

ISim ユーザー ガイド

UG660 (v14.3) 2012 年 10 月 16 日



Xilinx is disclosing this user guide, manual, release note, and/or specification (the “Documentation”) to you solely for use in the development of designs to operate with Xilinx hardware devices. You may not reproduce, distribute, republish, download, display, post, or transmit the Documentation in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Xilinx expressly disclaims any liability arising out of your use of the Documentation. Xilinx reserves the right, at its sole discretion, to change the Documentation without notice at any time. Xilinx assumes no obligation to correct any errors contained in the Documentation, or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information.

THE DOCUMENTATION IS DISCLOSED TO YOU “AS-IS” WITH NO WARRANTY OF ANY KIND. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DOCUMENTATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS. IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOSS OF DATA OR LOST PROFITS, ARISING FROM YOUR USE OF THE DOCUMENTATION.

© Copyright 2012 Xilinx, Inc. XILINX, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, 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. PCI, PCIe and PCI Express are trademarks of PCI-SIG and used under license

本資料は英語版 (v14.3) を翻訳したもので、内容に相違が生じる場合には原文を優先します。

資料によっては英語版の更新に対応していないものがあります。

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

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

改訂履歴

次の表は、このマニュアルの改訂履歴を表示しています。

日付	バージョン	改訂内容
2011/10/19	13.3	リリースに合わせて変更 <ul style="list-style-type: none">追加事項：<ul style="list-style-type: none">改訂履歴付録「その他のリソース」イーサネットの決定チュートリアルのリファレンス更新事項：<ul style="list-style-type: none">「シミュレーション結果の保存」で WCFG と WDB の説明を向上文書全体を向上fuse コマンドからサポートされるボードを削除。追加事項 : サポートされるボードのリストを含む「サポートされるボード」ISim GUI の説明の向上 : グラフィック ユーザー インターフェイスの概要
2011/11/30	13.3	<ul style="list-style-type: none">「サポートされるボード」にプロパティ、選択オプション、および新しい Type 値を追加サポートされる OS 情報へのリファレンスを削除
2011/01/19	13.4	<ul style="list-style-type: none">日付およびバージョン番号のみを変更

日付	バージョン	改訂内容
2012/05/08	14.1	<ul style="list-style-type: none"> • 3 ページの「機能サポート」を更新 • 「シミュレーション」の章を第 3 章「コンパイルおよびシミュレーション」に統合 <ul style="list-style-type: none"> • 52 ページの「fuse、vhpcmp、vlogcomp コマンド オプション」を 1 つの表にまとめ、コマンドへリンク • 第 3 章の「<code>uselib</code> (Verilog 指示子)」を追加 • 第 3 章「コンパイルおよびシミュレーション」および 第 8 章「ハードウェア協調シミュレーションの使用」の fuse コマンド オプションに <code>-hil_zynq_ps</code> ハードウェア協調シミュレーション コマンドを追加 • 「VHDL 言語サポートの例外」および 付録 B の「Verilog 言語サポートの例外」を 2 つの例外の表に統合
2012/07/25	14.2	<ul style="list-style-type: none"> • 第 3 章に含まれる 1 文で <code>uselib</code> 指示子の説明をわかりやすく変更 • 若干の文法的変更 • 付録 D「その他のリソース」に含まれるリストを更新
2012/10/16	14.3	<ul style="list-style-type: none"> • URL を 14.3 用にアップデート

目次

改訂履歴.....	2
第 1 章 : ISim の概要	
シミュレーション ライブラリ.....	3
言語サポート	3
機能サポート	3
OS サポート	4
ISim の操作モード.....	4
シミュレーション手順の概要	4
ISim チュートリアル.....	8
第 2 章 : ISim GUI の使用	
ISim GUI の概要	9
ISim プリファレンスの設定	33
第 3 章 : コンパイルおよびシミュレーション	
デザイン ファイルの解析	36
プロジェクト ファイルの構文.....	43
Verilog シミュレーション用の定義済み XILINX_ISIM マクロ	43
デザインのシミュレーション	44
混合言語シミュレーション	45
タイミング シミュレーション (ゲートレベルシミュレーション).....	49
ISim の EXE コマンド	49
シミュレーションの一時停止	56
シミュレーション結果の保存	57
シミュレーションの終了.....	58
第 4 章 : 波形の解析	
波形コンフィギュレーションでの作業.....	59
波形コンフィギュレーションのカスタマイズ	61
波形コンフィギュレーションのナビゲーション.....	67
波形コンフィギュレーションの印刷.....	71
カスタム カラーの使用	72
第 5 章 : シミュレーション結果の表示	
波形データベース ファイルと波形コンフィギュレーション ファイル	73
スタティック シミュレーションを開く	74
第 6 章 : ソース レベルでのデバッグ	
シミュレーションの 1 行ずつの実行.....	77
ブレークポイントの使用.....	78

第 7 章：消費電力のアクティビティ データの書き出し

第 8 章：ハードウェア協調シミュレーションの使用

要件	83
モデルの使用	83
制限事項	84
コンパイルの使用法	85
fuse コマンド ライン フロー	85
ツール フロー	86
ハイブリッド協調シミュレーション フロー	88
ハードウェア ボード 使用方法	90
ハードウェア協調シミュレーション	90
ISim ハードウェア協調シミュレーションの Tcl コマンド	91
サポートされるボード	92
よく寄せられる質問 (FAQ)	95

第 9 章：ISim Tcl コマンド

シミュレーション コマンドの別名表記	102
ISim 波形ビューアー Tcl コマンド	103
Command Line Conventions	103
Tcl コマンド	104

付録 A：ライブラリ マップ ファイル (xilinxim.ini)

付録 B：VHDL および Verilog 言語サポートの例外

VHDL 言語サポートの例外	141
Verilog 言語サポートの例外	143

付録 C：ModelSim XE から ISim への移行

ModelSim XE の概要	147
ISim の概要	148
機能比較	148
シミュレーション プロセス	149

付録 D：その他のリソース

ザイリンクス リソース	157
ISim チュートリアル	158

第 1 章

ISim の概要

ザイリンクス ISim は、VHDL、Verilog、および VHDL/Verilog 混合言語デザインでビヘイビア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する HDL (ハードウェア記述言語) シミュレータです。

この文書では、ISim ツールの機能、ISim でサポートされる HDL 言語、インターフェイスの使用方法について説明します。ご使用の PDF リーダーで [View] → [Show/Hide] → [Toolbar Items] → [Page Navigation] で [Previous View] および [Next View] をオンにしておくと、リンクされた情報の前後へ移動がしやすくなります (これは Acrobat Reader X を使用した場合のもので、メニュー表示は PDF リーダーによって異なります)。

シミュレーション ライブラリ

ザイリンクスのシミュレーション デバイス ライブラリはあらかじめコンパイルされており、アップデートのインストール時に自動的に更新されます。

注記: ISim で使用するライブラリは、Simulation Library Compilation Wizard (Compplib) でコンパイルしないでください。

言語サポート

ISim では、次の言語がサポートされています。

- VHDL IEEE-STD-1076-1993
- Verilog IEEE-STD-1364-2001
- Standard Delay Format (SDF) バージョン 2.1
- VITAL-2000

機能サポート

ISim では、次の機能がサポートされています。

- インクリメンタル コンパイル
- ソース コードのデバッグ
- SDF のアノテーション
- VCD の生成
- SAIF を使用した消費電力解析および最適化
- ハード IP ブロック (MGT、PPC、PCIe® など) のサポート
- マルチスレッド コンパイル
- ハードウェア協調シミュレーション (HwCoSim)
- 混合 VHDL/Verilog
- メモリ エLEMENT の表示およびデバッグ用の Memory Editor

- 1 クリックでシミュレーションの再コンパイルおよび再起動
- 1 クリックで簡単に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

OS サポート

サポートされる OS の情報は、[『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』\(UG798\)](#) を参照してください。

最新のリリースの変更点については、この文書へのリンクは、[『ザイリンクス デザイン ツール：リリース ノート ガイド』\(UG631\)](#) を参照してください。これらの資料へのリンクは、[付録 D「その他のリソース」](#) にも記載されています。

ISim の操作モード

ISim には次の 2 つの操作モードがあります。

- **グラフィカル ユーザー インターフェイス モード (GUI)**
シミュレーション データをグラフィカル表示できます。シミュレーションの実行およびデータの検証、デバッグには、メニュー コマンド、文脈依存コマンド、およびツールバー ボタンを使用します。グラフィカル ユーザー インターフェイスでの作業の詳細は、[第 2 章「ISim GUI の使用」](#) を参照してください。
- **コマンド ライン モード**
GUI との対話はありません。コマンドは、コマンドプロンプトから実行されます。シミュレーション実行ファイルが実行されると、データを検証、デバッグするシミュレーション Tcl コマンドを入力する Tcl プロンプトが開きます。

シミュレーション実行ファイルに `-tclbatch <file_name>` オプションを指定すると、シミュレーションが開始された後に Tcl コマンドのセットを実行できます。完了後にシミュレーションを停止するには、最後の Tcl コマンドで停止するように指定しておく必要があります。詳細は、[第 3 章「コンパイルおよびシミュレーション」](#) を参照してください。

シミュレーション手順の概要

ISim でデザインをシミュレーションする手順は、次のとおりです。

- [手順 1：ファイルの準備とライブラリのマップ](#)
- [手順 2：デザインの解析とエラボレーション](#)
- [手順 3：デザインのシミュレーション](#)
- [手順 4：デザインの検証](#)
- [手順 5：デザインのデバッグ](#)

手順 1：ファイルの準備とライブラリのマップ

ISim でシミュレーションを実行するには、次のファイルが必要です。

- ステイミュラス ファイルを含むデザイン ファイル
- ユーザー ライブラリ
- その他データ ファイル

スティミュラス ファイル

HDL ベースのテストベンチをスティミュラス ファイルとして含めます。次のいずれかの方法を使用してテストベンチを作成、編集します。

- テキスト エディター：
任意のテキスト エディターで HDL テストベンチを作成、編集します。
- 言語テンプレート：
ISE ツールに含まれているテンプレートを使用してファイルに内容を含めます。詳細は、[ISE ヘルプ](#)の「言語テンプレートの使用」を参照してください。
- サードパーティ ツール：
任意のベンダー ツールで HDL テストベンチを作成、編集します。

ユーザー ライブラリ

ISim を起動するとき使用するユーザーモードに従って、次の2 つのユーザーライブラリ追加方法があります。

- Project Navigator から起動する場合は、ISE ツールでユーザーライブラリを定義します。詳細は、[ISE ヘルプ](#)の「VHDL ライブラリの使用」を参照してください。
- ISim をスタンドアロンで使用する場合は、対話型コマンドモードと非対話型モードの両方でライブラリ マップ ファイルを設定します (ユーザーの論理/物理ライブラリを指定する方法は[付録 A 「ライブラリ マップ ファイル \(xilinxsim.ini\)」](#)を参照してください)。
- PlanAhead™ ツールからISim を起動するときは、ツールでユーザーライブラリを定義します。詳細は、[付録 D 「その他のリソース」](#)にリンクが記載されている『PlanAhead ユーザー ガイド』(UG632) を参照してください。

手順 2 : デザインの解析とエラボレーション

シミュレーションの実行前に、ISim ではコードを 1 つまたは複数のライブラリに解析して、デザインが依存するデザイン コンポーネントをエラボレーションする必要があります。この手順では、シミュレーション実行ファイルが生成されます。

グラフィカル ユーザー インターフェイス モード

ISE またはPlanAhead からISim を起動すると、ISim のグラフィカル ユーザー インターフェイス (GUI) が起動し、デザインが解析され、デザインコンポーネントがエラボレーションされます。詳細は、[手順 3 : デザインのシミュレーション](#)の「ISE からのシミュレーション」または『PlanAhead ユーザーガイド』(UG632) を参照してください。デザインが解析され、次のセクションで示すようにコマンド ラインから手動でデザインがエラボレーションされます。生成したシミュレーション実行ファイルを -gui オプションを使用して実行し、グラフィカル ユーザー インターフェイスを起動します。

非対話型コマンド ライン モード

非対話型コマンド ライン モードには、次の 2 つの手順があります。

1. プロジェクトファイルを作成します。詳細は、[43 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。
2. fuse コマンドを使用します。詳細は、[38 ページの「fuse の実行」](#)を参照してください。

手順 3 : デザインのシミュレーション

デザインのコンパイルおよびエラボレーションが完了したら、シミュレーション実行ファイルを実行して、デザインをシミュレーションします。ISim を読み取り専用モードで実行する方法の詳細は、[第 5 章の「スタティック シミュレーションを開く」](#)を参照してください。

コマンド ラインからの GUI モード シミュレーション

シミュレーション実行ファイル (x.exe (デフォルト) またはユーザー指定の名前) を生成すると、コマンド ラインで `-gui` オプションを使用し、`my_sim.exe -gui` のように指定してシミュレーション実行ファイルを実行できます。このコマンドを実行すると、GUI が起動します。このシミュレーション実行ファイル コマンドではシミュレーションは開始されません。シミュレーションを開始するには[44 ページの「デザインのシミュレーション」](#)で説明されている `[Run]` シミュレーション コマンドのいずれかを使用します。

この後、波形コンフィギュレーションに信号を追加できます。詳細は、[59 ページの「波形コンフィギュレーションでの作業」](#)を参照してください。

また、オプションでシミュレーション実行ファイルを実行して GUI を起動し、`-tclbatch` オプションを使用して Tcl ファイルでシミュレーションを実行することもできます。次は、その例です。

```
my_sim.exe -gui -tclbatch my_sim.tcl
```

`my_sim.tcl` ファイルの最上位にすべての信号を追加する `wave add` コマンドを使用すると、GUI の起動時に自動的に信号をトレースして表示することができます。

ISE からのシミュレーション

解析、エラボレーション、およびシミュレーション実行コマンドのすべては、ISE または PlanAhead で次のいずれかのプロセスを実行するとバックグラウンドで実行されます。

- `[Simulate Behavioral Model]`
- `[Simulate Post-Place & Route Model]`

これらの プロセスでは GUI が起動して、デフォルトで最上位の信号がトレースされます。

オプションで、カスタム Tcl ファイルを指定して、GUI の起動時にトレースする信号を制御することもできます。

シミュレーションは ISE のシミュレーション プロセス プロパティの `[Simulation Run Time]` で指定された期間実行されます。詳細は、[ISE ヘルプ](#)の「シミュレーション プロパティ」を参照してください。

シミュレーションの時間を追加するには、[44 ページの「デザインのシミュレーション」](#)で説明されている `[Run]` シミュレーション コマンドのいずれかを使用します。

非対話型コマンド ライン モード

`my_sim.exe` などのシミュレーション実行ファイルを実行します。Tcl プロンプトで `run` コマンドを入力します。

また、`-tclbatch` オプションを使用してシミュレーション実行ファイル を Tcl ファイルと共に実行することもできます。たとえば、「`my_sim.exe -tclbatch my_sim.tcl`」と入力します。

この手順が正しく実行されたことを確認します。問題があった場合は、[エラー メッセージの確認のログ ファイルの確認](#)および[手順 5 : デザインのデバッグ](#)を参照してください。

手順 4：デザインの検証

デザインのシミュレーションが完了したら、デザイン仕様が満たされるようにデザインをデバッグします。

シミュレーション結果の検証は、次の方法で実行できます。

- 波形ウィンドウの信号の動作を確認します。
- [Console] パネルまたは Tcl プロンプトで結果を確認します。

デバッグ段階では、次の操作を実行できます。

- 結果を保存します。詳細は [57 ページの「シミュレーション結果の保存」](#) を参照してください。
- シミュレーション結果を読み取り専用スタティック シミュレーターで表示および検証します。詳細は、[第 5 章の「スタティック シミュレーションを開く」](#) を参照してください。

手順 5：デザインのデバッグ

問題が見つかった場合は、デバッグにより原因とその解決方法を見つける必要があります。ISim では、多くの方法でデザインをデバッグできます。デザインをデバッグするには、エラー メッセージとログ ファイルをまず確認する必要があります。

エラー メッセージの確認

まずエラー メッセージを確認してデザインにエラーが含まれていないかを確認します。エラー メッセージは、ISE の [Console] パネルおよびログ ファイル (次のセクションを参照) に表示されます。次の接頭辞で始まるエラー メッセージを確認してください。

- **HDL Compiler**
解析またはスタティック エラボレーション中にエラーが発生したことを示します。エラーがこのプロセス中に発生して正しく実行されなかった場合は、HDL コンパイラで問題が発生した可能性があります。「fuse -v 1」と入力して、問題の特定に役立つ情報を出力します。またエラー メッセージは、fuse.log ファイルおよび ISE ソフトウェアの [Console] タブ (ISE 統合モード) に表示されます。
- **Simulator**
実行コードの生成またはシミュレーション中にエラーが発生したことを示します。詳細は、[6 ページの「手順 3：デザインのシミュレーション」](#) を参照してください。メッセージに含まれているファイル名および行番号から問題を確認します。

ログ ファイルの確認

ログ ファイルを確認すると、デザイン エラーの原因を見つける際に役立ちます。次のログ ファイルを使用できます。

- **fuse.log**
解析およびエラボレーション プロセス中に fuse コマンドにより生成された出力を含むログファイル
- **isim.log**
シミュレーション プロセス中にシミュレーション実行ファイルにより生成された出力を含むログ ファイル。このファイルにはデザイン データが含まれていないため、ザイリックスのテクニカル サポートに問題を報告しても安全です。

- `isimcrash.log`
ツールで予期せぬエラーまたは状況が見つかるときに生成されるログ ファイルで、
`./isim/<simulation_executable>.sim` ディレクトリに生成されます。

このファイルをザイリンクスに提出してサポートを受けてください。このファイルにはデザインデータが含まれていないため、ザイリンクスのテクニカル サポートに問題を報告しても安全です。

Tcl シミュレーション コマンドの使用

一部のシミュレーション コマンドは、デバッグに役立ちます。これらのコマンドは、Tcl プロンプトまたは ISim インターフェイスの [Console] パネルで実行できます。

- `isim ptrace on`
- `isim ltrace on`
- `dump`
- `show`
- `isim force`
- `bp`
- `onerror`

デバッグ ストラテジについては、第 6 章「ソース レベルでのデバッグ」を参照してください。

詳細は、第 9 章「ISim Tcl コマンド」を参照してください。

ISim チュートリアル

詳細は、次のチュートリアルを参照してください。

- 『ISim チュートリアル』(UG682)
ISim を使用したデザインのシミュレーションおよびデバッグ方法が示されています。
- 『ISE ハードウェア協調シミュレーション チュートリアル：浮動小数点高速フーリエ変換のシミュレーションの高速化』(UG817)
ISim ハードウェア協調シミュレーションを使用して不動小数点高速フーリエ変換 (FFT) のシミュレーションを高速化する方法が示されています。

これらの文書へのリンクは、付録 D「その他のリソース」に含まれます。

第 2 章

ISim GUI の使用

ISim のグラフィック ユーザー インターフェイス (GUI) は、さまざまなパネルを含むメイン ウィンドウ、ワークスペース、ツールバー、およびステータス バーから構成されています。メイン ウィンドウでは、次を実行できます。

- シミュレーション可能なデザイン箇所を表示
- 波形コンフィギュレーションに信号を追加して表示
- コマンドを使用してシミュレーションを実行
- デザインを確認して必要に応じてデバッグを実行

ISim GUI の概要

ISim の GUI は、シミュレーション実行ファイルを ISE[®] ツール、コマンド ライン、または PlanAhead[™] ツールから実行すると起動します。

図 2-1 は、ISim の GUI を示しています。

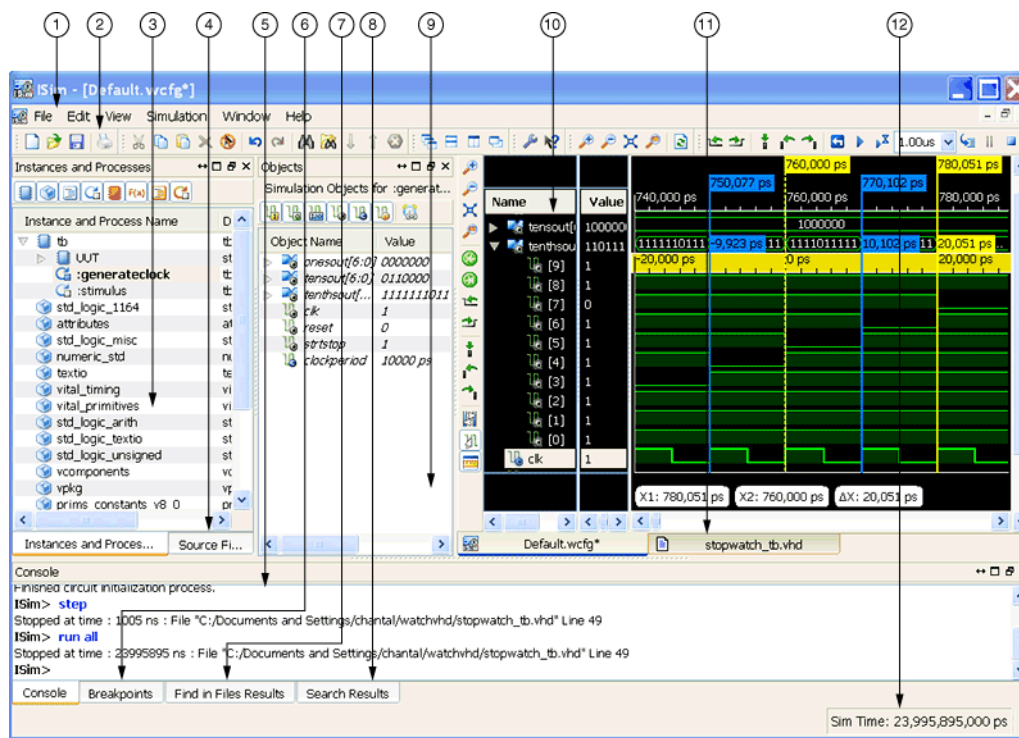


図 2-1 : ISim の GUI

ISim を閉じるには、[File] → [Exit] をクリックします。閉じる前に波形コンフィギュレーションを保存することを尋ねるダイアログ ボックスが表示されます。

表 2-1 は、9 ページの図 2-1 に示す ISim の GUI コンポーネントをリストしています。詳細は、各リンクをクリックしてください。

注記：ご使用の PDF リーダーで [View] → [Toolbars] → [More Tools] で [Previous View] および [Next View] をオンにしておくと、リンクされた情報の前後へ移動がしやすくなります (メニュー表示は PDF リーダーによって異なります)。

表 2-1 : ISim の GUI コンポーネント

#	説明
1. 「メニューおよびツールバー：コマンドおよびショートカット」	ツールで使用可能な操作のほとんどにアクセスできます。一部の操作は、文脈依存メニューからのみアクセス可能です。
2. 「[ISim] ツールバー」	よく使用するコマンドにアクセスできます。
3. 「[Instances and Processes] パネル」	シミュレーションに関連するブロック (インスタンスおよびプロセス) の階層が表示されます。
4. 「[Source Files] パネル」	デザインに関連するファイルのリストが表示されます。
5. 「[Console] パネル」	シミュレータで生成されるメッセージが表示されます。シミュレーションの Tcl コマンドをプロンプトに入力できます。
6. 「[Breakpoints] パネル」	デザインに現在設定されているブレイクポイントすべてがリスト表示されます。
7. 「[Find in Files Results] パネル」	複数のファイルから検索文字列と一致した結果が表示されます。
8. 「[Search Results] パネル」	検索条件と一致する結果が表示されます。
9. 「[Objects] パネル」	[Instances and Processes] パネルで選択されているブロックと関連するシミュレーション オブジェクトが表示されます。
10. 「波形ウィンドウ」	信号およびバスのリストとその波形、仕切り、カーソル、またはマーカーなどの波形オブジェクトから構成される波形コンフィギュレーションが表示されます。波形ウィンドウでは、複数の波形コンフィギュレーションを表示できます。
11. 「テキスト エディター ウィンドウ」	読み取り専用のハードウェア記述言語 (HDL) ファイルを表示します。
12. ステータス バー	カーソルが配置されているメニュー コマンドまたはツールバー ボタンの簡単な説明とシミュレーション時間が表示されます。

次のセクションでは、ISim の GUI コンポーネントについてそれぞれ説明します。

メニューおよびツール バー：コマンドおよびショートカット

ISim のメイン ウィンドウには、よく使用されるメイン メニュー オプションが実行できるさまざまなツールバーが含まれます。

メイン メニューからは、メニュー カテゴリに該当するコマンド オプションがさらに実行できます。メイン ウィンドウのツールバー アイコンは、ユーザー インターフェイスの上部に配置されています。

[View] → [Toolbars] → [<toolbar_name>] をクリックすると、ツールバーの表示/非表示を切り替えることができます。

[File] ツールバー

標準ツールバーを使用すると、頻繁に使用する [File] メニュー コマンドに簡単にアクセスできます。



[File] メニューおよび標準ツールバーには、次のオプションがあります。

- **[New]**
[New] ダイアログ ボックスを開き、作成するファイルの種類を選択します。作成できるのは、テキスト、回路図、またはシンボル ファイルです。
- **[Open]**
[Open] ダイアログボックスを開き、ディレクトリを検索してファイルを選択します。ファイルは、適切なアプリケーションで開きます。
- **[Save]**
作業中のファイルを以前に保存したファイルに上書き保存します。以前にファイルを保存していない場合、[Save As] ダイアログ ボックスが開き、作業中のファイルを保存できます。
- **[Save All]**
保存の必要なファイルがすべて保存されます。
- **[Print]**
[印刷] ダイアログボックスを開き、作業中のファイルを印刷します。

[Edit] ツールバー

[Edit] ツールバーを使用すると、頻繁に使用する [Edit] メニュー コマンドに簡単にアクセスできます。



- 使用できるコマンドは、[Cut]、[Copy]、[Paste]、[Delete]、[Undo]、[Redo]、[Find]、[Find in File] です。

[View] ツールバー

[View] ツールバーを使用すると、頻繁に使用する [View] メニュー コマンドに簡単にアクセスできます。



[View] ツールバーのボタンは、次のとおりです。

- [Zoom In]、[Zoom Out]、[Set View for all content to be visible]、[Zoom to Cursors] などがあります。

- [Refresh] ボタンをクリックすると、ファイルの表示が更新されます。



[View] メニューからは、次のようなオプションも実行できます。

- **[Panel]**
[Search Results]、[Find in Files Results]、[Breakpoints]、[Compilation Log]、[Source Files]、[Memory]、[Objects]、[Instances and Processes]、[Console] などのチェック ボックス オプションを含むダイアログ ボックスが開きます。
- **[Toolbars]**
ツールバーの表示/非表示を切り替えます。
- ステータスバーの表示は [Status Bar] チェック ボックスで切り替えることができます。

[ISim] ツールバー

[ISim] ツールバーを使用すると、頻繁に使用する ISim コマンドに簡単にアクセスできます。

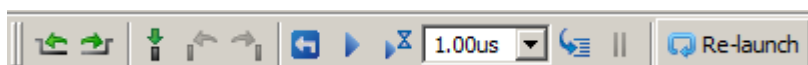


表 2-2 は、[ISim] ツールバーのボタンについて説明しています。

表 2-2 : [ISim] ツールバーのボタン



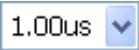



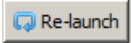
ボタン	説明
	カーソルの現在位置の左側にある一番近いマーカーに移動します。
	カーソルの現在位置の右側にある一番近いマーカーに移動します。
	波形のカーソルの位置にマーカーを追加します。
	シミュレーション時間を 0 にリセットします。
	イベントがすべて終了するか、[Stop] コマンドが実行されるか、またはブレークポイントに達するまでシミュレーションを実行します。
	指定した時間だけシミュレーションを実行します。
	シミュレーションが実行される時間を指定します。
	HDL ソース コードのシミュレーションを 1 行ずつ実行します。
	実行中のシミュレーションを停止します。Run コマンドのいずれかを使用するとシミュレーションを再開できます。

表 2-2 : [ISim] ツールバーのボタン (Cont'd)

ボタン	説明
	現在のシミュレーションを終了します。シミュレーション データは開いたままになります。
	シミュレーションをやり直します。

[Windows] ツールバー

[Window] ツールバーを使用すると、頻繁に使用する [Window] メニュー コマンドに簡単にアクセスできます。



[Windows] ツールバーのボタンでは、ウィンドウを重ねたり、縦や横に並べたり、手前に表示させたりできます。

[Help] ツールバー

[Help] ツールバーを使用すると、頻繁に使用する [Help] メニュー コマンドに簡単にアクセスできます。[Support and Services] では、ザイリンクスのサポート ページがデフォルトのウェブ ブラウザーで表示されます。

[What's This?] ヒントが表示されるようになります。ヘルプを必要とするメニューやツールバー ボタンの上にカーソルを置いてクリックすると、ヘルプが表示されます。

ショートカット キー

表 2-3 は、ISim のキーボード ショートカット キーをリストしています。

表 2-3 : ショートカット キー

ショートカット	メニュー コマンド
F1	[Help] → [Help Topics]
F3	[Edit] → [Find Next]
F5	[View] → [Run All]
F6	[View] → [Zoom Full View]
F7	[View] → [Zoom Out]
F8	[View] → [Zoom In]
F11	[Simulation] → [Step]
Delete	[Edit] → [Delete]
Ctrl + N	[File] → [New]
Ctrl + O	[File] → [Open]
Ctrl + S	[File] → [Save]
Ctrl + P	[File] → [Print]

表 2-3 : ショートカット キー (Cont'd)

ショートカット	メニュー コマンド
Ctrl + Z	[Edit] → [Undo]
Ctrl + Y	[Edit] → [Redo]
Ctrl + X	[Edit] → [Cut]
Ctrl + C	[Edit] → [Copy]
Ctrl + V	[Edit] → [Paste]
Ctrl + F	[Edit] → [Find]
Ctrl + G	[Edit] → [Go To]
Ctrl + A	[Edit] → [Select All]
Ctrl + W	[Add To Wave Configuration]
Ctrl + F4	[Window] → [Close]
Ctrl + Tab	[Window] → [Next]
Ctrl + Shift + Tab	[Window] → [Previous]
Ctrl + Home	[Go To Time 0]
Ctrl + End	[Go To Latest Time]
Ctrl + Shift + F5	[Restart]
Ctrl + マウス ホイール	表示の拡大/縮小
Shift + マウス ホイール	左側/右側の拡大表示
マウス ホイール	上方向/下方向にスクロール
左方向の矢印	[Previous Transition]
右方向の矢印	[Next Transition]
Pause	[Break]

[Instances and Processes] パネル

[Instances and Processes] パネルには波形ウィンドウの波形コンフィギュレーションと関連するブロック (インスタンスおよびプロセス) の階層が表示されます。インスタンスエートされてエラボレートされたエンティティおよびモジュールがツリー構造で表示されます。

このパネルには、次の列が含まれます。

- [Instance and Process Name]**
 インスタンス、プロセス、およびスタティック タスク/関数のボタンがデザインブロック階層を示すツリー構造
- [Design Unit]**
 1 列目のインスタンス、スタティック タスク/関数、またはプロセスに対応するデザイン ユニットの名前 (Verilog モジュールまたは VHDL エンティティ (アーキテクチャ))
- [Block Type]**
 インスタンス、スタティック タスク、関数、またはプロセス (例 : Verilog モジュール) の種類

[Instances and Processes] パネルには、次のタブが含まれます。

- **[Instance]**
インスタンス、プロセス、およびスタティック タスク、または関数がデザインのブロック階層を示すツリー構造で表示されます。
- **[Memory]**
デザイン オブジェクトのメモリが表示されます。詳細は、[27 ページの「メモリ エディターの使用」](#)を参照してください。
- **[Source Files]**
デザインのソース ファイルがリストされます。

15 ページの図 2-2 は、[Instances and Processes] パネルを表示したものです。

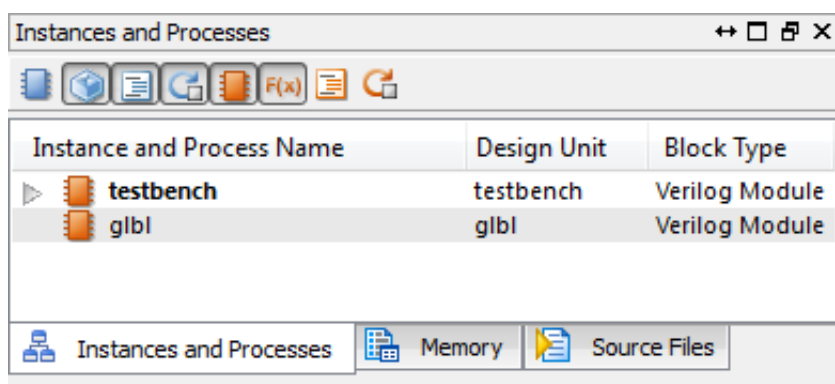


図 2-2: [Instances and Processes] パネル

デザイン階層ボタン

表 2-4 は、[Instances and Processes] パネルのデザイン階層ボタンを示しています。

表 2-4 : [Instances and Processes] パネルのボタン

アイコン	動作
	VHDL エンティティ
	VHDL パッケージ
	VHDL ブロック
	VHDL プロセス
	Verilog モジュール

表 2-4 : [Instances and Processes] パネルのボタン (Cont'd)

アイコン	動作
	Verilog タスクのフィルターの切り替え
	Verilog ブロックのフィルターの切り替え
	Verilog プロセスのフィルターの切り替え

階層ボタンを使用すると、次が実行できます。

- 階層を展開してコンポーネントを表示するには、矢印をクリックするか、または右クリックして [Expand] コマンドをクリックします (詳細は16 ページの「階層の展開/非展開」を参照)。
- [Design Unit] など、列タイトルをクリックすると、その列のデータを下に情報を並び替えることができます。
- パネルを非表示または復元するには、[View] → [Panel] → [Instances and Processes] をクリックします。

階層の展開/非展開

ネストされたグループのオブジェクトを含むウィンドウまたはパネルでは、次のいずれかの方法でその階層を展開または非展開できます。

- 三角矢印のクリック :
 - 矢印をクリックして階層を展開します。階層は 1 度に 1 つ展開できます。
 - 矢印をクリックして階層を閉じます。
- メニュー コマンドの使用 :
 1. オブジェクトを選択します。
 2. [Edit] → [Wave Objects] をクリックし、次のいずれかをクリックします。
 - [Expand]
選択されている階層オブジェクトを展開します。階層は 1 度に 1 つ展開できます。
 - [Collapse]
選択したオブジェクトの階層を非展開します。
- 文脈依存メニューの使用 :
 1. オブジェクトを選択します。
 2. 右クリックして次のいずれかをクリックします。

メイン ウィンドウの整列

ウィンドウ、パネル、およびツールバーは、次のいずれかの手順に従ってインターフェイス内で移動させることができます。

- [Window] メニュー コマンドの使用
[Window] メニュー コマンドは、波形ウィンドウおよびテキスト エディター ウィンドウでのみ使用できます。
- ドラッグアンドドロップの使用
パネルやメイン ウィンドウのツールバーなど、インターフェイスのその他の部分では、ドラッグアンドドロップを使用してオブジェクトを移動できます。次の手順に従います。
 1. 移動させるパネルのヘッダーをクリックしてホールドします。
 2. パネルを新しい位置に移動します。
パネルが配置される場所はグレーボックスで示されます。
 3. マウスを放し新しい位置にパネルを配置します。

ウィンドウの非表示および復元





メイン ウィンドウのパーツの多くは非表示にして後で復元できます。

注記：ウィンドウをデフォルトの位置に復元するには、[View] → [Restore Default Layout] をクリックします。

標準の [最大化]、[最小化]、[閉じる] コマンドは、波形ウィンドウおよびテキスト エディター ウィンドウの右上のボタンから実行できます。

これらのコマンドを使用すると、パネルを非表示、復元、フロート、またはドッキングできます。表 2-5 は、ボタンおよびその説明をリストしています。

表 2-5：パネル制御ボタン

アイコン	説明
	[Toggle Slide Out] パネルを最小化表示します。ウィンドウの端にあるパネル名にマウスを移動させてから、もう一度ボタンをクリックするとパネルが復元されます。
	[Toggle Maximized] パネルを最大化表示します。もう一度クリックすると、パネル サイズが復元されます。
	[Toggle Floating] パネルをフロートさせます。もう 1 度クリックすると、元の場所に戻ります。
	[Close] パネルを閉じます。パネルを復元するには、[View] → [Panels] → [(パネル名)] をクリックします。

波形ウィンドウ

波形ウィンドウには、信号、バス、およびこれらの波形が表示されます。波形ウィンドウの各タブには、信号およびバスのリストとそのプロパティ、仕切り、カーソル、またはマーカーなどの波形オブジェクトから構成される波形コンフィギュレーションが表示されます。

GUI では波形コンフィギュレーションの信号およびバスがシミュレーション中にトレースされるため、波形コンフィギュレーションはシミュレーション結果を調べるときに使用されます。デザイン階層および信号の遷移は波形コンフィギュレーションの一部ではなく、別のデータベース (.wdb) ファイルに保存されます。

波形コンフィギュレーション ファイル (.wcfg)

波形コンフィギュレーションは、信号のリスト、色や基数などのプロパティ、仕切り、グループ、カーソルなどの波形オブジェクトから構成されています。波形コンフィギュレーションはカスタマイズが可能で、シミュレーションの実行中以外るときならいつでも信号や波形オブジェクトを追加または削除できます。

初期ファイル `Default.wcfg` は、ファイルを保存するまで保存されません。波形コンフィギュレーション ファイルには、信号のリスト、信号のプロパティおよび波形オブジェクトが保存されます。

複数の波形コンフィギュレーションを作成、シミュレーションし、これらの波形コンフィギュレーションを個別に保存できます。

波形コンフィギュレーションの保存方法の詳細は、[23 ページの「波形コンフィギュレーションの保存」](#)を参照してください。

`Default.wcfg` ファイルは、シミュレータを起動したときに作成されます。波形コンフィギュレーション ファイルを `.wcfg` ファイルとして保存するには、ファイル名を指定する必要があります。

- GUI モードで ISim を終了する場合は、[名前を付けて保存] ダイアログ ボックスが表示されるのでファイル名を入力します。
- バッチ モードでは、ISim を終了する前に **wcfg save** コマンドを使用して `Default.wcfg` の内容を保存する必要があります。

アクティブ ウィンドウ

シミュレータを起動すると、最初のアクティブ ウィンドウは `Default.wcfg` になります。ウィンドウ タブをクリックするか、または `wave add` コマンドを使用すると、アクティブなウィンドウを変更できます。

- [File] → [New] または [File] → [Open] をクリックすると、新しく開いた波形コンフィギュレーション ウィンドウにアクティブ ウィンドウを切り替えることができます。
- Tcl では、「`wcfg new`」または「`wcfg open`」コマンドを使用すると同様の操作を実行できます。

波形ウィンドウの信号およびバス ボタン

波形ウィンドウの信号およびバスは、次のデザイン オブジェクトのいずれかで表現され、各オブジェクトにはそれぞれアイコンがあります。

[表 2-6](#) は、ISim の信号のボタンをリストしています。[19 ページの表 2-7](#) は、バスの信号のボタンをリストしています。

表 2-6 : ISim 信号のボタン

アイコン	説明
	入力ポート
	出力ポート
	入出力、双方向ポート
	内部信号
	定数、パラメーター、およびジェネリック信号
	[Variable]
	リンケージ信号 (VHDL のみ)
	バッファー信号

表 2-7 は、ISim のバスのボタンをリストしています。

表 2-7 : ISim のバスのボタン

ボタン	説明
	入力バス
	出力バス
	入出力、双方向バス
	内部バス

表 2-7 : ISim のバスのボタン (Cont'd)

ボタン	説明
	定数、パラメーター、およびジェネリック バス
	変数バス
	リンケージ バス
	バッファー バス

波形コンフィギュレーションのオブジェクト

カーソル

波形コンフィギュレーションではメイン カーソルとセカンダリ カーソルを使用して時間を特定し (メイン カーソル)、時間を計測します (メイン カーソルとセカンダリ カーソルを同時に使用)。カーソルは、さまざまなナビゲート操作の焦点として機能します。

- **メイン カーソル**
メイン カーソルは波形と交差する線で、交差点の値が [Value] 列に表示されます。カーソルではシミュレーション実行中の現在のシミュレーション時間が示されます。シミュレーション時間はカーソルの上端に表示されます。詳細は、[第 4 章の「カーソル」](#)を参照してください。
- **セカンダリ カーソル**
セカンダリ カーソルは点線で、時間の範囲を識別するときにメイン カーソルと共に使用されます。時間の範囲を使用すると、その範囲を拡大表示したり印刷したりできます。

マーカー

特定の時間に印を付けて後で参照できるようにします。マーカーは、波形と交差する縦線で、波形と交差する点の信号値が表示されます。マーカーの時間はマーカー上部に表示されます。また、複数のマーカーを設定すると、カーソルを前後に移動させて値の変化を解析できます。詳細は、[第 4 章の「マーカー」](#)を参照してください。

マーカーの追加およびマーカーを使用した波形値の表示

中空円/中が塗りつぶされた円

カーソルまたはマーカーを配置または移動するときに [Snap to Transition] ボタンを使用すると、信号遷移上に厳密にカーソル/マーカーを配置できます。

- カーソルまたはマーカーを配置または移動すると、中空円が表示されます。
- 1 つの遷移上を移動するときは中が塗りつぶされた丸が表示されます。



仕切り

波形コンフィギュレーションに含まれる信号を分けるときに使用します。

グループ

波形コンフィギュレーションに含まれる信号およびバスを関連信号セットとしてフォルダーにまとめ、整理する方法です。グループにはグループ アイコンとグループ名が表示されます。



グループ自体には波形データが表示されず、展開したときにその内容を表示できます。詳細は、[第 4 章の「グループの追加」](#)を参照してください。

仮想バス



論理スカラーおよび配列をまとめるグループです。仮想バスには、アイコンおよび仮想バス名が表示されます。仮想バスには、バスの波形が表示されます。仮想バスはその下に昇順で表示される信号の波形で構成されており、1 次元配列にフラット化されます。詳細は、[第 4 章の「仮想バスの追加」](#)を参照してください。

波形ウィンドウのツールバー ボタン

[表 2-8](#) は、波形ウィンドウのツールバー ボタンについて説明しています。

表 2-8：波形ウィンドウのツールバー ボタン

ボタン	説明
	[Zoom Out] : 表示しているオブジェクトのサイズを縮小します。
	[Zoom In] : 表示しているオブジェクトのサイズを拡大します。
	[Zoom to Full View] : 作業ウィンドウでビュー全体を表示します。
	[Zoom to Cursors] : 2 つのカーソルが波形の左端と右端に表示されるようにします。セカンダリ カーソルがない場合は、ズーム レベルを変更しないままメインカーソルの位置が中心に来るように表示します。
	[Go To Time 0] : メイン カーソルを 0 時間に移動します。
	[Go To Latest Time] : メイン カーソルをシミュレーションの最後に移動します。
	[Go To Next Transition] : メイン カーソルを次の遷移に移動します。
	[Go To Previous Transition] : メイン カーソルを 1 つ前の遷移に移動します。
	[Add Marker] : 波形のカーソルの位置にマーカーを追加します。
	[Previous Marker] : カーソルの現在位置の左側にある一番近いマーカーに移動します。

表 2-8 : 波形ウィンドウのツールバー ボタン (Cont'd)

ボタン	説明
	[Previous Marker] : カーソルの現在位置の右側にある一番近いマーカーに移動します。
	[Swap Cursors] : メイン カーソルとセカンダリ カーソルが設定されているときにこの 2 つをスワップします。
	[Snap to Transition] : 遷移に近いエリアにカーソルを配置したときに、カーソルを遷移に移動します。このモードはオン、オフに切り替えることができます。
	[Floating Ruler] : フロート ルーラーは波形ウィンドウの任意の位置に移動可能で、表示/非表示を切り替えることができます。

波形コンフィギュレーションでの作業

作業中のセッションで任意の数の波形コンフィギュレーションを作成できます。波形コンフィギュレーションには、信号のリスト、そのプロパティ、および追加された波形オブジェクトが保存されます。

波形コンフィギュレーションを作成するには、次を実行します。

1. [File] → [New] をクリックします。
[New] ダイアログ ボックスが表示されます。
2. [Wave Configuration] をクリックします。
3. [OK] をクリックします。

名前の付いていない新しい波形コンフィギュレーションが開きます。この波形コンフィギュレーションは、信号を追加するまで空の状態です (詳細は、[「波形コンフィギュレーションへの信号の追加」](#)を参照)。

複数の複数コンフィギュレーションが開いている場合は、次のいずれかを実行します。

- 波形コンフィギュレーションのタブをクリックします。
- [Window] → [Next] または [Window] → [Previous] をクリックして、波形コンフィギュレーションを切り替えます。

波形コンフィギュレーションへの信号の追加

GUI のメニュー コマンドまたはドラッグアンドドロップ手法を使用するか、または [Console] パネルで Tcl (ツール コマンド言語) コマンドを入力すると、波形ウィンドウにデザインの信号を表示できます。

注記：波形コンフィギュレーションの作成や信号の追加などの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は一時的に変更されている状態です。詳細は、[「波形コンフィギュレーションと WCFG ファイル」](#)を参照してください。

GUI からの信号の追加

1. [Instances and Processes] パネルでデザイン階層を展開してアイテムを選択します。
選択したインスタンスまたはプロセスに対応するオブジェクトが [Objects] パネルに表示されます。

2. [Objects] パネルでオブジェクトを選択します。
3. 次のいずれかの方法を使用して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。
 - 右クリックして [Add to Wave Window] をクリックします。
 - [Objects] パネルからオブジェクトを波形ウィンドウの [Name] 列にドラッグアンドドロップします。
 - [Console] パネルで 「[wave add](#)」 コマンドを実行します。

Tcl を使用した信号の追加

- オプションですが [Instances and Processes] パネルおよび [Objects] パネルでデザイン階層をナビゲートするか、または [Console] パネルで 「[scope](#)」 コマンドを入力して、追加するオブジェクトを識別します。
- [Console] パネルで 「[wave add](#)」 コマンドを使用して個別のオブジェクトまたはオブジェクトグループを追加します。

波形コンフィギュレーションと WCFG ファイル

波形コンフィギュレーションと .wcfg ファイルは両方とも波形リストのカスタマイズを指しますが、これら 2 つには概念的な違いがあります。

- 波形コンフィギュレーションは、メモリに読み込んで作業するものです。
 - 波形コンフィギュレーションには名前を付けることもできますが、名前を付けないままにしておくこともできます。名前は、波形コンフィギュレーション ウィンドウ タブに表示されます。
 - GUI から Tcl コマンドを入力して波形コンフィギュレーションを .wcfg ファイルに保存するとき、.wcfg ファイルの名前はコマンドの引数で指定されます。
 - 波形コンフィギュレーションを .wcfg ファイルから読み込むと、波形コンフィギュレーションの名前は .wcfg ファイルの名前になります。
- .wcfg ファイルは波形コンフィギュレーションをディスクに保存した形態を指します。

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は保存できます。複数の波形コンフィギュレーションを開いている場合は、それぞれ名前を付けて保存できます。

波形コンフィギュレーションを保存するには、次の手順のいずれかを実行します

- [File] → [Save] をクリックします。
- Ctrl+S を押します。
- [Save] ボタンをクリックします。



注記：別の名前で保存する場合は [File] → [Save As] をクリックします。

オブジェクトの検索

[Search] コマンドを使用するとデザインに含まれるオブジェクトを検索できます。このコマンドは、[Instances and Processes] パネルおよび [Objects] パネルで使用できます。検索では、文字列およびオブジェクトの種類を指定できます。

オブジェクトを検索するには、次の手順に従います。

1. [Objects] パネルまたは [Instances and Processes] パネルにカーソルを置きます。

2. 右クリックして [Search] をクリックします。
3. [Search] ダイアログ ボックスで、文字列を入力します。検索文字には、アスタリスク (*) をワイルドカードとして使用できます。
4. 検索するオブジェクト タイプを選択します。必要の場合は、[Match case] をオンにします。
5. [OK] をクリックします。

検索条件と一致したオブジェクトが [Search Results] パネルに表示されます。

HDL ソース ファイルを開く

ISim では、HDL (ハードウェア記述言語) ソース ファイルを ISim Text Editor で開くことができます。

HDL ソース ファイルを表示するには、次を実行します。

1. [Instances and Processes] パネル、[Objects] パネル、または [Source Files] パネルでファイルを 1 つ選択します。
2. ファイルをダブルクリックするか、または右クリックして [Go To Source] をクリックします。オブジェクトに関連する HDL ソース ファイルがテキスト エディターで開きます。

[File] → [Open] をクリックしてファイルを開いた場合、ファイルは書き込みモードになります。

[ファイルを開く] ダイアログ ボックスで [ファイルの種類] を Verilog または VHDL に変更してからファイルを指定し、[開く] をクリックします。詳細は、[26 ページの「ソース ファイルの変更」](#)を参照してください。

[Source Files] パネル

[Source Files] パネルは、[Instances and Processes] のタブとして表示されます。このタブを選択すると、デザインに関連するファイルがリストされます。このファイルのリストは、fuse コマンドによりデザインの解析およびエラボレーション中に供給されます。この操作は、GUI ではバックグラウンドで実行されます。

ソース コード ファイルを開くには、次を実行します。

1. リストからファイルを選択します。
2. [Go To Source Code] ボタンをクリックします。

このコマンドは、右クリックして [Go To Source Code] をクリックするか、またはファイルをダブルクリックしても実行できます。



[Objects] パネル

[Objects] パネルでは、[Instances and Processes] パネルで選択したインスタンスおよびプロセスに関連するシミュレーション オブジェクト (ポート、信号、変数、定数、パラメーター、およびジェネリック) がすべて表示されます。

パネルの上部には [Instances and Processes] パネルで選択されているインスタンス/プロセスが表示され、そのオブジェクトおよび値は [Objects] パネルに表示されます。




次に、[Objects] パネルの表の各列について説明します。

- **[Object Name]**
シミュレーション オブジェクト名とそのタイプを示すシンボルが表示されます。

- **[Value]**
[Sync Time] ボタンに基づき現在のシミュレーション時間またはメイン カーソルの場所のシミュレーション オブジェクトの値を表示します。
- **[Data Type]**
シミュレーション オブジェクト、ロジック、またはアレイのデータ タイプを表示します。

切り替えボタンは [Objects] パネルから使用できます。詳細は、表 2-9 を参照してください。

表 2-9 : [Objects] パネルのツールバー ボタン

ボタン	説明
	入力ポートのオン/オフを切り替えます。
	出力ポートのオン/オフを切り替えます。
	入出力、双方向ポートのオン/オフを切り替えます。
	内部信号のオン/オフを切り替えます。
	定数、パラメーター、ジェネリックのオン/オフを切り替えます。
	変数のオン/オフを切り替えます。
	<p>[Sync Time] 機能をオン/オフを切り替えます。</p> <ul style="list-style-type: none"> • オンの場合 [Objects] パネルの値は波形ウィンドウのメインカーソルの位置に基づきます。 • オフのときはステータス バーに表示されている [Sim Time] の値 (シミュレーション終了時間) と同一になります。

[Show Drivers] コマンドの使用

[Show Driver] コマンドを使用すると、信号値またはオブジェクト値での変更に関連するドライバーを表示できます。このコマンドを使用して値の変化の原因を特定し、回路の接続が正しいかどうか判断します。ISim では、[Console] パネルに信号またはオブジェクトのドライバーが表示されます。

このコマンドは、次のエリアでオブジェクトをプローブするときに使用できます。

- [Objects] パネル
- 波形ウィンドウ
- [Console] パネル (「show driver」コマンド)

ドライバーを表示するには、次の手順に従います。

1. オブジェクトまたは信号を選択します。
2. [Edit] → [Wave Objects] → [Show Drivers] をクリックします。

[Console] パネルでは、オブジェクトまたは信号のドライバーが表示されます。ドライバーがない場合は、その旨を伝えるメッセージが表示されます。

注記：このコマンドは、[Console] パネルに「show driver」と入力しても実行できます。

エレメントの表示

[Objects] パネルでは、各構成オブジェクトで表示する子エレメントをあらかじめ設定されている最大表示数のみを表示するか、しないかを制御できます。この最大数は、[Preferences] ダイアログボックスで変更できます。

子エレメントをすべて表示するには、次の手順に従います。

1. [Objects] パネルのオブジェクト リスト内で右クリックします。
2. 右クリックして [Show All Elements] をクリックします。

オブジェクト階層に表示される子エレメントの数が表示されます。

子エレメントの表示数を制限するには、[Objects] パネルのオブジェクト リスト内で右クリックし、[Limit Elements] をクリックします。

子エレメントに対してあらかじめ設定されている最大表示数を変更するには、プリファレンス設定を次のように設定します。

1. [Edit] → [Preferences] をクリックします。
2. [Preferences] ダイアログ ボックスの左側ペインで [ISim Simulator] をクリックします。
3. [Limit the maximum number of elements displayed to] をオンにします。
4. [Apply] をクリックしてから [OK] をクリックします。

波形ウィンドウでのオブジェクトの選択

[Objects] パネルでオブジェクトの信号をハイライトするには、次の手順に従います。

1. [Objects] パネルでオブジェクトを選択します。
2. 右クリックして [Select in Wave Window] をクリックします。

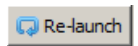
テキスト エディター ウィンドウ

テキスト エディター ウィンドウを使用すると、下位 HDL ソース ファイルを表示できます。

ソース ファイルの変更

ソース ファイルを変更するには、次の手順に従います。

1. ISim Text Editor でソース ファイルを開きます。
2. 変更の問題がないことを確認したら、[Re-Launch] を実行してデザインをシミュレーションし直します。



ISE プロジェクトにより、ソース ファイルの変更が自動的に保存されます。

ブレークポイントの設定

HDL ファイルの実行行にブレークポイントを設定し、第 6 章の「ブレークポイントの使用」に記述されているように、ブレークポイントが設定されているソース コード行に到達するまでコードを継続して実行できます。

注記：ブレークポイントを設定できるのは、実行コード行のみです。

メモリ エディターの使用

メモリ エディターでは、デザインのコンパイルおよびエラボレーションを再実行せずに、シミュレーション中にデザインの 2 次元メモリ アレイの内容を検索、変更できます。メモリ オブジェクトは、[Memory] タブ、[Objects] パネル、および [Search Results] タブの 3 箇所に表示されます。メモリ エディターを開くには、次のいずれかの手順に従います。

- デザインの 2 次元アレイのロジック タイプすべてを含む [Memory] タブで、メモリ エディターに表示するメモリをダブルクリックします。
- [Objects] パネルで 2 次元アレイのロジックタイプを右クリックして、[Memory Editor] をクリックします。
- [Instance and Processes] パネルでメモリ名を検索します。検索結果が [Search Results] パネルに表示されるので、該当するメモリを右クリックして [Memory Editor] をクリックします。

注記： 2 次元アレイ以外のロジックのオブジェクトでは、[Memory Editor] メニューは使用できず灰色表示されます。

メモリ エディターには、次のフィールドが表示されます。

- **[Address]**
表示されているメモリの特定のロケーションに移動します。
- **[Columns]**
各行に表示するエレメント数を選択します。[auto] では、2 のべき乗のエレメントが表示されます。
- **[Address Radix]**
メモリ エディターで表示されるアドレスの基数を選択します。
- **[Value Radix]**
メモリ エディターで表示される値の基数を選択します。

メモリ エディターは状態を保持したままフロートさせることができます。

メモリ エディター内は矢印キーを使用するとナビゲートでき、選択したアイテムの現在の位置が現在のアドレス基数に基づいてステータス バーに表示されます。

[Console] パネル

[Console] パネルでは、ISim で生成されるメッセージ ログを確認し、コマンド プロンプトで標準および ISim 特有の Tcl コマンドを入力できます。[Console] パネルには、次が表示されます。

メッセージ

エラー、警告、情報メッセージなど、ISim で生成されるすべてのメッセージが表示されます。また、ISim GUI のグラフィカル制御から実行されたシミュレータ コマンドもすべて表示されます。

シミュレーション コマンド

コマンド プロンプトでは、シミュレーション Tcl コマンドを入力したり、[Console] パネルに dump コマンドの出力を表示したりすることができます。詳細は、[第 3 章の「デザインのシミュレーション」](#)を参照してください。

[Console] パネルで右クリックすると、内容を管理できるメニューが表示されます。

[Breakpoints] パネル

ブレークポイントは、ソース コードに含まれるユーザー定義の停止ポイントで、ISim で使用してデザインをデバッグするときに使用します。[Breakpoints] パネルでは、デザインに現在設定されているブレークポイントがリスト表示されます。詳細は、第 6 章の「ブレークポイントの使用」を参照してください。

[Breakpoints] パネルには、ソース ファイルに設定されている各ブレークポイントに対して、ファイルの保存場所、ファイル名、および行数が表示されます。[Breakpoints] パネルのツールバーや文脈依存メニューを使用して、選択した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したり、ソース コードに移動できます。

ブレークポイントを設定するには、次のいずれかをクリックします。

- [View] → [Breakpoint] → [Toggle Breakpoint]
- [Toggle Breakpoint] ボタン
- HDL ファイルでコード行の行番号の右側をクリックします。
- Tcl コンソールに「**bp** <option>」と入力します。



コード行を右クリックして [Toggle Breakpoint] をクリックしても、同じ操作を実行できます。

ブレークポイントを挿入すると、シミュレーション ブレークポイント アイコン () がコード行の横に表示されます。






ブレークポイントのリストは、[Breakpoints] パネルに表示されます。実行行以外の行にブレークポイントを配置しても、追加されません。

ブレークポイントを削除するには、シミュレーション ブレークポイント アイコンをクリックします。

[Breakpoints] パネルのツールバー ボタン

表 2-10 は、ブレークポイント ボタンを示しています。

表 2-10 : ブレークポイント ボタン

ボタン	説明
	[Delete Breakpoint] : [Breakpoints] パネルで選択した行を削除し、HDL ソース ファイルからブレークポイントを削除します。
	[Delete all breakpoints] : HDL ソース ファイルからブレークポイントをすべて削除します。
	[Go To Source Code] : テキスト エディターで HDL ソース ファイルを開きブレークポイントを示します。






[Search Results] パネル

[Search Results] パネルには、[Search] コマンドの検索条件と一致した結果が表示されます。結果には、オブジェクトの種類のアイコンおよびオブジェクトの位置が表示されます。

[Search Results] パネルのツールバー ボタン

表 2-11 は、[Search Results] パネルのボタンを示しています。

表 2-11 : [Search Results] パネルのツールバー ボタン

ボタン	説明
	[Clear All] : [Search Results] パネルの内容を消去します。
	[Add To Wave Configuration] : 選択した検索結果に関連する信号を波形ウィンドウの波形コンフィギュレーションに追加します。
	[Go to Source Code] : テキスト エディタで HDL ソース ファイルを開き、検索対象のデザイン ユニットが定義されている行を表示します。
	[Go To Instantiation Source Code] : テキスト エディターで HDL ソース ファイルを開き、検索対象のデザイン ユニットがインスタンス化されている行を表示します。
	[Stop Searching] : 検索を終了します。

[Find in Files Results] パネル

複数のファイルから文字列を検索するには、次の手順に従います。

[Find in Files] を使用方法は、次のとおりです。

表 2-12 : [Find in File] パネルのボタンとその動作






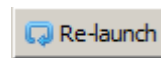
ボタン	動作
	[Edit] → [Find in Files] をクリックするか、ツールバーの [Find Text in Files] ボタン をクリックします。 [Find in Files] ダイアログ ボックスで検索するテキストを入力し、検索オプションを選択してから [Find] をクリックします。 [Find in Files Results] パネルで、次のいずれかの操作を実行します。
	パネルに表示されている結果をすべてクリアするには、[Clear All] ボタンをクリックします。
	結果に表示されているファイルをワークスペースに開くには、ファイルを選択し、[Show Current Result] ボタン をクリックします。 パネル上でファイルをダブルクリックしても、同じ操作を実行できます。
	次の結果を表示するには、[Show Next Result] ボタンをクリックします。
	前の結果を表示するには、[Show Previous Result] ボタンをクリックします。

表 2-12 : [Find in File] パネルのボタンとその動作 (Cont'd)

ボタン	動作
	現在実行中の検索を停止するには、[Stop Job] ボタンをクリックします。
	検索結果を CSV (Comma Separated Value) ファイルに保存するには、[Save Results as a Text File] ボタンをクリックします。

シミュレーションの再起動

[Re-launch] ボタン をクリックすると、検出されているハードウェア記述言語 (HDL) の問題を修正した後に ISim シミュレーションを再実行できます。また、GUI から再コンパイルも実行できます。



コンパイルおよびシミュレーションの再実行は完全に自動化されています。ダイアログ ボックスに表示されるメッセージには、問題の箇所が示されます。Project Navigator または PlanAhead ツールからフローを起動した場合は、[Re-launch] でコンパイル時に設定されたオプションがすべて保持した状態で指定時間だけシミュレーションが自動的に実行されます。

- シミュレーションが正しく再実行された場合、エラーが発生することなくシミュレーションが完了します。
- シミュレーションが正しく再実行されない場合、ソースコードをコンパイルできなかった原因となる構文エラーがダイアログボックスに表示されます。このメッセージに含まれるリンクをクリックすると、エラーを含むソースコードに移動できます。リンクで示されたエラーをリストされている順番に修正し、[Re-launch] ボタンをクリックして再コンパイルし、修正した問題を確認してください。

スティミュラスの適用

[Force Selected Signal] ダイアログ ボックスを使用すると、VHDL 信号、Verilog ワイヤ、または Verilog レジスタに強制的に定数値を割り当てることができます。このダイアログ ボックスは、信号を選択して右クリックし、[Force Constant] をクリックすると開きます。強制的に割り当てると、HDL コードで割り当てられた値や、以前に適用された定数または force コマンド値は、新しく割り当てられる値で上書きされます。[Apply] をクリックし、すべての変更を反映させます。図 2-3 は、[Force Selected Signal] ダイアログ ボックスを示しています。

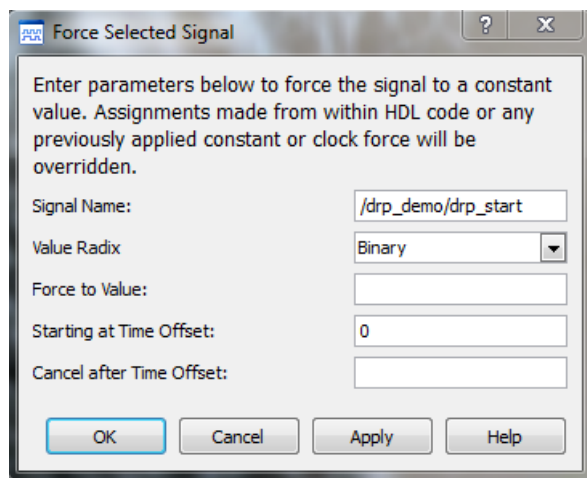


図 2-3: [Force Selected Signal] ダイアログ ボックス

[Force Selected Signal] ダイアログ ボックスのオプションは、次のとおりです。

- **[Signal Name]**
デフォルトの信号名が表示されます。デフォルトの信号名は、[Objects] パネルまたは波形で選択されている信号の完全パスで表示されます。この名前は変更可能です。不正な信号名が入力されると、ボックスが赤色表示されます。
- **[Value Radix]**
選択されている信号の現在の基数設定が表示されます。[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)、および [ASCII] から選択できます。
- **[Force to Value]**
[Value Radix] で定義された基数を使用し、強制的に適用する定数値を指定します。
- **[Starting at Time Offset]**
特定の時間後に force コマンドを開始するときの時間を指定します。デフォルトの開始時間は 0 です。時間は、10、10 ns などの文字列で指定できます。単位なしで数字が入力された場合、デフォルトのシミュレーション時間単位が使用されます。
- **[Cancel after Time Offset]**
特定の時間後に force コマンドをキャンセルするときの時間を指定します。時間は、10、10 ns などの文字列で指定できます。単位なしで数字が入力された場合、デフォルトのシミュレーション時間の単位が使用されます。

クロック スティミュラスの適用

[Force Clock] を右クリックして開く [Define Clock] ダイアログ ボックスを使用すると、VHDL 信号、Verilog ワイヤ、または Verilog レジスタに強制的に別のパターン (クロック) を割り当てることができます。HDL コードで割り当てられた値や、以前に割り当てられた定数または force コマンドの値は、新しく割り当てられるクロックパターンで上書きされます。[Apply] をクリックし、変更を反映させます。

[Define Clock] ダイアログ ボックス

[Force Clock] を右クリックすると、[Define Clock] ダイアログ ボックスが開きます。このダイアログ ボックスに含まれるオプションは、次のとおりです。

- **[Signal Name]**
デフォルトの信号名は、[Objects] パネルまたは波形で選択されている信号の完全パスで表示されます。この名前は変更可能です。不正な信号名が入力されると、ボックスが赤色表示されます。
注記: restart コマンドを実行すると、現在適用されている isim force コマンドがすべてキャンセルされます。
- **[Value Radix]**
選択されている信号の現在の基数設定が表示されます。[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)、および [ASCII] から選択できます。
- **[Leading Edge Value]**
クロック パターンの最初のエッジを指定できます。この値では、[Value Radix] で定義されている基数が使用されます。
- **[Trailing Edge Value]**
クロック パターンの 2 つ目のエッジを指定できます。この値では、[Value Radix] で定義されている基数が使用されます。

- **[Starting at Time Offset]**

現在のシミュレーション時間からある特定の時間後に **force** コマンドを実行するときの時間を指定します。デフォルトの開始時間は 0 です。時間は、10、10 ns などの文字列で指定できます。単位なしで数字が入力された場合、Tcl コマンド `「isim get userunit」` を実行したときに戻されるデフォルトのユーザー単位が使用されます。

- **[Cancel after Time Offset]**

現在のシミュレーション時間からある特定の時間後に **force** コマンドをキャンセルするときの時間を指定します。時間は、10、10 ns などの文字列で指定できます。単位なしで数字が入力された場合、デフォルトのシミュレーション時間単位が使用されます。

- **[Duty Cycle (%)]**

クロック パルスがアクティブな状態である時間の割合を指定します。許容値は 0 ～ 100 です。

- **[Period]**

クロック パルスの長さを時間で指定します。時間は、10、10 ns などの文字列で指定できます。

[Define Clock] ダイアログ ボックスの設定例

信号に永久的クロック (100MHz) を割り当てる場合は、次を設定します。

- [Leading Edge Value] : 1
- [Trailing Edge Value] : 0
- [Starting at Time Offset] : 0
- [Cancel after Time Offset] : 空白
- [Duty Cycle (%)] : 50
- [Period] : 10 ns

特定の時間信号にクロックを割り当てる場合 (100ns でトグルを開始し、1ms 後にトグルを停止) は、次を設定します。

- [Leading Edge Value] : 1
- [Trailing Edge Value] : 0
- [Starting at Time Offset] : 100 ns
- [Cancel after Time Offset] : 1ms
- [Duty Cycle (%)] : 50
- [Period] : クロック周期を指定

信号のトグル値を割り当てる場合 (1us 間 16 進数 F と 16 進数 A 間を 50ns ごとにトグル) する場合は、次を設定します。

- [Value Radix] : [Hexadecimal] (16 進数)
- [Leading Edge Value] : F
- [Trailing Edge Value] : A
- [Starting at Time Offset] : 0
- [Cancel after Time Offset] : 1us
- [Duty Cycle (%)] : 50
- [Period] : 50 ns

ISim プリファレンスの設定

ISim に関するプリファレンスを表示したり、変更できます。プリファレンスを設定するには、次の手順に従います。

1. [Edit] → [Preferences] をクリックします。
2. [Preferences] ダイアログ ボックスの左側ペインで任意のカテゴリをクリックします。
 - [「ISE Text Editor のプリファレンス」](#)
 - [「ISim シミュレータのプリファレンス」](#)
3. 設定に変更を加えます。
4. [Apply] をクリックし、[OK] をクリックします。

プリファレンス設定が保存され、ほとんどの設定がすぐに ISim に適用されます。

ISE Text Editor のプリファレンス

ISE Text Editor に関連するプリファレンス設定では、ISim で開いている ハードウェア記述言語 (HDL) ファイルの動作のみが制御されます。プリファレンス設定の詳細は、[ISE Text Editor ヘルプ](#) を参照してください。

ISim シミュレータのプリファレンス

[Preferences] ダイアログ ボックスの [Simulator] ページで設定できます。このページを表示するには、[Edit] → [Preferences] をクリックして左側ペインで[ISim Simulator] をクリックします。

[Draw Waveform Shadow]

波形ウィンドウで信号の影背景の表示/非表示を切り替えます。

[Limit the maximum number of elements displayed to]

[Object] ウィンドウに表示するオブジェクトの子エレメントの制限数を設定します。詳細は、[「エレメントの表示」](#) を参照してください。

[Default Radix]

波形コンフィギュレーション、[Objects] パネル、および [Console] パネルで表示されるデフォルトの基数値を設定します。詳細は、[「デフォルトの基数の変更」](#) を参照してください。

[Console text font]

ボタンの右側のボックスに、指定したフォントの例が表示されます。[Change] ボタン をクリックすると、[Console] パネルで使用するフォントを指定する [Select Font] ダイアログ ボックスが表示されます。

ISim カラー プリファレンス

[Preferences] ダイアログ ボックスの [Colors] ページでは、波形の表示色を設定できます。[Apply] をクリックし、変更を反映させます。カラー プリファレンスには、次のようなオプションがあります。

- **[Current Color Scheme]**
デフォルトの表示色名および作成した表示色名が表示されます。
- **[New]**
新しい表示色を作成する場合はクリックします。[Current Color Scheme] ボックスに新しい名前を入力し、スキーム表で色を変更します。

- **[Delete]**

選択した表示色を削除します。表示色は変更することもできます。

詳細は、[33 ページの「ISim プリファレンスの設定」](#)を参照してください。

時間フォーマットのプリファレンス

表示される時間の表示方法は、[Preferences] ダイアログボックスの [Time Format] ページでカスタマイズします。このページを表示するには、[Edit] → [Preferences] をクリックして左側ペインで [Time Format] をクリックします。時間フォーマットの設定は、[Waveform Window] および [Other GUI Elements] の2つのフィールドから設定できます。

[Waveform Window]

ここで設定した内容が、波形ビューアー ウィンドウ内の GUI に適用されます。時間フォーマットには、次のようなオプションがあります。

- **[Link All Waveform Time Units To Ruler]**

[Ruler] への変更に合わせて [Cursors/Markers] および [Measure Bubble] の単位が変更されます。このオプションはデフォルトでオンになっています。

- **[Rulers]**

波形ウィンドウ上部のメイン ルーラーおよびフロート ルーラーに適用されます。

- **[Cursors/Markers]**

すべてのカーソルおよびマーカーに表示される時間の値に適用されます。

- **[Measure Bubbles]**

波形ウィンドウ下部に表示されるカーソル値を表示するバブルに適用されます。

[Other GUI Elements]

ここで設定した内容が、波形ビューアー ウィンドウ外の GUI に適用されます。

- **[All Time Values]**

メイン ウィンドウの右下に表示される現在のシミュレーション時間および [Objects] パネルに表示される時間の値に適用されます。上記の時間フォーマットでは、次のフィールドを使用して表示する値の時間の単位と精度を設定できます。

- **[Units]**

時間の値の単位を選択するボックスです。[Other GUI Elements] のデフォルト設定は [Default] で、[Waveform Window] のデフォルト設定は [Auto] です。

- **[Decimal Places]**

時間の値を表示するときに使用する小数点以下の桁数を設定します。デフォルト設定は、どちらのフィールドでも Maximum になっています。

- **[Reset To Defaults]**

値をデフォルト設定に戻します。

コンパイルおよびシミュレーション

シミュレーションは、VHDL、Verilog または混合言語コンポーネントに対して実行できます。

- 論理シミュレーションは、デザイン プロセスの早期段階で実行できます。
- タイミング シミュレーションは、配置配線 (PAR) 後に実行する必要があります。

また、これらの 2 タイプのシミュレーションは、次のオプションで実行できます。

- Tcl (ツール コマンド言語) バッチ ファイルまたは Tcl コマンドを使用してコマンド ラインから実行
- ISE[®] Design Suite または PlanAhead[™] ツールから起動可能な GUI モードから実行

論理シミュレーションまたはタイミング シミュレーションのいずれかをコマンド ラインから実行する場合は、次の手順に従ってください。

1. デザイン ファイルの解析
2. シミュレーション実行ファイルの生成
3. デザインのシミュレーション

タイミング シミュレーションの場合は、さらに手順が必要となります。詳細は、[49 ページの「タイミング シミュレーション \(ゲートレベル シミュレーション\)」](#)を参照してください。

次のセクションでは、上記の手順について説明します。

デザイン ファイルの解析

VHDL ファイルの解析には `vhpcomp`、Verilog ファイルの解析には `vlogcomp` コマンドを使用します。

vlogcomp コマンド

`vlogcomp` コマンドは Verilog ソース ファイルを解析し、2 進数表記の Verilog ファイルを生成します。`vlogcomp` で生成された 2 進数表記は `fuse` コマンドで使用され、シミュレーション実行ファイルが作成されます。

プロジェクト ファイルか、コンパイルする Verilog ファイルを指定する必要があります。いずれも指定されていない場合は、`vlogcomp` を実行するとエラーが発生します。プロジェクト ファイルの詳細は、43 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。

vlogcomp コマンドの構文

コマンドの説明については、それぞれのリンクをクリックしてください。

注記：ご使用の PDF リーダーで [View] → [Toolbars] → [More Tools] で [Previous View] および [Next View] をオンにしておくと、リンクされた情報の前後へ移動がしやすくなります (メニュー表示は PDF リーダーによって異なります)。

```
vlogcomp
[-d <macro_definition>=<value>=<value>]
[-f <cmd_file>]
[-h]
[-i "<include_path>"]
[-intstyle [ise | xflow | silent | default]]
[-initfile <sim_init_file>]
[[-L|-lib <search_lib>[=<lib_path>]]]
[-prj <prj_file>.prj]
[-sourcelibdir <directory_name>]
[-sourcelibext <file_extension>]
[-sourcelibfile <file_name>]
[-v [-verbose] <value>]
[-version]
[<verilog_files>...]
[-work [<work_library>[=<library_path>] ] <filenames>...]
```

オプションの説明については、[fuse](#)、[vhpcomp](#)、[vlogcomp](#) コマンド オプションセクションを参照してください。

vlogcomp コマンド例

Verilog ファイルを 2 つ使用し、ソース ライブラリ ディレクトリを指定し、Verilog ファイルの拡張子を指定します。

```
vlogcomp tb.v fft.v -sourcelibdir ./mylib -sourcelibext .v
```

Verilog ファイルとソース ライブラリ ファイルを指定します。

```
vlogcomp dff.v -sourcelibfile ./mylib/dff_lowest.v
```

work ディレクトリと Verilog ファイルを指定します。

```
vlogcomp -work my_lib tb.v
```

2 つのファイルを使用します。


```
vlogcomp top_testbench.v top_timesim.v
```

mydir/cells ディレクトリに含まれる未解決のセルが検索されます。

```
vlogcomp -work mywork1 file1.v -sourcelibdir mydir/cells
```

たとえば、file1.v で未解決の DFF および DMUX がインスタンス化される場合、このコマンドにより mydir/cells ディレクトリに含まれる DFF および DMUX という名前のファイルが検索されます。モジュール DFF および DMUX は、ファイル DFF および DMUX で定義されます。

ソース ライブラリのサポート

次に示すコマンドの引数では、Verilog-XL 規格と同等のソース ライブラリがサポートされます。

次のコマンド オプションを vlogcomp コマンドに付けて実行してください。

```
vlogcomp -sourcelibdir <library_location>
```

注記：-sourcelibdir では Verilog-XL の -y オプションに類似した機能が提供されます。

コマンド ラインのソース ファイルが解析された後にモジュールに未解決のリファレンスがある場合、このコマンドでソース ライブラリが検索されます。

この検索中に、vlogcomp は未解決のインスタンス化されたデザイン ユニットと指定された -sourcelibdir ディレクトリに含まれている同じ名前のファイル名を一致させようとします。

ファイルが存在する場合は、vlogcomp でそのファイルが解析されます。

vlogcomp は -sourcelibdir オプションを処理中に、-sourcelibext が指定されない場合、.v、.h などの拡張子のファイルを無視します。

注記：-sourcelibext では Verilog-XL の +libext+ オプションに類似した機能が提供されます。このコマンド ラインの引数は、ソース ライブラリ ファイルに拡張子があるとき、-sourcelibdir と共に使用できます。

例：

```
vlogcomp -sourcelibdir /project/mysources tb.v fft.v  
vlogcomp -sourcelibdir /project/mysources tb.v fft.v -sourcelibext .v
```

すべての未解決のモジュールの定義を含むソース Verilog ライブラリ ファイルを指定することもできます。

例：

```
-sourcelibfile ./library/lib_abc.v
```

vhpcomp コマンド

vhpcomp コマンドは VHDL ソース ファイルを解析し、2 進数表記の HDL ファイルを保存します。vhpcomp で生成された 2 進数表記は fuse コマンドで使用され、シミュレーション実行ファイルが作成されます。

プロジェクト ファイルか VHDL ファイルを指定します。プロジェクト ファイルも VHDL ファイルも指定しない場合、エラー メッセージが表示されます。プロジェクト ファイルの詳細は、[43 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。

vhpcomp コマンドの構文

コマンド オプションの説明については、それぞれのリンクをクリックしてください。

注記：ご使用の PDF リーダーで [View] → [Toolbars] → [More Tools] で [Previous View] および [Next View] をオンにしておくと、リンクされた情報の前後へ移動がしやすくなります (メニュー表示は PDF リーダーによって異なります)。

次に、vhpcomp コマンドの構文を示します。

```
vhpcomp
[-f <cmd_file>]
[-h]
[-i "<include_path>"]
[-intstyle [ise | xflow | silent | default]]
[-initfile <sim_init_file>]
[[-L|-lib <search_lib>[=<lib_path>]]]
[-prj <prj_file>.prj]
[-rangecheck]
[-v [-verbose] <value>]
[-version]
[<vhdl_files>...]
[-work [<work_library>[=<library_path>] ] <filenames>...]
```

オプションの説明については、[fuse](#)、[vhpcomp](#)、[vlogcomp](#) コマンド オプションセクションを参照してください。

vhpcomp コマンド例

2 つのファイルを使用します。

```
vhpcomp suba.vhd subb.vhd
```

fuse の実行

fuse コマンドでは、次が実行されます。

- 解析ノードでデザインのスタティック エラボレーションを実行
- 各モジュール インスタンスのオブジェクト コードを生成
- 生成したオブジェクト コードとシミュレーション エンジン ライブラリをリンク付けして、シミュレーション実行ファイルを作成

fuse コマンドは、デザインを構成する各デザイン ユニットのオブジェクト コードおよびデータ ファイルを生成し、これらのファイルを isim/<simulation_executable>.sim ディレクトリに保存します。

注記: isim/<simulation_executable>.sim ディレクトリを削除しないでください。削除するとデザインをシミュレーションできません。

fuse コマンドの構文

コマンド オプションの説明については、次の各リンクをクリックしてください。

注記: ご使用の PDF リーダーで [View] → [Toolbars] → [More Tools] で [Previous View] および [Next View] をオンにしておくと、リンクされた情報の前後へ移動がしやすくなります (メニュー表示は PDF リーダーによって異なります)。次に、fuse コマンドの構文を示します。

```
fuse
[-d <macro_definition>=<value>=<value>]
[-f <cmd_file>]
[-generic_top "<parameter>=<value>"]
[-h]
[-hil_zynq_ps]
[-hwcosim_board <arg>]
[-hwcosim_clock <arg>]
[-hwcosim_instance <arg>]
[-hwcosim_no_combinatorial_path]
[-hwcosim_incremental <arg>]
[-i "<include_path>"]
[-incremental]
[-initfile <sim_init_file>]
[-intstyle [ise | xflow | silent | default]]
[[-L|-lib <search_lib>[=<lib_path>]]]
[-log <file_name>]
[-maxdeltaid <number>]
[-maxdelay]
[-maxdesigndepth <depth>]
[-mindelay]
[-mt <value>]
[-nodebug]
[-nolog]
[-nospecify]
[-notimingchecks]
[-o <sim_exe> ]
[-override_timeprecision ]
[-override_timeunit]
[-prj <prj_file>.prj]
[-rangecheck]
[-sdfnoerror]
[-sdfnowarn]
[-sdfmin] | [-sdftyp] [-sdfmax] <root=file>]
[-sdfroot <root_path>]
[-sourcelibdir <directory_name>]
[-sourcelibext <file_extension>]
[-sourcelibfile <file_name>]
[-timeprecision_vhdl <time_precision>]
[-timescale <time_unit/time_precision>]
[-typdelay]
[-v [-verbose] <value>]
[-version]
```

オプションの説明については、[fuse](#)、[vhpcomp](#)、[vlogcomp](#) コマンド オプションセクションを参照してください。

fuse コマンド例

fuse コマンドにプロジェクト ファイルおよびソース ライブラリを指定して実行します。

```
fuse -prj test.prj test -sourcelibfile ./mylib1/lib_abc.v
-sourcelibfile ./mylib1/lib_cde.v
```

test.prj には、verilog work test.v が含まれます。

fuse では test.v で使用されるモジュールの -sourcelibfile オプションで指定したファイルが使用され、モジュールが解析され、test デザインが生成されます。

test.v ファイルにインスタンス化されている <module_name> という名前の未解決のモジュールに対し、コンパイラでは ./mylib1、./mylib2 ディレクトリの順で modulename.v という名前のファイルが検索されます。

fuse コマンドに work.glbl オプションを付けて実行します。

```
fuse work.testbench work.glbl -prj design.prj -L simprims_ver -o
isim.exe
```

注記： タイミング シミュレーションで fuse を使用する場合、glbl を <top_name> として使用する必要があります。-o オプションは、必須ではありません。使用しない場合、シミュレーション実行ファイルの名前がデフォルト名 x.exe になります。

例：

```
fuse topunit work.glbl -prj <mydesign>.prj -o <my_sim>.exe
```

注記： PRJ ファイルに含まれる特定の行を除外するには、"--" オプションを使用します。

Verilog 検索順

fuse コマンドでは次の検索順で検索を実行し、インスタンス化された Verilog デザイン ユニットをバインドします。

1. ライブラリは、Verilog コードの `uselib で指定されます。

例：

```
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. コマンド ラインで -lib|-L オプションを使用して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 インスタンス化 シュン ユニット

Verilog デザインにコンポーネントがインスタンス化されている場合、fuse コマンドでそのコンポーネント名が Verilog ユニットとして処理され、Verilog モジュールがユーザーにより指定された順序のユニファイド論理ライブラリで検索されます。

- Verilog ユニットが見つければ、そのユニットにバインドされ、検索が終了します。

- Verilog ユニットが見つからない場合は、インスタンス化されているモジュール名が VHDL エンティティ名として処理され、次のように検索が続行されます。
fuse コマンドでは、インスタンス化されたモジュール名と同じエンティティ名がユーザーにより指定された順序のユニファイド論理ライブラリで検索され、最初に一致した名前が見つかり、検索を終了します。
- 大文字/小文字を区別した検索が実行されない場合、fuse コマンドでは、拡張識別子として作成された VHDL デザイン ユニット名がユーザーにより指定された順序のユニファイド論理ライブラリで検索されます。
- 一致するユニットが見つかり、そのユニット名が選択され、検索は終了します。

注記： 混合言語デザインでは、インスタンス化された Verilog モジュールから VHDL エンティティに関連付けられたポート名の大文字と小文字は区別されません。defparam 文で VHDL ジェネリックを変更することはできないことにも注意してください。

VHDL インスタンス化 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ス化されている場合、fuse コマンドではそのコンポーネント名が VHDL ユニットと判断され、論理ライブラリ work が検索されます。

- VHDL ユニットが見つかり、そのユニットにバインドされ、検索が停止します。
- fuse コマンドで VHDL ユニットが見つからなかった場合は、大文字と小文字を維持したコンポーネント名が Verilog モジュール名として処理され、ユーザーにより指定された順序のユニファイド論理ライブラリで検索が続行されます。一致する名前が見つかり、検索は停止します。
- 大文字/小文字を区別した検索が実行されない場合、fuse コマンドでは、Verilog モジュール (大文字/小文字の識別あり) がユーザーにより指定された順序のユニファイド論理ライブラリで検索されます。一致するユニットが見つかり、そのユニットがバインドされ、検索が終了します。

`uselib (Verilog 指示子)

Verilog 指示子の `uselib は ISim でサポートされ、ライブラリ検索順序に従って実行されます。

`uselib 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 の使用規則

`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。例：

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

この構文は問題ありません。

1 つの `uselib 指示子で複数のライブラリを指定することができます。

この場合、ライブラリの指定順序が検索順序になります。

例：

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

インスタンスエート済みモジュールの検索順は、mylib が最初で、次に yourlib となります。

`uselib dir、`uselib file および `uselib libext と同様、`uselib lib は別の `uselib が使用されるまで、または前に使用された `uselib (すべての Verilog XL の `uselib 指示子を含む) が HDL ソースで使用されたままになっている場合を除いて、指定した解析が開始される際に HDL ファイルで使用されます。

`uselib に何も引数を使用しない場合は、現在アクティブになっている効果が削除されます。

`uselib lib|file|dir|libext.

次のモジュール検索方法は、インスタンスエート モジュールまたは Verific の Verilog エラボレーション アルゴリズムによる UDP を解決するために使用されます。

- まず、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インスタンス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、fuse コマンド ラインを使用し、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ス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ス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 をインスタンスエートし、モジュール B がモジュール C をインスタンスエートする場合、C の親モジュールである B のライブラリである mylib 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 1 つである work ライブラリでインスタンスエート済みモジュールを検索します。
 - HDL ソースのコンパイルされるライブラリ
 - work ライブラリとして設定されたライブラリ
 - work という名前の付いたデフォルトの work ライブラリ

`uselib の例

half_adder.v ファイルを adder_lib という名前の論理ライブラリにコンパイル	full_adder.v ファイルを work という名前の論理ライブラリにコンパイル
<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>

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 というファイルを作成し、次の構文をそのプロジェクト ファイル内で使用します。

```
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。1 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。1 行に 1 つの VHDL ファイルを指定します。
- Verilog の場合、[-d <macro>] を使用するとオプションで 1 つ以上のマクロを定義できるようになります。
- Verilog の場合、[-i <include_path>] を使用するとオプションで 1 つ以上の <include_path> ディレクトリを定義できるようになります。

Verilog シミュレーション用の定義済み XILINX_ISIM マクロ

XILINX_ISIM は、Verilog 定義済みマクロで、値は1 です。この定義済みマクロを使用すると、ツール特定のファンクションを実行したり、またはデザインフローで使用するツールを特定できます。

```
module  
  isim_predefined_macro;  
  integer fp;  
  initial  
  begin  
    `ifdef XILINX_ISIM  
      $display("XILINX_ISIM defined");  
      fp = $fopen("ISIM.dat");  
    `else  
      $display("XILINX_ISIM not defined");  
      fp = $fopen("other.dat");  
    `endif  
    $fdisplay (fp, "results");  
  end  
endmodule
```

デザインのシミュレーション

シミュレーションは、デザインのロジックおよびタイミングを検証するプロセスで、ISim の GUI、Tcl バッチ ファイルまたはコマンド ラインを使用して実行できます。デザイン ファイルを解析し、fuse を使用してシミュレーション実行ファイルを作成すると、論理またはタイミング シミュレーションが実行できるようになります。

シミュレーションの実行

シミュレーションは、次のいずれかの方法で実行します。

- コマンド ラインの使用
- GUI の使用

コマンド ラインの使用

fuse コマンドで生成された実行ファイルを実行すると、シミュレーションをコマンド シェルで開始できます。シミュレーション実行ファイルに `-gui` オプションを指定しなければ、シミュレーションはコマンド ライン モードで開始します。

たとえば、次のように入力します。

```
x.exe
```

ISim の Tcl プロンプトが開きます。このプロンプトにシミュレーション Tcl コマンドをインタラクティブに入力することで、シミュレーションを実行したり、デザインをデバッグできます。

`-tclbatch` コマンドを使用すると、1 つのファイルに複数のコマンドを含めて、シミュレーションの開始時にこれらのコマンドを実行させることができます。

たとえば、次のコマンドを含む `run.tcl` というファイルがあるとします。

```
run 20ns
show time
quit
```

シミュレーションを開始するときにこれらのコマンドを実行するには、次のように入力します。

```
x.exe -tclbatch run.tcl
```

GUI の使用

次の例では、シミュレーション実行ファイルを実行し、GUI を開いています。

```
<executable_name>.exe -gui
```

次の GUI のメニュー コマンドを使用してもシミュレーションを実行できます。

- **[Simulation] → [Restart]**
シミュレーションを停止してシミュレーション時間を 0 に戻します。デザインを読み込み直さずにシミュレーションを再実行するには、**[Step]**、**[Run All]**、または **[Run for the time specified on the toolbar]** を使用します。または、**[Console]** パネルに Tcl コマンドの `restart` を入力します。
- **[Simulation] → [Run All]**
すべてのイベントが実行されるまでシミュレーションを実行します。または、**[Console]** パネルに Tcl コマンドの `run -all` を入力します。
- **[Simulation] → [Run]**
シミュレーションを 100ns または指定した時間分実行します。時間および単位は **[Value]** ボツ

クスに入力します。または、Tcl コマンドの `run` に `-length` および `-unit` オプションを付けて実行することもできます。

- **[Simulation] → [Step]**

HDL 命令に対して1 つずつシミュレーションを実行します。

第 6 章の「シミュレーションの 1 行ずつの実行」および Tcl コマンドの `step` を参照してください。

また、HDL ソースコードの特定の位置までシミュレーションを実行することも可能です。この場合は、ブレークポイントを使用して `run -all` コマンドを実行します。詳細は、第 6 章「ソースレベルでのデバッグ」を参照してください。

注記：現在のシミュレーション時間は、右下端に表示されます。

混合言語シミュレーション

ISim では、混合言語のプロジェクト 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ています。VHDL デザインには Verilog モジュールが、Verilog デザインには VHDL モジュールを含めることができます。

シミュレーションでの混合言語の制限

- VHDL と 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単位でのみ可能です。
- VHDL デザインへの Verilog モジュールのインスタンスエート、Verilog デザインへの VHDL コンポーネントのインスタンスエートのみがサポートされます。それ以外の混合方法はサポートされません。
- Verilog での階層参照では VHDL ユニットの参照できず、VHDL の展開/選択名では Verilog ユニットの参照できません。
- Verilog モジュールとの境界では、一部の VHDL の型、ジェネリック、ポートのみ使用可能です。同様に、VHDL デザイン ユニットの境界では、一部の Verilog の型、パラメーター、ポートのみ使用可能です。
- Verilog モジュールの VHDL ユニットへのバインドには、コンポーネントのインスタンスエーションに基づくデフォルトのバインドが使用されます。具体的には、VHDL デザイン ユニットのインスタンスエートされている Verilog モジュールには、コンフィギュレーション仕様、直接インスタンスエーション、およびコンポーネントのコンフィギュレーションは使用できません。

混合言語シミュレーションでの主要手順

1. オプションで、混合言語プロジェクトのデザイン ライブラリに含まれる VHDL エンティティまたは Verilog モジュールでの検索順を指定します。
2. 混合言語プロジェクトのデザイン ライブラリに含まれる VHDL エンティティまたは Verilog モジュールのバインド順を指定するには、`fuse -L` オプションを使用します。

注記：-L で指定したライブラリの検索順は、Verilog モジュールをほかの Verilog モジュールにバインドする際にも使用されます。

混合言語デザインでのバインドと検索

VHDL コンポーネントまたは Verilog モジュールをインスタンス化するとき、fuse コマンドでは、次が実行されます。

- 最初に同じ言語のユニットをインスタンス化するデザインユニットとして検索します。
- 同じ言語のユニットが存在しない場合は、-lib オプションで指定されたライブラリで別の言語のデザインユニットが検索されます。

ライブラリは、fuse のコマンドラインに入力した順に検索されます。詳細は、[40 ページの「Verilog 検索順」](#)を参照してください。

注記：ISE® Design Suite を使用する場合は、ライブラリ検索順は自動的に指定されるので、ユーザーが指定する必要はありません。

混合言語コンポーネントのインスタンス化

混合言語デザインでは、VHDL デザイン ユニットに Verilog モジュールを、Verilog デザイン ユニットに VHDL モジュールをインスタンス化できます。

混合言語デザインでの境界およびマップに関する注意事項

VHDL および Verilog のデザイン ユニット/モジュールの境界では、次のような制限があります。

- デザイン ユニット レベルが VHDL と Verilog の境界となります。
- VHDL デザインには、1 つ以上の Verilog モジュールをインスタンス化できます。
- VHDL デザインへの Verilog UDP のインスタンス化はサポートされていません。
- Verilog デザインには、VHDL エンティティに対応する VHDL コンポーネントのみインスタンス化可能です。
- Verilog デザインへの VHDL コンフィギュレーションのインスタンス化はサポートされていません。

VHDL デザイン ユニットへの Verilog モジュールのインスタンス化

1. Verilog モジュールと同じ名前で VHDL コンポーネントを宣言します (大文字と小文字を区別)。

例：

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

2. 名前の関連付けを使用して Verilog モジュールをインスタンス化します。

例：

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

ポート タイプが正しく一致しているかどうか確認する方法については、[47 ページの「ポートのマップ」](#)を参照してください。

ポートのマッピング

混合言語プロジェクトで使用されるポート マッピングには、次の規則および制限があります。

サポートされるポート タイプ

表 3-1 は、サポートされるポート タイプをリストしています。

表 3-1：サポートされるポート タイプ

VHDL ¹	Verilog ²
IN	INPUT
OUT	OUTPUT
INOUT	INOUT

1. バッファポートおよびリンケージポートはサポートされません。
2. Verilog では、双方向バス スイッチへの接続はサポートされません。混合デザインの境界では、名前のない Verilog ポートを使用することはできません。

表 3-2 に、混合言語デザインの境界のポートで使用可能な VHDL および Verilog データ型を示します。

表 3-2：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	ネット
std_ulogic	ネット
std_logic	ネット
bit_vector	ベクター ネット
std_ulogic_vector	ベクター ネット
std_logic_vector	ベクター ネット

注記：Verilog の出力ポートでは、reg 型がサポートされます。境界では、reg ポートは出力ネット (ワイヤ) ポートとして扱われます。

注記：混合言語デザインの境界でその他のデータ型を使用すると、エラーが発生します。

ジェネリック (パラメーター) のマッピング

ISim でサポートされる VHDL ジェネリックとそれに対応する Verilog パラメーターは次のとおりです。

- 整数
- 実数
- 文字列
- ブール代数

注記：混合言語デザインの境界でその他のジェネリック型を使用すると、エラーが発生します。

VHDL/Verilog の値のマッピング

表 3-3 は `std_logic` および `bit` にマッピングされる Verilog ステートを一覧しています。

表 3-3 : `std_logic` および `bit` にマッピングされる 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 `strength` は無視されます。VHDL には、`strength` に対応するものではありません。

表 3-4 は、Verilog ステートにマッピングされる VHDL 型の `bit` を一覧しています。

表 3-4 : Verilog ステートにマッピングされる VHDL の `bit`

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

表 3-5 は、Verilog ステートにマッピングされる VHDL 型の `std_logic` を一覧しています。

表 3-5 : Verilog ステートにマッピングされる VHDL の `std_logic`

<code>std_logic</code>	Verilog
U	X
X	X
0	0
1	1
Z	Z
W	X
L	0
H	1
-	X

Verilog では大文字と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名前の関連付けおよびローカルポート名は対応する Verilog ポート名と大文字/小文字も一致させる必要があります。

Verilog デザイン ユニットへの VHDL モジュールのインスタンス化

Verilog デザイン ユニットに VHDL モジュールをインスタンス化するには、VHDL エンティティを Verilog モジュールのようにインスタンス化します。

例:

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

タイミング シミュレーション (ゲートレベル シミュレーション)

タイミング シミュレーションを開始するには、タイミング シミュレーション モデルとバックアノテーション用の標準遅延フォーマット (SDF) ファイルが必要です。まず、NetGen を使用してこれらのファイルを生成してください。詳細は、『合成/シミュレーション デザイン ガイド』(UG626) の「ゲートレベル ネットリストの生成 (NetGen の実行)」を参照してください。この文書へのリンクは、[付録 D「その他のリソース」](#)に含まれます。

コマンド ラインからの Verilog デザインのタイミング シミュレーション

Verilog デザインのタイミング シミュレーションでは、次の規則に従う必要があります。

- \$XILINX/verilog/src/glbl.v を work ライブラリにコンパイルします。
- fuse コマンドで **work.glbl** を <library_name>.<top_name> の 1 つとして指定します。
- fuse コマンドで **-L** <simprims_ver> を指定します。

ISim の EXE コマンド

注記: コマンドは、大文字/小文字が区別されます。

次のセクションでは、ISim の実行、コンパイル、エラボーレーション コマンドの概要とそのコマンド構文とオプションについて説明します。

ISim の EXE ファイルはユーザーが定義できます。このファイルをコマンド ラインで実行すると、シミュレーションが起動します。実行ファイル名は fuse コマンドに **-o** オプションを付けて指定します。ユーザーが定義しない場合、デフォルトの実行ファイル名は **x.exe** になります。

ISim の EXE 構文

次に、このコマンドの構文を示します。

```
<executable_name>.exe <options>
```

説明:

- <executable_name>.exe はユーザーが定義する実行ファイル名で、デフォルトでは **x.exe** です。
- <options> には [表 3-6](#) で定義されるオプションが指定できます。

ISim の EXE コマンドのオプション

表 3-6 は、ISim の EXE コマンドで実行可能なオプションをリストしています。

表 3-6：ISim コマンド オプション

オプション	説明
-f <cmd_file>	コマンド オプションをテキスト ファイルに保存し、今後使用できるようにします。 <cmd_file> で指定したファイルに保存されたオプションが読み出されて実行されます。
-gui	ISim を GUI モードで起動します。
-h	すべてのコマンド ライン オプションとその使用方法を表示します。
-intstyle [ise xflow silent default]	メッセージの表示方法を指定します。 <ul style="list-style-type: none"> • ise: メッセージが ISE のログ ウィンドウに表示されます。 • xflow: XFLOW のメッセージが表示されます。 • silent: メッセージは表示されません。 • default: デフォルトのメッセージ設定になります。
-log <file_name>	<file_name> で指定した名前のログ ファイルが生成されます。
-maxdeltaid <number>	デルタの最大値を整数で指定します。
-nolog	ログ ファイルを生成しません。
-sdfnowarn	SDF の警告メッセージを表示しません。
-sdfnoerror	SDF ファイルで検出されるエラーを警告として処理します。
[-sdfmin -sdftyp -sdfmax] <root=file>]	ISim で使用する遅延のタイプを指定します。 <ul style="list-style-type: none"> • -sdfmin: <root> で <file> が最小の遅延でアノテートされます。 • -sdftyp: <root> で <file> が標準遅延でアノテートされます。 • -sdfmax: <root> で <file> が最大の遅延でアノテートされます。
-sdfroot <root_path>	デザイン階層で SDF のアノテーションが適用されるデフォルトの位置を設定します。
-tclbatch <file_name>	シミュレーション開始後に実行する Tcl スクリプト ファイルを指定します。 <file_name> では、Tcl コマンドを含むファイル名を指定します。 <file_name> の Tcl コマンドを実行後にシミュレーションを終了する場合は、 <file_name> の最後のコマンドに quit を指定しておく必要があります。

表 3-6 : ISim コマンド オプション (Cont'd)

オプション	説明
-testplusarg <string string_value>	<p>シミュレータでこのコマンド ラインの引数文字列と Verilog デザイン ファイルの \$test\$plusarg または \$value\$plusarg システム関数が一致すると、このシステム関数に関連したテストまたはデザイン動作の変更が実行されます。<string> には文字列を入力します。</p> <p>たとえば、-testplusarg HELLO の場合、Verilog ファイルで \$test\$plusargs("HE") が使用される場合、true が返されます。</p> <p><string_value> には Verilog フォーマット指示子の適切な文字列を入力します。この文字列は、\$value\$plusargs システム関数呼び出しに含まれる変数に対する値を供給します。</p> <p>たとえば、-testplusarg FINISH=10000 の場合、Verilog ファイルで \$value\$plusargs("FINISH=%d", stop_clock) が使用され、Verilog のフォーマット指示子 %d が 10000 と一致する場合は、stop_clock で値 10000 が取得され、関数で true が戻されます。Verilog ファイルで指定されている動作を実行するには、同じ文字列または文字列と値をこのコマンド ラインオプションおよびシステム関数の両方で設定する必要があります。</p>
-transport_int_delays	インターコネクト遅延用の転送モデルを使用します。インターコネクト遅延でパルス破棄はありません。
-vcdfile <vcd_file>	Verilog 専用のオプションです。Verilog プロジェクトの VCD 出力ファイルを指定します。デフォルト名は dump.vcd です。
-vcdunit <unit>	Verilog 専用のオプションです。VCD 出力ファイルの時間の単位を指定します。使用可能な値は、fs、ps、ns、us、ms、sec のいずれかです。デフォルトでは ps です。
-view <waveform_file>.wcfg	-gui オプションと組み合わせて使用して、ISim のグラフィカル ユーザー インターフェイスで特定の波形ファイルを開きます。
-wdb <waveform_file>.wdb	<p>シミュレーションデータを指定した WBD ファイルに保存します。</p> <p>たとえば、x.exe -wdb my.wdb を実行すると、シミュレーション データがデフォルトの isimgui.wdb の代わりに my.wdb に保存されます。</p>

ISim の EXE コマンドの例

```
<executable_name>.exe -tclbatch <tcl_file_name> -sdfmin  
<instance>=<sdf_file_name>
```

説明：

- <executable_name>.exe : fuse -o オプションを使用して指定しない限り x.exe が使用されます。
- -tclbatch : 別の Tcl コマンドを実行する際に使用するオプションです (必須ではありません)。
- -sdfmin : 使用する遅延タイプ (最小) を指定します。
- <instance> : SDF (Standard Delay Format) バック アノテーションを実行するインスタンスの階層パス名を指定します。
- <sdf file name> : アノテートする SDF ファイル名を指定します。

fuse、vhpcomp、vlogcomp コマンド オプション

表 3-7 は、fuse、vhpcomp、および vlogcomp コマンドのオプションをリストしています。

表 3-7 : fuse、vhpcomp、vlogcomp コマンド オプション

オプション	説明
-d <macro_definition>=<value>=<value>	Verilog 専用のオプションです。Verilog ファイルで使用されるマクロおよび必要な値を指定します。-d オプションは複数指定できます。 注記： 等号 (=) と値の間にスペースを入れないようにします。スペースを入れると、値の一部と認識されます。スペースを含むパスはダブルクォーテーション (") で囲みます。
-f <cmd_file>	コマンド オプションをテキスト ファイルに保存し、今後使用できるようにします。<cmd_file> で指定したファイルに保存されたオプションが読み出されて実行されます。
-generic_top "<parameter>=<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特定の値で上書きします。たとえば「-generic_top "P=10"」と入力した場合、生成前に最上位のパラメーター P に値 10 が適用されます。
-gui	ISim を GUI モードで起動します。
-h	すべてのコマンド ライン オプションとその使用方法を表示します。
-hil_zynq_ps	Zynq Processor System (PS) Hardware In Loop (HIL) シミュレーションをイネーブルにします。
-hwcosim_board <arg>	このハードウェア協調シミュレーション (HWCosim) ではボード名を指定します。
-hwcosim_clock <arg>	HWCosim インスタンスのクロック ポートを指定します。
-hwcosim_instance <arg>	HWCosim オプションで、FPGA で実行されるインスタンスの階層名を指定します。 例：/testbench/UUT
-hwcosim_no_combinatorial_path	HWCosim オプションで、FPGA で実行されるデザインに入力から出力までの純粋な組み合わせパスが含まれない場合に、シミュレーションの速度を上げます。
-hwcosim_incremental <arg>	HWCosim のオプションで、インプリメンテーション プロセスをスキップして、以前に作成した BIT ファイルを再利用します。 使用できる値は、00 (デフォルト) と 1 です。
-i "<include_path>"	Verilog でのみ使用できます。 fuse で vlogcomp が呼び出された場合、Verilog の 'include で指定されているパスを使用します。 1 つの 'include パスにつき 1 つの -i を使用できます。-i は、複数指定できます。この場合、パスをクォーテーションで囲み、パス間にスペースを入力します。
-incremental	最後のコンパイルから変更されたファイルのみをコンパイルします。

表 3-7: fuse、vhpcomp、vlogcomp コマンド オプション (Cont'd)

オプション	説明
-initfile <sim_init_file>	デフォルトの xilinxsim.ini ファイルで提供される論理ロケーションから物理ロケーションへのマップに追加または上書きするためのライブラリの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。
-intstyle [ise xflow silent default]	メッセージの表示方法を指定します。 <ul style="list-style-type: none"> ise: メッセージが ISE のログ ウィンドウに表示されます。 xflow: XFLOW のメッセージが表示されます。 silent: メッセージは表示されません。 default: デフォルトのメッセージ設定になります。
-ise <file>	ザイリンクス ISE ファイルを指定します。
[-L -lib <search_lib>[=<lib_path>]]	ほかのライブラリを指定し、さらにオプションでそれらのライブラリの物理パスを指定します。 リソース ライブラリとして処理される -L オプションは複数回使用できます。このオプションで物理パスを指定すると、xilinxsim.ini ファイルで指定されているマップが無視されます。<search_lib> は指定のライブラリの論理名、<lib_path> は物理ライブラリへのパスを指定します。 注記: 等号 (=) と値の間にスペースを入れないようにします。スペースを入れると、値の一部と認識されます。スペースを含むパスはダブルクォーテーション (" ") で囲みます。
-log <file_name>	<file_name> で指定した名前のログ ファイルが生成されます。
-maxdeltaid <number>	デルタの最大値を整数で指定します。
-maxdelay	Verilog 専用のオプションです。fuse で vlogcomp が呼び出された場合、最大遅延を使用します。
-maxdesigndepth <depth>	fuse エラボーレーターで許容されるデザインの最大幅を上書きします。最大幅を超えると、fuse エラボーレーターでエラーが発生します。fuse でデザインに無限に反復されるインスタンシエーションがあると誤って判断されるような場合は、このオプションを使用して幅を増やすことができます。
-mindelay	Verilog 専用のオプションです。vlogcomp が呼び出された場合、最小遅延を使用します。
-mt <value>	平行して実行するサブコンパイル ジョブ数を指定します。on、off、または 2 以上の整数を指定できます。 デフォルトでは on が設定されており、コンパイラによりシステムのコア数に基づいて自動的に値が選択されます。
-nodebug	HDL コードのデバッグ情報を含まない出力を生成します。出力にデバッグ情報を含まないようにすると、シミュレーションが高速になります。デフォルトでは、HDL デバッグ ユニットが生成されます。
-nolog	ログ ファイルを生成しません。

表 3-7：fuse、vhpcomp、vlogcomp コマンド オプション (Cont'd)

オプション	説明
-nospecify	Verilog 専用のオプションです。ブロック指定機能をディスエーブルにします。
-notimingchecks	Verilog 専用のオプションです。タイミング チェックをディスエーブルにします。
-o <sim_exe>	シミュレーション実行出力ファイルの名前を指定します。<sim_exe> はファイル名です。このオプションを使用しない場合、デフォルトの実行ファイル名は次のとおりです。<work_library>/<mod_name>/<platform>/x.exe <ul style="list-style-type: none"> • <work_library>: 作業ライブラリ • <module_name>: 最初に指定される最上位モジュール • <platform>: オペレーティング システム
-override_timeprecision	Verilog 専用のオプションです。-timescale オプションで指定されている時間精度を使用してデザインに含まれる Verilog モジュールの時間精度を上書きします。
-override_timeunit	-timescale オプションで指定されている時間単位を使用してデザインに含まれるすべての Verilog モジュールの時間単位 (遅延計測単位) を上書きします。
-prj <prj_file>.prj	入力として使用するプロジェクト ファイルを指定します。プロジェクト ファイルは、デザインに関連するすべてのファイルをリストしたものです。このファイルは、ISE ツールにより使用される主要ソース ファイルです。 <prj_file> のファイル拡張子は .prj である必要があります。
-rangecheck	VHDL 専用のオプションです。VHDL での割り当てに値範囲チェックを実行します。このオプションは配列のインデックス範囲チェックには影響しません。ISim では、配列のインデックスが許容範囲にあることが常にチェックされます。 例： <ul style="list-style-type: none"> • 信号が正と宣言されている場合は、fuse で信号が負の値に割り当てられていないかがチェックされ、 • std_logic と宣言されている場合は、信号に有効な std_logic 値 (U、X、0、1、Z、W、L、H、-) のみが割り当てられているかがチェックされます。 <p>注記：このオプションは、インデックス範囲のチェックには関係ありません。シミュレータでは、常にインデックスの範囲がチェック ボックスされます。</p> <p>デフォルトでは -rangecheck はオフです。</p>
-sdfnoerror	SDF ファイルで検出されるエラーを警告として処理します。
-sdfnowarn	SDF の警告メッセージを表示しません。

表 3-7 : fuse、vhpcomp、vlogcomp コマンド オプション (Cont'd)

オプション	説明
<code>[-sdfmin] [-sdftyp] [-sdfmax]</code> <code><root=file></code>	<p>使用する遅延のタイプを指定します。</p> <ul style="list-style-type: none"> -sdfmin : <root> で <file> が最小遅延でアノテートされます。 -sdftyp : <root> で <file> が標準遅延でアノテートされます。 -sdfmax : <root> で <file> が最大遅延でアノテートされます。
<code>-sdfroot <root_path></code>	SDF アノテーションを適用するデザイン階層のデフォルト ディレクトリを設定します。
<code>-sourcelibdir <directory_name></code>	ライブラリ モジュールのソース ディレクトリを指定します。
<code>-sourcelibext <file_extension></code>	モジュールのソースファイルの拡張子を指定します。-sourcelibdir オプションではこれらのファイルのディレクトリを指定します。
<code>-sourcelibfile <file_name></code>	ライブラリ モジュールのファイル名を指定します。
<code>-timeprecision_vhdl</code> <code><time_precision></code>	<p>VHDL 専用のオプションです。すべての VHDL デザイン ユニットに対する時間精度を指定します。time_precision には、数値 (1 10 100 ...) に続けて単位 (fs ps ns us ms s) を入力します。デフォルトは 1ps です。</p>
<code>-timescale <time_unit/</code> <code>time_precision></code>	<p>Verilog 専用のオプションです。効率のよいタイムスケールがない Verilog モジュールに対してデフォルトのタイムスケールを指定します。</p> <ul style="list-style-type: none"> <time_unit> では遅延計測単位を指定します。 <time_precision> では精度の単位を指定します。 <p><time_unit> および <time_precision> には、数値 (1 10 100 ...) に続けて単位 (fs ps ns us ms s) を入力します。デフォルトは 1ns/1ps です。</p>
<code>-typdelay</code>	Verilog 専用のオプションです。vlogcomp が呼び出された場合、標準遅延を使用します。
<code>-timeprecision_vhdl</code> <code><time_precision></code>	<p>VHDL 専用のオプションです。すべての VHDL デザイン ユニットに対する時間精度を指定します。time_precision には、数値 (1 10 100 ...) に続けて単位 (fs ps ns us ms s) を入力します。デフォルトは 1ps です。</p>

表 3-7 : fuse、vhpcomp、vlogcomp コマンド オプション (Cont'd)

オプション	説明
-v [-verbose] <value>	<p>メッセージの表示レベルを指定します。使用できる値は 0、1、または 2 です。デフォルトは 0 です。例：</p> <p>fuse -v 1 では便利なデバッグ情報が表示され、ISim コンパイラで発生する問題の検出に役立ちます。詳細レベルは 1 です。</p> <ul style="list-style-type: none"> 使用可能なライブラリ マップ ファイル (xilinxsim.ini) すべてを読み込んだ後に ISim コンパイラで見られるライブラリ マップを表示します。 デザイン エラポレーターから詳細なメッセージを取得します。 コンパイラの動作に影響する環境変数の現在の値を取得します。 コンパイラで共有されるオブジェクトのリストを取得します。 バージョン番号およびプロセッサなどのオペレーティング システム情報を表示します。 生成コードをコンパイルするのに使用する GCC コンパイラのパスを表示します。
<verilog_files>...	コンパイルする Verilog ソース ファイルを指定します。
<vhdl_files>...	コンパイルする VHDL ソース ファイルを指定します。
-version	コンパイラのバージョンを表示します。
-wdb <waveform_file>.wdb	<p>シミュレーション データを指定した WBD ファイルに保存します。</p> <p>たとえば、x.exe -wdb my.wdb を実行すると、シミュレーション データがデフォルトの isimgui.wdb の代わりに my.wdb に保存されます。</p>
-work [<work_library> [=<library_path>]] <filenames>...	<p>work ライブラリを指定し、さらにオプションで作業ライブラリの物理パスを指定します。このオプションで物理パスを指定すると、xilinxsim.ini ファイルで指定されているマップが無視されます。デフォルトの作業ライブラリは、論理ライブラリ \work です。</p> <p><work_library> は指定の作業ライブラリの論理名、 <library_path> は物理ライブラリへのパスを指定します。</p> <p>例：mywork=C:/home/worklib.</p> <p>注記：等号 (=) と値の間にスペースを入れないようにします。スペースを入れると、値の一部と認識されます。スペースを含むパスはダブルクォーテーション (") で囲みます。</p>

シミュレーションの一時停止

シミュレーションを任意の時間実行している間、[Break] コマンドを使用してシミュレーションを一時停止し、シミュレーション セッションを開いたままにできます。

シミュレーションを一時停止するには、次の手順のいずれかを実行します

- [Simulation] → [Break] をクリックします。
- [Break] ボタンをクリックします。

- コマンド プロンプトで **Ctrl + C** キーを入力します。

シミュレーションが次の HDL 実行行で停止します。シミュレーションが停止した行は、テキストエディターに表示されます。

注記：この動作は、`-nodebug` オプションを使用してコンパイルされていないデザインで発生します。

シミュレーションは、[Run All]、[Run]、[Run for the time specified on the toolbar] (ツールバーのみ)、[Step] コマンドを使用するといつでも再開できます。詳細は、[77 ページの「シミュレーションの 1 行ずつの実行」](#)を参照してください。

シミュレーション結果の保存

オブジェクト (VHDL 信号または Verilog レジスタ/ワイヤ) のシミュレーション結果は、作業ディレクトリに含まれている波形データベース (WDB) ファイル (`<filename>.wdb`) に保存されます。波形ウィンドウにオブジェクトを追加してシミュレーションを実行した場合は、完全デザインのデザイン階層および追加されたオブジェクトの遷移が自動的に WDB ファイルに保存されます。信号順、命名スタイル、基数および色など、波形コンフィギュレーション設定も任意で波形コンフィギュレーション (WCFG) に保存されます。詳細は、[第 4 章「波形の解析」](#)を参照してください。

WDB ファイルへのデータベースの保存

ISim の起動される方法によって、WDB ファイルの名前の付け方は異なります。

- ISE ツールまたは PlanAhead ツールから起動すると、WDB ファイルの名前は [ISim Properties] ダイアログ ボックスで指定されている名前に基づいて付けられます。
- コマンド ラインから起動する場合は、`-wdb` オプションでファイル名を指定します。シミュレーションが実行されると、オブジェクト (VHDL 信号、Verilog レジスタ/ワイヤ) の結果が自動的に WDB に保存されます。別のシミュレーションが現在のシミュレーションとして同じ作業ディレクトリの同じデザインで実行される場合、この新しいシミュレーションの WDB ファイル名は、最初のシミュレーション名に整数が付けられた名前になります。つまり、最初のシミュレーションは上書きされません。たとえば、WDB ファイルが `isim.wdb` という名前の場合、後続のシミュレーション結果は `isim1.wdb`、`isim2.wdb` といった WDB ファイルに書き込まれます。

注記：データベース ファイルの名前は、シミュレーション実行中は変更できません。

WCFG ファイルへの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 (WCFG) ファイルを保存すると、ファイルに自動的に関連する波形データベース (WDB) ファイルへのリファレンスが追加されます。1 つの WDB ファイルに複数の WCFG ファイルを持たせることができます。

WCFG ファイルには、シミュレーション オブジェクトの順番およびそのプロパティ、波形ウィンドウに表示されている仕切り、マーカーなどの追加された波形オブジェクトが保存されます。詳細は、[第 4 章の「波形コンフィギュレーションでの作業」](#)を参照してください。

WCFG ファイルを保存するには [File] → [Save] をクリックして `.wcfg` ファイルのファイル名を指定します。

シミュレーションの終了

シミュレーションは、次のいずれかのコマンドで終了できます。

- [File] → [Exit] をクリックします。
- [Console] パネルのプロンプトに「quit -f」と入力します。
- メイン ウィンドウの右上端の **X** (閉じるボタン) をクリックします。

第 4 章

波形の解析

波形解析を始める前に、次のいずれかの方法で ISim の GUI を開いておく必要があります。

- 読み出し専用モードで前のシミュレーションからデータを表示または解析する場合は、[74 ページの「スタティック シミュレーションを開く」](#)を参照してください。
- ISE® または PlanAhead™ から起動すると、最上位信号を含む波形コンフィギュレーションが表示されます。この後、信号を追加したり、シミュレーションを実行できます。詳細は、[第 3 章の「シミュレーションの実行」](#)を参照してください。
- コマンド ラインから起動する場合は、-gui オプションを付けてシミュレーション実行ファイルを実行します。この場合は、空の波形コンフィギュレーションが表示されますので、シミュレーションを実行する前に波形コンフィギュレーションに信号を追加する必要があります。詳細は、[第 3 章の「シミュレーションの実行」](#)を参照してください。

デザイン データは、[Objects] パネルや [Instances and Processes] パネルなどの GUI のほかのエリアに表示されます。

波形コンフィギュレーションでの作業

波形コンフィギュレーション ファイルに信号およびバスを追加し、そのコンフィギュレーションを WDB ファイルに保存できます。詳細は、[第 5 章の「波形コンフィギュレーションおよび波形データベースを開く」](#)を参照してください。

波形コンフィギュレーションへの信号の追加

GUI のメニュー コマンドまたはドラッグアンドドロップ手法を使用するか、または [Console] パネルで Tcl (ツール コマンド言語) コマンドを入力すると、波形ウィンドウにデザインの信号を表示できます。

注記：波形コンフィギュレーションの作成や信号の追加などの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は一時的に変更されている状態です。詳細は、「シミュレーション結果の保存」を参照してください。

GUI からの信号の追加

1. [Instances and Processes] パネルでデザイン階層を展開してアイテムを選択します。
選択したインスタンスまたはプロセスに対応するオブジェクトが [Objects] パネルに表示されます。
2. [Objects] パネルでオブジェクトを選択します。
3. 次のいずれかの方法を使用して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。
 - 右クリックして [Add to Wave Window] をクリックします。

- [Objects] パネルからオブジェクトを波形ウィンドウの [Name] 列にドラッグアンドドロップします。
- [Console] パネルで **wave add** コマンドを実行します。

Tcl を使用した信号の追加

1. オプションですが [Instances and Processes] パネルおよび [Objects] パネルでデザイン階層をナビゲートするか、または [Console] パネルで **scope** コマンドを入力して、追加するオブジェクトを識別することができます。
2. [Console] パネルで **wave add** コマンドを入力して個別のオブジェクトまたはオブジェクト グループを追加します。

波形コンフィギュレーションと WCFG ファイル

波形コンフィギュレーションと WCFG ファイルは両方とも波形リストのカスタマイズを指しますが、これら 2 つには概念的な違いがあります。

- 波形コンフィギュレーションは、メモリに読み込んで作業するオブジェクト
- WCFG ファイルは波形コンフィギュレーションをディスクに保存した形態

波形コンフィギュレーション名と WCFG ファイル名

波形コンフィギュレーションは名前を付けたり、無名 (Untitled) にできます。名前は、波形コンフィギュレーション ウィンドウ タブに表示されます。

- GUI から Tcl コマンドを入力して波形コンフィギュレーションを WCFG ファイルに保存するとき、WCFG ファイルの名前はコマンドの引数で指定されます。
- 波形コンフィギュレーションを WCFG ファイルから読み込むとき、波形コンフィギュレーションの名前は WCFG ファイルの名前になります。

信号/バスのコピーの追加

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

信号またはバスのコピーを追加するには

1. 波形ウィンドウの波形コンフィギュレーションで信号またはバスを選択します。
2. [Edit] → [Copy] をクリックするか、または **Ctrl + C** キーを押します。
信号/バス名がクリップボードにコピーされます。
3. [Paste] コマンドをクリックするか、または **Ctrl + V** キーを押します。

信号またはバスが波形コンフィギュレーションにコピーされます。信号またはバスは、必要に応じてドラッグアンドドロップして移動できます。

波形コンフィギュレーションのカスタマイズ

表 4-1 にリストされる機能を使用すると波形コンフィギュレーションをカスタマイズできます。詳細は、機能の名前をクリックすると表示されます。

注記：ご使用の PDF リーダーで [View] → [Toolbars] → [More Tools] で [Previous View] および [Next View] をオンにしておくと、リンクされた情報の前後へ移動がしやすくなります (メニュー表示は PDF リーダーによって異なります)。

表 4-1：

機能	説明
カーソル	波形ウィンドウでメイン カーソルとセカンダリ カーソルを使用すると、時間を表示、計測でき、さまざまなナビゲート操作の焦点として機能します。
マーカー	マーカーを波形に追加すると、波形内をナビゲートしながら、特定時間の波形値を表示できます。
仕切り	仕切りを追加して、信号をグループにまとめることができます。
グループ	波形コンフィギュレーションに含まれる信号およびバスを関連信号セットとしてフォルダーにまとめ、整理する方法です。
仮想バス	仮想バスを波形コンフィギュレーションに追加すると、論理スカラーおよび配列追加できます。
オブジェクト名の変更	オブジェクト、信号、バス、グループの名前は変更できます。
名前の表示	名前は、階層名を含めた完全名で表示するか ([Long Name])、信号またはバス名のみを表示するか ([Short Name])、またはカスタム名で表示できます。
基数	デフォルトの基数では、波形コンフィギュレーション、[Objects] パネル、および [Console] パネルで表示されるバスの基数を設定します。
バス ビット順	バス ビット順は最上位ビット (MSB) から最下位ビット (LSB)、またはその逆に変更可能です。

カーソル

カーソルは、主に時間の一時的な指標として使用し、2 つの波形エッジ間の時間を計測するときなど、頻繁に移動して使用します。複数の計測値の時間ベースを確立させるなど、永久的な指標として使用する場合は、波形ウィンドウにマーカーを追加してください。詳細は、62 ページの「マーカー」を参照してください。

メイン カーソルの配置

波形ウィンドウでクリックすると、メイン カーソルがその位置に配置されます。

セカンダリ カーソルの配置

次の手順に従い、セカンダリ カーソルを配置します。

1. 波形をクリックしてホールドし、右側または左側にドラッグします。

これでセカンダリ カーソルが配置されます。

2. **Shift** キーを押しながら波形をクリックします。

セカンダリ カーソルがない場合は、セカンダリ カーソルが現時点でメイン カーソルが配置されている場所に設定され、メイン カーソルはクリックした位置に移動します。

注記：メイン カーソルの配置中にセカンダリ カーソルの位置を保持するには、**Shift** キーを押したままにします。

注記：セカンダリ カーソルをドラッグして配置するときには、ある程度の距離をドラッグしないとセカンダリ カーソルが表示されません。

マーカの移動

手のシンボルが表示されるまでマウスを移動してからクリックして、任意の場所にドラッグします。

波形ウィンドウでカーソルをドラッグするときに **[Snap to Transition]** ボタンがオンの場合 (デフォルト)、中空円または中が塗りつぶされた円が表示されます。

- 中空円は、選択した信号の波形の遷移間にカーソルが置かれたときに表示されます。
- 中が塗りつぶされた円は、選択した信号の波形の遷移上にカーソルが置かれたときに表示されます。



波形ウィンドウのカーソル、マーカ、またはフロート ルーラーがない位置でクリックすると、非表示にできます。

マーカ

マーカは追加、移動、削除できます。

マーカの追加

マーカは波形コンフィギュレーション上のカーソルの位置に追加されます。

1. 波形ウィンドウでマーカを追加する時間または遷移をクリックしてメイン カーソルを配置します。
2. **[Edit]** → **[Markers]** → **[Add Marker]** または **[Add Marker]** ボタン をクリックします。



マーカがカーソルに配置されます。マーカが既にカーソルの位置に存在する場合は、わずかにずれた位置にマーカが配置されます。マーカの時間はマーカ上部に表示されます。

マーカーの移動

マーカーの追加後にドラッグアンドドロップを使用して波形内の別の位置にマーカーを移動できます。

1. マーカー上部にあるマーカー ラベルをクリックして任意の位置にドラッグします。
 - マーカーが移動可能であることを示すドラッグ シンボルが表示されます。波形ウィンドウでマーカーをドラッグするときに **[Snap to Transition]** ボタンがオンの場合 (デフォルト)、中空円または中が塗りつぶされた円が表示されます。
 - 中が塗りつぶされた円は、選択した信号の波形または別のマーカー上にカーソルが置かれたときに表示されます。
 - マーカー上では中が塗りつぶされた円が白色で表示されます。
 - 中空円は、選択した信号の波形の遷移間にカーソルが置かれたときに表示されます。
2. 新しい位置にマーカーをドロップします。



マーカーの削除

コマンド 1 つを使用して 1 つまたはすべてのマーカーを削除できます。

1. マーカーを右クリックします。
2. 次のいずれかを実行します。
 - 文脈依存メニューから **[Delete Marker]** を選択して、マーカー 1 つを削除します。
 - 文脈依存メニューから **[Delete All Markers]** を選択して、マーカーをすべて削除します。

注記： Delete キーを使用しても、選択したマーカーを削除できます。

マーカーの削除を取り消すには、**[Edit] → [Undo]** をクリックします。

仕切り

仕切りは、信号間を視覚的に分けるためのものです。

仕切りの追加

波形コンフィギュレーションに仕切りを追加して、信号をグループにまとめることができます。

1. 波形ウィンドウの **[Name]** 列で信号をクリックすると、その信号の下に仕切りが追加されます。
2. **[Edit] → [New Divider]** をクリックするか、または右クリックして **[New Divider]** をクリックします。

この変更は視覚的なものであり、HDL コードには何も追加されません。新しい仕切りはファイルが保存されるときに波形コンフィギュレーション ファイルに保存されます。

仕切りの変更

仕切りに対しては、次を変更できます。

- 仕切りの名前を変更できます。詳細は、[65 ページの「オブジェクト名の変更」](#)を参照してください。
- 仕切りは、名前をドラッグアンドドロップすると、波形内の別の位置に移動できます。

仕切りの削除

仕切りを削除するには、ハイライトしてから **Delete** キーを押すか、右クリックして **[Delete]** をクリックします。

グループ

グループとは、展開したり閉じたりできるカテゴリのコレクションのことで、波形コンフィギュレーションに信号およびバスを追加して、関連する信号同士をまとめることができます。グループ自体には波形データが表示されず、展開したときにその内容を表示できます。

グループの追加

グループを追加するには、次の手順に従います。

1. 波形コンフィギュレーションで、グループに追加する信号またはバスを選択します。
注記：グループには、仕切り、仮想バス、およびその他のグループを含めることもできます。
2. **[Edit]** → **[New Group]** をクリックするか、または右クリックして **[New Group]** をクリックします。

選択した信号またはバスを含むグループが波形コンフィギュレーションに追加されます。グループにはグループアイコンが表示されます。この変更は視覚的なものであり、HDL コードには何も追加されません。



信号またバスは名前をグループにドラッグアンドドロップすると移動できます。

新しいグループおよびそのネストされた信号/バスは、波形コンフィギュレーション ファイルを保存するときに保存されます。

グループの変更

グループは、次の手順で変更できます。

- グループの名前は変更できます。詳細は、[65 ページの「オブジェクト名の変更」](#)を参照してください。
- グループは、**[Name]** 列内の任意の場所にドラッグアンドドロップすると移動できます。

グループの削除

グループを削除するには、ハイライトしてから **[Edit]** → **[Wave Objects]** → **[Ungroup]** をクリックするか、または右クリックして **[Ungroup]** をクリックします。グループに含まれていた信号/バスは波形コンフィギュレーション階層の最上位に配置されます。

注意： **Delete** キーを押すと、グループおよびネストされた信号およびバスが波形コンフィギュレーションから削除されます。

仮想バス

仮想バスを波形コンフィギュレーションに追加すると、論理スカラーおよび配列追加できます。仮想バスには、バスの波形が表示されます。仮想バスはその下に昇順で表示される信号の波形で構成されており、1次元配列にフラット化されます。

仮想バスの追加

仮想バスを追加するには、次の手順に従います。

1. 波形コンフィギュレーションで、仮想バスに追加する信号またはバスを任意の数だけ選択します。
2. [Edit] → [New Virtual Bus] をクリックするか、または右クリックして [New Virtual Bus] をクリックします。

仮想バスは、[Virtual Bus] ボタンで表示されます。この変更は視覚的なものであり、HDL コードには何も追加されません。

信号またはバスは名前を仮想バスにドラッグアンドドロップすると移動できます。新しい仮想バスおよびそのネストされた信号/バスは、波形コンフィギュレーション ファイルを保存するときに保存されます。

仮想バスの変更

仮想バスでは、次を変更できます。

- 仮想バスの名前は変更できます。詳細は、[65 ページの「オブジェクト名の変更」](#)を参照してください。
- 仮想バスは名前をドラッグアンドドロップすると、[Name] 列内の別の位置に移動できます。

仮想バスの削除

仮想グループを削除してグループに含まれていたアイテムをハイライトするには、次を実行します。[Edit] → [Wave Objects] → [Ungroup] をクリックするか、または右クリックして [Ungroup] をクリックします。

注意：Delete キーを押すと、仮想バスおよびネストされた信号およびバスが波形コンフィギュレーションから削除されます。

オブジェクト名の変更

波形ウィンドウに含まれている信号、仕切り、グループ、バスなどのオブジェクト名は変更できます。

1. [Name] 列でオブジェクト名を選択します。
2. 右クリックして [Rename] をクリックします。
3. 名前を変更します。
4. Enter キーをクリックするか、名前以外の箇所をクリックして、名前を反映させます。

また、オブジェクト名をダブルクリックしても、名前を変更できます。

変更はすぐに反映されます。波形コンフィギュレーションでのオブジェクト名の変更は、デザインのソース コードには影響しません。

名前の表示

名前は、階層名を含めた完全名で表示するか ([Long Name])、信号またはバス名のみを表示するか ([Short Name])、またはカスタム名で表示できます。信号/バス名は、波形コンフィギュレーションの [Name] 列に表示されます。名前が非表示の場合は、次の手順に従います。

- 信号の完全名が表示されるまで [Name] 列の幅を広げます。
- また、[Name] 列の下にあるスクロール バーを使用しても、完全な信号名を表示できます。

表示名の変更

表示名を変更するには、次の手順に従います。

1. 1 つまたは複数の信号/バス名を選択します。複数の信号を選択する場合は、Shift キーまたは Ctrl キーを使用します。
2. 右クリックして [Name] をクリックし、次のいずれかを選択します。
 - [Long] : 階層の完全名を表示します。
 - [Short] : 信号またはバスのみの名前を表示します。
 - [Custom] : 信号のカスタム名を表示します。詳細は、[65 ページの「オブジェクト名の変更」](#)を参照してください。

名前の表示が変更されます。

基数

デフォルトの基数では、波形コンフィギュレーション、[Objects] パネル、および [Console] パネルで表示されるバスの基数を設定します。

デフォルトの基数の変更

デフォルトの基数は 2 進数です。基数を変更するには、次の手順に従います。

1. [Edit] → [Preferences] をクリックします。
2. [Preferences] ダイアログ ボックスの左側で [ISim Simulator] をクリックします。
3. [Default Radix] ドロップダウン リストから基数を選択します。
4. [Apply] をクリックしてから [OK] をクリックします。

個々の基数の変更

[Objects] パネルに含まれている個々の信号 (HDL オブジェクト) の基数は、次の手順に従うと変更できます。

1. [Objects] パネルでバスを右クリックします。
2. [Radix] を選択し、ドロップダウン メニューから該当する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]

注記：[Objects] パネル内の信号の基数を変更しても、波形ウィンドウまたは [Console] パネルの値には影響しません。

波形ウィンドウに含まれる個々の信号の基数を変更するには、波形ウィンドウの文脈依存メニューを使用します。[Console] パネルで基数を変更するには、Tcl コマンドの `isim set radix` を使用します。

バス ビット順

波形コンフィギュレーションではバス ビットを逆転させて、MSB 優先および LSB 優先の信号表現を切り替えることができます。ビット順を反転するには、次の手順に従います。

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

バス ビットの順番が反転されます。[Reverse Bit Order] コマンドの左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。

波形コンフィギュレーションのナビゲーション

波形コンフィギュレーションは、さまざまな方法でナビゲーションできます。

- [階層の展開/非展開](#)
- [ズーム機能](#)
- [フロート ルーラーの表示](#)
- [マーカーを使用した波形値の表示](#)
- [信号遷移の波形値の表示](#)
- [カーソルを使用した時間の計測](#)
- [\[Go To Time\] コマンドの使用](#)
- [\[Show Drivers\] コマンドの使用](#)

階層の展開/非展開

ネストされたグループのオブジェクトを含むウィンドウまたはパネルでは、次のいずれかの方法でその階層を展開または非展開できます。

矢印の使用

矢印をクリックして階層を展開します。階層は 1 度に 1 つ展開できます。

矢印をクリックして階層を閉じます。



メニューの使用

1. オブジェクトを選択します。
2. [Edit] → [Wave Objects] をクリックし、次のいずれかをクリックします。
 - [Expand]
選択されている階層オブジェクトを展開します。階層は 1 度に 1 つ展開できます。
 - [Collapse]
選択したオブジェクトの階層を非展開します。

文脈依存メニューの使用

1. オブジェクトを選択します。
2. 右クリックして次のいずれかをクリックします。
 - [Expand]
選択されている階層オブジェクトを展開します。階層は 1 度に 1 つ展開できます。
 - [Collapse]
選択したオブジェクトの階層を非展開します。

ズーム機能

ズーム機能を使用して波形ウィンドウの波形コンフィギュレーションを表示します。詳細は、[11 ページの「\[View\] ツールバー」](#)を参照してください。

フロート ルーラーの表示

フロート ルーラーでは、波形ウィンドウ上部の標準ルーラーに表示されている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計測を補助します。

フロート ルーラーは表示/非表示を切り替えることが可能で、波形ウィンドウの任意の位置に移動させることができます。フロート ルーラーの時間ベース (時間 0) はセカンダリ カーソルに基づいています。セカンダリ カーソルがない場合は選択したマーカーに基づきます。

セカンダリ カーソル (または選択したマーカー) が存在するときのみフロート ルーラー ボタンおよびフロート ルーラーが表示されます。

1. フロート ルーラーの表示/非表示は、次のいずれかの方法で切り替えることができます。
 - セカンダリ カーソルを配置します。
 - マーカーを選択します。

2. [View] → [Floating Ruler] または [Floating Ruler] ボタン をクリックします。



この手順は 1 度だけ実行する必要がある場合があります。セカンダリ カーソルを配置したり、マーカーを選択するたびにフロート ルーラーが表示されます。

非表示にするには、コマンドを再度選択します。

マーカーを使用した波形値の表示

マーカーは特定時間の波形と交差する線で、波形コンフィギュレーションをナビゲートしたり各マーカーの [Value] 列で信号およびバスの値を表示するのに使用できます。次の手順に従うと、カーソルをマーカー間で移動して、波形値を表示できます。

1. [62 ページの「マーカーの追加」](#)の手順に従い、波形ウィンドウの波形コンフィギュレーションでマーカーを追加します。


マーカーが 1 つある場合は、カーソルとマーカーが同じ位置にあるとき、[Value] 列に信号とバスの値が表示されます。これで作業が完了しました。

複数マーカーがある場合は、次の手順に従います。

2. [Edit] → [Markers] → [Next Marker] または [Next Marker] ツールバー ボタン をクリックします。





カーソルが波形コンフィギュレーションに含まれているマーカー間を順番に移動します。

3. 各マーカーの [Value] 列で値を確認します。
4. [Edit] → [Markers] → [Previous Marker] または [Previous Marker] ツールバー ボタンをクリックします。
カーソルが波形コンフィギュレーションに含まれているマーカー間を逆方向に移動します。
5. 各マーカーの [Value] 列で値を確認します。

信号遷移の波形値の表示

波形で各遷移での信号値を表示するには、[Next Transition] または [Previous Transition] コマンドを使用します。開始点はカーソルです。

[Next Transition] および [Previous Transition] コマンドの使用

1. 信号を選択します。
開始点は、波形のカーソルの位置です。
2. 次の遷移に進めるには、[View] → [Cursors] → [Next Transition] をクリックするか、[Next Transition] ボタンをクリックします。
3. マーカーが信号の次の遷移まで進みます。その遷移でのすべての信号の値が [Value] 列に表示されます。
4. 手順 2 を必要に応じて繰り返します。
5. 前の遷移に戻るには、次の手順に従います。
[Edit] → [Markers] → [Previous Transition] または [Previous Transition] ツールバー ボタンをクリックします。
6. マーカーが信号の前の遷移まで戻ります。その遷移でのすべての信号の値が [Value] 列に表示されます。
7. 手順 4 を必要に応じて繰り返します。

カーソルを移動したり戻したりすると、信号の値がそれに応じて更新されます。

カーソルを使用した時間の計測

メイン カーソルとセカンダリ カーソルを波形コンフィギュレーションで使用すると、時間範囲を計測できます。この時間範囲は、時間の計測に加えて、カーソル間の拡大表示や範囲の印刷などの実行時に焦点としても機能します。

遷移間または 2 つの信号波形間の時間を計測するには、次の手順に従います。

注記： [Snap to Transition] ボタンはデフォルトでオンになっています。カーソルが遷移付近に配置されるとその遷移にスナップされるので、信号遷移に厳密にカーソルを配置できます。

1. 最初の遷移にマウスを置き、マウスの左ボタンを押したままにします。
2. マウスを 2 番目の遷移にドラッグします。
3. マウス ボタンを放します。
セカンダリ カーソルが 1 番目の遷移に、メイン カーソルが 2 番目の遷移に配置されます。
4. 波形コンフィギュレーション下部で値を確認します。
 - X1 : メイン カーソルの時間
 - X2 : セカンダリ カーソルの時間

- **Delta X**：カーソル間の時間範囲

フロート ルーラーが表示されている場合は、カーソルの時間値がルーラーの上部に表示されます。



5. オプション：[Swap Cursors] ボタンをクリックするとカーソルの位置をスワップできます。
6. 時間範囲は、波形コンフィギュレーションをクリックして新しいカーソルを配置するまで表示されます。

[セカンダリ カーソルの配置](#)に従いセカンダリ カーソルを移動すると、自動的に時間範囲が更新されます。

マーカーを使用した時間の計測

フロート ルーラーが表示されているとき、選択されているマーカー上のフロート ルーラーの時間ベースと波形のメイン カーソルおよびマーカー間の時間計測を表示できます。

1. 時間ベースとして選択したマーカーを使用してフロート ルーラーを表示します。
2. さらにマーカーを追加します。
3. 波形内のロケーションにマーカーを移動します。

フロート ルーラーのマーカー ラベルでは、選択したマーカーと新しいマーカー間の時間間隔が表示されます。

時間ベースは、マーカーを選択するだけで切り替えることができます。

また、カーソルおよびマーカーを組み合わせても時間を計測できます。この場合は、セカンダリカーソルを時間ベースとして使用することで、フロート ルーラーでセカンダリ カーソルに対するマーカーおよびメイン カーソルの時間計測を表示できます。

[Go To Time] コマンドの使用

[Go To Time] コマンドを使用すると、カーソルを波形コンフィギュレーションの特定時間にジャンプさせることができます。

ユーザー指定の時間へのジャンプ

波形コンフィギュレーションを表示します。

1. [Edit] → [Go To] をクリックします。
[Go To Time] ボックスが波形ウィンドウの下部に表示されます。
2. カーソルをジャンプさせる先の時間とその単位を入力します。または、場合によってはドロップダウン リストから時間と単位を選択することも可能です。
3. Enter キーを押します。

シミュレーションの最初または最後へのジャンプ

波形コンフィギュレーションを表示します。

1. 波形コンフィギュレーションでシミュレーションの最初にカーソルを移動するには、[Go To Time 0] ボタンをクリックします。
2. 波形コンフィギュレーションでシミュレーションの最後にカーソルを移動するには、[Go To Latest Time] ボタンをクリックします。



[Show Drivers] コマンドの使用

[Show Driver] コマンドを使用すると、信号値またはオブジェクト値での変更に関連するドライバーを表示します。このコマンドを使用して値の変化の原因を特定し、回路の接続が正しいかどうか判断します。ISim では、[Console] パネルに信号またはオブジェクトのドライバーが表示されます。

このコマンドは、次のエリアでオブジェクトをプローブするときに使用できます。

- [Objects] パネル
- 波形ウィンドウ
- [Console] パネル

ドライバーを表示するには、次の手順に従います。

1. オブジェクトまたは信号を選択します。
2. [Edit] → [Wave Objects] → [Show Drivers] をクリックするか、または右クリックして [Show Drivers] をクリックします。

[Console] パネルでは、オブジェクトまたは信号のドライバーが表示されます。ドライバーがない場合は、その旨を伝えるメッセージが [Console] パネルに表示されます。

注記：このコマンドは、[Console] パネルに「show driver」と入力しても実行できます。

波形コンフィギュレーションの印刷

波形コンフィギュレーションは、印刷セットアップの設定を使用して 1 度に 1 つ印刷できます。波形コンフィギュレーションの背景は常に白色で印刷されます。

印刷プレビューの表示

1. [File] → [Print Preview] をクリックします。
2. [Print Preview] ダイアログ ボックスで波形コンフィギュレーションが予期どおりに表示されていることを確認してください。
3. [Print] をクリックするか、または [Setup] をクリックして印刷オプションおよびレイアウトをカスタマイズします。
4. [Close] をクリックして [Print Preview] ダイアログ ボックスを閉じます。

印刷プレビューでは、デフォルトのプリンターの定義に従い波形が白黒またはカラー表示されます。別のプリンターを選択して印刷することもできます。

印刷

1. [File] → [Print] をクリックします。
2. [Print Setup] ダイアログ ボックスで [Page Orientation] (印刷方向)、[Time Range] (時間の範囲)、[Fit Time Range To] (範囲を含めるページ数)などを設定します。

注記：[Time Range] は、波形コンフィギュレーションにメイン カーソルおよびセカンダリ カーソルの両方が配置されている場合はその時間範囲が自動的に表示されます。

3. [OK] をクリックします。
4. [印刷] ダイアログ ボックスでプリンターを選択し、[印刷] をクリックします。

波形コンフィギュレーションがプリンターの設定に従って印刷されます。

カスタム カラーの使用

個々の信号またはバスの表示色を変更して、比較しやすくすることができます。通常の色設定は、[Preferences] ダイアログ ボックスの [Colors] ページで指定されているカラー スキームに含まれています。詳細は、第 2 章の「ISim カラー プリファレンス」を参照してください。

定義済みのカラー スキームを使用することもできますが、カラー スキームを作成することもできます。

信号またはバスの表示色を変更すると、通常設定より優先されます。

1. 信号またはバスを右クリックします。
2. [Signal Color] をクリックして、色を選択します。

信号またはバスの波形が新しい色で表示されます。

注記：カスタム カラーを使用する場合は、X および Z などの特殊な値を含むすべての論理値がその色で表示されます。

第 5 章

シミュレーション結果の表示

ライブシミュレーションは、次から構成されています。

- すべてのシミュレーション データを含む波形データベース ファイル (WDB)
- 波形コンフィギュレーションに含まれるオブジェクトに関連する順序および設定を含む波形コンフィギュレーション ファイル (WCFG)

波形データベース ファイルと波形コンフィギュレーション ファイル

シミュレーションを実行すると、WDB が自動的に開きます。シミュレーションを実行する方法については、第 3 章の「シミュレーションの実行」を参照してください。

ISE[®] ツールまたは PlanAhead[™] ツールからシミュレーションを実行すると、デフォルトの波形コンフィギュレーションが自動的に開きます。波形コンフィギュレーションは複数開くことができます。

シミュレーションをコマンド プロンプトまたはバッチ スクリプトを使用して実行するときは、GUI の起動時にデフォルトで波形コンフィギュレーションは開かないので、手動で開くか作成する必要があります。詳細は、次のセクションおよび第 3 章の「シミュレーションの実行」を参照してください。

GUI から波形コンフィギュレーションを開く方法

1. [File] → [Open] をクリックします。
2. [ファイルの種類] で **.wcfg** を選択します。
3. 開くファイルを選択したら、[OK] をクリックします。

波形コンフィギュレーションが波形ウィンドウに表示されます。1 つのシミュレーション セッションで複数の波形コンフィギュレーションを開くことができます。波形コンフィギュレーションを表示するときは、タブをクリックします。

コマンド プロンプトから波形コンフィギュレーションを開く方法

GUI の起動時にデフォルトで波形コンフィギュレーションが開かないため、既存の波形コンフィギュレーションを開く `-view` オプションを含めることができます。

シミュレーション実行ファイルを次の構文を使用して実行します。

```
<sim_exe>.exe -gui -wdb <wdb>.wdb -view <wcfg>.wcfg
```

説明:

- `-gui`: GUI を起動します。
- `-wdb <wdb>.wdb`: シミュレーション データを格納するファイル名を指定します。

- `-view <wcfg>.wcfg`: 指定した波形ファイルを GUI で開きます。

GUI に新しいデータベース (ライブ シミュレーション) が開きます。WCFG のシミュレーション オブジェクトがデータベースに含まれるシミュレーション オブジェクトに対応する場合は、データベースからデータがあらかじめ波形コンフィギュレーションに入力されます。

波形コンフィギュレーションの新規作成方法の詳細は、第 4 章の「[波形コンフィギュレーションでの作業](#)」を参照してください。

スタティック シミュレーションを開く

スタティックの読み出し専用のシミュレーションには、次が含まれます。

- 波形コンフィギュレーションに含まれるオブジェクトに関連する順序および設定を含む波形コンフィギュレーション ファイル (WCFG)
- 前に実行したライブ シミュレーションからのシミュレーション データを含む WDB ファイル

波形コンフィギュレーション ファイルでは、波形データベースが参照されます。シミュレーションは、スタティック シミュレータでは実行できません。ライブ シミュレーションを開く方法の詳細は、第 3 章の「[デザインのシミュレーション](#)」を参照してください。

波形コンフィギュレーションおよび波形データベースを開く

直前のシミュレーションの波形コンフィギュレーション (WCFG) ファイルおよびシミュレーション データ (WDB) を開く場合は、次の方法のいずれかを使用します。

波形コンフィギュレーションおよび波形データベースを開くには、次の手順に従ってください。

1. **isimgui.exe** を実行して、スタティック シミュレータを開きます。
2. [File] → [Open] をクリックし、[ファイルの種類] で **.wcfg** を選択し、波形コンフィギュレーション (WCFG) ファイルを選択します。

または、**isimgui.exe -open <wcfg_file>.wcfg** と入力します。

説明:

- **isimgui.exe**: アプリケーション実行ファイルです。
- **-open**: 指定したファイルを開くようにツールに命令します。
- **-view <wcfg>.wcfg**: 指定した波形ファイルを GUI で開きます。

スタティック シミュレータでは、トレースされたすべての信号および関連する波形データベースと共に波形コンフィギュレーションが表示されます。

既存の WCFG および関連しない波形データベースを開く

シミュレーション データ (WDB) を読み込んでデータベースには関係しない WCFG ファイルを表示できます。この方法でスタティック シミュレーションを開くと、複数のエンジニアが同じシミュレーション結果 (WDB ファイルに格納されている遷移) のさまざまな表示 (WCFG ファイルでキャプチャ) を確認するときには有益です。

ISim では、WCFG に含まれているが WDB ファイルで見つからないオブジェクト名に警告メッセージを発行し、一致するオブジェクトのみを表示します。

コマンド プロンプトで次のいずれかを実行します。

```
isimgui.exe -open <wdb>.wdb -view <wcfg>.wcfg
```

説明：

- `isimgui.exe`：アプリケーション実行ファイルです。
- `-open <wdb>.wdb`：指定した波形データベースを ISim グラフィカル ユーザー インターフェイスに開きます。
- `-view <wcfg>.wcfg`：指定した波形ファイルを ISim グラフィカル ユーザー インターフェイスに開きます。

波形データベースおよび新しいデフォルトの WCFG を開く

解析を実行するシミュレーション データ (WDB) があるが、以前に使用した WCFG を開かない場合は、次の方法を使用して波形データベースおよび新しいデフォルトの WCFG を開くことができます。

コマンド プロンプトで次を入力します。

```
isimgui.exe -view <wdb_file>.wdb
```

説明：

- `isimgui.exe`：アプリケーション実行ファイルです。
- `-view <wdb>.wdb`：指定した波形データベースを ISim グラフィカル ユーザー インターフェイスに開きます。

スタティック ビューアーでは、以前のシミュレーションのデータおよび波形ウィンドウに含まれている WDB ファイルのオブジェクトを最大 1000 個まで表示する `Default.wcfg` という名前の新しい波形コンフィギュレーション ファイルが表示されます。デフォルトの WCFG ファイルに信号を追加または削除して保存すると、次回に表示できます。

波形データ ベースのみを開く

以前のシミュレーションから WDB のみを開く場合は、次の手順に従います。

1. コマンド プロンプトで次を入力します。

```
isimgui.exe
```

これにより、スタティック シミュレータが開きます。

2. [File] → [Open] をクリックし、[ファイルの種類] で `.wdb` を選択し、直前のシミュレーションの WDB ファイルを選択します。

または、次を入力します。

```
isimgui.exe -open <wdb_file>.wdb
```

説明：

- `isimgui.exe`：アプリケーション実行ファイルです。
- `-open <wdb_name>.wdb`：指定した波形データベースをグラフィカル ユーザー インターフェイスに開きます。

スタティック ビューアーでは、[Objects] パネルおよび [Instances and Processes] パネルに含まれている以前のシミュレーションのデータが表示されます。波形ウィンドウには波形データは開きません。

[File] → [Open] では既存の波形コンフィギュレーションを開くことができ、[File] → [New] では新しい波形コンフィギュレーションを作成できます。

第 6 章

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、デザインが予期どおりに実行されていることを検証できます。デバッグでは、ソース コードの実行を制御し、問題が発生する可能性がある箇所を特定します。デバッグに使用できるストラテジは、次のとおりです。

- 1 行ずつの実行
開発中のどの段階のデザインでも、[Step] コマンドを使用して HDL デザインのソース コードを 1 行ずつ実行し、デザインが予期どおりに動作するかを検証できます。コードの行ごとに [Step] コマンドが実行され、解析が続行されます。詳細は、[\\$paratext](#)を参照してください。
- HDL コードの特定行にブレークポイントを設定し、ブレークポイントに到達するまでシミュレーションを実行

大型のデザインでは、HDL ソース コードを 1 行ずつ実行するのは面倒な場合があります。ブレークポイントは、HDL ソース コードのあらかじめ決められたポイントに設定でき、シミュレーションが各ブレークポイントで停止しながら実行されます。シミュレーションは、テストベンチの最初からでも現在の位置からでも実行できます。[Step]、[Run All]、または [Run for the time specified on the toolbar] を使用して、シミュレーションを続行します。詳細は、[78 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 1 行ずつの実行

HDL ソース コードをデバッグするために、シミュレーションのどの地点でも [Step] コマンドを使用できます。このコマンドでは、HDL ソース コードを 1 行ずつ実行して、デザインが予期どおりに機能しているかを検証できます。黄色の矢印により、現在実行されている行が示されます。



このコマンドの実行中に、さらに停止ポイントを設定するためにブレークポイントを作成することも可能です。ISim でのデバッグ方法の詳細は、[78 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. シミュレーションを 1 行ずつ実行するには、次の手順に従います。
- 現在の実行時間から、次の手順のいずれかを実行します
 - [Simulation] → [Step] をクリックします。
 - [Step] をクリックします。
 - [Console] パネルで「[\\$paratext](#)」と入力します。



波形ウィンドウに新しいタブが開き、最上位デザイン ユニットに関連する HDL ファイルが表示されます。

- 開始 (Ons) から、シミュレーションを再開します。テストベンチの開始点に時間をリセットするには、[Restart] コマンドを使用します。詳細は、[第 3 章の「シミュレーションの実行」](#)を参照してください。

2. [Window] → [Tile Horizontally] (または [Window] → [Tile Vertically]) をクリックして波形と HDL コードを同時に表示します。
3. デバッグが完了するまで [Step] コマンドを繰り返します。

各行が実行されるたびに、黄色の矢印が 1 行ずつ進むことが確認できます。シミュレータで別のファイルの行が実行される場合は、新しいファイルが開きコード内を黄色の矢印が 1 行ずつ進みます。多くのシミュレーションでは、[Step] コマンドの実行中に複数ファイルが開きます。[Console] タブでは、[Step] コマンドが HDL コードでどこまで進んでいるかも表示されます。

ブレークポイントの使用

ブレークポイントは、ソース コードに含まれるユーザー定義の停止ポイントで、デザインをデバッグするときに使用します。ブレークポイントは、Step コマンドを使用してコードの各行でシミュレーションを停止すると時間がかかりすぎる可能性がある大型のデザインのデバッグで特に役に立ちます。

HDL ファイルの実行行にブレークポイントを設定し、ブレークポイントが設定されているソースコード行に到達するまでコードを継続して実行できます。

注記：ブレークポイントを設定できるのは、実行コード行のみです。実行行以外の行にブレークポイントを配置しても、追加されません。

ブレークポイントの設定

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] をクリックするか、[Toggle Breakpoint] ボタン をクリックします。
2. HDL ファイルでコード行の行番号の右側をクリックします。行の横にブレークポイント アイコンが表示されます。



注記：コード行を右クリックして [Toggle Breakpoint] をクリックしても、同じ操作を実行できます。

終了すると、シミュレーション ブレークポイント アイコンがコード行の横で開き、ブレークポイントのリストが [Breakpoints] パネルに表示されます。

ブレークポイントを使用したデザインのデバッグ

1. HDL ソース ファイルを開きます。詳細は、24 ページの「HDL ソース ファイルを開く」を参照してください。
2. 78 ページの「ブレークポイントの設定」に示すように、HDL ソース ファイルで実行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を設定するまで、手順 1 と 2 を繰り返します。
4. 波形ウィンドウをクリックして波形に戻ります。
 - 最初から実行するには、[Simulation] → [Restart] をクリックします。
 - [Simulation] → [Run All] または [Simulation] → [Run for Specified Time] を使用します。

シミュレーションはブレークポイントまで実行され、停止します。HDL ソース ファイルが表示され、ブレークポイントの停止位置が黄色の矢印で示されます。



5. 波形ウィンドウに戻り、信号値の変化などデザインの動作がブレイクポイントで予期どおりであることを確認します。
6. 結果に満足するまで、ブレイクポイントごとに上記の手順を繰り返します。

HDL ソース ファイルで設定したブレイクポイントに停止しながら、制御されたシミュレーションが実行されます。


デザインのデバッグ中に [Simulation] → [Step] を実行して、一行ずつシミュレーションを進めることでデザインを厳密にデバッグすることもできます。

ブレイクポイントの削除

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

1 つのブレイクポイントの削除

次のいずれかの手順で 1 つのブレイクポイントを削除します。

- [Breakpoint] ボタンをクリックします。
- Tcl プロンプトに次を入力します。
 - 「**bp list**」と入力してデザインに含まれるブレイクポイントすべてをリストし、各ブレイクポイントのインデックス番号および行数を示します。
 - 「**bp del**」または「**bp remove**」に続けてブレイクポイントのインデックス番号を入力します。詳細は、104 ページの「bp」を参照してください。

注記：ブレイクポイントは [Breakpoints] パネルで右クリックして [Delete] をクリックするか [Delete] ボタン をクリックしても削除できます。

全ブレイクポイントの削除

すべてのブレイクポイントを削除するには、次のいずれかの手順を使用してください。

- [View] → [Breakpoint] → [Delete All Breakpoints] をクリックします。
- [Delete All Breakpoints] ボタン をクリックします。
- Tcl プロンプトに「**bp clear**」と入力します。

消費電力のアクティビティ データの書き出し

ISim では、デザインのスイッチング アクティビティ データを含むファイルを書き出すことができ、次の 2 つの操作に役立ちます。

- XPower Analyzer などの消費電力解析ツールを使用した消費電力の概算
- マップ、配置配線 (PAR) ツールを使用した消費電力を最適にするデザインのインプリメンテーション

スイッチング アクティビティ データは RTL レベルまたは配置配線 (PAR) が完了しているデザインのシミュレーションから書き出すことができます。マップ、PAR、および XPower Analyzer では、RTL および配置配線後のシミュレーションから生成されたスイッチング アクティビティ データを使用できます。

消費電力解析および消費電力を最適化したインプリメンテーションの精度が高くなるよう、配置配線シミュレーションで生成したスイッチング アクティビティ データを使用するようにしてください。これでデータが配置配線の結果から生じたデザイン内部ノードと一致します。

また、消費解析および消費電力を最適にしたインプリメンテーションで RTL シミュレーションから生成されるスイッチング アクティビティ データを使用することも可能です。このシミュレーションは配置配線後のシミュレーションよりも高速です。ただし、デザインの入力および出力のアクティビティ データのみが考慮されます。ツールでは、デザインの内部ノードのアクティビティを予測するためにベクターレス解析アルゴリズムが使用されます。

これらのツールでスイッチング アクティビティ データを使用する方法についての詳細は、『コマンドライン ツール ユーザー ガイド』でインプリメンテーション ツール (MAP と PAR) を、消費解析方法は XPower Analyzer ヘルプを参照してください。これらの文書へのリンクは、[付録 D「その他のリソース」](#)に含まれます。

アクティビティ ファイルの作成

次の 2 種類のスティミュラス ファイルを書き出すことができます。

- SAIF
Switching Activity Interchange format (SAIF) ファイルには、デザインに含まれる信号のトグル カウント (遷移数) が含まれています。また、信号の時間を 0、1、X、Z に指定するタイミング属性も含まれています。SAIF ファイルは VCD ファイルよりも小さいため、消費電力に関連するタスク (消費電力解析または消費電力を最適にするインプリメンテーション) で使用することを推奨します。
- VCD
Value Change Dump (VCD) ファイルは、ヘッダー情報、変数定義、およびシミュレーションの各ステップでの値の変化の詳細を含む ASCII ファイルです。このファイルを使用すると、デザインの消費電力を概算できます。このファイルの計算時間は長くなる可能性があり、またファイル サイズは SAIF ファイルより大きくなります。

アクティビティ ファイルを書き出すには、次の手順に従います。

1. 消費電力の概算に使用するシミュレーション中の信号遷移を収集するアクティビティ ファイルを作成します。
 - SAIF: Tcl プロンプトで **saif** コマンドを使用します。
 - VCD: Tcl プロンプトで **vcd** コマンドを使用するか、コマンド ラインでシミュレーション実行ファイルに対して **-vcdfile** オプションを指定します。
2. シミュレーションを実行します。たとえば、次のように実行したとします。

```
run 1000 ns
```

コマンド ラインでシミュレーション実行ファイルを使用してシミュレーションを実行する場合は、手順 1 と 2 を同時に実行できます。

3. シミュレーションが完了したら **saif** または **vcd** コマンドを使用して SAIF または VCD ファイルを閉じます。次はその例です。

```
saif close
```

```
vcd dumpoff
```

4. 作業ディレクトリから SAIF または VCD ファイルを取り出してほかのツールで使用します。

第 8 章

ハードウェア協調シミュレーションの使用

ハードウェア協調シミュレーション (HwCoSim) は、ツールベースの HDL シミュレーションの補足フローとして統合されています。この機能を使用すると、デザインまたはデザインの一部分のシミュレーションをハードウェアで実行できます。これにより、複雑なデザインのシミュレーションを迅速化してデザインがハードウェアで正しく動作することを検証できます。

要件

ハードウェア協調シミュレーションを実行するには、次の要件を満たす必要があります。

- Xilinx® ISE® Design Suite 14.x (いずれかのエディション)
 - 32 ビットまたは 64 ビットの Windows または Linux
 - JTAG ヘッダーが付いている FPGA ボード
- サポートされる FPGA デバイスは、次のとおりです。
- Virtex®-4、Virtex-5、Virtex-6、Virtex-7
 - Spartan®-3、Spartan-3E、Spartan-3A、Spartan-3AN、Spartan-3A DSP、Spartan-6
 - Artix™-7、Kintex™-7
- ザイリンクス パラレル ケーブル IV またはプラットフォーム ケーブル USB

モデルの使用

ハードウェア協調シミュレーション (HwCoSim) では、ロジック ベース デザイン向けモデルとハイブリッド デザイン向けのモデルの 2 つのモデルがサポートされています。

完全ロジック ベース デザイン

完全ロジック ベース デザインの協調シミュレーションは、ISim シミュレータとロックステップ方式 (並列処理) で実行されます。協調シミュレーションを実行するモジュールには、通常次の特徴があります。

- LUT、フリップフロップ、ブロック RAM、および DSP プリミティブのみで構成
- ポートが ISim で制御されており、テストベンチからアクセス可能 (外部 I/O なし)
- モジュールの機能とモジュールのクロック周波数は無関係のため、連続したクロックまたは特定の周波数のクロックで実行する必要がありません。

完全ロジックのロックステップ方式に基づいたハードウェア協調シミュレーションには、次の利点があります。

- 計算量の多いデザインでシミュレーションを高速化

- ハードウェア内での論理検証
- ツール シミュレーションをビットおよびサイクル単位で正確に実行

ハイブリッド デザイン

完全にロジックに基づいている使用モデルはセットアップが単純ですが、ハード IP、外部 I/O、および特定のクロック周波数を必要とするようなデザインには不向きです。ISim のハードウェア協調シミュレーションでは、次のような特徴のデザインをサポートするハイブリッドの協調シミュレーション フローが提供されています。

- ハード IP ブロック、DCM/PLL、および MGT で構成
- ツール シミュレーションとロックステップしてエミュレートされたクロック ソースを使用するクロックもあれば、外部クロック ソースを使用するフリー ランニングのクロックもある。
- 一部のポートを ISim で制御されずツールのテストベンチからもアクセスできない外部 I/O にマップ可能

ハイブリッド協調シミュレーション フローには、次の利点があります。

- シミュレーションの高速化
- ハードウェアで機能を検証
- 通常の協調シミュレーション セットアップより優れており、ハードウェアとソフトウェア間でカスタマイズした通信や複雑な通信が可能

制限事項

ハードウェア協調シミュレーション (HwCoSim) には、次のような制限があります。

- デザインのインスタンスを 1 つのみ選択でき、そのインスタンスに最上位のテストベンチを選択することはできません。
- ハードウェア協調シミュレーションに選択したインスタンスは、XST を使用して合成し、選択したボードのターゲット FPGA デバイスにインプリメントできるようにする必要があります。

ロックステップ方式のハードウェア協調シミュレーションでは、クロック供給および I/O にも制限があります。

- ハードウェアで協調シミュレーションされるインスタンスは、ISim で制御されるエミュレートクロック ソースによりクロックが供給されます。このクロックは、シミュレーションとは非同期です。このため、協調シミュレーションはハードウェアで実行されるデザインのシナリオが完全にはモデル化されず、タイミング シミュレーションとしては機能しません。
- 協調シミュレーションのインスタンスは、外部 I/O または MGT (Multi-Gigabit Transceiver) にアクセスしたり、継続クロックや特定の周波数のクロックを必要とする DCM や PLL などのプリミティブをインスタンス化できません。
- 協調シミュレーションのインスタンスのポートはすべてスライス レジスタまたは LUT に配線できるようにする必要があります。FPGA の一部のリソースでは、IOB やプリミティブの特定ポートなどへの専用配線が必要なため、協調シミュレーションのインスタンスのポートには接続できません。

コンパイルの使用法

ツール ベースの HDL シミュレーションと同様、ハードウェア協調シミュレーションを実行する前にシミュレーション実行ファイルにデザインをコンパイルする必要があります。コンパイルするには、fuse コマンドをコマンド ラインまたは ISE/PlanAhead™ から起動し、協調シミュレーション実行ファイル、ハードウェア協調シミュレーション ビットストリーム、および協調シミュレーション プロジェクト ファイルを作成します。

fuse コマンド ライン フロー

fuse コマンドでは、ハードウェア協調シミュレーション用にデザインをコンパイルする場合、複数のコンパイル オプションがあります。

使用方法

```
fuse -prj <project file> <top level modules>
```

```
-hil_zynq_ps  
-hwcosim_board <board>  
-hwcosim_clock <clock>  
-hwcosim_constraints <constraints file>  
-hwcosim_incremental [0|1]  
-hwcosim_instance <instance>  
-hwcosim_no_combinatorial_path
```

説明:

- **-hil_zynq_ps** : Zynq™ プロセッサ システム (PS) ハードウェア イン ループ (HIL) シミュレーションをイネーブルにします。
- **-hwcosim_board**: 協調シミュレーションに使用するハードウェア ボードを指定します。サポートされるボードのリストは、[92 ページの「サポートされるボード」](#)を参照してください。
- **-hwcosim_clock**: インスタンスのクロック入力のポート名を指定します。
複数のクロックを使用するデザインでは、このオプションで最高速のクロックを指定し、ISim でシミュレーションが最適化されるようにします。その他のクロック ポートは、通常のデータ ポートとして処理されます。
- **-hwcosim_constraints** (オプション): ハードウェア協調シミュレーション用にインスタンスをインプリメントするための追加制約を含むカスタム制約ファイルを指定します。この制約ファイルは、ハードウェア協調シミュレーション フローで外部 I/O またはクロックにマップするインスタンスのポートを指定するときにも使用します。
- **-hwcosim_incremental** (オプション): fuse で前回生成されたハードウェア協調シミュレーション ビットストリームを再利用し、インプリメンテーション フローをスキップするよう指定します。
- **-hwcosim_instance**: ハードウェアで協調シミュレーションするためにインスタンスの完全階層パスを指定します。
- **-hwcosim_no_combinatorial_path**: FPGA で実行されるデザインに入力から出力までの純粋な組み合わせパスが含まれない場合に、シミュレーションの速度を上げます。

ツール フロー

1. プロジェクトのシミュレータに **ISim** が選択されていることを確認します。**[Simulation]** ビューに切り替えます。
2. **[Hierarchy]** ペインでハードウェアで協調シミュレーションするインスタンスを選択して右クリックします。
3. **[Source Properties]** をクリックし、**[Source Properties]** ダイアログ ボックスを開きます。
4. **[Source Properties]** ダイアログ ボックス左側の **[Category]** リストで **[Hardware Co-Simulation]** をクリックします。
5. ハードウェア協調シミュレーションに対して次のプロパティを設定します。

- **[Enable Hardware Co-Simulation]** をオンにします。

ハードウェア協調シミュレーションではインスタンス 1 つのみをイネーブルにできるので、このプロパティをオンにすると、以前にハードウェア協調シミュレーションでイネーブルにされていた別のインスタンスがディスエーブルにされます。

- **[Clock Port]** フィールドにインスタンスのクロック ポート名を入力します。複数のクロックを使用するインスタンスの場合は、最高速のクロック ポートの名前を指定してください。
- **[Target Board for Hardware Co-Simulation]** ドロップダウン リストからボードを選択します。リストには、同じプロジェクト デバイス ファミリの **FPGA** のボードのみが表示されます。
- 以前のハードウェア協調シミュレーションのビットストリームがあり、協調シミュレーションのインスタンスを変更しない場合は、**[Reuse Last Bitstream File]** をオンにして、ハードウェア協調シミュレーションのインプリメンテーション フローをスキップします。
- **[OK]** をクリックします。

ハードウェア協調シミュレーションがイネーブルにされているインスタンスの横に特別なアイコン が表示されます。



[Hierarchy] ペインでテストベンチ モジュールを選択してシミュレーションを実行します。ハードウェア協調シミュレーションは、選択したインスタンスより上の階層レベルで開始する必要があります。

6. テストベンチの **[Instances and Processes]** パネルで次を実行します。
[Simulate Behavior Model] をダブルクリックし、コンパイルおよびシミュレーション プロセスを開始します。
7. **[Simulate Behavior Model]** のプロセス プロパティを開くと、コンパイルおよびシミュレーション プロセスを開始する前に **ISim** のオプションを設定できます。[図 8-1](#) から [88 ページの図 8-3](#) は、手順 1 ～ 8 を示しています。

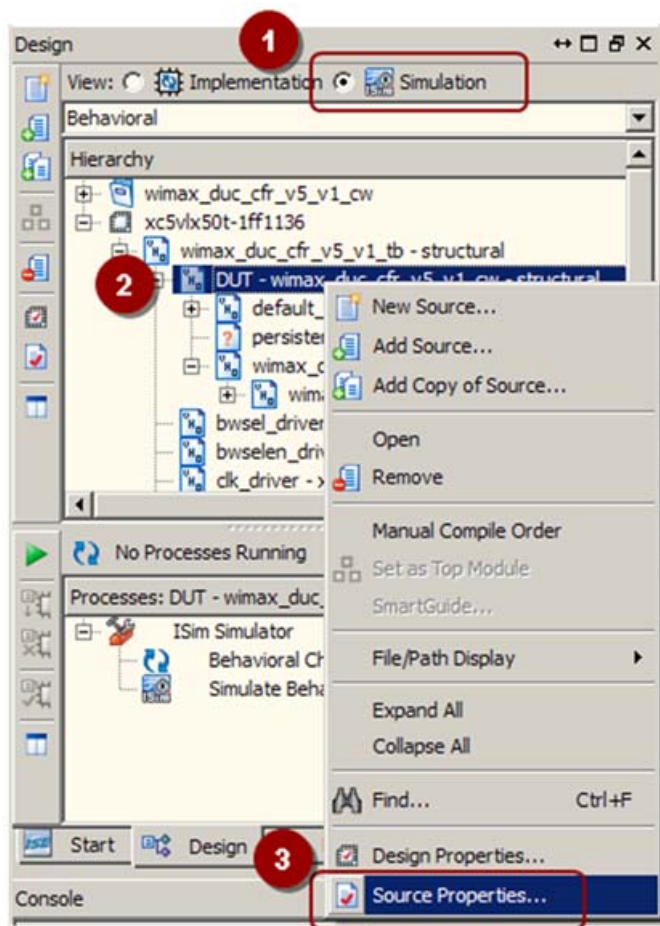


図 8-1 : 手順 1、2、3

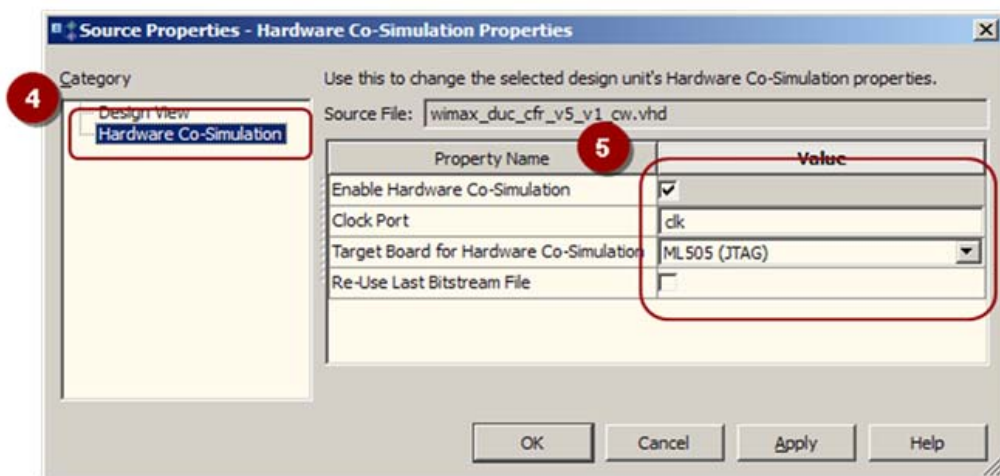


図 8-2 : 手順 4、5

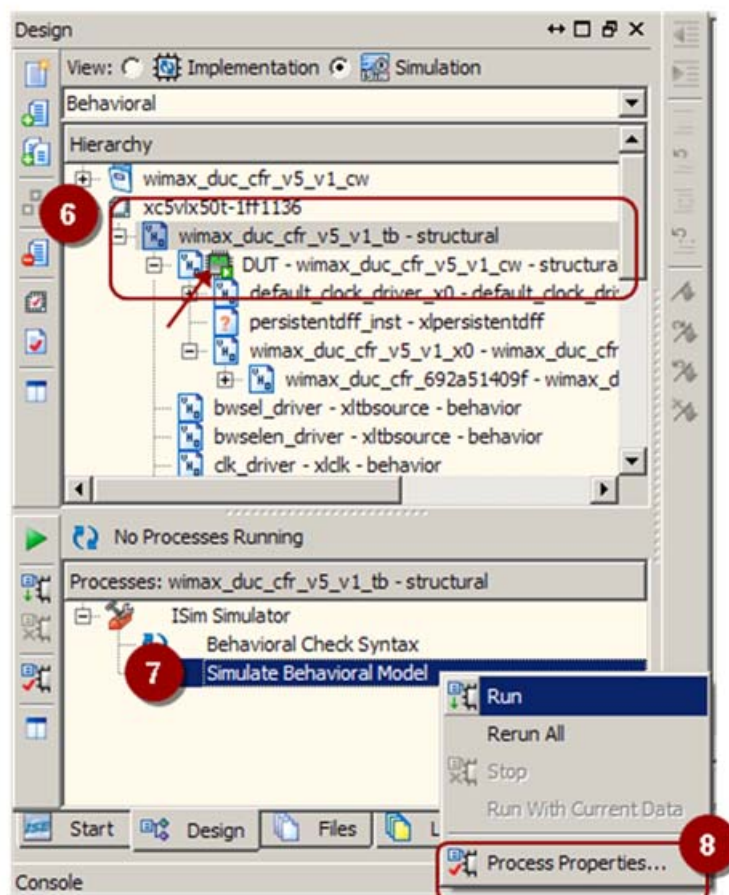


図 8-3：手順 6、7、8

ハイブリッド協調シミュレーション フロー

外部ピンおよびフリー ランニング クロックを使用するには、カスタム制約ファイル (UCF) を供給する必要があります。このフローでは、-hwcosim_constraints オプションで指定した制約ファイルが読み出され、FPGA の IOB にマップするピンが決定されます。

1. ISim シミュレーションとロックステップ方式で実行するデザイン箇所およびフリー ランニングになる箇所を決定します。89 ページの図 8-4 は、このコンセプトを簡単に示しています。

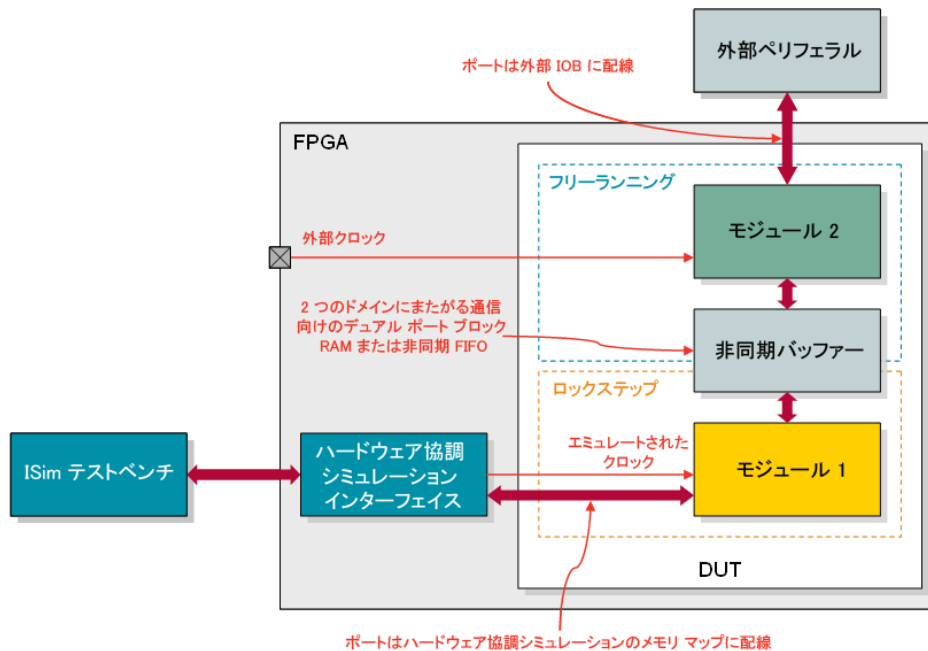


図 8-4 : ロックステップ方式とフリー ランニング方式の適用箇所

2. 元のデザイン制約ファイルをコピーして、カスタム制約ファイルのベースとして使用します。
3. ISim で制御するピンの LOC 制約をコメントアウトするようにカスタム制約ファイルを編集します。LOC 制約が設定されているほかのピンは、外部と想定されます。
4. たとえば、次のような書き込み側をシングル ステップ、読み出し側をフリー ランニングにするような FIFO デザインがあるとします。

```
module FIFO (WCLK, WDATA, WE, RCLK, RDATA, RE);
```

isim_hwcosim.ucf は、次のようになります。

```
# The following pin LOCs commented out as they are driven by ISim
# NET "WCLK" PERIOD = 5 ns HIGH 50%;
# NET "WCLK" LOC = "A1"; # <--- this becomes single-step clock
# NET "WDATA" LOC = "A2"; # <--- these are accessible from ISim
testbench
# NET "WE" LOC = "A3";
NET "RCLK" PERIOD = 10 ns HIGH 50%;
NET "RCLK" LOC = "B1"; # <-- this becomes free-running clock
NET "RDATA" LOC = "B2"; # <-- these go to external IOBs
NET "RE" LOC = "B3";
```

ハードウェア ボード使用方法

次に、ハードウェア協調シミュレーションを実行するためのハードウェアのインストールおよびセットアップ方法を示します。

1. ハードウェア ボードの電源がオフになっていることを確認します。
2. ザイリンクス パラレル ケーブル IV を使用している場合は、手順 2a ~ 2d に従います。
 - a. ザイリンクス パラレル ケーブル IV の DB25 プラグ コネクタを IEEE 1284 準拠 PC パラレル (プリンター) ポート コネクタに接続します。
 - b. 幅の狭い (14 ピン) 6 インチ高速リボン ケーブルを使用して、ザイリンクス パラレル ケーブル IV をボード上 FPGA の JTAG ヘッダーに接続します。
 - c. 電源ジャック ケーブルを PC の キーボード/マウスのコネクタに接続します。
 - d. 必要に応じてケーブル/マウス ケーブルのオス端子をザイリンクス電源ジャック ケーブル (スプリッター ケーブル) のメス コネクタに接続します。
3. ザイリンクス パラレル ケーブル USB を使用している場合は、手順 3a ~ 3b に従います。
 - a. ザイリンクス プラットフォーム ケーブル USB を PC の USB ポートに接続します。
 - b. 幅の狭い (14 ピン) 6 インチ高速リボン ケーブルを使用して、ザイリンクス プラットフォーム ケーブル USB をボード上 FPGA の JTAG ヘッダーに接続します。
4. ボードでポイント ツー ポイント (P2P) イーサネット協調シミュレーションがサポートされており、イーサネット P2P を使用して高速協調シミュレーションを実行する場合は、イーサネット ケーブルを使用して PC のイーサネット ポートをボードのイーサネット ポートに接続します。コンピューターに複数のイーサネット カードまたはポートがある場合は、ボードを接続したポートの MAC アドレスを確認してください。ISim の Tcl コマンド `hwcosim set ethernetInterfaceID` を使用してこのアドレスを ISim エンジンに渡します。
5. ボードの電源を入れて、ケーブルの LED が緑色に点灯することを確認します。
6. ザイリンクス ケーブル ドライバーをインストールします。イーサネットの詳細については、[94 ページの「イーサネットの決定」](#)を参照してください。

ハードウェア協調シミュレーション

ツール シミュレーションの実行ファイルと異なり、ハードウェア協調シミュレーションの実行ファイルではハードウェア ボードと通信して、選択したデザイン箇所のシミュレーションをハードウェアにオフロードします。この実行ファイルは、ツール シミュレーション フローと同様に起動できます。

- 実行ファイルを起動すると、Tcl シェル インターフェイスが開きシミュレーションを制御できます。
- `-gui` オプションと共に起動すると、ISim の GUI フロント エンドが開いて波形が表示されます。

シミュレーションの開始前、およびシミュレーションがリスタートされるたびに、実行ファイルでは FPGA がハードウェア協調シミュレーション ビットストリームでコンフィギュレーションされます。このコンフィギュレーション プロセスは JTAG ケーブルの速度によっては数秒から数十秒かかる場合があります。ISim では、コンフィギュレーションが終了すると [Console] パネルにそれを示すメッセージが表示されます。

ISim ハードウェア協調シミュレーションの Tcl コマンド

ISim では、ハードウェア協調シミュレーションでデザイン階層に含まれるインスタンスのプロパティにアクセスできるよう、次の Tcl (ツール コマンド言語) コマンドを提供しています。これらのコマンドはスコープに影響を受けるので、最初に「**scope**」コマンドを使用してデザイン階層でハードウェア協調シミュレーション向けにイネーブルにされているインスタンスを選択する必要があります。

- **hwcosim get <property>**

現在のスコープでハードウェア協調シミュレーション インスタンスのプロパティを取得します。たとえば、次の Tcl コマンドでは /mytestbench/top/hwcosim_inst にあるハードウェア協調シミュレーション インスタンスの **cableParameters** プロパティを取得します。

```
scope /mytestbench/top/hwcosim_inst
hwcosim get cableParameters
```

- **hwcosim set <property> <value>**

現在のスコープでハードウェア協調シミュレーション インスタンスのプロパティを設定します。hwcosim set コマンドは、初期化が終了した後 (init/restart コマンドの実行後) とシミュレーション実行の前 (run コマンドの実行前) で適用されます。また、このコマンドはシミュレーション実行中には適用されないため、init または restart コマンドを呼び出した後に -hwcosim set コマンドを呼び出す必要があります。たとえば、次の Tcl コマンドでは 2 回のシミュレーション実行で /mytestbench/top/hwcosim_inst にあるハードウェア協調シミュレーション インスタンスの **skipConfig** プロパティが設定されます。

```
init
scope /mytestbench/top/hwcosim_inst
hwcosim set skipConfig 1
run 1000 ns
restart
scope /mytestbench/top/hwcosim_inst
hwcosim set skipConfig 1
run 1000 ns
```

次のハードウェア協調シミュレーションのプロパティは、シミュレーションの実行前に変更できます。

- **skipConfig**

デフォルトは 0 です。FPGA コンフィギュレーションをスキップする場合は 1 に設定します。コンフィギュレーションをスキップするには、FPGA が有効なハードウェア協調シミュレーション ビットストリームでコンフィギュレーションされている必要があります。されていない場合は、予期せぬ動作が発生する可能性があります。

- **cableParameters**

デフォルトは、空の文字列になっています。このプロパティは、iMPACT または ChipScope™ Analyzer でサポートされているサードパーティの JTAG ケーブルを指定するときに使用します。ケーブルのプラグイン パラメーターの指定については、iMPACT ヘルプおよび『ChipScope Pro ソフトウェアおよびコア ユーザー ガイド』(UG029) を参照してください。この文書へのリンクは、付録 D「その他のリソース」に含まれます。

- **shareCable**

デフォルトは 0 です。このプロパティは、JTAG ベースの協調シミュレーション インターフェイスでのみ使用できます。JTAG ケーブルを Xilinx Microprocessor Debugger (XMD) または

ChipScope Analyzer と共有して同時アクセスする場合は **1** に設定します。このモードはハードウェア協調シミュレーションシミュレーションのパフォーマンスを大幅に下げる可能性があるため、必要なときのみ使用してください。

- **ethernetInterfaceID**

デフォルトは、空の文字列になっています。このプロパティは、イーサネット ベースの協調シミュレーション インターフェイスでのみ使用できます。

複数のイーサネット カードがホスト マシンにある場合は、FPGA ボードに接続されているイーサネット カードを選択する必要があります。この場合、イーサネット カードの **MAC** アドレス (xx:xx:xx:xx:xx:xx) を設定します。詳細は、[94 ページの「イーサネットの決定」](#)を参照してください。

サポートされるボード

ISim でハードウェア協調シミュレーション用に新しい FPGA ボードをサポートするには、そのボードに **JTAG** ヘッダーが必要です。次のボード情報を記録したボード サポート ファイルが必要になります。

- **FPGA** パーツ情報
- システム クロックの周期およびピン ロケーション
- **JTAG** バウンダリ スキャン チェーン情報

ボード情報をボード サポート ファイルに含めると、ハードウェア協調シミュレーションでそのボードを使用できます。このボード サポート ファイルを生成する GUI フロントエンドはありません。

別のボードをサポートするには、デフォルトのボード サポート ファイルを変更するか、または別のボード サポート ファイルを供給します。別のボード サポート ファイルを供給する場合は、ファイル名は `hwcossim.bsp` にして、`fuse` を起動するディレクトリに含める必要があります。ボード サポート ファイルには、特定のフォーマットで記述されたボード仕様が含まれます。

次の例では `ml402-jtag` がボード識別子として `fuse` コマンドに供給されて、そのボード向けにデザインがコンパイルされます。

```
'ml402-jtag' => {
  'Description' => 'ML402 (JTAG)',
  'Vendor' => 'Xilinx',
  'Type' => 'jtag',
  'Part' => 'xc4vsx35-10ff668',
  'Clock' => {
    'Period' => 10,
    'Pin' => 'AE14',
  },
  'BoundaryScanPosition' => 3,
},
```

ボード識別子には、次のプロパティが含まれています。

- **Description**: ボードの説明
- **Vendor**: ボードのベンダー
- **Type**: 使用する協調シミュレーション インターフェイスの種類。使用できる値は、次のとおりです。
 - `jtag`

- ppethernet : (ポイント ツー ポイントのイーサネット ベースのハードウェア協調シミュレーション)
- Part : ボード上 FPGA のパーツ名
- Clock : システム クロック情報Period (および VariablePeriods) : サポートされるクロック周期 (ns)
- Pin : クロック ピンのロケーション。差動クロック ソースでは Positive および Negative クロック ピンの両方を入力します。
- BoundaryScanPosition : JTAG バウンダリ スキャン チェーンの FPGA の位置 (1 から順に指定)。この情報は、ザイリンクス iMPACT ツールを実行することにより確認できます。

注記 : ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションでは、さらにフィールドを指定する必要があります。例として、\$XILINX/sysgen/hwcosim/data/hwcosim.bsp ファイルの ml605-ppethernet の設定を確認してください。

ボード サポート ファイル

デフォルトでは、次のザイリンクス ボードがサポートされています。デフォルトのボード サポート ファイルは、ISE® 14.x インストールの次のディレクトリに含まれています。

\$XILINX/sysgen/hwcosim/data/hwcosim.bsp

次に、デフォルトでサポートされるボードを示します。

- **ml401-jtag** - Xilinx® Virtex®-4 ML401 評価プラットフォーム
- **ml402-jtag** - Xilinx ML402 評価プラットフォーム
- **ml403-jtag** - Xilinx ML403 評価プラットフォーム
- **ml405-jtag** - Xilinx ML405 評価プラットフォーム
- **ml410-jtag** - Xilinx ML410 評価プラットフォーム
- **ml501-jtag** - Xilinx Virtex-5 ML501 評価プラットフォーム
- **ml505-jtag** - Xilinx ML505 評価プラットフォーム
- **ml506-jtag** - ML506 ppethernet - Xilinx ML506 評価プラットフォーム
- **ml507-jtag** - Xilinx ML507 評価プラットフォーム
- **xupv5-jtag** - Xilinx University Program XUPV5-LX110T 開発システム
- **ml510-jtag** - Xilinx ML510 評価プラットフォーム
- **ml605-jtag** - ML605 ppethernet Xilinx Virtex-6 ML605 評価プラットフォーム
- **kc705-jtag** - Xilinx Kintex™-7 FPGA KC705 評価キット
- **zc702-jtag** - Zynq™-7000 EPP ボード
- **vc707-jtag** - Xilinx Virtex-7 FPGA VC707 評価キット
- **s3e-sk-jtag** - Xilinx Spartan®-3E スターター キット
- **s3e-mb-jtag** - Xilinx Spartan-3E MicroBlaze 開発キット
- **s3a-sk-jtag** - Xilinx Spartan-3A スターター キット
- **s3an-sk-jtag** - Xilinx Spartan-3AN スターター キット
- **s3adsp1800a-jtag** - Xilinx Spartan-3A DSP 1800A スターター プラットフォーム
- **s3adsp3400a-jtag** - Xilinx Spartan-3A DSP 3400A 開発プラットフォーム
- **sp601-jtag** - sp601 ppethernet Xilinx Spartan-6 SP601 評価プラットフォーム

- **sp605-jtag** - sp605 ppethernet Xilinx Spartan-6 SP605 評価プラットフォーム

イーサネットの決定

複数のイーサネット インターフェイスが存在する場合にイーサネット ベースのハードウェア協調シミュレーションを実行するには、協調シミュレーションを実行するイーサネット インターフェイスを選択する必要があります。

以前のハードウェア協調シミュレーションをポイント ツー ポイント インターフェイス オプションで実行すると、次のエラー メッセージが表示されます。

```
"ERROR:In process wrapper AHIL_INITIALIZE Failed to open hardware
co-simulation instance.Error in Point-to-point Ethernet Hardware
Co-simulation.There are multiple Ethernet interfaces available.Please
select an interface."
```

次の手順に従ってイーサネット ポートを決定し、イーサネットのアドレスを設定、確認して、シミュレーション **run** を検証します。

1. 協調シミュレーション ボードが接続されているイーサネット ポートを決定します。
 - a. システムのコマンド プロンプトでコマンド ターミナル ウィンドウを開きます (**cmd**)。
 - b. コマンド ウィンドウで「**ipconfig -all**」と入力して、イーサネット ポートおよびその接続のリストを表示します。図 8-5 は、コマンド ウィンドウのコマンドを示しています。

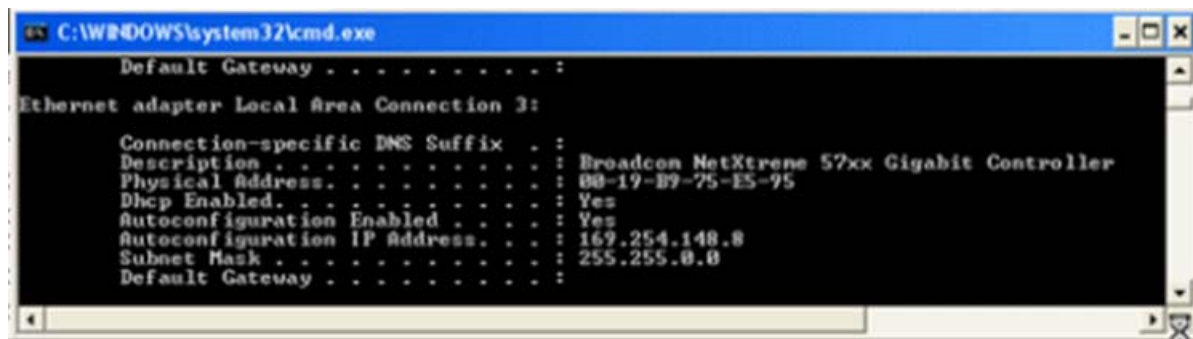


図 8-5：cmd.exe での「ipconfig -all」の入力

- c. 協調シミュレーション ボードに接続されているイーサネット ポートの物理アドレスを検索します。
 - d. 物理アドレスの区切り文字をダッシュ (-) からコロン (:) に変更します。
例: 00:19:B9:75:E5:95
2. 次の手順に従い、イーサネット ポートを設定、確認します。
 - a. GUI を起動します。
 - b. DUT (Design Under Test、被試験デバイス) を選択します。
 - c. Tcl コンソールを表示します。
 - d. Tcl コンソールに次のコマンドを入力します。
 - i. イーサネット アドレスを次のように設定します。

hwcosim set

ethernetInterfaceID <##:##:##:##:##:##> <physical address>

- ii. イーサネット アドレスを確認します。

hwcosim get ethernetInterfaceID

iii. シミュレーションが実行されるか確認します。

run 10us

図 8-6 は、GUI でのプロセスを示しています。

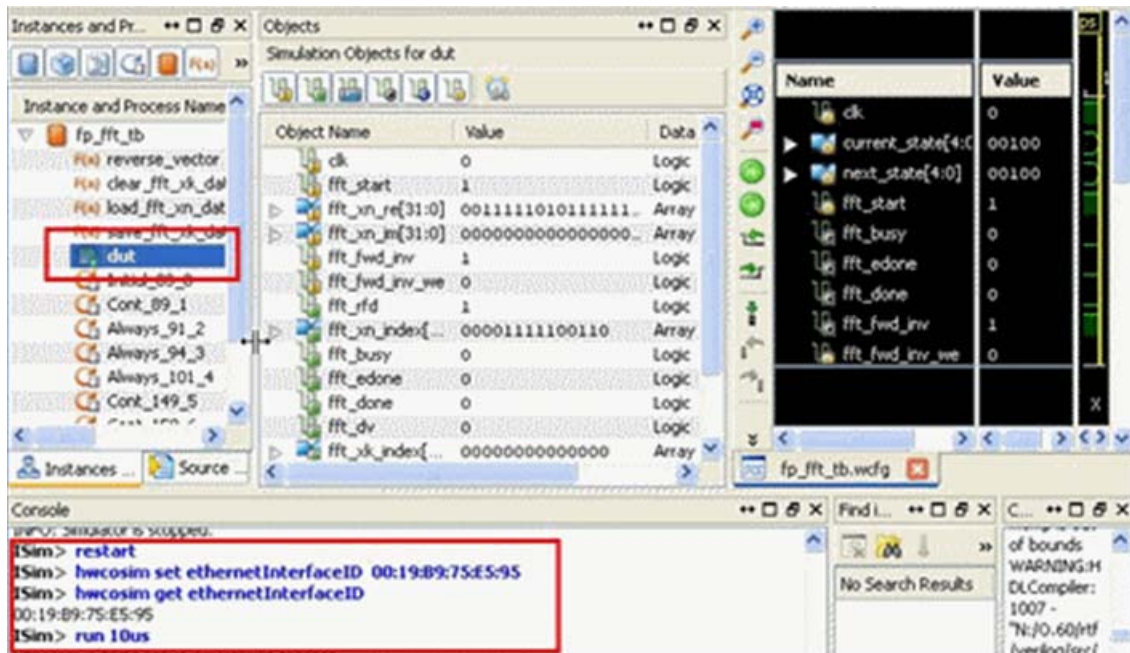


図 8-6 : GUI コンソールでのハードウェア協調シミュレーション プロセス

よく寄せられる質問 (FAQ)

次の質問では、FPGA で協調シミュレーションされるデザイン箇所について説明する際に DUT (Design Under Test) を使用して回答しています。

一般

1. 質問: ハードウェア協調シミュレーション (HWCosim) はすべてのタイプのデザインでサポートされますか。

回答: 84 ページの「制限事項」に記載されているように、一部制限があります。

2. 質問: ハードウェア協調シミュレーションは、論理シミュレーションですか。それともタイミングシミュレーションですか。

回答: ハードウェアで支援される論理シミュレーションで、完全なツールシミュレーションに対してビットおよびサイクルの正確なシミュレーションです。

3. 質問: 独自のボードはどのように使用できますか。

回答: 92 ページの「サポートされるボード」に記載されているようにボード サポート ファイルにボードを追加できます。

コンパイル

1. 質問：デザインに含まれる複数のインスタンスをハードウェアで協調シミュレーションできますか。

回答：いいえ。ハードウェア協調シミュレーションを選択できるのは 1 つのインスタンスのみです。複数のインスタンスの親インスタンスを協調シミュレーションするか、または複数のインスタンスを 1 つのインスタンスにグループ化してください。

2. 質問：ChipScope デバッグ ツールの ICON のように、DUT で既に BSCAN プリミティブをインスタンス化している場合はどうなりますか。

回答：ハードウェア協調シミュレーション インターフェイスでは、ロケーション 1 の BSCAN プリミティブが使用されるので、DUT で別の BSCAN プリミティブがロケーション 1 にインスタンス化される場合に MAP でエラーが発生する可能性があります。

ChipScope ICON で BSCAN プリミティブに別のロケーションを使用するように変更する必要がある場合があります。Spartan-3 などの一部のデバイス ファミリには BSCAN プリミティブが 1 つしかないので、ハードウェア協調シミュレーション インターフェイスは、ChipScope ICON モジュールと共存できません。

ChipScope Analyzer または XMD と JTAG ケーブルを共有するには、Tcl コマンドの `hwcosim set shareCable 1` を実行する必要があります。

シミュレーション

1. 質問：複数の協調シミュレーションを実行するとき、FPGA コンフィギュレーションをスキップして最後にダウンロードしたビットストリームを使用できますか。

回答：この操作を実行するためのコマンド ラインまたは GUI はありませんが、Tcl コマンドの `hwcosim set skipConfig 1` を実行できます。ビットストリーム コンフィギュレーションをスキップすると、複数のシミュレーション実行中、FPGA で実行しているデザインで前の状態が保持されることに注意してください。新しいシミュレーション実行を開始する際は、テストベンチでデザインを正しくリセットする必要があります。

2. 質問：ハードウェアで協調シミュレーションされる DUT に含まれる信号をプローブできますか。

回答：いいえ。ISim ではハードウェア協調シミュレーション インターフェイスを介して DUT のポート インターフェイスのみにアクセスできます。内部信号をデバッグするには、DUT のポートに信号を配線する必要があります。

クロック供給および I/O

1. 質問：DUT にはどのクロックが供給されますか。協調シミュレーション中は DUT のクロック供給はどのように処理されますか。

回答：ボード サポート ファイルで指定されたハードウェア ボードのクロック ピンがマスタークロック ソースですが、DUT の駆動には直接使用されません。このクロック ソースは DCM/PLL を介して特定のクロック周波数 (通常は 25 ~ 100MHz) に調整されます。このクロックはゲーティッド クロック バッファ (BUFGCTRL) を介して DUT のクロック ポートに到達します。ゲーティッド クロック バッファでは、ツール シミュレーションおよび DUT 実行に同期するようにシミュレーション サイクルごとに DUT にクロック パルスが生成されます。

2. 質問：DUT のクロックをフリー ランニングにできますか。

回答：ハイブリッド協調シミュレーションでは、1 つまたは複数の クロック ポートを外部 I/O にマップすることで、DUT の一部をフリー ランニングにできます。ただし、少なくとも DUT の 1 つのクロックはロックステップ方式でツール シミュレーションと同期して供給する必要があります。このようにすることで、DUT をロックステップで実行される部分とフリー ランニングで実行される部分に効果的に分割できます。ISim の ハードウェア協調シミュレーションでは、2 つの部分にまたがるクロック ドメインは挿入されないことにも注意してください。

非同期バッファまたは追加の同期化ロジックが必要になります。

3. 質問：DUT を特定のクロック周波数で実行できますか。

回答：ハイブリッド協調シミュレーションでは、DUT のフリー ランニング部分に外部クロックを供給できます。ただし、ロックステップ方式部分のクロックはツール シミュレーションと同期するよう ISim で制御されるので、特定のクロック周波数に固定することはできません。ロックステップ方式部分の効果的なクロック レートは、入力クロック周波数に関わらず低速になります。

高速クロックで DUT を駆動しても、主なボトルネックがツールとハードウェア間の通信なので、シミュレーションのパフォーマンスは向上しません。MAP や配置配線などのコンパイルプロセスが高速で実行されるように DUT は低速クロック周波数を使用して制約されています。

4. 質問：DUT の一部のポートを DDR メモリ モジュールなどの外部 I/O に接続できますか。

回答：外部 I/O およびクロックは、[88 ページの「ハイブリッド協調シミュレーションフロー」](#)に記載されているように、ハイブリッド協調シミュレーション モードでカスタム制約ファイルを使用するときにサポートされています。

ISim Tcl コマンド

シミュレーション コマンドを使用すると、コマンド プロンプトで対話型シミュレーションを実行できます。

注記：これらのコマンドでは、大文字/小文字が区別されます。

シミュレーション コマンドの入力方法

シミュレーション コマンドは、次のいずれかの方法で入力します。

- **ISim の GUI**
ISim の [Console] パネルにシミュレーション コマンドを入力します。
- **コマンド ライン**
コマンド ラインの Tcl プロンプトでシミュレーション コマンドを入力します。
- **Tclbatch**
Tcl ファイルにシミュレーション コマンドを入力し、-tclbatch オプションを使用してシミュレーション実行ファイルを実行し、Tcl ファイルを参照させます。

個別のコマンドを入力したり、またはシミュレーション スクリプトを作成できます。シミュレーションの Tcl スクリプトの例は、言語テンプレートの [Simulation Constructs] フォルダを参照してください。

言語テンプレートに含まれているこれらの例を表示するには、次の手順に従います。

1. [Edit] → [Language Templates] をクリックします。
2. 言語テンプレートで [Tcl] → [Tools] → [ISim] フォルダを展開します。

よく使用するシミュレーション コマンドに変数を設定すると、すばやくコマンドを実行できます。詳細は、102 ページの「シミュレーション コマンドの別名表記」を参照してください。

注記：Tcl の使用方法の詳細は、<http://www.tcl.tk/> を参照してください。

シミュレーション コマンドのサマリ

表 9-1 には、使用可能なシミュレーション用 Tcl コマンドを簡単な説明と共にリストしています。コマンドの詳細、構文、オプション、具体例については、各コマンドのリンクをクリックしてください。

ご使用の PDF リーダーで [View] → [Toolbars] → [More Tools] で [Previous View] および [Next View] をオンにしておくと、リンクされた情報の前後へ移動がしやすくなります (メニュー表示は PDF リーダーによって異なります)。

表 9-1 : シミュレーション コマンドのサマリ

コマンド	説明
bp	HDL ソース コードでブレークポイントを設定/削除します。
describe	HDL データまたはブロック オブジェクトの情報を表示します。
divider add	新しい仕切りを追加します。
dump	現在のデザイン階層にある変数、ジェネリック、パラメーター、およびネットの名前とその値をリスト表示します。
group add	新しいグループを追加します。
help	指定した ISim コマンドの説明、使用方法、および構文を表示します。
isim condition	ネットまたは Verilog レジスタの特定な条件に基づいたコマンド セットを実行します。
isim force	VHDL 信号、Verilog ワイヤ、または Verilog レジスタに強制的に値を付けるか、または値を削除します。
isim set arraydisplaylength	アレイ型 HDL オブジェクトのエレメント数の制限数を表示します。
isim get radix	使用されているグローバル基数を取得します。
isim get userunit	単位が設定されていない時間の値すべての現在の単位を表示します。
isim ltrace	トレースのオン/オフを切り替えます。
isim ptrace	プロセス実行トレースのオン/オフを切り替えます。
isim set arraydisplaylength	アレイ型 HDL オブジェクトのエレメント数の制限数を設定します。
isim set radix	グローバル基数を設定します。
isim set userunit	単位が設定されていない時間の値すべてに対してデフォルト単位を設定します。
marker add	新しいマーカーを追加します。
onerror	バッチ モードでエラーが発生した Tcl シミュレーション コマンドの直後の動作を制御します。
put	特定のビット、スライス、変数、または信号に値を割り当てます。
quit	コマンド オプションに従い、シミュレーションまたはツールのいずれかを終了します。
restart	シミュレーションを再開してシミュレーション時間を 0 に戻します。

表 9-1 : シミュレーション コマンドのサマリ (続き)

コマンド	説明
<code>resume</code>	<code>onerror</code> コマンドと共に使用して、エラーが発生した後にコマンドを継続して実行します。
<code>run</code>	シミュレーションを開始します。
<code>saif</code>	SAIF (Switching Activity Interchange Format) ファイルを作成し、概算消費電力を記録します。
<code>scope</code>	デザイン階層を移動します。
<code>sdfanno</code>	SDF ファイルの遅延情報を HDL デザインにバックアノテートします。
<code>show</code>	デザインの選択した部分を [Sim Console] パネルに表示します。
<code>step</code>	HDL デザインでシミュレーションを 1 行ずつ実行し、デバッグを援助します。
<code>test</code>	ネットまたはバスの実際の値がコマンドで入力した値と同じであるかどうかを調べます。
<code>vcd</code>	シミュレーション結果を VCD フォーマットで生成します。
<code>virtualbus add</code>	新しい仮想バスを追加します。
<code>wave add</code>	シミュレーション オブジェクトまたはブロックを ISim グラフィック ユーザー インターフェイスで表示されている特定の波形コンフィギュレーションに追加します。
<code>wave log</code>	HDL オブジェクトのシミュレーション出力を波形データベースに記録します。
<code>wcfg new</code>	新しい波形コンフィギュレーションを作成します。
<code>wcfg open</code>	指定された名前の波形コンフィギュレーションを開きます。
<code>wcfg save</code>	波形コンフィギュレーションを保存します。
<code>wcfg select</code>	ウィンドウに表示する波形コンフィギュレーション ファイルを指定します。

Tcl で特殊文字を含むオブジェクト名を使用する方法

`wave add` などの一部のコマンドでは、Tcl で特殊な意味合いを持つ文字を含む引数を使用できます。これらの引数は、波かっこ `{ }` で囲み、Tcl で不本意に処理されないようにします。次に特殊文字を含むケースを示します。

バスのインデックス

角かっこ `[]` は Tcl に特殊の意味合いがあるので、これを使用するインデックス付き (ビットまたは一部の選択) バスは、波かっこで囲む必要があります。たとえば、波形ウィンドウに角かっこを使用して bus エlement 4 を追加するときは、「`wave add {bus[4]}`」と記述する必要があります。

バスのインデックスにはかっこ `()` も使用できます。かっこには特殊な意味合いがないため、波かっこを使用せずにコマンドを記述することができます。

例：

```
wave add bus(4)
```

Verilog エスケープ文字

Verilog で特殊な意味を持つ文字を含む Verilog 識別子は、Verilog ソース コードおよび ISim コマンド ラインの両方で識別子の前にバックスラッシュ (\)、識別子の後にスペースを付けてエスケープ処理する必要があります。

また、Tcl コマンド ラインではエスケープ文字を波かっこで囲う必要があります。たとえば、ワイヤ my_wire を波形ウィンドウに追加する場合、「wave add {\my_wire}」のように、コマンドを記述し、最後の文字と閉じる波かっこ間にスペースを含めます。

注記： Verilog では、どの識別子もエスケープ処理できますが、ISim コマンド ラインでは、エスケープする必要がない Verilog 識別子もエスケープ処理する必要があります。たとえば、波形ウィンドウにワイヤ w を追加する場合、ISim では「wave add {\w}」が有効なコマンドになりません。このため、「wave add w」またはオプションで「wave add {w}」と記述する必要があります。

エスケープ文字に波かっこが含まれる場合、識別子を波かっこで囲う方法が使用できないため、Tcl では波かっこ内に波かっこ内であっても特殊文字として解釈されます。このため、次に示す VHDL 拡張識別子の手法を使用する必要があります。

VHDL 拡張識別子

VHDL 拡張識別子には、バックスラッシュ (\) が含まれており、この文字は Tcl で特殊文字とされます。Tcl では、バックスラッシュと閉じる波かっこ \} が閉じる波かっこ文字として識別されるので、VHDL 拡張識別子は波かっこを使用して記述できません。

このため、波かっこは使用せず、特殊文字それぞれの前にバックスラッシュを付ける必要があります。たとえば、信号 \my_sig を波形ウィンドウにする場合は、「wave add \\my_sig\\」と記述する必要があります。

拡張識別子の一部であるバックスラッシュの両方および識別子内のスペースの前には、バックスラッシュが付けられます。

シミュレーション コマンドの別名表記

よく使用するシミュレーション コマンドに変数を設定すると、すばやくコマンドを実行できます。この変数を使用すると、コマンドを毎回入力せずに何回でも実行できます。コマンドを表現する変数を設定することを、エイリアシングと呼びます。

[Console] パネルでの変数設定

[Console] パネル (GUI モード) または Tcl プロンプト (コマンド ライン モード) で次の変数を入力します。

```
set svc "show value count"
```

説明：

- set : 変数を作成することを示します。
- svc : 変数名です。
- "show value count" : 変数名で表現されるシミュレーション コマンドです。

Tcl 変数 svc が設定されて値が表示されます。

この変数を実行するには、次を入力します。

```
eval $svc
```

ISim 波形ビューアー Tcl コマンド

ISim 波形ビューアー Tcl コマンドは、作業中のウィンドウで使用できます。ISim を起動すると、最初にアクティブになるウィンドウは、Default.wcfg です。

アクティブ ウィンドウは、次の方法で変更できます。

- ウィンドウ タブをクリックするか、`wcfg select` コマンドを使用します。
- [File] → [New] および [File] → [Open] をクリックすると、新しく開いた波形コンフィギュレーション ウィンドウにアクティブ ウィンドウを切り替えることができます。
- Tcl の場合は、`wcfg new` および `wcfg open` コマンドでアクティブ ウィンドウを新規作成のウィンドウに変更できます。

Command Line Conventions

メニュー コマンドを使用する代わりに、ISim コンソール パネルのプロンプトにでコマンドを入力して操作を実行できます。コンソール コマンドでは、メニュー コマンドで表示されるようなダイアログ ボックスは表示されません。

表 9-2 は、シミュレーション コマンドで使用する構文の表記規則を示しています。

表 9-2：シミュレーション コマンドの構文

構文	説明
[]	必須ではないコマンド オプションを示します。 注記： Tcl ではネストしたコマンドに [] を使用できます。[] に入力されたコマンドが実行され、その結果が別の Tcl コマンドで使用する値として戻されます。たとえば、 <code>set var <show time></code> では var が現在の時間に設定されます。
	可能性のあるパラメーターの選択肢を示します。オプションが複数のパラメータで構成されている場合、各パラメータは角かっこで区切られています。
...	1 つまたは複数のオプションをスペースで区切って入力できることを示します。
<>	ユーザーが値を指定する必要がある変数であることを示します。

Tcl コマンド

エンジン コマンド

bp

bp コマンドでは、シミュレーションする HDL ソース コードのブレークポイントを設定/削除します。ブレークポイントは、デバッグのためシミュレーションを中断するのに使用します。

bp コマンドの構文

```
bp add <file_name> <line_number>clear
del <index> [<index>...]list
remove <file_name> <line_number>
```

bp コマンドのオプション

表 9-3 : bp コマンドのオプション

オプション	説明
add <file_name> <line_number>	HDL ファイルの指定行にブレークポイントを追加します。 <file_name> には、ブレークポイントを設定するシミュレーション中の HDL ソース ファイルを指定します。<line_number> には、HDL コードのシミュレーションを中断する行番号を指定します。
clear	読み込んでいるすべての HDL ファイルのブレークポイントをすべて削除します。複数のファイルにブレークポイントが設定されている場合、すべてのブレークポイントが削除されます。
del <index> [<index>...]	HDL コードから指定したブレークポイントを削除します。このコマンドを実行する前に、bp list コマンドを実行してブレークポイントのインデックス番号を取得する必要があります。詳細は、list オプションを参照してください。 <index> は、ブレークポイントに割り当てられたインデックス番号です。デザインの各ブレークポイントに、固有の番号が割り当てられます。 注記： このインデックス番号は、ソース ファイルの行番号とは異なります。

表 9-3 : bp コマンドのオプション (続き)

オプション	説明
list	<p>デザインのブレークポイントをすべてリストし、bp del コマンドで使用するインデックス番号を示します。複数のファイルにブレークポイントが設定されている場合、すべてのブレークポイントが表示されます。bp list を実行すると、次が表示されます。</p> <pre><index> <directory_path> <file_name>/<line_number></pre> <p>説明 :</p> <ul style="list-style-type: none"> • <index> : bp del コマンドで使用するインデックス番号 • <directory_path> is the fully qualified path to your source files. • <file_name> : ブレークポイントを含むソース ファイル名 • <line_number> : ソース ファイルでのブレークポイントが設定されている行番号
remove <file_name> <line_number>	<p>ファイル <file_name> の <line_number> 行目にあるブレークポイントを削除します。</p> <ul style="list-style-type: none"> • <file_name> は、削除するブレークポイントが設定されている HDL ソース ファイルを指定します。 • <line_number> は、HDL コードのブレークポイントが設定されている行番号を指定します。

bp コマンドの例

statmach.vhd というファイルの 2 行目にブレークポイントを設定するには、次のように入力します。

```
bp add statmach.vhd 2
```

全ブレークポイントの表示

シミュレーションに関連するすべてのファイルのブレークポイントをインデックス番号と共に表示するには、次のようにコマンドを入力します。

```
bp list
```

シミュレーションのブレークポイントをすべて削除するには、次のように入力します。

```
bp clear
```

ブレークポイントをインデックス番号を使用して削除するには、次のように入力します。

1. 最初に、bp list コマンドを使用してブレークポイントのインデックスを取得します。

```
bp list
```

次の情報が示されます。

```
1 C:/examples/watchvhd/stopwatch_tb.vhd::46
```

```
2 C:/examples/watchvhd/stopwatch_tb.vhd::55
```

2. stopwatch_tb.vhd ファイルの 46 行目にあるブレークポイントを削除するには、次のように入力します。

```
bp del 1
```

statmach.vhd というファイルの 2 行目のブレークポイントを削除するには、次のように入力します。

```
bp remove statmach.vhd 2
```

describe

describe コマンドを実行すると、HDL データまたはブロック オブジェクトの情報を表示できます。

describe コマンドの構文

```
describe <object_name>
```

<object_name> オプションは、現在のシミュレーション スコープに含まれる HDL オブジェクトまたは HDL ブロックの情報を表示します。

describe コマンドの例

describe コマンドは、次のように使用します。

```
describe param
```

次が戻されます。

```
Verilog Instance:{param}  
Path:{/parameter8_hn/param}  
Location:{/home/test5.v:42}  
Instantiation:{/home/test5.v:37}
```

dump

dump コマンドでは、現在のデザイン階層に含まれるすべての VHDL 信号およびジェネリック、Verilog ワイヤ、サブプログラム以外のレジスタおよびパラメーターの値が表示されます。

デザイン階層をナビゲートするには、scope コマンドを使用します。dump コマンドでは、isim set radix コマンドを使用してデフォルトの基数セットが使用されます。

dump コマンドの構文

```
dump
```

dump コマンドの例

現在のデザイン階層にある信号名とその値すべてを表示するには、次のように入力します。

```
dump
```

help

help コマンドでは、指定した ISim Tcl コマンドの説明、使用方法、および構文を表示します。コマンドを指定しない場合は、help コマンドではすべての ISim Tcl コマンドとその構文がリストされます。

help コマンドの構文

```
help [command_name]
```

help コマンドのオプション

指定したコマンドの説明を表示します。

help コマンドの例

help コマンドは、次のように使用します。

すべての ISim コマンドの説明を表示するには、次のように入力します。

help

bp コマンドの説明を表示するには、次のように入力します。

help bp

isim condition

isim condition コマンドでは、条件付きアクションのリストを追加、削除、または生成できます。条件付きアクションは、VHDL の process 文または Verilog の always 文と同等で、追加されるとシミュレーション中に isim condition 表現に含まれる信号が継続して監視されます。この条件表現は、信号の変化が検出されるたびに評価されます。指定の条件表現に合う場合は、指定のコマンドが実行されます。

- isim condition remove では信号の監視が停止し、
- isim condition list ではアクティブな条件付きアクションのリストがラベルと ID と共に表示されます。

isim condition コマンドの構文

```
isim condition add|remove|list
[<condition expression> <command>]
[-radix <radix_type>]
[-label <label_name>]
[-index <index_name>]
-all
```

isim condition コマンドのオプション

表 9-4 : isim condition コマンドのオプション

オプション	説明
add remove list	条件を追加、削除、または条件のリストを表示します。
<condition expression> <command>	<p><condition expression>: add 関数に関連しており、指定した <command> をいつ実行するかが定義されます。</p> <p>この条件表現に使用された操作には、 != (非等号)、== (等号)、&& (AND) および (OR) が含まれます。</p> <p>表現および演算子の間には、「clk == St1」のようにスペースを入力する必要があります。</p> <p><command>: Tcl コマンドまたはスクリプトで、条件が true のときに実行されます。このコマンドは、かっこ { } で囲まれます。このコマンドには、標準の Tcl コマンドおよびシミュレーション Tcl コマンドを含めることができます。ただし、run、restart、init、および step は含めることはできません。この条件表現に使用される tcl 変数には、かっこ { } の代わりに引用符 " " で囲まれます。</p> <p>構文例を参照してください。</p>

表 9-4 : isim condition コマンドのオプション (続き)

オプション	説明
-radix <radix_type>	条件表現の値を読み出すのに使用されるオプションの引数です。 サポートされる基数タイプは、default、dec、bin、oct、hex、unsigned、および ascii です。基数タイプが指定されていない場合は、isim set radix コマンドで設定されるグローバル基数タイプが使用され、このコマンドでも基数タイプが設定されていない場合は、default が基数として使用されます。
-label <label_name>	条件名を指定するオプションの引数で、sim condition add コマンドでラベルが指定されていないときは、シミュレーションで条件を識別するときに使用するインデックスが生成されます。
-index <index_name>	条件を識別する引数で、isim condition remove コマンドでのみ使用できます。
-all	現在のシミュレーションに含まれる条件をすべて削除するのに使用されるオプションの引数です。

isim condition コマンドの例

isim condition add コマンドの例

信号 asig が 8 のときに停止して、条件の名前に label0 を付けるような条件を追加するには、次のように入力します。

```
isim condition add {/top/asig == 8} {stop} -label label0 -radix hex
```

信号 asig が 1 のときに停止して、条件の名前に label1 を付けるような VHDL 特有の信号条件を追加するには、次のように入力します。

```
isim condition add {/top/asig == '1'} {stop} -label label1
```

信号 asig が 変化するときに停止して、条件の名前に label2 を付けるような条件を追加するには、次のように入力します。

```
isim condition add /top/asig {stop} -label label2
```

信号 clk が St1 のときに停止して、条件の名前に label3 を付けるような VHDL 特有の信号条件を追加するには、次のように入力します。

```
isim condition add {clk == St1} {stop} -label label3
```

信号 asig (3:0) が 0001 でリセットが 1 のときに停止するような条件を追加するには、次のように入力します。

```
isim condition add {asig(3:0) == 0001 && reset == 1} {stop}
```

isim condition remove コマンドの例

現在のシミュレーションに含まれる条件すべてを削除する場合は、次を入力します。

```
isim condition remove
```

または

```
isim condition remove -all
```

label0、label1、label2 という名前の条件を削除するには、次を入力します。


```
isim condition remove -label {label0 label1 label2}
```

信号文を削除するには、次を入力します。

```
isim condition remove -index 2
```

または

```
isim condition remove -label label3
```

isim condition list コマンドの例

デザインに追加した isim condition すべてを表示するには、次を入力します。

```
isim condition list
```

isim force

isim force コマンドでは、VHDL 信号、Verilog ワイヤ、または Verilog レジスタに強制的に値を付けたり、特定の時間に繰り返しパターンを割り当てます。このコマンドで割り当てられた値は、HDL コードまたは以前に isim force コマンドで付けられた値に上書きされます。このコマンドはキャンセル時間が指定されている場合はその時間まで、または isim force remove コマンドが発行されるまで有効です。

- VHDL 信号または Verilog ワイヤでこのコマンドが削除されると、信号またはワイヤの値が現在駆動されている値に戻ります。
- Verilog レジスタでは、このコマンドが削除された後も Verilog レジスタに書き込む HDL プロセスのうちの 1 つでレジスタに新しい値が割り当てられるまで、強制された値が保持されます。

isim condition コマンドの構文

```
isim force add|remove
[<object_name>]
[-value <value>]
[-radix <radix_type>]
[-time <time>]
[-cancel <time>]
[-repeat <time>]
```

isim force コマンドのオプション

表 9-5 : sim force コマンドのオプション

オプション	説明
add remove	バス/信号に値を割り当てるか、またはバス/信号の値を削除します。
<object_name>	VHDL 信号、Verilog ワイヤ、または Verilog レジスタの名前を指定します。
-value <value>	追加する値を指定します。

表 9-5 : sim force コマンドのオプション (続き)

オプション	説明
-radix <radix_type>	基数を指定します。サポートされる基数タイプは、default、dec、bin、oct、hex、unsigned、および ascii です。基数タイプが指定されていない場合は、 <code>isim set radix</code> コマンドで設定されるグローバル基数タイプが使用され、このコマンドでも基数タイプが設定されていない場合は、default が使用されます。
-time <time>	時間は、10、10 ns、"10 ns" などの文字列で指定できます。値が単位なしで入力された場合は、シミュレータの単位である ps が使用されます。時間は、コマンドの実行時間に相対します。
-cancel <time>	特定の時間後に force コマンドをキャンセルします。
-repeat <time>	特定の時間後にサイクルを繰り返します。

isim force コマンドの例

isim force コマンドは、次のように使用します。

値の割り当て

現在のシミュレーション時間で rst 信号に 0 を割り当てるには、次を入力します。

```
isim force rst 0
```

現在のシミュレーション時間から 10ns 後に rst 信号に 1 を割り当て、現在のシミュレーション時間から 50ns 後にコマンドをキャンセルするには、次を入力します。

```
isim force rst 1 -time 10 ns -cancel 50 ns
```

clk 信号が現在のシミュレーション時間で 1 になり 20ns 後に 0 に戻った後、現在のシミュレーション時間から 1us に到達するまで 40ns ごとにこれを繰り返す (つまりはデューティ サイクルが 50% のクロックを生成して 1us 間 40ns ごとにサイクルを繰り返す) ようなクロックを割り当てるには、次を入力します。

```
isim force clk 1 -value 0 -time 20 ns -repeat 40 ns -cancel 1 us
```

次を実行する場合は、その後に記述するコマンドを入力します。

- 現在のシミュレーション時間で data_in 信号に 1 を割り当てる。
- 現在のシミュレーション時間 + 50 ns で data_in 信号に 0 を割り当てる。
- 現在のシミュレーション時間 + 75 ns で data_in 信号を 1 に戻す。
- このパターンを 100ns ごとに 5000ns 間繰り返す。

```
force add data_in 1 -value 0 -time 50 ns -value 1 -time 75 ns -repeat  
100 ns -cancel 5000 ns
```

値の削除

信号 s、s1、s2 のすべて値を削除するには、次を入力します。

```
isim force remove s s1 s2
```

`isim get arraydisplaylength`

`isim get arraydisplaylength` コマンドを使用すると、アレイ型の HDL オブジェクトに対するエレメントの制限数を表示できます。制限数は、`isim set arraydisplaylength` コマンドを使用して設定できます。

`isim get arraydisplaylength` コマンドの構文

```
isim get arraydisplaylength
```

オプションはありません。

`isim get arraydisplaylength` コマンドの例

次のコマンドを入力し、配列の表示の長さを決定します。この場合、配列は 64 ビットです。

```
isim get arraydisplaylength
```

次が戻されます。64

`isim get radix`

`isim get radix` コマンドを使用すると、デフォルトの基数を文字列として表示できます。基数は、`isim set radix` コマンドを使用して設定します。

`isim get radix` コマンドの構文

```
isim get radix
```

オプションはありません。

`isim get radix` コマンドの例

基数を表示するには、次を入力します。

```
isim get radix
```

現在使用されているグローバル基数が表示されます。

`isim get userunit`

`isim get userunit` コマンドを使用すると、単位が指定されていない時間値すべてに対する現在の単位を表示できます。単位は、`isim set userunit` コマンドを使用して設定できます。

`isim get userunit` コマンドの構文

```
isim get userunit (オプションはありません)
```

`isim get userunit` コマンドの例

```
isim get userunit
```

1 ps が戻されます。

`isim ltrace`

`isim ltrace` コマンドを使用すると、行トレースのオン、オフを切り替えることができます。行トレースがオンのときは、デバッグで行ごとに解析を実行できます。実行される HDL 行は、シミュレーション時間、ファイルパスおよびファイル名、および行番号の情報と共に画面に表示されます。

注記: `isim ltrace on` コマンドを実行すると、シミュレーションの速度が低下する可能性があります。

isim ltrace コマンドの構文

```
isim ltrace [on | off]
```

isim ltrace コマンドのオプション

isim ltrace コマンドのオプションは、[on | off] のいずれかになります。デフォルトは off です。

isim ltrace コマンドの例

現在実行されている行を表示するには、次を入力します。

```
isim ltrace on  
run
```

出力には、シミュレーション時間、ファイル名、行番号が次のように表示されます。

```
1005 ns "C:/Data/ISE_Projects/freqm/watchver/stopwatch_tb.v":261005 ns  
"C:/Data/ISE_Projects/freqm/watchver/stopwatch_tb.v":271005 ns(3) "C:/  
Data/ISE_Projects/freqm/watchver/statmach.v":631005 ns(3) "C:/Data/  
ISE_Projects/freqm/watchver/statmach.v":64
```

isim ptrace

isim ptrace コマンドを [Console] パネルに入力すると、バッチ モードのオン、オフを切り替えることができます。このオプションをオンにすると、現在実行されている VHDL または Verilog プロセスの名前が [Console] パネルに表示されます。この機能は、シミュレーションが無限ループでスタックした場合に便利で、シミュレータがスタックしたプロセスがコメントアウトされます。

isim ptrace コマンドの構文

```
isim ptrace [on | off]
```

isim ptrace コマンドのオプション

オプションは、[on | off] のいずれかになります。デフォルトは off です。

isim ptrace コマンドの例

現在実行されているプロセスの名前を表示するには、次のように入力します。

```
isim ptrace on
```

isim set arraydisplaylength

isim set arraydisplaylength コマンドでは、配列型の HDL オブジェクトに対するエレメントの制限数を設定します。デフォルト値は 64 です。この制限数は次の値に影響します。

- GUI の [Objects] パネルの値
- show value コマンドに対する応答

isim set arraydisplaylength コマンドの構文

```
isim set arraydisplaylength <size>
```

<size> は使用可能なオプションです。表示する配列型 HDL オブジェクトのエレメント数を入力します。長さを無制限にするには 0 を設定します。デフォルト値は 64 です。

isim set arraydisplaylength コマンドの例

次を入力します。

```
isim set arraydisplaylength 2
show value xcountout
```

次が戻されます。

```
00
00
```

[Objects] パネルで配列の [Value] 列を確認します。

```
isim set arraydisplaylength 64
show value xcountout
```

次が戻されます。

```
0001000000
0001000000
```

```
isim set radix
```

isim set radix コマンドを使用すると、グローバル基数をシミュレーションに設定できます。この基数タイプは、show value、put、test、dump、isim force、および isim condition コマンドなど、その他のコマンドで使用されます。

isim set radix コマンドの構文

```
isim set radix <radix_type>
```

説明：

<radix_type>: 現在のシミュレーションにグローバルな基数を設定します。シミュレータでは、show value、put、test、dump、isim force、および isim condition コマンドなど、その他のコマンドで使用されます。

サポートされる基数タイプは、default、dec、bin、oct、hex、unsigned、および ascii です。

isim set radix コマンドの例

16 進数の基数を設定し、カウント値を表示するには、次のように入力します。

```
isim set radix hex
show value count
```

a が戻されます。

dec の基数を設定し、カウント値を表示するには、次のように入力します。

```
isim set radix dec
show value count //count is defined as reg[3:0] count
```

-4 が戻されます。

符号なしの基数を設定し、カウント値を表示するには、次のように入力します。

```
isim set radix unsigned
show value count
```

10 が戻されます。

`isim set userunit`

`isim set userunit` コマンドを使用すると、単位が指定されていない時間値すべてに対する現在の単位を設定できます。デフォルトの時間単位は、`fuse` コマンド の `-timescale` または `-override_timeunit` オプションで設定されている単位と同じです。これらの `fuse` オプションが指定されていない場合は、タイムスケールは次で決定します。

- デフォルトの時間単位 (1ps)
- Verilog に含まれる ‘timescale シミュレータ指示子

isim set userunit コマンドの構文

```
isim set userunit <1|10|100> <fs|ps|ns|us|ms|s>
```

例

シミュレータのタイムスケールを 1 ps に設定します。

```
isim set userunit 1 ps
```

`onerror`

`onerror` コマンドを使用すると、エラーが発生した **Tcl** シミュレーション コマンドの直後の動作を制御できます。エラーの表示、次のコマンドの再開などのさまざまな使用例は、次に示す例を参照してください。このコマンドを使用すると、シミュレーション コマンド エラーのデバッグが可能で、エラーが含まれている **Tcl** スクリプトを実行するときに特に便利です。

注記 : **Tcl** プロンプトに **Tcl** コマンドを 1 つずつ入力する場合には、このコマンドを使用する必要はありません。

onerror コマンドの構文

```
onerror [<list_of_Tcl_commands>] [<source Tcl_script>]
```

説明 :

- `<list_of_Tcl_commands>` : シミュレーション **Tcl** コマンドのリストを入力し、シミュレーション コマンド エラーが発生したときの動作を制御できます。
- `<source Tcl_script>` : リストされる **Tcl** コマンドを含む **Tcl** スクリプト ファイルをソースに指定できます。

onerror コマンドの例

発生したエラーと現在の範囲内のすべての値を表示してから、停止します。

```
onerror {showtime;dump;quit -f}
```

エラーが発生した時間および現在の階層に含まれる値すべてを表示した後に **Tcl** スクリプトの次のコマンドの読み出しが継続されます。

```
onerror {show time;dump;resume}
```

エラーが発生したときにソースの **Tcl** ファイルが読み出され、そのコマンドが実行されます。

```
onerror {source myerror.tcl}
```

put

シミュレーション中に、信号およびバスの値を変更できます。put コマンドは、次に使用します。

- 特定の信号またはバス
- 信号またはバスの配列
- 信号またはバスを含むレコードまたはレコードの配列
- 基数が指定されている値

put コマンドは、**HDL** (ハードウェア記述言語) ソース コードで信号として宣言されている信号またはバスにのみ使用できます。

put コマンドを使用して、**VHDL** 変数、**VHDL** ジェネリック、または **Verilog** パラメーターに値を割り当てることはできません。値は、信号全体、信号の 1 ビット、または信号のビット範囲に割り当てることができます。また、信号の階層にもアクセスできます。このコマンドは、上書きされる可能性があり、デザインのステイミュラスがこのコマンドより優先されます。

put コマンドの構文

```
put <signal_name>|<vhdl_process_name>|<process_variable_name> [element  
reference...] <value> [<object> <value>] [-radix <radix_type>]
```

put コマンドのオプション

表 9-6 : put コマンドのオプション

オプション	説明
<code><signal_name> </code> <code><vhdl_process_name> </code> <code><process_variable_name></code> <code>[element reference...]</code> <code><value></code>	<ul style="list-style-type: none"> • <code><signal name></code>: 値を割り当てる信号またはバスの名前を指定します。信号またはバスの配列、信号またはバスを含むレコード配列あるいはバスまたは信号の配列を含むレコードも指定できます。 • <code><vhdl_process_name/process_variable_name></code>: 値を割り当てるプロセスおよびプロセス変数の名前を指定します。プロセス変数に値を割り当てるには、その変数を含むプロセスの名前も指定する必要があります。プロセス名とプロセス変数名は、スラッシュ (/) で区切ります。 • <code>[element reference]</code>: 参照する信号名のサブエレメントを指定します (オプション)。信号の個々のサブエレメントを参照することで信号を詳細に指定できます。詳細は、次の例を参照してください。 <p>信号のタイプに基づいて、次の値を割り当てることができます。</p> <ul style="list-style-type: none"> • <code>integer</code> 型には、正または負の整数を指定できます。 • <code>bit_vector</code> 型には 0 または 1、あるいは 0 と 1 の配列を指定できます。 • VHDL の場合、<code>std_logic</code> 型には U、X、0、1などを指定できます。 • Verilog の場合、<code>bit_values</code> 型には U、X、0、1 を指定できます。 • <code>strength</code> 値はサポートされていません。
<code>[<object> <value>]</code> <code>[-radix <radix_type>]</code>	<p>特定の基数が付いた値をオブジェクトのデータタイプに変換し、オブジェクトに値を書き込みます。</p> <p><code><object></code>: 変更する信号、バス、またはオブジェクトを指定します。</p> <p><code><value></code> : オブジェクトに追加する値を指定します。サポートされる基数タイプは、<code>-radix [radix_types] default, dec, bin, oct, hex, unsigned</code> および <code>ascii</code> です。基数タイプが指定されていない場合は、<code>isim set radix</code> コマンドで設定されるグローバル基数タイプが使用され、このコマンドでも基数タイプが設定されていない場合は、<code>default</code> が基数として使用されます。</p>

put コマンドの例

バスまたは信号への値の割り当て

`clk` という信号に値を割り当てるには、次のように入力します。

```
put clk 1
```

または

```
put clk 1 -radix bin
```

または

```
put clk "1" -radix bin
```

`busx` という 4 ビット バスに値を割り当てるには、次のように入力します。

```
put busx 0101
```

`A` という信号に値 `FF` を割り当てるには、次のように入力します。

```
put A FF -radix hex
```


現在の階層にインスタンス化されているモジュール u1 にある信号 count(6) にビット値 1 を割り当てるには、次のように入力します。

```
put u1/count(6) 1
```

標準ロジック ベクターへの値の割り当て

この後の例は、次のように宣言されている sigx という標準ロジック ベクターを使用しています。

```
signal sigx : std_logic_vector(0 to 5);
```

sigx のビット 0 を 1 に設定するには、次のように入力します。

```
put sigx(0) 1
```

sigx のビット 1 ～ 2 を 11 に設定するには、次のように入力します。

```
put sigx(1:2) 11
```

sigx を 101010 に設定するには、次のように入力します。

```
put sigx 101010
```

オブジェクトの配列への値の割り当て

次は、次のように宣言されている標準ロジック ベクターの配列に値を割り当てています。

```
signal sigarray:(0 to 3) vectorarray(0 to 5, 1 to 4, 2 to 6);
```

sigarray ベクター配列要素の各ビットを 1 に設定するには、次のように入力します。

```
put sigarray(0,1,2) 1111
```

sigarray ベクター配列要素の最初の 2 ビットを 10 に設定するには、次のように入力します。

```
put sigarray(0,1,2) (1:2) 10
```

sigarray ベクター配列要素のビット 3 を 1 に設定するには、次のように入力します。

```
put sigarray(0,1,2) (3) 1
```

次での例では、宣言されている標準ロジック ベクターの配列を含むレコード配列を使用しています。

```
type ram_3d_vector is array(0 to 10, 7 downto 0, 0 to 2) of  
std_logic_vector(1 to 4);
```

```
type rectype is record
```

```
  a: integer;
```

```
  b: string(1 to 7);
```

```
  c: std_logic_vector(0 to 3);
```

```
  d: ram_3d_vector;
```

```
end record;
```

```
type recarray is array(0 to 3, 4 downto 1) of rectype;
```

```
signal recarrsig: recarray;
```

```
signal recsig: rectype;
```

レコード recsig の 2 番目の要素 (b) を文字列 abc に設定するには、次のように入力します。

```
put recsig.b(2:4) abc
```

レコード recsig の 3 次元配列 d の座標 2,3,1 で示される 4 ビット幅のベクターを 0110 に設定するには、次のように入力します。

```
put recsig.d(2,3,1) 0110
```

レコード `recsig` の 3 次元配列 `d` の座標 `2,3,1` で示される 4 ビット幅のベクターの最初の 2 ビットを `01` に設定するには、次のように入力します。

```
put recsig.d(2,3,1) (1:2) 01
```

2 次元配列 `recarrsig` の座標 `2,2` で示されるレコード `recsig` の 3 次元配列 `d` の座標 `2,3,1` で示される 4 ビット幅のベクター値を設定するには、次のように入力します。

```
put recarrsig(2,2).d(2,3,1) 0011
```

`quit`

`quit` コマンドでは、コマンド オプションに従い、シミュレーションまたはツールのいずれかを終了します。オプションがない場合は、グラフィカル ユーザー インターフェイスでは終了を示すプロンプトが表示されてから **ISim** が終了し、コマンド ラインではすぐに **I Sim** が終了します。

quit コマンドの構文

```
quit [-f] [-s]
```

説明 :

- `-f`
現在のシミュレーションを停止して、**ISim** ツールを終了します。波形コンフィギュレーションに変更を加えたときでも保存を尋ねるダイアログ ボックスは表示されません。コマンド ラインでは、ソフトウェアがすぐに終了します。
- `-s`
グラフィカル ユーザー インターフェイスを開いたまま現在のシミュレーションを停止します。この場合、**ISim** では [File] → [Open] で WDB ファイルを開いてスタティック波形データベースを読み込む以外の作業は実行できません。コマンド ラインでは、ソフトウェアがすぐに終了します。

quit コマンドの例

ISim を終了して Tcl プロンプトを閉じます。

```
quit
```

ISim を終了して波形を保存します。

```
quit -f
```

現在のシミュレーションを終了します。

```
quit -s
```

`restart`

シミュレーションを停止してシミュレーション時間を `0` に戻します。これにより、デザインを読み込み直さずにシミュレーションを実行し直すことができます。GUI で [Simulation] → [Restart] を実行しても、この操作を実行できます。

restart コマンドの構文

```
restart [onerror] [scope] [saif] [vcd] [isim force] [put]
```

restart コマンドのオプション

表 9-7 : restart コマンドのオプション

オプション	説明
onerror	onerror スクリプトを削除します。
scope	階層が /top にリセットされます。
saif	ファイルを閉じます。
vcd	ファイルを閉じます。
isim force	force 指定を削除します。
put	初期値に戻ります。

restart コマンドの例

シミュレーション時間を 0 に戻してシミュレーションを開始するには、次を入力します。

```
restart
```

resume

resume コマンドは onerror コマンド と共に使用して、エラーが発生した後にコマンドの実行を継続させます。

注記：このコマンドのみを入力しても効力はありません。

resume コマンドの構文

```
resume
```

このコマンドにはオプションがありません。

resume コマンドの例

エラーが発生した時間および現在の階層に含まれる値すべてを表示した後に Tcl スクリプトの次のコマンドの読み出しが継続されます。

```
onerror {show time;dump;resume}
```

run

run コマンドを実行すると、シミュレーションが開始します。このコマンドをオプションを使用せずに実行すると、シミュレーションが 100ns 間実行されます。GUI でこれを実行するには、次をクリックします。

- [Simulation] → [Run All]
- [Simulation] → [Run]

run コマンドの構文

```
run [all] [continue] [<time> <unit>]
```

run コマンドのオプション

表 9-8 : run コマンドのオプション

オプション	説明
all	イベントがすべて終了するかブレークポイントに達するまで、シミュレーションを実行します。ブレークポイントの設定および削除に関する詳細は、「bp コマンド」を参照してください。
continue	ブレークポイントでシミュレーションが停止した後に、シミュレーションを再開します。このオプションは、run all を実行するのと同じです。
<time> <unit>	<ul style="list-style-type: none"> • <time> では、シミュレーションを実行する時間を指定します。この値には、正の整数または少数を使用できます。 • <unit> は、時間の単位を指定します。使用可能な値は、fs、ps、ns、us、ms、sec のいずれかです。デフォルトでは ps です。

run コマンドの例

run コマンドは、次のように使用します。

イベントがすべて終了するかブレークポイントに達するまで、シミュレーションを実行します。

```
run all
```

シミュレーションを 2000ns 間実行するには、次のように入力します。

```
run 2000 ns
```

シミュレーションを 1.2ns 間実行するには、次のように入力します。

```
run 1.2 ns
```

シミュレーションを 100ns 間実行します。

```
run
```

saif

saif コマンドを使用すると、SAIF (Switching Activity Interchange format) ファイルを生成してポートおよび信号のスイッチング レートを記録できます。詳細は、[第 7 章「消費電力のアクティビティ データの書き出し」](#) を参照してください。

saif コマンドの構文

```
saif open[-scope <path_name>] [-file <file_name>] [-allnets]
close [-level <number_of_levels>]
```

saif コマンドのオプション

表 9-9 : saif コマンドのオプション

オプション	説明
open [-scope <path_name>] [-file <file_name>] [-allnets]	open : 消費電力概算用の SAIF ファイルを生成します。 <ul style="list-style-type: none"> -scope <path_name> : 特定の階層および再帰的階層の消費電力概算データを作成します。相対パス、絶対パスのいずれかを使用できます。パスが指定されていない場合は、現在の階層が使用されます。 -file <file_name> : 新しい SAIF ファイルを生成します。デフォルト名は xpower.saif です。シミュレーション実行中に開くことができる SAIF ファイルは 1 つのみです。 -allnets : 消費概算に内部ネットおよびポート信号が含まれます。このオプションを使用しない場合は、ポート信号の変化だけが監視されます。
close	監視を停止して SAIF ファイルを出力します。
[-level <number_of_levels>]	デザイン階層の階層レベルを必要なだけ含めることができます。 <ul style="list-style-type: none"> -level 0 : 指定した階層下すべてのレベルの信号遷移が監視され、SAIF ファイルに含まれます。 -level 1 : 指定した階層のみの信号遷移が監視され、SAIF ファイルに含まれます。 -level 2 : 指定した階層の 2 つのレベルの信号遷移が監視され、SAIF ファイルに含まれます。

saif コマンドの例

saif コマンドは次のように使用します。

現在の階層にあるデザインのすべてのポートが xpower.saif ファイルに書き込まれます。

```
saif open
```

現在の階層にあるデザインのすべてのポートおよび内部ネットが xpower.saif ファイルに書き込まれます。

```
saif open -allnets
```

UUT にあるデザインのすべてのポートおよび内部ネットが uut_backward.saif ファイルに書き込まれます。

```
saif open -scope uut -file uut_backward.saif -allnets
```

scope

scope コマンドを使用して、デザイン階層を移動します。オプションを使用せずに実行すると、現在のモジュール情報が表示されます。

scope コマンドの構文

```
scope..<path_name>
```

説明 :

- ..
現在のモジュールの 1 つ上位にあるモジュールの情報を表示します。
- <path_name>
モジュール情報を表示するモジュールへのパスを指定します。
相対パス、絶対パスのいずれかを使用できます。

scope コマンドの例

scope コマンドは、次のように使用します。

1 つ上の階層に移動するには、次のように入力します。

```
scope..
```

現在のモジュールにインスタンス化されている UUT というモジュールに移動するには、次のように入力します。

```
scope UUT
```

配線後のネットリストの子インスタンスに scope コマンドを使用するには、次のように入力します。

```
X_IPAD \CLK/PAD (  
.PAD (CLK)  
);
```

\CLK/PAD は拡張された識別子です。

```
scope /testbench/UUT/\CLK/PAD\
```

拡張された識別子の CLK および PAD の前後にバックスラッシュ (\) を付ける必要があります。

sdfanno

sdfanno コマンドでは、SDF (Standard Delay Format) ファイルの VITAL 遅延を、VITAL 準拠の VHDL モデルで作成された VHDL デザインにバックアノテートします。また、このコマンドでは、Verilog モジュールの specify ブロックで指定されたタイミングにバックアノテートすることもできます。

sdfanno コマンドの構文

```
sdfanno -min | -typ | -max | <file_name>  
[<file_name>] [-nowarn] [-noerror] [-root <root_path>]
```

説明 :

-min | -typ | -max | and <file_name> are required. オプションについては、[123 ページの表 9-10](#) を参照してください。

sdfanno コマンドのオプション

表 9-10 : sdfanno コマンドのオプション

オプション	説明
-min -typ -max	-min、-typ、-max のいずれかから遅延オプションを指定する必要があります。 <ul style="list-style-type: none"> • -min: VHDL/Verilog ファイルを最小遅延値でアノテートし、ホールド タイムのタイミング シミュレーションを指定します。 • -typ: VHDL/Verilog ファイルを標準遅延値でアノテートします。 • -max: VHDL/Verilog ファイルを最大遅延値でアノテートし、セットアップ タイムのタイミング シミュレーションを指定します。
<file_name>	遅延情報を含む SDF ファイルの名前を指定します。sdfanno コマンドでは、ファイル名を指定する必要があります。
-nowarn	警告メッセージを非表示にします。
-noerror	エラー メッセージを警告メッセージにします。このオプションを使用すると、SDF バックアノテーションにエラーがあっても、シミュレーションを続行できます。
-root <root_path>	アノテーションを実行するデザイン内の位置を指定します。SDF ファイルで指定されているパスは、<root_path> で指定されたデザイン階層の位置を基準として決定されます。デフォルトでは、デザインの最上位がルートに設定されています。

sdfanno コマンドの例

sdfanno -typ の例

mysubdesign.sdf ファイルの標準遅延をサブモジュール subdesign にアノテートするには、次のように入力します。

```
sdfanno -typ mysubdesign.sdf -root /subdesign
```

design.sdf ファイルの標準遅延を現在のデザインの最上位にアノテートして、エラーまたは警告をすべて無視するには、次のように入力します。

```
sdfanno -typ design.sdf -noerror -nowarn
```

sdfanno -min の例

mysubdesign.sdf ファイルの最小遅延をサブモジュール "subdesign" にアノテートするには、次のように入力します。

```
sdfanno -min mysydesign.sdf -root /subdesign
```

sdfanno -max の例

design.sdf ファイルの最大遅延を現在のデザインの最上位にアノテートするには、次のように入力します。

```
sdfanno -max design.sdf -nowarn
```

show

show コマンドでは、デザインの選択した部分を表示します。

show コマンドの構文

```
show [child | child -r] [constant] [driver <signal_name>]
[load <signal_name>] [port] [scope] [signal] [time]
value [<generic_name> | <parameter_name> |
<process_name>/<process_variable_name>] [<signal_name>]
[element references...]<object> [-radix <radix_type>] variable
```

show コマンドのオプション

表 9-11 : show コマンドのオプション

オプション	説明
child child -r	child では現在のブロックの子ブロック (1 レベルのみ) が表示され、child -r では現在のブロックの子プロセスを含む、すべての子ブロックを再帰的にリストします。
constant	現在のブロックに含まれている定数、ジェネリック、およびパラメーターをリストします。
driver <signal_name>	<signal_name> で指定された信号を駆動するプロセスを表示します。 可能な場合、そのドライバーを記述する HDL コードの行番号も表示します。
load <signal_name>	<signal_name> で指定された信号のロードをすべて表示します。
port	現在のブロック内にあるポート信号を表示します。信号が入力であるか出力であるかが示されます。
scope	デザイン階層の現在の位置を表示します。階層での位置を表示するだけで、変更はできません。scope コマンドを <path_name> オプションなしで実行した場合と同じです。
signal	ポート信号も含む、現在のモジュール内にある信号を表示します。
time	シミュレータの現在の時間を表示します。
value [<generic_name> <parameter_name> <process_name>/<process_variable_name>]	<ul style="list-style-type: none"> • <generic_name> : コマンド実行の対象となる VHDL ジェネリックの名前を指定します。 • <parameter_name> : コマンド実行の対象となる VHDL パラメーターの名前を入力します。 • <process_name/process_variable_name> : コマンド実行の対象となるプロセスおよびプロセス変数の名前を指定します。プロセス変数の値を表示するには、その変数を含むプロセスの名前も指定する必要があります。プロセス名とプロセス変数名は、スラッシュ (/) で区切ります。

表 9-11 : show コマンドのオプション (続き)

オプション	説明
<code>[<signal_name></code> <code>[element references...]</code>	<ul style="list-style-type: none"> • <code><signal_name></code> は、コマンド実行の対象となる信号の名前で、信号またはバスの配列、信号またはバスを含むレコード配列あるいはバスまたは信号の配列を含むレコードも指定できます。 • レコードの階層を区切るには、次のようにピリオド (.) を使用します。 <code>show value recsig.c</code> • 下位階層の信号の値を表示するには、次のようにスラッシュ (/) を使用して信号の階層名を区切ります。 <code>show value mymod/mysig [element reference]</code> : 参照する信号名のサブエレメントを指定します。 <p>信号の個々のサブエレメントを参照することで信号を詳細に指定できます。次の例を参照してください。</p> <ul style="list-style-type: none"> • <code>show value (3:0)</code> のように <code>value</code> の後にかっこで囲み、コロンで区切った 2 つの整数を入力すると、<code><signal_name></code> で指定されているベクターの値が表示されます。 • <code>show value (2,3)</code> のように、<code>value</code> の後にかっこで囲み、コンマで区切った整数を入力すると、<code><signal_name></code> で指定されている多次元配列のエレメントの値が表示されます。
<code><object></code> <code>[-radix <radix_type>]</code>	<ul style="list-style-type: none"> • <code><object></code> <code>[-radix <radix_type>]</code> : 特定の基数と共に値が表示されます。 <code><object></code> : HDL オブジェクト データ タイプを設定します。 • サポートされる <code><radix_type></code> : <code>default</code>, <code>dec</code>, <code>bin</code>, <code>oct</code>, <code>hex</code>, <code>unsigned</code>, and <code>ascii</code> <p>基数タイプが指定されていない場合は、<code>isim set radix</code> コマンドで設定されるグローバル基数タイプが使用され、このコマンドでも基数タイプが設定されていない場合は、<code>default</code> が使用されます。</p>
<code>variable</code>	<p>現在のブロックにある変数すべてを表示します。VHDL プロセス内の変数を表示するには、<code>scope</code> コマンドを使用して VHDL プロセスまでナビゲートしてから <code>show variable</code> を実行します。</p>

show コマンドの例

show child

`fifo_controller` というデザインの最上位階層から「`show child`」を実行すると、次の階層情報が表示されます。

```
Block Name:<fifo_controller>
```

「`show child -r`」と入力すると、現在の階層および再帰的階層の情報が表示されます。

show driver

`fifo_count` というデザインの最上位階層から「`show driver fifocount`」を実行すると、`fifocount` 信号に関する次の情報が表示されます。

```
<Driver for fifocount>
fifocount_cc_v2.v:221
```

2 行目の最後の数値 221 は、ソース ファイルでの行番号を示します。

show load

fifo_count というデザインの最上位階層から「show load fifocount」を実行すると、fifocount 信号に関する次の情報が表示されます。

```
<Load for fifocount>
Signal <Hex(0)> (Block: fifo_count/Lsbled/)
Signal <Hex(1)> (Block: fifo_count/Lsbled/)
Signal <Hex(2)> (Block: fifo_count/Lsbled/)
Signal <Hex(3)> (Block: fifo_count/Lsbled/)
```

show scope

fifo_count というデザインの最上位階層から「show scope」を実行すると、次の情報が表示されます。

```
<Block> /tb_cc_func/
```

show value (信号)

clk という信号の値を表示するには、次のように入力します。

```
show value clk
```

busx という 4 ビット バスの値を表示するには、次のように入力します。

```
show value busx
```

addr の値を表示するには、次のように入力します。

```
show value addr
• 「0111010101011101」が表示されます。
show value addr -radix hex
• 「755D」が表示されます。
show value addr -radix dec
• 「30045」が表示されます。
```

show value (オブジェクト)

この後の例は、次のように宣言されている sigx という標準ロジック ベクターを使用しています。

```
signal sigx : std_logic_vector(0 to 5);
```

sig のビット 0 の値を表示するには、次のように入力します。

```
show value sigx(0)
```

sigx のスライス 1 ~ 2 の値を表示するには、次のように入力します。

```
show value sigx(1:2)
```

sigx のすべての値を表示するには、次のように入力します。

```
show value sigx
```

オブジェクトの配列の値の表示

この後の例は、次のように宣言されている標準ロジック ベクターの配列を使用しています。

```
signal sigarray: vectorarray(0 to 5, 1 to 4, 2 to 6);
```

sigarray のベクタ配列要素のすべてのビット値を表示するには、次のように入力します。

```
show value sigarray(0,1,2)
```

sigarray の各ベクター配列要素の最初の 2 ビット値を表示するには、次のように入力します。

```
show value sigarray(0,1,2) (1:2)
```

sigarray の各ベクター配列要素のビット 3 の値を表示するには、次のように入力します。

```
show value sigarray(0,1,2) (3)
```

この後の例は、次のように宣言されている標準ロジック ベクターの配列を含むレコード配列を使用しています。

```
type ram_3d_vector is array(0 to 10, 7 downto 0, 0 to 2) of  
std_logic_vector(1 to 4);
```

```
type rectype is record
```

```
  a: integer;
```

```
  b: string(1 to 7);
```

```
  c: std_logic_vector(0 to 3);
```

```
  d: ram_3d_vector;
```

```
end record;
```

```
type recarray is array(0 to 3, 4 downto 1) of rectype;
```

```
signal recarrsig: recarray;
```

```
signal recsig: rectype;
```

レコード recsig の 2 番目の要素 (b) の値を表示するには、次のように入力します。

```
show value recsig.b(2:4)
```

レコード recsig の 3 次元配列 d の座標 2,3,1 で示される 4 ビット幅のベクターの値を表示するには、次のように入力します。

```
show value recsig.d(2,3,1)
```

レコード recsig の 3 次元配列 d の座標 2,3,1 で示される 4 ビット幅のベクターの最初の 2 ビットの値を表示するには、次のように入力します。

```
show value recsig.d(2,3,1) (1:2)
```

2 次元配列 recarrsig の座標 2,2 で示されるレコード recsig の 3 次元配列 d の座標 2,3,1 で示される 4 ビット幅のベクターの値を表示するには、次のように入力します。

```
show value recarrsig(2,2).d(2,3,1)
```

```
step
```

1 回目のシミュレーションを実行した後、HDL デザインのソース コードを 1 行ずつ実行して、デザインが予期どおりに動作するかを検証できます。

step

このコマンドを使用すると、HDL ファイル (Verilog または VHDL) の実行コードの次の行までシミュレーションが進みます。GUI で [Simulation] → [Step] を実行しても、この操作を実行できます。

step コマンドの構文

```
step
```

オプションはありません。

step コマンドの例

HDL ソース コードを 1 行ずつ実行するには、次のように入力します。

```
step
```

test

test コマンドでは、VHDL 信号、Verilog ワイヤ、Verilog レジスタ、VHDL ジェネリック、Verilog パラメーター、または VHDL の process 変数の実数と入力した値を比較します。

この 2 つの値が一致している場合は何も表示されませんが、一致していない場合は、正しい値が表示され、ISim でエラーがレポートされます。このテストは、ベクター エレメントの 1 ビットまたは全体値に実行できます。

test コマンドの構文

```
test <signal_name>|<vhdl_process_name /<process_variable_name>| [element  
reference...]<value>  
<object> <value> [-radix <radix_type>]
```

test コマンドのオプション

表 9-12 : test コマンドのオプション

オプション	説明
<pre><signal_name>/ <vhdl_process_name / <process_variable_name > [element reference...] <value></pre>	<ul style="list-style-type: none"> • <signal_name> は、比較対象となる信号の名前で、信号またはバスの配列、信号またはバスを含むレコード配列あるいはバスまたは信号の配列を含むレコードも指定できます。 • <vhdl_process_name/process_variable_name> : 比較するプロセスおよびプロセス変数の名前を指定します。プロセス変数の値を比較するには、その変数を含むプロセスの名前も指定する必要があります。プロセス名とプロセス変数名は、スラッシュ (/) で区切ります。 • {element reference} : 参照する信号名のサブエレメントを指定します。信号の個々のサブエレメントを参照することで信号を詳細に指定できます。 • <value> : 信号またはバスの実際の値と比較する値を入力します。
<pre><object> <value> [-radix <radix_type>]</pre>	<p>特定の基数が付いた値とオブジェクトの値を比較します。</p> <ul style="list-style-type: none"> • <object> : テストする信号、バス、またはオブジェクトを指定します。 • <value> : オブジェクトに追加する値を指定します。サポートされる基数タイプは、default、dec、bin、oct、hex、unsigned、および ascii です。基数タイプが指定されていない場合は、isim set radix コマンドで設定されるグローバル基数タイプが使用され、このコマンドでも基数タイプが設定されていない場合は、default が使用されます。

test コマンドの例

test コマンドは、次のように使用します。

現在の階層にインスタンス化されているモジュール u1 にある信号 count (6) が 1 であるかを調べるには、次のように入力します。

```
test u1/count (6) 1
```

信号 A の値と FF を比較するには、次のように入力します。

```
test A FF -radix hex
```

値と clk の値を比較するには、次のように入力します。

```
test clk "U"
```

次が戻されます。1

値と clk の値を比較するには、次のように入力します。

```
test clk "0"
```

次が戻されます。0

/top/rst が 0 の場合にシミュレーションを停止するには、次のように入力します。

```
if {[test /top/rst 0]} {stop} else...
```

UUT というブロックの Reset 信号を比較するために、「test Reset 1」と入力すると、次のメッセージが表示されます。

```
test failed Command failed: test Reset 1 1 Net Reset has value 0 not 1
as expected.
```

UUT というブロックのバス Lsbcnt を比較するために、「test Lsbcnt 1001」と入力すると、次のメッセージが表示されます。

```
test passed 0
```

この後の例は、次のように宣言されている sigx という標準ロジック ベクターを使用しています。

```
signal sigx : std_logic_vector(0 to 5);
```

sigx のビット 0 を 1 と比較するには、次のように入力します。

```
test sigx(0) 1
```

sigx のビット 1 ～ 2 の値が 11 であるかを調べるには、次のように入力します。

```
test sigx(1:2) 11
```

sigx のすべてを 101010 と比較するには、次のように入力します。

```
test sigx 101010
```

この後の例は、次のように宣言されている標準ロジック ベクターの配列を使用しています。

```
signal sigarray: vectorarray(0 to 5, 1 to 4, 2 to 6);
```

sigarray のベクター配列要素のすべてのビットが 1 であるかを調べるには、次のように入力します。

```
test sigarray(0,1,2) 1111
```

sigarray の各ベクター配列要素の最初の 2 ビットが 10 であるかを調べるには、次のように入力します。

```
test sigarray(0,1,2) (1:2) 10
```

sigarray の各ベクター配列要素のビット 3 が 1 であるかを調べるには、次のように入力します。

```
test sigarray(0,1,2) (3) 1
```

この後の例は、次のように宣言されている標準ロジック ベクターの配列を含むレコード配列を使用しています。

```
type ram_3d_vector is array(0 to 10, 7 downto 0, 0 to 2) of
std_logic_vector(1 to 4);
type rectype is record
a: integer;
b: string(1 to 7);
c: std_logic_vector(0 to 3);
d: ram_3d_vector;
end record;
type recarray is array(0 to 3, 4 downto 1) of rectype;
signal recarrsig: recarray;
signal recsig: rectype;
```

レコード recsig の 2 番目の要素 (b) が文字列 abc であるかを調べるには、次のように入力します。

```
test recsig.b(2:4) abc
```

レコード recsig の 3 次元配列 d の座標 2,3,1 で示される 4 ビット幅のベクターが 0110 であるかを調べるには、次のように入力します。

```
test recsig.d(2,3,1) 0110
```

レコード `recsig` の 3 次元配列 `d` の座標 `2,3,1` で示される 4 ビット幅のベクターの最初の 2 ビットが `01` であるかを調べるには、次のように入力します。

```
test recsig.d(2,3,1) (1:2) 01
```

2 次元配列 `recarrsig` の座標 `2,2` で示されるレコード `recsig` の 3 次元配列 `d` の座標 `2,3,1` で示される 4 ビット幅のベクターが `0011` であるかを調べるには、次のように入力します。

```
test recarrsig(2,2).d(2,3,1) 0011
```

vcd

`vcd` コマンドでは、シミュレーション結果を **VCD (Value Change Dump)** フォーマットで生成します。このコマンドを使用すると、次を実行できます。

- VCD ファイルへの指定インスタンスの書き出し
- VCD ファイルの命名
- 記述プロセスの開始および停止

その他の関数詳細は、[第 7 章「消費電力のアクティビティ データの書き出し」](#)を参照してください。

vcd コマンドの構文

```
vcd [dumpfile <file_name>] [dumpvars -m <module_name> [-l <level>]]
[dumpoff] [dumpon] [dumpall] [dumplimit <file_size>] [dumpflush]
```

vcd コマンドのオプション

表 9-13: vcd コマンドのオプション

オプション	説明
<code>dumpfile <file_name></code>	VCD ファイルの名前を指定します。デフォルト名は <code>dump.vcd</code> です。Verilog の <code>\$dumpfile</code> 関数を呼び出します。
<code>dumpvars -m <module_name> [-l <level>]</code>	指定の変数およびその値を VCD ファイルに書き出します。 <code>-m <module_name></code> : モジュール名を出力します。 <code>-l <level> {0 1}</code> <ul style="list-style-type: none"> • 0: 特定モジュールおよびそのモジュールの下位にあるすべてのモジュールのインスタンスに含まれる変数をすべて出力します。引数 0 は、モジュール インスタンスを指定する後続の引数にのみに適用され、個々の変数には適用されません。 • 1: <code>-m</code> で指定されたモジュール内の変数すべてを出力します。ただし、このモジュールでインスタンス化されたモジュールに含まれる変数は出力されません。 Verilog の <code>\$dumpfile</code> 関数を呼び出します。
<code>dumpoff</code>	書き出しプロセスを一時的に中断し、選択された変数をすべて X 値として書き出します。Verilog の <code>\$dumpoff</code> 関数を呼び出します。
<code>dumpon</code>	<code>dumpoff</code> オプションで中断した書き出しプロセスを再開し、 <code>dumpon</code> を呼び出した時に選択されているすべての値を書き出します。Verilog の <code>\$dumpon</code> 関数を呼び出します。
<code>dumpall</code>	選択したすべての変数の現在の値を書き出すチェックポイントを VCD ファイルに作成します。Verilog の <code>\$dumpall</code> 関数を呼び出します。

表 9-13 : vcd コマンドのオプション (続き)

オプション	説明
dumplimit <file_size>	VCD ファイルのサイズを制限します。<file_size> には VCD ファイルの最大サイズをバイトで指定します。VCD ファイルのサイズがこの最大値に達すると、書き出しプロセスが停止し、最大値に達したことを示すコメントが VCD ファイルに記述されます。Verilog の \$dumplimit 関数を呼び出します。
dumpflush	OS の VCD ファイル バッファを空にし、バッファのすべてのデータが確実に VCD ファイルに保存されるようにします。この処理が終了すると、書き出しプロセスが再開し、値が失われることはありません。Verilog の \$dumpflush 関数を呼び出します。

vcd コマンドの例

vcd コマンドは、次のように使用します。

シミュレーションを 1000 ns 間実行した後、モジュール UUT の VCD シミュレーション値を VCD ファイルに書き出すには、次のコマンドを使用します。

書き出すファイルを指定するには、次のように入力します。

```
vcd dumpfile adder.vcd
```

書き出すモジュール ネットを指定するには、次のように入力します。

```
vcd dumpvars -m /UUT
```

シミュレーション時間を指定してシミュレーションを実行するには、次のように入力します。

```
run 1000 ns
```

VCD ファイルにデータを記述するには、次のように入力します。

```
vcd dumpflush
```

wave log

wave log コマンドでは、HDL オブジェクトのシミュレーション出力を波形データベース (wdb) ファイルに記録します。VHDL 信号、Verilog ワイヤ、および Verilog レジスタ型を記録できます。VHDL 変数は、記録できません。

wave log コマンドの構文

```
wave log [-r] [<object_name>]
```

説明 :

- -r
指定ブロックの子モジュールすべてを再帰的に追加します。
- <object_name>
波形データベースにシミュレーション出力を記録する HDL オブジェクトを指定します。<object_name> には、ブロックの階層インスタンス名 (/tb/UUT など) も指定可能で、この場合はブロックに含まれるすべての HDL オブジェクトが記録されます。
アスタリスク (*) などのワイルドカードは使用できません。
ブロックのインスタンス内すべての HDL オブジェクトを追加するには、ブロックのインスタンス名を使用できます。たとえば wave add /UUT は、アスタリスクがサポートされていると仮定した場合、wave add /UUT/* と同じです。

wave log コマンドの例

モジュール インスタンス /tb/UUT および /tb/child/gt に関連付けられている信号を波形データベースに記録するには、次のように入力します。

```
wave log /tb/UUT /tb/child/gt
```

デザインの信号をすべて記録するには、次を入力します。

```
wave log -r /
```

波形ウィンドウ コマンド

wcfg new

wcfg new コマンドでは、新しい波形コンフィギュレーションを作成し、新しいウィンドウに表示します。

wcfg new コマンドの構文

```
wcfg new
```

wcfg new コマンドの例

新しい波形コンフィギュレーションを作成します。

```
wcfg new
```

wcfg open

wcfg open コマンドでは、波形コンフィギュレーションを新しいウィンドウに開きます。

wcfg open コマンドの構文

```
wcfg open <filename>
```

<filename> には、開く WCFG ファイル名を指定します。

wcfg open コマンドの例

WCFG ファイルの名前 (toplevel.wcfg) を指定します。

```
wcfg open toplevel.wcfg
```

wcfg save

wcfg save コマンドでは、作業中の波形コンフィギュレーションをファイルに保存します。

wcfg save コマンドの構文

```
wcfg save <filename>
```

wcfg save コマンドのオプション

<filename> には、開く WCFG ファイル名を指定します。

wcfg save コマンドの例

toplevel.wcfg という名前の WCFG ファイルに波形コンフィギュレーションを保存するには、次を入力します。

```
wcfg save toplevel.wcfg
```

wcfg select

wcfg select コマンドでは、指定の波形コンフィギュレーションをアクティブ ウィンドウにします。

wcfg select コマンドの構文

```
wcfg select <wave_config_name>
```

wcfg select コマンドのオプション

wcfg select コマンドのオプション <wave_config_name> には、アクティブにする波形コンフィギュレーション名を指定します。<wave_config_name> にすでに開いている波形コンフィギュレーションを指定しない場合は、エラー メッセージが表示されます。

wcfg select コマンドの例

design という波形コンフィギュレーションをアクティブにするには、次を入力します。

```
wcfg select design
```

wave add

wave add コマンドでは、ISim グラフィカル ユーザー インターフェイスに表示されている作業中の波形コンフィギュレーションに HDL オブジェクトを追加し、HDL オブジェクトのシミュレーション出力を波形データベース (wdb) ファイルに記録します。波形データベース ファイルの名前はデフォルトで isim.wdb で、-wdb オプションを使用すると変更できます。波形コンフィギュレーションは、波形ウィンドウに表示されます。

GUI の場合は、波形ウィンドウで右クリックし、[Add to Wave Window] をクリックします。

wave add コマンドの構文

```
wave add [-into <ID>] [-reverse] [-radix <radix>] [-color <color>] [-name  
         <custom_name>] [-r] [<object_name>]
```

wave add コマンドのオプション

表 9-14 : wave add コマンドのオプション

オプション	説明
-into <ID>	オブジェクトを追加するグループ オブジェクトの ID または仮想バス オブジェクトの ID を指定します。
-reverse	バス ビット順を反転します (MSB から LSB、またはその逆)。
-radix <radix>	信号の値を表示するときの基数を指定します。<radix> の値には、default (デフォルト)、bin (2 進数)、oct (8 進数)、hex (16 進数)、dec (10 進数)、unsigned (符号なし)、または ascii を使用できます。

表 9-14 : wave add コマンドのオプション (続き)

オプション	説明
-color <color>	<p><color> で指定し色にシミュレーションオブジェクトを設定します。値は、RGB フォーマットで定義します。</p> <p>たとえば、青色なら #0000FF、赤色なら #FF0000、緑色なら #00FF00 を指定します。</p> <p>一部のよく使用される色では、色名をテキストでも入力できます。次の色をテキスト入力できます。black, red, darkred, green, darkgreen, blue, darkblue, cyan, darkcyan, magenta, darkmagenta, darkyellow, gray, darkgray, lightgray. これらの色の RGB 値は、RGB 表で定義されています。</p>
-name <custom_name>	波形オブジェクトにカスタム名を付けます。
-r	各ブロックに関連するオブジェクト (最下位の階層のものまで) を波形に追加します。このオプションを指定しない場合は、<object_name> で入力されているブロックの最初のレベルのオブジェクトが追加されます。
<object_name>	<p>波形データベースにシミュレーション出力を記録する HDL オブジェクトを指定します。</p> <p><object_name> には、ブロックの階層インスタンス名 (/tb/UUT など) も指定可能で、この場合はブロックに含まれるすべての HDL オブジェクトが記録されます。このオプションには、ワイルドカード文字は使用できません。すべての HDL オブジェクトをブロックのインスタンス内に追加するには、そのブロック インスタンスの名前を使用します。</p>

wave add コマンドの例

wave add コマンドは次のように使用します。

オブジェクト UUT に関連する信号を現在の波形コンフィギュレーションに追加するには、次を入力します。

```
wave add /tb/UUT
```

最上位信号をすべて追加するには、次を入力します。

```
wave add /
```

デザインに含まれている RGB 値が #00FF10 の信号をすべて追加するには、次を入力します。

```
wave add -r / -color #00FF10
```

基数が 16 進数で赤色の信号を追加するには、次を入力します。

```
wave add /tb/clk /tb/UUT/data -radix hex -color red
```

divider add

divider add コマンドでは、仕切りを追加します。

divider add コマンドの構文

```
divider add [-into <ID>] [-color <color>]
```

説明：

- -into <ID>
仕切りを追加するグループのオブジェクト ID を指定します。
- -color <color>
<color> で指定し色に仕切りの色を設定します。値は、RGB フォーマットで定義します。
たとえば、青色なら #0000FF、赤色なら #FF0000、緑色なら #00FF00 を指定します。

一部のよく使用される色では、色名をテキストでも入力できます。black、red、darkred、green、darkgreen、blue、darkblue、cyan、darkcyan、magenta、darkmagenta、darkyellow、gray、darkgray、lightgray などの色をテキスト入力できます。これらの色の RGB 値は、RGB 表で定義されています。

divider add コマンドの例

Inputs という名前の仕切りを作業中の波形コンフィギュレーションに追加するには、次のように入力します。

```
divider add Inputs
```

Outputs という名前の赤色の仕切りを追加するには、次のように入力します。

```
divider add Outputs -color red
```

グループに仕切りを追加するには、次を入力します。

```
set test_group_id [group add test_group]
wave add "dcm_clk_s" /tb/data2 -into $test_group_id
divider add data -color blue -into $test_group_id
wave add "addr1" /tb/UUT/addr2 -into $test_group_id
divider add address -color red -into $test_group_id
```

group add

group add コマンドでは、グループを追加します。

group add コマンドの構文

```
group add [-into <ID>]
```

説明：

- -into <ID>

新しいグループを追加する既存グループのオブジェクト ID を指定します。

group add コマンドの例

Inputs という名前のグループを作業中の波形コンフィギュレーションに追加するには、次のように入力します。

```
group add Inputs
```

シミュレーション オブジェクトを追加するグループ dcm_clk_s をグループに追加するには、次を入力します。

```
set test_group_id [group add test_group]
wave add "dcm_clk_s" -into $test_group_id
```

グループ内にグループを作成するには、次を入力します。

```
set group_id [group add test_group]
set group_id_1 [group add group_1 ?into $group_id]
set group_id_2 [group add group_2 ?into $group_id]
wave add clk read_ok -into $group_id_1
wave add data_w -into $group_id_2
```

virtualbus add

virtualbus add コマンドでは、現在作業中の波形コンフィギュレーションに仮想バスを追加します。バスは空の状態で作成されます。その後に、このバスに任意の HDL オブジェクトを追加できます。

virtualbus add コマンドの構文

```
virtualbus add <name> [-into <ID>] [-reverse] [-radix <radix>]
[-color <color>]
```

virtualbus add コマンドのオプション

オプション	説明
<name>	仮想バスの名前を指定します。
-into <ID>	新しい仮想バスを追加する既存の仮想バスのオブジェクト ID を指定します。
-reverse	バスの順序を反転します。
-radix <radix>	信号の値を表示するときの基数を指定します。 <radix> の値には、bin (2 進数)、oct (8 進数)、hex (16 進数)、signed (符号付き)、dec (10 進数)、または ascii を使用できます。
-color <color>	仮想バスの色を指定どおりに設定します。値は、RGB フォーマットで定義します。たとえば、青色なら #0000FF、赤色なら #FF0000、緑色なら #00FF00 を指定します。 一部のよく使用される色では、色名をテキストでも入力できます。black、red、darkred、green、darkgreen、blue、darkblue、cyan、darkcyan、magenta、darkmagenta、darkyellow、gray、darkgray、lightgray などの色をテキスト入力できます。これらの色の RGB 値は、RGB 表で定義されています。

virtualbus add コマンドの例

基数が 16 進数で名前が mybus という仮想バスを作業中の波形に追加するには、次を入力します。

```
virtualbus add <mybus> -radix hex
```

仮想バスを作成して 2 個のシミュレーション オブジェクト sigA および sigB, を追加するには、次のように入力します。

```
set vbusId [virtualbus add mybus -radix hex]
wave add sigA -into $vbusId
wave add sigB -into $vbusId
```

marker add

marker add コマンドでは、マーカーを追加します。

marker add コマンドの構文

```
marker add <time>
```

<time>: 新しいマーカーを追加する時間の位置を指定します。時間の単位が指定されていない場合、isim get userunit コマンドで取得されるデフォルトのユーザー時間単位が使用されます。

marker add コマンドの例

作業中のコンフィギュレーションの 10ns にマーカーを追加するには、次を入力します。

```
marker add 10 ns
```

ライブラリ マップ ファイル (xilinxim.ini)

ISim HDL コンパイル プログラム vhpcomp、vlogcomp、および fuse では、xilinxim.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 の論理ライブラリの定義および物理ロケーションが識別されます。

コンパイラは、次のリストしたディレクトリ順に xilinxim.ini ファイルを検索します。

1. \$XILINX/vhdl/hdp/<platform>
2. -initfile オプションで指定したユーザー ファイル-initfile オプションが指定されていない場合は、作業中のディレクトリに含まれる xilinxim.ini ファイルが検索されます。

xilinxim.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>
<logical_libraryn> = <physical_dir_pathn>
```

次に、xilinxim.ini ファイルの例を示します。

```
VHDL
std=C:/libs/vhdl/hdp/
stdieee=C:/libs/vhdl/hdp/ieee
work=C:/workVerilog
unisims_ver=$XILINX/rtf/verilog/hdp/nt/unisims_ver
xilinxcorelib_ver=C:/libs/verilog/hdp/nt/xilinxcorelib_ver
mylib=./mylib
work=C:/work
```

xilinxim.ini ファイルでは、次の点に注意してください。

- xilinxim.ini ファイルで指定するライブラリ/パスは、1 行に 1 つずつ記述する必要があります。
- 物理パスに該当するディレクトリがない場合は、コンパイラで書き込みが行われるときに vhpcomp または vlogcomp によってディレクトリが作成されます。
- 物理パスは、環境変数を使用して記述できます。環境変数は、\$ で始める必要があります。
- 論理ライブラリのデフォルトの物理ディレクトリは次のとおりです。
isim/<logical_library_name>.
- ファイルのコメントは、-- で開始する必要があります。

付録 B

VHDL および Verilog 言語サポートの例外

VHDL 言語サポートの例外

ISim では、次がサポートされます。

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

例外については、次の表の [例外] の列を参照してください。

表 B-1 : VHDL 言語サポートの例外

サポートされる VHDL 構文	例外
abstract_literal	基底付きリテラルで表現されている浮動小数点値はサポートされません。
aggregate	aggregate 内で choice を混合すること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、サポートされていません。特に次のものは、サポートされません。 <ul style="list-style-type: none"> • エイリアスのエイリアス • subtype_indication のないエイリアス宣言 • エイリアス宣言でのシグネチャ • 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 • alias_designator としての文字列リテラル
alias_designator	<ul style="list-style-type: none"> • alias_designator としての operator_symbol • alias_designator としての character_literal
association_element	結合エレメント内のアクチュアルのスライスに、グローバル/ローカルのスタティック範囲を使用できます。
attribute_name	接頭辞の後の signature はサポートされません。
binding_indication	entity_aspect を使用せずに使用することはサポートされていません。
bit_string_literal	空の bit_string_literal (" ") はサポートされません。
block_statement	guard_expression はサポートされません。たとえばガード付きブロック、ガード付き信号、ガード付きターゲット、およびガード付き割り当てはサポートされていません。

表 B-1 : VHDL 言語サポートの例外 (続き)

サポートされる VHDL 構文	例外
choice	case 文で choice として aggregate を使用することはサポートされません。
concurrent_assertion_statement	postponed はサポートされません。
concurrent_signal_assignment_statement	postponed はサポートされません。
concurrent_statement	wait 文が含まれる同時処理プロシージャ呼び出しはサポートされません。
conditional_signal_assignment	ガード付き信号代入はサポートされないの、オプションの一部としてキーワード guarded を使用することはサポートされません。
configuration_declaration	コンフィギュレーションに使用する generate インデックスは、非ローカル スタティックにできません。
entity_class	リテラル、ユニット、ファイル、およびグループを entity_class として使用することはサポートされません。
entity_class_entry	グループ テンプレートで使用することを目的としたオプションの <> はサポートされません。
file_logical_name	文字列値を評価するワイルドカードも許容されていますが、ファイル名としては文字列リテラルと識別子しか使用できません。
function_call	名前付き引数では、分割、指標付け、または選択は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のインスタンシエーションはサポートされていません。
mode	リンケージ ポートおよびバッファポートの一部はサポートされていません。
options	guarded はサポートされません。
primary	primary を使用した場所では、allocator は展開されます。
procedure_call	名前付き引数では、分割、指標付け、または選択はサポートされません。
process_statement	postponed はサポートされません。
selected_signal_assignment	ガード付き信号代入はサポートされないの、オプションの一部としてキーワード guarded を使用することは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 はガード付き信号の宣言に使用されますが、ガード付き信号はサポートされません。
subtype_indicationd	結合の resolved サブタイプ (配列およびレコード) はサポートされません。

表 B-1 : VHDL 言語サポートの例外 (続き)

サポートされる VHDL 構文	例外
waveform	unaffected はサポートされません。
waveform_element	null 波形エレメントは、ガード付き信号にのみ関係するので、サポートされません。

Verilog 言語サポートの例外

次の表は、サポートされる Verilog 言語サポートの例外をリストしています。

表 A-1 : Verilog 言語サポートの例外

Verilog 構文要素	例外
コンパイラ指示子構文	
`celldefine	サポートなし
`endcelldefine	サポートなし
`undefs	パラメータ指定された `define マクロがサポートされています。
`unconnected_drive	サポートなし
`nounconnected_drive	サポートなし
属性	
attribute_name	サポートなし
attr_spec	サポートなし
attr_name	サポートなし
プリミティブのゲート型およびスイッチ型	
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スエーション	

表 A-1 : Verilog 言語サポートの例外 (続き)

Verilog 構文要素	例外
generated_instantiation	<p>module_or_generate_item は、サポートされません。</p> <p>Production from 1364-2001 Verilog standard:</p> <pre>generate_item_or_null ::= generate_conditional_statement generate_case_statement generate_loop_statement generate_block module_or_generate_item</pre> <p>シミュレータでのサポート :</p> <pre>generate_item_or_null ::= generate_conditional_statement generate_case_statement generate_loop_statement generate_blockgenerate_condition</pre>
genvar_assignment	<p>一部サポート</p> <p>All generate blocks must be named.</p> <p>Production from 1364-2001 Verilog standard:</p> <pre>generate_block ::= begin [: generate_block_identifier] { generate_item } end</pre> <p>シミュレータでのサポート :</p> <pre>generate_block ::= begin: generate_block_identifier { generate_item } end</pre>
ソース テキスト構文	
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 ファイルの include 文を示します (IEEE 1364-2001、セクション 13.2 を参照)。コンパイラの `include 文のことではありません。
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

表 A-1 : Verilog 言語サポートの例外 (続き)

Verilog 構文要素	例外
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 モデル化タスク	
\$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
\$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
VCD (Value Change Dump) ファイル	
\$dumpportson \$dumpports \$dumpportsoff、\$dumpportsflush、 \$dumpportslimit \$vcdplus	サポートなし

付録 C

ModelSim XE から ISim への移行

ModelSim XE シミュレーション環境からザイリンクスの ISim シミュレーション環境へは、既存の環境を極端に変更しなくても移行できます。

ここでは、ModelSim XE から ISim への移行ガイドラインおよびその他の注意事項を記載します。

ISim を最大限に使用するには、次のビデオ デモおよびチュートリアルを参照してください。

- チュートリアル : http://japan.xilinx.com/support/documentation/dt_ise.htm
- ISim ビデオ デモ :
http://japan.xilinx.com/products/design_resources/design_tool/resources/index.htm
- ISim 製品ページ : <http://japan.xilinx.com/tools/isim.htm>

ModelSim XE の概要

ModelSim XE は、ModelSim Xilinx Edition の略で、Mentor Graphics 社の OEM 製品です。ModelSim XE では、デザインの論理モデルおよびタイミング モデルや HDL ソース コードを検証する HDL シミュレーション環境が提供されます。ModelSim XE は、Xilinx ISE[®] Design Suite 12.4 リリースで廃盤になっています。詳細は、『ModelSim Xilinx Edition-III ブロードキャスト製品の製造中止通知』を参照してください。

http://japan.xilinx.com/support/documentation/customer_notices/xcn10028.pdf

ModelSim XE は、ISE Design Suite 12.3 リリースから各メジャー リリースに含まれるようになっています。ModelSim XE には、次の 2 つのバージョンがあります。

- ModelSim XE Starter : ザイリンクス ウェブサイトからダウンロード可能な無償バージョン。この製品を使用するには、スターター ライセンスが必要です。
- ModelSim XE Full : Mentor Graphics 社の OEM バージョンで PE 製品ラインに基づいています。

ISim の概要

ISim はザイリンクスのシミュレーション製品で、ISE ツール、PlanAhead™ ツール、エンベデッドデザイン キット (EDK)、および System Generator に統合されている機能を完全に備えた HDL シミュレータです。

ISim は、ザイリンクス ツールのメジャー リリースすべてに含まれており、次の 2 つのバージョンがあります。

- **ISim Lite : ISE Simulator** の限定評価版。このバージョンでは、デザインとテストベンチの HDL コード行が 50,000 を超えるときに、パフォーマンスが低下します。
- **ISim Full : ISE Simulator** のフルバージョン。

機能比較

表 C-1 : ModelSim と ISim の機能の比較

機能	ModelSim XE	Starter ModelSim XE	Full ISim Lite	ISim Full
行制限 (文)	10,000	40,000	50,000	なし
パフォーマンス	ModelSim PE または ModelSim DE の 30%	ModelSim PE または ModelSim DE の 40%	ModelSim XE と同じ	ModelSim XE と同じ
混合言語	なし	なし	あり	あり
VHDL	あり	あり	あり	あり
Verilog	あり	あり	あり	あり
System Verilog for Design	なし	なし	なし	なし
System Verilog for Verification	なし	なし	なし	なし
デバッグ環境	あり	あり	あり	あり
スタンドアロン波形ビューアー	あり	あり	あり	あり
メモリ ビューアー / エディター	あり	あり	あり	あり
Verilog PLI/VPI	あり	あり	なし	なし
VHDL FLI/VHPI	なし	なし	なし	なし
コード カバレッジ	なし	なし	なし	なし
SecureIP/HardIP サポート	なし	なし	あり	あり
EDK サポート	なし	なし	あり	あり
System Generator サポート	なし	なし	あり	あり
CORE Generator サポート	あり	あり	あり	あり
MIG サポート	なし	なし	あり	あり

表 C-1 : ModelSim と ISim の機能の比較 (Cont'd)

機能	ModelSim XE	Starter ModelSim XE	Full ISim Lite	ISim Full
フローティング ライセンス	なし	なし	あり	あり
32 ビット OS サポート	Windows	Windows	Windows/Linux	Windows/Linux
64 ビット (ネイティブ) OS サポート	なし	なし	Windows/Linux	Windows/Linux

シミュレーション プロセス

このセクションでは、さまざまなモードのシミュレーションおよびシミュレーションの手順について説明します。各サブ セクション 2 つのシミュレータの違いについて説明します。

図 C-1 に、シミュレーションでの各ステップおよびそのプロセスを示します。

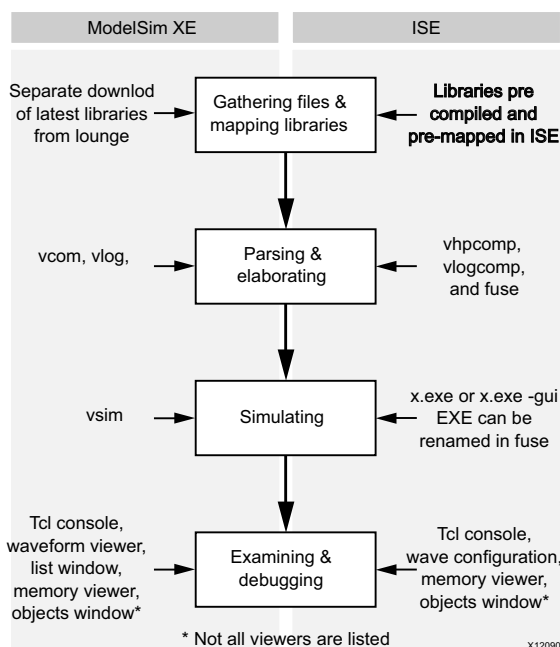


図 C-1 : シミュレーションの手順

手順 1 : ファイルの準備とライブラリのマップ

ModelSim XE のフロー

ModelSim XE ライブラリは、次からダウンロードできます。

<http://japan.xilinx.com/support/download/index.htm>

ISE Design Suite の新しいバージョンがリリースされる度に、このサイトからライブラリを個別にダウンロードする必要があります。行数制限でザイリンクス ライブラリが除外されるよう、これらのライブラリを使用する必要があります。

ModelSim XE と共に配布される modelsim.ini ファイルは、正しいザイリンクス ライブラリにあらかじめマップされています。

ISim のフロー

ISim のライブラリは、標準ザイリンクス インストールの一部としてアップデートされます。別のステップは不要です。マップもザイリンクスにより自動的に処理されます。シミュレーションを開始するのに、ザイリンクス ライブラリをダウンロードしたりマップする必要はありません。

手順 2 : デザインの解析とエラボレーション

ModelSim XE のフロー

ModelSim XE では、コンパイルおよびエラボレーションに次のコマンドが使用されます。VCOM オプション (VHDL コンパイラ) では、VHDL コンパイラが実行され、VHDL ファイルが指定ディレクトリにコンパイルされます。VLOG オプション (Verilog コンパイラ) では、Verilog コンパイラが実行され、VHDL ファイルが指定ディレクトリにコンパイルされます。VSIM オプション (VSIM シミュレーター) では、シミュレーションのロードがエラボレートされます。

これらのコマンドには、コンパイルおよびエラボレーションをさらに制御する複数のオプションがあります。同等の ModelSim XE コマンドをすべて示したリストは、[付録 B「VHDL および Verilog 言語サポートの例外」](#)を参照してください。

ISim のフロー

ISim では、コンパイルおよびエラボレーションに次のコマンドが使用されます。

- `vhpcomp` : (VHDL コンパイラ) : VHDL コンパイラが実行され、VHDL ファイルが指定ディレクトリにコンパイルされます。
- `vlogcomp` : (Verilog コンパイラ) : Verilog コンパイラが実行され、VHDL ファイルが指定ディレクトリにコンパイルされます。
- `fuse` : (VSIM シミュレータ) : シミュレーションのロードがエラボレートされ、シミュレーションを実行するのに起動する必要がある実行ファイルが作成されます。

これらのコマンドには、コンパイルおよびエラボレーションをさらに制御する複数のオプションがあります。

手順 3 : デザインのシミュレーション

ModelSim XE のフロー

VSIM を実行すると、デザインがエラボレートされ、シミュレーションが実行されます。デフォルトでは、`vsim` を実行すると GUI が起動します。

コマンド ライン モードで実行するには、`-c` オプションを使用します。

ISim のフロー

`fuse` を実行すると、ファイル名が付けられた実行ファイルが作成されます。この実行ファイルを実行して、シミュレーションを起動します。この実行ファイルは、デフォルトで `x.exe` という名前が付けられていますが、変更できます。

実行ファイルを実行すると、デフォルトではシミュレーションがコマンドライン モードで実行されます。GUI を起動するには、`-gui` オプションを使用します。

手順 4：デザインの確認およびデバッグ

波形操作のカスタマイズ

ModelSim XE および ISim では、波形ウィンドウをカスタマイズする機能が提供されていますが、カスタマイズ方法は異なります。ModelSim XE では標準の Tcl (ツール コマンド言語) コマンドがすべての波形操作に使用されるのに対し、ISim では Tcl コマンドのサブセットが使用されるものの、大部分が GUI からカスタマイズされ、その結果が波形コンフィギュレーション ファイルに保存されます。

ISim の波形コンフィギュレーション ファイルは XML ベースのファイルで編集できませんが、ModelSim XE の波形 Tcl コマンドは変更できます。ISim インプリメンテーションでの波形コンフィギュレーションの読み込み時間は、XML ファイルの読み込みが複数の Tcl コマンドを実行するよりも早いため、短くなります。

注記：ISim にはすべての波形操作に対する Tcl サポートがありません。

マーカーおよびカーソルを使用した計測

ModelSim XE と ISim では、マーカーおよびカーソルを使用した計測方法が多少異なります。ModelSim XE では、任意の 2 地点間を計測するカーソルが提供されます。必要に応じてカーソルを追加でき、新規カーソルは既存のカーソルの下に追加されます。波形ビューアーでは、カーソル間の距離が自動的に表示されます。図 C-2 は、カーソルを使用した ModelSim XE 波形を示しています。

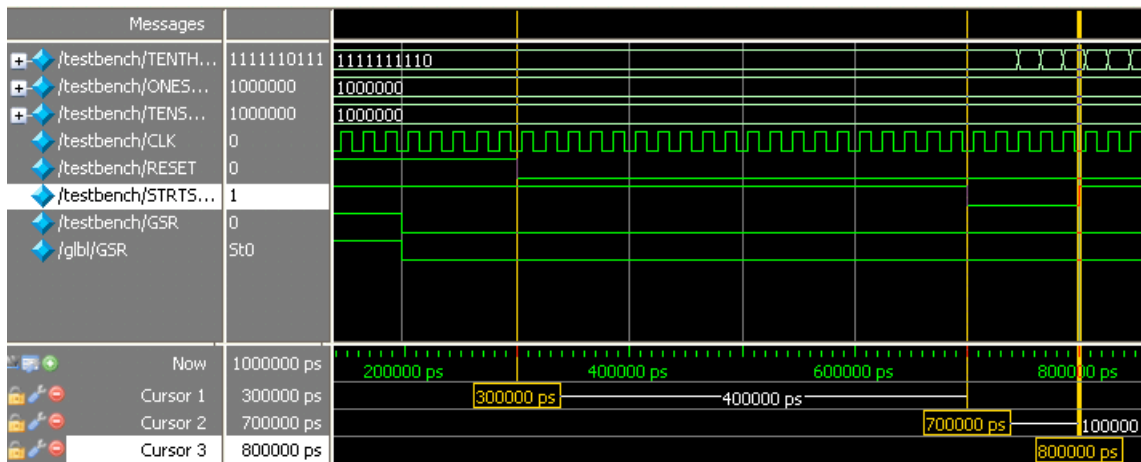


図 C-2：カーソルを使用した ModelSim XE の波形表示

ISim の計測方法は異なります。ISim ではカーソルとマーカーの両方が使用されます。ModelSim XE ではカーソルが永久的な計測手段として使用されるのに対し、ISim ではカーソルが一時的手段として使用されます。ISim にはメイン カーソルとセカンダリ カーソルがあり、これら 2 つを使用して 2 地点間の距離を計測できます。ISim では、複数地点間の距離を計測できます。図 C-3 は、ISim の計測を示しています。

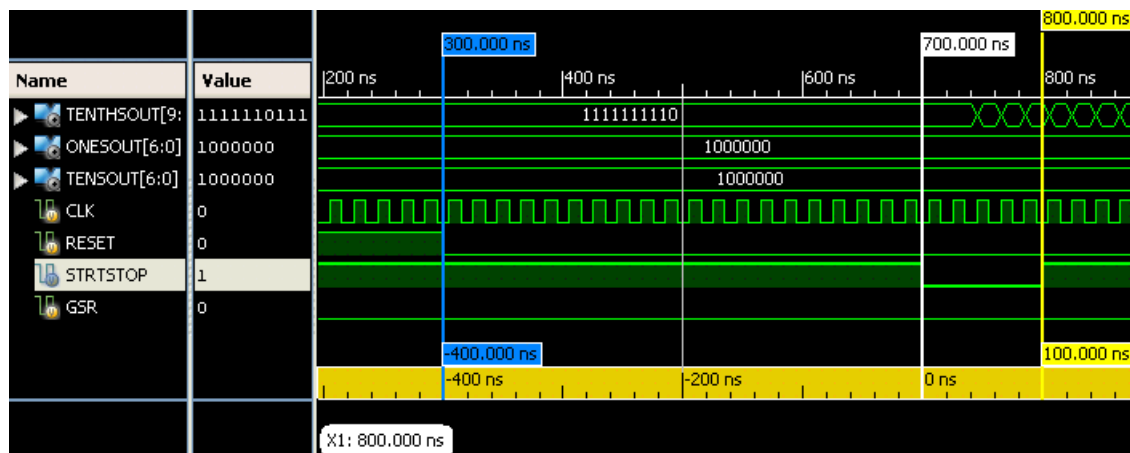


図 C-3 : ISim での計測

ISim では、フレームのルーラーも提供されます。選択されているマーカーまたはカーソルの位置は、計測されるその他すべてのマーカーに対して 0 になります。図 C-3 では、ISim で任意の地点間を計測する方法が示されています。

注記 : ISim では、マーカーの名前を変更できません。

アナログ波形

入手に関する詳細は、ザイリンクス テクニカル サポートまでお問い合わせください。

シングル クリック コンパイルおよび再読み込み

ModelSim XE では、スタンドアロン GUI にテキスト エディタがビルトインされているので、HDL コードの変更、再コンパイル、および再シミュレーションが実行できます。

ISim の GUI には HDL ファイルのみにテキスト ビューアがあります。ファイルに変更しても、再コンパイルおよび再シミュレーションは実行できません。既存のシミュレーションを終了し、ISE または PlanAhead ツールのテキスト エディターで HDL を変更してから ISim でシミュレーションを再実行します。

Project Navigator の統合

ISE Project Navigator では ModelSim XE が有効なシミュレータ チョイスではなく、統合されているその他のシミュレータを選択する必要があることが通知されます。次は、Project Navigator で ISim を選択する画面を示しています。

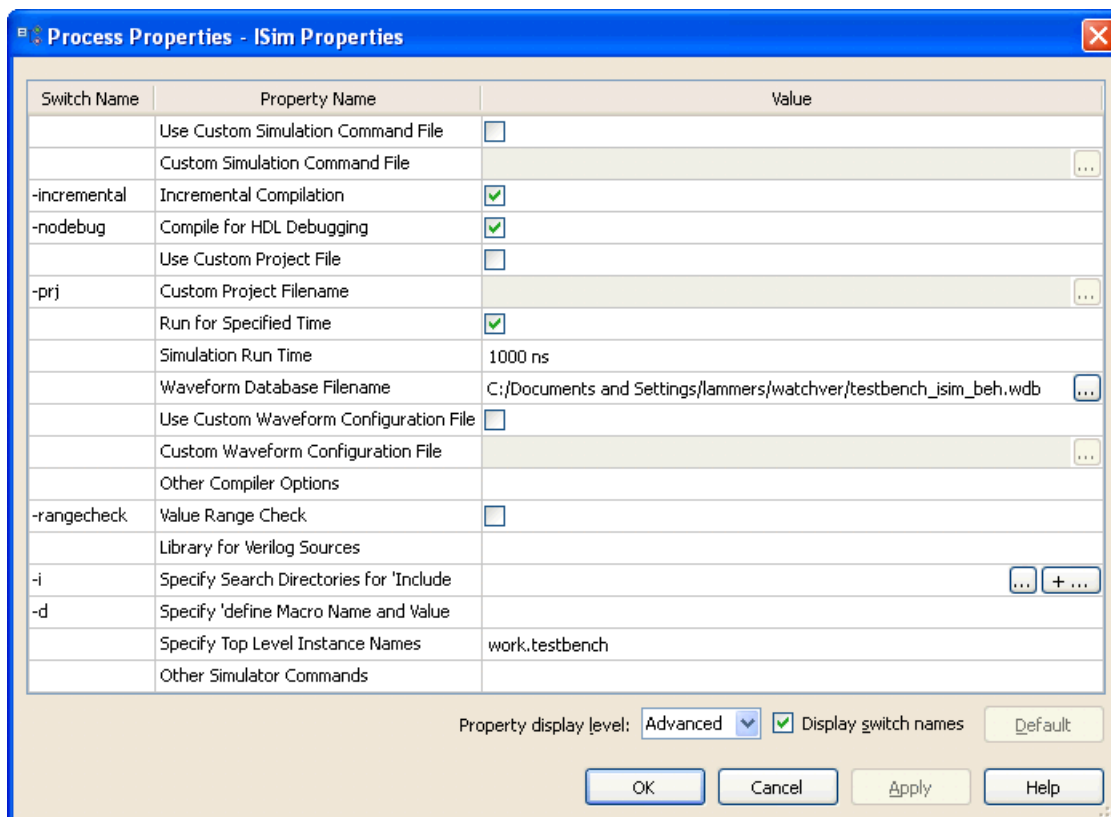


図 C-4 : ISim が選択されている Project Navigator の [Process Properties] ダイアログ ボックス

ModelSim XE と ISim のシミュレーション プロパティは類似してますが、異なる部分を次の表に示します。

表 C-2 : Project Navigator のシミュレーション プロパティ

ModelSim XE [Property Name]	ISim [Property Name]	コメント
ライブラリ コンパイル		
[Compiled Library Directory]	なし	ISE Design Suite インストールで配布される ISim 用コンパイル済みライブラリ
[Ignore Pre-Compiled Library Warning Check]	なし	

表 C-2 : Project Navigator のシミュレーション プロパティ (Cont'd)

ModelSim XE [Property Name]	ISim [Property Name]	コメント
[Generate Verbose Library Compilation Messages]	なし	
カスタム ユーザー コマンド		
[Use Custom Do File]	[Use Custom Simulation Command File] [Use Custom Wave Configuration File]	ISim では、エンジン操作を制御する Tcl コマンドと共通 GUI 操作のほとんどを制御するコマンドの両方がサポートされています。また、波形コンフィギュレーション ファイルを使用して波形ウィンドウをすばやく設定できます。
[Custom Do File]	[Custom Simulation Command File] [Custom Wave Configuration File]	
[Use Automatic Do File]	なし	Project Navigator による ISim スクリプトの作成を回避できません。
[Custom Compile File List]	[Use Custom Project File] [Custom Project Filename] (アドバン ス プロパティ)	ファイルのコンパイル順を変更できます。
なし	[Waveform Database Filename]	シミュレーションで別のデータベースを指定できます。
カスタム コンパイラ コマンド		
[Other VSIM Command Line Options]	[Other Compiler Options] [Other Simulator Commands]	ISim では VSIM コマンドが fuse コマンドと実行ファイル コマンドに分割されます。
[Other VLOG Command Line Options]	[Other Compiler Options]	ISim の fuse コマンドにオプションを渡します。
[Other VCOM Command Line Options]		

表 C-2 : Project Navigator のシミュレーション プロパティ (Cont'd)

ModelSim XE [Property Name]	ISim [Property Name]	コメント
ランタイム設定		
[Simulation Run Time]	[Simulation Run Time]	
[Simulation Resolution]	なし	ISim のデフォルトは 1ps です。
言語設定		
[VHDL Syntax]	なし	ISim のデフォルトは 93 です。
[Use Explicit Declarations Only]	なし	なし
[Other VCOM Command Line Options]	[Value Range Check]	ModelSim XE にはこれに対する特定のオプションはありませんが、[Other Command Line Options] プロパティで指定できます。
	[Specify Search Directories for `include] インクリメンタル コンパイル	
	[Specify `define Macro Name and Value] インクリメンタル コンパイル	
なし	[Compile for HDL Debugging]	
その他の設定		
[Use Configuration Name]	なし	
[Configuration Name]	なし	
[Log All Signals in Simulation]	なし	
[Other VSIM Command Line Options]	[Specify Top-Level Instance Name]	

付録 D

その他のリソース

ザイリンクス リソース

- デバイスのユーザー ガイド
http://japan.xilinx.com/support/documentation/user_guides.htm
- ザイリンクス用語集
<http://japan.xilinx.com/company/terms.htm>
- 『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)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_3/iil.pdf
- 『ザイリンクス デザイン ツール：リリース ノート ガイド』(UG631)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_3/irn.pdf
- 製品サポートおよびマニュアル
<http://japan.xilinx.com/support>
- 『合成/シミュレーション デザイン ガイド』(UG626)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_3/sim.pdf
- 『PlanAhead ユーザー ガイド』(UG632)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_3/PlanAhead_UserGuide.pdf
- 『コマンド ライン ツール ユーザー ガイド』(UG682)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_3/devref.pdf
- 『ChipScope Pro ソフトウェアおよびコア ユーザー ガイド』(UG029)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_3/chipscope_pro_sw_cores_ug029.pdf
- ISE ヘルプ
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_3/isehelp_start.htm
- XPower ヘルプ
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_3/isehelp_start.htm#xpa_c_overview.htm

ISim チュートリアル

- 『ISim チュートリアル』(UG682)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug682.pdf
- 『ISE ハードウェア協調シミュレーション チュートリアル：浮動小数点高速フーリエ変換のシミュレーションの高速化』(UG817)
http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug817_fft_sim_tutorial.pdf