボタンをクリックするとカウンターが開始します。これで、count\_out\_OBUF の値は、0 から 15 までの値を二乗した値でカウントされるようになります (1、4、9、16、25 とカウントされる)。  
**注記:** ハードウェア マネージャーの内部クロックの相対的な周波数やサンプリング レートを考慮すると、すべての値を順番に確認できるわけではありません。
10. リセット ボタンを押し、デフォルト ステートにデザインを戻します。カウントが最初の 0 から 15 までの範囲で再開します。
11. デザインを理解するため、これらのボタンを何度か押してみてください。ローカル ボードがある場合は、vio\_select をトグルし、ボード上のボタンや LED を使用して、同じ動作になるかどうかを確認します。
12. ILA ダッシュボードに切り替えます。ここまでは、スタティック デザインにある VIO を使用してきました。乗算器の結果は確認できますが、この RM 内の波形を確認する必要がある場合は、その中にある ILA を使用して確認できます。
13. [Trigger Setup] ウィンドウで [+] ボタンをクリックし、my\_math/mult[7:0] プロローブを追加します。([Trigger Setup] および波形ウィンドウの両方で) 基数を [unsigned decimal] に変更し、その値を 196 (14x14 など) に設定します。

図 74: mult RM 内のプローブの定義



14. ILA の設定ウィンドウでトリガー位置を 512 に変更します。

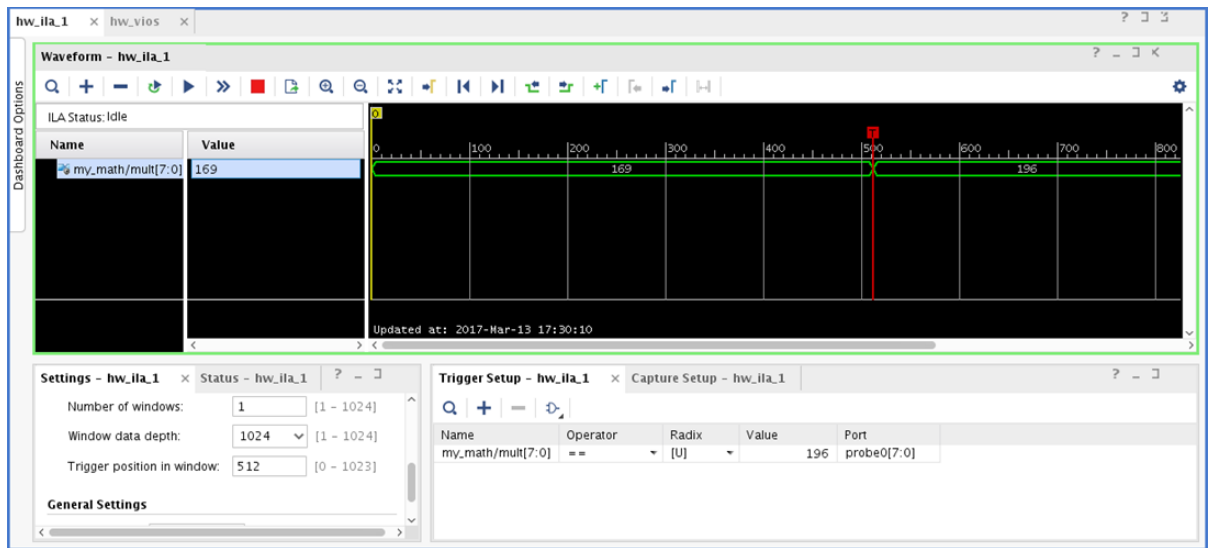
図 75: トリガー位置の設定



15. 波形のツールバーにある [run trigger] ボタンをクリックします。波形が 169 から 196 へと (132 から 142) 遷移するのが確認されます。

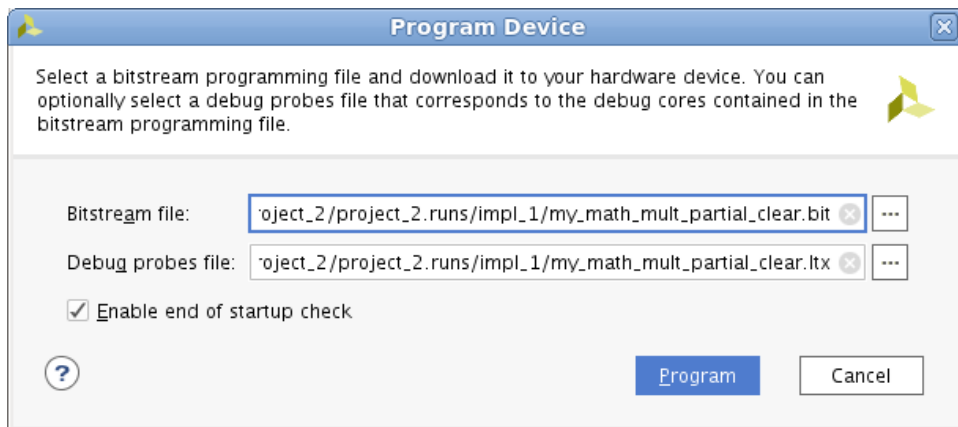
**注記:** VIO がデザインを停止させていないことを確認してください。デザインが停止していると、トリガーが発生しません。

図 76: 乗算ファンクションの確認



16. 加算器のパーシャル ビットストリームを読み込みます。ハードウェア ビューでターゲット パーツを右クリックし、[ Program Device] を選択します。
17. UltraScale パーツをターゲットにしている場合は、まずクリア ビットストリームをプログラムして、次のパーシャル ビットストリームのデザインを準備する必要があります。ビットストリーム ファイルには、乗算器のクリア ビットストリームを選択します。project\_1.runs/impl1/ ディレクトリに移動し、[my\_math\_mult\_partial\_clear.bit] ファイルを参照します。

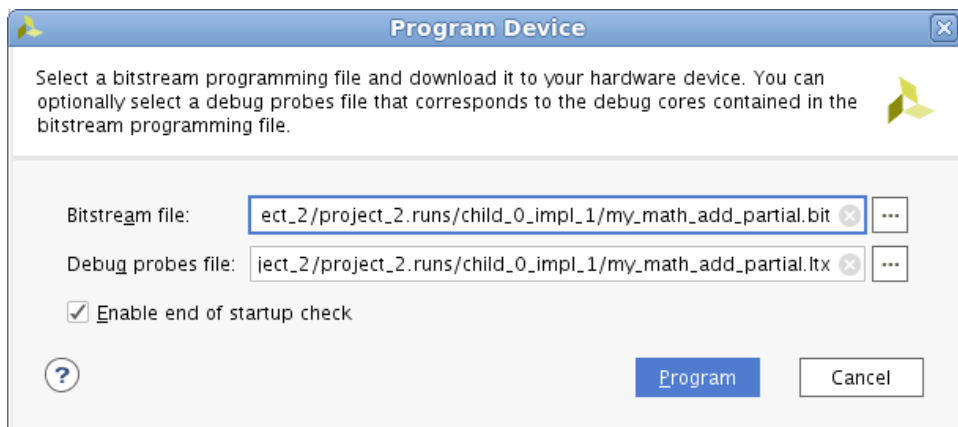
図 77: 現在の RM に対応するクリア ビットストリームの選択



LTX ファイルのペアが自動的に選択されます。[Program] をクリックします。

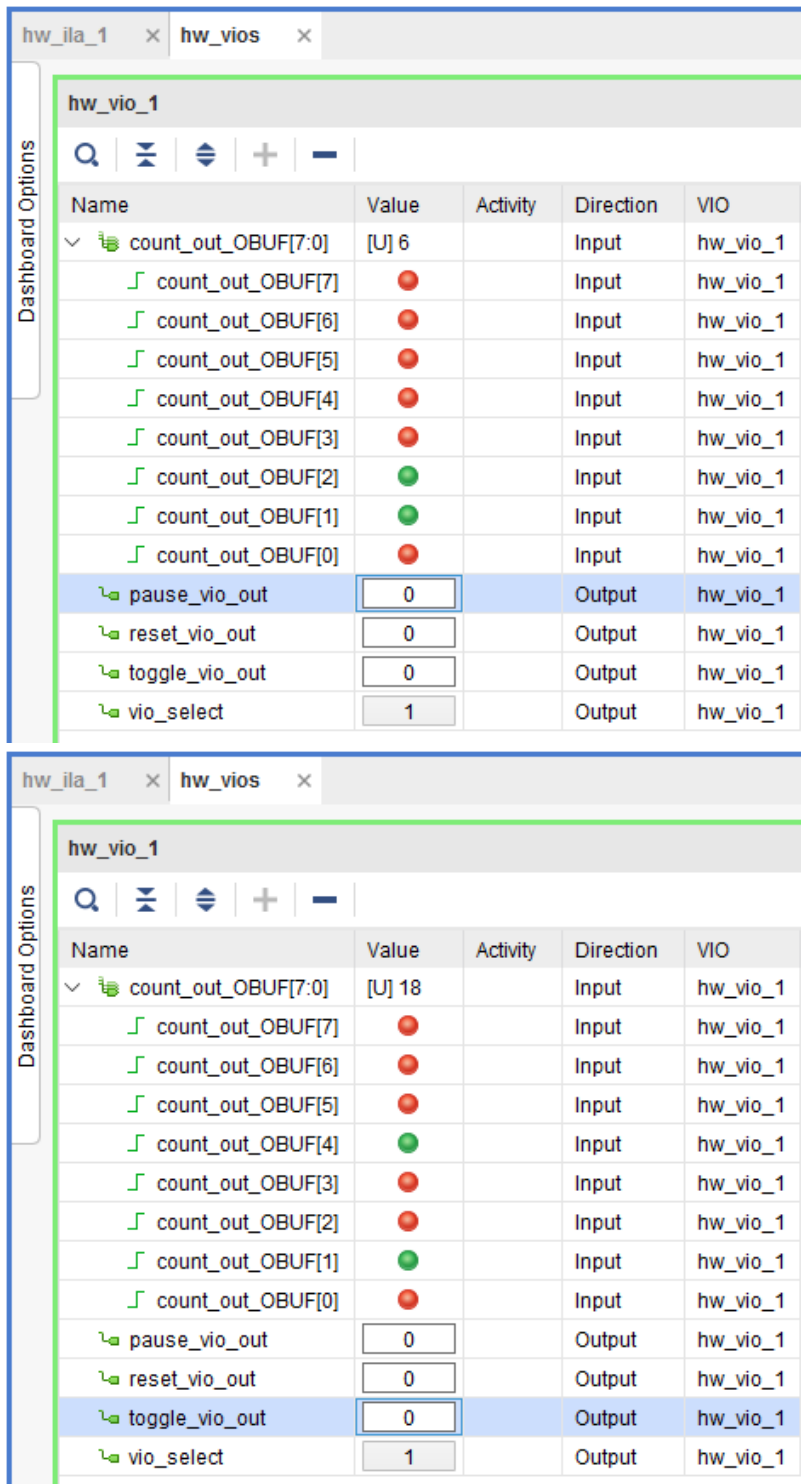
18. VIO ダッシュボードに切り替え、カウンタがまだカウント実行中であることを確認します。乗算器の出力に切り替えるのにトグル ボタンをクリックする場合は、値が 255 に保持されます。これは、RP のロジックが現在ディスエーブルになっているためです。トグル ボタンをクリックして、カウンタに戻ります。リモート制御するには、vio\_select を 1 に設定する必要があることを思い出してください。
19. [Hardware] ビューでターゲット パーツを右クリックし、[Program Device] を選択します。
20. ビットストリーム ファイルの場合は、project\_1.runs/child\_0\_impl\_1/ ディレクトリに移動し、[my\_math\_add\_partial.bit] ファイルを参照します。

図 78: 新規パーシャル ビットストリームの選択



ここでも LTX ファイルのペアが自動的に選択されます。[Program] をクリックします。

図 79: 動作中の加算ファンクション



**Top Screenshot: Initial State**

Name	Value	Activity	Direction	VIO
count_out_OBUF[7:0]	[U] 6		Input	hw_vio_1
count_out_OBUF[7]			Input	hw_vio_1
count_out_OBUF[6]			Input	hw_vio_1
count_out_OBUF[5]			Input	hw_vio_1
count_out_OBUF[4]			Input	hw_vio_1
count_out_OBUF[3]			Input	hw_vio_1
count_out_OBUF[2]			Input	hw_vio_1
count_out_OBUF[1]			Input	hw_vio_1
count_out_OBUF[0]			Input	hw_vio_1
pause_vio_out	0		Output	hw_vio_1
reset_vio_out	0		Output	hw_vio_1
toggle_vio_out	0		Output	hw_vio_1
vio_select	1		Output	hw_vio_1

**Bottom Screenshot: After Toggle**

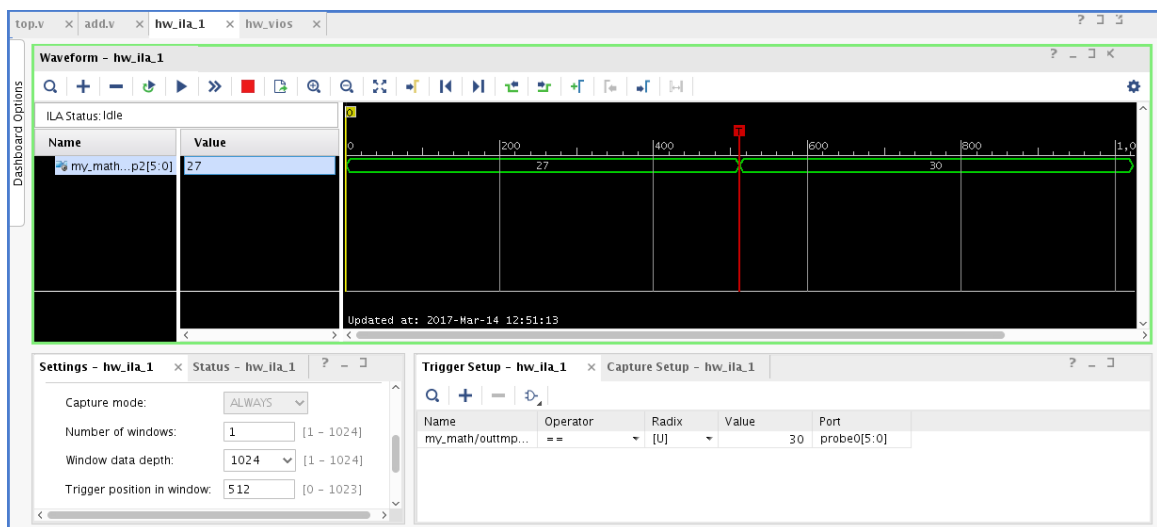
Name	Value	Activity	Direction	VIO
count_out_OBUF[7:0]	[U] 18		Input	hw_vio_1
count_out_OBUF[7]			Input	hw_vio_1
count_out_OBUF[6]			Input	hw_vio_1
count_out_OBUF[5]			Input	hw_vio_1
count_out_OBUF[4]			Input	hw_vio_1
count_out_OBUF[3]			Input	hw_vio_1
count_out_OBUF[2]			Input	hw_vio_1
count_out_OBUF[1]			Input	hw_vio_1
count_out_OBUF[0]			Input	hw_vio_1
pause_vio_out	0		Output	hw_vio_1
reset_vio_out	0		Output	hw_vio_1
toggle_vio_out	0		Output	hw_vio_1
vio_select	1		Output	hw_vio_1

21. VIO ダッシュボードで、[pause] を選択します。この場合、値は 6 で停止します。1 回トグルした後、値は 18 になります。加算器により同じ値が 3 回追加されます。

22. ILA ダッシュボードに切り替えます。[Trigger Setup] ウィンドウで [+] をクリックし、my\_math/outtemp2[5:0] バスを追加します。次のトリガー設定を変更します。
  - 基数 = 符号なし 10 進数
  - 値 = 30
23. ILA の設定ウィンドウでトリガー位置を 512 に変更します。
24. 波形ウィンドウで [+] ボタンをクリックし、my\_math/outtemp2[5:0] プローブを波形に追加します。プローブを右クリックし、基数を符号なしの 10 進数に設定します。
25. 波形ウィンドウで、ILA のトリガー ボタンをクリックします。27 (9+9+9) から 30 (10+10+10) へと遷移するのが確認されます。

**注記:** VIO がデザインを停止させていないことを確認してください。

図 80: 加算ファンクションの確認



すべてが正しく機能していることが確認できたら、ハードウェア マネージャーを閉じます。

## まとめ

RM レベルのデバッグを追加すると、Dynamic Function eXchange デザインのあらゆる部分がデバッグできるようになります。ハードウェア マネージャー内でさまざまな RM を簡単に切り替えて、フラット デザインの場合と同じようにデザイン アクティビティを確認できます。

## 演習 5

# 7 シリーズ デバイスの DFX Controller IP

## 手順 1: チュートリアル デザイン ファイルの抽出

1. ザイリンクス ウェブサイトから[リファレンス デザイン ファイル](#)をダウンロードします。
2. ZIP ファイルの内容を書き込み可能なディレクトリに抽出します。
3. \dfxc\_7s に移動します。

## 手順 2: Dynamic Function eXchange (DFX) Controller IP のカスタマイズ

DFX Controller IP をカスタマイズするには、いくつかの詳細設定が必要です。各リコンフィギュラブル パーティション (RP) およびリコンフィギュラブル モジュール (RM) に関する情報をすべて特定すると、ターゲットの FPGA のリコンフィギュレーション ニーズをすべて 把握するコントローラーが作成されます。この IP 内で、デザインのリコンフィギュラブル部分は仮想ソケットと呼ばれ、RP だけでなく、それを管理するために使用される関連スタティック ロジック (デカップリングやハンドシェイク ロジックなど) がすべて含まれています。コア パラメーターは操作中にカスタマイズ可能ですが、この手順内で入力できるパラメーターが多いほうがよいです。こうすることで、フロントエンド デザインの記述を最終的にインプリメントされたデザインとさらに正確に一致させることができます。

1. Vivado IDE を開き、[Tasks] セクションで [Manage IP] タスクをクリックします。[New IP Location] を選択して [Next] をクリックします。次の詳細を入力してから、[Finish] をクリックします。
  - [Part]: [Boards] をクリックしてターゲットのパーツを選択します。この演習では、KC705、VC707 および VC709 がサポートされています。
  - [IP Location]: <Extract\_Dir>/Sources/ip
2. IP カタログで、[Dynamic Function eXchange] のカテゴリを展開し、DFX Controller IP をダブルクリックします。

図 81: IP カタログの Dynamic Function eXchange Controller

Dynamic Function eXchange				
AXI HB ICAP	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_hbicap:1.0
DFX AXI Shutdown Manager	AXI4	Production	Included	xilinx.com:ip:dfx_axi_shutdown_manager:1.0
DFX Bitstream Monitor	AXI4	Production	Included	xilinx.com:ip:dfx_bitstream_monitor:1.0
DFX Controller	AXI4	Production	Included	xilinx.com:ip:dfx_controller:1.0
DFX Decoupler		Production	Included	xilinx.com:ip:dfx_decoupler:1.0

**注記:** Vivado 2020.1 では、パーシャル リコンフィギュレーション ユーティリティ IP をすべて新しい Dynamic Function eXchange という名前に置き換えました。機能は同じですが、IP の名前が違っているので、IP 自体は新しいものとみなされます。

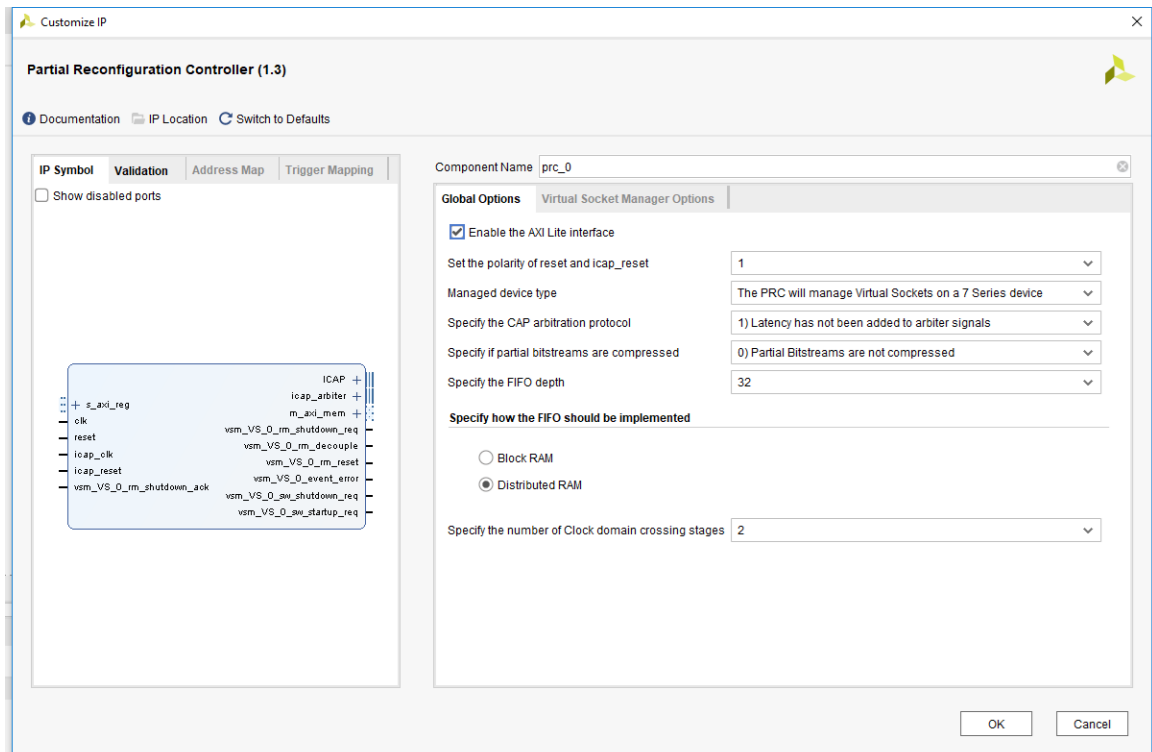
DFX Controller IP の GUI には左側に 4 つのタブがあり、IP の現在のコンフィギュレーションに対するフィードバックが確認できます。[Validation] タブにコア パラメーターを入力するときに発生する可能性のあるエラーが表示されます。エラーがあると、コアはコンパイルされません。

GUI の右側には、2 つのタブがあり、ここでカスタマイズがすべて実行されます。入力する情報のほとんどが [Virtual Socket Manager Options] タブに表示されます。

3. コンポーネント名は「dfx\_controller\_0」のままにしておきます。最終デザインで使用される DFX Controller のバージョンは自動的に [手順 3: デザインのコンパイル](#) でコンパイルされます。
4. [Global Options] タブで次の 3 点を変更します。
  - a. [Set the polarity of reset and icap\_reset] = 1
  - b. [Specify the CAP arbitration protocol] = 1. Latency has not been added to arbiter signals
  - c. [Specify the number of Clock domain crossing stages] = 2

[Managed device type] が [7 Series] に設定されていることを確認してください。DFX Controller の GUI は次のようになります。

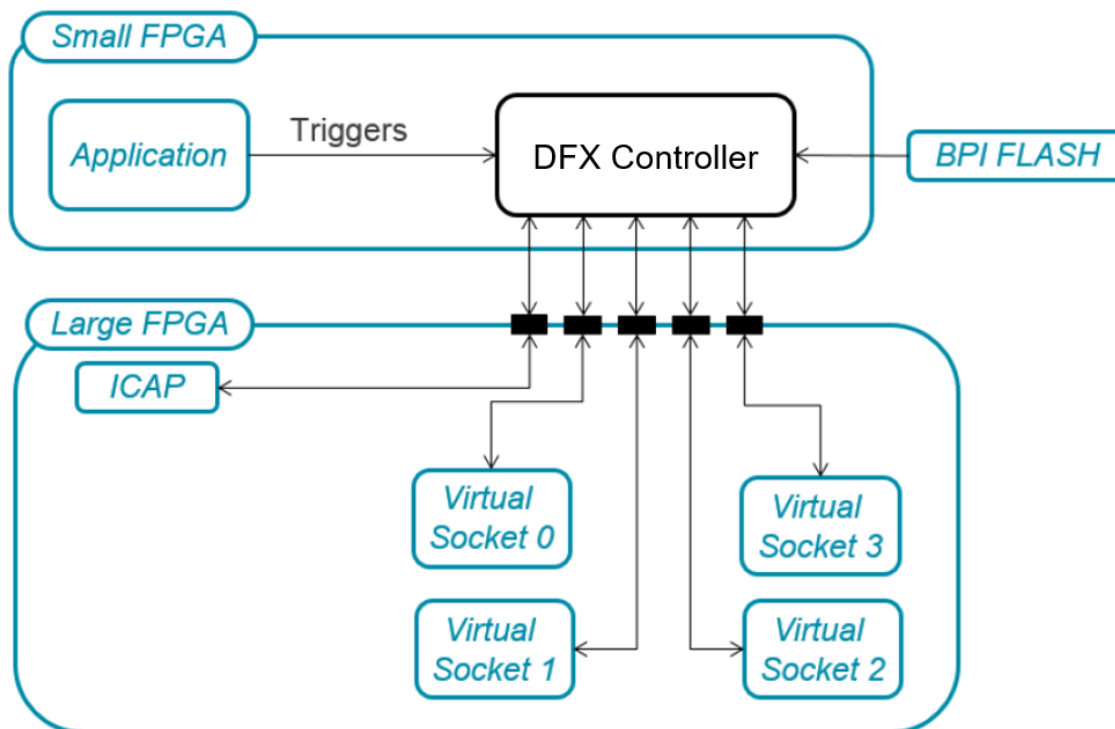
図 82: コンポーネント名および [Global Options] を設定した状態



この DFX Controller は 7 シリーズ、UltraScale、または UltraScale+ デバイスの仮想ソケットを管理できます。含まれている同じデバイスのリコンフィギュレーションを管理するだけでなく、別デバイスの ICAP に接続し、そのデバイスのリコンフィギュレーションを管理することも可能です。



図 83: DFX Controller を使用したマルチチップソリューションの例



- 次に、[Virtual Socket Manager Options] タブをクリックし、仮想ソケットおよびその RM の情報を定義します。  
DFX Controller IP には、仮想ソケット 1 つ、RM 1 つが既に読み込まれています。

まず、シフト機能の仮想ソケット マネージャー (VSM) を定義します。

- [Virtual Socket Name] フィールドで、現在の VSM の名前を「VS\_0」から「vs\_shift」に変更します。
- [Reconfigurable Module Name] フィールドで、現在の RM の名前を「RM\_0」から「rm\_shift\_left」に変更します。



#### 注意:

- VSM および RM のプルダウン メニューには選択肢が表示されていません。プルダウン メニューの下 の [Name] (ID) ラベルのほうがより正確です。
- GUI の任意フィールドに新しい値を入力し、それを受け入れるには、GUI のほかのフィールドをクリックするか、Tab キーを押します。Enter キーを押すと IP のコンパイルが始まってしまうため、Enter キーは押さないでください。

- [New Reconfigurable Module] ボタンをクリックし、この VSM の新しい RM を作成します。フォームが変更されたことに注目してください。この新しくジェネリックな RM には「RM\_1」という名前が付けられます。これを「rm\_shift\_right」に変更します。



ヒント: 1 つの RM で最大 128 個の RM を管理できます。

- vs\_shift VSM を次のように設定します。

- [Has Status Channel] = オンにする
- [Has PoR RM] = rm\_shift\_right

- [Number of RMs allocated] = 4

[Has PoR RM] は、初期フル デザインのコンフィギュレーション ファイル内に含まれている RM を示すので、FPGA のスタートアップ時に、どのトリガーやイベントが適切なのかは VSM で認識されます。この VSM により、ソケット内の現在のアクティブな RM が確認されます。

この仮想ソケットに対し定義した RM は 2 つだけですが、スペース的には 4 つ分の RM が定義できるので、後で拡張が可能です。追加 RM は AXI4-Lite インターフェイスを使用して特定できますが、RM 用にスペースが予約されている場合のみ可能です。

10. これらの RM のそれぞれに対し、次の値を入力します。[Reconfigurable Module To configure] プルダウンは、2 つの RM を切り替えるときに使用します。

- rm\_shift\_left の場合:
  - [Reset type] = [Active High]
  - [Duration of Reset] = [3]
- rm\_shift\_right の場合:
  - [Reset type] = [Active High]
  - [Duration of Reset] = [10]

**注記:** RM ごとに要件が異なる可能性があるため、要件に合わせてリセット期間をそれぞれの RM に割り当てるようにする必要があります。リセット期間はクロック サイクルで指定します。

11. 各 RM に対し、ビットストリーム サイズと、BPI フラッシュ デバイス内のビットストリームの格納場所を指定します。これらの設定は、ターゲット ボードによって異なります。

- KC705 をターゲットにしている場合:
  - rm\_shift\_left の場合:
    - [Bitstream 0 address] = 0x00AEA000
    - [Bitstream 0 size (bytes)] = 482828
  - rm\_shift\_right の場合:
    - [Bitstream 0 address] = 0x00B60000
    - [Bitstream 0 size (bytes)] = 482828
- VC707 をターゲットにしている場合:
  - rm\_shift\_left の場合:
    - [Bitstream 0 address] = 0x01355C00
    - [Bitstream 0 size (bytes)] = 708260
  - rm\_shift\_right の場合:
    - [Bitstream 0 address] = 0x01402C00
    - [Bitstream 0 size (bytes)] = 708260
- VC709 をターゲットにしている場合:
  - rm\_shift\_left の場合:
    - [Bitstream 0 address] = 0x00800000
    - [Bitstream 0 size (bytes)] = 889252

- rm\_shift\_right の場合:
  - [Bitstream 0 address] = **0x008D9400**
  - [Bitstream 0 size (bytes)] = **889252**

ビットストリームのサイズは RP の Pblock の構成に基づいており、ビットストリームのアドレスはストレージの詳細に基づいているので、この情報は、デザイン サイクルの初期段階ではわからないのが普通です。デザインをシリコン上でテストする段階に至るまでは、これらの値を 0 に設定しておくことができます。デザインが確立され、DFX Controller を使用したハードウェア テストが開始できる段階になれば、この情報を追加できます。ビットストリーム アドレスの情報は、PROM ファイル生成中に渡される情報と一致する必要があります。一部のビットストリーム生成オプション、特にビットストリームの圧縮オプションは、同じ RP であってもコンフィギュレーションが異なると、最終的なサイズが変わる可能性があります。

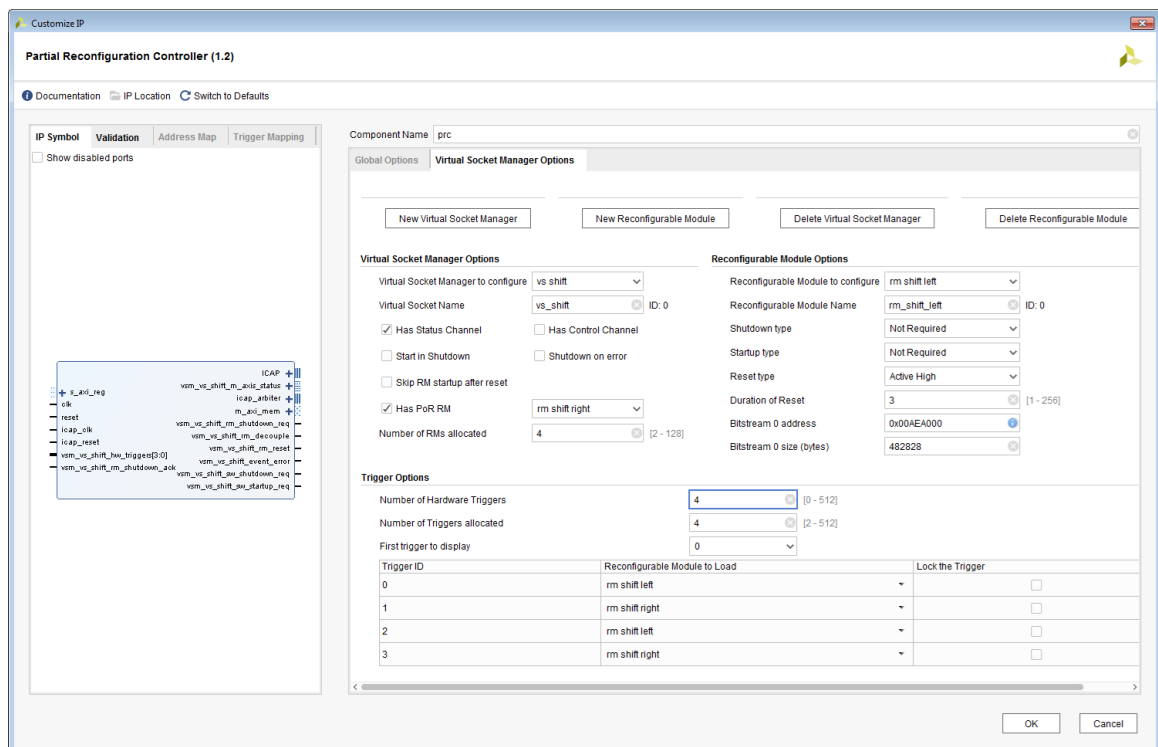
## 12. シフト機能の [Trigger Options] を定義します。

- [Number of Hardware Triggers] = 4
- [Number of Triggers allocated] = 4

この 4 つのトリガーは自動的に割り当てられます。これらは AXI4-Lite を使用してデバイス操作中に変更可能です。AXI4-Lite は、フィールド システム アップグレード中に同じ方法で新規 RM を追加したときに特に便利です。

この時点で IP の GUI は次のようになるはずです (ここでは rm\_shift\_left を表示)。

図 84: 完了した VSM の vs\_shift



次に、オプション設定は若干異なりますが、基本的には同じ手順で、カウンター用の仮想ソケットを作成し、値を設定します。

## 13. [New Virtual Socket Manager] ボタンをクリックして、新しい VSM を作成します。

14. [New Reconfigurable Module] ボタンをクリックして、次の名前およびプロパティを設定した RM を 2 つ追加します。

- [RM Name] = rm\_count\_up
  - [Reset type] = Active High
  - [Duration of Reset] = 12
- [RM Name] = rm\_count\_down
  - [Reset type] = Active High
  - [Duration of Reset] = 16

この仮想ソケットの場合は、ビットストリームのアドレスおよびサイズをデフォルトの 1 のままにしておきます。ここで定義されたものだけでなく、ビットストリームのサイズ情報は、DFX Controller の Tcl API を介して配線済みコンフィギュレーション チェックポイントに追加するか、または、AXI4-Lite インターフェイスを使用してアクティブ デザインに追加できます。カウンター用の仮想ソケットの場合は、ビットストリームのアドレスおよびサイズ情報は、配置配線後、ビットストリーム生成前に、Tcl コマンドを使用して追加されます。

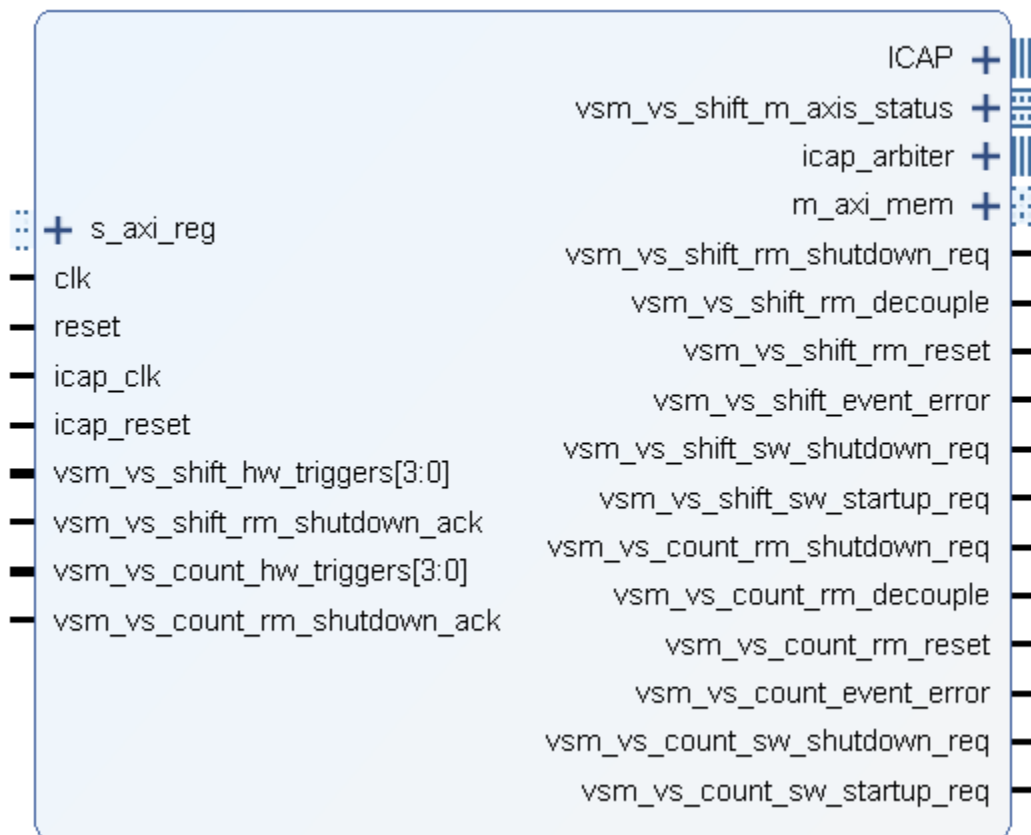
**注記:** DFX Controller の Tcl API の使用方法については、『Dynamic Function eXchange Controller 製品ガイド』(PG374) のこのセクションを参照してください。

<Extract\_Dir>/Sources/scripts ディレクトリにある Tcl スクリプトを確認します。update\_dfxc\_<board>.tcl では、dfx\_info\_<board>.tcl に格納されているビットストリームのアドレスおよびサイズをアップデートするのに DFX Controller Tcl API が使用されます。このファイルは、演習の後半で <Extract\_Dir>/design.tcl から実行します。

15. [VSM] タブで、これらの VSM 設定をデフォルト値から変更します。

- [Virtual Socket Manager name] = **vs\_count**
- [Start in Shutdown] = **オン**にする
- [Shutdown on error] = **オフ**にする
- [Has PoR RM] = **オン**にし、**rm\_count\_up** に設定

図 85: 最終的な DFX Controller シンボル



## 手順 3: デザインのコンパイル

DFX Controller IP は作成されましたが、デザインはまだコンパイルされていません。必要なフルおよびパースシャルのイメージをすべて使用して PROM イメージを作成するには、次のコマンドを使用して Tcl モードで次のスクリプトを実行する必要があります。



**重要:** この Tcl スクリプトを実行する前に、xboard 変数の値を設定するため、スクリプトを開きます。KC705 がデフォルトですが、VC707 または VC709 を選択することもできます。

- `vivado -mode tcl -source design.tcl:`

design.tcl を実行すると、必要な IP すべてが生成され (DFX Controller を含む)、デザイン全体が合成およびインプリメントされ (3 つのコンフィギュレーション)、DFX Controller の Tcl API を使用して vs\_count VSM がアップデートされ、ビットストリームが生成されます。

この IP のカスタマイズはスクリプト化されます。gen\_ip\_<board>.tcl スクリプト (<Extract\_Dir>/Sources/scripts にある) を開き、DFX Controller など、自動化された IP 作成のパラメーターをすべて確認します。IP の GUI を使用して作成する DFX Controller のインスタンスは、フル デザインの処理に実際には使用されないため、デザイン全体をコンパイルする手順 2 を完了させる必要はありません。

- `vivado -mode tcl -source create_prom_file_<board>.tcl:`

ボード別の `create_prom_file.tcl` を実行すると、ターゲット ボードの PROM イメージが作成されます。このスクリプトには、プロジェクト全体のビットストリーム アドレスのハードコード化された値が含まれています。このデザインが変更された結果、ビットストリーム サイズもフル/パーシャルに関係なく変更になる場合は、これらの値も変更する必要があります。ターゲット デバイスの変更、Pblock のサイズや形状の変更、ビットストリームの圧縮やフレームごとの CRC のオプションなどを変更すると、ビットストリーム サイズに影響します。

プロパティを設定してから `write_cfgmem` を呼び出すと、このスクリプトで PROM オプションが定義されます。DFX Controller は、データが AXI でバイトで格納されるので、バイト アドレスで機能します。このリニア フラッシュ PROM は、ハーフ ワード (16 ビット) でデータが格納されるので、ハーフ ワード アドレスが使用されます。ROM アドレスを 2 で割って AXI アドレスを計算します。たとえば、KC705 の `shift_left` アドレスは、DFX Controller をカスタマイズしたときに 00AEA000 に設定されていますが、`write_cfgmem` を呼び出す場合は 00575000 (前出のアドレスの半分) になります。バイト アドレスの境界で各ストリームが開始するようにするため、開始アドレスは常に 1024 (0x0400) の倍数になります。また、VC709 の初期コンフィギュレーション ファイルが、リニア フラッシュ メモリにフィットするように圧縮されている点に注意してください。それに続く最初のパーシャル ビットストリームのアドレスは、この初期コンフィギュレーション ファイルを拡張させるため、パッドが付いています。

この演習ディレクトリにあるファイルは `dfxc_bitstream_sizes_lab5.xlsx` という名前です。このファイルでは、ビットストリーム サイズは、黄色くハイライトされたフィールドに基づいてユーザーによって入力されます。次のバイト境界での各パーシャル ビットストリームの開始アドレスは 16 進数で計算されます。青くハイライトされている値は、DFX Controller IP のカスタマイズ用で、この IP の GUI、`gen_ip_<board>.tcl` スクリプト、配線後の API 変更用に使用される `dfx_info_<board>.tcl` を使用して入力されます。緑色にハイライトされている値は、`create_prom_file_<board>.tcl` スクリプトの PROM ファイル生成で使用され、アドレスを 2 で割った値です。

## 手順 4: ボードの設定

部分的にリコンフィギュレーション可能なデザインが完成し、機能することが確認できたら、次は、コアに接続してステータスの確認、トリガーの実行、調整ができます。

1. プログラミング用のターゲット ボードの準備をします。
  - a. Micro-USB 接続を介して、JTAG ポートをコンピューターに接続します。
    - KC705 の場合: U59
    - VC707 または VC709 の場合: U26
  - b. アドレス DIP スイッチ (SW13) を 00010 (ビット 4 が High) に設定して、コンフィギュレーション モードを 010 (BPI) に設定します。
  - c. ボードの電源をオンにします。
2. Vivado IDE を開く。
3. [Flow] → [Open Hardware Manager] をクリックします。
4. [Open Target] をクリックし、[Auto Connect] をクリックして、ターゲット デバイスを認識させます。
5. BPI コンフィギュレーション フラッシュ メモ리를 プログラムするには、デバイス (例: xc7k325t\_0) を右クリックし、[Add Configuration Memory Device] を選択します。
6. 表示されるリストの中から適切な BPI フラッシュ メモリを選択し、[OK] を 2 回クリックします。
  - KC705 の場合は、[28f00ap30t] を選択します。



- VC707 または VC709 の場合は、[28f00ag18f] を選択します。
7. [Configuration file] フィールドで、チュートリアル ディレクトリのビットストリームのサブディレクトリにある dfx\_prom.mcs を検索します。[OK] をクリックして、このファイルを選択し、さらにもう一度 [OK] をクリックしてフラッシュ メモリをプログラムします。

この時点で、チュートリアル デザインを使用してボードを操作する準備が整いました。電源を切ってもう一度電源を入れたり、ハード リセットを実行すると、このサンプル デザインを使用してザイリンクス FPGA が自動的にプログラムされます。

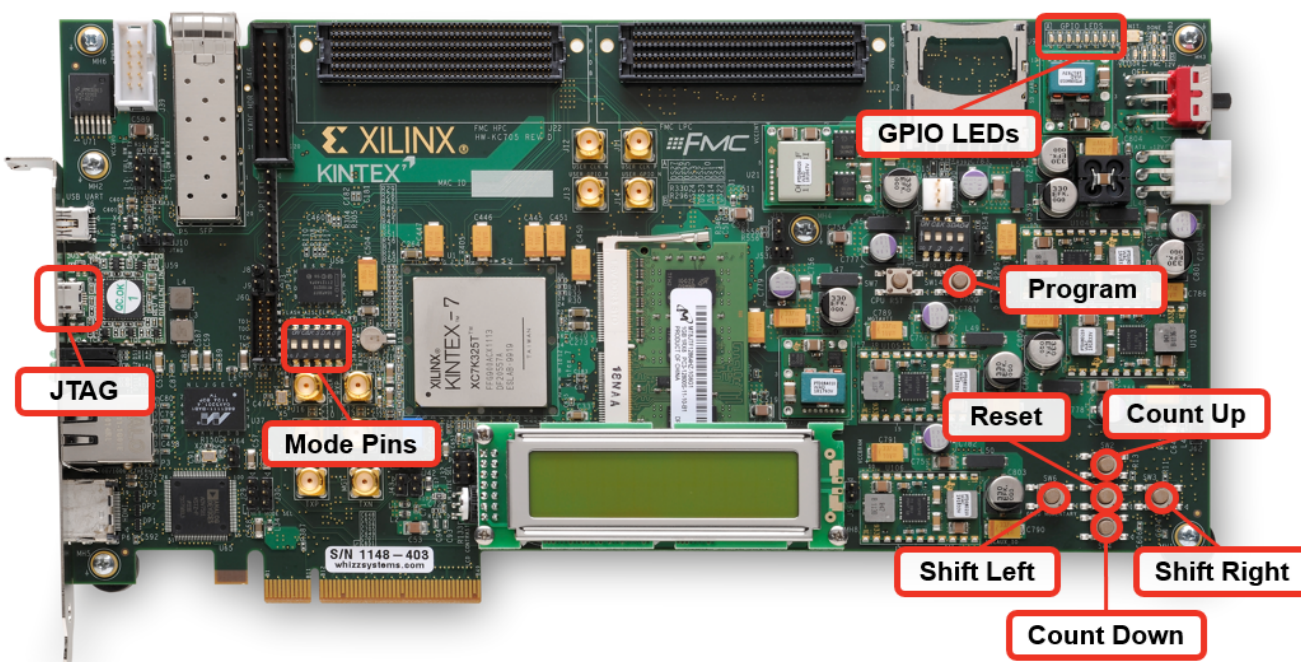
## 手順 5: サンプル デザインの操作

ボード上の文字が読みやすいように、ボードを置きます。LCD 画面が手前にあり、右側に電源、左側に JTAG があります。ボード右手前に 5 つのプッシュボタンがあり、中央右には PROG プッシュボタンがあります。

KC105 を使用している場合、これらのボタンの機能は次のようになります。

- PROG (SW14) - BPI フラッシュ メモリからデバイスをプログラム
- 上 (SW2) - カウントアップのパーシャル ビット ファイルを読み込む
- 下 (SW4) - カウントダウンのパーシャル ビット ファイルを読み込む
- 右 (SW3) - 右シフトのパーシャル ビット ファイルを読み込む
- 左 (SW6) - 左シフトのパーシャル ビット ファイルを読み込む
- 中央 (SW5) - デザインをリセット

図 86: KC705 デモ ボード上のプッシュボタン、スイッチ、接続



VC707 および VC709 を使用している場合、これらのボタンの機能は次のようになります。

- PROG (SW9) - BPI フラッシュ メモリからデバイスをプログラム
- 上 (SW3) - カウントアップのパーシャル ビット ファイルを読み込む
- 下 (SW5) - カウントダウンのパーシャル ビット ファイルを読み込む
- 右 (SW4) - 右シフトのパーシャル ビット ファイルを読み込む
- 左 (SW7) - 左シフトのパーシャル ビット ファイルを読み込む
- 中央 (SW6) - デザインをリセット

図 87: VC707 デモ ボード上のプッシュボタン、スイッチ、接続

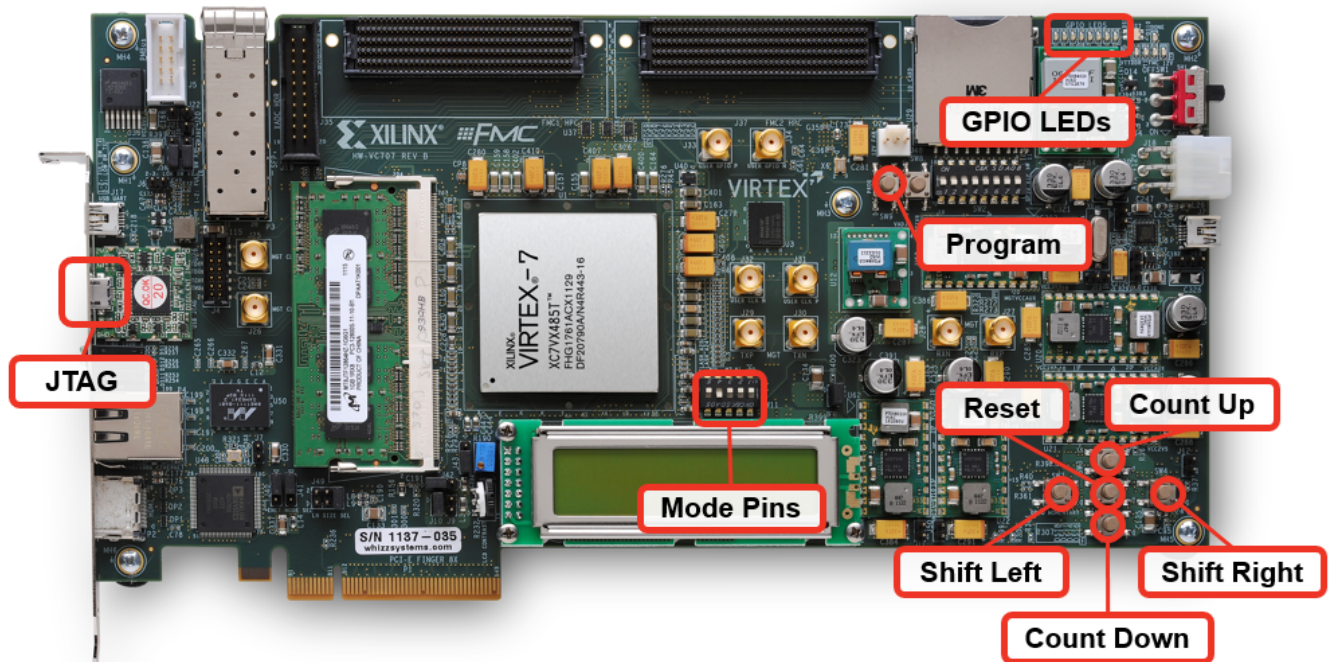
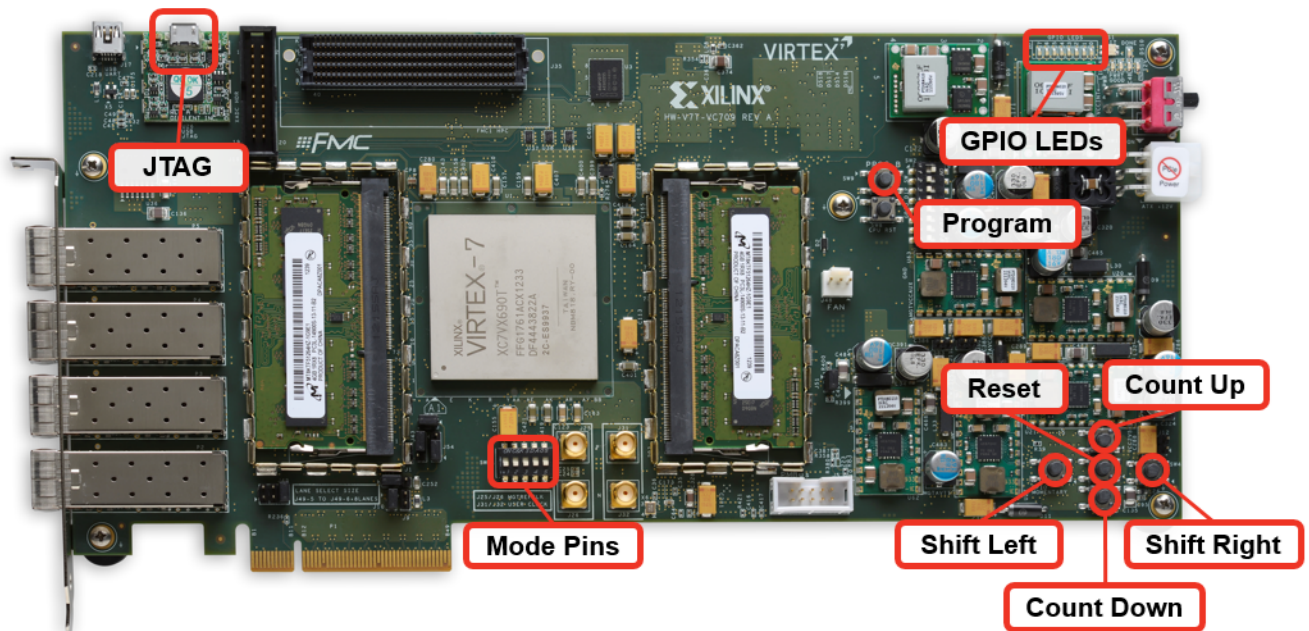




図 88: VC709 デモ ボード上のプッシュボタン、スイッチ、接続



1. [PROG] プッシュボタンを押して FPGA をプログラムします。右手奥に 8 つの GPIO LED があり、DONE LED が High になった後に操作を開始します。

この時点で、GPIO バンクの左側の 4 ビットがカウントアップし、右側の 4 ビットが右にシフトしています。

2. [Shift Left] および [Shift Right] ボタンを交互に押します。

プッシュボタンを押すたびに、DFX Controller により BPI フラッシュ メモリからパーシャル ビット ファイルが読み出され、ICAP に配信され、その RP の機能が変更されます。この状態になると、どのボタンを押すかによって、LED のシフト方向が変更します。

3. [Count Down] および [Count Up] ボタンを交互に押します。

プッシュボタンを押しても、何も変わりません。DFX Controller をコンフィギュレーションしたとき、カウンターの仮想ソケットはシャットダウン モードで開始するようプログラムされていました。これをアクティブ モードに変更するまでは、どのハードウェアまたはソフトウェア トリガーにも反応しません。

## 手順 6: FPGA での DFX Controller のクエリ

この手順では、ハードウェア マネージャーから JTAG を介してコアを使用し、コアのステータスを理解し、ソフトウェア トリガーを出力します。

Vivado のハードウェア マネージャーで、JTAG を介してデバイスとのリンクを確立するため [Refresh Device] をクリックする必要がある場合があります。XADC および 6 個の ILA コア、そして hw\_axi リンクが、[Hardware] ビューでデバイスの下に表示されています。

1. Tcl コンソールで、cd コマンドを使用して DFX Controller のチュートリアル ディレクトリに移動し、AXI4-Lite コマンドの Tcl スクリプトを実行します。

```
source ./Sources/scripts/axi_lite_procs.tcl
```

これで、今後 DFX Controller を使用するときのプロシージャ セットが簡単になります。これらのプロシージャがどのように定義されているかを確認するため、このファイルを開きます。これらはこのデザイン用にハードコードされているので、ほかのデザインの仮想ソケットへ参照する場合は変更する必要があります。この詳細は、『Dynamic Function eXchange (DFX) Controller 製品ガイド』 (PG374) を参照してください。

2. 次のプロシージャを実行して、DFX Controller との通信を確立します。

```
dfxc_jtag_setup
```

3. 各仮想ソケットのステートをチェックし、モードがシャットダウン モードかどうかを確認します。

```
is_vsm_in_shutdown vs_shift  
is_vsm_in_shutdown vs_count
```

シフトの仮想ソケットはアクティブ モード (値 = 0)、カウンターの仮想モードはシャットダウン モード (値 = 1) になっているはずです。

4. 各仮想ソケットのステータスを確認します。

```
dfxc_decode_status vs_shift  
dfxc_decode_status vs_count
```

戻されるデータを確認する前に、『Dynamic Function eXchange Controller 製品ガイド』 (PG374) のこのセクションにある表 2-4 を参照してください。このセクションの表に、STATUS レジスタの値が定義されています。これは 32 ビット レジスタなのですが、上位 8 ビットは UltraScale デバイスの仮想ソケット マネージャー (VSM) 用に使用されるので、最下位から 24 ビットのみに注目するだけで十分です。

vs\_shift のステータスは 263 で、2 進数で表すと 0000\_0000\_0000\_0001\_0000\_0111 です。vs\_shift のステータスは 7 で、唯一の違いは RM\_ID が 0 になっている点です。

- RM\_ID (ビット 23:8) = 1。つまり RM 1 が (rm\_shift\_right) 読み込まれます。また、RM\_ID (ビット 24:8) = 0 であるかのように見える場合もあります。その場合は RM 0 (rm\_shift\_left) が読み込まれます。
- SHUTDOWN (ビット 7) = 0。この VSM はシャットダウン ステートではありません。
- ERROR (ビット 6:3) = 0000。エラーはありません。
- STATE (ビット 2:0) = 111。仮想ソケットがいっぱいです。

vs\_count のステータスは 129 で、2 進数で表すと 0000\_0000\_0000\_0000\_1000\_0001 です。

- RM\_ID (ビット 23:8) = 0。その場合は RM 0 (rm\_count\_up) が読み込まれます。
- SHUTDOWN (ビット 7) = 1。この VSM はシャットダウン ステートです。
- ERROR (ビット 6:3) = 0000。エラーはありません。
- STATE (ビット 2:0) = 001。この VSM はハードウェアのシャットダウンを実行しているので、RM\_SHUTDOWN\_ACK は 1 になります。

次の詳細は、この Tcl プロシージャからの戻り値に、ステータス レジスタの内訳としてレポートされます。

5. シフトの仮想ソケットにソフトウェア トリガーを送信します。

```
dfxc_send_sw_trigger vs_shift 1  
dfxc_send_sw_trigger vs_shift 0
```

DFX Controller のカスタマイズ中に定義されたように、値が 0 および 2 のときは左シフト、値が 1 および 3 のときは右シフトです。

6. カウンターの仮想ソケットの RM のコンフィギュレーションを確認します。

```
dfxc_show_rm_configuration vs_count 1
dfxc_show_rm_configuration vs_count 0
```

ビットストリームのサイズおよびアドレスの値がここにレポートされます。これらの値は、レポートされてから、ビットストリームのサイズや位置を調整するために変更できます。新しい RM を挿入するため、異なるインデックスを追加できます。vs\_shift VSM はシャットダウン ステートではないので、vs\_shift にはこのクエリは実行できない点に注意してください。

7. カウンターの VSM をアクティブ モードにします。

```
dfxc_restart_vsm_no_status vs_count
```

これで、カウントアップおよびカウントダウンのプッシュボタンは、DFX Controller を使用してこれらのパシカル ビットストリームを読み込むことができる状態になりました。

## 手順 7: FPGA での DFX Controller の変更

この最終手順では、シフトの VSM に新しい RM を追加します。create\_prom.tcl スクリプトを開くと、ブラック ボックス モジュールが 2 つ既に生成されているのが確認できます。これらは、スタティック デザインがフィールドで運用された後に作成された可能性のある新規 RM です。DFX Controller 設定で、サイズ、アドレス、プロパティ、トリガー条件を割り当てて、この 2 つの RM の 1 つにアクセスします。

1. シフトの VSM をシャットダウンして、変更できるようにします。

```
dfxc_shutdown_vsm vs_shift
```

現在、RM ID 2 のマッピングは RM ID 0 のパシカル ビット ファイルと同じなので、同じ左シフトのパシカル ビットストリームが読み込まれます。これは、コアのカスタマイズ中に初期トリガーをマッピングしたときの設定どおりの動作です。

2. 最初の 3 つの RM ID のステータスをチェックして、それらのレジスタ バンクの割り当てを確認します。

```
dfxc_show_rm_configuration vs_shift 0
```

```
dfxc_show_rm_configuration vs_shift 1
```

```
dfxc_show_rm_configuration vs_shift 2
```

3. MCS ファイルが PROM に対して作成されると、既に BPI フラッシュ メモリに読み込まれている追加のブランキング RM が追加されます。vs\_shift のブランキング RM にポイントするよう、スロット 2 のトリガー マッピングを割り当て直すには、次のコマンド シーケンスを使用します。

```
dfxc_write_register vs_shift_rm_control2 0
```

これで、スロット 2 の RM\_CONTROL レジスタが定義されます。シャットダウン、スタートアップ、リセットは必要ありません。スロット 2 以外の 2 つのスロットに対しては、リセット期間が異なるので、制御値も異なります。

```
dfxc_write_register vs_shift_rm_bs_index2 2
```

これで、この RM ID に新しいビットストリーム リファレンスが割り当てられます。

```
dfxc_write_register vs_shift_trigger2 2
```

これで、トリガー インデックス 2 で RM 2 が回復されるよう、トリガー マッピングが割り当てられます。

```
dfxc_show_rm_configuration vs_shift 2
```

これは現在の RM ID 2 の状態を示しています。このコマンドを前回呼び出したときから変更になっている箇所に注目してください。

- ビットストリームの詳細を設定して、RM ID 2 のカスタマイズを完了させます。

KC705 の場合:

```
dfxc_write_register vs_shift_bs_size2 482828
```

```
dfxc_write_register vs_shift_bs_address2 13496320
```

VC707 の場合:

```
dfxc_write_register vs_shift_bs_size2 708260
```

```
dfxc_write_register vs_shift_bs_address2 23108608
```

VC709 の場合:

```
dfxc_write_register vs_shift_bs_size2 889252
```

```
dfxc_write_register vs_shift_bs_address2 11960320
```

- スロット 2 にはプッシュボタンが割り当てられていないので、VSM を再開させてから、ソフトウェアを使用して VSM にトリガー イベントを出力します。

```
dfxc_restart_vsm_no_status vs_shift
```

```
dfxc_send_sw_trigger vs_shift 2
```

値を 0、1、2 と切り替えて、異なるパーシャル ビットストリームを再読み込みます。スロット 2 のブランキング ビットストリームは、シフター機能を削除するので、LED 上のアクティビティは見られません。

PROM イメージには (KC705 の場合)、アドレス 13979648 にサイズが 541812 のパーシャル ビットストリームがカウンターのブラック ボックスにあります。カウンターの VSM は現在コンフィギュレーション中であるため、この同じイベントシーケンスをカウンター VSM には実行できません。DFX Controller をカスタマイズしたとき、VSM に含める RM の数は 2 つのみに設定されているので、拡張はできません。

## まとめ

これで演習 5 は終了です。この演習では、次の作業を実行しました。

- Dynamic Function eXchange (DFX) Controller IP をカスタマイズしました。
- 仮想ソケットを作成し、それに RM を追加しました。
- デザインをコンパイルし、PROM ファイルを作成しました。
- KC705、VC707 または VC709 ボード上でリニア フラッシュ メモリをプログラムしました。
- プッシュボタンを使用してハードウェア トリガーを出力しました。

- AXI4-Lite インターフェイスを使用して、コアのステータスを確認し、ソフトウェア トリガーを出力しました。
- 既に運用されているデザインに新しく RM を追加しました。

## 演習 6

# UltraScale デバイスの DFX Controller

## 手順 1: チュートリアル デザイン ファイルの抽出

1. ザイリンクス ウェブサイトから[リファレンス デザイン ファイル](#)をダウンロードします。
2. ZIP ファイルの内容を書き込み可能なディレクトリに抽出します。
3. \dfxc\_us に移動します。

## 手順 2: Dynamic Function eXchange (DFX) Controller IP のカスタマイズ

DFX Controller IP をカスタマイズするには、いくつかの詳細設定が必要です。各リコンフィギュラブル パーティション (RP) およびリコンフィギュラブル モジュール (RM) に関する情報をすべて特定すると、ターゲットの FPGA のリコンフィギュレーション ニーズをすべて把握するコントローラーが作成されます。この IP 内で、デザインのリコンフィギュラブル部分は仮想ソケットと呼ばれ、RP だけでなく、それを管理するために使用される関連スタティック ロジック (デカップリングやハンドシェイク ロジックなど) がすべて含まれています。コア パラメーターは操作中にカスタマイズ可能ですが、この手順内で入力できるパラメーターが多いほうがよいです。こうすることで、フロントエンド デザインの記述を最終的にインプリメントされたデザインとさらに正確に一致させることができます。

1. Vivado IDE を開き、[Tasks] セクションで [Manage IP] タスクをクリックし、[New IP Location,] を選択して [Next] をクリックします。次の詳細を入力してから、[Finish] をクリックします。

- [Part]: [Boards] をクリックして [VCU108] を選択します。
- [IP Location]: <Extract\_Dir>/Sources/ip

**注記:** KCU105 開発ボードのブート フラッシュが QSPI デバイスなので、このボードはサポートされていません。QSPI および同期モードの BPI コンフィギュレーションは、UltraScale デバイスの Dynamic Function eXchange ではサポートされていません。『Vivado Design Suite ユーザー ガイド: Dynamic Function eXchange』([UG909](#)) のこのセクションの表 8-1 を参照してください。

2. [IP Catalog] で、[Dynamic Function eXchange (DFX)] のカテゴリを展開し、Dynamic Function eXchange (DFX) Controller IP をダブルクリックします。

図 89: IP カタログの Dynamic Function eXchange Controller

Partial Reconfiguration				
Partial Reconfiguration Controller	Production	Included	xilinx.com:ip:prc:1.2	
Partial Reconfiguration Decoupler	Production	Included	xilinx.com:ip:pr_decoupler:1.0	

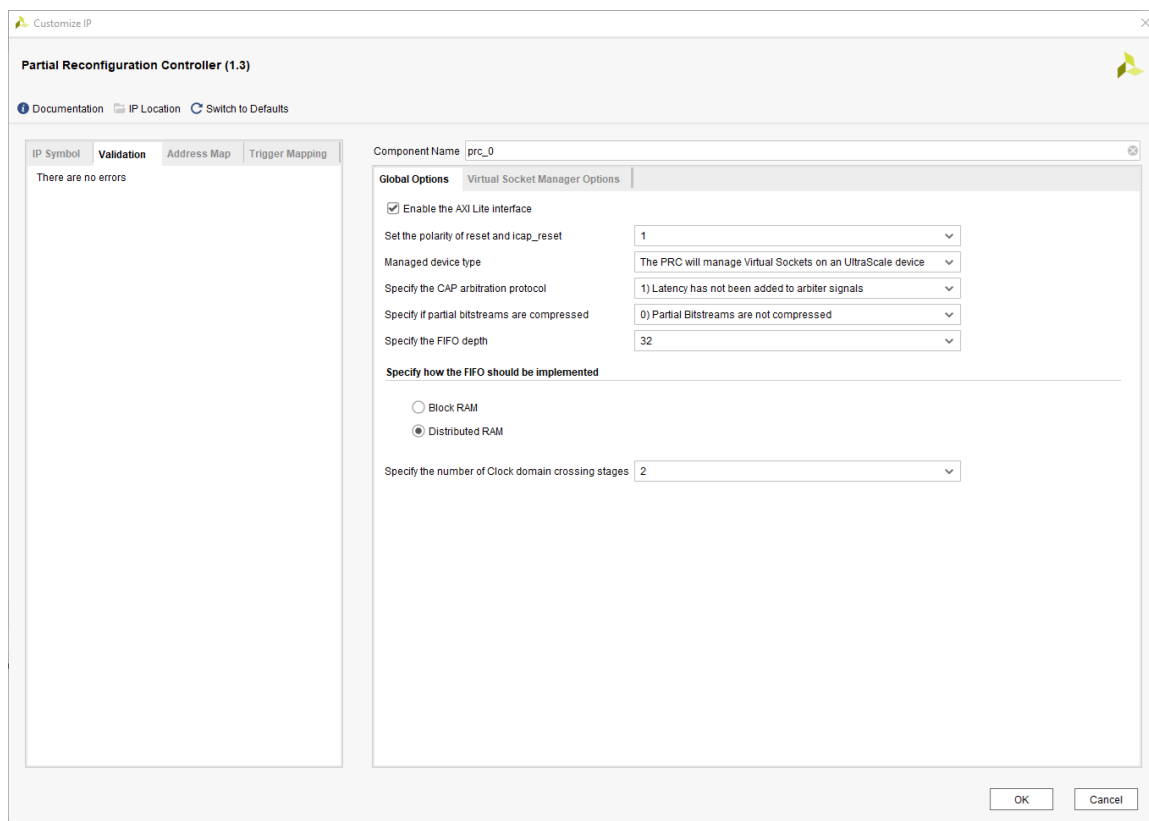
DFX Controller IP の GUI には左側に 4 つのタブがあり、IP の現在のコンフィギュレーションに対するフィードバックが確認できます。[Validation] タブにコア パラメータを入力するときに発生する可能性のあるエラーが表示されます。エラーがあると、コアはコンパイルされません。

GUI の右側には、2 つのタブがあり、ここでカスタマイズがすべて実行されます。入力する情報のほとんどが [Virtual Socket Manager Options] タブに表示されます。

3. コンポーネント名は「dfx\_controller\_0」のままにしておきます。最終デザインで使用する DFX Controller のバージョンは自動的に [手順 3: デザインのコンパイル](#) でコンパイルされます。
4. [Global Options] タブで次の 3 点を変更します。
  - a. [Set the polarity of reset and icap\_reset] = [1]
  - b. [Specify the CAP arbitration protocol] = 1) Latency has not been added to arbiter signals
  - c. [Specify the number of Clock domain crossing stages] = [2]

[Managed device type] が [UltraScale] に設定されていることを確認してください。DFX Controller の GUI は次のようになります。

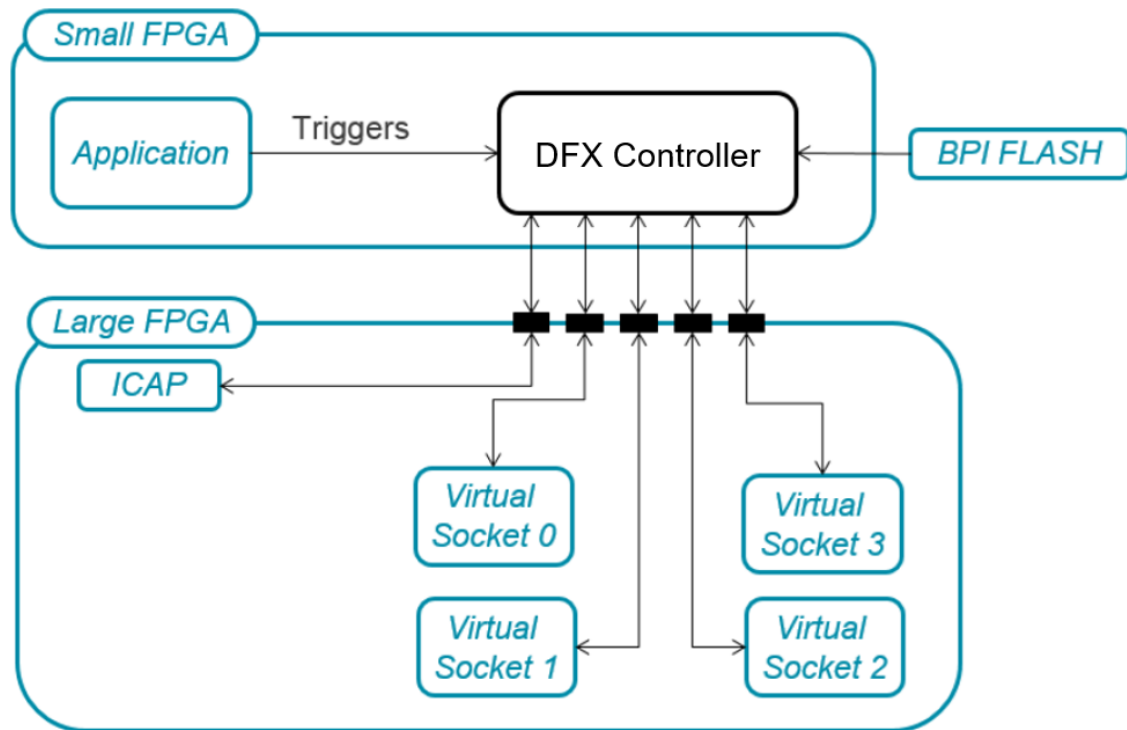
図 90: コンポーネント名および [Global Options] を設定した状態



この DFX Controller は 7 シリーズ、UltraScale、または UltraScale+ デバイスの仮想ソケットを管理できます。含まれている同じデバイスのリコンフィギュレーションを管理するだけでなく、別デバイスの ICAP に接続し、そのデバイスのリコンフィギュレーションを管理することも可能です。次は、DFX Controller を使用したマルチチップ ソリューションの例です。



図 91: DFX Controller を使用したマルチチップソリューションの例



5. 次に、[Virtual Socket Manager Options] タブをクリックし、仮想ソケットおよびその RM の情報を定義します。  
DFX Controller IP には、仮想ソケット 1 つ、RM 1 つが既に読み込まれています。  
まず、シフト機能の仮想ソケット マネージャー (VSM) を定義します。
6. 現在の VSM の名前を [VS\_0] から [vs\_shift] に変更します。
7. 現在の VSM の名前を [VS\_0] から [vs\_shift] に変更します。
8. 現在の RM の名前を [RM\_0] から [rm\_shift\_left] に変更します。


**注意:**

- VSM および RM のプルダウン メニューには選択肢が表示されていません。プルダウン メニューの下の [Name] (ID) ラベルのほうがより正確です。

GUI の任意フィールドに新しい値を入力し、それを受け入れるには、GUI のほかのフィールドをクリックするか、Tab キーを押します。Enter キーを押すと IP のコンパイルが始まってしまうため、Enter キーは押さないでください。

9. [New Reconfigurable Module] ボタンをクリックし、この VSM の新しい RM を作成します。[Reconfigurable Module Name] フィールドで [rm\_shift\_right] という名前に設定します。



**ヒント:** 1 つの RM で最大 128 個の RM を管理できます。

10. vs\_shift VSM を次のように設定します。
  - [Has Status Channel] = [checked]にする
  - [Has PoR RM] = [rm\_shift\_right]



- [Number of RMs allocated] = [ 4]

[Has PoR RM] は、初期フル デザインのコンフィギュレーション ファイル内に含まれている RM を示すので、FPGA のスタートアップ時に、どのトリガーやイベントが適切なかは VSM で認識されます。この VSM により、ソケット内の現在のアクティブな RM が確認されます。

この仮想ソケットに対し定義した RM は 2 つだけですが、スペース的には 4 つ分の RM が定義できるので、後で拡張が可能です。追加 RM は AXI4-Lite インターフェイスを使用して特定できますが、RM 用にスペースが予約されている場合のみ可能です。

11. これらの RM のそれぞれに対し、次の値を入力します。[Reconfigurable Module to configure] プルダウンは、2 つの RM を切り替えるときに使用します。
  - a. rm\_shift\_left の場合:
  - b. [Reset type] = Active High
  - c. [Duration of Reset] = 3
  - d. rm\_shift\_right の場合:
  - e. [Reset type] = [Active High]
  - f. [Duration of Reset] = [10]

[Note]: RM ごとに要件が異なる可能性があるので、要件に合わせてリセット期間をそれぞれの RM に割り当てることができるようになっています。リセット期間はクロック サイクルで指定します。

12. 各 RM に対し、ビットストリーム サイズと、BPI フラッシュ デバイス内のビットストリームの格納場所を指定します。
  - rm\_shift\_left の場合:
    - [Bitstream 0 address] = 0x00B00000
    - [Bitstream 0 size (bytes)] = 375996
    - [Bitstream 0 is a clearing bitstream] = オフにする
    - [Bitstream 1 address] = 0x00B5C000
    - [Bitstream 1 size (bytes)] = 26036
    - [Bitstream 1 is a clearing bitstream] = オンにする
  - rm\_shift\_right の場合:
    - [Bitstream 0 address] = 0x00B62800
    - [Bitstream 0 size (bytes)] = 375996
    - [Bitstream 0 is a clearing bitstream] = オフにする
    - [Bitstream 1 address] = 0x00BBE800
    - [Bitstream 1 size (bytes)] = 26036
    - [Bitstream 1 is a clearing bitstream] = オンにする

ビットストリームのサイズは RP の Pblock の構成に基づいており、ビットストリームのアドレスはストレージの詳細に基づいているので、この情報は、デザイン サイクルの初期段階ではわからないのが普通です。デザインをシリコン上でテストする段階に至るまでは、これらの値を 0 に設定しておくことができます。デザインが確立され、DFX Controller を使用したハードウェア テストが開始できる段階になれば、この情報を追加できます。ビットストリーム アドレスの情報は、PROM ファイル生成中に渡される情報と一致する必要があります。一部のビットストリーム生成オプション、特にビットストリームの圧縮オプションは、同じ RP であってもコンフィギュレーションが異なると、最終的なサイズが変わる可能性があります。

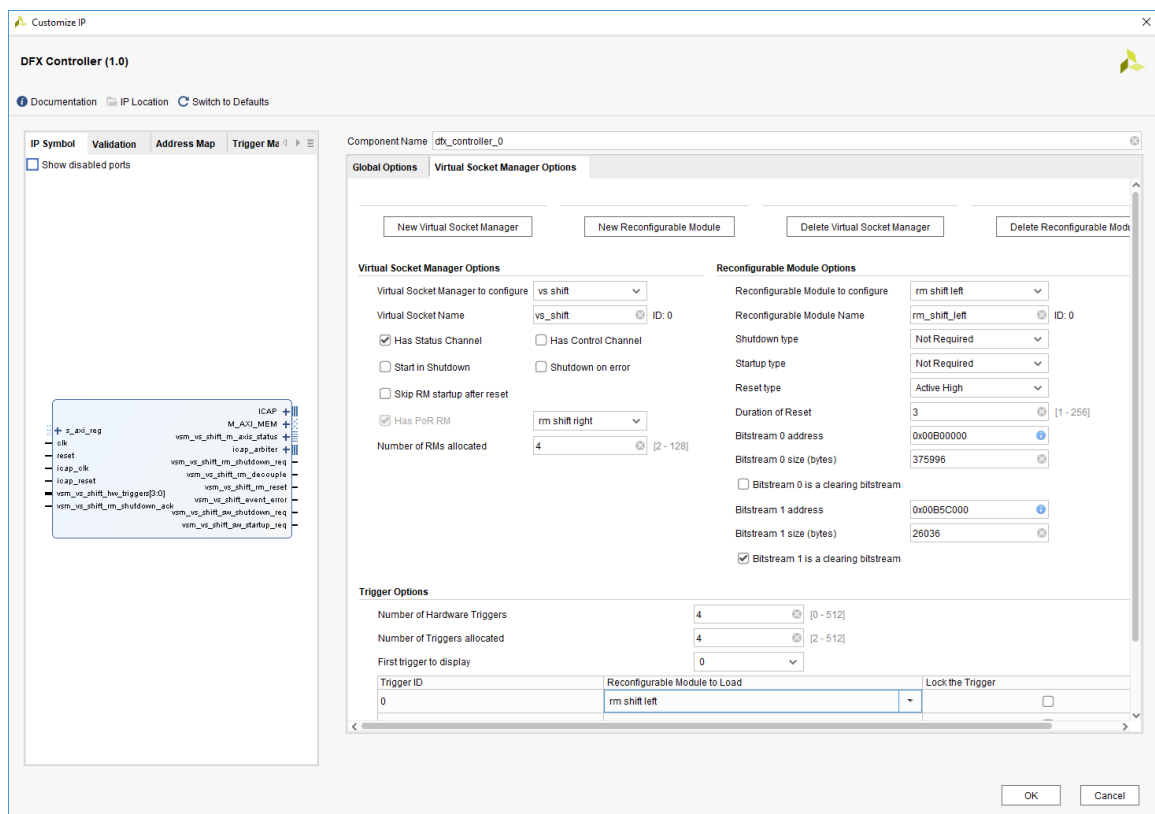
### 13. シフト機能の [Trigger Options] を定義します。

- [Number of Hardware Triggers] = [4]
- [Number of Triggers allocated] = [4]

この 4 つのトリガーは自動的に割り当てられます。これらは AXI4-Lite を使用してデバイス操作中に変更可能です。AXI4-Lite は、フィールド システム アップグレード中に同じ方法で新規 RM を追加したときに特に便利です。

この時点で IP の GUI は次のようになるはずです (ここでは rm\_shift\_left を表示)。

図 92: 完了した VSM の vs\_shift



次に、オプション設定は若干異なりますが、基本的には同じ手順で、カウンター用の仮想ソケットを作成し、値を設定します。

### 14. [New Virtual Socket Manager] ボタンをクリックして、新しい VSM を作成します。

### 15. 次の名前およびプロパティで RM を 2 つ追加します。

- [RM Name] = rm\_count\_up

- [Reset type] = Active High
- [Duration of Reset] = 12
- [RM Name] = rm\_count\_down
- [Reset type] = Active High
- [Duration of Reset] = 16

この仮想ソケットの場合は、ビットストリームのアドレスおよびサイズ情報はデフォルトの 0 のままにしておきますが、ビットストリームは 1 に設定してビットストリームを一掃します。ここで定義されたものだけでなく、ビットストリームのサイズ情報は、DFX Controller の Tcl API を介して配線済みコンフィギュレーション チェックポイントに追加するか、AXI4-Lite インターフェイスを使用してアクティブ デザインに追加できます。カウンター用の仮想ソケットの場合は、ビットストリームのアドレスおよびサイズ情報は、配置配線後、ビットストリーム生成前に、Tcl コマンドを使用して追加されます。



**ヒント:** DFX Controller の Tcl API の使用方法については、『Dynamic Function eXchange Controller 製品ガイド』 (PG374) の [このセクション](#) を参照してください。

<Extract\_Dir>/Sources/scripts ディレクトリにある Tcl スクリプトを確認します。  
update\_dfxc\_vcu108.tcl では、dfx\_info\_vcu108.tcl に格納されているビットストリームのアドレスおよびサイズをアップデートするのに DFX Controller の Tcl API が使用されます。このファイルは、演習の後半で <Extract\_Dir>/design.tcl から実行します。

16. [VSM] タブで、これらの VSM 設定をデフォルト値から変更します。

- [Virtual Socket Manager name] = [vs\_count]
- [Start in Shutdown] = [checked] にする
- [Shutdown on error] = [unchecked] にする
- [Has PoR RM] = [checked]、[rm\_count\_up] にする

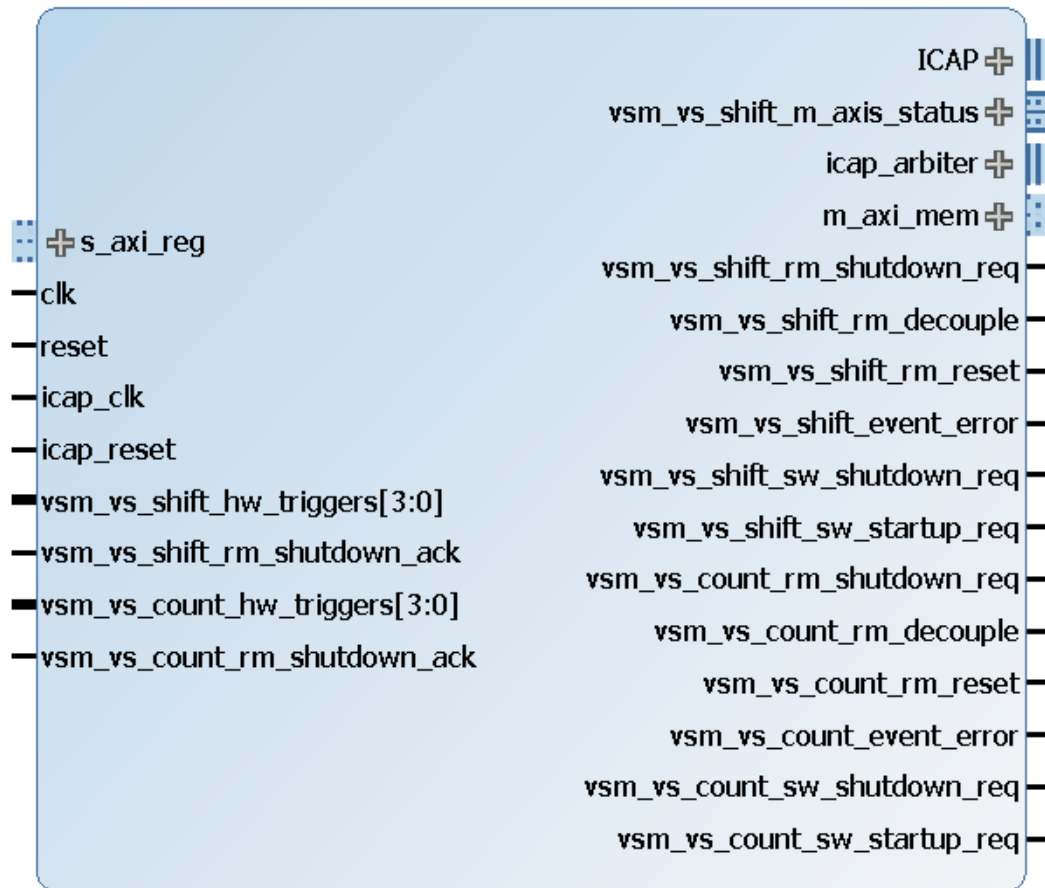
17. カウンター機能の [Trigger Options] を定義します。

- [Number of Hardware Triggers] = [4]
- [Number of Triggers allocated] = [4]

これで、このチュートリアルでの DFX Controller IP のカスタマイズは終了です。

18. [OK] をクリックしてから、[Generate] をクリックして、コアのコンパイルおよびアウト オブ コンテキスト合成を開始します。

図 93: 最終的な DFX Controller シンボル



## 手順 3: デザインのコンパイル

DFX Controller IP は作成されましたが、デザインはまだコンパイルされていません。必要なフルおよびパースシャルのイメージをすべて使用して PROM イメージを作成するには、次のコマンドを使用して Tcl モードで次のスクリプトを実行する必要があります。

- `vivado -mode tcl -source design.tcl:`

`design.tcl` を実行すると、必要な IP すべてが生成され (DFX Controller を含む)、デザイン全体が合成およびインプリメントされ (3 つのコンフィギュレーション)、DFX Controller の Tcl API を使用して `vs_count` VSM がアップデイトされ、ビットストリームが生成されます。

この IP のカスタマイズはスクリプト化されます。gen\_ip\_vcu108.tcl スクリプト (<Extract\_Dir>/Sources/scripts にある) を開き、DFX Controller など、自動化された IP 作成のパラメーターをすべて確認します。IP の GUI を使用して作成する DFX Controller のインスタンスは、フル デザインの処理に実際には使用されないため、デザイン全体をコンパイルする手順 2 を完了させる必要はありません。

- `vivado -mode tcl -source create_prom_file_vcu108.tcl:`

`create_prom_file_vcu108.tcl` を実行すると、VCU108 の PROM イメージが作成されます。このスクリプトには、プロジェクト全体のビットストリーム アドレスのハードコード化された値が含まれています。このデザインが変更された結果、ビットストリーム サイズもフル/パーシャルに関係なく変更になる場合は、これらの値も変更する必要があります。ターゲット デバイスの変更、Pblock のサイズや形状の変更、ビットストリームの圧縮やフレームごとの CRC のオプションなどを変更すると、ビットストリーム サイズに影響します。

プロパティを設定してから `write_cfgmem` を呼び出すと、このスクリプトで PROM オプションが定義されます。DFX Controller は、データが AXI でバイトで格納されるので、バイト アドレスで機能します。このリニア フラッシュ PROM は、ハーフ ワード (16 ビット) でデータが格納されるので、ハーフ ワード アドレスが使用されます。ROM アドレスを 2 で割って AXI アドレスを計算します。たとえば `shift_left` アドレスは、DFX Controller をカスタマイズしたときに 00B00000 に設定されていますが、`write_cfgmem` を呼び出す場合は 00580000 (前出のアドレスの半分) になります。バイト アドレスの境界で各ストリームが開始するようにするため、開始アドレスは常に 1024 (0x0400) の倍数になります。

この演習ディレクトリにあるファイルは `dfxc_bitstream_sizes_lab6.xlsx` という名前です。このファイルでは、ビットストリーム サイズは、黄色くハイライトされたフィールドに基づいてユーザーによって入力されます。次のバイト境界での各パーシャル ビットストリームの開始アドレスは 16 進数で計算されます。青くハイライトされている値は、DFX Controller IP のカスタマイズ用で、この IP の GUI、`gen_ip_vcu108.tcl` スクリプト、配線後の API 変更用で使用される `dfx_info_vcu108.tcl` を使用して入力されます。緑色にハイライトされている値は、`create_prom_file_vcu108.tcl` スクリプトの PROM ファイル生成で使用され、アドレスを 2 で割った値です。

## 手順 4: ボードの設定

部分的にリコンフィギュレーションが可能なデザインが完成し、機能することが確認できたら、次は、コアに接続してステータスの確認、トリガーの実行、調整ができます。

1. プログラミング用の VCU108 ボードの準備をします。
  - a. Micro-USB を介して、JTAG ポート (J106) をコンピューターに接続します。
  - b. アドレス DIP スイッチ (SW16) を 00010 (ビット 4 が High) に設定して、コンフィギュレーション モードを 010 (BPI) に設定します。
  - c. ボードの電源をオンにします。
2. Vivado IDE を開きます。
3. [Flow] → [Open Hardware Manager] をクリックします。
4. [Open Target] をクリックし、[Auto Connect] をクリックします。Virtex UltraScale VU095 デバイスが認識されます。
5. BPI コンフィギュレーション フラッシュ メモリをプログラムするには、デバイス (xcvu095\_0) を右クリックし、[Add Configuration Memory Device] を選択します。
6. 表示されるリストの中から適切な Micron フラッシュ メモリ 28f00ag18f を選択し、[OK] を 2 回クリックします。

7. [Configuration file] フィールドで、チュートリアル ディレクトリのビットストリームのサブディレクトリにある dfx\_prom.mcs を検索します。[OK] をクリックして、このファイルを選択し、さらにもう一度 [OK] をクリックしてフラッシュ メモリをプログラムします。

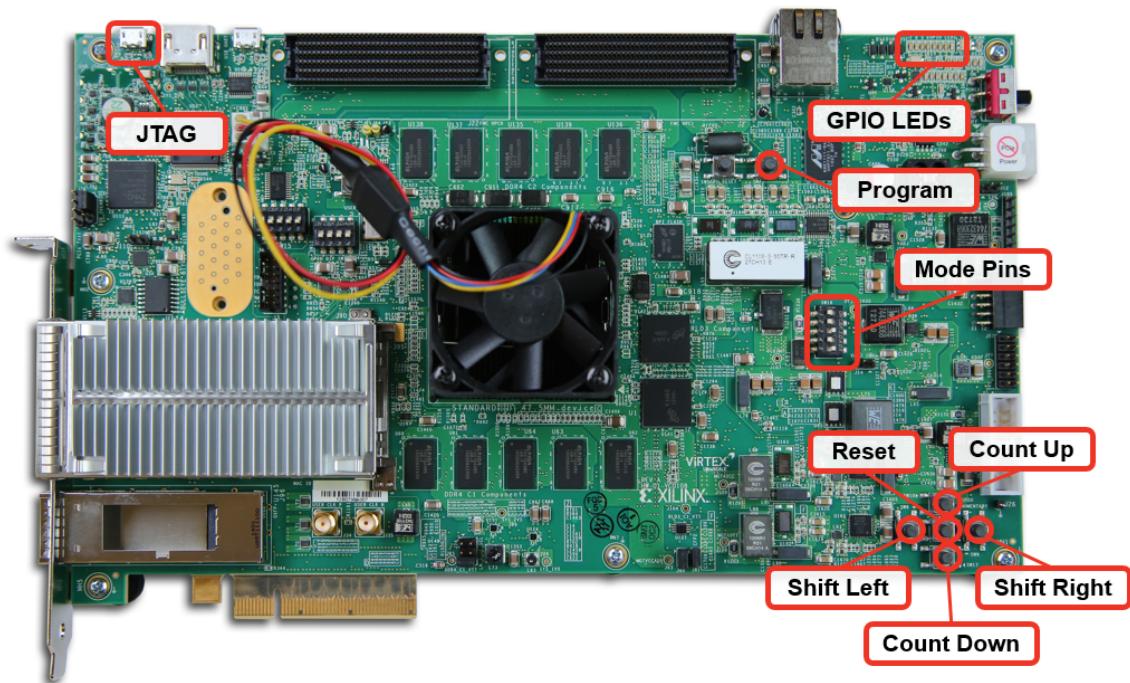
この時点で、チュートリアル デザインを使用してボードを操作する準備が整いました。電源を切ってもう一度電源を入れたり、ハード リセットを実行すると、このサンプル デザインを使用して UltraScale FPGA が自動的にプログラムされます。

## 手順 5: サンプル デザインの操作

ボード上の文字が読みやすいように、ボードを置きます。LCD 画面が手前にあり、右側に電源、左側に JTAG があります。ボード右手前に 5 つのプッシュボタンがあり、中央右には PROG プッシュボタンがあります。これらのプッシュボタンの機能は次のとおりです。

- PROG (SW4) - BPI フラッシュ メモリからデバイスをプログラム
- 上 (SW10) - カウントアップのパーシャル ビット ファイルを読み込む
- 下 (SW8) - カウントダウンのパーシャル ビット ファイルを読み込む
- 右 (SW9) - 右シフトのパーシャル ビット ファイルを読み込む
- 左 (SW6) - 左シフトのパーシャル ビット ファイルを読み込む
- 中央 (SW7) - デザインをリセット

図 94: VCU108 デモ ボード上のプッシュボタン、スイッチ、接続



1. PROG プッシュボタンを押して FPGA をプログラムします。右手奥に 8 つの GPIO LED があり、DONE LED が High になった後に操作を開始します。



この時点で、GPIO バンクの左側の 4 ビットがカウントアップし、右側の 4 ビットが右にシフトしています。

2. Shift Left および Shift Right ボタンを交互に押します。

プッシュボタンを押すたびに、DFX Controller により BPI フラッシュ メモリからパーシャル ビット ファイルが読み出され、ICAP に配信され、その RP の機能が変更されます。この状態になると、どのボタンを押すかによって、LED のシフト方向が変更します。

3. Count Down および Count Up ボタンを交互に押します。

プッシュボタンを押しても、何も変わりません。DFX Controller をコンフィギュレーションしたとき、カウンターの仮想ソケットはシャットダウン モードで開始するようプログラムされていました。これをアクティブ モードに変更するまでは、どのハードウェアまたはソフトウェア トリガーにも反応しません。

## 手順 6: FPGA での DFX Controller のクエリ

この手順では、ハードウェア マネージャーから JTAG を介してコアを使用し、コアのステータスを理解し、ソフトウェア トリガーを出力します。

Vivado のハードウェア マネージャーで、JTAG を介してデバイスとのリンクを確立するため [Refresh Device] をクリックする必要がある場合があります。XADC および 6 個の ILA コア、そして hw\_axi リンクが、[Hardware] ビューでデバイスの下に表示されています。

1. Tcl コンソールで、cd コマンドを使用して DFX Controller のチュートリアル ディレクトリに移動し、AXI4-Lite コマンドの Tcl スクリプトを実行します。

```
source ./Sources/scripts/axi_lite_procs_us.tcl
```

これで、今後 DFX Controller を使用するときのプロシージャ セットが簡単になります。これらのプロシージャがどのように定義されているかを確認するため、このファイルを開きます。これらはこのデザイン用にハードコードされているので、ほかのデザインの仮想ソケットへ参照する場合は変更する必要があります。この詳細は、『Dynamic Function eXchange (DFX) Controller 製品ガイド』 (PG374) を参照してください。

2. 次のプロシージャを実行して、DFX Controller との通信を確立します。

```
dfxc_jtag_setup
```

3. 各仮想ソケットのステータスをチェックし、モードがシャットダウン モードかどうかを確認します。

```
is_vsm_in_shutdown vs_shift
is_vsm_in_shutdown vs_count
```

シフトの仮想ソケットはアクティブ モード (値 = 0)、カウンターの仮想モードはシャットダウン モード (値 = 1) になっているはずです。

4. 各仮想ソケットのステータスを確認します。

```
dfxc_decode_status vs_shift
dfxc_decode_status vs_count
```

戻されるデータを確認する前に、『Dynamic Function eXchange Controller 製品ガイド』 (PG374) のこのセクションにある表 2-4 を参照してください。このセクションの表に、STATUS レジスタの値が定義されています。これは 32 ビット レジスタなのですが、上位 8 ビットは UltraScale デバイスの仮想ソケット マネージャー (VSM) 用に使用されるので、最下位から 24 ビットのみに注目するだけで十分です。

vs\_shift のステータスは 263 で、2 進数で表すと 0000\_0000\_0000\_0001\_0000\_0111 です。vs\_shift のステータスは 7 で、唯一の違いは RM\_ID が 0 になっている点です。

- RM\_ID (ビット 23:8) = 1。つまり RM 1 が (rm\_shift\_right) 読み込まれます。また、RM\_ID (ビット 24:8) = 0 であるかのように見える場合もあります。その場合は RM 0 (rm\_shift\_left) が読み込まれます。
- SHUTDOWN (ビット 7) = 0。この VSM はシャットダウン ステートではありません。
- ERROR (ビット 6:3) = 0000。エラーはありません。
- STATE (ビット 2:0) = 111。仮想ソケットがいっぱいです。

vs\_count のステータスは 129 で、2 進数で表すと 0000\_0000\_0000\_0000\_1000\_0001 です。

- RM\_ID (ビット 23:8) = 0。その場合は RM 0 (rm\_count\_up) が読み込まれます。
- SHUTDOWN (ビット 7) = 1。この VSM はシャットダウン ステートです。
- ERROR (ビット 6:3) = 0000。エラーはありません。
- STATE (ビット 2:0) = 001。この VSM はハードウェアのシャットダウンを実行しているので、RM\_SHUTDOWN\_ACK は 1 になります。

次の詳細は、この Tcl プロシージャからの戻り値に、ステータス レジスタの内訳としてレポートされます。

- シフトの仮想ソケットにソフトウェア トリガーを送信します。

```
dfxc_send_sw_trigger vs_shift 0
dfxc_send_sw_trigger vs_shift 1
```

DFX Controller のカスタマイズ中に定義されたように、値が 0 および 2 のときは左シフト、値が 1 および 3 のときは右シフトです。

- カウンターの仮想ソケットの RM のコンフィギュレーションを確認します。

```
dfxc_show_rm_configuration vs_count 0
dfxc_show_rm_configuration vs_count 1
```

クリア ビットストリームおよびパーシャル ビットストリームの両方のビットストリームのサイズおよびアドレスの値がここにレポートされます。これらの値は、レポートされてから、ビットストリームのサイズや位置を調整するために変更できます。新しい RM を挿入するため、異なるインデックスを追加できます。vs\_shift VSM はシャットダウン ステートではないので、vs\_shift にはこのクエリは実行できない点に注意してください。

- カウンターの VSM をアクティブ モードにします。

```
dfxc_restart_vsm_no_status vs_count
```

これで、カウントアップおよびカウントダウンのプッシュボタンは、DFX Controller を使用してこれらのパーシャル ビットストリームを読み込むことができる状態になりました。

## 手順 7: FPGA での DFX Controller の変更

この最終手順では、シフトの VSM に新しい RM を追加します。create\_prom.tcl スクリプトを開くと、ブラックボックス モジュールが 2 つ既に生成されているのが確認できます。これらは、スタティック デザインがフィールドで運用された後に作成された可能性のある新規 RM です。DFX Controller 設定で、サイズ、アドレス、プロパティ、トリガー条件を割り当てて、この 2 つの RM の 1 つにアクセスします。

- シフトの VSM をシャットダウンして、変更できるようにします。

```
dfxc_shutdown_vsm vs_shift
```



- 最初の 3 つの RM ID のステータスをチェックして、それらのレジスタ バンクの割り当てを確認します。

```
dfxc_show_rm_configuration vs_shift 0
dfxc_show_rm_configuration vs_shift 1
dfxc_show_rm_configuration vs_shift 2
```

現在、RM ID 2 は、どのパーシャル ビットストリームにも割り当てられていません。これは、コアのカスタマイズ中に初期トリガーをマッピングしたときの設定どおりの動作です。

- MCS ファイルが PROM に対して作成されると、既に BPI フラッシュ メモリに読み込まれている追加のブランキング RM が追加されます。vs\_shift のブランキング RM にポイントするよう、スロット 2 のトリガー マッピングを割り当て直すには、次のコマンド シーケンスを使用します。

```
dfxc_write_register vs_shift_rm_control2 0
```

これで、スロット 2 の RM\_CONTROL レジスタが定義されます。シャットダウン、スタートアップ、リセットは必要ありません。スロット 2 以外の 2 つのスロットに対しては、リセット期間が異なるので、制御値も異なります。

```
dfxc_write_register vs_shift_rm_bs_index2 327684
```

これで、この RM ID に新しいビットストリーム リファレンスが割り当てられます。

```
dfxc_write_register vs_shift_trigger2 2
```

これで、トリガー インデックス 2 で RM 2 が回復されるよう、トリガー マッピングが割り当てられます。DFX Controller 内の RM\_BS\_INDEX レジスタは 32 ビットですが、2 つのフィールドに分かれています。UltraScale デバイスには、クリア ビットストリームおよびパーシャル ビットストリームが必要です。それぞれのビットストリームに ID があり、その ID で識別されますが、このフィールドでは一緒に参照されます。

327684 という値は 00000000000000101\_0000000000000100 の 2 進数に変換されます。または単純に、ID5 は CLEAR\_BS\_INDEX の上位 16 ビット、ID 4 は BS\_INDEX の下位 16 ビットになります。この振り分けによって、クリアビット ストリームおよびパーシャル ビットストリームの ID が同時に設定されます。

```
dfxc_show_rm_configuration vs_shift 2
```

これは現在の RM ID 2 のステータスを示しています。このコマンドを前回呼び出したときから変更になっている箇所に注目してください。

- ビットストリームの詳細を設定して、RM ID 2 のカスタマイズを完了させます。

```
dfxc_write_register vs_shift_bs_size4 375996
dfxc_write_register vs_shift_bs_address4 12935168
dfxc_write_register vs_shift_bs_size5 26036
dfxc_write_register vs_shift_bs_address5 13312000
```

- スロット 2 にはプッシュボタンが割り当てられていないので、VSM を再開させてから、ソフトウェアを使用して VSM にトリガー イベントを出力します。

```
dfxc_restart_vsm_no_status vs_shift
dfxc_send_sw_trigger vs_shift 2
```

値を 0、1、2 と切り替えて、異なるパーシャル ビットストリームを再読み込みます。スロット 2 のブランキング ビットストリームは、シフター機能を削除するので、LED 上のアクティビティは見られません。

PROM イメージには、アドレス 13338624 にサイズが 274104 のパーシャル ビットストリームがカウンターのブラック ボックスにありますが、カウンターの VSM は現在コンフィギュレーション中であるため、この同じイベント シーケンスをカウンター VSM には実行できません。DFX Controller をカスタマイズしたとき、VSM に含める RM の数は 2 つのみに設定されているので、拡張はできません。

---

## まとめ

これで演習 6 は終了です。この演習では、次の作業を実行しました。

- Dynamic Function eXchange (DFX) Controller IP をカスタマイズしました。
- 仮想ソケットを作成し、それに RM を追加しました。
- デザインをコンパイルし、PROM ファイルを作成しました。
- VCU108 ボード上でリニア フラッシュ メモリをプログラムしました。
- プッシュボタンを使用してハードウェア トリガーを出力しました。
- AXI4-Lite インターフェイスを使用して、コアのステータスを確認し、ソフトウェア トリガーを出力しました。
- 既に運用されているデザインに新しく RM を追加しました。

## 演習 7

# UltraScale+ デバイスの DFX Controller IP

## 手順 1: チュートリアル デザイン ファイルの抽出

1. ザイリンクス ウェブサイトから[リファレンス デザイン ファイル](#)をダウンロードします。
2. ZIP ファイルの内容を書き込み可能なディレクトリに抽出します。
3. \dfxc\_usp に移動します。

## 手順 2: チュートリアル デザイン ファイルの処理

ここでは、デザイン処理ではなく、最終的なランタイム機能およびソフトウェア管理が目的でデザインを使用します。インプリメンテーション フローの詳細はここではあまり説明しません。Dynamic Function eXchange デザイン フローについては、この資料の前半にある演習を参照してください。

1. チュートリアル デザインのアーカイブ ファイルを解凍します。
2. コマンド シェルで Vivado を起動し、サンプル デザインのプロジェクト作成スクリプトを実行します。これは、該当するボードのスクリプトが保存されているディレクトリから起動する必要があります (dfxc\_vcu118.tcl および dfxc\_kcu116.tcl)。

```
vivado -mode tcl -source project_dfxc_kcu116.tcl
```

または

```
vivado -mode tcl -source project_dfxc_vcu118.tcl
```

3. スクリプトが完了したら、「start\_gui」と入力して Vivado IDE を開きます。
4. IP をアップデートする必要があるかどうかを確認します。[Reports]→[Report IP Status] をクリックし、日付が古い IP があればそれをアップデートします。

新しいバージョンの Vivado を使用している場合、ILA など、若干のリビジョン変更がある可能性があります。このチュートリアルは Vivado 2020.1.\* バージョンにのみ使用してください。アップデートが必要な場合は、コア コンテナーがディスエーブルになっているデフォルト 設定を使用しますが、IP モジュールは合成しないので、そのプロセスは割愛してください。合成は次の手順で実行します。

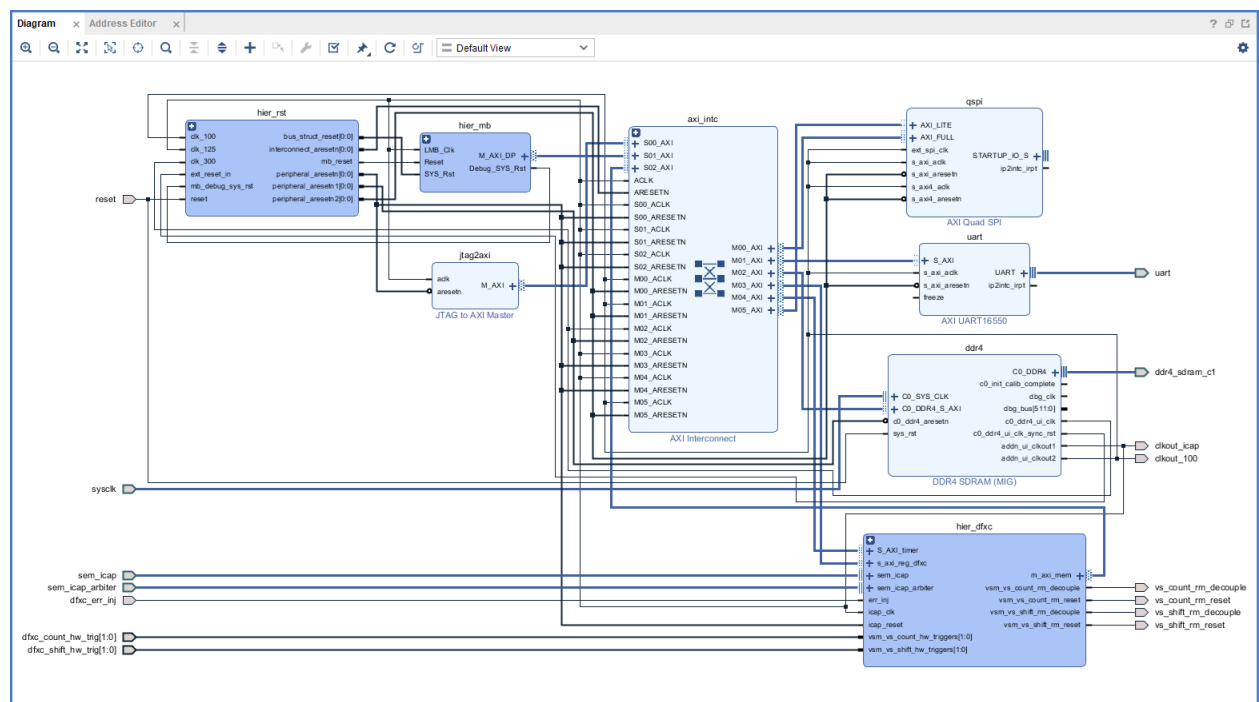
5. Flow Navigator で [Run Implementation] をクリックし、合成およびインプリメンテーションまでデザインを一気に処理します。

**注記:** このデザインは、デフォルト設定ですべてのタイミング制約をパスするはずですが、Vivado のバージョンによっては、追加操作が必要になる場合があります。その場合、[impl\_1] デザイン run を選択し、その run の [Options] タブで、[Place Design] および [Route Design to Explore] に `-directive` オプションを設定するのが最も簡単です。

配置配線を実行しているときに、デザインを確認します。最上位は基本的に LED、シフト、カウンターのデザインで、このチュートリアルのほかの演習で使ったデザインがベースになっています。このバージョンのデザインにはブロック図 (mb\_dfxc) が挿入されており、AXI サブシステム内のいくつかの機能を処理するようになっています。

- MicroBlaze がデザイン管理の中心になっていて、Uart を介してユーザー インターフェイスに接続します。
- DFX Controller IP で DFX イベントが管理されます。ILA コアおよびタイマーで、デザインで実行されている内容を視覚的に確認できます。
- DDR4 および QSPI インターフェイスがこのデザインに含まれています。

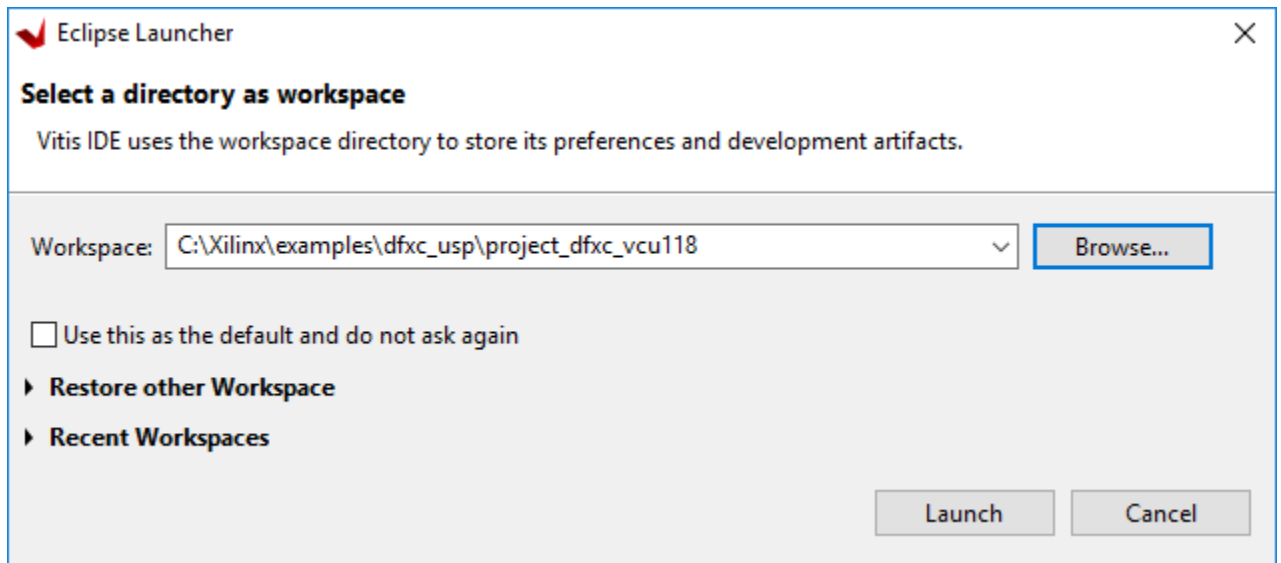
図 95: Dynamic Function eXchange 管理のブロック図



最上位デザインで、シフトおよびカウンターの RM がインスタンス化されます。また SEM IP もここに含まれています。VU9P が SSI デバイスなので、Virtex UltraScale+ と Kintex UltraScale+ とで SEM IP インスタンス化は少し異なります。SSI のサポートには、FRAME\_ECC コンポーネントの複数のインスタンス化 (SLR ごとに 1 つ) が必要です。

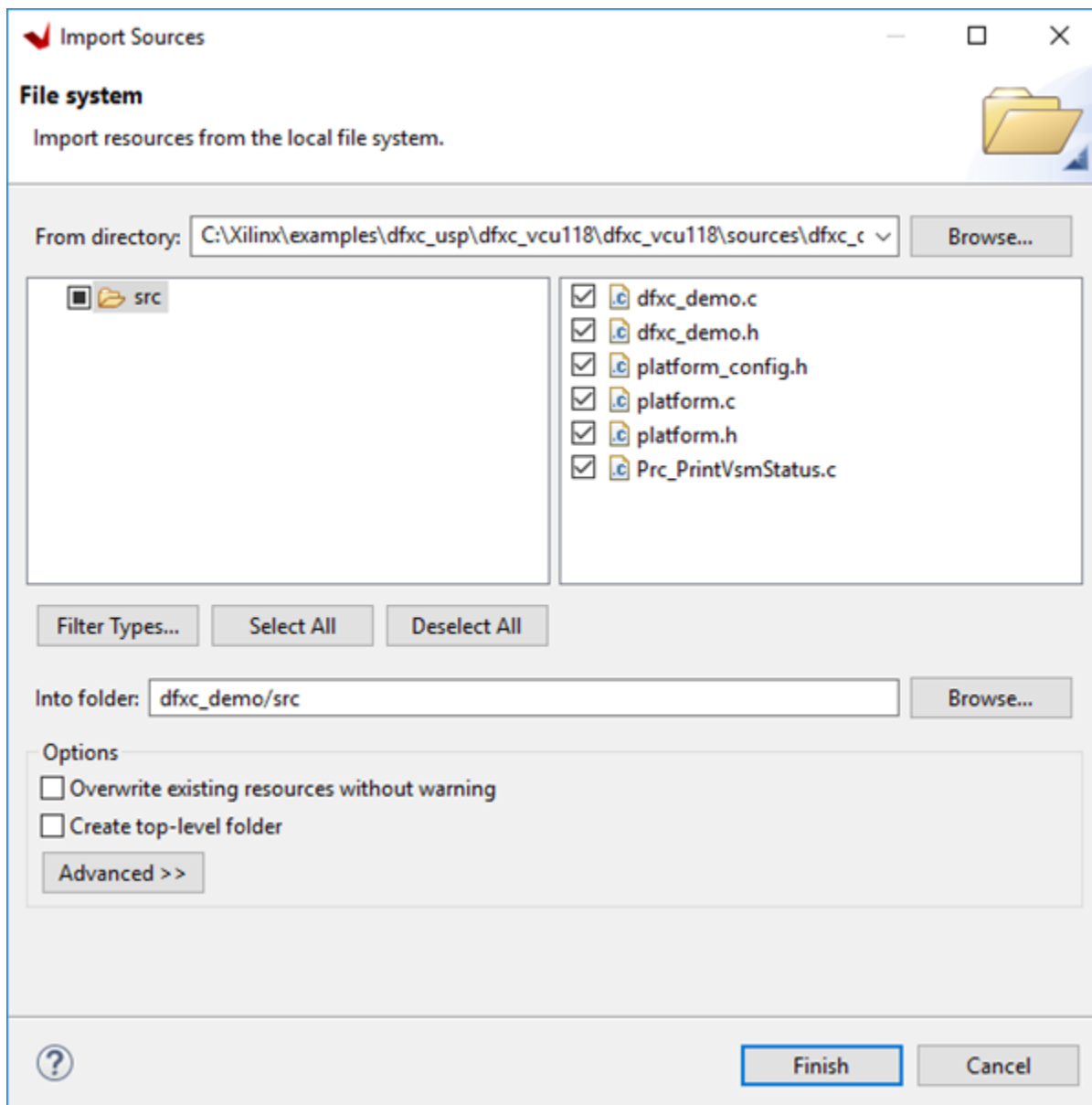
- インプリメンテーションが完了しても、ビットストリームは生成しないでください。ダイアログ ボックスが開くので、そこで [Cancel] をクリックします。
- [File] → [Export] → [Export Hardware] をクリックします。[Platform type] を [Fixed] に設定したまま [Next] をクリックし、[Output] を [Pre-synthesis] に設定したまま [Next] をクリックします。XSA ファイル名をデフォルトの「top」のままにして [Next] をクリックし、[Finish] をクリックして、Vitis™ ソフトウェア プラットフォームのデザイン イメージを構築します。
- [Tools] → [Launch Vitis IDE] をクリックすると、Eclipse Launcher のダイアログ ボックスが開きます。
- ワークスペースが現在のプロジェクト ディレクトリにマップされているのを確認し、[Launch] をクリックして、このサンプル デザインのソフトウェアをコンパイルします。

図 96: [Create a New Application Project] ページ



10. Vitis で [File]→[New]→[Application Project] をクリックします。
11. [Next] をクリックしてから [Create a new platform from hardware (XSA)] タブをクリックし、[top.xsa] を参照して、Vivado からエクスポートされたファイルをインポートします。[Next] をクリックします。  
**注記:** プロジェクトのソフトウェア プラットフォームはスタンドアロンで、言語は C 言語です。
12. [Next] をクリックします。新しいプロジェクトに「dfxc\_demo」という名前を付け、[Next] をクリックし、もう一度 [Next] をクリックします。
13. [Empty Application] を選択し、[Finish] をクリックします。
14. [Project Explorer] ウィンドウで [dfxc\_demo] を展開表示させます。[src] を右クリックして、[Import Sources] を選択します。sources/dfxc\_demo/src ディレクトリを指定し、[OK] をクリックします。最後に、そのフォルダー内にある C (.c) および H (.h) ソース ファイルのチェック ボックスを 6 つともすべてオンにして、[Finish] をクリックします。

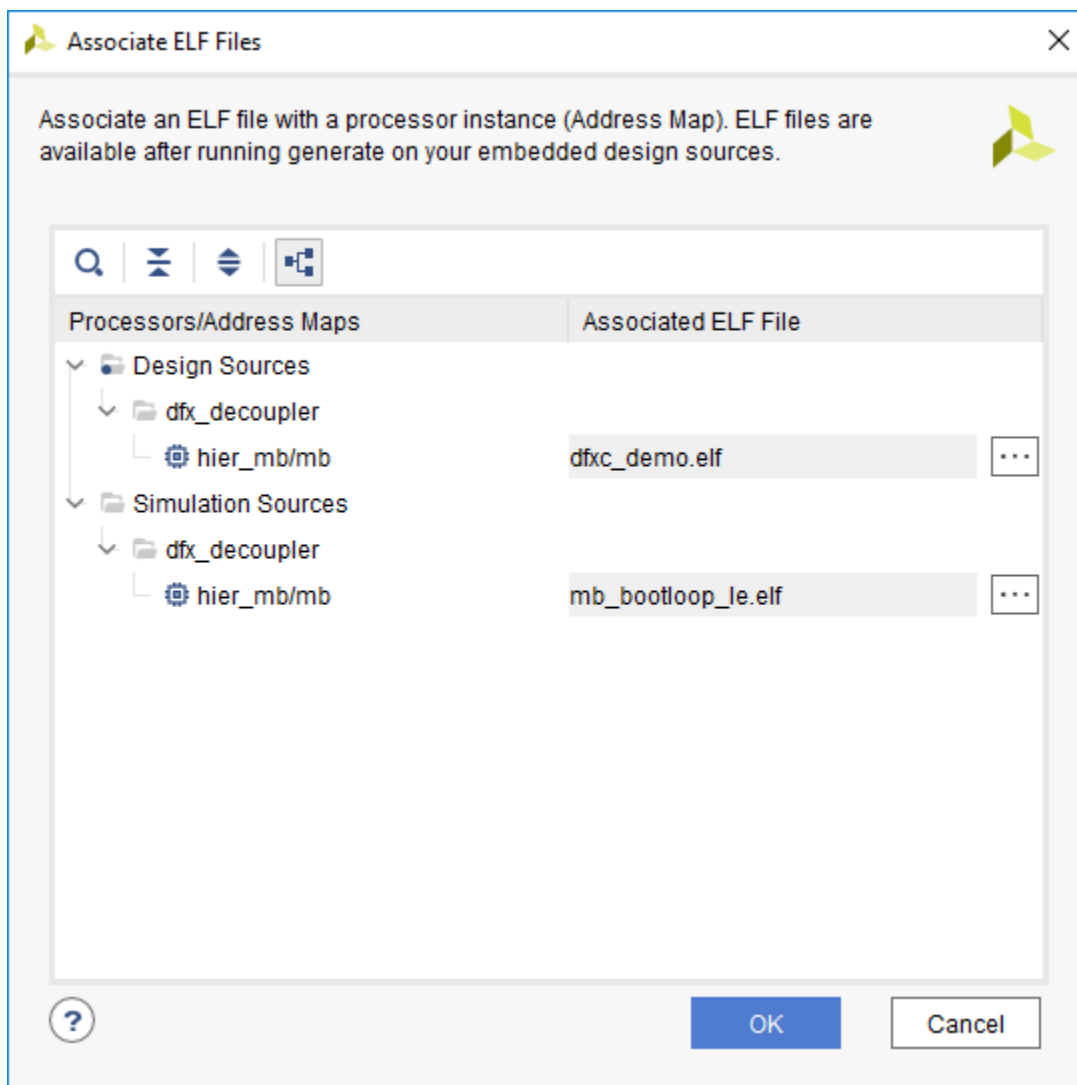
図 97: Vitis プロジェクトのソース ファイル



ファイルを追加したら、dfxc\_demo プロジェクトを右クリックし、[Build Project] をクリックします。これでプロジェクトがコンパイルされ、dfxc\_demo.elf ファイルが作成されます。プロジェクトのソースに変更が追加されるたびにプロジェクトを手動でビルドし、アップデートされた ELF ファイルを取得します。

15. src ディレクトリを展開させ、dfxc\_demo.c を開きます。このファイルには、後で確認するソフトウェア コードのほとんどが含まれています。パーシャル ビットストリームのロケーションおよびサイズは dfxc\_demo.h に格納されています。その計算は dfxc\_bitstream\_sizes\_lab7.xlsx で確認できます。
16. Vitis を終了します。
17. Vivado IDE で [File]→[Add Sources] をクリックします。
18. [Add or create design sources] をクリックしてから [Next] をクリックします。

19. `project_dfxc-$board/dfxc_demo/Debug` フォルダにある `dfxc_demo.elf` を追加します。[Copy Sources into project] をオフにして、[Finish] をクリックします。
20. [Sources] ウィンドウで `dfxc_demo.elf` ファイルを右クリックし、[Associate ELF Files] をクリックします。デザインは既にコンパイルされているので、[Skip Generate] を選択します。
21. ウィンドウが表示されるので、そこでデザインソースの一番上に表示されている参照ファイルを、先ほど追加した `dfxc_demo.elf` ファイルに変更します。[OK] をクリックします。

 図 98: MicroBlaze のインスタンスに `prc_demo.elf` ファイルを関連付ける


22. [Design Runs] ウィンドウで `[impl_1]` を右クリックして [Generate Bitstream] を選択し、[OK] をクリックします。

この操作で、右シフトおよびカウントアップのコンフィギュレーションのフル ビットストリーム (ELF ファイルに MicroBlaze コードが含まれている) と、各 RM のパーシャル ビットストリームが作成されます。ここでは、親 run からのフル コンフィギュレーション ビットストリームのみが使用されるので、`child_0_impl_1 run` からビットストリームを生成する必要はありません。すべてのパーシャル ビットストリームのサイズは、ログにビット数でレポートされます。

23. 次のスクリプトを実行して、すべてのビット ファイルを作成します。Tcl コンソールで、このスクリプトのある `project_dfxc-<board>` ディレクトリのすぐ上のディレクトリにいることを確認します。

```
source create_all_bitstreams_kcu116.tcl
```

または

```
source create_all_bitstreams_vcu118.tcl
```

ビットストリームをフルにするかパーシャルにするかの設定は、このデザインで設定されているコンフィギュレーション モードおよびオプションを考慮して、異なっている必要があります。これは、前出の手順で作成したフル ビットストリームをコピーしてから、すべての RM に必要なパーシャル ビットストリームをすべて作成する Tcl スクリプト内で設定されます。



**重要:** Vivado IDE のプロジェクト モードでは、ビットストリームをフルにするか、パーシャルにするかなどのさまざまなオプションをユーザーが設定できません。この機能は、今後の Vivado リリースで導入され、DFX ウィザードに新たなページが加えられる予定です。

このスクリプトを確認してみると、CONFIG\_MODE をデフォルトの SPIx4 (初期コンフィギュレーション) から SELECTMAP32 (ICAP に配信されるパーシャル ビットストリーム) に変更する必要があることがわかります。2 つのバージョンのパーシャル ビットストリームが生成され、1 つはフレームごとの CRC 機能がイネーブルになっているもの、もう 1 つはそうでないものです。

24. これら 2 つのスクリプトの 1 つを実行し、ターゲット ボードの PROM イメージを作成します。

```
source create_prom_file_kcu116.tcl
```

または

```
source create_prom_file_vcu118.tcl
```

これで、QSPI ブート イメージが作成されます。まずはフル ビットストリームのイメージ、そしてその後にすべてのパーシャル ビットストリームが続きます。このスクリプトにリストされているアドレスは、各パーシャル ビットストリームのサイズに基づいて計算されています。すべてのパーシャル ビットストリームのサイズは、ログにビット数でレポートされています。その計算は `dfxc_bitstream_sizes_lab7.xlsx` で確認できます。

## 手順 3: チュートリアル デザインの実行

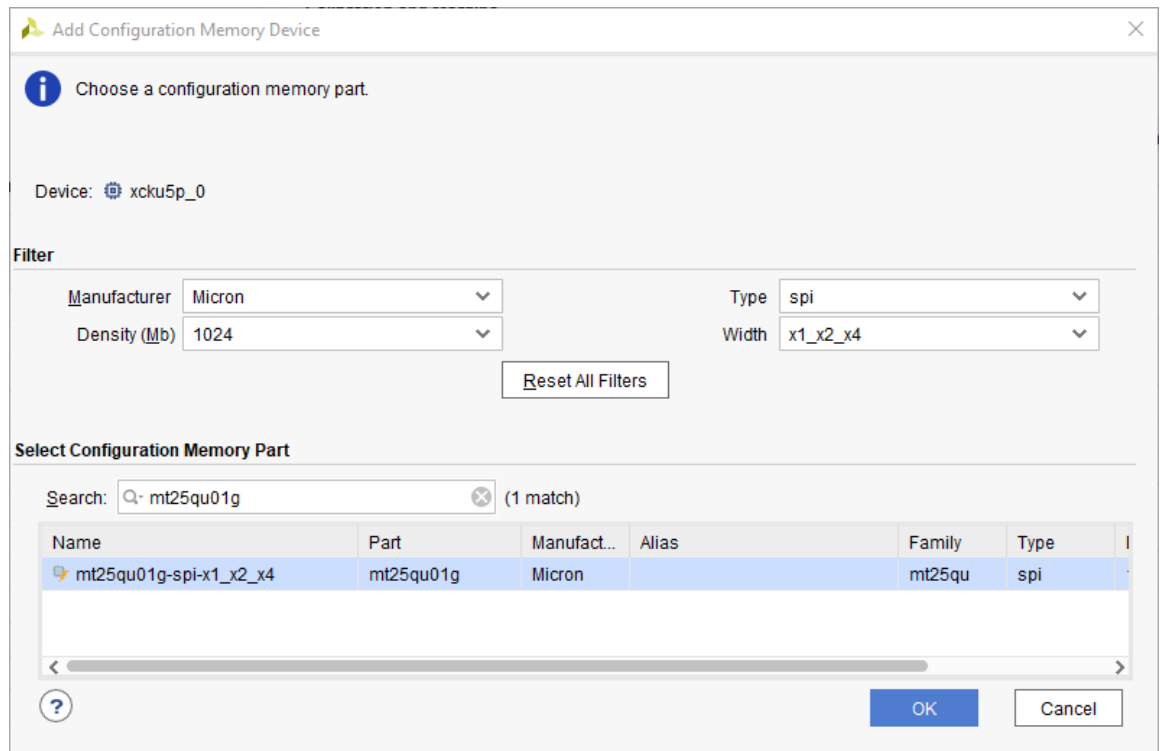
すべてのビットストリームおよび PROM イメージを作成したら、ハードウェアでデザインを実行できます。このチュートリアルで実際に確認できる機能は数多くあります。デバイスがプログラムされた後は、どの順番でこれらの機能を確認しても問題ありません。

### QSPI フラッシュ デバイスのプログラム

1. コンピューターにターゲット ボードを接続し、そのボードに電源を投入します。JTAG および UART 接続用の micro-USB ポートを両方接続します。
2. コンフィギュレーション メモリ デバイスをプログラムします (QSPI)。
  - a. Vivado ハードウェア マネージャーを開き、ターゲット ボードに接続します。
  - b. デバイスを右クリックし、[Add Configuration Memory Device] をクリックします。
  - c. x1、x2 および x4 モードをサポートする Micron の mt25qu01g を選択します。



図 99: QSPI フラッシュのプログラム



- d. プロンプトが表示されたら、プログラミング ファイルを追加します。ターゲット ファイルは、プロジェクト ディレクトリのビットストリーム フォルダにある `dfx_prom.mcs` です。
3. PROM がプログラムされたら、PROG ボタンを使用して、このブート フラッシュから FPGA をリコンフィギュレーションします。

プッシュボタンは FPGA デザインでの作業を制御します。左右のボタンは、左シフトおよび右シフトのパーシャル ビットストリームをそれぞれ読み込みます。上下のボタンは、カウントアップおよびカウントダウンのパーシャル ビットストリームをそれぞれ読み込みます。中央のボタンはまだ押さないでください。

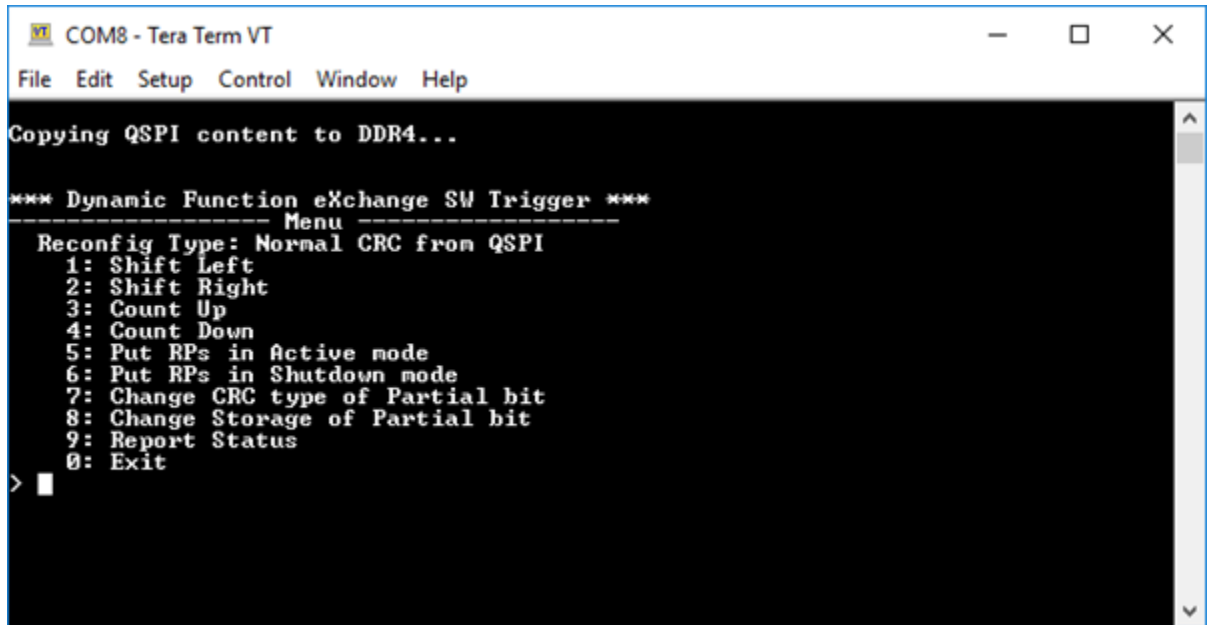
## ソフトウェアを介したリコンフィギュレーションの管理

1. MicroBlaze で実行しているソフトウェアと通信するため、UART ターミナルを開きます。
  - a. 使用しているコンピューターに適切な値に COM ポートを設定します。
  - b. ボーレートを 115200 に設定します。
  - c. ボード上の [PROG] ボタンを押して、UART ターミナルを開いた状態でデザインを再開します。

**注記:** UART ドライバーへの USB が必要な場合は、『Silicon Labs CP210x USB-to-UART』([UG1033](#)) を参照してください。

パーシャル ビットストリームがすべて QSPI フラッシュ デバイスから DDR4 メモリにコピーされている点に注目してください。ソフトウェアのメニューは次のようになります。

図 100: pr\_demo ソフトウェア アプリケーションの実行



次のメニュー オプションがあります。

- 1 - 4: DFX Controller を介して ICAP にパーシャル ビットストリームを読み込むための 4 つのトリガー オプションです。ボード上のプッシュボタンと同じです。
- 5 - 6: 両方の RP に対し、DFX Controller のステータスをアクティブからシャットダウンに、またはその逆に切り替えます。RP がシャットダウン モードの場合、トリガー (ソフトウェアまたはハードウェア) は無視されます。
- 7: 使用されているパーシャル ビットストリームを、標準パーシャルからフレームごとの CRC がイネーブルになっているパーシャルへ、またはその逆に切り替えます。
- 8: パーシャル ビットストリームの格納場所を切り替えます (QSPI または DDR4 メモリ)。
- 9: 各仮想ソケットの現在のステータスをレポートします。

これらの機能を使用すると、ソフトウェアからフィードバックを得ることができます。リコンフィギュレーション時間は、ソフトウェアからリコンフィギュレーションを実行するたびにレポートされます。

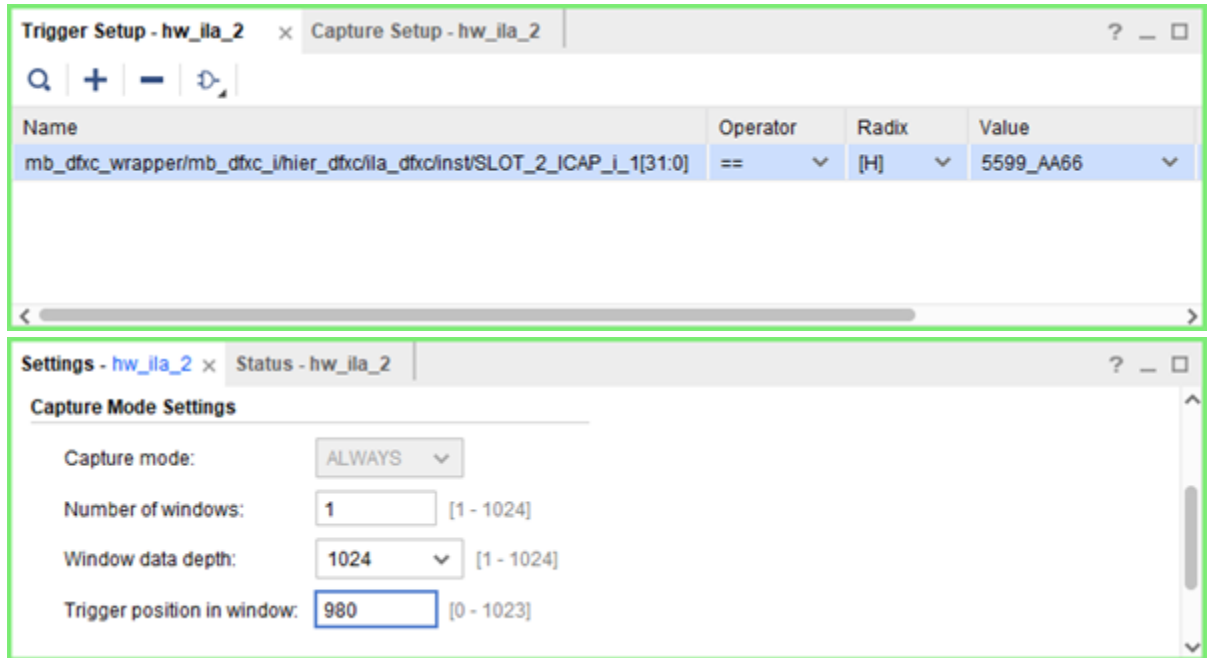
## デバッグ コアを介した Dynamic Function eXchange の監視

このデザインには Vivado デバッグ コアが挿入されていて、Dynamic Function eXchange イベント中のアクティビティを監視できます。ICAP を介したコンフィギュレーションの場合、各 BIN ファイルに正しいビットおよびバイト順序を設定し、パーシャル ビットストリームを準備しておくことが重要です。

1. Vivado ハードウェア マネージャーで、デバイスをリフレッシュして、すべての Vivado デバッグ コアを検索します。このデザインには、ILA コアが 3 つ、VIO コアおよび MIG コアが 1 ずつあります。
2. [Hardware] メニューでパーツを右クリックし、[Hardware Device Properties] をクリックします。[General] タブでプローブ ファイルを `Bitstreams/top_count_up_shift_right.ltx` にポイントさせます。
3. [hw\_ila\_2] で [+] をクリックし、[Trigger Setup] ウィンドウでプローブを追加します。
4. [SLOT\_2\_ICAP\_i\_1[31:0]] を選択し、[OK] をクリックします。[Radix] を 16 進数を表す [H] に変更します。

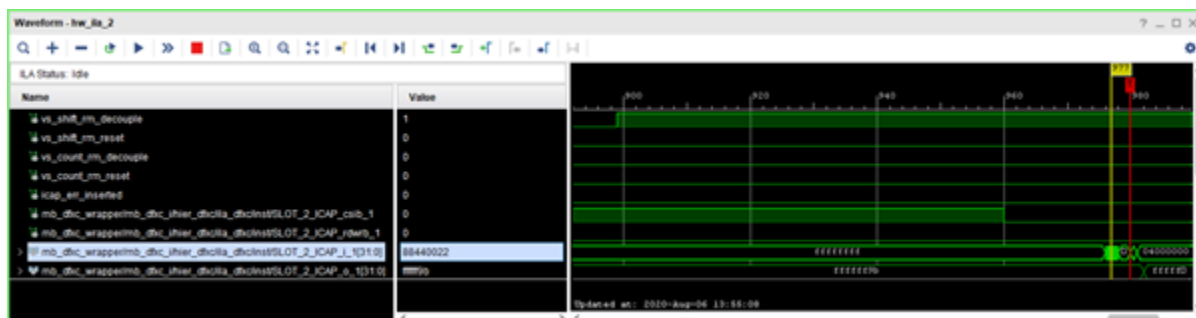
5. [Trigger Setup] ウィンドウの [Value] に、コンフィギュレーション同期ワード (ビットスワップ) の値である 5599\_AA66 を設定します。
6. [Settings] ウィンドウで [trigger position in window] を 980 に設定します。

図 101: 同期ワードの ICAP 入力キャプチャ設定



7. ハードウェア マネージャーの GUI で [Run Trigger] ボタンをクリックします。
8. ボード上で、中央ボタン以外のプッシュボタンをどれか 1 つ押し、シフトまたはカウンターのリコンフィギュレーションをトリガーさせます。または、UART ターミナルを介してこの操作を実行します。
9. この結果取り込まれた波形で、次の点を確認します。
  - 一番左にある `rm_decouple` 信号の 1 つ (リコンフィギュレーションにどの RP を指定しているかによる) が Low から High に遷移しています。これは、パースシャル ビットストリームが配信される前にデザインで隔離されています。
  - 同期ワードの前には、000000dd があり、その次に 88440022 があります。これらは、ビットスワップされたバス幅が検出されたことを示しています。
  - ICAP 出力は ffffff9b (同期なし、エラーなし) から fffffdb (同期、エラーなし) へ遷移しています。この遷移は同期ワードが認識されたことを示し、コンフィギュレーション エンジンにビットストリーム データが送信されるはずです。
  - 波形のかなり右のほうでは、PRDONE が High から Low へと遷移し、この波形の範囲外にあります。

図 102: 同期ワードが ICAP に遷移する状態をキャプチャした波形



**注記:** ビットストリームは複数のセグメントから構成されているので、ビットストリームの中には複数の同期/非同期ペアがあります。たとえば、マルチ SLR デバイスには、SLR ごとにビットストリームがフォーマットされているので、このペアがさらに多くあります。

10. ICAP\_i ポートの値を 0000\_00B0 (非同期ワード、ビットスワップ) に変更します。
11. [trigger position in window] を 512 に設定します。
12. もう一度トリガーを出力し、リコンフィギュレーションを実行します。

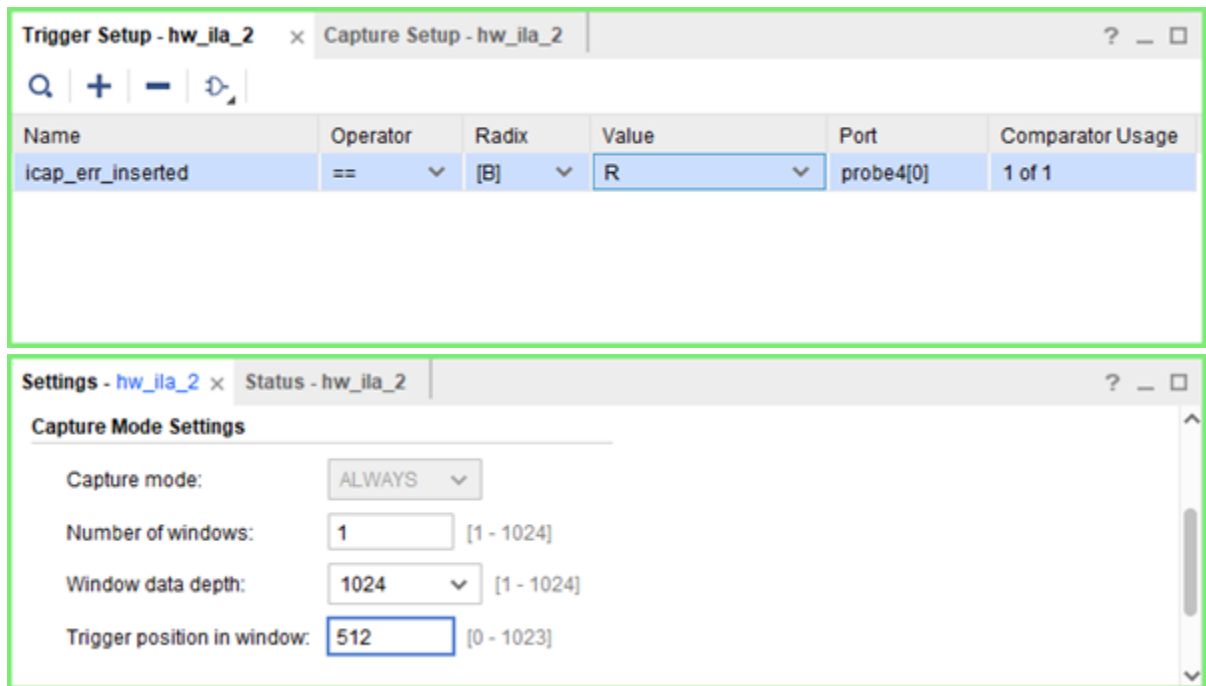
結果として出力された波形には、リコンフィギュレーション シーケンスのこの部分の終わりが示されています。また、非同期ワードが確認された後に PRDONE が数サイクル間 High になっています。

## CRC エラーの挿入および FPGA 応答の表示

パースシャル ビットストリームは、フレームごとの CRC チェックの有無にかかわらず、作成され、PROM ファイルの一部として QPSI フラッシュ デバイスに読み込まれました。これで、ICAP にファイルを読み込む直前に CRC 値のいくつかのビットをスワップし、CRC エラーを挿入できます。圧縮されていないパースシャル ビットストリーム (DFX Controller の機能ではなく、ビットストリーム生成を使用したもの) には、デバイスの応答を確認するため、この方法でエラーを挿入できます。これは、中央のプッシュボタンで制御されます。

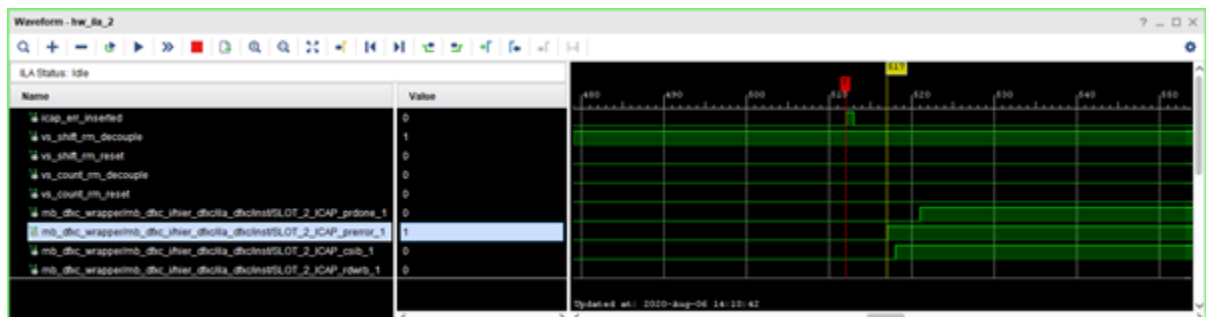
1. 前出の手順でまだ作業をしていない場合: Vivado ハードウェア マネージャーでデバイスをリフレッシュさせ、Vivado デバッグ コアをすべて検索します。  
このデザインには、ILA コアが 3 つ、VIO コアおよび MIG コアが 1 ずつあります。
2. 前出の手順でまだ作業をしていない場合: ILA コア ウィンドウの 1 つで [Specify the probes file] リンクをクリックし、Bitstreams/top\_count\_up\_shift\_right.ltx を検索します。
3. [OK] をクリックして、[Refresh] をクリックします。
4. [hw\_ila\_2] で [+] をクリックし、[Trigger Setup] ウィンドウでプローブを追加します。
5. [icap\_err\_inserted] を選択して、[OK] をクリックします。
6. [Settings] ウィンドウでトリガーを icap\_error\_inserted の立ち上がりエッジに設定します。
7. [Settings] ウィンドウで [trigger position in window] を 512 に設定します。

図 103: icap\_err\_inserted のトリガー設定



- ハードウェア マネージャーの GUI で [Run Trigger] ボタンをクリックします。
- UART ターミナルで、[Reconfig Type] を [Normal CRC from DDR4] に設定します (オプション 7)。これがデフォルト設定です。
- 中央のプッシュボタンを押し (これで CRC エラーが挿入されます)、プッシュボタンまたは UART ターミナルを介して必要なモジュールをリコンフィギュレーションします。パーシャル ビットストリームの最後の CRC 値がスワップされ、CRC エラーが発生させます。この結果、INIT\_B が Low になり (INIT\_B LED が赤く点灯)、CRC エラーが発生していることを示します。ボード上では、リコンフィギュレーションされていない機能がまだ動作していることに注意してください。
- ILA の波形に、icap\_err\_inserted がアサートされるトリガー位置が記されます。エラー挿入後、PRERROR がまず High になってから、PRDONE が High になります。また、rm\_decouple 信号もずっと High の状態を保ち、RP がまだ隔離されていることを示します。

図 104: icap\_err\_inserted、PRERROR、PRDONE およびデカップルがアサートされた状態の波形



**注記:** dfxc\_vsm\_vs\_\*\_event\_error 信号は Low ですが、Dynamic Function eXchange (DFX) Controller のレイテンシのため、キャプチャされた波形の外側で High になります。

標準 CRC で CRC エラーが検出される場合 (パースシャル ビットストリームの最後でのみ検出される)、間違ったビットストリームが既にデバイスに読み込まれています。どこに間違ったビットがあるのかを知ることはできませんし、その悪影響がリコンフィギュラブル デザインに出るのか、スタティック デザインに出るのかは不明です。この状態から確実にフル リカバリするには、デバイスをフル リコンフィギュレーションするしかありません。このチュートリアルでは、CRC 値のみがスワップされるので、エラーがスタティック デザインに影響を及ぼすことはありません。

12. CRC エラーが発生すると、DFX Controller がシャットダウン モードになります。UART ターミナルで [Report Status] (9 番目のオプション) を見ると、RP がシャットダウン モードになっていて、BS ERROR がレポートされていることがわかります。このエラー ステータスから回復させるには、ターミナルの [Put RPs in Active Mode] (5 番目のオプション) を選択して、RP をアクティブ モードに戻し、正しいパースシャル ビット ファイルでリコンフィギュレーションします。その後、INIT\_B は High に戻り (LED は緑色に点灯)、デザインが標準動作に戻ります。

次は、フレームごとの CRC 値を入力する方法で試してみます。

13. ターミナルで、[Reconfig Type] を [per Frame CRC] (7 番目のオプション) に設定します。
14. ハードウェア マネージャーで [Run Trigger] ボタンをクリックして、再度トリガーします。次に、ターミナルまたはプッシュボタンから Dynamic Function eXchange を実行します。
15. ILA の波形では、パースシャル ビットストリームの最初のフレームに挿入されたエラーを確認できます。vs\_rm\_\*\_decouple が High になった後まもなく、リコンフィギュレーションが開始するのが確認できます。ただし、フレームごとの CRC を使用する場合は、エラーが挿入されたフレームがデバイスにはまだ読み込まれていないので、フル デザインをリコンフィギュレーションする必要はありません。完成していない RP を有効なパースシャル ビットストリームでリコンフィギュレーションするだけで十分です。エラー ステータスから回復させるには、手順 8 からの作業を繰り返します。

## まとめ

これで演習 7 は終了です。この演習では、次の作業を実行しました。

- Dynamic Function eXchange (DFX) Controller を含む UltraScale+ バージョンのデザインをインプリメントしました。
- Dynamic Function eXchange イベントを管理するソフトウェアを使用して MicroBlaze コアをインプリメントしました。
- KCU116 または VCU118 ボード上で QSPI をプログラムしました。
- VCU118 ボード上で QSPI をプログラムしました。
- QSPI または DDR4 メモリからの Dynamic Function eXchange を管理するため、UART インターフェイスを使用しました。
- CRC の機能を確認するため、ビットストリームの配信エラーを挿入しました。

**注記:** デザイン内に SEM コアはありますが、この演習のハードウェアテストでは使用しません。エラー イベントを検出するため、ハードウェア上で SEM コアを実行している場合は、Dynamic Function eXchange を実行する前に、それを一時停止する必要があります。

## 演習 8

# Nested DFX

## 概要

この演習では、4 つの UltraScale™ または UltraScale+™ デモ ボードのうちの 1 つをターゲットにした Nested Dynamic Function eXchange (Nested DFX) の簡単な例を取り上げます。Nested DFX は、ダイナミック領域内に 1 つまたは複数のダイナミック領域を配置し、デバイスを細分化して、より粒度の細かい再構成を可能にするという概念です。この機能を使用すると、それぞれがパーシャル リコンフィギャラブルなリコンフィギャラブル パーティション (RP) をより小さな領域に分割できます。

この演習のデザインは、この資料のほかの演習で使用されている LED シフト カウントの設計を変更したものです。単に異なるシフターまたはカウンターを入れ替えるのではなく、現在のデザインに 2 つのシフターまたは 2 つのカウンターを持つことができるように、リコンフィギャラブルな層が追加されています。これらのシフターまたはカウンターは、それぞれ部分的にリコンフィギャラブルです。

Nested DFX はプロジェクト モードではまだサポートされていないため、デザイン フローでは演習 2 で使用した Tcl スクリプト ソリューションを使用します。このチュートリアル内のセグメント化された Tcl スクリプトに示されているように、明示的な命令に従うか、完全デザインを最初から最後まで実行する単一のフル スクリプトを使用する選択肢があります。

## 手順 1: チュートリアル デザイン ファイルの抽出

1. チュートリアル デザイン ファイルの取得方法については、チュートリアル デザインの説明を参照してください。
2. 抽出したディレクトリ内で `\nested_dfx` に移動します。`nested_dfx` データ ディレクトリは、この演習では `<Extract_Dir>` と表記されます。

## 手順 2: スクリプトの確認

まずは、デザイン アーカイブにある Tcl スクリプトを確認します。



## 合成スクリプト

run\_synth.tcl、design\_settings.tcl、advanced\_settings.tcl ファイルは、ルート ディレクトリにあります。run\_synth.tcl スクリプトには、この Dynamic Function eXchange デザインの合成部分の実行に必要な最低限の設定が含まれています。design\_settings.tcl スクリプトは、ターゲット デバイスとボードを選択し、プロジェクトの相対パスを設定します。advanced\_settings.tcl には、デフォルト フローの設定が含まれているので、上級ユーザーでない限り、このファイルは変更しないでください。

この演習用の run\_synth.tcl では、フロー制御の下で、どのモジュールを合成するかを制御できます。スクリプト名のとおり、このスクリプトで実行されるのは合成のみです。インプリメンテーション、検証、ビットストリーム生成はインタラクティブに実行されます。演習 2 の DFX スクリプト全体は、現在のところ Nested DFX 用には設定されていません。

design\_settings.tcl の「Define target demo board」の下で、このデザインでサポートされている 4 つのデモ ボードの中からボードを 1 つ選択できます。スクリプトは VCU118 用なので、別のボードをターゲットにする場合は、ここで編集します。この演習では、次のザイリンクス開発プラットフォームをターゲットにしています。

- KCU116 (Kintex® UltraScale+™)
- VCU118 (Virtex® UltraScale+™)
- KCU105 (Kintex UltraScale)
- VCU108 (Virtex UltraScale)

## Nested DFX スクリプト

この演習では、合成後の Nested DFX フローを個々の Tcl コマンドごとに手順を追って説明します。この演習は、リコンフィギャラブルな第 2 層を挿入するのに必要な独自の詳細を示すことを目的としているので、これを達成するのに必要な新しい手順を説明しますが、ソリューション全体をスクリプト化することもできます。特定のセクションは、フローのサブセクションをコンパイルするために実行可能なスクリプトにグループ分けされます。これらのスクリプトでは明示的な名前とパスが使用されますが、変更して新しいデザインで使用することもできます。次がそのスクリプトです。

- `implement_parent_config.tcl`: 最上位のスタティック デザインをインプリメントし、後で分割される一次リコンフィギャラブル パーティション (inst\_RP) を構築します。
- `subdivide_shifters.tcl`: 一次リコンフィギャラブル パーティションをそれぞれ部分的にリコンフィギュレーション可能な 2 つの二次シフト ファンクションに分割します。
- `subdivide_counters.tcl`: 一次リコンフィギャラブル パーティションをそれぞれ部分的にリコンフィギュレーション可能な 2 つの二次カウント ファンクションに分割します。
- `implement_sub_shifters.tcl`: inst\_RP の下の 2 つの二次 RP に shift\_right および shift\_left リコンフィギャラブル モジュールをインプリメントします。
- `implement_sub_counters.tcl`: inst\_RP の下の 2 つの二次 RP に count\_up および count\_down リコンフィギャラブル モジュールをインプリメントします。
- `verify_configurations.tcl`: デザイン チェックポイントのペアで pr\_verify を実行し、その中に含まれるリコンフィギャラブル モジュールの互換性を確認します
- `generate_all_bitstreams.tcl`: チェックポイントを 1 つずつ開いて、このデザイン全体に対して可能なすべてのパーシャル ビットストリームを作成します
- `run_all.tcl`: 合成からビットストリーム生成まで、上記のすべてのスクリプトを実行して、チュートリアル デザイン全体をコンパイルします。

デフォルトのボードは VCU118 ですが、ほかに 3 つのボードを選択できます。ボードは、`design_settings.tcl` 内で 1 回だけ選択できます。値は各インプリメンテーション スクリプトで取得されるので、新しい Vivado セッションが起動された場合でも、デザイン設定が認識されます。

## 手順 3: デザインの合成

`run_synth.tcl` スクリプトは、このチュートリアル of 合成フェーズを自動化します。最上位のスタティック デザインに 1 回、一次リコンフィギュラブル モジュールに 2 回、二次リコンフィギュラブル モジュールに 4 回の、合計 7 回、合成の繰り返し実行が呼び出されます。

1. Vivado Tcl シェルを開きます。

Windows の場合は、ザイリンクスの Vivado デスクトップアイコンまたは [Start] → [All Programs] → [Xilinx Design Tools] → [Vivado 2020.1] → [Vivado 2020.1 Tcl Shell] をクリックします。

Linux の場合は「`vivado -mode tcl`」と入力します。

2. シェルで、`<Extract_Dir>` ディレクトリに移動します。
3. ターゲットボードが `run_synth.tcl` の `xboard` 変数で選択されていることを確認します。
4. 次を入力して、`run_synth.tcl` を実行します。

```
source run_synth.tcl -notrace
```

Vivado 合成で 7 回の繰り返し実行をすべて完了しても、Vivado Tcl シェルは開いたままです。各モジュールのログやレポート ファイル、最終チェックポイントは、`Synth` サブディレクトリに各モジュール名の付いたフォルダーがあって、その中に生成されています。



**ヒント:** `<Extract_Dir>` ディレクトリには、複数のログ ファイルが作成されます。

1. `run.log` には、Tcl シェルのウィンドウに表示されたサマリが書き込まれます。
2. `command.log` には、スクリプトにより実行された個々のステップすべてが書き込まれます。
3. `critical.log` には、実行中に出力されたクリティカル警告メッセージがすべて書き込まれます。

## 手順 4: デザインのまとめおよびインプリメンテーション

各モジュールおよび最上位の合成済みチェックポイントが生成されたので、デザインをまとめる準備が整いました。すべてのフロー ステップは Tcl コンソールから実行されますが、インタラクティブなイベントに対しては IDE 内の機能 (フロアプランニング ツールなど) を使用できます。

### インプリメンテーション デザイン フロー

この演習では、Nested DFX デザインの各コンフィギュレーションのインプリメントに使用されるコマンドを順を追って実行する Tcl スクリプトのセットを使用します。実行する前に、各スクリプトのそれぞれの動作を確認してください。ほとんどのコマンド (`link_design`、`route_design`、`pr_verify`、`write_bitstream` など) はこれまでと同じですが、`pr_subdivide` や `pr_recombine` は新しいものです。後のスクリプトが前のスクリプトに依存するため、重要なのは実行される順序です。

スクリプトが完了したら、チェックポイント (たとえば `top_route_design.dcp` または `top_count_up_up_route_design.dcp`) を開いて結果を確認し、それらの作成に使用された DFX 属性とコマンドの影響を確認します。

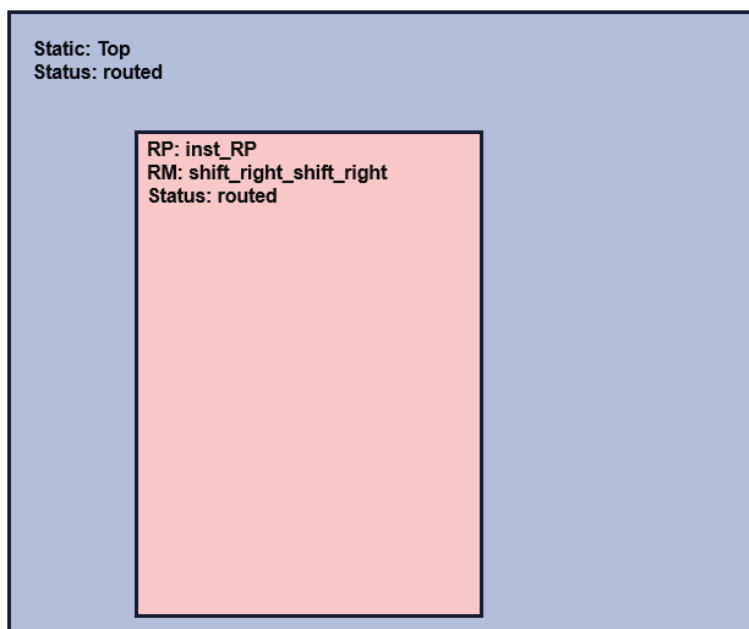
最初のインプリメンテーション デザインの実行では、スタティック デザインと一次リコンフィギャラブル パーティションが構築されます。この段階では、フローは標準的な DFX デザインフローと異なりません。inst\_RP はデザイン内で唯一の RP であり、そのレベルよりの下に存在するシフター モジュールはその inst\_RP ロジックの残りを使用してインプリメントされます。二次リコンフィギャラブル パーティションはまだ存在しません。

1. 実行スクリプトを読み込んで、親コンフィギュレーションをインプリメントします。

```
source implement_parent_config.tcl -notrace
```

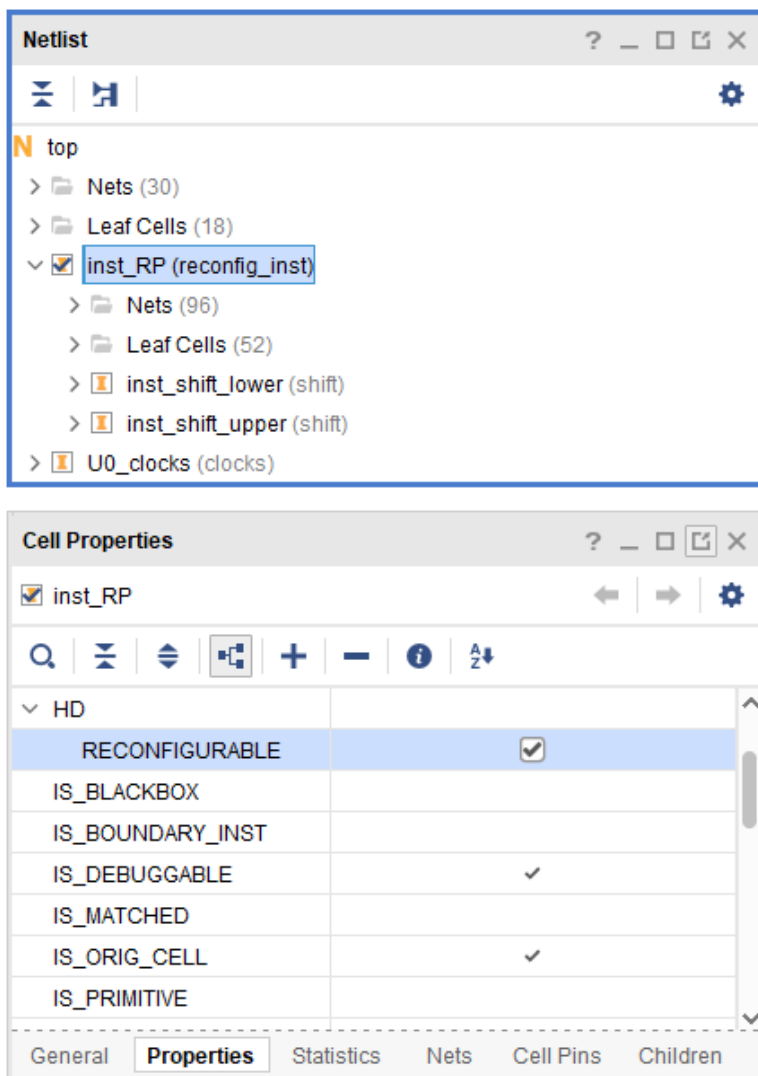
作成されるチェックポイント (`top_route_design.dcp`) は、単一の RP を含む完全なデザイン イメージです。この時点では、RP をブラック ボックスにしたり、スタティック デザインをロックしたりするなどの追加の DFX ステップは実行されていません。このチェックポイントは、ロックされたスタティック デザイン イメージを作成するためにのみ使用され、以降のすべてのデザイン イテレーションで共通のものになります。

図 105: 一次インプリメンテーション



`implement/top_static` フォルダに書き込まれた `top_route_design.dcp` を開き、これが標準の DFX であることを確認します。inst\_RP には、HD.RECONFIGURABLE プロパティと、そのリコンフィギャラブル パーティションの関連 Pblock が含まれます。

図 106: 一次インプリメンテーション後にリコンフィギャラブルなパーティションは inst\_RP のみ



- このスクリプトを読み込んで、二次リコンフィギャラブル パーティションを作成します。

```
source subdivide_shifters.tcl
```

このスクリプトは、inst\_rp モジュールを二次リコンフィギャラブル パーティションに分割します。pr\_subdivide コマンドは、inst\_RP から HD.RECONFIGURABLE プロパティを削除し、inst\_shift\_upper と inst\_shift\_lower の両方に適用します。この後、inst\_RP に HD.RECONFIGURABLE\_CONTAINER プロパティがタグ付けされ、これがかつて RP であったことが示されます。これは、top\_static\_shifters.dcp チェックポイントを見るとわかります。

図 107: シフター ファンクションの pr\_subdivide 後のデザイン

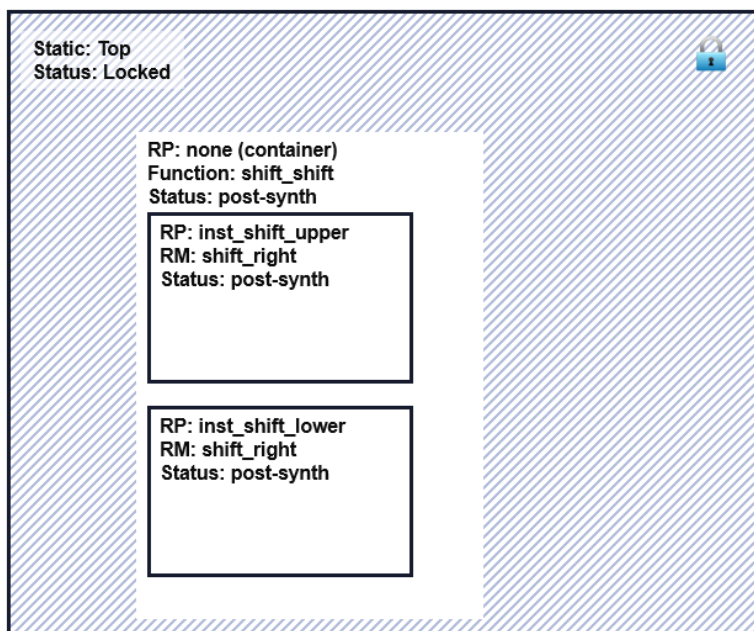
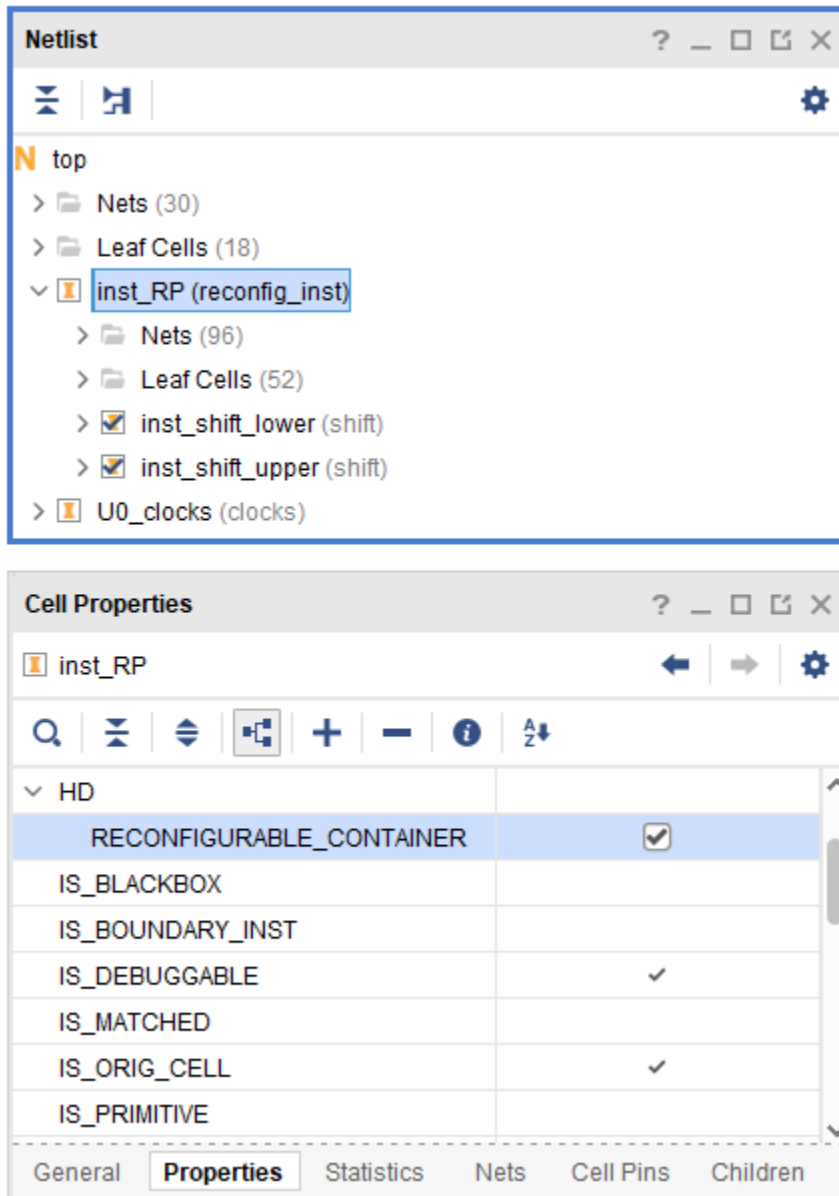


図 108: HD.RECONFIGURABLE\_CONTAINER プロパティの付いた inst\_RP



3. 二次リコンフィギャラブル パーティションにシフター サブモジュールをインプリメントします。

```
source implement_sub_shifters.tcl -notrace
```

これで、二次 RP に shift\_right および shift\_left ファンクションを配置配線する 2 つインプリメンテーション フローが実行されます。ここで使用されるコマンドは、最初のコンフィギュレーションの開始点にロックされた最上位スタティック デザインが含まれるという点を除き、標準 DFX フローと同じです。インプリメンテーションでは、inst\_RP レベル (reconfig\_shifters) の論理デザインがスタティックとして扱われます。これは、最初のコンフィギュレーション完了後に lock\_design -level routing コマンドでロックされる階層レベルです。

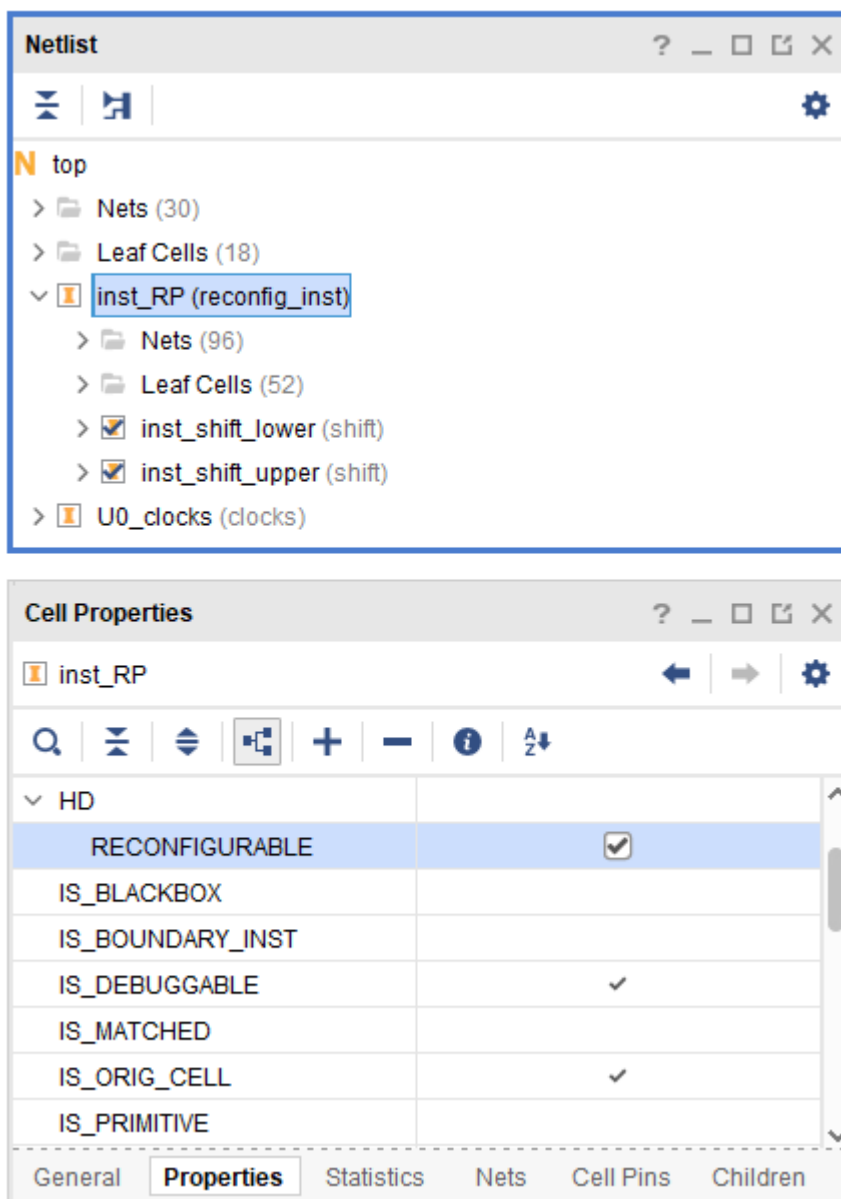
図 109: `shift_right` をインプリメントしてから `shift_left` を 2 つの RP にインプリメントする 2 つの二次 run


このスクリプトは、`pr_recombine` の呼び出しで終了し、`shift_right` と `shift_right` の組み合わせの配線済みデザインチェックポイントを作成し、`HD.RECONFIGURABLE` プロパティを `inst_RP` レベルに戻します。

`top_shift_right_right_recombined.dcp` の階層を見ると、このプロパティが `inst_RP` インスタンスに返されたことがわかります。



図 110: 再結合されたシフター コンフィギュレーションのデザイン階層とプロパティ



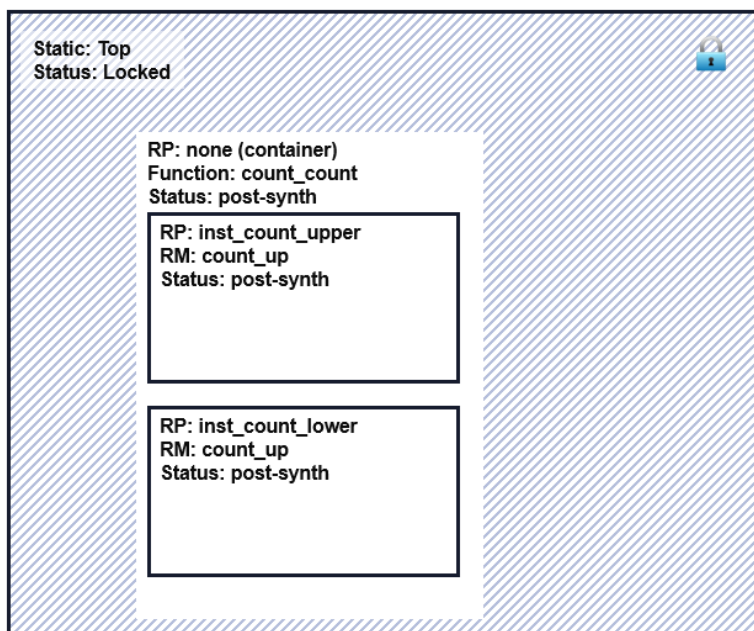
- このスクリプトを読み込む、もう 1 つの二次リコンフィギャラブル パーティションを作成します。

```
source subdivide_counters.tcl
```

最初の分割スクリプトと同様、これは初期コンフィギュレーション (top\_route\_design.dcp) から開始し、inst\_RP レベルを分割しますが、今回は 2 つのカウンター ファンクションに分割します。このデザイン バージョンの最上位スタティックは、シフターに使用されるバージョンと同じです。



図 111: カウンター ファンクションの pr\_subdivide 後のデザイン



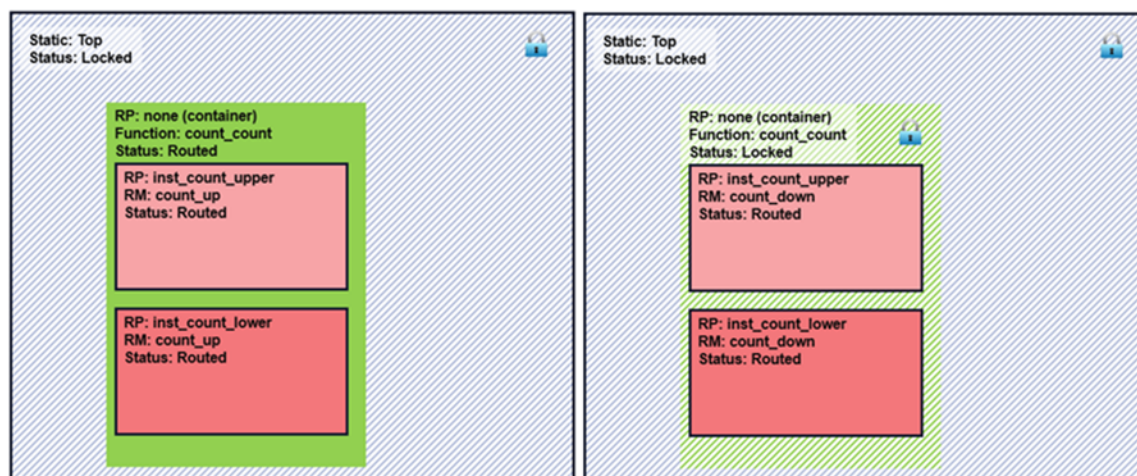
5. 二次リコンフィギャラブル パーティションにカウンター サブモジュールをインプリメントします。

```
source implement_sub_counters.tcl -notrace
```

シフター パスの場合と同様、標準 DFX フローを使用して、二次リコンフィギャラブル パーティションごとに 2 つのカウンター モジュール (count\_up、count\_down) を処理します。また、シフターと同様、再結合されたデザイン チェックポイントは、二次フローを通過する最初のパスから作成されます。

2 つの二次インプリメンテーション スクリプト (およびそれより前の分割スクリプト) は、2 つの固有の Vivado セッションで並行して実行できます。どちらも同じロック済みの最上位スタティック デザインに依存しますが、inst\_RP レベル以下は固有になります。このバージョンのフロアプランは二次 RP のものとは若干異なりますが、数も異なる場合があります。

図 112: count\_up をインプリメントしてから count\_down を 2 つの RP にインプリメントする 2 つの二次 run



## スタティック デザインのアップデート

標準 DFX デザイン フローと同様に、インプリメンテーションの結果はトップダウンのインコンテキストで作成されます。いずれかの時点でスタティックと見なされているデザインの一部をアップデートする必要がある場合、そのスタティックな部分以下のリコンフィギュラブル モジュールのすべての結果をインプリメントし直して、すべてが同期した状態を維持できるようにする必要があります。

たとえば、最上位のスタティックの部分に設計変更があった場合、既存の結果はすべて古いものと見なされ、すべてを再コンパイルする必要があります。一次 RM (reconfig\_shifters または reconfig\_counters) のいずれかにアップデートがあった場合、変更されたモジュールに依存するすべての結果を再コンパイルする必要があります。これらの個々のスクリプトは、必要に応じて結果をアップデートするために単独で呼び出すことができます。

## 検証パス

標準的な DFX デザイン フローと同様、Nested DFX デザイン イメージも pr\_verify を使用してチェックし、すべてのイメージが同期することを確認する必要があります。コア インプリメンテーション ツール (opt\_design など) と同様、pr\_verify はリコンフィギュラブルとマークされた現在のセルに基づいて実行されます。このことを念頭に置いて、現在のスタティック デザインを使用して、同一条件の比較を実行します。次のスクリプトを読み込んで、互換性のあるすべてのコンフィギュレーションを検証します。

```
source verify_configurations.tcl
```

このスクリプトは、3 つの対になった配線済みデザインを比較します。それぞれが、まったく同じになるはずのスタティック ロジックを使用して、チェックポイントの対比較を実行します。このセクションでは、実行される比較と、次のステップで作成される互換性のあるビットストリームについて説明します。

1. pr\_verify への 1 回目の呼び出しで 2 つの再結合されたチェックポイントが比較されます。これらはそれぞれ、単一のリコンフィギュラブル パーティション (inst\_RP) を使用しており、最上位のみが同じスタティック インプリメンテーション結果になるはずで、これらのチェックポイントは、それぞれが適切な二次パーシャル ビットストリームを受け取ることができる場合でも、ネストなしの標準 DFX デザインを表します。

ほかのチェックポイントが二次モジュール (shift\_left、count\_down など) で作成されてから再結合された場合は、pr\_verify を使用して比較して、それらの inst\_RP ビットストリームをこの互換性リストに追加できます。これは、分割された二次 RP がなくても、inst\_RP のほかのすべての RM で同様です。

2. pr\_verify への 2 回目の呼び出しでは、shift\_right と shift\_left の 2 番目のレベルのチェックポイントが比較されます。これらには上位および下位のサブモジュールにロックダウンされたスタティック 部分があるので、比較は階層の top レベルと reconfig\_shifters レベルのこのスタティック ロジック間で実行されます。
3. 2 回目と同様、pr\_verify の 3 回目の呼び出しでは、count\_up チェックポイントと count\_down の 2 番目のレベルのチェックポイントが比較されます。これらには、top および reconfig\_counters にロックされたスタティック 部分があるので、比較は上位および下位のリコンフィギュラブル パーティションまでのこのスタティック ロジック間で実行されます。

## ビットストリームの作成

Nested DFX 設計手法では、HD.RECONFIGURABLE プロパティを階層内で上下に移動します。インプリメンテーション ツールは、現在どのセルがリコンフィギュラブルとして定義されているかによって、標準の DFX 設計規則に従います。これは write\_bitstream にも当てはまります。パーシャル ビットストリームは、現在 HD.RECONFIGURABLE プロパティを保持しているセルに対してのみ作成されます。

Vivado で完全に配線済みのデザイン チェックポイントを使用する場合は、write\_bitstream でフルおよびパーシャル ビットストリームを生成します。デフォルトでは、このコマンドはデバイス全体の標準フル ビットストリームと、現在リコンフィギュラブル パーティションとして定義されている各セルのパーシャル ビットストリームを生成します。次の 2 つのオプションがあります。

- -cell オプションは、指定したセルのパーシャル ビットストリームのみを生成します。
- -no\_partial\_bitfile オプションは、標準的なフル デバイス ビットストリームのみを生成します。

次のスクリプトを実行して、インプリメンテーションされた既存のコンフィギュレーションのフルおよびパーシャル ビットストリームのコレクションを作成します。時間とスペースを節約するため、1 つのフル デバイス ビットストリームのみが作成されます。

```
source generate_all_bitstreams.tcl
```

このスクリプトは、各チェックポイントを 1 つずつ開き、特定のフルまたはパーシャル ビットストリームを書き出します。ビットストリームは互換性に基づいてサブフォルダーに分けられます。各ビットストリームは、-no\_partial\_bitfile オプション (次にリストする最初のビットストリーム) または -cell オプション (それ以外のすべてのビットストリーム) のいずれかを使用して作成されます。後者を使用する場合、パーシャル ビット ファイル名は任意の名前にできます。ファンクション、バージョン、および互換性を明確に示す名前を使用してください。次は、各フォルダーで生成される 11 個のビットストリームです。

- Bitstreams
  - top\_shift\_right\_right.bit
- Bitstreams/inst\_RP
  - inst\_RP\_shift\_right\_right\_recombined\_partial.bit
  - inst\_RP\_count\_up\_up\_recombined\_partial.bit
- Bitstreams/inst\_shift
  - shift\_right\_upper\_partial.bit
  - shift\_right\_lower\_partial.bit
  - shift\_left\_upper\_partial.bit
  - shift\_left\_lower\_partial.bit
- Bitstreams/inst\_count
  - count\_up\_upper\_partial.bit
  - count\_up\_lower\_partial.bit
  - count\_down\_upper\_partial.bit
  - count\_down\_lower\_partial.bit

これらに加え、UltraScale の場合は、上記にリストされるパーシャル ビット ファイルごとに 1 つずつ、クリア ビット ファイルも生成されます。ベース名は同じになりますが、ファイル名の最後に \_clear が付きます。これらの使用方法については、この演習の次のセクションで説明します。

また、グレー ボックス コンフィギュレーション用のパーシャル ビットストリームを作成、インプリメント、および生成するためのコマンドについても説明します。これらはソリューションには必要ありませんが、特定のリコンフィギュラブル パーティション内でアクティビティをオフにするために使用できます。グレー パラメーターを true に設定してから、generate\_all\_bitstreams.tcl スクリプトを読み込んで、これらのオプションのパーシャル ビットストリームを作成します。

グレー ボックスのパーシャル (およびクリア) ビットストリームは、二次 RP それぞれ (合計 4 つ) および一次 RP (inst\_RP) に対して作成され、二次 RP のグレー ボックスを含む各 inst\_RP RM インスタンス (reconfig\_shifters、reconfig\_counters) ごとにも作成されます。

- Bitstreams/inst\_RP
  - inst\_RP\_grey\_partial.bit
  - reconfig\_shifters\_grey\_grey\_partial.bit
  - reconfig\_counters\_grey\_grey\_partial.bit
- Bitstreams/inst\_shift
  - shift\_upper\_grey\_partial.bit
  - shift\_lower\_grey\_partial.bit
- Bitstreams/inst\_count
  - count\_upper\_grey\_partial.bit
  - count\_lower\_grey\_partial.bit

initial\_top\_route\_design.dcp チェックポイントから作成されるビットストリームはありません。これは必要がないからです。このデザインの最上位のスタティック イメージはほかのすべてのデザインと同じで、shift\_right-shift\_right ファンクションは最初に分割された run と論理的に同じであるためです。後者のインプリメンテーション結果は、新しい RP が導入されたために異なりますが、分割前から shift\_right-shift\_right パーシャル イメージを読み込む場合は、二次シフターを個別にスワップ アウトできませんでした。

この演習では、フル デバイス ビットストリームは、デザインのこの shift\_right-shift\_right バージョンに対してのみ作成されますが、一次および二次リコンフィギュラブル モジュールの有効な組み合わせに対しては、フル デバイス ビットストリームを生成できます。これは、デバイスの初期動作をどのようにするかによって異なります。count\_up-count\_down バージョン、または二次 RP ごとにグレー ボックスが付いたシフター バージョンを作成できます。これには、配線済みモジュールのチェックポイントをロックされた最上位スタティックにリンクしてから、write\_bitstream を呼び出します。

つまり、pr\_subdivide ファンクションを使用して相対的なスタティック層をそれぞれロックし、トップダウンでデザイン結果を作成します。そのあと、上位レベルのリコンフィギュラブル パーティションに戻るため、pr\_recombine を使用して、そのレベルでパーシャル ビットストリームを生成するチェックポイントを作成します。

## 手順 5: ハードウェアでのデザインのテスト

作成したフル ビットストリームおよびパーシャル ビットストリームのセットを使用し、4 つのデモ ボードのいずれかでデザインをテストできます。現在のデザインでは、KCU105、VCU108、KCU116、VCU118 ボード、リビジョンは 1.0 およびそれ以降のものがサポートされています。



## フル イメージを使用したデバイスのコンフィギュレーション

1. プラットフォーム ケーブル USB を使用してコンピューターにボードを接続し、ボードに電源を投入します。
2. Vivado IDE で [Flow] → [Open Hardware Manager] をクリックします。
3. 緑色のバナー上の [Open target] をクリックします。ボードとの通信を確立するため、ウィザードの手順に従います。
4. [Xilinx device] (たとえば、xcku040\_0) を右クリックし、[Program Device] をクリックします。
5. [bitstreams] フォルダを参照し、`top_shift_right_right.bit` を選択してから、[Program] をクリックしてデバイスをプログラムします。

これで 2 つのタスクを実行している GPIO LED のバンクを確認できるはずです。4 つの LED の 2 セットが右にシフトしています。フル デバイスのコンフィギュレーションにかかる時間に注目してください。

現在動作しているデバイスには、最上位 (スタティック)、一次 RM `reconfig_shifters`、二次 RM `shift_right` (上) および `shift_right` (下) を含んでいます。

## デバイスのパーシャル リコンフィギュレーション

現在読み込まれているデザインと互換性がある場合にのみ、これまでに作成したパーシャル ビットストリームのいずれかを使用して、デバイスをパーシャル リコンフィギュレーションする準備が整いました。UltraScale デバイスの場合は、常に該当するクリア ビットストリームでまず作業を開始します。このため、次のセクションはデバイス ファミリー別に分かれています (UltraScale+ の手順は簡単であるため)。次の手順は、わかりやすくするためにファミリー別に分かれています。

### UltraScale+ デバイスのパーシャル ビットストリームの読み込み

VCU118 または KCU116 をターゲットにする場合は、次の手順に従います。このセクションでは、UltraScale デバイスの手順について詳しく説明します

まず、「上部」のシフター位置をリコンフィギュレーションします。

1. 緑色のバナー上の [Program device] をクリックします (またはターゲット デバイスを右クリックして [Program device] をクリックします)。Bitstreams/inst\_shift フォルダで `shift_left_upper_partial.bit` を選択し、[Program] をクリックしてデバイスをプログラムします。上部シフト部分は左にシフトするようになりましたが、下部はまだ右にシフトしています。Done も High (オン) を返しました。

カウンター ファンクションに移行するには、一次 `reconfig_counters` RM をまず読み込む必要があります。この時点で二次 `count_up` または `count_down` パーシャル ビットストリームを読み込むと、これらのファンクションが最上位のスタティック デザインに接続しないため、正しく機能しません。

2. 緑色のバナー上の [Program device] をもう一度クリックします。Bitstreams/inst\_RP フォルダで `inst_RP_count_up_recombined_partial.bit` を選択し、[Program] をクリックしてデバイスをプログラムします。LED の 2 つのセクションがカウント アップされるようになりました。

`reconfig_counters` 一次ファンクションができたので、その階層の下にあるリコンフィギュラブル パーティションを部分的にリコンフィギュレーションできるようになりました。

3. 緑色のバナーの [Program device] を最後にもう一度選択します。Bitstreams/inst\_count フォルダで `count_down_lower_partial.bit` を選択し、[OK] をクリックしてデバイスをプログラムします。上部シフト部分はカウント アップしたままですが、下部はカウント ダウンするようになりました。

これで UltraScale+ デバイスの演習を終了します。

## UltraScale デバイスのパーシャル ビットストリームの読み込み

VCU108 または KCU105 をターゲットにする場合は、次の手順に従います。まず、「上部」のシフター位置をリコンフィギュレーションします。

1. 緑色のバナー上の [Program device] をクリックします (またはターゲット デバイスを右クリックして [Program device] をクリックします)。Bitstreams/inst\_shift フォルダーで shift\_right\_upper\_partial\_clear.bit を選択し、[Program] をクリックしてデバイスをプログラムします。LED の上部のシフト部分が停止しますが、下部のシフト部分はリコンフィギュレーションの影響を受けず、シフトし続けます。今回のコンフィギュレーションには前回ほど時間がかからず、DONE LED がオフになった点に注目してください。
2. 緑色のバナー上の [Program device] をもう一度クリックします。Bitstreams/inst\_shift フォルダーで shift\_left\_upper\_partial.bit を選択し、[Program] をクリックしてデバイスをプログラムします。上部シフト部分は左にシフトするようになりましたが、下部はまだ右にシフトしています。Done も High (オン) を返しました。

カウンター ファンクションに移行するには、一次 reconfig\_counters RM をまず読み込む必要があります。この時点で二次 count\_up または count\_down パーシャル ビットストリームを読み込むと、これらのファンクションが最上位のスタティック デザインに接続しないため、正しく機能しません。



**重要:** さらに、UltraScale デバイスでは、新しい一次パーシャル ビットストリームが配信できるようになる前に、クリア ビットストリームをボトムアップで適用する必要があります。各クリア ビットストリームは、階層のそのレベルで現在読み込まれているファンクションと一致する必要があります。二次クリア ビットストリームの順序は重要ではありませんが、一次クリア ビットストリームよりも前にする必要があります。

3. 緑色のバナー上の [Program device] をクリックし、これらのクリア ビットストリームを使用してデバイスを一度にプログラムします。

- inst\_shift/shift\_left\_upper\_partial\_clear.bit
- inst\_shift/shift\_right\_lower\_partial\_clear.bit
- inst\_RP/inst\_RP\_shift\_right\_right\_recombined\_partial\_clear.bit

これらの実行後、シフター機能と最上位スタティックへの接続がアクティブ デザインから削除されたため、すべての LED アクティビティが停止しました。

4. 緑色のバナー上の [Program device] をもう一度クリックします。Bitstreams/inst\_RP フォルダーで inst\_RP\_count\_up\_up\_recombined\_partial.bit を選択し、[Program] をクリックしてデバイスをプログラムします。LED の 2 つのセクションがカウント アップされるようになりました。

reconfig\_counters 一次ファンクションができたので、その階層の下にあるリコンフィギュラブル パーティションを部分的にリコンフィギュレーションできるようになりました。

5. 緑色のバナー上の [Program device] をクリックします。Bitstreams/inst\_count フォルダーで count\_up\_lower\_partial\_clear.bit を選択し、[Program] をクリックしてデバイスをプログラムします。これにより、LED の下部セットのカウント ファンクションを駆動しているカウンターが停止します。
6. 緑色のバナーの [Program device] を最後にもう一度選択します。Bitstreams フォルダーで count\_down\_lower\_partial.bit を選択し、[OK] をクリックしてデバイスをプログラムします。上部シフト部分はカウント アップしたままですが、下部はカウント ダウンするようになりました。

これで UltraScale デバイスの演習を終了します。

## まとめ

これで演習 8 は終了です。この演習では、次の作業を実行しました。

- Nested Dynamic Function eXchange のインプリメンテーションを準備するため、ボトムアップでデザインを合成しました。
- リCONFIGURABLE パーティションの入れ子レベルを作成するために、pr\_subdivide と pr\_recombine を適用しました。
- スクリプトを介して複数のコンフィギュレーションをインプリメントしました。
- スタティック デザインを一貫させるために、チェックポイントをペアで比較しました。
- フレーム セットを確認し、この 2 つのコンフィギュレーションを検証しました。
- 一次および二次パースシャル イメージを使用し、FPGA をコンフィギュレーションしてから、部分的にリCONFIGURATIONしました。

# 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。

---

## 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<sup>®</sup> 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。

---

## その他のリソース

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: Dynamic Function eXchange』 ([UG909](#))
- 資料のナビゲーションには、Dynamic Function eXchange (DFX) に関する資料など PR 関連のリソースへのリンクを集めた DFX デザイン ハブがあります。ハブには、[ザイリンクス サポート サイト](#)からもアクセスできます。
- 『Dynamic Function eXchange Controller IP LogiCORE IP 製品ガイド』 ([PG374](#))



- 『Dynamic Function eXchange Decoupler IP LogiCORE IP 製品ガイド』 (PG375)
- 『Dynamic Function eXchange Bitstream Monitor IP LogiCORE IP 製品ガイド』 (PG376)
- 『Dynamic Function eXchange Shutdown Manager IP LogiCORE IP 製品ガイド』 (PG377)

## Please Read: Important Legal Notices

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。

### AUTOMOTIVE APPLICATIONS DISCLAIMER

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、

### Copyright

© Copyright 2012-2020 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。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。そのほかすべての商標は、それぞれの所有者に帰属します。

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](mailto: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。