

# ISim ユーザー ガイド

UG660 (v.12.2) 2010 年 7 月 23 日

# 目次

---

1: 入門.....	9
ISim の概要 .....	9
シミュレーション ライブラリ.....	9
言語サポート.....	9
機能サポート.....	9
ISim の OS サポート .....	10
ISim 12.1 の新機能 .....	10
一般的な機能 .....	10
GUI での改善点.....	10
Tcl コマンドでの改善点 .....	10
シミュレーションの手順 .....	11
ISim の操作モード .....	11
シミュレーション手順の概要 .....	12
手順 1: ファイルの準備とライブラリのマップ .....	12
手順 2: デザインの解析とエラボレーション .....	13
手順 3: デザインのシミュレーション .....	14
手順 4: デザインの検証 .....	15
手順 5: デザインのデバッグ .....	16
チュートリアル .....	17
サポートされなくなった機能およびコマンド.....	17
サポートされなくなったファイルの種類 .....	17
サポートされなくなったコマンド .....	17
サポートされなくなったコマンドラインのオプション .....	18
その他の変更.....	18
2: ISim グラフィカル ユーザー インターフェイスの使用.....	19
グラフィック ユーザー インターフェイスの概要 .....	19
GUI の起動 .....	19
GUI の説明 .....	20
デザイン階層およびオブジェクトのアイコン .....	21
デザイン階層アイコン .....	21
デザイン オブジェクト アイコン .....	21
メイン ウィンドウの整列 .....	22
ウィンドウの整列 .....	22
ウィンドウの非表示および復元.....	23
階層の展開/非展開 .....	24
波形ウィンドウ .....	25
波形ウィンドウの概要 .....	25
波形コンフィギュレーションでの作業 .....	28
[Instances and Processes] パネル .....	30
[Instances and Processes] パネルの概要 .....	30
オブジェクトの検索 .....	30
HDL ソース ファイルを開く.....	31
[Objects] パネル .....	31
[Objects] パネルの概要 .....	31

オブジェクトの検索 .....	32
[Show Drivers] コマンドの使用 .....	32
エレメントの表示 .....	33
波形ウィンドウでのオブジェクトの選択 .....	33
[Source Files] パネル .....	33
[Source Files] パネルの概要 .....	33
テキスト エディタ ウィンドウ .....	34
テキスト エディタ ウィンドウの概要 .....	34
HDL ソース ファイルを開く .....	34
HDL ソース ファイルを表示する .....	35
ブレークポイントの設定 .....	35
メモリ エディタ ウィンドウ .....	36
ISimメモリ エディタ ウィンドウの概要 .....	36
[Console] パネル .....	36
[Console] パネルの概要 .....	36
[Breakpoints] パネル .....	37
[Breakpoints] パネルの概要 .....	37
[Search Results] パネル .....	37
[Search Results] パネルの概要 .....	37
[Find in Files Results] パネル .....	37
[Find in Files] コマンドの使用 .....	37
ツールバー コマンドおよびショートカット .....	38
ISim ツールバー コマンド .....	38
ショートカット .....	41
スティミュラスの適用 .....	42
[Force Selected Signal] ダイアログ ボックス .....	42
[Define Device] ダイアログ ボックス .....	42
ISim のプリファレンス .....	43
ISim プリファレンスの設定 .....	43
ISE Text Editor のプリファレンス .....	44
ISim シミュレータのプリファレンス .....	44
[Source Properties] ダイアログ ボックス - [Hardware Co-Simulation Properties] ページ .....	44
ISim カラー プリファレンス .....	45
時間フォーマットのプリファレンス .....	45
<b>3: VHDL シミュレーション .....</b>	<b>47</b>
VHDL シミュレーションの概要 .....	47
コマンド ラインからの論理シミュレーションの実行 (VHDL デザイン) .....	47
方法 1: プロジェクト ファイルの使用 (推奨) .....	47
方法 2: vhpcomp を使用したファイルの解析 .....	48
シミュレーション .....	49
コマンド ラインからのタイミング シミュレーションの実行 (VHDL デザイン) .....	49
タイミング シミュレーション モデルの生成 .....	49
方法 1: プロジェクト ファイルの使用 (推奨) .....	49
方法 2: vhpcomp の使用 .....	50
シミュレーション .....	51
ライブラリ マップ ファイル .....	52

検索順 .....	52
構文 .....	52
例 .....	52
機能/制限 .....	53
コマンドライン モードでの対話型シミュレーション .....	53
<b>4 : Verilog シミュレーション .....</b>	<b>55</b>
Verilog シミュレーションの概要 .....	55
コマンドラインからの論理シミュレーションの実行 (Verilog デザイン) .....	55
方法 1 : プロジェクトファイルの使用 (推奨) .....	55
方法 2 : vlogcomp を使用したファイルの解析 .....	56
シミュレーション .....	57
コマンドラインからのタイミング シミュレーションの実行 (Verilog デザイン) .....	58
タイミング シミュレーション モデルの生成 .....	58
方法 1 : プロジェクトファイルの使用 (推奨) .....	58
方法 2 : vlogcomp を使用したファイルの解析 .....	59
シミュレーション .....	60
Verilog デザイン ユニットのインスタンスの検索順位 .....	61
ソース ライブラリのサポート .....	61
ライブラリのディレクトリ (-sourcelibdir) .....	61
ソース ファイルの拡張子 (-sourcelibext) .....	61
ソース ファイル (-sourcelibfile) .....	61
ライブラリ マップ ファイル .....	62
検索順 .....	62
構文 .....	63
例 .....	63
機能/制限 .....	63
Verilog シミュレーション用の定義済み XILINX_SIM マクロ .....	64
コマンドライン モードでの対話型シミュレーション .....	64
<b>5 : 混合言語シミュレーション .....</b>	<b>65</b>
混合言語シミュレーションの概要 .....	65
シミュレーションでの混合言語の制限 .....	65
混合言語シミュレーションでの主要手順 .....	65
混合言語コンポーネントのインスタンス化 .....	66
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 .....	66
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 .....	66
混合言語デザインでのバインドと検索 .....	66
VHDL インスタンス化 ユニット .....	67
Verilog インスタンス化 ユニット .....	67
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 .....	67
一般 .....	68
ポートのマップ .....	68
ジェネリック (パラメータ) のマップ .....	69
VHDL/Verilog の値のマップ .....	69
<b>6 : 波形の解析 .....</b>	<b>71</b>
解析の実行前 .....	71
ISim GUI の起動 .....	71

波形コンフィギュレーションへの信号の追加 .....	71
信号/バスのコピーの追加 .....	72
ISim でのシミュレーションの実行 .....	73
シミュレーションの一時停止 .....	73
ISim の終了 .....	74
波形コンフィギュレーションのカスタマイズ .....	74
カーソルの配置 .....	74
マーカーの設定 .....	75
仕切りの追加 .....	76
グループの追加 .....	76
仮想バスの追加 .....	77
オブジェクト名の変更 .....	77
表示名の変更 .....	78
基数の変更 .....	78
バスビット順の反転 .....	79
波形コンフィギュレーションのナビゲーション .....	79
階層の展開/非展開 .....	79
ズーム機能 .....	80
フロート ルーラの表示 .....	81
マーカーを使用した波形値の表示 .....	82
信号遷移の波形値の表示 .....	82
カーソルを使用した時間の計測 .....	83
マーカーを使用した時間の計測 .....	83
[Go To Time] コマンドの使用 .....	84
[Show Drivers] コマンドの使用 .....	84
波形コンフィギュレーションの印刷 .....	84
印刷プレビューを表示するには .....	85
印刷するには .....	85
カスタム カラーの使用 .....	85
カスタム カラー スキームの作成 .....	85
信号の表示色の変更 .....	85
<b>7: シミュレーション結果の保存および表示 .....</b>	<b>87</b>
シミュレーション結果の保存 .....	87
WDB ファイルへのデータベースの保存 .....	87
WCFG ファイルへの波形コンフィギュレーションの保存 .....	87
WDB と WCFG の関係 .....	87
ライブ シミュレーションを開く .....	88
波形データ ベースを開く .....	88
波形ウィンドウを開く .....	88
スタティック シミュレーションを開く .....	89
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 .....	89
波形データ ベースのみを開く .....	90
<b>8: デバッグ .....</b>	<b>91</b>
ソース コードのデバッグの概要 .....	91
1 行ずつの実行 .....	91
シミュレーションの 1 行ずつの実行 .....	91
ブレークポイント .....	92
ブレークポイントの設定 .....	92
ブレークポイントを使用したデザインのデバッグ .....	92

ブレイクポイントの削除.....	93
<b>9： 消費電力概算向けアクティビティ データの書き出し.....</b>	<b>95</b>
デザインのアクティビティ データの書き出し.....	95
アクティビティ ファイルの種類.....	95
消費電力のアクティビティ ファイルを書き出すには.....	96
<b>10： Tcl シミュレーション コマンドの使用 .....</b>	<b>97</b>
シミュレーション コマンドの概要 .....	97
シミュレーション コマンドの入力方法 .....	97
シミュレーション コマンドのサマリ.....	97
シミュレーション コマンドの入力 .....	99
GUI モードでコマンドを入力するには .....	99
対話型コマンド ライン モードでコマンドを入力するには.....	99
非対話型バッチ モードでコマンドを入力するには.....	100
シミュレーション コマンドの別名表記 .....	101
ISim 波形ビューア Tcl コマンド.....	101
コマンド ラインの表記規則.....	101
Tcl コマンド .....	102
エンジン コマンド .....	102
波形コンフィギュレーション入力/出力コマンド .....	131
波形コンフィギュレーション編集コマンド .....	133
マーカー コマンド.....	137
波形ビューア リソース コマンド.....	137
<b>付録： リファレンス.....</b>	<b>139</b>
シミュレーション実行コマンド .....	139
シミュレーション実行コマンドの概要 .....	139
ISE シミュレーション実行ファイル .....	140
fuse .....	143
vlogcomp.....	149
vhpcomp .....	152
サードパーティ コマンドの等価性 .....	154
サードパーティのシミュレーション コマンドのサポートの概要 .....	154
サードパーティのコンパイラ コマンド.....	155
サードパーティの Tcl コマンド.....	155
HDL 言語のサポート.....	158
VHDL 言語のサポート (a ~ m).....	158
VHDL 言語のサポート (n ~ z).....	163
Verilog 言語のサポート .....	165



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 2002–2010 Xilinx Inc. All Rights Reserved. XILINX, the Xilinx logo, the Brand Window and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners. The PowerPC name and logo are registered trademarks of IBM Corp., and used under license. All other trademarks are the property of their respective owners.

本資料は英語版 (v.12.2) を翻訳したもので、内容に相違が生じる場合には原文を優先します。  
資料によっては英語版の更新に対応していないものがあります。  
日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。







## 入門

---

### ISim の概要

Xilinx® ISim は、VHDL、Verilog、および VHDL/Verilog 混合デザインで論理シミュレーションおよびタイミング シミュレーションを実行する HDL (Hardware Description Language) シミュレータです。

### 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。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。

言語	サポート
VHDL	IEEE-STD-1076-2000
Verilog	IEEE-STD-1364-2001
SDF	ザイリンクス NetGen で生成された SDF ファイル
VITAL	VITAL-2000
混合 VHDL/Verilog	あり
VHDL FLI/VHPI	なし
Verilog PLI	なし
SystemVerilog	なし
ほかのアサート ベースの言語	なし

### 機能サポート

ISim では、次の機能がサポートされています。

機能	サポート
インクリメンタル コンパイル	あり
ソース コードのデバッグ	あり
SDF のアノテーション	あり
VCD の生成	あり
SAIF のサポート	あり
ハード IP : MGT、PPC、PCIe® など	あり
マルチスレッド	あり

## ISim の OS サポート

ISim の OS サポートは、『ISE Design Suite：インストール、ライセンス、およびリリース ノート』を参照してください。

**メモ：** Windows XP/Vista (64 ビット) システムで 32 ビット/64 ビット モードを使用すると、ISim を実行できます。32 ビット/64 ビット モードでザイリンクス ツールを実行する方法の詳細は、『ISE Design Suite：インストール、ライセンス、およびリリース ノート』を参照してください。

## ISim 12.1 の新機能

### 一般的な機能

- ・ ISim のネイティブ Win 64 ポート。詳細は、『ISE Design Suite: インストール、ライセンス、およびリリース ノート』を参照してください。
- ・ [メモリ エディタ ウィンドウ](#)
- ・ Project Navigator でサポートされるマップ後および変換後フロー
- ・ XPS および Project Navigator に含まれるエンベデッド デザインのシミュレーションのサポート
- ・ [ISim ハードウェア協調シミュレーション](#) (アクセス制限あり)

### GUI での改善点

- ・ 波形ビューアに表示される[時間の値](#)の単位と小数点の桁数 (精度) の表示をカスタマイズできるようになりました。
- ・ [Objects] パネルおよび波形ビューアで [\[Define Clock\] ダイアログ ボックス](#)または [\[Force Selected Signal\] ダイアログ ボックス](#)を開いて、`isim force` Tcl コマンドと同様の操作を実行できるようになりました。
- ・ Verilog レジスタにも `isim force` コマンドを使用できるようになりました。

### Tcl コマンドでの改善点

[ISim 波形ビューア Tcl コマンド](#)

## シミュレーションの手順

### ISim の操作モード

ISimには次の 3 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
- ・ 対話型コマンド ライン
- ・ 非対話型バッチ

操作モード	機能	ISim の起動方法
グラフィカル ユーザー インターフェイス	<p>シミュレーション データのグラフィカル表示。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、コンテキストコマンド、およびツールバー ボタンを使用します。また、[Console] タブに Tcl コマンドを入力しても実行できます。</p> <p>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、「<a href="#">グラフィカル ユーザー インターフェイスの概要</a>」を参照してください。</p>	<ul style="list-style-type: none"> <li>・ ISE® からの場合：[Simulate Behavioral Model] など、デザインでシミュレーション プロセスを実行します。</li> <li>・ コマンド プロンプトからの場合：<code>my_sim.exe -gui</code> など、シミュレーション実行ファイルを <code>-gui</code> オプションと共に実行します。</li> <li>・ コマンド プロンプトからの場合：波形コンフィギュレーション ファイルと直前のシミュレーションを開くには、シミュレーション実行ファイルを <code>-gui</code> オプションおよび <code>-view &lt;file.wcfg&gt;</code> と共に実行します。</li> </ul> <p><b>メモ：</b> また、<code>isimgui.exe -view &lt;wcfg_file&gt;.wcfg</code> を使用すると、読み取り専用モードで GUI を開くことも可能です。</p>
対話型コマンドライン	<p>グラフィカル ユーザー インターフェイス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。</p> <p>詳細は、「<a href="#">VHDL シミュレーションの概要</a>」、「<a href="#">Verilog シミュレーションの概要</a>」、または「<a href="#">混合言語シミュレーションの概要</a>」を参照してください。</p>	<p>コマンド プロンプトからの場合：</p> <ol style="list-style-type: none"> <li>1. シミュレーション実行ファイルを生成するコマンドを実行します。 例：<code>fuse -prj my_prj.prj tb -L unisims_ver -L userlib -o my_sim.exe</code></li> <li>2. シミュレーション実行ファイルを実行します。例：<code>my_sim.exe</code></li> </ol>

操作モード	機能	ISim の起動方法
非対話型バッチ	<p>グラフィカル ユーザー インターフェイスとの対話はありません。コマンド シーケンス 1 つが実行され、Tcl コマンドを含む 1 つのバッチ ファイルに含まれているコマンド オプションおよび内容によりすべての操作が制御されます。</p> <p>詳細は、「<a href="#">ISim シミュレーション実行ファイル コマンドの概要と構文</a>」を参照してください。</p>	<p>コマンド プロンプトからの場合：</p> <ol style="list-style-type: none"> <li>1. シミュレーション実行ファイルを生成するコマンドを実行します。 例：<code>fuse -prj my.prj tb -L mylib -L yourlib -o my_sim.exe</code></li> <li>2. Tcl コマンドを含むファイルを生成します。</li> <li>3. シミュレーション実行ファイルを <code>-tclbatch</code> オプションと共に実行します。例：<code>my_sim.exe -tclbatch cmd.tcl</code></li> </ol>

## 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ときの基本的な手順を次で説明します。

- ・ [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

次のファイルは、ISim でシミュレーションを実行するのに必要です。

- ・ スティミュラス ファイルを含むデザイン ファイル
- ・ ユーザー ライブラリ
- ・ その他のデータ 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**：任意のテキスト エディタで HDL テストベンチを作成、編集します。
- ・ **言語テンプレート**：ISE® ソフトウェア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 を起動するときに使用するユーザー 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 ISim を起動する場合は、ISE ソフトウェアでユーザー 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 スタンドアロンを使用する場合は、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 ファイルにユーザーの論理/物理ライブラリを指定するように設定します。

## 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

ISim グラフィカル ユーザー インターフェイスを起動する際、次の 2 つのオプションがあります。

- ・ デザインが解析され、ISE® から ISim を起動したときにデザイン 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」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** オプションと共に実行してグラフィカル 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、(2) **fuse** を使用してプロジェクト ファイルからデザインを解析してデザインをエラボレーションし、シミュレーション実行ファイルを生成します。

## プロジェクト ファイルの作成

プロジェクト ファイル (PRJ) は **fuse** コマンドと使用され、デザインに関連するすべてのファイルのリストを供給します。PRJ ファイルには、言語、ライブラリ名、およびデザイン ファイルが含まれています。

1. テキスト ファイルを作成して、.prj ファイル拡張子を割り当てます。
2. プロジェクト ファイルの最初の行に、ライブラリおよびソース ファイル情報を次のように入力します。

```
verilog|vhdl <library_name> {<file_name_1>.v|.vhd}  
verilog|vhdl <library_name> {<file_name_2>.v|.vhd}  
.  
verilog|vhdl <library_name> {<file_name_n>.v|.vhd}
```

説明：

- ・ `verilog|vhdl`：ソース ファイルが Verilog または VHDL ファイルであることを示します。`verilog` または `vhdl` を含めます。
- ・ `<library_name>`：指定行のソースがコンパイルされるライブラリを指定します。デフォルト ライブラリは `work` です。
- ・ `<file_name>`：ライブラリに関連するソース ファイルを指定します。1 行に複数の Verilog ソース ファイルを指定できます。VHDL ソース ファイルは 1 行に 1 つ指定できます。

## fuse の使用

PRJ を使用してデザインを解析し、デザインをエラボレーションし、シミュレーション実行ファイルを生成するには、**fuse** コマンドを使用します。次に例を示します。

```
fuse -prj my_project.prj work.top work.glbl -o my_sim.exe
```

`fuse` コマンドの構文および使用可能なオプションの詳細は、「[fuse コマンドの概要と構文](#)」を参照してください。

この手順が正しく実行されたことを確認します。実行されていない場合は、「[手順 5：デザインのデバッグ](#)」の「エラー メッセージの確認」および「ログ ファイルの確認」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、「[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。

## グラフィカル ユーザー インターフェイス モード

コマンド ラインからのシミュレーション

1 つ前のプロセスでシミュレーション実行ファイルが生成されました (**x.exe** (default) またはユーザー指定の **my\_sim.exe**)。このシミュレーション実行ファイルを「**my\_sim.exe -gui**」のように **-gui** オプションと共に実行します。これで ISim グラフィカル ユーザー インターフェイス (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「[ISim でのシミュレーションの実行](#)」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。また、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、「[ISim GUI の起動](#)」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。たとえば、「**my\_sim.exe -gui -tclbatch my\_sim.tcl**」と入力します。my\_sim.tcl ファイルで **wave add** コマンド (すべての信号を最上位に追加する **wave add /** など) を使用して 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ファイル コマンドのすべては、ISE® ソフトウェア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]** : 詳細は、ISE ヘルプの「[ビヘイビア シミュレーションの実行](#)」を参照してください。
- ・ **[Simulate Post-Place & Route Model]** : 詳細は、ISE ヘルプの「[タイミング シミュレーションの実行](#)」を参照してください。

これらの ISE プロセスでは ISim 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、ISE でカスタム Tcl ファイルを指定して、ISim GUI の起動時にトレースする信号を制御できます。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。

シミュレーションをさらに実行するには「[ISim でのシミュレーションの実行](#)」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。詳細は、「[ISim GUI の起動](#)」を参照してください。

## 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 -gui -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。実行されていない場合は、「[手順 5：デザインのデバッグ](#)」の「エラー メッセージの確認」および「ログ ファイルの確認」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、ほとんどの場合デザイン仕様が満たされるようにデザインをデバッグする必要があります。

シミュレーション結果の検証は、次の 2 つの方法で実行できます。

1. [波形ウィンドウ](#)の信号の動作を確認します。
2. [\[Console\] パネル](#)または Tcl プロンプトで結果を確認します。詳細は、「[シミュレーション コマンドの概要](#)」を参照してください。

結果は保存できます。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。

シミュレーション結果は、読み取り専用のスタティック シミュレータでも表示できます。詳細は、「[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。

### 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、問題の追究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。「[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。

- ・ **`fuse.log`** : [コンパイルおよびエラボレーション](#) プロセス中に `fuse` 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。
- ・ **`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](#)

詳細は、「[シミュレーション コマンドの概要](#)」を参照してください。

## グラフィカル ユーザー インターフェイスのデバッグ

GUI を使用したデバッグ手法の詳細は、「[ソースコードのデバッグの概要](#)」を参照してください。

## チュートリアル

ISim の使用方法に関するチュートリアルは、『ISim In-Depth Tutorial』を参照してください。

このチュートリアルでは、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。このチュートリアルを表示するには、ISim で [Help] → [Tutorial] をクリックします。

## サポートされなくなった機能およびコマンド

### サポートされなくなったファイルの種類

ISE® 11.1 リリースより、スタティック波形ビューアで開いたザイリンクス波形 (.xwv) ファイルは波形データベース (.wdb) ファイルに置き換えられました。ISE® 11.1 リリースより前のリリースで作成されたデザインは、ISE® 12 ソフトウェアで再実行して ISim でシミュレーションし、WDB ファイルを開いて保存する必要があります。

### サポートされなくなったコマンド

次のコマンドは ISE® 11.1 リリースからサポートされなくなりました。使用しても無視されるか、エラーになります。

サポートされなくなったコマンド	代わりに使用するコマンド
<b>isimwave</b>	<b>isimgui -view &lt;wcfg_file&gt;.wcfg</b> または <b>isimgui -view &lt;wdb&gt;.wdb</b> (Linux)  <b>isimgui.exe -view &lt;wcfg_file&gt;.wcfg</b> または <b>isimgui.exe -view &lt;wcfg_file&gt;.wdb</b> (Windows)  詳細は、「 <a href="#">スタティック シミュレーションを開く</a> 」を参照してください。

サポートされなくなった Tcl コマンド	代わりに使用するコマンド
<code>dump -p</code>	<code>show child</code>
<code>dump -p &lt;process_name&gt;</code>	<code>scope &lt;process_name&gt;</code> <code>dump</code>
<code>isim batch &lt;on/off&gt;</code>	なし
<code>ntrace</code>	<code>wave log</code> または <code>wave add</code> コマンドを代わりに使用
<code>show status</code>	なし
<code>stop</code>	コマンドライン モードで <b>Ctrl + C</b> をクリックします。 GUI モードで [Break] ボタン (  ) をクリックします。

## サポートされなくなったコマンド ラインのオプション

次のコマンド オプションは ISE® 11.1 リリースからサポートされなくなりました。推奨する構文を次に示します。

サポートされなくなった fuse コマンド オプション	代わりに使用する構文
<code>fuse -top</code> (例 : <code>fuse -top yourtop -top glbl -top thirddtop -work yourlib</code> )	モジュールは <code>&lt;library_name&gt;.&lt;top_name&gt;</code> で指定されます。 (例 : <code>yourlib.yourtop yourlib.glbl yourlib.thirddtop</code> ) 詳細は「 <a href="#">fuse コマンド</a> 」を参照してください。
<code>fuse -work</code> (例 : <code>fuse -top yourtop -top glbl -top thirddtop -work yourlib</code> )	ライブラリは <code>&lt;library_name&gt;.&lt;top_name&gt;</code> で指定されます。 例 : <code>fuse work.yourtop work.glbl yourlib.thirddtop</code>

## その他の変更

- ・ 11.1 リリースより、デフォルトのシミュレーション タイムスケールが 1fs/1fs から 1ns/1ps に変更されました。
- ・ 11.3 リリースでは [Simulation] → [Stop] メニュー コマンドおよび [Stop] ツールバー ボタンが削除されました。ISim を閉じる場合は、別の方法を使用してください。

## ISim グラフィカル ユーザー インターフェイスの使用

---

### グラフィック ユーザー インターフェイスの概要

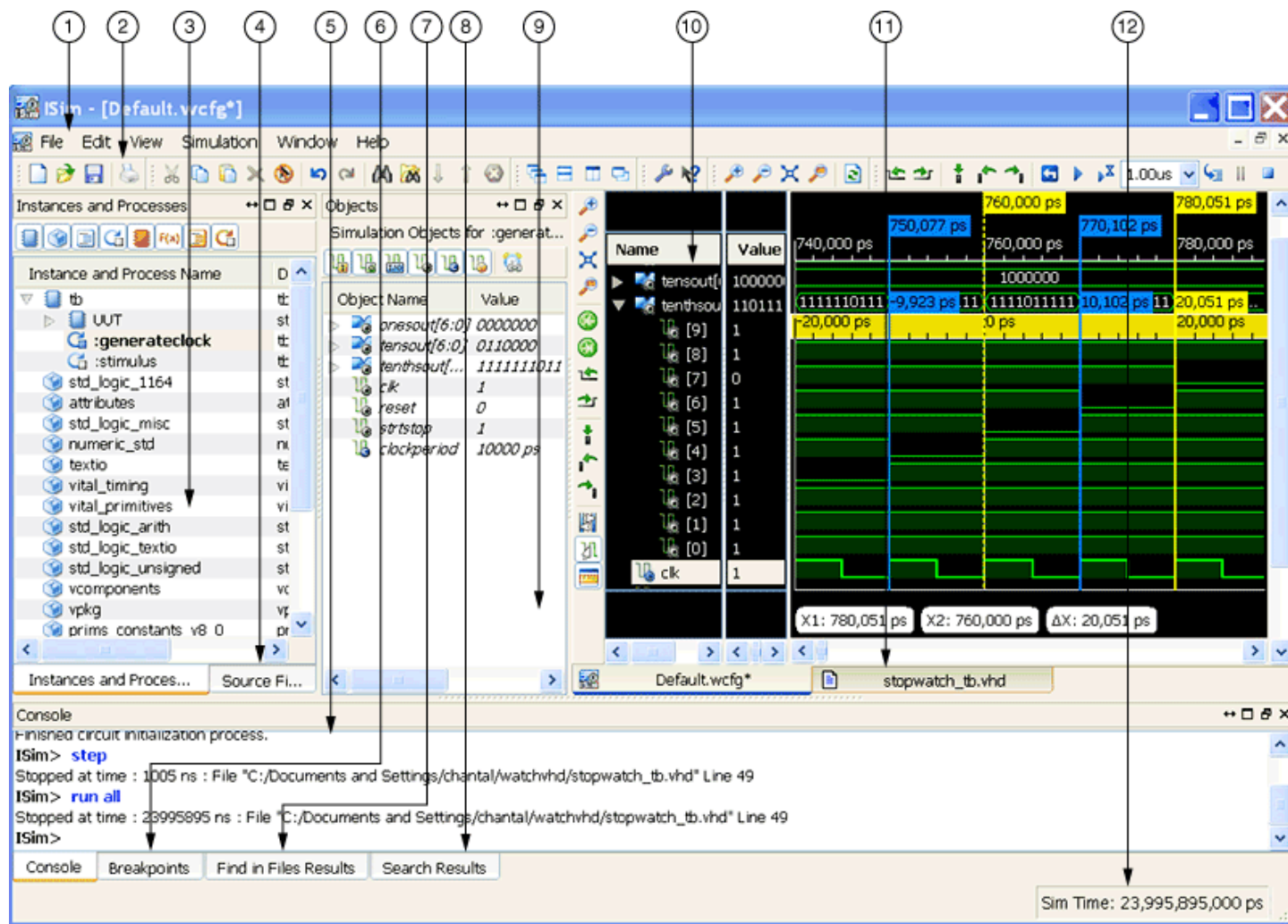
ISim のグラフィック ユーザー インターフェイス (GUI) は、さまざまなパネルを含むメイン ウィンドウ、ワークスペース、ツールバー、およびステータス バーから構成されています。メイン ウィンドウでは、デザインのシミュレーションが可能な箇所の表示、波形コンフィギュレーションでの信号の追加および表示、コマンドを使用したシミュレーションの実行、デザインの検証、およびデバッグを実行できます。

### GUI の起動

ISim の GUI は、シミュレーション実行ファイルを ISE® ソフトウェアまたはコマンドラインから実行すると起動します。詳細は、「[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。

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。

## GUI の説明



#	GUI パーツ	説明
1	メニュー コマンド	ソフトウェア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、コンテキスト メニューからのみアクセス可能です。
2	ISim ツールバー コマンド	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ー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。詳細は、「[Find in Files] コマンドの使用」を参照してください。

#	GUI パーツ	説明
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 ウィンドウ	読み取り専用の HDL ファイルが表示されます。
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。

## デザイン階層およびオブジェクトのアイコン

### デザイン階層アイコン







デザイン エンティティ/モジュールが [Instances and Processes] パネルのデザイン階層に表示されます。










	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
	Verilog タスクまたはファンクション
	Verilog ブロック
	Verilog プロセス

### デザイン オブジェクト アイコン

次のデザイン オブジェクトは、[Objects] パネルおよび波形ウィンドウで表示されています。

#### 信号

	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、およびジェネリック信号
	変数

	リンケージ信号
	バッファ信号
<b>バス</b>	
	入力バス
	出力バス
	入出力、双方向バス
	内部バス
	定数、パラメータ、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 バス






## メイン ウィンドウの整列

### 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 ウィンドウ](#)でのみ使用できます。

-  [Cascade] : 作業中のウィンドウを一番上にして、すべてのウィンドウを重ねて表示します。
-  [Tile Horizontally] : ウィンドウを重ねないように上下に並べて表示します。
-  [Tile Vertically] : ウィンドウを重ねないように左右に並べて表示します。
-  [Float] : 作業中のファイルをフロートさせます。
-  [Dock] : フロートさせたウィンドウをインターフェイスの元の位置に戻します。このコマンドメニューは、フロートしているウィンドウがあるときのみ表示されます。

#### 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。

1. 移動させるパネルのヘッダ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。

**メモ：** 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。

### メニュー コマンド

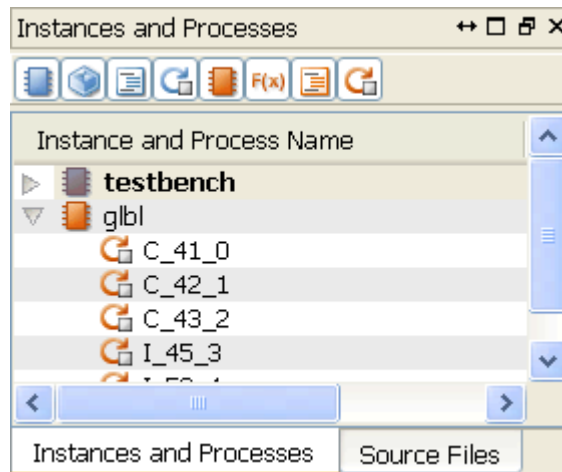
[View] メニュー コマンドを使用すると、メイン ウィンドウのパネル、ツールバー、およびステータス バーを非表示にできます。

- ・ **[View] → [Panels]：** 次の ISim パネルを非表示または復元します。
  - [Console]
  - [Search Results]
  - [Source Files]
  - [Breakpoints]
  - [Objects]
  - [Instances and Processes]
  - [Find in Files Results]
- ・ **[View] → [Toolbar]：** 次の ISim ツールバーを非表示または復元します。
  - [Standard]
  - [Edit]
  - [View]
  - [ISim]
  - [Window]
  - [Help]
- ・ **[View] → [Status Bar]：** メイン ウィンドウの下部に配置されているステータス バーを非表示または復元します。





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 ウィンドウ](#)の右上のアイコンから実行できます。

### トグル アイコン

トグル アイコンは、[Instances and Processes] パネルや [Objects] パネルなどの ISim ウィンドウのパネルの右上に配置されています。





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。

-  **[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。  
フロートしているパネルを復元するにはもう一度クリックします。
-  **[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。

## 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。

### 矢印マークのクリック

-  : 矢印マーク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。
-  : 矢印マークをクリックして階層を非展開します。

### メニュー コマンド

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。



## コンテキスト 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。

## 波形ウィンドウ

### 波形ウィンドウの概要

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、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、「[シミュレーション結果の保存](#)」を参照してください。

### Default.wcfg の詳細

Default.wcfg ファイルは、GUI またはバッチ モードで ISim を起動すると作成されます。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 を終了するとき、[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。保存しない場合は、Default.wcfg の内容はディスクに保存されません。


### アクティブ ウィンドウ

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。ウィンドウ タブをクリックするか、または wcfg select 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] 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 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。









## 波形コンフィギュレーションの信号/バス

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。

### 信号

	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、およびジェネリック信号
	変数
	リンケージ信号
	バッファ信号

### バス

	入力バス
	出力バス
	入出力、双方向バス
	内部バス
	定数、パラメータ、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 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。詳細は「[カーソルの配置](#)」を参照してください。


- ・ **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。
- ・ **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**：マーカーは、後で参照できるように特定の時間に印を付けるために使用します。マーカーは、波形と交差する縦線です。マーカーでは、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、「[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](#)」を参照してください。

**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き、遷移間を移動する際に中空円 (○) 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仕切りを使用します。詳細は、「[仕切りの追加](#)」を参照してください。

**グループ**：グループは、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にまとめ、整理する方法です。グループには、 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、「[グループの追加](#)」を参照してください。

**仮想バス**：仮想バスは、論理スカラおよび配列をまとめるグループまたはフォルダです。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、「[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

波形ウィンドウのツールバー アイコンでは、よく使用するコマンドを簡単に実行できます。

-  [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。
-  [Next 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

### 新しい波形コンフィギュレーションの作成

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

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。

波形コンフィギュレーションが複数開いている場合は、波形コンフィギュレーションのタブをクリックするか、[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

グラフィカル ユーザー インターフェイスのメニュー 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 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。

**メモ:** 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。
  - ・ 次に示すように [Cosole] タブに [wave add コマンド](#)を入力します。

### 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] パネルの `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。
2. [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あるのに対し、.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。これら 2 つは Word ドキュメントと .doc ファイルの関係と同じです。


### 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。波形コンフィギュレーションの名前は基本的に .wcfg ファイル名と関係がないですが、波形コンフィギュレーションの読み込みコマンドや保存コマンドでは波形コンフィギュレーションが .wcfg ファイルに関連付けられ、ファイル ディレクトリのリストに表示されるので、波形コンフィギュレーション名とその .wcfg ファイル名の間には関係があります。GUI Tcl コマンドで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名は、ファイルに保存されるときに .wcfg ファイル名と一致するように変更されます。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。

## 波形コンフィギュレーションの保存

作業中の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

[File] → [Save] をクリックするか、Ctrl + S キーを押すか、または [Save] ボタン (  ) をクリックします。

作業中のシミュレーションの波形コンフィギュレーションが保存されます。

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

## 波形コンフィギュレーションを開く、閉じる

### 波形コンフィギュレーション ファイルを開くには

波形コンフィギュレーション ファイルを開く方法は多数あります。次を参照してください。

- ・ [ライブ シミュレーションを開く](#)
- ・ [スタティック シミュレーションを開く](#)

### 波形コンフィギュレーション ファイルを閉じるには

波形コンフィギュレーションおよびすべてのウィンドウを閉じるには、[File] → [Close] をクリックします。

## [Instances and Processes] パネル

### [Instances and Processes] パネルの概要

[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。インスタンス化されてエラポレートされたエンティティ/モジュールはツリー構造で表示されます。

このパネルには 3 列が含まれており、最初の列には、インスタンス、プロセス、およびスタティック タスク/ファンクション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。2 列目では、1 列目のインスタンス、スタティック タスク/ファンクション、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)) が表示されます。3 列目では、インスタンス、スタティック タスク/ファンクション、またはプロセスの種類が表示されます。

次にこのパネルで表示されるアイテムに使用されるアイコンの説明を示します。

	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
	Verilog タスクまたはファンクション
	Verilog ブロック
	Verilog プロセス

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、「[階層の展開/非展開](#)」を参照してください。

[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。

## 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

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。

5. 必要な場合は、[Match case] をオンにします。
6. [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 ソース ファイルをテキスト エディタで開いて表示できます。ファイルはすべて読み取り専用で開きます。

### HDL ソース ファイルを開くには

1. [\[Instances and Processes\] パネル](#)、[\[Objects\] パネル](#)、または [\[Source Files\] パネル](#) でアイテムを 1 つ選択します。
2. 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 エディタで開きます。ただし、読み取り専用です。

[File] → [Open] をクリックしてもファイルを開くことができます。[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。この方法でファイルを開いた場合は、読み取り専用になりません。

## [Objects] パネル

### [Objects] パネルの概要

[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、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。



## [Objects] パネルのツールバー



入力ポートの表示/非表示を切り替えます。



出力ポートの表示/非表示を切り替えます。



入出力、双方向ポートの表示/非表示を切り替えます。



内部信号の表示/非表示を切り替えます。



定数、パラメータ、およびジェネリックの表示/非表示を切り替えます。



変数の表示/非表示を切り替えます。



[Sync Time] 機能をオン/オフを切り替えます。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。

## 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

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。
5. 必要な場合は、[Match case] をオンにします。
6. [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。

## [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を表示します。このコマンドを使用して値変更の原因を突き止め、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を表示します。

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)

### ドライバを表示するには

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が表示されます。ドライバがない場合は、その旨を伝えるメッセージが表示されます。

メモ：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

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更新されます。

### 子エレメントの表示数を制限するには

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. [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. 数値を入力します。
5. [Apply] をクリックしてから [OK] 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更新されます。

## 波形ウィンドウでのオブジェクトの選択

次の手順に従い [Objects] パネルでオブジェクトを選択し、その信号をハイライトします。

### 波形ウィンドウでオブジェクトを選択するには

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。


オブジェクトに関連する信号がハイライトされます。

## [Source Files] パネル

### [Source Files] パネルの概要

[Source Files] パネルには、デザインに関連するファイルのリストが表示されます。ファイルのリストは、GUI のバックグラウンドで実行されるデザインの解析およびエラボレーション中に **fuse** コマンドにより供給されます。HDL ソース ファイルは、ソース コードを読み取り専用で開くことができます。

## ソース コードを開くには

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。

**メモ:** 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。

テキスト エディタ ウィンドウに選択したファイルが読み取り専用で開きます。

## テキスト エディタ ウィンドウ

### テキスト エディタ ウィンドウの概要

ISim のテキスト エディタ ウィンドウを使用すると、基本 HDL ソース ファイルを簡単に表示できます。次の基本ステップを参照してください。

- ・ [HDL ソース ファイルを開く](#)
- ・ [HDL ソース ファイルを表示する](#)
- ・ ソース ファイルに [ブレークポイントを設定](#)してデバッグする

HDL ファイルは ISim のテキスト エディタで編集しないでください。ファイルを編集すると ISE® ソフトウェアのプロジェクトと競合する可能性があります。

## ソース ファイルを安全に変更するには

次の手順に従うと、ソース ファイルを安全に変更できます。デザインの競合を回避するには、必ず HDL ファイルを ISim 外で編集してください。

1. ISE で ISE Text Editor またはサードパーティのテキスト エディタからソース ファイルを開きます。
2. 必要に応じて編集します。
3. ISE ツールでデザインを実行して、デザインをアップデートします。
4. デザインをシミュレーションします。

## HDL ソース ファイルを開く

ISim では、HDL ソース ファイルをテキスト エディタで開いて表示できます。ファイルはすべて読み取り専用で開きます。

### HDL ソース ファイルを開くには

1. [\[Instances and Processes\]](#) パネル、[\[Objects\]](#) パネル、または [\[Source Files\]](#) パネルでアイテムを 1 つ選択します。
2. 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 エディタで開きます。ただし、読み取り専用です。

[File] → [Open] をクリックしてもファイルを開くことができます。[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。この方法でファイルを開いた場合は、読み取り専用になりません。

## HDL ソース ファイルを表示する

ISim では、[テキスト エディタ ウィンドウ](#)で基本の HDL ソース ファイルを表示し、デザインを検証できます。

標準のテキスト エディタでは、拡大表示、検索などを使用して HDL ファイルを表示できます。操作に関する情報は、ISE Text Editor ヘルプを参照してください。

HDL ファイルは ISim のテキスト エディタで編集しないでください。ファイルを編集すると ISE® ソフトウェアのプロジェクトと競合する可能性があります。

### ソース ファイルの内容を表示するには

1. [HDL ソース ファイルを開きます](#)。
2. スクロール、検索、レイアウトのプリファレンスなどの機能を使用して、ファイルの内容を表示します。

HDL ファイルは ISim のテキスト エディタで編集しないでください。ファイルを編集すると ISE ソフトウェアのプロジェクトと競合する可能性があります。

**メモ** : ISim で HDL ファイルを変更した場合は、保存しないでください。

## ブレークポイントの設定

ISim では HDL ファイルの実行行にブレークポイントを設定できます。ブレークポイントを設定すると、「[ブレークポイントを使用したデザインのデバッグ](#)」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。

**メモ** : 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。

### ブレークポイントを設定するには

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。

**メモ** : 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。また、ブレークポイントをクリックすると削除できます。

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。

**メモ** : 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リストは、[\[Breakpoints\] パネル](#)に表示されます。

## メモリ エディタ ウィンドウ

### ISimメモリ エディタ ウィンドウの概要

メモリ エディタでは、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。次に、ISim でメモリ エディタを開く方法を 3 つ示します。

- ・ [Memory] パネルの [Memory] タブをクリックすると、デザインに含まれている 2 次元アレイのロジック タイプがすべて表示されます。2 次元アレイのロジックを右クリックして、[Memory Editor] をクリックします。
- ・ [Objects] パネルを表示される 2 次元アレイのロジックを右クリックして、[Memory Editor] をクリックします。
- ・ [Instance and Processes] パネルで検索機能を使用してメモリ名を検索します。[Search Results] タブに検出されたメモリが表示されるので、このメモリを右クリックして [Memory Editor] をクリックします。

選択した 2 次元アレイに対応するメモリ エディタが波形ビューアやテキスト エディタ タブと共にメイン ウィンドウに表示されます。

**メモ** : 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淡色表示されます。

- ・ **[Address]** : メモリ エディタ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 エディタで表示される値の基数を選択します。

メモリ エディタ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] パネルの概要

[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。

- ・ **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 インターフェイス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、「[シミュレーション コマンドの入力](#)」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。これらのコマンドには、切り取り、コピー、貼り付け、コンソールの消去などが含まれています。

## [Breakpoints] パネル




### [Breakpoints] パネルの概要

[Breakpoints] パネルでは、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。ブレイクポイントは、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。詳細は、「[ブレイクポイントを使用したデザインのデバッグ](#)」を参照してください。

また、「[ブレイクポイントの設定](#)」および「[ブレイクポイントの削除](#)」も参照してください。

[Breakpoints] パネルには、ソースファイルに設定されている各ブレイ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コンテキストメニューを使用して、選択したブレイクポイントまたはすべてのブレイクポイントを削除したり、ソースコードに移動できます。

### [Breakpoints] パネルのツールバー

-  [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イ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イクポイントをすべて削除します。
-  [Go To Source Code] : テキスト エディタで HDL ソース ファイルを開きブレイクポイントを示します。






## [Search Results] パネル

### [Search Results] パネルの概要

[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

[Search Results] パネルでは、次の機能を使用できます。




-  [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





### [Find in Files] コマンドの使用

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。

## 複数のファイルから文字列を検索するには

1. [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。
2. [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。
3. [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。

**メモ：** 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。
  - ・ 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 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。





## ツールバー コマンドおよびショートカット

### ISim ツールバー コマンド

ISim メイン ウィンドウで使用可能なツールバーは、さまざまな機能別ツールバーで構成されています。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。









### 標準ツールバー

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。[View] → [Toolbars] → [Standard] をクリックすると、表示/非表示を切り替えることができます。

-  [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。
-  [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。
-  [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。[View] → [Toolbars] → [Edit] をクリックすると、表示/非表示を切り替えることができます。

-  [Cut] : ワークスペースで選択したテキストまたはオブジェクトを切り取り、クリップボードに保存します。
-  [Copy] : ワークスペースで選択したテキストまたはオブジェクトをクリップボードにコピーします。
-  [Paste] : クリップボードのテキストまたはオブジェクトを作業中のウィンドウに貼り付けます。
-  [Delete] : ワークスペースまたは [Transcript] ウィンドウに表示されたテキストやオブジェクトを削除します。
-  [Unselect All] : アクティブ ウィンドウで選択されているものがすべて選択解除されます。
-  [Undo] : 最後の操作を元に戻します。このコマンドは、元に戻すことができるコマンドが実行された後にのみ使用できます。
-  [Redo] : [Undo] を使用して元に戻したコマンドをやり直します。
-  [Find] : [Find] ボックスが表示され、作業中のウィンドウ内で文字列を検索します。





## [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。[View] → [Toolbars] → [Help] をクリックすると、表示/非表示を切り替えることができます。

-  [Support and Services] : ザイリンクスのサポート ページがデフォルトの Web ブラウザで表示されます。
-  [What's This] : ヘルプを必要とするメニュー項目やツールバー 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。

## [Window] 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。[View] → [Toolbars] → [Window] をクリックすると、表示/非表示を切り替えることができます。

-  [Cascade] : 作業中のウィンドウを一番上にして、すべてのウィンドウを重ねて表示します。
-  [Tile Horizontally] : ウィンドウを重ねないように上下に並べて表示します。
-  [Tile Vertically] : ウィンドウを重ねないように左右に並べて表示します。
-  [Float] : 作業中のファイルをフロートさせます。

## [View] ツールバー









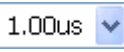




[View] ツールバーを使用すると、頻繁に使用する [View] メニュー コマンドに簡単にアクセスできます。[View] → [Toolbars] → [View] をクリックすると、表示/非表示を切り替えることができます。



-  [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 Out] : 表示しているオブジェクトのサイズを縮小します。
-  [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。
-  [Refresh] : ファイルの表示を更新します。

## ISim ツールバー

デザイン サマリのツールバーを使用すると、頻繁に使用するコマンドに簡単にアクセスできます。[View] → [Toolbars] → [ISim] をクリックすると、表示/非表示を切り替えることができます。

-  [Go To Previous Transition] : メインカーソルを 1 つ前の遷移に移動します。
-  [Go To Next Transition] : メインカーソルを次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。
-  [Next 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。
-  [Restart] : シミュレーション時間を 0 にリセットします。
-  [Run All] : 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、ブ레이크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 for the time specified on the toolbar] : 指定した時間だけシミュレーションを実行します。
-  [Run for the time specified on the toolbar] コマンド (  ) を実行するときのシミュレーション時間を指定します。矢印ドロップダウンをクリックすると、以前に使用した値がリスト表示されます。
-  [Step] : 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。
-  [Break] : 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。
-  [Quit Simulation] : 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。



## ショートカット

### ショートカット キー

ショートカット キー	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]
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]

## スティミュラスの適用

### [Force Selected Signal] ダイアログ ボックス

[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。HDL コードで割り当てられた値や、以前に適用された isim force コマンド値は、新しく割り当てられる定数で上書きされます。

[Apply] をクリックし、すべての変更を反映させます。

#### [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]

強制的に適用する定数値を指定します。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]

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。10、10ns などを入力できます。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。

### [Define Device] ダイアログ ボックス

[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、すべての変更を反映させます。

#### [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。

**メモ** : 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 番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。10、10ns などを入力できます。単位なしで数字が入力された場合、Tcl コマンドの 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 isim 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。

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 のプリファレンス

### 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

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]
  - ・ [ISim Simulator]
3. 設定に変更を加えます。
4. 変更後、[Apply] をクリックします。
5. 変更後の設定を確認し、[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、で開いている 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 [ISim Simulator] ページでは、ISim の基本的なプリファレンスを設定できます。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、「[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] タブ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。

## [Source Properties] ダイアログ ボックス – [Hardware Co-Simulation Properties] ページ

**メモ：** ハードウェア協調シミュレーションは限定カスタム向け (LCA) の機能であり、特別なライセンスが必要です。この機能をイネーブルにする方法については、フィールド アプリケーション エンジニア (FAE) までお問い合わせください。

ISim を使用してビヘイビア シミュレーションを実行する際に、選択したソース ファイルに対してハードウェア協調シミュレーションをイネーブルにします。[Design] パネルの [Hierarchy] ペイン、[Files] パネル、または [Libraries] パネルでソース ファイルを選択し、[Source] → [Source Properties] をクリックすると表示されます。

### [Enable Hardware Co-Simulation]

選択したインスタンスに対してハードウェア協調シミュレーションをイネーブルにします。

**メモ：** ハードウェア協調シミュレーションをイネーブルにできるのは 1 つのインスタンスのみです。あるインスタンスに対してハードウェア協調シミュレーションをイネーブルにすると、以前にイネーブルにしていたインスタンスではディスエーブルになります。

**[Clock Port]**

インスタンスのクロック ポートの名前を指定します。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]**

ハードウェア協調シミュレーションに使用するターゲット ボードの名前を指定します。

**[Reuse Last Bitstream File]**

前回のハードウェア協調シミュレーションからのビットストリーム ファイルを使用するかどうかを指定します。

**メモ** : このオプションを使用するには、今回と前回のシミュレーションで同じインスタンスに対してハードウェア協調シミュレーションがイネーブルになっている必要があります。

## 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。色の変更に関する詳細は、[カスタム カラー スキームの設定] を参照してください。

[Apply] をクリックし、すべての変更を反映させます。

**[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。

**[You can edit the color of this scheme]**

この表には、ISim メイン ウィンドウでカスタマイズ可能なアイテムの色選択が含まれています。[Apply] をクリックして、変更を反映させます。

## 時間フォーマットのプリファレンス

ISim GUI で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 ウィンドウ内の GUI に適用されます。次の 3 つのフォーマットを設定できます。

**[Rulers]**

波形ウィンドウ上部のメイン ルーラーおよびフローティング ルーラーに適用されます。ほとんどのユーザーは、このオプションのみ設定する必要があります。

**[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 ウィンドウ以外の 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します。[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。

**次のボタンをクリックすると、時間のフォーマット設定すべてに適用されます。**

**[Reset To Defaults]**

値をデフォルト設定に戻します。

## VHDL シミュレーション

---

### VHDL シミュレーションの概要

コマンドラインからシミュレーションを実行するときの基本的な手順は、次の 3 つです。

1. デザイン ファイルの解析
2. ISim 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

詳細は、次のトピックを参照してください。

- ・ [コマンドラインからの論理シミュレーションの実行](#)
- ・ [コマンドラインからのタイミング シミュレーションの実行](#)

### コマンドラインからの論理シミュレーションの実行 (VHDL デザイン)

#### 方法 1: プロジェクト ファイルの使用 (推奨)

##### デザイン ファイルの解析

<proj\_name>.prj という名前のファイルを作成して、次の構文を含めます。

```
vhdl <library_name> <file_name_1>.vhd
vhdl <library_name> <file_name_2>.vhd
.
vhdl <library_name> <file_name_n>.vhd
```

説明:

- ・ <library\_name>: 指定行のソースがコンパイルされるライブラリを指定します。  
<library\_name> は、デフォルトのライブラリ work 以外を指定するときのみ必要です。
- ・ <file\_name\_1>.vhd: ソース ファイル名を指定します。1 行に 1 つの VHDL ファイルのみを含めることができます。

例:

```
vhdl work top.vhd
vhdl mylib_for_testbench testbench.vhd
```

メモ: 最上位ファイル testbench.vhd には、testbench という名前のエンティティが含まれています。

### ISim シミュレーション実行ファイルの生成 : fuse の実行

HDL リンカ fuse では解析ノードでデザインのスタティック エラボレーションの実行、各モジュール インスタンスのオブジェクトコードの生成、および生成したオブジェクトコードの ISim シミュレーション エンジン ライブラリへのリンク付けを実行して、ISim シミュレーション実行ファイルを作成します。

構文 :

```
fuse {[<library_name> .]<top_name>} -prj <proj_name>.prj -o  
<output_file_name>
```

説明 :

- ・ `{[<library_name> .]<top_name>}` : ライブラリおよび最上位のデザイン ユニット名を指定します。`<library_name>` は、デフォルトのライブラリ `work` 以外を指定するときのみ必要です。指定されている場合は、最上位を含む HDL ファイルをプロジェクトファイルで使用されている関連ライブラリ名でコンパイルする必要があります。たとえばテスト ベンチファイルのデザイン ユニット名を入力します。
- ・ **-o** : オプションで ISim 実行ファイルの名前を指定できます。このオプションを使用しないと、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。

プロジェクト ファイルを指定した fuse コマンドの例

```
fuse mylib_for_testbench.testbench -prj proj_name.prj
```

fuse コマンドの詳細は、「[fuse コマンドの概要と構文](#)」を参照してください。

## 方法 2 : vhpcomp を使用したファイルの解析

### デザイン ファイルの解析

構文 :

```
vhpcomp [-work] <library_name> <file_name>.vhd
```

説明 :

- ・ **-work** : オプションで、デフォルトの `work` ライブラリ以外を指定する場合に使用する必要があります。
- ・ `<library_name>` : `<file_name>` で指定されたソースがコンパイルされるライブラリを指定します。1 行に複数の VHDL ファイル名を指定できます。

例 :

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

このコマンドの詳細は、「[vhpcomp コマンドの概要と構文](#)」を参照してください。

### ISim シミュレーション実行ファイルの生成 : fuse の実行

構文 :

```
fuse {[<library_name> .]<top_name>} -o <output_file_name>
```



説明 :

- ・ `{<library_name>}.<top_name>` : ライブラリおよび最上位のデザイン ユニット名を指定します。`<library_name>` は、デフォルトのライブラリ `work` 以外を指定するときのみ必要です。指定されている場合は、最上位を含む HDL ファイルをプロジェクトファイルで使用されている関連ライブラリ名でコンパイルする必要があります。たとえばテスト ベンチファイルのデザイン ユニット名を入力します。

**メモ :** ビヘイビア デザインで UNISIM プリミティブがインスタンス化される場合、`glbl` を `top_name` として使用する必要があります。

- ・ `-o` : シミュレーション実行ファイルの名前 (`my_sim.exe` など) を指定できます。このオプションを使用しないと、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 work.topunit work.glbl -o my_sim.exe
```

`fuse` コマンドの詳細は、「[fuse コマンドの概要と構文](#)」を参照してください。

## シミュレーション

コンパイルおよび ISim シミュレーション実行ファイルの生成が完了したら、シミュレーションを実行します。シミュレーションを実行するには、`fuse` で生成した実行ファイルを実行します。

コマンド	動作
<code>x.exe</code> (デフォルト名) または <code>my_sim.exe</code> などのユーザー定義の実行ファイル	デザインがシミュレーションされ、完了すると Tcl コマンド プロンプトが開き Tcl コマンドが入力できるようになります。
<code>x.exe -gui</code> または <code>my_sim.exe -gui</code>	デザインがシミュレーションされ、完了すると ISim GUI が起動します。GUI のコマンドおよび Tcl コマンドを使用してデザインの解析、シミュレーションの再実行などを実行できます。
<code>x.exe -tclbatch &lt;tcl_file_name&gt;</code> または <code>my_sim.exe -tclbatch &lt;tcl_file_name&gt;</code>	デザインがシミュレーションされ、Tcl ファイルで指定されている Tcl コマンドが実行されます。最後に実行されるコマンドは <code>quit</code> です。

このコマンドの詳細は、「[ISim シミュレーション実行ファイル コマンドの概要と構文](#)」を参照してください。

## コマンド ラインからのタイミング シミュレーションの実行 (VHDL デザイン)

### タイミング シミュレーション モデルの生成

タイミング シミュレーションを起動する前に、バックアノテーション用にタイミング シミュレーション モデルおよび遅延ファイルが必要です。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』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。

### 方法 1 : プロジェクト ファイルの使用 (推奨)

ファイルのコンパイル

`<proj_name>.prj` という名前のファイルを作成して、次の構文を含めます。

```

vhdl <library_name> <your_testbench>.vhd
vhdl <library_name> <topleve_timesim>.vhd
.
vhdl <library_name> <file_name_n>.vhd

```

- ・ *<library\_name>*: 指定行のソースがコンパイルされるライブラリを指定します。デフォルトのライブラリ名は *work* です。
- ・ *<your\_testbench>.vhd*: スティミュラス ファイルを指定します。
- ・ *<topleve\_timesim>.vhd*: NetGen で生成されるタイミング シミュレーション モデルを入力します (「タイミング シミュレーション モデルの生成」を参照)。
- ・ *<file\_name\_n>.vhd*: 補助テストベンチ ファイルなど、テストベンチに必要なソース ファイルを指定します。

### ISim シミュレーション実行ファイルの生成 : fuse の実行

HDL リンカ *fuse* では解析ノードでデザインのスタティック エラボレーションの実行、各モジュール インスタンスのオブジェクト コードの生成、および生成したオブジェクト コードの ISim シミュレーション エンジン ライブラリへのリンク付けを実行して、ISim シミュレーション実行ファイルを作成します。

構文 :

```
fuse {[<library_name> .]<top_name>} -prj <proj_name>.prj -o
<output_file_name>
```

説明 :

- ・ *{[<library\_name> .]<top\_name>}*: ライブラリおよび最上位のデザイン ユニット名を指定します。ライブラリ名はオプションで、指定されていない場合はデフォルトで *work* になります。指定されている場合は、最上位を含む HDL ファイルをプロジェクト ファイルで使用されている関連ライブラリ名でコンパイルする必要があります。たとえばテストベンチ ファイルのデザイン ユニット名を入力します。

**メモ**: *glbl* は *top\_name* として使用する必要があります。

- ・ **-prj** <proj\_name>.prj: タイミング シミュレーション中はオプションです。
- ・ **-o**: オプションです。このオプションを使用しないと、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。

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

*fuse* コマンドの詳細は、「[fuse コマンドの概要と構文](#)」を参照してください。

## 方法 2 : vhpcomp の使用

ファイルの解析

構文 :

```

vhpcomp [-work <library_name> ]<file_name>.vhd
vhpcomp -work <library_name><file_name> .vhd

```

説明 :

- ・ **-work** : オプションで、デフォルトの work ライブラリ以外を指定する場合に使用する必要があります。
- ・ `<library_name>` : `<file_name>` で指定されたソースがコンパイルされるライブラリを指定します。1 行に複数の VHDL ファイル名を指定できます。

このコマンドの詳細は、「[vhpcomp コマンドの概要と構文](#)」を参照してください。

#### ISim シミュレーション実行ファイルの生成 : fuse の実行

構文 :

```
fuse {[<library_name>.]<top_name> }-o <output_file_name>
```

説明 :

- ・ `{[<library_name>.]<top_name>}` : ライブラリおよび最上位のデザイン ユニット名を指定します。たとえばテスト ベンチ ファイルのデザイン ユニット名を入力します。ライブラリ名を含めるのはオプションです。ライブラリ名が指定されない場合は、デフォルトのライブラリ名 work が使用されます。例 : `work.topunit` , `work.glbl` , `mylib.glbl`
- ・ **-o** : オプションです。このオプションを使用しないと、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。

fuse コマンドの詳細は、「[fuse コマンドの概要と構文](#)」を参照してください。

## シミュレーション

コンパイルおよび ISim シミュレーション実行ファイルの生成が完了したら、シミュレーションを実行します。fuse コマンドで生成された ISim シミュレーション実行ファイルを実行するとシミュレーションが開始します。

開始後にファイルに含まれる Tcl コマンドを実行する場合は、**-tclbatch** オプションを使用します。

また、SDF ファイルに含まれるタイミング遅延を使用するように指定することもできます。

構文 :

```
<executable_name>.exe -tclbatch <tcl_file_name>  
-sdfmin|-sdftyp|-sdfmax [<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。
- ・ **-sdfmin|-sdftyp|-sdfmax** : ISim で使用する遅延の種類 (minimum、typical、または maximum) を指定します。
- ・ `<instance>` : SDF 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。

このコマンドの詳細は、「[ISim シミュレーション実行ファイル コマンドの概要と構文](#)」を参照してください。

## ライブラリ マップ ファイル

**メモ** : 次の情報は、アドバンス ユーザーを対象としています。

ISim HDL コンパイル プログラム [vhpcomp](#)、[vlogcomp](#)、および [fuse](#) では、`xilinxsim.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。

### 検索順

コンパイラは、次のリストしたディレクトリ順に `xilinxsim.ini` ファイルを検索します。

1. `$XILINX/vhdl/hdp/<platform>`
2. **vlogcomp**、**vhpcomp**、または **fuse** コマンドの **-initfile** オプションで指定されたユーザー 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 `xilinxsim.ini` ファイル。

### 構文

`xilinxsim.ini` ファイルのフォーマットは、次のとおりです。

```
<logical_library1> = <physical_dir_path1>
<logical_library2> = <physical_dir_path2>
.
.
<logical_libraryn> = <physical_dir_pathn>
```

### 例

次に、`xilinxsim.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:/work
```

Verilog

```
unisims_ver=$XILINX/rtf/verilog/hdp/nt/unisims_ver
xilinxcorelib_ver=C:/libs/verilog/hdp/nt/xilinxcorelib_ver
mylib=./mylib
work=C:/work
```

## 機能/制限

xilinxsim.ini ファイルでは、次の点に注意してください。

- ・ xilinxsim.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 isim/<logical\_library\_name> です。
- ・ このファイルのコメントは、「--」で開始します。

## コマンドライン モードでの対話型シミュレーション

コマンドライン モードでシミュレーションを実行するときは、Tcl プロンプトでシミュレーション Tcl コマンドを入力して、シミュレーションを実行し、デザインを解析し、デザインをデバッグできます。シミュレーション コマンドの詳細は、「[シミュレーション コマンドの概要](#)」を参照してください。コマンドの入力方法のヒントは、「[シミュレーション Tcl コマンドの入力](#)」を参照してください。



## Verilog シミュレーション

---

### Verilog シミュレーションの概要

コマンドラインからシミュレーションを実行するときの基本的な手順は、次の 3 つです。

1. デザイン ファイルの解析
2. ISim 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

詳細は、次のトピックを参照してください。

- ・ [コマンドラインからの論理シミュレーションの実行](#)
- ・ [コマンドラインからのタイミング シミュレーションの実行](#)

### コマンドラインからの論理シミュレーションの実行 (Verilog デザイン)

Verilog デザインの論理 (ビヘイビア) シミュレーションで UNISIM プリミティブが使用される場合、次の手順に従う必要があります。

- ・ `$XILINX/Verilog/src/glbl.v` をライブラリ `work` にコンパイルします。
- ・ `fuse` で `work.glbl` を `<library_name>.<top_name>` の 1 つとして指定します。
- ・ `fuse` で `-L unisims_ver` を指定します。

#### 方法 1: プロジェクト ファイルの使用 (推奨)

##### デザイン ファイルの解析

`<proj_name>.prj` という名前のファイルを作成して、次の構文を含めます。

```
verilog <library_name> {<file_name_1>.v} {[-d <macro>] [-i  
<include_path>]}
```

```
verilog <library_name> {<file_name_2>.v} {[-d <macro>] [-i  
<include_path>]}
```

```
.
```

```
verilog <library_name> {<file_name_n>.v} {[-d <macro>] [-i  
<include_path>]}
```

説明 :

- ・ `verilog` : ソース ファイルが Verilog ファイルであることを示します。
- ・ `<library_name>` : 指定行のソースがコンパイルされるライブラリを指定します。1 行に複数の Verilog ソース ファイルを指定できます。
- ・ `[-d <macro>]` : `[-i <include_path>]` で指定されているロケーションに配置されているマクロを定義します。

例 :

```
verilog work top.v testbench.v
```

ISim シミュレーション実行ファイルの生成 : `fuse` の実行

HDL リンカ `fuse` では解析ノードでデザインのスタティック エラボレーションの実行、各モジュール インスタンスのオブジェクト コードの生成、および生成したオブジェクト コードの ISim シミュレーション エンジン ライブラリへのリンク付けを実行して、ISim シミュレーション実行ファイルを作成します。

構文 :

```
fuse { [<library_name> . ] <top_name> } -prj <proj_name>.prj -L  
<Verilog_library> -o <output_file_name>
```

説明 :

- ・ `{ [<library_name> . ] <top_name> }` : ライブラリおよび最上位のデザイン ユニット名を指定します。ライブラリ名はオプションで、指定されていない場合はデフォルトで `work` になります。指定されている場合は、最上位を含む HDL ファイルをプロジェクト ファイルで使用されている関連ライブラリ名でコンパイルする必要があります。たとえばテスト ベンチ ファイルのデザイン ユニット名を入力します。

**メモ** : ビヘイビア デザインで UNISIM プリミティブがインスタンス化される場合、`glbl` を `top_name` として使用する必要があります。

- ・ `-L <Verilog_library>` : ビヘイビア シミュレーションが UNISIM プリミティブに基づいている場合、`unimacro_ver` や `xilinxcorelib_ver` などのザイリンクス ライブラリに加えて `unisims_ver` を含める必要があります。
- ・ `-o` : オプションです。このオプションを使用しないと、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 work.test_bench work.glbl -prj mydesign.prj -L unisims_ver  
-L unimacro_ver -L xilinxcorelib_ver -o test_bench.exe
```

`fuse` コマンドの詳細は、「[fuse コマンドの概要と構文](#)」を参照してください。

## 方法 2 : `vlogcomp` を使用したファイルの解析

デザイン ファイルの解析

構文 :

```
vlogcomp [-work <library_name> ] <file_name>.v { [-d <macro>] [-i  
<include_path> ]
```



説明 :

- ・ **-work** : オプションで、デフォルトの work ライブラリ以外を指定する場合に使用する必要があります。
- ・ `<library_name>` : `<file_name>` で指定されたソースがコンパイルされるライブラリを指定します。1 行に複数の Verilog ソース ファイルを指定できます。
- ・ `[-d<macro>]` : `[-i <include_path>]` で指定されているロケーションに配置されているマクロを定義します。

例 :

```
vlogcomp suba.v subb.v
```

vlogcomp コマンドの詳細は、「[vlogcomp コマンドの概要と構文](#)」を参照してください。

ISim シミュレーション実行ファイルの生成 : fuse の実行

構文 :

```
fuse {[<library_name> .]<top_name>} {-L <Verilog_library>} -o  
<output_file_name>
```

説明 :

- ・ `{[<library_name> .]<top_name>}` にはライブラリおよび最上位のデザイン ユニット名を入力します。たとえばテスト ベンチ ファイルのデザイン ユニット名を入力します。そのうちの 1 つは `glbl` にする必要があります。ライブラリ名を含めるのはオプションです。ライブラリ名が指定されない場合は、デフォルトのライブラリ名 `work` が使用されます。

**メモ** : グローバル モジュールの詳細は、『合成./シミュレーション デザイン ガイド』の「Verilog でのグローバル セット (GSR) とグローバル トライステート (GTS)」を参照してください。

- ・ **-L <Verilog\_library>** : `unimacro` や `xilinxcorelib` などのザイリンクス ライブラリに加えて `unisim` を含める必要があります。
- ・ **-o** : オプションです。このオプションを使用しないと、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 work.test_bench work.glbl -L unisims_ver -L unimacro_ver -L  
xilinxcorelib_ver -o test_bench.exe
```

fuse コマンドの詳細は、「[fuse コマンドの概要と構文](#)」を参照してください。

## シミュレーション

コンパイルおよび ISim シミュレーション実行ファイルの生成が完了したら、シミュレーションを実行します。シミュレーションを実行するには、fuse で生成した実行ファイルを実行します。

コマンド	動作
<b>x.exe</b> (デフォルト名) または <b>my_sim.exe</b> などのユーザー定義の実行ファイル	デザインがシミュレーションされ、完了すると Tcl コマンド プロンプトが開き Tcl コマンドが入力できるようになります。
<b>x.exe -gui</b> または <b>my_sim.exe -gui</b>	デザインがシミュレーションされ、完了すると ISim GUI が起動します。GUI のコマンドおよび Tcl コマンドを使用してデザインの解析、シミュレーションの再実行などを実行できます。
<b>x.exe -tclbatch &lt;tcl_file_name&gt;</b> または <b>my_sim.exe -tclbatch &lt;tcl_file_name&gt;</b>	デザインがシミュレーションされ、Tcl ファイルで指定されている Tcl コマンドが実行されます。最後に実行されるコマンドは <b>quit</b> です。

このコマンドの詳細は、「ISim シミュレーション実行ファイル コマンドの概要と構文」を参照してください。

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

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` をライブラリ `work` にコンパイルします。
- ・ **fuse** で **work.glbl** を `<library_name>.<top_name>` の 1 つとして指定します。
- ・ **fuse** で **-L simprims\_ver** を指定します。

### タイミング シミュレーション モデルの生成

タイミング シミュレーションを起動する前に、バックアノテーション用にタイミング シミュレーション モデルおよび遅延ファイルが必要です。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』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。

### 方法 1 : プロジェクト ファイルの使用 (推奨)

ファイルのコンパイル

`<proj_name>.prj` という名前のファイルを作成して、次の構文を含めます。

```
verilog <library_name> {<file_name>.v} {[-d <macro>] [-i <include_path>]}
```

説明 :

- ・ *verilog* : ソース ファイルが Verilog ファイルであることを示します。1 行に複数の Verilog ソース ファイルを指定できます。
- ・ `<library_name>` : 指定行のソースがコンパイルされるライブラリを指定します。
- ・ `[-d <macro>]` : `[-i <include_path>]` で指定されているロケーションに配置されているマクロを定義します。これらのオプションはオプションです。

例 :

```
verilog work top.v testbench.v
```

```
verilog work glbl.v
```

```
verilog work top_timesim.v
```

ISim シミュレーション実行ファイルの生成 : fuse の実行

HDL リンカ fuse では解析ノードでデザインのスタティック エラボレーションの実行、各モジュール インスタンスのオブジェクトコードの生成、および生成したオブジェクトコードの ISim シミュレーション エンジン ライブラリへのリンク付けを実行して、ISim シミュレーション実行ファイルを作成します。

構文 :

```
fuse {[<library_name>.]<top_name>} -prj <proj_name>.prj {-L  
<verilog_library_name> }-o <output_file_name>
```

説明 :

- ・ `{[<library_name>.]<top_name>}` : ライブラリおよび最上位のデザイン ユニット名を入力します。たとえばテスト ベンチ ファイルのデザイン ユニット名を入力します。そのうちの 1 つは `glbl` にする必要があります。ライブラリ名を含めるのはオプションです。ライブラリ名が指定されない場合は、デフォルトのライブラリ名 `work` が使用されます。
- ・ **-L** <Verilog\_library> : ザイリンクス ライブラリに加えて **simsprims\_ver** を含める必要があります。
- ・ **-o** : オプションです。このオプションを使用しないと、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 work.testbench work.glbl -prj design.prj -L simprims_ver -o  
isim.exe
```

fuse コマンドの詳細は、「[fuse コマンドの概要と構文](#)」を参照してください。

## 方法 2 : vlogcomp を使用したファイルの解析

ファイルの解析

構文 :

```
vlogcomp [-work <library_name> ]<file_name>.v
```

説明 :

- ・ **-work** : オプションで、デフォルトの `work` ライブラリ以外を指定する場合に使用する必要があります。
- ・ <library\_name> : <file\_name> で指定されたソースがコンパイルされるライブラリを指定します。1 行に複数の Verilog ソース ファイルを指定できます。

例 :

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

vlogcomp コマンドの詳細は、「[vlogcomp コマンドの概要と構文](#)」を参照してください。

ISim シミュレーション実行ファイルの生成 : fuse の実行

構文 :

```
fuse {[<library_name>.]<top_name>} {-L <Verilog_library> } -o  
<output_file_name>
```

説明 :

- ・ `{<library_name>}<top_name>` : ライブラリおよび最上位のデザイン ユニット名を入力します。たとえばテスト ベンチ ファイルのデザイン ユニット名を入力します。そのうちの 1 つは `glbl` にする必要があります。ライブラリ名を含めるのはオプションです。ライブラリ名が指定されない場合は、デフォルトのライブラリ名 `work` が使用されます。
- ・ `-L <Verilog_library>` : ザイリンクス ライブラリに加えて `simsprims_ver` を含める必要があります。
- ・ `-o` : オプションです。このオプションを使用しないと、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 work.textbench work.glbl -L simprims_ver -o timesim.exe
```

`fuse` コマンドの詳細は、「[fuse コマンドの概要と構文](#)」を参照してください。

## シミュレーション

コンパイルおよび ISim シミュレーション実行ファイルの生成が完了したら、シミュレーションを実行します。`fuse` コマンドで生成された ISim シミュレーション実行ファイルを実行するとシミュレーションが開始します。

開始後にファイルに含まれる Tcl コマンドを実行する場合は、`-tclbatch` オプションを使用します。

また、SDF ファイルに含まれるタイミング遅延を使用するように指定することもできます。

構文 :

```
<executable_name>.exe -tclbatch <tcl_file_name>  
-sdfmin|-sdftyp|-sdfmax [<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。
- ・ `-sdfmin|-sdftyp|-sdfmax` : ISim で使用する遅延の種類 (minimum、typical、または maximum) を指定します。
- ・ `<instance>` : SDF 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。

このコマンドの詳細は、「[ISim シミュレーション実行ファイル コマンドの概要と構文](#)」を参照してください。

## Verilog デザイン ユニットのインスタンスの検索順位

HDL リンカ fuse では、次の検索順位を使用してデザインにインスタンシエートされている Verilog デザイン ユニットを検索しバインドします。

1. 'uselib 指示子で指定されたライブラリ
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. ローカルの作業ライブラリ

## ソース ライブラリのサポート

次に示すコンパイラの引数では、Verilog-XL と同様にソース ライブラリがサポートされます。各引数の説明は、「[vlogcomp コマンドのオプション](#)」または「[fuse コマンドのオプション](#)」を参照してください。

この機能を使用するには、次のコマンド オプションを vlogcomp Verilog コンパイラに渡す必要があります。

### ライブラリのディレクトリ (-sourcelibdir)

**メモ** : **-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 ファイルがコンパイル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、コンパイラでソース ライブラリが検索されます。この検索中に、コンパイラは未解決のインスタンシエートされたデザイン 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、コンパイラでそのファイルが解析されます。コンパイラでは **-sourcelibext** が共に使用されない場合は、拡張子が .v、.h などのファイルがデフォルトで無視されることに注意してください。

**-sourcelibdir <library\_first> -sourcelibdir <library\_second>**

### ソース ファイルの拡張子 (-sourcelibext)

**メモ** : **-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。

**-sourcelibext .v**

### ソース ファイル (-sourcelibfile)

**メモ** : **-sourcelibfile** では Verilog-XL の **-v** オプションに類似した機能が提供されます。

ISim では、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 ファイルを供給できます。

**-sourcelibfile ./library/lib\_abc.v**

#### バッチモードの例

次に、これらのコマンド オプションの使用例を示します。

vlogcomp

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

コンパイラによりディレクトリ `mydir/cells` に含まれる未解決のセルが検索されます。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、コンパイラにより `mydir/cells` ディレクトリに含まれる名前が DFF および DMUX のファイルが検索されます。ファイル DFF および DMUX では、モジュール DFF および DMUX が定義されているはずです。

`fuse`

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

`test.prj` には次が含まれます。

```
verilog work test.v
```

コンパイラでは `test.v` で使用されるモジュールの `-sourcelibfile` オプションで渡されるファイルが使用されます。モジュールが解析され、`test` デザインが生成されます。

<proj\_name>.prj

```
fuse -prj test.prj test
```

`test.prj` には次が含まれます。

```
verilog work test.v -sourcelibdir ./mylib1 -sourcelibdir ./mylib2  
-sourcelibext .v
```

`test.v` ファイルにインスタンス化されている `modulename` 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。

## ライブラリ マップ ファイル

**メモ：** 次の情報は、アドバンス ユーザーを対象としています。

ISim HDL コンパイル プログラム `vhpcomp`、`vlogcomp`、および `fuse` では、`xilinxsim.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。

### 検索順

コンパイラは、次のリストしたディレクトリ順に `xilinxsim.ini` ファイルを検索します。

1. `$XILINX/vhdl/hdp/<platform>`
2. **vlogcomp**、**vhpcomp**、または **fuse** コマンドの **-initfile** オプションで指定されたユーザー 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 `xilinxsim.ini` ファイル。

## 構文

xilinxsim.ini ファイルのフォーマットは、次のとおりです。

```
<logical_library1> = <physical_dir_path1>
<logical_library2> = <physical_dir_path2>
.
.
<logical_libraryn> = <physical_dir_pathn>
```

## 例

次に、xilinxsim.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:/work
```

Verilog

```
unisims_ver=$XILINX/rtf/verilog/hdp/nt/unisims_ver
xilinxcorelib_ver=C:/libs/verilog/hdp/nt/xilinxcorelib_ver
mylib=./mylib
work=C:/work
```

## 機能/制限

xilinxsim.ini ファイルでは、次の点に注意してください。

- ・ xilinxsim.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 isim/<logical\_library\_name> です。
- ・ このファイルのコメントは、「--」で開始します。

## Verilog シミュレーション用の定義済み XILINX\_SIM マクロ

XILINX\_ISIM は、ISim 特有の 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
```

## コマンドライン モードでの対話型シミュレーション

コマンドライン モードでシミュレーションを実行するときは、Tcl プロンプトでシミュレーション Tcl コマンドを入力して、シミュレーションを実行し、デザインを解析し、デザインをデバッグできます。シミュレーションコマンドの詳細は、「[シミュレーション コマンドの概要](#)」を参照してください。コマンドの入力方法のヒントは、「[シミュレーション Tcl コマンドの入力](#)」を参照してください。



# 混合言語シミュレーション

## 混合言語シミュレーションの概要

メモ：次の情報は、アドバンス ユーザーを対象としています。

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 モジュールが含まれます。ただし、一部制限があります。

### 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化、Verilog デザインへの VHDL コンポーネントのインスタンス化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、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化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化されている Verilog モジュールには、コンフィギュレーション仕様、直接インスタンス化、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。 [詳細情報](#)

### 混合言語シミュレーションでの主要手順

- ・ 混合言語コンポーネントをインスタンス化します。詳細は、「[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](#)」または「[Verilog モジュールへの VHDL デザイン ユニットのインスタンス化](#)」を参照してください。
- ・ 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。

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。**-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。 [詳細情報](#)

- ・ シミュレーションを実行します。

## 混合言語コンポーネントのインスタンス化

### 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

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

#### VHDL デザイン ユニットに Verilog モジュールをインスタンス化するには

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

次の例です。

```
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。

次の例です。

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

「[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](#)」のポート マップ規則を参照し、ポート タイプが正しく一致しているか確認してください。

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

混合言語デザインでは、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。

#### 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

Verilog モジュールのインスタンス化と同様に、VHDL エンティティをインスタンス化します。

次の例です。

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

## 混合言語デザインでのバインドと検索

メモ： 次の情報は、アドバンス ユーザーを対象としています。

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** リンカでは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もう 1 つの言語のデザイン 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。Verilog ライブラリの検索順の詳細は、「[Verilog デザイン ユニットのインスタンスの検索順位](#)」を参照してください。

**メモ：** ISE® 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。

## VHDL インスタンス化 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、そのコンポーネント名は 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。VHDL ユニットが見つからない場合は、大文字と小文字を維持したコンポーネント名を Verilog モジュール名として扱い、次のように検索が続行されます。

- ・ 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 Verilog モジュールが大文字と小文字の区別を維持しながら検索されます。最初に一致したユニットがバインドされ、検索が終了します。
- ・ 大文字と小文字の区別を維持した検索条件で一致するユニットが見つからない場合は、大文字と小文字を区別しない名前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。

## Verilog インスタンス化 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、そのコンポーネント名は Verilog ユニットと判断され、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 ユニットが見つかり、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を VHDL エンティティ名として扱い、次のように検索が続行されます。

- ・ 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大文字と小文字の区別を無視して検索されます。最初に一致したユニットがバインドされ、検索が終了します。
- ・ 拡張識別子として作成された VHDL デザイン ユニット名（大文字と小文字を維持）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、検索は終了します。

**メモ：** 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。

## 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

**メモ：** 次の情報は、アドバンス ユーザーを対象としています。

## 一般

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。

## ポートのマッピング

混合言語プロジェクトで使用するポート 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 VHDL ポート タイプ：
  - IN
  - OUT
  - INOUT

**メモ：** 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。
- ・ サポートされる Verilog ポート タイプ：
  - INPUT
  - OUTPUT
  - INOUT

**メモ：** Verilog では、双方向パス 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。
- ・ 次の表に、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。

VHDL ポート	Verilog ポート
ビット	ネット
std_ulogic	ネット
std_logic	ネット
bit_vector	ベクタ ネット
std_ulogic_vector	ベクタ ネット
std_logic_vector	ベクタ ネット

**メモ：** 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。

**メモ：** 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）のマップ

次の VHDL ジェネリック型（および対応する Verilog の型）がサポートされます。

- ・ 整数
- ・ 実数
- ・ 文字列
- ・ ブール代数

**メモ：** 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 の値のマップ

次の表に示すように、Verilog のステートは std\_logic および bit 型にマップされます。

Verilog	std_logic	ビット
Z	'Z'	'0'
0	'0'	'0'
1	'1'	'1'
X	'X'	'0'

**メモ：** 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。

VHDL の bit 型は、次の表に示すように Verilog のステートにマップされます。

ビット	Verilog
'0'	0
'1'	1

VHDL の std\_logic 型は、次の表に示すように Verilog のステートにマップされます。

std_logic	Verilog
'U'	X
'X'	X
'0'	0
'1'	1
'Z'	Z
'W'	X
'L'	0
'H'	1
'_'	X



## 波形の解析

---

### 解析の実行前

#### ISim GUI の起動

##### GUI を起動するには

ISim の GUI は、シミュレーション実行ファイルを ISE® ソフトウェアまたはコマンドラインから実行すると起動します。詳細は、「[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。

ISim グラフィック ユーザー インターフェイスを読み取り専用モードで開いて以前のシミュレーションのデータを表示または解析するときの詳細は、「[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。

#### 操作後の結果

##### ISE ソフトウェアからのシミュレーション

ISE® から ISim を起動すると、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。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。次に[信号を追加](#)したり、[ISim でシミュレーションを実行](#)できます。

##### コマンドラインからのシミュレーション

コマンドラインでシミュレーション実行ファイルを **-gui** オプションで実行して ISim を起動すると、空の波形コンフィギュレーションが表示されます。デザイン 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。ISim 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。

### 波形コンフィギュレーションへの信号の追加

グラフィカル ユーザー インターフェイスのメニュー 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 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。

**メモ：** 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。
  - ・ 次に示すように [Cosole] タブに `wave add` コマンドを入力します。

## 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] パネルの `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。
2. [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あるのに対し、.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。これら 2 つは Word ドキュメントと .doc ファイルの関係と同じです。

## 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。波形コンフィギュレーションの名前は基本的に .wcfg ファイル名と関係がないですが、波形コンフィギュレーションの読み込みコマンドや保存コマンドでは波形コンフィギュレーションが .wcfg ファイルに関連付けられ、ファイル ディレクトリのリストに表示されるので、波形コンフィギュレーション名とその .wcfg ファイル名の間には関係があります。GUI Tcl コマンドで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名は、ファイルに保存されるときに .wcfg ファイル名と一致するように変更されます。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/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/バスのコピーは、**グループ**や**仮想バス**など、波形コンフィギュレーションの任意の位置に配置できます。

## 信号/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/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/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。

## ISim でのシミュレーションの実行

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 またはコマンドラインを使用して実行できます。

### ISim GUI からシミュレーションを実行するには

次の GUI メニューを使用して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 [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、[restart](#) Tcl コマンドを参照してください。
- ・ [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、Tcl コマンドの [run](#) で all オプションを実行しても同様の操作を実行できます。
- ・ [Simulation] → [Run] (▶): シミュレーションを 100ns 間実行するか、またはツールバーに特定のシミュレーション時間を入力して実行します。時間とその単位は、ボタンの右横にある [Value] ボックスに入力します。[run](#) Tcl コマンドを使用しても、シミュレーション時間およびその単位を指定できます。
- ・ [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。詳細は「[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。また、[step](#) Tcl コマンドも参照してください。

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、「[ソースコードのデバッグの概要](#)」を参照してください。

**メモ:** 現在のシミュレーション時間は、右下端に表示されます。

## 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。

ISim セッションを閉じる場合は、「[ISim の終了](#)」を参照してください。

### シミュレーションを一時停止するには

次の手順に従って [Break] コマンドを使用すると、シミュレーションを一時停止できます。

- ・ [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 エディタに表示されます。

**メモ:** 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、「[ISim でのシミュレーションの実行](#)」を参照してください。

## ISim の終了

シミュレーションを終了し、ISim セッションを閉じることができます。

### ISim を終了するには

- ・ [File] → [Exit] をクリックします。
- ・ [Console] パネルのプロンプトに **quit -f** コマンドを入力します。この場合、終了を尋ねるダイアログ ボックスが表示されずに ISim が終了します。
- ・ 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。

シミュレーションが終了し、ISim が閉じます。

## 波形コンフィギュレーションのカスタマイズ

### カーソルの配置

[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。

### メイン カーソルを配置するには

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 カーソルを配置するには

次の手順に従い、セカンダリ カーソルを配置します。


- ・ 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。
- ・ 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。

**メモ：** 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。

**メモ：** 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。

### カーソルを移動するには

 シンボルが表示されるまでマウスを移動してからカーソルをクリックして、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。

- ・ 塗りつぶされた円 (●) では、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。
- ・ 中空円 (○) では、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。

## マーカーの設定

### マーカーの追加

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。マーカー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。

#### マーカーを追加するには

1. 波形ウィンドウでマーカー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。

マーカーがカーソルに配置されます。マーカー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ーが配置されます。マーカーの時間はマーカー上部に表示されます。

### 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。

#### マーカーを移動するには

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 (●) では、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 (○) では、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。  
マーカーが新しい位置に移動します。

### 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。

#### マーカーを削除するには

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。

**メモ：** Delete キーを使用しても、選択したマーカーを削除します。

マーカーが波形から削除されます。

[Undo] コマンド ([Edit] → [Undo]) を使用すると削除したマーカーを復元できます。

## 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。

### 仕切りを追加するには

1. **波形ウィンドウ**の [Name] 列で、仕切りを追加する信号を選択します。
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。

仕切りが波形コンフィギュレーションに追加されます。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。

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。

仕切りでは、次を変更できます。

- ・ 仕切りに名前を付けることができます。詳細は、「[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。

## グループの追加


グループを波形コンフィギュレーションに追加すると、信号およびバスを関連信号セットとしてフォルダにまとめ、整理できます。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。

### グループを追加するには

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任意の数だけ選択します。

**メモ：** 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  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。

グループでは、次を変更できます。

- ・ グループの名前を変更できます。詳細は、「[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。

## 仮想バスの追加

仮想バスを波形コンフィギュレーションに追加すると、論理スカラおよび配列をグループまたはフォルダにまとめることができます。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。

### 仮想バスを追加するには

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。

選択した信号またはバスを含む仮想バス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  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、「[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、[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。

### オブジェクト名を変更するには

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、デザインソースコードには影響しません。

## 表示名の変更

名前は、階層名を含めた完全名で表示するか ([Long Name])、信号/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。

### 表示名を変更するには

- 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、「[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。

## 基数の変更

### デフォルトの基数を設定するには

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。デフォルトの基数を変更するには、次を実行します。

1. [Edit] → [Preferences] をクリックします。
2. [Preferences] ダイアログ ボックスの左側で [ISim Simulator] をクリックします。
3. [Default Radix] 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。

## 個々の基数を変更するには

[Object] 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] から次のサブメニューを選択します。
  - ・ [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]

**メモ：** [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、`isim set radix` Tcl コマンドを使用します。

## バス ビット順の反転

波形コンフィギュレーションではバス 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。

### バスのビット順を反転にするには

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

### 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。

### 矢印マークのクリック

- ▶：矢印マーク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。
- ▼：矢印マークをクリックして階層を非展開します。

### メニュー コマンド

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。






## コンテキスト 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。

## コマンドの使用

動作	コマンド	ショートカット キー
拡大	[View] → [Zoom] → [In] または [Zoom In] ボタン 	<ul style="list-style-type: none"> <li>・ F8 キーを押します。</li> <li>・ 波形コンフィギュレーションで Ctrl キーを押しながら、左下方向にマウスをドラッグして線を描画します。</li> </ul> <p><b>メモ</b>：Ctrl キー の代わりにマウスの中ボタンを使用できます。</p>
縮小	[View] → [Zoom] → [Out] または [Zoom Out] ボタン 	<ul style="list-style-type: none"> <li>・ F7 キーを押します。</li> <li>・ 波形コンフィギュレーションで Ctrl キーを押しながら、右上方向にマウスをドラッグして線を描画します。</li> </ul> <p><b>メモ</b>：Ctrl キー の代わりにマウスの中ボタンを使用できます。</p>
ウィンドウに波形全体を表示	[View] → [Zoom] → [To Full View] またはツールバーの [To Full View] ボタン 	<ul style="list-style-type: none"> <li>・ F6 キーを押します。</li> <li>・ 波形コンフィギュレーションで Ctrl キーを押しながら、左上方向にマウスをドラッグして線を描画します。</li> </ul> <p><b>メモ</b>：Ctrl キー の代わりにマウスの中ボタンを使用できます。</p>
選択した範囲全体を表示	なし	<p>波形コンフィギュレーションで Ctrl キーを押しながら左上端から右下端に向かってボックスを描画します。</p> <p><b>メモ</b>：Ctrl キー の代わりにマウスの中ボタンを使用できます。</p>



動作	コマンド	ショートカット キー
2 つのカーソル間の時間範囲を表示	1. 波形にカーソルを配置します。 2. [View] → [Zoom] → [To Cursors] またはツールバーの [To Cursors] ボタン 	なし


## マウス ホイールの使用

機能	コマンド
表示の拡大/縮小	Ctrl キーを押しながらマウス ホイールを動かします。
左側/右側のスクロール	Shift キーを押しながらマウス ホイールを動かします。
上方向/下方向にスクロール	マウス ホイールを動かします。

## フロート ルーラの表示

フロート ルーラでは、波形ウィンドウ上部の標準ルーラ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 ボタンおよびフロート ルーラが表示されます。

### フロート ルーラを表示するには

- 次のいずれかを実行します。
    - セカンダリ カーソルを配置します。
    - または
    - マーカーを選択します。
  - [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必要があります。手順 1 に従った後は、セカンダリ カーソルが配置されるかマーカーが選択されるたびにフロート ルーラが表示されます。

フロート ルーラが表示されます。

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。

### マーカーを使用して波形値を表示するには

1. 「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] コマンドを使用するには

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。

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。

3. 手順 2 を必要に応じて繰り返します。

4. 前の遷移に戻るには、[View] → [Cursors] → [Previous Transition] をクリックするか、[Previous Transition] ボタン (⬅) をクリックします。

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。

5. 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。

### カーソルを使用して時間を計測するには

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。

**メモ**：[Snap to Transition] ボタンはデフォルトでオンで、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](#)が表示されている場合は、カーソルの時間値がルーラ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。

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。

「[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。

### マーカーを使用して時間を計測するには

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簡単に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。このコマンドに関連したツールバーが 2 つあり、シミュレーションの最初または最後にジャンプさせることができます。信号およびバスの値は、カーソルの位置に対応して更新されます。

### ユーザー指定の時間にジャンプするには

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. [Go To Time] ボックスに、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にジャンプするには

波形コンフィギュレーションを表示します。

- ・ 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。
- ・ 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を表示します。このコマンドを使用して値変更の原因を突き止め、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を表示します。

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)

### ドライバを表示するには

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が表示されます。ドライバがない場合は、その旨を伝えるメッセージが表示されます。

**メモ：** 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。

## 印刷プレビューを表示するには

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。次の「印刷するには」を参照してください。
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。別のプリンタを選択して印刷することもできます。

## 印刷するには

1. [File] → [Print] をクリックします。
2. [ISim 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。

**メモ：** [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を選択します。
5. [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の設定に従って印刷されます。

## カスタム カラーの使用

### カスタム カラー スキームの作成

波形コンフィギュレーションで使用される色は、カスタムの波形カラー スキームを作成して変更できます。

### カスタム カラー スキームを作成するには

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] を展開して [Colors] をクリックします。
3. [Colors] ページで [New] をクリックします。
4. カラー スキームの名前を入力します。
5. [Color] 列に表示されている色をクリックして、色を変更します。
6. [Apply] をクリックして、新しいカラー スキームを設定します。
7. [OK] をクリックします。

カスタムのカラー スキームが反映されます。

## 信号の表示色の変更

こ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。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。

## 信号の表示色を変更するには

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。

**メモ：** 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。

# シミュレーション結果の保存および表示

## 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) に自動的に保存されます。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。

### WDB ファイルへのデータベースの保存

ISim を ISE® から起動すると、[ISim Properties] ダイアログ ボックスで指定されている名前に従って WDB ファイル名が付けられます。コマンドラインから起動する場合は、**-wdb** オプションを使用して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。

**メモ：** データベース ファイルの名前は、シミュレーションで使用されると変更できません。

### WCFG ファイルへの波形コンフィギュレーションの保存

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、「[波形ウィンドウの概要](#)」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。

波形コンフィギュレーション ファイルを開き、スタティック シミュレータで結果を表示できます。詳細は、「[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。

### WDB と WCFG の関係

波形コンフィギュレーション (WCFG) ファイルを保存すると、このファイルのリファレンスが関連する波形データベース (WDB) ファイルに自動的に追加されます。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。



## ライブ シミュレーションを開く

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
- ・ 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

シミュレーション結果の保存方法の詳細は、「[シミュレーション結果の保存](#)」を参照してください。

## 波形データ ベースを開く

WBD ファイルは、シミュレーションの実行時に自動的に開きます。シミュレーションの実行に関する詳細は、「[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。

## 波形ウィンドウを開く

ISE® からシミュレーションを実行すると、ISim で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。

波形コンフィギュレーションを ISim で開くには、次の手順に従います。

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. ファイルを選択します。
4. [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には、次の手順に従います。

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` を実行します。

説明：

- ・ **-gui**：ISim グラフィカル ユーザー インターフェイスを起動します。
- ・ **-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 ISim グラフィカル ユーザー インターフェイスに開きます。

ISim 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。

新しい波形コンフィギュレーションの作成方法の詳細は、「[新しい波形コンフィギュレーションの作成](#)」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

読み取り専用のスタティック シミュレーションは、波形コンフィギュレーションに表示されるオブジェクトに関連した順番および設定を含む波形コンフィギュレーション ファイルと直前に実行したライブ シミュレーションのシミュレーション データを含む波形データベース ファイルから構成されています。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始する方法の詳細は、「[ライブ シミュレーションを開く](#)」を参照してください。

## 波形コンフィギュレーションおよび波形データベースを開く

### 関連する波形データベースを使用して既存の WCFG 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。

コマンド プロンプトで次のいずれかを実行します。

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

1. **isimgui.exe -open <wcfg\_file>.wcfg** を実行してスタティック シミュレータで WCFG ファイルを開きます。

説明：

- ・ **isimgui.exe**：アプリケーション実行ファイルです。
- ・ **-open <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。

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。このスタティック シミュレーションを開く方法は、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) を表示するときに有益です。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。

1. **isimgui.exe -open <wdb>.wdb -view <wcfg>.wcfg** を実行します。

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 プロンプトで次のいずれかを実行します。

1. **isimgui.exe -view <wdb\_file>.wdb** を実行します。

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。

コマンド プロンプトで次のいずれかを実行します。

1. スタティック シミュレータを開くには **isimgui.exe** を実行します。
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

1. **isimgui.exe -open <wdb\_file>.wdb** を実行します。

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。

スタティック ビューアでは、[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。

# デバッグ

## ソースコードのデバッグの概要

ISim では、HDL ソースコードをデバッグして、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。

次に、ISim でのデバッグで利用できる手法の一部を説明します。

- ・ 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、「[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。

- ・ 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、「[ブレークポイントを使用したデザインのデバッグ](#)」を参照してください。


## 1 行ずつの実行

### 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、「[ソースコードのデバッグの概要](#)」を参照してください。

## シミュレーションを 1 行ずつ実行するには

1. 次のいずれかを実行します。
  - ・ [Simulation] → [Step] をクリックします。
  - ・ [Step] ボタン (  ) をクリックします。
  - ・ [Console] タブで「step」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。

**メモ：** [Step] コマンドは現在のシミュレーション実行時間から開始されます。開始点 (0ns) からシミュレーションを 1 行ずつ実行する場合は、シミュレーションを再スタートします。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、「ISim で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。


## ブレイクポイント

### ブレイクポイントの設定


ISim では HDL ファイルの実行行にブレイクポイントを設定できます。ブレイクポイントを設定すると、「[ブレイクポイントを使用したデザインのデバッグ](#)」に記述されているように、ブレイ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。

**メモ：** ブレイクポイントを設定できるのは、実行コード行のみです。

### ブレイクポイントを設定するには

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。

**メモ：** 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。また、ブレイクポイントをクリックすると削除できます。

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




**メモ：** 実行行以外の行にブレイクポイントを配置しても、追加されません。

ブレイクポイントのリストは、[\[Breakpoints\] パネル](#)に表示されます。


## ブレイクポイントを使用したデザインのデバッグ

ブレイクポイントは、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。ブレイクポイントは、[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。ISim でのデバッグ方法の詳細は、「[ソースコードのデバッグの概要](#)」を参照してください。

## ブレークポイントを使用してデザインをデバッグするには

1. HDL ソース ファイルを開きます。詳細は、「[HDL ソース ファイルを開く](#)」を参照してください。
2. 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。詳細は、「[ブレークポイントの設定](#)」を参照してください。
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。
5. デバッグを開始する場合は、[Simulation] → [Restart] () をクリックします。
6. シミュレーションは、[Simulation] → [Run All] () または [Simulation] → [Run for Specified Time] () をクリックして実行します。  
シミュレーションは、最初のブレークポイントに到達するまで実行され、ブレークポイントで停止します。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。
7. 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ークポイントで予期どおりであることを確認します。
8. 結果が満たされるまで手順 6、7 を繰り返すか、またはデバッグ プロセスを停止します。

HDL ソース ファイルで設定したブレー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。詳細は、「[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。


## ブレークポイントの削除

ISim では、次の手順に従うと HDL ソース コードからブレークポイントを削除できます。

### ブレークポイントを削除するには

次の手順のいずれかを実行します。

- ・ 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。
- ・ Tcl プロンプトで次を実行します。
  1. 「**bp list**」と入力してデザインに含まれるブレークポイントすべてをリストし、各ブレークポイントのインデックス番号および行数を示します。
  2. 「**bp del**」または「**bp remove**」に続けてブレークポイントのインデックス番号を入力します。構文および例は、「[bp コマンド](#)」を参照してください。

**メモ：** ブレークポイントは [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。

信号のブレークポイントが削除されます。

### すべてのブレークポイントを削除するには

次の手順のいずれかを実行します。

- ・ [View] → [Breakpoint] → [Delete All Breakpoints] をクリックします。
- ・ [Delete All] 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。

HDL に含まれるすべてのブレークポイントが削除されます。

# 消費電力概算向けアクティビティ データの書き出し

## デザインのアクティビティ 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ます。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るスイッチング アクティビティ データを使用できます。電力解析および消費電力が最適化されたインプリメンテーションでの正確度を高めるため、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。データは、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概算するためにベクタを使用しない解析アルゴリズムが使用されます。

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 (マップと PAR)、消費解析に XPower Analyzer ヘルプを参照してください。

## アクティビティ ファイルの種類

2 つのステイミュラス ファイルを ISim シミュレーションに書き出すことができます。

- ・ **SAIF ファイル** : 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 ファイル** : Value Change Dump (VCD) ファイルは、ヘッダ情報、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。

## 消費電力のアクティビティ ファイルを書き出すには

スイッチング アクティビティ ファイルは、次の手順で書き出すことができます。

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 **-vcdfile** オプションを使用します。
2. **シミュレーションを実行します** (例 : 1000ns)。

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 を一度に実行できます。
3. シミュレーションが完了したら適切な 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。例 : **saif close** または **vcd dumpoff**
4. **isim** 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。



## Tcl シミュレーション コマンドの使用

### シミュレーション コマンドの概要

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。

### 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、次のように入力します。詳細は、「[シミュレーション コマンドの入力](#)」を参照してください。

- ・ **ISim 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 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、シミュレーション 実行 ファイルを `-tclbatch` オプションを使用して実行し、Tcl ファイルを参照させます。詳細は、「[ISim シミュレーション 実行 ファイル コマンドの概要と構文](#)」を参照してください。

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。これらの例を参照するには、[\[Edit\]](#) → [\[Language Templates\]](#) をクリックして言語 テンプレートを表示し、[\[Tcl\]](#) → [\[Tools\]](#) → [\[ISim\]](#) 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、「[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。

**メモ** : Tcl の使用方法の詳細は、[Tcl/Tk Documentation ページ](#)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

次に、使用可能なシミュレーション コマンドを表示します。各シミュレーション コマンドでは、構文、オプション、および使用例を参照できます。

**メモ** : これらのコマンドでは、大文字/小文字が区別されます。

- ・ **bp** : HDL ソース コードでブレイクポイントを設定/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、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 get 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。 **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。
- ・ **marker add** : 新しいマーカーを追加します。
- ・ **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ソフトウェアのいずれかを終了します。
- ・ **restart** : シミュレーションを再実行します。シミュレーション時間は 0 に戻ります。
- ・ **resume** : **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を継続して実行します。
- ・ **run** : シミュレーションを開始します。
- ・ **saif** : 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。
- ・ **scope** : デザイン階層を移動します。
- ・ **sdfanno** : 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。
- ・ **show** : デザインの選択した部分を [Sim Console] タブに表示します。
- ・ **step** : 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。
- ・ **test** : 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かどうかを調べます。
- ・ **vcd** : シミュレーション結果を VCD フォーマットで生成します。
- ・ **virtualbus add** : 新しい仮想バスを追加します。
- ・ **wcfg new** : 新しい波形コンフィギュレーションを作成します。
- ・ **wcfg open** : 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。
- ・ **wcfg save** : 波形コンフィギュレーションを保存します。
- ・ **wcfg select** : 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。
- ・ **wave log** : 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。
- ・ **wave add** : 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。

## シミュレーション コマンドの入力

シミュレーション コマンドの入力方法は、ISim の実行方法によって異なります。次に各モードでの入力方法を示します。

シミュレーション コマンドおよびコマンドの構文の詳細は、「[シミュレーション コマンドの概要](#)」を参照するか、「**help**」または「**help** <command\_name>」とプロンプトに入力します。

### GUI モードでコマンドを入力するには

ISim を起動すると、[Console] パネルから Tcl コマンドを実行できます。

1. [\[Console\] パネル](#)をクリックします。
2. 新しい行にカーソルを置きます。
3. 正しい構文を使用してシミュレーション コマンドを入力します。
4. Enter キーを押してコマンドを実行します。

コマンドが実行されます。コマンド ログが [Console] パネルに表示されます。

### 対話型コマンド ライン モードでコマンドを入力するには

解析して VHDL、Verilog、または混合言語デザインを生成した後に、次の手順に従ってシミュレーションを実行してデザインを解析できます。

1. コマンド ラインで fuse オプションを使用して生成した ISim シミュレーション実行ファイル (例： **my\_sim.exe**) を実行します。

次の Tcl プロンプトが表示されます。

This is a Full version of ISim.

Time resolution is 1 ps

ISim>

2. 正しい構文を使用してシミュレーション コマンドを入力します。
3. Enter キーを押してコマンドを実行します。

コマンドが実行されます。シミュレーション出力が stdout および isim.log ファイルの両方に自動的に含められます。

**メモ**： Ctrl + C キーを使用するとシミュレーションを停止できます。

次に典型的なシーケンス例を示します。

1. **scope** : デザイン階層の現在の位置を表示します。
2. **dump** : デザイン階層の現在の位置にある信号の値を表示します。
3. **show value clk** : clk 信号の値を表示します。
4. **run 1000 ns** : シミュレーションを 1000ns 実行します。
5. **restart** : シミュレーション時間を 0 にリセットします。
6. **wave log /** : 最上位の VHDL 信号、Verilog ワイヤ、および Verilog レジスタすべて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。
7. **run 1000 ns** : シミュレーションを 1000ns 実行します。
8. **wave log /tb/UUT/clk** : /tb/UUT/clk のシミュレーション出力を現在のシミュレーション時間 (1000 ns) から波形データベース (wdb) に出力します。
9. **run 1000 ns** : シミュレーションをさらに 1000ns 実行します。
10. **quit** : シミュレーションを終了します。

シミュレーションを終了すると、作業ディレクトリに **isim.wdb** というファイルが生成されます。この **isim.wdb** ファイルは波形データベース ファイルで、シミュレーション中に記録された信号のシミュレーション出力が含まれています。**isimgui -view isim.wdb** を実行すると、波形ビューアでこのファイルを開いて遷移を確認できます。

## 非対話型バッチ モードでコマンドを入力するには

バッチ モードでは 1 つの Tcl ファイルにすべての Tcl コマンドを含め、このファイルを参照してシミュレーション実行ファイルを実行する必要があります。

1. **isim.tcl** など、拡張子が .tcl のファイルを作成して、実行するシミュレーション コマンドをファイルに含めます。

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、Project Navigator に含まれている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。

- a. Project Navigator を起動します。
- b. [Edit] → [Language Templates] をクリックします。
- c. [Tcl] → [Tools] を展開して [ISim] フォルダを展開します。

**メモ** : Tcl ファイルを作成するとき、ファイルの実行が完了したときにシミュレーションが完全に終了するよう、必ずファイルの最後の行に **quit コマンド** を含めてください。

次に **isim.tcl** の内容例を示します。

```
wave add /  
run 1000 ns
```

2. **isim.tcl** のコマンドを実行するには、コマンドライン プロンプトに次を入力します。

```
stopwatchsim.exe -tclbatch isim.tcl
```

シミュレーション出力が stdout および **isim.log** ファイルの両方に自動的に含まれます。

## 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も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。

### [Sim Console] タブで変数を設定するには

[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"
```

上記の構文は次の部分から構成されています。

- ・ **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。

変数が自動的に設定されます。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、「eval \$svc」と入力します。

## ISim 波形ビューア Tcl コマンド

ISim 波形ビューア Tcl コマンドは、アクティブ 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。ウィンドウ タブをクリックするか、または **wcfg select** 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] 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 では、**wcfg new** および **wcfg open** コマンドを実行すると、同様にアクティブ ウィンドウを変更できます。

### ISim 波形ビューア コマンド グループ

ISim 波形ビューア Tcl コマンドは、次の 3 つのグループに分類されます。

- ・ **波形コンフィギュレーション入力/出力コマンド** : 波形コンフィギュレーションの作成、保存、選択などを実行します。
- ・ **波形コンフィギュレーション編集コマンド** : 波形コンフィギュレーションの編集に使用します。
- ・ **マーカー コマンド** : 波形に新しいマーカーを追加します。

## コマンド ラインの表記規則

メニュー コマンドを使用する代わりに、シミュレーション コンソール ウィンドウのプロンプトに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。

シミュレーション コマンドで使用する構文の表記規則は、次のとおりです。

構文	説明
( )	構文内でコマンドのオプションがかっこ ( ) で囲まれている場合、オプションを入力する必要があります。
[ ]	オプションが角かっこ [ ] で囲まれている場合、オプションは入力しなくてもかまいません。  <b>メモ</b> : Tcl ではネストしたコマンドに [ ] を使用できません。[ ] 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、set var [show time] では var が現在の時間に設定されます。
	オプションが縦棒   で区切られている場合、表示されているいずれかのオプションを選択する必要があります。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。
...	オプションに ... が含まれている場合、1 つまた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。
{ }	中かっこ { } で囲まれている場合、かっこ内にスペースが複数含まれていても 1 つの引数として扱われます。これにより、スペースなどの特殊文字を含んだ引数がコマンドに渡されます。例 : set var {I have spaces} では、var を「I have spaces」に設定します。
< >	山かっこ < > は変数を示します。この値は必ず指定する必要があります。

**メモ** : イタリック文字は、変数を示すときに使用されます。

## Tcl コマンド

### エンジン コマンド

#### bp コマンド

**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。

**メモ** : このコマンドでは、大文字/小文字が区別されます。

#### 構文

**bp** (*options*)

## オプション

<b>add</b> <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。
<b>clear</b>	ISim に読み込んであるすべての 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。
<b>del</b> <index> [<index>... ]	<p>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、<b>bp list</b> 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、<b>list</b> オプションを参照してください。</p> <p>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。</p> <p><b>メモ：</b> 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。</p>
<b>list</b>	<p>デザインのブレークポイントをすべてリストし、<b>bp del</b> 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。</p> <p><b>bp list</b> コマンドを実行すると、次の情報が示されます。</p> <p>index directory_path/file_name::line_number</p> <p>説明：</p> <ul style="list-style-type: none"> <li>index: bp del コマンドで使用するインデックス番号</li> <li>directory_path: ソース ファイルへの完全パス</li> <li>file_name: ブレークポイントを含むソース ファイル名</li> <li>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</li> </ul>
<b>remove</b> <file_name> <line_number>	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。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。

## 例

**bp** コマンドは、次のように使用します。

## ブレイクポイントの設定

statmach.vhd というファイルの 2 行目にブレイ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
```

## 全ブレイクポイントの表示

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

```
bp list
```

## ブレイクポイントの削除

シミュレーションのブレイ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
```

ブレイ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。

- 最初に、**bp list** コマンドを使用してブレイ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
```

- stopwatch\_tb.vhd ファイルの 46 行目にあるブレイ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 行目のブレイ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 コマンド

**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。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

### 構文

```
describe <object_name>
```

### オプション

<object_name>	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。
---------------	--

### 例

**describe** コマンドは、次のように使用します。

```
ISim> describe param
```



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

## dump コマンド

**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。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

### 構文

**dump**

### 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 -p
```

## help コマンド

**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 Tcl コマンドとその構文がリストされます。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

### 構文

**help** [*command\_name* ]

*command\_name* はオプションです。

### オプション

<i>command_name</i>	指定したコマンドの説明を表示します。シミュレーション コマンドの一覧は、「 <a href="#">シミュレーション コマンドの概要</a> 」を参照してください。
---------------------	--

### 例

**help** コマンドは、次のように使用します。

### すべてのコマンドのヘルプを表示

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
```

### 1 つのコマンドのヘルプの表示

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 コマンド

**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。

**メモ** : このコマンドでは、大文字/小文字が区別されます。

### 構文

```
isim condition (add|remove|list) [ <condition expression>
<command>] [<radix_type>] [<label_name>] [<index_name>]
```

### オプション

(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>&lt;condition expression&gt; : <b>add</b> ファンクションに関連しており、指定した &lt;command&gt; をいつ実行するかが定義されます。この条件表現に使用された操作には、!= (非等号)、== (等号)、&amp;&amp; (AND) および    (OR) が含まれます。表現および演算子の間には、「<b>clk == St1</b>」のようにスペースを入力する必要があります。</p> <p>&lt;command&gt; : 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>
-radix <radix_type>	<p>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。</p> <p>サポートされる基数タイプは、<b>default</b>、<b>dec</b>、<b>bin</b>、<b>oct</b>、<b>hex</b>、<b>unsigned</b>、および <b>ascii</b> です。基数タイプが指定されていない場合は、<b>isim set radix</b> 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、<b>default</b> が基数として使用されます。</p>
-label <label_name>	条件名を指定するオプションの引数で、 <b>sim condition add</b> 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、 <b>isim condition remove</b> コマンドでのみ使用できます。

<b>-all</b>	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。
-------------	--

## 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 コマンド

**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。ただし、Verilog レジスタでは、このコマンドが削除された後でも 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。

**メモ** : このコマンドでは、大文字/小文字が区別されます。

### 構文

```
isim force (add|remove) < object_name > < value > [options]
```

### オプション

(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。
-radix <radix_type>	基数を指定します。サポートされる基数タイプは、 <b>default</b> 、 <b>dec</b> 、 <b>bin</b> 、 <b>oct</b> 、 <b>hex</b> 、 <b>unsigned</b> 、および <b>ascii</b> です。基数タイプが指定されていない場合は、 <b>isim set radix</b> 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、 <b>default</b> が基数として使用されます。
-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。

### 例

**isim force** コマンドは、次のように使用します。

#### 値を割り当てるには

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
```

現在のシミュレーション時間で data\_in 信号の値を強制的に 1 にし、その 50ns 後に 0 にした後、現在のシミュレーション時間から 75ns 後に 1 に戻すパターンを 100ns ごとに 5000ns 間繰り返すには、次を入力します。

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

値を削除するには

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 コマンド

**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。

**メモ:** このコマンドでは、大文字/小文字が区別されます。

#### 構文

```
isim get arraydisplaylength
```

オプションはありません。

#### 例

次を入力します。

```
isim get arraydisplaylength
```

次が戻されます。

64

### isim get radix コマンド

**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。

**メモ:** このコマンドでは、大文字/小文字が区別されます。

#### 構文

```
isim get radix
```

オプションはありません。

#### 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 コマンド

**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。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

#### 構文

```
isim get userunit
```

オプションはありません。

#### 例

次を入力します。

```
isim get userunit
```

次が戻されます。

```
1 ps
```

### isim ltrace コマンド

**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。

**メモ**：**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。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

#### 構文

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

#### オプション

[on   off]	行トレースのオン、オフを切り替えます。[Console] ウィンドウに現在実行されている行の情報が表示されます。デフォルトはオフです。
------------	---

#### 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":26
1005 ns "C:/Data/ISE_Projects/freqm/watchver/stopwatch_tb.v":27
1005 ns(3) "C:/Data/ISE_Projects/freqm/watchver/statmach.v":63
1005 ns(3) "C:/Data/ISE_Projects/freqm/watchver/statmach.v":64
```

## isim ptrace コマンド

**isim ptrace** コマンドを [Sim Console] タブに入力すると、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 [Sim Console] タブに表示されます。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非常に便利で、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。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

### 構文

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

### オプション

[on   off]	プロセストレースのオン/オフを切り替えます。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 [Sim Console] タブに表示されます。デフォルトはオフです。
------------	--

### 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 コマンド

**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。この制限数は次の値に影響します。

- ・ グラフィカル ユーザー インターフェイスの [Objects] パネルの値
- ・ **show value** コマンドに対する応答

デフォルト値は 64 です。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

### 構文

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

### オプション

<size>	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。  デフォルトは 64 です。
--------	--

### 例

次を入力します。

```
isim set arraydisplaylength 2
```

```
show value xcountout
```

次が戻されます。

```
00
00
```

また、ISim グラフィカル ユーザー インターフェイス (GUI) の [Objects] パネルに含まれているアレイの [Value] 値も確認してください。

次を入力します。

```
isim set arraydisplaylength 64
```

```
show value xcountout
```

次が戻されます。

```
0001000000
0001000000
```

また、ISim GUI の [Objects] パネルに含まれているアレイの [Value] 値も確認してください。

## isim set radix コマンド

**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。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

### 構文

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

### オプション

<b>&lt;radix_type&gt;</b>	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。この基数タイプは、 <a href="#">show value</a> 、 <a href="#">put</a> 、 <a href="#">test</a> 、 <a href="#">dump</a> 、 <a href="#">isim force</a> 、および <a href="#">isim condition</a> 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、 <b>default</b> 、 <b>dec</b> 、 <b>bin</b> 、 <b>oct</b> 、 <b>hex</b> 、 <b>unsigned</b> 、および <b>ascii</b> です。
---------------------------	---

### 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。

10 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 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 コマンド

**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

**メモ**：このコマンドでは、大文字/小文字が区別されます。

#### 構文

```
isim set userunit (options)
```

#### オプション

<[1 10 100]fs ps ns us ms s>	<i>userunit</i> 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。
------------------------------	--

#### 例

シミュレータのタイムスケールを 1ps に設定します。

```
isim set userunit 1 us
```

### onerror コマンド

**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。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。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

#### 構文

```
onerror( options )
```

#### オプション

{ 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 }	シミュレーション 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。これらのコマンドでは、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します。

## 例

この例では、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 ISim が終了します。

```
onerror { showtime;dump;quit -f }
```

この例では、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 ISim で 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 }
```

この例では、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 コマンド

**put** コマンドでは、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 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。このため、このコマンドは一時的です。

**メモ** : このコマンドでは、大文字/小文字が区別されます。

## 構文

```
put <signal_name/vhdl_process_name/process_variable_name >  
[element reference, element reference, ...] <value> | <object>  
<value> [-radix <radix_type>]
```

## オプション

<pre>&lt;signal_name/vhdl_process_name/ process_variable_name&gt; [element reference, element reference, ... ] &lt;value&gt;</pre>	<p><i>&lt;signal name&gt;</i>: 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。</p> <p><i>&lt;vhdl_process_name/process_variable_name&gt;</i>: 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。</p> <p><i>[element reference]</i>: 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。詳細は、次に示す例を参照してください。</p> <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"> <li>integer 型には、正または負の整数を指定できます。</li> <li>bit_vector 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。</li> <li>VHDL の場合、std_logic 型には U、X、0、1などを指定できます。</li> <li>Verilog の場合、bit_values 型には U、X、0、1を指定できます。</li> <li>strength 値はサポートされていません。</li> </ul>
<pre>&lt;object&gt; &lt;value&gt; [-radix &lt;radix_type&gt; ]</pre>	<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><i>&lt;object&gt;</i>: 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><i>&lt;value&gt;</i>: オブジェクトに追加する値を指定します。</p> <p>サポートされる基数タイプは、<b>default</b>、<b>dec</b>、<b>bin</b>、<b>oct</b>、<b>hex</b>、<b>unsigned</b>、および <b>ascii</b> です。基数タイプが指定されていない場合は、<b>isim set radix</b> 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、<b>default</b> が基数として使用されます。</p>

## 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

この後の例は、次のように宣言されている標準ロジック ベクタの配列を使用しています。

```
signal sigarray : (0 to 3) vectorarray(0 to 5, 1 to 4, 2 to 6);
```

sigarray ベクタ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
```

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 コマンド

**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ソフトウェア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 ISim が終了します。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

### 構文

```
quit [options]
```

## オプション

<b>-f</b>	現在のシミュレーションを停止して、ISim ソフトウェアを終了します。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。
<b>-s</b>	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] から 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。

## 例

ISim を終了し、Tcl プロンプトを開いたままにするには、次のように入力します。

```
quit
```

ISim を終了し、波形を保存するには、次のように入力します。

```
quit -f
```

現在の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。

```
quit -s
```

```
Quit the current simulation?
```

```
Yes
```

## restart コマンド

**restart** コマンドでは、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、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** では、次のコマンド設定がリセットされます。

- ・ [onerror](#) : onerror スクリプトが削除されます。
- ・ [scope](#) : 階層が /top にリセットされます。
- ・ [saif](#) : ファイルが閉じます。
- ・ [vcd](#) : ファイルが閉じます。
- ・ [isim force](#) : 適用されている isim force コマンドが削除されます。
- ・ [put](#) : 初期値に戻ります。

**メモ** : このコマンドでは、大文字/小文字が区別されます。

## 構文

```
restart
```

## 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 コマンド

**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。

**メモ**：このコマンドのみを入力しても効力はありません。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

### 構文

**resume**

オプションはありません。

### 例

この例では、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 ISim で 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 コマンド

**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。[\[Simulation\] → \[Run All\]](#) および [\[Simulation\] → \[Run\]](#) をクリックしても同じ操作を実行できます。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

### 構文

**run** [*options*]

### オプション

<b>all</b>	イベントがすべて終了するかブレイ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。ブレイクポイントの設定および削除に関する詳細は、「 <a href="#">bp コマンド</a> 」を参照してください。
<b>continue</b>	ブレイ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、 <b>run all</b> を実行するのと同じです。
<b>&lt;time&gt; &lt;unit&gt;</b>	<i>time</i> 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。  <i>unit</i> は、時間の単位を指定します。使用可能な値は、 <b>fs</b> 、 <b>ps</b> 、 <b>ns</b> 、 <b>us</b> 、 <b>ms</b> 、 <b>sec</b> のいずれかです。デフォルトは <b>ps</b> です。

### 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イクポイントに達するまで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 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 間実行するには、次のように入力します。

```
run
```

## saif コマンド

**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。「[デザインのアクティビティ データの書き出し](#)」も参照してください。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

### 構文

```
saif <options>
```

### オプション

<b>open</b> [-scope <path_name> ] [-file <file_name> ] [-allnets]	<p><b>open</b>：消費電力概算用の SAIF ファイルを生成します。</p> <p><b>-scope</b> &lt;path_name&gt;：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。</p> <p><b>-file</b> &lt;file_name&gt;：新しいSAIF ファイルを生成します。デフォルト名は <b>xpower.saif</b> 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。</p> <p><b>-allnets</b>：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。</p>
<b>close</b>	監視を停止して SAIF ファイルを出力します。
<b>-level</b> <number_of_levels>	<p><b>-level 0</b> では、指定した階層下すべてのレベルの信号遷移が監視されます。<b>-level 1</b>：指定した階層の信号遷移のみが監視されます。<b>-level 2</b>：階層の 2 つのレベルの信号が監視されます。</p>

### 例

**saif** コマンドは次のように使用します。

この例では、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
```

この例では、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
```

この例では 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 コマンド

**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。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

### 構文

```
scope [options]
```

### オプション

..	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>	path_name は、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。

### 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。

例：

```
X_IPAD \CLK/PAD (
.PAD(CLK)
);
```

\CLK/PAD は拡張された識別子です。

次を入力します。

```
scope /testbench/UUT/\CLK/PAD\
```

\CLK の前と PAD の後にバックスラッシュ (\) を追加する必要があることに注意してください。

## sdfanno コマンド

**sdfanno** コマンドでは、SDF 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。

メモ：このコマンドでは、大文字/小文字が区別されます。

## 構文

**sdfanno** (**-min** | **-typ** | **-max**) <file\_name> [options]

**-min** | **-typ** | **-max** および *file\_name* は必ず入力する必要がありますが、*options* はオプションです。

## オプション

( <b>-min</b>   <b>-typ</b>   <b>-max</b> )	<p><b>-min</b>、<b>-typ</b>、<b>-max</b> 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。</p> <ul style="list-style-type: none"> <li>• <b>-min</b> : VHDL/Verilog ファイルを最小遅延値でアノテートします。ホールド タイムのタイミング シミュレーションを実行できます。</li> <li>• <b>-typ</b> : VHDL/Verilog ファイルを標準遅延値でアノテートします。</li> <li>• <b>-max</b> : VHDL/Verilog ファイルを最大遅延値でアノテートします。セットアップ タイムのタイミング シミュレーションを実行できます。</li> </ul>
<file_name>	遅延情報を含む SDF ファイルの名前を指定します。 <b>sdfanno</b> コマンドでは、ファイル名を指定する必要があります。
<b>-nowarn</b>	警告メッセージを非表示にします。
<b>-noerror</b>	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。
<b>-root</b> <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、 <i>root_path</i> 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。

## 例

### sdfanno -typ コマンドの例

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 コマンドの例

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 コマンドの例

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 コマンド

**show** コマンドでは、デザインの選択した部分を表示します。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

## 構文

```
show (options)
```

## オプション

<b>child</b>   <b>child -r</b>	<b>child</b> では現在のブロックのすべての子ブロック (1 レベルのみ) が表示され、 <b>child -r</b> 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。
<b>constant</b>	現在のブロックに含まれているすべての定数、ジェネリック、およびパラメータをリストします。
<b>driver</b>	<i>signal_name</i> 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、そのドライバを記述する HDL コードの行番号も表示します。
<b>load</b>	<i>signal_name</i> で指定された信号のロードをすべて表示します。
<b>port</b>	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。
<b>scope</b>	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。 <b>scope コマンド</b> を path_name オプションなしで実行した場合と同じです。
<b>signal</b>	現在のモジュール内にある信号を、ポート信号も含めすべて表示します。
<b>time</b>	シミュレータの現在の時間を表示します。
<b>value</b> <generic_name>   <parameter_name>   <process_name>/<process_variable_name>   <signal_name> [element reference, element reference, ...]   <object> [-radix <radix_type>]	<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。  <signal_name> 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

	<p>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。</p> <ul style="list-style-type: none"> <li>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。</li> </ul> <pre>show value recsig.c</pre> <ul style="list-style-type: none"> <li>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。</li> </ul> <pre>show value mymod/mysig</pre> <p>[<i>element reference</i>]: 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。詳細は、次に示す例を参照してください。</p> <ul style="list-style-type: none"> <li>value 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、<i>signal_name</i> で指定されているベクタの値が表示されます。例: <b>show value(3:0)</b></li> <li>value 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、<i>signal_name</i> 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。例: <b>show value(2,3)</b></li> </ul> <p>&lt;<i>object</i>&gt; [-<b>radix</b> &lt;<i>radix type</i>&gt;] には、特定の基数と共に値が表示されます。&lt;<i>object</i>&gt; には、HDL オブジェクト データ タイプを設定します。サポートされる基数タイプは、<b>default</b>、<b>dec</b>、<b>bin</b>、<b>oct</b>、<b>hex</b>、<b>unsigned</b>、および <b>ascii</b> です。基数タイプが指定されていない場合は、<b>isim set radix</b> 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、<b>default</b> が基数として使用されます。</p>
<b>variable</b>	<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、<b>scope</b> 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 <b>show variable</b> を実行します。</p>

## 例

### 階層に含まれる子の表示

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、次の情報が表示されます。

Block Name: <fifo\_controller>

**show child -r** を入力すると、現在の階層および再帰的階層の情報が表示されます。

### ドライバの表示

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 という信号に関して次の情報が表示されます。

```
<Driver for fifocount>
  fifoctlr_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。

### ロードの表示

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 という信号に関して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/)
```

### 範囲の表示

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/
```

### 信号値の表示

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。

### オブジェクト値の表示

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 ベクタを使用しています。

```
signal sigx : std_logic_vector(0 to 5);
```

sigx 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 のビット 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 のすべてのビット値を表示するには、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 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 コマンド

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。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。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

### 構文

#### step

オプションはありません。

#### 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 コマンド

**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、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 エLEMENT の 1 ビットまたは全体値に実行できます。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

### 構文

```
test <signal_name/vhdl_process_name/process_variable_name>
{element reference, element reference, ...} <value> | <object>
<value> -radix <radix_type>
```

### オプション

<pre>&lt;signal_name / vhdl_process_name/process_variable_name&gt; {element reference, element reference, ...} &lt;value&gt;</pre>	<p>&lt;signal_name&gt;: 比較する信号またはバスの名前を入力します。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。</p> <p>&lt;vhdl_process_name/process_variable_name&gt;: 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。</p> <p>{element reference}: 参照する信号名のサブELEMENTを指定します。信号の個々のサブELEMENTを参照することで信号を詳細に指定できます。詳細は、次に示す例を参照してください。</p> <p>&lt;value&gt;: 信号またはバスの実際の値と比較する値を入力します。</p>
--	--

<pre>&lt;object&gt; &lt;value&gt; -radix &lt;radix_type&gt;</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> <p>&lt;object&gt; には、テストする信号、バス、またはオブジェクトを指定します。</p> <p>&lt;value&gt; には、オブジェクトに追加する値を指定します。</p> <p>サポートされる基数タイプは、<b>default</b>、<b>dec</b>、<b>bin</b>、<b>oct</b>、<b>hex</b>、<b>unsigned</b>、および <b>ascii</b> です。基数タイプが指定されていない場合は、<b>isim set radix</b> 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、<b>default</b> が基数として使用されます。</p>
---	---

### 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'
```

1 が戻されます。

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'
```

0 が戻されます。

`/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.
```

For the bus `Lsbcnt` in `Uut`, the command **test Lsbcnt 1001** displays the following message:

```
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 ベクタを使用しています。

```
signal sigx : std_logic_vector(0 to 5);
```



sigx のビット 0 の値を 1 であるかを調べるには、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 の値が 101010 であるかを調べるには、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 コマンド

**vcd** コマンドでは、シミュレーション結果を VCD フォーマットで生成します。このコマンドを使用すると、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、その他の関数を実行できます。「[デザインのアクティビティ データの書き出し](#)」も参照してください。

**メモ** : このコマンドでは、大文字/小文字が区別されます。

## 構文

**vcd** (*options*)

## オプション

<b>dumpfile</b> <file_name>	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。
<b>dumpvars</b> -m <module_name> [-1 <level>]	指定の変数およびその値を VCD ファイルに書き出します。  -m <module_name> : モジュール名を出力します。  -1 <level>  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適用され、個々の変数には適用されません。  1 : -m 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。
<b>dumppoff</b>	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>\$dumppoff</code> 関数を呼び出します。
<b>dumpon</b>	<b>dumppoff</b> オプションで中断した書き出しプロセスを再開します。 <b>dumpon</b> を呼び出した時に選択されているすべての値が書き出されます。Verilog の <code>\$dumpon</code> 関数を呼び出します。
<b>dumpall</b>	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。
<b>dumplimit</b> <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せん。Verilog の \$dumplimit 関数を呼び出します。
<b>dumpflush</b>	OS の VCD ファイル バッファを空にし、バッファ内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。

### 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
```

## 波形コンフィギュレーション入力/出力コマンド

### wcfg new コマンド

**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。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

### 構文

```
wcfg new
```

### 例

新しい波形コンフィギュレーションを作成するには、次のように入力します。

```
wcfg new
```

### wcfg open コマンド

**wcfg open** コマンドでは、波形コンフィギュレーションを開きます。

開いた波形コンフィギュレーションは、アクティブ ウィンドウで表示されます。ファイルを開くことができない場合は、エラーがレポートされます。次に、このコマンドの構文を示します。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

**構文**

```
wcfg open <filename>
```

**オプション**

<filename>	開く波形コンフィギュレーション ファイル名を指定します。
------------	------------------------------

**例**

toplevel.wcfg という名前の WCFG ファイルを開くには、次を入力します。

```
wcfg open topLevel.wcfg
```

**wcfg save コマンド**

**wcfg save** コマンドでは、波形コンフィギュレーションをファイルに保存します。

次に、このコマンドの構文を示します。

**メモ** : このコマンドでは、大文字/小文字が区別されます。

**構文**

```
wcfg save <filename>
```

**オプション**

<filename>	アクティブ波形コンフィギュレーションを保存するファイル名を指定します。このコマンドで使われる名前が波形コンフィギュレーションでも使用されます。ファイルが保存できない場合には、エラーがレポートされます。
------------	--

**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 コマンド**

**wcfg select** コマンドでは、アクティブ 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。

次に、このコマンドの構文を示します。

**メモ** : このコマンドでは、大文字/小文字が区別されます。

**構文**

```
wcfg select <wave_config_name>
```

**オプション**

<wave_config_name>	アクティブ ウィンドウに表示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。
--------------------	---

## 例

「design」という名前の波形コンフィギュレーションをウィンドウに表示するには、次を入力します。

```
wcfg select design
```

## 波形コンフィギュレーション編集コマンド

## wave add コマンド

**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 で [\[Add to Wave Window\]](#) をクリックしても同じ操作を実行できます。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

## 構文

```
wave add [-into <ID>][-wcfg <wave_config_name>][-reverse][-radix
<radix>][-color <color>][-name <custom_name>]
[-r]{<object_name>}
```

## オプション

<i>&lt;object_name&gt;</i>	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。
<b>-into</b> <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。
<b>-wcfg</b> <wave_config_name>	オブジェクトを追加する波形コンフィギュレーション名を指定します。指定する名前のコンフィギュレーションが見つからない場合は、新しいファイルが作成されます。  このオプションは、廃止される予定です。このオプションを使用しないでください。代わりに <a href="#">wcfg new</a> または <a href="#">wcfg select</a> を使用してシミュレーション オブジェクトを任意の波形コンフィギュレーションに追加し、 <a href="#">wcfg save</a> <filename> でその波形コンフィギュレーションを特定のファイルに保存してください。
<b>-reverse</b>	バスの順序を反転します。
<b>-radix</b> <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。

<b>-color</b> <color>	シミュレーション オブジェクトの色を設定します。<color> の値は、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。次の色をテキスト入力できます：black (黒)、red (赤)、darkred (濃紅)、green (緑)、darkgreen (深緑)、blue (青)、darkblue (群青)、cyan (シアン)、darkcyan (ダークシアン)、magenta (マゼンタ)、darkmagenta (ダーク マゼンタ)、darkyellow (濃黄)、gray (グレー)、darkgray (ダーク グレー)、lightgray (ライト グレー)。これらの色の RGB 値は、RGB 表で定義されています。
<b>-name</b> <custom_name>	波形オブジェクトにカスタム名を付けます。
<b>-r</b>	このオプションは、特定のブロック名に適用されます。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。

### 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 コマンド

**divider add** コマンドでは、仕切りを追加します。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

### 構文

次に、このコマンドの構文を示します。

```
divider add [-into <ID> ][-color <color> ]
```

## オプション

<b>-into</b> <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。
<b>-color</b> <color>	仕切りの色を設定します。<color> の値は、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。次の色をテキスト入力できます：black (黒)、red (赤)、darkred (濃紅)、green (緑)、darkgreen (深緑)、blue (青)、darkblue (群青)、cyan (シアン)、darkcyan (ダークシアン)、magenta (マゼンタ)、darkmagenta (ダーク マゼンタ)、darkyellow (濃黄)、gray (グレー)、darkgray (ダーク グレー)、lightgray (ライト グレー)。これらの色の RGB 値は、RGB 表で定義されています。

## 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 コマンド

**group add** コマンドでは、グループを追加します。

メモ：このコマンドでは、大文字/小文字が区別されます。

## 構文

次に、このコマンドの構文を示します。

```
group add [-into <ID>]
```

## オプション

<b>-into</b> <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。
-------------------	--------------------------------------

### 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、次を入力します。

```
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 コマンド

**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。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

### 構文

次に、このコマンドの構文を示します。

```
virtualbus add <name> [-into <ID>] [-reverse] [-radix  
<radix>] [-color <color>]
```

### オプション

<b>&lt;name&gt;</b>	仮想バスの名前を指定します。
<b>-into &lt;ID&gt;</b>	新しく作成された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。
<b>-reverse</b>	バスの順序を逆にします。
<b>-radix &lt;radix&gt;</b>	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。
<b>-color &lt;color&gt;</b>	仮想バスの色を設定します。<color> の値は、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。次の色をテキスト入力できます：black (黒)、red (赤)、darkred (濃紅)、green (緑)、darkgreen (深緑)、blue (青)、darkblue (群青)、



cyan (シアン)、darkcyan (ダークシアン)、magenta (マゼンタ)、darkmagenta (ダーク マゼンタ)、darkyellow (濃黄)、gray (グレー)、darkgray (ダーク グレー)、lightgray (ライト グレー)。これらの色の RGB 値は、RGB 表で定義されています。
--

**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
```

**マーカー コマンド****marker add コマンド**

**marker add** コマンドでは、マーカーを追加します。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

**構文**

次に、このコマンドの構文を示します。

```
marker add <time>
```

**オプション**

<i>&lt;time&gt;</i>	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」コマンドで返されるデフォルトのユーザー時間単位が使用されます。
---------------------	---

**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
```

**波形ビューア リソース コマンド****wave log コマンド**

**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。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

## 構文

次に、このコマンドの構文を示します。

```
wave log [-r] {<object_name> }
```

## オプション

<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。
-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。

## 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 /
```

## リファレンス

### シミュレーション実行コマンド

#### シミュレーション実行コマンドの概要

**メモ：** 次の情報は、アドバンス ユーザーを対象としています。

ISE® ソフトウェアまたは ISim インターフェイスを使用する代わりにコマンド ラインから ISim を実行できます。

#### コマンド ラインからのシミュレーションの実行

Verilog デザインをコマンド ラインからシミュレーションする場合は、次を参照してください。

- ・ [コマンド ラインからの論理シミュレーションの実行 \(Verilog デザイン\)](#)
- ・ [コマンド ラインからのタイミング シミュレーションの実行 \(Verilog デザイン\)](#)

VHDL デザインをコマンド ラインからシミュレーションする場合は、次を参照してください。

- ・ [コマンド ラインからの論理シミュレーションの実行 \(VHDL デザイン\)](#)
- ・ [コマンド ラインからのタイミング シミュレーションの実行 \(VHDL デザイン\)](#)

#### 実行コマンドのサマリ

コマンド ラインからシミュレーションを実行するときに使用するコマンドが多数あります。

コンパイル コマンド

- ・ **VHDL コンパイラ (vhpcomp)：** VHDL コンパイラ vhpcomp では、VHDL ソース ファイルに含まれているデザイン ユニットがすべて解析されます。
- ・ **Verilog コンパイラ (vlogcomp)：** Verilog コンパイラ vlogcomp では、Verilog ソース ファイルに含まれているデザイン ユニットがすべて解析されます。

エラボレーション コマンド

**HDL リンカ (fuse)：** HDL リンカ fuse では解析ノードでデザインのスタティック エラボレーションの実行、各モジュール インスタンスのオブジェクト コードの生成、および生成したオブジェクト コードの ISim シミュレーション エンジン ライブラリへのリンク付けを実行して、シミュレーション実行ファイルを作成します。この生成したシミュレーション実行ファイルを実行すると、検討中のデザインにシミュレーションを実行できます。

## シミュレーション コマンド

**ISE シミュレーション実行ファイル**：fuse コマンドにより生成されます。ISim で、デザインのシミュレーションを実行するには、このファイルを実行します。ISim を ISE インターフェイスから実行した場合は、このファイルは実行されています。コマンドラインからデザインのシミュレーションを実行するには、このファイルを指定する必要があります。シミュレーション実行ファイルでは、イベントドリブン型のシミュレーションが実行でき、Tcl コマンドを使用したシミュレーションの実行およびプローブが多様にサポートされています。

## プロジェクト ファイル

プロジェクトファイルを使用すると、vhpcomp、vlogcomp、および fuse を実行できます。すべてのデザイン ファイルをプロジェクト ファイルにリストし、**-prj** オプションを使用してプロジェクト ファイルを指定します。プロジェクト ファイルには、VHDL ファイルまたは Verilog ファイルを含めることができます。

プロジェクト ファイルの各行には、次の構文を使用します。

```
<language> [<library>] <filename> { [-d <macro>] [-i  
<include_path> ] }
```

上記の構文は次の部分から構成されています。

<language>：vhdl または verilog

<library>：作業ライブラリ名 (オプション)

## Tcl コマンド

シミュレーション実行ファイルは、Tcl インターフェイスを使用してバッチ モードで実行できます。Tcl コマンドを使用すると、シミュレータを制御し、シミュレーションの結果を表示できます。[詳細情報](#)

一連の Tcl コマンドをバッチ ファイルに記述し、そのファイルを **-tclbatch** オプションで指定してシミュレーション実行ファイルを実行することも可能です。

## ISE シミュレーション実行ファイル

### ISim シミュレーション実行ファイル コマンドの概要と構文

**メモ**：次の情報は、アドバンス ユーザーを対象としています。

ISim の <executable\_name>.exe ファイルは、ユーザー定義の実行ファイルです。このようなファイルをコマンドラインで実行すると、ISim シミュレーションが起動します。実行ファイルの名前は、[fuse コマンドの例](#)に含まれている **-o** オプションで指定します。定義しない場合は、デフォルトの実行ファイル名 **x.exe** を使用します。

次に、このコマンドの構文を示します。

```
<executable_name> .exe (option)
```

<executable\_name>.exe には、ユーザー定義の実行ファイルまたはデフォルトの **x.exe** を指定します。

**メモ**：このコマンドでは、大文字/小文字が区別されます。

### ISim シミュレーション実行ファイル コマンドのオプション

次に、ISim シミュレーション実行ファイル コマンドのオプションを示します。

メモ：このコマンドでは、大文字/小文字が区別されます。

<b>-f</b> <cmd_file>	ISim エンジン オプションは、次回使用できるようにテキスト ファイルに記述して保存できます。 -f オプションを使用すると、cmd_file に保存されたオプションを使用してシミュレーションが実行されます。
<b>-gui</b>	ISim グラフィカル ユーザー インターフェイスを起動します。
<b>-h</b>	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。
<b>-intstyle</b> <b>ise</b>   <b>xflow</b>   <b>silent</b>	メッセージの表示方法を指定します。 <b>ise</b> に設定するとメッセージが ISE のログ ウィンドウに表示され、 <b>xflow</b> に設定すると XFLOW のメッセージが表示されます。 <b>silent</b> に設定すると、メッセージは表示されません。
<b>-log</b> <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。
<b>-maxdeltaid</b> <number>	デルタの最大値を指定します。number は、任意の整数です。
<b>-nolog</b>	ログ ファイルを生成しません。
<b>-sdfnowarn</b>	SDF の警告メッセージを表示しません。
<b>-sdfnoerror</b>	SDF ファイルで検出されるエラーを警告として扱います。
<b>-sdfmin</b>   <b>-sdftyp</b>   <b>-sdfmax</b> <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"> <li>・ <b>-sdfmin</b> : &lt;root&gt; で SDF により &lt;file&gt; が最小の遅延でアノテートされます。</li> <li>・ <b>-sdftyp</b> : &lt;root&gt; で SDF により &lt;file&gt; が通常の遅延でアノテートされます。</li> <li>・ <b>-sdfmax</b> : &lt;root&gt; で SDF により &lt;file&gt; が最大の遅延でアノテートされます。</li> </ul>
<b>-sdfroot</b> <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。
<b>-tclbatch</b> <file_name>	バッチ モードをオンにします。デフォルトでは、バッチ モードはオフです。バッチ モードがオフの場合、シミュレーションが実行されていても、[Console] パネルに Tcl コマンドを入力できません。バッチ モードがオンの場合、指定したバッチ ファイルのコマンドがすべて実行されるまで、コマンド プロンプトで新しく入力されたコマンドは無視されます。file_name では、実行する Tcl コマンドを含むファイル名を指定します。  ISim の Tcl コマンドの詳細は、「 <a href="#">シミュレーションコマンドの概要</a> 」を参照してください。

<b>-testplusarg</b> <string stringvalue>	<p>シミュレータが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 システム関数を一致させたときに、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。</p> <p><i>string</i> には任意の文字列を含めることができます。たとえば「<b>-testplusarg HELLO</b>」と入力できます。Verilog ファイルで (\$test\$plusargs("HE")) が使用される場合、関数で true が戻されます。</p> <p><i>stringvalue</i> 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。たとえば「<b>-testplusarg FINISH=10000</b>」と入力します。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。</p> <p>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>
<b>-vcdfile</b> <vcd_file>	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。
<b>-vcdunit</b> <unit>	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、 <b>fs</b> 、 <b>ps</b> 、 <b>ns</b> 、 <b>us</b> 、 <b>ms</b> 、 <b>sec</b> のいずれかです。デフォルトは <b>ps</b> です。
<b>-view</b> <waveform_file.wcfg>	<b>-gui</b> オプションと組み合わせて使用して、ISim 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。
<b>-wdb</b> <waveform_database_file.wdb>	シミュレーション データが特定のファイルに保存されます。例： <b>x.exe -wdb my.wdb</b> 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。

## ISim シミュレーション実行ファイル コマンドの例

ISim のシミュレーション実行ファイル コマンドは、次のように使用します。

watchvhdl というシミュレーション実行ファイルをバッチ モードで実行するには、次のように入力します。

**watchvhdl.exe -tclbatch batchfile**

watchvhdl という名前のシミュレーション実行ファイルを、adder.sdf ファイルの遅延情報を使用してバックアノテーションして実行するには、次のように入力します。

**watchvhdl.exe -sdftyp adder.sdf**

シミュレーション実行ファイルを対話式に実行するには、次のように入力します。

**watchvhd.exe**

ISim GUI で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。

**watchvhd.exe -gui**

次の Verilog コードを使用して、シミュレーションを 1000 クロック サイクル間実行してから終了する場合：

```
real frequency;
reg [8*32:1] testname;
integer stop_clock;
if ($value$plusargs("FINISH=%d", stop_clock))
begin
repeat (stop_clock) @(posedge clk);
$finish;
end
```

次を入力します。

**x.exe -testplusarg FINISH=10000**

## ISim ハードウェア協調シミュレーションの概要

**メモ：** ハードウェア協調シミュレーションは限定カスタム向け (LCA) の機能であり、特別なライセンスが必要です。この機能をイネーブルにする方法については、フィールド アプリケーション エンジニア (FAE) までお問い合わせください。

ハードウェア協調シミュレーションは ISim にソフトウェア 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サブ モジュールのシミュレーションをハードウェア (汎用ボードに搭載されているザイリンクス FPGA) 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。

- ・ **モデルの使用：** ISim のハードウェア協調シミュレーションでは、現段階でロジックベースのデザイン向けモデルと外部入力および出力を含む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。
- ・ **使用方法：** ソフトウェア 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。コンパイルは、コマンド ラインまたは Project Navigator から ISim コンパイラ [fuse](#) を使用して実行できます。詳細は、「[\[Hardware Co-Simulation Properties\] ダイアログ ボックス](#)」を参照してください。インプリメンテーション ツールは自動的に実行され、ハードウェア協調シミュレーション ビットストリームがコンパイルの最後で生成されます。このビットストリームは、ハードウェアで実行する部分と ISim で実行する部分を協調シミュレーションするのに使用します。

## fuse

### fuse コマンドの概要と構文

**メモ：** 次の情報は、アドバンス ユーザーを対象としています。

fuse コマンドは、ISim で使用される Hardware Description Language (HDL) コンパイラ、エラボレータ、およびリンカです。vhpcomp (VHDL コンパイラ) または vlogcomp (Verilog コンパイラ) でコンパイルされたデザイン ユニットをリンクし、シミュレーション実行ファイルを作成します。また、このコマンドでは混合言語のプロジェクト ファイルを使用してデザイン ユニットをコンパイルすることも可能です。このコマンドの引数では、最上位デザイン ユニット名を指定する必要があります。最上位デザイン ユニットでスタティック エラボレーションを実行し、これらのユニットをコードにコンパイルします。次にデザイン ユニットのオブジェクト コードが共にリンクされて、シミュレーション実行ファイルが作成されます。

- ・ 最上位のデザイン ユニット名を `{<library_name>.<top_name>}` として入力します。たとえばテスト ベンチ ファイルのデザイン ユニット名を入力します。デザインで Verilog UNISIM または Verilog SIMPRIM ライブラリが使用される場合は、1 つを必ず `g1b1` にする必要があります。ライブラリ名を含めるのはオプションです。ライブラリ名が指定されない場合は、デフォルトのライブラリ名 `work` が使用されます。
- ・ **-prj** オプションを使用すると、必要に応じて vhpcomp または vlogcomp が自動的に呼び出され、HDL コードがコンパイルされます。
- ・ **-o <sim\_exe>** オプションを使用すると、デフォルトの `x.exe` からシミュレーション実行ファイル名とそのディレクトリを変更できます。

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。

**メモ：** `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。

構文

**fuse** (*option* )

*option* には、「[fuse コマンドのオプション](#)」に示すどのオプションも入力できます。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

## fuse コマンドのオプション

次に、**fuse** コマンドのオプションを示します。

<b>-d</b> <code>&lt;macro_definition&gt; [ = &lt;value&gt; ]</code>	このオプションは、Verilog でのみ使用可能です。Verilog ファイルで使用するマクロおよび必要な値を指定します。 <b>-d</b> は、複数指定できます。  <b>メモ：</b> 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。
<b>-f</b> <code>&lt;cmd_file&gt;</code>	fuse オプションをテキストファイルに記述して保存すると、fuse を実行する際に毎回オプションを入力する必要がないので便利です。 <b>-f</b> オプションを使用すると、 <code>cmd_file</code> に保存されたオプションを使用して vlogcomp が実行されます。
<b>-generic_top</b> <code>"&lt;parameter&gt;=&lt;value&gt;"</code>	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を特定の値で上書きします。たとえば「 <b>-generic_top "P=10"</b> 」と入力した場合、生成前に最上位のパラメータ P に値 10 が適用されます。
<b>-h</b>	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。



<b>-i</b> <include_path>	このオプションは、Verilog でのみ使用可能です。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 <b>-i</b> を使用できます。 <b>-i</b> 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。
<b>-incremental</b>	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。
<b>-initfile</b> <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。
<b>-intstyle</b> <b>ise</b>   <b>xflow</b>   <b>silent</b>	メッセージの表示方法を指定します。 <b>ise</b> では、ISE の [Console] ウィンドウのメッセージを表示し、 <b>xflow</b> では XFLOW のメッセージを表示します。 <b>silent</b> に設定すると、メッセージは表示されません。デフォルトでは、すべてのメッセージが表示されます。
<b>-ise</b> <file>	ザイリンクス ISE ファイルを指定します。
<b>-L</b>   <b>-lib</b> <search_lib> [ = <lib_path> ]	<p>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。複数の <b>-L</b> オプションを使用可能で、リソースライブラリとして処理されます。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。</p> <p><i>search_lib</i> は指定のライブラリの論理名、<i>lib_path</i> は物理ライブラリへのパスを指定します。例：</p> <p><b>-L mylib=C:\home\mylib</b></p> <p>Verilog デザインでは、<b>-L</b> オプションが記述順にライブラリが検索されます。例：</p> <p><b>fuse -L unisim -L abcsim -L xyzsim mytop</b></p> <p>デザイン ユニットは、UNISIM、abcsim、xyzsim の順番に検索されます。デザイン ユニットが abcsim と xyzsim の両方で定義されている場合は、abcsim の定義が使用されます。</p> <p>この例で、<b>-lib</b> の順序を次のように入れ替えたとします。</p> <p><b>fuse -L unisim -L xyzsim -L abcsim mytop</b></p> <p>この場合、同じユニットが xyzsim と abcsim で定義されていると、xyzsim の定義が使用されます。</p>
<b>-maxdelay</b>	このオプションは、Verilog でのみ使用可能です。vlogcomp が呼び出された場合、最大遅延を使用します。

<b>-maxdesigndepth</b> <depth>	エラボレータ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、エラボレータでエラーが発生します。エラボレータでデザインに無限に反復されるインスタンス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。
<b>-mindelay</b>	このオプションは、Verilog でのみ使用可能です。vlogcomp が呼び出された場合、最小遅延を使用します。
<b>-mt</b> <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。 <b>on</b> 、 <b>off</b> 、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。
<b>-nodebug</b>	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、デバッグ用に HDL ユニットが生成されます。
<b>-nospecify</b>	このオプションは、Verilog でのみ使用可能です。ブロック指定機能をディスエーブルにします。
<b>-notimingchecks</b>	このオプションは、Verilog でのみ使用可能です。タイミング チェックをディスエーブルにします。
<b>-o</b> <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。 <i>sim_exe</i> 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。  <i>work_lib/mod_name/platform/x.exe</i>  説明： <ul style="list-style-type: none"> <li>・ <i>work_lib</i> : 作業ライブラリ</li> <li>・ <i>mod_name</i> : 最上位モジュール</li> <li>・ <i>platform</i> : Windows</li> </ul>
<b>-override_timeprecision</b>	<b>-timescale</b> 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すべての Verilog モジュールの時間精度を上書きします。
<b>-override_timeunit</b>	<b>-timescale</b> 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。
<b>-prj</b> <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® ソフトウェアにより使用される主要ソース ファイルです。  <i>prj_file</i> はプロジェクト ファイルで、拡張子は <b>.prj</b> である必要があります。

<b>-rangecheck</b>	<p>このオプションは、VHDL でのみ使用可能です。VHDL での割り当てに値範囲チェックを実行します。</p> <p><b>メモ：</b> 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 では配列内のインデックスが許容範囲にあるかが常にチェックされます。</p> <p>デフォルトではオフです。</p>
<b>-sourcelibdir</b>	<p>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。詳細は「<a href="#">ソース ライブラリのサポート</a>」を参照してください。</p>
<b>-sourcelibext</b>	<p>モジュールのソース ファイルの拡張子を指定します。<b>-sourcelibdir</b> 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。詳細は「<a href="#">ソース ライブラリのサポート</a>」を参照してください。</p>
<b>-sourcelibfile</b>	<p>ライブラリ モジュールのファイル名を指定します。詳細は「<a href="#">ソース ライブラリのサポート</a>」を参照してください。</p>
<b>-timeprecision_vhdl</b> <time_precision>	<p>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。</p> <p><i>time_precision</i> 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。</p> <p>デフォルトは 1ps です。</p>
<b>-timescale</b> <time_unit/time_precision>	<p>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。<i>time_unit</i> では遅延計測単位を指定し、<i>time_precision</i> では精度の単位を指定します。</p> <p><i>time_unit</i> および <i>time_precision</i> 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。</p> <p>デフォルトのタイムスケールは 1ns/1ps です。</p>
<b>-typdelay</b>	<p>このオプションは、Verilog でのみ使用可能です。vlogcomp が呼び出された場合、標準遅延を使用します。</p>
<b>-v</b> <value>	<p>メッセージの表示レベルを指定します。0、1、2 のいずれかを設定でき、デフォルトは 0 です。</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。</p> <ul style="list-style-type: none"> <li>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。</li> <li>デザイン エラボレータから詳細なメッセージを取得します。</li> </ul>

	<ul style="list-style-type: none"> <li>・ ISim 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。</li> <li>・ ISim 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。</li> <li>・ 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。</li> <li>・ 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。</li> </ul>
<b>-version</b>	コンパイラのバージョンを表示します。

## fuse コマンドの例

### コンパイル済みの HDL の使用

次に、コンパイル済みの HDL ファイルを使用して fuse コマンドを実行する例を示します。

VHDL で最上位コンフィギュレーションを使用する場合：

```
fuse work.yourtop
```

Verilog または混合言語デザインですべての最上位モジュールを使用する場合：

```
fuse work.top_level_module_name_1 work.top_level_module_name_2  
work.glbl -L simprims_ver -L logicalLib1 -o mysim.exe
```

### HDL ソースの使用

#### VHDL ソース ファイルを使用して fuse コマンドを実行した例

この例では、x.prj というプロジェクト ファイルにリストされている VHDL ソース ファイルから tb.exe という実行ファイルを生成します。プロジェクト ファイルの内容は、次のとおりです。

```
VHDL work x1.vhd  
VHDL work x2.vhd  
VHDL work x3.vhd  
VHDL work tb.vhd
```

```
fuse -prj x.prj work.tb_top -o tb.exe
```

シミュレーションを開始するには、次のコマンドを実行します。

```
tb.exe
```

#### Verilog ソース ファイルを使用して fuse コマンドを実行した例

この例では、x.prj というプロジェクト ファイルにリストされている Verilog ソース ファイルから tb.exe という実行ファイルを生成します。最上位デザイン ユニットは tb で、tb.v ファイルで定義されています。myproj.prj プロジェクト ファイルの内容は、次のとおりです。

```
Verilog work x1.v  
Verilog work x2.v  
Verilog work x3.v  
Verilog work tb.v
```

次のコマンドを使用して fuse を実行します。

```
fuse -prj myproj.prj work.tb work.glbl -o tb.exe
```

シミュレーションを開始するには、次のコマンドを実行します。

#### **tb.exe**

**メモ：** Verilog では、work ライブラリ以外のライブラリにコンパイルされているモジュールがデザインにインスタンス化される場合、これらのライブラリを **-L** コマンドライン オプションを使用して fuse に渡し、fuse でこれらのデザイン ユニットが検出されてシミュレーション実行ファイルにリンクされるようにする必要があります。

#### **VHDL/Verilog 混合 デザイン ソース ファイルを使用して fuse コマンドを実行した例**

この例では、**x.prj** というプロジェクト ファイルにリストされている Verilog および VHDL ソース ファイルから **tb.exe** という実行ファイルを生成します。このデザインには、**tb.vhd** で定義された **tb** という VHDL 最上位デザイン ユニットと、**glbl.v** で定義された **glbl** という Verilog 最上位デザイン ユニットがあります。**myproj.prj** プロジェクト ファイルの内容は、次のとおりです。

```
Verilog work x1.v
VHDL work x2.vhd
Verilog work x3.v
VHDL work x4.vhd
Verilog work glbl.v
VHDL work tb.vhd
```

次のコマンドを使用して fuse を実行します。

```
fuse work.tb work.glbl -prj x.prj -o tb.exe
```

シミュレーションを開始するには、次のコマンドを実行します。

#### **tb.exe**

**メモ：** 混合言語デザインでは、モジュールが言語の境界にある場合や Verilog モジュールが work ライブラリ以外のライブラリにコンパイルされている場合、これらのライブラリを **-L** コマンドライン オプションを使用して任意の検出順序で fuse に渡し、fuse でこれらのデザイン ユニットが検出されてシミュレーション実行ファイルにリンクされるようにする必要があります。詳細は、「[混合言語シミュレーションの概要](#)」を参照してください。

#### **コマンド ファイル オプションの使用**

**-f** オプションを使用する場合の構文は、次のとおりです。

```
fuse -f my_design.cmd
```

次に、Verilog のコマンド ファイルの例を示します。

```
-nodebug
-intstyle xflow
-incremental
top_level_module_name_1
top_level_module_name_n
-L logicalLib1
-L logicalLib2
```

### **vlogcomp**

#### **vlogcomp コマンドの概要と構文**

**メモ：** 次の情報は、アドバンス ユーザーを対象としています。

ISim では、Verilog コンパイラ vlogcomp を使用して Verilog ソース ファイルを解析し、これらのファイルに含まれているすべてのデザイン ユニットのオブジェクト コードを生成します。vlogcomp で生成されたオブジェクト コードは fuse によりリンクされ、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、エラーが発生します。

## 構文

**vlogcomp** (*option*)

*option* には、「[vlogcomp コマンドのオプション](#)」に示すどのオプションも入力できます。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

**メモ：**

## vlogcomp コマンドのオプション

次に、**vlogcomp** コマンドのオプションを示します。

<b>&lt;verilog_file&gt;</b>	コンパイルする Verilog ソース ファイルを指定します。
<b>-d &lt;macro_definition&gt; [=&lt;value&gt;]</b>	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。 <b>-d</b> は、複数指定できます。
<b>-f &lt;cmd_file&gt;</b>	vlogcomp オプションをテキスト ファイルに記述して保存すると、vlogcomp を実行する際に毎回オプションを入力する必要がないので便利です。 <b>-f</b> オプションを使用すると、 <i>cmd_file</i> に保存されたオプションを使用して vlogcomp が実行されます。
<b>-h</b>	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。
<b>-i &lt;include_path&gt;</b>	Verilog の 'include 文のパスを指定します。 <b>-i</b> は、複数指定できます。
<b>-incremental</b>	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。
<b>-initfile&lt;sim_init_file&gt;</b>	デフォルトの xilinxsim.ini ファイルの代わりに使用するユーザー定義の INIT ファイルへの物理パスを指定します。
<b>-intstyle ise  xflow  silent</b>	メッセージの表示方法を指定します。 <b>ise</b> に設定するとメッセージが ISE® の [Cosole] タブに表示され、 <b>xflow</b> に設定すると XFLOW のメッセージが表示されます。 <b>silent</b> に設定すると、メッセージは表示されません。デフォルトでは、すべてのメッセージが表示されます。
<b>-ise &lt;file&gt;</b>	ザイリンクス ISE ファイルを指定します。

<b>-L -lib</b> <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。複数の <b>-lib</b> オプションを指定でき、指定したライブラリはリソース ライブラリとして扱われます。このオプションで物理パスを指定すると、 <b>xilinxsim.ini</b> ファイルで指定されているマップが無視されます。  <i>search_lib</i> は指定のライブラリの論理名、 <i>lib_path</i> は物理ライブラリへのパスを指定します。例： <b>-lib mylib=C:/home/mylib</b>
<b>-maxdelay</b>	最大遅延を使用します。
<b>-mindelay</b>	最小遅延を使用します。
<b>-nodebug</b>	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。
<b>-nospecify</b>	Verilog のパス遅延およびタイミング チェックを無視します。
<b>-notimingchecks</b>	Verilog の specify ブロックに含まれる構文のタイミング チェックを無視します。
<b>-prj</b> <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 ソフトウェアで使用する主要ソース ファイルで、 <i>prj_file</i> にはプロジェクト ファイルのパスを入力し、プロジェクト ファイル名は拡張子 <b>.prj</b> と共に入力します。プロジェクト ファイルには、絶対パスまたは相対パスを指定できます。相対パスには、../ を含める必要があります。
<b>-sourcelibdir</b>	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。詳細は「 <a href="#">ソース ライブラリのサポート</a> 」を参照してください。
<b>-sourcelibext</b>	モジュールのソース ファイルの拡張子を指定します。 <b>-sourcelibdir</b> 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。詳細は「 <a href="#">ソース ライブラリのサポート</a> 」を参照してください。
<b>-sourcelibfile</b>	ライブラリ モジュールのファイル名を指定します。詳細は「 <a href="#">ソース ライブラリのサポート</a> 」を参照してください。
<b>-typdelay</b>	標準遅延を使用します。
<b>-v</b> <value>	メッセージの表示レベルを指定します。0、1、2 のいずれかを設定でき、デフォルトは 0 です。
<b>-version</b>	コンパイラのバージョンを表示します。

<pre>-work &lt;work_lib&gt; [= &lt;lib_path&gt;]</pre>	<p>作業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、<code>xilinxsim.ini</code> 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 <b>work</b> です。</p> <p><code>work_lib</code> は指定の作業ライブラリの論理名、<code>lib_path</code> は物理ライブラリへのパスを指定します。</p> <p>例： <code>mywork=C:/home/worklib</code></p>
--	---

## vlogcomp コマンドの例

**vlogcomp** コマンドは、次のように使用します。

`run32.txt` というファイルに保存されたオプションを使用して Verilog コンパイラを実行するには、次のように入力します。

```
vlogcomp -f run32.txt
```

`/home/smithjj/mylib` ディレクトリにある論理名が `mysimwork` の作業ライブラリを使用して Verilog コンパイラを起動し、プロジェクト ファイル `dsp64.prj` にリストされている Verilog ファイルをすべてコンパイルするには、次のように入力します。

```
vlogcomp -work mysimwork=/home/smithjj/mylib -prj dsp64.prj
```

`xilinxsim.ini` ファイルで指定されたデフォルトの作業ライブラリを使用して Verilog コンパイラを起動し、`suba.v` および `subb.v` という Verilog ファイルをコンパイルするには、次のように入力します。

```
vlogcomp suba.v subb.v
```

## vhpcomp

### vhpcomp コマンドの概要と構文

**メモ：** 次の情報は、アドバンス ユーザーを対象としています。

ISim では、VHDL コンパイラ `vhpcomp` を使用して VHDL ソース ファイルを解析し、これらのファイルに含まれているすべてのデザイン ユニットのオブジェクト コードを生成します。`vhpcomp` で生成されたオブジェクト コードは `fuse` によりリンクされ、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 VHDL ファイルを指定する必要があります。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。

#### 構文

```
vhpcomp (option)
```

*option* には、「[vhpcomp コマンドのオプション](#)」に示すどのオプションも入力できます。

**メモ：** このコマンドでは、大文字/小文字が区別されます。

### vhpcomp コマンドのオプション

次に、**vhpcomp** コマンドのオプションを示します。



<code>&lt;vhdl_file&gt;</code>	コンパイルする VHDL ソース ファイルを指定します。
<code>-f &lt;cmd_file&gt;</code>	vhpcomp オプションをテキストファイルに記述して保存すると、vhpcomp を実行する際に毎回オプションを入力する必要がないので便利です。-f オプションを使用すると、 <i>cmd_file</i> に保存されたオプションを使用して vlogcomp が実行されます。
<code>-h</code>	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。
<code>-incremental</code>	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。
<code>-intstyle</code> <code>ise   xflow   silent</code>	メッセージの表示方法を指定します。 <b>ise</b> では、ISE® の [Console] パネルのメッセージを表示し、 <b>xflow</b> では XFLOW のメッセージを表示します。 <b>silent</b> に設定すると、メッセージは表示されません。デフォルトでは、すべてのメッセージが表示されます。
<code>-ise &lt;file&gt;</code>	ザイリンクス ISE ファイルを指定できます。
<code>-L -lib &lt;search_lib&gt;</code> <code>[=&lt;lib_path&gt;]</code>	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。複数の <b>-lib</b> オプションを指定でき、指定したライブラリはリソース ライブラリとして扱われます。このオプションで物理パスを指定すると、 <b>xilinxsim.ini</b> ファイルで指定されているマップが無視されます。  <i>search_lib</i> は指定のライブラリの論理名、 <i>lib_path</i> は物理ライブラリへのパスを指定します。例： <b>-lib mylib=C:/home/mylib</b>
<code>-nodebug</code>	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。
<code>-prj &lt;prj_file&gt; .prj</code>	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 ソフトウェアにより使用される主要ソース ファイルです。 <i>prj_file</i> はプロジェクト ファイルへのパスを指定します。拡張子は <b>.prj</b> である必要があります。プロジェクト ファイルには、絶対パスまたは相対パスを指定できます。相対パスには、../ を含める必要があります。

<b>-rangecheck</b>	<p>ランタイムの値の範囲チェックを有効にします (VHDL のみ)。このオプションを指定すると、VHDL 信号に割り当てられた値が有効な範囲内にあるかどうかチェックされます。たとえば、信号が正と宣言されている場合は、信号は負の値に割り当てられていないかどうかチェックされ、信号が <code>std_logic</code> と宣言されている場合は、信号に有効な <code>std_logic</code> 値 (U、X、0、1、Z、W、L、H、-) のみが割り当てられているかどうかチェックされます。</p> <p><b>メモ：</b> このオプションは、インデックスの範囲のチェックには関係ありません。インデックスの範囲は常にチェックされます。</p> <p>デフォルトではオフです。</p>
<b>-v &lt;value&gt;</b>	<p>メッセージの表示レベルを指定します。0、1、2 のいずれかを設定でき、デフォルトは 0 です。</p>
<b>-work &lt;work_lib&gt; [=&lt;lib_path&gt;]</b>	<p>作業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、<code>xilinxsim.ini</code> 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 <b>work</b> です。</p> <p><i>work_lib</i> は指定の作業ライブラリの論理名、<i>lib_path</i> は物理ライブラリへのパスを指定します。 例： <b>mywork=C:/home/worklib</b></p>

## vhpcomp コマンドの例

**vhpcomp** コマンドは、次のように使用します。

**run32.txt** というファイルに保存されたオプションを使用して VHDL コンパイラを実行するには、次のように入力します。

```
vhpcomp -f run32.txt
```

/home/smithjj/mylib ディレクトリにある論理名が **mysimwork** の作業ライブラリを使用して VHDL コンパイラを起動し、プロジェクト ファイル **dsp64.prj** にリストされている VHDL ファイルをすべてコンパイルするには、次のように入力します。

```
vhpcomp -work mysimwork=/home/smithjj/mylib -prj dsp64.prj
```

`xilinxsim.ini` ファイルで指定されたデフォルトの作業ライブラリを使用して VHDL コンパイラを起動し、**suba.vhd** および **subb.vhd** という VHDL ファイルをコンパイルするには、次のように入力します。

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

## サードパーティ コマンドの等価性

### サードパーティのシミュレーション コマンドのサポートの概要

ISim では、サードパーティのコマンドがサポートされません。ISim で DO ファイル (\*.do) ファイルに含まれるシミュレーション コマンドを使用する場合は、**run** などの完全に同一の ISim コマンドがない限り認識されません。

DO ファイルのコマンドは、次の情報に従い、ISim コマンドにマップできます。

- ・ サードパーティのコンパイラ コマンド
- ・ サードパーティの Tcl コマンド

## サードパーティのコンパイラ コマンド

次の表は、DO ファイル コンパイル コマンドを ISim コマンドにマップするときに参照してください。

コンパイラ コマンド互換性表

DO ファイル コマンド	ISim コマンド	説明
<b>vcom</b> -work <libname> -93 <file_name>	<b>vhpcomp</b> <file_name>	VHDL ファイルをコンパイルします。
<b>vlog</b> -work <libname> <file_name>	<b>vlogcomp</b> <file_name>	Verilog ファイルをコンパイルします。
<b>vsim</b> <lib_name>.<design_name>	<b>fuse</b> -lib <lib_name> <design_name>	シミュレーション実行ファイル を生成します。
<b>vsim</b> <lib_name>.<design_name> <mti.do>	<executable_name>.exe <b>-tclbatch</b> <design_name>.&b>tclbatch	シミュレーションを実行します。
<b>vsim</b> [-sdfmin   -sdfmax ] [<instance>=<sdf_filename>] [-sdfnoerror] [-sdfnowarn] [+sdf_verbose]  sdf コマンドは、vsim コマンドの コマンド引数としてのみ起動で きます。	<b>sdfanno</b> {-min -typ -max} <file_name> [-nowarn] [-noerror] [-root<path_name>]	SDF アノテーションを実行 します。

## サードパーティの Tcl コマンド

次の表は、DO ファイル シミュレーション コマンドを ISim コマンドにマップするときに参照してください。

シミュレーション Tcl コマンド互換性表

DO ファイル コマンド	ISim コマンド	説明
<b>bd</b> id# (複数指定可能)	<b>bp del</b> <index>[<index> ... ]	インデックスに基づいてブレークポイントを削除します。index は、削除する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。
<b>bd</b> <file_name> <line_number>	<b>bp remove</b> <file_name> <line_number>	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。

DO ファイル コマンド	ISim コマンド	説明
<b>bd</b> <file_name> <line_number>   <id#>	<b>bp remove</b> <file_name> <line_number>	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。
<b>bp</b> <file_name> <line_number>	<b>bp add</b> <file_name> <line_number>	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。無視されるオプション：[-id <id#>]、[-inst <region>]、[-cond {<condition_expression>}]
<b>bp -query</b> <file_name>	<b>bp list</b>	ブレークポイントをすべて表示します。
<b>drivers</b> <item name>	<b>show driver</b> <net_name>	指定された <net_name> を駆動するドライバをすべて表示します。
<b>env</b>	<b>scope</b>	デザイン階層の現在の位置を表示します。
<b>env ..</b>	<b>scope ..</b>	現在の階層の親に変更します。
<b>env</b> <pathname>	<b>scope</b> path_name	<path_name> で指定された階層に変更します。  無視されるオプション： <b>-nodataset</b> 、 <b>-dataset</b>
<b>examine</b> <signal_name>	<b>show value</b> <signal_name>	信号の値を表示します。
<b>exit</b>	<b>exit</b>	ISE Simulator を終了します。
<b>force -deposit</b> <signal_name> <value> [<time>]	<b>put</b>	新しい値に変更します。ただし新しい値は HDL での割り当てにより上書きされます。
<b>force -freeze</b>	<b>isim force add</b>	HDL でのすべての割り当てを上書きし、信号/ワイヤを特定の値に固定します。
<b>help</b>	<b>help</b>	すべての Tcl コマンドとその使用方法を表示します。
<b>help</b> [command   topic]	<b>help</b> <command>	コマンドのヘルプ情報を表示します。
<b>if</b> { [exa sig_a] == "0011ZZ" } {echo"Signal value matches"}	<b>test</b> <signal_name> <value>	表示されている信号値をテストします。
<b>noforce</b> <signal>	<b>isim force remove</b> <signal>	信号の値を削除します。isim force remove コマンドが発行されるまで信号に値が適用されます。このコマンドは、VHDL の信号および Verilog のワイヤのみで使用できます。Verilog のレジスタには使用できません。
<b>quit</b>	<b>quit</b>	Tcl プロンプトを終了します。  無視されるオプション：[-f   -force] [-sim]

DO ファイル コマンド	ISim コマンド	説明
<b>radix</b>	<b>isim get radix</b>	Tcl 変数の文字列としてデフォルトの基数を戻し、デフォルトの基数を stdout に表示します。
<b>radix -&lt;radix_type&gt;</b>	<b>isim get radix &lt;radix_type&gt;</b>	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。
<b>restart</b>	<b>restart</b>	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。
<b>run &lt;length&gt; &lt;unit&gt;</b>	<b>run &lt;length&gt; &lt;unit&gt;</b>	<length> <unit> 時間シミュレーションを実行します。
<b>run -all -continue</b>	<b>run {all   continue}</b>	イベントがすべて終了するまで、シミュレーションを実行します。
<b>run</b> では、時間ステップおよび時間ユニット分シミュレーションを実行します。  メモ： また、modelsim.ini ファイルに含まれる <i>RunLength</i> および <i>UserTimeUnit</i> 変数と共に設定します。	<b>run</b>	シミュレーションを 100ns 間実行します。
<b>show</b>	<b>show scope</b>	デザイン階層の現在の位置を表示します。
<b>show</b>	<b>show signal</b>	現在のブロック内にある信号およびポートをすべて表示します。
<b>show -all</b>	<b>show child -r</b>	現在のブロックの子ブロックすべてを表示します。
<b>vcd add -r</b>	<b>vcd dumpvars -m &lt;module instance&gt; -l &lt;int&gt;</b>	レベルを使用して特定のインスタンスをダンプします。  無視されるオプション： [-in] [-out] [-inout] [-internal] [-ports].
<b>vcd file &lt;file_name&gt;</b>	<b>vcd dumpfile &lt;file_name&gt;</b>	<file_name> に出力が書き出されます。  無視されるオプション： <b>-dumpports、-map</b>
<b>vcd flush</b>	<b>vcd dumpflush</b>	VCD データをファイルに書き出します。  <file_name> のサポートはありません。vcd dumpfile” で指定されたファイルが書き出されます。
<b>vcd limit &lt;size&gt;</b>	<b>vcd dumplimit &lt;no_of_bytes&gt;</b>	var ダンプ サイズを制限します。
<b>vcd on   off [&lt;file_name&gt;]</b>	<b>vcd {dumpon   dumpoff}</b>	VCD のトレースのオン、オフを切り替えます。

## HDL 言語のサポート

### VHDL 言語のサポート (a ~ m)

次に、ISim でサポートされる VHDL の構文要素をアルファベット順に示します。例外も示します。リストの後半は「[VHDL 言語のサポート \(n ~ z\)](#)」を参照してください。

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。
access_type_definition	
actual_designator	
actual_parameter_part	
actual_part	
adding_operator	
aggregate	aggregate 内で choice direction 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"> <li>・ エイリアスのエイリアス</li> <li>・ subtype_indication のないエイリアス宣言</li> <li>・ エイリアス宣言でのシグネチャ</li> <li>・ alias_designator としての演算子シンボル</li> <li>・ 演算子シンボルのエイリアス</li> <li>・ alias_designator としての文字列リテラル</li> </ul>
alias_designator	<ul style="list-style-type: none"> <li>・ alias_designator としての operator_symbol は サポートされません。</li> <li>・ alias_designator としての character_literal は サポートされません。</li> </ul>
allocator	
architecture_body	
architecture_declarative_part	
architecture_statement_part	
array_type_definition	
assertion	
assertion_statement	
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 るようになりました。フォーマル内のローカル以外 のスタティック インデックス/スライスはエラーで あり、メッセージが表示されます。ただし、フォー マル名の接頭辞がインデックス/スライス/選択さ れている場合、インデックス/スライス/選択された フォーマル名はサポートされません。

サポートされる VHDL 構文	例外
association_list	
attribute_declaration	
attribute_designator	
attribute_name	<p>接頭辞の後の signature はサポートされません。次の定義済み属性がサポートされています。</p> <ul style="list-style-type: none"> <li>・ A'ACTIVE、A'ASCENDING([N])、A'HIGH([N])、A'LENGTH([N])、A'LEFT([N])、A'LOW([N])、A'RANGE([N])、A'REVERCE_RANGE([N])、A'RIGHT([N])</li> <li>・ S'DELAYED[(T)]、S'EVENT、S'LAST_ACTIVE、S'LAST_EVENT、S'LAST_VALUE</li> <li>・ T'ASCENDING、T'BASE、T'HIGH、T'IMAGE(X)、T'LEFT、T'LEFTOF(X)、T'LOW、T'POS(X)、T'PRED(X)、T'RIGHT、T'RIGHTOF(X)、T'SUCC(X)、T'VAL(X)</li> </ul>
attribute_specification	
base	
base_specifier	
base_unit_declaration	
based_integer	
based_literal	
basic_character	
basic_graphic_character	
basic_identifier	
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。
bit_value	
block_configuration	
block_declarative_item	
block_declarative_part	
block_header	
block_specification	
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。
case_statement	
case_statement_alternative	
character_literal	

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。
choices	
component_configuration	
component_declaration	
component_instantiation_statement	
component_specification	
composite_type_definition	
concurrent_assertion_statement	postponed はサポートされません。
concurrent_procedure_call_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。
condition	
condition_clause	
conditional_signal_assignment	ガード付き信号代入はサポートされないため、オプションの一部としてキーワード <b>guarded</b> を使用することはサポートされません。
conditional_waveform	
configuration_declaration	コンフィギュレーション使用するインデックスを生成するのにローカル以外のスタティックを使用することはサポートされません。
configuration_declarative_item	
configuration_declarative_part	
configuration_item	
configuration_specification	
constant_declaration	
constrained_array_definition	
constraint	
context_clause	
context_item	
decimal_literal	
declaration	
delay_mechanism	
design_file	
design_unit	
designator	
direction	
discrete_range	
element_association	



サポートされる VHDL 構文	例外
element_declaration	
element_subtype_definition	
entity_aspect	
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。
entity_class_entry_list	
entity_declaration	
entity_declarative_item	
entity_declarative_part	
entity_designator	
entity_header	
entity_name_list	
entity_specification	
entity_tag	
enumeration_literal	
enumeration_type_definition	
exit_statement	
exponent	
expression	
extended_digit	
extended_identifier	
factor	
file_declaration	
file_logical_name	file_logical_name には文字列値を評価するワイルドカードも許可されていますが、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。
file_open_information	
file_type_definition	
floating_type_definition	
formal_designator	
formal_parameter_list	
formal_part	
full_type_declaration	
function_call	function_call 内の名前付きパラメータ連結では、フォーマルのスライス、インデックス、選択はサポートされません。
generate_statement	
generate_scheme	

サポートされる VHDL 構文	例外
generic_clause	
generic_list	
generic_map_aspect	
graphic_character	
identifier	
identifier_list	
if_statement	
incomplete_type_declaration	
index_constraint	
index_specification	
index_subtype_definition	
indexed_name	
instantiated_unit	ダイレクト コンフィギュレーションのインスタネーションはサポートされていません。
instantiation_list	
integer	
integer_type_definition	
interface_constant_declaration	
interface_declaration	
interface_element	
interface_file_declaration	
interface_list	
interface_signal_declaration	
interface_variable_declaration	
iteration_scheme	
label	
letter	
letter_or_digit	
library_clause	
library_unit	
literal	
logical_name	
logical_name_list	
logical_operator	
loop_statement	
miscellaneous_operator	
mode	リンケージ ポートおよびバッファ ポートは、完全にはサポートされません。
multiplying_operator	

## VHDL 言語のサポート (n ~ z)

ISim では次の VHDL 構文がサポートされています。例外がある場合は記述されています。構文はアルファベット順に示されています。リストの前半は、「[VHDL 言語のサポート \(a ~ m\)](#)」を参照してください。

サポートされる VHDL 構文	例外
name	
next_statement	
numeric_literal	
object_declaration	
operator_symbol	
options	guarded はサポートされません。
package_body	
package_body_declarative_item	
package_body_declarative_part	
package_declaration	
package_declaration_item	
package_declarative_part	
parameter_specification	
physical_literal	
physical_type_definition	
port_clause	
port_list	
port_map_aspect	
prefix	
primary	primary を使用した場所では、allocator は展開されます。
primary_unit	
procedure_call	procedure_call 内の名前付きパラメータ連結では、フォーマルのスライス、インデックス、選択はサポートされません。
procedure_call_statement	
process_declarative_item	
process_declarative_part	
process_statement	実行延期プロセスはサポートされません。
process_statement_part	
qualified_expression	
range	
range_constraint	
record_type_definition	
relation	
relational_operator	

サポートされる VHDL 構文	例外
report_statement	
return_statement	
scalar_type_definition	
secondary_unit	
secondary_unit_declaration	
selected_name	
selected_signal_assignment	ガード付き信号代入はサポートされないため、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。
selected_waveform	
sensitivity_clause	
sensitivity_list	
sequence_of_statements	
sequential_statement	
shift_expression	
shift_operator	
sign	
signal_assignment_statement	
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。
signal_list	
signature	
simple_expression	
simple_name	
slice_name	
string_literal	
subprogram_body	
subprogram_declaration	
subprogram_declarative_item	
subprogram_declarative_part	
subprogram_kind	
subprogram_specification	
subprogram_statement_part	
subtype_declaration	
subtype_indication	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。
suffix	
target	
term	

サポートされる VHDL 構文	例外
timeout_clause	
type_conversion	
type_declaration	
type_definition	
type_mark	
unconstrained_array_defintion	
use_clause	
variable_assignment_statement	
variable_declaration	
wait_statement	
waveform	<b>unaffected</b> 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 言語のサポート

### ビヘイビア文構文

次の Verilog ビヘイビア文構文が ISim でサポートされています。

#### 継続代入文

Verilog 構文	ISim サポート
continuous_assign	あり
list_of_net_assignments	あり
net_assignment	あり

#### 手続きブロックおよび代入文

Verilog 構文	ISim サポート
initial_construct	あり
always_construct	あり
blocking_assignment	あり
nonblocking_assignment	あり
procedural_continuous_assignment	あり
function_blocking_assignment	あり
function_statement_or_null	あり

## 並列および配列ブロック

Verilog 構文	ISim サポート	コメント
function_seq_block	あり	
variable_assignment	あり	
par_block	一部あり	タスクまたは関数内では fork 文および join 文はサポートされません。
seq_block	あり	

## 文

Verilog 構文	ISim サポート
statement	あり
statement_or_null	あり
function_statement	あり

## タイミング制御文

Verilog 構文	ISim サポート
delay_control	あり
delay_or_event_control	あり
disable_statement	あり
event_control	あり
event_trigger	あり
event_expression	あり
procedural_timing_control_statement	あり
wait_statement	あり

## 条件文

Verilog 構文	ISim サポート
conditional_statement	あり
if_else_if_statement	あり
function_conditional_statement	あり
function_if_else_if_statement	あり

## case 文

Verilog 構文	ISim サポート
case_statement	あり
case_item	あり
function_case_statement	あり
function_case_item	あり

## loop 文

Verilog 構文	ISim サポート
function_loop_statement	あり
loop_statement	あり

## タスク イネーブル文

Verilog 構文	ISim サポート
system_task_enable	あり
task_enable	あり

## コンパイラ指示子構文

次の Verilog コンパイラ指示子構文が ISim でサポートされています。

## コンパイラ指示子構文

Verilog 構文	ISim サポート	コメント
'celldefine	サポートなし	
'endcelldefine	サポートなし	
'default_nettype	あり	
'define	あり	
'undef	あり	ISim では、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。
'ifdef	あり	
'ifndef	あり	
'elsif	あり	
'else	あり	
'endif	あり	
'include	あり	
'resetall	あり	
'line	あり	
'timescale	あり	
'unconnected_drive	サポートなし	
'nounconnected_driv	サポートなし	

## 宣言構文

次の Verilog 宣言構文が ISim でサポートされています。

## モジュール パラメータ宣言

Verilog 構文	ISim サポート
local_parameter_declaration	あり
parameter_declaration	あり
specparam_declaration	あり

## 型宣言

Verilog 構文	ISim サポート
event_declaration	あり
genvar_declaration	あり
integer_declaration	あり
net_declaration	あり
real_declaration	あり
reg_declaration	あり
time_declaration	あり

## ネット型および変数型

Verilog 構文	ISim サポート
net_type	あり
output_variable_type	あり
real_type	あり
variable_type	あり

## 強度

Verilog 構文	ISim サポート
drive_strength	あり
strength0	あり
strength1	あり
charge_strength	サポートなし

## 遅延

Verilog 構文	ISim サポート
delay2	あり
delay3	あり
delay_value	あり



## 宣言リスト

Verilog 構文	ISim サポート
list_of_event_identifiers	あり
list_of_genvar_identifiers	あり
list_of_net_decl_assignments	あり
list_of_net_identifiers	あり
list_of_param_assignments	あり
list_of_port_identifiers	あり
list_of_real_identifiers	あり
list_of_specparam_assignments	あり
list_of_variable_identifiers	あり
list_of_variable_port_identifiers	あり

## 宣言代入文

Verilog 構文	ISim サポート
net_decl_assignment	あり
param_assignment	あり
specparam_assignment	あり
pulse_control_specparam	あり
error_limit_value	あり
reject_limit_value	あり
limit_value	あり

## 宣言範囲

Verilog 構文	ISim サポート
dimension	あり
range	あり

## 関数宣言

Verilog 構文	ISim サポート
function_declaration	あり
function_item_declaration	あり
function_port_list	あり
range_or_type	あり

## タスク宣言

Verilog 構文	ISim サポート
task_declaration	あり
task_item_declaration	あり
task_port_list	あり
task_port_item	あり
tf_input_declaration	あり
tf_output_declaration	あり
tf_inout_declaration	あり
task_port_type	あり

## ブロック アイテム宣言

Verilog 構文	ISim サポート
block_item_declaration	あり
block_reg_declaration	あり
list_of_block_variable_identifiers	あり
block_variable_type	あり

## 論理表現構文

次の Verilog 論理表現構文が ISim でサポートされています。

## 連結

Verilog 構文	ISim サポート
concatenation	あり
constant_concatenation	あり
constant_multiple_concatenation	あり
module_path_concatenation	あり
module_path_multiple_concatenation	あり
multiple_concatenation	あり
net_concatenation	あり
net_concatenation_value	あり
variable_concatenation	あり
variable_concatenation_value	あり

## 関数呼び出し

Verilog 構文	ISim サポート
constant_function_call	あり
function_call	あり
system_function_call	あり

## 論理表現

Verilog 構文	ISim サポート
base_expression	あり
conditional_expression	あり
constant_base_expression	あり
constant_expression	あり
constant_mintypmax_expression	あり
constant_range_expression	あり
dimension_constant_expression	あり
expression1	あり
expression2	あり
expression3	あり
expression	あり
lsb_constant_expression	あり
mintypmax_expression	あり
module_path_conditional_expression	あり
module_path_expression	あり
module_path_mintypmax_expression	あり
msb_constant_expression	あり
range_expression	あり
width_constant_expression	あり

## プライマリ

Verilog 構文	ISim サポート
constant_primary	あり
module_path_primary	あり
primary	あり

## 論理式の左辺

Verilog 構文	ISim サポート
net_lvalue	あり
variable_lvalue	あり

## 演算子

Verilog 構文	ISim サポート
unary_operator	あり
binary_operator	あり
unary_module_path_operator	あり
binary_module_path_operator	あり

## 数値

Verilog 構文	ISim サポート
number	あり
real_number	あり
exp	あり
decimal_number	あり
binary_number	あり
octal_number	あり
hex_number	あり
sign	あり
size	あり
non_zero_unsigned_number	あり
unsigned_number	あり
binary_value	あり
octal_value	あり
hex_value	あり
decimal_base	あり
binary_base	あり
octal_base	あり
hex_base	あり
non_zero_decimal_digit	あり
decimal_digit	あり
binary_digit	あり
octal_digit	あり
hex_digit	あり
x_digit	あり
z_digit	あり

## 文字列

Verilog 構文	ISim サポート
string	あり

## 一般構文

次の Verilog 一般構文が ISim でサポートされています。

## 属性

Verilog 構文	ISim サポート
attribute_name	サポートなし
attr_spec	サポートなし
attr_name	サポートなし

## コメント

Verilog 構文	ISim サポート
comment	あり
one_line_comment	あり
block_comment	あり
comment_text	あり

## 識別子

Verilog 構文	ISim サポート
arrayed_identifier	あり
block_identifier	あり
cell_identifier	サポートなし
config_identifier	サポートなし
escaped_arrayed_identifier	あり
escaped_hierarchical_identifier	あり
escaped_identifier	あり
event_identifier	あり
function_identifier	あり
gate_instance_identifier	あり
generate_block_identifier	あり
genvar_identifier	あり
genvar_function_identifier	サポートなし
hierarchical_block_identifier	あり
hierarchical_event_identifier	あり
hierarchical_function_identifier	あり
hierarchical_identifier	あり
hierarchical_net_identifier	あり
hierarchical_variable_identifier	あり
hierarchical_task_identifier	あり
identifier	あり
inout_port_identifier	あり
input_port_identifier	あり

Verilog 構文	ISim サポート
instance_identifier	あり
library_identifier	サポートなし
memory_identifier	あり
module_identifier	あり
module_instance_identifier	あり
net_identifier	あり
output_port_identifier	あり
parameter_identifier	あり
port_identifier	あり
real_identifier	あり
simple_arrayed_identifier	あり
simple_hierarchical_identifier	あり
simple_identifier	あり
specparam_identifier	あり
system_function_identifier	あり
system_task_identifier	あり
task_identifier	あり
terminal_identifier	あり
text_macro_identifier	あり
topmodule_identifier	あり
udp_identifier	あり
udp_instance_identifier	あり
variable_identifier	あり

### 分岐

Verilog 構文	ISim サポート
simple_hierarchical_branch	あり
escaped_hierarchical_branch	あり

### 空白

Verilog 構文	ISim サポート
white_space	あり

## プリミティブおよびモジュールのインスタンス構文

次の Verilog プリミティブ インスタンスおよびモジュール インスタンス構文が ISim でサポートされています。

## プリミティブ インスタンスレーション

Verilog 構文	ISim サポート
gate_instantiation	あり
cmos_switch_instance	サポートなし
enable_gate_instance	あり
mos_switch_instance	サポートなし
n_input_gate_instance	あり
n_output_gate_instance	あり
pass_switch_instance	サポートなし
pass_enable_switch_instance	サポートなし
pull_gate_instance	あり
name_of_gate_instance	あり

## プリミティブ強度

Verilog 構文	ISim サポート
pulldown_strength	あり
pullup_strength	あり

## プリミティブ終端

Verilog 構文	ISim サポート
enable_terminal	あり
inout_terminal	あり
input_terminal	あり
ncontrol_terminal	あり
output_terminal	あり
pcontrol_terminal	あり

## プリミティブのゲート型およびスイッチ型

Verilog 構文	ISim サポート
cmos_switchtype	サポートなし
enable_gatetype	あり
mos_switchtype	サポートなし
n_input_gatetype	あり
n_output_gatetype	あり
pass_en_switchtype	サポートなし
pass_switchtype	サポートなし

## モジュール インスタンス化

Verilog 構文	ISim サポート	コメント
module_instantiation	あり	
parameter_value_assignment	あり	
list_of_parameter_assignments	あり	
ordered_parameter_assignment	あり	
named_parameter_assignment	あり	
module_instance	あり	
name_of_instance	一部あり	モジュール インスタンス配列はサポートされません。
list_of_port_connections	あり	
ordered_port_connection	あり	
named_port_connection	あり	

## 生成されたインスタンス化

Verilog 構文	ISim サポート	コメント
generated_instantiation	あり	
generate_item_or_null	一部あり	<p>module_or_generate_item は、サポートされません。</p> <p>1364-2001 Verilog 標準規格：</p> <pre>generate_item_or_null ::= generate_conditional_statement   generate_case_statement   generate_loop_statement   generate_block   module_or_generate_item</pre> <p>ISim でのサポート：</p> <pre>generate_item_or_null ::= generate_conditional_statement   generate_case_statement   generate_loop_statement   generate_block</pre>
generate_item	あり	
generate_conditional_statement	あり	
generate_case_statement	あり	
generate_case_item	あり	
generate_loop_statement	あり	
genvar_assignment	一部あり	<p>すべての generate ブロックに名前を付ける必要があります。</p> <p>1364-2001 Verilog 標準規格：</p> <pre>generate_block ::= begin [ : generate_block_identifier ] { generate_item } end</pre> <p>ISim でのサポート：</p>



Verilog 構文	ISim サポート	コメント
		<pre>generate_block ::= begin : generate_block_identifier { generate_item } end</pre>

## ソース テキスト構文

次の Verilog ソース テキスト構文が ISim でサポートされています。

### ライブラリ ソース テキスト

Verilog 構文	ISim サポート	コメント
library_text	サポートなし	
library_descriptions	サポートなし	
library_declaration	サポートなし	
file_path_spec	サポートなし	
include_statement	サポートなし	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

Verilog 構文	ISim サポート
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
inst_clause	サポートなし
inst_name	サポートなし
cell_clause	サポートなし
liblist_clause	サポートなし
use_clause	サポートなし

### モジュールおよびプリミティブのソース テキスト

ISim サポート	ISim サポート
source_text	あり
説明	あり
module_declaration	あり
module_keyword	あり

## モジュール パラメータおよびポート

Verilog 構文	ISim サポート
module_parameter_port_list	あり
list_of_ports	あり
list_of_port_declarations	あり
port	あり
port_expression	あり
port_reference	あり
port_declaration	あり

## モジュール アイテム

Verilog 構文	ISim サポート
module_item	あり
module_or_generate_item	あり
module_or_generate_item_declaration	あり
non_port_module_item	あり
parameter_override	あり

## Specify 関数構文

次の Verilog Specify 関数構文が ISim でサポートされています。

## Specify ブロック宣言

Verilog 構文	ISim サポート
specify_block	あり
specify_item	あり
pulsetyle_declaration	あり
showcancelled_declaration	あり

## Specify パス宣言

Verilog 構文	ISim サポート
path_declaration	あり
simple_path_declaration	あり
parallel_path_declaration	あり
full_path_description	あり
list_of_path_inputs	あり
list_of_path_outputs	あり

## Specify ブロック終端

Verilog 構文	ISim サポート
specify_input_terminal_descriptor	あり
specify_output_terminal_descriptor	あり
input_identifier	あり
output_identifier	あり

## Specify パス遅延

Verilog 構文	ISim サポート
path_delay_value	あり
list_of_path_delay_expressions	あり
t_path_delay_expression	あり
trise_path_delay_expression	あり
tfall_path_delay_expression	あり
tz_path_delay_expression	あり
t01_path_delay_expression	あり
t10_path_delay_expression	あり
t0z_path_delay_expression	あり
tz1_path_delay_expression	あり
t1z_path_delay_expression	あり
tz0_path_delay_expression	あり
t0x_path_delay_expression	あり
tx1_path_delay_expression	あり
t1x_path_delay_expression	あり
tx0_path_delay_expression	あり
txz_path_delay_expression	あり
tzx_path_delay_expression	あり
path_delay_expression	あり
edge_sensitive_path_declaration	あり
parallel_edge_sensitive_path_declaration	あり
full_edge_sensitive_path_declaration	あり
data_source_expression	あり
edge_identifier	あり
state_dependent_path_declaration	あり
polarity_operator	あり

## システム タイミング チェック コマンド

Verilog 構文	ISim サポート
system_timing_check	あり
\$hold_timing_check	あり
\$setuphold_timing_check	あり
\$recovery_timing_check	あり
\$removal_timing_check	あり
\$recrem_timing_check	あり
\$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
\$period_timing_check	あり
\$width_timing_check	あり
\$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

Verilog 構文	ISim サポート
checktime_condition	サポートなし
controlled_reference_event	あり
data_event	あり
delayed_data	サポートなし
delayed_reference	サポートなし
end_edge_offset	サポートなし
event_based_flag	サポートなし
notify_reg	あり
reference_event	あり
remain_active_flag	サポートなし
stamptime_condition	サポートなし
start_edge_offset	サポートなし
threshold	あり
timing_check_limit	あり

## システム タイミング チェック イベント定義

Verilog 構文	ISim サポート
timing_check_event	あり
controlled_timing_check_event	あり
timing_check_event_control	あり
specify_terminal_descriptor	あり
edge_control_specifier	あり
edge_descriptor	あり
zero_or_one	あり
z_or_x	あり
timing_check_condition	あり
scalar_timing_check_condition	あり
scalar_constant	あり

## システム タスクと関数の構文

次の Verilog システム タスク/関数構文が ISim でサポートされています。

## 表示システム タスク

Verilog 構文	ISim サポート
\$display	あり
\$displayb	あり
\$displayh	あり
\$displayo	あり
\$monitor	あり
\$monitorb	あり
\$monitorh	あり
\$monitro	あり
\$monitoroff	あり
\$monitoron	あり
\$strobe	あり
\$strobeb	あり
\$strobeh	あり
\$strobo	あり
\$write	あり
\$writeb	あり
\$writeh	あり
\$writeo	あり

## ファイル I/O タスク

Verilog 構文	ISim サポート
\$fclose	あり
\$fdisplay	あり
\$fdisplayb	あり
\$fdisplayh	あり
\$fdisplayo	あり
\$ferror	あり
\$fflush	あり
\$fgetc	あり
\$fgets	あり
\$fmonitor	あり
\$fmonitorb	あり
\$fmonitorh	あり
\$fmonitro	あり
\$fopen	あり
\$fread	あり
\$fscanf	あり
\$fseek	あり
\$fstrobe	あり
\$fstrobeb	あり
\$fstrobeh	あり
\$fstrobeo	あり
\$ftell	あり
\$fwrite	あり
\$fwriteb	あり
\$fwriteh	あり
\$fwriteo	あり
\$readmemb	あり
\$readmemh	あり
\$rewind	あり
\$sdf_annotate	あり
\$sformat	あり
\$sscanf	あり
\$swrite	あり
\$swriteb	あり
\$swriteh	あり
\$swriteo	あり

Verilog 構文	ISim サポート
\$ungetc	あり

### タイムスケール タスク

Verilog 構文	ISim サポート
\$printtimescale	あり
\$timeformat	あり

### シミュレーション制御タスク

Verilog 構文	ISim サポート
\$finish	あり
\$stop	あり

### PLA モデル化タスク

Verilog 構文	ISim サポート
\$async\$and\$array	サポートなし
\$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

### 確率分析タスク

Verilog 構文	ISim サポート
\$q_add	あり
\$q_exam	あり
\$q_full	あり
\$q_initialize	あり
\$q_remove	あり

## シミュレーション時間関数

Verilog 構文	ISim サポート
\$realtime	あり
\$stime	あり
\$time	あり

## 変換関数

Verilog 構文	ISim サポート
\$bitstoreal	あり
\$realtobits	あり
\$itor	あり
\$rtoi	あり
\$signed	あり
\$unsigned	あり

## 確率分布関数

Verilog 構文	ISim サポート
\$dist_chi_square	あり
\$dist_erlang	あり
\$dist_exponential	あり
\$dist_normal	あり
\$dist_poisson	あり
\$dist_t	あり
\$dist_uniform	あり
\$random	あり

## コマンドライン入力

Verilog 構文	ISim サポート
\$test\$plusargs	あり
\$value\$plusargs	あり



## VCD (Value Change Dump) ファイル

Verilog 構文	ISim サポート
\$dumpall	あり
\$dumpfile	あり
\$dumpflush	あり
\$dumplimit	あり
\$dumpoff	あり
\$dumpon	あり
\$dumpports	サポートなし
\$dumpportsall	サポートなし
\$dumpportsflush	サポートなし
\$dumpportslimit	サポートなし
\$dumpportsoff	サポートなし
\$dumpportson	サポートなし
\$dumpvars	あり

## UDP 宣言とインスタネーション構文

次の Verilog UDP インスタネーション構文が ISim でサポートされています。

## UDP 宣言

Verilog 構文	ISim サポート
udp_declaration	あり

## UDP ポート

Verilog 構文	ISim サポート
udp_port_list	あり
udp_declaration_port_list	あり
udp_port_declaration	あり
udp_output_declaration	あり
udp_input_declaration	あり
udp_reg_declaration	あり

## UDP 本体

Verilog 構文	ISim サポート
udp_body	あり
combinational_body	あり
combinational_entry	あり
sequential_body	あり
udp_initial_statement	あり
init_val	あり
sequential_entry	あり
seq_input_list	あり
level_input_list	あり
edge_input_list	あり
edge_indicator	あり
current_state	あり
next_state	あり
output_symbol	あり
level_symbol	あり
edge_symbol	あり

## UDP インスタンスエーション

Verilog 構文	ISim サポート	コメント
udp_instantiation	あり	
udp_instance	あり	
name_of_udp_instance	一部あり	UDP インスタンス配列はサポートされません。