ChipScope Pro 13.3 ソフトウェアおよびコア

ユーザー ガイド

UG029 (v13.3) 2011 年 10 月 19 日

XILINX®

本資料は英語版 (v13.3) を翻訳したもので、内容に相違が生じる場合には原文を優先します。
資料によっては英語版の更新に対応していないものがあります。
日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

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

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	改訂内容
2010 年 4 月 19 日	12.1	• 12.1 ツールと互換性を持たせるためすべての章を更新 • Virtex-5 FPGA GTX トランシーバー用 IBERT v2.0 を追加 • JTAG プラグインを開くための Analyzer のサポートを追加 • ByteTools 社 Catapult EJ-1 イーサネット – JTAG接続ケーブルのサポートを追加 • 第 4 章に「トリガー実行モード」(単一および反復) を追加 • 第 4 章に「トリガーおよびキャプチャ ステータス」を追加 • csejtag_target is_connected コマンドの追加 • csefpga_configure_device_with_file コマンドを追加 • csefpga_is_configured コマンドの追加
2010 年 9 月 21 日	12.3	ISE 12.3 リリース用に改訂

日付	バージョン	改訂内容
2011 年 3 月 1 日	13.1	• ロジック デバッグに 7 シリーズのサポートを追加 • IBA/PLB (IBA/PLB46 ではない) を削除 • IBA/OPB を削除 • IBERT V4 GT11 を削除 • スタートアップ トリガー モードを追加 • ChipScope Pro Analyzer IBERT スイープ テスト プロットを追加 • スタンドアロン IBERT プロット ビューアーを追加 • GTH トランシーバーの 1/2、1/4、1/8 ライン レート サポートを追加 • ICON、ILA、VIO、および ATC2 を追加 • MARK_DEBUG を PlanAhead ユーザー ガイドに追加 • CSE/Tcl セクションに新しいコマンドの説明を追加
2011 年 7 月 6 日	13.2	• Kintex™-7 FPGA デバイスのコアおよびトランシーバーのサポート情報を追加 • Digilent 社製 JTAG-SMT1 および JTAG-HS1 USB-to-JTAG ダウンロード ケーブルの情報を第一章の「通信要件」に追加 • 全体的にマイナー改訂
2011 年 10 月 19 日	13.3	• 第 2 章 : コアの生成に関するセクションを削除し、データシートに含まれる同コンテンツへのリンクを提供。それに伴い文書中のリファレンスをアップデート • 第 3 章 :「ILA コア キャプチャ パラメーターの設定」セクションの内容を大幅に向上 • 全体的に若干の修正を加え内容を明確化

目次

XILINX®

第 1 章

概要

ChipScope Pro ツールの概要

FPGA デバイスの集積度が高くなるにつれて、テスト対象デバイスにテスト装置プローブを接続することが実用的ではなくなってきています。ChipScope Pro ツールは、ISE® Design Suite 製品表 [211 ページのリファレンス 16 を参照] にリストされているザイリンクス FPGA デバイスに含まれるターゲット デザインに主要なロジック アナライザーおよびテスト/計測ハードウェア コンポーネントを統合します。ChipScope Pro ツールは、これらのコンポーネントと通信してロジック解析を提供します。

ChipScope Pro シリアル I/O ツールキットでは、ザイリンクス FPGA の高速シリアル トランシーバーの I/O 機能を使用してデザインのエラボレーションとデバッグを実行する機能が提供されています。IBERT (Internal Bit Error Ratio Tester) コアおよび関連するソフトウェアでは、高速シリアル トランシーバーへのアクセスを提供し、これらのトランシーバーで構成されたチャネルでのビット エラー率の解析を実行します。このマニュアルでは、トランシーバーを MGT (マルチギガビット トランシーバー) と呼びます。IBERT コアでは、ISE Design Suite 製品表 [211 ページのリファレンス 16 を参照] にリストされているザイリンクス Kintex™-7、Virtex®-5、Virtex-6、および Spartan®-6 FPGA デバイスの高速シリアル トランシーバー がサポートされています。

ChipScope Pro ツールの説明

次の表に、各種 ChipScope Pro ソフトウェア ツールおよびコアの簡単な説明を示します。

表 1-1 : ChipScope Pro ツールの説明

ツール	説明
ザイリンクス CORE Generator™ ツール	サポートされるすべての FPGA デバイス ファミリをターゲットにして ICON (Integrated Controller)、ILA (Integrated Logic Analyzer)、VIO (Virtual Input/Output)、および ATC2 (Agilent Trace Core) コアを生成できます。また、Kintex-7、Virtex-5、Virtex-6、および Spartan-6 FPGA ファミリをターゲットにして IBERT v2.0 コアを生成することもできます。ザイリンクス CORE Generator は、ザイリンクス ISE Design Suite ソフトウェア ツールに含まれています。
IBERT Core Generator	Virtex-5 デバイスをターゲットにして IBERT v1.0 コアのデザインを完全に生成できます。IBERT Core Generator では、ユーザーが選択した MGT およびデザインを制御するパラメーターに基づいて、ISE Design Suite でコンフィギュレーション ファイルを生成します。
ChipScope Pro Core Inserter ツール	合成されたユーザー デザインに ICON、ILA、ATC2 コアを自動的に挿入します。

表 1-1 : ChipScope Pro ツールの説明 (続き)

ツール	説明
PlanAhead™ デザイン解析ツール	デザインのネットリストに ICON および ILA コアを自動的に挿入します。この機能の詳細は、PlanAhead デザイン解析ツール [211 ページのリファレンス 17 を参照] を参照してください。
ChipScope Pro Analyzer ツール	ICON、ILA、VIO、および IBERT コアのインシステム デバイス コンフィギュレーション、トリガー設定、トレース表示、制御、およびステータスを提供します。
ChipScope Engine Tcl (CSE/Tcl) スクリプト インターフェイス	CSE/Tcl スクリプト コマンド インターフェイスによって、Tcl シェルから JTAG (Joint Test Action Group、IEEE 規格) チェーン内のデバイスとの通信が可能になります(1)。

注記 :

1. Tcl は Tool Command Language の略です。CSE/Tcl インターフェイスでは、ChipScope Pro および ISE ツールまたは ActiveState [212 ページのリファレンス 24 を参照] の ActiveTcl 8.4 シェルに含まれている xtclsh と呼ばれる Tcl シェル プログラムが必要です。

次に、ChipScope Pro ツールを使用して追加したデバッグ コアを含むシステムのブロック図を示します。CORE Generator ツールを使用してコアを生成し、それらを HDL ソース コードにインスタンシエートすることによって、デザインに ICON、ILA、VIO、および ATC2 コア (総称 ChipScope Pro コア) を配置できます。また、ChipScope Pro Core Inserter または PlanAhead ツールを使用すると、ICON、ILA、および ATC2 コアを合成済みデザインのネットリストに直接挿入できます。デザインは、ISE インプリメンテーションツールを使用して配置配線されます。次に、デバイスにビットストリームをダウンロードして ChipScope Pro Analyzer ツールでデザインを解析します。

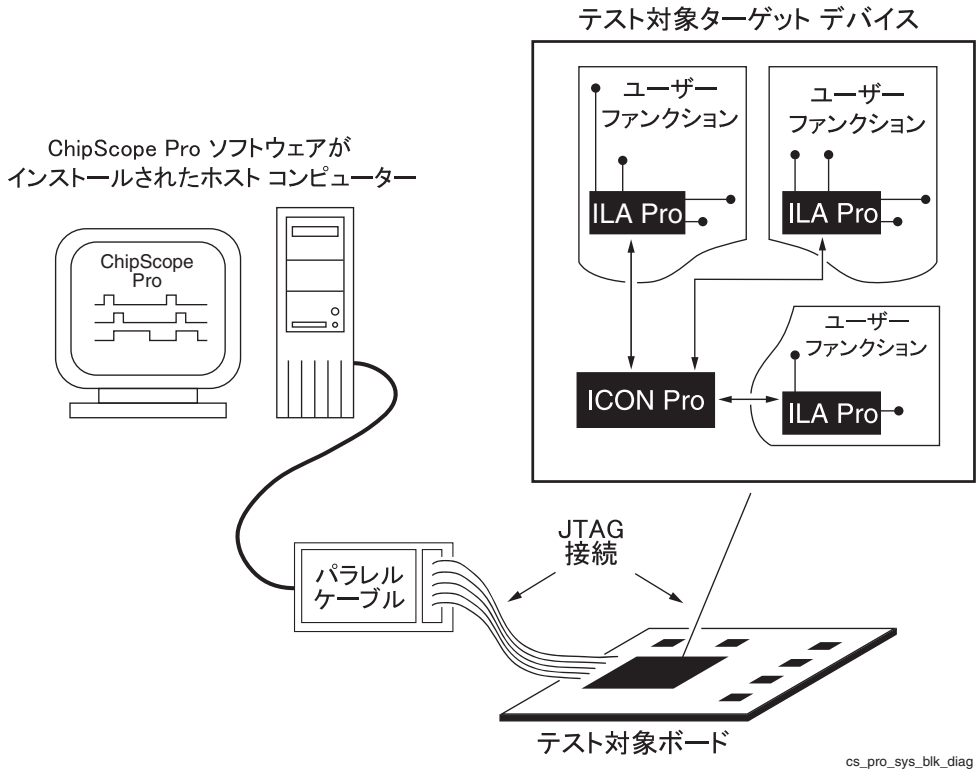

図 1-1 : ChipScope Pro システムのブロック図

ChipScope Pro Analyzer では、コンピューターと JTAG バウンダリ スキャン チェーン内のデバイス間通信に、次のダウンロード ケーブルを使用できます。

- プラットフォーム ケーブル USB
- パラレル ケーブル IV
- Digilent 社製 USB-to-JTAG ケーブル
- ByteTools 社製 Catapult EJ-1 イーサネット-JTAG 接続ケーブル

ChipScope Pro Analyzer には、ロジックを検証する多数の機能が含まれています (表 1-2)。1 ～ 4,096 までのデータ チャネル、256 ～ 131,072 までのサンプル バッファー ワード数を選択可能です。また、ユーザー ロジックに影響を与えずに即座にトリガーを変更できます。ChipScope Pro Analyzer では、トリガー変更からキャプチャしたデータの解析までのプロセスを順番に実行できます。

表 1-2 : ChipScope Pro のロジック デバッグ機能および利点

機能	利点
1 ～ 4,096 までのデータ チャネルを選択可能	広範囲のデータ バスの動作を正確にキャプチャします。
256 ～ 131,072 までのサンプル バッファー ワード数を選択可能	サンプルするワード数を増やすと精度が高くなり、不定期に発生するイベントをキャプチャする確率が上がります。
最大 16 個のトリガー ポートを使用でき、それぞれに対して 1 ～ 256 までのチャネルを選択可能 (合計 4096 チャネルまで)	複数のトリガー ポートを個別に設定できるため、イベント検出の柔軟性が高くなり、必要になるサンプル ストレージが減少します。
各トリガー ポートに最大 16 個までの比較ユニットを使用でき、トリガー条件ごとに合計で 16 の異なる比較を実行可能	トリガー ポートごとに複数の比較ユニットがあり、有用なリソースを節約しながら、イベント検出の柔軟性を高めます。
すべてのデータおよびトリガー処理は、最大 500MHz のユーザー クロックに同期	トリガー イベント検出およびデータ キャプチャを高速で実行できます。
トリガー条件によりブール式または最大 16 個の比較演算子のトリガー シーケンスをインプリメント	ブール式または 16 レベルのトリガー シーケンサーを使用する最大 16 個のトリガー ポートの比較演算子を組み合わせることができます。
データ ストレージ必要条件で最大 16 個の比較演算子のブール式をインプリメント	ブール式を使用する最大 16 個のトリガー ポートの比較演算子を組み合わせて、キャプチャおよび格納するデータ サンプルを決定できます。
ユーザー ロジックに影響を与えずに、システム内でトリガー条件およびストレージ必要条件を変更可能	ロジック解析のためにデザインをシングル ステップまたは停止する必要がありません。
操作が容易な GUI を提供	簡単に適切なオプションを選択できます。

表 1-2 : ChipScope Pro のロジック デバッグ機能および利点 (続き)

機能	利点
各デバイスに、最大 15 個の ILA、VIO、または ATC2 コアを使用可能	ロジックを分割でき、大規模デザインの小セクションをテストできるため、精度の高い結果を得ることができます。
複数のトリガー設定	イベント数および時間を一致数および範囲と共に記録することで、精度および柔軟性が高まります。
ザイリンクス ウェブ サイトからダウンロード可能	これらのツールには、ChipScope スイートから簡単にアクセスできます。[211 ページのリファレンス 18 を参照]

デザイン フロー

ChipScope Pro ツールのデザイン フロー (図 1-2) は、一般的な HDL 合成ツールおよび ISE インプリメンテーション ツールを使用するすべての標準的な FPGA デザイン フローの一部として簡単に実行できます。

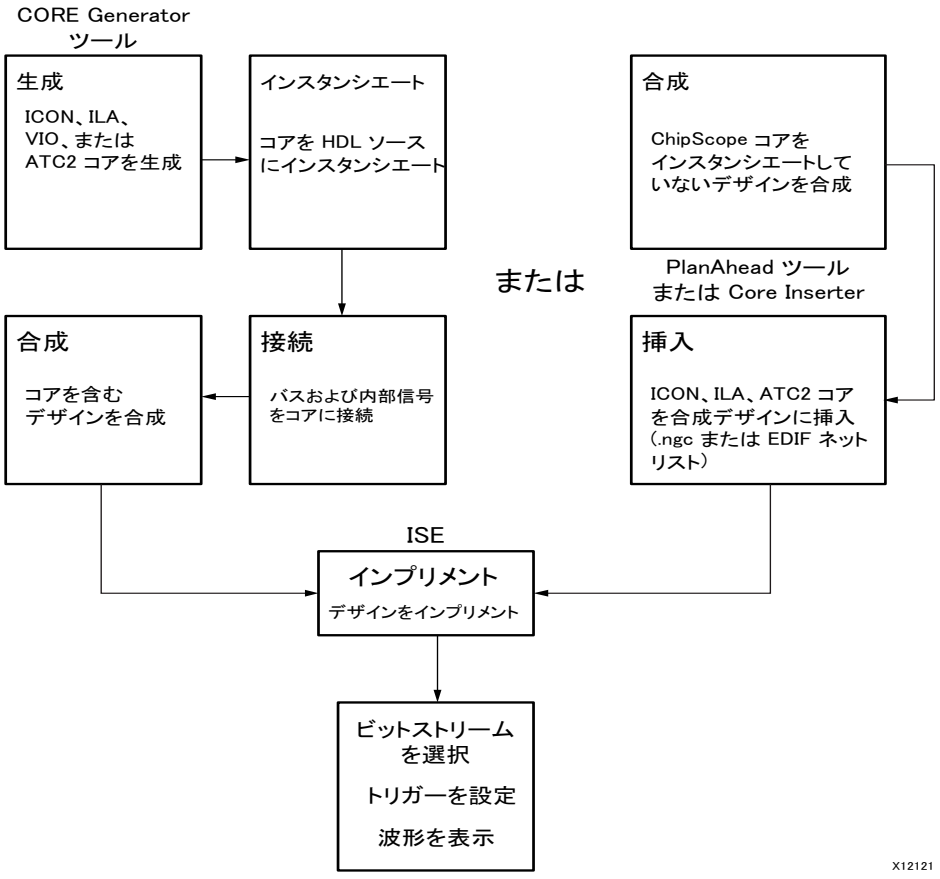

図 1-2 : ChipScope Pro ツールのデザイン フロー

PlanAhead ツールでの ChipScope Pro コアの使用

次のいずれかの方法を使用すると、PlanAhead ツールを使用してデザインに ChipScope Pro コアを追加できます。

- HDL インスタンシエーション
- ネットリスト挿入

HDL インスタンシエーションでは、次の 2 つの手順を実行します。

1. PlanAhead ツールに含まれる IP カタログから ChipScope Pro デバッグ コアを選択して、カスタマイズ、生成
2. PlanAhead ツールに含まれる HDL Editor を使用して IP コンポーネント インスタンスを HDL ソースに手動でインスタンシエート

HDL インスタンシエーションは、IP コアのすべてのパラメーターおよび HDL デザインに含まれる信号の接続を完全に制御することを必要とするユーザーに適しています。ただし、HDL インスタシエーションでは、ソース コードを変更する必要があります。また、複数の階層で構成されるデザインではデバッグの際に信号をデバッグ コア インスタンスに導く必要があるため、デバッグが困難になる可能性もあります。

ネットリスト挿入では、次の 2 つの手順を実行します。

1. デバッグするデザインで信号またはネットを選択
2. これらの信号のデバッグ IP コアへの接続方法を指定

PlanAhead ツールでは、デバッグ IP コアの生成、デザインのネットリストへのコアの挿入、およびネットへの接続が実行されます。ただし、ネットリスト挿入を実行する場合、デバッグする HDL 信号が最適化されてしまったり、合成プロセス中に不明瞭にある可能性があります。レジスタ、ブロック RAM などの出力など、デバッグする信号のほとんどは合成プロセスでこのような影響を受けません。後でデバッグできるよう信号を確実に保持するには、デザイン ソース (HDL または制約ファイル) で信号に MARK_DEBUG 属性/プロパティを付けます。MARK_DEBUG 属性およびその他の制約に関する詳細は、『制約ガイド』[211 ページのリファレンス 14 を参照] を参照してください。

MARK_DEBUG プロパティには、次のような利点があります。

- HDL インスタンシエーションに経費をかけずに、デザイン ソースで信号をデバッグ
- 合成済みネットリストでデバッグする信号を確実に保持
- XST (Xilinx Synthesis Technology) およびサードパーティ FPGA 合成ツール (Synopsys、Synplify Pro、および Mentor Graphics Precision) と互換

PlanAhead ツールの ChipScope コアのデバッグに関する詳細は、『PlanAhead ユーザー ガイド』[211 ページのリファレンス 17 を参照] を参照してください。

エンベデッド プロセッサおよび DSP ツール フローでの ChipScope Pro コアの使用

コア (ICON、ILA、IBA、VIO、および ATC2) は、エンベデッド プロセッサおよび DSP デザイン向けの EDK および System Generator for DSP ツール フローでも使用できます。ChipScope Pro コアの使用方法は、EDK Platform Studio [211 ページのリファレンス 15 を参照] および System Generator for DSP [211 ページのリファレンス 19 を参照] の資料を参照してください。

ChipScope Pro コアの概要

ICON コア

すべてのコアは、JTAG バウンダリ スキャン ポートを使用し、JTAG ダウンロード ケーブルを介してホスト コンピューターと通信します。ICON コアは、ターゲット FPGA の JTAG バウンダリ スキャン ポートと最大 15 個の ILA、VIO、および ATC2 コア間の通信パスを提供します (8 ページの図 1-1 を参照)。

Spartan-3、Spartan-3E、Spartan-3A、および Spartan-3A DSP ファミリ デバイスの場合、ICON コアは BSCAN プリミティブを介した通信に USER1 または USER2 JTAG バウンダリ スキャン命令を使用します。また、BSCAN プリミティブの未使用 USER1 または USER2 スキャン チェーンは、必要に応じてエクスポートし、アプリケーションで使用できます。

その他のデバイスの場合、BSCAN プリミティブを介して使用可能な USER1、USER2、USER3、または USER4 スキャン チェーンのいずれかを使用します。各 BSCAM プリミティブで 1 つのスキャン チェーンがインプリメントされるので、未使用の USER スキャン チェーンをエクスポートする必要はありません。

ILA コア

ILA コアは、カスタマイズ可能なロジック アナライザー コアで、デザインに含まれる任意の内部信号を監視できます。ILA コアは監視中のデザインに同期しており、このコア内のコンポーネントにも、デザインに指定したすべてのクロック制約が適用されます。ILA コアは、主に 3 つのコンポーネントで構成されています。

- トリガー入力および出力ロジック
  - トリガー入力ロジックは、トリガー イベントを検出します。
  - トリガー出力ロジックは、外部テスト装置およびその他のロジックをトリガーします。
- データ キャプチャ ロジック
  - オンチップのブロック RAM リソースを使用してトレース データ情報をキャプチャし、その情報を格納します。
- 制御およびステータス ロジック
  - ILA コアの動作を管理します。

ILA トリガー入力ロジック

ILA コアのトリガー機能には、トリガー イベント検出に必要な多くの機能が含まれます。これらの機能は、表 1-3 に記載されています。

表 1-3 : ILA コアのトリガー機能

機能	説明
ワード数の大きなトリガー ポート	各トリガー ポートは 1 ～ 256 ビット幅に設定できます。
複数のトリガー ポート	各コアで最大 16 個までのトリガー ポートを使用できます。複数の比較ユニットを使用してさまざまな信号またはバスを監視する必要がある複雑なシステムでは、複数のトリガー ポートを使用する必要があります。
各トリガー ポートに複数の比較ユニット	各トリガー ポートは、最大 16 個までの比較ユニットに接続できます。この機能により、複数のトリガー ポート信号を比較できます。
ブール式のトリガー条件	トリガー条件は、最大 16 個の比較ユニット演算子の AND または OR ブール式で表現できます。
複数レベルのトリガー シーケンサー	トリガー条件は、最大 16 個の比較ユニット演算子の複数レベルのトリガー シーケンサーで表現できます。
ブール式のストレージ必要条件	ストレージ必要条件は、最大 16 個の比較ユニット演算子の AND または OR ブール式で表現できます。

表 1-3 : ILA コアのトリガー機能 (続き)

機能	説明
比較ユニット タイプの選択	トリガー ポートに接続される比較ユニットは、次のいずれかのタイプとなります。 ・ 基本コンパレーター • = および <> 比較を実行 • LUT4[a] ベースのデバイスでスライスごとに最大 8 ビットまで比較 • Virtex-5 および Spartan-6 デバイスでスライスごとに最大 19 ビットまで比較 • LUT6[b] ベースのデバイスでスライスごとに最大 20 ビットまで比較 ・ 基本コンパレーター (エッジ付き) • = および <> 比較を実行 • High から Low および Low から High のビット遷移を検出 • LUT4 ベースのデバイスでスライスごとに最大 4 ビットまで比較 • LUT6 ベースのデバイスでスライスごとに最大 8 ビットまで比較 ・ 拡張コンパレーター • =、<>、>、>=、<、および <= 比較を実行 • LUT4 ベースのデバイスでスライスごとに最大 2 ビットまで比較 • LUT6 ベースのデバイスでスライスごとに最大 8 ビットまで比較 ・ 拡張コンパレーター (エッジ付き) • =、<>、>、>=、<、および <= 比較を実行 • High から Low および Low から High のビット遷移を検出 • LUT4 ベースのデバイスでスライスごとに最大 2 ビットまで比較 • LUT6 ベースのデバイスでスライスごとに最大 8 ビットまで比較 ・ 範囲コンパレーター • =、<>、>、>=、<、<=、in range、および not in rage 比較を実行 • LUT4 ベースのデバイスでスライスごとに最大 1 ビットまで比較 • LUT6 ベースのデバイスでスライスごとに最大 4 ビットまで比較 ・ 範囲コンパレーター (エッジ付き) • =、<>、>、>=、<、<=、in range、および not in rage 比較を実行 • High から Low および Low から High のビット遷移を検出 • LUT4 ベースのデバイスでスライスごとに最大 1 ビットまで比較 • LUT6 ベースのデバイスでスライスごとに最大 4 ビットまで比較 1 つのトリガー ポートに接続されたすべての比較ユニットは、すべて同一タイプとなります。

表 1-3 : ILA コアのトリガー機能 (続き)

機能	説明

イベント カウンターの比較演算子の選択	トリガー ポートのすべての比較ユニットは、イベント カウンターと共にコンフィギュレーションでき、カウンターのサイズは 1 ～ 32 ビットで選択可能です。このカウンターは、次の方法でイベントをカウントするように、動作時にコンフィギュレーションできます。
	• 厳密に n 回
	  • 厳密に n 回の連続的または非連続的なイベントが発生するときのみ一致
	• 最低 n 回発生した場合のみ
	  • 最低 n 回の連続的または非連続的なイベントが発生すると一致し、アサートを保持
	• 最低 n 回連続的に発生した場合のみ
	  • n 回の連続的なイベントが発生すると一致し、比較演算子を満たさなくなるまでアサートを保持

トリガー出力ポート	オプションのトリガー出力ポートを使用すると、ILA コアの内部トリガー条件にアクセスできます。この信号は、出力ピンに接続することによって、外部テスト装置用のトリガーとして使用できます。 内部ロジックの割り込みまたはトリガーとして、または複数の ILA コアのカスケード接続用にも使用可能です。 ILA のトリガー出力ポートには、10 クロック サイクルのレイテンシがあります。 トリガー出力のレベル/パルスおよびアクティブ エッジ (High または Low) は、動作時に制御できます。

a. LUT4 ベースのデバイス ファミリには、Spartan-3、Spartan-3E、Spartan-3A、Spartan-3A DSP、および Virtex-4 FPGA が含まれます。

b. LUT6 ベースのデバイス ファミリには、Virtex-5、Virtex-6、Spartan-6、Artix™-7、Kintex-7、および Virtex-7 FPGA が含まれます。

複数のトリガー ポートの使用

デザインで異なるタイプの信号またはバスを監視できるようにするには、複数のトリガー ポートが必要となります。たとえば、デザインで制御、アドレス、およびデータ信号を含む内部システム バスを使用している場合、これらにそれぞれトリガー ポートを割り当てて、各信号グループを監視できます (図 1-3)。

これらの信号およびバスを 1 つのトリガー ポートに接続する場合は、アドレス バスが指定された範囲内にあるかを確認している間に CE、WE、および OE 信号の各ビット遷移を監視することはできません。さまざまなタイプの比較ユニットから選択可能であるため、最低限のリソースを使用しながら、必要なトリガー向けに ILA コアをカスタマイズできます。

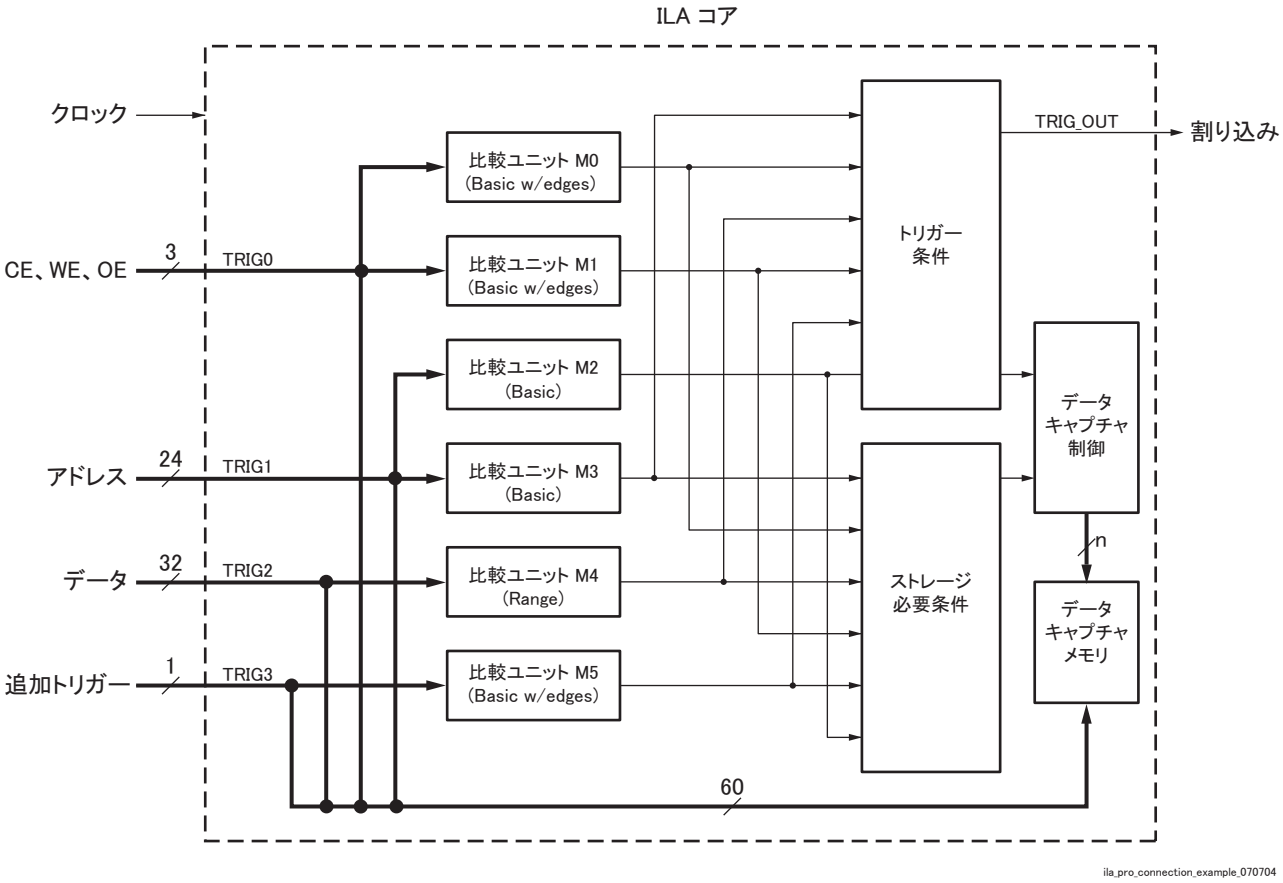

図 1-3 : ILA コアの接続例

トリガー条件およびストレージ必要条件の使用

ILA コアでは、トリガーおよびストレージ必要条件ロジックの両方がインプリメントされます。トリガー条件は、コアのトリガー ポートに接続されている比較ユニット コンパレータで検出されるイベントのブール式またはシーケンシャルな組み合わせです。トリガー条件は、データ キャプチャ ウィンドウで明確な開始点を示すために使用され、データ キャプチャ ウィンドウの開始点、終了点、あるいは任意の位置に指定できます。

同様に、ストレージ必要条件も、コアのトリガー ポートに接続されている比較ユニット コンパレーターで検出されるイベントのブール式組み合わせです。ただし、この条件は、個別のデータ サンプルをキャプチャおよび格納するかを決定するために、トリガー ポートの比較ユニットのイベントを評価する点でトリガー条件と異なります。トリガー条件およびストレージ必要条件を共に使用し、キャプチャ プロセスの開始時とキャプチャするデータを決定できます。

16 ページの図 1-3 に示す ILA コアの例では、次が実行されます。

- Address = 0xFF0000 への最初のメモリ書き込みサイクル (CE = 立ち上がりエッジ、WE = 1、OE = 0) でトリガー
- データ値が 0x00000000 ～ 0x1000FFFF の間の場合に、Address = 0x23AACC からのメモリ読み出しサイクル (CE = 立ち上がりエッジ、WE = 0、OE = 1) のみをキャプチャ

これらの条件を正しくインプリメントするには、TRIG0 および TRIG1 トリガー ポートの両方にそれぞれ比較ユニット 2 個 (トリガー条件用 1 個とストレージ必要条件用 1 個) が接続されていることを確認する必要があります。次に、トリガーおよびストレージ必要条件の設定方法とそれらの条件を満たすための各比較ユニットの設定方法を示します。

- トリガー条件 = M0 && M2
  - M0[2:0] = CE、WE、OE = “R10” (R は立ち上がりエッジを示す)
  - M2[23:0] = アドレス = “FF0000”
- ストレージ必要条件 = M1 && M3 && M4
  - M1[2:0] = CE、WE、OE = “R10” (R は立ち上がりエッジを示す)
  - M3[23:0] = アドレス = “23AACC”
  - M4[31:0] = データ = 範囲は 0x00000000 ～ 0x1000FFFF

ILA コアのトリガーおよびストレージ必要条件を設定することにより、オンチップ メモリ リソースを浪費せずに、必要な情報のみを正確に検索し、キャプチャできます。

ILA トリガー出力ロジック

ILA コアでは TRIG_OUT と呼ばれるトリガー出力ポートがインプリメントされます。TRIG_OUT ポートは、ChipScope Pro Analyzer を使用して動作時に設定されるトリガー条件の出力です。トリガー出力のレベル/パルスおよびアクティブ エッジ (High または Low) も、動作時に制御できます。入力トリガー ポートに対する TRIG_OUT のレイテンシは、10 クロック サイクルです。

TRIG_OUT ポートは非常に柔軟性があり、多用途に使用できます。このポートをデバイス ピンに接続し、オシロスコープおよびロジック アナライザーなどの外部テスト装置をトリガーできます。また、デバイスに組み込まれた PowerPC™ または MicroBlaze™ プロセッサの割り込みラインに接続すると、ソフトウェア イベントを発生させることができます。さらに、別のコアのトリガー入力ポートに接続すると、オンチップ デバッグ ソリューションのトリガーおよびデータ キャプチャ機能を拡張できます。

ILA データ キャプチャ ロジック

各 ILA コアは、オンチップ ブロック RAM リソースを使用して、デザインに含まれるその他すべてのコアから独立してデータをキャプチャできます。また、[Window] または [N Samples] のいずれかのキャプチャ モードでデータをキャプチャできます。

[Window] キャプチャ モード

このモードでは、サンプル バッファーを 1 つまたは複数の等サイズのサンプル ウィンドウに分割できます。このモードの場合、1 つのトリガー条件イベント (個々のトリガー比較ユニット イベントのブール式組み合わせ) を使用して、サンプル ウィンドウを満たすのに十分なデータが収集されます。

サンプル ウィンドウのワード数が 131,072 サンプルまでの 2 のべき乗の場合、トリガー位置はサンプル ウィンドウの開始点 (最初にトリガーしてからデータを収集)、終了点 (トリガー イベントまでデータを収集)、またはそれら 2 点間の任意の位置に設定できます。

ウィンドウのワード数が 2 のべき乗以外の場合、トリガー位置はサンプル ウィンドウの開始位置にのみ設定できます。

サンプル ウィンドウが満たされると、ILA コアでトリガー条件が自動的に再設定され、トリガー条件イベントが継続して監視されます。このプロセスは、サンプル バッファーのすべてのサンプル ウィンドウが満たされるか、ユーザーが ILA コアを停止するまで繰り返されます。

[N Samples] キャプチャ モード

このモードは、ウィンドウ キャプチャ モードと類似していますが、次の 2 点が異なります。

- ウィンドウごとのサンプル数は、1 ～ （サンプル バッファー サイズ - 1）の範囲で、任意の整数 N に設定可能
- トリガー位置は常にウィンドウの位置 0 に設定

このモードは、キャプチャ ストレージ リソースを浪費せずに、各トリガーで必要なサンプル数のみをキャプチャする場合に役立ちます。

トリガー マーク

トリガー イベントと一致するサンプル ウィンドウ内のデータ サンプルには、トリガー マークが付けられます。このトリガー マークによって、ウィンドウ内のトリガー位置が Chipscope Pro Analyzer に伝えられます。トリガー マークは、サンプル バッファー内の 1 サンプルに対して 1 ビットを使用します。

データ ポート

トリガー機能を実行するトリガー ポートとは別のポート上のデータをキャプチャできます。この機能は、コアのトリガーに使用される情報と同じ情報のキャプチャおよび確認が有用ではなく、キャプチャするデータ量を比較的少ない量に制限する際に役立ちます。

ただし、通常は、コアのトリガーに使用されるデータと同一データのキャプチャおよび確認が有用です。このような場合、データが 1 つまたは複数のトリガー ポートで構成されるように選択できます。この機能により、キャプチャに必要なトリガー情報を選択できる柔軟性を活用しながら、リソースを節約できます。

ILA 制御およびステータス ロジック

ILA コアには、コアの通常動作を維持するために使用する制御およびステータス ロジックが少数含まれます。ILA コアを適切に認識し、通信するのに必要なすべてのロジックが制御およびステータス ロジックによってインプリメントされます。

VIO コア

VIO（Virtual Input/Output) は、内部 FPGA 信号を即時に監視および駆動できるカスタマイズ可能なコアです。ILA コアとは違い、オンチップ RAM やオフチップ RAM は必要ありません。VIO コアでは、次の 4 種類の信号が使用できます。

- 非同期入力
  - JTAG ケーブルから駆動される JTAG クロック信号を使用してサンプリングされます。
  - 入力値は定期的に読み戻され、ChipScope Pro Analyzer で表示されます。
- 同期入力
  - デザイン クロックを使用してサンプリングされます。
  - 入力値は定期的に読み戻され、ChipScope Pro Analyzer に表示されます。
- 非同期出力
  - ChipScope Pro Analyzer で定義する信号で、コアから周りのデザインへ出力されます。
  - 各非同期出力には、論理値 0 または 1 を定義できます。
- 同期出力
  - ChipScope Pro Analyzer で定義する信号で、デザイン クロックに同期しており、コアから周辺デザインへ出力されます。
  - 各同期出力には、論理値 1 または 0 を定義できます。また、1 および 0 の 16 クロック サイクル分のパルス列も定義できます。

アクティビティ検出器

VIO コア入力には、入力の遷移をキャプチャするためのセルが別にあります。デザイン クロックが ChipScope Pro Analyzer のサンプル周期よりも速いことがほとんどなので、連続するサンプル間で信号の遷移を何度も監視できます。アクティビティ検出器はこの動作を検出し、結果と値を ChipScope Pro Analyzer に表示します。

同期入力の場合は、非同期イベントと同期イベントを監視するアクティビティ セルが使用されます。この機能は、同期信号上でのグリッチや同期遷移を検出する場合にも使用できます。

パルス列

VIO の同期出力すべてに、スタティック 1、スタティック 0、または連続する値のパルス列を出力する機能があります。パルス列とは、連続したデザイン クロック サイクルでコアから駆動される、16 クロック サイクル分の 1 および 0 のシーケンスです。パルス列シーケンスは、ChipScope Pro Analyzer で定義され、コアに読み込まれた後 1 度だけ実行されます。

ATC2 コア

ATC2 (Agilent Trace Core 2) は、カスタマイズ可能なデバッグ キャプチャ コアで、最新のアジレント テクノロジー社ロジック アナライザーと機能するように設計されています。ATC2 コアでは、外部のアジレント テクノロジー社ロジック アナライザーにより FPGA デザイン内部のネットへアクセスできます (図 1-4)。

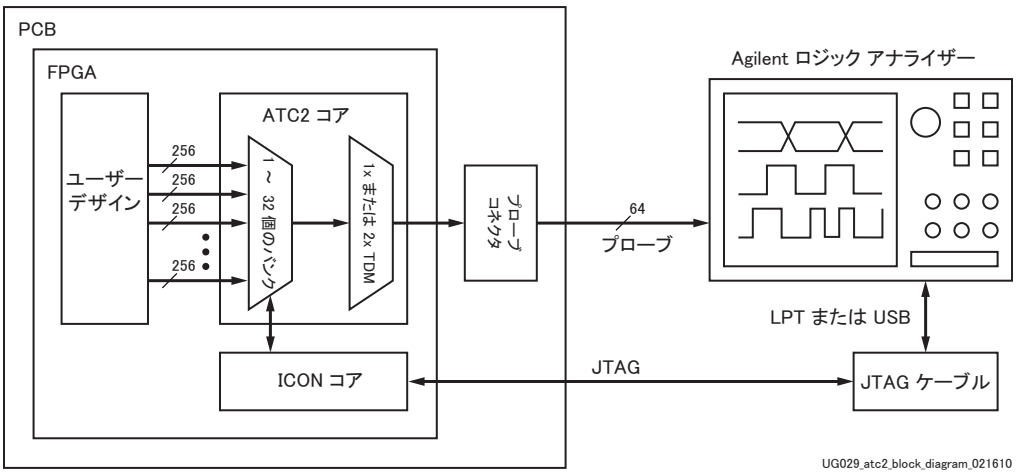

図 1-4 : ATC2 コアおよびシステム ブロック図

ATC2 コアのデータ パスについて

ATC2 コアのデータ パスは、次で構成されています。

- ユーザー FPGA デザインに接続される、実行時に選択可能な最大 64 個の入力信号バンク
- アジレント テクノロジー社ロジック アナライザーのプローブ コネクタに接続される最大 64 個の出力データ ピン
- オプションで信号バンクの幅を 64 から 2倍の 128 ビットにする 2x TDM (時分割多重) を各出力データ ピンで使用可能
- 非同期タイミングおよび同期ステート キャプチャ モードを共にサポート
- それぞれの出力データ ピンに対して、有効な I/O 規格、駆動電流、および出力スルー レートをサポート
- アジレント テクノロジー社のプローブ接続技術をサポート [212 ページのリファレンス 25 を参照]

動作時に使用可能なデータ プローブ ポイントの最大数は、次の式で求められます。
(64 データ ポート) * (データ ポートごとに 64 ビット) * (2x TDM) = 8192 プローブ ポイント

ATC2 コアのデータ キャプチャおよび実行時の制御

外部の アジレント テクノロジー社ロジック アナライザーを使用し、ATC2 コアを通過するデータをトリガーおよびキャプチャします。これにより、アジレント テクノロジー社ロジック アナライザーの複雑なトリガー、ワード数の多いトレース メモリ、およびシステム レベルのデータ相関機能を十分に活用でき、同時に ATC2 コアが示す内部デザイン ノードがよりわかりやすくなります。また、アジレント テクノロジー社ロジック アナライザーは、JTAG ポート接続を介して ATC2 コアと通信することによって、動作時にアクティブ データ ポート選択を制御する場合にも使用されます (図 1-4)。

IBERT コア

IBERT コアには、制御、監視、トランシーバー パラメーターの変更、およびビット エラー比率テストを実行するすべてのロジックが含まれています。IBERT コアには、主に 3 つのコンポーネントがあります。

- BERT ロジック
  - BERT ロジックはトランシーバー コンポーネントをインスタンシエートし、パターン ジェネレーターおよびチェッカーを含んでいます。単純なクロック タイプ パターンから PRBS パターンやフレーム付きカウンター パターンまでさまざまなパターンを使用できます。
- ダイナミック リコンフィギュレーション ポート (DRP) ロジック
  - 各トランシーバーには、ダイナミック リコンフィギュレーション ポート (DRP) があり、トランシーバーの属性をシステムで変更できます。すべての属性および DRP アドレスは IBERT コアで読み出し/書き込み可能です。各トランシーバーの DRP は個別にアクセスできます。
- 制御およびステータス ロジック
  - IBERT コアの操作を管理します。

IBERT デザイン フロー

IBERT は内蔵型デザインのため、デザイン フローは非常に単純です。ChipScope IBERT Core Generator を使用して Virtex-5 デバイス向けの IBERT コア デザインを生成すると、デザイン ディレクトリおよび BIT ファイル名が指定され、オプションが選択され、ビットストリーム生成を含むインプリメンテーション フローすべてがワンステップで実行されます。

Kintex-7、Virtex-6、および Spartan-6 デバイスの IBERT コア デザインを生成するデザイン フローは、ザイリンクス CORE Generator を使用するという点を除き類似しています。主な違いは、デザイン ディレクトリとデバイス情報がザイリンクスの CORE Generator プロジェクトで指定されるという点です。両方の場合で、IBERT コアのデザイン BIT ファイルを生成するために別のザイリンクス ソフトウェアを実行する必要はありません。

IBERT の機能

IBERT コアの機能は、ターゲットにする FPGA デバイスのアーキテクチャによって異なります。サポートされる MGT 機能は、次のとおりです。

- Virtex-5 FPGA GTP および GTX トランシーバー用 IBERT v1.0 コア (23 ページの表 1-4)
  - 差動スイング、エンファシス、RX イコライゼーション、および DFE を含む PMA (Physical Medium Attachment) の完全制御
  - 実行時にライン レートおよびリファレンス クロック ソースを変更可能
  - ループバックおよび 8B/10B エンコードのイネーブル/ディスエーブルを含む PCS (Physical Coding Sublayer) サポート (制限あり)。クロック コレクションおよびチャネル ボンディングはサポートされていません。
  - GTP トランシーバーに 2 バイト ファブリック幅、GTX トランシーバーに 4 バイト ファブリック幅
- Virtex-5 FPGA GTX トランシーバー用 IBERT v2.0 コア (24 ページの表 1-5)
  - 差動スイング、エンファシス、RX イコライゼーション、および DFE を含む PMA の完全制御
  - 実行時にライン レートを変更可能

  - ループバックを含む制限付き PCS サポート (8b/10b エンコード、クロック コレクション、およびチャネル ボンディングはサポートされていません。)
  - 40 ビットのファブリック データ幅 (4 バイト モード)
- Virtex-6 FPGA GTX トランシーバー用 IBERT v2.0 コア (25 ページの表 1-6)
  - 差動スイング、エンファシス、RX イコライゼーション、および DFE を含む PMA の完全制御
  - 実行時にライン レートを変更可能
  - 生成時にリファレンス クロック ソースを設定可能
  - ループバックを含む制限付き PCS サポート。パターン エンコード、クロック コレクション、およびチャネル ボンディングはサポートされていません。
- Virtex-6 FPGA GTH トランシーバー用 IBERT v2.0 コア (26 ページの表 1-7)
  - 差動スイング、エンファシス、RX イコライゼーション、および DFE を含む PMA の完全制御
  - 生成時にリファレンス クロック ソースを設定可能
  - ループバックを含む制限付き PCS サポート。パターン エンコード、クロック コレクション、およびチャネル ボンディングはサポートされていません。
  - TX 差動スイング
  - TX プリエンファシスおよびポストエンファシス
- Spartan-6 FPGA GTP トランシーバー用 IBERT v2.0 コア (27 ページの表 1-8)
  - 差動スイング、エンファシス、RX イコライゼーション、および DFE を含む PMA の完全制御
  - 実行時にライン レートを変更可能
  - 生成時にリファレンス クロック ソースを設定可能
  - ループバックを含む制限付き PCS サポート。パターン エンコード、クロック コレクション、およびチャネル ボンディングはサポートされていません。
  - TX 差動スイング
  - TX プリエンファシス
- Kintex-7 および Virtex-7 FPGA GTX トランシーバー用 IBERT v2.00a コア (28 ページの表 1-9)
  - 差動スイングおよびエンファシスを含む PMA 制御
  - 生成時にライン レートを変更可能
  - 生成時にリファレンス クロック ソースを設定可能
  - ループバックを含む制限付き PCS サポート。パターン エンコード、クロック コレクション、およびチャネル ボンディングはサポートされていません。

表 1-4 : Virtex-5 FPGA GTP および GTX トランシーバー用 IBERT v1.0 コア

機能	説明
複数のマルチギガビット トランシーバー	デザインに最大 8 個のトランシーバーを選択可能
パターン ジェネレーター	選択したトランシーバーごとに 1 つのパターン ジェネレーターが使用されます。基本的なパターン ジェネレーターを選択する場合は、PRBS (Pseudo Random Bit Sequence) 7 ビット、PRBS 23 ビット、PRBS 31 ビット、およびユーザー定義のパターンが使用されます。完全なパターン ジェネレーターを選択する場合は、上述のパターンに加えて、代替 PRBS 7 ビット、PRBS 9 ビット、PRBS 11 ビット、PRBS 15 ビット、PRBS 20 ビット、PRBS 29 ビット、フレーム付きカウンター、およびアイドル パターンが使用されます。すべてのトランシーバーで使用可能なパターン セットはコンパイル時に一度選択されるのに対し、そのセットの特定のパターンは実行時に各トランシーバーで個別に選択できます。
パターン チェッカー	選択したトランシーバーごとに 1 つのパターン チェッカーが使用されます。同じパターン セットをパターン ジェネレーターとして使用できます。パターンは、ランタイム時に各トランシーバーでそれぞれ選択できます。
ファブリック幅	GTP トランシーバーに対する FPGA ファブリックのインターフェイスは、2 バイト モードで固定されています。GTX トランシーバーに対する FPGA ファブリックのインターフェイスは、4 バイト モードで固定されています。
BERT パラメーター	受信したエラーを含むビット数および受信したワード数の合計が即時に集計されて ChipScope Pro Analyzer で読み出されます。
極性	各トランシーバーの TX または RX 側の極性を実行時に変更できます。
8b/10b エンコード/デコードのサポート	8b/10b エンコード/デコードは、デュアルトランシーバー (GTP_DUAL または GTX_DUAL タイル) ごとに実行時にイネーブルにできます。TX エンコードおよび RX デコードが同時に選択されます。 **注記** : 8B/10B エンコード/デコードがイネーブルの場合は、フレーム付きカウンター パターンおよびアイドル パターンのみを使用できます。
リセット	各トランシーバーの BER カウンターを個別にリセットできます。すべてのトランシーバーおよび BER カウンターを一度にリセットするグローバル リセットも使用できます。
リンクおよびロック ステータス	各トランシーバーのリンク、DCM、および PLL ロック ステータスを集めます。
DRP 読み出し	各トランシーバーのダイナミック リコンフィギュレーション ポート (DRP) のコンテンツを個別に読み出すことができます。
DRP 書き込み	各トランシーバーの DRP のコンテンツを実行時にシングル ビット精度で変更できます。
ステータス	コア全体のダイナミック ステータス情報を実行時に読み出すことができます。

表 1-5 : Virtex-5 FPGA GTX トランシーバー用 IBERT v2.0 コア

機能	説明
複数の GTX トランシーバー	デザインに最大 8 個のトランシーバーを選択可能
パターン ジェネレーター	選択した GTX トランシーバーごとに 1 つのパターン ジェネレーターが使用されます。使用できるパターンは、PRBS 7 ビット、PRBS 15 ビット、PRBS 23 ビット、PRBS 31 ビット、Clk 2x、および Clk 10x パターンです。各 GTX トランシーバーに対して、任意のパターンを実行時に選択できます。
パターン チェッカー	選択した GTX トランシーバーごとに 1 つのパターン チェッカーが使用されます。同じパターン セットをパターン ジェネレーターとして使用できます。パターンは、ランタイム時に各 GTX トランシーバーでそれぞれ選択できます。
ファブリック幅	GTX_DUAL タイルへの FPGA ファブリック インターフェイスは、32 または 40 ビット幅にでき、生成時に選択できます。
BERT パラメーター	受信したエラーを含むビット数および受信したワード数の合計が即時に集計されて ChipScope Pro Analyzer で読み出されます。
極性	各 GTX トランシーバーの TX または RX 側の極性を実行時に変更できます。
リセット	各 GTX トランシーバーおよびその BER カウンターを個別にリセットできます。PLL を含む MGT 全体をリセットするリセットもあります。
リンクおよびロック ステータス	各 GTX トランシーバーのリンク、DCM、および PLL ロック ステータスを集めます。
DRP 読み出し	各 GTX トランシーバーのダイナミック リコンフィギュレーション ポート (DRP) のコンテンツを個別に読み出すことができます。
DRP 書き込み	各 GTX トランシーバーの DRP のコンテンツを実行時にシングルビット精度で変更できます。
ポートの読み出し	GTX トランシーバーのポートを監視するレジスタのコンテンツを個別に読み出すことができます。
ポートへの書き込み	GTX トランシーバーのポートを制御するレジスタのコンテンツを実行時に変更できます。
ステータス	コア全体のダイナミック ステータス情報を実行時に読み出すことができます。

表 1-6 : Virtex-6 FPGA GTX トランシーバ用 IBERT v2.0 コア

機能	説明
複数の GTX トランシーバー	デザインに最大 8 個のトランシーバーを選択可能
パターン ジェネレーター	選択した GTX トランシーバーごとに 1 つのパターン ジェネレーターが使用されます。使用できるパターンは、PRBS 7 ビット、PRBS 15 ビット、PRBS 23 ビット、PRBS 31 ビット、Clk 2x、および Clk 10x パターンです。各 GTX トランシーバに対して、任意のパターンを実行時に選択できます。
パターン チェッカー	選択した GTX トランシーバーごとに 1 つのパターン チェッカーが使用されます。同じパターン セットをパターン ジェネレーターとして使用できます。パターンは、ランタイム時に各 GTX トランシーバーでそれぞれ選択できます。
ファブリック幅	GTX トランシーバーへの FPGA ファブリック インターフェイスは、16 または 20 ビット幅にでき、生成時に選択できます。
BERT パラメーター	受信したエラーを含むビット数および受信したワード数の合計が即時に集計されて ChipScope Pro Analyzer で読み出されます。
極性	各 GTX トランシーバーの TX または RX 側の極性を実行時に変更できます。
リセット	各 GTX トランシーバーおよびその BER カウンターを個別にリセットできます。PLL を含む MGT 全体をリセットするリセットもあります。
リンクおよびロック ステータス	各 GTX トランシーバーのリンク、DCM、および PLL ロック ステータスを集めます。
DRP 読み出し	各 GTX トランシーバーのダイナミック リコンフィギュレーション ポート (DRP) のコンテンツを個別に読み出すことができます。
DRP 書き込み	各 GTX トランシーバーの DRP のコンテンツを実行時にシングルビット精度で変更できます。
ポートの読み出し	GTX トランシーバーのポートを監視するレジスタのコンテンツを個別に読み出すことができます。
ポートへの書き込み	GTX トランシーバーのポートを制御するレジスタのコンテンツを実行時に変更できます。
ステータス	コア全体のダイナミック ステータス情報を実行時に読み出すことができます。

表 1-7 : Virtex-6 FPGA GTH トランシーバー用 IBERT v2.0 コア

機能	説明
複数の GTH トランシーバー	デザインに最大 16 個のトランシーバーを選択可能
パターン ジェネレーター	選択した GTH トランシーバーごとに 1 つのパターン ジェネレーター (クワッドごとに 4 つ) が使用されます。使用できるパターンは、PRBS 7 ビット、PRBS 15 ビット、PRBS 23ビット、PRBS 31 ビット、Clk 2x、および Clk 10x パターンです。各 GTH トランシーバーに対して、任意のパターンを実行時に選択できます。
パターン チェッカー	選択した GTH トランシーバーごとに 1 つのパターン チェッカー (クワッドごとに 4 つ) が使用されます。同じパターンセットをパターン ジェネレーターとして使用できます。パターンは、ランタイム時に各 GTH トランシーバーでそれぞれ選択できます。
ファブリック幅	GTH QUAD トランシーバーへの FPGA ファブリック インターフェイスは、16 または 20 ビット幅にでき、生成時に選択できます。
BERT パラメーター	受信したエラーを含むビット数および受信したワード数の合計が即時に集計されて ChipScope Pro Analyzer で読み出されます。
極性	各 GTH トランシーバーの TX または RX 側の極性を実行時に変更できます。
リセット	各 GTH トランシーバーの BER カウンターを個別にリセットできます。PLL を含む GTH クワッド全体をリセットするリセットもあります。
リンクおよびロック ステータス	各 GTH トランシーバーのリンク、DCM、および PLL ロック ステータスを集めます。
DRP 読み出し	各 GTH トランシーバーのダイナミック リコンフィギュレーション ポート (DRP) のコンテンツを個別に読み出すことができます。
DRP 書き込み	各 GTH トランシーバーの DRP のコンテンツを実行時にシングル ビット精度で変更できます。
ポートの読み出し	GTH トランシーバーのポートを監視するレジスタのコンテンツを個別に読み出すことができます。
ポートへの書き込み	GTH トランシーバーのポートを制御するレジスタのコンテンツを実行時に変更できます。
ステータス	コア全体のダイナミック ステータス情報を実行時に読み出すことができます。

表 1-8 : Spartan-6 FPGA GTP トランシーバー用 IBERT v2.0 コア

機能	説明
複数の GTP トランシーバー	デザインに最大 8 個のトランシーバーを選択可能
パターン ジェネレーター	選択した GTP トランシーバーごとに 1 つのパターン ジェネレーター (クワッドごとに 2 つ) が使用されます。使用できるパターンは、PRBS 7 ビット、PRBS 15 ビット、PRBS 23ビット、PRBS 31 ビット、Clk 2x、および Clk 10x パターンです。各 GTP トランシーバーに対して、任意のパターンを実行時に選択できます。
パターン チェッカー	選択した GTP トランシーバーごとに 1 つのパターン チェッカー (デュアルごとに 2 つ) が使用されます。同じパターンセットをパターン ジェネレーターとして使用できます。パターンは、ランタイム時に各 GTP トランシーバーでそれぞれ選択できます。
ファブリック幅	GTP トランシーバーに対する FPGA のファブリック インターフェイス幅は 20 ビットです。
BERT パラメーター	受信したエラーを含むビット数および受信したワード数の合計が即時に集計されて ChipScope Pro Analyzer で読み出されます。
極性	各 GTP トランシーバーの TX または RX 側の極性を実行時に変更できます。
リセット	各 GTP トランシーバーの BER カウンターを個別にリセットできます。PLL を含む GTP トランシーバー全体をリセットするリセットもあります。
リンクおよびロック ステータス	各 GTP トランシーバーのリンク、DCM、および PLL ロックステータスを集めます。
DRP 読み出し	各 GTP トランシーバーのダイナミック リコンフィギュレーション ポート (DRP) のコンテンツを個別に読み出すことができます。
DRP 書き込み	各 GTP トランシーバーの DRP のコンテンツを実行時にシングル ビット精度で変更できます。
ポートの読み出し	GTP トランシーバーのポートを監視するレジスタのコンテンツを個別に読み出すことができます。
ポートへの書き込み	GTP トランシーバーのポートを制御するレジスタのコンテンツを実行時に変更できます。
ステータス	コア全体のダイナミック ステータス情報を実行時に読み出すことができます。

表 1-9 : Kintex-7 および Virtex-7 FPGA GTX トランシーバー用 IBERT v2.00a コア

機能	説明
複数の GTX トランシーバー	デザインに最大 8 個のトランシーバーを選択可能
パターン ジェネレーター	選択した GTX トランシーバーごとに 1 つのパターン ジェネレーターが使用されます。使用できるパターンは、PRBS 7 ビット、PRBS 15 ビット、PRBS 23ビット、PRBS 31 ビット、Clk 2x、および Clk 10x パターンです。各 GTX トランシーバーに対して、任意のパターンを実行時に選択できます。
パターン チェッカー	選択した GTX トランシーバーごとに 1 つのパターン チェッカーが使用されます。同じパターン セットをパターン ジェネレーターとして使用できます。パターンは、ランタイム時に各 GTX トランシーバーでそれぞれ選択できます。
ファブリック幅	GTX トランシーバーへの FPGA ファブリック インターフェイスは、32 または 40 ビット幅にでき、生成時に選択できます。
極性	各 GTX トランシーバーの TX 側の極性を実行時に変更できます。
リセット	各 GTX トランシーバーを個別にリセットできます。PLL および CPLL を含む MGT 全体をリセットするリセットもあります。
リンクおよびロック ステータス	各 GTX トランシーバーのリンクおよび CPLL/QPLL ロック ステータスを集めます。
DRP 読み出し	各 GTX トランシーバーのダイナミック リコンフィギュレーション ポート (DRP) のコンテンツを個別に読み出すことができます。
DRP 書き込み	各 GTX トランシーバーの DRP のコンテンツを実行時にシングル ビット精度で変更できます。
ポートの読み出し	GTX トランシーバーのポートを監視するレジスタのコンテンツを個別に読み出すことができます。
ポートへの書き込み	GTX トランシーバーのポートを制御するレジスタのコンテンツを実行時に変更できます。
ステータス	コア全体のダイナミック ステータス情報を実行時に読み出すことができます。

ILA、VIO、および ATC2 コアのオプションの多くは、再合成せずに変更できます。ただし、データ ポート幅またはサンプル バッファーのワード数などの選択可能なパラメーターの変更後には、新規コアでデザインを再合成する必要があります。表 1-10 に、再合成が必要なデザインを示します。

表 1-10 : デザインのパラメーター変更および再合成

デザインで変更するパラメーター	再合成の必要
トリガー パターンの変更	なし
トリガーの実行および停止	なし
外部トリガーのイネーブル	なし
トリガー信号のソース変更	なし(1)
データ信号のソース変更	なし(1)
ILA のクロック信号の変更	あり
サンプル バッファーのワード数変更	あり

注記 :

1. 既存のトリガーおよびデータ信号のソースの両方またはいずれかの変更機能は、ISE FPGA Editor でサポートされています。

システム要件

OS 要件

ChipScope Pro の OS 要件は、『ISE Design Suite 13 : インストールおよびライセンス ガイド』および『ISE Design Suite 13 : リリース ノート ガイド』[211 ページのリファレンス 14 を参照] に記載されています。

ソフトウェア要件

ザイリンクス CORE Generator、ChipScope Pro Core Inserter、IBERT Core Generator、および CSE/Tcl ツールでは、ISE インプリメンテーション ツールがシステムにインストールされていることを前提とします (Tcl とは Tool Command Language の略語であり、Tcl シェル は Tcl スクリプトの実行に使用されるシェル プログラムです)。CSE/Tcl では、ChipScope Pro および ISE ツールのインストールに含まれている Tcl シェル (`xtclsh` と呼ぶ) が必要です。

注記 : ChipScope Pro のバージョンは、ChipScope Pro コアを含むデザインをインプリメントするときに使用する ISE ツールのバージョンと一致させる必要があります (アップデート リビジョンを含む)。

通信要件

ChipScope Pro Analyzer ツールでは、PC と JTAG バウンダリ スキャン チェーン内のデバイスとの通信用に、次のケーブルを使用できます (表 1-11 を参照)。

- プラットフォーム ケーブル USB II
- プラットフォーム ケーブル USB
- パラレル ケーブル IV
- Digilent 社製 JTAG-SMT1 および JTAG-HS1 USB - JTAG ダウンロード ケーブル
- ByteTools 社製 Catapult EJ-1 イーサネット - JTAG 接続ケーブル [212 ページのリファレンス 27 を参照]

注記 : ChipScope Pro Analyzer でデバイス内の ILA コアと通信しながら、iMPACT でデバイスをコンフィギュレーションするなど、ケーブルまたはデバイスで競合操作を行うと、DUT (テスト対象デザイン) が使用できなくなる可能性があります。競合しているケーブル/デバイス操作の結果が不確かなときは、競合する操作が完了するまで ChipScope Pro Analyzer のケーブル接続を解除してください。

表 1-11 : ChipScope Pro がサポートするダウンロード ケーブル

ダウンロード ケーブル	機能
プラットフォーム ケーブル USB II およびプラットフォーム ケーブル USB(1)	· USB ポート (USB 2.0 または USB 1.1) を使用して、テスト対象ボードのバウンダリ スキャン チェーンと通信 · 最大 12Mb/s スループットでダウンロード · 5V ～ 1.5V で動作するシステムおよびデバイス I/O との通信を可能にする調整可能な電圧インターフェイスを含む · Windows および Red Hat Linux OS のサポート
パラレル ケーブル IV(1)	· プリンター ポートなどのパラレル ポートを使用して、テスト対象ボードのバウンダリ スキャン チェーンと通信 · 最大 5Mb/s スループットでダウンロード · 5 V ～ 1.5 V で動作するシステムおよびデバイス I/O との通信を可能にする調整可能な電圧インターフェイスを含む · Windows および Red Hat Linux OS のサポート

表 1-11 : ChipScope Pro がサポートするダウンロード ケーブル

ダウンロード ケーブル	機能
Digilent 社製 JTAG-SMT1 および JTAG-HS1 USB - JTAG ダウンロード ケーブル	・ USB ポート (USB 2.0 または USB 1.1) を使用して、テスト対象ボードのバウンダリ スキャン チェーンと通信 ・ 最大 30Mb/s スループットでダウンロード ・ 電圧が調整可能なインターフェイスにより 5V ～ 1.5V で動作するデバイス I/O をサポート ・ Windows および Linux OS をサポート 詳細は、Digilent 社ウェブサイトを参照 : http://www.digilentinc.com
ByteTools 社製 Catapult EJ-1 イーサネット-JTAG 接続ケーブル	・ イーサネット ポートを使用して、テスト対象ボードのバウンダリ スキャン チェーンと通信 ・ 詳細は、ByteTools 社のウェブサイトを参照 [212 ページのリファレンス 27 を参照]

注記 :

1. パラレル ケーブル IV およびプラットフォーム ケーブル USB は、ザイリンクス オンライン ストア [211 ページのリファレンス 20 を参照] から購入可能です ([Buy Online] → [Programming Cables] をクリックしてください。ただし、日本のお客様は [購入情報] にリストされている販売代理店までお問い合わせください)。

ボード要件

テスト対象ボードで ChipScope Pro Analyzer とダウンロード ケーブルを適切に動作させるには、次のボード レベル要件を満たす必要があります。

- サポートされているデバイスを TDI、TMS、TCK、および TDO ピンを含む JTAG ヘッダーに接続する必要があります。
- 別のデバイスがターゲット デバイスを含む JTAG チェーンの TDI、TMS、TCK ピンを駆動する場合には、これらのソースをディスエーブルにしてダウンロード ケーブルでの競合を回避できるよう、これらの信号にジャンパーが必要です。
- ダウンロード ケーブルとしてパラレル ケーブル IV、プラットフォーム ケーブル USB、Digilent 社製 JTAG-HS1、または ByteTools 社製ダウンロード ケーブルを使用する場合、VREF (1.5 ～ 5.0V) および GND ヘッダーが、パラレル ケーブル IV への接続用に使用可能である必要があります。

ソフトウェア インストールおよびライセンス

ChipScope Pro Analyzer ソフトウェアは、ChipScope Pro Analyzer のみが必要なラボ環境などではスタンドアロン ISE ラボ ツールとしてインストール可能で、それ以外の場合 ISE Design Suite ツールの一部としてインストールできます。ソフトウェアのインストールおよびライセンスの手順は、ISE Design Suite 資料 [211 ページのリファレンス 14 を参照] の『ISE Design Suite 13 : インストールおよびライセンス ガイド』を参照してください。

XILINX®

第 2 章

コア生成ツールの使用方法

概要

この章では、ザイリンクス CORE Generator™ ツールを使用して ChipScope™ Pro コアを生成する手順を説明します。これらのコアは総称して ChipScope Pro ロジック デバッグ コアと言われます。

コアの生成後、CORE Generator ツールで生成されるインスタンシエーション テンプレートを使用して、これらのコアを VHDL または Verilog デザインに迅速かつ容易に挿入できます。インスタンシエーションを完了して、合成を実行した後は、ISE® インプリメンテーション ツールを使用してデザインをインプリメントできます。

ザイリンクス CORE Generator での ChipScope Pro コアの使用

ChipScope Pro コアを選択して生成する前に、CORE Generator ツールでプロジェクトを設定する必要があります。適切な設定でプロジェクトを設定した後、左上部のパネルの [View by Function] タブで [Debug & Verification] → [ChipScope Pro] を展開表示すると ChipScope Proコアが表示されます。また、[View by Name] タブでも ChipScope Pro コアを検索できます。

注記：ChipScope Pro コアのコア インスタンシエーション テンプレートは、.vho ファイル (VHDL 言語フロー用) と .veo ファイル (Verilog 言語フロー用) として提供され、コア生成プロセスの一部として作成されます。詳細は、CORE Generator ヘルプを参照してください。

注記 : ChipScope Pro コアの本質上、ザイリンクス CORE Generator ツールで生成される ChipScope Pro コア用のシミュレーション ファイルはシミュレーションで使用できません。ChipScope Pro コアを含むデザインをシミュレーションするときは、VHDL には空のブラック ボックス エンティティ アーキテクチャ、Verilog にはモジュールを使用する必要があります。

コアの生成

ICON、ILA、VIO、および ATC2 コアの生成

ICON、ILA、VIO、および ATC2 コアの生成方法は、次に示すデータシートに含まれています。データシートは、http://japan.xilinx.com/support の [IP] タブで [エンベデッド プロセッシング] → [デバッグとトレース] を表示すると入手できます。

- DS646 『LogiCORE IP ChipScope Pro Integrated Controller (ICON) (v1.05a)』
- DS299『LogiCORE IP ChipScope Pro Integrated Logic Analyzer (ILA) (v1.04a)』
- DS284 『LogiCORE IP ChipScope Pro Virtual Input/Output (VIO) (1.04a)』
- DS650 『Agilent Trace Core 2 (ATC2) (v1.04a)』

Virtex-5 FPGA 用 IBERT v1.0 コアの生成

IBERT Core Generator を使用すると、Virtex®-5 FPGA GTP/GTX トランシーバー用の IBERT v1.0 コアをカスタマイズして、生成できます。IBERT のパラメーターをすべて設定すると、ビットストリームを含む完全デザインが生成されます。この IBERT コアは、スタンドアロン デザインでのみ生成でき、ユーザー デザインに含めることはできません。デザインのネットリスト ファイル (.ngc または .edn) は生成されず、代わりに IBERT Core Generator により ISE が実行されてビットストリーム ファイル (.bit) が生成されます。

IBERT Core Generator の 1 ページ目で生成するコアのタイプに IBERT を選択します。[IBERT (Integrated Bit Error Ratio Tester)] を選択して [Next] をクリックします。

IBERT コアの標準オプションの設定

標準オプションは、IBERT Core Generator の 2 ページ目で設定します。

- ファイル保存先の選択

  IBERT ビットストリーム ファイル (ibert.bit) の保存先は、[Output Bitstream] に表示されます。デフォルトのディレクトリは、IBERT Core Generator のインストール パスになっていますが、任意のディレクトリに変更することもできます。デザインが生成されたら、すべてのインプリメンテーション ファイルが自動的にこのディレクトリに含められます。

- ターゲット デバイスの選択

  IBERT コアを生成する場合、デザインが ISE ツールで完全にインプリメントされるため、デバイス、パッケージ、およびスピード グレードを選択する必要があります。ChipScope Pro IBERT Core Generator ツールでは、Virtex-5 LXT/SXT/FXT ファミリがサポートされています。

- シリコン リビジョン (ステッピング レベル) の選択

  デバイス、パッケージ、およびスピード グレードの情報に加えて、Virtex-4 FPGA ファミリの適切なシリコン リビジョンを選択する必要があります。次に、Virtex-5 LXT/SXT/FXT ファミリで使用できるシリコン リビジョンを示します。

表 2-1 : Virtex-5 LXT/SXT/FXT ファミリのシリコン リビジョン (ステッピング レベル)

シリコン デバイス リビジョン	選択するシリコン リビジョン
すべてのエンジニアリング サンプル (ES) リビジョン	[CES]
すべての製品リビジョン	[Production]

IBERT クロック オプションの設定

IBERT コアの標準オプションを設定したら、[Next] をクリックします。

Virtex-5 LXT/SXT/FXT ファミリ用 IBERT クロック オプション

Virtex-5 LXT/SXT/FXT ファミリの IBERT コア デザインで唯一必要なクロック オプションは、次に説明する [System Clock Setting] のみです。

- システム クロックの設定

  IBERT コアでは、IBERT 制御ロジックのファブリック部分を駆動するのにフリーランニングのシステム クロックが必要です。システムク ロ ックソースの周波数は、10MHz ～ 100MHz にする必要があります。クロックは、内部で分周または倍周されて、50 ～ 100MHz になります。

システム クロックのオプションは、次の手順に従い設定します。

1. [System Clock Settings] フィールドを確認します。
2. [I/O Standard] で I/O 規格を選択します。
3. [P Source Pin] にシステム クロックのピン ロケーションを入力します。

注記：差動システム クロックの入力には、P ピンのロケーションのみ入力します。N のピン ロケーションは、ISE インプリメンテーション ツールで自動的に決定されます。

4. [Frequency] にシステム クロックの周波数 (MHz) を入力します。

注記：システムク ロックの周波数は、IBERT デザインが正しく動作するよう正確に入力する必要があります。このクロックは、ユーザー定義クロックで、MGT クロック選択とは別個のものであり、トランシーバーに関連するクロックから派生しません。

Virtex-5 LXT/SXT/FXT ファミリの GTP トランシーバー クロック構造は、現段階では図 2-1 に示すように固定されています。クロック構造は、Virtex-5 LXT/SXT/FXT ファミリの IBERT コア デザインでイネーブルにされている各 GTP_DUAL/GTX_DUAL タイルで複製されます。

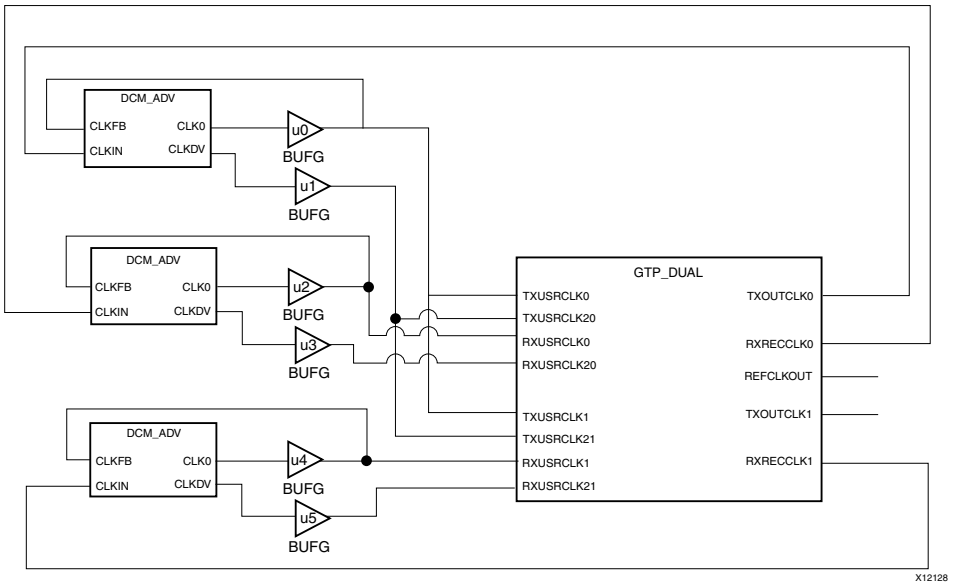

図 2-1 : Virtex-5 LXT/SXT/FXT ファミリ用 IBERT の GTP_DUAL/GTX_DUAL タイルのクロック構造

MGT/GTP/GTX オプションの選択

IBERT コアのクロック オプションを選択したら、[Next] をクリックして MGT/GTP/GTX オプション設定ページに進みます。

Virtex-5 LXT/SXT/FXT ファミリ用 IBERT の GTP/GTX オプション

Virtex-5 LXT/SXT/FXT ファミリ用 IBERT の GTP/GTX オプションには、次の 3 つのセクションがあります。

- [Resource Usage]
- [Pattern Settings]
- [GTP Settings]/[GTX Settings]

[Resource Usage]

[IBERT Options] ページの上部には、IBERT コアでのリソース使用量が表示されます。GTP_DUAL/GTX_DUAL タイルをオンにする度に、その分が使用量に追加されます。デジタル クロック マネージャー (DCM) の使用数も表示されます。DCM は、FPGA デバイスで使用可能な DCM 数までしか使用できません。

[Pattern Settings]

[Pattern Settings] には、すべての GTP_DUAL/GTX_DUAL タイルに対して生成、検出できるパターンが表示されます。使用できるパターン タイプは、PRBS (Pseudo Random Bit Sequence) 7 ビット ($X^7 + X^6 + 1$)、別の PRBS 7 ビット ($X^7 + X + 1$)、PRBS 9 ビット、PRBS 11 ビット、PRBS 15 ビット、PRBS 20 ビット、PRBS 23 ビット、PRBS 29 ビット、PRBS 31 ビット、ユーザー パターン (クロック パターンを含む、任意の 20 ビット データ パターンの生成に使用可能)、フレーム付きカウンター、およびアイドル パターンです。デフォルトでは、PRBS 7 ビット、PRBS 23 ビット、PRBS 31 ビット、およびユーザー パターンが選択されています。

[GTP Settings]/[GTX Settings]

GTP/GTX トランシーバー設定セクションでは、Virtex-5 LXT/SXT/FXT ファミリ デバイスの各 GTP_DUAL/GTX_DUAL タイルをイネーブルにして、それぞれ設定できます。GTP_DUAL/GTX_DUAL タイルをオンにする場合、最大ライン レートおよびリファレンス クロック周波数を指定する必要があります。最大ライン レートを設定すると有効なリファレンス クロック周波数 (FB、REF、および DIVSEL PLL 設定) のみが [Ref Clock Freq] に表示されます。

汎用 I/O (GPIO) オプションの設定

IBERT コアの GTP/GTX オプションを選択したら、[Next] をクリックして GPIO オプション設定ページに進みます。GPIO オプションでは、VIO コアで制御される、SFP 光モジュールなどの FPGA 外部のデバイスの制御に使用可能な同期出力ピンのみを設定できます。これらの出力は、IBERT システム クロックに同期しています。

- [Add VIO Controlled Output Pins]

  オンにすると、VIO 制御の出力ピンを IBERT デザインに追加できます。これにより、その他の GPIO 出力ピンの設定オプションも表示されます。

- [Number of Output Pins]

  デザインに追加する VIO 制御の同期出力ピン数を 1 ～ 256 で入力します。

- [Edit Output Pin Types Individually]

  GPIO 出力ピンのロケーションおよびその他の特性は、IBERT Core Generator で指定する必要があります。[Edit Output Pin Types Individually] をオンにすると、各 GPIO_OUT ピンのロケーション、I/O 規格、出力電流、およびスルー レートを制御できます。オフのままにする場合は、個別の出力ピンではなくすべてのピンに同じ設定を反映させることができます。

  [Pin Name]

  IBERT コアでは、GPIO_OUT[*n*] という名前の GPIO 出力ピンのみがサポートされており、*n* は GPIO_OUT バスのビット インデックスを示します。ピン名は変更できません。

  [Pin Loc]

  [Pin Loc] 列では、GPIO_OUT ピンのロケーションを設定します。

[IO Standard]

[IO Standard] では、各 GPIO_OUT ピンの I/O 規格を設定します。使用できる I/O 規格は、デバイス ファミリによって異なります。現段階では、IBERT GPIO 出力ピンではシングルエンドの I/O 規格のみがサポートされています。I/O 規格名は、ザイリンクス ソフトウェア マニュアル [211 ページのリファレンス 14 を参照] の『制約ガイド』の「IOSTANDARD」セクションに含まれている名前と一致しています。

[VCCO]

[VCCO] 列には、ピン ドライバーの出力電圧が表示されます。この値は、選択した I/O 規格によって異なります。

[Drive]

[Drive] 列には、出力ピンの最大駆動電流が 2mA ～ 24mA の範囲で表示され、これらの値は選択した I/O 規格によって異なります。

[Slew Rate]

[Slew Rate] 列では、各 GPIO_OUT ピンのスルー レートを [FAST] または [SLOW] のいずれかに設定できます。

サンプルおよびテンプレート オプションの設定

IBERT コアの GPIO オプションを選択したら、[Next] をクリックして [Example and Template Options] ページに進みます。

[Generate Batch Mode Argument Example File (.arg)] をオンにすると、バッチ モードの引数サンプル ファイル (ibert.arg など) を作成できます。この ibert.arg ファイルは、generate と呼ばれるコマンド ライン プログラムで使用されます。また、このファイルには、IBERT Core Generator ツールを使用せずに、IBERT デザインを作成するために必要なすべての引数が含まれています。

IBERT コアは、Windows の場合はコマンド プロンプトで ibertgenerate.exe <*ibert_type*> -f=ibert.arg」と入力し、Linux の場合は UNIX シェル プロンプトで「ibertgenerate.sh <*ibert_type*> -f=ibert.arg」と入力すると生成できます。<*ibert_type*> は、使用するデバイス ファミリに従い ibert、ibertgtp、または ibertgtx に置き換えてください。

[Output Log File Settings] では、出力ログ ファイルを生成するかどうかを決定します。また、この出力ログ ファイルに加えて、ISE インプリメンテーション ツール (ngdbuild、map、par) でも個々にレポート ファイルが生成されます。設定可能なオプションは、次のとおりです。

- [Generate Output Log File]
  - オンにすると、<*design name*>.log という名前の出力ログ ファイルが生成されます。このファイルには、IBERT デザイン生成プロセス中にメッセージ ペインに表示されたすべてのメッセージが含まれます。
  - オフの場合は、出力ログ ファイルは生成されません。
- [Verbose Log File]
  - オンにすると、生成プロセス中に ISE インプリメンテーション ツールからメッセージ ペインに出力されたメッセージすべてが含まれます。
  - オフの場合は、ツールによる出力が圧縮され、通常は生成にかかるランタイムが短縮されます。

デザインの生成

IBERT コアのパラメーターの入力が終了したら、[Generate Design] をクリックし、カスタマイズした FPGA デザインを作成するのに必要なすべてのファイルを実行します。メッセージ ウィンドウが開き、進捗情報が表示されます。「IBERT Design Generation Completed」と表示されると、コア生成プロセスの完了です。この後、前の画面に戻り、異なるオプションを指定してコアを生成し直したり、または [Start Over] をクリックして、新規コアを生成できます。

注記 : IBERT の生成では、ISE ツールすべてが実行されるため、その他の ChipScope コアにかかる生成時間よりも長くなります。多くの GTP/GTX を使用するデザインでは、使用するコンピューターの速度によっては生成に何時間もかかる場合があります。

IBERT v2.0 コアの生成

IBERT v2.0 コアの生成方法は、次に示すデータシートに含まれています。データシートは、http://japan.xilinx.com/support の [IP] タブをで [エンベデッド プロセッシング] → [デバッグとトレース] を表示すると入手できます。

- DS774 『 Virtex-5 GTX 用 ChipScope Pro IBERT (Integrated Bit Error Ratio Test)』
- DS732 『Virtex-6 GTX 用 ChipScope Pro IBERT (Integrated Bit Error Ratio Test)』
- DS775 『Virtex-6 GTH 用 ChipScope Pro IBERT (Integrated Bit Error Ratio Test)』
- DS782 『Spartan-6 GTP 用 ChipScope Pro IBERT (Integrated Bit Error Ratio Test)』
- DS855 『Kintex-7 GTX 用 ChipScope Pro IBERT (Integrated Bit Error Ratio Test) (v2.00.a)』
- DS867 『Virtex-7 GTX 用 ChipScope Pro IBERT (Integrated Bit Error Ratio Test) (v2.00.a)』

XILINX®

第 3 章

ChipScope Pro Core Inserter の使用

ChipScope Pro Core Inserter の概要

ChipScope™ Pro Core Inserter は合成後に使用し、パラメーターを設定した ICON (Integrated CoNtroller)、ILA (Integrated Logic Analyzer)、および ATC2 (Agilent Trace Core) コアを必要に応じてデザインに挿入してネットリストを生成するツールです。ChipScope Pro Core Inserter では、デバッグ機能を迅速かつ容易に使用して HDL をインスタンシエートせずに、合成されたデザインを解析できます。

注記：VIO (Virtual Input/Output) および IBERT (Internal Bit Error RaTio) コアは、ChipScope Pro Core Inserter ツールでサポートされていません。

PlanAhead での ChipScope Pro Core Inserter の使用

ChipScope Pro Core Inserter は、PlanAhead™ ツールと組み合わせて使用するようには設計されていません。PlanAhead ツール環境には、デバッグ コアを挿入する機能が組み込まれています。ChipScope Pro Core Inserter ツールから ISE® PlanAhead ツール環境に簡単に移行できるよう、PlanAhead ツールには CDC インポート コマンドが提供されています。このコマンドでは、ChipScope Pro Core Inserter のプロジェクト ファイルを PlanAhead ツール プロジェクトにインポートできます。PlanAhead ツールでの ChipScope コアの挿入に関する詳細は、『PlanAhead ユーザー ガイド』[211 ページのリファレンス 17 を参照] を参照してください。

ISE Project Navigator での ChipScope Pro Core Inserter の使用

このセクションは、Windows または Linux バージョンの ChipScope Pro および ISE ツールを使用しているユーザーを対象にしています。

ChipScope Pro Core Inserter の CDC ファイルは、Project Navigator のソース ファイル リストに新規ソース ファイルとして追加できます。また、Project Navigator では、インプリメンテーション フローの適切な段階で ChipScope Pro Core Inserter ツールを認識し起動させることが可能です。Project Navigator と ChipScope Pro Core Inserter の統合に関する詳細は、ISE ソフトウェア マニュアル [211 ページのリファレンス 14 を参照] の Project Navigator セクションを参照してください。

ChipScope の定義および接続ソース ファイル

ChipScope Pro Core Inserter ツールを使用して Project Navigator で処理されるデザインにコアを挿入するには、次の手順に従います。

1. CDC (.cdc) ファイルをプロジェクトに追加し、該当するデザイン モジュールと関連付けます。
   a. 新規の CDC ファイルを作成するには、[Project] → [New Source] → [ChipScope Definition and Connection File] をクリックし、ファイル名を入力します。[Next] をクリックして [Summary] ページに進み、[Finish] をクリックしてファイルを作成します。

      メモ：Project Navigator で同一バージョンの ChipScope Pro のインストールが認識された場合のみ、ソース タイプとして [ChipScope Definition and Connection File] が表示されます。

   b. 既存の CDC ファイルを追加するには、[Project] → [Add Source] または [Project] → [Add Copy of Source] をクリックして、既存の CDC ファイルを参照します。

      CDC ファイルをブラウザーで選択して [開く] をクリックし、[Adding Source Files] ダイアログ ボックスで [OK] をクリックします。CDC ファイルは、ソース ウィンドウで関連デザイン モジュールの下に表示されます。

2. コアを作成し、信号接続を完了するには、[Design] パネルに表示されるこの CDC ファイルをダブルクリックします。この後、必要な場合は合成と変換が実行され、ChipScope Pro Core Inserter で CDC ファイルが開きます。
3. 必要に応じてコアおよび接続を変更して (44 ページの「ChipScope Pro Core Inserter の機能」で説明) ツールを終了します。
4. Project Navigator で関連する最上位デザインがインプリメントされると、変換フローの一部としてコアが自動的にデザインのネットリストに挿入されます。この操作を実行するために、プロパティを設定する必要はありません。コアは、CDC ファイルがプロジェクト内にあり、インプリメントされるデザイン モジュールに関連付けられていると自動的に挿入されます。

有用な Project Navigator の設定

次の Project Navigator 設定を使用すると、コアをデザインにインプリメントする際に役立ちます。

1. XST 合成ツールを使用している場合は、[Keep Hierarchy] オプションを [Yes] または [Soft] に設定してデザイン階層を維持し、デザインすべての階層の最適化を回避します。[Keep Hierarchy] オプションを使用した場合、コアの挿入フロー中のネット名およびその他のコンポーネント名を維持できます。[Keep Hierarchy] オプションを使用しない場合は、ネット/コンポーネントがほかのロジックと結合されて新しいコンポーネントが作成されるか、または最適化されて削除される可能性があります。デザイン階層を維持するには、次の手順に従います。
   a. [Processes] ペインで [Synthesize - XST] を右クリックして [Process Properties] をクリックします。
   b. [Synthesis Options] ページで [Keep Hierarchy] を [Soft] または [Yes] に設定して、[OK] をクリックします。
2. ChipScope Pro Analyzer ツールで古いデバイスを使用する場合は、次に示すようにビットストリームの生成オプションを正しく生成する必要があります。次に、これらの手順に従う必要があるデバイスを示します。
   - Virtex®-4 デバイス(またはそれ以前のデバイス)
   - Spartan®-3、Spartan-3E、Spartan-3A、Spartan-3AN (またはそれ以前の) デバイス

a. Project Navigator で [Generate Programming File] を右クリックして、[Process Properties] を選択します。

b. [Startup Options] ページを表示します。

c. [FPGA Start-Up Clock] に [JTAG Clock] を選択します。

注記：このオプションは、新しい FPGA デバイス ファミリでは不要です。また、FPGA デバイスを PROM またはフラッシュ デバイスからコンフィギュレーションする場合も不要です。

コマンド ライン インプリメンテーションでの ChipScope Pro Core Inserter の使用

コマンド ライン フローの概要

ChipScope Pro Core Inserter では、バッチ コア挿入の基本的なコマンド ラインがサポートされています。図 3-1 に示すように、ChipScope Pro Core Inserter のコマンド ライン フローは、次の手順で構成されています。

1. CDC プロジェクトの作成
2. CDC プロジェクトの変更
3. コアの挿入

ChipScope Pro Core Inserter は、NGDBuild の前に実行し、デザインにデバッグ コアをインスタンシエートします。デバッグするネットは CDC プロジェクト変更ステップで選択して GUI に表示し、プロジェクトに保存できます。デザインを正しくインプリメントし、そのビットストリームを使用してザイリンクス デバイスをコンフィギュレーションした後には、ChipScope Pro Analyzer を使用してインサーキット デザインのデバッグおよび検証を実行します。コンフィギュレーション後にデバッグ ネットの選択およびコアの設定を変更する場合は、CDC プロジェクトの変更ステップに戻ってフローを実行し直して、新しいビットストリームを生成し、デバッグおよび検証を実行してください。

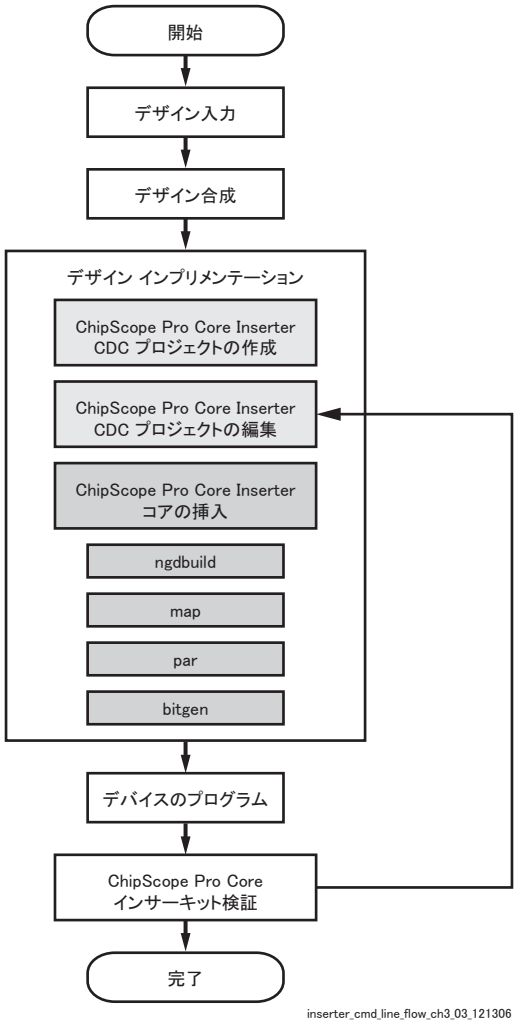

図 3-1： ChipScope Pro Core Inserter のコマンド ライン フロー

CDC プロジェクトの作成

図 3-2 に示すように、空の CDC プロジェクトファイルを作成します。コマンド ラインには、次のように入力します。

```
inserter -create <project.cdc>
```

注記：この手順では、ChipScope Pro Core Inserter の GUI は起動しません。

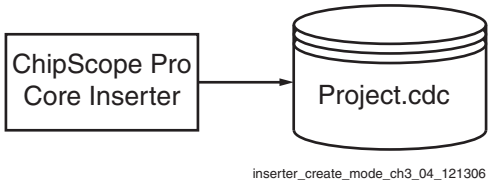

図 3-2 : CDC プロジェクトの作成

CDC プロジェクトの編集

この手順では、ChipScope Pro Core Inserter の GUI を起動して既存の CDC プロジェクトを変更します (図 3-3)。この手順では、NGCBuild ツールが特定の引数と共に呼び出されます。NGCBuild ツールでは、デザインに関連するネットリストすべてが 1 つの NGC ネットリスト ファイルに統合されます。これにより、ChipScope Pro Core Inserter でデザイン内のすべてのレベルおよびノードに対して完全にデバッグを実行できます。

コマンド ラインには、次のように入力します。

```
inserter -edit <project.cdc> -ngcbuild [-p <partname>] [{-sd
<source_dir>}] [-dd <output_dir>] [-i] <inputdesign.{edn|ngc}>
<outputdesign.ngc>
```

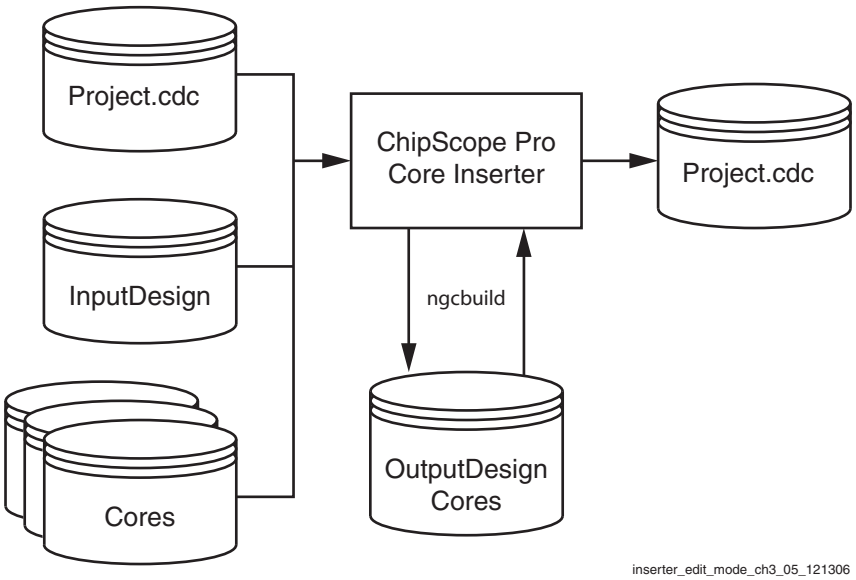

図 3-3 : CDC プロジェクトの変更

コアの挿入

この手順では、既存の CDC プロジェクトに基づきデザインにコアを挿入します (図 3-4)。この手順では、NGCBuild ツールが特定の引数と共に呼び出されます。NGCBuild ツールでは、デザインに関連するネットリストすべてが 1 つの NGC ネットリスト ファイルに統合されます。コアは、1 つの完全ネットリストに挿入されます。

コマンド ラインには、次のように入力します。

```
inserter -insert <project.cdc> -ngcbuild [-p <partname>] [{-sd
<source_dir>}] [-dd <output_dir>] [-i] <inputdesign.{edn|ngc}>
<outputdesign.ngc>
```

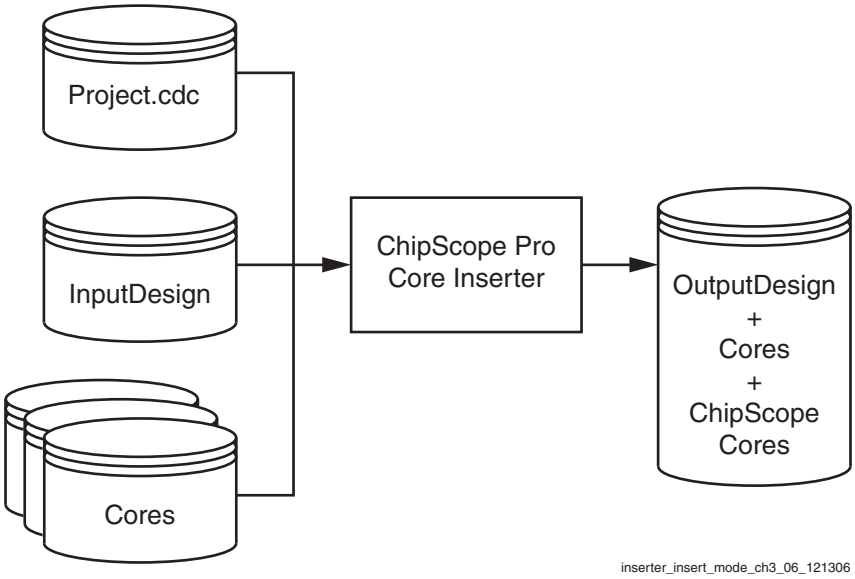

図 3-4 : コアの挿入

ChipScope Pro Core Inserter の機能

プロジェクトでの作業

ChipScope Pro Core Inserter で保存したプロジェクトには、ソース ファイル、デスティネーション ファイル、コアのパラメーター、およびコアの設定に関連する情報がすべて含まれます。このため、このファイルを使用して、コアの挿入に関する情報を保存および再現できます。また、プロジェクト ファイル (.cdc) を、ChipScope Pro Analyzer への入力として使用して、信号名をインポートできます。

ChipScope Pro Core Inserter を起動したときに表示される初期画面はすべて空欄になっています。また、[File] → [New] をクリックしても同じ画面を表示できます。

既存のプロジェクトを開く

既存のプロジェクトを開く場合、最近開いたプロジェクト リストから、または [File] → [Open Project] をクリックして [Browse] でプロジェクト ディレクトリに移動します。プロジェクトを選択してダブルクリックするか、または [開く] をクリックします。

プロジェクトの保存

プロジェクトを変更した場合、ChipScope Pro Core Inserter を終了する前にプロジェクトの保存を尋ねるダイアログ ボックスが表示されるので、保存してください。また、[File] → [Save] からも保存できます。プロジェクト名を変更する場合、または別のファイル名で保存する場合は、[File] → [Save As] をクリックし、ファイル名を入力してから [保存] をクリックします。

ネットリストの更新

ChipScope Pro Core Inserter で前回読み込んだときからネットリストが変更されている場合は、自動的にリロードされます。ただし、[File] → [Refresh Netlist] をクリックすると、手動でネットリストを更新することもできます。

ユニットの挿入および削除

プロジェクトに新規ユニットを挿入する場合は、[Edit] → [New ILA Unit] または [Edit] → [New ATC2 Unit] をクリックします。ユニットを削除する場合は、ユニットを選択して [Edit] → [Remove Unit] をクリックします。

プリファレンスの設定

ChipScope Pro Core Inserter プロジェクトのプリファレンスを設定する場合は、[Edit] → [Preferences] をクリックします。プリファレンスは、[Tools]、[ISE Integration]、および [Miscellaneous] の 3 つのセクションに分かれています。プリファレンスの設定の詳細は、54 ページの「プロジェクトのプリファレンス設定」を参照してください。

コアの挿入

ICON、ILA、および ATC2 コアの挿入は、フロー完了後に実行されます。または、[Insert] → [Insert Core] をクリックします。キャプチャしたすべてのコアが有効な信号に接続されていない場合は、エラー メッセージが表示されます。

ChipScope Pro Core Inserter の終了

ChipScope Pro Core Inserter を終了するには、[File] → [Exit] をクリックします。

入力および出力ファイルの指定

ChipScope Pro Core Inserter は、次の手順で実行します。

1. [Input Design Netlist] で入力デザイン ネットリストを指定します。
2. [Browse] をクリックしてネットリストのディレクトリを参照します。
3. 必要に応じて、[Output Design Netlist] および [Output Directory] を変更します。これらフィールドは、デフォルトで自動入力されています。

注記：ChipScope Pro Core Inserter を Project Navigator から起動した場合、[Input Design Netlist]、[Output Design Netlist]、[Output Directory and Device Family] フィールドは自動入力されます。この場合、これらのフィールドは Project Navigator でのみ変更でき、ChipScope Pro Core Inserter では変更できません。

プロジェクト レベルのパラメーター

プロジェクトでは、3 つのパラメーター (デバイス ファミリ、SRL 使用の有無、RPM 使用の有無) を設定する必要があります。

ターゲット デバイス ファミリの選択

[Device Family] でターゲット デバイス ファミリを指定します。ここで選択したデバイス ファミリ用に、ICON およびキャプチャ コアの構造が最適化されます。プルダウン リストからデバイス ファミリを選択してください。

デフォルトでは、[Virtex-6] が選択されています。

SRL の使用

[Use SRLs] では、コアを SRL16/SRL32 コンポーネントを使用して生成するかどうかを選択します。このオプションをオフにした場合は、SRL16 コンポーネントの代わりにフリップフロップおよ

びマルチプレクサーが使用されます。この場合、生成されたコアのサイズやパフォーマンスに影響します。

デフォルトでは、[Use SRLs] がオンになっており、SRL を使用してコアが生成されます。

RPM の使用

[Use RPMs] をオンにすると、コアが相対配置マクロ (RPM) にされます。オンのときは配置配線ツールで、コアのすべてのロジックが最適化されて 1 つのエリアに配置されます。デバイスのほとんどのリソースを使用するデザインでは、これらの配置制約は満たされない可能性があります。デフォルトでは、[Use RPMs] がオンになっており、配置が最適化されたコアが生成されます。すべて設定したら、[Next] をクリックします。

コアの使用量

ChipScope Pro Core Inserter の左側にある [Core Utilization] パネルでは、デザインのネットリストに挿入する ChipScope コアで消費されるルックアップ テーブル (LUT)、フリップフロップ (FF)、およびブロック RAM (BRAM) の概算数が表示されます。コアのリソース使用量は、コアの構成に影響するパラメーターの設定に基づいて更新されます。

注記：[LUT Count] および [FF Count] は、Spartan-3、Spartan-3E、Spartan-3A、Spartan-3A DSP、および Virtex-4 デバイス ファミリでのみ表示されます。[BRAM Count] は、すべてのデバイス ファミリで表示されます。

ICON オプションの選択

最初に指定する必要があるオプションは、ICON コアです。ICON コアは、すべての ILA コアおよび ATC2 コアを JTAG (Joint Test Action Group、IEEE 規格) バウンダリ スキャン チェーンと接続するコントローラー コアです。

バウンダリ スキャン チェーンの選択

ChipScope Pro Analyzer では、USER1、USER2、USER3、または USER4 のいずれかのバウンダリ スキャン チェーンを使用してコアと通信できます。[Boundary Scan Chain] リストから任意のスキャン チェーンを選択してください。このオプションは、Spartan-3、Spartan-3E、Spartan-3A、および Spartan-3A DSP デバイス ファミリを使用する場合は使用できません。

ILA のトリガー オプションおよびパラメーターの選択

[New ILA Unit] ボタンをクリックすると、新しい ILA ユニットが左側のデザイン階層に追加されます。次に ILA ユニットを設定する必要があります。最初のタブ パネル ([Trigger Parameters]) では、ILA コアのトリガー オプションを設定します。

[Number Of Trigger Ports]

各 ILA コアでは、最大 16 個のトリガー ポートを個別に設定できます。[Number of Input Trigger Ports] のプルダウン リストからトリガー ポート数を選択すると、各ポートのオプション グループが表示されます。各トリガー ポートに関連するオプショングループには TRIG*n* というグループ名が付いています (*n* は 0 ～ 15 までのトリガー ポート番号です)。このオプションには、トリガー幅、トリガー ポートに接続されている比較ユニット数、比較ユニットタイプなどがあります。

[Trigger Width]

各トリガー ポートは、信号またはビットで構成されてるバスです。トリガー ポートを構成するビット数を、トリガー幅といいます。各トリガー ポートの幅は、TRIG*n* グループ オプションの [Trigger Width] で設定できます。トリガー ポート幅には、1 ～ 256 を設定できます。

[# Match Units]

比較ユニットは、トリガー ポートと接続しているコンパレーターであり、トリガー ポートのイベントを検出します。1 つまたは複数の比較ユニットの結果を結合して総体的なトリガー条件が形成され、データ キャプチャが制御されます。各トリガー ポート (TRIG*n*) に接続する比較ユニット数は、[# Match Units] リストから設定できます。

比較ユニット数を 1 に設定する場合は、トリガー イベントの検出での柔軟性を多少残しながら、リソースを節約できます。2 以上に設定する場合は、その結合数が大きいほど、より柔軟性の高いトリガー条件を形成できます。ただし、各トリガー ポートの比較ユニット数を増やすと、ロジック リソース使用量も増加します。

注記 : 1 つの ILA コアで使用する比較ユニットの総計数は、使用するトリガ ポート数に関係なく、最大 16 個までです。

[Match Type]

トリガ ポートの比較ユニットが実行できるさまざまな比較や比較演算子は、比較ユニット タイプによって異なります。ILA コアでは、6 個の比較ユニット タイプをサポートしています (表 3-1)。

表 3-1 : ILA のトリガー ポートの比較ユニット タイプ

タイプ	ビット値[1]	比較関数	スライスごとのビット数[2]	説明
[Basic]	0、1、X	=、<>	LUT4 ベース : 8 Virtex-5、Spartan-6 : 19 その他の LUT6 ベース : 20	遷移検出が重要ではないデータ信号を比較するために使用します。最もビットを節約できる比較ユニット タイプです。
[Basic w/edges]	0、1、X、R、F、B、N	=、<>	LUT4 ベース : 4 LUT6 ベース : 8	遷移検出 (例 : Low から High、High から Low など) が重要となる制御信号の比較に使用します。
[Extended]	0、1、X	=、<>、>、>=、<、<=	LUT4 ベース : 2 LUT6 ベース : 16	大きさ (大小) が重要となるアドレスまたはデータ信号の比較に使用します。
[Extended w/edges]	0、1、X、R、F、B、N	=、<>、>、>=、<、<=	LUT4 ベース : 2 LUT6 ベース : 8	大きさ (大小) が重要となるアドレスまたはデータ信号の比較に使用します。
[Range]	0、1、X	=、<>、>、>=、<、<=、in range、not in range	LUT4 ベース : 1 LUT6 ベース : 8	値の範囲が重要となるアドレスまたはデータ信号の比較に使用します。

表 3-1 : ILA のトリガー ポートの比較ユニット タイプ (続き)

タイプ	ビット値[1]	比較関数	スライスごとのビット数[2]	説明
[Range w/edges]	0、1、X、R、F、B、N	=、<>、>、>=、<、<=、in range、not in range	LUT4 ベース : 1 LUT6 ベース : 4	値の範囲と遷移検出が重要となるアドレスまたはデータ信号の比較に使用します。

メモ :

1. ビット値 : 0 は論理値 0、1 は論理値 1、X はドントケア、R は 0 から 1 への遷移、F は 1 から 0 への遷移、B は任意の遷移、N は遷移なしを指します。
2. スライスごとのビット数は、各比較ユニット タイプの相対的なリソース使用数を示すための概算値です。正確な概算値として使用しないでください。LUT4 ベースのデバイス ファミリには、Spartan-3、Spartan-3E、Spartan-3A、Spartan-3A DSP、および Virtex-4 FPGA が含まれます。LUT6 ベースのデバイス ファミリには、Virtex-5、Virtex-6、Spartan-6、Artix™-7、Kintex™-7、および Virtex-7 FPGA が含まれます。

TRIG*n* の [Match Type] リストから比較ユニットのタイプを選択すると、このトリガー ポートに接続されるすべての比較ユニットにこの設定が適用されます。比較ユニットの機能性が高くなると、機能のインプリメントに必要なリソースも増加することに注意してください。このように設定に柔軟性があることから、リソース使用量を確認しながらトリガー モジュールの機能をカスタマイズできます。

[Counter Width]

比較ユニット カウンターは、トリガー ポートの各比較ユニットの出力に接続されているコンフィギュレーション可能なカウンターです。このカウンターは実行時に設定でき、比較ユニットのイベント数をカウントします。トリガー ポートの各比較ユニットに比較カウンターを含めるには、[Counter Width] でカウンター幅を 1 ～ 32 の中から選択します。[Counter Width] を [Disabled] に設定すると、比較ユニットに比較カウンターは含まれません。デフォルト設定は、[Disabled] です。

[Enable Trigger Sequencer]

トリガー条件シーケンサーは、一般的なブール式のトリガー条件であり、[Enable Trigger Sequencer] をオンにすると、オプションのトリガー シーケンサーと共に使用できます。図 3-5 に、トリガー シーケンサーのブロック図を示します。

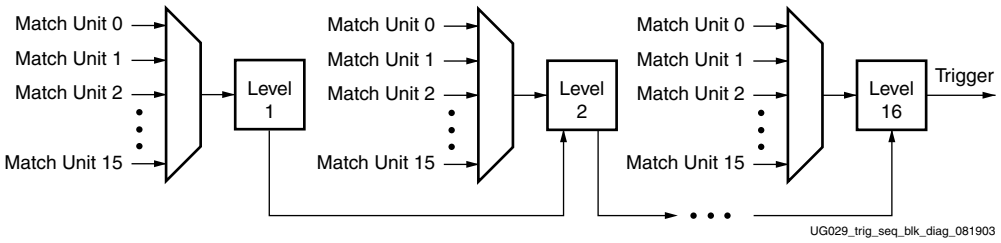

図 3-5 : トリガー シーケンサーのブロック図 (16 ステート、16 比較ユニット)

トリガ シーケンサーは、循環型のステート マシンとしてインプリメントされ、トリガー条件が満たされるまで最大 16 ステート (レベル) まで遷移可能です。ステート遷移は、トリガー シーケンサーと接続されている比較ユニットのイベントによって発生します。どの比較ユニットも、実行時にステート遷移をレベルごとに選択できます。トリガー シーケンサーは、比較関数イベントのシーケンスが連続または不連続のどちらの場合でも、実行時にステート遷移を設定できます。

[Enable Storage Qualification]

ILA コアは、トリガー条件に加えて、ストレージ必要条件の設定も可能です。ストレージ必要条件は、比較関数イベントのブール式の組み合わせです。これらの比較関数イベントは、コアのトリガー ポートに接続されている比較ユニット コンパレーターで検出されます。ストレージ必要条件は、トリガー条件とは異なり、トリガー ポートの比較ユニットのイベントを検証し、各データ サンプルのキャプチャまたは格納を決定します。トリガー条件とストレージ必要条件を併用すると、キャプチャ プロセスを開始するタイミングやキャプチャするデータを定義できます。このストレージ必要条件をイネーブルにする場合は、[Enable Storage Qualification] をオンにします。

[Enable Trigger Output Port]

ILA コアのトリガー条件モジュールのトリガ出力ポートは、ChipScope Pro Core Inserter ではイネーブルにできません。このポートは、CORE Generator™ でのみイネーブルにできます。

ILA コアのキャプチャ パラメーターの設定

ILA コアのキャプチャ パラメーターは、ChipScope Pro Core Inserter の 2 番目のタブ ([Capture Parameters]) で設定します。

[Data Depth]

ワード数とは、ILA コアがサンプル バッファーに格納できる最大データ サンプル ワードを指します。このワード数により、ILA ユニットで使用される各ブロック RAM のデータ幅が決定します。CORE Generator および ChipScope Pro Core Inserter には、リソース使用量予測機能があり、特定のデータ幅およびワード数設定で使用されるブロック RAM リソース数が示されます。

[Data Width]

データ幅とは、ILA コアに格納されている各データ サンプル ワード幅を指します。データ ワードとトリガー ワードがそれぞれ独立している場合、最大許容データ幅はターゲット デバイス タイプおよびワード数によって異なります。ただし、どの組み合わせでも最大許容データ幅は 4096 ビット (Spartan-3、Spartan-3E、Spartan-3A、Spartan-3A DSP、および Virtex-4 デバイスでは 256 ビット) です。

[Data Same As Trigger]

ILA トリガー ポートでキャプチャされるデータが次の 2 つで制御されます。

- オンの場合 :
  - トリガー ポートに接続されている信号をトリガーおよびデータ キャプチャで使用できます。このモードは、コアのトリガーに使用するデータをキャプチャ /修正できるため、ロジック解析ツールでよく使用されます。
  - データ ポートに各トリガー ポートを含むように選択できます。この設定の場合、ILA コアのポート マップに DATA 入力ポートは含まれません。
  - このモードでは、ILA コアで使用する CLB および配線リソースを節約できますが、最大データ サンプル ワード幅は 4096 ビット (Spartan-3、Spartan-3E、Spartan-3A、Spartan-3A DSP、および Virtex-4 デバイスでは 256 ビット) に制限されます。
- オフの場合 :
  - データ ポートとトリガー ポートが互いに独立しています。
  - このモードは、キャプチャするデータ量を制限する場合に役立ちます。

- データがトリガーと異なる場合は、[Data Width] を指定する必要があります。

[Trigger Ports Used As Data]

[Data Same As Trigger] をオンにすると、データ オプション画面に各 TRIG*n* ポートのチェックボックスが表示されます。データ ポートに含める場合は、これらのトリガー ポートのチェックボックスをオンにします。最大データ幅の 4096 ビット (Spartan-3、Spartan-3E、Spartan-3A、Spartan-3A DSP、および Virtex-4 デバイスでは 256 ビット) がすべてのトリガー ポートに適用されます。

ATC2 のデータ キャプチャ設定

ATC2 コアを挿入する場合は、次の ATC2 データ キャプチャ オプションを設定する必要があります。

[Capture Mode]

ATC2 コアのキャプチャ モードは 2 種類あり、CLK 入力信号に対して同期データをキャプチャする STATE モードと、非同期データをキャプチャする TIMING モードのいずれかを設定できます。STATE モードの場合、ATC2 コアを通るデータ パスでパイプライン化されたフリップフロップが使用され、これらのフリップフロップのクロックは、CLK 入力ポート信号により供給されます。TIMING モードの場合、ATC2 コアを通るデータ パスは出力ピンに至るまで完全に組み合わせロジックで構成されます。また、TIMING モードの ATCK ピンは追加データ ピンとして使用されます。

[Max Frequency Range]

[Max Frequency Range] のパラメーターを選択して、ATC2 コアの動作周波数の値を指定します。ATC2 コアのインプリメンテーションは、選択した最大周波数範囲に最適化されます。選択可能な最大周波数範囲は、[0 ～ 100 MHz]、[101 ～ 200 MHz]、[201 ～ 300 MHz]、および [301 ～ 500 MHz] です。最大周波数範囲の選択は、[Capture Mode] で [STATE] を選択した場合のみ、コアのインプリメンテーションに影響します。

[Enable Auto Setup]

[Enable Auto Setup] をオンにすると、アジレント テクノロジー社のロジック アナライザーによりポッド接続に最適な ATC2 ピンを自動的に設定できます。また、各 ATC2 ピンに最適な位相および電圧サンプリング オフセットも自動的に決定されます。このオプションは、デフォルトでイネーブルにされています。

[Enable "Always On" Mode]

オンにすると、ATC2 コアの内部ロジックおよび出力バッファーが常にイネーブルにされます。オンの場合、FPGA デバイスのコンフィギュレーション時に信号バンク 0 により ATD ピンが駆動されます。このモードでは、最初に ATC2 コアを手動で設定せずに、デバイス コンフィギュレーション直後に発生するイベントをキャプチャできます。このオプションは、デフォルトでディスエーブルに設定されており、キャプチャ モードが [TIMING] モードの場合のみ使用できます。

[Pin Edit Mode]

ピンの I/O 規格ピン、駆動電流、およびスルー レートを個別に設定するか、またはグループとして設定するかを選択できます。[Individual] に設定すると、各ピンのパラメーターを個別に設定できます。また 、[Same as ATCK] に設定すると、すべての ATCK ピンが同じパラメーターになります。各ピンの配置は、[Pin Edit Mode] の設定に関わらず、それぞれ設定する必要があります。

[ATD Pin Count]

ATC2 コアは、ATD 出力ピンを 4 ～64 個の範囲内でインプリメントできます。

[Endpoint Type]

[Endpoint Type] では、ATCK および ATD 出力ピンでシングル エンド ([SINGLE-ENDED]) または差動出力 ([DIFFERENTIAL]) ドライバーのいずれかを選択します。すべての ATCK および ATD ピンで同一のドライバー終端タイプを使用する必要があります。

[Signal Bank Count]

ATC2 コアには、実行時に選択可能な内部データ信号バンク マルチプレクサーが含まれています。[Signal Bank Count] には、このマルチプレクサーでインプリメントするデータ入力ポート数/信号バンク数を入力します。設定可能な値は、1、2、4、8、16、32、64 です。

[TDM Rate]

TDM (時分割多重) レートの設定では、各データ ピンに伝送されるデータ量を 200% に増加させることができます。ATC2 コアは、キャプチャしたトレース データの格納にオンチップ メモリ リソースを使用しません。その代わりに、専用コネクタを使用して FPGA ピンに接続しているアジレント テクノロジー社の Logic Analyzer へデータを送ります。TDM レートを 1X に設定した場合、データがデバイス ピンへ伝送される速度は、DATA ポートへの入力と同一ですが、2X に設定すると、DATA ポートの 2 倍速になります。TDM レートは、キャプチャ モードが [STATE] の場合のみ 2X に設定できます。

[Data Width]

ATC2 コアの各入力データ ポート幅は、キャプチャ モードおよび TDM レートによって異なります。STATE モードの場合、各データ ポート幅は、(ATD ピン数) * (TDM レート) です。TIMING モードの場合、ATCK ピンは追加データ ピンとして使用されるため、各データ ポート幅は、(ATD ピン数 + 1) * (TDM レート) になります。

[Individual Pin Settings]

[Individual Pin Settings] 表では、各 ATCK および ATD ピンのロケーション、I/O 規格、出力駆動電流、およびスルー レートを設定します。出力クロック (ATCK) およびデータ (ATD) ピンは、あらかじめ ATC2 コアにインスタンシエートされています。つまり、ATCK および ATD ピンは、ほかのデザイン階層から最上位に手動で移動する必要はありませんが、CORE Generator でこれらのピンのロケーションおよび特性を指定する必要があります。これらのピン属性は、ATC2 コアの *.ncf ファイルに追加されます。

[Pin Name]

ATC2 コアの出力ピンには ATCK および ATD の 2 種類があります。ATCK ピンは、キャプチャ モードが [STATE] の場合はクロック ピンとして使用され、[TIMING] の場合はデータ ピンとして使用されます。ATD ピンは、常にデータ ピンとして使用されます。ピン名は変更できません。

[Pin Loc]

[Pin Loc] では、ATCK または ATD ピンのロケーションを設定します。

[IO Standard]

[IO Standard] では、各 ATCK または ATD ピンの I/O 規格を設定します。設定可能な I/O 規格は、デバイス ファミリおよびドライバーの終端タイプによって異なります。I/O 規格名は、ザイリンクス ソフトウェア マニュアル [211 ページのリファレンス 14 を参照] の『制約ガイド』の「IOSTANDARD」セクションに含まれている名前と一致しています。

[VCCO]

[VCCO] 列には、ピン ドライバーの出力電圧が表示されます。この値は、選択した I/O 規格によって異なります。

[Drive]

[Drive] には、ピン ドライバーの最大出力駆動電流が 2 ～ 24mA の範囲内で示されており、この値は選択した I/O 規格に依存しています。

[Slew Rate]

[Slew Rate] では、各 ATCK または ATD ピンのスルー レートを [FAST] または [SLOW] のいずれかに設定できます。

[Core Utilization]

ATC2 コア ジェネレーターには、コアのリソース使用量を監視する機能があり、ATC2 コアが使用するルックアップ テーブル (LUT) 数およびフリップフロップ数が予測されます。この値は、パラメーター設定に依存しています。ATC2 コアは、ブロック RAM または追加クロック リソース (BUFG、DCM コンポーネントなど) を使用しません。

ILA 信号のネット接続の選択

[Net Connections] タブでは、ILA コアに接続する信号を選択します。トリガーとデータが異なる場合は、クロック、トリガー、データ ポートをそれぞれ指定する必要があります。トリガーとデータが同じ場合は、クロックとトリガー/データ ポートを指定します。[CLOCK PORT] をダブルクリックまたはプラス記号 (+) マークを展開します。未接続の場合は、赤色で表示されます。

ATC2 の [Net Connections] タブでは、ATC2 コアに接続する信号を選択できます。クロックおよびデータ ポートを指定する必要があります。[Clock Net] を展開表示します。未接続の場合は、赤色で表示されます。

コア接続を変更する場合は、[Modify Connections] をクリックします。[Select Net] 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、ILA または ATC2 コアに接続するネットを簡単に選択できるインターフェイスです。[Select Net] ダイアログ ボックスの左上にある [Structure/Nets] ペインで、デザインの階層構造を確認できます。左下ペインには、選択した階層レベルのすべてのネットが表示されます。この表に含まれる情報は次のとおりです。

- [Net Name] : ネット名が EDIF ネットリストでの記載と同様に表示されます。このネット名は、合成プロセスでの名前変更または最適化の結果、HDL ソースの対応する信号名と異なる場合があります。
- [Source Instance] : 現在の階層のネットが駆動される下位階層にあるコンポーネントのインスタンス名です。このソース インスタンスは、必ずしもネットのドライバー元を示しているのではありません。
- [Source Component] : ソース インスタンスで定義されたコンポーネント タイプが表示されます。
- [Base Type] : ネットを駆動する最下位のコンポーネントが表示されます。この列には、プリミティブかブラックボックス コンポーネントのいずれかが表示されます。

表の上に表示されているドロップダウンから検索対象を選択し、[Pattern] ボックスに検索するネット識別子の文字列を入力してから [Filter] をクリックすると、検索対象のみを表に表示できます。また、表中の各コラム (ネット識別子) のヘッダーをクリックすると、ネット識別子の順番が降順/昇順に切り替わります。

注記：ネット名はアルファベット順またはバス エレメント順に並び替えられます。バス エレメントを認識するには、区切り文字 (かっこ (および各かっこ [など) を使用します。

[Select Net] ダイアログ ボックスの右上には、ILA コアのクロック信号、トリガー信号、およびデータ信号を示すタブがあります。ATC2 コアの [Select Net] ダイアログ ボックスを開いている場合は、データ信号タブのみが表示されます。トリガーまたはデータ ポートが複数ある場合は、[Net Selections] ペインの下部に複数のタブが表示されます。階層レベルで選択されたネットは、ILA または ATC2 キャプチャ コアの入力と接続できます。手順は次のとおりです。

1. [Select Net] ダイアログ ボックスの左下の表で、キャプチャ コアに接続するネットを選択します。

   注記：キャプチャ コア入力接続と同数のネットを選択できます。ネットを連続して選択する場合は Shift キーを押しながら、左クリックします。ネットを不連続に選択する場合は、Ctrl キーを押しながらクリックします。また、ネット 1 つと複数のキャプチャ コア ポート信号を選択して、このネットを複数のキャプチャ コア入力信号に接続することも可能です。

2. [Select Net] ダイアログボックスの右上ペインで、キャプチャコア入力カテゴリ ([Clock Signals]、[Trigger Signals]、または [Data Signals] (トリガーとデータが同一の場合は、[Trigger/Data Signals]) を選択します。

3. 右側のキャプチャ コア入力の表で、選択したネットと接続するチャネルを選択します。

   注記：ネット数と同数のキャプチャ コア入力を選択できます。ILA コア入力を連続して選択する場合は Shift キーを押しながら、左クリックします。ILA コア入力を不連続に選択する場合は、Ctrl キーを押しながらクリックします。また、ネット 1 つと複数のキャプチャ コア ポート信号を選択して、このネットを複数のキャプチャ コア入力信号に接続することも可能です。

4. [Select Net] ダイアログ ボックスの右下にある [Make Connections] をクリックすると、選択したネットとキャプチャ コアが接続されます。

既存の接続を削除する場合は、[Remove Connections] をクリックします。また、選択した接続の順序を変更する場合は、[Move Nets Up] または [Move Nets Down] を使用します。ネット接続終了後、[OK] をクリックして ChipScope Pro Core Inserter の画面に戻ります。

このような手順で、すべてのトリガーおよびデータ ネットを接続します。すべてのネットがバスに接続されると、ILA または ATC2 バス名の色が赤から黒に変わります。

クロック、トリガー、データ ネットを指定したら、[Insert] をクリックします。

ChipScope Pro Core Inserter をスタンドアロンで起動している場合、Core Insertion の実行を尋ねるダイアログ ボックスが表示されます。[はい] をクリックした場合、コアが生成されてネットリストに挿入され、EDIF2NGD ツールで NGO ファイルが生成されます。プロセスの詳細は、下部のメッセージ ウインドウに表示されます。ChipScope でコア挿入が正常に完了した場合は、「Core Generation Complete」と表示されます。

Project Navigator から ChipScope Pro Core Inserter を起動している場合は、Project Navigator へ戻ることを確認するダイアログ ボックスが表示されます。[はい] をクリックした場合、ChipScope Pro Core Inserter の設定が保存されて、Project Navigator の画面が表示されます。実際のコア生成およびコア挿入プロセスは、Project Navigator で適切な順序で実行されます。

ユニットの追加

各デバイスは、使用できるブロック RAM 数とパラメーターに従い、最大 15 個までの ILA または ATC2 ユニットをサポートできます。

- ILA ユニットをプロジェクトに追加する場合は、[Edit] → [New ILA Unit] をクリックします。または、ウインドウ左のツリーにある ICON をクリックし、[New ILA Unit] をクリックします。

- ATC2 ユニットをプロジェクトに追加する場合は、[Edit] → [New ATC2 Unit] をクリックします。または、ウインドウ左のツリーにある ICON をクリックし、[New ATC2 Unit] をクリックします。

追加したユニットのパラメーター設定方法は、前述した手順と同じです。

ネットリストへのコアの挿入

コアを挿入するには、[Insert] → [Insert Core] をクリックするか、またはツールバーの [Insert Core Into Design] をクリックします。

注記：ISE Project Navigator で ChipScope Pro Core Inserter フローを使用している場合は、[Insert] → [Insert Core] をクリックせずに [Return to Project Navigator] をクリックします。これにより、Project Navigator の変換プロセスで自動的にコアが挿入されます。詳細は、39 ページの「ISE Project Navigator での ChipScope Pro Core Inserter の使用」を参照してください。

プロジェクトのプリファレンス設定

プリファレンスは、[Tools]、[ISE Integration]、および [Miscellaneous] の 3 つのセクションに分かれています。

[Tools] では、ChipScope Pro Core Inserter が EDIF2NGD ツールを起動するときに使用するコマンドライン引数を設定します。

[ISE Integration] では、ChipScope Pro Core Inserter と ISE Project Navigator の統合方法を設定します。ISE 統合オプションがイネーブル (デフォルト) の場合、ChipScope Pro Core Inserter で ISE の一時的ネットリスト ディレクトリ (_ngo) の作業ディレクトリが自動的に検索されます。有効な _ngo ディレクトリが確認されると、ChipScope Pro Core Inserter のプロジェクトで ISE プロジェクトの中間 NGD ファイルが自動的に上書きされます。中間 NGD ファイルが上書きされる前にポップアップ ウィンドウを表示するには、[Prompt then Backup] を選択します。

[Miscellaneous] では、ChipScope Pro Core Inserter のその他の動作を設定します。たとえば、[Select Net] ダイアログ ボックスでポートを表示するように設定できます。これは、コアを最上位ではなく、下位の EDIF ネットリストに挿入する場合などに有効です。これらのポートのネットは、[Select Net] ダイアログ ボックスで灰色表示されます。また、違反している接続を [Select Net] ダイアログ ボックスに表示するオプションもあります。このオプションをオンにすると、[Select Net] ダイアログ ボックスには違反している接続が赤色表示されます。

また、ソース コンポーネントのインスタンス名、ソース コンポーネント タイプ、ベース ネット ドライバー タイプを [Select Net] ダイアログ ボックスに表示しないように設定できます。ChipScope Pro Core Inserter のプロジェクトのプリファレンス設定をデフォルトに戻す場合は、[Reset] をクリックします。

XILINX®

第 4 章

ChipScope Pro Analyzer の使用

ChipScope Pro Analyzer の概要

ChipScope™ Pro Analyzer ツールは、ChipScope Pro ロジック アナライザー コアと総称される ICON (Integrated CONtroller)、ILA (Integrated Logic Analyzer)、VIO (Virtual Input/Output)、および IBERT (Internal Bit Error Ratio) コアと直接インターフェースするツールです。

注記：ChipScope Pro Analyzer ツールでは、ATC2 (Agilent Trace Core) コアを認識できますが、実際に ACT2 コアと通信して制御するには、アジレント テクノロジー社のロジック アナライザーと JTAG (Joint Test Action Group、IEEE 規格) ケーブルを接続する必要があります。

このツールでは、デバイスのコンフィギュレーション、トリガーの選択、コンソールの設定、キャプチャ結果の表示が即時に実行できます。また、データ表示やトリガーをさまざまな方法で操作可能で、デザインの機能を容易かつ速やかに検証できます。

ChipScope Pro Analyzer ツールは、サーバーとクライアントの 2 つのアプリケーションで構成されています。サーバーは、コマンドライン アプリケーションで、30 ページの表 1-11 に含まれている JTAG ダウンロード ケーブルを使用してターゲット システムの JTAG チェーンに接続します。クライアントは GUI ベースのアプリケーションで、JTAG チェーンで接続されたデバイスや、それらのデバイス内のコアを操作できます。

ローカル ホスト モードでは、同じマシン上でサーバーとクライアントが実行でき、リモート モードであれば、別々のマシン上でも動作します。リモート モードは、次の場合に便利です。

- 別のロケーションにあるシステムのデバッグ
- ほかのチーム メンバーとの 1 つのシステム リソースの共有
- 別の場所にいる相手への機能や問題のデモ

ChipScope Pro Analyzer のサーバー インターフェイス

JTAG ダウンロード ケーブルを介して、直接ローカル マシンに接続したターゲット システムをデバッグする場合には、サーバーを手動で起動する必要はありません。リモート クライアントからサーバーに通信する場合にのみ、手動でサーバー アプリケーションを起動する必要があります。

注記：ChipScope Pro Analyzer サーバーのアプリケーションが一度に接続できるクライアントの数は 1 つです。

サーバーは、次のコマンドで起動できます。

- 32 ビット Windows マシンの場合：

  ```
  <XILINX_ISE_INSTALL>\bin\nt\cse_server.exe <command line options>
  ```

- 64 ビット Windows マシンの場合：

```
<XILINX_ISE_INSTALL>\bin\nt64\cse_server.exe <command line options>
```

- 32 ビット Linux マシンの場合 :

```
<XILINX_ISE_INSTALL>/bin/lin/cse_server <command line options>
```

- 64 ビット Linux マシンの場合 :

```
<XILINX_ISE_INSTALL>/bin/lin64/cse_server <command line options>
```

XILINX_ISE_INSTALL はザイリンクス ISE® Design Suite ツールのインストール ディレクトリを指します。サーバー アプリケーションには、表 4-1 に示すような複数のコマンド ライン オプションがあります。サーバーのスクリプトは、必要に応じてカスタマイズできます。

表 4-1 : ChipScope Pro Analyzer サーバーのコマンド ライン オプション

コマンド ライン オプション	説明
-port <*portnumber*>	クライアントからサーバーへの接続を確立する場合に、TCP/IP のポート番号を指定します。デフォルトのポート番号は 50001 です。
-password <*password*>	権限のないアクセスからサーバーを保護します。パスワードは、デフォルトでは設定されていません。
-l <*logfile*>	ログ ファイルの場所を保存指定します。デフォルトのディレクトリは次のとおりです。 `$HOME/.chipscope/cs_analyzer_<portnumber>.log` `$HOME` は、ユーザーのホーム ディレクトリ、*<portnumber>* はサーバーで使用する TCP/IP のポート番号です。

クライエント アプリケーションからサーバー アプリケーションに接続する方法の詳細は、64 ページの「サーバー ホストの接続設定」を参照してください。

ChipScope Pro Analyzer のクライアント インターフェイス

ChipScope Pro Analyzer のクライアントの GUI は、次の 4 つで構成されています。

- プロジェクト ツリー : ウィンドウの左側で、上下に分かれた上側のウィンドウ
- 信号ブラウザー : ウィンドウの左側で、上下に分かれた下側のウィンドウ
- メッセージ ペイン : ウィンドウの下部
- メイン ウィンドウ エリア

プロジェクト ツリーと信号ブラウザーとに分割されたペイン、およびメッセージ ペインは、[View] メニューで選択を解除すると非表示にできます。また、それぞれの大きさは、ウィンドウの端を任意の位置までドラッグして調整できます。各ウィンドウの境界にあるスクロール バーの矢印をクリックすると、ウィンドウのサイズを最小または最大に調整できます。

プロジェクト ツリー

プロジェクト ツリーには、JTAG チェーンと、そのチェーンに接続されたデバイスのコアがグラフィックで表示されます。チェーン内のデバイスすべてがツリーに表示されますが、有効なターゲット デバイスとコアのみに操作を実行できます。プロジェクト ツリーでさらに操作が実行可能である場合には、階層ノードが表示されます。たとえば、コアが挿入されているビット ストリームでデバイスをコンフィギュレーションした場合、各ユニットの階層ノードが表示されます。ツリーの各階層では、文脈依存のメニューを使用できます。メニューにアクセスするには、ツリー内のノード

で右クリックします。デバイスやユニットを別名に変更する、サブ ウィンドウを開く、デバイスをコンフィギュレーションする、またはプロジェクトを操作するなどの操作は、すべてこれらのメニューから実行できます。

プロジェクト ツリーのデバイスやコア ユニットのノードを別名で保存するには、ノード上で右クリックして、[Rename] をクリックします。編集を終了するには、Enter キーか上下の矢印キー、またはツリー内の他のノードをクリックします。

信号ブラウザー

信号ブラウザーには、プロジェクト ツリーで選択した ILA、IBA、または VIO コアの信号がすべて表示されます。信号は別名に変更したり、バスにグループ化したり、信号ブラウザーの文脈依存メニューを使用してさまざまなデータ ビューに追加できます。

信号名、バス名、トリガー ポート名の変更

信号ブラウザーで信号名、バス名、トリガー ポート名を変更するには、それぞれをダブルクリックするか、または右クリックして [Rename] をクリックします。編集を終了するには、Enter キーか上下の矢印キー、またはツリー内の他のノードをクリックします。

ビューでの信号の追加/削除

[Waveform] または [Listing] ビューからすべての信号を削除するには、信号ブラウザーのデータ信号またはバスを右クリックし、[Clear All] → [Waveform] または [Clear All] → [Listing] をクリックします。VIO コアでコンソールの信号をすべて削除する場合は、VIO コンソールの信号またはバス上で右クリックして [Clear All]→ [Console] をクリックします。また、[Add All to View] メニューを選択して、すべての信号とバスを [View] に追加することができます。選択した信号やバスは、[Add to View] メニューで追加します。

信号やバスを連続して複数選択する場合は、最初の信号をクリックしてから Shift キーを押したままグループ最後の信号をクリックします。信号やバスを非連続に複数選択するには、Ctrl キーを押したまま信号やバスを順にクリックします。この方法を使用すると、バス内の信号の順序は選択した順になります。

バスに信号を組み合わせる、または追加するには

ILA コアで組み合わせてバスに含めることが可能な信号は、データ信号のみです。VIO コアの場合、特定のタイプの信号をグループ化してバスを構成できます。信号を組み合わせてバスに含めるには、前述の方法と同様に、Shift キーまたは Ctrl キーを使用して信号を選択します。Shift キーを使用する場合、ツリー内の一番上の信号が生成したバスの LSB (最下位ビット) になります。Ctrl キーを使用する場合、バス内の信号順序はクリックした順序になり、最初にクリックした信号が LSB になります。

選択後は、信号上で右クリックし、[Move to Bus] → [New Bus] をクリックします。ILA コアの場合は [Data Signals and Buses] の一番上に新しいバスが生成され、VIO コアの場合は特定のサブツリーの一番上に生成されます。既存のバスに 1 つまたは複数の信号を追加するには、まず信号を右クリックし、[Move to Bus] のサブメニューでバス名を選択します。信号は常にバスの MSB (最上位ビット) 側に追加されます。このコマンド を実行すると、信号はバスに含められた後に、信号リストから削除されます。

バス内の配列順序の反転

バス内のビット順序を反転する (LSB を MSB にする) には、バス上で右クリックして [Reverse Bus Order] をクリックします。信号ブラウザーおよびバスを含むすべてのデータ ビューには変更がすぐに反映され、バスの値が再計算されます。

バスの基数

各バスは、次の基数のいずれかでデータ ビューに表示されます。

- ASCII
- 2 進数
- 16 進数
- 8 進数
- 符号付き 10 進数
- トークン
- 符号なし 10 進数

ACII は、バスのビット数が 8 で割り切れる場合に使用できます。基数を変更すると、各データ ビューに含まれるバスの基数も変わります。

符号付き 10 進数と符号なし 10 進数

バスの値は、次の等式を使用して符号付き 10 進数と符号なし 10 進数のいずれかに置き換えることができます。

バスの値 = (<scale factor> * Data) + <offset>

精度= <precision>

いずれの バス基数を選択しても、スケール係数 <scale factor>、オフセット <offset>、および 精度 <precision> を入力するダイアログ ボックスが表示されます。

- スケール係数

  「* Data」の前のテキスト ボックスには、データ値を乗算するのに使用する定数のスケール係数を入力します。デフォルトのスケール係数は 10 です。

- オフセット

  2 番目のテキスト ボックスには、スケール係数処理されたデータ値に追加される定数のオフセット値を入力します。

- 精度

  3 番目のテキスト ボックスには、精度を小数点以下の桁数で入力します。デフォルトの精度は 0 です。

たとえば、小数点以下 10 桁の精度で、-0.5 ～ 1.5 の範囲のサイン波を表した 16 ビット バスを表示する場合、次のようにパラメーターを設定します。

スケール係数 = 3.0517578125E-5 (1/215 と同じ)

オフセット = 0.5

精度 = 10

トークン

トークンは、別の ASCII ファイルで定義された文字列のラベルで、特定のバス値に割り当てることができます。このラベルは、アドレスのデコードやステート マシンなどのアプリケーションで便利です。トークン ファイル (拡張子は .tok) は単純なフォーマットなので、どのテキスト エディターでも作成や編集ができます。トークンは NAME=VALUE という形式になっており、NAME はトークン名、VALUE はトークン値 (16 進数、2 進数、または 10 進数) を指します。値はデフォルトで 16 進数になっています。値の基数を指定する場合は、値に \b (2 進数)、\u (符号なし 10 進数)、\h (16 進数) を付けてください。

デフォルト トークンは、VALUE の一致が見つからないときにデフォルトのトークン値を設定するのに使用できます。@DEFAULT_TOKEN キーを使用すると、デフォルトのトークン名を設定できます。@DEFAULT_TOKEN 行が使用されている場合、トークン名 HEX が使用されます。トークン ファイルのコメントは # で始まります。トークン ファイルの最初のコメント以外の行は、@FILE_VERSION=1.0.0 である必要があります。

注記：= は予約文字で、TOKEN 文字列に使用できません。

次に、トークン ファイルの例を示します。

```
#File version
@FILE_VERSION=1.0.0

# Default token value
@DEFAULT_TOKEN=ERROR

# Explicit token values
ZERO=00
ONE=01
TWO=02
THREE=11\b
FOUR=4\h
FIVE=101\b
SIX=6
SEVEN=111\b
EIGHT=1000\b
NINE=9\h
TEN=A\h
```

トークンは、バスを右クリックして [Bus Radix] → [Token] をクリックすると選択できます。トークン ファイルを選択するダイアログ ボックスが表示されます。バスが 8 ビット幅で指定したトークン値が 4 ビット幅しかないときなど、バス幅がトークン値より大きい場合、バスと同じ幅にするようトークンの最上位ビット側に 0 が挿入されます。

バスの削除

バスを削除するには、そのバスを右クリックし、[Delete Bus] をクリックします。バスは、関連するすべてのデータ ビューから削除されます。

[Type] と [Activity Persistence] (VIO のみ)

VIO 信号には 2 つの付加プロパティ、[Type] と [Activity Persistence] があります。これらのプロパティの詳細は、80 ページの「VIO コアのバスおよび信号のアクティビティ持続時間」を参照してください。

[Message] ペイン

[Messages] ペインには、ステータス メッセージのリストが表示されます。エラー メッセージは赤で表示されます。ウィンドウの境界をドラッグして任意の位置まで移動させると、[Message] ペインのサイズを変更できます。プロジェクト ツリー/信号ブラウザーのペインの境界も変更できます。

メイン ウィンドウ

メイン ウィンドウでは、同時に複数のサブ ウィンドウ ([Trigger]、[Waveform]、[Listing]、[Plot]) が表示されます。各ウィンドウ サイズは、必要に応じて変更できます。

ChipScope Pro Analyzer の機能

プロジェクトでの作業

プロジェクトには、信号名、信号の順序、バス構成、トリガー条件など、ChipScope Pro Analyzer プログラムの内容に関する重要な情報が含まれています。これらの情報は、ChipScope Pro Analyzer の使用中に自由に読み出したり保存できます。

ChipScope Pro Analyzer ツールを最初に起動したとき、「new project」という名前の新規プロジェクトが自動で生成されます。既存のプロジェクトを開くには、[File]→ [Open Project] をクリックするか、[File] メニューから最近使用したプロジェクトをクリックします。ChipScope Pro Analyzer のタイトル バーとプロジェクト ツリーには、プロジェクト名が表示されます。新規プロジェクトを保存せずに終了しようとすると、プロジェクトを保存するかどうかを尋ねるダイアログ ボックスが表示されます。

新規プロジェクトの作成と保存

新規プロジェクトを作成するには、[File] → [New Project] をクリックします。「new project」という新規プロジェクトが作成され、プロジェクトがアクティブになります。プロジェクトを別名で保存するには、[File] → [Save Project] をクリックします。プロジェクト ファイルの拡張子は .cpj です。

プロジェクトの保存

作業中のプロジェクト名を変更したり、またはファイルを別名で保存するには、[File]→ [Save Project As] をクリックし、新しいファイル名を入力後、[保存] をクリックします。

波形を印刷する

ChipScope Pro の特長の 1 つに、[File]→ [Print] を実行すると計測したデータの波形が印刷できるという点があります。[File] → [Print] をクリックして、Print Wizard を起動します。

Print Wizard は、次の 3 つの連続ウィンドウで構成されています。

1. (1 of 3) は、印刷オプションと設定のウィンドウです。
2. (2 of 3) は、印刷波形のプレビューのナビゲーター ウィンドウです。
3. (3 of 3) は、印刷を確認するウィンドウです。

Print Wizard (1 of 3) ウィンドウ

[Print Wizard] ウィンドウでは、さまざまな波形印刷のオプション設定ができます。これらの波形印刷オプションの詳細を説明します。

[Horizontal Scaling]

ページのカラムに印刷されるの波形データの量は、次の 2 つの方法のいずれかで調節できます。

- [Fit To] : 波形を指定のページ内に収めます。
- [Fixed] : 指定した数の波形サンプルを各ページに収めます。

デフォルトでは、波形全体が 1 ページに印刷されます。

[Signal/Bus Selection]

印刷する波形の信号やバスは、次の 3 つの方法のいずれかで調節できます。

- [Current View] : 作業中の [Waveform] ウィンドウで表示されているすべての信号とバスの波形データを印刷します。
- [All] : コア ユニット全体で使用可能なすべての信号とバスの波形データを印刷します。
- [Selected] : [Waveform] ウィンドウで選択した信号とバスのみの波形データを印刷します。

デフォルトでは、[Current View] で印刷されます。

[Time/Sample Range]

印刷時の単位やサンプル数は、次の 4 つの方法のいずれかを用いて設定できます。

- [Current View] : 現在表示されている波形と同じ範囲やサンプル設定で波形を印刷します。
- [Full Range] : サンプル バッファー全体に含まれるすべてのサンプルを範囲とする波形データを印刷します。
- [Between X/O Cursors] : X カーソルから O カーソルまでをサンプリング範囲とした波形データを印刷します。
- [Custom View] : 次で定義されるサンプル範囲を使用して波形データを印刷します。
  - 開始ウィンドウ番号
  - 開始ウィンドウ内のサンプル番号
  - 終了ウィンドウ番号
  - 終了ウィンドウ内のサンプル番号

デフォルトでは、[Current View] で印刷されます。

[Print Signal Names]

信号名を、各ページに印刷するか、または最初のページにのみ印刷するかを選択します。[Show Cursor Values] がオンになっている場合、この値が X/O カーソル値の表示にも影響します。

[X/O Cursor Values]

X/O カーソル値を印刷に含めるかどうかも選択できます。X/O カーソル値を表示して印刷する場合、[Print Signal Names] の設定によって、全ページに表示されるか、最初のページだけに表示されるかが決まります。

[Footer]

[Show Footer] をオンにすると、各ページの下部にフッターを含めることができます。フッターには、ChipScope Pro Analyzer のプロジェクト名、波形設定、印刷設定、およびページ番号など、役立つ情報が表示されます。

ナビゲーション ボタン

[Print Wizard (1 of 3)] ウィンドウの下部のボタンで、次の操作を実行できます。

- [Page Setup] : [ページ設定] ウィンドウを開きます。
- [Next] : [Print Wizard (2 of 3)] ウィンドウを開きます。
- [Cancel] : 印刷をキャンセルして [Print Wizard] ウィンドウを閉じます。

[Next] をクリックすると、[Print Wizard (2 of 3)] ウィンドウが表示されます。

Print Wizard (2 of 3) ウィンドウ

[Print Wizard (2 of 3)] ウィンドウでは、印刷する波形のプレビューが表示されます。

ページ プレビュー ボタン

次に示すように、ページ上部のボタンで印刷する波形をプレビューします。

- [<<] および [>>] で、プレビュー ページの先頭ページ、最終ページにそれぞれ移動します。
- [<] および [>] で、前のページ、次のページにそれぞれ移動します。
- 中央のテキスト ボックスでは、指定したページのプレビューに移動します。

ナビゲーション ボタン

[Print Wizard (2 of 3)] ウィンドウの下部のボタンで、次の操作を実行できます。

- [Back] : [Print Wizard (1 of 3)] ウィンドウに戻ります。
- [Send to PDF] : [Print Wizard (3 of 3)] ウィンドウを開き、PDF ファイルとして印刷します。
- [Send to Printer] : [Print Wizard (3 of 3) ウィンドウ] を開き、プリンターに送信します。
- [Close] : 印刷をキャンセルして [Print Wizard] ウィンドウを閉じます。

バスの展開表示と非表示

印刷プレビューで、バスを展開表示したり非表示したりして、波形の操作ができます。たとえば、バスを展開表示すると、そのページの信号やバスが次のページに移動して、ページ上部の印刷プレビューの合計ページ数が変わります。

Print Wizard (3 of 3) ウィンドウ

[Print Wizard (2 of 3)] ウィンドウで、[Send to PDF] をクリックすると、[Print Wizard (3 of 3): PDF] の確認ウィンドウが表示されます。[Yes] をクリックすると、波形が PDF ファイルに印刷され、[No] をクリックすると、[Print Wizard (2 of 3)] ウィンドウに戻ります。[Change File] をクリックすると、ファイルのブラウザー ウィンドウが起動し、PDF ファイルを新規作成したり選択できます。

[Print Wizard (2 of 3)] ウィンドウで、[Send to Printer] をクリックすると、[Print Wizard (3 of 3): Printer] の確認ウィンドウに移動します。[はい] をクリックすると波形データがプリンターに送信され、[いいえ] をクリックすると、[Print Wizard (2 of 3)] ウィンドウに戻ります。

ページ設定

[ページ設定] ウィンドウは、[Print Wizard (1 of 3)] ウィンドウまたは [File] → [Page Setup] メニューから開くことができます。

注記：ChipScope Pro Analyzer では、デフォルトのシステム プリンターでのみ印刷できます。印刷設定ウィンドウでプリンターを変更しても反映されません。プリンターを変更するには、ChipScope

Pro Analyzer プログラムを終了し、デフォルトのシステム プリンターを変更してから ChipScope Pro Analyzer を再起動します。

信号名のインポート

プロジェクトの開始時は、各コアの信号名はすべて一般名称です。信号名は 57 ページの「信号名、バス名、トリガー ポート名の変更」に示すようにそれぞれ変更するか、または 1 つまたは複数のコアに含まれる名称すべてが含まれているファイルをインポートします。このようなファイルは、CORE Generator™、ChipScope Pro Core Inserter、EDK Platform Studio、System Generator for DSP、PlanAhead™ ツール、および FPGA Editor ツールで作成できます。ファイルから信号名をインポートするには、[File] → [Import] をクリックし、[Signal Import] ダイアログ ボックスを表示します。

[Select New File] をクリックして、信号名のインポート ファイルを選択します。[Open Signal File] ダイアログ ボックスでインポート ファイルを選択します。ファイルを選択すると、ファイルに含まれるコアのタイプに従って [Unit/Device] の値が更新されます。インポート ファイルに複数のコアの信号名が含まれる場合、ChipScope Pro キャプチャ コアのみを含むデバイスすべてのデバイス番号がコンボ ボックスに表示されます。

インポート ファイルに 1 つのコアのみの信号名が含まれる場合、信号名のインポート ファイルで指定したタイプに合致するコア名がコンボ ボックスに表示されます。

信号名をインポートするには、[OK] をクリックします。ファイルのパラメーターがターゲット コアのパラメーターと異なる場合は、警告メッセージが表示されます。続行する場合は、信号名はコアに規定通りに適用されます。

データのエクスポート

ILA コアを使用してキャプチャしたデータをファイルにエクスポートし、後から表示や編集ができます。データをエクスポートするには、[File] → [Export] をクリックし、[Export Signals] ダイアログ ボックスを表示します。

エキスポートには、3 つのフォーマット、[VCD] (value change dump) フォーマット、タブ区切りの [ASCII] フォーマット、アジレント テクノロジー社の [FBDF] (Fast Binary Data Format) があります。使用するフォーマットのラジオ ボタンをオンにします。[Core] ボックスからエクスポートするコアを選択します。

別の信号やバスのセットもエクスポートできます。[Signals to Export] の選択肢は次の通りです。

- [All Signals/Buses] : 特定コアの信号およびバス
- [Waveform Signals/Buses] : [Waveform] ビューに表示されている信号とバス
- [Listing Signals/Buses] : [Listing] ビューに表示されている信号とバス
- [Bus Plot Buses] : コアの [Bus Plot] ビューに含まれる信号とバス

信号をエクスポートするには、[Export] をクリックし、表示されるダイアログ ボックスでターゲット ディレクトリとファイル名を指定します。

ChipScope Pro Analyzer を閉じる、または終了する

[File] → [Exit] をクリックして ChipScope Pro Analyzer を終了します。終了時には、作業中のプロジェクトが自動的に保存されます。

オプションの確認

ChipScope Pro Analyzer ウィンドウ左側ペインおよび下部の [Message] ペインは、表示/非表示を切り替えることができます。デフォルトでは、いずれのウィンドウも初回起動時に表示されます。プロジェクト ツリー /信号ブラウザー ペインを非表示にするには、[View] → [Project Tree] をクリックします。[Message] ペインを非表示にするには、[View] → [Messages] をクリックします。

サーバー ホストの接続設定

ChipScope Pro Analyzer クライアントの GUI アプリケーションを実行するには、ローカルまたはリモート システムで動作中の ChipScope Pro Analyzer サーバー アプリケーションに接続する必要があります。[JTAG Chain] → [Server Host Setting] をクリックし、サーバーを設定するダイアログ ボックスを表示します。

ローカル モードで作業する場合、[Server] は常に「**localhost:<port>**」に設定する必要があります。<port> には、未使用の TCP/IP ポート番号を入力できます。デフォルトの **<port>** 番号は 50001 です。ローカル モードでは、パスワードの設定は不要です。

注記：ローカル モードでは、サーバーが自動的に起動します。

リモート モードで作業する場合、[Server] に IP アドレス (または適切なシステム名) およびポート番号を「192.168.0.1:50001」(または「servername:50001」) の形式で設定する必要があります。[Password] には、リモート システムでサーバー起動時に使用したパスワードを入力する必要があります。リモート モードでは、JTAG ダウンロード ケーブルで接続を開始するまで実際の接続が確立されません。

注記：55 ページの「ChipScope Pro Analyzer のサーバー インターフェイス」で説明したとおり、リモート モードではサーバーを手動で起動する必要があります。

パラレル ケーブルで接続する

パラレル ケーブルで接続するには、ケーブルが PC のパラレル ポートに接続されているかどうかを確認します。[JTAG Chain] → [Xilinx Parallel Cable] をクリックし、パラレル ケーブルを選択するダイアログ ボックスを表示します。[Xilinx Parallel III] または [Xilinx Parallel IV] 選択するか、[Auto Detect Cable Type] を選択してケーブルを自動検出します。

[Xilinx Parallel IV] または [Auto Detect Cable Type] を選択した場合、ケーブルの速度に 10MHz、5MHz (デフォルト)、2.5MHz、1.25MHz、625kHz のいずれかを選択できます。その場合は、被試験ボードで最も適切な速度を選択します。次に [Port] でプリンター ポート名を選択し (通常はデフォルトの LPT1 が適切) 、[了解] をクリックします。その後、ChipScope Pro Analyzer で バウンダリ スキャン チェーンの構成が確認されます (67 ページの「バウンダリ スキャン (JTAG) チェーンの設定」を参照)。

「Failed to Open Communication Port」というエラー メッセージが表示された場合は、ケーブルが LPT ポートに接続されているかどうかを確認します。パラレル ケーブル ドライバーをインストールしていない場合は、ChipScope Pro ツールのインストール プログラム手順に従って、デバイス ドライバー ソフトウェアをインストールしてください。

プラットフォーム ケーブル USB で接続する

プラットフォーム USB で接続するには、ケーブルがコンピューターの USB ポートに接続されているかどうかを確認します。[JTAG Chain]→ [XilinxPlatform USB Cable] をクリックし、プラットフォーム ケーブル USB を設定するダイアログ ボックスを表示します。

USB ポートの選択

ChipScope Pro Analyzer ツールでは、接続しているサーバー マシンに接続されているザイリンクス プラットフォーム ケーブル USB JTAG ケーブルすべてが [Port] ボックスに自動的に表示されます。[Port] には、選択可能なUSB ポートが USB2<n> という形式で表示されます。<n> は 1 ～ 127 の整数値で、デフォルトのポートは USB21 に設定されています。USB ポート番号は、プラットフォーム ケーブル USB ダウンロード ケーブルがシステムの USB ポートに接続される順番に基づいています。たとえば、システムに接続されている最初のプラットフォーム ケーブル USB ダウンロード ケーブルが USB21 に割り当てられる場合、次のケーブルは USB22 に割り当てられます。新しいプラットフォーム ケーブル USB ダウンロード ケーブルでは、ケーブルから読み出される電子シリアル番号 (ESN) が表示されます。

注記：列挙は、システムの電源が投入されるときは保持されない場合があります。また、特定の順番にケーブルをシステムに接続しない限り、プラットフォーム ケーブル USB を識別できません。

ケーブルの LED の点滅

選択したケーブルの LED を点滅させるには、ダイアログ ボックスに含まれている [Blink LED] をクリックします。

ケーブル速度の選択

ケーブルの速度には、12MHz、6MHz、3MHz (デフォルト)、1.5MHz、または 750kHz のいずれかを使用できます。被試験ボードで最も適切な速度を選択してください。

Digilent 社製 USB JTAG ケーブルに接続する

TAG-SMT1、JTAG-HS1 ケーブルなどの Digilent 社製 USB JTAG ケーブルに接続する場合、コンピューターの USB ポートのいずれかにケーブルを接続する必要があります。次に、[JTAG Chain] → [Digilent USB JTAG Cable] をクリックします。

ケーブル デバイスの選択

ChipScope Pro Analyzer ツールでは、接続しているサーバー マシンに接続されている Digilent USB JTAG ケーブルすべてが [Device] ボックスに自動的に表示されます。任意の Digilent USB JTAG ケーブルをリストから選択できます。

ケーブル速度の選択

ケーブル デバイスを選択したら、その選択したケーブルでサポートされる TCK 速度が [Speed] ボックスに表示されます。任意の速度をリストから選択できます。

注記：通常、ケーブルの TCK 最大速度は被試験ボードが実行可能な速度に制限されます。場合によってはケーブル自体の最大速度をはるかに下回る可能性があります。

プラットフォーム ケーブル USB 接続の複数使用

ChipScope Pro Analyzer で複数のケーブルを使用するには、次の 3 つが必要です。

- 1 つのマシンに接続されている複数のザイリンクス JTAG ケーブル
- 1 つのマシンで実行されている cs_server アプリケーションの複数インスタンス (それぞれが異なるポートと通信)
- 同じマシンまたはリモート サーバー機能を使用して別のマシンで実行されている ChipScope Pro Analyzer の複数インスタンス

1 つのマシンに複数のザイリンクス JTAG ケーブルを接続する

同じマシンに接続されている複数の JTAG ケーブルと通信するには、複数のプラットフォーム ケーブル USB、パラレル ケーブル III、またはパラレル ケーブル IV をマシンに接続できることを確認する必要があります。プラットフォーム ケーブル USB では、必要なケーブル数に応じて 1 つまたは複数の USB ハブを使用する必要がある場合があります。パラレル ケーブル III、またはパラレル ケーブル IV では、1 つまたは複数のパラレル ポート拡張カードが必要になる場合があります。

注記：現時点では、特定の物理的プラットフォーム ケーブル USB と列挙が関連付けられていません。このため、マシンを再起動すると、列挙と物理ケーブルの関連付けが異なる結果になる場合があります。すべてのケーブルの接続を解除し、任意の列挙順に接続し直すと、この問題を回避できます。

cs_server の複数インスタンスを使用できるように ChipScope Pro Analyzer を設定する

複数のケーブルを使用できるように設定するには、まず同じマシン上で cs_server.exe (Windows アプリケーション) または cs_server.sh (Linux アプリケーション) の複数インスタンスを別々のポートを使用して起動します。次は、Linux で異なる 2 つのポートを使用して 2 つのサーバーを起動する例です。

```
# cs_server.sh -port 50001
# cs_server.sh -port 50002
```

ChipScope Pro Analyzerの複数インスタンスを開始/設定する

複数の ChipScope Pro Analyzerクライアント インスタンスを起動、設定します (表 4-2)。ChipScope Pro Analyzer のインスタンスはそれぞれ異なる cs_server と USB ポートに接続します。

表 4-2 : 複数のクライアント インスタンスの設定

ChipScope Pro Analyzer インスタンス番号	サーバー ホスト設定	プラットフォーム ケーブル USB ポート番号
1	<IP Address>:50001	USB21
2	<IP Address>:50002	USB22

JTAG チェーン プラグインに接続する

ChipScope Pro Analyzer では、[JTAG Chain] → [Open Plugin] コマンドを使用して JTAG チェーンのプラグイン接続を実行できます。各 JTAG チェーンのプラグインには特定のパラメーターがあり、これらの値は [Open Plug-in] ダイアログ ボックスの [Plug-in Parameters] に表示されます。プラグイン パラメーターを選択した後に [了解] をクリックして、JTAG プラグインへの接続を開きます。

自動コアのステータスのポーリング

コアを搭載すると、インターフェイス ケーブルより定期的にコアでのキャプチャのステータスが確認されます。ChipScope Pro Analyzer とほかのプログラムで同時にケーブルを使用する場合は、ポーリングをオフにした方がいいことがよくあります。オフにするには、[JTAG Chain] → [Auto Core Status Poll] でチェック ボックスをオフにします。オフの場合、[Run] または [Trigger Immediate] を実行したときに、ChipScope Pro Analyzer でコアのステータスは自動的に確認されません。

オフにすることでケーブルとの通信が完全に不可能になるのではなく、コアを搭載した場合に限り、定期的なポーリングが実行できなくなります。ポーリングのディスエーブル後に1 つまたは複数のコアをトリガーすると、[Auto Core Status Poll] オプションをオンにしない限り、キャプチャ バッファーがデバイスからダウンロードができず、データ ビューに表示されません。

ターゲット デバイスのコンフィギュレーション

ChipScope Pro Analyzer ツールは、複数のターゲット デバイスに使用できます。まず、バウンダリ スキャン チェーンですべてのデバイスを設定します。

バウンダリ スキャン (JTAG) チェーンの設定

ChipScope Pro Analyzer でダウンロード ケーブルとの通信を確立すると、バウンダリ スキャン (JTAG) チェーンの構成が自動的に確認されます。すべてのザイリンクス FPGA、CPLD、PROM、および System ACE™ デバイスが自動的に検出されます。有効なターゲット デバイスに対し IDCODE 全体を検証できます。チェーンの構成を確認するには、[JTAG Chain] → [JTAG Chain Setup] をクリックします。ダイアログ ボックスに検出されたすべてのデバイスが順に表示されます。

自動的に検出されないデバイスは、コアと確実に通信できるよう IR (命令レジスタ) の長さを指定する必要があります。この情報は、デバイスの BSDL ファイルに記述されています。USERCODE は、[Read USERCODEs] をクリックしてターゲット FPGA デバイスから読み出すことができます。

ChipScope Pro Analyzer ツールでは、デフォルトで JTAG チェーンに接続されたデバイスのテスト アクセス ポート (TAP) のステータスが自動的に記録されます。ChipScope Pro Analyzer を System ACE CompactFlash (CF) コントローラーやプロセッサのデバッグ ツールなど、ほかの JTAG コントローラーと併用する場合には、ターゲット デバイスの実際の TAP ステートは、ChipScope Pro Analyzer の記録コピーとは異なります。この場合 JTAG トランザクション シーケンスの実行前に、ChipScope Pro Analyzer で TAP コントローラーを [Run-Test] や [Idle] などの既知ステートにしておく必要があります。[JTAG Chain Device Order] ダイアログ ボックスで [Advanced] をクリックすると、ダイアログ ボックスに JTAG トランザクションの開始と終了を制御するパラメーターが表示されます。JTAG チェーンをほかの JTAG コントローラーと併用する場合は、[Start transactions in Test-Logic/Reset, End in Run-Test/Idle] を選択します。

デバイスのコンフィギュレーション

ChipScope Pro Analyzer では、JTAG モードに限り、次のダウンロード ケーブルを使用してターゲット FPGA デバイスのコンフィギュレーションができます。

JTAG ポート経由でダウンロード ケーブルを使用してターゲット デバイスをプログラムする場合には、[Device] メニューからコンフィギュレーションするデバイス名を選択し、[Configure] をクリックします。有効なターゲット デバイスのみコンフィギュレーション可能となり、[Configure] オプションが使用できます。プロジェクト ツリーのデバイス上で右クリックしても、[Device] と同じメニューを表示できます。

[Configure] をクリックすると、JTAG コンフィギュレーションを設定するダイアログ ボックスが表示されます。このダイアログ ボックスには、JTAG デバイス コンフィギュレーション ファイルを選択するセクションと、デザイン レベルの CDC ファイルを選択するセクションがあります。デバイスにダウンロードするコンフィギュレーション ファイルを選択するには、[JTAG Configuration] セクションで [Select New File] をクリックします。[Open Configuration File] ダイアログ ボックスが開くので、ブラウザーからターゲット デバイスのコンフィギュレーションに使用するデバイス コンフィギュレーション ファイルを選択します。適切な BIT ファイルを選択したら、[開く] をクリックして元のダイアログ ボックスに戻ります。また、[Clean previous project

setting] をオンにすると、デバイスをコンフィギュレーションする前に既存のプロジェクト設定を消去することも可能です。

注記：オンにした場合、プロジェクトに含まれるすべての信号名、バス、およびその他の設定が削除されます。この操作は、元に戻すことができません。

注記：適切な BitGen 設定を使用して生成された BIT ファイルを選択することが重要です。特に、BitGen で -g StartupClk:JtagClk オプションが使用されることを確認してください。

ChipScope Pro Core Inserter または PlanAhead ツールで生成されたデザイン レベルの CDC ファイルを選択するには、[Import Design-level CDC File] をオンにしてから [Select New File] をクリックします。[Open CDC File] ダイアログ ボックスが開くので、ブラウザーからターゲット デバイスで使用する CDC ファイルを選択します。適切な CDC ファイルを選択したら、[開く] をクリックして元のダイアログ ボックスに戻ります。[Auto-create buses] をオンにすると、CDC ファイルに含まれる信号名から自動的にバスが作成されます。ChipScope Pro Analyzer では、バス エレメントの識別にアルゴリズムが使用されます。エレメント番号は信号名の最後に表示され、( )、[ ]、または { } で囲まれるか、またはアンダースコアを挟んで表示されます。

注記：デザイン レベルの CDC ファイルがコンフィギュレーション BIT ファイルと同じディレクトリにある場合、CDC ファイルが自動的に選択されます。

BIT とオプションの CDC ファイルを選択したら [OK] をクリックしてデバイスをコンフィギュレーションします。

コンフィギュレーション進捗状況の確認

デバイスのコンフィギュレーション中、ChipScope Pro Analyzer のウィンドウ下部にステータスが表示されます。DONE ステータスが表示されない場合は、コンフィギュレーション中に発生した問題を示すダイアログ ボックスが表示されます。ダウンロードが正しく完了した場合、ターゲット デバイスで ChipScope Pro コアが自動的に検索され、存在するコアの数だけプロジェクト ツリーが更新されます。各コア ユニットにフォルダーが 1 つ作成され、そのユニットには下位ノードが表示されます。

JTAG ユーザーと ID コードの表示

ターゲット デバイスが正しくコンフィギュレーションされたことを確認する方法の 1 つは、ターゲット デバイスからユーザー定義の ID コードとデバイス ID コードをアップロードすることです。ユーザー定義の ID コードは 8桁の 16 進数コードで、BitGen オプション -g UserID を使用して設定できます。

特定デバイスのユーザー定義 ID コードをアップロードして表示するには、特定のデバイスを右クリックして [Device]→ [Show USERCODE] をクリックします。特定のデバイスの固定デバイス ID を表示する場合は、特定のデバイスを右クリックして [Device] → [Show IDCODE] をクリックします。これらのクエリ結果は、メッセージ ペインに表示されます。IDCODE および USERCODE は、[JTAG Chain] → [JTAG Chain Setup] で表示される [JTAG Chain Device Order] ダイアログ ボックスにも表示されます。

コンフィギュレーション ステータスの表示

32 ビットのコンフィギュレーション ステータス レジスタには、コンフィギュレーション ピンとその他の内部信号のステータスの情報などが含まれています。コンフィギュレーションで問題が発生した場合は、ターゲット デバイスを右クリックして [Device] → [Show Configuration Status] をクリックしてメッセージ ペインを確認します。

注記：すべてのターゲット デバイスには、ステータス情報を示す内部レジスタとして、1) コンフィギュレーション ステータス レジスタ (32 ビット) と 2) JTAG 命令レジスタ (長さはデバイスにより

異なる) の 2 つがあります。有効なターゲット デバイスにのみにコンフィギュレーション ステータス レジスタがあります。すべてのデバイスに読み出し可能な JTAG 命令レジスタがありますが、ステータスが表示されるかどうかは、特定デバイスのインプリメンテーションで決定されます。各コンフィギュレーション ステータス ビットの定義は、特定の FPGA デバイスのコンフィギュレーションに関する資料を参照してください。

デバイスによっては、JTAG 命令レジスタにステータス情報も保持されます。JTAG チェーンに接続されているどのデバイスでも [Device] → [Show JTAG Instruction Register] をクリックして、その情報をメッセージ ペインに表示できます。

[Trigger Setup] ウィンドウ

ILA コアでトリガーを設定するには、[Window] → [New Unit Windows] からコアを選択します。コアのダイアログ ボックスが表示され、[Trigger Setup]、[Waveform]、[Listing]、[Bus Plot] およびコンソール ウィンドウを任意の組み合わせで表示できます。このダイアログ ボックスからはウィンドウを閉じることはできません。

トリガーの設定は、プロジェクト ツリーの [Trigger Setup] をダブルクリックするか、または右クリックして [Open Trigger Setup] をクリックしても同様に実行できます。

各 ILA コアにはそれぞれトリガーを設定できる [Trigger Setup] ウィンドウがあります。各コア内部のトリガーは、デザインをコンパイルし直さずに実行時に編集できる機能を備えています。次にトリガー機能の 3 つのコンポーネントの変更方法を説明します。

- [Match Functions] : 各比較ユニットとの一致または比較を定義します。
- [Trigger Conditions] : バイナリ式または 1 つあるいは複数の比較演算子のシーケンスに基づいて、全体のトリガー条件を定義します。
- [Capture Settings] : キャプチャするサンプル数、ウィンドウ数、ウィンドウ内のトリガーの位置を定義します。

[Trigger Setup] ウィンドウでは、各コンポーネントの表示を展開/非展開できます。展開するには、ウィンドウ下部のボタンをクリックします。非展開するには、展開しているコンポーネント左側のボタンをクリックします。

[Capture Settings]

[Trigger Setup] ウィンドウの [Capture Settings] で、トリガー イベントを発生させるウィンドウ数と、ウィンドウ中のイベントの位置を定義します。ウィンドウとは、1 度のトリガー イベントを含む連続したサンプル シーケンスを指します。パラメーターに無効な数が入力されると、テキスト フィールドが赤になります。

[Type]

使用するウィンドウのタイプを定義します。[Window] を選択した場合は、各ウィンドウのサンプル数は 2 のべき乗にする必要があります。ただし、トリガーはウィンドウのどの位置にでも設定できます。[N Samples] を選択した場合には、バッファーにはトリガーごとに定義したサンプル数に対して可能な限りのウィンドウ数が含められます。このオプションを選択した場合、トリガーは常にウィンドウの最初のサンプルとなります。

[Windows]

[Type] で [Window] を選択した場合のみ、このテキスト フィールドが使用できます。このフィールドで定義できる値は、正の整数 1 ～ キャプチャ バッファーのワード数です。

[Depth]

[Type] で [Window] を選択した場合のみ、このテキスト フィールドが使用できます。このボックスでは、各キャプチャ ウィンドウのワード数を定義します。[Windows] に入力した値に基づいて、このボックスに選択肢が自動的に含められます。選択可能な値は、2 のべき乗数のみです。

注記：トリガー条件全体で、[Occurring in at least ***n*** cycles] または [Lasting for at least ***n*** consecutive cycles] が設定されているカウンターを含む比較ユニットが 1 つでも含まれる場合、ウィンドウのワード数またはトリガーごとのサンプル数を 8 以上にする必要があります。これは、ILA コアに含まれるトリガー ロジックがパイプライン化されているためです。

[Position]

[Type] で [Window] を選択した場合のみ使用できます。このボックスでは、各ウィンドウのトリガー発生地点を定義します。0 ～ キャプチャ バッファーのワード数 - 1 の値までの整数を入力できます。

[Samples Per Trigger]

[Type] で [N Samples] を選択した場合のみ使用できます。このボックスでは、トリガー条件が発生した後にキャプチャするサンプル数を定義します。1 ～ キャプチャ バッファーのワード数 － 1 の正の整数を入力できます。トリガー マークは、ウィンドウに sample 0 として常に表示されます。表示可能なサンプル ウィンドウ数は、全体のサンプル ワード数を前提として、キャプチャ可能な限りの数となります。

注記：トリガー条件全体で、[Occurring in at least ***n*** cycles] または [Lasting for at least ***n*** consecutive cycles] が設定されているカウンターを含む比較ユニットが 1 つでも含まれる場合、ウィンドウのワード数またはトリガーごとのサンプル数を 8 以上にする必要があります。これは、ILA コアに含まれるトリガー ロジックがパイプライン化されているためです。

[Storage Qualification]

ストレージ必要条件とは、コアのトリガー ポートに接続されている比較ユニットのコンパレータで検出されたイベントのブール式の組み合わせです。この条件では、各々のデータ サンプルのキャプチャと保存の実行を決定するトリガー ポートに接続されている比較ユニットのイベントが評価されます。トリガー条件とストレージ必要条件を併用し、キャプチャ プロセスの開始と終了、キャプチャするデータをそれぞれ定義できます。

[Storage Condition] ダイアログ ボックスには、すべての比較ユニットが表示されます。比較ユニットは 1 行ずつ表示されます。[Enable] 列には、比較ユニットがトリガー条件の一部かどうかが示されます。[Negate] 列では、トリガー条件で比較ユニットが個別にネゲート (ブール式の NOT) されるかどうかが示されます。

ストレージ必要条件では、すべてのデータをキャプチャするよう設定するか、またはイネーブルにされているすべての比較ユニットをブール式の AND または OR で組み合わせたものを満たすデータのみをキャプチャするよう設定できます。表の右上もある [Negate Whole Equation] をクリックすると、全体のブール式をネゲートすることもできます。最終的な関数式は、ウィンドウ下部の [Storage Condition Equation] に表示されます。

[Match Functions]

比較演算子では、1 つの比較ユニットに対するトリガー値を定義します。すべての比較演算子は、[Trigger Setup] ウィンドウの [Match Functions] セクションで定義します。[Trigger Conditions] セクションでは、1 つまたは複数の比較演算子をブール式またはシーケンスで定義し、ChipScope Pro コア全体のトリガー条件を指定できます。

[Match Unit]

このフィールドには、比較演算子が適用される比較ユニットが示されます。比較ユニット番号横のアイコンをクリック (またはフィールドをダブルクリック) して、比較ユニットを展開すると、ツリー構造で各トリガー ポート ビットを表示できます。その後、各ビットの値の確認および設定ができます。

[Function]

このボックスでは、比較のタイプを選択します。コンパレータは、比較ユニットで使用可能なときのみ表示されます。

[Value]

このフィールドでは、比較ユニットに適用するトリガー値を選択します。この値は、[Radix] フィールドに基づいて表示されます。ダブルクリックすると、編集できるようになります。変更する値の前にカーソルを置いて値を入力すると、値を上書きできます。または、フィールドを一度クリックした後に、入力します。各種基数で使用可能な文字は、次のとおりです。

- [Hex] : X、0 ～ 9、A ～ F。X は、そのニブルの 4 ビットすべてがドントケア (don't care) であることを示します。クエスチョン マーク (?) は、1、0、X、R、F、および B を組み合わせたニブルであることを示します。
- [Octal] : X、?、0 ～ 7。
- [Binary] : X (ドントケア)、0、1、R (立ち上がり)、F (立ち下がり)、B (いずれかの遷移)、および N (遷移なし) です。R、F、B、および N は、比較ユニットが遷移 ([Basic w/edges]、[Extended w/edges]、[Range w/edges]) を検出できる場合にのみ有効です。
- [Unsigned] :0 ～ 9 (1 つの n ビット バスに対して、0 ～ $2n$-1)
- [Signed] : 0 ～ 9 (1 つの n ビット バスに対して、-2^{n-1} ～ $2^{n-1}-1$)

[Bin] を基数に選択した場合、特殊文字にマウス ポインターを置くと、ビット名や位置を示したツールのヒントが表示されます。

[Radix]

[Value] に表示する基数を選択します。選択肢は、[Hex]、[Octal]、[Bin]、[Signed] ([In Range] 比較および [Out of Range] 比較の場合は使用不可)、[Unsigned] です。

[Counter]

比較ユニットに比較カウンターがある場合は、[Counter] 列のテキスト文字が黒になります。カウンターがない場合は、[Counter] 列のテキストが淡色表示になります。比較カウンターの値を変更するには、カウンター セルをクリックし、比較ユニット カウンターのダイアログ ボックスを表示させます。

このフィールドでは、比較演算子イベントを何度発生させるか選択します。

- [Occurring in exactly *n* clock cycles] を選択した場合は、n 回の連続または n 回の断続イベントが比較演算子カウンターの条件を満たします。
- [Occurring in at least *n* clock cycles] を選択した場合は、n 回の連続または n 回の断続イベントが、比較演算子カウンターの条件を満たし、全体のトリガー条件が満たされるまでこの状態を維持します。
- [Occurring for at least *n* consecutive cycles] を選択した場合は、n 回の連続イベントが、比較演算子カウンターの条件を満たし、全体のトリガー条件が満たされるか、比較演算子の値が満たされなくなるまでこの状態を維持します。

注記：トリガー条件全体で、[Occurring in at least ***n*** cycles] または [Lasting for at least *n* consecutive cycles] が設定されているカウンターを含む比較ユニットが 1 つでも含まれる場合、ウィンドウのワード数またはトリガーごとのサンプル数を 8 以上にする必要があります。これは、ILA コアに含まれるトリガー ロジックがパイプライン化されているためです。

[Trigger Conditions]

トリガー条件は、ブール式または 1 つ以上の比較演算子のシーケンスです。コアでは、トリガー条件に基づいてデータがキャプチャされます。トリガー条件は、複数設定できます。トリガー条件を新規に追加する場合は、[Add] をクリックします。トリガー条件を削除する場合には、セルをハイライトして、[Del] をクリックします。1 つのコアに対して複数のトリガー条件が定義できますが、1 回に選択 (アクティブに) できる トリガー条件は 1 つだけです。

[Active]

[Active] 列には、現在アクティブなトリガー条件を示すボタンが表示されます。

[Trigger Condition Name]

[Trigger Condition Name] 列には、特定のトリガー条件の名前が表示されます。デフォルトでは、デフォルトの「Trigger Condition *n*」と表示されます。

[Trigger Condition Equation]

[Trigger Condition Equation] には、全体のトリガー条件を構成するブール式または比較演算子のステート シーケンスが表示されます。デフォルトのトリガー条件は、存在するすべての比較演算子の論理 AND (各比較ユニットに対して 1 つの比較演算子) です。トリガー条件を変更するには、[Condition Equation] をクリックして、[Trigger Condition] ダイアログ ボックスを表示します。

[Trigger Condition] ダイアログ ボックス

コアにトリガーのシーケンスが含まれる場合、[Trigger Condition] ダイアログ ボックスに [Boolean] と [Sequencer] の 2 つのタブが表示されます。[Boolean] タブを選択すると、トリガー条件として使用可能な比較ユニットのブール式を使用できます。[Sequencer] タブを選択すると、トリガー条件はステート マシンとなり、比較演算子を満たす場合にそれぞれのステート遷移でトリガーされます。

[Boolean] タブには、すべての比較ユニットが表形式で表示されます。比較ユニットは 1 行ずつ表示されます。[Enable] 列には、比較ユニットがトリガー条件の一部かどうかが示されます。[Negate] 列では、トリガー条件で比較ユニットが個別にネゲート (ブール式の NOT) されるかどうかが示されます。

また、イネーブルにしたすべての比較ユニットは、[AND Equation] または [OR Equation] を選択して、ブール式の AND または OR に組み合わせて含めることができます。[Negate Whole Equation] をオンにすると、全体のブール式をネゲートすることもできます。最終的な関数式は、ウィンドウ下部の [Trigger Condition Equation] に表示されます。

[Trigger Condition] ウィンドウの [Sequencer] タブでは、トリガー シーケンスに含めるレベル数を選択するコンボ ボックスと全レベルを示した表が表示されます。シーケンサはレベル 1 から始まり、レベル 1 で指定した比較ユニットが満たされるとレベル 2 に進みます。レベル数はコアのパラメーターで、最大 16 レベルまで設定できます。各レベルでは、比較ユニットが満たされたかどうかが確認されます。特定の比較演算子が存在しないことを確認する場合など、レベルをネゲートするには、[Negate] をオンにします。シーケンスは、ウィンドウ下部の [Trigger Condition Equation] に表示されます。

M0 = M1 = M3 で定義されているトリガー シーケンスは、比較ユニットのイベント M0、M1、M3 が順に発生することで (これらのイベント間にイベントが発生したかどうかに関わらず) 満たすことができます。[Use Contiguous Match Events Only] をオンにすると、M0、M1、M3 の順に連続的に遷移する場合にのみトリガー シーケンスが満たされます。つまり、M0 の後に M1、!M1、M3 に遷移する場合は、トリガー シーケンスは満たされません。

[Output Enable]

コアにトリガー出力がある場合、[Output Enable] 列を設定できます。この列のボックスで、ILA コアの trig_out ポートで駆動する信号タイプを選択します。

- [Disabled] : 出力は常に定数 0 です。
- [Pulse (High)] : ロジック 1 の 1 クロック サイクルのパルスで、トリガー イベントから 10 クロック サイクル後に出力されます。
- [Pulse (Low)] : ロジック 0 の 1 クロック サイクルのパルスで、トリガー イベントから 10 クロック サイクル後に出力されます。
- [Level (High)] : 出力は、トリガー イベントから 10 サイクル後に 0 から 1 に遷移します。
- [Level (Low)] *:* 出力は、トリガー イベントから 10 サイクル後に 1 から 0 に遷移します。

トリガー設定の保存と再利用

[Trigger Setup] ウィンドウのすべての情報は、ファイルに保存して、作業中のプロジェクトまたはほかのプロジェクトで再利用できます。作業中のトリガー設定を保存するには、[Trigger Setup] → [Save Trigger Setup] をクリックし、[Save Trigger Setup As file] ダイアログ ボックスで任意の場所に .ctj という拡張子で保存します。作業中のプロジェクトにトリガー設定を読み込むには、[Trigger Setup] → [Read trigger Setup] をクリックし、[Read Trigger Setup] ダイアログ ボックスでトリガー設定ファイル (.ctj) を選択します。トリガー設定ファイルを選択した後に [開く] をクリックすると、ファイルに保存されている設定が [Trigger Settings] ウィンドウに読み込まれます。

トリガーの実行と待機

トリガーを設定した後は、[Trigger Setup] → [Run] をクリックしてトリガーを作動する状態にします。トリガーは、トリガー条件が満たされるか解除されるまで作動する状態のままに保持されます。トリガー条件が満たされると、キャプチャ設定に従ってデータがコアにキャプチャされます。サンプル バッファーがフルになると、データのキャプチャが停止します。その後、コアからデータが読み込まれ、[Waveform] ウィンドウおよび [Listing] ウィンドウ、またはそのいずれかに表示されます。

トリガーを強制実行するには、[Trigger Setup] → [Trigger Immediate] をクリックします。これで、ユニットでトリガー条件とストレージ必要条件が無視され、トリガー位置がサンプル 0 に設定されているサンプル ウィンドウ 1 つを使用してトリガーがすぐに実行されます。サンプル バッファーのデータがフルになると、トリガーが解除され、キャプチャされたデータが [Waveform] ウィンドウおよび [Listing] ウィンドウ、あるいはそのいずれかに表示されます。

トリガーの中止と解除

トリガーを解除するには、[Trigger Setup] → [Stop Acquisition] をクリックします。[Trigger Setup] → [Run] を続けてクリックすると、トリガーが再設定されます。

注記 : データ取得を中止すると、アクティブなトリガーが解除され、その時点までにキャプチャされたデータが失われます。

トリガー実行モード

ChipScope Pro Analyzer ツールでは、ILA コアで 3 つのトリガー実行モードがサポートされています。

- [Single]
- [Repetitive]
- [Startup]

トリガー実行モードは、[Trigger Setup] → [Trigger Run Mode] または [Trigger Run Mode] ツールバー ボタン ([Trigger Setup] ウィンドウがアクティブなときのみ) で選択できます。

シングル トリガー実行モード

シングル トリガー実行モードでは、ILA コアのトリガーを設定し、[Trigger Setup] ウィンドウでの指定に従いデータをキャプチャし、キャプチャしたデータをアップロード/表示して、停止します。停止後は、ユーザーが通信するときのみ、トリガーが再設定されてプロセスが繰り返されます。

反復トリガー実行モード

反復トリガー実行モードはシングル トリガー実行モードと類似してますが、次の点が異なります。

反復トリガー実行モードでは、トリガーが実行されてキャプチャされたデータがアップロード/表示された後に停止する代わりに、ILA コアのトリガーが自動的に再設定されます。直前のトリガー イベントでキャプチャされたデータは、後続のトリガーが発生しない限り、継続してデータ ビューア - ([Waveform]、[Listing]、[Bus Plat]) に表示されます。後続のトリガーが発生し、データ キャプチャ バッファーがフルになると、データ ビューアーには新しいデータが表示されます。このプロセスは、手動でトリガーを停止しない限り繰り返されます。

反復トリガーの記録

各反復トリガー実行でキャプチャされたデータは、ファイルに記録できます。記録する場合は、[Trigger Setup] → [Setup Repetitive Trigger Logging] をクリックしてダイアログ ボックスを開きます。[Browse] ボタンをクリックして、ログ ファイルの保存場所を選択します。ChipScope Pro Analyzer で生成される各ファイルには、反復トリガー実行の繰り返しに対応するシーケンス番号が含まれています。以前に使用したデータ ログ ファイルに上書きする場合は、[Overwrite any existing files] をオンにします。

データ ログ ファイルのフォーマットは、VCD (Value Change Dump)、ASCII (タブ区切り)、またはアジレント テクノロジー社の FBDF (Fast Binary Data Format) から選択できます。使用するフォーマットのラジオ ボタンをオンにします。

[Signals to Export] ボックスからエクスポートする信号およびバスのセットを選択します。次に、エクスポート可能な信号を示します。

- [All Signals/Buses] : 特定コアの信号およびバス
- [Waveform Signals/Buses] : [Waveform] ビューに表示されている信号とバス
- [Listing Signals/Buses] : [Listing] ビューに表示されている信号とバス
- [Bus Plot Buses] : コアの [Bus Plot] ビューに含まれる信号とバス

シングル トリガー実行モードの信号の記録またはエクスポートについては、データのエクスポート を参照してください。

スタートアップ トリガー実行モード

スタートアップ トリガー実行モードを使用すると、ChipScope Pro Analyzer を使用しなくても、FPGA デバイスの起動後に発生するイベントでトリガーするように ILA コアを設定できます。このモードを使用するには、次の 3 つの手順に従う必要があります。

1. トリガー設定を指定し、スタートアップ トリガー設定 (CTJ) とデザイン制約ファイル (UCF) を保存します。
2. このスタートアップ トリガー設定とデザイン制約ファイルを使用してデザインをインプリメントし直します。
3. ChipScope Pro Analyzer をスタートアップ トリガー実行モードで使用してデバイスをコンフィギュレーションし、トリガー発生後にキャプチャされたデータをアップロードおよび表示します。

トリガー実行モードを使用したトリガー設定を指定するには、まず [Trigger Run Mode] を [Single] に設定します。比較関数、トリガー条件、およびキャプチャを設定します。トリガー設定を指定したら、[Trigger Setup] → [Save Trigger Startup Files] をクリックして、次の必須ファイルを指定します。

- テスト中の FPGA デバイスのデザインに対応する NGD デザインファイル

  Project Navigator を使用している場合、NGD デザイン ファイルは通常 FPGA デバイスをコンフィギュレーションするのに使用した BIT ファイルと同じディレクトリにあります。

  PlanAhead を使用している場合、NGD デザイン ファイルはプロジェクト ディレクトリのサブディレクトリ <project>.runs/<implementation run name> にあります。ここでの <project> はデフォルトで project_1、<implementation run name> は impl_1 を指します。

- トリガー設定を含む CTJ ファイル
- 必要なトリガー設定に該当する ILA コアの初期化設定を含んだ UCF ファイル

注記：デザインをインプリメントするのに PlanAhead ツールを使用する場合は、インプリメンテーション run ディレクトリに CTJ または UCF を保存しないようにしてください。PlanAhead ツールでは、デザインの再インプリメント前に、このディレクトリのファイルがすべて削除されます。

トリガー スタートアップ ファイルを保存すると、ChipScope Pro Analyzer で CTJ トリガー設定が UCF デザイン制約に変換されるので、この制約をデザインに適用する必要があります。この新しい UCF ファイルをプロジェクトに追加し、Project Navigator か PlanAhead ツールを使用してデザインをインプリメントし直す必要があります。

注記：このスタート アップトリガー UCF ファイルは、元の UCF デザイン制約ファイルとは別にプロジェクトに追加しておく必要があります。デザインに追加するスタートアップ トリガー UCF ファイルは、ILA コアにつき 1 つのみにします。

デザインをインプリメントし直し、新しい BIT ファイルを作成したら、ChipScope Pro Analyzer ツールでこの BIT を使用してデバイスをコンフィギュレーションします。スタートアップ トリガー条件が発生したかどうかを確認するには、まず [Trigger Run Mode] を [Startup] に変更します。BIT ファイルを生成するためにインプリメンテーション プロセスで使用されたスタートアップ トリガー ファイルを指定するダイアログボ ックスが表示されます。スタートアップ トリガー実行モードの場合、[Apply Settings and Arm Trigger] および [Stop Acquisition] ツールバー ボタンが黄色でハイライトされ、[Trigger Immediate] ボタンはオフになっています。[Trigger Setup] ウィンドウもオフになっており、BIT ファイルに保存されたリガー設定と矛盾する変更ができないようになっています。

[Run] ツールバー ボタンをクリックすると、ILA コアのステータスが確認できます。ILA コアがトリガーされデータ キャプチャ バッファーがフルになると、キャプチャされたデータがアップロード

および表示されます。ILA コアがトリガーされない場合、またはデータ キ ャプチャ バッファーがフルにならない場合は、ILA コアのステータスがトリガー設定ウィンドウの一番下に表示されます。

注記：スタートアップ トリガー モードは、FPGA デバイスがスタートアップ ステートから変更された直後にのみ設定されます。トリガーを設定し直す場合は、デバイスをコンフィギュレーションし直す必要があります。スタートアップ トリガー設定を変更するには、[Trigger Run Mode] を [Single] に変更し、このセクションで説明した手順を繰り返す必要があります。

トリガーおよびキャプチャ ステータス

各 ILA コアのトリガー ロジックのステータスは、[Trigger Setup] ウィンドウ下部のステータス バーに表示されます。次の情報が表示されます。

- 「Waiting for trigger」(トリガー待機中)、「Capture started」(キャプチャ開始)、または「Slow or stopped clock」(低速またはクロックの停止) などのトリガー ステートがステータス バー左側に表示されます。
- キャプチャしたサンプル数を示すキャプチャ バッファー ステートもスタータス バー左側に表示されます。
- 「SINGLE RUN」(シングル実行)、「REPETITIVE RUN」(反復実行)、または「IDLE」(アイドル) などのトリガー モードは、ステータス バー右側に表示されます。

ILA コアのトリガーおよびキャプチャ ロジックのステートを示すアイコンも表示されます。

- 半分塗りつぶされた円は、トリガーとキャプチャ ロジックの片方または両方がアクティブであることを示します。
- 完全に塗りつぶされた円は、キャプチャ バッファーがフルでトリガー ロジ ックがアイドルであることを示します。

トリガー ステータス アイコンは、最小化表示されている [Trigger Setup] ウィンドウのタイトル バーにも表示されます。この表示は、複数の ILAコアのトリガー ステータスを監視するときに便利です。

トリガーおよびキャプチャ ステータス情報の詳細は、メッセージ ペインにも表示されます。

[Waveform] ウィンドウ

特定の ILA コアの波形を表示するには、[Window] → [New Unit Windows] をクリックしてそのコアを選択します。そのコア ユニット向けにダイアログ ボックスが表示され、[Trigger Setup]、[Waveform]、[Listing]、[Bus Plot] ウィンドウのすべてまたはいずれの組み合わせでも選択して、表示できます。このダイアログ ボックスからはウィンドウを閉じることはできません。プロジェクト ツリーの [Waveform] をダブルクリックするか、または右クリックして [Open Waveform] をクリックしても、波形を表示できます。

[Waveform] ウィンドウには、多くのロジック アナライザーやシミュレータ同様にサンプル バッファーの波形が表示されます。[Waveform] ウィンドウでは、バスの作成、基数選択、名前の変更など、すべての信号ブラウザー操作を実行できます。信号で操作を実行するには、[Bus/Signal] 列の信号またはバスを右クリックします。

バスおよび信号の並べ替え

バスや信号は [Waveform] ウィンドウで並べ替えることができます。信号やバスを選択し、任意の位置にドラッグすると、[Bus/Signal] 列の移動可能な位置に赤線が表示されます。

注記：信号は、内で移動できます。この場合、まずバス内の信号を 1 つまたは複数選択し、Alt + ↑ キーと Alt + ↓ キーを押します。

信号およびバスの切り取り、コピー、貼り付け、削除

信号またはバス上で右クリックして表示されるメニューから、信号やバスの切り取り、コピー、貼り付け、削除を実行できます。1 つまたは複数の信号とバス、あるいはそのいずれかを選択して右クリックし、メニューから任意の作業を実行します。この作業は、Windows のキー操作でも実行できます (Ctrl + X キーで切り取り、Ctrl + C キーでコピー、Ctrl + V キーで貼り付け、Del キーで削除)。

表示の拡大と縮小

[Waveform] → [Zoom] → [Zoom In] をクリックして表示された波形の中央を拡大するか、波形部分で右クリックし、[Zoom] → [Zoom In] をクリックします。波形を縮小表示するには [Waveform] → [Zoom] → [Zoom Out] をクリックするか、波形部分で右クリックし、[Zoom]→ [Zoom Out] をクリックします。

波形全体を表示するには、[Waveform] → [Zoom] → [Zoom Fit] をクリックするか、波形部分で右クリックし、[Zoom] → [Zoom Fit] をクリックします。

波形を部分的に拡大するには、拡大するエリアをドラッグして四角形で囲み、ポップアップ ウィンドウで [Zoom Area] をクリックて拡大します。

X 軸または O 軸で設定した部分を拡大するには、[Waveform] → [Zoom] → [Zoom X] または [Zoom O] をクリックするか、波形部分で右クリックして [Zoom] → [Zoom X] または [Zoom O] をクリックします。その他のズーム機能として、直前の拡大倍数で拡大するには、[Zoom] → [Zoom Previous] をクリックし、その次の拡大倍数で拡大するには [Zoom] → [Zoom Forward] をクリックします。また、特定のサンプル範囲を拡大するには、[Zoom] → [Zoom Sample] をクリックします。

波形の中央寄せ

特定の地点で波形を中央寄せにするには、次の 2 つの方法があります。

- [Waveform] → [Go To] → [Go To X Cursor] または [Go To O Cursor] をクリックして、X マーカーおよび O マーカーを軸にするか、[Go To Trigger] から直前のトリガー位置 ([Previous])、次のトリガー位置 [(Next)] をクリックします。

カーソル

波形ウィンドウでは、2 つのカーソル、X および O が使用できます。カーソルを挿入する場合には、波形上で右クリックし、[Place X Cursor] または [Place O Cursor] をクリックします。垂直線はカーソルの位置を示します。また、その地点の信号やバスのステータスは、X または O 列に示されます。両カーソルの位置とその差異は、[Waveform] ウィンドウ下部に表示されます。両カーソルは、初めはサンプル 0 に配置されています。

カーソルを移動する場合には、波形の別の位置で右クリックするか、波形フォームのヘッダーで X または O ラベルのハンドルをドラッグするか、または波形上でカーソルの線そのものをドラッグします。マウスをカーソルに合わせると、ドラッグ アイコンが表示されます。

サンプルの表示数

波形の横軸に、サンプル ウィンドウ (デフォルト) に対するサンプル数、またはバッファーの総サンプル数が表示されます。各ウィンドウで 0 からサンプル数を表示するには、右クリックしてメニューを表示させ、[Ruler] → [Sample # in Window] をクリックします。バッファーの総サンプル数をサンプル数として表示するには、右クリックしてメニューを表示させ、[Ruler] → [Sample # in Buffer] をクリックします。右クリック メニューの [Ruler] → [Negative Time/Samples] をクリックすると、サンプルの正の範囲と負の範囲を切り替えることができます。

マーカーの表示

トリガーの各地点には、固定された赤い線が垂直方向に表示されます。サンプルがキャプチャされなかった期間を示すウィンドウでは、ウィンドウとウィンドウの間に固定された黒い線が表示されます。これらマーカーのいずれかを非表示する場合は、右クリックでメニューを表示させ、[Markers] → [Window Markers] または [Markers] → [Trigger Markers] をクリックしてオフにします。

データ キャプチャ タイム スタンプ

キャプチャされたデータがアップロードされ表示された日付および時間を示すタイム スタンプは [Waveform] ウィンドウの左下に表示されます。反復トリガー実行モードを使用する場合は、ILA コアがトリガ-されデータがアップロードされた回数を示すシーケンス番号も表示されます。

[Listing] ウィンドウ

特定の ILA コアの [Listing] ウィンドウを表示するには、[Window] → [New Unit Windows] をクリックしてそのコアを選択します。そのコア ユニット向けにダイアログ ボックスが表示され、[Trigger Setup]、[Waveform]、[Listing]、[Bus Plot] ウィンドウのすべてまたはいずれの組み合わせでも選択して、表示できます。このダイアログ ボックスからはウィンドウを閉じることはできません。プロジェクト ツリーの [Listing] をダブルクリックするか、[Listing] を右クリックして [Open Listing] をクリックしても、[Listing] ウィンドウを表示できます。

[Listing] ウィンドウには、サンプル バッファーの値一覧が表形式で表示されます。個々の信号およびバスが列に表示されます。[Listing] ウィンドウでは、バスの作成、基数選択、名前の変更などのすべての信号ブラウザー操作を実行できます。信号で操作を実行するには、列のヘッダーで信号またはバスを右クリックします。

バスおよび信号の並べ替え

バスや信号は [Listing] ウィンドウで並べ替えることができます。表の信号あるいはバスのヘッダーをクリックし、任意の位置にドラッグします。

信号およびバスの削除

信号やバスは、それぞれ列の任意の位置で右クリックし、[Remove] をクリックすると、[Listing] ウィンドウから削除できます。[Remove All] をクリックすると、すべての信号およびバスが削除されます。

カーソル

[Listing] ウィンドウでも、[Waveform] ウィンドウ同様にカーソルを使用できます。カーソルを配置するには、[Listing] ウィンドウのデータ部分で右クリックし、[Place X Cursor] または [Place O Cursor] をクリックします。表には、カーソルと同様の色の線が表示されます。表の任意の位置にカーソルを移動させる場合には、前述と同じ要領でマウスを任意の位置で右クリックするか、最初の列でカーソルのハンドルを右クリックして任意の位置にドラッグします。

Go To カーソル

カーソルの位置まで自動的に [Listing] ウィンドウをスクロールするには、右クリックしてメニューを表示させ、[Go To] → [Go To X Cursor] または 、[Go To] → [Go To O Cursor] をクリックします。

[Bus Plot] ウィンドウ

ILA バスの特定のセットを [Bus Plot] ウィンドウで確認するには、[Window] → [New Unit Windows] をクリックして適切なコアを選択します。そのコア ユニット向けにダイアログ ボックスが表示され、[Trigger Setup]、[Waveform]、[Listing]、[Bus Plot] ウィンドウのすべてまたはいずれの組み合わせでも選択して、表示できます。このダイアログ ボックスからはウィンドウを閉じることはできません。プロジェクト ツリーの [Bus Plot] をダブルクリックするか、またはクリックして [Open Bus Plot] をクリックしても同様に [Bus Plot] ウィンドウを表示できます。

特定のコアのバスはいずれも [Bus Plot] ウィンドウで表示できます。[Bus Plot] ウィンドウで、時間軸で見たバスの値、またはバス間の値比較をグラフ化できます。

[Plot]

ウィンドウ左上のボタンで 2 つのプロット タイプ、[data vs. time] と [data vs. data] を選択できます。[data vs. time] をオンにすると、任意の数のバスをまとめて表示できます。オンにしたときは、2 つのバスを選択し、次に従う必要があります。

- プロットの x 座標 = 特定の時間の片方のバスの値
- y 座標 = 同じ時間のもう一方のバスの値

バスはそれぞれ基数に基づいて色別表示されます。16 進数、2 進数、8 進数、トークン、および ASCII 基数は、スケール係数が 1.0、精度が 0 の符号なし 10 進数として表示されます。

[Display]

バスのグラフは、[lines]、[points]、[lines & points] で表示できます。選択した [Display] のタイプは、表示されるすべてのバスの値に反映されます。

[Bus Selection]

[Bus Selection] で、座標に示すバスを個別に選択して波形を表示するか (data vs. time モード)、または複数のバスを選択して比較表示 (data vs. data モード) できます。各バスの色は、バス名の隣のカラー ボックスをクリックすると変更できます。

[Min/Max]

[Min/Max] で、表示中のバスの軸の最小値および最大値が表示されます。

カーソルのトラッキング

[X:] および [Y:] ボックスは、バス グラフの下部にあり、[Bus Plot] ウィンドウ表示中はマウスのカーソルの現在の x 座標と y 座標の位置を示します。

VIO コアのコンソール ウィンドウ

VIO コアの [Console] ウィンドウを開くには、[Window] → [New Unit Windows] をクリックしてコアを選択します。そのコア ユニット向けにダイアログ ボックスが表示されるので、[Console] を選択します。このダイアログ ボックスからはウィンドウを閉じることはできません。

コンソール ウィンドウは VIO コア用です。特定の VIO コアのコンソールを開くには、プロジェクト ツリーの [VIO Console] をダブルクリックします。このウィンドウでは、VIO コアの入力信号のステータスや動作確認および VIO コアの出力信号のステータスを変更できます。

このウィンドウでは、信号またはバスを右クリックして、バスの作成、基数の選択、名前の変更などの信号ブラウザー操作をすべて実行できます。ウィンドウは、[Bus/Signal] と [Value] の 2 列で構成されています。

[Bus/Signal] 列

[Bus/Signal] には、VIO コアのバス名および信号名が含まれます。バスの場合は、バスを構成する信号の展開/非展開表示、または非表示にできます。さまざまな信号管理に加え、マウスの右クリックでさらに 2 種類のパラメーター、[Type] と [Activity Persistence] を設定できます。

VIO コアのバスおよび信号のタイプ

信号のタイプによってコンソール ウィンドウの [Value] 列での表示方法が異なります。VIO 信号のタイプに応じて、表示されるタイプが異なります。

- VIO 入力信号には、次のような表示タイプがあります。
  - テキスト : ASCII 文字
  - LED
    - LED は、赤、青、緑から選択
    - アクティブ High または アクティブ Low のいずれか
- VIO 入力バスの表示タイプは 1 種類のみ : テキスト
- VIO 出力信号には、次のような制御タイプがあります。
  - テキスト : ASCII 形式のテキスト フィールド
  - プッシュ ボタン (アクティブ High か アクティブ Low のいずれか)
  - トグル ボタン
  - パルス列 (同期出力のみ)
  - シングル パルス (同期出力のみ)
- VIO 出力バスには 2 種類の制御タイプがあります。
  - テキスト
  - パルス列 (同期出力バスのみ)

VIO コアのバスおよび信号のアクティビティ持続時間

信号の持続時間では、[Value] に表示させる信号のアクティビティ時間を示します (信号のアクティビティについては 81 ページの「[Value] 列」を参照)。

[Activity Persistence] では次を設定できます。

- [Infinite] : アクティビティが永久的に表示されます。
- [Long] : サンプル周期 80 回分のアクティビティが表示されます。
- [Short] : サンプル周期 8 回分のアクティビティが表示されます。

設定時間を過ぎると、新規のアクティビティが表示されます。最後のサンプル周期中にアクティビティが発生しなかった場合には、[Value] にアクティビティは表示されません。

バスおよび信号の並べ替え

バスや信号は [Waveform] ウィンドウで並べ替えることができます。信号やバスを選択し、任意の位置にドラッグすると、[Bus/Signal] 列の移動可能な位置に赤線が表示されます。

信号およびバスの切り取り、コピー、貼り付け、削除

マウスを右クリックしてメニューを表示させ、信号とバスのそれぞれで切り取り、コピー、貼り付け、削除を実行できます。信号またはバスにマウスを合わせて右クリックし、メニューから任意の動作を選択するか、標準の Windows のキーの組み合わせを使用します (Ctrl + X キーで切り取り、Ctrl + C キーでコピー、Ctrl + V キーで貼り付け、Del キーで削除)。

[Value] 列

[Value] 列には、コンソール ウィンドウに含まれる各信号の現在値が表示されます。VIO コアの入力では、これらのセルは編集できません。また、バスは選択した基数に基づいて表示されます。VIO コアの入力値は、[JTAG Scan Rate] の選択に従い定期的に更新されます。VIO コアの各入力では、信号の現在値の変化によって、前回の入力確認後のアクティビティ情報がキャプチャされます。高速デザインの場合には、信号が 0 のときにサンプリングされ、0 から 1 に遷移し、再びサンプリングが実行される前に信号が 0 に戻ることが考えられます。

同期入力の場合には、デザイン クロックに対してアクティビティが検知され、グリッチを検知する際に有効です。0 から 1 への遷移が検出された場合、値と共に上向きの矢印が表示されます。また、1 から 0 への遷移が検出された場合には、下向きの矢印が表示されます。その両方を検知すると、両方向矢印が表示されます。表に示されたアクティビティの時間を、信号の持続時間と呼びます。持続時間は、右クリックで表示される [Activity Persistence] メニューから、個別に選択することもできます。

注記：アクティビティを示す矢印は、同期の場合は黒、非同期の場合は赤で示されます。

VIO 信号やバスの値のタイプは、それぞれ右クリックし、[Type] から選択できます。

[Text Field]

[Type] に [Text Field] を選択した場合は、次の文字のみを使用して入力します。

- 信号およびバイナリ バスには 0 または 1
- 16 進数のバスには 0 ～ 9 および A ～ F
- 8 進数のバスには 0 ～ 7
- 有効な符号付きまたは符号なしの整数

[Push Button]

[Type] に [Push Button] を選択した場合は、PCB の実際のプッシュ ボタンがシミュレーションされます。アクティブ High のときに 0、アクティブ Low のときに 1 が押されないと、無効な値が設定されます。ボタンが押されている間は、VIO コアから有効な値が出力されます。

[Toggle Button]

[Type] に [Toggle Button] を選択した場合は、クリックごとに 1 や 0 に変換できます。

[Pulse Train] (同期出力のみ)

[Type] に [Pulse Train] を選択した場合は、同期出力を制御できます。パルス列は 1 および 0 で構成された 16 サイクル列で、ユーザーが定義します。パルス列を変更するには、[Edit] をクリックし、[Pulse Train] ダイアログ ボックスを表示します。パルス列の各サイクルごとにテキスト フィールドが 1 つあります。テキスト フィールドは、前のバスまたは信号の値に基づいて、デフォルトで割り当てられます。バスの場合には、このフィールドは常にバイナリで表示され、個別の信号を明示的に制御します。

[Run] をクリックすると、パルス列が一度に実行されます。これにより、デザイン クロックに対する出力制御の精度が高まります。

[Single Pulse] (同期出力のみ)

[Signal Pulse] は、特殊なプッシュ ボタンです。ボタンが押されると、コアがその間一定の有効な値を 1 つ駆動する代わりに High サイクルを 1 つ含むパルス列が一度だけ実行されます。

VIO コア用のメニューとツールバー

VIO コンソールの使用中は、VIO コア専用メニューおよびツールバーを使用して、必要に応じて VIO コアの入出力動作を変更できます。ツールバーの左から右方向へ順に説明します。

[JTAG Scan Rate]

VIO コアの入力を読み出す際の [JTAG Scan Rate] は、プルダウン リストで選択できます。デフォルトのスキャン レートは 250ms です。サンプリング周期には、[500ms]、[1s]、[2 s]、[5 s]、または [Manual Scan] も設定できます。[Manual Scan] を選択した場合、[Sample Once (S!)] を使用できます。VIO コアの入力を読み出すだけの場合は、ツールバーで [Sample Once (S!)] をクリックするか、または [VIO] → [Sample Once] をクリックします。

[Update Static Outputs]

デフォルトでは、VIO コアの出力のいずれかを変更すると、その出力設定情報がすぐに VIO コアに送信されます。パルス列以外のすべての出力を一度に更新するには、ツールバーで [Update Static Outputs (U!)] をクリックするか、または [VIO] → [Update Static Outputs] をクリックします。

[Reset All Outputs]

すべての出力をデフォルト設定にリセットする (テキスト フィールドとトグル ボタンは 0、パルス列はすべて 0) には、ツールバーで [Reset All Outputs] をクリックするか、[VIO] → [Reset All Outputs] をクリックします。

[Clear All Activity]

VIO コアのすべての入力のアクティビティ表示をリセットする場合は、ツールバーで [Clear All Activity] をクリックするか、または [VIO] → [Clear All Activity] をクリックします。設定してある持続時間にかかわらず、すべての入力のアクティビティがリセットされます。

システム モニター

Virtex®-5、Virtex-6、および Kintex™-7 デバイスには、Xilinx Analog-to-Digital Converter (XADC) と呼ばれるシステム モニター機能が含まれています。XADC は、多数のオンチップ センサーと組み合わせると、オンチップ電源電圧およびチップ温度などの FPGA の物理的な動作パラメーターを計測できます。詳細は、次の資料を参照してください。

- 『Virtex-5 FPGA システム モニター ユーザー ガイド』[211 ページのリファレンス 22 を参照]
- 『Virtex-6 FPGA システム モニター ユーザー ガイド』[211 ページのリファレンス 23 を参照]
- 『7 シリーズ FPGA の XADC 12 ビット 1MSPS デュアル アナログ - デジタル コンバーター』(UG480) [211 ページのリファレンス 13 を参照]

ChipScope Pro Analyzer では、システム モニター プリミティブのオンチップ電圧および温度センサーに JTAG を介してリアル タイムでアクセスできます。すべてのオンチップ センサーは、FPGA デバイスが有効なビットストリームでコンフィギュレーションされる前後で使用できます。システム モニターの機能を使用するのに、デザインにシステム モニター プリミティブ ブロックをインス

タンシエートする必要はありません。ただし、FPGA デバイスのシステム モニター特有ピンがシステム ボードに正しく接続されている必要があります。

ChipScope Pro Analyzer のプロジェクト ツリーでは、JTAG チェーンに含まれる FPGA デバイスに [System Monitor] ノードが表示されます。このノードを右クリックすると、[System Monitor Console] ビューアーを表示するオプションが表示されます。左クリックすると、信号ブラウザーのさまざまなセンサーが表示されます。信号ブラウザでセンサー名や表示単位を変更できます。

[System Monitor Console]

各システム モニター センサーの値は、[System Monitor Console] の [History] 列に表示するか、またはログ ファイルに書き出すことができます。各センサーで次の値を表示できます。

- システム モニターのセンサーから直接読み出される現在の値
- システム モニターのセンサー ピーク検出器により直接読み出されるデバイスの最小/最大値
- JTAG ケーブル接続を開いたとき、または最後にシステム モニターをリセットしたときから ChipScope Pro Analyzer で収集されたすべてのセンサー値から派生するサンプリング最大/最小値
- JTAG ケーブル接続を開いたとき、または最後にシステム モニターをリセットしたときから ChipScope Pro Analyzer で収集されたすべてのセンサー値のスライディング ウィンドウから算出されるウィンドウごとの平均/最大/最小値

[System Monitor Console] で算出されるサンプリング値またはウィンドウごとの値は、ツールバーの [Reset] をクリックするとリセットできます。

注記 : システム モニターで有効なセンサー データがレポートされない場合は、[System Monitor Console] に「Invalid Data」というメッセージが表示されます。

[System Monitor Console] のツールバー

[System Monitor Console] のツールバーおよび右クリック メニューを使用すると、[System Monitor Console] をカスタマイズしたり通信できます。

[JTAG Scan Rate]

システム モニター センサーのデータを読み出す際の [JTAG Scan Rate] は、プルダウン リストで選択できます。デフォルト値は [1 s] ですが、[2 s]、[5 s]、[10 s]、[30 s]、[1 min]、または [Manual Scan] も設定できます。[Manual Scan] を選択した場合、[Sample Once (S!)] を使用できます。システム モニターのデータを読み出す場合は、ツールバーの [Sample Once] をクリックするか、または [System Monitor] → [Sample Once]をクリックします。

[Window Depth]

[System Monitor] ビューアーでのスラインディング ウィンドウの計算で使用されるウィンドウの深さは、ツールバーの [Windows Depth] ボックスに入力するか、または [System Monitor] → [Window Depth] から設定できます。サンプリング ウィンドウの深さは、2、4、8、16、32、64、128 サンプルに設定できます。

[External Input]

System Monitor コンポーネントでは、外部センサーの電圧レベルが監視されます。[External Input] を使用すると、どの外部センサー入力でも一度に 1 つずつ確認できます。有効な外部入力には、次が含まれています。

- 16 個のユーザー定義 VAUXP/VAUXN 外部センサーのいずれか

- V_P/V_N 専用外部センサー
- V_REFP 参照電圧入力
- V_REFN 参照電圧入力

外部入力の表示をディスエーブルにするには [No Input] を選択します (デフォルト)。

注記：7 シリーズ FPGA の XADC ブロックには、外部入力選択で選択可能なオンチップ ブロック メモリ電圧センサー (VCCBRAM) が含まれています。

リセット

クリックすると、[System Monitor Console] の表示がリセットされます。

[Enable Logging]

ツールバーの [Enable Logging] ボタンおよび [System Monitor] → [Enable Logging] メニュー コマンドでは、オフライン解析で使用するシステム モニター センサーのデータをテキスト ファイルに保存するファイル記録機能を使用できます。

システム モニターのデータ記録

[System Monitor] → [Setup Logging] をクリックすると、ダイアログ ボックスが表示されます。このダイアログ ボックスを使用して記録機能の設定をカスタマイズします。

[Log File]

[Browse] ボタンをクリックして、システム モニターのログ ファイルの保存場所を選択します。デフォルトの保存場所は、`< CHIPSCOPE_INSTALL>/bin/<PLATFORM>/system_monitor.log` で、*<CHIPSCOPE_INSTALL>* はインストール ディレクトリ、*<PLATFORM>* はオペレーティング システムのプラットフォーム (nt、nt64、lin、lin64、または sol) を指します。

[Log File Format]

システム モニターのログ ファイルはテキスト ファイルで、マシン処理向け CSV (カンマ区切り) ファイル可読形式のファイルの 2 種類のフォーマットにすることができます。

[Log File Limit]

システム モニターの記録システムでは、ディスク容量を多く消費するデータが生成される可能性があります。この問題を回避するため、ログ ファイル制限に基づいて複数のファイルに分割できます。ログ ファイル制限は、特定のサンプル数またはファイル サイズ (kb) に基づくことができます。

Virtex-5 FPGA GTP および GTX トランシーバー用 IBERT コンソール ウィンドウ

Virtex-5 FPGA GTP および GTX トランシーバー用の IBERT コアのコンソールを開くには、[Window] → [New Unit] → [Windows] でコアを選択します。このコア ユニット向けのダイアログ ボックスが表示されるので [IBERT Console] を選択します。このダイアログ ボックスからはウィンドウを閉じることはできません。

プロジェクト ツリーの [IBERT Console] をダブルクリックするか、または右クリックして [Open IBERT Console] をクリックしてもコンソール ウィンドウを表示できます。

Virtex-5 FPGA GTP および GTX 用の IBERT コンソール ウィンドウは、「[Clock Settings] パネル」、87 ページの「[MGT/BERT Settings] パネル」、および 90 ページの「[Sweep Test Settings] パネル」で構成されています。

[Clock Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では特定のアクティブ GTP_DUAL/GTX_DUAL が表示され、各行では特定の GTP_DUAL/GTX_DUAL 制御またはステータス設定が表示されます。

[CLKP/CLKN Settings]

[CLKP/CLKN Settings] では、特定の GTP_DUAL/GTX_DUAL のリファレンス クロック ピンを表示して制御します。

[GTP_DUAL/GTX_DUAL Alias] には、GTP_DUAL/GTX_DUAL の MGT (マルチギガビット トランシーバー) 番号に初期設定されていますが、新しい値を入力できます。

[GTP_DUAL/GTX_DUAL Location] には、デバイスに含まれる GTP_DUAL/GTX_DUAL の X/Y 座標が表示されます。

[GTP_DUAL/GTX_DUAL Power] では、GTP_DUAL/GTX_DUAL のリファレンス クロック回路の電源を制御します。GTP_DUAL/GTX_DUAL 機能を使用するには、この電源制御を [On] にする必要があります。また、この電源制御はリファレンス クロック ソースまたは中間の通過タイルとしてリファレンス クロックの共有に関わっているいずれの GTP_DUAL/GTX_DUAL でも [On] にする必要があります。

[CLKP/CLKN Coupling] では、GTP_DUAL/GTX_DUAL リファレンス クロック ピンで使用されるカップリングのタイプを制御します。有効な設定は [AC] または [DC] です。

[CLKP/CLKN Freq (MHz)] には、特定の GTP_DUAL/GTX_DUAL コンポーネントの CLKP および CLKN ピンに接続されているリファレンス クロック ソースの周波数が表示されます。デフォルト値は、IBERT コアの生成時に指定されたリファレンス クロックの周波数になります。周波数を変更するには、セルに別の値を入力します。

[REFCLK Settings]

[REFCLK Settings] では、特定の GTP_DUAL/GTX_DUAL のリファレンス クロック入力ソースおよびその他の関連パラメーターを表示して制御します。

[REFCLK Input] では、システムに含まれている有効な GTP_DUAL/GTX_DUAL から特定の GTP_DUAL/GTX_DUAL のリファレンス クロック ソースを選択できます。リファレンス クロック入力を選択するとき、次の規則に従う必要があります。

1. リファレンス クロック ソースは、デスティネーション GTP_DUAL/GTX_DUAL から上下タイル 3 個以内の GTP_DUAL/GTX_DUAL にする必要があります。たとえば、GTP_DUAL_X0Y5 のリファレンス クロックは、3 タイル離れている GTP_DUAL_X0Y2 に設定できても、4 タイル離れている GTP_DUAL_X0Y1 には設定できません。
2. リファレンス クロック ソース GTP_DUAL/GTX_DUAL、デスティネーション GTP_DUAL/GTX_DUAL、およびそれらの間のすべての GTP_DUAL/GTX_DUAL に同じ REFCLK 入力を選択する必要があります。たとえば、GTP_DUAL_X0Y2 で GTP_DUAL_X0Y0 をリファレンス クロック入力ソースとして使用するには、GTP_DUAL_X0Y1 でも GTP_DUAL_X0Y0 をリファレンス クロック入力ソースとして使用する必要があります。
3. リファレンス クロック ソース GTP_DUAL/GTX_DUAL、デスティネーション GTP_DUAL/GTX_DUAL、およびその間にあるすべての GTP_DUAL/GTX_DUAL で [GTP_DUAL/GTX_DUAL Power] を [On] にする必要があります。

[GTP_DUAL/GTX_DUAL PLL Status] では、GTP_DUAL/GTX_DUAL コンポーネントに含まれる PLL のロック ステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[REFCLKOUT Freq (MHz)] では、GTP_DUAL/GTX_DUAL の REFCLKOUT ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[CH0 Clock Status]

[CH0 Clock Status] : 特定の GTP_DUAL/GTX_DUAL のチャネル 0 のさまざまな TX および RX クロック出力のステータスが表示されます。

[TXOUTCLK0 DCM Status] : GTP_DUAL/GTX_DUAL の TXOUTCLK0 ポートに接続されている DCM のロック ステータスが表示されます。このステータス インジケータの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[TXOUTCLK0 Freq (MHz)] : GTP_DUAL/GTX_DUAL の TXOUTCLK0 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[TXUSRCLK0 Freq (MHz)] : GTP_DUAL/GTX_DUAL の TXUSRCLK0 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[TXUSRCLK20 Freq (MHz)] : GTP_DUAL/GTX_DUAL の TXUSRCLK20 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXRECCLK0 DCM Status] : GTP_DUAL/GTX_DUAL の RXRECCLK0 ポートに接続されている DCM のロック ステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[RXRECCLK0 Freq (MHz)] : GTP_DUAL/GTX_DUAL の RXRECCLK0 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXUSRCLK0 Freq (MHz)] : GTP_DUAL/GTX_DUAL の RXUSRCLK0 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXUSRCLK20 Freq (MHz)] : GTP_DUAL/GTX_DUAL の RXUSRCLK20 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[CH1 Clock Status]

[CH1 Clock Status] : 特定の GTP_DUAL/GTX_DUAL のチャネル 1 のさまざまな RX クロック出力のステータスが表示されます。

[RXRECCLK1 DCM Status] : GTP_DUAL/GTX_DUAL の RXRECCLK1 ポートに接続されている DCM のロック ステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[RXRECCLK1 Freq (MHz)] : GTP_DUAL/GTX_DUAL の RXRECCLK1 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXUSRCLK1 Freq (MHz)] : GTP_DUAL/GTX_DUAL の RXUSRCLK1 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXUSRCLK21 Freq (MHz)] : GTP_DUAL/GTX_DUAL の RXUSRCLK21 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[MGT/BERT Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTP/GTX トランシーバー チャネルが表示されます。各行では特定の GTP/GTX または GTP_DUAL/GTX_DUAL 制御またはステータス設定が表示されます。

[MGT Settings]

[MGT Settings] では、特定の GTP_DUAL/GTX_DUAL チャネルのさまざまな設定を表示して制御します。

[MGT Alias] は、GTP/GTX トランシーバー チャネルの MGT 番号およびチャネル番号に初期設定されていますが、新しい値を入力できます。

[GTP_DUAL/GTX_DUAL Location] には、デバイスに含まれる GTP_DUAL/GTX_DUAL の X/Y 座標が表示されます。

[MGT Link Status] には、特定の GTP/GTX トランシーバー チャネルのレシーバーに接続されているリンク検出ロジックのステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[REFCLKOUT PLL Status] には、GTP_DUAL/GTX_DUAL の REFCLKOUT ポートに接続されている PLL のロック ステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[Loopback Mode] では、特定の GTP/GTX トランシーバー チャネルのループバック モードを制御します。次にループバック モードの選択肢を示します。

- [None] : フィードバック パスは使用されません。
- [Near-End PCS] : 回路は近端 GTP/GTX トランシーバー チャネルに完全に含まれています。TX ファブリック インターフェースから開始して、PCS を通過し、GTP/GTX トランシーバー チャネルの PMA サイドを通過せずに RX ファブリックに戻ります。
- [Near-End PMA] : 回路は近端 GTP/GTX トランシーバー チャネルに完全に含まれています。TX ファブリックから開始して、PCS、PMA、PCS を通過し、RX ファブリック インターフェイスに戻ります。
- [Far-End PMA] : 回路は、テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTP/GTX トランシーバー チャネルを通過して元に戻ります。この GTP/GTX トランシーバー ループバック モードでは、信号が RX ピンに入力され、PMA 回路を通過して TX ピンに戻ります。
- [Far-End PCS] : 回路は、テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTP/GTX トランシーバー チャネルを通過して元に戻ります。この GTP/GTX トランシーバー ループバック モードでは、信号が RX ピンに入力され、PMA、PCS、PMA を通過して TX ピンに戻ります。
- [Far-End Fabric] : 回路は、テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTP/GTX トランシーバー チャネルおよび関連するファブリック ロジック

を通過して元に戻ります。この GTP/GTX トランシーバー ループバック モードでは、信号が RX ピンに入力され、PMA、PCS、ワード数が少ないファブリック ベースの FIFO、PCS、PMA を通過して TX ピンに戻ります。

[Channel Reset] ボタンをクリックすると、内部 PMA および PCS 回路に加え関連するファブリック インターフェイスがクリア、リセットされ、GTP/GTX トランシーバー チャネルがリセットされます。

MGT のすべての属性をダイナミック リコンフィギュレーションポート (DRP) から表示して変更できます。[Edit DRP] ボタンをクリックすると、その GTP_DUAL/GTX_DUAL の [Edit DRP] ダイアログ ボックスが表示されます。

[By Attribute Name] タブの [Attribute Name] から表示または変更する属性を選択します。属性はアルファベット順に表示されます。属性を選択すると、その時点の DRP が読み出され、その属性の現在の値が [Current Value] に表示されます。ダイアログ ボックス下部にあるラジオ ボタンでは、2 つの値フィールドの基数が示されています。新しい値を入力するには、基数タイプ ([Binary Value]、[Hex Value]、[UCF Value]) を選択してから [New Value] にテキストを入力して、[Apply] をクリックします。

特定の属性ではなく特定の DRP アドレスを変更する場合は、[By DRP Address] タブをクリックします。このタブは、上級ユーザー向けです。[Address] から変更するアドレスを選択します。ラジオ ボタン ([Hex]/[Binary]) の選択に従い現在の値が 16 進数または 2 進数で表示されます。値を変更するには、[New Value] に新しい値を入力して、[Apply] をクリックします。ダイアログ ボックスを閉じるには [Close] をクリックします。

[Show Settings] をクリックすると、関連する GTP/GTX トランシーバー チャネル ポートの現在の設定および DRP 属性設定が表示されます。[Export Settings] をクリックすると、これらの設定がファイルにエクスポートされます。

[TX/RX Termination] では、GTP トランシーバー チャネルの終端を選択します。選択できる設定は [50W] と [75W] です。この設定は、Virtex-5 LXT/SXT ファミリの GTP トランシーバーでのみ使用できます。Virtex-5 FXT ファミリの GTX トランシーバーでは使用できません。

[Edit Line Rate] ボタン (メモを参照) をクリックすると、GTP_DUAL/GTX_DUAL のライン レートおよびさまざまな PLL 設定に関連しているパラメーターを指定できます。ライン レートを編集すると、GTP_DUAL/GTX_DUAL コンポーネント内の両チャネルにその設定が適用されます。[GTP_DUAL/GTX_DUAL] は、GTP_DUAL/GTX_DUAL コンポーネントを選択するのに使用します。[REFCLK Input Freq (MHz)] は読み取り専用フィールドで、入力リファレンス クロックの周波数が表示されます。[Internal Data Width] は、GTP_DUAL コンポーネントでは [10 bit]、GTX_DUAL コンポーネントでは [20 bit] に固定されています。[Target Line Rate (Mb/s)] には、REFCLK 入力周波数から派生した有効なライン レートと関連する PLL 設定 ([FB]、[REF]、および [DIVSEL]) がすべて表示されます。[PLL VCO Freq (MHz)] には、PLL の電圧制御オシレーター (VCO) の出力周波数が表示されます。[Edit Line Rate] ウィンドウ下部には、GTP_DUAL/GTX_DUAL のライン レートに関するすべての属性が表示されます。

注記：[Edit Line Rate] ボタンは Virtex-5 FPGA GTP および GTX トランシーバー用 IBERT の v1.0 でのみ使用でき、v2.0 では使用できません。v2.0 コアでライン レートを変更する場合は、任意のライン レートと REFCLK 情報を使用してコアを再生成し直し、デザインでタイミングが満たされることを確認する必要があります。

[Coding] では、GTP/GTX トランシーバー チャネルの TX 側および RX 側それぞれで使用されるエンコードおよびデコードのタイプを選択します。選択肢は、[None] および [8B/10B] です。

注記：8B/10B エンコード/デコードがイネーブルの場合は、フレーム付きカウンター パターンおよびアイドル パターンのみを使用できます。

[TX Settings]

[TX Settings] では、特定の GTP/GTX トランシーバー チャネルのさまざまな TX 設定を表示して、制御します。

[TXOUTCLK0 DCM Status] には、GTP_DUAL/GTX_DUAL の TXOUTCLK0 ポートに接続されている DCM のロック ステータスが示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[Invert TX Polarity] では、GTP/GTX トランシーバー チャネルの TX ピンから送信されたデータの極性を制御します。GTP/GTX トランシーバーの TX 側の極性を反転するには、[Invert TX Polarity] をオンにします。

[Inject TX Bit Error] をクリックすると、1 つの送信ワードに含まれる 1 ビットの極性が反転されます。このトランスミッターに接続されているチャネルのレシーバー エンドポイントでは、1 ビット エラーが検出されるはずです。

[TX Diff Boost] では、TX_DIFF_BOOST 属性のステータスを制御します。この属性は、GTP/GTX トランシーバー チャネルの [TX Diff Output Swing] (TXDIFFCTRL および TXBUFDIFFCTRL ポートで制御されるトランスミッターの差動出力幅としても知られる) および [TX Pre-Emphasis] (TXPREEMPHASIS ポートで制御されるトランスミッターのプリエンファシスとしても知られる) ポート設定を向上させるために使用します。[TX Diff Boost] は、Virtex-5 LXT/SXT ファミリの GTP トランシーバーでのみ使用できます。Virtex-5 FXT ファミリの GTX トランシーバーでは使用できません。これらの設定で値の有効な組み合わせは、『Virtex-5 FPGA RocketIO GTP トランシーバー ユーザー ガイド』[211 ページのリファレンス 2 を参照] または『Virtex-5 FPGA RocketIO GTX トランシーバー ユーザー ガイド』[211 ページのリファレンス 3 を参照] を参照してください。

[RX Settings]

[RX Settings] では、特定の GTP/GTX トランシーバー チャネルのさまざまな RX 設定を表示して制御します。

[RXOUTCLK DCM Status] では、GTP/GTX トランシーバー チャネルの RXOUTCLK ポートに接続されている DCM のロック ステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[Invert RX Polarity] では、GTP/GTX チャネルの RX ピンから受信したデータの極性を制御します。GTP/GTX の RX 側の極性を反転するには、[Invert RX Polarity] をオンにします。

[RX Coupling] および [RX Termination Voltage] を共に使用して GTP/GTX チャネル レシーバーのカップリングおよび終端ネットワークを制御します。これらの設定で値の有効な組み合わせは、『Virtex-5 FPGA RocketIO GTP トランシーバー ユーザー ガイド』[211 ページのリファレンス 2 を参照] または『Virtex-5 FPGA RocketIO GTX トランシーバー ユーザー ガイド』[211 ページのリファレンス 3 を参照] を参照してください。

[Enable RX EQ] は、GTP/GTX チャネルのレシーバーのイコライゼーションをイネーブルにします。レシーバーのイコライゼーションをイネーブルにすると、[RX EQ WB/HP Ratio] および [RX EQ HP Pole Loc] を使用して、GTP/GTX チャネル レシーバーの広帯域/ハイ パス フィルター率とハイ パス フィルターの極位置を制御できます。これらの設定で値の有効な組み合わせは、『Virtex-5 FPGA RocketIO GTP トランシーバー ユーザー ガイド』[211 ページのリファレンス 2 を参照] または『Virtex-5 FPGA RocketIO GTX トランシーバー ユーザー ガイド』[211 ページのリファレンス 3 を参照] を参照してください。

[RX Sampling Point] スライダーでは、PMA_CDR_SCAN 属性を変更することでトランシーバーのクロック データ リカバリ (CDR) ユニットの水平サンプリング ポイントを制御します。スライダー制御の整数値は、現在の設定が表示され、0 ～ 127 の値を指定できます。0 は UI (ユニット イ

ンターバル) の最も左のサンプル位置、127 は UI の最も右のサンプル位置を意味します。UI の位置は、スライダ制御の右側にも表示されます。

注記：[RX Sampling Point control] は、GTP_DUAL/GTX_DUAL チャネルの PLL_RXDIVSEL_OUT属性が 1 に設定されているときのみイネーブルにされます。PLL_RXDIVSEL_OUT 属性の設定を確認する場合は、[Edit Line Rate] を使用します。

[BERT Settings]

[BERT Settings] では、特定の GTP/GTX トランシーバー チャネルのさまざまなビット エラー率設定を表示し制御します。

[TX/RX Data Pattern] では、トランスミッターのパターン ジェネレーターおよびレシーバのパターン チェッカーで使用するデータ パターンを選択します。使用できるパターン タイプは、IBERT コアの生成中にイネーブルにされるパターンによって異なりますが、PRBS (Pseudo Random Bit Sequence) 7 ビット ($X^7 + X^6 + 1$)、代替 PRBS 7 ビット ($X^7 + X + 1$)、PRBS 9 ビット、PRBS 11 ビット、PRBS 15 ビット、PRBS 20 ビット、PRBS 23 ビット、PRBS 29 ビット、PRBS 31 ビット、ユーザー パターン (クロック パターンを含む、任意の 20 ビット データ パターンの生成に使用可能)、フレーム付きカウンター、およびアイドル パターンが含まれます。

注記：8B/10B エンコード/デコードがイネーブルの場合は、フレーム付きカウンター パターンおよびアイドル パターンのみを使用できます。

[RX Bit Error Ratio] には、GTP/GTX トランシーバー チャネルに対して算出されたビット エラー率が含まれています。この値は指数で表現されます。たとえば、1.000E-12 は 1 兆ビット受信するたびに 1 ビット エラーが発生することを意味します。

[RX Line Rate] には、GTP/GTX トランシーバー チャネル向けに算出されたライン レートが表示されます。デザインの時間を計測するのにシステム クロックが使用されるため、不正確または不安定なシステム クロックを使用すると、この値が不正になったりまたは変動します。リンクがない場合は、「N/A」と表示されます。

[RX Received Bit Count] には、受信したビット数の総計が表示されます。このカウントは、[BERT Reset] をクリックしたときにリセットされます。

[RX Bit Error Count] には、検出されたビット エラー数の総計が表示されます。このカウントは、[BERT Reset] をクリックしたときにリセットされます。

このボタンをクリックすると、ビット エラー カウンターおよび受信ビット カウンターがリセットされます。GTP/GTX トランシーバー チャネルがリンクされ安定したら、BERT カウンターをリセットしてください。

[Sweep Test Settings] パネル

このパネルでは、Virtex-5 FPGA GTP および GTX トランシーバーのさまざまな設定をスイープするチャネル テストを設定します。TX および RX の両方の設定のスイープは、トランシーバーが近端または外部ループバック モードのいずれかに設定されているときのみ機能します。RX パラメーターのスイープは、リンクの対応する TX エンドポイントが別のデバイスまたは同じデバイスに含まれる別のトランシーバーに含まれるときのみ実行できます。

このパネルは [Sweep Test Setup] および [Sweep Test Results] サブパネルの 2 つのセクションから構成されています。

[Sweep Test Setup]

[Sweep Test Setup] では、スイープ対象、スイープ テスト結果ファイルの設定、およびスイープ実行時間を設定します。

スイープ パラメーターの設定

次に、スイープに使用するトランシーバーのパラメーターを示します。

- [TX Diff Boost] (GTP のみ)
- [TX Diff Output Swing]
- [TX Pre-Emphasis]
- [RX EQ Enable]
- [RX EQ WB/HP Ratio]
- [RX EQ HP Pole Loc]
- [RX Sampling Point]

これらのスイープ パラメーターは、次のいずれかの方法で初期化できます。

- [Clear All Parameters] をクリックしてすべてのパラメーターをクリアします。
- [Set Parameters to Current Values] をクリックして、[MGT/BERT Settings] パネルの現在の値にパラメーターを設定します。

[Sweet Parameter] 表のパラメーターの順序は、パラメータのスイープ順序を示しています。表上部のパラメーターの値は、表下部のパラメーターに比べスイープされる頻度が低くなります。つまり、表上部のパラメーターはスイープ アルゴリズムの外側ループにあり、表下部のパラメーターはスイープ アルゴリズムの内側ループにあるということです。表のパラメーターの順序は変更できません。

各パラメーターで開始値と終了値を設定する必要があります。パラメーターの順序は変更できません。開始値を選択すると、その値に対して有効な終了値が自動的に表示されます。パラメーターでスイープの設定を希望しない場合は、開始値と終了値を同じ値にします。[Sweep Value Count] 列では、特定のパラメーターでスイープされる値の数が示されます。すべてのスイープ パラメーターに有効な開始値と終了値が設定されると、スイープ実行回数が [Total Iterations] に表示されます。

スイープ テスト結果ファイルの設定

スイープテストの結果は、[Sweep Test Status] パネルに表示されます。またスイープ テスト結果ファイルに書き出すこともできます。[Sweep Test Result File Settings] をクリックすると、ダイアログ ボックスが表示されます。

このダイアログ ボックスでは、ファイルの保存場所と各ファイルに含めるスイープ実行回数を設定できます。スイープ実行回数がファイルの制限を越える場合は、最初の結果ファイルと同じディレクトリにベース ファイル名の開始実行番号が付けられた複数のファイルが作成されます。

スイープ テスト結果

スイープ テストを設定したら、[Start] をクリックしてテストを開始します。クリックした後は、スイープ パラメーター表がディスエーブルになりテストが実行されます。

スイープ テストの実行の過程で現在のスイープ結果ファイル、現在の実行回数、経過時間、および概算残り時間ステータスが表示されます。スイープ結果は、画面下のテキスト エリアに表示されます。スイープ テストは [Pause] をクリックして一時停止したり、[Reset] をクリックして完全に停止することができます。

IBERT のツールバーおよびメニュー

[Reset All]

IBERT コアのすべてのチャネルをリセットするには、[IBERT_V5GTP/GTX] → [Reset All] をクリックするか、またはツールバーの [Reset All] をクリックします。

[JTAG Scan Rate] および [Scan Now]

ツールバーの [JTAG Scan Rate] および [IBERT_V5GTP/GTX] → [JTAG Scan Rate] メニューをクリックすると、ChipScope Pro Analyzer ツールで IBERT コアのステータス情報を確認する頻度を選択できます。デフォルトは [1 s] ですが、[250 ms]、[500 ms]、[2 s]、[5 s]、または [Manual Scan] も選択できます。[Manual Scan] を選択した場合、[IBERT_V5GTP/GTX] → [Scan Now] またはツールバーの [Scan Now] ([S!]) をクリックすると、IBERT コアのクエリをすぐに実行できます。

Virtex-5 FPGA GTX トランシーバー用 IBERT v2.0 コンソール ウィンドウ

Virtex-5 FPGA GTX トランシーバー用の IBERT v2.0 コアのコンソールを開くには、[Window] → [New Unit Windows] でコアを選択します。このコア ユニット向けのダイアログ ボックスが表示されるので [IBERT V5 GTX Console] を選択します。このダイアログ ボックスからはウィンドウを閉じることはできません。

プロジェクト ツリーの [IBERT V5 GTX Console] をダブルクリックするか、または右クリックして [Open IBERT V5 GTX Console] をクリックしてもコンソール ウィンドウを表示できます。

Virtex-5 FPGA GTX トランシーバー用 IBERT v2.0 コンソール ウィンドウは、次から構成されています。

- [MGT/BERT Settings] パネル
- [DRP Settings] パネル
- [Port Settings] パネル
- [Sweep Test Settings] パネル
- IBERT v2.0 Virtex-5 FPGA GTX トランシーバーのツールバーおよびメニュー オプション

[MGT/BERT Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTX トランシーバーが表示されます。各行では特定の制御またはステータス設定が表示されます。

[MGT Settings]

[MGT Alias] は、GTX トランシーバー MGT 番号に初期設定されていますが、新しい値を入力できます。

[Tile Location] には、デバイスに含まれる GTX トランシーバーの X/Y 座標が表示されます。

[MGT Link Status] には、特定の GTX トランシーバー チャネルのレシーバーに接続されているリンク検出ロジックのステータスが表示されます。チャネルがリンクされている場合は緑色で計測されたライン レートが表示され、リンクされていない場合は「NOT LOCKED」(赤色) と表示されます。

[PLL Status] には GTX トランシーバーに接続されている PLL のロック ステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[Loopback Mode] では、特定の GTX トランシーバー チャネルのループバック モードを制御します。次にループバック モードの選択肢を示します。

- [None] : フィードバック パスは使用されません。
- [Near-End PCS] : 回路は近端 GTX トランシーバー チャネルに完全に含まれています。TX ファブリック インターフェースから開始して、PCS を通過し、GTX トランシーバー チャネルの PMA 側を通過せずに RX ファブリックに戻ります。
- [Near-End PMA] : 回路は近端 GTX トランシーバー チャネルに完全に含まれています。TX ファブリックから開始して、PCS、PMA、PCS を通過し、RX ファブリック インターフェイスに戻ります。
- [Far-End PMA] : 回路は、外部テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTX トランシーバー チャネルを通過して元に戻ります。この GTX トランシーバー ループバック モードでは、信号が RX ピンに入力され、PMA 回路を通過して TX ピンに戻ります。
- [Far-End PCS] : 回路は、外部テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTX トランシーバー チャネルを通過して元に戻ります。この GTX ループバック モードでは、信号が RX ピンに入力され、PMA、PCS、PMA を通過して TX ピンに戻ります。

[DUAL Reset] をクリックすると、内部 PMA および PCS 回路に加え関連するファブリック インターフェイスがクリア、リセットされ、DUAL に含まれる GTX トランシーバーがリセットされます。

[Channel Reset] をクリックすると、内部 PMA および PCS 回路に加え関連するファブリック インターフェイスがクリア、リセットされ、GTX トランシーバー チャネルがリセットされます。

[TX Polarity Invert] では、GTX トランシーバー チャネルの TX ピンから送信されたデータの極性を制御します。GTX トランシーバーの TX 側の極性を反転するには、このチェック ボックスをオンにします。

[TX Bit Error Inject] をクリックすると、1 つの送信ワードに含まれる 1 ビットの極性が反転されます。このトランスミッターに接続されているチャネルのレシーバー エンドポイントでは、1 ビット エラーが検出されるはずです。

[TX Diff Output Swing] では、トランスミッターの差動振幅を制御します。値を変更する場合はボックスに入力します。

[TX Pre-Emphasis] では、送信信号のプリエンファシス量を制御します。値を変更する場合はボックスに入力します。

[RX Polarity Invert] では、GTX チャネルの RX ピンから受信したデータの極性を制御します。GTX トランシーバーの RX 側の極性を反転するには、このチェック ボックスをオンにします。

[RX AC Coupling Enabled] では、ビルトイン AC カップリング キャパシタをイネーブルにするかどうかを制御します。

[RX Termination Voltage] では、RX 終端ネットワークで使用する電源を制御します。

[RX Equalization] では、内部 RX イコライゼーション回路を制御します。

[BERT Settings]

[TX Data Pattern] および [RX Data Pattern] は、トランスミッターのパターン ジェネレーターおよびレシーバーのパターン チェッカーで使用されるデータ パターンを選択するのに使用します。これらのパターンには、PRBS7、15、23、31、および Clk 2x、10x が含まれています。

[RX Bit Error Ratio] には、GTX トランシーバー チャネルに対して算出されたビット エラー率が表示されます。この値は指数で表現されます。たとえば、1.000E-12 は 1 兆ビット受信するたびに 1 ビット エラーが発生することを意味します。

[RX Received Bit Count] には、受信したビット数の総計が示されています。このカウントは、[BERT Reset] をクリックしたときにリセットされます。

[RX Bit Error Count] には、検出されたビット エラー数の総計が示されています。このカウントは、[BERT Reset] をクリックしたときにリセットされます。

このボタンをクリックすると、ビット エラー カウンターおよび受信ビット カウンターがリセットされます。GTX チャネルがリンクされ安定したら BERT カウンターをリセットしてください。

[Clocking Settings]

[TX DCM Reset] では TXOUTCLK 出力を使用して TXUSRCLK および TXUSRCLK2 クロックを生成する DCM をリセットします。

[RX DCM Reset] では RXRECCLK 出力を使用して RXUSRCLK および RXUSRCLK2 クロックを生成する DCM をリセットします。

[TXUSRCLK Freq (MHz)] には、GTX トランシーバーの TXUSRCLK ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[XUSRCLK2 Freq (MHz)] には、GTX トランシーバーの TXUSRCLK2 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXUSRCLK Freq (MHz)] には、GTX トランシーバーの RXUSRCLK ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXUSRCLK2 Freq (MHz)] には、GTX トランシーバーの RXUSRCLK2 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[DRP Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTX トランシーバーが表示されます。各行では、特定の DRP 属性またアドレスが表示されます。

パネル下部の [View By Attribute Name] をオンにすると、すべての DRP 属性がアルファベット順で表示されます。[Radix] では [Hex] (16 進数) または [Bin] (2 進数) のいずれかを選択できます。値を変更するには、テキスト フィールドをクリックして新しい値を入力してから、Enter キーを押します。新しい値が MGT トランシーバーに反映されます。

パネル下部の [View By Address] がオンになっていると、行アドレスが数字順に表示されます。[Radix] では [Hex] (16 進数) または [Bin] (2 進数) のいずれかを選択できます。値を変更するには、テキスト フィールドをクリックして新しい値を入力してから、Enter キーを押します。新しい値が MGT トランシーバーに反映されます。

[Port Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTX トランシーバーが表示されます。各行には、特定の MGT ポートが表示されます。一部のポー

トは IBERT デザインでデータの送受信に使用されるため、すべてのポートが表示されるわけではありません。

[Radix] では、値の基数に [Hex] (16 進数) または [Bin] (2 進数) を選択できます。一部のポートは読み取り専用で編集できません。これらのセルは、ラベルのように表示されます。編集可能な表内のポートは、テキスト フィールドのように表示され、カーソルを置くと新しい値が入力でき、Enter キーを押すとその値がすぐに反映されます。

[Sweep Test Settings] パネル

このパネルは、さまざまなトランシーバー設定をスイープするチャネル テストを設定するのに使用します。TX および RX 設定は、同じ GTX トランシーバー向けです。TX および RX の両方の設定のスイープは、トランシーバーが近端または外部ループバック モードのいずれかに設定されているときのみ機能します。RX パラメーターのスイープは、リンクの対応する TX エンドポイントが別のデバイスまたは同じデバイスに含まれる別のトランシーバーに含まれるときのみ実行できます。

このパネルは、[Parameter Settings]、[Sampling Point Region]、[Test Controls]、および [Test Results] セクションの 4 つのセクションから構成されています。

[Parameter Settings]

このセクションでは、スイープ パラメーターを設定します。

次に、スイープに使用できる GTX トランシーバーのパラメーターを示します。

- [TX Diff Swing]
- [TX Pre-Emphasis]
- [RX EQ]
- [DFETAP1]
- [DFETAP2]

これらのスイープ パラメーターは、次のいずれかの方法で初期化できます。

- [Clear All Parameters] をクリックして [Select] オプションのすべてのパラメーターをクリアします。
- [Set Parameters to Current Values] をクリックして、[MGT/BERT Settings] パネルの現在の値にパラメーターを設定します。

[Sweet Parameter] 表のパラメーターの順序は、パラメーターのスイープ順序を示しています。表上部のパラメーターの値は、表下部のパラメーターに比べスイープされる頻度が低くなります。つまり、表上部のパラメーターはスイープ アルゴリズムの外側ループにあり、表下部のパラメーターはスイープ アルゴリズムの内側ループにあるということです。

各パラメーターで開始値と終了値を設定する必要があります。パラメーターの順序は変更できません。開始値を選択すると、その値に対して有効な終了値が自動的に表示されます。パラメーターでスイープの設定を希望しない場合は、開始値と終了値を同じ値にします。[Sweep Value Count] 列では、特定のパラメーターでスイープされる値の数が示されます。すべてのスイープ パラメーターに有効な開始値と終了値が設定されると、スイープ実行回数が [Total Iterations] に表示されます。

パラメーターを表に追加または削除するには、[Add/Remove Parameters] をクリックします。ダイアログ ボックスが表示され、左側には使用可能なポート/属性すべて、右側にはスイープするパラメーターが表示されます。スイープするパラメーターを追加するには、左側のリストからいずれかをクリックして、右方向 (>) ボタンをクリックします。スイープするパラメーターを削除するには、右側のリストからいずれかをクリックして、左方向 (<) ボタンをクリックします。パラメーターの

スイープ順を指定するには、右側のリストのいずれかをクリックしてから [Up] または [Down] をクリックします。スイープ属性およびその順序をデフォルトの設定に戻す場合は、[Reset to Default] をクリックします。[OK] をクリックして設定を適用するか、[Cancel] をクリックして保存せずに終了します。

[Sampling Point Region]

サンプリング ポイントは、アイとサンプル間の水平ポイントです。左端の遷移領域を視覚化してから、右端を視覚化します。[RX Sampling Point] は、128 個の不連続のサンプリング位置の 1 つです。このセクションでは、サンプリング領域の左端および右端を選択します。

[Test Controls]

スイープ テストを設定したら、[Start] をクリックしてテストを開始します。クリックした後は、スイープ パラメーター表がディスエーブルになりテストが実行されます。

スイープ テストの実行の過程で現在のスイープ結果ファイル、現在の実行回数、経過時間、および概算残り時間ステータスが表示されます。スイープ結果は、画面下のテキスト エリアに表示されます。スイープ テストは [Pause] をクリックして一時停止したり、[Reset] をクリックして完全に停止することができます。

スイープテストの結果は、[Test Results] パネルに表示されます。またスイープ テスト結果ファイルに書き出すこともできます。[Log File Settings] をクリックすると、ダイアログ ボックスが表示されます。

このダイアログ ボックスでは、ファイルの保存場所と各ファイルに含めるスイープ実行回数を設定できます。スイープ実行回数がファイルの制限を越える場合は、最初の結果ファイルと同じディレクトリにベース ファイル名の開始実行番号が付けられた複数のファイルが作成されます。

[Test Results]

このパネルには、現在の実行、経過時間、および残り時間の概算が表示されます。このステータス情報の下には、スイープ テスト結果の実行ログが表示されています。この結果もログ ファイルに保存されます。

IBERT v2.0 Virtex-5 FPGA GTX トランシーバーのツールバーおよびメニュー オプション

[IBERT Console Options]

[IBERT Console Options] ダイアログ ボックスでは、コンソール ウィンドウに表示する列および行を選択できます。左側パネルでは、MGT をロケーションで選択します。すべて選択する場合は [Check All]、すべて選択しない場合は [Uncheck All] をクリックします。

右側パネルでは、[MGT/BERT Settings] パネルに表示される行を選択します。すべて選択する場合は [Check All]、すべて選択しない場合は [Uncheck All] をクリックします。[Default] をクリックすると、チャネルの有効性を決定するのに必要な行の基本セットがコンソール ウィンドウに表示されます。

[Import/Export] ダイアログ ボックス

このダイアログ ボックスでは、特定の MGT の設定を保存して復元したり、デザインの別の MGT に適用できます。設定をインポートまたはエクスポートするには、[IBERT_V5GTX] → [Import/Export Wizard] をクリックするか、またはツールバーの [Import/Export Wizard] をクリックします。

ウィザードの最初の画面では、MGT 設定のソースに [MGT] または [File] のいずれかを選択します。[MGT] を選択した場合は、コンボ ボックスに表示される MGT から選択します。[File] を選択した場合は、[Browse] をクリックして設定ファイルを指定します。[Next] をクリックして、次の画面に進みます。

次の画面では、デスティネーションを指定します。IBERT デザインに含まれる MGT とファイルを自由に組み合わせることができます。[File] をイネーブルした場合は、[Browse] をクリックしてファイル デスティネーションを指定します。

3 番目の画面には、設定のソースとデスティネーションのサマリが表示されます。[Apply] をクリックしてインポートまたはエクスポートします。この操作は、元に戻すことができません。

[Reset All]

IBERT コアのすべてのチャネルをリセットするには、[IBERT_V5GTX] → [Reset All] をクリックするか、またはツールバーの [Reset All] をクリックします。

[JTAG Scan Rate] および [Scan Now]

ツールバーの [JTAG Scan Rate] および [IBERT_V5GTX] → [JTAG Scan Rate] をクリックすると、ChipScope Pro Analyzer ツールで IBERT コアのステータス情報を確認する頻度を選択できます。デフォルトは [1 s] ですが、[250 ms]、[500 ms]、[2 s]、[5 s]、または [Manual Scan] も選択できます。[Manual Scan] を選択した場合、[IBERT_V5GTX] → [Scan Now] をクリックするかツールバーの [Scan Now] ([S!]) をクリックして IBERT コアのクエリをすぐに実行できます。

Virtex-6 FPGA GTX トランシーバー用 IBERT コンソール ウィンドウ

Virtex-6 LXT/SXT/CXT ファミリの IBERT コアのコンソールを開くには、[Window] → [New Unit] → [Windows] でコアを選択します。このコア ユニット向けのダイアログ ボックスが表示されるので [IBERT V6 GTX Console] を選択します。このダイアログ ボックスからはウィンドウを閉じることはできません。

プロジェクト ツリーの [IBERT V6 GTX Console] をダブルクリックするか、または右クリックして [Open IBERT V6 GTX Console] をクリックしてもコンソール ウィンドウを表示できます。

Virtex-6 LXT/SXT/CXT ファミリの GTX トランシーバー用 IBERT コンソール ウィンドウは、「[MGT/BERT Settings] パネル」、99 ページの「[DRP Settings] パネル」、100 ページの「[Port Settings] パネル」、および 103 ページの「IBERT Virtex-6 FPGA GTX トランシーバーのツールバーおよびメニュー オプション」で構成されています。

[MGT/BERT Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTX トランシーバーが表示されます。各行では特定の制御またはステータス設定が表示されます。

[MGT Settings]

[MGT Alias] は、GTX トランシーバー MGT 番号に初期設定されていますが、新しい値を入力できます。

[Tile Location] には、デバイスに含まれる GTX トランシーバーの X/Y 座標が表示されます。

[MGT Link Status] には、特定の GTX トランシーバー チャネルのレシーバーに接続されているリンク検出ロジックのステータスが表示されます。チャネルがリンクされている場合は緑色で計測されたライン レートが表示され、リンクされていない場合は「NOT LOCKED」(赤色) と表示されます。

[TX PLL Status] には GTX トランシーバーに接続されている TX PLL のロック ステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[RX PLL Status] には GTX トランシーバーに接続されている RX PLL のロック ステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[Loopback Mode] 設定は、特定の GTX トランシーバー チャネルのループバック モードを制御するのに使用します。次にループバック モードの選択肢を示します。

- [None] : フィードバック パスは使用されません。
- [Near-End PCS] : 回路は近端 GTX トランシーバー チャネルに完全に含まれています。TX ファブリック インターフェースから開始して、PCS を通過し、GTX トランシーバー チャネルの PMA 側を通過せずに RX ファブリックに戻ります。
- [Near-End PMA] : 回路は近端 GTX トランシーバー チャネルに完全に含まれています。TX ファブリックから開始して、PCS、PMA、PCS を通過し、RX ファブリック インターフェイスに戻ります。
- [Far-End PMA] : 回路は、外部テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTX トランシーバー チャネルを通過して元に戻ります。この GTX トランシーバー ループバック モードでは、信号が RX ピンに入力され、PMA 回路を通過して TX ピンに戻ります。
- [Far-End PCS] : 回路は、外部テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTX トランシーバー チャネルを通過して元に戻ります。この GTX ループバック モードでは、信号が RX ピンに入力され、PMA、PCS、PMA を通過して TX ピンに戻ります。
- [Far-End Fabric] : 回路は、外部テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTP/GTX トランシーバー チャネルおよび関連するファブリック ロジックを通過して元に戻ります。この GTX トランシーバー ループバック モードでは、信号が RX ピンに入力され、PMA、PCS、ワード数が少ないファブリック ベースの FIFO、PCS、PMA を通過して TX ピンに戻ります。

[Channel Reset] をクリックすると、内部 PMA および PCS 回路に加え関連するファブリック インターフェイスがクリア、リセットされ、GTX トランシーバー チャネルがリセットされます。

[TX Polarity Invert] では、GTX トランシーバー チャネルの TX ピンから送信されたデータの極性を制御します。GTX トランシーバーの TX 側の極性を反転するには、このチェック ボックスをオンにします。

[TX Bit Error Inject] をクリックすると、1 つの送信ワードに含まれる 1 ビットの極性が反転されます。このトランスミッターに接続されているチャネルのレシーバー エンドポイントでは、1 ビット エラーが検出されるはずです。

[TX Diff Output Swing] では、トランスミッターの差動振幅を制御します。値を変更する場合はボックスに入力します。

[TX Pre-Emphasis] では、送信信号のプリエンファシス量を制御します。値を変更する場合はボックスに入力します。

[RX Polarity Invert] では、GTX チャネルの RX ピンから受信したデータの極性を制御します。GTX トランシーバーの RX 側の極性を反転するには、チェック ボックスをオンにします。

[RX AC Coupling Enabled] では、ビルトイン AC カップリング キャパシタをイネーブルにするかどうかを制御します。

[RX Termination Voltage] では、RX 終端ネットワークで使用する電源を制御します。

[RX Equalization] では、内部 RX イコライゼーション回路を制御します。

[BERT Settings]

[TX/RX Data Pattern] は、トランスミッターのパターン ジェネレーターおよびレシーバのパターン チェッカーで使用されるデータ パターンを選択するのに使用します。これらのパターンには、PRBS7、15、23、31、および Clk 2x、10x が含まれています。

[RX Bit Error Ratio] には、GTX トランシーバー チャネルに対して算出されたビット エラー率が表示されます。この値は指数で表現されます。たとえば、1.000E-12 は 1 兆ビット受信するたびに 1 ビット エラーが発生することを意味します。

[RX Received Bit Count] には、受信したビット数の総計が表示されます。このカウントは、[BERT Reset] をクリックしたときにリセットされます。

[RX Bit Error Count] には、検出されたビット エラー数の総計が表示されます。このカウントは、[BERT Reset] をクリックしたときにリセットされます。

このボタンをクリックすると、ビット エラー カウンターおよび受信ビット カウンターがリセットされます。GTX チャネルがリンクされ安定したら BERT カウンターをリセットしてください。

[Clocking Settings]

[TXOUTCLK Freq (MHz)] には、GTX トランシーバーの TXOUTCLK0 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[TXUSRCLK Freq (MHz)] には、GTX トランシーバーの TXUSRCLK ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[TXUSRCLK2 Freq (MHz)] には、GTX トランシーバーの TXUSRCLK2 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXUSRCLK Freq (MHz)] には、GTX トランシーバーの RXUSRCLK ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXUSRCLK2 Freq (MHz)] には、GTX トランシーバーの RXUSRCLK2 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[DRP Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTX トランシーバーが表示されます。各行では、特定の DRP 属性またアドレスが表示されます。

パネル下部の [View By Attribute Name] をオンにすると、すべての DRP 属性がアルファベット順で表示されます。[Radix] では [Hex] (16 進数) または [Bin] (2 進数) のいずれかを選択できます。値を変更するには、テキスト フィールドをクリックして新しい値を入力してから、Enter キーを押します。新しい値が MGT トランシーバーに反映されます。

パネル下部の [View By Address] がオンになっていると、行アドレスが数字順に表示されます。[Radix] では [Hex] (16 進数) または [Bin] (2 進数) のいずれかを選択できます。値を変更するには、

テキスト フィールドをクリックして新しい値を入力してから、Enter キーを押します。新しい値が MGT トランシーバーに反映されます。

[Port Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTX トランシーバーが表示されます。各行には、特定の MGT ポートが表示されます。一部のポートは IBERT デザインでデータの送受信に使用されるため、すべてのポートが表示されるわけではありません。

[Radix] では、値の基数に [Hex] (16 進数) または [Bin] (2 進数) を選択できます。一部のポートは読み取り専用で編集できません。これらのセルは、ラベルのように表示されます。編集可能な表内のポートは、テキスト フィールドのように表示され、カーソルを置くと新しい値が入力でき、Enter キーを押すとその値がすぐに反映されます。

[Sweep Test Settings] パネル

このパネルは、さまざまなトランシーバ設定をスイープするチャネル テストを設定するのに使用します。TX および RX 設定は、同じ GTX トランシーバー向けです。TX および RX の両方の設定のスイープは、トランシーバーが近端または外部ループバック モードのいずれかに設定されているときのみ機能します。RX パラメーターのスイープは、リンクの対応する TX エンドポイントが別のデバイスまたは同じデバイスに含まれる別のトランシーバーに含まれるときのみ実行できます。

このパネルは、[Parameter Settings]、[Sampling Point Region]、[Test Controls]、および [Test Results] セクションの 4 つのセクションから構成されています。

[Parameter Settings]

このセクションでは、スイープ パラメーターを設定します。

次に、スイープに使用できる GTX トランシーバーのパラメーターを示します。

- [TX Diff Swing]
- [TX Pre-Emphasis]
- [TX Post-Emphasis]
- [RX EQ]

これらのスイープ パラメーターは、次のいずれかの方法で初期化できます。

- [Clear All Parameters] をクリックしてすべてのパラメーターをクリアします。
- [Set Parameters to Current Values] をクリックして、[MGT/BERT Settings] パネルの現在の値にパラメータを設定します。

[Sweet Parameter] 表のパラメーターの順序は、パラメーターのスイープ順序を示しています。表上部のパラメーターの値は、表下部のパラメーターに比べスイープされる頻度が低くなります。つまり、表上部のパラメーターはスイープ アルゴリズムの外側ループにあり、表下部のパラメーターはスイープ アルゴリズムの内側ループにあるということです。

各パラメーターで開始値と終了値を設定する必要があります。パラメーターの順序は変更できません。開始値を選択すると、その値に対して有効な終了値が自動的に表示されます。パラメーターでスイープの設定を希望しない場合は、開始値と終了値を同じ値にします。[Sweep Value Count] 列では、特定のパラメーターでスイープされる値の数が示されます。すべてのスイープ パラメーターに有効な開始値と終了値が設定されると、スイープ実行回数が [Total Iterations] に表示されます。

パラメーターを表に追加または削除するには、[Add/Remove Parameters] をクリックします。ダイアログ ボックスが表示され、左側には使用可能なポート/属性すべて、右側にはスイープするパラメーターが表示されます。スイープするパラメーターを追加するには、左側のリストからいずれかをクリックして、右方向 (>) ボタンをクリックします。スイープするパラメーターを削除するには、右側のリストからいずれかをクリックして、左方向 (<) ボタンをクリックします。パラメーターのスイープ順を指定するには、右側のリストのいずれかをクリックしてから [Up] または [Down] をクリックします。スイープ属性およびその順序をデフォルトの設定に戻す場合は、[Reset to Default] をクリックします。[OK] をクリックして設定を適用するか、[Cancel] をクリックして保存せずに終了します。

[Sampling Point Region]

サンプリング ポイントは、アイとサンプル間の水平ポイントです。左端の遷移領域を視覚化してから、右端を視覚化します。[RX Sampling Point] は、128 個の不連続のサンプリング位置の 1 つです。このセクションでは、サンプリング領域の左端および右端を選択します。

[Test Controls]

スイープ テストを設定したら、[Start] をクリックしてテストを開始します。クリックした後は、スイープ パラメーター表がディスエーブルになりテストが実行されます。

スイープ テストの実行の過程で現在のスイープ結果ファイル、現在の実行回数、経過時間、および概算残り時間ステータスが表示されます。スイープ結果は、画面下のテキスト エリアに表示されます。スイープ テストは [Pause] をクリックして一時停止したり、[Reset] をクリックして完全に停止することができます。

スイープテストの結果は、[Test Results] パネルに表示されます。またスイープ テスト結果ファイルに書き出すこともできます。[Log File Settings] をクリックすると、ダイアログ ボックスが表示されます。

このダイアログ ボックスでは、ファイルの保存場所と各ファイルに含めるスイープ実行回数を設定できます。スイープ実行回数がファイルの制限を越える場合は、最初の結果ファイルと同じディレクトリにベース ファイル名の開始実行番号が付けられた複数のファイルが作成されます。

[Test Results]

このパネルには、現在の実行、経過時間、および残り時間の概算が表示されます。このステータス情報の下には、スイープ テスト結果を示す次のタブが表示されます。

- [Sweep Test Log]
- [Sweep Test Plots]
- [Sweep Test Info]

[Sweep Test Log]

[Sweep Test Log] は常に表示されるタブで、スイープ テスト結果の実行ログが含まれます。このタブの情報は、テキストでのみ表示されます。スイープ テストの結果は、CSV ログフ ァ イルにも含まれます。

[Sweep Test Plots]

[Sweep Test Plots] はスイープ テストの実行が停止した後にのみ表示されるタブです。スイープ テストプロットのウィンドウに表示されるグラフィック データは、CSV ログ ファイルに保存されたデータと同じです。プロットはそれぞれ受信信号のユニット インターバル (UI) のスイープに該当します。データ プロットの左端と右端は、[Sample Point Region] パネルの設定に対応します。

スイープ テスト プロットは、一番低いビット エラー率 (BER) でのアクティブ プロットの UI の開口幅で主に計測されます。水平マーカーはクリックして上下に、垂直マーカーは左右にドラッグできます。[Sweep Test Plots] タブの右側のプロット リスト エリアを右クリックすると、各プロットの表示/非表示を切り替えたり、名前や色を付け替えたり、アクティブに設定したりできます。

[Sweep Test Info]

[Sweep Test Info] もスイープ テストの実行が停止した後にのみ表示されるタブです。[Sweep Test Info] の表形式のデータは、[Sweep Test Plots] のグラフィカルなデータに対応します。表の各行が 1 つのスイープ テスト プロットに該当します。表の列では、それぞれ次を示します。

- [Enable Plot] : [Sweep Test Plots] パネルのプロットの表示/非表示を切り替えます。オンにすると表示、オフにすると非表示になります。
- [Line Color] : [Sweep Test Plots] パネルで使用される線の色を指定します。クリックすると、新しい色を選択できます。
- [Plot Name] : プロットの名前を指定します。テキスト フィールドをクリックすると、プロット名を変更できます。
- [Opening at the Lowest BER Level] : 一番低い BER レベルでエラーなく実行される最長幅が示されます。
- 残りの列には、スイープ テスト プロットを作成するために使用されたパラメーター値が表示されます。

列ヘッダーをクリックすると、パネルの行を並び替えることができます。

注記：[Opening at the Lowest BER Level] 列ヘッダーをクリックし、プロットを UI 開口幅の広い順に並び替えることをお勧めします。

スタンドアロンの IBERT スイープ テスト プロット ビューアー

IBERT コンソール ウィンドウの [Sweep Test] パネルのプロット ビューアーは、テスト デバイスに接続されている場合のスイープテスト結果にのみ使用できます。スイープ テスト結果をオフラインで表示する場合は、スタンドアロンの IBERT スイープ テストプロット ビューアーを使用できます。

- Windows (32 ビット) プラットフォーム : `<install dir>\bin\nt\ibertplotter.exe`
- Windows (64 ビット) プラットフォーム : `<install dir>\bin\nt64\ibertplotter.exe`
- Linux (32 ビット) プラットフォーム : `<install dir>/bin/lin/ibertplotter`
- Linux (64 ビット) プラットフォーム : `<install dir>/bin/lin64/ibertplotter`

スタンドアロンの IBERT スイープ テスト プロット ビューアーでは、次のトランシーバーで IBERT スイープ テストを実行して作成された CSV 形式のスイープ テスト結果ファイルを表示できます。

- Spartan®-6 FPGA GTP トランシーバー
- Virtex®-5 FPGA GTX トランシーバー
- Virtex-6 FPGA GTX トランシーバー
- Virtex-6 FPGA GTH トランシーバー

IBERT スイープ テスト プロット ビューアーでは、複数のスイープ テスト結果ファイル (CSV) を同時に表示することもできるので、さまざまなスイープ テスト結果を比較しやすくなっています。

IBERT Virtex-6 FPGA GTX トランシーバーのツールバーおよびメニュー オプション

[IBERT Console Options]

[IBERT Console Options] ダイアログ ボックスでは、コンソール ウィンドウに表示する列および行を選択できます。左側パネルでは、MGT をロケーションで選択します。すべて選択する場合は [Check All]、すべて選択しない場合は [Uncheck All] をクリックします。

右側パネルでは、[MGT/BERT Settings] パネルに表示される行を選択します。すべて選択する場合は [Check All]、すべて選択しない場合は [Uncheck All] をクリックします。[Default] をクリックすると、チャネルの有効性を決定するのに必要な行の基本セットがコンソール ウィンドウに表示されます。[Default] で表示される行は、[Tile Location]、[TX PLL Status]、[RX PLL Status]、[Loopback Mode]、[Channel Reset]、[TX Error Inject]、[Rx Sampling Point]、[TX Data Pattern]、[RX Data pattern]、[Rx Bit Error Ratio]、[Rx Received Bit Count]、[Rx Bit Error Count]、[BERT Reset]、[TXUSRCLK2 Freq]、および [RXUSRCLK2 Freq] です。

[Import/Export] ダイアログ ボックス

このダイアログ ボックスでは、特定の MGT の設定を保存して復元したり、デザインの別の MGT に適用できます。設定をインポートまたはエクスポートするには、[IBERT_V6GTX] → [Import/Export Wizard] をクリックするか、またはツールバーの [Import/Export Wizard] をクリックします。

ウィザードの最初の画面では、MGT 設定のソースに [MGT] または [File] のいずれかを選択します。[MGT] を選択した場合は、コンボ ボックスに表示される MGT から選択します。[File] を選択した場合は、[Browse] をクリックして設定ファイルを指定します。[Next] をクリックして、次の画面に進みます。

次の画面では、デスティネーションを指定します。IBERT デザインに含まれる MGT とファイルを自由に組み合わせることができます。[File] をイネーブルした場合は、[Browse] をクリックしてファイル デスティネーションを指定します。

3 番目の画面には、設定のソースとデスティネーションのサマリが表示されます。[Apply] をクリックしてインポートまたはエクスポートします。この操作は、元に戻すことができません。

[Reset All]

IBERT コアのすべてのチャネルをリセットするには、[IBERT_V6GTX] → [Reset All] をクリックするか、またはツールバーの [Reset All] をクリックします。

[JTAG Scan Rate] および [Scan Now]

ツールバーの [JTAG Scan Rate] および [IBERT_V6GTX] → [JTAG Scan Rate] メニューをクリックすると、ChipScope Pro Analyzer ツールで IBERT コアのステータス情報を確認する頻度を選択できます。デフォルトは [1 s] ですが、[250 ms]、[500 ms]、[2 s]、[5 s]、または [Manual Scan] も選択できます。[Manual Scan] を選択した場合、[IBERT_V6GTX] → [Scan Now] またはツールバーの [Scan Now] ([S!]) をクリックすると、IBERT コアのクエリをすぐに実行できます。

Virtex-6 FPGA GTH トランシーバー用 IBERT コンソール ウィンドウ

Virtex-6 HXT ファミリ GTH トランシーバー用の IBERT コアのコンソールを開くには、[Window] → [New Unit Windows] でコアを選択します。このコア ユニット向けのダイアログ ボックスが表示されるので [IBERT V6 GTH Console] を選択します。このダイアログ ボックスからはウィンドウを閉じることはできません。

プロジェクト ツリーの [IBERT V6 GTH Console] をダブルクリックするか、または右クリックして [Open IBERT V6 GTH Console] をクリックしてもコンソール ウィンドウを表示できます。

Virtex-6 HXT ファミリの GTH トランシーバー用 IBERT コンソール ウィンドウは、「[MGT/BERT Settings] パネル」、99 ページの「[DRP Settings] パネル」、100 ページの「[Port Settings] パネル」、および 103 ページの「IBERT Virtex-6 FPGA GTX トランシーバーのツールバーおよびメニュー オプション」 で構成されています。

[MGT/BERT Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTH トランシーバーが表示されます。各行では特定の制御またはステータス設定が表示されます。

[MGT Settings]

[MGT Alias] は、GTH トランシーバーの MGT 番号に初期設定されていますが、新しい値を入力できます。

[Tile Location] には、デバイスに含まれる GTH トランシーバーの X/Y 座標が表示されます。

[MGT Link Status] には、特定の GTH トランシーバー チャネルのレシーバーに接続されているリンク検出ロジックのステータスが表示されます。チャネルがリンクされている場合は緑色で計測されたライン レートが表示され、リンクされていない場合は「NOT LOCKED」(赤色) と表示されます。

[PLL Status] には GTH QUAD に含まれている PLL のロック ステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[Loopback Mode] では、特定の GTH トランシーバー チャネルのループバック モードを制御します。次にループバック モードの選択肢を示します。

- [None] : フィードバック パスは使用されません。
- [Near-End PCS] : 回路は近端 GTH トランシーバー チャネルに完全に含まれています。TX ファブリック インターフェースから開始して、PCS を通過し、GTH トランシーバー チャネルの PMA 側を通過せずに RX ファブリックに戻ります。
- [Far-End PCS] : 回路は、外部テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTH トランシーバー チャネルを通過して元に戻ります。この GTH ループバック モードでは、信号が RX ピンに入力され、PMA、PCS、PMA を通過して TX ピンに戻ります。

[QUAD Reset] をクリックすると、内部 PMA および PCS 回路に加え関連するファブリック インターフェイスがクリア、リセットされ、GTH QUAD (トランシーバー 4 個) がリセットされます。

[TX Bit Error Inject] をクリックすると、1 つの送信ワードに含まれる 1 ビットの極性が反転されます。このトランスミッターに接続されているチャネルのレシーバー エンドポイントでは、1 ビット エラーが検出されるはずです。

[TX Diff Output Swing] では、トランスミッターの差動振幅を制御します。値を変更する場合はボックスに入力します。

[TX Pre-Emphasis] では、送信信号のプリエンファシス量を制御します。値を変更する場合はボックスに入力します。

[TX Post-Emphasis] では、送信信号のポストエンファシス量を制御します。値を変更する場合はボックスに入力します。

[RX Equalization] では、内部 RX イコライゼーション回路を制御します。

[BERT Settings]

[TX/RX Data Pattern] は、トランスミッターのパターン ジェネレーターおよびレシーバのパターン チェッカーで使用されるデータ パターンを選択するのに使用します。これらのパターンには、PRBS7、15、23、31、および Clk 2x、10x が含まれています。

[RX Bit Error Ratio] には、GTH トランシーバー チャネルに対して算出されたビット エラー率が表示されます。この値は指数で表現されます。たとえば、1.000E-12 は 1 兆ビット受信するたびに 1 ビット エラーが発生することを意味します。

[RX Received Bit Count] には、受信したビット数の総計が表示されます。このカウントは、[BERT Reset] をクリックしたときにリセットされます。

[RX Bit Error Count] には、検出されたビット エラー数の総計が表示されます。このカウントは、[BERT Reset] をクリックしたときにリセットされます。

このボタンをクリックすると、ビット エラー カウンターおよび受信ビット カウンターがリセットされます。GTH チャネルがリンクされ安定した後に BERT カウンターをリセットしてください。

[Clocking Settings]

[TXUSERCLKOUT Freq (MHz)] には、GTH トランシーバーの TXUSERCLKOUT ポートのクロック周波数の概算値が MHz で表示されます。この概算値の精度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXUSERCLKOUT Freq (MHz)] には、GTH トランシーバーの RXUSERCLKOUT ポートのクロック周波数の概算値が MHz で表示されます。この概算値の精度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[DRP Settings] パネル

このパネルでは、DRP 属性またはアドレスを示す表が表示されます。各列には、特定のアクティブな GTH QUAD が表示されます。各行では、特定の DRP 属性またアドレスが表示されます。

パネル下部の [View By Attribute Name] をオンにすると、すべての DRP 属性がアルファベット順で表示されます。[Radix] では [Hex] (16 進数) または [Bin] (2 進数) のいずれかを選択できます。値を変更するには、テキスト フィールドをクリックして新しい値を入力してから、Enter キーを押します。新しい値が GTH トランシーバーに反映されます。

パネル下部の [View By Address] がオンになっていると、行アドレスが数字順に表示されます。[Radix] では [Hex] (16 進数) または [Bin] (2 進数) のいずれかを選択できます。値を変更するには、テキスト フィールドをクリックして新しい値を入力してから、Enter キーを押します。新しい値が GTH トランシーバーに反映されます。

[Port Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTH QUAD が表示されます。各行には、特定のトランシーバー ポートが表示されます。一部のポートは IBERT デザインでデータの送受信に使用されるため、すべてのポートが表示されるわけではありません。

[Radix] では、値の基数に [Hex] (16 進数) または [Bin] (2 進数) を選択できます。一部のポートは読み取り専用で編集できません。これらのセルは、ラベルのように表示されます。編集可能な表内のポートは、テキスト フィールドのように表示されます。値を変更する場合は、これらのフィールドにカーソルを置いて新しい値を入力し、Enter キーを押します。新しい値が GTH トランシーバーに反映されます。

IBERT Virtex-6 FPGA GTH トランシーバーのツールバーおよびメニュー オプション

[IBERT Console Options]

[IBERT Console Options] ダイアログ ボックスでは、コンソール ウィンドウに表示する列および行を選択できます。左側パネルでは、トランシーバをロケーションで選択します。すべて選択する場合は [Check All]、すべて選択しない場合は [Uncheck All] をクリックします。

右側パネルでは、[MGT/BERT Settings] パネルに表示される行を選択します。すべて選択する場合は [Check All]、すべて選択しない場合は [Uncheck All] をクリックします。[Default] をクリックすると、チャネルの有効性を決定するのに必要な行の基本セットがコンソール ウィンドウに表示されます。[Default] で表示される行は、[Tile Location]、[TX PLL Status]、[RX PLL Status]、[Loopback Mode]、[Channel Reset]、[TX Error Inject]、[Rx Sampling Point]、[TX Data Pattern]、[RX Data pattern]、[Rx Bit Error Ratio]、[Rx Received Bit Count]、[Rx Bit Error Count]、[BERT Reset]、[TXUSRCLK2 Freq]、および [RXUSRCLK2 Freq] です。

[Import/Export] ダイアログ ボックス

このダイアログ ボックスでは、特定の GTH トランシーバーの設定を保存して復元したり、デザインの別のトランシーバーに適用できます。設定をインポートまたはエクスポートするには、[IBERT_V6GTH] → [Import/Export Wizard] をクリックするか、またはツールバーの [Import/Export Wizard] をクリックします。

ウィザードの最初の画面では、トランシーバー設定のソースに [MGT] または [File] のいずれかを選択します。[MGT] を選択した場合は、コンボ ボックスに表示されるトランシーバーから選択します。[File] を選択した場合は、[Browse] をクリックして設定ファイルを指定します。[Next] をクリックして、次の画面に進みます。

次の画面では、デスティネーションを指定します。IBERT デザインに含まれる GTH トランシーバーとファイルを自由に組み合わせることができます。[File] をイネーブルした場合は、[Browse] をクリックしてファイル デスティネーションを指定します。

3 番目の画面には、設定のソースとデスティネーションのサマリが表示されます。[Apply] をクリックしてインポートまたはエクスポートします。この操作は、元に戻すことができません。

[Reset All]

IBERT コアのすべてのチャネルをリセットするには、[IBERT_V6GTH] → [Reset All] をクリックするか、またはツールバーの [Reset All] をクリックします。

[JTAG Scan Rate] および [Scan Now]

ツールバーの [JTAG Scan Rate] および [IBERT_V6GTH] → [JTAG Scan Rate] メニューをクリックすると、ChipScope Pro Analyzer ツールで IBERT コアのステータス情報を確認する頻度を選択できます。デフォルトは [1 s] ですが、[250 ms]、[500 ms]、[2 s]、[5 s]、または [Manual Scan] も選択できます。[Manual Scan] を選択した場合、[IBERT_V6GTH] → [Scan Now] またはツールバーの [Scan Now] ([S!]) をクリックすると、IBERT コアのクエリを実行できます。

Spartan-6 FPGA GTP トランシーバー用 IBERT コンソール ウィンドウ

Spartan-6 LXT デバイスの GTP トランシーバー用の IBERT コアのコンソールを開くには、[Window] → [New Unit] → [Windows] でコアを選択します。このコア ユニット向けのダイアログ ボックスが表示されるので [IBERT S6 GTP Console] を選択します。このダイアログ ボックスからはウィンドウを閉じることはできません。

プロジェクト ツリーの [IBERT S6 GTP Console] をダブルクリックするか、または右クリックして [Open IBERT S6 GTP Console] をクリックしてもコンソール ウィンドウを表示できます。

Spartan-6 LXT プラットフォームの IBERT コンソール ウィンドウは、[MGT/BERT Settings] パネル、[DRP Settings] パネル、および [Port Settings] パネルで構成されています。

[MGT/BERT Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTP トランシーバーが表示されます。各行では特定の制御またはステータス設定が表示されます。

[MGT Settings]

[MGT Alias] は、GTP トランシーバー MGT 番号に初期設定されていますが、新しい値を入力できます。

[Tile Location] には、デバイスに含まれる GTP トランシーバーの X/Y 座標が表示されます。

[MGT Link Status] には、特定の GTP トランシーバー チャネルのレシーバに接続されているリンク検出ロジックのステータスが表示されます。チャネルがリンクされている場合は緑色で計測されたライン レートが表示され、リンクされていない場合は「NOT LOCKED」(赤色) と表示されます。

[PLL Status] では GTP トランシーバーに接続されている PLL のロック ステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[Loopback Mode] は、特定の GTP トランシーバー チャネルのループバック モードを制御します。次にループバック モードの選択肢を示します。

- [None] :フィードバック パスは使用されません。
- [Near-End PCS] : 回路は近端 GTP トランシーバー チャネルに完全に含まれています。TX ファブリック インターフェースから開始して、PCS を通過し、GTP トランシーバー チャネルの PMA サイドを通過せずに RX ファブリックに戻ります。
- [Near-End PMA] : 回路は近端 GTP トランシーバー チャネルに完全に含まれています。TX ファブリックから開始して、PCS、PMA、PCS を通過し、RX ファブリック インターフェイスに戻ります。
- [Far-End PMA] : 回路は、テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTP トランシーバー チャネルを通過して元に戻ります。この GTP ループバック モードでは、信号が RX ピンに入力され、PMA 回路を通過して TX ピンに戻ります。
- [Far-End PCS] : 回路は、テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTP トランシーバー チャネルを通過して元に戻ります。この GTP ループバック モードでは、信号が RX ピンに入力され、PMA、PCS、PMA を通過して TX ピンに戻ります。
- [Far-End Fabric] : 回路は、テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTP トランシーバー チャネルおよび関連するファブリック ロジックを通過して元に戻ります。この GTP ループバック モードでは、信号が RX ピンに入力され、PMA、PCS、ワード数が少ないファブリック ベースの FIFO、PCS、PMA を通過して TX ピンに戻ります。

[Channel Reset] ボタンをクリックすると、内部 PMA および PCS 回路に加え関連するファブリック インターフェイスがクリア、リセットされ、GTP トランシーバー チャネルがリセットされます。

[TX Polarity Invert] では、GTP チャネルの TX ピンから送信されたデータの極性を制御します。GTP トランシーバーの TX 側の極性を反転するには、このチェック ボックスをオンにします。

[TX Bit Error Inject] をクリックすると、1 つの送信ワードに含まれる 1 ビットの極性が反転されます。このトランスミッターに接続されているチャネルのレシーバー エンドポイントでは、1 ビット エラーが検出されるはずです。

[TX Diff Output Swing] では、トランスミッターの差動振幅を制御します。値を変更する場合はボックスに入力します。

[TX Pre-Emphasis] では、送信信号のプリエンファシス量を制御します。値を変更する場合はボックスに入力します。

[RX Polarity Invert] では、GTP チャネルの RX ピンから受信したデータの極性を制御します。GTP トランシーバーの RX 側の極性を反転するには、このチェック ボックスをオンにします。

[RX AC Coupling Enabled] では、ビルトイン AC カップリング キャパシタをイネーブルにするかどうかを制御します。

[RX Termination Voltage] では、RX 終端ネットワークで使用する電源を制御します。

[RX Equalization] では、内部 RX イコライゼーション回路を制御します。

[BERT Settings]

[TX/RX Data Pattern] は、トランスミッターのパターン ジェネレーターおよびレシーバのパターン チェッカーで使用されるデータ パターンを選択するのに使用します。これらのパターンには、PRBS7、15、23、31、および Clk 2x、10x が含まれています。

[RX Bit Error Ratio] フィールドには、GTP トランシーバー チャネルに対して算出されたビット エラー率が含まれています。この値は指数で表現されます。たとえば、1.000E-12 は 1 兆ビット受信するたびに 1 ビット エラーが発生することを意味します。

[RX Received Bit Count] には、受信したビット数の総計が表示されます。このカウントは、[BERT Reset] をクリックしたときにリセットされます。

[RX Bit Error Count] には、検出されたビット エラー数の総計が表示されます。このカウントは、[BERT Reset] をクリックしたときにリセットされます。

このボタンをクリックすると、ビット エラー カウンターおよび受信ビット カウンターがリセットされます。GTP トランシーバー チャネルがリンクされ安定した後に BERT カウンタをリセットしてください。

[Clocking Settings]

[TXUSRCLK Freq (MHz)] には、GTP トランシーバーの TXUSRCLK ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[TXUSRCLK2 Freq (MHz)] には、GTP トランシーバーの TXUSRCLK2 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXUSRCLK Freq (MHz)] には、GTP トランシーバーの RXUSRCLK ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[RXUSRCLK2 Freq (MHz)] には、GTP トランシーバーの RXUSRCLK2 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[DRP Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTP トランシーバーが表示されます。各行では、特定の DRP 属性またはアドレスが表示されます。

パネル下部の [View By Attribute Name] をオンにすると、すべての DRP 属性がアルファベット順で表示されます。[Radix] では [Hex] (16 進数) または [Bin] (2 進数) のいずれかを選択できます。値を変更するには、テキスト フィールドをクリックして新しい値を入力してから、Enter キーを押します。新しい値が MGT トランシーバーに反映されます。

パネル下部の [View By Address] がオンになっていると、行アドレスが数字順に表示されます。[Radix] では [Hex] (16 進数) または [Bin] (2 進数) のいずれかを選択できます。値を変更するには、テキスト フィールドをクリックして新しい値を入力してから、Enter キーを押します。新しい値が MGT トランシーバーに反映されます。

[Port Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTP トランシーバーが表示されます。各行には、特定の MGT ポートが表示されます。一部のポートは IBERT デザインでデータの送受信に使用されるため、すべてのポートが表示されるわけではありません。

[Radix] では、値の基数に [Hex] (16 進数) または [Bin] (2 進数) を選択できます。一部のポートは読み取り専用で編集できません。これらのセルは、ラベルのように表示されます。編集可能な表内のポートは、テキスト フィールドのように表示され、カーソルを置くと新しい値が入力でき、Enter キーを押すとその値がすぐに反映されます。

IBERT S6 GTP のツールバーおよびメニュー オプション

[IBERT Console Options]

[IBERT Console Options] ダイアログ ボックスでは、コンソール ウィンドウに表示する列および行を選択できます。左側パネルでは、MGT をロケーションで選択します。すべて選択する場合は [Check All]、すべて選択しない場合は [Uncheck All] をクリックします。

右側パネルでは、[MGT/BERT Settings] パネルに表示される行を選択します。すべて選択する場合は [Check All]、すべて選択しない場合は [Uncheck All] をクリックします。[Default] をクリックすると、チャネルの有効性を決定するのに必要な行の基本セットがコンソール ウィンドウに表示されます。[Default] で表示される行は、[Tile Location]、[TX PLL Status]、[RX PLL Status]、[Loopback Mode]、[Channel Reset]、[TX Error Inject]、[Rx Sampling Point]、[TX Data Pattern]、[RX Data pattern]、[Rx Bit Error Ratio]、[Rx Received Bit Count]、[Rx Bit Error Count]、[BERT Reset]、[TXUSRCLK2 Freq]、および [RXUSRCLK2 Freq] です。

[Import/Export] ダイアログ ボックス

このダイアログ ボックスでは、特定の MGT の設定を保存して復元したり、デザインの別の MGT に適用できます。設定をインポートまたはエクスポートするには、[IBERT_S6GTP] → [Import/Export Wizard] をクリックするか、またはツールバーの [Import/Export Wizard] をクリックします。

ウィザードの最初の画面では、MGT 設定のソースに [MGT] または [File] のいずれかを選択します。[MGT] を選択した場合は、コンボ ボックスに表示される MGT から選択します。[File] を選択した場合は、[Browse] をクリックして設定ファイルを指定します。[Next] をクリックして、次の画面に進みます。

次の画面では、デスティネーションを指定します。IBERT デザインに含まれる MGT とファイルを自由に組み合わせることができます。[File] をイネーブルした場合は、[Browse] をクリックしてファイル デスティネーションを指定します。

3 番目の画面には、設定のソースとデスティネーションのサマリが表示されます。[Apply] をクリックしてインポートまたはエクスポートします。この操作は、元に戻すことができません。

[Reset All]

IBERT コアのすべてのチャネルをリセットするには、[IBERT_S6GTP] → [Reset All] をクリックするか、またはツールバーの [Reset All] をクリックします。

[JTAG Scan Rate] および [Scan Now]

ツールバーの [JTAG Scan Rate] および [IBERT_S6GTP] → [JTAG Scan Rate] メニューをクリックすると、ChipScope Pro Analyzer ツールで IBERT コアのステータス情報を確認する頻度を選択できます。デフォルトは [1 s] ですが、[250 ms]、[500 ms]、[2 s]、[5 s]、または [Manual Scan] も選択できます。[Manual Scan] を選択した場合、[IBERT_S6GTP] → [Scan Now] またはツールバーの [Scan Now] ([S!]) をクリックすると、IBERT コアのクエリを実行できます。

Kintex-7 FPGA GTX トランシーバー用 IBERT コンソール ウィンドウ

Kintex-7 ファミリ GTX トランシーバー用の IBERT コアのコンソールを開くには、[Window] → [New Unit Windows] でコアを選択します。このコア ユニット向けのダイアログ ボックスが表示されるので [IBERT K7 GTX Console] を選択します。このダイアログ ボックスからはウィンドウを閉じることはできません。

プロジェクト ツリーの [IBERT K7 GTX Console] をダブルクリックするか、または右クリックして [Open IBERT K7 GTX Console] をクリックしてもコンソール ウィンドウを表示できます。

Kintex-7 GTX トランシーバー用 IBERT コンソール ウィンドウは、次で構成されています。

- [MGT/BERT Settings] パネル
- [DRP Settings] パネル
- [Port Settings] パネル
- IBERT Kintex-7 FPGA GTX トランシーバーのツールバーおよびメニュー オプション

[MGT/BERT Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTX トランシーバーが表示されます。各行では特定の制御またはステータス設定が表示されます。

[MGT Settings]

[MGT Alias] は、GTX トランシーバー MGT 番号に初期設定されていますが、新しい値を入力できます。

[Tile Location] には、デバイスに含まれる GTX トランシーバーの X/Y 座標が表示されます。[MGT Link Status] には、特定の GTX トランシーバー チャネルのレシーバーに接続されているリンク検出ロジックのステータスが表示されます。チャネルがリンクされている場合は緑色で計測されたライン レートが表示され、リンクされていない場合は「NOT LOCKED」(赤色) と表示されます。

[CPLL/QPLL Status] には GTX トランシーバーに接続されている CPLL/QPLL のロック ステータスが表示されます。このステータス インジケーターの有効なステートは、LOCKED (緑色) または NOT LOCKED (赤色) です。

[Loopback Mode] 設定は、特定の GTX トランシーバー チャネルのループバック モードを制御するのに使用します。次にループバック モードの選択肢を示します。

- フィードバック パスは使用されません。
- [Near-End PCS] : 回路は近端 GTX トランシーバー チャネルに完全に含まれています。TX ファブリック インターフェースから開始して、PCS を通過し、GTX トランシーバー チャネルの PMA 側を通過せずに RX ファブリックに戻ります。
- [Near-End PMA] : 回路は近端 GTX トランシーバー チャネルに完全に含まれています。TX ファブリックから開始して、PCS、PMA、PCS を通過し、RX ファブリック インターフェイスに戻ります。
- [Far-End PMA] : 回路は、外部テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTX トランシーバー チャネルを通過して元に戻ります。この GTX トランシーバー ループバック モードでは、信号が RX ピンに入力され、PMA 回路を通過して TX ピンに戻ります。
- [Far-End PCS] : 回路は、外部テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTX トランシーバー チャネルを通過して元に戻ります。この GTX ループバック モードでは、信号が RX ピンに入力され、PMA、PCS、PMA を通過して TX ピンに戻ります。
- 回路は、外部テスト装置やほかのデバイスの一部などの外部チャネルのエンドポイントから始まり、GTX トランシーバー チャネルおよび関連するファブリック ロジックを通過して元に戻ります。この GTX トランシーバー ループバック モードでは、信号が RX ピンに入力され、PMA、PCS、ワード数が少ないファブリック ベースの FIFO、PCS、PMA を通過して TX ピンに戻ります。

[Channel Reset] をクリックすると、内部 PMA および PCS 回路に加え関連するファブリック インターフェイスがクリア、リセットされ、GTX トランシーバー チャネルがリセットされます。

[TX Polarity Invert] では、GTX トランシーバー チャネルの TX ピンから送信されたデータの極性を制御します。GTX トランシーバーの TX 側の極性を反転するには、このチェック ボックスをオンにします。

[TX Bit Error Inject] をクリックすると、1 つの送信ワードに含まれる 1 ビットの極性が反転されます。このトランスミッターに接続されているチャネルのレシーバー エンドポイントでは、1 ビット エラーが検出されるはずです。

[TX Diff Output Swing] では、トランスミッターの差動振幅を制御します。値を変更する場合はボックスに入力します。

[TX Pre-Emphasis] では、送信信号のプリエンファシス量を制御します。値を変更する場合はボックスに入力します。

[BERT Settings]

[TX/RX Data Pattern] は、トランスミッターのパターン ジェネレーターおよびレシーバーのパターン チェッカーで使用されるデータ パターンを選択するのに使用します。これらのパターンには、PRBS7、15、23、31、および Clk 2x、10x が含まれています。

[Clocking Settings]

[TXUSRCLK Freq (MHz)] には、GTX トランシーバーの TXUSRCLK ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[TXUSRCLK2 Freq (MHz)] には、GTX トランシーバーの TXUSRCLK2 ポートのクロック周波数の概算値が MHz で表示されます。このステータスの正確度は、コンパイル時に指定されたシステム クロックの周波数によって異なります。

[DRP Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTX トランシーバーが表示されます。各行では、特定の DRP 属性またはアドレスが表示されます。

パネル下部の [View By Attribute Name] をオンにすると、すべての DRP 属性がアルファベット順で表示されます。[Radix] では [Hex] (16 進数) または [Bin] (2 進数) のいずれかを選択できます。値を変更するには、テキスト フィールドをクリックして新しい値を入力してから、Enter キーを押します。新しい値が MGT トランシーバーに反映されます。

パネル下部の [View By Address] がオンになっていると、行アドレスが数字順に表示されます。[Radix] では [Hex] (16 進数) または [Bin] (2 進数) のいずれかを選択できます。値を変更するには、テキスト フィールドをクリックして新しい値を入力してから、Enter キーを押します。新しい値が MGT トランシーバーに反映されます。

[Port Settings] パネル

このパネルには、複数の列と行を含むテーブルが表示されます。各列では、特定のアクティブな GTX トランシーバーが表示されます。各行には、特定の MGT ポートが表示されます。一部のポートは IBERT デザインでデータの送受信に使用されるため、すべてのポートが表示されるわけではありません。

[Radix] では、値の基数に [Hex] (16 進数) または [Bin] (2 進数) を選択できます。一部のポートは読み取り専用で編集できません。これらのセルは、ラベルのように表示されます。編集可能な表内のポートは、テキスト フィールドのように表示され、カーソルを置くと新しい値が入力でき、Enter キーを押すとその値がすぐに反映されます。

IBERT Kintex-7 FPGA GTX トランシーバーのツールバーおよびメニュー オプション

[IBERT Console Options]

[IBERT Console Options] ダイアログ ボックスでは、コンソール ウィンドウに表示する列および行を選択できます。左側パネルでは、MGT をロケーションで選択します。すべて選択する場合は [Check All]、すべて選択しない場合は [Uncheck All] をクリックします。

右側パネルでは、[MGT/BERT Settings] パネルに表示される行を選択します。[Default] をクリックすると、チャネルの有効性を決定するのに必要な行の基本セットがコンソール ウィンドウに表示されます。デフォルトで表示される行は、[Tile Location]、[PLL Status]、[Loopback Mode]、[Channel Reset]、[TX Polarity Invert]、[TX Error Inject]、[TX Diff Output Swing]、[TX Pre-Cursor]、[TX Post-Cursor]、[TX Data Pattern]、[TXUSRCLK Freq]、および [TXUSRCLK2 Freq] です。

[Import/Export] ダイアログ ボックス

このダイアログ ボックスでは、特定の MGT の設定を保存して復元したり、デザインの別の MGT に適用できます。設定をインポートまたはエクスポートするには、[IBERT_K7GTX] → [Import/Export Wizard] をクリックするか、またはツールバーの [Import/Export Wizard] をクリックします。

ウィザードの最初の画面では、MGT 設定のソースに [MGT] または [File] のいずれかを選択します。[MGT] を選択した場合は、コンボ ボックスに表示される MGT から選択します。[File] を選択した場合は、[Browse] をクリックして設定ファイルを指定します。

次の画面では、デスティネーションを指定します。IBERT デザインに含まれる MGT とファイルを自由に組み合わせることができます。[File] をイネーブルした場合は、[Browse] をクリックしてファイル デスティネーションを指定します。

3 番目の画面には、設定のソースとデスティネーションのサマリが表示されます。[Apply] をクリックしてインポートまたはエクスポートします。この操作は、元に戻すことができません。

[Reset All]

IBERT コアのすべてのチャネルをリセットするには、[IBERT_K7GTX] → [Reset All] をクリックするか、またはツールバーの [Reset All] をクリックします。

[JTAG Scan Rate] および [Scan Now]

ツールバーの [JTAG Scan Rate] および [IBERT_K7GTX] → [JTAG Scan Rate] メニューをクリックすると、ChipScope Pro Analyzer ツールで IBERT コアのステータス情報を確認する頻度を選択できます。デフォルトは [1 s] ですが、[250 ms]、[500 ms]、[2 s]、[5 s]、または [Manual Scan] も選択できます。[Manual Scan] を選択した場合、[IBERT_K7GTX] → [Scan Now] をクリックするかツールバーの [Scan Now] ([S!]) をクリックして IBERT コアのクエリをすぐに実行できます。

ヘルプの表示

ヘルプ ページの表示

ChipScope Pro Analyzer のヘルプ ページには、現在開いているバージョンのツールとコア ユニットの情報のみが表示されます。[Help] → [About] をクリックすると、ツールのバージョンを確認できます。[Help] → [About: Cores] をクリックすると、検出されたコアそれぞれのパラメーターの詳細が表示されます。個々のコアのパラメーターを表示するには、プロジェクト ツリーのユニットで右クリックし、[Show Core Info] をクリックします。

ChipScope Pro ILA 波形ツールバー機能

ChipScope Pro Analyzer メニュー下のツールバーでは、メニュー オプションに加え、その他の波形コマンドを使用できます。セカンド セットのツールバーは、[Trigger Setup] ウィンドウの表示中にのみ使用できます。サード セット、フォース セットは、[Waveform] ウィンドウの使用中にのみ表示されます。

ツールバー ボタンは、次のメニュー オプションと同等です。

- [Open Cable/Search JTAG Chain] : ケーブルを自動的に検出し、JTAG チェーンの構成を確認します。
- [Turn On/Off Auto Core Status Polling] : 緑のアイコンはポーリングがオン、赤のアイコンはポーリングがオフを示します。[JTAG Chain] → [Auto Core Status Poll] からも実行できます。
- [Run] : [Trigger Setup] → [Run] (F5) からも実行できます。
- [Stop] : [Trigger Setup] → [Stop Acquisition] (F9) からも実行できます。
- [Trigger Immediate] : [Trigger Setup] → [Trigger Immediate] (Ctrl+F5) からも実行できます。
- [Go To X Marker] : [Waveform] → [Go To] → [Go To X Marker] からも実行できます。

- [Go To O Marker] : [Waveform] → [Go To] → [Go To O Marker] からも実行できます。
- [Go To Previous Trigger] : [Waveform] → [Go To] → [Trigger] → [Previous] からも実行できます。
- [Go To Next Trigger] : [Waveform] → [Go To] → [Trigger] → [Next] からも実行できます。
- [Zoom In] : [Waveform] → [Zoom] → [Zoom In] からも実行できます。
- [Zoom Out] : [Waveform] → [Zoom] → [Zoom Out] からも実行できます。
- [Fit Window] : [Waveform] → [Zoom] → [Zoom Fit] からも実行できます。

ChipScope Pro Analyzer のコマンド ライン オプション

Windows システムでは、コマンド ラインまたは [スタート] メニューから ChipScope Pro Analyzer を起動できます。

- 32 ビット Windows システム上では、コマンド ラインに次のように入力すると ChipScope Pro Analyzer を起動できます。

  `<XILINX_ISE_INSTALL>\bin\nt\analyzer.exe`

- 64 ビット Windows システム上では、コマンド ラインに次のように入力すると ChipScope Pro Analyzer を起動できます。

  `<XILINX_ISE_INSTALL>\bin\nt64\analyzer.exe`

- 32 ビット Linux システムでは、コマンド ラインに次のように入力すると ChipScope Pro Analyzerを 起動できます。

  `<XILINX_ISE_INSTALL>/bin/lin/analyzer`

- 64 ビット Linux システムでは、コマンド ラインに次のように入力すると ChipScope Pro Analyzer を起動できます。

  `<XILINX_ISE_INSTALL>/bin/lin64/analyzer`

  `<XILINX_ISE_INSTALL>` は ISE Design Suite ツールのインストール ディレクトリを指します。

オプションの引数

コマンド ラインから ChipScope Pro Analyzer を起動した場合には、コマンド ラインで次のオプションが使用できます。

`-geometry <width>x<height>+<left edge x coord>+<top edge y coord>`

ChipScope Pro Analyzer プログラム ウィンドウの位置、幅、高さを設定します。

`-project <path and filename>`

起動時に、特定のプロジェクト ファイルを読み込みます。デフォルトでは起動時にプロジェクト ファイルが読み込まれません。

`-init <path and filename>`

起動時に、指定された init ファイルを読み込んで、ChipScope Pro Analyzer の終了時に同じファイルに書き込みます。デフォルトは、`%userprofile%\.chipscope\cs_analyzer.ini` です。

`-log <path and filename>`

`-log stdout`

メッセージ ログを指定のファイルに書き込みます。stdout を指定した場合、標準出力に書き込みます。デフォルトは、$HOME/.chipscope/cs_analyzer.log です。

Windows のコマンド ライン例

```
C:\Xilinx\12.3\ISE_DS\ISE\bin\nt\analyzer.exe -log c:\proj\t\t.log -
init C:\proj\t\t.ini -project c:\proj\t\t.cpj -geometry 1000x300+30+600
```

XILINX®

第 5 章

ChipScope エンジン Tcl インターフェイス

概要

このインターフェイスでは、Tcl スクリプトを使用して、ChipScope™ ロジック アナライザー エンジンの通信ライブラリ経由で JTAG (Joint Test Action Group、IEEE 規格) ダウンロード ケーブルにアクセスできます。CSE/Tcl インターフェイスの目的は、基本的な JTAG、FPGA、VIO (Virtual Input/Output) コア ファンクションへアクセスするためのシンプルなスクリプト システムを提供することです。数行の Tcl スクリプトコマンドを使用することで、標準的なザイリンクス ケーブルを使用した JTAG チェーンのデバイスのスキャンおよび操作が可能になります。

JTAG の詳細は、『バウンダリ スキャン (JTAG) を使用した Virtex FPGA のコンフィギュレーションとリードバック』[211 ページのリファレンス 12 を参照] を参照してください。Tcl の詳細は、『Tcl Developer Xchange』[212 ページのリファレンス 26 を参照] を参照してください。

必要条件

- 『ISE Design Suite 13 : インストール、ライセンス、リリース ノート』[211 ページのリファレンス 14 を参照] に記述されるサポート OS が使用されたコンピューター システム
- プラットフォーム ケーブル USB、パラレル ケーブル IV、パラレル ケーブル III などのサポートされる JTAG ケーブル
- Tcl シェル (ChipScope Pro および ISE® Design Suite ツールのインストールに含まれる `xtclsh`) または ActiveTcl 8.4 シェル [212 ページのリファレンス 24 を参照]
- 必要な環境変数は xtclsh.exe (Windows) または xtclsh (Linux) を使用して設定

制限

ChipScope Engine Tcl インターフェイスでは、パフォーマンスよりも簡素化が重視されています。::chipscope::csejtag_tap_shift_chain_ir および ::chipscope::csejtag_tap_shift_chain_dr などのコマンドは、パックされたバイナリ データ構造ではなく、16 進数の文字列としてビットを送信します。大規模なデータ文字列を転送する場合は、ある程度パフォーマンスが低下しますが、API (アプリケーション プログラミング インターフェイス) のシンプルなデザインと Tcl スクリプト言語が使用されるため、JTAG チェーン接続されたデバイスおよびコアへのアクセスには、CSE/Tcl インターフェイスを使用するのが便利です。

注記 : CSE/Tcl インターフェイスは、JTAG ケーブル通信デバイス (ChipScope Pro Analyzer ソフトウェア ツールおよびザイリンクスの EDK (エンベデッド開発キット) に含まれるデバッガー ツールなど) とインターフェイスする CSE/Tcl を使用するソフトウェアとのみ互換性があります。

CSE/Tcl コマンド サマリ

CSE/Tcl インターフェイス コマンドは ::chipscope:: という名前空間に属します。CSE/Tcl インターフェイスのコマンドは 4 つのカテゴリに分類されます (表 5-1を参照)。

表 5-1 : CSE/Tcl コマンド カテゴリ

カテゴリ	説明
CseJtag	JTAG インターフェイス ステータスおよび制御コマンド (「CseJtag Tcl コマンド」を参照)
CseFpga	FPGA ステータスおよびコンフィギュレーション コマンド (121 ページの「CseFpga Tcl コマンド」を参照)
CseCore	ChipScope Pro コア ステータス コマンド (121 ページの「CseCore Tcl コマンド」を参照)
CseVIO	ChipScope Pro VIO コア ステータス コマンド (122 ページの「CseVIO Tcl コマンド」を参照)

CseJtag Tcl コマンド

CseJtag Tcl コマンドのカテゴリには、4 つのコマンドが含まれ (表 5-2 を参照)、それぞれのコマンドに 1 つまたは複数のサブコマンドが含まれます。

表 5-2 : CseJtag Tcl コマンド

コマンド	説明
`::chipscope::csejtag_session`	CseJtag セッションを管理します。セッションは、JTAG ターゲットに関するすべてのデータおよびメッセージを維持するために使用されます。このコマンドのサブコマンドについては、表 5-3 を参照してください。
`::chipscope::csejtag_db`	CseJtag JTAG データベースにアクセスします。CseJtag JTAG データベースには、既知の JTAG デバイスに関するすべてのデータが含まれます。このコマンドのサブコマンドについては、表 5-4 を参照してください。
`::chipscope::csejtag_target`	JTAG ダウンロード ケーブル、JTAG エミュレーター、その他 JTAG デバイスなどの CseJtag ターゲットへの接続を管理します。このコマンドのサブコマンドについては、119 ページの表 5-5 を参照してください。
`::chipscope::csejtag_tap`	CseJtag ターゲットの JTAG テスト アクセス ポート (TAP) にアクセスします。TAP ステート マシンのナビゲーション、TAP のデータのシフトなどの操作が含まれます。このコマンドのサブコマンドについては、120 ページの表 5-6 を参照してください。

表 5-3 は、CseJtag Tcl サブコマンドのサマリを示しています。これらのコマンドに関するその他の詳細は、123 ページの「CseJtag Tcl コマンド」を参照してください。

注記 : すべての CseJtag Tcl グローバル変数の宣言については、ChipScope Pro ツールのインストール ディレクトリの csejtagglobals.tcl ファイルを参照してください。

表 5-3 : ::chipscope::csejtag_session サブコマンドのサマリ

サブコマンド	説明
create	セッションを作成して初期化します。
destroy	既存セッションで使用されたメモリ リソースを削除して、メモリを空にします。
get_api_version	CseJtag API ライブラリ バージョン情報を取得します。
send_message	セッション　メッセージのルーター機能を使用してメッセージを送信します。

表 5-4 : ::chipscope::csejtag_db サブコマンドのサマリ

サブコマンド	説明
add_device_data	JTAG データベースへデバイス レコードを追加します。
lookup_device	JTAG データベースのデバイス情報を検索します。
get_device_name_for_idcode	IDCODE を使用して JTAG データベースからデバイス名を取得します。
parse_bsdl	バウンダリスキャン記述言語 (BSDL) バッファーを解析して JTAG デバイスのデバイス データを抽出します。
parse_bsdl_file	バウンダリスキャン記述言語 (BSDL) ファイルを解析して JTAG デバイスのデバイス データを抽出します。

表 5-5 : ::chipscope::csejtag_target サブコマンドのサマリ

サブコマンド	説明
open	JTAG ターゲットへの接続を開いて、セッションと関連付けます。
close	開いている JTAG ターゲットへの接続を終了して、セッションから削除します。
is_connected	ターゲットの接続ステータスをテストしてリターンします。
lock	JTAG ターゲットのロックを取得しようとします。
unlock	JTAG ターゲットのロックを解除します。
get_lock_status	JTAG ターゲットのロック ステータスを取得します。
clean_locks	すべてのケーブル　ロックを解除し、ロックに関連するリソースをクリーンアップします。
flush	JTAG ターゲットのデータ バッファーをフラッシュします。
set_pin	JTAG ターゲットの TAP ピンの値を設定します。
get_pin	JTAG ターゲットの TAP ピンの値を取得します。
pulse_pin	JTAG ターゲットの TAP ピンのパルスを送ります。

表 5-5 : ::chipscope::csejtag_target サブコマンドのサマリ (続き)

サブコマンド	説明
wait_time	指定した時間分待機します。
get_info	JTAG ターゲットに関連する情報を取得します。

表 5-6 : ::chipscope::csejtag_tap サブコマンドのサマリ

サブコマンド	説明
autodetect_chain	現在ターゲットに接続された JTAG チェーンに関する情報をすべて自動的に検出します。
interrogate_chain	JTAG チェーンをスキャンして、チェーンの長さとチェーンの各デバイスの IDCODE 情報を決定します。
get_device_count	JTAG チェーン内のデバイス数を取得します。
set_device_count	JTAG チェーン内のデバイス数を設定します。
get_irlength	デバイスの命令レジスタ (IR) の長さを取得します。
set_irlength	デバイスの命令レジスタ (IR) の長さを設定します。
get_device_idcode	デバイスの IDCODE を取得します。
set_device_idcode	デバイスの IDCODE を設定します。
navigate	JTAG TAP ステートへナビゲートします。
shift_chain_ir	ビットストリームを JTAG チェーンの命令レジスタに対してシフトインおよびシフトアウトします。
shift_chain_dr	ビットストリームを JTAG チェーンのデータ レジスタに対してシフトインおよびシフトアウトします。
shift_device_ir	ビットストリームを JTAG チェーンの特定デバイスの命令レジスタに対してシフトインおよびシフトアウトします。
shift_device_dr	ビットストリームを JTAG チェーンの特定デバイスのデータ レジスタに対してシフトインおよびシフトアウトします。

CseFpga Tcl コマンド

CseFpga Tcl コマンドのカテゴリには、複数のコマンドが含まれます (表 5-7 参照)。

注記 : すべての CseFpga Tcl グローバル変数の宣言については、ChipScope Pro ツールのインストール ディレクトリの csefpgaglobals.tcl ファイルを参照してください。

表 5-7 : CseFpga Tcl コマンド

コマンド	説明
::chipscope::csefpga_configure_device	.bit、.rbt、または .mcs ファイルの内容を含むバイト アレイで FPGA デバイスをコンフィギュレーションします。
::chipscope::csefpga_configure_device_with_file	.bit、.rbt、または .mcs ファイルの内容で FPGA デバイスをコンフィギュレーションします。
::chipscope::csefpga_get_config_reg	ターゲット FPGA デバイスのコンフィギュレーション レジスタ ビットを読み出します。
::chipscope::csefpga_get_instruction_reg	ターゲット FPGA デバイスの命令レジスタを読み出して、コンフィギュレーション特有のステータス ビットをフォーマットします。
::chipscope::csefpga_get_usercode	ターゲット FPGA デバイスの USERCODE レジスタを読み出します。
::chipscope::csefpga_get_user_chain_count	ターゲット FPGA デバイスの USER チェーン レジスタを決定します。
::chipscope::csefpga_is_config_supported	ターゲット FPGA デバイスでコンフィギュレーションがサポートされるかどうかテストします。
::chipscope::csefpga_is_configured	FPGA デバイスのコンフィギュレーション ステータスをリターンします。
::chipscope::csefpga_is_sys_mon_supported	ターゲット FPGA デバイスでシステム モニターのコマンドがサポートされるかどうかテストします。
::chipscope::csefpga_run_sys_mon_command_sequence	システム モニターのレジスタの読み出しと書き込みのシーケンスを実行します。
::chipscope::csefpga_get_sys_mon_reg	システム モニターのレジスタから読み出します。
::chipscope::csefpga_set_sys_mon_reg	システム モニターのレジスタに書き込みます。
::chipscope::csefpga_assign_config_data_to_device	バッファーに含まれているコンフィギュレーション マスク データを特定のデバイスに割り当てます。
::chipscope::csefpga_assign_config_data_file_to_device	ファイルに含まれているコンフィギュレーション マスク データを特定のデバイスに割り当てます。

CseCore Tcl コマンド

CseCore Tcl コマンドのカテゴリには、複数のコマンドが含まれます (表 5-8 参照)。

注記 : すべての CseCore Tcl グローバル変数の宣言については、ChipScope Pro ツールのインストール ディレクトリの csecoreglobals.tcl ファイルを参照してください。

表 5-8 : CseCore Tcl コマンド

コマンド	説明
::chipscope:: csecore_get_core_count	ターゲット FPGA デバイスの ICON コアと特定の USER スキャン チェーン レジスタに接続されたコアの数を取得します。
::chipscope:: csecore_get_core_status	ターゲット ChipScope Pro コアからスタティック ステータス ワードを読み出します。
::chipscope:: csecore_is_cores_supported	ターゲット FPGA デバイスで ChipScope Pro コアがサポートされるかどうかテストします。

CseVIO Tcl コマンド

CseVIO Tcl コマンドのカテゴリには、複数のコマンドが含まれます (表 5-9 参照)。

注記 : すべての CseVIO Tcl グローバル変数の宣言については、ChipScope Pro ツールのインストール ディレクトリの csevioglobals.tcl ファイルを参照してください。

表 5-9 : CseVIO Tcl コマンド

コマンド	説明
::chipscope::csevio_get_core_info	ターゲット VIO コアからスタティック ステータス ワードを読み出します。
::chipscope::csevio_is_vio_core	ターゲット コアが VIO コアかどうか判別します。
::chipscope::csevio_init_core	ターゲット VIO コアに関連するグローバル変数を初期化します。
::chipscope::csevio_terminate_core	ターゲット VIO コアに関連するグローバル変数を削除してメモリを解放します。
::chipscope::csevio_define_signal	指定した VIO 信号ビットの名前を定義します。
::chipscope::csevio_define_bus	VIO 信号ビットのグループ (バス) 名を定義します。
::chipscope::csevio_undefine_name	VIO 信号/バス名と関連するすべての情報を削除します。
::chipscope::csevio_write_values	ターゲット VIO コアの指定した信号/バスに値を書き込みます。
::chipscope::csevio_read_values	ターゲット VIO コアの指定した信号/バスから値を読み出します。

CseJtag Tcl コマンド

ここでは、次の CseJtag Tcl コマンドの詳細について説明します。

- ::chipscope::csejtag_session create
- ::chipscope::csejtag_session destroy
- ::chipscope::csejtag_session get_api_version
- ::chipscope::csejtag_session send_message
- ::chipscope::csejtag_target open
- ::chipscope::csejtag_target close
- ::chipscope::csejtag_target is_connected
- ::chipscope::csejtag_target lock
- ::chipscope::csejtag_target unlock
- ::chipscope::csejtag_target get_lock_status
- ::chipscope::csejtag_target clean_locks
- ::chipscope::csejtag_target flush
- ::chipscope::csejtag_target set_pin
- ::chipscope::csejtag_target get_pin
- ::chipscope::csejtag_target pulse_pin
- ::chipscope::csejtag_target wait_time
- ::chipscope::csejtag_target get_info
- ::chipscope::csejtag_tap autodetect_chain
- ::chipscope::csejtag_tap interrogate_chain
- ::chipscope::csejtag_tap get_device_count
- ::chipscope::csejtag_tap set_device_count
- ::chipscope::csejtag_tap get_irlength
- ::chipscope::csejtag_tap set_irlength
- ::chipscope::csejtag_tap get_device_idcode
- ::chipscope::csejtag_tap set_device_idcode
- ::chipscope::csejtag_tap navigate
- ::chipscope::csejtag_tap shift_chain_ir
- ::chipscope::csejtag_tap shift_device_ir
- ::chipscope::csejtag_tap shift_chain_dr
- ::chipscope::csejtag_tap shift_device_dr
- ::chipscope::csejtag_db add_device_data
- ::chipscope::csejtag_db lookup_device
- ::chipscope::csejtag_db get_device_name_for_idcode
- ::chipscope::csejtag_db get_irlength_for_idcode
- ::chipscope::csejtag_db parse_bsdl
- ::chipscope::csejtag_db parse_bsdl_file

::chipscope::csejtag_session create

これは、通常 ChipScope Engine への最初のサブコマンド呼び出しです。このコマンドでリターンされるセッション ハンドルでは、JTAG ターゲットを開いて制御できます。また、このコマンドはデフォルト ディレクトリのさまざまなデータ ファイルから取得されたデータを使用してセッションを初期化します。デフォルト ディレクトリは *<LIBCSEJTAG_DLL_PATH>*/../../data/cse (<LIBCSEJTAG_DLL_PATH>は libCseJtag.dll ファイルの絶対パス ディレクトリ) です。

構文

```
::chipscope::csejtag_session create messageRouterFn [opt_args...]
```

注記 : [opt_args...] は文字列または文字列形式のリストのオプションの引数リストです。

引数

表 5-10 : ::chipscope::csejtag_session create サブコマンドの引数

引数	タイプ	説明
messageRouterFn	必須	メッセージ ルーター関数の名前。すべてのメッセージを stdout にルートするには値に 00 を使用します。次は、関数宣言文の例です。 `proc messageRouterFn {handle msgFlags msg} { ...}` msgFlags で次のいずれかがリターンされます。 $CSE_MSG_ERROR $CSE_MSG_WARNING $CSE_MSG_STATUS $CSE_MSG_INFO $CSE_MSG_NOISE $CSE_MSG_DEBUG
-server *<host>*	オプション	*<cs_server_host_name>*で示される ChipScope サーバー ホスト名に関連するセッションを作成します。
-port *<portnum>*	オプション	*<cs_server_port_number>*で示される ChipScope サーバー ポート数に関連するセッションを作成します。

リターン

セッション ハンドル。コマンドがエラーになると例外になります。

例

1. オプションの引数を使用しないで新しいセッションを作成します。

   ```
   %set handle [::chipscope::csejtag_session create messageRouterFn]
   ```

2. ポート 50001 の lab_machine というサーバーにクライアント/サーバー ライブラリを使用して新しいセッションを作成します。

   ```
   %set handle [::chipscope::csejtag_session create messageRouterFn
   -server “lab_machine” -port “50001”]
   ```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_session destroy

このコマンドは既存セッションを削除して、そのセッションで前に使用されたリソースすべてを空にします。

構文

```
::chipscope::csejtag_session destroy handle
```

引数

表 5-11 : ::chipscope::csejtag_session create サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル

リターン

コマンドがエラーになると例外になります。

例

指定したセッションを削除します。

```
%::chipscope::csejtag_session destroy $handle
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_session get_api_version

このコマンドは CseJtag API ライブラリのバージョンを読み出します。

構文

```
::chipscope::csejtag_session get_api_version
```

引数

このコマンドには引数はありません。

リターン

API バージョン情報を含む Tcl リスト。リスト エレメント形式は次のとおりです。

```
{apiVersion versionString}
```

apiVersion は API バージョン番号で versionString はビルド バージョン番号です。コマンドがエラーになると例外になります。

例

API バージョン番号とビルド番号バージョンの文字列を含むリストを取得します。

```
%set api_info [::chipscope::csejtag_session get_api_version]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_session send_message

このサブコマンドは、CseJtag ライブラリのメッセージ ルーター関数にメッセージを送信します。

構文

```
::chipscope::csejtag_session send_message handle msgType msg
```

引数

表 5-12 : ::chipscope::csejtag_session send_message サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
msgType		メッセージ タイプは、次のいずれかに設定する必要があります。 ・ $CSE_MSG_ERROR ・ $CSE_MSG_WARNING ・ $CSE_MSG_STATUS ・ $CSE_MSG_INFO ・ $CSE_MSG_NOISE ・ $CSE_MSG_DEBUG
msg		メッセージ文字列

リターン

コマンドがエラーになると例外になります。

例

メッセージ ルーター関数に "Hello World!" メッセージを送信します。

```
%::chipscope::csejtag_session send_message $handle $CSE_MSG_INFO
"Hello World!"
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target open

このサブコマンドは、JTAG ターゲット デバイスを開いて、セッションと関連付けます。

注記：現在のところ、セッションごとに開くことのできる JTAG ターゲットは 1 つだけです。

構文

```
::chipscope::csejtag_target open handle targetName
progressCallbackFunc [opt_args...]
```

注記：[opt_args...]は文字列または文字列形式のリストのオプションの引数リストです。

引数

表 5-13 : ::chipscope::csejtag_target open サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
targetName		開く JTAG ターゲットの名前。使用可能な targetName と [optional args...] の組み合わせについては、表 5-14 を参照してください。targetName が $CSEJTAG_TARGET_AUTO に設定されている場合、最初の使用可能な JTAG ケーブル ターゲットが開きます。
progressCallbackFunc		JTAG ターゲットの操作の進捗状況を監視するために使用できるプログレス コールバック関数です。この関数の形式は、次のようになります。 `proc progressCallbackFunc (handle totalCount CurrentCount progressStatus) {...}` プログレス コールバック関数は $CSE_STOP または $CSE_CONTINUE をリターンします。プログレス コールバック関数が必要でない場合は、引数に 0 を使用します。

表 5-14 は、targetName 引数値とそれらのオプションの引数の有効な組み合わせを示しています。

表 5-14 : targetName 引数と [optional args...] の組み合わせ

targetName	[optional args...]
$CSEJTAG_TARGET_AUTO	なし
$CSEJTAG_TARGET_PARALLEL	"port={LPT1 | LPT2 | LPT3}" "frequency={5000000 | 2500000 | 200000}"
$CSEJTAG_TARGET_PLATFORMUSB	"port=USB2 (aliased to USB21) | USB21 | USB22 | USB23 | ..." "ESN=<electronic serial number string>" "frequency={12000000 | 6000000 | 3000000 | 1500000 | 750000}"

リターン

フォーマットのリストは、次のようになります。

```
{target_name plugin_name fw_ver driver_ver plugin_ver vendor frequency
port full_name target_uid rawinfo target_flags}
```

説明：

値	説明
target_name	targetName 文字列と同じ
plugin_name	プラグイン ライブラリ名の文字列
fw_ver	ファームウェア バージョンの文字列
driver_ver	ドライバー バージョンの文字列
plugin_ver	プラグイン バージョンの文字列
vendor	ベンダー文字列
frequency	周波数文字列
port	ポート文字列
full_name	ターゲット名すべての文字列
target_uid	ターゲット特有の ID 文字列。ザイリンクス プラットフォーム ケーブル USB の場合、これは ESN (Electronic Serial Number) です。
rawinfo	そのままのターゲット情報文字列
target_flags	ターゲット特有のフラグを含んだ整数

注記：コマンドがエラーになると例外になります。

例

1. 自動検出するようにし、ターゲット ケーブルを開き、開いたターゲットの情報をリターンします。

```
%set targetInfo [::chipscope::csejtag_target open $handle
$CSEJTAG_TARGET_AUTO progressFunc]
```

2. LPT1 ポートのパラレル ケーブルを周波数 200000 で開き、開いたターゲットの情報をリターンします。

```
%set targetInfo [::chipscope::csejtag_target open $handle
$CSEJTAG_TARGET_PARALLEL progressFunc “port=LPT1” “frequency=200000”]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target close

このサブコマンドは、前に開いた JTAG ターゲット デバイスを閉じます。

構文

```
::chipscope::csejtag_target close handle
```

引数

表 5-15 : ::chipscope::csejtag_target close サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル

リターン

コマンドがエラーになると例外になります。

例

指定したセッションで現在のターゲットを閉じます。

```
%::chipscope::csejtag_target close $handle
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target is_connected

このサブコマンドは、JTAG ターゲット デバイスの接続ステータスをテストします。

構文

```
::chipscope::csejtag_target is_connected handle
```

引数

表 5-16 : ::chipscope::csejtag_target is_connected サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル

リターン

接続ステータスは、次のようにリターンされます。

- 1 の場合は、ターゲットへの接続が開いていて、アクティブなことを示します。
- 0 の場合は、閉じていることを示します。

コマンドがエラーになると例外になります。

例

指定したセッションで現在のターゲットをリターンします。

```
%set isConnected (::chipscope::csejtag_target is_connected $handle)
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target lock

このサブコマンドは、前に開いた JTAG ターゲット デバイスのロックを取得します。

構文

```
::chipscope::csejtag_target lock handle msWait
```

引数

表 5-17 : ::chipscope::csejtag_target lock サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
msWait		停止する前の待機時間 (ミリ秒)。-1 は lock が取得されるまで待機を意味します。

リターン

ロック ステータスの形式は、次のいずれかになります。

- $CSEJTAG_LOCKED_ME
- $CSEJTAG_LOCKED_OTHER

- $CSEJTAG_UNKNOWN

コマンドがエラーになると例外になります。

例

ターゲット ロックを取得しようとし、最低 1000ms 待機し、ロックのステータスを取得します。

```
%set lockStatus [::chipscope::csejtag_target lock $handle 1000]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target unlock

このサブコマンドは、前に開いたロックされている JTAG ターゲット デバイスのロックを解除します。

構文

```
::chipscope::csejtag_target unlock handle
```

引数

表 5-18 : ::chipscope::csejtag_target unlock サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル

リターン

コマンドがエラーになると例外になります。

例

指定したセッションでターゲットのロックを解除します。

```
%::chipscope::csejtag_target unlock $handle
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target get_lock_status

このサブコマンドは、ターゲット デバイスのロック ステータスを読み出します。

構文

```
::chipscope::csejtag_target get_lock_status handle
```

引数

表 5-19 : ::chipscope::csejtag_target get_lock_status サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル

リターン

ロック ステータスの形式は、次のいずれかになります。

- $CSEJTAG_LOCKED_ME
- $CSEJTAG_LOCKED_OTHER
- $CSEJTAG_UNKNOWN

コマンドがエラーになると例外になります。

例

現在のロックの テータスを取得します。

```
%set lockStatus [::chipscope::csejtag_target get_lock_status $handle]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target clean_locks

このサブコマンドは、すべての JTAG ターゲット ロックをクリーンアップします。

注記：このサブコマンドは、最後の手段としてのみ使用してください。このサブコマンドを実行すると、ほかのプロセスおよびアプリケーションで使用されるものも含め、すべての共有セマフォが停止 (kill) されます。現在のところは、JTAG ケーブル ターゲットのロックのみがクリーンアップされます。

構文

```
::chipscope::csejtag_target clean_locks handle
```

引数

表 5-20 : ::chipscope::csejtag_target clean_locks サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル

リターン

コマンドがエラーになると例外になります。

例

アプリケーションが予期せず閉じたり、::chipscope::csejtag_target open でターゲットを問題なく開くことができなかったりするので、このコマンドは最後の手段として使用します。

```
%::chipscope::csejtag_target clean_locks $handle
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target flush

このサブコマンドは、前に開いたロックされている JTAG ターゲット デバイスに関連するバッファーをフラッシュします。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

構文

```
::chipscope::csejtag_target flush handle
```

引数

表 5-21 : ::chipscope::csejtag_target flush サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル

リターン

コマンドがエラーになると例外になります。

例

開いているロックされた JTAG ターゲットのバッファーをフラッシュして、データ書き込みがすぐにできるようにします。

```
%::chipscope::csejtag_target flush $handle
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target set_pin

このサブコマンドは、前に開いたロックされている JTAG ターゲット デバイスの JTAG TAP ピンの値を設定します。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

この関数を使用して JTAG TAP ステートを変更する場合、CseJtag Tcl ライブラリは CseJtag Tcl ステートを記録しないことに注意してください。::chipscope::csejtag_tap サブコマンドのいずれかを使用する場合は、その前に ::chipscope::csejtag_tap navigate サブコマンドを使用して JTAG TAP ステート マシンを $CSEJTAG_TEST_LOGIC_RESET ステートに設定しておきます。

構文

```
::chipscope::csejtag_target set_pin handle pin value
```

引数

表 5-22 : ::chipscope::csejtag_target set_pin サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
pin		JTAG TAP ピン識別子 {$CSEJTAG_TMS | $CSEJTAG_TDI}$CSEJTAG_TCK ピンを変更するには、::chipscope::csejtag_target pulse_pin サブコマンドを使用します。
value		JTAG TAP ピンの値 {1=set, 0=clear}

リターン

コマンドがエラーになると例外になります。

例

TMS ピンを 1 に設定します。

```
%::chipscope::csejtag_target set_pin $handle $CSEJTAG_TMS 1
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target get_pin

このサブコマンドは、前に開いたロックされている JTAG ターゲット デバイスの JTAG TAP ピンの値を読み出します。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

構文

```
::chipscope::csejtag_target get_pin handle pin
```

引数

表 5-23 : ::chipscope::csejtag_target get_pin サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
pin		JTAG TAP ピン識別子 {$CSEJTAG_TMS | $CSEJTAG_TCK | $CSEJTAG_TDI | $CSEJTAG_TDO}

リターン

JTAG TAP ピンの値 {1=set, 0=clear}

コマンドがエラーになると例外になります。

例

TDO ピンの現在の値を取得します。

```
%set value [::chipscope::csejtag_target set_pin $handle $CSEJTAG_TDO]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target pulse_pin

このサブコマンドは、前に開いたロックされている JTAG ターゲット デバイスの JTAG TAP ピンの値にパルスを送ります。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

この関数を使用して JTAG TAP ステートを変更する場合、CseJtag Tcl ライブラリで CseJtag Tcl ステートが記録されないことに注意してください。::chipscope::csejtag_tap サブコマンドのいずれかを使用する場合は、その前に ::chipscope::csejtag_tap navigate サブコマンドを使用して JTAG TAP ステート マシンを $CSEJTAG_TEST_LOGIC_RESET ステートに設定しておきます。

構文

```
::chipscope::csejtag_target pulse_pin handle pin count
```

引数

表 5-24 : ::chipscope::csejtag_target pulse_pin サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
pin		JTAG TAP ピン識別子 {$CSEJTAG_TMS | $CSEJTAG_TCK | $CSEJTAG_TDI}
count		JTAG TAP にパルスを送る回数 (パルスとは、ピンに 0 を駆動して 1 を駆動してから、0 を駆動することを意味します)

リターン

コマンドがエラーになると例外になります。

例

TCK ピンに 5 回パルスを送ります。

```
%::chipscope::csejtag_target pulse_pin $handle $CSEJTAG_TCK 5
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target wait_time

このサブコマンドは、指定した時間 (マイクロ秒) 分待機します。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

構文

```
::chipscope::csejtag_target wait_time handle usecs
```

引数

表 5-25 : ::chipscope::csejtag_target wait_time サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
usecs		待機するマイクロ秒

リターン

コマンドがエラーになると例外になります。

例

JTAG ターゲットに別の操作を実行する前に 1000 マイクロ秒待機するように命令します。

```
%::chipscope::csejtag_target wait_time $handle 1000
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_target get_info

このサブコマンドは、前に開いた JTAG ターゲットから情報を読み出します。

注記：この関数を呼び出す前に JTAG ターゲット ロックを取得する必要はありません。

構文

```
::chipscope::csejtag_target get_info handle
```

引数

表 5-26 : ::chipscope::csejtag_target get_info サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル

リターン

フォーマットのリストは、次のようになります。

```
{target_name plugin_name fw_ver driver_ver plugin_ver vendor frequency
port full_name target_uid rawinfo target_flags}
```

説明：

値	説明
target_name	JTAG ターゲットの名前
plugin_name	プラグイン ライブラリ名の文字列
fw_ver	ファームウェア バージョンの文字列
driver_ver	ドライバー バージョンの文字列
plugin_ver	プラグイン バージョンの文字列
vendor	ベンダー文字列
frequency	周波数文字列
port	ポート文字列
full_name	ターゲット名すべての文字列
target_uid	ターゲット特有の ID 文字列。
rawinfo	そのままのターゲット情報文字列
target_flags	ターゲット特有のフラグを含んだ整数

注記：コマンドがエラーになると例外になります。

例

現在の JTAG ターゲットに関する情報を取得します。

```
%set targetInfo [::chipscope::csejtag_target get_info $handle]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap autodetect_chain

このサブコマンドは、JTAG チェーンの構成を自動的に検出します。まず、JTAG チェーンに含まれるデバイス数とその IDCODE を取得し、IDCODE を持つ JTAG チェーンのデバイスの命令レジスタ (IR) 長を決定します。デバイスの IR 長に対応する IDCODE がない場合は、手動で割り当てる必要があります。問題なく終了すると、すべての関連するデバイス情報が決定され、セッションで設定されます。IEEE 1149.1 と互換性のないデバイスの中には、このサブコマンドを認識せず、チェーン全体が間違って検出されたり、全く検出されなかったりするものもあります。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

構文

```
::chipscope::csejtag_tap autodetect_chain handle algorithm
```

引数

表 5-27 : ::chipscope::csejtag_tap autodetect_chain サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
algorithm		JTAG チェーンの構成決定するために使用するアルゴリズムで、{$CSEJTAG_SCAN_DEFAULT | $CSEJTAG_SCAN_TLRSHIFT | $CSEJTAG_SCAN_WALKING_ONES} のいずれかに設定できます。 CSEJTAG_SCAN_WALKING_ONES アルゴリズムでは、次を実行できます。 ・ 1 の長いストリームを IR にシフトして、各デバイスを BYPASS に設定します。 ・ DR パターンを TDI にシフトし、TDO のパターンを待ちます。シフト数によって JTAG チェーン内のデバイス数が決まります。 ・ CSEJTAG_SCAN_TLRSHIFT アルゴリズムを実行すると、各デバイスの IDCODE が取得されます。 CSEJTAG_SCAN_TLRSHIFT アルゴリズムでは、次を実行できます。 ・ TLR にナビゲートします。 すべての IDCODE (または BYPASS ビット) が読み出されるまでビットをシフトアウトします。

リターン

サブコマンドでチェーンが完全に検出できなかった場合は、例外になります。このようなエラーの場合、JTAG チェーンのデバイスを検出し、手動で割り当てる必要があります。

例

デフォルトのアルゴリズムを使用してチェーンが自動的に検出されるようにします。

```
%::chipscope::csejtag_tap autodetect_chain $handle
$CSEJTAG_SCAN_DEFAULT
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap interrogate_chain

このサブコマンドは JTAG チェーンをスキャンし、チェーンの IDCODE およびデバイス数を取得します。IEEE 1149.1 と互換性のないデバイスの中には、このサブコマンドを認識せず、チェーン全体が間違って検出されたり、全く検出されなかったりするものもあります。このコマンドにより、各デバイスの命令レジスタ (IR) はアップデートされません。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

構文

```
::chipscope::csejtag_tap interrogate_chain handle algorithm
```

引数

表 5-28 : ::chipscope::csejtag_tap interrogate_chain サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
algorithm		JTAG チェーンの構成決定するために使用するアルゴリズムで、{$CSEJTAG_SCAN_DEFAULT | $CSEJTAG_SCAN_TLRSHIFT | $CSEJTAG_SCAN_WALKING_ONES} のいずれかに設定できます。 CSEJTAG_SCAN_WALKING_ONES アルゴリズムでは、次を実行できます。 ・ 1 の長いストリームを IR にシフトして、各デバイスを BYPASS に設定します。 ・ DR パターンを TDI にシフトし、TDO のパターンを待ちます。シフト数によって JTAG チェーン内のデバイス数が決まります。 ・ CSEJTAG_SCAN_TLRSHIFT アルゴリズムを実行すると、各デバイスの IDCODE が取得されます。 CSEJTAG_SCAN_TLRSHIFT アルゴリズムでは、次を実行できます。 ・ TLR にナビゲートします。 ・ すべての IDCODE (または BYPASS ビット) が読み出されるまでビットをシフトアウトします。

リターン

コマンドがエラーになると例外になります。

例

デフォルトのアルゴリズムを使用してチェーンの情報を取得します。

```
%::chipscope::csejtag_tap interrogate_chain $handle
$CSEJTAG_SCAN_DEFAULT
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap get_device_count

このサブコマンドは、現在の JTAG チェーンに含まれるデバイス数を取得するために使用します。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

構文

```
::chipscope::csejtag_tap get_device_count handle
```

引数

表 5-29 : ::chipscope::csejtag_tap get_device_count サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル

リターン

チェーンのデバイス数

コマンドがエラーになると例外になります。

例

JTAG チェーン内のデバイス数を取得します。

```
%set deviceCount [::chipscope::csejtag_tap get_device_count $handle]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap set_device_count

このサブコマンドは、現在の JTAG チェーンに含まれるデバイス数を設定するために使用します。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

構文

```
::chipscope::csejtag_tap set_device_count handle count
```

引数

表 5-30 : ::chipscope::csejtag_tap set_device_count サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
count		JTAG チェーン内のデバイス数

リターン

コマンドがエラーになると例外になります。

例

JTAG チェーン内のデバイス数を 4 に設定します。

```
%::chipscope::csejtag_tap set_device_count $handle 4
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap get_irlength

このサブコマンドは、現在の JTAG チェーンに含まれるデバイスの命令レジスタ (IR) 長を読み出します。IR 長は、命令をデバイス レジスタにシフトするために必要なパディングの量を指定するために使用します。TAP シフトとナビゲートは、すべてのデバイスの IR 長が正しく設定されるまで実行されません。::chipscope::csejtag_tap autodetect_chain サブコマンドを使用すると、IDCODE コマンドをサポートするチェーンのデバイスすべての IR 長を自動的に設定されます。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。また、デバイス カウントはこのサブコマンドを呼び出す前に ::chipscope::csejtag_tap set_device_count を使用して設定されている必要があります。

構文

```
::chipscope::csejtag_tap get_irlength handle deviceIndex
```

引数

表 5-31 : ::chipscope::csejtag_tap get_irlength サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)

リターン

デバイスの IR 長を指定します。

コマンドがエラーになると例外になります。

例

インデックス 0 のデバイスの IR 長を取得します。

```
%set irLength [::chipscope::csejtag_tap get_irlength $handle 0]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap set_irlength

このサブコマンドは、現在の JTAG チェーンに含まれる 1 つのデバイスの命令レジスタ (IR) 長を設定します。IR 長は、命令をデバイス レジスタにシフトするために必要なパディングの量を指定するために使用します。TAP シフトとナビゲートは、すべてのデバイスの IR 長が正しく設定されるまで実行されません。::chipscope::csejtag_tap autodetect_chain サブコマンドを使用すると、IDCODE コマンドをサポートするチェーンのデバイスすべての IR 長を自動的に設定されます。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。また、デバイス カウントはこのサブコマンドを呼び出す前に ::chipscope::csejtag_tap set_device_count を使用して設定されている必要があります。

構文

```
::chipscope::csejtag_tap set_irlength handle deviceIndex irLength
```

引数

表 5-32 : ::chipscope::csejtag_tap set_irlength サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
irLength		IR の長さ (ビット)

リターン

コマンドがエラーになると例外になります。

例

インデックス 0 から 11 ビットのデバイスの IR 長を設定します。

```
%::chipscope::csejtag_tap set_irlength $handle 0 11
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap get_device_idcode

このサブコマンドは、現在の JTAG チェーンの中の指定デバイスの 32 ビット IDCODE をリターンします。デバイスで IDCODE 命令がサポートされない場合は、null 文字列がリターンされます。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。また、デバイス カウントはこのサブコマンドを呼び出す前に ::chipscope::csejtag_tap set_device_count を使用して設定されている必要があります。

構文

```
::chipscope::csejtag_tap get_device_idcode handle deviceIndex
```

引数

表 5-33 : ::chipscope::csejtag_tap get_device_idcode サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)

リターン

デバイスの 32 ビットの IDCODE を表す 1 と 0 の 32 文字の文字列

コマンドがエラーになると例外になります。

例

インデックス 0 のデバイスの IDCODE を取得します。

```
%set idcode [::chipscope::csejtag_tap get_device_idcode $handle 0]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap set_device_idcode

このサブコマンドは、現在の JTAG チェーンの中の指定デバイスの IDCODE を設定します。null 文字列を渡すことで、デバイスで IDCODE 命令がサポートされないことが示されます。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。また、デバイス カウントはこのサブコマンドを呼び出す前に ::chipscope::csejtag_tap set_device_count を使用して設定されている必要があります。

構文

```
::chipscope::csejtag_tap set_device_idcode handle deviceIndex idcode
```

引数

表 5-34 : ::chipscope::csejtag_tap set_device_idcode サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
idcode		デバイスの 32 ビットの IDCODE を表す 1 と 0 の 32 文字の文字列

リターン

コマンドがエラーになると例外になります。

例

インデックス 0 ～ 01010101010101010101010101010101 のデバイスの IDCODE を設定します。

```
%::chipscope::csejtag_tap set_device_idcode $handle 0
“01010101010101010101010101010101”
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap navigate

このサブコマンドは、JTAG チェーンに含まれるデバイスの TAP ステートを変更するために使用します。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

構文

```
::chipscope::csejtag_tap navigate handle newState clockRepeat
microseconds
```

引数

表 5-35 : ::chipscope::csejtag_tap navigate サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
newState		ナビゲートする新規ステート
clockRepeat		新規ステートになった後に追加で TCK ピンにパルスを送る回数
microseconds		新規ステートにナビゲート後にスリープ状態になるマイクロ秒数

リターン

コマンドがエラーになると例外になります。

例

TAP ステートから Test Logic Reset にナビゲートし、5 クロック サイクル追加でこのステートが維持されるようにします。

```
%::chipscope::csejtag_tap navigate $handle $CSEJTAG_TEST_LOGIC_RESET 5 0
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap shift_chain_ir

このサブコマンドは、ビットストリームを JTAG チェーンの命令レジスタに対してシフトインおよびシフトアウトするために使用します。デバイス パディングは実行されません。デバイス インデックスの付いた IR シフトについては、151 ページの「::chipscope::csejtag_tap shift_device_ir」を参照してください。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

構文

```
::chipscope::csejtag_tap shift_chain_ir handle shiftMode exitState
progressCallbackFunc bitCount hextdibuf [-hextdimask hextdimaskval] [-
hextdomask hextdomaskval]
```

引数

表 5-36 : ::chipscope::csejtag_tap shift_chain_ir サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
shiftMode		{CSJTAG_SHIFT_READ | CSJTAG_SHIFT_WRITE | CSJTAG_SHIFT_READWRITE)
exitState		シフトが完了した後の終了ステート (ステートを変更しない場合は CSEJTAG_SHIFT_IR)
progressCallbackFunc		JTAG ターゲットの操作の進捗状況を監視するために使用できるプログレス コールバック関数です。この関数の形式は、次のようになります。 proc progressCallbackFunc (handle totalCount CurrentCount progressStatus) {...} プログレス コールバック関数は $CSE_STOP または $CSE_CONTINUE をリターンします。プログレス コールバック関数が必要でない場合は、引数に 0 を使用します。
bitCount		シフトするビット数
hextdibuf		TDL に書き込むデータ ビットを維持するデータ バッファー。最小位ビット (LSB) は TDI にまずシフトされます。
-hextdimask hextdimaskval	オプション	データが JTAG TAP の TDI ピンにシフトされる前にマスク ワードの hextdimaskval がデータ バッファーのビットに適用されるように指定します。
-hextdomask hextdomaskval		データが JTAG TAP の TDO ピンからシフトアウトされた後にマスク ワードの hextdomaskval がデータ バッファーのビットに適用されるように指定します。

リターン

JTAG TAP の TDO ピンからシフトアウトされるデータでフルになったバッファー。

コマンドがエラーになると例外になります。

例

命令レジスタに 64 個の 1 をシフトインし、その 64 ビットの受信データをキャプチャし、終了したら Run Test Idle ステートにナビゲートします。

```
%set hextdobuf [::chipscope::csejtag_tap shift_chain_ir $handle
$CSEJTAG_SHIFT_READWRITE $CSEJTAG_RUN_TEST_IDLE progressFunc 64
“FFFFFFFFFFFFFFFF”]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap shift_device_ir

このサブコマンドは、ビットストリームを JTAG チェーン内の特定デバイスの命令レジスタに対してシフトインおよびシフトアウトするために使用します。デバイス パディングは、その他すべてのデバイスを BYPASS モードにすると実行されます。このサブコマンドは ::chipscope::csejtag_tap shift_device_dr よりも前に呼び出してターゲット以外のデバイスをすべて BYPASS モードにしておかないと、予測、意図しない結果になる可能性があります。そのままのデータをチェーンの IR にシフトする場合は、149 ページの「::chipscope::csejtag_tap shift_chain_ir」を参照してください。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。また、デバイス IR にシフトされるビット数がデバイスの IR 長と一致しない場合、このサブコマンドはエラーになります。

構文

```
::chipscope::csejtag_tap shift_device_ir handle deviceIndex shiftMode
exitState progressCallbackFunc bitCount hextdibuf [-hextdimask
hextdimaskval] [-hextdomask hextdomaskval]
```

引数

表 5-37 : ::chipscope::csejtag_tap shift_device_ir サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
shiftMode		{CSJTAG_SHIFT_READ | CSJTAG_SHIFT_WRITE | CSJTAG_SHIFT_READWRITE)
exitState		シフトが完了した後の終了ステート (ステートを変更しない場合は CSEJTAG_SHIFT_IR)
progressCallbackFunc		JTAG ターゲットの操作の進捗状況を監視するために使用できるプログレス コールバック関数です。この関数の形式は、次のようになります。 proc progressCallbackFunc (handle totalCount CurrentCount progressStatus) {...} プログレス コールバック関数は $CSE_STOP または $CSE_CONTINUE をリターンします。プログレス コールバック関数が必要でない場合は、引数に 0 を使用します。
bitCount		シフトするビット数
hextdibuf		TDL に書き込むデータ ビットを維持するデータ バッファー。最小位ビット (LSB) は TDI にまずシフトされます。
-hextdimask hextdimaskval	オプション	データが JTAG TAP の TDI ピンにシフトされる前にマスク ワードの hextdimaskval がデータ バッファーのビットに適用されるように指定します。
-hextdomask hextdomaskval		データが JTAG TAP の TDO ピンからシフトアウトされた後にマスク ワードの hextdomaskval がデータ バッファーのビットに適用されるように指定します。

リターン

JTAG TAP の TDO ピンからシフトアウトされるデータでフルになったバッファー。

コマンドがエラーになると例外になります。

例

インデックス 1 のデバイスの命令レジスタに 11 個の 1 をシフトインし、その 11 ビットの受信データをキャプチャし、終了したら Run Test Idle ステートにナビゲートします。

```
%set hextdobuf [::chipscope::csejtag_tap shift_device_ir $handle 1
$CSEJTAG_SHIFT_READWRITE $CSEJTAG_RUN_TEST_IDLE progressFunc 11 “7FF”]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap shift_chain_dr

このサブコマンドは、ビットストリームを JTAG チェーンのデータ レジスタ (DR) に対してシフトインおよびシフトアウトするために使用します。デバイス パディングは実行されません。デバイス インデックスの付いた DR シフトについては、::chipscope::csejtag_tap shift_device_dr サブコマンドを参照してください。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。

構文

```
::chipscope::csejtag_tap shift_chain_dr handle shiftMode exitState
progressCallbackFunc bitCount hextdibuf [-hextdimask hextdimaskval] [-
hextdomask hextdomaskval]
```

引数

表 5-38 : ::chipscope::csejtag_tap shift_chain_dr サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
shiftMode		{CSJTAG_SHIFT_READ | CSJTAG_SHIFT_WRITE | CSJTAG_SHIFT_READWRITE}
exitState		シフトが完了した後の終了ステート (ステートを変更しない場合は CSEJTAG_SHIFT_DR)
progressCallbackFunc		JTAG ターゲットの操作の進捗状況を監視するために使用できるプログレス コールバック関数です。この関数の形式は、次のようになります。 `proc progressCallbackFunc {handle totalCount CurrentCount progressStatus} {...}` プログレス コールバック関数は $CSE_STOP または $CSE_CONTINUE をリターンします。プログレス コールバック関数が必要でない場合は、引数に 0 を使用します。
bitCount		シフトするビット数
hextdibuf		TDI に書き込むデータ ビットを保持するデータ バッファー。最小位ビット (LSB) は TDI にまずシフトされます。
-hextdimask hextdimaskval	オプション	データが JTAG TAP の TDI ピンにシフトされる前にマスク ワードの hextdimaskval がデータ バッファーのビットに適用されるように指定します。
-hextdomask hextdomaskval		データが JTAG TAP の TDO ピンからシフトアウトされた後にマスク ワードの hextdomaskval がデータ バッファーのビットに適用されるように指定します。

リターン

JTAG TAP の TDO ピンからシフトアウトされるデータでフルになったバッファー。

コマンドがエラーになると例外になります。

例

命令レジスタに 64 個の 1 をシフトインし、その 64 ビットの受信データをキャプチャし、終了したら Run Test Idle ステートにナビゲートします。

```
%set hextdobuf [::chipscope::csejtag_tap shift_chain_dr $handle
$CSEJTAG_SHIFT_READWRITE $CSEJTAG_RUN_TEST_IDLE progressFunc 64
“FFFFFFFFFFFFFFFF”]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_tap shift_device_dr

このサブコマンドは、ビットストリームを JTAG チェーン内の特定デバイスのデータ レジスタに対してシフトインおよびシフトアウトするために使用します。ターゲット以外のデバイスがすべて BYPASS モードの場合は、デバイス パディングが実行され、必要な立ち上がりおよび立ち下がりビットが追加され、チェーン内のターゲット デバイスの位置が調整されます。そのままのデータをチェーンの DR にシフトする場合は、::chipscope::csejtag_tap shift_chain_dr サブコマンドを参照してください。

注記 : JTAG ターゲットは、このサブコマンドを呼び出す前に ::chipscope::csejtag_target lock サブコマンドを使用してロックされている必要があります。このサブコマンドは ::chipscope::csejtag_tap shift_device_dr よりも前に呼び出してターゲット以外のデバイスをすべて BYPASS モードにしておかないと、予測、意図しない結果になる可能性があります。

構文

```
::chipscope::csejtag_tap shift_device_dr handle deviceIndex shiftMode
exitState progressCallbackFunc bitCount hextdibuf [-hextdimask
hextdimaskval] [-hextdomask hextdomaskval]
```

引数

表 5-39 : ::chipscope::csejtag_tap shift_device_dr サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
shiftMode		{CSJTAG_SHIFT_READ | CSJTAG_SHIFT_WRITE | CSJTAG_SHIFT_READWRITE)
exitState		シフトが完了した後の終了ステート (ステートを変更しない場合は CSEJTAG_SHIFT_DR)
progressCallbackFunc		JTAG ターゲットの操作の進捗状況を監視するために使用できるプログレス コールバック関数です。この関数の形式は、次のようになります。 proc progressCallbackFunc (handle totalCount CurrentCount progressStatus) {...} プログレス コールバック関数は $CSE_STOP または $CSE_CONTINUE をリターンします。プログレス コールバック関数が必要でない場合は、引数に 0 を使用します。
bitCount		シフトするビット数
hextdibuf		TDI に書き込むデータ ビットを保持するデータ バッファー。最小位ビット (LSB) は TDI にまずシフトされます。
-hextdimask hextdimaskval	オプション	データが JTAG TAP の TDI ピンにシフトされる前にマスク ワードの hextdimaskval がデータ バッファーのビットに適用されるように指定します。
-hextdomask hextdomaskval		データが JTAG TAP の TDO ピンからシフトアウトされた後にマスク ワードの hextdomaskval がデータ バッファーのビットに適用されるように指定します。

リターン

JTAG TAP の TDO ピンからシフトアウトされるデータでフルになったバッファー。

コマンドがエラーになると例外になります。

例

インデックス 1 のデバイスのデータ レジスタに 11 個の 1 をシフトインし、その 11 ビットの受信データをキャプチャし、終了したら Run Test Idle ステートにナビゲートします。

```
%set hextdobuf [::chipscope::csejtag_tap shift_device_dr $handle 1
$CSEJTAG_SHIFT_READWRITE $CSEJTAG_RUN_TEST_IDLE progressFunc 11 “7FF”]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_db add_device_data

このサブコマンドは、ファイルからデバイス レコードを読み出し、それを CseJtag ライブラリ内のメモリ ベースのルックアップ テーブルに追加するために使用します。

注記 : ファイル形式とデバイス レコードの構造は、idcode.lst ファイルと同じです。

構文

```
::chipscope::csejtag_db add_device_data handle filename buf bufLen
```

引数

表 5-40 : ::chipscope::csejtag_db add_device_data サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
filename		デバイス レコードが読み出されるファイル名を含む文字列
buf		ファイル形式とデバイス レコードの構造は、idcode.lst ファイルと同じ
bufLen		バッファーのサイズ (バイトまたは文字数)

リターン

コマンドがエラーになると例外になります。

例

my_idcode.lst ファイルから内部デバイス データベースにデータを追加します。また、データ レコード バッファーとバッファー サイズをローカル変数に保存します。

```
%::chipscope::csejtag_db add_device_data $handle “my_idcode.lst”
$my_idcode_buf $my_idcode_bufLen
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_db lookup_device

このサブコマンドは、デバイスの IDCODE を使用してデータベースでデバイスを検索するために使用します。

構文

```
::chipscope::csejtag_db lookup_device handle idcode
```

引数

表 5-41 : ::chipscope::csejtag_db lookup_device サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
idcode		該当デバイスの IDCODE

リターン

フォーマットのリストは、次のようになります。

```
{deviceName irlen cmd_bypass}
```

説明 :

deviceName
: デバイス名を含む文字列

irlen
: デバイスの IR のビット数

cmd_bypass
: デバイスの BYPASS 命令を含む文字列 (通常すべて 1)

コマンドがエラーになると例外になります。

例

IDCODE の 01010101010101010101010101010101 に含まれるデバイス情報をデータベースで検索します。

```
%set deviceInfo [::chipscope::csejtag_db lookup_device $handle
"01010101010101010101010101010101"]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_db get_device_name_for_idcode

このサブコマンドは、デバイスの IDCODE を使用してデータベースからデバイス名を取得するために使用します。

構文

```
::chipscope::csejtag_db get_device_name_for_idcode handle idcode
```

引数

表 5-42 : ::chipscope::csejtag_db get_device_name_for_idcode サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
idcode		該当デバイスの IDCODE

リターン

デバイス名を含む文字列

コマンドがエラーになると例外になります。

例

IDCODE の 01010101010101010101010101010101 に含まれるデバイス名をデータベースで検索します。

```
%set deviceName [::chipscope::csejtag_db get_device_name_for_idcode
$handle “01010101010101010101010101010101”]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_db get_irlength_for_idcode

このサブコマンドは、デバイスの IDCODE を使用してデータベースからデバイスの IR 長を取得するために使用します。

構文

```
::chipscope::csejtag_db get_irlength_for_idcode handle idcode
```

引数

表 5-43 : ::chipscope::csejtag_db get_irlength_for_idcode サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
idcode		該当デバイスの IDCODE

リターン

IR のサイズ (ビット) を含む文字列

コマンドがエラーになると例外になります。

例

IDCODE の 01010101010101010101010101010101 に含まれる IR 長をデータベースで検索します。

```
%set irlen [::chipscope::csejtag_db get_irlength_for_idcode $handle
“01010101010101010101010101010101”]
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_db parse_bsdl

このサブコマンドは、バウンダリスキャン記述言語 (BSDL) バッファーからデバイス情報を抽出するために使用します。

構文

```
::chipscope::csejtag_db parse_bsdl handle filename buf bufLen
```

引数

表 5-44 : Arguments for Subcommand ::chipscope::csejtag_db parse_bsdl

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
filename		ローカル BSDL ファイルのファイル名 (デバッグの場合のみ)
buf		BSDL ファイル全体の内容を含むバッファー
bufLen		バッファー buf のサイズ (バイトまたは文字数)

リターン

フォーマットのリストは、次のようになります。

```
{deviceName irlen idcode cmd_bypass}
```

説明 :

deviceName
: デバイス名を含む文字列

irlen
: デバイスの IR のビット数

idcode
: デバイスの IDCODE

cmd_bypass
: デバイスの BYPASS 命令を含む文字列 (通常すべて 1)

コマンドがエラーになると例外になります。

例

bsdl_bufLen サイズの bsdl_buf バッファーの device.bsd ファイルからデバイス情報を抽出します。

```
%::chipscope::csejtag_db parse_bsdl $handle “device.bsd” $bsdl_buf
$bsdl_bufLen
```

CseJtag Tcl コマンドのリストに戻る

::chipscope::csejtag_db parse_bsdl_file

このサブコマンドは、バウンダリスキャン記述言語 (BSDL) ファイルからデバイス情報を抽出するために使用します。

構文

```
::chipscope::csejtag_db parse_bsdl_file handle filename
```

引数

表 5-45 : ::chipscope::csejtag_db parse_bsdl_file サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
filename		ローカル BSDL ファイルのファイル名

リターン

フォーマットのリストは、次のようになります。

```
{deviceName irlen idcode cmd_bypass}
```

説明 :

deviceName
: デバイス名を含む文字列

irlen
: デバイスの IR のビット数

idcode
: デバイスの IDCODE

cmd_bypass
: デバイスの BYPASS 命令を含む文字列 (通常すべて 1)

コマンドがエラーになると例外になります。

例

device.bsd ファイルからデバイス情報を抽出します。

```
%::chipscope::csejtag_db parse_bsdl_file $handle “device.bsd”
```

CseJtag Tcl コマンドのリストに戻る

CseFpga コマンド

ここでは、次の CseFpga コマンドの詳細について説明します。

- ::chipscope::csefpga_configure_device
- ::chipscope::csefpga_configure_device_with_file
- ::chipscope::csefpga_get_config_reg
- ::chipscope::csefpga_get_instruction_reg
- ::chipscope::csefpga_get_usercode
- ::chipscope::csefpga_get_user_chain_count
- ::chipscope::csefpga_is_config_supported
- ::chipscope::csefpga_is_configured
- ::chipscope::csefpga_is_sys_mon_supported
- ::chipscope::csefpga_run_sys_mon_command_sequence
- ::chipscope::csefpga_get_sys_mon_reg
- ::chipscope::csefpga_set_sys_mon_reg

::chipscope::csefpga_configure_device

.bit、.rbt、または .mcs ファイルの内容を含むバイト アレイで FPGA デバイスをコンフィギュレーションします。

構文

```
::chipscope::csefpga_configure_device handle deviceIndex format
fileData fileDataByteLen progressFunc<optional args>
```

引数

表 5-46 : ::chipscope::csefpga_configure_device サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex	必須	*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
format	必須	コンフィギュレーション ファイルの形式。bit、rbt、mcs から選択します。
fileData	必須	バイト アレイのコンフィギュレーション ファイルの内容。bit ファイルはバイナリ モードで読み出す必要があります。ほかのフォーマットはバイナリ モードまたはテキスト モードで読み出すことができます。fileData からは Windows/Unix の行末文字を削除しないようにしてください。
fileDataByteLen	必須	fileData バイト アレイの長さ (バイト)
<optional args>	オプション	その他のデバイス コンフィギュレーション オプションをオンにします。コンフィギュレーション オプションのリストは表 5-47 を参照してください。

progressFunc	必須	コンフィギュレーション データをデバイスにシフトしている間の進捗状況を示す関数。この関数の形式は、次のようになります。
		`proc progressFunc (handle totalCount`
		`CurrentCount progressStatus) {...}`
		プログレス コールバック関数は $CSE_STOP または $CSE_CONTINUE をリターンします。プログレス コールバック関数が必要でない場合は、引数に 0 を使用します。

表 5-47 : コンフィギュレーション オプション

オプション名	値	説明
reset_device	reset_device=true, reset_device=false	デバイスがコンフィギュレーション中にリセットされるかどうかを制御します。デフォルトは reset_device=true です。
shutdown_sequence	shutdown_sequence=true, shutdown_sequence=false	JTAG SHUTDOWN コマンドを使用して、デバイスをシャットダウンします。Spartan®-3 および Spartan-6 FPGA デバイスの場合、このオプションを使用すると、reset_device=true を設定したのと同じ結果になります。デフォルトは shutdown_sequence=false です。
verify_internal_done	verify_internal_done=true, verify_internal_done=false	コンフィギュレーション後にデバイスの JTAG 命令レジスタからの内部デバイスの DONE ステータスを読み出します。デフォルトは verify_internal_done=true です。
verify_external_done	verify_external_done=true, verify_external_done=false	コンフィギュレーション後にコンフィギュレーション ステータス レジスタから外部デバイスの DONE ステータスを読み込みます。DONE デバイス パッケージ ピン同士を接続すると、DONE ステータスは接続されたすべてのデバイスの DONE ピンの論理 AND の結果になります。デフォルトは verify_external_done=false です。
verify_crc	verify_crc=true, verify_crc=false	コンフィギュレーション後にコンフィギュレーション ステータス レジスタから CRC ステータスを読み込みます。デフォルトは verify_crc=false です。
use_assigned_config_data	use_assigned_config_data=true, use_assigned_config_data=false	::chipscope::csefpga_assign_config_data_to_device または ::chipscope::csefpga_assign_config_data_file_to_device で提供されるコンフィギュレーションおよびマスク データを使用します。デフォルトは use_assigned_config_data=false です。

リターン

コンフィギュレーションの結果ステータスを含むビットフィールドが表示されます。ビットフィールドでは、次の値の 1 つまたは複数を使用する論理 AND を介すことで、対応するステータス情報を確認できます。

```
$CSE_INTERNAL_DONE_HIGH_STATUS
$CSE_EXTERNAL_DONE_HIGH_STATUS
$CSE_CRC_ERROR_STATUS
$CSE_BITSTREAM_READ_ENABLED
$CSE_BITSTREAM_WRITE_ENABLED
```

コマンドがエラーになると例外になります。

例

mydesign.bit ファイルを使用して JTAG チェーンの 3 つ目のデバイスをコンフィギュレーションします。

```
%set filename "mydesign.bit"
%set fp [open $filename r]
```

```
%fconfigure $fp -translation binary -blocking 1
%set fileData [read $fp]
%close $fp
%set configStatus [::chipscope::csefpga_configure_device $handle 2
"bit" $CSE_DEFAULT_OPTIONS $fileData [file size $filename]
"progressCallBack"]
```

CseFpga コマンドのリストに戻る

::chipscope::csefpga_configure_device_with_file

.bit、.rbt、または .mcs ファイルの内容で FPGA デバイスをコンフィギュレーションします。

構文

```
::chipscope::csefpga_configure_device_with_file handle deviceIndex
filename <optional args> progressFunc
```

引数

表 5-48 : ::chipscope::csefpga_configure_device_with_file サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex	必須	*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
filename	必須	bit、rbt、または mcs コンフィギュレーション ファイルのファイル名
<optional args>	オプション	その他のデバイス コンフィギュレーション オプションをオンにします。コンフィギュレーション オプションのリストは表 5-47 を参照してください。

progressFunc	必須	コンフィギュレーション データをデバイスにシフトしている間の進捗状況を示す関数。この関数の形式は、次のようになります。
		`proc progressFunc (handle totalCount`
		`CurrentCount progressStatus) {...}`
		プログレス コールバック関数は $CSE_STOP または $CSE_CONTINUE をリターンします。プログレス コールバック関数が必要でない場合は、引数に 0 を使用します。

リターン

コンフィギュレーションの結果ステータスを含むビットフィールドが表示されます。ビットフィールドでは、次の値の 1 つまたは複数を使用する論理 AND を介すことで、対応するステータス情報を確認できます。

```
$CSE_INTERNAL_DONE_HIGH_STATUS
$CSE_EXTERNAL_DONE_HIGH_STATUS
$CSE_CRC_ERROR_STATUS
$CSE_BITSTREAM_READ_ENABLED
$CSE_BITSTREAM_WRITE_ENABLED
```

コマンドがエラーになると例外になります。

例

mydesign.bit ファイルを使用して JTAG チェーンの 3 つ目のデバイスをコンフィギュレーションします。

```
%set fileName "mydesign.bit"
```

```
%set configStatus [::chipscope::csefpga_configure_device $handle 2
$fileName $CSE_DEFAULT_OPTIONS "progressCallBack"]
```

CseFpga コマンドのリストに戻る

::chipscope::csefpga_get_config_reg

ターゲット FPGA デバイスのコンフィギュレーション レジスタ ビットを読み出します。

構文

```
::chipscope::csefpga_get_config_reg handle deviceIndex bitCount
```

引数

表 5-49 : ::chipscope::csefpga_get_config_reg サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
bitCount		コンフィギュレーション レジスタの長さ (ビット)

リターン

フォーマットのリストは、次のようになります。

```
{hexReg bitNameBuf}
```

説明 :

`hexReg`

レジスタ値を含む文字列 (16 進数)

`bitNameBuf`

コンフィギュレーション レジスタ ビット名を示す文字列のカンマ区切りのリスト

コマンドがエラーになると例外になります。

例

JTAG チェーンの 3 つ目のデバイスのコンフィギュレーション レジスタのコンテンツを読み出します。

```
%set ConfigReg [csefpga_get_config_reg $handle 2 $DeviceBitCount]
```

CseFpga コマンドのリストに戻る

::chipscope::csefpga_get_instruction_reg

ターゲット FPGA デバイスの命令レジスタを読み出して、コンフィギュレーション特有のステータス ビットをフォーマットします。

構文

```
::chipscope::csefpga_get_instruction_reg handle deviceIndex bitCount
```

引数

表 5-50 : ::chipscope::csefpga_get_instruction_reg サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
bitCount		コンフィギュレーション レジスタの長さ (ビット)

リターン

フォーマットのリストは、次のようになります。

```
{hexReg bitNameBuf}
```

説明 :

`hexReg`

レジスタ値を含む文字列 (16 進数)

`bitNameBuf`

コンフィギュレーション レジスタ ビット名を示す文字列のカンマ区切りのリスト

コマンドがエラーになると例外になります。

例

JTAG チェーンの 3 つ目のデバイスのコンフィギュレーション レジスタのコンテンツを読み出します。

```
%set InstReg [csefpga_get_instruction_reg $handle 2 $DeviceBitCount]
```

CseFpga コマンドのリストに戻る

::chipscope::csefpga_get_usercode

ターゲット FPGA デバイスの USERCODE レジスタを読み出します。

構文

```
::chipscope::csefpga_get_usercode handle deviceIndex
```

引数

表 5-51 : ::chipscope::csefpga_get_usercode サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)

リターン

USERCODE レジスタの内容 (16 進数)

コマンドがエラーになると例外になります。

例

JTAG チェーンの 3 つ目のデバイスの USERCODE レジスタのコンテンツを読み出します。

```
%set usercode [csefpga_get_usercode $handle 2]
```

CseFpga コマンドのリストに戻る

::chipscope::csefpga_get_user_chain_count

ターゲット FPGA デバイスの USER チェーン レジスタを決定します。

構文

```
::chipscope::csefpga_get_user_chain_count handle idcode
```

引数

表 5-52 : ::chipscope::csefpga_get_user_chain_count サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
idcode		該当デバイスの IDCODE

リターン

デバイスの USER スキャン チェーンのレジスタ数 (USER スキャン チェーン レジスタがデバイスに含まれない場合は 0)

コマンドがエラーになると例外になります。

例

$idcode で指定されたデバイスでサポートされる USER スキャン チェーン レジスタの数を取得します。

```
%set numUserRegs [csefpga_get_user_chain_count $handle $idcode]
```

CseFpga コマンドのリストに戻る

::chipscope::csefpga_is_config_supported

ターゲット FPGA デバイスでコンフィギュレーションがサポートされるかどうかテストします。

構文

```
::chipscope::csefpga_is_config_supported handle idcode
```

引数

表 5-53 : ::chipscope::csefpga_is_config_supported サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
idcode		該当デバイスの IDCODE

リターン

idcodeで指定されたデバイスのコンフィギュレーションが csefpga_configure_device コマンドでサポートされる場合は 1、されない場合は 0。

コマンドがエラーになると例外になります。

例

$idcode で指定されたデバイスがコンフィギュレーション可能かどうかを決定します。

```
%set isConfigurable [csefpga_is_config_supported $handle $idcode]
```

CseFpga コマンドのリストに戻る

::chipscope::csefpga_is_configured

FPGA デバイスのコンフィギュレーション ステータスをリターンします。

構文

```
::chipscope::csefpga_is_configured handle deviceIndex
```

引数

表 5-54 : ::chipscope::csefpga_is_configured サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)

リターン

deviceIndex で指定されるデバイスがコンフィギュレーションされる場合は 1、されない場合は 0。

コマンドがエラーになると例外になります。

例

JTAG チェーン内の 3 つ目のデバイスのコンフィギュレーション ステータスを取得します。

```
%set isConfigured [csefpga_is_configured $handle 2]
```

CseFpga コマンドのリストに戻る

::chipscope::csefpga_is_sys_mon_supported

ターゲット FPGA デバイスでシステム モニターのコマンドがサポートされるかどうかテストします。

構文

```
::chipscope::csefpga_is_sys_mon_supported handle idcode
```

引数

表 5-55 : ::chipscope::csefpga_is_sys_mon_supported サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
idcode		該当デバイスの IDCODE

リターン

idcode で指定されるデバイスにシステム モニター ブロックが含まれる場合は 1、されない場合は 0。

コマンドがエラーになると例外になります。

例

$idcode で指定されたデバイスにシステム モニター ブロックが含まれているかを決定します。

```
%set hasSysMon [csefpga_is_sys_mon_supported $handle $idcode]
```

CseFpga コマンドのリストに戻る

::chipscope::csefpga_run_sys_mon_command_sequence

システム モニターのレジスタの読み出しと書き込みのシーケンスを実行します。

構文

```
::chipscope::csefpga_run_sys_mon_command_sequence handle deviceIndex
[list hexAddresses...][list hexInData...][list setModes...]
commandCount
```

引数

表 5-56 : ::chipscope::csefpga_run_sys_mon_command_sequence サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
[list hexAddresses...]		アクセスするシステム モニターの DRP レジスタ アドレスのリスト。このリストのエレメント数は commandCount で検出されます。
[list hexInData...]		hexAddresses のリストで指定されるシステム モニターの DRP レジスタに記述されるデータのリスト。hexInDat エレメントは対応する setModes エレメントが 0 以外の場合にのみ書き込まれます。このリストのエレメント数は commandCount で検出されます。
[list setModes...]		setMode フラグのリスト。0 の場合はレジスタからの読み出しデータ、0 以外の場合は書き込みデータを示します。このリストのエレメント数は commandCount で検出されます。
commandCount		コマンドの数 (および各リスト引数のエレメント数)

リターン

それぞれがレジスタの読み出し値を表す commandCount エレメントを含むリスト。対応する setModes エレメントが 0 であれば、データ エレメントは有効です。

コマンドがエラーになると例外になります。

例

JTAG チェーンの 2 つ目のデバイスで、システム モニターの DRP レジスタのアドレス 0x10 に 0x55AA を書き込み、DRP レジスタのアドレス 0x11 から読み出します。

```
%set hexOutData [csefpga_run_sys_mon_command_sequence $handle 1 [list
10 11] [list 55AA 0000] [list 1 0] 2]
```

CseFpga コマンドのリストに戻る

::chipscope::csefpga_get_sys_mon_reg

システム モニターのレジスタから読み出します。

構文

```
::chipscope::csefpga_get_sys_mon_reg handle deviceIndex hexAddress
```

引数

表 5-57 : ::chipscope::csefpga_get_sys_mon_reg サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
hexAddress		読み出すシステム モニターの DRP レジスタのアドレス

リターン

システム モニターの DRP レジスタのアドレス hexAddress から読み出したデータ値 (16 進数)。

コマンドがエラーになると例外になります。

例

JTAG チェーンの 2 つ目のデバイスで System Monitor レジスタのアドレス 0x07 からデータを読み出します。

```
%set hexOutData [csefpga_get_sys_mon_reg $handle 1 7]
```

CseFpga コマンドのリストに戻る

::chipscope::csefpga_set_sys_mon_reg

システム モニターのレジスタに書き込みます。

構文

```
::chipscope::csefpga_set_sys_mon_reg handle deviceIndex hexAddress
hexInData
```

引数

表 5-58 : ::chipscope::csefpga_set_sys_mon_reg サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
hexAddress		書き込むシステム モニターの DRP レジスタのアドレス (16 進数)
hexInData		システム モニターの DRP レジスタに書き込むデータ値 (16 進数)

リターン

コマンドがエラーになると例外になります。

例

JTAG チェーンの 2 つ目のデバイスでシステム モニターのレジスタのアドレス 0x09 に 0xABCD を書き込みます。

```
%csefpga_set_sys_mon_reg $handle 1 9 abcd
```

CseFpga コマンドのリストに戻る

CseCore コマンド

ここでは、次の CseFpga コマンドの詳細について説明します。

- ::chipscope::csecore_get_core_count
- ::chipscope::csecore_get_core_status
- ::chipscope::csecore_is_cores_supported

::chipscope::csecore_get_core_count

ターゲット FPGA デバイスの ICON コアと特定の USER スキャン チェーン レジスタに接続されたコアの数を取得します。

構文

```
::chipscope::csecore_get_core_count handle deviceIndex userRegNumber
```

引数

表 5-59 : ::chipscope::csecore_get_core_count サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
deviceIndex		*n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1)
userRegNumber		BSCAN ブロックの USER レジスタ番号 (1 から開始)

リターン

コアの番号。

コマンドがエラーになると例外になります。

例

JTAG チェーンの 3 つ目のデバイスの USER3 レジスタの ICON コアのコア番号を取得します。

```
%set coreCount [csecore_get_core_count $handle 2 3]
```

CseCore コマンドのリストに戻る

::chipscope::csecore_get_core_status

ターゲット ChipScope Pro コアからスタティック ステータス ワードを読み出します。

構文

```
::chipscope::csecore_get_core_status handle [list deviceIndex
userRegNumber coreIndex] bitCount
```

引数

表 5-60 : ::chipscope::csecore_get_core_status サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
[list deviceIndex userRegNumber coreIndex]		次の 3 つのエレメントを含むリスト : ・ *n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1) ・ BSCAN ブロックの USER レジスタ番号 (1 から開始) ・ コア ユニットのインデックス。ICON に接続される最初のコア ユニットのインデックスは 0 です。
bitCount		ステータス ワードの長さ (ビット数)

リターン

ネストされたリスト。外部リストには、内部リストとコア ステータスを示す文字列 (16 進数) の 2 つのエレメントが含まれ、内部リストには、次のエレメントが含まれます。

エレメント	説明
manufacturerId	Manufacturer ID (整数)
coreType	コア タイプ (整数)
coreMajorVersion	コアのメジャー バージョン (整数)
coreMinorVersion	コアのマイナー バージョン (整数)
coreRevision	コアのリビジョン (整数)

注記 : コマンドがエラーになると例外になります。

例

2 つ目の USER レジスタの 4 つ目のデバイスの中の ICON コアに接続される最初のコアのステータスを取得します。

```
%set coreRef [list 3 2 0]
%set coreStatus [csecore_get_core_status $handle $coreRef]
```

CseCore コマンドのリストに戻る

::chipscope::csecore_is_cores_supported

ターゲット FPGA デバイスで ChipScope Pro コアがサポートされるかどうかテストします。

構文

```
::chipscope::csecore_is_cores_supported handle idcode
```

引数

表 5-61 : ::chipscope::csecore_is_cores_supported サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
idcode		該当デバイスの IDCODE

リターン

idcode で指定されるデバイスで ChipScope Pro コアがサポートされる場合は 1、されない場合は 0。

コマンドがエラーになると例外になります。

例

$idcode で指定されたデバイスで ChipScope Pro コアがサポートされるかどうかを決定します。

```
%set supportsCores [csecore_is_cores_supported $handle $idcode]
```

CseCore コマンドのリストに戻る

CseVIO コマンド

ここでは、次の CseFpga コマンドの詳細について説明します。

- ::chipscope::csevio_get_core_info
- ::chipscope::csevio_is_vio_core
- ::chipscope::csevio_init_core
- ::chipscope::csevio_terminate_core
- ::chipscope::csevio_define_signal
- ::chipscope::csevio_define_bus
- ::chipscope::csevio_undefine_name
- ::chipscope::csevio_write_values
- ::chipscope::csevio_read_values

::chipscope::csevio_get_core_info

ターゲット VIO コアからスタティック ステータス ワードを読み出します。

構文

```
::chipscope::csevio_get_core_info handle [list deviceIndex
userRegNumber coreIndex] coreInfoTclArray
```

引数

表 5-62 : ::chipscope::csevio_get_core_info サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
[list deviceIndex userRegNumber coreIndex]		次の 3 つのエレメントを含むリスト： · *n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1) · BSCAN ブロックの USER レジスタ番号 (1 から開始) · コア ユニットのインデックス。ICON に接続される最初のコア ユニットのインデックスは 0 です。
coreInfoTclArray		Tcl アレイ名。このコマンドが問題なく実行されると、アレイに次の「リターン」セクションに記述される情報が含まれます。

リターン

次のエレメントを含み、coreInfoTclArray 引数からコア情報をリターンします。

エレメント	説明
manufacturerId	Manufacturer ID (整数)
$CSEVIO_MANUFACTURER_ID	製造元の ID。ザイリンクス社の場合は 1
$CSEVIO_CORE_TYPE	コア タイプ フィールド。各 ChipScope コアで異なり、VIO の場合は 9
$CSEVIO_CORE_MAJOR_VERSION	メジャー リリース バージョン
$CSEVIO_CORE_MINOR_VERSION	マイナー リリース バージョン
$CSEVIO_CORE_REVISION	リビジョン
$CSEVIO_CG_MAJOR_VERSION	CoreGen 用フィールド (10.1 以降で生成されたコアの場合)
$CSEVIO_CG_MINOR_VERSION	CoreGen 用フィールド (10.1 以降で生成されたコアの場合)
$CSEVIO_CG_MINOR_VERSION_ALPHA	CoreGen 用フィールド (10.1 以降で生成されたコアの場合)
$CSEVIO_ASYNC_INPUT_COUNT	使用される非同期入力信号の数
$CSEVIO_SYNC_INPUT_COUNT	使用される同期入力信号の数
$CSEVIO_ASYNC_OUTPUT_COUNT	使用される非同期出力信号の数
$CSEVIO_SYNC_OUTPUT_COUNT	使用される同期出力信号の数

注記 : コマンドがエラーになると例外になります。

例

2 つ目の USER レジスタの 4 つ目のデバイスの中の ICON コアの最初の制御ポートに接続される VIO コアのコア情報を取得します。VIO コアの非同期入力信号の数を表記します。

```
%set coreRef [list 3 2 0]
%csevio_get_core_info $handle $coreRef coreInfoTclArray
%puts stdout ・coreInfoTclArray($CSEVIO_ASYNC_INPUT_COUNT)
```

CseVIO コマンドのリストに戻る

::chipscope::csevio_is_vio_core

ターゲット コアが VIO コアかどうか判別します。

構文

```
::chipscope::csevio_is_vio_core handle [list deviceIndex userRegNumber coreIndex]
```

引数

表 5-63 : ::chipscope::csevio_is_vio_core サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
[list deviceIndex userRegNumber coreIndex]		次の 3 つのエレメントを含むリスト : ・ *n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1) ・ BSCAN ブロックの USER レジスタ番号 (1 から開始) ・ コア ユニットのインデックス。ICON に接続される最初のコア ユニットのインデックスは 0 です。

リターン

コアが VIO コアの場合は 1、それ以外の場合は 0。

コマンドがエラーになると例外になります。

例

2 つ目の USER レジスタの 4 つ目のデバイスの中の ICON コアに接続される最初のコアが VIO コアかどうかを決定します。

```
%set coreRef [list 3 2 0]
%set isVIO [csevio_is_vio_core $handle $coreRef]
```

CseVIO コマンドのリストに戻る

::chipscope::csevio_init_core

ターゲット VIO コアに関連するグローバル変数を初期化します。

構文

```
::chipscope::csevio_init_core handle [list deviceIndex userRegNumber coreIndex]
```

引数

表 5-64 : ::chipscope::csevio_init_core サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
[list deviceIndex userRegNumber coreIndex]		次の 3 つのエレメントを含むリスト : ・ *n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1) ・ BSCAN ブロックの USER レジスタ番号 (1 から開始) ・ コア ユニットのインデックス。ICON に接続される最初のコア ユニットのインデックスは 0 です。

リターン

コマンドがエラーになると例外になります。

例

2 つ目の USER レジスタの 4 つ目のデバイスの中の ICON コアに接続された VIO コアを初期化します。

```
%set coreRef [list 3 2 0]
%csevio_init_core $handle $coreRef
```

CseVIO コマンドのリストに戻る

::chipscope::csevio_terminate_core

ターゲット VIO コアに関連するグローバル変数を削除してメモリを解放します。

構文

```
::chipscope::csevio_terminate_core handle [list deviceIndex
userRegNumber coreIndex]
```

引数

表 5-65 : ::chipscope::csevio_terminate_core サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
[list deviceIndex userRegNumber coreIndex]		次の 3 つのエレメントを含むリスト : ・ *n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1) ・ BSCAN ブロックの USER レジスタ番号 (1 から開始) ・ コア ユニットのインデックス。ICON に接続される最初のコア ユニットのインデックスは 0 です。

リターン

コマンドがエラーになると例外になります。

例

2 つ目の USER レジスタの 4 つ目のデバイスの中の ICON コアに接続された VIO コアを終了します。

```
%set coreRef [list 3 2 0]
%csevio_terminate_core $handle $coreRef
```

CseVIO コマンドのリストに戻る

::chipscope::csevio_define_signal

指定した VIO 信号ビットの名前を定義します。csevio_init_coreが最初に呼び出される必要があります。

構文

```
::chipscope::csevio_define_signal handle [list deviceIndex
userRegNumber coreIndex] name flags bitIndex
```

引数

表 5-66 : ::chipscope::csevio_define_signal サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
[list deviceIndex userRegNumber coreIndex]		次の 3 つのエレメントを含むリスト : ・ *n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1) ・ BSCAN ブロックの USER レジスタ番号 (1 から開始) ・ コア ユニットのインデックス。ICON に接続される最初のコア ユニットのインデックスは 0 です。
name		信号に指定する名前。すべての入力信号名はほかの入力信号と異なっている必要があり、出力信号名もすべてほかの出力信号と異なっている必要があります。
flags		信号のポート タイプを決定するために使用されるフラグ。フラグ値は次のいずれかになります。 ・ $CSEVIO_SYNC_OUTPUT ・ $CSEVIO_SYNC_INPUT ・ $CSEVIO_ASYNC_OUTPUT ・ $CSEVIO_ASYNC_INPUT
bitIndex		ポートへのビット インデックス。どのポート信号を name に割り当てるか決定するために使用します。

リターン

コマンドがエラーになると例外になります。

例

2 つ目の USER レジスタの 4 つ目のデバイスの中の ICON コアに接続された VIO コアで、ASYNC_INPUT ポートのビット 0 に割り当てられた status_bit という信号を定義します。

```
%set coreRef [list 3 2 0]
%set csevio_define_signal $handle $coreRef “status_bit”
$CSEVIO_ASYNC_INPUT 0
```

CseVIO コマンドのリストに戻る

::chipscope::csevio_define_bus

VIO 信号ビットのグループ (バス) 名を定義します。csevio_init_coreが最初に呼び出される必要があります。

構文

```
::chipscope::csevio_define_bus handle [list deviceIndex userRegNumber coreIndex] name flags bitIndexArray arrayLen
```

引数

表 5-67 : ::chipscope::csevio_define_bus サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
[list deviceIndex userRegNumber coreIndex]		次の 3 つのエレメントを含むリスト : ・ *n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1) ・ BSCAN ブロックの USER レジスタ番号 (1 から開始) ・ コア ユニットのインデックス。ICON に接続される最初のコア ユニットのインデックスは 0 です。
name		バスに指定する名前。 すべての入力バス名はほかの入力バスと異なっている必要があり、出力バス名もすべてほかの出力バスと異なっている必要があります。
flags		バスのポート タイプを決定するために使用されるフラグ。フラグ値は次のいずれかになります。 ・ $CSEVIO_SYNC_OUTPUT ・ $CSEVIO_SYNC_INPUT ・ $CSEVIO_ASYNC_OUTPUT ・ $CSEVIO_ASYNC_INPUT
[list bitIndicies...]		バスの信号のビット インデックスを含むリスト。リストの一番左のエレメントが LSB です。

リターン

コマンドがエラーになると例外になります。

例

2 つ目の USER レジスタの 4 つ目のデバイスの中の ICON コアに接続された VIO コアで、SYNC_OUTPUT ポートのビット 3:0 に割り当てられた control_bus というバスを定義します。

```
%set coreRef [list 3 2 0]
%set csevio_define_bus $handle $coreRef “control_bus”
$CSEVIO_SYNC_OUTPUT [list 0 1 2 3]
```

CseVIO コマンドのリストに戻る

::chipscope::csevio_undefine_name

VIO 信号/バス名と関連するすべての情報を削除します。

構文

```
::chipscope::csevio_undefine_name handle [list deviceIndex
userRegNumber coreIndex] name flags
```

引数

表 5-68 : ::chipscope::csevio_undefine_name サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
[list deviceIndex userRegNumber coreIndex]		次の 3 つのエレメントを含むリスト : ・ *n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1) ・ BSCAN ブロックの USER レジスタ番号 (1 から開始) ・ コア ユニットのインデックス。ICON に接続される最初のコア ユニットのインデックスは 0 です。
name		削除する信号またはバスの名前
flags		信号またはバスのポート タイプを決定するために使用されるフラグ。フラグ値は次のいずれかになります。 ・ $CSEVIO_SYNC_OUTPUT ・ $CSEVIO_SYNC_INPUT ・ $CSEVIO_ASYNC_OUTPUT ・ $CSEVIO_ASYNC_INPUT

リターン

コマンドがエラーになると例外になります。

例

2 つ目の USER レジスタの 4 つ目のデバイスの中の ICON コアに接続された VIO コアで、SYNC_OUTPUT ポートのビットに割り当てられた control_bus というバスの定義を解除します。

```
%set coreRef [list 3 2 0]
%set csevio_undefine_name $handle $coreRef “control_bus”
$CSEVIO_SYNC_OUTPUT
```

CseVIO コマンドのリストに戻る

::chipscope::csevio_write_values

ターゲット VIO コアの指定した信号/バスに値を書き込みます。

構文

```
::chipscope::csevio_write_values handle [list deviceIndex
userRegNumber coreIndex] outputTclArray
```

引数

表 5-69 : ::chipscope::csevio_write_values サブコマンドの引数

引数	タイプ	説明
handle	必須	::chipscope::csejtag_session create でリターンされたセッションへのハンドル
[list deviceIndex userRegNumber coreIndex]		次の 3 つのエレメントを含むリスト： ・ *n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1) ・ BSCAN ブロックの USER レジスタ番号 (1 から開始) ・ コア ユニットのインデックス。ICON に接続される最初のコア ユニットのインデックスは 0 です。
outputTclArray		Tcl アレイの名前。アレイへのインデックスは csevio_define_signal で定義された出力信号の名前、または csevio_define_bus で定義されたバスの名前です。SYNC_OUTPUT ポートの信号の名前には .pulsetrain を末尾に付けられるほか、文字列の 16 進数値が指定できます (LSB は一番右の文字)。パルス列を使用するには、最初の値が右側へ送信され、16 の値が渡される必要があります。値はそれぞれバイト アライメントされる必要があります。つまり、1 つの信号は値ごとに 2 つの 16 進数文字を必要とします。値が 8 ビットを超える場合は、値ごとに 2 つ文字を追加する必要があります。アレイを再利用するためにこのコマンドを呼び出した後は、アレイの各エレメントの設定を手動で解除する必要があります。

リターン

コマンドがエラーになると例外になります。

例

例では、次が前提とされています。

- VIO コアには既に coreRef が設定済みです。
- reset という信号が SYNC_OUTPUT ポートのビットとして定義されています。
- instruction というバスが ASYNC_OUTPUT ポートの 8 ビット バスとして定義されています。

1. reset 信号を 0 に、instruction バスをフリップフロップ (FF) に設定します。

```
%set outputTclArray(reset) 0
%set outputTclArray(instruction) FF
%csevio_write_values $handle $coreRef outputTclArray
```

2. 0 の後に 1 つのクロック サイクル パルスの 1 を reset 信号に送信します。

```
%set outputTclArray(reset.pulsetrain) 00000000000000000000000000000001
%csevio_write_values $handle $coreRef outputTclArray
```

CseVIO コマンドのリストに戻る

::chipscope::csevio_read_values

ターゲット VIO コアの指定した信号/バスから値を読み出します。

構文

```
::chipscope::csevio_read_values handle [list deviceIndex userRegNumber
coreIndex] inputTclArray
```

引数

表 5-70 : ::chipscope::csevio_read_values サブコマンドの引数

引数	タイプ	説明
handle		::chipscope::csejtag_session create でリターンされたセッションへのハンドル
[list deviceIndex userRegNumber coreIndex]		次の 3 つのエレメントを含むリスト : ・ *n*-length の JTAG チェーンのデバイス インデックス (0 ～ *n*-1) ・ BSCAN ブロックの USER レジスタ番号 (1 から開始) ・ コア ユニットのインデックス。ICON に接続される最初のコア ユニットのインデックスは 0 です。
inputTclArray	必須	Tcl アレイの名前。アレイへのインデックスは csevio_define_signal で定義された入力信号の名前、または csevio_define_bus で定義された入力バスの名前です。入力信号またはバスのさまざまなステートを指定するには、次のような接尾語を使用します。 ・ .value は信号/バスの値を指定します (接尾語なしと同じ)。 ・ .activity_up は非同期の Low から High の動作を指定します。 ・ .activity_down は非同期の High から Low の動作を指定します。 ・ .sync_activity_up は同期の Low から High の動作を指定します (SYNC_INPUT 信号/バスの場合にのみ有効)。 ・ .sync_activity_down は同期の High から Low の動作を指定します (SYNC_INPUT 信号/バスの場合にのみ有効)。

リターン

コマンドがエラーになると例外になります。

例

例では、次が前提とされています。

- VIO コアには既に coreRef が設定済みです。
- status という信号が SYNC_INPUT ポートのビットとして定義されています。
- data_bus というバスが ASYNC_INPUT ポートの 8 ビット バスとして定義されています。

1. status および data_bus の値を取得して stdout に表示します。

```
%csevio_read_values $handle $coreRef inputTclArray
% puts stdout “status = $inputTclArray(status.value)”
% puts stdout “data_bus = $inputTclArray(data_bus)”
```

2. status のさまざまなステートを取得して stdout に表示します。

```
%csevio_read_values $handle $coreRef inputTclArray
% puts stdout “up = $inputTclArray(status.activity_up)”
% puts stdout “dn = $inputTclArray(status.activity_down)”
% puts stdout “sup = $inputTclArray(status.sync_activity_up)”
% puts stdout “sdn = $inputTclArray(status.sync_activity_down)”
```

CseVIO コマンドのリストに戻る

CSE/Tcl の例

ChipScope Pro をインストールすると、CseJtag Tcl インターフェイスを使用する Tcl スクリプトの例が含まれます。この例では、ザイリンクス パラレル ケーブルまたはザイリンクス プラットフォーム USB ケーブルを開いて JTAC チェーンをスキャンし、チェーンで検出されたデバイスに関する情報をリターンします。この例のスクリプトは、次のディレクトリに含まれます。

```
<XILINX_ISE_INSTALL>\cse\tcl\csejtag_example1.tcl
```

このスクリプトは、ザイリンクス ISE Design Suite ソフトウェアに含まれる Tcl シェル (xtclsh) または ActiveState Software 社の ActiveTcl 8.4 Tcl シェル (tclsh) で実行できます ([212 ページのリファレンス 24 を参照])。コマンド ライン シェルで Tcl 例を実行するには、csejtag_example1.tcl のあるディレクトリ (上記参照) に変更し、OS 別の次の手順に従ってください。

- 32 ビット Windows の場合 :
  - ザイリンクス パラレル ケーブルを使用するには、次を入力します。

    ```
    <XILINX_ISE_INSTALL>\bin\nt\xtclsh csejtag_example1.tcl -par
    ```

  - ザイリンクス プラットフォーム ケーブル USB を使用するには、次を入力します。

    ```
    <XILINX_ISE_INSTALL>\bin\nt\xtclsh csejtag_example1.tcl -usb
    ```

- 64 ビット Windows の場合 :
  - ザイリンクス パラレル ケーブルを使用するには、次を入力します。

    ```
    <XILINX_ISE_INSTALL>\bin\nt64\xtclsh csejtag_example1.tcl -par
    ```

  - ザイリンクス プラットフォーム ケーブル USB を使用するには、次を入力します。

    ```
    <XILINX_ISE_INSTALL>\bin\nt64\xtclsh csejtag_example1.tcl -usb
    ```

- 32 ビット Linux の場合 :
  - ザイリンクス パラレル ケーブルを使用するには、次を入力します。

    ```
    <XILINX_ISE_INSTALL>/bin/lin/xtclsh csejtag_example1.tcl -par
    ```

  - ザイリンクス プラットフォーム ケーブル USB を使用するには、次を入力します。

    ```
    <XILINX_ISE_INSTALL>/bin/lin/xtclsh csejtag_example1.tcl -usb
    ```

- 64 ビット Linux の場合 :
  - ザイリンクス パラレル ケーブルを使用するには、次を入力します。

    ```
    <XILINX_ISE_INSTALL>/bin/lin64/xtclsh csejtag_example1.tcl -par
    ```

  - ザイリンクス プラットフォーム ケーブル USB を使用するには、次を入力します。

    ```
    <XILINX_ISE_INSTALL>/bin/lin64/xtclsh csejtag_example1.tcl -usb
    ```

その他の Tcl スクリプト例 (csevio_example1.tcl という CSE VIO の Tcl スクリプト例など) は、csejtag_example1.tcl と同じディレクトリにあります。これらのスクリプトは、その他の CSE/Tcl 関数呼び出しの例です。

XILINX®

付録 A

ChipScope Pro ツール トラブルシューティング ガイド

概要

この付録では、ChipScope™ Pro ツールのインストールおよび ISE® ソフトウェアとの統合に関するトラブルシュート方法を説明します。ChipScope Pro ツールを使用する際によく発生するエラーや問題について説明するほか、ChipScope Pro およびザイリンクス JTAG ベースのプログラム ケーブル インストールのトラブルシュート方法についても説明します。各問題には、それぞれ次のようなセクションが含まれます。

- 問題：このセクションには、問題がエラーおよび警告メッセージで表示される問題と同じものかどうかを確認する方法がリストされます。
- 回避策：このセクションには、問題の解決方法がリストされます。

ChipScope Pro ツールのインストールに関するトラブルシューティング

表 A-1 は、ChipScope Pro ツールのインストールでよく発生するエラー メッセージや問題についてまとめたものです。

表 A-1 : ChipScope Pro ツールのインストールに関するトラブルシューティング

問題	回避策
Windows または Linux のいずれの場合でも、ISE ツールで filename.cdc ファイルをダブルクリックしたときに ChipScope Pro Core Inserter が起動されず、次のエラー メッセージがコンソールに表示されます。 ERROR: Unable to find the Chipscope Exe at NotHere/inserterlauncher.exe	1. 次の 3 つのパラメーターが正しく設定されていることを確認することで、環境が正しく設定されているかどうかを確認してください。 → CHIPSCOPE 環境変数 : ザイリンクスの ChipScope Pro ツールのインストール ディレクトリに設定する必要があります。 - Windows の場合、[スタート] → [設定] → [コントロール パネル] → [システム] → [詳細設定] タブ → [環境変数] ボタンをクリックします。CHIPSCOPE 環境変数でインストール ディレクトリを指定します。通常、このディレクトリは C:\Xilinx\<version number>\ChipScope です。 - Linux の場合、setenv CHIPSCOPE /tools/xilinx/version number/chipscope のように指定します。 → XILINX 環境変数 : ChipScope Pro ツールが正しく動作するには、ザイリンクスの ISE ツールのインストール ディレクトリに設定する必要があります。 - Windows の場合、[スタート] → [設定] → [コントロール パネル] → [システム] → [詳細設定] タブ → [環境変数] ボタンをクリックします。XILINX 環境変数でインストール ディレクトリを指定します。通常、このディレクトリは C:\Xilinx\<version number>\ISE です。 - Linux の場合、インストールが終了すると、環境変数ファイルが自動的に作成されます。ザイリンクス ISE ソフトウェア インストール ディレクトリに移動し、<settings file> を指定します (settings file は OS およびシェル タイプによって、settings32.sh、settings32.csh、settings64.sh、または settings64.csh になります)。 2. ISE ツールで [Edit] → [Preferences] → [ISE General] → [Integrated Tools] が <install>/bin/<platform> に設定されているかどうか確認します。 → <install> は ChipScope Pro ツールのインストール ディレクトリです。 → <platform> は 32 ビット Linux の場合 lin、64 ビット Linux の場合 lin64、32 ビット Windows の場合 nt、64 ビット Windows の場合 nt64 になります。ChipScope Pro ツールの <platform> は ISE ツールのプラットフォームと同じである必要があります。
Windows の [スタート] メニューから Core Inserter を実行すると、スプラッシュ画面が表示されるのにツールが起動されません。	
Windows の場合、Core Inserter をコマンド ラインから起動すると、次のメッセージを示すポップアップ ダイアログ ボックスが表示されます。 inserter.exe - entry point not found	
Linux の場合、inserter.sh スクリプトを起動して Core Inserter を実行すると、次のようなエラー メッセージが表示されます。 ERROR: Unable to locate Xilinx ISE release number tools in path! ERROR: You must set the CHIPSCOPE environment variable before running this tool	

ザイリンクス JTAG プログラム ケーブルに関するトラブルシューティング

このセクションでは、ザイリンクス JTAG (Joint Test Action Group、IEEE 規格) プログラム ケーブルが正しく接続されているかどうか確認する方法やザイリンクス JTAG ケーブルのよくある接続問題をトラブルシュートする方法について説明します。このセクションで説明される問題は、次のとおりです。

- ケーブルが正しく接続されているか確認する方法については、197 ページの表 A-3 を参照してください。
- 「INFO: Cable connection failed」というメッセージに関する問題のトラブルシューティングについては、197 ページの表 A-4 を参照してください。
- 「ERROR:iMPACT:2246 - A reference voltage has not been detected...」というメッセージに関する問題のトラブルシューティングについては、199 ページの表 A-5 を参照してください。
- 「ERROR: No devices detected while scanning the JTAG chain」、「ERROR: Failed detecting JTAG device chain」、または「ERROR: Opened Xilinx Platform USB Cable but failed to detect JTAG Chain」というメッセージに関する問題のトラブルシューティングについては、200 ページの表 A-6 を参照してください。
- 「ERROR: Socket Open Failed. localhost/127.0.0.1:50001」というメッセージに関する問題のトラブルシューティングについては、202 ページの表 A-7 を参照してください。

表 A-2 : プラットフォーム ケーブル USB が正しく接続されているかの検証

問題	回避策

プラットフォーム ケーブル USB に正しく接続されているかどうか、どうすれば確認できますか。	1. ChipScope Pro Analyzer ツールを起動します。
	2. [JTAG Chain] メニュー オプションを選択します。
	3. [Platform Cable USB] をクリックします。
	4. [Speed] および [Port] オプションが正しく設定されているかどうか確認します。
	5. [OK] をクリックします。
	6. 次のようなメッセージが表示されるかどうかを確認します。
	`COMMAND: open_platform_usb_cable FREQUENCY=3000000 PORT=USB21`
	`INFO: Started ChipScope host (localhost:50001)`
	`INFO: Opened socket connection: localhost 50001 localhost/127.0.0.1`
	`INFO: Connecting to cable (Usb Port - USB21).`
	`INFO: Checking cable driver.`
	`INFO: Driver file xusbdfwu.sys found.`
	`INFO: Driver version: src=1027, dest=1027.`
	`INFO: Driver windrvr6.sys version = 8.1.1.0.`
	`INFO: WinDriver v8.11 Jungo (c) 1997 - 2006 Build Date:Oct 16 2006 X86 32bit SYS 12:35:07, version = 811.`
	`INFO: Cable PID = 0008.`
	`INFO: Max current requested during enumeration is 300 mA.`
	`INFO: Type = 0x0005.`
	`INFO: Cable Type = 3, Revision = 0.`
	`INFO: Setting cable speed to 3 MHz.`
	`INFO: Cable connection established.`
	`INFO: Firmware version = 2301.`
	`INFO: File version of C:/Xilinx/11.1/ChipScope/xilinx/data/xusb_xp2.hex = 2401.`
	`INFO: Firmware hex file version = 2401.`
	`INFO: Downloading C:/Xilinx/11.1/ChipScope/xilinx/data/xusb_xp2.hex.`
	`INFO: Downloaded firmware version = 2401.`
	`INFO: PLD file version = 200Dh.`
	`INFO: PLD version = 200Dh.`

表 A-3 : JTAG パラレル ケーブル IV が正しく接続されているかの検証

問題	回避策

パラレル ケーブル IV に正しく接続されているかどうか、どうすれば確認できますか。	1. ChipScope Pro Analyzer ツールを起動します。
	2. [JTAG Chain] メニュー オプションを選択します。
	3. [Platform Cable USB] をクリックします。
	4. [Speed] および [Port] オプションが正しく設定されているかどうか確認します。
	5. [OK] をクリックします。
	6. 次のようなメッセージが表示されるかどうかを確認します。
	`COMMAND: open_parallel_cable FREQUENCY=5000000 PORT=LPT1`
	`INFO: Started ChipScope host (localhost:50001)`
	`INFO: Opened socket connection: localhost 50001 localhost/127.0.0.1`
	`INFO: Connecting to cable (Parallel Port - LPT1).`
	`INFO: Checking cable driver.`
	`INFO: Driver windrvr6.sys version = 8.1.1.0.`
	`INFO: WinDriver v8.11 Jungo (c) 1997 - 2006 Build Date:Oct 16 2006 X86 32bit SYS 12:35:07, version = 811.`
	`INFO: LPT base address = 0378h.`
	`INFO: ECP base address = 0778h.`
	`INFO: ECP hardware is detected.`
	`INFO: Cable connection established.`
	`INFO: Connecting to cable (Parallel Port - LPT1) in ECP mode.`
	`INFO: Checking cable driver.`
	`INFO: Driver xpc4drvr.sys version = 1.0.4.0.`
	`INFO: LPT base address = 0378h.`
	`INFO: Cable Type = 1, Revision = 10.`
	`INFO: Setting cable speed to 5 MHz.`
	`INFO: Cable connection established.`

表 A-4 : プラットフォーム ケーブル USB の接続に関するトラブルシューティング

問題	回避策
1. ケーブルに接続しようとすると、次のようなメッセージがコンソールに表示されます。 INFO: Cable connection failed. ERROR: Failed to open Xilinx Platform USB Cable.See message(s) above. このメッセージは、ケーブルはインストールされているのに接続されていない場合に表示されます。ケーブルが接続されているのにこのメッセージが表示される場合は、ケーブルが破損しているので、ファームウェアのアップデートが必要です。	まず、ケーブルが接続されているかどうか確認します。 問題 2 に進んでください。
2. ケーブルは適切な USB ポートに挿入されていますか。	いいえ : ケーブルを接続し、ChipScope Pro Analyzer で接続してください。 はい : 問題 3 に進んでください。

表 A-4 : プラットフォーム ケーブル USB の接続に関するトラブルシューティング (続き)

問題	回避策
3. 正しいケーブルが接続時に指定されていましたか。	いいえ/わからない : ChipScope Pro Analyzer ツールの [JTAG Chain] メニューで正しいケーブルが選択されていたかどうか確認します。正しいケーブルを選択し、接続し直します。 はい : 問題 4 に進んでください。
4. サーバー ホスト設定は正しく設定されていますか。	いいえ/わからない : ChipScope Pro Analyzer ツールで [JTAG Chain] → [Server Host Settings] をクリックします。ダイアログ ボックスで正しいサーバー ホスト名 (または IP アドレス) とポートが使用されているかどうか確認します。ローカル システムのケーブルに接続する場合は、localhost:50001 を使用してください。 はい : 問題 5 に進んでください。
5. 特定のプラットフォーム ケーブル USB に接続する場合、正しいポート設定を選択していますか。	いいえ/わからない : ChipScope Pro Analyzer ツールで [JTAG Chain] → [Platform Cable USB] をクリックします。ダイアログ ボックスでポート設定がそのケーブルの正しいポート数に指定されているかどうか確認します。たとえば、最初のケーブルには、ポート USB21 を選択します。 はい : 問題 6 に進んでください。
6. ファームウェア アップデートが必要な可能性があります。	ファームウェアをアップデートするには、次の手順に従います。 1. DOS シェルを開いて、次を入力して環境変数を設定します。 SET XIL_IMPACT_ENV_USB2_FORCE_CPLD_UPDATE=TRUE 2. DOS シェルに「impact」と入力して iMPACT を起動します。 3. [Cable Communication Setup] ダイアログ ボックスでザイリンクス USB ケーブルを選択し、アップデートが完了するのを待ちます。 4. iMPACT を終了します。 5. DOS シェルに次を入力して環境変数を一掃します。 SET XIL_IMPACT_ENV_USB2_FORCE_CPLD_UPDATE= 環境変数の設定方法の詳細と Linux ベースの OS での環境変数設定方法については、(ザイリンクス アンサー 11630) を参照してください。 これで解決しない場合は、ザイリンクス テクニカル サポートで次の情報を含めてケースを開いてください。 • XInfo • cs_analyzer.log

表 A-5 : ケーブルの参照電圧に関するトラブルシューティング

問題	回避策
1. ケーブルに接続しようとすると、ケーブルの LED がオレンジになり (緑にならないで)、次のようなメッセージがコンソールに表示されます。 ERROR: ERROR:iMPACT:2246 - A reference voltage has not been detected on the ribbon cable interface to the target system (pin 2).Check that power is applied to the target system and that the ribbon cable is properly seated at both ends.The status LED on Platform Cable USB is GREEN if target voltage is in the proper range and applied to the correct pin.	この問題は、通常 VCC の電圧が正しくないために発生します。問題 2 に進んでください。
2. ターゲット ボードに電源は投入されていますか。	いいえ : ボードに正しく電源が投入されているかどうか確認します。 はい : 問題 3 に進んでください。
3. リボン ケーブルがターゲット ボード コネクターとプラットフォーム ケーブル USB コネクターにしっかりささっていますか。	いいえ : リボン ケーブルを両方のコネクターに差し直します。 はい : 問題 4 に進んでください。
4. ケーブルの VCC 電圧のレベルは正しいですか。	いいえ/わからない : テスト ツールでボードの電圧を調べて、電圧が正しい範囲内にあるようにします。 はい : 次の情報を含めてザイリンクス テクニカル サポートからケースを開きます。 • XInfo • cs_analyzer.log • 可能であれば、ターゲット ボードのケーブル接続時のケーブルの VCC 電圧レベルのスクリーンショット

表 A-6 : JTAG デバイス検出に関するトラブルシューティング

問題	回避策
1. ケーブルに接続しようとすると、次のようなメッセージがコンソールに表示されます。 ERROR: No devices detected while scanning the JTAG chain ERROR: Failed detecting JTAG device chain ERROR: Opened Xilinx Platform USB Cable but failed to detect JTAG Chain. この問題の原因は、TDI または TDO が未接続か High または Low になった場合に発生する JTAG チェーンの問題であることがよくあります。通常はボードに関連する問題であることを示しています。	問題 2 に進んでください。
2. ケーブル TDI と TDO はケーブル ヘッダーで正しく接続されていますか。	いいえ/わからない : 可能であれば、別のボード、ケーブル、ケーブル コネクターを使用して、エラーの原因を見つけてください。有効なチェーンを含めるように接続を修正します。リボン ケーブルまたはフライ リードを変えると、接続できることもあります。別のボードを使用すると、問題が発生しないこともあります。 はい : 問題 3 に進んでください。
3. ボードのスイッチ ノイズが原因で JTAG チェーンが検出されませんか。	はい/わからない : 可能であれば、ボード レベルのリセットを使用して、ほかのデバイスをリセット ステートにし、JTAG チェーンを検出し直してみます。このリセットをアサートすると、JTAG 操作中のボードのノイズが削減されることがあります。 いいえ : 問題 4 に進んでください。
4. ザイリンクス デバイス以外のデバイスが JTAG チェーン内にありますか。	はい : ザイリンクス デバイス以外のデバイスのアクティブ Low の TRST# ピンが High になっているかどうか確認し、JTAG チェーンを検出し直します。 いいえ : 問題 5 に進んでください。
5. JTAG チェーンに含まれる Virtex®-4、Virtex、Virtex-E、または Spartan®-II/-E デバイスのアクティブ Low の PROG# ピンが Low に保持されていますか。	はい : これらのデバイスの PROG# ピンは必ず High になるようにします。PROG# のパルスが少ないと、これらのデバイスの JTAG TAP コントローラーがリセットされ、チェーン上のすべての動作が実行されなくなります。 いいえ : 問題 6 に進んでください。

表 A-6 : JTAG デバイス検出に関するトラブルシューティング (続き)

問題	回避策
6. JTAG チェーンに含まれるデバイスが 5 個よりも多く、バッファーを介さない TCK および TMS ネットを使用していますか。	はい : TCK および TMS 信号にバッファーが必要である可能性があります。大まかですが、デバイスの数が 5 個を超える場合は、バッファーを使用する必要があります。LS244 は、ザイリンクスのデバイスとバッファーを使用した例です。IEEE 1149.1 規格で定められているように、TMS および TDI ピンには内部プルアップ抵抗があります。これらの 50 ～ 150kΩの内部プルアップ抵抗は、選択されているモードに関係なくアクティブです。抵抗値は各 FPGA デバイス ファミリ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。 • Spartan-3 FPGA コンフィギュレーション ユーザー ガイド [参照 7] • Spartan-6 FPGA コンフィギュレーション ユーザー ガイド [参照 8] • Virtex-4 FPGA コンフィギュレーション ユーザー ガイド [参照 9] • Virtex-5 FPGA コンフィギュレーション ユーザー ガイド [参照 10] • Virtex-6 FPGA コンフィギュレーション ユーザー ガイド [参照 11] いいえ : 問題 7 に進んでください。
7. TCK の実行速度が速すぎませんか。	はい/わからない : [JTAG Chain] → [Xilinx Platform USB Cable] → [Speed] でケーブル速度を削減して、TCK ピンの周波数を最低速度設定になるように下げます。 いいえ : 次の情報を含めてザイリンクス テクニカル サポートからケースを開きます。 • XInfo • cs_analyzer.log • JTAG 操作中の JTAG ライン (TDI、TDO、TCK および TMS) のスクリーンショット (可能であれば、ターゲット FPGA に近く TCK の立ち上がりエッジの 1 つに焦点をあてたスクリーンショット)

表 A-7 : サーバー ホスト接続に関するトラブルシューティング

問題	回避策
1. ケーブルに接続しようとすると、次のようなメッセージがコンソールに表示されます。 ERROR:Socket Open Failed. localhost/127.0.0.1:50001 java.net.ConnectException:Connection refused これらのメッセージは、別のアプリケーション (たとえば、iMPACT ソフトウェア ツール) がケーブルを制御している場合に表示されます。また、システムにある別のデバイスまたはアプリケーションがそのポート / ソケットへのアクセスを拒否している場合にも表示されます。	問題 2 に進んでください。
2. ChipScope Pro Analyzer ツールが TCP/IP ソケットへ接続されないようにするファイアウォール アプリケーションを実行していますか。	はい : TCP/IP ソケットへのアクセスを拒否する可能性のあるアプリケーションをすべてオフにし、ケーブルを接続し直します。 いいえ : 問題 3 に進んでください。
3. ほかのザイリンクス ソフトウェア アプリケーションを使用してケーブルにアクセスしましたか。	はい : そのアプリケーションがケーブル ロックを解除していない可能性があります。該当するアプリケーションを閉じると、この問題は回避できることがほとんどです。それでも回避できない場合は、iMPACT バッチ モードで次のコマンドを実行して、停止した状態のケーブル ロックを削除します。 > impact -batch # cleancablelock # exit いいえ : 次の情報を含めてザイリンクス テクニカル サポートからケースを開きます。 • XInfo • cs_analyzer.log

ChipScope Pro Analyzer コアのトラブルシューティング

このセクションでは、ChipScope Pro Analyzer ツールからケーブルおよび JTAG チェーンにアクセスできるのに、ザイリンクス FPGA のデバッグ コアを検出できないか、ILA コアをトリガーできないか、キャプチャされた ILA コア データが表示されないといった問題について説明します。

- 「INFO: Found 0 Core Units in the JTAG device Chain」というメッセージに関する問題のトラブルシューティングについては、表 A-8 を参照してください。
- 「Waiting for upload」というメッセージに関する問題のトラブルシューティングについては、206 ページの表 A-9 を参照してください。
- 「ERROR: Fatal - Did not find trigger mark in buffer. Data buffer may be corrupt!」というメッセージに関する問題のトラブルシューティングについては、207 ページの表 A-10 を参照してください。

表 A-8 : コア検出に関するトラブルシューティング

問題	回避策
1. ケーブルに接続しようとすると、次のようなメッセージがコンソールに表示されます。 INFO: Found 0 Core Units in the JTAG device Chain ChipScope Pro Analyzer ツールでコア ユニットが検出されない場合、いくつかの原因が考えられます。ChipScope Pro Analyzer ツールは、デバイスに含まれるコアの数やタイプを示すステータス ワードを得るために、JTAG チェーンをポーリングします。ステータス ワードを読み込むと、JTAG TAP 信号のノイズやデザインに含まれるタイミング問題のいずれかが原因でデータが破損し、コアに影響を与えることがあります。	問題 2 に進んでください。
2. ChipScope Pro ICON (Integrated CoNtroller)、ILA (Integrated)、VIO (Virtual Input/Output)、および ATC2 (Agilent Trace Core) デバッグ コアが正しくデザインにインプリメントされているか、または存在しますか。	いいえ/わからない : デザインにこれらのコアが含まれるかどうかは、次の手順で確認できます。 1. FPGE Editor を開いて、配置配線済みの NCD ファイルを編集します。 2. [Tools] から [ILA] を選択すると、すべてのプローブ済みの信号が表示されます。エラー メッセージに「There is no ILA core」と表示されている場合は、デザインに ILA デバッグ コアが含まれていません。 コアが検出されなかった場合は、デザインに戻って、なぜコアがインプリメントされていないのかを確認する必要があります。ILA コアおよび ICON コアを含め、ネットリストすべてが正しくインプリメントされているかどうかは、合成および変換レポートから確認できます。コアに関連付けられた NCF (ネットリスト制約ファイル) も適用されているかどうか確認します。コアのネットリスト ファイル (*.ngc/ngo) がインプリメンテーション中に移動された場合、関連する制約ファイル (*.ncf) が適切に移動されていない可能性があります。 はい : 問題 3 に進んでください。

表 A-8 : コア検出に関するトラブルシューティング (続き)

問題	回避策
3. プログラム ツールで JTAG ロジックを解放しましたか。	いいえ/わからない : ChipScope Pro Analyzer ツールの代わりに iMPACT ツールを使用して FPGA をプログラムすると、この問題は回避できます。次を実行してください。 1. iMPACT を起動して FPGA をプログラムします。 2. iMPACT を終了します。 3. ChipScope Pro Analyzer ツールを実行します。 はい : 問題 4 に進んでください。
4. ザイリンクス System ACE™ MPM デバイスが JTAG チェーン内にありますか。	はい : JTAG チェーンに System ACE MPM が含まれている場合、既知の問題があり、コア ユニットが検出されないことがあります。この問題を回避するには、ChipScope Pro Analyzer ツールのプロジェクト ファイル (.cpj) に次の行を追加し、ケーブルをオープンする前に読み込みます。 avoidUserRegDeviceX=1,2 「X」を V50E デバイスの位置指数に置き換えます。チェーンの最初のデバイスのインデックスは 0 です。V50E デバイスのスキャンをスキップすることを示すメッセージが cs_analyzer.log ファイルに表示されます。 アップデートされた .cpj ファイルを使用する場合は、上記の手順に従います。ChipScope Pro Analyzer ツールの起動直後に .cpj ファイルをまず読み込んでください。 いいえ : 問題 5 に進んでください。
5. ザイリンクス デバイス以外のデバイスが JTAG チェーン内にありますか。	はい : チェーンにザイリンクス以外のデバイスが含まれる場合は、ChipScope Pro Analyzer の GUI で命令レジスタ長を入力する必要があります。命令レジスタ長は、デバイスに対する BSDL ファイルの次の行に表示されています。 attribute INSTRUCTION_LENGTH of <entity name> : entity is XX; 正しい命令レジスタ長を入力しないと、ChipScope Pro Analyzer で JTAG デバイスのオフセットが正しく設定できず、デバイスまたはデバッグ コアを識別できなくなります。 いいえ : 問題 6 に進んでください。
6. デバイスでスタートアップ シーケンスが問題なく完了しましたか。	いいえ : コンフィギュレーション オプションが適切に設定されていないために、ChipScope Pro Analyzer ツールでコアを検出できない可能性があります。BitGen オプションの LCK_cycle (Project Navigator では [Release DLL] オプション) が Nowait に設定されていない場合、GWE の解放と共に ChipScope Pro コアが初期化されるため、ChipScope Pro Analyzer でコアが検出されないことがあります。このオプションを Nowait (デフォルト) に設定すると、問題が回避されることがあります。 はい : 問題 7 に進んでください。

表 A-8 : コア検出に関するトラブルシューティング (続き)

問題	回避策
7. FPGA デバイスの DONE ピンはボードで Low に保持されていますか。 ボードに複数の FPGA があり DONE ピンが互いに接続されていて、コンフィギュレーションされていないデバイスにより DONE が Low に保持されコンフィギュレーションが完了していない場合に、このエラーが見られることがあります。	はい : 問題を回避するには、ボードの FPGA がすべてコンフィギュレーションされていることを確認、または bitgen オプションの DriveDONE がターゲット デバイスに対し設定されていることを確認します。 いいえ : 問題 8 に進んでください。
8. コア制約は正しく適用されていますか。	いいえ/わからない : ILA のクロックとして使用されるクロックに PERIOD 制約が追加されていることを確認します。ISE プロジェクトでこのネットに制約が付けられていない場合、ChipScope 制約は正しく適用されず、コアが認識されないことがあります。コアに関連付けられた NCF ファイルも適用されているかどうか確認します。コアのネットリスト ファイル (*.ngc/ngo) がインプリメンテーション中に移動された場合、関連する制約ファイル (*.ncf) が移動されていない可能性があります。 はい :次の情報を含めてザイリンクス テクニカル サポートからケースを開きます。 • cs_analyzer.log • Inserter プロジェクト (<project_name>.cdc) も含めた ISE ソフトウェア プロジェクトの圧縮ファイル

表 A-9 : ILA コアのトリガに関するトラブルシューティング

問題	回避策
1. ILA コアにトリガーを付けると、次のようなメッセージがステータス バーに表示されます。 Waiting for upload アップデートの待機中を示すメッセージですが、この後何も発生しません。この問題には、次のような原因が考えられます。 • トリガー条件が満たされていない • ILA コアにマップされたトリガー クロックが停止している • BUFG が ICON の JTAG CLK で使用されていない	問題 2 に進んでください。
2. トリガー条件は満たされていますか。	いいえ/わからない : ChipScope Pro Analyzer ツールのウィンドウ下部に表示されるメッセージを確認します。「Waiting for trigger, Sample buffer has 0 samples(0%)」のようなメッセージが表示された場合は、次の手順に従ってください。 [Trigger Setup] および [Trigger Immediate] ウィンドウを確認します。ChipScope Pro Analyzer ツールが取得を開始し、サンプルの波形を表示した場合、デザインには問題ありません。クロック信号は入ってきていますが、トリガー条件が満たされていません。 イベント (トリガー条件) が確実にこのデザインで生じている場合、[Trigger Setup] ウインドウで、条件が正しく設定されているかを確認します。 はい : 問題 3 に進んでください。
3. ILA コアに接続されているクロックは実行されていますか。	いいえ/わからない : ウィンドウ下部に「Waiting for Core to be armed, slow or stopped clock」のようなメッセージが表示された場合、トリガー条件は問題の原因ではありません。ILA コアには有効なクロックがなく、取得を開始できません。 この問題を修正するには、ChipScope Pro Core Inserter ツールまたは RTL コード (生成したコアを手動で使用する場合) を使用して有効なクロックをマップする必要があります。ILA コアにマップしたクロックが動作しているか分からない場合は、代わりにシステム クロック (または、確実に動作しているクロック) を接続します。 はい : 問題 4 に進んでください。
4. ICON JTAG クロックに BUFG を使用しましたか。	いいえ : JTAG クロックに BUFG が使用されない場合、「Waiting for upload」というメッセージが表示されます。この属性が確実に ICON コアの生成用に設定されるようにするには、デザインをインプリメントし直す必要があります。 はい : 問題 5 に進んでください。

表 A-9 : ILA コアのトリガに関するトラブルシューティング (続き)

問題	回避策
5. コア制約は正しく適用されましたか。	いいえ/わからない : ILA コアへの CLK 入力として使用されるクロックに PERIOD 制約が追加されていることを確認します。ISE プロジェクトでこのネットに制約が付けられていない場合、タイミング制約は正しく適用されず、コアが認識されないことがあります。コアに関連付けられた NCF ファイルも適用されているかどうか確認します。コアのネットリスト ファイル (*.ngc/ngo) がインプリメンテーション中に移動された場合、関連する制約ファイル (*.ncf) が適切に移動されていない可能性があります。 はい : 問題 6 に進んでください。
6. JTAG TCK クロックの実行速度が速すぎませんか。	はい/わからない : [JTAG Chain] → [Xilinx Platform USB Cable] → [Speed] でケーブル速度を削減して、TCK ピンの周波数を最低速度設定になるように下げます。 いいえ : 次の情報を含めてザイリンクス テクニカル サポートからケースを開きます。 • cs_analyzer.log • Inserter プロジェクト (filename.cdc) も含めた ISE ソフトウェア プロジェクトの圧縮ファイル

表 A-10 : 破損した ILA コアのデータ バッファーに関するトラブルシューティング

問題	回避策
1. ILA コアにトリガーを付けると、次のようなメッセージがコンソールに表示されます。 ERROR: Fatal - Did not find trigger mark in buffer.Data buffer may be corrupt! サンプリングされるデータに影響するデザインまたはコアのタイミング問題である可能性が高く、この結果バッファーのデータが破損してしまいます。	問題 2 に進んでください。
2. OFFSET IN/OFFSET OUT および PERIOD 制約を適用しましたか。	はい : タイミング問題がこの問題を引き起こさないように、これらの制約が適切な信号すべてに適用されているかどうか確認してください。 いいえ : 問題 3 に進んでください。
3. ICON JTAG クロックに BUFG を使用しましたか。	いいえ : 使用されない場合、「ERROR: Fatal - Did not find trigger mark in buffer. Data buffer may be corrupt!」というメッセージが表示される可能性があります。この属性が確実に ICON コアの生成用に設定されるようにするには、デザインをインプリメントし直す必要があります。 はい : 問題 4 に進んでください。

表 A-10 : 破損した ILA コアのデータ バッファーに関するトラブルシューティング (続き)

問題	回避策
4. コア制約は正しく適用されましたか。	いいえ/わからない : ILA コアへの CLK 入力として使用されるクロックに PERIOD 制約が追加されていることを確認します。ISE ソフトウェア プロジェクトでこのネットに制約が付けられていない場合、タイミング制約が正しく適用されず、ILA コアの制御ロジックが正しく動作しないことがあります。コアに関連付けられた NCF ファイルも適用されているかどうか確認します。コアのネットリスト ファイル (*.ngc/ngo) がインプリメンテーション中に移動された場合、関連する制約ファイル (*.ncf) が適切に移動されていない可能性があります。 はい : 問題 5 に進んでください。
5. デザインおよびコアはタイミングを満たしていますか。	いいえ : デザインが再びタイミングを満たすようにするには、トリガー ポート数、各ポートのトリガー ビット数、データ幅、データ深さなどを削減して、ILA コアを簡素にする必要があります。 はい : 次の情報を含めてザイリンクス テクニカル サポートからケースを開きます。 • cs_analyzer.log • Inserter プロジェクト (filename.cdc) も含めた ISE ソフトウェア プロジェクトの圧縮ファイル

ザイリンクス テクニカル サポートに提出する情報の取得方法

Xinfo 情報の取得

Xinfo は、ザイリンクス ツールで使用されるシステム情報を収集するために使用されるアプリケーションで、インストール ログおよびデバッグに便利な環境の詳細情報などが入手できます。

Windows の場合

1. [スタート] → [ファイル名を指定して実行] をクリックし、「Xinfo」と入力します。
2. Xinfo アプリケーションで [File] → [Export] → [Save as Text file] をクリックします。

Linux の場合

1. シェル プロンプトから xinfo を実行します。
2. Xinfo アプリケーションで [File] → [Export] → [Save as Text file] をクリックします。

ChipScope Pro Analyzer ログ ファイル情報の取得

cs_analyzer.log ファイルには、最新の ChipScope Pro Analyzer ツール セッションからのコンソール メッセージのログが含まれます。

Windows の場合

cs_analyzer.log ファイルは %homepath%/.chipscope ディレクトリに保存されています。このディレクトリは、通常 C:/Documents and Settings/<username>/.chipscope になります。

Linux の場合

cs_analyzer.log ファイルは $HOME/.chipscope ディレクトリに保存されています。

ChipScope Pro Core Inserter ツールのログ ファイル情報の取得

cs_inserter.log ファイルには、最新の ChipScope Pro Core Inserter ツール セッションからのコンソール メッセージのログが含まれます。

Windows の場合

cs_analyzer.log ファイルは %homepath%/.chipscope ディレクトリに保存されています。このディレクトリは、通常 C:/Documents and Settings/<username>/.chipscope になります。

Linux の場合

cs_analyzer.log ファイルは $HOME/.chipscope ディレクトリに保存されています。

圧縮された ISE ツール プロジェクトの取得

1. ISE Project Navigator ツールで [Project] → [Archive] をクリックします。
2. すべてのプロジェクトを含む project_name.zip ファイルが作成されます。プロジェクトで Core Inserter ツールを使用した場合は、圧縮前にプロジェクトに <filename>.cdc が含まれているかどうか確認してください。

付録 B

参考資料

マルチギガビット シリアル トランシーバーに関する資料 :

1. UG076 :『Virtex-4 FPGA RocketIO マルチギガビット トランシーバー ユーザー ガイド』
2. UG196 :『Virtex-5 FPGA RocketIO GTP トランシーバー ユーザー ガイド』
3. UG198 :『Virtex-5 FPGA RocketIO GTX トランシーバー ユーザー ガイド』
4. UG366 :『Virtex-6 FPGA GTX トランシーバー ユーザー ガイド』
5. UG371 :『Virtex-6 FPGA GTH トランシーバー ユーザー ガイド』
6. UG386 :『Spartan-6 FPGA GTP トランシーバー ユーザー ガイド』

ザイリンクス Virtex® FPGA および Spartan® FPGA に関する資料 :

7. UG332:『Spartan-3 ジェネレーション コンフィギュレーション ユーザー ガイド』
8. UG380 :『Spartan-6 FPGA コンフィギュレーション ユーザー ガイド』
9. UG071 :『Virtex-4 FPGA コンフィギュレーション ユーザー ガイド』
10. UG191 :『Virtex-5 FPGA コンフィギュレーション ユーザー ガイド』
11. UG360 :『Virtex-6 FPGA コンフィギュレーション ガイド』
12. XAPP139 :『バウンダリ スキャン (JTAG) を使用した Virtex FPGA のコンフィギュレーションとリードバック』
13. UG480 :『7 シリーズ FPGA の XADC 12 ビット 1MSPS デュアル アナログ - デジタル コンバーター』

ザイリンクスのツールおよびソリューション

14. ISE® Design Suite のマニュアル
15. エンベデッド開発キット (EDK) のマニュアル
16. ISE Design Suite ソフトウェア マトリックス
17. PlanAhead™ デザイン解析ツール
18. ザイリンクス サポート
19. System Generator for DSP
20. ザイリンクス オンライン ストア
21. シリコン ステッピング
22. UG192 :『Virtex-5 システム モニター ユーザー ガイド』
23. UG370 :『Virtex-6 システム モニター ユーザー ガイド』

その他の資料 :

24. ActiveState
25. アジレント テクノロジー
26. Tcl Developer Xchange
27. Byte Tools
28. UG480 :『7 シリーズ FPGA の XADC 12 ビット 1MSPS デュアル アナログ - デジタル コンバーター』