

Vivado ユーザー ガイド

ロジック シミュレーション

UG900 (v2012.4) 2012 年 12 月 18 日



Notice of Disclaimer

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

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

本資料は英語版 (v2012.4) を翻訳したもので、内容に相違が生じる場合には原文を優先します。

資料によっては英語版の更新に対応していないものがあります。

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

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

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	改訂内容
2012年7月25日	2012.2	初版
2012年10月16日	2012.3	<p>2012.3 での変更点は次のとおりです。</p> <ul style="list-style-type: none"> 第1章「ロジック シミュレーションの概要」: <ul style="list-style-type: none"> 合成後およびインプリメンテーション後の機能に関する概要情報を追加 「シミュレーション モード」を移動 第2章「Vivado シミュレーション コンポーネントの理解」: <ul style="list-style-type: none"> GTXE2_CHANNEL/GTXE2_COMMON を追加 LBL_VHD 情報を「グローバル トライステートおよびグローバル セット/リセット信号の使用」へ追加 第3章「Vivado シミュレータの使用」: <ul style="list-style-type: none"> 番号付きの GUI 部分の表記方法を基準に合わせて、説明を追加 「シミュレーション設定」の ModelSim 情報を明確化 「シミュレーション ソース ファイルの追加または作成」のアクティブ シミュレーション セットの説明を明確化 「合成後のシミュレーションの実行」および「インプリメンテーション後のシミュレーションの実行」を追加 章全体のアクティブ シミュレーション セットの説明を明確化 第4章「デザインのコンパイルとシミュレーション」の Vivado シミュレータのスナップショット ビヘイビアに関する説明を追加 第5章「波形を使用した解析」: <ul style="list-style-type: none"> 機能や使用状況でセクションを分類 プロジェクト モードおよび非プロジェクト モードの情報を分類 付録A 「Vivado IDE 外でサードパーティ シミュレータを使用したシミュレーションの実行」: <ul style="list-style-type: none"> 「modelsim.ini ファイル」を追加 「タイミング シミュレーションの実行」の情報をアップデート 付録B 「Verilog および VHDL の例外」で、該当するパラメーター関連付けでスライス、インデックス、フォーマルの選択などがサポートされない関数にそれを記述
2012年12月18日	2012.4	<p>2012.4 での変更点：</p> <ul style="list-style-type: none"> 11 ページの表 1-1 に合成後およびインプリメンテーション後の論理およびタイミング シミュレーション情報を追加 18 ページの「UNISIM ライブライ」の構文を修正 22 ページの「GHTE2_CHANNEL/GHTE2_COMMON」を追加 25 ページの「SAIF のダンプ」を追加 -snapshot コンパイル オプションを削除 XELAB オプションから -vcdfile および -vcdfunit を削除 30 ページの「ザイリンクスライブライを使用した RTL シミュレーション」、32 ページの「同期エレメントの X 伝搬のディスエーブル」、33 ページの「コンフィギュレーション インターフェイスのシミュレーション」、36 ページの「シミュレーションでのブロック RAM の競合チェックのディスエーブル方法」を追加 50 ページの「合成後の論理シミュレーションの実行」、50 ページの「合成後のタイミングシミュレーションの実行」、51 ページの「インプリメンテーション後のシミュレーションの実行」、51 ページの「インプリメンテーション後のタイミングシミュレーションの実行」、52 ページの「シミュレーションの種類」、53 ページの「Vivado シミュレータのスタンドアロン フローの例」を追加 66 ページの表 4-2 で set_property コマンドに -mt オプションを使用する方法に説明を追加 96 ページの「サードパーティ シミュレータでの SAIF のダンプ」を追加

目次

改訂履歴 2

第 1 章 : ロジック シミュレーションの概要

概要	6
シミュレーションフロー	6
サポートされるシミュレータ	9
Vivado シミュレータの機能	10
言語サポート	10
OS サポートおよびリリースの変更点	10
シミュレーションライブラリ	10
シミュレーションモード	12

第 2 章 : Vivado シミュレーションコンポーネントの理解

概要	15
テストベンチまたはスティミュラスファイル	15
シミュレーションライブラリ	17
ネットリスト生成プロセス(非プロジェクトモード)	24
グローバルリセットおよびトライステート	28
ザイリンクスライブラリを使用した RTL シミュレーション	30
同期エレメントの X 伝搬のディスエーブル	32
コンフィギュレーションインターフェイスのシミュレーション	33
シミュレーションでのブロック RAM の競合チェックのディスエーブル方法	36

第 3 章 : Vivado IDE でのシミュレーションの実行およびソースコードのデバッグ

概要	37
シミュレーション設定	37
シミュレーションソースの管理	41
Vivado シミュレータの使用	44
合成後およびインプリメンテーション後のタイミングシミュレーション	49
合成後のシミュレーションの実行	50
インプリメンテーション後のシミュレーションの実行	51
シミュレーションの種類	52
シミュレーションの一時停止	52
シミュレーション結果の保存	52
シミュレーションの終了	52
Vivado シミュレータのスタンドアロンフローの例	53
ソースレベルでのデバッグ	53

第 4 章 : デザインのコンパイルとシミュレーション

概要	56
----------	----

デザイン ファイルの解析	56
xelab を使用したスナップショットのエラボレーションおよび生成	58
xsim によるデザイン スナップショットのシミュレーション	62
プロジェクト ファイルの構文	63
定義済み XILINX_SIMULATOR マクロ (Verilog シミュレーション用)	63
非プロジェクト モードでのデザイン シミュレーション	64
ビヘイビア シミュレーションの実行	65
合成後およびインプリメンテーション後のシミュレーション	66
ライブラリ マップ ファイル (xsim.ini)	67
xelab、xvhd、xvlog コマンド オプション	68
混合言語シミュレーションの使用	71

第 5 章：波形を使用した解析

概要	75
波形コンフィギュレーションと波形ビュー	76
前のシミュレーション設定からのシミュレーションデータの表示 (スタティック シミュレーション)	77
波形コンフィギュレーションの HDL オブジェクト	78
波形のカスタマイズ	81
波形表示の制御	86
波形の分類	88
波形の解析	89
非プロジェクト モードの使用	92

付録 A : Vivado IDE 外でサードパーティ シミュレータを使用したシミュレーションの実行

概要	93
modelsim.ini ファイル	93
RTL/ビヘイビア シミュレーションの実行	94
ネットリスト シミュレーションの実行	95
タイミング シミュレーションの実行	95
サードパーティ シミュレータでの SAIF のダンプ	96

付録 B : Verilog および VHDL の例外

概要	98
VHDL 言語サポートの例外	98
Verilog 言語サポートの例外	100

付録 C : その他のリソース

ザイリンクス リソース	103
ソリューション センター	103
参照資料	103

ロジック シミュレーションの概要

概要

シミュレーションは、現実のデザイン ビヘイビアをソフトウェア環境でエミュレートするプロセスです。シミュレーションをすると、ステイミュラスを挿入してデザイン出力を観察することで、デザインの機能を検証できます。シミュレータでは、ハードウェア記述言語 (HDL) コードが回路機能に変換され、論理結果が表示されます。

本書では、ザイリンクス Vivado™ Integrated Design Environment (IDE) を使用した場合に使用できるシミュレーション オプションについて説明します。

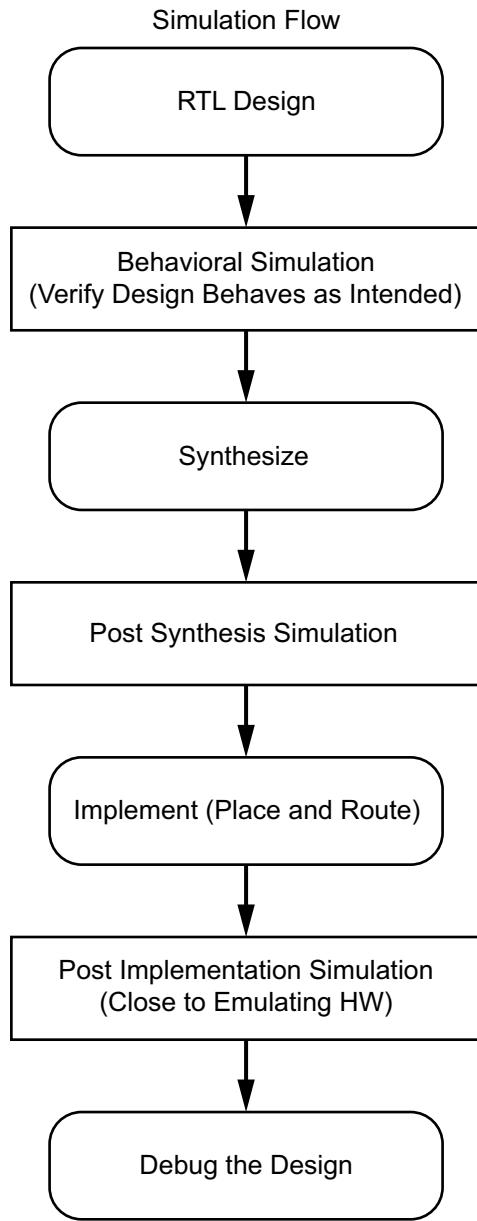
この章では、シミュレーションプロセスの概要および Vivado IDE のシミュレーション オプションについて記述します。Vivado IDE では、プログラマブル ロジック デザインのソリューションを提供する複数の HDL シミュレーション ツールを使用できます。

シミュレーション フロー

シミュレーションは、デザイン フローのさまざまな段階で実行できます。デザイン入力後の最初の段階の 1 つであり、インプリメンテーション後の最後の段階の 1 つ (最終的な機能とデザイン パフォーマンスの検証の一部) でもあります。

シミュレーションは、対話型プロセスで、デザイン機能とタイミングの両方の条件が満たされるまで繰り返す必要があります。

7 ページの図 1-1 は、典型的なデザインのシミュレーション フローを示しています。



X12960

図 1-1: シミュレーション フロー

レジスタ トランスマッパー レベルのビヘイビア シミュレーション

レジスタ トランスマッパー レベル (RTL) のビヘイビア シミュレーションには、次が含まれます。

- RTL コード
- インスタンシエート済み UNISIM ライブライコンポーネント
- インスタンシエート済み UNIMACRO コンポーネント
- XILINXCORELIB および UNISIM ゲートレベル モデル (Vivado ロジック アナライザ用)
- SECUREIP ライブライ

RTL レベルのシミュレーションでは、システムまたはチップ レベルで記述を検証またはシミュレーションできます。

最初のパス シミュレーションは、通常コード構文を検証し、コードが予測どおりに機能しているかどうかを確認するために実行されます。この段階では、タイミング情報は提供されておらず、シミュレーションはレース競合の可能性を回避するため、ユニット遅延モードで実行されます。

RTL シミュレーションは、デザインにインスタンシエートされた UNISIM ライブラリが含まれない限り、アーキテクチャごとに異なることはありません。ザイリンクスでは、このインスタンシエーションをサポートするため、UNISIM および XILINXCORELIB ライブラリを提供しています。

デザインがビヘイビア RTL でインプリメントされるより前に検証する場合は、早めに必要な変更をすることで、デザイン サイクルを節約できます。



ヒント : 最初にデザインを作成する場合は、コード ビヘイビアを維持する必要があります。必要でない限り、特定コンポーネントはインスタンシエートしないでください。コンポーネントが推論可能ではない場合、コンポーネントをインスタンシエートする必要のあることがあります。

最初のデザイン作成をビヘイビア コードにのみ限定すると、次が可能になります。

- より読みやすいコード
- より高速でシンプルなシミュレーション
- コード ポータビリティ (別のデバイス ファミリへ移行可能)
- コード再利用性 (今後別のデザインに同じコードを使用可能)

合成後のシミュレーション

合成済みのネットリストをシミュレーションすると、合成後のデザインが論理要件を満たしているか、予測どおりに動作するかどうかが検証できます。

一般的ではありませんが、このシミュレーション段階でも、概算されたタイミング数値を使用してタイミング シミュレーションを実行することはできます。

論理シミュレーションネットリストは、階層状になっており、プリミティブ モジュールおよびエンティティ レベルに展開されます。階層の最下位にはプリミティブおよびマクロ プリミティブが含まれます。これらのプリミティブは Verilog の場合は UNISIMS_VER ライブラリに、VHDL の場合は UNISIM ライブラリに含まれます。詳細は、[18 ページの「UNISIM ライブラリ」](#) を参照してください。

合成後のシミュレーションについては、次を参照してください。

- [50 ページの「合成後の論理シミュレーションの実行」](#)
- [66 ページの「合成後およびインプリメンテーション後のシミュレーション」](#)
- [67 ページの「タイミング シミュレーションのアノテーション コマンド」](#)

インプリメンテーション後のシミュレーション

インプリメンテーション後は論理シミュレーションまたはタイミングシミュレーションを実行できます。タイミングシミュレーションは、実際にデバイスにダウンロードされるデザインに一番近いエミュレーションで、インプリメント済みデザインが論理要件およびタイミング要件を満たし、デバイスで予測どおりのビヘイビアになることが確認できます。



重要: タイミングシミュレーション全体を実行すると、終了したデザインはエラーのない状態になります（次のようなエラーが発見されない場合を除く）。

- 次が原因で、合成後およびインプリメンテーション後に機能が変更された場合：
 - 不一致になる合成属性または制約 (full_case および parallel_case など)
 - ザイリンクス デザイン制約 (XDC) ファイルへの UNISIM 属性の適用
 - 異なるシミュレータによる合成中の言語変換
- デュアルポート RAM の競合
- タイミング制約の適用がないか、不正適用
- 非同期バスの操作
- 最適化手法による論理問題

詳細は、次を参照してください。

- 51 ページの「インプリメンテーション後のシミュレーションの実行」
- 66 ページの「合成後およびインプリメンテーション後のシミュレーション」
- 67 ページの「タイミングシミュレーションのアノテーションコマンド」

サポートされるシミュレータ

Vivado では、次のシミュレータがサポートされます。

- Vivado シミュレータ (Vivado IDE に統合)
- Mentor Graphics QuestaSim/ModelSim (Vivado IDE に統合)
- Cadence Incisive Enterprise Simulator (IES)
- Synopsys VCS および VCS MX
- Aldec Active-HDL および Rivera-PRO*

注記: (* Aldec シミュレータは互換性はありますが、ザイリンクステクニカルサポートではサポートされません。サポートに関する問題については、Aldec にお尋ねください)。

Vivado シミュレータの機能

Vivado シミュレータでは、次の機能がサポートされます。

- ソース コードのデバッグ
- SDF アノテーション
- VCD ダンプ
- SAIF ダンプ (電力解析および最適化用)
- HardIP ブロック (MGT および PCIe® など) のネイティブ サポート
- マルチスレッド コンパイル
- 混合言語 (VHDL および Verilog) の使用
- 1 クリックでシミュレーションを再コンパイルおよび再起動
- 1 クリックでコンパイルおよびシミュレーション
- ビルトイン ザイリンクス シミュレーション ライブラリ

サポートされるシミュレータのバージョンについては、『ザイリンクス デザイン ツール：リリース ノート ガイド』(UG631)[[参照 1](#)] を参照してください。

注記：カードパーティ シミュレータは Vivado IDE の外部で実行する必要があります (Vivado IDE から選択して起動可能な QuestaSim/ModelSim は例外)。

言語サポート

次の言語がサポートされます。

- VHDL IEEE-STD-1076-1993
- Verilog IEEE-STD-1364-2001
- 標準遅延フォーマット (SDF) バージョン 2.1
- VITAL-2000

OS サポートおよびリリースの変更点

最新のリリースでの変更点については、『ザイリンクス デザイン ツール：リリース ノート ガイド』(UG631)[[参照 1](#)] を参照してください。

OS サポートについては、『ザイリンクス デザイン ツール：インストールおよびライセンス ガイド』(UG798) [[参照 2](#)] を参照してください。

シミュレーション ライブラリ

シミュレーション フローをサポートするために必要なライブラリは、次のとおりです。

- SIMPRIMS_VER : タイミングを含む Verilog UNISIM
- XILINXCORELIB : VHDL IP
- XILINXCORELIB_VER : Verilog IP
- UNISIM : VHDL UNISIM
- UNISIMS_VER : Verilog UNISIM
- UNIMACRO : VHDL UNIMACRO
- UNIMACRO_VER : Verilog UNIMACRO
- SECUREIP : Verilog Hard IP
- UNIFAST : 高速シミュレーション VHDL ライブラリ
- UNIFAST_VER : 高速シミュレーション Verilog ライブラリ

シミュレーション ライブラリについては、[第 2 章の「シミュレーション ライブラリ」](#)を参照してください。

また、『7 シリーズ FPGA ライブラリ ガイド (HDL 用)』(UG768) [\[参照 6\]](#)を参照してください。

注記 : Vivado シミュレータでは、プリコンパイル済みのシミュレーション デバイス ライブラリが使用され、アップデートがインストールされると自動的にこれらのライブラリがアップデートされます。

表 1-1 は、シミュレーション ポイントとどのシミュレーション ライブラリがどのシミュレーション ポイントで使用されるかをリストしています。

表 1-1: シミュレーション ポイントと関連ライブラリ

	UNISIM	Unifast	UNIMACRO	XILINXCORELIB モデル	SECUREIP	SIMPRIMS (Verilog のみ)	標準遅延 フォーマット (SDF)
1. レジスタ トランシスファー レベル (RTL)	○	○	○	○	○	該当なし	X
2. 合成後シミュレーション (論理)	○	○	該当なし	該当なし	○	X	X
3. 合成後シミュレーション (タイミング)	該当なし	該当なし	該当なし	該当なし	○	○	○
4. インプリメンテーション後シミュレーション (論理)	○	○	該当なし	該当なし	○	該当なし	該当なし
5. インプリメンテーション後シミュレーション (タイミング)	該当なし	該当なし	該当なし	該当なし	○	○	○

シミュレーション ライブラリの詳細は、[第 2 章「Vivado シミュレーション コンポーネントの理解」](#)を参照してください。

シミュレーション モード

Vivado シミュレータには、プロジェクト モードと非プロジェクト モードの 2 つのモードがあります。

プロジェクト モード : GUI の使用

Vivado IDE でプロジェクトを使用してシミュレーションを実行すると、Vivado シミュレータのグラフィカルユーザーインターフェイス (GUI) が開き、シミュレーションデータが画像表示されます。メニュー コマンドおよびツールバー ボタンを使用してシミュレーションを実行し、デザインを検証し、データをデバッグします。

プロジェクト モードの場合、次が実行できます。

- プロジェクトを開く、または作成します。
- テストベンチを作成、または追加します。
- デザインファイルを追加します。

詳細は、「[テストベンチまたはスティミュラス ファイル](#)」を参照してください。

- Vivado IDE の Flow Navigator の [Simulation] セクションから、[Simulation Settings] をクリックし、シミュレーション オプションを選択および設定します。

詳細は、「[シミュレーション設定](#)」および[第 3 章の「」](#)を参照してください。

- Flow Navigator の [Run Simulation] をクリックし、ビヘイビア RTL シミュレーションを開始します。
- Vivado IDE の実行、エラー、ログ メッセージを参照してください。

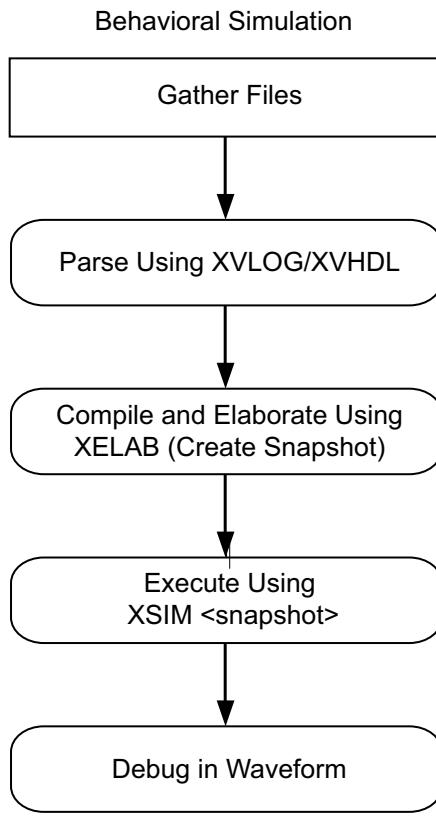
詳細は、[第 3 章「Vivado IDE でのシミュレーションの実行およびソース コードのデバッグ」](#)を参照してください。

- デザインを Vivado IDE の波形ビューアーで解析およびデバッグします。

詳細は、[第 5 章「波形を使用した解析」](#)を参照してください。

注記 : 合成後およびインプリメンテーション後のシミュレーションは、非プロジェクト モードでのみサポートされます。

[13 ページの図 1-2](#) は、プロジェクト モードのシミュレーションフローを示しています。



X12984

図 1-2 : Vivado シミュレーションのプロジェクト モード フロー

非プロジェクト モード : コマンドおよびバッチ スクリプト オプション

非プロジェクト モードの場合、次が実行できます。

- 論理 RTL、合成後、またはインプリメンテーション後のシミュレーション用のテストベンチを作成または追加します。
- ソース ファイル、ライブラリ、ファイル コンパイル順を指定します。
- Vivado シミュレータ コマンドを使用してデザインを解析およびエラボレートします。

詳細は、[第 4 章「デザインのコンパイルとシミュレーション」](#) を参照してください。

- Vivado シミュレータを実行します。
- Vivado シミュレータ波形ビューアーを使用してデザインをデバッグします。

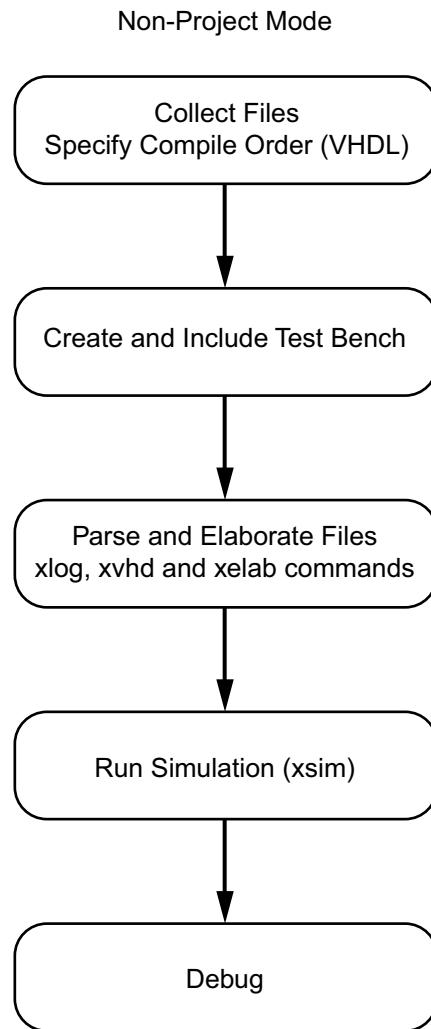
詳細は、[第 4 章「デザインのコンパイルとシミュレーション」](#) を参照してください。

詳細は、[付録 A 「Vivado IDE 外でサードパーティ シミュレータを使用したシミュレーションの実行」](#) を参照してください。



重要: 非プロジェクト モードの場合、ユーザーがソース ファイルを管理する必要があります。

図 1-3 は、非プロジェクト モードのシミュレーション フローを示しています。



X12961

図 1-3: 非プロジェクト モードのシミュレーション フロー

Vivado シミュレーション コンポーネントの理解

概要

本章では、ザイリンクス FPGA を Vivado™ Integrated Design Environment (IDE) でシミュレーションする際に必要なコンポーネントについて説明します。

注記 : シミュレーションライブラリは、Vivado シミュレータで使用するために Vivado Design Suite でプリコンパイルされます。サードパーティシミュレータを使用する場合は、必ずライブラリをコンパイルする必要があります。

シミュレーションプロセスには、次が含まれます。

- 実行するシミュレーションアクションを反映するテストベンチを作成
- 使用する必要のあるライブラリを選択して宣言
- ライブラリをコンパイル (サードパーティシミュレータを使用する場合)
- ネットリストを記述 (合成後またはインプリメンテーション後のシミュレーションを実行する場合)
- ザイリンクスデバイスのグローバルリセットとトライステートの使用を理解

次のセクションでは、これらの必要なコンポーネントについて説明します。

テストベンチまたはスティミュラスファイル

テストベンチは、次を実行するシミュレータ用に記述されるハードウェア記述言語 (HDL) コードです。

- デザインをインスタンシエート
- デザインを初期化
- スティミュラスを生成および適用
- デザイン出力結果を監視および論理精度を確認 (オプション)

テストベンチは、ファイル、波形または表示画面へのシミュレーション出力を表示するために設定できます。テストベンチの構造はシンプルにし、特定入力にスティミュラスを順次適用することができます。

テストベンチは、複雑にすることもできます。この場合、次を含めることができます。

- サブルーチン呼び出し
- 外部ファイルから読み出されるスティミュラス
- 条件付きスティミュラス

- その他のより複雑な構造

テストベンチを使用すると、対話型シミュレーションに比べて次のような利点があります。

- デザインプロセスでの反復シミュレーションが可能になります。
- テスト条件の資料が提供されます。



ヒント : GUI でターゲット シミュレータに Vivado シミュレータを選択し (第 3 章の「シミュレーション設定」参照)、[Run Simulation] をクリックします。これにより、デザインをシミュレーションするために必要なすべてのソースが含まれる project/project.sim/sim_1 ディレクトリに <testbench>.prj ファイルが作成されます。このファイルは、ファイルを集める際の開始点となります。次は、効率的なテストベンチを作成する際の推奨事項です。

- テストベンチ ファイルでメイン モジュールまたはエンティティ名に対して名前を testbench と指定します。
- Verilog テストベンチ ファイルで `timescale を常に指定します。
- テストベンチ内でデザインへのすべての入力をシミュレーション タイム 0 で初期化して、既知の値でシミュレーションが正しく開始されるようにします。
- UNISIM および SIMPRIMS ベースのシミュレーションで使用されるデフォルトのグローバル セット/リセット (GSR) パルスのため、ステイミュラス データを 100ns 後に適用します。
- グローバル セット/リセット (GSR) が解除される前にクロック ソースを開始します。

詳細は、28 ページの「グローバル リセットおよびトライステート」を参照してください。

テストベンチの詳細は、ザイリンクス アプリケーション ノート 『Writing Efficient TestBenches』(XAPP199)[参照 5] を参照してください。

シミュレーション ライブラリ

デザインにコンポーネントをインスタンシエートする際、シミュレータでコンポーネントの機能を記述したライブラリが参照されていないと、シミュレーションが正しく実行されません。

表 2-1 は、ザイリンクスの提供するシミュレーション ライブラリをリストしています。

表 2-1: シミュレーション ライブラリ

ライブラリ名	説明	VHDL ライブラリ名	Verilog ライブラリ名
UNISIM	ザイリンクス プリミティブの論理シミュレーション	UNISIM	UNISIMS_VER
UNIMACRO	ザイリンクス マクロの論理シミュレーション	UNIMACRO	UNIMACRO_VER
UNIFAST	高速シミュレーション ライブラリ	UNNIFAST	UNIFAST_VER
XILINXCORELIB	ザイリンクス コアの論理シミュレーション	XILINXCORELIB	XILINXCORELIB_VER
SIMPRIM	ザイリンクス プリミティブのタイミングシミュレーション	該当なし	SIMPRIMS_VER
SECUREIP	次のようなザイリンクス デバイス機能の論理シミュレーションおよびタイミングシミュレーション両方のシミュレーション ライブラリ • PCIe® IP • ギガビット ブラウザ	SECUREIP	SECUREIP

次に注意してください。

- シミュレーション ポイントによって異なるシミュレーション ライブラリを指定する必要があります。
- インプリメンテーション前とインプリメンテーション後のネットリストのゲート レベルセルは異なります。

表 2-2 は、各シミュレーション ポイントで必要なシミュレーション ライブラリがリストされます。

表 2-2: シミュレーション ポイントで必要なライブラリ

シミュレーション ポイント	必要なライブラリ
レジスタ トランスファー レベル (RTL)	UNISIM UNIFAST UNIMACRO XILINXCORELIB SECUREIP
合成後のシミュレーション	UNISIM (論理ネットリスト) SIMPRIMS_VER (タイミングネットリスト) SECUREIP
インプリメンテーション後のシミュレーション	UNISIM (論理ネットリスト) SIMPRIMS_VER (タイミングネットリスト) SECUREIP

注記: Verilog SIMPRIMS_VER では、UNISIM と同じソースに加え、タイミングアノテーション用に特別なブロックも使用されます。これは、UNISIM ソースコードの `ifdef XIL_TIMING でインペブルにできます。

表 2-3 は、ライブラリのディレクトリをリストしています。

表 2-3: シミュレーション ライブラリのディレクトリ

ライブラリ	HDL タイプ	ディレクトリ
UNISIM	Verilog	<Xilinx Install>/PlanAhead/data/verilog/unisims
	VHDL	<Xilinx Install>/PlanAhead/data/vhdl/unisims
UNIFAST	Verilog	<Xilinx Install>/PlanAhead/data/verilog/unifast
	VHDL	<Xilinx Install>/PlanAhead/data/vhdl/unifast
UNIMACRO	Verilog	<Xilinx Install>/PlanAhead/data/verilog/unimacro
	VHDL	<Xilinx Install>/PlanAhead/data/vhdl/unimacro
SECUREIP	Verilog	<Xilinx Install>/PlanAhead/data/secureip/ncsim/<simulator>/<simulator>_cell.list

次のセクションでは、これらのライブラリについてさらに詳細に説明します。

UNISIM ライブラリ

UNISIM は、論理シミュレーションで使用されるライブラリで、すべてのデバイス プリミティブまたは下位の構築ブロックの記述を含みます。

VHDL UNISIM ライブラリ

VHDL UNISIM ライブラリは、次のファイルに分割されています。

- コンポーネント宣言 (unisim_VCOMP.vhd)
- パッケージファイル (unisim_PKG.vhd)
- エンティティおよびアーキテクチャ宣言 (unisim_VITAL.vhd)

ザイリンクス デバイス ファミリのプリミティブはこれらのファイルで指定されています。

これらのプリミティブを使用するには、各ファイルの最初に次の 2 行を追加する必要があります。

```
library UNISIM;
use UNISIM.Vcomponents.all;
```

ライブラリをコンパイルし、シミュレータにマップする必要があります。この方法はシミュレータによって異なります。

注記 : Vivado シミュレータの場合、ライブラリのコンパイルとマップはビルドインされています。

Verilog UNISIM ライブラリ

Verilog の場合は、各ライブラリ コンポーネントを別のファイルに指定します。これで、Verilog の -y ライブラリ指定オプションでライブラリ展開が自動で実行されるようになります。

Verilog モジュール名とファイル名を大文字で指定します。

たとえば、モジュール BUFG は BUFG.v、モジュール IBUF は IBUF.v と指定します。

-y オプションの使用例は、[付録 A 「Vivado IDE 外でサードパーティ シミュレータを使用したシミュレーションの実行」](#) を参照してください。

注記: Verilog では大文字/小文字が区別されるので、UNISIM プリミティブのインスタンシエーションではすべて大文字の命名規則に従うようにしてください。

プリコンパイル ライブラリを使用する場合は、正しいシミュレータ コマンド ライン オプションを使用して、そのプリコンパイル ライブラリを指定するようにしてください。次は、Vivado シミュレータの例です。

```
-L unisims_ver
```

UNIMACRO ライブラリ

UNIMACRO は、論理シミュレーションで使用されるライブラリで、選択デバイス プリミティブのマクロ記述を含みます。UNIMACRO ライブラリは、『7 シリーズ ライブラリ ガイド (HDL 用)』(UG768)[参照 6] にリストされるデバイスマクロを含める場合は常に指定する必要があります。

VHDL UNIMACRO ライブラリ

次のライブラリ宣言を HDL ファイルの一番上に追加します。

```
library UNIMACRO;
use UNIMACRO.Vcomponents.all;
```

Verilog UNIMACRO ライブラリ

Verilog の場合は、各ライブラリ コンポーネントを別のファイルに指定します。これにより、-y ライブラリ指定オプションを使用してライブラリが自動的に拡張されるようになります。



重要: Verilog モジュール名とファイル名は大文字です。たとえば、モジュール BUFG は BUFG.v、モジュール IBUF は IBUF.v と指定します。UNISIM プリミティブ インスタンシエーションが大文字の命名規則に従うようにしてください。

ライブラリをコンパイルおよびマップする必要があります。この方法は、シミュレータによって異なります。

XILINXCORELIB ライブラリ

XILINXCORELIB ライブラリは、ザイリンクス IP カタログで作成された特定のコアを含むデザインの RTL ビヘイビアシミュレーションで使用します。

SIMPRIMS ライブラリ

SIMPRIMS ライブラリは、合成またはインプリメンテーション後に生成されるタイミング シミュレーション ネットリストをシミュレーションする際に使用します。



重要: タイミング シミュレーションは Verilog でのみサポートされます。VHDL バージョンの SIMPRIMS ライブラリはありません。

このライブラリは次のように指定します。

```
-L SIMPRIMS_VER
```

説明:

- 。 -L はライブラリ指定コマンドです。
- 。 SIMPRIMS_VER は Verilog SIMPRIM がマップされている論理ライブラリ名です。

SECUREIP シミュレーション ライブラリ

SECUREIP ライブラリは GT などの複雑な FPGA コンポーネントの論理およびタイミング シミュレーションに使用します。

注記 : IP ブロックは Vivado シミュレータで完全にサポートされています。追加で設定をする必要はありません。

Verilog LRM - IEEE Std 1364.2001 で指定された暗号化手法が使用されています。ライブラリ コンパイル プロセスでは、自動的に暗号化がされます。

Verilog コードを使用してシミュレーションを実行する場合は、SECUREIP ライブラリを参照する必要があります。このライブラリは、ほとんどのシミュレータで、次のように -L をシミュレータへの引数として使用すると参照できます。

-L SECUREIP

注記 : シミュレータでライブラリを指定するコマンド ライン オプションについては、それぞれのシミュレータの資料を参照してください。

表 2-4 は、これらのライブラリを使用する際にシミュレータ ベンダーごとに注意する点をリストしています。

表 2-4 : SECUREIP ライブラリを使用する際の考慮事項

シミュレータ名	ベンダー	要件
ModelSim SE	Mentor Graphics	デザイン入力が VHDL の場合は、混合言語ライセンスまたは SECUREIP OP が必要です。詳細はベンダーにお尋ねください。
ModelSim PE		
ModelSim DE		
QuestaSim		
IES	Cadence	輸出管理規制ライセンスが必要です。
VCS	Synopsys	VCS コマンドに -lca オプションを付ける必要があります。

VHDL SECUREIP ライブラリ

SECUREIP を VHDL で使用するには、各ファイルの最初に次の 2 行を追加する必要があります。

```
Library UNISIM;
use UNISIM.vcomponents.all;
```

Verilog SECUREIP ライブラリ

Verilog SECUREIP ライブラリは、コンパイル時に -f オプションを付けると使用できます。次は、VCS のコマンド ラインの例です。

```
vcs -f <Xilinx Install>/PlanAhead/data/secureip/VCS/vcs_cell.list.f \
<other simulation commands>
```

プリコンパイル ライブラリを使用する場合は、正しい指示子でそのプリコンパイル ライブラリを指定する必要があります。次は、Vivado シミュレータの例です。

-L SECUREIP

UNIFAST ライブラリ

UNIFAST ライブラリは RTL ビヘイビア シミュレーションで使用できるオプションのライブラリで、シミュレーション ランタイムを速くします。



重要: このモデルは、タイミング ドリブン シミュレーションには使用できません。

注記: デザインの最初の検証には UNIFAST ライブラリを使用し、完全な検証は UNISIM ライブラリを使用することをお勧めします。

シミュレーション ランタイムの高速化は、シミュレーション モードのプリミティブ機能のサブセットをサポートすることで達成されます。

注記: このシミュレーション モデルでは、サポートされない属性のみがチェックされます。

MMCME2

シミュレーション ランタイムを減らすため、高速の MMCME2 シミュレーション モデルではフルモデルに比べて次が変更されています。

1. 高速シミュレーション モデルには基本的なクロック生成ファンクションしかありません。DRP、精密位相シフト、クロック停止およびクロック カスケードなどのその他のファンクションはサポートされません。
2. 入力クロックは周波数および位相変更がなくても安定していると仮定されます。入力クロック周波数サンプルは LOCKED が high になると停止します。
3. 出力クロック周波数、位相、デューティ サイクルおよびその他の機能は入力クロック周波数とパラメーター設定から直接計算されます。

注記: 出力クロック周波数は入力から VCO クロックまで、VCO から出力クロックまでは生成されません。

4. LOCKED のアサート時間は標準および高速 MMCME2 シミュレーション モデル間で異なります。
 - 標準モデルは M および D 設定によって異なります。M および D の値が大きいと、標準 MMCME2 シミュレーション モデルのロック時間は比較的長くなります。
 - 高速シミュレーション モデルの場合、この時間は短くなります。

RAMB18E1/RAMB36E1

シミュレーション ランタイムを削減するため、高速ブロック RAM シミュレーション モデルではフルモデルと比べて、次の機能がディスエーブルになっています。

- エラー修正コード (ECC)
- 競合チェック
- カスケード モード

FIFO18E1/FIFO36E1

シミュレーション ランタイムを削減するため、高速 FIFO シミュレーション モデルでは次の機能がフルモデルから削除されています。

- ECC
- RESET、almostempty、almostfull オフセットのデザインルールチェック (DRC)
- 出力パディング: データ出力の場合は X、カウンターの場合は 1
- 最初のワード中断
- almostempty および almostfull フラグ

DSP48E1

シミュレーション ランタイムを削減するため、高速 DSP48E1 シミュレーション モデルでは次の機能がフルモデルから削除されています。

- パターン検出
- OverFlow/UnderFlow
- DRP インターフェイスのサポート

GHTE2_CHANNEL/GHTE2_COMMON

シミュレーション ランタイムを削減するため、高速 GHTE2 シミュレーション モデルでは次の機能が異なっています。

- GTH リンクは、近くのエンド リンク パートナーと遠くのエンド リンク パートナー間に違いのない Parts Per Million (PPM) レートと同期する必要があります。
- GTH を介したレイテンシは正確には記述されません。

GTXE2_CHANNEL/GTXE2_COMMON

シミュレーション ランタイムを削減するため、高速 GTXE2 シミュレーション モデルでは次の機能が異なっています。

- GTX リンクは、近くのエンド リンク パートナーと遠くのエンド リンク パートナー間に違いのない Parts Per Million (PPM) レートと同期する必要があります。
- GTX を介したレイテンシは正確には記述されません。

Verilog UNIFAST ライブラリの使用

UNIFAST モデルを使用してシミュレーションするには、次の 2 つの方法があります。方法 1 は、すべての UNIFAST モデルを使用してシミュレーションする場合に推奨される方法です。方法 2 : UNIFAST モデルと一緒に使用するモジュールを指定(アドバンス ユーザー用)

方法 1: ライブラリまたはファイルコンパイル順の使用 (推奨)

- Vivado シミュレータ

プリコンパイル ライブラリを使用した例 :

```
xelab -L unifast_ver -L unisims_ver testbench glbl -s mysim
```

インライン コンパイルを使用した例 :

```
xvlog -sourcelibdir ../verilog/src/unifast
-sourcelibext .v -sourcelibdir ../verilog/src/unisims;
xelab testbench glbl -s mysim
```

- ModelSim

プリコンパイル ライブラリを使用した例 :

```
vsim -c -L unifast_ver -L unisims_ver testbench glbl
```

インライン コンパイルを使用した例 :

```
vlog -y ../verilog/src/unifast +libext+.v -y ../verilog/src/unisims;
vsim -c testbench glbl
```

- IES

プリコンパイル ライブラリを使用した例 :

```
ncelab -libname unifast_ver -libname unisims_ver
```

インライン コンパイルを使用した例：

```
ncverilog -y ../verilog/src/unifast +libext+.v -y
../verilog/src/unisims +libext+.v \
```

- VCS

プリコンパイル ライブラリを使用した例：

```
ncelab -libname unifast_ver -libname unisims_ver
```

インライン コンパイルを使用した例：

```
ncverilog -y ../verilog/src/unifast +libext+.v -y
../verilog/src/unisims +libext+.v \
```

UNIFAST ライブラリを使用した例

```
vcs -work work -y ../Verilog/src/unifast +libext+.v -y
../verilog/src/unisims +libext+.v ../netlist.v -top testbench -top glbl
```

方法 2: Verilog でのコンフィギュレーション

方法 2 では、次を config.v ファイルで指定します。

- 最上位モジュールまたはコンフィギュレーションの名前を指定します。
例 :config cfg_xilinx;
- デザイン コンフィギュレーションを適用する名前を指定します。
例 :design testbench;
- 明示的に呼び出されないセルまたはインスタンスのライブラリ検索順を定義します。
例 :default liblist unisims_ver unifast_ver;
- 特定の CELL または INSTANCE を特定のライブラリにマップします
例 :instance testbench.inst.O1 use unifast_ver.MMCME2;)

注記 : ModelSim (vsim) の場合のみ :genblk が階層名に追加されます (例: instance testbench.genblk1.inst.genblk1.O1 use unifast_ver.MMCME2; - VSIM)。

config.v の例

```
config cfg_xilinx;
design testbench;
default liblist unisims_ver unifast_ver;
//Use fast MMCME for all MMCME blocks in design
cell MMCME2 use unifast_ver.MMCME2;
//use fast DSO48E1 for only this specific instance in the design
instance testbench.inst.O1 use unifast_ver.DSP48E1;
//If using ModelSim or Questa, add in the genblk to the name
(instance testbench.genblk1.inst.genblk1.O1 use unifast_ver.DSP48E1)
endconfig
```

VHDL UNIFAST ライブラリの使用

VHDL UNIFAST ライブラリは、Verilog と同じ基本構造を持つので、アーキテクチャまたはライブラリとも使用できます。このライブラリはテストベンチ ファイルに含めることができます。次の例では、for 呼び出しで掘り下げた階層を使用しています。

```
library unisim;
library unifast;
configuration cfg_xilinx of testbench
```

```
is for xilinx
.. for inst:netlist
... use entity work.netlist(inst);
.....for inst
.....for all:MMCME2
.....use entity unifast.MMCME2;
.....end for;
.....for O1 inst:DSP48E1;
.....use entity unifast.DSP48E1;
.....end for;
...end for;
..end for;
end for;
end cfg_xilinx;
```

シミュレーションライブラリのコンパイル

デザインをシミュレーションする前に、適用可能なライブラリをコンパイルして、シミュレータにマップしておく必要があります。

Vivado Design Suite の Tcl コンソールに Tcl コマンドの `compile_simlib` を入力し、ザイリンクスの HDL ベースのシミュレーションライブラリをサードパーティのシミュレーションベンダー用にコンパイルします。ライブラリは、通常新しいシミュレータバージョンがインストールされると、または新しいバージョンの Vivado IDE にアップデートされると、コンパイル(またはリコンパイル)されます。

詳細は、『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835)[[参照 4](#)] を参照してください。

ネットリスト生成プロセス(非プロジェクトモード)

合成済みまたはインプリメント済みデザインのシミュレーションを実行するには、ネットリスト生成プロセスを実行する必要があります。Tcl のネットリスト生成コマンドは合成済みまたはインプリメント済みデザインデータベースを使用し、デザイン全体に対して 1 つのネットリストを書き出します。

ネットリスト生成コマンドでは、SDF およびデザイン ネットリストを書き出すことができます。Vivado Design Suite では、次のネットリストコマンドが提供されています。

- `write_verilog`: Verilog ネットリスト
- `write_vhdl`: VHDL ネットリスト
- `write_sdf`: SDF 生成

これらのコマンドは、デザインプロセスのどの段階ででも論理およびタイミングシミュレーションネットリストを生成できます。



ヒント: SDF 値はデザインプロセスの早期段階(合成中など)では概算にすぎません。デザインプロセスが進むと、データベースにより多くの情報が含まれるようになり、タイミング数値の精度も増します。

シミュレーション用論理ネットリストの生成

Vivado Design Suite では論理シミュレーション用の Verilog または VHDL 構造ネットリストの書き出しがサポートされています。このネットリストの目的は、シミュレーションをタイミング情報なしで実行して、Vivado シミュレータでインプリメントされた構造ネットリストのビヘイビアが予測されるビヘイビア モデル (RTL) シミュレーションと一致するかどうかチェックすることにあります。

論理シミュレーション ネットリストは階層構造で、モジュールまたはエンティティ レベル (プリミティブおよびマクロ プリミティブを含む下位階層) に展開可能なネットリストです。これらのプリミティブは、次のライブラリに含まれます。

- Verilog シミュレーションの場合は UNISIMS_VER シミュレーション ライブラリ
- VHDL シミュレーションの場合は UNISIM シミュレーション ライブラリ

多くの場合、ビヘイビア シミュレーションで使用したのと同じテストベンチを使用して、さらに精度の高いシミュレーションを実行できます。

次は、論理シミュレーション ネットリストを生成する Verilog および VHDL 構文です。

```
write_verilog -mode funcsim <verilogNetlistName>
write_vhdl -mode funcsim <vhdlNetlistName>
```

シミュレーション用タイミングネットリストの生成

Verilog タイミング シミュレーションを使用すると、ワースト ケースの配置配線遅延の計算後に回路動作を検証できます。

多くの場合、論理シミュレーションで使用したのと同じテストベンチを使用して、さらに精度の高いシミュレーションを実行できます。

2 つのシミュレーションからの結果を比較し、デザインが最初に指定したとおりに実行されているかどうかを検証します。

タイミング シミュレーション ネットリストを生成する手順は、次のとおりです。

1. デザインのネットリスト ファイルを生成
2. タイミング遅延すべてをアノテートした遅延ファイルを生成

次のセクションでは、必要なファイルについて説明します。

注記 : Vivado IDE では、Verilog タイミング シミュレーションしかサポートされません。

次は、タイミング シミュレーション ネットリストを生成する構文です。

```
write_verilog -mode timesim -sdf_anno true <VerilogNetlistName>
```

SAIF のダンプ

SAIF (Switching Activity Interchange Format) は、EDA ツールで生成されたスイッチング アクティビティを抽出して並び替えることのできる ASCII 形式のレポートです。

このスイッチング アクティビティは、ザイリンクスの消費電力解析および最適化ツールにアノテートし戻して消費電力の測定および概算に使用できます。

Vivado シミュレータでは、Vivado の report_power コマンドおよび XPE ツールでそれぞれ使用可能な SAIF ダンプがサポートされます。SAIF ダンプは、ザイリンクス消費電力ツール用に最適化され、次の HDL タイプをダンプします。

- Verilog:
 - 入力、出力、入出力ポート
 - 内部ワイヤ宣言
- VHDL:
 - std_logic、std_ulegic、およびbit(スカラー、ベクター、配列)型の入力、出力、入出力ポート

注記: タイミングシミュレーション用の VHDL ネットリストは、Vivado で生成されませんので、VHDL ソースは RTL レベルのコード専用で、ネットリストシミュレーションには使用できません。

RTL レベルのシミュレーションの場合、ブロックレベルのポートのみが生成され、内部信号は生成されません。

SAIF ダンプの生成

log_saif コマンドを使用する前に、open_saif を呼び出す必要があります。log_saif ではオブジェクトまたは値が戻されません。オプションは、log_wave コマンドで使用されるものと同じです。

1. RTL コードを -debug typical オプションでコンパイルして、SAIF ダンプをイネーブルにします。

```
xelab -debug typical top -s mysim
```

2. 次の Tcl コマンドを使用して SAIF ダンプを開始します。

- open_saif コマンドを次のように入力して SAIF ファイルを指定します。
open_saif <saif_file_name>
- log_saif コマンドを次のように入力して生成する範囲および信号を追加します。
log_saif hdl_objects

3. run コマンドのいずれかを使用してシミュレーションを実行します。

4. 次のように入力して SAIF ファイルを閉じます。

```
close_saif
```

次は、log_saif コマンドの例です。

SAIF コマンドの例

範囲内の信号すべての SAIF を記録するには、次を入力します。

```
/tb: log_saif /tb/*
```

範囲内のポートすべての SAIF を記録するには、次を入力します。

```
/tb/UUT
```

a で始まって b で終わり、ab 間に数字が入るような名前のオブジェクトの SAIF を記録するには、次を入力します。

```
log_saif [ get_objects -filter {type == port} -regexp {^a[0-9]+b$} ]
```

current_scope および children_scope に含まれるオブジェクトの SAIF を記録するには、次を入力します。

```
log_saif [get_objects -r *]
```

current_scope に含まれるオブジェクトの SAIF を記録するには、次を入力します。

```
log_saif * or log_saif [ get_objects ]
```

scope /tb/UUT 内のポートのみの SAIF を記録するには、次を入力します。

```
log_saif [get_objects -filter { type == port } /tb/UUT/* ]
```

scope /tb/UUT 内の内部信号のみの SAIF を記録するには、次を入力します。

```
log_saif [get_objects -filter { type == internal_signal } /tb/UUT/* ]
```

Tcl シミュレーションファイルの例

```
sim.tcl:
open_saif xsim_dump.saif
log_saif /tb/dut/*
run all
close_saif
quit
```

sim.tcl ファイルを使用したシミュレーション実行

sim.tcl ファイルを使用してシミュレーションを実行するには、次を入力します。

```
xsim mysim -tcl sim.tcl
```

SAIF ファイルダンプの例

```
(SAIFFILE
  (SAIFVERSION "2.0")
  (DIRECTION "backward")
  (DESIGN )
  (DATE "Thu Nov  8 07:08:55 2012")
  (VENDOR "Xilinx, Inc")
  (PROGRAM_NAME "Vivado Simulator")
  (VERSION "2012.3")
  (DIVIDER '/')
  (TIMESCALE 1 ps)
  (DURATION 1010000)
  (INSTANCE tb
    (INSTANCE dut
      (NET
        (a (T0 500000) (T1 510000) (TX 0) (TZ 0) (TB 0) (TC 48))
        (b (T0 520000) (T1 490000) (TX 0) (TZ 0) (TB 0) (TC 52))
        (ci (T0 450000) (T1 560000) (TX 0) (TZ 0) (TB 0) (TC 44))
        (co (T0 470000) (T1 540000) (TX 0) (TZ 0) (TB 0) (TC 44))
        (sum (T0 530000) (T1 480000) (TX 0) (TZ 0) (TB 0) (TC 48))
      Brief description about the Timing/Toggle attributes in the SAIF file:
      T0: the total time the design object has value 0.
      T1: the total time the design object has value 1.
      TX: the total time the design object has unknown value.
      TZ: the total time the design object is in a floating bus state.
      Floating bus is defined as the state during which all drivers on a particular bus are
      disabled and the bus has a floating logic value.
      TB: the total time the design object is in a bus contention state.
      Bus contention is defined as the state during which two or more drivers
      simultaneously drive a bus to different logic levels.
      TC: the number of 0 to 1 and 1 to 0 transitions. This is usually referred to as the
      toggle count.
```

SDF ファイルのアノテーション

次は、SDF ファイルをアノテーションするための構文です。

```
write_sdf -process_corner fast test.sdf
```

セットアップおよびホールド チェックの実行

SDF ファイルには、指定したプロセス コーナーに基づいて異なる min および max 数値が含まれます。2 つの異なるシミュレーションを実行して、セットアップおよびホールド違反をチェックすることをお勧めします。

セットアップ チェックを実行するには、-process コーナーを slow にして SDF を作成し、SDF からの max 列を使用します。

たとえば、ModelSim では次のように指定します。

```
-sdfmax
```

ホールド チェックを実行するには、-process コーナーを fast にして SDF を作成し、SDF からの min 列を使用します。たとえば、ModelSim では次のように指定します。

```
-sdfmin
```

4 つのタイミング シミュレーションすべてを確認するには、次のように指定します。

- slow コーナー : SDFMIN および SDFMAX
- fast コーナー : SDFMIN および SDFMAX



ヒント : インターコネクト間でパルスがフィルターされないようにするには、シミュレータ設定ダイアログ ボックスで次のオプションを使用して Vivado IDE タイミング シミュレーションを実行する必要があります。

グローバル リセットおよびトライステート

ザイリンクス® デバイスには、デバイスのすべてのレジスタに接続される専用の配線および回路が含まれます。

グローバル セットおよびリセット ネット

専用グローバル セット/リセット (GSR) ネットをアサートすると、そのネットはデバイスのコンフィギュレーション直後に解除されます。すべてのフリップフロップおよびラッチはこのリセットを受信し、レジスタがどのように定義されているかによって、セットまたはリセットのいずれかになります。

コンフィギュレーション後に GSR ネットにアクセスはできますが、手動リセットの代わりに GSR 回路を使用するのをお勧めしません。これは FPGA デバイスがシステム リセットのようなファンアウトの大きい信号に対して高速バックボーン配線を提供しているためです。このバックボーン配線は専用 GSR 回路よりも高速で、GSR 信号を転送する専用グローバル配線よりも解析が簡単です。

合成後およびインプリメンテーション後のシミュレーションでは、GSR 信号には自動的に最初の 100ns 間パルスが送られ、コンフィギュレーション後に発生したリセットがシミュレーションされます。GSR パルスはオプションで早期 (合成前) 論理シミュレーションで提供されますが、すべてのレジスタをリセットするローカル リセットがデザインに含まれる場合は必要ありません。



ヒント: テストベンチを作成する場合は、GSR パルスが合成後およびインプリメンテーション後のシミュレーションで自動的に発生します。これにより、すべてのレジスタがシミュレーションの最初の 100ns 間リセット状態に保持されます。

グローバルトライステートネット

専用グローバル GSR だけではなく、専用グローバルトライステート (GTS) ネットを使用してコンフィギュレーションモード中に出力バッファーをハイインピーダンスステートに設定することもできます。すべての汎用目的出力は通常操作中にそれが標準、トライステート、双方向出力のいずれであるかによって変わります。これにより、FPGA がコンフィギュレーションされたときに出力が間違ってほかのデバイスを駆動することはありません。

シミュレーションでは、GTS 信号は通常駆動されません。GTS を駆動する回路は、合成後およびインプリメンテーション後のシミュレーションで使用できます。早期(合成前)論理シミュレーションにはオプションで追加できますが、GTS パルス幅はデフォルトで 0 に設定されます。

グローバルトライステートおよびグローバルセット/リセット信号の使用

図 2-1 は、グローバルトライステート (GTS) およびグローバルセット/リセット (GSR) 信号が FPGA デバイスでどのように使用されているかを示しています。

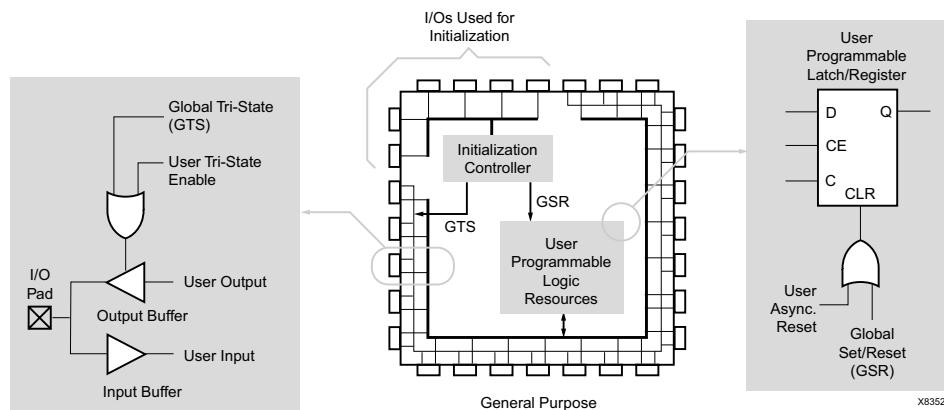


図 2-1: ビルトイン FPGA 初期化回路図

Verilog の GSR および GTS

グローバルセット/リセット (GSR) およびグローバルトライステート (GTS) 信号は \$XILINX_PLANAHEAD/data/verilog/src/glbl.v モジュールで定義されています。ほとんどの場合、GSR および GTS をテストベンチで定義する必要はありません。

lbl.v ファイルはグローバル GSR および GTS 信号を宣言し、自動的に GSR に 100 ns 間パルスを送ります。

VHDL の GSR および GTS

グローバルセット/リセット (GSR) およびグローバルトライステート (GTS) 信号は \$XILINX_PLANAHEAD/data/vhdl/src/glbl.vhd モジュールで定義されています。このコンポーネントを使用するには、それらをテストベンチにインスタンシエートする必要があります。

GLBL_VHD ファイルはグローバル GSR および GTS 信号を宣言し、自動的に GSR に 100 ns 間パルスを送ります。

次のコードは、テストベンチに GLBL_VHD をインスタンシエートし、Reset on Configuration のパルスを 90ns に変更する例を示しています。

```
GLBL_VHD inst:GLBL_VHD generic map (ROC_WIDTH => 90000; -- the value is in ps
```

ザイリンクス ライブラリを使用した RTL シミュレーション

ザイリンクス シミュレーション ライブラリは、VHDL-93 および Verilog-2001 言語規格をサポートするシミュレータを使用するとシミュレーションできます。ライブラリにはザイリンクス ハードウェア デバイスを正しくシミュレーションするのに必要な特定の遅延およびモデル情報がビルトインされています。

データ信号は、論理シミュレーションの場合でも、クロック エッジで変更しないようにしてください。シミュレータは、同じシミュレータ時刻で変更される信号同士の間にユニット遅延を追加します。データがクロックと同時に変更される場合は、シミュレータでデータ入力がクロック エッジ後に発生するようにスケジュールできます。最初のクロック エッジより前にデータをクロック入力させるようではありますが、データは次のクロック エッジまで通過しません。このような意図しないシミュレーション結果にならないようにするために、データ信号とクロック信号は同時に切り替えないようにしてください。

デルタ サイクルとレース コンディション

ザイリンクスでは、イベントベースのシミュレータをサポートしています。イベント シミュレータでは、指定したシミュレーション時刻に複数のイベントを処理できます。これらのイベントが処理中は、シミュレータはシミュレーション時間を早めることはできません。この時間は、通常「デルタ サイクル」と呼ばれます。指定したシミュレーション時間には複数のデルタ サイクルが含まれることもあります。シミュレーション時間は、処理するトランザクションがない場合にのみ早めることができます。このために、シミュレータが予測しない結果を出力する可能性があります。次の VHDL コードは、予測しない結果になる例を示しています。

予測しない結果になる VHDL コード例

```
clk_b <= clk;
clk_prcs : process (clk)
begin
  if (clk'event and clk='1') then
    result <= data;
  end if;
end process;

clk_b_prcs : process (clk_b)
begin
  if (clk_b'event and clk_b='1') then
    result1 <= result;
  end if;
end process;
```

この例には、2 つの同期プロセスがあります。

- clk
- clk_b

シミュレータは、シミュレーション時間を進める前に `clk <= clk_b` 代入を実行します。この結果、2 クロック エッジ内で発生するはずのイベントが 1 クロック エッジ内で発生し、レース コンディションになります。

このような状況を回避するには、次に注意してください。

- クロックとデータは同時に変更しないでください。出力ごとに遅延を挿入します。
- 同じクロックを使用します。
- 次の例に示すように一時的な信号を使用してデルタ遅延を強制的に使用します。

```
clk_b <= clk;
clk_prcs : process (clk)
begin
end if;
end process;
result_temp <= result;
clk_b_prcs : process (clk_b)
begin
if (clk_b'event and clk_b='1') then
  result1 <= result_temp;
end if;
end process;
```

ほとんどのイベントベース シミュレータは、デルタ サイクルを表示できるようになっています。シミュレーション問題をデバッグする場合はこの表示を使用してください。

推奨されるシミュレーション解像度

ザイリンクスでは、シミュレーション解像度 1ps でシミュレーションを実行することをお勧めしています。DCMなどのザイリンクスのプリミティブ コンポーネントの中には、論理またはタイミング シミュレーションのいずれかで正しく動作するためには 1ps の解像度が必要なものがあります。

ザイリンクス シミュレーション モデルではこれより荒い解像度を使用してもシミュレータ パフォーマンスが良くなることはありません。多くのシミュレーション時間がデルタ サイクルで使われるため、デルタ サイクルがシミュレーション解像度によって影響を受けることはなく、シミュレーションパフォーマンスはあまり取得できません。

ザイリンクスでは、fs のようなより精度の高い解像度で実行することをお勧めしていません。シミュレータの中には数値を丸めたり、切り捨てたりするものもあります。

テスト ツールはタイミングをすべて一番近いピコ秒 (ps) に対してのみ計測するので、最小解像度としてはピコ秒が使用されます。ザイリンクスでは、すべての HDL シミュレーションで ps を使用することをお勧めしています。

ASYNC_REG 制約の使用

ASYNC_REG 制約には、次のような特徴があります。

- デザインの非同期レジスタを識別します。
- これらのレジスタの X 伝搬をディスエーブルにします。

ASYNC_REG 制約は、次を使用してフロントエンド デザインのレジスタに適用できます。

- HDL コードの属性
 - または
 - ザイリンクス デザイン制約 (XDC) の制約

ASYNC_REG が適用されているレジスタは、タイミング シミュレーション中に前の値を保持し、シミュレーションに X を出力しません。

タイミング違反エラーは、まだ発生する可能性があります。新しい値もクロック入力された可能性があるので、注意してください。

ASYNC_REG 制約は CLB および入出力ブロック (IOB) レジスタおよびラッチにのみ適用できます。



推奨: 非同期データのクロック入力を回避できない場合は、IOB または CLB レジスタにのみ ASYNC_REG 制約を適用することをお勧めします。RAM、シフト レジスタ LUT (SRL)、またはその他の同期エレメントへの非同期信号へクロック入力しても、決定的な結果にはなりにくいので、入力しないようにしてください。

まず、レジスタ、ラッチまたは FIFO の非同期信号を正しく同期してから、RAM、シフト レジスタ LUT (SRL) またはその他の同期エレメントへ書き込むことをお勧めします。詳細は、『Vivado Design Suite チュートリアル: 制約の使用』(UG945)[\[参照 9\]](#) を参照してください。

同期エレメントの X 伝搬のディスエーブル

タイミングシミュレーション中にタイミング エラーが発生すると、ラッチ、レジスタ、RAM またはその他の同期エレメントはデフォルトで X をシミュレータに出力します。

これは、実際の出力値が未知のためです。レジスタの出力は、次のようにになります。

- 前の値を保持
- 新しい値へアップデート
- 同期エレメントのクロック供給後しばらく経つまで値が決定されないメタステーブル状態

この値は決定できないので、正確なシミュレーション結果は保証できません。エレメントは未知の値を示す X を出力します。X 出力は次のクロック サイクルまでそのまま保持されます。出力は、別の違反が発生しない場合、次のクロック付きの値でアップデートされます。

X の生成はシミュレーションに大きく影響します。たとえば、1 つのレジスタで生成された X は続くクロック サイクルのほかのレジスタに伝搬される可能性があります。これにより、テスト中のデザインの大部分が未知の状態になってしまいます。

これを修正するには、次の手順に従います。

- 同期パスでは、パスを解析し、このパスおよびほかのパスに関するタイミング問題を修正し、問題なく動作する回路にします。
- 非同期パスでは、タイミング違反をほかの方法で回避できない場合にのみ、タイミング違反中に同期エレメントの X 伝搬をディスエーブルにします。

X 伝搬がディスエーブルになると、レジスタの出力には前の値が保持されます。実際のシリコンでは、レジスタは新しい値に変更されることもあります。

X 伝搬をディスエーブルにすると、シリコン ビヘイビアとは一致しないシミュレーション結果になることがあります。



注意: このオプションを使用するには、注意が必要です。タイミング違反をこれ以外の方法で回避できない場合にのみ使用してください。

コンフィギュレーション インターフェイスのシミュレーション

このセクションでは、次のコンフィギュレーション インターフェイスのシミュレーションについて説明します。

- JTAG シミュレーション
- SelectMAP シミュレーション

JTAG シミュレーション

BSCAN コンポーネントのシミュレーションはすべてのデバイスでサポートされます。

シミュレーションでは、JTAG ポートと一部の JTAG 操作コマンドがサポートされます。スキャン チェーンへのインターフェイスも含め、JTAG インターフェイスは完全にはサポートされていません、このインターフェイスをシミュレーションするには、次を実行します、

1. BSCANE2 コンポーネントをインスタンシエートし、デザインに接続します。
2. JTAG_SIME2 コンポーネントをデザインではなく、テストベンチにインスタンシエートします。

この結果、次になります。

- 外部 JTAG 信号 (TDI、TDO、TCK など) へのインターフェイス
- BSCAN コンポーネントへの通信チャネル

コンポーネント間の通信は VPKG VHDL パッケージ ファイルまたは Verilog グローバル モジュール gbl で発生します。従って、特定の JTAG_SIME2 とデザイン間、または特定の BSCANE2 シンボル間に暗示的接続は必要ありません。

ステイミュラスはテストベンチ内の特定の JTAG_SIME2 コンポーネントから駆動および表示できます。これにより、JTAG/BSCAN の動作が理解できます。これらのコンポーネント両方のインスタンシエーション テンプレートは、Vivado Design Suite テンプレートおよびそのデバイスのライブラリ ガイドから入手できます。

SelectMAP シミュレーション

インスタンシエーション テンプレートを含むコンフィギュレーション シミュレーション モデル (SIM_CONFIG2) を使用すると、サポートされるコンフィギュレーション インターフェイスがシミュレーションできるようになります。最終的に DONE ピンが High になります。これは、サポートされるコンフィギュレーション インターフェイスでサポートされるデバイスがステイミュラスにどのように反応するかを示すモデルです。表 2-5 には、サポートされるインターフェイスおよびデバイスをリストしています。

表 2-5: サポートされるコンフィギュレーション デバイスおよびモード

デバイス	SelectMAP	シリアル	SPI	BPI
7 シリーズおよび Zynq-7000™ AP SoC デバイス	○	○	X	X

このモデルでは制御信号の動作とビット ファイルのダウンロードが処理されます。

CRC、IDCODE のような内部レジスタ設定とステータス レジスタが含まれます。Sync Word がデバイスに入力され、スタートアップ シーケンスで処理されるところも監視できます。

図 2-2 は、システムをハードウェアからシミュレーション環境にマップする方法を示しています。コンフィギュレーション プロセスの概要は、各デバイスのコンフィギュレーション ユーザー ガイドに記載されています。これらのガ

イドには、コンフィギュレーション シーケンスおよびコンフィギュレーション インターフェイスに関する情報が含まれます。

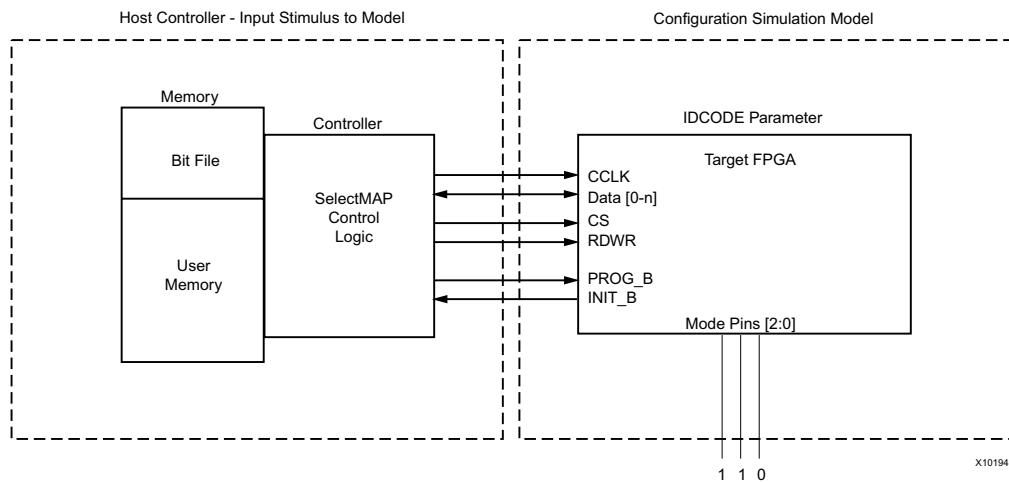


図 2-2: モデルの相互作用を示すブロック図

システム レベルの記述

`SIM_CONFIGE2` モデルを使用すると、ハードウェアが使用可能になるよりも前にコンフィギュレーション インターフェイス制御ロジックをテストでき、デバイス全体をシミュレーションでき、次のアプリケーションのシステム レベルで使用できます。

- ワイヤ、制御信号処理、データ入力アライメントが正しく実行されるよう、コンフィギュレーション ロジックを制御するためにプロセッサを使用するアプリケーション
- データアライメントが正しく実行されるよう CS (SelectMAP Chip Select) または CLK 信号を使用してデータ読み込みプロセスを制御するアプリケーション
- SelectMAP ABORT または Readback を実行する必要のあるシステム

このモデルに関連する ZIP ファイルは、次から入手できます。

http://www.xilinx.com/txpatches/pub/documentation/misc/config_test_bench.zip

この ZIP ファイルには、SelectMAP ロジックを実行するプロセッサをシミュレーションするサンプル テストベンチが含まれます。これらのテストベンチには、SelectMAP インターフェイスを制御するプロセッサをエミュレートする制御ロジックが含まれ、フルコンフィギュレーション、ABORT、および IDCODE とステータス レジスタのリードバックなどの機能があります。

シミュレーションされたホスト システムには、ファイル配信および制御信号管理などの機能が必要です。これらの制御信号はデバイスのコンフィギュレーション ユーザー ガイドで説明されるようにデザインする必要があります。

`SIM_CONFIGE2` モデルは、ビット ファイルがデバイスに読み込まれたときに、コンフィギュレーション プロセッサー中にデバイス内で何が発生しているかも示します。BIT ファイルのダウンロード中、モデルは各コマンドを実行し、ハードウェアの変更を監視するレジスタ設定を変更します。

CRC レジスタを監視して、CRC 値が累積していくのを監視できます。このモデルは、デバイスがさまざまなコンフィギュレーション状態を通過していくのに合わせて、設定されたステータス レジスタ ビットがどのように変更するかも示します。

モデルを使用したデバッグ

SIM_CONFIGE2 モデルには、正しいコンフィギュレーション例が含まれます。問題が発生したときにこの例を使用すると、デバッグがしやすくなります。

ステータス レジスタは iMPACT ツールを使用すると、JTAG を介して読み出すことができます。レジスタには、デバイスの現在のステータスに関する情報が含まれるので、リソースのデバッグに役立ちます。ボードで問題がある場合は、まず iMPACT にステータス レジスタを読み込んでデバッグを始めます。

ステータス レジスタが読み込まれたら、シミュレーションにマップして、デバイスのコンフィギュレーション段階を指定します。

たとえば、GHIGH ビットはデータの読み込み後に設定されます。このビットが設定されていないと、データの読み込みは完了しません。スタートアップ シーケンスでリリースされる BitGen で設定された GTW、GWE および DONE 信号も監視できます。

SIM_CONFIGE2 モデルでは、エラー挿入もできます。データの読み込みが一時停止され、問題がそのままで再開されると、アクティブな CRC ロジックで問題が検出されます。BIT ファイルに手動で挿入されたビット フリップも検出され、デバイスがこのエラーを処理するのと同じように処理されます。

サポートされる機能

サポートされる各コンフィギュレーション インターフェイス同士の関係については、各デバイスのコンフィギュレーション ユーザー ガイドを参照してください。表 2-6 は、コンフィギュレーション ユーザー ガイドで説明されるサポートされる機能を示しています。

SIM_CONFIGE2 モデルには、次のような特徴があります。

- コンフィギュレーション データのリードバックはサポートされません。
- 提供されるコンフィギュレーション データは格納されませんが、CRC 値は計算されます。
- リードバックは特定のレジスタでのみ実行でき、有効なコマンド シーケンスおよび信号処理がデバイスに対して提供されるようになります。
- リードバック データ ファイルが生成されるようにするためのモデルではありません。

表 2-6: モデルをサポートするスレーブ SelectMAP およびシリアルの機能

スレーブ SelectMAP およびシリアルの機能	サポートあり
マスター モード	X
デイジーチェーン - スレーブ パラレル デイジーチェーン	X
SelectMAP データ読み込み	○
継続 SelectMAP データ読み込み	○
非継続 SelectMAP データ読み込み	○
SelectMAP ABORT	○
SelectMAP リコンフィギュレーション	X
SelectMAP データ順序付け	○
リコンフィギュレーションおよびマルチブート	X
コンフィギュレーション CRC - コンフィギュレーション中の CRC	○
コンフィギュレーション CRC - コンフィギュレーション後の CRC	X

シミュレーションでのブロック RAM の競合チェックのディスエーブル方法

ザイリンクス ブロック RAM メモリは True デュアルポート RAM で、どちらのポートもいつでもどのメモリ ロケーションからでもアクセスできますが、同じアドレス空間が同時に読み出しあり書き込みされると、ブロック RAM アドレスの競合が発生します。これらは有効な競合で、読み出しポートから読み出されるデータが有効ではないために発生します。

ハードウェアでは、読み出された値が古いデータか、新しいデータか、古いデータと新しいデータの混合したものである可能性があります。シミュレーションでは、読み出される値が未知のために、X が出力されます。ブロック RAM の競合の詳細については、デバイスのユーザー ガイドを参照してください。

一部のアプリケーションでは、この状態が回避または設計変更できないこともあります。この場合、ブロック RAM はこれらの違反を検出しないように設定できます。これは、ブロック RAM プリミティブのジェネリック (VHDL) またはパラメーター (Verilog) の `SIM_COLLISION_CHECK` で制御できます。

SIM_COLLISION_CHECK 文字列

表 2-7 は、競合が発生したときのシミュレーション ビヘイビアを制御する `SIM_COLLISION_CHECK` に合わせて使用できる文字列オプションを示しています。

表 2-7: SIM_COLLISION_CHECK 文字列

文字列	書き込み競合メッセージ	出力での X の書き込み
ALL	○	○
WARN_ONLY	○	X 競合発生時にのみ適用され、同じアドレス空間の次に続く読み出しが X を出力する可能性あり
GENERATE_X_ONLY	X	○
なし	X	X 競合発生時にのみ適用され、同じアドレス空間の次に続く読み出しが X を出力する可能性あり

インスタンス レベルで `SIM_COLLISION_CHECK` を使用すると、各ブロック RAM インターフェイスの設定を変更できます。

Vivado IDE でのシミュレーションの実行およびソースコードのデバッグ

概要

本章では、Vivado™ Integrated Design Environment (IDE) シミュレータのグラフィカルユーザー インターフェイス (GUI) について説明します。このシミュレータは、Vivado IDE で混合言語シミュレータを設定する際に選択できます。本章では、53 ページの「ソース レベルでのデバッグ」についても説明します。

Vivado IDE には、ビヘイビア シミュレーションに Vivado シミュレータを使用するための統合シミュレーション環境が提供されています。

シミュレーション設定

Flow Navigator で [Simulation Settings] をクリックすると、Vivado IDE でシミュレーション設定ができます。図 3-1 は Flow Navigator の [Simulation] を示しています。



図 3-1 : Flow Navigator の [Simulation]

- [Simulation Settings] : Vivado シミュレータの選択および設定ができる [Simulation Settings] ダイアログ ボックスを開きます。
 - [Run Simulation] : シミュレーション設定に基づいてデザインをコンパイル、エラボレート、シミュレーションするコマンド オプションを設定します。デザインの合成前にシミュレーションを実行すると、Vivado シミュレータではビヘイビア シミュレーションが実行され、波形ビューが開き (38 ページの図 3-2)、信号およびバスの値の付いた HDL オブジェクトがデジタルまたはアナログ形式で表示されます。
- 各デザイン段階 (合成後とインプリメンテーション後) ごとに、論理シミュレーションおよびタイミングシミュレーションを実行できます。
- [Open Static Simulation] : 前に作成した波形コンフィギュレーション (WDB) ファイルを開くことができます。詳細は、[第 5 章の「前のシミュレーション設定からのシミュレーションデータの表示 \(スタティック シミュレーション\)」](#) を参照してください。

[Simulation Settings] を選択すると、図 3-2 に示すような [Project Settings] ダイアログ ボックスが開きます。

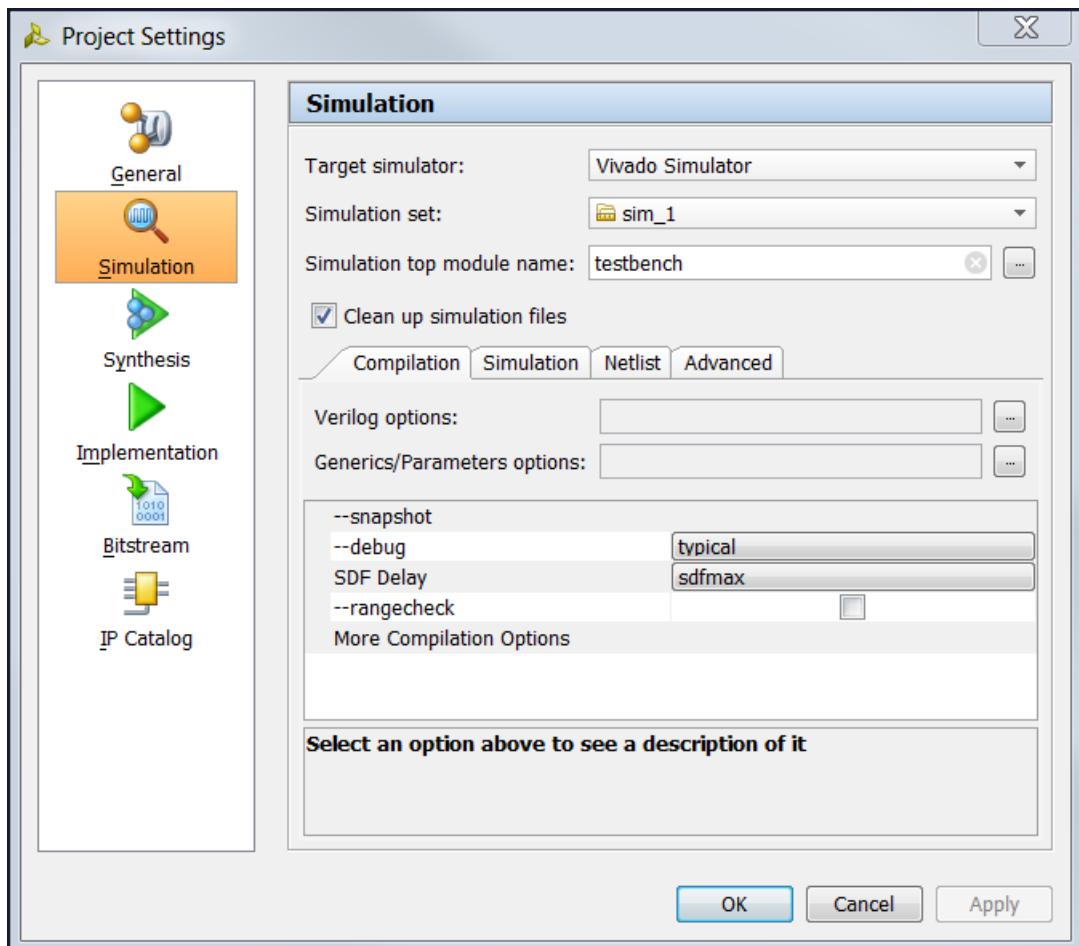


図 3-2 : [Project Settings] ダイアログ ボックスのターゲット シミュレータ オプション

[Project Settings] ダイアログ ボックスには、次のオプションが表示されます。

- [Target Simulator] : シミュレーターをビヘイビア シミュレーションまたはタイミング シミュレーション用に起動するように指定します。使用できるオプションは次のとおりです。
 - [Vivado Simulator] : ターゲット シミュレータを Vivado シミュレータに指定します。

重要 : Vivado シミュレータにはプリコンパイル ライブラリが含まれているので、ライブラリ ディレクトリを指定する必要はありません。

- [QuestaSim/ModelSim] : ターゲット シミュレーターを Mentor Graphics® ModelSim または Questa® Advanced Simulator ツールに指定します。

シミュレーターはインストールされて \$PATH パスに表示されていないと、シミュレーションを実行したときに起動されません。

[Simulation] ページには、[QuestaSim/ModelSim] を選択した場合、ライブラリ ディレクトリ用の追加フィールドが表示されます。

[Simulation] ページでは、次が設定できます。

- [Simulation set] :既存のシミュレーション設定を選択するか、[Create simulation set] を使用します (43 ページの「シミュレーション セットの編集」を参照)。
- [Simulation top module name] :シミュレーション最上位モジュールを設定します。
- [Compiled library location] :[QuestaSim/ModelSim] を [Target simulator] で選択すると、このフィールドにプリコンパイル済みのライブラリが表示されます。
 - シミュレーションライブラリを指定する方法については、17 ページの「シミュレーション ライブラリ」を参照してください。
 - QuestaSim/ModelSim に関する詳細は、付録 A 「Vivado IDE 外でサードパーティ シミュレータを使用したシミュレーションの実行」を参照してください。
- [Clean up simulation files] :オンにすると、sim ディレクトリに格納されていないシミュレーション ファイルが削除されます。
- [Compilation View] タブ :よく使用されるコンパイル オプションのオプションを参照および選択できます。
 - [Verilog options] : 使用する Verilog コードのバージョンを選択します。
 - [Generics/Parameters options] :必要な VHDL ジェネリックまたは Verilog パラメーターを選択します。
 - [Command options] :
 - `-debug` :シミュレーションを高速に実行するため、デフォルトで `typical` に設定されています。その他に `off` および `all` というオプションもあります。 `typical` 設定には、`wave` および `line` オプションが含まれます。
 - `[SDF Delay]` :デフォルトで `sdfmax` に設定されます。ドロップダウン メニューから `sdfmin` を選択することもできます。
 - `-rangecheck` :シミュレーションを高速に実行するため、デフォルトでオフになっています。
- [Simulation] タブ :使用可能なシミュレーションオプションを提供します。オプションをクリックすると、その説明が表示されます。[Simulation] タブでは次のオプションを選択できます。
 - [Simulation Run Time] : シミュレーションが開始される際に自動で実行されるシミュレーション時間を指定します。デフォルトは 1000 ns です。
 - `-view` :前に保存した波形コンフィギュレーション (WCFG) ファイルを開くことができます。波形コンフィギュレーションは、波形ウィンドウで表示する HDL オブジェクトのリストです。

オプションをクリックすると、その説明が表示されます。



ヒント : 最初の run からの WCFG ファイルには、指定した信号を仕切り付きのアナログまたはデジタル波形で保存できます。後の run で `-view` オプションを使用して GUI を開くと、この WCFG ファイルが開くので、シミュレーション波形を設定する時間が節約できます。

- [Netlist] タブ : 図 3-3 のように、シミュレーションの write_verilog ネットリスト設定がリストされます。オプションをクリックすると、その説明が表示されます。

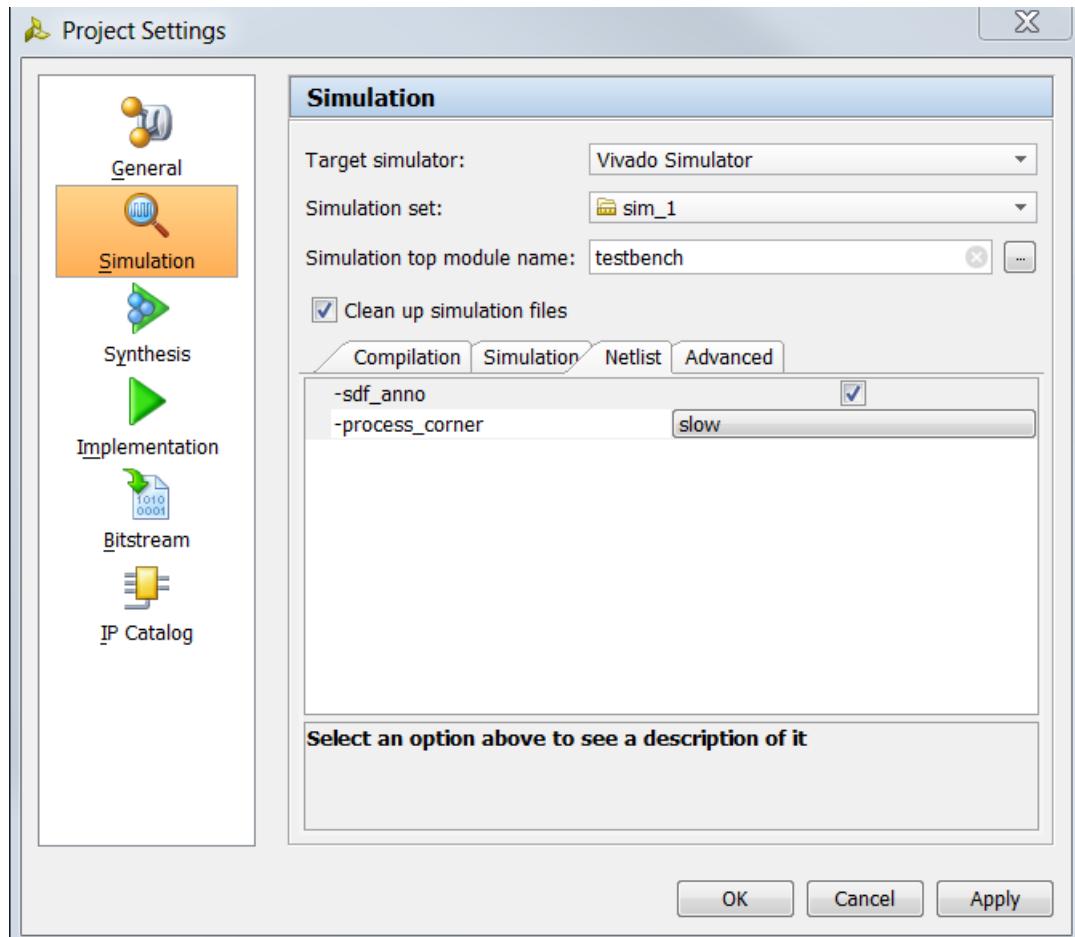


図 3-3 : [Netlist] ビュー

- ネットリストの write_verilog オプションは、次のとおりです。
 - sdf_anno : オンにすると、SDF アノテーションがイネーブルになります。
 - process_corner : デフォルトで slow に設定されます。その他のドロップダウン オプションには、fast があります。
- [Advanced] タブ : 41 ページの図 3-4 に示すように、シミュレーションにすべてのデザイン ソースを含めるオプションがあります。オフにすると、シミュレーションするファイルのみを含めることができます。

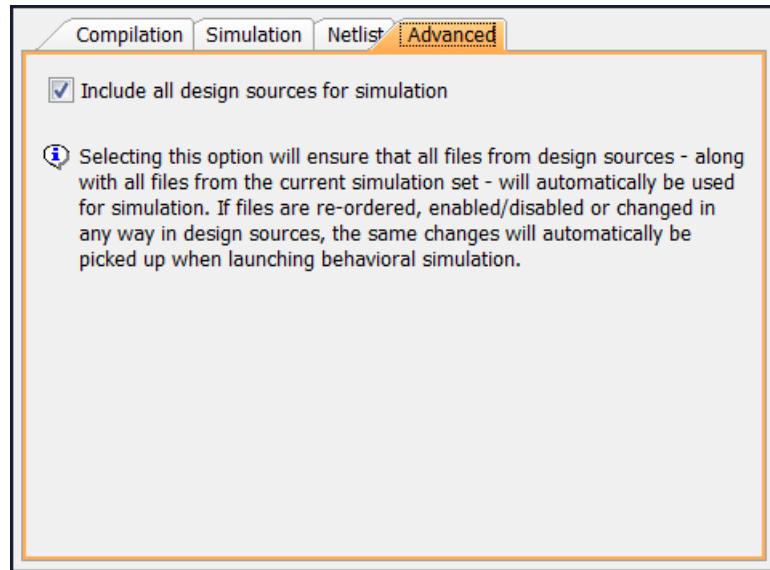


図 3-4: アドバンス シミュレーション設定

シミュレーション ソースの管理

Vivado Design Suite には、シミュレーション ソースを追加できます。シミュレーション セットには、シミュレーションのステイミュラスとして使用する HDL ベースのテストベンチ ファイルおよびその他のソース ファイルが含まれます。

Vivado Design Suite では、シミュレーション ソース ファイルはシミュレーション セットに格納され、[Sources] ビューにフォルダーとして表示されます。リモートのものを参照するか、ローカル プロジェクト ディレクトリに保存されているものを使用できます。

シミュレーション セットにより、デザインの異なる段階に異なるソースを使用できます。たとえば、エラボレート済みデザインまたはデザインのモジュールのビヘイビアーシミュレーション用にステイミュラスを供給するシミュレーション ソースを使用し、インプリメント済みデザインのタイミング シミュレーション用にステイミュラスを供給する別のテストベンチを使用できます。

シミュレーション ソースをプロジェクトに追加する際、ソースを追加するシミュレーション ソース セットを指定できます。

シミュレーション ソース ファイルの追加または作成

シミュレーション ソースをプロジェクトに追加するには、次の手順に従います。

1. [File] → [Add Sources] をクリックします。

Add Sources ウィザードが表示されます。

2. [Add or Create Simulation Sources] をオンにし、[Next] をクリックします。

図 3-5 に示す [Add or Create Simulation Sources] ページが表示されます。

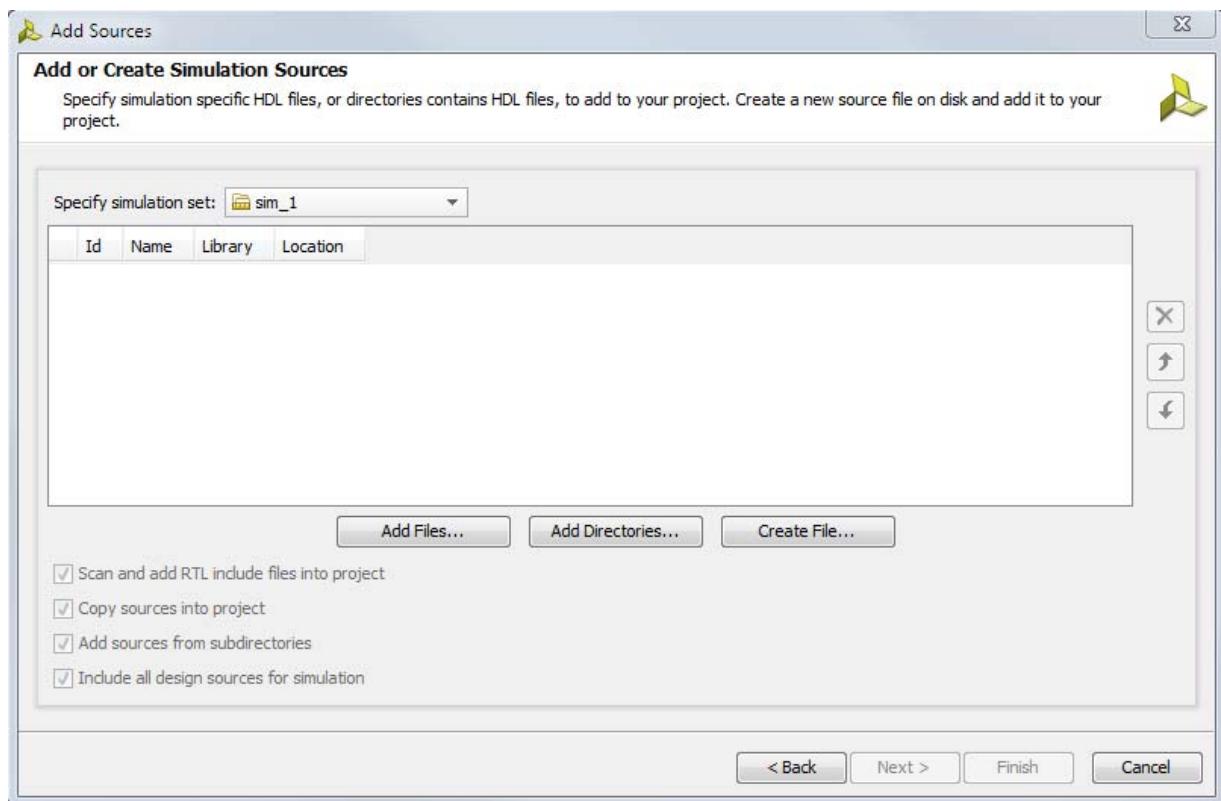


図 3-5 : Add Sources ウィザード : [Add or Create Design Sources] ページ

[Add or Create Simulation Sources] ページでは、次のオプションが設定できます。

- [Specify Simulation Set] : テストベンチ ファイルを含むシミュレーション セットの名前とディレクトリ (デフォルトは sim_1, sim_2...) を入力します。ドロップダウン メニューから [Create Simulation Set] コマンドを選択すると、新しいシミュレーション セットが定義できます。複数のシミュレーション セットが使用可能な場合は、Vivado シミュレータにより、どのシミュレーション セットがアクティブな (現在使用中の) セットであるかが表示されます。
- [Add Files] : プロジェクトに追加するシミュレーション ソース ファイルを選択します。
- [Add Directories] : 選択したディレクトリに含まれるすべてのシミュレーション ソース ファイルを追加します。指定したディレクトリにある有効なソース ファイルがすべてプロジェクトに追加されます。
- [Create File] : 新規シミュレーション ファイルを作成する [Create Source File] ダイアログ ボックスが開きます。プロジェクト ソース ファイルの詳細は、『Vivado Design Suite ユーザー ガイド : Vivado IDE の使用』(UG893)[参照 3] の「ソース ファイルの追加と作成」を参照してください。

ダイアログ ボックスには、次のようなボタンがあります。

- [Remove] : 選択したソース ファイルを削除します。 
- [Move Selected File Up] : ファイルをリストの上方向へ移動します。 
- [Move Selected File Down] : ファイルをリストの下方向へ移動します。 

ウィザードのチェックボックスには、次のオプションがあります。

- [Scan and add RTL include files into project] : 追加した RTL ファイルをスキャンして、参照されたインクルード ファイルを追加します。
- [Copy sources into project] : ソース ファイルをプロジェクト ディレクトリにコピーします。プロジェクトではローカルにコピーされたバージョンが使用されます。

[Add Directories] ボタンをクリックしてソース ファイルのディレクトリを追加した場合は、ファイルがローカルのプロジェクトにコピーされる際にディレクトリ構造もそのまま保持されます。

- [Add sources from subdirectories] : [Add Directories] で指定したディレクトリのサブディレクトリに含まれるソース ファイルをすべて追加します。
- [Include all design sources for simulation] : シミュレーション用にデザイン ソースすべてを含めます。

シミュレーション セットの編集

シミュレーション セットを変更するには、次を実行します。

1. [Sources] ビューで [Simulation Sources] を右クリックし、[Edit Simulation Sets] をクリックします(図 3-6)。

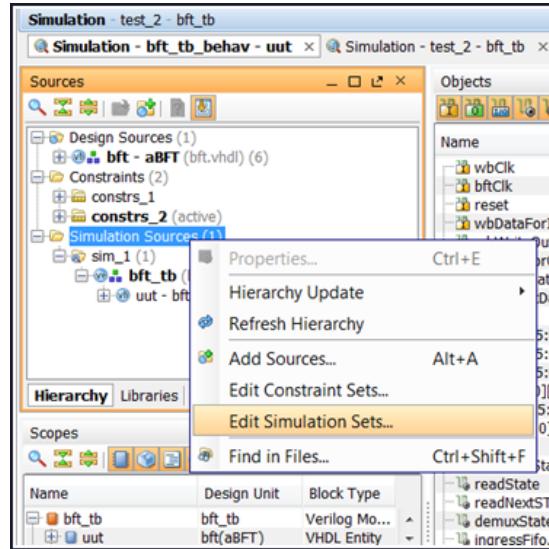


図 3-6 : [Edit Simulation Sets] オプション

42 ページの図 3-5 に示す [Add or Create Simulation Sources] ページが表示されます。

2. [Add Files] をクリックしてファイルを選択します。

これで、プロジェクトに関するソースが新しく作成されたシミュレーション セットに追加されます。

3. 必要に応じてほかのファイルも追加します。

選択したシミュレーション セットがアクティブなデザイン run に使用されます。



重要: 前に定義されたシミュレーションセットのコンパイルおよびシミュレーション設定は、新しく定義されたシミュレーションセットには適用されません。

Vivado シミュレータの使用

Flow Navigator から [Run Simulation] をクリックし、Vivado シミュレータの GUI を表示します (44 ページの図 3-7)。Vivado IDE のメニューの詳細は、『Vivado Design Suite ユーザー ガイド : Vivado IDE の使用』(UG893)[[参照 3](#)] を参照してください。

Vivado シミュレータの GUI の主なコンポーネントは、次のとおりです。

1. 「メイン ツールバー」
2. 「[Run] メニュー」
3. 「[Objects] ビュー」
4. 「[Simulation] ツールバー」
5. 「波形オブジェクト」
6. 「波形ビュー」
7. 「[Scopes] ビュー」
8. 「[Sources] ビュー」

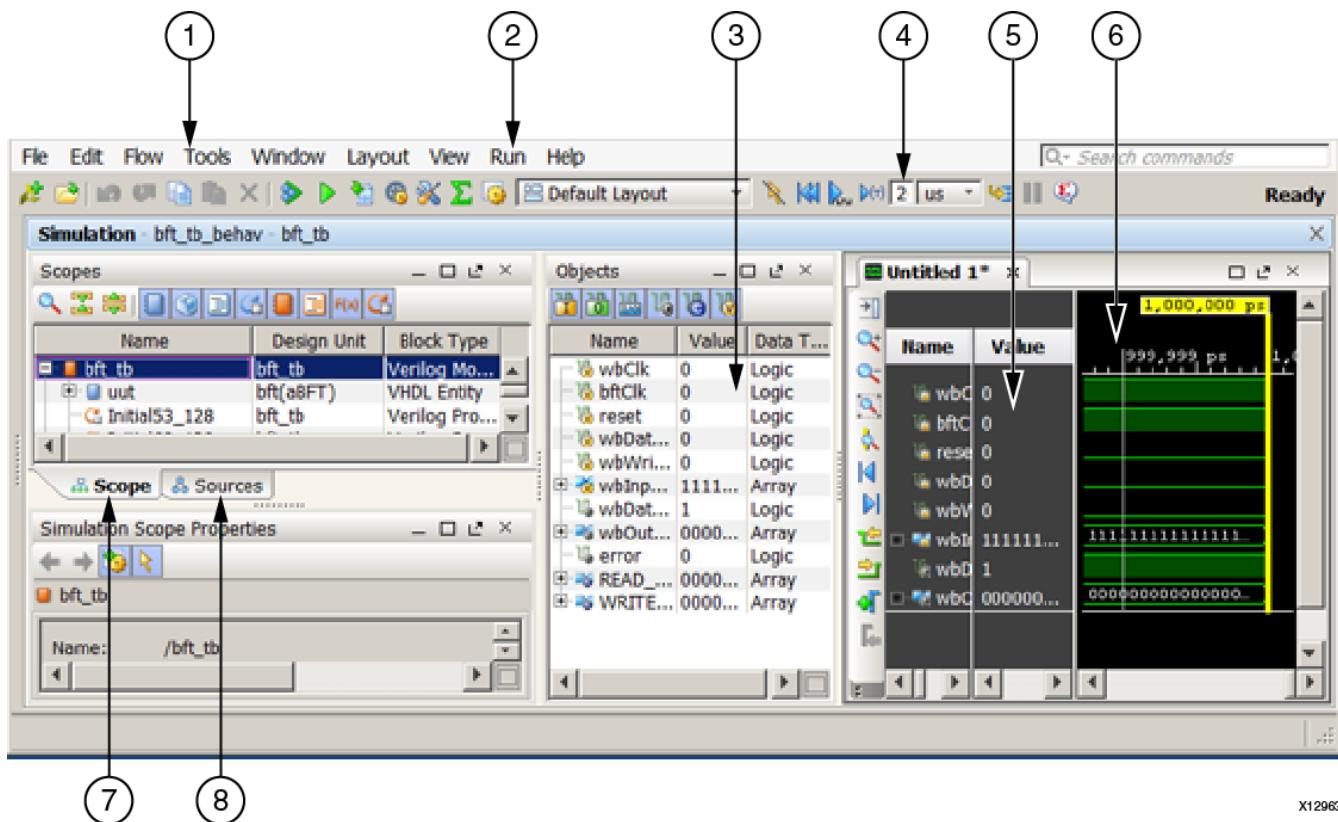


図 3-7 : Vivado シミュレータの GUI

メイン ツールバー

Vivado IDE で最もよく使用されるコマンドに 1 クリックでアクセスできます。オプションの上にカーソルを置くと、詳細を示すツール ヒントが表示されます。

[Run] メニュー

メニューは Vivado IDE と同じですが、シミュレーションを実行した後には [Run] メニューが追加されます。

シミュレーションの [Run] メニューは、図 3-8 のようになります。

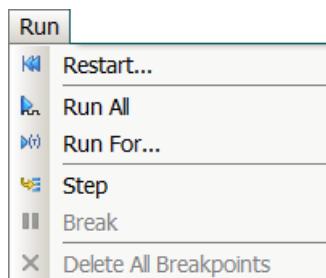


図 3-8: シミュレーションのメニュー オプション

[Simulation] メニュー オプションは次のとおりです。

- [Restart] : 既存のシミュレーション時間を 0 に戻して再開します。
- [Run All] : 開いているシミュレーションを最後まで実行します。
- [Run For] : 実行するシミュレーション時間を指定します。
- [Step] : 次の HDL ソース ラインまでシミュレーションを実行します。
- [Break] : 実行中のシミュレーションを停止します。
- [Delete All Breakpoints] : すべてのブレークポイントを削除します。

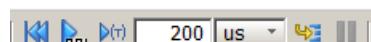
[Objects] ビュー

HDL オブジェクトが表示されます。ビューの詳細は、『Vivado Design Suite ユーザー ガイド : Vivado IDE の使用』(UG893)[[参照 3](#)] を参照してください。

HDL オブジェクトの横には、言語またはプロセス タイプが表示されます。このビューにはシミュレーション オブジェクトの名前、デザイン ユニット、ブロック タイプなどがリストされます。[Object] ボタンの上にカーソルを置くと、詳細が表示されます。

[Simulation] ツールバー

Vivado シミュレータを実行すると、シミュレーション専用のツールバーが Vivado ツールバーの横に表示されます。



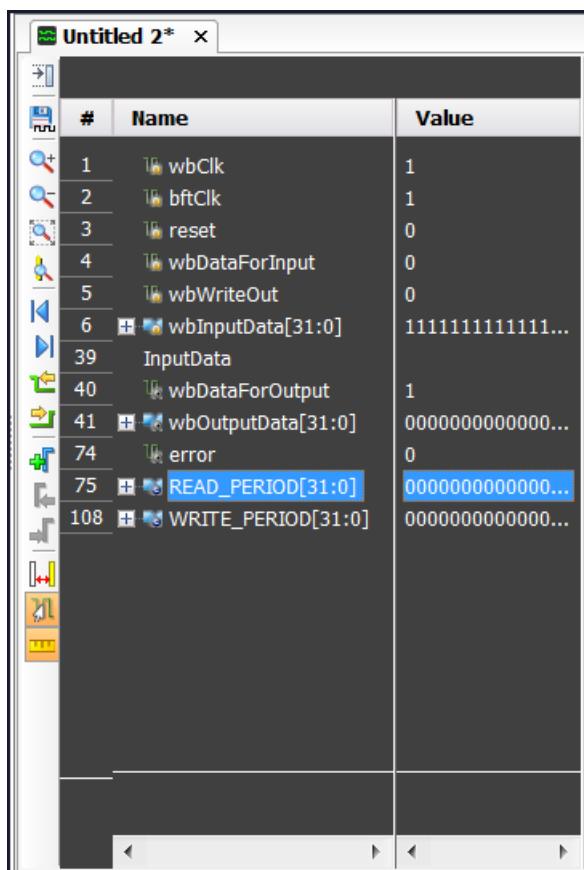
ツールバー ボタンの上にカーソルを置くと、そのボタンの機能を示すツール ヒントが表示されます。ボタンは、図 3-8 と同じ名前になります。

波形オブジェクト

Vivado IDE 波形ビューは、多くの Vivado Design Suite ツールで共通して使用されます。

波形の詳細は、[第 5 章「波形を使用した解析」](#)を参照してください。

次の図 3-9 は、波形コンフィギュレーションに含まれる波形オブジェクトの例を示しています。



The screenshot shows the Vivado IDE's Waveform View window titled "Untitled 2*". The main area is a table with columns for "#", "Name", and "Value". The table lists the following HDL objects:

#	Name	Value
1	wbClk	1
2	bftClk	1
3	reset	0
4	wbDataForInput	0
5	wbWriteOut	0
6	wbInputData[31:0]	11111111111111...
39	InputData	
40	wbDataForOutput	1
41	wbOutputData[31:0]	00000000000000...
74	error	0
75	READ_PERIOD[31:0]	00000000000000...
108	WRITE_PERIOD[31:0]	00000000000000...

図 3-9: 波形コンフィギュレーションの HDL オブジェクト

波形ビューには、HDL オブジェクトとその値および波形に加え、グループ、仕切り、仮想バスなどの HDL オブジェクトを整頓するためのアイテムも表示されます。

HDL オブジェクトおよび整頓されたアイテムは「波形オブジェクト」と呼ばれます。波形ビューの波形部分には、カーソル、マーカー、時間軸などの時間尺度のアイテムも表示されます。

Vivado IDE ではシミュレーション中に波形コンフィギュレーションの HDL オブジェクトがトレースされるので、波形コンフィギュレーションを使用してシミュレーション結果を検証します。デザイン階層および波形は波形コンフィギュレーションの一部ではなく、別の WDB データベース ファイルに格納されます。

波形ビューア

シミュレータを起動すると、シミュレーションでトレース可能な HDL オブジェクトの最上位モジュールを含む新しい波形コンフィギュレーションを表示する波形ビューアがデフォルトで表示されます(図 3-10)。

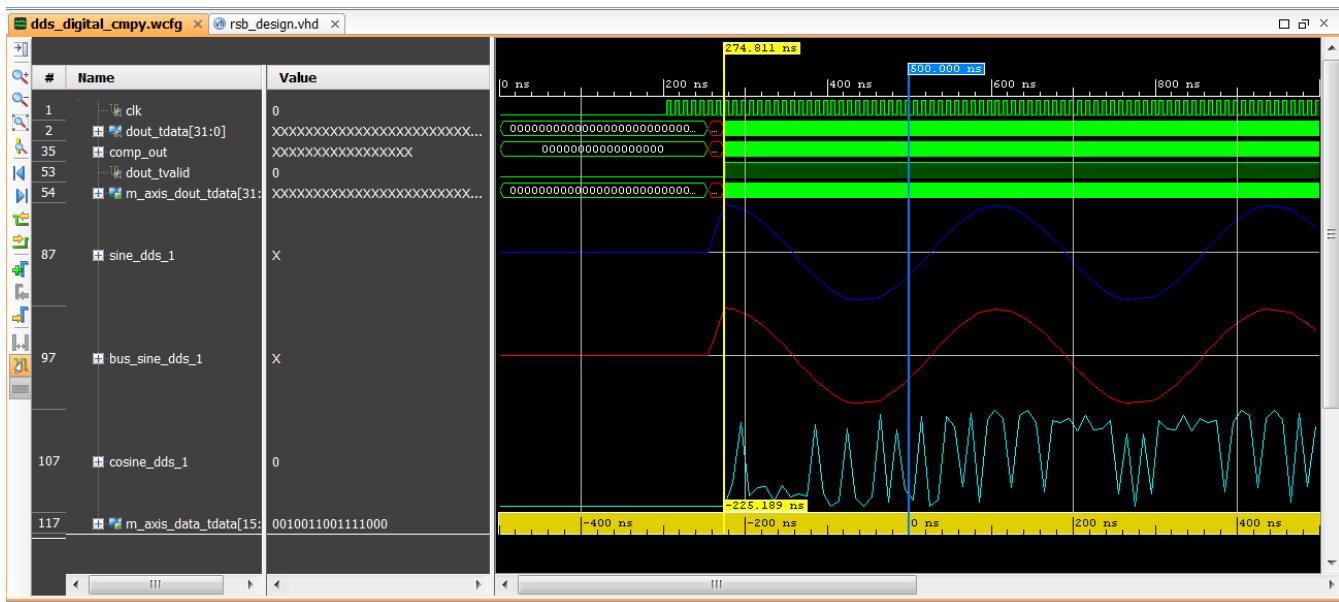


図 3-10: 波形ビューア

波形の保存

新規波形コンフィギュレーションはディスクに自動的には保存されません。[File] → [ave Waveform Configuration As] をクリックして、ファイル名を指定して、WCFG ファイルを作成します。

複数波形コンフィギュレーションの作成と使用

シミュレーション セッションで複数の波形コンフィギュレーション(それぞれ独自の波形ビューアあり)を作成および使用できます。複数の波形ビューアが表示されている場合は、一番最近に作成したビューアまたは最近使用したビューアが「アクティブビューア」になります。アクティブビューアは、現在表示されているビューアだけでなく、ビューアに対する外部コマンド([HDL Objects] → [Add to Wave Window] など)が使用されると表示される波形ビューアです。

別の波形ビューアもそのビューアのタイトルをクリックすると、アクティブビューアにできます。

[Scopes] ビューア

図 3-11 は、[Scope] ビューアを示しています。このビューアでは上部のフィルター ボタンを使用して HDL オブジェクトをタイプ別にフィルターできます。ボタンの上にカーソルを置くと、オブジェクト タイプを示すツール ヒントが表示されます。

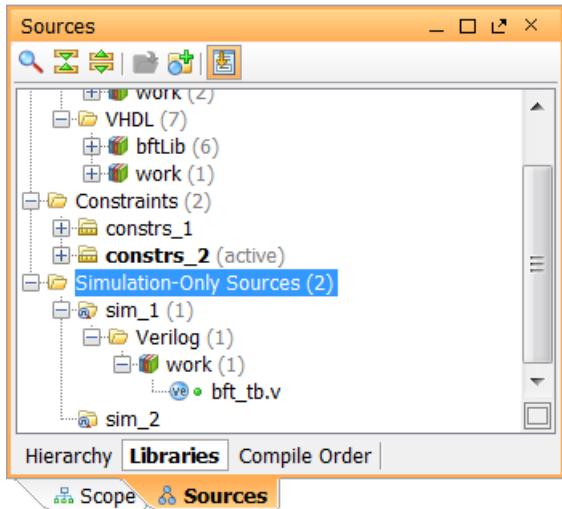


図 3-11 : [Scopes] ビュー

[Sources] ビュー

[Sources] ビューには階層ツリーでシミュレーションソースが [Hierarchy]、[IP Sources]、[Libraries]、[Compile Order] などのタブで表示されます (図 3-12)。

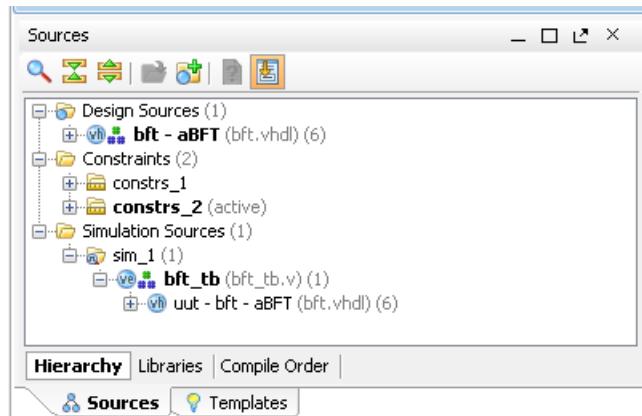


図 3-12 : [Sources] ビュー

[Sources] ビューのボタンの上にカーソルを置くと、その説明がツールヒントとして表示されます。ボタンを使用すると、ファイルの検証、展開/非展開、追加、開く、フィルターおよびスクロールなどことができます。

スコープまたは [Scope] ビューのいずれかで [Show Search] ボタン  をクリックすると、検索フィールドが表示されます。Vivado IDE のメニューの詳細は、『Vivado Design Suite ユーザーガイド : Vivado IDE の使用』(UG893)[参照 3] を参照してください。Vivado シミュレータでの [Scopes] ビューの使用方法については、90 ページの「カーソルの使用」を参照してください。

合成後およびインプリメンテーション後のタイミングシミュレーション

合成後とインプリメンテーション後のタイミングシミュレーションには、オプションで次のいずれかを含めることができます。

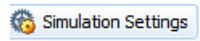
- `simprim` ライブラリ コンポーネントを含むゲートレベルのネットリスト
- SECUREIP
- 標準遅延フォーマット (SDF) ファイル

合成後のタイミングシミュレーションでは、合成後の概算のタイミング数値が使用されます。

インプリメンテーション後のタイミングシミュレーションは、Vivado IDE でインプリメンテーション (配置配線) プロセスが完了してから実行されます。これにより、デザインが実際の回路でどのように動作するか確認することができます。デザインの全体的な機能は最初に定義されていますが、インプリメントされたデザインをシミュレーションする場合は、実際のタイミング情報が使用できます。

Vivado IDE ではネットリストライター (`write_verilog -mode timesim`) および SDF アノテーター (`write_sdf`) が呼び出され、ネットリストおよび SDF が作成されます。

これらのオプションは、[37 ページの「シミュレーション設定」](#) で説明される [Simulation Settings] から変更できます。



重要: 合成後とインプリメンテーション後のタイミングシミュレーションは、Verilog でのみサポートされます。VHDL のタイミングシミュレーションはサポートされません。

詳細は、[66 ページの「合成後およびインプリメンテーション後のシミュレーション」](#) を参照してください。



重要: タイミングシミュレーションを実行する前に [67 ページの「タイミングシミュレーションのインターフェクト遅延」](#) をお読みください。

合成後のシミュレーションの実行

問題なく合成が終了したら、合成後のシミュレーション(論理シミュレーションまたはタイミングシミュレーション)を実行できます。

合成後の論理シミュレーションの実行

合成が問題なく終了すると、[Run Simulation] → [Post-Synthesis Functional Simulation] が使用できるようになります(図 3-13)。

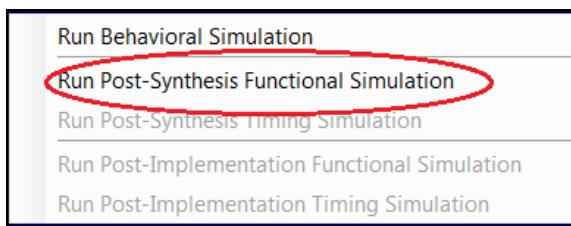


図 3-13: 合成後の論理シミュレーションの実行

合成後には、シミュレーション情報がさらに増えるので、デザイン論理がどれくらい要件を満たしているかが詳細にチェックできます。

合成後の論理シミュレーションを選択すると、論理ネットリストが生成され、その UNISIM ライブラリがシミュレーションに使用されます。

合成後のタイミングシミュレーションの実行

合成が問題なく終了すると、[Run Simulation] → [Post-Synthesis Timing Simulation] が使用できるようになります(図 3-14)。

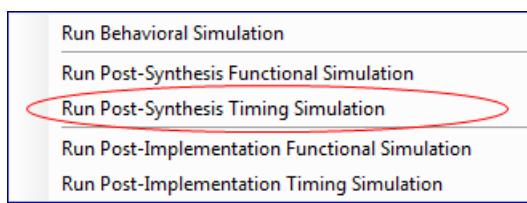


図 3-14: 合成後のタイミングシミュレーションの実行

合成後のタイミングシミュレーションを選択すると、タイミングネットリストと SDF ファイルが生成されます。ネットリストファイルには \$sdf_annotation コマンドが含まれるので、生成された SDF ファイルが自動的に指定されます。

インプリメンテーション後のシミュレーションの実行

問題なくインプリメンテーションが終了したら、インプリメンテーション後の論理シミュレーションまたはタイミングシミュレーションを実行できます。

インプリメンテーション後の論理シミュレーションの実行

インプリメンテーションが問題なく終了すると、[Run Simulation] → [Post-Implementation Functional Simulation] が使用できるようになります(図 3-15)。

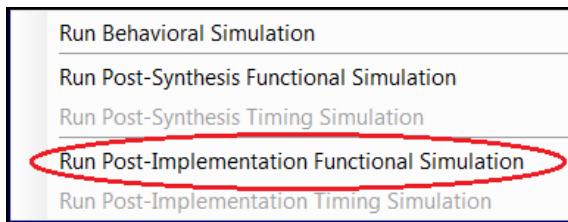


図 3-15: インプリメンテーション後の論理シミュレーションの実行

インプリメンテーション後には、シミュレーション情報がさらに増えるので、デザイン論理がどれくらい要件を満たしているかどうかが詳細にチェックできます。

インプリメンテーション後の論理シミュレーションを選択すると、論理ネットリストが生成され、その UNISIM ライブラリがシミュレーションに使用されます。

インプリメンテーション後のタイミングシミュレーションの実行

インプリメンテーションが問題なく終了すると、[Run Simulation] → [Post-Implementation Timing Simulation] が使用できるようになります(図 3-16)。

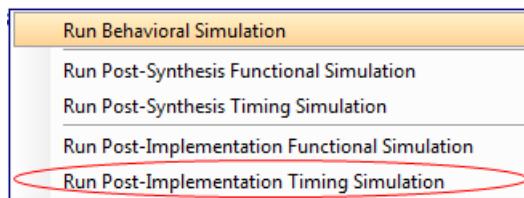


図 3-16: 合成後のタイミングシミュレーションの実行

インプリメンテーション後のタイミングシミュレーションを選択すると、タイミングネットリストと SDF ファイルが生成されます。ネットリストファイルには \$sdf_annotation コマンドが含まれるので、生成された SDF ファイルが自動的に指定されます。

シミュレーションの種類

デザインに対して複数のシミュレーションを実行した場合、Vivado シミュレータでは、現在ハイライトされているビューのシミュレーションタイプを GUI 上部に示します(図 3-17)。

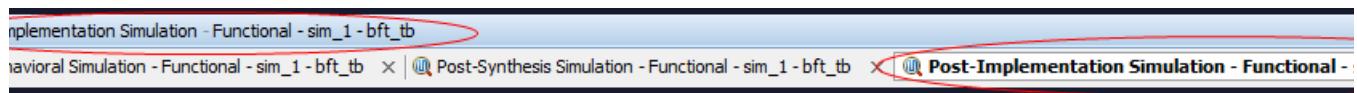


図 3-17: アクティブなシミュレーション タイプ

シミュレーションの一時停止

シミュレーションを実行中に [Break] コマンドを使用すると、シミュレーション セッションを開いたままでシミュレーションを一時停止できます。シミュレーションを一時停止するには、[Simulation] → [Break] をクリックするか、[Break] ボタンをクリックします。

シミュレータは次の実行可能な HDL 行で停止します。シミュレーションの停止した行がテキスト エディターで表示されます。

注記: このビヘイビアは -debug <kind> オプションでコンパイルしたデザインにも適用されます。

シミュレーションは [Run All]、[Run]、または [Step] コマンドを使用するといつでも再開できます。詳細は、53 ページの「シミュレーションのステップ実行」を参照してください。

シミュレーション結果の保存

Vivado シミュレータは、project/simset ディレクトリの波形データベース (WDB) ファイル (<filename>.wdb) にオブジェクト (VHDL 信号や Verilog reg または wire など) のシミュレーション結果を保存します。

オブジェクトを波形ビューに追加してシミュレーションを実行すると、終了したデザインの階層と追加したオブジェクトのトランザクションが自動的に WDB ファイルに保存されます。

信号順、名前形式、基数、色などの波形コンフィギュレーション設定は必要に応じて波形コンフィギュレーション (WCFG) ファイルに保存されます。詳細は、第 5 章「波形を使用した解析」を参照してください。

シミュレーションの終了

シミュレーションを終了するには、[File] → [Exit] をクリックするか、プロジェクト ビューの右上の X マークをクリックします。

Vivado シミュレータのスタンドアロン フローの例

コマンド ラインに Tcl コマンドを入力すると、スタンドアロン フローは次の例のようになります。

```
Vivado -mode Tcl
vivavdo% create_project -force prj1
vivavdo% read_verilog dut.v
vivavdo% synth_design -top dut
vivavdo% place_design
vivavdo% route_design
vivavdo% write_verilog -mode simprim
vivado% postRoute_netlist.v -sdf_file
vivavdo% postRoute.sdf
vivavdo% write_sdf postRoute.sdf
vivavdo% close_project
```

Vivado シミュレーション コマンドを実行すると、フローは次の例のようになります。

```
vivavdo% xvlog postSynth_netlist.v
vivavdo% xvlog tb.v
vivavdo% xelab tb glbl -L simprims_ver -s Test
vivavdo% xsim Test -R
```

統合フロー

詳細は、『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835)[[参照 4](#)] から Tcl コマンドの `launch_xsim` および `launch_modelsim` に関する記述を参照してください。

ソース レベルでのデバッグ

HDL ソース コードをデバッグすると、予測されないビヘイビアを検出できます。デバッグでは、ソース コードの実行を制御することで、問題の原因を見つけます。デバッグで使用可能なストラテジは、次のとおりです。

- [Step through the code line by line] : どの開発段階でも、[Step] コマンドを使用して HDL ソース コードを 1 行ずつデバッグして、デザインが予測どおりかどうかを検証できます。コードの各行で [Step] コマンドを再び使用して、解析を続行します。詳細は、「[シミュレーションのステップ実行](#)」を参照してください。
- [Set breakpoints on the specific lines of HDL code, and run the simulation until a breakpoint is reached] : 大きいデザインの場合、HDL ソース コードの各行が実行されるたびに停止すると時間がかかりすぎるため、HDL ソース コードの任意の位置にブレークポイントを設定し、シミュレーションの実行時に(テストベンチの最初からか、現在の地点から)各ブレークポイントで停止するようにします。停止後にシミュレーションを進めるには、[Step]、[Run All]、または [Run For] コマンドを使用します。詳細は、[54 ページの「ブレークポイントの使用」](#) を参照してください。

シミュレーションのステップ実行

HDL ソース コードを 1 行ずつ実行する [Step] コマンドを使用すると、デザインが予測どおりかどうか検証できます。

黄色の矢印は、現在実行されている行を示します。



ブレークポイントを作成して、ステップ実行中に停止する箇所を増やすこともできます。シミュレータでのストラテジのデバッグについては、54 ページの「ブレークポイントの使用」を参照してください。

1. シミュレーションでステップ実行するには、次の手順に従ってください。
 - 現在の実行時間から [Run] → [Step] をクリックするか、[Step] ボタンをクリックします。
 - 最上位デザイン ユニットに関する HDL が波形ビューに新しいビューとして開きます。
 - スタート地点 (0ns) からシミュレーションを再開します。[Restart] コマンドを使用すると、テストベンチの開始地点に時間をリセットできます。詳細は、第 3 章「Vivado IDE でのシミュレーションの実行およびソースコードのデバッグ」を参照してください。
2. 波形および HDL コードを同時に表示するには、[Window] → [Tile Horizontally] または [Window] → [Tile Vertically] をクリックします。
3. デバッグが終了するまで、[Step] を繰り返します。

行が実行されるにつれ、黄色の矢印がコードの下方に移動していきます。シミュレータが別のファイルの行を実行する場合は、そのファイルが開いて、黄色の矢印でコードがステップ実行されます。大抵のシミュレータで、[Step] コマンドを実行すると複数ファイルが開かれることがよくあります。Tcl コンソールにも step コマンドで HDL コードがどれくらい進んだかが示されます。

ブレークポイントの使用

ブレークポイントは、ユーザーの指定するソース コード内の停止地点で、デザインをデバッグする際に使用できます。



ヒント : ブレークポイントは、[Step] コマンドで 1 行ずつ停止すると時間がかかりすぎてしまうような大規模デザインをデバッグする場合に使用すると、特に便利です。

HDL ファイルの実行可能な行にブレークポイントを設定しておくことで、ブレークポイントに到達するまでコードが実行され続けます。

注記 : ブレークポイントは実行可能なコードにのみ設定できます。実行不可能なコードの行にブレークポイントを設定しても、ブレークポイントは追加されません。

ブレークポイントを設定するには、次の手順に従います。

1. [View] → [Breakpoint] → [Toggle Breakpoint] をクリックします。
または、[Toggle Breakpoint] ボタンをクリックします。
2. HDL ファイルで行番号の右側のコード行をクリックします。

[Breakpoint] ボタン  が行の横に表示されます。

注記 : または、コード行を右クリックし、[Toggle Breakpoint] を選択します。

終了すると、シミュレーションブレークポイント ボタンがコード行の横に表示され、[Breakpoints] ビューに使用可能なブレークポイントがリストされます。

ブレークポイントを使用してデザインをデバッグするには、次の手順に従ってください。

1. HDL ソース ファイルを開きます。
2. HDL ソース ファイルの実行可能な行にブレークポイントを設定します。
3. すべてのブレークポイントが設定されるまで手順 1 および 2 を繰り返します。
4. 次の実行オプションを使用してシミュレーションを実行します。
 - 最初から実行する場合は、[Run] → [Restart] をクリックします。

- 。 [Run] → [Run All] または [Run] → [Run for Specified Time] コマンドを使用します。

ブレークポイントに到達するまでシミュレーションが実行されて、停止します。



HDL ソース ファイルが表示され、
黄色の矢印でブレークポイントの停止地点が示されます。

- 手順 4 を繰り返して、結果に満足するまでシミュレーションをブレークポイントごとに進めていきます。

シミュレーションは、HDL ソース ファイルに設定した各ブレークポイントで停止します。

デザイン デバッグ中には、[Run] → [Step] コマンドを実行して 1 行ごとにコードを検証することで、さらに詳細にデザインを検証することもできます。

HDL ソース コードから 1 つのブレークポイントまたはすべてのブレークポイントを削除できます。

1 つのブレークポイントを削除するには、[Breakpoint] ボタンをクリックします。

すべてのブレークポイントを削除するには、[Run] → [Breakpoint] → [Delete All Breakpoints] をクリックするか、
[Delete All Breakpoints] ボタンをクリックします。

デザインのコンパイルとシミュレーション

概要

ビヘイビアまたはタイミングシミュレーションのいずれかをコマンドラインから実行するには、次を実行する必要があります。

1. デザインファイルの解析
2. シミュレーションスナップショットのエラボレーションと生成
3. デザインのシミュレーション

次のセクションでは、これらの手順について説明します。

次のセクションに示すように、タイミングシミュレーションにはさらに要件があります。

- 第2章の「シミュレーション用タイミングネットリストの生成」
 - 66ページの「合成後およびインプリメンテーション後のシミュレーション」.
-

デザインファイルの解析

`xvhdl` および `xvlog` コマンドは、それぞれ VHDL および Verilog ファイルを解析します。各オプションの詳細は、68ページの表4-2を参照してください。コマンドの詳細は、それぞれのリンクをクリックしてください。

注記: PDF リーダーで、たとえば [Previous View] および [Next View] ボタンをクリックして、前後のページを表示しやすくしておくことをお勧めします。

xvhdl

xvhdl コマンドは VHDL アナライザー(パーサー)です。

xvhdl の構文

```
xvhdl
[-f [-file] <filename>]
[-cryptodumps]
[-h [-help]]
[-initfile <init_filename>]
[-L [-lib] <library_name>[=<library_dir>]]
[-log <filename>]
[-nolog]
[-prj <filename>]
[-relax]
[-v [verbose] [0|1|2]]
[-work <library_name>[=<library_dir>]] ...
```

このコマンドは VHDL ソース ファイルを解析し、解析済みデータをディスクの HDL ライブラリに保存します。

xvhdl の例

```
xvhdl file1.vhd file2.vhd
xvhdl -work worklib file1.vhd file2.vhd
xvhdl -prj files.prj
```

xvlog

xvlog コマンドは Verilog パーサーです。xvlog コマンドは Verilog ソース ファイルを解析し、解析済みデータをディスクの HDL ライブラリに保存します。

xvlog の構文

```
xvlog
[-d [define] <name>[=<val>]]
[-cryptodumps]
[-f [-file] <filename>]
[-h [-help]]
[-i [include] <directory_name>]
[-initfile <init_filename>]
[-L [-lib] <library_name>[=<library_dir>]]
[-log <filename>]
[-nolog]
[-relax]
[-prj <filename>]
[-sourcelibdir <sourcelib_dirname>]
[-sourcelibext <file_extension>]
[-sourcelibfile <filename>]
[-v [verbose] [0|1|2]]
[-version]
```

xvlog の例

```
xvlog file1.v file2.v
```

```
xvlog -work worklib file1.v file2.v
xvlog -prj files.prj
```

xelab を使用したスナップショットのエラボレーションおよび生成

Vivado シミュレータを使用するシミュレーションは、次の 2 段階で実行されます。

- 最初の段階では、シミュレータコンパイラの `xelab` により、HDL モデルがスナップショットにコンパイルされます。スナップショットは、シミュレータが実行できる形式でモデルを表記したものです。
- 次の段階では、`xsim` コマンドにより、スナップショットが読み込まれて実行され、モデルがシミュレーションされます。非プロジェクトモードの場合は、最初の段階を飛ばして次の段階を繰り返すことで、スナップショットを再利用できます。

シミュレータでスナップショットが作成されると、モデルの最上位モジュールの名前に基づいてスナップショット名が割り当てられますが、コンパイラに対するオプションとしてスナップショット名を指定すると、このデフォルトを上書きできます。スナップショット名は、ディレクトリまたは SIMSET 内で重複しないようにします。デフォルトでもカスタムでも、同じスナップショット名を使用すると、前に構築したスナップショットがその名前で上書きされてしまいます。



重要: 同じディレクトリまたは SIMSET 内で同じスナップショット名を使用して、2 つのシミュレーションを実行することはできません。

`xelab` コマンドは、指定した最上位ユニットに対して次を実行します。

- 言語結合規則および `-L <library>` コマンド ラインで指定した HDL ライブラリのどちらかを使用して、子デザインユニットを読み込みます。
- デザインのスタティック エラボレーション (パラメータ、ジェネリックの処理、`generate` 文の実行など) を実行します。
- 実行可能なコードを生成します。
- 生成された実行可能コードをシミュレーション カーネル ライブラリにリンクして、実行可能なシミュレーションスナップショットを作成します。

このあと、出力された実行可能なシミュレーションのスナップショット名を `xsim` コマンドのオプションとして、別のオプションと共に使用します。



ヒント: `xelab` は解析コマンドの `xvlog` および `xvhdl` を暗示的に呼び出すことができます。解析手順は `xelab -prj` オプションで組み込みます。プロジェクトファイルの詳細は、[63 ページの「プロジェクトファイルの構文」](#) を参照してください。

xelab コマンド構文オプション

各オプションの詳細は、[68 ページの表 4-2](#) を参照してください。コマンドの詳細は、それぞれのリンクをクリックしてください。

注記: PDF リーダーで、たとえば [Previous View] および [Next View] ボタンをクリックして、前後のページを表示しやすくしておくことをお勧めします。

```
xelab
[-d [define] <name>[=<val>]]
```

```

[「-debug <kind>」]
[「-f [-file] <filename>」]
[「-generic_top <value>」]
[「-h [-help]」]
[「-i [include] <directory_name>」]
[「-initfile <init_filename>」]
[「-log <filename>」]
[「-L [-lib] <library_name>[=<library_dir>]」]
[「-maxdesigndepth arg」]
[「-mindelay」]
[「-typdelay」]
[「-maxdelay」]
[「-mt arg」]
[「-nolog」]
[「-notimingchecks」]
[「-nosdfinterconnectdelays」]
[「-nospecify」]
[「-override_timeunit」]
[「-override_timeprecision」]
[「-prj <filename>」]
[「-pulse_e arg」]
[「-pulse_r arg」]
[「-pulse_int_e arg」]
[「-pulse_int_r arg」]
[「-pulse_e_style arg」]
[「-r [-run]」]
[「-R [-runall」]
[「-rangecheck」]
[「-relax」]
[「-s [-snapshot] arg」]
[「-sdfnowarn」]
[「-sdfnoerror」]
[「-sdfroot <root_path>」]
[「-sdfmin arg」]
[「-sdftyp arg」]
[「-sdfmax arg」]
[「-sourcelibdir <sourcelib_dirname>」]
[「-sourcelibext <file_extension>」]
[「-sourcelibfile <filename>」]
[「-timescale」]
[「-timeprecision_xvhdl arg」]
[「-transport_int_delays」]
[「-v [verbose] [0|1|2]」]
[「-version」]

```

xelab の例

```

xelab work.top1 work.top2 -s cpusim
xelab lib1.top1 lib2.top2 -s fftsim
xelab work.top1 work.top2 -prj files.prj -s pciesim
xelab lib1.top1 lib2.top2 -prj files.prj -s ethernetsim

```

Verilog 検索順

xelab コマンドは次の検索順を使用して、インスタンシエートされた Verilog デザイン ユニットを検索および結合します。

1. Verilog コードの `uselib` 指示子で指定されたライブラリ。次に例を示します。

```

module
full_adder(c_in, c_out, a, b, sum)
input c_in,a,b;
output c_out,sum;
wire carry1,carry2,sum1;
`uselib lib = adder_lib
half_adder adder1(.a(a),.b(b),.c(carry1),.s(sum1));
half_adder adder1(.a(sum1),.b(c_in),.c(carry2),.s(sum));
c_out = carry1 | carry2;
endmodule

```

2. -lib|-L オプションを使用してコマンド ラインで指定されたライブラリ
3. 親デザイン ユニットのライブラリ
4. work ライブラリ

Verilog インスタンシエーション ユニット

Verilog デザインにコンポーネントがインスタンシエーションされると、xelab コマンドにより、コンポーネント名が Verilog ユニットとして処理され、ユーザーの指定したユニファイド論理ライブラリのリストからユーザーの指定した順序で Verilog モジュールが検索されます。

- 検出されると、xelab はユニットを結合して、検索を停止します。
- xelab で Verilog ユニットが検出できなかった場合は、インスタンシエーションされたモジュールが VHDL エンティティ名として処理され、大文字/小文字の区別のない検索が続行されます。

xelab コマンドは、ユーザーの指定したリストでユニファイド論理ライブラリ順に、インスタンシエートされたモジュール名と同じ名前のエンティティを検索し、最初に一致した名前のモジュールを選択し、検索を停止します。

- 大文字/小文字の区別のある検索がうまくいかなかった場合、xelab は今度は区別のない検索を実行し、ユーザーの指定したリストとユニファイド論理ライブラリの順序で、拡張識別子としてコンストラクトされた VHDL デザイン ユニット名を検出します。
- xelab で 1 つのライブラリに対する独自の結合が見つかった場合は、その名前を選択して、検索を停止します。

注記 : 混合言語デザインの場合、Verilog モジュールでインスタンシエートされる VHDL エンティティへの関連付けに使用されるポート名では、常に大文字/小文字が区別されます。また、VHDL ジェネリックを変更するのに defparam 文は使用できないことにも注意してください。詳細は、[71 ページの「混合言語シミュレーションの使用」](#) を参照してください。

VHDL インスタンシエーション ユニット

VHDL デザインにコンポーネントがインスタンシエートされると、xelab コマンドはそのコンポーネント名を VHDL ユニットとして処理し、論理 work ライブラリでそれを検索します。

- VHDL ユニットが検出されると、xelab コマンドでそれが結合されて、検索が停止されます。
- xelab で VHDL ユニットが検出されない場合、大文字/小文字の維持されたコンポーネント名が Verilog モジュール名として処理され、ユーザーの指定したリストおよびユニファイド論理ライブラリの順で大文字/小文字の区別のある検索が続行されます。最初に一致する名前が見つかると、検索は停止されます。
- 大文字/小文字の区別のある検索がうまくいかなかった場合、xelab は今度は区別のない検索をユーザーの指定したリストとユニファイド論理ライブラリの順序で実行します。1 つのライブラリに対して独自の結合が見つかれば、検索は停止されます。

Verilog の `uselib 指示子

ライブラリ検索順を設定する Verilog の `uselib 指示子がサポートされています。

`uselib の構文

```

<uselib compiler directive> ::= `uselib [<Verilog-XL uselib directives>|<lib
directive>]

<Verilog-XL uselib directives> ::= dir = <library_directory> | file = <library_file>
| libext = <file_extension>
<lib directive> ::= <library reference> {<library reference>}
<library reference> ::= lib = <logical library name>

```

`uselib lib 指示子

`uselib lib 指示子は、Verilog-XL の `uselib 指示子とは一緒に使用できません。たとえば、次のコードは不正になります。

```
`uselib dir=./ file=f.v lib=newlib
```

1 つの `uselib ライブラリで複数のライブラリが指定できます。

ライブラリが指定される順が検索順になります。次に例を示します。

```
`uselib lib=mylib lib=yourlib
```

これは、インスタンシエートされたモジュールの検索が mylib で最初に実行され、次に yourlib で実行されるようにしています。

`uselib dir、`uselib file、`uselib libext などの指示子と同様、`uselib lib 指示子も HDL ファイル全体で指定した分析および解析があると、処理され続けます。別の `uselib 指示子があるまでは、HDL ソースの `uselib(すべての Verilog XL `uselib を含む) 指示子が使用されます。引数なしで `uselib を使用すると、現在アクティブな `uselib <lib|file|dir|libext> の状態が削除されます。

次のモジュール検索メカニズムは、Verific の Verilog エラボレーションアルゴリズムでインスタンシエート済みモジュールまたは UDP の決定に使用されます。

- まず、インスタンシエート済みモジュールを現在アクティブな `uselib lib の論理ライブラリの順序リストで検索します。
- 見つからない場合は、xelab コマンド ラインで検索ライブラリとして提供されているライブラリの順序リストでインスタンシエート済みモジュールを検索します。
- 見つからない場合は、親モジュールのライブラリでインスタンシエート済みモジュールを検索します。たとえば、work ライブラリのモジュール A が mylib ライブラリのモジュール B をインスタンシエートし、モジュール B がモジュール C をインスタンシエートする場合、mylib ライブラリ (C の親モジュール B のライブラリ) でモジュール C を検索します。
- 見つからない場合は、次のいずれかの work ライブラリでインスタンシエートされたみモジュールを検索します。
 - HDL ソースがコンパイルされるライブラリ
 - work ライブラリとして明示的に設定されるライブラリ
 - work という名前のデフォルトの作業ライブラリ

`uselib の例

adder_lib という名前の論理ライブラリにコンパイルされる half_adder.v ファイル	work という名前の論理ライブラリにコンパイルされる full_adder.v ファイル
<pre>module half_adder(a,b,c,s); input a,b; output c,s; s = a ^ b; c = a & b; endmodule</pre>	<pre>module full_adder(c_in, c_out, a, b, sum) input c_in,a,b; output c_out,sum; wire carry1,carry2,sum1; `uselib lib = adder_lib half_adder adder1(.a(a),.b(b),. c(carry1),.s(sum1)); half_adder adder1(.a(sum1),.b(c_in),.c (carry2),.s(sum)); c_out = carry1 carry2; endmodule</pre>

xsim によるデザイン スナップショットのシミュレーション

xsim コマンドは、シミュレーション スナップショットを読みこんで、バッチ モードのシミュレーションを実行するか、GUI または Tcl ベースの対話型シミュレーション環境を提供します。

xsim の構文

コマンド構文は、次のようになります。

`xsim <options> <snapshot>`

説明 :

- xsim はコマンドです。
- <options> はオプションです (表 4-1)。
- <snapshot> はシミュレーション スナップショットです。

xsim のオプション

表 4-1 : xsim コマンドのオプション

xsim オプション	説明
<code>-f [-file] <filename></code>	ファイルからコマンド ライン オプションを読み込みます。
<code>-g [-gui]</code>	対話型 GUI で実行します。
<code>-h [-help]</code>	ヘルプ メッセージを画面に表示します。
<code>-log <filename></code>	ログ ファイルの名前を指定します。
<code>-maxdeltaid arg (= -1)</code>	最大デルタ値を指定します。同時に最大シミュレーション ループを上回る場合はエラーをレポートします。
<code>-nolog</code>	ログ ファイルが生成されないようにします。

表 4-1: xsim コマンドのオプション (続き)

xsim オプション	説明
-onfinish <quit stop>	シミュレーション終了時のビヘイビアを指定します。
-onerror <quit stop>	シミュレーション ランタイム エラーの発生した場合のビヘイビアを指定します。
-t [-tclbatch] <filename>	バッチ モード実行用の Tcl ファイルを指定します。
-R [-runall]	シミュレーションを最後まで実行します(例: do 'run all;quit')。
-testplusarg <arg>	plusargs が \$test\$plusargs および \$value\$plusargs システム関数で使用されるように指定します。
-vcdfilename <vcdf_filename>	VCD 出力ファイルを指定します。
-vcdunit <vcdunit>	VCD 出力単位を指定します。デフォルトはエンジンの精度単位になります。
-view <wavefile.wcfg>	波形コンフィギュレーション ファイルを開きます。このオプションは -gui オプションと一緒に使用します。
-wdb <filename.wdb>	波形データベース出力ファイルを指定します。
-tp	実行されているプロセスの階層名が画面に表示されるようになります。
-tl	実行されている文のファイル名および行番号が画面に表示されるようになります。

プロジェクト ファイルの構文

プロジェクト ファイルを使用してデザイン ファイルを解析するには、<proj_name>.prj というファイル名を作成し、そのプロジェクト ファイル内で次の構文を使用します。

```
verilog <work_library> <file_names>... [-d <macro>]... [-i <include_path>]...
vhdl <work_library> <file_name>
```

説明 :

<work_library>: 指定した行の HDL ファイルがコンパイルされるライブラリ

<file_names>: Verilog ソース ファイル。各行に複数の Verilog ファイルを指定できます。

<file_name>: VHDL ソース ファイル。各行に複数の VHDL ファイルを指定できます。

- Verilog の場合、[-d <macro>] を使用すると、オプションで 1 つまたは複数のマクロを定義できます。

- VHDL の場合、[-i <include_path>] を使用すると、オプションで 1 つまたは複数の <include_path> ディレクトリを定義できます。

定義済み XILINX_SIMULATOR マクロ (Verilog シミュレーション用)

XILINX_SIMULATOR は Verilog の定義済みマクロです。このマクロの値は 1 です。定義済みマクロでは、ツール専用の関数が実行されるか、デザイン フローで使用するツールが認識されます。次は使用例です。

```
`ifdef VCS
    // VCS specific code
`endif
```

```
 `ifdef INCA
    // NCSIM specific code
 `endif
 `ifdef MODEL_TECH
    // MODELSIM specific code
 `endif
 `ifdef XILINX_ISIM
    // ISE Simulator (ISim) specific code
 `endif
 `ifdef XILINX_SIMULATOR
    // Vivado Simulator (XSim) specific code
 `endif
```

非プロジェクト モードでのデザインシミュレーション

シミュレーションコマンドを Tcl ファイルに入力し、その Tcl ファイルを `-tclbatch` オプションを付けて呼び出すことができます。

`-tclbatch` オプションは、ファイル内にコマンドを含め、シミュレーション開始時にそれらのコマンドを実行するために使用できます。たとえば、次を含む `run.tcl` というファイルがあるとします。

```
run 20ns
current_time
quit
```

次に、シミュレーションを次のように実行します。

```
xsim <snapshot> -tclbatch run.tcl
```

シミュレーションコマンドを表す変数を設定しておくと、よく使用されるシミュレーションコマンドを素早く実行できます。

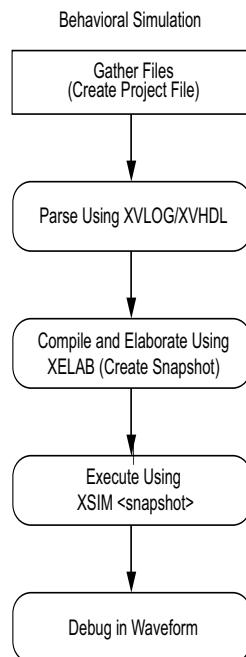
Tcl の使用方法については、次を参照してください。

- <http://www.tcl.tk/>
- 『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835) [参照 4]
- 『Vivado Design Suite ユーザー ガイド :Tcl スクリプト機能の使用』(UG894) [参照 7]

`xelab` を使用してデザインを解析してシミュレーションの実行可能なスナップショットを作成したら、論理シミュレーションまたはタイミングシミュレーションが実行できます。

ビヘイビアシミュレーションの実行

図 4-1 は、ビヘイビアシミュレーションプロセスを示しています。



X1284

図 4-1:ビヘイビアシミュレーションプロセス

Tcl コンソールでビヘイビアシミュレーションを実行するには、次を入力します。

```
launch_xsims -mode behavioral
```

合成後およびインプリメンテーション後のシミュレーション

合成後およびインプリメンテーション後のシミュレーションでは、論理シミュレーションまたは Verilog タイミングシミュレーションが実行できます。

図 4-2 は、合成後およびインプリメンテーション後のシミュレーションプロセスを示しています。

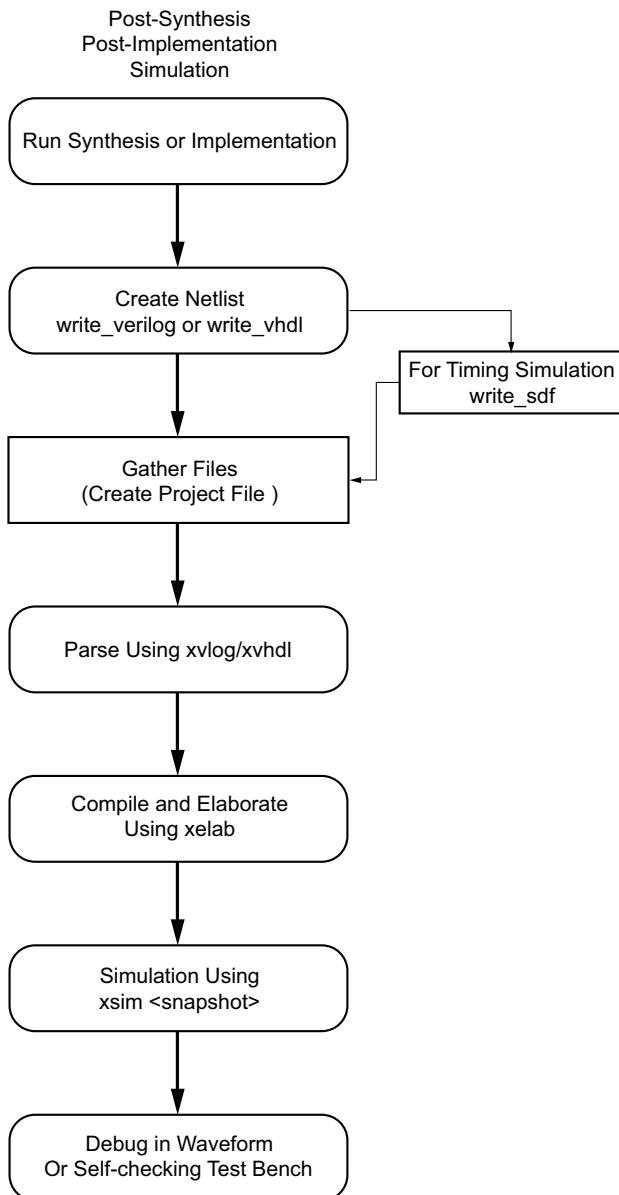


図 4-2 : 合成後およびインプリメンテーション後のシミュレーション

次は、コマンド ラインから合成後の論理シミュレーションを実行する例です。

```
synth_design -top top -part xc7k70tfg676-2
open_run synth_1 -name netlist_1
write_verilog -mode funcsim test_synth.v
xvlog -work work test_synth.v
xvlog -work work testbench.v
xelab -L unisims_ver testbench glbl -s test
xsim test -gui
```

タイミングシミュレーションのアノテーションコマンド

合成後およびインプリメンテーション後のタイミングシミュレーションを実行する場合、`write_verilog` の後に `write_sdf` コマンドを実行する必要があります。また、エラボレーションおよびシミュレーションには適切なアノテート コマンドが必要です。

タイミングシミュレーションのインターフェクト遅延

Vivado シミュレータでは、インターフェクト遅延が使用されるので、タイミングシミュレーションが正しく実行されるには、次のようなコマンドが追加で必要になります。

```
-transport_int_delays -pulse_r 0 -pulse_int_r 0
```

ライブラリマップファイル (xsim.ini)

HDL コンパイラプログラムの `xvhdl`、`xvlog`、`xelab` では、`xsim.ini` コンフィギュレーションファイルを使用して、VHDL および Verilog 論理ライブラリの定義および物理的な位置が検出されます。

コンパイラはこれらの位置から次の順序で `xsim.ini` を読み出そうとします。

1. `$XILINX_PLANAHEAD/data/xsim`
2. ユーザー ファイルは `-initfile` オプションで指定します。 `-initfile` が指定されない場合、プログラムは現在の作業ディレクトリで `xsim.ini` を検索します。

`xsim.ini` ファイルの構文は、次のとおりです。

```
<logical_library1> = <physical_dir_path1>
<logical_library2> = <physical_dir_path2>
```

次は `xsim.ini` ファイルの例です。

```
VHDL
std=C:/libs/vhdl/hdl/
stdieee=C:/libs/vhdl/hdl/ieee
work=C:/workVerilog
unisims_ver=$XILINX_PLANAHEAD/data/verilog/hdl/nt/unisims_ver
xilinxcorelib_ver=C:/libs/verilog/hdl/nt/xilinxcorelib_ver
mylib=../mylib
work=C:/work
```

`xsim.ini` ファイルの機能および制限は、次のとおりです。

- `xsim.ini` ファイルでは、各行のライブラリパスは 1 つだけです。

- このファイルは、物理パスに対応するディレクトリがない場合、コンパイラが最初に書き込もうとする際に、**xvhdl** または **xvlog** で作成されます。
- 物理パスは環境変数の観点から説明できます。環境変数は \$ 文字で開始する必要があります。
- 論理ライブラリのデフォルトの物理ディレクトリは、**xsim/<logical_library_name>** です。次は論理ライブラリ名の例です。

```
$XILINX_PLANAHEAD/data/verilog/src/glbl.v
```

ファイルコメントは -- で開始する必要があります。

xelab、xvhdl、xvlog コマンド オプション

表 4-2 は、xelab、xvhdl、xvlog コマンドのオプションをリストしています。

表 4-2 : xelab、xvhdl、xvlog コマンド オプション

コマンド オプション	説明
-d [define] <name>[=<val>]	Verilog マクロを定義します。各 Verilog マクロに対して -d --define を使用します。マクロのフォーマットは <name>[=<val>] で、 <name> はマクロの名前、 <value> はマクロのオプションの値を示します。
-debug <kind>	指定したデバッグ機能をオンにしてコンパイルします。 <kind> オプションには、次のいずれかを指定できます。 <ul style="list-style-type: none"> typical: line および wave などの最もよく使用される機能 line: HDL のブレークポイント wave: 波形生成、条件付き実行、強制値 xlibs: ザイリンクスプリコンパイル ライブラリの可視性。このコマンドは、コマンドラインでのみ使用できます。 off: すべてのデバッグ機能をオフにします(デフォルト)。 all: すべてのデバッグ オプションを使用します。
-encryptdumps	コンパイルされるデザイン ユニットの解析済みダンプを暗号化します。
-f [-file] <filename>	指定したファイルから追加オプションを読み出します。
-generic_top <value>	最上位デザイン ユニットのジェネリックまたはパラメーターを指定した値で上書きします。例: -generic_top "P1=10"
-h [-help]	このヘルプ メッセージを表示します。
-i [include] <directory_name>	Verilog の `include を使用して、含まれるファイルを検索するディレクトリを指定します。指定検索ディレクトリごとに -i --include を使用します。
-initfile <init_filename>	デフォルトの xsim.ini ファイルで提供される設定へ追加または上書きするユーザー定義のシミュレータ初期化ファイルを指定します。
-incremental	最後にコンパイルしてから変更されたファイルのみをコンパイルします。
-L [-lib] <library_name>[=<library_dir>]	インスタンシエートされた VHDL 以外のデザイン ユニット (Verilog デザイン ユニットなど) の検索ライブラリを指定します。検索ライブラリごとに -L --lib を使用します。引数のフォーマットは <name>[=<dir>] で、 <name> はライブラリの論理名、 <library_dir> はライブラリのオプションの物理ディレクトリを表します。
-log <filename>	ログ ファイルの名前を指定します。デフォルトは xvlog xvhdl xelab.log です。
-maxdelay	Verilog デザイン ユニットを最小遅延でコンパイルします。

表 4-2 : xelab、xvhd、xvlog コマンド オプション (続き)

コマンド オプション	説明
-mindelay	Verilog デザイン ユニットを最大遅延でコンパイルします。
-mt arg	並列実行可能なサブコンパイル ジョブの数を指定します。使用できる値は、auto、off、または 1 より大きい整数です。 auto が指定されると、xelab でホスト マシンの CPU 数に基づいて並列ジョブの数が選択されます(デフォルト = auto)。 -mt オプションをさらに詳細に制御するには、Tcl プロパティを次のように設定します。 <code>set_property XELAB.MT_LEVEL off N [get_filesets_sim_1]</code>
-maxdesigndepth arg	エラボレーターで許容される最大のデザイン階層の深さ (デフォルト = 5000) を上書きします。
-nolog	ログ ファイルが生成されないようにします。
-notimingchecks	Verilog 指定ブロックでタイミング チェック コンストラクトを無視します。
-nosdfinterconnectdelays	SDF の SDF ポートおよびインターフェクト遅延コンストラクトを無視します。
-nospecify	Verilog パス遅延とタイミング チェックを無視します。
-override_timeunit	-timescale オプションで指定した値で、すべての Verilog モジュールの時間単位を上書きします。
-override_timeprecision	-timescale オプションで指定した時間精度で、すべての Verilog モジュールの時間精度を上書きします。
-pulse_e arg	パス遅延のパーセントでパス パルス エラー制限を指定します。使用できる値は 0 ~ 100 です(デフォルトは 100)。
-pulse_r arg	パス遅延のパーセントでパス パルス拒否制限を指定します。使用できる値は 0 ~ 100 です(デフォルトは 100)。
-pulse_int_e arg	遅延のパーセントでインターフェクト パルス拒否制限を指定します。使用できる値は 0 ~ 100 です(デフォルトは 100)。
-pulse_int_r arg	遅延のパーセントでインターフェクト パルス拒否制限を指定します。使用できる値は 0 ~ 100 です(デフォルトは 100)。
-pulse_e_style arg	パルスがモジュール パス遅延よりも短いエラーを処理すべきかどうかを指定します。選択肢は次のとおりです。 ondetect : 違反が検出されるとエラーをレポート onevent : モジュールのパス遅延後にエラーをレポート (デフォルト : onevent)
-prj <filename>	vhdl verilog <work lib> <HDL file name> の入力が 1 つまたは複数含まれる XSim プロジェクト ファイルを指定します。
-rangecheck	ランタイム値の範囲のチェックをイネーブルにします(VHDL)。
-r [-run]	コマンド ラインの対話型モードで生成した実行可能スナップショット ファイルを実行します。
-R [-runall]	シミュレーションの最後まで生成した実行可能スナップショット ファイルを実行します。
-relax	厳密な言語規則を緩めます。
-s [-snapshot] arg	出力されるシミュレーション スナップショットの名前を指定します。デフォルトは <worklib>.<unit> です(例 : work.top)。追加のユニット名は # で連結します(例 : work.t1#work.t2)。

表 4-2 : xelab、xvhd、xvlog コマンド オプション (続き)

コマンド オプション	説明
-sdfnowarn	SDF 警告を出力しません。
-sdfnoerror	SDF ファイルで見つかったエラーを警告として処理します。
-sdfmin arg	<root=file> SDF は <file> を最小遅延を含めて <root> でアノテートします。
-sdftyp arg	<root=file> SDF は <file> を通常遅延を含めて <root> でアノテートします。
-sdfmax arg	<root=file> SDF は <file> を最大遅延を含めて <root> でアノテートします。
-sdfroot <root_path>	SDF アノテーションが適用されるデフォルト デザイン階層
-sourcelibdir <sourcelib dirname>	コンパイルされていないモジュールの Verilog ソース ファイルのディレクトリ。ソース ディレクトリごとに -sourcelibdir <sourcelib dirname> を使用します。
-sourcelibext <file_extension>	コンパイルされていないモジュールの Verilog ソース ファイルのファイル拡張子。ソース ファイル拡張子に -sourcelibext <file extension> を使用します。
-sourcelibfile <filename>	コンパイルされていないモジュールを使用した Verilog ソース ファイルのファイル名。 -sourcelibfile <filename> を使用します。
-timescale	Verilog モジュールのデフォルトの時間単位を指定します。デフォルトは 1ns/1ps です。
-timeprecision_xvhdl arg	VHDL デザインの時間精度を指定します。デフォルトは 1ps です。
-transport_int_delays	インターフェクト遅延に転送モデルを使用します。
-typdelay	Verilog デザイン ユニットを通常遅延(デフォルト)でコンパイルします。
-v [verbose] [0 1 2]	表示メッセージの詳細レベルを指定します。デフォルトは 0 です。
-version	コンパイラ バージョンを画面に表示します。
-work <library_name>[=<library_dir>]	work ライブラリを指定します。この引数のフォーマットは <name>[=<dir>] です。 <ul style="list-style-type: none">• <name>: ライブラリの論理名• <library_dir>: ライブラリのオプションの物理ディレクトリ

混合言語シミュレーションの使用

Vivado シミュレータでは、混合言語プロジェクトファイルおよび混合言語シミュレーションがサポートされるので、Verilog モジュールを VHDL デザインに含めたり、VHDL モジュールを Verilog デザインに含めたりできます。

シミュレーションでの混合言語の制限

- VHDL および Verilog の混合は、モジュールインスタンスまたはコンポーネントにのみ制限されます。
- VHDL デザインは Verilog モジュールをインスタンシエートでき、Verilog デザインは VHDL コンポーネントをインスタンシエートできます。それ以外の VHDL および Verilog の混合はサポートされません。
- Verilog 階層参照では VHDL ユニットは参照できず、VHDL 拡張/選択名では Verilog ユニットは参照できません。
- VHDL の型、ジェネリック、ポートの小さいサブセットのみが Verilog モジュールへのバウンダリ上で使用できます。同様に、Verilog の型、パラメーター、ポートの小さいサブセットのみが VHDL デザインユニットへのバウンダリ上で使用できます。
- Verilog モジュールと VHDL デザインユニットの結合には、コンポーネントインスタンシエーションベースのデフォルトの結合が使用されます。特に、VHDL デザインユニット内にインスタンシエートされる Verilog モジュールでは、コンフィギュレーション仕様、直接インスタンシエーション、およびコンポーネントコンフィギュレーションがサポートされません。

混合言語シミュレーションでの重要な手順

1. 混合言語プロジェクトのデザインライブラリ内で VHDL エンティティまたは Verilog モジュールの検索順を指定できます(オプション)。
2. `xelab -L` を使用すると、混合言語プロジェクトのデザインライブラリ内で VHDL エンティティまたは Verilog モジュールの結合順を指定できます。

注記: Verilog モジュールと別の Verilog モジュールの結合にも、`-L` で指定したライブラリ検索順が使用されます。

混合言語の結合と検索

VHDL コンポーネントまたは Verilog モジュールをインスタンシエートするには、`xelab` コマンドを使用します。

- まず、インスタンシエートするデザインユニットとして同じ言語のユニットを検索します。
- 同じ言語のユニットが見つからない場合は、`xelab` を使用し、`-lib` オプションで指定したライブラリ内で両方の言語のデザインユニットを検索します。

この検索順は、`xelab` コマンド ラインでライブラリが表示される順序と同じです。詳細は、[22 ページの「方法 1: ライブラリまたはファイルコンパイル順の使用\(推奨\)](#)」を参照してください。

注記: Vivado IDE を使用する場合、ライブラリ検索順は自動的に指定されます。ユーザーは設定する必要がないので、設定できません。

混合言語コンポーネントのインスタンシエーション

混合言語デザインの場合、次のセクションで説明するように、Verilog モジュールを VHDL デザインユニットに、または VHDL モジュールを Verilog デザインユニットにインスタンシエートできます。

混合言語のバウンダリとマップ規則

VHDL および Verilog デザイン ユニット/モジュール間のバウンダリには、次の制限が適用されます。

- VHDL および Verilog 間のバウンダリはデザイン ユニット レベルになります。
- VHDL デザインには 1 つ以上の Verilog モジュールをインスタンシエートできます。
- VHDL デザイン内への Verilog UDP のインスタンシエーションはサポートされません。
- Verilog デザインには、VHDL エンティティに対する VHDL コンポーネントのみをインスタンシエートできます。
- Verilog デザイン内への VHDL コンフィギュレーションのインスタンシエーションはサポートされません。

VHDL デザイン ユニットへの Verilog モジュールのインスタンシエーション

1. VHDL コンポーネントをインスタンシエートする Verilog モジュールと同じ名前(大文字/小文字の区別あり)を付けて宣言します。次に例を示します。

```
COMPONENT MY_VHDL_UNIT PORT (
  Q : out STD_ULOGIC;
  D : in STD_ULOGIC;
  C : in STD_ULOGIC );
END COMPONENT;
```

2. 名づけた関連付けを使用し、Verilog モジュールをインスタンシエートします。次に例を示します。

```
UUT :MY_VHDL_UNIT PORT MAP(
  Q => O,
  D => I,
  C => CLK);
```

ポート タイプが一致しているかどうかは、「ポート マップおよびサポートされるポート タイプ」を参照してください。

ポート マップおよびサポートされるポート タイプ

表 4-3 は、サポートされるポート タイプをリストしています。

表 4-3: サポートされるポート タイプ

VHDL ¹	Verilog ²
IN	INPUT
OUT	OUTPUT
INOUT	INOUT

1. VHDL のバッファおよびリンクエージ ポートはサポートされません。
2. Verilog の双方向パス オプションへの接続はサポートされません。名前の付いていない Verilog ポートは混合デザインのバウンダリでは使用できません。

表 4-4 は、混合言語デザインのバウンダリでサポートされるポートの VHDL および Verilog データ型を示しています。

表 4-4: サポートされる VHDL および Verilog データ型

VHDL ポート	Verilog ポート
bit	net
std_logic	net
std_logic	net

表 4-4: サポートされる VHDL および Verilog データ型 (続き)

VHDL ポート	Verilog ポート
bit_vector	vector net
std_ulogic_vector	vector net
std_logic_vector	vector net

注記: Verilog 出力ポートの `reg` 型は、混合言語バウンダリでサポートされます。バウンダリ上では、出力ポート `reg` が出力ネット(ワイヤ)ポートのように処理されます。これ以外の型が混合言語バウンダリで使用されると、エラーとなります。

ジェネリック(パラメーター)のマップ

Vivado シミュレータでは、次の VHDL ジェネリック タイプ(および Verilog の対応するもの)がサポートされます。

- `integer`
- `real`
- `string`
- `boolean`

注記: これ以外のジェネリック型が混合言語バウンダリで使用されると、エラーとなります。

VHDL および Verilog 値のマップ

表 4-5 は、`std_logic` および `bit` への Verilog ステートのマップをリストしています。

表 4-5: `std_logic` および `bit` へマップされる Verilog ステート

Verilog	<code>std_logic</code>	<code>bit</code>
Z	Z	0
0	0	0
1	1	1
X	X	0

注記: Verilog の駆動電流は無視されます。VHDL には、これに対応するマップがありません。

表 4-6 は、Verilog ステートにマップされる VHDL 型 `bit` をリストしています。

表 4-6: Verilog ステートにマップされる VHDL の `bit`

<code>bit</code>	Verilog
0	0
1	1

表 4-7 は、Verilog ステートにマップされる VHDL 型 `std_logic` をリストしています。

表 4-7: Verilog ステートにマップされる VHDL の `std_logic`

<code>std_logic</code>	Verilog
U	X
X	X

表 4-7: Verilog ステートにマップされる VHDL の std_logic

std_logic	Verilog
0	0
1	1
Z	Z
W	X
L	0
H	1
-	X

Verilog では大文字/小文字が区別されるので、コンポーネント宣言で使用する関連付けおよびローカルポート名は Verilog ポート名の大文字/小文字と一致している必要があります。

Verilog デザインユニットへの VHDL モジュールのインスタンシエーション

Verilog デザインユニットに VHDL モジュールをインスタンシエートするには、VHDL エンティティを Verilog モジュールのようにインスタンシエートする必要があります。次に例を示します。

```
module testbench ;
  wire in, clk;
  wire out;
  FD FD1(
    .Q(Q_OUT),
    .C(CLK),
    .D(A);
  );

```

波形を使用した解析

概要

Vivado™ Integrated Design Environment (IDE) シミュレータ GUI を開くと、波形ウィンドウを使用してデザインを解析し、コードをデバッグできます。シミュレータは、[Objects] ビューおよび[Scopes] ビューなどの GUI のほかのエリアでデザインデータを生成します。

通常、シミュレーションはシミュレーションをする HDL オブジェクトを定義するテストベンチで設定します。テストベンチの詳細は、『Writing Efficient Testbenches』(XAPP199)[\[参照 5\]](#) を参照してください。

Vivado IDE からのシミュレーション

Vivado シミュレータを起動すると、最上位 HDL オブジェクトを含む波形コンフィギュレーションが表示されます。Vivado シミュレータは、[Objects] ビューおよび[Scopes] ビューなどの GUI のほかのエリアでデザインデータを生成します。この後、別の HDL オブジェクトを追加したり、[Run] コマンドを使用してシミュレーションを実行したりできます。詳細は、[76 ページの「波形コンフィギュレーションと波形ビュー」](#) を参照してください。

非プロジェクト モードでのシミュレーション

非プロジェクト モードでのシミュレーション波形と GUI コマンドについては、次を参照してください。

- [92 ページの「非プロジェクト モードの使用」](#)
- 『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835) [\[参照 4\]](#)

波形コンフィギュレーションと波形ビュー

波形コンフィギュレーションおよび WCFG ファイルの両方がカスタマイズされた波形リストを参照しますが、この 2 つの概念は異なります。

- 波形コンフィギュレーションは、作業可能なメモリに読み込まれたオブジェクトです。
- WCFG ファイルはディスクの波形コンフィギュレーションが保存されたものです。

波形コンフィギュレーションには名前が付くか、Untitled# が付きます。名前は波形コンフィギュレーションビューに表示されます。

波形コンフィギュレーションの新規作成

波形を表示するために新しい波形コンフィギュレーションを作成します。

1. [File] → [New Waveform Configuration] をクリックします。

新しい波形ビューが開いて、新規のタイトルのない波形コンフィギュレーションが表示されます。

2. [78 ページの「波形コンフィギュレーションの HDL オブジェクト」](#) に示す方法で HDL オブジェクトを波形コンフィギュレーションに追加します。

注記 : WCFG に、HDL デザイン階層が開いたときにシミュレーションに存在しない HDL オブジェクトへの参照が含まれると、Vivado シミュレータではこれらの HDL オブジェクトが無視され、読み込まれた波形コンフィギュレーションから削除されます。

波形波形の新規作成の詳細は、[第 3 章「Vivado IDE でのシミュレーションの実行およびソース コードのデバッグ」](#) を参照してください。

WCFG ファイルを開く

WCFG ファイルを開いて、スタティックシミュレーションと一緒に使用します。

1. [File] → [Open Waveform Configuration] をクリックします。

[Specify Simulation Results] ダイアログ ボックスが表示されます。

2. WCFG ファイルを選択します。

注記 : WCFG ファイルにスタティックシミュレーションの HDL デザイン階層にはない HDL オブジェクトへの参照が含まれる場合、Vivado シミュレータではこれらの HDL オブジェクトが無視され、読み込まれた波形コンフィギュレーションから削除されます。

波形ビューが開き、WCFG ファイルのリストされた波形オブジェクト用にシミュレータで検出された波形データが表示されます。

波形コンフィギュレーションの保存

波形コンフィギュレーションを WCFG ファイルに保存するには、[File] → [Save Waveform Configuration As] をクリックし、波形コンフィギュレーションの名前を入力します。

前のシミュレーション設定からのシミュレーションデータの表示(スタティックシミュレーション)

シミュレーションを実行して HDL オブジェクトを波形ビューで表示する場合、シミュレーションを実行することで、表示されている HDL オブジェクトの波形アクティビティを含む波形データベース (WDB) ファイルが作成されます。WDB ファイルには、シミュレーションされたデザインの HDL スコープとオブジェクトすべてに関する情報も含まれます。

「スタティックシミュレーション」は Vivado シミュレータのモードで、シミュレーションを実行して得たデータの代わりに、WDB ファイルからのデータがビューに表示されます。

このモードの場合、下位に制御するライブシミュレーション モデルがないので、run コマンドのようなシミュレーションを制御または監視するコマンドは使用できません。



重要: WDB ファイルには後方置換性はなく、OS 互換性もありません。WDB ファイルは、作成したのと同じバージョンで、同じタイプの OS で開く必要があります。WCFG ファイルには後方互換性も OS 互換性もあります。

ただし、波形および HDL デザイン階層を表示することはできます。シミュレータで波形コンフィギュレーションがデフォルトで作成されることはないので、新しい波形コンフィギュレーションを作成するか、WCFG ファイルを開く必要があります。

波形データベース ファイルの表示

波形データベース (WDB) ファイルを開いて、シミュレーションとして表示するには、次の手順に従ってください。

1. Vivado IDE を開きます。
2. シミュレーションを実行した既存の Vivado プロジェクトを開きます。[Open Static Simulation] の機能を使用するには、この手順が必ず必要です。
3. Flow Navigator の [Simulation] セクションで [Open Static Simulation] をクリックします。

78 ページの図 5-1 に示す [Specify Simulation Results] ダイアログ ボックスが表示されます。

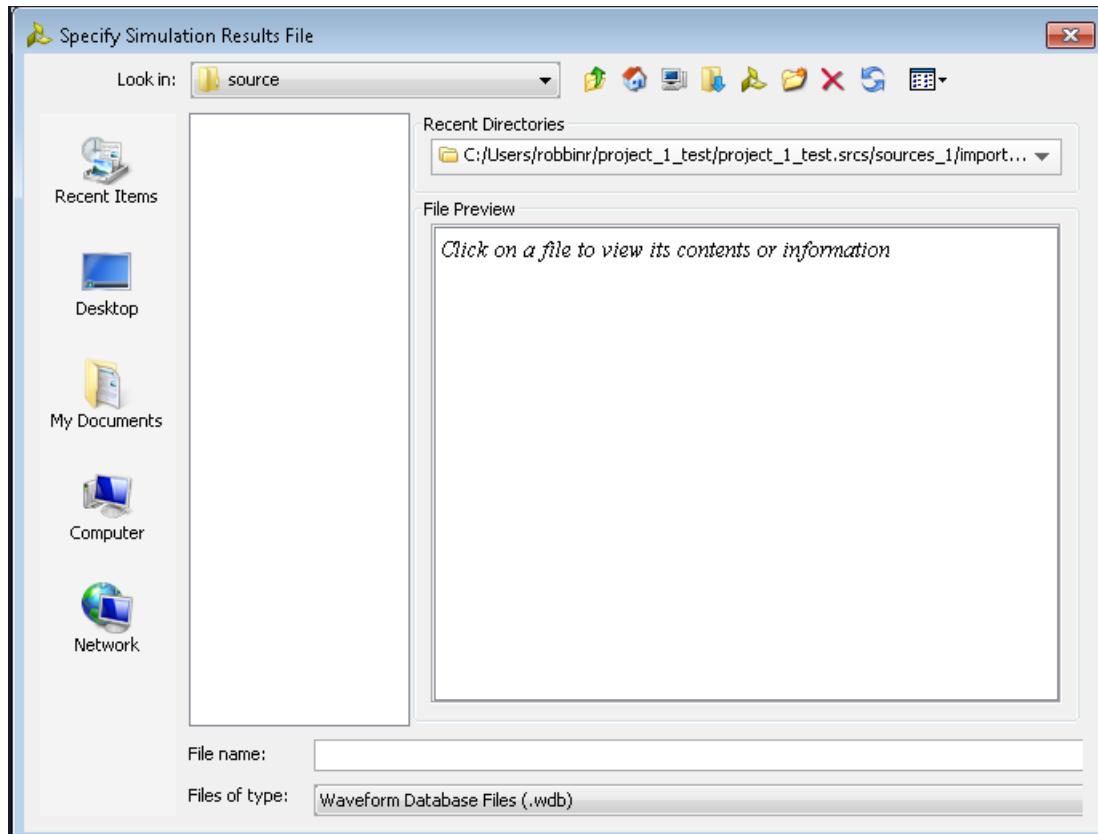


図 5-1 : [Specify Simulation Results] ダイアログ ボックス

4. WDB ファイルを選択します。

波形ビューのない [Simulation] ビューが表示されます。波形を表示するには、76 ページの「波形コンフィギュレーションの新規作成」および76 ページの「WCFG ファイルを開く」の手順に従ってください。

波形コンフィギュレーションの HDL オブジェクト

HDL オブジェクトを波形コンフィギュレーションに追加すると、波形ビューアーで HDL オブジェクトの「波形オブジェクト」が作成されます。波形オブジェクトは、HDL オブジェクトにリンクはされますが、別のものです。

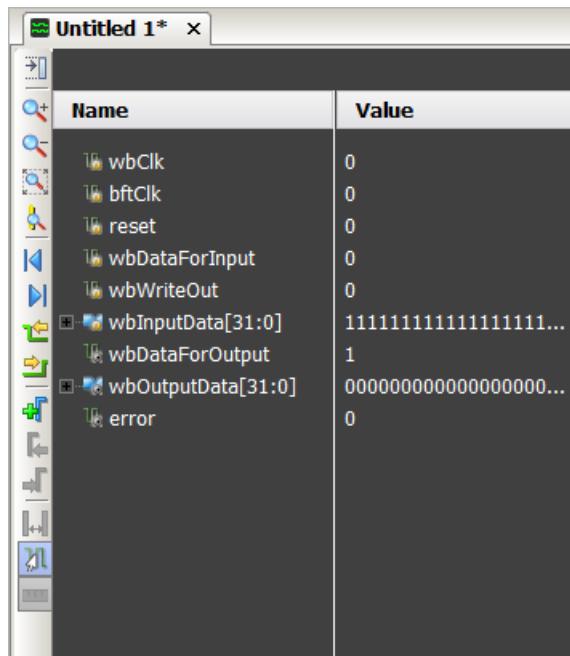
同じ HDL オブジェクトから複数の波形オブジェクトを作成でき、各波形オブジェクトの表示プロパティを別々に設定できます。

たとえば、myBus という HDL オブジェクトの波形オブジェクトの 1 つの値を 16 進数で表示し、myBus の別の波形オブジェクトを 10 進数で表示させることができます。

仕切り、グループ、仮想バスなどの別の種類の波形オブジェクトも波形コンフィギュレーションに表示できます。

HDL オブジェクトから作成された波形オブジェクトは、特に「デザイン波形オブジェクト」と呼ばれます。これらのオブジェクトは、それを示すアイコンで表示されます。デザイン波形オブジェクトの場合、アイコンの背景でそのオブジェクトがスカラー  なのか、Verilog ベクター  や VHDL レコード  などの複合型  なのかがわかります。

図 5-2 に、波形コンフィギュレーションビューの HDL オブジェクトの例を示します。



Name	Value
wbClk	0
bftClk	0
reset	0
wbDataForInput	0
wbWriteOut	0
wbInputData[31:0]	1111111111111111...
wbDataForOutput	1
wbOutputData[31:0]	0000000000000000...
error	0

図 5-2: 波形 HDL オブジェクト

デザイン オブジェクトには、名前と値が表示されます。

- [Name] : デフォルトでは、HDL オブジェクトの名前が表示されます。表示されるのは名前だけで、オブジェクトの階層パスは表示されません。階層パスすべてを含めて名前を表示する、または表示するテキストを指定してカスタム名を割り当てるように変更することもできます。
- [Value] : オブジェクトの値を波形ビューのメイン カーソルに示される時間で表示します。値のフォーマットは、同じ HDL オブジェクトにリンクされたほかのデザイン波形オブジェクトのフォーマットに関係なく、および [Objects] ビューおよびソース コード ビューに表示される値のフォーマットに関係なく、変更できます。

[Scopes] ビューの使用

図 5-3 は、Vivado シミュレータの [Scopes] ビューを示しています。

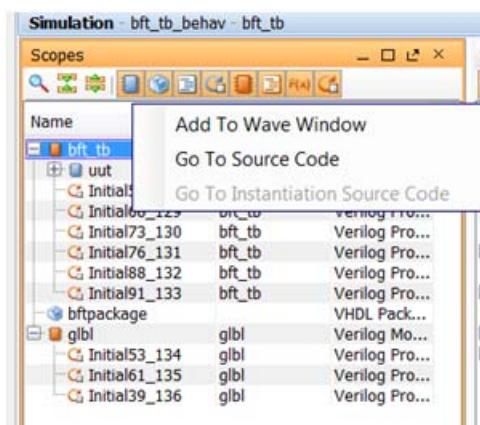


図 5-3: [Scopes] ビュー

[Scopes] ビュー内でスコープをフィルターするには、次のいずれかの方法を使用します。

- 一部のスコープを表示させないようにするには、次を実行します。

1つまたは複数のスコープ フィルター ボタンをクリックします。

- 特定の文字列を含むスコープに表示を限定するには、[Zoom] ボタンをクリックします。



テキスト ボックスに文字列を入力します。

スコープをクリックしても [Scopes] ビュー内のオブジェクトをフィルターできます。スコープを選択すると、[Scopes] ポップアップ メニューには次のオプションが含まれます。

- [Add to Wave Window] : 選択したスコープの表示可能な HDL オブジェクトすべてを波形コンフィギュレーションに追加します。

または、オブジェクトを [Objects] ビューから波形ビューの [Name] 列までドラッグ アンド ドロップします。



重要 : オブジェクトの波形には、オブジェクトがビューに追加されたときのシミュレーション時間のみが表示されます。波形コンフィギュレーションの作成や HDL オブジェクトの追加など、波形コンフィギュレーションへの変更は、WCFG ファイルを保存するまで永久ではありません。

- [Go To Source Code] : 選択したスコープの定義でソース コードを開きます。

- [Go To Instantiation Source code] : Verilog モジュールおよび VHDL エンティティ インスタンスの場合、選択したインスタンスのインスタンシエーション時点でのソース コードを開きます。

ソース コード テキスト エディターでファイルの識別子にカーソルを置くと値が表示されます (80 ページの [図 5-4](#))。

重要 : この機能を使用するには、[Scopes] ビューで正しいスコープが選択されている必要があります。

```

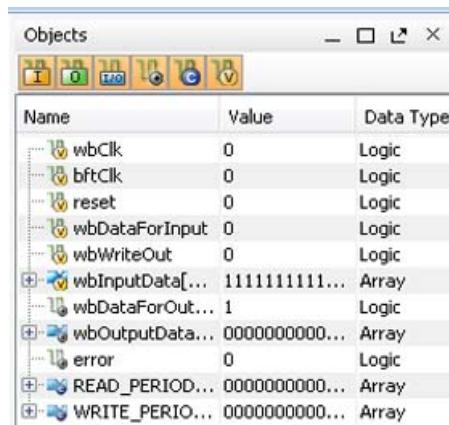
22 // 
23 //////////////////////////////////////////////////////////////////
24 
25 module bft_tb;
26 
27 // Inputs
28 reg wbClk;
29 reg bftClk;
30 reg reset;
31 reg wbDataForInput;
32 reg wbWriteOut;
33 reg [31:0] wbInputData;
34 
35 // Outputs
36 wire wbDataForOutput;
37 wire [31:0] wbOutputData;
38 wire error;
39 
40 // Instantiate the Unit Under Test (UUT)
41 bft uut (
42     .wbClk(wbClk),
43     .bftClk(bftClk),
44     .reset(reset),
45     .wbDataForInput(wbDataForInput),
46     .wbWriteOut(wbWriteOut),
47     .wbDataForOutput(wbDataForOutput),
48     .wbInputData(11111111111111111111111111111100),
49     .wbOutputData(wbOutputData),
50     .error(error)
51 );
52 
53 initial begin
54     // Initialize Inputs
55     wbClk = 0;

```

図 5-4 : 識別子が表示されたソース コード

[Objects] ビューの使用

図 5-5 は、Vivado シミュレータの [Objects] ビューを示しています。



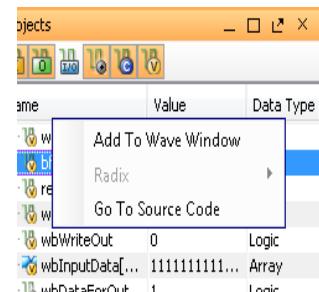
Name	Value	Data Type
wbClk	0	Logic
bftClk	0	Logic
reset	0	Logic
wbDataForInput	0	Logic
wbWriteOut	0	Logic
wbInputData[...]	1111111111...	Array
wbDataForOut[...]	1	Logic
wbOutputData[...]	0000000000...	Array
error	0	Logic
READ_PERIOD[...]	0000000000...	Array
WRITE_PERIOD[...]	0000000000...	Array

図 5-5 : [Objects] ビュー

一部の HDL オブジェクト タイプは、次の方法で非表示にできます。
1つまたは複数のオブジェクト フィルター ボタンをクリックします。ボタンの上にカーソルを置くと、オブジェクト タイプを示すツール ヒントが表示されます。

オブジェクトを選択すると、ポップアップ メニューには次のオプションが含まれます。

- [Add to Wave Window] : 選択したオブジェクトを波形コンフィギュレーションに追加します。
または、オブジェクトを [Objects] ビューから波形ビューの [Name] 列まで ドラッグ アンド ドロップします。
- [Radix] : 選択したオブジェクトの値を [Objects] ビューおよびソース コード ビューで表示する際に使用する数形式を選択します。
- [Go To Source Code] : 選択したオブジェクトの定義でソース コードを開きます。



ヒント : Verilog イベント、Verilog パラメーター、VHDL 定数、最大トレース可能サイズ (『Vivado Tcl コマンド リファレンス ガイド』(UG835)[\[参照 4\]](#) の trace_limit プロパティを参照) よりも多いエレメントなど、波形としては表示できない HDL オブジェクトもあります。または、Tcl コンソールに trace_limit -help と入力します。

波形のカスタマイズ

次のセクションでは、波形をカスタマイズする際に使用できるオプションについて説明します。

アナログ波形の使用

ここでは、アナログ波形の機能および要件について説明します。

基數およびアナログ波形

バスの値が数値として処理される方法は、バス波形オブジェクトの基數設定によって決まります。

- 2進数、8進数、16進数、ASCII、および符号なしの10進数の基數を使用すると、バスの値が符号なしの整数として処理されます。
- 0または1以外のビットを使用すると、全体の値が0として解釈されます。
- 符号付きの10進数基數を使用すると、バスの値が符号付き整数として解釈されます。
- 実数基數を使用すると、バスの値は固定小数点または浮動小数点の実数として処理されます。これは、[82ページの図5-6](#)に示す[Real Settings]ダイアログボックスの設定によって決まります。

波形の基數を[Real]に設定すると、オブジェクトの値が実数で表示されるようにできます。この基數を選択する前に、その値のビットの変換方法を波形ビューアーで指定しておく必要があります。

波形オブジェクトの基數を実数に設定する手順は、次のとおりです。

- [Real Settings]ダイアログボックスを開きます。
- 波形コンフィギュレーションビューでHDLオブジェクトを選択し、右クリックのポップアップメニューから開きます。

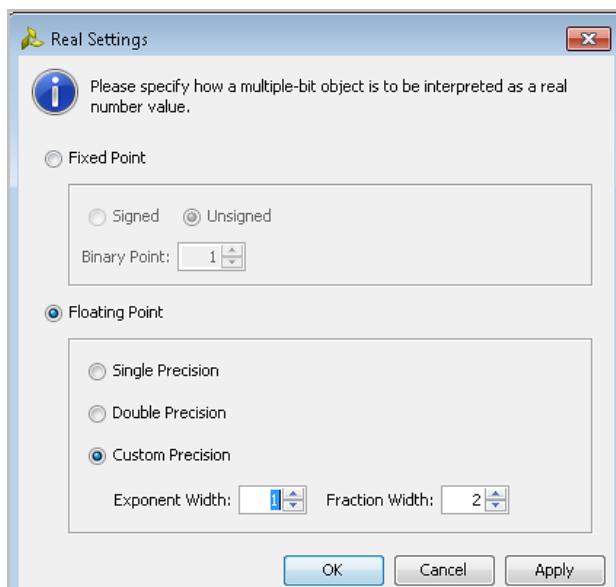


図5-6:[Real Settings]ダイアログボックス

[Real Setting]ダイアログボックスでは、次のオプションを設定できます。

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

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

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



ヒント: 行番号を分けるラインが表示されない場合は、86 ページの「[Waveform Options] スライドアウト」でオンにして、表示されるようにします。

波形のアナログ表示

HDL バスオブジェクトをアナログ波形で表示して予測される波形を出力する場合、HDL オブジェクトのデータの性質と一致する基数を選択することが重要になります。

次に例を示します。

- バスでエンコードされるデータが 2 の補数の符号付き整数の場合は、符号付きの基数を選択する必要があります。
- データが IEEE 形式にエンコードされる浮動小数点の場合は、基数に real を選択する必要があります。

アナログ波形表示のカスタマイズ

アナログ波形の表示は次のようにカスタマイズできます。

- 波形ビューの [Name] でバスを右クリックします。
- [Waveform Style] をクリックし、次のいずれかをクリックします。
 - [Analog] : デジタル波形をアナログに設定します。
 - [Digital] : アナログ波形オブジェクトをデジタルに設定します。
 - [Analog Settings] : [Analog Settings] ダイアログ ボックスを開きます。

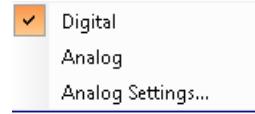


図 5-7 に、アナログ波形表示を設定する [Analog Settings] ダイアログ ボックスを示します。



図 5-7 : [Analog Settings] ダイアログ ボックス

[Analog Settings] ダイアログ ボックスでは、次のオプションを設定できます。

- [Row Height] : 選択した波形オブジェクトの高さをピクセルで指定します。行の高さを変更しても波形の垂直方向の表示域は変わりませんが、波形の高さの伸縮が変わります。
- アナログとデジタルを切り替えるとき、行の高さはそれぞれに合った適切なデフォルトの高さに設定されます（デジタルの場合は 20、アナログの場合は 100）。
- [Y Range] : 波形エリアに表示される数値の範囲を指定します。
 - [Auto] : 表示されている時間の範囲の値が現在の範囲を超えたときに、表示範囲が拡大されます。
 - [Fixed] : 時間範囲を一定にします。

- [Min] : 波形エリアの一番下に表示される値を指定します。
 - [Max] : 波形エリアの一番上に表示される値を指定します。
- どちらの値も浮動小数点として指定できますが、波形オブジェクトの基数が整数の場合、値は整数に切り捨てられます。
- [Interpolation Style] : データ ポイントを接続するラインの描画方法を指定します。
 - [Linear] : 2 つのデータ ポイント間のラインを直線にします。
 - [Hold] : 2 つのデータ ポイントのうち、左のポイントから右のポイントの X 軸に向かって水平ラインを描画し、そのラインから右のポイントに向かって別のラインを L 字型に描画します。
 - [Off Scale] : 波形エリアの Y 軸を超えた値をどのように描画するかを指定します。
 - [Hide] : 範囲外にある値を非表示にします。波形エリアの上下の範囲外にあるものは、範囲内に戻るまでは非表示になります。
 - [Clip] : 範囲外にある値は変更され、波形エリアの上下境界線を超えると範囲内に戻るまでは水平ラインとして表示されます。
 - [Overlap] : 波形エリアの境界線を越えていて、ほかの波形と重なっていても、波形ウィンドウの境界に達するまでは波形の値がどこにあっても波形が描画されます。
 - [Horizontal Line] : 指定した値で水平方向のラインを描画するかどうか指定します。このチェックボックスがオンの場合、グリッド ラインが指定した Y 軸の位置で描画されます(値が波形の Y 軸の範囲内にある場合)。
- [Min] および [Max] の場合と同様、Y 軸の値には浮動小数点値を指定できますが、選択した波形オブジェクトの基数が整数の場合は、整数値に切り捨てられます。



重要: ズーム設定は波形コンフィギュレーションには保存されません。

基数

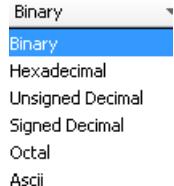
バスのデータ型を理解することは重要です。デジタルおよびアナログの波形オプションを効果的に使用するには、基数設定とデータ型の関係を知っておく必要があります。基数設定およびそのアナログ波形解析への影響については、83 ページの「波形のアナログ表示」を参照してください。

デフォルト基数の変更

デフォルトの波形基数では、明示的に設定していない基数の波形オブジェクトすべての値が数形式で制御されます。デフォルトの波形基数は、binary です。

デフォルトの波形基数を変更するには、次の手順に従ってください。

1. 波形ビューのサイドバーで [Waveform Options] ボタンをクリックし、 波形オプションのビューを開きます。
2. [General] ページで [Default Radix] ドロップダウン メニューをクリックします。
3. ドロップダウン リストから基数を選択します。



波形オブジェクトごとの基数の変更

各波形オブジェクトの基数を次のように変更できます。

1. [Objects] ビューでバスを選択します。
2. [Radix] をクリックし、ドロップダウン メニューからフォーマットを選択します。
 - [Binary] (2 進数)
 - [Hexadecimal] (16 進数)

- [Unsigned Decimal] (符号なし 10 進数)
- [Signed Decimal] (符号付き 10 進数)
- [Octal] (8 進数)
- [ASCII] (デフォルト)



重要 :[Objects] ビューで基底を変更しても、波形ビューまたは [Tcl Console] の値は変更されません。波形ビューで個々の波形オブジェクトの基底を変更するには、波形ビューのポップアップメニューを使用してください。

波形オブジェクトの命名

オブジェクト名は変更したり、表示したり、表示を変更したりできます。

オブジェクト名の変更

デザイン波形オブジェクト、仕切り、グループ、仮想バスなどの波形コンフィギュレーションに含まれる波形オブジェクトの名前は変更できます。

- [Name] 列でオブジェクト名を選択します。
- 右クリックし、[Rename] をクリックします。
- [Rename] ダイアログ ボックスが開きます。
- [Rename] ダイアログ ボックスに新しい名前を入力し、[OK] をクリックします。

波形コンフィギュレーションのデザイン波形オブジェクトの名前を変更しても、その下位の HDL オブジェクトの名前は変わりません。



ヒント :波形オブジェクトの名前を変更すると、名前表示モードが [Custom] に変わります。元の表示モードに戻すには、表示モードを次のセクションで説明するように [Long] または [Short] に戻す必要があります。

オブジェクト名の表示変更

完全な階層名 (long name)、信号またはバス名のみ (short name)、または各デザイン波形オブジェクトのカスタム名を表示できます。オブジェクト名は、波形コンフィギュレーションの [Name] 列に表示されます。名前が非表示になっている場合は、次の操作を実行します。

- 名前全体が表示されるように [Name] 列の幅を調整します。
- [Name] 列でスクロールバーを使用して名前を表示します。

表示名を変更するには、次の手順に従います。

- 信号名またはバス名を 1 つ以上選択します。複数の信号を選択する場合は、Shift キーまたは Ctrl キーを押しながらクリックします。
- [Name] をクリックし、次を選択します。
 - [Long] : 完全な階層名を表示します。
 - [Short] : 信号名またはバス名のみを表示します。
 - [Custom] : 信号のカスタム名を表示します。詳細は、[85 ページの「オブジェクト名の変更」](#) を参照してください。

注記 :[Long] および [Short] の名前はデザイン波形オブジェクトに対してのみ意味があります。他のオブジェクト (仕切り、グループ、仮想バス) はデフォルトで [Custom] 名で表示され、[Long] および [Short] 名に対する ID 文字列が表示されます。

バス ビット順の反転

波形コンフィギュレーションでバスビット順を反転して、MSB の最上位バイト (ビッグ エンディアン) および LSB の最下位バイトのビット順を切り替えることで、バス値を表示できます。

ビット順序を逆にするには、次の手順に従います。

1. バスを選択します。
2. 右クリックし、[Reverse Bit Order] をクリックします。

これでバスビットの順序が逆になります。[Reverse Bit Order] コマンドの横にチェックマークが表示され、適用されていることが示されます。

[Waveform Options] スライドアウト

[Waveforms Options] ボタン  をクリックすると、[Waveforms Options] スライドアウトが開きます (図 5-8)。

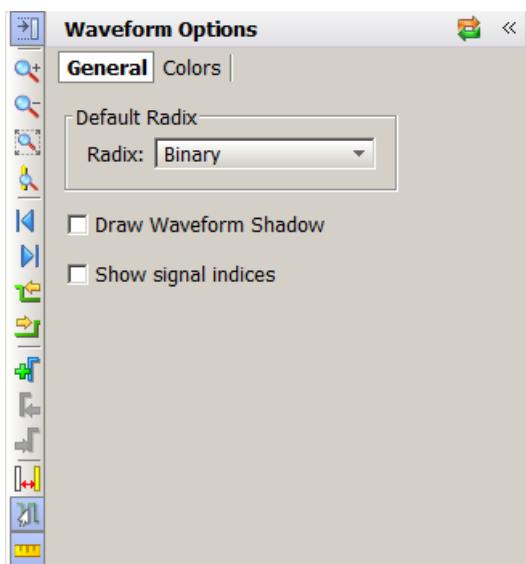


図 5-8 : [Waveform Options] スライドアウト

[General Waveform Options] には、次のオプションがあります。

- [Default Radix] : 新しく作成した波形オブジェクトに使用する数値形式を設定します。
- [Draw Waveform Shadow] : 影付きの波形が作成されます。
- [Show signal indices] : オンにすると、波形オブジェクト名の左に行番号を表示できます。行番号を分けているラインをドラッグすると、波形オブジェクトの高さを変更できます。
- [Colors] ページでは、波形ビュー内のアイテムの色を設定できます。

波形表示の制御

波形表示は、次のように制御できます。

- [Objects] ビューのサイドバーにあるズーム ボタン

- マウス ホイールを使用したズームの組み合わせ
- Vivado IDE Y 軸のズーム機能
- Vivado シミュレーションの X 軸のズーム機能 Vivado IDE の X 軸のズーム機能の詳細は、『Vivado Design Suite ユーザーガイド：Vivado IDE の使用』(UG893) [参照 3] を参照してください。

注記：ほかの Vivado のグラフィック ビューとは異なり、波形ビューの拡大/縮小は Y 軸に関係なく、X (時間) 軸にのみ適用されます。このため、ビューを拡大/縮小する範囲を指定する [Zoom Range X] が、ほかの Vivado ビューの [Zoom to Area] の代わりに使用されます。

ズーム機能の使用

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



マウス ホイールを使用したズーム

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

Y 軸のズーム機能

X 軸方向のズームでサポートされている機能に加え、アナログ波形の場合は、図 5-9 に示す追加のズーム機能があります。

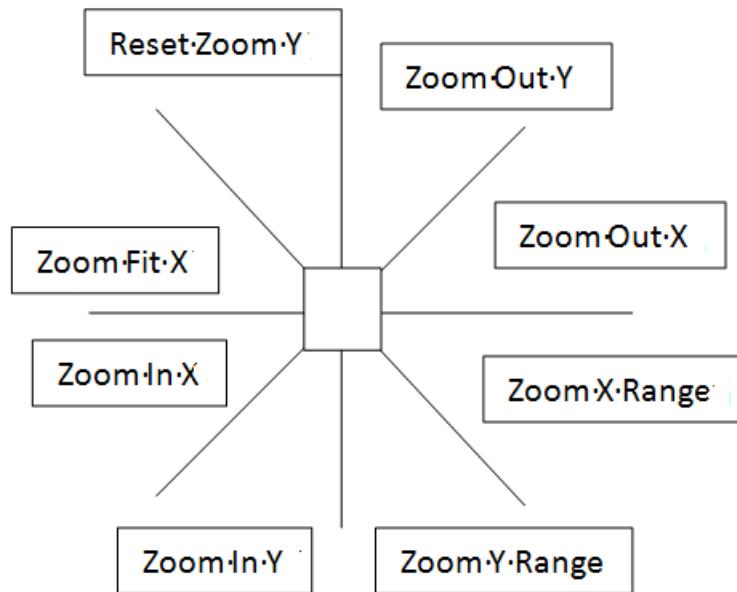


図 5-9: アナログ ズームのオプション

ズーム機能を使用するには、マウスの左ボタンを押したまま、図で示されている方向にマウスをドラッグします。この図の中央がマウスの位置です。

次の追加ズーム機能があります。

- [Zoom Out Y] : 開始点からマウス ボタンを放した位置までの距離により、2 のべき乗分 Y 軸方向にズーム アウトします。開始点のマウス位置の Y 値をそのまま維持してズームが実行されます。

- [Zoom Y Range] : 縦方向にラインを描き、マウス ボタンを離した位置までの Y 軸の範囲を表示します。
- [Zoom In Y] : 開始点からマウス ボタンを放した位置までの距離により、2 のべき乗分 Y 軸方向にズーム インします。開始点のマウス位置の Y 値をそのまま維持してズームが実行されます。
- [Reset Zoom Y] : [Reset Zoom Y] : 波形ウィンドウに現在表示されている値に Y の範囲をリセットし、Y の範囲 モードを [Auto] に設定します。

Y 軸の方向のズーム機能はすべて Y の範囲のアナログ値を設定します。[Reset Zoom Y] は Y の範囲を [Auto] に設定しますが、ほかのズーム機能は [Fixed] に設定します。

次の制限に注意してください。

- 実数での最大バス幅は 64 ビットです。
- Verilog real および VHDL real はアナログ波形のようにサポートされません。
- 浮動小数点では 32 ビットおよび 64 ビットの配列のみがサポートされています。

波形の分類

ここでは、波形内の情報を分類するオプションについて説明します。

グループの使用

グループとは、波形コンフィギュレーションに波形オブジェクトを追加するための拡張可能な入れ物のことです、これにより関連した波形オブジェクトセットが整頓できます。グループ自体は波形データを表示しませんが、その内容の表示/非表示を切り替えることができます。グループは追加、変更、削除できます。

グループを追加するには、次の手順に従います。

1. 波形ビューで、グループに追加する波形オブジェクトを 1 つまたは複数選択します。
2. [Edit] → [New Group] をクリックするか、右クリックして [New Group] をクリックします。

これにより、選択した波形オブジェクトを含むグループが波形コンフィギュレーションに追加されます。

グループは、グループアイコンで表されます。HDL オブジェクトをドラッグ アンド ドロップして、グループに信号やバスを追加することもできます。

波形コンフィギュレーションファイルを保存すると、新しいグループとそれに含まれる波形オブジェクトも保存されます。

グループは、次の方法で移動または削除できます。

- グループを移動するには、[Name] 列のグループ名を別の位置にドラッグ アンド ドロップします。
- グループを削除するには、グループを選択して [Edit] → [Wave Objects] → [Ungroup] をクリックするか、グループを右クリックして [Ungroup] をクリックします。グループに含まれていた波形オブジェクトは、波形コンフィギュレーションの一番上に配置されます。

グループ名も変更できます。詳細は、85 ページの「オブジェクト名の変更」を参照してください。



注意 : Delete キーを押すと、グループと、それに含まれている波形オブジェクトが波形コンフィギュレーションから削除されます。

仕切りの使用

仕切りは、HDL オブジェクトを見やすく区切れます。波形コンフィギュレーションに仕切りを追加するには次の手順に従います。

1. 波形ビューの [Name] 列で、下に仕切りを追加する信号をクリックします。
2. [Edit] → [New Divider] をクリックするか、右クリックして [New Divider] をクリックします。

新しい区切りマークは、波形コンフィギュレーション ファイルを保存したときに保存されます。

仕切りは、次の方法で移動または削除できます。

- 仕切りを移動するには、名前を別の位置にドラッグ アンド ドロップします。
- 仕切りを削除するには、Delete キーを押すか、または右クリックしてポップアップ メニューから [Delete] をクリックします。

仕切りの名前も変更できます。詳細は、85 ページの「オブジェクト名の変更」を参照してください。

仮想バスの使用

仮想バスを波形コンフィギュレーションに追加すると、論理スカラーおよびベクターを追加できます。

仮想バスはバス波形を表示します。バス波形の値は、仮想バスの下に表示される上から順番に、追加されたスカラーおよび配列から対応する値を取り出して、1 次元のベクターに平坦化することで作成されます。

仮想バスを追加するには、次の手順に従います。

1. 波形コンフィギュレーションで、仮想バスに追加する波形オブジェクトを 1 つまたは複数選択します。
2. [Edit] → [New Virtual Bus] をクリックするか、右クリックして [New Virtual Bus] をクリックします。

仮想バスは、仮想バス アイコン  で表されます。

信号名やバス名をドラッグ アンド ドロップすると、仮想バスに論理スカラーおよび配列を移動できます。波形コンフィギュレーション ファイルを保存すると、新しい仮想バスとそれに含まれるものも保存されます。また、仮想バスの名前を波形の別位置にドラッグ アンド ドロップして移動できます。

仮想バスの名前を変更するには、85 ページの「オブジェクト名の変更」を参照してください。

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



注意：Delete キーを押すと、仮想バスと、それに含まれている HDL オブジェクトが波形コンフィギュレーションから削除されます。

波形の解析

ここでは、波形内のデータを解析する機能について説明します。

カーソルの使用

カーソルは、時間を一時的に示すために使用します。2つの波形エッジ間の距離(時間)を計測する場合には、カーソルを頻繁に移動します。



ヒント: WCFG ファイルには、カーソル位置は記録されません。複数の計測の時間軸を設定する場合やシミュレーションでの重要なイベントを示す場合など、一時的ではないインジケーターを設定する必要がある場合は、波形ビューにマーカーを追加します。詳細は、91 ページの「マーカーの使用」を参照してください。

メイン カーソルと 2 つ目のカーソル

波形を 1 回クリックすると、メイン カーソルが配置されます。

2 つ目のカーソルを配置するには、Ctrl キーを押しながらクリックし、マウスを左または右にドラッグします。カーソルの上に位置を示すフラグが表示されます。または、Shift キーを押したまま波形のどこかをクリックします。

2 つ目のカーソルがオンになっていない場合、この動作により 2 つ目のカーソルがメイン カーソルの現在の位置に設定され、メイン カーソルがクリックした位置に配置されます。

注記: 2 つ目のカーソルの位置を保持しながらメイン カーソルの位置を変更するには、Shift キーを押しながらクリックします。2 つ目のカーソルをドラッグして配置する場合、最小間隔以上ドラッグしないと 2 つ目のカーソルは表示されません。

カーソルの移動

カーソルを移動するには、ポインターが手のひらのマークになるまでマウスを動かし、クリックして次の位置までカーソルをドラッグします。

カーソルをドラッグする際、[Snap to Transition] がオンになっていると(デフォルト)、中が塗りつぶされていない丸または中が塗りつぶされた丸が表示されます。

- 中が塗りつぶされていない丸 は、選択した信号の波形の遷移間にあることを示します。
- 中が塗りつぶされた丸 は、カーソルがマウスの下またはマーカーの波形の遷移でロックされていることを示します。

カーソル、マーカー、フロートしているルーラーがない場所をクリックすると、2 つ目のカーソルが非表示になります。

次または前の遷移の検索

波形ビューのサイドバーには、メイン カーソルを現在の位置から選択した波形の次または前の遷移に移動させるためのボタンが含まれます。

メイン カーソルを波形の次または前の遷移に移動するには、次を実行します。

- 波形の波形オブジェクト名をクリックしてアクティブにしておきます。

これで波形オブジェクトが選択され、オブジェクトの波形が通常 も太いラインで表示されます。

- サイドバーの [Next Transition] または [Previous Transition] ボタン をクリックするか、キーボードの右矢印または左矢印キーを使用して、次または前の遷移に移動します。



ヒント: 複数の波形オブジェクトを一緒に選択すると、波形セットの一番近い遷移に移動します。

フロート ルーラーの使用

波形ウィンドウの上部にある標準ルーラーの絶対シミュレーション時間以外の時間ベースを使用して時間を計測するには、フロート ルーラーを使用すると便利です。

フロート ルーラーは表示（または非表示）にでき、ドラッグして波形ウィンドウで垂直方向の位置に変更することができます。このルーラーの時間ベース（時間 0）は 2 番目のカーソルで、このカーソルがない場合は、選択されたマークになります。

フロート ルーラー ボタン  およびフロート ルーラーは、2 番目のカーソルまたはマーカーがある場合にのみ表示されます。

1. ルーラーの表示/非表示を切り替えるには、次のいずれかを実行します。
 - 2 番目のカーソルを配置
 - マーカーを選択
2. [Floating Ruler] ボタンをクリックします。 

この操作は最初に 1 回だけ実行し、繰り返す必要はありません。フロート ルーラーは 2 番目のカーソルが置くたび、またはマーカーが選択されるたびに表示されます。

ルーラーを非表示にするには、このコマンドをもう一度クリックします。

マーカーの使用

波形内の重要イベントを恒久的にマークする必要がある場合はマーカーを使用します。マーカーを使用すると、マーカーが付けられたイベントに関連した時間を計測できます。

マーカーは、次のように追加、移動、削除できます。

- メイン カーソルの位置に波形コンフィギュレーションにマーカーを追加します。
 - 波形ビューの時間または遷移をクリックして、マーカーを追加する時間の箇所にメイン カーソルを置きます。
 - [Edit] → [Add Marker] をクリックするか、または [Add Marker] ボタンをクリックします。 

カーソル位置にマーカーが配置されます。マーカーがその位置に既にある場合は、若干オフセットされます。マーカーの時間が上部に表示されます。

- マーカーを波形ビューの別の位置に移動するには、ドラッグ アンド ドロップします。マーカー ラベル（マーカー上部またはマーカー ライン）をクリックしてドラッグします。
 - ドラッグシンボル  は、マーカーが移動可能であることを示します。マーカーをドラッグする際、[Snap to Transition] がオンになっていると（デフォルト）、中が塗りつぶされていない丸または中が塗りつぶされた丸が表示されます。
 - 中が塗りつぶされている丸  は、選択した信号の波形の遷移地点、または別のマーカー上であることを示します。
 - マーカーの場合は、丸は白く塗りつぶされています。
 - 中が塗りつぶされていない丸  は、マーカーがマウスの下またはマーカーの波形の遷移でロックされていることを示します。

新しい位置にマーカーをドロップするには、マウスのボタンを放します。

- 1 つのコマンドでマーカーを 1 つ、またはすべて削除できます。マーカーを右クリックして、次のいずれかの操作を実行します。
 - マーカーを 1 つ削除するには、ポップアップメニューから [Delete Marker] をクリックします。
 - マーカーをすべて削除するには、ポップアップメニューから [Delete All Markers] をクリックします。

注記：また、Delete キーを使用して選択したマーカーを削除することもできます。

非プロジェクト モードの使用

次は、Tcl コンソールまたはバッチ スクリプトないで使用できるコマンドの一部です。これらのコマンドの詳細については、『Vivado Tcl コマンド リファレンス ガイド』(UG835)[\[参照 4\]](#) を参照してください。

Vivado シミュレータの GUI の起動

Vivado シミュレータを起動するには、次を入力します。

```
xsim -gui -s <snapshot_name>
```

空の波形コンフィギュレーションが表示されます。

デザイン階層の表示

[Scopes] および [Objects] ビューを使用する代わりに、Tcl コンソールに `current_scope` と入力して現在のスコープを設定または表示することもできます。

スコープおよびオブジェクトをリストするには、現在のスコープ内で `report_scopes` および `report_values` コマンドをそれぞれ使用します。

波形ビューへの HDL オブジェクトの追加

各 HDL オブジェクトやオブジェクト セットを波形ビューに追加するには、Tcl コンソールに次を入力します。

```
add_wave <HDL_objects>
```

`add_wave` コマンドを使用すると、HDL オブジェクトへの絶対パスまたは相対パスを指定できます。

たとえば、現在のスコープが `/bft_tb/uut` の場合、`uut` の下のリセット レジスタへの絶対パスは `/bft_tb/uut/reset`、相対パスは `reset` になります。



ヒント : `add_wave` コマンドには、HDL スコープおよび HDL オブジェクトが指定できます。`add_wave` にスコープを指定した場合、[Sources] ビューの [Add To Wave Window] コマンドと同じ動作になります。

波形コンフィギュレーションの保存

波形コンフィギュレーションを WCFG ファイルに保存するには、Tcl コンソールに次を入力します。

```
save_wave_config <filename.wcfg>
```

指定したコマンド引数の名前で WCFG ファイルが保存されます。

表示された値の数形式の変更

表示された値の数形式を変更するには、Tcl コンソールに次を入力します。

```
set_property radix <radix> [current_sim]
```

`<radix>` は、`bin`、`unsigned`、`hex`、`dec`、`ascii`、`oct` のいずれかになります。

Vivado IDE 外でサードパーティシミュレータを使用したシミュレーションの実行

概要

ザイリンクスでは、次のサードパーティシミュレータがサポートされます。

- Mentor Graphics QuestaSim/ModelSim
- Cadence Incisive Enterprise Simulator (IES)
- Synopsis VCS および VCS MX
- Aldec Active-HDL および Rivera-PRO*

注記 : * Aldec シミュレータは互換性はありますが、ザイリンクス テクニカル サポートではサポートされません。サポートに関する問題については、Aldec にお尋ねください。

ModelSim は、Vivado™ Integrated Design Environment (IDE) からもサポートされます。Vivado IDE の使用については、『Vivado Design Suite ユーザー ガイド : Vivado IDE の使用』(UG893)[\[参照 3\]](#) を参照してください。

この付録では、Vivado IDE 外でシミュレーションを実行する方法を簡単に説明します。

サポートされるサードパーティシミュレーションツールおよびベンダーについては、『ザイリンクス デザインツール : リリース ノート ガイド』(UG631)[\[参照 1\]](#) を参照してください。

modelsim.ini ファイル

ModelSim または QuestaSim の使用をサポートするためには、これらのシミュレーターで使用できるようにザイリンクス シミュレーション ライブラリをコンパイルする必要があります。Vivado IDE の Tel コマンドに `compile_simlib` と入力し、ModelSim および QuestaSim シミュレータのシミュレーション ライブラリをコンパイルします。

ライブラリがコンパイルされると、シミュレーターは `modelsim.ini` ファイルを使用してこれらのコンパイル済みライブラリを参照します。`modelsim.ini` ファイルは、デフォルトの初期化ファイルで、リファレンス ライブラリパス、最適化、コンパイラ、シミュレーターの設定を指定する制御変数が含まれます。

[Simulation] ページには、[QuestaSim/ModelSim] を選択した場合、ライブラリ ディレクトリ用の追加フィールドが表示されます。

Vivado ツールの検索順は、次のとおりです。

- [Simulation Settings] の [Compiled library location] オプションで指定したディレクトリ
- このオプションは、[General] ページで [QuestaSim/ModelSim] がターゲット シミュレータとして設定されている場合のみ [Simulation] ページに表示されます。

- `compxlib compile_simlib` の実行時に設定された `compxlib.compiled_library_dir` 変数で指定されるパス
- MODELSIM 環境変数で定義されたパス



重要: `modelsim.ini` ファイルがこれらのディレクトリにない場合は、ザイリンクス プリミティブを含めるデザインをシミュレーションできません。

RTL/ビヘイビアシミュレーションの実行

次は、ザイリンクス デザインをシミュレーションする際の手順です。

1. `compile_simlib` コマンドを使用してシミュレーションライブラリをコンパイルします。
2. ソース ファイルを集めて、テストベンチを作成します。デザインに IP が含まれる場合は、『Vivado Design Suite ユーザー ガイド : IP を使用したデザイン』(UG896)[[参照 8](#)] を参照してください。
3. Verilog を使用する場合は、`glbl.v` をコンパイルします。詳細は、[第 2 章の「グローバル リセットおよびトライステート」](#) を参照してください。
4. デザインに SecureIP が含まれる場合は、次を実行します。
 - a. ModelSim の場合 : プリコンパイル ライブラリを使用するには、VSIM で `-L` オプションを使用してライブラリを指定します。

例 :

```
vsim -t ps -L secureip -L unisims_ver work.<testbench> work.glbl
```

- b. IES 次のいずれかを使用します。
 - IES の 1 ステップ プロセス

この場合、ザイリンクス ライブラリをコンパイルするのに、`compxlib` を実行する必要はありません。次の例に示すように、1 つのオプションのみを追加する必要があります。

```
-f $XILINX_PLANAHEAD/data/secureip/ncsim/ies_secureip_cell.list.f
```

- IES の 1 ステップ プロセスの例

```
irun design>.v testbench>.v $XILINX_PLANAHEAD/data/verilog/src/glbl.v \
-f $XILINX_PLANAHEAD/data/secureip/ncsim/ies_secureip_cell.list.f \
-y $XILINX_PLANAHEAD/data/verilog/src/unisims +libext+.v \
-y $XILINX_PLANAHEAD/data/verilog/src/unimacro +libext+.v \
-y $XILINX_PLANAHEAD/data/verilog/src/retarget +libext+.v \
+access+r+w
```

- IES の 3 ステップ プロセス

SecureIP ライブラリをコンパイルし、`CDS.lib` および `HDL.var` に加えます。

- c. VCS : 次のコマンドをシミュレーション コマンド ラインに追加し、SecureIP ファイルがシミュレータで使用されるようにします。

```
-f $XILINX_PLANAHEAD/data/secureip/vcs/vcs_secureip_cell.list.f
```

VCS の例

```
vcs -f $XILINX_PLANAHEAD/data/secureip/vcs/vcs_secureip_cell.list.f \
-y $XILINX_PLANAHEAD/data/verilog/src/unisims\
-y $XILINX_PLANAHEAD/data/verilog/src/XILINXCORELIB\
+incdir+$XILINX_PLANAHEAD/data/verilog/src +libext+.v\
```

```
$XILINX_PLANAHEAD/data/verilog/src/glbl.v \
-Mupdate -R <testfixture>.v <design>.v
```

5. デザインをコンパイルしてシミュレーションします。指定したシミュレータのユーザー ガイドを参照してください。

注記: 正しくシミュレーションされるには、UNISIM、XILINXCORELIB、SECUREIP、UNIMACRO、およびUNIFAST ライブラリが参照されている必要があります。詳細は、[第2章の「シミュレーションライブラリ」](#)を参照してください。

ネットリストシミュレーションの実行

ネットリストシミュレーションプロセスの手順は[「RTL/ビヘイビアシミュレーションの実行」](#)と同じになります。

1. シミュレーションライブラリをコンパイルします。
2. シミュレーション用のファイルを集めます(例:[66ページの図4-2](#))。
 - a. RTLシミュレーションに使用されたシミュレーション テストベンチは、ほとんどのデザインで再利用できます。
 - b. `write_verilog` または `write_vhdl` を使用し、シミュレーションネットリストを生成します。
 - 論理ネットリストの場合は、`write_verilog -mode funcsim` を使用します。
 - タイミングネットリストの場合は、`write_verilog -mode timesim` を使用します。
 - SDFには、`write_sdf` を使用します。

次は、その具体例です。

```
synth_design -top top -part xc7k70tfbg676-2 -flatten_hierarchy none
open_run synth_1 -name netlist_1
write_verilog -mode funcsim test_synth.v
write_verilog -mode timesim -sdf_file test.sdf test_synth_timing.v
write_sdf test.sdf
```

これで、`test.sdf` を使用し、論理シミュレーションには `test_synth.v`、タイミングシミュレーションには `test_synth_timing.v` を使用できるようになりました。

3. Verilogを使用する場合は、`lbl.v`をコンパイルします。詳細は、[第2章の「グローバルリセットおよびトライステート」](#)を参照してください。
4. SECUREIPライブラリを使用する場合は[94ページの手順4](#)を参照してください。
5. デザインをコンパイルしてシミュレーションします。使用するシミュレータのユーザー ガイドを参照してください。

注記: 正しくシミュレーションされるには、UNISIM、XILINXCORELIB、SECUREIP、およびUNIFAST ライブラリが参照されている必要があります。詳細は、[第2章の「シミュレーションライブラリ」](#)を参照してください。

タイミングシミュレーションの実行

タイミングシミュレーションでは SIMPRIM ライブラリが使用されます。タイミングシミュレーションプロセス中は、正しいライブラリを参照する必要があります。



重要: UNIMACRO、XILINXCORELIB、UNIFAST、または UNISIM ライブラリは、タイミング シミュレーションには必要ありません。

タイミング シミュレーションでは、ほかのオプションを使用して、シミュレータでパルスが正しく処理されるようにする必要があります。シミュレータ コマンドには、次のオプションを追加する必要があります。

- Mentor Graphics QuestaSim/ModelSim および Synopsys VCS
+transport_int_delays +pulse_int_e/0 +pulse_int_r/0
- Cadence IES/IUS
-pulse_r 0 -pulse_int_r

サードパーティ シミュレータでの SAIF のダンプ

SAIF (Switching Activity Interchange Format) に関する詳細は、[25 ページの「SAIF のダンプ」](#) を参照してください。

ModelSim

ModelSim では、専用の消費電力コマンドを使用して、SAIF ファイルをダンプします。

1. 次のように入力し、ダンプする範囲または信号を指定します。

```
power add <hdl_objects>
```

2. 特定時間 (または run -all)、シミュレーションを実行します。

3. 次のように入力し、消費電力レポートをダンプします。

```
power report -all filename.saif
```

各コマンドの使用方法や詳細については、ModelSim のユーザー ガイドを参照してください。

do ファイルの例

```
power add tb/fpga/*
run 500us
power report -all -bsaif routed.saif
quit
```

VCS

VCS には、SAIF を特定要件で生成するための消費電力コマンドが含まれます。

1. 次のように入力し、生成する範囲または信号を指定します。

```
power <hdl_objects>
```

2. SAIF ダンプをイネーブルにします。シミュレータ GUI のコマンド ラインを使用できます。

```
power -enable
```

3. 特定時間、シミュレーションを実行します。

4. 次のように入力し、消費電力のダンプをディスエーブルにし、SAIF をレポートします。

```
power -disable
power -report filename.saif
```

各コマンドの使用方法や詳細については、VCS のユーザー ガイドを参照してください。

Tcl コマンド ファイルの例

```
power tb/fpga/*
power -enable
run 500us
power -disable
power -report routed.saif
```

IES

IES では、Vivado シミュレータの `dumpvars` コマンドに似た `dumpsaif` コマンドが使用されます。

1. 次のように入力し、ダンプする範囲と出力する SAIF ファイルの名前を指定します。

```
dumpsaif -scope hdl_objects -output filename.saif
```

2. シミュレーションを実行します。

3. 次のように入力して SAIF ダンプを終了します。

```
dumpsaif -end
```

各コマンドの使用方法や詳細については、IES のユーザー ガイドを参照してください。

Tcl コマンド ファイルの例

```
dumpsaif -scope /tb/fpga/* -output routed.saif
run 500ns
dumpsaif -end
quit
```

Verilog および VHDL の例外

概要

この付録では、Verilog および VHDL サポートの例外についてリストします。

VHDL 言語サポートの例外

Vivado™ Integrated Design Environment (IDE) では、次がサポートされます。

- VHDL IEEE-STD-1076-1993
- Verilog IEEE-STD-1364-2001

これらの言語のコンストラクトの中には、Vivado シミュレータでサポートされないものもあります。次の表は、これらの例外をリストしています。

表 B-1: VHDL 言語サポートの例外

サポートされる VHDL コンストラクト	例外
abstract_literal	底付き定数として表示される浮動小数点はサポートされません。
aggregate	aggregate 内での choice 指示の混合はサポートされません。
alias_declaration	オブジェクト以外へのエイリアスは通常、特に次の場合サポートされません。 エイリアスのエイリアス subtype_indication なしのエイリアス宣言 エイリアス宣言のシグナチャ alias_designator としての演算子シンボル 演算子シンボルのエイリアス エイリアス宣言としての文字リテラル
alias_designator	alias_designator としての operator_symbol alias_designator としての character_literal
association_element	結合エレメントの実際のスライスについては、グローバル、ローカルな固定範囲を使用できます。
attribute_name	接頭語の後のシグナチャはサポートされません。
binding_indication	entity_aspect を使用しない binding_indication はサポートされません。
bit_string_literal	空の bit_string_literal (" ") はサポートされません。

表 B-1: VHDL 言語サポートの例外 (Cont'd)

サポートされる VHDL コンストラクト	例外
block_statement	guard_expression はサポートされません。たとえば、保護付きブロック、保護付き信号、保護付きターゲットおよび保護付き代入はサポートされません。
choice	case 文での choice としての aggregate の使用はサポートされません。
concurrent_assertion_statement	postponed はサポートされていません。
concurrent_signal_assignment_statement	postponed はサポートされていません。
concurrent_statement	wait 文を含む並列手続き呼び出しはサポートされません。
conditional_signal_assignment	オプションの一部であるキーワード guarded は、保護付き信号代入がサポートされないため、サポートされません。
configuration_declaration	コンフィギュレーションで使用される generate インデックスのローカルではない固定範囲はサポートされません。
entity_class	エンティティ クラスとしてのリテラル、ユニット、ファイル、グループはサポートされません。
entity_class_entry	グループ テンプレートと共に使用する目的ではオプションの <> はサポートされません。
file_logical_name	file_logical_name では、ワイルドカードを使用してストリング値を評価できますが、リテラル文字列および識別子のみがファイル名として使用できます。
function_call	function_call 内のパラメーター関連付けでは、スライス、インデックスおよびフォーマルの選択がサポートされません。
instantiated_unit	ダイレクト コンフィギュレーション インスタンシエーションはサポートされません。
mode	リンクエージおよびバッファ ポートは完全にはサポートされません。
options	guarded はサポートされません。
primary	primary が使用された場所で、アロケーターが拡張されます。
procedure_call	procedure_call 内のパラメーター関連付けでは、スライス、インデックスおよびフォーマルの選択がサポートされません。
process_statement	postponed プロセスはサポートされません。
selected_signal_assignment	オプションの一部であるキーワード guarded は、保護付き信号代入がサポートされないため、サポートされません。
signal_declaration	signal_kind はサポートされません。signal_kind はサポートされない保護付き信号を宣言するために使用されます。
subtype_indicationd	分解された複合体(配列およびレコード)のサブタイプはサポートされません。

表 B-1: VHDL 言語サポートの例外 (Cont'd)

サポートされる VHDL コンストラクト	例外
waveform	unaffected はサポートされていません。
waveform_element	空の waveform エレメントは、保護付き信号にのみ関連するため、サポートされません。

Verilog 言語サポートの例外

次の表では、Verilog の言語サポートの例外についてリストします。

表 B-2: Verilog 言語サポートの例外

Verilog コンストラクト	例外
コンパイラ指示子のコンストラクト	
`celldefine	サポートなし
`endcelldefine	サポートなし
`undef	パラメーター指定可能な `define マクロをサポート
`unconnected_drive	サポートなし
`nounconnected_drive	サポートなし
属性	
attribute_instance	サポートなし
attr_spec	サポートなし
attr_name	サポートなし
プリミティブゲートおよびスイッチ タイプ	
cmos_switchtype	サポートなし
mos_switchtype	サポートなし
pass_en_switchtype	サポートなし
生成されたインスタンシエーション	
generated_instantiation	module_or_generate_item 選択肢はサポートされません。 1364-2001 Verilog 規格からの出力 generate_item_or_null ::= generate_conditional_statement generate_case_statement generate_loop_statement generate_block module_or_generate_item シミュレータでサポートされる出力 generate_item_or_null ::= generate_conditional_statement generate_case_statement generate_loop_statement generate_block generate_condition

表 B-2: Verilog 言語サポートの例外 (Cont'd)

Verilog コンストラクト	例外
genvar_assignment	<p>部分的にサポート。 すべての generate ブロックに名前を付ける必要があります。</p> <pre>1364-2001 Verilog 規格からの出力 generate_block ::= begin [: generate_block_identifier] { generate_item } end</pre> <p>シミュレータでサポートされる出力</p> <pre>generate_block ::= begin: generate_block_identifier { generate_item } end</pre>
ソース テキスト コンストラクト	
ライブラリ ソース テキスト	
library_text	サポートなし
library_descriptions	サポートなし
library_declaration	サポートなし
include_statement	ライブラリ マップ ファイル内の文を含めることを示します (IEEE 1364-2001、セクション 13.2)。これは、`include コンパイラ指示子は参照しません。
コンフィギュレーション ソース テキスト	
config_declaration	サポートなし
design_statement	サポートなし
config_rule_statement	サポートなし
default_clause	サポートなし
システム タイミング チェック コマンド	
\$skew_timing_check	サポートなし
\$timeskew_timing_check	サポートなし
\$fullskew_timing_check	サポートなし
\$nochange_timing_check	サポートなし
システム タイミング チェック コマンドの引数	
checktime_condition	サポートなし
PLA モデリング タスク	
\$async\$nand\$array	サポートなし
\$async\$nor\$array	サポートなし
\$async\$or\$array	サポートなし

表 B-2: Verilog 言語サポートの例外 (Cont'd)

Verilog コンストラクト	例外
\$sync\$and\$array	サポートなし
\$sync\$nand\$array	サポートなし
\$sync\$nor\$array	サポートなし
\$sync\$or\$array	サポートなし
\$async\$and\$plane	サポートなし
\$async\$nand\$plane	サポートなし
\$async\$nor\$plane	サポートなし
\$async\$or\$plane	サポートなし
\$sync\$and\$plane	サポートなし
\$sync\$nand\$plane	サポートなし
\$sync\$nor\$plane	サポートなし
\$sync\$or\$plane	サポートなし
Value Change Dump (VCD) ファイル	
\$dumppportson \$dumppports \$dumppportsoff \$dumppportsflush \$dumppportslimit \$vcplus	サポートなし

その他のリソース

ザイリンクス リソース

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

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

ザイリンクス資料で使用される用語集は、次を参照してください。

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

ソリューションセンター

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

参考資料

Vivado Design Suite 2012.4 の資料

1. 『ザイリンクス デザイン ツール：リリース ノート ガイド』(UG631)
2. 『ザイリンクス デザイン ツール：インストールおよびライセンス ガイド』(UG798)
3. 『Vivado Design Suite ユーザー ガイド：Vivado IDE の使用』(UG893)
4. 『Vivado Design Suite Tcl コマンド リファレンス ガイド』(UG835)
5. 『Writing Efficient Testbenches』(XAPP199)
6. 『7 シリーズ ライブドリブン ガイド』(HDL 用) (UG768)
7. 『Vivado Design Suite ユーザー ガイド：Tcl スクリプト機能の使用』(UG894)
8. 『Vivado ユーザー ガイド：IP を使用したデザイン』(UG896)
9. 『Vivado Design Suite チュートリアル：制約の使用』(UG945)