

# Vivado Design Suite ユーザー ガイド :

## プログラムおよびデバッグ

UG908 (v2012.2) 2012 年 7 月 25 日



#### Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

本資料は英語版 (v2012.2) を翻訳したもので、内容に相違が生じる場合には原文を優先します。

資料によっては英語版の更新に対応していないものがあります。

日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

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

---

## 改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	改訂内容
2012 年 7 月 25 日	1.0	初版

# 目次

改訂履歴 .....	2
<b>第 1 章：概要</b>	
はじめに .....	5
<b>第 2 章：デバイスのプログラム</b>	
概要 .....	6
ビットストリームの生成 .....	6
ビットストリーム ファイルのフォーマット設定の変更 .....	7
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 .....	8
FPGA デバイスのプログラム .....	8
iMPACT の起動 .....	8
Vivado ハードウェア セッションを使用した FPGA デバイスのプログラム .....	9
<b>第 3 章：デザインのデバッグ</b>	
概要 .....	14
RTL レベル デザイン シミュレーション .....	14
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 .....	15
インシステム デバッグ .....	15
<b>第 4 章：インシステム デバッグ フロー</b>	
概要 .....	16
インシステム デバッグ用のデザインのプローブ .....	16
ネットリスト挿入デバッグ プローブ フロー .....	17
HDL インスタンス化 プローブ フローの概要 .....	25
HDL インスタンス化 デバッグ プローブ フロー .....	26
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 .....	30
<b>第 5 章：ハードウェアでのデザインのデバッグ</b>	
概要 .....	31
ChipScope Pro Analyzer を使用したデザインのデバッグ .....	31
Vivado ロジック解析を使用したデザインのデバッグ .....	32
ハードウェア ターゲットに接続して FPGA デバイスをプログラム .....	32
計測のための ILA コアの設定 .....	33
ILA コアのトリガー条件の設定 .....	33
ILA コアのトリガー位置の設定 .....	33
ILA プローブ情報の記述 .....	34
ILA プローブ情報の読み出し .....	34
ILA プローブの表示 .....	35
ILA プローブのトリガー比較値の設定 .....	35
ILA プローブの比較値の設定 .....	36

ILA コアのトリガーの供給.....	37
ILA コアのトリガーの停止.....	37
ILA コアのステートの表示.....	38
ILA コアからのデータを波形ビューアーで表示.....	38
ILA コアでキャプチャされたデータの保存および復元.....	38
ラボ環境での Vivado ロジック解析の使用.....	39
Vivado ロジック解析機能と ChipScope Pro Analyzer の同時使用 .....	40
ハードウェア セッションの Tcl オブジェクトおよびコマンド .....	42
ハードウェア セッションの Tcl コマンドの使用 .....	45
 <b>第 6 章：波形ウィンドウを使用した ILA プローブ データの表示</b>	
概要 .....	46
波形ウィンドウの機能および制限事項 .....	46
波形コンフィギュレーションの信号およびバス .....	47
波形コンフィギュレーションの ILA プローブ .....	48
波形コンフィギュレーションのカスタマイズ .....	49
オブジェクト名の変更 .....	52
基数およびアナログ波形 .....	54
ズーム機能 .....	57
 <b>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</b>	
デバイス コンフィギュレーション設定の説明.....	58
 <b>付録 B：その他のリソース</b>	
ザイリンクス リソース .....	64
ソリューション センター .....	64
参考資料 .....	64

# 概要

## はじめに

デザインをインプリメントしたら、FPGA デバイスをプログラムし、デザインをインシステムでデバッグしながら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] にあります (図 1-1 参照)。

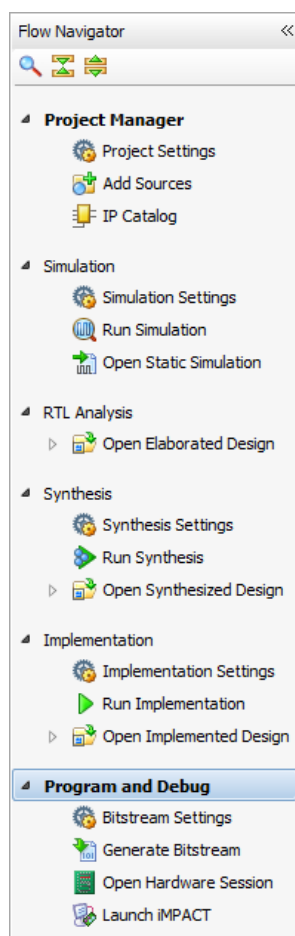


図 1-1 : Flow Navigator の [Program and Debug]

# デバイスのプログラム

---

## 概要

ハードウェアのプログラムは、次の 2 つの段階に分けることができます。

1. インプリメント済みデザインからビットストリーム データ プログラム ファイルを生成
  2. ハードウェアに接続し、プログラム ファイルをターゲット FPGA デバイスにダウンロード
- 

## ビットストリームの生成

ビットストリーム データ ファイルを生成する前に、ビットストリーム設定が正しいかどうかを確認することが重要です。

Vivado™ IDE では、次の 2 つのビットストリーム設定があります。

1. ビットストリーム 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

Flow Navigator の [Bitstream Settings] をクリックするか、[Flow] → [Bitstream Settings] をクリックすると、[Project Settings] ダイアログ ボックスの [Bitstream] ページが開きます (図 2-1)。ビットストリーム設定が正しいことを確認したら、`write_bistream` Tcl コマンドまたは 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。

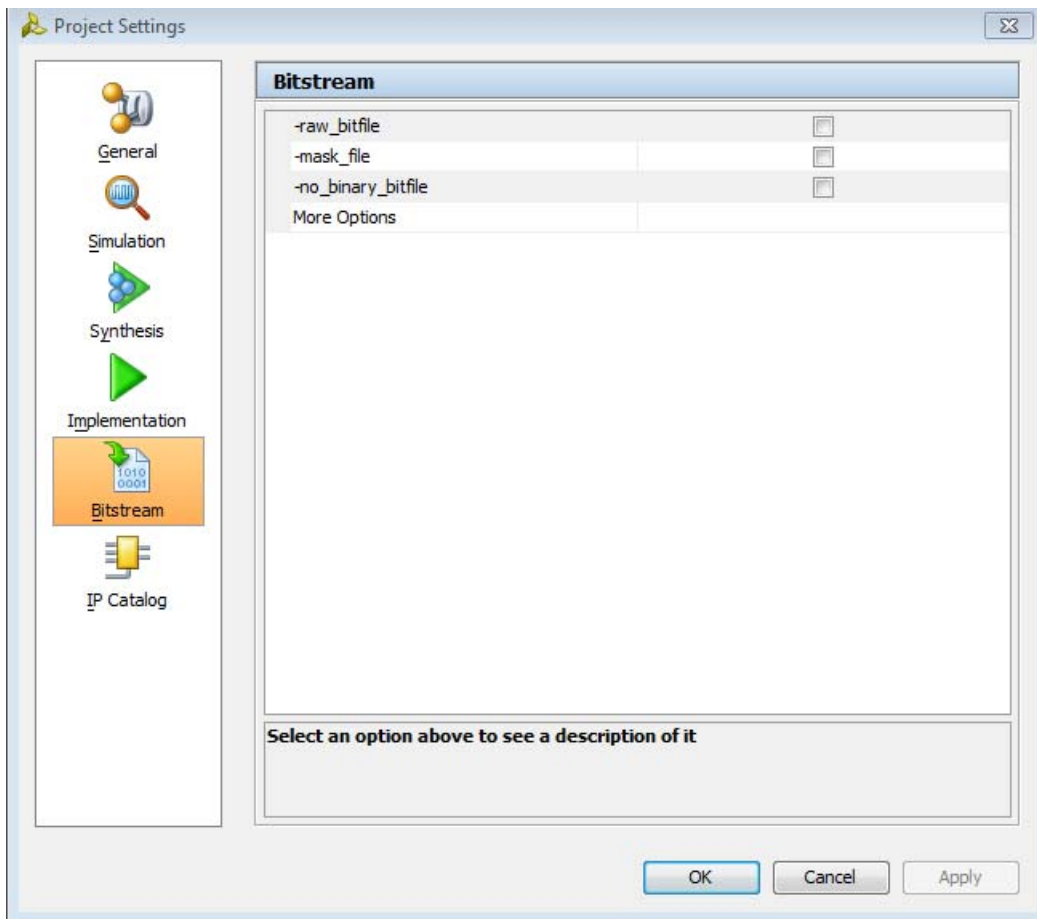


図 2-1 : [Project Settings] ダイアログ ボックスの [Bitstream] ページ

## 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_bistream` 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。

- `-raw_bitfile` : ロー ビット ファイル (.rbit) 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.rbit` となります。
- `-mask_file` : 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、マスク ビットが 1 の場合はそのビットは検証されません。出力ファイル名は `file.msk` となります。
- `-no_binary_bitfile` : 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。

# 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、XDC ファイルで `set_property` コマンドを使用して変更します。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。

## FPGA デバイスのプログラム

ビットストリーム データ プログラム ファイルを生成したら、ターゲット FPGA デバイスにダウンロードします。これには、2 つの方法があります。

- Flow Navigator で [Launch iMPACT] をクリックするか、[Flow] → [Launch iMPACT] をクリックして、iMPACT デバイス プログラム ツールを起動します。
- ハードウェア セッションを開き、Vivado IDE に含まれているネイティブ インシステム デバイス プログラミング機能を使用します。

## iMPACT の起動

iMPACT ツールでは、デバイス コンフィギュレーションとファイルの生成を実行できます。

- デバイス コンフィギュレーションでは、JTAG ダウンロード ケーブル (ザイリンクス パラレル ケーブル IV、ザイリンクス プラットフォーム ケーブル USB、ザイリンクス プラットフォーム ケーブル USB II、または Digilent JTAG ケーブル) を使用して、ザイリンクス FPGA および PROM を直接コンフィギュレーションできます。
- 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、CPLD、PROM をコンフィギュレーションまたはプログラムできます。
- ファイル生成では、System ACE™ CF、PROM、SVF、STAPL、および XSVF などのプログラム ファイルを作成できます。

iMPACT では、次も実行できます。

- デザインのコンフィギュレーションデータのリードバックおよび検証
- コンフィギュレーション エラーのデバッグ
- SVF および XSVF ファイルの実行

iMPACT は、[Generate Bitstream] コマンドが実行されているどのインプリメント済みデザインでも、Vivado IDE から直接起動できます。iMPACT を起動するには、Flow Navigator で [Launch iMPACT] をクリックします。

Vivado ツールから iMPACT を起動した場合、BIT ビットストリーム ファイルが自動的に iMPACT に読み込まれます。iMPACT の詳細は、iMPACT ヘルプを参照してください。



# Vivado ハードウェア セッションを使用した FPGA デバイスのプログラム

Vivado IDE ツールには、1 つ以上の FPGA デバイスを含むハードウェアに接続し、それらの FPGA デバイスをプログラムし、それらの FPGA デバイスに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 IDE GUI または Tcl コマンドで実行できます。どちらの場合も、ハードウェアに接続し、ターゲット FPGA 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 セッションを開きます。
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. ビットストリーム データ プログラム ファイルを適切な FPGA デバイスに関連付けます。
4. プログラム ファイルをハードウェア デバイスにプログラム (ダウンロード) します。

## ハードウェア セッションを開く

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 セッションを開きます。ハードウェア セッションを開くには、次のいずれかを実行します。

- Flow Navigator で [Program and Debug] → [Open Hardware Session] をクリックします。
- [Flow] → [Open Hardware Session] をクリックします。

## 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 (CSE サーバーとも呼ばれる) に接続します。これには、次のいずれかを実行します。

- [Hardware] ビューの [Open New Hardware Target] リンクをクリックし、ウィザードを使用してハードウェア ターゲットへの新しい接続を開きます。
- [Hardware] ビューの [Open Recent Hardware Target] リンクをクリックし、最近接続したハードウェア ターゲットへの接続を開きます。
- 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。

## 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。次の手順に従います。

1. ターゲット ボードが接続されているマシン上のハードウェア ターゲットを制御する、ホスト名と CSE サーバー (ハードウェア サーバーまたは `cse_server` と呼ばれる) のポートを指定します (図 2-2)。

**注記 :** ホスト名を `[localhost]` にすると、Vivado ツールを実行しているマシンで `cse_server` プロセスが自動的に開始し、ウィザードのこの後のページで使用されます。

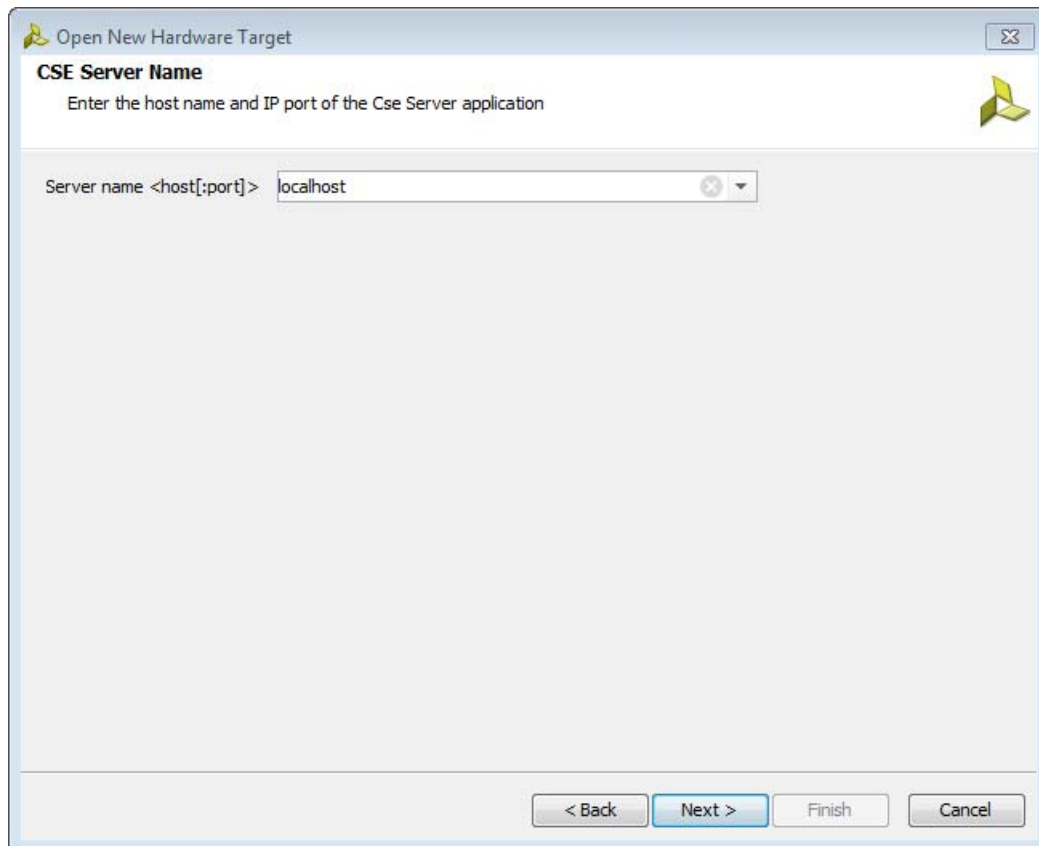


図 2-2 : CSE サーバー名の指定

- 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 (図 2-3)。

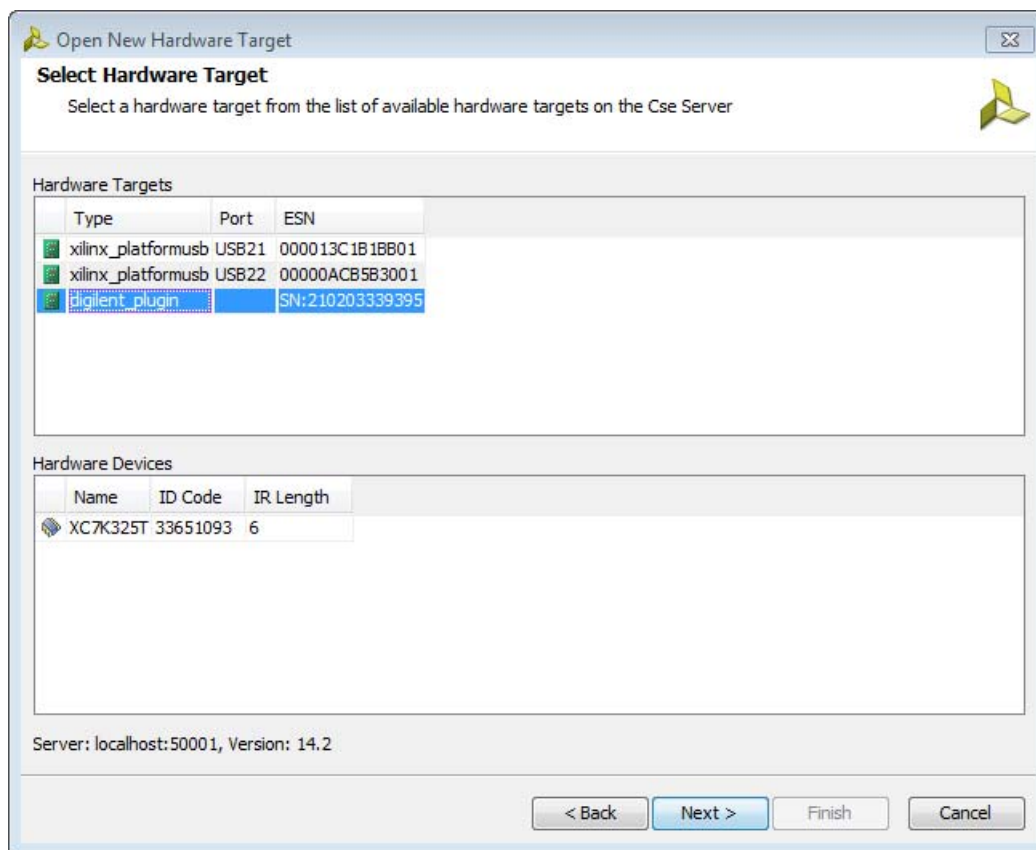


図 2-3 : ハードウェア ターゲットの選択

- TCK クロック ピンの周波数など、ハードウェア ターゲットのプロパティを設定します。ハードウェア ターゲットによって、設定可能なプロパティは異なります。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。

## 最近開いたハードウェア ターゲットを開く

Open New Hardware Target ウィザードは、[Open Recent Hardware Target] リンクをクリックした場合に使用される最近接続したハードウェア ターゲットのリストもアップデートします。[Hardware] ビューの [Open Recent Hardware Target] リンクをクリックすると、ウィザードを使用してハードウェア ターゲットに接続する代わりに、最近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。

## 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:50001 上の cse\_server で制御される diligent\_plugin ターゲット (シリアル番号 210203339395) に接続するには、次の Tcl コマンドを使用します。

```
connect_hw_server -host localhost -port 50001
current_hw_target [get_hw_targets */diligent_plugin/SN:210203339395]
open_hw_target
```

ハードウェア ターゲットへの接続を開くと、[Hardware] ビューにハードウェア サーバー、ハードウェア ターゲット、およびターゲットのハードウェア デバイスが表示されます (図 2-4)。

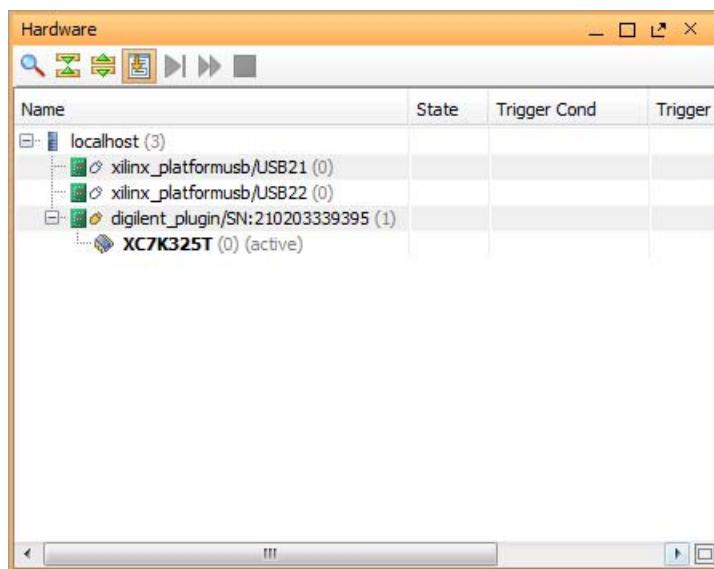


図 2-4: ハードウェア ターゲットへの接続を開いた後の [Hardware] ビュー

## 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、FPGA 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] ビューでハードウェア デバイスを選択し、[Properties] ビューで [Programming File] プロパティが適切なビットストリーム データ ファイル (.bit) 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 .bit ファイルが自動的に使用されます。

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:/design.bit} [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] ビュー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] ビューの DONE のステータスで確認できます (図 2-5)。

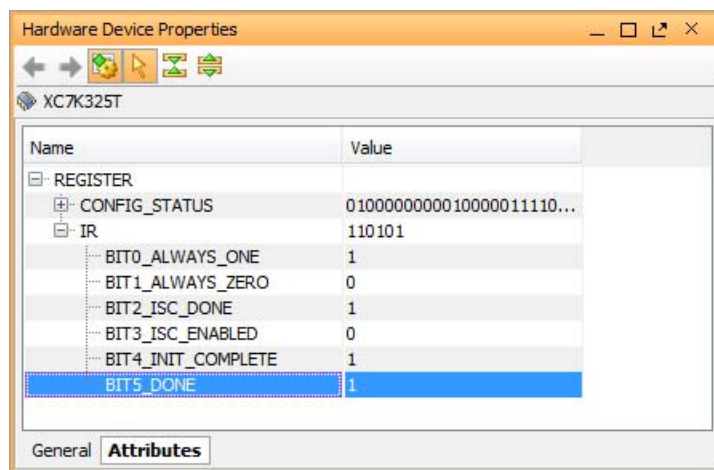


図 2-5 : FPGA デバイスの DONE ステータスを確認

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。

## 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] ビューでハードウェア 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_platformusb/USB21}
```

## 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] ビューでハードウェア 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
```

# デザインのデバッグ

---

## 概要

FPGA デザインのデバッグは複数の段階を含む反復作業です。複雑な問題を処理する場合と同様に、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。この手法は、デザインフローの次の段階で使用できます。

- RTL レベル 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 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ついては、『Vivado Design Suite ユーザーガイド：ロジックシミュレーション』(UG937) [\[参照 1\]](#) 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 デバッグ

Vivado IDE 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、システム スピードで、タイミング精度の高いデバッグを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はデザイン/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可能性があ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、ニーズに応じた方法でデザインをデバッグできます。Vivado IDE のインシステム ロジック デバッグ機能については、[第 4 章「インシステム デバッグ フロー」](#)で説明します。

# インシステム デバッグ フロー

## 概要

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: 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: 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: 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、デザインの一部分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します。
2. デザインにデバッグ コアを追加する方法を決めます。

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 インスタンス化フロー)、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 (ネットリスト挿入フロー) を決定すると有益です。表 4-1 に、異なるデバッグ方法の利点と欠点を示します。

表 4-1: 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	<ul style="list-style-type: none"><li>• 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</li><li>• Set up Debug ウィザードを使用して、ネットリスト挿入フローを実行する</li></ul>
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	<ul style="list-style-type: none"><li>• 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</li><li>• Set up Debug ウィザードを使用して、ネットリスト挿入フローを実行する</li></ul>



表 4-1: 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 プローブ フローを自動化する	<ul style="list-style-type: none"> <li>• set_property Tcl コマンドを使用し、デバッグするネットに mark_debug プロパティを設定する</li> <li>• ネットリスト挿入プローブ 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</li> </ul>
HDL ソースで ILA デバッグ コア インスタンスに信号を接続する	<ul style="list-style-type: none"> <li>• デバッグする HDL 信号を特定する</li> <li>• HDL インスタンスエーション プローブ 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</li> </ul>

## ネットリスト挿入デバッグ プローブ 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) v2.0 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] ビューを使用して、個々のコア、ポート、およびパラメーターを設定します。[Debug] ビューを開くには、レイアウト セレクターで [Debug] を選択、[Layers] → [Debug] をクリック、または [Windows] → [Debug] をクリックします。
- Tcl デバッグ コマンドを手動で入力するか、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] ビューの [Unassigned Debug Nets] の下に表示されます。

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。
- XST (Xilinx Synthesis Technology) 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。有効な値は TRUE または FALSE だけでなく、SOFT に設定して指定ネットを最適化できます。

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場合、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 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。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場合、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。

## 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;
```

## XST での mark\_debug の構文例

次に、XST 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"}
```



---

**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、次の Synopsys 社のサイトから登録をすると入手できます。

<http://www.synopsys.com/Community/Interoperability/Pages/TapinSDC.aspx>

## 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 IDE 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。デザインのさまざまな合成方法は、『Vivado Design Suite ユーザー ガイド : 合成』(UG901) [\[参照 2\]](#) 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] ビューを表示します。デバッグ用にマークした HDL 信号に対応するネットが、[Debug] ビューの [Unassigned Debug Nets] フォルダーの下に表示されます (図 4-1)。

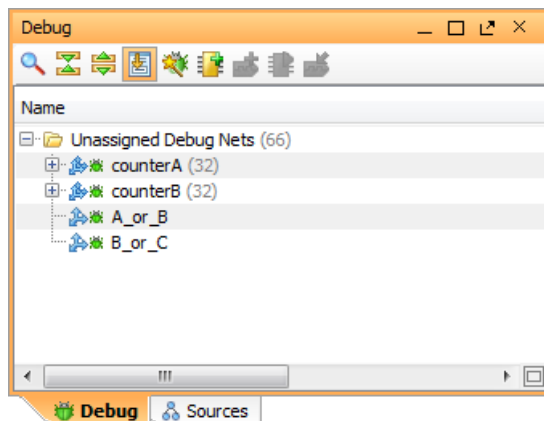


図 4-1 : [Debug] ビューの [Unassigned Debug Nets] フォルダー

合成済みデザイン ネットリストでデバッグするネットを追加でマークするには、次のいずれかの方法を使用します。

- [Netlist]、[Schematic] などの任意のビューでネットを右クリックし、[Mark Debug] をクリックします。
- 任意のビューでネットを選択し、[Debug] ビューの [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を選択します。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。

## 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 IDE 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] ビューの [Unassigned Debug Nets] フォルダを使用するか、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 IDE メイン メニューから [Tools] → [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 (図 4-2)。
4. [Add/Remove Nets] をクリックし、ネットを追加するか、既存のネットを削除します。
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 ウィザードでは、同期エレメントのパスを検索することにより、デバッグ ネットに適切なクロック ドメインが自動的に選択されます。自動的に選択されたクロック ドメインを変更する必要がある場合に、[Select Clock Domain] ダイアログ ボックスを使用します。ただし、表に示される各クロック ドメインに対して個別の ILA v2.0 コア インスタンスが作成されるので、注意が必要です。

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 ウィザードは、クロック ドメインにつき 1 つの ILA コアを挿入します。デバッグ用に選択されたネットは、挿入された ILA v2.0 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. 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 v2.0 コアを挿入および接続します。

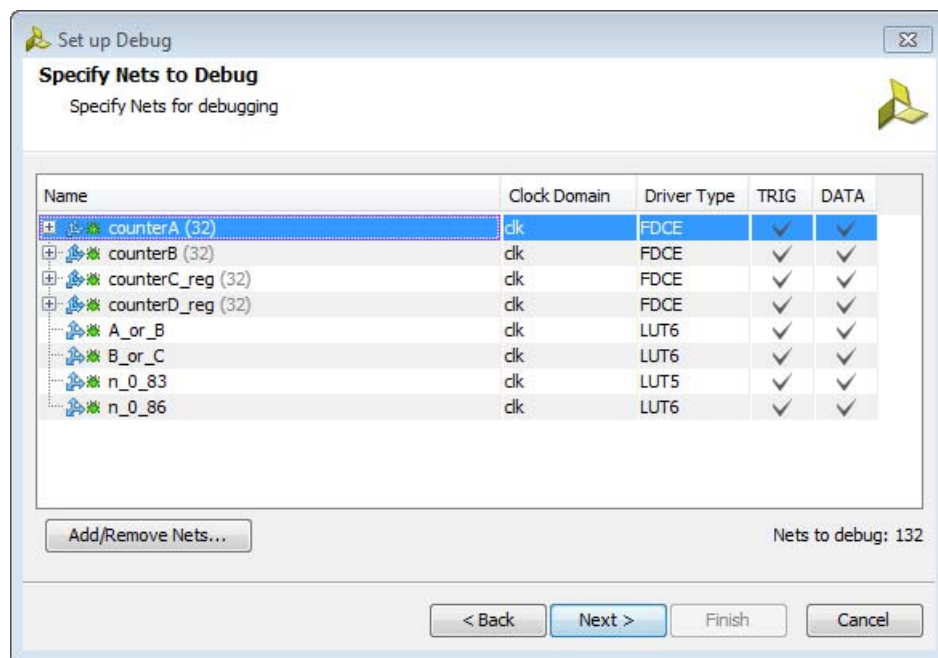


図 4-2 : Set up Debug ウィザード

## [Debug] ビューを使用したデバッグ コアの追加とカスタマイズ

[Debug] ビューでは、Set up Debug ウィザードにない ILA v2.0 コア挿入に関する詳細な設定を実行できます。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] ビューには、次のものが表示されます。

- debug\_core\_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 メニューまたはビュー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] ビューでデバッグ コアを生成するには、ツールバーの [Create Debug Core] をクリックします。[Create Debug Core] ダイアログ ボックス (図 4-3) 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] ビューでコアを右クリックし、[Delete] をクリックします。

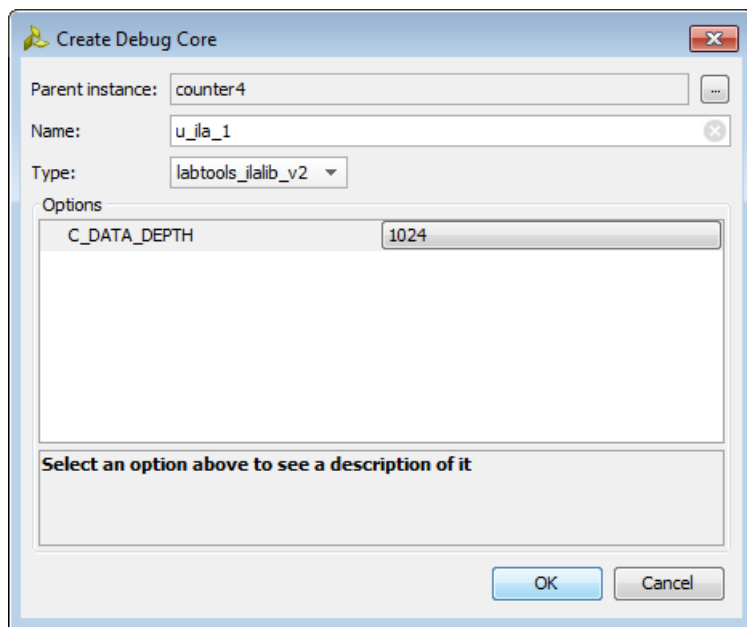


図 4-3 : [Create Debug Core] ダイアログ ボックス

## 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] ビューでデバッグ 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 (図 4-4)。
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] ビューでポートを右クリックし、[Delete] をクリックします。

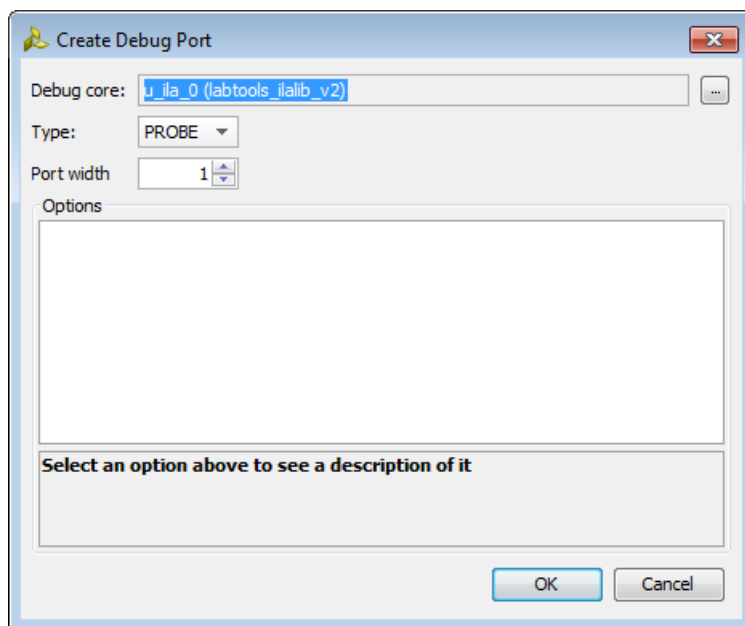


図 4-4 : [Create Debug Port] ダイアログ ボックス

## 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] ビューからデバッグ コアのポートにドラッグアンドドロップできます。ネット選択内容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。



**重要:** 複数のネットまたはバスを、ILA v2.0 コアの同じプローブ ポートに割り当てないでください。Vivado ロジック解析機能では、このようなプローブ ポートの処理に制限があります。各ネットまたはバスを 1 つのプローブ ポートに割り当ててください。

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 コアのプロパティの変更については、29 ページの「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 v2.0 デバッグ コアのプロパティも変更できます。たとえば、ILA v2.0 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 (図 4-5)。

1. [Debug] ビューで ILA コア (u\_ila\_0 など) を選択します。
2. [Instance Properties] ビューで [Debug Core Options] タブをクリックします。
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。

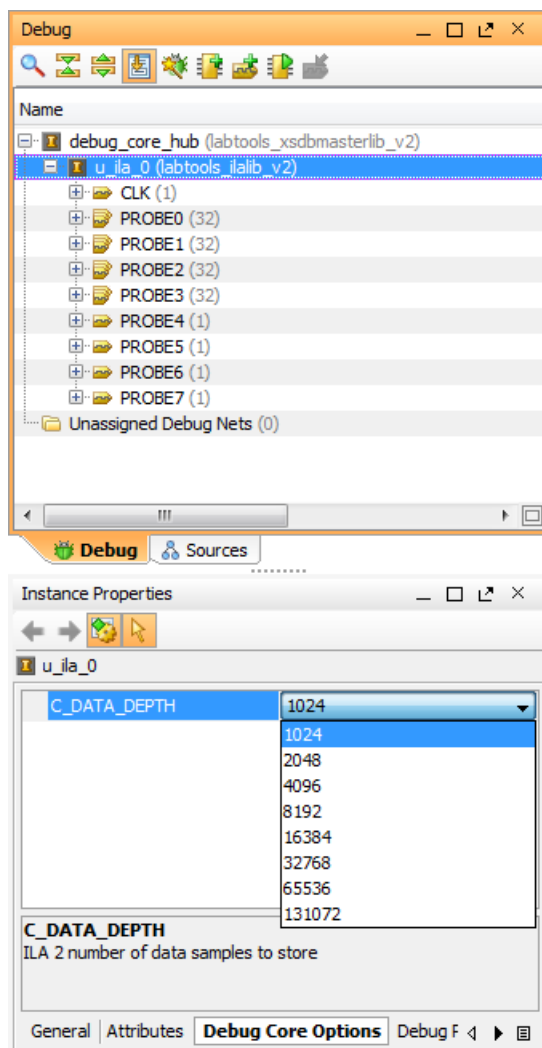


図 4-5 : ILA v2.0 コアデータの深さを変更

## Tcl コマンドを使用したデバッグ コア挿入

Set up Debug ウィザードの使用に加え、Tcl コマンドを使用してデバッグ コアを作成、接続、および合成済みデザイン ネットリストに挿入できます。

1. ILA v2.0 コアのブラック ボックスを作成します。  

```
create_debug_core u_ila_0 labtools_ilalib_v2
```
2. ILA v2.0 コアのデータ深さプロパティを設定します。  

```
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
```
3. ILA v2.0 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 v2.0 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、改めて作成する必要はありません。
4. 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。  

```
set_property port_width 1 [get_debug_ports u_ila_0/PROBE0]
```

注記: ILA v2.0 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、改めて作成する必要はありません。
5. 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]]
```
6. (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]}]]to debug
```
7. (オプション) デバッグ コアを合成し、合成済みデザインのほかの部分と共にフロアプランできるようにします。  

```
implement_debug_cores [get_debug_cores]
```

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 コンソールで「help -category ChipScope」と入力してください。

## 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。



## HDL インスタンスエーション プローブ フローの概要

HDL インスタンスエーション 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、インスタンスエート、および接続します。表 4-2 に、このフローでサポートされるデバッグ コアを示します。

表 4-2 : HDL インスタンスエーション プローブ フローで使用可能なデバッグ コア

デバッグ コア	バージョン	説明	解析ツール
ICON (Integrated Controllor)	v1.06a	ILA 1.05a および VIO 1.05a コアを JTAG チェーンに接続するためのデバッグ コア ハブ	ChipScope Pro Analyzer
VIO (Virtual Input/Output)	v1.05a	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 (ICON コアへの接続が必要)	ChipScope Pro Analyzer
ILA (Integrated Logic Analyzer)	v1.05a	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 (ICON コアへの接続が必要)	ChipScope Pro Analyzer
ILA (Integrated Logic Analyzer)	v2.0	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

ICON、VIO、および ILA v1.x コアは、これらのコアを含むレガシ デザインとの互換性のため、Vivado ツールでサポートされています。ILA v2.0 コアには、ILA v1.x コアと比較して次のような利点があります。

- Vivado ロジック解析で機能します。詳細は、31 ページの「ハードウェアでの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。



**重要 :** ILA v2.0 の HDL インスタンスエーション フローでは、合成済みデザイン ネットリストにブラック ボックス インスタンスが作成されます。このブラック ボックスは、デザイン インプリメンテーション プロセスの `opt_design` または `place_design` の段階で実際の ILA v2.0 コアに置き換えられます。

# HDL インスタンスエーション デバッグ プローブ フロー

HDL インスタンスエーション 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 v2.0 デバッグ コアをカスタマイズして生成します。
2. デバッグ コアを含むデザインを合成します。
3. (オプション) デバッグ コアのプロパティを変更します。
4. 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 (図 4-6)。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 v2.0 コアのカスタマイズの詳細は、『Integrated Logic Analyzer (ILA) v2.0 データシート』(DS875) 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] ビューに追加されます。

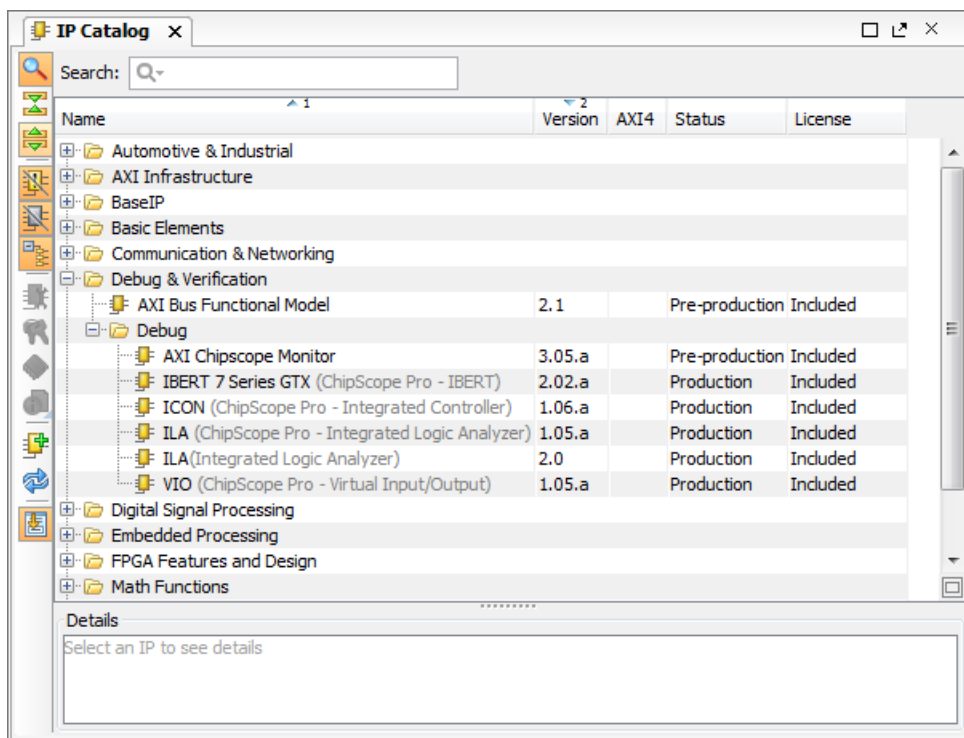


図 4-6: IP カタログのデバッグ コア

## デバッグ コアのインスタンス化 ション

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 v2.0 インスタンスの例を示します。

```
ila_v2_0_0 i_ila
(
    .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:** レガシ VIO および ILA v1.x コアでは、ICON v1.x コアへの接続が必要です。ILA v2.0 コアでは、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 `debug_core_hub` デバッグ コアが合成済みデザイン ネットリストに自動的に挿入され、ILA v2.0 コアと JTAG スキャン チェーンの間が接続されます。

## デバッグ コアを含むデザインの合成

次に、Vivado IDE 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。デザインのさまざまな合成方法は、『Vivado Design Suite ユーザー ガイド : 合成』(UG901) [\[参照 2\]](#) を参照してください。



**重要:** デザインに追加した ILA v2.0 コアはブラック ボックス インスタンスであり、デザイン インプリメンテーション プロセスの `opt_design` または `place_design` の段階で実際の ILA v2.0 コアに置き換えられます。合成中に表示される次のクリティカル警告は無視しても問題ありません。

CRITICAL WARNING:[Designutils 20-1022] Could not resolve non-primitive black box cell 'ila' instantiated in module 'inst'.

## 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] ビューを表示します。debug\_core\_hub に接続されている ILA v2.0 コアが表示されます (図 4-7)。

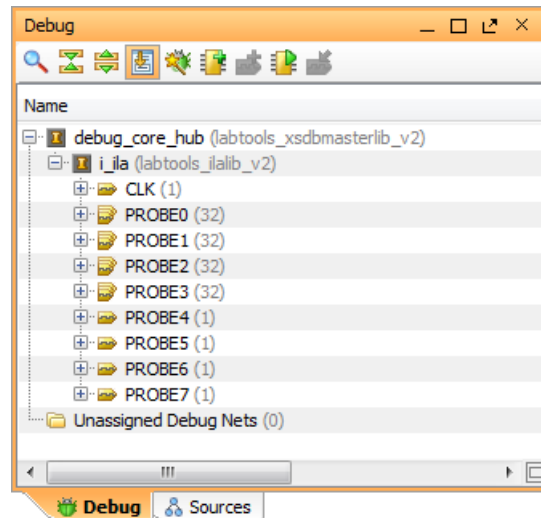


図 4-7 : debug\_core\_hub と ILA v2.0 コアが表示された [Debug] ビュー

## 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

debug\_core\_hub 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] ビューで debug\_core\_hub を選択し、[Instance Properties] ビューで [Debug Core Options] をクリックして、[C\_USER\_SCAN\_CHAIN] の値を変更します (図 4-8)。



**重要:** レガシ ICON、ILA、または VIO v1.x コアと ILA v2.0 コアを両方使用する場合は、debug\_core\_hub の C\_USER\_SCAN\_CHAIN プロパティを ICON v1.x コアのパウンダリ スキャン チェーン設定と競合しない値に設定する必要があります。そうでない場合、インプリメンテーションフローの後の方でエラーが発生します。

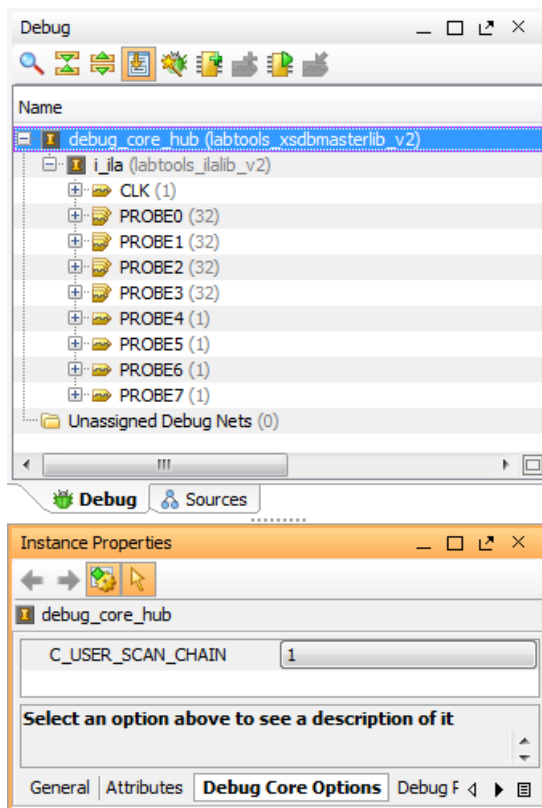


図 4-8 : debug\_core\_hub のユーザー スキャン チェーン プロパティの変更

# 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、`debug_core_hub` デバッグ コアは、最初ブラックボックスとして作成されます。これらのコアは、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。

## デバッグ コアのインプリメンテーション

デバッグ コアのインプリメンテーションは、デザインをインプリメントすると自動的に実行されますが、フロアプランまたはタイミング解析用に手動でインプリメントすることもできます。コアを手動でインプリメントするには、次のいずれかを実行します。

- [Debug] ビューのツールバーの [Implement Debug Cores] をクリックします。
- [Debug] ビューでデバッグ コアを右クリックし、[Implement Debug Cores] をクリックします。

各ブラック ボックス デバッグ コアが生成され、合成されます。この処理には、多少時間がかかる場合があります。この間、進捗状況を示すダイアログ ボックスが表示されます。デバッグ コアのインプリメンテーションが完了すると、デバッグ コアのブラック ボックスが処理され、生成されたインスタンスにアクセスできます。

## 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 IDE 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。デザインのさまざまなインプリメンテーション方法は、『Vivado Design Suite ユーザー ガイド：インプリメンテーション』(UG904) [参照 3] を参照してください。

# ハードウェアでのデザインのデバッグ

---

## 概要

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。デザインに含まれるデバッグ コアによって、次の 2 つのツールを使用できます。

- ChipScope™ Pro Analyzer : ICON v1.x、ILA v1.x、VIO v1.x、およびすべての IBERT デバッグ コアで使用
- Vivado™ ロジック解析機能 : ILA v2.0 デバッグ コアで使用

デザインに ICON/ILA/VIO v1.x と ILA v2.0 デバッグ コアの両方が含まれている場合は、ChipScope Pro Analyzer ツールと Vivado ロジック解析機能を同時に使用して、同じハードウェア ターゲット ボード上で動作している同じデザインをデバッグできます。詳細は、32 ページの「[Vivado ロジック解析を使用したデザインのデバッグ](#)」を参照してください。

---

## ChipScope Pro Analyzer を使用したデザインのデバッグ

ChipScope Pro Analyzer ツールは、デザインに含まれる ICON v1.x、ILA v1.x、VIO v1.x デバッグ コアにアクセスするために使用します。ChipScope Pro Analyzer ツールがインストールされている場合は、[Generate Bitstream] が実行されているインプリメント済みデザインに対して、Vivado IDE から直接起動できます。

ChipScope Pro Analyzer を起動するには、次のいずれかの操作を実行します。

- メイン メニューから [Flow ] → [Launch ChipScope Analyzer] をクリックします。
- Tcl コンソールから `launch_chipscope_analyzer` Tcl コマンドを実行します。

ビットストリーム ファイル (BIT) と CDC ネット接続名ファイルが、自動的に ChipScope Pro Analyzer で読み込まれます。ChipScope Pro Analyzer の詳細は、[http://japan.xilinx.com/support/documentation/dt\\_chipscopepro.htm](http://japan.xilinx.com/support/documentation/dt_chipscopepro.htm) を参照してください。

## 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、ChipScope Pro Analyzer ツールは、デザインに含まれる ILA v2.0 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 Session] 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 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 コアのトリガー条件とプローブ比較条件を設定します。
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。

## ハードウェア ターゲットに接続して FPGA 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、第 2 章の「[Vivado ハードウェア セッションを使用した FPGA デバイスのプログラム](#)」で説明されている手順と同じです。ILA v2.0 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] ビューにデバイスのスキャンで検出された ILA コアが表示されます (図 5-1)。

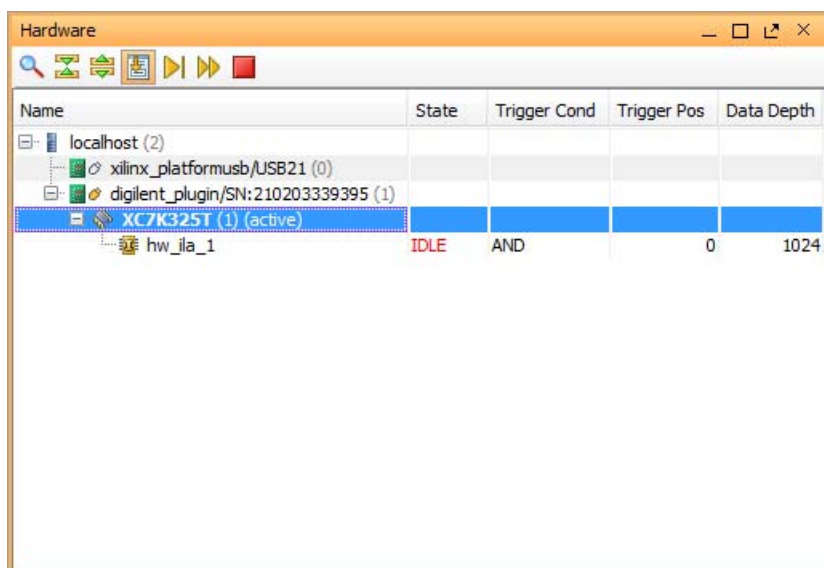


図 5-1 : ILA デバッグ コアが表示された [Hardware] ビュー



## 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] ビューのターゲット 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] ビュー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 v2.0 コアが含まれているかを確認してください。

ILA コア (図 5-1 では hw\_ila\_1) を選択すると、[ILA Core Properties] ビューにプロパティが表示されます。[Hardware] ビューでも、ILA コアのいくつかの設定を変更できます。

- トリガー条件
- トリガー位置
- データの深さ

## ILA コアのトリガー条件の設定

[Hardware] ビューの [Trigger Cond] または [ILA Core Properties] ビューの [Trigger Condition] プロパティを使用して、トリガー条件を [AND] または [OR] に設定します。[AND] に設定すると、すべての ILA プローブ比較が一致した場合にトリガー イベントが発生します。[OR] に設定すると、ILA プローブ比較のいずれかが一致した場合にトリガー イベントが発生します。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]
```

## ILA コアのトリガー位置の設定

[Hardware] ビューの [Trigger Pos] または [ILA Core Properties] ビューの [Trigger Position] プロパティを使用して、キャプチャ データ バッファのトリガー マークの位置を設定します。トリガー位置は、キャプチャ データ バッファ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 バッファ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 バッファの最初 (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 バッファの最後 (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 バッファ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]
```

## ILA コアデータの深さの設定

[Hardware] ビューの [Data Depth] または [ILA Core Properties] ビューの [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 バッファのデータ深さを設定します。データ深さは、1 から最大データ深さの間の 2 のべき乗に設定できます。

**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、[23 ページの「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]
```

---

## ILA プローブ情報の記述

[ILA Probes] ビューには、ILA v2.0 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。この情報はデザインから抽出され、通常は .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、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. Tcl コマンド write\_debug\_probes filename.ltx 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] ビューで FPGA デバイスを選択します。
2. [Hardware Device Properties] ビューの [Probes file] でファイルの場所を設定します。
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 プローブの表示

[ILA Probes] ビューには、各 ILA コアに関連付けられているプローブが表示されます。FPGA ハードウェア デバイスにプローブ ファイルを関連付けたら、ハードウェア デバイスを右クリックし、[Refresh Device] をクリックして、[ILA Probes] ビューを更新します (図 5-2)。

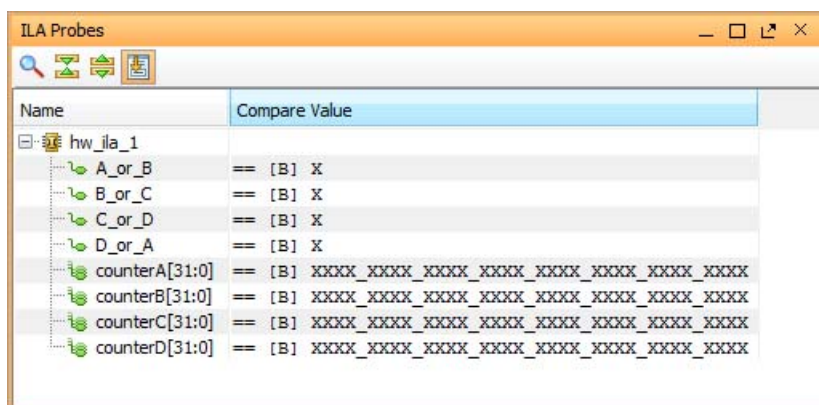


図 5-2 : [ILA Probes] ビュー

## ILA プローブのトリガー比較値の設定

ILA v2.0 コアのプローブ入力における等価条件または不等価条件の検出には、ILA 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 AND または OR で統合した結果です。詳細は、33 ページの「ILA コアのトリガー条件の設定」を参照してください。ILA プローブの比較値を指定するには、[ILA Probes] ビューで [Compare Value] セルを選択し、[Compare Value] ダイアログ ボックスを開きます (図 5-3)。

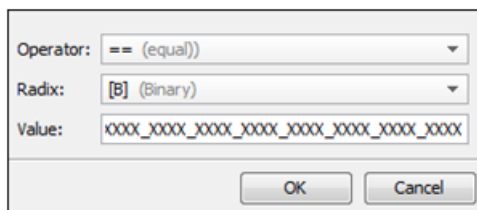


図 5-3 : ILA プローブの [Compare Value] ダイアログ ボックス

## ILA プローブの比較値の設定

[Compare Value] ダイアログ ボックスでは、次の 3 つのフィールドを設定できます。

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)
  - [A] ASCII
  - [U] (符号なし 10 進数)
  - [S] (符号付き 10 進数)
3. [Value]: ILA v2.0 デバッグ コアのプローブ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。
  - 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)
  - 16 進数
    - X: ドントケア
    - 0 ~ 9: 0 ~ 9 の値
    - A ~ F: 10 ~ 15 の値
  - 8 進数
    - X: ドントケア
    - 0 ~ 7: 0 ~ 7 の値
  - ASCII
    - ASCII 文字で構成される文字列
  - 符号なし 10 進数
    - 正の整数値
  - 符号付き 10 進数
    - 整数値

## ILA コアのトリガーの供給

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] : [Hardware] ビューで ILA コアを選択し、ツールバーの [Run Trigger] をクリックすると、ILA コアでトリガー条件およびプローブ比較値で定義されたトリガー イベントを検出できる状態になります。
- [Run Trigger Immediate] : [Hardware] ビューで 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 コアトリガー条件およびプローブ比較値の設定にかかわらず、すぐに ILA コアがトリガーされます。このコマンドは、ILA コアのプローブ入力にある任意の値をキャプチャする場合に有益です。

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 5-4)。

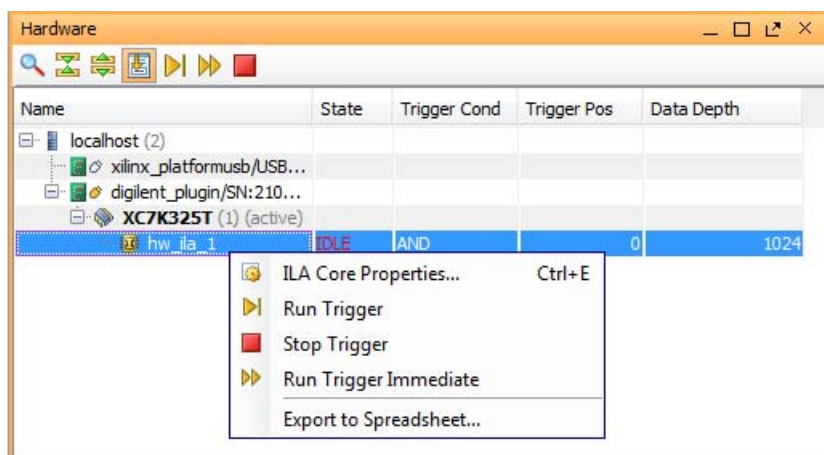


図 5-4 : ILA コアのトリガー コマンド

## ILA コアのトリガーの停止

ILA コアのトリガーを停止するには、[Hardware] ビューで ILA コアを選択し、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 (図 5-4)。

## ILA コアのステートの表示

[Hardware] ビューの [State] 列には、各 ILA コアのステートが示されます (表 5-1)。

表 5-1 : ILA コアのステート

ILA コアのステート	説明
IDLE	ILA コアはアイドル状態であり、トリガー待機状態になっていません。
ARMED	ILA コアはトリガー待機状態であり、トリガー条件が満たされるのを待機中です。トリガー位置 0 でトリガー待機状態にされたステートです。
CAPTURING	ILA コアはデータをデータ キャプチャ バッファに取り込み中です。トリガー位置が 0 より大きい値に設定されている場合、このステートはトリガー待機状態であることも表します。
FULL	ILA コア キャプチャ バッファがフルであり、ホストに表示するようアップロード中です。

## 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、第 6 章「[波形ウィンドウを使用した ILA プローブ データの表示](#)」を参照してください。

## 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 [upload_hw_ila_data hw_ila_1]
```

### 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]
```

## 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 の一部なので、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する必要があります。

- ラボ マシンに Vivado IDE をインストールして実行します。
- ラボ マシンに ISE ラボ ツール 14.2 をインストールし、ローカル マシン上の Vivado ロジック解析機能を使用して CSE サーバーのリモート インスタンスに接続します。

### ラボ マシンに Vivado IDE をインストールして実行

ラボ マシンに Vivado IDE をインストールする際の要件は、『ザイリンクス デザイン ツール：インストールおよびライセンス ガイド』(UG798) [参照 4] を参照してください。

Vivado ロジック解析機能のグラフィカル ユーザー インターフェイスはプロジェクト モードでのみ使用可能ですが、デザインを作成するのに使用した元のプロジェクトは必要ありません。Vivado ロジック解析機能で元のプロジェクトから必要なのは、ビットストリーム プログラム ファイル (.bit) とプローブ ファイル (.ltx) の 2 つのみです。ラボ マシンで Vivado ロジック解析機能を使用するには、次の手順に従います。

1. ラボ マシンに Vivado IDE をインストールします。
2. ビットストリーム プログラム ファイル (.bit) とプローブ ファイル (.ltx) をラボ マシンにコピーします。
3. Vivado IDE を GUI モードで起動します。
4. ソース、IP、または制約ファイルを含まないサンプルプロジェクトを作成します。
5. Flow Navigator で [Open Hardware Session] をクリックします。
6. 「ハードウェア ターゲットに接続して FPGA 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。ラボ マシンにコピーしたビットストリーム プログラム ファイル (.bit) でターゲット FPGA デバイスをプログラムします。
7. 「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。「ILA プローブ情報の読み出し」の手順では、ラボ マシンにコピーしたプローブ ファイル (.ltx) を使用します。

### ラボ マシンで動作中のリモート CSE サーバーへの接続

ラボ マシンにネットワークで接続されている場合、リモート ラボ マシン上で動作中の CSE 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 CSE サーバーに接続するには、次の手順に従います。

1. ラボ マシンに ISE ラボ ツール 14.2 をインストールします。
2. リモート ラボ マシンで cse\_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、ISE ラボ ツール 14.2 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\14.2\LabTools\LabTools\bin\nt64\cse_server -port 50001
```
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 FPGA 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 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。

# Vivado ロジック解析機能と ChipScope Pro Analyzer の同時使用

Vivado ロジック解析機能と ChipScope Pro Analyzer ツールを同時に使用して、同じターゲット ボード上の同じデザインをデバッグできます。この使用法は、次のような場合に重要です。

- デザインに ILA v2.0 デバッグ コアと VIO v1.x デバッグ コアが含まれており、それぞれに Vivado ロジック解析機能と ChipScope Pro Analyzer を使用してアクセスする必要がある場合
- デザインに含まれる ILA v2.0 デバッグ コアに Vivado ロジック解析機能を使用してアクセスし、ChipScope Pro Analyzer ツールのシステム モニターを使用して XADC の温度または電圧センサーを監視する場合
- 7 シリーズ デバイスと 6 シリーズ デバイスを、それぞれに Vivado ロジック解析機能と ChipScope Pro Analyzer を使用して同時にデバッグする必要がある場合

このセクションでは、デザインに ILA v2.0 デバッグ コアと VIO v1.x デバッグ コアが含まれている最初のケースについて説明します。この機能を活用するには、次のようにする必要があります。

1. 各 JTAG コントローラー コアに個別の BSCAN ユーザー スキャン チェーンを使用します。
2. 各ランタイム アプリケーションに対して個別の CSE サーバーを起動します。
3. Vivado ロジック解析機能と ChipScope Pro Analyzer ツールを使用します。

## 個別の BSCAN ユーザー スキャン チェーンの使用

ILA v2.0 コアを BSCANE2 プリミティブに接続する debug\_core\_hub コアと、VIO v1.x コアを BSCANE2 コアに接続する ICON v1.x コアが、異なるユーザー スキャン チェーンを使用してコンフィギュレーションされていることを確認します。詳細は、[第 4 章の「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。

## 個別の CSE サーバーの設定

ターゲット ボードに接続されているマシンで、2 つの CSE サーバー インスタンスが動作していることを確認します。Windows プラットフォームでは、2 つの cmd ウィンドウで次を実行します。

- 最初の cmd ウィンドウで「cse\_server -port 50001」を実行します。
- 2 つ目の cmd ウィンドウで「cse\_server -port 50002」を実行します。

ポート 50001 上の CSE サーバーは Vivado ロジック解析機能で、ポート 50002 上の CSE サーバーは ChipScope Pro Analyzer ツールで使用されます。

## Vivado ロジック解析機能と ChipScope Pro Analyzer ツールの実行

デザインと CSE サーバーを設定したら、次の手順に従って Vivado ロジック解析機能と ChipScope Pro Analyzer ツールを使用してデザインをデバッグします。

1. Vivado IDE を GUI モードで起動します。
2. Flow Navigator で [Open Hardware Session] をクリックします。
3. localhost:50001 上の CSE サーバーに接続し、ターゲット デバイスをプログラムします。詳細は、[第 2 章の「Vivado ハードウェア セッションを使用した FPGA デバイスのプログラム」](#)を参照してください。
4. [Flow] → [Launch ChipScope Analyzer] をクリックして ChipScope Pro Analyzer を起動します。



- ChipScope Pro Analyzer ツールで [JTAG Chain] → [Server Host Setting] をクリックします。サーバーを localhost:50002 に変更します (図 5-5)。

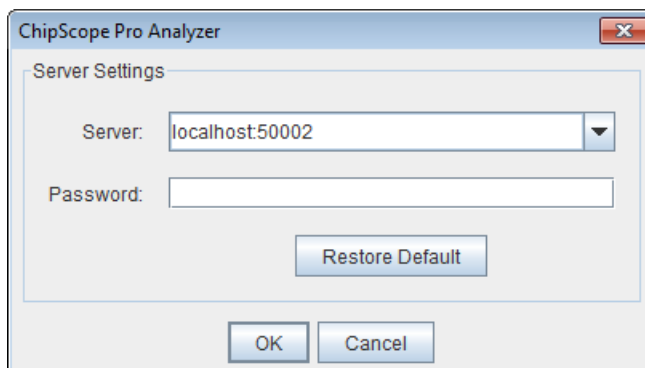


図 5-5 : ChipScope Pro Analyzer ツールでサーバー ホスト設定を変更

- ChipScope Pro Analyzer で、[JTAG Chain] メニューの適切なケーブルを使用してターゲット ボードに接続します。
- 各解析ツールを使用して、それぞれのデバッグ コアにアクセスします (図 5-6)。

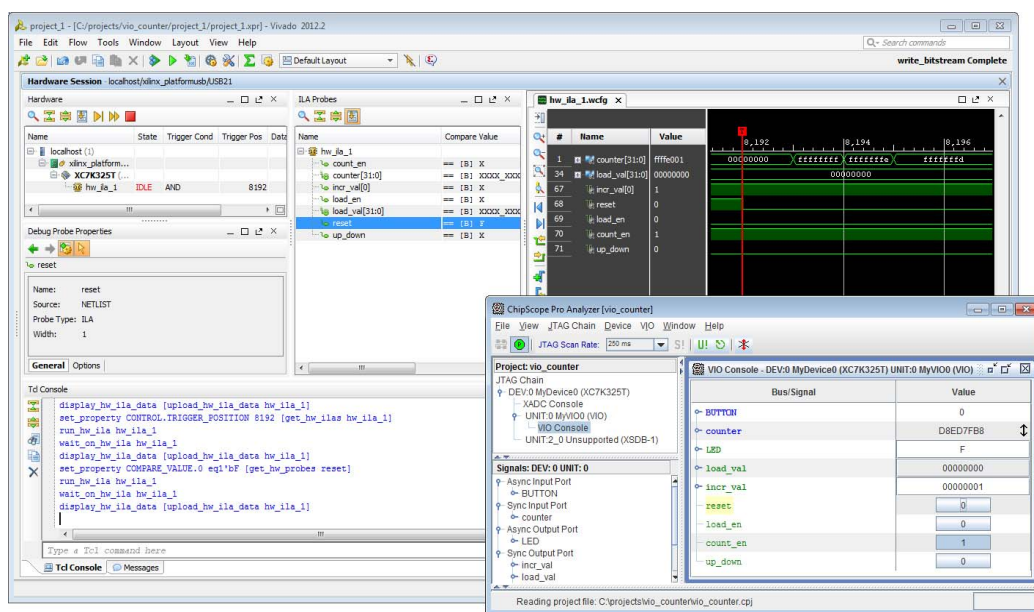


図 5-6 : Vivado ロジック解析機能と ChipScope Pro Analyzer ツールを使用してデザインをデバッグ

# ハードウェア セッションの 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、階層第一級 Tcl オブジェクトがあります (表 5-2)。

表 5-2: ハードウェア セッションの Tcl オブジェクト

Tcl オブジェクト	説明
hw_server	CSE 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 オブジェクトには、1 つの hw_ila_data オブジェクトのみを関連付けることができます。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 コアのプロープ入力を参照するオブジェクト。

ハードウェア セッション コマンドの詳細は、Tcl コンソールで「help -category hardware」と入力してください。

## hw\_server の Tcl コマンド

表 5-3 に、ハードウェア サーバー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 5-3: hw\_server の Tcl コマンド

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	CSE 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

表 5-4 に、ハードウェア ターゲット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 5-4 : 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

表 5-5 に、ハードウェア デバイスにアクセスするために使用する hw\_device の Tcl コマンドを示します。

表 5-5 : 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_device	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

表 5-6 に、ILA v2.0 デバッグ コアにアクセスするために使用する hw\_ila の Tcl コマンドを示します。

表 5-6 : 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

表 5-7 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 hw\_ila\_data の Tcl コマンドを示します。

表 5-7 : 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。
get_hw_ila_data	hw_ila_data オブジェクトのリストを取得します。
read_hw_ila_data	ファイルから hw_ila_data を読み込みます。
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

表 5-8 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 5-8 : hw\_probe の Tcl コマンド

Tcl コマンド	説明
get_hw_probes	ハードウェア プロブのリストを取得します。

## ハードウェア セッションの 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:50001 上の 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)
- デザインに ILA コアが 1 つ含まれており、KC705 ボード上の XC7K325T デバイスで実行
- ILA コアに counter[3:0] というプローブが含まれる

### Tcl コマンド スクリプト例

```
# Connect to the Digilent Cable on localhost:50001
connect_hw_server -host localhost -port 50001
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]
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]
```

# 波形ウィンドウを使用した ILA プローブデータの表示

---

## 概要

Vivado™ Integrated Design Environment (IDE) シミュレータを開くと、波形ウィンドウを使用してデザインを解析し、コードをデバッグできます。Vivado ロジック解析では、ハードウェアや ILA プローブのビューなどでもデザイン データを確認できます。

---

## 波形ウィンドウの機能および制限事項

### 波形機能のサマリ

- 2 以上の幅の ILA プローブ ベクターでアナログ波形をサポート
- 実数の基数をデジタルまたはアナログ波形に適用

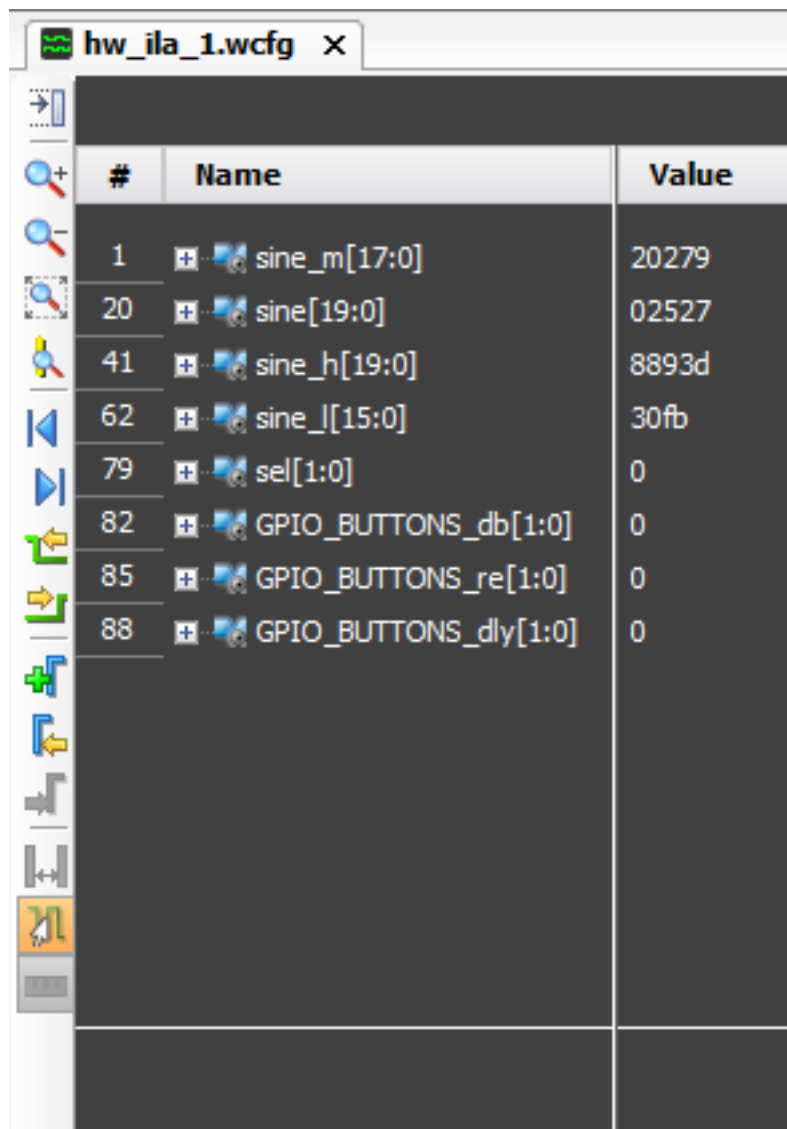
### 波形の制限事項

- 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の場合は、間違った値になる可能性があります。
- 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。
- アナログとデジタルの波形を切り替えると、行の高さがデフォルト値に戻ります (アナログでは 100 ピクセル、デジタルでは 20 ピクセル)。
- 64 ビットに近い広い配列で実数以外の基数が使用されていると、最下位ビット (LSB) での丸めの問題が発生する可能性があります。

## 波形コンフィギュレーションの信号およびバス

波形ウィンドウのスカラーおよびベクターの ILA プローブは、波形のデザイン オブジェクトです。

ILA プローブの横には、そのタイプを示すアイコンが表示されます。アイコン上にマウス ポインターを置くと、詳細が表示されます。図 6-1 に、波形コンフィギュレーションの ILA プローブの例を示します。











#	Name	Value
1	 sine_m[17:0]	20279
20	 sine[19:0]	02527
41	 sine_h[19:0]	8893d
62	 sine_l[15:0]	30fb
79	 sel[1:0]	0
82	 GPIO_BUTTONS_db[1:0]	0
85	 GPIO_BUTTONS_re[1:0]	0
88	 GPIO_BUTTONS_dly[1:0]	0

図 6-1 : 波形の ILA プローブ

ILA プローブの ID、名前、値が表示されています。ツールバー ボタンをクリックすると、次のセクションで説明するナビゲーション機能を使用できます。


## ズーム機能の使用

波形コンフィギュレーションのズーム機能には、ツールバー ボタンを使用します。

波形をクリックして Ctrl キーを押しながらマウス ホイールを使用すると、オシロスコープのダイヤル操作のようにズーム表示をすることもできます。



## [Waveform Options] スライドアウト

[Waveforms Options] ボタン  をクリックすると、[Waveforms Options] スライドアウトが開きます (図 6-2)。

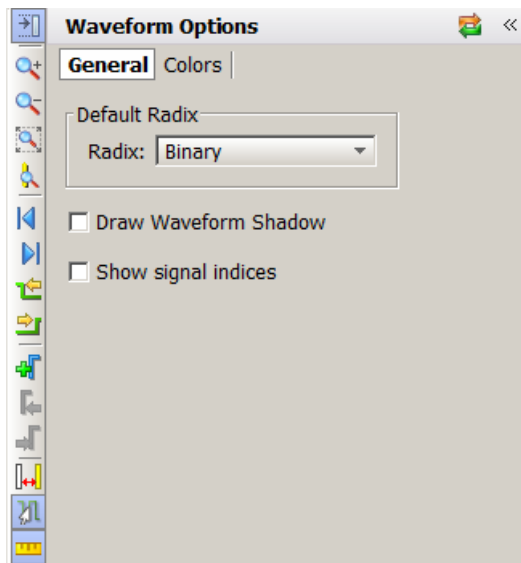


図 6-2 : [Waveform Options] スライドアウト

設定できるオプションは次のとおりです。

- [General] : デフォルトの基数を設定します。
- [Show signal indices] : このチェックボックスをオンにすると、信号番号の間に短い定義文が表示されます。
- [Colors] : 波形内のオブジェクトの色を設定します。

---

## 波形コンフィギュレーションの ILA プローブ

ILA プローブのスカラー (信号) およびベクター (バス) を波形コンフィギュレーション ファイルに追加し、そのコンフィギュレーションを WDB ファイルに保存できます。ILA コアからのプローブを波形ウィンドウに表示するには、メニュー コマンドを使用するか、または Tcl コンソールで Tcl コマンドを使用します。

波形コンフィギュレーションに ILA プローブを追加するには、次の手順に従います。

1. [ILA Probes] ビューで ILA コアを展開し、プローブを選択します。
2. 右クリックして [Add to Wave] をクリックします。

波形を比較するため、同じ信号またはバスのコピーを波形コンフィギュレーションに追加できます。同じ信号またはバスのコピーは、グループや仮想バスなど、波形コンフィギュレーションの任意の位置に配置できます。



信号またはバスのコピーを追加するには、次の手順に従います。

1. 波形コンフィギュレーションで信号またはバスを選択します。
2. [Edit] → [Copy] をクリックするか、または Ctrl + C キーを押します。

信号名がクリップボードにコピーされます。

3. [Paste] をクリックするか、Ctrl + V キーを押します。

これで信号またはバスが波形コンフィギュレーションにコピーされます。ドラッグ アンド ドロップでこの信号やバスを移動させることができます。

## 波形コンフィギュレーションのカスタマイズ

表 6-1 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 6-1: 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	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	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。
名前の表示	完全な階層名、信号またはバス名のみ、または各信号のカスタム名を表示できます。
基数	デフォルト基数は、波形コンフィギュレーション、[Objects] ビュー、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。

## 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、50 ページの「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 つ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。


## 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した距離 (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。

- メイン カーソルの位置に波形コンフィギュレーションにマーカーを追加します。
  - a. 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。
  - b. [Edit] → [Add Marker] をクリックするか、または [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、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 バッファのトリガー 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 (第 5 章の「ILA コアのトリ

[ガー位置の設定](#)」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は移動させることはできません。その位置は ILA コアのトリガー位置プロパティ設定で設定されています。

## 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. 波形ウィンドウの [Name] 列で、下に仕切りを追加する信号をクリックします。
2. [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。新しい仕切り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。

仕切りの名前も変更できます。詳細は、[52 ページの「オブジェクト名の変更」](#)を参照してください。

## グループ

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。グループは、展開表示したり、閉じたりできます。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、[52 ページの「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。

## 仮想バス

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、仮想バス 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、その中に含まれるものをグループ解除するには、仮想バスを選択し、[Edit] → [Wave Objects] → [Ungroup] をクリックするか、または右クリックしてポップアップ メニューの [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。

---

## オブジェクト名の変更

波形ウィンドウの信号、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 コアのプロープ入力に接続されているネット名には影響しません。

## 名前の表示

名前は、完全な階層名 ([Long Name])、信号またはバス名のみ ([Short Name])、またはカスタム名で表示できます。波形ウィンドウの信号またはバス名は、波形コンフィギュレーションの [Name] 列に表示されます。名前が非表示になっている場合は、次の操作を実行します。

- 信号名全体が表示されるよう [Name] 列の幅を調整します。
- [Name] 列のスクロールバーを使用します。

表示名を変更するには、次の手順に従います。

1. 信号名またはバス名を 1 つ以上選択します。複数の信号を選択する場合は、Shift キーまたは Ctrl キーを押しながらクリックします。
2. 右クリックし、[Name] をクリックし、次のいずれかを選択します。
  - [Long]: 完全な階層名を表示します。
  - [Short]: 信号名またはバス名のみを表示します。
  - [Custom]: 信号のカスタム名を表示します。

名前の表示変更はすぐに反映されます。

## 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、[54 ページの「基数およびアナログ波形」](#)を参照してください。

波形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. バスを右クリックします。
2. [Radix] をクリックし、ドロップダウンメニューからフォーマットを選択します。
  - [Binary] (2 進数)
  - [Hexadecimal] (16 進数)
  - [Unsigned Decimal] (符号なし 10 進数)
  - [Signed Decimal] (符号付き 10 進数)
  - [Octal] (8 進数)
  - [ASCII]



**重要:** [Objects] ビューで基数を変更しても、波形ウィンドウまたは [Tel Console] の値は変更されません。波形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、波形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の場合には不正な値になる可能性があります。
- 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。

## フロート ルーラーの使用

波形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 ベースを使用して時間を計測するには、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、波形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、こ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
  - 。 マーカーを選択
2. [View] → [Floating Ruler] をクリックするか、または [Floating Ruler] ボタンをクリックします。



この操作は最初に 1 回だけ実行し、繰り返す必要はありません。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。

---

## 基数およびアナログ波形

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。
- 実数基数を使用すると、バスの値は固定小数点または浮動小数点の実数として処理されます。これは、[55 ページの図 6-3](#) に示す [Real Settings] ダイアログ ボックスの設定によって決まります。

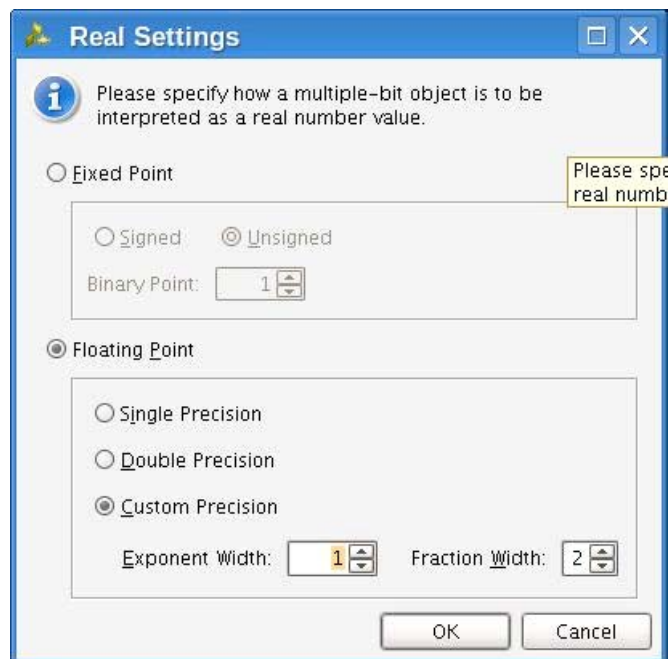


図 6-3 : [Real Settings] ダイアログ ボックス

設定できるオプションは次のとおりです。

- [Fixed Point] : 選択したバス波形オブジェクトのビットが固定小数点の符号付きまたは符号なしの実数として処理されます。
- [Binary Point] : 2 進数小数点の右側のビット数を指定します。[Binary Point] で指定する値が波形オブジェクトのビット幅よりも大きい場合、波形オブジェクトの値は固定小数点としては処理されず、波形オブジェクトがデジタル波形で表示されたときにすべての値が <Bad Radix> と表示されます。アナログ波形として表示される場合、すべての値は 0 として処理されます。
- [Floating Point] : 選択したバス波形オブジェクトのビットが IEEE 浮動小数点の実数として処理されます。

注記 : 単精度および倍精度 (および単/倍精度に設定されている値のカスタム精度) のみがサポートされています。

その他の値は、[Fixed Point] を使用した場合と同様 <Bad Radix> 値になります。[Exponent Width] および [Fraction Width] は、波形オブジェクトのビット幅に必ず追加する必要があり、追加されない場合は <Bad Radix> 値になります。

## 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. [Wave] ビューの [Name] 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。

図 6-4 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 6-4 : [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。波形エリアの上下の範囲外にあるものは、範囲内に戻るまでは非表示になります。
  - [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] : 波形エリアの境界線を越えていて、ほかの波形と重なっていても、波形ウィンドウの境界に達するまでは波形の値がどこにあっても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がオンの場合、グリッドラインが指定した Y 軸の位置で描画されます (値が波形の Y 軸の範囲内にある場合)。

[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。

## 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、図 6-5 に示す追加のズーム機能があります。

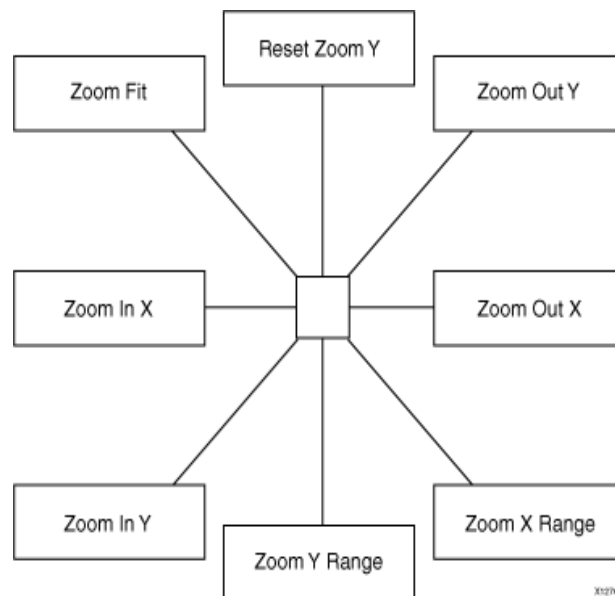


図 6-5: アナログズームのオプション

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]: 開始点からマウスボタンを放した位置までの距離により、2 のべき乗分 Y 軸方向にズームアウトします。開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]: 縦方向にラインを描き、マウスボタンを離した位置までの Y 軸の範囲を表示します。
- [Zoom In Y]: 開始点からマウスボタンを放した位置までの距離により、2 のべき乗分 Y 軸方向にズームインします。開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]: 波形ウィンドウに現在表示されている値に Y の範囲をリセットし、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。

# 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 コンフィギュレーション設定の説明

次の表に、Vivado™ ロジック解析機能で使用可能なデバイス コンフィギュレーション設定を示します。

表 A-1: デバイス コンフィギュレーション設定

設定	デフォルト値	有効な値	説明
BITSTREAM.COFIG.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.COFIG.BPI_PAGE_SIZE	1	1、4、8	BPI コンフィギュレーションで、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。 <ul style="list-style-type: none"> <li>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。</li> <li>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。</li> <li>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 ファミリをサポートする設定を使用します。</li> </ul>
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。
BITSTREAM.CONFIG.CONFIGRATE	3	3、6、9、12、16、22、26、33、40、50、66	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.DCIUPDATESMODE	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。

表 A-1: デバイス コンフィギュレーション設定 (続き)

設定	デフォルト値	有効な値	説明
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. EXTMASTERCLK_EN	Disable	Disable、 div-8、div-4、 div-2、div-1	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 USERCLK ピンに接続する必要があります。
BITSTREAM.CONFIG. INITPIN	Pullup	Pullup、 Pullnone	INIT ピンにプルアップ抵抗を追加するか、フロートしたままにするかを指定します。
BITSTREAM.CONFIG. INITSIGNALSError	Enable	Enable、 Disable	Enabled に設定すると、CFG エラーが検出された場合に init_b ピンが 0 に設定されません。
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	MultiBoot 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、General1 および General2 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. OVERTEMPPowerDown	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、 CTLReg、 X1、X8、 X16、X32、 SPI1、SPI2、 SPI4、BPI8、 BPI	SelectMAP モード ピンをユーザー I/O として使用できないようにします。SelectMAP モードと関連のピンについては、データシートを参照してください。このオプションは、リードバックおよびパーシャル リコンフィギュレーションに SelectMAP コンフィギュレーション ピンを使用する場合に必要で、SelectMAP またはシリアル モードを使用している場合に使用します。このオプションの設定は SelectMAP ピンのみに適用されますが、コンフィギュレーション後に JTAG 以外のコンフィギュレーションピンにアクセスする際にも使用します。
BITSTREAM.CONFIG. PROGPIN	Pullup	Pullup、 Pullnone	ProgPin 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. 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。

表 A-1: デバイス コンフィギュレーション設定 (続き)

設定	デフォルト値	有効な値	説明
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。
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、256Mb 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	1	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	TdoPin 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 タイマーの値を設定します。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	ユーザー モードでのウォッチドッグ タイマーの値を設定します。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	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。

表 A-1: デバイス コンフィギュレーション設定 (続き)

設定	デフォルト値	有効な値	説明
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 hhh mmmmm sssss (dddd = 日、MM = 月、yy = 年 (2000 年は 0000)、hhh = 時、mmmm = 分、ssss = 秒) です。このレジスタの内容は、FPGA ファブリックにより 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	bbram	bbram、efuse	使用する AES 暗号化キーの場所を、バックアップ機能付き RAM (BBRAM) または eFUSE レジスタのいずれかに指定します。  注記: 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	Pick	Pick、 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、妨害、コピーに AES と HMAC キーの両方が必要です。  pick に設定すると、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. KEY0	Pick	Pick、 16 進文字列	ビットストリーム暗号化の AES 暗号化キーを設定します。pick に設定すると、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. 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. STARTCBC	Pick	Pick、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。pick に設定すると、ランダムな値が選択されます。
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、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。

表 A-1: デバイス コンフィギュレーション設定 (続き)

設定	デフォルト値	有効な値	説明
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。 <ul style="list-style-type: none"> <li>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。</li> <li>各フレームを個別に読み込みます。</li> <li>各フレーム後に巡回冗長検査 (CRC) を実行します。</li> <li>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。</li> </ul>
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.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 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. 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。  注記: 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、パーシャル リコンフィギュレーション中も XADC が継続して機能します。Enable に設定すると、パーシャル リコンフィギュレーション中は 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 エッジが到着してから、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。



表 A-1: デバイス コンフィギュレーション設定 (続き)

設定	デフォルト値	有効な値	説明
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、BRAM 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。
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 の一致は、BitGen で設定された 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。
BITSTREAM.STARTUP. STARTUPCLK	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。 <ul style="list-style-type: none"> <li>• Cclk : FPGA デバイスで供給される内部クロックに同期します。</li> <li>• 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。</li> <li>• JtagClk : JTAG で供給されるクロックに同期します。このクロックは、JTAG の制御ロジックとして使用される TAP コントローラーをシーケンスします。</li> </ul>

# その他のリソース

---

## ザイリンクス リソース

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

<http://japan.xilinx.com/support>

ザイリンクス資料で使用する用語集については、次を参照してください。

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

---

## ソリューション センター

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

---

## 参考資料

次の資料は、本書を補足するためのものです。

Vivado Design Suite 2012.2 の資料

[http://japan.origin.xilinx.com/support/documentation/dt\\_vivado2012-2.htm](http://japan.origin.xilinx.com/support/documentation/dt_vivado2012-2.htm)

1. 『Vivado Design Suite ユーザー ガイド : ロジック シミュレーション』(UG937)
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 : インプリメンテーション』(UG904)
4. 『Vivado Design Suite インストールおよびライセンス ガイド』(UG798)

ISE Design Suite 14.2 の資料

5. iMPACT ヘルプ :  
[http://japan.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_2/isehelp\\_start.htm#pim\\_c\\_overview](http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_2/isehelp_start.htm#pim_c_overview)

LogiCORE IP ChipScope Pro Integrated Logic Analyzer (ILA) (v2) データシート』(DS875) :

[http://japan.xilinx.com/support/documentation/ipembedprocess\\_debugtrace\\_ila.htm](http://japan.xilinx.com/support/documentation/ipembedprocess_debugtrace_ila.htm)