インテル® Quartus® Prime プロ・エディション ユーザーガイド

デバッグツール

インテル® Quartus® Prime 開発デザイントーストの更新情報: 18.1
目次

1. システム・デバッグ・ツールの概要………………………………………………………………………………… 7
  1.1. システム・デバッグ・ツールのポートフォリオ………………………………………………………………… 7
  1.1.1. システム・デバッグ・ツールの比較………………………………………………………………………… 7
  1.1.2. 一般的なデバッグ要件に対する推奨ツール………………………………………………………………… 8
  1.1.3. エコシステムのデバッグ…………………………………………………………………………………… 9
  1.2. RTL ノードのモニタリング・ツール……………………………………………………………………………10
    1.2.1. リソース使用率……………………………………………………………………………………………… 10
    1.2.2. ピンの使用方法…………………………………………………………………………………………… 12
    1.2.3. ユーザビリティの強化…………………………………………………………………………………… 12
  1.3. スティミュラス対応ツール………………………………………………………………………………………13
    1.3.1. In-System Sources and Probes……………………………………………………………………………… 13
    1.3.2. In-System Memory Content Editor……………………………………………………………………… 14
    1.3.3. System Console…………………………………………………………………………………………… 14
  1.4. 部分リコンフィグレーション・デザインのデバッグ……………………………………………………………20
    1.4.1. パーシャル・リコンフィグレーション・デザイン用のデバッグ・ファブリック…………………………20
  1.5. システム・デバッグ・ツールの概要改訂履歴………………………………………………………………… 21

2. Signal Tap Logic Analyzer を使用したデザインのデバッグ………………………………………………… 23
  2.1. Signal Tap Logic Analyzer………………………………………………………………………………………23
    2.1.1. ハードウェアおよびソフトウェア要件…………………………………………………………………… 24
    2.1.2. Signal Tap Logic Analyzer の機能と利点………………………………………………………………… 24
    2.1.3. 旧バージョンのインテル Quartus Prime 開発ソフトウェアとの後方互換性…………………………25
  2.2. Signal Tap Logic Analyzer タスクフローの概要…………………………………………………………… 25
    2.2.1. デザインへのSignal Tap Logic Analyzer の追加……………………………………………………… 26
    2.2.2. Signal Tap Logic Analyzer のコンフィグレーション…………………………………………………… 26
    2.2.3. トリガー条件の定義………………………………………………………………………………………… 27
    2.2.4. デザインのコンパイル……………………………………………………………………………………… 27
    2.2.5. ターゲットデバイスまたはデバイスのプログラム………………………………………………………… 27
    2.2.6. Signal Tap Logic Analyzer の実行……………………………………………………………………… 27
    2.2.7. キャプチャーしたデータの表示、解析、および使用……………………………………………………… 28
  2.3. Signal Tap Logic Analyzer のコンフィグレーション………………………………………………………… 28
    2.3.1. 取得クロックの割り当て…………………………………………………………………………………… 28
    2.3.2. Signal Tap ファイルへの信号の追加…………………………………………………………………… 29
    2.3.3. ブラグインを使用した信号の追加………………………………………………………………………… 31
    2.3.4. サンプル深度の指定………………………………………………………………………………………… 32
    2.3.5. 特定の RAM タイプの信号のキャプチャー……………………………………………………………… 33
    2.3.6. バッファー取得モードの選択………………………………………………………………………………… 33
    2.3.7. パイプライン設定の指定…………………………………………………………………………………… 35
    2.3.8. 関連サンプルのフィルタリング…………………………………………………………………………… 36
フィードバック

2.5. デザインのコンパイル................................................................. 68
  2.5.1. 再コンパイルを必要とする変更の防止................................. 68
  2.5.2. プロジェクト再コンパイルする必要性的確認....................... 68
  2.5.3. ラピッド・リコンパイルを使用したインクリメンタル配線.............. 69
  2.5.4. Signal Tap Logic Analyzer を使用したタイミングの保持............. 70
  2.5.5. パフォーマンスおよびリソースに関する考慮事項..................... 71

2.6. ターゲットデバイスのプログラム.................................................. 72
  2.6.1. .stp ファイルと .sof ファイル間の互換性の設定確認............... 73

2.7. Signal Tap Logic Analyzer の実行............................................... 73
  2.7.1. Runtime Reconfigurable オプション....................................... 74
  2.7.2. Signal Tap のステータスメッセージ....................................... 76

2.8. キャプチャーしたデータの表示, 解析, および使用.......................... 77
  2.8.1. セグメント化フィルターを使用したデータのキャプチャー............ 77
  2.8.2. 各取得モード間のプレフィル書き込み動作の相違点................... 79
  2.8.3. ビットパターン用ニーモニックの作成..................................... 80
  2.8.4. プラグインを使用した自動ニーモニック................................. 80
  2.8.5. デザイン内でのノードの検索................................................ 81
  2.8.6. キャプチャーしたデータの保存............................................. 81
  2.8.7. キャプチャーしたデータの他のファイル・フォーマットへのエクスポート..... 82
  2.8.8. Signal Tap リストファイルの作成....................................... 82

2.9. Signal Tap Logic Analyzer を使用したバージャル・リコンフィグレーション・デザインのデバッグ...... 82
  2.9.1. PR デザインのデバッグ時の推奨事項.................................... 83
  2.9.2. デバッグ用パーシャル・リコンフィグレーション・デザインのセットアップ...... 83
  2.9.3. PR デザイン内でのデータ取得の実行................................. 84

2.10. Signal Tap Logic Analyzer を使用したブロックベースのデザインのデバッグ............... 85
  2.10.1. コア・パーティションの再利用による Signal Tap...................... 85
  2.10.2. ルート・パーティションを再利用した Signal Tap..................... 89
  2.10.3. インポートしたスナップショットのデバッグ.......................... 91

2.11. その他の機能..................................................................... 92
  2.11.1. デザイン・インスタンスからの Signal Tap ファイルの作成......... 92
  2.11.2. Signal Tap MATLAB MEX 関数を使用したデータのキャプチャー...... 94
  2.11.3. ラボ環境での Signal Tap の使用...................................... 96
  2.11.4. Signal Tap Logic Analyzer を使用したリモートデバッグ........... 96
  2.11.5. コンフィグレーション・ビットストリーム・セキュリティ付きのデバイスにおける Signal Tap Logic Analyzer の使用.......................... 97
  2.11.6. Signal Tap ロジック・アナライザーで使用する FPGA リソースのモニタリング........ 97

2.12. デザイン例: Signal Tap Logic Analyzer の使用方法.................. 97
  2.13. カスタムトリガー・フローのアプリケーション例.......................... 98
2.13.1. デザイン例1：カスタムトリガー位置の指定

2.13.2. デザイン例2：triggercond1がtriggercond2とtriggercond3の間で10回発生する場合のトリガー

2.14. Signal Tapスクリプトのサポート


2.14.2. コマンドラインからのデータのキャプチャー

2.15. Signal Tap Logic Analyzerを使用したデザインのデバッグ・改訂履歴

4. Signal Probeを使用した迅速なデザイン検証

4.1. Signal Probeとラビット・リコンパイルを使用したデバッグフロー

4.1.1. Signal Probeピンの予約

4.1.2. デザインのコンパイル

4.1.3. Signal Probeピンへのノードの割り当て

4.1.4. デザインの再コンパイル

4.1.5. Fitterレポートの接続テーブルの確認

4.2. Signal Probeを使用した迅速なデザイン検証・改訂履歴

5. 外部ロジック・アナライザーを使用したインシステム・デバッグ

5.1. インテルQuartus Primeロジック・アナライザー・インターフェイス

5.2. ロジック・アナライザーの選択

5.2.1. 必要なコンポーネント

5.3. LAIの使用フロー

5.3.1. ロジック・アナライザー・インターフェイスのパラメーターの定義

5.3.2. 使用可能なI/OピンへのLAIファイルピンのマッピング

5.3.3. 内部信号のLAIバンクへのマッピング

5.3.4. インテルQuartus Primeプロジェクトのコンパイル

5.3.5. インテルでサポートしているデバイスのLAIを使用したプログラミング

5.4. ランタイムのアクティブバンクの制御

5.4.1. ロジック・アナライザーでのデータの取得

5.5. LAIコアのパラメーター

5.6. 外部ロジック・アナライザーを使用したインシステム・デバッグ

6. メモリーおよび定数のインシステム修正

6.1. ISMCEをサポートするIPコア

6.2. In-System Memory Content Editorを使用したデバッグフロー

6.3. デザイン内インスタンスのランタイム修正のイネーブル

6.4. In-System Memory Content Editorを使用したデバイスのプログラミング

6.5. メモリーインスタンスのISMCEへのロード

6.6. メモリー内のロケーションのモニタリング

6.7. Hex Editorを使用したメモリー内容の編集

6.8. メモリー・ファイルのインポートおよびエクスポート

6.9. 複数のデバイスへのアクセス

6.10. スクリプティング・サポート

6.10.1. insystem_memory_edit Tclパッケージ

6.11. メモリーおよび定数のインシステム修正・改訂履歴

7. In-System Sources and Probesを使用したデザインのデバッグ

7.1. ハードウェアおよびソフトウェア要件

7.2. In-System Sources and Probes Editorを使用したデザインフロー
8. System Console を使用したデザインの解析とデバッグ

8.1. System Console の概要

8.2. System Console の起動

8.3. System Console と相互作用する IP コア

8.4. System Console のGUI

8.5. System Console の起動

8.6. System Console のサービス

8.7. System Console の例とチュートリアル

8.8. System Console の使用

8.9. Toolkit

8.10. System Console の例とチュートリアル

8.11. On-Board デザインの起動
8.12.1. サポートされている MATLAB API コマンド...................................................... 201
8.12.2. 高レベルのフロー............................................................................................ 202
8.13. 廃止予定のコマンド.......................................................................................... 202

9. トランシーバー・リンクのデバッグ.............................................................................. 204
9.1. デバイスのサポート............................................................................................... 205
9.2. Channel Manager.................................................................................................. 205
9.2.1. チャネル表示モード....................................................................................... 206
9.3. トランシーバー・デバッグ・フローの手順............................................................... 206
9.4. デザインを変更してトランシーバーをデバッグ可能にする..................................... 207
9.4.1. トランシーバー IP コアのデバッグに関係するパラメーター............................... 207
9.5. インテル FPGA へのデザインのプログラミング.................................................. 212
9.6. Transceiver Toolkit へのデザインのロード.......................................................... 212
9.7. ハードウェア・リソースのリンク......................................................................... 212
9.7.1. 1 つのデザインを 1 つのデバイスに対してリンクする....................................... 213
9.7.2. 2 つのデザインを 2 つのデバイスに対してリンクする....................................... 214
9.7.3. 1 つのデザインを 2 つのデバイス上でリンクする............................................. 214
9.7.4. デザインとデバイスを別々のボード上でリンクする........................................... 214
9.7.5. ハードウェア接続の検証.................................................................................. 214
9.8. トランシーバー・チャネルの特定.......................................................................... 215
9.8.1. トランシーバー・チャネルの制御....................................................................... 215
9.9. トランシーバー・リンクの作成................................................................................. 216
9.10. リンクトテストの実行.......................................................................................... 216
9.10.1. BER テストの実行.......................................................................................... 216
9.10.2. リンクトテスト最適化テスト......................................................................... 218
9.10.3. Eye Viewer テストの実行.............................................................................. 218
9.11. PMA アナログ設定の制御................................................................................... 220
9.11.1. インテル Arria 10 および インテル Cyclone 10 GX の PAM 設定...................... 220
9.11.2. インテル Stratix 10 L- および L-タイルの PMA 設定........................................ 224
9.11.3. インテル Stratix 10 E-タイルの PMA 設定..................................................... 226
9.12. ユーザー・インタフェース設定リファレンス............................................................ 229
9.13. 一般的なエラーのトラブルシューティング............................................................ 233
9.14. API リファレンスのスクリプティング................................................................... 233
9.14.2. Data Pattern Generator のコマンド................................................................. 238
9.14.3. Data Pattern Checker のコマンド.................................................................... 240
9.15. トランシーバー・リンクのデバッグ 改訂履歴......................................................... 241

A. インテル Quartus Prime プロ・エディション・ユーザーガイド................................. 244
1. システム・デバッグ・ツールの概要

この章では、インテル® Quatrus® Prime 開発ソフトウェアのシステム・デバッグ・スイートで使用可能なツールの概要を簡単に説明します。また、デバッグ要件に最適なツールの選択基準についても説明します。

1.1. システム・デバッグ・ツールのポートフォリオ

インテル Quatrus Prime 開発ソフトウェアで提供しているシステム・デバッグ・ツールのポートフォリオは、リアルタイムのデザイン検証用です。

システム・デバッグ・ツールによって提供される可視性は、デザイン内の信号の配線（またはタッピング）をデバッグロジックに対して行うことで得られます。Compilerでは、デバッグログラフィックをデザインに内包し、プログラミング・ファイルを生成します。このファイルを FPGAまたはCPLDにダウンロードして解析に使用します。

システム・デバッグ・ツールの各ツールでは、使用可能なメモリー、ロジック、配線リソースの組み合わせを使用して、デバッグプロセスを支援します。異なるデザインでは制約や要件が異なるため、特定のデザイン要件に合うツールを選択します。要件には、使用可能なスペアピンの本数や物理デバイスに残っているロジックやメモリリソースの量などがあります。

1.1.1. システム・デバッグ・ツールの比較

表1. システム・デバッグ・ツールのポートフォリオ

<table>
<thead>
<tr>
<th>ツール</th>
<th>詳細</th>
<th>一般的な使い方</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Console</td>
<td>• リアルタイムのインシステム・デバッグ機能を提供します。</td>
<td>システムレベルのデバッグを実行する必要がある場合。例えば、Avalon®-MMスレーブまたはAvalon-STインタフェイスがある場合は、デザインのデバッグをトランザクションレベルで行います。</td>
</tr>
<tr>
<td></td>
<td>• システム内のMemory Mappedコンポーネントからの読み出しおよび書き込みを可能にします。プロッサーやまたは追加のソフトウェアは使用しません。</td>
<td>ボードレベルの信号アインテグリティのデバッグまたは最適化は、デザインの完了前に行う必要がありました。</td>
</tr>
<tr>
<td></td>
<td>• デザイン内のハードウェア・モジュールとの通信をTclインタフェースを介して行います。</td>
<td>デザインレベルのインタフェースのデバッグまたはトランザクションレベルのデバッグ。</td>
</tr>
<tr>
<td></td>
<td>• Tcスクリプト言語のすべての機能が利用できます。</td>
<td>デザインレベルのデバッグまたはトランザクションレベルのデバッグ。</td>
</tr>
<tr>
<td></td>
<td>• JTAGおよびTCP/IP接続をサポートします。</td>
<td>デザインレベルのデバッグまたはトランザクションレベルのデバッグ。</td>
</tr>
<tr>
<td>Transceiver Toolkit</td>
<td>• トランシーバー・リンク信号の品質の検証および調整を測定基準を組み合わせて行うことができます。</td>
<td>デザインの完了前に行う必要がありました。</td>
</tr>
<tr>
<td></td>
<td>• フジカル・メディア・アタッチメント (PMA)設定のAutoSweepingは、最適なパラメータ値の検出に役立ちます。</td>
<td>デザインの完了前に行う必要がありました。</td>
</tr>
<tr>
<td>Signal Tap Logic Analyzer</td>
<td>• FPGAリソースを使用します。</td>
<td>予備のオンチップメモリがあり、ハードウェアで動作するデザインの機能検証が必要な場合。</td>
</tr>
<tr>
<td></td>
<td>• テストノードをサンプリングし、その情報をインテルQuatrus Prime開発ソフトウェアに出力して、表示および解析を行います。</td>
<td>予備のオンチップメモリがあり、ハードウェアで動作するデザインの機能検証が必要な場合。</td>
</tr>
<tr>
<td>Signal Probe</td>
<td>内部信号のインクリメンタル配線をI/Oピンに対してすると同時に、前回の配置配線の結果を保存します。</td>
<td>予備のI/Oピンがあり、少数の制御ビンの動作検証が必要な場合。このとき外部ロジック・アナライザーまたはオシロスコープのいずれかを使用します。</td>
</tr>
</tbody>
</table>

continued...
1. システム・デバッグ・ツールの概要

表2. 一般的なデバッグ要件に対する推奨ツール (1)

<table>
<thead>
<tr>
<th>要件</th>
<th>Signal Probe</th>
<th>Logic Analyzer Interface (LAI)</th>
<th>Signal Tap Logic Analyzer</th>
<th>詳細</th>
</tr>
</thead>
<tbody>
<tr>
<td>より多くのデータストリーム</td>
<td>N/A</td>
<td>X</td>
<td>—</td>
<td>外部ロジック・アナライザーのLAIツールを使用すると、Signal Tap Logic Analyzerを使用する場合に比べて、キャプチャしたデータをより多く格納することができます。これは、外部ロジック・アナライザーではより大きなバッファへのアクセスが可能だからです。Signal Probeツールでは、データのキャッシュ状態のまま格納します。</td>
</tr>
<tr>
<td>より高速のデバッグ</td>
<td>X</td>
<td>X</td>
<td>—</td>
<td>LAIまたはSignal Probeツールは、外部機器（オシロスコープやミックスド・シグナル・オシロスコープ(MSO)など）と併用可能です。この機能により、タイミングに対するアクセスが可能で、結合されたデータのストリームのデバッグができます。</td>
</tr>
<tr>
<td>ロジックデザインへの最小限の影響</td>
<td>X</td>
<td>X (1)</td>
<td>X (2)</td>
<td>Signal Probeツールでは、ノードのインクリメンタル配線をピンに対して行います。このときデザインロジックに影響はありません。LAIによってデザインに追加されるロジックは最小限であるため、必要とされるディスクリソースは少なくすみます。Signal Tap Logic Analyzerによるデザインへの影響がほとんどない理由は、Compilerではデバッグロジックを別のデザインバーティションとして見なすからです。</td>
</tr>
<tr>
<td>短いコンパイルおよび再コンパイル時間</td>
<td>X</td>
<td>X (2)</td>
<td>X (2)</td>
<td>Signal Probeでは、インクリメンタル配線を使用して、信号の接続を予約済みのピンに対して行います。この機能によって可能になるのは、ソース信号の選択を変更する際の迅速な再コンパイルです。</td>
</tr>
<tr>
<td>高度なトリガー機能</td>
<td>N/A</td>
<td>N/A</td>
<td>X</td>
<td>Signal Tap Logic Analyzerのトリガー機能は、市販のロジック・アナライザーと同等です。</td>
</tr>
<tr>
<td>低I/O使用率</td>
<td>—</td>
<td>—</td>
<td>X</td>
<td>Signal Tap Logic Analyzerでは追加の出力ピンは必要ありません。LAIおよびSignal Probeの両方でI/Oピン・アサインメントが必要です。</td>
</tr>
<tr>
<td>迅速なデータ収集</td>
<td>N/A</td>
<td>—</td>
<td>X</td>
<td>Signal Tap Logic Analyzerでのデータ取得は、200 MHzを超える速度で可能です。</td>
</tr>
</tbody>
</table>

continued...
1. システム・デバッグ・ツールの概要

UG-20139 | 2018.09.24

<table>
<thead>
<tr>
<th>要件</th>
<th>Signal Probe</th>
<th>Logic Analyzer Interface (LAI)</th>
<th>Signal Tap Logic Analyzer</th>
<th>詳細</th>
</tr>
</thead>
</table>
| JTAG 接続が不要               | X            | —                             | X                        | Signal Probe および Signal Tap では、デバッグのためのホストは不要です。LAIのあるFPGAデザインでは、アクティブJTAG接続が必要。Intel Quartus Prime開発ソフトウェアを実行しているホストに対して必要です。
| 外部機器が不要                | —            | —                             | X                        | Signal Tap Logic Analyzer が必要とされるのは、Intel Quartus Prime開発ソフトウェアまたはスタンドアロンのSignal Tap Logic Analyzerを実行しているホストからのJTAG接続のみです。Signal ProbeおよびLAIでは、外部デバッグ装置を使用する必要があります。例えば、マルチメーター、オシロスコープ、ロジック・アナライザーなどです。

注：
1. X：当該機能に対する推奨ツールです。
2. —：ツールは当該機能に対して使用可能ですが、最良の結果が得られない場合もあります。
3. N/A：当該機能は選択したツールには適用されません。

1.1.3. エコシステムのデバッグ

インテル Quartus Prime開発ソフトウェアでは、デバッグツールと連携して、テスト対象ロジックの実行・解析およびクロージャーの最大化を行うことができます。

システム・デバッグ・ツールの非常に重要な特徴は、デザインとの通信方法です。インテル Quartus Prime開発ソフトウェアのデバッグツールを使用すると、デザインノードからの情報の読み出しと実行中のデータ入力が可能になりますが、ランタイムのデータ入力ができるのは、サブセットのみです。

表 3. システム・デバッグ・ツールの分類

<table>
<thead>
<tr>
<th>デバッグツール</th>
<th>デザインからのデータ読み出し</th>
<th>デザインへの値の入力</th>
<th>備考</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Tap Logic Analyzer</td>
<td>可能</td>
<td>不可</td>
<td>汎用のトラブルシューティング・ツール。レジスター転送レベル（RTL）のネットリストでの信号のプロービング用に最適化されています。</td>
</tr>
<tr>
<td>Logic Analyzer Interface</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signal Probe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-System Sources and Probes</td>
<td>可能</td>
<td>可能</td>
<td>これらのツールでは次の内容が実行可能です。定義したブレークポイントからのデータ読み出し実行中のデザインへの値の入力</td>
</tr>
<tr>
<td>Virtual JTAG Interface</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Console</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transceiver Toolkit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-System Memory Content Editor</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

オンチップ・デバッグ・ツールを組み合わせることで、デバッグ・エコシステムが形成されます。このツールの組み合わせによって、テスト対象のロジックへのスティミュラスの生成および要請がされ、完全なソリューションが得られます。
図-1：ランタイムでのエコシステムのデバッグ

1.2. RTL ノードのモニタリング・ツール

Signal Tap Logic Analyzer, Signal Probe, および LAIツールは、RTL信号のプローブおよびデバッグをシステムスピードで実行するのに役立ちます。このような汎用の解析ツールを使用することで、配線可能なノードをFPGAからタップして解析することが可能になります。

- デザインに予備のロジックとメモリリソースがある場合、Signal Tap Logic Analyzerによる迅速な機能検証を実行中のデザインに対して行うことができます。
- 逆に、ロジックおよびメモリースペースに余裕がない場合、LAIおよびSignal Probeツールの両方での内部デザイン信号のモニタリング簡素化は、外部機器を使用して行われます。

関連情報
- Signal Probe を使用した迅速なデザイン検証 (104 ページ)
- Signal Tap Logic Analyzer を使用したデザインのデバッグ (23 ページ)
- 外部ロジック・アナライザーを使用したインシステム・デバッグ (108 ページ)

1.2.1. リソース使用率

この3つのツールの選択基準で最も重要のは、デザイン実装後にデバイス上に残ったリソースとスペアピンの数です。

デバッグオプションの評価をデザイン計画プロセスの初期段階で行い、適切なオプションのサポートをボード、インテル Quartus Primeプロジェクト、およびデザインで確実に行ってください。初期段階に計画することで、デバッグ時間が短縮され、デバッグ方法に対処するために最終段階で変更する必要がなくなるでしょう。
1.2.1.1. オーバーヘッド・ロジック

デバッグツールに JTAG 接続が必要とされる場合は、SLD インフラストラクチャー・ロジックが必要です。これによって JTAG インターフェイスとの通信や、インスタンス化されたデバッグモジュール間のアービトレーションが行われます。このオーバーヘッド・ロジックでは、約 200 個のロジックエレメント (LE) を使用します。LE はリソースの一部として、サポートされているデバイスで使用可能です。デザインで使用可能なデバッグモジュールすべてで、このオーバーヘッド・ロジックを共有します。Signal Tap Logic Analyzer と LAI の両方で JTAG 接続を使用します。

1.2.1.1.1. Signal Tap Logic Analyzer の場合

Signal Tap Logic Analyzer では、ロジックリソースとメモリーリソースの両方が必要です。使用するロジックリソース量は、タップする信号の数とトリガー・ロジックの複雑度によって異なります。それでも、Signal Tap Logic Analyzer で使用するロジックリソース量は、一般的にはほとんどのデザインのわずかなる割合です。

ベースライン・コンフィグレーションの構成が、SLD アービトレーション・ロジックとベース・トリガー・ロジックを持つシングルノードの場合、そのベースライン・コンフィグレーションには、約 300 から 400 のロジックエレメント (LE) が含まれています。ノードをベースライン・コンフィグレーションに追加すると、それぞれにつき約 11 個の LE が追加されます。ロジックリソースと比較すると、メモリーリソースは、より重要な要素としてデザインで考慮すべきです。メモリー使用量は、かなりの量になる場合があり、データをキャプチャーするための Signal Tap Logic Analyzer インスタンスのコンフィグレーション方法やデザインのデバッグに必要なサンプル深度によって異なります。Signal Tap Logic Analyzer を追加する利点は、外部機器が必要ないことです。これはトリガーロジックとストレージがすべてチップ上にあるからです。

1.2.1.1.2. Signal Probe の場合

Signal Probe のリソース使用量は最小限です。これは、Signal Probe では JTAG 接続が必要なく、ロジックおよびメモリーリソースが不要だからです。Signal Probe に必要なのは、内部信号をデバッグ・テスト・ポイントに配線するためのリソースのみです。

1.2.1.1.3. Logic Analyzer パンフレットの場合

LAI では、SLD インフラストラクチャー・ロジックのほかに、少量のロジックが必要とされ、これによりテスト対象の信号間の多重化機能を実装します。データサンプルの記憶はチップ上に格納されないため、LAI ではメモリーリソースは使用しません。
1.2.1.2. リソースの見積もり

Signal Tap Logic Analyzer および LAI の見積もり機能を使用すると、十分なオンチップリソースが使用可能かどうかの判断が、デザインでのツールのコンパイル前に迅速にできます。

図-3: リソースの見積もり

1.2.2. ピンの使用方法

1.2.2.1. Signal Tap Logic Analyzer の場合

Signal Tap Logic Analyzer では、JTAG テストピン以外には追加のピンを使用しません。すべてのデータでは、バッファーにオンチップメモリーを使用し、Signal Tap ロジック・アナライザー GUI への通信は、JTAG テストポートを介して行われます。

1.2.2.2. Signal Probe の場合

使用するピン数と Signal Probe 機能用にタップされる信号の個数の比率は、1 対 1 です。この機能では、フリーピンの消費が早いため、この機能の一般的な用途は、コントロール信号をスペアピンに配線したデバッグです。

1.2.2.3. Logic Analyzer インターフェイスの場合

LAI では、最大 256 個までの信号のマッピングが各デバッグピンに対して可能で、使用できる配線リソースによって異なります。JTAG ポートでは、スペア I/O ピンにマッピングされたアクティブ信号を制御します。このような特徴を持った LAI が最適なのは、データバスを一連のテストピンに配線して解析に使用する場合です。

1.2.3. ユーザビリティーの強化

Signal Tap Logic Analyzer, Signal Probe, および LAI ツールを既存のデザインに追加する場合、影響は最小ですみます。ノード・ファインダーを使用すると、デバッグモジュールに配線する信号の検索ができます。このとき、HDL ファイルに変更を加える必要はありません。Signal Probe では、信号の挿入をフィッティング後のデータベースから直接行います。Signal Tap Logic Analyzer および LAI では、合成前ネットリストとフィッティング後ネットリストの両方からの信号の挿入をサポートしています。

1.2.3.1. インクリメンタル配線

Signal Probe では、インクリメンタル配線機能を使用します。インクリメンタル配線機能が実行できるのは、コンパイルの Fitter ステージでのみです。これにより、コンパイルされたデザインの変更は、新たに配線されたノード以外は行われません。Signal Probe を使用すると、フルコンパイルの 90%ものコンパイル時間が節約できます。
1.2.3.2. スクリプティングを使用したオートメーション

生産性向上の一環として、オンチップ・デバッグ・ツールセットに含まれるツールすべてで、quartus_stp Tcl パッケージを使用したスクリプティングをサポートしています。Signal Tap Logic Analyzer と LAI では、スクリプティングを使用すると、データ収集用のユーザー定義のオートメーショ
ンをラボでのデバッグ中に行うことが可能になります。System Console には、スクリプティング用の完全なTcl インタープリターが含まれています。

1.2.3.3. リモートデバッグ

システムのリモートデバッグの実行に インテル Quartus Prime 開発ソフトウェアを使用する場合は、System Console を介します。この機能を使用すると、フィールドでデプロイされた機器のデバッグに既存のTCP/IP 接続が使用できます。

- System Console を持つ Nios II システムを設定してリモートデバッグを実行する場合の詳細は、Application Note 624 を参照してください。
- インテル Quartus Prime FPGA SoC を設定してリモートデバッグをインテル SLD ツールで実行する場合の詳細は、アプリケーション・ノート 693 を参照してください。

関連情報
- Application Note 624: Debugging with System Console over TCP/IP
- Application Note 693: Remote Debugging over TCP/IP for Intel FPGA SoC

1.3. スティミュラス対応ツール

In-System Memory Content Editor, In-System Sources and Probes、仮想 JTAG インターフェイスでは、JTAG インターフェイスを汎用通信ポートとして使用することができます。

3 つのツールをすべて使用して同じ結果を得ることは可能ですが、特定のアプリケーションにおいては、次の考慮事項によって、特定のツールがより使いやすくなる場合があります。

- In-System Sources and Probes は、制御信号のトグルに最適です。
- In-System Memory Content Editor は、大規模なテストデータの入力に役立ちます。
- 仮想 JTAG インターフェイスは、上級ユーザーがカスタムの JTAG ソリューションを開発する場合に適しています。

System Console では、システムレベルのデバッグをトランザクション・レベル (Avalon-MM スレーブや Avalon-ST インターフェイスなど) で提供しています。チップへの通信は、JTAG および TCP/IP プロトコルを介して行います。System Console では、Tcl インタープリターを使用してハードウェア・モジュールと通信し、そのモジュールをデザインにインスタンス化します。

1.3.1. In-System Sources and Probes

In-System Sources and Probes を使用すると、デザインの書き込みおよび読み出しが、JTAG ソースへのアクセスによって可能になります。

インテル FPGA IP をユーザーの HDL コードにインスタンス化します。このインテル FPGA IP コアには、デザイン内の信号に接続するソースポートとプローブポートが含まれています。また、JTAG インターフェイスのトランザクションの詳細も抽象化します。
さらに、In-System Sources and Probes の GUI では、インスタンスごとのソースポートとプローブポートを表示し、プローブポートからの読み出しとソースポートへの駆動が可能です。この機能により、In-System Sources and Probes は、デバッグ処理中のコントロール信号のトグルに最適なものになっています。

関連情報
In-System Sources and Probes を使用したデザインのデバッグ (127 ページ)

1.3.1. プッシュボタン機能
プロジェクトの開発段階では、デザインをデバッグするために、In-System Sources and Probes の GUI をプッシュボタンや LED の代わりに使用します。さらに In-System Sources and Probes では、一連のスクリプトコマンドを読み出しと書き込み用にサポートしており、これには Signal Tap Logic Analyzer を使用します。独自の Tk グラフィカル・インターフェイスの構築には、Toolkit API を使用します。この機能が最適なのは、仮想フロントパネルの構築をデザインの試作段階で行う場合です。

関連情報
• Toolkit API (161 ページ)
• Signal Tap スクリプトのサポート (100 ページ)

1.3.2. In-System Memory Content Editor
In-System Memory Content Editor を使用すると、メモリーコンテンツを迅速に表示・変更するのに、GUI インタフェイスまたは Tcl スクリプトコマンドが使用できます。In-System Memory Content Editor を動作させるには、シンプルポート RAM ブロックをデュアルポート RAM ブロックに変換します。1つのポートをクロックドメインとデータ信号に接続し、もう1つのポートを JTAG クロックおよびデータ信号に接続して、編集または表示に使用します。

関連情報
メモリーおよび定数のインシステム修正 (118 ページ)

1.3.2.1. テストベクトルの生成
大規模なデータセットの変更が簡単にできるため、In-System Memory Content Editor の便利な用途は、テストベクトルのデザイン用の生成です。例えば、未使用のメモリーブロックのインスタンス化、テスト対象のロジックへの出力ポートの接続 (システム側でテスト対象のロジックと同じクロックを使用)、アドレス生成およびメモリーの制御用のグルーロジックの作成です。ランタイムに、メモリーコンテンツの変更をスクリプトまたは In-System Memory Content Editor の GUI を使用して行い、データコンテンツのバースト・トランザクションをテスト対象のロジックと同期している変更済みの RAM ブロック内で実行することができます。

1.3.3. System Console
System Console はフレームワークとして、インテル Quartus Prime 開発ソフトウェアから起動し、様々なデバッグツスク実行用のサービスが開始できます。System Console の Tcl スクリプトと GUI では、ブラッドフォーム・デザイナー (旧: Qsys) システム統合ツールにアクセスして、デザインの下位レベルのハードウェア・デバッグの実行や、パスによるモジュールの識別、ブラッドフォーム・デザイナー (旧: Qsys) モジュールへの接続の開閉を行います。デザインへのアクセスはシステムレベルで可能で、その目的は、デザインのロード、アンロード、転送を複数のデバイスに対して行うことです。さらに、System Console では、グラフィカル・インターフェイス構築用の Tk ソールキットもサポートしています。
関連情報
System Console を使用したデザインの解析とデバッグ（141 ページ）

1.3.3.1. シグナル・インテグリティーのテスト
System Console によってアクセスできるコマンドでは、テストパターンの生成方法の制御や、テストパターンによって生成されるデータ精度の検証ができます。JTAG デバッグコマンドを System Console で使用して、JTAG チェーンの機能性とシグナル・インテグリティーを検証することができます。また、クロックおよびリセット信号のテストも可能です。

1.3.3.2. ボードの立ち上げと検証
System Console を使用して、開発ボードのプログラマブル・ロジック・デバイスにアクセス、ボードの立ち上げおよび検証の実行ができます。さらに、Nios II またはインテル FPGA SoC プロセッサーで実行中のソフトウェアや、パイストリームを生成あるいは消費するモジュールへのアクセスも可能です。

1.3.3.3. Transceiver Toolkit を使用したリンク・シグナル・インテグリティーの検証
Transceiver Toolkit を System Console フレームワークから実行すると、トランシーバー・リンクの自動テストを実行して、トランシーバー・デザインのデバッグおよび最適化することができます。Transceiver Toolkit の GUI を使用してトランシーバー・デバイスのチャネルリンクを設定し、パラメーターを実行時に変更して、シグナル・インテグリティーを測定することができます。一部のデバイスでは、Transceiver Toolkit でアイ・コンター・テストの実行および表示もできます。

1.4. 仮想 JTAG インターフェイス インテル FPGA IP
仮想 JTAG インターフェイス インテル FPGA IP では、JTAG リソースの操作用に最高レベルの細分性が提供されます。この インテル FPGA IP を使用すると、独自の JTAG スキャンの構築のために、すべての JTAG コントロール信号が公開され、JTAG 命令レジスター (IR) および JTAG データレジスター (DR) のコンフィグレーションが行われます。ライタイム中、IR/DR チェーンの制御には、Tcl API または System Console を使用します。この機能は、JTAG インターフェイスを十分に理解し、使用するリソースの数と種類の正確な制御を希望しているユーザーを対象としています。

関連情報
- 仮想 JTAG (altera_virtual_jtag) IP コアのユーザーガイド
- Virtual JTAG Interface (VJI) インテル FPGA IP
  Intel Quartus Prime Help 内

1.5. システムレベルのデバッグ・ファブリック
インテル Quartus Prime では、コンパイル中に JTAG Hub を生成して、1 つのデザイン内で複数のデバッグツールのインスタンスを使用できるようにします。

Intel FPGA オンチップ・デバッグ・ツールのほとんどで JTAG ポートを使用し、テスト対象のデバッグロジックおよび信号からのデータの制御とリードバックを行います。JTAG Hub では、JTAG リソースの共有を管理します。

注意: System Console の場合、デバッグ IP コアを明示的にデザインに挿入し、デバッグをイネーブルします。
The SLD JTAG Bridge では、デバッグ・ファブリックをパーティション間で展開して、上位パーティション (静的領域またはルート・パーティション) から下位パーティション (パーシャル・リコンフィグレーション領域またはコア・パーティション) 内のデバッグ信号にアクセスします。

このブリッジを構成するのは次の 2 つの IP コンポーネントです。

1. **SLD JTAG Bridge Agent インテル FPGA IP** - 上位パーティションにあります。JTAG デバッグ・ファブリックを上位パーティションから下位パーティションに展開します。下位パーティションには SLD JTAG ブリッジホスト IP が含まれています。SLD JTAG Bridge Agent IP を上位パーティションでインスタンス化します。

2. **SLD JTAG ブリッジホスト インテル FPGA IP** - 下位レベルのパーティションにあります。片側で PR JTAG ハブに接続し、上位パーティションで SLD JTAG Bridge Agent に接続します。下位レベルの JTAG デバッグ・ファブリックを上位パーティションに接続します。上位パーティションには SLD JTAG Bridge Agent IP が含まれています。SLD JTAG Bridge Host IP を下位パーティションでインスタンス化します。

図 -4: SLD JTAG Bridge の信号

デバッグする PR 領域または予約済みコア・パーティションごとに、SLD JTAG Bridge Agent 1 つを上位パーティションに、SLD JTAG Bridge Host 1 つを下位パーティションにインスタンス化してください。

**1.6.1. SLD JTAG Bridge Index**

The SLD JTAG Bridge Index では、デザイン内に存在する SLD JTAG Bridge のインスタンスを一意に識別します。Bridge Index に関する情報は、合成レポートを参照してください。

インテル Quartus Prime 開発ソフトウェアでは、デザイン内にある SLD JTAG Bridge の複数のインスタンスをサポートしています。Compiler では、インデックス番号を割り当てて各インスタンスを区別します。ルート・パーティションのブリッジインデックスは、常に None です。

Signal Tap ロジック・アナライザーをルート・パーティションに対してコンフィグレーションする場合は、Bridge Index 値を JTAG Chain Configuration ウィンドウで None に設定します。
図-5: JTAG Chain Configuration Bridge Index

図-6: 複数のSLD JTAG Bridgeを使用したデザイン

Compilation ReportのBridge Index Information

デザイン合成後、Compilation Reportにデザイン内のSLD JTAG Bridge Agentのインデックス番号が一覧表示されます。Synthesis > In-System Debugging > JTAG Bridge Instance Agent Information レポートを開くと、ブリッジ・インデックスの列挙方法の詳細が確認できます。レポートには、階層パスと関連インデックスが表示されます。

合成レポート（<base revision>.syn.rpt）では、この情報はJTAG Bridge Agent Instance Information に表示されます。
1.6.2. SLD JTAG Bridge Agent のインスタンス化

SLD JTAG Bridge Agent インテル FPGA IP の生成・インスタンス化を実行するには、次の手順を実行します。

1. IP Catalog（Tools > IP Catalog）で、SLD JTAG Bridge Agent と入力します。

2. SLD JTAG Bridge Agent インテル FPGA IP をダブルクリックします。

3. Create IP Variant ダイアログボックスでファイル名を入力し、Create をクリックします。

IP Parameter Editor Pro 画面に IP パラメーターが表示されます。ほとんどの場合、デフォルト値を変更する必要はありません。
図-10: SLD JTAG Bridge Agent インテル FPGA IP のパラメーター

4. Generate HDL をクリックします。
5. 生成が問題なく完了したら Close をクリックします。
6. インスタンス・テンプレートが必要な場合は、IP Parameter Editor Pro で Generate > Show Instantiation Template をクリックします。

1.6.3. SLD JTAG Bridge Host のインスタンス化

SLD JTAG Bridge Host インテル FPGA IP を生成しインスタンス化するには、次を実行します。
1. IP Catalog (Tools > IP Catalog) で SLD JTAG Bridge Host と入力します。
図-11: IP Catalog での検索

2. SLD JTAG Bridge Host インテル FPGA IP をダブルクリックします。
3. Create IP Variant ダイアログボックスでファイル名を入力し、Create をクリックします。
図-12: IP Variant ダイアログボックスの作成

IP Parameter Editor Pro 画面に IP パラメーターが表示されます。ほとんどの場合、デフォルト値を変更する必要はありません。

図-13: SLD JTAG Bridge Host インテル FPGA IP のパラメーター

4. Generate HDL をクリックします。
5. 生成が問題なく完了したら、Close をクリックします。
6. インスタンス・テンプレートが必要な場合は、IP Parameter Editor Pro で Generate > Show Instantiation Template をクリックします。

1.7. パーシャル・リコンフィグレーション・デザインのデバッグ

Signal Tap ロジック・アナライザーやでは、デザインの静的領域または PR 領域をデバッグすることができ、静的領域だけをデバッグする場合は、In-System Sources and Probes Editor、In-System Memory Content Editor、または JTAG Avalon Master Bridge を使用します。

関連情報
Signal Tap Logic Analyzer を使用したパーシャル・リコンフィグレーション・デザインのデバッグ (82 ページ)

1.7.1. パーシャル・リコンフィグレーション・デザイン用のデバッグ・ファブリック

PR デバッグ用のデザインは、プランニング段階の早期に準備して、静的領域と PR 領域の両方を確実にデバッグできるようにしてください。
パーシャル・リコンフィグレーションを使用したデザインの場合、Compiler によって生成される集中型デバッグネーミング（またはハブ）には、システムレベルのデバッグ・エージェントを含む各領域（静的および PR）が含まれています。各ハブでは、そのパーティション内のデバッグ・エージェントを処理します。デザイン階層では、静的領域に対応するハブは auto_fab_0 です。

親パーティションと子パーティションのハブを接続するには、SLD JTAG ブリッジ 1 つのインスタンス化をデバッグする PR 領域ごとに行ってください。

関連情報
- デバッグ用パーシャル・リコンフィグレーション・デザインのセットアップ (83 ページ)
- Signal Tap Logic Analyzer を使用したパーシャル・リコンフィグレーション・デザインのデバッグ (82 ページ)

1.7.1.1. PR デバッグ・インフラストラクチャーの生成
コンパイル時に、合成エンジンでは次の機能を実行します。
- メインの JTAG ハブを静的領域に生成します。
- 静的領域に Signal Tap インスタンスが含まれる場合、そのインスタンスをメインの JTAG ハブに接続します。
- ブリッジ・エージェントとブリッジホストのインスタンスを検出します。
- SLD JTAG ブリッジ・エージェントのインスタンスをメインの JTAG ハブに接続します。
- Signal Tap インスタンスを含む PR 領域内の各ブリッジ・ホスト・インスタンスに対しては、次を実行します。
  - PR JTAG ハブを PR 領域に生成します。
  - PR 領域内のすべての Signal Tap インスタンスを PR JTAG ハブに接続します。
  - SLD JTAG ブリッジホストのインスタンスを検出します。
  - PR JTAG ハブを JTAG ブリッジホストに接続します。

1.8. システム・デバッグ・ツールの概要 改訂履歴
次の改訂履歴はこの章に適用されます。

<table>
<thead>
<tr>
<th>ドキュメント・バージョン</th>
<th>インテル Quartus Prime のバージョン</th>
<th>変更内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.09.24</td>
<td>18.1.0</td>
<td>SLD JTAG Bridge に関する説明を追加しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ブロックベースのデザインに関する情報を追加しました。</td>
</tr>
<tr>
<td>2018.05.07</td>
<td>18.0.0</td>
<td>PR デザインのデバッグ・ファブリックに関する情報を Signal Tap Logic Analyzer を使用したデバッグの対処法を説明しました。</td>
</tr>
<tr>
<td>2017.05.08</td>
<td>17.0.0</td>
<td>「アルテラの JTAG インターフェイスを必要とするアーキテクチャ・ロジック」のトピックを、新しいトピックの「システムレベルのデバッグ・インフラストラクチャー」に統合しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>次のトピックを追加しました。システムレベルのデバッグ・ツールを使用したパーシャル・リコンフィグレーション・デザインのデバッグツール</td>
</tr>
<tr>
<td>2016.10.31</td>
<td>16.1.0</td>
<td>インテルへのブランド変更を行いました。</td>
</tr>
<tr>
<td>2015.11.02</td>
<td>15.1.0</td>
<td>Quartus II のインスタンスを変更して インテル Quartus Prime にしました。</td>
</tr>
<tr>
<td>2014 年 6 月</td>
<td>14.0.0</td>
<td>System Console で Tk ツールキットをサポートしているという情報を追加しました。</td>
</tr>
</tbody>
</table>

continued...
ドキュメント・バージョン | インテル Quartus Prime のバージョン | 変更内容
---|---|---
2013年11月 | 13.1.0 | Dita 変換。アルテラ SoC アプリケーションノートの TCP/IP 経由のリモートデバッグへのリンクを追加しました。
2012年6月 | 12.0.0 | メンテナンス・リリース
2011年11月 | 10.0.2 | メンテナンス・リリース。ドキュメントのテンプレートを更新しました。
2010年12月 | 10.0.1 | メンテナンス・リリース。ドキュメントのテンプレートを更新しました。
2010年7月 | 10.0.0 | 初版

関連情報
Documentation Archive
以前のバージョンの インテル Quartus Prime ハンドブックは、文書アーカイブを検索してください。
2. Signal Tap Logic Analyzer を使用したデザインのデバッグ

2.1. Signal Tap Logic Analyzer

Signal Tap Logic Analyzer では、FPGA デザイン内の信号のビヘイビアをリアルタイムでキャプチャーおよび表示して、デザインの通常動作中に内部信号のビヘイビアを調べることができます。このとき、余分な I/O ピンや外部のラボ機器を使用する必要はありません。

デバッグプロセスを容易にするために、キャプチャしたデータをデバイスメモリーに保存してそれを後で解析することができます。デバッグに無関係なデータをフィルタリングするために、カスタムのトリガ条件ロジックを定義することもできます。Signal Tap Logic Analyzer でサポートしているチャネル数、サンプル深度、およびクロック速度は、プログラマブル・ロジック市場のあらゆるロジック・アナライザーの中で最大・最速です。

図 -14:  Signal Tap Logic Analyzer のブロック図

Signal Tap Logic Analyzer は、スタンドアロン・パッケージまたはソフトウェア・サブスクリプションとして入手可能です。

注意: インテル Quartus Prime プロ・エディション開発ソフトウェアでは、新しい方法を使用して設定と割り当てを行います。例えば、Signal Tap アサインメントに含まれるのは instance 名のみで、entity:instance 名は含まれません。既存の Signal Tap ファイル (.stp) のインテル Quartus Prime プロ・エディションへの移行に関する詳細は、インテル Quartus Prime プロ・エディションへの移行を参照してください。

関連情報

インテル Quartus Prime プロ・エディションへのマイグレーション
インテル Quartus Prime プロ・エディション使い始めユーザーガイド 内
2.1.1. ハードウェアおよびソフトウェア要件

Signal Tap ロジック・アナライザーを使用してロジック解析を実行するには、次のハードウェアとソフトウェアが必要となります。

- **Signal Tap ロジック・アナライザー**
  
  シグナルタップ・ロジック・アナライザーには次のソフトウェアが含まれています。一インテル Quartus Prime デザイン・ソフトウェア
  一インテル Quartus Prime Lite Edition

  これ以外に、Signal Tap ロジック・アナライザースタンドアロン・ソフトウェアとスタンドアロン Programmer ソフトウェアを使用します。

- **ダウンロード/アップロード・ケーブル**

- **テスト対象のデバイスへの JTAG 接続を備えたインテル開発キットまたはユーザー・デザイン・ボード**

  データ取得中、デバイス内のメモリーブロックでは、キャプチャーされたデータを格納し、次にそのデータのロジック・アナライザーへの転送あるいはインテル FPGA ダウンロード・ケーブルなどの JTAG 通信ケーブルを介して行います。

2.1.1.1. スタンドアロン Signal Tap Logic Analyzer GUI を開く

1. 新規 Signal Tap をコマンドラインで開くには、次の通りに入力します。

   ```sh
quartus_stpw <stp_file>.stp
```

2.1.2. **Signal Tap Logic Analyzer の機能と利点**

<table>
<thead>
<tr>
<th>機能</th>
<th>利点</th>
</tr>
</thead>
<tbody>
<tr>
<td>クイック・アクセス・ツールバー</td>
<td>ワンクリックで使用頻度の高いメニュー項目の操作ができます。アイコンの上にマウスを重ねると、ツールヒントが表示されます。</td>
</tr>
<tr>
<td>1つのデバイス内で複数のロジック・アナライザーが使用可能</td>
<td>デザイン内の複数のクロックドメインのデータを同時にキャプチャーできます。</td>
</tr>
<tr>
<td>1つの JTAG ケース内の複数デバイスで、複数のロジック・アナライザーが使用可能</td>
<td>JTAG ケース内の複数のデバイスからのデータを同時にキャプチャーできます。</td>
</tr>
<tr>
<td>Nios II プラグインのサポート</td>
<td>Nios II プロセッサなどの IP に対するノード、トリガー、信号ノーミニックが指定できます。</td>
</tr>
<tr>
<td>アナライザーエンスタスごとに最大 10 個の基本、比較または高速トリガー条件の設定が可能</td>
<td>複雑なデータ・キャプチャー・コマンドをロジック・アナライザーに通信できるため、精度の向上および問題の分離が実現します。</td>
</tr>
<tr>
<td>パワーアップトリガー</td>
<td>トリガーの信号データをキャプチャーします。対象となるトリガーの発生タイミングは、デバイスのプログラミング後、かつ手動でのロジック・アナライザーの起動前です。</td>
</tr>
<tr>
<td>カスタムトリガー HDL オブジェクト</td>
<td>独自のトリガーのコーディングを Verilog HDL または VHDL で行い、デザイン階層内の任意の箇所にあるモジュールのインスタンスをタップすることができます。このとき、必要な接続をすべて手動接続する必要はありません。これにより、デザイン全体に広がるノードをタップするプロセスが簡素化されます。</td>
</tr>
<tr>
<td>State-based トリガーフロー</td>
<td>トリガー条件を整理して、ロジック・アナライザーでキャプチャーする内容を適切に定義することができます。</td>
</tr>
</tbody>
</table>

continued...
機能 | 利点
---|---
ラピッド・リコンパイルを使用したインクリメンタル配線 | トリガー入力、データ入力、ストレージ検査子入力、ノード数を手動で割り当ててフルコンパイルを実行することで、Signal Tap Logic Analyzerを使用して設計に含めることができます。その後、設計内の異なるノードに対する接続、切断、およびブロックを直接的に行います。ラピッド・リコンパイルを使用してインクリメンタル配線を実行すると、最初のフルコンパイルと比較して2〜4倍の高速化が達成可能です。
柔軟なバッファ取得モード | バッファ取得コントロールを使用すると、アクションをパッファ変数に書き込まれるデータを正確に制御することができます。セグメント化バッファとStorage Qualificationを持つ非セグメント化バッファの両方で、デザインのデバッグに関係のデータサンプルを破棄できます。
MATLAB*とインクルードされたMEX機能との統合 | Signal Tap Logic AnalyzerでキャプチャーしたデータをMATLAB数マトリックスにまとめます。
ロジック・アナライザーインスタンスあたり最大2,048チャンネル | 各チャネルで大容量サンプルセットのキャプチャーが可能です。
インスタンスあたり最大128Kのサンプリング | データノードの同期サンプリングのため、テスト対象のロジックをドライブするクロックツリーを使用します。
リソース使用量の推定 | Signal Tap Logic Analyzerコンフィグレーションで使用するロジックとメモリーデバイスのリソースの量を推定します。
追加コストが不要 | インテルQuartus PrimeサブスクリプションとインテルQuartus Prime Lite Editionには、Signal Tap Logic Analyzerが含まれています。
他のオンチップ・デバッグ・ツールとの互換性 | Signal Tap Logic AnalyzerをIn-System Memory Content EditorなどのJTAGベースのオンチップ・デバッグ・ツールと連携して使用すると、信号値の変更がリアルタイムで可能です。
浮動小数点表示フォーマット | Edit > Bus Display Format > Floating-pointの順でクリックして有効にします。次のフォーマットがサポートされています。
• 単精度浮動小数点フォーマットIEEE754 Single (32-bit)
• 倍精度浮動小数点フォーマットIEEE754 Double (64-bit)

関連情報
システム・デバッグ・ツールの概要 (7ページ)

2.1.3 旧バージョンのインテルQuartus Prime開発ソフトウェアとの後方互換性

旧バージョンのインテルQuartus Prime開発ソフトウェアで作成された.stpファイルを新しいバージョンのソフトウェアで開くと、その.stpファイルは旧バージョンのインテルQuartus Prime開発ソフトウェアで開くことができません。

インテルQuartus Primeプロジェクト・ファイルを旧バージョンのソフトウェアで作成した場合、新バージョンのソフトウェアで開くと、その.stpファイルは旧バージョンのインテルQuartus Prime開発ソフトウェアで開くことができません。

プロジェクトの再コンパイルには、.stpコンフィグレーション・ファイルの更新が必要になることがあります。コンフィグレーション・ファイルの更新には、Signal Tap Logic Analyzerを用います。コンフィグレーションの更新が必要な場合、表示されるプロンプトで、.stpファイルを更新して、インテルQuartus Prime開発ソフトウェアの現行バージョンに適合させるかどうかを選択できます。

2.2  Signal Tap Logic Analyzerタスクフローの概要

Signal Tap Logic Analyzerを使用してデザインをデバッグするには、いくつかのタスクを実行して、ロジック・アナライザーを追加、コンフィグレーション、および実行します。
図 -15: Signal Tap Logic Analyzer のタスクフロー

2.2.1. デザインへの Signal Tap Logic Analyzer の追加

stp またはロジック・アナライザのパラメーター化された HDL インスタンス表現の作成には、IP Catalog およびパラメーター・エディターを使用します。複数のクロックドメインを同時にモニタリングする場合、ロジック・アナライザのインスタンスをデザインに追加します。これが制限されるのは、デバイスで使用可能なリソースによってのみです。

2.2.2. Signal Tap Logic Analyzer のコンフィグレーション

Signal Tap Logic Analyzer をデザインに追加した後、ロジック・アナライザーのコンフィグレーションを行って、希望の信号をモニタリングします。

信号の追加には、手動または Nios II プロセッサープラグインなどのプラグインを使用し、特定の IP に関連する信号のセット全体を追加します。

データ・キャプチャー・パッファーの設定を指定します。これには、パッファーのサイズや Signal Tap Logic Analyzer でデータをキャプチャーして格納する方法が含まれます。使用するデバイスでメモリータイプの選択がサポートされている場合、メモリータイプを指定してパッファーに使用できます。
2.2.3. トリガー条件の定義

Signal Tap Logic Analyzer ではデフォルトで、データの連続キャプチャーをロジック・アナライザーの動作中に行います。特定の信号データのキャプチャーおよび格納には、トリガーを設定し、条件を指定してデータのキャプチャーの開始または停止を行います。

Signal Tap Logic Analyzer では、トリガー条件の定義ができます。これには、1つの信号の立ち上がりエッジなどの非常に単純なトリガー条件もあれば、非常に複雑で、信号グループ、追加ロジック、および複数条件などを含む条件もあります。Power-Up Trigger を使用すると、トリガーイベントからのデータのキャプチャーが可能です。このトリガーイベントが発生するのは、コンフィグレーション後にデバイスがユーザーモードに入った直後です。

関連情報
トリガーの定義 (44 ページ)

2.2.4. デザインのコンパイル

.stp ファイルをコンフィグレーションし、トリガー条件を定義したら、ロジック・アナライザーを含むプロジェクトをデザインにコンパイルします。

関連情報
デザインのコンパイル (68 ページ)

2.2.5. ターゲットデバイスまたはデバイスのプログラム

デバイスのデバッグに Signal Tap Logic Analyzer を使用する場合、ターゲットデバイスのプログラミングは、インテル Quartus Prime Programmer を使用しなくても .stp から直接実行できます。また、複数のデバイスのプログラミングは、デザインが異なっていても可能で、それらのデバイスの同時デバッグができます。

関連情報
- ターゲットデバイスのプログラム (72 ページ)
- 複数の Signal Tap ファイルとコンフィグレーションの管理 (42 ページ)

2.2.6. Signal Tap Logic Analyzer の実行

デバイスの通常の動作では、ロジック・アナライザーの制御を JTAG 接続を介して行い、トリガー条件の検索を開始するタイミングを指定します。これによりデータのキャプチャーが開始します。Runtime または Power-Up Trigger では、キャプチャーされたデータの読み出しおよび転送をオンチップ・バッファーから .stp 対して行い、それを解析に使用します。

関連情報
Signal Tap Logic Analyzer の実行 (73 ページ)
2.2.7. キャプチャーしたデータの表示、解析、および使用

キャプチャーして .stp ファイルに読み出されたデータは、解析とデバッグに使用することができます。データは、保存して後で解析したり、他のフォーマットに変換して共有やさらなる調査を行ったりすることも可能です。

- キャプチャーした信号データの読み出しや解釈の簡素化には、ニーモニック・テーブルの設定を手動またはプラグインを使用して行います。
- デバッグの処理速度を上げるには、Signal Tapnode リストの Locate 機能を使用して、インテル Quartus Prime 開発ソフトウェアの他のツール内で問題のあるノードの箇所を検索します。

関連情報
キャプチャーしたデータの表示、解析、および使用 (77 ページ)

2.3. Signal Tap Logic Analyzer のコンフィグレーション

Signal Tap Logic Analyzer のインスタンスのコンフィグレーションは、Signal Tap Logic Analyzer ウィンドウの Signal Configuration ペインで行います。

図 -16: Signal Tap Logic Analyzer の Signal Configuration ペイン

Signal Tap Logic Analyzer のインスタンスのコンフィグレーションには、クロック信号を割り当ててください。

ロジック・アナライザーでは、データをサンプリング取得クロックのすべての正の（立ち上り）エッジで行います。ロジック・アナライザーでは、取得クロックの負の（立ち下がり）エッジでのサンプリングはサポートしていません。

デザイン内の信号はいずれも取得クロックとして使用できますが、最良のデータ取得結果を得るためには、テスト対象の信号と同期したグローバルな非ゲートクロックを使用してください。ゲートクロックを取得クロックとして使用すると、結果が予期せぬデータとなり、デザインのエラービアを正しく反映しない場合があります。インテル Quartus Prime のスタティック・タイミング解析ツールでは、取得クロックの最大周波数が示され、その周波数でデザインを実行できます。ロジック・アナライザー・クロックの最大周波数を確認するには、Compilation Report の Timing Analysis のセクションを参照してください。
注意: 注意が必要なのは、トランシーバーからのリカバリーロックをSignal Tap Logic Analyzerの取得クロックとして使用する場合です。リカバリーロックが不正確または予期しないビヘイビアの原因となることがあり、トランシーバーのリカバリーロックがパワーアップ・トリガー機能を備えた取得クロックである場合は特にそうです。

取得クロックを割り当てにSignal Tap Logic Analyzer Editorを使用しない場合は、インテルQuartus Prime開発ソフトウェアによってauto_stp_external_clkという名前のクロックピンが自動作成されます。このピンにピン・アサインメントを作成し、デザインのクロック信号によって取得クロックが正しく駆動されることを確認してください。

関連情報
- プラグインを使用した信号の追加 (31ページ)
- Managing Device I/O Pins
  Design Constraints User Guide: Intel Quartus Prime Pro Edition内

2.3.2. Signal Tap ファイルへの信号の追加

モニタリングする信号を.stpノードリストに追加します。トリガーを定義する信号を選択することも可能です。次の2つの信号タイプを割り当てます。

- Pre-synthesis - この信号が存在するのは、デザイン・エラボレーション後ですが、合成最適化前であります。この信号のセットは、Register Transfer Level (RTL)信号を反映する必要があります。
- Post-fitting : この信号が存在するのは、物理合成最適化および配置配線の後です。インテルQuartus Prime開発ソフトウェアでは、Signal Tapウィンドウの波形図表示でのモニタリングに使用可能な信号数の制限はありませんが、使用可能なチャネル数は、デバイス内のロジック・モジュール（ALM）の個数に比例します。したがって、モニタリングに使用できるチャネル数には物理的制限があります。青色の文字で表示されている信号は、フィッティング後のノード名です。黒色の文字で表示されている信号は、合成前のノード名です。

注意: インテルQuartus Primeプロ・エディション開発ソフトウェアでは、インスタンス名のみを使用し、エンティティ名は使用しません。形式は次のとおりです。

a|b|c

a_entity:a|b_entity:b|c_entity:cではありません。

Analysis and Elaborationが正常に終了すると、無効な信号が赤色で表示されます。これらの信号が有効であることが確認できない限りは、.stpファイルからこれらの信号を削除し、正しく動作させます。Signal Tap Status Indicatorでは、無効なノード名が.stpファイルに存在するかどうかを表示します。

信号のタッピングが可能なお、配線リソース（ロウまたはカラムの相互接続）が存在し、接続の配線がSignal Tapインスタンスに対して行われる場合です。例えば、I/Oエレメント（IOE）内に存在する信号をタップできない理由は、IOE内の信号からコア・ロジック・エレメントへの直接配線リソースが存在しないからです。入力ピンの場合、ロジック・アレイ・ブロック（LAB）を駆動する信号のタッピングができます。出力ピンの場合、IOEを駆動するLABからの信号のタッピングができます。

関連情報
Setup Tab (Signal Tap Logic Analyzer)
Intel Quartus Prime Help内
### 2.3.2.1. 合成前の信号

合成前の信号を追加する場合は、Signal Tap Logic Analyzer へのすべての接続を合成前に行います。Compiler では、ロジックおよび配線リソースを割り当てて、デザインファイルを変更した場合と同様に接続を確立します。IOE の内外に駆動する信号では、合成前の信号名はピンの信号名と一致します。

### 2.3.2.2. ポストフィット信号

ポストフィット信号をタップすると、ポストフィット・ネットリストの実際のアトムに接続している状態になります。タップできるのは、ポストフィット・ネットリストに存在している信号のみで、既存の配線リソースが使用可能である必要があります。

ポストフィット出力信号の場合は、IOE ブロックを駆動する COMBOUT または REGOUT 信号をタップします。ポストフィット入力信号の場合、コアロジックに駆動する信号はピンの信号名と一致します。

**注意:** NOT ゲート・プッシュ・バックはタップしたレジスターに適用されるため、アトムからの信号が反転する可能性があります。これを確認するには、信号の検索を Resource Property Editor または Technology Map Viewer のいずれかで行います。また、Technology Map ビューアーと Resource Property Editor を使用して、ポストフィット・ノード名を検索することもできます。

### 関連情報

- Design Flow with the Netlist Viewers

#### 2.3.2.2.1. Technology Map Viewer を使用したデータ信号の割り当て

Technology Map Viewer では、ポストフィット信号の追加ができます。

1. コンパイル後、Technology Map Viewer をインテル Quartus Prime 開発ソフトウェアで起動します、Tools > Netlist Viewers > Technology Map Viewer (Post-Fitting) の順にクリックします。
2. タップしたいノードを検索します。
3. ノードをデザインのアクティブな .stp または新しい .stp にコピーします。

### 2.3.2.3. 信号の保持

インテル Quartus Prime 開発ソフトウェアで提供している合成属性では、Compiler による特定の信号の最適化の実行を防止し、それをポストフィット・ネットリストに保持できるようにします。

インテル Quartus Prime ソフトウェアでは、RTL 信号の最適化を合成中および配置配線中に行います。RTL 信号名は、最適化後のポストフィット・ネットリストに表示されない場合があります。

最適化属性は次のとおりです。
- **keep** - 組み合わせ信号が最適化中に削除されるのを防止します。
- **preserve** - レジスターが最適化中に削除されるのを防止します。

ただし、属性を保存すると、デバイスリソースの使用率が増加したり、タイミング・パフォーマンスが低下したりする可能性があります。

ノードの保持が必要になることが多いのは、IP の信号グループを追加するのにプラグインを使用する場合です。Nios II CPU などの暗号化された IP コアをデバッグしている場合、ノードをコアから保持し、それを保存して Signal Tap Logic Analyzer でデバッグできるようにすることが必要な場合があります。
2.3.2.4. ノードリスト信号の使用オプション

ノードリストに信号を追加する際、オプションを選択して、ロジック・アナライザーによる信号の使用方法を指定します。

信号による解析のトリガーを防止するには、その信号のTrigger Enable オプションを .stp ファイル内に無効にします。このオプションが有用なのは、信号でキャプチャーしたデータだけを表示したい場合です。

信号表示機能をオフにするには、Data Enable カラムを .stp ファイルでインナーブルします。このオプションが有用なのは、信号のトリガーの実行時に、信号のデータ表示が必要な場合です。

関連情報
トリガーの定義 (44 ページ)

2.3.2.4.1. Signal Tap インスタンスのインナーブルとディスエーブル

Signal Tap インスタンスをインナーブル/ディスエーブルは、Instance Manager ペインで実行します。インスタンスを物理的に追加・削除するには、Signal Tap インスタンスのディスエーブル/インナーブル後に再コンパイルを実行する必要があります。

2.3.2.5. Signal Tap デバッグに使用できない信号

デザイン内のポストフィット信号の中には、Node Finder ダイアログボックスの Signal Tap: post-fitting filter にないものもあります。

次の信号タイプはいずれもタッピング不可です。

- ポストフィット出力ピン - ポストフィット出力ピンを直接タップすることはできません。出力信号を表示するには、出力ピンを駆動するレジスターまたはパッファーをタップします。これには、双方向として定義されるピンが含まれます。

- キャリーチェーンの一部としての信号 - ロジックエレメントのキャリーアウト (cout0 または cout1) 信号をタップすることはできません。アーキテクチャー上の制限のため、キャリーアウト信号でフィードできるのは別の LE のキャリーインのみです。

- JTAG 信号 - JTAG コントロール (TCK、TDI、TDO、および TMS) 信号をタップすることはできません。

- ALTGXB IP カラ - ALTGXB のインスタンス化のポートは直接タップすることはできません。

- LVDS - シリアルライザ/デシリアルライザ (SERDES) ブロックから出力されたデータをタップすることはできません。

- DQ および DQS 信号 - DDR/DDRII デザインで DQ や DQS 信号を直接タップすることはできません。

2.3.3. プラグインを使用した信号の追加

個別の信号またはグループ化された信号の追加を Node Finder にする代わりに、プラグインを使用して特定タイプの IP に関連する信号グループの追加ができます。信号を簡単に追加できる機能のほかに、プラグインで提供される機能としては、トリガー作成とデータ表示に役立つデザイン済みのコードを展開する機能などがあります。Signal Tap Logic Analyzer に付属のプラグインは、Nios II プロセッサー用です。
Nios II プラグインでは、次の例のように、Setup タブで 1 つのニーモニック・テーブル、Data タブで 2 つのニーモニック・テーブルを作成します。

- **Nios II Instruction (Setup タブ)** - すべての必要な信号をキャプチャーして、選択した命令アドレスでのトリガーに使用します。
- **Nios II Instance Address (Data タブ)** - 実行された命令のアドレスの表示は、オプションの Executable and Linking Format (.elf) ファイルで定義される場合は 16 進数、またはプログラミング・シンボル名としています。
- **Nios II Disassembly (Data タブ)** - 対応するアドレスから逆アセンブルしたコードを表示します。

信号を .stp ファイルに追加するのにプラグインを使用するには、デザインで Analysis and Elaboration を実行した後で次の手順を実行します。

1. 必要な信号がすべて使用可能であることを確認するには、インテル Quartus Prime 開発ソフトウェアで Assignments > Settings > Compiler Settings > Advanced Settings (Synthesis) の順でクリックします。Create debugging nodes for IP cores をオンにします。
   - プラグインに含まれるすべての信号がノードリストに追加されます。
2. ノードリストを右クリックします。Add Nodes with Plug-In サブメニューで使用するプラグイン名を選択します。例えば、搭載されているプラグインで、名前は Nios II です。
   - Select Hierarchy Level ダイアログボックスが表示され、デザインの IP 階層が確認できます。
   - 選択したプラグインの IP がデザインに存在しない場合、メッセージ表示によって選択したプラグインが使用できないことが分かります。
3. モニタリングする信号が含まれている IP をプラグインで選択して、OK をクリックします。
   - プラグインのすべての信号が使用可能な場合は、選択したプラグインによってはダイアログボックスが表示され、そこでプラグインに対して使用可能なオプションを設定できます。
4. Nios II プラグインを使用すると、.elf をオプションで選択することができます。これには、Nios II Integrated Development Environment (IDE) ソフトウェア・デザインのプログラムシンボルが含まれます。オプション指定を選択したプラグインに対して行い、OK をクリックします。

関連情報

- トリガーの定義 (44 ページ)
- キャプチャーしたデータの表示、解析、および使用 (28 ページ)

### 2.3.4. サンプル深度の指定

Sample depth 設定で指定するサンプル数は、Signal Tap Logic Analyzer でキャプチャーして格納したものです。この指定は、キャプチャーしたデータパッファーの各信号に対して行います。サンプル深度の指定には、次を実行します。

1. 希望の数を Sample Depth ドロップダウン・メニューで選択します。
   - サンプル容量の範囲は 0 から 128K です。

デバイスのメモリリソースに制限がある場合、コンパイルの実行が、選択したサンプルパッファーのサイズが原因でできないことがあります。その場合は、サンプル深度を減らしてリソース使用量を抑えてみてください。
関連情報
Signal Configuration Pane (View Menu) (Signal Tap Logic Analyzer)
Intel Quartus Prime Help 内

2.3.5. 特定の RAM タイプへのデータのキャプチャー

Signal Tap ロジック・アナライザーが取得データを格納する RAM タイプを選択するオプションが選択可能です。特定の RAM ブロックに Signal Tap ロジック・アナライザーを割り当てる場合、RAM ブロック全体がロジック・アナライザーの専用リソースとなります。

RAM を選択すると、デザインの特定のメモリーブロックを保持し、メモリーの別領域を Signal Tap ロジック・アナライザーのデータ取得に割り当てることができます。

たとえば、大量のメモリーやデータに必要であるブロックを選択する場合、およびデータを別のブロックで保持する必要がある場合があります。これにより、RAM を選択することで、メモリーの構成を効率的に使用することができます。

Signal Tap ロジック・アナライザーのバッファーウィンドウの Signal Configuration を開き、ドロップダウンメニューから 1 つの Ram type を選択します。

この機能は、取得したデータが選択した RAM タイプで使用可能なメモリーよりも小さい場合にのみ使用できます。データの量は、Signal Tap の Resource Estimator に表示されます。

関連情報
Signal Configuration Pane (View Menu) (Signal Tap Logic Analyzer)
Intel Quartus Prime Help 内

2.3.6. バッファー取得モードの選択

キャプチャーされたデータをバッファーをロジック・アナライザーで編成する方法を指定すると、データ取得のために Signal Tap に必要なメモリーリソースを低減することができます。

Signal Tap Logic Analyzer には、非セグメント化 (または円形) バッファーとセグメント化バッファーの 2 種類のアクvizion・バッファーがあります。

- 非セグメント化バッファーの場合は、Signal Tap Logic Analyzer では、メモリー領域全体を单一の FIFO として処理し、バッファーの充填を行います。充填は、ロジック・アナライザーが一連の定義済みトリガー条件に達するまで続きます。

- セグメント化バッファーの場合、メモリー領域は分割されて別々のバッファーになります。各バッファーヒ、独自のトリガー条件セットを持つ個々の FIFO として機能し、非セグメント化バッファーとして動作します。1 つのバッファーだけが、取得中にアクティブです。Signal Tap Logic Analyzer が次のセグメントに進むのは、トリガー条件またはアクティブセグメントの条件達成後です。

非セグメント化バッファーを使用する場合は、Storage Qualification 機能を使用して、アクvizion・バッファーに書き込むサンプルを決定します。セグメント化バッファーと Storage Qualification 機能を備えた非セグメント化バッファーは両方とも、使用可能なメモリー領域の最大限の活用に役立ちます。
図 -17: Signal Tap Logic Analyzer で使用するバッファータイプの比較
次の図に示すのは、2つのバッファータイプの違いです。

非セグメント化バッファーとセグメント化バッファーの両方で、プリセットのトリガーよりトリガが発生する前（Pre-Trigger）、中心（Center Trigger）、トリガーよりトリガが発生した後（Post-Trigger）のデータをキャプチャーできます。もしくは、State-Based Triggering タブを使用してカスタムのトリガーポジションの指定も可能です。詳細は、トリガーポジションの指定を参照してください。

関連情報
• トリガー位置の指定 (65 ページ)
• 関連サンプルのフィルタリング (36 ページ)

2.3.6.1. 非セグメント化バッファー
非セグメント化バッファーは、Signal Tap Logic Analyzer のデフォルトのバッファータイプです。

ランタイムのロジック・アナライザーによるバッファ内へのデータの格納は、バッファが満たされるまで実行されます。その時点で新たに新しいデータが格納されるために、得られたデータが上書きされ、特定のトリガーイベントが発生するまで続きます。バッファーでキャプチャーするトリガーイベント後のデータは、Trigger position 設定により次のよう異なります。

• データの大半をトリガー発生前にキャプチャーするには、Post trigger position を選択します。
• データの大半をトリガー後にキャプチャーするには、Pre trigger position を選択します。
• トリガー位置をデータ内で中心に配置するには、Center trigger position を選択します。

もしくは、カスタムの State-based trigger フローを使用して、カスタムのトリガーポジションの定義をキャプチャーバッファ内で行うことができます。

関連情報
トリガー位置の指定 (65 ページ)

2.3.6.2. セグメント化バッファー
セグメント化バッファーでは、取得メモリーは分割されて同一サイズのセグメントになります。トリガー条件セットの定義を各セグメントに対して行います。各セグメントは、非セグメント化バッファーとして機能します。セグメント化バッファーの使用によりデバッグできるシステムにおいて、比較的低頻度で繰り返されるイベントが含まれています。

個別のトリガー条件の設定をセグメント化バッファーより行う場合は、State based trigger フローを使用してください。この図で示すのは、セグメント化バッファーシステムの一例です。
図-18: 繰り返しイベントを生成するシステム
このデザインでは、正しいデータがSRAMコントローラーに確実に書き込まれるようにするため、RDATAポートのモニタリングは、アドレスH'0F0F0F0FがRADDRポートに送信されるたびに行います。

パッファー取得機能を使用すると、SRAMデバイスからの複数の読み出しトランザクションをモニタリングできます。このとき、Signal Tap Logic Analyzerを再度実行する必要はありません。これは、メモリを分割して、同じイベントを複数回キャプチャーするからで、割り当てられたメモリを浪費することはありません。パッファーでキャプチャーするセグメント数は、Signal ConfigurationペインのData設定で定義したサイクル数と同じです。

パッファー取得をイネーブル/コンフィグレーションするには、Signal Tap Logic Analyzer EditorでSegmentedを選択し、使用するセグメント数を決定します。図の例では、64個のサンプルセグメントを選択しているので、64個の読み出しサイクルがキャプチャーできます。

関連情報
セグメント化パッファーを使用したデータのキャプチャー（77ページ）

2.3.7. パイブライン設定の指定

Pipeline factor設定で表示されるパイプラインレジスターの個数は、Signal Tap開発ソフトウェアで追加可能で、これによりインテルQuartus Prime Logic AnalyzerのfMAXがブーストされます。

Signal Tap GUIからPipeline factorを指定するには、次を実行します。

• Signal Configurationペインで、Pipeline factorの指定を0から5の範囲で行います。デフォルト値は0です。

注意: Pipeline factorの設定により、fMAXが必ずしも増加するわけではないかもしれません。これは、パイプラインレジスターがクリティカルパス内に存在しないことがあるためです。

注意: Signal TapインテルFPGA IPの最適化は、インテルStratix®10アーキテクチャーに対しては行われません。

2.3.7.1. プラットフォーム・デザイナー（旧：Qsys）からのパイブライン設定の指定

Pipeline factor設定で個数表示されるパイプラインレジスターを追加して、Signal Tap Logic AnalyzerのfMAXのブーストができます。

注意: Signal TapインテルFPGA IPの最適化は、インテルStratix10アーキテクチャーに対しては行われません。
Pipeline factor の指定は、Signal Configuration ペインで行うことができます。Pipeline factor の範囲は 0 から 5 で、デフォルト値は 0 です。

Pipeline factor の指定を、プラットフォーム・デザイナー (旧: Qsys) Logic Analyzer コンポーネントのインスタンス化を Signal Tap システムからすると行うには、次を実行します。
1. IP Catalog で Signal Tap Logic Analyzer コンポーネントをダブルクリックします。
2. Pipeline Factor を指定し、同時に他のパラメーター値も指定します。

図 -19: プラットフォーム・デザイナー (旧: Qsys) からの Pipeline Factor の指定

2.3.8. 関連サンプルのフィルタリング

Storage Qualifier 機能を使用すると、デザインのデバッグに関連のない個々のサンプルをフィルターすることができます。

Signal Tap Logic Analyzer によるスナップショットの提供は、アクイジョン・バッファーへのデータ格納時に行われます。デフォルトでは、Signal Tap Logic Analyzer による取得メモリーへのデータサンプルの書き込みは、すべてのクロックサイクルで行われます。非セグメント化バッファーの場合、1 つのデータウィンドウによってデータストリームの包括的スナップショットが表わされます。対反に、セグメント化バッファーの場合は、複数のより小さなサンプリング・ウィンドウをより長時間にわたって使用し、各サンプリング・ウィンドウによって連続データセットが表わされます。

アクイジョン・バッファーを使用した解析の場合は、選択した信号セットの機能エラーの大半をキャプチャーできます。これは、トリガー条件が適切で、サンプル深度が取得に十分な場合です。ただし、各データウィンドウには、かなりの量の不要なデータが含まれていることがあります。例えば、データバースト間の長周期アイドル信号などがです。Signal Tap Logic Analyzer のデフォルトのビヘイビアでは、冗長なサンプルビットは破棄しません。

Storage Qualifier 機能では、条件を確立し、それをバッファーへの書き込みイネーブルとしてデータ取得の各クロックサイクル中に動作させることができるので、より効率的な取得メモリーの使用が、より長時間にわたる解析において可能になります。

不連続性の作成は、バッファー内のどの 2 つのサンプル間でも可能です。このため、Storage Qualifier 機能は、カスタムのセグメントバッファーを作成し、そのバッファー内でセグメント境界の数およびサイズが調整可能であることと同じです。

注意: Storage Qualifier 機能が使用可能なのは、非セグメント化バッファーでのみです。IP Catalog フローでサポートしているのは、Storage Qualifier 機能の Input Port モードのみです。
図 -20: アクイジョン・バッファー制御の異なるモードを使用したデータ取得

注:
1. 非セグメント化バッファーでキャプチャーするのは、連続したデータの固定サンプルウィンドウです。
2. セグメント化バッファーでは、バッファーは分割されて固定サイズのセグメントになります。各セグメントのサンプル深度は同じです。
3. Storage Qualifier 機能では、カスタムのサンプリング・ウィンドウの定義を資格条件を持つ各セグメントに対して行うことができるので、より大きい範囲のタイムスケールが可能になることがあります。

次の 6 種類の Storage Qualifier が Storage Qualifier 機能で使用可能です。
- Continuous (デフォルト) Storage Qualifier をオフにします。
- Input port
- Transitional
- Conditional
- Start/Stop
- State-based
図 -21: Storage Qualifier の設定

取得開始時に、Signal Tap Logic Analyzer では各クロックサイクルを調査し、バッファへのデータ書き込みを Storage Qualifier の種類と条件をもとに行います。取得が停止するのは、定義されたトリガー条件が発生したときです。

Signal Tap ロジック・アナライザーでは、Storage Qualifier 条件のトリガー条件を個別に評価します。

関連情報
トリガー条件の定義 (27 ページ)

2.3.8.1. Input Port モード

Input Port モードを使用すると、Signal Tap Logic Analyzer では、デザインからの信号を入力として受け取ります。取得実行中、信号がクロックエッジで High の場合、Signal Tap Logic Analyzer ではデータをバッファに格納します。信号がクロックエッジで Low の場合、そのデータサンプルは無視されます。内部ノードを指定しない場合、Logic Analyzer ではこの入力ポートにピンを作成して接続します。

.stp ファイルから Signal Tap Logic Analyzer のインスタンスを作成する場合、Storage Qualifier 信号を指定するために、Setup タブに位置する Input port 領域を使用します。プロジェクトをコンパイルする際には、このポートを指定してください。

パラメーター・エディターを使用する場合、Storage Qualification 入力ポートは、指定があれば、生成されたインスタンス化テンプレートに表示されます。その後、このポートを RTL の信号に接続します。

図 -22: 繰り返しデータパターンのデータ取得における Continuous モードと Input Port Capture モードの比較

- Continuous モード:

- Input Port Storage Qualifier:
2.3.8.2. Transitional モード

Transitional モードの場合、Logic Analyzer では、信号セットの変化をモニタリングします。新しいデータをアクイジョン・バッファーへ書き込むのは、変化の検出後のみです。モニタリングする信号の選択には、Storage Qualifier カラムのチェックボックスを使用します。

図 -23: Transitional Storage Qualifier の設定

図 -24: 繰り返しデータパターンのデータ取得における Continuous モードと Transitional Capture モードの比較

- Continuous モード:

- Transitional モード:

冗長なアイドルサンプルは破棄されます。

2.3.8.3. Conditional モード

Conditional モードの場合、Signal Tap Logic Analyzer では、サンプルを保存するかどうかを決定するために、ノードリスト内の定義済み信号の組み合わせ関数を評価します。Signal Tap Logic Analyzer でのバッファーへの書き込みは、クロックサイクル中に行います。ここでは、指定した条件の評価が TRUE です。

Basic AND, Basic OR, Comparison、または Advanced Storage Qualifier 条件のいずれかを選択できます。Basic AND または Basic OR 条件では、各信号を次のいずれかに一致させます。

- Don’t Care
- Low
- High
- Falling Edge
- Rising Edge
- Either Edge
Basic AND storage qualifier 条件を複数の信号に対して指定した場合、Signal Tap Logic Analyzer ではその条件の論理 AND を評価します。

他の組み合わせまたは関係演算子の指定は、Storage Qualification 用にイネーブルされた信号セットを使用して、高度なストレージ条件で処理します。

Storage Qualification 条件の定義は、トリガー条件の定義と同様の方法で行います。

図 - 25: **Conditional Storage Qualifier の設定**
この図で表すのは、.stp ファイル内の Conditional Storage Qualifier 設定の詳細です。

図 - 26: 繰り返しデータパターンのデータ取得における Continuous モードと Transitional Capture モードの比較
データパターンはどちらの場合でも同じです。
- Continuous sampling capture モード：
- Conditional sampling capture モード：

関連情報
- 基本的なトリガー条件 (44 ページ)
- Comparison トリガーの条件 (46 ページ)
- アドバンスド・トリガー条件 (48 ページ)
2.3.8.4. Start/Stop モード

Start/Stop モードで使用する条件は、データ・キャプチャーの開始とデータ・キャプチャーの停止の 2 つです。開始条件による評価が TRUE の場合、Signal Tap Logic Analyzer では、各クロックサイクルでバッファーデータの格納を行います。これは、停止条件による評価が TRUE になるまで続き、その後データ・キャプチャーは一時停止します。Logic Analyzer では、追加のスタート信号の受信がデータ・キャプチャー開始後にされた場合は、その信号は無視されます。開始と停止の両方での評価が同時に TRUE になった場合、Logic Analyzer でキャプチャーするのは 1 つのサイクルです。

注意: トリガーの制御には Stop ボタンを押します。これは、バッファーの完了が、停止条件が理由で正常に行われない場合に実行します。

図 -27: Start/Stop モードの Storage Qualifier 設定

図 -28: 繰り返しデータパターンに対する Continuous モードと Start/Stop Acquisition モードの比較

- Continuous モード:

- Start/Stop Storage Qualifier:

2.3.8.5. State-Based

State-Based Storage Qualification モードは、ステートベース・トリガー・フローの一部です。ステートベース・トリガー・フローでは、条件付き制御を評価して、Signal Tap Logic Analyzer によるバッファーエージェントデータの書き込み方法を定義します。ステートベース・トリガー・フローでは、ブール値演算子および関係演算子を使用して、ターゲットとするアクイジョン・バッファーの実行フローを導きます。
Storage Qualifier 機能のイネーブルをステートベース・フローに対して行うと、`start_store` と `stop_store` の 2 つのコマンドが使用可能となります。このコマンドは、Start/Stop キャプチャー条件に似ています。取得開始時、Signal Tap Logic Analyzer によるバッファへのデータ書き込みは、`start_store` アクションの実行までは行われません。`stop_store` コマンドでは取得を一時停止します。`start_store` と `stop_store` のアクションが同一サイクル内で発生した場合、Logic Analyzer では、単一のサンプルをアクイジョン・バッファに格納します。

関連情報
State-Based Triggering (58 ページ)

### 2.3.8.6. データの不連続性の表示

`Record data discontinuities` をオンにすると、Signal Tap Logic Analyzer ではサンプルをマークし、その間、Storage Qualifier からの取得を一時停止します。このマーカーの波形ビューでの表示は、収集完了後にされます。

### 2.3.8.7. Storage Qualifier のディスエーブル

`Disable Storage Qualifier` オプションで Storage Qualifier をすばやくオフにして、連続キャプチャーの実行ができます。このオプションでは、リアルタイムでのリコンフィグレーションが可能です。Storage Qualifier モードを `Type` フィールドから変更するには、プロジェクトを再コンパイルが必要です。

関連情報
Runtime Reconfigurable オプション (74 ページ)

### 2.3.9. 複数の Signal Tap ファイルとコンフィグレーションの管理

デザイン内のさまざまなブロックのデバッグには、関連するモニタリング信号のグループ化を行います。同様に、信号のグループを使用して、複数のトリガー条件を定義することも可能です。信号、キャプチャー設定、およびトリガー条件の各組み合わせによってデバッグ・コンフィグレーションが決定され、1 つのコンフィグレーションには 0 またはそれ以上の関連データログを持たせることができます。

Signal Tap Logic Analyzer では、デバッグ・コンフィグレーションを複数の `.stp` ファイルに保存することができます。また、複数のコンフィグレーションと同じ `.stp` ファイルに埋め込み、Data Log を管理ツールとして使用することも可能です。

**注意:** 各 `.stp` ファイルは、プログラミング (.sof) ファイルに関連付けられています。正常に動作させるには、ランタイムに使用する `.stp` ファイルの設定を、デバイスのプログラミングに使用する `.sof` ファイルの Signal Tap 設定と一致させる必要があります。

関連情報
`.stp` ファイルと `.sof` ファイル間の互換性の設定確認 (73 ページ)

### 2.3.9.1. Data Log ペイン

Data Log ペインで表示されるのは、単一の `.stp` ファイル内に格納されたすべての Signal Tap コンフィグレーションとデータ・キャプチャーの結果です。
• 現在のコンフィグレーションあるいはキャプチャーを Data Log および .stp ファイルに保存するには、Edit > Save to Data Log の順にクリックします。もしくは、Data Log ペイン上部の Save to Data Log アイコン をクリックします。

• ログエントリーの生成をすべてのデータ・キャプチャー後に行うには、Edit > Enable Data Log の順にクリックします。もしくは、Data Log ペイン上部のボックスをチェックします。

Data Log では、その内容をツリー階層で表示します。アクティブなアイテムには別のアイコンが表示されます。

表 4. Data Log のアイテム

<table>
<thead>
<tr>
<th>項目</th>
<th>アイコン</th>
<th>1 つ以上含まれるもの</th>
<th>備考</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>非選択時</td>
<td>選択時</td>
</tr>
<tr>
<td>Instance</td>
<td>🎉</td>
<td>Signal Set</td>
<td></td>
</tr>
<tr>
<td>Signal Set</td>
<td>🎉</td>
<td>Trigger</td>
<td>Trigger が変更されるのは、Signal Set に新しい信号を追加したときです。Signal Set の変更後は再コンパイルが必要です。</td>
</tr>
<tr>
<td>Trigger</td>
<td>🎉</td>
<td>Capture Log</td>
<td>Trigger が変更されるのは、トリガー条件を変更したときです。このような変更には再コンパイルは不要です。</td>
</tr>
<tr>
<td>Capture Log</td>
<td>🎉</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

各エントリー上の名前に表示されるのは、Signal Tap ロジック・アナライザーがトリガーしたときのウォールクロック・タイムと、取得開始からアクティブ・ビジョンのトリガーまでの経過時間です。エントリー名は変更しても、識別しやすいものにすることが可能です。

コンフィグレーションの切り替えには、Data Log のエントリーをダブルクリックします。その結果、Setup タブが更新され、アクティブな信号リストとトリガー条件が表示されます。

例-1: シンプルな Data Log

この例では、Data Log で表示する 1 つのインスタンスには、3 つの信号セットのコンフィグレーションがあります。

2.3.9.2. SOF Manager

SOF Manager は JTAG Chain Configuration ペイン内にあります。
SOF Managerを使用すると、複数のSOFを1つの.stpファイルに埋め込むことができます。このアクションにより、.stpファイルを同じコンピューターあるいはネットワーク間の異なるロケーションに移動できます。このとき、関連する.sofを個別に含める必要はありません。新規のSOFを.stpファイルに埋め込むには、Attach SOF Fileアイコンをクリックします。

図-29: SOF Manager

コンフィグレーションの切り替えをData Logで行うと、特定のコンフィグレーションと互換性のあるSOFの抽出ができます。

新規のSOFをFPGAにダウンロードするには、SOF ManagerでProgram Deviceアイコンをクリックします。これは、.stpのコンフィグレーションがターゲットデバイスにプログラミングされたデザインと適合していることを確認した後に行ってください。

関連情報
Data Logペイン(42ページ)

2.4. トリガーの定義

ランタイムにSignal Tap Logic Analyzerでは、アクティビティーのサンプリングをモニターした信号から継続して行います。トリガーがアクティブ化される、つまりロジック・アナライザーが停止してデータを表示するのは、モニターサした信号が、1つまたは複数の指定条件に達したときです。トリガー条件の指定は、Signal Tap Logic AnalyzerのSignal Configurationペインで行ってください。

2.4.1. 基本的なトリガー条件

Basic ANDまたはBasic Orのトリガータイプを選択した場合、トリガーパターンの指定を.stpに追加した各信号に対して行う必要があります。トリガーパターンの指定には、Trigger Conditionsカラムを右クリックして、希望のパターンをクリックします。トリガーパターンの設定は次のいずれかの条件にします。

- Don’t Care
- Low
- High
- Falling Edge
- Rising Edge
- Either Edge
バスに対しては、パターンの入力をバイナリであるか、あるいは右クリックして Insert Value を選択し、パターンの入力に他の数値形式を使用します。X を入力すると、「don't care」値のセットの指定を 16 进数またはバイナリー文字列のいずれかですることができます。関連するニーモニック・テーブルを持つ .stp ファイルの信号に対しては、右クリックしてテーブルからエントリーを選択し、トリガー用の定義済み条件を指定します。

信号の並加にプラグインを使用する場合、基本トリガーの作成には、定義済みのニーモニック・テーブル・エントリーを使用します。例えば、Nios II プラグインの場合、.elf ファイルの指定を Nios II IDE デザインからするには、ファクション名の入力を Nios II コードからします。ロジック・アナライザーがトリガーされるのは、Nios II 命令アドレスが、指定したコードのファクション名のアドレスと一致したときです。

データのキャプチャーが停止し、Logic Analyzer によってデータがバッファーに保存されるのは、すべての信号の論理 AND の評価が TRUE になったときです。

関連情報
キャプチャーしたデータの表示、解析、および使用 (77 ページ)

2.4.1.1. ネストされたグループによる Basic OR トリガー条件の使用

信号セットをネストされたグループ（グループのグループ）として指定するのに Basic OR トリガーイープを使用すると、Signal Tap Logic Analyzer ではアドバンスド・トリガー条件を生成します。この条件では、グループ内の信号を並び替え、デザインを再コンパイルする必要性を最小限に抑えます。ノードの親子関係が一定に保たれている限り、アドバンスド・トリガー条件は変更されません。ノードの兄弟関係の変更は可能で、その際、デザインの再コンパイルは不要です。

ネストされたトリガー条件の評価優先順位の開始は、リーフグループを持つ最下位レベルからです。Logic Analyzer は、結果として得られる論理値を使用して、親グループの論理値を計算します。グループの値を手動で設定する場合、グループメンバーの論理値による影響は、グループトリガーの結果に対してもありません。ネストされたトリガー条件を作成するには次を実行します。

1. Trigger Conditions で Basic OR を選択します。
2. Setup タブでノードをいくつか選択します。複数のグループを選択に含めます。
3. Setup タブを右クリックして Group を選択します。
4. ネストされたグループを選択し、右クリックして、Reduction AND、OR、NAND、NOR、XOR、XNOR、あるいは論理 TRUE または FALSE を適用するグループトリガー条件を設定します。

注意: OR および AND グループトリガー条件のみを、下位レベルのグループ（子としてのグループを持たないグループ）に対して選択します。
2.4.2. Comparison トリガーの条件

Comparison トリガーを使用すると、バスの複数のグループ化ビットと予想整数値との比較が可能になります。これには、簡単な比較条件をバスノード上で指定します。Comparison トリガーでは、Basic OR に含まれるすべてのトリガー条件を保持します。Comparison トリガーの使用には、他のトリガーとの組み合わせが可能です。また、Basic OR トリガーと Comparison トリガーの切り替えをランタイムすることが可能で、このとき再コンパイルは不要です。

Signal Tap Logic Analyzer では、次のタイプの Comparison トリガー条件をサポートしています。

- **Single-value comparison** - バスノードの値と指定した数値を比較します。比較には、次のオペランドの 1 つを使用します。>, >=, =, <=, <。バスノードが指定した数値と一致すると、1 が返されます。
- **Interval check** - バスノードの値が定義したインターバル内であるかどうかを検証します。ノードの値が指定した範囲のインターバル内にあれば、1 が返されます。

次の規則に従って、Comparison トリガー条件を使用してください。

- Comparison トリガーの適用は、リーフノードで構成されるバスノードに対してのみです。
- サブグループをバスノード内に形成しないでください。
- トリガーノードの個別イネーブル/ディスエーブルをバスノード内でしないでください。
- 比較値 (Single-value comparison の場合) または境界値 (Interval check の場合) の指定の際は、選択したノードのバス幅を超えないようにしてください。

2.4.2.1. Comparison トリガー条件の指定

次の手順に従って、Comparison トリガー条件を指定してください。

1. Setup タブから Trigger Conditions の Comparison を指定します。
2. トリガーエディター内のノードを右クリックして、Compare を選択します。
図 -31: Comparison トリガー条件の選択

3. Compare ウィンドウから Comparison type を選択します。
   - Single-value comparison を Comparison type に選択する場合、オペランドと値を指定します。
   - Interval check を Comparison type に選択する場合、インターバルの下限と上限の境界値を入力します。
     また、境界値を含めるか除外するかを指定することもできます。

図 -32: Comparison Value (比較値) の指定

4. OK をクリックします。トリガーエディターでは、結果の比較式を Group Node Condition テキストボックスに表示します。
   注意: 比較条件の変更は、テキストボックスで有効な式を使用して行います。
2.4.3. アドバンスド・トリガー条件

データのキャプチャーを所定の条件の組み合わせに対して行うには、アドバンスド・トリガーを作成します。Signal Tap Logic AnalyzerのAdvanced Triggerタブは、GUIを使用した複雑なトリガー式の作成に役立ちます。

Advanced Triggerタブを開くには、Trigger Conditionsドロップダウン・メニューのAdvancedを選択します。

複雑なトリガー条件を式ツリーで構築するには、演算子をObject LibraryペインとNode ListペインからAdvanced Trigger Configuration Editorウィンドウにドラッグアンドドロップします。
演算子の設定のコンフィグレーションを行うには、配置した演算子をダブルクリックあるいは右クリックして、Propertiesをクリックします。

表 5. アドバンスト・トリガー演算子

<table>
<thead>
<tr>
<th>カテゴリー</th>
<th>名前</th>
</tr>
</thead>
<tbody>
<tr>
<td>信号検出</td>
<td>Edge and Level Detector</td>
</tr>
<tr>
<td>入力オブジェクト</td>
<td>Bit</td>
</tr>
<tr>
<td>比較</td>
<td>Less Than</td>
</tr>
<tr>
<td>比較</td>
<td>Less Than or Equal To</td>
</tr>
<tr>
<td>比較</td>
<td>Equality</td>
</tr>
<tr>
<td>比較</td>
<td>Inequality</td>
</tr>
<tr>
<td>比較</td>
<td>Greater Than or Equal To</td>
</tr>
<tr>
<td>比較</td>
<td>Greater Than</td>
</tr>
<tr>
<td>ビット単位</td>
<td>Bitwise Complement</td>
</tr>
<tr>
<td>ビット単位</td>
<td>Bitwise AND</td>
</tr>
<tr>
<td>ビット単位</td>
<td>Bitwise OR</td>
</tr>
<tr>
<td>ビット単位</td>
<td>Bitwise XOR</td>
</tr>
<tr>
<td>論理</td>
<td>Logical NOT</td>
</tr>
<tr>
<td>論理</td>
<td>Logical AND</td>
</tr>
<tr>
<td>論理</td>
<td>Logical OR</td>
</tr>
<tr>
<td>論理</td>
<td>Logical XOR</td>
</tr>
<tr>
<td>リダクション</td>
<td>Reduction AND</td>
</tr>
<tr>
<td>リダクション</td>
<td>Reduction OR</td>
</tr>
<tr>
<td>リダクション</td>
<td>Reduction XOR</td>
</tr>
<tr>
<td>シフト</td>
<td>Left Shift</td>
</tr>
<tr>
<td>シフト</td>
<td>Right Shift</td>
</tr>
<tr>
<td>カスタムトリガー HDL</td>
<td></td>
</tr>
</tbody>
</table>

Advanced Trigger Condition Editorに追加するオブジェクトが多すぎると、作業スペースが乱雑になり、読み出しが困難になる可能性があります。オブジェクトが整理された状態を保つながらアドバンスト・トリガー条件を作成するには、ショートカット・メニューを使用してArrange All Objectsを選択します。もしくは、Zoom-Outを使用して、より多くのオブジェクトがAdvanced Trigger Condition Editorウィンドウに収まるようにします。

2.4.3.1. アドバンスト・トリガー式の例

以下の例で示しているのは、アドバンスト・トリガーの使用方法です。
図-36: バス outa が outb より大きいあるいは等しい場合にトリガーします。

図-37: イネーブル信号に立ち上がりエッジがあるバス outa がバス outb より大きいあるいは等しく、かつイネーブル信号に立ち上がりエッジがある場合にトリガーします。

図-38: ビット単位 AND 演算

2.4.4. カスタムトリガーの HDL オブジェクト

Signal Tap Logic Analyzer では、独自の HDL モジュールを使用してカスタムのトリガー条件を作成することができます。Custom Trigger HDL オブジェクトを使用すると、トリガーロジックをシミュレーションし、ロジック自体に障害がないことを確認することができます。さらに、モジュールのインスタンスのタップが、デザインの階層内の任意の箇所で可能です。このとき、必要なすべての接続を手動配線する必要はありません。

Custom Trigger HDL オブジェクトの表示は、Advanced Trigger エディターの Object Library ペインにされます。
2.4.4.1. Custom Trigger HDL オブジェクトの使用

カスタムのトリガーフローの定義には、次の手順を実行します。

1. 編集するトリガーを選択します。
2. Advanced Trigger タブを開くには、Trigger Conditions ドロップダウンメニューで Advanced を選択します。
3. プロジェクトにトリガーモジュールを含む HDL ソースファイルを追加するには、Project Navigator を使用します。
   一 あるいは、トリガーモジュール用の HDL をプロジェクトに既に含まれているソースファイルに追加します。

図 -40: Project Navigator タブの HDL Trigger

4. Custom Trigger HDL モジュールに必要な入力と出力を実装します。
5. Custom Trigger HDL オブジェクトをドラッグして、オブジェクトのデータ入力バスと結果出力ビットを最終のトリガー結果に接続します。
図-41: Custom Trigger HDL オブジェクト

図-42: オブジェクト・プロパティーのコンフィグレーション

6. Custom Trigger HDL オブジェクトを右クリックし、オブジェクトのプロパティーをコンフィグレーションします。

7. デザインをコンパイルします。

8. Signal Tap でのデータ取得を Custom Trigger HDL オブジェクトを使用して行います。

例-2: Verilog HDL トリガー

次のトリガーでは、コンフィグレーション・ビットストリームを使用します。

```verilog
module test_trigger
  ( input acq_clk, reset, input[3:0] data_in, input[1:0] pattern_in, output reg trigger_out
);
  always @(pattern_in) begin case (pattern_in)     2'b00:     trigger_out = &data_in;     2'b01:     trigger_out = |data_in;     2'b10:     trigger_out = 1'b0;     2'b11:     trigger_out = 1'b1; endcase
endmodule
```

このトリガーには、コンフィグレーション・ビットストリームは含まれません。

```verilog
module test_trigger_no_bs
  ( input acq_clk, reset, input[3:0] data_in, output reg trigger_out
);
  assign trigger_out = &data_in;
endmodule
```
2.4.4.2. Custom Trigger HDL モジュールに必要な入力と出力

表 6. Custom Trigger HDL モジュールに必要な入力と出力

<table>
<thead>
<tr>
<th>名称</th>
<th>詳細</th>
<th>入力/出力</th>
<th>要/任意</th>
</tr>
</thead>
<tbody>
<tr>
<td>acq_clk</td>
<td>Signal Tap で使用する取得クロック</td>
<td>入力</td>
<td>要</td>
</tr>
<tr>
<td>reset</td>
<td>キャプチャー再開時に Signal Tap で使用するリセット</td>
<td>入力</td>
<td>要</td>
</tr>
<tr>
<td>data_in</td>
<td>• Advanced Trigger Editor で接続したデータ入力</td>
<td>入力</td>
<td>要</td>
</tr>
<tr>
<td></td>
<td>• トリガーのためにモジュールで使用するデータ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pattern_in</td>
<td>• コンフィグレーションビットストリーム・プロパティー</td>
<td>入力</td>
<td>任意</td>
</tr>
<tr>
<td></td>
<td>• Signal Tap GUI で設定可能な実行時のコンフィグレーションが可能なプロパティー、トリガーロジックの動作を変更します。</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trigger_out</td>
<td>モジュールの出力信号。トリガー条件が満たされたときにアサートします。</td>
<td>出力</td>
<td>要</td>
</tr>
</tbody>
</table>

2.4.4.3. Custom Trigger HDL モジュールのプロパティー

表 7. Custom Trigger HDL モジュールのプロパティー

<table>
<thead>
<tr>
<th>プロパティー</th>
<th>詳細</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom HDL Module Name</td>
<td>トリガーロジックのモジュール名。</td>
</tr>
<tr>
<td>Configuration Bitstream</td>
<td>• 実行時に設定できるトリガーロジックの作成が可能で、コンフィグレーション・ビットストリームの値に基づいています。</td>
</tr>
<tr>
<td></td>
<td>• Signal Tap ロジック・アナライザでは、コンフィグレーション・ビットストリーム・プロパティーをバイナリとして読み出すため、ビットストリームに含めるのは、1 と 0 の文字のみにしてください。</td>
</tr>
<tr>
<td></td>
<td>• ビット幅（1 と 0）は、pattern_in ビット幅と一致させてください。</td>
</tr>
<tr>
<td></td>
<td>• 空白のコンフィグレーションビットストリームが意味するのか、モジュールにpattern_in 入力がないことです。</td>
</tr>
<tr>
<td>Pipeline</td>
<td>トリガーロジックのパイプライン・ステージ数を指定します。例えば、LA でトリガー入力の受信後に、トリガー出力のアサートのために 3 クロックサイクルが必要な場合、パイプライン値を 3 に指定します。</td>
</tr>
</tbody>
</table>

2.4.5. Trigger Condition Flow の制御

Trigger Condition Flow を使用すると、トリガー条件のセット間の関係を定義できます。Signal Tap Logic Analyzer の Signal Configuration ペインで提供される次の 2 つのフロー制御メカニズムは、選択トリガー条件の整理に使用できます。

- **Sequential Triggering** - デフォルトのトリガーフローです。Sequential Triggering では、トリガーレベルの定義を最大 10 段階で設定できます。このトリガーレベルは、アクションをトリガーに移すためのキャプチャーが終了する前に満たす必要があります。

- **State-Based Triggering** - アクションをトリガーの最大限の制御が可能です。カスタムベースのトリガーでは、トリガー条件を整理して、定義する条件フロー上で State-Based にすることが可能です。

Sequential Triggering と State-Based Triggering は、セグメント化バッファーと非セグメント化バッファーのいずれとでも使用できます。
2.4.5.1. Sequential Triggering

Sequential Triggering を指定すると、Signal Tap Logic Analyzer では各条件を順次評価します。Sequential Triggering フローでは、最大 10 段階のレベルのトリガー条件をカスケードが可能です。

最後のトリガー条件の評価が TRUE になると、Signal Tap Logic Analyzer でのデータ取得が開始します。セグメント化バッファーの場合、2 番目以降の獲得セグメントはすべて、指定された最後の条件で始まります。Simple Sequential Triggering 機能で指定できるのは、基本トリガー、比較トリガー、詳細トリガー、またはこの 3 つの組み合わせです。

図 -43: Sequential Triggering フロー
この図で示すのは、非セグメント化バッファーとセグメント化バッファーに対する単純な Sequential Triggering フローです。

注:
1. アクイジョンバッファーによるキャッチャーの開始は、n 個のトリガーレベルがすべて満たされたときです。ここでは、n ≤ 10 です。

Signal Tap Logic Analyzer では、外部トリガーをレベル 0 と見なし、外部トリガーを他のトリガー条件より前に評価します。

2.4.5.1.1. Sequential Triggering フローのコンフィグレーション

Signal Tap Logic Analyzer を Sequential Triggering 用にコンフィグレーションするには、次を実行します。
1. Trigger Flow Control で Sequential を選択します。
2. Trigger Conditions で、トリガー条件の数をドロップダウンリストから選択します。Node List ペインに表示される数が、トリガー条件カラムと同じになります。
3. 各トリガー条件のコンフィグレーションを Node List ペインで行います。
   任意のトリガー条件のイネーブル/ディスエーブルは、カラムヘッダーから行います。
2.4.5.1.2 条件達成後にクロックサイクルをスキップするトリガー

例-3: 条件1の達成後にサンプルの3クロックサイクルをスキップするトリガーフロー記述

コード:

State 1: ST1
  start_store
  if (condition1)
    begin
      stop_store;
      goto ST2;
    end
  
State 2: ST2
  if (c1 < 3)
    increment c1; //skip three clock cycles; c1 initialized to 0
  else if (c1 == 3)
    begin
      start_store; //start_store necessary to enable writing to finish
                    //acquisition
      trigger;
    end

図で示すのは、連続キャプチャーでのデータトランザクションと、トリガーフロー記述を適用したときのデータキャプチャーです。

図-45: データトランザクションの継続的なキャプチャー

図-46: トリガーフロー記述を適用したデータトランザクションのキャプチャー
2.4.5.1.3. ポストフィル・カウント値が m より小さい Storage Qualification

例-4: 前のシナリオの実データ取得

図-47: ポストフィル・カウント値が m より小さい Storage Qualification (取得が正常に完了)
データ・キャプチャーが正常に終了します。使用するバッファーのサンプル深度は 64, \( m = n = 10 \), post-fill count = 5 です。
図-48: ポストフィル・カウント値がmより大きいStorage Qualification（取得が無期限に停止）
ロジック・アナライザーは、stop_store条件が原因のトリガー条件発生後でも、無期限に一時停止します。このシナリオで使用するサンプル深度は64で、m = n = 10、post-fill count = 15です。

ステータスバーおよび現在の値のフィールドにデータ取得のリアルタイム・ステータスが表示されます。
追加されたフラグによってトリガーフロー記述がトリガーされ、ランタイム中の実行の確定に役立てます。

図-49: 解析の強制停止後の波形
カウンター、ブール演算子および関係演算子の組み合わせを start_store および stop_store コマンドと併用すると、クロックサイクル・レベルの解像度が得られ、それを使用してアクイジション・バッファに書き込まれるサンプルを制御します。

### 2.4.5.2. State-Based Triggering

State-Based Triggering では、状態図に従ってアクイジション・バッファーをトリガーするイベントが構成されます。状態（ステート）では、アクイジション・バッファーで実行するアクションすべてをキャプチャーし、各状態に含まれる条件式で遷移条件が定義されます。

カスタムの State-Based Triggering により、トリガー条件の配置を制御する権限が譲渡されます。Logic Analyzer でキャプチャーする目的是目的のサンプルのみであるため、カスタムの State-Based Triggering では、アクイジション・バッファーで使用可能なスペースのより効率的な使用が可能です。

トリガー条件間の関係を説明するため、State-Based Triggering によってツールチップがフロー GUI 内で提供されます。さらに、Signal Tap Trigger Flow Description Language も使用可能です。この記述言語は条件式に基づいています。

図 -50: **State-Based Triggering フロー**

注：
1. 最大 20 の異なるステートが定義可能です。
2. ロジック・アナライザーで評価する外部トリガー入力の定義は、カスタムの State-Based Triggering フローの他の条件より先に行います。

各ステートでは、一連の条件式の定義ができます。条件式はブール式で、トリガー条件、カウンター、およびステータスフラグの組み合わせに依存します。トリガー条件のコンフィグレーションは Setup タブで行います。Signal Tap Logic Analyzer のカスタムベースのトリガーフローによってカウンターとステータスフラグが提供されます。

各条件式内で一連のアクションを定義します。アクションに含まれるのは、キャプチャー停止のためのアクイジション・バッファーのトリガー、カウンターまたはステータスフラグの変更、または状態遷移です。

トリガーアクションの適用対象は、セグメント化アクイジション・バッファーのうち 1 つのセグメント、または非セグメント化アクイジション・バッファー全体です。各トリガーアクションによって提供されるオプションのカウンターまたはサンプル数が指定されます。このサンプルは、ロジック・アナライザーによる現在のセグメントの取得停止前にバッファでキャプチャーされます。count 引数を使用すると、トリガーイベントの発生前にバッファーでキャプチャーするデータ量の制御ができます。
リソース操作アクションを使用すると、カウンターの増減や、ステータスフラグの設定およびクリアができます。ロジックアナライザーでは、カウンターおよびステータスフラグのリソースを条件式のオプション入力として使用します。カウンターおよびステータスフラグが役立つのは、特定のイベントの発生回数を数えたり、フロー制御をトリガーしたりする場合です。

State-Based Triggerringフローを使用すると、時間の連続性がない可能性がある一連のイベントのキャプチャーができます。例えば、2つのデバイス間の通信トランザクションに一連の肯定応答を含むハンドシェイク・プロトコルが含まれる場合です。

2.4.5.2.1. State-Based Triggering Flow タブ

State-Based Trigger Flow タブは、カスタムのState-based triggeringフローの制御インターフェイスです。

このタブが使用可能ならば、Trigger Flow ControlリストでState-Basedを選択した場合のみです。Trigger Flow ControlをSequentialに指定すると、State-Based Trigger Flow タブは表示されません。

![State-Based Triggering Flow タブ](image)

State-Based Trigger Flow タブには次の3つのペインが含まれています。

**State Diagram ペイン**

State Diagram ペインでは、トリガーフローの概要がグラフィック表示されます。このペインには、使用可能なステート数とそのステートの遷移が表示されます。使用可能なステート数の調整には、概要のグラフィック表示の上にあるメニューを使用します。
State Machine ペイン

**State Machine** ペインのテキスト入力ボックスでは、各ステートに関連付けられたトリガーフローとアクションを定義します。

- トリガーフローの定義には、Signal Tap Trigger Flow Description Language を使用します。これは単純な言語で、「if-else」条件宣言に基づいています。
- マウスをカーソルの上に移動するとツールヒントが表示され、ステートボックスへのコマンド入力が誘導されます。
- GUI では、フロー記述の構文チェックをリアルタイムで提供し、テキストフロー内のエラーをハイライト表示します。

State Machine の説明テキストボックスには、デフォルトで、ステートごとに 1 つのテキストボックスが表示されます。また、全フローの説明表示を 1 つのテキストフィールドですることもできます。このオプションが役立つのは、フロー記述をテンプレートまたは外部テキストエディターからコピーして貼り付ける場合です。ステートごとのウィンドウ間、あるいは 1 つのウィンドウ内のすべてのステートをトグルするには、適切なオプションをState Display mode で選択します。

関連情報

Signal Tap Trigger Flow Description Language (61 ページ)

Resources ペイン

**Resources** ペインでは、ステータスフラグとカウンターの宣言を Custom Triggering Flow の条件式に対して行うことができます。

- カウンターの増分/減分やステータスフラグの設定/クリアは、トリガーフロー内で実行できます。
- 指定できるのは、最大 20 個のカウンターと 20 個のステータスフラグです。
- カウンターフラグおよびステータスフラグの初期化には、表中の行を右クリックし、Set Initial Value を選択します。
- カウンター幅の指定には、表中のカウンターを右クリックし、Set Width を選択します。
- トリガーフロー仕様のデバッグの支援のため、ロジック・アナライザーでは、カウンターおよびフラグ値の動的更新を、取得開始後に行います。

Configurable at runtime 設定では、ランタイムに変更するオプションの制御ができます。このとき、再コンパイルは不要です。

<table>
<thead>
<tr>
<th>設定</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>goto アクションのデスティネーション</td>
<td>状態遷移のデスティネーションをランタイムに変更できます。</td>
</tr>
<tr>
<td>比較値</td>
<td>ブール式の比較値をランタイムに変更できます。さらに、segment_trigger およびトリガーアクション・ポストフィール count 引数もランタイムに変更できます。</td>
</tr>
<tr>
<td>比較演算子</td>
<td>ブール式の演算子をランタイムに変更できます。</td>
</tr>
<tr>
<td>論理演算子</td>
<td>ブール式の論理演算子をランタイムに変更できます。</td>
</tr>
</tbody>
</table>

関連情報

- パフォーマンスおよびリソースに関する考慮事項 (71 ページ)
- Runtime Reconfigurable オプション (74 ページ)
2.4.5.2.2. Trigger Lock モード

Trigger Lock モードによる制限のため、変更できるのはコンフィグレーション設定のうち、ユーザーが Configurable at runtime と指定した場合のみです。Custom Trigger Flow タブのランタイムにコンフィグレーションが可能な設定は、デフォルトではオフとなっています。

注意: パフォーマンスの向上は、ランタイムにコンフィグレーションが可能なオプションの一部をディスアーブルすることで得られることがあります。

Signal Tap コンフィグレーションへの変更に制限を設けて、再コンパイルが不要なオプションだけを含めることができます。Trigger Lock モードでは、デバイス内ですぐに反映される変更を加えることができます。

1. Setup タブで Lock Mode を指定し、Allow trigger condition changes only を選択します。

2. トリガーフロー条件を変更します。

2.4.5.3. Signal Tap Trigger Flow Description Language

Trigger Flow Description Language は、ステートごとの条件式のリストに基づいており、一連のアクションを定義します。

Logic Analyzer で評価するアクションの記述を、1 つのステートに達した際に行うには、次の構文に従います。

Trigger Flow Description Language の構文

```
state <state_label>:
  <action_list>
  if (<boolean_expression>)
    <action_list>
  [else if (<boolean_expression>)
    <action_list>
  ]
  [else
    <action_list>
  ]
```

- 非端子は「<>」で区切られます。
- オプションの引数は「[]」で区切られます。
- 条件宣言の評価への優先順位は上から下です。
- Trigger Flow Description Language では、複数の else if 条件を許可します。

<state_label> (62 ページ)
<boolean_expression> (62 ページ)
<action_list> (63 ページ)

関連情報
カスタム・トリガー・フローのアプリケーション例 (98 ページ)
2.4.5.3.1. <state_label>

所定のステートを識別します。ステートラベルを使用して Logic Analyzer で評価するアクションの記述を開始するのは、前述のステートに達した後です。ステートラベルは、 goto コマンドと併用することもできます。

ステートを記述するヘッダーの構文は次のとおりです。

```
state <state_label>
```

ステートの記述が終了するのは、別のステートの開始時、またはすべてのトリガーフローの記述の完了時です。

2.4.5.3.2. <boolean_expression>

ブール結果に評価される演算子とオペランドの集合体です。演算子は論理演算子と関係演算子のどちらでも可能です。演算子に応じて、オペランドでトリガー条件、カウンターおよびレジスター、または数値を参照します。オペランドのセットを式の中でグループ化するには、括弧を使用します。

表 9. 論理演算子

論理演算子は、ブール式をオペランドとして受け入れます。

<table>
<thead>
<tr>
<th>演算子</th>
<th>説明</th>
<th>構文</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>NOT 演算子</td>
<td>! expr1</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>AND 演算子</td>
<td>expr1 &amp;&amp; expr2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

表 10. 関係演算子

関係演算子は、カウンターまたはステータスフラグに使用します。

<table>
<thead>
<tr>
<th>演算子</th>
<th>説明</th>
<th>構文</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>より大きい</td>
<td>&lt;identifier&gt; &gt; &lt;numerical_value&gt;</td>
</tr>
<tr>
<td>&gt;=</td>
<td>より大きいまたは等しい</td>
<td>&lt;identifier&gt; &gt;= &lt;numerical_value&gt;</td>
</tr>
<tr>
<td>==</td>
<td>等しい</td>
<td>&lt;identifier&gt; == &lt;numerical_value&gt;</td>
</tr>
<tr>
<td>!=</td>
<td>等しくない</td>
<td>&lt;identifier&gt; != &lt;numerical_value&gt;</td>
</tr>
<tr>
<td>&lt;=</td>
<td>より小さいまたは等しい</td>
<td>&lt;identifier&gt; &lt;= &lt;numerical_value&gt;</td>
</tr>
<tr>
<td>&lt;</td>
<td>より小さい</td>
<td>&lt;identifier&gt; &lt; &lt;numerical_value&gt;</td>
</tr>
</tbody>
</table>

注:
1. <identifier> は、カウンターあるいはステータスフラグを意味します。
2. <numerical_value> は整数を意味します。

注意:
- if 宣言の <boolean_expression> には、単一あるいは複数のイベント条件を含めることができます。
- ブール式によって TRUE と評価されると、ロジック・アナライザーでは <action_list> 内のすべてのコマンドを同時に評価します。
2.4.5.3.3. <action_list>

条件が満たされたときに、ロジック・アナライザーによって1つのステート内で実行されるアクションのリストです。

- 各アクションの最後はセミコロン（;）にしてください。
- 複数のアクションを1つのifあるいはelse if句内で指定する場合、action_listをbeginトークンとendトークンで区切ってください。

可能なアクションは次のとおりです。

リソース操作アクション

トリガーフロー記述で使用するリソースは、カウンターまたはステータスフラグです。

表11. リソース操作アクション

<table>
<thead>
<tr>
<th>アクション</th>
<th>説明</th>
<th>構文</th>
</tr>
</thead>
<tbody>
<tr>
<td>increment</td>
<td>カウンタリソースを1増分します。</td>
<td>increment &lt;counter_identifier&gt;;</td>
</tr>
<tr>
<td>decrement</td>
<td>カウンタリソースを1減分します。</td>
<td>decrement &lt;counter_identifier&gt;;</td>
</tr>
<tr>
<td>reset</td>
<td>カウンタリソースをリセットして初期値にします。</td>
<td>reset &lt;counter_identifier&gt;;</td>
</tr>
<tr>
<td>set</td>
<td>ステータスフラグを1に設定します。</td>
<td>set &lt;register_flag_identifier&gt;;</td>
</tr>
<tr>
<td>clear</td>
<td>ステータスフラグを0に設定します。</td>
<td>clear &lt;register_flag_identifier&gt;;</td>
</tr>
</tbody>
</table>

バッファー制御アクション

アクイジョン・バッファーを制御するアクションです。

表12. バッファー制御アクション

<table>
<thead>
<tr>
<th>アクション</th>
<th>説明</th>
<th>構文</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger</td>
<td>現在のバッファーの取得を停止し、解析を終了します。このコマンドはすべてのフロー定義で必要です。</td>
<td>trigger &lt;post-fill_count&gt;;</td>
</tr>
<tr>
<td>segment_trigger</td>
<td>Segmented acquisitionモードでのみ使用可能です。現在のセグメントの取得を停止します。Signal Tap Logic Analyzerでは、このコマンドの評価後、次のセグメントから取得を開始します。すべてのセグメントが書き込まれている場合、Logic Analyzerでは、最も古いセグメントを最新のサンプルで上書きします。トリガーアクションが評価されると取得は停止します。</td>
<td>segment_trigger &lt;post-fill_count&gt;;</td>
</tr>
<tr>
<td>start_store</td>
<td>State-based storage qualifierモードでのみアクティブになります。write_enableをSignal Tapアクイジョン・バッファーにアサートします。</td>
<td>start_store</td>
</tr>
<tr>
<td>stop_store</td>
<td>State-based storage qualifierモードでのみアクティブになります。write_enable信号をSignal Tapアクイジョン・バッファーにディアサートします。</td>
<td>stop_store</td>
</tr>
</tbody>
</table>

triggerとsegment_triggerの両方のアクションで、オプションのpost-fill_count引数を受け入れます。
関連情報
ポストフィル・カウンター (66 ページ)

ステート遷移アクション
カスタムの State control フローで次のステートを指定します。構文は次のとおりです。
\[
goto <state_label>;
\]

2.4.5.4. State-Based Storage Qualifier 機能
State-Based Storage Qualifier タイプを選択すると、start_store および stop_store アクションが有効になります。これらのアクションを State-based trigger フローの式と併用すると、アクイジョン・バッファーに書き込まれるデータの制御に最大限の柔軟性が得られます。

注意: start_store および stop_store コマンドが適用できるのは、非セグメント化バッファーに対してのみです。

start_store および stop_store コマンドは、start / stop Storage Qualifier モードの開始および停止条件に似ています。Storage Qualification がイネーブルされると、Signal Tap Logic Analyzer では、データのアクイジョン・バッファーへの書き込みを start_store コマンドの発生まで行いません。ただし、State-Based Storage Qualification タイプでは、trigger コマンドをトリガーフローの記述の一部として含める必要があります。この trigger コマンドは、取得を完了し、結果を波形ディスプレイに表示するために必要です。

2.4.5.4.1. State-Based Trigger フローのための Storage Qualification 機能
このトリガーフローの記述に含まれている 3 つのトリガー条件は、Start Analysis をクリックした後のさまざまなタイミングで発生します。

State 1: ST1:
\[
\begin{align*}
&\text{if ( condition1 ) } \\
&\quad \text{start_store;} \\
&\text{else if ( condition2 ) } \\
&\quad \text{trigger value;} \\
&\text{else if ( condition3 ) } \\
&\quad \text{stop_store;}
\end{align*}
\]

図 -53: State-Based Trigger フローを使用した Storage Qualification のためのキャプチャー・シナリオ

トリガーフローを下図のシナリオに適用すると、次のようにになります。
1. Signal Tap Logic Analyzer によるアクイジョン・バッファーへの書き込みは、Condition 1 が発生するまで行われません（サンプル a）。
2. Condition 2 が発生すると（サンプル b）、ロジック・アナライザーでは trigger value コマンドを評価し、バッファーへの書き込みを続けて、取得を終了します。
3. トリガーフローでは、stop_store コマンドをサンプル c で指定します。この発生は、トリガーポイントから m サンプル後です。
4. データ取得によるポストフィル取扱取サンプルの終了が Condition 3 の発生前の場合、ロジック・アナライザーでは取得を終了し、波形の内容を表示します。この場合、キャプチャーが終了するのは、ポストフィル後のカウント値が <m の場合です。
5. Trigger Flow 記述 1 のポストフィル・カウント値が >m サンプルの場合、バッファーでの取扱が無期限に一時停止します。ただし、その条件は、ロジック・アナライザーをトリガーする Condition 1 の繰り返しによってデータのキャプチャーが再開されないことです。

Signal Tap Logic Analyzer では、stop_store および start_store コマンドの評価をトリガーの評価後も続けます。取得が一時停止した場合は、Stop Analysis をクリックして手動で取得をトリガーします。カウンター値、フラグ、および State 図を使用してトリガーフローを実行します。カウンター値、フラグ、および現在の状態の更新は、データ取得中にリアルタイムで行われます。

2.4.6. トリガー位置の指定

Logic Analyzer の取得データ量の指定は、トリガーイベントの前後で行うことができます。Runtime と Power-Up トリガーの位置は別々です。

Signal Tap Logic Analyzer で提供している 3 つの定義済み比率は、次のとおり、トリガー前からトリガー後までのデータに対するものです。
- **Pre** - トリガー後に発生した信号アクティビティーを保存します。（トリガー前 12%, トリガー後 88%）
- **Center** - トリガー前のデータ 50%およびトリガー後のデータ 50%を保存します。
- **Pre** - トリガー後に発生した信号アクティビティーを保存します。（トリガー後 88%, トリガー前 12%）

この定義済み比率の適用対象は、非セグメント化バッファーとセグメント化バッファーの両方です。

関連情報
State-Based Triggering (58 ページ)
2.4.6.1. ポストフィル・カウンター

カスタムの State-based triggering フローでは、segment_trigger および trigger パッファー制御アクションを使用する場合、post-fill_count 引数を使用してカスタムトリガー位置を指定します。

- post-fill_count 引数を使用しない場合、影響されるバッファーのデフォルトのトリガー位置は、Setup タブで指定したトリガー位置です。
- trigger バッファー制御アクション (非セグメント化バッファー用) では、post-fill_count によって、データ取得停止前にキャプチャーするサンプル数を指定されます。
- segment_trigger バッファー制御アクション (セグメント化バッファー用) では、post-fill_count によってデータセグメントが指定されます。

注: segment_trigger で現在のバッファーの取得が即時停止するのは、後続のトリガー動作が次のステートで発行された場合であり、現在のバッファーのポストフィル・カウントは関係ありません。Logic Analyzer によって、現在のバッファーに残っている未充填ポストカウント保持は破棄され、サンプルとしてデータウィンドウでグレー表示されます。

Signal Tap データウィンドウのキャプチャー・データの表示でのトリガー位置は、最後の取得セグメントまたはバッファーからのポストカウントのサンプル数になります。

トリガー位置のサンプル数 = \( N - \) ポストフィル・カウント

この場合の \( N \) は、取得セグメントまたは非セグメント化バッファーのいずれかのサンプル深度です。

関連情報
バッファー制御アクション (63 ページ)

2.4.7. パワーアップ・トリガー

パワーアップ・トリガーでは、デバイスの初期化中に発生するイベントを FPGA の電源投入またはリセットの直後にキャプチャーします。

Signal Tap Logic Analyzer の一般的な使用方法は、通常のデバイス動作中に発生するイベントのトリガーです。手動解析の開始は、ターゲットデバイスの電源が完全に入り、デバイスへの JTAG 接続が使用可能になった時点で行います。Signal Tap Power-Up Trigger 機能を使用すると、Signal Tap Logic Analyzer によるデータのキャプチャーがデバイス初期化の直後に可能です。

別の Power-Up Trigger のロジック・アナライザーの各インスタンスへの追加は、Signal Tap Instance Manager ペインで行います。

2.4.7.1. Power-Up Trigger のイネーブル

Power-Up Trigger をロジック・アナライザーのインスタンスに対してイネーブルするには、次を実行します。
・ インスタンスを右クリックして、Enable Power-Up Trigger をクリックします。

図 -54: Signal Tap Logic Analyzer Editor での Power-Up Trigger のイネーブル

Power-Up Trigger は、選択したインスタンスの名前の下に子インスタンスとして表示されます。ノードリストには、デフォルトのトリガー条件が表示されます。

Power-Up Trigger をディスエーブルするには、インスタンスを右クリックして Disable Power-Up Trigger をクリックします。

2.4.7.2. Power-Up Trigger 条件のコンフィグレーション

・ Power-Up Trigger 条件に変更を加えた場合は、Signal Tap Logic Analyzer の再コンパイルが必要で、類似の変更を Run-Time Trigger 条件へ加えた場合に再コンパイルが不要なのは異なります。

・ トリガー条件の強制は、Signal Tap Logic Analyzer と In-System Sources and Probes の併用でも可能です。In-System Sources and Probes 機能を使用すると、JTAG チェーン上で選択したネットに対して値を駆動してサンプリングすることができます。

関連情報
In-System Sources and Probes を使用したデザインのデバッグ (127 ページ)

2.4.7.3. Run-Time および Power-Up Trigger 条件を使用した Signal Tap インスタンスの管理

2 つのタイプのトリガー条件を持つインスタンスでは、Power-Up Trigger 条件は明るい青色で色分けされていますが、Run-Time Trigger 条件は白色のままです。

・ Power-Up Trigger と Run-Time Trigger のトリガー条件を切り替えるには、Instance Manager でインスタンス名または Power-Up Trigger 名をダブルクリックします。

・ Run-Time Trigger から Power-Up Trigger に、またはその逆にトリガー条件をコピーするには、Instance Manager でトリガー名を右クリックして、Duplicate Trigger をクリックします。あるいは、トリガー名を選択して Edit > Duplicate Trigger をクリックします。

注意: ランタイムトリガー条件で調整可能な項目は、パワーアップ・トリガー条件よりも少数です。

2.4.8. 外部トリガー

外部トリガー入力を使用すると、Signal Tap Logic Analyzer のトリガーを外部ソースから行うことができます。

外部トリガー入力の動作は、トリガー条件 0 に類似しています。この場合、条件の評価結果は、ロジック・アナライザーによる他のトリガー条件の評価前に TRUE である必要があります。
Signal Tap Logic Analyzer で供給する信号では、外部デバイスまたは他のロジック・アナライザー・インスタンスをトリガーします。この機能により、外部ロジック解析機器と内部ロジック・アナライザーが同期できます。Power-Up Trigger では、外部トリガー機能を使用できますが、関連する Run-Time Trigger と同一ソースまたはターゲット信号を使用する必要があります。

外部トリガーを使用して、ハード・プロセッサー・システム (HPS) でのクロストリガーを次のように実行できます。

- プロセッサー・デバッガーを使用すると、HPS のコンフィグレーションで、FPGA からのクロストリガー要求に従うか無視するか、またはクロストリガー要求の発行を FPGA に対してするかを選択できます。
- プロセッサー・デバッガーと Signal Tap 外部トリガー機能の組み合わせによって、クロストリガー動作の動的組み合わせが開発できます。
- システムレベルのデバッグ・ソリューションを FPGA SoC へ実装するのに、Intel ARM Development Studio 5 (DS-5) ソフトウェアのクロストリガー機能が使用できます。

関連情報
- FPGA-Adaptive Software Debug and Performance Analysis white paper
- Signal Configuration Pane
  Intel Quartus Prime Help 内

2.5. デザインのコンパイル

Signal Tap ロジックをデザインに統合し、JTAG 接続をイネーブルするには、プロジェクトをコンパイルする必要があります。.stp ファイルをプロジェクトに追加すると、Signal Tap Logic Analyzer はデザインの一部になります。デザインのデバッグに従来の外部ロジック・アナライザーを使用する場合は、モニターする信号とトリガー条件を頻繁に変更することが必要です。

2.5.1. 再コンパイルを必要とする変更の防止

.stp をコンフィグレーションして、通常は再コンパイルが必要になる変更を防止します。これを実行するには、Setup タブのノードリストの上部から Lock mode を選択します。コンフィグレーションをロックするには、Allow trigger condition changes only を選択します。

図 -55: Allow Trigger Conditions Change Only (トリガー条件のみの変更の許可)

関連情報
プロジェクトを再コンパイルする必要性の確認 (68 ページ)

2.5.2. プロジェクトを再コンパイルする必要性の確認

デバッグセッションの開始前には、プロジェクトの再コンパイルを必要とする変更を .stp 設定に対しても加えないでください。
変更内容にプロジェクトの再コンパイルが必要かどうかを検証するには、Signal Tap ステータス表示を Instance Manager ペインの最上部で確認してください。この機能を使用すると、変更を元に戻すことができるため、プロジェクトの再コンパイルが不要になります。

関連情報
再コンパイルを必要とする変更の防止 (68 ページ)

2.5.3. ラピッド・リコンパイルを使用したインクリメンタル配線

インクリメンタル配線とラピッド・リコンパイルを併用すると、コンパイル時間が短縮できます。デザインのフルコンパイルの実行後、インクリメンタル配線フローを使用して、フラットコンパイルに対して2倍から4倍の高速化が実現できます。インクリメンタル配線フローには、パーシャル・リコンフィギュレーションとの互換性はありません。

インテル Quartus Prime プロ・エディション開発ソフトウェアでサポートしているラピッドコンパイルを使用したインクリメンタル配線の対象は、インテル Arria® 10、インテル Cyclone® 10 GX、およびインテル Stratix 10 デバイスファミリーです。

関連情報
Running Rapid Recompile

2.5.3.1. インクリメンタル配線フローの使用

インクリメンタル配線フローを使用するには、次の手順を実行します。

1. デザインを開き、Analysis & Elaboration（またはフルコンパイル）を実行して、Signal Tap でノードを可視性します。
2. Signal Tap をデザインに追加します。
3. Signal Tap Signal Configuration ペインで、Nodes Allocated フィールドの Manual を Trigger and Data ノード（使用していれば Storage Qualifier）に対して指定します。
図 -56: ノードの手動割り当て

ノードの手動割り当てによって、デザインでコンパイルするノード数の制御ができます。これは、インクリメンタル配線フローには重要です。

Auto 割り当てを選択すると、デザインでコンパイルされるノード数が Setup タブのノード数と一致します。ノードを後で追加すると、デバイスに必要なノード数とコンパイルされるノード数にミスマッチが生じ、フルコンパイルの実行が必要になります。

4. デバッグプロセスに必要であると推定されるノード数を指定します。ノード数は後で増やすことができますが、コンパイル時間が長くなります。

5. タップするノードを追加します。

6. プロジェクトのコンパイルが完了していない場合は、フルコンパイルを実行してください。それ以外の場合は、ラピッド・リコンパイルを使用してインクリメンタル・コンパイルを開始します。

7. 対象の追加信号をデバッグして確定します。

8. Allow incremental route changes only lock-mode を選択します（オプション）。

図 -57: Incremental Route Lock-Mode

9. ノードの追加を Signal Tap Setup タブで行います。

- 指定した手動割り当てノード数を超えないようにしてください。
- ランタイム以外のコンフィグレーション可能な設定は変更しないでください。

10. タスルバーのラピッド・リコンパイル・アイコン をクリックします。もしくは、Processing > Start Rapid Recompile の順でクリックします。

注意: ここまでの手順でインクリメンタル配線の設定はできますが、実際のインクリメンタル配線プロセスはラピッド・リコンパイルの実行時に開始します。
2.5.3.2. 最大限の高速化を達成するためのヒント

- Basic AND (適用対象は Storage Qualifier のほかにトリガー入力) がインクリメンタル配線フローに対して最速です。
- Basic OR はインクリメンタル配線フローに対しては最速ではありませんが、グループ内ノードの親子関係の変更を避ける場合は、コンパイルタイムへの影響を最小限に抑えることができます。ノードの兄弟関係は変更できます。
  - Basic OR およびアドバンスト・トリガーで再合成が必要なのは、タップしたノードの数/名前を変更する場合です。
- Incremental Route lock-mode を使用して、フルコンパイルが必要となる意図せぬ変更を回避してください。

2.5.4. Signal Tap Logic Analyzer を使用したタイミングの保持

機能検証に加えて、タイミング・クロージャーはデザインの正常な動作において最も重要なプロセスの 1つです。

インテル Quartus Prime プロ・エディション開発ソフトウェアによるインテル Arria 10 デザイン内のポストフィット・タップのタイミング保存のサポートでは、ラピッド・リコンパイル機能を使用します。ラピッド・リコンパイルは、再コンパイル時にデザインの検証済み部分を再処理するのではなく、自動的に再利用します。

注意: Signal Tap インテル FPGA IP の最適化は、インテル Stratix 10 アーキテクチャーに対しては行われません。

次の方法は、タイミングの維持に役立ちます。
- クリティカル・パス信号は .stp ファイルに追加しない。
- .stp ファイルに追加する組み合わせ信号数は最小にとどめ、可能な場合はレジスターを追加する。
- fMAX 制約をデザインの各クロックに対して指定する。

関連情報
Timing Closure and Optimization

2.5.5. パフォーマンスおよびリソースに関する考慮事項

デザインのロジック解析を実行すると、ランタイムの柔軟性、タイミング・パフォーマンス、リソース使用率の間で必然的なトレードオフが存在することが分かります。

Signal Tap Logic Analyzer を使用すると、ランタイムのコンフィグレーションが可能なパラメーターを選択して、ランタイムの柔軟性、スピード、およびエリアに必要な均衡を保つことができます。

ランタイムのコンフィグレーションが可能なパラメーターのデフォルト値では、最大限の柔軟性が提供されており、デバッグをできるだけ迅速に完了することができます。ただし、これらの設定を調整して、デザインにより適したコンフィグレーションがあるかどうかを判断できます。パフォーマンス結果はデザインに依存するので、これらのオプションをさまざまな組み合わせて試してみて、機能、パフォーマンス、および使用率の希望のバランスを実現してください。
2.5.5.1. クリティカル・パス内の Signal Tap ロジック

Signal Tap ロジックがクリティカル・パスの一部になっている場合、次の手順を実行して、Signal Tap Logic Analyzer のパフォーマンスを向上させます。

- **Disable runtime configurable options** - 部分のリソースの割り当ては、ランタイムの柔軟性に対応するように行われます。アドバンスト・トリガーあるいは State-based フローのいずれかを使用する場合、runtime configurable parameters をディスアブルして、Signal Tap ロジックの fMAX をブーストしてください。
  - State-based triggering フローを使用している場合は、Goto state destination オプションのディスアブルして、もう一つのランタイムのコンフィグレーション可能なオプションをディスアブルしてください。Goto state destination オプションでは、他のランタイムのコンフィグレーション可能なオプションと比較して、fMAX に最大限の影響を与えます。

- **Minimize the number of signals that have Trigger Enable selected** - デフォルトで Signal Tap Logic Analyzer では、Trigger Enable オプションを .stp ファイルに追加したすべての信号に対してイネーブルします。トリガーとして使用する予定のない信号に対しては、このオプションをオフにしてください。

- **Turn on Physical Synthesis for register retiming** - イネーブルされた多数の（LAB に収まる入力数より多い）トリガー信号が、Gate-based triggering 条件 (advanced trigger タブの basic trigger 条件あるいは logical reduction operator) のロジックにファンインする場合、Perform register retiming をオンにしてください。これは、LAB 間の組み合わせロジックの均衡を保つ一助となります。

2.5.5.2. クリティカル・リソースを使用した Signal Tap ロジック

デザインのリソースに制限がある場合、次のヒントに従って Signal Tap ロジック・アナライザーで使用するロジックあるいはメモリーの低減を行ってください。

- **Disable runtime configurable options** - アドバンスト・トリガー条件または State-based triggering フローのランタイムのコンフィグレーション可能なオプションをディスアブルすることで、LE の低減ができます。

- **Minimize the number of segments in the acquisition buffer** - Signal Tap Logic Analyzer で使用するロジック・リソースの低減は、サンプリングバッファー内のセグメントの制限によって可能です。

- **Disable the Data Enable for signals that you use only for triggering** - デフォルトで Signal Tap Logic Analyzer では、すべての信号に対し data enable オプションをイネーブルします。data enable オプションをトリガー入力としてのみ使用する信号に対してのみオフにすることで、メモリー・リソースが節約できます。

2.6. ターゲットデバイスのプログラム

Signal Tap Logic Analyzer をプロジェクトに追加し、再コンパイラした後、FPGA ターゲットデバイスのコンフィグレーションができます。

複数のデザインを同時にデバッグする場合、デザインのコンフィグレーションは、インテル Quartus Prime Programmer ではなく .stp から行います。これにより、複数の .stp ファイルを開き、複数のデザインをプログラミングすることが可能となります。
2.6.1. .stp ファイルと .sof ファイル間の互換性の設定確認

stp ファイルが .sof ファイルと互換性を持つのは、キャプチャー・バッファーのサイズやモニタリング及びトリガー信号などのロジック・アナライザーの設定が、ターゲットデバイスのプログラミング設定に一致する場合です。ファイルに互換性がなくてもデバイスのプログラミングは可能です。ロジック・アナライザーの実行や制御を Signal Tap Logic Analyzer Editorから行うことはできません。

- プログラミングの互換性を確実にするには、デバイスのプログラミングには、直近のコンパイルで生成された .sof ファイルを使用します。
- 特定の .sof が現在の Signal Tap コンフィグレーションと互換性があるかどうかを確認するには、.sof を SOF マネージャーに添付します。

注意:
SignalTap Logic Analyzer による非互換性の検出が解析開始後にされた場合、インテル Quartus Prime 開発ソフトウェアではシステム・エラー・メッセージが生成されます。このメッセージには 2 つの CRC 値（期待値とデバイス上の .stp インスタンスからの取得値）が含まれています。CRC 値の取得は、コンパイルに影響を与える Signal Tap 設定のすべてから行われます。

ベスト・プラクティスとして、.stp ファイルをインテル Quartus Prime プロジェクトで使用します。プロジェクト・データベースには、現在の Signal Tap Logic Analyzer セッションの整合性に関する情報が含まれています。プロジェクト・データベースがなければ、現在の .stp ファイルとデバイス内の .sof ファイルとの一致を確認する方法はありません。.stp ファイルが .sof ファイルと一致しない場合、Signal Tap Logic Analyzer で誤ったデータをキャプチャする可能性があります。

関連情報
複数の Signal Tap ファイルとコンフィグレーションの管理 (42 ページ)

2.7. Signal Tap Logic Analyzer の実行

Signal Tap Logic Analyzer のデバッグは、外部ロジック・アナライザーを使用する場合と似ています。ロジック・アナライザーを初期化するには、解析を開始します。トリガーイベントが発生すると、ロジック・アナライザーではデバイスのメモリーバッファーにキャプチャーされたデータを格納した上で、そのデータの .stp ファイルへの転送を JTAG 接続を介して行います。

また、強制トリガーの同等機能の実行も可能です。この機能では、現在バッファーにあるキャプチャー・データの表示が可能で、トリガーイベントは発生しません。

次のフローチャートで示すのは、Signal Tap Logic Analyzer の実行方法です。Power-Up トリガーおよび Runtime トリガーイベントが発生する場合、およびこれらのイベントからキャプチャーされたデータが解析用に使用可能になる時期を示しています。
図 -58: Power-Up および Runtime トリガーイベントのフローチャート

In-System Sources and Probes と Signal Tap Logic Analyzer を併用して、トリガー条件を強制することも可能です。In-System Sources and Probes 機能を使用すると、選択した信号に対する値の駆動およびサンプリングを JTAG チェーンを介して行うことができます。

関連情報
In-System Sources and Probes を使用したデザインのデバッグ (127 ページ)

2.7.1. Runtime Reconfigurable オプション

Runtime Trigger モードを使用する場合、.stp 内のいくつかの設定の変更は、デザインをコンパイルしなくても可能です。

表 13. Runtime Reconfigurable の機能

<table>
<thead>
<tr>
<th>Runtime Reconfigurable の設定</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic トリガー条件および Basic Storage Qualification 条件</td>
<td>Basic トリガー条件値の変更には、トリガー条件がオンになっている信号すべての再コンパイルは必要ありません。</td>
</tr>
<tr>
<td>Comparison トリガー条件および Comparison Storage Qualifier 条件</td>
<td>すべての比較オペランド、比較数値、インターバル境界値は、ランタイム・コンフィグレーション可能です。Comparison トリガーから Basic OR トリガーへのランタイムの切り替えには、再コンパイルは不要です。</td>
</tr>
</tbody>
</table>

continued...
Runtime Reconfigurable の設定 | 説明
---|---
アドバンスド・トリガー条件とアドバンスド Storage Qualifier 条件 | 演算子の多くには、ランタイム・コンフィグレーション可能な設定が含まれています。例えば、すべての比較演算子はランタイム・コンフィグレーション可能です。コンフィグレーション可能な設定の表示は、ブロック図内で白い背景になります。このコンフィグレーション可能なオプションを、Object Properties ダイアログボックスでオンにします。
Storage-qualified と Continuous acquisition の切り替え | Storage-qualified モードでの Continuous capture モードへの切り替えには、デザインの再コンパイルは必要ありません。この機能をインペールするには、disable storage qualifier をオンにします。
State-Based Trigger フローのパラメーター | State-Based Triggering フローでランタイム・リコンフィグレーション機能をオフにして、パフォーマンスの向上とエリア使用量を低減します。

Runtime Reconfigurable オプションを使用すると、デバッグサイクル時間の節約が、より広範のイベントのシナリオを網羅することで可能になります。このとき、デザインの再コンパイルは不要です。パフォーマンスおよびロジック使用量に多少の影響が生じることがあります。アドバンスド・トリガー条件および State-Based Trigger フロー・パラメーターのランタイム・リコンフィグレーション機能をオフにして、パフォーマンスの向上とエリア使用量を低減します。

.state ファイルのコンフィグレーションを行って、通常は再コンパイルが必要となる変更を防止するには、Setup タブでノードリストの上の Allow Trigger Condition changes only を選択します。

Incremental Route lock モードでは、Allow incremental route changes only によって制限されるので、変更はフルコンパイルではなく Incremental Route のみを必要とする場合に可能です。

次の例で示すのは、Runtime Reconfigurable 機能に使用可能なユースケースです。Storage qualified がインペールされた State-Based Trigger フローの記述および、キャプチャー・ウィンドウのサイズ変更を再コンパイルなしで行う方法を示しています。この例の同等機能は、単一のトリガー条件を持つセグメント化バッファーに対するもので、そのセグメントサイズはランタイム・リコンフィグレーション可能です。

```plaintext
state ST1:
if ( condition1 && (c1 <= m) )// each "segment" triggers on condition
begin                           // m = number of total "segments"
  start_store;
  increment c1;
  goto ST2;
End
else (c1 > m )                // This else condition handles the last
begin
  start_store
  Trigger (n-1)
end
state ST2:
if ( c2 >= n)                // n = number of samples to capture in each
begin
  reset c2;
  stop_store;
  goto ST1;
end
else (c2 < n)
begin
  increment c2;
  goto ST2;
end
```

2. Signal Tap Logic Analyzer を使用したデザインのデバッグ

UG-20139 | 2018.09.24
注意: \( m \times n \) は、サンプル深度と等しいものにして、サンプルバッファーのスペースを効率的に使用できるようにしてください。

次の図で表すセグメント化バッファーは、トリガーフロー例によって記述されます。

図-59: **Storage Qualifier と State-Based トリガーを使用して作成したセグメント化バッファー**

サンプル深度の合計は固定されています。\( m \times n \) はサンプル深度と等しくなければいけません。

ランタイム、\( m \) と \( n \) の値は変更可能です。トリガーフロー記述で \( m \) と \( n \) の値を変更すると、セグメント境界の調整が再コンパイルなしでできます。

次の例は、前の例に追加のステートが挿入されたものです。この追加ステートを使用して、Storage qualifier 機能を使用しない別のトリガー条件を指定します。ステータスフラグを条件文に挿入するために、トリガーフローの実行を制御します。

```plaintext
state ST1:
  if (condition2 && f1) // additional state added for a non-segmented
    begin
    start_store;      trigger end
  else if (! f1) goto ST2;
state ST2:
  if (condition1 && (c1 <= m) && f2) // f2 status flag used to mask state.
    Set f2
    // to enable begin
    start_store;    increment c1;    goto ST3: end
  else (c1 > m )
    begin
    Trigger (n-1)
    end
state ST3:
  if ( c2 >= n) begin
    reset c2;     stop_store;     goto ST1;
  end
  else (c2 < n)
    begin
    increment c2; goto ST2;
end
```

### 2.7.2. Signal Tap のステータスメッセージ

次の表のテキストメッセージは、データ取得前、取得中、または取得後に Instance Manager ペインの Signal Tap Status Indicator に表示されることがあります。これらのメッセージによって、ロジック・アナライザーの状態をモニタリングし、Logic Analyzer で実行している動作が識別できます。

<table>
<thead>
<tr>
<th>メッセージ</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Not running</strong></td>
<td>Signal Tap Logic Analyzer は動作していません。このメッセージが表示されるのは、デバイスに接続していないか、デバイスがコンフィグレーションされていない場合です。</td>
</tr>
<tr>
<td><em>(Power-Up Trigger)</em> Waiting for clock (1)</td>
<td>Signal Tap Logic Analyzer で Runtime または Power-Up トリガー取得を実行中であるが、クロック信号の遷移が待機中のためです。</td>
</tr>
</tbody>
</table>

---

インテル Quartus Prime プロ・エディション ユーザーガイド: デバッグツール

76
メッセージ | 説明
---|---
Acquiring (Power-Up) pre-trigger data (1) | トリガー条件は未評価です。取得モードが非セグメント化バッファーであり、Storage qualifier タイプが連続である場合、Signal Tap Logic Analyzer ではデータのフルバッファーを収集します。
Trigger In conditions met | トリガーイン条件が発生しました。Signal Tap Logic Analyzer は最初のトリガー条件発生の待機中です。このメッセージが表示されるのは、トリガーイン条件が存在する場合のみです。
Waiting for (Power-up) trigger (1) | Signal Tap Logic Analyzer はトリガーイベント発生の待機中です。
Trigger level <x> met | トリガー条件 x が発生しました。Signal Tap Logic Analyzer は条件 x + 1 発生の待機中です。
Acquiring (power-up) post-trigger data (1) | 全トリガーイベントが発生しました。Signal Tap Logic Analyzer はトリガー後データの取得中です。
Offload acquired (Power-Up) data | JTAG チェーンで インテル Quartus Prime ソフトウェアにデータを転送中です。
Ready to acquire | Signal Tap Logic Analyzer はユーザーによるアナライザーの開始の待機中です。

注意: セグメント取得モードでは、トリガー前およびトリガー後は適用されません。

2.8. キャプチャーしたデータの表示、解析、および使用

Signal Tap Logic Analyzer インターフェイスでは、手動またはトリガーを使用してキャプチャーされたデータを調べることができます。Data 画面から目的のデータを分離するには、ドラッグ＆ズーム機能を使用します。この機能は、左クリックでインナーブルできます。

関連情報
- メモリー内のロケーションのモニタリング (121 ページ)
- Read Information from In-System Memory Commands (Processing Menu) Intel Quartus Prime Help 内
- Stop In-System Memory Analysis Command (Processing Menu) Intel Quartus Prime Help 内

2.8.1. セグメント化バッファーを使用したデータのキャプチャー

Segmented Acquisition バッファーによるキャプチャーの実行には、各取得セグメントに対して別々のトリガー条件が使用できます。このバッファーを使用すると、繰り返し発生するイベントや長期間にわたる一連のイベントのキャプチャーができます。

各取得セグメントは、非セグメント化バッファーとして機能し、アクティブ化後もデータを継続的にキャプチャーします。解析の実行にセグメント化バッファーを使用すると、Signal Tap Logic Analyzer では、連続データのキャプチャーをデータバッファー内の各取得セグメントに対して行います。トリガーフロー、あるいは各バッファーに対するトリガー条件評価のタイプと順序の定義は、Sequential trigger フロー・コントロールまたはカスタムの State-based trigger フロー・コントロールで行います。
次の図で示すセグメント化取得バッファーには4つのセグメントがあり、それを4つの別々の非セグメント化バッファーとして表しています。

図 -60: セグメント化取得バッファー

Signal Tap Logic Analyzerでは、1つのセグメントの取得が終了すると、次のセグメントに進んで新しい取得を開始します。波形ビューアーに表示されるデータ・キャプチャーは、トリガー条件の発生時期によって異なります。この図で示しているのはデータ取得方法です。トリガーマーカー（トリガー1、トリガー2、トリガー3、およびトリガー4）で参照するのは、カスタムState-based Triggerフローにおけるsegment_triggerおよびtriggerコマンドの評価です。シーケンシャル・フローでは、Triggerマーカーで参照するのはSetUpタブで指定したトリガー条件です。

Segment 1 Bufferがアクティブなセグメントで、トリガー1が発生した場合、Signal Tap Logic Analyzerではトリガー2の評価を直ちに開始します。Segment 2 Bufferのデータ取得が始まるのは、Segment 1 Bufferでポストフィル・カウントが終了するか、またはTrigger2の評価がTRUEになるかのいずれかの状態が先に発生したときです。したがって、トリガー条件がデータ・キャプチャー・シーケンス内の次のバッファーに関連付けられている場合、現在アクティブなバッファーのポストフィル・カウントを先に実行することができます。これにより、Signal Tap Logic Analyzerでは、発生したすべてのトリガー条件の正確なキャプチャーができます。未使用のサンプルは、波形ビューアーに空白のスペースとして表示されます。

図 -61: 取得セグメントのプリエンプションによるセグメント化キャプチャー

この図で言うキャプチャーでは、シーケンシャル・フロー・コントロールを使用し、各セグメントに対するトリガー条件の指定はDon’t Careです。

最後から1つ前の各セグメントでキャプチャーするサンプルは1つだけです。これは、次のトリガー条件では現在のバッファーのキャプチャーを先に実行するためです。すべてのセグメントのトリガー位置は、トリガー前として指定されます（データの10%はトリガー条件より前、データの90%はトリガー位置より後です）。最後のセグメントはトリガー条件により直ちに開始するので、セグメントに含まれるのはトリガー後のデータのみです。最後のセグメントにある3つの空のサンプルは、トリガー前サンプルからの残りで、Signal Tap Logic Analyzerによってバッファーに割り当てられたものです。

シーケンシャル・トリガー・フローの場合、Trigger Positionオプションはバッファー内のすべてのセグメントに適用されます。カスタムのstate-basedトリガー・フローによって提供される最大限の柔軟性によって、トリガー位置が定義されます。デバッグ要件に固有のトリガー位置を調整することで、割り当てられたバッファースペースを最大限に活用することができます。

関連情報
セグメント化バッファー (34ページ)
2.8.2. 各取得モード間のプレフィル書き込み動作の相違点

取得モードが異なると、キャプチャーするデータ量は、Signal Tap Logic Analyzer 実行直後とトリガー条件発生前の両方で異なります。

Continuous モードの非セグメント化パッファー

コンフィグレーションに非セグメント化パッファーがあって、Continuous モードで実行されている場合、パッファーは、トリガー条件の評価前にサンプルデータで満たされている必要があります。パッファーが満たされた後で初めて、Signal Tap ロジック・アナライザーでは JTAG 接続を介してデータの取得を開始し、トリガー条件を評価します。

Stop Analysis を実行すると、Signal Tap トリガー条件前の初回取得中にパッファーがダンプされないようにします。

Storage Qualification を持つパッファー

パッファーで Storage Qualification モードを使用している場合、Signal Tap Logic Analyzer では、すべてのトリガー条件を直ちに評価しながらサンプルデータを取得メモリーに書き込みます。この評価が特に重要になるのは、Storage Qualification をデータセットに使用する場合です。ロジック・アナライザーでトリガー条件を見逃す可能性があるのは、パッファー全体のデータ・キャプチャーを持った結果、トリガー条件を評価する場合です。

トリガーのアクティブ化が指定量のトリガー前データの発生前にされる場合、Signal Tap Logic Analyzer では、トリガー後データを用いてメモリーの充填を開始します。これには、指定したトリガー前データの量は関係ありません。例えば、トリガー位置を 50% に設定し、ロジック・アナライザーをプロセッサのリセット時にトリガーするように設定した場合、ロジック・アナライザーが起動してからターゲットシステムの電源を入れると、トリガーがアクティブになります。ただし、ロジック・アナライザーのメモリーにはトリガー後データのみが含まれ、トリガー前データは含まれません。これは、トリガーイベントの優先順位がトリガー前データのキャプチャーよりも高いためです。

2.8.2.1. 例

図 -62: Signal Tap Logic Analyzer の連続データ・キャプチャー

連続データ・キャプチャーと条件付きデータ・キャプチャーの図で示しているのは、continuous モードの非セグメント化パッファーと Storage Qualifier を使用した非セグメント化パッファーとの違いです。ロジック・アナライザー波形のコンフィグレーションは、ベーストリガー条件、64 ビットのサンプル深度、および Post trigger position です。

連続データ・キャプチャーでは、Trig1 がデータパッファーで数回発生する場合は、Signal Tap Logic Analyzer のトリガーがアクティブ化される前です。パッファーのフル充填は、ロジック・アナライザーによるとトリガー条件の評価を行う必要があります。トリガー条件の発生後、ロジック・アナライザーでは 8 つの追加サンプル（「トリガー後」位置で定義されている通り）パッファーの 12% を続けて取得します。
注:
1. 条件付きキャプチャーでは、ストレージは常にイネーブルされており、ポストフィル・カウントです。
2. Signal Tap Logic Analyzerでは、繰り返しパターンのキャプチャーに非セグメント化バッファーをConditionalモードで使用します。ロジックアナライザーのコンフィグレーションは、ベースタックトリガー条件「Trig1」と64ビットのサンプル深度です。Trigger in条件はDon't careなので、バッファーはすべてのサンプルをキャプチャーします。

条件付きキャプチャーでロジック・アナライザーによるトリガーは即時です。連続キャプチャーの場合と同様、ロジック・アナライザーによる取得の完了に使用するのは、8つのサンプルまたは64の12%（アクション・バッファーのサンプル容量）です。

2.8.3. ビットパターン用ニーモニックの作成
ニーモニック・テーブルを使用すると、バスなどのビットパターンのセットに対して、意味のある名前を割り当てることができます。ニーモニック・テーブルを作成するには次の手順を実行します。
1. Signal TapインスタンスのSetupまたはDataタブを右クリックし、Mnemonic Table Setupをクリックします。
2. ニーモニック・テーブルの作成には、ビットパターンのセットを入力し、各パターンを表すラベルを指定します。
3. テーブルを信号グループに割り当てるため、グループを右クリックし、Bus Display Formatをクリックして、ニーモニック・テーブルを選択します。
4. Setupタブで、意味のある名前を持つ基本トリガー作成のため、Trigger Conditionsカラムのエントリーを右クリックし、信号グループに割り当てたテーブルからラベルを選択します。

Dataタブで、キャプチャーしたデータが、割り当てられたニーモニック・テーブルに含まれるビットパターンと一致する場合は、Signal Tap GUIによって信号グループのデータが適切なラベルに置き換えられ、予想されるデータパターンの目視検査が簡素化されます。

2.8.4. プラグインを使用した自動ニーモニック
プラグインを使用して.stpに信号を追加すると、追加されたシグナルのニーモニック・テーブルが自動的に作成され、プラグインで定義されている信号に割り当てられます。これらのニーモニック・テーブルを手動でイネーブルするには、信号または信号グループの名前を右クリックします。Bus Display Formatショートカットメニューで、プラグインと一致するニーモニック・テーブルの名前をクリックします。

一例として、Nios IIプラグインを使用すると、デザインの信号アクティビティーのモニタリングをコード実行時にすることができます。ロジックアナライザーをセットアップして、.elfからのデータに基づくNios IIコードのファンション名でトリガーするようにすると、ファンション名の確認がトリガーク性能のInstance Address信号グループ内でできるようになり、対応する逆アセンブルコードの確認
認も**Disassembly**信号グループ内でできます。これについて図64で示しています。トリガー周辺のキャプチャー・データのサンプルは、トリガー・ファンクション名からのオフセットアドレスとして参照されます。

図-64: Nios II プラグイン使用時のDataタブ

2.8.5. デザイン内でのノードの検索

デザイン内のエラー発生源の検出にSignal Tap Logic Analyzerを使用する場合、ノード検索機能を使用してその信号を特定します。この機能は、インテル Quartus Prime開発ソフトウェアにある多数のツールのほか、デザインファイル内でも使用可能です。これにより、問題の発生源を素早く見つけ、デザインを変更して欠陥を修正することができます。インテル Quartus Prime Logic Analyzerからの信号の特定をSignal Tap開発ソフトウェアのツールまたはデザインファイル内であるには、.stpでその信号を右クリックし、Locate in <tool name>をクリックします。

ノードリストからの信号を特定するには、次のツールを使用します。
- Assignment Editor
- Pin Planner
- Timing Closure Floorplan
- Chip Planner
- Resource Property Editor
- Technology Map Viewer
- RTL Viewer
- デザインファイル

2.8.6. キャプチャーしたデータの保存

データ・キャプチャーを保存すると、Signal Tap Logic Analyzerでは、このデータをアクティブな.stpファイルに保存します。また、Data Logでは、このキャプチャーを現在のコンフィグレーションでログエントリーとして追加します。

解析の設定が**Auto-run mode**になっている場合、Logic Analyzerでは個別のエントリーをData Logに作成し、キャプチャーしたデータの保存をトリガーが発生するたびに行います。これにより、キャプチャーしたデータの確認がトリガーエベントごとにできます。

デフォルトのログ名は、Logic Analyzerによるデータ取得時のタイムスタンプに基づいています。ベスト・プラクティスとしては、データログ名をより分かりやすいものに変更します。

ログの組織は階層的になっています。つまり、Logic Analyzerではトリガーセットでキャプチャーされたデータの類似ログをグループ化します。
2.8.7. キャプチャーしたデータの他のファイル・フォーマットへのエクスポート

キャプチャーしたデータは、次のファイル・フォーマットでエクスポートして、EDA シミュレーション・ツールで使用できます。

- カンマ区切り値ファイル (.csv)
- テーブルファイル (.tbl)
- 値変更ダンプファイル (.vcd)
- ベクトル波形ファイル (.vwf)
- グラフィックス・フォーマット・ファイル (.jpg, .bmp)

キャプチャーしたデータを Signal Tap Logic Analyzer からエクスポートするには、File メニューの Export をクリックして、File Name, Export Format および Clock Period を指定します。

2.8.8. Signal Tap リストファイルの作成

.stp リストファイルには、ロジック・アナライザーで 1 つのトリガーイベントに対してキャプチャーするすべてのデータがテキスト形式で格納されます。

リストファイルの各行が対応しているのは、バッファ内にキャプチャーされたサンプル 1 つです。列が対応しているのは、そのサンプルに対してキャプチャーされた信号または信号グループの各値です。ニューモニック・テーブルの定義をキャプチャーされたデータに対して行った場合、テーブルの一一致するエントリーによってリストの数値が置き換えられます。

.stp リストファイルが特に有用になるのは、命令コードの逆アセンブリーを含むプラグインと組み合わせた場合です。命令コードの実行順序の表示は、トリガーイベントと同じ期間にすることが可能です。

.stp リストファイルを インテル Quartus Prime 開発ソフトウェアで作成するには、File > Create/Update > Create Signal TapList File の順でクリックします。

関連情報
プラグインを使用した信号の追加 (31 ページ)

2.9. Signal Tap Logic Analyzer を使用したパーシャル・リコンフィグレーション・デザインのデバッグ

PR デザインをデバッグするには Signal Tap を使用します。Signal Tap Logic Analyzer の PR サポートには、静的領域と PR 領域でのデータ取得が含まれています。さらに、1 つまたは複数の PR 領域に存在する複数のペルソナをデバッグできます。

特定のデバイスをターゲットにした PR デザインのデバッグの例については、AN 841: Signal Tap Tutorial for Intel Partial Reconfiguration Design または AN 845: Signal Tap Tutorial for Intel Partial Reconfiguration Design を参照してください。

関連情報
- AN 841: Signal Tap Tutorial for Intel Partial Reconfiguration Design
- AN 845: Signal Tap Tutorial for Intel Partial Reconfiguration Design
2.9.1. PR デザインのデバッグ時の推奨事項

次のガイドラインに従うことにより、Signal Tap Logic Analyzer を使用した PR デザインのデバッグ時に最良の結果を得ることができます。

- リビジョンごとに 1 つの .stp ファイルを含めます。
- 合成前ノードのみをタップします。Node Finder の Signal Tap: pre-synthesis でフィルターシします。
- デフォルトのペルソナ（ベースリビジョンのコンパイルで使うペルソナ）内のノードはタップしないでください。PR 実装のリビジョンを新たに作成し、それによってデフォルトのペルソナをインスタンス化します。新しいリビジョンのノードをタップします。
- PR ペルソナからタップしたすべてのノードを 1 つの .stp ファイルに保存し、ペルソナ全体のデバッグが Signal Tap ウィンドウ 1 つのみの使用でできるようにします。
- PR 領域間、または同一 .stp ファイル内の静的領域から PR 領域にタップしないでください。
- 各 Signal Tap ウィンドウで開くのは .stp ファイル 1 つだけです。したがって、複数のパーティションを同時にデバッグするには、スタンドアロン Signal Tap ウィンドウをコマンドラインから開いてください。

関連情報

パーシャル・リコンフィグレーション・デザインの作成
インテル Quartus Prime プロ・エディション パーシャル・リコンフィグレーション ユーザーガイド内

2.9.2. デバッグ用パーシャル・リコンフィグレーション・デザインのセットアップ

PR 領域をデバッグするには、SLD JTAG デザインのインスタンス化をベースリビジョンの生成時に行い、デバッグ・コンポーネントの定義をすべての PR ペルソナに対して行ってください。必要に応じて、信号を指定して静的領域でタップすることもできます。

図 -65: Signal Tap でのデバッグ用 PR デザインのセットアップ

デザイン内のすべての PR ペルソナのコンフィグレーション後に PR デザインフローを続行できます。

関連情報

- パーシャル・リコンフィグレーション・デザイン用のデバッグ・ファブリック (20 ページ)
- パーシャル・リコンフィグレーション・デザイン・フロー
2.9.2.1. デバッグ用静的領域の準備

PR デザインの静的領域をデバッグするには、次を実行します。
1. 静的領域内のノードを排他的にタップします。
2. .stp ファイルを保存します。静的領域を持つファイルを識別する名前を使用します。
3. プロジェクトで Signal Tap をイネーブルし、ベースリビジョンに .stp ファイルを含めます。

注意: デフォルトの PR ベルソナ内の信号はタップしないでください。

関連情報
Signal Tap ファイルへの信号の追加 (29 ページ)

2.9.2.2. デバッグ用ベースリビジョンの準備

ベースリビジョンで、デザインでデバッグする各 PR 領域に対して、次を実行します。
1. 静的領域の SLD JTAG Bridge Agent IP をインスタンス化します。
2. デフォルトのベルソナの PR 領域内の SLD JTAG Bridge Host IP をインスタンス化します。

IP Catalog または Platform Designer を使用して、SLD JTAG Bridge コンポーネントをインスタンス化します。

関連情報
- SLD JTAG Bridge Agent のインスタンス化 (18 ページ)
- SLD JTAG Bridge Host のインスタンス化 (19 ページ)

2.9.2.3. デバッグ用 PR ベルソナの準備

ベルソナのリビジョンをデザイン内に作成する前に、デバッグ IP コンポーネントをインスタンス化して信号をタップしてください。

デバッグする各 PR ベルソナに対して次を実行します。
1. PR ベルソナ内の SLD JTAG Bridge Host をインスタンス化します。
2. PR ベルソナ内の合成前ノードのみをタップします。
3. 新しい .stp ファイルに保存します。ベルソナを識別する名前を選択します。
4. 新しい .stp ファイルを実装リビジョンで使用します。

デバッグしたくない特定のベルソナがある場合は、tdo 出力信号を 0 に駆動します。

関連情報
- SLD JTAG Bridge Host のインスタンス化 (19 ページ)
- Signal Tap ファイルへの信号の追加 (29 ページ)

2.9.3. PR デザイン内でのデータ取得の実行

.sof ファイルと .rbf ファイルを、デバッグするリビジョンに対して生成したら、デバイスのプログラミングと Signal Tap Logic Analyzer を使用したデバッグができます。
データ取得には、次を実行します。
1. ベースイメージをデバイスにプログラミングします。
2. デバイスの実装にディスパッチ・リコンフィグレーションを実行します。
3. Signal Tap Logic Analyzer を開くため、インテル Quartus Prime 開発ソフトウェアで Tools > Signal Tap Logic Analyzer の順でクリックします。
   Logic Analyzer が起動し、現在のアクティビティ設定で設定した .stp ファイルがロードします。
4. デザイン内の他の領域をデバッグするには、新しい Signal Tap ヴィンデウを開きます。これには、他の領域の .stp ファイルを インテル Quartus Prime のメイン・ウィンドウから開きます。
   もしくは、次のコマンドラインを使用します。
   ```
quartus_stpw <stp_file_other_region.stp>
```
5. Signal Tap を使用してデザインをデバッグします。

別の中リビジョンをデバッグするには、デザインのパーシャル・リコンフィグレーションを実行してください。このとき、対応する .rbf ファイルを使用します。

関連情報
- ターゲットデバイスのプログラム (72 ページ)
- Signal Tap Logic Analyzer の実行 (73 ページ)
- キャプチャーしたデータの表示、解析、および使用 (77 ページ)

2.10. Signal Tap Logic Analyzer を使用したブロックベースのデザインのデバッ グ

インテル Quartus Prime プロ・エディション開発ソフトウェアでサポートしているブロックベースのデザインフローの検証には、Signal Tap ロジック・アナライザーを使用します。

ブロックベースのデザインの検証が必要なのは、パーティション内のロジックの可視性と Signal Tap ロジック・アナライザとの通信を確保するためのプランニングです。準備手順は、再利用するかコア・パーティションかルート・パーティションかによって異なります。

再利用可能なブロックを使用したデザインについては、インテル Quartus Prime プロ・エディション ユーザーガイド ブロックベースのデザインを参照してください。ステップバイステップのブロックベースのデザインのデバッグ手順については、AN 847: デザインブロック再利用 Signal Tap のチュートリアル: インテル Arria 10 FPGA 開発ボード用を参照してください。

関連情報
- インテル Quartus Prime プロ・エディション ユーザーガイド ブロックベースのデザイン
- AN 847：デザインブロックを再利用した Signal Tap のチュートリアル：インテル Arria 10 FPGA 開発ボード用

2.10.1. コア・パーティションの再利用による Signal Tap

再利用可能なコア・パーティションで検証を実行するには、デザインマップのプロジェクトで目的の信号を識別し、その信号が Signal Tap ロジック・アナライザーのインスタンスで表示されるようにしてください。インテル Quartus Prime 開発ソフトウェアでサポートしている 2 通りの方法では、コア・パーティション信号を表示して検証に使用します。
図 -66: 再利用コア・パーティションを使用したコンシューマー・デバッグのセットアップ

2.10.1.1. パーティション境界ポート

パーティション境界ポートでは、コア・パーティション・ロジックを最上位パーティションに公開します。境界ポートでは、階層ブロックの管理を簡素化するために、RTL を変更せずにロジック層をトンネリングします。

デベロッパー・プロジェクトでは、境界ポートの識別と作成をコア・パーティションのすべての潜在的 Signal Tap ポイントに対して行ってください。パーティション境界ポートの作成には、QSF アサインメントを介するか、または Assignment Editor の Create Partition Boundary Ports アサインメントを使用します。パスを割り当てると、アサインメントはデバッグポートのルート名に適用され、各ビットが列挙されます。

デベロッパー・プロジェクトでは、パーティション境界ポートをブラック・ボックス・ファイルに含めてください。このアクションにより、これらのポートのタッピングをコンシューマー・プロジェクトの合成前ノードまたはポストフィット・ノードとして行うことができます。

コンシューマー・プロジェクトでは、再利用パーティションの合成後で、Create Partition Boundary Ports 持つ有効なポートすべてが表示されます。次に、境界ポートをタップし、最上位パーティションの Signal Tap インスタンスに接続します。ロジックのタッピングは、最上位パーティションからこの Signal Tap インスタンスにすることもできます。したがって、コンシューマー・プロジェクトに必要のは 1 つの Signal Tap インスタンスだけで、最上位と再利用コア・パーティションの両方をデバッグします。

コンパイル後、パーティション境界ポートの検証を Create Partition Boundary Ports レポートで行います。このレポートは、Synthesis レポートの In-System Debugging フォルダーに生成されます。

2.10.1.1.1. パーティション境界ポートを使用したコア・パーティション再利用のデベロッパー・フロー

デバッグ用デザインの準備をパーティション境界ポートを使用してデベロッパー・プロジェクトで行うには、次の手順に従います。
1. コア・パーティションを作成します。
2. 合成プロジェクトで、コアロジックのデザイン・パーティションを定義します。
3. パーティション境界ポートの定義
4. デザインをコンパイルします。
   デザインの QSF に EXPORT_PARTITION_SNAPSHOT_SYNTHESIZED または EXPORT_PARTITION_SNAPSHOT_FINAL アサインメントが含まれている場合、Compiler では output_files ディレクトリに .qdb を自動生成します。
5. 必要に応じて、コンパイルレポートをチェックし、パーティション境界ポートのリストを検索します。
5. コンパイルで .qdb ファイルが自動生成されない場合は、Project > Export Design Partition をクリックします。

デフォルトでは、.qdb ファイルには Signal Tap パーティションに関連付けられている任意の HDL インスタンスが含まれています。ただし、そのインスタンスを削除し、コアを再コンパイルして再エクスポートした場合は例外です。

6. ブラック・ボックス・ファイルを作成します。

ブラック・ボックス・ファイルに含まれるのは、ポート、モジュール、またはエンティティーの定義のみです。ロジックは含まれません。

7. ファイルをコンシューマー・プロジェクトにコピーします。

必要に応じて、デベロッパー・プロジェクトのルート・パーティションとコア・パーティション内の信号の検証に Signal Tap Logic Analyzer を使用します。

各手順の詳細インストラクションについては、AN 847：デザインブロックを再利用した Signal Tap のチュートリアル：インテル Arria 10 FPGA 開発ボード用のコア・パーティション再利用のデバッグ (デベロッパー) を参照してください。

関連情報

• デザイン・パーティションの作成
  インテル Quartus Prime プロ・エディション ユーザーガイド ブロックベースのデザイン 内

• コア・パーティション再利用のデバッグ (デベロッパー)
  AN 847：デザインブロックを再利用した Signal Tap のチュートリアル：インテル Arria 10 FPGA 開発ボード用 内

### 2.10.1.1.2. パーティション境界ポートを使用したコア・パーティション再利用のコンシューマー・フロー

コンシューマー・プロジェクトでの再利用パーティションのインスタンス化にパーティション境界ポートを使用している場合、そのプロジェクトのデバッグに Signal Tap Logic Analyzer を使用するには、次的手順に従います。

1. デベロッパー・プロジェクトで生成されたブラック・ボックス・ファイルを追加して合成を実行します。

2. Signal Tap ファイルを作成するには、Signal Tap HDL インスタンスのインスタンス化を最上位パーティションで行うか、あるいは Signal Tap GUI を使用します。

3. 再利用コア・パーティションから、パーティション境界ポートを HDL インスタンスに接続するか、合成後またはフィット後 Signal Tap ノードを GUI へ追加します。

4. パーティションを作成し、.qdb ファイルを割り当てます。

5. デザインをコンパイルします。

6. デバイスをプログラムします。

7. データ取得を実行します。

各ステップの詳細インストラクションについては、AN 847：デザインブロックを再利用した Signal Tap のチュートリアル：インテル Arria 10 FPGA 開発ボード用のコア・パーティション再利用のデバッグ (コンシューマー) を参照してください。
関連情報
コア・パーティション再利用のデバッグ（コンシューマー）
AN 847：デザインブロックを再利用したSignal Tapのチュートリアル：インテル Arria 10 FPGA開発ボード用内

2.10.1.2. Signal Tap HDLインスタンス

デベロッパープロジェクトでは、Signal Tap HDLインスタンスを再利用可能なコア・パーティション内に作成し、目的の信号をそのインスタンスに接続します。Compilerによってパーティション内のSignal Tapインスタンスのトップレベルの可視性が保証されます。ルート・パーティションとコア・パーティションによってHDLインスタンスが分離されているため、Signal Tapファイルも別々です。

コンシューマーは、Signal Tapファイルをデザイン内にある各HDLインスタンスに対して生成する必要があります。

2.10.1.2.1. Signal Tap HDLインスタンスを使用したコア・パーティション再利用のデベロッパー・フロー

コア・パーティション・デバッグ用デザインの準備をSignal Tap HDLインスタンスを使用してデベロッパー・プロジェクトで行うには、次の手順に従います。

1. コア・パーティションを作成します。
   合成プロジェクトで、デザイン・パーティションの定義をコアロジックに対して行います。

2. Signal Tap HDLインスタンスをコア・パーティションに追加します。

3. コア・パーティション内の目的のノードをSignal Tap HDLインスタンスに追加して、.stpファイルに保存します。
   注意: コアにあるインスタンス内ルートレベル・パーティションからの信号はタップしないでください。

4. デザインをコンパイルします。
   デザインのQSFにEXPORT_PARTITION_SNAPSHOT_SYNTHESIZEDまたはEXPORT_PARTITION_SNAPSHOT_FINALアサインメントが含まれている場合、Compilerではoutput_filesデレクトリに.qdbを自動生成します。

5. コンパイルで.qdbファイルが自動生成されない場合は、Project > Export Design Partitionをクリックします。
   デフォルトでは、.qdbファイルにはSignal Tapパーティションに関連付けられている任意のHDLインスタンスが含まれています。ただし、そのインスタンスを削除し、コアを再コンパイルして再エクスポートした場合は例外です。

6. ブラック・ボックス・ファイルを作成します。
   ブラック・ボックス・ファイルに含まれるのは、ポート、モジュール、またはエンティティーの定義のみです。ロジックは含まれません。

7. ファイルをコンシューマー・プロジェクトにコピーします。
   .qdbファイル、ブラック・ボックス・ファイル、その他必要なデータを含めます。

コア・パーティションをセットアップしてコンシューマー・プロジェクトで検証するほかに、コア・パーティションまたはルート・パーティションのデバッグの実行にもデベロッパー・プロジェクト上のSignal Tap Logic Analyzerを使用することができます。
2. Signal Tap Logic Analyzer を使用したデザインのデバッグ
UG-20139 | 2018.09.24

2.10.1.2.2. Signal TapHDL インスタンスを使用したコア・パーティション再利用のコンシューマー・フロー

再利用パーティションをインスタンス化するコンシューマー・プロジェクトをデバッグするには Signal Tap HDL インスタンスを使用します。

1. デベロッパー・プロジェクトで生成されたブラック・ボックス・ファイルを追加して合成を実行します。
2. パーティションを作成し、.qdb ファイルを割り当てます。
3. Signal Tap ファイルを作成するには、Signal Tap HDL インスタンスのインスタンス化を最上位パーティションで行うか、あるいは Signal Tap GUI を使用します。
4. デザインをコンパイルします。
5. Signal Tap ファイルを再利用コア・パーティションに対して作成するために、quartus_stp コマンドを使用します。
6. デバイスをプログラムします。
7. 最上位パーティションのハードウェア検証を手順 3 で定義した Signal Tap インスタンスを使用して行います。
8. 再利用コア・パーティションのハードウェア検証を手順 5 で定義した Signal Tap インスタンスを使用して行います。

2.10.2. ルート・パーティションを再利用した Signal Tap

ルート・パーティーを再利用するデザインでは、ルート・パーティーとコア・パーティーのデバッグを個別にイネーブルします。このとき、別々の.stp ファイルを各パーティーで使用します。デベロッパー・プロジェクトで Signal Tap をルート・パーティーにイネーブルします。さらに、デバッグ・ファブリックの予約済みコア・パーティーへの拡張をデバッグブリッジを使用して行います。このブリッジにより、Signal Tap の後続のインスタンスが、コンシューマー・プロジェクト内のコア・パーティー行できることになります。

デバッグブリッジを使用するには、SLD JTAG Bridge Agent インテル FPGA IP と SLD JTAG Bridge Host インテル FPGA IP のインスタンス化をデザインの各予約済みコア境界で行う必要があります。SLD JTAG Bridge Agent IP のインスタンス化をルート・パーティションで行い、SLD JTAG Bridge Host IP のインスタンス化をコア・パーティションで行います。

図 -67: 再利用コア・パーティションを使用したデバッグのセットアップ
関連情報

SLD JTAG Bridge (16 ページ)

2.10.2.1. ルート・パーティション再利用のデベロッパーフロー

デベロッパープロジェクトで、再利用可能なルート・パーティションを作成し、SLD JTAG Bridge をインスタンス化します。このセットアップにより、コア・パーティションの後続の検証が可能になります。

1. 予約済みコア・パーティションを作成し、Logic Lock 領域を定義します。
2. SLD JTAG Bridge Agent の生成とインスタンス化をルート・パーティションで行います。
   エージェントとホストを組み合わせると、予約済みコア・パーティションのデバッグをコンシューマー・プロジェクトで行うことが可能になります。
3. SLD JTAG Bridge Agent の生成とインスタンス化をコア・パーティションで行います。
4. Signal Tap をルート・パーティションに追加して、目的の信号をタップします。
   このアクションにより、ルート・パーティションのデバッグをデベロッパープロジェクトとコンシューマー・プロジェクトで実行することが可能になります。
5. ルート・パーティションのコンパイル、エクスポートを合成スナップショットまたは最終スナップショットで行い、ファイルをコンシューマー・プロジェクトにコピーします。
   コンシューマー・プロジェクトにコピーする必要があるファイルは、デザインのターゲットデバイスによって異なります。
   - インテル Arria 10 デバイスファミリーをターゲットにしたデザインでは、.qdb および .sdc ファイルをコピーします。
   - インテル Stratix 10 デバイスファミリーをターゲットにしたデザインでは、.qdb ファイルをコピーします。
   複数の子パーティションを含むデザインでは、デザイン内の階層パスと JTAG Bridge Instance Agent の関連インデックスをコンシューマーに提供する必要があります。

必要に応じて、デベロッパープロジェクトのデザインの検証ができます。
各ステップの詳細インストラクションについては、AN 847 : デザインブロックを再利用した Signal Tap のチュートリアル : インテル Arria 10 FPGA 開発ボード用 のルート・パーティション再利用のデバッグ (デベロッパー) を参照してください。

関連情報

ルート・パーティション再利用のデバッグ (デベロッパー)
AN 847 : デザインブロックを再利用した Signal Tap のチュートリアル : インテル Arria 10 FPGA 開発ボード用 内
Signal Tap ルート・パーティションを再利用したデザインを検証を実行するには、次の手順を実行します。
1. カスタマー・プロジェクトにファイルを追加します。
2. SLD JTAG Bridge Host の生成とインスタンス化をコア・パーティションで行います。
3. 合成
4. HDL または Signal TapGUI を使用して Signal Tap インスタンスをコア・パーティションに作成し、合成前の信号を追加します。
   注意: コア・パーティション内の信号のみをタップできます。
5. デザインをコンパイルします。
6. Signal Tap ファイルを再利用コア・パーティションに対して作成するために、quartus_stp コマンドを使用します。
7. デバイスをプログラムします。
8. 予約済みコア・パーティションのハードウェア検証を、ステップ 3 で定義した Signal Tap インスタンスを使用して行います。
9. 再利用ルート・パーティションのハードウェア検証を、ステップ 4 で定義した Signal Tap インスタンスを使用して行います。
各ステップの詳細インストラクションについては、AN 847: デザインブロックを再利用した Signal Tap のチュートリアル：インテル Arria 10 FPGA 開発ボード用のルート・パーティション再利用のデバッグ（コンシューマー）を参照してください。

関連情報
ルート・パーティション再利用のデバッグ（コンシューマー）
AN 847: デザインブロックを再利用した Signal Tap のチュートリアル：インテル Arria 10 FPGA 開発ボード用 内

2.10.3. インポートしたスナップショットのデバッグ
コンシューマー・プロジェクトでインポートする .qdb ファイルには、再利用可能なパーティションのスナップショットが含まれています。スナップショットの種類によっては、信号を追加して、それを Signal Tap Logic Analyzer でのデバッグに使用することができます。

重要: ベスト・プラクティスとして、目的の信号を指定し、デベロッパープロジェクトでのデバッグに使用します。
新しい信号を再利用パーティションの Signal Tap インスタンスに追加するには、Fitter がこれらの信号に接続して配線できるようにする必要があります。これが可能の場合は次の場合一のものです。
- 再利用パーティションに合成スナップショットが含まれている場合 - 再利用パーティションに配置済みまたは最終のスナップショットが含まれている場合は、Signal Tap インスタンスへの信号の追加はできません。これは、境界ポートは追加で作成できないからです。
- タップする信号がポストフィットの場合 - 合成前 Signal Tap 信号の追加はできません。これは、追加にはパーティションの再合成が必要だからです。

関連情報
Signal Tap デバッグに使用できない信号 (31 ページ)
2.10.3.1. フィット後ノードを使用した合成スナップショットのデバッグ

コンシューマー・プロジェクトで、ポストフィット信号を追加してそれを Signal Tap Logic Analyzer でデバッグに使用することができるノードをインポートした場合です。

コンシューマー・プロジェクトでポストフィット・ノードをタップするには、次の手順を実行します。
1. コンシューマー・プロジェクトのフィッターステージを通じてパーティションをコンパイルします。
2. コンシューマー・デザインに Signal Tap を追加し、ポストフィット Signal Tap ノードを追加します。
3. デザインの再コンパイルを Place ステージから行うため、Processing > Start > Start Fitter (Place) をクリックします。
　フィッターによって Signal Tap ノードを既存の合成ノードに接続します。

2.11. その他の機能

Signal Tap Logic Analyzer で提供しているオプション機能は、タスクフローに限定されません。次の手法は特定のシナリオで有用です。

2.11.1. デザイン・インスタンスからの Signal Tap ファイルの作成

.stp ファイルの生成に対する GUI サポートのほかに、インテル Quartus Prime 開発ソフトウェアでは、HDL ソースファイルで定義されたロジックからの Signal Tap インスタンスの生成もサポートしています。この手法が有用なのは、ランタイムでコンフィグレーション可能なトリガー条件の変更、データの取得、Signal Tap ユーティリティーを介して取得したデータログ上のデータ表示を行う場合です。

2.11.1.1. デザイン・インスタンスからの.stp ファイルの生成

.stp ファイルの生成は、デザイン内のパラメーター化された HDL インスタンスから行うには、次の手順で行います。
1. インテル Quartus Prime ロジック・アナライザーの 1 つまたは複数の HDL インスタンスを含む Signal Tap プロジェクトを展開あるいは作成します。
2. Processing > Start > Start Analysis & Synthesis の順でクリックします。
3. File > Create/Update > Create Signal Tap File from Design Instance(s) の順でクリックします。
4. 生成する .stp ファイルの位置を指定し、Save をクリックします。
図 -68: **Signal Tap File from Design Instances** ダイアログボックスからのファイルの作成

プロジェクトにパーシャル・リコンフィグレーション・パーティションが含まれる場合、Create **Signal Tap File from Design Instance(s)** ダイアログボックスには、プロジェクト内の PR パーティションのツリービューが表示されます。パーティションをこのビューから選択し、Create **Signal Tap file** をクリックします。この結果生成される .stp ファイルには、対応する PR 領域のすべての HDL インスタンスが含まれます。この結果としての .stp ファイルには、ネスト化されたパーシャル・リコンフィグレーション・パーティションのインスタンスは含まれません。

図 -69: .stp ファイル生成用パーティションの選択

・stp ファイルの生成が成功すると、**Signal TapLogic Analyzer** が表示されます。ランタイムでコンフィグレーション可能なトリガー条件を除き、すべてのフィールドは読み出し専用です。
図 -70: 生成された .stp ファイル

関連情報

- Create Signal Tap File from Design Instances
  Intel Quartus Prime Help 内
- カスタムトリガーの HDL オブジェクト (50 ページ)

2.11.2. Signal Tap MATLAB MEX 関数を使用したデータのキャプチャー

MATLAB を DSP デザインに使用する場合、データ取得を インテル Quartus Prime Logic Analyzer
から MATLAB 環境のマトリクスへ直接行うために、Signal Tap 開発ソフトウェアに組み込まれている
MATLAB MEX 関数 alt_signaltap_run を呼び出します。MATLAB MEX 関数をループ内で使
用する場合、同じ時間内で実行できる取得数は、インテル Quartus Prime を Signal Tap 開発ソフト
ウェアで使用する場合と同じです。

注意: Signal Tap MATLAB MEX 関数の使用は、インテル Quartus Prime 開発ソフトウェアの
Windows*バージョンと Linux バージョンで可能です。この関数と互換性があるのは、MATLAB
Release 14 オリジナル・リリース・バージョン 7 およびそれ以降のバージョンです。

インテル Quartus Prime 開発ソフトウェアおよび MATLAB 環境を設定して Signal Tap の取得を実
行するには、次を実行します。
1. インテル Quartus Prime 開発ソフトウェアで .stp ファイルを作成します。
2. Signal Tap Logic Analyzer Editor の Data タブのノードリストで、信号および信号グループを
編成し、希望順で MATLAB マトリクスに表示するようにします。

インポートしたマトリクスの各列で表されるのは、単一の Signal Tap 取得サンプルです。一方、各
行で表されるのは信号または信号のグループで、その順序は Data タブで定義したとおりになっています。
信号グループが Signal Tap Logic Analyzer で取得され、MATLAB MEX 関数へ転送される場合、その幅制限は 32 信号です。使用する MATLAB* MEX 関数のバスまたは信号グループに 32 を超える信号が含まれている場合、そのグループを小さいグループに分割して、幅制限を超えないようにします。

3. .stp ファイルを保存し、デザインをコンパイルします。デバイスをプログラムリングし、Signal Tap Logic Analyzer を実行して、トリガー条件と信号の取得が正常に機能するようにします。

4. MATLAB 環境では、インテル Quartus Prime バイナリ－ディレクトリ－をパスに追加するのに次のコマンドを使用します。

```matlab
addpath <Quartus install directory> \win
```

MEX 関数のヘルプファイルの表示には、次のコマンドを MATLAB に入力します。演算子は付けません。

```matlab
alt_signaltap_run
```

5. MATLAB MEX 関数を使用してデバイスへの JTAG 接続を開き、Signal Tap Logic Analyzer を実行してデータを取得します。データの取得終了後、JTAG 接続を閉じます。

JTAG 接続を開いて、キャプチャーしたデータの取得を stp* という MATLAB マトリックスに直接取得するには、次のコマンドを使用します。

```matlab
stp = alt_signaltap_run \
{('<stp filename>','signed'|'unsigned'),['<instance names>','<signalset name>','<trigger name>']);
```

データをキャプチャーする際はファイル名の割り当てが必要です。例えば、<stp filename>が MATLAB MEX 関数の要件です。他の MATLAB MEX 関数のオプションを次の表に示します。

<table>
<thead>
<tr>
<th>オプション</th>
<th>使用方法</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>signed</td>
<td>'signed'</td>
<td>Signal Tap 関数では、信号のグループデータを 32 ビットの 2 の補数符号付き整数に変換します。そのグループの MSB は、Signal Tap Data タブで定義されているとおり、符号ビットです。signed オプションは、データを符号なし整数として保持します。デフォルトは signed です。</td>
</tr>
<tr>
<td>unsigned</td>
<td>'unsigned'</td>
<td>Signal Tap インスタンスを指定するのは、複数のインスタンスが定義されている場合です。デフォルトは、.stp の最初のインスタンス auto_signaltap_0 です。</td>
</tr>
<tr>
<td>&lt;instance name&gt;</td>
<td>'auto_signaltap_0'</td>
<td>Signal Tap データログから信号のセットとトリガーを指定するのは、複数のコンフィグレーションが .stp に存在する場合です。デフォルトは、ファイル内のアクティブな信号セットとトリガーです。</td>
</tr>
</tbody>
</table>

データ取得中、Verbose モードをイネーブル／ディスエーブルすると、ロジック－アナライザーのステータスが確認できます。Verbose モードをイネーブルまたはディスエーブルするには、次のコマンドを使用します。

```matlab
alt_signaltap_run('VERBOSE_ON');-alt_signaltap_run('VERBOSE_OFF');
```

データ取得完了後、JTAG 接続を閉じるために次のコマンドを使用します。

```matlab
alt_signaltap_run('END_CONNECTION');
```

MATLAB での MATLAB MEX ファンクションの使用に関する詳細は、MATLAB Help を参照してください。
2.11.3. ラボ環境での Signal Tap の使用

スタンドアロン・バージョンの Signal Tap Logic Analyzer がインストール可能です。このバージョンが特に有用なのは、ラボ環境で完全版 インテル Quartus Prime のインストール要件を満たすワーク・ステーションがない場合や、インテル Quartus Prime 開発ソフトウェアの完全インストールのライセンスがない場合です。スタンドアロン・バージョンの Signal Tap Logic Analyzer は、インテル Quartus Prime スタンダードの Programmer に含まれており、インテル FPGA およびプログラマブル・デバイスのウェブサイトのダウンロード・センターでダウンロードできます。

2.11.4. Signal Tap Logic Analyzer を使用したリモートデバッグ

2.11.4.1. ローカル PC と SoC を使用したデバッグ

Intel Logic Analyzer の System Console を使用して、インテル FPGA SoC をリモートデバッグします。この方法で必要なのは、ローカル PC1 台、既存の TCP/IP 接続、リモート・プログラミング・デバイスおよびインテル FPGA SoC です。

関連情報
TCP/IP を介したリモート・ハードウェアのデバッグ

2.11.4.2. ローカル PC とリモート SoC を使用したデバッグ

Signal Tap Logic Analyzer を使用して、遠隔地にある PC に接続されているデバイス上で動作するデザインをデバッグできます。

リモート・デバッグ・セッションを実行するには、次のセットアップが必要です。

• インテル Quartus Prime 開発ソフトウェアをローカル PC にインストール済み
• インテル Quartus Prime の Signal Tap Logic Analyzer を含むスタンドアロンの Programmer にインストール済み
• 遠隔地にある PCB 上のデバイスに接続されたプログラミング・ハードウェア
• TCP/IP プロトコル接続

2.11.4.2.1. 機器のセットアップ

1. 遠隔地にある PC に、Signal Tap スタンダードの Programmer に含まれているスタンドアロン・バージョンの インテル Quartus Prime Logic Analyzer、または インテル Quartus Prime 開発ソフトウェアをインストールします。
2. リモート・コンピューターをまたは インテル FPGA ダウンロード・ケーブルなどの Intel プログラミング・ハードウェアに接続します。
3. ローカル PC にフルバージョンの インテル Quartus Prime 開発ソフトウェアをインストールします。
4. ローカル PC のリモート PC への接続を、LAN 経由で TCP/IP プロトコルを使用して行います。
2.11.5. コンフィグレーション・ビットストリーム・セキュリティ付きのデバイスにおける
Signal Tap Logic Analyzer の使用

一部のデバイスファミリーでサポートしているコンフィグレーション実行中のビットストリームの復号化
では、オンドバイスの AES 復号化エンジンを使用します。これまで同様、Signal Tap Logic Analyzer
を使用して、FPGA 内の機能データの解析を行うことは可能ですが、注意が必要なのは、JTAG コンフィ
グレーションの実行は、セキュリティキーをデバイスにプログラミングした後はできないということです。

インテル FPGA では、デザインのプロトタイプとデバッグのフェーズでの非暗号化ビットストリームの使
用をお勧めしています。非暗号化ビットストリームを使用することで、新規のプログラミング・ファイルの
生成および JTAG 接続を介したデバイスのリコンフィグレーションがデバッグサイクル中に可能になり
ます。

Signal Tap ロジック・アナライザーを暗号化ビットストリームで使用しなければならない場合は、まず、
暗号化コンフィグレーション・ファイルでデバイスをコンフィグレーションするために、Passive Serial
（PS）、Fast Passive Parallel (FPP)、または Active Serial (AS) コンフィグレーション・モードを使用
します。このデザインには、Signal Tap Logic Analyzer のインスタンスを少なくとも 1 つ含める必要
があります。FPGA コンフィグレーションをデザインの Signal Tap ロジック・アナライザーのインス
タンスで行った後、Signal Tap 開発ソフトウェアで インテル Quartus Prime ロジック・アナライザーを開く
ときにチェーンをスキャンすると、データ取得を JTAG 接続で行う準備が整います。

2.11.6. Signal Tap ロジック・アナライザーで使用する FPGA リソースのモニタリング

Signal Tap ロジック・アナライザーに内蔵されている Resource Estimator では、各ロジック・アナラ
イザーのインスタンスで使用するロジックリソースとメモリー量の計算が行われます。さらに、ロジック・
アナライザーで最も要求の多いオンチップリソースはメモリー使用量なので、Resource Estimator で
は、デザイン内の RAM の合計使用量と使用可能な RAM の合計との比率を最後のコンパイル結果に基
づいてレポートします。Resource Estimator で警告が表示されるのは、「no-fit」が発生する可能性が
ある場合です。

リソース使用量（インスタンス別および合計）の確認は、Signal Tap Logic Analyzer Editor の
Instance Manager ペインの列でできます。この機能を使用するのは、デザインのリソー
スが不足している場合です。

ロジックエレメントの値は、Resource Usage Estimator による報告と実際のリソース使用量では、
10％程度異なる場合があります。

2.12. デザイン例：Signal Tap Logic Analyzer の使用方法

このデザイン例のシステムに含まれる多くのコンポーネントには、Nios プロセッサー、ダイレクト・メモ
リー・アクセス (DMA) コントローラー、オンチップメモリー、外部 SDRAM メモリーへのインタフェイ
スなどがあります。ボタンを押すと、プロセッサーでは DMA 転送を開始します。この解析には、Signal
Tap Logic Analyzer を使用します。この例の Nios プロセッサーでは、単純な C プログラムをオンチッ
プメモリーから実行し、ボタンが押されるまで待機します。
2.13. カスタム・トリガー・フローのアプリケーション例

Signal Tap Logic Analyzer のカスタム・トリガー・フローが非常に有用なのは、多数のトリガー条件の整理およびアクイジョン・バッファーの正確な制御を行う場合です。この項の２つのアプリケーション例では、Signal Tap Logic Analyzer 内でカスタム・トリガー・フローを定義しています。両方の例は、簡単に State Machine Description Box に直接コピー・アンド・ペーストできます。このときステート・ディスプレイ・モードの All states in one window を使用します。

関連情報
On-Chip Debugging Design Examples

2.13.1. デザイン例 1 : カスタムトリガー位置の指定

アクイジョン・バッファーに対するアクションでは、オプションのポストカウント引数の受け取りができます。このポストカウント引数を使用すると、カスタムトリガー位置の定義をアクイジョン・バッファー内の各セグメントに対して行うことができます。

この例で示すのは、アクイジョン・バッファー内のすべてのセグメントに対するトリガー位置の適用方法です。この例の記述は、４つのセグメントに分割されたアクイジョン・バッファーのトリガーフローです。各取得セグメントの深度が 64 サンプルの場合、各バッファーのトリガー位置はサンプル #34 になります。取得停止は、すべてのセグメントが一度満たされた後です。

``` exacerbation
if (c1 == 3 && condition1)
    trigger 30;
else if (condition1)
    begin
        segment_trigger 30;
        increment c1;
    end
```

各セグメントは、非セグメント化バッファーとして機能し、信号値を使用してメモリー内容を連続的に更新します。

Data タブに表示されるバッファー停止前の最後の取得は、影響を受けるセグメントの最後のサンプル番号です。この後、影響を受けるセグメントのトリガー位置が N - post count fill によって定義されます。この N はセグメントあたりのサンプル数です。
図 -71: カスタムトリガー位置の指定

2.13.2. デザイン例 2: triggercond1 が triggercond2 と triggercond3 の間に 10 回発生する場合のトリガー

カスタム・トリガー・フローの記述は、多くの場合、イベントのシーケンスのカウントをアクイジション・バッファーのトリガー前に行う際に役立ちます。この例で示すのは、そのようなサンプルフローです。この例で使用している 3 つの基本トリガー条件は、Signal Tap Setup タブでコンフィグレーションされたものです。

この例でのアクイジション・バッファーのトリガーは、condition1 が condition3 の後に発生し、また condition3 の前にも 10 回発生したときです。condition3 が発生するのが condition1 の 10 回の繰り返し前の場合、ステートマシンは遷移し、永続的なウェイトステートになります。

```plaintext
state ST1:
if ( condition2 )
begin
    reset c1;
    goto ST2;
end
state ST2:
if ( condition1 )
    increment c1;
else if (condition3 && c1 < 10)
    goto ST3;
else if (condition3 && c1 >= 10)
    trigger;
ST3:
goto ST3;
```
2.14. Signal Tap スクリプトのサポート

インテル Quartus Prime では、スクリプト環境内の Signal Tap プロシージャーの自動化をサポートしています。これは、Tcl スクリプトとして、または実行可能ファイル quartus_stp を介して行われます。スクリプティング・コマンド・オプションの詳細については、インテル Quartus Prime Command-Line および Tcl API Help ブラウザーを参照してください。Help ブラウザを実行するには、コマンドプロンプトで quartus_sh --qhelp と入力します。

関連情報
- Tcl スクリプティング
  インテル Quartus Prime プロ・エディション スクリプティング ユーザーガイド
- コマンドライン・スクリプティング
  インテル Quartus Prime プロ・エディション スクリプティング ユーザーガイド

2.14.1. Signal Tap コマンドライン・オプション

次のオプションは、quartus_stp 実行ファイルで使用可能です。

<table>
<thead>
<tr>
<th>オプション</th>
<th>使用方法</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>--stp_file &lt;stp_filename&gt;</td>
<td>必須</td>
<td>.stp ファイル名を指定します。</td>
</tr>
<tr>
<td>--enable</td>
<td>任意</td>
<td>ENABLE_SIGNALTAP オプションの設定を、プロジェクトの .qsf ファイルで ON にします。これにより、Signal Tap Logic Analyzer が次のコンパイルで実行されます。このオプションを省略した場合、インテル Quartus Prime 開発ソフトウェアでは、.qsf ファイルの ENABLE_SIGNALTAP の現行値を使用します。後続の Signal Tap アサインメントの書き込みは、.qsf に表示される .stp に対応して行います。.qsf ファイルによる .stp ファイルの指定がない場合、--stp_file オプションを使用してください。</td>
</tr>
<tr>
<td>--disable</td>
<td>任意</td>
<td>ENABLE_SIGNALTAP オプションの設定は、プロジェクトの .qsf ファイルで OFF にします。これにより、Signal Tap Logic Analyzer は次のコンパイルでは実行されません。--disable オプションを省略すると、インテル Quartus Prime 開発ソフトウェアでは、.qsf ファイルの ENABLE_SIGNALTAP の現行値を使用します。</td>
</tr>
</tbody>
</table>

2.14.2. コマンドラインからのデータのキャプチャー

quartus_stp 実行コマンドでサポートしている Tcl インターフェースにより、データのキャプチャーが可能になります。このとき インテル Quartus Prime GUI の実行は必要ありません。

注意: Signal Tap Tcl コマンドの実行は、インテル Quartus Prime 開発ソフトウェアの Tcl コンソール内からはできません。

Signal Tap Logic Analyzer の Tcl コマンドを含む Tcl スクリプトを実行するには、次を使用します。

```
quartus_stp -t <Tcl file>
```
例-5: データの連続的なキャプチャー

次の抜粋で示すコマンドを使用すると、データの連続的なキャプチャーが可能です。キャプチャーによってトリガー条件が満たされると、Signal Tap Logic Analyzerではキャプチャーを開始し、データをデータログに格納します。

```bash
# Open Signal Tap session
open_session -name stp1.stp

### Start acquisition of instances auto_signaltap_0 and auto_signaltap_1 at the same time

# Calling run_multiple_end starts all instances
run_multiple_start

run -instance auto_signaltap_0 -signal_set signal_set_1 -trigger trigger_1 -data_log log_1 -timeout 5
run -instance auto_signaltap_1 -signal_set signal_set_1 -trigger trigger_1 -data_log log_1 -timeout 5

run_multiple_end

# Close Signal Tap session
close_session
```

関連情報

::quartus::stp
Intel Quartus Prime Help 内

### 2.15. Signal Tap Logic Analyzer を使用したデザインのデバッグ

この章には次の改訂履歴が適用されます。

<table>
<thead>
<tr>
<th>ドキュメント・バージョン</th>
<th>インテル Quartus Prime バージョン</th>
<th>変更内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.09.24</td>
<td>18.1.0</td>
<td>• ブロックベースのフローでのデザインのデバッグに関する内容を追加しました。• トピック名をタップできない信号から Signal Tap のデバッグに使用できない信号に変更しました。</td>
</tr>
<tr>
<td>2018.08.07</td>
<td>18.0.0</td>
<td>ドキュメントのタイトルをデバッグツールユーザーガイド：インテル Quartus Prime プロ・エディションに戻しました。</td>
</tr>
<tr>
<td>2018.07.30</td>
<td>18.0.0</td>
<td>パーシャル・リコンフィグレーションの項を更新して PR フローの変更を反映しました。</td>
</tr>
<tr>
<td>2018.05.07</td>
<td>18.0.0</td>
<td>• Stratix 10 デバイス用に最適化されていない Signal Tap IP を記載した注記を追加しました。• PR デザインのデバッグ・ファブリックに関する情報をシステム・デバッグ・ツールの概要の章に移動しました。• インテル Stratix 10 デバイスの Rapid Recompile サポートの制限を削除しました。</td>
</tr>
<tr>
<td>2017.11.06</td>
<td>17.1.0</td>
<td>• インテル Stratix 10 デバイスのインクリメンタル配線のサポートを追加しました。• サポートされていない FSM 自動検出を削除しました。• Data Log ベインに関する情報を明確になりました。• Data Log の図を更新し、名前をシンプルな Data Log に変更しました。• Advanced Trigger Condition タブへのアクセスの図を追加しました。• コマンドライン・フローに関する古い情報を削除しました。</td>
</tr>
</tbody>
</table>

continued...
2. Signal Tap Logic Analyzer を使用したデザインのデバッグ

<table>
<thead>
<tr>
<th>ドキュメント・バージョン</th>
<th>インテル Quartus Prime バージョン</th>
<th>変更内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017.05.08</td>
<td>17.0.0</td>
<td>• オープン・スタンドアロン Signal Tap Logic Analyzer GUI を追加しました。&lt;br&gt;• Signal Tap Logic Analyzer を使用したパーシャル・リコンフィグレーション・デザインのデバッグを追加しました。&lt;br&gt;• Create Signal Tap File from Design Instance(s) の数値を更新しました。</td>
</tr>
<tr>
<td>2016.10.31</td>
<td>16.1.0</td>
<td>• インテルへのブランド変更を行いました。&lt;br&gt;• Create SignalTap II File from Design Instance(s) を追加しました。&lt;br&gt;• サポートされていない Talkback 機能への参照を削除しました。</td>
</tr>
<tr>
<td>2016.05.03</td>
<td>16.0.0</td>
<td>• Pipeline Factor の指定を追加しました。&lt;br&gt;• Comparison トリガー条件を追加しました。</td>
</tr>
<tr>
<td>2015.11.02</td>
<td>15.1.0</td>
<td>• Quartus II のインスタンスを変更してインテル Quartus Prime にしました。&lt;br&gt;• サポートされていない Talkback 機能への参照を削除しました。</td>
</tr>
<tr>
<td>2015.05.04</td>
<td>15.0.0</td>
<td>• 浮動小数点表示フォーマットの内容を SignalTap II ロジック・アナライザの機能と利点の表に追加しました。</td>
</tr>
<tr>
<td>2014.12.15</td>
<td>14.1.0</td>
<td>• フィッター設定、Analysis &amp; Synthesis の設定、およびフィジカルシンセシス最適化の位置をコンバイラー設定に更新しました。</td>
</tr>
<tr>
<td>2014 年 12 月</td>
<td>14.1.0</td>
<td>• MAX 10 をサポート対象のデバイスとして追加しました。&lt;br&gt;• Full Incremental Compilation 設定および Post-Fit (Strict) ネットリスト・タイプの設定情報を削除しました。&lt;br&gt;• 「SignalTap II Logic Analyzer でのインクリメンタル・コンパイルの使用」から古い GUI イメージを削除しました。</td>
</tr>
<tr>
<td>2014 年 6 月</td>
<td>14.0.0</td>
<td>• DITA 変換。&lt;br&gt;• MegaWizard Plug-In Manager および Megawfunction の内容を IP Catalog およびパラメーター・エディターの内容に置き換えました。&lt;br&gt;• カスタムトリガー HDL オブジェクト、Rapid Recompile を使用した Incremental Route、および Basic OR を使用したネストされたグループのフローを追加しました。&lt;br&gt;• 次の GUI を変更しました。ツールバー、ドラッグして拡大表示、インスタンスのディスエプル/イネーブル、ログのタイムスタンプのトリガー。</td>
</tr>
<tr>
<td>2013 年 11 月</td>
<td>13.1.0</td>
<td>HardCopy 素材を削除しました。DS-5 ツールでのクロストリガーの使用に関する項を追加し、ホワイトペーパー 01198 へのリンクを追加しました。アルテラ SoC のリモートデバッグに関する項を追加し、アプリケーションノート 693 へのリンクを追加しました。MEX 機能のサポートを更新しました。</td>
</tr>
<tr>
<td>2013 年 5 月</td>
<td>13.0.0</td>
<td>• 個別トリガー条件、基本的な ORトリガー条件、およびハード・プロセッサーセンサーシステム (HPS) 外部トリガーに関する情報を含むセグメント化バッファーに対して、state-based フローを使用するための推奨事項を追加しました。&lt;br&gt;• 13-17 ページの「セグメント化バッファー」、13-21 ページの「条件付きモード」、13-16 ページの「基本トリガー条件の作成」、および 13-44 ページの「外部トリガーの使用」を更新しました。</td>
</tr>
<tr>
<td>2012 年 6 月</td>
<td>12.0.0</td>
<td>13-16 ページの表 13-5 および 13-10 ページの「SignalTap II ファイルへの信号の追加」を更新しました。</td>
</tr>
<tr>
<td>2011 年 11 月</td>
<td>11.0.1</td>
<td>テンプレートを更新しました。編集上の軽微な更新を行いました。</td>
</tr>
<tr>
<td>2011 年 5 月</td>
<td>11.0.0</td>
<td>スタンドアロンの SignalTap II ソフトウェアの要件を更新しました。</td>
</tr>
<tr>
<td>2010 年 12 月</td>
<td>10.0.1</td>
<td>ドキュメントのテンプレートを更新しました。</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>ドキュメント・バージョン</th>
<th>インテル Quartus Prime バージョン</th>
<th>変更内容</th>
</tr>
</thead>
</table>
| 2010 年 7 月             | 10.0.0                           | • 新しいアクイジション・バッファーの内容を「取り込みデータの表示、解析、および使用」セクションに追加しました。  
• プログラムされたデバイスで 16 進数の CRC 値を生成するためのスクリプト例を追加しました。  
• プロシージャーの重複するコンテンツに関する Quartus II Help への相互参照を作成しました。 |
| 2009 年 11 月            | 9.1.0                            | 内容に対する変更はありません。 |
| 2009 年 3 月             | 9.0.0                            | • 表 3–1 を更新しました。  
• 13–45 ページの「SignalTap II Logic Analyzer でのインクルメンタル・コンパイルの使用」を更新しました。  
• 図 13–33 を新たに追加しました。  
• 編集上的軽微な更新を行いました。 |
| 2008 年 11 月            | 8.1.0                            | Quartus II ソフトウェア・バージョン 8.1 のリリースに合わせて次の通り更新しました。  
• 14–25 ページの「Storage Qualifier 機能の使用」の項を新たに追加しました。  
• 14–36 ページの「Trigger Condition Flow の制御」の項の start_store および stop_store コマンドの説明を追加しました。  
• 14–63 ページの「ランタイム・コンフィグレーション可能オプション」の項を新たに追加しました。 |
| 2008 年 5 月             | 8.0.0                            | Quartus II ソフトウェア・バージョン 8.0 用に次の更新を行いました。  
• 14–24 ページの「有限ステートマシンのデバッグ」  
• State-based trigger flow control タブの resource estimator、バス検索機能、カウンターフトプラグリソースの動的表示の更新など、さまざまな GUI ユーザビリティの機能強化を文書化しました。  
• 14–16 ページの「セグメント化バッファーを使用したデータのキャプチャー」を追加しました。  
• 章全体に参照文書へのハイパーリンクを追加しました。  
• 編集上の軽微な更新を行いました。 |

関連情報
ドキュメント・アーカイブ
以前のバージョンの インテル Quartus Prime ハンドブックは、文書アーカイブを検索してください。
4. Signal Probe を使用した迅速なデザイン検証

この章で紹介するデバッグ方法では、早期アクセスを内部デバイス信号に対して提供し、デザインには影響を与えません。

インテル Quartus Prime プロ・エディションソフトウェアの Signal Probe 機能を使用すると、内部ノードのトップレベル I/O への配線が可能になります。完全配線されたデザインで開始する場合、デバッグ信号の選択および配線を、以前に予約した I/O ピン、または現在使用していない I/O ピンに対して行うことができます。

ラピッド・リコンパイル実行中、Compiler では以前の合成およびフィッティング結果を可能な限り再利用し、未変更のデザインブロックの再処理は行いません。小規模のデザイン変更を行う場合、Rapid Recompile を使用することでタイミングのばらつきと再コンパイルの合計時間が定減できます。

インテル Quartus Prime プロ・エディション Signal Probe 機能では、インテル Arria 10 およびインテル Stratix 10 デバイスファミリーをサポートしています。

関連情報
システム・デバッグ・ツールの概要 (7 ページ)

4.1. Signal Probe とラピッド・リコンパイルを使用したデバッグフロー

検証機能をデザインに追加するのに Signal Probe 配線機能を使用する場合は、次の項を参照してください。

Signal Probe ピンの予約 (104 ページ)
デザインのコンパイル (105 ページ)
Signal Probe ピンへのノードの割り当て (105 ページ)
デザインの再コンパイル (105 ページ)
Fitter レポートの接続テーブルの確認 (106 ページ)

4.1.1. Signal Probe ピンの予約

Signal Probe 用のピンの作成および予約を Tcl コマンドを使用して行います。

```
set_global_assignment -name CREATE_SIGNALPROBE_PIN <pin_name>
```

*pin_name* は Signal Probe ピン名を指します。

必要に応じて Signal Probe ピンの位置の割り当てもできます。位置の割り当てを実行しない場合は、Fitter でピンを自動配置します。
注意: デバッグプロセスの開始時から配線する内部信号が分かっている場合は、ピンの予約とノードの割り当てをコンパイル前にすることができます。この早期の割り当てによって、リコンパイル・ステップをフローから省略できます。

例-6: **Signal Probe** ピンの予約に使用する Tcl コマンド

```tcl
set_global_assignment -name CREATE_SIGNALPROBE_PIN wizard
set_global_assignment -name CREATE_SIGNALPROBE_PIN probey
```

関連情報

Constraining Designs with Tcl Scripts
インテル Quartus Prime プロ・エディション デザイン制約ユーザーガイド 内

### 4.1.2. デザインのコンパイル

デザインのフルコンパイルを実行します。インテル Quartus Prime 開発ソフトウェア、コマンドライン実行ファイル、または Tcl コマンドが多く使用可能です。

例-7: デザインのコンパイルに使用する Tcl コマンド

```tcl
execute_flow -compile
```

デザインフローのこの時点で、デバッグするノードを決定します。

関連情報

デザインのコンパイル
インテル Quartus Prime プロ・エディション Compiler ユーザーガイド 内

### 4.1.3. **Signal Probe** ピンへのノードの割り当て

コンパイル後のネットリスト内の任意のノードを Signal Probe ピンに割り当てることができます。インテル Quartus Prime 開発ソフトウェアで、**View > Node Finder** の順にクリックし、**Signal Tap: post-fitting** でフィルターを実行して配線可能なノードを表示します。

Signal Probe ピンに接続するノードの指定を Tcl コマンドを使用して行います。

```tcl
set_instance_assignment -name CONNECT_SIGNALPROBE_PIN <pin_name> -to <node_name>
```

*pin_name* ノードに接続する Signal Probe ピン名を指定します。

*node_name* 配線するノードの完全な階層パスを指定します。

例-8: 内部ノードにピンを接続する Tcl コマンド

```tcl
# Make assignments to connect nodes of interest to pins
set_instance_assignment -name CONNECT_SIGNALPROBE_PIN wizard -to sprobe_me1
set_instance_assignment -name CONNECT_SIGNALPROBE_PIN probey -to sprobe_me2
```

### 4.1.4. デザインの再コンパイル

Signal Probe ピンへのノードの割り当て後、Rapid Recompile を実行します。Rapid Recompile では、タイミングの保持とコンパイル時間の短縮のため、以前の結果を可能な限り再利用します。
Rapid Recompile の実行は、インテル Quartus Prime 開発ソフトウェア、コマンドライン実行ファイル、または Tcl スクリプトから行います。

例-9: デザインの再コンパイルに使用する Tcl コマンド

```bash
# Run the fitter with --recompile to preserve timing
# and quickly connect the Signal Probe pins
execute_module -tool fit -args (--recompile)
```

再コンパイル後、デバイスのプログラミングとデバッグが実行可能になります。

関連情報
Using Rapid Recompile
インテル Quartus Prime プロ・エディション Compiler ユーザーガイド 内

4.1.5. Fitter レポートの接続テーブルの確認

Signal Probe ピンを含むデザインをコンパイルすると、インテル Quartus Prime 開発ソフトウェアによって接続レポートテーブルが生成されます。このレポートを表示するには、Processing > Compilation Report の順でクリックし、Fitter > In-System Debugging フォルダーを開き、Connections to Signal Probe pins をクリックします。

Status カラムの表示で、ノードから Signal Probe ピンへの配線が正常に行われたかどうかを確認できます。

<table>
<thead>
<tr>
<th>ステータス</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected</td>
<td>配線は正常に行われました。</td>
</tr>
</tbody>
</table>
| Unconnected     | 配線は正常に行われませんでした。考えられる理由は次のとおりです。
                  | • ノードが IO セルまたは別のハード IP に属しているため、配線は不可能です。
                  | • ノード階層パスがデザインに存在しません。
                  | • ノードが Signal Tap: post-fitting ではありません。                   |

例-10: Compilation レポートで表示される Signal Probe ピンへの接続

また、Signal Probe 接続の情報は、Fitter レポートファイル（<project_name>.fit.rpt）で確認することができます。
例-11: top.fit.rptでのSignal Probeピンへの接続

+-----------------------------------------------------------------------------------------------
| ; Connections to Signal Probe pins ;                                                      |
+-----------------------------------------------------------------------------------------------

Signal Probe Pin Name: probey
Status: Connected
Attempted Connection: sprobe_me2
Actual Connection: sprobe_me2
Details:

Signal Probe Pin Name: wizard
Status: Connected
Attempted Connection: sprobe_me1
Actual Connection: sprobe_me1
Details:

+-----------------------------------------------------------------------------------------------

関連情報

• Signal Tap デバッグに使用できない信号 (31 ページ)
• テキストベースのレポートファイル
  インテル Quartus Prime プロ・エディション スクリプティング ユーザーガイド 内

4.2. Signal Probeを使用した迅速なデザイン検証 改訂履歴

<table>
<thead>
<tr>
<th>ドキュメント・バージョン</th>
<th>インテル Quartus Prime バージョン</th>
<th>変更内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.05.07</td>
<td>18.0.0</td>
<td>インテル Quartus Prime プロ・エディション開発ソフトウェアの初版</td>
</tr>
</tbody>
</table>
5. 外部ロジック・アナライザーを使用したインシステム・デバッグ

5.1. インテル Quartus Prime ロジック・アナライザー・インターフェイス

インテル Quartus Prime ロジック・アナライザー・インターフェイス（LAI）により、外部ロジック・アナライザーおよびインテルでサポートしているデバイスの最小限のI/Oピンを使用して、内部信号のビヘイビアを確認する一方で、インテルでサポートしているデバイス内でのデザインのフルスピードでの実行が可能です。

LAIでは、多数の内部デバイス信号を少数の出力ピンに接続します。これらの出力ピンの外部ロジック・アナライザーへの接続をデバッグを目的とすることで、インテル Quartus Prime LAI 内の内部信号は、グループ化された後、ユーザーによるコンフィグレーションが可能なマルチプレクサに分配され、インテルでサポートしているデバイス上で使用可能なI/Oピンに力されます。インテル Quartus Prime LAIでは、内部信号と出力ピンの間に 1 対1の関係を持たせるのではなく、多くの内部信号をより少数の出力ピンのマッピングが可能です。出力ピンにマッピングできる内部信号の正確な数は、インテル Quartus Prime LAIでのマルチプレクサの設定により異なります。

注意：「ロジック・アナライザー」という用語がこのドキュメント内で使われる場合、それに含まれるのは、ロジック・アナライザーとデジタルチャネルを備えたオシロスコープ（一般的にはミックス・シグナル・アナライザーまたはMSOと呼ばれる）の両方です。

LAIではハード・プロセッサーシステム（HPS）I/Oはサポートしていません。

関連情報
デバイス・サポートセンター

5.2. ロジック・アナライザーの選択

インテル Quartus Prime 開発ソフトウェアの次の2つの汎用オンチップ・デバッグ・ツールは、デザインから多数のRTL信号をデバッグするためのものです。

• Signal Tap Logic Analyzer
• 外部ロジック・アナライザー（インテル Quartus Primeでサポートしているデバイスの内部信号への接続にIntel LAIを使用します。）

表 18. Signal Tap Logic Analyzerとロジック・アナライザー・インターフェイスとの比較

<table>
<thead>
<tr>
<th>機能</th>
<th>説明</th>
<th>推奨ロジック・アナライザー</th>
</tr>
</thead>
<tbody>
<tr>
<td>サンプル深度</td>
<td>より広範囲のサンプル深度にアクセスできるのは、外部ロジック・アナライザーを使用した場合です。Signal Tapロジック・アナライザーの最大サンプル深度は128 Kbに設定されており、これにLAIが追加される。</td>
<td></td>
</tr>
</tbody>
</table>

continued...
機能 | 説明 | 推奨ロジック・アナライザー
--- | --- | ---
| | もはデバイスの制約です。外部ロジック・アナライザーを使用すると、デバイスの制約がないため、より広範囲のサンプル深度が得られます。 | LAI |
タイミング問題のデバッグ | 外部ロジック・アナライザーを使用すると、「タイミング」モードへのアクセスができます。このモードでは、データの組み合わせストリームのデバッグが可能です。 | LAI |
パフォーマンス | 配置配線に使用できる配線リソースに限りがあるのでは、Signal Tap Logic Analyzer をデザインで使用する場合です。外部ロジック・アナライザーの場合は、追加ロジックは最小限のため、配置配線におけるリソースの制限がなくなります。 | LAI |
トリガー機能 | Signal Tap Logic Analyzer のトリガー機能は、外部ロジック・アナライザーと互換性があります。 | LAI または Signal Tap |
出力ピンの使用 | Signal Tap Logic Analyzer の使用には、追加の出力ピンは不要です。外部ロジック・アナライザーの使用には、追加の出力ピンが必要です。 | Signal Tap |
取得スピード | Signal Tap Logic Analyzer では、200 MHz を超える速度でのデータ・アクイジョンが可能です。外部ロジック・アナライザーと同じ取得速度が達成可能です。シグナル・インテグリティに関する問題の考慮が必要です。 | Signal Tap |

関連情報
システム・デバッグ・ツールの概要 (7 ページ)

5.2.1. 必要なコンポーネント
LAI を使用して解析を実行するには、次のコンポーネントが必要です。
- インテル Quartus Prime 開発ソフトウェア・バージョン 15.1 以降
- テスト用デバイス
- 外部ロジック・アナライザー
- インテル FPGA 通信ケーブル
- インテルがサポートするデバイスと外部ロジック・アナライザーを接続するケーブル
図 72: LAI とハードウェアのセットアップ

注:
1. LAI のコンフィグレーションおよび制御に使用するコンピューターには、インテル Quartus Prime 開発ソフトウェアが JTAG ポート経由でロードされています。
2. LAI のコンフィグレーションおよび制御に使用するサードパーティー・ベンダーのロジック・アナライザーは、JTAG ポート経由です。サポートはベンダーにより異なります。
5.3. LAI の使用フロー

図 -73: LAI のワークフロー

注:
1. LAI のコンフィグレーションおよび制御に使用するコンピューターには、インテル Quartus Prime 開発ソフトウェアが JTAG ポート経由でロードされています。
2. LAI のコンフィグレーションおよび制御に使用するサードパーティー・ベンダーのロジック・アナライザーは、JTAG ポート経由です。サポートはベンダーにより異なります。

5.3.1. ロジック・アナライザー・インターフェイスのパラメーターの定義

Logic Analyzer Interface Editor を使用すると、LAI のパラメーターの定義ができます。
図-74: Logic Analyzer Interface Editor

- Tools > Logic Analyzer Interface Editor の順でクリックします。　

図 -74: Logic Analyzer Interface Editor

- Setup View リストで Core Parameters を選択します。　
- LAI インスタンスのパラメーターを指定します。

関連情報　
LAI コアのパラメーター (115 ページ)

5.3.2. 使用可能な I/O ピンへの LAI ファイルピンのマッピング

LAI に対するピン位置の割り当てについては次を実行します。
1. Setup View リストで Pins を選択します。
図-75: LAIファイルピンのマッピング

2. Nameカラムの予約ピンの隣のLocationカラムをクリックします。
3. 選択したピンを右クリックし、Pin Plannerで検索します。

関連情報
Managing Device I/O Pins
インテルQuartus Primeプロ・エディションデザイン制約ユーザーガイド内

5.3.3. 内部信号のLAIバンクへのマッピング

バンク数の指定をCore Parameters設定ページで行った後、内部信号をLAIの各バンクに対して割り当ててください。
1. Setup Viewの矢印をクリックし、Bank nまたはAll Banksを選択します。
2. バンク接続をすべて表示するには、Setup ViewをクリックしてAll Banksを選択します。
3. バンクのアサインメントを作成する前に、NodeリストをクリックしてAdd Nodesを選択し、Node Finderを開きます。
4. 取得する信号を検索します。
5. Node Finderダイアログボックスからその信号をバンクSetup Viewにドラッグします。
   信号を追加する際、インクリメンタル配線されていないインスタンスの場合はSignal Tap: pre-synthesisを使用し、インクリメンタル配線されたインスタンスにはSignal Tap: post-fittingを使用します。
   アサインメントの作成をバンクのSetup Viewで続けると、Logical ViewペインのLAIの回路図にその変更内容が反映されます。
6. 各バンクに対するアサインメントの作成をSetup Viewで継続して行い、取得する内部信号をすべて追加します。

関連情報
Node Finder Command
Intel Quartus Prime Help内
5.3.4. インテル Quartus Prime プロジェクトのコンパイル

.lai ファイルを保存すると、ダイアログボックスが表示され、ロジック・アナライザー・インターフェイスのインスタンスをアクティブなプロジェクトに対してイネーブルするよう求められます。代わりに、プロジェクトで使用する .lai ファイルの定義を Global Project Settings ダイアログボックスですることも可能です。.lai ファイル名の指定後、プロジェクトをコンパイルします。

ロジック・アナライザー・インターフェイスがプロジェクトに適切にコンパイルされていることを確認するには、Compilation Report タブを開き、パーティションの「auto_fab_0」の下にネストされた Resource Utilization by Entity を選択します。LAI IP インスタンスは、内部モジュールの auto_fab_0 の下にネストされた Compilation Hierarchy Node カラムに表示されます。

図 -76: Compilation レポートの LAI インスタンス

5.3.5. インテルでサポートしているデバイスの LAI を使用したプログラミング

コンパイル完了後、LAI の使用前にインテルでサポートしているデバイスのコンフィグレーションが必要です。

LAI の使用は、JTAG チェーン内の複数のデバイスで可能です。また、JTAG チェーンは、LAI をサポートしていないデバイスや、インテル以外の JTAG 準拠のデバイスで構成されている場合もあります。インテルでサポートしている複数のデバイスで LAI を使用するには、.lai ファイルを作成し、インテルでサポートしているデバイスごとに .lai ファイルのコンフィグレーションを行います。

5.4. ランタイムのアクティブバンクの制御

インテルでサポートしているデバイスのプログラミング後、予約済みの .lai 出カビピンにマッピングするバンクが制御できます。マッピングするバンクを制御するには、Logical View の回路図でそのバンクを右クリックし、Connect Bank をクリックします。
図-77: バンクのコンフィグレーション

5.4.1. ロジック・アナライザーでのデータの取得

データの取得をロジック・アナライザー上で行うには、デバイスと外部ロジック・アナライザーとの間の接続を確立する必要があります。このプロセスの詳細およびデバッグヘッダーとロジック・アナライザー間の接続を確立する方法に関するガイドラインは、ご使用のロジック・アナライザーのドキュメントを参照してください。

5.5. LAI コアのパラメーター

次の表は、LAI ファイルコアのパラメーターの一覧です。

表 19. LAI ファイルコアのパラメーター

<table>
<thead>
<tr>
<th>パラメーター</th>
<th>範囲値</th>
<th>説明</th>
</tr>
</thead>
</table>
| Pin Count    | 1 - 255 | LAI 専用のピン数です。このピンは、ボード上のデバッグヘッダーに接続してください。
|              |        | デバイス内では、Compiler による各ピンのマッピングがユーザー設定可能な数の内部信号に対して行われます。 |
| Bank Count   | 1 - 255 | 各ピンにマッピングする内部信号の数です。
|              |        | 例えば、Bank Count が 8 であれば、各ピンに 8 つの内部信号を接続することを意味します。 |

Output/Capture Mode

アキュジョン/モードを指定します。これには次の 2 つのオプションがあります。

- **Combinational/Timing** - このアクション/モードでは、外部ロジック・アナライザーの内部クロックを使用して、データをサンプリングするタイミングを決定します。
  このアクション/モードでは、サンプル周波数を手動で決定して、システムをデバッグおよび検証する必要があります。このデータ・サンプリングがインテルでサポートしているデバイスとは同期であるためです。
  このモードが有効なのは、チャネル間のスキーなどを測定する場合です。サンプリング周波数と動作速度に関する詳細は、外部ロジック・アナライザーのデータシートを参照してください。

- **Registered/State** - この取得モードでは、サンプリングするタイミングの決定を、検証中のシステム上の信号から行います。そのため、このデータサンプリングは、インテルでサポートしているデバイスとは同期です。
  Registered/State モードでは、インテルでサポートしているデバイスの機能ビューリセットがデバイスの実行中にされます。このモードが有効なのは、デザインの機能を検証する場合です。

Clock

サンプルクロックを指定します。デザイン内の信号はいずれもサンプルクロックとして使用できますが、最良の結果を得るため、使用するクロックは動作周波数が十分早いものにして、アクション/データのサンプリングができるようにします。
### パラメーター

<table>
<thead>
<tr>
<th>項目</th>
<th>番値</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>注意</td>
<td></td>
<td>Clock パラメーターが使用可能なのかは、Output/Capture Mode が Registered State に設定されている場合のみです。</td>
</tr>
<tr>
<td>Power-Up State</td>
<td>15.1.0</td>
<td>LAI での使用を指定したピンのパワーアップ状態を指定します。トライステートをすべてのピンに対して選択するか、またはイネーブルする特定のバンクを選択します。</td>
</tr>
</tbody>
</table>

### 関連情報

ロジック・アナライザー・インターフェイスのパラメーターの定義 (111 ページ)

### 5.6. 外部ロジック・アナライザーを使用したインシステム・デバッグ

次の改訂履歴はこの章に適用されます。

<table>
<thead>
<tr>
<th>ドキュメント/バージョン</th>
<th>インテル Quartus Prime バージョン</th>
<th>変更内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.05.07</td>
<td>18.0.0</td>
<td>• LAI ファイリコアのパラメーターのリストをファイルコアのパラメーターのコンフィグレーションから LAI コアのパラメーターのトピックに移動し、リンクを追加しました。</td>
</tr>
<tr>
<td>2017.05.08</td>
<td>17.0.0</td>
<td>• インテル Quartus Prime プロジェクトのコンパイルを使用した変更が完了しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Compilation レポートの LAI インスタンスの図を更新しました。</td>
</tr>
<tr>
<td>2016.10.31</td>
<td>16.1.0</td>
<td>• インテルにブランド名を変更しました。</td>
</tr>
<tr>
<td>2015.11.02</td>
<td>15.1.0</td>
<td>• Quartus II のインスタンスを変更して インテル Quartus Prime にしました。</td>
</tr>
<tr>
<td>2014 年 6 月</td>
<td>14.0.0</td>
<td>• DITA 変換</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• HPS I/O サポートに関する制限を追加しました。</td>
</tr>
<tr>
<td>2012 年 6 月</td>
<td>12.0.0</td>
<td>• サーベイリンクを削除しました。</td>
</tr>
<tr>
<td>2011 年 11 月</td>
<td>10.1.1</td>
<td>• ドキュメントのテンプレートを変更しました。</td>
</tr>
<tr>
<td>2010 年 12 月</td>
<td>10.1.0</td>
<td>• 軽微な編集上の更新を行いました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ドキュメントのテンプレートを更新しました。</td>
</tr>
<tr>
<td>2010 年 8 月</td>
<td>10.0.1</td>
<td>• リンクを修正しました。</td>
</tr>
<tr>
<td>2010 年 7 月</td>
<td>10.0.0</td>
<td>• Intel Quartus Prime Help へのリンクを作成しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 編集上の更新を行いました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 参考資料の項を削除しました。</td>
</tr>
<tr>
<td>2009 年 11 月</td>
<td>9.1.0</td>
<td>• 古いデバイスへの参照を削除しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 編集上の更新を行いました。</td>
</tr>
<tr>
<td>2009 年 3 月</td>
<td>9.0.0</td>
<td>• 編集上の軽微な更新を行いました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 8.1 バージョンから段 15-4, 15-5, および 15-11 を削除しました。</td>
</tr>
<tr>
<td>2008 年 11 月</td>
<td>8.1.0</td>
<td>• ページサイズを 8-1/2 x 11 に変更しました。内容に変更はありません。</td>
</tr>
<tr>
<td>2008 年 5 月</td>
<td>8.0.0</td>
<td>• ベージ 15-3 のデバイスサポートのリストを更新しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 整全体に参照資料のリンクを追加しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 「参考資料」の項を追加しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 第 5 章 インシステム・デバッグへの参照を追加しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 編集上の軽微な更新を行いました。</td>
</tr>
</tbody>
</table>
関連情報
ドキュメント・アーカイブ
以前のバージョンの インテル Quartus Prime ハンドブックは、文献アーカイブを検索してください。
6. メモリーおよび定数のインシステム修正

Intel In-System Memory Content Editor (ISMCE) を使用すると、メモリーおよび定数の表示と更新をランタイムに JTAG インターフェースを介して行うことができます。FPGA 内のメモリーコンテンツの変更の検証がデザインの実行中にできるので、問題の特定、テスト、および解決が可能となります。

メモリーと定数からデータを読み出す機能は、問題の原因を特定するのに役立ちます。また、書き込み機能により、機能的な問題を回避することができます。これには、予期されるデータの書き込みをします。

In-System Memory Content Editor と Signal Tap ロジック・アナライザーを併用すると、デザインの表示およびデバッグがハードウェア・ラボで可能になります。

関連情報
- システム・デバッグ・ツールの概要 (7 ページ)
- Signal Tap Logic Analyzer を使用したデザインのデバッグ (23 ページ)

6.1. ISMCE をサポートする IP コア

インテル Arria 10 および インテル Stratix 10 デバイスファミリーでは、RAM: 1 PORT および ROM: 1 PORT IP コアで ISMCE が使用可能です。

注意: ISMCE ソールを使用するデザインが、古いデバイスから インテル Stratix 10 デバイスに移行したものである場合は、altsyncram インテル FPGA IP のインスタンスを altera_syncram インテル FPGA IP に置き換えます。

関連情報
- インテル Stratix 10 エンベドド・メモリー IP コアのリファレンス
- インテル Stratix 10 エンベドド・メモリー・ユーザーガイド
- エンベドド・メモリー IP コアについて
  エンベドド・メモリー (RAM : 1 ポート, RAM : 2 ポート, ROM : 1 ポート, および ROM : 2 ポート) ユーザーガイド
- Intel FPGA IP Cores/LPM
  Intel Quartus Prime Help 内

6.2. In-System Memory Content Editor を使用したデバッグフロー

In-System Memory Content Editor を使用してデザインをデバッグするには、次を実行します。
1. ランタイムにアクセスするメモリーと定数を確認します。
2. デザイン内でランタイム修正でなければならないメモリーまたは定数を指定します。
3. フルコンパイルを実行します。
4. デバイスをプログラムします。
5. In-System Memory Content Editor を起動します。
   InSystem Memory Content Editor では、ランタイムのコンフィグレーションが可能なメモリーおよび定数のすべてのインスタンスを取得するために、JTAG チェーンをスキャンし、クエリーの送信を JTAG Chain Configuration ペインで選択したデバイスに対して行います。
6. メモリーまたは定数の値を修正して、結果を確認します。
   例えば、メモリー内のパリティービットに誤りがある場合は、In-System Memory Content Editor を使用して正しいパリティービット値を RAM に書き込み、システムの機能を継続させます。デザインのエラー処理機能をチェックするには、誤ったパリティービット値を RAM に意図的に書き込みます。

6.3. デザイン内インスタンスのランタイム修正のイーネーブル

メモリーあるいは定数のインスタンスのランタイム修正を可能にするには、次を実行します。
1. Parameter Editor でインスタンスを開きます。
2. Parameter Editor で Allow In-System Memory Content Editor to capture and update content independently of the system clock をオンにします。
3. デザインを再コンパイルします。
   メモリーまたは定数がランタイム修正可能になるように指定すると、インテル Quartus Prime 開発ソフトウェアでは、デフォルトの実装を変更し、ランタイム修正を可能にします。このとき、デザイン機能の変更は必要ありません。これには、次を実行します。
   • シングルポート RAM をデュアルポート RAM に変換します。
   • ロジックを追加して、メモリー衝突を回避し、読み出し/書き込みコヒーレンシーを維持します。対象となるデバイスファミリーは、トゥルー・デュアルポート RAM をサポートしていないインテル Stratix 10 などです。

6.4. In-System Memory Content Editor を使用したデバイスのプログラミング

コンパイル後、FPGA でデザインをプログラムしてください。JTAG Chain Configuration ペインを使用して、デバイスのプログラミングを In-System Memory Content Editor 内から行います。
6.5. メモリー・インスタンスの ISMCE へのロード

リコンフィグレーション可能なメモリー・インスタンスの内容を表示するには、次を実行します。

1. インテル Quartus Prime 開発ソフトウェアで、Tools > In-System Memory Content Editor の順でクリックします。

2. JTAG Chain Configuration ペインで、Scan Chain をクリックします。

In-System Memory Content Editor では、デバイスへのクエリー送信を JTAG Chain Configuration ペインで行い、ランタイムのコンフィグレーションが可能なメモリーおよび定数のすべてのインスタンスを取得します。

Instance Manager ペインには、ランタイム修正可能なメモリーおよび定数のすべてのインスタンスが一覧表示されます。Hex Editor ペインには、各メモリーまたは定数のインスタンスが表示されます。Hex Editor ペインのメモリー内容の表示は、デバイスが読み込まれるまで赤いクエスチョン・マークになっています。

図 -78: JTAG Chain スキャン後の Hex Editor

3. インスタンス 1 つを Instance manager でクリックし、をクリックして、そのインスタンスの内容をロードします。

Hex Editor にそのインスタンスの内容が表示されます。
6.6. メモリー内のロケーションのモニタリング

ISMCE を使用すると、メモリー領域内の情報のモニタリングができます。例えば、カウンターの増分や所定ワードの変更を決定できます。NIOS プロセッサーに接続されているメモリーの場合、ソフトウェアによるメモリーの主要領域の使用方法を観察できます。
6. メモリーおよび定数のインシステム修正

UG-20139 | 2018.09.24

図-79: 手動による内容編集後の Hex Editor

インスタンスの内容を編集するには、値を Hex Editor ペインに直接入力します。

6.7. Hex Editor を使用したメモリー内容の編集

インスタンスの内容を編集するには、値を Hex Editor ペインに直接入力します。

注意: (インテル Stratix 10 のみ) ISMCE ロジックで読み出し/書き込み操作が実行可能なのは、デザインロジックがアイドル状態の場合のみです。デザインロジックによって読み込み操作が実行可能であれば、アドレス変更操作の試みがされると、メモリー内容が更新され、ISMCE 操作がタイムアウトします。エラーメッセージによって、In-System Memory Content Editor のインスタンスに接続されているメモリーが使用中で、メモリー内容が更新されていないことが分かります。

関連情報

- キャプチャーしたデータの表示、解析、および使用 (77 ページ)
- Read Information from In-System Memory Commands (Processing Menu) Intel Quartus Prime Help 内
- Stop In-System Memory Analysis Command (Processing Menu) Intel Quartus Prime Help 内
Hex Editor ペインで黒色表示された内容が意味するのは、読み出された値が前回の同期と同じということです。

1. 内容をペインに入力します。Hex Editor の青色表示部分は、変更内容でデバイスに同期されていないものです。

図 -80: 手動による内容編集後の Hex Editor

2. をクリックして、内容をデバイスに同期させます。

注意: ( インテル Stratix 10 のみ) ISMCE ロジックで読み出し/書き込み操作が実行可能とは、デザインロジックがアイドル状態の場合のみです。デザインロジックによって書き込み操作またはアドレス変更操作の試みがされると、デザインロジックが優先され、ISMCE 操作がタイムアウトします。エラーメッセージによって、In-System Memory Content Editor のインスタンスに接続されているメモリーが使用中であることが分かり、デザインロジックによるメモリーへのアクセス要求前に正常に行われた書き込み数がレポートされます。

関連情報
• Custom Fill Dialog Box
  Intel Quartus Prime Help 内
• Write Information to In-System Memory Commands (Processing Menu)
  Intel Quartus Prime Help 内
• Go To Dialog Box
  Intel Quartus Prime Help 内
• Select Range Dialog Box
  Intel Quartus Prime Help 内
6.8. メモリーファイルのインポートおよびエクスポート

In-System Memory Content Editorを使用すると、実行時に変更可能なメモリーのデータのインポートおよびエクスポートができます。データファイルからインポートすることで、すべてのメモリーイメージの迅速なロードが可能になります。データファイルにエクスポートすることで、メモリー内容を保存して後で使用することが可能になります。

ファイルのインポートおよびエクスポートは、hexまたはmif形式で行います。

1. ファイルをインポートするには、Edit > Import Data from File...の順にクリックし、インポートするファイルを選択します。

ファイルに互換性がない場合、予期せぬデータがHex Editorに表示されます。

2. メモリー内容をファイルにエクスポートするには、Edit > Export Data to File...をクリックし、名前を指定します。

関連情報
• Import Data
  Intel Quartus Prime Help 内
• Export Data
  Intel Quartus Prime Help 内
• Hexadecimal (Intel-Format) File (.hex) Definition
  Intel Quartus Prime Help 内
• Memory Initialization File (.mif) Definition
  Intel Quartus Prime Help 内

6.9. 複数のデバイスへのアクセス

複数のデバイスがインシステムのコンフィグレーション可能なメモリーまたは定数を持ち、JTAGチェーン内に存在する場合は、複数のIn-System Memory Content EditorをインテルQuartus Prime開発ソフトウェア内で起動し、それぞれのデバイスのメモリーと定数にアクセスします。In-System Memory Content Editorの各ウィンドウでは、単一のデバイスメモリーと定数へのアクセスが可能です。

6.10. スクリプティング・サポート

インテルQuartus Prime開発ソフトウェアを使用すると、メモリーと定数のランタイムでの変更が、スクリプト化されたフローで実行可能です。

メモリーおよび定数のインスタンスのランタイム修正をイネーブルするには、HDLコードを使用します。さらに、In-System Memory Content Editorでは、メモリー内容の読み出しと書き込みのサポートにinsystem_memory_editパッケージのTclコマンドを使用します。

関連情報
• Tcl Scripting
  インテルQuartus Prime プロ・エディション スクリプティング ユーザーガイド 内
• Command Line Scripting
  インテルQuartus Prime プロ・エディション スクリプティング ユーザーガイド 内
6.10.1. insystem_memory_edit Tcl パッケージ

::quartus::insystem_memory_edit Tcl パッケージに含まれている Tcl 関数では、インテル FPGA デバイス内のメモリー内容の読み出しと変更の実行に、In-System Memory Content Editor を使用します。quartus_stp および quartus_stp_tcl コマンドライン実行ファイルは、このパッケージをデフォルトでロードします。

::quartus::insystem_memory_edit に関する最新情報については、インテル Quartus Prime Help を参照してください。

関連情報
::quartus::insystem_memory_edit
Intel Quartus Prime Help 内

6.10.1.1. insystem_memory_edit パッケージに関する情報の入手

insystem_memory_edit パッケージについての情報は、コマンドラインから直接入手することができます。

• パッケージに関する一般的な情報を入手するには、次のとおり入力します。
  quartus_stp --tcl_eval help -pkg insystem_memory_edit

• パッケージのコマンドに関する情報を入手するには、次のとおり入力します。
  quartus_stp --tcl_eval help -cmd <command_name>

6.11. メモリーおよび定数のインシステム修正 改訂履歴

次の改訂履歴はこの章に適用されます。

<table>
<thead>
<tr>
<th>ドキュメント/バージョン</th>
<th>インテル Quartus Prime バージョン</th>
<th>変更内容</th>
</tr>
</thead>
</table>
| 2018.05.07             | 18.0.0                           | • インテル Stratix 10 デバイスファミリーのサポートを追加しました。
|                        |                                  | • 古い例を削除しました。 |
| 2016.10.31             | 16.1.0                           | • インテルにブランド名を変更しました。 |
| 2015.11.02             | 15.1.0                           | • Quartus II のインスタンスを変更して、インテル Quartus Prime にしました。 |
| 2014年 6月             | 14.0.0                           | • DITA 変換しました。
|                        |                                  | • メン・ソール系への参照を削除し、IP コアに置き換えました。 |
| 2012年 6月             | 12.0.0                           | • サーベイリンクを削除しました。 |
| 2011年 11月            | 10.0.3                           | • デンプレートを更新しました。 |
| 2010年 12月            | 10.0.2                           | • 新しい文書テンプレートに変更しました。内容は変更はありません。 |
| 2010年 8月             | 10.0.1                           | • リンクを修正しました。 |
| 2010年 7月             | 10.0.0                           | • インテル Quartus Prime Help へのリンクを追加しました。
|                        |                                  | • 参考資料の項を削除しました。 |
| 2009年 11月            | 9.1.0                            | • APEX デバイスへの参照を削除しました。
|                        |                                  | • 形式を変更しました。 |

continued...
ドキュメント・バージョン | インテル Quartus Prime バージョン | 変更内容
--- | --- | ---
2009年3月 | 9.0.0 | 内容には変更ありません。
2008年11月 | 8.1.0 | ページサイズを8-1/2 x 11に変更しました。内容に変更はありません。
2008年5月 | 8.0.0 | • インテル Quartus Prime ハンドブック Volume 3のセクションV.インシステム・デバッグ (16-1ページ)への参照を追加しました。
• Mercury デバイスは現在「成熟」デバイスと見なされているため、Mercury デバイスへの参照を削除しました。
• 章全体に参照資料へのリンクを追加しました。
• 軽微な編集上の更新を行いました。

関連情報
ドキュメント・アーカイブ
以前のバージョンのインテル Quartus Prime ハンドブックは、文書アーカイブを検索してください。
7. In-System Sources and Probes を使用したデザインのデバッグ

Signal Tap Logic Analyzer および Signal Probe を使用すると、ランタイム中の内部ロジック信号の読み出しあるいは「タップ」をログックデザインのデバッグ手法として実行できます。

従来のデバッグ手法では、多くの場合、外部のパターン・ジェネレーターを使用してロジックを実行し、ロジック・アナライザーを使用してランタイムの出力波形を調べます。

内部信号の駆動をデザイン内で手動で行う場合、デバッグサイクルがより効率的となるため、次のアクションが実行可能です。

- 強制的なトリガー条件の発生を Signal Tap Logic Analyzer でセットアップします。
- 簡単なテストベクトルを作成してデザインを実行します。このとき外部試験装置は不要です。
- ランタイム・コントロール信号の動的制御を JTAG チェーンを使用して行います。

インテル Quartus Prime 開発ソフトウェアの In-System Sources and Probes Editor により、検証ツールのポートフォリオが拡張され、内部信号が簡単に制御でき、完全な動的デバッグ環境が得られます。Signal Tap Logic Analyzer または Signal Probe のいずれかと組み合わせることで、In-System Sources and Probes Editor によって得られる強力なデバッグ環境で、スティミュラスが生成され、ロジックデザインからの応答の要求が行われます。

さらに、Virtual JTAG IP コアおよび In-System Memory Content Editor の特性を利用すると、仮想入力をデザインへ駆動することができます。これに インテル Quartus Prime 開発ソフトウェアでは、様々なオンチップ・デバッグ・ツールを提供しています。

In-System Sources and Probes Editor の構成は、ALTSOURCE_PROBE IP コアと、ランタイム中に ALTSOURCE_PROBE IP コアのインスタンスを制御するインタフェースです。各 ALTSOURCE_PROBE IP コア・インスタンスによって、ソース出力ポートおよびプローブ入力ポートが提供されます。ソースポートでは選択した信号を駆動し、プローブポートでは選択した信号をサンプリングします。デザインをコンパイルすると、ALTSOURCE_PROBE IP コアでは、レジスタースキャンをセットアップして、ロジックデザインで選択したノードの駆動またはサンプリングを行います。ランタイム中、In-System Sources and Probes Editor では、JTAG 接続を使用し、ALTSOURCE_PROBE IP コア・インスタンスとの間でデータをシフトします。次の図で示しているのは、In-System Sources and Probes Editor を構成するコンポーネントのブロック図です。
図 -81: In-System Sources and Probes Editor のブロック図

ALTSOURCE_PROBE IP コアによって、デザイン内に計装された JTAG コントローラーとレジスターとの間の詳細なトランザクションが非表示になります。これによって得られる基本的なビルディング・ブロックでデザインを刺激しプロービングします。さらに、In-System Sources and Probes Editor では、選択したロジックノードにシングルサイクル・サンプルおよびシングルサイクル・ライトを提供しています。この機能を使用して、仮想スティミュラスの簡単な入力や、インストルメント化されたノードの現在の値のキャプチャーができます。In-System Sources and Probes Editor によってデザイン内のロジックノードへのアクセスができるようになるため、下位レベルのコンポーネントの入力のトグルがデバッグ過程で可能です。Signal Tap Logic Analyzer と組み合わせて使用すると、トリガー条件を強制して、問題点の切り分けやデバッグプロセスの短縮化に役立つことができます。

In-System Sources and Probes Editor を使用すると、デザイン内のコントロール信号を仮想スティミュラスとして簡単に実装できます。この機能が特に役立つのは、デザインのプロトタイプを作成する場合で、それには次のような動作があります。

- 仮想プッシュボタンの作成
- デザインとインターフェイスで接続する仮想フロントパネルの作成
- 外部センサーデータのエミュレート
- オンザフライでのランタイム定数のモニターおよび変更

In-System Sources and Probes Editor では、すべての ALTSOURCE_PROBE IP コア・インスタンスとインターフェイスで接続する Tcl コマンドをサポートしており、オートメーションのレベルを向上させます。
関連情報
System Debugging Tools
インテル Quartus Prime ソフトウェア・オンチップ・デバッグ・ツール・スイートで使用可能なすべてのツールの概要および比較

7.1. ハードウェアおよびソフトウェア要件
次のコンポーネントには、In-System Sources and Probes Editor を使用する必要があります。
• インテル Quartus Prime 開発ソフトウェア
または
• インテル Quartus Prime ライト・エディション
• ダウンロード・ケーブル (USB-BlasterTM ダウンロード・ケーブルまたは ByteBlasterTM ケーブル)
• テスト対象のデバイスへの JTAG 接続を備えたインテル FPGA 開発キットまたはユーザー・デザイン・ボード

In-System Sources and Probes Editor では、次のデバイスファミリーをサポートしています。
• Arria®シリーズ
• Stratix®シリーズ
• Cyclone®シリーズ
• MAX®シリーズ

7.2. In-System Sources and Probes Editor を使用したデザインフロー

In-System Sources and Probes Editor では、RTL フローをサポートしています。In-System Sources and Probes Editor で表示される信号は、In-System Sources and Probes IP コアのインスタンスに接続されます。

デザインのコンパイル後、In-System Sources and Probes Editor ペイン、または Tcl インタフェイスから各インスタンスの制御ができます。
図 -82: In-System Sources and Probes Editor を使用した FPGA のデザインフロー

7.2.1. In-System Sources and Probes IP コアのインスタンス化

In-System Sources and Probes IP コアをデザインでインスタンス化するには、次を実行します。
1. IP Catalog (Tools > IP Catalog) で In-System Sources and Probes と入力します。
2. In-System Sources and Probes をダブルクリックし、パラメター・エディターを開きます。
3. IP バリエーション名を指定します。
4. IP バリエーションのパラメーターを指定します。
7. In-System Sources and Probes を使用したデザインのデバッグ

UG-20139 | 2018.09.24

IP コアでは各ソースで最大 512 ビットをサポートしています。デザインに含めることができるこの IP コアのインスタンスは最大 128 個です。

5. **Generate** あるいは **Finish** をクリックして、仕様に適合する IP コアの合成ファイルおよびシミュレーション・ファイルを生成します。

6. 生成されたテンプレートを使用して、In-System Sources and Probes IP コアをデザインでインスタンス化します。

**注意**: In-System Sources and Probes Editor では、シミュレーションをサポートしていません。In-System Sources and Probes IP は、シミュレーション・ネットリストの作成前に削除します。

### 7.2.2. In-System Sources and Probes IP コアのパラメーター

テンプレートを使用して、バリエーション・ファイルをデザインでインスタンス化します。

#### 表 20. In-System Sources and Probes IP のポート情報

<table>
<thead>
<tr>
<th>ポート名</th>
<th>要/不要</th>
<th>入力/出力</th>
<th>注記</th>
</tr>
</thead>
<tbody>
<tr>
<td>probe[ ]</td>
<td>不要</td>
<td>入力</td>
<td>デザインからの出力です。</td>
</tr>
<tr>
<td>source_clk</td>
<td>不要</td>
<td>入力</td>
<td>Source Data は、このクロックに同期して書き込まれます。この入力が必要なのは、Source Clock をパラメーター・エディターの Advanced Options ボックスでオンにしている場合です。</td>
</tr>
<tr>
<td>source_ena</td>
<td>不要</td>
<td>入力</td>
<td>source_clk へのクロックイネーブル信号です。この入力が必要なのは、パラメーターア・エディターの Advanced Options ボックスで指定されている場合です。</td>
</tr>
<tr>
<td>source[ ]</td>
<td>不要</td>
<td>出力</td>
<td>ユーザーデザインへの入力の駆動に使用します。</td>
</tr>
</tbody>
</table>

最大 128 個の In-System Sources and Probes IP コアのインスタンスをデザインに含めることができるのは、デザイン用に使用可能なソースがある場合です。IP コアの各インスタンスでは、信号ごとに 1 対のレジスターを、IP コアで最も幅広いポートに対して使用します。これ以外にも、固定されたオーバーヘッド・ロジックがいくつかあり、IP コア・インスタンスと JTAG コントローラー間での通信に対応します。また、送信元ポートごとに該当のレジスターを追加で指定して同期に使用することもできます。

### 7.3. デザインのコンパイル

In-System Sources and Probes IP コアを含むデザインをケンパイルすると、In-System Sources and Probes IP コアおよび SLD Hub Controller IP コアとコンパイル階層に自動的に追加されます。このような IP コアによって、JTAG コントローラーと組み合わせられたロジックの間に通信が提供されます。

デザインへの接続数を修正するには、In-System Sources and Probes IP コアを編集します。修正するデザインインスタンスをパラメーター・エディターで開くには、Project Navigator のインスタンスをダブルクリックします。その後、HDL ソースファイル内の接続を修正することができます。変更を加えた後は、デザインの再コンパイルが必要です。

### 7.4. In-System Sources and Probes Editor の実行

In-System Sources and Probes Editor を使用すると、デザイン内のすべての ALTSOURCE_PROBE IP コアインスタンスが制御できます。このエディターで表示可能なのは、デザイン内の ALTSOURCE_PROBE IP コアのラインタイム制御可能なインスタンスすべてです。また、このエディターのプッシュボタン・インターフェイスではすべてのソースノードを観察し、ログ機能ではプローブおよびソースデータを格納します。
7. In-System Sources and Probes Editor を実行するには、

- Tools メニューから In-System Sources and Probes Editor をクリックします。

7.4.1. In-System Sources and Probes Editor の GUI

In-System Sources and Probes Editor には 3 つのペインがあります。

- **JTAG Chain Configuration** - プ로그ラミング・ハードウェア、デバイス、およびファイル設定の指定ができます。これを使用して、In-System Sources and Probes Editor でデバイスからのデータのプログラミングや取得を行います。
- **Instance Manager** - デザインのコンパイル時に生成されたインスタンスに関する情報が表示され、In-System Sources and Probes Editor で取得したデータの制御ができます。
- **In-System In-System Sources and Probes Editor** - 選択したインスタンスから読み出された全データが記録され、デバイスに書き込まれるソースデータの修正が可能になります。

In-System Sources and Probes Editor を使用する場合は、インテル Quartus Prime 開発ソフトウェア・プロジェクトを開く必要はありません。In-System Sources and Probes Editor では、ALTSOURCE_PROBE IP コアのすべてのインスタンスを取得するために JTAG Chain Configuration ペインで選択したデバイスにクエリーを送信し、JTAG チェーンをスキャンします。また、以前に保存したコンフィグレーションを使用して、In-System Sources and Probes を実行することもできます。

各 In-System Sources and Probes Editor ペインでは、単一のデバイス内の ALTSOURCE_PROBE IP コア・インスタンスへのアクセスが可能です。JTAG チェーン内の IP コア・インスタンスを含むデバイスが複数ある場合は、複数の In-System Sources and Probes Editor ペインを開き、各デバイスで IP コア・インスタンスにアクセスします。

7.4.2. JTAG チェーン・コンフィグレーションを使用したデバイスのプログラミング

プロジェクトのコンパイル後、FPGA のコンフィグレーションを行ってください。これは、In-System Sources and Probes Editor の使用前に行う必要があります。

デバイスをコンフィグレーションして、In-System Sources and Probes Editor が使用できるようにするには、次の手順を実行します。

1. In-System Sources and Probes Editor を開きます。
2. JTAG Chain Configuration ペインで Hardware にカーソルを移動し、ハードウェア・コンフィグレーション・デバイスを選択します。ハードウェアのコンフィグレーションを指示するプロンプトが表示されるので、Setup をクリックします。
3. Device リストから、デザインをダウンロードする FPGA デバイスを選択します（デバイスが自動検索される場合もあります。）ターゲットデバイスの検索には Scan Chain をクリックします。
4. JTAG Chain Configuration ペインで、In-System Sources and Probes インスタンスを含む SRAM Object File (.sof) をクリックして開きます。 (.sof は自動検出される場合もあります。)
5. Program Device をクリックして、ターゲットデバイスをプログラムします。

7.4.3. Instance Manager

Instance Manager ペインでは、デザイン内のすべての ALTSOURCE_PROBE インスタンスが一覧表示され、データ・アクイジョンのコンフィグレーションができます。
Instance Manager ペインには、次のボタンとサブペインが含まれます。

- **Read Probe Data** - 選択したインスタンス内のプローブデータをサンプリングし、そのプローブデータの表示を In-System Sources and Probes Editor ペインで行います。

- **Continuously Read Probe Data** - 選択したインスタンスのプローブデータを継続的にサンプリングし、そのプローブデータの表示を In-System Sources and Probes Editor ペインで行います。サンプルレートは、Probe read interval 設定で修正可能です。

- **Stop Continuously Reading Probe Data** - 選択したインスタンスのプローブの継続的なサンプリングを取り消します。

- **Read Source Data** - 選択したインスタンス内のソースのデータを読み出します。

- **Probe Read Interval** - デザイン内のすべての In-System Sources and Probe のサンプル・インターバルを表示します。サンプル・インターバルの修正には、Manual をクリックします。

- **Event Log** - In-System Sources and Probes Editor ペインに表示されるイベントログを制御します。

- **Write Source Data** - 手動または継続的なシステムへのデータの書き込みができます。

各エントリー以外にも、Instance Manager ペインにはインスタンスのステータスが表示されます。インスタンスのステータスは、Not running Offloading data, Updating data, Unexpected JTAG communication error のいずれかになります。

### 7.4.4. In-System Sources and Probes Editor ペイン

In-System Sources and Probes Editor ペインでは、デザイン内のすべてのソースとプローブからのデータを表示することができます。

データの編成は、インスタンスのインデックス番号に従います。エディターを使用すると、信号の管理が簡単になり、信号名の変更や信号のバスへのグループ化ができるようになります。In-System Sources and Probes ノードから収集されたすべてのデータは、イベントログに記録され、こうしたデータはタイミング図として表示することができます。

#### 7.4.4.1. プローブデータの読み出し

データの読み出しには、Instance Manager ペインで ALTSOURCE_PROBE インスタンスを選択し、Read Probe Data をクリックします。

この操作により、プローブデータの単一のサンプルが生成され、In-System Sources and Probes Editor ペインで選択したインデックスのデータカラムが更新されます。Instance Manager ペインで Save data to event log オプションをオンにすると、データをイベントログに保存することができます。

データのサンプリングをプローブ・インスタンスから継続的に行う場合は、Instance Manager ペインで読み出すインスタンスを選択し、Continuously read probe data をクリックします。読み出し中は、アクティブ・インスタンスのステータス表示は Unloading となります。継続的読み出しは、複数のインスタンスからすることもできます。

読み出しデータへのアクセスは、Instance Manager ペインのショートカット・メニューからもできます。
プローブの読み出し間隔を調整するには、Instance Manager ペインで Probe read interval サブペインの Manual オプションをオンにし、サンプルレートの指定を Manual オプションの隣のテキストフィールドで行います。最大サンプルレートは、使用するコンピューターの設定によって異なります。実際のサンプルレートの表示は、Current interval ボックスにされます。イベントログ・ウィンドウのバッファーサイズの調整は、Maximum Size ボックス内ですることができます。

7.4.4.2. データの書き込み

ALTSOURCE_PROBE インスタンスへ書き込むソースデータを修正するには、変更する信号のネームフィールドをクリックします。信号のバスの場合は、データフィールドをダブルクリックし、ALTSOURCE_PROBE インスタンスへドライブアウトする値を入力します。In-System Sources and Probes Editor では、変更後のソースデータの値を一時バッファに格納します。

ALTSOURCE_PROBE インスタンスに書き込まれない編集済みの値は、赤色で表示されます。ALTSOURCE_PROBE インスタンスを更新するには、インスタンスを Instance Manager ペインでハイライトし、Write source data をクリックします。Write source data 機能の使用は、Instance Manager ペインのショートカット・メニューからも可能です。

In-System Sources and Probes Editor のオプションを使用すると、各 ALTSOURCE_PROBE インスタンスを継続的に更新することができます。更新を継続的に行うことで、ソース・データ・バッファーエインスタンスに写込まれます。ALTSOURCE_PROBE インスタンスを継続的に更新するには、Write source data フィールドを Manually から Continuously に変更します。

7.4.4.3. データの編成

In-System Sources and Probes Editor ペインでは、信号のバスへのグループ化や、データバッファーエインスタンスへの表示オプションの変更ができます。

信号のグループを生成するには、グループ化を行うノード名を選択し、右クリックで Group を選択します。表示形式の修正は、Bus Display Format および Bus Bit オーダー・ショートカット・メニューでできます。

In-System Sources and Probes Editor ペインでは、信号名の変更が可能ですが、信号名を変更するには、信号をダブルクリックし、新しい信号名を入力します。

イベントログには最新サンプルのレコードが含まれています。バッファーサイズは最大 128k サンプルまで調整可能です。各サンプルのタイムスタンプは記録され、データサンプルの上にカーソルを移動させるとアクティブ・インスタンスのイベントログの上部に表示されます。

変更内容や記録データは、Sources and Probes File （.spf）に保存します。変更を保存するには、File メニューで Save をクリックします。このファイルには、信号グループへの全修正内容および現在のデータ・イベント・ログが含まれます。

7.5. In-System Sources and Probes Editor の Tcl インターフェイス

In-System Sources and Probes Editor では、オートメーションのサポートのため、この章で記載されているプロシージャーを Tcl コマンド形式でサポートしています。In-System Sources and Probes Editor 用の Tcl パッケージは、quartus_stp 実行時にデフォルトで含まれています。

In-System Sources and Probes Editor 用の Tcl インターフェイスで提供されている強力なプラットフォームでは、デザインのデバッグに役立つことができます。Tcl インターフェイスが特に有用なのは、デザインのデバッグ時に複数のコントロール入力のセットのトグルが必要な場合です。複数のコマンドを Tcl スクリプトと組み合わせて、カスタム・コマンド・セットを定義します。
### In-System Sources and ProbesのTclコマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_insystem_source_probe</td>
<td>-device_name &lt;device name&gt; -hardware_name &lt;hardware name&gt;</td>
<td>デバイスへのハンドルを指定のハードウェアで開きます。このコマンドの呼び出しは、トランザクションの開始前にいます。</td>
</tr>
<tr>
<td>get_insystem_source_probe_instance_info</td>
<td>-device_name &lt;device name&gt; -hardware_name &lt;hardware name&gt;</td>
<td>デザイン内のすべてのALTSOURCE_PROBEインスタンスのリストを返します。返される各レコードのフォーマットは次のとおりです。{&lt;instance Index&gt;,&lt;source width&gt;,&lt;probe width&gt;,&lt;instance name&gt;}</td>
</tr>
<tr>
<td>read_probe_data</td>
<td>-instance_index &lt;instance_index&gt; -value_in_hex (オプション)</td>
<td>プローブの現在の値を取得します。各プローブのステータスを特定する文字列が返されます。MSBは一番左側のビットになります。</td>
</tr>
<tr>
<td>read_source_data</td>
<td>-instance_index &lt;instance_index&gt; -value_in_hex (オプション)</td>
<td>ソースの現在の値を取得します。各ソースのステータスを特定する文字列が返されます。MSBは一番左側のビットになります。</td>
</tr>
<tr>
<td>write_source_data</td>
<td>-instance_index &lt;instance_index&gt; -value &lt;value&gt; -value_in_hex (オプション)</td>
<td>ソースの値を設定します。バイナリ文字列がソースポートへ送られます。MSBは一番左側のビットになります。</td>
</tr>
<tr>
<td>end_insystem_source_probe</td>
<td>なし</td>
<td>JTAGチェーンをリリースします。このコマンドの発行は、全てのトランザクションの終了後にいます。</td>
</tr>
</tbody>
</table>

次の例のTclスクリプトの抜粋では、プロシージャーによってデザインのALTSOURCE_PROBEインスタンスが制御されます。これについて下の図に示します。デザイン内に含まれるDCFIFOのALTSOURCE_PROBEインスタンスでは、DCFIFOへの読み書きを行います。一連のコントロール・マルチプレクサがデザインに追加され、DCFIFOへのデータフロー制御を、入力ピンとALTSOURCE_PROBEインスタンスとの間で行います。パルス・ジェネレーターが読み出しおよび書き込み要求のコントロール・ラインに追加され、単一サンプルの読み書きが確保されます。ALTSOURCE_PROBEインスタンスを次の例のスクリプトで使用すると、FIFOのコンテンツが可視化されます。これには、単一サンプルの読み書き動作を実行し、フルおよび空のステータスフラッグをレポートします。

Tclスクリプトをデバッグ時に使用して、デザイン内でFIFOを空にするか、プリロードします。例えば、この機能を使用してFIFOをプリロードし、Signal Tap Logic Analyzerで設定したトリガー条件に適合させます。
図 -83:  

Tcl スクリプトによって制御される DCFIFO のデザイン例

```
# Setup USB hardware - assumes only USB Blaster is installed and
# an FPGA is the only device in the JTAG chain
set usb [lindex [get_hardware_names] 0]
set device_name [lindex [get_device_names -hardware_name $usb] 0]
## write procedure : argument value is integer
proc write {value} {
    global device_name usb
    variable full
    start_insystem_source_probe -device_name $device_name -hardware_name $usb
    #read full flag
    set full [read_probe_data -instance_index 0]
    if {($full == 1) [end_insystem_source Probe} return "Write Buffer Full"
    ##toggle select line, drive value onto port, toggle enable
    ##bits 7:0 of instance 0 is S_data[7:0]; bit 8 = S_write_req;
    ##bit 9 = Source_write_sel
    ##int2bits is custom procedure that returns a bitstring from an integer
    ##argument
    write_source_data -instance_index 0 -value {[int2bits [expr 0x200 | $value]]}
    write_source_data -instance_index 0 -value {[int2bits [expr 0x300 | $value]]}
    ##clear transaction
    write_source_data -instance_index 0 -value 0
    end_insystem_source_probe
}
proc read {} {
    global device_name usb
    variable empty
    start_insystem_source_probe -device_name $device_name -hardware_name $usb
    ##read empty flag : probe port[7:0] reads FIFO output; bit 8 reads empty_flag
    set empty [read_probe_data -instance_index 1]
    if {([regexp {1........} $empty]} {end_insystem_source_probe
    return "FIFO empty" }
    ## toggle select line for read transaction
```

7. In-System Sources and Probes を使用したデザインのデバッグ

UG-20139 | 2018.09.24
関連情報

- Tcl スクリプティング
- Intel Settings File Manual
- コマンドライン・スクリプティング

7.6. デザイン例：ダイナミック PLL リコンフィグレーション

In-System Sources and Probes Editor を使用すると、仮想フロントパネルをデザインのプロトタイプ段階で作成するのに役立ちます。比較的シンプルで高性能なデザインの作成が短期間で可能です。次に示す PLL リコンフィグレーションの例で示しているのは、In-System Sources and Probes Editor を使用して GUI を提供し、Stratix PLL のダイナミック・リコンフィグレーションを行う方法です。

Stratix PLL では、ランタイムの PLL 係数の動的更新が可能です。Stratix デバイスに内蔵された各エンハンスト PLL に含まれるレジスターチェーンでは、プリスケール・カウンター (値 m と n)、出力分周カウンター、および遅延カウンターの変更ができます。さらに、ALTPPLL_RECONFIG IP コアに備えられた簡単なインターフェイスでは、このレジスターチェーンへのアクセスができます。また、ALTPPLL_RECONFIG IP コアのキャッシュには、変更可能な PLL パラメーターがすべて含まれています。PLL パラメーターをすべてキャッシュ内で更新した後、ALTPPLL_RECONFIG IP コアでは PLL レジスターチェーンを駆動し、変更されたパラメーターで PLL を更新します。次の図で示す Stratix のエンハンスト PLL には、リコンフィグレーション可能な係数が含まれています。
図 -84: リコンフィグレーション可能な係数を持つ Stratix のエンハンスト PLL

次のデザイン例では、ALTSOURCE_PROBE インスタンスを使用して、ALTPLL_RECONFIG IP コアキャッシュの PLL パラメーターを更新します。ALTPLL_RECONFIG IP コアでは、Stratix FPGA のエンハンスト PLL に接続し、リコンフィグレーション可能な PLL 係数を含むレジスターチェーンを駆動します。このデザイン例では Tcl/Tk スクリプトを使用して GUI を生成します。その GUI 上で、エンハンスト PLL の新しい m および n の値が入力できます。Tcl スクリプトでは、m と n の値を GUI から抽出し、その値を ALTSOURCE_PROBE インスタンスへシフトアウトして ALTPLL_RECONFIG IP コアキャッシュ内に更新し、また、リコンフィグレーション信号のアサートを ALTPLL_RECONFIG IP コアで行います。ALTPLL_RECONFIG IP コアのリコンフィグレーション信号では、レジスターチェーン・トランザクションを開始し、リコンフィグレーション可能な PLL 係数すべてを更新します。

図 -85: ダイナミック PLL リコンフィグレーション・デザイン例のブロック図
このデザイン例の作成には、Nios II Development KitのStratix Editionを使用しています。
sourceprobe_DE_dynamic_pll.zipファイルには、このデザイン例の実行に必要なファイルが全て含まれており、次のファイルも含まれます。
• Readme.txt - テキストファイル。デザイン例に含まれるファイルを記述し、下図に示すTk GUIの実行方法を説明します。
• Interactive_Reconfig.qar - このデザイン例のためにアーカイブされたインテルQuartus Primeプロジェクトです。

図 -86: TkおよびIn-System Sources and Probes Tcl Packageを使用して作成したインタラクティブPLLリコンフィグレーションGUI

関連情報
On-Chip Debugging Design Examples
In-System Sources and Probes Editorの例がダウンロードできます。

7.7. In-System Sources and Probesを使用したデザインのデバッグの改訂履歴

次の改訂履歴はこの章に適用されます。

<table>
<thead>
<tr>
<th>ドキュメント・バージョン</th>
<th>インテル Quartus Prime バージョン</th>
<th>変更内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.05.07</td>
<td>18.0.0</td>
<td>IP CatalogでのIn-System Sources and Probesの検索についての詳細を追加しました。</td>
</tr>
<tr>
<td>2016.10.31</td>
<td>16.1.0</td>
<td>インテルにブランド名を変更しました。</td>
</tr>
<tr>
<td>2015.11.02</td>
<td>15.1.0</td>
<td>Quartus IIのインスタンスを変更してインテルQuartus Primeになりました。</td>
</tr>
<tr>
<td>2014年6月</td>
<td>14.0.0</td>
<td>表示形式を更新しました。</td>
</tr>
<tr>
<td>2012年6月</td>
<td>12.0.0</td>
<td>サーベイリンクを削除しました。</td>
</tr>
<tr>
<td>2011年11月</td>
<td>10.1.1</td>
<td>テンプレートを更新しました。</td>
</tr>
<tr>
<td>2010年12月</td>
<td>10.1.0</td>
<td>軽微な訂正を行いました。ドキュメントのテンプレートを更新しました。</td>
</tr>
<tr>
<td>2010年7月</td>
<td>10.0.0</td>
<td>軽微な訂正を行いました。</td>
</tr>
<tr>
<td>2009年11月</td>
<td>9.1.0</td>
<td>古いデバイスへの参照を削除しました。</td>
</tr>
</tbody>
</table>

continued...
### 関連情報

ドキュメント・アーカイブ

以前のバージョンの インテル Quartus Prime ハンドブックは、文書アーカイブを検索してください。
8. System Consoleを使用したデザインの解析とデバッグ

8.1. System Consoleの概要

System Consoleを使用すると、デザインが可視化され、FPGA上のシステムレベルのデバッグをランタイムに実行できます。System Consoleでのテストの実行は、デバッグがイネーブルされたプラットフォーム・デザイナー（旧：Qsys）のインスタンス化IPコアで行われます。次のタスクの実行には、System ConsoleおよびSystem Consoleに搭載されたツールを使用します。

- 最終デザインおよび部分的に完成したデザインでのボードの立ち上げ
- インターネット・アクセスを使用したリモートデバッグの実行
- システム内の複数のデバイスでスクリプティングを使用したランタイム検証の自動化
- Transceiver Toolkitのポイント・アンド・クリック・コンフィグレーション・チューニングを使用したシリアルリンクのテスト
- External Memory Interface Toolkitを使用したメモリー・インターフェースのデバッグ
- デバッグ・プラットフォームへのデバッグIPの統合
- MATLAB/Simulinkを使用したシステム検証の実行

フィードバック

Intel Corporation.

*その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。
図 - 87: System Console のツール

(Tools) で表示するのは、System Console と相互作用するアプリケーションです。System Console API でサポートしているサービスでは、動作中にデザインにアクセスします。一部のサービスには特定のハードウェア要件があります。

注意: デバッグリンクを使用して、デバッグするターゲットにホストを接続してください。

関連情報
- Introduction to Intel® Memory Solution
  External Memory Interface Handbook Volume 1 内
- トランシーバー・リンクのデバッグ (204 ページ)
- Application Note 693: Remote Hardware Debugging over TCP/IP for Intel SoC
- Application Note 624: Debugging with System Console over TCP/IP
- White Paper 01208: Hardware in the Loop from the MATLAB/Simulink Environment
- System Console Online Training

8.2. System Console のデバッグフロー

System Console でデザインをデバッグするには、次の手順を実行してください。
1. プラットフォーム・デザイナー（旧: Qsys）システムに IP コアを追加します。
2. プラットフォーム・デザイナー（旧: Qsys）システムを生成します。
3. デザインをコンパイルします。
4. ボードを接続し、FPGA をプログラムします。
5. System Console を起動します。
6. System Console サービスを検索して開きます。
7. サービスを使用してデバッグ動作を実行します。
8. サービスを閉じます。

8.3. System Console と相互作用する IP コア

System Console は、ホスト・コンピューター上で動作し、デバッグ・エージェントを介して実行中のデザ
インと通信します。デバッグ・エージェントは一部の IP コアに組み込まれたソフトロジックです。これに
より、ホスト・コンピューターとのデバッグ通信が可能になります。

デバッグ IP コアのインスタンス化を、プラットフォーム・デザイナー (旧: Qsys) IP Catalog を使用して
実行します。一部の IP コアは、デバッグに向けたデフォルトでインポートされています。また、その他の IP
コアのインポートには、パラメーター・ディレクターのオプションを使用します。一部のデバッグ・エージェント
には、複数の目的があります。

エンベドッド・デバッグ機能を持つ IP をデザインで使用すると、デザインの大部分がアクセス可能になり
ます。デバッグ・エージェントを使用すると、メモリーへの読み出しと書き込みおよびホスト・コンピュータ
ーからのペリフェラル・レジスターの変更ができます。

実行中のデザイン内のデバッグ・エージェントに接続されたサービスは、必要に応じて開始および停
止できます。System Console では、デバッグ・エージェントとの通信プロトコルを決定します。通信プ
ロトコルでは、コマンドおよびデータの伝送に最適なボード接続を決定します。

Programmable SRAM Object File (．sof) では、System Console にチャネル通信情報を提供
します。デザイン展開中に System Console を インテル Quartus Prime 開発ソフトウェアまたはプラ
ットフォーム・デザイナー (旧: Qsys) で開くと、既存の．sof が自動検出され、検知された実行中の
デバイスにリンクされます。複雑なシステムでは、場合によってはデザインとデバイスとのリンクを手動で
行う必要があります。

関連情報
WP-0170 System-Level Debugging and Monitoring of FPGA Designs

8.3.1. デバッグ・エージェントによって提供されるサービス

適切なデバッグ・エージェントを選択することで、System Console のサービスでデバッグ・
エージェントの関連機能が使用できるようになります。

表 22. System Console の一般的なサービス

<table>
<thead>
<tr>
<th>サービス</th>
<th>機能</th>
<th>サービスを提供するデバッグ・エージェント</th>
</tr>
</thead>
</table>
| マスター | マスター・インターフェイスに接続されたメモリーマップ(Avalon-MM または AXI) スレーブにアクセスします。 | • デバッグ機能を搭載した Nios II  
• JTAG to Avalon Master Bridge  
• USB Debug Master |
| スレーブ | ホストから単一のスレーブへのアクセスが可能になります。ホストのメモリーマップ内のスレーブの位置情報は必要は
ありません。System Console のマスターにアクセス可能
なスレーブでは、いずれもこのサービスを提供できます。 | • デバッグ機能を搭載した Nios II  
• JTAG to Avalon Master Bridge  
• USB Debug Master |

continued...
<table>
<thead>
<tr>
<th>サービス</th>
<th>機能</th>
<th>サービスを提供するデバッグ・エージェント</th>
</tr>
</thead>
<tbody>
<tr>
<td>プロセッサー</td>
<td>開始、停止、またはステップを実行します。</td>
<td>デバッグ機能を持つ Nios II</td>
</tr>
<tr>
<td>JTAG UART</td>
<td>JTAG UART は、Avalon-MM スレーブデバイスです。System Console と組み合わせてバイトストリームの送受信に使用できます。</td>
<td>JTAG UART</td>
</tr>
</tbody>
</table>

注意: IP Catalog 内の次の IP コアでは、インテル Quartus Prime 開発ソフトウェアの現行バージョンでの VHDL シミュレーションの生成はサポートしていません。
- JTAG Debug Link
- JTAG Hub Controller System
- USB Debug Link

関連情報
- System Console の例とチュートリアル (198 ページ)
- System Console のコマンド (147 ページ)

8.4. System Console の起動

8.4.1. Nios II コマンドシェルからの System Console の起動

1. Windows の Start メニューで, All Programs > Intel > Nios II EDS <version> > Nios II<version> > Command Shell の順でクリックします。
2. system-console と入力します。
3. -- help と入力し、System Console のヘルプを開きます。
4. system-console --project_dir=<project directory> と入力し、.qsf ファイルまたは .sof ファイルが含まれているディレクトリを指定します。

8.4.2. スタンドアロンの System Console の起動

スタンドアロンの System Console は、インテル Quartus Prime 開発ソフトウェアの Programmer and Tools インストーラーの一部として、インテルのウェブサイトから入手できます。
1. Download Center ページに進み、Additional Software タブをクリックします。
2. Windows の Start メニューで, All Programs > Intel FPGA <version> > Programmer and Tools > System Console の順でクリックします。

関連情報
インテル ダウンロードセンター
8.4.3. プラットフォーム・デザイナーからの System Console の起動

Tools > System Console の順でクリックします。

8.4.4. インテル Quartus Prime からの System Console の起動

Tools > System Debugging Tools > System Console の順でクリックします。

8.4.5. 起動のカスタマイズ

System Console 環境のカスタマイズには次の実行を行います。

- コマンドを次のロケーションにある system_console_rc コンフィグレーション・ファイルに追加します。
  ```
  → <$HOME>/system_console/system_console_rc.tcl
  ```
  このロケーションにあるファイルは、ユーザーによるコンフィグレーションが可能なファイルで、それによって影響されるのはホーム・ディレクトリの所有者のみです。

- ユーザー独自のデザイン・スタートアップ・コンフィグレーション・ファイルを指定するには、コマンドライン引数の --rc_script=<path_to_script> を使用します。これを行うのは、Nios II コマンドシェルから System Console を起動する場合です。

  ```system_console_rc.tcl``` ファイルは、カスタムの rc_script.tcl ファイルと組み合わせて使用することができます。この場合、```system_console_rc.tcl``` ファイルでは System Console の動作を実行し、```rc_script.tcl``` ファイルではデバッグ動作を実行します。

System Console では起動時に、Tcl コマンドを上記のファイルで自動的に実行します。

```system_console_rc.tcl``` ファイルのコマンドが最初に実行され、```rc_script.tcl``` ファイルのコマンドがそれに続きます。

8.5. System Console の GUI

System Console の GUI を構成するメインウィンドウには、複数のペインがあります。System Console の GUI を使用すると、実行中のデザインとホスト・コンピューター上で通信できます。

- **System Explorer** - デザイン内の System Console 仮想ファイルシステムの階層を表示します。表示されるのは、ボード接続、デバイス、デザイン、スクリプトなどです。

- **Workspace** - 使用可能なツールキットを表示します。表示されるのは、ADC Toolkit、Transceiver Toolkit、Toolkit、GDB Server Control Panel、Bus Analyzer などです。Tools メニューをクリックしてアプリケーションを起動します。

- **Tcl Console** - このウィンドウでは、Tcl スクリプトを使用したデザインとの通信が可能です。例えば、スクリプトのソーシング、プロシージャーの作成や System Console API の使用などができます。

- **Messages** - 接続およびデバッグ動作に関連するステータス、警告、およびエラーメッセージを表示します。
図 -88: System Console の GUI

8.5.1. System Explorer ペイン

System Explorer ペインでは、接続されているデバッグ IP の仮想ファイルシステムすべてが表示され、次の情報が含まれています。

- **Devices** フォルダー - System Console に接続されているすべてのデバイスに関する情報を表示します。
- **Scripts** フォルダー - スクリプトを格納して、実行を容易にします。
- **Connections** フォルダー - インテル FPGA ダウンロード・ケーブルなどの System Console に表示されるボード接続に関する情報を表示します。複数の接続が可能です。
- **Designs** フォルダー - System Console に接続された インテル Quartus Prime デザインに関する情報を表示します。各デザインは、ロードされた .sof ファイルを表します。

Devices フォルダーに含まれているサブフォルダーは、System Console に接続された各デバイスに対するものです。各デバイスのサブフォルダーには、(link) フォルダー 1 つが含まれ、場合によっては、(files) フォルダー 1 つが含まれます。(link) フォルダーに表示されるデバッグ・エージェント（および他のハードウェア）は、System Console にアクセス可能です。(files) フォルダーに含まれる情報は、デバイス用に インテル Quartus Prime プロジェクトからロードされたデザインファイルに関するものです。
図 89: System Explorer ペイン
次の図で示すのは、Device フォルダー内の EP4SGX230 フォルダーです。このフォルダーには (link) フォルダーが含まれています。この (link) フォルダーに含まれている JTAG フォルダーでは、デバイスへのアクティブなデバッグ接続を記述します。例えば、JTAG、USB、イーサネットや、JTAG 接続を介して EP4SGX230 デバイスに接続されたエージェントなどです。

• コンテキスト・メニューを持つフォルダーには、コンテキスト・メニューのアイコンが表示されます。このフォルダーを右クリックすると、コンテキスト・メニューが表示されます。例えば、上の Connections フォルダーには、コンテキスト・メニューのアイコンが表示されています。

• メッセージを含むフォルダーには、メッセージアイコンが表示されます。これらのフォルダーにマウスを重ねるとメッセージが表示されます。例えば、上の例では Scripts フォルダーにはメッセージアイコンがあります。

• デバッグ・エージェントでは、ターゲットのクロックおよびリセット状態を感知すると、クロックステータスのアイコンを含む情報またはエラーメッセージを表示します。このアイコンの表示によって、クロックが実行中（情報、緑色）、停止中（エラー、赤色）、または実行中だがリセット中（エラー、赤色）のいずれかが分かります。例えば、図の trace_system_jtag_link.h2t フォルダーには、自走クロックが 1 つあります。

8.6. System Console のコマンド

コンソールコマンドを使用するとテストの実行が可能です。コンソールコマンドを使用して、パス別にサービスを識別し、接続の開閉を行います。サービスを識別する path は、大多数の System Console コマンドの最初の引数です。

サービスの接続を開始するには次のコマンドを実行します。
1. get_service_paths コマンドでパスを指定してサービスを特定します。
2. claim_service コマンドでそのサービスへの接続を開きます。
3. Tcl コマンドおよび System Console コマンドを使用して、接続されたデバイスをテストします。
4. close_service コマンドでそのサービスへの接続を閉じます。
注意: すべての Tcl コマンドに対して、<format>を最初の引数にします。

表 23. System Console のコマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_service_types</td>
<td>itchen日本語文字列</td>
<td>System Console で管理するサービスタイプのリストを返します。サービスタイプの例は、マスター、バイスター、プロセッサー、sid、jtag_debug、デバイス、およびデザインなどです。</td>
</tr>
<tr>
<td>get_service_paths</td>
<td>&lt;service-type&gt;、&lt;device&gt;、&lt;hpath&gt;、&lt;type&gt;</td>
<td>反されたサービスのフィルタリングができます。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>使用するサービスの部分のより細かい制御ができます。claim_service によって返される新しいパスは、そのサービスの使用を表し、それぞれの使用は独立しています。claim_service を複数回呼び出すと、返される値は毎回異なりますが、サービスへのアクセスは、それぞれの値が閉じられるまで可能です。</td>
</tr>
<tr>
<td>claim_service</td>
<td>&lt;service-type&gt;、&lt;service-path&gt;、&lt;claim-group&gt;、&lt;claims&gt;</td>
<td>指定バスで指定したサービスタイプを閉じます。</td>
</tr>
<tr>
<td>close_service</td>
<td>&lt;service-type&gt;、&lt;service-path&gt;</td>
<td>該当なし</td>
</tr>
<tr>
<td>is_service_open</td>
<td>&lt;service-type&gt;、&lt;service-type&gt;</td>
<td>1 が返されるのは、パスによって提供されるサービスタイプが開いている場合です。0 が返されるのは、サービスタイプが閉じている場合です。</td>
</tr>
<tr>
<td>get_services_to_add</td>
<td>該当なし</td>
<td>add_service コマンドを使用したインスタンス化が可能なすべてのサービスのリストを返します。</td>
</tr>
<tr>
<td>add_service</td>
<td>&lt;service-type&gt;、&lt;instance-name&gt;、optional-parameters</td>
<td>指定したサービスタイプのサービスを所定のインスタンス名で追加します。get_services_to_add を実行して、インスタンス化が可能なサービスのリストを取得します。このコマンドでは、サービスが追加されたパスが返されます。help add_service &lt;service-type&gt; を実行すると、そのサービスタイプについての特定のヘルプが表示されます。例えば、そのサービスに必要なパラメータなどです。</td>
</tr>
<tr>
<td>add_service gdbserver</td>
<td>&lt;Processor Service&gt;、&lt;port number&gt;</td>
<td>gdbserver をインスタンス化します。</td>
</tr>
<tr>
<td>add_service tcp</td>
<td>&lt;instance name&gt;、&lt;ip_addr&gt;、&lt;port_number&gt;</td>
<td>イーサネットを介してデバッグリンクを提供する TCP/IP ポートへの接続ができます。詳細に関しては、AN693 (Remote Hardware Debugging over TCP/IP for Intel FPGA SoC) を参照してください。</td>
</tr>
<tr>
<td>add_service transceiver_channel_rx</td>
<td>&lt;data_pattern_checker&gt;、&lt;path&gt;、&lt;transceiver path&gt;</td>
<td>Transceiver Toolkit のレシーバーチャネルをインスタンス化します。</td>
</tr>
</tbody>
</table>

continued
<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_service</td>
<td>&lt;transceiver_channel_address&gt;</td>
<td>Transceiver Toolkit のトランスミッター・チャネルをインスタンス化します。</td>
</tr>
<tr>
<td>transceiver_channel_tx</td>
<td>&lt;data_pattern_generator &gt;</td>
<td></td>
</tr>
<tr>
<td>get_version</td>
<td>該当なし</td>
<td>現在の System Console のバージョンとビルド番号を返します。</td>
</tr>
<tr>
<td>get_claimed_services</td>
<td>&lt;claim&gt;</td>
<td>所定の claim グループの場合、要求されたサービスのリストを返します。返されるリストはパスとサービスタイプのペアで構成されています。各ペアが 1 つの要求されたサービスです。</td>
</tr>
<tr>
<td>refresh_connections</td>
<td>該当なし</td>
<td>使用可能なハードウェアをスキャンし、変更があった場合は使用可能なサービスパスを更新します。</td>
</tr>
<tr>
<td>send_message</td>
<td>&lt;level&gt;</td>
<td>所定のレベルのメッセージをメッセージウィンドウに送信します。使用可能なレベルは、info、warning、error、および debugです。</td>
</tr>
<tr>
<td></td>
<td>&lt;message&gt;</td>
<td></td>
</tr>
</tbody>
</table>

関連情報
Remote Hardware Debugging over TCP/IP for SoC Devices

8.7. コマンドライン・モードでの System Console の実行

System Console をコマンドライン・モードで実行し、インタラクティブに作業するか、Tcl スクリプトを実行します。System Console では、出力をコンソールウィンドウに表示します。

- --cli - System Console をコマンドライン・モードで実行します。
- --project_dir=<project dir> - System Console をハードウェア・プロジェクトのロケーションに送ります。GUI モードでも動作します。
- --script=<your script>.tcl - System Console に指示して Tcl スクリプトを実行します。
- --help - 使用可能なすべてのコマンドをリストします。--help <command name> と入力すると、そのコマンドの構文と引数が得られます。

System Console によるコマンドの完了は、コマンドの先頭文字を入力し、Tab キーを押下した場合です。
8.8. System Console サービス

インテルのSystem Consoleサービスでは、FPGAでインスタンス化されたハードウェア・モジュールへのアクセスが提供されています。サービスは、提供されるデバッグアクセスのタイプによって異なります。

8.8.1. 使用可能なサービスの検索

System Consoleでは、仮想ファイルシステムを使用し、使用可能なサービスを整理します。これは、Linuxシステムの/dev locationに似ています。ボード接続、デバイスタイプ、およびIP名はすべて、サービスパスの一部です。サービスのインスタンスは、ファイルシステム内の固有のサービスパスによって参照されます。特定のサービスへのサービスパスを取得するには、get_service_paths <service-type>を使用します。

例-12: サービスパスの検索

```bash
# We are interested in master services.
set service_type "master"

# Get all the paths as a list.
set master_service_paths [get_service_paths $service_type]

# We are interested in the first service in the list.
set master_index 0

# The path of the first master.
set master_path [lindex $master_service_paths $master_index]

# Or condense the above statements into one statement:
set master_path [lindex [get_service_paths master] 0]
```

System Consoleのコマンドにサービスパスが必要なのは、アクセスするサービス・インスタンスを識別するためです。異なるコンポーネント用のパスの変更は、System Consoleの各実行間とバージョン間で可能です。get_service_pathsコマンドを使用してサービスパスを取得します。

サービスパスの文字列の値は、ツールのリリースによって変わります。marker_node_infoコマンドを使用して、パスから情報を取得します。

System Consoleでは、ほとんどのサービスを起動時に自動検出し、すべてのJTAGおよびUSBベースのサービス・インスタンスを自動スキャンし、サービスパスを取得しますが、TCP/IPなどの一部のサービスは自動検出しません。add_serviceコマンドを使用して、System Consoleにこれらのサービスを通知します。

例-13: Marker_node_info

```bash
marker_node_info コマンドを使用して、検出されたサービスに関する情報を取得します。

set slave_path [get_service_paths -type altera_avalon_uart.slave slave]
array set uart_info [marker_node_info $slave_path]
echo $uart_info(full_hpath)
```

8.8.2. サービスの開始と終了

特定のサービス・インスタンスへのサービスパスを取得した後は、そのサービスにアクセスし、使用することができます。
claim_service コマンドの指示によって、System Console による特定のサービス・インスタンスの使用が開始します。追加の引数は使用せずに、専用のサービス・インスタンスを要求します。

例-14: サービスの開始

```bash
set service_type "master"
set claim_path [claim_service $service_type $master_path mylib]; #Claims service.
```

追加の引数を claim_service コマンドに渡し、System Console に指示して、サービス・インスタンスの特定の部分へのアクセスを開始できます。例えば、マスターサービスを使用してメモリーにアクセスする場合、claim_service を使用して 0x0 と 0x1000 との間のアドレス空間へのみアクセスします。System Console では、他のユーザーによる他のメモリー範囲へのアクセスを許可しますが、要求されたメモリー範囲へのアクセスは拒否します。claim_service コマンドでは、新たに作成したサービスパスを返し、ユーザーはそれを使用して要求したリソースへのアクセスにします。

サービスにアクセスできるのは、そのサービスを開いた後です。サービス・インスタンスへのアクセスを終了したら、close_service コマンドを使用して System Console に指示し、そのリソースを他のユーザーが使用できるようにします。

例-15: サービスの終了

```bash
close_service master $claim_path; #Closes the service.
```

8.8.3. SLD Service

SLD Service では、SLD ノードの命令およびデータレジスターに値をシフトし、前の値をキャプチャーします。SLD ノードと通信する場合は、まず開始したサービスのノードへの排他的アクセスを取得します。

例-16: SLD Service

```bash
set timeout_in_ms 1000
set lock_failed [sld_lock $sld_service_path $timeout_in_ms]
```

このコードでは、選択した SLD ノードのロックを試みます。SLD ノードが既にロックされている場合、sld_lock は指定されたタイムアウトまで待機します。プロセッサーによってゼロ以外の値が返されていることを処理前に確認します。次のとおり、命令レジスターを設定し、以前のものを次のとおりキャプチャーします。

```bash
if {$lock_failed} {
    return
}
set instr 7
set delay_us 1000
set capture [sld_access_ir $sld_service_path $instr $delay_us]
```

1000 マイクロ秒の遅延によって、次の SLD コマンドの実行が少なくとも 1000 マイクロ秒後に実行されることが保証されます。データ・レジスター・アクセスの動作も同様です。

```bash
set data_bit_length 32
set delay_us 1000
set data_bytes [list 0xEF 0xBE 0xAD 0xDE]
set capture [sld_access_dr $sld_service_path $data_bit_length $delay_us $data_bytes]
```
シフトカウントの指定はビットですが、データ内容の指定はバイトのリストです。キャプチャーの戻り値もバイトのリストです。SLD サービス終了後は、必ず SLD ノードのロックを解除します。

```bash
sld_unlock $sld_service_path
```

関連情報
アルテラ仮想 JTAG (altera_virtual_jtag) IP コアのユーザーガイド

### 8.8.3.1. SLD コマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>sld_access_ir</td>
<td>&lt;claim-path&gt; &lt;ir-value&gt; &lt;delay&gt; (単位: µs)</td>
<td>指定したノードの命令レジスターに命令値をシフトします。命令前の値を返します。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;delay&gt; パラメーターがゼロ以外の場合、JTAG クロックはアクセス後にこの時間だけ一時停止します。</td>
</tr>
<tr>
<td>sld_access_dr</td>
<td>&lt;service-path&gt; &lt;size_in_bits&gt; &lt;delay-in-µs&gt; &lt;list_of_byte_values&gt;</td>
<td>バイト値は、SLD ノードのデータレジスターにシフトします。サイズは指定されたビット数までです。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;delay&gt; パラメーターがゼロ以外の場合、JTAG クロックは、アクセス後に少なくともこの時間だけ一時停止します。</td>
</tr>
<tr>
<td>sld_lock</td>
<td>&lt;service-path&gt; &lt;timeout-in-milliseconds&gt;</td>
<td>SLD チェーンをロックして、排他的アクセスを保証します。成功すると 0 を返します。SLD チェーンが既に別のユーザーによってロックされている場合は、Tcl エラーを送出する前に ＜timeout＞ms を試みます。catch コマンドが使用可能なのかは、このエラーを処理する場合です。</td>
</tr>
<tr>
<td>sld_unlock</td>
<td>&lt;service-path&gt;</td>
<td>SLD チェーンのロックを解除します。</td>
</tr>
</tbody>
</table>

### 8.8.4. In-System Sources and Probes

In-System Sources and Probes (ISSP) サービスで提供されている `altsource_probe` IP コアへのスクリプト可能アクセスは、インテル Quartus Prime 開発ソフトウェアの In-System Sources and Probes Editor の使用方法と類似しています。

#### 例-17: ISSP サービス

ISSP サービスの使用前に、In-System Sources and Probes Editor でデザインが動作することを確認してください。System Console で、ISSP インスタンスのサービスを開きます。

```bash
set issp_index 0
set issp [lindex [get_service_paths issp] 0]
set claimed_issp [claim_service issp $issp mylib]
```

この特定の ISSP インスタンスに関する情報を表示します。

```bash
array set instance_info [issp_get_instance_info $claimed_issp]
set source_width $instance_info(source_width)
set probe_width $instance_info(probe_width)
```

インテル Quartus Prime 開発ソフトウェアでは、プローブ幅と等しい長さの単一のビット文字列としてプローブデータを読み出します。

```bash
set all_probe_data [issp_read_probe_data $claimed_issp]
```
一例として、次のプロシージャーを定義して、個々のプローブラインのデータを抽出することができます。

```tcl
proc get_probe_line_data {all_probe_data index} {
    set line_data [expr { ($all_probe_data >> $index) & 1 }]
    return $line_data
}
```

```tcl
set initial_all_probe_data [issp_read_probe_data $claim_issp]
set initial_line_0 [get_probe_line_data $initial_all_probe_data 0]
set initial_line_5 [get_probe_line_data $initial_all_probe_data 5]
# ...
set final_all_probe_data [issp_read_probe_data $claimed_issp]
set final_line_0 [get_probe_line_data $final_all_probe_data 0]
```

同様に、インテル Quartus Prime 開発ソフトウェアでは、ソース幅に等しい長さの単一のビット文字列としてソースデータを書き込みます。

```tcl
set source_data 0xDEADBEEF
issp_write_source_data $claimed_issp $source_data
```

現在設定されているソースデータも取得可能です。

```tcl
set current_source_data [issp_read_source_data $claimed_issp]
```

一例として、32 ビット幅に対するデータの反転には、次を実行します。

```tcl
set current_source_data [issp_read_source_data $claimed_issp]
set inverted_source_data [expr { $current_source_data ^ 0xFFFFFFFF }]
issp_write_source_data $claimed_issp $inverted_source_data
```

8.8.4.1. In-System Sources and Probes 用コマンド

注意: ISSP クレームの有効値は、read_only, normal, exclusive などです。

**表 25. In-System Sources and Probes 用コマンド**

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>issp_get_instance_info</td>
<td>&lt;service-path&gt;</td>
<td>In-System Sources and Probes インスタンスのコンフィグレーションのリストを返します。次はその例です。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>instance_index</td>
</tr>
<tr>
<td></td>
<td></td>
<td>instance_name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>source_width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>probe_width</td>
</tr>
<tr>
<td>issp_read_probe_data</td>
<td>&lt;service-path&gt;</td>
<td>プローブ入力の現在の値を取得します。プローブポートの値を表す 16 進文字列が返されます。</td>
</tr>
<tr>
<td>issp_read_source_data</td>
<td>&lt;service-path&gt;</td>
<td>ソース出力ポートの現在の値を取得します。ソースポートの値を表す 16 進文字列が返されます。</td>
</tr>
<tr>
<td>issp_write_source_data</td>
<td>&lt;service-path&gt;</td>
<td>ソース出力ポートの値を設定します。この値は、System Console Tcl インタープリターでサポートしている 16 進文字列または 10 進値のいずれかとなります。</td>
</tr>
</tbody>
</table>

8.8.5. モニターサービス

モニターサービスは、マスターサービスの上に構築され、Avalon-MM スレーブの読み出しを通常のインターバルで可能です。このサービスは、完全にソフトウェアベースです。モニターサービスでは追加のソフトウェアは不要です。このサービスでは、インターバル読み出しを実行するロジックが効率化され、読み出しにマスターサービスを手動で実行するよりも優れたパフォーマンスが得られます。
例-18: モニターサービス

1. ポーリングするマスターとメモリーのアドレス範囲を決定します。

```bash
set master_index 0
set master [lindex [get_service_paths master] $master_index]
set address 0x2000
set bytes_to_read 100
set read_interval_ms 100
```

最初のマスターで 100 バイトを読み出します。最初はアドレス 0x2000 からで、ミリ秒毎に行います。

2. モニターサービスを開きます。

```bash
set monitor [lindex [get_service_paths monitor] 0]
set claimed_monitor [claim_service monitor $monitor mylib]
```

モニターサービスによってマスターサービスが自動的に開きます。

3. モニターサービスを使用して、アドレスの範囲とタイム・インターバルを登録します。

```bash
monitor_add_range $claimed_monitor $master $address $bytes_to_read
monitor_set_interval $claimed_monitor $read_interval_ms
```

4. 範囲をさらに追加し、各インターバルでの結果を定義します。

```bash
global monitor_data_buffer
set monitor_data_buffer [list]
```

5. データを収集し、そのデータにグローバル変数を追加します。

```bash
proc store_data {monitor master address bytes_to_read} {
    global monitor_data_buffer
    # monitor_read_data returns the range of data polled from the running
design as a list
    # (in this example, a 100-element list).
    set data [monitor_read_data $claimed_monitor $master $address
    $bytes_to_read]
    # Append the list as a single element in the monitor_data_buffer global
    list.
    lappend monitor_data_buffer $data
}
```

注: このプロシージャーにかかる時間がインターバル時間よりも長い場合、モニターサービスで
は、場合によってはプロシージャーへの次の 1 つ以上の呼び出しをスキップする必要がありま
す。この場合、monitor_read_data では、最後にポーリングされたデータを返します。

6. このコールバックの登録には、開いているモニターサービスを使用します。

```bash
set callback [list store_data $claimed_monitor $master $address
$bytes_to_read]
monitor_set_callback $claimed_monitor $callback
```

7. このコールバック変数を使用した呼び出しの実行は、モニターでインターバルを完了するときです。

```bash
monitor_set_enabled $claimed_monitor 1
```

モニターでは、指定された範囲をデバイスから直ちに読み出し、指定されたインターバルでコールバッ
クを呼び出します。この検証に、monitor_data_buffer の内容を確認します。モニターを
オフにするには、上記のコマンドの 1 の代わりに 0 を使用します。

8.8.5.1. Monitor コマンド

Monitor コマンドを使用すると、多数の Avalon-MM スレーブメモリーのロケーションの読み出しが通
常のインターバルで可能です。
通常の負荷では、モニターサービスでは各インターバル後にデータを読み出し、その後コールバックを呼び出します。読み出した値がタイミングの影響を受けやすい場合、monitor_get_read_interval コマンドを使用して、データが読み出されたインターバルの正確な時間を読み出します。

負荷が大きい場合、またはコールバックの実行に時間がかかる場合は、モニターサービスではいくつかのコールバックをスキップします。読み出すレジスターに（リセットからのイベント総数を読み出すなど）の副作用がない場合、コールバックをスキップしてもコードに影響はありません。

読み出すレジスターに（最後の読み出しさらイベントの数を返すなどの）副作用がある場合、読み出されたデータへのアクセスが必要ですが、そのデータのコールバックはスキップされています。

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>monitor_add_range</td>
<td>&lt;service-path&gt; &lt;target-path&gt; &lt;address&gt; &lt;size&gt;</td>
<td>連続するメモリアドレスをモニターしたメモリリストに追加します。</td>
</tr>
<tr>
<td>monitor_get_all_read_intervals</td>
<td>&lt;service-path&gt; &lt;target-path&gt; &lt;address&gt; &lt;size&gt;</td>
<td>monitor_read_all_data によって返されるデータ内の 2 つの読み出し間のインターバルのリストをミリ秒単位で返します。</td>
</tr>
<tr>
<td>monitor_get_interval</td>
<td>&lt;service-path&gt;</td>
<td>ポーリングアクションの頻度を指定する現在の間隔セッティングを返します。</td>
</tr>
<tr>
<td>monitor_get_missing_event_count</td>
<td>&lt;service-path&gt;</td>
<td>最後の Tcl コールバックでの評価中に欠落したコールバック・イベント数を返します。</td>
</tr>
<tr>
<td>monitor_get_read_interval</td>
<td>&lt;service-path&gt; &lt;target-path&gt; &lt;address&gt; &lt;size&gt;</td>
<td>monitor_read_data によって返される最後の 2 つのデータ読み出し間に経過したミリ秒を返します。</td>
</tr>
<tr>
<td>monitor_read_all_data</td>
<td>&lt;service-path&gt; &lt;target-path&gt; &lt;address&gt; &lt;size&gt;</td>
<td>すべての最新値から読み出された 8 ビット値のリストを返します。この最新値は、最後の Tcl コールバック以降デバイス読み出されたものです。 monitor_add_range の範囲内でメモリ範囲を指定してください。</td>
</tr>
<tr>
<td>monitor_read_data</td>
<td>&lt;service-path&gt; &lt;target-path&gt; &lt;address&gt; &lt;size&gt;</td>
<td>すべての最新値から読み出された 8 ビット値のリストを返します。この最新値は、デバイスから読み出されたもので、monitor_read_all_data の範囲内でメモリー範囲を指定してください。</td>
</tr>
</tbody>
</table>

continued...
### 8.8.6. デバイスサービス

デバイスサービスでは、デバイスレベルのアクションをサポートします。

#### 例-19: プログラミング

Tcl スクリプトでデバイスサービスを使用すると、デバイスのプログラミングが実行可能になります。

```tcl
set device_index 0 ; #Device index for target
set device [lindex [get_service_paths device] $device_index]
set sof_path [file join project_path output_files project_name.sof]
device_download_sof $device $sof_path
```

プログラミングに必要なのは、デバイス・サービス・パスと .sof へのファイル・システム・パスです。マスターサービスなどの他のサービスがターゲットデバイスで開いていないことを確認してください。開いていないとコマンドは成功しません。その後、次の手順を実行して、デバイスにリンクされたデザインがプログラムされたものと同様であることを確認します。

```tcl
device_get_design $device
```

### 8.8.6.1. デバイスコマンド

デバイスコマンドを使用すると、ボード上のプログラマブル・ロジック・デバイスへのアクセスができます。このコマンドを使用する前に、ボード上のプログラマブル・ロジック・デバイスへのパスの確認を `get_service_paths` を使用して行ってください。

#### 表 27. デバイスコマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>device_download_sof</td>
<td><code>&lt;service_path&gt;</code> <code>&lt;sof-file-path&gt;</code></td>
<td>.sof をパスで指定したデバイスにロードします。</td>
</tr>
<tr>
<td>device_get_connections</td>
<td><code>&lt;service_path&gt;</code></td>
<td>指定したパスでデバイスに接続するすべての接続を返します。</td>
</tr>
<tr>
<td>device_get_design</td>
<td><code>&lt;device_path&gt;</code></td>
<td>このデバイスが現在リンクされているデザインを返します。</td>
</tr>
</tbody>
</table>

### 8.8.7. デザインサービス

デザイン・サービス・コマンドを使用し、インテル Quartus Prime デザイン情報を処理します。
例-20: ロード

System Console をインテル Quartus Prime 開発ソフトウェア、またはプラットフォーム・デザイナー (旧：Qsys) から開く場合、現在のプロジェクトのデバッグ情報が自動的にソースされるのは、.sof がビルドされている場合です。これ以外の状況では、手動でのロードが可能です。

```tcl
set sof_path [file join project_dir output_files project_name.sof]
set design [design_load $sof_path]
```

これで、System Console によってこの特定の .sof がロードされたことが認識されます。

例-21: リンク

.sof がロードされると、System Console では、接続されたデバイスにデザイン情報を自動リンクします。この結果のリンクは維持され、ユーザーの選択によって、同じ .sof を持つ同等デバイスのリンクの解除または再利用ができます。

手動でのリンクが可能です。

```tcl
set device_index 0; # Device index for our target
set device [lindex [get_service_paths device] $device_index]
design_link $design $device
```

手動でのリンクが失敗するのは、ターゲットデバイスとデザインサービスが適合しない場合です。

リンクの失敗は、ターゲットにプログラムされた .sof がデザインの .sof と異なる場合でも起きます。

### 8.8.7.1. デザイン・サービス・コマンド

デザイン・サービス・コマンドはシステムレベルでデザインをロードし、処理します。

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
</table>
| design_load                | `<quartus-project-path>`、`<sof-file-path>` または `<qpf-file-path>` | インテル Quartus Prime デザインのモデルを System Console へロードします。デザインパスを返します。例えば、インテル Quartus Prime Project File (.qpf) が c:/projects/loopback に存在する場合は、次のコマンドを入力します。design_load {c:\projects\loopback}\]
| design_link                | `<design-path>`、`<device-service-path>` | インテル Quartus Prime ロジックデザインと物理デバイスをリンクさせます。例えば、2c35_quartus_design と呼ばれるインテル Quartus Prime デザインは、2c35 デバイスに接続可能です。このリンクの作成後、System Console では、インテル Quartus Prime プロジェクトの論理サブモジュールと物理サブモジュールの間に適切な対応点を作成します。 |
| design_extract_debug_files | `<design-path>`、`<zip-file-name>` | デバッグファイルを .sof から zip ファイルに抽出します。このファイルを E メールで Intel FPGA Support に送ると解析に使用できます。デザインパスの [] を指定して、デバイスのリンク解除およびそのデバイスへの自動リンクのディスエーブルができます。 |
| design_get_warnings         | `<design-path>` | このデザインの警告リストを取得します。デザインが正常にロードされている場合は空のリストが返されます。 |
8.8.8. バイトストリーム・サービス

バイトストリーム・サービスによって、バイトのストリームを生成または消費するモジュールへのアクセスが可能になります。バイトストリーム・サービスを使用すると、Altera JTAG UARTやAvalon-ST JTAGインターフェイスなどのバイトストリーム・インターフェイスを提供するIPコアと直接通信できます。

例-22: バイトストリーム・サービス

次のコードでは、使用中のインターフェイス用のバイトストリーム・サービスを検索して開きます。

```
set bytestream_index 0
set bytestream [lindex [get_service_paths bytestream] $bytestream_index]
set claimed_bytestream [claim_service bytestream $bytestream mylib]
```

発信データをバイトのリストとして指定し、開いたサービスを介してそのデータを送信します。

```
set payload [list 1 2 3 4 5 6 7 8]
bytestream_send $claimed_bytestream $payload
```

受信データもまた、バイトのリストになります。

```
set incoming_data [list]
while {[llength $incoming_data] ==0} {
    set incoming_data [bytestream_receive $claimed_bytestream 8]
}
```

完了後はサービスを閉じます。

```
close_service bytestream $claimed_bytestream
```

8.8.8.1. バイトストリーム・コマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytestream_send</td>
<td>&lt;service-path&gt;</td>
<td>バイトのリストを、指定したバイトストリーム・サービスに送信します。Values引数は、送信するバイトのリストです。</td>
</tr>
<tr>
<td></td>
<td>&lt;values&gt;</td>
<td></td>
</tr>
<tr>
<td>bytestream_receive</td>
<td>&lt;service-path&gt;</td>
<td>指定したサービス受信キューにある現在使用可能なバイトのリストを、指定した制限まで返します。Length引数は、受信するバイトの最大数です。</td>
</tr>
<tr>
<td></td>
<td>&lt;length&gt;</td>
<td></td>
</tr>
</tbody>
</table>

8.8.9. JTAG Debug サービス

JTAG Debug サービスを使用すると、デザイン内のクロックとリセットの状態が確認できます。

次に、JTAG Debug デザインフローの例です。

1. 使用可能なJTAG Debugパスを識別します。

```
get_service_paths jtag_debug
```

2. JTAG Debugパスを選択します。

```
set jtag_debug_path [lindex [get_service_paths jtag_debug] 0]
```

3. JTAG Debugサービスパスを要求します。

```
set claim_jtag_path [claim_service jtag_debug$jtag_debug_path mylib]
```
4. JTAG Debug ビーを実行します。

```
jtag_debug.reset_system $claim_jtag_path
jtag_debug_loop $claim_jtag_path [list 1 2 3 4 5]
```

### 8.8.9.1 JTAG Debug コマンド

JTAG Debug コマンドは、デバイスに接続されている JTAG チェーンのデバッグに役立ちます。

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>jtag.debug.loop</td>
<td>&lt;service-path&gt;</td>
<td>指定したバイトのリストのループは、システムレベル・デバッグ (SLD)ノードの tdi および tdo のループバックを介して行います。バイト値のリストは、受信した順番で返されます。ここで、バイトのプロックはすべてのバイトの受信までです。バイト値は 0x (16 進数) の接頭部を持ち、スペースで区切られています。</td>
</tr>
<tr>
<td>jtag.debug.sample_clock</td>
<td>&lt;service-path&gt;</td>
<td>モジュールのシステム・インターフェイスを駆動するシステムクロックのクロック信号を返します。クロック値は非同期にサンプリングされるので、クロックを数回サンプリングして、クロックが確実に切り替わるようにしてください。</td>
</tr>
<tr>
<td>jtag.debug.sample_reset</td>
<td>&lt;service-path&gt;</td>
<td>Avalon-ST JTAG モジュールの reset_n 信号の値を返します。reset_n が Low (アサートされている状態) の場合、値は 0 です。reset_n が High (ディアサートされている状態) の場合、値は 1 です。</td>
</tr>
<tr>
<td>jtag.debug.sense_clock</td>
<td>&lt;service-path&gt;</td>
<td>システムクロック動作をモニタリングするディスティッキー・ビットを返します。このコマンドの最後の実行から一度でもクロックが切り替わった場合は 1 が返されます。それ以外の場合は 0 が返されます。ディスティッキー・ビットは、読み出し時に 0 にリセットされます。</td>
</tr>
<tr>
<td>jtag.debug.reset_system</td>
<td>&lt;service-path&gt;</td>
<td>リセット要求を指定されたサービスに対して発行します。デバイス内の接続によって、システムのどの部分がリセットされるかが決まります。</td>
</tr>
</tbody>
</table>

### 8.9 Toolkit の使用

Toolkit API を使用すると、カスタムツールを作成して、デザイン・デバッグ・データを視覚化して操作することができます。Toolkit API による視覚的ウィジェットは、ボタンやテキストフィールドといった形式で、ユーザーの入力を活用して、デバッグ・ロジックと通信します。 Toolkit API は、インテル Quartus Prime 開発ソフトウェアのバージョン 14.1 以降で使用可能です。 Toolkit API は Dashboard サービスの後継です。

Toolkit API を使用して作成したツールキットには、次のファイルが必要です。
- ツールキットを記述する XML ファイル (.toolkit) ファイル
- ツールキット GUI を実装する Tcl ファイル

### 8.9.1. ツールキット API への Dashboard スクリプトの変換

Dashboard スクリプトを変換してツールキット API で作業するには、次の手順を実行します。

1. .toolkit ファイルを作成します。
2. ダッシュボード・スクリプトを変更します。
   a. `add_service dashboard <name of service>` コマンドを削除します。
   b. `dashboard_<command>` を `toolkit_<command>` に変更します。
C. open_service を claim_service に変更します。
次はその一例です。

```plaintext
open_service slave $path
master_read_memory $path address count
```
上記が次のようになります。

```plaintext
set c [claim_service slave $path lib {}]
master_read_memory $c address count
```

### 8.9.2. ツールキット記述ファイルの作成

ツールキット記述ファイル (.toolkit) は XML ファイルです。ツールキット記述ファイルによって、ツールキットの登録データが得られます。

次の属性をツールキット記述ファイルに含めてください。

<table>
<thead>
<tr>
<th>属性名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>内部ツールキット・ファイル名です。</td>
</tr>
<tr>
<td>displayName</td>
<td>GUI に表示されるツールキットの表示名</td>
</tr>
<tr>
<td>addMenuItem</td>
<td>システムコンソール Tools &gt; Toolkits メニューによるツールキットの表示有無の設定</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>属性名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>ツールキットの目的の記述</td>
</tr>
<tr>
<td>file</td>
<td>ツールキットの実装を含む .tcl ファイルへのパス</td>
</tr>
<tr>
<td>icon</td>
<td>システムコンソールでツールキット起動ボタンとして表示するアイコンへのパス</td>
</tr>
<tr>
<td>requirement</td>
<td>ツールキットが特定の種類のハードウェアと動作する場合、この属性によってハードウェアのデバッグタイプの名前が指定されます。この属性によって、ツールキットの自動検出がイネーブルになります。</td>
</tr>
</tbody>
</table>

例-23: ツールキット記述ファイル

```xml
<?xml version="1.0" encoding="UTF-8"?>
<toolkit name="toolkit_example" displayName="Toolkit Example"
addMenuItem="true">
<file> toolkit_example.tcl </file>
</toolkit>
```

関連情報
ツールキットと IP コアの組み合わせ (161 ページ)
8.9.3. ツールキットの登録

System Console で toolkit_register コマンドを使用して、ツールキットが使用できるようにします。.toolkit ファイルへのパスを指定します。ツールキットを登録しても、ツールキット GUI のインスタンスは作成されません。

```
toolkit_register <toolkit_file>
```

8.9.4. ツールキットの起動

System Console で事前に登録したツールキットを起動するには、次のいくつかの方法があります。

- **Tools > Toolkits** の順でクリックします。
- **Toolkits** タブを使用します。各ツールキットには、説明、検出されたハードウェアのリスト、および起動ボタンがあります。
- 次のコマンドを使用します。

```
toolkit_open <.toolkit_file_name>
```

ツールキットの起動は、ツールキットのタイプに関連付けられたハードウェア・リソースのコンテキストで行うことができます。

```
toolkit_open <toolkit_name> <context>
```

上のコマンドを使用する場合、ツールキット Tcl でのコンテキストの取得には、次のコマンドを入力します。

```
set context [toolkit_get_context]
```

関連情報

```
toolkit_get_context (172 ページ)
```

8.9.5. ツールキットと IP コアの組み合わせ

ツールキットには、任意の IP コアを組み合わせることが可能です。

- ツールキットでは、IP の検索時にデバッグマーカーを探し、IP コアをツールキットの要件と適合させます。ツールキット・ファイルで属性要件を使用して、デバッグタイプを次のように指定します。

```
<requirement><type>debug.type-name</type></requirement>
```

- hw.tcl でデバッグ・アサインメントを IP コア用に作成します。hw.tcl が使用できるのは、デザインを System Console にロードする場合です。

- System Console では、ハードウェアの識別子からデバッグマーカーを検出し、IP と関連付けます。このとき、デザインの直接的な情報は不要です。

8.9.6. Toolkit API

Toolkit API サービスを使用すると、GUI を構築してデバッグデータの視覚化と通信に使用することができます。Toolkit API は、視覚的ウィジェットのレイアウトのグラフィック画面です。これにはボタンやテキストフィールドが含まれます。ウィジェットでは、データこれらの例の手順は、他の System Console のサービスから行います。同様に、ウィジェットではサービスを使用して、ユーザー入力を活用し、デザインのデバッグロジックに作用します。
プロパティー

ウィジェット・プロパティーでは、情報のブッシュとプルをユーザー・インターフェイスに対して行うことが
できます。ウィジェットのプロパティーはそのタイプに固有のものです。例えば、あるボタンをクリックする
と、ボタン・プロパティーの onClick によって動作が実行されます。ラベル・ウィジェットに同じプロパ
ティーがないのは、ウィジェットによるアクションがクリック操作では行われないためですが、ボタンとラ
ベル・ウィジェットの両方にある text プロパティーによってテキスト文字列が表示されます。

レイアウト

Toolkit API サービスで作成されるウィジェット階層では、ツールキットはトップレベルにあります。この
サービスによって実装されるグループ型ウィジェットには子ウィジェットが含まれます。レイアウト・プロ
パティーによって決定されるレイアウト動作は、親が子に対して実行します。例えば、expandableX
プロパティーを True に設定すると、ウィジェットが水平方向に展開され、使用可能なスペースすべてが
含まれます。また、visible プロパティーを True に設定すると、ウィジェットによって GUI が表示
できます。

ユーザーチャイス

一部のウィジェットでは、ユーザーによる操作が可能です。例えば、textField ウィジェットは、ユー
ザーによる入力が可能なテキストボックスです。テキストボックスの内容にアクセスするには、text プ
ロパティーを使用します。Tcl スクリプトによる textField ウィジェットの内容の取得または設定に
は、text プロパティーを使用します。

コールバック

一部のウィジェットによって実行されるユーザー指定の動作は、コールバックと呼ばれます。
textField ウィジェットの onChange プロパティーは、テキストの内容が変更されると呼び出され
ます。button ウィジェットの onClick は、ボタンをクリックすると呼び出されます。コールバックに
によるウィジェットの更新やサービスとの通信は、テキストフィールドの内容や他のウィジェットの状態に
基づいて行われます。

8.9.6.1. Toolkit API ウィジェットのカスタマイズ

toolkit_set_property コマンドを使用して、インスタンス化するウィジェットと通信します。こ
の toolkit_set_property コマンドが非常に有用になるのは、コールバックの実行の一部を
変更する場合です。

8.9.6.2. Toolkit API スクリプトの例

例-24: System Console での Toolkit の可視化

toolbox_set_property コマンドを使用して、ルート・ツールキットの visible プロパティー
を変更します。プロパティーがツールキット全体に適用されている場合は、ワード self を使用します。
それ以外の場合は、all を使用してルート・ツールキットを参照します。

toolkit_set_property self visible true

例-25: ウィジェットの追加

toolbox_add コマンドを使用してウィジェットを追加します。

toolkit_add my_button button all
次のコマンドでは、ラベルウィジェットのmy_labelをルートツールキットに追加します。GUIでは、このラベルの表示はWidget Labelになります。

```
set name "my_label"
set content "Widget Label"
toolkit_add $name label all
toolkit_set_property $name text $content
```

GUIでは、表示テキストは変更されて新しい値になります。ラベルをもう1つ追加するには、次を実行します。

```
toolkit_add my_label_2 label all
toolkit_set_property my_label_2 text "Another label"
```

新しいラベルは、最初のラベルの右側に表示されます。
新しいラベルを最初のラベルの下に配置するには、次のコマンドを使用します。

```
toolkit_set_property self itemsPerRow 1
```

例-26: 入力の収集

ユーザー入力をToolkit APIに組み込むには、次を実行します。

1. 次のコマンドを使用してテキストフィールドを作成します。

```
set name "my_text_field"
set widget_type "textField"
set parent "all"
toolkit_add $name $widget_type $parent
```

2. ウィジェットのサイズは非常に小さいため、ウィジェットで水平方向のスペースを埋めるには次のコマンドを使用します。

```
toolkit_set_property my_text_field expandableX true
```

3. これで、テキストフィールドは完全に可視化されます。クリックでフィールドにテキストを入力できます。フィールドの内容を取得するには、次のコマンドを使用します。

```
set content [toolkit_get_property my_text_field text]
puts $content
```

このコマンドによって内容がコンソールに表示されます。

例-27: ユーザーイベント時のウィジェットの更新

コールバックを使用する場合、Toolkit APIによる動作の実行は、インタラクティブ入力なしでも可能です。

1. まず、テキストフィールドの内容で最初のラベルを更新するプロシージャーを定義します。

```
proc update_my_label_with_my_text_field{
    set content [toolkit_get_property my_text_field text]
    toolkit_set_property my_label text $content
}
```
2. `update_my_label_with_my_text_field` コマンドを `Tcl Console` で実行します。これで、最初のラベルがテキストフィールドの内容と一致します。

3. `update_my_label_with_my_text_field` コマンドは、テキストフィールドの変更のたびに使用します。

```tcl
toolkit_set_property my_text_field onChange
update_my_label_with_my_text_field
```

ToolKitでは、テキストフィールドの変更のたびに `onChange` プロパティを実行します。このプロパティを実行すると、最初のフィールドが変更されて入力した内容と一致するようになります。

例-28:

ボタン

ボタンを使用してアクションをトリガーします。

1. 2番目のラベルを変更するボタンを作成します。

```tcl
proc append_to_my_label_2 {suffix} {
    set old_text [toolkit_get_property my_label_2 text]
    set new_text "${old_text}${suffix}"
    toolkit_set_property my_label_2 text $new_text
}
set text_to_append ', and more'
toolkit_add my_button button all
toolkit_set_property my_button onClick [append_to_my_label_2 $text_to_append]
```

2. ボタンをクリックして、2番目のラベルにテキストを追加します。

例-29:

グループ

`itemsPerRow` プロパティは、グループ内のウィジェットのレイアウトを決定します。レイアウトが複雑で、行ごとのウィジェットの数が異なる場合は、ネストされたグループを使用します。新しいグループの追加を行ごとにより多くのウィジェットでは、次のコマンドを使用します。

```tcl
toolkit_add my_inner_group group all
toolkit_set_property my_inner_group itemsPerRow 2
```

これらのコマンドで作成する行には、2つのボタンのグループがあります。ネストされたグループをよりシームレスにするには、次のコマンドを使用して、グループ名を持つボーダーを削除します。

```tcl
toolkit_set_property my_inner_group title ""
```

`title` プロパティを他の文字列に設定して、ボーダーおよびタイトルテキストが確実に表示されるようにします。

例-30:

タブ

タブを使用してウィジェットの可視性を管理します。

```tcl
toolkit_add my_tabs tabbedGroup all
toolkit_set_property my_tabs expandableX true
toolkit_add my_tab_1 group my_tabs
toolkit_add my_tab_2 group my_tabs
toolkit_add tabbed_label_1 label my_tab_1
toolkit_add tabbed_label_2 label my_tab_2
toolkit_set_property tabbed_label_1 text "in the first tab"
toolkit_set_property tabbed_label_2 text "in the second tab"
```
これらのコマンドによって2つのタブのセットが追加され、それぞれのタブにラベルを含むグループができます。タブをクリックすると、表示のグループ/ラベルが変更されます。

8.9.6.3. Toolkit API GUIの例

次の例で示すのは、インタラクティブGUIウィンドウを含むツールキットの登録および起動方法です。
1. ツールキット記述ファイルを作成します。実際の例については、ツールキット記述ファイルの作成を参照してください。
2. .tclファイルを生成するために、Toolkit API GUI Example .tcl Fileのテキストを使用します。
3. System Consoleを起動します。
4. ツールキットの登録をTcl Consoleペインで行います。使用するファイルのロケーションへの相対パスを含めます。

図-90: 使用するツールキットの登録

ToolkitがToolkitsタブに表示されます。

図-91: Toolkit Example登録後のToolkitsタブ

5. Launchリンクをクリックします。
新しいタブが表示されます。これに含まれるウィジェットは、Tcl ファイルで指定したものです。

図 -92: Toolkit Example GUI

Send Data フィールドにテキストを入力し、Launch をクリックすると、そのテキストが Receive Data フィールドに表示されます。

関連情報
ツールキット記述ファイルの作成 (160 ページ)

8.9.6.3.1. Toolkit API GUI Example .tcl ファイル

次の Toolkit API .tcl ファイルで作成される GUI ウィンドウでは、デザインとのデバッグ・インタラクションが提供されます。

```tcl
namespace eval Test {  
    variable ledValue 0  
    variable dashboardActive 0  
    variable Switch_off 1  

    proc toggle { position } {  
        set ::Test::ledValue ${position}  
        ::Test::updateDashboard  
    }  

    proc sendText {} {  
        set sendText [toolkit_get_property sendTextText text]  
        toolkit_set_property receiveTextText text $sendText  
    }  

    proc dashBoard {} {  
        if { ${::Test::dashboardActive} == 1 } {  
            return -code ok "dashboard already active"  
        }  

        set ::Test::dashboardActive 1  
        # top group widget  
        toolkit_add topGroup group self  
        toolkit_set_property topGroup expandableX false  
        toolkit_set_property topGroup expandableY false
    }
```
toolkit_set_property topGroup itemsPerRow 1
toolkit_set_property topGroup title ""

# leds group widget
#
toolkit_add ledsGroup group topGroup
toolkit_set_property ledsGroup expandableX false
toolkit_set_property ledsGroup expandableY false
toolkit_set_property ledsGroup itemsPerRow 2
toolkit_set_property ledsGroup title "LED State"

# leds widgets
#
toolkit_add led0Button button ledsGroup
toolkit_set_property led0Button enabled true
toolkit_set_property led0Button expandableX false
toolkit_set_property led0Button expandableY false
toolkit_set_property led0Button text "Toggle"
toolkit_set_property led0Button onClick {::Test::toggle 1}

toolkit_add led0LED led ledsGroup
toolkit_set_property led0LED expandableX false
toolkit_set_property led0LED expandableY false
toolkit_set_property led0LED text "LED 0"
toolkit_set_property led0LED color "green_off"

toolkit_add led1Button button ledsGroup
toolkit_set_property led1Button enabled true
toolkit_set_property led1Button expandableX false
toolkit_set_property led1Button expandableY false
toolkit_set_property led1Button text "Turn ON"
toolkit_set_property led1Button onClick {::Test::toggle 2}

toolkit_add led1LED led ledsGroup
toolkit_set_property led1LED expandableX false
toolkit_set_property led1LED expandableY false
toolkit_set_property led1LED text "LED 1"
toolkit_set_property led1LED color "green_off"

# sendText widgets
#
toolkit_add sendTextGroup group topGroup
toolkit_set_property sendTextGroup expandableX false
toolkit_set_property sendTextGroup expandableY false
toolkit_set_property sendTextGroup itemsPerRow 1
toolkit_set_property sendTextGroup title "Send Data"

toolkit_add sendTextText text sendTextGroup
toolkit_set_property sendTextText expandableX false
toolkit_set_property sendTextText expandableY false
toolkit_set_property sendTextText preferredWidth 200
toolkit_set_property sendTextText preferredHeight 100
toolkit_set_property sendTextText htmlCapable false
toolkit_set_property sendTextText text ""

toolkit_add sendTextButton button sendTextGroup
toolkit_set_property sendTextButton enabled true
toolkit_set_property sendTextButton expandableX false
toolkit_set_property sendTextButton expandableY false
toolkit_set_property sendTextButton text "Send Now"
toolkit_set_property sendTextButton onClick {::Test::sendText}

# receiveText widgets
#
toolkit_add receiveTextGroup group topGroup
8.9.6.4. Toolkit API コマンド

Tktool API コマンドは、固有のツールキット・インスタンスに関連して実行されます。

- **toolkit_register** (169 ページ)
- **toolkit_open** (170 ページ)
- **get_quartus_ini** (171 ページ)
- **toolkit_get_context** (172 ページ)
- **toolkit_get_types** (173 ページ)
- **toolkit_get_properties** (174 ページ)
- **toolkit_add** (175 ページ)
- **toolkit_get_property** (176 ページ)
- **toolkit_set_property** (177 ページ)
- **toolkit_remove** (178 ページ)
- **toolkit_get_widget_dimensions** (179 ページ)
8.9.6.4.1. toolkit_register

説明
プラグイン (.toolkit ファイル) を記述する XML ファイルを指します。

使用方法
toolkit_register <toolkit_file>

戻り値
戻り値はありません。

引数
<toolkit_file> ツールキット定義ファイルへのパスです。

例

```sh
toolkit_register /path/to/toolkit_example.toolkit
```
8.9.6.4.2. toolkit_open

説明
System Console でツールキットのインスタンスを開きます。

使用方法
toolkit_open <toolkit_id> [context]

戻り値
戻り値はありません。

引数
<toolkit_id> 開くツールキット・タイプ名です。

コンテキスト オプションのコンテキストです。例えば、開くツールキットに関連付けられているハードウェア・リソースのサービスパスです。

例
toolkit_open my_toolkit_id
8.9.6.4.3. get_quartus_ini

説明
ini 設定の値を インテル Quartus Prime 開発ソフトウェアの .ini ファイルから返します。

使用方法
get_quartus_ini <ini> <type>

戻り値
ini 設定の値です。

引数
<br />
<br /></br> <ini> インテル Quartus Prime 開発ソフトウェア .ini 設定の名称です。
<br />
<br /></br> <type> (オプション) .ini 設定のタイプです。既知のタイプは、string と enabled です。このタイプが enabled の場合、.ini 設定の値は 1 を返し、enabled 以外の場合は 0 を返します。

例
<br />
<br /></br> set my_ini_enabled [get_quartus_ini my_ini enabled]
<br />
<br /></br> set my_ini_raw_value [get_quartus_ini my_ini]
8.9.6.4.4. toolkit_get_context

説明
ツールキットを開いたときに指定したコンテキストを返します。コンテキストが指定されていない場合は、空の文字列を返します。

使用方法

```
toolkit_get_context
```

戻り値
コンテキストです。

引数
引数はありません。

例

```
set context [toolkit_get_context]
```
8.9.6.4.5. **toolkit_get_types**

**説明**
ウィジェット・タイプのリストを返します。

**使用方法**
toolkit_get_types

**戻り値**
ウィジェット・タイプのリストです。

**引数**
引数はありません。

**例**
```
set widget_names [toolkit_get_types]
```
8.9.6.4.6. toolkit_get_properties

説明
特定のタイプのウィジェット用のツールキット・プロパティーのリストを返します。

使用方法
toolkit_get_properties <widgetType>

戻り値
ツールキット・プロパティーのリストです。

引数
<widgetType>　ウィジェットのタイプです。

例
set widget_properties [toolkit_get_properties xyChart]
8. System Console を使用したデザインの解析とデバッグ
UG-20139 | 2018.09.24

8.9.6.4.7. toolkit_add

説明
現在のツールキットにウィジェットを追加します。

使用方法
toolkit_add <id> <type> <groupid>

戻り値
戻り値はありません。

引数

<id> 追加されるウィジェットの一意の ID です。

<type> 追加されるウィジェットのタイプです。

<groupid> 新しいウィジェットを含む親グループの ID です。ツールキット・ベース・グループには self を使用します。

例
toolkit_add my_button button parentGroup
8.9.6.4.8. toolkit_get_property

説明
特定のウィジェットのプロパティー値を返します。

使用方法
```
toolkit_get_property <id> <propertyName>
```

戻り値
プロパティー値です。

引数

```
<i id>
クエリーされるウィジェットの一意のIDです。

<propertyName>
ウィジェット・プロパティーの名前です。
```

例
```
set enabled [toolkit_get_property my_button enabled]
```
8.9.6.4.9. toolkit_set_property

説明
特定のウィジェットのプロパティー値を設定します。

使用方法
toolkit_set_property <id><propertyName> <value>

戻り値
戻り値はありません。

引数

<id> 編集されるウィジェットの一意の ID です。

<propertyName> 設定されるウィジェット・プロパティーの名前です。

<value> ウィジェット・プロパティーの新しい値です。

例

```
toolkit_set_property my_button enabled 0
```
8.9.6.4.10. toolkit_remove

説明
指定したツールキットからウィジェットを削除します。

使用方法
toolkit_remove <id>

に戻り値
戻り値はありません。

引数

<iid> 削除されるウィジェットの一意の ID です。

例
toolkit_remove my_button
8.9.6.4.11. toolkit_get_widget_dimensions

説明
指定したウィジェットの幅と高さを返します。

使用方法
toolkit_get_widget_dimensions <id>

戻り値
指定したウィジェットの幅と高さです。

引数

<id> 追加されるウィジェットの一意の ID です。

例

set dimensions [toolkit_get_widget_dimensions my_button]
8.9.6.5. ツールキット API プロパティー

ツールキット API ウィジェットのプロパティーは次のとおりです。

ウィジェットのタイプとプロパティー (181 ページ)
barChart プロパティー (182 ページ)
button プロパティー (183 ページ)
checkBox プロパティー (184 ページ)
comboBox プロパティー (185 ページ)
dia プロパティー (186 ページ)
fileChooserButton プロパティー (187 ページ)
group プロパティー (188 ページ)
label プロパティー (189 ページ)
led プロパティー (190 ページ)
lineChart プロパティー (191 ページ)
list プロパティー (192 ページ)
pieChart プロパティー (193 ページ)
table プロパティー (194 ページ)
text プロパティー (195 ページ)
textField プロパティー (196 ページ)
timeChart プロパティー (197 ページ)
xyChart プロパティー (198 ページ)
### 8.9.6.5.1. ウィジェットのタイプとプロパティー

<table>
<thead>
<tr>
<th>パラメータ名</th>
<th>詳細</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>ウィジェットをイネーブル/ディスエーブルします。</td>
</tr>
<tr>
<td>expandable</td>
<td>ウィジェットを展開可能にするかどうかを制御します。</td>
</tr>
<tr>
<td>expandableX</td>
<td>ウィジェットが存在するセルに使用可能なスペースがある場合、ウィジェットのサイズを水平方向に変更できます。</td>
</tr>
<tr>
<td>expandableY</td>
<td>ウィジェットが存在するセルに使用可能なスペースがある場合、ウィジェットのサイズを垂直方向に変更できます。</td>
</tr>
<tr>
<td>foregroundColor</td>
<td>文字表示色を設定します。</td>
</tr>
<tr>
<td>maxHeight</td>
<td>ウィジェットのexpandableYが設定されている場合、これがウィジェットが取ることができるピクセル単位での最大の高さです。</td>
</tr>
<tr>
<td>minHeight</td>
<td>ウィジェットのexpandableYが設定されている場合、これがウィジェットが取ることができるようにピクセル単位での最小の高さです。</td>
</tr>
<tr>
<td>maxWidth</td>
<td>ウィジェットのexpandableXが設定されている場合、これがウィジェットが取ることができるようにピクセル単位での最大の幅です。</td>
</tr>
<tr>
<td>minWidth</td>
<td>ウィジェットのexpandableXが設定されている場合、これがウィジェットが取ことができるようにピクセル単位での最小の幅です。</td>
</tr>
<tr>
<td>preferredHeight</td>
<td>expandableYが設定されていない場合のウィジェットの高さです。</td>
</tr>
<tr>
<td>preferredWidth</td>
<td>expandableXが設定されていない場合のウィジェットの幅です。</td>
</tr>
<tr>
<td>toolTip</td>
<td>マウスオーバーのツールチップを実装します。</td>
</tr>
<tr>
<td>visible</td>
<td>ウィジェットを表示します。</td>
</tr>
</tbody>
</table>
### 8.9.6.5.2. barChart プロパティー

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>チャートのタイトルです。</td>
</tr>
<tr>
<td>labelX</td>
<td>X 軸のラベルテキストです。</td>
</tr>
<tr>
<td>label</td>
<td>X 軸のラベルテキストです。</td>
</tr>
<tr>
<td>range</td>
<td>Y 軸の値の範囲です。デフォルトはオートレンジです。Tclリストを使用して範囲を指定します。次はその一例です。</td>
</tr>
<tr>
<td>itemValue</td>
<td>Tclリストを使用して値を指定します。次はその一例です。</td>
</tr>
</tbody>
</table>

Tclリストを使用して範囲を指定します。次はその一例です。
```tcl
[list lower_numerical_value upper_numerical_value]
```
8.9.6.5.3. button プロパティ

### 表 35. ツールキット API button プロパティ

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>onClick</td>
<td>ボタンをクリックすると実行されるTclコマンドを指定します。通 常、コマンドはprocです。</td>
</tr>
<tr>
<td>text</td>
<td>ボタン上のテキストです。</td>
</tr>
</tbody>
</table>
8.9.6.5.4. checkBox プロパティ

表 36. ツールキット API checkBox プロパティ

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>checked</td>
<td>チェックボックスの状態を指定します。</td>
</tr>
<tr>
<td>onClick</td>
<td>チェックボックスをクリックするたびに実行されるTclコマンドを指定します。通常、コマンドはprocです。</td>
</tr>
<tr>
<td>text</td>
<td>チェックボックスのテキストです。</td>
</tr>
</tbody>
</table>
8.9.6.5.5. comboBox プロパティ

表 37. ツールキット API comboBox プロパティ

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>onChange</td>
<td>コンボボックスの値が変更された場合に実行されるTclコールバックです。</td>
</tr>
<tr>
<td>options</td>
<td>コンボボックスに表示する項目のリストです。</td>
</tr>
<tr>
<td>selectedItem</td>
<td>コンボボックスで選択した項目です。</td>
</tr>
</tbody>
</table>
### 8.9.6.5.6. dial プロパティ

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>ダイヤルで表示できる最大値です。</td>
</tr>
<tr>
<td>Min</td>
<td>ダイヤルで表示できる最小値です。</td>
</tr>
<tr>
<td>ticksize</td>
<td>ダイヤルの異なるチェックマーク間のスペースです。</td>
</tr>
<tr>
<td>title</td>
<td>ダイヤルのタイトルです。</td>
</tr>
<tr>
<td>value</td>
<td>ダイヤルの針がマークする値です。この値は、最小値と最大値の間でなければなりません。</td>
</tr>
</tbody>
</table>
### 8.9.6.5.7. fileChooserButton プロパティ

#### 表 39. タールキット API fileChooserButton プロパティ

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>ボタン上のテキストです。</td>
</tr>
<tr>
<td>onChoose</td>
<td>ボタンをクリックするたびに実行される Tcl コマンドです。通常、コマンドは proc です。</td>
</tr>
<tr>
<td>title</td>
<td>ダイアログボックスのタイトルです。</td>
</tr>
<tr>
<td>chooserButtonText</td>
<td>ダイアログボックスの承認ボタンのテキストです。デフォルト値は Open です。</td>
</tr>
</tbody>
</table>
| filter            | 拡張子に基づいたファイルフィルターです。フィルターによってサポートされるのは、1つの拡張子のみです。デフォルトでは、フィルターではすべてのファイル名の使用を許可します。構文の [list filter_description file_extension] を使用してフィルターを指定します。次はその一例です。
|                   | [list "Text Document (.txt)" "txt"]                                   |
| mode              | 選択可能なファイルまたはディレクトリーの種類を指定します。デフォルトは、files_only です。選択可能なオプションは、files_only と directories_only です。 |
| multiSelectionEnabled | 複数のファイルを選択可能にするかどうかを制御します。デフォルト値は false です。|
| paths             | このプロパティは読み出し専用です。ファイル選択ダイアログボックスで選択したファイルパスのリストを返します。このプロパティが非常に有用なのは、onclick スクリプト内、またはダイアログボックスを閉じた後に結果を更新するプロシージャー内で使用する場合です。 |

---

フィードバック

インタール Quartus Prime プロ・エディション ユーザーガイド: デバッガツール
### 8.9.6.5.8. group プロパティー

**表 40. ツールキット API group プロパティー**

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemsPerRow</td>
<td>次の行に移る前にグループによって１行で左から右に配置できるウィジェットの数です。</td>
</tr>
<tr>
<td>title</td>
<td>グループのタイトルです。タイトルを持つグループでは、グループの周りに境界線を持つことができ、空のタイトルを設定すると境界線が削除されます。</td>
</tr>
</tbody>
</table>
### 8.9.6.5.9. label プロパティー

表 41. ソールキット API label プロパティー

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>ラベルに表示するテキストです。</td>
</tr>
</tbody>
</table>
表 42. ツールキット API led プロパティ

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>LED の色です。オプションは次の通りです。red_off、red、yellow_off、yellow、green_off、green、blue_off、blue、black。</td>
</tr>
<tr>
<td>text</td>
<td>LED の隣に表示するテキストです。</td>
</tr>
</tbody>
</table>
### 8.9.6.5.11. lineChart プロパティ

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>タイトル</td>
<td>チャートのタイトルです。</td>
</tr>
<tr>
<td>labelX</td>
<td>X 軸のラベルテキストです。</td>
</tr>
<tr>
<td>labelY</td>
<td>Y 軸のラベルテキストです。</td>
</tr>
</tbody>
</table>
| range | Y 軸の値の範囲です。デフォルトはオートレンジです。Tcl リストを使用して範囲を指定します。次はその一例です。

```
[list lower_numerical_value upper_numerical_value]
```

| itemValue | アイテムの値です。Tcl リストを使用して値を指定します。次はその一例です。

```
[list bar_category_str numerical_value]
```
### 8.9.6.5.12. list プロパティ

#### 表 44. ツールキット API list プロパティー

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>selected</td>
<td>コンボボックスで選択した項目のインデックスです。</td>
</tr>
<tr>
<td>options</td>
<td>表示するオプションのリストです。</td>
</tr>
<tr>
<td>onChange</td>
<td>Tcl コールバックです。リスト内で選択された項目が変更された場合に実行されます。</td>
</tr>
</tbody>
</table>
### 8.9.6.5.13. pieChart プロパティ

表 45. ツールキット API pieChart プロパティ

<table>
<thead>
<tr>
<th>パラメータ名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>タイトル</td>
<td>チャートのタイトルです。</td>
</tr>
<tr>
<td>itemValue</td>
<td>アイテムの値です。Tclリストを使用して値を指定します。次の例で</td>
</tr>
<tr>
<td></td>
<td>[list bar_category_str numerical_value]</td>
</tr>
</tbody>
</table>
### 8.9.6.5.14. table プロパティ

表 46. ツールキット API table プロパティ

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnCount</td>
<td>列の数 (必須) (デフォルト値は 0)。</td>
</tr>
<tr>
<td>rowCount</td>
<td>行の数 (必須) (デフォルト値は 0)。</td>
</tr>
<tr>
<td>headerReorderingAllowed</td>
<td>列のドラッグを可能にするかどうかを制御します (デフォルト値は false)。</td>
</tr>
<tr>
<td>headerResizingAllowed</td>
<td>すべての列幅を変更可能にするかどうかを制御します (デフォルト値は false)。注意: 個々の列のサイズは、columnWidthResizable プロパティで調整可能です。</td>
</tr>
<tr>
<td>rowSorterEnabled</td>
<td>列のセルの値を並び替え可能にするかどうかを制御します (デフォルト値は false)。</td>
</tr>
<tr>
<td>showGrid</td>
<td>水平線と垂直線の両方を描画可能にするかどうかを制御します (デフォルト値は true)。</td>
</tr>
<tr>
<td>showHorizontalLines</td>
<td>水平線を描画可能にするかどうかを制御します (デフォルト値は true)。</td>
</tr>
<tr>
<td>rowIndex</td>
<td>現在の行のインデックスです。この行はゼロベースです。この値は、以下のすべてのプロパティの一部に影響します (デフォルト値は 0)。</td>
</tr>
<tr>
<td>columnIndex</td>
<td>現在の列のインデックスです。この列はゼロベースです。この値は、以下のすべてのプロパティの一部に影響します (デフォルト値は 0)。</td>
</tr>
<tr>
<td>cellText</td>
<td>現在の rowIndex および columnIndex によって付与されたセル内のテキストを指定します (デフォルトは Empty)。</td>
</tr>
<tr>
<td>selectedRows</td>
<td>行の選択を制御または取得します。</td>
</tr>
<tr>
<td>columnHeader</td>
<td>列のヘッダーのテキストです。</td>
</tr>
<tr>
<td>columnHeaders</td>
<td>チャートの列を定義する名前のリストです。</td>
</tr>
<tr>
<td>columnHorizontalAlignment</td>
<td>指定した列のセルテキストの配置です。サポートされているタイプは、leading (デフォルト)、left、center、right、trailing です。</td>
</tr>
<tr>
<td>columnRowSorterType</td>
<td>並び替え方法の種類です。これは、rowSorterEnabled が true の場合にのみ適用可能です。各列には固有のソート方法があります。選択可能な種類は、string (デフォルト)、int、float です。</td>
</tr>
<tr>
<td>columnWidth</td>
<td>列幅内のピクセル数です。</td>
</tr>
<tr>
<td>columnWidthResizable</td>
<td>ユーザーによる列幅の変更を可能にするかどうかを制御します (デフォルト値は false)。</td>
</tr>
<tr>
<td>contents</td>
<td>リストとしてのチャートの内容。列 A、B、C を持つチャートの場合、リストの形式は、(A1 B1 C1 A2 B2 C2 etc) です。</td>
</tr>
</tbody>
</table>
### 8.9.6.5.15. text プロパティ

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>editable</td>
<td>テキストボックスを編集可能にするかどうかを制御します。</td>
</tr>
<tr>
<td>htmlCapable</td>
<td>テキストボックスで HTML をフォーマット可能にするかどうかを制御します。</td>
</tr>
<tr>
<td>text</td>
<td>テキストボックスに表示するテキストです。</td>
</tr>
</tbody>
</table>
### 8.9.6.5.16. textField プロパティー

表 48. ツールキット API textField プロパティー

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>editable</td>
<td>テキストボックスを編集可能にするかどうかを制御します。</td>
</tr>
<tr>
<td>onChange</td>
<td>Tcl コールバック。テキストボックスの内容を変更する場合に実行されます。</td>
</tr>
<tr>
<td>text</td>
<td>テキストボックス内のテキストです。</td>
</tr>
</tbody>
</table>
### 8.9.6.5.17. timeChart プロパティー

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>概要</th>
</tr>
</thead>
<tbody>
<tr>
<td>labelX</td>
<td>X 軸のラベルです。</td>
</tr>
<tr>
<td>labelY</td>
<td>Y 軸のラベルです。</td>
</tr>
<tr>
<td>latest</td>
<td>系列内の最新の値です。</td>
</tr>
<tr>
<td>maximumItemCount</td>
<td>履歴レコードに表示するサンプルポイントの数です。</td>
</tr>
<tr>
<td>title</td>
<td>チャートのタイトルです。</td>
</tr>
<tr>
<td>range</td>
<td>チャートの範囲を設定します。この形式の範囲は、(low, high)です。このlow/highの値は2倍になります。</td>
</tr>
<tr>
<td>showLegend</td>
<td>系列の凡例をグラフに表示するかどうかを指定します。</td>
</tr>
</tbody>
</table>
8.9.6.5.18. xyChart プロパティー

表 50. ツールキット API xyChart プロパティー

<table>
<thead>
<tr>
<th>パラメーター名</th>
<th>プロパティ</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>チャートのタイトルです。</td>
</tr>
<tr>
<td>labelX</td>
<td>X 軸のラベルテキストです。</td>
</tr>
<tr>
<td>labelY</td>
<td>Y 軸のラベルテキストです。</td>
</tr>
<tr>
<td>range</td>
<td>チャートの範囲を設定します。この形式の範囲は、{low, high} です。この low/high の値は 2 倍になります。</td>
</tr>
<tr>
<td>maximumItemCount</td>
<td>データ系列に保持するデータ値の最大数を指定します。この設定が影響するのは、チャートの新しいデータのみです。maximumItemCount を超えるデータ値を追加する場合、最後の maximumItemCount のエントリーのみが保持されます。</td>
</tr>
<tr>
<td>series</td>
<td>チャートに一連のデータを追加します。series の最初の値はその系列の識別子です。同じ識別子が 2 回設定されると、ツールキット API は最も新しい系列を選択します。識別子に系列データが含まれていない場合、その系列はチャートから削除されます。Tcl リストで次のように系列を指定します。{identifier, x-1 y-1, x-2 y-2}</td>
</tr>
<tr>
<td>showLegend</td>
<td>系列の凡例をグラフに表示するかどうかを指定します。</td>
</tr>
</tbody>
</table>

8.10. System Console の例とチュートリアル

インテルで提供している例は、ボードの立ち上げ、単純なダッシュボードの作成、および Nios II プロセッサーのプログラミング用です。System_Console.zip ファイルに含まれているデザインファイルは、ボードの立ち上げの例です。また、Nios II Ethernet Standard .zip ファイルに含まれているデザインファイルは、Nios II プロセッサーの例です。

注意: これらの例の手順は、ユーザーにインテル Quartus Prime 開発ソフトウェア、Tcl コマンド、およびプラットフォーム・デザイナー (旧: Qsys) についての一定の知識があることが前提となっています。

関連情報
On-Chip Debugging Design Examples
ダウンロード可能なデザイン例のデザインファイルが含まれています。

8.10.1. Nios II プロセッサーの例

このデザイン例では、Nios II プロセッサーをボード上でプログラミングして、Nios II 実装に含まれるカウント・バイナリー・ソフトウェアのデザイン例を実行します。この簡単なプログラムでは、8 ビットの変数を使用して、0x00 から 0xFF まで繰り返しカウントします。この変数の出力はボード上の LED に表示されます。Nios II プロセッサーのプログラミング後、System Console のプロセッサーコマンドを使用して、プロセッサーの開始および停止を実行します。

この例を実行するには、次の手順を実行します。
1. Nios II Ethernet Standard Design Example をアルテラのウェブサイトからダウンロードして、ボードで使用します。
2. デザイン抽出用のフォルダーを作成します。この例では、C:\Count_binary を使用します。
3. Nios II Ethernet Standard Design Example を C:\Count_binary に解凍します。
4. Nios II のコマンドシェルで、新規プロジェクトのディレクトリーに変更します。
5. ボードをプログラミングします。Nios IIのコマンドシェルで次を入力します。

```
nios2-configure-sof niosii_ethernet_standard_<board_version>.sof
```

6. Nios II Software Build Tools for Eclipseを使用して、Templateから新規のNios II ApplicationとBSPを作成します。これには、Count Binaryテンプレートを活用し、Nios II Ethernet Standard Design Exampleをターゲットとします。

7. Executable and Linking Format (ELF)ファイル(.elf)をアプリケーション用にビルドするには、Count Binaryプロジェクトを右クリックし、Build Projectを選択します。

8. .elfファイルをボードにダウンロードするために、Count Binaryプロジェクトを右クリックし、Run As, Nios II Hardwareを選択します。

   • ボードのLEDに、新たにライトが点灯します。

9. 次のとおり入力します。

```
list:

system-console; #Start System Console.

#Set the processor service path to the Nios II processor.
set niosii_proc [lindex [get_service_paths processor] 0]
set claimed_proc [claim_service processor $niosii_proc mylib]; #Open the service.
processor_stop $claimed_proc; #Stop the processor.
#The LEDs on your board freeze.
processor_run $claimed_proc; #Start the processor.
#The LEDs on your board resume their previous activity.
processor_stop $claimed_proc; #Stop the processor.
close_service processor $claimed_proc; #Close the service.
```

   • processor_step、processor_set_register、およびprocessor_get_registerコマンドによって、Nios IIプロセッサーに追加の制御が提供されます。

関連情報

- Nios II Ethernet Standard Design Example
- Nios II Gen2 Software Developer's Handbook
8.10.1.1. プロセッサ・コマンド

<table>
<thead>
<tr>
<th>コマンド(1)</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor_download_elf</td>
<td>&lt;service-path&gt; &lt;elf-file-path&gt;</td>
<td>所定の Executable and Linking Format File (.elf) をダウンロードします。プロセッサーに関連付けられたマスターサービスを使用します。プロセッサーのプログラムカウンターを .elf エントリーポイントに設定します。</td>
</tr>
<tr>
<td>processor_in_debug_mode</td>
<td>&lt;service-path&gt;</td>
<td>プロセッサがデバッグモードの場合に、ゼロ以外の値を返します。</td>
</tr>
<tr>
<td>processor_reset</td>
<td>&lt;service-path&gt;</td>
<td>プロセッサーをリセットし、デバッグモードにします。</td>
</tr>
<tr>
<td>processor_run</td>
<td>&lt;service-path&gt;</td>
<td>プロセッサーを実行モードにします。</td>
</tr>
<tr>
<td>processor_stop</td>
<td>&lt;service-path&gt;</td>
<td>プロセッサーをデバッグモードにします。</td>
</tr>
<tr>
<td>processor_step</td>
<td>&lt;service-path&gt;</td>
<td>1 つのアセンブリー命令を実行します。</td>
</tr>
<tr>
<td>processor_get_register_names</td>
<td>&lt;service-path&gt;</td>
<td>プロセッサによってアクセス可能な全レジスターの名前リストを返します。</td>
</tr>
<tr>
<td>processor_get_register</td>
<td>&lt;service-path&gt; &lt;register_name&gt;</td>
<td>指定したレジスターの値を返します。</td>
</tr>
<tr>
<td>processor_set_register</td>
<td>&lt;service-path&gt; &lt;register_name&gt; &lt;value&gt;</td>
<td>指定したレジスターの値を設定します。</td>
</tr>
</tbody>
</table>

関連情報
Nios II プロセッサーの例 (198 ページ)

8.11. On-Board インテル FPGA ダウンロード・ケーブル II のサポート

System Console では、USB Debug Master IP コンポーネントを介した On-Board インテル FPGA ダウンロード・ケーブル II 回路をサポートしています。この IP コアではマスターサービスをサポートしています。

8.12. システム検証フローにおける MATLAB と Simulink*

System Console でシステム開発をテストするには、MATLAB および Simulink*を使用します。また、システム検証フローを設定するには、Intel FPGA Hardware in the Loop (HIL) ソールを使用します。この方法では、デザイン・ハードウェアを配置してリアルタイムで実行し、システム周辺のコンポーネントのシミュレーションをソフトウェア環境で行い、HIL アプローチでは、ソフトウェア・ツールの柔軟性とハードウェアの実際の精度および速度を併用できます。ハードウェア・コンポーネントの段階的な追加をシステム検証テストベンチに対して実行することができます。この手法を使用すると、経験プロセスをより一層制御することが、システムの調整および検証時に可能になります。完全なシステムが統合されると、HIL アプローチでは、スティミュラスをソフトウェアを介して提供し、システムの検証をさまざまなシナリオで行うことができます。

(1) システムにデータキャッシュ付きの Nios II/f コアが含まれていると、デバッグプロセスが複雑になる可能性があります。Nios II/f コアが非決定的な間隔でデータキャッシュからメモリに書き込みを行っていると思われる場合、つまり System Console によって書き込まれるデータを上書きしている場合、Nios II/f コアのキャッシュのディスエーブルをデバッグ中にすることがあります。
HILアプローチの利点
• 処理率の高いアルゴリズムの計算遅延を回避します。
• APIは、MATLAB環境内のすべてのFPGAデザインの制御、デバッグ、可視化、および検証に役立ちます。
• FPGAの結果のリードバックが、MATLABソフトウェアによってされ、後の解析および表示に使用されます。

必要なツールおよびコンポーネント
• MATLABソフトウェア
• DSP Builder for Intel®FPGAソフトウェア
• インテルQuartus Prime開発ソフトウェア
• インテルFPGA

注意: DSP Builder for Intel FPGAのインストレーション・バンドルには、System Console MATLAB APIが含まれています。

図-93: ホストとターゲット間のループをセットアップするハードウェア

関連情報
Hardware in the Loop from the MATLAB Simulink Environment white paper

8.12.1.サポートされているMATLAB APIコマンド

MATLAB環境から作業を実行し、System Consoleを使用してマスターとスレーブに対する読み出しや書き込みを行います。サポートされているMATLAB APIコマンドによって、System Consoleソフトウェアを起動する手間が省けます。サポートされているコマンドは次の通りです。
• SystemConsole.refreshMasters;
• M = SystemConsole.openMaster(1);
• M.write(type, byte address, data);
• M.read(type, byte address, number of words);
• M.close
例-31: MATLAB API スクリプトの例

```
SystemConsole.refreshMasters; %Investigate available targets
M = SystemConsole.openMaster(1); %Creates connection with FPGA target
%%%% User Application %%%%%%%%
....
M.write('uint32',write_address,data); %Send data to FPGA target
....
data = M.read('uint32',read_address,size); %Read data from FPGA target
....
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M.close; %Terminates connection to FPGA target
```

8.12.2. 高レベルのフロー

1. DSP Builder for Intel FPGA ソフトウェアをインストールすると、このフローをイネーブルするために必要なライブラリが使用可能になります。
2. Simulink と DSP Builder for Intel FPGA ライブラリを使用してデザインを構築します。DSP Builder for Intel FPGA は、Simulink デザインの HDL への変換に役立ちます。
3. デザインに Avalon-MM コンポーネントを含めます (DSP Builder for Intel FPGA では、Avalon-MM 以外のコンポーネントの移植ができます)。
4. デザインに Signals および Control ブロックを含めます。
5. 境界ブロックを使用して、合成可能な論理と合成不可能な論理を分離します。
6. プラットフォーム・デザイナー (旧: Qsys) に DSP システムを統合します。
7. インテル FPGA をプログラムします。
8. サポートされている MATLAB API コマンドを使用して、インテル FPGA と相互作用させます。

8.13. 廃止予定のコマンド

次の一覧表は、廃止予定のコマンドです。これらのコマンドは現在はサポートされていますが、System Console から削除される予定です。

注意: すべての dashboard_<name> コマンドは廃止され、インテル Quartus Prime 開発ソフトウェア・バージョン 15.1 以降から toolkit_<name> コマンドと置き換えられています。

<table>
<thead>
<tr>
<th>表 52. 廃止予定のコマンド</th>
</tr>
</thead>
<tbody>
<tr>
<td>コマンド</td>
</tr>
<tr>
<td>open_service</td>
</tr>
<tr>
<td>open_service</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

8.14. System Console を使用したデザインの解析とデバッグ 改訂履歴

次の改訂履歴はこの章に適用されます。
### インテル Quartus Prime のバージョン

<table>
<thead>
<tr>
<th>ドキュメント・バージョン</th>
<th>インテル Quartus Prime のバージョン</th>
<th>変更内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.05.07</td>
<td>18.0.0</td>
<td>次の廃止された項を削除しました。System Console を使用したボードの起動チュートリアル</td>
</tr>
<tr>
<td>2017.05.08</td>
<td>17.0.0</td>
<td>• 次のトピックを作成しました。Dashboard スクリプトの Toolkit API への変更</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• サービスの登録例を Toolkit API スクリプト例から削除し、対応するコードスニペットをツールキットの登録に追加しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ツールキット記述ファイルの作成の下の、toolkit 記述ファイルの例を移動しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Toolkit API GUI の例の toolkit ファイルを、toolkit 記述ファイルの例に変更しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Toolkit API の例を更新して、現在サポートされている構文を反映しました。</td>
</tr>
<tr>
<td>2016.10.31</td>
<td>16.1.0</td>
<td>• インテルにブランド名を変更しました。</td>
</tr>
<tr>
<td>2015.11.02</td>
<td>15.1.0</td>
<td>• Toolkit API の内容とコマンド形式の編集を行いました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Toolkit API デザイン例を追加しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• System Console の概要にグラフィックを追加しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ダッシュボードを廃止予定にしました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Quartus II のインスタンスを変更してインテル Quartus Prime にしました。</td>
</tr>
<tr>
<td>2015年10月</td>
<td>15.1.0</td>
<td>• Toolkit API の次の内容を追加しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‒ 必要な .toolkit および Tcl ファイル</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‒ ツールキットの登録と起動</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‒ ツールキットの検出および IP とツールキットの組み合わせ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‒ Toolkit API コマンド表</td>
</tr>
<tr>
<td>2015年5月</td>
<td>15.0.0</td>
<td>System Console をダウンロードしてスタンドアロンで起動する方法に関する情報を追加しました。</td>
</tr>
<tr>
<td>2014年12月</td>
<td>14.1.0</td>
<td>• MAX 10 デバイスで ADC Toolkit を使用するための概要および手順を追加しました。</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• システム検証用に System Console の現状を MATLAB/Simulink 環境を使用するための概要を追加しました。</td>
</tr>
<tr>
<td>2014年6月</td>
<td>14.0.0</td>
<td>ボードの立ち上げ、ダッシュボード・サービス、Nios II プロセッサー、デザインサービス、デバイスサービス、モニターサービス、バイストリーム・サービス、SLD サービス、および ISSP サービスのデザイン例を更新しました。</td>
</tr>
<tr>
<td>2013年11月</td>
<td>13.1.0</td>
<td>項を再編成しました。ブロック図、ワークフロー、SLD の概要、使用例、および Tcl スクリプトの例を含む高度な情報を追加しました。</td>
</tr>
<tr>
<td>2013年6月</td>
<td>13.0.0</td>
<td>Tcl コマンドテーブルを更新しました。ボード起動デザイン例を追加しました。SOPC Builder のコンテンツを削除しました。</td>
</tr>
<tr>
<td>2012年11月</td>
<td>12.1.0</td>
<td>コンテンツの再編成を行いました。</td>
</tr>
<tr>
<td>2012年8月</td>
<td>12.0.1</td>
<td>Transceiver Toolkit のコマンドを Transceiver Toolkit の章に移動しました。</td>
</tr>
<tr>
<td>2012年6月</td>
<td>12.0.0</td>
<td>メンテナンス・リリース、System Console の新しい機能を章に追加しました。</td>
</tr>
<tr>
<td>2011年11月</td>
<td>11.1.0</td>
<td>メンテナンス・リリース、System Console の新しい機能を章に追加しました。</td>
</tr>
<tr>
<td>2011年5月</td>
<td>11.0.0</td>
<td>メンテナンス・リリース、System Console の新しい機能を章に追加しました。</td>
</tr>
<tr>
<td>2010年12月</td>
<td>10.1.0</td>
<td>メンテナンス・リリース、Qsys の新しいコマンドとリファレンスを章に追加しました。</td>
</tr>
<tr>
<td>2010年7月</td>
<td>10.0.0</td>
<td>初版。以前は System Console ユーザーガイドとしてリリースされていましたが、現在は廃止されています。新しいコマンドを章に追加しました。</td>
</tr>
</tbody>
</table>

### 関連情報

#### ドキュメント・アーカイブ

以前のバージョンのインテル Quartus Prime ハンドブックは、文書アーカイブを検索してください。

### フィードバック

インテル Quartus Prime プロ・エディション ユーザーガイド: デバッグツール

203
9. トランシーバー・リンクのデバッグ

Transceiver Toolkit を使用すると、ボードデザインの高速シリアルリンクの最適化ができます。このために、ボード上で実行されているトランシーバー・リンクのリアルタイム制御、モニタリング、およびデバッグが行われます。

Transceiver Toolkit を使用すると、次のことが可能になります。

- トランスミッター・チャネルおよびレシーバーチャネルを制御し、トランシーバーの設定およびハードウェア機能を最適化します。
- ビット・エラー・レート (BER) のテストを複数のリンクのターゲット・データ・レートでの実行中に行います。
- 内部パターン・ジェネレーターおよびチェッカーを制御し、ループバック・モードをイネーブルします。
- 自動スイープテストを実行し、最良のフィジカル・メディア・アタッチメント (PMA) の設定を各リンクに対して特定します。
- インテル Stratix 10 デバイスでは、レシーバーの水平方向とおよび垂直方向のアイマシンの確認をテスト中に行います。
- 複数のデバイスの検証を複数のボードで同時に行います。

注意: Transceiver Toolkit は、System Console のフレームワークから実行します。

Transceiver Toolkit を起動するには、Tools > System Debugging Tools > Transceiver Toolkit の順でクリックします。もしくは、Tcl スクリプトを次のコマンドラインから実行します。

```
system-console --script=<name of script>
```

Transceiver Toolkit を使用したオンライン・デモで、高速リンクテストをいずれかのデザイン例で実行したものについては、アルテラのウェブサイトの Transceiver Toolkit Online Demo を参照してください。

関連情報

- トランシーバーのデザイン例およびリファレンス・デザイン
- Transceiver Toolkit Online Demo
- Transceiver Toolkit for インテル Arria 10 Devices (OTCVRKITA10)
  26 分オンラインコース
9.1. デバイスのサポート

インテル Quartus Prime プロ・エディション Transceiver Toolkit では、次のデバイスファミリーをサポートしています。

- インテル Arria 10
- インテル Cyclone 10 GX
- インテル Stratix 10 L-, H-, および E-タイル

特に記述のない限り、この章で説明されている機能の適用対象は、サポートされているすべてのデバイスです。

9.2. Channel Manager

Channel Manager は、Transceiver Toolkit のグラフィカル・コンポーネントです。Channel Manager を使用すると、トランシーバーのチャネルおよびリンクを設定・制御し、プログラム可能なアナログ設定を調整してリンクの信号の整合性を向上させることができます。Channel Manager は、System Console の Workspace 領域にあります。

Channel Manager を構成する 3 つのタブでは、コンポーネントの表示をスプレッドシート形式で行います。

- Transmitter Channels
- Receiver Channels
- Transceiver Links

各タブの列は、各デバイスのパラメーター設定によって異なります。

図 -94: 例：インテル Stratix 10 E-タイルでの Channel Manager の Receiver Channels タブ

| Columns vary depending on device family. Right-click a column header to hide. |
| Click to view all settings of selected channel or link |
Channel Manager の機能

Channel Manager によって次のアクションが簡略化されます。

• 設定のコピーと貼り付け - PMA 設定をチャンネルとの間でコピー、貼り付け、インポート、およびエクスポートします。

• 設定のインポートとエクスポート - PMA 設定をテキストファイルにエクスポートするには、Channel Manager で行を選択します。PMA 設定の適用をテキストファイルから行うには、Channel Manager で 1 つ以上の行を選択します。テキストファイルの PMA 設定が適用されるのは単一のチャンネルです。PMA 設定をテキストファイルからインポートすると、1 セットの PMA 設定の複製が、選択したすべてのチャンネルに対して行われます。

• テストの開始と停止 - Channel Manager では、テストの開始および停止が可能で、これにはチャンネルを右クリックします。2 つ以上の行を Channel Manager で選択すると、テストの開始または停止を複数チャンネルに対して行うことができます。

関連情報

• System Explorer ペイン (146 ページ)
• System Console の GUI (145 ページ)
• ユーザー・インターフェイス設定リファレンス (229 ページ)

9.2.1. チャネル表示モード

チャネル表示モードは次の 3 種類です。

• Current (デフォルト) - デバイスからの現在の値を表示します。青色の文字は設定が有効であることを示します。

• Min/Max - 自動スイープで使用する最小値と最大値を表示します。

• Best - 最後に完了した自動スイープの実行から最良のテスト値を示します。

注意: Transmitter Channels タブの表示は、Current 表示モードのみです。Transceiver Toolkit には、自動スイープテストを実行するためのレシーバーチャネルが必要です。

9.3. トランシーバー・デバッグ・フローの手順

以下の手順は、トランシーバーのデバッグの高度なプロセスです。これには Transceiver Toolkit を使用します。

1. デザインを変更してトランシーバーをデバッグ可能にします。
2. 変更したデザインを FPGA にロードします。
3. デザインを Transceiver Toolkit にロードします。
4. ハードウェア・リソースをリンクします。
5. ハードウェア接続を検証します。
6. トランシーバー・チャネルを識別します。
7. リンクトストを実行するか、PMA アナログ設定を制御します。
9.4. デザインを変更してトランシーバーをデバッグ可能にする

デザイン機能を有効にするには、1つ以上のトランシーバー インテル FPGA IP Core のパラメーターを変更してください。

関連情報
デバイスのサポート (205 ページ)

9.4.1. トランシーバー IP コアのデバッグに係るパラメーター

インテル Quartus Prime プロ・エディションがサポートするすべてのデバイスにおいて、トランシーバー PHY インテル FPGA IP の次のパラメーターをイネーブルする必要があります。

表 53. トランシーバー PHY IP のデバッグをイネーブルするパラメーター

<table>
<thead>
<tr>
<th>パラメーター</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Dynamic Reconfiguration</td>
<td>デバイスをパワーダウンすることなくトランシーバー・チャネルと PLL の動作が変更可能です。</td>
</tr>
<tr>
<td>Enable Altera Debug Master Endpoint (AMDE)</td>
<td>システムコンソールを介したトランシーバーおよび PLL レジスターへのアクセスが可能です。デ ザインを再コンパイルする際、インテル Quartus Prime 開発ソフトウェアは ADME デバッグ・ ファブリックとエンベッド・ロジックを挿入します。</td>
</tr>
<tr>
<td>Enable control and status registers</td>
<td>エンベッド・デバッグを介して PHY インターフェイスにステータス信号の読み出しとコントロー ル信号の書き込みをするために、ソフトレジスターをイネーブルします。</td>
</tr>
<tr>
<td>Enable PRBS Soft Accumulators</td>
<td>ハード PRBS ジェネレーターおよびチェッカーを使用する際、PRBS ビットおよびエラーの累積を 実行するためにソフトロジックを有効にします。</td>
</tr>
</tbody>
</table>

インテル Stratix 10 L- および H-タイルデバイスをターゲットとするデザインでは、他の インテル FPGA IP のデバッグ機能も有効にする必要があります。

表 54. インテル Stratix 10 L- および H-タイルデバイスでデバッグをイネーブルするトランシーバー IP およびパラメーター

<table>
<thead>
<tr>
<th>インテル FPGA IP</th>
<th>イネーブルするパラメーター</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transceiver ATX PLL</td>
<td>• Enable Dynamic Reconfiguration</td>
</tr>
<tr>
<td>CMU PLL</td>
<td>• Enable Altera Debug Master Endpoint</td>
</tr>
<tr>
<td>PLL</td>
<td></td>
</tr>
</tbody>
</table>

関連情報
Debug IP コアのインスタンス化とパラメーター化 (211 ページ)

9.4.1.1. Parameter Editor での インテル Arria 10 および インテル Cyclone 10 GX  Transceiver IP のパラメーターのデバッグ

次のスクリーンショットで表しているパラメーターをイネーブルして、インテル Arria 10 および インテル Cyclone 10 GX デザインでトランシーバーをデバッグしてください。
図 -95: インテル Arria 10 および インテル Cyclone 10 GX Transceiver Native PHY IP コアのダイナミック・リコンフィグレーション・パラメーター

図 -96: インテル Arria 10 および インテル Cyclone 10 GX Transceiver ATX PLL コアのダイナミック・リコンフィグレーション・パラメーター

インテル Arria 10 デバイスのダイナミック・リコンフィグレーション・パラメーターの詳細については、インテル Arria 10 トランシーバー PHY ユーザーガイドを参照してください。

インテル Cyclone 10 GX デバイスのダイナミック・リコンフィグレーション・パラメーターの詳細については、インテル Cyclone 10 GX Transceiver PHY User Guide を参照してください。

関連情報

- Dynamic Reconfiguration Parameters
  インテル Arria 10 トランシーバー PHY ユーザーガイド

- Reconfiguration Interface and Dynamic Reconfiguration
  インテル Cyclone 10 GX Transceiver PHY User Guide
9.4.1.2. Parameter Editor での インテル Stratix 10 L- および H-タイル Transceiver IP のパラメーターのデバッグ

次のスクリーンショットで表しているパラメーターをイネーブルして、インテル Stratix 10 L- および H-タイルデザインでトランシーバーをデバッグしてください。

図 -97: インテル Stratix 10 L- および H-タイル Transceiver Native PHY IP コアのダイナミック・リコンフィギュレーション・パラメーター

図 -98: インテル Stratix 10 L- および H-タイル Transceiver ATX PLL IP コアのダイナミック・リコンフィギュレーション・パラメーター
図 -99: インテル Stratix 10 L- および H-タイル Transceiver CMU PLL IP コアのダイナミック・リコンフィグレーション・パラメーター

図 -100: インテル Stratix 10 L- および H-タイル Transceiver fPLL IP コアのダイナミック・リコンフィグレーション・パラメーター

インテル Stratix 10 L- および H-タイルデバイスのダイナミック・リコンフィグレーション・パラメーターの詳細については、インテル Stratix 10 L- および H-タイル Transceiver PHY User Guide を参照してください。

関連情報
Dynamic Reconfiguration Parameters
インテル Stratix 10 L- および H-タイル Transceiver PHY User Guide

9.4.1.3. Parameter Editor での インテル Stratix 10 E-タイル Transceiver IP のパラメーターのデバッグ

次の図で示すパラメーターをインーブルして、インテル Stratix 10 E-タイルデザインでトランシーバーをデバッガしてください。
図 -101: インテル Stratix 10 E-タイル Transceiver IP のトランシーバー・デバッグ用パラメーター

インテル Stratix 10 E-タイルデバイスのダイナミック・リコンフィグレーション・パラメーターの詳細については、インテル Stratix 10 E-タイル Transceiver PHY User Guide を参照してください。

関連情報
Dynamic Reconfiguration Parameters
インテル Stratix 10 E-タイル Transceiver PHY User Guide

9.4.1.4. Debug IP コアのインスタンス化とパラメーター化

この設定をアクティブ化するタイミングは、これらのコンポーネントを最初にインスタンス化するとき、または初期コンパイル後にインスタンスを変更するときです。

どの IP コアを変更する必要があるかについては、Debug Parameters for Transceiver IP Cores を参照してください。

各トランシーバー IP コアに次の手順を実行します。
1. Project Navigator の IP Components タブで IP インスタンスを右クリックし、Edit in Parameter Editor をクリックします。
2. デバッグ設定をオンにします。
   Parameter Editor での Transceiver IP のデバッグ・パラメーターのスクリーンショットを参照してください。
3. 該当する場合は、デバッグログジックに必要なリファレンス信号を接続します。
   ADME には、クロック信号とリセット信号の接続が必要です。周波数要件の詳細については、対応する Transceiver PHY ユーザーガイドのポートおよびパラメーターを参照してください。
4. Generate HDL をクリックします。

デザイン内のすべての IP のパラメーターをイネーブルした後、プロジェクトを再コンパイルします。

関連情報
• トランシーバー IP コアのデバッグに関係するパラメーター (207 ページ)
• Ports and Parameters
   インテル Arria 10 トランシーバー PHY ユーザーガイド
• Ports and Parameters
   インテル Cyclone 10 GX Transceiver PHY User Guide
• Ports and Parameters
   インテル Stratix 10 L- およびH-タイル Transceiver PHY User Guide
9.5. インテル FPGA へのデザインのプログラミング

デザインにデバッグ・コンポーネントを含めて、プログラミング・ファイルのコンパイルと生成を実行すると、デザインのプログラミングをインテル FPGA することができます。

関連情報

Programming Intel FPGA Devices
インテル Quartus Prime プロ・エディション Programmer ユーザーガイド 内

9.6. Transceiver Toolkit へのデザインのロード

ロード時に FPGA のプロジェクトでのプログラミングがすでにされている場合、Transceiver Toolkit では、ツールキット内のターゲット・ハードウェアにデザインを自動的にリンクします。ツールキットでは、同じチャネルのトランスミッターとレシーバーとの間のリンクを自動検出します。

デバイスのロード前に、ハードウェアが接続されていることを確認してください。デバイスと JTAG 接続の表示は、System Explorer ペインの Device フォルダーおよび Connections フォルダーにされます。

デザインを Transceiver Toolkit にロードするには、次の手順を実行します。

1. System Console で、File > Load Design の順でクリックします。
2. .sof プログラミング・ファイルをトランシーバー・デザインに対して選択します。

プロジェクトのロードが完了すると、System Explorer のパネルの designs フォルダーおよび design instances フォルダーに、デザイン名、System Console と通信可能なデザイン内のブロックなどのデザインに関する情報が表示されます。

関連情報

System Explorer ペイン (146 ページ)

9.7. ハードウェア・リソースのリンク

ハードウェア・リソースをリンクすると、ロードしたプロジェクトがターゲット FPGA にマッピングされます。複数のデザイン・プロジェクトを複数の FPGA ににロードする場合、リンクによってどのプロジェクトがそれぞれの FPGA にあるかが表示されます。ツールキットでは、接続したハードウェアとデザインを自動検出します。また、System Explorer では、接続されたハードウェア・リソースにデザインを手動でリンクすることも可能です。

複数のインテル FPGA ボードを使用している場合、テストのセットアップと同じデザインにリンクされた複数のデバイスで実行することができます。このセットアップが便利なのは、リンクテストを 2 つの別のデバイスのトランスミッターとレシーバーとの間で実行する場合です。複数のインテル Quartus Prime プロジェクトをロードし、異なるシステム間でリンクすることも可能です。個別の無関係のシステムのテストは、単一の インテル Quartus Prime インスタンスで実行できます。
9.7.1. 1つのデザインを1つのデバイスに対してリンクする

1つのデザインを1つのデバイスに対して1つのインテルFPGAダウンロード・ケーブルでリンクするには、次を実行します。
1. インテル Quartus Prime プロジェクト用にデザインをロードします。
2. デザインが自動リンクされていない場合は、各デバイスを適切なデザインにリンクします。
3. テストするデバイス上のチャネル間のリンクを作成します。

9.7.2. 2つのデザインを2つのデバイスに対してリンクする

2つのデザインのリンクを1つのインテル FPGA ダウンロード・ケーブルケーブルで接続された同じボード上の2つの別々のデバイスに対して行うには、次を実行します。
1. 必要なすべてのインテル Quartus Prime プロジェクト・ファイルにデザインをロードします。
2. デザインが自動リンクされていない場合は、各デバイスを適切なデザインにリンクします。
3. 2つのデバイスの接続を接続します。
4. JTAG チェーン上の2番目のデバイスを2番目のデザインにリンクさせます。(デザインが自動リンクする場合は除く。)
5. テストするデバイスのチャネル間のリンクを作成します。

9.7.3. 1つのデザインを2つのデバイス上でリンクする

同一のデザインを2つの別々のデバイス上でリンクするには、次を実行します。
1. Transceiver Toolkit で、両方のデバイスで使用している .sof を開きます。
2. 最初のデバイスをこのデザイン・インスタンスにリンクします。
3. 2つのデバイスをこのデザインにリンクします。
4. テストするデバイスのチャネル間のリンクを作成します。

9.7.4. デザインとデバイスを別々のボード上でリンクする

2つのデバイスのリンクを、2つの異なるデバイスに対して、別々のインテル FPGA ダウンロード・ケーブルに接続している個別のボード上で行うには、次を実行します。
1. 必要なすべてのインテル Quartus Prime プロジェクト・ファイルにデザインをロードします。
2. デザインが自動リンクされていない場合は、各デバイスを適切なデザインにリンクします。
3. テストするデバイスのチャネル間のリンクを作成します。
4. 2つのインテル FPGA ダウンロード・ケーブルに接続しているデバイスのリンクを2つのデバイスに対して行います。
5. テストするデバイスのチャネル間のリンクを作成します。

9.7.5. ハードウェア接続の検証

リンクを作成したら、ハードウェア上でのチャネルの正しい接続と適切なループバックを確認します。この予防措置によってワークフローの時間を節約できます。
ツールキットを使用して、データパターンを正確に送受信します。
1. Receiver タブで RX CDR locked to Data が Locked に設定されていることを確認します。
図 -104: RX CDR Locked to Data

2. トランスミッター・チャネルでジェネレーターを起動します。
3. レシーバー・チャネルでチェッカーを起動します。
4. Lock to Data があることと、上記の 2 者間の Bit Error Rate が非常に低いかゼロであることを確認します。

トランスミッターとレシーバー間の通信を検証したら、2 つのトランシーバーの間リンクを作成し、Auto Sweep テストと Eye Viewer (2) テストを実行します。

9.8. トランシーバー・チャネルの特定

Transceiver Toolkit でチャネルの検出が正しく行われているかどうかを確認します。レシーバーとトランシミッターでトランシーバー・チャネルを共有している場合は、ツールキットによってチャネルが特定されます。

Transceiver Toolkit では、トランスミッター・チャネルとレシーバーチャネルの特定と表示を、Channel Manager の Transmitter Channels タブおよび Receiver Channels タブ上で行います。また、手動でトランシーバー・チャネル内のトランスミッターとレシーバーを特定し、この 2 つの間にリンクを作成してそれをテストに使用することもできます。

9.8.1. トランシーバー・チャネルの制御

トランスミッターまたはレシーバーの設定の調整またはモニタリングをチャネル動作中に行うには、次のように行います。

- Transmitter Channels タブで Control Transmitter Channel をクリックします。
- Receiver Channels タブで Control Receiver Channel をクリックします。
- Transceiver Links タブで Control Receiver Channel をクリックします。

例えば、トランシーバー・リンク全体にデータパターンを送信し、受信したデータの信号品質をレポートします。

(2) Eye Viewer は インテル Stratix 10 デバイスにのみ使用可能
9.9. トランシーバー・リンクの作成

リンクを作成すると、物理的に接続する Transmitter チャネルと Receiver チャネルが指定されます。トランシーバーツールキットによってリンクが自動作成されるのは、レシーバーとトランスミッターによってトランシーバー・チャネルが共有された場合です。トランスミッター・チャネルとレシーバーチャネル間のリンクは、手動でも作成・削除ができます。

トランシーバー・リンクを作成するには、次を実行します。
1. Channel Manager で Setup をクリックします。
2. 制御するジェネレーターとチェッカーを選択します。
3. 制御するトランスミッターとレシーバーの組み合わせを選択します。
4. Create Transceiver Link をクリックします。
5. Close をクリックします。

Transceiver Toolkit では、リンク名を自動生成しますが、このリンク名をより短く分かりやすい名前に変更するには、Link Alias セルに入力します。

9.10. リンクトストの実行

デバッグ用のトランシーバー・チャネルを特定したら、リンクテストを実行します。Transceiver Links タブを使用してリンクテストを制御します。

リンクテストの実行時、チャネルの色のハイライ트によってテストの状態が分かります。

表 55. チャネルの色のハイライト

<table>
<thead>
<tr>
<th>色</th>
<th>トランスミッター・チャネル</th>
<th>レシーバーチャネル</th>
</tr>
</thead>
<tbody>
<tr>
<td>赤色</td>
<td>チャネルが閉じているか、ジェネレーター・クロックが動作しています。</td>
<td>チャネルが閉じているか、チェッカークロックが動作していません。</td>
</tr>
<tr>
<td>緑色</td>
<td>ジェネレーターによるパターンの送信中です。</td>
<td>チェッカーによるチェック中です。データパターンはロックされています。</td>
</tr>
<tr>
<td>中間色（背景と同色）</td>
<td>チャネルが閉じており、ジェネレーター・クロックが動作中です。ジェネレーターによるパターンの送信はされていません。</td>
<td>チャネルが閉じており、チェッカークロックが動作中です。チェッカーによるチェックは行われていません。</td>
</tr>
<tr>
<td>黄色</td>
<td>該当なし</td>
<td>チェッカーによるチェック中です。データパターンはロックされています。</td>
</tr>
</tbody>
</table>

9.10.1. BER テストの実行

BER テストはシグナル・インテグリティーの評価に役立ちます。トランシーバー・パラメーターが異なれば、BRT 値も異なります。

BER テストは、Transceiver Link Basic タブから実行します。
図 -105: 例 : Control Transceiver Link タブ（インテル Arria 10 デバイス）

BER テストをトランシーバー・リンクで実行するには、次の手順を実行します。
1. テストするトランシーバー・リンクを Channel Manager で選択し、Control Transceiver Link をクリックします。

図 -106: Control Transceiver Link

Basic タブが開きます。
2. Test pattern で PRBS テストパターンを指定します。
3. Reconfiguration 設定の指定を Transmitter チャネルおよび Receiver チャネルに対して行います。
4. Start をクリックしてテストを実行します。
ビット・エラー・レートは、Checker セクションの Receiver Channel カラムに表示されます。

5. エラーカウンターをリセットするには、Reset をクリックします。

6. テストを停止するには、Stop をクリックします。

7. 別のリコンフィグレーション・パラメーターでエラーレートをテストする場合は、パラメーターを変更してから Start をクリックします。

リコンフィグレーション可能なトランシーバーのパラメーターのリストについては、ユーザー・インターフェイス設置リファレンスを参照してください。

関連情報
ユーザー・インターフェイス設置リファレンス (229 ページ)

9.10.2. リンク最適化テスト

Transceiver Toolkit の自動スイープテストでは、PMA 範囲を自動スイープし、トランシーバー設定を決定して最良のシグナル・インテグリティが得られるようにします。このツールキットを使用すると、テストの実行履歴の保存と、最良の PMA 設定の記録がされます。

9.10.3. Eye Viewer テストの実行

Transceiver Toolkit では、インテル Stratix 10 デバイスのアイテストの実行をサポートしています。インテル Stratix 10 L- および H- タイプデバイスの場合、アイ測定は、チャネルの内部ループバック中に行うことができます。

Eye Viewer ツールを使用すると、アイテスト、ビットエラーのモニタリングおよび位相オフセットデータの設定および実行が可能になります。

1. レシーバーチャネルあるいはトランシーバー・リンク・チャネルで行をクリックしてから、Receiver/Link Eye Viewer をクリックします。

図 -107: Eye Viewer をクリックして開く

Advanced タブが開きます。テストモードの設定が Eye Viewer になっています。

2. テスト条件の設定を希望に合わせて行います。
自動スイープテストを実行して、許容可能な設定を判断します。

3. Eye Viewer settings で、水平と垂直のステップ間隔を指定します。

4. Start をクリックします。
   テスト実行中、Eye Viewer ツールではチャネルの現在の設定を収集し、その設定を使用して位相スイープを開始します。

5. Eye Viewer ステータスをチェックして、テストが完了したかどうかを確認します。

6. PMA 設定を変更して別のテストを実行するには、Stop, Reset の順でクリックし、Eye Viewer テストを再開します。

実行が完了すると Heat Map ウィンドウが開いて、テストの実行結果が表示されます。

図-108: 例：インテル Stratix 10 E-タイルデバイスのデザインに対するアイテストの結果
このアイ・ダイアグラムは、NRZ エンコーディングに対応しています。PAM4 エンコーディングのトランシーバー用のアイ・ダイアグラムの特徴に関する情報については、AN 835: PAM4 Signaling Fundamentals を参照してください。

---

注意: インテル Quartus Prime のバージョン 18.0 以降では、リンク・テスト・ファイルのインポートを以前のリリースからすることはできません。同様に、インテル Quartus Prime 18.0 およびそれ以降のバージョンで生成されたレポートのインポートは、旧バージョンの Quartus に対してはできません。

Advanced タブで Create Report をクリックしてデータを表示し、テーブル・フォーマットへデータをエクスポートします。
図 - 109: 例：インテル Stratix 10 E-タイルデバイスのデザインのアイ・テスト・レポート

関連情報
- AN 835: PAM4 Signaling Fundamentals
- ユーザー・インターフェース設定リファレンス (229 ページ)

9.11. PMA アナログ設定の制御

Transceiver Toolkit を使用すると、PMA アナログ設定の直接制御がリンク実行中にできます。各パラメーターの詳細については、対応するデバイスの PHY ユーザーガイドを参照してください。

PMA のアナログ設定を制御するには、次の手順を実行します。
1. Channel Manager で Setup をクリックします。
2. Transmitter Channels タブで、ジェネレーターのないトランスミッターを指定し、Create Transmitter Channel をクリックします。
3. Receiver Channels タブで、ジェネレーターのないレシーバーを指定し、Create Receiver Channel をクリックします。
4. Transceiver Links タブで、制御するトランスミッターとレシーバーを選択し、Create Transceiver Link をクリックします。
5. Close をクリックします。
6. Control Receiver Channel, Control Transmitter Channel, あるいは Control Transceiver Link をクリックして、PMA 設定の直接制御を行います。

9.11.1. インテル Arria 10 および インテル Cyclone 10 GX の PAM 設定

次の図で示すのは、インテル Arria 10 および インテル Cyclone 10 GX デバイスの PMA アナログ設定です。
図-110: トランスミッターチャネルのPMA設定（インテルArria 10およびインテルCyclone 10 GX）

<table>
<thead>
<tr>
<th>Test pattern:</th>
<th>PRBS15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generator mode:</td>
<td>Hard PRBS</td>
</tr>
</tbody>
</table>

Transmitter channel: `TxPHY_80000_address_1_3`

- **Transceiver**
  - Channel address: 3
  - Data rate: 10625.00 Mbps
  - PLL refclk frequency: 0.00 MHz
  - TX/CMU PLL status: N/A

- **Reconfiguration**
  - Channel address: 3
  - \( V_{OD} \) control: 31
  - Pre-emphasis 1st post-tap: 4
  - Pre-emphasis pre-tap: 3
  - Pre-emphasis 2nd post-tap: 3
  - Pre-emphasis 2nd pre-tap: 4

- **Generator**
  - Preamble word: 0
  - Number of preamble beats: 1
  - Use preamble upon start

[Start]  [Stop]  [Inject Error]
図 -111: レシーバーチャネルの PMA 設定（インテル Arria 10 および インテル Cyclone 10 GX）

![設定画面のスクリーンショット](image_url)

### トランシーバー・リンクのデバッグ

UG-20139 | 2018.09.24
図 -112: トランシーバー・リンクの PMA 設定（インテル Arria 10 および インテル Cyclone 10 GX）

関連情報
- Debug IP コアのインスタンス化とパラメータ化 (211 ページ)
- PMA Parameters
  インテル Arria 10 トランシーバー PHY ユーザーガイド
- PMA Parameters
  インテル Cyclone 10 GX Transceiver PHY User Guide
9.11.2. インテル Stratix 10 L- および H-タイルの PMA 設定

次の図で示すのは、インテル Stratix 10 L- および H-タイルデバイスの PMA 設定です。

図 -113: トランスミッター・チャネルの PMA 設定 ( インテル Stratix 10 L- および H-タイル)
図-114: レシーバーチャネルの PMA 設定（インテル Stratix 10 L- および H-タイル）
図 -115: トランスミッター・リンクの PMA 設定（インテル Stratix 10 L- および H-タイル）

関連情報
- Debug IP コアのインスタンス化とパラメーター化
- Analog PMA Settings Parameters
  インテル Stratix 10 L- および H- タイル Transceiver PHY User Guide

9.11.3. インテル Stratix 10 E- タイレの PMA 設定
次の図で示すのは、インテル Stratix 10 E- タイレデバイスの PMA 設定です。
図-116: インテル Stratix 10 E-タイルトランシーバー・チャネルの PMA 設定

Transceiver Toolkit による Rules Based Configuration の有効性チェックのサポートは、インテル Stratix 10 E-タイルデバイスでは提供されていません。正しいパラメーターの組み合わせは、赤色で表示されます。トランシーバ設定の有効性を確認するには、インテル Stratix 10 E-タイル Transceiver PHY User Guide を参照してください。

テストパターン: PRBS31
generator mode: Hard PRBS

Transmitter channel

- Channel address: 0
- Data rate: 57800.00 Mbps

Reconfiguration

- Channel address: 0
- VCcD control:
- Pre-emphasis 1st post-tap: 2
- Pre-emphasis 1st pre-tap: 2
- Pre-emphasis 2nd pre-tap: 0
- Pre-emphasis 3rd pre-tap: 0

Generator

Start Stop Inject Error
図 -117: インテル Stratix 10 E-タイルレシーバーチャネルの PMA 設定

図 -118: インテル Stratix 10 E-タイルトランシーバー・リンクの PMA 設定
9.12. ユーザー・インターフェイス設定リファレンス

Transceiver Toolkitのユーザー・インターフェイスには、次の設定が含まれています。

表 56. Transceiver Toolkit コントロール・ペインの設定

<table>
<thead>
<tr>
<th>設定</th>
<th>説明</th>
<th>デバイスファミリー</th>
<th>コントロール・ペイン</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>チャネルに対して選択する名前です。</td>
<td></td>
<td>Transmitter ペイン</td>
</tr>
<tr>
<td></td>
<td>サポートされているすべてのデバイスファミリー</td>
<td></td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td>Auto Sweep status</td>
<td>現在および最良のテスト済みビット、エラー、ビット・エラーレート、および現在のAuto Sweepテストのケースカウントを報告します。</td>
<td></td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td>Bit error rate (BER)</td>
<td>チェッカーの最後のリセット以降にテストされたビットで割ったエラーの数を報告します。</td>
<td></td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td>Channel address</td>
<td>トランシーバー・チャネルの論理アドレス番号です。</td>
<td></td>
<td>Transmitter ペイン</td>
</tr>
<tr>
<td></td>
<td>トランシーバー・チャネルの論理アドレス番号です。</td>
<td></td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td>CTLE AC Gain</td>
<td>レシーバーのContinuous Time Linear Equalization (CTLE) AC Gainを指定します。ACゲインのフルレンジは、ピーク周波数(設定0)で約-2 dBからピーク周波数(設定15)での+10 dBになります。</td>
<td>英テスタス 10 L-およびH-タイル</td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td>CTLE EQ Gain</td>
<td>レシーバーのCTLE EQ Gainを指定します。EQゲインのフルレンジは、ピーク周波数(設定0)での0 dBからピーク周波数(設定47)での-16 dBになります。</td>
<td>英テスタス 10 L-およびH-タイル</td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td>CTLE DFE mode</td>
<td>アダプティブ CTLE、アダプティブ VGA、1タップ・アダプティブDFE</td>
<td></td>
<td>英テスタス Arria 10および インテル Cyclone 10 GX</td>
</tr>
<tr>
<td></td>
<td>マニュアル CTLE、マニュアル VGA、DFEオフ</td>
<td></td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td></td>
<td>アダプティブ CTLE、アダプティブ VGA、オール・タップ・アダプティブDFE</td>
<td></td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td></td>
<td>アダプティブ CTLE、アダプティブ VGA、オール・タップ・アダプティブDFE</td>
<td></td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td>Data rate</td>
<td>プロジェクト・ファイルに表示されるチャネルのデータレート、または周波数検出器で測定するデータレートです。周波数検出器を使用するには、Data Pattern Checker IPコアまたはData Pattern Generator IPコアのEnable Frequency Counterをオンにし、IPコアを再生し、デザインを再コンパイルします。測定されるデータレートは、プロジェクト・ファイルに表示されるAvalon®管理クロック周波数に依存しています。設定を変更し、再度データレートをサンプリングする場合は、Data rateの横にあるRefreshボタンをクリックしてください。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Transmitter ペイン</td>
</tr>
<tr>
<td></td>
<td>すべてのデバイスファミリー</td>
<td></td>
<td>Receiver ペイン</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>設定</th>
<th>説明</th>
<th>デバイスファミリー</th>
<th>コントロール・ペイン</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC gain</td>
<td>周波数スペクトラム間の入力信号にブーストを等分にブーストします。</td>
<td>インテル Arria 10 および Cyclone 10 GX</td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td>DFE mode</td>
<td>入力信号に対するディジタルフィードバック・イコライゼーション (DFE)です。</td>
<td>インテル Arria 10 および Cyclone 10 GX</td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td></td>
<td><strong>デバイス</strong></td>
<td><strong>値</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>インテル Arria 10 および Cyclone 10 GX</td>
<td>1-11</td>
<td></td>
</tr>
</tbody>
</table>

インテル Arria 10 デバイスでは、DFE モードは Off, Manual, Adaptation Enabled です。Adaptation Enabled モードの DFE では、最適なタップ値を自動検索します。

### E-Tile Adaptation Mode
レシーバーイコライザーパラメーターのチューニング・モードを指定します。
- Continuous Adaptation
- One-Time Adaptation
注: One-Time Adaptation Mode を実行する場合、Adaptation Enabled モードの DFE を実行するモードは Manual または Adaptation Enabled モードです。

インテル Stratix 10 E-Tile Receiver ペイン

### Equalization control
入力信号の高周波ゲインをブーストして、物理媒体のローパスフィルター効果を補償します。このオプションを DFE と併用する場合、DFE を使用するのでは Manual モードあるいは Adaptation Enabled モードです。

インテル Arria 10 および Cyclone 10 GX Receiver ペイン

### Equalization mode
インテル Arria 10 デバイスの場合、Equalization Mode の設定は、Manual あるいは Triggered にします。

インテル Arria 10 および Cyclone 10 GX Receiver ペイン

### Error rate limit
エラーレートの制限をオン/オフします。Start checking after で設定されるビット数をソールキットで待機してから、次の 2 回のチェックのピタ・エラーレート (BER) を調べます。

Bit error rate achieves below では、ビット・エラーレートの上限を設定します。そのエラーレートが設定されたエラーレートよりも良い場合、テストは終了します。

Bit error rate exceeds では、ビット・エラーレートの下限を設定します。そのエラーレートが設定されたエラーレートよりも悪い場合、テストは終了します。

サポートされるすべてのデバイスファミリー

### Generator/Checker mode
BER テストに対して、Data pattern checker または Serial bit comparator のいずれかを指定します。

Serial bit comparator をイネーブルすると、Data Pattern Generator によって PRBS パターンが送信されますが、シリアル・ビット・コンパレーターではそのパターンをチェックします。

Bypass mode で Start をクリックすると、シリアル・ビット・コンパレーターでカウントが開始されます。

BER テストの場合
- インテル Arria 10 デバイスでは、Data Pattern Checker と Hard PRBS をサポートしています。

サポートされているすべてのデバイスファミリー

### Increase test range
選択したコントロールのセットに対して、テストスパンを広げます。これには、最小を 1 単位増加し、最大を 1 単位減少させます。

サポートされるすべてのデバイスファミリー

Receiver ペイン

continued...
### 設定

<table>
<thead>
<tr>
<th>設定</th>
<th>説明</th>
<th>デバイスファミリー</th>
<th>コントロール・ペイン</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum tested bits</td>
<td>各テストの反復でテストされるビットの最大数を設定します。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Receiverペイン</td>
</tr>
<tr>
<td>Number of bits tested</td>
<td>チェッカーの最後のリセット以降にテストされるビットの数を指定します。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Receiverペイン</td>
</tr>
<tr>
<td>Number of error bits</td>
<td>チェッカーの最後のリセット以降に遭遇するビットの数を指定します。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Receiverペイン</td>
</tr>
<tr>
<td>PLL refclk freq</td>
<td>プロジェクト・ファイルに表示されるチャネル・リファレンス・クロック周波数、または測定データテーブルから算出されたリファレンス・クロック周波数です。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Transmitterペイン Receiverペイン</td>
</tr>
<tr>
<td>Populate with</td>
<td>Advancedパネルを右クリックしてデバイスの現在の値をロードして開始点とするか、または自動スイープによって決定される最適な設定を最初にロードします。インテルQuartus Prime開発ソフトウェアでは、Transmitter設定とReceiver設定のドロップダウン・リストで指定した値を目適的に適用します。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Receiverペイン</td>
</tr>
<tr>
<td>Pre-emphasis</td>
<td>このプログラム可能なモジュールでは、各送信バッファー信号の送信データの高周波をブーストします。この動作によって、伝送媒体の潜在的な減衰を防止します。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Transmitterペイン</td>
</tr>
<tr>
<td>Receiver channel</td>
<td>選択した受信チャネルの名前を指定します。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Receiverペイン</td>
</tr>
<tr>
<td>Refresh Button</td>
<td>.pofファイルをロードした後、ダイナマック・リコンフィグレーション実行後に新しい設定をレジスターからロードします。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Transmitterペイン Receiverペイン</td>
</tr>
<tr>
<td>Reset</td>
<td>現在のテストをリセットします。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Receiverペイン</td>
</tr>
<tr>
<td>Rules Based Configuration (RBC) validity checking</td>
<td>以前の設定に基づいて、無効な設定の組み合わせを赤色で表示します。表示場所は、Transmitter settingsとReceiver settingsの各リストです。このオプションをイネーブルすると、設定が赤色で表示され、現在の組み合わせが無効であることが示されます。この動作によって、デザインにコンパイルできない無効な設定を手動でテストすることがになり、長時間にわたってデバイスを無効なモードに設定することなくなくなるため、回路に損傷を与える可能性を防ぐことができます。</td>
<td>インテル Arria 10 インテル Cyclone 10 GX インテル Stratix 10 L-およびH-タイル</td>
<td>Receiverペイン</td>
</tr>
<tr>
<td>Run length</td>
<td>テスト実行用にカバレッジ・パラメーターを設定します。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Transmitterペイン Receiverペイン</td>
</tr>
<tr>
<td>RX CDR PLL status</td>
<td>レシーバーがLock-To-Reference（LTR）モードであることを表示します。Auto-modeレシーバーデータがロック不可能な場合、この信号がLTDモードで切り替わるのは、CDRがデータにロックされている場合です。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Receiverペイン</td>
</tr>
</tbody>
</table>

continued...
9. トランシーバー・リンクのデバッグ

<table>
<thead>
<tr>
<th>設定</th>
<th>説明</th>
<th>デバイスファミリー</th>
<th>コントロール・ペイン</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX CDR data status</td>
<td>レシーバーが Lock-To-Data (LTD) モードであることを表示します。Auto-mode でデータがロック不可能な場合、この信号が High に維持されるのは、データにロックされて切り替えがない場合です。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Receiver ペイン</td>
</tr>
<tr>
<td>Serial loopback enabled</td>
<td>バッファーの前にシリアル・ループバックを挿入し、デバイスの同じ物理チャネル上のトランスミッターとレシーバーの組み合わせにリンクを形成できるようにします。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Transmitter ペイン, Receiver ペイン</td>
</tr>
<tr>
<td>Start</td>
<td>チャネル上のパターン・ジェネレーターまたはチェックャーを開始して、着信データを検証します。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Transmitter ペイン, Receiver ペイン</td>
</tr>
<tr>
<td>Stop</td>
<td>パターンの生成とチャネルのテストを停止します。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Transmitter ペイン, Receiver ペイン</td>
</tr>
<tr>
<td>Test pattern</td>
<td>トランスミッター・チャネルによって送信されるテストパターンです。</td>
<td>サポートされているすべてのデバイスファミリー</td>
<td>Transmitter ペイン, Receiver ペイン</td>
</tr>
</tbody>
</table>

デバイスファミリー  | 使用可能なテストパターン |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>インテル Arria 10 および インテル Cyclone 10 GX</td>
<td>PRBS9, PRBS15, PRBS23, PRBS31,</td>
</tr>
<tr>
<td>インテル Stratix 10 L-および H-タイル</td>
<td>PRBS7, PRBS9, PRBS15, PRBS23, および PRBS31,</td>
</tr>
</tbody>
</table>
| インテル Stratix 10 E-タイル | • NRZ: PRBS7, PRBS9, PRBS11, PRBS13, PRBS15, PRBS23, および PRBS31,  
  • PAM4: PRBS7Q, PRBS9Q, PRBS11Q, PRBS13Q, PRBS15Q, PRBS23Q, および PRBS31Q, |

| Time limit | タイムリミットの単位と値を指定し、最大の境界タイムリミットを各テストのイタレーションに持たせます。 | サポートされているすべてのデバイスファミリー | Receiver |
| Transmitter channel | 選択したトランスミッター・チャネルの名前を指定します。 | サポートされているすべてのデバイスファミリー | Transmitter ペイン |
| TX/CMU PLL status | トランスミッター・チャネル PLL をリファレンス・クロックにロックさせるかどうかを指定します。 | サポートされているすべてのデバイスファミリー | Transmitter ペイン |
| Use preamble upon start | オンにすると、プリアンブル・ワードの送信がテストパターンの前に行われます。オフにすると、テストパターンがすぐに送信されます。 | サポートされているすべてのデバイスファミリー | Transmitter ペイン |

continued...
設定 | 説明 | デバイスファミリー | コントロール・ペイン
---|---|---|---
VGA | 可変ゲインアンプ (VGA) では、信号振幅を増幅し、一定の電圧振幅を保証します。これは、データがサンプリングのために Clock Data Recovery (CDR) ブロックに入る前に行われます。この割り当てによって VGA 出力電圧振幅が制御されます。許容値は 0 から 7 までです。 | インテル Arria 10 および Cyclone 10 GX | Receiver ペイン
VGA DC gain | レシーバーの VGA ゲインを指定します。VGA ゲインのフルレンジは、約 -5 dB (設定 0) から +7 dB (設定 31) になります。 | インテル Stratix 10 L- および H-タイル | Receiver ペイン
V_{00} control | プログラマブルトランスミッター差動出力電圧です。サポートされているすべてのデバイスファミリー | | Transmitter ペイン

関連情報
Channel Manager (205ページ)

9.13. 一般的なエラーのトラブルシューティング

高速リンクピン接続がない場合

ピン接続を確認して、高速リンク (tx_p/n and rx_p/n) の有無を特定してください。古いデザインをインテル Quartus Prime 開発ソフトウェアの最新バージョンに移植する場合、これらの接続の存在が移植後であることを確認してください。

リセットに関する問題

Transceiver Native PHY, Transceiver Reset Controller, および ATX PLL インテル FPGA IP へのリセット入力がアクティブになっていないことを確認してください (1'b1)。Transceiver Toolkit で赤色で強調表示されるのは、セットアップ中のすべての Transceiver Native PHY チャネルです。

未接続の reconfig_clk

reconfig_clk 入力は、Transceiver Native PHY および ATX PLL インテル FPGA IP に接続・駆動してください。接続されていない場合、ツールキットではトランシーバー・リンク・チャネルが表示されません。

9.14. API リファレンスのスクリプトング

インテル Quartus Prime 開発ソフトウェアの API では、Tcl コマンドを使用した Transceiver Toolkit の機能のほか、デバイスリソースのリンクや高速シリアルリンクの特定などのスクリプトタスクにアクセスできます。

プロジェクト設定を Tcl スクリプトで保存して、それを後の検証作業で使用するには、次を実行します。
1. 物理システム全体を記述するリンクを設定し、定義します。
2. Save Tcl Script をクリックして設定を保存し、後に使用できるようにします。

カスタムのテスト・ルーチン・スクリプトの作成も可能です。
スクリプトの実行には、System Explorer のスクリプトフォルダー内のスクリプト名をダブルクリックします。
使用可能な Tcl コマンドの説明のリストを Tcl Console ウィンドウで表示するには、次を実行します。これには使用例が含まれます。

1. Tcl コンソールで、`help help` と入力します。Console で表示されるのは、すべての Transceiver Toolkit Tcl コマンドです。

2. `help <command name>` と入力します。Console で表示されるのはコマンドの説明です。


次の表で一覧表示するのは、使用可能な Transceiver Toolkit のスクリプティング・コマンドです。

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>transceiver_channel_rx_get_data</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>現在のチェッカーデータのリストを返します。結果は、ビット数、エラー数、ビット・エラー・レートの順です。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_get_dcgain</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>レシーバーチャネルの DC ゲイン値を取得します。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_get_dfe_tap_value</code></td>
<td><code>&lt;service-path&gt;</code> <code>&lt;tap position&gt;</code></td>
<td>指定したチャネルの現在のタップ値を指定のタップ位置で取得します。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_get_eqctrl</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>レシーバーチャネルのイコライゼーション・コントロール値を取得します。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_get_pattern</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>現在のデータのチェッカーパターンを名前別に返します。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_has_dfe</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>指定したチャネルで DFE 機能が使用可能かどうかをレポートします。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_is_checking</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>チェッカーが実行されている場合、ゼロ以外の値を返します。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_is_dfe_enabled</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>DFE 機能が指定したチャネルでイネーブルされているかどうかをレポートします。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_is_locked</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>チェッカーが受信データにロックされている場合、ゼロ以外の値を返します。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_reset_counts</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>チェッカー内のビットカウンターとエラーカウンターをリセットします。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_reset</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>指定したチャネルをリセットします。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_set_dcgain</code></td>
<td><code>&lt;service-path&gt;</code> <code>&lt;value&gt;</code></td>
<td>レシーバーチャネルの DC ゲイン値を設定します。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_set_dfe_enabled</code></td>
<td><code>&lt;service-path&gt;</code> <code>&lt;disable(0)/enable(1)&gt;</code></td>
<td>指定したチャネルの DFE 機能をイネーブルまたはディスエーブルします。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_set_dfe_tap_value</code></td>
<td><code>&lt;service-path&gt;</code> <code>&lt;tap position&gt;</code> <code>&lt;tap value&gt;</code></td>
<td>指定したチャネルの現在のタップ値を、指定のタップ位置で、指定した値に設定します。</td>
</tr>
<tr>
<td><code>transceiver_channel_rx_set_dfe_adaptive</code></td>
<td><code>&lt;service-path&gt;</code> <code>&lt;adaptive-mode&gt;</code></td>
<td>指定したチャネルの DFE アダプテーション・モードを設定します。</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>値</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>オフ</td>
</tr>
</tbody>
</table>

continued...
### コマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>transceiver_channel_rx_set_eqctrl</td>
<td>&lt;service-path&gt; &lt;value&gt;</td>
<td>受信チャネルのイコライゼーション・コントロール値を設定します。</td>
</tr>
<tr>
<td>transceiver_channel_rx_start_checking</td>
<td>&lt;service-path&gt;</td>
<td>チェッカーを開始します。</td>
</tr>
<tr>
<td>transceiver_channel_rx_stop_checking</td>
<td>&lt;service-path&gt;</td>
<td>チェッカーを停止します。</td>
</tr>
<tr>
<td>transceiver_channel_rx_set_pattern</td>
<td>&lt;service-path&gt; &lt;pattern-name&gt;</td>
<td>予期されるパターンを設定して、パターン名で指定したパターンにします。</td>
</tr>
<tr>
<td>transceiver_channel_rx_set_word_aligner_enabled</td>
<td>&lt;service-path&gt; &lt;disable(0)/enable(1)&gt;</td>
<td>指定したチャネルのワードアライナーをイネーブルまたはディスエーブルします。</td>
</tr>
<tr>
<td>transceiver_channel_rx_is_word_aligner_enabled</td>
<td>&lt;service-path&gt; &lt;disable(0)/enable(1)&gt;</td>
<td>ワードアライナー機能のイネーブルが、指定したチャネルでされているかどうかをレポートします。</td>
</tr>
<tr>
<td>transceiver_channel_rx_is_locked</td>
<td>&lt;service-path&gt;</td>
<td>チェッカーが受信信号にロックされている場合、ゼロ以外の値を返します。</td>
</tr>
<tr>
<td>transceiver_channel_rx_is_rx_locked_to_data</td>
<td>&lt;service-path&gt;</td>
<td>トランシーバーがLock to Data (LTD) 모드の場合、1を返します。その他のモードの場合は、0を返します。</td>
</tr>
<tr>
<td>transceiver_channel_rx_is_rx_locked_to_ref</td>
<td>&lt;service-path&gt;</td>
<td>トランシーバーがLock to Reference (LTR) モードの場合、1を返します。その他のモードの場合は、0を返します。</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>transceiver_channel_tx_disable_preamble</td>
<td>&lt;service-path&gt;</td>
<td>生成開始時にプリアンブル・モードをディスエーブルします。</td>
</tr>
<tr>
<td>transceiver_channel_tx_enable_preamble</td>
<td>&lt;service-path&gt;</td>
<td>生成開始時にプリアンブル・モードをイネーブルします。</td>
</tr>
<tr>
<td>transceiver_channel_tx_get_number_of_preamble_beats</td>
<td>&lt;service-path&gt;</td>
<td>ビート数を返して、プリアンブル・ワードを送信します。</td>
</tr>
<tr>
<td>transceiver_channel_tx_get_pattern</td>
<td>&lt;service-path&gt;</td>
<td>パターンを返します。</td>
</tr>
<tr>
<td>transceiver_channel_tx_get_preamble_word</td>
<td>&lt;service-path&gt;</td>
<td>プリアンブル・ワードを返します。</td>
</tr>
<tr>
<td>transceiver_channel_tx_get_preemph_pretap1</td>
<td>&lt;service-path&gt;</td>
<td>プリエンファシスの最初のプリタップ値をトランスミッター・チャネルで取得します。</td>
</tr>
<tr>
<td>transceiver_channel_tx_get_preemph_posttap1</td>
<td>&lt;service-path&gt;</td>
<td>プリエンファシスの最初のポストタップ値をトランスミッター・チャネルで取得します。</td>
</tr>
<tr>
<td>transceiver_channel_tx_get_preemph_posttap2</td>
<td>&lt;service-path&gt;</td>
<td>プリエンファシスの2番目のポストタップ値をトランスミッター・チャネルで取得します。</td>
</tr>
<tr>
<td>transceiver_channel_tx_get_preemph_pretap2</td>
<td>&lt;service-path&gt;</td>
<td>プリエンファシスの2番目のプリタップ値をトランスミッター・チャネルで取得します。</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>transceiver_channel_tx_get_vodctrl</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>VOD コントロール値をトランスミッター・チャネルで取得します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_inject_error</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>1 ビットエラーをジェネレーターの出力に注入します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_is_generating</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>ジェネレーターが実行中の場合、ゼロ以外の値を返します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_is_preamble_enabled</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>プリアンブル・モードがイネーブルの場合、ゼロ以外の値を返します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_set_number_of_preamble_beats</code></td>
<td><code>&lt;service-path&gt; &lt;number-of-preamble-beats&gt;</code></td>
<td>ビート数を設定して、プリアンブル・ワードを送信します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_set_pattern</code></td>
<td><code>&lt;service-path&gt; &lt;pattern-name&gt;</code></td>
<td>出力パターンを設定して、パターン名で指定したパターンにします。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_set_preamble_word</code></td>
<td><code>&lt;service-path&gt; &lt;preamble-word&gt;</code></td>
<td>送信されるプリアンブル・ワードを設定します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_set_preemph_preta p1</code></td>
<td><code>&lt;service-path&gt; &lt;value&gt;</code></td>
<td>プリエンファシスの最初のプリタップ値をトランスミッター・チャネルで設定します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_set_preemph_postt ap1</code></td>
<td><code>&lt;service-path&gt; &lt;value&gt;</code></td>
<td>プリエンファシスの最初のポストタップ値をトランスミッター・チャネルで設定します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_set_preemph_postt ap2</code></td>
<td><code>&lt;service-path&gt; &lt;value&gt;</code></td>
<td>プリエンファシスの2番目のポストタップ値をトランスミッター・チャネルで設定します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_set_preemph_preta p2</code></td>
<td><code>&lt;service-path&gt; &lt;value&gt;</code></td>
<td>プリエンファシスの2番目のプリタップ値をトランスミッター・チャネルで設定します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_set_vodctrl</code></td>
<td><code>&lt;service-path&gt; &lt;vodctrl value&gt;</code></td>
<td>VOD コントロール値をトランスミッター・チャネルで設定します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_start_generation</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>ジェネレーターを起動します。</td>
</tr>
<tr>
<td><code>transceiver_channel_tx_stop_generation</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>ジェネレーターを停止します。</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>transceiver_debug_link_get_pattern</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>テストの実行にリンクで使用するパターンを取得します。</td>
</tr>
<tr>
<td><code>transceiver_debug_link_is_running</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>テストがリンクで実行されている場合、ゼロ以外の値を返します。</td>
</tr>
<tr>
<td><code>transceiver_debug_link_set_pattern</code></td>
<td><code>&lt;service-path&gt; &lt;data pattern&gt;</code></td>
<td>テストの実行にリンクで使用するパターンを設定します。</td>
</tr>
<tr>
<td><code>transceiver_debug_link_start_running</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>テストの実行を現在選択されているテストパターンで開始します。</td>
</tr>
<tr>
<td><code>transceiver_debug_link_stop_running</code></td>
<td><code>&lt;service-path&gt;</code></td>
<td>テストの実行を停止します。</td>
</tr>
</tbody>
</table>

表 59. Transceiver Toolkit の Transceiver Toolkit debug_link コマンド
表 60. Transceiver Toolkit の reconfig_analog コマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>transceiver_reconfig_analog_get_logic</td>
<td>&lt;service-path&gt;</td>
<td>現在設定されているトランシーバー・ロジックのチャネルアドレスを取得します。</td>
</tr>
<tr>
<td>transceiver_reconfig_analog_get_rx_dc_gain</td>
<td>&lt;service-path&gt;</td>
<td>現在の論理チャネルアドレスで指定されているレシーバー・チャネルの DC ゲイン値を取得します。</td>
</tr>
<tr>
<td>transceiver_reconfig_analog_get_rx_eq_ctrl</td>
<td>&lt;service-path&gt;</td>
<td>現在の論理チャネルアドレスで指定されているレシーバー・チャネルのイコライゼーション・コントロール値を取得します。</td>
</tr>
<tr>
<td>transceiver_reconfig_analog_get_tx_preampl</td>
<td>&lt;service-path&gt;</td>
<td>現在の論理チャネルアドレスで指定されているトランスミッター・チャネルのプリエンファシスの最初のプリタップ値を取得します。</td>
</tr>
<tr>
<td>transceiver_reconfig_analog_get_tx_preampl</td>
<td>&lt;service-path&gt;</td>
<td>現在の論理チャネルアドレスで指定されているトランスミッター・チャネルのプリエンファシスの最初のポストタップ値を取得します。</td>
</tr>
<tr>
<td>transceiver_reconfig_analog_get_tx_vodctrl</td>
<td>&lt;service-path&gt;</td>
<td>現在の論理チャネルアドレスで指定されているトランスミッター・チャネルの $V_{OD}$ コントロール値を取得します。</td>
</tr>
<tr>
<td>transceiver_reconfig_analog_set_logic</td>
<td>&lt;logic channel address&gt;</td>
<td>トランシーバーの論理チャネルアドレスを設定します。</td>
</tr>
<tr>
<td>transceiver_reconfig_analog_set_rx_dc_gain</td>
<td>&lt;dc_gain value&gt;</td>
<td>現在の論理チャネルアドレスで指定されているレシーバー・チャネルの DC ゲイン値を設定します。</td>
</tr>
<tr>
<td>transceiver_reconfig_analog_set_rx_eq_ctrl</td>
<td>&lt;eqctrl value&gt;</td>
<td>現在の論理チャネルアドレスで指定されているレシーバー・チャネルのイコライゼーション・コントロール値を設定します。</td>
</tr>
<tr>
<td>transceiver_reconfig_analog_set_tx_preampl</td>
<td>&lt;value&gt;</td>
<td>現在の論理チャネルアドレスで指定されているトランスミッター・チャネルのプリエンファシスの最初のプリタップ値を設定します。</td>
</tr>
<tr>
<td>transceiver_reconfig_analog_set_tx_preampl</td>
<td>&lt;value&gt;</td>
<td>現在の論理チャネルアドレスで指定されているトランスミッター・チャネルのプリエンファシスの最初のポストタップ値を設定します。</td>
</tr>
</tbody>
</table>
9.14.2. Data Pattern Generator のコマンド

Data Pattern Generator のコマンドを使用してトランシーバー・チャネルのデバッグ用にデータパターンを制御します。Data Pattern Generator のコンポーネントをインスタンス化して、このコマンドをサポートしてください。

### 表 61．チャネル・タイプ・コマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_channel_type</td>
<td>&lt;service-path&gt; &lt;logical-channel-num&gt;</td>
<td>&lt;service-path&gt; に位置するリコンフィグレーション・ブロックに対するチャネル&lt;logical-channel-num&gt;の検出タイプ (GX/GT) をレポートします。</td>
</tr>
<tr>
<td>set_channel_type</td>
<td>&lt;service-path&gt; &lt;logical-channel-num&gt; &lt;channel-type&gt;</td>
<td>&lt;service-path&gt; に位置するリコンフィグレーション・ブロックに対するチャネル&lt;logical-channel-num&gt;の検出チャネルタイプを上書きして、指定したタイプ (0:GX, 1:GT) にします。</td>
</tr>
</tbody>
</table>

### 9.14.2. Data Pattern Generator のコマンド

Data Pattern Generator のコマンドを使用してトランシーバー・チャネルのデバッグ用にデータパターンを制御します。Data Pattern Generator のコンポーネントをインスタンス化して、このコマンドをサポートしてください。

### 表 62．Soft Data Pattern Generator のコマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_pattern_generator_start</td>
<td>&lt;service-path&gt;</td>
<td>データパターン・ジェネレーターを起動します。</td>
</tr>
<tr>
<td>data_pattern_generator_stop</td>
<td>&lt;service-path&gt;</td>
<td>データパターン・ジェネレーターを停止します。</td>
</tr>
<tr>
<td>data_pattern_generator_is_generating</td>
<td>&lt;service-path&gt;</td>
<td>ジェネレーターが実行中の場合、ゼロ以外の値を返します。</td>
</tr>
<tr>
<td>data_pattern_generator_inject_error</td>
<td>&lt;service-path&gt;</td>
<td>1 ビットエラーをジェネレーターの出力に注入します。</td>
</tr>
<tr>
<td>data_pattern_generator_set_pattern</td>
<td>&lt;service-path&gt; &lt;pattern-name&gt;</td>
<td>&lt;pattern-name&gt; で指定する出力パターンを設定します。</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>値</th>
<th>詳細</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRBS7</td>
<td>模擬ランダムバイナリーシーケンスです。PRBS ファイルはクリアテキストであり、編集が可能です。</td>
</tr>
<tr>
<td>PRBS15</td>
<td>PRBS23</td>
</tr>
<tr>
<td>HF</td>
<td>高周波かつ、0 と 1 を交互に繰り返す定数パターンを出力。</td>
</tr>
</tbody>
</table>

continued...
## コマンド・引数・機能

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_pattern_generator_get_pattern</td>
<td>&lt;service-path&gt;</td>
<td>現在選択している出力パターンを返します。</td>
</tr>
<tr>
<td>data_pattern_generator_get_available_patterns</td>
<td>&lt;service-path&gt;</td>
<td>使用可能なデータパターンのリストを名前別に返します。</td>
</tr>
<tr>
<td>data_pattern_generator_enable_preamble</td>
<td>&lt;service-path&gt;</td>
<td>プリアンブル・モードのイネーブルを生成開始時にします。</td>
</tr>
<tr>
<td>data_pattern_generator_disable_preamble</td>
<td>&lt;service-path&gt;</td>
<td>生成開始時にプリアンブル・モードをディスエーブルします。</td>
</tr>
<tr>
<td>data_pattern_generator_is_preamble_enabled</td>
<td>&lt;service-path&gt;</td>
<td>プリアンブル・モードがイネーブル場合を、ゼロ以外の値を返します。</td>
</tr>
<tr>
<td>data_pattern_generator_set_preamble_word</td>
<td>&lt;preamble-word&gt;</td>
<td>プリアンブル・ワードを設定します（32ビットまたは40ビット）。</td>
</tr>
<tr>
<td>data_pattern_generator_get_preamble_word</td>
<td>&lt;service-path&gt;</td>
<td>プリアンブル・ワードを取得します。</td>
</tr>
<tr>
<td>data_pattern_generator_set_preamble_beats</td>
<td>&lt;service-path&gt; &lt;number-of-preamble-beats&gt;</td>
<td>ビート数を設定しTEプリアンブル・ワードを送信します。</td>
</tr>
<tr>
<td>data_pattern_generator_get_preamble_beats</td>
<td>&lt;service-path&gt;</td>
<td>現在設定されているビート数を返し、プリアンブル・ワードを送信します。</td>
</tr>
<tr>
<td>data_pattern_generator_fcounter_start</td>
<td>&lt;service-path&gt; &lt;max-cycles&gt;</td>
<td>最大サイクルカウントを設定し、周波数カウンターを開始します。</td>
</tr>
<tr>
<td>data_pattern_generator_check_status</td>
<td>&lt;service-path&gt;</td>
<td>現在のステータスのデータパターン・ジェネレーターをクエリーします。ステータスを示すビットマップを返します。ビットの定義は次のとおりです。</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>値</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>イネーブルされています</td>
</tr>
<tr>
<td>1</td>
<td>バイパスがイネーブルされています</td>
</tr>
<tr>
<td>2</td>
<td>Avalon</td>
</tr>
<tr>
<td>3</td>
<td>Sinkがready状態です</td>
</tr>
<tr>
<td>4</td>
<td>Sourceが有効です</td>
</tr>
<tr>
<td>5</td>
<td>周波数カウンターがイネーブルされています</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_pattern_generator_fcounter_report</td>
<td>&lt;service-path&gt;</td>
<td>現在測定されているクロック比をレポートし、&lt;force-stop&gt;に応じて最初のカウントを停止します。</td>
</tr>
</tbody>
</table>
表 63. Hard Data Pattern Generator のコマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>hard_prbs_generator_start</td>
<td>&lt;service-path&gt;</td>
<td>指定したジェネレーターを起動します。</td>
</tr>
<tr>
<td>hard_prbs_generator_stop</td>
<td>&lt;service-path&gt;</td>
<td>指定したジェネレーターを停止します。</td>
</tr>
<tr>
<td>hard_prbs_generator_is_generating</td>
<td>&lt;service-path&gt;</td>
<td>生成のステータスを確認します。次の値を返します。</td>
</tr>
<tr>
<td>hard_prbs_generator_set_pattern</td>
<td>&lt;service-path&gt;</td>
<td>&lt;pattern&gt; に指定するハード PRBS ジェネレーターのパターンを設定します。</td>
</tr>
<tr>
<td>hard_prbs_generator_get_pattern</td>
<td>&lt;service-path&gt;</td>
<td>&lt;pattern&gt; ステータスを取得します。</td>
</tr>
<tr>
<td>hard_prbs_generator_get_available_patterns</td>
<td>&lt;service-path&gt;</td>
<td>所定のハード PRBS ジェネレーター用の使用可能なパターンを返します。</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>値</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>生成中</td>
</tr>
<tr>
<td>1</td>
<td>その他</td>
</tr>
</tbody>
</table>

9.14.3. Data Pattern Checker のコマンド

Data Pattern Checker のコマンドを使用して、生成したデータパターンを検証します。Data Pattern Checker のコンポーネントをインスタンス化して、このコマンドをサポートしてください。

表 64. Soft Data Pattern Checker のコマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_pattern_checker_start</td>
<td>&lt;service-path&gt;</td>
<td>データ・パターン・チェッカーを起動します。</td>
</tr>
<tr>
<td>data_pattern_checker_stop</td>
<td>&lt;service-path&gt;</td>
<td>データ・パターン・チェッカーを停止します。</td>
</tr>
<tr>
<td>data_pattern_checker_is_checking</td>
<td>&lt;service-path&gt;</td>
<td>チェッカーが実行されている場合、ゼロ以外の値を返します。</td>
</tr>
<tr>
<td>data_pattern_checker_is_locked</td>
<td>&lt;service-path&gt;</td>
<td>チェッカーが受信データにロックされている場合、ゼロ以外の値を返します。</td>
</tr>
<tr>
<td>data_pattern_checker_set_pattern</td>
<td>&lt;service-path&gt;</td>
<td>&lt;pattern-name&gt; に予想パターンを設定します。</td>
</tr>
<tr>
<td>data_pattern_checker_get_pattern</td>
<td>&lt;service-path&gt;</td>
<td>現在選択されている予想パターンを名前別に返します。</td>
</tr>
<tr>
<td>data_pattern_checker_get_available_patterns</td>
<td>&lt;service-path&gt;</td>
<td>使用可能なデータパターンのリストを名前別に返します。</td>
</tr>
<tr>
<td>data_pattern_checker_get_data</td>
<td>&lt;service-path&gt;</td>
<td>現在のチェッカーデータのリストを返します。結果は、ビット数、エラー数、およびビット・エラーキーの順になります。</td>
</tr>
<tr>
<td>data_pattern_checker_reset_counters</td>
<td>&lt;service-path&gt;</td>
<td>チェッカー内のビットカウンターとエラーカウンターをリセットします。</td>
</tr>
<tr>
<td>コマンド</td>
<td>引数</td>
<td>機能</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>data_pattern_checker_fcncer_start</td>
<td>&lt;service-path&gt; &lt;max-cycles&gt;</td>
<td>最大サイクルカウントを設定し、周波数カウンターを開始します。</td>
</tr>
<tr>
<td>data_pattern_checker_check_status</td>
<td>&lt;service-path&gt; &lt;service-path&gt;</td>
<td>現在のステータスのデータパターン・チェッカーをクエリします。ステータスを示すビットマップを返します。</td>
</tr>
<tr>
<td>data_pattern_checker_fcncer_report</td>
<td>&lt;service-path&gt; &lt;force-stop&gt;</td>
<td>現在測定されているクロック比をレポートし、&lt;force-stop&gt;に応じて最初のカウントを停止します。</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>値</th>
<th>ステータス</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>イネーブルされています</td>
</tr>
<tr>
<td>1</td>
<td>ロックされています</td>
</tr>
<tr>
<td>2</td>
<td>バイパスがイネーブルされています</td>
</tr>
<tr>
<td>3</td>
<td>Avalon</td>
</tr>
<tr>
<td>4</td>
<td>Sink が ready 状態です</td>
</tr>
<tr>
<td>5</td>
<td>Source が有効です</td>
</tr>
<tr>
<td>6</td>
<td>周波数カウンターがイネーブルされています</td>
</tr>
</tbody>
</table>

表 65. Hard Data Pattern Checker のコマンド

<table>
<thead>
<tr>
<th>コマンド</th>
<th>引数</th>
<th>機能</th>
</tr>
</thead>
<tbody>
<tr>
<td>hard_prbs_checker_start</td>
<td>&lt;service-path&gt;</td>
<td>指定したハード PRBS チェッカーを開始します。</td>
</tr>
<tr>
<td>hard_prbs_checker_stop</td>
<td>&lt;service-path&gt;</td>
<td>指定したハード PRBS チェッカーを停止します。</td>
</tr>
<tr>
<td>hard_prbs_checker_is_checking</td>
<td>&lt;service-path&gt;</td>
<td>指定したハード PRBS チェッカーの実行ステータスをチェックします。チェッカーが実行されている場合はゼロ以外の値を返します。</td>
</tr>
<tr>
<td>hard_prbs_checker_set_pattern</td>
<td>&lt;service-path&gt; &lt;pattern&gt;</td>
<td>指定したハード PRBS チェッカーのパターンをパラメーター&lt;pattern&gt;に設定します。</td>
</tr>
<tr>
<td>hard_prbs_checker_get_pattern</td>
<td>&lt;service-path&gt;</td>
<td>所定のハード PRBS チェッカーの現在のパターンを返します。</td>
</tr>
<tr>
<td>hard_prbs_checker_get_available_patterns</td>
<td>&lt;service-path&gt;</td>
<td>所定のハード PRBS チェッカーの使用可能なパターンを返します。</td>
</tr>
<tr>
<td>hard_prbs_checker_get_data</td>
<td>&lt;service-path&gt;</td>
<td>現在のビットおよびエラーカウント・データを指定したハード PRBS チェッカーから返します。</td>
</tr>
<tr>
<td>hard_prbs_checker_reset_counters</td>
<td>&lt;service-path&gt;</td>
<td>指定したハード PRBS チェッカーのビットおよびエラーカウント・データをリセットします。</td>
</tr>
</tbody>
</table>

9.15. トランシーバー・リンクのデバッグ 改訂履歴

次の改訂履歴はこの章に適用されます。
<table>
<thead>
<tr>
<th>ドキュメント・バージョン</th>
<th>インテル Quartus Prime バージョン</th>
<th>変更内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.07.03</td>
<td>18.0.0</td>
<td>次のトピックを追加しました。デバイスのサポート。インテル Stratix 10 TX / E-タイルのサポートを追加しました。デバイスファミリーの列を次の表に追加しました。Transceiver Toolkit のコントロール・ペインの設定。インテル Stratix 10 L-タイルおよびH-タイルの PMA設定の図を更新しました。</td>
</tr>
<tr>
<td>2017.11.27</td>
<td>17.1.0</td>
<td>サポートされていない DFE アダプテーション・モードを Transceiver Toolkit の channel_rx コマンドの表から削除しました。次の表を削除しました。Transceiver Toolkit Decision Feedback Equalization (DFE) コマンド。次の表を削除しました。ループバック・コマンド</td>
</tr>
<tr>
<td>2017.11.06</td>
<td>17.1.0</td>
<td>Stratix 10 デバイスのサポートを追加しました。EyeQ の名前を変更して、Eye Viewer にしました。Eye Viewerツールを使用したインテル Stratix 10 H-タイル量産デバイスのテスト実行に関する項を追加しました。トランシーバーデバッグ・フローのトピックを更新し、トランシーバーデバッグ・フロー・チュートリアルに名前を変更しました。デバイスのコンフィグレーションに関する廃止予定のトピックを削除しました。Debug IP コアのインタスタンス化およびパラメータ化の手順を更新しました。次の図を削除しました。Altera Debug Master エンドポイント・ブロック図。デバッグフローの一部としてプログラミング・デザインの手順を追加しました。トランシーバーリンクのデバッグに関する情報を更新しました。</td>
</tr>
<tr>
<td>2016.10.31</td>
<td>16.1.0</td>
<td>インテルにブランド名を変更しました。インテル Arria 10 の EyeQ サポートを削除しました。Continuous Adaptation の名前を変更して、Adaptation Enabled にしました。</td>
</tr>
<tr>
<td>2015年5月</td>
<td>15.0.0</td>
<td>Stratix V と Arria 10 の実装上の違いに関する項を追加しました。Quartus II ソフトウェアを使用した Arria 10 Transceiver Toolkit デザインの推奨フローに関する項を追加しました。Transceiver Toolkit のトラブルシューティングに関する項を追加しました。次の項で Transceiver Toolkit と Arria 10 デバイスの併用に関する情報を更新しました。シリアル・ビット・コンパレーター・モード Arria 10 のサポートおよび制限事項。BER テストのコンフィグレーション。PRBS シグナル・アイ・テストの設定。アルテラデザイン例の適応。デザイン例の変更。カスタム・トラフィック・シグナル・アイ・テストのコンフィグレーション。リンク最適化テストの設定。PMA アナログ設定コントロールの設定。BER テストの実行。ツールキット GUI 設定リファレンス。次の表を修正しました。Transceiver Toolkit IP コアの設定。次の図を差し替えました。2 回のテスト実行の結果を示す EyeQ 設定とステータスの図と 3 回のテスト実行の結果を示す EyeQ 設定とステータスの図。次の図を追加しました。Arria 10 アルテラ・デバッグ・マスターエンドポイントのブロック図。次の図を追加しました。BER テストのコンフィグレーション (Arria10/ Gen 10/ 20nm) のブロック図。次の図を追加しました。PRBS 信号テストのコンフィグレーション (Arria10/ Gen 10/ 20nm) のブロック図。次の図を追加しました。カスタム・トラフィック・シグナル・アイ・テストのコンフィグレーション (Arria 10/ Gen 10/ 20nm) のブロック図。</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>ドキュメント・バージョン</th>
<th>インテル Quartus Prime バージョン</th>
<th>変更内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014 年 12 月</td>
<td>14.1.0</td>
<td>次の図を追加しました。PMA アナログ設定制御のコンフィグレーション (Arria 10/Gen 10/20nm) のブロック図。次の図を追加しました。One Channel Loopback モード (Arria 10/20nm) のブロック図。次の図を追加しました。Four Channel Loopback モード (Arria 10/Gen 10/20nm) のブロック図。ソフトウェア・バージョン 15.0 の制限事項。Transceiver Toolkit では、Arria 10 デザインのEyeQ をサポートしています。EyeQ 用のオプションのハード・アクセラレーションをサポートします。これにより、EyeQ データの収集がはるかに高速で行われます。これを Arria 10 トランシーバー・ネイティブ PHY IP コアの Dynamic Configuration タブで有効にします。Enable ODI acceleration logic をオンにします。</td>
</tr>
<tr>
<td>2014 年 6 月</td>
<td>14.0.0</td>
<td>ポップアップ・メニュー、IP Catalog, Quartus II, Qsys で Channel Manager の GUI の変更を更新しました。Arria 10 の ADME および JTAG デバッグリンク情報を追加しました。コマンドラインから Tcl スクリプトを実行する手順を追加しました。ヒートマップ表示オプションを追加しました。reconfig_clk の生成のために内部 PLL を使用する手順を追加しました。RX CDR PLL ステータスが LTD モードでトグルできることを記載した注記を追加しました。</td>
</tr>
<tr>
<td>2013 年 11 月</td>
<td>13.1.0</td>
<td>再編成と DITA への変換を行いました。</td>
</tr>
<tr>
<td>2013 年 5 月</td>
<td>13.0.0</td>
<td>Conduit モードサポート、シリアル・ビット・コンバータ、必要なファイル、Tcl コマンドの表を追加しました。</td>
</tr>
<tr>
<td>2012 年 11 月</td>
<td>12.1.0</td>
<td>編集上の軽微な更新を行いました。Tcl ヘルプ情報を追加し、Tcl コマンド・テーブルを削除しました。28 Gbps トランシーバー・サポートの項を追加しました。</td>
</tr>
<tr>
<td>2012 年 8 月</td>
<td>12.0.1</td>
<td>全体的な再構成を実行し、アルテラデザイン例の変更についての手順を更新しました。</td>
</tr>
<tr>
<td>2012 年 6 月</td>
<td>12.0.0</td>
<td>Transceiver Toolkit 機能のアップデートのためのメンテナンス・リリース。</td>
</tr>
<tr>
<td>2011 年 11 月</td>
<td>11.1.0</td>
<td>Transceiver Toolkit 機能のアップデートのためのメンテナンス・リリース。</td>
</tr>
<tr>
<td>2011 年 5 月</td>
<td>11.0.0</td>
<td>新しい Tcl シナリオを追加しました。</td>
</tr>
<tr>
<td>2010 年 12 月</td>
<td>10.1.0</td>
<td>新しい文書テンプレートに変更しました。10.1 リリースの新機能を追加しました。</td>
</tr>
<tr>
<td>2010 年 8 月</td>
<td>10.0.1</td>
<td>リンクを修正しました。</td>
</tr>
<tr>
<td>2010 年 7 月</td>
<td>10.0.0</td>
<td>初版</td>
</tr>
</tbody>
</table>

関連情報

Documentation Archive
以前のバージョンの インテル Quartus Prime ハンドブックは、文書アーカイブを検索してください。
A. インテル Quartus Prime プロ・エディション・ユーザーガイド

インテル Quartus Prime プロ・エディション FPGA デザインフローのすべての段階の包括的ガイドについては次のユーザーガイドを参照してください。

関連情報

• インテル Quartus Prime プロ・エディション スタート・ユーザーガイド
  インテル Quartus Prime プロ・エディション開発ソフトウェアの基本機能、ファイル、およびデザイントフローについて説明します。インテル Quartus Prime プロ・エディションプロジェクトおよび IP の管理、初期デザイン時の考慮事項、旧ソフトウェア・バージョンからのプロジェクト・マイグレーションが含まれます。

• インテル Quartus Prime プロ・エディション Platform Designer ユーザーガイド
  Platform Designer を使用したシステムの作成および最適化について説明します。Platform Designer は、システム統合ツールとして、カスタマイズされた IP コアのプロジェクトへの統合を簡素化します。Platform Designer では、相互接続ロジックを自動生成して、知的財産(IP)機能とサブシステムを接続します。

• インテル Quartus Prime プロ・エディションデザイン推奨事項 ユーザーガイド
  インテル Quartus Prime プロ・エディション開発ソフトウェアを使用して FPGA をデザインするための最適なデザイン手法について説明します。HDL コーディング・スタイルと同期デザイン的手法は、デザインのパフォーマンスに大きな影響を与えます。推奨される HDL コーディング・スタイルに従うことで、インテル Quartus Prime プロ・エディション合成によってデザインが最適な形でハードウェアに実装されます。

• インテル Quartus Prime プロ・エディション デザインのコンパイル ユーザーガイド
  インテル Quartus Prime プロ・エディション Compiler の全ステージでのセットアップ、実行方法、最適化について説明します。Compiler では、デザインの合成、配置、および配線をデバイス・プログラミング・ファイルの生成前に実行します。

• インテル Quartus Prime プロ・エディション デザイン最適化ユーザーガイド
  ここで説明するインテルの設定、ツール、および手法を使用すると、インテル Quartus Prime プロ・エディション FPGA で最大限のデザインパフォーマンスが達成されます。手法は、デザイン・ネットリストの最適化、リタイミングおよびタイミング・クロージャーを制限するクリティカル・チェーンの処理、およびデバイスリソース使用量の最適化が含まれます。

• インテル Quartus Prime プロ・エディション Programmer ユーザーガイド:
  インテル Quartus Prime プロ・エディション Programmer の操作について説明します。Intel Programmer を使用すると、インテル FPGA デバイスのコンフィグレーションや、CPLD およびコンフィグレーション・デバイスのプログラムを、インテル Quartus Prime プロ・エディション FPGA ダウンロード・ケーブルとの接続を介して行うことができます。

• インテル Quartus Prime プロ・エディションブロックベースのデザイン ユーザーガイド
  ブロックベースのデザインフローについて説明します。これにより、モジュラーミまたは階層型のデザインフローも呼ばれます。この高度なフローにより、デザインブロック(または階層デザイン・インスタンスを構成するロジック)をプロジェクト内で保持し、デザインブロックを他のプロジェクトで再利用することができます。
インテル Quartus Prime プロ・エディション ユーザーガイド

A. インテル Quartus Prime プロ・エディション ユーザーガイド

インテル Quartus Prime プロ・エディション パーシャル・リコンフィグレーション ユーザーガイド
ここで説明するパーシャル・リコンフィグレーションは、高度なデザインフローです。これによって FPGA の一部の動的リコンフィグレーションが可能です。その間、FPGA デザインの他の部分は引き続き動作します。複数のペルソナを特定のデザイン領域に対して定義し、他のエリアの動作には影響を与えません。

インテル Quartus Prime プロ・エディション サードパーティ・シミュレーション ユーザーガイド
RTL およびゲートレベルのデザイン・シミュレーション・サポートについて説明します。サポート対象のサードパーティ・シミュレーションツールは、Aldec*、Cadence*、Mentor Graphics*、および Synopsys* によるものです。このツールを使用すると、デザイン動作の検証をデバイス・プログラミング前にすることができます。このユーザーガイドには、シミュレータ・サポート、シミュレーション・フロー、およびインテル FPGA IP のシミュレーションが含まれます。

インテル Quartus Prime プロ・エディション サードパーティ合成ユーザーガイド
デザインのオプション合成のサポートについて説明します。サポート対象のサードパーティー合成ツールは、Mentor Graphics* および Synopsys* によるものです。このユーザーガイドには、デザインフローのステップ、生成されたファイルの説明、および合成ガイドラインが含まれます。

インテル Quartus Prime プロ・エディション サードパーティーのロジック等価性検証ツール ユーザーガイド
デザインのロジック等価性検証 (LEC) のサポートについて説明します。サポート対象は、OneSpin* によるサードパーティの LEC ツールです。コンパイル・ネットリスト間でのロジック等価性の検証方法について説明します。

インテル Quartus Prime プロ・エディション デバッグツール ユーザーガイド
インテル Quartus Prime プロ・エディション インシステム・デバッグ・ツールのポータフリオについて説明します。このツールを使用して、デザインのリアルタイム検証を行います。このツールによる可視性は、デザイン内の信号をデバックロジックに配線 (または「タップ」) することで得られます。このツールには、System Console、Signal Tap ロジック・アナライザー、Transceiver Toolkit、In-System Memory Content Editor、および In-System Sources and Probes Editor が含まれています。

インテル Quartus Prime プロ・エディション Timing Analyzer ユーザーガイド
静的タイミング解析の基本原理との使用方法について説明します。インテル Quartus Prime プロ・エディション Timing Analyzer では、強力な ASIC 形式のタイミング解析ツールとして、デザイン内のすべてのロジックのタイミング・パフォーマンスを検証します。これには、業界標準の制約、解析、およびレポート手法を使用します。

インテル Quartus Prime プロ・エディション 電力解析および最適化ユーザーガイド
インテル Quartus Prime プロ・エディション Power Analysis ツールについて説明します。このツールを使用すると、デバイスの消費電力の高精度な見積もりが可能です。デバイスの消費電力を見積もって、電力パッケージの策定や電源、電圧、ヒートシンク、および冷却システムのデザインを行います。

インテル Quartus Prime プロ・エディション デザイン制約ユーザーガイド
ここで説明するタイミングおよびロジック制約によって、Compiler でのデザイン実装方法が影響を受けます。実装方法には、ビン・アサインメント、デザインオプション、ロジックオプション、およびタイミング制約などが含まれます。Interface Planner を使用して、インターフェイス実装のプロトタイプの作成、クロックのプランニング、および有効なデバイス・フロアプランの迅速な定義を行います。Pin Planner を使用して、すべての I/O アサインメントの可視化、変更、および確認をターゲットデバイスのグラフィック表示で行います。

フィードバック

インテル Quartus Prime プロ・エディション ユーザーガイド: デバッグツール

245
• インテル Quartus Prime プロ・エディション PCB デザインツール ユーザーガイド
  Mentor Graphics* および Cadence*によるオプションのサードパーティー PCB デザインツールのサポートについて説明します。シグナル・インテグリティー解析および HSPICE モデルと IBIS モデルによるシミュレーションに関する情報も含まれます。

• インテル Quartus Prime プロ・エディション スクリプティング ユーザーガイド
  Tcl およびコマンドライン・スクリプトを使用した、インテル Quartus Prime プロ・エディション開発ソフトウェアの制御および幅広い機能の実行について説明します。これには、プロジェクト管理、制約指定、コンパイルまたはタイミング解析の実行、レポート生成などが含まれます。