

Vivado Design Suite ユーザー ガイド

プログラムおよびデバッグ

UG908 (v2020.1) 2020 年 6 月 3 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 6 月 3 日 バージョン 2020.1	
資料全体	Vivado 2020.1 リリース用にアップデート。

目次

改訂履歴.....	2
第 1 章: 概要.....	8
入門.....	8
デバッグ用語集.....	9
第 2 章: Vivado Lab Edition.....	12
インストール.....	12
Vivado Lab Edition の使用.....	13
Vivado Lab Edition プロジェクト.....	15
プログラム機能.....	18
デバッグ機能.....	18
第 3 章: ビットストリームまたはデバイス イメージの生成.....	19
ビットストリーム ファイルのフォーマット設定の変更.....	20
デバイス コンフィギュレーション ビットストリーム設定の変更.....	20
第 4 章: デバイスのプログラム.....	22
ハードウェア マネージャーを開く.....	22
ハードウェア ターゲット接続を開く.....	22
hw_server を使用したハードウェア ターゲットへの接続.....	23
新しいハードウェア ターゲットを開く.....	24
ハードウェア ターゲットのトラブルシューティング.....	26
プログラム ファイルをハードウェア デバイスに関連付け.....	27
ハードウェア デバイスのプログラム.....	27
ハードウェア ターゲットを閉じる.....	31
ハードウェア サーバーへの接続を閉じる.....	31
低い JTAG クロック周波数でのターゲット デバイスへの接続.....	32
JTAG チェーンに 32 個を超えるデバイスを含むサーバーへの接続.....	33
第 5 章: Vivado でのリモート デバッグ.....	34
イーサネットから Vivado ハードウェア サーバーを使用したデバッグ.....	34
ザイリンクス仮想ケーブル (XVC).....	34
第 6 章: コンフィギュレーション メモリ デバイスのプログラム.....	44
コンフィギュレーション メモリ デバイスで使用するビットストリームの生成.....	44
コンフィギュレーション メモリ ファイルの作成.....	45
SPI デュアル クワッド (x8) デバイス用のコンフィギュレーション メモリの作成.....	47

Vivado でハードウェア ターゲットに接続.....	48
コンフィギュレーション メモリ デバイスの追加.....	48
コンフィギュレーション メモリ デバイスのプログラム.....	50
デバイスの起動.....	52
マスター モードのコンフィギュレーション エラー.....	53
第 7 章: アドバンス プログラム機能.....	54
リードバックおよび検証.....	54
7 シリーズ デバイス用の暗号化および認証されたファイルの生成.....	57
UltraScale および UltraScale+ デバイス用の暗号化および認証されたファイルの生成.....	60
7 シリーズ デバイスの AES キーのプログラム.....	65
UltraScale および UltraScale+ デバイスの AES キーのプログラム.....	66
eFUSE レジスタのアクセスおよびプログラム.....	69
eFUSE プログラミングのケーブル サポート.....	69
7 シリーズ デバイスの eFUSE レジスタのアクセスおよびプログラム.....	70
UltraScale および UltraScale+ デバイスの eFUSE レジスタのアクセスおよびプログラム.....	75
eFUSE NKZ ファイル.....	82
システム モニター.....	83
第 8 章: SVF (Serial Vector Format) ファイルを使用したプログラム.....	85
SVF ターゲットの作成.....	85
SVF ターゲットへのデバイスの追加.....	88
ザイリンクス デバイスへのコンフィギュレーション メモリ パーツの追加.....	93
SVF チェーンでの操作.....	95
SVF ファイルの保存.....	97
SVF ファイルの実行.....	100
第 9 章: デザインのデバッグ.....	101
RTL レベルのデザイン シミュレーション.....	101
インプリメンテーション後のデザイン シミュレーション.....	101
インシステム ロジック デザインのデバッグ.....	101
インシステム シリアル I/O デザインのデバッグ.....	102
第 10 章: インシステム ロジック デザインのデバッグ フロー.....	103
インシステム デバッグ用のデザインのプローブ.....	103
ネットリスト挿入デバッグ プローブ フロー.....	104
HDL インスタンス化 デバッグ プローブ フローの概要.....	118
HDL インスタンス化 デバッグ プローブ フロー.....	118
IP インテグレーターでのデバッグ フロー.....	128
デバッグ コアを含むデザインのインプリメンテーション.....	131
ILA コアとタイミングに関する考慮事項.....	131
デバッグ コアのクロッキング ガイドライン.....	132
パーシャル リコンフィギュレーション デザインへの Vivado デバッグ コアの追加.....	135
第 11 章: ハードウェアでのロジック デザインのデバッグ.....	136
Vivado ロジック解析を使用したデザインのデバッグ.....	136

ハードウェア ターゲットに接続してデバイスをプログラム.....	136
Vivado ハードウェア マネージャーのダッシュボード	137
計測のための ILA コアの設定.....	147
ILA プローブ情報の記述.....	173
ILA プローブ情報の読み出し.....	173
ILA コアからのデータを波形ビューアーで表示.....	174
波形 ILA トリガーおよびエクスポート機能の使用.....	174
ILA コアでキャプチャされたデータの保存および復元.....	176
プローブ値の列挙名.....	178
ハードウェア マネージャーでの AXI インターフェイスのデバッグ	185
計測のための VIO コアの設定.....	194
VIO コアのスレータスの表示.....	195
VIO コアの出力プローブ.....	199
JTAG-to-AXI Master デバッグ コアを使用したハードウェア システム通信.....	201
ラボ環境での Vivado ロジック解析の使用.....	204
ハードウェア マネージャーの Tcl オブジェクトおよびコマンド.....	205
Tcl コマンドを使用した JTAG-to-AXI Master コアへのアクセス.....	208
ILA を測定する Tcl コマンドの使用.....	209
スタートアップ時のトリガー.....	210
メモリ キャリブレーションのデバッグ.....	211
Vivado ハードウェア マネージャーでのパシヤル リコンフィギャラブル デザインのデバッグ.....	212
HBM (High Bandwidth Memory) モニター.....	212
第 12 章: 波形ビューアーを使用した ILA プローブ データの表示.....	214
ILA データと波形の関係.....	214
波形ビューアーのレイアウト	215
波形ビューアーの操作.....	216
波形からのプローブの削除.....	216
波形へのプローブの追加.....	217
波形 ILA トリガーおよびエクスポート機能の使用.....	217
ズーム機能の使用.....	219
波形設定.....	219
波形コンフィギュレーションのカスタマイズ	220
オブジェクト名の変更.....	223
バスの基数.....	225
アナログ波形の表示.....	225
バス プロット ビューアー.....	227
ズーム機能.....	230
第 13 章: インプリメンテーション後のデザインのデバッグ.....	232
Vivado ECO フローを使用した既存のデバッグ プローブの置き換え.....	232
配置配線済みデザイン チェックポイントのデバッグ プローブの置換.....	233
Vivado ECO Tcl フローを使用した既存のデバッグ プローブの置き換え.....	238
デバッグ コア (ILA) を変更した場合のインクリメンタル コンパイル.....	238
第 14 章: シリアル I/O ハードウェア デバッグ フロー.....	242
シリアル I/O ハードウェア デバッグ フロー.....	242

第 15 章: ハードウェアでのシリアル I/O デザインのデバッグ	246
Vivado シリアル I/O 解析を使用したデザインのデバッグ	246
スライサー アイ、ヒストグラム、および信号対ノイズ比の表示 (GTM トランシーバーのみ)	260
付録 A: デバイス コンフィギュレーション ビットストリーム設定	262
7 シリーズ デバイスのビットストリーム設定	262
Zynq-7000 のビットストリーム設定	267
UltraScale のビットストリーム設定	272
Virtex および Kintex UltraScale+ のビットストリーム設定	278
Zynq UltraScale+ MPSoC のビットストリーム設定	284
付録 B: トリガー ステート マシンの言語記述	286
ステート	286
goto アクション	286
条件分岐	286
カウンタ	287
フラグ	288
条件文	288
付録 C: 下位 SVF JTAG コマンド	293
HDR (Header Data Register)、HIR (Header Instruction Register)	293
TDR (Trailer Data Register)、TIR (Trailer Instruction Register)	294
scan_ir_hw	295
scan_dr_hw	295
マルチチェーン SVF 操作	296
付録 D: hw_server でサポートされる JTAG ケーブルおよびデバイス	300
付録 E: コンフィギュレーション メモリのサポート	301
Artix-7 コンフィギュレーションのメモリ デバイス	301
Kintex-7 コンフィギュレーションのメモリ デバイス	305
Spartan-7 コンフィギュレーションのメモリ デバイス	308
Virtex-7 コンフィギュレーションのメモリ デバイス	310
Kintex UltraScale コンフィギュレーションのメモリ デバイス	313
Kintex UltraScale+ コンフィギュレーションのメモリ デバイス	317
Virtex UltraScale コンフィギュレーション メモリ デバイス	320
Virtex UltraScale+ コンフィギュレーション メモリ デバイス	323
Zynq-7000 コンフィギュレーション メモリ デバイス	326
Zynq UltraScale+ MPSoC コンフィギュレーション メモリ デバイス	344
Zynq UltraScale+ RFSoC コンフィギュレーション メモリ デバイス	350
付録 F: hw_server のコマンド ライン オプション	359
標準 hw_server オプション	359
アドバンス オプション	360

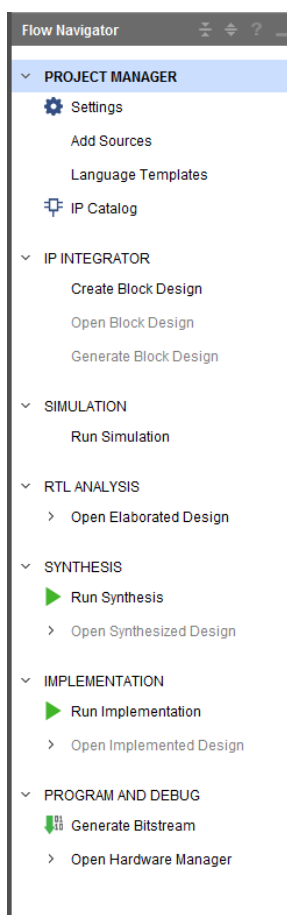
付録 G: その他のリソースおよび法的通知.....	366
ザイリンクス リソース.....	366
ソリューション センター.....	366
Documentation Navigator およびデザイン ハブ.....	366
参考資料.....	367
トレーニング コース.....	368
お読みください: 重要な法的通知.....	368

概要

入門

デザインをインプリメントしたら、FPGA をプログラムし、デザインをインシステムでデバッグし、ハードウェアでデザインを実行します。FPGA デバイスのプログラムおよびインシステム デバッグの実行に必要なコマンドはすべて、Vivado® 統合設計環境 (IDE) の [Flow Navigator] の [Program and Debug] の下にあります。

図 1: Flow Navigator の [PROGRAM AND DEBUG]



デバッグ用語集

ILA

ILA (Integrated Logic Analyzer) 機能を使用すると、FPGA デバイスのインプリメント後のデザインをインシステムでデバッグできます。この機能は、デザインで信号を監視する必要がある場合に使用します。ハードウェア イベントをトリガーし、データをシステム速度でキャプチャするためにも使用できます。

ILA コアは、RTL コードにインスタンス化するか、Vivado デザイン フローの合成後に挿入します。ILA コアの詳細および Vivado® Design Suite での使用方法は、第 10 章「インシステム ロジック デザイン」および第 11 章「ハードウェアでのロジック デザインのデバッグ」を参照してください。ILA コア IP の詳細は、『Integrated Logic Analyzer LogiCORE IP 製品ガイド』(PG172) を参照してください。

関連情報

[インシステム ロジック デザインのデバッグ フロー](#)
[ハードウェアでのロジック デザインのデバッグ](#)

VIO

VIO (Virtual Input/Output) は、内部 FPGA 信号をリアルタイムに監視および駆動できるデバッグ機能です。ターゲット ハードウェアへの物理的なアクセスがない場合は、このデバッグ機能を使用して、実際のハードウェアにある信号を駆動および監視できます。

このデバッグ コアは RTL コードにインスタンス化するので、どのネットを駆動するのか前もって決めておく必要があります。このコアは IP カタログの Debug カテゴリに含まれます。VIO コアの詳細およびその Vivado Design Suite での使用方法は、第 11 章「ハードウェアでのロジック デザインのデバッグ」を参照してください。VIO コア IP の詳細は、『Virtual Input/Output LogiCORE IP 製品ガイド』(PG159) を参照してください。

関連情報

[ハードウェアでのロジック デザインのデバッグ](#)

IBERT

IBERT (Integrated Bit Error Ratio Tester) Serial Analyzer デザインを使用すると、インシステム シリアル I/O の検証およびデバッグが可能になります。この機能を使用すると、FPGA ベース システムの高速シリアル I/O リンクを計測および最適化できます。単純なクロックや接続の問題から複雑なマージン解析およびチャネル最適化の問題まで、さまざまなインシステム デバッグおよび検証の問題を解決するには、IBERT Serial Analyzer デザインを使用することをお勧めします。

IBERT Serial Analyzer デザインは、受信信号にレシーバー イコライゼーションが適用された後の信号の質を計測するためにも使用できます。これにより、TX から RX へのチャネルの最適なポイント、つまり実際の正しいデータが計測されます。このデザインには、IP カタログから IBERT コアを選択、設定、生成して、このコアの [Open Example Design] を選択するとアクセスできます。IBERT コアの詳細および Vivado Design Suite での使用方法は、第 14 章「シリアル I/O ハードウェア デバッグ フロー」および第 15 章「ハードウェアでのシリアル I/O デザインのデバッグ」を参照してください。IBERT デザインの詳細は、『Integrated Bit Error Ratio Tester 7 Series GTX Transceivers LogiCORE IP 製品ガイド』(PG132)、『Integrated Bit Error Ratio Tester 7 Series GTP Transceivers LogiCORE IP 製品ガイド』(PG133)、および『Integrated Bit Error Ratio Tester 7 Series GTH Transceivers LogiCORE IP 製品ガイド』(PG152) を参照してください。

関連情報

[ハードウェアでのシリアル I/O デザインのデバッグ](#)
[シリアル I/O ハードウェア デバッグ フロー](#)

JTAG-to-AXI Master

JTAG-to-AXI Master デバッグ機能は、ハードウェアで動作中のさまざまな AXI フルおよび AXI Lite スレーブ コアと通信する AXI トランザクションを生成するために使用されます。ザイリンクスでは、AXI トランザクションを生成し、ランタイムに FPGA 内部で AXI 信号をデバッグおよび駆動するためにこのコアを使用することをお勧めします。このコアは、プロセッサのないデザインでも使用できます。

このコアは IP カタログの Debug カテゴリに含まれます。JTAG-to-AXI Master コアの詳細およびその Vivado Design Suite での使用方法は、第 11 章「ハードウェアでのロジック デザインのデバッグ」を参照してください。JTAG-to-AXI IP コアの詳細は、『JTAG to AXI Master LogiCORE IP 製品ガイド』(PG174)を参照してください。

関連情報

[ハードウェアでのロジック デザインのデバッグ](#)

Debug Hub

Vivado デバッグ ハブは、FPGA デバイスの JTAG バウンダリスキャン (BSCAN) インターフェイスと、次のタイプのコアを含む Vivado デバッグ コアとの間のインターフェイスを提供します。

- ILA (Integrated Logic Analyzer)
- VIO (Virtual Input/Output)
- IBERT (Integrated Bit Error Ratio Test)
- JTAG-to-AXI
- メモリ IP



重要: Vivado デバッグ ハブ コアをデザインにインスタンス化することはできません。これは `opt_design` の段階で Vivado により挿入されます。

System ILA

System ILA (System Integrated Logic Analyzer) 機能を使用すると、FPGA デバイスのインプリメント後のデザインをインシステムでデバッグできます。IP インテグレーター ブロック デザインでインターフェイスおよび信号を監視する必要がある場合は、この IP を使用します。インターフェイスおよび信号関連のハードウェア イベントをトリガーし、データをシステム速度でキャプチャするためにも使用できます。FPGA デバイス上でデザインをデバッグするときに、ハードウェア マネージャーでインターフェイス イベントをわかりやすく表示できます。この IP には、AXI インターフェイスのデバッグおよび監視機能と、AXI4-MM および AXI4-Stream プロトコル チェックが含まれます。

System ILA コアは監視しているデザインに同期するので、そのデザインに適用されるデザイン クロック制約はすべて、System ILA コアのコンポーネントにも適用されます。System ILA コア IP の詳細は、『System Integrated Logic Analyzer LogiCORE IP 製品ガイド』(PG261)を参照してください。

Debug Bridge

Debug Bridge IP コアは、デザイン内のデバッグ コアと通信するためのオプションを複数提供するコントローラーです。

Debug Bridge は、XVC (ザイリンクス仮想ケーブル) を使用して、JTAG ケーブルを使わずに、イーサネットなどのインターフェイスを介してデザインをリモートでデバッグするためのものです。

また、Dynamic Function eXchange およびフィールド アップデートを利用した Tandem デザインのデバッグにもよく使用されます。フィールド アップデートを利用した Tandem デザインのデバッグおよび Debug Bridge の詳細は、『UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP 製品ガイド』 (PG213: [英語版](#)、[日本語版](#)) を参照してください。

Debug Bridge は、JTAG が推奨される通信およびデバッグ メカニズムではないシステムでも PCIe® コアと一緒に使用できます。PCIe コアと Debug Bridge 使用した XVC フローの詳細は、『UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP 製品ガイド』 (PG213: [英語版](#)、[日本語版](#)) を参照してください。

Debug Bridge コア IP の詳細は、『Debug Bridge LogiCORE IP 製品ガイド』 ([PG245](#)) を参照してください。

In-System IBERT

注記: In-System IBERT は UltraScale および UltraScale+ でのみサポートされます。

In-System IBERT IP は、デザインの UltraScale および UltraScale+ トランシーバーの 2D のアイスキャンを Vivado Serial IO Analyzer を使用して実行します。IP はデザインからのデータを使用し、リアル タイムでトランシーバーのアイスキャンを出力しつつ、システムのほかの部分とも通信します。この IP は、デザインのユーザー ロジック、または GT ウィザードや Aurora などザイリンクス トランシーバー ベースの IP と通信できます。

In-System IBERT IP の詳細は、『In-System IBERT LogiCORE IP 製品ガイド』 ([PG246](#)) を参照してください。

IBERT GTR

IBERT UltraScale+ GTR は Zynq UltraScale+ MPSoC デバイスの GTR トランシーバーを評価および監視するために使用できます。この機能を使用すると、次のタスクを実行できます。

- ユーザー データを使用したアイ スキャンを実行
- GTR 設定を変更
- リンク ステータスを表示
- すべての GTR レーンで使用されるすべての PLL のロック ステータスをチェック

ただし、IBERT GTR には次の機能がありません。

- 生の PRBS データ パターンを使用したアイ スキャン
- ビット エラー率 (ビットまたはエラー カウンターなし) の測定

このソリューションはソフトウェア ベースなので、デバイスのプログラマブル ロジックに IP またはロジックが必要ありません。

Vivado Lab Edition

Vivado[®] Lab Edition はフル バージョンの Vivado Design Suite をスタンドアロンにインストールできるエディションで、ザイリンクス FPGA のビットストリームを生成した後に、FPGA をプログラムおよびデバッグするのに必要なすべての機能を備えています。マシンのディスク容量、メモリ、コネクティビティに関してマシン リソースが少ないラボ環境でのプログラムおよびデバッグに使用するのが一般的です。Vivado Lab Edition はインストール パッケージ サイズが 1 GB、インストール後にハードディスクを占める容量は約 2.4 GB と軽量です。

インストール

Vivado Lab Edition をインストールするには、Lab Edition インストーラーが必要です。

詳細なインストール、ライセンスおよびリリース情報については、『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。

Windows での Vivado Lab Edition の起動

Vivado Lab Edition を起動するには、次の順にクリックしてください。

[Start] → [All Programs] → [Xilinx Design Tools] → [Vivado Lab 2020.x] → [Vivado Lab 2020.x]

Windows または Linux のコマンド ラインからの Vivado Lab Edition の起動

コマンド プロンプトに次のコマンドを入力します。

```
vivado_lab
```



ヒント: コマンド プロンプトで vivado_lab を実行するには、使用する OS に応じて、次のいずれかのスクリプトを使用して環境を設定します。

```
C:\Xilinx\Vivado_Lab\2020.x\settings32.(bat|sh)
```

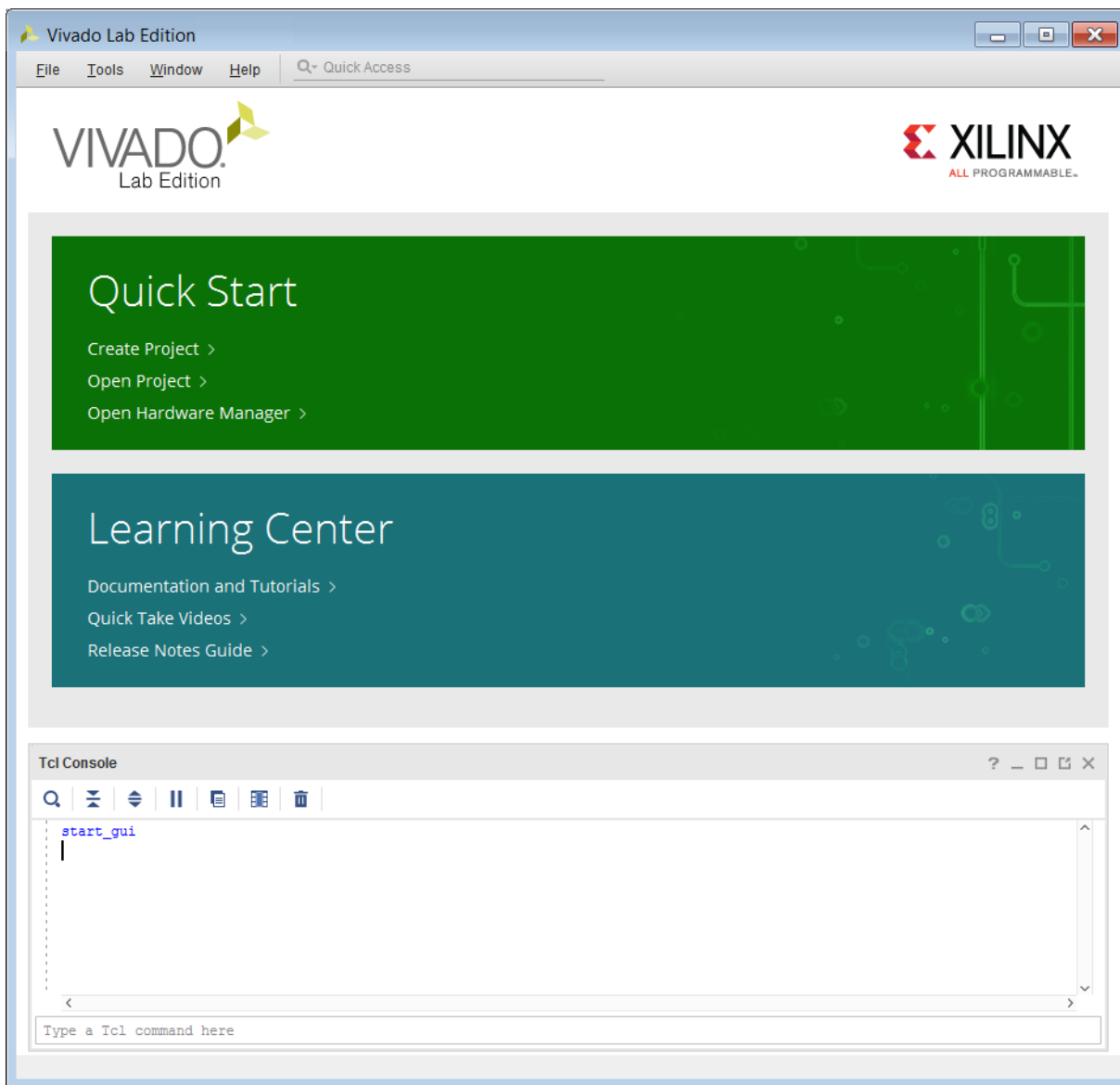
```
C:\Xilinx\Vivado_Lab\2020.x\settings64.(bat|sh)
```

Vivado Lab Edition はどのディレクトリからでも起動できますが、ログ ファイルおよびジャーナル ファイルが起動ディレクトリに書き込まれるので、書き込み権限のあるプロジェクト ディレクトリから実行することをお勧めします。コマンド プロンプトから実行する場合、プロジェクト ディレクトリから Vivado IDE を起動するか、`vivado_lab -log` および `-journal` オプションを使用してディレクトリを指定します。Windows ショートカットを使用する場合は、ショートカットを右クリックして [プロパティ] をクリックし、[作業フォルダー] を変更する必要があります。書き込み権限のないプロジェクト ディレクトリから起動すると、ツールから警告メッセージが表示されたり、予期しない動作が見られます。

Vivado Lab Edition の使用

Vivado Lab Edition を起動すると、Vivado Lab Edition を使用し始めるのに役立つさまざまなリンクを含む最初の画面が表示されます。

図 2: Vivado Lab Edition の最初の画面



プロジェクトの開始

デザインをプログラムまたはデバッグするには、プロジェクトを作成するか開き、ターゲット サーバーおよびデバイスに接続します。Getting Started ページの [Quick Start] セクションには、次のタスクを実行するためのボタンが表示されます。

- プロジェクトを作成。
- 既存のプロジェクトを開く。

注記: [Recent Projects] リストから最近開いたプロジェクトを開くこともできます。

ハードウェア マネージャーを開く

Vivado Design Suite ハードウェア マネージャーを開くと、デザイン ビットストリームをデバイスにダウンロードできます。Vivado ロジック解析機能および Vivado シリアル I/O 解析機能を使用すると、デザインをデバッグできます。たとえば、ILA、VIO および JTAG-to-AXI コアをデザインに追加して Vivado ロジック解析機能でデバッグしたり、ザイリンクス IP カタログから IBERT サンプル デザインを使用し、Vivado シリアル I/O 解析機能で GT をテストおよび設定したりできます。

資料およびビデオ

Getting Started ページからザイリンクス Documentation Navigator を使用して、ユーザー ガイド、チュートリアル、ビデオ、リリース ノートなどの資料にアクセスできます。

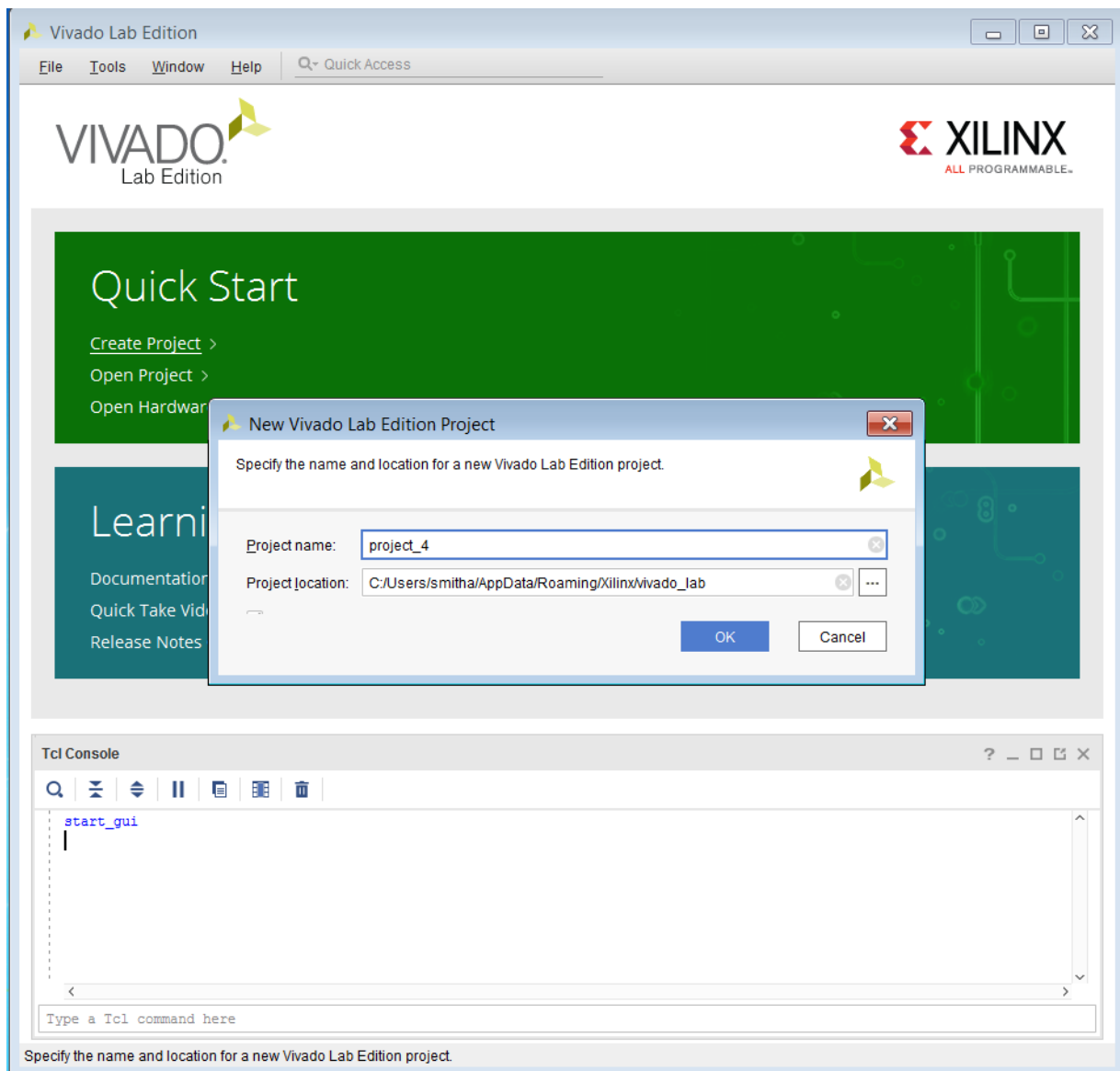
Vivado Lab Edition プロジェクト

Vivado Lab Edition を使用すると、ラボ環境でプロジェクトを作成できます。プログラムおよびランタイム デバッグに関するすべてのプリファレンスや基本設定はプロジェクトに保存されます。プロジェクトを開き直すと、設定およびプリファレンスが復元されます。Vivado Lab Edition プロジェクトは、Vivado Lab Edition ツールおよび Vivado Design Suite の両方で作成できます。

新規プロジェクトの作成

Vivado Lab Edition で新しくプロジェクトを作成するには、Getting Started ページで [Create New Project] をクリックします。[New Vivado Lab Edition Project] ダイアログ ボックスにプロジェクト名およびディレクトリを入力します。新しいプロジェクトを作成すると、Vivado Lab Edition によりプロジェクト ファイルが作成されます。このプロジェクト ファイルの名前は、[New Vivado Lab Edition Project] ダイアログ ボックスで入力したプロジェクト名に `.lpr` という拡張子が付いたものになります。次の図を参照してください。

図 3: Vivado Lab Edition での新規プロジェクトの作成



Tcl コマンドを使用したプロジェクトの作成

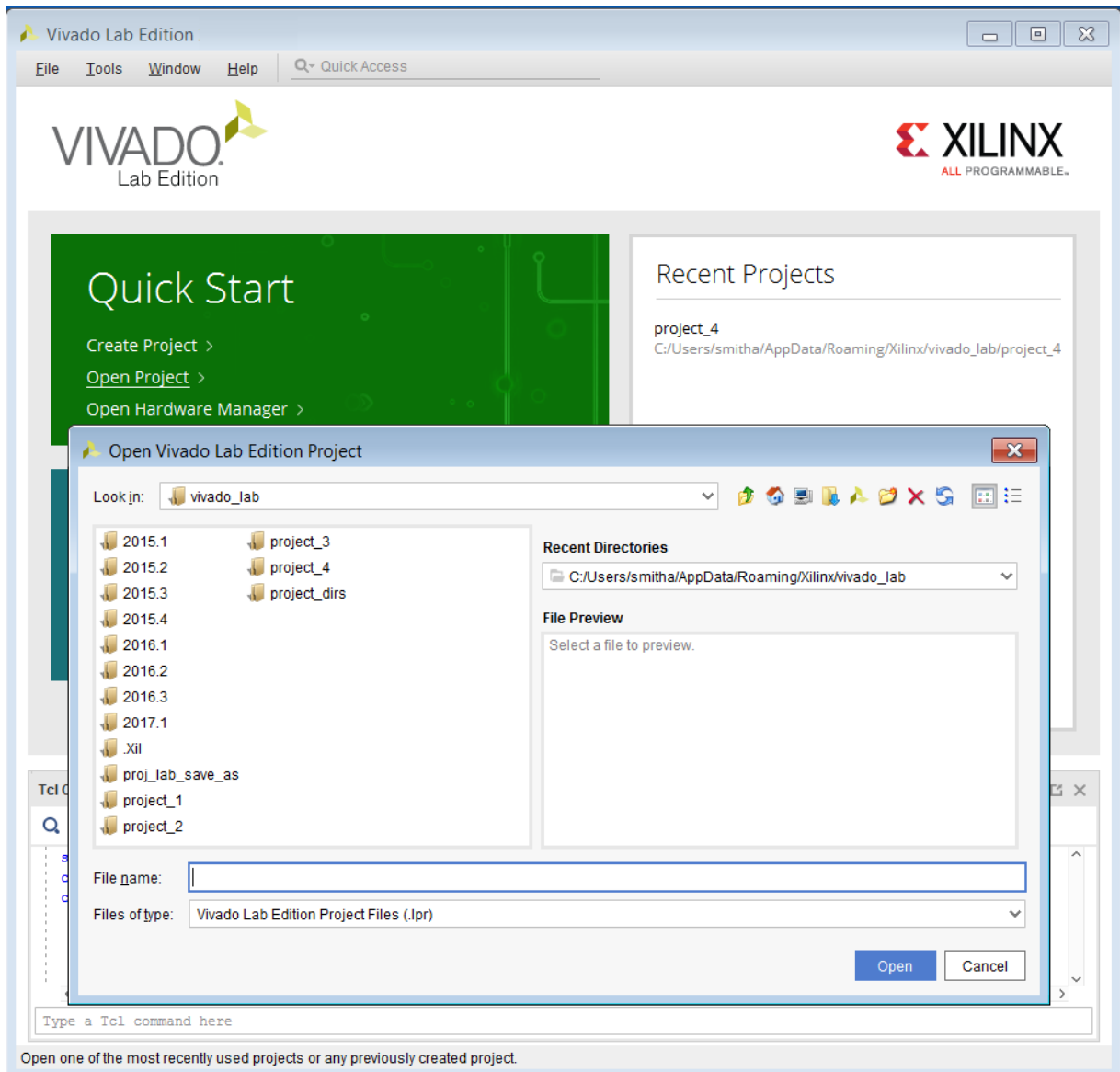
プロジェクトは、Tcl コマンドを使用しても作成できます。Vivado Lab Edition の [Tcl Console] ウィンドウに次のコマンドを入力するか、source コマンドを使用して Tcl ファイルから読み込みます。

```
create_project project_1 C:/Lab-edition/project_1
```

プロジェクトを開く

既存のプロジェクトを開くには、次の図に示す [Open Project] アイコンをクリックするか [Recent Projects] リストにあるプロジェクトをダブルクリックします。Vivado Lab Edition プロジェクト ファイル (.lpr) を開くダイアログボックスが表示されます。[Recent Projects] リストには、デフォルトで 10 個のプロジェクトが表示されます。この数を変更するには、[Tools]→[Settings] をクリックしてプロジェクト オプションをアップデートします。プロジェクトが表示される前に、そのプロジェクト データが存在するかどうかを確認されます。

図 4: [Open Vivado Lab Edition Project] ダイアログ ボックス



Tcl コマンドを使用してプロジェクトを開く

プロジェクトは Tcl コマンドを使用して開くこともできます。Vivado Lab Edition の [Tcl Console] ウィンドウに次のコマンドを入力するか、.tcl コマンドを使用して Tcl ファイルから読み込みます。

```
open_project C:/Lab_edition/project_1/project_1.lpr
```

Vivado Design Suite からの既存の .lpr プロジェクトを使用

Vivado Design Suite では、プロジェクトの開始時に .lpr ファイルが作成され、ハードウェア マネージャーを使用してプロジェクトのデザインをプログラムまたはデバッグするときに、このファイルに適切な詳細情報が記述されます。このファイルは、project_name.hw ディレクトリに project_name.lpr という名前で保存されます。このプロジェクト ファイルを Vivado Lab Edition で開くことができます。

典型的なフローは次のとおりです。

1. Vivado Lab Edition の最初の画面で [Open Project] をクリックします。
2. Vivado IDE プロジェクト ディレクトリにある project_name.hw ディレクトリに移動します。
3. .lpr ディレクトリ内にある project_name.hw プロジェクト ファイルを選択し、[OK] をクリックします。
4. ハードウェアに接続します。
5. 該当する Vivado の run ディレクトリの中から正しいデバイス イメージ ファイルおよび .ltx ファイルを使用してプログラムおよびデバッグを実行します。
6. ユーザー プリファレンス、ランタイム マネージャー デバッグ ダッシュボード、ウィンドウ設定は、プロジェクトを開いたときに復元されます。

プログラム機能

プロジェクトを開き、ハードウェア マネージャーをターゲット デバイスに接続したら、Vivado Design Suite で使用できるすべてのプログラム 機能を Vivado Lab Edition から使用できます。プログラム関連の Tcl コマンドはすべて Vivado Lab Edition でサポートされています。プログラム機能の詳細は、「[コンフィギュレーション メモリ デバイスのプログラム](#)」を参照してください。

関連情報

[コンフィギュレーション メモリ デバイスのプログラム](#)

デバッグ機能

プロジェクトを開き、ハードウェア マネージャーをターゲット デバイスに接続したら、Vivado Design Suite で使用できるすべてのデバッグ 機能を Vivado Lab Edition から使用できます。デバッグ関連の Tcl コマンドはすべて Vivado Lab Edition でサポートされています。デバッグ機能の詳細は、第 11 章「ハードウェアでのロジック デザインのデバッグ」を参照してください。

関連情報

[ハードウェアでのロジック デザインのデバッグ](#)

ビットストリームまたはデバイス イメージの生成

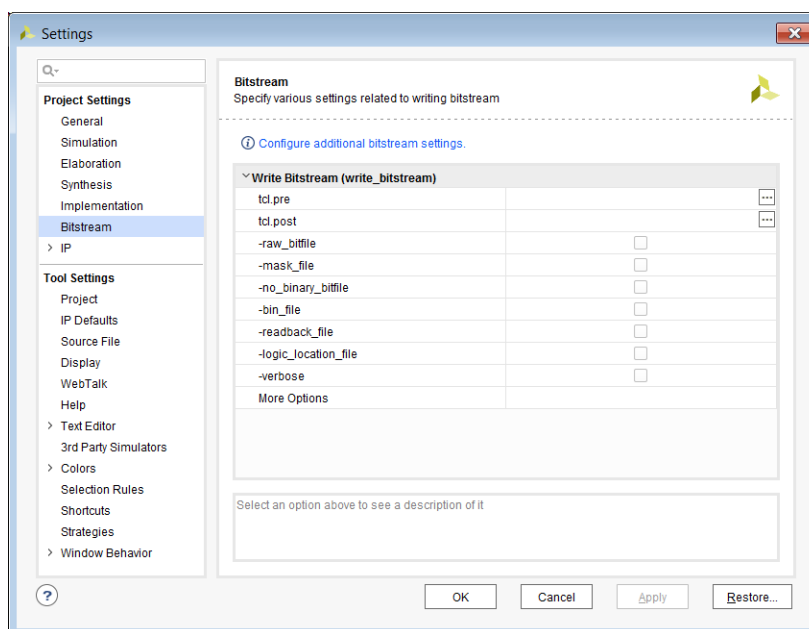
ビットストリームまたはデバイス イメージを生成する前に、それらの設定が正しいかどうかを確認することが重要です。

Vivado® IDE には、次の 2 つのタイプのビットストリームおよびデバイス イメージ設定があります。

1. ビットストリームまたはデバイス イメージ ファイルのフォーマット設定。
2. デバイス コンフィギュレーション設定。

Vivado Flow Navigator で [Settings] → [Bitstream] をクリックするか、[Flow] → [Settings] → [Bitstream Settings...] をクリックして [Bitstream Settings] ダイアログ ボックスを開きます (次の図を参照)。ビットストリーム設定が正しいことを確認したら、`write_bitstream` Tcl コマンドまたは Vivado Flow Navigator の [Generate Bitstream] を使用してビットストリーム データ ファイルを生成できます。

図 5: [Settings] ダイアログ ボックスの [Bitstream] ページ



ビットストリーム ファイルのフォーマット設定の変更

デフォルトでは、`write_bitstream Tcl` コマンドでバイナリ ビットストリーム ファイル(.bit) が生成されます。生成されるファイルのフォーマットを変更するには、次のオプションを使用します。`write_bitstream`

- `-raw_bitfile`: (オプション): ロー ビット ファイル(.rbt) を生成します。ロー ビット ファイルには、バイナリ ビットストリーム ファイルと同じ情報が ASCII 形式で含まれます。出力ファイル名は `filename.rbt` となります。
- `-mask_file` (オプション): マスク ファイル(.msk) を生成します。マスク ファイルには、ビットストリーム ファイルのコンフィギュレーション データが含まれる場所を示すマスク データが含まれます。このファイルは、検証時にビットストリームのどのビットをリードバック データと比較するべきかを判断するために使用します。マスク ビットが 0 の場合はそのビットはビットストリーム データに対して検証され、1 の場合は検証されません。出力ファイル名は `file.msk` となります。
- `.bit` (オプション): バイナリ ビットストリーム ファイル(.bit) を生成しません。このオプションは、バイナリ ビットストリーム ファイルを生成せずに、ASCII 形式のビットストリーム ファイルまたはマスク ファイルを生成したり、ビットストリーム レポートを生成したりする場合に使用します。
- `.ll` (オプション): ラッチ、フリップフロップ、LUT、ブロック RAM、I/O ブロック入力および出力のビットストリーム位置を示す ASCII 形式のロジック ロケーション ファイル(.ll) を生成します。このロケーション ファイルではビットがフレームおよびビット番号で示されるので、FPGA レジスタの内容を調べるのに役立ちます。
- `-bin_file` (オプション): デバイス プログラム データのみを含むバイナリ ファイル(.bin) を生成します。通常のビットストリーム ファイル(.bit) に含まれるヘッダー情報は含まれません。
- `-reference_bitfile <arg>` (オプション): 参照ビットストリーム ファイルを読み込み、指定した参照ファイルからの変更部分のみを含むインクリメンタル ビットストリーム ファイルを生成します。このパーシャル ビットストリームは、既存のデバイスをアップデートされたデザインでインクリメンタルにプログラムする場合に使用できます。

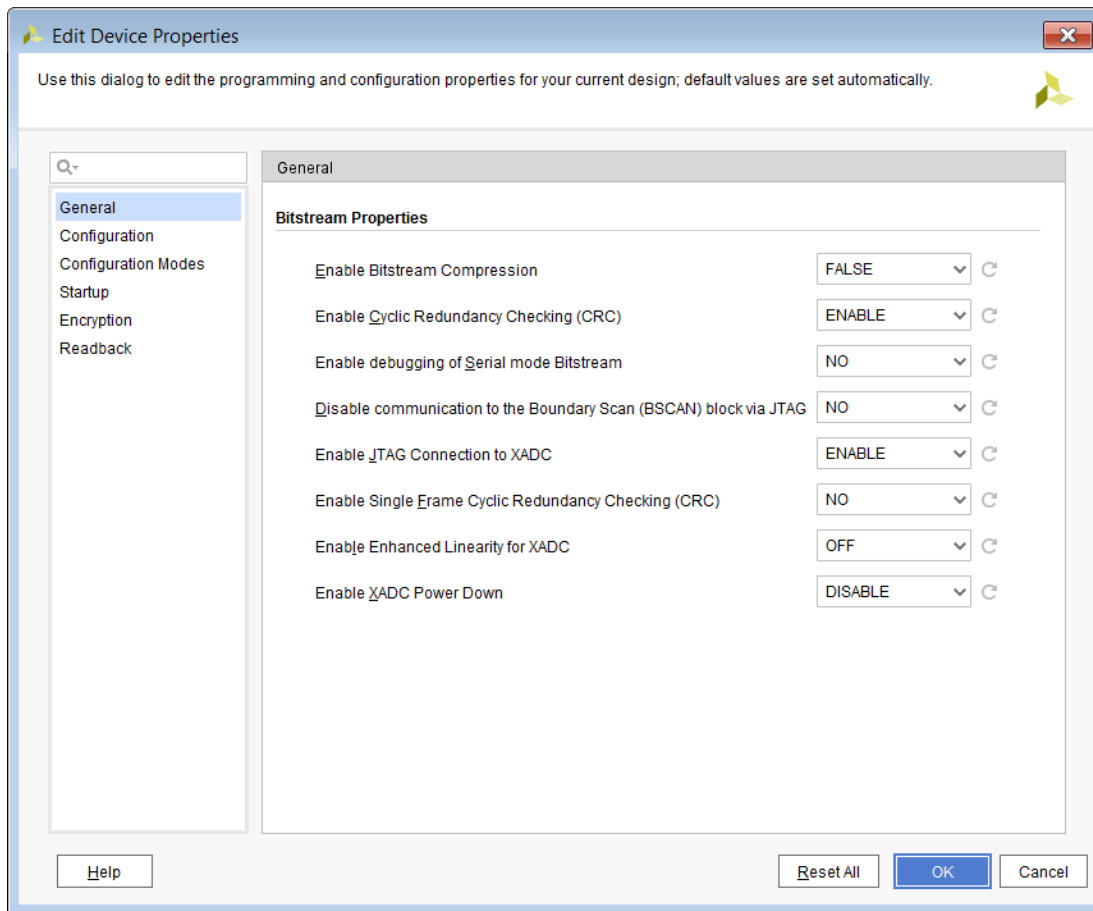
デバイス コンフィギュレーション ビットストリーム設定の変更

コンフィギュレーション設定で最も頻繁に変更されるのは、デバイス コンフィギュレーション設定です。これらの設定はデバイス モデルのプロパティであり、選択した合成済みまたはインプリメント済みデザイン ネットリストの [Edit Device Properties] ダイアログ ボックスで変更します。この方法でさまざまなビットストリーム プロパティを変更するには、次の手順に従います。

1. [Tools] → [Edit Device Properties] をクリックします。
2. [Edit Device Properties] ダイアログ ボックスの左側のカテゴリからいずれかを選択します。



ヒント: 検索フィールドにプロパティを入力します。たとえば、JTAG プログラムに関連するプロパティを検索および選択するには、「jtag」と入力します。



3. プロパティに値を設定し、[OK] をクリックします。
4. [File]→[Constraints]→[Save] をクリックし、アップデートしたプロパティをターゲット XDC ファイルに保存します。

ビットストリーム プロパティは、XDC ファイルで `set_property` コマンドを使用して設定することも可能です。次の例では、スタートアップ DONE サイクル プロパティを変更しています。

```
set_property BITSTREAM.STARTUP.DONE_CYCLE 4 [current_design]
```

その他の例は、Vivado テンプレートに含まれています。付録 A「デバイス コンフィギュレーション ビットストリーム設定」に、すべてのデバイス コンフィギュレーション設定が説明されています。



重要: 使用されるコンフィギュレーション モードに関連するデバイス コンフィギュレーション ビットストリーム設定のみを編集してください。それ以外の設定はすべてデフォルトのままにしておきます。

関連情報

[デバイス コンフィギュレーション ビットストリーム設定](#)

デバイスのプログラム

デバイス イメージを生成したら、次はそれをターゲット デバイスにダウンロードします。Vivado IDE には、この作業のためのインシステム デバイス プログラミング機能があります。

Vivado Design Suite および Vivado Lab Edition には、1 つまたは複数の FPGA デバイスを含むハードウェアに接続し、そのデバイスをプログラムしてアクセスする機能が含まれています。ハードウェアへの接続は Vivado Lab Edition または Vivado Design Suite の GUI、または Tcl コマンドで実行できます。いずれの場合も、ハードウェアに接続し、ターゲット デバイスをプログラムする手順は同じです。

1. ハードウェア マネージャーを開きます。
2. ホスト コンピューター上で稼働中のハードウェア サーバーで制御されているハードウェア ターゲットを開きます。
3. デバイス イメージを正しいハードウェア デバイスに関連付けます。
4. デバイス イメージをハードウェア デバイスにプログラムまたはダウンロードします。

ハードウェア マネージャーを開く

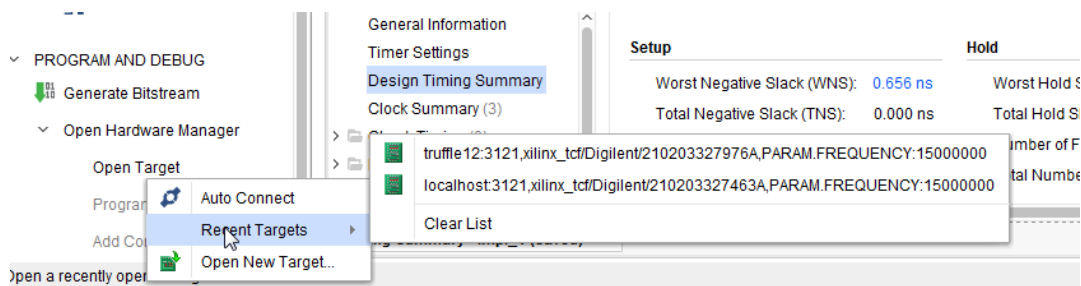
デザインをハードウェアにプログラムしたりデバッグするには、まずハードウェア マネージャーを開きます。ハードウェア マネージャーを開くには、次のいずれかを実行します。

- プロジェクトを開いている場合は、Vivado Flow Navigator で [PROGRAM AND DEBUG] → [Open Hardware Manager] をクリックします。
- [Flow] → [Open Hardware Manager] をクリックします。
- [Tcl Console] コンソール ウィンドウで `open_hw_manager` コマンドを実行します。

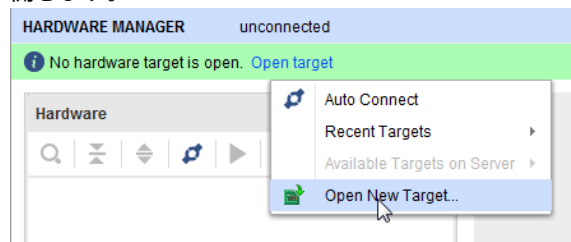
ハードウェア ターゲット接続を開く

次に、ハードウェア ターゲット (1 つ以上の FPGA デバイスで構成される JTAG チェーンを含むハードウェア ボードなど) を開き、ハードウェア ターゲットへの接続を制御するハードウェア サーバーに接続します。これには、次のいずれかを実行します。

- Flow Navigator で [Program and Debug] → [Hardware Manager] → [Open Target selection] をクリックし、新しいハードウェア ターゲットまたは最近開いたハードウェア ターゲットを開きます。



- [Open Target] → [Recent targets] またはハードウェア マネージャーの一番上の緑のバナーで [Open Target] → [Open New Target] をクリックし、最近開いたハードウェア ターゲットまたは新しいハードウェア ターゲットを開きます。



- Tcl コマンドを使用して、ハードウェア ターゲットへの接続を開きます。



ヒント: [Auto Connect] を使用すると、ローカル ハードウェア ターゲットに自動的に接続されます。

hw_server を使用したハードウェア ターゲットへの接続

ローカル マシン上のターゲットに接続すると、Vivado により hw_server が自動的に実行されます。hw_server は、ローカル マシンまたはリモート マシン上で手動で実行することもできます。Windows プラットフォーム上の Vivado フルインストールでは、cmd コマンド プロンプトで次のコマンドを実行します。

```
C:\Xilinx\Vivado\<Vivado_version>\bin\hw_server.bat
```

Windows プラットフォーム上のスタンドアロンのハードウェア サーバー インストールを使用する場合は、cmd コマンド プロンプトで次のコマンドを実行します。

```
c:\Xilinx\HWSRVR\<Vivado_version>\bin\hw_server.bat
```

次のセクションの手順に従って、このエージェントを使用して新しいハードウェア ターゲットへの接続を開きます。

互換性のある JTAG ダウンロード ケーブルおよびデバイスのリストは、[付録 D: hw_server でサポートされる JTAG ケーブルおよびデバイス](#) を参照してください。

SmartLynq データ ケーブルの使用に関する詳細は、『SmartLynq データ ケーブル ユーザー ガイド』([UG1258](#)) を参照してください。



重要: Vivado ハードウェア マネージャーが hw_server に接続されており、hw_server が停止した場合、ハードウェア マネージャーによりこの状況が自動的に検出され、サーバーからの接続が解除されます。

新しいハードウェア ターゲットを開く

Open New Hardware Target ウィザードでは、ウィザードの指示に従いながら、ハードウェア サーバーとターゲットを接続できます。ウィザードでのプロセスは次のとおりです。

1. ハードウェア ターゲットを接続するローカルまたはリモート サーバーを選択します。

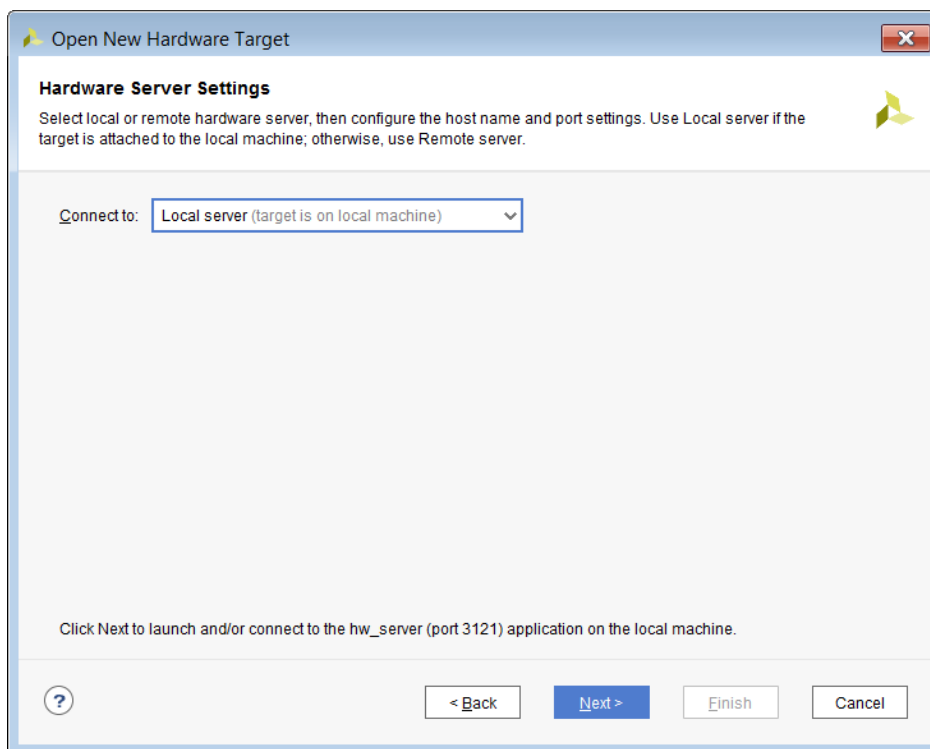
- [Local server]: ハードウェア ターゲットを Vivado Lab Edition または Vivado IDE を実行しているマシンに接続する場合に選択します。Vivado により、ローカル マシン上で Vivado ハードウェア サーバー (hw_server) アプリケーションが自動的に開始されます。
- [Remote server]: ハードウェア ターゲットを Vivado Lab Edition または Vivado IDE を実行している別のマシンに接続する場合に選択します。リモート マシンのホスト名または IP アドレスと、そのマシン上で実行中のハードウェア サーバー (hw_server) アプリケーションのポート番号を指定します。リモート デバッグの詳細は、「ラボ マシンで動作中の hw_server サーバーへの接続」を参照してください。

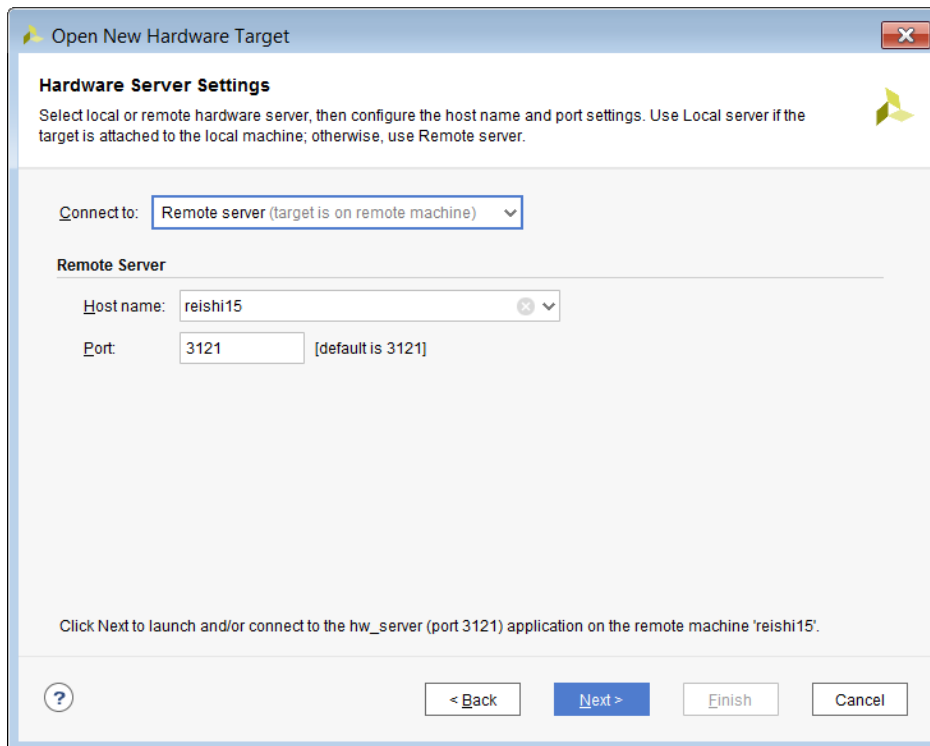


重要: リモート サーバーを使用する場合は、ハードウェア サーバーに接続するのに使用する Vivado のバージョンと同じバージョンの Vivado ハードウェア サーバー (hw_server) アプリケーションを手動で開始する必要があります。

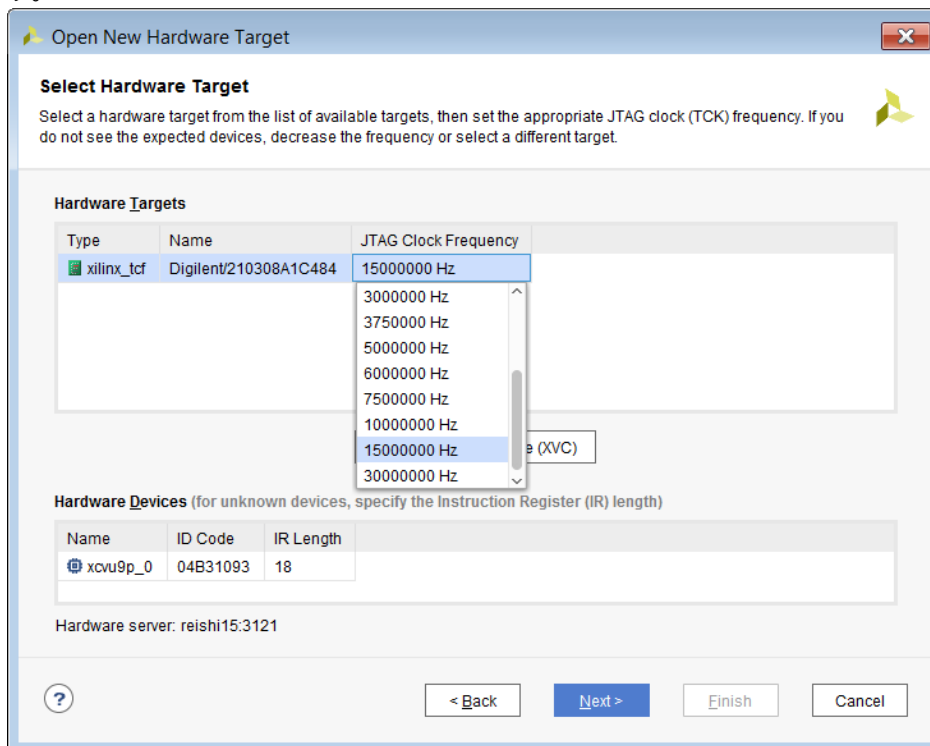


ヒント: ラボ マシンにリモートで接続する場合は、そのリモート マシン上に Vivado Design Suite のフルバージョンをインストールする必要はありません。軽量の Vivado ハードウェア サーバー (スタンドアロン) ツールをインストールできます。





2. ハードウェア サーバーで制御されているターゲットのリストから、適切なハードウェア ターゲットを選択します。ターゲットを選択すると、そのハードウェア ターゲットで使用可能なハードウェア デバイスが表示されます。



重要: JTAG チェーンにサードパーティ デバイスがある場合は、[ザイリンクス アンサー 61312](#) を参照し、その未知のデバイスの IDCODE、IR 幅、名前を追加してください。

関連情報

[ラボ マシンで動作中の hw_server サーバーへの接続](#)

ハードウェア ターゲットのトラブルシューティング

ハードウェア ターゲットに接続する際に問題が発生することがあります。次に、よく発生する問題とその解決方法を示します。

- ターゲット上のハードウェア デバイスを正しく特定できない場合は、ハードウェアがデフォルトのターゲット周波数で動作できない可能性があります。ハードウェア ターゲットまたはケーブルの TCK ピンの周波数は調整できます (前の図を参照)。ハードウェア ターゲットの各タイプでプロパティが異なることがあります。プロパティの詳細は、各ハードウェア ターゲットの資料を参照してください。
- Vivado ハードウェア サーバーにより JTAG チェーンに含まれるすべてのデバイスの 命令レジスタ (IR) 長の自動的な検出が試みられますが、まれに正しく検出できないことがあります。不明なデバイスの IR 幅が正しいことを確認してください。IR 幅を指定する必要がある場合は、Open New Hardware Target ウィザードの [Hardware Devices] の表に直接入力できます ([新しいハードウェア ターゲットを開く] を参照)。

関連情報

[新しいハードウェア ターゲットを開く](#)

最近開いたハードウェア ターゲットを開く

Open New Hardware Target ウィザードでは、前に接続したハードウェア ターゲットのリストも生成されます。ウィザードを使用してハードウェア ターゲットに接続しなくても、Hardware Manager 環境で [Open recent target] リンクをクリックし、リストから最近接続したハードウェア サーバー/ターゲットの組み合わせを選択すると、前に接続したハードウェア ターゲットへの接続を再開できます。この最近使用されたターゲットのリストには、Vivado IDE の Flow Navigator の [PROGRAM AND DEBUG] → [Hardware Manager] → [Open Target] からアクセスできます。

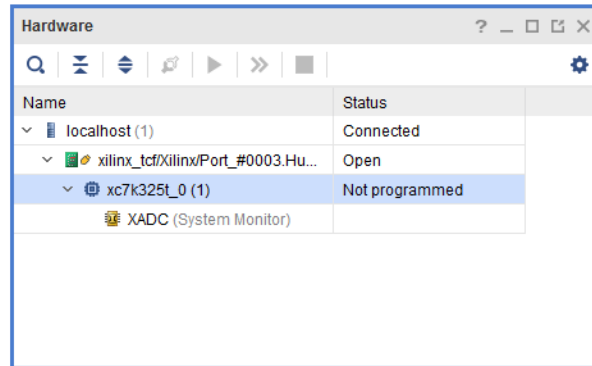
Tcl コマンドを使用してハードウェア ターゲットを開く

Tcl コマンドを使用して、ハードウェア サーバー/ターゲットに接続することも可能です。たとえば、localhost:3121 上の hw_server で制御される diligent_plugin ターゲット (シリアル番号 210203339395A) に接続するには、次の Tcl コマンドを使用します。

```
connect_hw_server -url localhost:3121
current_hw_target [get_hw_targets */xilinx-tcf/Digilent/210203339395A]
set_property PARAM.FREQUENCY 15000000 [get_hw_targets \
*/xilinx-tcf/Digilent/210203339395A]
open_hw_target
```

ハードウェア ターゲットへの接続を開くと、[Hardware] ウィンドウにハードウェア サーバー、ハードウェア ターゲット、および開いているターゲット用のさまざまなハードウェア デバイスが表示されます。

図 6: ハードウェア ターゲットへの接続を開いた後の [Hardware] ウィンドウ



プログラム ファイルをハードウェア デバイスに関連付け

ハードウェア ターゲットに接続したら、デバイスをプログラムする前に、ビットストリーム データ プログラム ファイルをデバイスに関連付ける必要があります。[Hardware] ウィンドウでハードウェア デバイスを選択し、[Properties] ウィンドウで [Programming File] プロパティが適切なプログラム ファイルに設定されていることを確認します。

注記: Vivado IDE では、開いているハードウェア ターゲットの最初のデバイスの [Programming File] プロパティ値として、現在のインプリメント済みデザインのプログラム ファイルが自動的に使用されます。この機能は、Vivado IDE をプロジェクト モードで使用した場合にのみ使用できます。Vivado IDE を非プロジェクト モードで使用する場合は、このプロパティを手動で設定する必要があります。

また、`set_property Tcl` コマンドを使用してハードウェア デバイスの `PROGRAM.FILE` プロパティを設定できます。

```
set_property PROGRAM.FILE {C:/<path_to_programming_file>} [lindex
[get_hw_devices] 0]
```

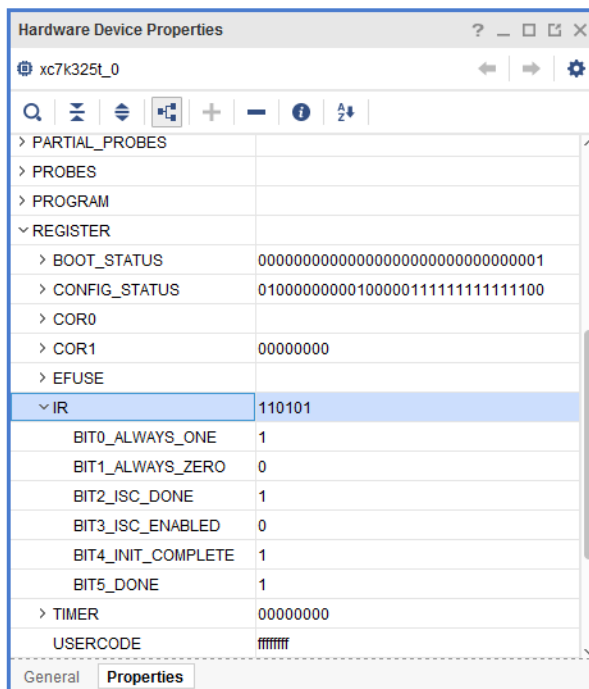
ハードウェア デバイスのプログラム

プログラム ファイルをハードウェア デバイスに関連付けたら、[Hardware] ウィンドウでデバイスを右クリックし、[Program Device] をクリックして、ハードウェア デバイスをプログラムします。`program_hw_device Tcl` コマンドでも同じ操作を実行できます。たとえば、JTAG チェーンの最初のデバイスをプログラムするには、次の `Tcl` コマンドを使用します。

```
program_hw_devices [lindex [get_hw_devices] 0]
```

進捗状況インジケータでプログラムが 100% 完了したことが示されたら、プログラムが正常に完了したかを [Hardware Device Properties] ウィンドウの `DONE` のステータスで確認できます。

図 7: FPGA デバイスの DONE ステータスを確認



DONE のステータスは、`get_property Tcl` コマンドでも確認できます。たとえば、JTAG チェーンの最初のデバイスである Kintex®-7 デバイスの DONE ステータスを確認するには、次の Tcl コマンドを使用します。

```
get_property REGISTER.IR.BIT5_DONE [lindex [get_hw_devices] 0]
```

フラッシュ デバイスを使用したり、iMPACT ツールなどの外部デバイス プログラム ツールを使用するなど、別の方法でハードウェア デバイスをプログラムした場合は、ハードウェア デバイスを右クリックして [Refresh Device] をクリックするか、`refresh_hw_device` Tcl コマンドを実行すると、ハードウェア デバイスのステータスを更新できます。これにより、DONE ステータスだけでなく、デバイスのさまざまなプロパティが更新されます。



重要: デザインにデバッグ コアが含まれる場合は、JTAG クロックがデバッグ ハブ クロックよりも 2.5 倍遅くなるようにします。



重要: ユーザー スキャン チェーン: Vivado プログラマでは、デフォルトで指定したユーザー スキャン チェーン上のデバッグ コアの検出が試みられます。検出では、JTAG_CHAIN 1 コマンドがデバイスに出力されます。デバッグ コアを含まないデザイン、またはユーザー スキャン チェーン 2 または 4 のデバッグ コアを含むデザインをプログラムする場合は、警告メッセージが表示されます。

ユーザー スキャン チェーン設定を確認するには、インプリメント済みデザインを開いて次のコマンドを使用します。

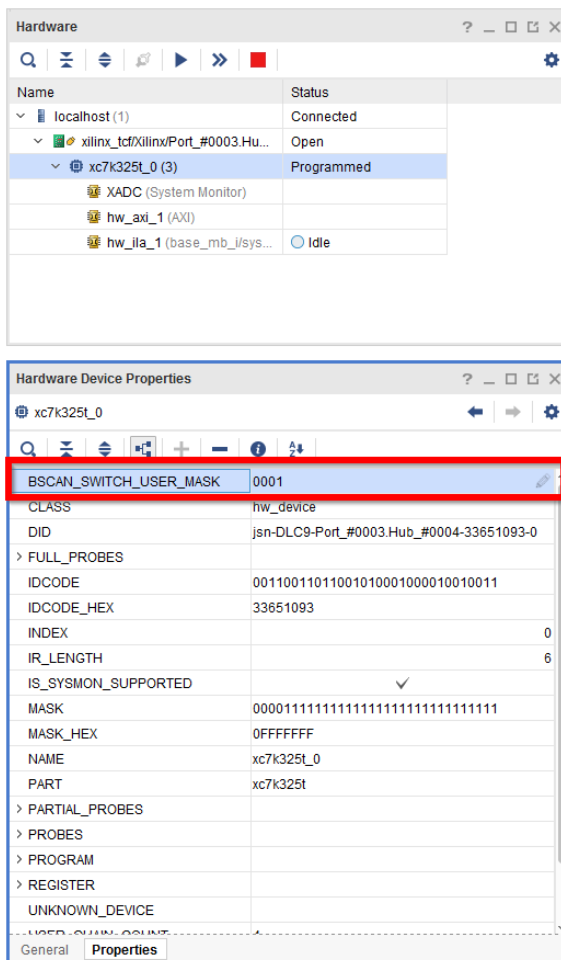
```
get_property C_USER_SCAN_CHAIN [get_debug_cores dbg_hub]
```

Vivado ハードウェア マネージャーで使用されるユーザー スキャン チェーンは変更できます。
BSCAN_SWITCH_USER_MASK はビット マスク値です。次の図を参照してください。

または、`hw_server` スタートアップにオプションとしてユーザー スキャン チェーンの値を指定することもできます。

```
hw_server -e "set bscan-switch-user-mask <user-bit-mask>"
```

図 8: BSCAN スイッチ ユーザー マスク



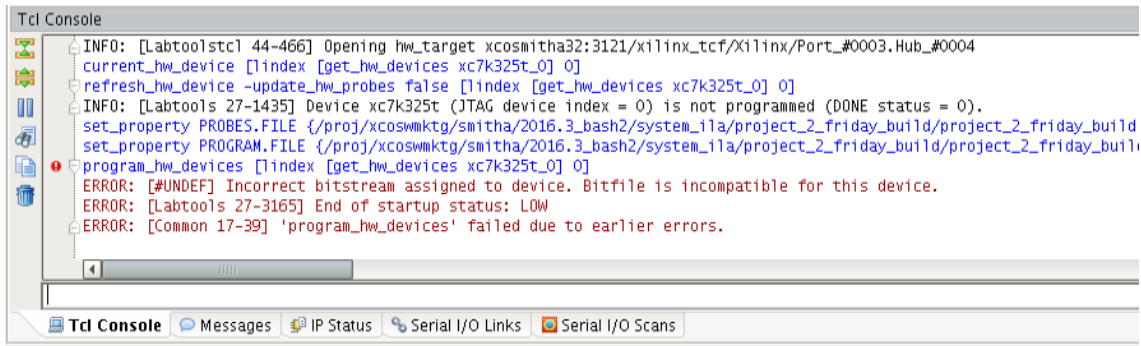
ヒント: Vivado 2016.3 よりも前のデザインの場合は、hw_server に `-e"set xsdb-user-bscan <C_USER_SCAN_CHAIN scan_chain_number>"` を付けて手動で起動し、ユーザー スキャン チェーン 2 または 4 のデバッグ ハブを検出してください。

間違ったビットストリーム割り当てられていることを示すメッセージ

次の場合、Vivado ハードウェア マネージャーで「incorrect bitstream assignment」というメッセージが生成されます。

- 別の FPGA 用に生成されたビットストリームで FPGA デバイスをプログラムしようとした。
たとえば、XC7VU190 のビットストリームで XCKU115 をプログラムしようとする、次のようなエラー メッセージが表示されます。

図 9: XCVU190 のビットストリームで XCKU115 をプログラムしようとした場合のエラー メッセージ

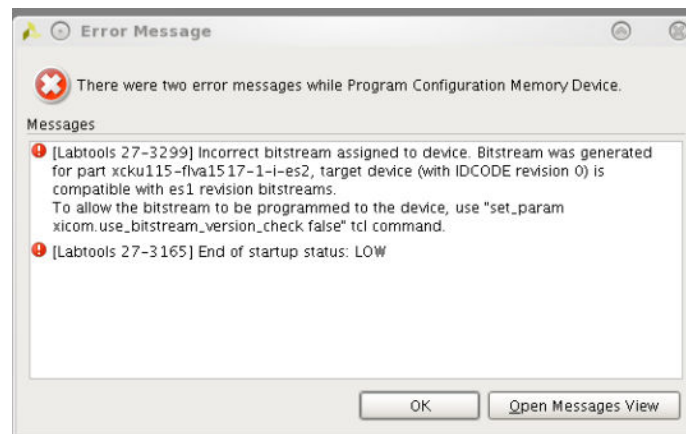


これを解決するには、実際にプログラムしている FPGA に合ったビットストリームを指定します。

別のシリコン リビジョン用に生成されたイメージを使用したデバイスのプログラム

- 別のシリコン リビジョン用に生成されたビットストリームで FPGA デバイスをプログラムすることはサポートされておらず、正しく機能しない場合があります、またデバイスが破損する可能性があります。
 - たとえば、XCKU115 デバイスには -es1 および -es2 というシリコン バージョンがあり、さらにプロダクションシリコン バージョンもあります。-es1 および -es2 というシリコン バージョンの接尾辞は、Vivado IDE プロジェクトの FPGA を選択するときに表示されます。プロダクション バージョンには接尾辞はありません。
 - 別のシリコン バージョン用に生成されたビットストリームでデバイスをプログラムしないでください。たとえば、-es1 シリコン用に生成されたビットストリームを -es2 シリコンにプログラムしたり、-es2 シリコン用に生成されたビットストリームを -es1 シリコンにプログラムしたりしないでください。どちらの場合でも、次の図に示すようなエラー メッセージが表示されます。

図 10: ビットストリームとシリコン バージョンが一致しない



FPGA デバイスに接続されたコンフィギュレーション メモリのプログラム

FPGA デバイスに接続されたコンフィギュレーション メモリをプログラムするには、まず Vivado ハードウェア マネージャーで FPGA デバイスにフラッシュ コントローラー ビットストリームをダウンロードします。その後、ハードウェア マネージャーにより FPGA デバイスの JTAG ポートを介してフラッシュ メモリに書き込むコマンドおよびデータが送信されてコントローラーで処理され、処理されたフラッシュに書き込むコマンドおよびデータがコントローラーからコンフィギュレーション メモリ インターフェイスに送信されます。

ハードウェア マネージャーによりダウンロードされるコントローラー ビットストリームは、FPGA デバイスの最新シリコン リビジョン用に生成されます。たとえば、2016.3 以降のバージョンでは、XCKU115 のコンフィギュレーション メモリ コントローラー ビットストリームは XCKU115-es2 シリコン用に生成されます。

FPGA に接続されたコンフィギュレーション メモリをプログラムする際、ボード上に XCKU115-es1 デバイスがある場合、「別のシリコン リビジョン用に生成されたイメージを使用したデバイスのプログラム」に示すようなエラーメッセージが表示されます。これは、ハードウェア マネージャーで -es2 のフラッシュ コントローラー ビットストリームを -es1 デバイスにダウンロードしようとしたからです。

関連情報

[別のシリコン リビジョン用に生成されたイメージを使用したデバイスのプログラム](#)

ハードウェア ターゲットを閉じる

ハードウェア ターゲットを閉じるには、[Hardware] ウィンドウでハードウェア ターゲットを右クリックし、[Close Target] をクリックします。Tcl コマンドでも同じ操作を実行できます。たとえば、localhost サーバー上の xilinx_platformusb/USB21 ターゲットを閉じるには、次の Tcl コマンドを使用します。

```
close_hw_target {localhost/xilinx_tcf/Digilent/210203339395A}
```



重要: ボードの電源がオフになった場合、またはケーブルの接続が解除された場合、Vivado IDE によりハードウェア マネージャーのハードウェア ターゲットが閉じられます。メインの Vivado スレッドに対する Vivado の操作もキャンセルされます。ボードの電源が再びオンになるか、ケーブルが再接続されると、Vivado ハードウェア マネージャーのハードウェア ターゲットが再び開きます。

ハードウェア サーバーへの接続を閉じる

ハードウェア サーバーへの接続を閉じるには、[Hardware] ウィンドウでハードウェア サーバーを右クリックし、[Close Server] をクリックします。Tcl コマンドでも同じ操作を実行できます。たとえば、localhost サーバーへの接続を閉じるには、次の Tcl コマンドを使用します。

```
disconnect_hw_server localhost
```



重要: Vivado ハードウェア マネージャーが hw_server に接続されており、hw_server が停止した場合、ハードウェア マネージャーによりこの状況が自動的に検出され、サーバーからの接続が解除されます。

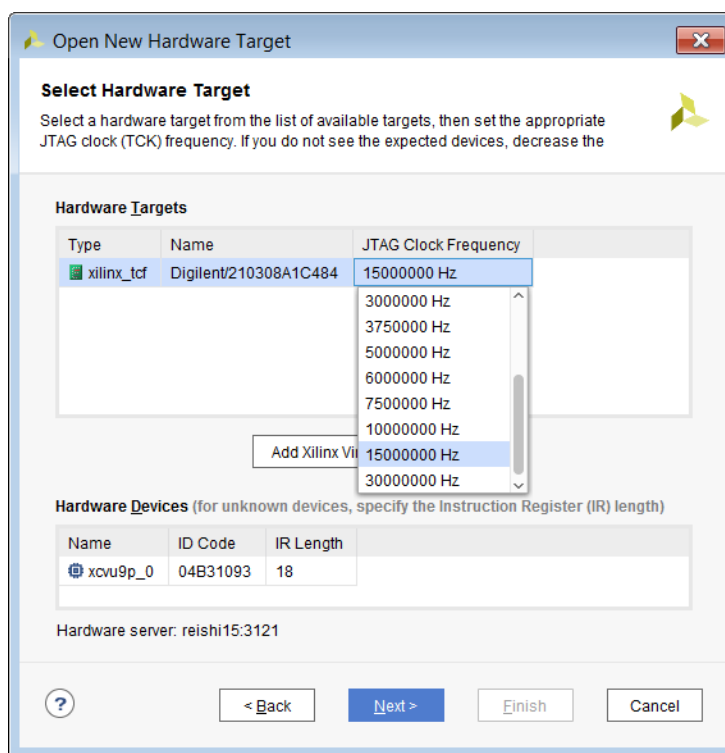
低い JTAG クロック周波数でのターゲット デバイスへの接続

JTAG チェーンは、チェーン内で一番遅いデバイスと同じ速さになります。JTAG クロックの周波数を下げるには、JTAG クロック周波数がデフォルトの JTAG クロック周波数よりも低いターゲット デバイスに接続します。

デフォルトの JTAG クロック周波数 (ケーブル接続の場合は 15 MHz、USB ケーブル接続の場合は 6 MHz) を使用して開いてみる必要があります。ザイリックス では、これらの速度で接続できない場合は、次に説明するようにデフォルトの JTAG 周波数をさらに下げることをお勧めします。

JTAG クロック周波数を変更するには、Vivado® Design Suite の Open New Hardware Target ウィザードを使用します。

図 11: Vivado で低い JTAG 周波数を選択



または、次の Tcl コマンド シーケンスを使用します。

```
open_hw_manager
```

```
connect_hw_server -url machinename:3121
```

```
current_hw_target [get_hw_targets */xilinx_tcf/Digilent/210203327962A]
```

```
set_property PARAM.FREQUENCY 250000 [get_hw_targets */xilinx_tcf/Digilent/  
210203327962A]  
open_hw_target
```

JTAG チェーンに 32 個を超えるデバイスを含むサーバーへの接続

Vivado では、JTAG チェーンに 32 個を超えるデバイスを含むサーバーへ接続することが可能です。スキャン チェーンでデバイスを検出するための機能を有効にするため、`max-jtag-devices` の開始時に `max-jtag-devices` オプションを指定する必要があります。この設定のデフォルト値は 32 です。この値を大きくすると、デバイス検出プロセスに時間がかかり、ケーブル アクセスが遅くなってしまう可能性があることに注意してください。

次のように、`max-jtag-devices` の開始時に `hw_server` オプションを指定します。

```
hw_server -e "set max-jtag-devices 64"
```

使用法

このオプションでは、64 ビットを超える IR 幅を使用できる状態で `hw_server` が開始できます。この設定のデフォルト値は 64 です。この値は、JTAG チェーンのデバイスの IR 幅がこれより大きい (たとえば 93 など) 場合に増加できます。この値を大きくするほど、デバイス検出プロセスに時間がかかり、ケーブル アクセスが遅くなってしまう可能性があることに注意してください。このため、この値は IR 幅が長く、デバイス数の多いシステムに対してのみ使用してください。

次のように、`hw_server` 起動時にオプションを指定します。

```
hw_server -e "set max-ir-length 93"
```

init オプション

`--init=script.txt` オプションを使用すると、ファイルを使用してこの設定を読み込むことができます。`init` オプションを使用するには、次の例に示すように初期化スクリプトを作成します。スクリプトで `set max-jtag-device` パラメーターを使用してください。

```
# Sample script.txt
set max-ir-length 93
```

次の例に示すように、`hw_server` を開始します。

```
hw_server --init=script.txt
```

Vivado でのリモート デバッグ

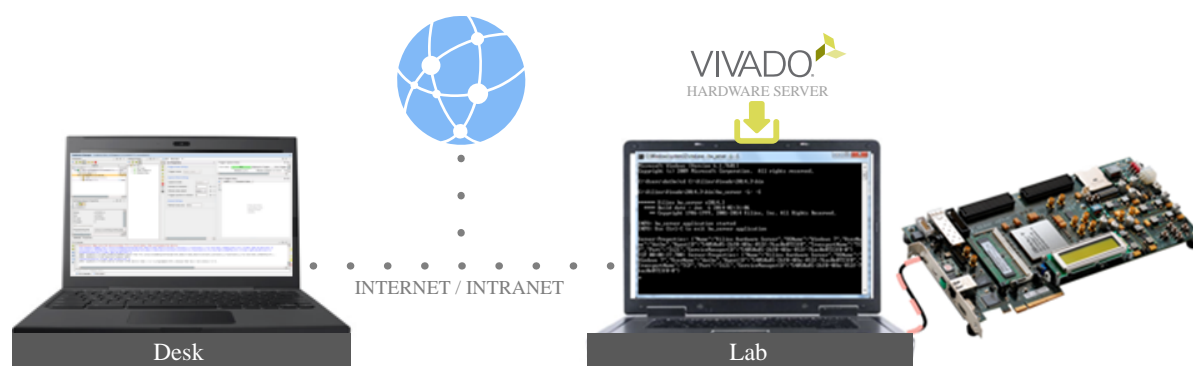
さまざまな状況でリモート デバッグが必要になることが多くなっています。たとえば、ラボへの物理的なアクセスのない場合にラボでデザインをデバッグしたり、組織内でリソースを共有するなど、製品のプロトタイプ段階でリモート デバッグが必要になる可能性があります。リモート デバッグは、問題を診断したり、製品ライフサイクルを伸ばすためのインフィールド デバッグに必要なこともあります。

ザイリンクスでは、デザインをリモートでデバッグする複数のソリューションを提供しています。これは、ラボでイリックス ハードウェア サーバー製品を使用してリモート コンピューターに接続すると実行できます。また、ザイリンクス仮想ケーブル (XVC) プロトコルをインプリメントしても、ネットワークに接続されたボードに接続できます。これらの各ソリューションについては、次のセクションで説明します。

イーサネットから Vivado ハードウェア サーバーを使用したデバッグ

リモート ラボ マシンには、Vivado ハードウェア サーバー製品を使用すると接続できます。これは、100 MB 未満の小さいサイズのスタンドアロン ダウンロードで、ラボ マシンにインストールします。このオプションを実行するには、イントラネットまたはインターネット アクセスが必要で、ユーザーの組織内部でのみ使用できます。

図 12: イントラネット/インターネットからハードウェア サーバーを使用したデバッグ



X14741-062315

ザイリンクス仮想ケーブル (XVC)

Vivado IDE では、ザイリンクス仮想ケーブル (XVC) プロトコルがサポートされます。ザイリンクス仮想ケーブルを使用すると、USB またはパラレル コンフィギュレーション ケーブルなしにザイリンクス デバイスにアクセスしてデバッグができます。この機能により、次のようなデザインを Vivado IDE でデバッグしやすくなります。

- ラボと PC が近くになく、アクセスしにくい位置に FPGA がある。
- ボードのデバイス ピンへの直接アクセスがない (例: JTAG ピンにはローカル マイクロプロセッサ インターフェイスからのみアクセス可能)。

XVC は JTAG ケーブルのように動作するインターネット ベース (TCP/IP) プロトコルで、とても基本的なケーブル コマンドを含みます。XVC を使用すると、イントラネット、場合によってはインターネットを介してシステムをデバッグできます。この機能により、経費のかかる出張などの移動を削減できるほか、リモートにあるシステムのデバッグにかかる時間が短縮できます。

XVC は共有システムにアクセスする必要があるチームが同じ場所にいない場合にもよく使用されます。また、JTAG コネクタが使用できない場合など、システムの使用に物理的な制約が課せられているときにも使用できます。XVC インプリメンテーションはプログラム言語やプラットフォームに依存しません。

プロセッサからターゲット デバイスへの適切な JTAG コマンドを作成するには、専用の JTAG ヘッダーを使用するよりも、既存のイーサネット接続を使用します。XVC v1.0 プロトコルを使用すると、イーサネット接続を使用して Vivado から同じ JTAG コマンドを使用でき、既存の Vivado デバッグ機能が引き続きサポートされます。



重要: XVC に Vivado Debug Bridge IP を使用する場合、Vivado IDE ではプログラム機能はサポートされません。XVC を使用してデザインをデバッグするには、デバイスをプログラムしておく必要があります。

Vivado Debug Bridge IP およびザイリンクス仮想ケーブル (XVC) フロー

Vivado Debug Bridge IP コアは、デザイン内のデバッグ コアと通信するためのオプションを複数提供するコントローラーです。この場合、デザインはフラット デザインまたは Dynamic Function eXchange デザインにできます。さらに、Debug Bridge IP コアは、JTAG ケーブルを使用するか、JTAG ケーブルを使用しないでイーサネット、PCIe、またはその他のインターフェイスを介してリモートでデザインをデバッグできるように設定することもできます。

Debug Bridge IP には異なるモードがあり、さまざまなユース ケースをサポートできるようになっています。

XVC モードの Debug Bridge

Debug Bridge には、ザイリンクス仮想ケーブル (XVC) インプリメンテーションで使用されるモードがあります。

- From AXI to BSCAN: Debug Bridge は XVC コマンドを AXI4-Lite スレーブ インターフェイスを介して受信します。
- From JTAG to BSCAN: このモードでは、Debug Bridge はユーザー ロジックで駆動される JTAG スレーブ インターフェイスを介して XVC コマンドを受信することになっています。
- From PCIe to BSCAN: Debug Bridge は XVC コマンドを PCIe 拡張コンフィギュレーション スレーブ インターフェイスを介して受信します。
- From PCIe to JTAG: Debug Bridge は XVC コマンドを PCIe 拡張コンフィギュレーション インターフェイスを介して受信します。この Debug Bridge により、I/O ピンを介して FPGA から JTAG ピンが接続されます。このモードは主に XVC を介して別のボードのデザインをデバッグするために使用されます。
- From AXI to JTAG: このモードでは、Debug Bridge は、AXI4-Lite を介して XVC コマンドを受信し、ターゲット デバイスへ JTAG ピンを介して送信します。

この 3 つのモードのすべてで、Debug Bridge は、ソフト BSCAN (バウンダリスキャン) インターフェイスを介して、デザインのほかのデバッグ コア/Debug Bridge インスタンスと通信できます。ソフト BSCAN マスター インターフェイスを使用すると、内部ユーザー定義されているスキャン チェーン/Debug Bridge インスタンスへ JTAG インターフェイスを拡張できます。

パーシャル リコンフィギュレーション (PR) での Debug Bridge IP の使用

Debug Bridge IP は、フラット デザインおよび PR デザインの両方で使用できます。PR デザインのスタティックまたはパーシャル リコンフィギュレーション (PR) 領域で使用される Debug Bridge 設定の詳細は、次のとおりです。デザイン要件によっては、1 つのパーティションに複数の Debug Bridge インスタンスを使用できます。

- BSCAN Primitive: BSCAN プリミティブを含む Debug Bridge がスタティック領域で必要な場合に使用します。この Debug Bridge の BSCAN マスター インターフェイスは、スタティックまたは PR 領域にある別の Debug Bridge インスタンスに接続でき、これらの領域をデバッグするための通信路を 1 つまたは複数提供します。
- From BSCAN to Debug Hub: Debug Bridge で Vivado ハードウェア マネージャーとの通信に BSCAN スレーブ インターフェイスが使用されます。関連スタティックまたは PR 領域内のデザイン コアとの通信には、Debug Hub インターフェイスが使用されます。また、オプションで、この Debug Bridge の出力にさらに BSCAN マスターを追加できます。これで、MicroBlaze デバッグ モジュール (MDM) やほかの Debug Bridge インスタンスなどのデバッグ コアをデバッグできるようになります。

注記: パーティションにインスタンス化されている Debug Bridge が 1 つしかない場合、RP のデバッグ コアはこの Debug Bridge へ自動的に接続されます。

- From AXI to BSCAN: Debug Bridge は XVC コマンドを AXI4-Lite スレーブ インターフェイスを介して受信します。Debug Bridge は、ソフト BSCAN (バウンダリスキャン) マスター インターフェイスを介して、デザインのほかのデバッグ コア/Debug Bridge インスタンスと通信できます。ソフト BSCAN インターフェイスを使用すると、内部ユーザー定義されているスキャン チェーン/Debug Bridge インスタンスへ JTAG インターフェイスを拡張できます。
- From JTAG to BSCAN: このモードでは、Debug Bridge はユーザー ロジックで駆動される JTAG スレーブ インターフェイスを介して XVC コマンドを受信することになっています。Debug Bridge は、ソフト BSCAN (バウンダリスキャン) マスター インターフェイスを介して、デザインのほかのデバッグ コア/Debug Bridge インスタンスと通信できます。ソフト BSCAN インターフェイスを使用すると、内部ユーザー定義されているスキャン チェーン/Debug Bridge インスタンスへ JTAG インターフェイスを拡張できます。
- From PCIe to BSCAN: Debug Bridge は XVC コマンドを PCIe 拡張コンフィギュレーション スレーブ インターフェイスを介して受信します。この Debug Bridge は、ソフト BSCAN (バウンダリスキャン) インターフェイスを介して、デザインのほかのデバッグ コア/Debug Bridge インスタンスと通信できます。ソフト BSCAN マスター インターフェイスを使用すると、内部ユーザー定義されているスキャン チェーン/Debug Bridge インスタンスへ JTAG インターフェイスを拡張できます。

注記: このモードは UltraScale+ および UltraScale デバイス アーキテクチャにのみ使用できます

- From PCIe to JTAG: Debug Bridge は XVC コマンドを PCIe 拡張コンフィギュレーション インターフェイスを介して受信します。この Debug Bridge により、I/O ピンを介して FPGA から JTAG ピンが接続されます。このモードは主に XVC を介して別のボードのデザインをデバッグするために使用されます。

注記: このモードは UltraScale+ および UltraScale デバイス アーキテクチャにのみ使用できます。

- From AXI to JTAG: このモードでは、Debug Bridge は、AXI4-Lite を介して XVC コマンドを受信し、ターゲット デバイスへ JTAG ピンを介して送信します。

JTAG フォールバック サポート

XVC ベースのデバッグ ソリューションは、PCIe XDMA IP などの AXI マスターで使用できます。AXI マスターがハング状態であったり、正しく機能していない場合は、これらの状態をデバッグする方法はありません。XVC ベースのソリューションと並行して JTAG ベースのフォールバック デバッグを提供するには、Debug Bridge を BSCAN プリミティブ モードで使用してください。BSCAN プリミティブ モードの Debug Bridge はスタティック領域にインスタンス化でき、BSCAN マスター インターフェイスは、JTAG フォールバック サポートをイネーブルで設定された 2 番目の Debug Bridge の BSCAN スレーブ インターフェイスに接続できます。JTAG フォールバック サポートには次の 2 タイプがあります。

1. JTAG フォールバックをイネーブルにする必要のある Debug Bridge が PR 領域にある場合は、外部 BSCAN マスターの JTAG フォールバック サポートをイネーブルにする必要があります。
2. JTAG フォールバックをイネーブルにする必要のある Debug Bridge がスタティック領域 (またはフラット デザイン) にある場合は、内部 BSCAN マスターの JTAG フォールバック サポートをイネーブルにする必要があります。

MicroBlaze デバッグ モジュール (MDM) サポート

MicroBlaze デバッグ モジュール (MDM) へのデバッグ アクセスも Debug Bridge でサポートされています。MDM BSCAN スレーブ入力は、複数の BSCAN マスター インターフェイスを出力でサポートする任意の Debug Bridge 設定モードに接続できます (たとえば、AXI to BSCAN モードで、BSCAN マスター カウントは 0 よりも大きな値)。

複数のデバッグ ツリー

Debug Bridge IP では、複数の独立したデバッグ ツリーの設定およびコンフィギュレーションがサポートされます。アプリケーションで複数のデバッグ ツリーを使用すると、特定のデバッグ ロジックをほかのユーザーには非表示にしつつ、特定のユーザー (システム管理者など) のみに表示されるようにできます。独立したデバッグ ツリーは、スタンドアロンと Dynamic Function eXchange デザインの両方で設定でき、それぞれサポートされるデバッグ コア (ILA、VIO など) に接続できます。

この機能を使用するには、イネーブルにするデバッグ ツリーごとに、[From AXI to BSCAN] か [From PCIe to BSCAN] モードのいずれかで Debug Bridge IP を 1 つインスタンス化する必要があります。たとえば、複数のユーザー クラスが DUT にアクセスするようなデータセンター デザインの場合、カスタマー可視のアドレス マップでは [From AXI to BSCAN] モードで、管理者可視のアドレス マップでは [From AXI to BSCAN] モードで Debug Bridge IP をインスタンス化できます。

管理者またはカスタマーがデザインをデバッグできるようになったら、デバッグ コアとの通信方法に基づいた正しいデバイス オフセットで、Vivado ハードウェア マネージャーを使用してデバッグ ブリッジに接続するだけですみます。このモードでの PCIe コアと Debug Bridge 使用した XVC フローの詳細およびデザイン例については、『UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP 製品ガイド』 (PG213: [英語版](#)、[日本語版](#)) を参照してください。

次の表に、異なる Debug Bridge モードとそれらのモードで使用可能な機能を示します。

表 1: Debug Bridge モード

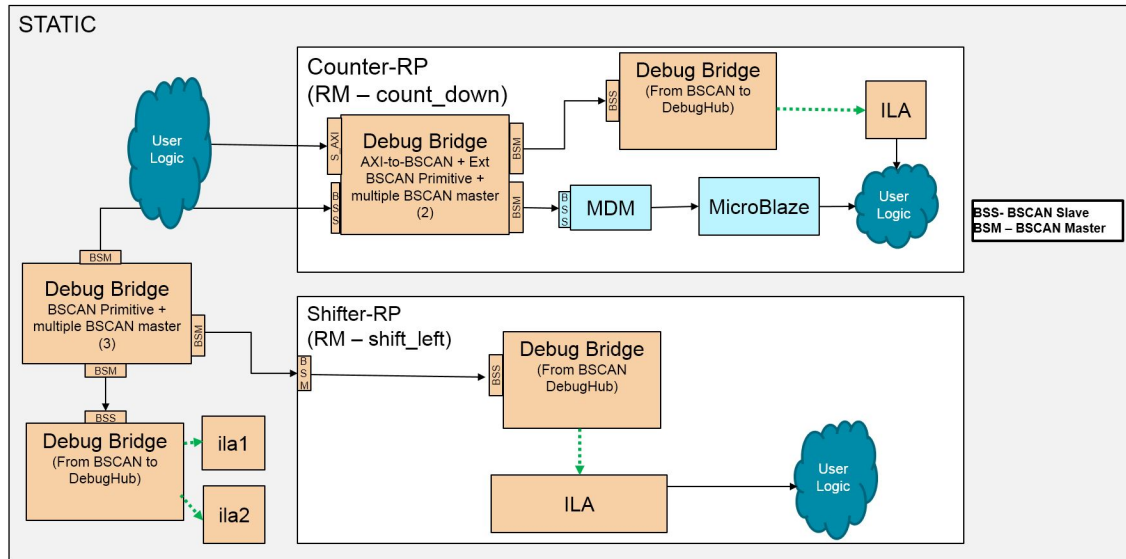
Debug Bridge モード	XVC サポート	リコンフィギュラブル パーティションで使用可能	JTAG フォールバック サポート	MDM サポート
From AXI to BSCAN	○	○ ¹	○ ²	○ ³
From JTAG to BSCAN	○	○ ¹	○ ²	○ ³
From PCIe to BSCAN	○	○ ¹	○ ²	○ ³
From PCIe to JTAG	○	○ ¹	該当なし	該当なし
From BSCAN to DebugHub	×	○ ¹	該当なし	○ ³
BSCAN Primitive	×	×	該当なし	○ ³
From AXI to JTAG	○	○	該当なし	該当なし

注記:

1. BSCAN マスター カウントは 0 よりも大きく、同じ RP 内の Debug Bridge インスタンスまたは MicroBlaze/MDM コアのみ に接続可能です。
2. 内部 BSCAN モードは Debug Bridge がスタティック パーティションにある場合にのみ使用可能で、外部 BSCAN モードは Debug Bridge がスタティック パーティションまたは RP にある場合に使用可能です。
3. BSCAN マスター カウントは 0 よりも大きく、同じ RP 内の Debug Bridge インスタンスまたは MicroBlaze/MDM コアのみ に接続可能です。

次の図は RP の XVC Debug Bridge を使用したデザインを表したものです。

図 13: RP の XVC Debug Bridge を使用した Dynamic Function eXchange デザイン



これは、カウンター RP およびシフター RP という 2 つのリコンフィギュラブル パーティションを持つ Dynamic Function eXchange デザインです。この図は、スタティクおよび RP の両方の領域で使用される Debug Bridge モードをそれぞれ示しています。

デザインのスタティク パーティションには、Debug Bridge IP が 2 つあります。1 つ目の Debug Bridge IP は BSCAN プリミティブ モードで、BSCAN マスター インターフェイスを 3 つ持つように設定されています。この 3 つの BSCAN マスター インターフェイスのうち 2 つは、カウンター RP およびシフター RP パーティションの Debug Bridge インスタンスに接続されていて、デバッグ目的で並列バスを提供しています。3 つ目の BSCAN マスター インターフェイスは、From BSCAN to Debug Hub モードで設定されているスタティク パーティション内の Debug Bridge インスタンスに接続されています。From BSCAN to Debug Hub モードで設定されている Debug Bridge は、デザインのさまざまなデバッグ IP (ILA、VIO、JTAG-to-AXI など) と通信できます。この場合は ILA IP と通信します。

このシステムでは、カウンター RP パーティションには AXI-to-BSCAN モードでインスタンス化されている Debug Bridge が含まれています。この Debug Bridge を XVC モードで使用すると、Debug Bridge は AXI4-Lite インターフェイスを介して XVC コマンドを受信します。この Debug Bridge は、ソフト BSCAN (バウンダリスキャン) インターフェイスを介して、デザインのほかの Debug Bridge インスタンスと通信できます。この Debug Bridge は 2 つの BSCAN マスター インターフェイスを含むように設定されているので、MDM、および From BSCAN to Debug Hub モードで設定されている Debug Bridge インスタンスと通信します。From BSCAN to Debug Hub モードで設定されている Debug Bridge は、デザインのさまざまなデバッグ IP (ILA、VIO、JTAG-to-AXI など) と通信できます。この場合は ILA IP と通信します。

一方、シフター RP パーティションには From BSCAN to Debug Hub モードで設定されている Debug Bridge インスタンスが 1 つだけ含まれていて、デザインのさまざまなデバッグ IP (ILA、VIO、JTAG-to-AXI など) と通信できます。この場合は ILA IP と通信します。

詳細は、『Debug Bridge LogiCORE IP 製品ガイド』 (PG245) を参照してください。

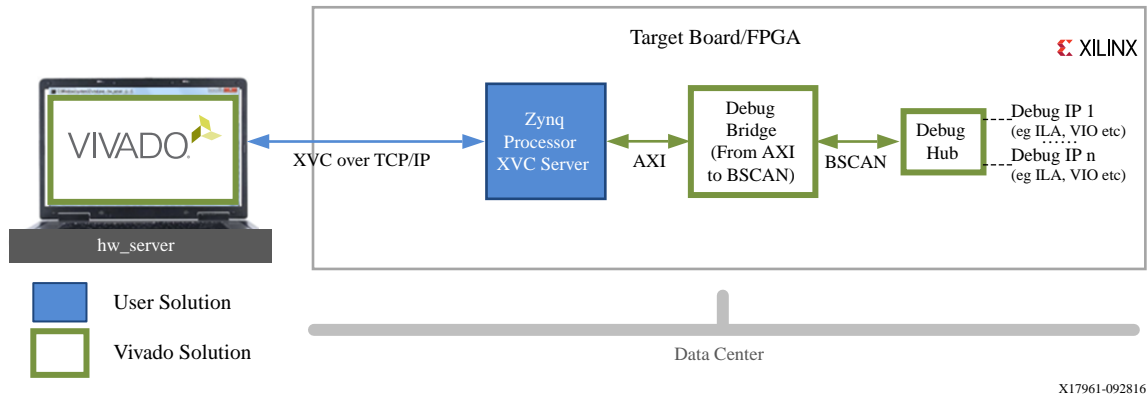
Debug Bridge モードの一部は次の図で説明されています。

From AXI to BSCAN

このブリッジ タイプは、ザイリンクス仮想ケーブル (XVC) を使用するデザインで、JTAG ケーブルを使用せずに、イーサネットなどのインターフェイスを介してリモートで FPGA または SoC デバイスをデバッグすることを目的にしています。このモードでは、Debug Bridge は AXI4-Lite インターフェイスを介して XVC コマンドを受信します。XVC を介して FPGA デバイス上でデザインをデバッグする場合は、このモードを使用します。

詳細は、『Debug Bridge LogiCORE IP 製品ガイド』 (PG245) を参照してください。

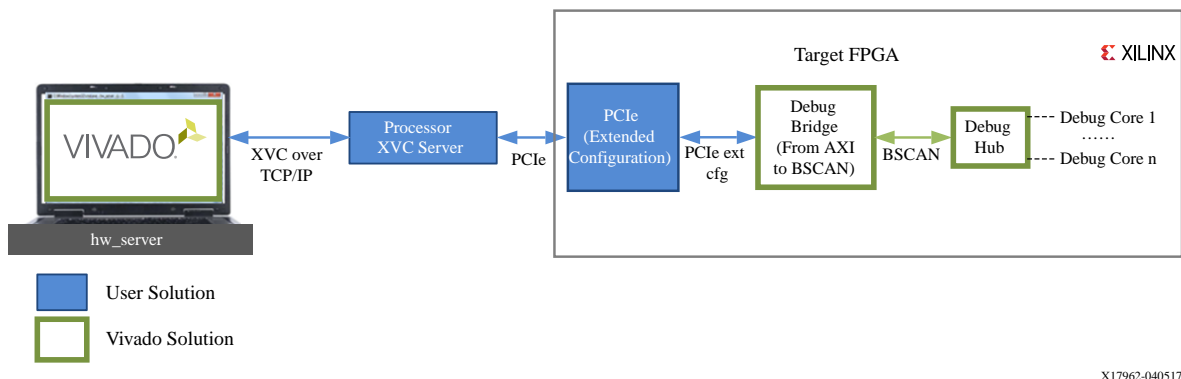
図 14: AXI to BSCAN モードの Debug Bridge



From PCIe to BSCAN

典型的な PCIe 設定では、PCIe to BSCAN モードの Debug Bridge を使用して、デバッグ コアと通信します。このモードでは、Debug Bridge が PCIe IP の拡張コンフィギュレーション インターフェイスに接続されます。ホスト PC への経路に PCIe が JTAG よりも推奨されるデータセンターでよく使用されます。このモードでの PCIe コアと Debug Bridge 使用した XVC フローの詳細およびデザイン例については、『UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP 製品ガイド』 (PG213: [英語版](#)、[日本語版](#)) を参照してください。

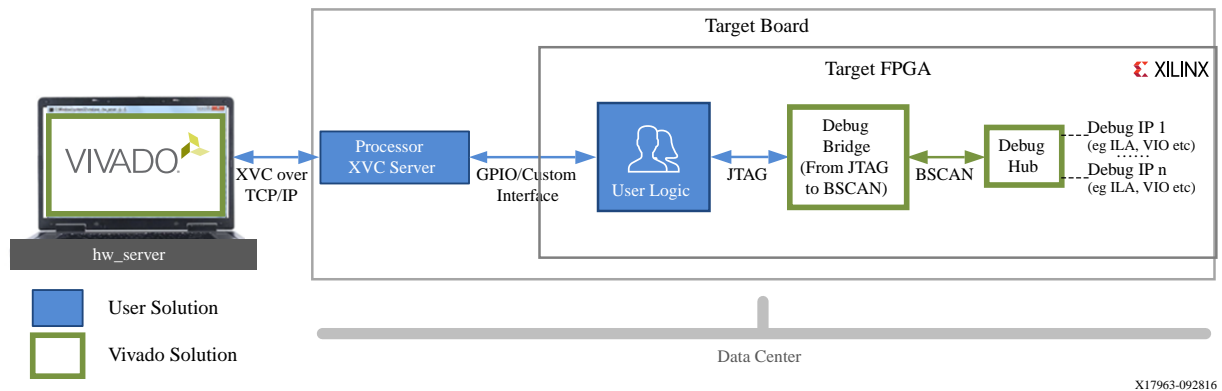
図 15: PCIe 拡張コンフィギュレーション インターフェイスを使用した PCIe to BSCAN モードの Debug Bridge



From JTAG to BSCAN

このブリッジ タイプは、ザイリンクス仮想ケーブル (XVC) を使用するデザインで、JTAG ケーブルを使用せずに、イーサネットなどのインターフェイスを介してリモートで FPGA または SoC デバイスをデバッグすることを目的にしています。このモードでは、Debug Bridge はユーザー ロジックで駆動される JTAG インターフェイスを介して XVC コマンドを受信することになっています。詳細は、『Debug Bridge LogiCORE IP 製品ガイド』 (PG245) を参照してください。

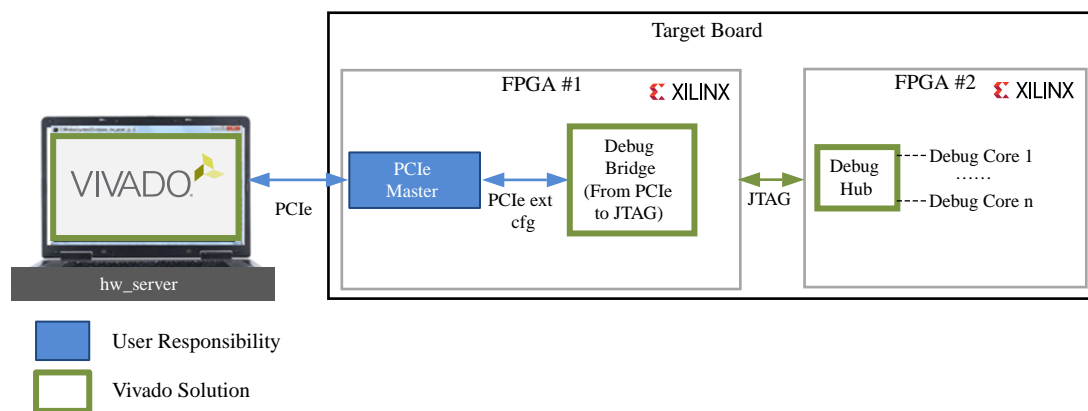
図 16: JTAG to BSCAN モードの Debug Bridge



From PCIe to JTAG

PCIe 設定では、Debug Bridge を PCIe to JTAG モードで使用して、デバッグ コアと通信できます。このモードでは、Debug Bridge が PCIe® IP の拡張コンフィギュレーション インターフェイスに接続され、別のターゲット FPGA のデバッグ ハブに JTAG を使用して通信できるようになります。

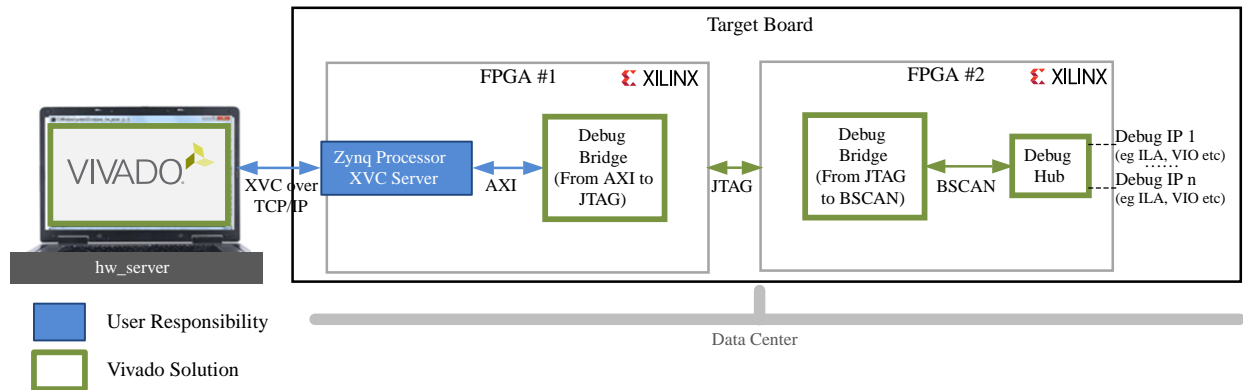
図 17: PCIe® 拡張コンフィギュレーション インターフェイスを使用した PCIe to JTAG モードの Debug Bridge



From AXI to JTAG

このブリッジ タイプは、XVC (Xilinx Virtual cable) を使用したデザインで、イーサネットなどのインターフェイスを介してリモートで FPGA または SoC デバイスをデバッグすることを目的にしています。このモードでは、Debug Bridge は、AXI4-Lite を介して XVC コマンドを受信し、ターゲット デバイスへ JTAG ピンを介して送信します。詳細は、『Debug Bridge LogiCORE IP 製品ガイド』 (PG245) を参照してください。

図 18: AXI to JTAG モードの Debug Bridge



X17966-092816

XVC サーバー インプリメンテーション

XVC プロトコルをインプリメントして、適切なプロセッサに XVC サーバーを作成する必要があります。

XVC プロトコル

XVC プロトコルを使用すると、イーサネットを介して Vivado IDE から JTAG コマンドをエンベデッドシステムに対して実行でき、ターゲット ザイリンクス デバイスがプログラムまたはデバッグできるようになります。これにより、使用ベンダーに関係なくザイリンクス デバイスをデバッグおよびプログラムできます。プログラム機能は従来の JTAG 接続と同じ機能がサポートされ、デバッグ機能には、ザイリンクス システム デバッガー (XSDB) または Vivado ハードウェア デバッグ IP を使用した操作が可能です。

デバイスへの JTAG コマンドは、プログラム ケーブルを使用して通信した場合または Digilent モジュールを使用した場合にデバイスへ転送されるのと同じコマンドです。このコマンドにより、すべての既存の Vivado ハードウェア デバッグ ツール間で機能が使用できます。

ユーザー XVC 1.0 コマンド

次の表は、XVC1.0 プロトコルをまとめたものです。

表 2: XVC コマンドの説明

コマンド	説明
getinfo	<p>コマンド形式:</p> <pre>getinfo: XCV サービス バージョンを取得します。getinfo を受信すると、サービスから次の文字列が返されます。 xvcServer_v1.0:<xvc_vector_len>\n <xvc_vector_len> は、サービスにシフト可能なベクターの最大幅です。</pre>

表 2: XVC コマンドの説明 (続き)

コマンド	説明
shift	<p>コマンド形式:</p> <pre>shift:[num bits][tms vector][tdi vector]</pre> <p>num_bits および tdi_vecto のバイト ベクターを使用して tms_vector でシフトを実行します。</p> <p>num_bits はリトル エンディアン モードの整数です。</p> <p>これは、ベクターをシフトアウトするのに必要な TCK clk トグルの数を示しています。</p> <p>tms_vector は、すべての TMS シフト ビットを使用したバイト サイズのベクターです。</p> <p>このベクターのバイト 0 のビット 0 が最初にシフトアウトされます。</p> <p>ベクターは num_bits で、最も近いバイトに丸められます。</p> <p>tdi_vector は tms_vector と似ていますが、すべての tdi ベクターがシフトインされることを示しています。</p> <p>このコマンドは、シフトインされるビットごとにサンプリングされる該当する tdo ビットを使用して、tms_vector と同じサイズのバイト ベクターを返します。</p> <p>このベクターのバイト 0 のビット 0 は、シフトから読み出される最初の tdo 値です。</p>
settick	<p>コマンド形式:</p> <pre>settick:[period]</pre> <p>サービス tck 周期を [period] に設定しようとしています。</p> <p>[period] は ns で指定されます。</p> <p>これはリトル エンディアンの整数値です。</p> <p>このコマンドは、settick: が終了したときに適用された周期を返します。</p> <p>返された値の単位は ns です。</p> <p>これはリトル エンディアンの整数値です。</p>

Vivado IDE の hw_server の初期化

Vivado IDE の hw_server (ハードウェア サーバー) が XVC 接続を使用して初期化されると、Vivado IDE で USB ケーブルと同じように XVC ケーブルが認識されます。これには、次の引数を使用して Vivado IDE の hw_server を開始します。

```
hw_server -e "set auto-open-servers xilinx-xvc:localhost:10200"
```

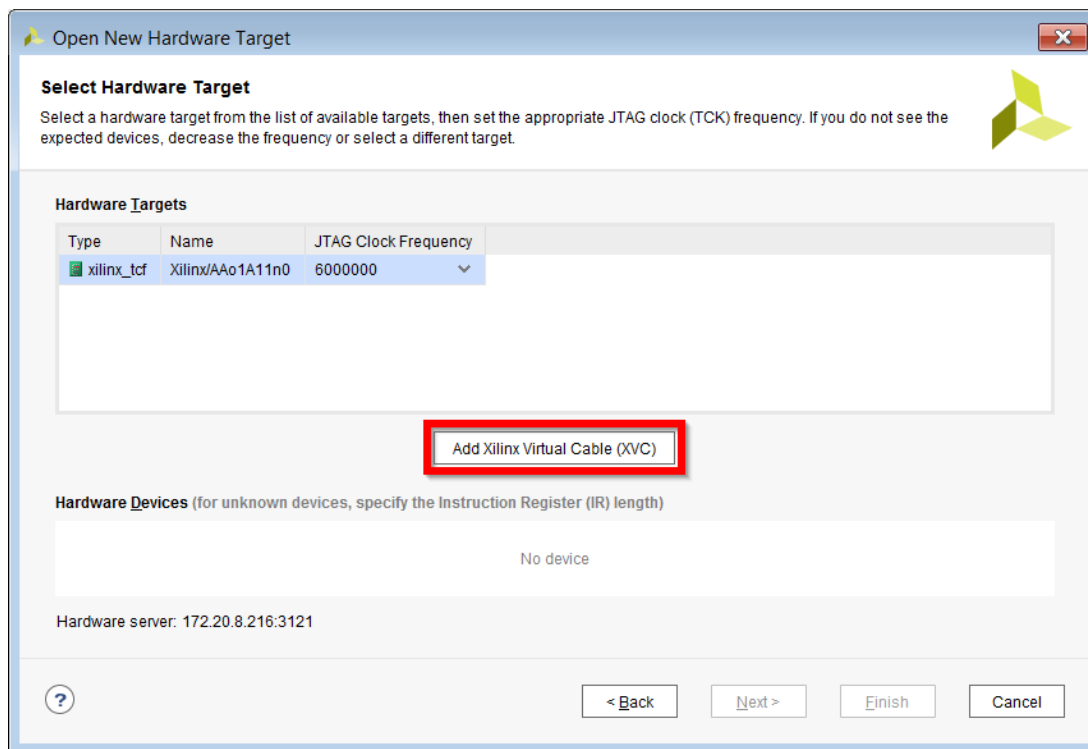
auto-open-servers オプションは、起動時に hw_server で初期化されるように XVC ケーブルをイネーブルにします。既存の XVC ケーブルへの接続が使用されるように、ハードウェア サーバーを初期化できます。サーバーでは、今後の接続で XVC ケーブルが自動的に認識されるようになります。

auto-open-servers への引数は、次のようになります。

```
xilinx-xvc:<xvc_host_name>:<xvc_port>
```

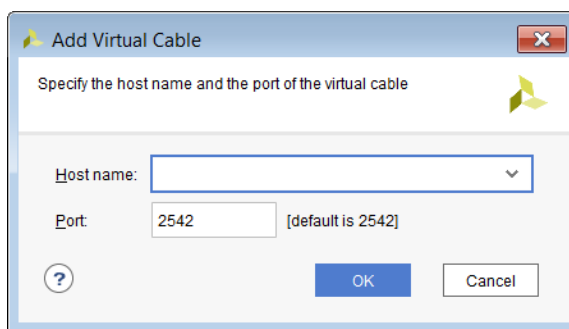
複数のサーバーを指定するには、カンマで区切ります。ハードウェア サーバーは開始すると、指定した XVC サーバーへの接続を構築しようとしています。または、Vivado ハードウェア マネージャーの Open New Hardware Target ウィザードを使用してターゲットに接続する場合は、次の図のように XVC サーバーを設定します。

図 19: Open New Hardware Target ウィザードのダイアログ ボックス



[Add Xilinx Virtual Cable] ボタンをクリックします。次の図に示す [Add Virtual Cable] ダイアログ ボックスが開きます。

図 20: [Add Virtual Cable] ダイアログ ボックス



ここで接続する XVC のホスト名およびポート番号を指定します。

この例は、『PetaLinux ツールを使用した Zynq-7000 でのザイリンクス仮想ケーブルの実行』(XAPP1251) を参照してください。

このアプリケーション ノートは、PetaLinux ツールで生成された Linux オペレーティング システムを使用して Zynq®-7000 デバイスでザイリンクス仮想ケーブル (XVC) を実行する方法を示しています。使用されている参照デザインは、Avnet MicroZed ボード用です。このアプリケーション ノートのターゲット デバイスは AC701 で、Linux で XVC を実行する MicroZed ボードでプログラムおよびデバッグされます。

注記: TCP/IP を介した XVC サーバー インプリメンテーションの例に関しては、<https://github.com/Xilinx/XilinxVirtualCable> を参照してください。

コンフィギュレーション メモリ デバイスのプログラム

Vivado® デバイス プログラム機能を使用すると、ザイリンクス FPGA デバイスを JTAG を介して直接プログラムできます。Vivado では、特定のフラッシュ ベースのコンフィギュレーション メモリ デバイスを JTAG を介して間接的にプログラムすることもできます。これには、まず JTAG とフラッシュ デバイス インターフェイスの間にデータパスを提供する特別なコンフィギュレーションを使用してザイリンクス FPGA デバイスをプログラムし、そのデータパスを使用してコンフィギュレーション メモリ デバイスの内容をプログラムします。

Vivado デバイス コンフィギュレーション機能を使用すると、ザイリンクス FPGA またはメモリ デバイスをザイリンクス ケーブルまたは Digilent ケーブルを使用して直接プログラムできます。適切なケーブルのリストについては、「hw_server を使用したハードウェア ターゲットへの接続」を参照してください。バウンダリスキャン モードで実行すると、ザイリンクス FPGA およびコンフィギュレーション メモリ デバイスをコンフィギュレーションまたはプログラムできます。

Vivado でサポートされるコンフィギュレーション メモリ デバイスのリストは、付録 D を参照してください。

Vivado で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、このデバイスから起動するには、次の手順に従います。

1. コンフィギュレーション メモリ デバイスで使用するビットストリームを生成します。
2. コンフィギュレーション メモリ ファイル (.mcs または .bin) を作成します。
3. Vivado でハードウェア ターゲットに接続します。
4. コンフィギュレーション メモリ デバイスを追加します。
5. Vivado IDE を使用して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。
6. FPGA デバイスを起動します (オプション)。

関連情報

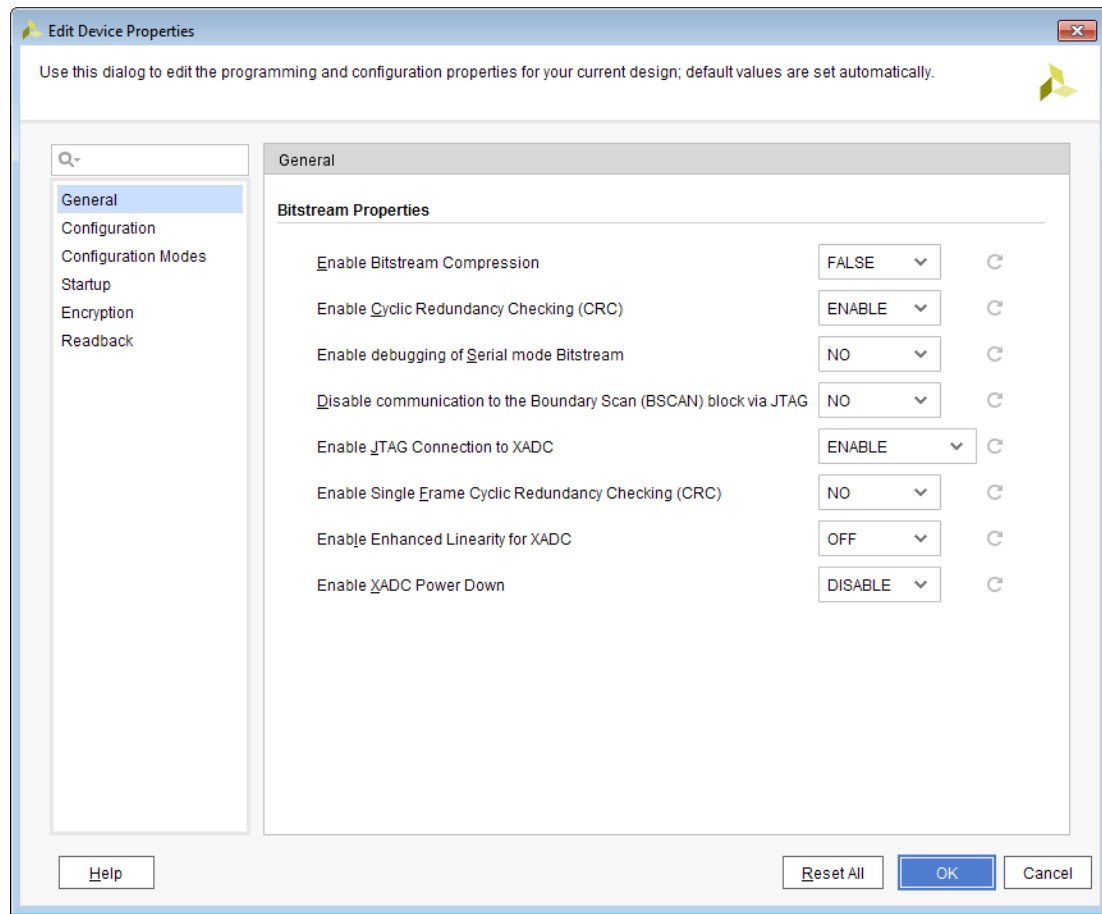
[hw_server を使用したハードウェア ターゲットへの接続](#)
[コンフィギュレーション メモリのサポート](#)

コンフィギュレーション メモリ デバイスで使用するビットストリームの生成

合成済みデザインまたはインプリメント済みデザインで [Tools] → [Edit Device Properties] をクリックし、[Edit Device Properties] ダイアログ ボックスを開きます。

合成済みデザインまたはインプリメント済みデザインで、Flow Navigator の [Settings] → [Bitstream] をクリックし、[Configure additional bitstream] リンクをクリックして [Edit Device Properties] ダイアログ ボックスを開きます。

図 21: [Edit Device Properties] ダイアログ ボックス: 検索フィールド



ダイアログ ボックスの左上にある検索フィールドを使用して、すべての SPI または BPI に関連するフィールドを検索し、適切なオプションを設定します。デバイス コンフィギュレーション 設定の詳細は、付録 A を参照してください。

関連情報

[デバイス コンフィギュレーション ビットストリーム設定](#)

コンフィギュレーション メモリ ファイルの作成

`write_cfgmem` Tcl コマンドを使用して `.mcs` または `.bin` プログラム ファイルを作成します。このファイルは、コンフィギュレーション メモリ デバイスのプログラムに使用します。

たとえば、1 つの 1G ビット BPI コンフィギュレーション メモリ デバイスの `.mcs` ファイルを生成するには、次のコマンドを使用します。

```
write_cfgmem -format mcs -interface bpix16 -size 128 \
  -loadbit "up 0x0 design.bit" -file design.mcs
```

注記: `write_cfgmem` コマンドの `-size` オプションはメガバイトで指定するので、単位がメガビットのフラッシュデバイスの容量とは異なります。そのため上記の `write_cfgmem` コマンドの例では、1 G ビットのフラッシュデバイスのサイズが 128 MB と指定されています。`write_cfgmem` コマンドでは、コンフィギュレーション メモリ ファイルのサイズが自動的にビットストリームのサイズに設定されます。

Vivado IDE では、`write_cfgmem` コマンドを使用して複数の `.bit` ファイルをチェーン接続できます。複数のビットストリームを含む 1 つの 1 G ビット BPI コンフィギュレーション メモリ デバイスの `.mcs` ファイルを生成するには、次のコマンドを使用します。

```
write_cfgmem -format mcs -interface bpix16 -size 128 \
  -loadbit "up 0 design1.bit up 0xFFFFF design2.bit" \
  -file design1_design2.mcs
```

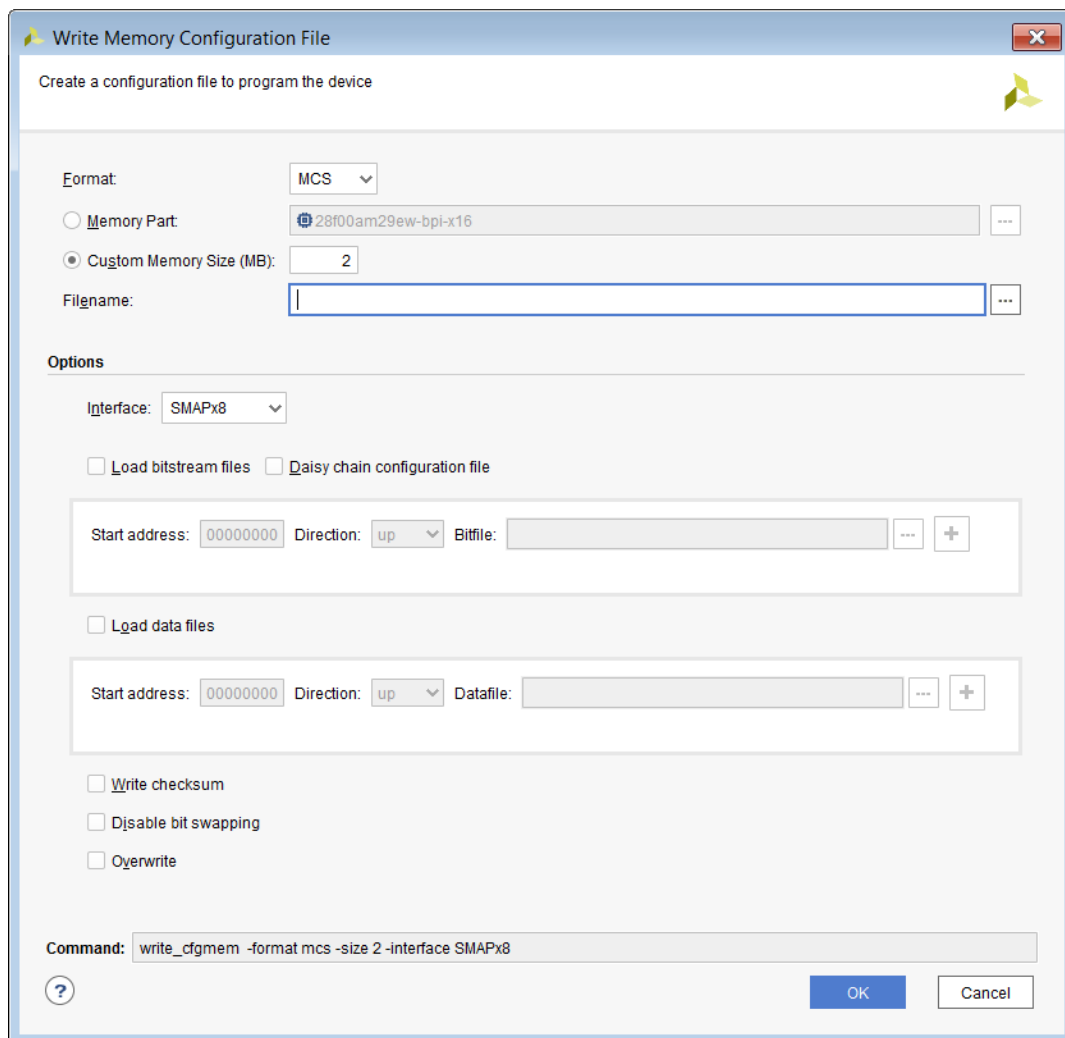
`write_cfgmem` コマンドの詳細は、『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835)を参照してください。



ヒント: Vivado Lab Edition でコンフィギュレーション メモリ ファイルを作成できます。

コンフィギュレーション メモリ ファイルは、Vivado IDE でも作成できます。[Tools]→[Generate Memory Configuration File] をクリックします。次の図に示す [Write Memory Configuration File] ダイアログ ボックスが表示されます。

図 22: [Write Memory Configuration File] ダイアログ ボックス



適切なフォーマットおよびオプションを選択し、[OK] をクリックしてコンフィギュレーション メモリ ファイルを生成します。

SPI デュアル クワッド (x8) デバイス用のコンフィギュレーション メモリの作成

`write_cfgmem` Tcl コマンドを使用すると、デュアル クワッド SPI (x8) デバイス用の `.mcs` イメージが生成されます。このコマンドでは、コンフィギュレーション データが 2 つの別々の `.mcs` ファイルに自動的に分割されます。

注記: SPIx8 用の `.mcs` を生成する際に指定するサイズは、2 つのクワッド フラッシュ デバイスの合計サイズです。

注記: `write_cfgmem` Tcl コマンドは、デュアル クワッド SPI (x8) モード用に `.mcs` ファイルを作成する際に、開始アドレスを 2 で割ります。

write_cfgmem の使用例

この例は、アドレス 0 でロードされるゴールデン ビットストリームとアドレス 0x0100_0000 でロードされるマルチブート ビットストリーム用に .mcs ファイルを生成する方法を示しています。

デバイス: 2x 256 Mib Quad SPI Flash デバイス: 256 Mib = 32 MiB

合計ストレージ サイズ: 2 * 32 MiB = 64 MiB

ロードアドレス:

ゴールデン: 0 * 2 = 0

マルチブート: 0x0100_0000 * 2 = 0x0200_0000

```
write_cfgmem -format mcs -interface spix8 -size 32 \
-loadbit "up 0 ./design1_spix8.bit up 0x02000000 ./design2_spix8.bit" \
-file design1_design2_spix8.mcs
```

Vivado でハードウェア ターゲットに接続

Vivado でハードウェア ターゲットに接続するには、次の手順に従います。

1. ハードウェア ターゲットの FPGA モード ピンで、コンフィギュレーション メモリ デバイスから FPGA をコンフィギュレーションするのに適切なコンフィギュレーション モード (マスター SPI またはマスター BPI) が選択されていることを確認します。
詳細は、ターゲット デバイスのコンフィギュレーション ユーザー ガイドを参照してください。
2. 「FPGA デバイスのプログラム」の手順に従って、ハードウェア ターゲットに接続します。



重要: ボードの電源がオフになった場合、またはケーブルの接続が解除された場合、Vivado IDE によりハードウェア ターゲットが閉じられます。メインの Vivado スレッドに対する Vivado の操作もキャンセルされます。

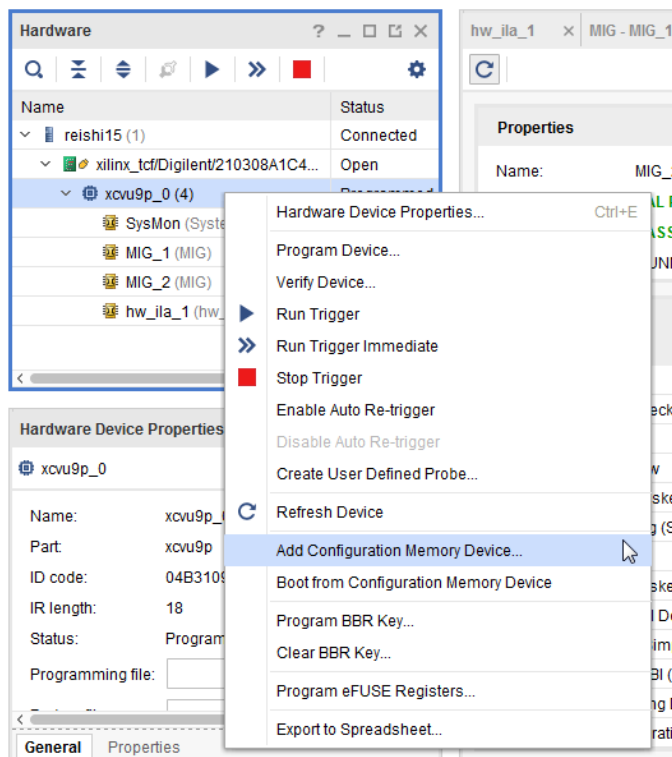
関連情報

[デバイスのプログラム](#)

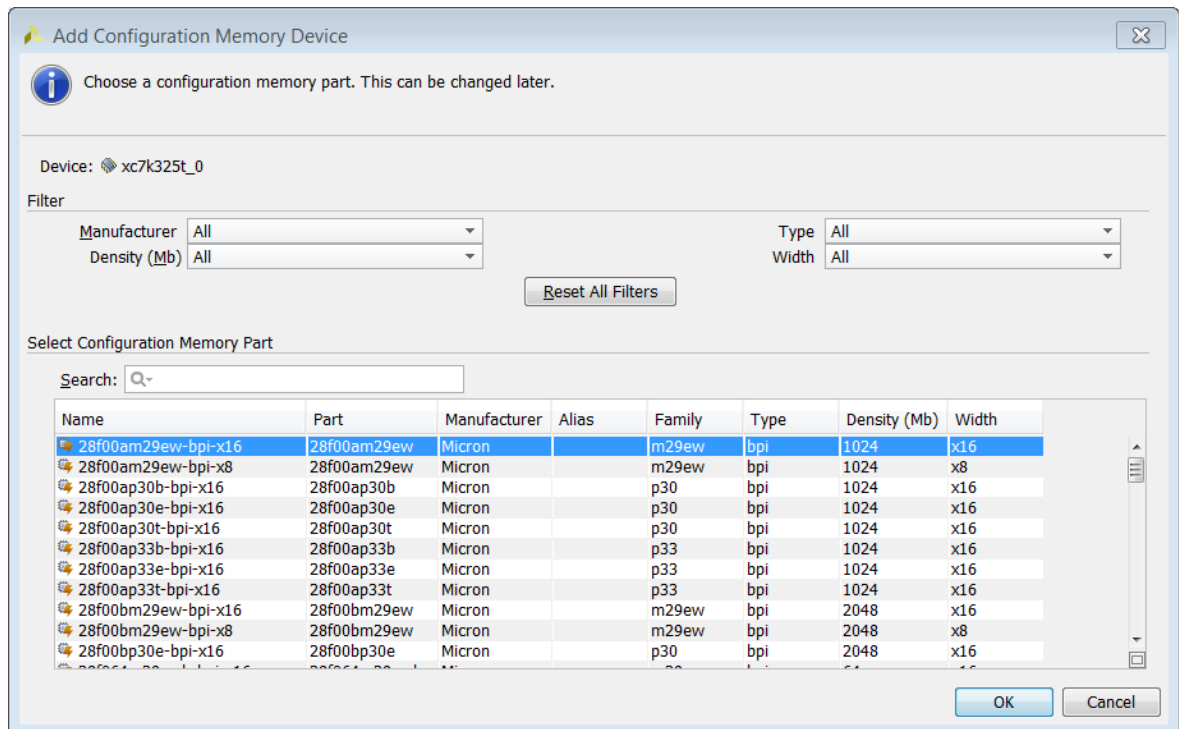
コンフィギュレーション メモリ デバイスの追加

Vivado デバイス プログラマでハードウェア ターゲットにコンフィギュレーション メモリ デバイスを追加するには、次の手順に従います。

1. 前述の手順に従ってハードウェア ターゲットに接続した後、次の図に示すようにハードウェア ターゲットを右クリックして [Add Configuration Memory Device] をクリックします。



このメニューをクリックすると、[Add Configuration Memory Device] ダイアログ ボックスが開きます。

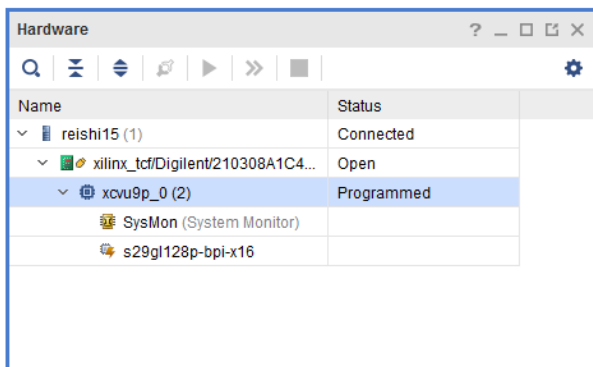


- 適切なコンフィギュレーション メモリ デバイスを選択し、[OK] をクリックします。



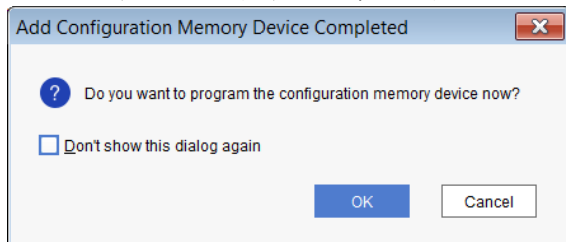
ヒント: [Search] フィールドを使用して、ベンダー、集積度、タイプなどで絞り込みます。

これでハードウェア ターゲット デバイスにコンフィギュレーション メモリ デバイスが追加されました。

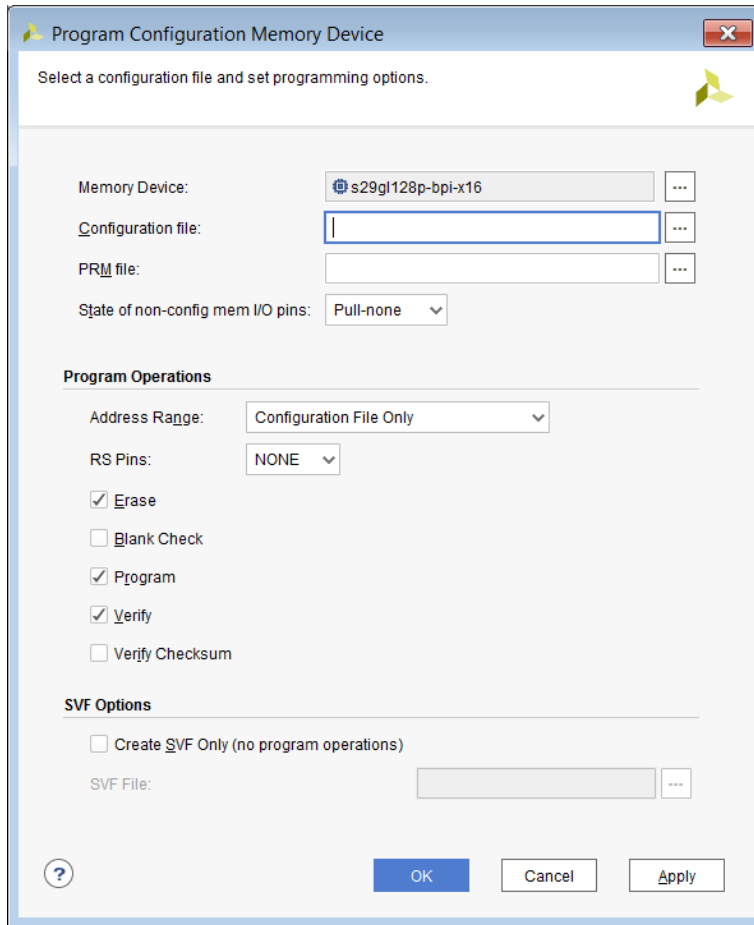


コンフィギュレーション メモリ デバイスのプログラム

1. コンフィギュレーション メモリ デバイスを作成すると、Vivado デバイス プログラマからコンフィギュレーション メモリ デバイスをすぐにプログラムするかどうかを確認するメッセージが表示されます。



[OK] をクリックすると、[Program Configuration Memory Device] ダイアログ ボックスが開きます。



2. このダイアログ ボックスのすべてのオプションを適切に設定します。

- [Configuration file]: コンフィギュレーション メモリ デバイスをプログラムするのに使用する .mcs または .bin ファイルを指定します。メモリ コンフィギュレーション ファイルは、write_cfgmem Tcl コマンドで作成します。詳細は、「コンフィギュレーション メモリ ファイルの作成」を参照してください。
- [State of non-config mem I/O pins]:
 - [Pull-none]: FPGA にプログラムされた間接コンフィギュレーション ビットストリームにある未使用 I/O ピンを [pull-none] に設定します。
 - [Pull-up]: FPGA にプログラムされた間接コンフィギュレーション ビットストリームにある未使用 I/O ピンを [pull-up] に設定します。
 - [Pull-down]: FPGA にプログラムされた間接コンフィギュレーション ビットストリームにある未使用 I/O ピンを [pull-down] に設定します。



重要: 非コンフィギュレーション メモリ I/O ピンの状態が write_bitstream プロパティで設定したものと同じようにしてください。このプロパティのデフォルト値は pull-down です。

- [Program Operations]: コンフィギュレーション メモリ デバイスで実行するプログラム操作を設定します。
 - [Address Range]: プログラムするコンフィギュレーション メモリ デバイスのアドレス範囲を指定します。有効なアドレス範囲の値は、次のとおりです。
 - [Configuration File Only]: 消去、ブランク チェック、プログラム、および検証にメモリ コンフィギュレーション ファイルに必要なアドレス空間のみを使用します。

- [Entire Configuration Memory Device]: デバイス全体を消去、ブランク チェック、プログラム、および検証します。
- [RS Pins] (オプション): BPI コンフィギュレーション メモリ デバイスのみで使用されるリビジョン セレクト ピンのマッピングを指定します。フラッシュの上位 2 つの FPGA アドレス ピンが FPGA RS[1:0] に接続されます。このオプションをイネーブルにすると、プログラム用に FPGA RS[1:0] が駆動されます。アプリケーションの使用法は、該当する FPGA コンフィギュレーション ユーザー ガイドを参照してください。
- [Erase]: コンフィギュレーション メモリ デバイスの内容を消去します。
- [Blank Check]: プログラムの前にコンフィギュレーション メモリ デバイ스에 데이터가 없는 것을 확인합니다.
- [Program]: 콘피규레이션 메모리 디바이스를 지정의 콘피규레이션 파일 (.mcs 또는 .bin) で 프로그램합니다.
- [Verify]: 프로그램 후에 콘피규레이션 메모리 디바이스의 내용이 콘피규레이션 파일 (.mcs 또는 .bin) に一致することを 검증합니다.
- [Verify Checksum]: 콘피규레이션 메모리 디바이스에 프로그램된 데이터를 검증합니다. 콘피규레이션 메모리 디바이스에 프로그램된 데이터에 기반하여 체크섬이 계산되며, 그 값이 .prm 파일에서 지정されている 체크섬 값と比較されます.



ヒント: ユーザーが `write_cftmem` コマンドに `-checksum write_cftmem` オプションを指定して `cfgmem` ファイルを生成します。この手順により、`cfgmem` 出力ファイルに関するチェックサム情報を含む `.prm` ファイルが作成されます。

- [Create SVF Only]: 指定したプログラム操作を含む `.svf` ファイルを作成します。サードパーティ ツールで `.svf` ファイルを使用して、Vivado 外でコンフィギュレーション メモリ デバイ스를 프로그램할 수 있습니다.



重要: このオプションをオンにすると、関連のプログラム オプションを含む `.svf` ファイルが生成されるだけです。コンフィギュレーション メモリ デ바이스는 프로그램되지 않습니다.

3. [OK] をクリックし、コンフィギュレーション メモリ デ바이스에 대해 이 대화 상자에서의 선택에 응じて 삭제, 브ランク 체크, 프로그램,および 검증 작업을 실행합니다. 각 작업이 종료されると、それが通知されます。

注記: [Apply] をクリックするとコンフィギュレーション 메모리 설정이 저장되지만, 콘피규레이션 메모리 디바이스는 프로그램되지 않습니다. [Apply] をクリックした後に [Cancel] をクリックすると、コンフィギュレーション 메모리 디바이스가 설정되며, 후에 프로그램할 수 있습니다.

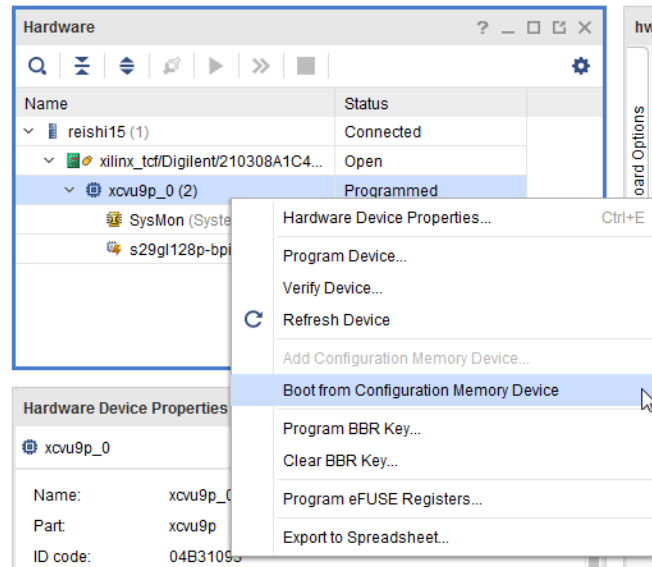
関連情報

[コンフィギュレーション 메모리 파일의作成](#)

デバイスの起動

コンフィギュレーション 메모리 디바이스를 프로그램したら、소프트 부트 (JPROGRAM など) を実行し、接続されている 콘피규레이션 메모리 디바이스에서 FPGA 콘피규레이션을 시작하고, 타겟 FPGA に対して 부트를 실행するには, 타겟 디바이스를 오른쪽 클릭하여 [Boot from Configuration Memory Device] を 클릭합니다.

図 23: コンフィギュレーション メモリ デバイスからのブート



★ **重要:** コンフィギュレーション メモリからのブート後に、システム ブートアップの考慮事項のためにデバッグ コアがすぐに表示されないことがあります。Vivado ハードウェア マネージャーの Tcl コンソールに Tcl コマンド `boot_hw_device` を使用して、ブート前に必要に応じて指定した時間分待つように設定することをお勧めします。

```
boot_hw_device after 1000 [refresh_hw_device]
```

1000 は指定できる最大の `wait_on` 値です。

マスター モードのコンフィギュレーション エラー

コンフィギュレーション エラーは、ボードがマスター BPI またはマスター SPI モードで、JTAG ケーブルが Vivado ハードウェア マネージャーに接続されている場合に発生することがあります。ハードウェア マネージャーのポーリングおよび復元機能のためにマスターモードのコンフィギュレーションができない場合は、電源投入時にコンフィギュレーション エラーが断続的に発生します。この問題を回避するには、Vivado ハードウェア マネージャーの Tcl コンソールで次のパラメーターを設定し、ステータス レジスタが更新されないようにします。

```
set_param xicom.allow_cfgin_commands false
```

注記: このパラメーターは、チェーン内のすべてのデバイスに影響します。

アドバンス プログラム機能

リードバックおよび検証

ビットストリームの検証およびリードバック

Vivado® IDE では、FPGA にダウンロードされるコンフィギュレーション データ (.bit ファイルなど) を検証およびリードバックできます。write_bitstream を使用して .bit ファイルを生成する際、-mask_file オプションを使用して対応するマスク ファイル (.msk) を作成します。ビットストリーム生成オプションの詳細は、Vivado IDE の [Tcl Console] ウィンドウで write_bitstream-help を実行します。

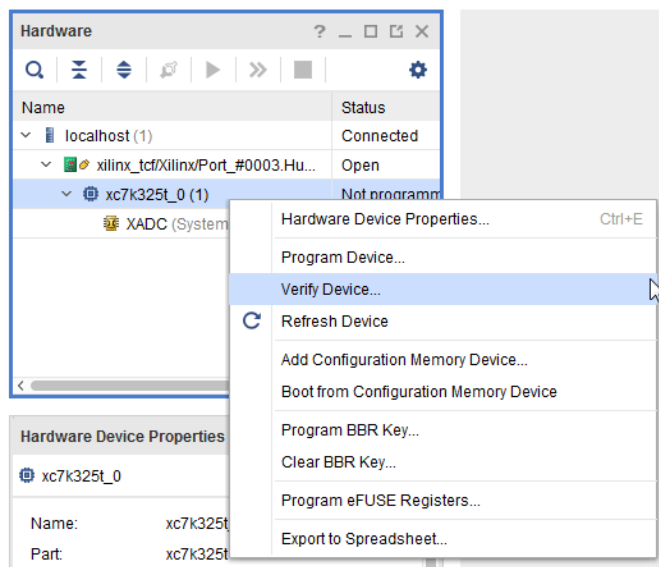
検証を実行するには、verify_hw_devices コマンドで FPGA からデータをリードバックし、.msk を使用して、スキップするリードバック データ ビットと、.bit ファイルの対応するビットと比較するビットを指定します。

次に、ビットストリーム検証の Tcl コマンド シーケンス例を示します。.bit および .msk ファイルは、write_bitstream で既に生成されています。

```
create_hw_bitstream -hw_device [current_hw_device] \  
-mask kcu105_cnt_ila_uncmpr.msk kcu105_cnt_ila_uncmpr.bit  
verify_hw_devices [current_hw_device]
```

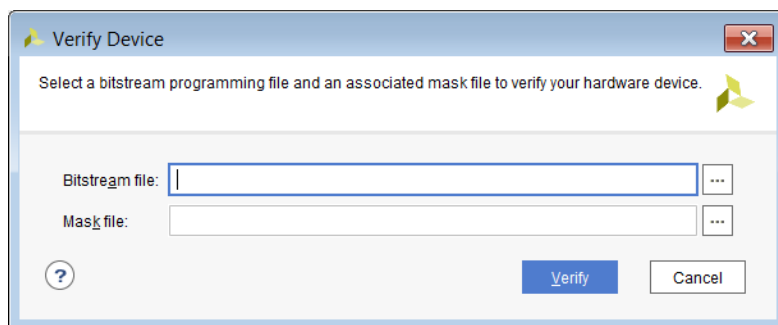
Vivado ハードウェア マネージャーを使用してコンフィギュレーション データを検証できます。次の図に示すように、デバイスを右クリックして [Verify Device] をクリックします。

図 24: [Verify Device] コマンド



このコマンドをクリックすると、[Verify Device] ダイアログ ボックスが開きます。

図 25: [Verify Device] ダイアログ ボックス



BIT ファイルと対応するマスク ファイル (.msk) を指定し、[Verify] をクリックして検証を実行します。

`readback_hw_device` コマンドを少なくとも次のいずれかのオプションを使用して実行し、FPGA コンフィギュレーション データをリードバックします。

- リードバック データを ASCII フォーマットで保存するには、次のオプションを使用します。

```
-readback_file <filename.rbd>
```

- リードバック データをバイナリ フォーマットで保存するには、次のオプションを使用します。

```
-bin_file <filename.bin>
```

例: FPGA コンフィギュレーション データを ASCII とバイナリ形式の両方でリードバックします。

```
readback_hw_device [current_hw_device] \
  -readback_file   kcu105_cnt_ila_uncmpr-rb.rbd \
  -bin_file        kcu105_cnt_ila_uncmpr-rb.bin
```

1. ビットストリームおよびリードバック操作は、[Tcl Console] ウィンドウから実行します。
2. 検証およびリードバック操作は、暗号化されたビットストリームを使用してプログラムされた FPGA には使用できません。暗号化されたビットストリームには、リードバックをディスエーブルするコマンドが含まれます。リードバックは、FPGA PROG ピンにパルスを送るか、または FPGA/ボードがパワーダウンされてから再びパワーアップされると、再びイネーブルになります。
3. `readback_hw_device` を使用したデータ リードバックには、コンフィギュレーション データのみが含まれ、コンフィギュレーション コマンドは含まれません。

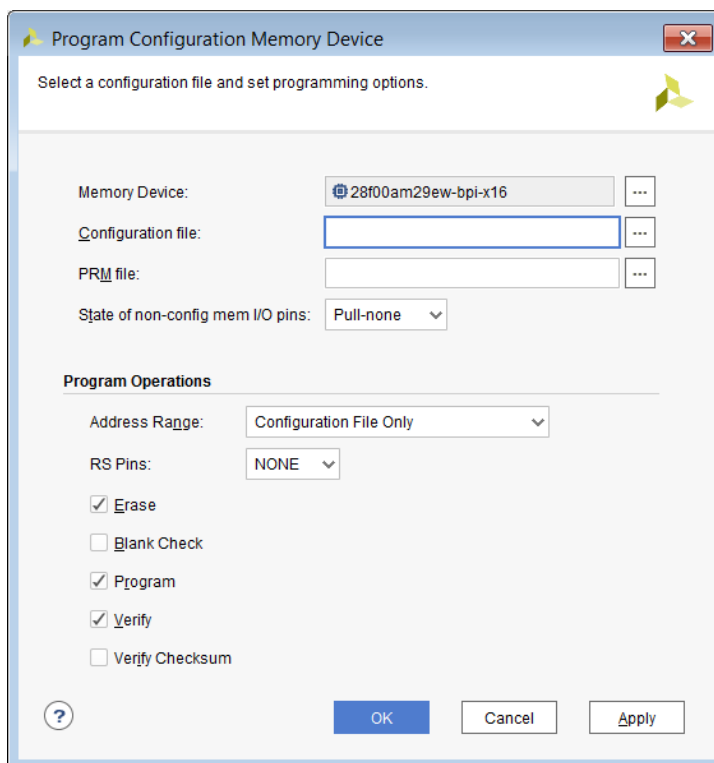
リードバックおよびマスク ファイルの詳細は、『UltraScale アーキテクチャ コンフィギュレーション ユーザー ガイド』 (UG570: [英語版](#)、[日本語版](#)) または 『7 シリーズ FPGA コンフィギュレーション ユーザー ガイド』 (UG470: [英語版](#)、[日本語版](#)) の「リードバック データの検証」セクションを参照してください。

コンフィギュレーション メモリの検証およびリードバック

ビットストリーム ファイル (.bit) を .mcs または .bin ファイルに変換してから、`write_cfgmem` コマンドを使用して、シリアル/SPI またはパラレル/BPI フラッシュなどのコンフィギュレーション メモリ デバイスにプログラムできます。詳細は、『Vivado Design Suite Tcl コマンド リファレンス ガイド』 ([UG835](#)) を参照してください。

Vivado Design Suite のハードウェア マネージャーを使用してコンフィギュレーション メモリ デバイスを検証します。

図 26: コンフィギュレーション メモリの検証



次のコードに示すように、HW_CFGMEM プロパティを設定して `program_hw_cfgmem` を呼び出すことにより、コンフィギュレーション メモリ デバイスを検証することもできます。

```
set_property PROGRAM.ADDRESS_RANGE {use_file} [ get_property
PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.FILES [list "H:/projects/k7_led/
k7_led_325t_afx_x16_33v.mcs" ] \
[ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0]]
set_property PROGRAM.BPI_RS_PINS {none} [ get_property PROGRAM.HW_CFGMEM
[lindex [get_hw_devices] 0 ]]
set_property PROGRAM.UNUSED_PIN_TERMINATION {pull-none} [ get_property \
PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.BLANK_CHECK 0 [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]
set_property PROGRAM.ERASE 0 [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]
set_property PROGRAM.CFG_PROGRAM 0 [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]
set_property PROGRAM.VERIFY 1 [ get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]
startgroup
if {[string equal [get_property PROGRAM.HW_CFGMEM_TYPE [lindex
[get_hw_devices] 0]] [get_property MEM_TYPE
[get_property CFGMEM_PART [get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]]]] } \
{ create_hw_bitstream -hw_device [lindex [get_hw_devices] 0] [get_property
PROGRAM.HW_CFGMEM_BITFILE \
[ lindex [get_hw_devices] 0]]; program_hw_devices [lindex [get_hw_devices]
0]; };
program_hw_cfgmem -hw_cfgmem [get_property PROGRAM.HW_CFGMEM [lindex
[get_hw_devices] 0 ]]
endgroup
```

コンフィギュレーション メモリの内容は、Vivado Design Suite の [Tcl Console] ウィンドウで次のコマンドを使用してリードバックできます。

```
readback_hw_cfgmem -file test.bin -hw_cfgmem \
[get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0]]
```

注記: コンフィギュレーション メモリ リードバックは、[Tcl Console] ウィンドウからのみ実行します。

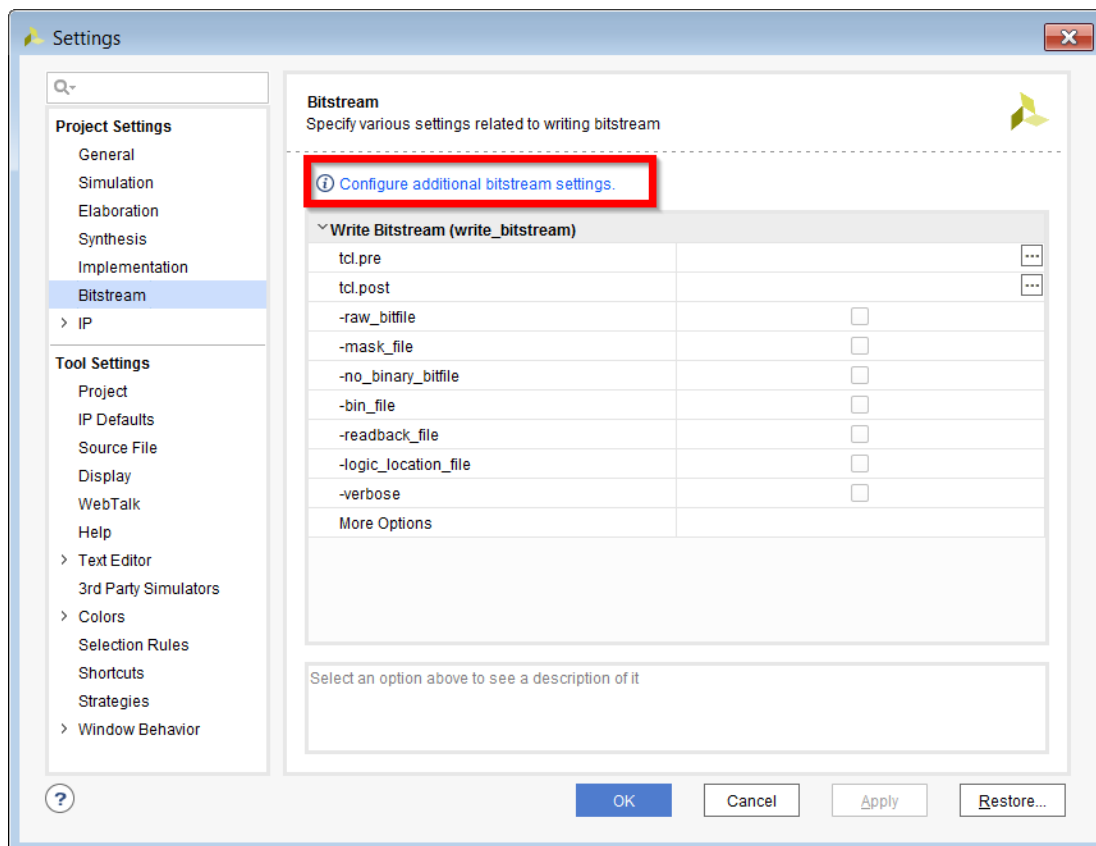
これらの機能の詳細は、『UltraScale アーキテクチャ コンフィギュレーション ユーザー ガイド』 (UG570: [英語版](#)、[日本語版](#)) または『7 シリーズ FPGA コンフィギュレーション ユーザー ガイド』 (UG470: [英語版](#)、[日本語版](#)) を参照してください。

7 シリーズ デバイス用の暗号化および認証されたファイルの生成

注記: 詳細は、『暗号化を使用して 7 シリーズ FPGA のビットストリームを保護』 (XAPP1239: [英語版](#)、[日本語版](#)) を参照してください。

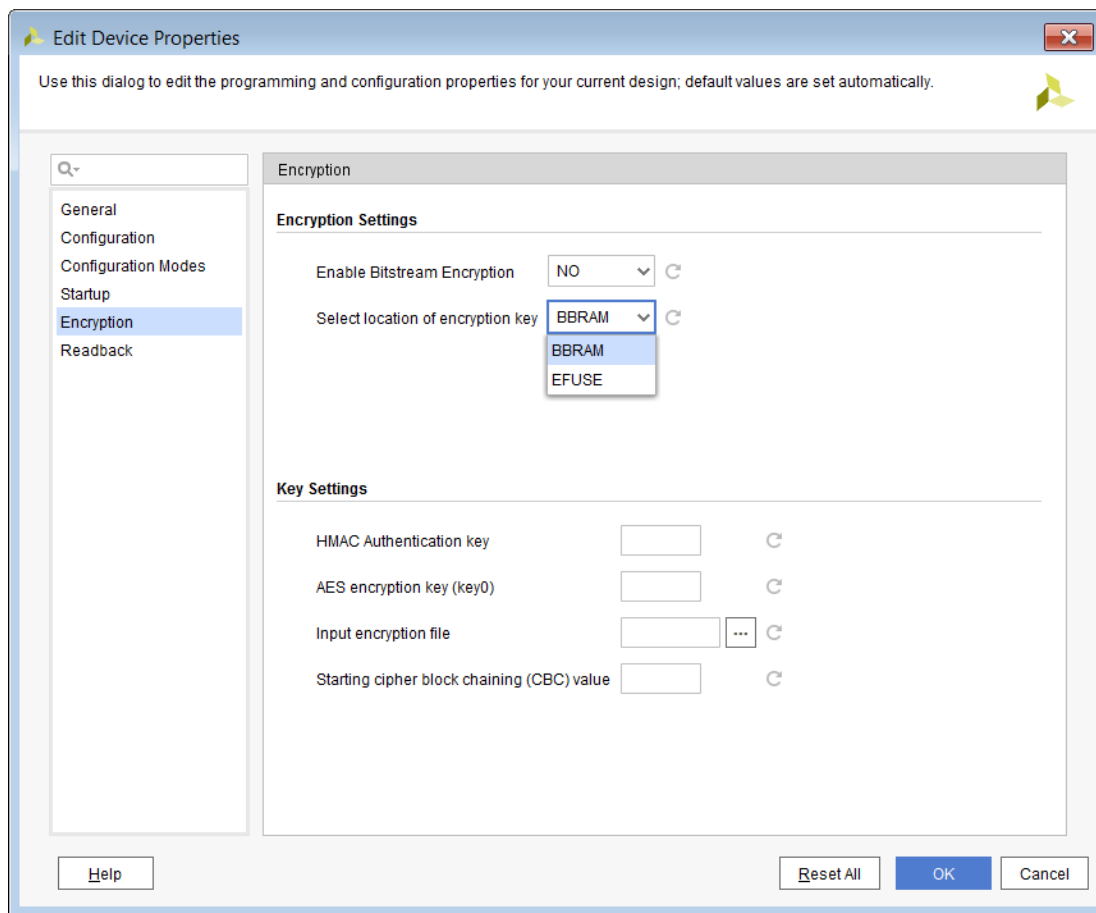
暗号化ビットストリームを生成するには、Vivado IDE でインプリメント済みデザインを開きます。メイン メニューから [Flow] → [Bitstream Settings] をクリックして [Settings] ダイアログ ボックスを開きます。ダイアログ ボックスの上部の [Configure Additional Bitstream Settings] リンクをクリックします。

図 27: 7 シリーズの設定



[Edit Device Properties] ダイアログ ボックスが開きます。左側のペインで [Encryption] をクリックします。

図 28: 7 シリーズ デバイスの暗号化設定



[Edit Device Properties] ダイアログ ボックスで、暗号化設定およびキー設定を指定します。

- [Encryption Settings]
 - [Enable Bitstream Encryption] を [YES] に設定します。
 - [Select location of encryption key] を [BBRAM] または [EFUSE] のいずれかに設定します。
 - キー ロケーションは暗号化されたビットストリームに埋め込まれます。
 - 暗号化されたビットストリームをデバイスにダウンロードすると、FPGA で BBR または eFUSE キー レジスタに読み込まれたキーが使用され、暗号化されたビットストリームが復号化されます。
- [Key Settings]
 - [HMAC Authentication key] および [Starting cipher block chaining (CBC) value] を指定します。
 - これらの値を指定しない場合、Vivado でランダムな値が生成されます。
 - これらの値は、暗号化されたビットストリームに埋め込まれるので、FPGA にプログラムする必要はありません。

注記: これらの値は、入力暗号化ファイルが指定されている場合を除き、現在のプロジェクトの制約ファイルに保存されます。この値が制約ファイルに保存されないようにするには、入力暗号化ファイルを指定してください。

- 。 [AES encryption key] でビットストリームを暗号化する際に使用する AES 暗号キーを指定します。64 個までの 16 進数文字を使用して、256 ビット キーを指定できます。
- キーは、拡張子が `.nky` のファイルに書き込まれます。キーを BBR に読み込む際、またはキーを eFUSE キー レジスタにプログラムする際に、このファイルを使用します。

注記: これらの値は、入力暗号化ファイルが指定されている場合を除き、現在のプロジェクトの制約ファイルに保存されます。この値が制約ファイルに保存されないようにするには、入力暗号化ファイルを指定してください。

- 。 [Input encryption file] で入力暗号化ファイルを指定します。
- 既存の `.nky` ファイルを指定して、暗号キー設定を取得します。このフィールドはオプションで、AES、HMAC、および CBC を手動で指定する場合は入力する必要はありません。

暗号化設定を指定したら、[OK] をクリックしてプロジェクトに設定を適用してビットストリームを再生成します。`write_bitstream` コマンドの処理が正常に完了すると、プログラム ファイルと暗号化ファイル (`.nky`) が生成されます。

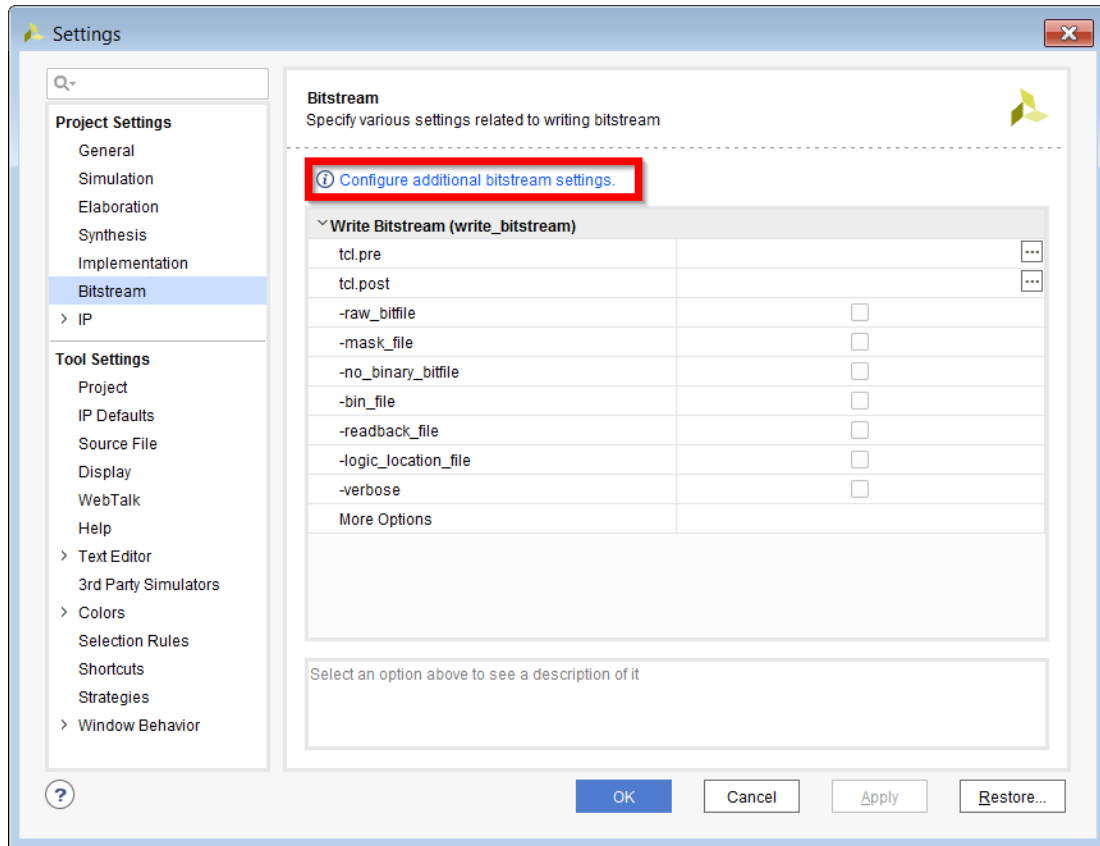
UltraScale および UltraScale+ デバイス用の暗号化および認証されたファイルの生成

注記: 詳細は、次を参照してください

『暗号化と認証を使用して UltraScale/UltraScale+ FPGA のビットストリームを保護』 (XAPP1267: [英語版](#)、[日本語版](#))

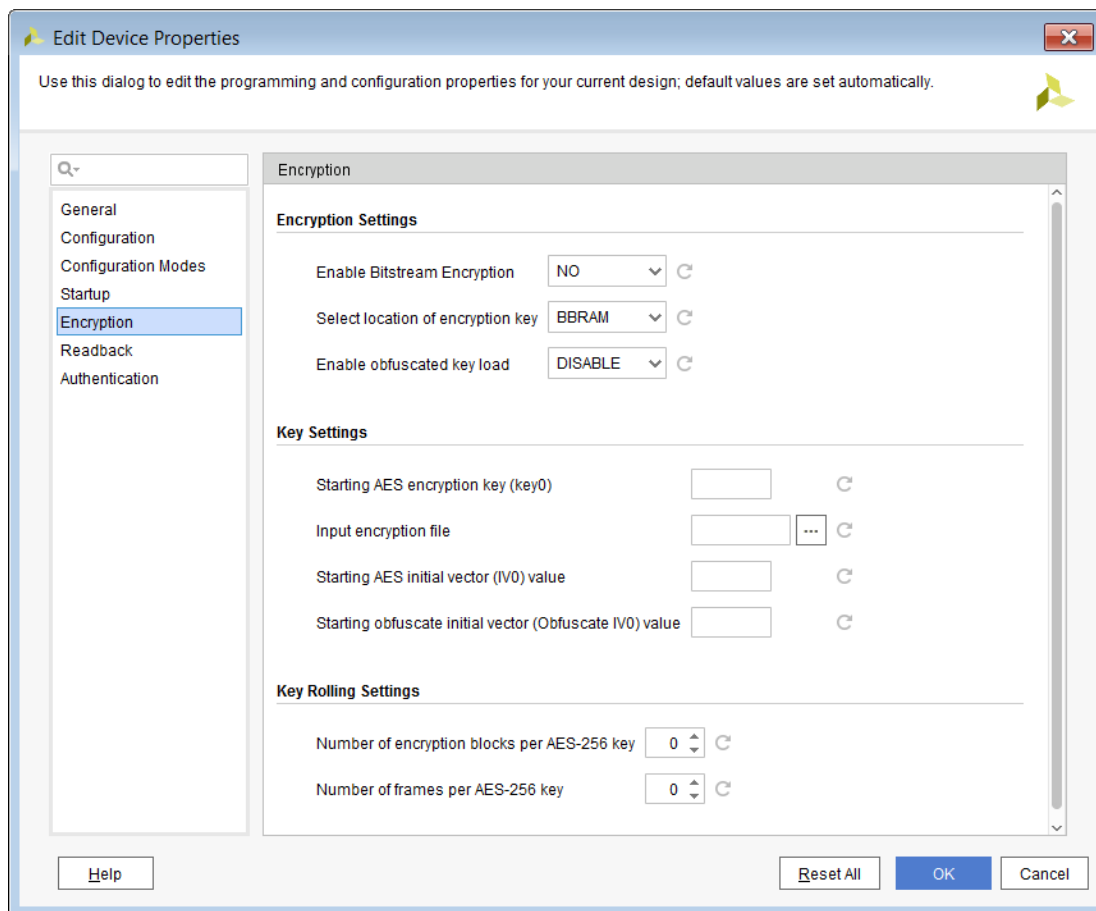
暗号化ビットストリームを生成するには、Vivado IDE でインプリメント済みデザインを開きます。メイン メニューから [Flow] → [Bitstream Settings] をクリックして [Settings] ダイアログ ボックスを開きます。ダイアログ ボックスの上部の [Configure Additional Bitstream Settings] リンクをクリックします。

図 29: UltraScale および UltraScale+ デバイスのビットストリーム設定



[Edit Device Properties] ダイアログ ボックスが開きます。左側のペインで [Encryption] をクリックします。

図 30: UltraScale コンフィギュレーションの暗号化設定



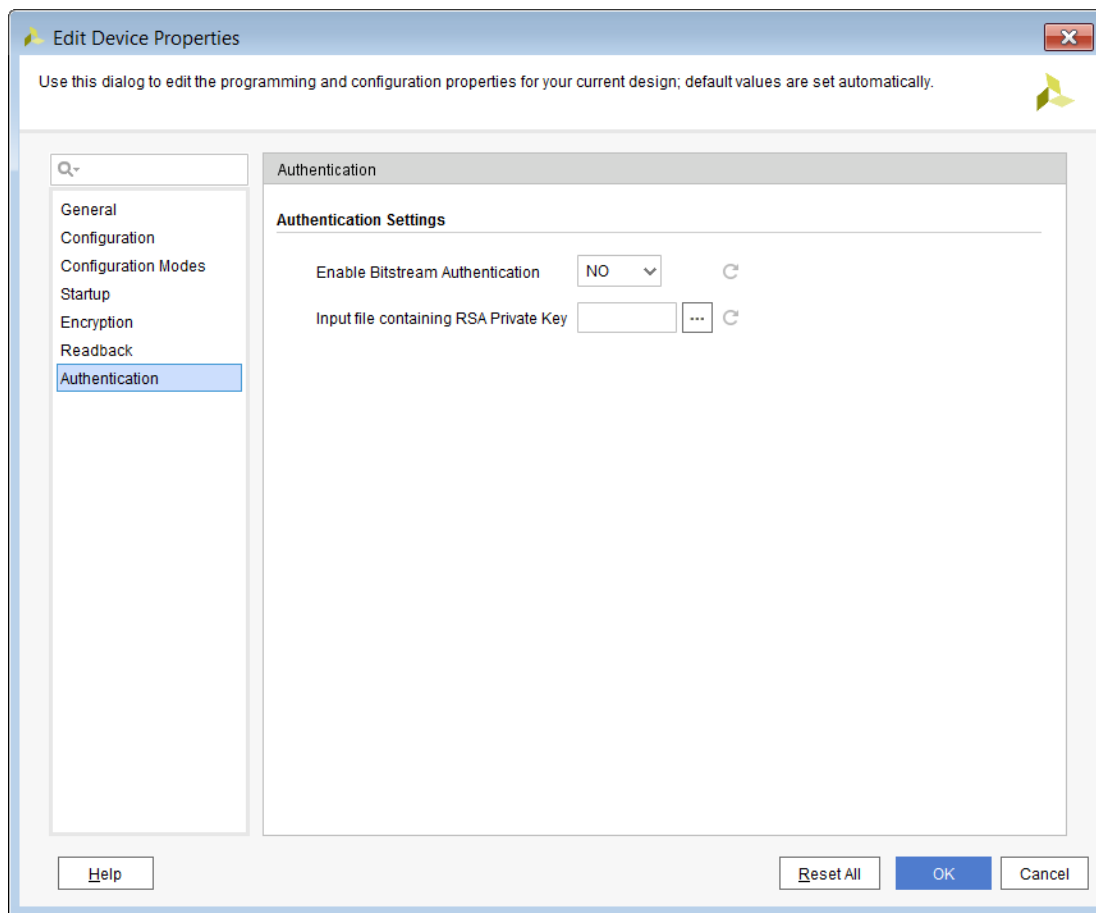
[Edit Device Properties] ダイアログ ボックスで、[Encryption Settings] および[Key Settings] を指定します。

- [Encryption Settings]
 - [Enable Bitstream Encryption] を [YES] に設定します。
 - [Select location of encryption key] を [BBRAM] または [EFUSE] のいずれかに設定します。
 - キー ロケーションは暗号化されたビットストリームに埋め込まれます。
 - 暗号化されたビットストリームをデバイスにダウンロードすると、FPGA で BBR または eFUSE キー レジスタに読み込まれたキーが使用され、暗号化されたビットストリームが復号化されます。
 - [Enable obfuscated key load] を [ENABLE] または [DISABLE] に設定します。
 - イネーブルにすると、ユーザーが生成したキーが BBRAM に保存される前に暗号化されます。ディスエーブルにすると、キーはそのまま BBRAM に保存されます。
- [Key Settings]
 - [Starting AES encryption key (key0)] でビットストリームを暗号化する際に使用する AES 暗号キーを指定します。64 個までの 16 進数文字を使用して、256 ビット キーを指定できます。
 - キーは、拡張子が `.nkey` のファイルに書き込まれます。キーを BBR に読み込む際、またはキーを eFUSE キー レジスタにプログラムする際に、このファイルを使用します。

- 。 **注記:** この値は、入力暗号化ファイルが指定されている場合を除き、現在のプロジェクトの制約ファイルに保存されます。この値が制約ファイルに保存されないようにするには、入力暗号化ファイルを指定してください。
- 。 [Input encryption file] を指定します。
 - 既存の .nky ファイルを指定して、暗号キー設定を取得します。このフィールドはオプションで、AES、HMAC、および CBC を手動で指定する場合は入力する必要はありません。
- 。 [Number of encryption blocks per key] および [Number of frames per AES-256 key] を指定します。
 - 暗号化ブロックおよびフレームの数は、ビットストリームを個別のキーを使用するいくつかのセクションに分割するかを指定します。
- 。 [Starting AES initial vector (IV0) value] を指定します。
 - 最初のキーの初期化ベクターです。各キーには、入力暗号化ファイルで供給可能な個別の初期化ベクターが必要です。
- 。 **注記:** この値は、現在のプロジェクトの制約ファイルに保存されます。この値が制約ファイルに保存されないようにするには、入力暗号化ファイルを指定してください。
- 。 [Starting obfuscate initial vector (Obfuscate IV0) value] を指定します。
 - 難読化キーの初期化ベクターです。
- 。 **注記:** この値は、現在のプロジェクトの制約ファイルに保存されます。この値が制約ファイルに保存されないようにするには、入力暗号化ファイルを指定してください。

認証設定を指定するには、左側のペインで [Authentication] をクリックします。

図 31: [Edit Device Properties] ダイアログ ボックス: [Authentication] ページ



[Edit Device Properties] ダイアログ ボックスの [Authentication] ページで、次のように設定します。

- [Authentication Settings]
 - [Enable Bitstream Authentication] を [YES] に設定します。
 - [Input file containing RSA Private Key] で RSA 秘密キーを含む入力ファイルを指定します。

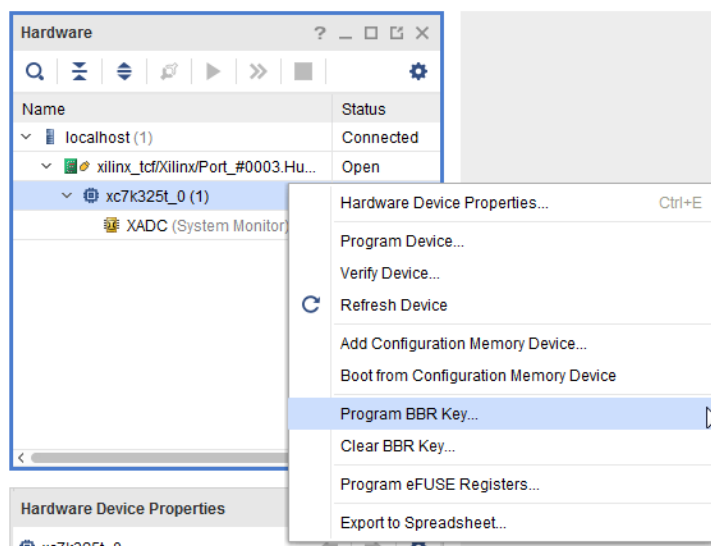
設定を終了したら、[OK] をクリックして設定をプロジェクトに適用します。インプリメンテーションを再実行してビットストリームを再生成します。write_bitstream コマンドの処理が正常に完了すると、生成された暗号キー ファイル (.nky) が暗号化ビットストリーム ファイルと同じ場所に保存されます。

256 ビットの AES (Advanced Encryption Standard) キーを使用してビットストリームを暗号化し、ビットストリームをダウンロードして認証された FPGA 上でのみ実行されるようにすると、ビットストリームの IP を保護できます。これには、暗号化ビットストリームをダウンロードする前に、256 ビットのキーを認証された FPGA の BBR レジスタにプログラムします。

7 シリーズ デバイスの AES キーのプログラム

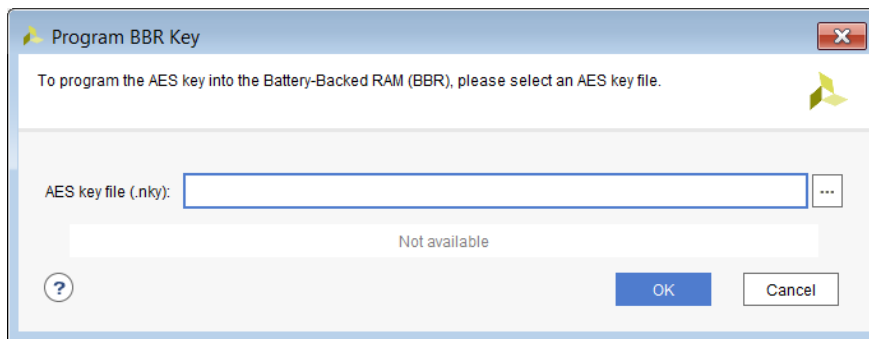
AES キーを BBR にプログラムするには、[Hardware] ウィンドウで FPGA デバイスを右クリックして [Program BBR Key] をクリックします。

図 32: BBR キーをプログラム



[Program BBR Key] ダイアログ ボックスで、ファイル名を入力するか参照ボタンをクリックして AES キー (.nky) ファイルを指定します。有効な .nky ファイルを指定すると、AES フィールドに自動的に値が表示されます。[OK] をクリックすると、ハードウェア マネージャーでキーが BBR にプログラム/読み込まれます。

図 33: [Program BBR Key] ダイアログ ボックス (7 シリーズ デバイス)



キーをプログラムしたら、次のような暗号化ビットストリームを使用して FPGA をプログラムします。

- BBR に読み込んだものと同じ AES キーを使用して暗号化されている。
- 暗号キー ロケーションとして選択した BBRAM がある。

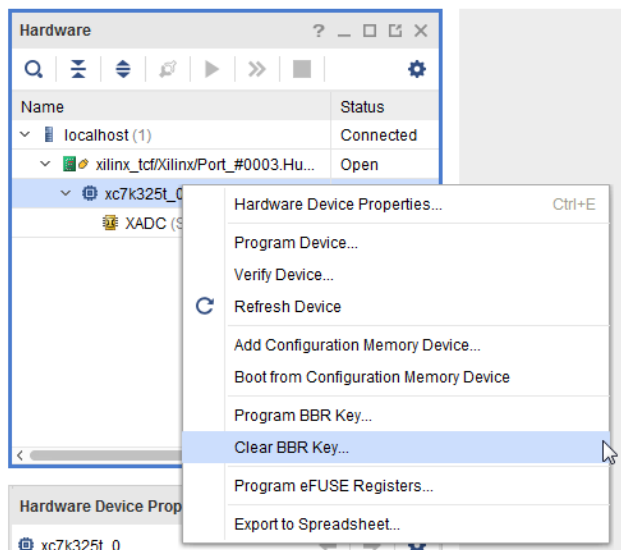
7 シリーズ デバイスの AES キーの消去

AES キーを手動で消去するには、Vbatt ピンの接続を解除し、ボードの電源を切って入れ直します。

注記: ボード/FPGA に電源が投入されたときに PROG ピンを押したり、PROG ピンにパルスを送っても、BBR レジスタはクリアされません。

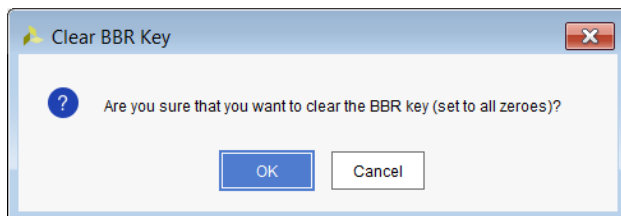
または、Vivado IDE の [Hardware] ウィンドウで FPGA デバイスを右クリックし、[Clear BBR Key] をクリックしても AES キーを消去できます。

図 34: 7 シリーズ デバイスの AES キーの消去



[Clear BBR Key] ダイアログ ボックスで [OK] をクリックしてデバイスからキーを消去します。

図 35: [Clear BBR Key] ダイアログ ボックス

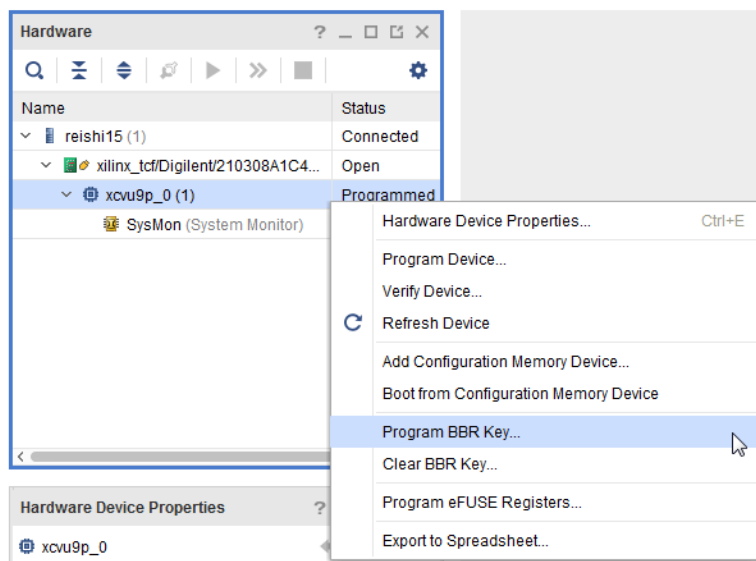


★ **重要:** `verify_hw_devices` を BBR キーで実行すると、エラー メッセージが表示されます。BBR キーを検証するには、キーを含むビットストリームで FPGA をプログラムして、これをテストする必要があります。Vivado では、プログラムされた BBR キーを検証するための BBR プログラム後の検証オプションはサポートされていません。

UltraScale および UltraScale+ デバイスの AES キーのプログラム

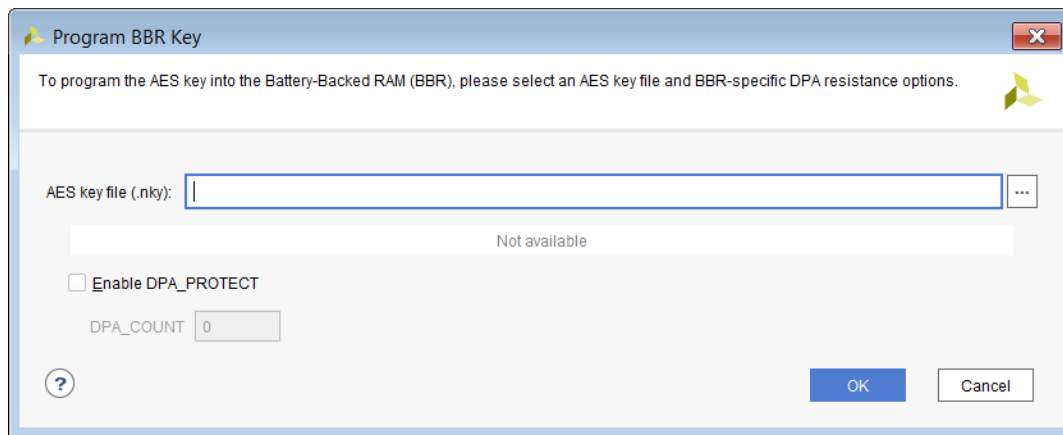
AES キーを BBR にプログラムするには、[Hardware] ウィンドウで FPGA デバイスを右クリックして [Program BBR Key] をクリックします。

図 36: [Hardware] ウィンドウから BBR キーをプログラム



[Program BBR Key] ダイアログ ボックスが表示されます。

図 37: [Program BBR Key] ダイアログ ボックス (UltraScale および UltraScale+ デバイス)



[Program BBR Key] ダイアログ ボックスで、AES キー ファイル (.nky) と [Enable DPA PROTECT] を指定します。

- [AES key file (.nky)]
 - ファイル名を入力するか参照ボタンをクリックして AES キー (.nky) ファイルを指定します。[AES key file (.nky)] 有効な .nky ファイルを指定すると、[AES key] フィールドに自動的に値が表示されます。
- [Enable DPA PROTECT]
 - [Enable DPA PROTECT] チェック ボックスをオンにします。
 - [DPA_COUNT] の値を指定します。有効な値は 1 ~ 256 です。

注記: BBR AES キーおよび DPA_PROTECT 機能の詳細は、『UltraScale アーキテクチャ コンフィギュレーション ユーザー ガイド』 (UG570: [英語版](#)、[日本語版](#)) を参照してください。

[OK] をクリックすると、ハードウェア マネージャーで BBR にキーがプログラムまたは読み込まれます。

キーをプログラムしたら、次のような暗号化ビットストリームを使用して FPGA をプログラムします。

- BBR に読み込んだものと同じ AES キーを使用して暗号化されている。
- 暗号キー ロケーションとして選択した [BBRAM] がある。

★ **重要:** UltraScale デバイスの場合、キーを BBR レジスタにプログラムする前に暗号化されたビットストリームをダウンロードすると、FPGA デバイスがロックされ、BBR キーを読み込むことができません。暗号化されていないビットストリームはダウンロードできますが、キーを BBR にダウンロードできないので、暗号化されたビットストリームはダウンロードできません。ボードの電源を切って入れ直すことにより UltraScale デバイスのロックを解除してから、BBR キーを読み込み直す必要があります。

★ **重要:** `verify_hw_devices` を BBR キーで実行すると、エラー メッセージが表示されます。BBR キーを検証するには、キーを含むビットストリームで FPGA をプログラムして、これをテストする必要があります。Vivado では、プログラムされた BBR キーを検証するための BBR プログラム後の検証オプションはサポートされていません。

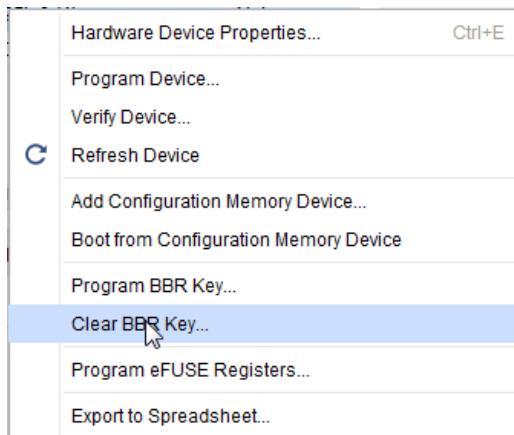
UltraScale および UltraScale+ デバイスの AES キーの消去

AES キーを手動で消去するには、Vbatt ピンの接続を解除し、ボードの電源を切って入れ直します。

注記: ボード/FPGA に電源が投入されたときに PROG ピンを押したり、PROG ピンにパルスを送っても、BBR レジスタはクリアされません。

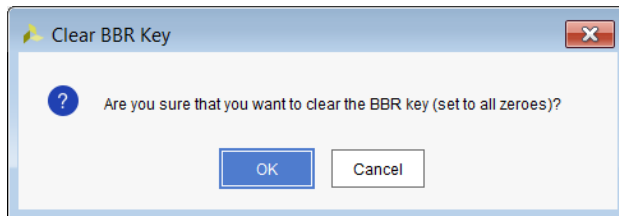
または、Vivado IDE の [Hardware] ウィンドウで FPGA デバイスを右クリックし、[Clear BBR Key] をクリックしても AES キーを消去できます。

図 38: UltraScale および UltraScale+ デバイスの AES キーの消去



[Clear BBR Key] ダイアログ ボックスで [OK] をクリックしてデバイスからキーを消去します。


図 39: [Clear BBR Key] ダイアログ ボックス





eFUSE レジスタのアクセスおよびプログラム

7 シリーズ、UltraScale、および UltraScale+ デバイスには、特定のファンクションを実行する eFUSE ビットというワンタイム プログラマブル ビットが含まれます。次のタイプの eFUSE ビットがあります。

- FUSE_DNA: 固有のデバイス識別子ビットを格納します (プログラム不可)。
- FUSE_USER: 32 ビットのユーザー定義コードを格納します。
- FUSE_KEY: AES ビットストリームの復号化に使用するキーを格納します。
- FUSE_CNTL: キーの使用と eFUSE レジスタへの読み出し/書き込みアクセスを制御します。
- FUSE_SEC: UltraScale および UltraScale+ デバイスの特別なデバイス セキュリティ 設定を制御します。

 **重要:** eFUSE レジスタ ビットは 1 回しかプログラムできません。eFUSE レジスタ ビットを一度プログラムすると (プログラムされていない 0 ステートからプログラムされた 1 ステート)、0 にリセットしたりプログラムし直すことはできません。eFUSE レジスタをプログラムする前に、必ず設定を注意して確認するようにしてください。

 **注意:** eFUSE レジスタ ビットが以前にプログラムされている場合 (たとえば、プログラムされていないことを示すステート 0 からプログラム済みのステート 1 になっている場合)、それらを再度プログラムしようとすると、Vivado ハードウェア マネージャーでビットの中にプログラム済みのものがあることを示すクリティカル警告メッセージが表示されます。ただし、この警告に関係なく、前の動作でプログラムされなかった eFUSE レジスタ ビット (プログラムされていないことを示すステート 0) はプログラムされます。

 **重要:** まず、FUSE_USER、FUSE_KEY、および FUSE_RSA レジスタをプログラムしてから、eFUSE プログラミング ウィザード実行し直して FUSE_SEC ビットをプログラムし、FPGA セキュリティ 設定を制御して、最後に FUSE_CNTL ビットをプログラムしてこれらの eFUSE ビットへの読み出し/書き込みアクセスを制御します。

eFUSE プログラミングのケーブル サポート

eFUSE プログラミングでサポートされる互換性のある JTAG ダウンロード ケーブルおよびデバイスは、次のとおりです。

- ザイリンクス SmartLynq データ ケーブル (HW-SMARTLYNQ-G/DLC20)
- ザイリンクス プラットフォーム ケーブル USB II (DLC10)
- Digilent 社 JTAG-HS1
- Digilent 社 JTAG-HS2
- Digilent 社 JTAG-HS3

7 シリーズ デバイスの eFUSE レジスタのアクセスおよびプログラム

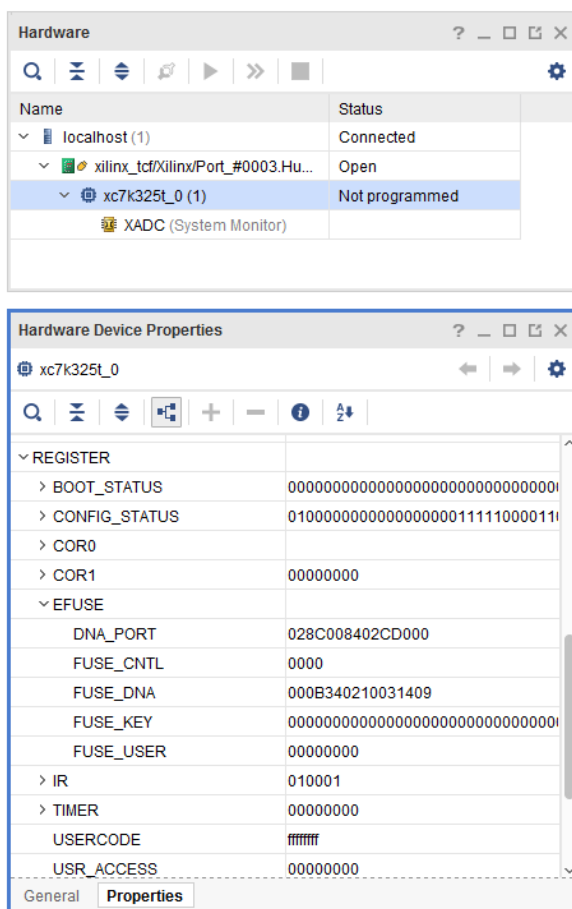
FUSE_DNA: Device DNA

各7シリーズ デバイスには、サイリンクスによってプログラムされた Device DNA というデバイス ID があります。7シリーズ デバイスは、64 ビットの Device DNA を持ちます。これらのビットを読み出すには、Vivado Design Suite の [Tcl Console] ウィンドウで次の Tcl コマンドを実行します。

```
get_property [lindex [get_hw_device] 0] REGISTER.EFUSE.FUSE_DNA
```

Device DNA は、Vivado Design Suite の [Hardware Device Properties] ウィンドウで eFUSE レジスタの [FUSE_DNA] により確認できます。

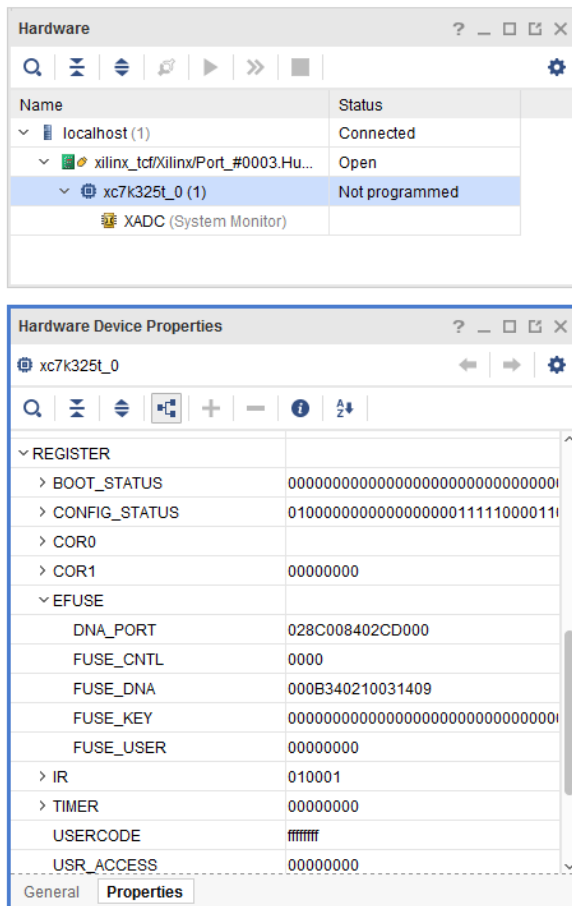
図 40: eFUSE DNA



これらの機能の詳細は、『7シリーズ FPGA コンフィギュレーション ユーザー ガイド』 (UG470: [英語版](#)、[日本語版](#)) を参照してください。

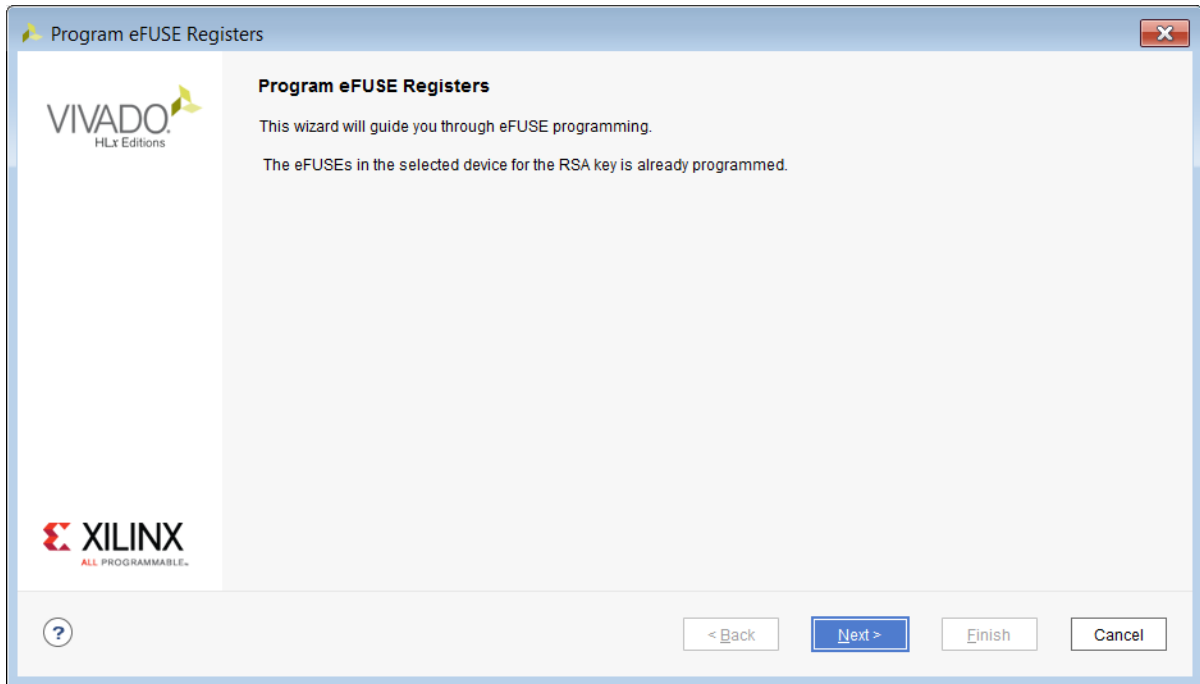
eFUSE レジスタをプログラムするには、[Hardware] ウィンドウで FPGA デバイスを右クリックして [Program eFUSE Registers] をクリックします。

☒ 41: [Program eFUSE Registers] コマンド



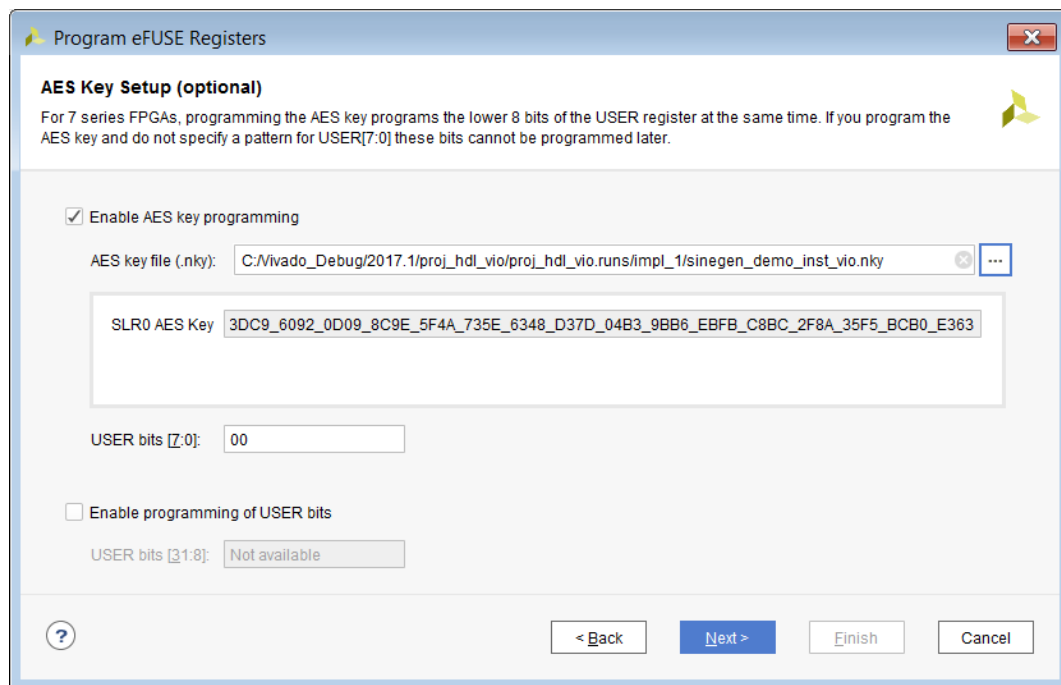
次の図に示す Program eFUSE Registers ウィザードが起動し、eFUSE レジスタのさまざまなオプションを設定できます。

図 42: Program eFUSE Registers ウィザード



[AES Key Setup] ページでは、次の設定を指定します。

図 43: Program eFUSE Registers ウィザード: [AES Key Setup] ページ

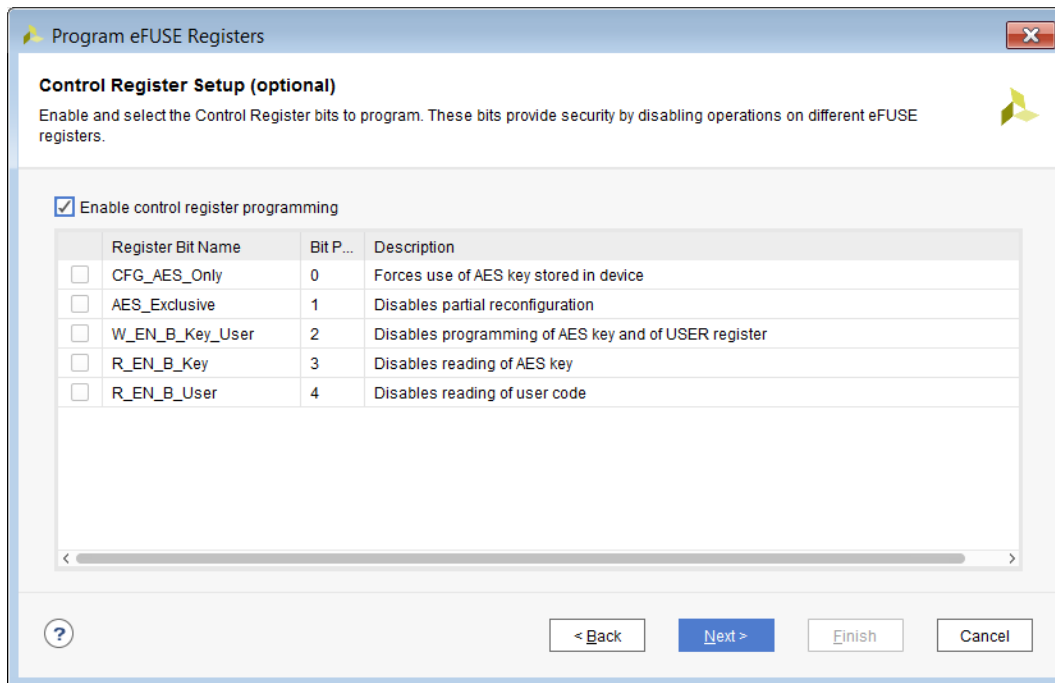


- [AES Key file]

- 。 ファイル名を入力するか参照ボタンをクリックして AES キー (.nky) ファイルを指定します。有効な .nky ファイルを指定すると、[AES key] フィールドに自動的に値が表示されます。
- [USER bits [7:0] and USER bits [31:8]]
 - 。 FUSE_USER ビットを使用すると、独自の特別な 32 ビット パターンをプログラムできます。FUSE_USER の下位 8 ビットは 256 ビットの AES (Advanced Encryption Engine) キーと同時にプログラムされます。上位 24 ビットは、後で AES キーと同時にプログラムされます。

[Control Register Settings] ページでは、次の設定を指定します。

図 44: [Control Register Setup]

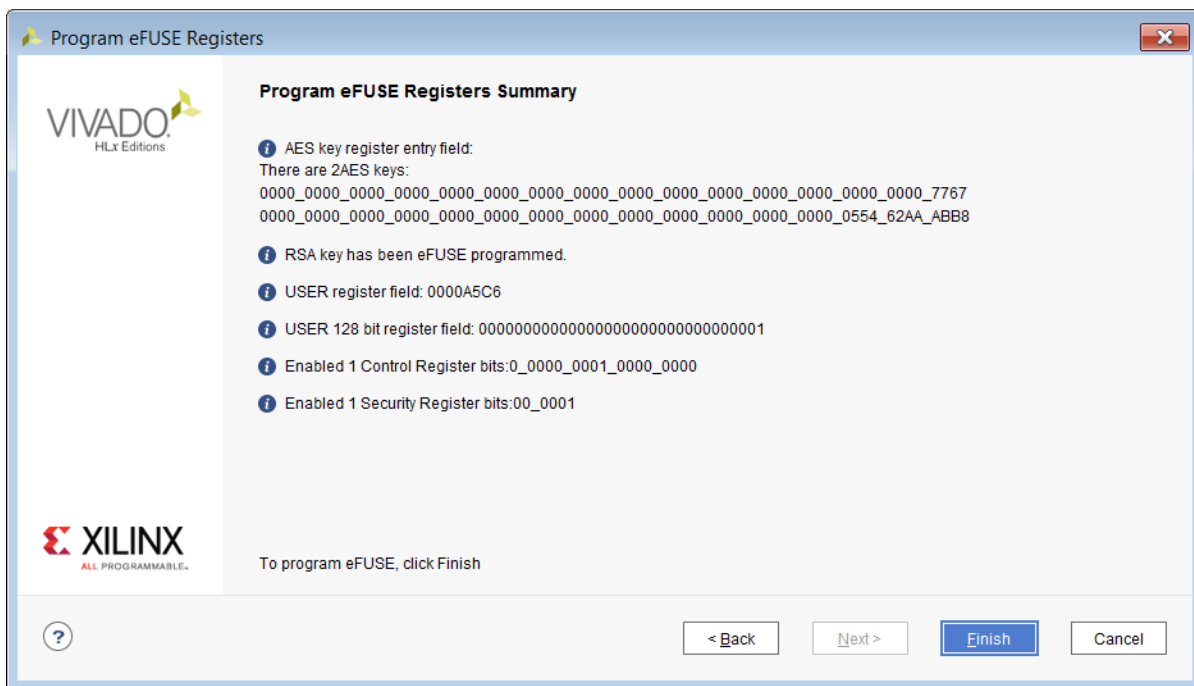


- CFG_AES_Only: 格納された AES キーを使用するよう強制します。
- AES_Exclusive: Dynamic Function eXchange の使用をディスエーブルにします。
- W_EN_B_Key_User: AES キーおよびユーザー レジスタのプログラムをディスエーブルにします。
- R_EN_B_Key: AES キーの読み出しをディスエーブルにします。
- R_EN_B_User: ユーザー コードの読み出しをディスエーブルにします。
- W_EN_B_Cntl: この制御レジスタのプログラムをディスエーブルにします。

これらの機能の詳細は、『7 シリーズ FPGA コンフィギュレーション ユーザー ガイド』 (UG470: [英語版](#)、[日本語版](#)) を参照してください。

[Program eFUSE Registers Summary] ページで eFUSE 設定を確認します。

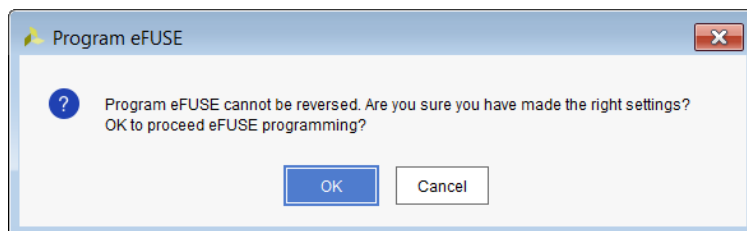
図 45: [Program eFUSE Registers Summary]



このページには、Program eFUSE Registers ウィザードで設定したすべてのビットが表示されます。個々のビットの設定が表示されるので、特定のプログラム設定を確認できます。このサマリ ページを注意深く確認し、すべてのビットが意図したとおりにプログラムされるよう設定されていることを確認します。

[Finish] をクリックすると、eFUSE のプログラムを確認するダイアログ ボックスが表示されます。

図 46: eFUSE のプログラムを確認するダイアログ ボックス



[OK] をクリックして指定した eFUSE ビットをプログラムします。

eFUSE プログラムの強制設定

ビットがレジスタ内に含まれるか、前にプログラムされたかどうかに関係なく、`program_hw_devices` コマンドに `-force_efuse` オプションを付けると、ビットを強制的に設定できます。このオプションを使用すると、基本的なレジスタ バウンダリ チェックのみが実行されます。

UltraScale および UltraScale+ デバイスの eFUSE レジスタのアクセスおよびプログラム

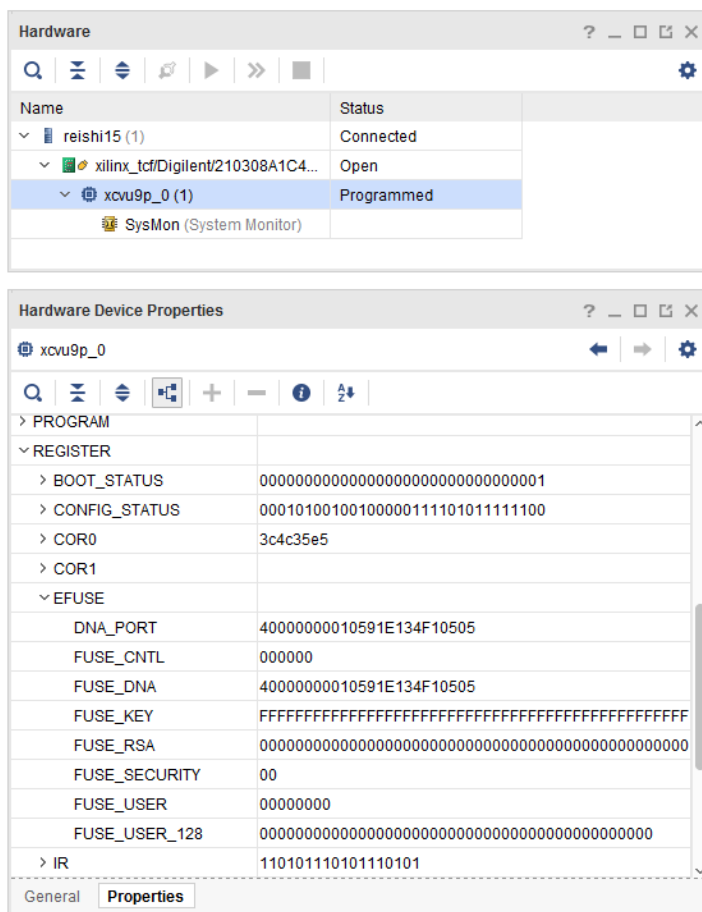
FUSE_DNA: Device DNA

各 UltraScale デバイスには、ザイリンクスによってプログラムされた Device DNA というデバイス ID があります。FUSE_DNA はユーザーがプログラムすることはできません。UltraScale デバイスは、96 ビットの Device DNA を持ちます。FUSE_DNA を読み出すには、Vivado Design Suite の [Tcl Console] ウィンドウで次の Tcl コマンドを実行します。

```
get_property [lindex [get_hw_device] 0] REGISTER.EFUSE.FUSE_DNA
```

Device DNA は、Vivado Design Suite の [Hardware Device Properties] ウィンドウで eFUSE レジスタの [FUSE_DNA] により確認できます。

図 47: eFUSE DNA (UltraScale、UltraScale+)

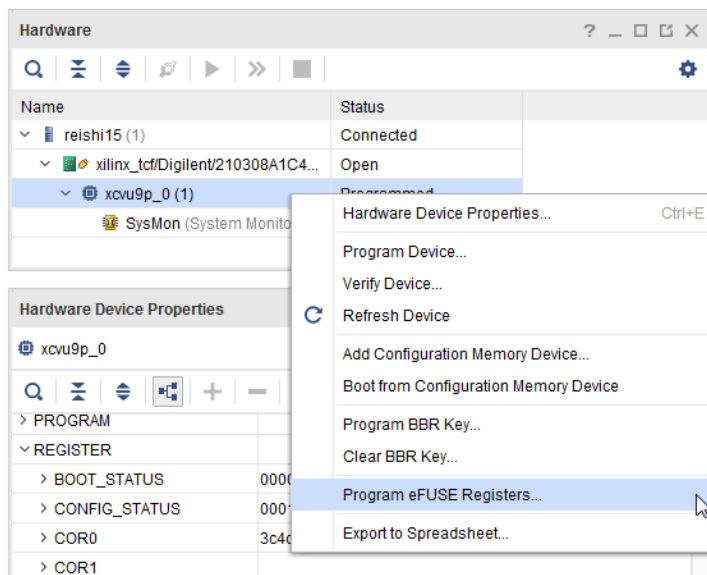


これらの機能の詳細は、『UltraScale アーキテクチャ コンフィギュレーション ユーザー ガイド』 (UG570: [英語版](#)、[日本語版](#)) を参照してください。

eFUSE レジスタのプログラム

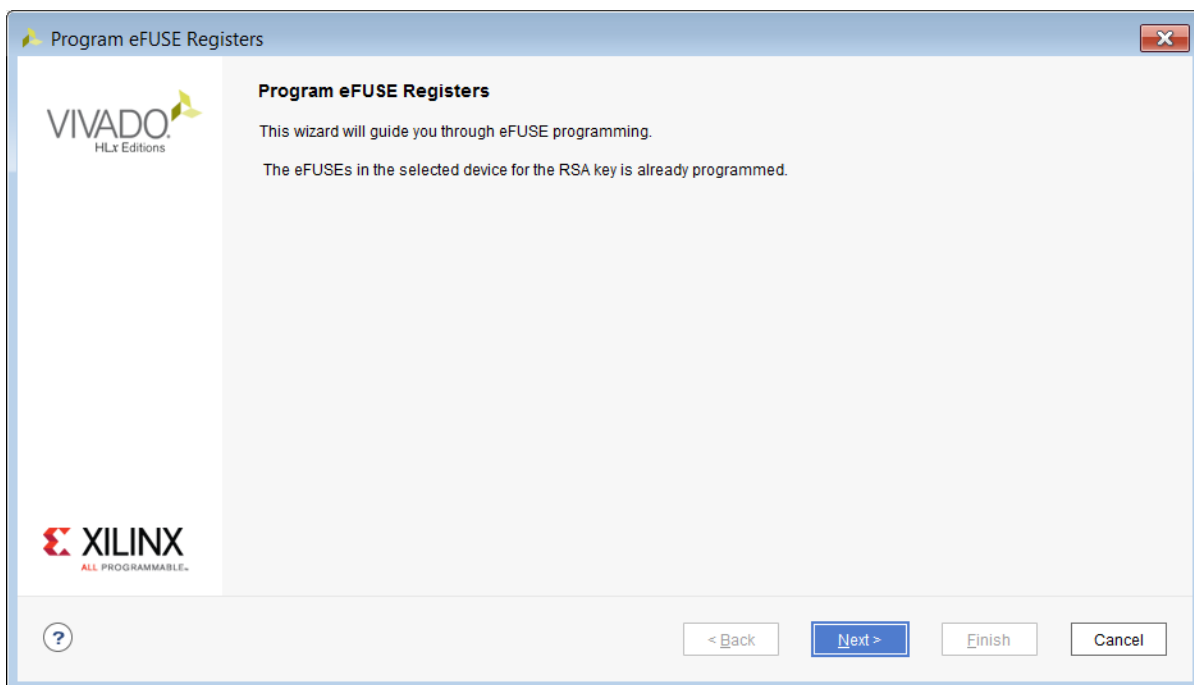
eFUSE レジスタをプログラムするには、[Hardware] ウィンドウで FPGA デバイスを右クリックして [Program eFUSE Registers] をクリックします。

図 48: [Program eFUSE Registers] コマンド (UltraScale、UltraScale+)



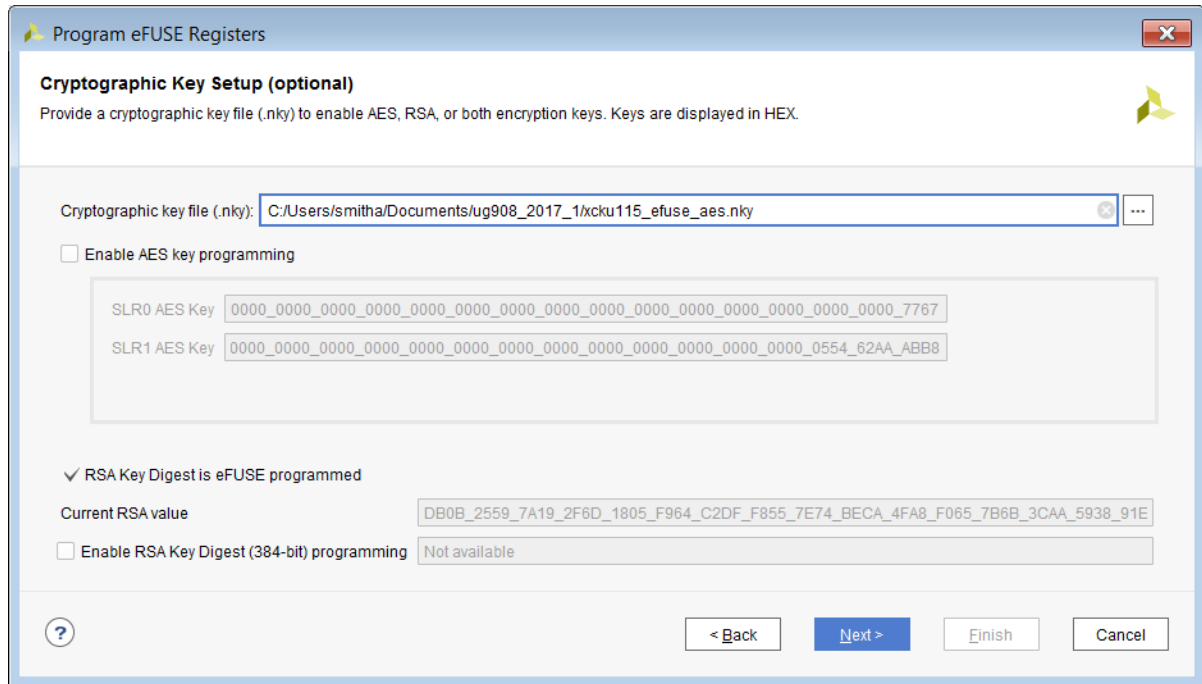
次の図に示す Program eFUSE Registers ウィザードが起動し、eFUSE レジスタのさまざまなオプションを設定できます。

図 49: Program eFUSE Registers ウィザード



[AES Key Setup] ページでは、次の設定を指定します。

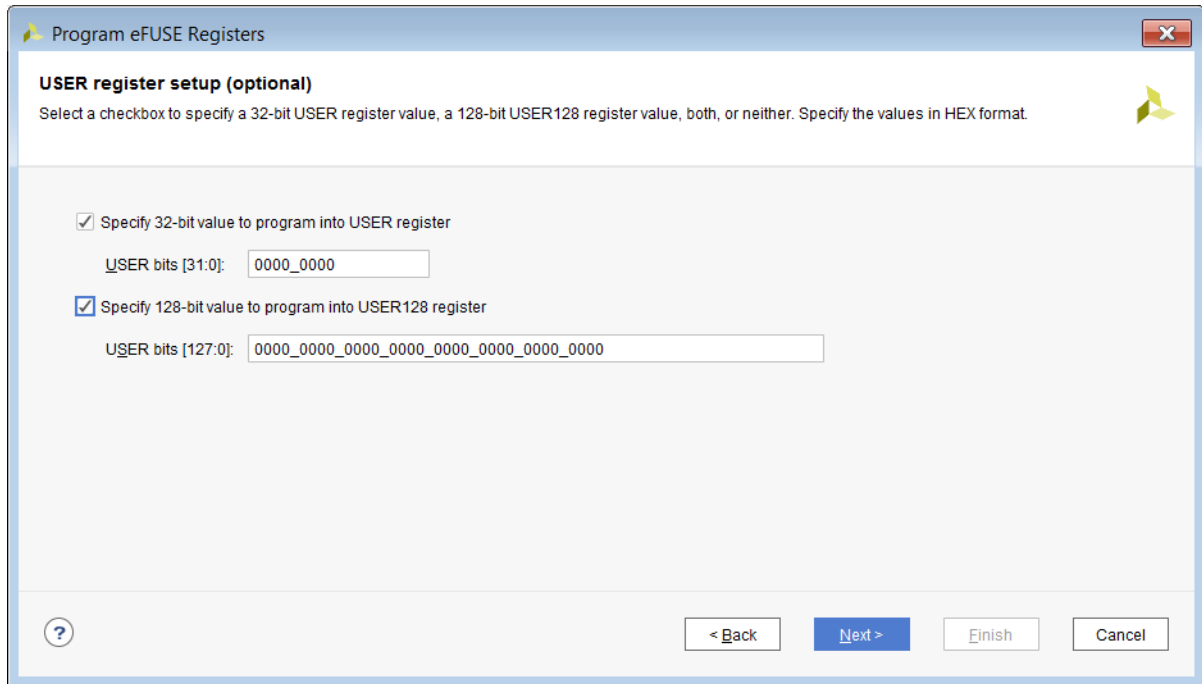
図 50: Program eFUSE Registers ウィザード: [Cryptographic Key Setup] ページ



[Cryptographic Key Setup] ページでは、次の設定を指定します。

- [Cryptographic file key (.nky)]: eFUSE AES および RSA キーを含む .nky ファイルを指定します。
- [AES Key (256-bit)]: 読み込まれた暗号化ビットストリームを復号化するために使用される指定した .nky から読み出された 256 ビット AES eFUSE キー。
- [RSA Key Digest (384-bit)]: RSA により使用される指定した .nky から読み出された 384 ビット RSA eFUSE キー。
- [USER register setup] ページで、32 ビットの USER レジスタまたは 128 ビットの USER レジスタを指定します。

図 51: Program eFUSE Registers ウィザード: [USER register setup] ページ



[USER register setup] ページで、ユーザー定義のレジスタ ビットを指定します。32 ビットの USER レジスタ (FUSE_USER) および 128 ビットの USER レジスタ (FUSE_USER128) は、ユーザー定義のワンタイム プログラマブル eFUSE ビットです。これらのレジスタのビットは、累積的にプログラムできます。たとえば、eFUSE プログラムセッションで 1 つのユーザー ビット (USER = 0x0000_0001 またはビット 0) のみをプログラムし、次の eFUSE プログラムセッションで残りの 0 ビットのいずれか (USER = 0x0000_0003 またはビット 1) をプログラムできます。

FUSE_USER および FUSE_USER_128 レジスタをプログラムすると、次の方法で読み出すことができます。

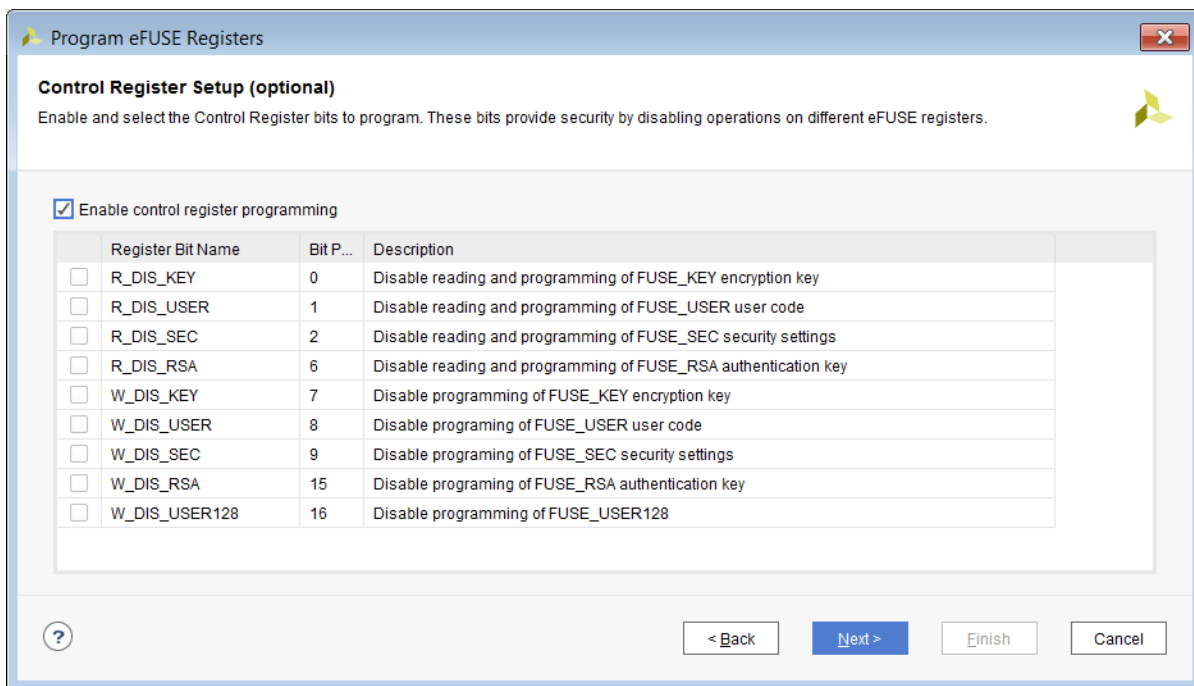
- Tcl コマンドを使用

```
report_property [lindex [get_hw_device] 0] REGISTER.EFUSE.FUSE_USER
report_property [lindex [get_hw_devices] 0] REGISTER.EFUSE.FUSE_USER_128
```

- refresh_hw_device を実行した後に Vivado の [Hardware Device Properties] ウィンドウで確認。

[Control Register Setup] ページでは、次の設定を指定します。

図 52: Program eFUSE Registers ウィザード: [Control Register Setup] ページ



[Control Register Settings] ページでは、eFUSE 制御設定を指定します。

- R_DIS_KEY: キーおよび FUSE_KEY 暗号化キーのプログラムを検証する CRC チェックをディスエーブルにします。
- R_DIS_USER: 32 ビットのユーザー ビット (FUSE_USER) の読み出しおよびプログラムをディスエーブルにします。
- R_DIS_SEC: セキュリティ ビット (FUSE_SEC) の読み出しおよびプログラムをディスエーブルにします。
- R_DIS_RSA: RSA キーレジスタ (FUSE_RSA) の読み出しおよびプログラムをディスエーブルにします。
- W_DIS_USER: 32 ビットのユーザー ビット (FUSE_USER) のプログラムをディスエーブルにします。
- W_DIS_SEC: セキュリティ ビット (FUSE_SEC) のプログラムをディスエーブルにします。
- W_DIS_RSA: RSA キーレジスタ (FUSE_RSA) のプログラムをディスエーブルにします。
- W_DIS_USER_128: 128 ビットのユーザー ビット (FUSE_USER128) のプログラムをディスエーブルにします。

FUSE_SEC レジスタの詳細は、『UltraScale アーキテクチャ コンフィギュレーション ユーザー ガイド』(UG570: [英語版](#)、[日本語版](#)) を参照してください。

制御レジスタ設定のディスエーブル

制御レジスタのプログラムをディスエーブルにするには、次の Tcl コマンドを実行します。

```
program_hw_devices -control_efuse {20} [lindex [get_hw_devices] $deviceIdx]
```

\$deviceIdx は、eFUSE 制御レジスタ ビットのプログラムをディスエーブルにする UltraScale または UltraScale+ デバイスのインデックスを設定します。

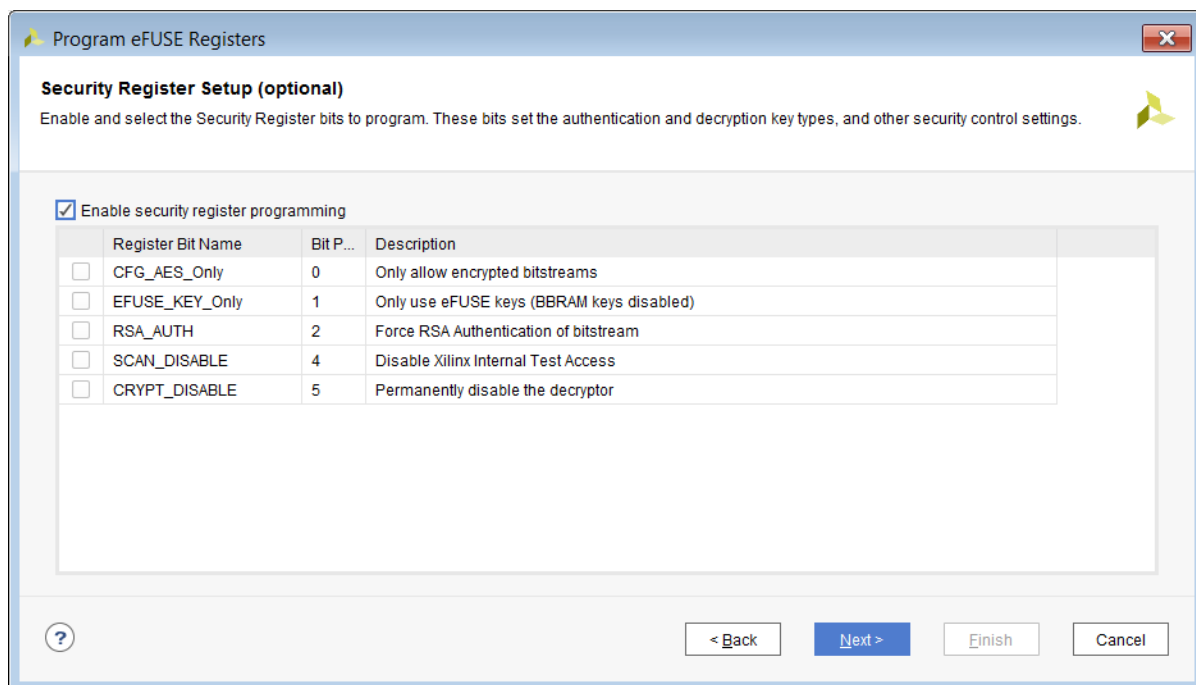
これにより、W_DIS_CNTL ビットが設定され、eFUSE 制御レジスタ ビットのプログラムがディスエーブルになります。



重要: W_DIS_CNTL ビットがプログラムされる場合、その他の eFUSE 制御レジスタ ビットのプログラムがディスエーブルになり、デバイスの制御レジスタがそれ以上編集されないようになります。

[Security Register Setup] ページで、次の設定を指定します。

図 53: Program eFUSE Registers ウィザード: [Security Register Setup] ページ



Register Bit Name	Bit P...	Description
<input type="checkbox"/> CFG_AES_Only	0	Only allow encrypted bitstreams
<input type="checkbox"/> EFUSE_KEY_Only	1	Only use eFUSE keys (BBRAM keys disabled)
<input type="checkbox"/> RSA_AUTH	2	Force RSA Authentication of bitstream
<input type="checkbox"/> SCAN_DISABLE	4	Disable Xilinx Internal Test Access
<input type="checkbox"/> CRYPT_DISABLE	5	Permanently disable the decryptor

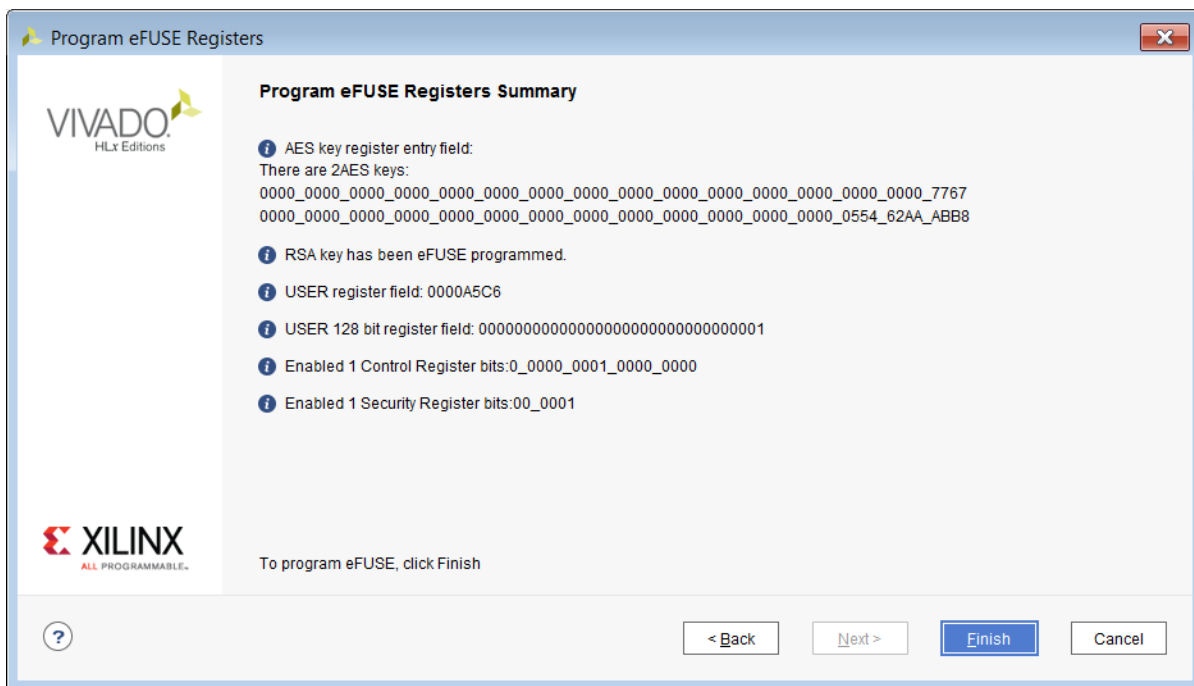
[Security Register Setup] ページでは、FPGA に読み込み可能なビットストリームのタイプに関するセキュリティ制御オプションを指定します。次の FUSE_SEC 設定があります。

- [CFG_AES_Only]: 暗号化されたビットストリームのみを読み込み可能にします。
- [EFUSE_KEY_Only]: 復号化に eFUSE キーのみを使用できるようにします。
- [RSA_AUTH]: ビットストリームの RSA 認証を必須にします。
- [SCAN_DISABLE]: 内部テスト レジスタへのザイリンクス アクセスをディスエーブルにします。
- [CRYPT_DISABLE]: 復号化を恒久的にディスエーブルにします。

FUSE_SEC レジスタの詳細は、『UltraScale アーキテクチャ コンフィギュレーション ユーザー ガイド』(UG570: [英語版](#)、[日本語版](#))を参照してください。

[Program eFUSE Registers Summary] ページで eFUSE 設定を確認します。

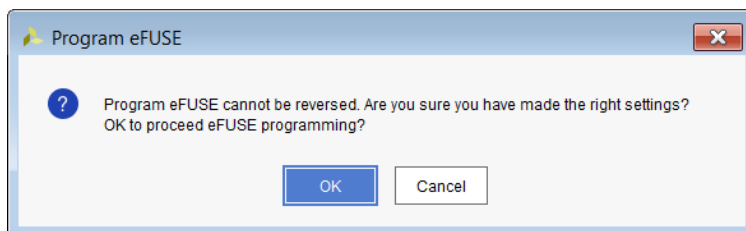
図 54: [Program eFUSE Registers Summary]



このページには、Program eFUSE Registers ウィザードで設定したすべてのビットが表示されます。個々のビットの設定が表示されるので、特定のプログラム設定を確認できます。このサマリ ページをよく確認し、すべてのビットが意図したとおりにプログラムされるよう設定されていることを確認します。

[Finish] をクリックすると、eFUSE のプログラムを確認するダイアログ ボックスが表示されます。

図 55: eFUSE のプログラムを確認するダイアログ ボックス



[OK] をクリックして指定した eFUSE ビットをプログラムします。

JTAG インターフェイスのディスエーブル

eFUSE レジスタを使用して JTAG インターフェイスをディスエーブルにするには、次の Tcl コマンドを実行します。

```
program_hw_devices -force_efuse -security_efuse {08} [lindex
[get_hw_devices] $deviceIdx]
```

\$deviceIdx は、JTAG インターフェイスをディスエーブルにした UltraScale または UltraScale+ デバイスのインデックスに設定されます。



重要: 7 シリーズの JTAG インターフェイスをディスエーブルにする Tcl コマンドは、上記の UltraScale または UltraScale+ デバイスに使用されたコマンドとは異なります。7 シリーズ デバイスの場合は、ザイリンクス アンサー レコード [65110](#) にリストされる XSK_EFUSEPL_DISABLE_JTAG_CHAIN の部分を参照してください。

注記: このプログラムは、ほかのすべての eFUSE ビットをプログラムした後、最後に実行してください。



重要: JTAG Distable ビットがプログラムされている場合、JTAG インターフェイスがディスエーブルになり、デバイスをテストおよびコンフィギュレーションできなくなります。このビットは、JTAG を使用してデバイスにアクセスする必要がない場合にのみプログラムしてください。

eFUSE プログラムの強制設定

ビットがレジスタ内に含まれるか、前にプログラムされたかどうかに関係なく、`program_hw_devices` コマンドに `-force_efuse` オプションを付けると、ビットを強制的に設定できます。このオプションを使用すると、基本的なレジスタ バウンダリ チェックのみが実行されます。

eFUSE NKZ ファイル

ザイリンクスでは、すべての eFUSE プログラミング設定を 1 つのファイルに取り込んで、eFUSE 設定をほかの eFUSE プログラム インプリメンテーションにエクスポートしたり、同じ eFUSE 設定を多くのデバイスに一気にプログラムしたりするため、NKZ というファイル形式 (拡張子は `.nkz`) を提供しています。NKZ 形式は既存の NKY 形式の上位集合で、すべての NKY フィールドと、プログラマブル eFUSE レジスタ設定すべてがサポートされます。

eFUSE エクスポート ファイル (NKZ)

eFUSE 設定は複数のパスでプログラムすることが推奨されるので、外部から可視可能な NKZ ファイルに常にエクスポートされ、各 eFUSE 操作ごとにアップデートされます。このファイルは、デバイスに適用された eFUSE 設定をすべて保存するためのもので、複数の eFUSE 操作中に適用されていてもすべて保存されます。ファイルは Vivado でトラックされるので、累積されたデバイスの eFUSE 設定を含む eFUSE エクスポート ファイル (NKZ 形式) は常に 1 つです。オプションを指定しない場合、このファイルのデフォルト名は次のようになります。

```
export_<FUSE_DNA>.nkz
```

<FUSE_DNA> はデバイスの FUSE_DNA レジスタ値です。

このデフォルト ファイルは Vivado® IDE の起動ディレクトリに含まれます。このファイルは、次の例のように、Tcl コマンドの `-efuse_export_file` に `program_hw_devices` オプションを付けると変更できます。

```
program_hw_devices -user_efuse {1} -efuse_export_file {my-settings.nkz}
```

これで、Tcl オプションでこのファイルを繰り返し指定しなくても、Vivado IDE がエクスポートした eFUSE 設定の NKZ ファイルを使用して開始されるようになります。以前に作成した eFUSE エクスポート ファイルは削除されません。このファイルには、eFUSE エクスポート ファイル名を変更する前に適用した eFUSE 設定がすべて含まれます。すべての eFUSE 操作が実行されたら、それらの設定が現在の eFUSE エクスポート ファイルに保存されます。

また、実際にデバイスをプログラムせずに、eFUSE 設定をエクスポートするだけのオプションもあります。これは、デバイスに影響を与えず、必要な eFUSE 設定すべてを含む NKZ ファイルを作成する際に便利です。さらに、このモードでの eFUSE プログラミングのミスは、単に eFUSE エクスポート ファイルを削除して、やり直すだけで簡単に修正できます。このエクスポートのみのモードは、次の Tcl コマンドで使用できます。

```
program_hw_devices -user_efuse {1} -only_export_efuse
```

このオプションは、各コマンドごとにのみ適用されるので、すべての eFUSE 操作で使用する、デバイスがプログラムされないようにする必要があります。eFUSE 設定はプログラミングがあったかどうかに関係なく常に NKZ ファイルにエクスポートされるので、プログラミング フローとエクスポートのみのフローを混せて使用しないようにしてください。混せて使用すると、出力される eFUSE エクスポート ファイルに、実際にデバイスにプログラムされた eFUSE 設定と NKZ ファイルにエクスポートされただけの設定の両方が混在してしまい、デバイス内の eFUSE レジスタの本当のステータスがいまいになってしまいます。このエクスポートのみのモードを使用すると、まったく同じ NKZ 形式の eFUSE エクスポート ファイルがプログラミングの場合と同様に作成されますが、プログラミングは実行されません。

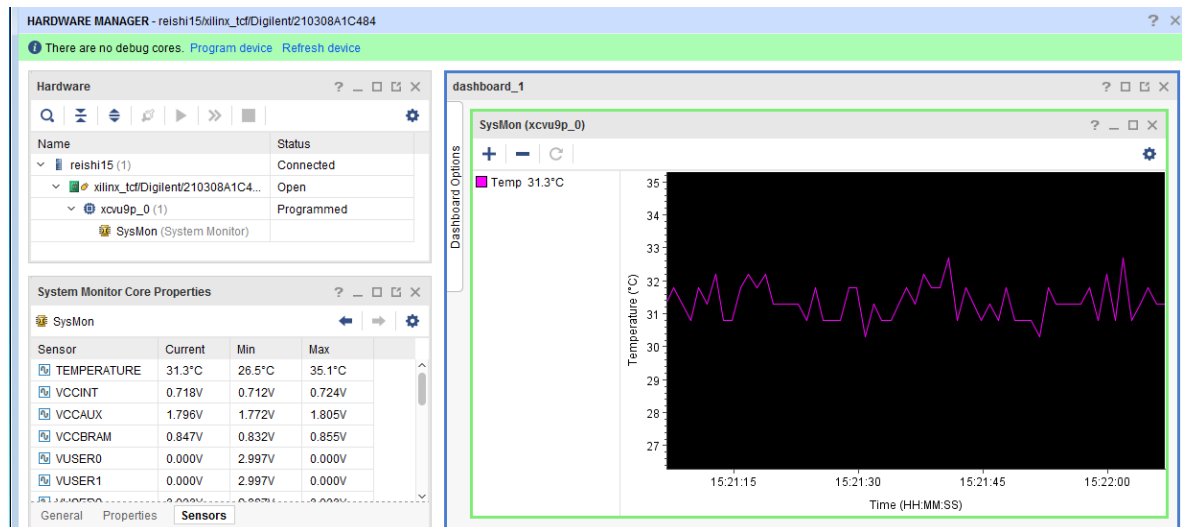
システム モニター

システム モニター (SYSMON) のアナログ/デジタル コンバーター (ADC) は、ハードウェア デバイスのダイ温度と電圧を計測します。SYSMON は、オンチップ温度および電源センサーを使用して物理環境を監視します。ADC では、広範囲のアプリケーションに高精度のアナログ インターフェイスが提供されます。

特定デバイス アーキテクチャの詳細は、次を参照してください。

- 『UltraScale アーキテクチャ システム モニター ユーザー ガイド』 (UG580: [英語版](#)、[日本語版](#))
- 『7 シリーズ FPGA および Zynq-7000 SoC XADC デュアル 12 ビット 1MSPS アナログ-デジタル コンバーター ユーザー ガイド』 (UG480: [英語版](#)、[日本語版](#))

図 56: システム モニター



システム モニター (hw_sysmon) のデータは、ハードウェア システム モニター レジスタ (hw_sysmon_reg) オブジェクトを使用してアクセス可能なステータス レジスタと呼ばれる専用レジスタに保存されます。システム モニター レジスタの内容を取得するには、get_hw_sysmon_reg コマンドを使用します。

システム モニターをサポートするすべてのデバイスには、`refresh_hw_device` が呼び出されたときに作成される `hw_sysmon` オブジェクトが 1 つまたは複数含まれます。`hw_sysmon` オブジェクトが作成されると、すべての電圧レジスタおよび制御レジスタに対して、1 つのプロパティが割り当てられます。`hw_sysmon` オブジェクトでは、温度および電圧レジスタに割り当てられた値は既に摂氏/華氏およびボルトに変換されています。

`get_hw_sysmon_reg` コマンドを使用してシステム モニターのレジスタに格納された 16 進数値を取得することもできますが、一部のレジスタの値は `hw_sysmon` オブジェクトのフォーマットされたプロパティとして取得することもできます。たとえば、レジスタの 16 進数に直接アクセスするのではなく、次のコードを使用して、指定の `hw_sysmon` オブジェクトの `TEMPERATURE` プロパティを取得できます。

```
set opTemp [get_property TEMPERATURE [lindex [get_hw_sysmons] 0]]
```

システム モニターのコマンド リストは、[hw_sysmon の Tcl コマンド](#) を参照してください。

SVF (Serial Vector Format) ファイルを使用したプログラム

FPGA およびコンフィギュレーション メモリ デバイスをプログラムする別の方法として、SVF (Serial Vector Format) ファイルを使用する方法があります。Vivado® Design Suite および Vivado Lab Edition で生成した SVF ファイルには、デバイスをプログラムするのに必要な低水準 JTAG 命令とデータが含まれます。SVF ファイルを生成すると、Vivado IDE の外部でバウンダリスキャン テスト ツールで使用できます。

SVF ファイルの生成手順は、次のとおりです。

1. SVF オフライン ターゲットを作成します。
2. 作成した SVF ターゲットを開きます。
3. ターゲットにデバイスを追加し、SVF JTAG スキャン チェーンを定義します。
4. FPGA または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。
5. SVF を記述します。
6. SVF ターゲットを閉じます。
7. (オプション)SVF を実行します。

手順 4 でプログラム操作が順に記録され 4、キャッシュ ファイルに保存されます。このキャッシュ ファイルを、手順 5 で SVF ファイルに記述し 5 ます。SVF ファイルを作成すると、バウンダリスキャン ツールで使用したり、Vivado Design Suite または Vivado Lab Edition ツールで実行したりできます。

★ **重要:** XSVF ファイル形式は Vivado IDE ではサポートされていません。

SVF ターゲットの作成

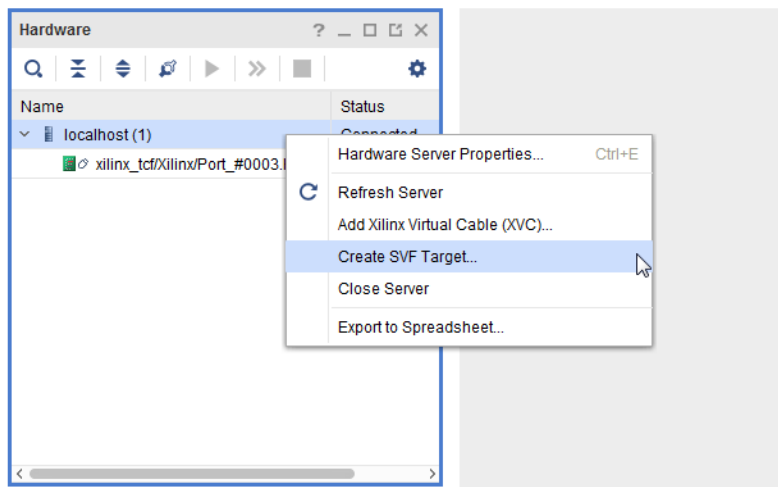
SVF ターゲットは、ザイリンクス プラットフォーム ケーブル USB または Digilent JTAG ケーブル ハードウェア ターゲットと似ています。プロパティおよび Tcl コマンドはすべて同じで、SVF ターゲットはアクティブ ライブ ケーブルではないことが違うだけです。つまり、このターゲットにどんな操作を実行しても、SVF を実行するまではハードウェアには操作は実行されません。SVF を作成するために、システムにケーブルを接続する必要はありません。

Vivado IDE の使用

Vivado ハードウェア マネージャーで SVF ターゲットを作成するには、Vivado または Vivado Lab Edition を起動して Vivado ハードウェア マネージャーを開きます。[Tools]→[Create SVF Target] をクリックすると SVF ターゲットを作成できます。これでローカル ホストのサーバーが自動的に開き、[Create SVF Target] ダイアログ ボックスが開きます。

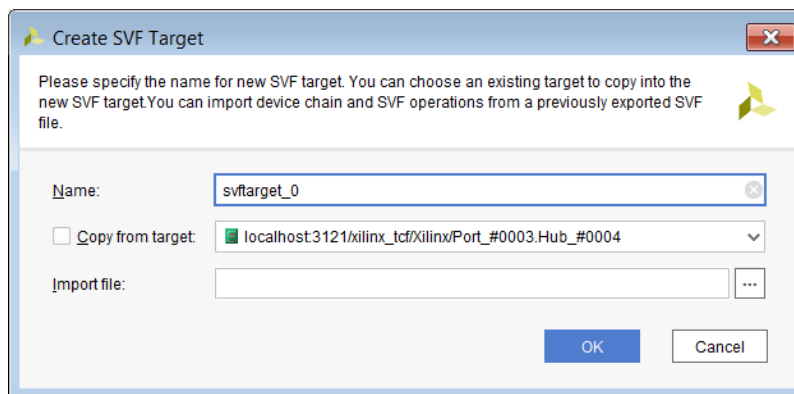
次に示す使用可能なサーバーでオフライン SVF ターゲットを作成できます。

図 57: SVF ターゲットの作成



次の図に示す [Create SVF Target] ダイアログ ボックスが開きます。

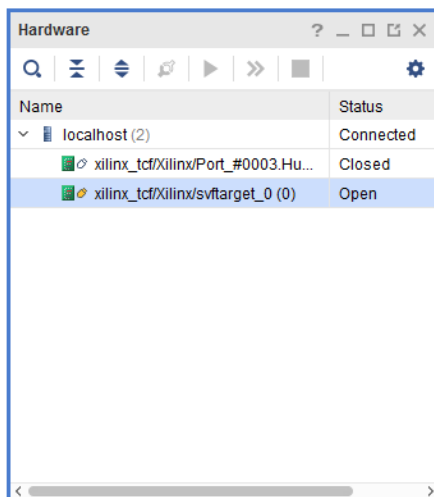
図 58: [Create SVF Target] ダイアログ ボックス



ヒント: [Copy from target] オプションをオンにすると、既存の SVF チェーンがコピーできます。または、フローの前の run から Vivado ハードウェア マネージャーを使用して作成した SVF ファイルを指定できます。Vivado IDE では SVF チェーンの詳細が保存されるので、読み戻すと、SVF チェーンを作成し直すことができます。

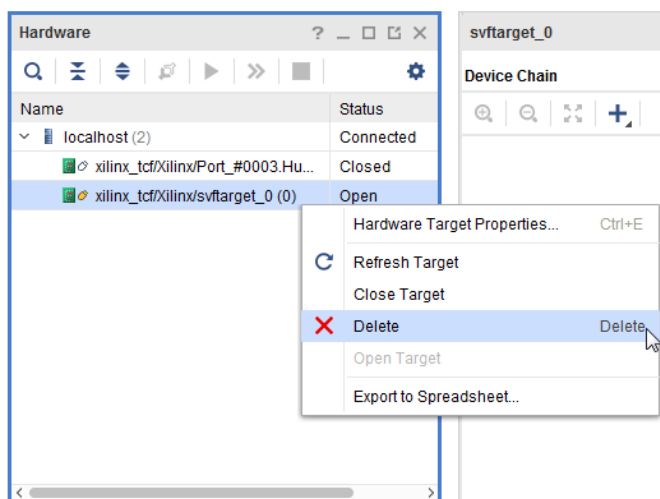
作成した SVF ターゲットが Vivado ハードウェア マネージャーの [Hardware] ウィンドウのサーバーの下に [Open] と表示されます。

図 59: [Hardware] ウィンドウの SVF ターゲット



既存の SVF ターゲットを削除するには、ウィンドウで SVF ターゲットを右クリックし、[Delete] をクリックします。

図 60: [Hardware] ウィンドウでの SVF ターゲットの削除



重要: ターゲットを削除すると、そのターゲット用に作成されたデバイスもすべて削除されます。また、ターゲットが開いていた場合は、削除すると閉じます。

Vivado の Tcl モードまたは Vivado IDE の [Tcl Console] ウィンドウを使用しても SVF ターゲットを作成できます。

Vivado Design Suite または Vivado Lab Edition を起動した後に SVF ターゲットを作成する手順は、次のとおりです。

コマンド ラインの使用

Vivado Design Suite または Vivado Lab Edition を起動した後に SVF ターゲットを作成する手順は、次のとおりです。

```
open_hw_manager
connect_hw_server
create_hw_target my_svf_target
if {[string length [get_hw_targets -quiet -filter
{IS_OPENED == TRUE}]] > 0} \
{close_hw_target [get_hw_targets * -filter {IS_OPENED == TRUE}
] }; \
open_hw_target [get_hw_targets *my_svf_target]
current_hw_target
```

サーバーに既に接続されている場合は、最初の 2 つのコマンドは不要です。create_hw_target コマンドを実行すると、my_svf_target が定義されます。1 つのセッションで同じ名前の 2 つのターゲットを使用することはできません。最後に、開いているターゲットを閉じて SVF ターゲットを開くと、create_hw_target コマンドが実行され、作成された my_svf_target の完全なハードウェア ターゲット ハンドル名が表示されます。

get_hw_targets および open_hw_target コマンドなどのターゲットに対する標準操作がすべてサポートされます。IS_SVF ハードウェア ターゲット プロパティを使用して、ライブ ターゲットと SVF ターゲットを区別できます。たとえば、次のサンプル コマンドは IS_SVF という名前のターゲットから my_svf_target プロパティを読み出します。

```
get_property IS_SVF [get_hw_targets -regexp .*my_svf_target]
```

次のコマンドを使用すると、現在のセッションで作成された SVF hw_targets をすべて表示できます。

```
get_hw_targets -filter {IS_SVF}
```

作成されたターゲットを削除するには、delete_hw_target コマンドを使用します。たとえば、次のコマンドを実行すると my_svf_target が削除されます。

```
delete_hw_target [get_hw_targets -regexp .*my_svf_target]
```



重要: ターゲットを削除すると、そのターゲット用に作成されたデバイスもすべて削除されます。また、ターゲットが開いていた場合は、削除すると閉じます。

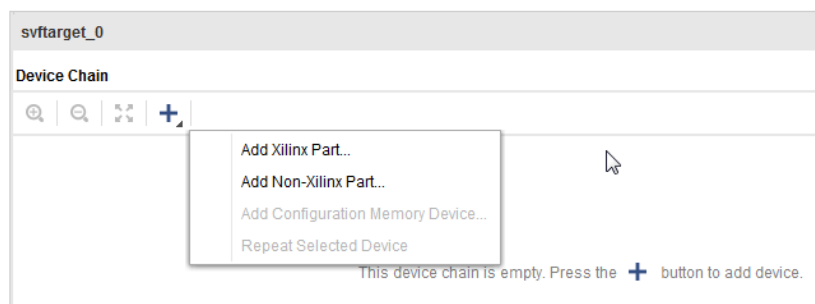
SVF ターゲットへのデバイスの追加

SVF ターゲットを作成したら、デバイスを追加して SVF JTAG デバイス チェーン コンフィギュレーションを定義します。SVF ファイルが正しく実行されるようにするため、SVF JTAG デバイス コンフィギュレーションはターゲット ハードウェア チェーンと一致させる必要があります。

Vivado IDE の使用

+ をクリックすると、ザイリンクスまたはザイリンクス以外のパーツを SVF チェーンに追加できます。

図 61: SVF ターゲットへのデバイスの追加



[Add Xilinx Part] をクリックすると、[Add Xilinx Device] ダイアログ ボックスが開きます。ザイリンクス デバイスを選択して SVF チェーンに追加します。

注記: デバイスは SVF デバイス チェーンにのみ追加できます。

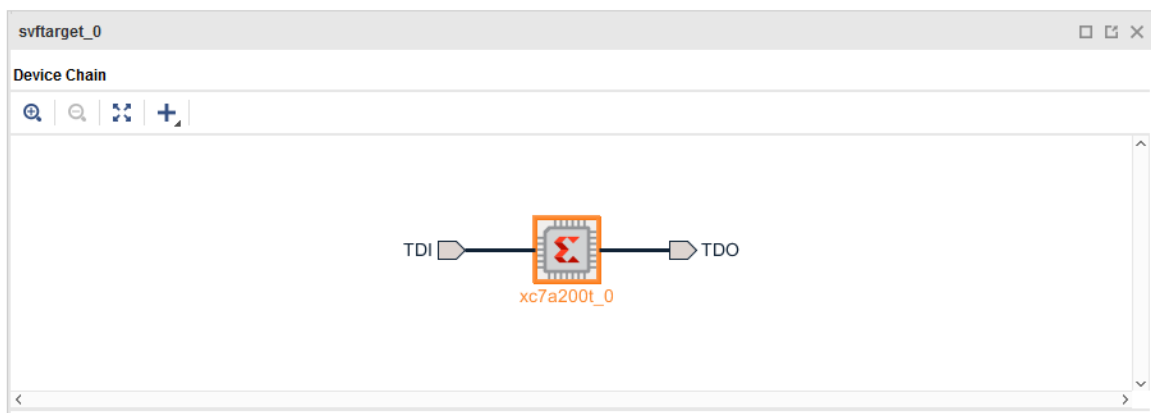
図 62: [Add Xilinx Device] ダイアログ ボックス



ヒント: このダイアログ ボックスは、Vivado Design Edition では少し異なります。

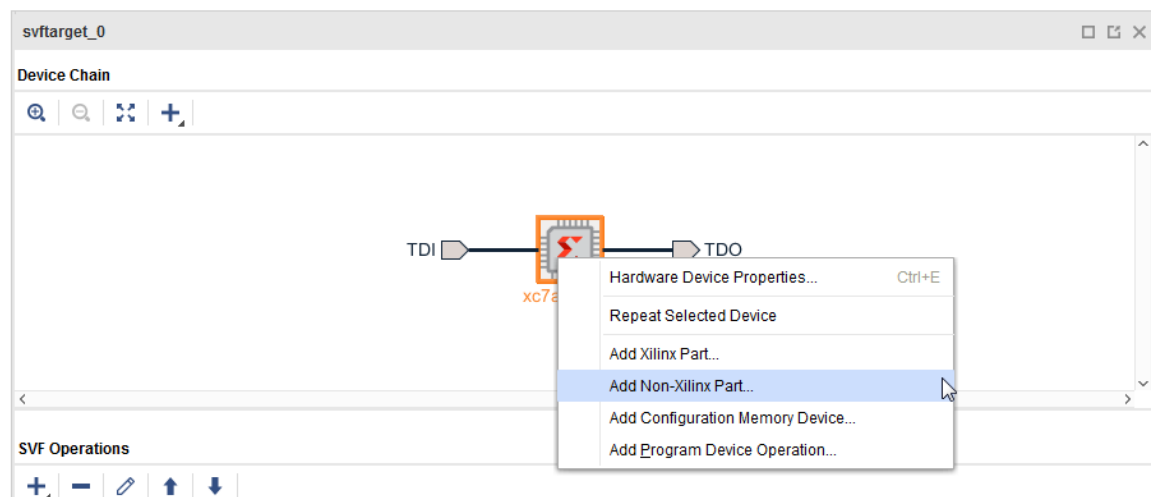
ザイリンクス デバイスを選択して [OK] をクリックすると、次の図のようにそのザイリンクス デバイスが SVF チェーンに追加されます。

図 63: SVF チェーンのザイリンクス デバイス



ザイリンクス以外のパーツを SVF デバイス チェーンを追加するには、SVF チェーンを右クリックし、[Add Non-Xilinx Part] をクリックします。

図 64: ザイリンクス以外のパーツの追加



[Add Non-Xilinx Device] ダイアログ ボックスが開きます。

図 65: [Add Non-Xilinx Device] ダイアログ ボックス



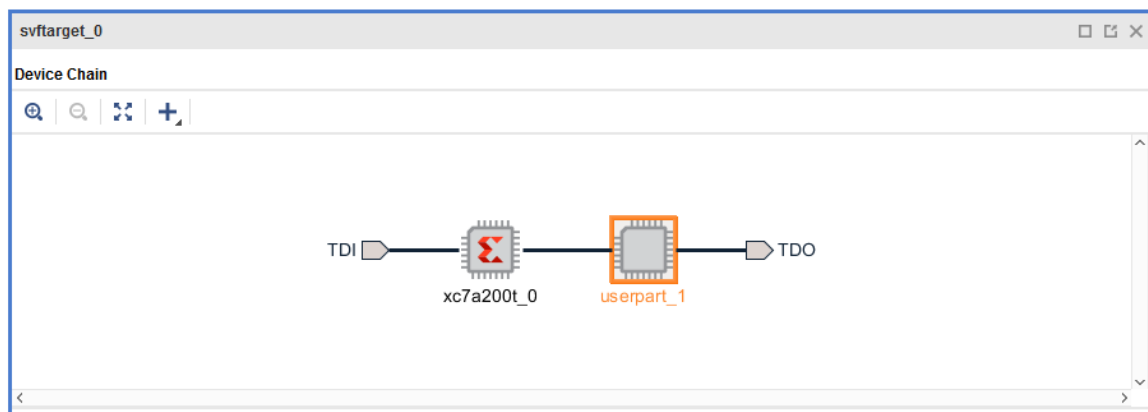
ダイアログ ボックスを次のように設定します。

- [Part Name] は、選択したパーツ名になります。
- [ID Code] は有効なデバイス ID コードを示す 16 進数です。
- [IR length] は命令レジスタ長を示す 10 進数値です。
- [Mask] は 16 進数ビットのマスク値です。

注記: [ID code]、[IR Length]、[Mask] の値は通常、シリコン ベンダーによりデバイス BSDL ファイルで提供されています。

[OK] をクリックすると、サイリンクス以外のパーツが SVF デバイス チェーンに追加されます。

図 66: SVF チェーンのサイリンクス以外のデバイス



コマンド ラインの使用

Vivado Tcl モードまたは Vivado IDE の [Tcl Console] ウィンドウを使用してチェーンを作成するには、SVF ターゲットを開いて `create_hw_device` を連続して実行します。たとえば、`xcku9p` パーツの後に `xcvu095` パーツを追加するには、次のようにコマンドを実行します。

```
current_hw_target my_svf_target
open_hw_target
create_hw_device -part xcku9p
create_hw_device -part xcvu095
refresh_hw_target
get_hw_devices
```

この例で、SVF が既に作成され開いている場合は、最初の 2 つのコマンドは不要です。`create_hw_device` コマンドでは、JTAG チェーンのデバイスを 1 つ目のデバイスから順に定義します。

注記: `create_hw_device` コマンドは、開いている SVF ハードウェア ターゲットにのみデバイスを作成します。

チェーンにユーザー定義のデバイスを追加するには、`-idcode` オプションと共に `-irlength`、`-mask`、および `-part options` オプションを使用します。たとえば、JTAG ID コードが 1234567、IR 幅が 8、マスクが ffffffff の `my_part` というパーツを追加するには、次のコマンドを使用します。

```
open_hw_target [current_hw_target]
create_hw_device -idcode 01234567 -irlength 8 -mask ffffffff -part my_part
# print IR length for user defined devices
puts [get_property IR_LENGTH [lindex [get_hw_devices -filter {PART ==
my_part}] 0]]
puts $idcode_hex
close_hw_target
```

注記: `create_hw_device` の `-idcode` オプションでは、有効なデバイス ID コードを指定する必要があります。ID コードおよび IR 幅は通常、シリコン ベンダーによりデバイス BSDL ファイルで提供されています。

ターゲットおよびそのデバイスをレポートするには、`report_hw_targets` コマンドを使用します。システムのアクティブ ターゲットすべての表際がレポートされます。このコマンドを実行すると、次のようにサーバー、ターゲット、およびデバイスのプロパティがレポートされます。

```
report_hw_targets
INFO: Server Property Information: localhost:3121
  CLASS: hw_server
  HOST: localhost
  NAME: localhost:3121
  PORT: 3121
  SID: TCP:localhost:3121
INFO: Target Property Information: localhost:3121/xilinx-tcf/Xilinx/
my_svf_target
  CLASS: hw_target
  DEVICE_COUNT: 3
  HW_JTAG: 0
  IS_OPENED: 1
  MAX_DEVICE_COUNT: 32
  NAME: localhost:3121/xilinx-tcf/Xilinx/my_svf_target
  FREQUENCY: 10000000
  TYPE: xilinx-tcf
  TID: jsn-XNC-my_svf_target
```

```

UID: Xilinx/my_svf_target
SVF: 1
Device: xcku9p_0
Device: xcvu095_1
Device: my_part_2

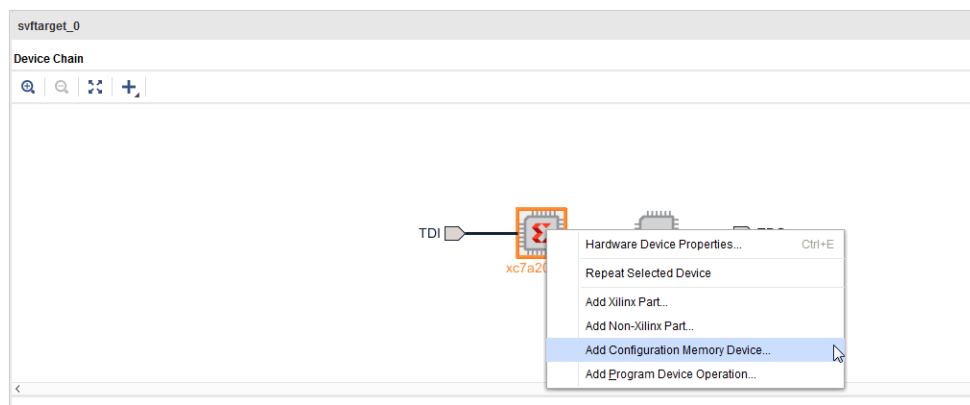
```

ザイリンクス デバイスへのコンフィギュレーション メモリ パーツの追加

Vivado IDE の使用

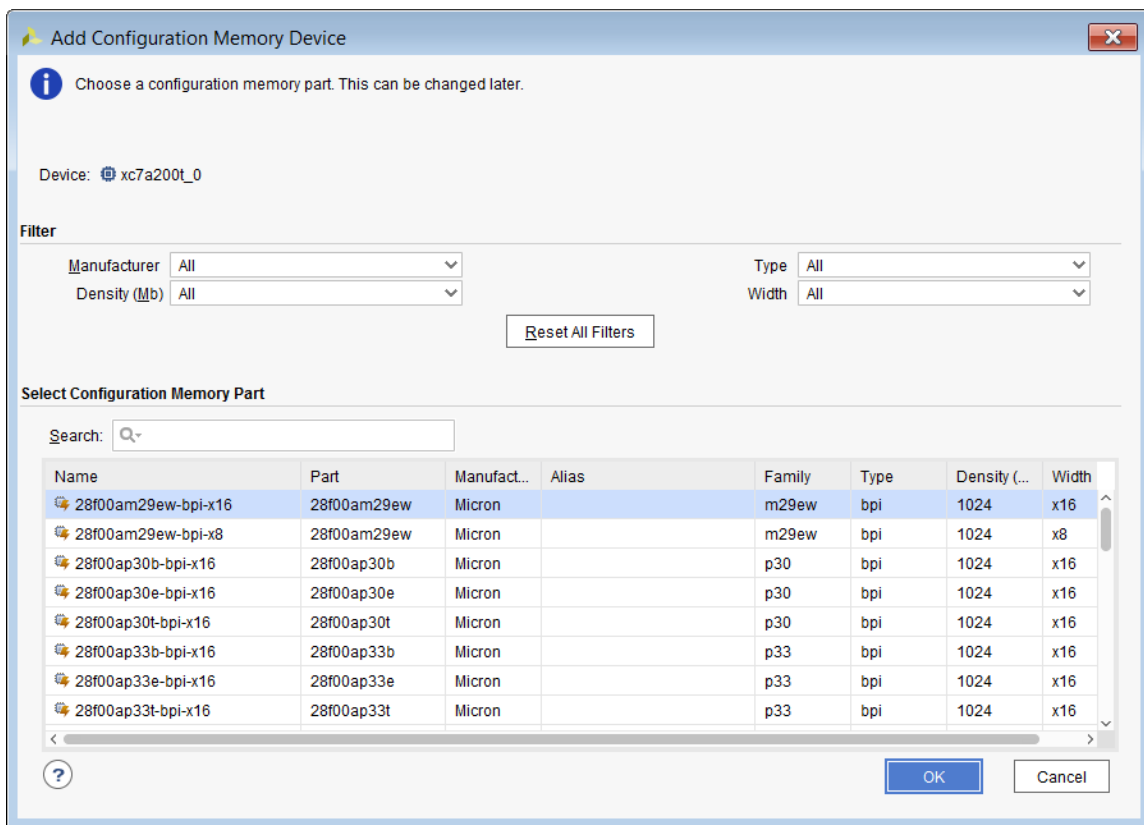
SVF チェーンでザイリンクス デバイス パーツを右クリックすると、コンフィギュレーション メモリ デバイスを作成してそれに関連付けるオプションが表示されます。

図 67: コンフィギュレーション メモリ デバイスの追加



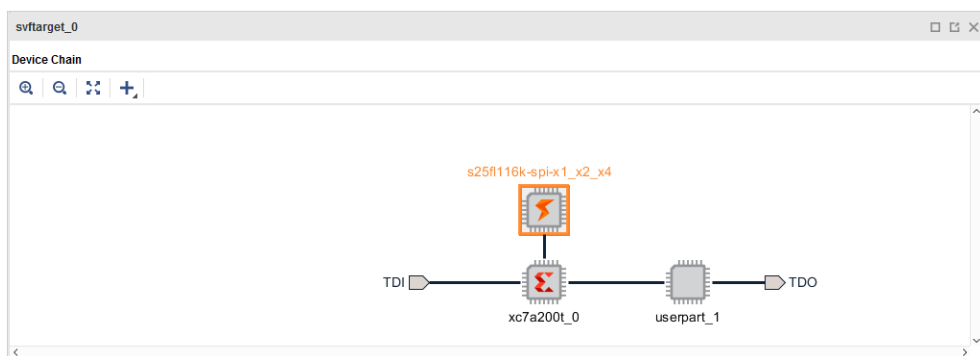
[Add Configuration Memory Device] ダイアログ ボックスが開きます。

図 68: [Add Configuration Memory Device] ダイアログ ボックス



適切なメモリ デバイスを選択し、[OK] をクリックします。デバイスがザイリンクス デバイスに関連付けられると、次のように SVF デバイス チェーンに表示されます。

図 69: SVF チェーンのコンフィギュレーション メモリ バイス



コマンド ラインの使用

Vivado Tcl モードまたは Vivado IDE の [Tcl Console] ウィンドウを使用してコンフィギュレーション メモリ デバイスを作成して関連付けるには、次のように Tcl コマンドの `create_hw_cfgmem` を使用します。

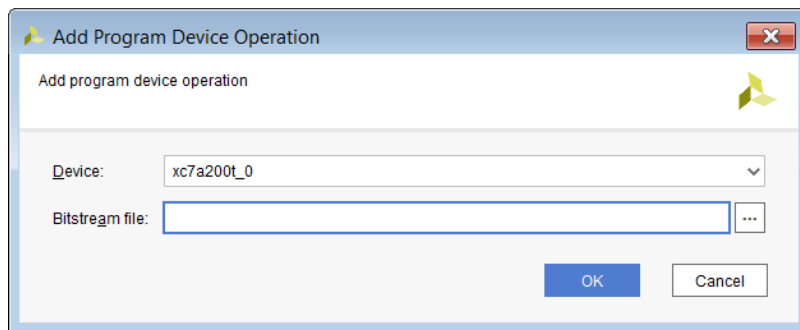
```
create_hw_cfgmem -hw_device [lindex [get_hw_devices xc7a200t_0] 0] [lindex  
[get_cfgmem_parts {s25fl116k-spi-x1-x2-x4}] 0]
```

SVF チェーンでの操作

すべてのデバイスおよびコンフィギュレーション メモリすべてを反映する SVF チェーンを正しい順序で作成したら、SVF チェーンのデバイスにプログラム操作を追加できます。

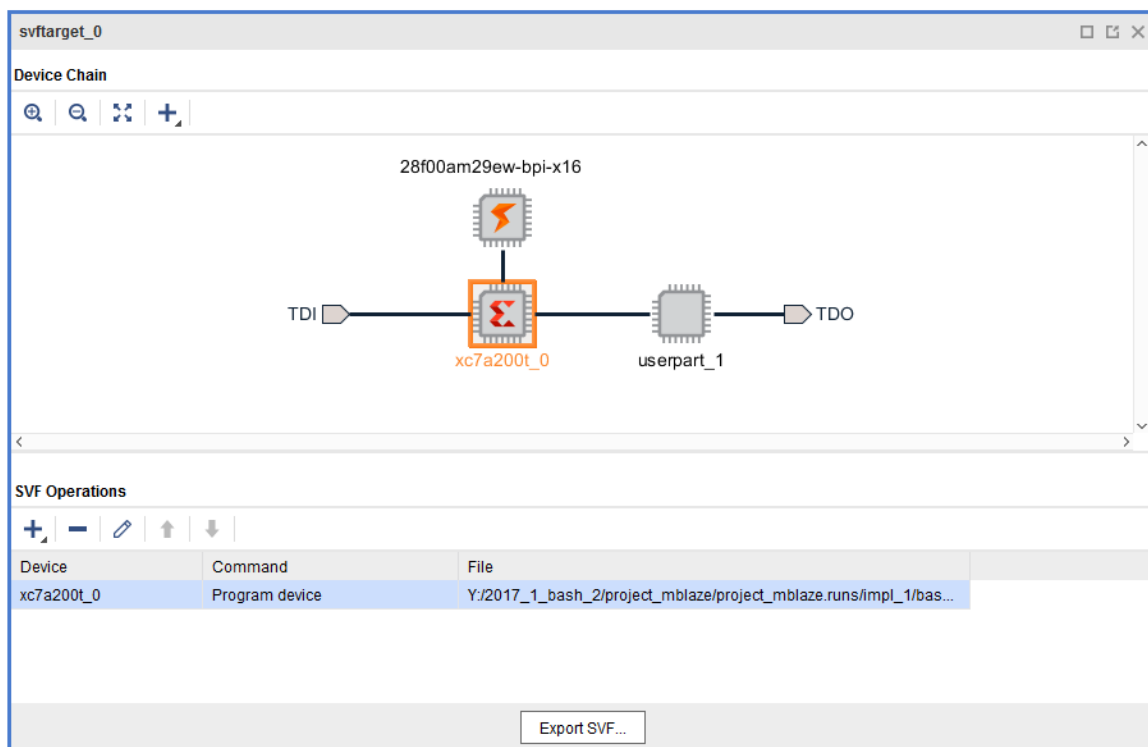
たとえば、チェーンの Xilinx a200t デバイスを右クリックし、[Add Program Device Operation] をクリックし、[Add Program Device Operation] ダイアログ ボックスを表示します。デバイスをプログラムするのに使用するビットストリーム ファイルを指定します。

図 70: [Add Program Device Operation] ダイアログ ボックス



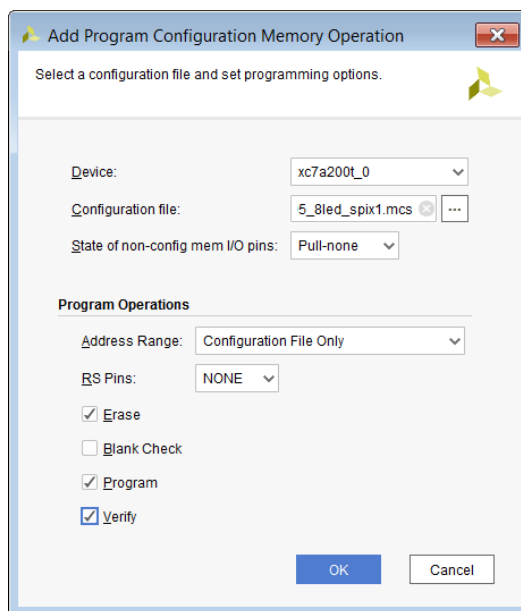
[OK] をクリックすると、プログラム デバイス操作が [SVF Operations] ウィンドウの下にリストされます。

図 71: [SVF Operations] ウィンドウ



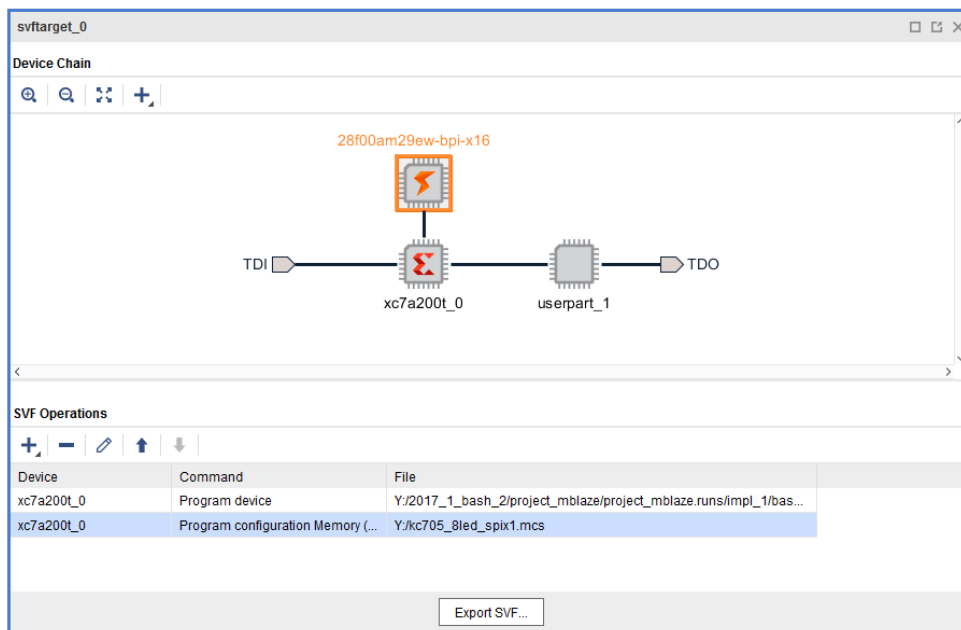
同様に、コンフィギュレーション メモリ デバイスをプログラムするには、メモリ デバイスを右クリックして [Add Program Configuration Memory] をクリックして [Add Program Configuration Memory] ダイアログ ボックスを開きます。メモリ デバイスをプログラムするのに使用するコンフィギュレーション ファイルを指定します。[Erase]、[Blankcheck]、[Verify] などのその他のメモリ デバイスのプログラム オプションも選択できます。

図 72: [Add Program Configuration Memory] ダイアログ ボックス



[OK] をクリックすると、プログラム コンフィギュレーション メモリ デバイス操作が [SVF Operations] ウィンドウの下にリストされます。

図 73: [SVF Operations] ウィンドウのコンフィギュレーション メモリ バイス

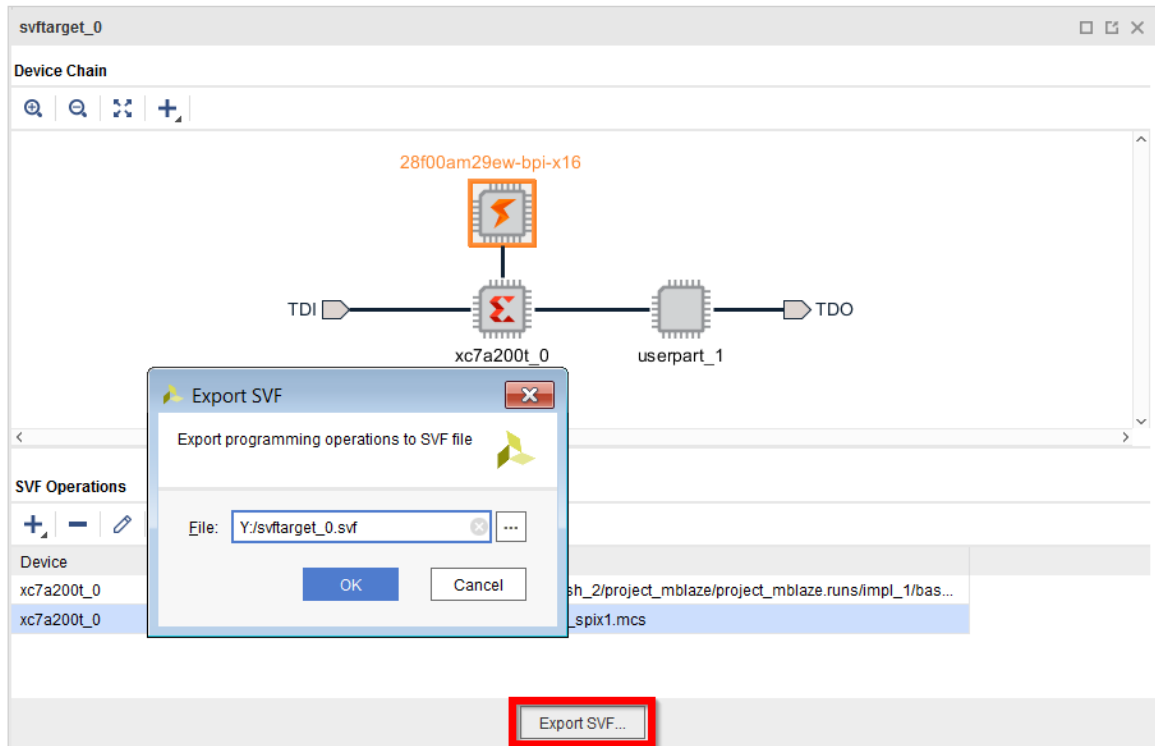


SVF ファイルの保存

Vivado IDE の使用

SVF チェーン設定とその操作は、[SVF Operations] ウィンドウの下で [Export SVF] をクリックするとファイルに保存できます。

図 74: SVF チェーン設定のエクスポート



重要: 既存の SVF チェーンは、フローの前の run から Vivado ハードウェア マネージャーを使用して作成した SVF ファイルを指定して生成し直すことができます。Vivado IDE では SVF チェーンの詳細がファイルに保存されるので、読み戻すと、SVF チェーンを作成し直すことができます。

コマンド ラインの使用

Vivado の Tcl モードまたは Vivado IDE の [Tcl Console] ウィンドウを使用して SVF ファイルを書き出すには、`write_hw_svf` コマンドを使用します。

SVF チェーン、直接 FPGA および間接フラッシュ プログラム操作は、一時ファイルに保存されます。`write_hw_svf` コマンドを実行すると、この一時ファイルがコマンドで指定したファイルに保存されます。`write_hw_svf` コマンドを実行した後は一時ファイルはリセットされ、その後のプログラム操作は SVF ファイル シーケンスの最初に追加されます。

次のコードは、xc7a200t デバイスの直接プログラム操作を含む `my_xc7a200t.svf` という名前のファイルを作成する Tcl コマンドを示しています。

```
create_hw_target my_svf_target
open_hw_target
set device0 [create_hw_device -part xc7a200t]
set_property PROGRAM.FILE {my_xc7a200t.bit} $device0
program_hw_devices $device0
write_hw_svf my_xc7a200t.svf
close_hw_target
```

このサンプル コードは、`create_hw_device` コマンドを使用して `xcku9p` デバイスを作成し、その戻り値を `device0` という一時変数に設定します。この一時変数値を使用してオブジェクトを参照し、`PROGRAM.FILE` プロパティを `my_xcku9p.bit` ファイルに設定します。次に、`program_hw_device` を参照して `device0` コマンドを実行します。この `program_hw_device` コマンドを実行すると、`my_xcku9p.bit` ファイルを `xcku9p` にプログラムするのに必要な SVF 操作を含む一時 SVF ファイルが作成されます。最後に `write_hw_svf` コマンドを実行し、一時ファイルを `myxcku9p.svf` ファイルに移動します。この段階で SVF ファイルの作成プロセスは終了したので、ターゲットを閉じることができます。



ヒント: SVF ファイルを記述する際、JTAG チェーンのすべてのデバイスを作成してからプログラム操作を実行してください。プログラム コマンドの間に `create_hw_device` コマンドを挿入すると、2 つの異なるチェーンシーケンスを含む SVF ファイルが生成されます。

- 間違った SVF ファイル作成手順の例:

```
create_hw_target my_svf_target
open_hw_target
set device0 [create_hw_device -part xcku9p]
set_property PROGRAM.FILE {my_xcku9p1.bit} $device0
# this program command will produce SVF instructions
# which account for only device0 in chain
program_hw_devices $device0
set device1 [create_hw_device -part xcku9p]
set_property PROGRAM.FILE {my_xcku9p2.bit} $device1
# this program command will produce SVF instructions
# which account for device0 and device1 in chain
program_hw_devices $device1
write_hw_svf my_bad_xcku9p.svf
close_hw_target
```

最初のプログラム コマンドでは、最初のデバイスを含むチェーン定義のみが使用されます。2 番目のプログラム コマンドにはチェーンの両方のデバイスが含まれます。この SVF ファイルを 2 つのデバイスを含むチェーンで実行すると、ライブ チェーンには 2 つのデバイスがあるので、最初のプログラムはエラーとなります。

この問題を修正するには、`create_hw_device` コマンドを先に実行してチェーンを完全に定義した後、プログラムを実行します (次の例を参照)。

- 正しい SVF ファイル作成手順の例:

```
create_hw_target my_svf_target
open_hw_target
# create device chain first
set device0 [create_hw_device -part xcku9p]
set device1 [create_hw_device -part xcku9p]
# program device0
set_property PROGRAM.FILE {my_xcku9p1.bit} $device0
program_hw_devices $device0
# program device1
set_property PROGRAM.FILE {my_xcku9p2.bit} $device1
program_hw_devices $device1
write_hw_svf my_good_xcku9p.svf
close_hw_target
```

SVF ファイルの実行

SVF ファイルを作成したら Vivado IDE を使用して実行できます。Vivado IDE では、SVF ファイル生成機能で生成された SVF ファイルを実行でき、検証テスト ツールとして使用することを意図しています。execute_hw_svf コマンドは汎用 SVF 実行コマンドではないので、Vivado IDE で作成された SVF ファイルのみを実行するようにしてください。

ライブ ターゲットに対して svf コマンドを実行するには、次のコマンドを使用します。

```
execute_hw_svf my_file.svf
INFO: [Labtoolstcl 44-548] Creating JTAG TCL script from SVF file
INFO: [Labtoolstcl 44-549] Re-opening target in JTAG mode
INFO: [Labtoolstcl 44-551] Sourcing JTAG TCL script: my_file.tcl
Pass: SVF Execution completed with no errors
INFO: [Labtoolstcl 44-550] Restoring target to original mode
INFO: [Labtoolstcl 44-570] Execute SVF completed successfully
```

この例では、my_file.svf を実行しています。実行フローの一部として、入力 SVF ファイルが HW_JTAG Tcl 操作により一時ファイルに変換されます。この Tcl コードが作成された後、ファイルが読み込まれて、変換された SVF 命令が実行されます。JTAG_TCL 操作を確認するには、execute_hw_svf コマンドを -verbose オプションを使用して実行します。コマンドが完了すると、実行できなかった命令でエラーが表示されるか、メッセージ ログの最後に「Pass」メッセージが表示されてコマンドが正しく実行されたことが示されます。

デザインのデバッグ

FPGA デザインのデバッグは複数の手順の繰り返しプロセスです。複雑な問題を処理する場合と同様に、FPGA デザインのデバッグ プロセスも、一度にデザイン全体を処理するのではなく、細分化してセクションごとに集中して作業するのがベストです。1 回に 1 モジュールを追加しながらデザイン フローを反復し、デザイン全体の中でそれを正しく機能させるようにするのが、実績のあるデザインおよびデバッグ手法の 1 つです。この設計およびデバッグ手法は、次のデザイン フロー段階のどの組み合わせでも使用できます。

- RTL レベルのデザイン シミュレーション
- インプリメンテーション後のデザイン シミュレーション
- インシステム デバッグ

RTL レベルのデザイン シミュレーション

シミュレーション検証プロセス中にデザインの機能をデバッグできます。ザイリンクスの Vivado[®] IDE では、フル デザイン シミュレーション機能が提供されています。デザインの RTL シミュレーションを実行するには、Vivado デザイン シミュレータを使用できます。RTL レベル シミュレーション環境でデザイン デバッグを実行すると、デザイン全体を完全に表示でき、デザイン/デバッグ サイクルをすばやく反復実行できるなどの利点がありますが、大型デザインを妥当な時間内にシミュレーションしたり、実際のシステム環境を正確にシミュレーションするのが困難であるなどの制限があります。Vivado シミュレータの使用に関する詳細は、『Vivado Design Suite ユーザー ガイド: ロジック シミュレーション』(UG900) を参照してください。

インプリメンテーション後のデザイン シミュレーション

Vivado シミュレータは、インプリメンテーション後のデザイン シミュレーションにも使用できます。Vivado シミュレータを使用してインプリメンテーション後のデザインをデバッグすると、デザインのタイミング精度の高いモデルを使用できるなどの利点がありますが、前のセクションで述べたように、ランタイムが長いことや、システム モデルでの正確さなどの制限があります。

インシステム ロジック デザインのデバッグ

Vivado Design Suite には、インプリメンテーション後の FPGA デザインをインシステムでデバッグできるロジック解析機能もあります。インシステムでのデザインのデバッグには、インプリメンテーション後のデザインを実際のシステム環境で、タイミング精度の高いデバッグをシステム スピードで実行できるという利点がありますが、シミュレーション モデルを使用した場合に比べてデバッグ信号を確認しづらい、デザインのサイズや複雑さによってデザイン/インプリメンテーション/デバッグの反復実行のランタイムが長くなるなどの制限があります。

Vivado ツールでは複数のデバッグ方法が提供されているので、必要に応じて、これらの方法のいずれかを使用してデザインをデバッグできます。Vivado Design Suite のインシステム ロジック デバッグ機能については、「インシステム ロジック デザインのデバッグ フロー」を参照してください。

関連情報

[インシステム ロジック デザインのデバッグ フロー](#)

インシステム シリアル I/O デザインのデバッグ

インシステム シリアル I/O の検証およびデバッグを可能にするため、Vivado Design Suite にはシリアル I/O 解析機能が含まれています。この機能を使用すると、FPGA ベース システムの高速シリアル I/O リンクを計測および最適化できます。Vivado シリアル I/O 解析機能は、単純なクロックや接続の問題から複雑なマージン解析およびチャネル最適化の問題まで、さまざまなインシステム デバッグおよび検証の問題を解決するために使用できます。Vivado シリアル I/O 解析機能を使用すると、ほかの外部装置を使用するのと比較して、受信信号にレシーバー イコライゼーションが適用された後の信号の質が計測されるという利点があります。これにより、TX から RX へのチャネルの最適なポイント、つまり実際の正しいデータが計測されます。

Vivado ツールでは、ギガビット トランシーバー エンドポイントを実行するために使用されるデザインおよびランタイム ソフトウェアを生成でき、高速シリアル I/O チャネルを計測し、最適化するのに利用できます。IBERT デザインを生成する方法は、「シリアル I/O ハードウェア デバッグ フロー」を参照してください。ランタイム Vivado シリアル I/O 解析機能の使用方法は、「ハードウェアでのシリアル I/O デザインのデバッグ」を参照してください。

関連情報

[シリアル I/O ハードウェア デバッグ フロー](#)

[ハードウェアでのシリアル I/O デザインのデバッグ](#)

インシステム ロジック デザインのデバッグ フロー

Vivado[®] ツールには、実際のハードウェア デバイス上でデザインのインシステム デバッグを実行する機能が多数含まれています。インシステム デバッグ フローには、次の 3 つの段階があります。

1. **プローブ フェーズ:** デザインでプローブする信号を特定し、プローブ方法を指定します。
2. **インプリメンテーション フェーズ:** プローブするネットに追加されたデバッグ IP を含むデザインをインプリメントします。
3. **解析フェーズ:** デザインに含まれるデバッグ IP にアクセスし、機能的な問題をデバッグおよび検証します。

このインシステム デバッグ フローは、前のセクションで説明した反復デザイン/デバッグ フローを使用することを意図しています。インシステム デバッグ フローを使用する場合は、デザイン サイクルのできるだけ早い段階で、デザインの一部がハードウェアで機能するようにすることをお勧めします。この章では、インシステム デバッグ フローの 3 つの段階を説明し、Vivado ロジック デバッグ機能を使用してデザインがハードウェア上で機能する方法を示します。

インシステム デバッグ用のデザインのプローブ

インシステム デバッグ フローのプローブ段階には、次の 2 つの段階があります。

1. プローブする信号またはネットを特定
2. デザインにデバッグ コアを追加する方法を決定

多くの場合、プローブする信号およびそのプローブ方法は、ほかの信号のプローブに影響します。まず、デザイン ソース コードにデバッグ IP コンポーネント インスタンスを手動で追加するか (HDL インスタンスエーション プローブ フロー)、合成済みネットリストに Vivado ツールで自動的にデバッグ コアが追加されるようにするか (ネットリスト 挿入プローブ フロー) を決定すると有益です。次の表に、異なるデバッグ方法の利点と欠点を示します。

表 3: デバッグ ストラテジ

デバッグ目標	推奨デバッグ プログラム フロー
HDL ソース コードでデバッグ信号を特定し、フローの後の方でデバッグをイネーブル/ディスエーブルにできるようにする。	mark_debug プロパティを使用して、HDL でデバッグ用の信号にタグを付ける。 Set Up Debug ウィザードを使用して、ネットリスト挿入プローブ フローを実行する。
HDL ソース コードを変更せずに、合成済みデザイン ネットリストでデバッグ ネットを特定する。	合成済みデザイン ネットリストでネットを右クリックして [Mark Debug] をクリックし、デバッグするネットを選択する。 Set Up Debug ウィザードを使用して、ネットリスト挿入プローブ フローを実行する。

表 3: デバッグ ストラテジ (続き)

デバッグ目標	推奨デバッグ プログラム フロー
Tcl コマンドを使用してデバッグ プロープ フローを自動化する。	<code>set_property</code> Tcl コマンドを使用し、デバッグするネットに <code>mark_debug</code> プロパティを設定する。 ネットリスト挿入プロープ フロー用の Tcl コマンドを使用し、デバッグ コアを作成してデバッグ ネットに接続する。
HDL ソースで ILA デバッグ コア インスタンスに信号を明示的に接続する。	デバッグする HDL 信号を特定する。 HDL インスタンスエーション プロープ フローを使用し、ILA (Integrated Logic Analyzer) コアを生成してインスタンスエートし、デザインのデバッグ信号に接続する。

ネットリスト挿入デバッグ プロープ フロー

Vivado ツールでデバッグ コアを挿入する方法は、さまざまなニーズに対応できるよう複数あります。

- シンプルなウィザードを使用し、デバッグするネットに基づいて、ILA (Integrated Logic Analyzer) コアを自動的に生成および設定します。これが一番簡単な方法です。
- [Debug] ウィンドウを使用して、個々のコア、ポート、およびパラメーターを設定します。[Debug] ウィンドウは、合成済みデザインを開いた状態でレイアウト セレクターまたは [Layout] メニューから [Debug] レイアウトを選択するか、[Window]→[Debug] をクリックすると表示されます。
- Tcl の XDC デバッグ コマンドのセットを手動で XDC 制約ファイルに入力するか、Tcl スクリプトを作成します。

これらの方法を組み合わせて利用し、デバッグ コアを挿入およびカスタマイズすることもできます。

デバッグする HDL 信号のマーク

合成の前に HDL ソース レベルでデバッグする信号を特定するには、`mark_debug` 制約を使用します。HDL でデバッグ用にマークされた信号に対応するネットが、[Debug] ウィンドウの [Unassigned Debug Nets] の下に表示されます。

注記: [Debug] ウィンドウの [Debug Nets] ビューはデバッグに選択したネットのネット中心の表示で、[Debug Cores] ビューはコア プロパティを表示および設定可能なコア中心の表示です。

デバッグ用にネットをマークする方法は、プロジェクトが RTL ソース ベースであるか合成済みネットリスト ベースであるかによって異なります。RTL ネットリスト ベースのプロジェクトの場合は、次の方法を使用します。




- Vivado 合成を使用する場合、VHDL および Verilog ソース ファイルで `mark_debug` 制約を使用してデバッグ用のネットをマークできます。`mark_debug` 制約に有効な値は、TRUE または FALSE です。Vivado 合成では、この制約の値を SOFT に設定することはできません。

合成済みネットリスト ベースのプロジェクトの場合は、次の方法を使用します。

- Synopsis® 社の Synplify® 合成ツールを使用すると、VHDL または Verilog で `mark_debug` および `syn_keep` 制約を使用するか、SDC (Synopsys Design Constraints) ファイルでは `mark_debug` 制約を使用して、デバッグ用にネットをマークできます (オプション)。Synplify では SOFT 値はサポートされません。これは、この動作が `syn_keep` 制約で制御されるためです。
- Mentor Graphics® 社の Precision® 合成ツールを使用すると、VHDL または Verilog で `mark_debug` 制約を使用してデバッグ用にネットをマークできます。

次のセクションに、Vivado 合成、XST、Synplify、および Precision ソース ファイルの構文例を示します。

ICON および ILA コア

- 白抜きになっている緑色のアイコン  は、ネットに MARK_DEBUG プロパティが設定されていますが、ILA コアにはネットが接続されていない状態を示します。
- 緑色のアイコン  は、ネットに MARK_DEBUG プロパティが設定されていて、ILA コアにネットが接続されている状態を示します。
- 黄色のアイコン  は、ネットに MARK_DEBUG は設定されていませんが、ILA コアにネットが接続されている状態を示します。

Vivado 合成での mark_debug の構文例

次に、Vivado 合成を使用する場合の VHDL および Verilog の構文例を示します。

- VHDL の構文例

```
attribute mark_debug : string;
attribute mark_debug of char_fifo_dout: signal is "true";
```

- Verilog の構文例

```
(* mark_debug = "true" *) wire [7:0] char_fifo_dout;
```

Synplify での mark_debug の構文例

次に、Synplify を使用する場合の VHDL、Verilog、SDC の構文例を示します。

- VHDL の構文例

```
attribute syn_keep : boolean;
attribute mark_debug : string;
attribute syn_keep of char_fifo_dout: signal is true;
attribute mark_debug of char_fifo_dout: signal is "true";
```

- Verilog の構文例

```
(* syn_keep = "true", mark_debug = "true" *) wire [7:0] char_fifo_dout;
```

- SDC の構文例

```
define_attribute {n:char_fifo_din[*]} {mark_debug} {"true"}
define_attribute {n:char_fifo_din[*]} {syn_keep} {"true"}
```



重要: SDC ソースのネット名には、接頭辞として n: を付ける必要があります。

注記: SDC (Synopsys Design Constraints) は、特にタイミング解析において設計の要件をツールに渡すための業界標準です。SDC 仕様のリファレンス コピーは、Synopsys 社のサイト (<https://www.synopsys.com/Community/Interoperability/Pages/TapinSDC.aspx>) から登録をすると入手できます。

Precision での mark_debug の構文例

次に、Precision を使用する場合の VHDL、Verilog、XCF の構文例を示します。

- VHDL の構文例

```
attribute mark_debug : string;
attribute mark_debug of char_fifo_dout: signal is "true";
```

- Verilog の構文例

```
(* mark_debug = "true" *) wire [7:0] char_fifo_dout;
```

デザインの合成

次に、Vivado Design Suite で [Run Synthesis] をクリックするか、次の Tcl コマンドを使用して、デバッグ コアを含むデザインを合成します。

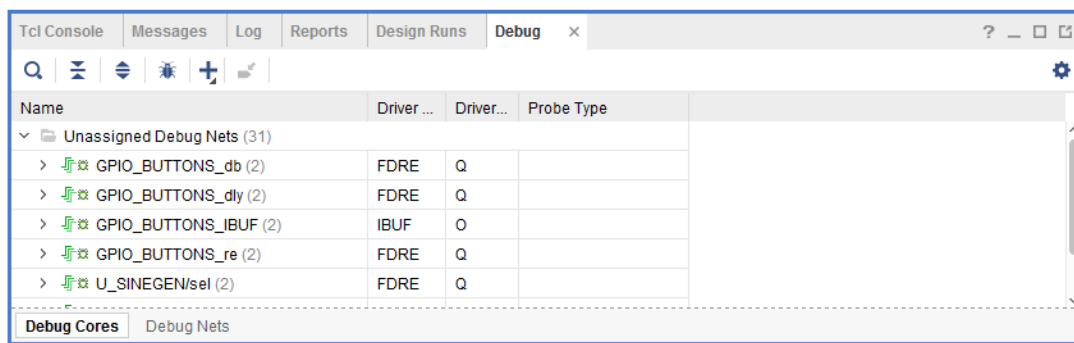
```
launch_runs synth_1
wait_on_run synth_1
```

synth_design Tcl コマンドを使用してデザインを合成することもできます。さまざまな合成方法の詳細は、『Vivado Design Suite ユーザー ガイド: 合成』 ([UG901](#)) を参照してください。

合成済みデザインでデバッグ用のネットをマーク

Flow Navigator で [Open Synthesized Design] をクリックして合成済みデザインを開き、[Debug] レイアウトを選択して [Debug] ウィンドウを表示します。デバッグ用にマークした HDL 信号に対応するネットが、[Debug] ウィンドウの [Unassigned Debug Nets] フォルダーの下に表示されます。

図 75: [Debug] ウィンドウの [Unassigned Debug Nets] フォルダー



- [Netlist]、[Schematic] などのデザイン ウィンドウでネットを右クリックし、[Mark Debug] をクリックします。
- デザイン ビューのいずれかでネットを選択し、そのネットを [Unassigned Debug Nets] フォルダーにドラッグ アンド ドロップします。
- [Set up Debug] ウィザードでネット セレクター使用します。詳細は、「Set Up Debug ウィザードを使用したデバッグ コアの挿入」を参照してください。

関連情報

[Set Up Debug ウィザードを使用したデバッグ コアの挿入](#)

Set Up Debug ウィザードを使用したデバッグ コアの挿入

デバッグ用にネットをマークしたら、それらのネットをデバッグ コアに割り当てます。Vivado Design Suite の Set Up Debug ウィザードを使用すると、デバッグ コアを自動作成し、デバッグ ネットをコアの入力に割り当てることができます。

Set Up Debug ウィザードを使用してデバッグ コアを挿入するには、次の手順に従います。

1. [Debug] ウィンドウの [Unassigned Debug Nets] フォルダーを使用するか、ネットを直接クリックして、デバッグするネットを選択します (オプション)。
2. Vivado Design Suite のメイン メニューから [Tools] → [Set up Debug] をクリックするか、Flow Navigator の [Synthesized Design] セクションの [Set up Debug] をクリックします。
3. [Next] をクリックします。[Specify Nets to Debug] ページが開きます。
4. オプションで、さらにネットを追加したり、一覧から既存のネットを削除するには、[Find Nets to Add] をクリックします。デバッグ ネットを右クリックして [Remove Nets] をクリックしても、表からネットを削除できます。



重要: [Netlist] またはその他のウィンドウでネットをクリックして、[Nets to Debug] のリストにドラッグすることもできます。

5. デバッグ ネットを右クリックして [Select Clock Domain] をクリックし、ネットの値をサンプリングするクロック ドメインを変更します。

注記: Set Up Debug ウィザードは、同期エレメントのパスを検索し、デバッグ ネットに最適なクロック ドメインを自動的に選択しようとします。この選択は必要に応じて [Select Clock Domain] ダイアログ ボックスで変更できますが、表に含まれる各クロック ドメインはそれぞれ別の ILA コア インスタンスになることに注意してください。



ヒント: ILA コアのタイミングへの影響を最小限に抑えるためのヒントは、『UltraFast 設計手法ガイド (Vivado Design Suite 用)』 (UG949) の「[ILA コアとタイミングに関する考慮事項](#)」を参照してください。

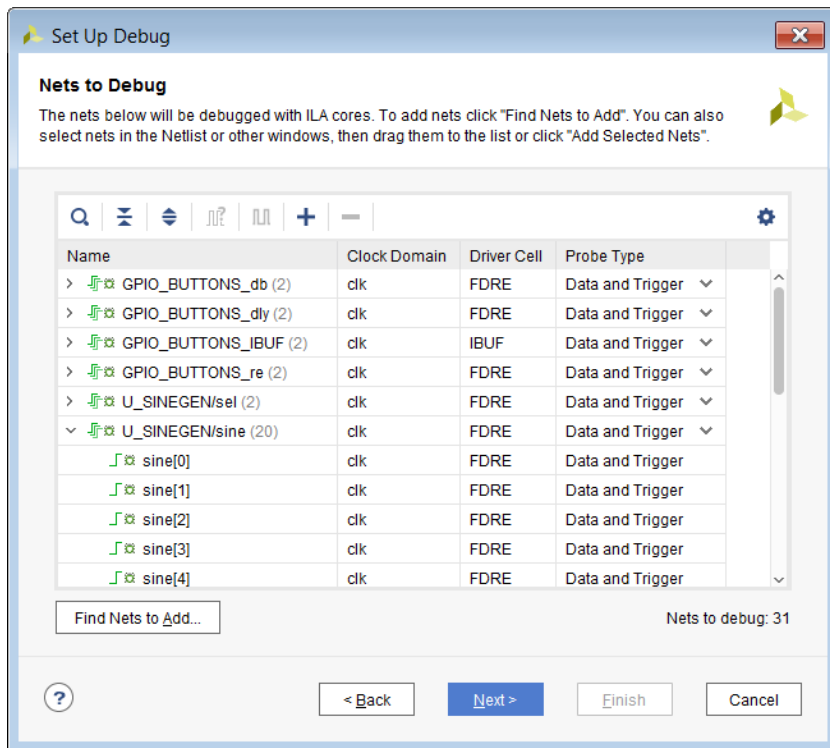
6. デバッグ ネットの選択が完了したら、[Next] をクリックします。

注記: Set Up Debug ウィザードにより、クロック ドメインにつき 1 つの ILA コアが挿入されます。デバッグのために選択されたネットは、挿入された ILA コアのプローブ ポートに自動的に割り当てられます。ウィザードの最終ページはコア生成のサマリ ページで、検出されたクロック数、生成および削除される ILA コアの数が表示されます。

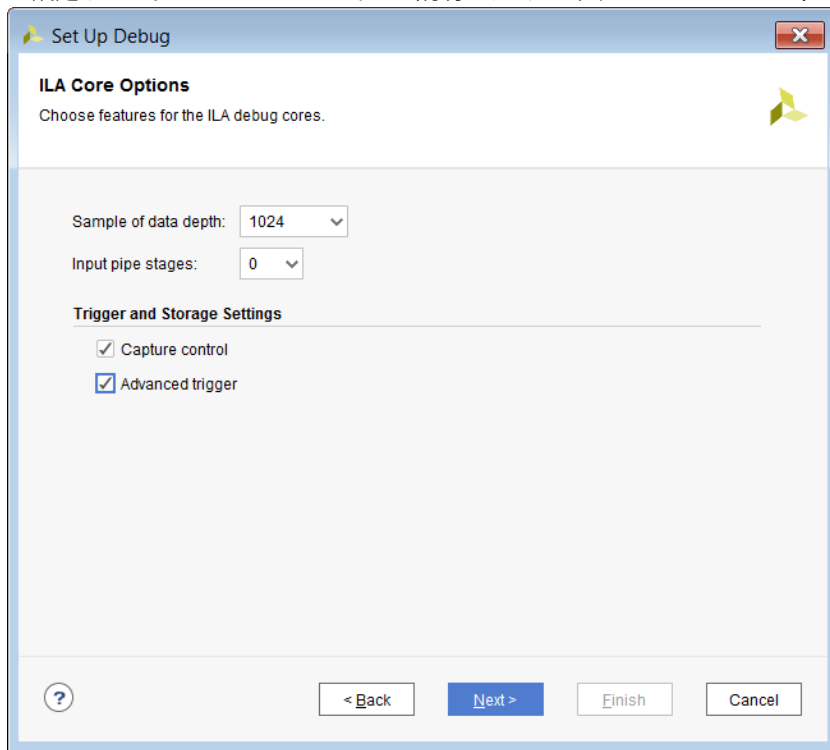
7. ADVANCED トリガー モードをイネーブルにする場合は [Advanced Trigger] チェック ボックスをオンに、BASIC キャプチャ モードをイネーブルにする場合は [Capture Control] チェック ボックスをオンにします。[Next] をクリックし、最後のページに進みます。

注記: Vivado ハードウェア マネージャーで使用する際の ADVANCED トリガー モードまたは BASIC キャプチャ モードについては、第 11 章「ハードウェアでのロジック デザインのデバッグ」を参照してください。

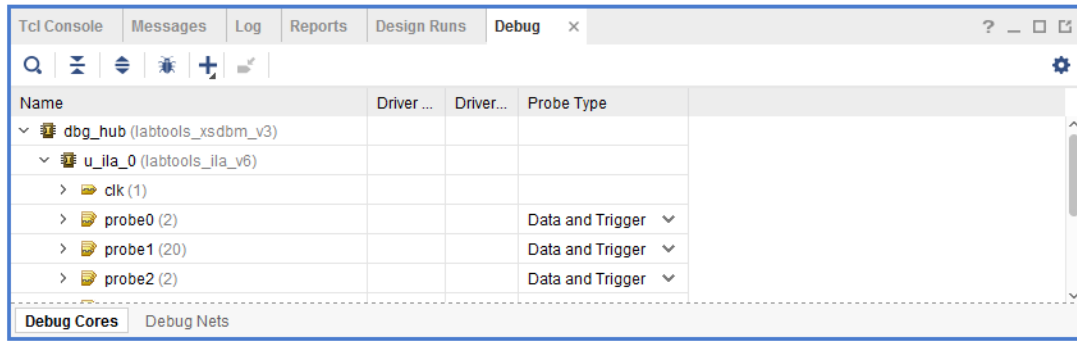
8. 内容を確認したら [Finish] をクリックし、合成済みデザイン ネットリストに ILA コアを挿入および接続します。



9. [ILA (Integrated Logic Analyzer) General Options] ページで ILA データ深さ ([Sample of Data Depth]、C_DATA_DEPTH)、入力パイプ段数 ([Input Pipe Stages]、C_INPUT_PIPE_STAGES)、キャプチャ制御機能のイネーブル ([Capture Control]、C_EN_STRG_QUAL)、ADVANCED トリガー機能 ([Advanced Trigger]、C_ADV_TRIGGER) を設定します。これらのオプションの説明は、「デバッグ コアのプロパティの変更」を参照してください。



10. これで、デバッグ ネットが ILA デバッグ コアに割り当てられました。



関連情報

[ILA コアとタイミングに関する考慮事項](#)

[ハードウェアでのロジック デザインのデバッグ](#)

[デバッグ コアのプロパティの変更](#)

[Debug] ウィンドウを使用したデバッグ コアの追加とカスタマイズ

[Debug] ウィンドウの [Debug Cores] ビューでは、Set Up Debug ウィザードにない ILA コアおよびデバッグ コア ハブの挿入に関する詳細な設定を制御できます。コアの生成および削除、デバッグ ネットの接続、コア パラメーターの設定などを実行できます。

[Debug] ウィンドウの [Debug Cores] ビューには、次のものが表示されます。

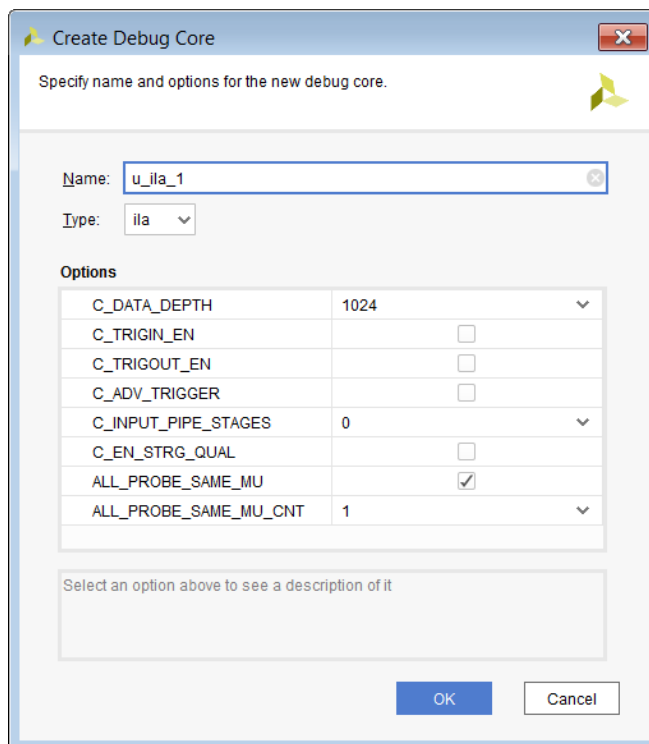
- Debug Hub (dbg_hub) コアに接続されているデバッグ コアのリスト。
- 割り当てられていないネットのリスト。

デバッグ コアおよびポートは、ポップアップ メニューまたはウィンドウの上部にあるツールバーから制御できます。

デバッグ コアの生成および削除

[Debug] ウィンドウでデバッグ コアを割くするには、[Create Debug Core] をクリックします。[Create Debug Core] ダイアログ ボックスを使用すると、親インスタンスおよびデバッグ コア名を変更したり、コアのパラメーターを設定できます。既存のデバッグ コアを削除するには、[Debug] ウィンドウでコアを右クリックして、[Delete] をクリックします。[Create Debug Core] ダイアログ ボックスに表示される ILA コアのオプションの説明は、「デバッグ コアのプロパティの変更」を参照してください。

図 76: [Create Debug Core] ダイアログ ボックス



Specify name and options for the new debug core.

Name:

Type:

Options

C_DATA_DEPTH	1024	▼
C_TRIGIN_EN		<input type="checkbox"/>
C_TRIGOUT_EN		<input type="checkbox"/>
C_ADV_TRIGGER		<input type="checkbox"/>
C_INPUT_PIPE_STAGES	0	▼
C_EN_STRG_QUAL		<input type="checkbox"/>
ALL_PROBE_SAME_MU		<input checked="" type="checkbox"/>
ALL_PROBE_SAME_MU_CNT	1	▼

Select an option above to see a description of it

OK Cancel

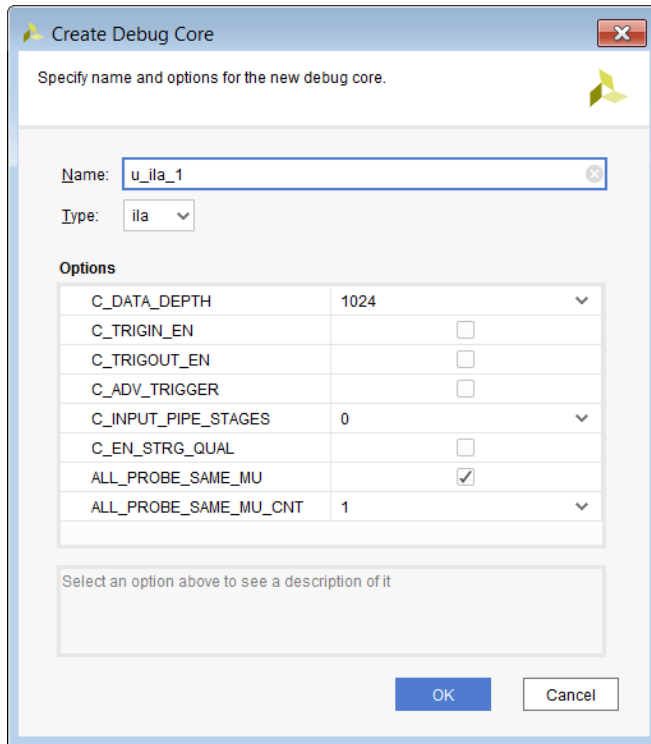
関連情報

[デバッグ コアのプロパティの変更](#)

デバッグ コア ポートの追加、削除、およびカスタマイズ

デバッグ コアの追加および削除だけでなく、各デバッグ コアのポートを追加、削除、およびカスタマイズできます。デバッグ ポートを追加するには、次の手順に従います。

1. [Debug] ウィンドウでデバッグ コアを選択します。
2. ツールバーの [Create Debug Port] をクリックします。[Create Debug Port] ダイアログ ボックスが表示されます。
3. ポート幅を指定します。
4. [OK] をクリックします。
5. デバッグ ポートを削除するには、[Debug] ウィンドウでポートを右クリックし、[Delete] をクリックします。



デバッグ コアへのネットの接続および接続解除

ネットおよびバス (バス ネット) を [Schematic] または [Netlist] ウィンドウからデバッグ コアのポートにドラッグアンドドロップできます。ネットの選択に応じてポートが自動的に拡張されます。また、ネットまたはバスを右クリックし、[Assign to Debug Port] をクリックしても、ネットまたはバスをデバッグ ポートに割り当てることができます。

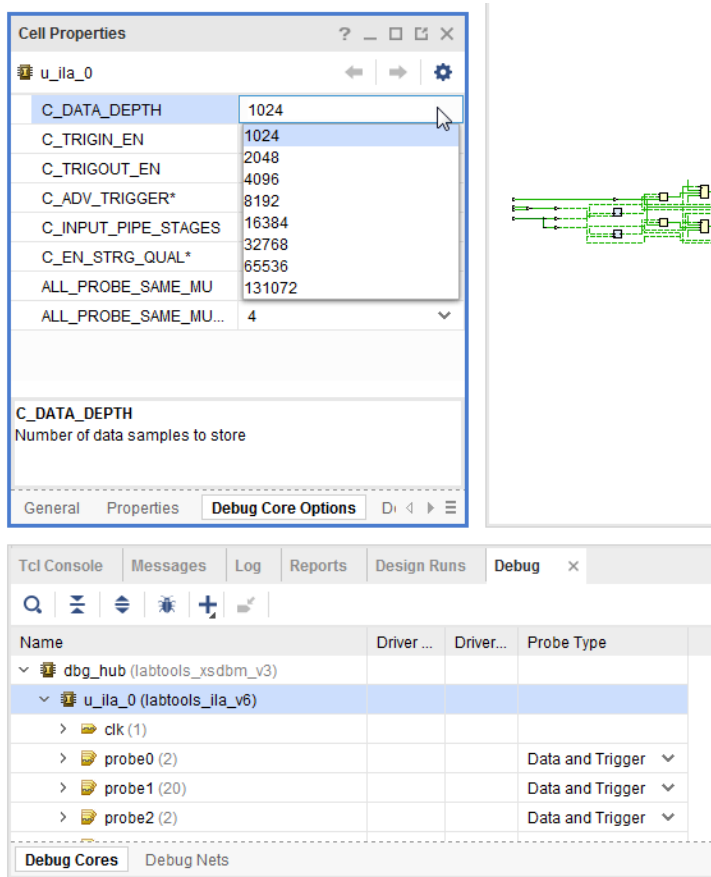
デバッグ コアのポートからネットの接続を解除するには、ポートに接続されているネットを右クリックし、[Disconnect Net] をクリックします。

デバッグ コアのプロパティの変更

各デバッグ コアには、コアの動作をカスタマイズするパラメーターがあります。debug_core_hub デバッグ コアのプロパティを変更する方法については、「デバッグ コア ハブの BSCAN ユーザー スキャン チェーンの変更」を参照してください。

ILA デバッグ コアのプロパティも変更できます。たとえば、ILA デバッグ コアでキャプチャされるサンプルの数を変更するには、次の手順に従います。

1. [Debug] ウィンドウで ILA コア (u_ila_0 など) を選択します。
2. [Cell Properties] ウィンドウで [Debug Core Options] ビューをクリックします。
3. [C_DATA_DEPTH] のドロップダウン リストから、キャプチャするサンプル数を選択します。



次の表に、ILA コアのプロパティの説明を示します。

表 4: ILA デバッグ コアのプロパティ

デバッグ コアのプロパティ	説明	設定可能な値
C_DATA_DEPTH	ILA コアで格納可能なデータ サンプルの最大数。この値を増加すると、ILA で使用されるブロック RAM 数が増加し、デザインパフォーマンスに悪影響を与えることがあります。	1024 (デフォルト) 2048 4096 8192 16384 32768 65536 131072
C_TRIGIN_EN	ILA コアの TRIG_IN および TRIG_IN_ACK ポートをイネーブルにします。これらのポートは、アドバンス ネットリスト変更コマンドを使用して、デザインのネットに接続する必要があります。ILA トリガー入力または出力信号を使用する場合は、HDL インスタンスエーション方法を使用して ILA をデザインに追加することを考慮してください。	false (デフォルト) true
C_TRIGOUT_EN	ILA コアの TRIG_OUT および TRIG_OUT_ACK ポートをイネーブルにします。これらのポートは、アドバンス ネットリスト変更コマンドを使用して、デザインのネットに接続する必要があります。ILA トリガー入力または出力信号を使用する場合は、HDL インスタンスエーション方法を使用して ILA をデザインに追加することを考慮してください。	false (デフォルト) true

表 4: ILA デバッグ コアのプロパティ (続き)

デバッグ コアのプロパティ	説明	設定可能な値
C_ADV_TRIGGER	ILA コアの ADVANCED トリガー モードをイネーブルにします。この機能の詳細は、「ハードウェアでのロジック デザインのデバッグ」を参照してください。	false (デフォルト) true
C_INPUT_PIPE_STAGES	ILA コアの PROBE 入力のパイプ段に追加レベル (例: フリップフロップレジスタ) を使用できるようにします。この機能を使用すると、Vivado ツールで ILA コアをデザインのクリティカルセクションから離して配置できるようになるので、デザインのタイミング パフォーマンスを改善できることがあります。	0 (デフォルト) 1 2 3 4 5 6
C_EN_STRG_QUAL	ILA コアの BASIC キャプチャ モードをイネーブルにします。この機能の詳細は、「ハードウェアでのロジック デザインのデバッグ」を参照してください。	false (デフォルト) true
C_ALL_PROBE_SAME_MU	ILA コアの PROBE 入力をすべてイネーブルにして、同じ数のコンパレータ (比較ユニット) が含まれるようにします。このプロパティは常に true に設定しておく必要があります。	true (デフォルト) false (推奨しません)
C_ALL_PROBE_SAME_MU_CNT	ILA コアの PROBE 入力ごとのコンパレータ (比較ユニット) の数。必要なコンパレータの数は、C_ADV_TRIGGER および C_EN_STRG_QUAL プロパティの設定によって異なります。 C_ADV_TRIGGER が false で C_EN_STRG_QUAL が false の場合は 1 ～ 16 に設定。 C_ADV_TRIGGER が false で C_EN_STRG_QUAL が true の場合は 2 ～ 16 に設定。 C_ADV_TRIGGER が true で C_EN_STRG_QUAL が false の場合は 1 ～ 16 に設定。 C_ADV_TRIGGER が true で C_EN_STRG_QUAL が true の場合は 2 ～ 16 に設定。 重要: 上記の規則に従わない場合、ILA コアを生成したときにインプリメンテーション中にエラー メッセージが表示されます。	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

関連情報

[デバッグ コア ハブの BSCAN ユーザー スキャン チェーンの変更](#)

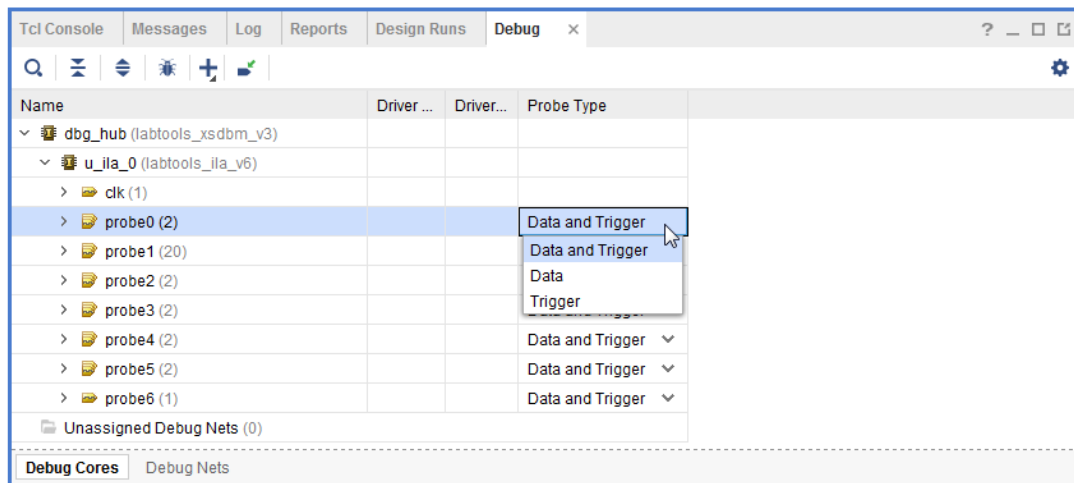
[ハードウェアでのロジック デザインのデバッグ](#)

プローブをデータ、トリガー、またはその両方として使用

Vivado ハードウェア マネージャーでは、プローブをデータ、トリガー、またはその両方として使用するようカスタマイズできます。トリガーまたは比較値をキャプチャするプローブは、トリガーのみのプローブとしてコンフィギュレーションされます。これにより、ILA コアでのブロック RAM の使用が最適化されます。通常、プローブのデータをキャプチャする必要がある場合、そのプローブはデータのみのプローブとしてコンフィギュレーションされます。プローブが比較値のトリガーに使用され、またそのデータをキャプチャする必要がある場合は、プローブをトリガーおよびデータとしてコンフィギュレーションする必要があります。

プローブは、Set up Debug ウィザードでデータ、トリガー、またはその両方としてコンフィギュレーションできます。

図 77: Set up Debug ウィザードでプローブをデータ、トリガー、またはその両方としてコンフィギュレーション

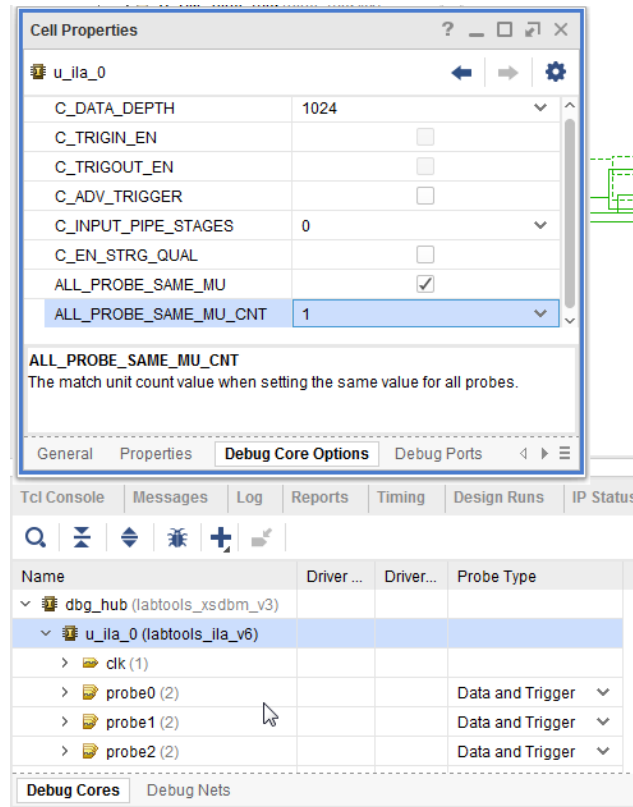


データのみとしてコンフィギュレーションされたプローブを含むデザインでデバイスをランタイムでプログラムする場合、これらのプローブをトリガーまたはキャプチャ セットアップ条件を設定するために使用することはできません。また、トリガーのみとしてコンフィギュレーションされたプローブは、[Waveform] ウィンドウでは使用できません。

使用するコンパレータの数の設定

合成後のネットリストに ILA コアを挿入したら、使用するコンパレータの数を 1 ～ 16 に設定できます。Vivado IDE でこれを設定するには、ILA コアの [Debug Core Options] タブで ALL_PROBE_SAME_MU_CNT プロパティの値を使用するコンパレータの数に設定します。

図 78: ILA コアの [Cell Properties] ウィンドウの [Debug Core Options] ビュー



または、[Tcl Console] ウィンドウで次の Tcl コマンドを使用して
ALL_PROBE_SAME_MU_CNTALL_PROBE_SAME_MU_CNT プロパティを設定します。

```
set_property ALL_PROBE_SAME_MU_CNT 10 [get_debug_cores u_ila_0]
```

ヒント: [Capture Control] がオンの場合は、使用可能なコンパレータの数は 1 ~ 15 です。1 つのコンパレータは、キャプチャ制御データ フィルター メカニズムにより使用されます。

重要: ネットリスト挿入フローを使用する場合、ILA の異なるプローブに対して異なるコンパレータ数を設定することはできません。異なるプローブに異なるコンパレータ数を設定するには、HDL インスタンスエーションフローを使用することをお勧めします。

XDC コマンドを使用したデバッグ コアの挿入

Set up Debug ウィザードの使用に加え、XDC コマンドを使用使用してもデバッグ コアを作成、接続し、合成済みデザイン ネットリストに挿入できます。次の手順に従って、[Tcl Console] ウィンドウに XDC コマンドを入力します。

1. synth_1. という合成 run から合成済みネットリストを開きます。

```
open_run synth_1
```

重要: 次の手順の XDC コマンドは、合成済みネットリストが開いている場合にのみ使用できます。

2. ILA コアのブラック ボックスを作成します。

```
create_debug_core u_ila_0 ila
```

3. ILA コアのプロパティを設定します。

```
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
```

4. ILA コアの clk ポートの幅を 1 に設定し、適切なクロック ネットに接続します。

```
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk ]]
```

注記: ILA コアの clk ポートは、create_debug_core コマンドで自動的に作成されるので、ユーザーが作成する必要はありません。



重要: デバッグ コアのデバッグ ポート名はすべて小文字です。大文字や小文字と大文字の混ざったポート名を使用すると、エラーになります。

5. probe0 ポートの幅を、そのポートに接続するネットの数に設定します。

注記: ILA コアの最初のプローブ ポート (probe0) は、create_debug_core コマンドで自動的に作成されるので、ユーザーが作成する必要はありません。

6. probe0 ポートを、そのポートに接続するネットに接続します。

```
connect_debug_port u_ila_0/probe0 [get_nets [list A_or-B]]
```

7. (オプション) 必要なだけプローブ ポートを作成し、幅を設定して、デバッグするネットに接続します。

```
create_debug_port u_ila_0 probe
set_property port_width 2 [get_debug_ports u_ila_0/probe1]
connect_debug_port u_ila_0/probe1 [get_nets [list {A[0]} {A[1]}]]
```

これらのコマンドおよびその他の関連コマンドの詳細は、[Tcl Console] ウィンドウに「help -category ChipScope」と入力してください。

デバッグ XDC コマンド実行後の制約の保存

Set up Debug ウィザードを使用した後、Vivado Design Suite を使用してデバッグ コアまたはポートを作成した後、および次の XDC コマンドを実行した後は、制約を保存する必要があります。

- create_debug_core
- create_debug_port
- connect_debug_port
- set_property (debug_core または debug_port オブジェクトに適用)

対応する XDC コマンドが _debug.xdc の付いた制約ファイルに保存され、インプリメンテーション中にデバッグ コアの挿入および接続に使用されます。



重要: プロジェクト モードで制約を保存すると、合成およびインプリメンテーションが更新必要な状態になることがあります。デバッグ XDC 制約はインプリメンテーションでしか使用されないため、合成を再実行する必要はありません。合成を強制的に最新の状態にするには、[Design Runs] ウィンドウで合成 run (例: synth_1) を右クリックし、[Force up-to-date] をクリックします。

デザインのインプリメンテーション

デバッグ コアを挿入、接続、カスタマイズしたら、デザインをインプリメントします。詳細は、「デバッグ コアを含むデザインのインプリメンテーション」を参照してください。

関連情報

[デバッグ コアを含むデザインのインプリメンテーション](#)

非プロジェクト モードでのデバッグ コアの挿入

デバッグ コアは、プロジェクト モードおよび非プロジェクト モード両方で挿入できます。次のサンプル Tcl スクリプトは、コアをデバッグし、デバッグ コアの属性を設定し、そのデバッグ コア プローブをプローブされるデザインの信号に接続します。非プロジェクト モードの場合、デバッグ コアはデザイン合成後、opt_design 段階の前に挿入する必要があります。



重要: デバッグ コアの挿入は、ILA コアに対してのみサポートされます。

次の Tcl スクリプトは、非プロジェクト フローでデバッグ コア挿入コマンドを使用した例です。

```
#read relevant design source files
read_vhdl [glob *.*.vhd]
read_verilog [ glob ./Sources/*.v ]
read_xdc ./target.xdc
#Synthesize Design
synth_design -top top -part xc7k325tffg900-2
#Create the debug core
create_debug_core u_ila_0 ila
#set debug core properties
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
#connect the probe ports in the debug core to the signals being probed in
the design
set_property port_width 1 [get_debug_ports u_ila_0/clock]
connect_debug_port u_ila_0/clock [get_nets [list clock]]
set_property port_width 1 [get_debug_ports u_ila_0/probe0]
connect_debug_port u_ila_0/probe0 [get_nets [list A_or_B]]
create_debug_port u_ila_0 probe
#Optionally, create more probe ports, set their width,
# and connect them to the nets you want to debug
#Implement design
opt_design
place_design
report_drc -file ./placed_drc_rpt.txt
```

```
report_timing_summary -file ./placed_timing_rpt.txt
route_design
report_drc -file ./routed_drc_rpt.txt
report_timing_summary -file ./routed_timing_rpt.txt
write_bitstream
```

HDL インスタンスレーション デバッグ プローブ フローの概要

HDL インスタンスレーション プローブ フローでは、HDL デザイン ソースで直接デバッグ コア コンポーネントをカスタマイズ、インスタンスiert、および接続します。次の図に、このフローでサポートされる新しいデバッグ コアを示します。

表 5: HDL インスタンスレーション プローブ フローで使用可能な Vivado IP カタログに含まれるデバッグ コア

デバッグ コア	バージョン	説明	ランタイム解析ツール
ILA (Integrated Logic Analyzer)	v6.2	ハードウェア イベントをトリガーし、データをシステム速度でキャプチャするために使用するデバッグ コア。	Vivado ロジック解析
VIO (Virtual Input/Output)	v3.0	デザインの信号を JTAG チェーン スキャン レートで監視または制御するために使用するデバッグ コア。	Vivado ロジック解析
JTAG-to-AXI Master	v1.2	ハードウェアで動作中のさまざまな AXI フルおよび AXI Lite スレーブ コアと通信する AXI トランザクションを生成するために使用されるデバッグ コア。	Vivado ロジック解析

新しい ILA コアをレガシ ILA v1.x コアと比較した場合の利点

- Vivado ロジック解析で機能します。詳細は、「ハードウェアでのロジック デザインのデバッグ」を参照してください。
- ICON コアの挿入および接続は必要ありません。

関連情報

[ハードウェアでのロジック デザインのデバッグ](#)

HDL インスタンスレーション デバッグ プローブ フロー

HDL インスタンスレーション フローの手順は、次のとおりです。

1. プローブする信号用に、適切な数のプローブ ポートを含む ILA および VIO デバッグ コアをカスタマイズして生成します。
2. (オプション) JTAG-to-AXI Master デバッグ コアをカスタマイズおよび生成して、デザインの AXI ペリフェラルまたはインターコネクト コアの AXI スレーブ インターフェイスに接続します。
3. デバッグ コアを含むデザインを合成します。

4. (オプション) Debug Hub コアのプロパティを変更します。
5. デバッグ コアを含むデザインをインプリメントします。

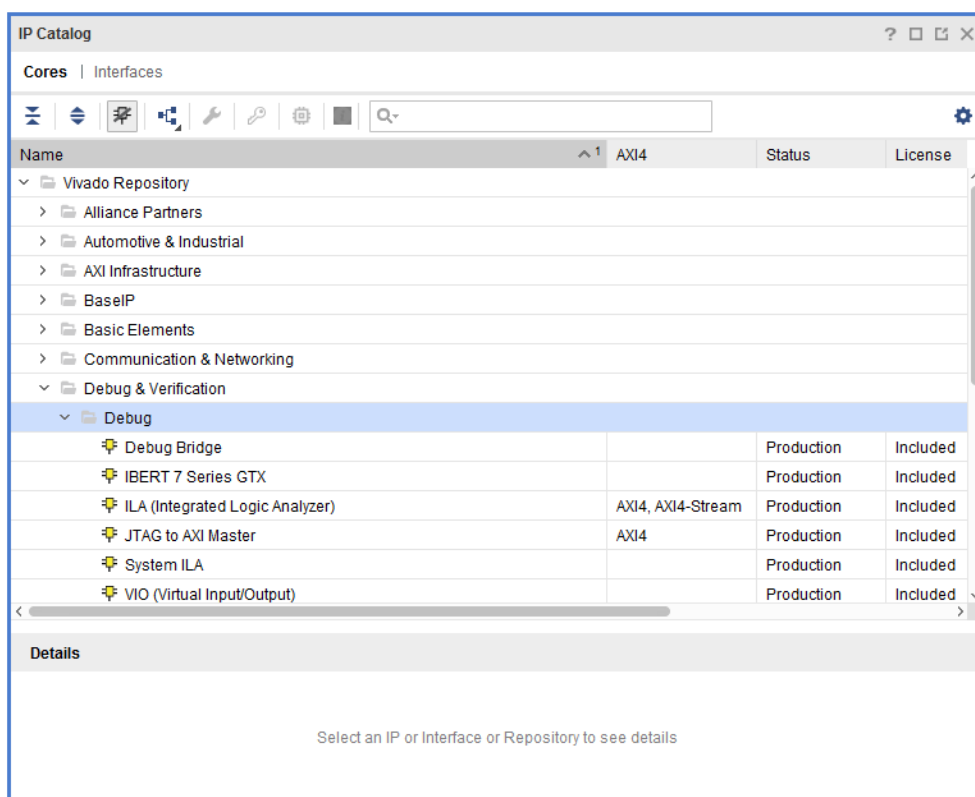
デバッグ コアのカスタマイズおよび生成

Flow Navigator で [Project Manager] → [IP Catalog] をクリックし、必要なデバッグ コアを選択してカスタマイズします。デバッグ コアは、IP カタログの [Debug & Verification] → [Debug] カテゴリにあります。デバッグ コアをカスタマイズするには、IP コアをダブルクリックするか、右クリックして [Customize IP] をクリックします。

- ILA コアのカスタマイズに関する詳細は、『Integrated Logic Analyzer LogiCORE IP 製品ガイド』 (PG172) を参照してください。
- VIO コアのカスタマイズに関する詳細は、『Virtual Input/Output LogiCORE IP 製品ガイド』 (PG159) を参照してください。
- JTAG-to-AXI Master コアのカスタマイズに関する詳細は、『JTAG to AXI Master LogiCORE IP 製品ガイド』 (PG174) を参照してください。

コアをカスタマイズしたら、[Generate] ボタンをクリックします。カスタマイズされたデバッグ コアが生成され、[Sources] ウィンドウに追加されます。

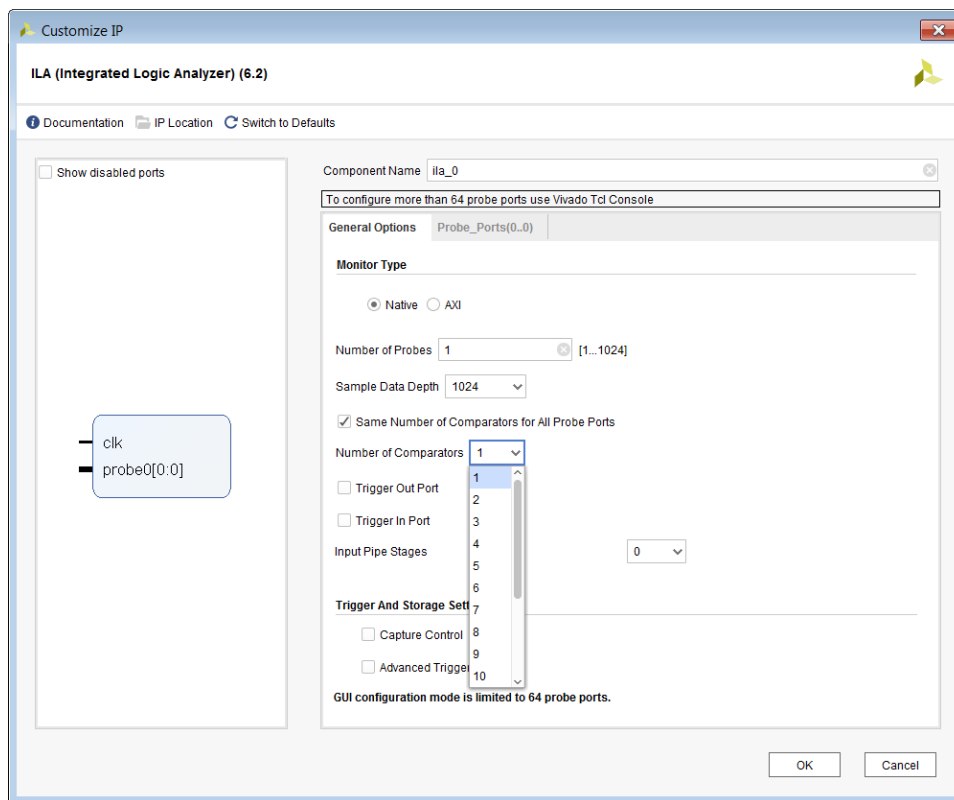
図 79: IP カタログのデバッグ コア



使用するコンパレータの数の設定

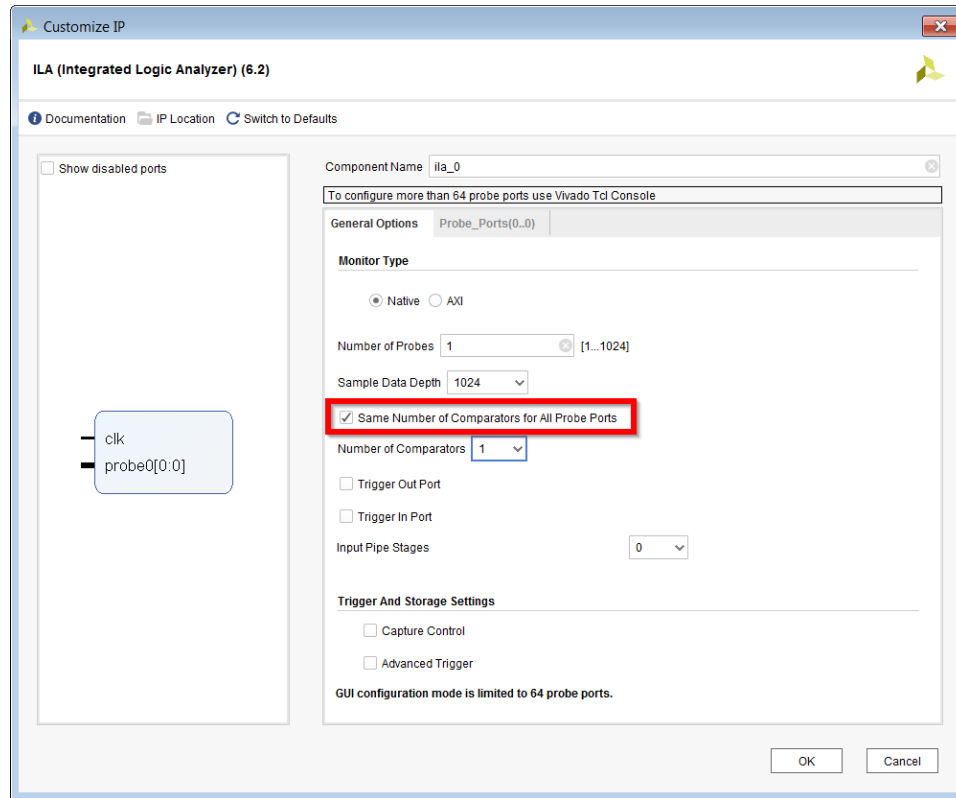
ILA IP をカスタマイズする際に、使用するコンパレータの数を設定できます。有効な値は 1 ～ 16 です。ILA IP のすべてのプローブに対する共通のコンパレータ数を設定することが可能です。

図 80: ILA IP のコンパレータ数の設定



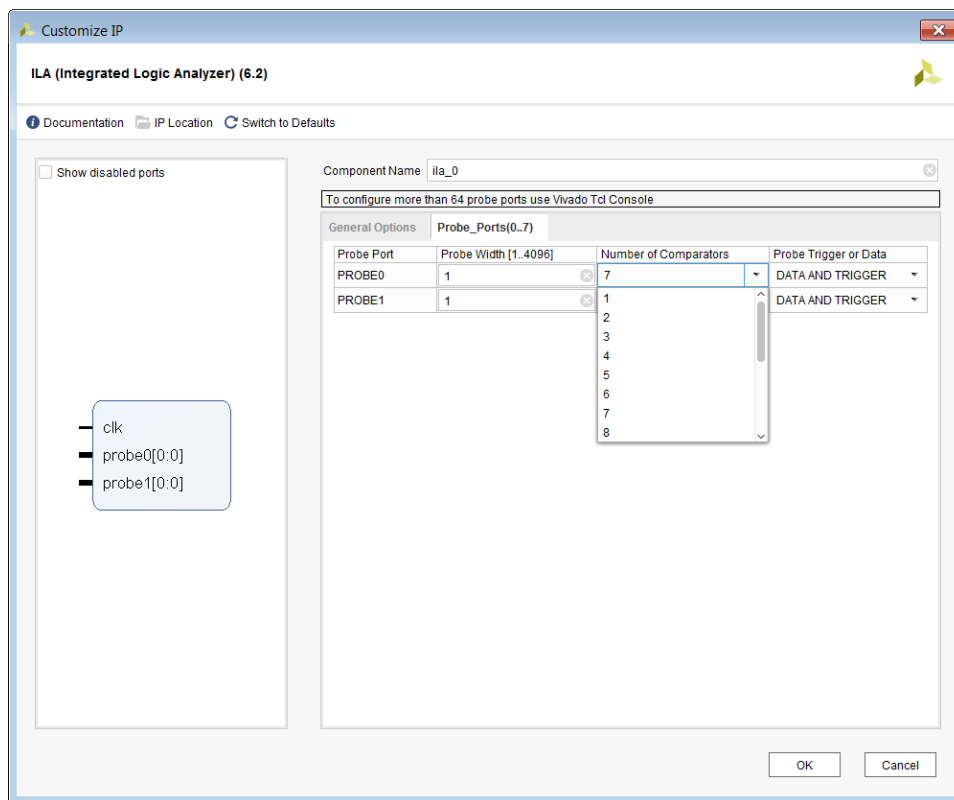
次に示すように各 IP に対してコンパレータを設定できます。1 つの ILA 内に異なる幅の複数のプローブを含めることができます。これには、[General Options] タブで [Same Number of Comparators for All Probe Ports] をオフにします。

図 81: [Same Number of Comparators for All Ports] オプション



その後、[Probe_Ports] タブで各プローブの [Number of Comparators] フィールドに使用するコンパレータ数を入力します。

図 82: 各プローブのコンパレータ数の設定



ヒント: [Capture Control] がオンの場合は、使用可能なコンパレータの数は 1 ～ 15 です。1 つのコンパレータは、キャプチャ制御データ フィルター メカニズムにより使用されます。

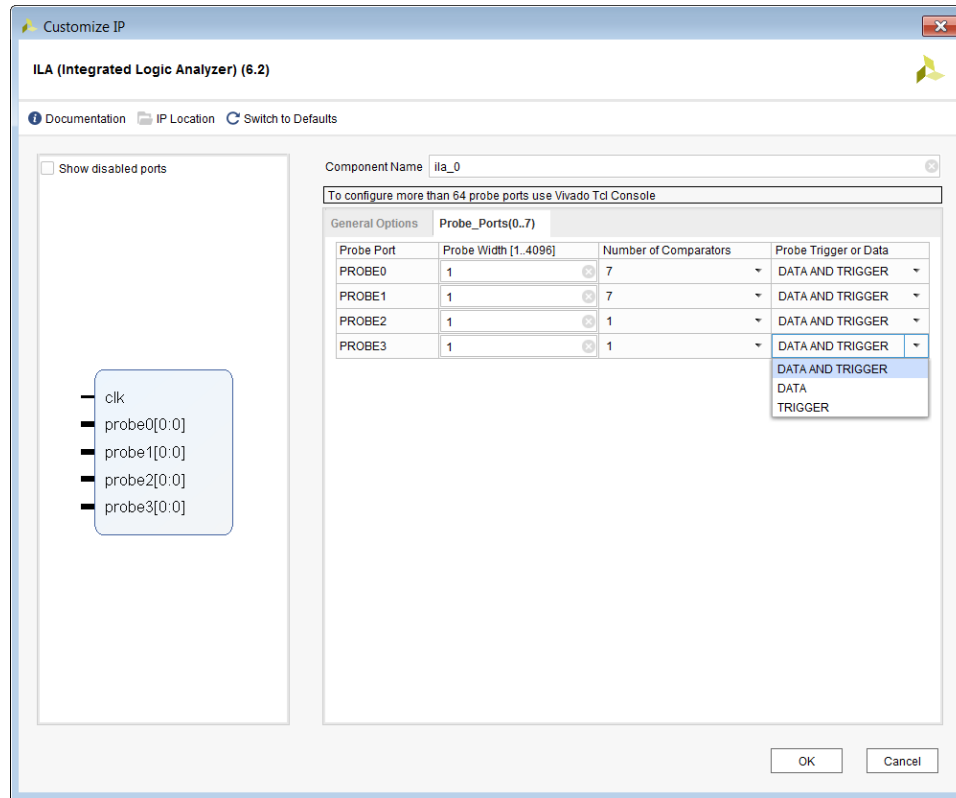


ヒント: 選択したコンパレータ数によって、ILA IP で使用可能なプローブ数が自動的に再算出されます。ILA ごとに使用可能なコンパレータの最大数は 1024 です。

プローブをデータまたはトリガーとして使用

プローブは、ILA IP の [Customize IP] ダイアログ ボックスでデータ、トリガー、またはその両方としてコンフィギュレーションできます。

図 83: プローブをトリガーおよびデータの両方としてコンフィギュレーション



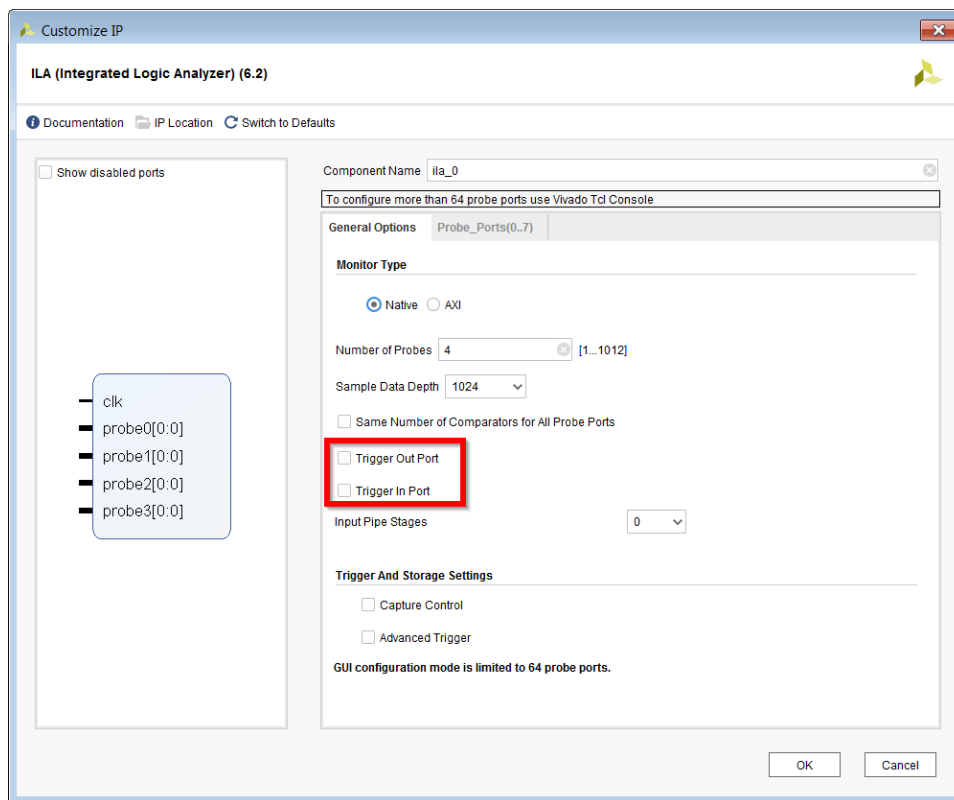
トリガーまたは比較値をキャプチャするプローブは、トリガーのみのプローブとしてコンフィギュレーションされます。これにより、ILA コアでのブロック RAM の使用が最適化されます。通常、プローブのデータをキャプチャする必要がある場合、そのプローブはデータのみのプローブとしてコンフィギュレーションされます。プローブが比較値のトリガーに使用され、またそのデータをキャプチャする必要がある場合は、プローブをトリガーおよびデータとしてコンフィギュレーションする必要があります。

ILA クロス トリガー

ILA クロス トリガー機能は、ILA コア間および ILA コアとプロセッサ (Zynq®-7000 SoC など) 間のクロス トリガーを可能にします。この機能は、異なるクロック ドメインにある 2 つの ILA コアをトリガーする場合や、プロセッサと ILA コアの間でハードウェア/ソフトウェア クロス トリガーを実行する場合に有益です。

クロス トリガー機能を使用するには、コア生成時に、ILA コアに専用トリガー入力ポート (TRIG_IN および TRIG_IN_ACK) と専用トリガー出力ポート (TRIG_OUT および TRIG_OUT_ACK) を設定する必要があります。ILA トリガー入力または出力信号を使用する場合は、HDL インスタンス化方法を使用して ILA をデザインに追加する必要があります。

図 84: ILA クロストリガー機能



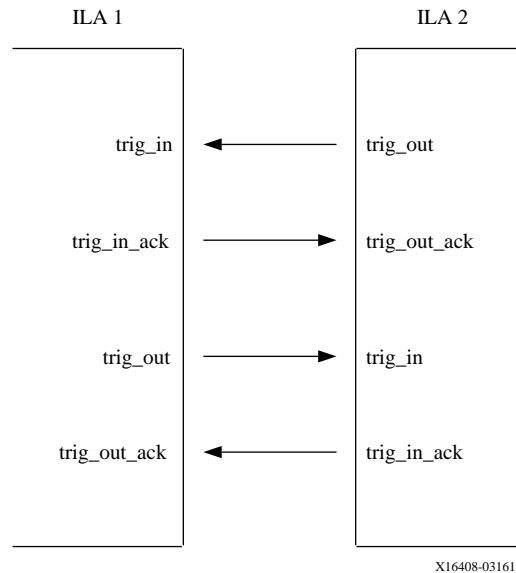
TRIG_OUT_ACK 信号は、TRIG_OUT が正しく受信されたことを ILA コア (別の ILA、ユーザー デザイン、またはプロセッサ) に通知し、個の信号が受信されると ILA の TRIG_OUT 信号が Low になります。

TRIG_OUT は、TRIG_OUT_ACK が受信されるまで High に保持されます。TRIG_OUT_ACK 信号を Low に接続した場合は、TRIG_OUT はユーザーが ILA をトリガー待機状態にするまで High に保持され、ILA がトリガー待機状態になると Low になります。TRIG_OUT_ACK が Low に接続されている場合、ユーザーが ILA をトリガー待機状態にできません。

次の図に、ILA2 が ILA1 にクロストリガーされる一般的なクロストリガー セットアップを示します。ILA2 の TRIG_OUT 信号は、ILA1 の TRIG_IN 信号に接続されます。ILA1 の TRIG_IN_ACK 信号は、ILA2 の TRIG_OUT_ACK 信号に接続されます。

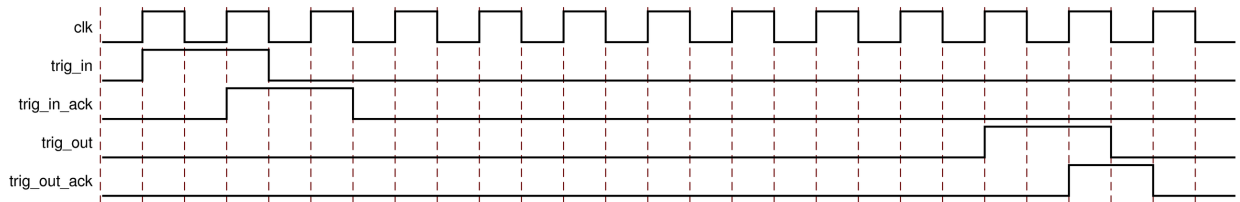
```
(ILA 2) trig_out -> (ILA 1) trig_in
(ILA 1) trig_in_ack -> (ILA 2) trig_out_ack
```

図 85: 一般的なクロス トリガー セットアップ



X16408-031616

図 86: ILA クロス トリガーのタイミング



- `trig_in` ポートを駆動するロジックは、ILA の `clk` に同期すると想定しています。
- `trig_in_ack` がアサートされてから `trig_in` 信号がアサートされるまでに 1 clk サイクルかかります。
- `trig_out` がアサートされてトリガー条件が満たされから `trig_in` 信号がアサートされるまでに 9 clk サイクルかかります。
- `trig_in_ack` および `trig_out_ack` 信号は、トリガー信号がディアサートされると Low になります。

FPGA ファブリックと Zynq-7000 SoC プロセッサの間でクロス トリガー機能を使用した詳細なチュートリアルは、『Vivado Design Suite チュートリアル: エンベデッド プロセッサ ハードウェア デザイン』(UG940)を参照してください。

デバッグ コアのインスタンス化

デバッグ コアを生成したら、HDL ソースにインスタンス化し、デバッグ用にプローブする信号に接続します。次に、Verilog HDL ソース ファイルの ILA インスタンスの例を示します。

```
u_ila_0
(
  .clk(clk),
  .probe0(counterA),
  .probe1(counterB),
  .probe2(counterC),
```

```
.probe3(counterD),
.probe4(A_or_B),
.probe5(B_or_C),
.probe6(C_or_D),
.probe7(D_or_A)
);
```

注記: 新しい ILA コア インスタンスでは、レガシ VIO および ILA v1.x コアとは異なり、ICON コア インスタンスへの接続は必要ありません。その代わりにデバッグ コア ハブ デバッグ (dbg_hub) が合成済みデザイン ネットリストに自動的に挿入され、新しい ILA コアと JTAG スキャン チェーンが接続されます。

デバッグ コアを含むデザインの合成

次に、Vivado Design Suite で [Run Synthesis] をクリックするか、次の Tcl コマンドを使用して、デバッグ コアを含むデザインを合成します。

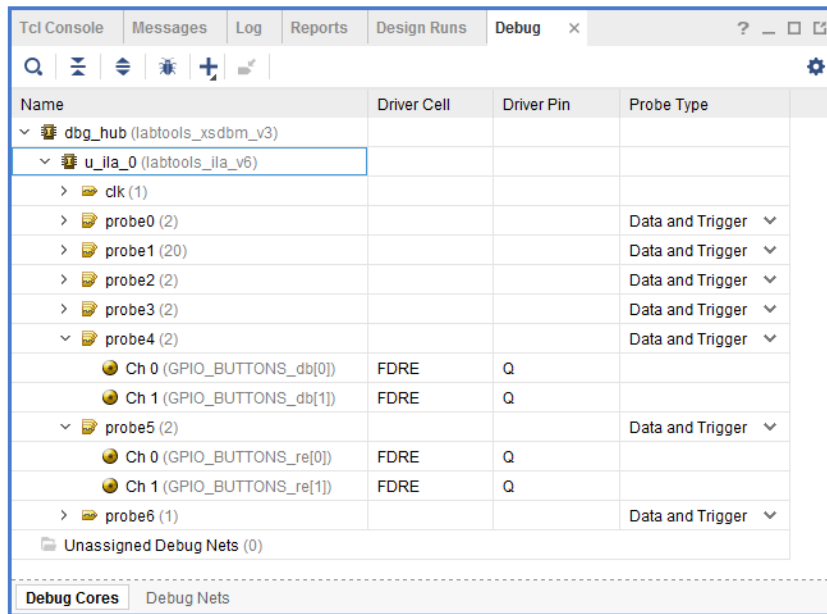
```
launch_runs synth_1
wait_on_run synth_1
```

synth_design Tcl コマンドを使用してデザインを合成することもできます。さまざまな合成方法の詳細は、『Vivado Design Suite ユーザー ガイド: 合成』 ([UG901](#)) を参照してください。

合成済みデザインでのデバッグ コアの表示

デザインを合成したら、合成済みデザインを開いてデバッグ コアを表示し、プロパティを変更できます。[Flow Navigator] で [Open Synthesized Design] をクリックして合成済みデザインを開き、[Debug] レイアウトを選択して、[Debug] ウィンドウで Debug Hub コア (dbg_hub) に接続された ILA デバッグ コアを確認します。

図 87: ILA コアとデバッグ ハブ コアを示す [Debug] ウィンドウ



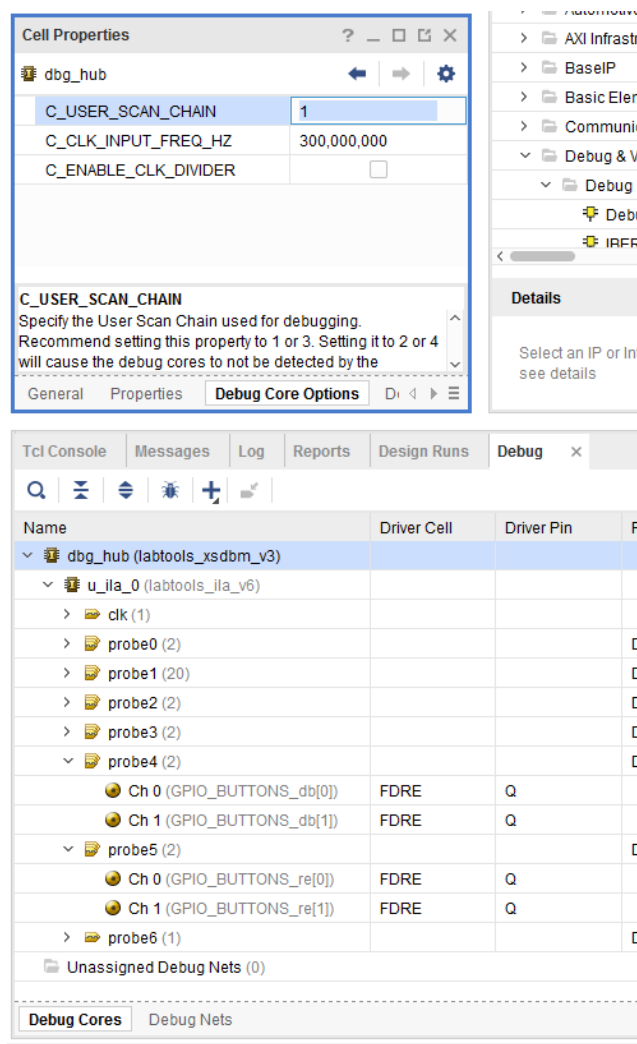
デバッグ コア ハブの BSCAN ユーザー スキャン チェーンの変更

デバッグ ハブ コアの BSCAN ユーザー スキャン チェーン インデックスを表示および変更するには [Debug] ウィンドウで [dbg_hub] を選択し [Properties] ウィンドウで [Debug Core Options] ビューをクリックして、[C_USER_SCAN_CHAIN] の値を変更します (次の図を参照)。

★ **重要:** デバッグ ハブ コアの C_USER_SCAN_CHAIN のデフォルト値は、1 です。デバッグ ハブ コアに 1 以外のスキャン チェーン値を使用する場合は、ハードウェア マネージャーのデバイスで手動で変更する必要があります。詳細は、「ハードウェア デバイスのプログラム」を参照してください。

★ **重要:** マイクロプロセッサ デバッグ モジュール (MDM) または BSCAN プリミティブを Vivado ロジック デバッグ コアと共に使用するその他の IP を使用する場合は、dbg_hub の C_USER_SCAN_CHAIN プロパティをほかの IP のバウンダリスキャン チェーン設定と競合しない値に設定しておかないと、インプリメンテーションフローでエラーが発生します。

図 88: デバッグ ハブ コアのユーザー スキャン チェーン プロパティの変更



関連情報

[ハードウェア デバイスのプログラム](#)

IP インテグレーターでのデバッグ フロー

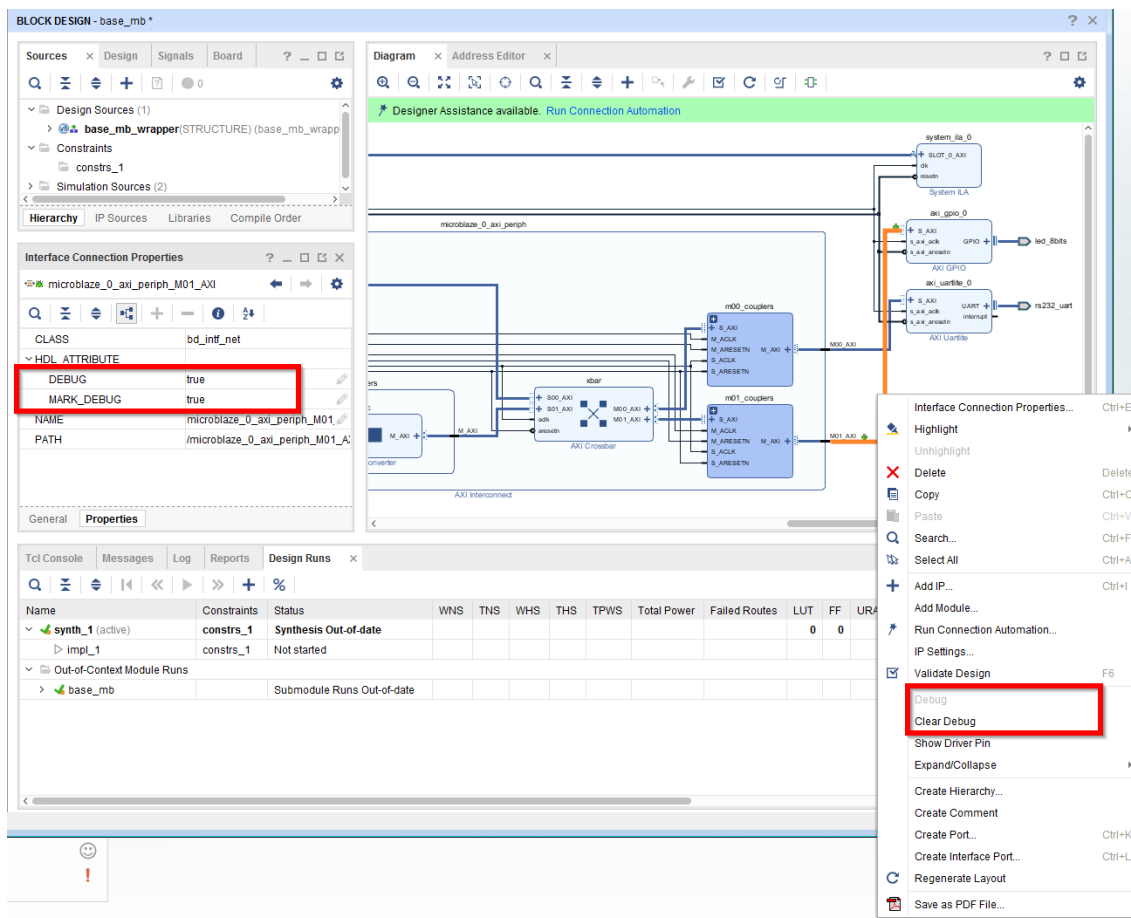
Vivado IP インテグレーターの System ILA IP を使用すると、FPGA デバイス上でインプリメント後のデザインのインシステム デバッグを実行できます。IP インテグレーター ブロック デザインでインターフェイスおよび信号を監視する必要がある場合は、この機能を使用します。この機能を使用すると、Vivado ハードウェア マネージャーで AXI 読み出しおよび書き込みトランザクションだけでなく、AXI 読み出しおよび書き込み、データ、アドレス チャネル イベントをデバッグできます。

ブロック デザインでインターフェイスおよびネットをデバッグする手順は、『Vivado Design Suite ユーザー ガイド: IP インテグレーターを使用した IP サブシステムの設計』(UG994) の[このセクション](#)を参照してください。

IP インテグレーター ブロック デザインでのネットおよびインターフェイスのデバッグ

IP インテグレーター ブロック デザイン キャンバスでは、ネットとインターフェイスの両方をデバッグできます。次に示すように、ブロック デザインでインターフェイスまたはネットを右クリックし、[Debug] をクリックします。これで DEBUG および MARK_DEBUG 属性が true に設定されます。また、設計アシスタンスも使用できるようになり、コネクション オートメーションが実行できるので、System Interface ILA コアへのネットまたはインターフェイスを選択したり、さまざまなデバッグ コアの属性をカスタマイズしたりできます。

図 89: IP インテグレーターでデバッグ用マークをした状態

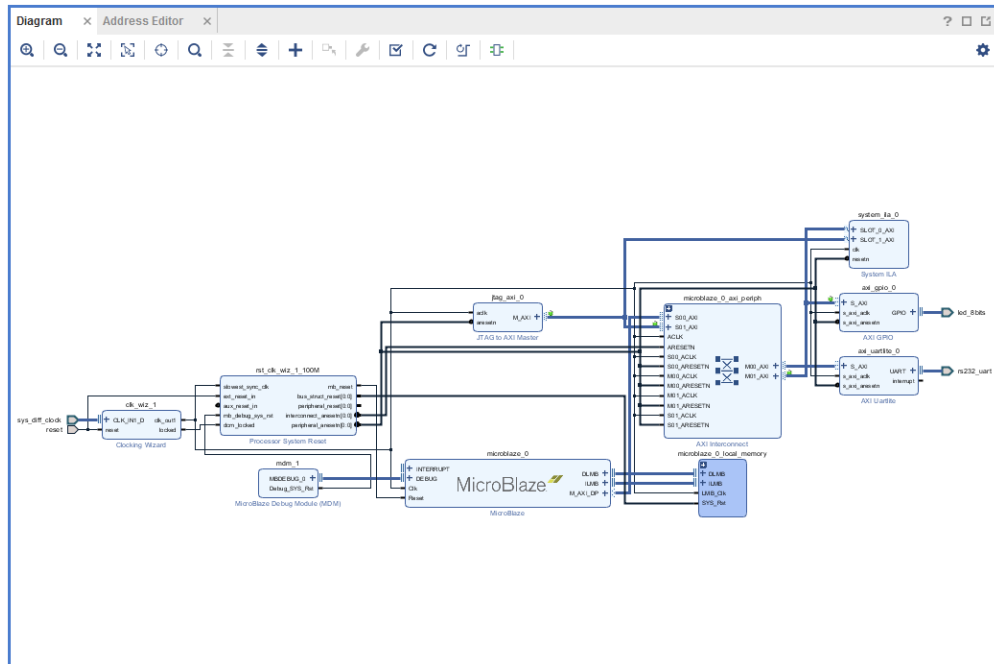


ネットまたはインターフェイスからデバッグ属性を削除するには、そのネット/インターフェイスを右クリックして [Clear Debug] をクリックします。

合成済みデザインでの System ILA デバッグ コアの表示

IP インテグレーター ブロック デザインの System ILA IP はインスタンス化する必要があります。次の図はブロック デザインのスナップショットで、デザイン、System ILA、および JTAG to AXI Master IP コアで 2 つのデバッグ コアがインスタンス化されています。

図 90: ブロック デザイン



このブロック デザインを検証および合成した後、合成済みのデザインで [Debug] ウィンドウを開き、デザインにインスタンス化および挿入されたデバッグ コアを表示させることができます。System ILA および JTAG to AXI Master デバッグ コアは次のように表示されます。

図 91: System ILA および JTAG to AXI Master デバッグ コア

Tcl Console Messages Log Reports Design Runs Debug				
Name	Driver ...	Driver ...	Probe Type	
dbg_hub (labtools_xsdbm_v3)				
jtag_axi_0 (labtools_xsdbslavelib_v2)				
system_ila_0				
microblaze_0_axi_periph_M01_AXI (SLOT_0_AXI)				
base_mb_i/microblaze_0_axi_periph_M01_AXI_BRESP (2)	GND	G	Data and Trigger	
base_mb_i/microblaze_0_axi_periph_M01_AXI_RDATA (32)	Multiple	Multiple	Data and Trigger	
base_mb_i/microblaze_0_axi_periph_M01_AXI_RRESP (2)	GND	G	Data and Trigger	
base_mb_i/microblaze_0_axi_periph_M01_AXI_WDATA (32)	LIT3	G	Data and Trigger	
Debug Cores	Debug Nets			

ハードウェア マネージャーでこれらのインターフェイスをデバッグ 用にどう使用するか、また AXI イベント レベルのデバッグを利用するにはどうしたらよいのかに関しては、「ハードウェア マネージャーでの AXI インターフェイスのデバッグ」を参照してください。

関連情報

ハードウェア マネージャーでの AXI インターフェイスのデバッグ

デバッグ コアを含むデザインのインプリメンテーション

Vivado ツールでは、デバッグ ハブ コアは最初ブラック ボックスとして作成されます。このコアは、配置配線を実行する前にインプリメントしておく必要があります。

デザインのインプリメンテーション

デバッグ コアを含むデザインをインプリメントするには、Vivado Design Suite で [Run Implementation] をクリックするか、次の Tcl コマンドを使用します。

```
launch_runs impl_1
wait_on_run impl_1
```

インプリメンテーション コマンド `opt_design`、`place_design` および `route_design` を使用して、デザインをインプリメントすることも可能です。さまざまなインプリメント方法の詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 ([UG904](#)) を参照してください。

ILA コアとタイミングに関する考慮事項

ILA コアをコンフィギュレーションすると、デザイン全体のタイミング目標の達成に影響します。タイミングへの影響を最小限に抑えるためには、次をお勧めします。

- プローブ幅を注意して選択します。プローブ幅が大きいほど、リソース使用量とタイミングの両方への影響も大きくなります。
- ILA コアのデータの深さを注意して選択します。データの深さが大きいほど、ブロック RAM リソース使用量とタイミングの両方への影響も大きくなります。
- ILA に選択するクロックはフリーランニング クロックにします。そうでないと、デザインがデバイスに読み込まれたときに、デバッグ コアと通信できなくなる可能性があります。
- `dbg_hub` へのクロックはフリーランニング クロックにします。そうでないと、デザインがデバイスに読み込まれたときに、デバッグ コアと通信できなくなる可能性があります。Tcl コマンドの `connect_debug_port` を使用すると、デバッグ ハブの `clk` ピンをフリーランニング クロックに接続できます。
- デバッグ コアを追加する前にデザインのタイミング クロージャを達成しておきます。デバッグ コアは、タイミング関連の問題をデバッグするためには使用しないでください。
- ILA デバッグ コアを追加したためにタイミングが悪化し、クリティカル パスが `dbg_hub` にある場合は、次を実行してください。
 1. 合成済みデザインを開きます。
 2. ネットリストで `dbg_hub` セルを見つけます。
 3. `dbg_hub` のプロパティを確認します。
 4. `C_CLK_INPUT_FREQ_HZ` プロパティを見つけます。
 5. `dbg_hub` に接続されるクロックの周波数 (Hz) をそれに設定します。
 6. `C_ENABLE_CLK_DIVIDER` プロパティを見つけて、オンにします。
 7. デザインをインプリメントし直します。

- ILA コアへのクロック入力がプローブされた信号と同期するようにします。こうしておかないと、デザインがデバイスにプログラムされたときに、タイミング問題が発生したり、デバッグ コアと通信できなくなる可能性があります。
- ハードウェアで実行する前にデザインがタイミングを満たすようにしておかないと、結果の信頼性が低くなります。

デバッグ コアのクロッキング ガイドライン

Vivado ハードウェア マネージャーでは、JTAG インターフェイスを使用して Vivado デバッグ ハブ コアと通信することで、FPGA デバイスの JTAG バウンダリスキャン (BSCAN) インターフェイスと Vivado デバッグ コア間のインターフェイスが提供されています。

- JTAG クロック: このクロックは JTAG バウンダリスキャン (BSCAN) インターフェイスの内部ステート マシン演算を同期します。通常は、ターゲット デバイスに接続した状態で Vivado ハードウェア マネージャーで JTAG クロック周波数を選択します。デザインにデバッグ コアが含まれる場合は、JTAG クロックがデバッグ ハブ クロックよりも 2.5 倍遅くなるようにします。

JTAG クロックの周波数を変更するには、Open New Hardware Target ウィザードまたは次の Tcl コマンドを使用します。

```
set_property PARAM.FREQUENCY 250000 [get_hw_targets  
*/xilinx-tcf/Digilent/210203327962A]
```

- [Debug Hub Clock]:

Vivado Debug Hub コアは、FPGA デバイスの JTAG バウンダリスキャン (BSCAN) インターフェイスと Vivado デバッグ コアとの間のインターフェイスを提供します。Vivado IDE では、デザイン内にデバッグ コアが検出されると、インプリメンテーション段階でデバッグ ハブ コアが自動的に挿入されます。デバッグ ハブ コアを駆動するクロックは、Vivado IDE のデザイン インプリメンテーション段階で自動的に選択されます。

JTAG クロックの速度は特に高周波数にする必要はないので、デバッグ ハブ コアのクロック周波数は約 100 MHz 以下にすることをお勧めします。

デバッグ ハブ クロックは、次の Tcl コマンドで変更できます。

```
connect_debug_port dbg_hub/clk [get_nets <clock net name>]
```

注記: このコマンドは、デザインの合成後、インプリメンテーション前に実行する必要があります。

また、デバッグ ハブ クロックの周波数は、次の Tcl コマンドで低くできます。

```
set_property C_CLK_INPUT_FREQ_HZ 200000000 [get_debug_cores dbg_hub]  
set_property C_ENABLE_CLK_DIVIDER true [get_debug_cores dbg_hub]
```

注記: このコマンドは、デザインの合成後、インプリメンテーション前に実行する必要があります。これは、かなり高速なクロックのデザインに推奨されます。このコマンドを実行すると、デバッグ ハブ コア内に MMCM ベースのクロック分周器が含まれるようになり、クロック周波数を 100 MHz にできます。

デバッグ コアのクロック

Vivado IP カタログから使用可能なデバッグ コアすべてに、監視される入力プロンプ、またはデバッグ コアにより駆動される出力信号と同期されるクロックが必要です。コアの検出およびデバッグ計測段階では、クロックがフリーランニングで安定しており、監視または駆動される信号に同期している必要があります。これらの条件が満たされない、サイクルが不正確なデータになる可能性があります。

デバッグ ハブ IP は、ホスト マシン (シリアル インターフェイスをサポートする BSCAN プリミティブを介す) とチップ上のデバッグ コア (パラレル インターフェイスをサポートする XSDB を介す) 間のブリッジとして使用します。BSCAN プリミティブのクロックは、直列でデバッグ ハブ IP に対するチップからのデータの入力と出力を切り替えます。デバッグ ハブ IP は、データを収集してから、デバッグ ハブ クロックを使用してパラレル インターフェイスのデバッグ コアすべてに送信します (またはデバッグ コアで受信します)。デバッグ コア クロックの中にフリーランニングでなかったり、安定していないものがある場合は、データが破損し、「Debug Cores not detected」というメッセージが表示されます。データが破損しないようにするには、デバッグ コアの検出プロセス中に JTAG クロックとデバッグ ハブ クロックがフリーランニングで安定するようにすることが重要です。

1. デバッグ ハブ クロックは、フリーランニングで安定している必要があります。ザイリンクスでは、制約が正しく付けられ、タイミングを満たしているクロック分周器からクロックが駆動されるようにすることをお勧めしています。
2. クロックが MMCM/PLL から駆動される場合は、MMCM/PLL の LOCKED 信号がデバッグ コアの測定前に High になるようにします。クロックがデバッグ コアに接続されていて、MMCM/PLL の LOCKED 信号がデバッグの最中に 0 に遷移する場合は、クロックにかなりのジッターが含まれ、デバッグ ロジックが予測不可能な状態になることがあります。
3. デバッグ コアを検出するには、これらのコアを使用して測定してからデータを取り込みます。関連するクロックはすべてフリーランニングで安定している必要があります。

次の表は、さまざまなデバッグ段階とその特定段階に必要なクロックを示しています。

表 6: デバッグ段階別クロック要件

デバッグ段階	JTAG クロック	デバッグ ハブ クロック	Debug Core Clock ²
ターゲットへの接続	安定 ¹	なし	なし
プログラミング	安定 ¹	なし	なし
デバッグ コアの検出	安定 ¹	安定	なし
デバッグ コアの測定 ³	安定 ¹	安定 ¹	安定

注記:

1. 安定クロック: イベント中に一時停止/停止しないクロック。
2. デバッグ コア クロックがデバッグ ハブ クロックとは異なると想定した場合。
3. デバッグ コアの測定段階には、デバッグ コアのプロパティの `get` または `set` を実行するすべての段階が含まれます。

Vivado ハードウェア マネージャーのクロッキング関連のエラー メッセージ

JTAG クロックがアクティブでなかったり、使用できない状態の場合は、ハードウェア ターゲットに接続できません。

デバッグ ハブ クロックがアクティブでなかったり、使用できない状態の場合は、Vivado ハードウェア マネージャーで次のエラー メッセージが表示されます。

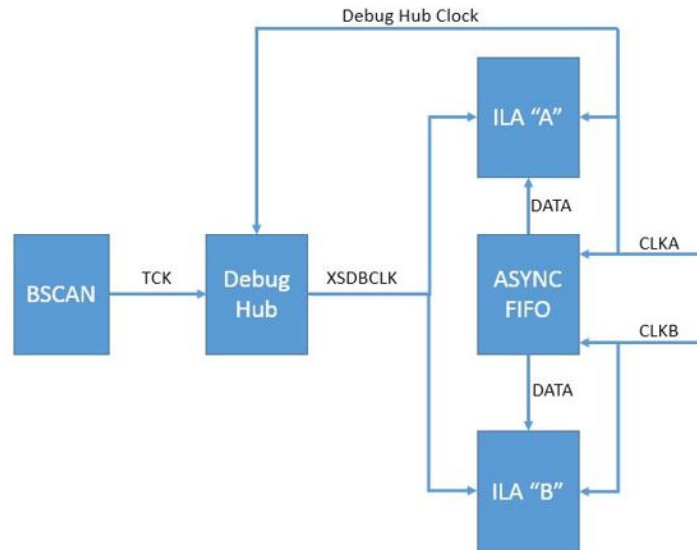
```
INFO: [Labtools 27-1434] Device xxx (JTAG device index = 0) is programmed
with a
design that has no supported debug core(s) in it.
WARNING: [Labtools 27-3123] The debug hub core was not detected at User
Scan Chain 1
or 3.
Resolution:
1. Make sure the clock connected to the debug hub (dbg_hub) core is a free
running
clock and is active OR
2. Manually launch hw_server with -e "set xsdb-user-bscan
<C_USER_SCAN_CHAIN
scan_chain_number>" to detect the debug hub at User Scan Chain of 2 or 4.
To determine
the user scan chain setting, open the implemented design and use:
get_property
C_USER_SCAN_CHAIN [get_debug_cores dbg_hub].
```

デバッグ コア クロックがアクティブでなかったり、使用できない状態の場合は、Vivado ハードウェア マネージャーで次のエラー メッセージが表示されます。

```
INFO: [Labtools 27-2302] Device xxx (JTAG device index = 1) is programmed
with a
design that has 1 ILA core(s).
CRITICAL WARNING: [Labtools 27-1433] Device xxx (JTAG device index = 1) is
programmed
with a design that has an unrecognizable debug core (slave type = 17) at
user chain
= 1, index = 0.
Resolution:
1) Ensure that the clock signal connected to the debug core and/or debug
hub is clean
and free-running.
2) Ensure that the clock connected to the debug core and/or debug hub meets
all timing
constraints.
3) Ensure that the clock connected to debug core and/or debug hub is faster
than the
JTAG clock frequency.
```

次の図は、2 つの ILA コアを含むデザイン例を示しています。

図 92: デバッグ コアのクロッキング例



このデザイン例には、ILA "A" および ILA "B" という 2 つの ILA コアが含まれます。

このデバッグ ネットワークのクロッキング トポロジは次のとおりです。デザインをデバイスにプログラムすると、Vivado ハードウェア マネージャーはデザインにデバッグ ハブ コアがあるかどうかを検出します。反対に、デバッグ ハブでは、接続されているデバッグ コアすべてを検出します。このデザインの場合、デバッグ コアは ILA "A" と ILA "B" です。CLKA は ILA "A" とデバッグ ハブ コアの両方を駆動し、CLKB は ILA "B" デバッグ コアを駆動しています。

ターゲットに接続してデバイスをプログラムしたら、JTAG クロックがアクティブになるはずですが、デバッグ コアの検出段階では、フリーランニングで安定したクロック (この例の場合 CLKA) がデバッグ ハブ コアを駆動するはずですが、デバッグ コアの測定段階では、アクティブな JTAG クロック、デバッグ ハブ クロック、およびデバッグ コア クロックが駆動し、ILA "B" でデータをトリガーして取り込む場合は、フリーランニングで安定した JTAG クロック、デバッグ ハブ クロック (CLKA)、およびデバッグ コア クロック (CLKB) が駆動するはずですが、

パーシャル リコンフィギュレーション デザインへの Vivado デバッグ コアの追加

Vivado デバッグ コアは、リコンフィギュラブル モジュール内も含めて、パーシャル リコンフィギュレーション デザインにインスタンス化できます。これらのコアを追加して接続するには、特定の要件と設計手法を使用する必要があります。これらの Vivado デバッグ コアを追加して接続するのに必要な設計手法については、『Vivado Design Suite ユーザー ガイド: Dynamic Function eXchange』 (UG909) の[このセクション](#)を参照してください。

Dynamic Function eXchange デザインにデバッグ コアをインスタンス化する場合および Vivado ハードウェア マネージャー内の機能の詳細は、『Vivado Design Suite チュートリアル: Dynamic Function eXchange』 (UG947) の[このセクション](#)を参照してください。

ハードウェアでのロジック デザインのデバッグ

デザインにデバッグ コアを追加したら、ランタイム ロジック解析機能を使用して、ハードウェア上でデザインをデバッグできます。

Vivado ロジック解析を使用したデザインのデバッグ

Vivado[®] ロジック解析機能は、デザインに含まれる新しい ILA、VIO、および JTAG-to-AXI Master デバッグ コアにアクセスするために使用します。Vivado ロジック解析機能を使用するには、Flow Navigator で [PROGRAM AND DEBUG] → [Open Hardware Manager] をクリックします。

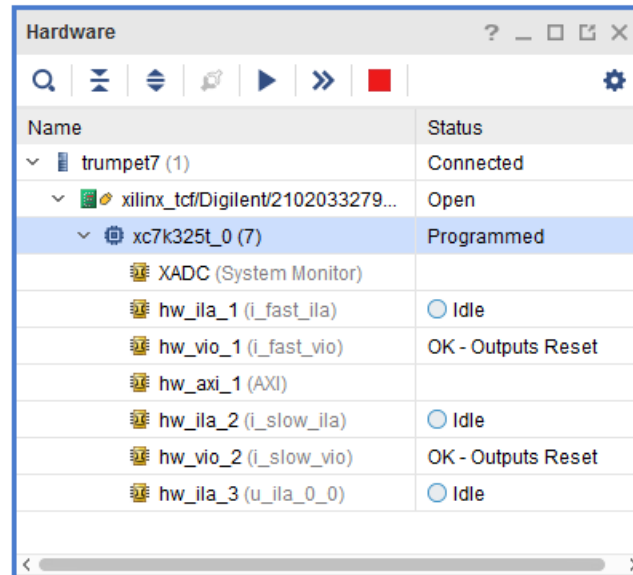
ILA デバッグ コアを使用したハードウェアでのデザインのデバッグ手順は、次のとおりです。

1. ハードウェア ターゲットに接続し、FPGA デバイスを `.bit` ファイルでプログラムします。
2. ILA デバッグ コアのトリガーおよびキャプチャ条件を設定します。
3. ILA デバッグ コアをトリガー待機状態にします。
4. 波形ビューアーで ILA デバッグ コアからのデータを表示します。
5. VIO デバッグ コアを使用して制御信号を駆動し、デザインのステータス信号を確認します。
6. JTAG-to-AXI Master デバッグ コアを使用して、デザイン内のさまざまな AXI スレーブ コアにアクセスするトランザクションを実行します。

ハードウェア ターゲットに接続してデバイスをプログラム

デバッグの前に FPGA デバイスをプログラムする手順は、「FPGA のプログラム」で説明されている手順と同じです。新しい ILA、VIO、および JTAG-to-AXI Master デバッグ コアを含む `.bit` ファイルでデバイスをプログラムすると、[Hardware] ウィンドウにデバイスのスキャンで検出されたデバッグ コアとその RTL インスタンス名がカッコ内に表示されます。

図 93: デバッグ コアが表示された [Hardware] ウィンドウ



ILA コアの使用に関する詳細は、「計測のための ILA コアの設定」を参照してください。VIO コアの使用に関する詳細は、「計測のための VIO コアの設定」を参照してください。



重要: JTAG クロックがデバッグ コアのクロック入力よりも低速であることを確認してください。JTAG クロック周波数を変更するには、Open New Hardware Target ウィザードを使用するか、Tcl コマンドの `set_property PARAM.FREQUENCY 250000 [get_hw_targets */xilinx_tcf/Digilent/210203327962A]` を使用します。

関連情報

[デバイスのプログラム](#)

[計測のための ILA コアの設定](#)

[計測のための VIO コアの設定](#)

Vivado ハードウェア マネージャーのダッシュボード

Vivado ハードウェア マネージャーのダッシュボードを使用すると、システム モニター、ILA、VIO デバッグ コアのさまざまなウィンドウを管理しやすくなります。ダッシュボードを使用して、Vivado Design Suite プロジェクトのこれらのウィンドウの設定を作成、変更、保存できます。

デフォルトのダッシュボード

ハードウェア デバイスを更新したときにデバッグ コアが検出されると、各デバッグ コアに対してデフォルトのダッシュボードが自動的に開きます。

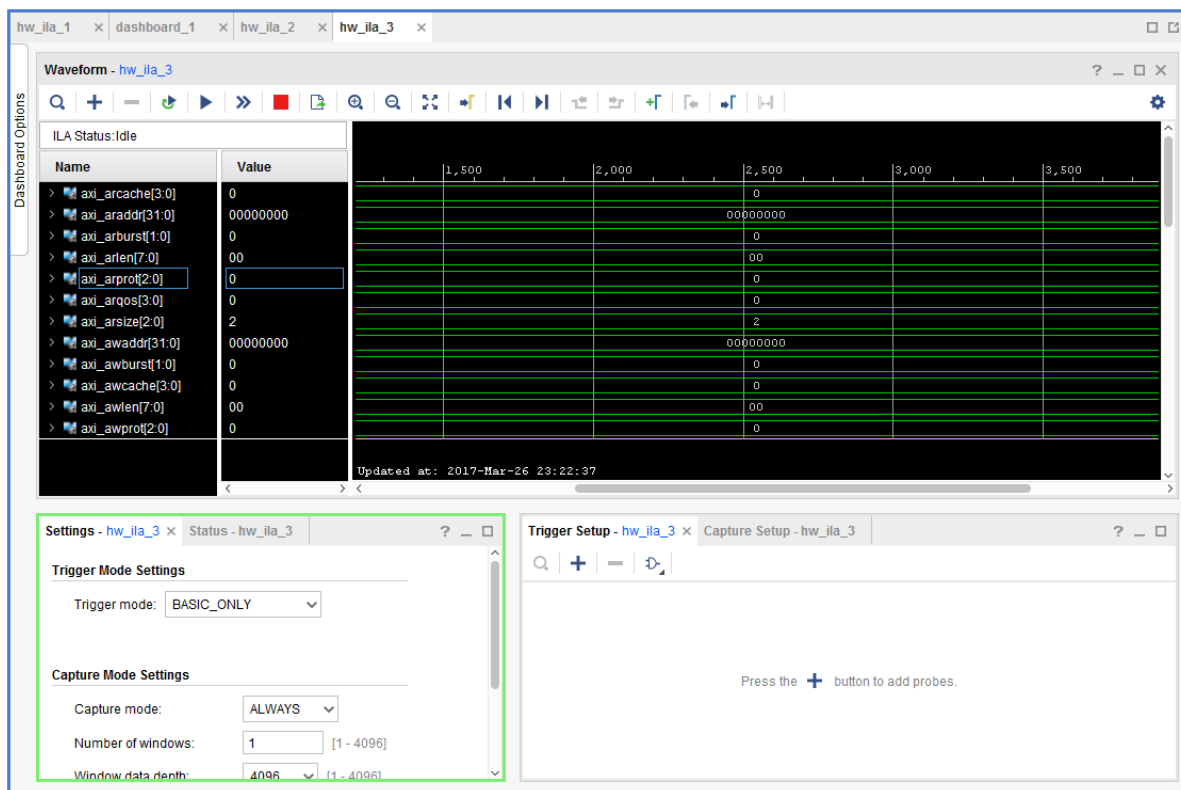
デフォルトのダッシュボード ウィンドウ

デフォルト ダッシュボードには、該当するデバッグ コアに関連するウィンドウが含まれます。ILA デバッグ コアに対して作成されたデフォルトのダッシュボードには、次の 5 つのウィンドウが含まれています。

- [Settings] ウィンドウ
- [Status] ウィンドウ
- [Trigger Setup] ウィンドウ
- [Capture Setup] ウィンドウ
- [Waveform] ウィンドウ

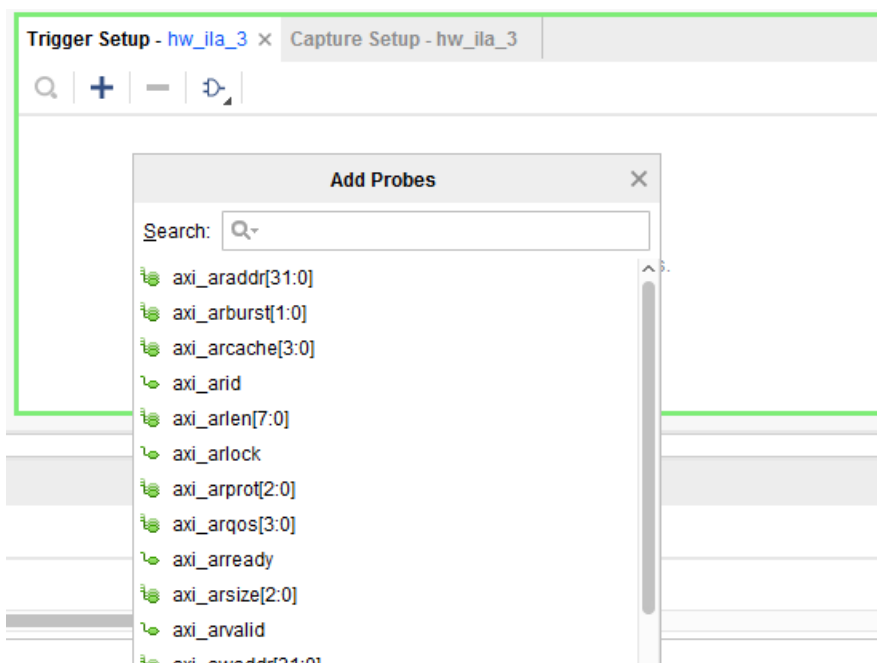
次の図に、デフォルトの ILA ダッシュボードの例を示します。

図 94: デフォルトの ILA ダッシュボード



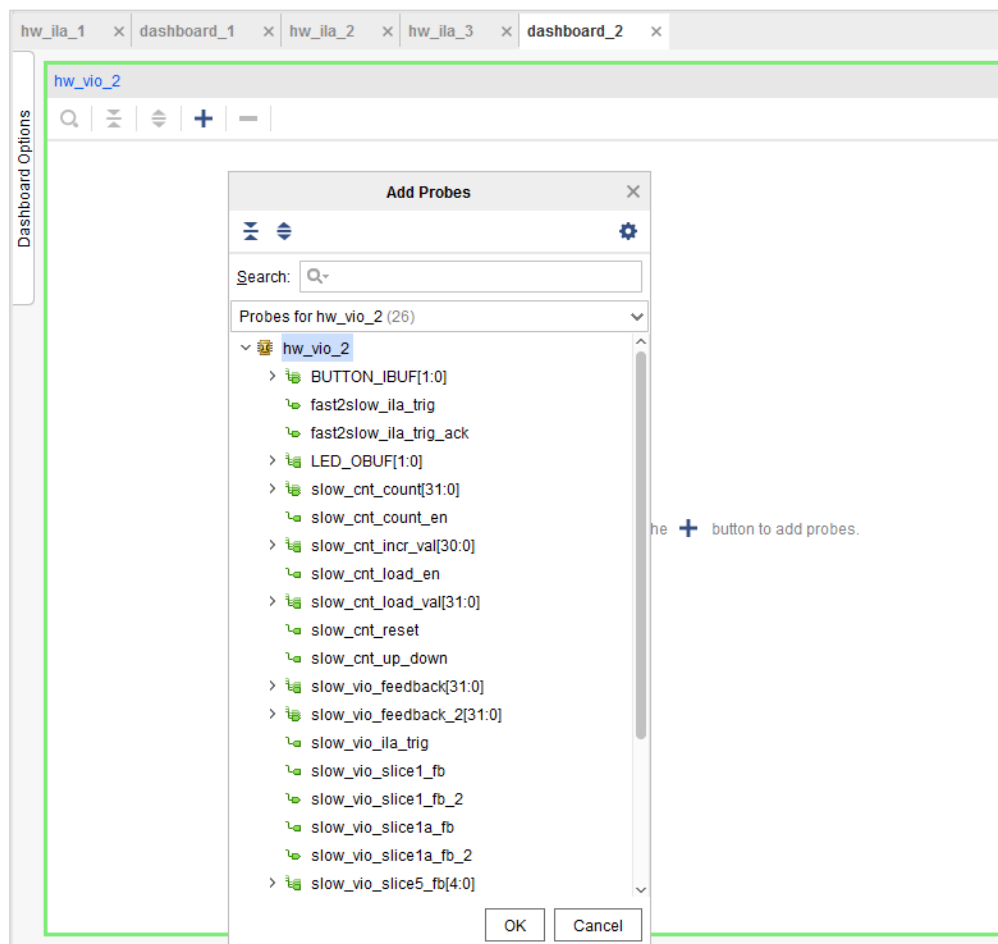
まず、[Trigger Setup] ウィンドウの中央にある [+] 記号をクリックし、[Add Probes] ウィンドウでプローブを選択して追加します。

図 95: [Add Probes] ウィンドウ



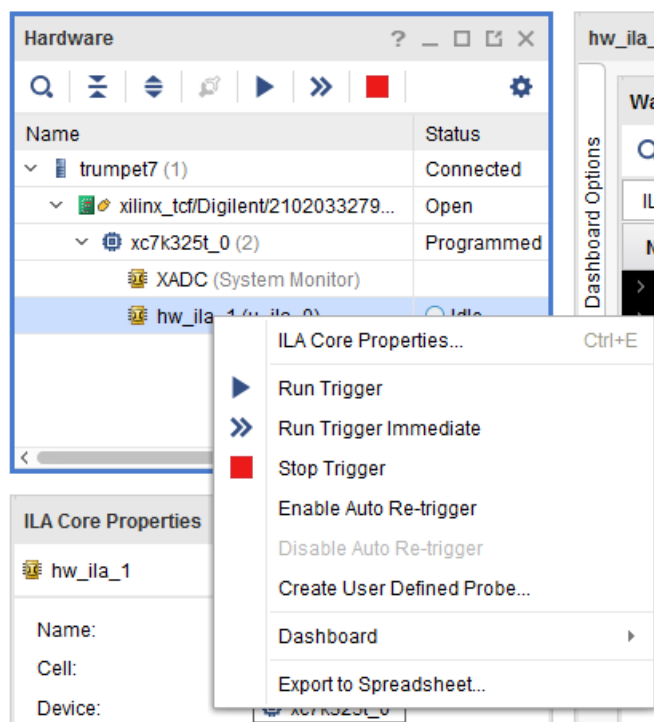
VIO のデフォルト ダッシュボードは、次の図に示すように最初は空白です。

図 96: VIO プロープの追加



デバッグ コアに関連付けられているダッシュボードを表示するには、[Hardware] ウィンドウでデバッグ コア オブジェクトを右クリックし、[Dashboard] をクリックしてダッシュボード名を選択します。[Hardware] ウィンドウでデバッグ コアをダブルクリックすると、そのコアに関連付けられているダッシュボードが開きます。

図 97: 関連付けられているダッシュボード



ダッシュボード内のウィンドウ制御

各ウィンドウのタイトル バーには、ウィンドウを操作できるボタンが含まれます。

- 最小化
- 最大化
- 閉じる

ウィンドウの移動

ウィンドウを移動するには、次の手順に従います。

1. ウィンドウのタブまたはタイトル バーを選択し、ドラッグします。グレーのアウトラインでウィンドウがどこに移動されるかが示されます。
2. マウス ボタンを放してウィンドウを配置します。

注記: 既存のウィンドウの上に別のウィンドウをドロップすると、同じ場所に 2 つのウィンドウ タブが表示されます。



重要: ウィンドウをワークスペース 外に移動することはできませんが、ワークスペース 内でサイズを変更したり、移動したりすることはできます。

ウィンドウのサイズ変更

- ウィンドウのサイズを変更するには、ウィンドウの枠をクリックしてドラッグします。

注記: ウィンドウの枠にカーソル置くと、カーソルが矢印からドラッグ用マークに変わるので、ドラッグしてサイズを変更します。

- ウィンドウを最大化するには、右上の [Maximize] ボタンをクリックします。
- ウィンドウを元のサイズに戻すには、ウィンドウのタイトル バーまたはタブをダブルクリックします。

ウィンドウを閉じる

- ウィンドウを閉じるには、ウィンドウ右上の [Close] ボタンをクリックします。
注記: このボタンがウィンドウ タブに表示されることもあります。
- ウィンドウのタブまたはタイトル バーを右クリックし、[Close] をクリックします。

ウィンドウ タブ

各ウィンドウにはタブがあり、タブをクリックするとそのタブがアクティブになります。[Trigger Setup] ウィンドウや [Capture Setup] ウィンドウなどのウィンドウでは、タブは下に表示されます。



ヒント: 次のタブを選択するには、Ctrl + Tab キーを押します。前のタブを選択するには、Ctrl + Shift + Tab キーを押します。ウィンドウを最大化または最小化するには、ウィンドウのタブをダブルクリックします。

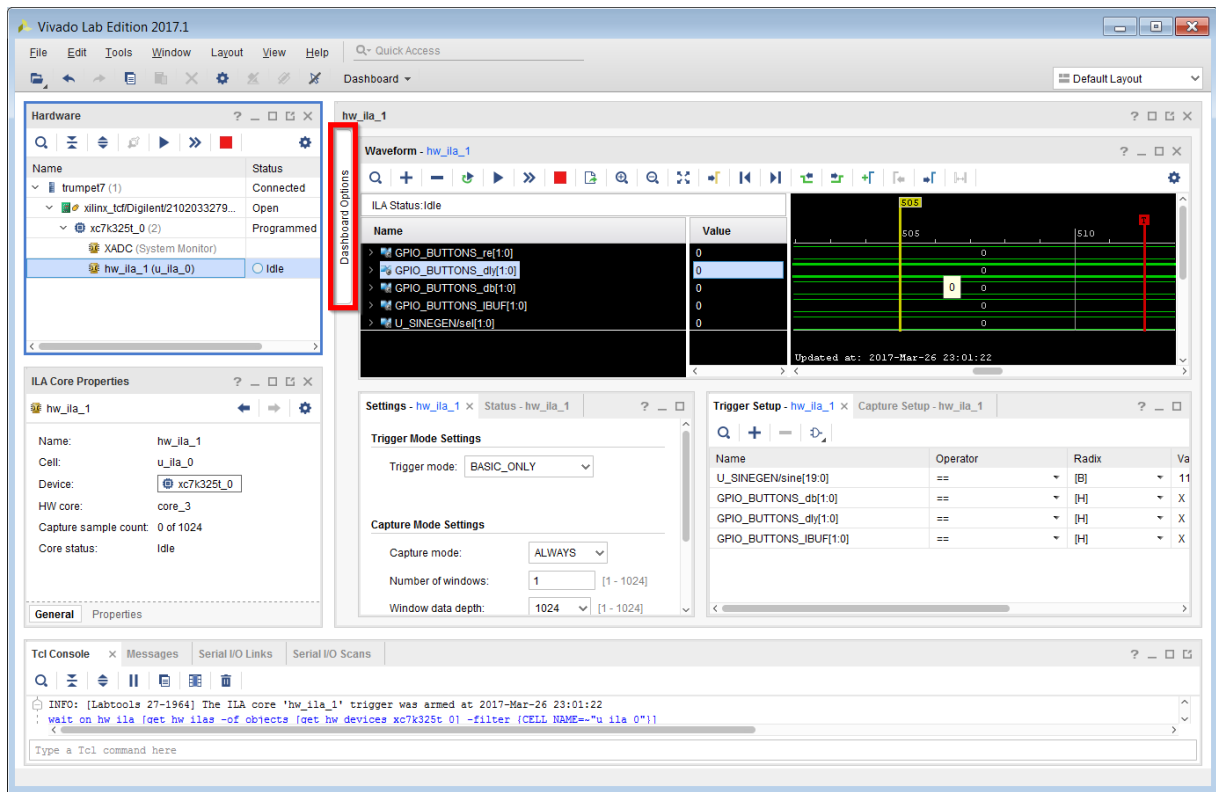
ダッシュボードのカスタマイズ

デザインをデバッグしたり、結果を表示するには、通常デフォルト ダッシュボードのウィンドウで十分ですが、ウィンドウを移動するなどダッシュボードをカスタマイズする場合があります。たとえば、ILA のステータスと [Waveform] ウィンドウの両方を表示しつつ、同じダッシュボードで VIO プロープを制御する場合などです。このような場合、必要に応じてダッシュボードをカスタマイズすることをお勧めします。

ダッシュボードのオプション

各ダッシュボードの左側には、ダッシュボード オプションを表示する [Dashboard Options] スライドアウトがあります。ダッシュボード左側の [Dashboard Options] ボタンをクリックすると、この [Dashboard Options] スライドアウトが開きます。[Dashboard Options] では、特定ダッシュボードに表示されるウィンドウを制御できます。たとえば、ILA ダッシュボードに VIO ウィンドウの 1 つを含めるようカスタマイズするとします。[Dashboard Options] で含める VIO ウィンドウをクリックすると、ILA ダッシュボードに表示されるようになります。これで VIO プロープを追加して、ILA ウィンドウをトリガーできます。

図 98: ダッシュボード オプションの追加

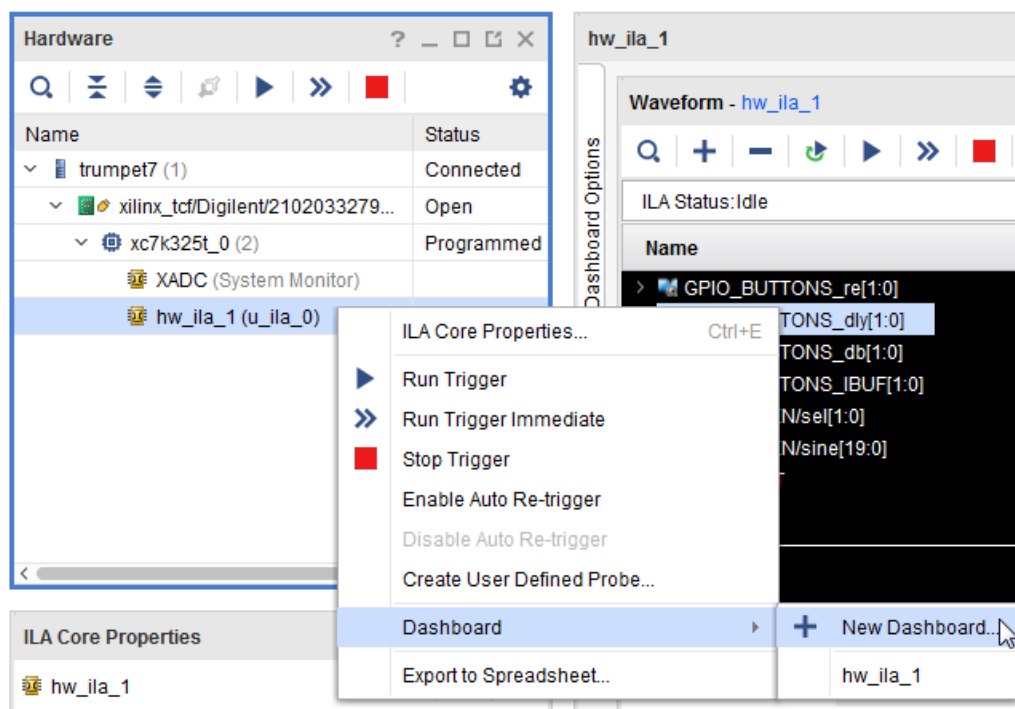


ダッシュボードの左側の [Dashboard Options] ボタンをクリックすると、[Dashboard Options] スライドアウトを開いたり閉じたりできます。

新規ダッシュボードの作成

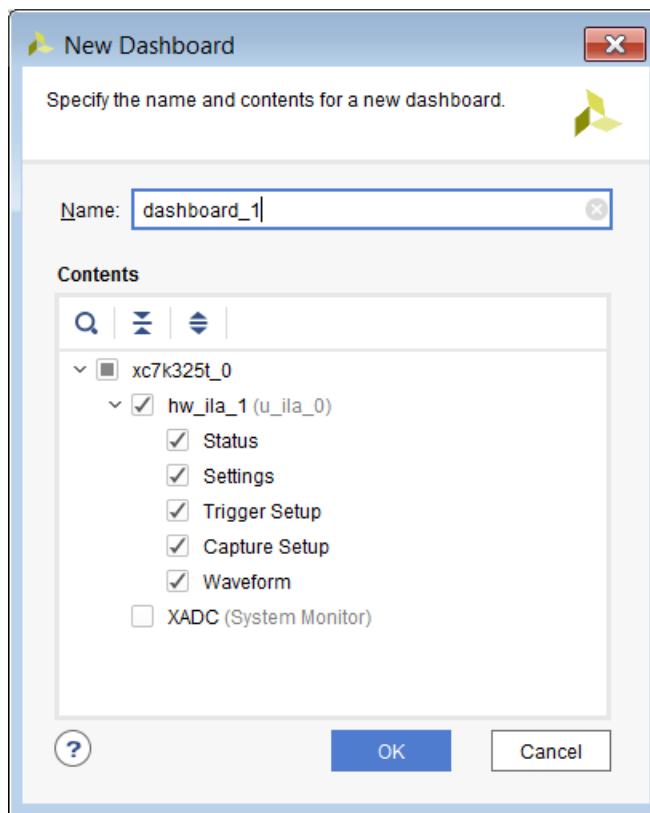
[Dashboard Options] を使用してデフォルトのダッシュボードをカスタマイズするだけでなく、新しいダッシュボードを作成することもできます。これには、[Hardware] ウィンドウでデバッグ コア オブジェクトを右クリックして [Dashboard]→[New Dashboard] をクリックします。

図 99: 新規ダッシュボードの作成



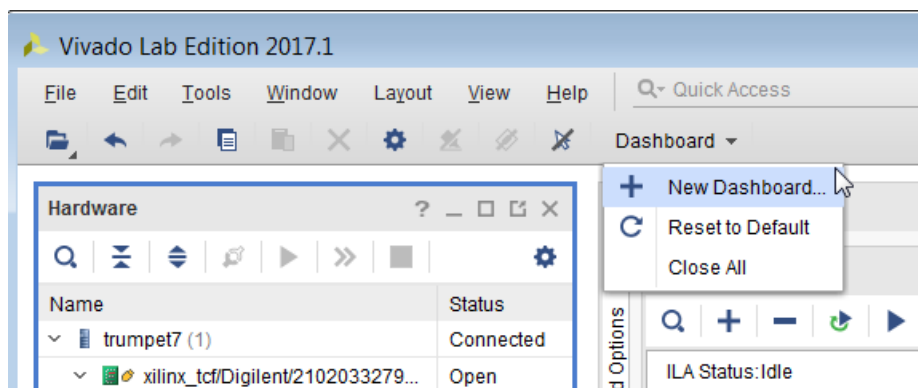
[New Dashboard] ダイアログ ボックスが表示されたら、必要に応じてダッシュボードをカスタマイズし、[OK] をクリックします。

図 100: [New Dashboard] ダイアログ ボックス



新しいダッシュボードは、次の図に示すように、ツールバーの [Dashboard] ボタンをクリックして作成することもできます。

図 101: ツールバーの [Dashboard] ボタン



ヒント: デバッグ コアに関連付けられているダッシュボードをすべて表示するには、[Hardware] ウィンドウでデバッグ コアを右クリックし、[Dashboard] をクリックします。または、[Hardware] ウィンドウでデバッグ コアをダブルクリックすると、そのコアに関連付けられているダッシュボードのリストが開きます。

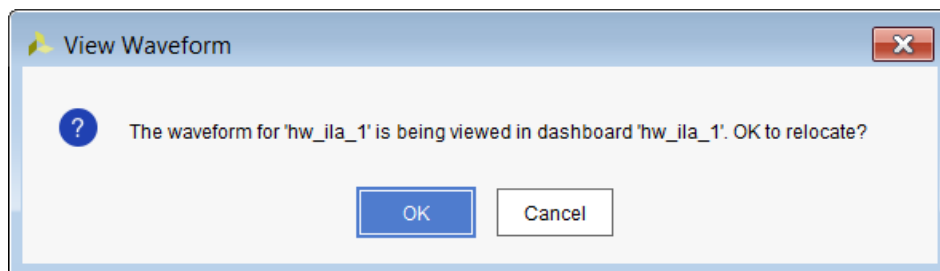


ヒント: ダッシュボードの 1 つのウィンドウをフロートさせる場合は、そのウィンドウだけのダッシュボードを作成して、そのダッシュボードをフロートさせることをお勧めします。

ダッシュボードでの ILA の [Waveform] ウィンドウ

ILA の [Waveform] ウィンドウは、1 つのダッシュボードに 1 つしか表示できません。別のダッシュボードにある [Waveform] ウィンドウをクリックすると、ウィンドウを移動させるかどうかを確認するメッセージが表示されます。

図 102: ILA の [Waveform] ウィンドウの移動を確認するメッセージ



[OK] をクリックすると、[Waveform] ウィンドウは指定のダッシュボードに移動します。

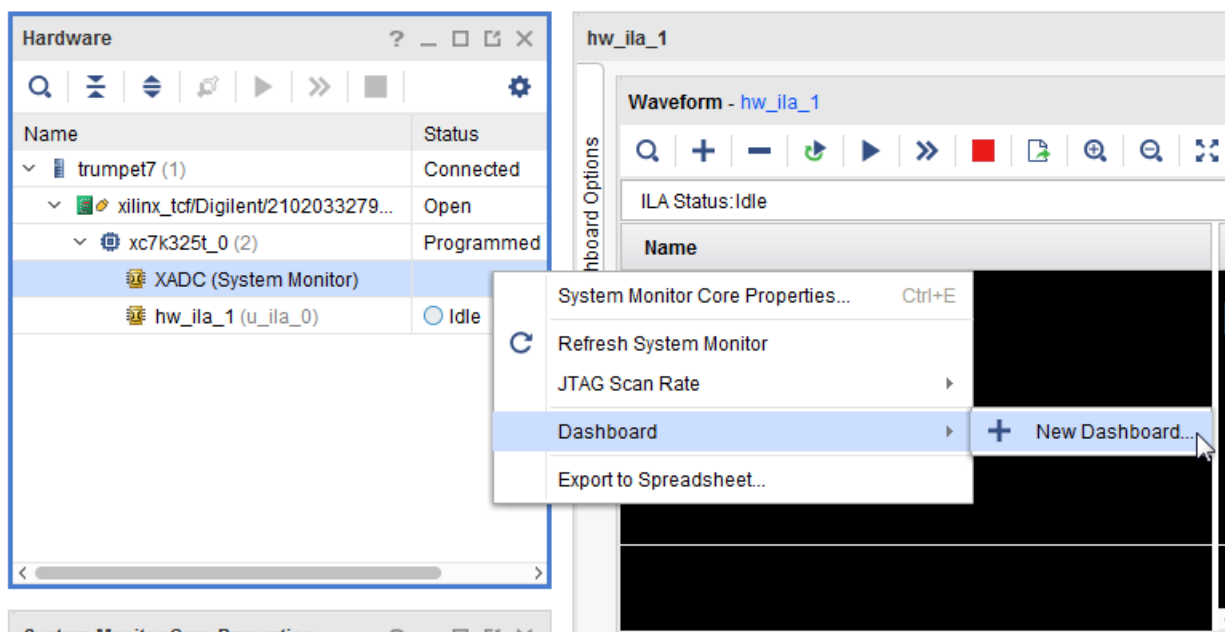


ヒント: [Waveform] ウィンドウを閉じる際は、ILA データを保存してください。

システム モニターのダッシュボード

XADC/システム モニターのウィンドウを別のダッシュボードに含めたり、それ自体のダッシュボードを作成することもできます。

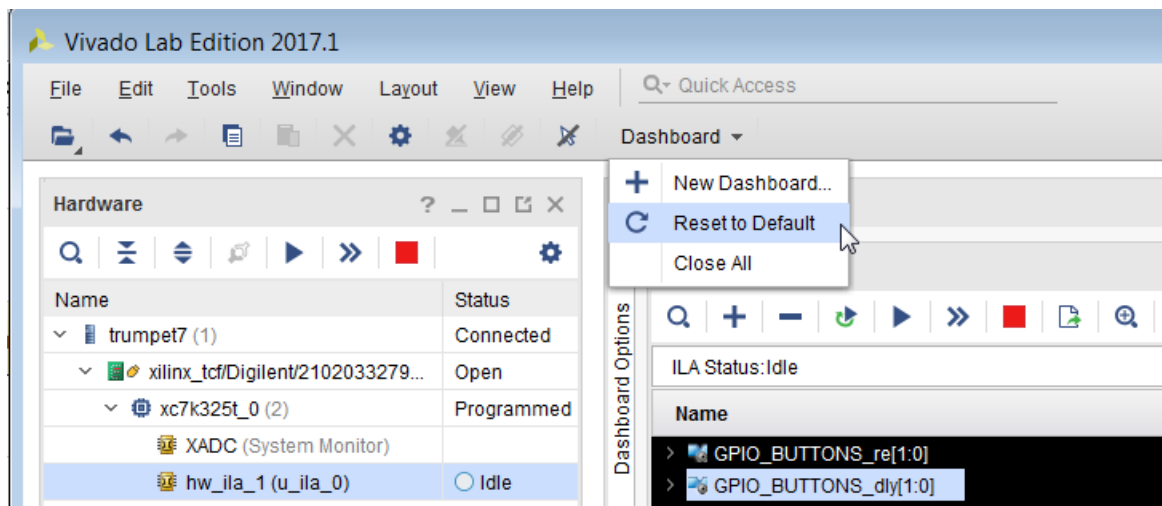
図 103: システム モニターのダッシュボード



デフォルト ダッシュボードへのリセット

ダッシュボードは、ツールバーの [Dashboard] をクリックして [Reset to Default] をクリックすることにより、デフォルトの状態にリセットできます。

図 104: デフォルト ダッシュボードへのリセット



ダッシュボードを閉じる

ツールバーの [Dashboard] をクリックし、[Close All] をクリックすると、開いているダッシュボードをすべて閉じることができます。ダッシュボードはすべて削除され、これらのダッシュボードのユーザー設定も削除されます。

1 つのダッシュボードのみを閉じるには、ウィンドウの右上にある [X] をクリックします。ダッシュボードとそのユーザー設定がすべて削除されます。

ユーザー ダッシュボード設定の保存

ユーザー ダッシュボード設定は、Vivado IDE で自動的に保存されます。プロジェクトを閉じて再び開くと、ハードウェア マネージャーにユーザー 設定が復元されます。

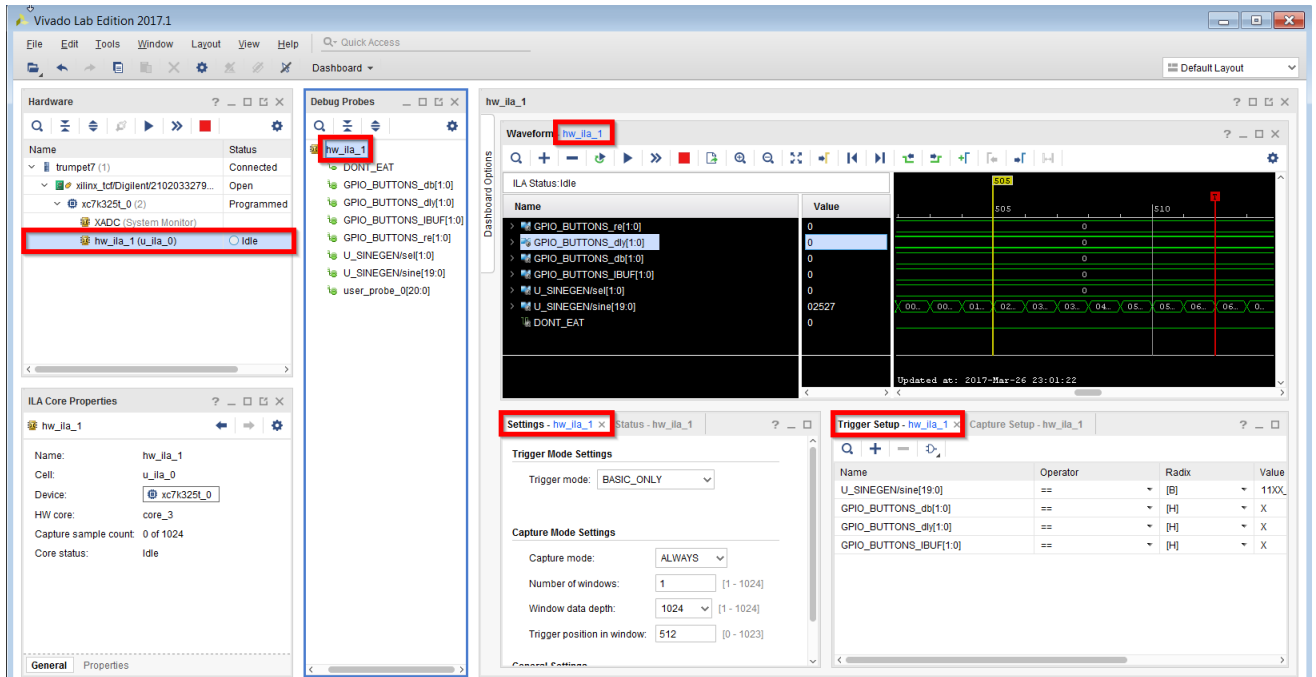
計測のための ILA コアの設定

デザインに追加した ILA コアは、[Hardware] ウィンドウのターゲット デバイスの下に表示されます。ILA コアが表示されていない場合は、デバイスを右クリックして [Refresh Device] をクリックします。FPGA デバイスが再度スキャンされ、[Hardware] ウィンドウの表示が更新されます。

注記: FPGA デバイスをプログラムまたは更新しても ILA コアが表示されない場合は、デバイスが正しい .bit ファイルでプログラムされているか、インプリメント済みデザインに ILA コアが含まれているかを確認してください。また、.bit ファイルに対応する適切な .ltx プローブ ファイルがデバイスに関連付けられているかどうかを確認してください。

ILA コア (次の図では hw_ila_1) を選択すると、[ILA Core Properties] ウィンドウにプロパティが表示されます。ILA コアに対応するプローブをすべて表示するには、[Windows]→[Debug Probes] を使用し、[Debug Probes] ウィンドウを開きます。

図 105: さまざまなウィンドウでの ILA コアを選択



プローブの追加

ILA ダッシュボードの特定のウィンドウに関連プローブを追加するには、ウィンドウのツールバーまたはワークスペースの [+] 記号をクリックします。

デバッグ プローブ情報の書き込み

[Debug Probes] ウィンドウには、ILA および VIO コアを使用してプローブされるデザインのネットに関する情報が表示されます。このデバッグ プローブ情報はデザインから抽出され、通常 .ltx という拡張子のデータ ファイルに保存されます。

デバッグ プローブ ファイルは、インプリメンテーション プロセス中に自動的に作成されますが、write_debug_probes Tcl コマンドを使用してデバッグ プローブ情報をファイルに書き出すこともできます。

1. 合成済みデザインまたはネットリスト デザインを開きます。
2. write_debug_probes filename.ltx コマンドを実行します。



重要: 非プロジェクト モードを使用している場合は、write_debug_probes コマンドのすぐ後に opt_design コマンドを手動で実行する必要があります。

デバッグ プローブ情報の読み出し

デバッグ プローブ ファイルは、Vivado IDE がプロジェクト モードで、`debug_nets.ltx` というプローブ ファイルがデバイスに関連付けられているビットストリーム プログラム ファイル(.bit)と同じディレクトリにある場合、自動的にハードウェア デバイスに関連付けられます。

プローブ ファイルの場所を指定するには、次の手順に従います。

1. [Hardware] ウィンドウでハードウェア デバイスを選択します。
2. [Hardware Device Properties] ウィンドウでプローブ ファイルのディレクトリを設定します。
3. [Hardware] ウィンドウでハードウェア デバイスを右クリックして [Refresh Device] をクリックし、デバッグ プローブ ファイルの内容を読み出し、ハードウェア デバイスで実行されるデザイン内のデバッグ コアとその情報を関連付けて検証します。

次の Tcl コマンドを使用してディレクトリを設定し、`C:\myprobes.ltx` というデバッグ プローブ ファイルをターゲット ボードの最初のデバイスに関連付けることもできます。

```
% set_property PROBES.FILE {C:/myprobes.ltx} [lindex [get_hw_devices] 0]
% refresh_hw_device [lindex [get_hw_devices] 0]
```

デバッグ プローブ名の変更

[Debug Probes] ウィンドウで、ILA または VIO コアのデバッグ プローブ名を変更できます。デバッグ プローブは、コアの既存の波形ビューアーに追加するか、ILA ダッシュボードのさまざまなトリガーおよびキャプチャ ウィンドウに追加できます。これらの名前は、デバッグ プローブに関連付けられたカスタム、ロング、ショートいずれかの形式にできます。

これを実行するには、ILA/VIO コアのデバッグ プローブを右クリックし、次のいずれかをクリックします。

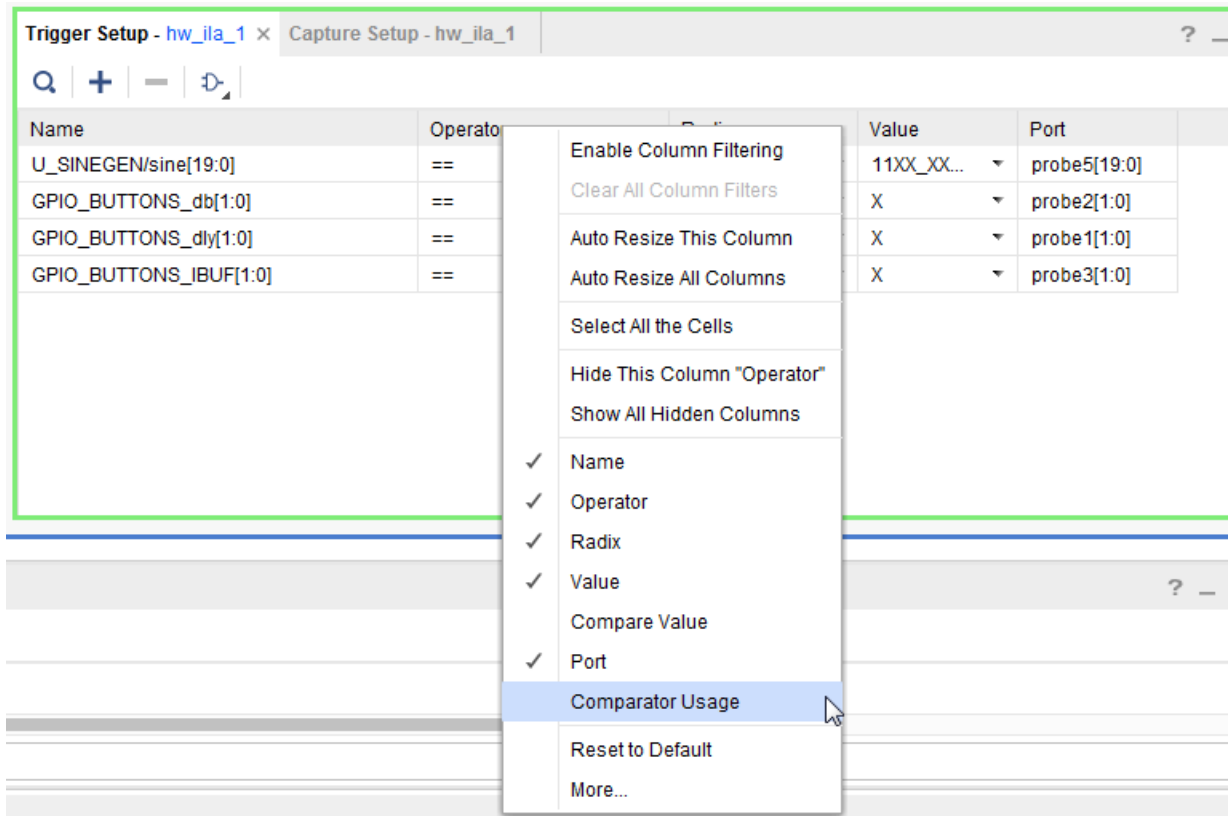
- [Rename]: プローブ名を変更します。
- [Name]: デバッグ プローブ名をロング、ショート、カスタムのいずれかの形式にします。Vivado IDE のウィンドウでは、デバッグ プローブがここで選択した名前で表示されます。
 - [Long]: プローブする信号またはバスの完全な階層名を表示します。
 - [Short]: プローブする信号またはバスの名前を表示します。
 - [Custom]: 信号またはバスの名前を変更したときに付けられたカスタム名を表示します。

複数のコンパレータの使用

プローブまたは ILA デバッグ コアをカスタマイズして BASIC/ADVANCED モードで複数のコンパレータを使用する場合は、これらのコンパレータを [Basic Trigger Setup] ウィンドウおよび [Advanced Trigger Setup] ウィンドウで使用できます。

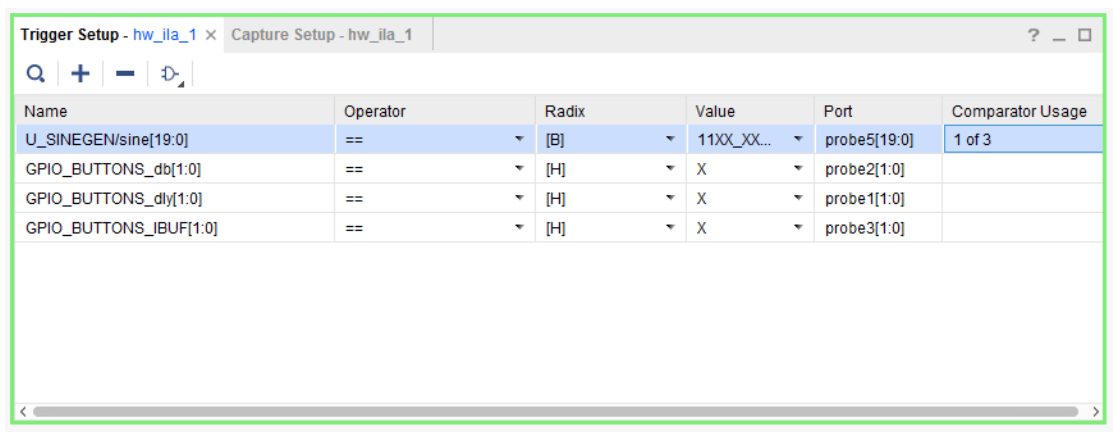
[Basic Trigger Setup] ウィンドウにプローブを追加して、トリガー条件を設定できます。[Trigger Setup] ウィンドウの [Comparator Usage] 列に、プローブに関連付けられているコンパレータの総数のうち、どのコンパレータが特定の比較条件に使用されかが示されます。

図 106: [Trigger Setup] ウィンドウ: [Comparator Usage] 列



ヒント: [Comparator Usage] は非表示の列です。これを表示させるには、次のように [Trigger Setup] 列の見出し行を右クリックし、[Comparator Usage] を選択します。

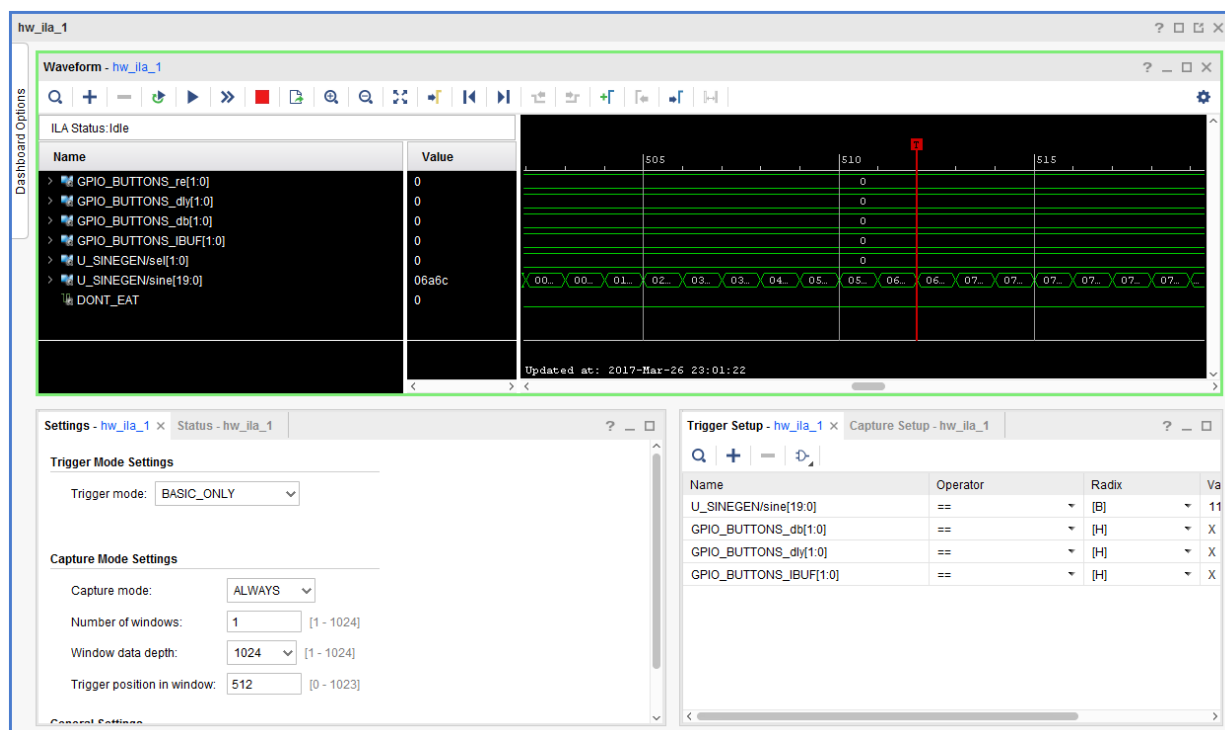
図 107: [Comparator Usage] 列



ILA デフォルト ダッシュボードの使用

ILA ダッシュボードには、ILA コアに関するすべてのステータスおよび制御情報が表示されます。ハードウェア デバイスの更新により ILA コアが最初に検出されると、ILA デフォルト ダッシュボードが自動的に開きます。ダッシュボードを手動で開いたり、開き直したりする必要がある場合は、[Hardware] ウィンドウで ILA コア オブジェクトを右クリックし、[Default Dashboard] をクリックします。

図 108: ILA ダッシュボード



ILA ダッシュボードから、ILA デバッグ コアを次のように操作できます。

- トリガー モード ([Trigger mode]) を設定して、トリガーするハードウェアのイベントを指定します。
 - [BASIC_ONLY]: デバッグ プローブ 比較結果の基本的な AND/OR 機能が満たされたときに ILA コアをトリガーします。
 - [ADVANCED_ONLY]: ILA コアをユーザー定義のステート マシンで指定したようにトリガーします。
 - [TRIG_IN_ONLY]: ILA コアの TRIG_IN ピンが Low から High に遷移したときに ILA コアをトリガーします。
 - [BASIC_OR_TRIG_IN]: ILA コアの TRIG_IN ピンと BASIC_ONLY トリガー モードの論理 OR の結果により ILA コアをトリガーします。
 - [ADVANCED_OR_TRIG_IN]: ILA コアの TRIG_IN ピンと ADVANCED_ONLY トリガー モードの論理 OR の結果により ILA コアをトリガーします。
- トリガー出力モード ([TRIG_OUT mode]) を設定します。
- キャプチャ モード ([Capture mode]) を [ALWAYS] または [BASIC] に設定して、キャプチャされるデータをフィルター処理します。
- ILA キャプチャ ウィンドウ ([Number of windows]) の数を設定します。
- ILA キャプチャ ウィンドウのデータの深さ ([Window data depth]) を設定します。

- トリガー位置 ([Trigger position in window]) をキャプチャ ウィンドウ内のサンプルに設定します。
- ILA デバッグ コアのトリガーおよびキャプチャ ステータスを監視します。

ユーザー定義のデバッグ プローブ

ハードウェア マネージャーでユーザー 定義のデバッグ プローブ (hw_probe と呼ばれる) を使用すると、物理的な ILA プローブ ポートと定数値を組み合わせることでプローブを作成できます。ハードウェア マネージャーの [Trigger Setup] または [Waveform] ウィンドウでこれらのプローブを使用できます。これらのプローブが正しく作成されると、関連付けられたデバッグ コアの一部として [Debug Probes] ウィンドウにリストされます。

次のタイプのユーザー定義プローブを作成できます。

- ILA プローブ ポート。
- 複数の定数値 (0 または 1)。
- ILA プローブ ポートと定数値の組み合わせ。



重要: 定数値を含むユーザー定義プローブは、[Waveform] ウィンドウでのみ使用可能です。[Trigger Setup] ウィンドウでは使用できません。



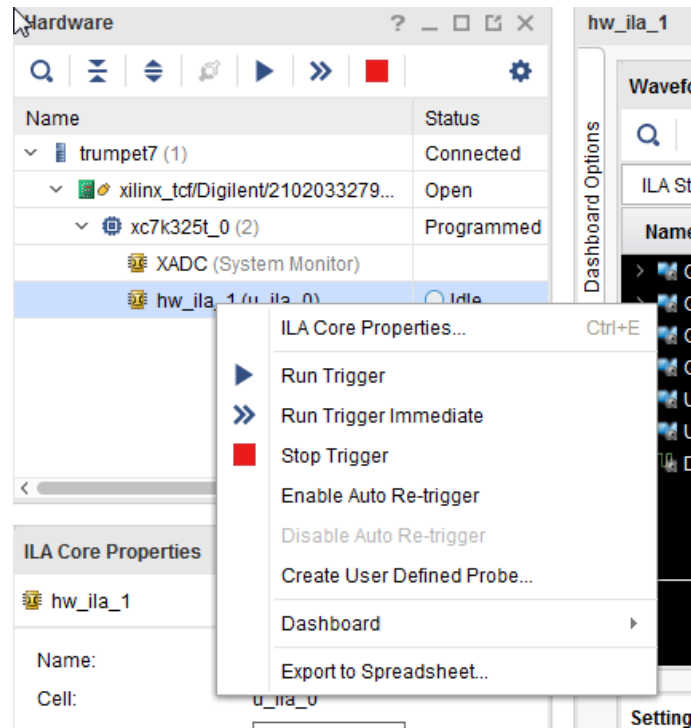
ヒント: ユーザー定義プローブは、ILA デバッグ コアにのみ作成可能です。VIO コアのユーザー定義デバッグ プローブの作成はサポートされていません。

ユーザー定義のデバッグ プローブの作成

GUI フロー

Vivado IDE ハードウェア マネージャーでユーザー 定義のデバッグ プローブを作成するには、[Hardware] ウィンドウでプローブする ILA コアを右クリックして [Create User Defined Probe] を選択します。

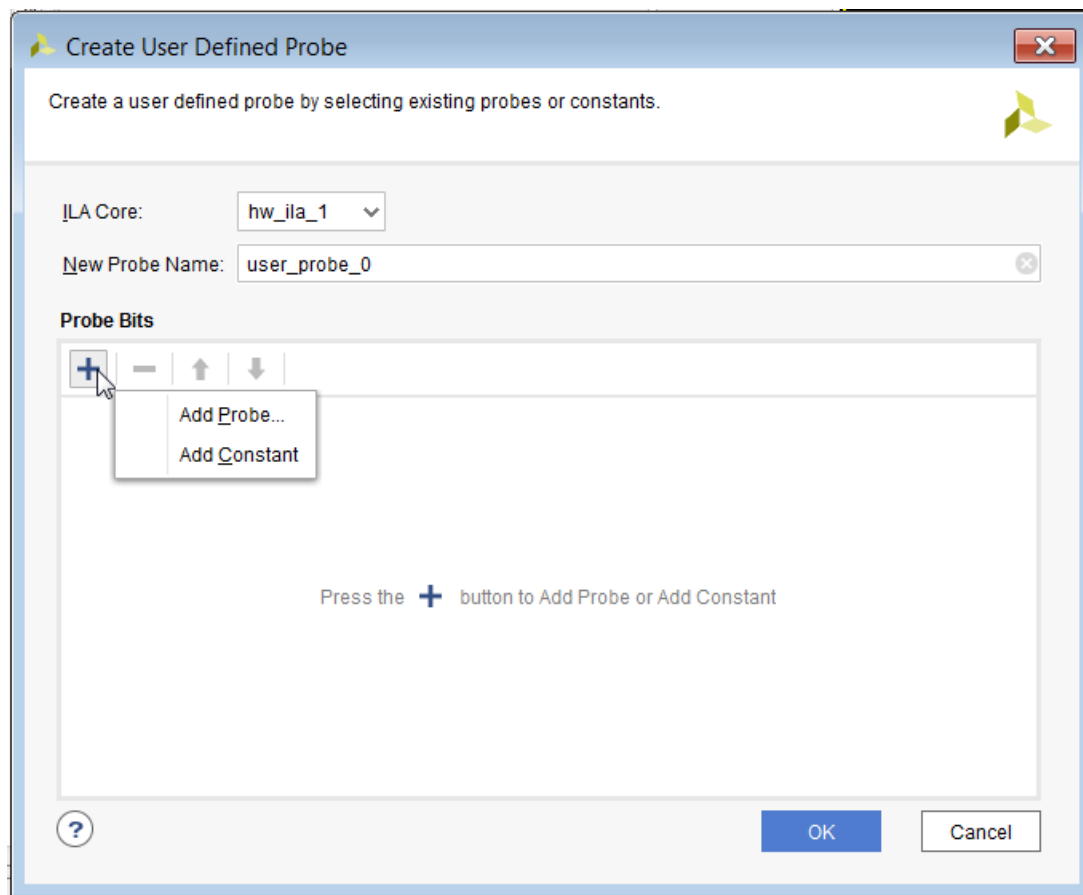
図 109: [Create User Defined Probe] を選択



これで、[Create User Defined Probe] ダイアログ ボックスが開きます。プローブを作成する ILA コアを選択し、新しいプローブの名前を入力し、プローブ ビットおよびこの新プローブの制約を設定します。

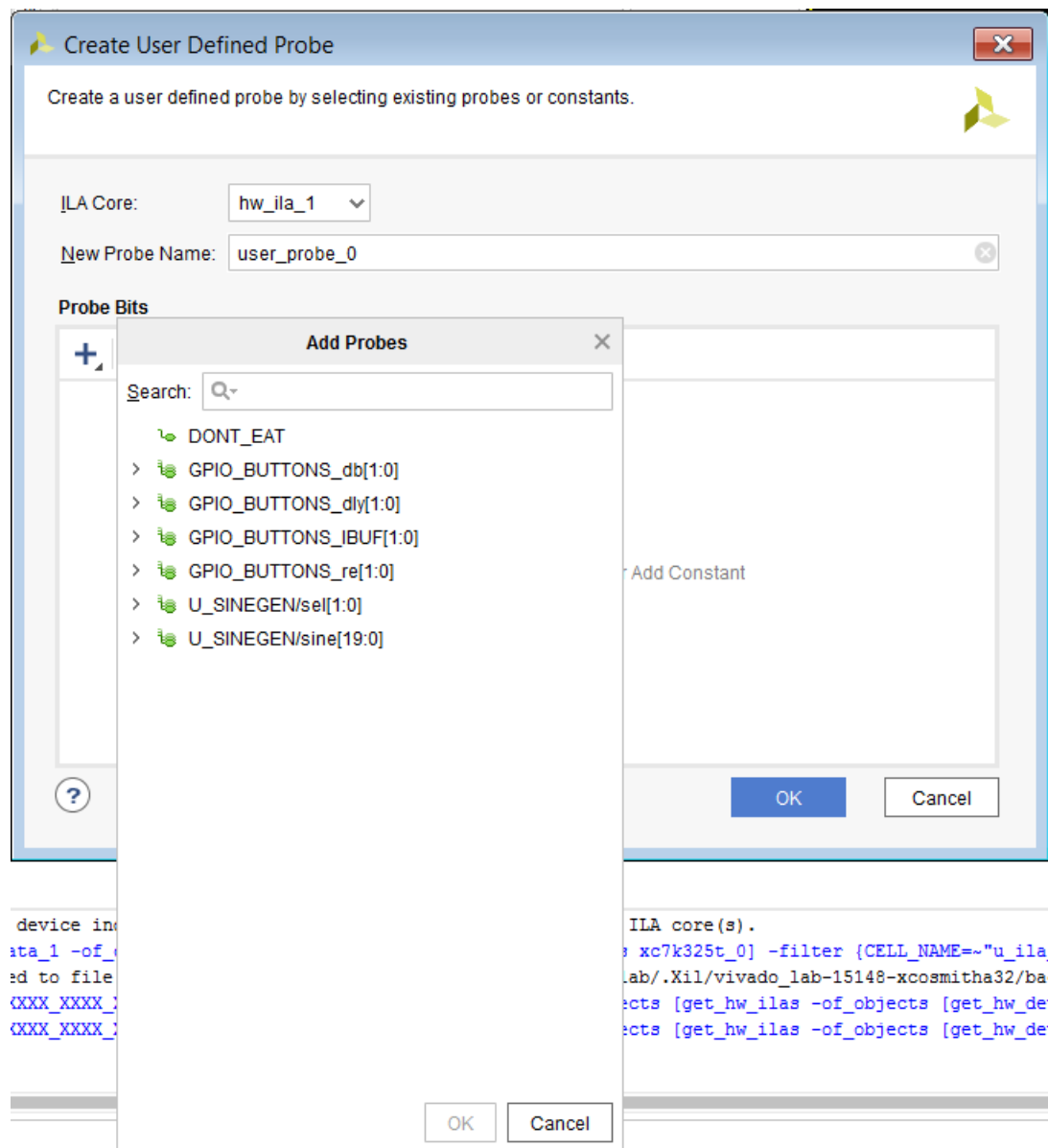
この新プローブに特定のプローブ ビットを追加するには [+] ボタンをクリックし、[Add Probe] を選択します。

図 110: [Create User Defined Probe] ダイアログ ボックス



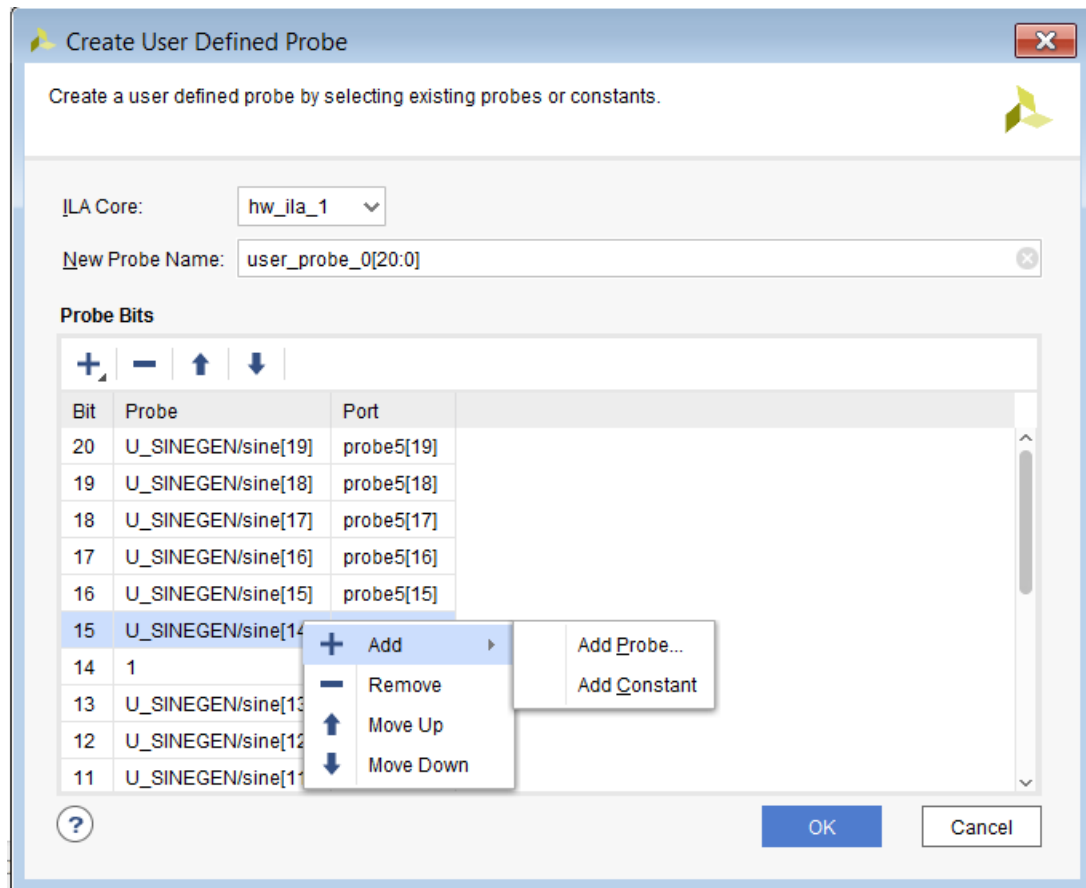
これで、[Add Probes] ダイアログ ボックスが開き、既存プローブの特定ビットを選択できます。

図 111: [Add Probes] ダイアログ ボックス



また、[Create User Defined] ダイアログ ボックスでビットを追加したり削除することもできます。次に示すように、特定ビットを上下移動させることもできます。

図 112: [Create User Defined Probe] ダイアログ ボックスでのビットの編集



Tcl フロー

ユーザー定義のデバッグ プローブを作成するには、`create_hw_probe` Tcl コマンドを使用します。

```
create_hw_probe [-verbose] [-map <arg>] <name> <core>
```

説明:

- **name:** `hw_probe` の名前です。同じデバイスの `hw_probes` には固有の名前を指定する必要があります。バス プローブは、`myNewProbe[31:0]` のように範囲で指定します。
- **core:** プロブを関連付ける `hw_ila` を指定します。
- **-map:** ユーザー定義プロブの基になるビットを宣言します。

次に、ユーザー定義デバッグ プロブの作成例を示します。

```
# Given a 512-bit counter "counterA[511:0]": Connects [255:223] to
#   ILA probe port 0 [31:0]
# Create a user-defined probe called foobar pointing at the
#   ILA buffer specified range [255:223]
create_hw_probe -map {probe0[31:0]} {foobar [255:223]} [get_hw_ilas
hw_ila_1]
# Constant only probe. NO triggering allowed on constant ONLY probes.
create_hw_probe -map {0} {my_constant_gnd[0:0]} [get_hw_ilas hw_ila_1]
```

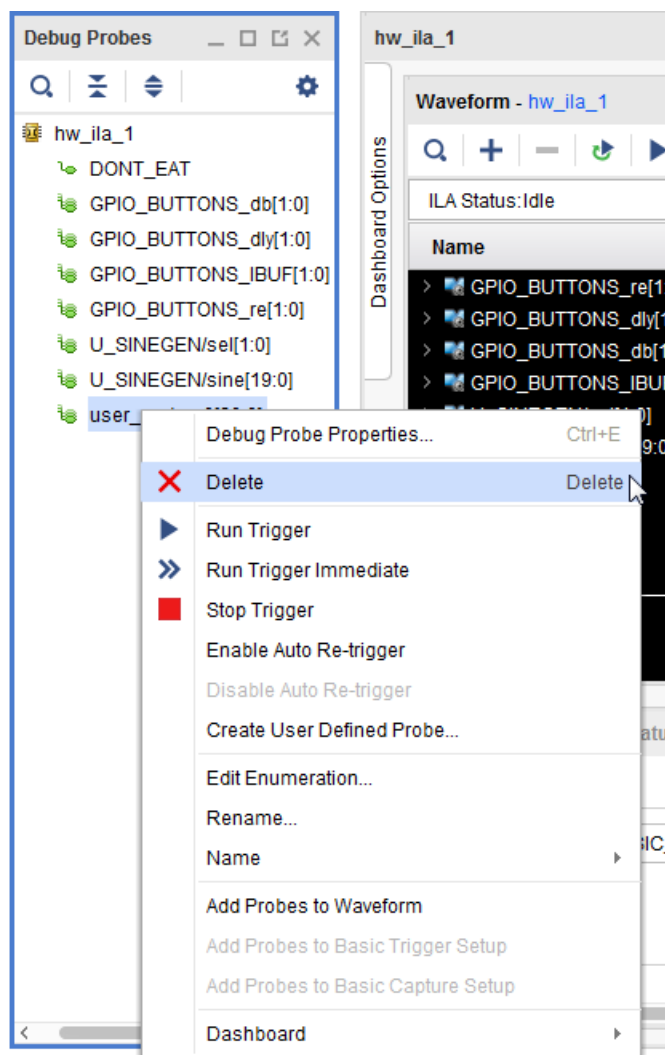
```
# Create a user-defined probe as a mix of constants and ILA probe ports
create_hw_probe -map {0000 probe0[31:0] 1010} {my-mixed-probe[47:8]}
[get_hw_ilas
hw_ila_1]
# Creating scalar hw_probe called "foobar" from probe1:
create_hw_probe -map {probe1} foobar [get_hw_ilas hw_ila_1]
# Creating scalar hw_probe called "foobar" from bit 3 of probe0:
create_hw_probe -map {probe0[3]} foobar [get_hw_ilas hw_ila_1]
# Creating vector hw_probe called "foobar[0:0]" from probe1:
create_hw_probe -map {probe1} foobar[0:0] [get_hw_ilas hw_ila_1]
# Creating vector probe called "foobar[3:0]" from probe0:
create_hw_probe -map {probe0} foobar[3:0] [get_hw_ilas hw_ila_1]
# Creating vector probe called "foobar[3:2]" from probe0[1:0]:
create_hw_probe -map {probe0[1:0]} foobar[3:2] [get_hw_ilas hw_ila_1]
```

ユーザー定義のデバッグ プローブの削除

GUI フロー

Vivado IDE ハードウェア マネージャーでユーザー 定義のプローブを削除するには、[Window]→[Debug Probe] をクリックします。これで、ハードウェア マネージャーのダッシュボードの横に [Debug Probes] ウィンドウが開きます。このウィンドウで該当プローブを右クリックし、次のように [Delete] をクリックします。

図 113: デバッグ プローブの削除



Tcl フロー

ユーザー定義デバッグ プローブを削除するには、`delete_hw_probe` Tcl コマンドを使用します。

たとえば、先ほどの例で `foobar` コマンドを使用して作成した `create_hw_probe` というプローブを削除するには、次のコマンドを使用します。

```
delete_hw_probe [get_hw_probes foobar -of_objects [get_hw_ilas -of_objects
[get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"i_fast_ila"}]]
```

ユーザー定義のデバッグ プローブの保持

プロジェクト フローでハードウェア マネージャーを使用して作成されたユーザー定義プローブは、ツールにより自動的に保持されます。次回プロジェクトを開いて Vivado ハードウェア マネージャーを使用してハードウェア ターゲットに接続すると、ユーザー定義プローブが復元されます。これらのユーザー定義デバッグ プローブが BASIC トリガー モードで使用されている場合や [Waveform] ウィンドウに追加されている場合は、プロジェクトを開いてハードウェア マネージャーを使用してターゲットに接続したときに、前回プロジェクトを閉じたときと同じようにすべてのウィンドウでプローブが設定されます。

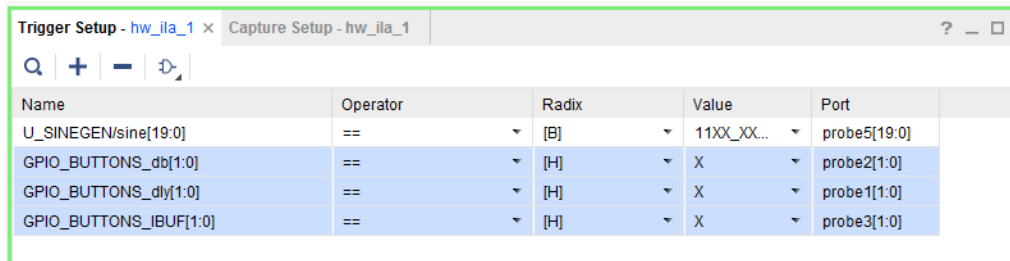
ユーザー定義プローブの使用

ハードウェア マネージャーで作成されたユーザー定義デバッグ プローブは、BASIC トリガー、ADVANCED トリガー、および [Waveform] ウィンドウで使用できます。ただし、定数値を含むユーザー定義デバッグ プローブは、[Waveform] ウィンドウでしか使用できません。

BASIC トリガー モードの使用

BASIC トリガー モードは、トリガー条件 (デバッグ プローブ コンパレータのグローバル ブール代数式) を記述するために使用されます。トリガー モードを BASIC_ONLY または BASIC_OR_TRIG_IN に設定するとイネーブルになります。このトリガー条件とデバッグ プローブ比較値を作成するには、[Basic Trigger Setup] ウィンドウを使用します。

図 114: ILA の [Basic Trigger Setup] ウィンドウ



Name	Operator	Radix	Value	Port
U_SINEGEN/sine[19:0]	==	[B]	11XX_XX...	probe5[19:0]
GPIO_BUTTONS_db[1:0]	==	[H]	X	probe2[1:0]
GPIO_BUTTONS_dly[1:0]	==	[H]	X	probe1[1:0]
GPIO_BUTTONS_IBUF[1:0]	==	[H]	X	probe3[1:0]

set_property Tcl コマンドを使用しても、ILA コアのトリガー モードを変更できます。たとえば、ILA コア hw_ila_1 のトリガー モードを BASIC_ONLY に変更するには、次のコマンドを使用します。

```
set_property CONTROL.TRIGGER_MODE BASIC_ONLY [get_hw_ilas hw_ila_1]
```

[Basic Trigger Setup] ウィンドウへのプローブの追加

BASIC トリガー モードを使用するには、まずトリガー条件にどの ILA デバッグ プローブを含めるかを決定します。これには、[Debug Probes] ウィンドウで ILA デバッグ プローブを選択し、右クリックして [Add Probes to Basic Trigger Setup] をクリックするか、プローブを [Basic Trigger Setup] ウィンドウにドラッグ アンド ドロップします。

注記: 最初のプローブは [Basic Trigger Setup] ウィンドウのどこにでもドラッグ アンド ドロップできますが、2 つ目からは最初のプローブの上にドロップする必要があります。新しいプローブは、常に前に追加されたプローブの上に追加されます。同じようにドラッグ アンド ドロップして、表内のプローブを並べ替えることもできます。



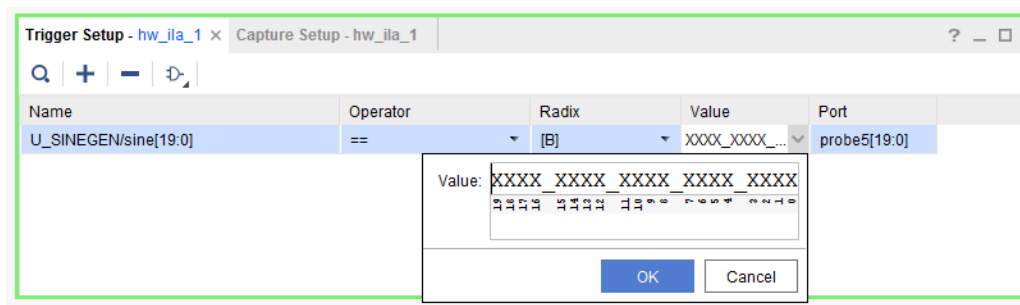
重要: トリガー条件には、[Basic Trigger Setup] ウィンドウ内のプローブのみを含めることができます。このウィンドウに含まれないプローブはドントケア値に設定され、トリガー条件には使用されません。

[Basic Trigger Setup] ウィンドウからプローブを削除するには、そのプローブを選択して [Delete] キーを押すか、右クリックして [Remove] をクリックします。

BASIC トリガー比較値の設定

ILA コアのプロープ入力における等価条件または不等価条件の検出には、ILA デバッグ プローブ トリガー コンパレータが使用されます。トリガー条件は、各 ILA プローブ トリガー コンパレータの結果をブール AND、OR、NAND、または NOR で計算した結果になります。ILA プローブの比較値を指定するには、[Basic Trigger Setup] ウィンドウの ILA デバッグ プローブの [Value] セルをクリックし、[Value] ダイアログ ボックスを開きます。

図 115: ILA プローブの [Compare Value] ダイアログ ボックス



ヒント: 基数を変更する前に、値が新しい基数に適用される文字列に設定されていることを確認してください。

ILA プローブの比較値の設定

[Basic Trigger Setup] ウィンドウには、プローブに対応している各行に 3 つのセルがあり、設定できます。

1. [Operator]: 比較演算子を指定します。有効な値は、次のとおりです。

- == (等価)
- != (不等価)
- < (小なり)
- <= (以下)
- > (大なり)
- >= (以上)

2. [Radix]: 値の基数を指定します。有効な値は、次のとおりです。

- [B] (2 進数)
- [H] (16 進数)
- [O] (8 進数)
- [U] (符号なし 10 進数)
- [S] (符号付き 10 進数)

3. [Value]: ILA デバッグ コアのプロープ入力に接続されているデザインのネット上の実際の値と、[Operator] で指定した演算子を使用して比較する値を指定します。[Radix] の設定によって、次の値を指定できます。
 - [Binary] (2 進数)
 - 0: 論理 0
 - 1: 論理 1
 - X: ドントケア
 - R: 立ち上がり遷移
 - F: 立ち下がり遷移
 - B: 立ち上がり遷移または立ち下がり遷移のいずれか
 - N: 遷移なし (現在のサンプル値は前の値と同じ)
 - [Hexadecimal] (16 進数)
 - X: 値の文字に対応するすべてのビットがドントケア値
 - 0-9: 0 ~ 9 の値
 - A-F: 10 ~ 15 の値
 - [Octal] (8 進数)
 - X: 値の文字に対応するすべてのビットがドントケア値
 - 0-7: 0 ~ 7 の値
 - [Unsigned Decimal] (符号なし 10 進数)
 - 正の整数値
 - [Signed Decimal] (符号付き 10 進数)
 - 整数値

BASIC トリガー条件の設定

[Basic Trigger Setup] ウィンドウの左側のツールバー ボタンの論理ゲートのアイコンをクリックすると、トリガー条件を設定できます。set_property Tcl コマンドを使用しても、ILA コアのトリガー条件を変更できます。

```
set_property CONTROL.TRIGGER_CONDITION AND [get_hw_ilas hw_ila_1]
```

次の表に、これら 4 つの値の意味を示します。

図 116: BASIC トリガー条件の設定

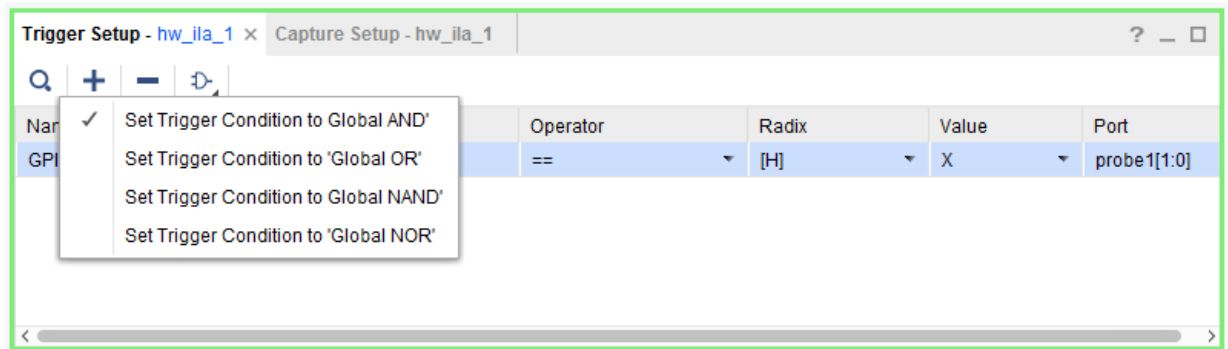


表 7: BASIC トリガー条件の設定の説明

GUI のトリガー条件設定	CONTROL.TRIGGER_CONDITION プロパティ値	トリガー条件出力
Global AND	AND	すべてのブローブ コンパレータが真の場合にトリガー条件が真となり、それ以外の場合は偽となります。
Global OR	OR	少なくとも 1 つのブローブ コンパレータが真の場合にトリガー条件が真となり、それ以外の場合は偽となります。
Global NAND	NAND	少なくとも 1 つのブローブ コンパレータが偽の場合にトリガー条件が真となり、それ以外の場合は偽となります。
Global NOR	NOR	すべてのブローブ コンパレータが偽の場合にトリガー条件が真となり、それ以外の場合は偽となります。



重要: ILA コアに 2 つ以上のデバッグ ブローブが含まれ、それらが ILA コアの物理的なブローブ ポート 1 つを共有するために連結されている場合、サポートされるトリガー条件は Global AND (AND) および Global NAND (NAND) のみになります。Global OR (OR) および Global NOR (NOR) 関数は、ブローブ ポート コンパレータ ロジックの制限のため、サポートされません。Global OR (OR) および Global NOR (NOR) トリガー条件設定を使用するには、各ネットまたはバス ネットを ILA コアの個別のブローブ ポートに割り当てるようにしてください。

ADVANCED トリガー モードの使用

ILA コアは、コア生成時または挿入時に次のアドバンス トリガー機能を含めるようにコンフィギュレーションできます。

- 最大 16 ステートまでのステート マシンをトリガー。
- 各ステートには 1 ～ 3 方向条件分岐を含めることが可能。
- トリガー ステート マシン プログラムに最大 4 つのカウンターを使用し、複数のイベントを追跡。
- トリガー ステート マシン プログラムに最大 4 つのカウンターを使用し、特定の分岐がいつ実行されたかを示す。
- ステート マシンで goto、trigger、さまざまなカウンターおよびフラグ関連アクションを実行可能。

ハードウェア デバイスで実行されるデザインに含まれる ILA コアにアドバンス トリガー機能がある場合、ILA ダッシュボードの [ILA Properties] ウィンドウの [Trigger mode] を [ADVANCED_ONLY] または [ADVANCED_OR_TRIG_IN] に設定すると、ADVANCED トリガー モードをイネーブルにできます。

トリガー ステート マシン プログラム ファイルの指定

[Trigger mode] を [ADVANCED_ONLY] または [ADVANCED_OR_TRIG_IN] に設定すると、ILA ダッシュボードで次の 2 つが変更されます。

1. [Trigger State Machine] という新しいウィンドウが [ILA Properties] ウィンドウ内に表示されます。
2. [Basic Trigger Setup] ウィンドウの代わりに [Trigger State Machine] コード エディター ウィンドウが表示されます。

ILA トリガー ステート マシン プログラムを初めて指定する場合、[Trigger State Machine] コード エディター ウィンドウに次のように表示されます。

図 117: トリガー ステート マシン プログラム ファイルを作成または開く

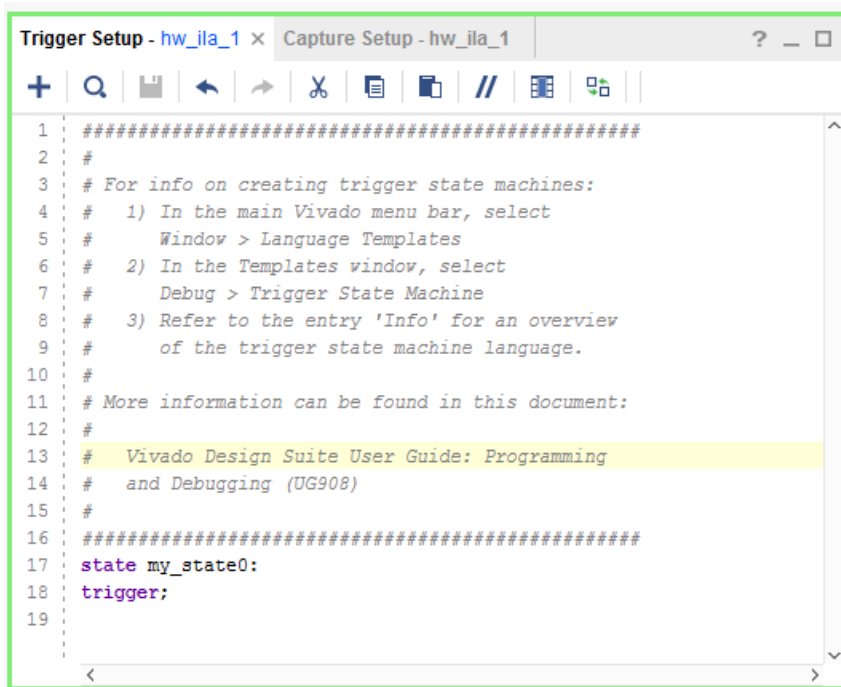


新しいトリガー ステート マシンを作成するには、[Create new trigger state machine] リンクをクリックします。既存のトリガー ステート マシンを開くには、[Open existing trigger state machine] リンクをクリックしてトリガー ステート マシン プログラム ファイル (.tsm) を開きます。既存のトリガー ステート マシン プログラム ファイルは、ILA ダッシュボードの [ILA Properties] ウィンドウの [Trigger state machine] テキスト フィールドまたは参照ボタンを使用しても開くことができます。

トリガー ステート マシン プログラムの編集

トリガー ステート マシン プログラム ファイルを作成したら、[Trigger State Machine] コード エディター ウィンドウに単純なトリガー ステート マシンがデフォルトで表示されます。

図 118: 単純なデフォルトのトリガー ステート マシン プログラム



```

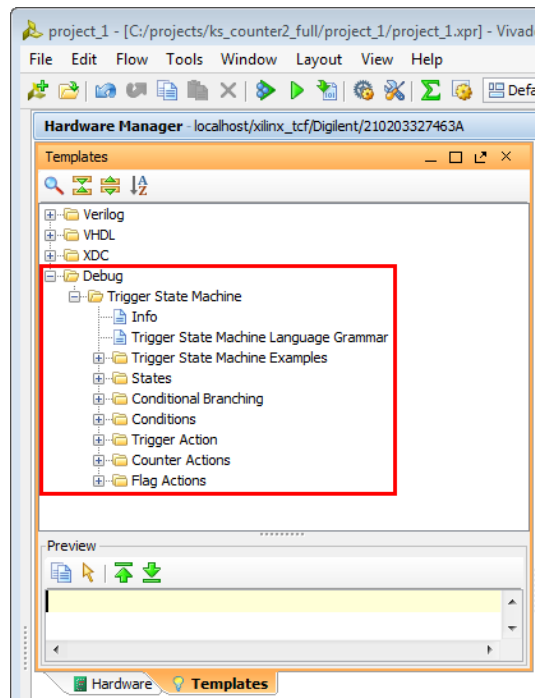
1 #####
2 #
3 # For info on creating trigger state machines:
4 # 1) In the main Vivado menu bar, select
5 #    Window > Language Templates
6 # 2) In the Templates window, select
7 #    Debug > Trigger State Machine
8 # 3) Refer to the entry 'Info' for an overview
9 #    of the trigger state machine language.
10 #
11 # More information can be found in this document:
12 #
13 # Vivado Design Suite User Guide: Programming
14 # and Debugging (UG908)
15 #
16 #####
17 state my_state0:
18 trigger;
19

```

単純なデフォルトのトリガー ステート マシン プログラムは、デバッグ プローブまたはトリガー設定に関係なく ILA コアのコンフィギュレーションを有効にするためのものです。これにより、トリガー ステート マシン プログラムを変更しなくても ILA コアに対して [Run Trigger] をクリックできます。

トリガー ステート マシン プログラムを変更して、高度なトリガー条件をインプリメントすることもできます。前の図に示す単純なステート マシンの冒頭のコメント部分には、Vivado IDE のビルトイン言語テンプレートを使用してトリガー ステート マシン プログラム (次の図) を構築する方法が説明されています。例も含めた ILA トリガー ステート マシン言語の説明は、「トリガー ステート マシンの言語記述」を参照してください。

図 119: トリガー ステート マシンの言語テンプレート



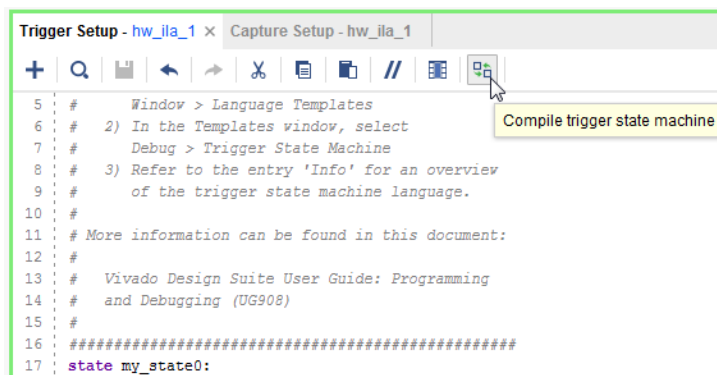
関連情報

[トリガー ステート マシンの言語記述](#)

トリガー ステート マシンのコンパイル

トリガー ステート マシンは、ILA をトリガーするたびにコンパイルされます。ILA をトリガーまたはトリガー待機状態にせずにトリガー ステート マシンをコンパイルするには、ILA ダッシュボードのツールバーにある [Compile trigger state machine] ボタンをクリックします。

図 120: トリガー待機状態にせずにトリガー ステート マシンをコンパイル



トリガー入力ポートおよび出力ポートのイネーブル

ILA コアは、専用トリガー入力ポート (TRIG_IN と TRIG_IN_ACK) および専用トリガー出力ポート (TRIG_OUT と TRIG_OUT_ACK) を含めるよう、コアの生成時にコンフィギュレーションできます。ILA コアでトリガー入力ポートがイネーブルになっている場合、次のトリガー モード設定を使用して TRIG_IN ポートにイベントをトリガーできます。

- BASIC_OR_TRIG_IN: ILA コアの TRIG_IN ピンと BASIC_ONLY トリガー モードの論理 OR の結果により ILA コアをトリガーします。
- ADVANCED_OR_TRIG_IN: ILA コアの TRIG_IN ピンと ADVANCED_ONLY トリガー モードの論理 OR の結果により ILA コアをトリガーします。
- TRIG_IN_ONLY: ILA コアの TRIG_IN ピンが Low から High に遷移したときに ILA コアをトリガーします。

ILA コアでトリガー出力ポートがイネーブルになっている場合、次の TRIG_OUT モードを使用して TRIG_OUT ポートへのトリガー イベントの伝搬を制御できます。

- DISABLED: TRIG_OUT ポートをディスエーブルにします。
- TRIGGER_ONLY: BASIC/ADVANCED トリガー条件の結果を TRIG_OUT ポートに伝搬します。
- TRIG_IN_ONLY: TRIG_IN ポートを TRIG_OUT ポートに伝搬します。
- TRIGGER_OR_TRIG_IN: BASIC/ADVANCED トリガー条件の論理 OR 結果と TRIG_IN ポートを TRIG_OUT ポートに伝搬します。

キャプチャ モード設定のコンフィギュレーション

ILA コアは、コアのステータスが Pre-Trigger、Waiting for Trigger、または Post-Trigger の場合にデータ サンプルをキャプチャできます (詳細は「トリガーおよびキャプチャ ステータスの表示」を参照)。各サンプルがキャプチャされる前に評価される条件は、[Capture mode] ドロップダウン リストから選択します。

- [ALWAYS]: キャプチャ条件に関係なく指定したクロック サイクル中のデータ サンプルを格納します。
- [BASIC]: キャプチャ条件が真の場合にのみ指定したクロック サイクル中のデータ サンプルを格納します。

set_property Tcd コマンドを使用しても、ILA コアのキャプチャ モードを変更できます。たとえば、ILA コア hw_ila_1 のキャプチャ モードを [BASIC] に変更するには、次のコマンドを使用します。

```
set_property CONTROL.CAPTURE_MODE BASIC [get_hw_ilas hw_ila_1]
```

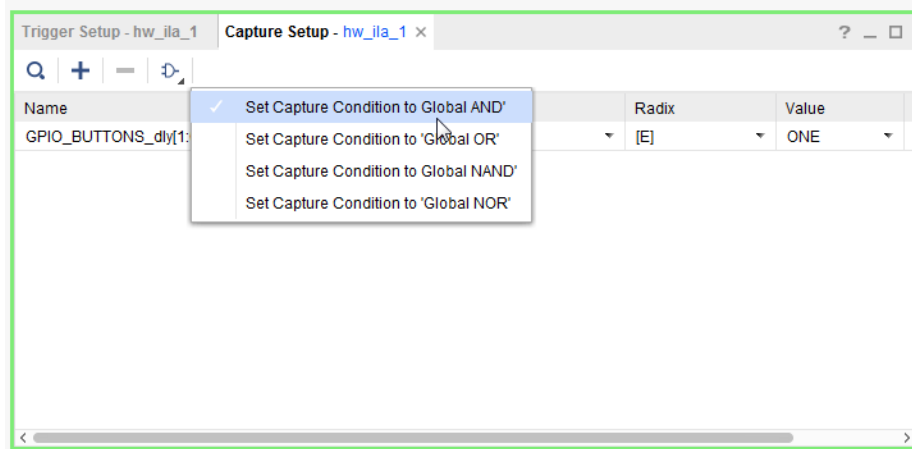
関連情報

[トリガーおよびキャプチャ ステータスの表示](#)

BASIC キャプチャ モードの使用

BASIC キャプチャ モードは、トリガー条件 (デバッグ プローブ コンパレータのグローバル ブール代数式) を記述するために使用されます。このキャプチャ条件とデバッグ プローブ 比較値を作成するには、[Basic Capture Setup] ウィンドウを使用します。

図 121: BASIC キャプチャ条件の設定



[Basic Capture Setup] ウィンドウの設定

[Basic Capture Setup] ウィンドウでデバッグ プローブと BASIC キャプチャ条件を設定するプロセスは、[Basic Trigger Setup] ウィンドウのデバッグ プローブを設定するプロセスとほぼ同じです。

- プローブを [Basic Capture Setup] ウィンドウに追加する方法は、「[Basic Trigger Setup] ウィンドウへのプローブの追加」を参照してください。
- [Basic Capture Setup] ウィンドウで各プローブの比較値を設定する方法は、「ILA プローブの比較値の設定」を参照してください。
- [Basic Capture Setup] ウィンドウで基本キャプチャ条件を設定する方法は、「BASIC トリガー条件の設定」を参照してください。主な違いは、キャプチャ条件を制御する ILA コアのプロパティが CONTROL.CAPTURE_CONDITION である点です。

関連情報

[\[Basic Trigger Setup\] ウィンドウへのプローブの追加](#)

[ILA プローブの比較値の設定](#)

[BASIC トリガー条件の設定](#)

キャプチャ ウィンドウの数の設定

ILA キャプチャ データ バッファは複数のキャプチャ ウィンドウに分割でき、各ウィンドウの深さは $1 \sim ((\text{バッファ サイズ}) / (\text{ウィンドウの数})) - 1$ の間の 2 のべき乗値です。たとえば、ILA データ バッファのサンプル深さが 1024 で、4 つのキャプチャ ウィンドウに分割する場合、各ウィンドウのサンプル深さは 256 までになります。各キャプチャ ウィンドウには、キャプチャ ウィンドウにデータが格納されるトリガー イベントに対応する独自のトリガー マークがあります。



ヒント: ILA データ キャプチャ バッファが完全にフルになる前に [Stop Trigger] をクリックすると、それまでにフルになったキャプチャ ウィンドウがアップロードされて表示されます。たとえば、ILA データ バッファが 4 つのウィンドウに分割されており、そのうち 3 つがフルになっている場合、[Stop Trigger] をクリックすると ILA コアが停止し、フルになっている 3 つのキャプチャ ウィンドウがアップロードされて表示されます。

次の表に、[Number of Capture Windows] を 1 より大きい値に、[Trigger Position] を 0 に設定した場合の Vivado ランタイム ソフトウェアとハードウェアの関係を示します。

表 8: [Number of Capture Windows] > 1 および [Trigger Position] = 0

ソフトウェア	ハードウェア
ユーザーが ILA コアでトリガーを実行	ウィンドウ 0: ILA がトリガー待機状態 ウィンドウ 0: ILA がトリガー ウィンドウ 0: ILA がキャプチャ ウィンドウ 0 を充填 ウィンドウ 1: ILA が再度トリガー待機状態 ウィンドウ 1: ILA がトリガー ウィンドウ 1: ILA がキャプチャ ウィンドウ 1 を充填 ... ウィンドウ n-1: ILA が再度トリガー待機状態 ウィンドウ n-1: ILA がトリガー ウィンドウ n-1: ILA がキャプチャ ウィンドウ 0 を充填 ILA キャプチャ バッファ全体がフルになる
データがアップロードされ表示される	ILA コアは、前のウィンドウの最後にキャプチャされたサンプル直後のクロック サイクルでトリガー準備が完了するように、再度トリガー待機状態になる。

次の表に、[Number of Capture Windows] を 1 より大きい値に、[Trigger Position] を 0 より大きい値に設定した場合の Vivado ランタイム ソフトウェアとハードウェアの関係を示します。

表 9: [Number of Capture Windows] > 1 および [Trigger Position] > 0

ソフトウェア	ハードウェア
ユーザーが ILA コアでトリガーを実行	ウィンドウ 0: ILA がトリガー待機状態 ウィンドウ 0: ILA がキャプチャ バッファをトリガー位置まで充填 ウィンドウ 0: ILA がトリガー ウィンドウ 0: ILA がキャプチャ ウィンドウ 0 の残りを充填 ウィンドウ 1: ILA が再度トリガー待機状態 ウィンドウ 1: ILA がキャプチャ バッファをトリガー位置まで充填 ウィンドウ 1: ILA がトリガー ウィンドウ 1: ILA がキャプチャ バッファを充填 ウィンドウ 1: ILA がウィンドウ 1 を充填 ... ウィンドウ n-1: ILA が再度トリガー待機状態 ウィンドウ n-1: ILA がキャプチャ バッファをトリガー位置まで充填 ウィンドウ n-1: ILA がトリガー ウィンドウ n-1: ILA がキャプチャ バッファを充填 ウィンドウ n-1: ILA がウィンドウ n を充填 ILA キャプチャ バッファ全体がフルになる
データがアップロードされ表示される	ILA がキャプチャ データをトリガー位置まで充填する必要があるため、2 つのウィンドウ間でトリガーがミスとなる可能性あり。

キャプチャ ウィンドウでのトリガー位置の設定

[Capture Mode Settings] ウィンドウの [Trigger position] または [ILA Core Properties] ウィンドウの [Trigger Position] プロパティを使用して、キャプチャされたデータ ウィンドウのトリガー マーカーの位置を設定します。トリガー位置は、キャプチャ データ ウィンドウの任意のサンプル番号に設定できます。たとえば、サンプル数が 1024 のキャプチャ データ ウィンドウの場合は次のようになります。

- サンプル番号 0 は、キャプチャ データ ウィンドウの最初 (一番左) のサンプルに対応します。

- サンプル番号 1023 は、キャプチャ データ ウィンドウの最後 (一番右) のサンプルに対応します。
- サンプル番号 511 および 512 は、キャプチャ データ ウィンドウの中央のサンプルに対応します。

set_property Tcl コマンドを使用しても ILA コアのトリガー位置を変更できます。

```
set_property CONTROL.TRIGGER_POSITION 512 [get_hw_ilas hw_ila_1]
```

キャプチャ ウィンドウでのデータ深さの設定

[Capture Mode Settings] ウィンドウの [Data Depth] または [ILA Core Properties] ウィンドウの [Capture data depth] プロパティを使用して、ILA コアのキャプチャ データ ウィンドウのデータ深さを設定します。データ深さは、1 から ILA コアの最大データ深さの間の 2 のべき乗値に設定できます (コアの生成時または挿入時に設定)。

注記: ネットリスト挿入プローブ フローでデザインに追加した ILA コアのキャプチャ バッファの最大データ深さを設定する方法は、「デバッグ コアのプロパティの変更」を参照してください。

set_property Tcl コマンドを使用しても ILA コアのデータ深さを変更できます。

```
set_property CONTROL.DATA_DEPTH 512 [get_hw_ilas hw_ila_1]
```

関連情報

[デバッグ コアのプロパティの変更](#)

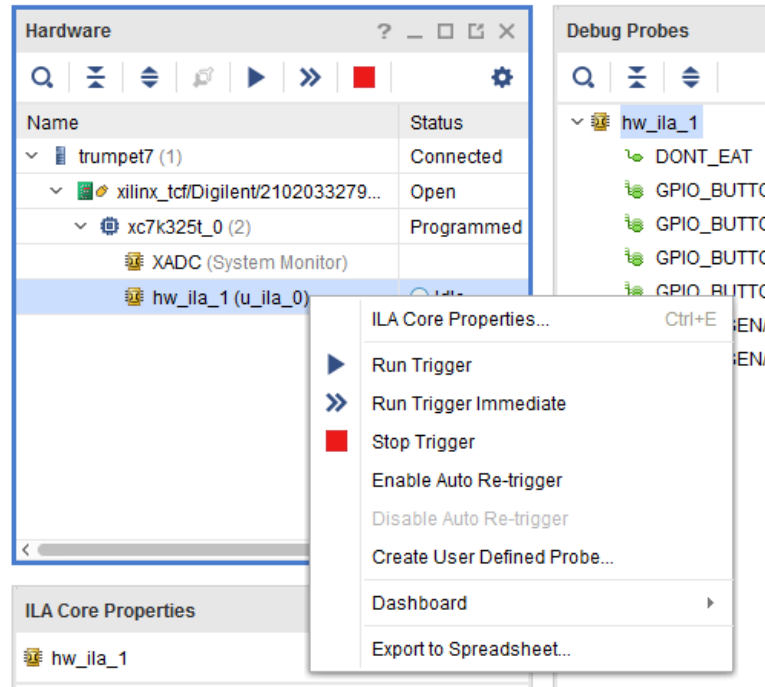
トリガーの実行

ILA コアにトリガーを供給する方法には、次の 2 つのモードがあります。

- [Run Trigger]: ILA ダッシュボードまたは [Hardware] ウィンドウで ILA コアを選択し、ツールバーの [Run Trigger] をクリックすると、ILA コアの BASIC または ADVANCED トリガー設定で定義されたトリガー イベントを検出できるようになります。
- [Run Trigger Immediate]: ILA ダッシュボードまたは [Hardware] ウィンドウで ILA コアを選択し、ツールバーの [Run Trigger Immediate] をクリックすると、ILA コアのトリガー設定に関係なく ILA コアが即座にトリガーされます。このコマンドは、ILA コアのプローブ入力のすべてのアクティビティをキャプチャして、デザインの動作を検出するのに便利です。

ILA コアを右クリックし、[Run Trigger] または [Run Trigger Immediate] をクリックしても、同じ操作を実行できます。

図 122: ILA コアのトリガー コマンド



ヒント: [Hardware] ウィンドウで ILA コアを選択し、[Run Trigger]、[Run Trigger Immediate]、または [Stop Trigger] ボタンを使用すると、複数の ILA コアのトリガーを実行または停止できます。また、[Hardware] ウィンドウでデバイスを選択し、ツールバーの適切なボタンをクリックすると、そのデバイスのすべての ILA コアのトリガーを実行または停止できます。

トリガーの停止

ILA コアのトリガーを停止するには、ILA コアを選択し、ILA ダッシュボードまたは [Hardware] ウィンドウのツールバーの [Stop Trigger] をクリックします。ILA コアを右クリックし [Stop Trigger] をクリックしても、同じ操作を実行できます (「トリガーの実行」を参照)。

関連情報

[トリガーの実行](#)

自動再トリガーの使用

ILA コアの右クリック メニューの [Enable Auto Re-Trigger] または ILA ダッシュボードのツールバーの対応するボタンをクリックすると、トリガー、アップロード、表示処理が正しく完了した後に、Vivado IDE で ILA コアが自動的にトリガー待機状態になります。ILA コアに対応する波形ビューアーに表示されたキャプチャ データは、トリガー イベントごとに上書きされます。[Auto Re-Trigger] オプションは、[Run Trigger] および [Run Trigger Immediate] コマンドと共に使用できます。動作中のトリガーを停止するには、[Stop Trigger] をクリックします。

次の表に、[Auto Re-Trigger] オプションを使用した際の Vivado IDE ランタイム ソフトウェアとハードウェアの関係を示します。

表 10: [Auto Re-Trigger] オプション

Software	[Hardware]
ILA コアで [Auto Re-Trigger] オプションをクリック	ILA が待機状態 ILA がトリガー ILA がキャプチャ バッファを充填 ILA がフルになる
データがアップロードされ表示される	ILA が再び待機状態 ILA がトリガー ILA がキャプチャ バッファを充填 ILA がフルになる ILA の「フル」イベントから ILA データの表示までに多くのサイクルがミスとなる。



重要: ILA データがフルになってから、データがアップロードされて GUI に表示されるまでに遅延があるので、これらのイベント間で ILA がトリガーできたかもしれないのに、ミスとなる可能性が高くなります。

トリガーおよびキャプチャ ステータスの表示

ILA デバッグ コアのトリガーおよびキャプチャ ステータスは、Vivado IDE では次の 2 箇所に表示されます。

- [Hardware] ウィンドウの ILA デバッグ コアの行の [Status] 列。
- ILA ダッシュボードの [Trigger Capture Status] ウィンドウ。

[Hardware] ウィンドウの [Status] 列には、各 ILA コアのステータスが示されます。

表 11: ILA コアのステータス

ILA コアのステータス	説明
Idle	ILA コアはアイドル状態であり、トリガーの実行を待機している状態です。トリガー位置が 0 の場合、トリガーが実行されると ILA コアが Waiting for Trigger ステータスになります。それ以外の場合は、Pre-Trigger ステータスになります。
Pre-Trigger	ILA コアがトリガー前のデータをデータ キャプチャ ウィンドウに取り込み中です。トリガー前のデータがキャプチャされると、ILA コアは Waiting for Trigger ステータスになります。
Waiting for Trigger	ILA コアのトリガー待機状態になり、BASIC または ADVANCED トリガー設定で定義されたトリガー イベントが発生するのを待機中です。トリガーが発生すると、トリガー位置がキャプチャ ウィンドウの最後のデータ サンプルに設定されている場合、ILA コアは Full ステータスになります。それ以外の場合は、Post-Trigger ステータスになります。
Post-Trigger	ILA コアがトリガー後のデータをデータ キャプチャ ウィンドウに取り込み中です。トリガー後のデータがキャプチャされると、ILA コアは Full ステータスになります。
Full	ILA コア キャプチャ ウィンドウがフルであり、ホストに表示するようアップロード中です。データがアップロードされて表示されると、ILA コアは Idle ステータスになります。

ILA ダッシュボードの [Trigger Capture Status] ウィンドウの内容は、ILA コアの [Trigger mode] の設定によって異なります。

部分的バッファー キャプチャ

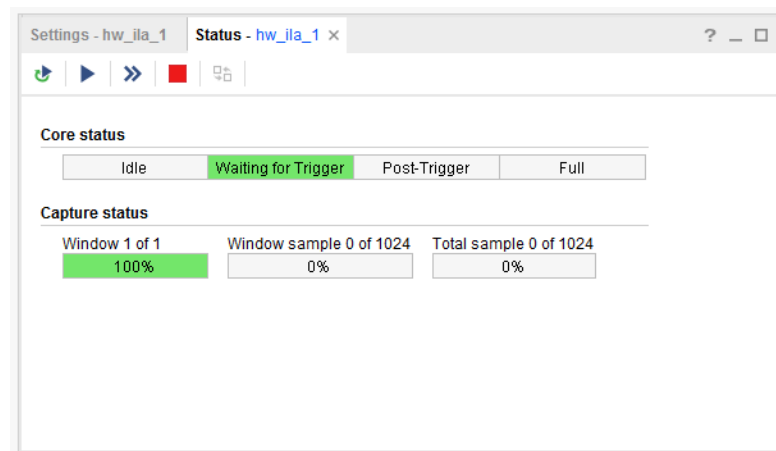
ILA データ キャプチャ バッファが完全にフルになる前に [Stop Trigger] をクリックすると、それまでに満たされたキャプチャ ウィンドウがすべてアップロードされて表示されます。たとえば、ILA データ バッファが 4 つのウィンドウに分割されており、そのうち 3 つが満たされている場合、[Stop Trigger] をクリックすると ILA コアが停止し、満たされている 3 つのキャプチャ ウィンドウがアップロードされて表示されます。また、[Stop Trigger] をクリックすると、ILA コアが停止して、一部満たされたキャプチャ ウィンドウが開きます (そのキャプチャ ウィンドウでトリガー イベントが発生している場合)。

BASIC トリガー モードのトリガーおよびキャプチャ ステータス

[Trigger mode] を BASIC に設定すると、[Trigger Capture Status] ウィンドウには次の 2 つのステータス インジケータが表示されます。

- [Core status]: ILA コアのトリガー/キャプチャ エンジンのステータスを示します (ステータス インジケータの詳細は「トリガーおよびキャプチャ ステータスの表示」を参照)。
- [Capture status]: 現在のキャプチャ ウィンドウ、現在のキャプチャ ウィンドウでキャプチャされた現時点でのサンプル数、ILA コアでキャプチャされたサンプルの総数を示します。これらの数値は、ILA コアのステータスが Idle になると 0 にリセットされます。

図 123: BASIC トリガー モードの [Trigger Capture Status] ウィンドウ



関連情報

[トリガーおよびキャプチャ ステータスの表示](#)

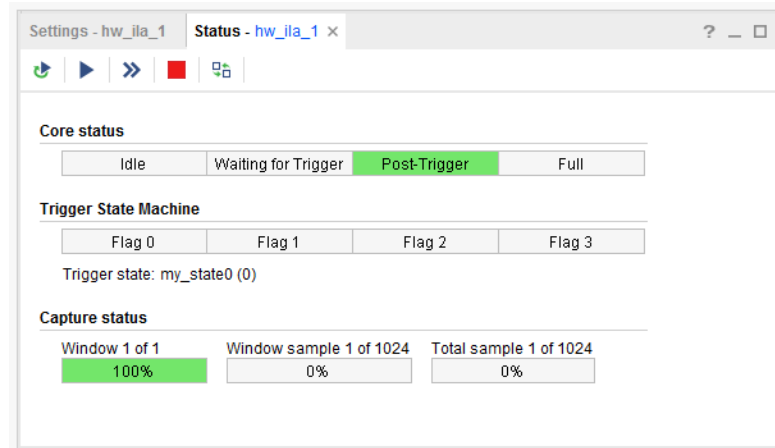
ADVANCED トリガー モードのトリガーおよびキャプチャ ステータス

[Trigger mode] を ADVANCED に設定すると、[Trigger Capture Status] ウィンドウには次の 4 つのステータス インジケータが表示されます。

- [Core status]: ILA コアのトリガー/キャプチャ エンジンのステータスを示します (ステータス インジケータの詳細は「トリガーおよびキャプチャ ステータスの表示」を参照)。
- [Trigger State Machine Flags]: 4 つのトリガー ステート マシン フラグの現在のステートを示します。
- [Trigger State]: コアのステータスが Waiting for Trigger の場合、トリガー ステート マシンの現在のステートを示します。

- [Capture status]: 現在のキャプチャ ウィンドウ、現在のキャプチャ ウィンドウでキャプチャされた現時点でのサンプル数、ILA コアでキャプチャされたサンプルの総数を示します。これらの数値は、ILA コアのステータスが Idle になると 0 にリセットされます。

図 124: ADVANCED トリガー モードの [Trigger Capture Status] ウィンドウ



関連情報

[トリガーおよびキャプチャ ステータスの表示](#)

ILA プローブ情報の記述

[Debug Probes] ウィンドウの [ILA Cores] ビューには、ILA コアを使用してプローブされるデザインのネットに関する情報が表示されます。この ILA プローブ情報はデザインから抽出され、通常 `.ltx` という拡張子のデータ ファイルに保存されます。

ILA プローブ ファイルは、ビットストリーム生成中に自動的に作成されますが、`write_debug_probes Tcl` コマンドを使用してデバッグ プローブ情報をファイルに書き出すこともできます。

1. プロジェクト モードの場合は、インプリメント済みデザインを開きます。非プロジェクト モードの場合は、インプリメント済みデザイン チェックポイントを開きます。
2. `write_debug_probes filename.ltx Tcl` コマンドを実行します。

ILA プローブ情報の読み出し

ILA プローブ ファイルの名前が `debug_nets.ltx` で、デバイスに関連付けられているビットストリーム プログラム ファイル (`.bit`) と同じディレクトリにある場合、自動的に FPGA ハードウェア デバイスに関連付けられます。

プローブ ファイルの場所を指定するには、次の手順に従います。

1. [Hardware] ウィンドウで FPGA を選択します。
2. [Hardware Device Properties] ウィンドウでプローブ ファイルのディレクトリを設定します。

3. [Apply] をクリックして変更を適用します。

プローブ ファイルの場所は、次の `set_property Tcl` コマンドを使用しても設定できます。

```
set_property PROBES.FILE {C:/myprobes.ltx} [lindex [get_hw_devices] 0]
```

ILA コアからのデータを波形ビューアーで表示

ILA コアでキャプチャされたデータが Vivado IDE にアップロードされると、波形ビューアーに表示されます。ILA コアでキャプチャされたデータの表示に波形ビューアーを使用する場合の詳細は、「ILA プローブ データの表示」を参照してください。

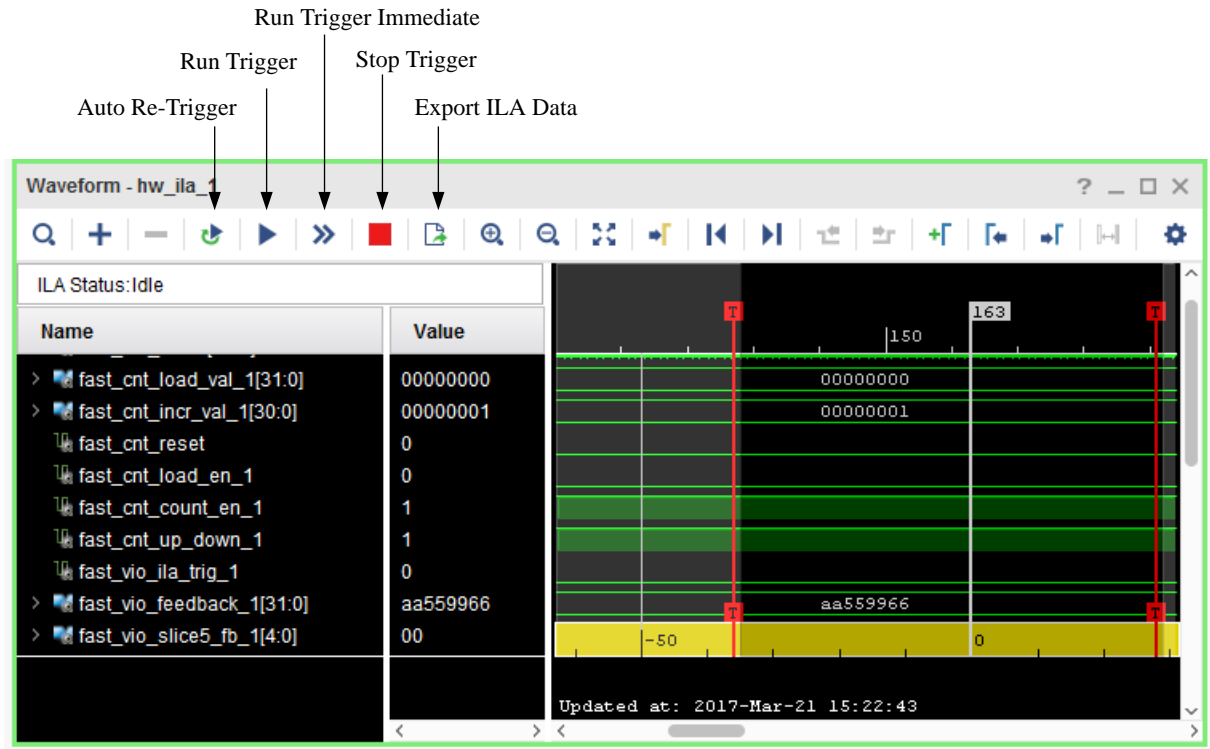
関連情報

[波形ビューアーを使用した ILA プローブ データの表示](#)

波形 ILA トリガーおよびエクスポート機能の使用

ILA の保護、トリガーの実行および停止、ILA データのエクスポートをするには、次のように、波形ウィンドウのアイコンを使用できます。

図 125: 波形 ILA トリガーおよびエクスポート機能



X16755-032717

[Enable Auto Re-Trigger]: [Waveform] ウィンドウの [Enable Auto Re-Trigger] ツールバー ボタンをクリックすると、Vivado IDE でトリガー、アップロード、表示操作が正しく完了した後に、[Waveform] ウィンドウに関連付けられている ILA コアが自動的にトリガー待機状態になります。

[Waveform] ウィンドウに表示された ILA コアに対応するキャプチャ データは、トリガー イベントごとに上書きされます。[Auto Re-Trigger] オプションは、[Run Trigger] および [Run Trigger Immediate] コマンドと共に使用できます。動作中のトリガーを停止するには、[Stop Trigger] をクリックします。

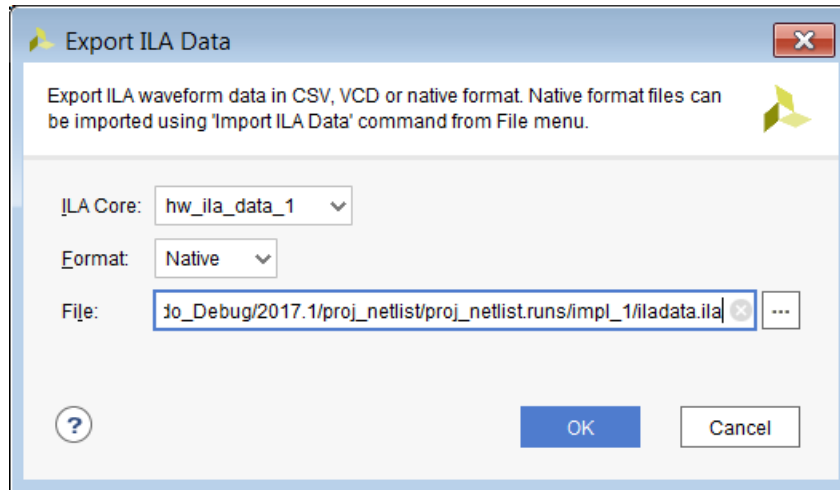
[Run Trigger]: [Waveform] ウィンドウに関連付けられている ILA コアをトリガー待機状態にし、ILA コアの BASIC または ADVANCED トリガー設定で定義されたトリガー イベントが検出されるようにします。

[Run Trigger Immediate]: [Waveform] ウィンドウに関連付けられている ILA コアをトリガー待機状態にし、ILA コアの BASIC または ADVANCED トリガー設定で定義されたトリガー イベントが検出されるようにします。このコマンドは、ILA コアのプローブ入力のすべてのアクティビティをキャプチャして、デザインの動作を検出するのに便利です。

[Stop Trigger]: [Waveform] ウィンドウに関連付けられている ILA の ILA コア トリガーを停止します。

[Export ILA Data]: ILA コアからのデータをキャプチャし、ファイルに保存します。データは、ネイティブ、.csv、または .vcd フォーマットで保存できます。[Waveform] ウィンドウのツールバー ボタンをクリックすると、次のダイアログ ボックスが表示されます。

図 126: [Export ILA Data] ダイアログ ボックス



[ILA Core] でデータをエクスポートする ILA デバッグ コアの名前を選択し、[Format] でフォーマットを [Native]、[CSV]、[VCD] のいずれかに設定します。

- [Native]: ネイティブ ILA フォーマットで ILA データをエクスポートするよう、`write_hw_ila_data` コマンドを設定します。

この ILA ファイルは、前にキャプチャされた ILA データを確認できるように、Vivado IDE にインポートし直すことができます。Vivado IDE に ILA データをインポートし直すには、`read_hw_ila_data` Tcl コマンドを使用できます。

```
read_hw_ila_data {./iladata.ila}
display_hw_ila_data
```

- [CSV]: `write_hw_ila_data` コマンドで ILA データをスプレッドシートやサードパーティ アプリケーションにインポート可能な `.csv` ファイル フォーマットでエクスポートします。
- [VCD]: `write_hw_ila_data` コマンドで ILA データをサードパーティ アプリケーションまたはビューアーにインポート可能な `.vcd` ファイル フォーマットでエクスポートします。

★ **重要:** ILA データは CSV、VCD、またはネイティブ ILA フォーマットにエクスポートできますが、Vivado にインポートできるのはネイティブ ILA フォーマットのみです。Vivado にインポートした ILA データは、以前にキャプチャされたデータをオフラインで表示するためにのみサポートされます。プローブ信号をトリガーするためなどほかの目的で使用することはできません。

ILA コアでキャプチャされたデータの保存および復元

ILA コアから直接アップロードされたキャプチャ データを表示するだけでなく、ファイルに保存して、ファイルからデータを読み込むこともできます。

ILA コアでキャプチャされたデータのファイルへの保存

ILA コアでキャプチャされたデータをアップロードしてファイルに保存するには、次の Tcl コマンドを使用します。

```
write_hw_ila_data my_hw_ila_data_file.ila [upload_hw_ila_data hw_ila_1]
```

この Tcl コマンドにより、ILA コアでキャプチャされたデータがアップロードされ、`my_hw_ila_data_file.ila` というアーカイブ ファイルに保存されます。このアーカイブ ファイルには、波形データベース ファイル、波形コンフィギュレーション ファイル、波形 CSV (Comma Separated Value) ファイル、およびデバッグ プロブ ファイルが含まれます。



ヒント: CSV ファイルを生成するには、`-csv_file` オプションを使用します。これにより、`write_hw_ila_data` ILA データがデフォルトのバイナリ ILA ファイル フォーマットではなく、スプレッドシートやサードパーティ アプリケーションにインポート可能な CSV ファイルの形式でエクスポートされます。



ヒント: VCD (Value Change Dump) ファイルを生成するには、`-vcd_file` オプションを使用します。これにより、`write_hw_ila_data` ILA データがデフォルトのバイナリ ILA ファイル フォーマットではなく、サードパーティ アプリケーションまたはビューアーにインポート可能な VCD ファイルの形式でエクスポートされます。

プロブ データの基数

各プロブには、`DISPLAY_RADIX` プロパティが関連付けられています。デフォルトでは、このプロパティは複数ビット プロブに対しては HEX に、1 ビット プロブに対しては BINARY に設定されます。`.csv` ファイルにエクスポートされたプロブ データは、プロブの基数を使用します。

デザインに含まれるすべての ILA のすべてのプロブに対して `DISPLAY_RADIX` プロパティを変更するには、Vivado ハードウェア マネージャーの Tcl コンソールで次のコマンドを使用します。

```
foreach probe [get_hw_probes -of [get_hw_ilas]] {
    set_property DISPLAY_RADIX binary $probe
    set_property DISPLAY_AS_ENUM false $probe
}
```

注記: この例では、すべての ILA のすべてのプロブの基数を BINARY に変更しています。基数を HEX に変更するには、次のスクリプトを使用します。

```
foreach probe [get_hw_probes -of [get_hw_ilas]] {
    set_property DISPLAY_RADIX hex $probe
    set_property DISPLAY_AS_ENUM false $probe
}
```

1 つのプロブに関連付けられているデータ サンプルのリスト

個別のプロブに関連付けられているデータ サンプルをリストするには、`list_hw_samples` Tcl コマンドを使用します。

次に、`fast_cnt_incr_val_1` という ILA の `i_fast_ila` というプロブに関連付けられているデータ サンプルをリストする例を示します。

```
list_hw_samples [get_hw_probes fast_cnt_incr_val_1 -of_objects [get_hw_ilas
-of_objects [get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"i_fast_ila"}]]
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001...
```

ILA コアでキャプチャされたデータのファイルからの読み込み

ILA コアでキャプチャされたデータをファイルから読み込むには、次の Tcl コマンドを使用します。

```
display_hw_ila_data [read_hw_ila_data my_hw_ila_data_file.ila]
```

この Tcl コマンドにより、保存されている ILA コア キャプチャ データが読み込まれ、[Waveform] ウィンドウに表示されます。

注記: ILA データの [Waveform] ウィンドウの波形コンフィギュレーション設定 (仕切り、マーカー、色、プローブの基数など) も、ILA キャプチャ データ アーカイブ ファイルに保存されます。保存されている ILA データを表示すると、保存されている波形コンフィギュレーション設定が使用されます。



重要: ILA キャプチャ データから作成した波形を開くのに `open_wave_config` コマンドを使用しないでください。このコマンドはシミュレーションのみのコマンドであり、ILA キャプチャ データ波形では正しく機能しません。

プローブ値の列挙名

この機能を使用すると、値を比較するトリガー/キャプチャの設定時および [Waveform] ウィンドウで、プローブ値をシンボル名で参照できるようになります。

次のような使用方法が一般的です。

- ステート マシンのステート値
- opcode
- プローブ一致値を値ではなく名前で参照。

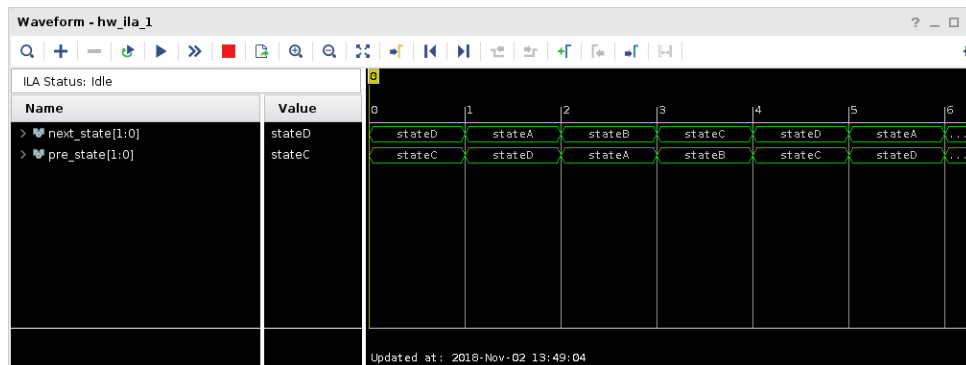
列挙名を設定するには、列挙名と値のペアを入力し、これらをプローブに関連付けます。列挙名と値のペアの関連付けは Tcl コマンドを使用して実行でき、セッション間で保持されます。

列挙型ステートを使用するステート マシンでステート レジスタをプローブするために `mark_debug` 属性を使用すると、そのステート列挙はインプリメンテーションで保持されて、[Waveform] ウィンドウに自動的に表示されます。

図 127: ステート エンコードを使用した RTL

```
13
14
15 (* mark_debug = "true" *) reg [1:0] pre_state;
16 (* mark_debug = "true" *) reg [1:0] next_state;
17
18 localparam stateA = 2'b00;
19 localparam stateB = 2'b01;
20 localparam stateC = 2'b10;
21 localparam stateD = 2'b11;
22
```

図 128: ステート エンコードが保持され、プローブ列挙として表示された波形ビューアー



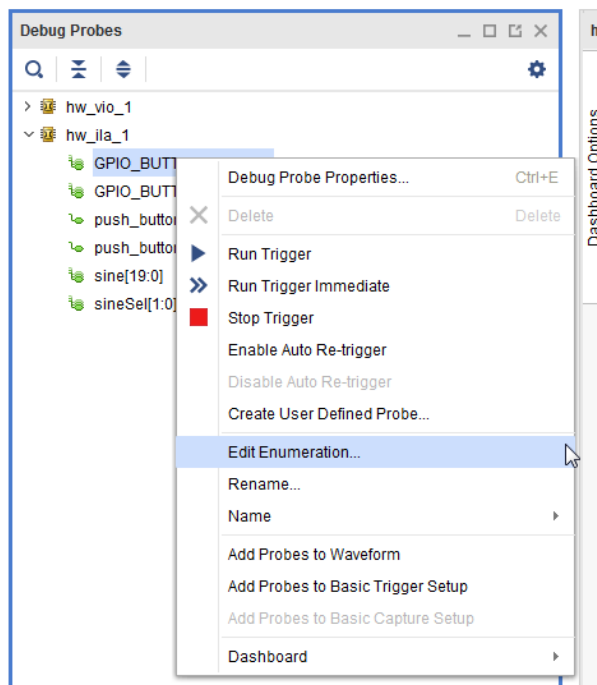
[Trigger Setup] および [Capture Setup] ウィンドウで、プローブ比較値を列挙名を使用して設定できます。プローブとその値に対応する列挙名は、[Waveform] ウィンドウでも表示できます。

列挙名の追加/編集

ハードウェア マネージャーを使用して新しい列挙名を定義

新しい列挙名と値のペアをデバッグ プローブに関連付けるには、[Debug Probes] ウィンドウまたは [Trigger/Capture Setup] ウィンドウでデバッグ プローブを右クリックし、[Edit Enumeration] をクリックします。

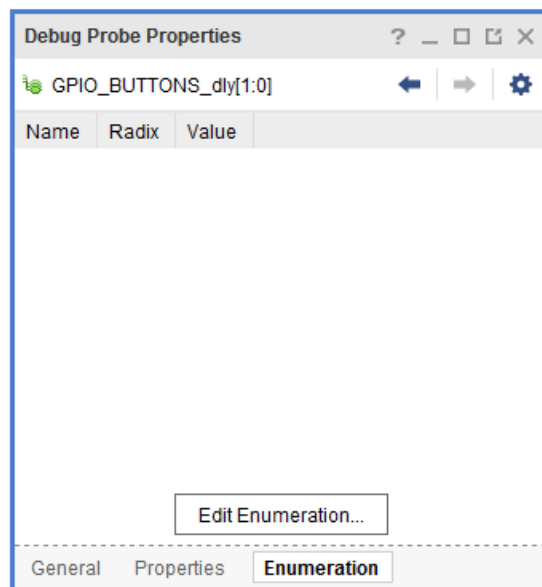
図 129: [Trigger Setup] ウィンドウから列挙名を指定



[Trigger Setup] ウィンドウでデバッグ プロープに関連付けられている列挙名を編集

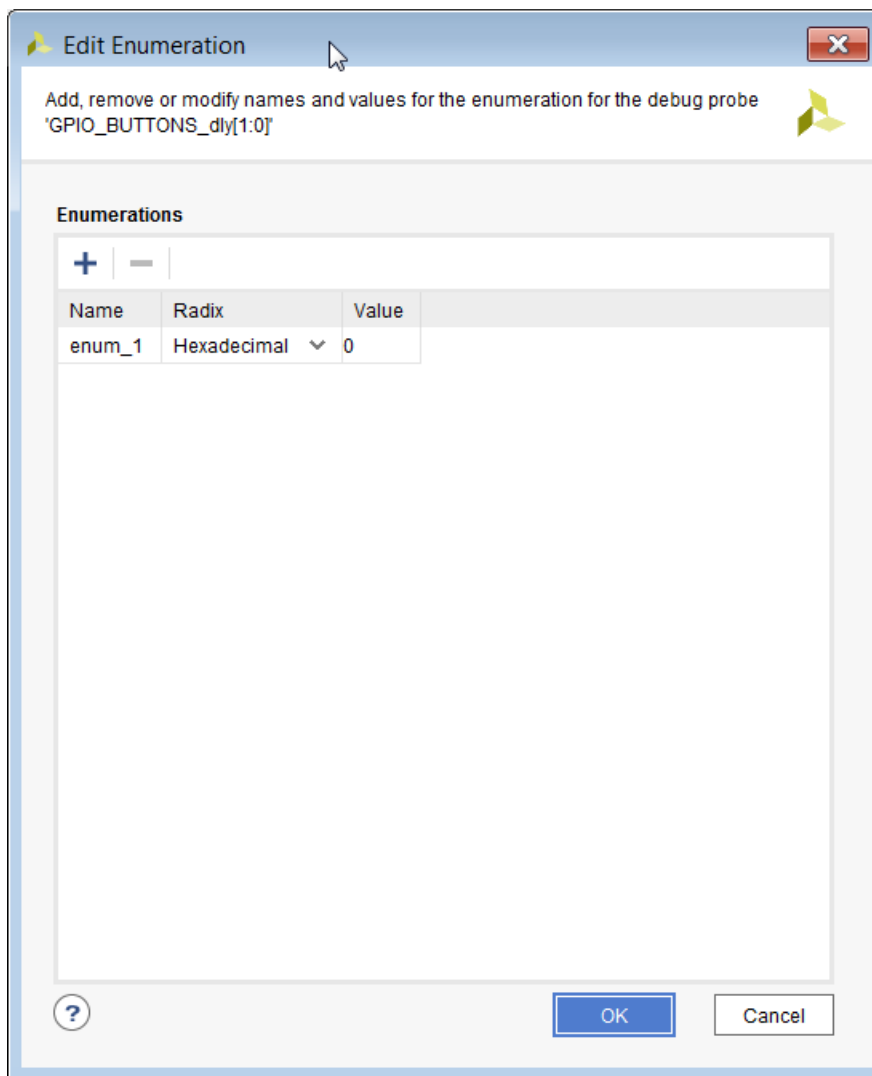
デバッグ プロープに対する新しい列挙名と値のペアの割り当ては、[Debug Probes] ウィンドウまたは [Trigger Setup] ウィンドウでデバッグ プロープを選択し、デバッグ プロープに対する新しい列挙名と値のペアの割り当ては、[Debug Probes] ウィンドウまたは [Trigger Setup] ウィンドウでデバッグ プロープを選択し、

図 130: [Debug Probe Properties] ウィンドウ



[Edit Enumeration] をクリックし [Edit Enumeration] ダイアログ ボックスを開きます。

図 131: [Edit Enumeration] ダイアログ ボックス



名前と値のペアを選択し、左側にある + および - ボタンをクリックして、列挙を追加および削除します。表の [Name]、[Radix]、および [Values] フィールドを変更できます。

Tcl コマンドを使用して列挙名を追加

列挙名と値のペアをデバッグ プローブに関連付けるには、`add_hw_probe_enum` コマンドを使用します。Tcl ファイルに `add_hw_probe` コマンドを追加し、別のファイルに定義を含めることができます。列挙名では入力した大文字/小文字が区別が保持されますが、検索は大文字/小文字を区別せずに実行されます。

例:

```
set probe [get_hw_probes fast_cnt_count -of_objects [get_hw_ilas -
of_objects
[get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"i_fast_ila"}]]
add_hw_probe_enum ZERO eq5'h00 $probe
add_hw_probe_enum TWELVE eq5'u12 $probe
```

```
add_hw_probe_enum THIRTEEN eq5'u13 $probe
add_hw_probe_enum FOURTEEN eq5'u14 $probe
add_hw_probe_enum FIFTEEN eq5'u15 $probe
add_hw_probe_enum SIXTEEN eq5'u16 $probe
add_hw_probe_enum SEVENTEEN eq5'u17 $probe
```

Tcl コマンドを使用して列挙名を削除

`remove_hw_probe_enum` コマンドを使用して、指定した列挙名を削除するか `hw_probe` のすべての列挙名を削除します。

例:

```
remove_hw_probe_enum -list {zero } [get_hw_probes U_SINEGEN/sel -of_objects
[get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter
{CELL_NAME=~"u_ila_0"}]]
```



ヒント: `-remove_all` の `remove_hw_probe_enum` オプションを使用すると、指定したプローブに関連付けられているすべての列挙名が削除されます。

列挙名へのアクセス

列挙名は `hw_probe` プロパティとして保存されるので、`set_property`、`get_property`、および `report_property` コマンドを使用してアクセスできます。

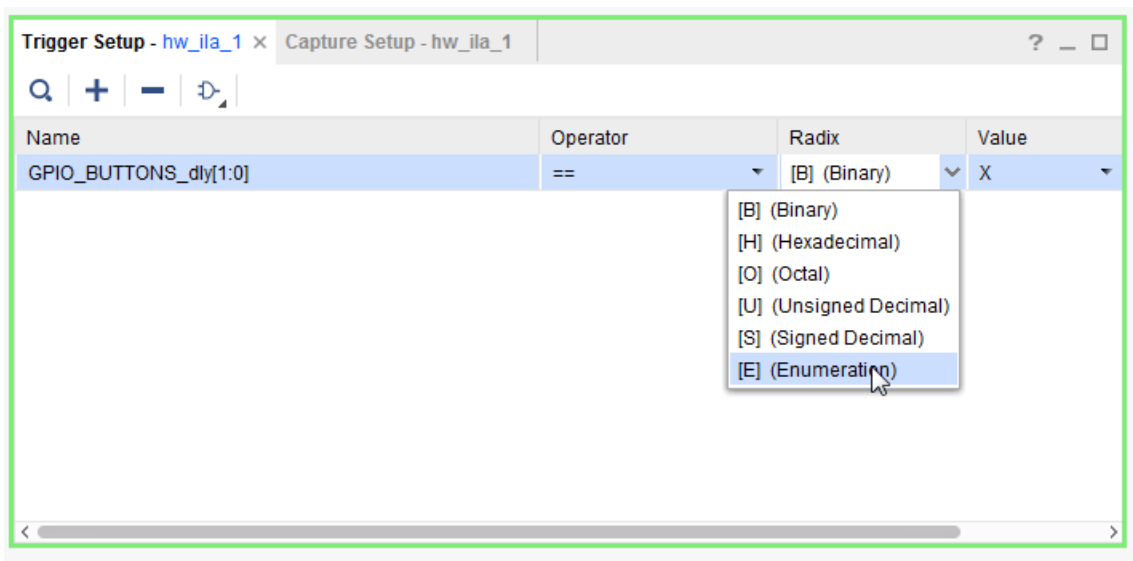
列挙プロパティには接頭辞 `ENUM` が付いており、`set_property` および `get_property` コマンドを使用するときに必要です。次に例を示します。

```
get_property ENUM.FIFTEEN -of_objects [get_hw_ilas -of_objects
[get_hw_devices
xc7k325t_0] -filter {CELL_NAME=~"i_fast_ila"}]]
eq5'u15
report_property [get_hw_probes fast_vio_slice5_fb -of_objects [get_hw_ilas
hw_ila_1]] ENUM*
Property      Type      Read-only Visible Value
ENUM.FIFTEEN  string true      true  eq5'u15
ENUM.FOURTEEN string true      true  eq5'u14
ENUM.SEVENTEEN string true      true  eq5'u17
ENUM.SIXTEEN  string true      true  eq5'u16
ENUM.THIRTEEN string true      true  eq5'u13
ENUM.TWELVE   string true      true  eq5'u12
ENUM.ZERO     string true      true  eq5'h00
set_property ENUM.FIFTEEN eq5'h0F [get_hw_probes fast_vio_slice5_fb -
of_objects
[get_hw_ilas hw_ila_1]]
```

[Trigger Setup] ウィンドウでの列挙名の使用

[Trigger Setup] ウィンドウで比較値に列挙名を設定する場合、構文は数値の比較値と似ています。基数の文字は「e」です。比較値の列挙名では、演算子 `eq` および `neq` のみがサポートされます。

図 132: [Trigger Setup] ウィンドウでの列挙名

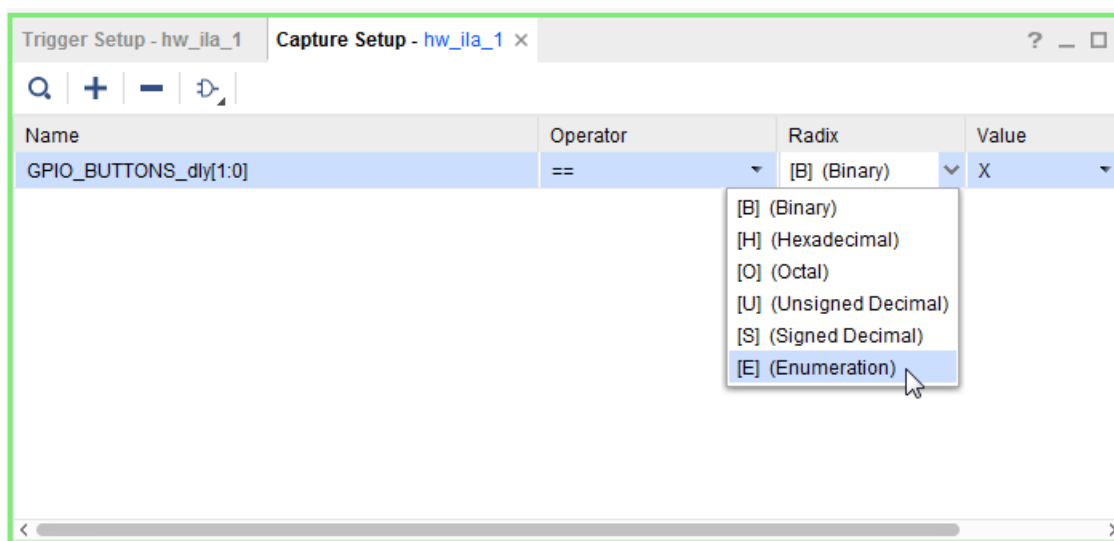


```
set_property TRIGGER_COMPARE_VALUE eq2'ethree [get_hw_probes U_SINEGEN/sel
-of_objects [get_hw_ilas -of_objects [get_hw_devices xc7k325t-0] -filter
{CELL_NAME=~"u_ila_0"}]]
```

[Capture Setup] ウィンドウでの列挙名の使用

Vivado IDE の [Capture Setup] ウィンドウまたは Tcl コマンドで、列挙名を使用して値を比較することもできます。

図 133: [Capture Setup] ウィンドウでの列挙名



```
set_property CAPTURE_COMPARE_VALUE eq2'eone [get_hw_probes locked -
of_objects
[get_hw_ilas -of_objects [get_hw_devices xc7k325t-0] -filter
{CELL_NAME=~"u_ila_0_0"}]]
```

アドバンス トリガー

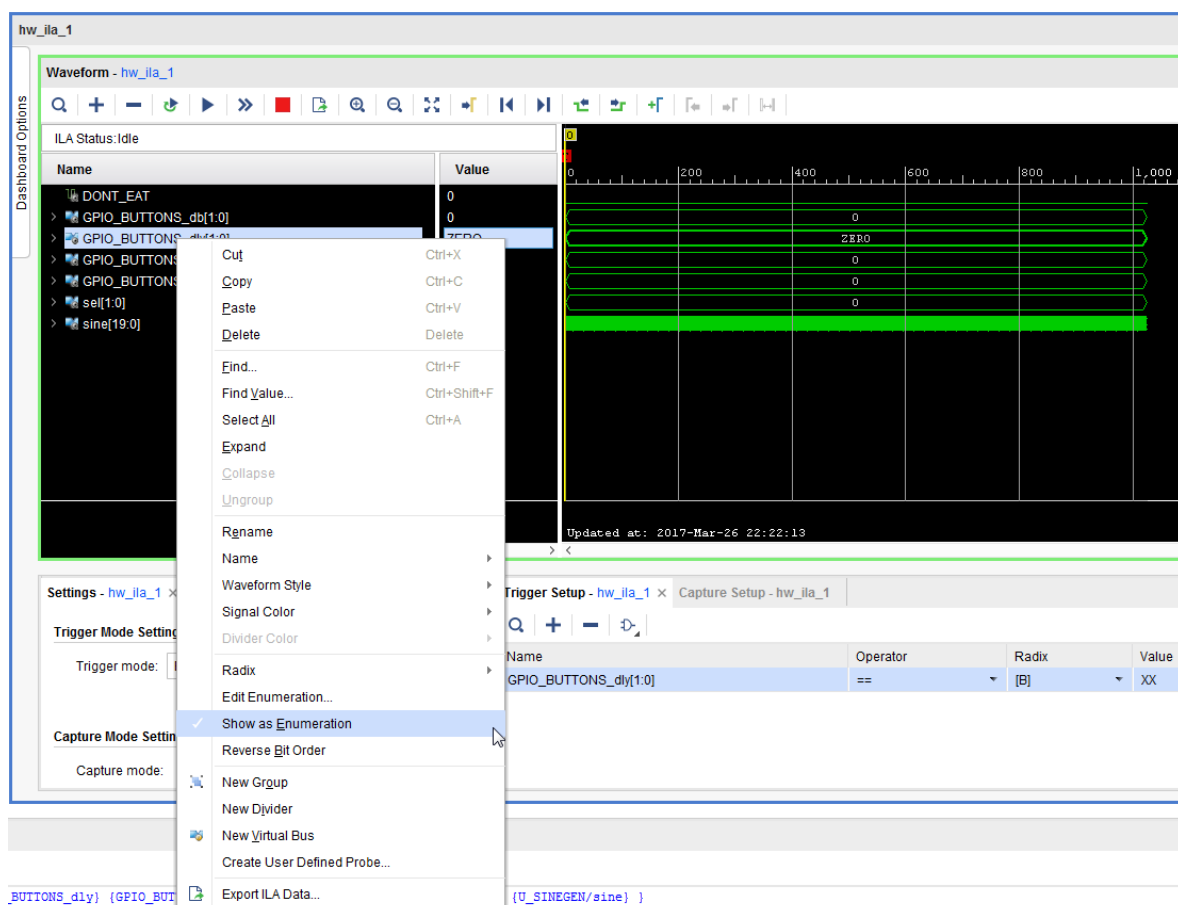
アドバンス トリガー ステート マシン スクリプトで、列挙名を使用して値を比較できます。次に例を示します。

```
state my_state0:
  if (fast_ila_slice5_fb == 5'eFIFTEEN) then
    set_flag $flag1;
    goto my_state0;
  elseif (fast_ila_slice5_fb == 5'eTWELVE) then
    trigger;
  else
    clear_flag $flag1;
    goto my_state0;
  endif
```

[Waveform] ウィンドウでの列挙名の使用

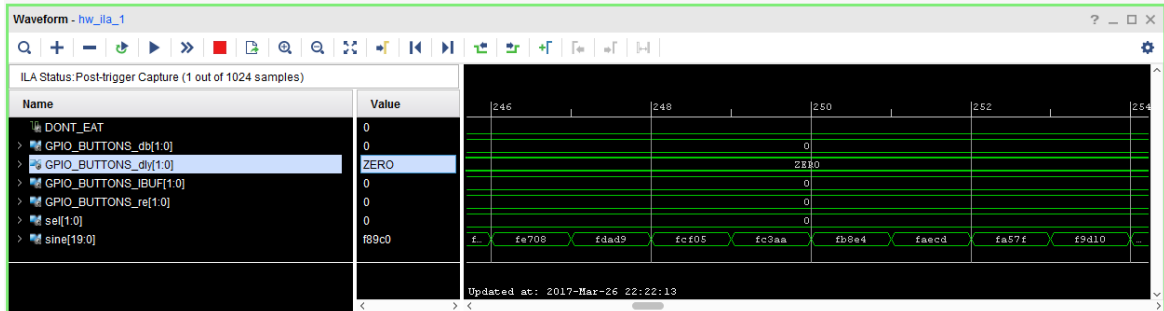
[Waveform] ウィンドウで各信号に対して [Show as Enumeration] オプションを選択すると、列挙名を表示できます。波形ウィンドウで信号を右クリックして [Show as Enumeration] をクリックします。列挙名を表示しない場合、通常の基数の選択に基づいてバス値が表示されます。

図 134: [Waveform] ウィンドウで [Show as Enumeration] をクリック



列举情報は波形データ ファイルに保存され、次回波形データを表示したときに使用されます。列举名が定義されている波形プローブは、デフォルトで列举名で表示されます。

図 135: 波形で列挙名を表示



波形オブジェクトに対して [Show as Enumeration] が選択されていると、列挙名が表示されます。波形値に対応する列挙名がない場合は、選択した基数に基づく値が表示されます。



重要: 列挙名を追加する前に波形が作成されていた場合、次の Tcl コマンドを使用して波形 ILA データを保存すると、波形に新しい列挙名が適用されます。

```
write_hw_ila_data -force data_ila-3.ila [upload_hw_ila_data hw_ila-3]
display_hw_ila_data [read_hw_ila_data ./data_ila-3.ila]
```

ハードウェア マネージャーでの AXI インターフェイスのデバッグ

IP インテグレーターの System ILA デバッグ コアを使用すると、FPGA 上でデザインにインシステム デバッグを実行できます。この機能は、IP インテグレーター デザインでインターフェイスおよび信号を監視する必要がある場合に使用します。

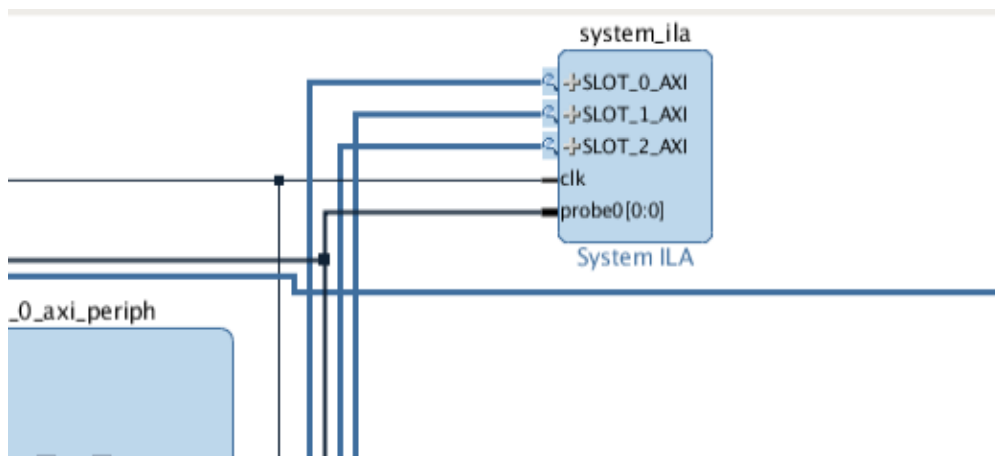
ブロック デザインでインターフェイスおよびネットをデバッグする手順は、『Vivado Design Suite ユーザー ガイド: IP インテグレーターを使用した IP サブシステムの設計』(UG994)のこのセクションを参照してください。

IP インテグレーター ブロックデザインに System ILA デバッグ コアをインスタンス化している場合は、波形ウィンドウで AXI トランザクションとその読み出しおよび書き込みイベントをデバッグおよび監視できます。

波形ウィンドウおよび AXI インターフェイス

System ILA デバッグ コアを使用すると、スロットとしてインターフェイスをデバッグおよび監視できます。各スロットは IP インテグレーター ブロック デザインでデバッグされているインターフェイスに対応しています。次の図には、スロット 0、スロット 1 に System ILA IP によりプローブされている AXI メモリ マップ インターフェイスがそれぞれ 1 つずつ、合計 2 つあります。

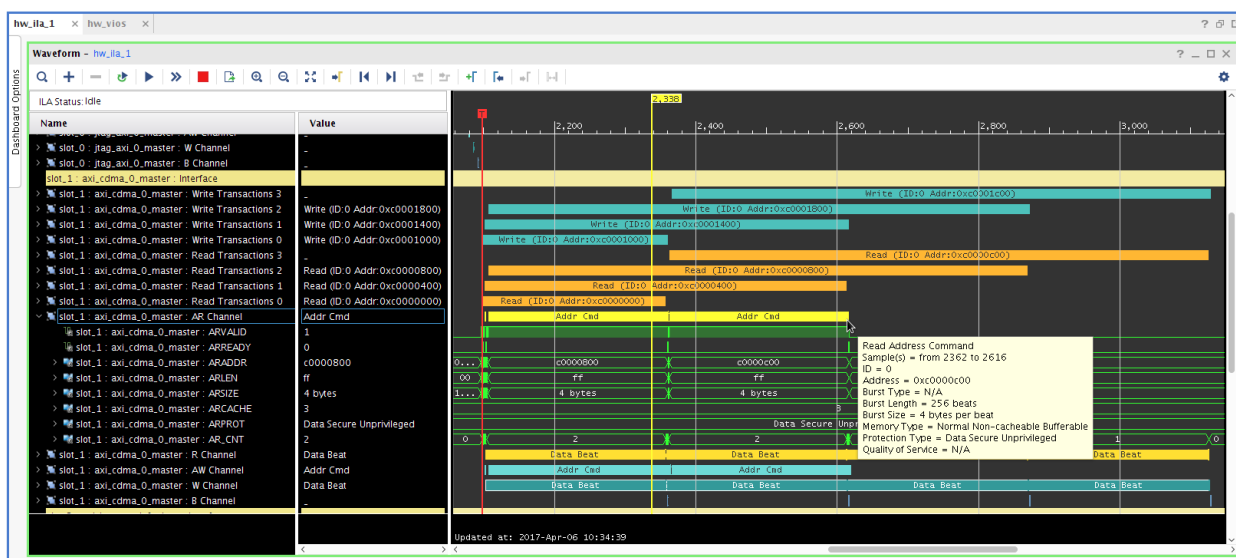
図 136: 2 つの AXI メモリ マップインターフェイスをプローブ



波形ビューアーの AXI トランザクション

System ILA でデバッグされる AXI3、AXI4、AXI4-Lite インターフェイスに関連するトランザクションは、次の図のように波形ビューアーで確認できます。

図 137: 波形ビューアーの AXI トランザクション



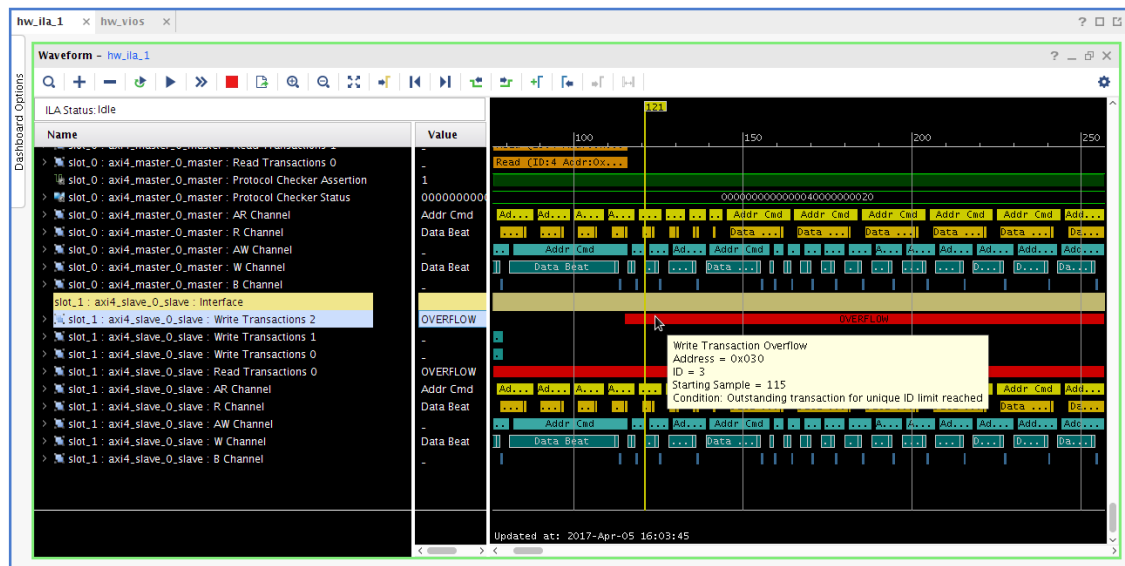
AXI トランザクションは、次のように定義されます。

- AR (Read Address) チャネルの Address Command イベントの始まりで開始される読み出しトランザクション。
- R (Read Data) チャネルの Last Read Data イベントで終了される読み出しトランザクション。
- AW (Write Address) チャネルの Address Command イベントの始まりで開始される書き込みトランザクション。
- B (Write Response) チャネルの Write Response イベントで終了される書き込みトランザクション。

トランザクションは、アドレス、データ、応答イベントのいずれかまたはすべての ID が同じ場合に表示されます。また、トランザクションは、開始イベントと終了イベントの両方がキャプチャされたデータ波形内にある場合にのみ波形に表示されます。複数の Multiple Outstanding (複数の未処理)/Multiple Overlapping (複数のオーバーラップ) トランザクションが [Waveform] ウィンドウに表示される場合、複数のトランザクション行が使用されます。

インターフェイスのトランザクションにより、System ILA IP 内でトランザクション トラッキング ロジックが未処理になって、次のようにオーバーフローしてしまう可能性があります。

図 138: AXI トランザクションのカウンター オーバーフロー

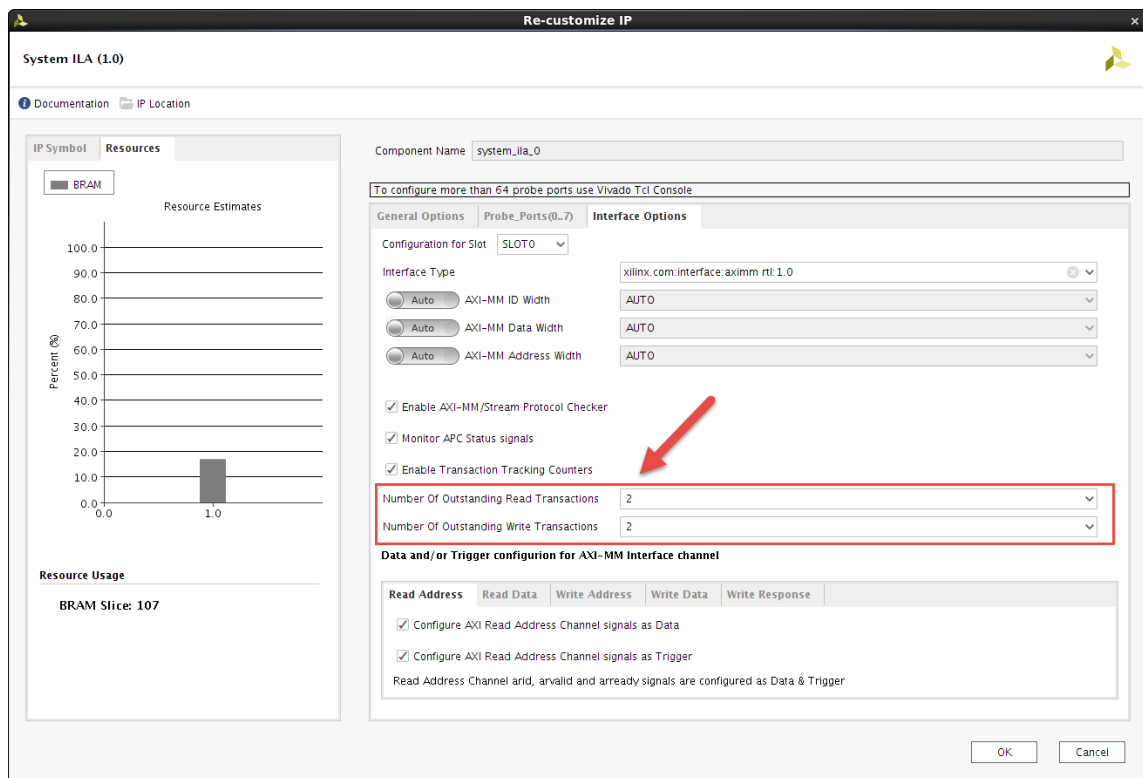


オーバーフローは、次の 2 つの状況下で発生します。

- 特定 ID の未処理のトランザクションの数により、トランザクション カウンターの容量がオーバーフローする場合。
- 未処理のトランザクションのある ID の数により、使用可能なカウンター数がオーバーフローする場合。

どちらの場合も、IP インテグレーター ブロック デザインで System ILA コアの未処理の読み出し/書き込みトランザクション数を増加してカスタマイズし直せば、オーバーフロー状況は回避できます。次の図を参照してください。

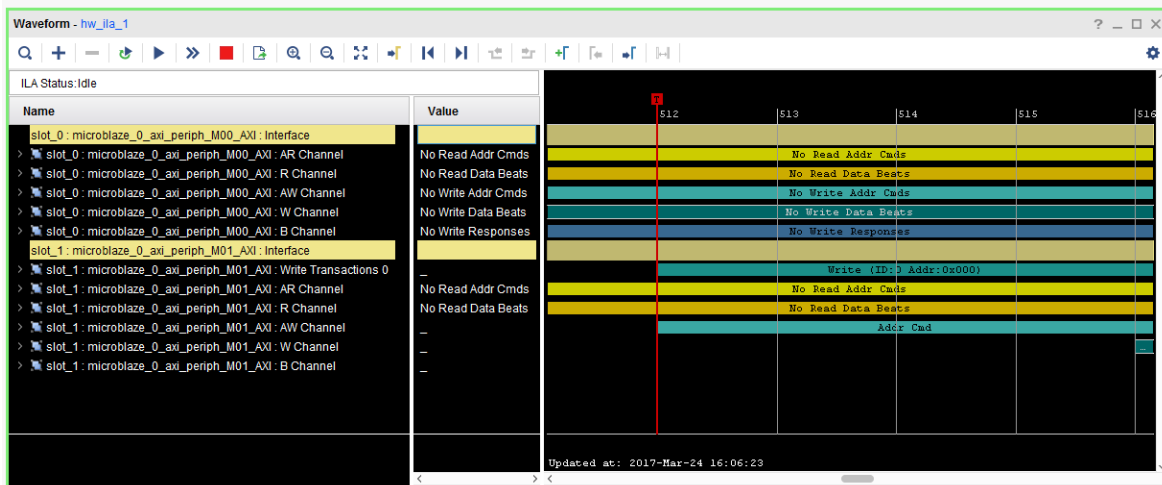
図 139: System ILA でのトラッキング可能な未処理トランザクションの数の増加



AXI インターフェイス イベント

Vivado ハードウェア マネージャーでは、System ILA IP でデザインの AXI インターフェイスをデバッグする場合、波形ウィンドウに System ILA でプローブされているインターフェイスに対応するスロット、イベント、信号グループが表示されます。次の図にあるように、[Waveform] ウィンドウには、System ILA IP でプローブされた 2 つのインターフェイス スロット両方が表示されています。スロット 1 には、AXI トランザクション、書き込みアドレス チャンネル イベント、書き込みデータ チャンネル イベントがあります。[Waveform] ウィンドウには Write Data CAXI インターフェイス スロットもあります。

図 140: AXI インターフェイスのトランザクションおよびイベント



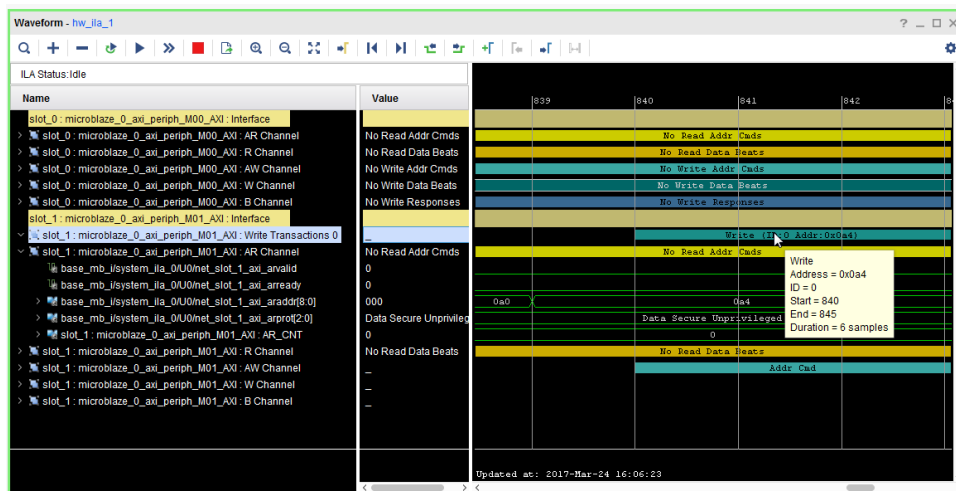
この波形は、読み出し、書き込み、アドレス、およびデータのチャンネル イベントで AXI インターフェイス関連のトランザクションおよびイベントをレポートします。

AXI トランザクション

AXI トランザクションでは、AXI 読み出しアドレス、読み出しデータ、書き込みアドレス、および書き込みデータ チャンネルでの読み出しおよび書き込みトランザクションがレポートされます。

[Waveform] ウィンドウで特定の読み出しまたは書き込みトランザクションにカーソルを置くと、その特定のトランザクションに関連するアドレス、ID、開始、終了、期間などが表示されます。

図 141: AXI トランザクション



AXI チャンネル イベント

AXI チャンネル イベント グループは、AXI 読み出しアドレス (AR)、読み出しデータ (R)、書き込みアドレス (AW)、書き込みデータ (W)、書き込み応答 (B) チャンネルでの AXI イベントをレポートします。

読み出しアドレス (AR) チャンネル イベント

名前	説明
No Read Addr Cmds	読み出しアドレス チャンネルでアドレス コマンド イベントが発生しなかったことを示します。
Addr Cmd	読み出しトランザクションの有効なアドレス コマンド フェーズを示します。このイベントは ARVALID = '1' で開始し、同じクロック サイクルで ARVALID = '1' および ARREADY = '1' の両方になると終了します。

読み出しアドレス チャンネルの信号グループ

この信号グループには、読み出しアドレス チャンネル イベントに関わる信号すべてが含まれます。信号は、次のとおりです。

- ネット名
 - ARVALID
 - ARREADY
 - ARID
 - ARADDR
 - ARBURST
 - ARLEN
 - ARSIZE
 - ARCACHE
 - ARPROT
 - ARLOCK
 - ARQOS
 - AR_CNT

読み出しデータ チャンネル イベント

名前	説明
No Read Data Beats	読み出しデータ チャンネルでイベントが発生しなかったことを示します。
Data Beat	読み出しトランザクションの最後以外のデータ ビートを示します。このイベントは、現在のクロック サイクルで RVALID = '1' および RREADY = '1' になると発生します。
Last Data	読み出しトランザクションの最後のデータ ビートを示します。このイベントは、現在のクロック サイクルで RVALID = '1' および RREADY = '1' および RLAST = '1' になると発生します。

読み出しデータ チャンネルの信号グループ

この信号グループには、読み出しデータ チャンネル イベントに関わる信号すべてが含まれます。信号は、次のとおりです。

- ネット名
 - RVALID
 - RREADY
 - RLAST
 - RID
 - RDATA
 - RRESP
 - R_CNT

書き込みアドレス チャンネル イベント

名前	説明
No Write Addr Cmds	書き込みアドレス チャンネルでアドレス コマンド イベントが発生しなかったことを示します。
Addr Cmd	書き込みトランザクションの有効なアドレス コマンド フェーズを示します。このイベントは AWVALID = '1' で開始し、同じサイクルで AWVALID = '1' および AWREADY = '1' になると終了します。

書き込みアドレス チャンネルの信号グループ

この信号グループには、書き込みアドレス チャンネル イベントに関わる信号すべてが含まれます。信号は、次のとおりです。

- ネット名
 - AWVALID
 - AWREADY
 - AWID
 - AWADDR
 - AWBURST
 - AWLEN
 - AWSIZE
 - AWCACHE
 - AWPROT
 - AWLOCK
 - AWQOS
 - AW_CNT

書き込みデータ チャンネル イベント

名前	説明
No Write Data Beats	書き込みデータ チャンネルでイベントが発生しなかったことを示す。
Data Beat	書き込みトランザクションの最後以外のデータ ビートを示します。このイベントは、現在のクロック サイクルで WVALID = '1' および WREADY = '1' になり、さらに前のクロック サイクルでいずれかの信号が 0 だった場合に発生します。
Last Data	書き込みトランザクションの最後のデータ ビートを示します。このイベントは、現在のクロック サイクルで WVALID = '1' および WREADY = '1' および WLAST = '1' になると発生します。

書き込みデータ チャンネルの信号グループ

この信号グループには、書き込みデータ チャンネル イベントに関わる信号すべてが含まれます。信号は、次のとおりです。

- ネット名
 - WVALID
 - WREADY
 - WLAST
 - WDATA
 - WSTRB

書き込み応答チャンネル イベント

名前	説明
No Write Responses	書き込み応答チャンネルでイベントが発生しなかったことを示す。
Write Response	書き込みトランザクションの応答フェーズを示します。このイベントは、現在のクロック サイクルで BVALID = '1' および BREADY = '1' になると発生します。

書き込み応答チャンネルの信号グループ

この信号グループには、書き込み応答チャンネル イベントに関わる信号すべてが含まれます。信号は、次のとおりです。

- ネット名
 - BVALID
 - BREADY
 - BID
 - BRESP
 - B_CNT

AXI アドレス コマンドおよびデータ ビートでのトリガー

AXI インターフェイスのデバッグには、アドレスの終了 (End of the Address) コマンド、データ ビートの終了 (End of Data Beat) コマンド、書き込み応答 (Write Response) コマンドの 3 つの AXI イベントでのトリガーが関係していることがよくあります。異なるインターフェイス チャンネルでは、これらのイベントの 1 つ以上でトリガーする必要がある場合が多くあります。たとえば、読み出しアドレスの終了コマンドまたは書き込みアドレスの終了コマンドのトリガー条件をインプリメントするには、次の式で計算します。

$$\text{トリガー条件} = (((\text{ARVALID} == 1) \&\& (\text{ARREADY} == 1)) \parallel ((\text{AWVALID} == 1) \&\& (\text{AWREADY} == 1)))$$

ただし、これには積和 (SOP) 形式のブール代数式が必要ですが、必須の AXI 信号 (ARVALID および ARREADY など) が異なるプロープポートにある場合はインプリメントできません。このタイプのトリガーを使用するため、次の図のように、必須の *VALID、*READY、および *LAST 制御信号が 1 つのプロープポートでまとめられます。

表 12: AXI チャンネル制御信号のプロープ

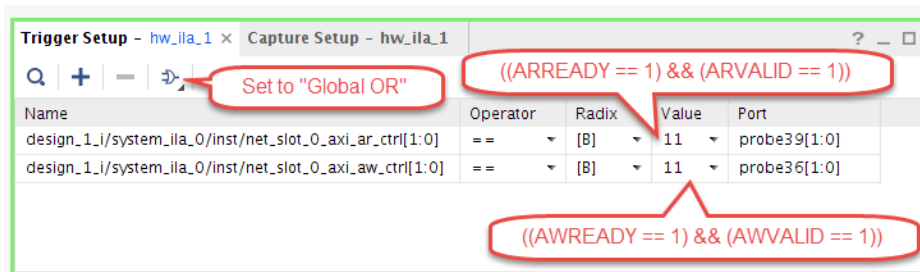
説明	プロープ名	ビット 2	ビット 1	ビット 0
読み出しアドレス チャンネルの制御信号	*ar_ctrl[1:0]	なし	ARREADY	ARVALID
読み出しデータ チャンネルの制御信号	*r_ctrl[2:0]	RLAST	RREADY	RVALID
書き込みアドレス チャンネルの制御信号	*aw_ctrl[1:0]	なし	AWREADY	AWVALID
書き込みデータ チャンネルの制御信号	*w_ctrl[2:0]	WLAST	WREADY	WVALID
書き込み応答チャンネルの制御信号	*b_ctrl[1:0]	なし	BREADY	BVALID

次の表に、各 AXI 制御信号プロープと AXI チャンネル制御プロープの両方を使用して、便利で基本的なトリガーおよびキャプチャ制御式をインプリメントする方法を示します。その後の図に、基本的なトリガー設定 GUI を使用して、読み出しアドレスの終了コマンドまたは書き込みアドレスの終了コマンドをインプリメントする方法を示します。

表 13: トリガーおよびキャプチャ制御イベント

AXI イベント	各 AXI 制御信号	統合 AXI チャンネル制御プロープ
読み出しアドレスの終了コマンド	((ARVALID == 1) && (ARREADY == 1))	(*ar_ctrl == 2'b11)
最後の読み出しデータ ビートの終了	((RVALID == 1) && (RREADY == 1) && (RLAST == 1))	(*r_ctrl == 3'b111)
最後以外の読み出しデータ ビートの終了	((RVALID == 1) && (RREADY == 1) && (RLAST == 0))	(*r_ctrl == 3'b011)
書き込みアドレスの終了コマンド	((AWVALID == 1) && (AWREADY == 1))	(*aw_ctrl == 2'b11)
最後の書き込みデータ ビートの終了	((WVALID == 1) && (WREADY == 1) && (WLAST == 1))	(*w_ctrl == 3'b111)
最後以外の読み出しデータ ビートの終了	((WVALID == 1) && (WREADY == 1) && (WLAST == 0))	(*w_ctrl == 3'b011)

図 142: 読み出しアドレスの終了コマンドまたは書き込みアドレスの終了コマンドの基本的なトリガー設定



計測のための VIO コアの設定

デザインに追加した VIO コアは、[Hardware] ウィンドウのターゲット デバイスの下に表示されます。VIO コアが表示されていない場合は、デバイスを右クリックして [Refresh Hardware] をクリックします。FPGA デバイスが再度スキャンされ、[Hardware] ウィンドウの表示が更新されます。

注記: FPGA デバイスをプログラムまたは更新しても VIO コアが表示されない場合は、デバイスが正しい .bit ファイルでプログラムされているか、インプリメント済みデザインに VIO コアが含まれているかを確認してください。また、.bit ファイルに対応する適切な .ltx プロブ ファイルがデバイスに関連付けられているかどうかを確認してください。

VIO コア (次の図では hw_vio_1) を選択すると、[VIO Core Properties] ウィンドウにプロパティが表示されます。VIO を選択すると、その VIO コアが [Debug Probes] ウィンドウおよび Vivado IDE ワークスペースの VIO ダッシュボードでも選択されます (次の図を参照)。

図 143: [Hardware] ウィンドウに表示される VIO コア

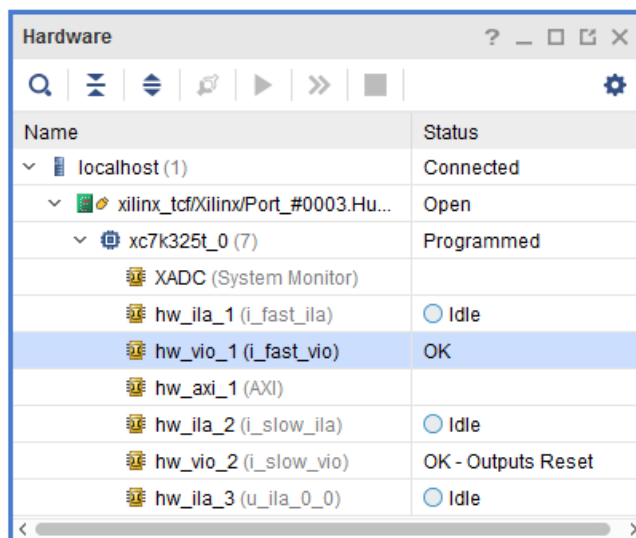
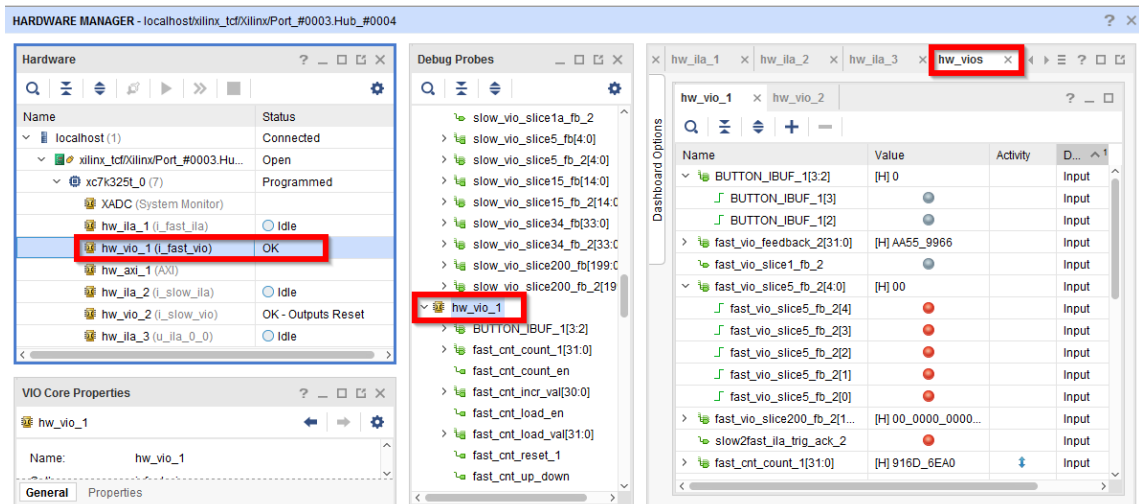


図 144: さまざまなウィンドウでの VIO コアを選択



VIO コアが Vivado IDE と同期しなくなることがあります。VIO ステータス インジケータの解釈方法については、「VIO コアのステータスの表示」を参照してください。

VIO コアの値を読み出したり書き込んだりするには、オブジェクト プロパティを使用します。

- VIO 入力プローブの値を読み出すには、まず hw_vio オブジェクトを VIO コアの値で更新し、hw_vio オブジェクトのプロパティ値を取得します。詳細は、「VIO コアの入力プローブ」セクションを参照してください。
- VIO 出力プローブの値を書き込むには、まず hw_probe オブジェクトのプロパティ値を設定し、これらのプロパティ値をハードウェアの VIO コアに確定して、値をコアの出力プローブ ポートに書き込みます。詳細は、「VIO コアの入力プローブ」セクションを参照してください。

関連情報

[VIO コアのステータスの表示](#)

[VIO コアの入力プローブ](#)

[VIO コアの出力プローブ](#)

VIO コアのステータスの表示

VIO コアには、入力プローブが 0 個以上、出力プローブが 0 個以上あります (少なくとも入力プローブまたは出力プローブが 1 個以上必要)。[Hardware] ウィンドウの示される VIO コアのステータスは、VIO コアの出力プローブの現在の状態を示します。次の表に、VIO コアのステータスと必要な操作を示します。

表 14: VIO コアのステータスと必要なユーザー操作

VIO のステータス	説明	必要なユーザー操作
OK - Outputs Reset	VIO コアの出力は Vivado IDE と同期しており、出力は初期 (リセット) 状態です。	なし
OK	VIO コアの出力は Vivado IDE と同期していますが、出力は初期 (リセット) 状態ではありません。	なし

表 14: VIO コアのステータスと必要なユーザー操作 (続き)

VIO のステータス	説明	必要なユーザー操作
Outputs out-of-sync	VIO コアの出力が Vivado IDE と同期していません。	次の 2 つのいずれかの操作を実行する必要があります。 [Hardware] ウィンドウで VIO コアを右クリックして [Commit VIO Core Outputs] をクリックし、Vivado IDE からの値を VIO コアに書き込みます。 [Hardware] ウィンドウで VIO コアを右クリックして [Refresh Input and Output Values from VIO Core] をクリックし、Vivado IDE を VIO コア出力プロブポートの現在の値でアップデートします。

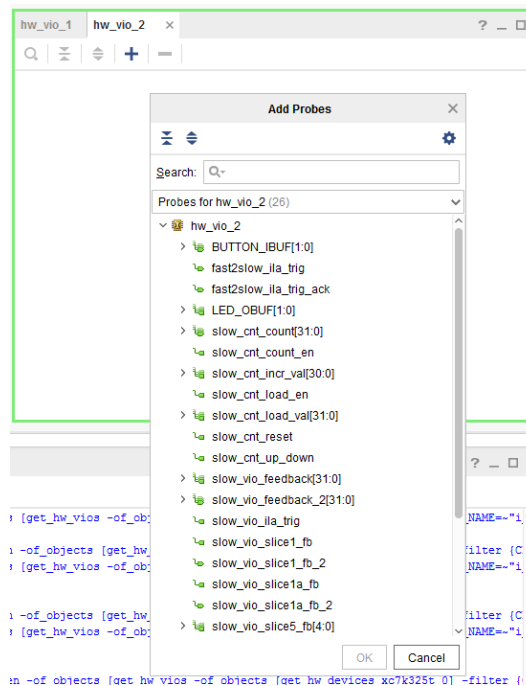
[Debug Probes] ウィンドウでの VIO コアの表示

VIO のダッシュボードで + 記号をクリックすると、VIO コアに関連付けられているデバッグ プロブを表示、追加、削除できます。

VIO ダッシュボードの使用

VIO のデフォルト ダッシュボードは、次の図に示すように最初は空白です。

図 145: VIO のデフォルト ダッシュボード



VIO ダッシュボードには、VIO コアに関するすべてのステータスおよび制御情報が表示されます。ハードウェア デバイスの更新により VIO コアが最初に検出されると、コアの VIO ダッシュボードが自動的に開きます。ダッシュボードを手動で開いたり、開き直したりする必要がある場合は、[Hardware] または [Debug Probes] ウィンドウで VIO コア オブジェクトを右クリックして [Open Dashboard] をクリックします。

VIO コアの入力プローブ

VIO コアの入力プローブは、実際のハードウェアの FPGA で実行されているデザインからの値を読み出すために使用します。VIO 入力プローブは通常、テスト中のデザインのステータス インジケータとして使用されます。VIO デバッグ プローブは、VIO ダッシュボードの [VIO Probes] ウィンドウに手動で追加する必要があります。詳細な方法は、「[Debug Probes] ウィンドウでの VIO コアの表示」を参照してください。VIO ダッシュボードの [VIO Probes] ウィンドウでは、VIO 入力プローブが次のように表示されます。

図 146: コアの入力プローブ

Name	Value	Activity	D...	VIO
▼ BUTTON_IBUF_1[3:2]	[H] 0		Input	hw_vio_1
└─ BUTTON_IBUF_1[3]			Input	hw_vio_1
└─ BUTTON_IBUF_1[2]			Input	hw_vio_1
> fast_vio_feedback_2[31:0]	[H] AA55_9966		Input	hw_vio_1
└─ fast_vio_slice1_fb_2			Input	hw_vio_1
▼ fast_vio_slice5_fb_2[4:0]	[H] 00		Input	hw_vio_1
└─ fast_vio_slice5_fb_2[4]			Input	hw_vio_1
└─ fast_vio_slice5_fb_2[3]			Input	hw_vio_1
└─ fast_vio_slice5_fb_2[2]			Input	hw_vio_1
└─ fast_vio_slice5_fb_2[1]			Input	hw_vio_1
└─ fast_vio_slice5_fb_2[0]			Input	hw_vio_1
> fast_vio_slice200_fb_2[1...	[H] 00_0000_0000...		Input	hw_vio_1
└─ slow2fast_ila_trig_ack_2			Input	hw_vio_1
> fast_cnt_count_1[31:0]	[H] 58C4_BC34		Input	hw_vio_1
└─ fast_vio_slice1a_fb_2	[B] 0		Input	hw_vio_1

関連情報

[\[Debug Probes\] ウィンドウでの VIO コアの表示](#)

[VIO Probes] ウィンドウを使用した VIO 入力の読み出し

VIO 入力プローブは、VIO ダッシュボードの [VIO Probes] ウィンドウに表示されます。各入力プローブが表で個別の行に表示されます。VIO 入力プローブの値は、[Value] 列に示されます（「VIO コアの入力プローブ」を参照）。VIO コアの入力値は、VIO コアの更新レートに基づいて、定期的にアップデートされます。更新レートを変更するには、[VIO Properties] ウィンドウで [Refresh Rate (ms)] の値を変更するか、次の Tcl コマンドを使用します。

```
set_property CORE_REFRESH_RATE_MS 1000 [get_hw_vios hw_vio_1]
```

注記: 更新レートを 0 に設定すると、VIO コアからのすべての自動更新が停止します。また、小さい値を設定すると、Vivado IDE の動作が遅くなることがあります。更新レートは 500 ms 以上に設定することをお勧めします。

VIO 入力プローブの値を手動で読み出すには、Tcl コマンドを使用します。たとえば、VIO コア hw_vio_1 の BUTTON_IBUF という入力プローブの値を更新して読み出すには、次の Tcl コマンドを使用します。

```
refresh_hw_vio [get_hw_vios {hw_vio_1}]
get_property INPUT_VALUE [get_hw_probes BUTTON_IBUF]
```

関連情報

[VIO コアの入力プローブ](#)

VIO 入力の表示タイプと基数の設定

VIO 入力プローブの表示タイプは、VIO ダッシュボードの [VIO Probes] ウィンドウで VIO 入力プローブを右クリックし、次のいずれかを選択すると設定できます。

- [Text]: 入力をテキスト フィールドとして表示します。これが、VIO 出力プローブ ベクター (幅が 2 ビット以上) の唯一の表示タイプです。
- [LED]: 入力を LED グラフィックで表示します。この表示タイプは、VIO 入力プローブ スカラーおよび VIO 入力プローブ ベクターの個々のエレメントにのみ選択可能です。High および Low の値を、次の 4 つの色のいずれかに設定できます。
 - 。 グレー (オフ)
 - 。 赤
 - 。 緑
 - 。 青

VIO 入力プローブの表示タイプを [Text] に設定すると、基数を変更できます。これには、VIO ダッシュボードの [VIO Probes] ウィンドウで VIO 入力プローブを右クリックし、次のいずれかをクリックします。

- [Radix] → [Binary]: 基数を 2 進数に設定します。
- [Radix] → [Octal]: 基数を 8 進数に設定します。
- [Radix] → [Hex]: 基数を 16 進数に設定します。
- [Radix] → [Unsigned]: 基数を符号なしの 10 進数に設定します。
- [Radix] → [Signed]: 基数を符号付きの 10 進数に設定します。

VIO 入力プローブの基数は、Tcl コマンドでも設定できます。たとえば、BUTTON_IBUF という VIO 入力プローブの基数を変更するには、次の Tcl コマンドを使用します。

```
set_property INPUT_VALUE_RADIX HEX [get_hw_probes BUTTON_IBUF]
```

VIO 入力アクティビティの監視と制御

VIO 入力プローブから値を読み出すだけでなく、VIO 入力プローブのアクティビティを監視することもできます。Vivado IDE の周期的なアップデートの間に VIO 入力の値が変化すると、それが示されます。

VIO 入力プローブのアクティビティは、VIO ダッシュボードの [VIO Probes] ウィンドウの [Activity] 列に示されます。

- 上向き矢印: アクティビティ 保持期間内に、入力プローブの値が 0 から 1 に遷移したことを示します。
- 下向き矢印: アクティビティ 保持期間内に、入力プローブの値が 1 から 0 に遷移したことを示します。
- 上下矢印: アクティビティ 保持期間内に、入力プローブの値が 1 から 0 に、および 0 から 1 に少なくとも 1 回以上遷移したことを示します。

入力アクティビティ ステータスの表示を保持する期間を変更するには、VIO ダッシュボードの [VIO Probes] ウィンドウで VIO 入力プローブを右クリックし、次のいずれかをクリックします。

- [Activity Persistence > Infinite]: アクティビティ 値をユーザーがリセットするまで累積し、保持します。
- [Activity Persistence > Long (80 samples)]: アクティビティ 値を長い期間累積し、保持します。
- [Activity Persistence > Short (8 samples)]: アクティビティ 値を短い期間累積し、保持します。

Tcl コマンドでもアクティビティ 保持期間を設定できます。たとえば、BUTTON_IBUF という VIO 入力プローブのアクティビティ 保持期間を長い期間に変更するには、次の Tcl コマンドを使用します。

```
set_property ACTIVITY_PERSISTENCE LONG [get_hw_probes BUTTON_IBUF]
```

コアのすべての入力プローブのアクティビティをリセットするには、[Hardware] ウィンドウで VIO コアを右クリックして [Reset All Input Activity] をクリックします。次の Tcl コマンドでも同じ操作を実行できます。

```
reset_hw_vio_activity [get_hw_vios {hw_vio_1}]
```

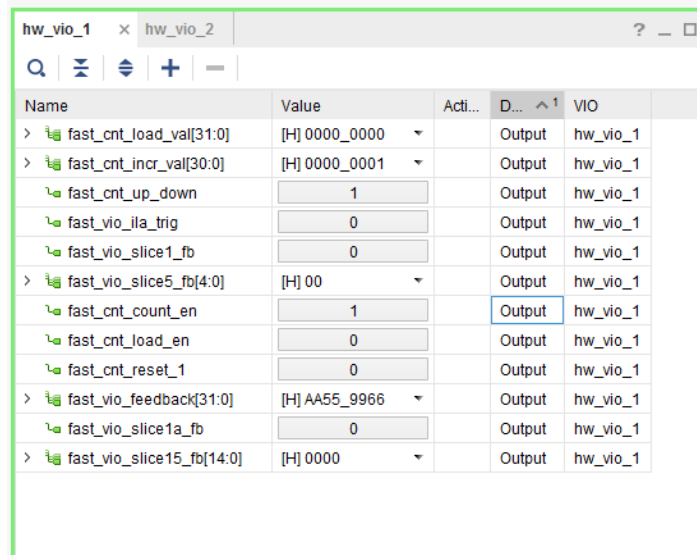


ヒント: VIO 入力プローブ ベクターの複数のスカラー メンバーのタイプ、基数、アクティビティ 保持期間は、プローブ全体またはプローブの複数メンバーを右クリックして、メニューをクリックすると変更できます。クリックしたメニューの設定は、選択したすべてのプローブ スカラーに適用されます。

VIO コアの出カプローブ

VIO コアの出カプローブは、実際のハードウェアの FPGA デバイスで実行されているデザインに値を書き込むために使用します。VIO 出カプローブは通常、テスト中のデザインの低帯域幅の制御信号として使用されます。VIO デバッグ プローブは、VIO ダッシュボードの [VIO Probes] ウィンドウに手動で追加する必要があります。詳細な方法は、「[Debug Probes] ウィンドウでの VIO コアの表示」を参照してください。VIO ダッシュボードの [VIO Probes] ウィンドウでは、VIO 出カプローブが次のように表示されます。

図 147: VIO ダッシュボードの [VIO Probes] ウィンドウの VIO 出カ



Name	Value	Acti...	D... ^1	VIO
> fast_cnt_load_val[31:0]	[H] 0000_0000		Output	hw_vio_1
> fast_cnt_incr_val[30:0]	[H] 0000_0001		Output	hw_vio_1
fast_cnt_up_down	1		Output	hw_vio_1
fast_vio_ila_trig	0		Output	hw_vio_1
fast_vio_slice1_fb	0		Output	hw_vio_1
> fast_vio_slice5_fb[4:0]	[H] 00		Output	hw_vio_1
fast_cnt_count_en	1		Output	hw_vio_1
fast_cnt_load_en	0		Output	hw_vio_1
fast_cnt_reset_1	0		Output	hw_vio_1
> fast_vio_feedback[31:0]	[H] AA55_9966		Output	hw_vio_1
fast_vio_slice1a_fb	0		Output	hw_vio_1
> fast_vio_slice15_fb[14:0]	[H] 0000		Output	hw_vio_1

関連情報

[\[Debug Probes\] ウィンドウでの VIO コアの表示](#)

[VIO Probes] ウィンドウを使用した VIO 出力の書き込み

VIO 出力プローブは、VIO ダッシュボードの [VIO Probes] ウィンドウで設定できます。各出力プローブが表で個別の行に示されます。VIO 出力プローブの値は、[Value] 列に示されます (「VIO コアの出カプローブ」を参照)。VIO コアの出カ値は、[Value] 列に新しい値を入力するとアップデートされます。[Value] 列をクリックすると、プルダウン ボックスが表示されます。[Value] フィールドに適切な値を入力し、[OK] をクリックします。

Tcl コマンドを使用して、VIO コアに新しい値を書き込むこともできます。たとえば、基数が BINARY に設定されている `vio_slice5_fb_2` という VIO 出力プローブに 2 進数 11111 を書き込むには、次の Tcl コマンドを使用します。

```
set_property OUTPUT_VALUE 11111 [get_hw_probes vio_slice5_fb_2]
commit_hw_vio [get_hw_probes {vio_slice5_fb_2}]
```

関連情報

[VIO コアの出カプローブ](#)

VIO 出力の表示タイプと基数の設定

VIO 出力プローブの表示タイプは、[VIO Probes] の [VIO Dashboard] ウィンドウで VIO 出力プローブを右クリックし、次のいずれかをクリックすると設定できます。

- [Text]: 出力をテキスト フィールドとして表示します。これが、VIO 出力プローブ ベクター (幅が 2 ビット以上) の唯一の表示タイプです。
- [Toggle Button]: 出力をトグル ボタン グラフィックで表示します。この表示タイプは、VIO 出力プローブ スカラーおよび VIO 出力プローブ ベクターの個々のエレメントにのみ選択可能です。

VIO 出力プローブの表示タイプを [Text] に設定すると、基数を変更できます。これには、[Debug Probes] ウィンドウの [VIO Cores] ビューで VIO 出力プローブを右クリックし、次のいずれかを選択します。

- [Radix] → [Binary]: 基数を 2 進数に変更します。
- [Radix] → [Octal]: 基数を 8 進数に設定します。
- [Radix] → [Hex]: 基数を 16 進数に設定します。
- [Radix] → [Unsigned]: 基数を符号なしの 10 進数に設定します。
- [Radix] → [Signed]: 基数を符号付きの 10 進数に設定します。

VIO 出力プローブの基数は、Tcl コマンドでも設定できます。たとえば、`vio_slice5_fb_2` という VIO 出力プローブの基数を 16 進数に設定するには、次の Tcl コマンドを使用します。

```
set_property OUTPUT_VALUE_RADIX HEX [get_hw_probes vio_slice5_fb_2]
```

VIO コアの出カプローブのリセット

VIO v2.0 コアでは、各出力プローブ ポートの初期値を指定できます。VIO コアの出カプローブ ポートをこれらの初期値にリセットするには、[Hardware] ウィンドウで VIO コアを右クリックし、[Reset VIO Core Outputs] をクリックします。Tcl コマンドで VIO コアの出カをリセットするには、次のコマンドを使用します。

```
reset_hw_vio_outputs [get_hw_vios {hw_vio_1}]
```

注記: VIO 出力プローブを初期値にリセットすると、出力プローブの値が Vivado IDE と同期しなくなる可能性があります。この状況の対処方法は、「VIO コアの実出力値を Vivado IDE に同期」を参照してください。

関連情報

[VIO コアの実出力値を Vivado IDE に同期](#)

VIO コアの実出力値を Vivado IDE に同期

VIO 出力をリセット、FPGA を再プログラム、または現在のインスタンスが開始する前に Vivado ツールの別のインスタンスで出力値を設定すると、VIO コアの実出力プローブが Vivado IDE と同期しなくなります。この場合、VIO ステータスが「Outputs out-of-sync」となり、次のいずれかの操作を実行する必要があります。

- [Hardware] ウィンドウで VIO コアを右クリックして [Commit VIO Core Outputs] をクリックし、Vivado IDE から値を VIO コアに書き込みます。次の Tcl コマンドでも同じ操作を実行できます。

```
commit_hw_vio [get_hw_vios {hw_vio_1}]
```

- [Hardware] ウィンドウで VIO コアを右クリックして [Refresh Input and Output Values from VIO Core] をクリックし、Vivado IDE を VIO コア出力プローブ ポートの現在の値でアップデートします。次の Tcl コマンドでも同じ操作を実行できます。

```
refresh_hw_vio -update_output_values 1 [get_hw_vios {hw_vio_1}]
```

JTAG-to-AXI Master デバッグ コアを使用したハードウェア システム通信

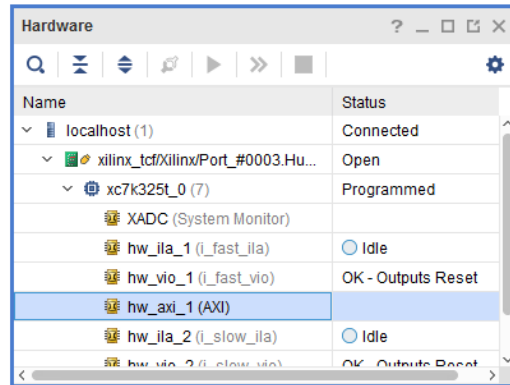
JTAG-to-AXI Master デバッグ コアはカスタマイズ可能なコアで、AXI トランザクションを生成し、ランタイム時に AXI 信号を FPGA 内部に駆動できます。このコアは、すべてのメモリ マップされた AXI および AXI4-Lite インターフェイスをサポートし、32 または 64 ビットのデータ幅のインターフェイスをサポートできます。

デザインに追加した JTAG-to-AXI Master (JTAG-AXI) コアは、[Hardware] ウィンドウのターゲット デバイスの下に表示されます。JTAG-AXI コアが表示されていない場合は、デバイスを右クリックして [Refresh Hardware] をクリックします。FPGA デバイスが再度スキャンされ、[Hardware] ウィンドウの表示が更新されます。

注記: FPGA デバイスをプログラムまたは更新しても ILA コアが表示されない場合は、デバイスが正しい .bit ファイルでプログラムされているか、インプリメント済みデザインに ILA コアが含まれているかを確認してください。

JTAG-AXI コア (次の図の hw_axi_1) をクリックすると、[AXI Core Properties] ウィンドウにプロパティが表示されます。

図 148: [Hardware] ウィンドウに表示される JTAG-to-AXI Master コア



ハードウェアの JTAG-to-AXI Master デバッグ コアへのアクセス

JTAG-to-AXI Master デバッグ コアは、Tcl コマンドを使用してのみアクセスできます。AXI 読み出しおよび書き込み トランザクションの作成と実行には、それぞれ `create_hw_axi_txn` および `run_hw_axi` コマンドを使用します。

JTAG-to-AXI Master デバッグ コアのリセット

トランザクションを作成して実行する前に、次の Tcl コマンドを使用して JTAG-to-AXI Master コアをリセットしておく必要があります。

```
reset_hw_axi [get_hw_axis hw_axi_1]
```

読み出し トランザクションの作成と実行

AXI トランザクションの作成に使用する Tcl コマンドは、`create_hw_axi_txn` です。このコマンドの使用方法は、Vivado IDE の [Tcl Console] ウィンドウに「`help create_hw_axi_txn`」と入力すると表示されます。次に、アドレス 0 からの 4 ワードの AXI 読み出しバースト トランザクションを作成する例を示します。

```
create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -type READ -address 00000000 -len 4
```

説明:

- `read_txn`: トランザクションのユーザー定義名
- `[get_hw_axis hw_axi_1]`: `hw_axi_1` オブジェクトを返す
- `-address 00000000`: 開始アドレス
- `-len 4`: AXI バースト長を 4 ワードに設定

この後、`run_hw_axi` コマンドを使用して作成したトランザクションを実行します。次にその例を示します。

```
run_hw_axi [get_hw_axi_txns read_txn]
```

最後に、読み出されたデータをトランザクションを実行した結果として取得します。`report_hw_axi_txn` または `report_property` コマンドのいずれかを使用するとデータを画面に表示でき、`get_property` コマンドを使用するとその戻り値をほかの箇所で使用できるようになります。

```
report_hw_axi_txn [get_hw_axi_txns read_txn]
0 00000000 00000000
8 00000000 00000000
report_property [get_hw_axi_txns read_txn]
Property Type Read-only Visible Value
CLASS string true true hw_axi_txn
CMD.ADDR string false true 00000000
CMD.BURST enum false true INCR
CMD.CACHE int false true 3
CMD.ID int false true 0
CMD.LEN int false true 4
CMD.SIZE enum false true 32
DATA string false true 00000000000000000000000000000000
HW_AXI string true true hw_axi_1
NAME string true true read_txn
TYPE enum false true READ
```

書き込みトランザクションの作成と実行

次に、アドレス 0 からの 4 ワードの AXI 書き込みバースト トランザクションを作成する例を示します。

```
create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -type WRITE -address
00000000 \
-len 4 -data {11111111_22222222_33333333_44444444}
```

ここで

- `write_txn`: トランザクションのユーザー定義名
- `[get_hw_axis hw_axi_1]`: `hw_axi_1` オブジェクトを返す
- `-address 00000000`: 開始アドレス
- `-len 4`: AXI バースト長を 4 ワードに設定
- `-data {11111111_22222222_33333333_44444444}`: LSB (例: address 0) から MSB (例: address 3) に向かって指定します。

この後、`run_hw_axi` コマンドを使用して作成したトランザクションを実行します。次にその例を示します。

```
run_hw_axi [get_hw_axi_txns write_txn]
```



重要: デバイスを再プログラムすると、既存のすべての `jtag_axi` トランザクションは削除されます。これらのトランザクションを再作成する必要がある場合があります。



ヒント: `run_hw_axi Tcl` コマンドで `-queue` オプションを使用すると、キュー モードの `hw_axi` トランザクションを指定できます。キュー モードでは、JTAG to AXI Master FIFO に 16 個の読み出しトランザクションと 16 個の書き込みトランザクションを待機させ、トランザクション間のレイテンシを短くし、パフォーマンスを向上するため、連続発行できます。キューに挿入しないトランザクションは、送信されると同時に実行されます。

ラボ環境での Vivado ロジック解析の使用

Vivado ロジック解析機能は Vivado IDE および Vivado Lab Edition に含まれています。ラボ環境で Vivado ロジック解析機能を使用してターゲット ボード上で実行されているデザインをデバッグするには、次のいずれかを実行します。

- ラボ マシンに Vivado Lab Edition をインストールして実行します。詳細は、「Vivado Lab Edition」を参照してください。
- ラボ マシンにフル Vivado IDE をインストールして実行します。
- リモート ラボ マシンに最新版の Vivado Design Suite または Vivado ハードウェア サーバー (スタンドアロン) をインストールし、ローカル マシンの Vivado ロジック解析機能を使用して Vivado ハードウェア サーバー (hw_server) のリモート インスタンスに接続します。

関連情報

[Vivado Lab Edition](#)

ラボ マシンで動作中の hw_server サーバーへの接続

ラボ マシンへのネットワーク接続がある場合、リモート ラボ マシン上で動作中のハードウェア サーバーに接続してターゲット ボードに接続できます。Vivado ロジック解析機能を使用してラボ マシンで動作中の Vivado ハードウェア サーバー (Windows プラットフォームでは hw_server.bat、Linux プラットフォームでは hw_server) に接続するには、次の手順に従います。

1. ラボ マシンに最新版の Vivado Design Suite または Vivado ハードウェア サーバー (スタンドアロン) をインストールします。



重要: リモート ハードウェア サーバー機能のみを使用する場合は、ラボ マシンに Vivado Design Suite のフル バージョンまたは Vivado Lab Edition をインストールする必要はありません。ただし、ラボ マシンで Vivado ロジック解析や Vivado シリアル I/O 解析などの Vivado ハードウェア マネージャーの機能を使用する場合は、ラボ マシンに Vivado Lab Edition をインストールする必要があります。ハードウェア サーバーまたは Vivado Design Suite または Vivado Lab Edition のハードウェア マネージャー機能を実行するのにソフトウェア ライセンスは必要ありません。

2. リモート ラボ マシンで hw_server アプリケーションを起動します。ラボ マシンが 64 ビット Windows マシンで、Vivado ハードウェア サーバー (スタンドアロン) がデフォルトの場所にインストールされている場合、次のコマンドを使用します。

```
C:\Xilinx\VivadoHWSRV\vivado_release.version\bin\hw_server.bat
```

3. ラボ マシン以外のマシンで Vivado IDE を GUI モードで起動します。
4. 「ハードウェア ターゲットに接続してデバイスをプログラム」の手順に従って、ラボ マシンに接続されているターゲット ボードへの接続を開きます。ただしここでは、localhost 上の Vivado CSE サーバーに接続するのではなく、ラボ マシンのホスト名を使用します。
5. 「計測のための ILA コアの設定」以降の手順に従って、ハードウェア上でデザインをデバッグします。

関連情報

[ハードウェア ターゲットに接続してデバイスをプログラム](#)

[計測のための ILA コアの設定](#)

ハードウェア マネージャーの Tcl オブジェクトおよびコマンド

テスト中のハードウェアにアクセスするには、Tcl コマンドを使用できます。ハードウェアには、次の図に示す階層ファースト クラス Tcl オブジェクトがあります。

表 15: ハードウェア マネージャーの Tcl オブジェクト

Tcl オブジェクト	説明
hw_server	ハードウェア サーバーを参照するオブジェクト。各 hw_server オブジェクトには、1 つまたは複数の hw_target オブジェクトを関連付けることができます。
hw_target	JTAG ケーブルまたはボードを参照するオブジェクト。各 hw_target オブジェクトには、1 つまたは複数の hw_device オブジェクトを関連付けることができます。
hw_device	ザイリンクス FPGA デバイスなど、JTAG チェーンに含まれるデバイスを参照するオブジェクト。各 hw_device オブジェクトには、1 つまたは複数の hw_ila オブジェクトを関連付けることができます。
hw_ila	ザイリンクス FPGA デバイスに含まれる ILA コアを参照するオブジェクト。各 hw_ila オブジェクトに関連付けることができる hw_ila_data オブジェクトは 1 つのみです。また、各 hw_ila オブジェクトには 1 つまたは複数の hw_probe オブジェクトを関連付けることができます。
hw_ila_data	ILA デバッグ コアからアップロードされたデータを参照するオブジェクト。
hw_probe	ILA デバッグ コアのプロブ入力を参照するオブジェクト。
hw_vio	ザイリンクス FPGA デバイスに含まれる VIO コアを参照するオブジェクト。

ハードウェア マネージャー コマンドの詳細は、[Tcl Console] ウィンドウに「help -category hardware」と入力してください。

hw_server の Tcl コマンド

次の表に、ハードウェア サーバーに関連する Tcl コマンドすべての説明を含めます。

表 16: hw_server Tcl コマンドの説明

Tcl コマンド	説明
connect_hw_server	ハードウェア サーバーへの接続を開きます。
current_hw_server	現在のハードウェア サーバーを取得または設定します。
disconnect_hw_server	ハードウェア サーバーへの接続を閉じます。
get_hw_servers	ハードウェア サーバーのハードウェア サーバー名のリストを取得します。
refresh_hw_server	ハードウェア サーバーへの接続を更新します。

hw_target の Tcl コマンド

次の表に、ハードウェア ターゲットに関連する Tcl コマンドすべての説明を含めます。

表 17: hw_target の Tcl コマンド

Tcl コマンド	説明
close_hw_target	ハードウェア ターゲットを閉じます。
current_hw_target	現在のハードウェア ターゲットを取得または設定します。
get_hw_targets	ハードウェア サーバーのハードウェア ターゲット名のリストを取得します。
open_hw_target	ハードウェア サーバー上のハードウェア ターゲットへの接続を開きます。
refresh_hw_target	ハードウェア ターゲットへの接続を更新します。

hw_device の Tcl コマンド

次の表に、Tcl コマンド hw_device の説明と、ハードウェア デバイスにアクセスするために使用する Tcl コマンドすべての説明を示します。

表 18: hw_device の Tcl コマンド

Tcl コマンド	説明
current_hw_device	現在のハードウェア デバイスを取得または設定します。
get_hw_devices	ターゲットのハードウェア デバイスのリストを取得します。
program_hw_device	ザイリンクス FPGA デバイスをプログラムします。
refresh_hw_device	ハードウェア デバイスを更新します。

hw_ila の Tcl コマンド

次の表に、ILA デバッグ コアにアクセスするために使用する Tcl コマンドを示します。

表 19: hw_ila の Tcl コマンド

Tcl コマンド	説明
current_hw_ila	現在のハードウェア ILA を取得または設定します。
get_hw_ilas	ターゲットのハードウェア ILA のリストを取得します。
reset_hw_ila	hw_ila の制御プロパティをデフォルト値にリセットします。
run_hw_ila	hw_ila をトリガー待機状態にします。
wait_on_hw_ila	すべてのデータがキャプチャされるまで待機します。

hw_ila_data の Tcl コマンド

次の表に、Tcl コマンド hw_ila_data の説明と、キャプチャされた ILA デバッグ コアにアクセスするために使用する Tcl コマンドすべての説明を示します。

表 20: hw_ila_data の Tcl コマンド

Tcl コマンド	説明
current_hw_ila_data	現在のハードウェア ILA データを取得または設定します。
display_hw_ila_data	hw_ila_data を波形ビューアーに表示します。

表 20: hw_ila_data の Tcl コマンド (続き)

Tcl コマンド	説明
get_hw_ila_data	hw_ila_data オブジェクトのリストを取得します。
list_hw_samples	1 つのハードウェア プロープに関連付けられているデータ サンプルをリストします。
read_hw_ila_data	ファイルから hw_ila_data を読み出します。
upload_hw_ila_data	ILA コアでのデータのキャプチャを停止し、キャプチャされたデータをアップロードします。
write_hw_ila_data	hw_ila_data をファイルに記述します。

hw_probe の Tcl コマンド

次の表に、キャプチャされた ILA データにアクセスするために使用する Tcl コマンドを示します。

表 21: hw_probe の Tcl コマンド

Tcl コマンド	説明
create_hw_probe	物理的な ILA プロープ ポートおよび定数値から新しいハードウェア プロープを作成します。
delete_hw_probe	create_hw_probe コマンドを使用して作成したユーザー定義ハードウェア プロープを削除します。
get_hw_probes	ハードウェア プロープのリストを取得します。

hw_vio の Tcl コマンド

次の表に、VIO コアにアクセスするために使用する Tcl コマンドを示します。

表 22: hw_vio の Tcl コマンド

Tcl コマンド	説明
commit_hw_vio	hw_probe の OUTPUT_VALUE プロパティの値を VIO コアに書き込みます。
get_hw_vios	hw_vio のリストを取得します。
refresh_hw_vio	hw_vio の INPUT_VALUE および ACTIVITY_VALUE プロパティを VIO コアから読み出した値でアップデートします。
reset_hw_vio_activity	指定の hw_vio オブジェクトに関連付けられている hw_probe に対し、ACTIVITY_VALUE プロパティをリセットします。
reset_hw_vio_outputs	VIO コアの出力を初期値にリセットします。

hw_axi および hw_axi_txn の Tcl コマンド

次の表に、JTAG-to-AXI Master コアにアクセスするために使用する Tcl コマンドを示します。

表 23: hw_axi および hw_axi_txn の Tcl コマンド

Tcl コマンド	説明
create_hw_axi_txn	ハードウェア AXI トランザクション オブジェクトを作成します。
delete_hw_axi_txn	ハードウェア AXI トランザクション オブジェクトを削除します。

表 23: hw_axi および hw_axi_txn の Tcl コマンド (続き)

Tcl コマンド	説明
get_hw_axi_txns	ハードウェア AXI トランザクション オブジェクトのリストを取得します。
get_hw_axis	ハードウェア AXI オブジェクトのリストを取得します。
refresh_hw_axi	ハードウェア AXI オブジェクトのステータスを更新します。
report_hw_axi_txn	フォーマットされたハードウェア AXI トランザクション データをレポートします。
reset_hw_axi	ハードウェア AXI コアのステートをリセットします。
run_hw_axi	ハードウェア AXI 読み出し/書き込みトランザクションを実行し、対応する hw_axi オブジェクトのトランザクション ステータスをアップデートします。

hw_sysmon の Tcl コマンド

次の表に、システム モニター コアにアクセスするために使用する Tcl コマンドを示します。

表 24: hw_sysmon の Tcl コマンド

Tcl コマンド	説明
commit_hw_sysmon	hw_sysmon オブジェクトに定義された現在のプロパティ 値をハードウェア デバイスのシステム モニター レジスタへ入れます。
get_hw_sysmon_reg	指定した hw_sysmon オブジェクトの指定アドレスに定義されたシステム モニター レジスタの 16 進数値を返します。
get_hw_sysmons	現在のハードウェア デバイスで定義されたシステム モニター デバッグ コア オブジェクトを返します。
refresh_hw_sysmon	hw_sysmon オブジェクトのプロパティを現在の hw_device からのシステム モニターの値で更新します。
set_hw_sysmon_reg	指定したアドレスのシステム モニター レジスタを指定した 16 進数値に設定します。

注記: これらの各コマンドの詳細なヘルプは、Vivado の [Tcl Console] ウィンドウに「<command name> -help」と入力すると取得できます。

Tcl コマンドを使用した JTAG-to-AXI Master コアへのアクセス

次のようなシステムにアクセスする Tcl コマンド スクリプトの例を示します。

- localhost:3121 上の Vivado hw_server を介してアクセス可能な 1 つの KC705 ボードの Digilent JTAG-SMT1 ケーブル (シリアル番号 12345) が使用されている。
- KC705 ボード上の XC7K325T デバイスで実行されるデザインに JTAG-to-AXI Master コアが 1 つ含まれている。
- JTAG-to-AXI Master コアが AXI BRAM Controller Slave コアを含む AXI ベース システムに含まれている。

Tcl コマンド スクリプト例

```
# Connect to the Digilent Cable on localhost:3121
connect_hw_server -url localhost:3121
current_hw_target [get_hw_targets */xilinx-tcf/Digilent/12345]
open_hw_target
# Program and Refresh the XC7K325T Device
current_hw_device [lindex [get_hw_devices] 0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices] 0]
set_property PROGRAM.FILE {C:/design.bit} [lindex [get_hw_devices] 0]
set_property PROBES.FILE {C:/design.ltx} [lindex [get_hw_devices] 0]
program_hw_devices [lindex [get_hw_devices] 0]
refresh_hw_device [lindex [get_hw_devices] 0]
# Reset the JTAG-to-AXI Master core
reset_hw_axi [get_hw_axis hw_axi_1]
# Create a read transaction bursts 128 words starting from address 0
create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -type read \
-address 00000000 -len 128
# Create a write transaction bursts 128 words starting at address 0
# using a repeating fill value of 11111111-22222222-33333333-44444444
# (where LSB is to the left)
create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -type write \
-address 00000000 -len 128 -data {11111111-22222222-33333333-44444444}
# Run the write transaction
run_hw_axi [get_hw_axi_txns wrte_txn]
# Run the read transaction
run_hw_axi [get_hw_axi_txns read_txn]
```

ILA を測定する Tcl コマンドの使用

次のようなシステムにアクセスする Tcl コマンド スクリプトの例を示します。

- localhost:3121 上の Vivado CSE サーバーを介してアクセス可能な 1 つの KC705 ボードの Digilent JTAG-SMT1 ケーブル (シリアル番号 12345) が使用されている。
- KC705 ボード上の XC7K325T デバイスで実行されているデザインに ILA コアが 1 つ含まれている。
- ILA コアに counter[3:0] というプローブが含まれている。

Tcl コマンド スクリプト例

```
# Connect to the Digilent Cable on localhost:3121
connect_hw_server -url localhost:3121
current_hw_target [get_hw_targets */xilinx-tcf/Digilent/12345]
open_hw_target
# Program and Refresh the XC7K325T Device
current_hw_device [lindex [get_hw_devices] 0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices] 0]
set_property PROGRAM.FILE {C:/design.bit} [lindex [get_hw_devices] 0]
set_property PROBES.FILE {C:/design.ltx} [lindex [get_hw_devices] 0]
program_hw_devices [lindex [get_hw_devices] 0]
refresh_hw_device [lindex [get_hw_devices] 0]
# Set Up ILA Core Trigger Position and Probe Compare Values
set_property CONTROL.TRIGGER_POSITION 512 [get_hw_ilas hw_ila_1]
set_property COMPARE_VALUE.0 eq4'b0000 [get_hw_probes counter]
# Arm the ILA trigger and wait for it to finish capturing data
```

```
run_hw_ila hw_ila_1
wait_on_hw_ila hw_ila_1
# Upload the captured ILA data, display it, and write it to a file
current_hw_ila_data [upload_hw_ila_data hw_ila_1]
display_hw_ila_data [current_hw_ila_data]
write_hw_ila_data my_hw_ila_data [current_hw_ila_data]
```

スタートアップ時のトリガー

スタートアップ時のトリガーは、デバイス スタートアップ直後にトリガーされるように、デザインの .bit ファイルの ILA コアのトリガー設定をコンフィギュレーションするために使用されます。これには、ハードウェアのデザインで通常実行される ILA コアに適用されたさまざまなトリガー設定を、インプリメントされたデザインの ILA コアに適用します。



重要: スタートアップ時のトリガーを使用した次のプロセスでは、ハードウェアで動作する有効な ILA デザインがあり、合成フローで ILA コアがフラット化されていないことを想定しています。

スタートアップ時のトリガー機能を使用するには、次の手順に従います。

- 1 回目の ILA フローを通常どおり実行し、トリガー条件を設定します。
 - a. ターゲットを開いてデバイスをコンフィギュレーションし、ILA ダッシュボードを立ち上げます。
 - b. ILA ダッシュボードで ILA コアに対するトリガー式を入力します。
2. Vivado の Tcl コマンド ラインから、ILA コアのトリガー レジスタ マップ ファイルをエクスポートします。このファイルには、インプリメント後のネットリストにすべてのレジスタ設定を戻すための情報が含まれます。この出力は、1 つのファイルです。

```
% run_hw_ila -file ila_trig.tas [get_hw_ilas hw_ila_1]
```

3. Vivado IDE に戻って前にインプリメントした配線済みデザインを開きます。これを実行する方法は、プロジェクト フローによって異なります。
 - a. プロジェクト モード: Flow Navigator を使用してインプリメント済みデザインを開きます。
 - b. 非プロジェクト モード: 配線済みのチェックポイント %open_checkpoint < ファイルを開きます。
>.dcp
4. [Tcl Console] ウィンドウでメモリ内の現在のデザイン (配線済みネットリスト) にトリガー設定を適用します。

```
%apply_hw_ila_trigger ila_trig.tas
```

注記: ILA コアが合成中にフラット化されていることを示すエラー メッセージが表示されたら、デザインを生成し直して、合成で ILA コアの階層が保持されるようにします。ハードウェアで動作する有効な ILA デザインが含まれており、ILA コアが合成フロー中にフラット化されないようにしてください。

5. [Tcl Console] ウィンドウで、スタートアップ時のトリガー設定を使用してビットストリームを書き出します。



重要: 配線済みデザインの変更を認識させるには、write_bitstream trig_at_startup.bit を Tcl コマンド コンソールで実行する必要があります。

6. ハードウェア マネージャー [Hardware Manager] に戻り、前の手順で生成した新しい .bit ファイルを使用してリコンフィギュレーションします。アップデートされた .bit ファイル ディレクトリのプロパティを GUI か Tcl コマンドのいずれかから設定する必要があります。ハードウェア ツールでのコンフィギュレーションで新しい .bit ファイルが使用されるように設定してください。
 - a. ハードウェア ツリーでデバイスを選択します。

- b. 手順 5 で生成した .bit ファイルを割り当てます。
7. 新しい .bit ファイルを使用してデバイスをプログラムします。

プログラムすると、新しい ILA コアはスタートアップ時にすぐにトリガー待機状態になります。ILA コアのトリガー キャプチャ ステータスに、その状態が表示されます。これで、トリガーまたはキャプチャ イベントが発生したときに、ILA コアにキャプチャされたデータ サンプルが含まれるようになります。

メモリ キャリブレーションのデバッグ

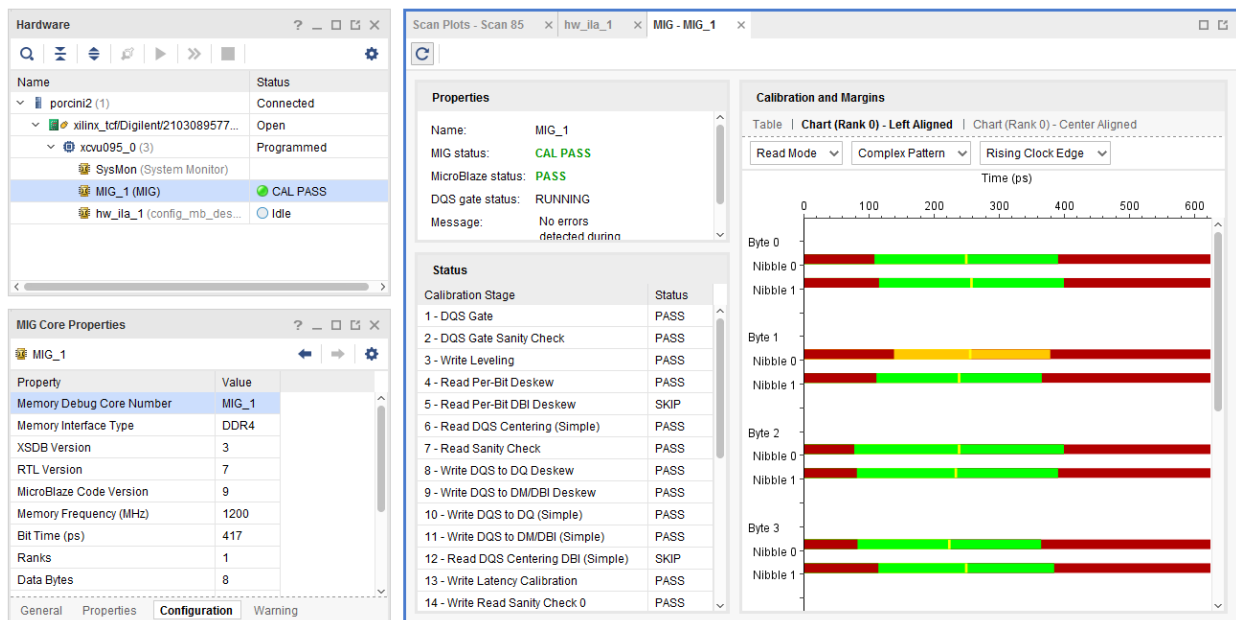
Vivado のメモリ インターフェイス IP では、キャリブレーション デバッグがサポートされます。これらは、Vivado ハードウェア マネージャーからアクセス可能なコアの設定、キャリブレーション、およびデータ ウィンドウ情報を格納します。メモリ キャリブレーションではこの情報をいつでも読み出して、メモリ インターフェイス IP からの有益な統計およびフィードバックを取得できます。情報は、Vivado ハードウェア マネージャーのメモリ キャリブレーション デバッグの GUI またはメモリ キャリブレーション デバッグの Tcl コマンドを使用して表示できます。

メモリ キャリブレーション デバッグの GUI 使用方法

デバイスをコンフィギュレーションすると、Vivado ハードウェア マネージャーにメモリ インターフェイスが表示されます。

デバッグ インターフェイスに表示されるメモリ キャリブレーションは、キャリブレーション ステータスをすばやく特定し、ウィンドウ マージンを読み出したり書き込んだりするために使用できます。このデバッグ インターフェイスは、生成されたメモリ インターフェイス (UltraScale および UltraScale+) デザインに常に含まれます。

図 149: メモリ キャリブレーション デバッグのインターフェイス



メモリ キャリブレーション デバッグの Tcl 使用方法

Vivado ハードウェア マネージャーでハードウェアに接続したら、Vivado の [Tcl Console] ウィンドウで次のコマンドを使用して、すべてのメモリ キャリブレーション デバッグの内容を Vivado IDE に表示できます。

- `get_hw_migs`
 - 。 デザインに存在するメモリ インターフェイスを表示します。
- `refresh_hw_mig [lindex [get_hw_migs] 0]`
 - 。 指定したインデックス番号のメモリ インターフェイスのみを更新します (インデックスは 0 から開始)。
- `report_propery[lindex [get_hw_migs] 0]`
 - 。 メモリ インターフェイスのすべてのパラメーターをレポートします。
 - 。 0 は、レポートするメモリ インターフェイスのインデックスです (インデックスは 0 から開始)。

詳細は、次の資料の UltraScale または 7 シリーズ メモリ キャリブレーション デバッグ コマンドを参照してください。

- [ザイリンクス アンサー 43879](#): MIG 7 Series DDR3/DDR2 - ハードウェア デバッグ ガイド
- 『UltraScale アーキテクチャ FPGA メモリ IP LogiCORE IP 製品ガイド』 (PG150: [英語版](#)、[日本語版](#))

Vivado ハードウェア マネージャーでのパーシャル リコンフィギュラブル デザインのデバッグ

Vivado® ハードウェア マネージャーでは、パーシャル リコンフィギュラブル デザインのデバッグがサポートされます。このようなデザインを問題なくデバッグするには、パーシャル ビットストリームをプログラムする前にフル デザイン ビットストリームをプログラムして、特定のリコンフィギュラブル モジュールを置換する必要があります。

Dynamic Function eXchange デザインにデバッグ コアをインスタンス化し、例および Vivado ハードウェア マネージャー内の機能の詳細は、『Vivado Design Suite チュートリアル: Dynamic Function eXchange』 ([UG947](#)) の [このセクション](#)を参照してください。

HBM (High Bandwidth Memory) モニター

一部の Virtex® UltraScale+ FPGA には、HBM (High Bandwidth Memory) コントローラーおよびメモリ スタックが含まれています。統合 HBM コントローラーおよびメモリ スタックには、パフォーマンス カウンターと温度センサーが含まれています。HBM モニターは、HBM ダイ上のパフォーマンス モニターおよび温度センサーにリアルタイムにアクセスし、情報を取得およびエクスポートするために使用できます。

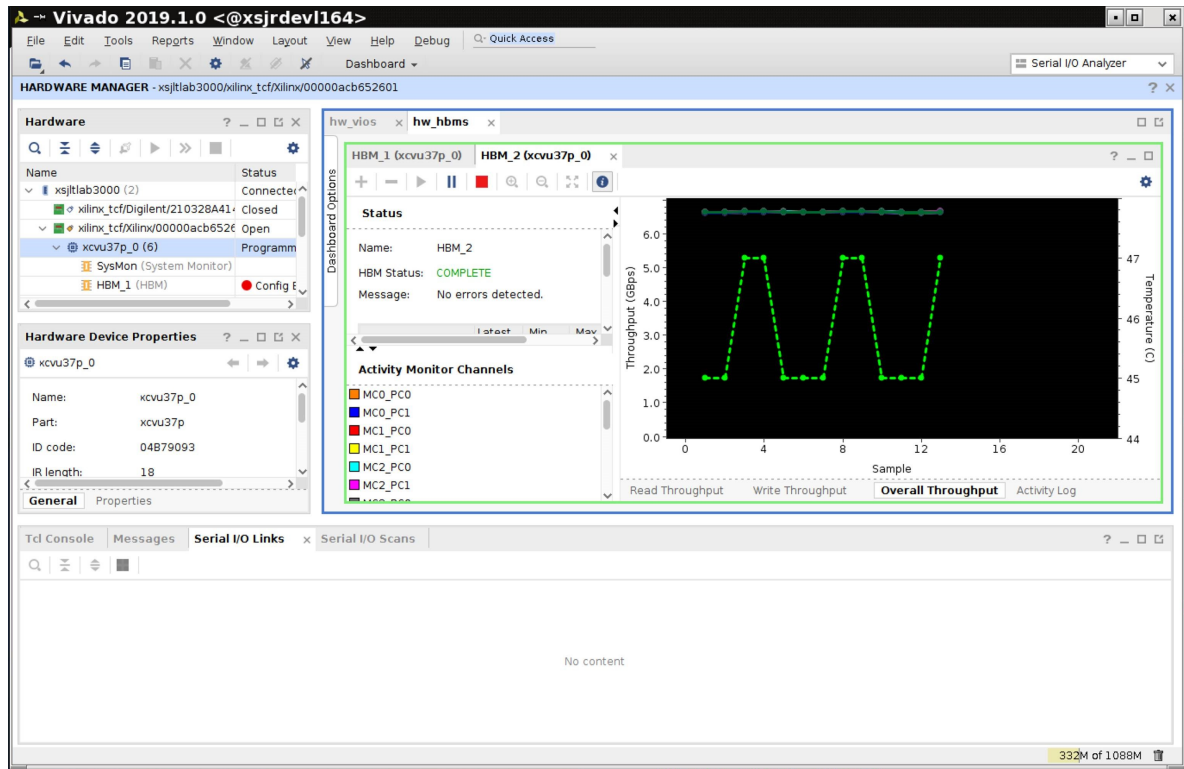
HBM モニターの GUI での使用

HBM をイネーブルにしたデバイスを AXI High Bandwidth Memory Controller を含むデザインでコンフィギュレーションすると、Vivado ハードウェア マネージャーに HBM インターフェイスが表示されます。

HBM モニターのサポートは、生成された High Bandwidth Memory Controller に常に含まれます。HBM モニターには、スタックの温度、読み出しと書き込みのスループット、および全体的なスループットが表示されます。

収集されたデータは、その後の処理または解析用に CSV (Comma Separated Value) フォーマットのテキスト ファイルにエクスポートできます。

図 150: リアルタイムのパフォーマンス表示



HBM モニターの Tcl での使用

Vivado ハードウェア マネージャーでハードウェアに接続したら、Vivado の [Tcl Console] ウィンドウで次のコマンドを使用して、HBM モニターにアクセスします。

- `get_hw_hbms`: デザインに含まれる HBM インターフェイスのリストを表示します。
- `refresh_hw_hbm [lindex [get_hw_hbms] 0]`: 指定したハードウェア HBM のステータスを更新します。
- `report_property [lindex [get_hw_hbms] 0]`: 指定した HBM インターフェイスで使用可能なすべてのパラメータをレポートします。
- `run_hw_hbm_amon [lindex [get_hw_hbms] 0]`: 指定したハードウェア HBM に対してアクティビティ モニターの実行をイネーブルにします。
- `stop_hw_hbm_amon [lindex [get_hw_hbms] 0]`: 指定したハードウェア HBM に対してアクティビティ モニターの実行をディスエーブルにします。

詳細および例は『AXI High Bandwidth Controller LogiCORE IP 製品ガイド』(PG276)の付録 D を参照してください。

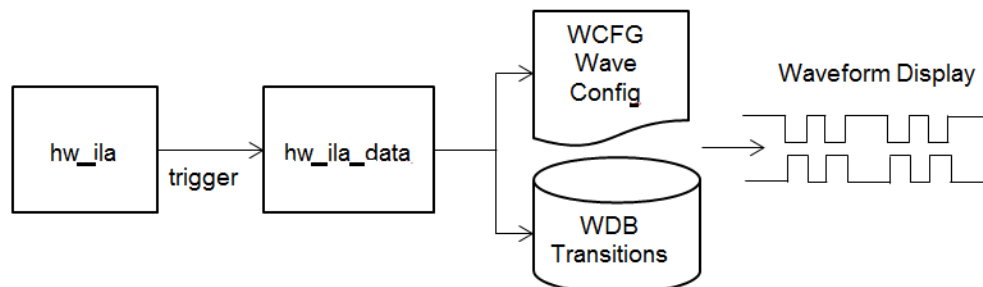
波形ビューアーを使用した ILA プローブデータの表示

Vivado® 統合設計環境 (IDE) の ILA 波形ビューアーを使用すると、ILA デバッグ コアからキャプチャされたデータを解析できます。ILA コアをトリガーしてデータをキャプチャすると、ILA コアから取得されたデータが対応する波形ビューアーに表示されます。Vivado をプロジェクト モードで使用している場合、色、基数、信号の順序などの設定可能な波形設定は保持され、その後の Vivado セッションでも使用されます。

ILA データと波形の関係

キャプチャされた ハードウェア ILA データ オブジェクト (hw_ila_data) と波形の関係を理解しておくとは有益です。この関係を次の図に示します。

図 151: ILA データと波形の関係



ハードウェア ILA Tcl オブジェクト (hw_ila) は、ハードウェア内の ILA を示します。ILA コアがキャプチャされたデータをアップロードするたびに、対応する hw_ila_data Tcl オブジェクトに保存されます。これらのオブジェクトにはわかりやすい名前が付けられ、ハードウェアの最初の ILA コア hw_ila_1 に対応する Tcl データ オブジェクトの名前は hw_ila_data_1 となります。オンラインでハードウェアを操作する際、前の図に示すように、各波形はメモリ内の hw_ila_data オブジェクトが基になっており、1 対 1 で対応しています。各 hw_ila_data Tcl ハードウェア ILA データ オブジェクトに対して、波形データベース (WDB) ファイルと波形コンフィギュレーション (WCFG) ファイルが作成され、Vivado プロジェクトのディレクトリに自動的に保存されます。前の図には、左側の hw_ila から右側の波形表示へのデータフローが示されています。

波形コンフィギュレーション (WCFG) ファイルと波形データベース (WDB) ファイルの組み合わせに、波形データベースと Vivado 波形ビューアーでの表示のカスタマイズ情報が含まれます。これらの波形ファイルは Vivado ILA フローで自動的に管理されるので、WDB または WCFG ファイルを直接変更しないでください。波形コンフィギュレーションを変更するには、波形ビューアーのオブジェクト (信号の色、バスの基数、信号の順序、マーカーなど) を変更します。これにより波形コンフィギュレーションの変更が Vivado プロジェクトの適切な WCFG ファイルに保存されます。

波形コンフィギュレーションおよびデータを後で表示するためにアーカイブするには、Tcl コマンド `write_hw_ila_data` を使用します。これにより、`hw_ila_data`、波形データベース、および波形コンフィギュレーションがアーカイブに保存されます。オフライン ストレージおよび波形の復元のために `read_hw_ila_data` および `write_hw_ila_data` を使用する方法については、「ILA コアでキャプチャされたデータの保存および復元」セクションを参照してください。

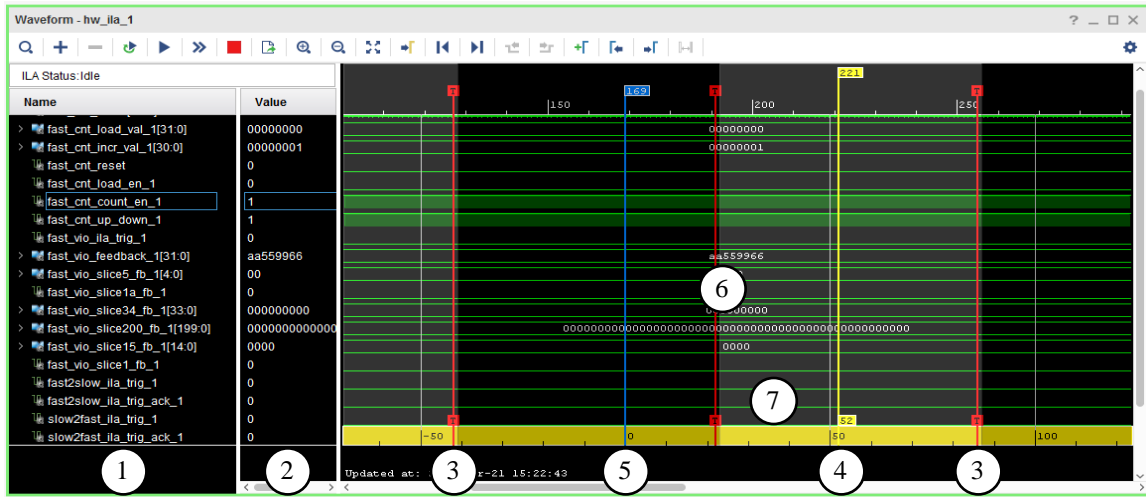
関連情報

[ILA コアでキャプチャされたデータの保存および復元](#)

波形ビューアーのレイアウト

ILA 波形ビューアー (波形コンフィギュレーションとも呼ばれる) は複数のダイナミック オブジェクトで構成されており、これらによりキャプチャされた ILA データが表示されます。

図 152: キャプチャされた ILA データを表示する波形ビューアー



X18959-032717

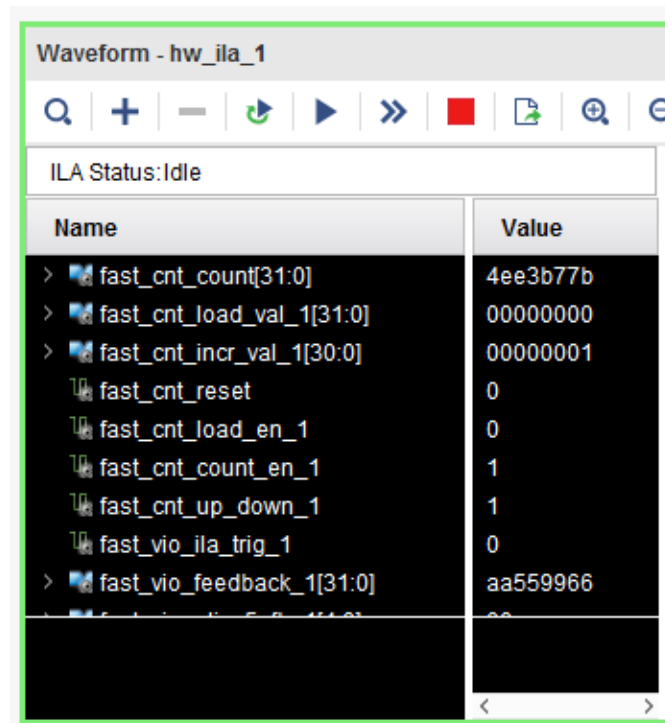
前の図に示す各要素は、次のとおりです。

1. ILA プローブ ファイル (.ltx) からのネット名またはバス名
2. カーソル位置でのネットまたはバスの値
3. トリガー マーカー (赤線)
4. カーソル (黄色の線)
5. マーカー (青線)
6. ILA キャプチャ ウィンドウの遷移 (クリア部分とグレー部分を交互に表示)
7. フロート計測ルーラー (黄色のバー)

波形ビューアーの操作

波形ビューアーの [Name] 列に示されるスカラーおよびバスは、波形のプローブ デザイン オブジェクトの名前です。これらは、ILA コアのハードウェア プローブに対応します (get_hw_probes Tcl コマンドを参照)。

図 153: 波形ビューアーに示される ILA プローブの名前と値



ILA データを初めてトリガーしてアップデートすると、波形ビューアーに ILA コアに接続されているすべてのプローブが表示されます。ビューアーに示されるプローブはカスタマイズ可能で、既存のプローブを削除したり新しいプローブを追加したりできます。このセクションでは、波形ビューアーの基本的な操作を説明します。

波形からのプローブの削除

初回のトリガーおよびアップロード操作により、すべてのプローブがデフォルトで波形に追加されます。波形にすべてのプローブを表示する必要がない場合は、ビューアーからプローブを削除できます。

波形ビューアーからプローブを削除するには [Name] 列で削除するスカラーまたはバスを右クリックし、[Delete] をクリックします。または、信号またはバスを選択して Delete キーを押しても削除できます。この操作によりプローブの遷移データがメモリから削除されるわけではなく、ビューでプローブが非表示になるだけです。

波形へのプローブの追加

波形にプローブを追加するには、[Debug Probes] ウィンドウで関連の ILA コアに追加するプローブを右クリックし、[Add Probes to Waveform] をクリックします。

信号またはバスのコピーを追加するには、[Waveform] ウィンドウで信号またはバスを選択し、[Edit]→[Copy] をクリックするか Ctrl+C キーを押してクリップボードにコピーして、[Edit]→[Paste] をクリックするか Ctrl+V キーを押して波形にオブジェクトのコピーを貼り付けます。

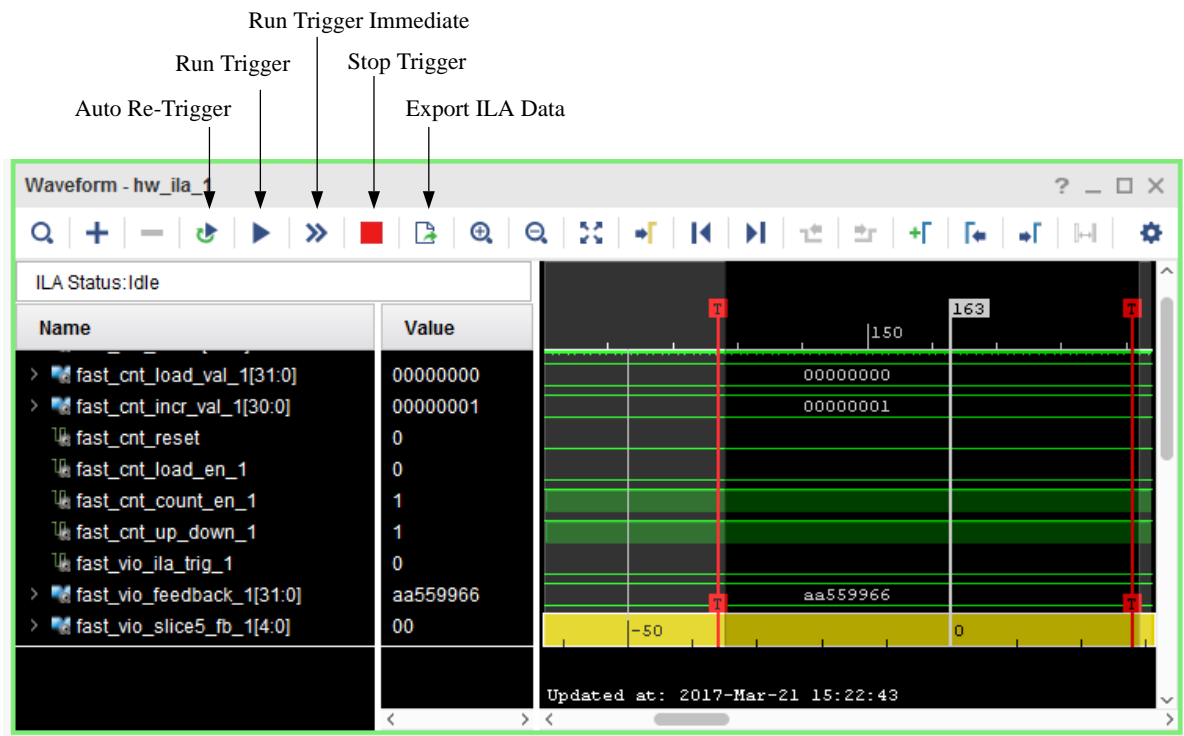
同じことは次のように Tcl コマンド `add_wave` を使用してできます。

```
add_wave -into {hw_ila_data_1.wcfg} -radix hex { {counter1} }
```

この例では、`counter1` が `hw_ila_1` の [Waveform Configuration] ウィンドウに追加され、[Waveform] ウィンドウの表示基数が `hex` に設定されます。

波形 ILA トリガーおよびエクスポート機能の使用

図 154: 波形 ILA トリガーおよびエクスポート機能



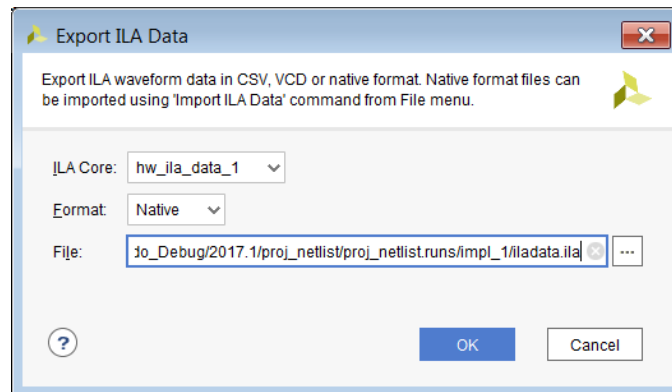
X16755-032717

- [Enable Auto Re-Trigger]: [Waveform] ウィンドウの [Enable Auto Re-Trigger] ツールバー ボタンをクリックすると、Vivado IDE でトリガー、アップロード、表示操作が正しく完了した後に、[Waveform] ウィンドウに関連付けられている ILA コアが自動的にトリガー待機状態になります。

[Waveform] ウィンドウに表示された ILA コアに対応するキャプチャ データは、トリガー イベントごとに上書きされます。[Auto Re-Trigger] オプションは、[Run Trigger] および [Run Trigger Immediate] コマンドと共に使用できます。動作中のトリガーを停止するには、[Stop Trigger] をクリックします。

- [Run Trigger]: [Waveform] ウィンドウに関連付けられている ILA コアをトリガー待機状態にし、ILA コアの BASIC または ADVANCED トリガー設定で定義されたトリガー イベントが検出されるようにします。
- [Run Trigger Immediate]: [Waveform] ウィンドウに関連付けられている ILA コアをトリガー待機状態にし、ILA コアの BASIC または ADVANCED トリガー設定で定義されたトリガー イベントが検出されるようにします。このコマンドは、ILA コアのプローブ入力のすべてのアクティビティをキャプチャして、デザインの動作を検出するのに便利です。
- [Stop Trigger]: [Waveform] ウィンドウに関連付けられている ILA の ILA コア トリガーを停止します。
- [Export ILA Data]: ILA コアからのデータをキャプチャし、ファイルに保存します。データは、ネイティブ、.csv、または .vcd フォーマットで保存できます。[Waveform] ウィンドウのツールバー ボタンをクリックすると、次のダイアログ ボックスが表示されます。

図 155: [Export ILA Data] ダイアログ ボックス



[ILA Core] でデータをエクスポートする ILA デバッグ コアの名前を選択し、[Format] でフォーマットを [Native]、[CSV]、[VCD] のいずれかに設定します。

- [Native]: `write_hw_ila_data` コマンドで ILA データをデフォルトの ILA ファイル フォーマットでエクスポートします。このフォーマットのファイルは Vivado にインポートして、以前にキャプチャした ILA データを表示するのに使用できます。
- [CSV]: `write_hw_ila_data` コマンドで ILA データをスプレッドシートやサードパーティ アプリケーションにインポート可能な .csv ファイル フォーマットでエクスポートします。
- [VCD]: `write_hw_ila_data` コマンドで ILA データをサードパーティ アプリケーションまたはビューアーにインポート可能な .vcd ファイル フォーマットでエクスポートします。



重要: ILA データは CSV、VCD、またはネイティブ ILA フォーマットにエクスポートできますが、Vivado にインポートできるのはネイティブ ILA フォーマットのみです。Vivado にインポートした ILA データは、以前にキャプチャされたデータをオフラインで表示するためにのみサポートされます。プローブ信号をトリガーするためなどほかの目的で使用することはできません。

ズーム機能の使用

ツールバーには、波形のズーム機能にアクセスするボタンがあります。または、Ctrl キーを押しながらマウス ホイールを使用すると、現在選択している波形を拡大および縮小表示できます。ズーム レベルは Vivado セッション間で保持されず、毎回リセットされます。

図 156: 波形のズーム ボタン

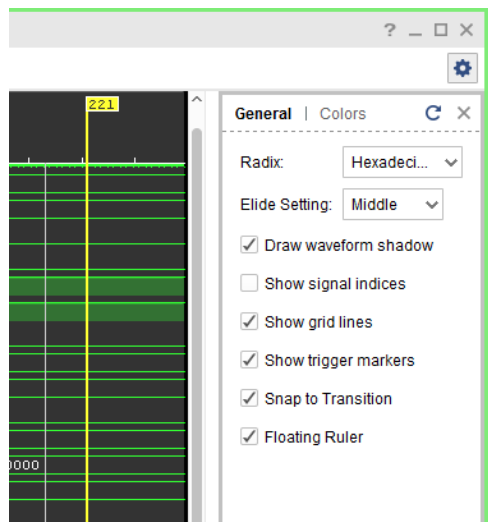


波形設定

波形ビューアーでは、オブジェクトの表示方法をカスタマイズできます。

[Waveforms Settings] ボタンをクリックすると、[Waveforms Settings] ウィンドウが開きます。

図 157: 波形設定



設定できるオプションは次のとおりです。

- [Colors] タブ: 波形オブジェクトの色を選択します。
- [Radix]: バス プローブのデフォルトの基数を設定します。
- [Draw waveform shadow]: 1 と 0 を見分けやすくするため、スカラー 1 の下に緑色の影を表示します。
- [Show signal indices]: スカラー名およびバス名の左側にインデックス位置番号を表示します。
- [Show trigger markers]: 波形ビューアーに赤色のトリガー マーカーを表示 (または非表示に) します。

波形コンフィギュレーションのカスタマイズ

次の表にリストされている機能を使用して、波形コンフィギュレーションをカスタマイズできます。この表には機能が簡単に説明されており、機能名をクリックするその機能を説明するセクションに移動できます。

表 25: 波形コンフィギュレーションのカスタマイズ機能

機能	説明
カーソル	2 つのカーソルを表示し、時間を計測できます。このカーソルの位置が、ナビゲーションの基準位置になります。
マーカー	マーカーを追加して波形をナビゲートし、特定時の波形値を表示できます。
仕切り	信号を見やすく分けるため仕切りを追加できます。
グループの使用	関連した信号およびバスをグループにまとめ、波形コンフィギュレーションに追加できます。
仮想バスの使用	論理スカラーおよび配列を追加できる仮想バスを波形コンフィギュレーションに追加できます。
オブジェクト名の変更	オブジェクト、信号、バス、グループの名前を変更できます。
基数	デフォルト基数は、波形コンフィギュレーション、[Objects] ウィンドウ、およびコンソールに表示されるバスの基数を指定します。
バス ビットの順序	最上位ビット (MSB) から最下位ビット (LSB)、またはその逆にバス ビット順を変更できます。

関連情報

[カーソル](#)

[マーカー](#)

[仕切り](#)

[グループの使用](#)

[仮想バスの使用](#)

[オブジェクト名の変更](#)

[基数](#)

[バス ビットの順序](#)

カーソル

カーソルは、サンプル位置を一時的に示すために使用します。波形エッジ間の距離 (サンプル数) を計測する場合には、カーソルを頻繁に移動します。



ヒント: 複数の計測の時間軸を設定する場合など、一時的ではないインジケータを設定する必要がある場合は、マーカーを追加します。詳細は、「マーカー」を参照してください。

波形を 1 回クリックすると、メイン カーソルが配置されます。

2 つ目のカーソルを配置するには、Ctrl キーを押しながらクリックし、マウスを左または右にドラッグします。カーソルの上に位置を示すフラグが表示されます。

または、Shift キーを押したまま波形のどこかをクリックします。メイン カーソルは元の位置に、もう一方のカーソルはクリックした位置に配置されます。

注記: 2 番目のカーソルの位置を保持しながらメイン カーソルの位置を変更するには、Shift キーを押しながらクリックします。2 つ目のカーソルをドラッグして配置する場合、最小間隔以上ドラッグしないと 2 つ目のカーソルは表示されません。

カーソルを移動するには、ポインターが手のひらのマークになるまでマウスを動かし、クリックして次の位置までカーソルをドラッグします。

カーソルをドラッグする際、[Snap to Transition] がオンになっていると (デフォルト)、中が塗りつぶされていない丸または中が塗りつぶされた丸が表示されます。

- 中が塗りつぶされていない丸 ○ は、選択した信号の波形の遷移間にあることを示します。
- 中が塗りつぶされている丸 ● は、選択した信号の波形の遷移地点にあることを示します。カーソル、マーカー、フロートしているルーラーがない場所をクリックすると、2 つ目のカーソルが非表示になります。


関連情報


[マーカー](#)

マーカー

波形内の重要イベントを恒久的にマークする必要がある場合はマーカーを使用します。マーカーが付けられたイベントに関連する距離 (サンプル数) を計測できます。

マーカーは、次のように追加、移動、削除できます。

- 波形設定のメイン カーソルの位置にマーカーを追加します。
 1. [Waveform] ウィンドウでマーカーを追加する位置のサンプル番号または遷移をクリックして、メイン カーソルを配置します。
 2. [Edit] → [Markers] → [Add Marker] をクリックするか、[Add Marker] ボタンをクリックします。 

カーソル位置にマーカーが配置されます。マーカーがその位置に既にある場合は、若干オフセットされます。マーカーのサンプル番号が上部に表示されます。
- マーカーを波形上の別の位置に移動するには、ドラッグ アンド ドロップします。マーカー上部のラベルをクリックしてドラッグします。
 - ドラッグ シンボル  は、マーカーが移動可能であることを示します。マーカーをドラッグする際、[Snap to Transition] がオンになっていると (デフォルト)、中が塗りつぶされていない丸または中が塗りつぶされた丸が表示されます。
 - 中が塗りつぶされている丸 ● は、選択した信号の波形の遷移地点、または別のマーカー上であることを示します。
 - マーカーの場合は、丸は白く塗りつぶされています。
 - 中が塗りつぶされていない丸 ○ は、選択した信号の波形の遷移間にあることを示します。
 - 新しい位置にマーカーをドロップするには、マウスのボタンを放します。
- 1 つのコマンドでマーカーを 1 つ、またはすべて削除できます。マーカーを右クリックして、次のいずれかの操作を実行します。
 - マーカーを 1 つ削除するには、ポップアップ メニューから [Delete Marker] をクリックします。
 - マーカーをすべて削除するには、ポップアップ メニューから [Delete All Markers] をクリックします。

注記: または、Delete キーを使用して選択したマーカーを削除することもできます。

 - マーカーの削除を取り消すには、[Edit] → [Undo] をクリックします。

トリガー マーカー

赤色のトリガー マーカー (赤文字の T) は、キャプチャ バッファでトリガー イベントの発生を示す特殊マーカーです。バッファのトリガー マーカーの位置は、トリガー位置設定に直接対応しています (「ILA デフォルト ダッシュボードの使用」を参照)。

注記: トリガー マーカーは、標準マーカーと同じ方法で移動させることはできません。位置は ILA コアの [Trigger Position] プロパティで設定します。

関連情報

[ILA デフォルト ダッシュボードの使用](#)

仕切り

仕切りは、信号を見やすく区切ります。波形コンフィギュレーションに仕切りを追加するには次の手順に従います。

1. [Waveform] ウィンドウの [Name] 列で、下に仕切りを追加する信号をクリックします。
2. ポップアップ メニューから [Edit] → [New Divider] をクリックするか、右クリックして [New Divider] をクリックします。

この変更は表示上のもので、HDL コードには何も追加されません。新しい区切りマークは、波形コンフィギュレーション ファイルを保存したときに保存されます。

仕切りは、次の方法で移動または削除できます。

- 仕切りを移動するには、名前を別の位置にドラッグ アンド ドロップします。
- 仕切りを削除するには、Delete キーを押すか、または右クリックしてポップアップ メニューから [Delete] をクリックします。

仕切りの名前を変更することもできます。詳細は、「オブジェクト名の変更」を参照してください。

関連情報

[オブジェクト名の変更](#)

グループの使用

関連した信号をまとめるため、波形コンフィギュレーションの信号およびバスをグループに追加できます。またグループの中に含まれているものを展開表示させたり、非表示にしたりできます。グループ自体は波形データを表示しませんが、その内容の表示/非表示を切り替えることができます。グループは追加、変更、削除できます。


グループを追加するには、次の手順に従います。

1. 波形コンフィギュレーションで、グループに追加する信号またはバスを 1 つ以上選択します。

注記: グループには、仕切り、仮想バス、ほかのグループを含めることができます。

2. [Edit] → [New Group] をクリックするか、右クリックして [New Group] をクリックします。

選択した信号またはバスを含むグループが波形コンフィギュレーションに追加されます。

グループは、グループ アイコンで表されます。 

この変更は表示上のもので、ILA コアには何も追加されません。

信号名やバス名をドラッグ アンド ドロップして、グループに信号やバスを追加することもできます。

グループは、次の方法で移動または削除できます。

- グループを移動するには、[Name] 列のグループ名を別の位置にドラッグ アンド ドロップします。
- グループを削除するには、グループを選択して [Edit] → [Wave Objects] → [Ungroup] をクリックするか、グループを右クリックして [Ungroup] をクリックします。グループに含まれていた信号またはバスは、波形コンフィギュレーションの一番上に配置されます。

グループ名も変更できます。詳細は、「オブジェクト名の変更」を参照してください。



注意: Delete キーを押すと、グループと、それに含まれている信号およびバスが波形コンフィギュレーションから削除されます。

関連情報


[オブジェクト名の変更](#)

仮想バスの使用

仮想バスを波形コンフィギュレーションに追加すると、論理スカラーおよび配列を追加できます。仮想バスには、バスの波形が表示されます。仮想バスはその下に昇順で表示される信号の波形で構成されており、1 次元配列にフラット化されます。追加した仮想バスは変更または削除できます。

仮想バスを追加するには、次の手順に従います。

1. 波形コンフィギュレーションで、仮想バスに追加する信号またはバスを 1 つ以上選択します。
2. [Edit] → [New Virtual Bus] をクリックするか、右クリックして [New Virtual Bus] をクリックします。

仮想バスは、Virtual Bus アイコン  で表されます。

この変更は表示上のもので、HDL コードには何も追加されません。

信号名やバス名をドラッグ アンド ドロップして、仮想バスに信号やバスを追加することもできます。波形コンフィギュレーション ファイルを保存すると、新しい仮想バスとそれに含まれる信号やバスも保存されます。また、仮想バスの名前を波形の別位置にドラッグ アンド ドロップして移動できます。

仮想バスの名前を変更するには、「オブジェクト名の変更」を参照してください。

仮想バスを削除し、その内容をグループ解除するには、仮想バスを選択して [Edit] → [Wave Objects] → [Ungroup] をクリックするか、右クリックして [Ungroup] をクリックします。



注意: Delete キーを押すと、仮想バスと、それに含まれている信号およびバスが波形コンフィギュレーションから削除されます。

関連情報

[オブジェクト名の変更](#)

オブジェクト名の変更

[Waveform] ウィンドウの信号、仕切り、グループ、仮想バスなどのオブジェクトの名前を変更できます。

1. [Name] 列でオブジェクト名を選択します。
2. 右クリックし、[Rename] をクリックします。
3. 新しい名前を入力します。
4. Enter キーを押すか、名前以外の場所をクリックして、変更を反映させます。

オブジェクト名をダブルクリックして新しい名前を入力することもできます。変更はすぐに反映されます。波形コンフィギュレーションでのオブジェクト名の変更は ILA コアのプローブ入力に接続されているネット名には影響しません。

基数

バスのデータ型を理解することは重要です。デジタルおよびアナログの波形オプションを効果的に使用するには、基数設定とデータ型の関係を知っておく必要があります。基数設定およびそのアナログ波形解析への影響については、「バスの基数」を参照してください。

[Waveform] ウィンドウで個々の信号 (ILA プローブ) の基数を変更するには、次の手順に従います。

1. [Waveform] ウィンドウでバスを右クリックします。
2. [Radix] をクリックし、ドロップダウン リストからフォーマットを選択します。
 - [Binary]
 - [Hexadecimal]
 - [Unsigned Decimal]
 - [Signed Decimal]
 - [Octal]

★ **重要:** [Objects] ウィンドウで基数を変更しても、[Waveform] ウィンドウまたは [Tcl Console] ウィンドウの値は変更されません。[Waveform] ウィンドウで個々の信号 (ILA プローブ) の基数を変更するには、[Waveform] ウィンドウのポップアップ メニューを使用してください。

- 実数での最大バス幅は 64 ビットです。64 ビットよりも幅の広いバスでは、正しい値にならない可能性があります。
- 浮動小数点では 32 ビットおよび 64 ビットの配列のみがサポートされています。

関連情報

[バスの基数](#)

フロート ルーラーの使用

[Waveform] ウィンドウの上部にある標準ルーラーの絶対サンプル値以外のサンプル値ベースを使用してサンプルを計測するには、フロート ルーラーを使用すると便利です。

フロート ルーラーは、表示/非表示を切り替えたり、[Waveform] ウィンドウの任意位置に移動させることができます。このルーラーのサンプル ベース (サンプル 0) は 2 番目のカーソルで、2 番目のカーソルがない場合は選択されたマーカーになります。

フロート ルーラー ボタンおよびフロート ルーラーは、2 番目のカーソル (または選択されたマーカー) がある場合にのみ表示されます。

1. ルーラーの表示/非表示を切り替えるには、次のいずれかを実行します。

- 2 番目のカーソルを配置します。
- マーカーを選択します。

2. [View]→[Floating Ruler] をクリックします。

この操作は初回のみ必要です。フロート ルーラーは 2 番目のカーソルが配置されるたび、またはマーカーが選択されるたびに表示されます。

ルーラーを非表示にするには、このコマンドをもう一度クリックします。

バス ビットの順序

波形コンフィギュレーションでバス ビットの順序を逆にして、信号を MSB から表示するか、LSB から表示するかを切り替えることができます。

ビット順序を逆にするには、次の手順に従います。

1. バスを選択します。
2. 右クリックし、[Reverse Bit Order] をクリックします。

これでバス ビットの順序が逆になります。[Reverse Bit Order] コマンドの横にチェック マークが表示され、適用されていることが示されます。

バスの基数

バスの値が数値として処理される方法は、バス波形オブジェクトの基数設定によって決まります。

- 2 進数、8 進数、16 進数、ASCII、および符号なしの 10 進数の基数を使用すると、バスの値が符号なしの整数として処理されます。バスのデータ フォーマットは基数設定と一致している必要があります。
- 0 または 1 以外のビットを使用すると、値すべてが 0 として処理されます。
- 符号付きの 10 進数基数を使用すると、バスの値が符号付き整数として処理されます。

アナログ波形の表示

デジタル波形をアナログに変換するには、次の手順に従います。

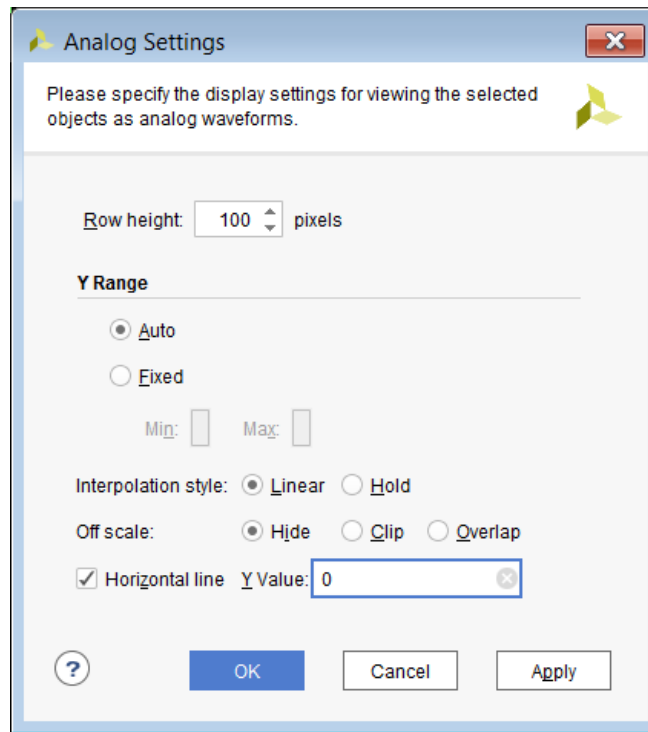
1. [Waveform] ウィンドウの [Name] 列でバスを右クリックします。
2. [Waveform Style] → [Analog Settings] をクリックして適切な設定を選択します。

バスのデジタル波形がアナログに変換されます。

アナログまたはデジタル波形の高さは、行をドラッグすると調節できます。

次の図に、アナログ波形表示を設定する [Analog Settings] ダイアログ ボックスを示します。

図 158: [Analog Settings] ダイアログ ボックス



[Analog Settings] ダイアログ ボックスのオプションは次のとおりです。

- [Row Height]: 選択した波形オブジェクトの高さをピクセルで指定します。行の高さを変更しても波形の垂直方向の表示域は変わりませんが、波形の高さの伸縮が変わります。

アナログとデジタルを切り替えるとき、行の高さはそれぞれに合った適切なデフォルトの高さに設定されます (デジタルの場合は 20、アナログの場合は 100)。

- [Y Range]: 波形エリアに表示される数値の範囲を指定します。
 - Auto: 表示されている時間の範囲の値が現在の範囲を超えたときに、表示範囲が拡大されます。
 - [Fixed]: 時間範囲を一定にします。
 - [Min]: 波形エリアの一番下に表示される値を指定します。
 - [Max]: 波形エリアの一番上に表示される値を指定します。

どちらの値も浮動小数点として指定できますが、波形オブジェクトの基数が整数の場合、値は整数に切り捨てられます。

- [Interpolation Style]: データ ポイントを接続するラインの描画方法を指定します。
 - [Linear]: 2 つのデータ ポイント間のラインを直線にします。
 - [Hold]: 2 つのデータ ポイントのうち、左のポイントから右のポイントの X 軸に向かって水平ラインを描画し、そのラインから右のポイントに向かって別のラインを L 字型に描画します。
 - [Off Scale]: 波形エリアの Y 軸を超えた値をどのように描画するかを指定します。

- [Hide]: 範囲外にある値を非表示にします。波形が波形エリアの上下の範囲外になると、値が範囲内に戻るまで非表示になります。
- [Clip]: 範囲外にある値は変更され、波形エリアの上下境界線を超えると範囲内に戻るまでは水平ラインとして表示されます。
- [Overlap]: 波形が波形エリアの範囲を超えてほかの波形と重なったとしても、波形ウィンドウの境界に達するまで値の位置に波形が描画されます。
- [Horizontal Line]: 指定した値で水平方向のラインを描画するかどうか指定します。このチェック ボックスをオンにすると、Y 軸の [Y Value] で指定した値の位置に水平線が描画されます (指定した値が波形の Y 範囲内である場合)。

[Min] および [Max] の場合と同様、Y 軸の値には浮動小数点値を指定できますが、選択した波形オブジェクトの基数が整数の場合は、整数値に切り捨てられます。



重要: アナログ設定は波形コンフィギュレーションに保存されますが、Y 軸方向のズーム コントロールは非常にインタラクティブであるため、基数などのほかの波形プロパティとは異なり、波形コンフィギュレーションの変更には影響しません。このため、ズーム設定は波形コンフィギュレーションには保存されません。

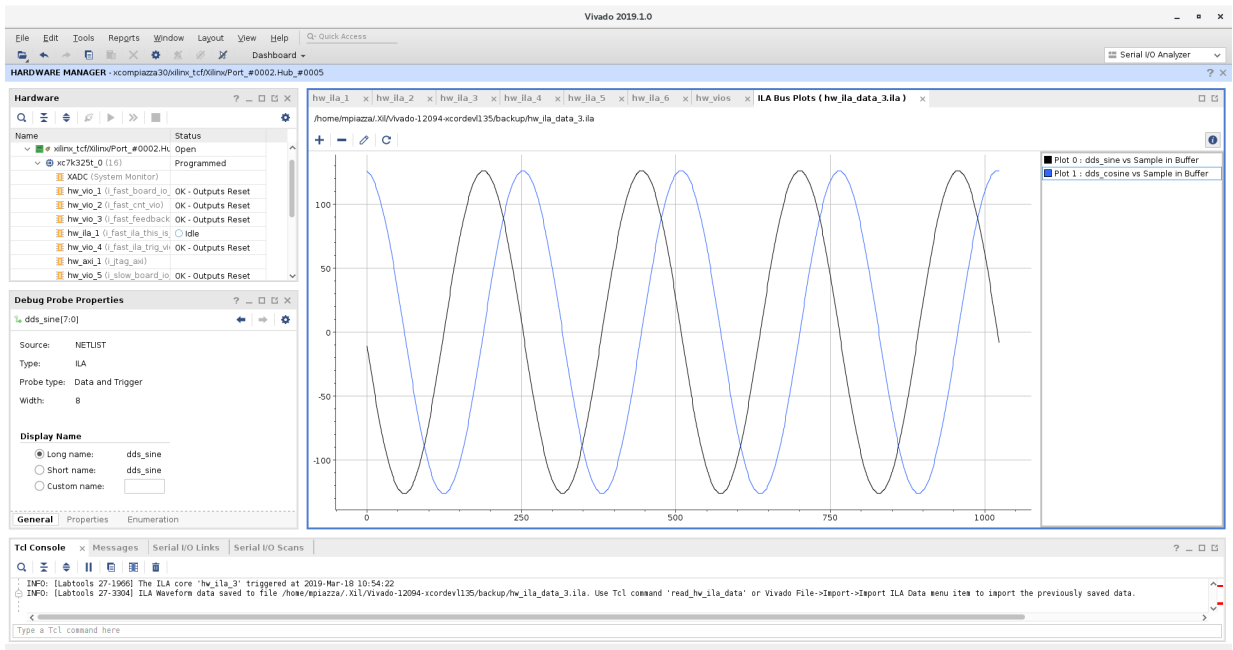
バス プロット ビューアー

Vivado® ハードウェア マネージャーでは、アナログ波形ビューアーに加え、バス プロット ビューアーがサポートされており、バスの値の時間経過に伴う変化や、XY 軸にプロットした 2 つのバスの値を表示できます。

バス プロットは、次を実行するのに便利です。

- アナログ サンプル データを時間に対してプロット
- 2 つのアナログ サンプル データをプロット

図 159: トリガー データとバッファのサンプルを示すバス プロット

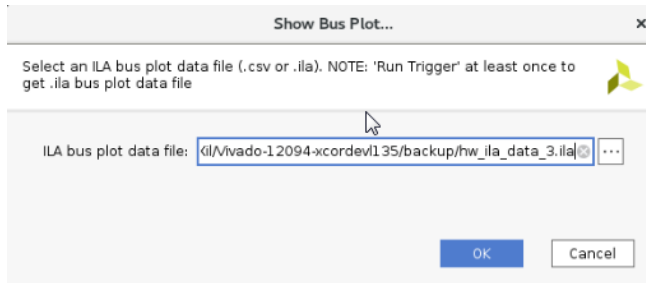


バス プロットの作成

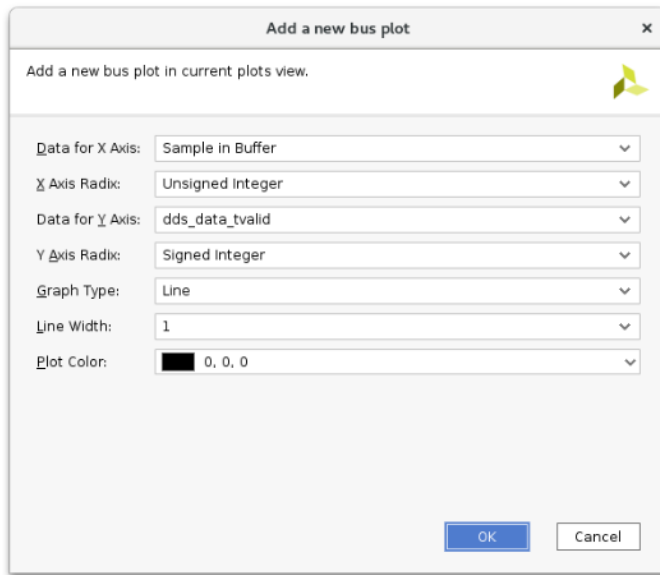
バス プロット ビューアーは、以前に収集された ILA トレース データをプロットするのに使用できます。この際、ILA バス プロット データ ファイル (.csv または .ila) へのパスを指定する必要があります。デフォルトでは、[Show Bus Plot] ダイアログ ボックスで最後に自動保存された ILA トリガー データが選択されます。

バス プロット作成の例

1. バス プロットを作成するには、Vivado ハードウェア マネージャーを開き、[Tools] → [Show Bus Plot] をクリックします。
2. ILA データ ファイルを選択する [Show Bus Plot] ダイアログ ボックスが表示されます。デフォルトでは、最後の ILA トレースに対応する自動保存された ILA トレースが選択されます。以前に保存した ILA データ ファイルを選択するには、該当する .ila または .csv ファイルへのパスを入力します。



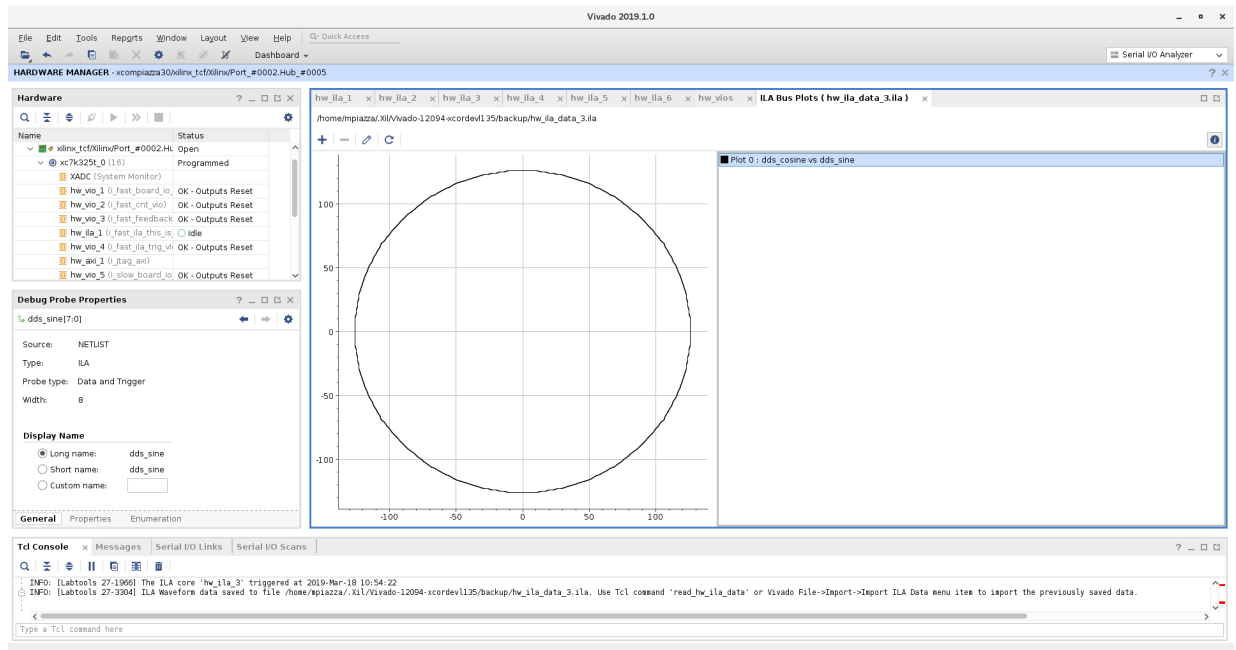
3. [OK] をクリックします。空のバス プロット ウィンドウが表示されます。バス プロットを追加するには、[+] ボタンをクリックします。新しいバス プロットのオプションを設定するダイアログ ボックスが表示されます。



[Add a new bus plot] ダイアログ ボックスに含まれるオプションは、次のとおりです。

- [Data for X Axis]: X 軸に使用するバス データを指定します。
 - [Sample in Buffer]: ILA キャプチャ バッファ内の ILA サンプル番号。
 - [Sample in Window]: キャプチャ ウィンドウの ILA サンプル番号。1 つのキャプチャ ウィンドウを選択している場合はこの番号はバッファ内のサンプルと同じですが、複数のキャプチャ ウィンドウを使用している場合は、この番号は指定のキャプチャ ウィンドウのサンプル番号を示します。
 - [TRIGGER]: キャプチャ ウィンドウのトリガー位置。
- [X Axis Radix]: X 軸データのプロットに使用する基数を指定します。
 - 符号付き整数。
 - 符号なし整数。
- [Data for Y Axis]: Y 軸に使用するバス データを指定します。
 - [Sample in Buffer]: ILA キャプチャ バッファ内の ILA サンプル番号。
 - [Sample in Window]: キャプチャ ウィンドウの ILA サンプル番号。1 つのキャプチャ ウィンドウを選択している場合はこの番号はバッファ内のサンプルと同じですが、複数のキャプチャ ウィンドウを使用している場合は、この番号は指定のキャプチャ ウィンドウのサンプル番号を示します。
 - [TRIGGER]: キャプチャ ウィンドウのトリガー位置。
- [Y Axis Radix]: Y 軸データのプロットに使用する基数を指定します。
 - 符号付き整数。
 - 符号なし整数。
- [Graph Type]
 - [Line]: 個別のサンプルを連続する線でつないだバス プロットを表示します。
 - [Point]: 個別のサンプルを点で表したバス プロットを表示します。
- [Line Width]: バス プロット ビューアーに信号を描画するのに使用する幅を指定します。
- [Plot Color]: バス プロットを描画する色を選択します。

- バス プロットを設定したら [OK] をクリックし、Vivado ハードウェア マネージャーにバス プロットを追加します。バス プロットが表示されます。

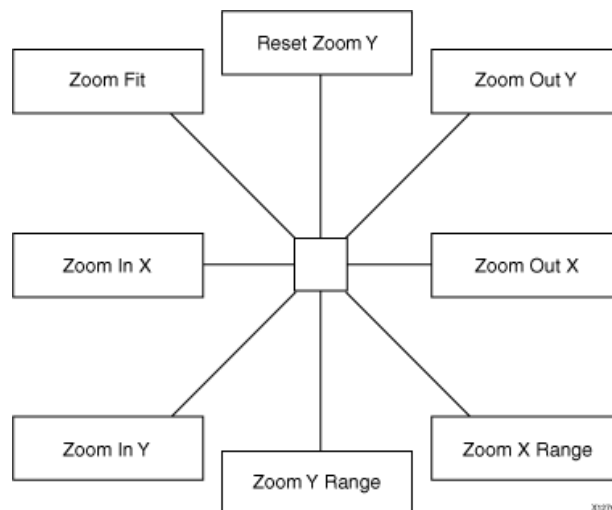


注記: Vivado ハードウェア マネージャーを閉じる際、バス プロット設定は保存されません。Vivado IDE を閉じる前に、必要な計測が完了していることを確認してください。

ズーム機能

X 軸方向のズームでサポートされている機能に加え、アナログ波形の場合は、次の図に示す追加のズーム機能があります。

図 160: アナログ ズームのオプション



ズーム機能を使用するには、マウスの左ボタンを押したまま、図で示されている方向にマウスをドラッグします。この図の中央がマウスの位置です。

次の追加ズーム機能があります。

- Zoom Out Y: 始点からマウス ボタンを放した位置までの距離により決定される 2 のべき乗分 Y 軸方向に縮小します。始点のマウス位置の Y 値をそのまま維持してズームが実行されます。
- Zoom Y Range: 垂直方向に線を描き、マウス ボタンを離れた位置までの Y 軸の範囲を表示します。
- Zoom In Y: 始点からマウス ボタンを放した位置までの距離により決定される 2 のべき乗分 Y 軸方向に拡大します。

始点のマウス位置の Y 値をそのまま維持してズームが実行されます。

- Reset Zoom Y: Y の範囲を波形ウィンドウに現在表示されている値にリセットし、Y の範囲モードを [Auto] に設定します。

Y 軸の方向のズーム機能はすべて Y の範囲のアナログ値を設定します。[Reset Zoom Y] は Y の範囲を [Auto] に設定しますが、ほかのズーム機能は [Fixed] に設定します。

インプリメンテーション後のデザインのデバッグ

インプリメンテーション後、デバッグ コアを変更、追加、または削除する必要があることがあります。Vivado® Design Suite では 2 つの方法があります。

ILA コアへの既存の接続を置き換えるには、ECO フローを使用することをお勧めします。ECO フローは、インプリメント済みチェックポイント (DCP) に対して実行され、デザイン全体を再配線するのにかかる時間を節約できます。

新規 ILA コアを追加、既存の ILA コアを削除、または既存の ILA コアを変更する場合は (プローブ幅の変更、データ深さの変更など)、インクリメンタル コンパイル フローを使用することをお勧めします。デバッグ コア用のインクリメンタル コンパイル フローは、合成済みデザインまたはチェックポイント (DCP) に対して実行し、基準のインプリメント済みチェックポイント (理想的には前回のインプリメンテーション run からのチェックポイント) を使用します。これにより、デザイン全体を再インプリメントするのにかかる時間を節約できます。

この後のセクションで、これらのデバッグに関連するフローを詳細に説明します。

Vivado ECO フローを使用した既存のデバッグ プローブの置き換え

配置配線済みのデザイン チェックポイントで、ILA コアに接続されているデバッグ ネットを置き換えることができます。これには、通常エンジニアリング チェンジ オーダー (ECO) を使用します。これは、完成に近づいており、ILA プローブ ポートに接続されているネットを置き換える必要があるデザインに使用されるアドバンス デザイン フローです。この方法には、次の 2 つの利点があります。

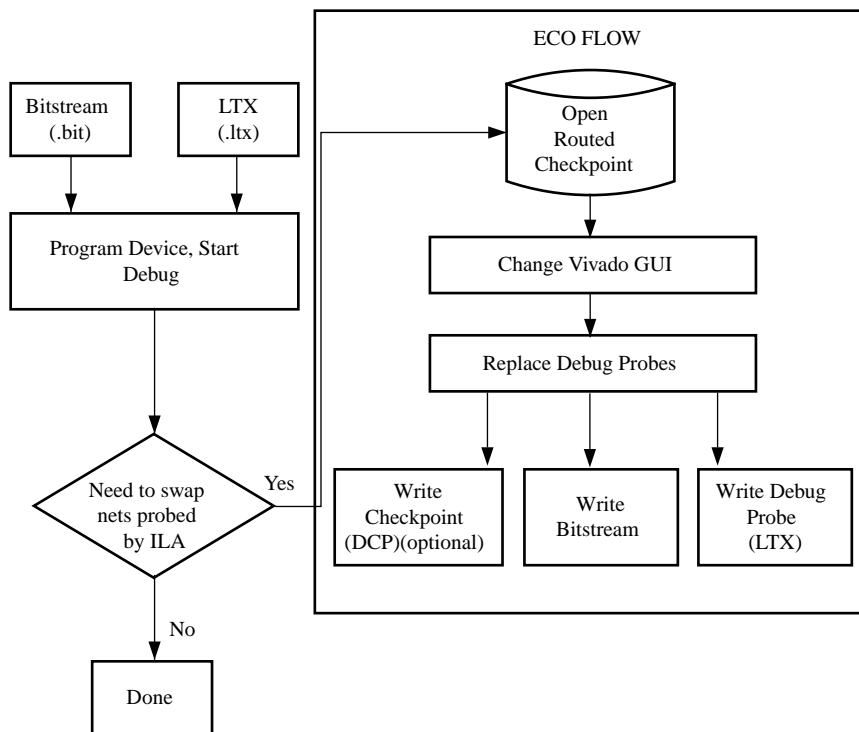
- **時間の節約:** 異なるネットをプローブしている既存のデバッグ ネットを置換できます。
- **変更が最小限:** プローブ ネットを置き換えた後、これらのネットをデバッグ コアの入力に配線する必要があります。デザインのそれ以外の部分は変更されず、以前のインプリメンテーション結果が保持されるだけでなく、再インプリメントによりバグが隠されてしまう可能性を削減します。



重要: このフローは、ILA コアが既にインスタンスiertまたは挿入されているデザインでのみ使用可能です。

次の図に、ECO デザイン フローを使用したデバッグ ネットの置換プロセスを示します。

図 161: デバッグ ECO デザイン フロー



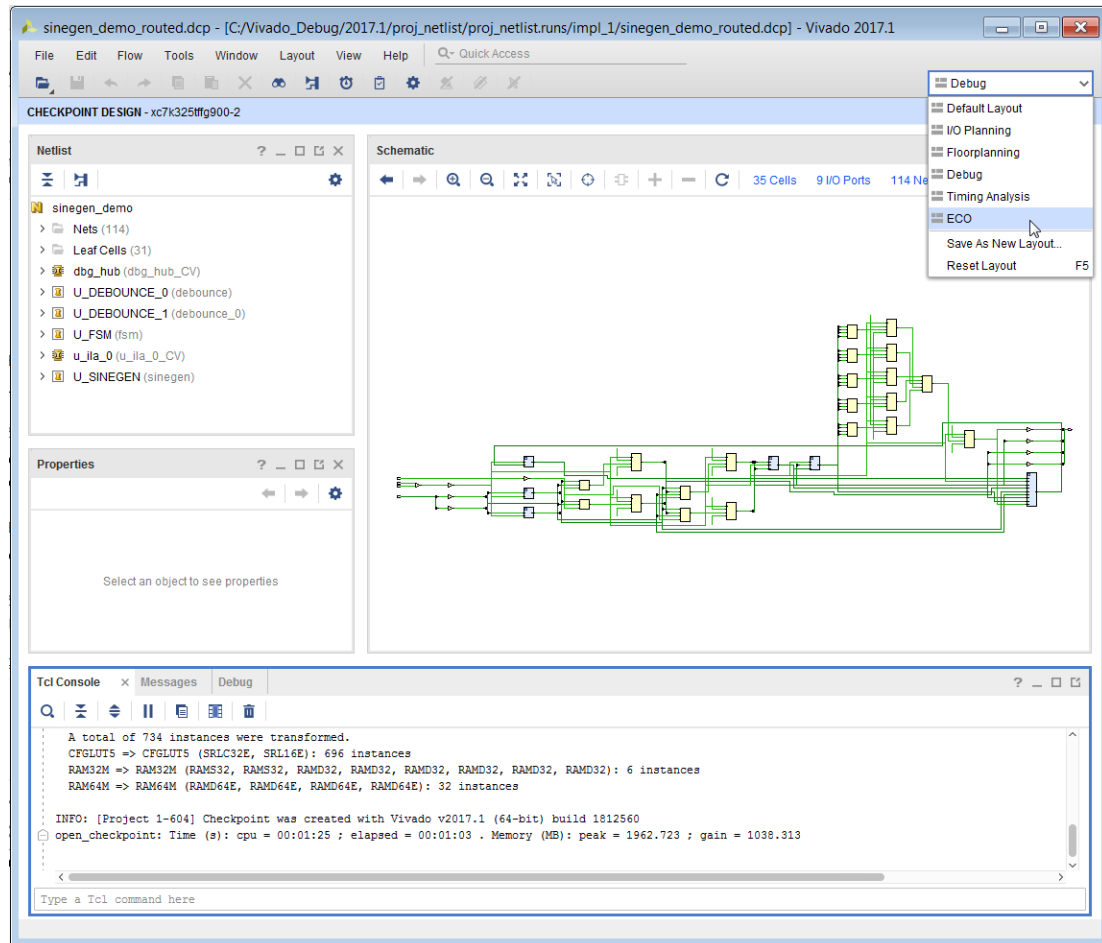
X16399-040516

配置配線済みデザイン チェックポイントのデバッグ プロ ープの置換

Vivado ハードウェア マネージャーを使用してデバイスにプログラムされたデザインをデバッグする際、デバッグ用にプローブするネットを別のネットに置き換える必要がある場合があります。RTL コードを変更したり、挿入されたデバッグ コアでプローブするネットを変更したりする代わりに、ECO フローを使用してデバッグ ネットを置き換えることができます。

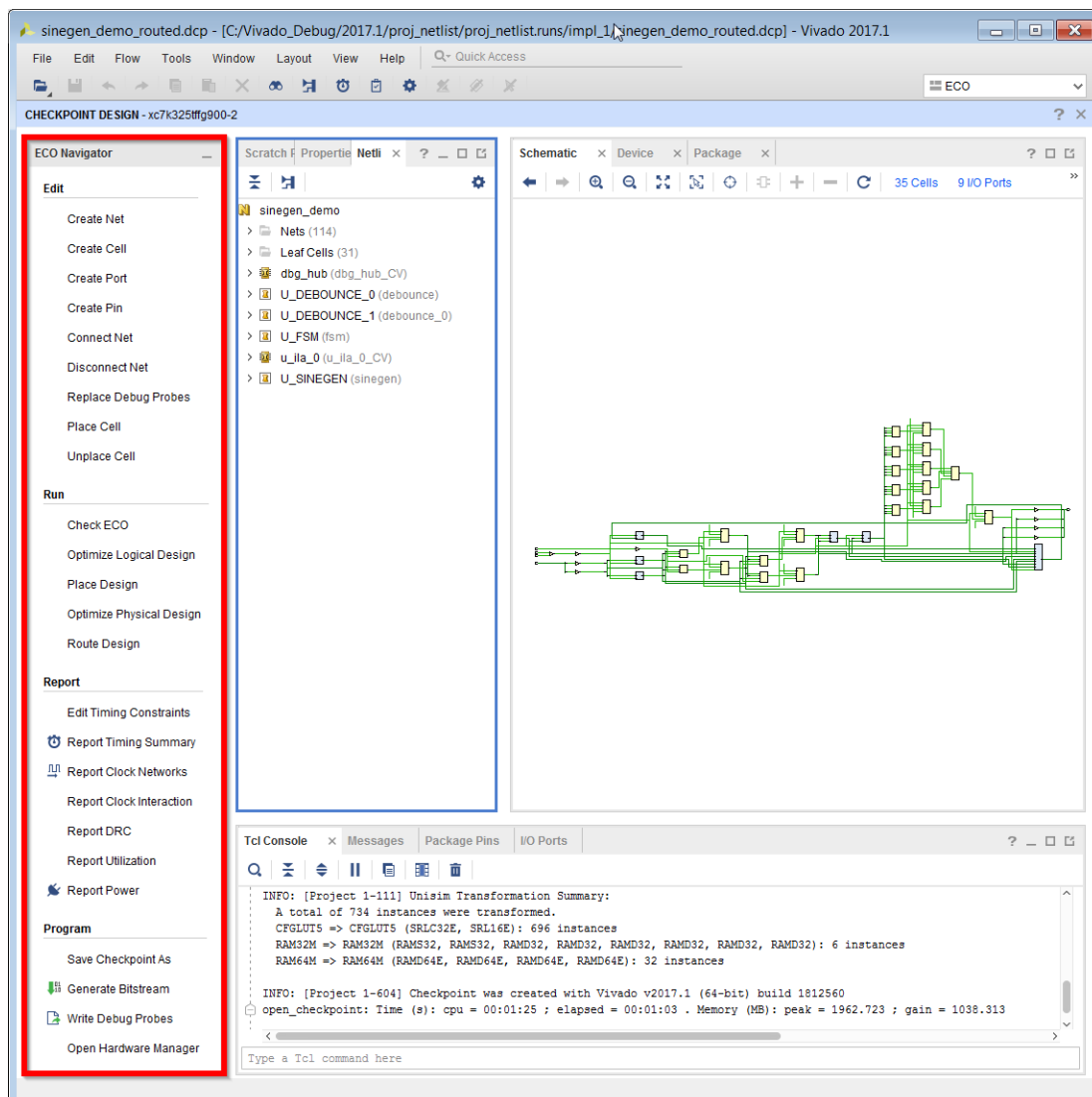
ECO フローを使用するには、Vivado IDE で配置配線済みチェックポイント (DCP) を開き、レイアウトを [ECO] に変更します。

図 162: [ECO] レイアウトの選択



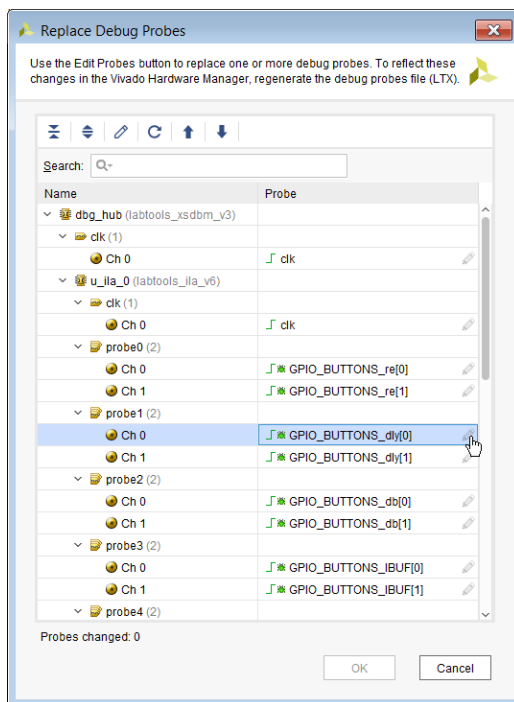
Flow Navigator が ECO Navigator に変更され、異なるオプションが表示されます。

図 163: ECO Navigator



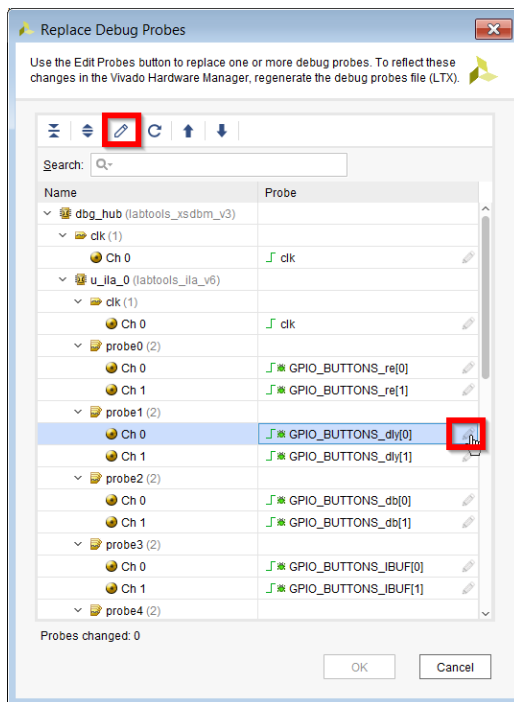
ECO Navigator で [Replace Debug Probes] をクリックして [Replace Debug Probes] ダイアログ ボックスを開きます。

図 164: [Replace Debug Probes] ダイアログ ボックス



[Replace Debug Probes] ダイアログ ボックスで、変更する必要があるネットのプローブを選択して [Edit Probes] ボタンをクリックします。個別のネットを変更するには、プローブの右側にある [Edit Probes] ボタンをクリックします。複数のプローブのネットを変更するには、左側のツールバーにある [Edit Probes] ボタンをクリックします。

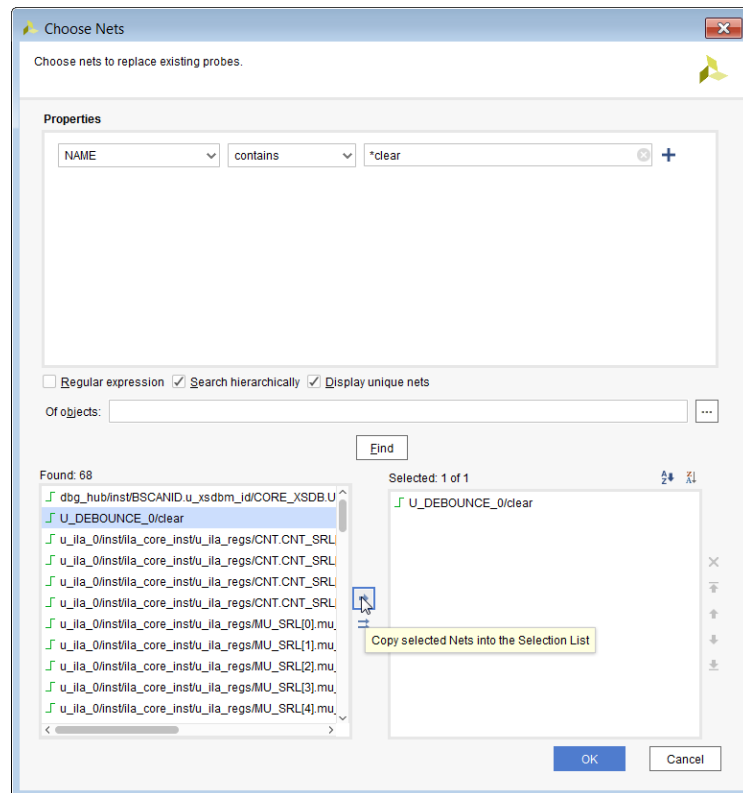
図 165: [Edit Probes] ボタン



[Edit Probes] ボタンをクリックすると、[Choose Nets] ダイアログ ボックスが開き、既存のネットを置き換えるネットを選択できます。

[Find] ボックスにキーワードを入力して、置き換える必要のあるネットを選択します。[Find] をクリックして、10000 を超える数のネットが検索された場合は、キーワードを変えて検索を絞り込んでください。左側の [Find Results] に表示されたネットから目的のものを選択し、矢印 ([→]) をクリックして、右側の [Selected Names] に追加します。[Selected Names] のネットの数が、置き換えようとしているネットの数と一致していることを確認します。[OK] をクリックして、作業を続けます。

図 166: [Choose Nets] ダイアログ ボックス



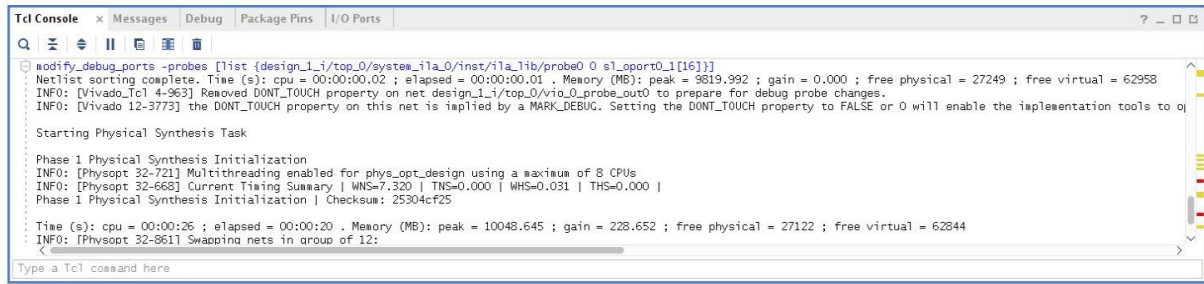
★ **重要:** 必要なデバッグ プローブをすべて置き換えたら、最配線してビットストリームを生成し直し、デバッグ プローブ ファイル (.ltx) を生成し直す必要があります。

💡 **ヒント:** 複数のネットまたはバスを選択するには、[Replace Debug Probes] ダイアログ ボックスの左側の [Edit Probes] ボタンをクリックします。

デバッグ コアの必要なネットをすべて置き換えたら、[OK] をクリックし、表示される確認ダイアログ ボックスで変更内容を確認します。

★ **重要:** [Tcl Console] ウィンドウに警告またはエラーが表示されていないことを確認します。

図 167: modify_debug_ports の Tcl メッセージ



プローブするネットのパス上にあるネット セグメントを削除すると、ハードウェア マネージャーに表示されるプローブ名が変更されることがあります。Vivado IDE では、MARK_DEBUG 属性が設定されたプローブするネットに最も近いネット セグメントが選択されます。MARK_DEBUG 属性が設定されたネット セグメントがない場合は、最上位ネットが選択されます。MARK_DEBUG 属性が設定されたネット セグメントが複数ある場合は、これらのいずれかがランダムに選択されます。

すべてのデバッグ プローブ ポートを置き換えたら、ECO Navigator で [Save Checkpoint As] をクリックして変更後の新しいチェックポイントを保存できます。ECO Navigator の [Replace Debug Probes] コマンドを実行して、デバッグ プローブの新しい .ltx ファイルを生成する必要があります。その後、新しい BIT ファイルを生成してデバイスをプログラムします。これで、Vivado ハードウェア マネージャーに接続して変更したデザインをデバッグできます。

Vivado ECO Tcl フローを使用した既存のデバッグ プローブの置き換え

前のセクションで説明される GUI フローの代わりに、Vivado Tcl フローを使用することもできます。次の Tcl コマンドを使用し、デバッグ コアでプローブされるネットを変更します。

```
modify_debug_ports -probes [list {top/x_ila/probe0 0 top/inst_A/net_0} \
    {top/x_ila/probe1 1 top/inst_A/net_a} {top/x_ila/probe1 2 top/inst_A/
net_b}]
```

このコマンドは、すべてのネットリスト変更を実行して、指定したプローブ ポートへの既存のネット接続を解除します。この例の場合、ILA のインデックス 0 のプローブ ポート 0、インデックス 1 およびインデックス 2 のプローブ ポート 1 への既存のネット接続が解除されます。この後、これらのプローブがそれぞれ net_0、net_a、および net_b と指定されたネットに接続されます。また、変更された接続も自動的に配線されます。このプロセスで接続が解除されたネットは、未接続のままになります。

デバッグ コア (ILA) を変更した場合のインクリメンタル コンパイル

インクリメンタル コンパイルは、完成に近づいており、小さな変更が必要なデザインに適したアドバンス フローです。少しの変更を加えて再合成する場合にこのフローを使用すると、次のような利点があります。

- 配置配線の実行時間が短縮されます。

- 基準デザインからの配置配線が再利用され、QoR (結果の品質) の予測性が保持されます。このフローは、変更されたデザインと基準デザインが 95% 以上類似している場合に最も効果的です。

インクリメンタル デバッグ変更は、インクリメンタル コンパイル デザイン フローを使用してデザインを再インプリメントすることにより、配置配線済みデザインに適用します。このフローは、次のいずれか状況で推奨されます。

- 既存のインプリメント済みデザインにデバッグ コアが存在しない。
- 既存のデバッグ コアのプロープ幅、データ深さなどを変更する必要がある。
- デザインからデバッグ コアを削除する必要がある。

インクリメンタル コンパイル フローのデザイン

インクリメンタル コンパイル フローでは、基準デザインとデバッグ コアを変更した現在のデザインの 2 つのデザインを使用します。

リファレンス デザイン

基準デザインは、通常はデザインの合成、配置、配線が完了した以前のバージョンです。配置、配線、またはその両方がどれだけ含まれたチェックポイントでも使用可能です。基準デザインには、タイミング クロージャを達成するためにコード変更、フロアプラン、制約変更した際のデザインの実行結果であるデザイン チェックポイント (DCP) を使用できます。現在のデザインを読み込んだら、`read_checkpoint -incremental <dcg>` コマンドを使用して基準デザイン チェックポイントを読み込みます。`-incremental` オプションを使用して基準デザイン チェックポイントを読み込むと、次の配置配線でインクリメンタル コンパイル デザイン フローがイネーブルになります。

現在のデザイン

現在のデザインは、基準デザインにデバッグに関連する変更を少し加えたものです。次のような変更が含まれます。

- デバッグ コアの RTL インスタンス化の変更
- デバッグ コア挿入の変更
- デバッグ コアに関する RTL の変更および挿入の変更の両方

インプリメント済みの既存のデザインでデバッグ コアを挿入、削除、または変更するには、合成済みの DCP またはデザインを開き、デバッグ挿入フローを使用します。デバッグ挿入フローの詳細は、「ネットリスト挿入デバッグ プロープ フロー」を参照してください。

既存のデバッグ コアを変更したり、新しいデバッグ コアを既存の RTL デザインにインスタンス化することでもできます。インクリメンタル コンパイル フローでは、新しいデバッグに関する変更と共に、基準のデザインからの配置配線が再利用されます。デバッグ インスタンス化 フローの詳細は、「HDL インスタンス化 デバッグ プロープ フローの概要」を参照してください。

関連情報

[ネットリスト挿入デバッグ プロープ フロー](#)

[HDL インスタンス化 デバッグ プロープ フローの概要](#)

インクリメンタル コンパイルの使用

プロジェクト モードと非プロジェクト モードのどちらでも、`read_checkpoint -incremental <reference_dcp_file>` コマンド (<reference_dcp_file> は基準デザイン チェックポイントのパスとファイル名) を使用して基準デザイン チェックポイントを読み込むと、インクリメンタル配置配線モードになります。-incremental オプションを使用して基準デザイン チェックポイントを読み込むと、次の配置配線でインクリメンタル コンパイル デザイン フローがイネーブルになります。非プロジェクト モードでは、`read_checkpoint -incremental` は `opt_design` の後、`place_design` の前に実行します。デバッグ挿入フローを使用する場合、デバッグ コアに関連する XDC コマンドは `opt_design` の前に実行する必要があります。

非プロジェクト モードでのインクリメンタル コンパイルの使用

非プロジェクト モードで基準デザインとして使用するデザイン チェックポイント ファイル (DCP) を指定してインクリメンタル配置を実行するには、次の手順に従います。

1. 現在のデザインを読み込みます。
2. デバッグ コア コマンドを実行します。
3. `opt_design` を実行します。



重要: `opt_design` オプションと指示子が元の参照している run と使用されているものと、できるだけ同じになるようにしてください。

4. `read_checkpoint -incremental <reference_dcp_file>` を実行します。
5. `place_design` を実行します。
6. `route_design` を実行します。

```
# to load the current design
link_design;
#Create the debug core
create_debug_core u_ila_0 ila
#set debug core properties
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER false [get_debug_cores u_ila_0]
set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL false [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBE_SAME_MU_CNT 1 [get_debug_cores u_ila_0]
#connect the probe ports in the debug core to the signals being probed
in the design
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get_nets [list clk ]]
set_property port_width 1 [get_debug_ports u_ila_0/probe0]
connect_debug_port u_ila_0/probe0 [get_nets [list A_or_B]]
create_debug_port u_ila_0 probe
opt_design
read_checkpoint -incremental <reference_dcp_file>
place_design
route_design
```



重要: デザインのデバッグ コアを変更するには、合成済みチェックポイントを開く必要があります。配線後のチェックポイントにデバッグ コアを挿入することはできません。

プロジェクト モードでのインクリメンタル コンパイルの使用

プロジェクト モードでは、[Design Runs] ウィンドウでインクリメンタル コンパイル オプションを設定できます。

インクリメンタル コンパイル モードを設定するには、次の手順に従います。

1. [Design Runs] ウィンドウで run を 1 つクリックします。
2. [Set Incremental Compile] をクリックします。
3. [Set Incremental Compile] ダイアログ ボックスで、基準デザイン チェックポイントを選択します。run でインクリメンタル コンパイル モードがイネーブルになります。
4. 合成済みネットリストを開き、オプションでデバッグ コアを追加または RTL にインスタンス化されたデバッグ コアを変更します。
5. Set Up Debug ウィザードを使用して、デバッグ コアを挿入するか、デザインに挿入されているデバッグ コアを削除または変更します。
6. デザインをインプリメントします。



重要: デザインのデバッグ コアを変更するには、合成済みデザインを開く必要があります。配線後のデザインにデバッグ コアを挿入することはできません。

インクリメンタル コンパイル機能の詳細は、『Vivado Design Suite ユーザー ガイド: インプリメンテーション』(UG904) の[このセクション](#)を参照してください。

基準デザインと現在のデザインの類似性の確認

`report_incremental_reuse` コマンドを使用すると、基準デザイン チェックポイントと現在のデザインの類似性をレポートできます。`report_incremental_reuse` コマンドは、基準デザイン チェックポイントからのネットリストをメモリ内の現在のデザインと比較し、セル、ネット、ポートの一致量をパーセントでレポートします。

デザイン間の差異が少ないほど、基準デザインからの配置配線の再利用が効率的に実行されています。基準デザインと現在のデザインの一致パーセントが高い方が、配置配線が再利用される確率が高くなります。

シリアル I/O ハードウェア デバッグ フロー

関連情報

[ハードウェアでのシリアル I/O デザインのデバッグ](#)

シリアル I/O ハードウェア デバッグ フロー

Vivado[®] IDE では、ザイリンクスの高速ギガビット トランシーバー (GT) テクノロジーを使用するシステムをデバッグおよび検証するためのデザインを簡単に生成できます。インシステム シリアル I/O デバッグ フローには、次の 3 つの段階があります。

1. IBERT コア生成: ハードウェア高速シリアル I/O 要件を最適に満たす IBERT コアをカスタマイズおよび生成します。
2. IBERT サンプル デザインの生成およびインプリメンテーション: 生成した IBERT コアのサンプル デザインを生成します。
3. シリアル I/O 解析: デザインに含まれる IBERT IP にアクセスし、高速シリアル I/O リンクの問題をデバッグおよび検証します。

この章では、最初の 2 つの段階について説明します。3 段階目については、「ハードウェアでのシリアル I/O デザインのデバッグ」を参照してください。

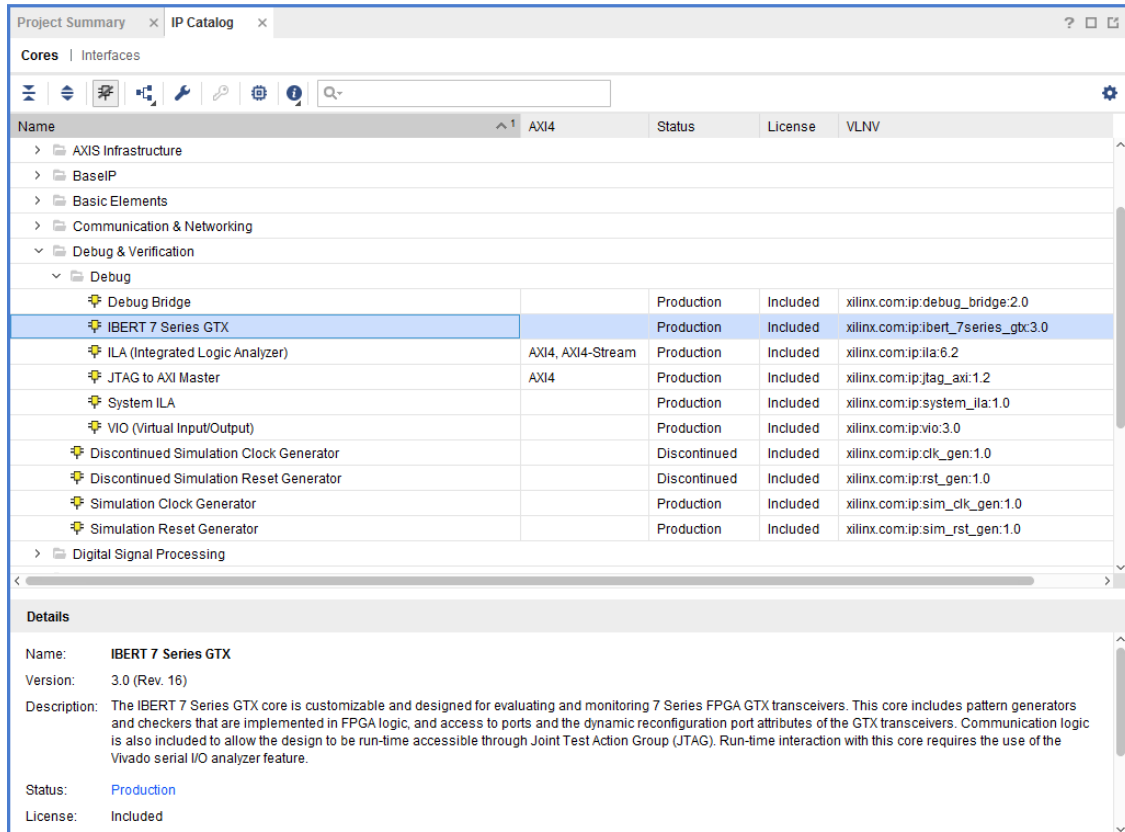
Vivado IP カタログを使用した IBERT コアの生成

システムの高速シリアル I/O インターフェイスをデバッグおよび検証するハードウェア デザインを生成するための最初の段階は、IBERT コアを生成することです。これには、次の手順に従います。

1. Vivado IDE を開く。
2. 最初の画面で [Manage IP] → [New IP Location] をクリックし、Open IP Catalog ウィザードが表示されたら [Next] をクリックします。
3. パーツ、ターゲット言語、ターゲット シミュレータ、および IP ディレクトリを選択します。[Finish] をクリックします。
4. IP カタログで [Debug and Verification] → [Debug] を展開すると、前の手順で選択したデバイスに応じて 1 つまたは複数の IBERT コアが表示されます。
5. 目的の IBERT アーキテクチャをダブルクリックし、コアの [Customize IP] ダイアログ ボックスを開きます。

ハードウェア システム要件に合わせて IBERT コアをカスタマイズします。さまざまな IBERT コアの詳細は、次の IP 資料を参照してください。

- 『Integrated Bit Error Ratio Tester 7 Series GTX Transceivers LogiCORE IP 製品ガイド』 (PG132)
- 『Integrated Bit Error Ratio Tester 7 Series GTP Transceivers LogiCORE IP 製品ガイド』 (PG133)
- 『Integrated Bit Error Ratio Tester 7 Series GTH Transceivers LogiCORE IP 製品ガイド』 (PG152)



IBERT サンプル デザインの生成とインプリメンテーション

IBERT IP コアを生成すると、[Sources] ウィンドウに `ibert_7series_gtx` または類似のコアが表示されます。サンプル デザインを生成するには、[Sources] ウィンドウで IBERT IP を右クリックして [Open IP Example Design] をクリックし、表示されるダイアログ ボックスでサンプル デザイン プロジェクトを保存するディレクトリを選択します。このコマンドにより、サンプル デザイン用の Vivado プロジェクト ウィンドウが新しく開き、最上位ラッパーおよび制約ファイルが追加されます。



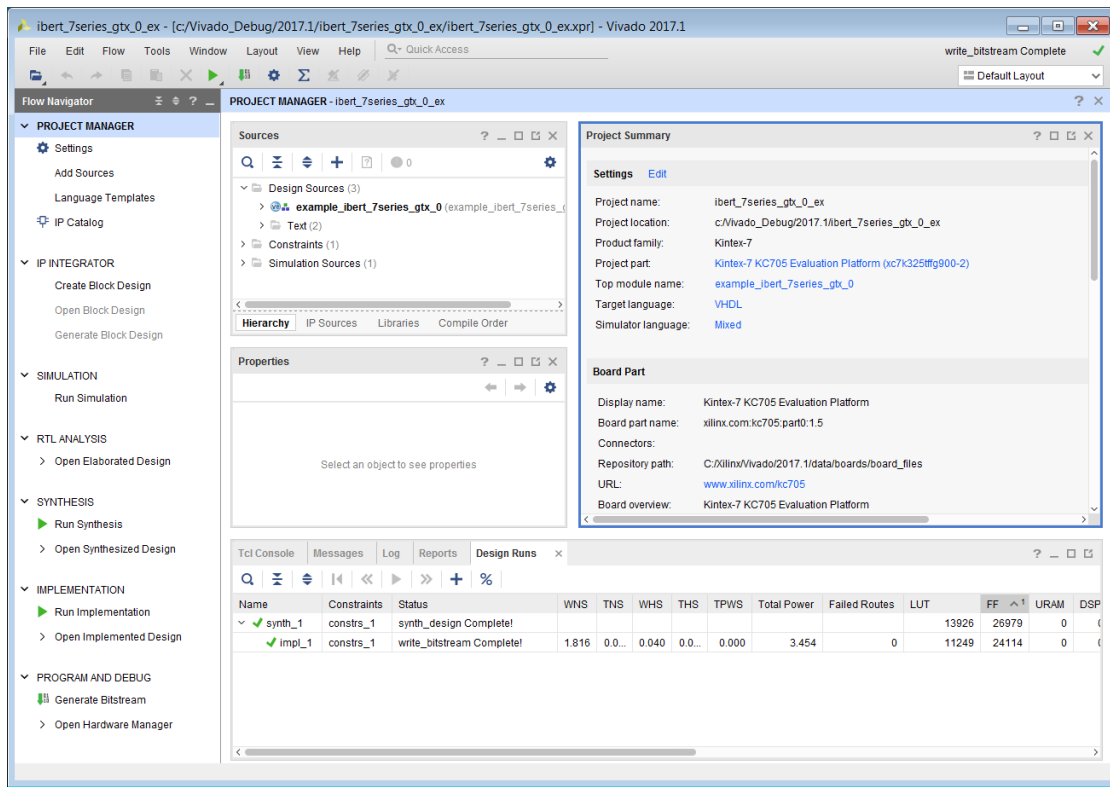
重要: IBERT IP サンプル デザインの変更は、ハードウェアで IBERT IP コアを使用する際に論理的な問題が発生する可能性があるためにお勧めしません。

サンプル デザインを生成したら、Vivado IDE の Flow Navigator で [Generate Bitstream] をクリックするか、次の Tcl コマンドを使用して、IBERT サンプル デザインのインプリメンテーションおよびビットストリーム生成を実行します。

```
launch_runs impl_1 -to_step write_bitstream
wait_on_run impl_1
```

さまざまなインプリメント方法の詳細は、『Vivado Design Suite ユーザー ガイド: デザイン フローの概要』 (UG892) を参照してください。

図 168: IBERT サンプル デザイン



In-System IBERT システム シリアル I/O デザインのデバッグ

In-System IBERT IP を使用すると、デザインの UltraScale および UltraScale+ トランシーバーの 2D のアイスキャンを Vivado Serial IO Analyzer を使用して実行できます。デザインからのデータを使用し、リアルタイムでトランシーバーのアイスキャンを出力しつつ、システムのほかの部分とも通信します。この IP は、デザインのユーザー ロジック、または GT ウィザードや Aurora などザイリンクス トランシーバー ベースの IP と通信できます。

インシステム シリアル I/O デバッグ フローには、次の 3 つの段階があります。

1. In-System IBERT コアの生成: ハードウェア高速シリアル I/O 要件を最適に満たす In-System IBERT コアをカスタマイズおよび生成します。
2. 統合: IP インスタンスエートし、それをデザインにインテグレートします。
3. シリアル I/O 解析: シリアル I/O 解析: デザインに含まれる In-System IBERT IP にアクセスし、高速シリアル I/O リンクの問題をデバッグおよび検証します。



重要: In-System IBERT コアは UltraScale および UltraScale+ デバイス ファミリーにのみ使用可能です。

In-System IBERT コア生成段階と統合段階の詳細は、この章の残りの部分を参照してください。シリアル I/O 解析の詳細は、「ハードウェアでのシリアル I/O デザインのデバッグ」を参照してください。

関連情報

[ハードウェアでのシリアル I/O デザインのデバッグ](#)

Vivado IP カタログを使用した In-System IBERT コアの生成

デザインの高速シリアル I/O インターフェイスのデバッグは、In-System IBERT コアの生成から始めます。

次の手順に従って作業をします。

1. Vivado IDE を開きます。
2. 最初の画面で [Manage IP] → [New IP Location] をクリックし、Open IP Catalog ウィザードが表示されたら [Next] をクリックします。
3. パーツ、ターゲット言語、ターゲット シミュレータ、および IP ディレクトリを選択します。[Finish] をクリックします。
4. [IP Catalog] で [Debug and Verification] → [Debug] を展開すると、前の手順で選択したデバイスに応じて 1 つまたは複数の In-System IBERT コアが表示されます。
5. 目的の In-System IBERT アーキテクチャをダブルクリックし、コアの [Customize IP] ダイアログ ボックスを開きます。

ハードウェア システム要件に合わせて In-System IBERT コアをカスタマイズします。In-System IBERT コアの詳細は、『In-System IBERT LogiCORE IP 製品ガイド』(PG246) を参照してください。

IP のインスタンスエートおよび In-System IBERT IP のユーザー デザインへの統合

In-System IBERT IP コアを生成したら、次の作業をします。

1. 最上位 RTL ファイルを開き、上記の手順で生成した In-System IBERT IP を追加します。
2. ツールで生成された In-System IBERT IP のインスタンスエーション テンプレートをコピーし、それを RTL ファイルにインスタンスエートします。
3. トランシーバーのポートを In-System IBERT IP に接続します。

In-System IBERT IP をユーザー デザインに統合する方法の例に関しては、『In-System IBERT LogiCORE IP 製品ガイド』(PG246) の第 5 章のサンプル デザインを参照してください。



ヒント: 『In-System IBERT LogiCORE IP 製品ガイド』(PG246) の FAQ を必ずお読みください。この IP をデザインに統合するときに発生する可能性のある問題に対する推奨事項が一部リストされています。

4. デザインを合成およびインプリメントします。

ハードウェアでのシリアル I/O デザインのデバッグ

IBERT コアをインプリメントしたら、ランタイム シリアル I/O 解析機能を使用して、ハードウェア上でデザインをデバッグできます。シリアル I/O 解析機能を使用してアクセスできるのは、IBERT コア v3.0 以降のみです。

Vivado シリアル I/O 解析を使用したデザインのデバッグ

Vivado[®] シリアル I/O 解析機能は、デザインに含まれる IBERT デバッグ IP コアにアクセスするために使用します。Vivado シリアル解析機能を使用するには、Flow Navigator で [PROGRAM AND DEBUG] → [Open Hardware Manager] をクリックします。

デザインのデバッグ手順は、次のとおりです。

1. ハードウェア ターゲットに接続し、FPGA を .bit ファイルでプログラムします。
2. リンクを作成します。
3. リンク設定を変更してステータスを確認します。
4. 必要に応じてスキャンを実行します。

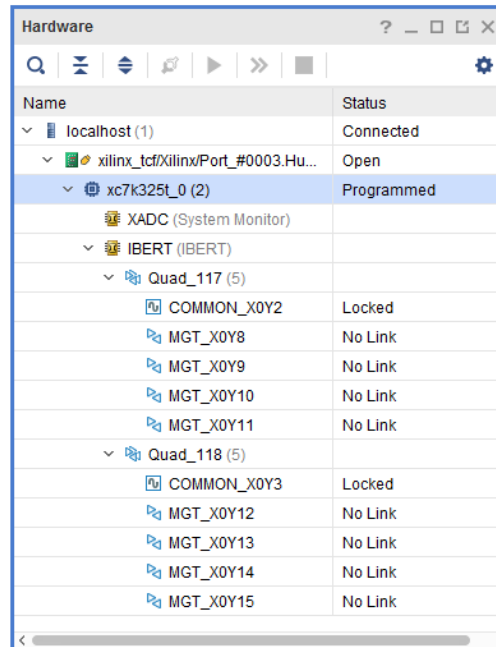
ハードウェア ターゲットに接続してデバイスをプログラム

デバッグの前に FPGA デバイスをプログラムする手順は、「FPGA のプログラム」で説明されている手順と同じです。IBERT コアを含む .bit ファイルでデバイスをプログラムすると、[Hardware] ウィンドウにデバイスのスキャンで検出された IBERT コアのコンポーネント (RTL インスタンス名はカッコ内に含有) が表示されます。



重要: UltraScale および UltraScale+ デザイン用の In-System IBERT IP を使用したデザインでは、[Hardware] ウィンドウに In-System IBERT コアが表示されます。

図 169: IBERT コアが表示された [Hardware] ウィンドウ



関連情報

デバイスのプログラム

リンクおよびリンク グループの作成

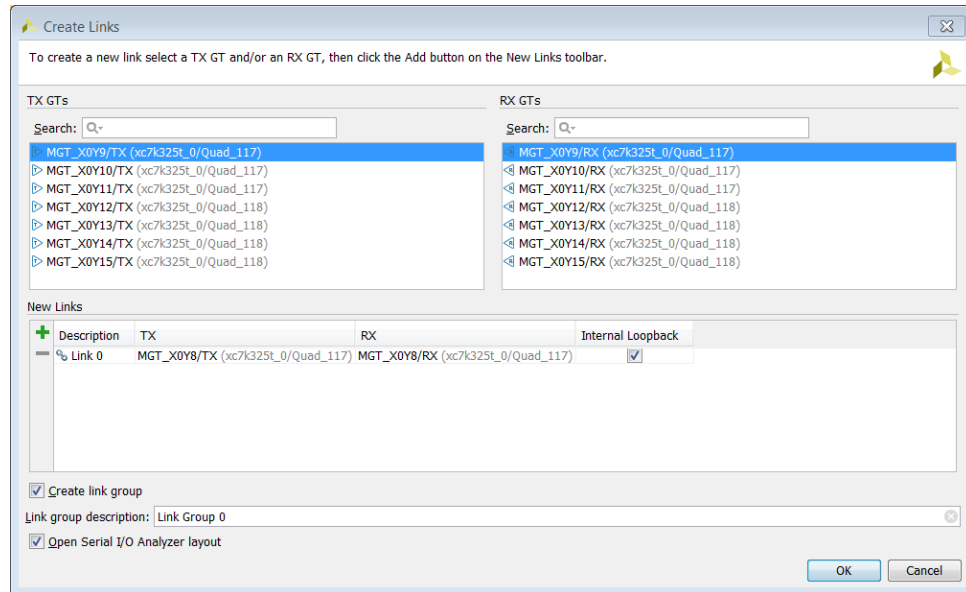
デザインに含まれる IBERT コアは、[Hardware] ウィンドウのターゲット デバイスの下に表示されます。コアが表示されない場合は、デバイスを右クリックして [Refresh Hardware] をクリックします。FPGA が再度スキャンされ、[Hardware] ウィンドウの表示が更新されます。

注記: FPGA デバイスをプログラムまたは更新しても IBERT コアが表示されない場合は、デバイスが正しい .bit ファイルでプログラムされているか、インプリメント済みデザインに IBERT v3.0 コアが含まれているかを確認してください。

Vivado シリアル I/O 解析機能は、リンクという概念に基づいて構築されています。リンクはボード上のチャネルと似ており、トランスミッターおよびレシーバーが含まれます。トランスミッターおよびレシーバーは、同じ GT、同じデバイス、または同じアーキテクチャである場合とそうでない場合があります。リンクはトランスミッターとレシーバーの両方に関連付けられる必要があるため、外部パターン ジェネレーターの 1 つの GT レシーバーへの接続はサポートされていません。1 つまたは複数のリンクを作成するには、Vivado の [Links] タブで [Create Links] ボタンをクリックするか、右クリックして [Create Links] をクリックします。[Create Links] ダイアログ ボックスが表示されます。

IBERT コアが検出されると、ハードウェア マネージャーにリンクが存在しないことを示す緑のバナーが上部に表示されます。[Create Links] をクリックして次の図のようなダイアログ ボックスを開きます。

図 170: [Create Links] ダイアログ ボックス



リストから TX または RX を選択します。[Search] フィールドに文字列を入力すると、リストをフィルターできます。["+"] (+) ボタンをクリックしてリンクをリストに追加します。必要なすべてのリンクに対して同じ操作を実行します。



重要: 各 TX または RX エンドポイントは、1 つのリンクにのみ含めることができます。

リンクはリンク グループに含めることもできます。デフォルトでは、新しいリンクすべてがグループ化されます。リンクをグループに追加しない場合は、[Create link group] をオフにします。リンク グループの名前は、[Link group description] で指定します。

[Links] ウィンドウでのリンク設定の表示と変更

リンクを作成すると、[Links] ウィンドウに追加されます。[Links] ウィンドウでは、リンクの設定を変更したり、ステータスを確認したりできます。

図 171: [Links] ウィンドウ

Tcl Console Messages Serial I/O Links x Serial I/O Scans									
Name	TX	RX	Status	Bits	Errors	BER	BERT Reset	TX Pattern	R
Ungrouped Links (0)									
Link Group SMA (1)							Reset	PRBS 7-bit	P
Link 0	MGT_X0Y8/TX	MGT_X0Y8/RX	7.988 Gbps	1.343E12	2.645E11	1.969E-1	Reset	PRBS 7-bit	P
Link Group Internal...							Reset	PRBS 7-bit	P
Link 1	MGT_X0Y9/TX	MGT_X0Y9/RX	7.987 Gbps	3.805E12	2.079E12	5.465E-1	Reset	PRBS 7-bit	P
Link 2	MGT_X0Y10/TX	MGT_X0Y10/RX	7.988 Gbps	3.805E12	2.175E12	5.715E-1	Reset	PRBS 7-bit	P

[Links] ウィンドウの各行は、各リンクを示します。一般的なステータスおよび有益なステータスはデフォルトでイネーブルになっており、リンクの状態をすばやく確認できます。次の表に、[Links] ウィンドウで表示可能な設定を示します。

表 26: [Links] ウィンドウに表示可能な設定

[Links] ウィンドウの列名	説明
Name	リンク名
TX	トランスミッターの GT ロケーション
RX	レシーバーの GT ロケーション
Status	リンクされている場合 (予測どおりの入力 RX データ) は計測されたライン レート、リンクされていない場合は「No Link」と表示されます。
Bits	受信されたビット数。
Errors	レシーバーでのビット エラー数。
BER	ビット エラー率 = (1 + エラー数)/(ビット数)。
BERT Reset	受信ビット カウンターおよびエラー カウンターをリセットします。
RX Pattern	レシーバーで受信されるパターンを選択します。
TX Pattern	トランスミッターで送信されるパターンを選択します。
TX Pre-Cursor	トランスミッターのプリカーソル エンファシスを選択します。
TX Post-Cursor	トランスミッターのポストカーソル エンファシスを選択します。
TX Diff Swing	トランスミッターの差動振幅値を選択します。
DFE Enabled	レシーバーで判定帰還型イコライザー (DFE) をイネーブルにするかどうかを選択します。使用できないアーキテクチャもあります。
Inject Error	送信パスに 1 つのビット エラーを挿入します。
TX Reset	トランスミッターをリセットします。
RX Reset	レシーバーおよび BERT カウンターをリセットします (「BERT Reset」を参照)。
Loopback Mode	レシーバー GT のループバック モードを選択します。 警告: この値を変更すると、システム トポロジによってはリンク ステータスに影響することがあります。
Termination Voltage	レシーバーの終端電圧を選択します。
RX Common Mode	レシーバーの RX 同相設定を選択します。
TXUSERCLK Freq	計測された TXUSERCLK 周波数を MHz で示します。
TXUSERCLK2 Freq	計測された TXUSERCLK2 周波数を MHz で表示します。
RXUSERCLK Freq	計測された RXUSERCLK 周波数を MHz で表示します。
RXUSERCLK2 Freq	計測された RXUSERCLK2 周波数を MHz で示します。
TX Polarity Invert	送信データの極性を反転します。
RX Polarity Invert	受信データの極性を反転します。

リンク グループに含まれるすべてのリンクのプロパティを変更するには、リンク グループ行で設定を変更します。たとえば、Link Group 0 行の [TX Pattern] を [PRBS 7-bit] に変更すると、このリンク グループに含まれるすべてのリンクの [TX Pattern] が [PRBS 7-bit] に変更されます。グループ内のすべてのリンクの設定が同じでない場合、リンク グループ行のプロパティ列に「Multiple」と表示されます。

In-System IBERT IP がデザインで使用される場合、リンク設定の一部のオプションしか使用されません。次の表は、使用されるリンク設定を示しています。

表 27: In-System IBERT の [Link] ウィンドウ設定

[Links] ウィンドウの列名	説明
TX	トランスミッターの GT ロケーション

表 27: In-System IBERT の [Link] ウィンドウ設定 (続き)

[Links] ウィンドウの列名	説明
RX	レシーバーの GT ロケーション
TX Pre-Cursor	トランスミッターのプリカーソル エンファシスを選択します。
TX Post-Cursor	トランスミッターのポストカーソル エンファシスを選択します。
TX Diff Swing	トランスミッターの差動振幅値を選択します。
DFE Enabled	レシーバーで判定帰還型イコライザー (DFE) をイネーブルにするかどうかを選択します。使用できないアーキテクチャもあります。

リンク スキャンの作成と実行

リンクのマージンを解析するには、ザイリンクス 7 シリーズ FPGA トランシーバー専用のアイ スキャン ハードウェアを使用してリンクのスキャンを実行すると有益です。Vivado シリアル I/O 解析機能では、リンク スキャンを定義、実行、保存、および呼び出しできます。

スキャンはリンク上で実行されます。スキャンを作成するには、[Link] ウィンドウでリンクを選択し、右クリックして [Create Scan] をクリックするか、[Link] ウィンドウ ツールバーの [Create Scan] ボタンをクリックします。[Create Scan] ダイアログ ボックスが開きます。[Create Scan] ダイアログ ボックスには、次の表に示すスキャンの実行に関する設定が表示されます。

生成できるスキャン タイプは 2D Eyescan または 1D Bathtub Plot の 2 タイプです。これらのスキャンのどちらでも、次の図に示す [Create Scan] ダイアログ ボックスで指定した設定が使用されます。ダイアログ ボックスの [Scan type] フィールドでは、生成するスキャンのタイプを指定できます。

図 172: [Create Scan] ダイアログ ボックス

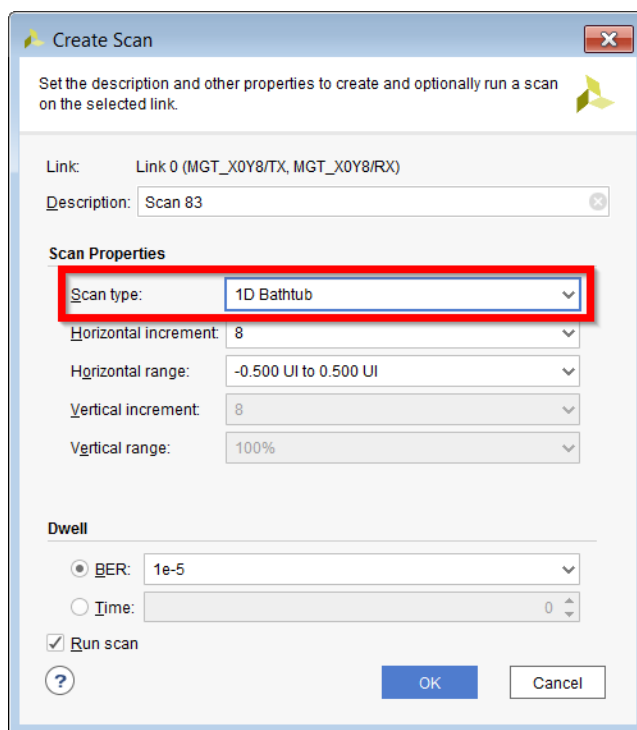


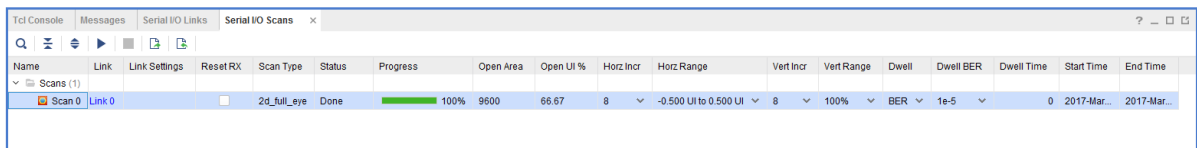
表 28: スキャン設定

スキャン設定	説明
Description	スキャンのユーザー定義名。
Scan Type	実行するスキャンのタイプ。2D Eyescan または 1D Bathtub plot を指定できます。
Horizontal Increment	水平コードをスキップすることにより、解像度を下げる代わりに速度を向上します。
Horizontal Range	水平範囲を削減して速度を向上します。デフォルトでは、アイ全体がスキャンされます (アイの中心に対して -1/2 ユニットインターバルから +1/2 まで)。
Vertical Increment	垂直コードをスキップすることにより、解像度を下げる代わりに速度を向上します。
Vertical Range	垂直範囲を削減して速度を向上します。デフォルトでは、アイ全体がスキャンされます。
Dwell BER	チャートの各点は特定の時間スキャンされます。このオプションでは、ビット エラー率 (BER) を選択することにより、スキャン深さを選択します。
Dwell Time	時間 (秒) を入力することにより、スキャン深さを選択します。 [Dwell Time] 設定は、In-System IBERT IP を使用するデザインではサポートされません。

デフォルトでは、スキャンを作成すると実行されます。スキャンを実行せずに定義だけする場合は、[Run Scan] チェック ボックスをオフにします。

スキャンが作成されても実行されない場合は、[Scans] ウィンドウでスキャンを右クリックして [Run Scan] をクリックします。スキャンを実行中にスキャンを停止するには、[Scans] ウィンドウでスキャンを右クリックして [Stop Scan] をクリックするか、ツールバーの [Stop Scan] ボタンをクリックします。

図 173: [Scans] ウィンドウ



Name	Link	Link Settings	Reset RX	Scan Type	Status	Progress	Open Area	Open UI %	Horiz Incr	Horiz Range	Vert Incr	Vert Range	Dwell	Dwell BER	Dwell Time	Start Time	End Time
Scan 0	Link 0			2d_full_eye	Done	100%	9600	66.67	8	-0.500 UI to 0.500 UI	8	100%	BER	1e-5	0	2017-Mar...	2017-Mar...

リンク スイープの作成と実行

リンクのマーzinを解析するには、異なる MGT 設定を使用してリンク スキャンを複数回実行すると有益な場合があります。これにより、どの設定が最適かがわかります。Vivado シリアル I/O 解析機能では、リンク スイープ (リンク スキャンのコレクション) を定義、実行、保存、および呼び出しできます。

スイープはリンク上で実行されます。スイープを作成するには、[Link] ウィンドウでリンクを選択し、右クリックして [Create Sweep] をクリックするか、[Link ウィンドウ ツールバー] の [Create Sweep] ボタンをクリックします。[Create Sweep] ダイアログ ボックスが表示されます。このダイアログ ボックスは [Create Scan] ダイアログ ボックスに似ていますが、どのプロパティをスイープするか、どのようにスイープするかを定義するオプションが含まれます。

図 174: [Create Sweep] ダイアログ ボックス

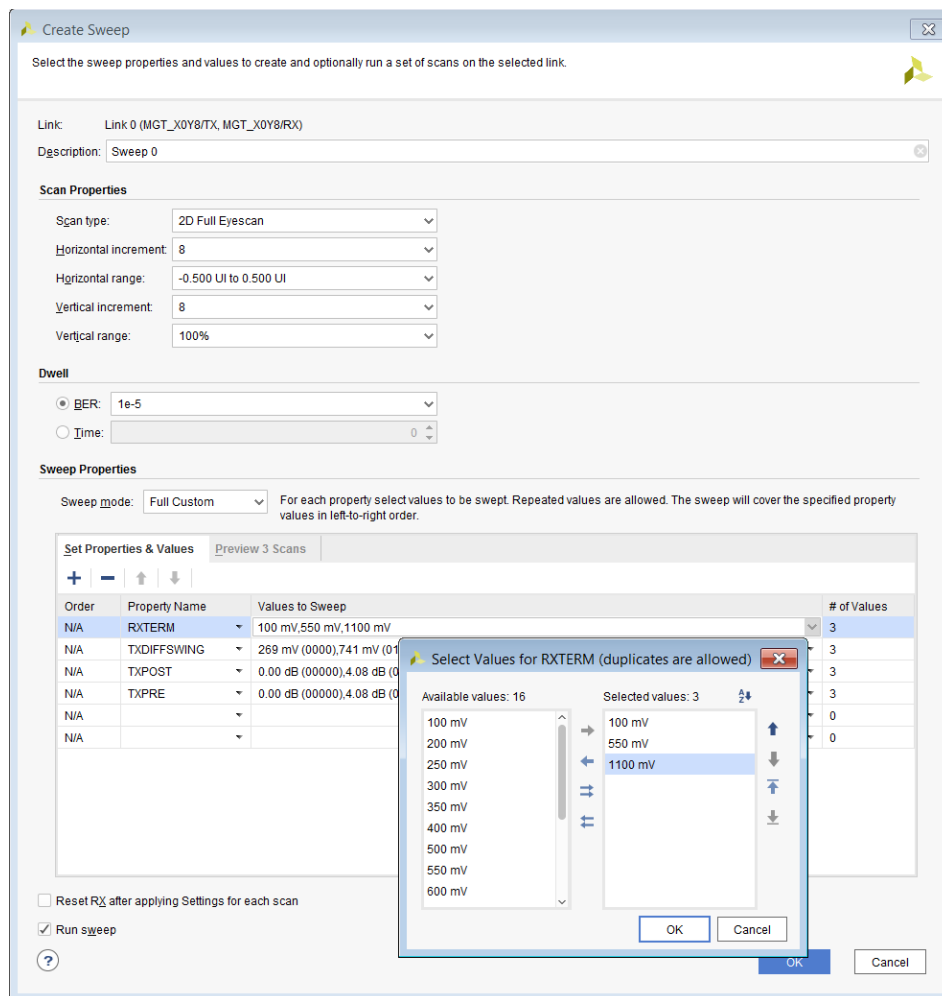
表 29: スイープ設定

スイープ設定	説明
説明	スイープのユーザー定義名。
Scan Type	実行するスキャンのタイプ。2D Eyescan または 1D Bathtub plot を指定できます。
Horizontal Increment	水平コードをスキップすることにより、解像度を下げる代わりに速度を向上します。
Horizontal Range	水平範囲を削減して速度を向上します。デフォルトでは、アイ全体がスキャンされます (アイの中心に対して -1/2 ユニット インターバルから +1/2 まで)。
Vertical Increment	垂直コードをスキップすることにより、解像度を下げる代わりに速度を向上します。
Vertical Range	垂直範囲を削減して速度を向上します。デフォルトでは、アイ全体がスキャンされます。
Dwell BER	チャートの各点は特定の時間スキャンされます。このオプションでは、ビット エラー率 (BER) を選択することにより、スキャン深さを選択します。
Dwell Time	時間 (秒) を入力することにより、スキャン深さを選択します。
Sweep Mode	実行するスイープのタイプ。[Semi Custom]、[Full Custom]、[Exhaustive] のいずれかを選択します。

これらの設定を選択したら、スイープ プロパティを設定します。リンクの書き込み可能なプロパティはすべてスイープ可能です。プロパティを追加するには、左側の [+] ボタンをクリックすると、表に新たな行が追加されます。スイープするプロパティを選択するには、[Property Name] セルをクリックします。

値を変更するには、[Values to Sweep Cell] セルのドロップダウンをクリックし、スイープする値を選択します。プロパティに列挙値が含まれない場合は、表示されるテキスト エリアの各行に 1 つずつ 16 進数値を入力します。

図 175: スイープ セルへの値の入力



- 次の図に示すように [Semi Custom] を選択している場合、選択したプロパティのすべての組み合わせが 1 つのスキャン用に定義され、そのスキャンがスイープ プロパティに基づいて実行されます。実行されるスイープの数およびその順序は、[Preview & Scans] タブをクリックするとプレビューできます。
- [Full Custom] の場合、リストされる各プロパティの最初の選択肢が最初のスキャンに使用され、2 つ目の選択肢が 2 つ目のスキャンに使用されるようになります。プロパティの 1 つの選択肢がほかのプロパティよりも少ない場合は、最後の選択肢がその後のすべてのスキャンに使用されます。同じプロパティの選択でスイープ モードが [Full Custom] の場合は、3 つのスキャンのみが実行されます。

図 176: [Sweep Properties] エリア

Create Sweep

Select the sweep properties and values to create and optionally run a set of scans on the selected link.

Link: Link 0 (MGT_X0Y8/TX, MGT_X0Y8/RX)

Description: Sweep 1

Scan Properties

Scan type: 2D Full Eyescan

Horizontal increment: 8

Horizontal range: -0.500 UI to 0.500 UI

Vertical increment: 8

Vertical range: 100%

Dwell

☒ BER: 1e-5

☐ Time: 0

Sweep Properties

Sweep mode: Full Custom

For each property select values to be swept. Repeated values are allowed. The sweep will cover the specified property values in left-to-right order.

Order	Property Name	Values to Sweep	# of Values
N/A	RXTERM	100 mV, 550 mV, 1100 mV	3
N/A	TXDIFFSWING	269 mV (0000), 741 mV (0111), 1119 mV (1111)	3
N/A	TXPOST	0.00 dB (00000), 4.08 dB (01111), 12.96 dB (11111)	3
N/A	TXPRE	0.00 dB (00000), 4.08 dB (01111), 6.02 dB (11111)	3

☐ Reset RX after applying Settings for each scan

☒ Run sweep

OK Cancel

- [Exhaustive] の場合、[Values to Sweep] は変更できなくなり、プロパティに対してすべての値が選択されます。

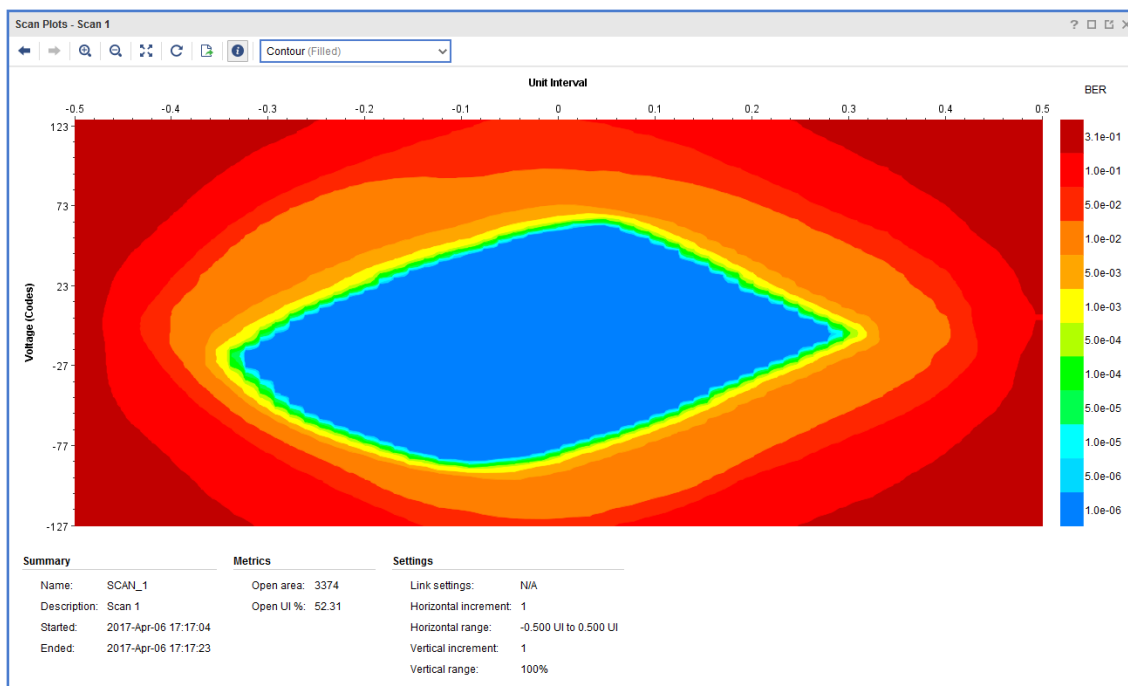
すべてのプロパティを設定したら、[Run Sweep] をオンにしたままにすると、各スキャンが順に実行されます。[OK] をクリックすると、スキャンのリストが [Scan] ウィンドウに表示されます。

スイープ中の進捗状況は [Scan] ウィンドウで確認できます。最新のスキャンの結果が表示されます。

スキャン プロットの表示とナビゲート

スキャンを作成すると、そのスキャンの [Scan Plots] ウィンドウが表示されます。2D アイスキャンでは、プロットは BER 値のヒート マップです。

図 177: [Scan Plots] ウィンドウ



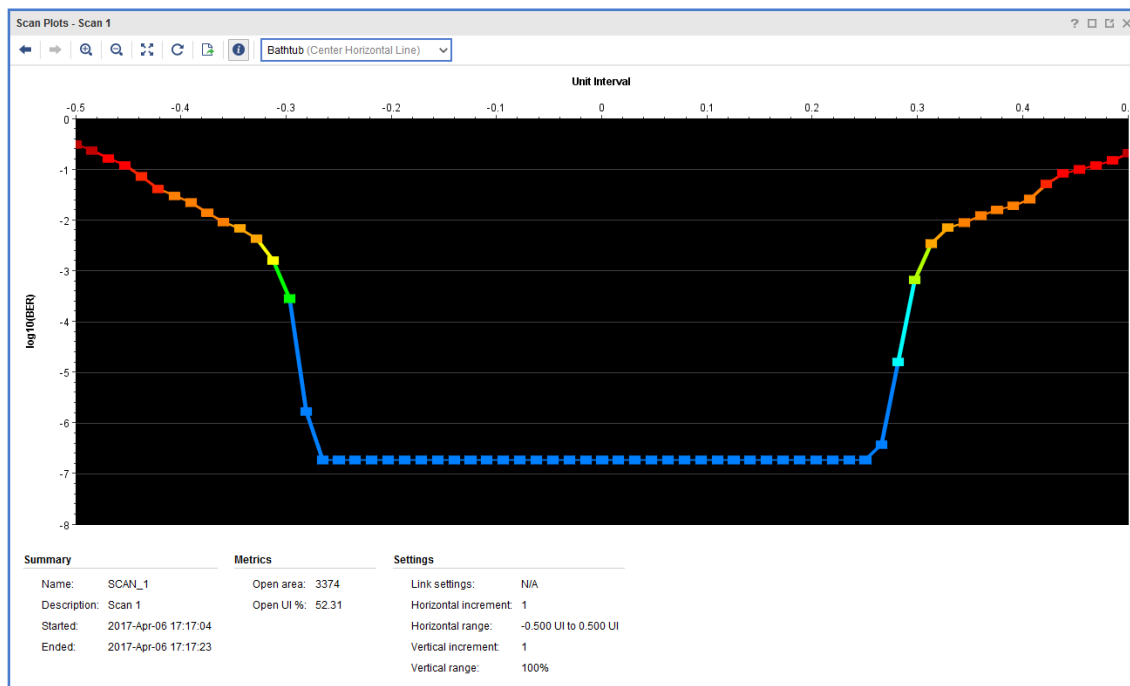
Vivado IDE のほかのチャートおよび表示と同様に、[Scan Plots] ウィンドウも次のように拡大/縮小できます。

- エリアの拡大: 拡大表示するエリアを左上から右下に向かってドラッグ
- 全体表示: 右下から左上へ向かってドラッグ
- 拡大: 右上から左下へ向かってドラッグ
- 縮小: 左下から右上へ向かってドラッグ

マウス カーソルをプロット上に置くと、ツール ヒントに現在の水平コード、垂直コード、およびスキャンされた BER 値が表示されます。プロット タイプを変更するには、プロット ビューで [Plot Type] ボタンをクリックし、[Show Contour (filled), Show Contour (lines), Bathtub (Center Horizontal Line), and Heat Map]、[Show Contour (lines)]、[Bathtub (Center Horizontal Line)] および [Heat Map] をクリックします。

スキャン プロットの下サマリには、スキャン設定とスキャンが実行された時間などの基本情報が表示されます。2D アイスキャンでは、エラーのないスキャンのピクセル数が算出され (水平および垂直インクリメントを考慮)、結果が [Open Area] として表示されます。[Scans] ウィンドウの内容はデフォルトで [Open Area] 列順にリストされるので、オープン エリアが最も大きいスキャンが一番上に表示されます。次の図は前の図と同じスキャンの Bathtub Plot です。

図 178: Bathtub Plot



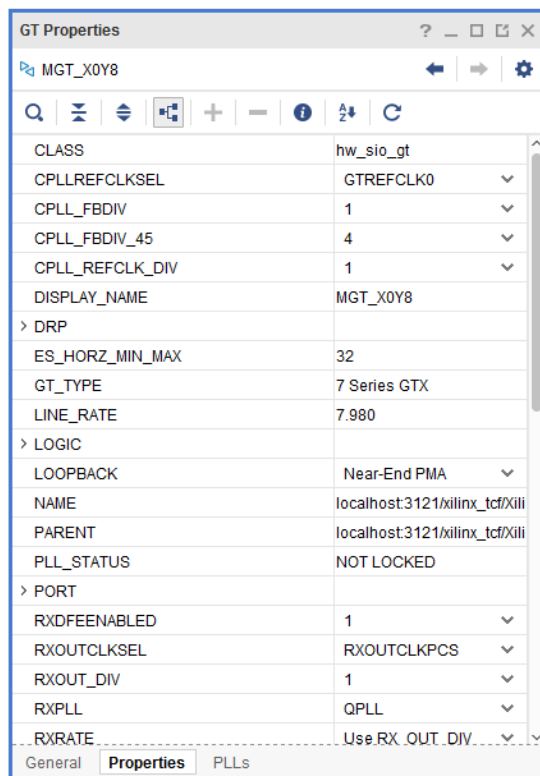
出力結果のファイルへの保存

部分的またはフル 2D アイスキャンによるスキャン データが存在する場合、[Scans] ウィンドウで [Write Scan] ボタンをクリックすると、スキャン結果を CSV ファイルに保存できます。スキャン結果が CSV 形式のファイルに保存され、スキャン プロットを複製するブロックに BER 値が含まれます。

[Properties] ウィンドウ

[Hardware] ウィンドウで GT または COMMON ブロック、[Links] ウィンドウでリンク、または [Scans] ウィンドウでスキャンを選択すると、[Properties] ウィンドウにそのオブジェクトのプロパティが表示されます。GT および COMMON に対しては、これらのオブジェクトの属性、ポート、およびその他の設定がすべて表示されます。これらの設定は、[Properties] ウィンドウまたは Tcl コマンドで変更できます。プロパティには、読み出し専用で変更できないものもあります。

図 179: [Properties] ウィンドウ



シリアル I/O 解析の Tcl オブジェクトおよびコマンド

テスト中のハードウェアにアクセスするには、Tcl コマンドを使用できます。ハードウェアには、次の図に示す階層ファースト クラス Tcl オブジェクトがあります。

表 30: シリアル I/O 解析の Tcl オブジェクト

Tcl オブジェクト	説明
hw_sio_ibert	IBERT コアを参照するオブジェクト。各 IBERT オブジェクトには、1 つまたは複数の hw_sio_gt または hw_sio_common オブジェクトを関連付けることができます。
hw_sio_gt	1 つのサイリンクス ギガビット トランシーバー (GT) を参照するオブジェクト。
hw_sio_gtgroups	GT の論理グループを参照するオブジェクト。4 進数または 8 進数に設定できます。
hw_sio_common	COMMON ブロックを参照するオブジェクト。
hw_sio_tx	hw_sio_gt のトランスミッター側を参照するオブジェクト。TX 関連のポート、属性、ロジック プロパティのみが hw_sio_tx に挿入されます。
hw_sio_rx	hw_sio_gt のレシーバー側を参照するオブジェクト。RX 関連のポート、属性、ロジック プロパティのみが hw_sio_rx に挿入されます。
hw_sio_pll	hw_sio_gt または hw_sio_common オブジェクトの PLL オブジェクトを参照するオブジェクト。関連のポート、属性、ロジック プロパティのみが hw_sio_pll に挿入されます。
hw_sio_link	リンク、TX-RX ペアを参照するオブジェクト。 リンクには、RX のみまたは RX のみが含まれる場合もあります。
hw_sio_linkgroup	リンク グループを参照するオブジェクト。
hw_sio_scan	マージン解析スキャンを参照するオブジェクト。

ハードウェア マネージャー コマンドの詳細は、[Tcl Console] ウィンドウに「help -category hardware」と入力してください。

ハードウェアにアクセスする Tcl コマンド

次の表に、IBERT コアにアクセスするために使用する Tcl コマンドを示します。



重要: get_property または set_property コマンドを使用した場合、IBERT コアから情報を読み出したり IBERT コアに情報を書き込むことはできません。ハードウェアから情報を読み出すには refresh_hw_sio コマンド、ハードウェアに情報を書き込むには commit_hw_sio コマンドを使用します。

表 31: hw_server Tcl コマンドの説明

Tcl コマンド	説明
refresh_hw_sio	オブジェクトのプロパティ値を読み出します。ハードウェアを参照するすべての hw_sio オブジェクトに対して使用できます。
commit_hw_sio	プロパティの変更をハードウェアに書き込みます。ハードウェアを参照するすべての hw_sio オブジェクトに対して使用できます。

hw_sio_link の Tcl コマンド

次の表に、リンクに関連する Tcl コマンドすべての説明を含めます。

表 32: hw_sio_link の Tcl コマンド

Tcl コマンド	説明
create_hw_sio_link	指定の hw_sio_rx および hw_sio_tx から hw_sio_link オブジェクトを作成します。
remove_hw_sio_link	指定したリンクを削除します。
get_hw_sio_links	指定したオブジェクトの hw_sio_link のリストを取得します。

hw_sio_linkgroup の Tcl コマンド

次の表に、リンクグループに関連する Tcl コマンドすべての説明を含めます。

表 33: hw_sio_linkgroup の Tcl コマンド

Tcl コマンド	説明
create_hw_sio_linkgroup	hw_sio_link オブジェクトから hw_sio_linkgroup オブジェクトを作成します。
remove_hw_sio_linkgroup	指定したリンクグループを削除します。
get_hw_sio_linkgroups	指定したオブジェクトの hw_sio_linkgroup のリストを取得します。

hw_sio_scan の Tcl コマンド

次の表に、スキャンに関連する Tcl コマンドすべての説明を含めます。

表 34: hw_sio_scan の Tcl コマンドの説明

Tcl コマンド	説明
create_hw_sio_scan	スキャン オブジェクトを作成します。
remove_hw_sio_scan	スキャン オブジェクトを削除します。
run_hw_sio_scan	スキャンを実行します。
stop_hw_sio_scan	スキャンを停止します。
wait_on_hw_sio_scan	run_hw_sio_scan が完了するまで Tcl コンソール プロンプトでコマンドを実行できないようにします。
display_hw_sio_scan	部分的または完了したスキャンをスキャン プロットに表示します。
write_hw_sio_scan	スキャン データをファイルに記述します。
read_hw_sio_scan	スキャン データをファイルからスキャン オブジェクトに読み込みます。
get_hw_sio_scans	hw_sio_scan オブジェクトのリストを取得します。

オブジェクトを取得する Tcl コマンド

次の表に、シリアル I/O オブジェクトを取得するために使用する Tcl コマンドを示します。

表 35: オブジェクトを取得する Tcl コマンド

Tcl コマンド	説明
get_hw_sio_iberts	IBERT オブジェクトのリストを取得します。
get_hw_sio_gts	GT のリストを取得します。
get_hw_sio_commons	COMMON ブロックのリストを取得します。
get_hw_sio_txs	トランスミッターのリストを取得します。
get_hw_sio_rxs	レシーバーのリストを取得します。
get_hw_sio_plls	PLL のリストを取得します。
get_hw_sio_links	リンクのリストを取得します。
get_hw_sio_linkgroups	リンク グループのリストを取得します。
get_hw_sio_scans	スキャンのリストを取得します。

IBERT を測定する Tcl コマンドの使用

次のようなシステムにアクセスする Tcl コマンド スクリプトの例を示します。

- localhost:3121 上の hw_server を介してアクセス可能な 1 つの KC705 ボードの Digilent JTAG-SMT1 ケーブル (シリアル番号 12345) が使用されている
- KC705 ボード上の XC7K325T デバイスで実行されているデザインに IBERT コアが 1 つ含まれている
- IBERT コアで Quad 117 および Quad 118 がイネーブルになっている

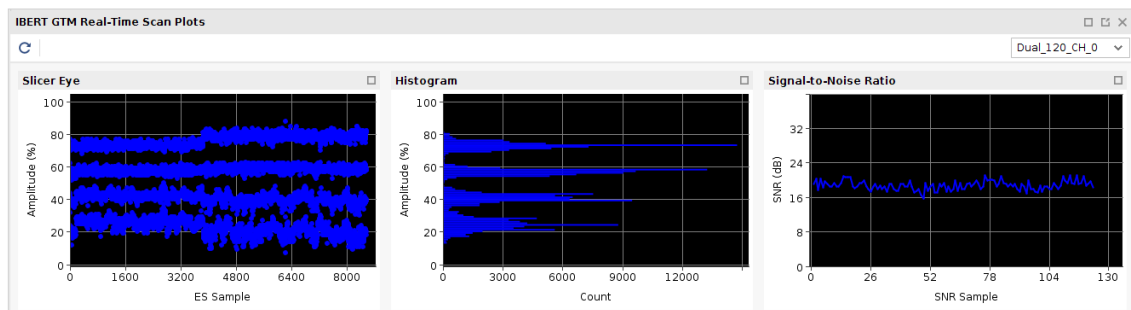
Tcl コマンド スクリプト例

```
# Connect to the Digilent Cable on localhost:3121
connect_hw_server -url localhost:3121
current_hw_target [get_hw_targets */digilent_plugin/SN:12345]
open_hw_target
# Program and Refresh the XC7K325T Device
current_hw_device [lindex [get_hw_devices] 0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices] 0]
set_property PROGRAM.FILE {C:/design.bit} [lindex [get_hw_devices] 0]
program_hw_devices [lindex [get_hw_devices] 0]
refresh_hw_device [lindex [get_hw_devices] 0]
# Set Up Link on first GT
set tx0 [lindex [get_hw_sio_txs] 0]
set rx0 [lindex [get_hw_sio_rxs] 0]
set link0 [create_hw_sio_link $tx0 $rx0]
set_property DESCRIPTION {Link 0} [get_hw_sio_links $link0]
# Set link to use PCS Loopback, and write to hardware
set_property LOOPBACK "Near-End PCS" $link0
commit_hw_sio $link0
# Create, run, display and save scan
set scan0 [create_hw_sio_scan 2d_full_eye [get_hw_sio_rxs -of $link0]]
run_hw_sio_scan $scan0
display_hw_sio_scan $scan0
write_hw_sio_scan "scan0.csv" $scan0
```

スライサー アイ、ヒストグラム、および信号対ノイズ比の表示 (GTM トランシーバーのみ)

GTM レシーバーは ADC ベースなので、以前のトランシーバー ファミリ (GTH、GTY トランシーバーなど) で使用されていた従来のアイスキャンは使用できません。そのため、GTM 用の IBERT ダッシュボードには、従来の [Scan Plots] ウィンドウの代わりに、スライサー アイ、ヒストグラム、および信号対ノイズ比の 3 つのプロットが表示されます。

図 180: スライサー アイ、ヒストグラム、および信号対ノイズ比プロット



リンクが作成されると、そのリンクのスライサー アイ、ヒストグラム、および信号対ノイズ比が表示されます。複数のリンクを作成した場合、右上のドロップダウン リストから必要な DUAL および Channel を選択するとアクティブ リンクを変更できます。

スライサー アイおよび GTM トランシーバー アーキテクチャの詳細は、『UltraScale FPGA GTM トランシーバー ユーザー ガイド』 (UG581: [英語版](#)、[日本語版](#)) を参照してください。

IBERT GTM の詳細は、『IBERT for UltraScale GTM Transceivers LogiCORE IP 製品ガイド』 ([PG342](#)) を参照してください。

デバイス コンフィギュレーション ビット ストリーム設定

7 シリーズ デバイスのビットストリーム設定

次の表に、Vivado® ツールの `set_property <Setting> <Value> [current_design]` Tcl コマンドで使用可能な 7 シリーズ デバイスのデバイス コンフィギュレーション設定を示します。

注記: BPI のビットストリーム設定は、Spartan®-7 デバイスでは使用できません。

表 36: 7 シリーズ デバイスのビットストリーム設定

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1、2、3、4	BPI コンフィギュレーションをフラッシュ デバイスのページ モード動作のタイミングと同期させる際に使用し、最初のページの有効読み出しのサイクル数を設定します。このオプションは、BPI_page_size を 4 または 8 に設定している場合にのみ有効です。
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1、4、8	BPI コンフィギュレーションのページ サイズを指定します。これは、フラッシュ メモリでページごとに必要な読み出し数に対応します。
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable、Type1、Type2	BPI フラッシュ デバイスの異なるタイプの BPI 同期コンフィギュレーション モードを設定します。 Disable (デフォルト): 同期コンフィギュレーション モードをディスエーブルにします。 Type1: 同期コンフィギュレーション モードをイネーブルにし、Micron G18(F) ファミリをサポートする設定を使用します。 Type2: 同期コンフィギュレーション モードをイネーブルにし、Micron (Numonyx) P30 および P33 ファミリをサポートする設定を使用します。
BITSTREAM.CONFIG.CCLKPIN	Pullup	Pullup、Pullnone	Cclk ピンに内部プルアップを追加します。Pullnone に設定すると、プルアップは使用されません。
BITSTREAM.CONFIG.CONFIGFALLBACK	Disable	Disable、Enable	コンフィギュレーションでエラーが発生した場合にデフォルトのビットストリームを読み込むかどうかを指定します。 マルチブート ソリューション設定 BITSTREAM.CONFIG.NEXT_CONFIG_ADDR を使用すると、BITSTREAM.CONFIG.FALLBACK が Enable になります。 フォールバック マルチブートは、Virtex-7 HT デバイスではサポートされません。
BITSTREAM.CONFIG.CONFIGRATE	3	3、6、9、12、16、22、26、33、40、50、66	マスター モードでコンフィギュレーションする場合、ビットストリームの生成でコンフィギュレーション クロック (Cclk) の生成に内部オシレーターが使用されます。このオプションを使用すると、Cclk のレートを選択できます。

表 36: 7 シリーズ デバイスのビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired、Continuous、Quiet	デジタル制御インピーダンス (DCI) 回路で DCI IOSTANDARD のインピーダンス一致をアップデートする頻度を指定します。
BITSTREAM.CONFIG.DONEPIN ¹	Pullup	Pullup、Pullnone	DONE ピンに内部プルアップを追加します。Pullnone に設定すると、プルアップは使用されません。このオプションは、外部プルアップ抵抗を DONE ピンに接続する場合にのみ使用してください。このオプションを使用しない場合、内部プルアップ抵抗が自動的に接続されます。
BITSTREAM.CONFIG.EXTMASTERCCLK_EN	Disable	Disable、Div-1、Div-2、Div-4、Div-8	すべてのマスター モードで外部クロックをコンフィギュレーション クロックとして使用できるようにします。外部クロックは、多目的 EMCCLK ピンに接続する必要があります。
BITSTREAM.CONFIG.INITPIN ¹	Pullup	Pullup、Pullnone	INIT ピンにプルアップ抵抗を追加するか、未接続のままにするかを指定します。
BITSTREAM.CONFIG.INITSIGNALSERROR	Enable	Enable、Disable	Enable の場合、コンフィギュレーション エラーが検出されると INIT_B ピンが 0 にアサートされます。
BITSTREAM.CONFIG.M0PIN ¹	Pullup	Pullup、Pulldown、Pullnone	M0 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M0 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.M1PIN ¹	Pullup	Pullup、Pulldown、Pullnone	M1 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M1 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.M2PIN ¹	Pullup	Pullup、Pulldown、Pullnone	M2 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M2 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	なし	文字列	マルチブート セットアップの次のコンフィギュレーションの開始アドレスを設定します。これは、WBSTAR レジスタに保存されます。
BITSTREAM.CONFIG.NEXT_CONFIG_REBOOT	Enable	Enable、Disable	Disable に設定すると、.bit ファイルから IPROG コマンドが削除されます。マルチブート セットアップでのマルチブート イメージにジャンプするのではなく、パワーアップ時にゴールデン イメージが読み込まれるようになります。
BITSTREAM.CONFIG.PERSIST	No	No、Yes	コンフィギュレーション後に多目的コンフィギュレーションピンへのコンフィギュレーション ロジック アクセスを保持します。主にコンフィギュレーション後のリードバック アクセス用に SelectMAP ポートを保持するために使用されますが、どのコンフィギュレーション モードでも使用できます。JTAG コンフィギュレーションでは、JTAG ポートは専用ポートなので、PERSIST は必要ありません。PERSIST と ICAP を同時に使用することはできません。 詳細は、ユーザー ガイドを参照してください。PERSIST は、SelectMAP コンフィギュレーション ピンを使用するリードバックおよび Dynamic Function eXchange に必要で、SelectMAP またはシリアル モードを使用している場合に使用する必要があります。
BITSTREAM.CONFIG.REVISIONSELECT	00	00、01、10、11	次のウォーム ブートのウォーム ブート開始アドレス (WBSTAR) レジスタの RS[1:0] 設定の内部値を指定します。
BITSTREAM.CONFIG.REVISIONSELECT_TRISTATE	Disable	Disable、Enable	ウォーム ブートのウォーム ブート開始アドレス (WBSTAR) のオプションを設定することにより、RS[1:0] トライステートをイネーブルにするかどうかを指定します。 0: RS トライステートをイネーブル 1: RS トライステートをディスエーブル

表 36: 7 シリーズ デバイスのビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.SELECTMAPABORT	Enable	Enable、Disable	SelectMAP モードのアボートシーケンスをイネーブルまたはディスエーブルにします。Disable に設定すると、デバイスピンのアボートシーケンスは無視されます。
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No、Yes	SPI 32 ビット アドレス形式をイネーブルにします。この形式は、256 Mb 以上のストレージを含む SPI デバイスで必要です。
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE、1、2、4	サードパーティ SPI フラッシュ デバイスからのマスター SPI コンフィギュレーションに対して、SPI バスをデュアル (x2) またはクワッド (x4) モードに設定します。
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No、Yes	FPGA で SPI データのキャプチャに立ち下がりエッジを使用するよう設定します。これによりタイミングマージンが向上し、コンフィギュレーションのクロックレートが上がる可能性があります。
BITSTREAM.CONFIG.TCKPIN ¹	Pullup	Pullup、Pulldown、Pullnone	TCK ピン、JTAG テストクロックにプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TDIPIN ¹	Pullup	Pullup、Pulldown、Pullnone	TDI ピン、JTAG 命令および JTAG レジスタへのシリアルデータ入力すべてに、プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TDOPIN ¹	Pullup	Pullup、Pulldown、Pullnone	TDO ピン、JTAG 命令およびデータレジスタへのシリアルデータ出力すべてに、プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TIMER_CFG	なし	8 桁の 16 進文字列	コンフィギュレーションモードでのウォッチドッグタイマーをイネーブルにして値を設定します。このオプションは、TIMER_USR と同時に使用することはできません。
BITSTREAM.CONFIG.TIMER_USR	0x00000000	8 桁の 16 進文字列	コンフィギュレーションモードでのウォッチドッグタイマーをイネーブルにして値を設定します。このオプションは、TIMER_CFG と同時に使用することはできません。
BITSTREAM.CONFIG.TMSPIN ¹	Pullup	Pullup、Pulldown、Pullnone	TMS ピン、TAP コントローラーへのモード入力信号にプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。TAP コントローラーは、JTAG の制御ロジックとして使用されます。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pulldown、Pullup、Pullnone	未使用の SelectIO ピン (IOB) にプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。コンフィギュレーション専用ピンには適用されません。コンフィギュレーション専用ピンのリストは、アーキテクチャによって異なります。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	8 桁の 16 進文字列	インプリメンテーションのリビジョンを特定します。ユーザー ID レジスタには、8 桁までの 16 進文字列を入力できます。
BITSTREAM.CONFIG.USR_ACCESS	なし	None、8 桁の 16 進文字列、TIMESTAMP	AXSS コンフィギュレーションレジスタに、8 桁の 16 進文字列またはタイムスタンプを記述します。タイムスタンプ値のフォーマットは、dddd MM yy hh mm ss (dddd = 日、MM = 月、yy = 年 (2000 年は 0000)、hh = 時、mm = 分、ss = 秒) です。このレジスタの内容は、FPGA デバイスにより USR_ACCESS プリミティブを介して直接アクセスできます。
BITSTREAM.ENCRYPTION.ENCRYPT	No	No、Yes	ビットストリームを暗号化します。

表 36: 7 シリーズ デバイスのビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT	bbram	bbram、efuse	使用する AES 暗号キーの場所を、バッテリー バックアップ式 RAM (BBRAM) または eFUSE レジスタのいずれかに指定します。 このプロパティは ENCRYPT オプションを Yes に設定している場合のみ使用可能です。
BITSTREAM.ENCRYPTION.HKEY	なし	16 進文字列	ビットストリーム暗号化の HMAC 認証キーを設定します。7 シリーズ デバイスには、ハードウェアにオンチップのビットストリーム キー付き HMAC (Hash Message Authentication Code) アルゴリズムがインプリメントされており、AES 復号化のみの場合よりセキュリティが強化されています。これらのデバイスでは、ビットストリームの読み込み、変更、遮断、コピーに AES と HMAC キーの両方が必要です。 このオプションを使用するには、ENCRYPT オプションを Yes に設定する必要があります。
BITSTREAM.ENCRYPTION.KEY0	なし	16 進文字列	ビットストリーム暗号化の AES 暗号キーを設定します。このオプションを使用するには、ENCRYPT オプションを Yes に設定する必要があります。
BITSTREAM.ENCRYPTION.KEYFILE	なし	文字列	入力暗号化ファイル (拡張子 .nky) の名前を指定します。このオプションを使用するには、ENCRYPT オプションを Yes に設定する必要があります。
BITSTREAM.ENCRYPTION.STAR_TCBC	なし	32 ビットの 16 進文字列	暗号文ブロック連鎖 (CBC) の開始値を設定します。
BITSTREAM.GENERAL.COMPRESS	False	True、False	ビットストリームの複数フレーム書き込み機能を使用し、ビットストリーム ファイル (.bit) ファイルだけでなく、ビットストリーム自体のサイズも縮小します。このオプションを使用しても、ビットストリームのサイズが縮小するとは限りません。
BITSTREAM.GENERAL.CRC	Enable	Enable、Disable	ビットストリームの巡回冗長検査 (CRC) 値の生成を制御します。Enable に設定すると、ビットストリームの内容に基づいて固有の CRC 値が算出されます。算出された CRC 値がビットストリームの CRC 値と一致しない場合は、デバイスはコンフィギュレーションされません。CRC がディスエーブルの場合、CRC 値の代わりに定数値がビットストリームに挿入され、デバイスで CRC 値は算出されません。 CRC デフォルト値は Enable ですが、BITSTREAM.ENCRYPTION.ENCRYPT の場合のみ Yes、CRC はディスエーブルです。
BITSTREAM.GENERAL.DEBUGBITSTREAM	No	No、Yes	デバッグ ビットストリームを生成します。デバッグ ビットストリームのサイズは、標準のビットストリームよりもかなり大きくなります。このオプションは、マスターおよびスレーブ シリアル コンフィギュレーションでのみ使用できます。バウンダリスキャンおよびスレーブ パラレル/SelectMAP では使用できません。デバッグ ビットストリームには、標準ビットストリームに加え、次の機能があります。 同期化ワードの後に LOUT レジスタに 32 個の 0 を書き込みます。 各フレームを個別に読み込みます。 各フレーム後に巡回冗長検査 (CRC) を実行します。 各フレーム後にフレーム アドレスを LOUT レジスタに書き込みます。
BITSTREAM.GENERAL.DISABLE_JTAG	No	No、Yes	コンフィギュレーション後に JTAG を介したバウンダリスキャン (BSCAN) ブロックへのアクセスをディスエーブルにします。
BITSTREAM.GENERAL.JTAG_XADC	Enable	Enable、Disable、StatusOnly	XADC への JTAG 接続をイネーブルまたはディスエーブルにします。

表 36: 7 シリーズ デバイスのビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.GENERAL.PERFRAMECRC	No	No, Yes	ビットストリームに一定間隔で CRC 値を挿入します。これらの値は入力されるビットストリームのインテグリティを検証して、コンフィギュレーション データがデバイスにロードされる前にエラー (INIT_B ピンおよび ICAP の PRERROR ポートに表示) を通知します。これはパーシャル ビットストリームに適していますが、Yes に設定すると、CRC 値がデバイス ストリーム全体を含め、すべてのビットストリームに挿入されます。
BITSTREAM.GENERAL.XADCENHANCEDLINEARITY	オフ	Off, On	INL が実際のアナログ パフォーマンスよりも悪くなるビルトイン デジタル キャリブレーション機能をディスエーブルにします。
BITSTREAM.READBACK.ACTIVERECONFIG	No	No, Yes	コンフィギュレーション中に GHIGH および GSR がアサートされないようにします。これは、アクティブ Dynamic Function eXchange 向上機能に必要です。
BITSTREAM.READBACK.ICAP_SELECT	Auto	Auto, Top, Bottom	上または下の ICAP ポートを選択します。
BITSTREAM.READBACK.READBACK	False	True, False	必要なリードバック ファイルを作成してリードバック機能を実行します。
BITSTREAM.READBACK.SECURITY	なし	None, Level1, Level2	リードバックおよびリコンフィギュレーションをディスエーブルにするかどうかを指定します。 Level1 に設定するとリードバックがディスエーブルになり、Level2 に設定するとリードバックとリコンフィギュレーションがディスエーブルになります。
BITSTREAM.READBACK.XADCPARTIALRECONFIG	Disable	Disable, Enable	Disable に設定すると、Dynamic Function eXchange 中も XADC が継続して機能します。Enable に設定すると、Dynamic Function eXchange 中は XADC はセーフ モードで機能します。
BITSTREAM.STARTUP.DONEPIPE	あり	Yes, No	CFG_DONE (DONE) ピンが High になり、最初のクロック エッジを待って、Done ステートに移動します。
BITSTREAM.STARTUP.DONE_CYCLE	4	4, 1, 2, 3, 5, 6, Keep	FPGA Done 信号をアクティブにするスタートアップ フェーズを選択します。DonePipe=Yes の場合、Done は遅延されます。
BITSTREAM.STARTUP.GTS_CYCLE	5	5, 1, 2, 3, 4, 6, Done, Keep	I/O バッファへの内部トライステート制御を解放するスタートアップ フェーズを選択します。
BITSTREAM.STARTUP.GWE_CYCLE	6	6, 1, 2, 3, 4, 5, Done, Keep	フリップフロップ、LUT RAM、およびシフトレジスタへの内部イネーブルをアサートするスタートアップ フェーズを選択します。BRAM もイネーブルにします。このスタートアップ フェーズの前は、ブロック RAM の書き込みおよび読み出しの両方がディスエーブルです。
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait, 0, 1, 2, 3, 4, 5, 6	DLL/DCM/PLL ロックまで待機するため、スタートアップ段階を選択します。NoWait に設定すると、スタートアップシーケンスは DLL/DCM/PLL がロックされるまで待機されません。

表 36: 7 シリーズ デバイスのビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto、NoWait、0、1、2、3、4、5、6	<p>デジタル制御インピーダンス (DCI) 一致信号がアサートされるまで待機するスタートアップ サイクルを指定します。DCI マッチは MATCH_CYCLE では開始しません。スタートアップシーケンスは DCI が一致するまでこのサイクルで待機します。DCI が一致するのにかかる時間にはさまざまな要素が影響するので、スタートアップ シーケンスが完了するのに必要な CCLK サイクル数は、同じシステムでも異なる場合があります。DONE が High になるまでコンフィギュレーション ソリューションで CCLK を駆動するのが理想的です。</p> <p>Auto に設定すると、write_bitstream によりデザインで DCI I/O 規格が検索されます。DCI 規格が存在する場合は write_bitstream で BITSTREAM.STARTUP.MATCH_CYCLE=2 が、存在しない場合は write_bitstream で BITSTREAM.STARTUP.MATCH_CYCLE=NoWait が使用されます。</p>
BITSTREAM.STARTUP.STARTUP_CLK	Cclk	Cclk、UserClk、JtagClk	<p>デバイスのコンフィギュレーション後の StartupClk シーケンスは、Cclk、ユーザー クロック、または JTAG クロックに同期させることができます。デフォルトは Cclk です。</p> <p>Cclk: FPGA デバイスで供給される内部クロックに同期します。</p> <p>UserClk: STARTUP シンボルの CLK ピンに接続されているユーザー定義信号に同期します。</p> <p>JtagClk: JTAG で供給されるクロックに同期します。このクロックにより、JTAG に制御ロジックを提供する TAP コントローラーの順序が決まります。</p> <p>Spartan-7 の 7s6/7s15 デバイスでは、STARTUPE2.CLK (UserClk) ユーザー スタートアップクロック ピンがサポートされません。</p>

注記:

1. ザイリンクスでは、専用コンフィギュレーション ピンには、ビットストリーム設定のデフォルトを使用することをお勧めしています。

Zynq-7000 のビットストリーム設定

次の表に、Vivado ツールの `set_property <Setting> <Value> [current_design] Tcl` コマンドで使用可能な Zynq[®]-7000 デバイスのデバイス コンフィギュレーション設定を示します。

注記: 暗号化のビットストリーム設定は、Zynq-7000 デバイスでは使用できません。

表 37: Zynq-7000 のビットストリーム設定

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1、2、3、4	BPI コンフィギュレーションをフラッシュ デバイスのページ モード動作のタイミングと同期させる際に使用し、最初のページの有効読み出しのサイクル数を設定します。このオプションは、BPI_page_size を 4 または 8 に設定している場合にのみ有効です。
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1、4、8	BPI コンフィギュレーションのページ サイズを指定します。これは、フラッシュ メモリでページごとに必要な読み出し数に対応します。

表 37: Zynq-7000 のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable、Type1、Type2	BPI フラッシュ デバイスの異なるタイプの BPI 同期コンフィギュレーション モードを設定します。 Disable (デフォルト): 同期コンフィギュレーション モードをディスエーブルにします。 Type1: 同期コンフィギュレーション モードをイネーブルにし、Micron G18(F) ファミリをサポートする設定を使用します。 Type2: 同期コンフィギュレーション モードをイネーブルにし、Micron (Numonyx) P30 および P33 ファミリをサポートする設定を使用します。
BITSTREAM.CONFIG.CCLKPIN ¹	Pullup	Pullup、Pullnone	Cclk ピンに内部プルアップを追加します。Pullnone に設定すると、プルアップは使用されません。
BITSTREAM.CONFIG.CONFIGFALLBACK	Enable	Disable、Enable	コンフィギュレーションでエラーが発生した場合にデフォルトのビットストリームを読み込むかどうかを指定します。
BITSTREAM.CONFIG.CONFIGRATE	3	3、6、9、12、16、22、26、33、40、50、66	マスター モードでコンフィギュレーションする場合、ビットストリームの生成でコンフィギュレーション クロック (Cclk) の生成に内部オシレーターが使用されます。このオプションを使用すると、Cclk のレートを選択できます。
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired、Continuous、Quiet	デジタル制御インピーダンス (DCI) 回路で DCI IOSTANDARD のインピーダンス一致をアップデートする頻度を指定します。
BITSTREAM.CONFIG.DONEPIN ¹	Pullup	Pullup、Pullnone	DONE ピンに内部プルアップを追加します。Pullnone に設定すると、プルアップは使用されません。このオプションは、外部プルアップ抵抗を DONE ピンに接続する場合にのみ使用してください。このオプションを使用しない場合、内部プルアップ抵抗が自動的に接続されます。
BITSTREAM.CONFIG.INITPIN ¹	Pullup	Pullup、Pullnone	INIT ピンにプルアップ抵抗を追加するか、未接続のままにするかを指定します。
BITSTREAM.CONFIG.INITSIGNALSERROR	Enable	Enable、Disable	Enable の場合、コンフィギュレーション エラーが検出されると INIT_B ピンが 0 にアサートされます。
BITSTREAM.CONFIG.M0PIN ¹	Pullup	Pullup、Pulldown、Pullnone	M0 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M0 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.M1PIN ¹	Pullup	Pullup、Pulldown、Pullnone	M1 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M1 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.M2PIN ¹	Pullup	Pullup、Pulldown、Pullnone	M2 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M2 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	なし	文字列	マルチブート セットアップの次のコンフィギュレーションの開始アドレスを設定します。これは、WBSTAR レジスタに保存されます。
BITSTREAM.CONFIG.NEXT_CONFIG_REBOOT	Enable	Enable、Disable	Disable に設定すると、.bit ファイルから IPROG コマンドが削除されます。マルチブート セットアップでのマルチブート イメージにジャンプするのではなく、パワーアップ時にゴールデンイメージが読み込まれるようになります。
BITSTREAM.CONFIG.OVERTEMPPOWERDOWN	Disable	Disable、Enable	XADC で温度が最大動作範囲を超えたことが検出された場合にデバイスがシャットダウンされるようにします。このオプションを使用するには、XADC に外部回路セットアップが必要です。

表 37: Zynq-7000 のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.PERSIST	No	No, Yes	コンフィギュレーション後に多目的コンフィギュレーションピンへのコンフィギュレーション ロジック アクセスを保持します。主にコンフィギュレーション後のリードバック アクセス用に SelectMAP ポートを保持するために使用されますが、どのコンフィギュレーション モードでも使用できます。JTAG コンフィギュレーションでは、JTAG ポートは専用ポートなので、PERSIST は必要ありません。PERSIST と ICAP を同時に使用することはできません。 詳細は、ユーザー ガイドを参照してください。PERSIST は、SelectMAP コンフィギュレーション ピンを使用するリードバックおよび Dynamic Function eXchange に必要で、SelectMAP またはシリアル モードを使用している場合に使用する必要があります。
BITSTREAM.CONFIG.REVISIONS ELECT	00	00、01、10、11	次のウォーム ブートのウォーム ブート開始アドレス (WBSTAR) レジスタの RS[1:0] 設定の内部値を指定します。
BITSTREAM.CONFIG.REVISIONS ELECT_ TRISTATE	Disable	Disable、Enable	ウォーム ブートのウォーム ブート開始アドレス (WBSTAR) のオプションを設定することにより、RS[1:0] トライステート イネーブルにするかどうかを指定します。 RS[1:0] ピンはトライステート イネーブル 0: RS トライステートをイネーブル 1: RS トライステートをディスエーブル
BITSTREAM.CONFIG.SELECTMAP ABORT	Enable	Enable、Disable	SelectMAP モードのアボート シーケンスをイネーブルまたはディスエーブルにします。Disable に設定すると、デバイス ピンのアボート シーケンスは無視されます。
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No, Yes	SPI 32 ビット アドレス形式をイネーブルにします。この形式は、256 Mb 以上のストレージを含む SPI デバイスで必要です。
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE、1、2、4	サードパーティ SPI フラッシュ デバイスからのマスター SPI コンフィギュレーションに対して、SPI バスをデュアル (x2) またはクワッド (x4) モードに設定します。
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No, Yes	FPGA で SPI データのキャプチャに立ち下がりエッジを使用するように設定します。これによりタイミング マージンが向上し、コンフィギュレーションのクロック レートが上がる可能性があります。
BITSTREAM.CONFIG.TCKPIN ¹	Pullup	Pullup、Pulldown、Pullnone	TCK ピン、JTAG テスト クロックにプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TDIPIN ¹	Pullup	Pullup、Pulldown、Pullnone	TDI ピン、JTAG 命令および JTAG レジスタへのシリアル データ入力すべてに、プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TDOPIN ¹	Pullup	Pullup、Pulldown、Pullnone	TDO ピン、JTAG 命令およびデータ レジスタへのシリアル データ出力すべてに、プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TIMER_CFG	なし	8 桁の 16 進文字列	コンフィギュレーション モードでのウォッチドッグ タイマーをイネーブルにして値を設定します。このオプションは、TIMER_USR と同時に使用することはできません。
BITSTREAM.CONFIG.TIMER_USR	0x00000000	8 桁の 16 進文字列	コンフィギュレーション モードでのウォッチドッグ タイマーをイネーブルにして値を設定します。このオプションは、TIMER_CFG と同時に使用することはできません。

表 37: Zynq-7000 のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.TMSPIN ¹	Pullup	Pullup、 Pulldown、 Pullnone	TMS ピン、TAP コントローラーへのモード入力信号にプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。TAP コントローラーは、JTAG の制御ロジックとして使用されます。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pulldown、 Pullup、 Pullnone	未使用の SelectIO ピン (IOB) にプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。コンフィギュレーション専用ピンには適用されません。コンフィギュレーション専用ピンのリストは、アーキテクチャによって異なります。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	8 桁の 16 進文字列	インプリメンテーションのリビジョンを特定します。ユーザー ID レジスタには、8 桁までの 16 進文字列を入力できます。
BITSTREAM.CONFIG.USR_ACCESS	なし	8 桁の 16 進文字列、 TIMESTAMP	AXSS コンフィギュレーション レジスタに、8 桁の 16 進文字列またはタイムスタンプを記述します。タイムスタンプ値のフォーマットは、dddd MM yy hh mm ss (dddd = 日、MM = 月、yy = 年 (2000 年は 0000)、hh = 時、mm = 分、ss = 秒) です。このレジスタの内容は、FPGA デバイスにより USR_ACCESS プリミティブを介して直接アクセスできます。
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT	bbram	bbram、efuse	使用する AES 暗号キーの場所を、バッテリー バックアップ式 RAM (BBRAM) または eFUSE レジスタのいずれかに指定します。(7 シリーズ)。 このプロパティは ENCRYPT オプションを Yes に設定している場合のみ使用可能です。
BITSTREAM.GENERAL.COMPRESS	False	True、False	ビットストリームの複数フレーム書き込み機能を使用し、ビットストリーム ファイル (.bit) ファイルだけでなく、ビットストリーム自体のサイズも縮小します。このオプションを使用しても、ビットストリームのサイズが縮小するとは限りません。
BITSTREAM.GENERAL.CRC	Enable	Enable、 Disable	ビットストリームの巡回冗長検査 (CRC) 値の生成を制御します。Enable に設定すると、ビットストリームの内容に基づいて固有の CRC 値が算出されます。算出された CRC 値がビットストリームの CRC 値と一致しない場合は、デバイスはコンフィギュレーションされません。CRC がディスエーブルの場合、CRC 値の代わりに定数値がビットストリームに挿入され、デバイスで CRC 値は算出されません。 CRC デフォルト値は Enable ですが、BITSTREAM.ENCRYPTION.ENCRYPT の場合のみ Yes、CRC はディスエーブルです。
BITSTREAM.GENERAL.DISABLE_JTAG	No	No、Yes	コンフィギュレーション後に JTAG を介したバウンダリスキャン (BSCAN) ブロックへのアクセスをディスエーブルにします。
BITSTREAM.GENERAL.JTAG_XADC	Enable	Enable、 Disable、 StatusOnly	XADC への JTAG 接続をイネーブルまたはディスエーブルにします。
BITSTREAM.GENERAL.XADCENHANCEDLINEARITY	Off	Off、On	INL が実際のアナログ パフォーマンスよりも悪くなるビルトイン デジタル キャリブレーション機能をディスエーブルにします。
BITSTREAM.GENERAL.PERFRAMECRC	No	No、Yes	ビットストリームに一定間隔で CRC 値を挿入します。これらの値は入力されるビットストリームのインテグリティを検証して、コンフィギュレーション データがデバイスにロードされる前にエラー (INIT_B ピンおよび ICAP の PRERRR ポートに表示) を通知します。これはパーシャル ビットストリームに適していますが、Yes に設定すると、CRC 値がデバイス ストリーム全体を含め、すべてのビットストリームに挿入されます。

表 37: Zynq-7000 のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.READBACK.ACTIVERECONFIG	No	No, Yes	コンフィギュレーション中に GHIGH および GSR がアサートされないようにします。これは、アクティブ Dynamic Function eXchange 拡張機能に必要です。
BITSTREAM.READBACK.ICAP_SELECT	Auto	Auto, Top, Bottom	上または下の ICAP ポートを選択します。
BITSTREAM.READBACK.READBACK	False	True, False	必要なリードバック ファイルを作成してリードバック機能を実行します。
BITSTREAM.READBACK.SECURITY	None	None, Level1, Level2	リードバックおよびリコンフィギュレーションをディスエーブルにするかどうかを指定します。 Level1 に設定するとリードバックがディスエーブルになり、Level2 に設定するとリードバックとリコンフィギュレーションがディスエーブルになります。
BITSTREAM.READBACK.XADCPARTIALRECONFIG	Disable	Disable, Enable	Disable に設定すると、Dynamic Function eXchange 中も XADC が継続して機能します。Enable に設定すると、Dynamic Function eXchange 中は XADC はセーフ モードで機能します。
BITSTREAM.STARTUP.DONEPIPE	Yes	Yes, No	CFG_DONE (DONE) ピンが High になり、最初のクロック エッジを待って、Done ステートに移動します。
BITSTREAM.STARTUP.DONE_CYCLE	4	4, 1, 2, 3, 5, 6, Keep	FPGA Done 信号をアクティブにするスタートアップ フェーズを選択します。DonePipe=Yes の場合、Done は遅延されます。
BITSTREAM.STARTUP.GTS_CYCLE	5	5, 1, 2, 3, 4, 6, Done, Keep	I/O バッファへの内部トライステート制御を解放するスタートアップ フェーズを選択します。
BITSTREAM.STARTUP.GWE_CYCLE	6	6, 1, 2, 3, 4, 5, Done, Keep	フリップフロップ、LUT RAM、およびシフトレジスタへの内部イネーブルをアサートするスタートアップ フェーズを選択します。BRAM もイネーブルになります。このスタートアップ フェーズの前は、ブロック RAM の書き込みおよび読み出しの両方がディスエーブルです。
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait, 0, 1, 2, 3, 4, 5, 6	DLL/DCM/PLL ロックまで待機するため、スタートアップ段階を選択します。NoWait に設定すると、スタートアップシーケンスは DLL/DCM/PLL がロックされるまで待機されません。
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto, NoWait, 0, 1, 2, 3, 4, 5, 6	デジタル制御インピーダンス (DCI) 一致信号がアサートされるまで待機するスタートアップサイクルを指定します。DCI マッチは MATCH_CYCLE では開始しません。スタートアップシーケンスは DCI が一致するまでこのサイクルで待機します。DCI が一致するのにかかる時間にはさまざまな要素が影響するので、スタートアップシーケンスが完了するのに必要な CCLK サイクル数は、同じシステムでも異なる場合があります。DONE が High になるまでコンフィギュレーションソリューションで CCLK を駆動するのが理想的です。 Auto に設定すると、write_bitstream によりデザインで DCI I/O 規格が検索されます。DCI 規格が存在する場合、write_bitstream が BITSTREAM.STARTUP.MATCH_CYCLE=2 を使用します。存在しない場合、write_bitstream は BITSTREAM.STARTUP.MATCH_CYCLE=NoWait を使用します。

表 37: Zynq-7000 のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.STARTUP.STARTUP_CLK	Cclk	Cclk、UserClk、JtagClk	デバイスのコンフィギュレーション後の StartupClk シーケンスは、Cclk、ユーザー クロック、または JTAG クロックに同期させることができます。デフォルトは Cclk です。 Cclk: FPGA デバイスで供給される内部クロックに同期します。 UserClk: STARTUP シンボルの CLK ピンに接続されているユーザー定義信号に同期します。 JtagClk: JTAG で供給されるクロックに同期します。このクロックにより、JTAG に制御ロジックを提供する TAP コントローラーの順序が決まります。

注記:

1. ザイリンクスでは、専用コンフィギュレーション ピンには、ビットストリーム設定のデフォルトを使用することをお勧めしています。

UltraScale のビットストリーム設定

次の表に、Vivado ツールの `set_property <Setting> <Value> [current_design] Tcl` コマンドで使用可能な UltraScale™ デバイスのデバイス コンフィギュレーション設定を示します。

表 38: UltraScale のビットストリーム設定

設定	デフォルト値	設定可能な値	説明
BITSTREAM.AUTHENTICATION.AUTHENTICATE	No	Yes、No	RSA 認証を使用するかどうかを指定します。No の場合、AES_GCM が使用されます。
BITSTREAM.AUTHENTICATION.RSAPRIVATEKEYFILE	なし	文字列	RSA-2048 認証ビットストリームをサインするために使用するべきキー ペアを含む OpenSSL .pem ファイルを指定します。
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1、2、3、4	BPI コンフィギュレーションをフラッシュ デバイスのページモード動作のタイミングと同期させる際に使用し、最初のページの有効読み出しのサイクル数を設定します。このオプションは、BPI_page_size を 4 または 8 に設定している場合にのみ有効です。
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1、4、8	BPI コンフィギュレーションのページ サイズを指定します。これは、フラッシュ メモリでページごとに必要な読み出し数に対応します。
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable、Type1、Type2	BPI フラッシュ デバイスの異なるタイプの BPI 同期コンフィギュレーション モードを設定します。 Disable (デフォルト): 同期コンフィギュレーション モードをディスエーブルにします。 Type1: 同期コンフィギュレーション モードをイネーブルにし、Micron G18(F) ファミリをサポートする設定を使用します。 Type2: 同期コンフィギュレーション モードをイネーブルにし、Micron (Numonyx) P30 および P33 ファミリをサポートする設定を使用します。
BITSTREAM.CONFIG.CCLKPIN ¹	Pullup	Pullup、Pullnone	Cclk ピンに内部プルアップを追加します。Pullnone に設定すると、プルアップは使用されません。
BITSTREAM.CONFIG.CONFIGFALLBACK	Enable	Disable、Enable	コンフィギュレーションでエラーが発生した場合にデフォルトのビットストリームを読み込むかどうかを指定します。

表 38: UltraScale のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.CONFIGRATE	3	3、6、9、12、22、33、40、50、57、69、82、87、90、110、115、130、148	マスター モードでコンフィギュレーションする場合、ビットストリームの生成でコンフィギュレーション クロック (Cclk) の生成に内部オシレーターが使用されます。このオプションを使用すると、Cclk のレートを選択できます。
BITSTREAM.CONFIG.D00_MOSI ¹	Pullup	Pullup、Pulldown、Pullnone	D00_MOSI ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。D00_MOSI ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.D01_DIN ¹	Pullup	Pullup、Pulldown、Pullnone	D01_DIN ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。D01_DIN ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.D02 ¹	Pullup	Pullup、Pulldown、Pullnone	D02 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。D02 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.D03 ¹	Pullup	Pullup、Pulldown、Pullnone	D03 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。D03 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired、Continuous、Quiet	デジタル制御インピーダンス (DCI) 回路で DCI IOSTANDARD のインピーダンス一致をアップデートする頻度を指定します。
BITSTREAM.CONFIG.DONEPIN ¹	Pullup	Pullup、Pullnone	DONE ピンに内部プルアップを追加します。Pullnone に設定すると、プルアップは使用されません。このオプションは、外部プルアップ抵抗を DONE ピンに接続する場合にのみ使用してください。このオプションを使用しない場合、内部プルアップ抵抗が自動的に接続されます。
BITSTREAM.CONFIG.EXTMASTERCCLK_EN	Disable	Disable、Div-1、Div-2、Div-3、Div-4、Div-6、Div-8、Div-12、Div-16、Div-24、Div-48	すべてのマスター モードで外部クロックをコンフィギュレーション クロックとして使用できるようにします。外部クロックは、多目的 EMCCLK ピンに接続する必要があります。
BITSTREAM.CONFIG.INITPIN ¹	Pullup	Pullup、Pullnone	INIT ピンにプルアップ抵抗を追加するか、未接続のままにするかを指定します。
BITSTREAM.CONFIG.INITSIGNALSERROR	Enable	Enable、Disable	Enable の場合、コンフィギュレーション エラーが検出されると INIT_B ピンが 0 にアサートされます。
BITSTREAM.CONFIG.M0PIN ¹	Pullup	Pullup、Pulldown、Pullnone	M0 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M0 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.M1PIN ¹	Pullup	Pullup、Pulldown、Pullnone	M1 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M1 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.M2PIN ¹	Pullup	Pullup、Pulldown、Pullnone	M2 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M2 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	none	文字列	マルチブート セットアップの次のコンフィギュレーションの開始アドレスを設定します。これは、WBSTAR レジスタに保存されます。

表 38: UltraScale のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.NEXT_CONFIG_REBOOT	Enable	Enable、Disable	Disable に設定すると、.bit ファイルから IPROG コマンドが削除されます。マルチブート セットアップでのマルチブート イメージにジャンプするのではなく、パワーアップ時にゴールデン イメージが読み込まれるようになります。
BITSTREAM.CONFIG.OVERTEMP_SHUTDOWN	Disable	Disable、Enable	システム モニターで温度が最大動作範囲を超えたことが検出された場合にデバイスがシャットダウンされるようにします。このオプションを使用するには、システム モニターに外部回路セットアップが必要です。
BITSTREAM.CONFIG.PERSIST	No	No、Yes	コンフィギュレーション後に多目的コンフィギュレーションピンへのコンフィギュレーション ロジック アクセスを保持します。主にコンフィギュレーション後のリードバック アクセス用に SelectMAP ポートを保持するために使用されますが、どのコンフィギュレーション モードでも使用できます。JTAG コンフィギュレーションでは、JTAG ポートは専用ポートなので、PERSIST は必要ありません。PERSIST と ICAP を同時に使用することはできません。 詳細は、ユーザー ガイドを参照してください。PERSIST は、SelectMAP コンフィギュレーション ピンを使用するリードバックおよび Dynamic Function eXchange に必要で、SelectMAP またはシリアル モードを使用している場合に使用する必要があります。
BITSTREAM.CONFIG.PROGPIN ¹	Pullup	Pullup、Pullnone	PROGRAM_B ピンに内部プルアップを追加します。Pullnone に設定すると、プルアップは使用されません。プルアップは、コンフィギュレーション後のピンに使用されます。
BITSTREAM.CONFIG.PUDC_B ¹	Pullup	Pullup、Pulldown、Pullnone	PUDC_B ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。PUDC_B ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.RDWR_B_FCS_B ¹	Pullup	Pullup、Pulldown、Pullnone	RDWR_B_FCS_B ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。 RDWR_B_FCS_B ピン。RDWR_B_FCS_B ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.REVISIONS_ELECT	00	00、01、10、11	次のウォーム ブートのウォーム ブート開始アドレス (WBSTAR) レジスタの RS[1:0] 設定の内部値を指定します。
BITSTREAM.CONFIG.REVISIONS_ELECT_TRISTATE	Disable	Disable、Enable	ウォーム ブートのウォーム ブート開始アドレス (WBSTAR) のオプションを設定することにより、RS[1:0] トライステートをイネーブルにするかどうかを指定します。 RS[1:0] ピンはトライステート イネーブル 0: RS トライステートをイネーブル 1: RS トライステートをディスエーブル
BITSTREAM.CONFIG.SELECTMAP_ABORT	Enable	Enable、Disable	SelectMAP モードのアボート シーケンスをイネーブルまたはディスエーブルにします。Disable に設定すると、デバイスピンのアボート シーケンスは無視されます。
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No、Yes	SPI 32 ビット アドレス形式をイネーブルにします。この形式は、256 Mb 以上のストレージを含む SPI デバイスで必要です。
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE、1、2、4	サードパーティ SPI フラッシュ デバイスからのマスター SPI コンフィギュレーションに対して、SPI バスをデュアル (x2) またはクワッド (x4) モードに設定します。
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No、Yes	FPGA で SPI データのキャプチャに立ち上がりエッジを使用するよう設定します。これによりタイミング マージンが向上し、コンフィギュレーションのクロック レートが上がる可能性があります。

表 38: UltraScale のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.TCKPIN ¹	Pullup	Pullup、 Pulldown、 Pullnone	TCK ピン、JTAG テスト クロックにプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TDIPIN ¹	Pullup	Pullup、 Pulldown、 Pullnone	TDI ピン、JTAG 命令および JTAG レジスタへのシリアル データ入力すべてに、プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TDOPIN ¹	Pullup	Pullup、 Pulldown、 Pullnone	TDO ピン、JTAG 命令およびデータ レジスタへのシリアル データ出力すべてに、プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TIMER_CFG	なし	8 桁の 16 進文字列	コンフィギュレーション モードでのウォッチドッグ タイマーをイネーブルにして値を設定します。このオプションは、TIMER_USR と同時に使用することはできません。
BITSTREAM.CONFIG.TIMER_USR	0x00000000	8 桁の 16 進文字列	コンフィギュレーション モードでのウォッチドッグ タイマーをイネーブルにして値を設定します。このオプションは、TIMER_CFG と同時に使用することはできません。
BITSTREAM.CONFIG.TMSPIN ¹	Pullup	Pullup、 Pulldown、 Pullnone	TMS ピン、TAP コントローラーへのモード入力信号にプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。TAP コントローラーは、JTAG の制御ロジックとして使用されます。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pulldown、 Pullup、 Pullnone	未使用の SelectIO ピン (IOB) にプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。コンフィギュレーション専用ピンには適用されません。コンフィギュレーション専用ピンのリストは、アーキテクチャによって異なります。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	8 桁の 16 進文字列	インプリメンテーションのリビジョンを特定します。ユーザー ID レジスタには、8 桁までの 16 進文字列を入力できます。
BITSTREAM.CONFIG.USR_ACCESS	なし	8 桁の 16 進文字列、 TIMESTAMP	AXSS コンフィギュレーション レジスタに、8 桁の 16 進文字列またはタイムスタンプを記述します。タイムスタンプ値のフォーマットは、dddd Mmmm yyyy hhhh mmmmm ssssss (dddd = 日、Mmmm = 月、yyyy = 年 (2000 年は 0000)、hhhhh = 時、mmmmm = 分、sssss = 秒) です。このレジスタの内容は、FPGA デバイスにより USR_ACCESS プリミティブを介して直接アクセスできます。
BITSTREAM.ENCRYPTION.ENCRYPT	No	No、Yes	ビットストリームを暗号化します。
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT	bbam	bbam、efuse	使用する AES 暗号キーの場所を、バッテリー バックアップ式 RAM (BBAM) または eFUSE レジスタのいずれかに指定します。 このプロパティは ENCRYPT オプションを Yes に設定している場合のみ使用可能です。
BITSTREAM.ENCRYPTION.FAMILY_KEY_FILEPATH	なし	familyKey.cfg へのパス	ファミリー キーのインストール ディレクトリを指定します。ディレクトリを必ず指定する必要があるわけではありません。 ファミリー キーはザイリンクス ツールスイートには含まれません。ファミリー キーが必要な場合は、 secure.solutions@xilinx.com までリクエストしてください。権限のあるカスタマーのみ、 https://japan.xilinx.com の製品ライセンス ページから入手できるようになります。
BITSTREAM.ENCRYPTION.KEY0	なし	16 進文字列	ビットストリーム暗号化の AES 暗号キーを設定します。このオプションを使用するには、ENCRYPT オプションを Yes に設定する必要があります。

表 38: UltraScale のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.ENCRYPTION.KEYFILE	なし	文字列	入力暗号化ファイル (拡張子 .nky) の名前を指定します。このオプションを使用するには、ENCRYPT オプションを Yes に設定する必要があります。
BITSTREAM.ENCRYPTION.KEYLIFE	32	4 ~ 2147483647	AES-GCM 認証ビットストリームに使用するべき単一のキーに対する 128 ビット暗号化ブロックの数。 0 に設定すると、これらのオプションがディスエーブルになります。
BITSTREAM.ENCRYPTION.RSAKEYLIFEFrames	8	8 ~ 2147483647	RSA 公開キー認証を指定する場合に、該当する AES-256 キーに使用するべきコンフィギュレーション フレーム数を指定します。コンフィギュレーション フレーム値に 8 を指定するのは、246 の暗号化ブロックのキーを使用するのと同じことです。 0 に設定すると、これらのオプションがディスエーブルになります。
BITSTREAM.ENCRYPTION.OBFUSCATEKEY	Disable	Enable、Disable	ビットストリームを暗号化するために使用されたキーが eFUSE またはバッテリー バックアップ式 RAM (BBR) に保存される前に難読化されるビットストリームを作成します。これにより、デバイス プログラムで元のカスタマー キーではなく難読化されたキーを使用できます。難読化されたキーは eFUSE または BBR に保存され、選択されたストレージ ロケーションの難読化キー フラグを使用して難読化されたキーとしてマークされます。
BITSTREAM.ENCRYPTION.STARTIVO			最初の AES-GCM メッセージで初期 GCM カウント値を指定するのに使用される初期化ベクターで、32 ビットの 16 進数値です。
BITSTREAM.ENCRYPTION.STARTIVOFUSCATE			難読初期化ベクター値 (Obfuscate IV0) を開始します。
BITSTREAM.GENERAL.COMPRESS	False	True、False	ビットストリームの複数フレーム書き込み機能を使用し、ビットストリーム ファイル (.bit) ファイルだけでなく、ビットストリーム自体のサイズも縮小します。このオプションを使用しても、ビットストリームのサイズが縮小するとは限りません。
BITSTREAM.GENERAL.CRC	Enable	Enable、Disable	ビットストリームの巡回冗長検査 (CRC) 値の生成を制御します。Enable に設定すると、ビットストリームの内容に基づいて固有の CRC 値が算出されます。算出された CRC 値がビットストリームの CRC 値と一致しない場合は、デバイスはコンフィギュレーションされません。CRC がディスエーブルの場合、CRC 値の代わりに定数値がビットストリームに挿入され、デバイスで CRC 値は算出されません。 CRC デフォルト値は Enable ですが、BITSTREAM.ENCRYPTION.ENCRYPT の場合のみ Yes、CRC はディスエーブルです。
BITSTREAM.GENERAL.DEBUGBITSTREAM	No	No、Yes	デバッグ ビットストリームを生成します。デバッグ ビットストリームのサイズは、標準のビットストリームよりかなり大きくなります。このオプションは、マスターおよびスレーブ シリアル コンフィギュレーションでのみ使用できます。バウンダリスキャンおよびスレーブ パラレル/SelectMAP では使用できません。デバッグ ビットストリームには、標準ビットストリームに加え、次の機能があります。 同期化ワードの後に LOUT レジスタに 32 個の 0 を書き込みます。 各フレームを個別に読み込みます。 各フレーム後に巡回冗長検査 (CRC) を実行します。 各フレーム後にフレーム アドレスを LOUT レジスタに書き込みます。

表 38: UltraScale のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.GENERAL.DISABLE_JTAG	No	No, Yes	コンフィギュレーション後に JTAG を介したバウンダリスキャン (BSCAN) ブロックへのアクセスをディスエーブルにします。
BITSTREAM.GENERAL.PERFRAMECRC	No	No, Yes	ビットストリームに一定間隔で CRC 値を挿入します。これらの値は入力されるビットストリームのインテグリティを検証して、コンフィギュレーション データがデバイスにロードされる前にエラー (INIT_B ピンおよび ICAP の PRERROR ポートに表示) を通知します。これはパースシャル ビットストリームに適していますが、Yes に設定すると、CRC 値がデバイスストリーム全体を含め、すべてのビットストリームに挿入されます。
BITSTREAM.GENERAL.JTAG_SYSMON	Enable	Enable、Disable、StatusOnly	SYSMON への JTAG 接続をイネーブルまたはディスエーブルにします。
BITSTREAM.GENERAL.SYSMON_POWERDOWN	Disable	Disable、Enable	SYSMON をパワーダウンできるようにして節電します。SYSMON を永久にパワーダウンする場合にのみ推奨されます。
BITSTREAM.MMCM.BANDWIDTH	DEFAULT	POSTCRC	MMCM の BANDWIDTH 設定をすべて OPTIMIZED から POSTCRC に変更します。
BITSTREAM.PLL.BANDWIDTH	DEFAULT	POSTCRC	PLL の BANDWIDTH 設定をすべて OPTIMIZED から POSTCRC に変更します。
BITSTREAM.READBACK.ACTIVE_RECONFIG	No	No, Yes	コンフィギュレーション中に GHIGH および GSR がアサートされないようにします。これは、アクティブ Dynamic Function eXchange 拡張機能に必要です。
BITSTREAM.READBACK.ICAP_SELECT	Auto	Auto、Top、Bottom	上または下の ICAP ポートを選択します。
BITSTREAM.READBACK.READBACK	False	True、False	必要なリードバック ファイルを作成してリードバック機能を実行します。
BITSTREAM.READBACK.SECURITY	None	None、Level1、Level2	リードバックおよびリコンフィギュレーションをディスエーブルにするかどうかを指定します。 Level1 に設定するとリードバックがディスエーブルになり、Level2 に設定するとリードバックとリコンフィギュレーションがディスエーブルになります。
BITSTREAM.STARTUP.DONE_CYCLE	4	4、1、2、3、5、6、Keep	FPGA Done 信号をアクティブにするスタートアップ フェーズを選択します。DonePipe=Yes の場合、Done は遅延されます。
BITSTREAM.STARTUP.GTS_CYCLE	5	5、1、2、3、4、6、Done、Keep	I/O バッファへの内部トライステート制御を解放するスタートアップ フェーズを選択します。
BITSTREAM.STARTUP.GWE_CYCLE	6	6、1、2、3、4、5、Done、Keep	フリップフロップ、LUT RAM、およびシフトレジスタへの内部イネーブルをアサートするスタートアップ フェーズを選択します。BRAM もイネーブルにします。このスタートアップフェーズの前は、ブロック RAM の書き込みおよび読み出しの両方がディスエーブルです。
BITSTREAM.STARTUP.LCK_CYCLE	NoWait	NoWait、0、1、2、3、4、5、6	DLL/DCM/PLL がロックされるまで待機するスタートアップフェーズを選択します。NoWait に設定すると、スタートアップシーケンスは DLL/DCM/PLL がロックされるまで待機されません。

表 38: UltraScale のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.STARTUP.MATCH_CYCLE	Auto	Auto、NoWait、0、1、2、3、4、5、6	<p>デジタル制御インピーダンス (DCI) 一致信号がアサートされるまで待機するスタートアップ サイクルを指定します。DCI マッチは MATCH_CYCLE では開始しません。スタートアップシーケンスは DCI が一致するまでこのサイクルで待機します。DCI が一致するのにかかる時間にはさまざまな要素が影響するので、スタートアップシーケンスが完了するのに必要な CCLK サイクル数は、同じシステムでも異なる場合があります。DONE が High になるまでコンフィギュレーション ソリューションで CCLK を駆動するのが理想的です。</p> <p>Auto に設定すると、write_bitstream によりデザインで DCI I/O 規格が検索されます。DCI 規格が存在する場合、write_bitstream が BITSTREAM.STARTUP.MATCH_CYCLE=2 を使用します。存在しない場合、write_bitstream は BITSTREAM.STARTUP.MATCH_CYCLE=NoWait を使用します。</p>

注記:

1. ザイリンクス専用コンフィギュレーション ピンの場合は、デフォルトのビットストリーム設定を使用することをお勧めします。

Virtex および Kintex UltraScale+ のビットストリーム設定

次の表に、Vivado ツールの `set_property <Setting> <Value> [current_design] Tcl` コマンドで使用可能な Virtex および Kintex® UltraScale+ デバイスのデバイス コンフィギュレーション設定を示します。

表 39: Virtex および Kintex UltraScale+ のビットストリーム設定

設定	デフォルト値	設定可能な値	説明
BITSTREAM.AUTHENTICATION.AUTHENTICATE	No	No、Yes	RSA 認証を使用するかどうかを指定します。No の場合、AES_GCM が使用されます。
BITSTREAM.AUTHENTICATION.RSAPRIVATEKEYFILE			RSA-2048 認証ビットストリームをサインするために使用するべきキー ペアを含む OpenSSL .pem ファイルを指定します。
BITSTREAM.CONFIG.BPI_1ST_READ_CYCLE	1	1、2、3、4	BPI コンフィギュレーションをフラッシュ デバイスのページ モード動作のタイミングと同期させる際に使用し、最初のページの有効読み出しのサイクル数を設定します。このオプションは、BPI_page_size を 4 または 8 に設定している場合にのみ有効です。
BITSTREAM.CONFIG.BPI_PAGE_SIZE	1	1、4、8	BPI コンフィギュレーションのページ サイズを指定します。これは、フラッシュ メモリでページごとに必要な読み出し数に対応します。

表 39: Virtex および Kintex UltraScale+ のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.BPI_SYNC_MODE	Disable	Disable、Type1、Type2	BPI フラッシュ デバイスの異なるタイプの BPI 同期コンフィギュレーション モードを設定します。 Disable (デフォルト): 同期コンフィギュレーション モードをディスエーブルにします。 Type1: 同期コンフィギュレーション モードをイネーブルにし、Micron G18(F) ファミリをサポートする設定を使用します。 Type2: 同期コンフィギュレーション モードをイネーブルにし、Micron (Numonyx) P30 および P33 ファミリをサポートする設定を使用します。
BITSTREAM.CONFIG.CCLKPIN	Pullup	Pullup、Pullnone	Cclk ピンに内部プルアップを追加します。Pullnone に設定すると、プルアップは使用されません。
BITSTREAM.CONFIG.PERSIST	No	No、Yes	コンフィギュレーション ピンをユーザー I/O として使用できないようにし、コンフィギュレーション後も保持します。
BITSTREAM.CONFIG.CONFIGRATE	2.7	2.7、5.3、8.0、10.6、21.3、31.9、36.4、51.0、56.7、63.8、72.9、85.0、102.0、127.5、170.0	コンフィギュレーションがマスター モードの場合、ビットストリームの生成でコンフィギュレーション クロック (Cclk) の生成に内部オシレーターが使用されます。このオプションは、Cclk のレートを選択します。
BITSTREAM.CONFIG.D00_MOSI	Pullup	Pullup、Pulldown、Pullnone	D00_MOSI ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。D00_MOSI ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.D01_DIN	Pullup	Pullup、Pulldown、Pullnone	D01_DIN ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。D01_DIN ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.D02	Pullup	Pullup、Pulldown、Pullnone	D02 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。D02 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.D03	Pullup	Pullup、Pulldown、Pullnone	D03 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。D03 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired、Quiet、Safe	デジタル制御インピーダンス (DCI) 回路で DCI IOSTANDARD のインピーダンス一致をアップデートする頻度を指定します。
BITSTREAM.CONFIG.DONEPIN	Pullup	Pullup、Pullnone	DONE ピンに内部プルアップを追加します。Pullnone に設定すると、プルアップは使用されません。このオプションは、外部プルアップ抵抗を DONE ピンに接続する場合にのみ使用してください。このオプションを使用しない場合、内部プルアップ抵抗が自動的に接続されます。
BITSTREAM.CONFIG.EXTMASTERCCLK_EN	Disable	Disable、Div-1、Div-2、Div-3、Div-4、Div-6、Div-8、Div-12、Div-16、Div-24、Div-48	すべてのマスター モードで外部クロックをコンフィギュレーション クロックとして使用できるようにします。外部クロックは、多目的 EMCCLK ピンに接続する必要があります。

表 39: Virtex および Kintex UltraScale+ のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.ENCRYPTION.FAMILY_KEY_FILEPATH	なし	familyKey.cfg へのパス	ファミリー キーのインストール ディレクトリを指定します。ディレクトリを必ず指定する必要があるわけではありません。 ファミリー キーはザイリンクス ツール スイートには含まれません。ファミリー キーが必要な場合は、 secure.solutions@xilinx.com までリクエストしてください。権限のあるカスタマーのみ、 https://japan.xilinx.com の製品ライセンス ページから入手できるようになります。
BITSTREAM.CONFIG.INITPIN	Pullup	Pullup、Pullnone	INIT ピンにプルアップ抵抗を追加するか、未接続のままにするかを指定します。
BITSTREAM.CONFIG.M0PIN	Pullup	Pullup、Pulldown、Pullnone	M0 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M0 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.M1PIN	Pullup	Pullup、Pulldown、Pullnone	M1 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M1 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.M2PIN	Pullup	Pullup、Pulldown、Pullnone	M2 ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。M2 ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	なし	文字列	マルチブート セットアップの次のコンフィギュレーションの開始アドレスを設定します。これは、WBSTAR レジスタに保存されます。
BITSTREAM.CONFIG.NEXT_CONFIG_REBOOT	Enable	Enable、Disable	Disable に設定すると、.bit ファイルから IPROG コマンドが削除されます。
BITSTREAM.CONFIG.SELECTMAP_ABORT	Enable	Enable、Disable	SelectMAP モードのアボート シーケンスをイネーブルまたはディスエーブルにします。Disable に設定すると、デバイス ピンのアボート シーケンスは無視されます。
BITSTREAM.CONFIG.CONFIGFALLBACK	Enable	Enable、Disable	コンフィギュレーションでエラーが発生した場合にデフォルトのビットストリームを読み込むかどうかを指定します。
BITSTREAM.CONFIG.PROGPIN	Pullup	Pullup、Pullnone	PROGRAM_B ピンに内部プルアップを追加します。Pullnone に設定すると、プルアップは使用されません。プルアップは、コンフィギュレーション後のピンに使用されます。
BITSTREAM.CONFIG.PUDC_B	Pullup	Pullup、Pulldown、Pullnone	PUDC_B ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。PUDC_B ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.RDWR_B_FCS_B	Pullup	Pullup、Pulldown、Pullnone	RDWR_B_FCS_B ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。RDWR_B_FCS_B ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.REVISIONSELECT	00	00、01、10、11	次のウォーム ブートのウォーム ブート開始アドレス (WBSTAR) レジスタの RS[1:0] 設定の内部値を指定します。
BITSTREAM.CONFIG.REVISIONSELECT_TRISTATE	Disable	Disable、Enable	ウォーム ブートのウォーム ブート開始アドレス (WBSTAR) のオプションを設定することにより、RS[1:0] トライステートをイネーブルにするかどうかを指定します。 RS[1:0] ピンはトライステート イネーブル 0: RS トライステートをイネーブル 1: RS トライステートをディスエーブル

表 39: Virtex および Kintex UltraScale+ のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.CONFIG.OVERTEMP SHUTDOWN	Disable	Disable、Enable	システム モニターで温度が最大動作範囲を超えたことが検出された場合にデバイスがシャットダウンされるようにします。このオプションを使用するには、システム モニターに外部回路セットアップが必要です。
BITSTREAM.CONFIG.SPI_32BIT_ADDR	No	No、Yes	SPI 32 ビット アドレス形式をイネーブルにします。この形式は、256 Mb 以上のストレージを含む SPI デバイスで必要です。
BITSTREAM.CONFIG.SPI_BUSWIDTH	NONE	NONE、1、2、4	サードパーティ SPI フラッシュ デバイスからのマスター SPI コンフィギュレーションに対して、SPI バスをデュアル (x2) またはクワッド (x4) モードに設定します。
BITSTREAM.CONFIG.SPI_FALL_EDGE	No	No、Yes	FPGA で SPI データのキャプチャに立ち下がりエッジを使用するように設定します。これによりタイミング マージンが向上し、コンフィギュレーションのクロック レートが上がる可能性があります。
BITSTREAM.CONFIG.TCKPIN	Pullup	Pullup、Pulldown、Pullnone	TCK ピン、JTAG テスト クロックにプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TDIPIN	Pullup	Pullup、Pulldown、Pullnone	TDI ピン、JTAG 命令および JTAG レジスタへのシリアル データ入力すべてに、プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TDOPIN	Pullup	Pullup、Pulldown、Pullnone	TDO ピン、JTAG 命令およびデータ レジスタへのシリアル データ出力すべてに、プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.TIMER_CFG			コンフィギュレーション モードでのウォッチドッグ タイマーをイネーブルにして値を設定します。このオプションは、TIMER_USR と同時に使用することはできません。
BITSTREAM.CONFIG.TIMER_USR			コンフィギュレーション モードでのウォッチドッグ タイマーをイネーブルにして値を設定します。このオプションは、TIMER_CFG と同時に使用することはできません。
BITSTREAM.CONFIG.TMSPIN	Pullup	Pullup、Pulldown、Pullnone	TMS ピン、TAP コントローラーへのモード入力信号にプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。TAP コントローラーは、JTAG の制御ロジックとして使用されます。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pullup、Pulldown、Pullnone	未使用の SelectIO ピン (IOB) にプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。コンフィギュレーション専用ピンには適用されません。コンフィギュレーション専用ピンのリストは、アーキテクチャによって異なります。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	0xFFFFFFFF	インプリメンテーションのリビジョンを特定します。ユーザー ID レジスタには、8 桁までの 16 進文字列を入力できます。
BITSTREAM.CONFIG.USR_ACCESS	なし	None、8 桁の 16 進文字列、TIMESTAMP	AXSS コンフィギュレーション レジスタに、8 桁の 16 進文字列またはタイムスタンプを記述します。タイムスタンプ値のフォーマットは、dddd MM yy hh mm ss (dddd = 日、MM = 月、yy = 年 (2000 年は 0000)、hh = 時、mm = 分、ss = 秒) です。このレジスタの内容は、FPGA デバイスにより USR_ACCESS プリミティブを介して直接アクセスできます。
BITSTREAM.CONFIG.INITSIGNALSERROR	Enable	Enable、Disable	Enable の場合、コンフィギュレーション エラーが検出されると INIT_B ピンが 0 にアサートされます。

表 39: Virtex および Kintex UltraScale+ のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.ENCRYPTION.ENCRYPT	No	No、Yes	ビットストリームを暗号化します。
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT	bbram	bbram、efuse	使用する AES 暗号キーの場所を、バッテリー バックアップ式 RAM (BBRAM) または eFUSE レジスタのいずれかに指定します。このプロパティは ENCRYPT オプションを Yes に設定している場合のみ使用可能です。
BITSTREAM.ENCRYPTION.OBFUSCATEKEY	Disable	Disable、Enable	AES キーが読み出し保護されていないので、キーが読み出されると、実際のキー値ではなく、キーの CRC ハッシュが返されます。
BITSTREAM.ENCRYPTION.KEY0			ビットストリーム暗号化の 64 ビット AES 暗号キーを設定します。この値をブランクのままにすると、ランダムな値が選択されます。このオプションを使用するには、ENCRYPT オプションを Yes に設定する必要があります。
BITSTREAM.ENCRYPTION.STARTIVO			128 ビットの開始 AES 初期ベクター値を設定します。このオプションを使用するには、ENCRYPT オプションを Yes に設定する必要があります。
BITSTREAM.ENCRYPTION.STARTIVOBUSCATE			32 ビットの難読化初期ベクター値を設定します。このオプションを使用するには、ENCRYPT オプションを Yes に設定する必要があります。
BITSTREAM.ENCRYPTION.KEYFILE			入力暗号化ファイル (拡張子 .nky) の名前を指定します。このオプションを使用するには、ENCRYPT オプションを Yes に設定する必要があります。
BITSTREAM.ENCRYPTION.KEYLIFE	32	4 ~ 2147483647	AES-GCM 認証ビットストリームに使用するべき単一のキーに対する 128 ビット暗号化ブロックの数。
BITSTREAM.ENCRYPTION.RSAKEYLIFEFrames	8	8 ~ 2147483647	RSA 公開キー認証を指定する場合に、該当する AES-256 キーに使用するべきコンフィギュレーション フレーム数を指定します。コンフィギュレーション フレーム値に 8 を指定するのは、246 の暗号化ブロックのキーを使用するのと同じことです。
BITSTREAM.GENERAL.COMPRESS	False	True、False	ビットストリームの複数フレーム書き込み機能を使用し、BIT ファイルだけでなく、ビットストリーム自体のサイズも縮小します。このオプションを使用しても、ビットストリームのサイズが縮小するとは限りません。
BITSTREAM.GENERAL.CRC	Enable	Enable、Disable	ビットストリームの巡回冗長検査 (CRC) 値の生成を制御します。Enable に設定すると、ビットストリームの内容に基づいて固有の CRC 値が算出されます。算出された CRC 値がビットストリームの CRC 値と一致しない場合は、デバイスはコンフィギュレーションされません。CRC がディスエーブルの場合、CRC 値の代わりに定数値がビットストリームに挿入され、デバイスで CRC 値は算出されません。 CRC デフォルト値は Enable ですが、BITSTREAM.ENCRYPTION.ENCRYPT の場合のみ Yes、CRC はディスエーブルです。
BITSTREAM.GENERAL.DEBUGBITSTREAM	No	No、Yes	デバッグ ビットストリームを生成します。デバッグ ビットストリームのサイズは、標準のビットストリームよりかなり大きくなります。このオプションは、マスターおよびスレーブ シリアル コンフィギュレーションでのみ使用できます。バウンダリスキャンおよびスレーブ パラレル/SelectMAP では使用できません。デバッグ ビットストリームには、標準ビットストリームに加え、次の機能があります。同期化ワードの後に LOUT レジスタに 32 個の 0 を書き込みます。各フレームを個別に読み込みます。各フレーム後に巡回冗長検査 (CRC) を実行します。各フレーム後にフレーム アドレスを LOUT レジスタに書き込みます。

表 39: Virtex および Kintex UltraScale+ のビットストリーム設定 (続き)

設定	デフォルト値	設定可能な値	説明
BITSTREAM.GENERAL.PERFRAM ECRC	No	No, Yes	ビットストリームに一定間隔で CRC 値を挿入します。これらの値は入力されるビットストリームのインテグリティを検証して、コンフィギュレーション データがデバイスにロードされる前にエラー (INIT_B ピンおよび ICAP の PRERROR ポートに表示) を通知します。これはパーシャル ビットストリームに適していますが、Yes に設定すると、CRC 値がデバイス ストリーム全体を含め、すべてのビットストリームに挿入されます。
BITSTREAM.GENERAL.SYSMONP OWERDOWN	Disable	Disable, Enable	SYSMON をパワーダウンできるようにして節電します。SYSMON を永久にパワーダウンする場合にのみ推奨されます。
BITSTREAM.GENERAL.DISABLE_J TAG	No	No, Yes	コンフィギュレーション後に JTAG を介したバウンダリスキャン (BSCAN) ブロックへのアクセスをディスエーブルにします。
BITSTREAM.GENERAL.JTAG_SYS MON	Enable	Enable, Disable, StatusOnly	SYSMON への JTAG 接続をイネーブルまたはディスエーブルにします。
BITSTREAM.READBACK.ICAP_SE LECT	Auto	Auto, Top, Bottom	上または下の ICAP ポートを選択します。
BITSTREAM.READBACK.ACTIVER ECONFIG	No	No, Yes	コンフィギュレーション中に GHIGH および GSR がアサートされないようにします。これは、アクティブ Dynamic Function eXchange 拡張機能に必要です。
BITSTREAM.READBACK.SECURIT Y	None	None, Level1, Level2	リードバックおよびリコンフィギュレーションをディスエーブルにするかどうかを指定します。Level1 に設定するとリードバックがディスエーブルになり、
BITSTREAM.STARTUP.DONE_CY CLE	4	4, 1, 2, 3, 5, 6	FPGA Done 信号をアクティブにするスタートアップ フェーズを選択します。DonePipe=Yes の場合、Done は遅延されます。
BITSTREAM.STARTUP.GTS_CYCL E	5	5, 1, 2, 3, 4, 6, Done, Keep	I/O バッファへの内部トライステート制御を解放するスタートアップ フェーズを選択します。
BITSTREAM.STARTUP.GWE_CYC LE	6	6, 1, 2, 3, 4, 5, Done, Keep	フリップフロップ、LUT RAM、およびシフトレジスタへの内部イネーブルをアサートするスタートアップ フェーズを選択します。BRAM もイネーブルにします。このスタートアップ フェーズの前は、ブロック RAM の書き込みおよび読み出しの両方がディスエーブルです。
BITSTREAM.STARTUP.LCK_CYCL E	NoWait	NoWait, 0, 1, 2, 3, 4, 5, 6	MMCM/PLL がロックされるまで待機するスタートアップ フェーズを選択します。NoWait に設定すると、スタートアップシーケンスは MMCM/PLL がロックされるまで待機されません。
BITSTREAM.STARTUP.MATCH_C YCLE	Auto	Auto, NoWait, 0, 1, 2, 3, 4, 5, 6	デジタル制御インピーダンス (DCI) 一致信号がアサートされるまで待機するスタートアップ サイクルを指定します。DCI マッチは MATCH_CYCLE では開始しません。スタートアップシーケンスは DCI が一致するまでこのサイクルで待機します。DCI が一致するのにかかる時間にはさまざまな要素が影響するので、スタートアップシーケンスが完了するのに必要な CCLK サイクル数は、同じシステムでも異なる場合があります。DONE が High になるまでコンフィギュレーションソリューションで CCLK を駆動するのが理想的です。 Auto に設定すると、write_bitstream によりデザインで DCI I/O 規格が検索されます。DCI 規格が存在する場合、write_bitstream が BITSTREAM.STARTUP.MATCH_CYCLE=2 を使用します。存在しない場合、write_bitstream は BITSTREAM.STARTUP.MATCH_CYCLE=NoWait を使用します。

Zynq UltraScale+ MPSoC のビットストリーム設定

次の表に、Vivado ツールの `set_property <Setting> <Value> [current_design]` Tcl コマンドで使用可能な Zynq® UltraScale+ MPSoC デバイスのデバイス コンフィギュレーション設定を示します。

表 40: Zynq UltraScale+ MPSoC のビットストリーム設定

設定	デフォルト値	有効な値	説明
BITSTREAM.CONFIG.DCIUPDATEMODE	AsRequired	AsRequired、Quiet、Safe	デジタル制御インピーダンス (DCI) 回路で DCI IOSTANDARD のインピーダンス一致をアップデートする頻度を指定します。
BITSTREAM.CONFIG.PUDC_B	Pullup	Pullup、Pulldown、Pullnone	PUDC_B ピンに内部プルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。PUDC_B ピンにプルアップ抵抗およびプルダウン抵抗のどちらも追加しない場合は、Pullnone に設定します。
BITSTREAM.CONFIG.OVERTEMPSHUTDOWN	Disable	Disable、Enable	システム モニターで温度が最大動作範囲を超えたことが検出された場合にデバイスがシャットダウンされるようにします。このオプションを使用するには、システム モニターに外部回路セットアップが必要です。
BITSTREAM.CONFIG.UNUSEDPIN	Pulldown	Pullup、Pulldown、Pullnone	未使用の SelectIO ピン (IOB) にプルアップまたはプルダウンを追加するか、どちらも追加しないかを指定します。コンフィギュレーション専用ピンには適用されません。コンフィギュレーション専用ピンのリストは、アーキテクチャによって異なります。Pullnone に設定すると、プルアップもプルダウンも使用されません。
BITSTREAM.CONFIG.USERID	0xFFFFFFFF	0xFFFFFFFF	インプリメンテーションのリビジョンを特定します。ユーザー ID レジスタには、8 桁までの 16 進文字列を入力できます。
BITSTREAM.CONFIG.USR_ACCESS	None	None、8 桁の 16 進文字列、TIMESTAMP	AXSS コンフィギュレーション レジスタに、8 桁の 16 進文字列またはタイムスタンプを記述します。タイムスタンプ値のフォーマットは、dddd MM yy hh mm ss (dddd = 日、MM = 月、yy = 年 (2000 年は 0000)、hh = 時、mm = 分、ss = 秒) です。このレジスタの内容は、FPGA デバイスにより USR_ACCESS プリミティブを介して直接アクセスできます。
BITSTREAM.CONFIG.INIT_SIGNAL_ERROR	Enable	Enable、Disable	Enable の場合、コンフィギュレーション エラーが検出されると INIT_B ピンが 0 にアサートされます。
BITSTREAM.GENERAL.COMPRESS	False	True、False	ビットストリームの複数フレーム書き込み機能を使用し、BIT ファイルだけでなく、ビットストリーム自体のサイズも縮小します。このオプションを使用しても、ビットストリームのサイズが縮小するとは限りません。
BITSTREAM.GENERAL.CRC	Enable	Enable、Disable	ビットストリームの巡回冗長検査 (CRC) 値の生成を制御します。Enable に設定すると、ビットストリームの内容に基づいて固有の CRC 値が算出されます。算出された CRC 値がビットストリームの CRC 値と一致しない場合は、デバイスはコンフィギュレーションされません。CRC がディスエーブルの場合、CRC 値の代わりに定数値がビットストリームに挿入され、デバイスで CRC 値は算出されません。
BITSTREAM.GENERAL.PERFRAMECRC	No	No、Yes	ビットストリームに一定間隔で CRC 値を挿入します。これらの値は入力されるビットストリームのインテグリティを検証して、コンフィギュレーション データがデバイスにロードされる前にエラー (INIT_B ピンおよび ICAP の PRERROR ポートに表示) を通知します。これはパーシャル ビットストリームに適していますが、Yes に設定すると、CRC 値がデバイス ストリーム全体を含め、すべてのビットストリームに挿入されます。

表 40: Zynq UltraScale+ MPSoC のビットストリーム設定 (続き)

設定	デフォルト値	有効な値	説明
BITSTREAM.GENERAL.SYSMON POWERDOWN	Disable	Disable、 Enable	SYSMON をパワーダウンできるようにして節電します。 SYSMON を永久にパワーダウンする場合にのみ推奨されます。
BITSTREAM.GENERAL.DISABLE_ JTAG	No	No、Yes	コンフィギュレーション後に JTAG を介したバウンダリスキャン (BSCAN) ブロックへのアクセスをディスエーブルにします。
BITSTREAM.GENERAL.JTAG_SYS MON	Enable	Enable、 Disable、 StatusOnly	SYSMON への JTAG 接続をイネーブルまたはディスエーブルにします。
BITSTREAM.READBACK.ICAP_SE LECT	Auto	Auto、Top、 Bottom	上または下の ICAP ポートを選択します。
BITSTREAM.READBACK.ACTIVER ECONFIG	No	No、Yes	コンフィギュレーション中に GHIGH および GSR がアサートされないようにします。これは、アクティブ パーシャル リコンフィギュレーション向上機能に必要です。
BITSTREAM.READBACK.SECURI TY	None	None、 Level1、Level2	リードバックおよびリコンフィギュレーションをディスエーブルにするかどうかを指定します。 Level1 に設定するとリードバックがディスエーブルになります。
BITSTREAM.STARTUP.DONE_CY CLE	4	4、1、2、3、 5、6、Keep	FPGA Done 信号をアクティブにするスタートアップ フェーズを選択します。DonePipe=Yes の場合、Done は遅延されます。
BITSTREAM.STARTUP.GTS_CYCL E	5	5、1、2、3、 4、6、Done、 Keep	I/O バッファへの内部トライステート制御を解放するスタートアップ フェーズを選択します。
BITSTREAM.STARTUP.GWE_CYC LE	6	6、1、2、3、 4、5、Done、 Keep	フリップフロップ、LUT RAM、およびシフトレジスタへの内部イネーブルをアサートするスタートアップ フェーズを選択します。BRAM もイネーブルにします。このスタートアップフェーズの前は、ブロック RAM の書き込みおよび読み出しの両方がディスエーブルです。
BITSTREAM.STARTUP.LCK_CYCL E	NoWait	NoWait、0、 1、2、3、4、 5、6	MMCM/PLL がロックされるまで待機するスタートアップフェーズを選択します。NoWait に設定すると、スタートアップシーケンスは MMCM/PLL がロックされるまで待機されません。
BITSTREAM.STARTUP.MATCH_C YCLE	Auto	Auto、 NoWait、0、 1、2、3、4、 5、6	デジタル制御インピーダンス (DCI) 一致信号がアサートされるまで待機するスタートアップサイクルを指定します。DCI マッチは MATCH_CYCLE では開始しません。スタートアップシーケンスは DCI が一致するまでこのサイクルで待機します。DCI が一致するのにかかる時間にはさまざまな要素が影響するので、スタートアップシーケンスが完了するのに必要な CCLK サイクル数は、同じシステムでも異なる場合があります。DONE が High になるまでコンフィギュレーションソリューションで CCLK を駆動するのが理想的です。 Auto に設定すると、write_bitstream によりデザインで DCI I/O 規格が検索されます。DCI 規格が存在する場合、write_bitstream が BITSTREAM.STARTUP.MATCH_CYCLE=2 を使用します。存在しない場合、write_bitstream は BITSTREAM.STARTUP.MATCH_CYCLE=NoWait を使用します。

トリガー ステート マシンの言語記述

トリガー ステート マシン言語は、ILA デバッグ コアのアドバンス トリガー ロジックにマップされる複雑なトリガー 条件を記述するために使用されます。トリガー ステート マシンには、次のような機能があります。

- 最大 16 までのステート。
- 複雑なステート トランザクションに 1 ～ 3 方向条件分岐を使用。
- イベントのカウント、タイマーのインプリメントなどに 4 つのビルトイン 16 ビット カウンターを使用。
- トリガー ステート マシンの実行ステータスを監視するために 4 つのビルトイン フラグを使用。
- トリガー アクション。

ステート

ステート マシン プログラムでは最大 16 ステートまでを宣言できます。各ステートは、ステート宣言と本体で構成されます。

```
state <state_name>:  
    <state_body>
```

goto アクション

ステート間のトランザクションには goto アクションが使用されます。次は、goto アクションを使用して、トリガー前に 1 つのステートから別のステートへの遷移する例です。

```
state my_state_0:  
    goto my_state_1;  
state my_state_1:  
    trigger;
```

条件分岐

トリガー ステート マシン言語では、ステートごとに 1 ～ 3 方向の条件分岐がサポートされます。

- 1 方向分岐では、if/elseif/else/endif 構文は使用されず、goto アクションが使用されます。

```
state my_state_0:
  goto my_state_1;
```

- 2 方向分岐では、goto アクションと共に if/else/endif 構文も使用されます。

```
state my_state_0:
  if (<condition1>) then
    goto my_state_1;
  else
    goto my_state_0;
  endif
```

- 3 方向分岐では、goto アクションと共に if/else/elseif/endif 構文も使用されます。

```
state my_state_0:
  if (<condition1>) then
    goto my_state_1;
  elseif (<condition2>) then
    goto my_state_2;
  else
    goto my_state_0;
  endif
```

上記に <condition1> および <condition2> で示されている条件文の作成方法は、「条件文」セクションを参照してください。

関連情報

[条件文](#)

カウンター

4 つのビルトイン 16 ビット カウンターには、それぞれ \$counter0、\$counter1、\$counter2、\$counter3 という名前が付いています。カウンターは、条件文内でリセット、インクリメント、使用できます。

- カウンターをリセットするには、reset_counter アクションを使用します。

```
state my_state_0:
  reset_counter $counter0;
  goto my_state_1;
```

- カウンターをインクリメントするには、increment_counter アクションを使用します。

```
state my_state_0:
  increment_counter $counter3;
  goto my_state_1;
```

条件文内でのカウンターの使用方法の詳細は、「条件文」を参照してください。

関連情報

[条件文](#)

フラグ

フラグは、トリガー ステート マシン プログラムが実行されているときの進捗状況を監視するために使用できます。4 つのビルトイン フラグの名前は、それぞれ \$flag0、\$flag1、\$flag2、\$flag3 です。フラグは、設定およびクリアできます。

- フラグを設定するには、set_flag アクションを使用します。

```
state my_state_0:
  set_flag $flag0;
  goto my_state_1;
```

- フラグをクリアするには、clear_flag アクションを使用します。

```
state my_state_0:
  clear_flag $flag2;
  goto my_state_1;
```

条件文

デバッグ プローブの条件

デバッグ プローブの条件は、2 方向および 3 方向の分岐条件文内で使用できます。各デバッグ プローブ条件では、そのデバッグ プローブが接続された ILA の PROBE ポートの 1 つのトリガー コンパレータが使用されます。



重要: 各 PROBE ポートは、1 ~ 16 個のトリガー コンパレータを含めてコンパイル時にコンフィギュレーションできます。つまり、特定のデバッグ プローブは、PROBE ポートにコンフィギュレーションされたコンパレータ数によって、トリガー ステート マシン プログラム全体においてデバッグ プローブ条件文で 1 ~ 16 回までしか使用できません。

デバッグ プローブ条件には、比較演算子と値が含まれます。有効なデバッグ プローブ条件の比較演算子は次のとおりです。

- == (等価)
- != (不等価)
- > (大なり)
- < (小なり)
- >= (以上)
- <= (以下)

有効な値は、次の形式になります。

```
<bit_width>'<radix><value>
```

説明:

- <bit width>: プローブ幅 (ビット)。

- `<radix>`: 次のいずれかになります。
 - `b` (2 進数)
 - `h` (16 進数)
 - `u` (符号なし 10 進数)
- `<value>`: 次のいずれかになります。
 - `0` (論理 0)
 - `1` (論理 1)
 - `X` (ドントケア)
 - `R` (0 から 1 への遷移): 1 ビット プローブの場合にのみ有効
 - `F` (1 から 0 への遷移): 1 ビット プローブの場合にのみ有効
 - `B` (両遷移): 1 ビット プローブの場合にのみ有効
 - `N` (遷移なし): 1 ビット プローブの場合にのみ有効

有効なデバッグ プローブ条件値の例を次に示します。

- 1 ビットの 2 進数値 0

```
1'b0
```

- 12 ビットの 16 進数値 7A

```
12'h07A
```

- 9 ビットの整数値 123

```
9'u123
```

デバッグ プローブ条件文の例を次に示します。

- 値が 0 の `abc` という 1 ビット デバッグ プローブ

```
if (abc == 1'b0) then
```

- 値が 456 以上の `xyz` という 23 ビット デバッグ プローブ

```
if (xyz >= 23'u456) then
```

- 値が 16 進数 A5 ではない `klm` という 23 ビット デバッグ プローブ

```
if (klm != 23'h0000A5) then
```

複数のデバッグ プローブ条件文の例を次に示します。

- 2 つのデバッグ プローブの比較を OR 関数で組み合わせ

```
if ((xyz >= 23'u456) || (abc == 1'b0)) then
```

- 2 つのデバッグ プローブの比較を AND 関数で組み合わせ

```
if ((xyz >= 23'u456) && (abc == 1'b0)) then
```

- 3つのデバッグ プローブの比較を OR 関数で組み合わせ

```
if ((xyz >= 23'u456) || (abc == 1'b0) || (klm != 23'h0000A5)) then
```

- 3つのデバッグ プローブの比較を AND 関数で組み合わせ

```
if ((xyz >= 23'u456) && (abc == 1'b0) && (klm != 23'h0000A5)) then
```

カウンター条件

カウンター条件は、2 方向および 3 方向の分岐条件文内で使用できます。各カウンター条件で 1 つのカウンター コンパレータが使用されます。



重要: 各カウンターに含まれるカウンター コンパレータは 1 つのみです。つまり、特定のカウンターはトリガー ステート マシン プログラム全体においてカウンター条件内で 1 回しか使用できません。

プローブ ポート条件には、比較演算子と値が含まれます。有効なプローブ条件の比較演算子は次のとおりです。

- == (等価)
- != (不等価)



重要: 各カウンターの幅は常に 16 ビットです。

有効なカウンター条件値の例を次に示します。

- 16 ビットの 2 進数値 0

```
16'b0000_0000_0000_0000
16'b0000000000000000
```

- 16 ビットの 16 進数値 7A

```
16'h007A
```

- 16 ビットの整数値 123

```
16'u123
```

カウンター条件文の例を次に示します。

- 2 進数 0 に等しいカウンター \$counter0

```
($counter0 == 16'b0000000000000000)
```

- 10 進数 23 でないカウンター \$counter2

```
($counter2 != 16'u23)
```

デバッグ プローブとカウンター条件の組み合わせ

デバッグ プローブ条件とカウンター条件を組み合わせ、1 つの条件を作成できます。この場合、次の規則に従う必要があります。

- すべてのデバッグ プローブ比較を同じ || (OR) または && (AND) 演算子を使用して組み合わせる必要があります。

- 組み合わせられたデバッグ プローブ条件は、デバッグ プローブ比較を組み合わせるのに使用した演算子に関係なく、|| (OR) または && 演算子のいずれかを使用してカウンタ条件と組み合わせることができます。

複数のデバッグ プローブとカウンタ条件文を組み合わせる例を次に示します。

- 2つのデバッグ プローブ比較を OR で組み合わせてから、AND を使用してカウンタ条件と組み合わせる場合

```
if ((xyz >= 23'u456) || (abc == 1'b0)) && ($counter0 == 16'u0023)) then
```

- 2つのデバッグ プローブ比較を AND で組み合わせてから、OR を使用してカウンタ条件と組み合わせる場合

```
if ((xyz >= 23'u456) && (abc == 1'b0)) || ($counter0 == 16'u0023)) then
```

- 3つのデバッグ プローブ比較を OR で組み合わせてから、AND を使用してカウンタ条件と組み合わせる場合

```
if ((xyz >= 23'u456) || (abc == 1'b0) || (klm != 23'h0000A5)) &&
($counter0 ==
16'u0023)) then
```

- 3つのデバッグ プローブ比較を AND で組み合わせてから、OR を使用してカウンタ条件と組み合わせる場合

```
if ((xyz >= 23'u456) && (abc == 1'b0) && (klm != 23'h0000A5)) ||
($counter0 ==
16'u0023)) then
```

トリガー ステート マシンの言語文法

注記:

- 大文字/小文字が区別されます。
- コメント文字は # で、# 文字の後に含まれるものは無視されます。
- 'THING': THING は終端
- {<thing>}: 0 以上の thing
- [<thing>]: 0 または 1 つの thing

```
<program> ::= <state_list>
<state_list> ::= <state_list> <state> | <state>
<state> ::= 'STATE' <state_label> ':' <if_condition> | <action_block>
```

```
<action_block> ::= <action_list> 'GOTO' <state_label> ';'
| <action_list> 'TRIGGER' ';'
| 'GOTO' <state_label> ';'
| 'TRIGGER' ';'
<action_list> ::= <action_statement> | <action_list> <action_statement>
<action_statement> ::= 'SET_FLAG' <flag_name> ';'
| 'CLEAR_FLAG' <flag_name> ';'
| 'INCREMENT_COUNTER' <counter_name> ';'
| 'RESET_COUNTER' <counter_name> ';'
<if_condition> ::= 'IF' '(' <condition> ')' 'THEN' <actionblock>
| 'ELSEIF' '(' <condition> ')' 'THEN' <actionblock>
| 'ELSE' <actionblock>
| 'ENDIF'
<condition> ::= <probe_match_list>
| <counter_match>
| <probe_counter_match>
<probe_counter_match> ::= '(' <probe_counter_match> ')'
| <probe_match_list> <boolean_logic_op> <counter_match>
```

```

| <counter_match> <boolean_logic_op> <probe_match_list>
<probe_match_list> ::= '(' <probe_match> ')'
| <probe_match>
<probe_match> ::= <probe_match_list> <boolean_logic_op> <probe_match_list>
| <probe_name> <compare_op> <constant>
| <constant> <compare_op> <probe_name>
<counter_match> ::= '(' <counter_match> ')'
| <counter_name> <compare_op> <constant>
| <constant> <compare_op> <counter_name>
<constant> ::= <integer_constant>
| <hex_constant>
| <binary_constant>
<compare_op> ::= '=' | '!=' | '>' | '>=' | '<' | '<='
<boolean_logic_op> ::= '&&' | '||'
--- The following are in regular expression format to simplify expressions:
--- [A-Z0-9] means match any single character in AB...Z,0..9
--- [AB]+ means match [AB] one or more times like A, AB, ABAB, AAA, etc
--- [AB]* means match [AB] zero or more times
<probe_name>      ::= [A-Z_\[\]</>][A-Z_0-9\[\]</>]+
<state_label>     ::= [A-Z_][A-Z_0-9]+
<flag_name>       ::= \$FLAG[0-3]
<counter_name>    ::= \$COUNTER[0-3]
<hex_constant>    ::= <integer>*'h'<hex_digit>+
<binary_constant> ::= <integer>*'b'<binary_digit>+
<integer_constant> ::= <integer>*'u'<integer_digit>+
<integer>         ::= <digit>+
<hex_digit>       ::= [0-9ABCDEFBN_]
<binary_digit>    ::= [01XRFBN_]
<digit>          ::= [0-9]

```

下位 SVF JTAG コマンド

下位 JTAG コマンドを使用すると、複数の FPGA JTAG チェーンをスキャンできます。チェーンの操作用に生成された SVF コマンドでは、これらの下位コマンドを使用してチェーンの FPGA にアクセスします。

この付録では、これらのコマンドの概要を説明します。詳細は、『[Serial Vector Format Specification document](#)』を参照してください。

HDR (Header Data Register)、HIR (Header Instruction Register)

構文

```
HDR length [TDI (tdi)] [TDO (tdo)] [MASK (mask)] [SMASK (smask)];  
HIR length [TDI (tdi)] [TDO (tdo)] [MASK (mask)] [SMASK (smask)];
```

目的

各スキャン操作の前にシフトインするデフォルトのヘッダー パターンを指定します。ヘッダー パターンは、スキャンパスの該当するコンポーネントより先にあるデバイスに対応するため、スキャン ステートメントを先行ビットでパディングする方法を指定します。

一般情報

HDR (Header Data Register) は、後続のすべての SDR コマンドの先頭に追加するデフォルトのヘッダー パターンを指定します。HIR (Header Instruction Register) は、後続のすべての SIR コマンドの先頭に追加するデフォルトのヘッダー パターンを指定します。ヘッダー コマンドには、対となるトレーラー コマンド (TIR, TDR) があります (次のセクションで説明)。ヘッダーを削除するには、長さを 0 に設定します。

TDR (Trailer Data Register)、TIR (Trailer Instruction Register)

構文

```
TDR length [TDI (tdi)] [TDO (tdo)] [MASK (mask)] [SMASK (smask)];  
TIR length [TDI (tdi)] [TDO (tdo)] [MASK (mask)] [SMASK (smask)];
```

目的

各スキャン操作の後にシフトインするデフォルトのトレーラー パターンを指定します。トレーラー パターンは、スキャン パスの該当するコンポーネントより後にあるデバイスに対応するため、スキャン ステートメントを後置ビットでパディングする方法を指定します。

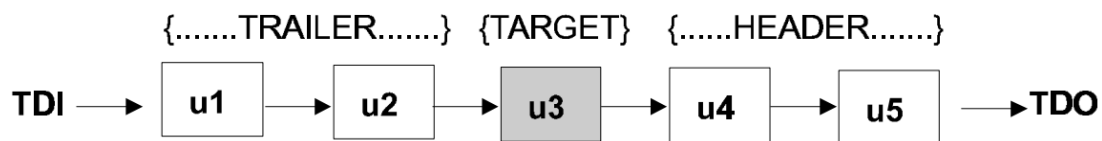
一般情報

TDR (Trailer Data Register) は、後続のすべての SDR コマンドの末尾に追加するトレーラー パターンを指定します。TIR (Trailer Instruction Register) は、後続のすべての SIR コマンドの末尾に追加するデフォルトのトレーラー パターンを指定します。トレーラーを削除するには、長さを 0 に設定します。

例

この例では、SVF ファイルは ASIC 用に開発されています。この ASIC を、次の図に示すように、ボードの u3 に配置します。

図 181: TDR 例



ASIC 用に開発された SVF ステートメントは、u3 の前後のデバイスに対応するよう適切なヘッダーおよびトレーラー ステートメントを定義することにより、最小限の変更で再利用できます。この例では、デバイス u4 および u5 用にヘッダー パターンを定義し、デバイス u2 および u1 用にトレーラー パターンを定義します。オプションのパラメーターは、どの順序で指定してもかまいませんが、それぞれ 1 回のみ指定可能です。TDI、TDO、MASK、または SMASK に指定する 16 進文字列は、length パラメーターで示される最大値を超える値にすることはできません。明示的に指定されていない場合は、先頭は 0 であると想定されます。

scan_ir_hw

ハードウェア JTAG (hw_jtag) で Shift-IR を実行します。

構文

```
scan_ir_hw_jtag [-tdi <arg>] [-tdo <arg>] [-mask <arg>] [-smask <arg>] [-quiet]
[-verbose] <length>
```

一般情報

scan_ir_hw_jtag コマンドは、JTAG インターフェイス ターゲットの命令レジスタに取り込むスキャン パターンを指定します。このコマンドは、open_hw_target -jtag_mode コマンドを使用してハードウェア ターゲット (hw_target) を JTAG モードで開いたときに作成されたハードウェア JTAG (hw_jtag) オブジェクトをターゲットとします。scan_ir_hw_jtag コマンドで指定したスキャン パターンをシフトする前に hw_jtag オブジェクトをターゲットとすると、最後に定義されたヘッダー プロパティ (HIR) が指定したデータ パターンの先頭に追加され、最後に定義されたトレーラー プロパティ (TIR) がデータ パターンの末尾に追加されます。

-tdi、-tdo、-mask、または -smask オプションで指定する 16 進数文字列で表されたビット数は、<length> で指定した最大値以下にする必要があります。

scan_ir_hw_jtag コマンドを実行すると、hw_jtag からキャプチャされた TDO データを含む 16 進配列が返されるか、正常に実行されなかった場合はエラーが返されます。

例

次の例では、JTAG 命令レジスタから 24 ビット値をスキャンしています。

```
scan_ir_hw_jtag 24
```

次の例では、24 ビット値 0x00_0010 (LSB が先頭) を TDI に送信し、TDO 出力をキャプチャして、0xF3_FFFF のマスクを適用して返された TDO 値を tdo 0x81_8181 で指定した値と比較しています。

```
scan_ir_hw_jtag 24 -tdi 000010 -tdo 818181 -mask F3FFFF -smask 0
```

scan_dr_hw

ハードウェア JTAG で Shift-DR を実行します。

構文

```
scan_dr_hw_jtag [-tdi <arg>] [-tdo <arg>] [-mask <arg>] [-smask <arg>] [-quiet]
[-verbose] <length>
```

一般情報

`scan_dr_hw_jtag` コマンドは、JTAG インターフェイス ターゲットのデータ レジスタに挿入するスキャン パターンを指定します。このコマンドは、`open_hw_target -jtag_mode` コマンドを使用してハードウェア ターゲット (`hw_target`) を JTAG モードで開いたときに作成されたハードウェア JTAG (`hw_jtag`) オブジェクトをターゲットとします。`scan_dr_hw_jtag` コマンドで指定したスキャン パターンをシフトする前に `hw_jtag` オブジェクトをターゲットとすると、最後に定義されたヘッダー プロパティ (HDR) が指定したデータ パターンの先頭に追加され、最後に定義されたトレーラー プロパティ (TDR) がデータ パターンの末尾に追加されます。

`scan_dr_hw_jtag` コマンドを実行すると、`hw_jtag` からキャプチャされた TDO データを含む 16 進配列が返されるか、正常に実行されなかった場合はエラーが返されます。

例

次の例では、JTAG データ レジスタから 24 ビット値をスキャンしています。

```
scan_dr_hw_jtag 24
```

次の例では、24 ビット値 0x00_0010 (LSB が先頭) を TDI に送信し、データ出力 TDO をキャプチャして、0xF3_FFFF のマスクを適用して返された TDO 値を `-tdo 0x81_8181` で指定した値と比較しています。

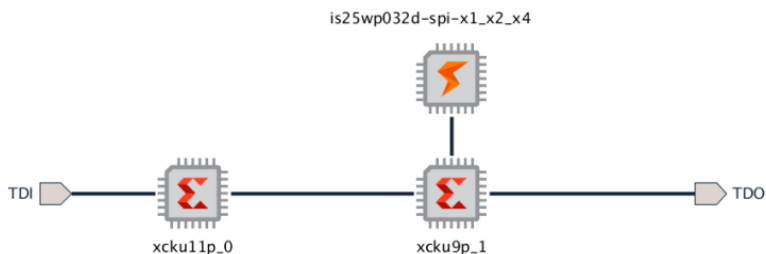
```
scan_dr_hw_jtag 24 -tdi 000010 -tdo 818181 -mask F3FFFF -smask 0
```

マルチチェーン SVF 操作

次の例に、SVF チェーンに対する操作の処理方法を示します。

2 つのデバイス `xcku11` および `xcku9` はチェーン接続されています。コンフィギュレーション メモリは、チェーンの 2 つ目のデバイス (`xcku9`) に接続されています。このコンフィギュレーション メモリにアクセスするには、SVF で HIR、HDR、TIR、および TDR コマンドを使用したコマンドを生成します。このコンフィギュレーション メモリに書き込むために生成されるコマンドでは、チェーンの長さが考慮され、この情報が下位 JTAG 操作に組み込まれます。

図 182: マルチチェーン SVF 操作の例



生成された .svf ファイルには、次の操作が含まれます。

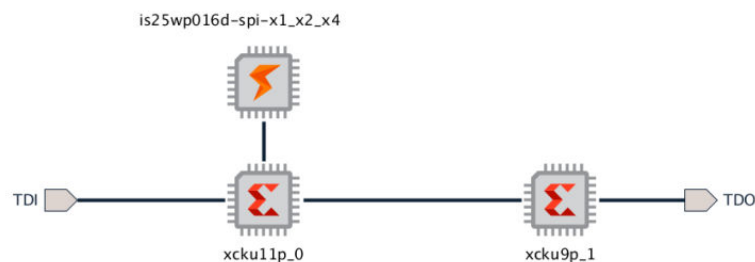
```
HIR 0 ;
TIR 6 TDI (3f) SMASK (3f) ;
HDR 0 ;
TDR 1 TDI (00) SMASK (01) ;
// config/idcode
SIR 6 TDI (09) ;
SDR 32 TDI (00000000) TDO (0484a093) MASK (0fffffff) ;
// config/jprog
STATE RESET;
STATE IDLE;
SIR 6 TDI (0b) ;
SIR 6 TDI (14) ;
// Modify the below delay for config_init operation (0.100000 sec typical,
0.100000
sec maximum)
RUNTEST 0.100000 SEC;
// config/jprog/poll
RUNTEST 10000 TCK;
SIR 6 TDI (14) TDO (11) MASK (31) ;
// config/slr
SIR 6 TDI (05) ;
```

コンフィギュレーション メモリが最初のデバイスに接続されている場合のマルチチェーン SVF 操作

この例では、ku9 デバイスにアクセスするため、TIR および TDR 命令に SMASK 値 0000 0011 1111 (0x3f) が使用されています。チェーンの 2 つ目のデバイスにアクセスするには、マスク値を挿入した後、SIR および SDR 命令を挿入します。SIR および SDR 命令は、HIR、HDR、TIR、および TDR 情報を統合します。

最初のデバイス (xcu11) に接続されているコンフィギュレーション メモリをプログラムする場合は、SVF で生成されるコマンドは異なるものになります。

図 183: コンフィギュレーション メモリが最初のデバイスに接続されている場合のマルチチェーン SVF 操作の例



```
HIR 6 TDI (3f) SMASK (3f) ;
TIR 0 ;
HDR 1 TDI (00) SMASK (01) ;
TDR 0 ;
// config/idcode
SIR 6 TDI (09) ;
SDR 32 TDI (00000000) TDO (04a4e093) MASK (0fffffff) ;
// config/jprog
STATE RESET;
STATE IDLE;
SIR 6 TDI (0b) ;
```

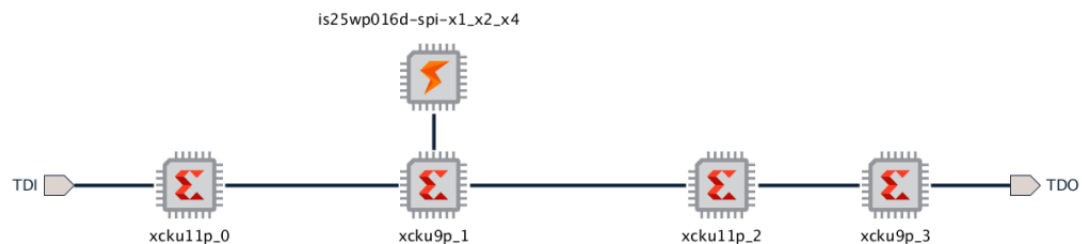
```
SIR 6 TDI (14) ;
// Modify the below delay for config_init operation (0.100000 sec typical,
0.100000
sec maximum)
RUNTEST 0.100000 SEC;
// config/jprog/poll
RUNTEST 10000 TCK;
SIR 6 TDI (14) TDO (11) MASK (31) ;
// config/slr
SIR 6 TDI (05) ;
```

コンフィギュレーション メモリがチェーンの 2 つ目のデバイスに接続されている場合のマルチチェーン SVF 操作

この例では、ku11 デバイスにアクセスするため、HIR および HDR 命令に SMASK 値 0011 1111 (0x3f) が使用されています。チェーンの最初のデバイスにアクセスするには、マスク値を挿入した後、SIR および SDR 命令を挿入します。SIR および SDR 命令は、HIR、HDR、TIR、および TDR 情報を統合します。

4 つのデバイス xcku11、xcku9、xcku11、および xcku9 が接続されているチェーンを考えます。コンフィギュレーション メモリは、チェーンの 2 つ目のデバイス (xcku9) に接続されています。このデバイスにアクセスするには、HIR と TIR 命令の両方を使用します。

図 184: コンフィギュレーション メモリがチェーンの 2 つ目のデバイスに接続されている場合のマルチチェーン SVF 操作の例



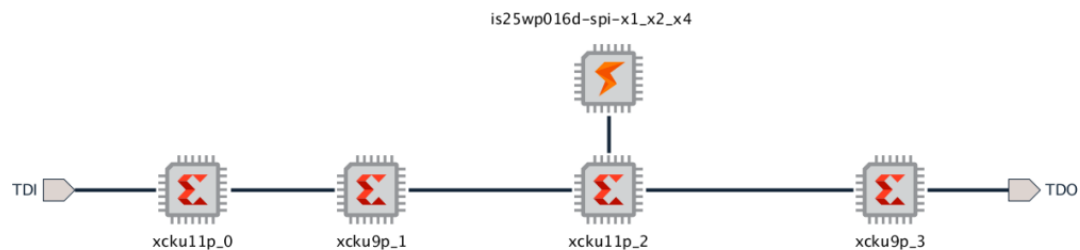
```
HIR 12 TDI (0fff) SMASK (0fff) ;
TIR 6 TDI (3f) SMASK (3f) ;
HDR 2 TDI (00) SMASK (03) ;
TDR 1 TDI (00) SMASK (01) ;
// config/idcode
SIR 6 TDI (09) ;
SDR 32 TDI (00000000) TDO (0484a093) MASK (0fffffff) ;
// config/jprog
STATE RESET;
STATE IDLE;
SIR 6 TDI (0b) ;
SIR 6 TDI (14) ;
// Modify the below delay for config_init operation (0.100000 sec typical,
0.100000
sec maximum)
RUNTEST 0.100000 SEC;
```

```
// config/jprog/poll
RUNTEST 10000 TCK;
SIR 6 TDI (14) TDO (11) MASK (31) ;
// config/slr
SIR 6 TDI (05) ;
```

コンフィギュレーション メモリがチェーンの 3 つ目のデバイスに接続されている場合のマルチチェーン SVF 操作

コンフィギュレーション メモリがチェーンの 3 つ目のデバイス (xcku9) に接続されている場合は、次のようになります。

図 185: コンフィギュレーション メモリがチェーンの 3 つ目のデバイスに接続されている場合のマルチチェーン SVF 操作の例



```
HIR 6 TDI (3f) SMASK (3f) ;
TIR 12 TDI (0fff) SMASK (0fff) ;
HDR 1 TDI (00) SMASK (01) ;
TDR 2 TDI (00) SMASK (03) ;
// config/idcode
SIR 6 TDI (09) ;
SDR 32 TDI (00000000) TDO (04a4e093) MASK (0fffffff) ;
// config/jprog
STATE RESET;
STATE IDLE;
SIR 6 TDI (0b) ;
SIR 6 TDI (14) ;
// Modify the below delay for config_init operation (0.100000 sec typical,
0.100000
sec maximum)
RUNTEST 0.100000 SEC;
// config/jprog/poll
RUNTEST 10000 TCK;
SIR 6 TDI (14) TDO (11) MASK (31) ;
// config/slr
SIR 6 TDI (05) ;
```

hw_server でサポートされる JTAG ケーブルおよびデバイス

hw_server でサポートされる互換性のある JTAG ダウンロード ケーブルおよびデバイスは、次のとおりです。

- ザイリンクス SmartLynq データ ケーブル (HW-SMARTLYNQ-G/DLC20)
- ザイリンクス プラットフォーム ケーブル USB II (DLC10)
- ザイリンクス プラットフォーム ケーブル USB (DLC9G、DLC9LP、DLC9)
- Digilent 社 JTAG-HS1
- Digilent 社 JTAG-HS2
- Digilent 社 JTAG-HS3
- Digilent 社 JTAG-SMT1
- Digilent 社 JTAG-SMT2

</

表 41: Artix-7 メモリ デバイスでサポートされるフラッシュ メモリ デバイスのサポート (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	m29ew	28f00bm29ew	2,048	x16、x8
BPI	Micron	m29w	m29w640gh	64	x16、x8
BPI	Micron	m29w	m29w640gl	64	x16、x8
BPI	Micron	m29w	m29w128gh	128	x16、x8
BPI	Micron	m29w	m29w128gl	128	x16、x8
BPI	Micron	m29w	m29w256gh	256	x16、x8
BPI	Micron	m29w	m29w256gl	256	x16、x8
BPI	Micron	mt28ew	mt28ew128a	128	x16、x8
BPI	Micron	mt28ew	mt28ew256a	256	x16、x8
BPI	Micron	mt28ew	mt28ew512a	512	x16、x8
BPI	Micron	mt28ew	mt28ew01ga	1,024	x16、x8
BPI	Micron	mt28fw	mt28fw02gb	2,048	x16
BPI	Micron	p30	28f640p30b	64	x16
BPI	Micron	p30	28f640p30t	64	x16
BPI	Micron	p30	28f128p30b	128	x16
BPI	Micron	p30	28f128p30t	128	x16
BPI	Micron	p30	28f256p30b	256	x16
BPI	Micron	p30	28f256p30t	256	x16
BPI	Micron	p30	28f512p30b	512	x16
BPI	Micron	p30	28f512p30e	512	x16
BPI	Micron	p30	28f512p30t	512	x16
BPI	Micron	p30	28f00ap30b	1,024	x16
BPI	Micron	p30	28f00ap30e	1,024	x16
BPI	Micron	p30	28f00ap30t	1,024	x16
BPI	Micron	p30	28f00bp30e	2,048	x16
BPI	Micron	p33	28f640p33b	64	x16
BPI	Micron	p33	28f640p33t	64	x16
BPI	Micron	p33	28f128p33b	128	x16
BPI	Micron	p33	28f128p33t	128	x16
BPI	Micron	p33	28f256p33b	256	x16
BPI	Micron	p33	28f256p33t	256	x16
BPI	Micron	p33	28f512p33b	512	x16
BPI	Micron	p33	28f512p33e	512	x16
BPI	Micron	p33	28f512p33t	512	x16
BPI	Micron	p33	28f00ap33b	1,024	x16
BPI	Micron	p33	28f00ap33e	1,024	x16
BPI	Micron	p33	28f00ap33t	1,024	x16
BPI	Spansion	s29glxxp	s29gl128p	128	x16、x8

表 41: Artix-7 メモリ デバイスでサポートされるフラッシュ メモリ デバイスのサポート (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Spansion	s29glxxp	s29gl256p	256	x16、x8
BPI	Spansion	s29glxxp	s29gl512p	512	x16、x8
BPI	Spansion	s29glxxp	s29gl01gp	1,024	x16、x8
BPI	Spansion	s29glxxp	s70gl02gp	2,048	x16
BPI	Spansion	s29glxxs	s29gl128s	128	x16
BPI	Spansion	s29glxxs	s29gl256s	256	x16
BPI	Spansion	s29glxxs	s29gl512s	512	x16
BPI	Spansion	s29glxxs	s29gl01gs	1,024	x16
BPI	Spansion	s29glxxs	s70gl02gs	2,048	x16
BPI	Spansion	s29glxxt	s29gl512t	512	x16、x8
BPI	Spansion	s29glxxt	s29gl01gt	1,024	x16、x8
BPI	Spansion	s29glxxt	s70gl02gt	2,048	x16、x8
BPI	Macronix	mx29gl	mx29gl128f	128	x16、x8
SPI	ISSI	is25lp	is25lp080d	8	x1、x2、x4
SPI	ISSI	is25lp	is25lp016d	16	x1、x2、x4
SPI	ISSI	is25lp	is25lp032d	32	x1、x2、x4
SPI	ISSI	is25lp	is25lp064a	64	x1、x2、x4
SPI	ISSI	is25lp	is25lp128f	128	x1、x2、x4
SPI	ISSI	is25lp	is25lp256d	256	x1、x2、x4
SPI	ISSI	is25lp	is25lp512m	512	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp080d	8	x1、x2、x4
SPI	ISSI	is25wp	is25wp016d	16	x1、x2、x4
SPI	ISSI	is25wp	is25wp032d	32	x1、x2、x4
SPI	ISSI	is25wp	is25wp064a	64	x1、x2、x4
SPI	ISSI	is25wp	is25wp128f	128	x1、x2、x4
SPI	ISSI	is25wp	is25wp256d	256	x1、x2、x4
SPI	ISSI	is25wp	is25wp512m	512	x1、x2、x4、x8
SPI	Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Micron	mt25ql	mt25ql512	512	x1、x2、x4
SPI	Micron	mt25ql	mt25ql01g	1,024	x1、x2、x4
SPI	Micron	mt25ql	mt25ql02g	2,048	x1、x2、x4
SPI	Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Micron	mt25qu	mt25qu512	512	x1、x2、x4
SPI	Micron	mt25qu	mt25qu01g	1,024	x1、x2、x4
SPI	Micron	mt25qu	mt25qu02g	2,048	x1、x2、x4
SPI	Micron	n25q	n25q32-1.8v	32	x1、x2、x4

表 41: Artix-7 メモリ デバイスでサポートされるフラッシュ メモリ デバイスのサポート (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	Micron	n25q	n25q32-3.3v	32	x1、x2、x4
SPI	Micron	n25q	n25q64-1.8v	64	x1、x2、x4
SPI	Micron	n25q	n25q64-3.3v	64	x1、x2、x4
SPI	Spansion	s25fl1	s25fl116k	16	x1、x2、x4
SPI	Spansion	s25fl1	s25fl132k	32	x1、x2、x4
SPI	Spansion	s25fl1	s25fl164k	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl064l	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl128l	128	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl256l	256	x1、x2、x4
SPI	Spansion	s25flxxxp	s25fl032p	32	x1、x2、x4
SPI	Spansion	s25flxxxp	s25fl064p	64	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxxx0 [s25fl127s-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxxx1	128	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl256sxxxxxx0	256	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl256sxxxxxx1	256	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl512s	512	x1、x2、x4
SPI	Macronix	mx25l	mx25v8035f	8	x1、x2、x4
SPI	Macronix	mx25l	mx25v1635f	16	x1、x2、x4
SPI	Macronix	mx25l	mx25l3233f	32	x1、x2、x4
SPI	Macronix	mx25l	mx25l6433f	64	x1、x2、x4
SPI	Macronix	mx25l	mx25l12845g [mx25l12835f-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Macronix	mx25l	mx25l25645g [mx25l25635f-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Macronix	mx25l	mx25l51245g [mx66l51235f-spi-x1_x2_x4]	512	x1、x2、x4
SPI	Macronix	mx25u	mx25u8033e	8	x1、x2、x4
SPI	Macronix	mx25u	mx25u1635f	16	x1、x2、x4
SPI	Macronix	mx25u	mx25u3235f	32	x1、x2、x4
SPI	Macronix	mx25u	mx25u6435f	64	x1、x2、x4
SPI	Macronix	mx25u	mx25u12835f	128	x1、x2、x4
SPI	Macronix	mx25u	mx25u25645g [mx25u25635f-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Macronix	mx25u	mx25u51245g [mx66u51235f-spi-x1_x2_x4]	512	x1、x2、x4
SPI	Macronix	mx66l	mx66l1g45g	1,024	x1、x2、x4
SPI	Macronix	mx66l	mx66l2g45g	2,048	x1、x2、x4
SPI	Macronix	mx66u	mx66u1g45g	1,024	x1、x2、x4
SPI	Macronix	mx66u	mx66u2g45g	2,048	x1、x2、x4

Kintex-7 コンフィギュレーションのメモリ デバイス

次の表には、Kintex®-7 デバイスのコンフィギュレーションにサポートされるフラッシュ デバイスで、Vivado® ツールで消去、ブランク チェック、プログラム、および検証可能なものを示します。

この付録にある表には、Vivado ソフトウェアで消去、ブランク チェック、プログラム、検証が可能な不揮発性メモリがザイリンクス ファミリーごとにリストされています。ザイリンクスでは、これらのコンポーネントを含むエンドプロダクトの長期メンテナンスをサポートするため、コンポーネントが新しいデザインには適さなくなったとしても、できる限りこのリストにコンポーネントをリストするようにしています。



重要: 不揮発性メモリの市場は変化が激しいため、利用可能なデバイスおよびそのライフサイクルに関しては、ザイリンクスでは不揮発性メモリの製造業者に確認を取ることをお勧めしています。表に掲載されている特定デバイスの情報は、現在または今後の使用可能状態を確約するものではありません。

表 42: Kintex-7 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	g18	28f128g18f	128	x16
BPI	Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	256	x16
BPI	Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	512	x16
BPI	Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	1,024	x16
BPI	Micron	m29ew	28f064m29ewb	64	x16、x8
BPI	Micron	m29ew	28f064m29ewh	64	x16、x8
BPI	Micron	m29ew	28f064m29ewl	64	x16、x8
BPI	Micron	m29ew	28f064m29ewt	64	x16、x8
BPI	Micron	m29ew	28f128m29ew	128	x16、x8
BPI	Micron	m29ew	28f256m29ew	256	x16、x8
BPI	Micron	m29ew	28f512m29ew	512	x16、x8
BPI	Micron	m29ew	28f00am29ew	1,024	x16、x8
BPI	Micron	m29ew	28f00bm29ew	2,048	x16、x8
BPI	Micron	m29w	m29w640gh	64	x16、x8
BPI	Micron	m29w	m29w640gl	64	x16、x8
BPI	Micron	m29w	m29w128gh	128	x16、x8
BPI	Micron	m29w	m29w128gl	128	x16、x8
BPI	Micron	m29w	m29w256gh	256	x16、x8
BPI	Micron	m29w	m29w256gl	256	x16、x8
BPI	Micron	mt28ew	mt28ew128a	128	x16、x8
BPI	Micron	mt28ew	mt28ew256a	256	x16、x8
BPI	Micron	mt28ew	mt28ew512a	512	x16、x8
BPI	Micron	mt28ew	mt28ew01ga	1,024	x16、x8
BPI	Micron	mt28fw	mt28fw02gb	2,048	x16
BPI	Micron	p30	28f640p30b	64	x16
BPI	Micron	p30	28f640p30t	64	x16

表 42: Kintex-7 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	p30	28f128p30b	128	x16
BPI	Micron	p30	28f128p30t	128	x16
BPI	Micron	p30	28f256p30b	256	x16
BPI	Micron	p30	28f256p30t	256	x16
BPI	Micron	p30	28f512p30b	512	x16
BPI	Micron	p30	28f512p30e	512	x16
BPI	Micron	p30	28f512p30t	512	x16
BPI	Micron	p30	28f00ap30b	1,024	x16
BPI	Micron	p30	28f00ap30e	1,024	x16
BPI	Micron	p30	28f00ap30t	1,024	x16
BPI	Micron	p30	28f00bp30e	2,048	x16
BPI	Micron	p33	28f640p33b	64	x16
BPI	Micron	p33	28f640p33t	64	x16
BPI	Micron	p33	28f128p33b	128	x16
BPI	Micron	p33	28f128p33t	128	x16
BPI	Micron	p33	28f256p33b	256	x16
BPI	Micron	p33	28f256p33t	256	x16
BPI	Micron	p33	28f512p33b	512	x16
BPI	Micron	p33	28f512p33e	512	x16
BPI	Micron	p33	28f512p33t	512	x16
BPI	Micron	p33	28f00ap33b	1,024	x16
BPI	Micron	p33	28f00ap33e	1,024	x16
BPI	Micron	p33	28f00ap33t	1,024	x16
BPI	Spansion	s29glxxxp	s29gl128p	128	x16、x8
BPI	Spansion	s29glxxxp	s29gl256p	256	x16、x8
BPI	Spansion	s29glxxxp	s29gl512p	512	x16、x8
BPI	Spansion	s29glxxxp	s29gl01gp	1,024	x16、x8
BPI	Spansion	s29glxxxp	s70gl02gp	2,048	x16
BPI	Spansion	s29glxxxs	s29gl128s	128	x16
BPI	Spansion	s29glxxxs	s29gl256s	256	x16
BPI	Spansion	s29glxxxs	s29gl512s	512	x16
BPI	Spansion	s29glxxxs	s29gl01gs	1,024	x16
BPI	Spansion	s29glxxxs	s70gl02gs	2,048	x16
BPI	Spansion	s29glxxxt	s29gl512t	512	x16、x8
BPI	Spansion	s29glxxxt	s29gl01gt	1,024	x16、x8
BPI	Spansion	s29glxxxt	s70gl02gt	2,048	x16、x8
BPI	Macronix	mx29gl	mx29gl128f	128	x16、x8
SPI	ISSI	is25lp	is25lp080d	8	x1、x2、x4

表 42: Kintex-7 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	ISSI	is25lp	is25lp016d	16	x1、x2、x4
SPI	ISSI	is25lp	is25lp032d	32	x1、x2、x4
SPI	ISSI	is25lp	is25lp064a	64	x1、x2、x4
SPI	ISSI	is25lp	is25lp128f	128	x1、x2、x4
SPI	ISSI	is25lp	is25lp256d	256	x1、x2、x4
SPI	ISSI	is25lp	is25lp512m	512	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp080d	8	x1、x2、x4
SPI	ISSI	is25wp	is25wp016d	16	x1、x2、x4
SPI	ISSI	is25wp	is25wp032d	32	x1、x2、x4
SPI	ISSI	is25wp	is25wp064a	64	x1、x2、x4
SPI	ISSI	is25wp	is25wp128f	128	x1、x2、x4
SPI	ISSI	is25wp	is25wp256d	256	x1、x2、x4
SPI	ISSI	is25wp	is25wp512m	512	x1、x2、x4、x8
SPI	Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Micron	mt25ql	mt25ql512	512	x1、x2、x4
SPI	Micron	mt25ql	mt25ql01g	1,024	x1、x2、x4
SPI	Micron	mt25ql	mt25ql02g	2,048	x1、x2、x4
SPI	Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Micron	mt25qu	mt25qu512	512	x1、x2、x4
SPI	Micron	mt25qu	mt25qu01g	1,024	x1、x2、x4
SPI	Micron	mt25qu	mt25qu02g	2,048	x1、x2、x4
SPI	Micron	n25q	n25q32-1.8v	32	x1、x2、x4
SPI	Micron	n25q	n25q32-3.3v	32	x1、x2、x4
SPI	Micron	n25q	n25q64-1.8v	64	x1、x2、x4
SPI	Micron	n25q	n25q64-3.3v	64	x1、x2、x4
SPI	Spansion	s25fl1	s25fl116k	16	x1、x2、x4
SPI	Spansion	s25fl1	s25fl132k	32	x1、x2、x4
SPI	Spansion	s25fl1	s25fl164k	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl064l	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl128l	128	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl256l	256	x1、x2、x4
SPI	Spansion	s25flxxxp	s25fl032p	32	x1、x2、x4
SPI	Spansion	s25flxxxp	s25fl064p	64	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxxx0 [s25fl127s-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxxx1	128	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl256sxxxxxx0	256	x1、x2、x4

表 42: Kintex-7 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	Spansion	s25flxxxs	s25fl256sxxxxx1	256	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl512s	512	x1、x2、x4
SPI	Macronix	mx25l	mx25v8035f	8	x1、x2、x4
SPI	Macronix	mx25l	mx25v1635f	16	x1、x2、x4
SPI	Macronix	mx25l	mx25l3233f	32	x1、x2、x4
SPI	Macronix	mx25l	mx25l6433f	64	x1、x2、x4
SPI	Macronix	mx25l	mx25l12845g [mx25l12835f-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Macronix	mx25l	mx25l25645g [mx25l25635f-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Macronix	mx25l	mx25l51245g [mx66l51235f-spi-x1_x2_x4]	512	x1、x2、x4
SPI	Macronix	mx25u	mx25u8033e	8	x1、x2、x4
SPI	Macronix	mx25u	mx25u1635f	16	x1、x2、x4
SPI	Macronix	mx25u	mx25u3235f	32	x1、x2、x4
SPI	Macronix	mx25u	mx25u6435f	64	x1、x2、x4
SPI	Macronix	mx25u	mx25u12835f	128	x1、x2、x4
SPI	Macronix	mx25u	mx25u25645g [mx25u25635f-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Macronix	mx25u	mx25u51245g [mx66u51235f-spi-x1_x2_x4]	512	x1、x2、x4
SPI	Macronix	mx66l	mx66l1g45g	1,024	x1、x2、x4
SPI	Macronix	mx66l	mx66l2g45g	2,048	x1、x2、x4
SPI	Macronix	mx66u	mx66u1g45g	1,024	x1、x2、x4
SPI	Macronix	mx66u	mx66u2g45g	2,048	x1、x2、x4

Spartan-7 コンフィギュレーションのメモリ デバイス

次の表には、Spartan®-7 デバイスのコンフィギュレーションにサポートされるフラッシュ デバイスで、Vivado® ツールで消去、ブランク チェック、プログラム、および検証可能なものを示します。

この付録にある表には、Vivado ソフトウェアで消去、ブランク チェック、プログラム、検証が可能な不揮発性メモリがザイリンクス ファミリごとにリストされています。ザイリンクスでは、これらのコンポーネントを含むエンドプロダクトの長期メンテナンスをサポートするため、コンポーネントが新しいデザインには適さなくなったとしても、できる限りこのリストにコンポーネントをリストするようにしています。



重要: 不揮発性メモリの市場は変化が激しいため、利用可能なデバイスおよびそのライフサイクルに関しては、ザイリンクスでは不揮発性メモリの製造業者に確認を取ることをお勧めしています。表に掲載されている特定デバイスの情報は、現在または今後の使用可能状態を確約するものではありません。

注記: シリアル ペリフェラル インターフェイス (SPI) フラッシュは、Spartan-7 デバイスでサポートされるコンフィギュレーション メモリ ストレージです。バイト ペリフェラル インターフェイス (BPI) フラッシュは、Spartan-7 デバイスではサポートされていません。

表 43: Spartan-7 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	ISSI	is25lp	is25lp080d	8	x1、x2、x4
SPI	ISSI	is25lp	is25lp016d	16	x1、x2、x4
SPI	ISSI	is25lp	is25lp032d	32	x1、x2、x4
SPI	ISSI	is25lp	is25lp064a	64	x1、x2、x4
SPI	ISSI	is25lp	is25lp128f	128	x1、x2、x4
SPI	ISSI	is25lp	is25lp256d	256	x1、x2、x4
SPI	ISSI	is25lp	is25lp512m	512	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp080d	8	x1、x2、x4
SPI	ISSI	is25wp	is25wp016d	16	x1、x2、x4
SPI	ISSI	is25wp	is25wp032d	32	x1、x2、x4
SPI	ISSI	is25wp	is25wp064a	64	x1、x2、x4
SPI	ISSI	is25wp	is25wp128f	128	x1、x2、x4
SPI	ISSI	is25wp	is25wp256d	256	x1、x2、x4
SPI	ISSI	is25wp	is25wp512m	512	x1、x2、x4、x8
SPI	Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Micron	mt25ql	mt25ql512	512	x1、x2、x4
SPI	Micron	mt25ql	mt25ql01g	1,024	x1、x2、x4
SPI	Micron	mt25ql	mt25ql02g	2,048	x1、x2、x4
SPI	Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Micron	mt25qu	mt25qu512	512	x1、x2、x4
SPI	Micron	mt25qu	mt25qu01g	1,024	x1、x2、x4
SPI	Micron	mt25qu	mt25qu02g	2,048	x1、x2、x4
SPI	Micron	n25q	n25q32-1.8v	32	x1、x2、x4
SPI	Micron	n25q	n25q32-3.3v	32	x1、x2、x4
SPI	Micron	n25q	n25q64-1.8v	64	x1、x2、x4
SPI	Micron	n25q	n25q64-3.3v	64	x1、x2、x4
SPI	Spansion	s25fl1	s25fl116k	16	x1、x2、x4
SPI	Spansion	s25fl1	s25fl132k	32	x1、x2、x4
SPI	Spansion	s25fl1	s25fl164k	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl064l	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl128l	128	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl256l	256	x1、x2、x4
SPI	Spansion	s25flxxxp	s25fl032p	32	x1、x2、x4
SPI	Spansion	s25flxxxp	s25fl064p	64	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxx0 [s25fl127s-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxx1	128	x1、x2、x4

表 43: Spartan-7 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	Spansion	s25flxxxs	s25fl256sxxxxxx0	256	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl256sxxxxxx1	256	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl512s	512	x1、x2、x4
SPI	Macronix	mx25l	mx25v8035f	8	x1、x2、x4
SPI	Macronix	mx25l	mx25v1635f	16	x1、x2、x4
SPI	Macronix	mx25l	mx25l3233f	32	x1、x2、x4
SPI	Macronix	mx25l	mx25l6433f	64	x1、x2、x4
SPI	Macronix	mx25l	mx25l12845g [mx25l12835f-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Macronix	mx25l	mx25l25645g [mx25l25635f-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Macronix	mx25l	mx25l51245g [mx66l51235f-spi-x1_x2_x4]	512	x1、x2、x4
SPI	Macronix	mx25u	mx25u8033e	8	x1、x2、x4
SPI	Macronix	mx25u	mx25u1635f	16	x1、x2、x4
SPI	Macronix	mx25u	mx25u3235f	32	x1、x2、x4
SPI	Macronix	mx25u	mx25u6435f	64	x1、x2、x4
SPI	Macronix	mx25u	mx25u12835f	128	x1、x2、x4
SPI	Macronix	mx25u	mx25u25645g [mx25u25635f-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Macronix	mx25u	mx25u51245g [mx66u51235f-spi-x1_x2_x4]	512	x1、x2、x4
SPI	Macronix	mx66l	mx66l1g45g	1,024	x1、x2、x4
SPI	Macronix	mx66l	mx66l2g45g	2,048	x1、x2、x4
SPI	Macronix	mx66u	mx66u1g45g	1,024	x1、x2、x4
SPI	Macronix	mx66u	mx66u2g45g	2,048	x1、x2、x4

Virtex-7 コンフィギュレーションのメモリ デバイス

次の表には、Virtex®-7 デバイスのコンフィギュレーションにサポートされるフラッシュ デバイスで、Vivado® ツールで消去、ブランク チェック、プログラム、および検証可能なものを示します。

この付録にある表には、Vivado ソフトウェアで消去、ブランク チェック、プログラム、検証が可能な不揮発性メモリがザイリンクス ファミリごとにリストされています。ザイリンクスでは、これらのコンポーネントを含むエンドプロダクトの長期メンテナンスをサポートするため、コンポーネントが新しいデザインには適さなくなったとしても、できる限りこのリストにコンポーネントをリストするようにしています。



重要: 不揮発性メモリの市場は変化が激しいため、利用可能なデバイスおよびそのライフサイクルに関しては、ザイリンクスでは不揮発性メモリの製造業者に確認を取ることをお勧めしています。表に掲載されている特定デバイスの情報は、現在または今後の使用可能状態を確約するものではありません。

表 44: Virtex-7 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	g18	28f128g18f	128	x16
BPI	Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	256	x16
BPI	Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	512	x16
BPI	Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	1,024	x16
BPI	Micron	m29ew	28f064m29ewb	64	x16、x8
BPI	Micron	m29ew	28f064m29ewh	64	x16、x8
BPI	Micron	m29ew	28f064m29ewl	64	x16、x8
BPI	Micron	m29ew	28f064m29ewt	64	x16、x8
BPI	Micron	m29ew	28f128m29ew	128	x16、x8
BPI	Micron	m29ew	28f256m29ew	256	x16、x8
BPI	Micron	m29ew	28f512m29ew	512	x16、x8
BPI	Micron	m29ew	28f00am29ew	1,024	x16、x8
BPI	Micron	m29ew	28f00bm29ew	2,048	x16、x8
BPI	Micron	m29w	m29w128gh	128	x16、x8
BPI	Micron	m29w	m29w128gl	128	x16、x8
BPI	Micron	m29w	m29w256gh	256	x16、x8
BPI	Micron	m29w	m29w256gl	256	x16、x8
BPI	Micron	mt28ew	mt28ew128a	128	x16、x8
BPI	Micron	mt28ew	mt28ew256a	256	x16、x8
BPI	Micron	mt28ew	mt28ew512a	512	x16、x8
BPI	Micron	mt28ew	mt28ew01ga	1,024	x16、x8
BPI	Micron	mt28fw	mt28fw02gb	2,048	x16
BPI	Micron	p30	28f640p30b	64	x16
BPI	Micron	p30	28f640p30t	64	x16
BPI	Micron	p30	28f128p30b	128	x16
BPI	Micron	p30	28f128p30t	128	x16
BPI	Micron	p30	28f256p30b	256	x16
BPI	Micron	p30	28f256p30t	256	x16
BPI	Micron	p30	28f512p30b	512	x16
BPI	Micron	p30	28f512p30e	512	x16
BPI	Micron	p30	28f512p30t	512	x16
BPI	Micron	p30	28f00ap30b	1,024	x16
BPI	Micron	p30	28f00ap30e	1,024	x16
BPI	Micron	p30	28f00ap30t	1,024	x16
BPI	Micron	p30	28f00bp30e	2,048	x16
BPI	Spansion	s29glxxp	s29gl128p	128	x16、x8
BPI	Spansion	s29glxxp	s29gl256p	256	x16、x8
BPI	Spansion	s29glxxp	s29gl512p	512	x16、x8

表 44: Virtex-7 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Spansion	s29glxxxp	s29gl01gp	1,024	x16、x8
BPI	Spansion	s29glxxxs	s29gl128s	128	x16
BPI	Spansion	s29glxxxs	s29gl256s	256	x16
BPI	Spansion	s29glxxxs	s29gl512s	512	x16
BPI	Spansion	s29glxxxs	s29gl01gs	1,024	x16
BPI	Spansion	s29glxxxs	s70gl02gs	2,048	x16
BPI	Spansion	s29glxxxt	s29gl512t	512	x16、x8
BPI	Spansion	s29glxxxt	s29gl01gt	1,024	x16、x8
BPI	Spansion	s29glxxxt	s70gl02gt	2,048	x16、x8
BPI	Macronix	mx29gl	mx29gl128f	128	x16、x8
SPI	ISSI	is25lp	is25lp080d	8	x1、x2、x4
SPI	ISSI	is25lp	is25lp016d	16	x1、x2、x4
SPI	ISSI	is25lp	is25lp032d	32	x1、x2、x4
SPI	ISSI	is25lp	is25lp064a	64	x1、x2、x4
SPI	ISSI	is25lp	is25lp128f	128	x1、x2、x4
SPI	ISSI	is25lp	is25lp256d	256	x1、x2、x4
SPI	ISSI	is25lp	is25lp512m	512	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp080d	8	x1、x2、x4
SPI	ISSI	is25wp	is25wp016d	16	x1、x2、x4
SPI	ISSI	is25wp	is25wp032d	32	x1、x2、x4
SPI	ISSI	is25wp	is25wp064a	64	x1、x2、x4
SPI	ISSI	is25wp	is25wp128f	128	x1、x2、x4
SPI	ISSI	is25wp	is25wp256d	256	x1、x2、x4
SPI	ISSI	is25wp	is25wp512m	512	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Micron	mt25qu	mt25qu512	512	x1、x2、x4
SPI	Micron	mt25qu	mt25qu01g	1,024	x1、x2、x4
SPI	Micron	mt25qu	mt25qu02g	2,048	x1、x2、x4
SPI	Micron	n25q	n25q32-1.8v	32	x1、x2、x4
SPI	Micron	n25q	n25q64-1.8v	64	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxx0 [s25fl127s-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxx1	128	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl256sxxxxx0	256	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl256sxxxxx1	256	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl512s	512	x1、x2、x4
SPI	Macronix	mx25u	mx25u8033e	8	x1、x2、x4
SPI	Macronix	mx25u	mx25u1635f	16	x1、x2、x4

表 44: Virtex-7 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	Macronix	mx25u	mx25u3235f	32	x1、x2、x4
SPI	Macronix	mx25u	mx25u6435f	64	x1、x2、x4
SPI	Macronix	mx25u	mx25u12835f	128	x1、x2、x4
SPI	Macronix	mx25u	mx25u25645g [mx25u25635f-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Macronix	mx25u	mx25u51245g [mx66u51235f-spi-x1_x2_x4]	512	x1、x2、x4
SPI	Macronix	mx66u	mx66u1g45g	1,024	x1、x2、x4
SPI	Macronix	mx66u	mx66u2g45g	2,048	x1、x2、x4

Kintex UltraScale コンフィギュレーションのメモリ デバイス

次の表には、Kintex UltraScale デバイスのコンフィギュレーションにサポートされるフラッシュ デバイスで、Vivado® ツールで消去、ブランク チェック、プログラム、および検証可能なものを示します。

この付録にある表には、Vivado ソフトウェアで消去、ブランク チェック、プログラム、検証が可能な不揮発性メモリがザイリンクス ファミリごとにリストされています。ザイリンクスでは、これらのコンポーネントを含むエンドプロダクトの長期メンテナンスをサポートするため、コンポーネントが新しいデザインには適さなくなったとしても、できる限りこのリストにコンポーネントをリストするようにしています。



重要: 不揮発性メモリの市場は変化が激しいため、利用可能なデバイスおよびそのライフサイクルに関しては、ザイリンクスでは不揮発性メモリの製造業者に確認を取ることをお勧めしています。表に掲載されている特定デバイスの情報は、現在または今後の使用可能状態を確約するものではありません。

表 45: Kintex UltraScale デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	g18	28f128g18f	128	x16
BPI	Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	256	x16
BPI	Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	512	x16
BPI	Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	1,024	x16
BPI	Micron	m29ew	28f064m29ewb	64	x16、x8
BPI	Micron	m29ew	28f064m29ewh	64	x16、x8
BPI	Micron	m29ew	28f064m29ewl	64	x16、x8
BPI	Micron	m29ew	28f064m29ewt	64	x16、x8
BPI	Micron	m29ew	28f128m29ew	128	x16、x8

表 45: Kintex UltraScale デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	m29ew	28f256m29ew	256	x16、x8
BPI	Micron	m29ew	28f512m29ew	512	x16、x8
BPI	Micron	m29ew	28f00am29ew	1,024	x16、x8
BPI	Micron	m29ew	28f00bm29ew	2,048	x16、x8
BPI	Micron	m29w	m29w640gh	64	x16、x8
BPI	Micron	m29w	m29w640gl	64	x16、x8
BPI	Micron	m29w	m29w128gh	128	x16、x8
BPI	Micron	m29w	m29w128gl	128	x16、x8
BPI	Micron	m29w	m29w256gh	256	x16、x8
BPI	Micron	m29w	m29w256gl	256	x16、x8
BPI	Micron	mt28ew	mt28ew128a	128	x16、x8
BPI	Micron	mt28ew	mt28ew256a	256	x16、x8
BPI	Micron	mt28ew	mt28ew512a	512	x16、x8
BPI	Micron	mt28ew	mt28ew01ga	1,024	x16、x8
BPI	Micron	mt28fw	mt28fw02gb	2,048	x16
BPI	Micron	p30	28f640p30b	64	x16
BPI	Micron	p30	28f640p30t	64	x16
BPI	Micron	p30	28f128p30b	128	x16
BPI	Micron	p30	28f128p30t	128	x16
BPI	Micron	p30	28f256p30b	256	x16
BPI	Micron	p30	28f256p30t	256	x16
BPI	Micron	p30	28f512p30b	512	x16
BPI	Micron	p30	28f512p30e	512	x16
BPI	Micron	p30	28f512p30t	512	x16
BPI	Micron	p30	28f00ap30b	1,024	x16
BPI	Micron	p30	28f00ap30e	1,024	x16
BPI	Micron	p30	28f00ap30t	1,024	x16
BPI	Micron	p30	28f00bp30e	2,048	x16
BPI	Micron	p33	28f640p33b	64	x16
BPI	Micron	p33	28f640p33t	64	x16
BPI	Micron	p33	28f128p33b	128	x16
BPI	Micron	p33	28f128p33t	128	x16
BPI	Micron	p33	28f256p33b	256	x16
BPI	Micron	p33	28f256p33t	256	x16
BPI	Micron	p33	28f512p33b	512	x16
BPI	Micron	p33	28f512p33e	512	x16
BPI	Micron	p33	28f512p33t	512	x16
BPI	Micron	p33	28f00ap33b	1,024	x16

表 45: Kintex UltraScale デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	p33	28f00ap33e	1,024	x16
BPI	Micron	p33	28f00ap33t	1,024	x16
BPI	Spansion	s29glxxxp	s29gl128p	128	x16、x8
BPI	Spansion	s29glxxxp	s29gl256p	256	x16、x8
BPI	Spansion	s29glxxxp	s29gl512p	512	x16、x8
BPI	Spansion	s29glxxxp	s29gl01gp	1,024	x16、x8
BPI	Spansion	s29glxxxp	s70gl02gp	2,048	x16
BPI	Spansion	s29glxxxs	s29gl128s	128	x16
BPI	Spansion	s29glxxxs	s29gl256s	256	x16
BPI	Spansion	s29glxxxs	s29gl512s	512	x16
BPI	Spansion	s29glxxxs	s29gl01gs	1,024	x16
BPI	Spansion	s29glxxxs	s70gl02gs	2,048	x16
BPI	Spansion	s29glxxxt	s29gl512t	512	x16、x8
BPI	Spansion	s29glxxxt	s29gl01gt	1,024	x16、x8
BPI	Spansion	s29glxxxt	s70gl02gt	2,048	x16、x8
BPI	Macronix	mx29gl	mx29gl128f	128	x16、x8
SPI	ISSI	is25lp	is25lp080d	8	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp016d	16	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp032d	32	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp064a	64	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp128f	128	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp256d	256	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp512m	512	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp080d	8	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp016d	16	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp032d	32	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp064a	64	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp128f	128	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp256d	256	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp512m	512	x1、x2、x4、x8
SPI	Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4、n25q128-3.3v-spi-x1_x2_x4_x8]	128	x1、x2、x4、x8
SPI	Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4、n25q256-3.3v-spi-x1_x2_x4_x8]	256	x1、x2、x4、x8
SPI	Micron	mt25ql	mt25ql512	512	x1、x2、x4、x8
SPI	Micron	mt25ql	mt25ql01g	1,024	x1、x2、x4、x8
SPI	Micron	mt25ql	mt25ql02g	2,048	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4、n25q128-1.8v-spi-x1_x2_x4_x8]	128	x1、x2、x4、x8

表 45: Kintex UltraScale デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4, n25q256-1.8v-spi-x1_x2_x4_x8]	256	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu512	512	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu01g	1,024	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu02g	2,048	x1、x2、x4、x8
SPI	Micron	n25q	n25q32-1.8v	32	x1、x2、x4
SPI	Micron	n25q	n25q32-3.3v	32	x1、x2、x4
SPI	Micron	n25q	n25q64-1.8v	64	x1、x2、x4
SPI	Micron	n25q	n25q64-3.3v	64	x1、x2、x4
SPI	Spansion	s25fl1	s25fl116k	16	x1、x2、x4
SPI	Spansion	s25fl1	s25fl132k	32	x1、x2、x4
SPI	Spansion	s25fl1	s25fl164k	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl064l	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl128l	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxl	s25fl256l	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxp	s25fl032p	32	x1、x2、x4
SPI	Spansion	s25flxxxp	s25fl064p	64	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxx0 [s25fl127s-spi-x1_x2_x4, s25fl127s-spi-x1_x2_x4_x8]	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl128sxxxxx1	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl256sxxxxx0	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl256sxxxxx1	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl512s	512	x1、x2、x4、x8
SPI	Macronix	mx25l	mx25v8035f	8	x1、x2、x4、x8
SPI	Macronix	mx25l	mx25v1635f	16	x1、x2、x4、x8
SPI	Macronix	mx25l	mx25l3233f	32	x1、x2、x4、x8
SPI	Macronix	mx25l	mx25l6433f	64	x1、x2、x4、x8
SPI	Macronix	mx25l	mx25l12845g [mx25l12835f-spi-x1_x2_x4, mx25l12835f-spi-x1_x2_x4_x8]	128	x1、x2、x4、x8
SPI	Macronix	mx25l	mx25l25645g [mx25l25635f-spi-x1_x2_x4, mx25u25635f-spi-x1_x2_x4_x8]	256	x1、x2、x4、x8
SPI	Macronix	mx25l	mx25l51245g [mx66l51235f-spi-x1_x2_x4, mx66l51235f-spi-x1_x2_x4_x8]	512	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u8033e	8	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u1635f	16	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u3235f	32	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u6435f	64	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u12835f	128	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u25645g [mx25u25635f-spi-x1_x2_x4, mx25u25635f-spi-x1_x2_x4_x8]	256	x1、x2、x4、x8

表 45: Kintex UltraScale デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	Macronix	mx25u	mx25u51245g [mx66u51235f-spi-x1_x2_x4, mx66u51235f-spi-x1_x2_x4_x8]	512	x1、x2、x4、x8
SPI	Macronix	mx66l	mx66l1g45g	1,024	x1、x2、x4、x8
SPI	Macronix	mx66l	mx66l2g45g	2,048	x1、x2、x4、x8
SPI	Macronix	mx66u	mx66u1g45g	1,024	x1、x2、x4、x8
SPI	Macronix	mx66u	mx66u2g45g	2,048	x1、x2、x4、x8

Kintex UltraScale+ コンフィギュレーションのメモリ デバイス

次の表には、Kintex UltraScale+ デバイスのコンフィギュレーションにサポートされるフラッシュ デバイスで、Vivado® ツールで消去、ブランク チェック、プログラム、および検証可能なものを示します。

この付録にある表には、Vivado ソフトウェアで消去、ブランク チェック、プログラム、検証が可能な不揮発性メモリがザイリンクス ファミリごとにリストされています。ザイリンクスでは、これらのコンポーネントを含むエンドプロダクトの長期メンテナンスをサポートするため、コンポーネントが新しいデザインには適さなくなったとしても、できる限りこのリストにコンポーネントをリストするようにしています。



重要: 不揮発性メモリの市場は変化が激しいため、利用可能なデバイスおよびそのライフサイクルに関しては、ザイリンクスでは不揮発性メモリの製造業者に確認を取ることをお勧めしています。表に掲載されている特定デバイスの情報は、現在または今後の使用可能状態を確約するものではありません。

表 46: Kintex UltraScale+ デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	g18	28f128g18f	128	x16
BPI	Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	256	x16
BPI	Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	512	x16
BPI	Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	1,024	x16
BPI	Micron	m29ew	28f064m29ewb	64	x16、x8
BPI	Micron	m29ew	28f064m29ewh	64	x16、x8
BPI	Micron	m29ew	28f064m29ewl	64	x16、x8
BPI	Micron	m29ew	28f064m29ewt	64	x16、x8
BPI	Micron	m29ew	28f128m29ew	128	x16、x8
BPI	Micron	m29ew	28f256m29ew	256	x16、x8
BPI	Micron	m29ew	28f512m29ew	512	x16、x8
BPI	Micron	m29ew	28f00am29ew	1,024	x16、x8

表 46: Kintex UltraScale+ デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	m29ew	28f00bm29ew	2,048	x16、x8
BPI	Micron	m29w	m29w128gh	128	x16、x8
BPI	Micron	m29w	m29w128gl	128	x16、x8
BPI	Micron	m29w	m29w256gh	256	x16、x8
BPI	Micron	m29w	m29w256gl	256	x16、x8
BPI	Micron	mt28ew	mt28ew128a	128	x16、x8
BPI	Micron	mt28ew	mt28ew256a	256	x16、x8
BPI	Micron	mt28ew	mt28ew512a	512	x16、x8
BPI	Micron	mt28ew	mt28ew01ga	1,024	x16、x8
BPI	Micron	mt28fw	mt28fw02gb	2,048	x16
BPI	Micron	p30	28f640p30b	64	x16
BPI	Micron	p30	28f640p30t	64	x16
BPI	Micron	p30	28f128p30b	128	x16
BPI	Micron	p30	28f128p30t	128	x16
BPI	Micron	p30	28f256p30b	256	x16
BPI	Micron	p30	28f256p30t	256	x16
BPI	Micron	p30	28f512p30b	512	x16
BPI	Micron	p30	28f512p30e	512	x16
BPI	Micron	p30	28f512p30t	512	x16
BPI	Micron	p30	28f00ap30b	1,024	x16
BPI	Micron	p30	28f00ap30e	1,024	x16
BPI	Micron	p30	28f00ap30t	1,024	x16
BPI	Micron	p30	28f00bp30e	2,048	x16
BPI	Spansion	s29glxxxp	s29gl128p	128	x16、x8
BPI	Spansion	s29glxxxp	s29gl256p	256	x16、x8
BPI	Spansion	s29glxxxp	s29gl512p	512	x16、x8
BPI	Spansion	s29glxxxp	s29gl01gp	1,024	x16、x8
BPI	Spansion	s29glxxxs	s29gl128s	128	x16
BPI	Spansion	s29glxxxs	s29gl256s	256	x16
BPI	Spansion	s29glxxxs	s29gl512s	512	x16
BPI	Spansion	s29glxxxs	s29gl01gs	1,024	x16
BPI	Spansion	s29glxxxs	s70gl02gs	2,048	x16
BPI	Spansion	s29glxxxt	s29gl512t	512	x16、x8
BPI	Spansion	s29glxxxt	s29gl01gt	1,024	x16、x8
BPI	Spansion	s29glxxxt	s70gl02gt	2,048	x16、x8
BPI	Macronix	mx29gl	mx29gl128f	128	x16、x8
SPI	ISSI	is25wp	is25wp080d	8	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp016d	16	x1、x2、x4、x8

表 46: Kintex UltraScale+ デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	ISSI	is25wp	is25wp032d	32	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp064a	64	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp128f	128	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp256d	256	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp512m	512	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4、n25q128-1.8v-spi-x1_x2_x4_x8]	128	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4、n25q256-1.8v-spi-x1_x2_x4_x8]	256	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu512	512	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu01g	1,024	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu02g	2,048	x1、x2、x4、x8
SPI	Micron	n25q	n25q32-1.8v	32	x1、x2、x4
SPI	Micron	n25q	n25q64-1.8v	64	x1、x2、x4
SPI	Spansion	s25fl1	s25fl116k	16	x1、x2、x4
SPI	Spansion	s25fl1	s25fl132k	32	x1、x2、x4
SPI	Spansion	s25fl1	s25fl164k	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl064l	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl128l	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxl	s25fl256l	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxp	s25fl032p	32	x1、x2、x4
SPI	Spansion	s25flxxxp	s25fl064p	64	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxx0 [s25fl127s-spi-x1_x2_x4、s25fl127s-spi-x1_x2_x4_x8]	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl128sxxxxx1	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl256sxxxxx0	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl256sxxxxx1	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl512s	512	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u8033e	8	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u1635f	16	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u3235f	32	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u6435f	64	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u12835f	128	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u25645g [mx25u25635f-spi-x1_x2_x4、mx25u25635f-spi-x1_x2_x4_x8]	256	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u51245g [mx66u51235f-spi-x1_x2_x4、mx66u51235f-spi-x1_x2_x4_x8]	512	x1、x2、x4、x8
SPI	Macronix	mx66u	mx66u1g45g	1,024	x1、x2、x4、x8
SPI	Macronix	mx66u	mx66u2g45g	2,048	x1、x2、x4、x8

Virtex UltraScale コンフィギュレーション メモリ デバイス

次の表には、Virtex® UltraScale™ デバイスのコンフィギュレーションにサポートされるフラッシュ デバイスで、Vivado® ツールで消去、ブランク チェック、プログラム、および検証可能なものを示します。

この付録にある表には、Vivado ソフトウェアで消去、ブランク チェック、プログラム、検証が可能な不揮発性メモリがサイリンクス ファミリごとにリストされています。サイリンクスでは、これらのコンポーネントを含むエンドプロダクトの長期メンテナンスをサポートするため、コンポーネントが新しいデザインには適さなくなったとしても、できる限りこのリストにコンポーネントをリストするようにしています。

★ **重要:** 不揮発性メモリの市場は変化が激しいため、利用可能なデバイスおよびそのライフサイクルに関しては、サイリンクスでは不揮発性メモリの製造業者に確認を取ることをお勧めしています。表に掲載されている特定デバイスの情報は、現在または今後の使用可能状態を確約するものではありません。

表 47: Virtex UltraScale デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	g18	28f128g18f	128	x16
BPI	Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	256	x16
BPI	Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	512	x16
BPI	Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	1,024	x16
BPI	Micron	m29ew	28f064m29ewb	64	x16、x8
BPI	Micron	m29ew	28f064m29ewh	64	x16、x8
BPI	Micron	m29ew	28f064m29ewl	64	x16、x8
BPI	Micron	m29ew	28f064m29ewt	64	x16、x8
BPI	Micron	m29ew	28f128m29ew	128	x16、x8
BPI	Micron	m29ew	28f256m29ew	256	x16、x8
BPI	Micron	m29ew	28f512m29ew	512	x16、x8
BPI	Micron	m29ew	28f00am29ew	1,024	x16、x8
BPI	Micron	m29ew	28f00bm29ew	2,048	x16、x8
BPI	Micron	m29w	m29w128gh	128	x16、x8
BPI	Micron	m29w	m29w128gl	128	x16、x8
BPI	Micron	m29w	m29w256gh	256	x16、x8
BPI	Micron	m29w	m29w256gl	256	x16、x8
BPI	Micron	mt28ew	mt28ew128a	128	x16、x8
BPI	Micron	mt28ew	mt28ew256a	256	x16、x8
BPI	Micron	mt28ew	mt28ew512a	512	x16、x8
BPI	Micron	mt28ew	mt28ew01ga	1,024	x16、x8
BPI	Micron	mt28fw	mt28fw02gb	2,048	x16
BPI	Micron	p30	28f640p30b	64	x16
BPI	Micron	p30	28f640p30t	64	x16

表 47: Virtex UltraScale デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	p30	28f128p30b	128	x16
BPI	Micron	p30	28f128p30t	128	x16
BPI	Micron	p30	28f256p30b	256	x16
BPI	Micron	p30	28f256p30t	256	x16
BPI	Micron	p30	28f512p30b	512	x16
BPI	Micron	p30	28f512p30e	512	x16
BPI	Micron	p30	28f512p30t	512	x16
BPI	Micron	p30	28f00ap30b	1,024	x16
BPI	Micron	p30	28f00ap30e	1,024	x16
BPI	Micron	p30	28f00ap30t	1,024	x16
BPI	Micron	p30	28f00bp30e	2,048	x16
BPI	Spansion	s29glxxp	s29gl128p	128	x16、x8
BPI	Spansion	s29glxxp	s29gl256p	256	x16、x8
BPI	Spansion	s29glxxp	s29gl512p	512	x16、x8
BPI	Spansion	s29glxxp	s29gl01gp	1,024	x16、x8
BPI	Spansion	s29glxxs	s29gl128s	128	x16
BPI	Spansion	s29glxxs	s29gl256s	256	x16
BPI	Spansion	s29glxxs	s29gl512s	512	x16
BPI	Spansion	s29glxxs	s29gl01gs	1,024	x16
BPI	Spansion	s29glxxs	s70gl02gs	2,048	x16
BPI	Spansion	s29glxxt	s29gl512t	512	x16、x8
BPI	Spansion	s29glxxt	s29gl01gt	1,024	x16、x8
BPI	Spansion	s29glxxt	s70gl02gt	2,048	x16、x8
BPI	Macronix	mx29gl	mx29gl128f	128	x16、x8
SPI	ISSI	is25lp	is25lp080d	8	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp016d	16	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp032d	32	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp064a	64	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp128f	128	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp256d	256	x1、x2、x4、x8
SPI	ISSI	is25lp	is25lp512m	512	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp080d	8	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp016d	16	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp032d	32	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp064a	64	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp128f	128	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp256d	256	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp512m	512	x1、x2、x4、x8

表 47: Virtex UltraScale デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	Micron	mt25ql	mt25ql128 [n25q128-3.3v-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Micron	mt25ql	mt25ql256 [n25q256-3.3v-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Micron	mt25ql	mt25ql512	512	x1、x2、x4
SPI	Micron	mt25ql	mt25ql01g	1,024	x1、x2、x4
SPI	Micron	mt25ql	mt25ql02g	2,048	x1、x2、x4
SPI	Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4、n25q128-1.8v-spi-x1_x2_x4_x8]	128	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4、n25q256-1.8v-spi-x1_x2_x4_x8]	256	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu512	512	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu01g	1,024	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu02g	2,048	x1、x2、x4、x8
SPI	Micron	n25q	n25q32-1.8v	32	x1、x2、x4
SPI	Micron	n25q	n25q32-3.3v	32	x1、x2、x4
SPI	Micron	n25q	n25q64-1.8v	64	x1、x2、x4
SPI	Micron	n25q	n25q64-3.3v	64	x1、x2、x4
SPI	Spansion	s25fl1	s25fl116k	16	x1、x2、x4
SPI	Spansion	s25fl1	s25fl132k	32	x1、x2、x4
SPI	Spansion	s25fl1	s25fl164k	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl064l	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl128l	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxl	s25fl256l	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxp	s25fl032p	32	x1、x2、x4
SPI	Spansion	s25flxxxp	s25fl064p	64	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxx0 [s25fl127s-spi-x1_x2_x4、s25fl127s-spi-x1_x2_x4_x8]	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl128sxxxxx1	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl256sxxxxx0	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl256sxxxxx1	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl512s	512	x1、x2、x4、x8
SPI	Macronix	mx25l	mx25v8035f	8	x1、x2、x4
SPI	Macronix	mx25l	mx25v1635f	16	x1、x2、x4
SPI	Macronix	mx25l	mx25l3233f	32	x1、x2、x4
SPI	Macronix	mx25l	mx25l6433f	64	x1、x2、x4
SPI	Macronix	mx25l	mx25l12845g [mx25l12835f-spi-x1_x2_x4]	128	x1、x2、x4
SPI	Macronix	mx25l	mx25l25645g [mx25l25635f-spi-x1_x2_x4]	256	x1、x2、x4
SPI	Macronix	mx25l	mx25l51245g [mx66l51235f-spi-x1_x2_x4]	512	x1、x2、x4
SPI	Macronix	mx25u	mx25u8033e	8	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u1635f	16	x1、x2、x4、x8

表 47: Virtex UltraScale デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	Macronix	mx25u	mx25u3235f	32	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u6435f	64	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u12835f	128	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u25645g [mx25u25635f-spi-x1_x2_x4, mx25u25635f-spi-x1_x2_x4_x8]	256	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u51245g [mx66u51235f-spi-x1_x2_x4, mx66u51235f-spi-x1_x2_x4_x8]	512	x1、x2、x4、x8
SPI	Macronix	mx66l	mx66l1g45g	1,024	x1、x2、x4
SPI	Macronix	mx66l	mx66l2g45g	2,048	x1、x2、x4
SPI	Macronix	mx66u	mx66u1g45g	1,024	x1、x2、x4、x8
SPI	Macronix	mx66u	mx66u2g45g	2,048	x1、x2、x4、x8

Virtex UltraScale+ コンフィギュレーション メモリ デバイス

次の表には、Virtex® UltraScale+™ デバイスのコンフィギュレーションにサポートされるフラッシュ デバイスで、Vivado® ツールで消去、ブランク チェック、プログラム、および検証可能なものを示します。

この付録にある表には、Vivado ソフトウェアで消去、ブランク チェック、プログラム、検証が可能な不揮発性メモリがザイリンクス ファミリごとにリストされています。ザイリンクスでは、これらのコンポーネントを含むエンドプロダクトの長期メンテナンスをサポートするため、コンポーネントが新しいデザインには適さなくなったとしても、できる限りこのリストにコンポーネントをリストするようにしています。



重要: 不揮発性メモリの市場は変化が激しいため、利用可能なデバイスおよびそのライフサイクルに関しては、ザイリンクスでは不揮発性メモリの製造業者に確認を取ることをお勧めしています。表に掲載されている特定デバイスの情報は、現在または今後の使用可能状態を確約するものではありません。

表 48: Virtex UltraScale+ デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	g18	28f128g18f	128	x16
BPI	Micron	g18	mt28gu256aax1e [28f256g18f-bpi-x16]	256	x16
BPI	Micron	g18	mt28gu512aax1e [28f512g18f-bpi-x16]	512	x16
BPI	Micron	g18	mt28gu01gaax1e [28f00ag18f-bpi-x16]	1,024	x16
BPI	Micron	m29ew	28f064m29ewb	64	x16、x8
BPI	Micron	m29ew	28f064m29ewh	64	x16、x8
BPI	Micron	m29ew	28f064m29ewl	64	x16、x8
BPI	Micron	m29ew	28f064m29ewt	64	x16、x8

表 48: Virtex UltraScale+ デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Micron	m29ew	28f128m29ew	128	x16、x8
BPI	Micron	m29ew	28f256m29ew	256	x16、x8
BPI	Micron	m29ew	28f512m29ew	512	x16、x8
BPI	Micron	m29ew	28f00am29ew	1,024	x16、x8
BPI	Micron	m29ew	28f00bm29ew	2,048	x16、x8
BPI	Micron	m29w	m29w128gh	128	x16、x8
BPI	Micron	m29w	m29w128gl	128	x16、x8
BPI	Micron	m29w	m29w256gh	256	x16、x8
BPI	Micron	m29w	m29w256gl	256	x16、x8
BPI	Micron	mt28ew	mt28ew128a	128	x16、x8
BPI	Micron	mt28ew	mt28ew256a	256	x16、x8
BPI	Micron	mt28ew	mt28ew512a	512	x16、x8
BPI	Micron	mt28ew	mt28ew01ga	1,024	x16、x8
BPI	Micron	mt28fw	mt28fw02gb	2,048	x16
BPI	Micron	p30	28f640p30b	64	x16
BPI	Micron	p30	28f640p30t	64	x16
BPI	Micron	p30	28f128p30b	128	x16
BPI	Micron	p30	28f128p30t	128	x16
BPI	Micron	p30	28f256p30b	256	x16
BPI	Micron	p30	28f256p30t	256	x16
BPI	Micron	p30	28f512p30b	512	x16
BPI	Micron	p30	28f512p30e	512	x16
BPI	Micron	p30	28f512p30t	512	x16
BPI	Micron	p30	28f00ap30b	1,024	x16
BPI	Micron	p30	28f00ap30e	1,024	x16
BPI	Micron	p30	28f00ap30t	1,024	x16
BPI	Micron	p30	28f00bp30e	2,048	x16
BPI	Spansion	s29glxxp	s29gl128p	128	x16、x8
BPI	Spansion	s29glxxp	s29gl256p	256	x16、x8
BPI	Spansion	s29glxxp	s29gl512p	512	x16、x8
BPI	Spansion	s29glxxp	s29gl01gp	1,024	x16、x8
BPI	Spansion	s29glxxs	s29gl128s	128	x16
BPI	Spansion	s29glxxs	s29gl256s	256	x16
BPI	Spansion	s29glxxs	s29gl512s	512	x16
BPI	Spansion	s29glxxs	s29gl01gs	1,024	x16
BPI	Spansion	s29glxxs	s70gl02gs	2,048	x16
BPI	Spansion	s29glxxxt	s29gl512t	512	x16、x8
BPI	Spansion	s29glxxxt	s29gl01gt	1,024	x16、x8

表 48: Virtex UltraScale+ デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
BPI	Spansion	s29glxxt	s70gl02gt	2,048	x16、x8
BPI	Macronix	mx29gl	mx29gl128f	128	x16、x8
SPI	ISSI	is25lp	is25lp512m	512	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp080d	8	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp016d	16	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp032d	32	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp064a	64	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp128f	128	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp256d	256	x1、x2、x4、x8
SPI	ISSI	is25wp	is25wp512m	512	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu128 [n25q128-1.8v-spi-x1_x2_x4、n25q128-1.8v-spi-x1_x2_x4_x8]	128	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu256 [n25q256-1.8v-spi-x1_x2_x4、n25q256-1.8v-spi-x1_x2_x4_x8]	256	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu512	512	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu01g	1,024	x1、x2、x4、x8
SPI	Micron	mt25qu	mt25qu02g	2,048	x1、x2、x4、x8
SPI	Micron	n25q	n25q32-1.8v	32	x1、x2、x4
SPI	Micron	n25q	n25q64-1.8v	64	x1、x2、x4
SPI	Spansion	s25fl1	s25fl116k	16	x1、x2、x4
SPI	Spansion	s25fl1	s25fl132k	32	x1、x2、x4
SPI	Spansion	s25fl1	s25fl164k	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl064l	64	x1、x2、x4
SPI	Spansion	s25flxxxl	s25fl128l	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxl	s25fl256l	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxp	s25fl032p	32	x1、x2、x4
SPI	Spansion	s25flxxxp	s25fl064p	64	x1、x2、x4
SPI	Spansion	s25flxxxs	s25fl128sxxxxx0 [s25fl127s-spi-x1_x2_x4、s25fl127s-spi-x1_x2_x4_x8]	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl128sxxxxx1	128	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl256sxxxxx0	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl256sxxxxx1	256	x1、x2、x4、x8
SPI	Spansion	s25flxxxs	s25fl512s	512	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u8033e	8	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u1635f	16	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u3235f	32	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u6435f	64	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u12835f	128	x1、x2、x4、x8

表 48: Virtex UltraScale+ デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
SPI	Macronix	mx25u	mx25u25645g [mx25u25635f-spi-x1_x2_x4, mx25u25635f-spi-x1_x2_x4_x8]	256	x1、x2、x4、x8
SPI	Macronix	mx25u	mx25u51245g [mx66u51235f-spi-x1_x2_x4, mx66u51235f-spi-x1_x2_x4_x8]	512	x1、x2、x4、x8
SPI	Macronix	mx66u	mx66u1g45g	1,024	x1、x2、x4、x8
SPI	Macronix	mx66u	mx66u2g45g	2,048	x1、x2、x4、x8

Zynq-7000 コンフィギュレーション メモリ デバイス

次の表には、Zynq-7000 デバイスのコンフィギュレーションにサポートされるフラッシュ デバイスで、Vivado® ツールで消去、ブランク チェック、プログラム、および検証可能なものを示します。

この付録にある表には、Vivado ソフトウェアで消去、ブランク チェック、プログラム、検証が可能な不揮発性メモリがザイリンクス ファミリごとにリストされています。ザイリンクスでは、これらのコンポーネントを含むエンドプロダクトの長期メンテナンスをサポートするため、コンポーネントが新しいデザインには適さなくなったとしても、できる限りこのリストにコンポーネントをリストするようにしています。



重要: 不揮発性メモリの市場は変化が激しいため、利用可能なデバイスおよびそのライフサイクルに関しては、ザイリンクスでは不揮発性メモリの製造業者に確認を取ることをお勧めしています。表に掲載されている特定デバイスの情報は、現在または今後の使用可能状態を確約するものではありません。

注記: コンフィギュレーション メモリ デバイス プログラムを構築するために使用される U-Boot タグは、次のとおりです。

インターフェイス	U-Boot タグ
qspi	xilinx-v2015.2.01
nor	xilinx-14.3-build1
nand	xilinx-v2015.2.01

表 49: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
NAND	Micron	mt29f	mt29f2g08ab	2,048	x8
NAND	Micron	mt29f	mt29f2g16ab	2,048	x16
NAND	Spansion	s34ml	s34ml01g1	1,024	x16、x8
NAND	Spansion	s34ml	s34ml02g1	2,048	x16、x8
NOR	Micron	m29ew	28f032m29ewt	32	x8
NOR	Micron	m29ew	28f064m29ewt	64	x8
NOR	Micron	m29ew	28f128m29ewh	128	x8
NOR	Micron	m29ew	28f256m29ewh	256	x8

表 49: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
NOR	Micron	m29ew	28f512m29ewh	512	x8
QSPI	ISSI	is25lp	is25lp080d	8	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	ISSI	is25lp	is25lp016d	16	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	ISSI	is25lp	is25lp032d	32	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	ISSI	is25lp	is25lp064a	64	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel

表 49: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	ISSI	is25lp	is25lp128f	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	ISSI	is25lp	is25lp256d	256	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	ISSI	is25lp	is25lp512m	512	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	ISSI	is25wp	is25wp080d	8	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	ISSI	is25wp	is25wp016d	16	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel

表 49: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	ISSI	is25wp	is25wp032d	32	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	ISSI	is25wp	is25wp064a	64	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	ISSI	is25wp	is25wp128f	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	ISSI	is25wp	is25wp256d	256	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	ISSI	is25wp	is25wp512m	512	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel

表 49: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Winbond	w25q	w25q128fv [w25q128jv-qspi-x1-dual_stacked, w25q128jv-qspi-x1-single, w25q128jv-qspi-x2-dual_stacked, w25q128jv-qspi-x2-single, w25q128jv-qspi-x4-dual_stacked, w25q128jv-qspi-x4-single, w25q128jv-qspi-x8-dual_parallel]	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Winbond	w25q	w25q128fw	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql128 [n25q128-3.3v-qspi-x1-dual_stacked, n25q128-3.3v-qspi-x1-single, n25q128-3.3v-qspi-x2-dual_stacked, n25q128-3.3v-qspi-x2-single, n25q128-3.3v-qspi-x4-dual_stacked, n25q128-3.3v-qspi-x4-single, n25q128-3.3v-qspi-x8-dual_parallel]	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql256 [n25q256-3.3v-qspi-x1-dual_stacked, n25q256-3.3v-qspi-x1-single, n25q256-3.3v-qspi-x2-dual_stacked, n25q256-3.3v-qspi-x2-single, n25q256-3.3v-qspi-x4-dual_stacked, n25q256-3.3v-qspi-x4-single, n25q256-3.3v-qspi-x8-dual_parallel]	256	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql512 [n25q512-3.3v-qspi-x1-dual_stacked, n25q512-3.3v-qspi-x1-single, n25q512-3.3v-qspi-x2-dual_stacked, n25q512-3.3v-qspi-x2-single, n25q512-3.3v-qspi-x4-dual_stacked, n25q512-3.3v-qspi-x4-single, n25q512-3.3v-qspi-x8-dual_parallel]	512	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel

表 49: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Micron	mt25ql	mt25ql01g [n25q00a-3.3v-qspi-x1-dual_stacked, n25q00a-3.3v-qspi-x1-single, n25q00a-3.3v-qspi-x2-dual_stacked, n25q00a-3.3v-qspi-x2-single, n25q00a-3.3v-qspi-x4-dual_stacked, n25q00a-3.3v-qspi-x4-single, n25q00a-3.3v-qspi-x8-dual_parallel]	1,024	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql02g	2,048	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu128 [n25q128-1.8v-qspi-x1-dual_stacked, n25q128-1.8v-qspi-x1-single, n25q128-1.8v-qspi-x2-dual_stacked, n25q128-1.8v-qspi-x2-single, n25q128-1.8v-qspi-x4-dual_stacked, n25q128-1.8v-qspi-x4-single, n25q128-1.8v-qspi-x8-dual_parallel]	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu256 [n25q256-1.8v-qspi-x1-dual_stacked, n25q256-1.8v-qspi-x1-single, n25q256-1.8v-qspi-x2-dual_stacked, n25q256-1.8v-qspi-x2-single, n25q256-1.8v-qspi-x4-dual_stacked, n25q256-1.8v-qspi-x4-single, n25q256-1.8v-qspi-x8-dual_parallel]	256	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu512 [n25q512-1.8v-qspi-x1-dual_stacked, n25q512-1.8v-qspi-x1-single, n25q512-1.8v-qspi-x2-dual_stacked, n25q512-1.8v-qspi-x2-single, n25q512-1.8v-qspi-x4-dual_stacked, n25q512-1.8v-qspi-x4-single, n25q512-1.8v-qspi-x8-dual_parallel]	512	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel

表 49: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Micron	mt25qu	mt25qu01g [n25q00a-1.8v-qspi-x1-dual_stacked, n25q00a-1.8v-qspi-x1-single, n25q00a-1.8v-qspi-x2-dual_stacked, n25q00a-1.8v-qspi-x2-single, n25q00a-1.8v-qspi-x4-dual_stacked, n25q00a-1.8v-qspi-x4-single, n25q00a-1.8v-qspi-x8-dual_parallel]	1,024	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu02g	2,048	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Micron	n25q	n25q64-1.8v	64	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Micron	n25q	n25q64-3.3v	64	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Spansion	s25flxxxl	s25fl064l	64	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel

表 49: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Spansion	s25flxxxp	s25fl129p	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl128s-1.8v [s25fl127s-1.8v-qspi-x4-dual_stacked, s25fl127s-1.8v-qspi-x4-single, s25fl127s-1.8v-qspi-x8-dual_parallel]	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl128s-3.3v [s25fl127s-3.3v-qspi-x4-dual_stacked, s25fl127s-3.3v-qspi-x4-single, s25fl127s-3.3v-qspi-x8-dual_parallel]	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl256s-1.8v	256	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl256s-3.3v	256	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel

表 49: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Spansion	s25flxxxs	s25fl512s-1.8v	512	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl512s-3.3v	512	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Spansion	s70flxxxp	s70fl01gs_00	1,024	x4-dual_stacked
QSPI	Macronix	mx25l	mx25l12835f [mx25l12833f-qspi-x1-dual_stacked, mx25l12833f-qspi-x1-single, mx25l12833f-qspi-x2-dual_stacked, mx25l12833f-qspi-x2-single, mx25l12833f-qspi-x4-dual_stacked, mx25l12833f-qspi-x4-single, mx25l12833f-qspi-x8-dual_parallel]	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12845g	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel

表 49: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Macronix	mx25l	mx25l25635f [mx25l25645g-qspi-x1-dual_stacked, mx25l25645g-qspi-x1-single, mx25l25645g-qspi-x2-dual_stacked, mx25l25645g-qspi-x2-single, mx25l25645g-qspi-x4-dual_stacked, mx25l25645g-qspi-x4-single, mx25l25645g-qspi-x8-dual_parallel]	256	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Macronix	mx25l	mx25l51245g	512	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Macronix	mx25u	mx25u12835f	128	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25635f [mx25u25645g-qspi-x1-dual_stacked, mx25u25645g-qspi-x1-single, mx25u25645g-qspi-x2-dual_stacked, mx25u25645g-qspi-x2-single, mx25u25645g-qspi-x4-dual_stacked, mx25u25645g-qspi-x4-single, mx25u25645g-qspi-x8-dual_parallel]	256	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Macronix	mx25u	mx25u51245g	512	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel

表 49: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Macronix	mx66l	mx66l51235f	512	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Macronix	mx66l	mx66l1g45g	1,024	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Macronix	mx66l	mx66l2g45g	2,048	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel
QSPI	Macronix	mx66u	mx66u2g45g	2,048	x1-dual_stacked、 x1-single、 x2-dual_stacked、 x2-single、 x4-dual_stacked、 x4-single、 x8-dual_parallel

表 50: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
NAND	Micron	mt29f	mt29f2g08ab	2,048	x8
NAND	Micron	mt29f	mt29f2g16ab	2,048	x16
NAND	Spansion	s34ml	s34ml01g1	1,024	x16、 x8

表 50: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
NAND	Spansion	s34ml	s34ml02g1	2,048	x16、x8
NOR			nor	0	なし
NOR	Micron	m29ew	28f032m29ewt	32	x8
NOR	Micron	m29ew	28f064m29ewt	64	x8
NOR	Micron	m29ew	28f128m29ewh	128	x8
NOR	Micron	m29ew	28f256m29ewh	256	x8
NOR	Micron	m29ew	28f512m29ewh	512	x8
QSPI	ISSI	is25lp	is25lp080d	8	x1-dual_stacked、 x1-single、x2-dual_stacked、 x2-single、x4-dual_stacked、 x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp016d	16	x1-dual_stacked、 x1-single、x2-dual_stacked、 x2-single、x4-dual_stacked、 x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp032d	32	x1-dual_stacked、 x1-single、x2-dual_stacked、 x2-single、x4-dual_stacked、 x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp064a	64	x1-dual_stacked、 x1-single、x2-dual_stacked、 x2-single、x4-dual_stacked、 x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp128f	128	x1-dual_stacked、 x1-single、x2-dual_stacked、 x2-single、x4-dual_stacked、 x4-single、x8-dual_parallel

表 50: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	ISSI	is25lp	is25lp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp080d	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp016d	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp032d	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp064a	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp128f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 50: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	ISSI	is25wp	is25wp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q128fv [w25q128jv-qspi-x1-dual_stacked、w25q128jv-qspi-x1-single、w25q128jv-qspi-x2-dual_stacked、w25q128jv-qspi-x2-single、w25q128jv-qspi-x4-dual_stacked、w25q128jv-qspi-x4-single、w25q128jv-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Winbond	w25q	w25q128fw	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql128 [n25q128-3.3v-qspi-x1-dual_stacked、n25q128-3.3v-qspi-x1-single、n25q128-3.3v-qspi-x2-dual_stacked、n25q128-3.3v-qspi-x2-single、n25q128-3.3v-qspi-x4-dual_stacked、n25q128-3.3v-qspi-x4-single、n25q128-3.3v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql256 [n25q256-3.3v-qspi-x1-dual_stacked、n25q256-3.3v-qspi-x1-single、n25q256-3.3v-qspi-x2-dual_stacked、n25q256-3.3v-qspi-x2-single、n25q256-3.3v-qspi-x4-dual_stacked、n25q256-3.3v-qspi-x4-single、n25q256-3.3v-qspi-x8-dual_parallel]	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql512 [n25q512-3.3v-qspi-x1-dual_stacked、n25q512-3.3v-qspi-x1-single、n25q512-3.3v-qspi-x2-dual_stacked、n25q512-3.3v-qspi-x2-single、n25q512-3.3v-qspi-x4-dual_stacked、n25q512-3.3v-qspi-x4-single、n25q512-3.3v-qspi-x8-dual_parallel]	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 50: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Micron	mt25ql	mt25ql01g [n25q00a-3.3v-qspi-x1-dual_stacked、n25q00a-3.3v-qspi-x1-single、n25q00a-3.3v-qspi-x2-dual_stacked、n25q00a-3.3v-qspi-x2-single、n25q00a-3.3v-qspi-x4-dual_stacked、n25q00a-3.3v-qspi-x4-single、n25q00a-3.3v-qspi-x8-dual_parallel]	1,024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql02g	2,048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu128 [n25q128-1.8v-qspi-x1-dual_stacked、n25q128-1.8v-qspi-x1-single、n25q128-1.8v-qspi-x2-dual_stacked、n25q128-1.8v-qspi-x2-single、n25q128-1.8v-qspi-x4-dual_stacked、n25q128-1.8v-qspi-x4-single、n25q128-1.8v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu256 [n25q256-1.8v-qspi-x1-dual_stacked、n25q256-1.8v-qspi-x1-single、n25q256-1.8v-qspi-x2-dual_stacked、n25q256-1.8v-qspi-x2-single、n25q256-1.8v-qspi-x4-dual_stacked、n25q256-1.8v-qspi-x4-single、n25q256-1.8v-qspi-x8-dual_parallel]	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu512 [n25q512-1.8v-qspi-x1-dual_stacked、n25q512-1.8v-qspi-x1-single、n25q512-1.8v-qspi-x2-dual_stacked、n25q512-1.8v-qspi-x2-single、n25q512-1.8v-qspi-x4-dual_stacked、n25q512-1.8v-qspi-x4-single、n25q512-1.8v-qspi-x8-dual_parallel]	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu01g [n25q00a-1.8v-qspi-x1-dual_stacked、n25q00a-1.8v-qspi-x1-single、n25q00a-1.8v-qspi-x2-dual_stacked、n25q00a-1.8v-qspi-x2-single、n25q00a-1.8v-qspi-x4-dual_stacked、n25q00a-1.8v-qspi-x4-single、n25q00a-1.8v-qspi-x8-dual_parallel]	1,024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu02g	2,048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 50: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Micron	n25q	n25q64-1.8v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	n25q	n25q64-3.3v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxl	s25fl064l	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxp	s25fl129p	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl128s-1.8v [s25fl127s-1.8v-qspi-x4-dual_stacked、s25fl127s-1.8v-qspi-x4-single、s25fl127s-1.8v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl128s-3.3v [s25fl127s-3.3v-qspi-x4-dual_stacked、s25fl127s-3.3v-qspi-x4-single、s25fl127s-3.3v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl256s-1.8v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 50: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Spansion	s25flxxxs	s25fl256s-3.3v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl512s-1.8v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl512s-3.3v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s70flxxxp	s70fl01gs_00	1,024	x4-dual_stacked
QSPI	Macronix	mx25l	mx25l12835f [mx25l12833f-qspi-x1-dual_stacked、mx25l12833f-qspi-x1-single、mx25l12833f-qspi-x2-dual_stacked、mx25l12833f-qspi-x2-single、mx25l12833f-qspi-x4-dual_stacked、mx25l12833f-qspi-x4-single、mx25l12833f-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l12845g	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25635f [mx25l25645g-qspi-x1-dual_stacked、mx25l25645g-qspi-x1-single、mx25l25645g-qspi-x2-dual_stacked、mx25l25645g-qspi-x2-single、mx25l25645g-qspi-x4-dual_stacked、mx25l25645g-qspi-x4-single、mx25l25645g-qspi-x8-dual_parallel]	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l51245g	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 50: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Macronix	mx25u	mx25u12835f [mx25u12832f-qspi-x1-dual_stacked, mx25u12832f-qspi-x1-single, mx25u12832f-qspi-x2-dual_stacked, mx25u12832f-qspi-x2-single, mx25u12832f-qspi-x4-dual_stacked, mx25u12832f-qspi-x4-single, mx25u12832f-qspi-x8-dual_parallel]	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25635f [mx25u25645g-qspi-x1-dual_stacked, mx25u25645g-qspi-x1-single, mx25u25645g-qspi-x2-dual_stacked, mx25u25645g-qspi-x2-single, mx25u25645g-qspi-x4-dual_stacked, mx25u25645g-qspi-x4-single, mx25u25645g-qspi-x8-dual_parallel]	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx25u	mx25u51245g	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx66l	mx66l51235f	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx66l	mx66l1g45g	1,024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx66l	mx66l2g45g	2,048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx66u	mx66u1g45g	1,024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel

表 50: Zynq-7000 デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Macronix	mx66u	mx66u2g45g	2,048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

Zynq UltraScale+ MPSoC コンフィギュレーション メモリ デバイス

次の表には、Zynq UltraScale+ MPSoC デバイスのコンフィギュレーションにサポートされるフラッシュ デバイスで、Vivado® ツールで消去、ブランク チェック、プログラム、および検証可能なものを示します。

この付録にある表には、Vivado ソフトウェアで消去、ブランク チェック、プログラム、検証が可能な不揮発性メモリがザイリンクス ファミリごとにリストされています。ザイリンクスでは、これらのコンポーネントを含むエンドプロダクトの長期メンテナンスをサポートするため、コンポーネントが新しいデザインには適さなくなったとしても、できる限りこのリストにコンポーネントをリストするようにしています。



重要: 不揮発性メモリの市場は変化が激しいため、利用可能なデバイスおよびそのライフサイクルに関しては、ザイリンクスでは不揮発性メモリの製造業者に確認を取ることをお勧めしています。表に掲載されている特定デバイスの情報は、現在または今後の使用可能状態を確約するものではありません。

表 51: Zynq UltraScale+ MPSoC デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス

インターフェイス	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
EMMC		jedec4.51-4gb	32,768	
EMMC		jedec4.51-8gb	65,536	
EMMC		jedec4.51-16gb	131,072	
EMMC		jedec4.51-32gb	262,144	
EMMC		jedec4.51	524,288	
EMMC		jedec4.51-64gb	524,288	
EMMC	mtfc	mtfc8gakajcn-4m	65,536	
NAND	mt29f	mt29f2g08ab	2,048	x8-dual、x8-single
NAND	mt29f	mt29f8g08ab	8,192	x8-dual、x8-single
NAND	mt29f	mt29f16g08ab	16,384	x8-dual、x8-single
NAND	mt29f	mt29f32g08ae	32,768	x8-dual、x8-single
NAND	mt29f	mt29f64g08ae	65,536	x8-dual、x8-single
NAND	s34ml	s34ml01g1	1,024	x8-dual、x8-single

表 51: Zynq UltraScale+ MPSoC デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
NAND	s34ml	s34ml02g1	2,048	x8-dual、x8-single
QSPI	is25lp	is25lp080d	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	is25lp	is25lp016d	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	is25lp	is25lp032d	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	is25lp	is25lp128f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	is25lp	is25lp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	is25lp	is25lp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	is25wp	is25wp080d	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	is25wp	is25wp016d	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 51: Zynq UltraScale+ MPSoC デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	is25wp	is25wp032d	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	is25wp	is25wp064a	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	is25wp	is25wp128f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	is25wp	is25wp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	is25wp	is25wp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mt25ql	mt25ql128 [n25q128-3.3v-qspi-x1-dual_stacked、n25q128-3.3v-qspi-x1-single、n25q128-3.3v-qspi-x2-dual_stacked、n25q128-3.3v-qspi-x2-single、n25q128-3.3v-qspi-x4-dual_stacked、n25q128-3.3v-qspi-x4-single、n25q128-3.3v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mt25ql	mt25ql256 [n25q256-3.3v-qspi-x1-dual_stacked、n25q256-3.3v-qspi-x1-single、n25q256-3.3v-qspi-x2-dual_stacked、n25q256-3.3v-qspi-x2-single、n25q256-3.3v-qspi-x4-dual_stacked、n25q256-3.3v-qspi-x4-single、n25q256-3.3v-qspi-x8-dual_parallel]	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mt25ql	mt25ql512 [n25q512-3.3v-qspi-x1-dual_stacked、n25q512-3.3v-qspi-x1-single、n25q512-3.3v-qspi-x2-dual_stacked、n25q512-3.3v-qspi-x2-single、n25q512-3.3v-qspi-x4-dual_stacked、n25q512-3.3v-qspi-x4-single、n25q512-3.3v-qspi-x8-dual_parallel]	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 51: Zynq UltraScale+ MPSoC デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	mt25ql	mt25ql01g [n25q00a-3.3v-qspi-x1-dual_stacked、n25q00a-3.3v-qspi-x1-single、n25q00a-3.3v-qspi-x2-dual_stacked、n25q00a-3.3v-qspi-x2-single、n25q00a-3.3v-qspi-x4-dual_stacked、n25q00a-3.3v-qspi-x4-single、n25q00a-3.3v-qspi-x8-dual_parallel]	1,024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mt25ql	mt25ql02g	2,048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mt25qu	mt25qu128 [n25q128-1.8v-qspi-x1-dual_stacked、n25q128-1.8v-qspi-x1-single、n25q128-1.8v-qspi-x2-dual_stacked、n25q128-1.8v-qspi-x2-single、n25q128-1.8v-qspi-x4-dual_stacked、n25q128-1.8v-qspi-x4-single、n25q128-1.8v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mt25qu	mt25qu256 [n25q256-1.8v-qspi-x1-dual_stacked、n25q256-1.8v-qspi-x1-single、n25q256-1.8v-qspi-x2-dual_stacked、n25q256-1.8v-qspi-x2-single、n25q256-1.8v-qspi-x4-dual_stacked、n25q256-1.8v-qspi-x4-single、n25q256-1.8v-qspi-x8-dual_parallel]	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mt25qu	mt25qu512 [n25q512-1.8v-qspi-x1-dual_stacked、n25q512-1.8v-qspi-x1-single、n25q512-1.8v-qspi-x2-dual_stacked、n25q512-1.8v-qspi-x2-single、n25q512-1.8v-qspi-x4-dual_stacked、n25q512-1.8v-qspi-x4-single、n25q512-1.8v-qspi-x8-dual_parallel]	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mt25qu	mt25qu01g [n25q00a-1.8v-qspi-x1-dual_stacked、n25q00a-1.8v-qspi-x1-single、n25q00a-1.8v-qspi-x2-dual_stacked、n25q00a-1.8v-qspi-x2-single、n25q00a-1.8v-qspi-x4-dual_stacked、n25q00a-1.8v-qspi-x4-single、n25q00a-1.8v-qspi-x8-dual_parallel]	1,024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mt25qu	mt25qu02g	2,048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	n25q	n25q64-1.8v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 51: Zynq UltraScale+ MPSoC デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	n25q	n25q64-3.3v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	s25flxxxl	s25fl256l	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	s25flxxxp	s25fl129p	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	s25flxxxs	s25fl128s-1.8v [s25fl127s-1.8v-qspi-x4-dual_stacked、s25fl127s-1.8v-qspi-x4-single、s25fl127s-1.8v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	s25flxxxs	s25fl128s-3.3v [s25fl127s-3.3v-qspi-x4-dual_stacked、s25fl127s-3.3v-qspi-x4-single、s25fl127s-3.3v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	s25flxxxs	s25fl256s-1.8v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	s25flxxxs	s25fl256s-3.3v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	s25flxxxs	s25fl512s-1.8v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 51: Zynq UltraScale+ MPSoC デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	s25flxxxs	s25fl512s-3.3v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	s70flxxxp	s70fl01gs_00	1,024	x4-dual_stacked
QSPI	mx25l	mx25l12833f [mx25l12833f-qspi-x1-dual_stacked、mx25l12833f-qspi-x1-single、mx25l12833f-qspi-x2-dual_stacked、mx25l12833f-qspi-x2-single、mx25l12833f-qspi-x4-dual_stacked、mx25l12833f-qspi-x4-single、mx25l12833f-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mx25l	mx25l25635f [mx25l25645g-qspi-x1-dual_stacked、mx25l25645g-qspi-x1-single、mx25l25645g-qspi-x2-dual_stacked、mx25l25645g-qspi-x2-single、mx25l25645g-qspi-x4-dual_stacked、mx25l25645g-qspi-x4-single、mx25l25645g-qspi-x8-dual_parallel]	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mx25l	mx25l51245g	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mx25u	mx25u12835f [mx25u12832f-qspi-x1-dual_stacked、mx25u12832f-qspi-x1-single、mx25u12832f-qspi-x2-dual_stacked、mx25u12832f-qspi-x2-single、mx25u12832f-qspi-x4-dual_stacked、mx25u12832f-qspi-x4-single、mx25u12832f-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mx25u	mx25u25635f [mx25u25645g-qspi-x1-dual_stacked、mx25u25645g-qspi-x1-single、mx25u25645g-qspi-x2-dual_stacked、mx25u25645g-qspi-x2-single、mx25u25645g-qspi-x4-dual_stacked、mx25u25645g-qspi-x4-single、mx25u25645g-qspi-x8-dual_parallel]	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mx25u	mx25u51245g	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mx66l	mx66l1g45g	1,024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 51: Zynq UltraScale+ MPSoC デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	mx66l	mx66l2g45g	2,048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mx66u	mx66u51235f	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mx66u	mx66u1g45g	1,024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	mx66u	mx66u2g45g	2,048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

Zynq UltraScale+ RFSoc コンフィギュレーション メモリ デバイス

次の表には、Zynq UltraScale+ RFSoc デバイスのコンフィギュレーションにサポートされるフラッシュ デバイスで、Vivado® ツールで消去、ブランク チェック、プログラム、および検証可能なものを示します。

この付録にある表には、Vivado ソフトウェアで消去、ブランク チェック、プログラム、検証が可能な不揮発性メモリがサイリンクス ファミリごとにリストされています。サイリンクスでは、これらのコンポーネントを含むエンドプロダクトの長期メンテナンスをサポートするため、コンポーネントが新しいデザインには適さなくなったとしても、できる限りこのリストにコンポーネントをリストするようにしています。



重要: 不揮発性メモリの市場は変化が激しいため、利用可能なデバイスおよびそのライフサイクルに関しては、サイリンクスでは不揮発性メモリの製造業者に確認を取ることをお勧めしています。表に掲載されている特定デバイスの情報は、現在または今後の使用可能状態を確約するものではありません。

表 52: Zynq UltraScale+ RFSoc デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
EMMC			jedec4.51-4gb	32,768	
EMMC			jedec4.51-8gb	65,536	
EMMC			jedec4.51-16gb	131,072	
EMMC			jedec4.51-32gb	262,144	
EMMC			jedec4.51	524,288	
EMMC			jedec4.51-64gb	524,288	
EMMC	Micron	mtfc	mtfc8gakajcn-4m	65,536	
NAND	Micron	mt29f	mt29f2g08ab	2,048	x8-dual、x8-single
NAND	Micron	mt29f	mt29f8g08ab	8,192	x8-dual、x8-single
NAND	Micron	mt29f	mt29f16g08ab	16,384	x8-dual、x8-single
NAND	Micron	mt29f	mt29f32g08ae	32,768	x8-dual、x8-single
NAND	Micron	mt29f	mt29f64g08ae	65,536	x8-dual、x8-single
NAND	Spansion	s34ml	s34ml01g1	1,024	x8-dual、x8-single
NAND	Spansion	s34ml	s34ml02g1	2,048	x8-dual、x8-single
QSPI			qspi	0	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp080d	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp016d	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 52: Zynq UltraScale+ RFSoC デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	ISSI	is25lp	is25lp032d	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp128f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25lp	is25lp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp080d	8	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp016d	16	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp032d	32	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 52: Zynq UltraScale+ RFSoc デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	ISSI	is25wp	is25wp064a	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp128f	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp256d	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	ISSI	is25wp	is25wp512m	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql128 [n25q128-3.3v-qspi-x1-dual_stacked、n25q128-3.3v-qspi-x1-single、n25q128-3.3v-qspi-x2-dual_stacked、n25q128-3.3v-qspi-x2-single、n25q128-3.3v-qspi-x4-dual_stacked、n25q128-3.3v-qspi-x4-single、n25q128-3.3v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql256 [n25q256-3.3v-qspi-x1-dual_stacked、n25q256-3.3v-qspi-x1-single、n25q256-3.3v-qspi-x2-dual_stacked、n25q256-3.3v-qspi-x2-single、n25q256-3.3v-qspi-x4-dual_stacked、n25q256-3.3v-qspi-x4-single、n25q256-3.3v-qspi-x8-dual_parallel]	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql512 [n25q512-3.3v-qspi-x1-dual_stacked、n25q512-3.3v-qspi-x1-single、n25q512-3.3v-qspi-x2-dual_stacked、n25q512-3.3v-qspi-x2-single、n25q512-3.3v-qspi-x4-dual_stacked、n25q512-3.3v-qspi-x4-single、n25q512-3.3v-qspi-x8-dual_parallel]	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 52: Zynq UltraScale+ RFSoc デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Micron	mt25ql	mt25ql01g [n25q00a-3.3v-qspi-x1-dual_stacked、n25q00a-3.3v-qspi-x1-single、n25q00a-3.3v-qspi-x2-dual_stacked、n25q00a-3.3v-qspi-x2-single、n25q00a-3.3v-qspi-x4-dual_stacked、n25q00a-3.3v-qspi-x4-single、n25q00a-3.3v-qspi-x8-dual_parallel]	1,024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25ql	mt25ql02g	2,048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu128 [n25q128-1.8v-qspi-x1-dual_stacked、n25q128-1.8v-qspi-x1-single、n25q128-1.8v-qspi-x2-dual_stacked、n25q128-1.8v-qspi-x2-single、n25q128-1.8v-qspi-x4-dual_stacked、n25q128-1.8v-qspi-x4-single、n25q128-1.8v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu256 [n25q256-1.8v-qspi-x1-dual_stacked、n25q256-1.8v-qspi-x1-single、n25q256-1.8v-qspi-x2-dual_stacked、n25q256-1.8v-qspi-x2-single、n25q256-1.8v-qspi-x4-dual_stacked、n25q256-1.8v-qspi-x4-single、n25q256-1.8v-qspi-x8-dual_parallel]	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu512 [n25q512-1.8v-qspi-x1-dual_stacked、n25q512-1.8v-qspi-x1-single、n25q512-1.8v-qspi-x2-dual_stacked、n25q512-1.8v-qspi-x2-single、n25q512-1.8v-qspi-x4-dual_stacked、n25q512-1.8v-qspi-x4-single、n25q512-1.8v-qspi-x8-dual_parallel]	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu01g [n25q00a-1.8v-qspi-x1-dual_stacked、n25q00a-1.8v-qspi-x1-single、n25q00a-1.8v-qspi-x2-dual_stacked、n25q00a-1.8v-qspi-x2-single、n25q00a-1.8v-qspi-x4-dual_stacked、n25q00a-1.8v-qspi-x4-single、n25q00a-1.8v-qspi-x8-dual_parallel]	1,024	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	mt25qu	mt25qu02g	2,048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 52: Zynq UltraScale+ RFSoc デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Micron	n25q	n25q64-1.8v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Micron	n25q	n25q64-3.3v	64	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxl	s25fl256l	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxp	s25fl129p	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl128s-1.8v [s25fl127s-1.8v-qspi-x4-dual_stacked、s25fl127s-1.8v-qspi-x4-single、s25fl127s-1.8v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl128s-3.3v [s25fl127s-3.3v-qspi-x4-dual_stacked、s25fl127s-3.3v-qspi-x4-single、s25fl127s-3.3v-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl256s-1.8v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 52: Zynq UltraScale+ RFSoc デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Spansion	s25flxxxs	s25fl256s-3.3v	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl512s-1.8v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s25flxxxs	s25fl512s-3.3v	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Spansion	s70flxxxp	s70fl01gs_00	1,024	x4-dual_stacked
QSPI	Macronix	mx25l	mx25l12835f [mx25l12833f-qspi-x1-dual_stacked、mx25l12833f-qspi-x1-single、mx25l12833f-qspi-x2-dual_stacked、mx25l12833f-qspi-x2-single、mx25l12833f-qspi-x4-dual_stacked、mx25l12833f-qspi-x4-single、mx25l12833f-qspi-x8-dual_parallel]	128	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l25635f [mx25l25645g-qspi-x1-dual_stacked、mx25l25645g-qspi-x1-single、mx25l25645g-qspi-x2-dual_stacked、mx25l25645g-qspi-x2-single、mx25l25645g-qspi-x4-dual_stacked、mx25l25645g-qspi-x4-single、mx25l25645g-qspi-x8-dual_parallel]	256	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel
QSPI	Macronix	mx25l	mx25l51245g	512	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

表 52: Zynq UltraScale+ RFSoc デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Macronix	mx25u	mx25u12835f [mx25u12832f-qspi-x1-dual_stacked, mx25u12832f-qspi-x1-single, mx25u12832f-qspi-x2-dual_stacked, mx25u12832f-qspi-x2-single, mx25u12832f-qspi-x4-dual_stacked, mx25u12832f-qspi-x4-single, mx25u12832f-qspi-x8-dual_parallel]	128	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx25u	mx25u25635f [mx25u25645g-qspi-x1-dual_stacked, mx25u25645g-qspi-x1-single, mx25u25645g-qspi-x2-dual_stacked, mx25u25645g-qspi-x2-single, mx25u25645g-qspi-x4-dual_stacked, mx25u25645g-qspi-x4-single, mx25u25645g-qspi-x8-dual_parallel]	256	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx25u	mx25u51245g	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx66l	mx66l1g45g	1,024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx66l	mx66l2g45g	2,048	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx66u	mx66u51235f	512	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel
QSPI	Macronix	mx66u	mx66u1g45g	1,024	x1-dual_stacked, x1-single, x2-dual_stacked, x2-single, x4-dual_stacked, x4-single, x8-dual_parallel

表 52: Zynq UltraScale+ RFSoc デバイス コンフィギュレーションでサポートされるフラッシュ メモリ デバイス (続き)

インターフェイス	製造業者	製造業者ファミリ	デバイス (エイリアス)	集積度 (メガビット)	データ幅 (ビット)
QSPI	Macronix	mx66u	mx66u2g45g	2,048	x1-dual_stacked、x1-single、x2-dual_stacked、x2-single、x4-dual_stacked、x4-single、x8-dual_parallel

hw_server のコマンド ライン オプション

次は、hw_server コマンド ライン オプションのリストです。



重要: PC で実行される hw_server にリモート接続する場合は、ファイアウォール ポリシーが正しく設定されていることを確認してください。hw_server.exe にポート 3121 の新しいソケット接続を受信するパーミッションがあるようにします。

標準 hw_server オプション

表 53: 標準 hw_server オプション

オプション	Purpose	例
-dd	デーモン モードで hw_server を実行 (出力はシステム ログに送信される)。	hw_server -d
--help	基本的なコマンド ライン ヘルプ。	
-I	指定した時間に構築されたターゲット接続がない場合は終了。この時間は秒で指定。	hw_server -I 20
-L	クライアントから hw_server への JTAG コマンドの記録をイネーブル。	hw_server -L- 出力を stdout へ記録。 hw_server -Lmy_file.log 出力を my_file.log ファイルへ記録。
-s	エージェント リスニング ポートとプロトコルを設定。	hw_server -step::3122 hw_server スタートアップ時にポート 3122 でローカル ホストを使用して接続。
-q	スタートアップ時にバージョン情報を表示しない。	
-p	ザイリンクス GDB サーバーで使用するポートの範囲を割り当てるか、機能をオフ。	hw_server -p<port> <port> はザイリンクス GDB サーバー用のベース ポート番号です。デフォルトは 3000 です。 サーバーはターゲット アーキテクチャごとにポートを 1 つ開きます。 3000: Arm; 3001: Arm64; 3002: MicroBlaze; 3003: MicroBlaze64 hw_server -p0 disables the port

表 53: 標準 hw_server オプション (続き)

オプション	Purpose	例
--init	hw_server へ初期化スクリプト ファイルを指定。	hw_server --init my_init.txt my_init.txt ファイルには、スタートアップ時に hw_server に適用されるオプションを含有。
	環境変数 HW_SERVER_INIT_FILE デフォルトの --init ファイルを指定するのを使用。	export HW_SERVER_INIT_FILE=~my_init.txt hw_server この場合、hw_server は環境変数 HW_SERVER_INIT_FILE で指定したファイルからの設定で初期化されます。

アドバンス オプション

表 54: アドバンス オプション

オプション	目的
detect-ir-length	<p>スキャン チェーンの発見中の IR 長の検出をディスエーブルにします。</p> <p>デバイス表に追加されたデバイスを使用して hw_server を開始します。デフォルトでは、hw_server は XICOM SQL jtag マスター テーブルからコンパイルされるデバイスのプリセットリストで開始されます。これらの設定は、set device-info-file オプションを使用して上書きまたは追加できます。指定されるファイルは .csv ファイルで、"#" で始まる行は無視されます。</p> <p>次のように hw_server 開始時に .csv ファイルを指定します。</p> <pre>hw_server -e "set device-info-file my_file.csv"</pre> <p>.csv ファイル (hw_server_device_info_file.csv) は次のようにフォーマットされます。</p> <pre>##### # File: hw_server_device_info_file.csv # Description: # This is a sample jtag ID table. This file can be used to define # additional devices to be detected by the hw_server application. # # In this file empty lines, lines with spaces, and lines that begin # with the '#' character will be ignored. # # The format of this file is as follows: # # ROW 1: fields # The standard JTAG id fields are "idcode,mask,irlen,name" # ROW 2: field types # For each field specified in ROW 1, the type is used to # interpret the field data read per device. The types # accepted are "i" for integer and "s" for string. If # you use the standard fields on ROW 1 then you should # use "i,i,i,s" in ROW 2 to set the fields idcode,mask # irlen to integer and name as string # # To use this table you start hw_server with the following # command line arguments: # # hw_server -e "set device-info-file <file>" # # Where <file> is replaced with this filename. ##### idcode,mask,irlen,name i,i,i,s 167784595,268435455,10,chipscope_soft_tap</pre>

表 54: アドバンス オプション (続き)

オプション	目的
device-info-file	使用するデフォルトのデバイス ファイル .csv を設定します。
max-jtag-devices	<p>スキャン チェーンで検出可能なデバイスの最大数を増加します。デフォルトは 32 です。</p> <p>スキャン チェーンのデフォルトのデバイス数を超える数を検出できる状態で hw_server を開始します。この設定のデフォルト値は 32 です。この値は、より長い JTAG チェーン用に増加できます。この値を大きくすると、デバイス検出プロセスに時間がかかり、ケーブル アクセスが遅くなってしまう可能性があることに注意してください。このため、この値はデバイス数の多いシステムの場合にのみ増加するようにしてください。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set max-jtag-devices 64"</pre>
xdb-user-bscan	<p>xsdb コアをスキャンするのに使用する bscan を設定します。</p> <p>別の bscan にある xsdb マスター コアの hw_server スキャンを開始します。デフォルトでは、hw_server はユーザー 1 および 3 の bscan がスキャンされますが、このオプションを使用すると、hw_server を開始したときに、別の bscan ユーザー スロットにある bscan を検索させることができます。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set xsdb-user-bscan 1,2,3,4"</pre> <p>このパラメーターの引数でパラメーターのリストを指定します。リストは、1 ～ 4 の範囲をカンマ (,) で区切ったものになります。指定する最小値は 1、最大値は 4 です。</p>
mdm-detect-bscan-mask	<p>mdm コアをスキャンするのに使用する bscan を設定します。</p> <p>別の bscan にある MicroBlaze マスター コアの hw_server スキャンを開始します。デフォルトでは、hw_server はユーザー 2 bscan をスキャンします。このオプションを使用すると、hw_server を開始したときに、別の bscan ユーザー スロットにある MicroBlaze を検索させることができます。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set mdm-detect-bscan-mask 2"</pre> <p>bitmask は hw_server で見つかった FPGA 用です。次はよくある bitmask 設定の例です。</p> <p>マスク値 スキャンされた BSCAN</p> <p>0 none</p> <p>1 User1</p> <p>3 User1, User2</p> <p>7 User1, User2, User3</p> <p>f User1, User2, User3, User4</p>
always-open-jtag	<p>開始時にすべてのターゲットを開くよう hw_server を設定します。</p> <p>hw_server が開始される場合、デフォルトではどのケーブルも初期化されません。最初の接続が開始されたら、ケーブルが検出されて開きます。この検出には、システムによって数秒から数分かかります。初期化が終了したら、ケーブルが読み込まれ、デバイスが検出されます。</p> <p>場合によっては、ケーブルを検出させて使用できるようにしておく必要があります。たとえば、Linux システムをボード サーバーとして設定する場合は、ケーブルを常に初期化して接続に使用できるようにしておく必要があることもあります。このような場合、always-open-jtag オプションを使用するとケーブルを開いておくことができます。</p> <p>次は開始時のデフォルト設定です。渡す必要のある引数はありません。</p> <pre>hw_server -e "set always-open-jtag 0"</pre> <p>ケーブルを常に開いておくには、この引数を次のように指定します。</p> <pre>hw_server -e "set always-open-jtag 1"</pre>

表 54: アドバンス オプション (続き)

オプション	目的
auto-open-servers	<p>XVC ケーブルを開くのに使用した特定のパラメーターでケーブル サーバーを開きます。</p> <p>これは、XVC ケーブルを自動的に開いて、どのタイプの USB ケーブルを hw_server で検出する必要があるのかを制御するためのデバッグ オプションです。auto-open-servers パラメーターの引数は、フィールドをカンマで区切ったリストです。フィールドのカテゴリはそれぞれコロンで区切られており、最初のサブフィールドはタイプを、残りのサブフィールドはタイプの詳細を示します。xilinx-xvc タイプの場合、そのサブフィールドは host および port です。auto-open-servers のデフォルト値は * で、この場合 hw_server ですべてのケーブル タイプが検出されます。XVC のようなパラメーターを必要とする Cables タイプには * は使用できません。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set auto-open-servers xilinx-xvc:localhost:10200"</pre> <p>使用する可能性のある 2 つのサーバーを開くには、次を実行します。</p> <pre>hw_server -e "set auto-open-servers xilinx-xvc:localhost:10200,xilinx-xvc:localhost:10210"</pre> <p>使用する可能性のあるすべての USB ベースのケーブルに加えて、2 つの XVC のサーバーを開くには、次を実行します。</p> <pre>hw_server -e "set auto-open-servers *:xilinx-xvc:localhost:10200,xilinx-xvc:localhost:10210"</pre>
auto-open-ports	<p>ポート (ケーブルまたはスキャン チェーン) を自動的に開くかどうかを指定します。</p> <p>このオプションでは、JTAG スキャン チェーンを自動で開くように制御できます。auto-open-ports パラメーターの引数はブール値で、1 の場合はすべての既知の JTAG スキャン チェーンが自動的に開き、0 の場合はクライアントが選択した JTAG スキャン チェーンが開きます。デフォルト値は 1 です。0 に設定するのは、たとえば複数の JTAG スキャン チェーンが 1 つのホストに接続され、各スキャン チェーンにアクセスするのに hw_server の異なるインスタンスが使用されている場合などです。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set auto-open-ports 0"</pre>
xvc-timeout	<p>XVC サーバーをデバッグしやすくするために XVC タイムアウト値を変更します。</p> <p>これは、XVC トランザクションが終了するのに必要なタイムアウト時間を増加するためのデバッグ オプションです。xvc-timeout パラメーターの引数は秒数で指定します。値 0 の場合はタイムアウトが無効になり、無期限に待機状態になります。</p> <p>次のように、hw_server 起動時にオプションを指定します。</p> <pre>hw_server -e "set xvc-timeout 100"</pre>

表 54: アドバンス オプション (続き)

オプション	目的
xvc-servers	<p>ケーブル用の XVC サーバーを開始します。</p> <p>hw_server で指定したケーブル用に XVC サーバーが開始されるようになります。xvc-servers パラメーターの引数は、XVC サーバーの詳細をカンマ (,) で区切ったリストです。XVC サーバーの詳細は、ケーブル ID、XVC サーバーのホスト名または数、およびポート番号などがそれぞれコロンで区切られたリストです。ケーブル ID は、製造元の名前とスラッシュ (/) 文字で区切られた一意の識別子で、ケーブル ID 名の一部が使用されることもあれば、ホスト名が空の場合もあります。空の場合は、サーバーが全ネットワーク インターフェイスで入ってくる接続を受信するはずであることを示します。</p> <p>XVC クライアントと XVC サーバーの両方が同じデバイス タイプのデバッガーで、XVC の locking モードが使用される場合にインターフェイスを回避する方法は、processor-debug-claim を参照してください。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set xvc-servers 210203356596A:localhost:3000"</pre> <p>この場合でも、まずこの hw_server インスタンスに接続して、ケーブル インターフェイスを初期化する必要があることに注意してください。ケーブルに接続しない場合、次のようなメッセージが表示されます。</p> <pre>TCF 19:11:02.417: XVC open port failed: Cannot find JTAG cable matching 210203A0314DA</pre> <p>XVC ケーブルが自動的に開いて XVC にロックされるようにするには、次のように always-open-jtag オプションを追加します。</p> <pre>hw_server -e 'set xvc-servers 210203A0314DA:xcoatslab-9:3122' -e 'set always-open-jtag 1'</pre>
xvc-packet-len	<p>XVC サーバーの最大パッケージ長を変更します。</p> <p>xvc-servers option を使用して開始された XVC サーバーで返される XVC パッケージ長を制御できます。現在のデフォルトのパッケージ長は 16000 ですが、新しいバージョンで変更することもできます。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set xvc-servers 210203356596A:localhost:3000" -e "set xvc-packet-len 1000"</pre>
xvc-version	<p>XVC サーバーの XVC プロトコル バージョンを変更します。</p> <p>このデバッグ オプションを xvc-servers と共に使用して、XVC クライアントに示す XVC プロトコル バージョンを制御できます。現在のデフォルトのプロトコル バージョンは 1.1 ですが、新しいバージョンの XVC プロトコルが定義されると変更される場合があります。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set xvc-servers 210203356596A:localhost:3000" -e "set xvc-version 1.0"</pre>
xvc-capabilities	<p>XVC サーバーの XVC 機能を変更します。</p> <p>このデバッグ オプションを xvc-servers と共に使用して、XVC クライアントに示す機能を制御できます。現在のデフォルト機能は locking、status、および state-aware ですが、今後のバージョンでさらに機能が追加される場合があります。xvc-version が 1.0 に設定されている場合は、このオプションを使用しても効果はありません。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set xvc-servers 210203356596A:localhost:3000" -e "set xvc-capabilities status,state-aware"</pre>

表 54: アドバンス オプション (続き)

オプション	目的
processor-debug-claim	<p>一部のデバイス タイプを自動的に要求し、デバッガーでそれらが使用されないようにします。 hw_server デバッガーで選択したデバイス タイプがデバッグに使用されないようにできます。 このオプションの引数はビット マスクです。デフォルトでは、hw_server でのデバッグにすべての既知のデバイス タイプが使用されます。ビット値は、次を意味します。</p> <p>ビット デバイス タイプ</p> <p>0 Arm DAP 1 MPSoC 2 FPGA</p> <p>このオプションは、xvc-servers を使用して上記のデバイスの 1 つまたは複数のデバッガーである XVC クライアントを指定して XVC サーバーを開始する場合や、hw_server を上記のデバイスの 1 つまたは複数のデバッガーである XVC サーバーに接続する場合などに便利です。どちらの場合も、ロッキング機能を使用してデバッガー間でのタイム シェアリングを可能にする場合にのみ関係します。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set processor-debug-claim 2"</pre>
jtag-poll-delay	<p>遅延 (uS) を指定します。デフォルトは 50000 です。 JTAG ポーリング周波数を削減するポーリング オプションを指定します。JTAG ポーリング周波数とは JTAG ポーリング動作間にかかる最小周期です。デフォルトの最小値は 50,000 uS です。 jtag-poll-delay パラメーターの引数は、uS の数値です。</p>
help	<p>hw_server e オプションを表示します。 hw_server に使用可能な "e" オプションすべてを表示します。 次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e help</pre>
show-all	<p>hw_serv に渡されるすべてのオプションを表示。 hw_server の "e" オプション設定すべてを表示します。 次のように、hw_server 起動時にオプションを指定します。</p> <pre>hw_server -e show-all</pre>
jtag-default-frequency	<p>すべてのケーブルのデフォルト 周波数を設定します。 デフォルトの JTAG TCK 周波数を設定します。jtag-default-frequency パラメーターは Hz の数値です。 次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set jtag-default-frequency 5000000"</pre>

表 54: アドバンス オプション (続き)

オプション	目的
jtag-port-filter	<p>JTAG ポート フィルターを設定します。</p> <p>このオプションは、JTAG ポート フィルターを制御するために使用します。設定すると、hw_server でフィルターに一致しない JTAG ポートは無視されます。このパラメーターの引数は、ポート ID のすべてまたは一部をカンマで区切ったリストです。ポート ID は <manufacturer>/<productid>/<serial><port> 形式の文字列です。フィルターは、ポート フィルターの文字列のいずれかがポート ID の文字列に含まれていると一致します。</p> <p>このパラメーターは、hw_server で処理されるケーブルを指定するために同じホストで hw_server の複数インスタンスを実行する際に便利です。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set jtag-port-filter Xilinx/DLC10/0000128f515601"</pre> <p>次の例の場合、ザイリンクス DLC9 または DLC10 ケーブルをフィルターし、hw_server がポート 3122 で開始するようになります。</p> <pre>hw_server -stcp::3122 -e "set jtag-port-filter DLC9,DLC10"</pre>
bscan-switch-user-mask	<p>bscan スイッチをイネーブルにします。</p> <p>bscan スイッチの検出を制御します。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set bscan-switch-user-mask <user-bit-mask>"</pre>
jtag-port-devices	<p>JTAG ポート デバイス リストを設定します。</p> <p>JTAG スキャン チェーンのスタティックなデバイス リストを指定します。指定すると、hw_server で IDCODE レジスタが読み込まれず、スキャン チェーンのデバイスが検出されなくなります。これはスキャン チェーンに IEEE 1149.1 仕様に準拠しないデバイスが含まれる場合に便利なオプションです。このパラメーターに指定する値は、スキャン チェーンのデバイスと同じ順番で IDCODE 値をカンマで区切ったリストです。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set jtag-port-devices 0xe970203f,0x03632093"</pre>
max-ir-length	<p>IR 幅が 64 ビットを超える JTAG チェーンのデバイスをイネーブルにします。</p> <p>このオプションでは、64 ビットを超える IR 幅を使用できる状態で hw_server が開始できます。この設定のデフォルト値は 64 です。この値は、JTAG チェーンのデバイスの IR 幅がこれより大きい場合 (たとえば、93 など) に増加できます。この値を大きくすると、デバイス検出プロセスに時間がかかり、ケーブル アクセスが遅くなってしまう可能性があることに注意してください。このため、この値は IR 長が長いデバイスを含むシステムの場合にのみ増加するようにしてください。</p> <p>次のように、hw_server 開始時にオプションを指定します。</p> <pre>hw_server -e "set max-ir-length 93"</pre>

その他のリソースおよび法的通知

ザイリンクス リソース

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

ソリューション センター

デバイス、ツール、IP のサポートについては、[ザイリンクス ソリューション センター](#)を参照してください。デザイン アシスタント、デザイン アドバイザリ、トラブルシューティングのヒントなどが含まれます。

Documentation Navigator およびデザイン ハブ

ザイリンクス Documentation Navigator (DocNav) では、ザイリンクスの資料、ビデオ、サポート リソースにアクセスでき、特定の情報を取得するためにフィルター機能や検索機能を利用できます。DocNav を開くには、次のいずれかを実行します。

- Vivado® IDE で [Help] → [Documentation and Tutorials] をクリックします。
- Windows で [スタート] → [すべてのプログラム] → [Xilinx Design Tools] → [DocNav] をクリックします。
- Linux コマンド プロンプトに「docnav」と入力します。

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

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

注記: DocNav の詳細は、ザイリンクス ウェブサイトの [Documentation Navigator](#) ページを参照してください。DocNav からは、日本語版は参照できません。ウェブサイトのデザイン ハブ ページをご利用ください。

参考資料

このガイドの補足情報は、次の資料を参照してください。

Vivado® Design Suite の資料

1. 『Vivado Design Suite ユーザー ガイド: ロジック シミュレーション』 (UG900)
2. 『Vivado Design Suite ユーザー ガイド: 合成』 (UG901)
3. 『Vivado Design Suite ユーザー ガイド: Dynamic Function eXchange』 (UG909)
4. 『Vivado Design Suite チュートリアル: Dynamic Function eXchange』 (UG947)
5. 『UltraFast 設計手法ガイド (Vivado Design Suite 用)』 (UG949)
6. 『Vivado Design Suite ユーザー ガイド: インプリメンテーション』 (UG904)
7. 『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)
8. 『Vivado Design Suite ユーザー ガイド: デザイン フローの概要』 (UG892)
9. 『Vivado Design Suite ユーザー ガイド: I/O およびクロック プランニング』 (UG899)
10. 『Vivado Design Suite チュートリアル: プログラムおよびデバッグ』 (UG936)
11. 『Vivado Design Suite Tcl コマンド リファレンス ガイド』 (UG835)
12. 『SmartLynq データ ケーブル ユーザー ガイド』 (UG1258)
13. 『7 シリーズ FPGA コンフィギュレーション ユーザー ガイド』 (UG470: [英語版](#)、[日本語版](#))
14. 『7 シリーズ FPGA および Zynq-7000 SoC XADC デュアル 12 ビット 1MSPS アナログ-デジタル コンバーター ユーザー ガイド』 (UG480: [英語版](#)、[日本語版](#))
15. 『UltraScale アーキテクチャ コンフィギュレーション ユーザー ガイド』 (UG570: [英語版](#)、[日本語版](#))
16. 『UltraScale アーキテクチャ システム モニター ユーザー ガイド』 (UG580: [英語版](#)、[日本語版](#))
17. 『Vivado Design Suite ユーザー ガイド: IP インテグレーターを使用した IP サブシステムの設計』 (UG994)
18. 『UltraScale FPGA GTM トランシーバー ユーザー ガイド』 (UG581: [英語版](#)、[日本語版](#))
19. 『Debug Bridge LogiCORE IP 製品ガイド』 (PG245)
20. 『PetaLinux ツールを使用した Zynq-7000 でのサイリンクス仮想ケーブルの実行』 (XAPP1251)
21. 『Vivado Design Suite チュートリアル: エンベデッド プロセッサ ハードウェア デザイン』 (UG940)
22. 『サイリンクス組み込みマイクロコントローラーを使用するのインシステム プログラミング機能』 (XAPP058: [英語版](#)、[日本語版](#))
23. 『暗号化を使用して 7 シリーズ FPGA のビットストリームを保護』 (XAPP1239: [英語版](#)、[日本語版](#))
24. 『暗号化と認証を使用して UltraScale/UltraScale+ FPGA のビットストリームを保護』 (XAPP1267: [英語版](#)、[日本語版](#))
25. 『Virtual Input/Output LogiCORE IP 製品ガイド』 (PG159)
26. 『Integrated Bit Error Ratio Tester 7 Series GTX Transceivers LogiCORE IP 製品ガイド』 (PG132)
27. 『Integrated Bit Error Ratio Tester 7 Series GTP Transceivers LogiCORE IP 製品ガイド』 (PG133)
28. 『Integrated Bit Error Ratio Tester 7 Series GTH Transceivers LogiCORE IP 製品ガイド』 (PG152)
29. 『Integrated Logic Analyzer LogiCORE IP 製品ガイド』 (PG172)

30. 『JTAG to AXI Master LogiCORE IP 製品ガイド』 (PG174)
31. 『System Integrated Logic Analyzer LogiCORE IP 製品ガイド』 (PG261)
32. 『UltraScale+ Devices Integrated Block for PCI Express LogiCORE IP 製品ガイド』 (PG213: [英語版](#)、[日本語版](#))
33. 『Debug Bridge LogiCORE IP 製品ガイド』 (PG245)
34. 『In-System IBERT LogiCORE IP 製品ガイド』 (PG246)
35. 『UltraScale アーキテクチャ FPGA メモリ IP LogiCORE IP 製品ガイド』 (PG150: [英語版](#)、[日本語版](#))
36. 『AXI High Bandwidth Controller LogiCORE IP 製品ガイド』 (PG276)
37. 『IBERT for UltraScale GTM Transceivers LogiCORE IP 製品ガイド』 (PG342)

トレーニング コース

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

1. [Vivado Design Suite を使用した FPGA の設計 1](#)
2. [Vivado Design Suite を使用した FPGA の設計 2](#)
3. [Vivado Design Suite を使用した FPGA の設計 3](#)
4. [Vivado Design Suite を使用した FPGA の設計 4](#)
5. [Vivado Design Suite QuickTake ビデオ: Vivado IP インテグレーターを使用した Zynq デバイスの設計](#)
6. [Vivado Design Suite QuickTake ビデオ: Vivado Design Suite でパシカル リコンフィギュレーションを実行](#)
7. [Vivado Design Suite QuickTake ビデオ: Vivado Design Suite でのリビジョン管理の使用](#)
8. [Vivado Design Suite QuickTake ビデオ チュートリアル](#)

お読みください: 重要な法的通知

本通知に基づいて貴殿または貴社 (本通知の被通知者が個人の場合には「貴殿」、法人その他の団体の場合には「貴社」。以下同じ) に開示される情報 (以下「本情報」といいます) は、ザイリンクスの製品を選択および使用することのためにのみ提供されます。適用される法律が許容する最大限の範囲で、(1) 本情報は「現状有姿」、およびすべて受領者の責任で (with all faults) という状態で提供され、ザイリンクスは、本通知をもって、明示、黙示、法定を問わず (商品性、非侵害、特定目的適合性の保証を含みますがこれらに限られません)、すべての保証および条件を負わない (否認する) ものとし、また、(2) ザイリンクスは、本情報 (貴殿または貴社による本情報の使用を含む) に関係し、起因し、関連する、いかなる種類・性質の損失または損害についても、責任を負わない (契約上、不法行為上 (過失の場合を含む)、その他のいかなる責任の法理によるかを問わない) ものとし、当該損失または損害には、直接、間接、特別、付随的、結果的な損失または損害 (第三者が起こした行為の結果被った、データ、利益、業務上の信用の損失、その他あらゆる種類の損失や損害を含みます) が含まれるものとし、それは、たとえ当該損害や損失が合理的に予見可能であったり、ザイリンクスがそれらの可能性について助言を受けていた場合であったとしても同様です。ザイリンクスは、本情報に含まれるいかなる誤りも訂正する義務を負わず、本情報または製品仕様のアップデートを貴殿または貴社に知らせる義務も負いません。事前の書面による同意のない限り、貴殿または貴社は本情報を再生産、変更、頒布、または公に展示してはなりません。一定の製品は、ザイリンクスの限定的保証の諸条件に従うこととなるので、<https://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。IP コアは、ザイリン

クスが貴殿または貴社に付与したライセンスに含まれる保証と補助的条件に従うことになります。ザイリンクスの製品は、フェイルセーフとして、または、フェイルセーフの動作を要求するアプリケーションに使用するために、設計されたり意図されたりしていません。そのような重大なアプリケーションにザイリンクスの製品を使用する場合のリスクと責任は、貴殿または貴社が単独で負うものです。<https://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。

自動車用のアプリケーションの免責条項

オートモーティブ製品 (製品番号に「XA」が含まれる) は、ISO 26262 自動車用機能安全規格に従った安全コンセプトまたは余剰性の機能 (「セーフティ 設計」) がない限り、エアバッグの展開における使用または車両の制御に影響するアプリケーション (「セーフティ アプリケーション」) における使用は保証されていません。顧客は、製品を組み込むすべてのシステムについて、その使用前または提供前に安全を目的として十分なテストを行うものとし、セーフティ設計なしにセーフティ アプリケーションで製品を使用するリスクはすべて顧客が負い、製品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、

商標

© 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。すべてのその他の商標は、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 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。