

Vivado Design Suite ユーザー ガイド: System Generator を 使用したモデル ベースの DSP デザイン

UG897 (v2020.1) 2020 年 6 月 3 日

この資料は表記のバージョンの英語版を翻訳したもので、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。



目次

改訂履歴.....	4
サポートされる MATLAB バージョンおよびオペレーティング システム.....	5
第 1 章: 概要	6
ザイリンクス DSP ブロックセット.....	6
FIR フィルターの生成.....	7
MATLAB のサポート.....	8
ハードウェア協調シミュレーション.....	9
システム統合プラットフォーム.....	11
System Generator での OS、MATLAB、シミュレータのサポート.....	11
第 2 章: インストール	12
ダウンロード.....	12
ザイリンクス インストーラーの使用.....	12
インストール後の操作.....	14
第 3 章: System Generator を使用したハードウェア設計	18
System Generator を使用したデザイン フロー.....	19
System Generator でのシステム レベルのモデリング.....	20
自動コード生成.....	35
MATLAB の FPGA へのコンパイル.....	43
System Generator デザインの大型システムへのインポート.....	64
コンフィギュラブル サブシステムと System Generator.....	64
FPGA デザインのパフォーマンスを向上するためのヒント.....	70
FDATool を使用したデジタル フィルター アプリケーション.....	74
複数の独立クロックのハードウェア デザイン.....	79
AXI インターフェイス.....	87
AXI4-Lite インターフェイスの生成.....	92
System Generator でのプラットフォーム ベースのアクセラレータの調整.....	103
System Generator のスーパー サンプル レート (SSR) ブロックの使用.....	109
第 4 章: System Generator での解析の実行	113
System Generator でのタイミング解析.....	114
System Generator でのリソース解析.....	121
第 5 章: ハードウェア協調シミュレーションの使用	129
ハードウェア協調シミュレーション用のモデルのコンパイル.....	130
標準ハードウェア協調シミュレーションの実行.....	134

バースト モードのハードウェア協調シミュレーションの実行.....	137
ハードウェア協調シミュレーションへの M コード アクセス.....	138
ハードウェア ボードの設定.....	139
ハードウェア協調シミュレーション ブロック.....	143
ハードウェア協調シミュレーションのクロック.....	147
ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション.....	148
ハードウェア協調シミュレーションのバースト データ転送.....	153
第 6 章: HDL モジュールのインポート	161
ブラック ボックス HDL の要件および制限事項.....	161
ブラック ボックス コンフィギュレーション M 関数.....	162
ブラック ボックスでの複数の独立クロックのサポート.....	176
HDL 協調シミュレーション.....	177
第 7 章: Black Box Configuration ウィザード	180
ウィザードの使用.....	180
Black Box Configuration ウィザードの詳細設定.....	181
第 8 章: System Generator のコンパイル タイプ.....	182
HDL ネットリスト コンパイル.....	182
ハードウェア協調シミュレーション コンパイル.....	183
IP カタログのコンパイル.....	184
合成済みチェックポイントのコンパイル.....	190
カスタム コンパイル ターゲットの作成.....	191
第 9 章: カスタム コンパイル ターゲットの作成.....	192
xilinx_compilation ベース クラス.....	192
新しいコンパイル ターゲットの作成.....	192
ベース クラス プロパティおよび API.....	194
カスタム コンパイル ターゲットの作成例.....	197
付録 A: System Generator の GUI ユーティリティ	204
Xilinx BlockAdd.....	205
[Xilinx Tools] → [Save as blockAdd default].....	206
[Xilinx BlockConnect].....	207
[Xilinx Tools] → [Terminate].....	209
ザイリンクス波形ビューアー	211
付録 B: その他のリソースおよび法的通知.....	221
ザイリンクス リソース.....	221
ソリューション センター.....	221
Documentation Navigator およびデザイン ハブ.....	221
参考資料.....	222
お読みください: 重要な法的通知.....	222

改訂履歴

次の表に、この文書の改訂履歴を示します。

セクション	改訂内容
2020 年 6 月 3 日 バージョン 2020.1	
資料全体	<ul style="list-style-type: none"> • MATLAB® コマンド プロンプトをアップデート。 • 「既存のリソース解析結果の表示」に注記を追加。 • ザイリンクス BlockAdd をアップデート。

サポートされる MATLAB バージョンおよびオペレーティング システム

System Generator では、次のバージョンの MATLAB[®] がサポートされます。

- 2019a
- 2019b
- 2020a

x86 および x86-64 プロセッサ アーキテクチャでは、次のオペレーティング システムがサポートされます。

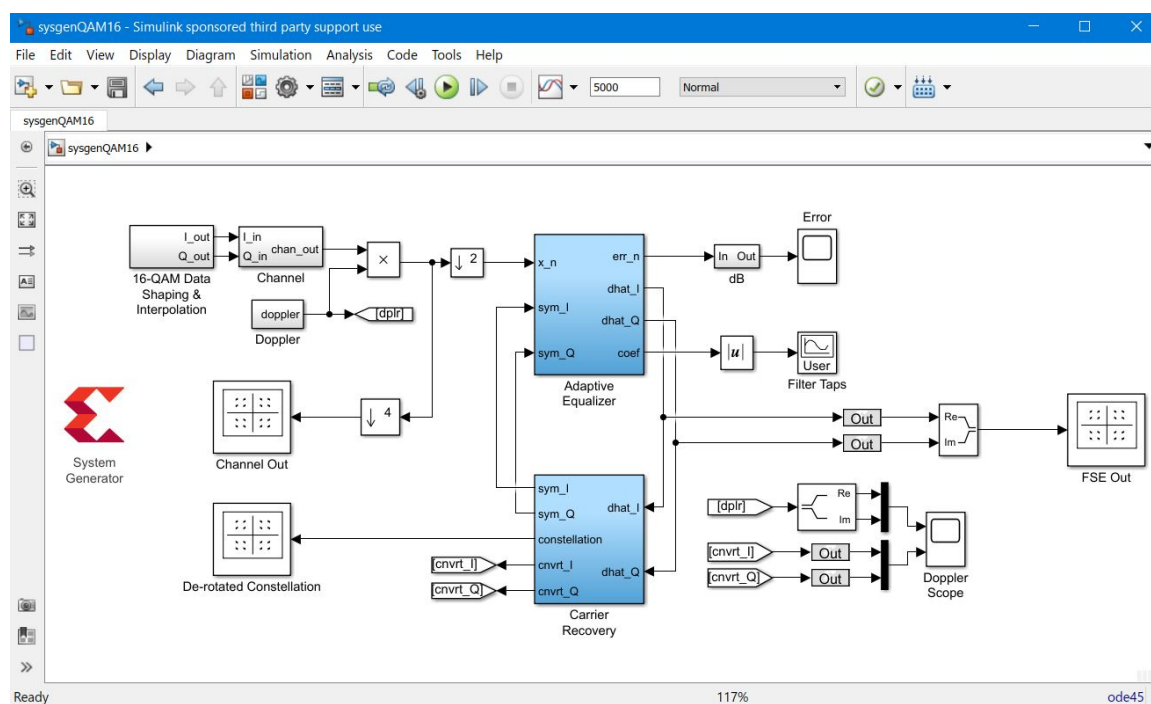
- Microsoft Windows 10.0 1809 Update、10.0 1903 Update、10.0 1909 Update (64 ビット)、英語版/日本語版
- Red Hat Enterprise Workstation/Server 7.4、7.5、7.6、7.7 (64 ビット)
- Ubuntu Linux 16.04.5 LTS、16.04.6 LTS、18.04.1 LTS、18.04.2 LTS、18.04.3 LTS、18.04.4 LTS (64 ビット)
- SUSE Linux Enterprise 12.4 (64 ビット)

注記: System Generator を MATLAB バージョン R2020a と共に使用する場合は、RHEL 7.4 OS はサポートされません。

概要

System Generator は、FPGA デザイン用に MathWorks モデル ベースの Simulink® デザイン環境を使用できるようにするザイリンクスの DSP デザイン ツールです。System Generator を使用するのに、ザイリンクス FPGA または RTL デザイン手法の使用経験は不要です。デザインは、DSP 用の Simulink モデリング環境でザイリンクスのブロックセットを使用して記述します。System Generator デザインは、IP カタログを使用して、Vivado® IDE プロジェクトにインポートできます。

図 1: System Generator デザイン

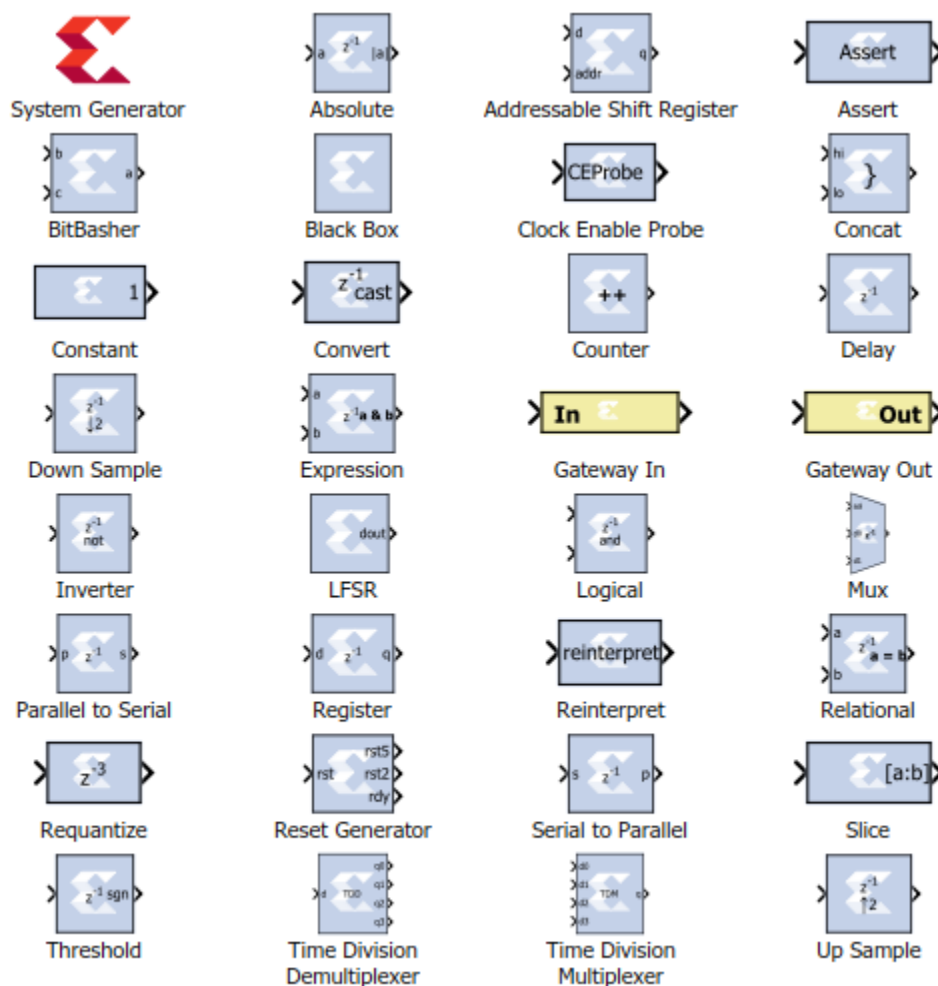


System Generator for DSP モデルを作成して Vivado IDE プロジェクトにインポートする演習および詳細な手順は、『Vivado Design Suite チュートリアル: System Generator を使用したモデル ベースの DSP デザイン』(UG948) を参照してください。

ザイリンクス DSP ブロックセット

Simulink® 用ザイリンクス DSP ブロックセットには、130 個以上の DSP 機能ブロックが含まれています。これらのブロックには、加算器、乗算器、レジスタなどのよく使用される DSP 構築ブロックが含まれるほか、フォワードエラー訂正ブロック、FFT、フィルター、メモリなどの複雑な DSP 機能ブロックも含まれます。これらのブロックを使用することで、ザイリンクスの IP コア生成を利用してデバイス用に最適化された結果を得ることができます。

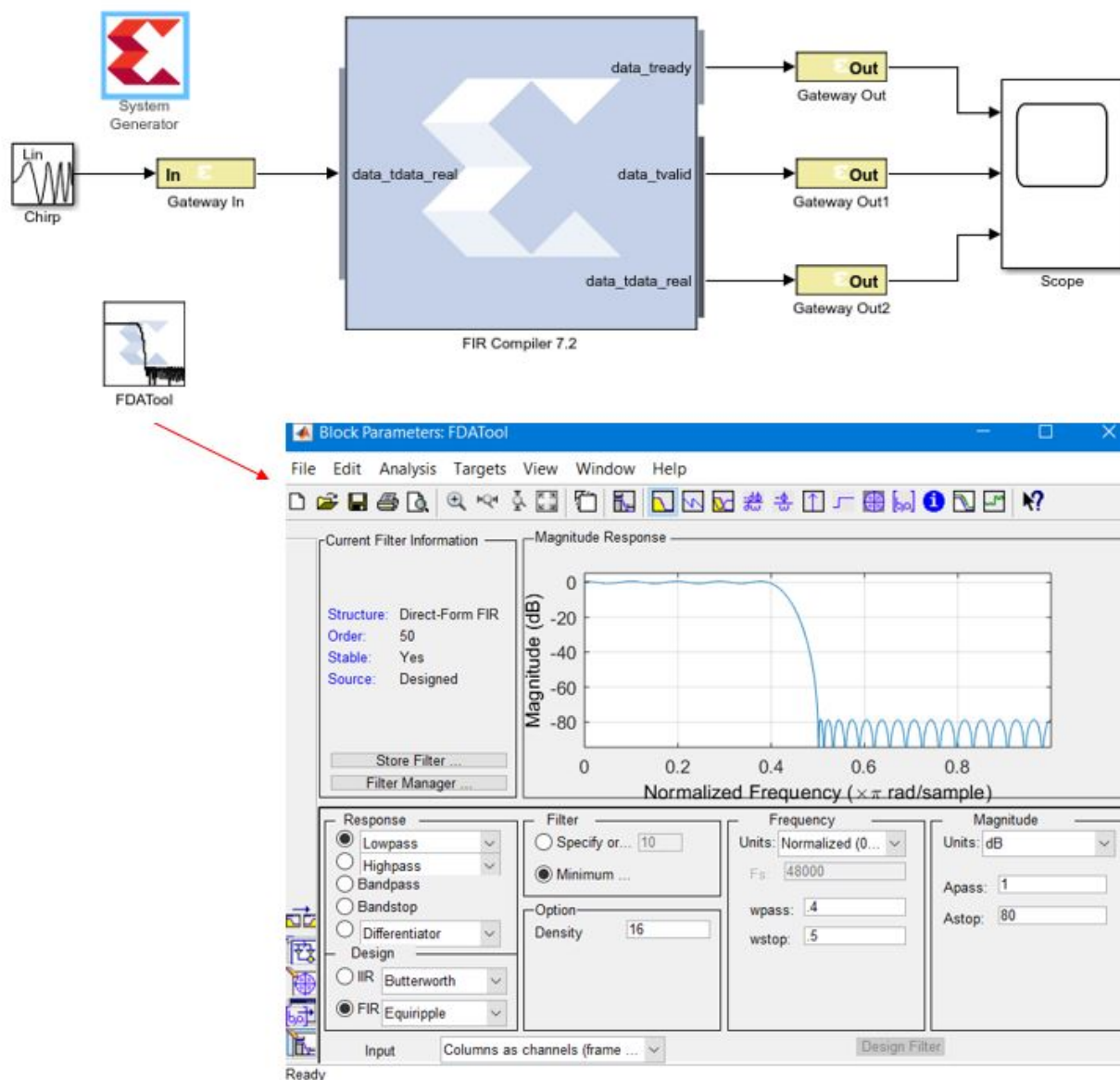
図 2: ザイリンクス DSP ブロックセット



FIR フィルターの生成

System Generator には、最適化されたインプリメンテーションを作成するため、7 シリーズおよび UltraScale™ デバイスの専用 DSP48E1 および DSP48E2 ハードウェア リソースをターゲットとする FIR Compiler ブロックが含まれています。シングル レート、補間、間引き、ヒルベルトのインプリメンテーションを生成するオプションがあります。fir2 のような標準 MATLAB® 関数または MathWorks FDA ツールを使用すると、ザイリンクス FIR Compiler の係数を作成できます。

図 3: FDA ツールの例

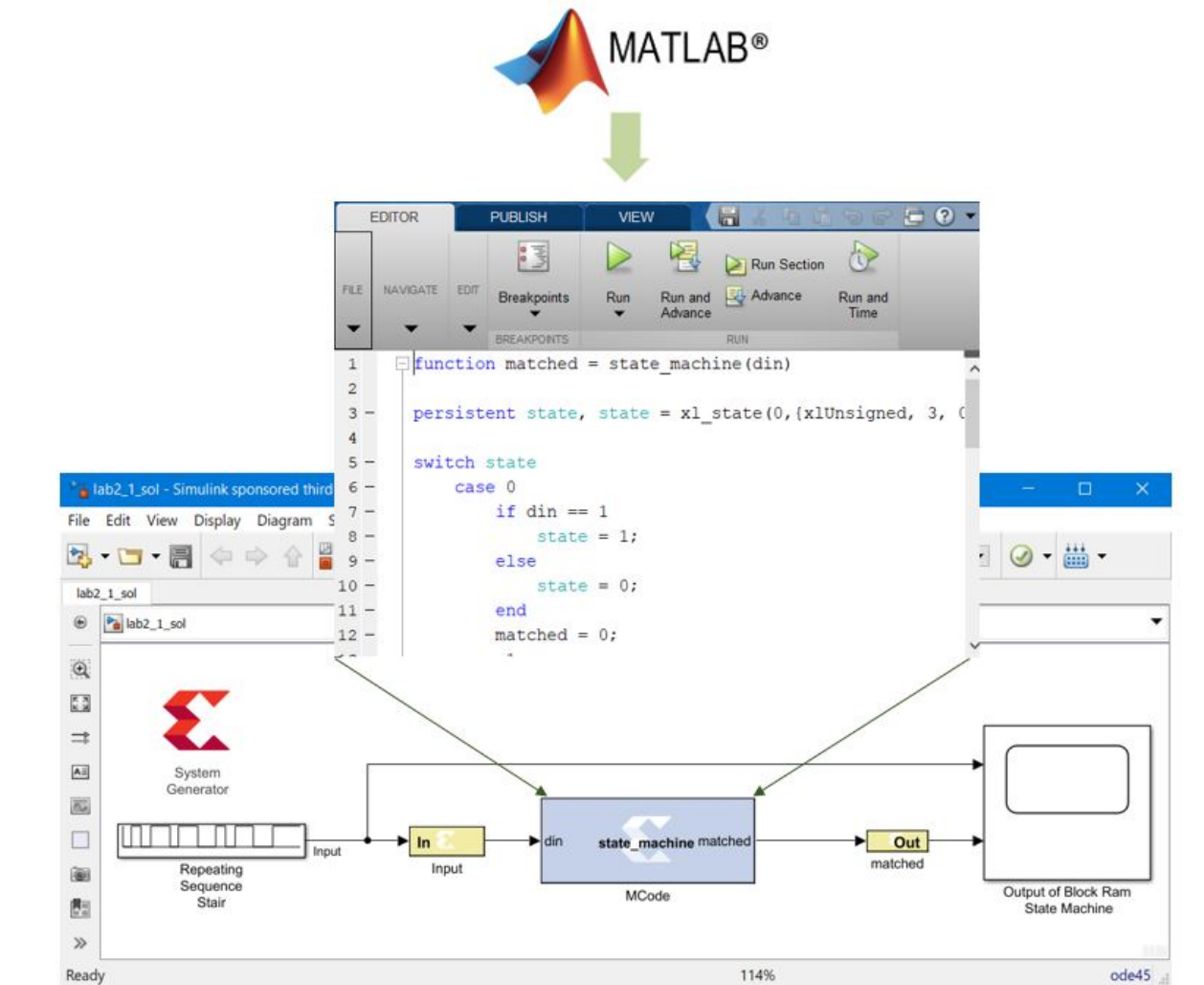


MATLAB のサポート

System Generator ライブラリは、アルゴリズムを使用しない MATLAB® を単純な制御操作のモデリングおよびインプリメンテーションに使用できるようにする MCode ブロックで構成されています。

このリリースの System Generator でサポートされている MATLAB リリースは、[サポートされる MATLAB バージョンおよびオペレーティング システム](#) を参照してください。

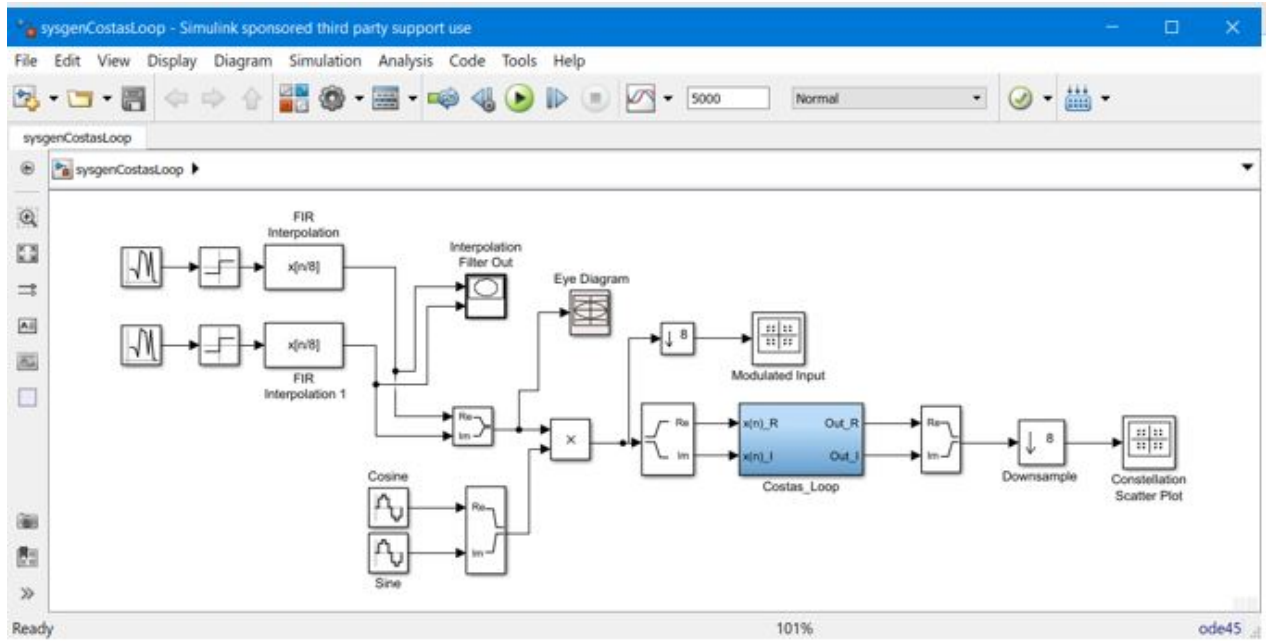
図 4: MCode ブロックの例



ハードウェア協調シミュレーション

System Generator では、ハードウェア協調シミュレーションを使用してシミュレーションを短時間で実行できます。System Generator では、サイリンクス DSP ブロックセットに取り込まれたデザインに対し、サポートされるハードウェア プラットフォームで実行可能なハードウェア シミュレーション トークンが自動的に作成されます。このハードウェアが Simulink® システムの残りの部分と協調シミュレーションされるので、シミュレーション パフォーマンスが最大 1000 倍向上します。

図 5: ハードウェア協調シミュレーション



システム統合プラットフォーム

System Generator では、DSP FPGA デザインに対するシステム統合プラットフォームが提供され、DSP システムの RTL、Simulink®、MATLAB®、および C/C++ コンポーネントを 1 つのシミュレーションおよびインプリメンテーション環境に統合できます。System Generator では、RTL を Simulink にインポートし、ModelSim またはザイリンクス Vivado® シミュレータを使用して協調シミュレーションできるブラック ボックス ブロックがサポートされ、C/C++ ソースを統合してシミュレーション可能な Vivado HLS ブロックが提供されています。

System Generator での OS、MATLAB、シミュレータのサポート

このリリースの System Generator でサポートされているオペレーティング システムは、『Vivado Design Suite ユーザー ガイド: リリース ノート、インストール、およびライセンス』 ([UG973](#)) の「サポートされるオペレーティング システム」を参照してください。

このリリースの System Generator でサポートされている MATLAB® リリースは、『Vivado Design Suite ユーザー ガイド: リリース ノート、インストール、およびライセンス』 ([UG973](#)) の「互換性のあるサードパーティ ツール」を参照してください。

インストール

ダウンロード

System Generator は、Vivado[®] Design Suite の一部であり、ザイリンクス ウェブサイトからダウンロードできます。ザイリンクス ウェブサイトの [System Generator for DSP](#) ページから、System Generator ソフトウェアを購入、登録、ダウンロードできます。

ハードウェア協調シミュレーションのサポート

FPGA 開発ボードでは、FPGA ハードウェア協調シミュレーションを Simulink[®] シミュレーションと使用する System Generator の機能を利用できます。System Generator ソフトウェアでは、すべてのザイリンクス開発ボードがサポートされています。System Generator ボード サポート パッケージは、ザイリンクス ウェブサイトの [ボードおよびキット](#) ページからダウンロードできます。

サポートされない UNC パス

System Generator では、UNC (Universal Naming Convention) のパスはサポートされていません。たとえば、System Generator では、ドライブを割り当てておかないと、共有ネットワーク ドライブにあるデザインで作業することはできません。

ザイリンクス インストーラーの使用

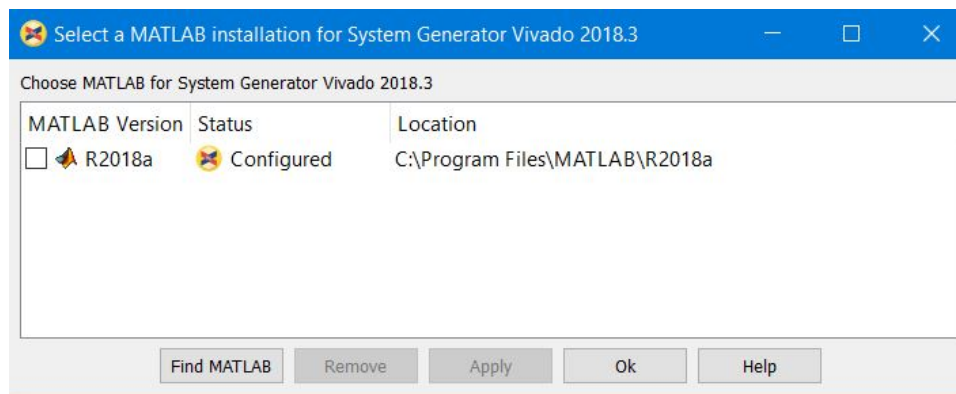
System Generator for DSP は、Vivado[®] Design Suite の一部です。System Generator をインストールするには、ザイリンクス デザイン ツールのインストーラーを使用する必要があります。

ザイリンクス デザイン ツールのインストーラーを起動する前に、MATLAB[®] のすべてのインスタンスを閉じてください。MATLAB のすべてのインスタンスを閉じたら、インストーラーを起動して画面の指示に従います。

System Generator 用の MATLAB の選択

Windows でのインストール

図 6: MATLAB の選択



このダイアログ ボックスでは、このバージョンの System Generator にサポートされる MATLAB® インストールを関連付けます。

このバージョンの System Generator と関連付ける MATLAB インストールのチェック ボックスをオンにし、関連付けるサイリンクス デザイン ツールを選択して、[Apply] をクリックします。[Apply] をクリックすると、[Status] 列が [Not Configured] から [Configured] に変わります。

使用可能なすべての MATLAB インストールがリストされます。[Status] 列には、次のいずれかの値が表示されます。

- [Unsupported]: このバージョンの MATLAB はこのバージョンの System Generator ではサポートされません。
- [Not Configured]: このバージョンの MATLAB は、このバージョンの System Generator と関連付けられていません。このバージョンの MATLAB を System Generator と関連付けるには、チェック ボックスをオンにして [Apply] をクリックします。
- [Configured]: System Generator はこのバージョンの MATLAB と使用できます。

MATLAB のバージョンがリストされていない場合は、[Find MATLAB] をクリックして有効なバージョンを検索します。

MATLAB の設定を変更するには、Windows メニューから次をクリックします。

[Start] → [All Programs] → [Xilinx Design Tools] → [2019.2] → [System Generator] → [System Generator MATLAB Configurator]

MATLAB が ISE Design Suite 用に設定されていて、その MATLAB を Vivado® IDE に設定し直す場合は、ステータスが [Configured] の MATLAB バージョンのチェック ボックスをオンにして [Remove] をクリックしてから、Vivado IDE 用に設定し直します。

Linux でのインストール

Linux で System Generator を起動するには、<Vivado_install_dir>/bin にある `sysgen` というシェル スクリプトを使用します。このスクリプトを実行する前に、PATH 環境変数に MATLAB 実行ファイルが指定されていることを確認してください。MATLAB 実行ファイルが指定されていると、System Generator を実行したときに PATH で最初に見出された MATLAB 実行ファイルが実行され、System Generator がその MATLAB セッションに関連付けられます。また、System Generator シェル スクリプトは MATLAB でサポートされるオプションをすべてサポートしており、これらをコマンド ライン引数として System Generator スクリプトに指定できます。

インストール後の操作

Linux でのインストール後の操作

ザイリンクス インストール ウィザードの指示に従ってインストールした後 System Generator を起動するには、「`sysgen`」と入力します。

これにより MATLAB® が起動し、その MATLAB セッションに System Generator が動的に追加されます。MATLAB コマンド ウィンドウの一番上に、次のメッセージが表示されます。

```
Type "xldoc" to open the Xilinx System Generator help documentation.  
Type "demo blockset xilinx" to view the demos available for Xilinx System  
Generator.
```

ザイリンクス HDL ライブラリのコンパイル

ModelSim SE で使用するライブラリをコンパイルするためのザイリンクス ツールは、`compile_simlib` です。

ザイリンクス HDL ライブラリをコンパイルするには、Vivado Design Suite を起動し、Vivado Tcl コンソールに「`compile_simlib`」と入力します。

注記: Vivado Tcl コンソールに「`compile_simlib -help`」と入力すると、この Tcl コマンドの実行に関する詳細が表示されます。

System Generator のインストールに含まれるサンプル デザイン

System Generator に含まれるサンプル モデルは、MATLAB コマンド ウィンドウの [Help] メニュー ([Help] → [Documentation] → [Xilinx]) から表示できる資料のザイリンクス セクションから入手できるほか、MATLAB コマンド プロンプトで次を入力しても入手できます。

```
>> demo blockset xilinx
```

System Generator キャッシュの管理

System Generator では、デザイン プロセスを繰り返し実行するときに時間を短縮するため、ディスク キャッシュが使用されます。キャッシュにより、シミュレーションおよび生成に関連するファイルがタグ付けされて格納されるので、次にシミュレーションおよび生成を実行するときに、これらのファイルを再生成せずに呼び出すことができ、処理時間を短縮できます。

System Generator でのボード サポートの指定

System Generator が Vivado Design Suite の一部としてインストールされている場合、System Generator で Vivado Design Suite と共にインストールされているすべてのザイリンクス開発ボードにアクセスできます。

ザイリンクス パートナーのボードも利用でき、ボードを定義するボード インターフェイス ファイル (board.xml) をパートナーのウェブサイトからダウンロードして、Vivado Design Suite の一部としてインストールできます。『Vivado Design Suite ユーザー ガイド: システム レベル デザイン入力』 (UG895) の付録 A 「ボード インターフェイス ファイル」に説明されているように、カスタム ボード インターフェイス ファイルを作成することもできます。パートナーのボードおよびカスタム ボードは、Vivado Design Suite と System Generator の両方でリポジトリに追加する必要があります。

Vivado Design Suite でボードを使用するための手順は、『Vivado Design Suite ユーザー ガイド: システム レベル デザイン入力』 (UG895) の「Vivado Design Suite プラットフォーム プラットフォームボード フローの使用」を参照してください。Vivado Design Suite では、ザイリンクス ターゲット デザイン プラットフォーム ボード (TDP) またはボード リポジトリに追加されたユーザー指定のボードを使用してプロジェクトを作成できます。特定のボードを選択すると、Vivado Design Suite ツールにボードに関する情報が表示され、IP カスタマイズおよび IP インテグレーター デザインで追加の設計アシスタンスを使用できるようになります。

パートナー ボードまたはカスタム ボードを使用できるように System Generator を設定するには、MATLAB® の startup.m ファイルにコマンドを追加する必要があります。これは、MATLAB を起動したときに実行されるファイルです。

System Generator で Simulink® モデルにボードを使用できるようにするには、次の手順に従います。

1. MATLAB コマンド ラインで `which startup.m` を実行し、MATLAB インストール ディレクトリに既に `startup.m` ファイルがあるかどうか確認します。

`which startup.m` コマンドは、MATLAB の検索パスにあるすべてのフォルダーで `startup.m` ファイルを検索します。`startup.m` ファイルが検索パスにあれば、`which startup.m` でそのファイルのフル パスが表示されます。

2. 次のように作業を続けます。

- MATLAB インストール ディレクトリに `startup.m` ファイルがある場合は、コマンド ラインに `edit startup.m` を入力し、`startup.m` ファイルを開きます。

または

- MATLAB インストール ディレクトリに `startup.m` ファイルがない場合は、MATLAB の検索パス内のフォルダーに `startup.m` ファイルを作成し、それを開きます。

`path` コマンドを実行すると、検索にあるフォルダーがリストされます。

3. `startup.m` ファイルに次のコマンドを入力します。

```
addpath([getenv('XILINX_VIVADO') '/scripts/sysgen/matlab']);
xilinx.environment.setBoardFileRepos({'<path1>', '<path2>', '...'});
```

`addpath` コマンドは、`setBoardFileRepos` ユーティリティのディレクトリを指定し、`setBoardFileRepos` は MATLAB をボード インターフェイス ファイルのディレクトリにポイントします。<path> には、ボード インターフェイス ファイル (board.xml) および board.xml ファイルで参照される part0_pins.xml や preset.xml などのファイルを含むディレクトリへのパスを指定します。また、<path> には個別のボード インターフェイス ファイルが保存されている複数のサブディレクトリを含むディレクトリを指定することもできます。

次に例を示します。

```
addpath([getenv('XILINX_VIVADO')] '/scripts/sysgen/matlab']);
xilinx.environment.setBoardFileRepos({'C:/Data/userBoards', 'C:/Data/
otherBoards'});
```

4. startup.m ファイル (MATLAB の検索パスにあるディレクトリ内) を閉じ、System Generator を閉じます。

System Generator を開くと、System Generator デザインでパートナー ボードまたはカスタム ボードがターゲット ボード (およびターゲットのザイリンクス デバイス) として使用できるようになっています。

System Generator で使用可能なパートナー ボードおよびカスタム ボードを確認するには、MATLAB コマンド ウィンドウに次のコマンドを入力します。

```
xilinx.environment.getBoardFiles
```

コマンド ウィンドウにボード インターフェイス ファイルのリストが表示されます。

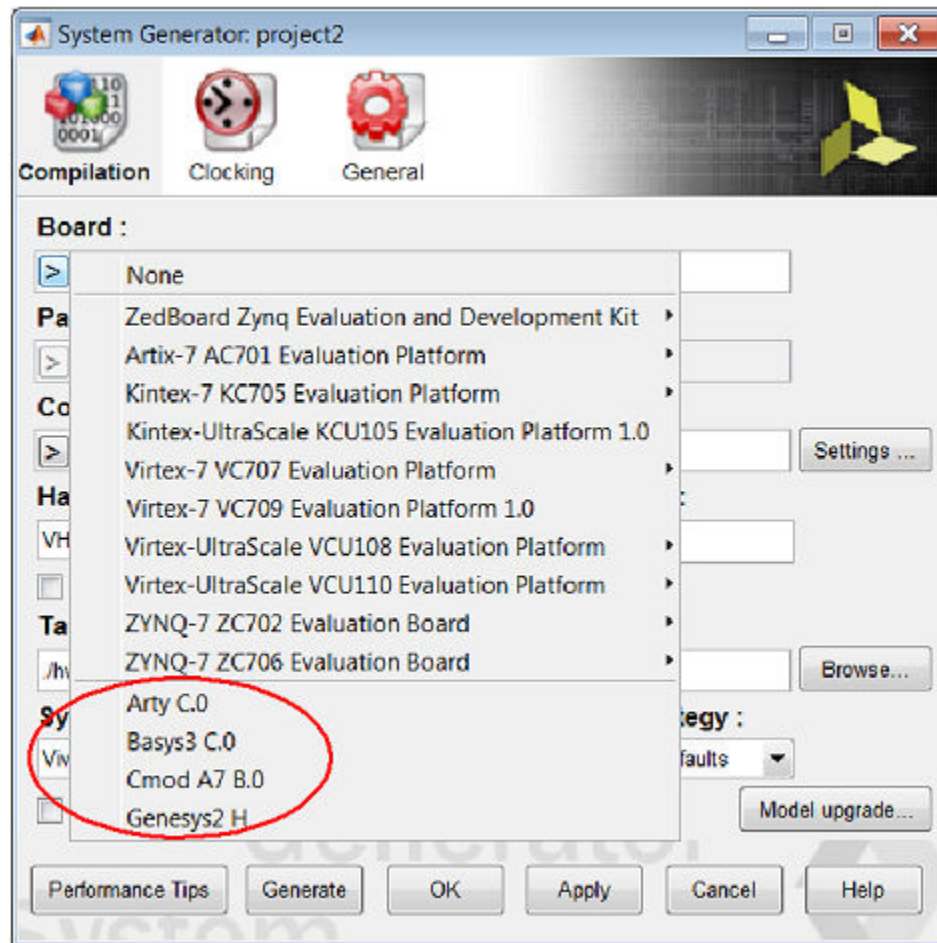
```
>> xilinx.environment.getBoardFiles

ans =

'C:\Data\usrBrds\arty\C.0\board.xml'
'C:\Data\usrBrds\basy3\C.0\board.xml'
'C:\Data\usrBrds\cm0d_a7\B.0\board.xml'
'C:\Data\usrBrds\genesys2\H\board.xml'
```

または、Simulink® モデルを開き、モデルの System Generator トークンをダブルクリックしても、System Generator で使用可能なパートナー ボードおよびカスタム ボードを確認できます。追加されたボードは System Generator トークンのプロパティ ダイアログ ボックスの [Board] にリストされます。

図 7: ボードの選択



ボード リポジトリにボードを追加するには、startup.m ファイルで
`xilinx.environment.setBoardFileRepos` 行を変更し、新しいボード インターフェイス ファイル
(board.xml) のディレクトリを指定します。ボード インターフェイス ファイルが
`xilinx.environment.setBoardFileRepos` で既に指定されているフォルダーのサブディレクトリにある場合、
その新しいボード ファイルは次回 System Generator を開いたときに使用可能になり、startup.m ファイルを変更する
必要はありません。

System Generator を使用したハードウェア設計

System Generator は、FPGA ハードウェアを設計するためのシステム レベルのモデリング ツールです。Simulink® をさまざまな面で拡張しており、ハードウェア設計に適したモデリング環境を提供します。デザインが高抽象度で示され、ボタンをクリックするだけで FPGA にコンパイルできます。また、低抽象度の FPGA リソースにもアクセスできるので、効率的な FPGA デザインを構築できます。

System Generator を使用したデザイン フロー	System Generator を使用したデザイン フローを説明します。
System Generator でのシステム レベルのモデリング	柔軟かつ高位のシステム モデリング環境から、特定のデバイス用のハードウェア デザインを直接インプリメントする System Generator の機能を説明します。
自動コード生成	System Generator デザインの自動コード生成について説明します。
MATLAB の FPGA へのコンパイル	MATLAB プログラム言語のサブセットを使用した、ステート マシンおよび算術演算の記述方法を説明します。記述した関数は、System Generator のブロックに含め、等価の HDL に自動的にコンパイルできます。
System Generator デザインの大型システムへのインポート	System Generator デザインから VHDL ネットリストを生成して合成し、大型デザインに組み込む方法を説明します。また、System Generator で作成した VHDL をシステム全体のシミュレーション モデルに組み込む方法も説明します。
コンフィギュラブル サブシステムと System Generator	System Generator でのコンフィギュラブル サブシステムの使用方法を説明します。コンフィギュラブル サブシステムの定義、ブロックの削除と追加、コンフィギュラブル サブシステムを使用したコンパイル結果の System Generator デザインへのインポートなどのタスクを説明します。
FPGA デザインのパフォーマンスを向上するためのヒント	FPGA に効率的で高パフォーマンスのデザインをインプリメントするため、System Generator で推奨される設計手法を示します。
FDATool を使用したデジタル フィルター アプリケーション	FDATool ブロックを使用して、FIR フィルターを指定、インプリメント、およびシミュレーションする例を示します。
複数の独立クロックのハードウェア デザイン	デザインをサブシステム ブロックのグループに分割し、各サブシステムにほかのサブシステムのサイクル周期から独立した共通サイクル周期を使用できます。
AXI インターフェイス	AMBA AXI4 の概要と、System Generator で AMBA AXI4 を使用する場合を説明します。
AXI4-Lite インターフェイスの生成	System Generator モジュール用に標準 AXI4-Lite インターフェイスを作成し、IP インテグレーターを使用して大型デザインに含めるためにそのモジュールを Vivado® IP カタログにエクスポートする System Generator の機能を説明します。
System Generator でのプラットフォーム ベースのアクセラレータの調整	Vivado IP インテグレーターで開発されたプラットフォーム フレームワークの一部であるアクセラレータを、System Generator で開発する方法を説明します。

System Generator を使用したデザイン フロー

System Generator はさまざまな状況で役立ちます。デザインをハードウェアに変換せずにアルゴリズムを検討する場合、System Generator デザインを大型デザインの一部として使用する場合、または大型デザインの一部としてではなく、System Generator デザインそのものを FPGA ハードウェアで使用する場合などが考えられます。ここではこれら 3 つの場合について説明します。

アルゴリズムの検討

System Generator は、アルゴリズムの検討、デザインのプロトタイプ作成、モデル解析において特に便利なツールです。System Generator を使用してアルゴリズムを検討することにより、デザインで発生する可能性のある問題を調査したり、ハードウェアへのインプリメンテーションにおけるコストやパフォーマンスを見積もることができます。これらの作業は準備作業なので、デザインをハードウェアに変換する必要はほとんどありません。

この時点では、細かい点やインプリメンテーションの詳細を考慮せずに、デザインの主な部分を組み立てます。Simulink ブロックと MATLAB の M コードを使用して、シミュレーションおよび結果の解析用にスティミュラスを供給します。リソース見積もりにより、ハードウェアにインプリメントしたデザインのコストを見積もることができます。ハードウェア生成を使用したテストにより、達成可能なハードウェア スピードが示されます。

うまくいきそうなアプローチを見極めることができれば、今度はデザインの詳細を固めていきます。System Generator では、デザインを段階的に調整でき、デザインの一部はハードウェアにインプリメントできるように準備しながら、ほかの部分は高抽象度の高位記述のままにしておくことができます。特に、System Generator のハードウェア協調シミュレーション機能は、デザインを部分ごとに調整していくのに特に有益です。

大型デザインの一部としてのインプリメント

System Generator は、大型デザインの一部をインプリメントするのによく使用されます。たとえば、System Generator はデータパスおよび制御をインプリメントするには適していますが、厳しいタイミング要件を持つ高度な外部インターフェイスにはあまり適していません。この場合は、System Generator を使用してデザインの一部をインプリメントし、ほかの部分を別のツールでインプリメントして、これらを組み合わせて 1 つの大型デザインを構築するのが良いでしょう。

このフローでは、デザイン全体を表す HDL ラッパーを作成し、System Generator の部分をコンポーネントとして使用するのが一般的な方法です。System Generator 以外のツールで作成した部分も、コンポーネントとしてラッパーに含めるか、ラッパーに直接インスタンス化できます。

完成したデザインのインプリメント

デザインに必要なものをすべて System Generator だけで構築できる場合もあります。この場合、System Generator ブロックのパラメーター ダイアログ ボックスで [Generate] ボタンをクリックするだけで、デザインを HDL に変換し、ダウンストリーム ツールを使用して HDL の処理に必要なファイルを生成できます。生成されるファイルは、次のとおりです。

- デザインをインプリメントする HDL ファイル。
- HDL テストベンチ。このテストベンチを使用すると、Simulink シミュレーションの結果をロジック シミュレータの結果と比較できます。
- System Generator HDL を Vivado IDE プロジェクトとして使用できるようにするファイル。

System Generator で生成されるファイルの詳細は、[コンパイル結果](#) を参照してください。

DSP 設計に関する注意事項

System Generator は、Simulink を拡張してハードウェア設計を可能にしたもので、高抽象度の記述を FPGA に自動的にコンパイルできます。演算の抽象記述は Simulink に適していますが (離散時間/空間動的システム シミュレーション)、System Generator では FPGA の機能にもアクセスできます。

並列処理やパイプライン処理の活用方法など、ハードウェアでの実現方法に関する理解が深まれば、より良いインプリメンテーションを得ることができます。IP コアを使用すると、FFT などの複雑なファンクションを含む効率的な FPGA デザインを作成できます。また、System Generator では、アプリケーションをより正確に表すようにモデルを調整することも可能です。

System Generator の資料のさまざまなセクションで、ハードウェア機能を活用するためにシステム パラメーターを使用する方法を説明しています。

ハードウェア設計に関する注意事項

System Generator は、ハードウェア記述言語 (HDL) ベースのデザインを置き換えるものではありませんが、デザインの重要な部分だけに集中することを可能にします。たとえば、ほとんどの DSP プログラミング ツールは、アセンブラーでのみプログラムするのではなく、C 言語のような高水準言語でプログラムを始め、パフォーマンス要件を満たすのに必要な箇所だけアセンブリ コードを記述します。

一般的には、内部ハードウェア クロックを (DDR や位相クロックを使用するなどして) 制御する必要のある部分は、HDL を使用してインプリメントするのが適切です。比較的重要度の低い部分は System Generator でインプリメントし、その後 HDL 部分と System Generator 部分を接続します。通常、外部インターフェイスを除き、信号処理システムにはこのレベルの制御は必要ありません。System Generator には、HDL コード部分をデザインにインポートする機能 (第 6 章: HDL モジュールのインポートを参照) が含まれています。

System Generator には、テスト ベクターを含む HDL テストベンチを自動生成する機能もあります。この機能については、HDL テストベンチを参照してください。

第 5 章: ハードウェア協調シミュレーションの使用で説明されているハードウェア協調シミュレーション インターフェイスを使用すると、Simulink で制御してデザインをハードウェアで実行でき、MATLAB と Simulink のデータ解析および可視化の機能を最大限に活用できます。

System Generator でのシステム レベルのモデリング

System Generator では、柔軟で高位のシステム モデリング環境でデバイス特定のハードウェア デザインを構築できます。System Generator デザインでは、信号は単なるビットではなく、符号付きまたは符号なしの固定小数点値にすることができ、デザインに変更を加えると、信号型も自動的に適切な型に変換されます。ブロックは単なるハードウェアの代用ではなく、その周辺の状況に応じて出力結果および生成されるハードウェアが自動調整されます。

System Generator では、さまざまな要素からデザインを構築できます。データフロー モデル、ハードウェア記述言語 (VHDL、Verilog)、および MATLAB プログラム言語による関数を併用でき、一緒にシミュレーションしたり、ハードウェアに合成したりできます。System Generator のシミュレーション結果は、ビット精度およびサイクル精度であり、ハードウェアでの結果と厳密に一致します。System Generator のシミュレーションは、従来の HDL シミュレータでのシミュレーションよりも非常に高速で、結果も解析しやすくなっています。

System Generator ブロックセット	System Generator ブロックのライブラリでの分類、ブロックのパラメーター設定方法および使用方法を説明します。
--------------------------	---

モデルをすばやく作成して解析するザイリンクス コマンド	Simulink ポップアップ メニューに追加されたザイリンクス コマンドを使用して、System Generator モデルを簡単に作成および解析します。
信号型	System Generator で使用されるデータ型と、ツールで自動的にデータ型を割り当てる方法を説明します。
ビット精度およびサイクル精度のモデリング	System Generator モデルの Simulink ベースのシミュレーションと、ハードウェアでの動作との関係を説明します。
タイミングとクロック	クロックのハードウェアへのインプリメント方法、および System Generator でのクロック インプリメンテーション制御方法を説明します。System Generator でマルチレート Simulink モデルがどのようにクロック同期ハードウェアに変換されるかを説明します。
同期化のメカニズム	高位 System Generator デザインでデータパス エLEMENT間のデータフローを同期化する方法と、制御パス ファンクションをインプリメントする方法を説明します。
ブロック マスクとパラメーターの伝搬	Simulink でパラメーター指定システムおよびサブシステムを作成する方法を説明します。

System Generator ブロックセット

Simulink® ブロックセットは、Simulink ブロック エディターで接続でき、ダイナミックなシステムのファンクション モデルを作成するためのブロックのライブラリです。システム モデリングでは、System Generator ブロックセットはほかの Simulink ブロックセットと同様に使用できます。ブロックは、数値演算、論理、メモリ、DSP 関数を表したものであり、高度な信号処理システムなどのシステムを構築するために使用できます。また、System Generator コード生成ソフトウェアだけでなく、FDATool、ModelSim などのソフトウェア ツールへのインターフェイスを提供するブロックもあります。

System Generator ブロックは、ビット精度およびサイクル精度です。ビット精度ブロックは、ハードウェアで生成される対応値に一致した値を Simulink で生成し、サイクル精度ブロックは対応する時間に対応する値を生成します。

ザイリンクス ブロックセット

ザイリンクス ブロックセット (Xilinx Blockset) は、基本的な System Generator ブロックを含むライブラリを集めたものです。デバイス別のハードウェアにアクセスする低位ブロックと、信号処理や高度な通信アルゴリズムなどをインプリメントする高位ブロックがあります。Gateway In/Gateway Out ブロックなど幅広く応用可能なブロックは、複数のライブラリに含まれています。Index ライブラリには、すべてのブロックが含まれています。次に、これらのライブラリについて説明します。

ライブラリ	説明
AXI4	AXI4 仕様に準拠するインターフェイスを使用したブロック
Basic Elements	デジタル ロジックの標準基本ブロック
Communication	デジタル通信システムでよく使用される前方エラー訂正ブロックおよびモジュレータ ブロック
Control Logic	制御回路およびステート マシン用のブロック
Data Types	データ型を変換するブロック (Gateway ブロックを含む)
DSP	DSP (デジタル信号処理) ブロック
Floating-Point	浮動小数点型をサポートするブロック
Index	ザイリンクス ブロックセットのすべてのブロック
Math	数学関数をインプリメントするブロック
Memory	メモリをインプリメントするブロックおよびメモリにアクセスするブロック

ライブラリ	説明
Tools	コード生成 (System Generator トークン)、リソース見積もり、HDL 協調シミュレーションなどを実行するユーティリティ ブロック

ザイリンクス リファレンス ブロックセット

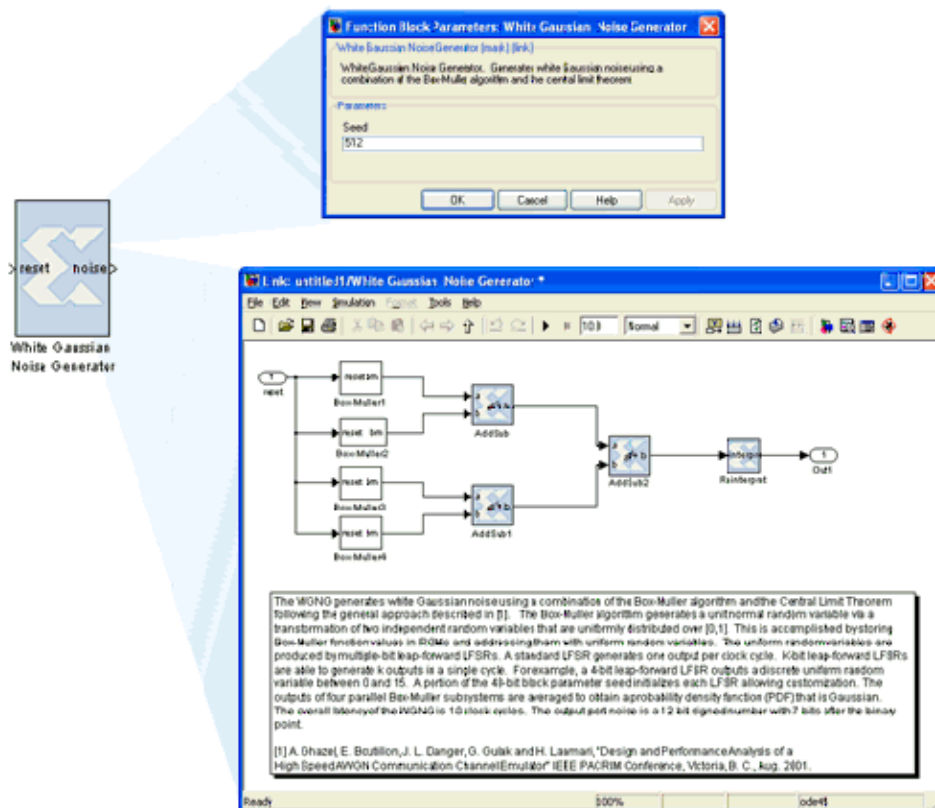
ザイリンクス リファレンス ブロックセット (Xilinx Reference Blockset) には、さまざまなファンクションをインプリメントする複合 System Generator ブロックが含まれます。このブロックセットのブロックは、ファンクション別にライブラリに分類されています。次に、これらのライブラリについて説明します。

ライブラリ	説明
Communication	デジタル通信システムでよく使用されるブロック
Control Logic	制御回路およびステート マシン用のブロック
DSP	DSP (デジタル信号処理) ブロック
Imaging	イメージ処理ブロック
Math	数学関数をインプリメントするブロック

このブロックセットの各ブロックは複合ブロックであり、ブロックを設定するパラメーターを使用してマスク サブシステムとしてインプリメントされます。

リファレンス ブロックセット ライブラリに含まれるブロックは、そのまま使用することもできますが、同じような特性を持つデザインを作成する際の開始点としても使用できます。各リファレンス ブロックには、インプリメンテーションの説明およびハードウェア リソース要件があります。

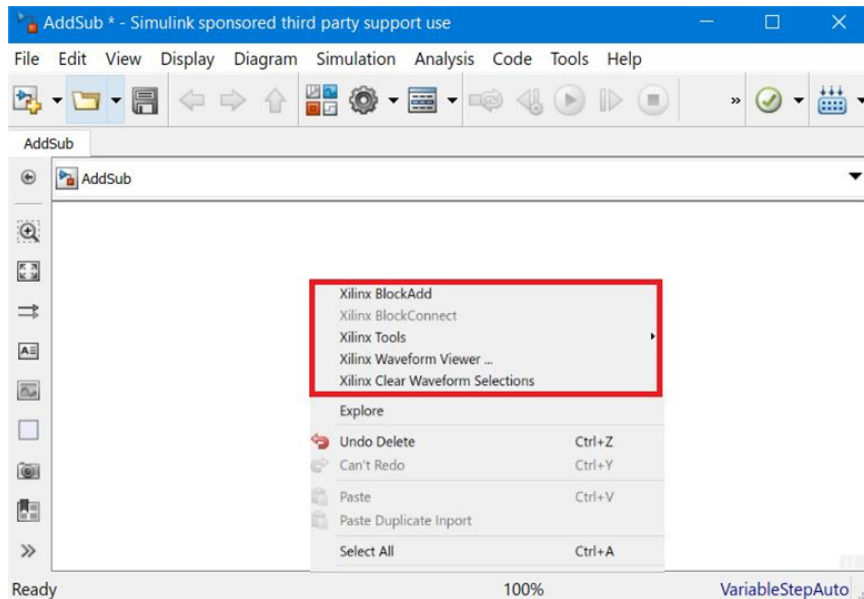
図 8: リファレンス ブロックセット ライブラリ



モデルをすばやく作成して解析するザイリンクス コマンド

System Generator デザインをすばやく作成および解析できるようにするため、ザイリンクスでは Simulink® の右クリックメニューにグラフィック コマンドを追加しています。これらのコマンドには、次の図に示すように、Simulink モデルのキャンバスを右クリックし、適切なザイリンクス コマンドをクリックするとアクセスできます。

図 9: ザイリンクス コマンド



これらのザイリンクス コマンドの使用方法的の詳細は、各コマンドのトピックを参照してください。

信号型

ハードウェアのビット精度シミュレーションを実行するには、System Generator ブロックがブール値、浮動小数点値、および任意精度の固定小数点値で動作する必要がありますが、Simulink® の基本的なスカラー信号型は、倍精度浮動小数点です。ザイリンクス ブロックとザイリンクス以外のブロックの間の接続には、Gateway ブロックを使用します。Gateway In ブロックは倍精度の信号をザイリンクス信号に変換し、Gateway Out ブロックはザイリンクス信号を倍精度の信号に変換します。Simulink® の連続時間信号は、Gateway In ブロックでサンプリングします。

ほとんどのザイリンクス ブロックはポリモーフィックで、入力型に応じて適切な出力型を推測できます。ブロックのパラメーター ダイアログ ボックスで完全精度を指定した場合、System Generator で精度が失われないような出力型が選択されます。符号拡張および 0 パディングは、必要に応じて自動的に実行されます。ユーザー指定の精度も使用できます。この場合、ブロックの出力型と、量子化およびオーバーフローの処理方法を指定できます。量子化のオプションには、正または負の無限大への不偏丸め (符号によって異なる) または切り捨てがあります。オーバーフローのオプションには、正または負の最大値を使用するか、切り捨てるか、オーバーフローをエラーとしてレポートするかがあります。

注記: System Generator のデータ型は、Simulink で [Display] → [Signals & Ports] → [Port Data Types] をクリックすると表示されます。データ型を表示すると、モデルの精度を簡単に判断できます。たとえば、ポートのデータ型が [Fix_11_9] の場合は、信号は小数点以下のビットが 9 桁の 2 の補数符号付き 11 ビット値であり、[Ufix_5_3] の場合は小数点以下のビットが 3 桁の符号なし 5 ビット値です。

Simulink モデルの System Generator 部分では、すべての信号をサンプリングする必要があります。サンプル時間は、Simulink の伝搬ルールにより自動的に設定されるようにするか、ブロックのカスタマイズ ダイアログ ボックスで設定できます。フィードバック ループがあると、System Generator でサンプル周期および信号型を推論できない場合もあり、その場合はエラー メッセージが表示されます。フィードバック ループには Assert ブロックを挿入して、この問題を回避する必要があります。ループ内のすべてのポイントに Assert ブロックを追加する必要はありません。通常、ループの 1 箇所に追加するだけで十分です。

注記: Simulink では、ブロックと信号を動作レート別に色分けして表示できます (Simulink メニューから [Display] → [Sample Time] → [Colors] をクリック)。これは、マルチレート デザインを解析するのに便利です。

浮動小数点型

Floating-Point ライブラリの System Generator ブロックでは、浮動小数点型がサポートされます。

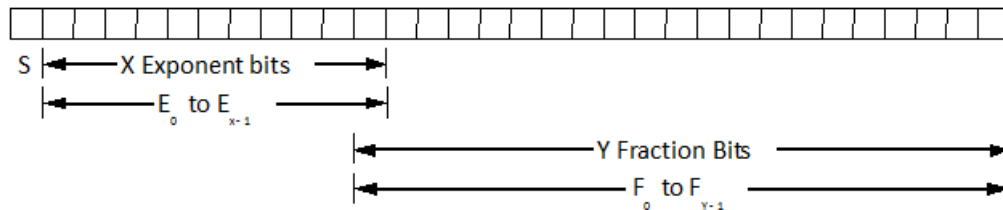
System Generator では、Floating-Point Operator v7.1 IP を使用して、加算/減算、乗算、比較、データ型変換などの演算がインプリメントされます。

浮動小数点型のサポートは、浮動小数点演算用の IEEE-754 規格に準拠します。サポートされる System Generator では、デザイン入力、データ型表示、データ レート、データ型伝搬 (RTP) に対して単精度、倍精度、カスタム精度の浮動小数点型がサポートされます。

浮動小数点型の IEEE-754 規格

次に示すように、浮動小数点型は 1 つの符号ビット (S)、X 指数部ビット、Y 仮数部ビットを使用して記述できます。符号ビットは、常に最上位ビット (MSB) です。

図 10: 浮動小数点データ



IEEE-754 規格によれば、浮動小数点値は正規化された形で表現および格納されます。この形式では、指数値 E がバイアス/正規化された値です。正規化された指数 E は、実際の指数部の値とバイアスの合計と同じです。正規化された形式では、仮数部の値を格納するのに Y-1 ビットが使用されます。F0 仮数ビットは、常に隠しビットで、値は 1 であるとみなされます。

S は値の符号を表します。S が 0 の場合、値は正の浮動小数点になり、それ以外は負の浮動小数点になります。その後の X ビットは、正規化された指数部の値 E を格納するのに使用され、最後の Y-1 ビットは正規化された仮数部の値を格納するのに使用されます。

指数部バイアスは、指数部のビット幅 (桁数) から次の式を使用して計算されます。

$$\text{Exponent_bias} = 2^{(X-1)} - 1$$

X は指数部のビット幅です。

IEEE 規格では、単精度浮動小数点データは 32 ビットを使用して記述されます。正規化された指数部および仮数部は、それぞれ 8 ビットと 24 ビットです。単精度の指数部バイアスは 127 です。同様に、倍精度浮動小数点データは合計 64 ビットを使用して記述され、指数部のビット幅は 11、仮数部のビット幅は 53 になります。倍精度の指数部バイアスは 1023 です。

正規化された浮動小数点の値は次のように記述されます。

$$\text{正規化された浮動小数点値} = (-1)^S \times F_0.F_1F_2 \dots F_{Y-2}F_{Y-1} \times (2)^E$$

実際の指数部の値 (E_{actual}) は E - Exponent_bias です。隠しビット F0 の値と E_{actual} 値を 1 とした場合、浮動小数点の数値は次のように計算できます。

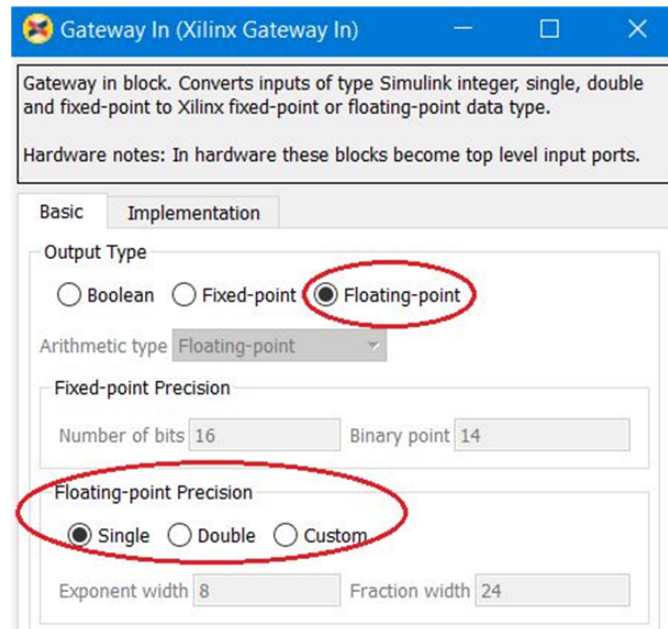
$$\text{FP_Value} = (-1)^S \times 1.F_1F_2 \dots F_{Y-2}F_{Y-1} \times (2)^{(E_{\text{actual}})}$$

System Generator での浮動小数点データ表記

System Generator の Gateway In ブロックでは、以前はブール型および固定小数点型のみがサポートされていたが、次に示すように、Gateway In ブロックの GUI と基本となるマスク パラメーターで浮動小数点型がサポートされるようになりました。浮動小数点型を指定した後、[Single]、[Double]、または [Custom] のいずれかの精度を選択できます。

たとえば、指数部の幅を 9、仮数部の幅を 31 に指定した場合、浮動小数点データ値は合計 40 ビットで、MSB が符号表記に、その後の 9 ビットがバイアス指数部に、残りの 30 ビットが仮数部を格納するために使用されます。

図 11: 浮動小数点の精度

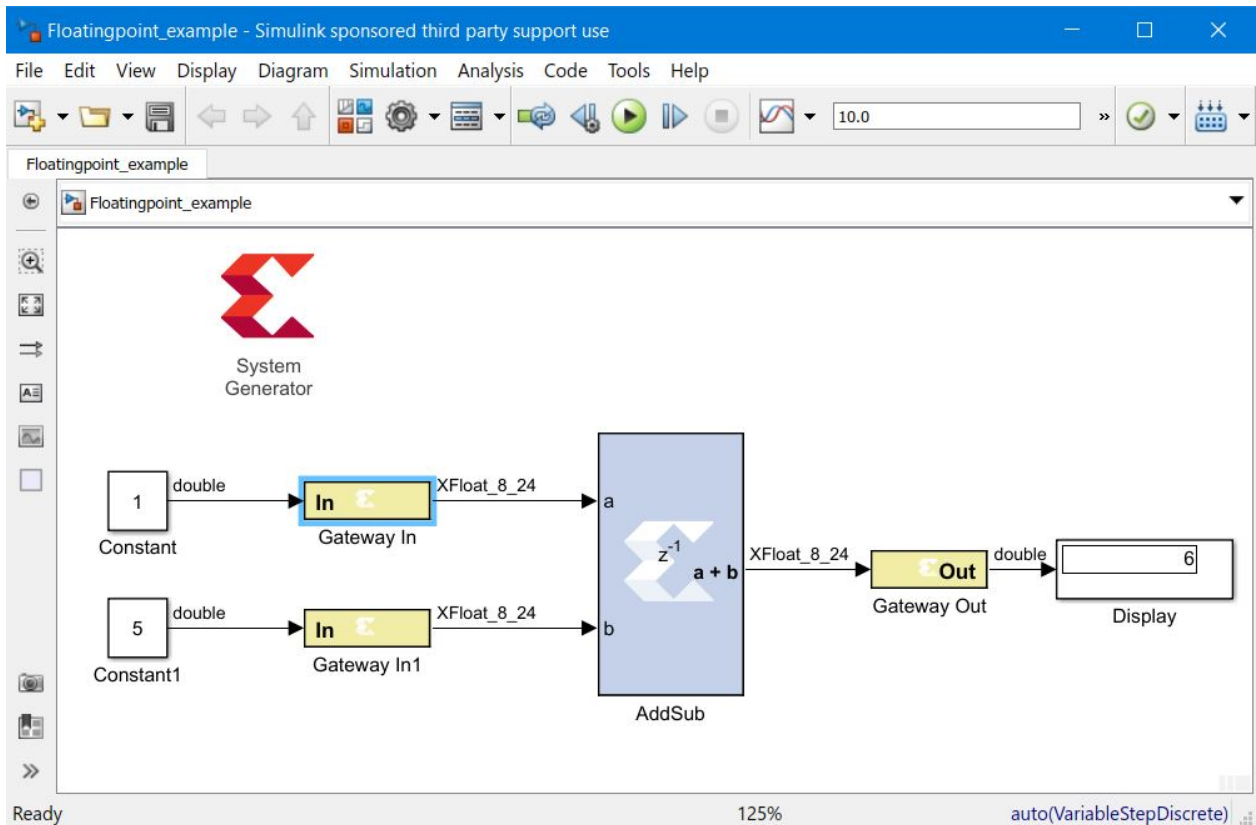


IEEE-754 規格に準拠するため、[Single] 精度を選択すると、ビット幅は合計 32 ビット (指数部 8 ビット、仮数 24 ビット) になります。[Double] 精度を選択すると、ビット幅は合計 64 ビット (指数部 11 ビット、仮数部 53 ビット) になります。[Custom] 精度を選択すると、[Exponent width] および [Fraction width] フィールドの値を指定できるようになります (デフォルトは 8 ビットと 24 ビット)。[Custom] 精度のデータの合計ビット幅は、指数部ビットと仮数部ビットを加算したものになります。[Single] および [Double] 精度型の仮数部のビット幅と同様、[Custom] 精度型の仮数部のビット幅にも隠しビット F0 を含める必要があります。

出力信号のデータ型の表示

レートおよびデータ型が正しく伝搬されると、次の図に示すように、各 System Generator ブロックの出力に浮動小数点型が表示されます。この図のように信号のデータ型を表示するには、[Display] → [Signals & Ports] → [Port Data Types] をクリックします。

図 12: 浮動小数点型



浮動小数点型は、XFloat-<exponent_bit_width>-<fraction_bit_width> という形式で表示されます。単精度 (Single) 型は XFloat_8_24、倍精度 (Double) 型は XFloat_11_53 という文字列を使用して表示されます。

カスタム精度 (Custom) 型で指数部のビット幅を 9、仮数部のビット幅を 31 に指定した場合は、XFloat_9_31 のように表示されます。浮動小数点データ値を格納するためには、40 ビットが使用されます。浮動小数点データは基本形式で格納されるので、仮数部の値は 30 ビットに格納されます。

System Generator では、固定小数点型は XFix-<total_data_width>-<binary_point_width> という形式で表示されます。たとえば、データ幅 40 および 2 進小数点の幅 31 の固定小数点型は、XFix_40_31 と表示されます。

固定小数点型で仮数部の値を格納するのに使用された実際のビット数は、浮動小数点型で使用するものとは異なります。上記の例では、固定小数点型の仮数部ビットを格納するために 31 ビットすべてが使用されます。

System Generator では、指数部のビット幅と仮数部のビット幅を使用して、Floating-Point Operator コアのインスタンスが設定および生成されます。

レートおよびデータ型の伝搬

浮動小数点データをサポートする System Generator でのデータ レートおよびデータ型の伝搬中は、次のデザイン ルールが検証されます。次の違反のいずれかが検出されると、該当するエラー メッセージが表示されます。

1. 浮動小数点データを転送する信号が浮動小数点型をサポートしない System Generator ブロックのポートに接続されている場合。

2. System Generator ブロックのデータ入力 (該当する場合は A および B の両方のデータ入力) とデータ出力が、同じ浮動小数点型でない場合。ブロックの 2 つの入力間と、ブロックの入力と出力間で、DRC チェックが実行されます。

[Custom] 精度の浮動小数点型を指定した場合、2 つのポートの指数部のビット幅および仮数部のビット幅が比較され、同じデータ型であるかどうか判断されます。

注記: Convert および Relational ブロックは、このチェックの対象外です。Convert ブロックでは、2 つの異なる浮動小数点型の間でのデータ型変換がサポートされます。Relational ブロック出力は、常に比較演算に対して true か false の結果を出力するので、常にブール型です。

3. データ入力固定小数点型で、データ出力が浮動小数点の場合、またはその逆の場合。

注記: Convert および Relational ブロックは、このチェックの対象外です。Convert ブロックでは、固定小数点から浮動小数点、浮動小数点から固定小数点への変換がサポートされます。Relational ブロック出力は、常に比較演算に対して true か false の結果を出力するので、常にブール型です。

4. 浮動小数点型をサポートするブロックの出力のデータ型にユーザー定義の精度が選択されている場合。たとえば、AddSub、Mult、CMult、および MUX などのブロックでは、データ入力が浮動小数点型であれば、サポートされる出力精度は Full のみです。
5. 浮動小数点型の演算で、Carry In ポートまたは Carry Out ポートが AddSub ブロックに使用されている場合。
6. Floating-Point Operator IP コアで、その IP 用に定義されている DRC ルールのエラー メッセージが表示された場合。

AXI 信号グループ

AXI4 ライブラリに含まれる System Generator ブロックには、AXI4 仕様に準拠するインターフェイスが含まれます。AXI4 インターフェイスを含むブロックでは、特定の AXI4 インターフェイスに関するポートがグループ化され、同じ色で表示されます。これにより、同じインターフェイスに属するデータおよび制御信号が見分けやすくなります。同様の AXI4 ポートがグループ化されていることにより、Simulink Bus Creator および Simulink Bus Selector ブロックを使用して信号グループを接続することも可能です。AXI4 の詳細は、[AXI インターフェイス](#) を参照してください。AMBA AXI4 仕様の詳細は、ザイリックス ウェブサイトの [AMBA AXI4 インターフェイス プロトコル](#) ページにあるザイリックス AMBA AXI4 の資料を参照してください。

ビット精度およびサイクル精度のモデリング

System Generator でのシミュレーションは、ビット精度またはサイクル精度で実行されます。ビット精度のシミュレーションとは、System Generator ブロックと System Generator 以外のブロックの境界で、シミュレーションで生成された値がハードウェアで生成された対応値とビット単位で同一であるということです。サイクル精度のシミュレーションとは、System Generator ブロックと System Generator 以外のブロックの境界で、対応する値が対応する時間に生成されるということです。デザインの境界は、System Generator の Gateway ブロックが配置されている箇所です。デザインがハードウェアに変換されると、Gateway In ブロックは最上位入力ポート、Gateway Out ブロックは最上位出力ポートになります。

タイミングとクロック

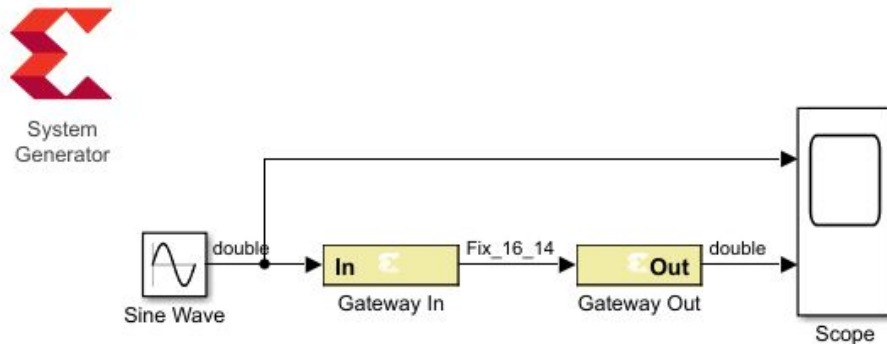
離散時間システム

System Generator のデザインは離散時間システムなので、信号とその信号を生成するブロックにはサンプル レートがあります。ブロックのサンプル レートは、ブロックのステートがアップデートされる頻度を決定します。System Generator では、ほとんどのサンプル レートが自動的に設定されますが、サンプル レートを明示的または暗示的に設定する必要のあるブロックもあります。

注記: Simulink 離散時間システムとサンプル時間の詳細は、MathWorks 社の資料『Using Simulink』を参照してください。

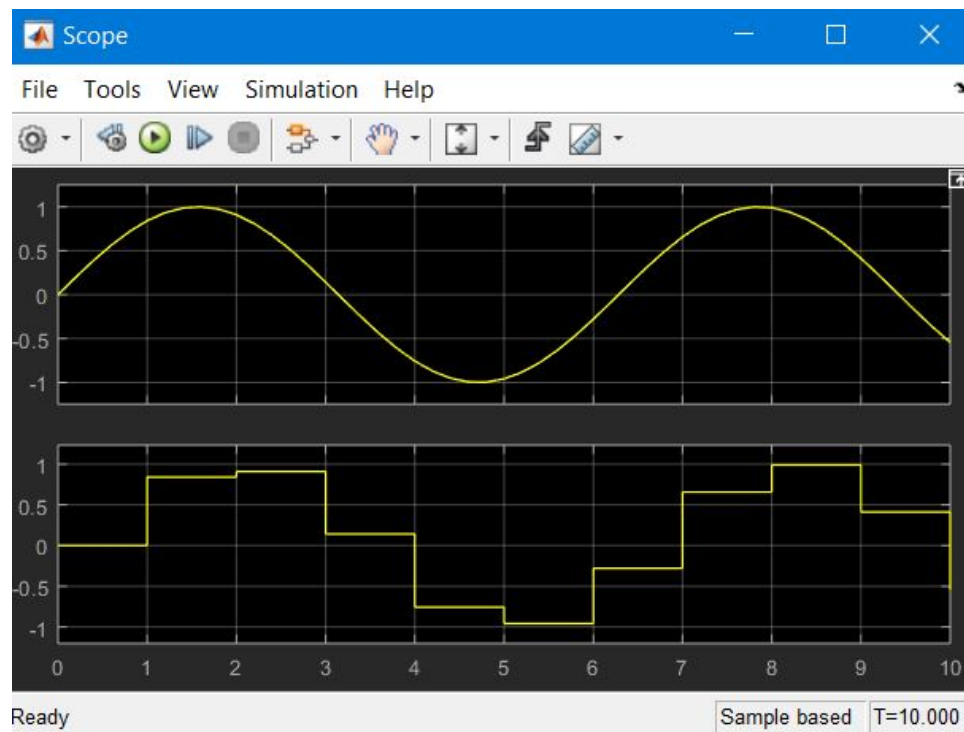
簡単な System Generator モデルで、離散時間システムの動作を示します。次の図に示すモデルがあるとします。Simulink ソース (Sine Wave) で Gateway In ブロックを駆動し、Gateway Out ブロックで Simulink シンク (Scope) を駆動しています。

図 13: 離散時間システム



Gateway In ブロックのサンプル周期は、1 秒に設定されています。Gateway Out ブロックは、ザイリンクス固定小数点信号を Simulink の Scope で解析できるように倍精度に変換しますが、サンプル レートは変更しません。Scope の出力は、サンプル レートが変更されていない、サンプリングされたサイン波となります。

図 14: Scope 出力



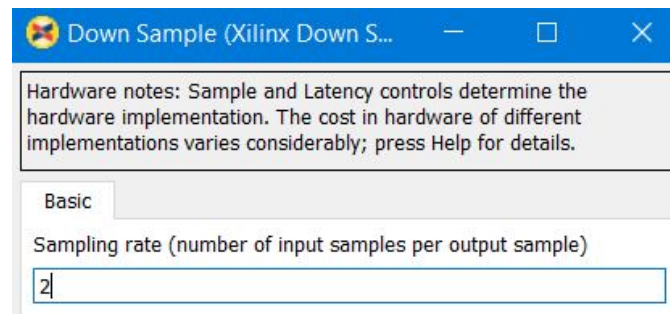
マルチレート モデル

System Generator では、複数のサンプル レートで動作する信号を含むマルチレート デザインがサポートされています。マルチレート モデルは、System Generator で自動的にハードウェアにコンパイルされます。これにより、マルチレート デザインが Simulink に適した直接的な方法でインプリメントされます。

レート変換ブロック

System Generator には、サンプル レートを変換するブロックが含まれています。最も基本的なレート変換ブロックは、Up Sample と Down Sample ブロックです。これらのブロックは、次の図に示すように、パラメーター ダイアログ ボックスで指定した固定の値を乗算することにより、レートを変換します。

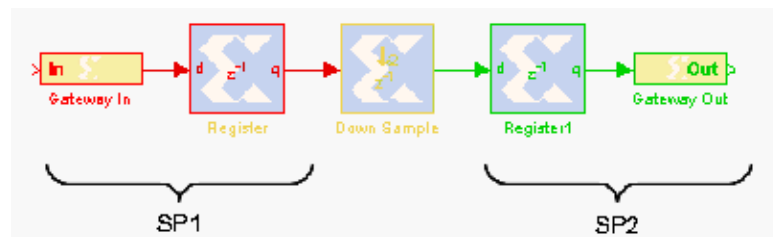
図 15: レートの変換



Parallel To Serial や Serial To Parallel などのその他のブロックは、ブロックのパラメーターで指定された方法でレートを変換します。

次のような単純なマルチレート システムがあるとします。このモデルでは、SP1 と SP2 の 2 つのサンプル周期が使用されます。サンプル周期 SP1 は、Gateway In のパラメーター ダイアログ ボックスで指定します。Down Sample ブロックによりモデルのレートが変更され、SP1 の 1/2 である新しいレート SP2 が作成されます。

図 16: マルチレート例



ハードウェア オーバーサンプリング

一部の System Generator ブロックはオーバーサンプリングされ、ブロックのデータ レートより高速のレートで内部処理が実行されます。ハードウェアでは、これはデータ サンプルを処理するのに複数のクロック サイクルが必要であることを意味します。Simulink では、このようなブロックはサンプル レートに計測可能な変化はありません。

Simulink ではオーバーサンプリングされるブロックでサンプル レートが変化することはありませんが、System Generator では、ハードウェア インプリメンテーション用のクロック ロジックを生成する際に、サンプル レートと共に内部ブロック レートも考慮されます。そのため、System Generator トークンのパラメーター ダイアログ ボックスで [Simulink system period] を指定する際に、オーバーサンプリングされるブロックの内部処理レートを考慮する必要があります。

非同期クロック

System Generator は、1 つのクロックに同期するハードウェアの設計に適していますが、場合によっては、複数のクロックを使用するシステムの設計にも使用できます。この場合、デザインをクロック ドメインに分割し、ドメイン間での情報転送をデュアル ポート メモリおよび FIFO で制御します。このセクションでは、System Generator のクロック同期について説明します。この内容は、1 つのクロックのデザインおよび複数クロックのデザインの両方に関係します。

同期クロック

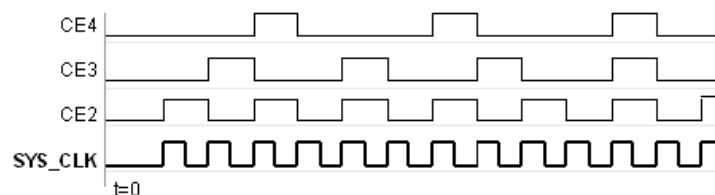
デフォルトでは、System Generator により同期クロックでデザインが作成され、複数のレートはクロック イネーブルを使用して生成されます。System Generator でモデルをハードウェアにコンパイルすると、ハードウェアの対応する部分が適切なレートで動作するように、System Generator でデザインのサンプル レート情報が保持されます。System Generator は、クロックとクロック イネーブル (1 つのレートに 1 つのイネーブル) を組み合わせて、関連するレートをハードウェアに生成します。各クロック イネーブルの周期は、システム クロックの周期の整数倍です。

Simulink 内では、System Generator デザインの信号としてクロックおよびクロック イネーブルは必要ありません。System Generator でデザインをハードウェアにコンパイルする際、デザインのサンプル レートから必要なクロック イネーブルが推論されます。具体的には、System Generator トークンの 2 つのユーザー指定の値である [Simulink system period] と [FPGA clock period] が使用されます。これらの値は、Simulink シミュレーションの時間、および実際のハードウェア インプリメンテーションでの時間のスケーリング係数を定義します。Simulink システム周期は、モデルに含まれるサンプル周期の最大公約数 (gcd) にする必要があり、FPGA のクロック周期 (ns) はシステム クロックの周期です。Simulink システム周期を p 、FPGA システム クロック周期を c とすると、Simulink で kp かかる処理は、ハードウェアではシステム クロックの k サイクル分 (kc ナノ秒) になります。

たとえば、3 つの Simulink サンプル周期 2、3、4 を含むモデルがあるとします。これらのサンプル周期の gcd は 1 で、そのモデルの [Simulink system period] フィールドで指定する必要があります。[FPGA clock period] は 10 ns に設定されているとします。これらの情報から、ハードウェアでの対応するクロック イネーブルの周期を決定できます。

Simulink のサンプル周期 2、3、4 に対応するハードウェアでのクロック イネーブルを CE2、CE3、CE4 とします。各クロック イネーブルの周期とシステム クロック周期の関係は、対応する Simulink サンプル周期を Simulink システム周期で割ることにより求められます。この結果、CE2、CE3、CE4 の周期はそれぞれ 2、3、4 システム クロック周期になります。次の図に、これらのクロック イネーブル信号のタイミングを示します。

図 17: タイミング図



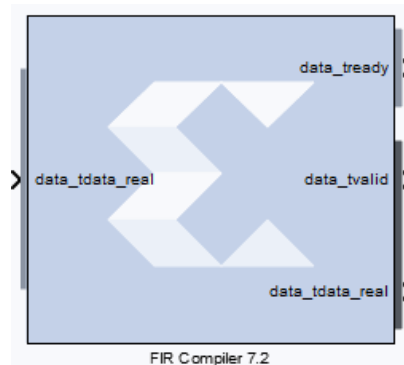
同期化のメカニズム

System Generator では、同期化のメカニズムは自動的に作成されません。設計者が明示的に作成する必要があります。

有効なポート

System Generator には、同期化に使用できるブロック (特に FIFO) が複数含まれています。複数のブロックでオプションの AXI 信号インターフェイスが提供されており、サンプルが有効なとき (TValid)、およびインターフェイスがデータ準備完了になったとき (TReady) を示します。IP の設定によっては、tvalid/tready ポートが確認できない場合があります。次の図に示すように、ブロックの各インターフェイスのポート群は色分けされています。インターフェイスのあるブロックはチェーン接続可能で、これで簡単にフロー制御できます。AXI インターフェイスのあるブロックには、FFT、FIR、DDS などがあります。

図 18: AXI インターフェイスを使用するブロック



不定のデータ

多くのハードウェア シミュレーション環境では、不定値があるのが一般的です。これらは、「ドントケア」または「X」と示されます。System Generator シミュレーションの値は、不定値である可能性があります。たとえば、デュアルポートメモリブロックでは、メモリの両方のポートで同じアドレスに同時にアクセスしようとすると、値が不定になります。ハードウェアでの実際の動作は、どちらのポートのクロックエッジが先に到着するかなどを決定するインプリメンテーションの詳細によって異なります。値が不定になることを許容すると、システム設計の柔軟性が増します。先ほどの例で、メモリで値が不定になっても、その後の処理がその値に依存していなければ、問題ありません。

HDL 協調シミュレーションによりシミュレーションに含まれる HDL モジュールは、一般的にデータサンプルが不定になる原因となります。System Generator で HDL 協調シミュレーション モジュールの入力に入力される不定値は、標準ロジックベクトル XXX...XX で表されます。

Gateway Out を駆動する不定値は、NaN (Not a Number) という値になります。Simulink の Scope では、NaN 値は表示されません。Gateway In を駆動する NaN も不定値になります。System Generator には、不定値を検出する Indeterminate Probe ブロックが含まれています。このブロックは、ハードウェアには変換されません。

System Generator では、演算信号が不定値になってもかまいませんが、ブール信号を不定値にすることはできません。シミュレーションでブール信号が不定値になる状況が発生した場合は、シミュレーションは中断され、エラーメッセージが表示されます。ザイリンクスブロックには、ブール信号のみを入力として使用可能な制御ポートが含まれているものが多数あります。これらのブロックでは、制御ポートのブール信号を不定値にすることはできません。

UFix_1_0 は、ブール信号と同等のデータ型ですが、不定値に関する上記の制限はありません。

ブロックマスクとパラメーターの伝搬

通常の Simulink ブロックに適用されるスコーピングルールとパラメーター伝搬ルールは、System Generator ブロックにも適用されます。つまり、ザイリンクスブロックセット内のブロックは、MATLAB の変数および論理式を使用してパラメーター指定できます。この機能により、MATLAB 言語の表現能力および計算能力を活用した高度なパラメーター指定デザインを作成できます。

ブロック マスク

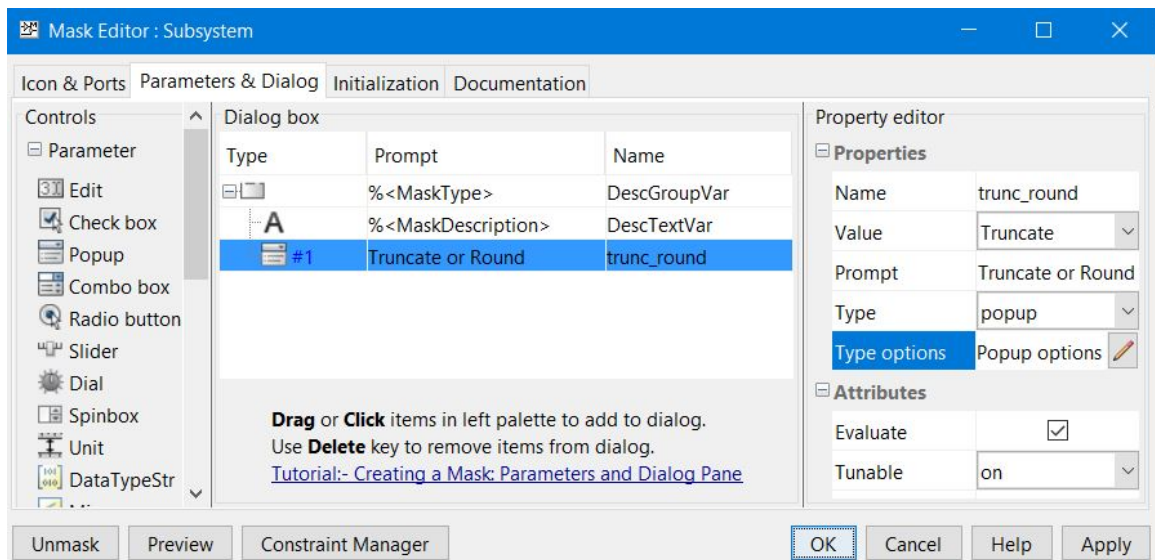
Simulink では、マスクと呼ばれるメカニズムでブロックのパラメーターを指定します。実際には、ブロックにマスク変数を割り当て、この変数の値をダイアログ ボックスで指定するか、マスク初期化コマンドで算出します。この変数は、マスク ワークスペースに保存されます。マスク ワークスペースは、マスクが適用されるブロックでのみ使用され、外部ブロックからアクセスすることはできません。

注記: マスクでグローバル変数および基本ワークスペースの変数にアクセスすることは可能です。基本ワークスペースの変数にアクセスするには、MATLAB の `evalin` 関数を使用します。MATLAB と Simulink の適用範囲規則の詳細は、MathWorks 社の資料『Using MATLAB』および『Using Simulink』を参照してください。

パラメーターの伝搬

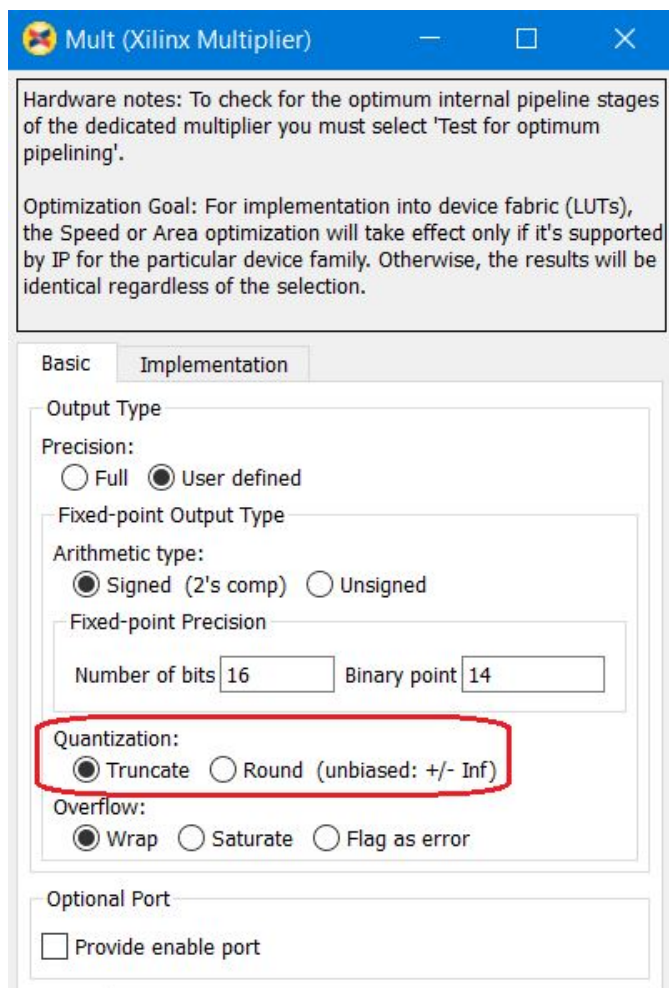
マスク サブシステム内のブロックに変数を渡す必要があることがよくあります。これにより、サブシステムのパラメーターによりブロックの設定を定義できます。この手法は、ザイリンクス ブロックセット内のブロックのパラメーターにも使用できます。たとえば、Mult ブロックと Accumulator ブロックで構成されるサブシステムを構築する場合、結果を切り捨てるか丸めるかを指定するパラメーターをサブシステム内に作成できます。次の図では、このパラメーターに `trunc_round` という名前が付けられています。

図 19: パラメーターの作成



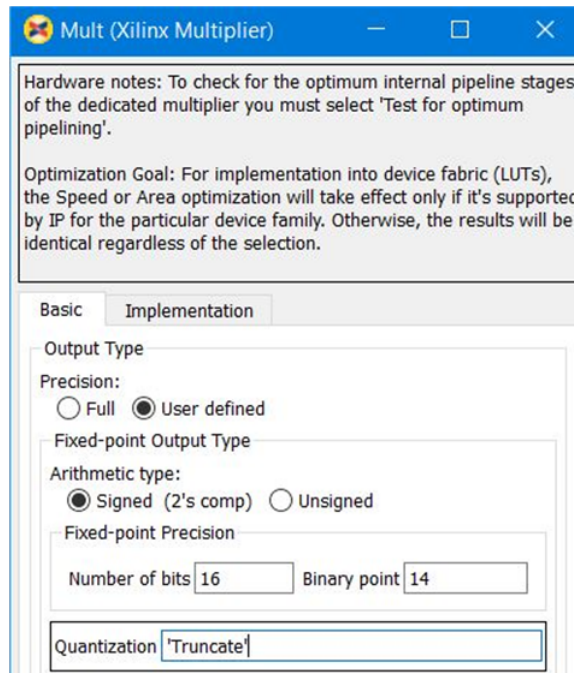
Mult ブロックと Accumulator ブロックのパラメーター ダイアログ ボックスには、[Truncate] (切り捨て) または [Round] (丸め) を選択するラジオ ボタンがあります。

図 20: パラメーターの変更



ラジオ ボタンではなくパラメーターを使用して選択する場合は、ラジオ ボタンを右クリックして [Define With Expression] をクリックすると、MATLAB の論理式を使用できます。次の例では、サブシステム マスクの trunc_round パラメーターを Mult ブロックと Accumulator ブロックの両方で使用して、サブシステムのマスク変数が両方に適用されるようにしています。

図 21: パラメーターの使用



自動コード生成

System Generator は、デザインを自動的に低級言語表現にコンパイルします。System Generator でのモデルのコンパイル方法は、System Generator トークンの設定によって異なります。ハードウェアの HDL 記述に加え、補助ファイルも生成されます。プロジェクト ファイルや制約ファイルなどのファイルはダウンストリーム ツールで使用され、VHDL テストベンチなどのファイルはデザインの検証に使用されます。

System Generator トークンを使用したコンパイルおよびシミュレーション	System Generator トークンを使用してデザインを低位 HDL にコンパイルする方法を説明します。
コンパイル結果	System Generator トークンで [HDL Netlist] を選択して [Generate] ボタンをクリックしたときに System Generator で生成される下位ファイルについて説明します。
Vivado プロジェクト	System Generator トークンで [HDL Netlist] または [IP Catalog] を選択して [Generate] ボタンをクリックしたときに System Generator で生成されるサンプル プロジェクトについて説明します。
HDL テストベンチ	System Generator で生成される VHDL テストベンチについて説明します。

System Generator トークンを使用したコンパイルおよびシミュレーション

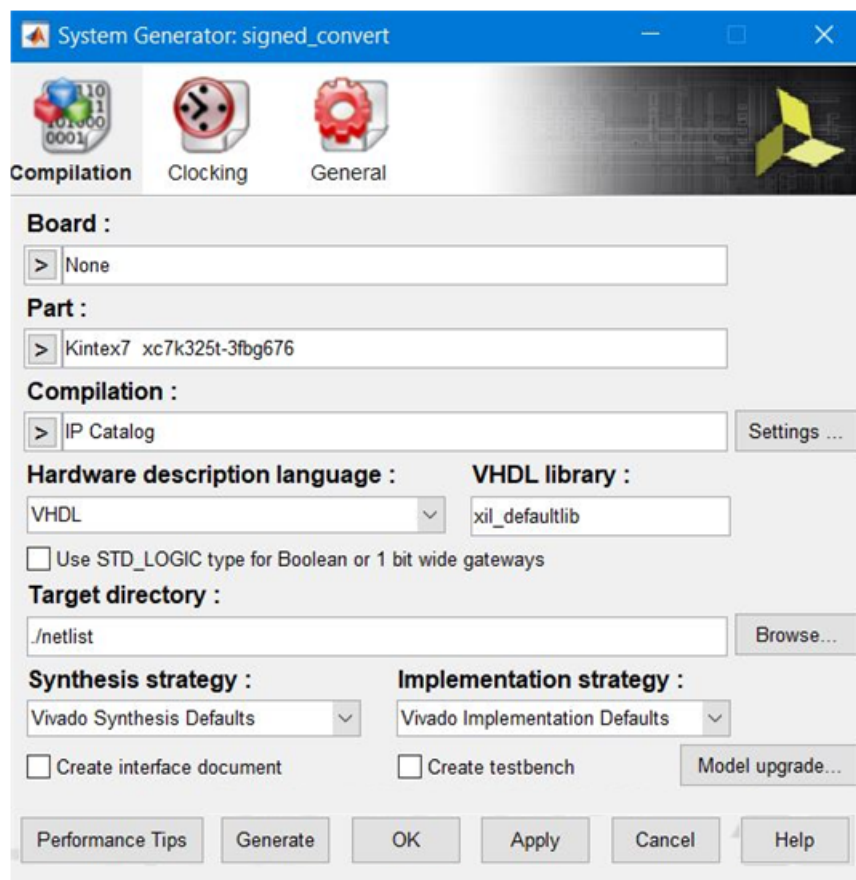
System Generator は、デザインを自動的に低級言語表現にコンパイルします。デザインをコンパイルまたはシミュレーションするには、System Generator トークンを使用します。このセクションでは、このトークンの使用法を説明します。

System Generator トークンは、ザイリンクス ブロックセットの Basic Element および Tools ライブラリに含まれています。System Generator トークンは、その他のザイリンクス ブロックと同様、Index ライブラリにも含まれています。

デザインには、System Generator トークンを 1 つ以上含める必要があり、複数の System Generator トークンを異なるレベルに含めることが可能です (レベルごとに 1 つ)。上位にほかの System Generator トークンがあるものはスレーブ、上位にほかの System Generator トークンがないものはマスターです。1 つの System Generator トークンの適用範囲は、トークンが挿入されているレベルと、その下にあるすべてのサブシステムです。[Simulink System Period] などの一部のパラメーターは、マスターでしか指定できません。

System Generator トークンを追加したら、コードの生成および合成の処理方法を指定できます。次の図に、System Generator トークンのパラメーター ダイアログ ボックスを示します。

図 22: System Generator トークンのパラメーター ダイアログ ボックス



コンパイル タイプと [Generate] ボタン

[Generate] ボタンをクリックすると、System Generator でデザインの一部分が等価の低級言語表現にコンパイルされます。コンパイルされる部分は、ブロックを含むサブシステムがルートとなっている部分です。デザイン全体をコンパイルするには、System Generator トークンをデザインの最上位に配置します。コンパイル タイプ ([Compilation]) は、生成する結果のタイプを指定します。次のオプションがあります。

- [IP Catalog]: デザインを Vivado IP カタログに追加して別のデザインで使用できる IP コアとしてパッケージします。

- [Hardware Co-Simulation (JTAG/Point-to-point Ethernet)]: ハードウェアのアルゴリズムを検証するハードウェア協調シミュレーション ブロックを生成します。
- [Synthesized Checkpoint]: Vivado IDE プロジェクトで使用できる合成済みチェックポイント ファイル (synth_1.dcp) を作成します。
- [HDL Netlist]: VHDL または Verilog RTL デザインを生成します。

表 1: System Generator トークンのパラメーター ダイアログ ボックスのオプション

GUI	説明
[Board]	<p>デザインをテストするのに使用するザイリンクス パートナーまたはカスタム ボードを指定します。</p> <p>パートナー ボードまたはカスタム ボードが [Board] リストに表示されるようにするには、ボードを記述するボード ファイルにアクセスするように System Generator を設定する必要があります。System Generator でのボード サポートについては、System Generator でのボード サポートの指定を参照してください。</p> <p>[Board] を選択すると、[Part] フィールドに [Board] で選択したボード上にあるザイリンクス デバイスの名前が表示されます。このパーツ名は変更できません。</p>
[Part]	<p>使用するザイリンクス デバイスを指定します。[Board] を選択すると、[Part] フィールドに [Board] で選択したボード上のザイリンクス デバイスの名前が表示されます。このパーツ名は変更できません。</p>
[Hardware description language]	<p>デザインの HDL ネットリストで使用する言語を指定します。[VHDL] または [Verilog] を選択できます。</p>
[VHDL library]	<p>コードを生成する VHDL work ライブラリを指定します。デフォルト名は xil_defaultlib です。</p>
[Use STD_LOGIC type for Boolean or 1 bit wide gateways]	<p>デザインのハードウェア記述言語 (HDL) が VHDL の場合、このオプションを選択すると、ブールまたは 1 ビット ポート (Gateway In または Gateway Out) が STD-LOGIC 型として宣言されます。このオプションを選択しない場合は、ブールまたは 1 ビット ポートは System Generator でベクターとして変換されます。</p>
[Target Directory]	<p>System Generator のコンパイル結果を保存するディレクトリを指定します。System Generator および FPGA インプリメンテーション ツールでは多数のファイルが生成されるので、個別のディレクトリ (Simulink モデル ファイルが含まれるディレクトリとは別のディレクトリ) を作成することをお勧めします。ディレクトリは、絶対パス (c:\netlist など) またはモデルを含むディレクトリを基準とした相対パス (netlist など) で指定できます。</p>
[Synthesis strategy]	<p>ドロップダウン リストの定義済みストラテジから合成ストラテジを選択します。</p>
[Implementation strategy]	<p>ドロップダウン リストの定義済みストラテジからインプリメンテーション ストラテジを選択します。</p>
[Create interface document]	<p>オンにして [Generate] ボタンをクリックすると、System Generator でネットリスト記述されるデザインを示す HTML ファイルが作成されます。このファイルは、ネットリスト フォルダーに含まれます。</p>

表 1: System Generator トークンのパラメーター ダイアログ ボックスのオプション (続き)

GUI	説明
[Create testbench]	System Generator で HDL テストベンチを作成するよう指定します。HDL シミュレータでテストベンチをシミュレーションし、コンパイルされたデザインのシミュレーション結果を Simulink シミュレーション結果と比較します。System Generator では、デザインを Simulink でシミュレーションし、Gateway ブロックで検出される値を保存することにより、テストベクターを作成します。テストベンチの最上位 HDL ファイルの名前は、<name>_tb.vhd/.v となります (<name> はテストするデザインの部分から導出された名前、拡張子はハードウェア記述言語により異なる)。
[FPGA clock period]	システム クロックの周期を ns で指定します。値は整数である必要はありません。ここで指定した周期は、制約ファイルでグローバル PERIOD 制約として設定され、サイリンクス インプリメンテーション ツールに渡されます。マルチサイクル パスは、この値の整数倍に制約されます。
[Clock pin location]	ハードウェア クロックのピン ロケーションを指定します。この情報は、制約ファイルを介してサイリンクス インプリメンテーション ツールに渡されます。

Simulink システム周期

System Generator ブロックのパラメーター ダイアログ ボックスで、Simulink システム周期 ([Simulink system period]) を指定する必要があります。この値は、デザインのシミュレーションを実行する基準レートを秒で指定します。Simulink システム周期は、デザインで使用されるすべてのサンプル周期の公約数にする必要があります。たとえば、サンプル周期 2、6、8 を使用するブロックを含むデザインでは、使用可能な最大の Simulink システム周期は 2 ですが、1 および 0.5 も使用可能です。サンプル周期は、明示的に指定するか、自動的に算出されるか、内部レートが変わるブロック内で暗示的に決まります。システム周期のハードウェア クロックへの影響については、[タイミングとクロック](#)を参照してください。

デザインをシミュレーションまたはコンパイルする前に、System Generator でシステム周期がデザインのすべてのサンプル周期の公約数になっているかが検証されます。

周期を調整できないために、一貫性がない System Generator モデルになってしまう可能性もあります。たとえば、システム レートで動作する必要のあるブロックで Up Sample ブロックを駆動すると、一貫性のないモデルになります。システム周期をアップデートしても System Generator で競合がレポートされる場合は、モデルに一貫性がないということであり、修正が必要です。

周期は階層で制御されます。詳細は、[階層制御](#)を参照してください。

ブロック アイコンの表示

モデルに表示するブロックのアイコンの表示を制御します。モデルをコンパイルした後 (生成またはシミュレーションを実行、あるいは [Ctrl-D] を押す)、ブロックの情報がこのオプションでの選択に応じて表示されます。

- [Default]: ポートの方向に関する基本的な情報が表示されます。
- [Sample rates]: 正規化サンプル周期やサンプル周波数 (MHz) など、各ポートのサンプル レートが表示されます。
- [Pipeline stages]: パイプライン段数が表示されます。
- [HDL port names]: ポート名が表示されます。
- [Input data types]: 各ポートの入力データ型が表示されます。
- [Output data types]: 各ポートの出力データ型が表示されます。

階層制御

System Generator トークンの [Simulink system period] オプション ([Simulink システム周期](#)を参照) は、階層で制御されます。System Generator トークンの設定はそのトークンのスコープに適用されますが、下位にある System Generator トークンの設定で変更できます。たとえば、デザインの最上位にある System Generator トークンで [Simulink system period] が設定されており、サブシステム S に配置されている System Generator トークンで別の値が設定されている場合、サブシステムでは 2 番目のシステム周期が使用されますが、デザインのその他の部分では最上位の System Generator トークンで設定されたシステム周期が使用されます。

コンパイル結果

このセクションでは、System Generator トークンで [HDL Netlist] を選択して [Generate] ボタンをクリックしたときに System Generator で生成される下位ファイルについて説明します。生成されるファイルには、デザインをインプリメントする HDL が含まれます。System Generator では、HDL ファイルとその他のハードウェア ファイルが 1 つの Vivado® IDE プロジェクトで管理されます。すべてのファイルは、System Generator トークンで指定したディレクトリに生成されます。テストベンチを作成するよう指定していない場合 ([Create testbench] をオフ)、System Generator で生成されるファイルは次のとおりです。

表 2: コンパイル ファイル

ファイル名/タイプ	説明
<design_name>.vhd/.v	階層構造ネットリストとクロック/クロック イネーブル制御が含まれます。
<design_name_entity_declarations>.vhd/.v	デザインの System Generator ブロックのモジュール定義のエンティティが含まれます。
<design_name>.xpr	Vivado IDE デザインの属性をすべて記述した Vivado IDE プロジェクト ファイル。

テストベンチを作成するよう指定している場合は ([Create testbench] をオン)、System Generator で上記のファイルに加え、シミュレーション結果を比較するためのファイルが生成されます。Simulink® でのシミュレーション結果と、ModelSim、Vivado シミュレータ、Model、VCS などの Vivado® IDE でサポートされる RTL シミュレータからのシミュレーション結果が比較されます。追加のファイルは次のとおりです。

表 3: 追加のコンパイル ファイル

ファイル名/タイプ	説明
DAT ファイル	Simulink でのシミュレーション結果が含まれます。
<design_name>_tb.vhd/.v	デザインをラップするテストベンチ。シミュレーションを実行すると、このテストベンチによりデジタル シミュレータのシミュレーション結果と Simulink のシミュレーション結果が比較されます。

Sysmte Generator 制約ファイルの使用

コード生成中にデザインをコンパイルすると、System Generator によりダウンストリーム ツールでのデザインの処理方法を指示する制約が生成されます。この制約ファイルにより、より質の高いインプリメンテーションが短時間で生成されます。制約には、次の情報が含まれています。

- システム クロックの周期。
- デザインのさまざまな部分のシステム クロックに対するスピード要件。
- ポートを配置する必要のあるピン ロケーション。

- ポートの動作スピード。

システム クロック周期 (デザインで最速のハードウェア クロックの周期) は、System Generator トークンで指定できます。System Generator で指定した周期は、制約ファイルに記述されます。ダウンストリーム ツールでデザインがインプリメントされる際、この周期が目標として使用されます。

マルチサイクル パス制約

多くのデザインは、異なるクロック レートで動作する複数の部分から構成されています。最速の部分ではシステム クロックが使用され、その他の部分のクロック周期は、システム クロック周期の整数倍になります。ダウンストリーム ツールに、デザインの各部分で達成する必要があるスピードを伝達する必要があります。この情報により、ツールの効率および効果が大幅に向上し、短いコンパイル時間でより良いハードウェアを実現できます。デザインの分割、それぞれの部分のスピードは、制約ファイルでマルチサイクル パス制約を使用して指定します。

IOB タイミング制約と配置制約

System Generator の Gateway In と Gateway Out ブロックは、ハードウェアに変換されると入力ポートと出力ポートになります。これらのポートの位置とスピードは、Gateway In および Gateway Out ブロックのパラメーター ダイアログ ボックスで入力します。ポートのロケーションとスピードは、制約ファイルで IOB タイミングごとに指定されます。

このセクションでは、System Generator で生成される HDL でハードウェア クロックがどのように処理されるかを説明します。<design> という名前のデザインがあり、<design> は HDL 識別子として有効であるとしてします。System Generator でデザインをコンパイルすると、複数の HDL エンティティまたはモジュールが記述され、最上位のものに <design> という名前が付けられ、<design>.vhd/.v というファイルに保存されます。

[Clock Enables] マルチレート インプリメンテーション

クロックとクロック イネーブルは対にして、HDL 全体に配置されます。典型的なクロック名は clk_1、clk_2、clk_3 で、その対になるクロック イネーブルの名前はそれぞれ ce_1、ce_2、および ce_3 です。名前からクロック/クロック イネーブルのペアの動作レートがわかります。clk_1 および ce_1 で駆動されるロジックはシステム レート (最速) で動作し、clk_2 および ce_2 で駆動されるロジックはシステム レートの 1/2 で動作します。クロックおよびクロック イネーブルは、<design> エンティティまたはモジュール 内では駆動されず、最上位入力ポートとなります。

System Generator で生成された HDL に含まれるクロックとクロック イネーブルの名前から、クロッキングは一般的であるように見えますが、そうではありません。これを説明するため、clk_1 と clk_2 というクロックと、対応するクロック イネーブル ce_1 および ce_2 が含まれるデザインがあるとします。この場合、ハードウェアで ce_1 および ce_2 信号を High にし、clk_2 を clk_1 の 1/2 のレートのクロック信号で駆動すればよさそうに思えますが、ほとんどの System Generator デザインはこれでは機能しません。その代わり、clk_1 と clk_2 を同じクロックで駆動し、ce_1 を High に接続して、ce_2 を clk_1 と clk_2 の 1/2 のレートにします。

IP インスタンスのキャッシュ

Vivado 合成を実行して出力ファイルを生成するコンパイル ターゲットに対しては、デザイン プロセスの繰り返し実行にかかる時間を短縮するため、System Generator でディスク キャッシュが使用されます。

デザインでキャッシュを有効にすると、コンパイルで合成用に IP インスタンスが生成され、Vivado 合成ツールで合成出力ファイルが生成されたときに、キャッシュ エリアにエントリが作成されます。

キャッシュにエントリが作成されると、IP を新しくカスタマイズしたときにまったく同じプロパティが使用されていれば、IP は合成されず、キャッシュが参照されて、キャッシュ内の該当する合成出力ファイルがデザインの出力ディレクトリにコピーされます。IP インスタンスは再度合成されません。このプロセスはデザインに含まれるすべての IP に対して繰り返されるので、出力ファイルの生成は短時間で終了します。

次のコンパイル ターゲットでは Vivado 合成が実行されますが、デザイン内の IP を合成するためキャッシュにアクセスします。

- ハードウェア協調シミュレーション
- 合成済みチェックポイント

また、デザインをコンパイルするときに [Perform analysis] に [Timing] または [Resource] を指定した場合も、コンパイル ターゲットに関係なく Vivado 合成が実行されます。タイミング解析またはリソース解析は設計中何度も実行するので、IP キャッシュを有効にしておくと、全体的なパフォーマンスが改善します。[Perform analysis] コンパイル オプションの詳細は、[タイミング解析の実行](#) または [リソース解析の実行](#) を参照してください。

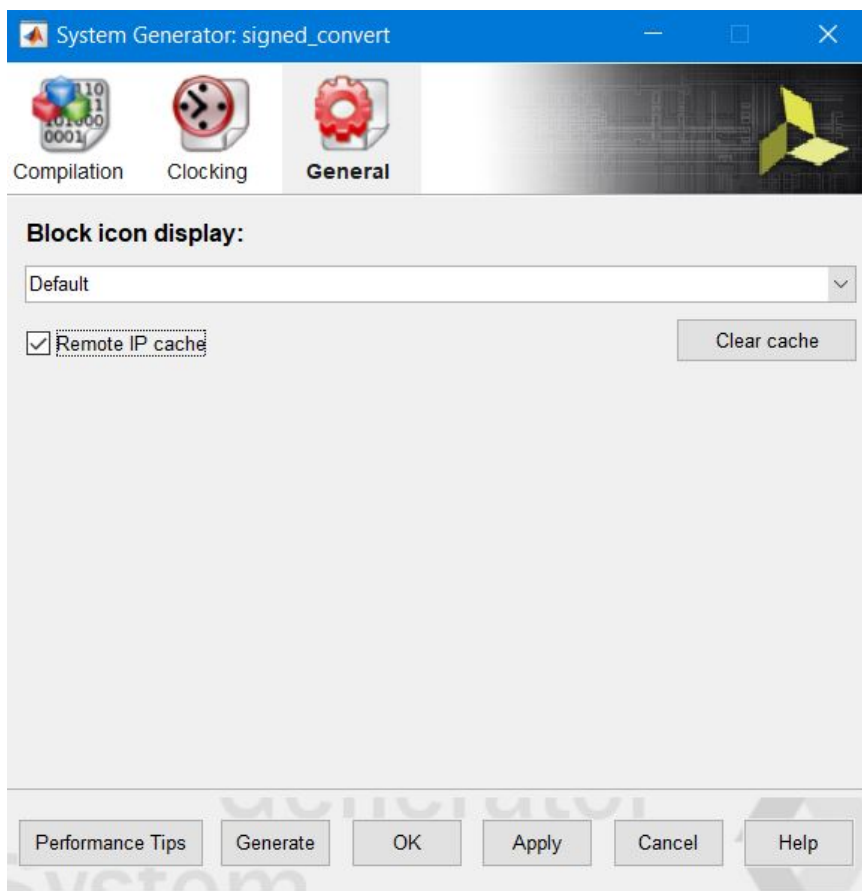
IP キャッシュは、システム上の複数の Simulink モデル間で共有されます。1 つのデザインに含まれる IP を別のデザインで再利用し、IP のカスタマイズがまったく同じで、両方の Simulink モデルのパーツおよび言語設定が同じ場合、いずれかのデザインをコンパイルする際にキャッシュが使用されます。

IP キャッシュを有効にするには、System Generator トークンのダイアログ ボックスで [Remote IP cache] をオンにします。キャッシュは、この後コンパイルが実行されるたびに参照されます。



注意: IP キャッシュは、デザインに存在する IP の数によってサイズが大きくなる可能性があります。

図 23: ブロック アイコンの表示



System Generator トークンのダイアログ ボックスで [Clear cache] をクリックすると、キャッシュをクリアしてディスク スペースを節約できます。

システム内の IP キャッシュ ディレクトリは、MATLAB コマンド ラインで「`xilinx.environment.getipcachepath`」と入力すると確認できます。MATLAB コマンド ウィンドウに IP キャッシュ ディレクトリの完全パスが表示されます。

```
>> xilinx.environment.getipcachepath
ans =
C:/Users/your_id/AppData/Local/Xilinx/Sysgen/SysgenVivado/win64.o/ip
```

System Generator での IP キャッシュは、『Vivado Design Suite ユーザー ガイド: IP を使用した設計』(UG896)のこのセクションに説明されている Vivado Design Suite での IP キャッシュと同様です。ただし、System Generator デザインの IP キャッシュのディレクトリと Vivado プロジェクトの IP キャッシュのディレクトリは異なります。

Vivado プロジェクト

コンパイル ターゲットとして [HDL Netlist] および [IP Catalog] を選択すると、コード生成の結果の統合方法を示すサンプル Vivado プロジェクトも生成されます。

[HDL Netlist] を選択した場合、System Generator で設計されたモジュールが Vivado プロジェクトの最上位に設定され、IP のインスタンスも含まれます。System Generator トークンで [Create testbench] をオンにした場合は、テストベンチおよびスティミュラス ファイル (*.dat) もプロジェクトに追加されます。

[IP Catalog] を選択した場合は、次の機能を含むサンプル プロジェクトが作成されます。

- System Generator から生成された IP がプロジェクトに関連付けられた IP カタログに追加されており、RTL フローおよび IP インテグレーター ベース フローで使用できます。
- <design>_stub の下に、RTL フローでの IP のインスタンス化方法を示す <ip>_0 という IP の RTL インスタンス化が含まれます。
- RTL フローで IP をインスタンス化するための <design>_tb というテストベンチが含まれます。

注記: Gateway In または Gateway Out ブロックで [AXI4-Lite] スレーブ インターフェイス生成が選択されている場合は、テストベンチは作成されません。

- このサンプル プロジェクトで選択されているパーツが Zynq-7000 SoC の場合は、プロジェクトに Zynq-7000 サブシステムを含むサンプル IP インテグレーター図が含まれます。その他すべてのパーツには、MicroBlaze™ ベースのサブシステムが作成されます。

HDL テストベンチ

通常、System Generator デザインはビット精度およびサイクル精度であり、Simulink でのシミュレーション結果はハードウェアでの動作と完全に一致します。ただし、場合によっては、Simulink シミュレーションの結果と HDL シミュレータによるシミュレーション結果を比較すると役立つことがあります。デザインにブラック ボックスが含まれている場合は特にそうです。System Generator トークンのパラメーター ダイアログ ボックスで [Create Testbench] をオンにすると、これが可能になります。

<design> というデザインの最上位に System Generator トークンが配置されているとします。このトークンのパラメーター ダイアログ ボックスでは、[Compilation] が [HDL Netlist] に設定されており、[Create Testbench] がオンになっています。[Generate] をクリックすると、System Generator により、通常デザインに対して生成されるファイルに加えて次のファイルが生成されます。

1. テストベンチ HDL エンティティを含む <design>_tb.vhd/.v。
2. HDL テストベンチ シミュレーションで使用するベスト プラクティスを含む .dat ファイル。

3. Vivado Integrated Design Environment (IDE) を使用して RTL シミュレーションを実行できます。詳細は、『Vivado Design Suite ユーザー ガイド: ロジック シミュレーション』 (UG900) を参照してください。

System Generator で生成される .dat ファイルには、Gateway ブロックを通過する値が保存されます。HDL シミュレーションでは、.dat ファイルからの入力値がスティミュラス、出力値が予測される結果となります。テストベンチは、デザインの HDL にスティミュラスを供給し、HDL の結果と予測結果を比較するためのラッパーです。

MATLAB の FPGA へのコンパイル

System Generator では、MCode ブロックにより MATLAB が直接サポートされています。MCode ブロックは、入力値を M 関数に適用し、サイリンクスの固定小数点型を使用して評価します。評価は、サンプル周期ごとに実行されます。MCode ブロックでは、持続型状態変数を使用することにより、内部ステートを保持できます。入力ポートは指定の M 関数の入力引数、出力ポートは M 関数の出力引数により決定されます。MCode ブロックは、有限ステート マシン、制御ロジック、計算負荷の高いシステムを構築するのに便利です。

MCode ブロックを使用するには、M 関数を記述する必要があります。M ファイルは、M ファイルを使用するモデルのディレクトリまたは MATLAB パスに配置します。

次に、MCode ブロックを使用する例を示します。

- 例 1: [単純なセクター](#) - 入力の最大値を返すファンクションをインプリメントする方法を示します。
- 例 2: [単純な算術演算](#) - 単純な演算処理をインプリメントする方法を示します。
- 例 3: [レイテンシのある複素乗算器](#) - レイテンシを持つ複素乗算器を構築する方法を示します。
- 例 4: [シフト操作](#) - シフト操作をインプリメントする方法を示します。
- 例 5: [MCode ブロックにパラメーターを渡す](#) - MCode ブロックにパラメーターを渡す方法を示します。
- 例 6: [オプションの入力ポート](#) - MCode ブロックにオプションの入力ポートをインプリメントする方法を示します。
- 例 7: [有限ステートマシン](#) - 有限ステート マシンのインプリメント方法を示します。
- 例 8: [パラメーター指定可能なアキュムレータ](#) - パラメーター指定可能なアキュムレータを構築する方法を示します。
- 例 9: [FIR 例とシステム検証](#) - FIR ブロックをモデリングし、システム検証を実行する方法を示します。
- 例 10: [RPN カリキュレーター](#) - RPN カリキュレーター (スタック マシン) をモデリングする方法を示します。
- 例 11: [disp 関数の例](#) - disp 関数を使用して変数値をプリントする方法を示します。

単純なセクター

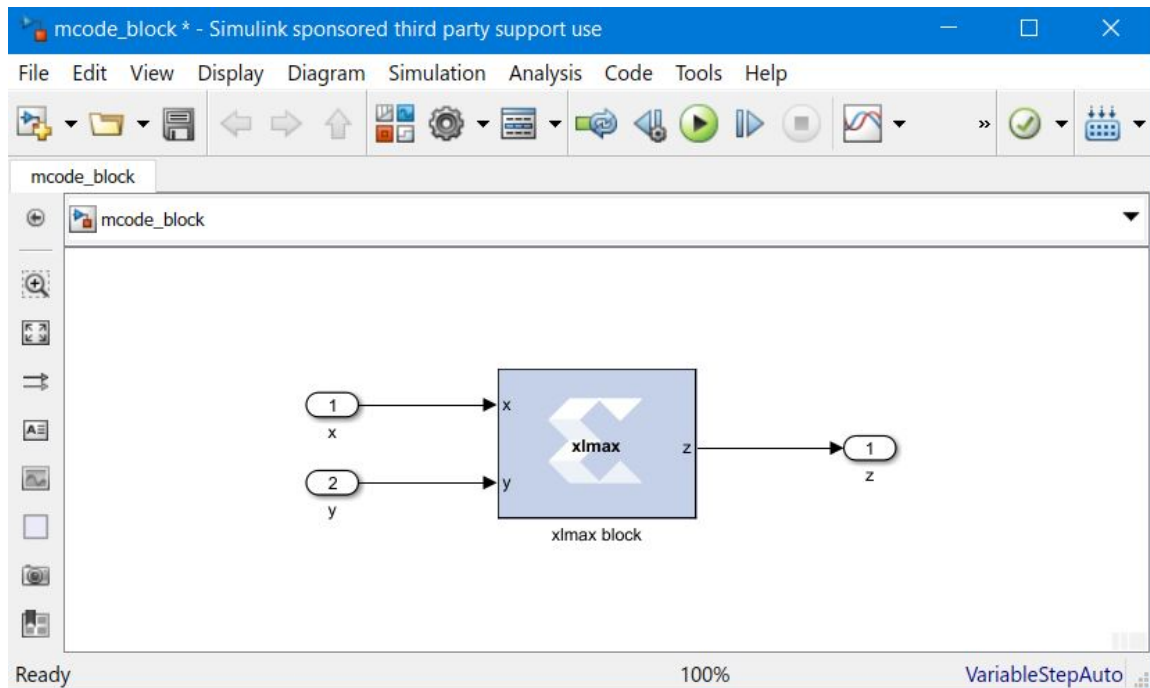
この例では、2 つの入力の最大値を出力に割り当てる、単純なデータパス コントローラーを示します。M 関数は次のように定義され、xlmax.m という M ファイルに保存されます。

```
function z = xlmax(x, y)
    if x > y
        z = x;
    else
        z = y;
    end
```

xlmax.m ファイルは、モデル ファイルと同じディレクトリまたは MATLAB パスに保存する必要があります。
xlmax.m を適切な場所に保存したら、MCode ブロックをモデルにドラッグしてパラメーター ダイアログ ボックスを開き、[MATLAB Function] に「xlmax」と入力します。[OK] をクリックすると、ブロックに入力ポート x と y、出力ポート z が表示されます。

次の図に、モデルをコンパイルした後のブロックを示します。ブロックで計算が実行され、出力ポートに必要な固定小数点型が設定されます。

図 24: 単純なセレクトアー デザイン



単純な算術演算

この例では、単純な算術演算と型変換を示します。xlSimpleArith M 関数を定義する xlSimpleArith.m ファイルの内容は、次のとおりです。

```
function [z1, z2, z3, z4] = xlSimpleArith(a, b)
% xlSimpleArith demonstrates some of the arithmetic operations
% supported by the Xilinx MCode block. The function uses xfix()
% to create Xilinx fixed-point numbers with appropriate
% container types.%
% You must use a xfix() to specify type, number of bits, and
% binary point position to convert floating point values to
% Xilinx fixed-point constants or variables.
% By default, the xfix call uses xlTruncate
% and xlWrap for quantization and overflow modes.
% const1 is Ufix_8_3
const1 = xfix({xlUnsigned, 8, 3}, 1.53);
% const2 is Fix_10_4
const2 = xfix({xlSigned, 10, 4, xlRound, xlWrap}, 5.687);
z1 = a + const1;
z2 = -b - const2;
```

```
z3 = z1 - z2;
% convert z3 to Fix_12_8 with saturation for overflow
z3 = xfix({xlSigned, 12, 8, xlTruncate, xlSaturate}, z3);
% z4 is true if both inputs are positive
z4 = a>const1 & b>-1;
```

この M 関数では、加算演算子と減算演算子を使用しています。MCode ブロックは、これらの演算子を完全精度で計算します。つまり、出力の精度は、これらの演算を情報を失わずに実行するのに十分であるということです。

ここで、`xfix` という関数の呼び出しに注目してください。この関数には、固定小数点型の精度と値の 2 つの引数が必要です。精度は、セル配列で指定します。精度のセル配列の最初の要素はデータ型で、`xlUnsigned`、`xlSigned`、`xlBoolean` のいずれかです。2 番目の要素は固定小数点値のビット数、3 番目の要素は 2 進小数点の位置です。データ型が `xlBoolean` の場合は、ビット数および 2 進小数点の位置を指定する必要はありません。ビット数と 2 進小数点の位置は、対にして指定する必要があります。4 番目の要素は量子化モード、5 番目の要素はオーバーフロー モードです。量子化モードは、`xlTruncate`、`xlRound`、または `xlRoundBanker` のいずれかに指定します。オーバーフロー モードは、`xlWrap`、`xlSaturate`、または `xlThrowOverflow` のいずれかに指定します。量子化モードとオーバーフロー モードは、対にして指定する必要があります。量子化モードとオーバーフロー モードを指定しない場合は、`xfix` 関数で符号付きおよび符号なしの値に対して `xlTruncate` と `xlWrap` が使用されます。`xfix` 関数の 2 番目の引数は、倍精度またはザイリンクス固定小数点値で指定します。定数が整数値である場合は、`xfix` 関数を使用する必要はありません。MCode ブロックで、自動的に適切な固定小数点値に変換されます。

MCode ブロックのパラメーター ダイアログ ボックスで [MATLAB function] に「`xlSimpleArith`」を入力した場合は、ブロックに 2 つの入力ポート `a` および `b` と、4 つの出力ポート `z1`、`z2`、`z3`、および `z4` が表示されます。

図 25: `xlSimpleArith` MCode パラメーター

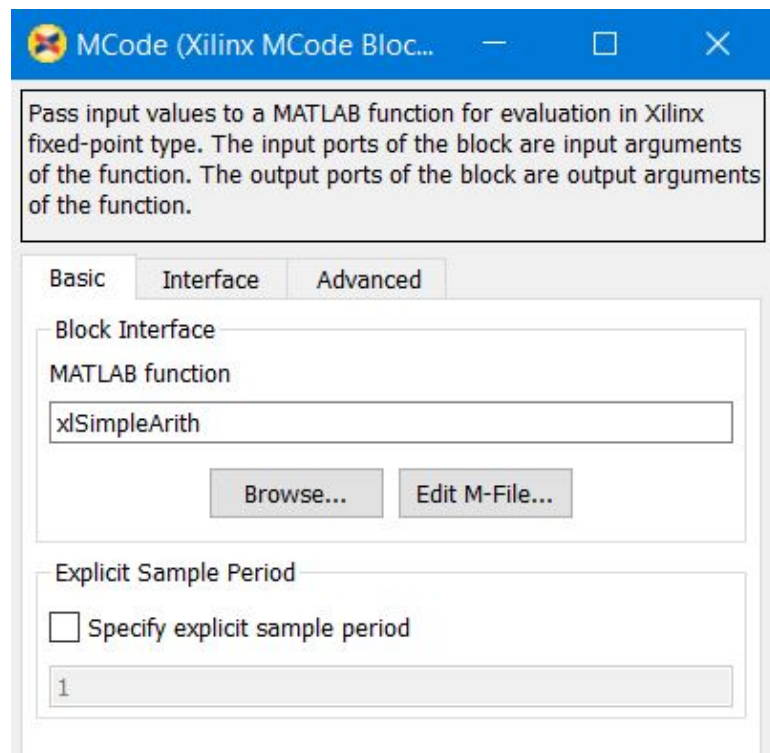
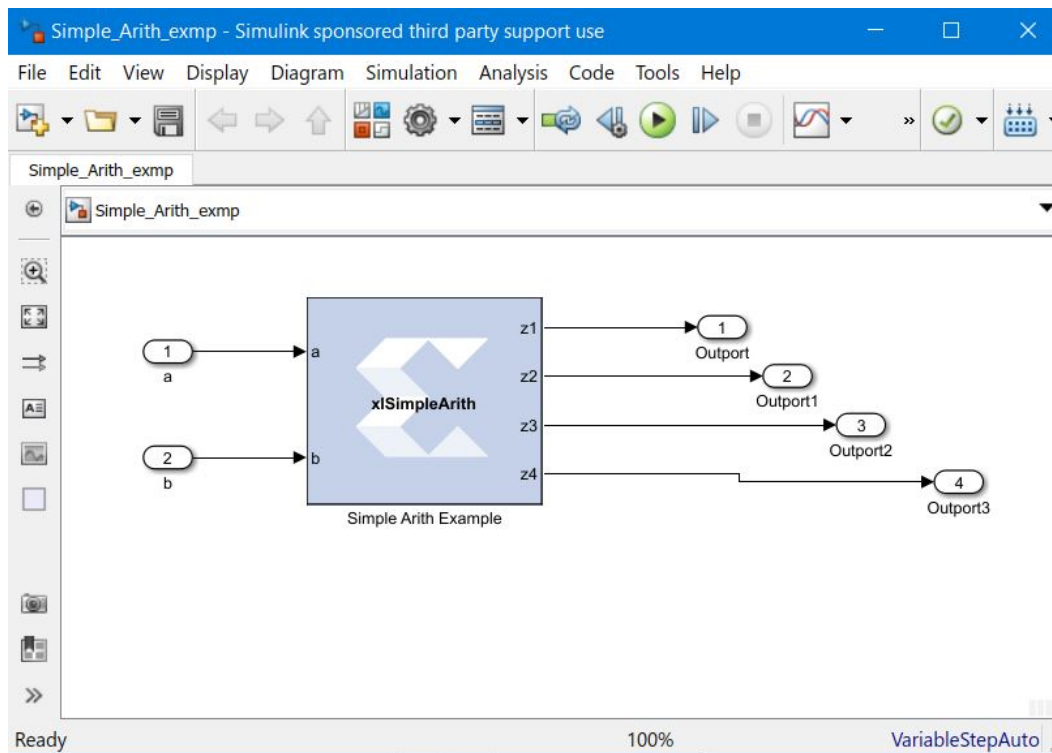


図 26: xISimpleArith デザイン



ザイリンクスのデータ型および関数を使用する M 関数は、MATLAB のコマンド ウィンドウでテストできます。たとえば、MATLAB のコマンド ウィンドウに「`[z1, z2, z3, z4] = xISimpleArith(2, 3)`」と入力すると、次の行が表示されます。

```
UFix(9, 3): 3.500000
Fix(12, 4): -8.687500
Fix(12, 8): 7.996094
Bool: true
```

2 つの引数 2 および 3 は、自動的に固定小数点値に変換されています。引数として浮動小数点値を使用する場合は、`xfix` 関数呼び出しが必要です。

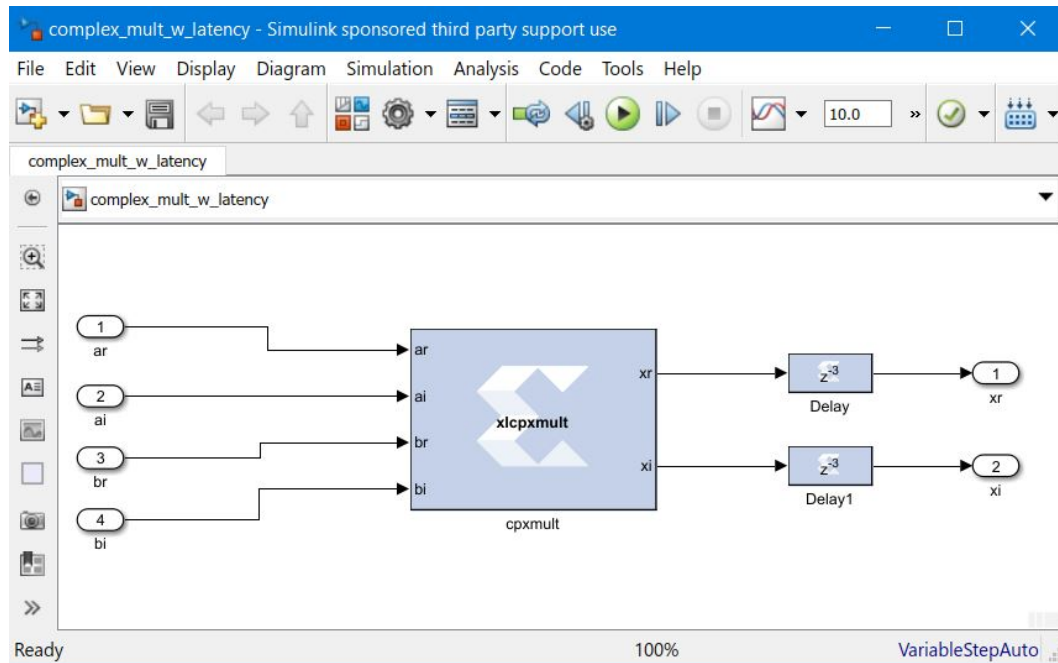
レイテンシのある複素乗算器

この例では、複素乗算器の作成方法を示します。xlcpxmult 関数を定義する xlcpxmult.m ファイルの内容は、次のとおりです。

```
function [xr, xi] = xlcpxmult(ar, ai, br, bi)
    xr = ar * br - ai * bi;
    xi = ar * bi + ai * br;
```

次の図に、サブシステムを示します。

図 27: 複素乗算器サブシステム



MCode ブロックの後に、2 つの Delay ブロックが追加されています。Delay ブロックのパラメーター ダイアログ ボックスの [Implementation] タブで [Implement using behavioral HDL] をオンにすると、ダウンストリームのロジック合成ツールで適切な最適化を実行してパフォーマンスを向上できます。

シフト操作

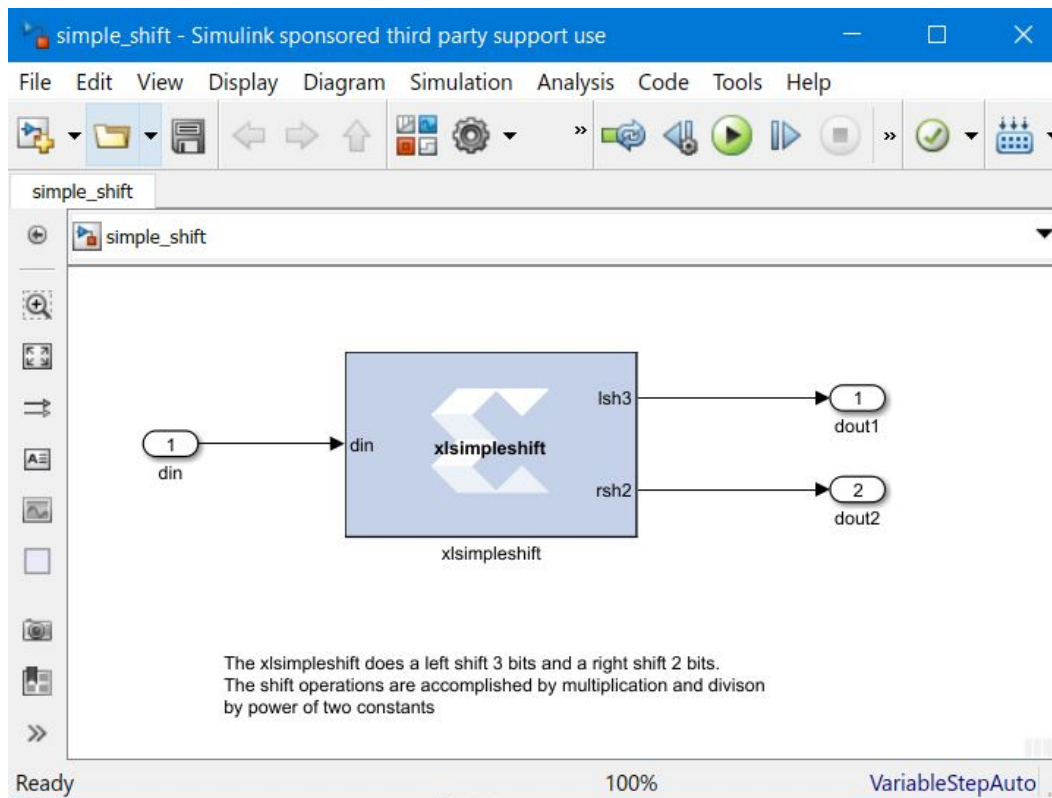
この例では、MCode ブロックを使用したビットシフト操作のインプリメント方法を示します。シフト操作は、2 のべき乗での乗算および除算で達成されます。たとえば、4 で乗算することは 2 ビットの左シフトと同じであり、8 で割ることは 3 ビットの右シフトと同じです。シフト操作は、2 進小数点の位置の移動および必要に応じてビット幅の拡張によりインプリメントされます。Fix_8_4 値を 4 で乗算すると Fix_8_2 値になり、Fix_8_4 を 64 で乗算すると Fix_10_0 値になります。

次に、1 左シフトと 1 右シフトを定義する `xlsimpleshift.m` ファイルの内容を示します。

```
function [lsh3, rsh2] = xlsimpleshift(din)
% [lsh3, rsh2] = xlsimpleshift(din) does a left shift
% 3 bits and a right shift 2 bits.
% The shift operation is accomplished by
% multiplication and division of power
% of two constant.
lsh3 = din * 8;
rsh2 = din / 4;
```

次の図に、コンパイル後のサブシステムを示します。

図 28: シフト操作



MCode ブロックにパラメーターを渡す

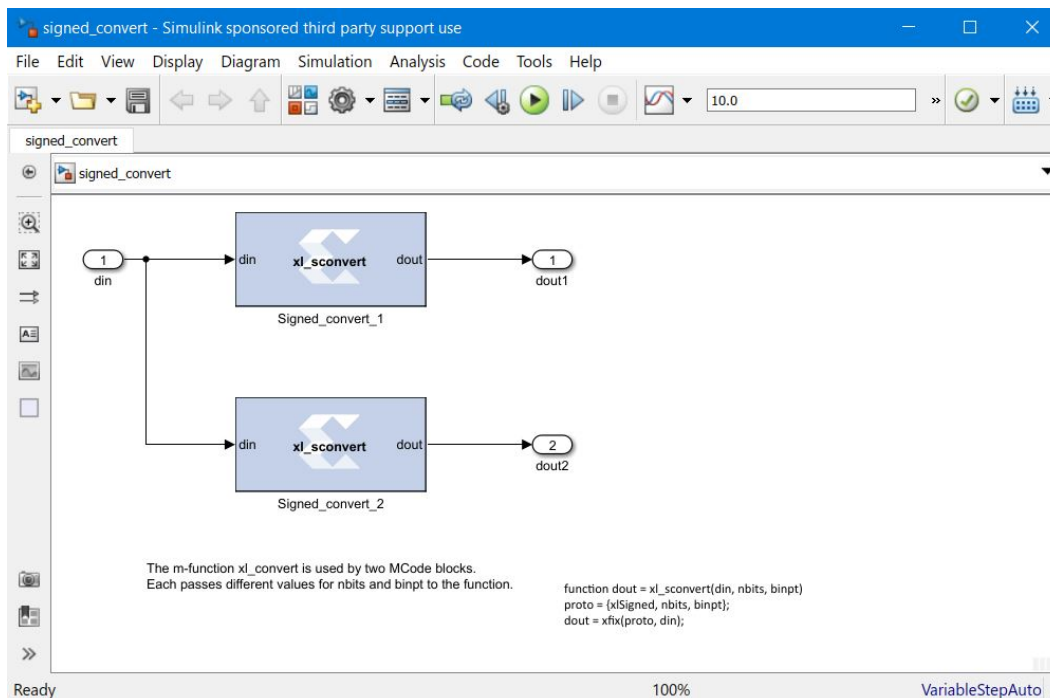
この例では、MCode ブロックにパラメーターを渡す方法を示します。M 関数への入力引数は、MCode ブロックの入力ポートとして、またはブロックの内部パラメーターとして解釈されます。

次の M コードは、`xl_sconvert.m` に含まれる M 関数 `xl_sconvert` を定義します。

```
function dout = xl_sconvert(din, nbits, binpt)
    proto = {xlSigned, nbits, binpt};
    dout = xfix(proto, din);
```

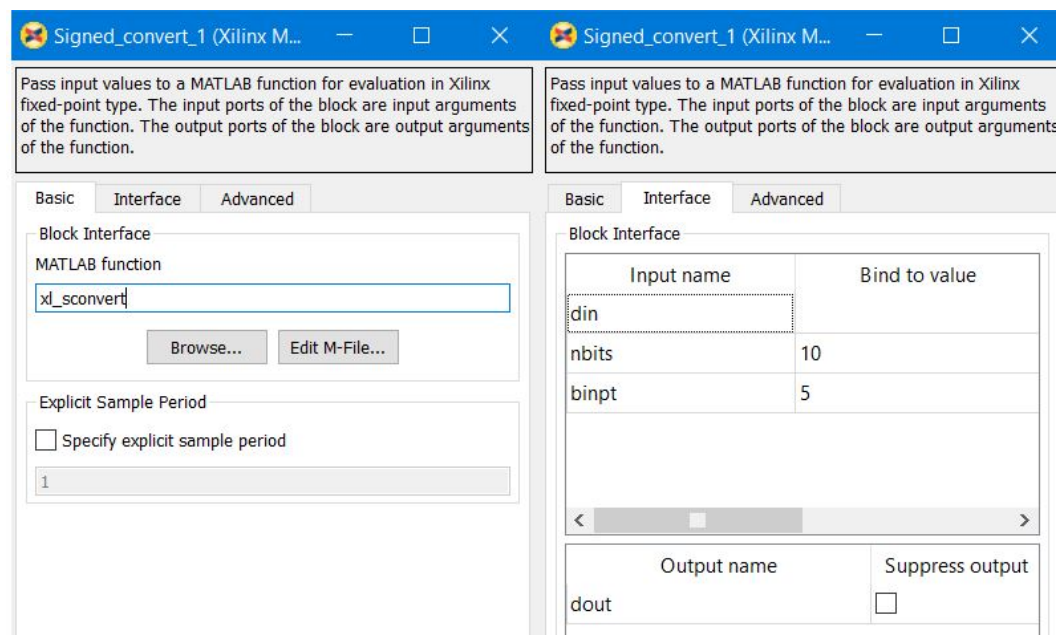
次の図に、M 関数 `xl_sconvert` を使用する 2 つの MCode ブロックを含むサブシステムを示します。M 関数の引数 `nbits` と `binpt` には、各 MCode ブロックに異なるパラメーターを渡すことにより、異なる値が指定されます。
`signed convert 1` という MCode ブロックでは、渡されるパラメーターにより入力データが `Fix_16_8` から `Fix_10_5` に変換されます。`signed convert 2` という MCode ブロックでは、渡されるパラメーターにより入力データが `Fix_16_8` から `Fix_8_4` に変換されます。

図 29: 2 つの MCode ブロックを含むサブシステム



各 MCode ブロックにパラメーターを渡すには、MCode ブロックのパラメーター ダイアログ ボックスで [Edit M-File] タブをクリックし、M 関数の引数を設定します。次に、MCode ブロック `signed convert 1` の設定を示します。

図 30: MCode ブロックのマスク



上の図の [interface] タブでは、M 関数の引数 `[nbits]` を 10 に、`[binpt]` を 5 に設定しています。次に、MCode ブロック `signed convert 2` の設定を示します。

図 31: MCode ブロック signed convert 2 のマスク

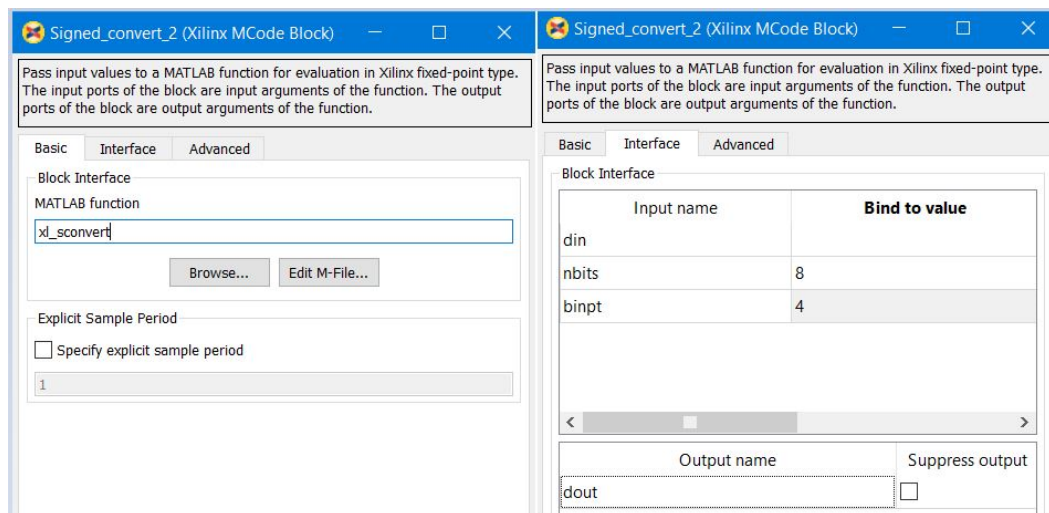
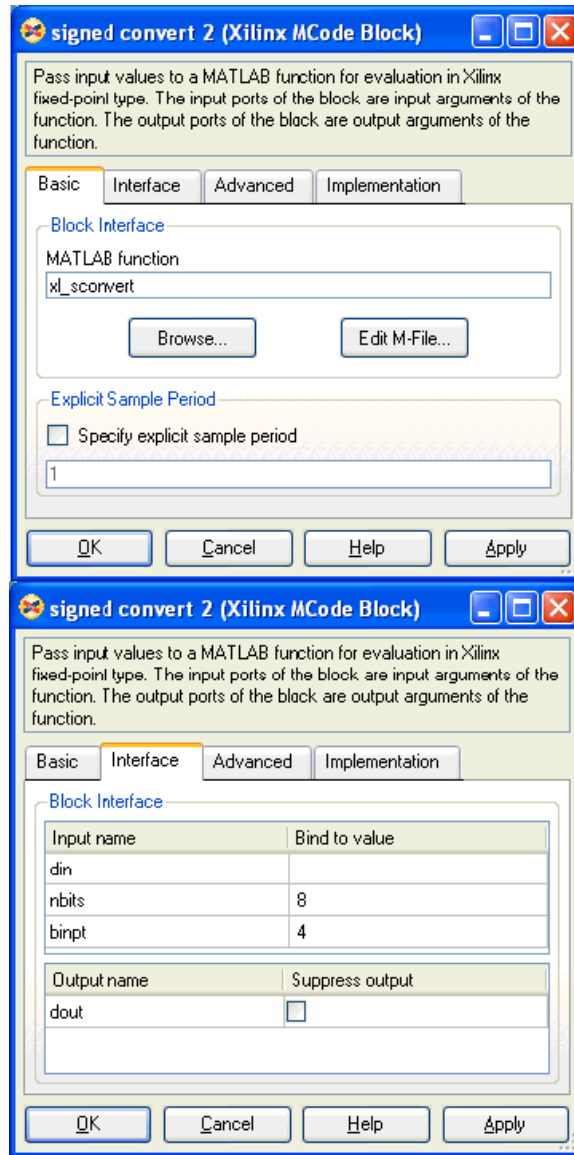


図 32: パラメーター



上の図の [interface] タブでは、M 関数の引数 [nbits] を 8 に、[binpt] を 4 に設定しています。

オプションの入力ポート

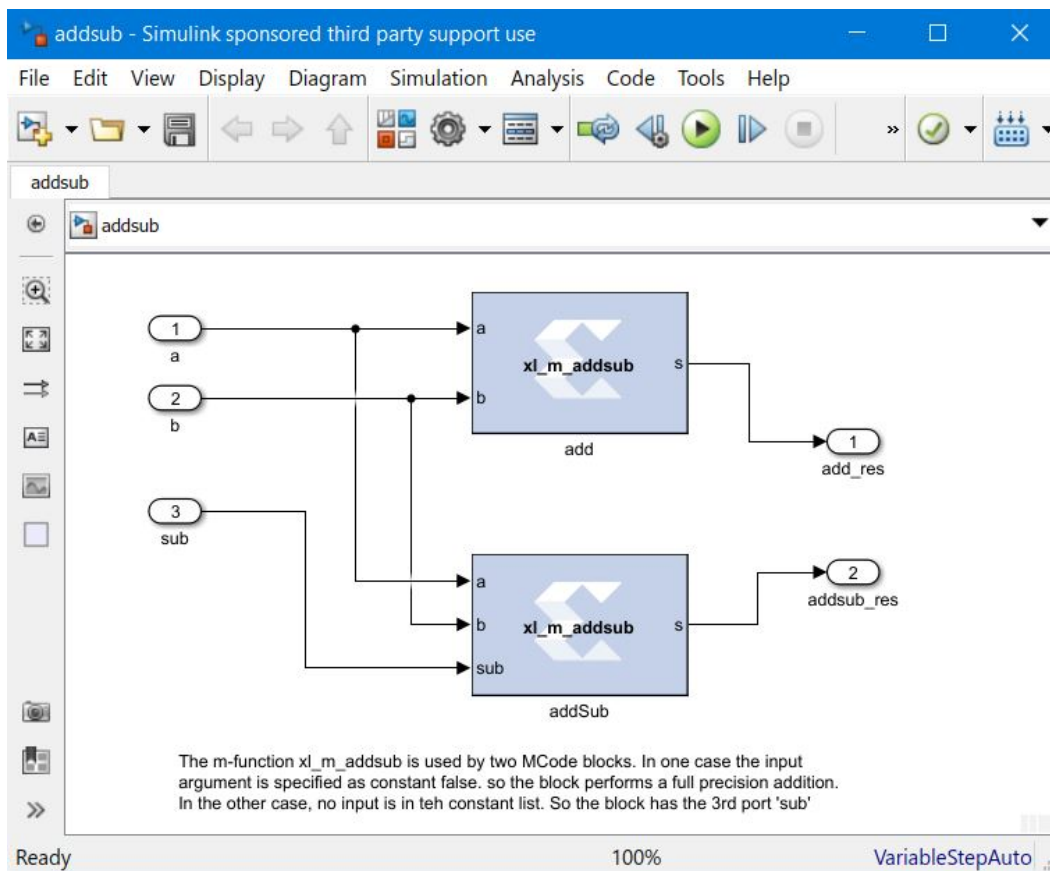
この例では、MCode ブロックにパラメーターを渡す機能を使用して、MCode ブロックでオプションの入力ポートを使用するかどうかを指定する方法を示します。

次の M コードは、xl_m_addsub.m に含まれる M 関数 xl_m_addsub を定義します。

```
function s = xl_m_addsub(a, b, sub)
    if sub
        s = a - b;
    else
        s = a + b;
    end
end
```

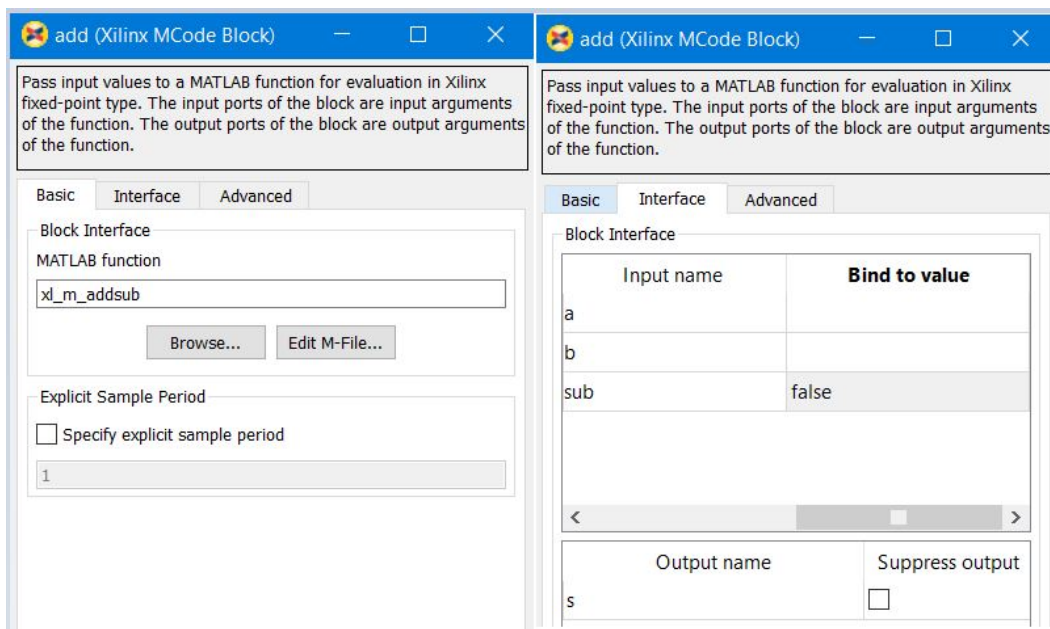
次の図に、M 関数 xl_m_addsub を使用する 2 つの MCode ブロックを含むサブシステムを示します。

図 33: M 関数を使用する 2 つの MCode ブロック



次に、add の設定を示します。

図 34: add ブロックのパラメーター ダイアログ ボックス

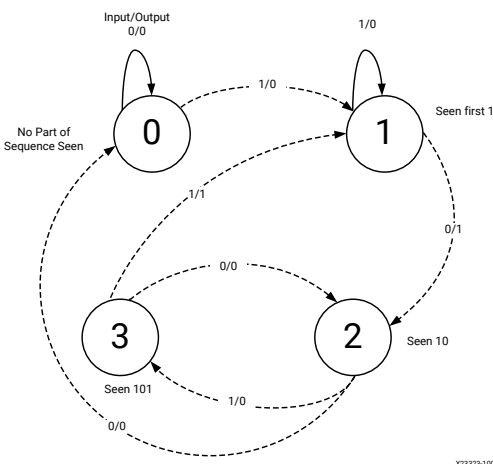


この設定により、add ブロックは入力ポート a と b を持ち、全精度加算が実行されます。addsub という MCode ブロックの入力パラメーター sub には、範囲は設定されていません。そのため、addsub ブロックは入力ポート a、b、および sub を持ち、入力ポート sub の値によって全精度加算または減算を実行します。

有限ステートマシン

この例では、MCode ブロックと内部ステート変数を使用して有限ステートマシンを作成する方法を示します。次の図に示すステートマシンでは、入力データの 1011 というパターンが検出されます。

図 35: 有限ステート マシンの図



MCode ブロックで使用される M 関数には、現在の入力のステートに基づいて次のステートを算出する遷移関数が含まれています。例 3 とは異なり、この例の M 関数では持続型ステート変数を定義して、MCode ブロックに有限ステート マシンのステートを保存します。次の M コードは、detect1011_w_state.m に含まれる M 関数 detect1011_w_state を定義します。

```
function matched = detect1011_w_state(din)
% This is the detect1011 function with states for detecting a
% pattern of 1011.

seen_none = 0; % initial state, if input is 1, switch to seen_1
seen_1 = 1;    % first 1 has been seen, if input is 0, switch
               % seen_10
seen_10 = 2;   % 10 has been detected, if input is 1, switch to
               % seen_1011
seen_101 = 3;  % now 101 is detected, is input is 1, 1011 is
               % detected and the FSM switches to seen_1

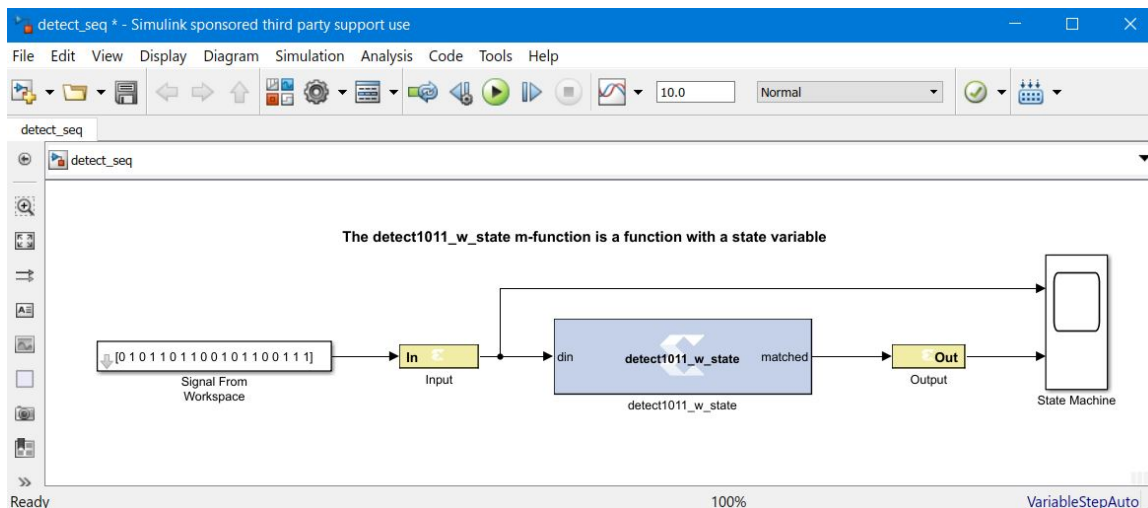
% the state is a 2-bit register
persistent state, state = xl_state(seen_none, {xlUnsigned, 2, 0});

% the default value of matched is false
matched = false;

switch state
case seen_none
    if din==1
        state = seen_1;
    else
        state = seen_none;
    end
case seen_1 % seen first 1
    if din==1
        state = seen_1;
    else
        state = seen_10;
    end
case seen_10 % seen 10
    if din==1
        state = seen_101;
    else
        % no part of sequence seen, go to seen_none
        state = seen_none;
    end
case seen_101
    if din==1
        state = seen_1;
        matched = true;
    else
        state = seen_10;
        matched = false;
    end
end
end
```

次の図に、M 関数 detect1101_w_state を使用する MCode ブロックを含むステート マシン サブシステムを示します。

図 36: MCode ブロックを含むサブシステム (コンパイル後)



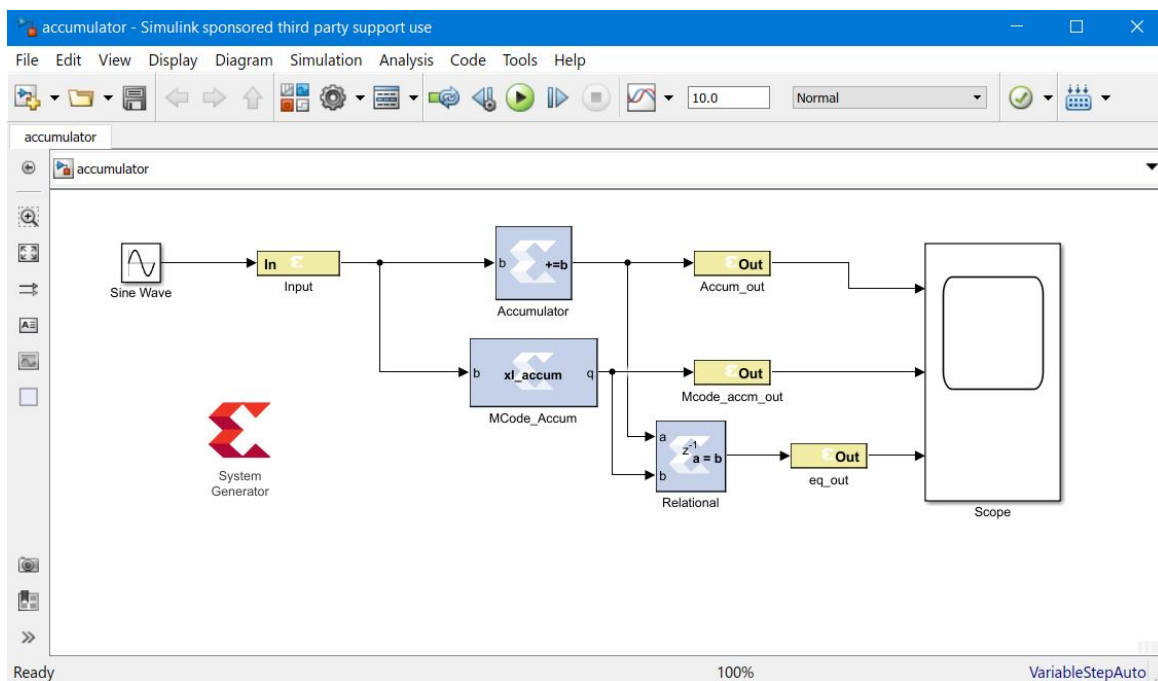
パラメーター指定可能なアキュムレータ

この例では、MCode ブロックを使用して、柔軟なインプリメンテーションを可能にするため持続性状態変数とパラメーターを定義したアキュムレータを作成する方法を示します。次の M コードは、`xl_accum.m` に含まれる M 関数 `xl_accum` を定義します。

```
function q = xl_accum(b, rst, load, en, nbits, ov, op, feed_back_down_scale)
% q = xl_accum(b, rst, nbits, ov, op, feed_back_down_scale) is
% equivalent to our Accumulator block.
binpt = xl_binpt(b);
init = 0;
precision = {xlSigned, nbits, binpt, xlTruncate, ov};
persistent s, s = xl_state(init, precision);
q = s;
if rst
    if load
        % reset from the input port
        s = b;
    else
        % reset from zero
        s = init;
    end
else
    if ~en
        else
            % if enabled, update the state
            if op==0
                s = s/feed_back_down_scale + b;
            else
                s = s/feed_back_down_scale - b;
            end
        end
    end
end
end
```

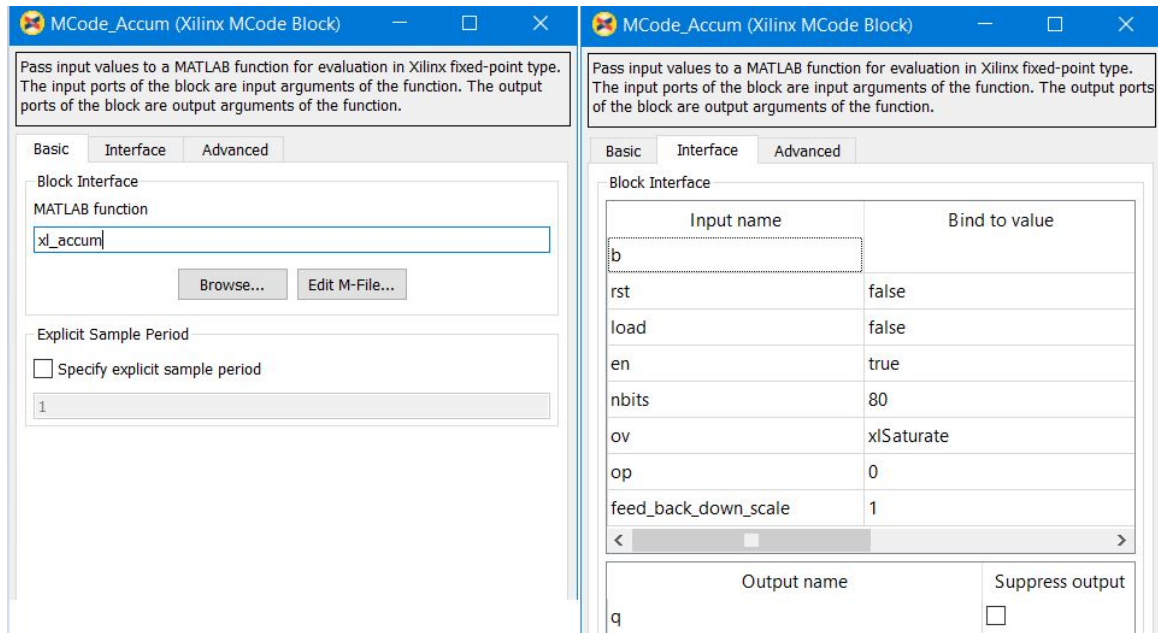
次の図に、M 関数 `xl_accum` を使用するアキュムレータ MCode ブロックを含むサブシステムを示します。MCode ブロックには、MCode Accumulator という名前が付いています。サブシステムには、比較のため、Accumulator というザイリンクス アキュムレータ ブロックも含まれています。MCode ブロックは、ザイリンクス アキュムレータ ブロックと同じ機能を持っていますが、MCode ブロックのパラメーターをマスク インターフェイス (パラメーター ダイアログ ボックスの [Interface] タブ) で指定している点が異なります。

図 37: MCode アキュムレータ



Accum_MCode1 ブロックのオプションの入力 `rst` および `load` は、[Interface] タブでディスエーブルに設定されています。次に、MCode Accumulator ブロックのパラメーター ダイアログ ボックスを示します。

図 38: MCode アキュムレータのマスク

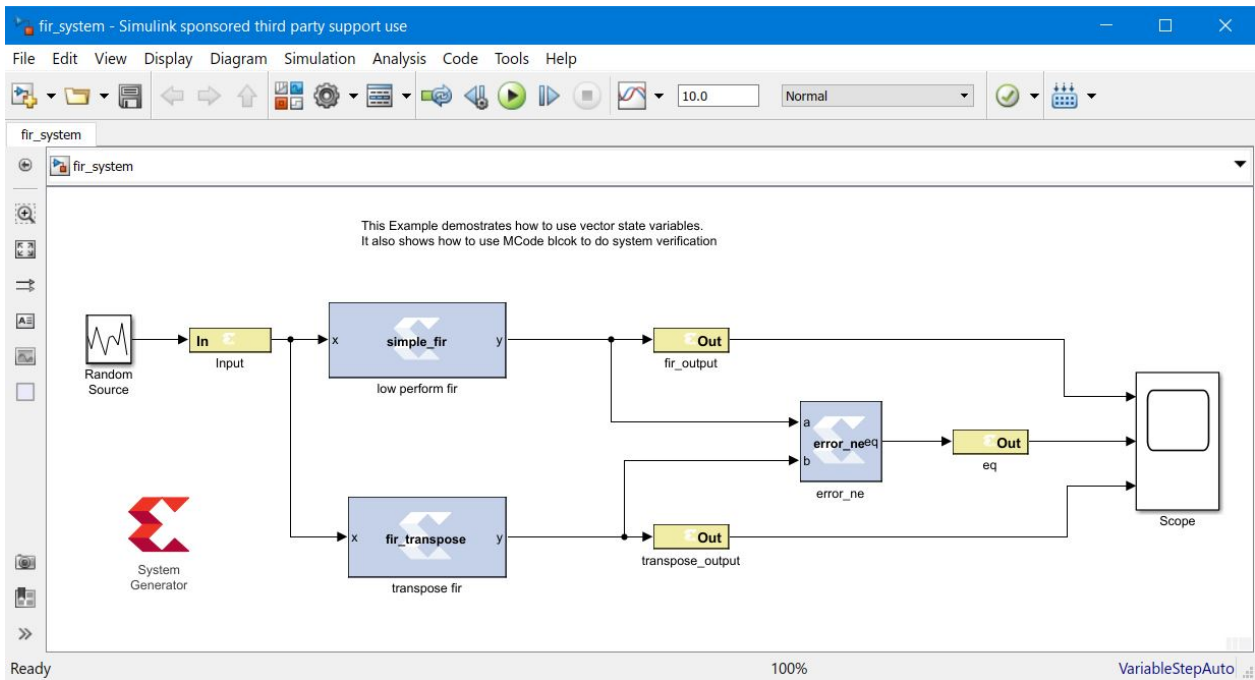


この例には、同じ M 関数を使用した MCode ブロックのアキュムレータ サブシステムがさらに 2 つ含まれていますが、異なるパラメーター設定を使用して、異なるアキュムレータを作成しています。

FIR 例とシステム検証

この例では、MCode ブロックを使用して FIR をモデリングする方法と、MCode ブロックでシステム検証を実行する方法を示します。

図 39: FIR 例



この例には、FIR ブロックが2つ含まれています。これらのブロックは MCode ブロックで定義されており、どちらも合成可能です。次に、これら2つのブロックを定義する2つの関数を示します。

```
function y = simple_fir(x, lat, coefs, len, c_nbits, c_binpt, o_nbits,
o_binpt)
    coef_prec = {xlSigned, c_nbits, c_binpt, xlRound, xlWrap};
    out_prec = {xlSigned, o_nbits, o_binpt};

    coefs_xfix = xfix(coef_prec, coefs);
    persistent coef_vec, coef_vec = xl_state(coefs_xfix, coef_prec);
    persistent x_line, x_line = xl_state(zeros(1, len-1), x);
    persistent p, p = xl_state(zeros(1, lat), out_prec, lat);

    sum = x * coef_vec(0);
    for idx = 1:len-1
        sum = sum + x_line(idx-1) * coef_vec(idx);
        sum = xfix(out_prec, sum);
    end
    y = p.back;
    p.push_front_pop_back(sum);
    x_line.push_front_pop_back(x);
function y = fir_transpose(x, lat, coefs, len, c_nbits, c_binpt, o_nbits,
o_binpt)
    coef_prec = {xlSigned, c_nbits, c_binpt, xlRound, xlWrap};
    out_prec = {xlSigned, o_nbits, o_binpt};
    coefs_xfix = xfix(coef_prec, coefs);
    persistent coef_vec, coef_vec = xl_state(coefs_xfix, coef_prec);
    persistent reg_line, reg_line = xl_state(zeros(1, len), out_prec);
    if lat <= 0
        error('latency must be at least 1');
    end
    lat = lat - 1;
    persistent dly,
    if lat <= 0
```

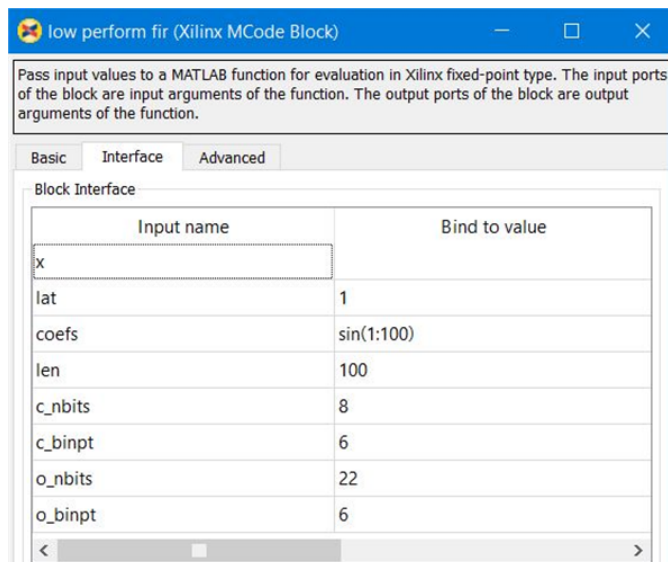
```

y = reg_line.back;
else
    dly = xl_state(zeros(1, lat), out_prec, lat);
    y = dly.back;
    dly.push_front_pop_back(reg_line.back);
end
for idx = len-1:-1:1
    reg_line(idx) = reg_line(idx - 1) + coef_vec(len - idx - 1) * x;
end
reg_line(0) = coef_vec(len - 1) * x;

```

パラメーターは、次のように設定されます。

図 40: パラメーター



2つのブロックの機能が同一であることを検証するため、MCode ブロックをもう 1つ使用して 2つのブロックの出力を比較します。2つの出力が等しくない場合は、エラー チェック ブロックによりエラーがレポートされます。エラー チェックは、次の関数により実行されます。

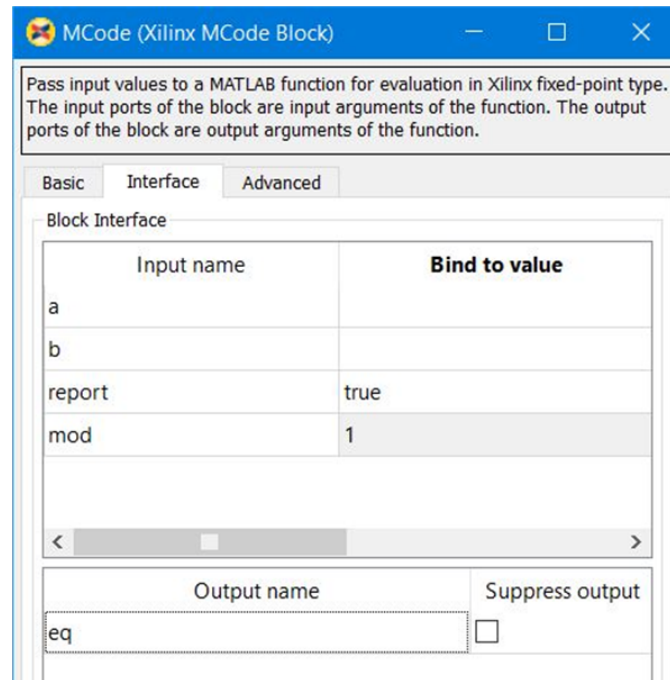
```

function eq = error_ne(a, b, report, mod)
    persistent cnt, cnt = xl_state(0, {xlUnsigned, 16, 0});
    switch mod
        case 1
            eq = a==b;
        case 2
            eq = isnan(a) || isnan(b) || a == b;
        case 3
            eq = ~isnan(a) && ~isnan(b) && a == b;
        otherwise
            eq = false;
            error(['wrong value of mode ', num2str(mod)]);
    end
    if report
        if ~eq
            error(['two inputs are not equal at time ', num2str(cnt)]);
        end
    end
    cnt = cnt + 1;
end

```

このブロックは、次のように設定されます。

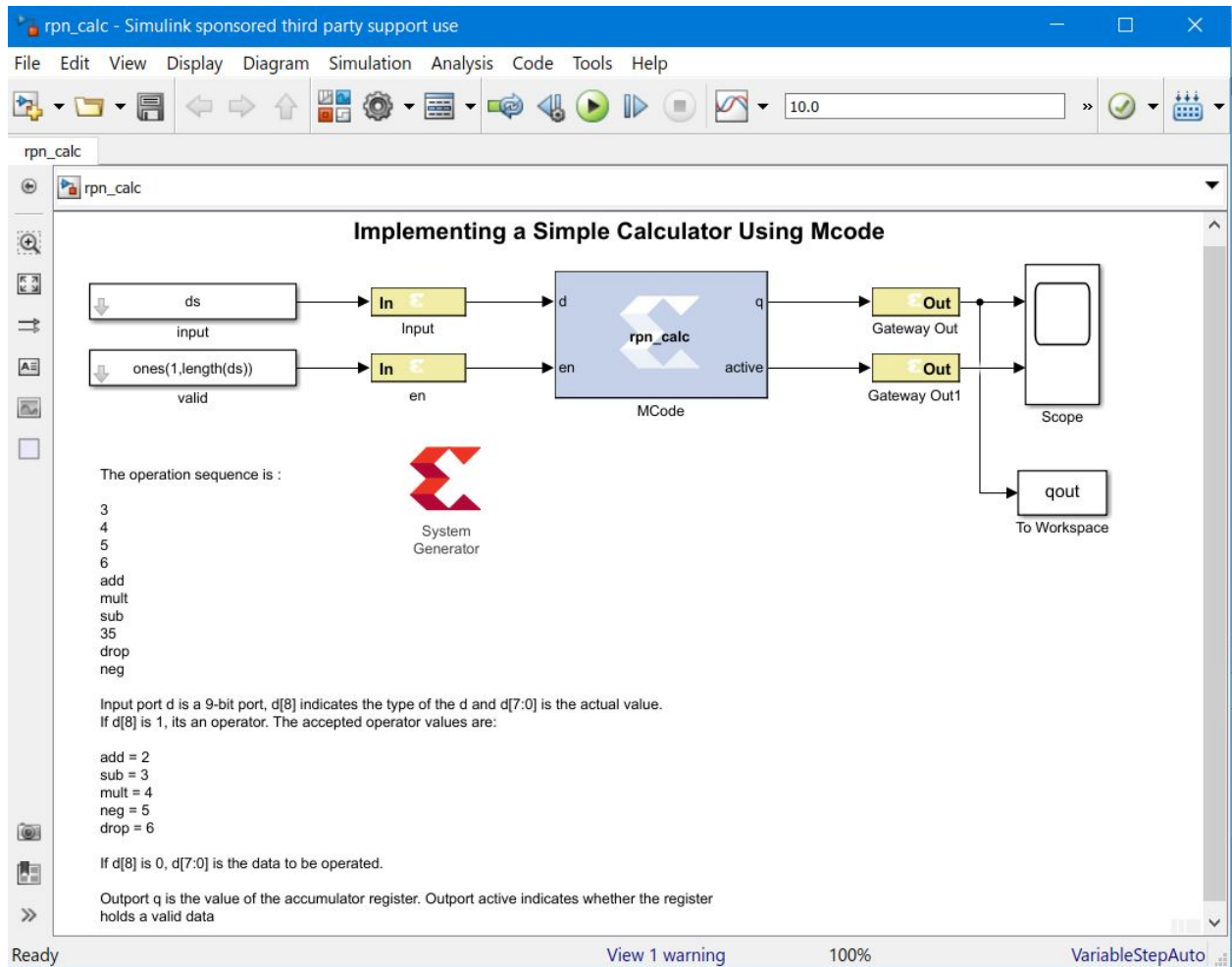
図 41: ブロックの設定



RPN カリキュレーター

この例では、MCode ブロックを使用して、スタック マシンである RPN カリキュレーターを構築する方法を示します。このブロックは、合成可能です。

図 42: RPN カリキュレーター



次の関数は、RPN カリキュレーターを定義します。

```

function [q, active] = rpn_calc(d, rst, en)
    d_nbits = xl_nbits(d);
    % the first bit indicates whether it's a data or operator
    is_oper = xl_slice(d, d_nbits-1, d_nbits-1)==1;
    din = xl_force(xl_slice(d, d_nbits-2, 0), xlSigned, 0);
    % the lower 3 bits are operator
    op = xl_slice(d, 2, 0);
    % acc the A register
    persistent acc, acc = xl_state(0, din);
    % the stack is implemented with a RAM and
    % an up-down counter
    persistent mem, mem = xl_state(zeros(1, 64), din);
    persistent acc_active, acc_active = xl_state(false, {xlBoolean});
    persistent stack_active, stack_active = xl_state(false, ...
                                                {xlBoolean});

    stack_pt_prec = {xlUnsigned, 5, 0};
    persistent stack_pt, stack_pt = xl_state(0, {xlUnsigned, 5, 0});
    % when en is true, it's action
    OP_ADD = 2;
    OP_SUB = 3;

```

```

OP_MULT = 4;
OP_NEG = 5;
OP_DROP = 6;
q = acc;
active = acc_active;
if rst
    acc = 0;
    acc_active = false;
    stack_pt = 0;
elseif en
    if ~is_oper
        % enter data, push
        if acc_active
            stack_pt = xfix(stack_pt_prec, stack_pt + 1);
            mem(stack_pt) = acc;
            stack_active = true;
        else
            acc_active = true;
        end
        acc = din;
    else
        if op == OP_NEG
            % unary op, no stack op
            acc = -acc;
        elseif stack_active
            b = mem(stack_pt);
            switch double(op)
                case OP_ADD
                    acc = acc + b;
                case OP_SUB
                    acc = b - acc;
                case OP_MULT
                    acc = acc * b;
                case OP_DROP
                    acc = b;
            end
            stack_pt = stack_pt - 1;
        elseif acc_active
            acc_active = false;
            acc = 0;
        end
    end
end
stack_active = stack_pt ~= 0;

```

disp 関数の例

次の MCode 関数は、disp 関数を使用して変数値を指定する方法を示します。

```

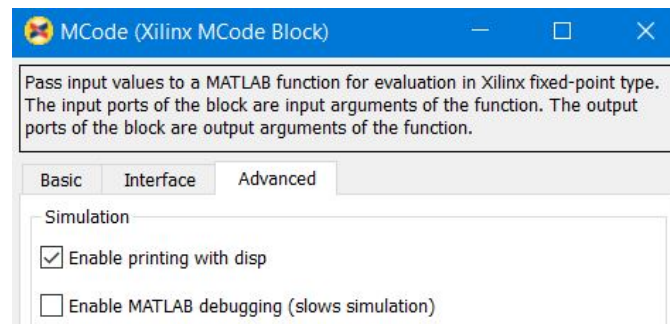
function x = testdisp(a, b)
    persistent dly, dly = xl_state(zeros(1, 8), a);
    persistent rom, rom = xl_state([3, 2, 1, 0], a);
    disp('Hello World!');
    disp(['num2str(dly) is ', num2str(dly)]);
    disp('disp(dly) is ');
    disp(dly);
    disp('disp(rom) is ');
    disp(rom);
    a2 = dly.back;
    dly.push_front_pop_back(a);
    x = a + b;

```

```
disp(['a = ', num2str(a), ', ', ' ', ...
      'b = ', num2str(b), ', ', ' ', ...
      'x = ', num2str(x)]);
disp(num2str(true));
disp('disp(10) is');
disp(10);
disp('disp(-10) is');
disp(-10);
disp('disp(a) is ');
disp(a);
disp('disp(a == b)');
disp(a==b);
```

[Enable printing with disp] をオンにします。

図 43: [Enable printing with disp] をオン



次に、最初のシミュレーション ステップで MATLAB のコマンド ウィンドウに表示される行を示します。

```
mcode_block_disp/MCode (Simulink time: 0.000000, FPGA clock: 0)
Hello World!
num2str(dly) is [0.000000, 0.000000, 0.000000, 0.000000, 0.000000,
0.000000, 0.000000, 0.000000]
disp(dly) is
type: Fix_11_7,
maxlen: 8,
length: 8,
0: binary 0000.00000000, double 0.000000,
1: binary 0000.00000000, double 0.000000,
2: binary 0000.00000000, double 0.000000,
3: binary 0000.00000000, double 0.000000,
4: binary 0000.00000000, double 0.000000,
5: binary 0000.00000000, double 0.000000,
6: binary 0000.00000000, double 0.000000,
7: binary 0000.00000000, double 0.000000,
disp(rom) is
type: Fix_11_7,
maxlen: 4,
length: 4,
0: binary 0011.00000000, double 3.0,
1: binary 0010.00000000, double 2.0,
2: binary 0001.00000000, double 1.0,
3: binary 0000.00000000, double 0.0,
a = 0.000000, b = 0.000000, x = 0.000000
1
disp(10) is
type: UFix_4_0, binary: 1010, double: 10.0
disp(-10) is
```

```
type: Fix_5_0, binary: 10110, double: -10.0
disp(a) is
type: Fix_11_7, binary: 0000.0000000, double: 0.000000
disp(a == b)
type: Bool, binary: 1, double: 1
```

System Generator デザインの大型システムへのインポート

System Generator デザインは、大型 HDL デザインに組み込まれることがよくあります。このセクションでは、2 つの System Generator デザインを大型デザインに組み込む方法、System Generator で作成した VHDL をシステム全体のシミュレーション モデルに組み込む方法を説明します。

HDL ネットリスト コンパイル

[System Generator] ダイアログ ボックスでコンパイル タイプに [HDL Netlist] を選択すると、デザインをインプリメントするための HDL とその他の関連ファイルが生成されます。また、Vivado シミュレータを使用したデザインのシミュレーションや Vivado 合成を使用した論理合成などのダウンストリーム処理を簡略化する補助ファイルも生成されます。詳細は、[第 8 章: System Generator のコンパイル タイプ](#)を参照してください。

デザイン ルールの統合

System Generator モデルを大型デザインに組み込む場合、次の 2 つのデザイン ルールに従う必要があります。

- ルール 1: Gateway ブロックまたは System Generator トークンで IOB/CLK ロケーション制約を指定しないでください。

また、IOB タイミング制約は none に設定する必要があります。

- ルール 2: System Generator デザインに含まれる I/O ポートを最上位デザインのポートにする必要がある場合、適切なバッファを最上位の HDL コードにインスタンス化する必要があります。

コンフィギュラブル サブシステムと System Generator

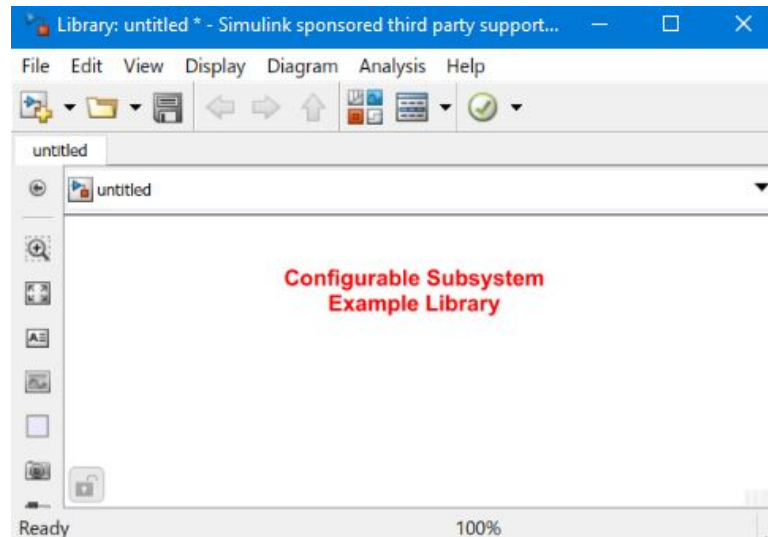
コンフィギュラブル サブシステムは、Simulink の標準パーツとして使用可能なブロックであり、基本ブロックを複数指定できるブロックです。各ブロックはそれぞれ可能なインプリメンテーションであり、どのインプリメンテーションを使用するかを自由に選択できます。たとえば、System Generator で汎用 FIR フィルターをコンフィギュラブル サブシステムとして指定し、そのサブシステムの基になるブロックとして特定の FIR フィルターを複数指定します。高速だがハードウェア リソースを多く必要とするフィルター、比較的低速だがハードウェア リソースをあまり必要としないフィルターなどを指定できます。フィルターの選択を切り替えることにより、ハードウェア コストまたはスピードを優先した場合の動作を調べることができます。

コンフィギュラブル サブシステムの定義

コンフィギュラブル サブシステムを定義するには、Simulink® ライブラリを作成します。コンフィギュラブル サブシステムの基になるブロックは、このライブラリで管理されます。ライブラリを作成するには、次の手順に従います。

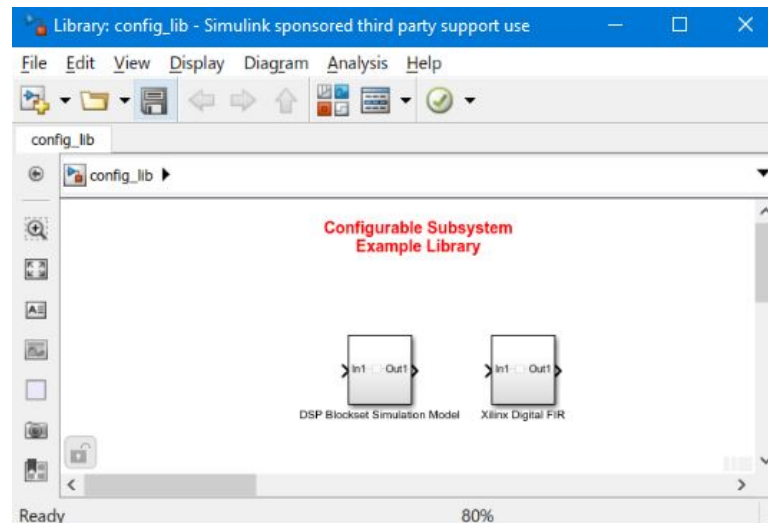
1. 空のライブラリを作成します。

図 44: 新しい空のライブラリ



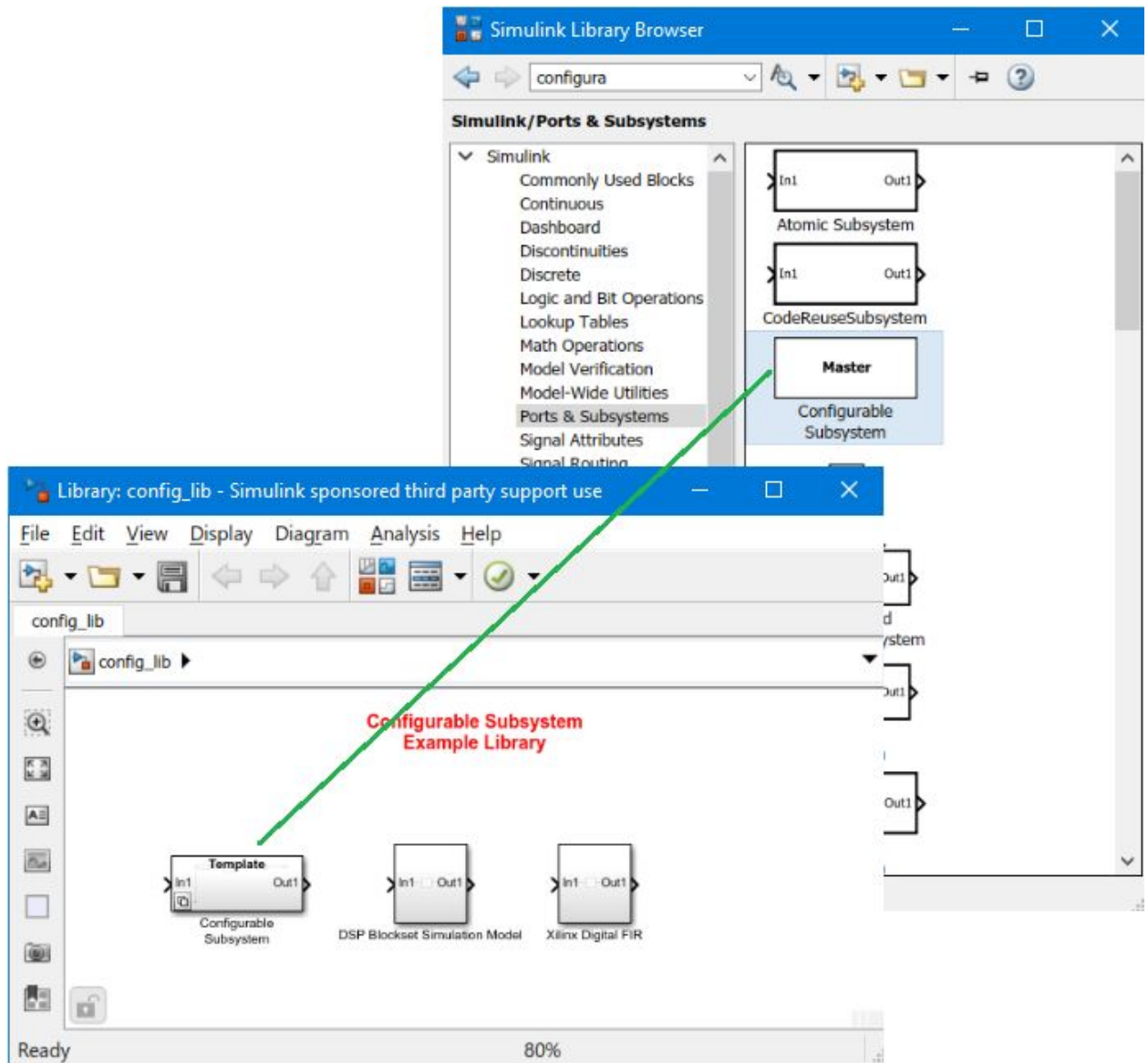
2. 作成したライブラリに基になるブロックを追加します。

図 45: 基になるブロックの追加



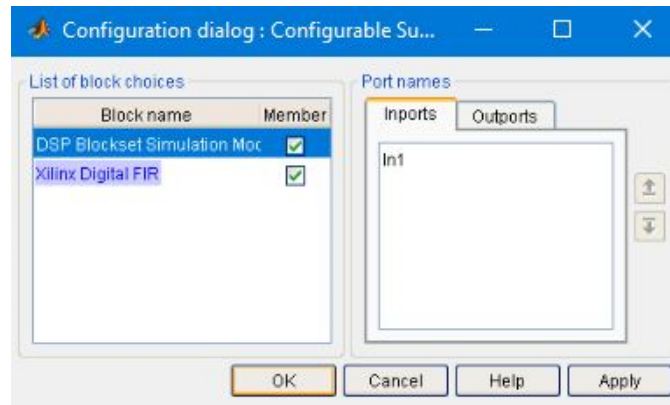
3. テンプレート ブロックをライブラリにドラッグします。テンプレートは、Simulink ライブラリ ブラウザーの [Simulink] → [Ports & Subsystems] → [Configurable Subsystem] にあります。

図 46: テンプレート



4. 必要に応じて、テンプレート ブロックの名前を変更します。
5. ライブラリを保存します。
6. ライブラリのテンプレートをダブルクリックして開きます。
7. テンプレートの設定ダイアログ ボックスで、インプリメントするブロックのチェック ボックスをオンにします。

図 47: インプリメントするブロックのチェック ボックスをオン



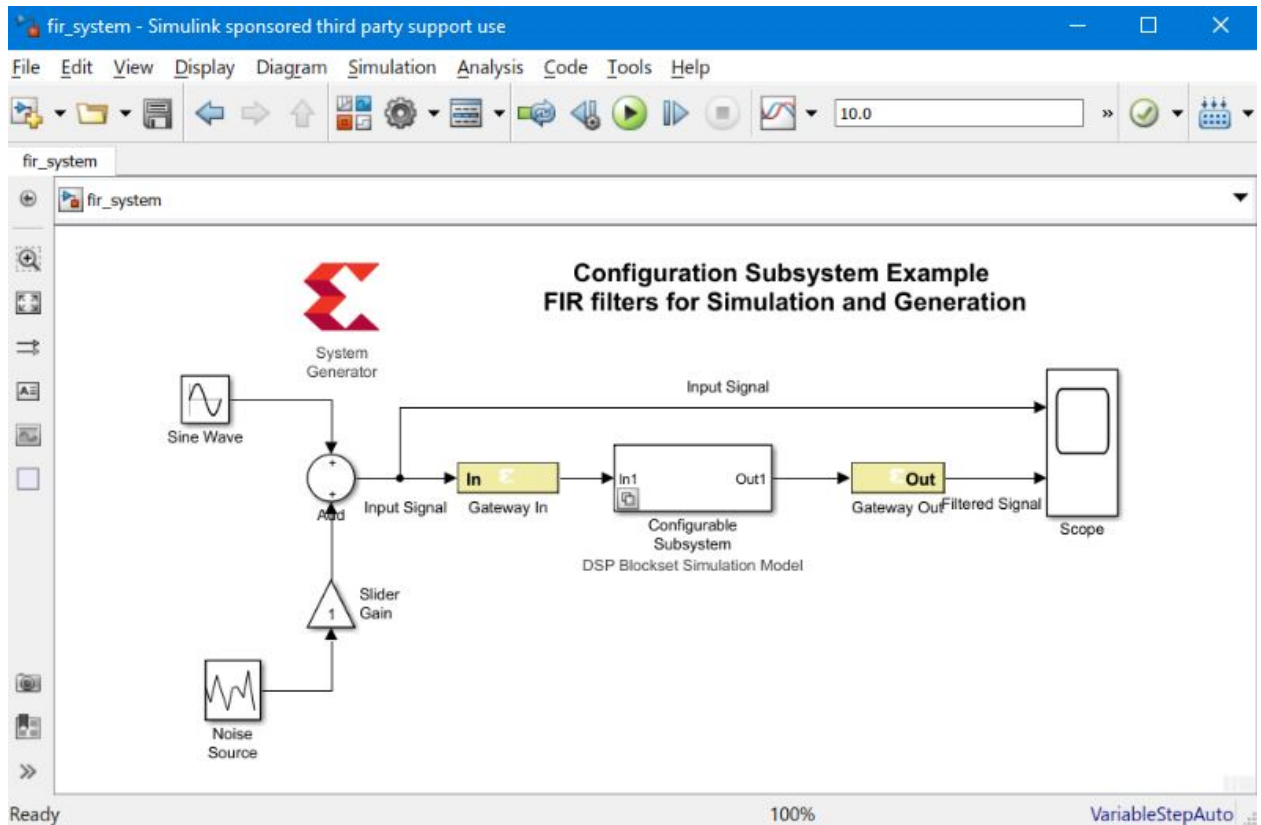
8. [OK] をクリックし、その後ライブラリを保存します。

コンフィギュラブル サブシステムの使用

デザインでコンフィギュラブル サブシステムを使用するには、次の手順に従います。

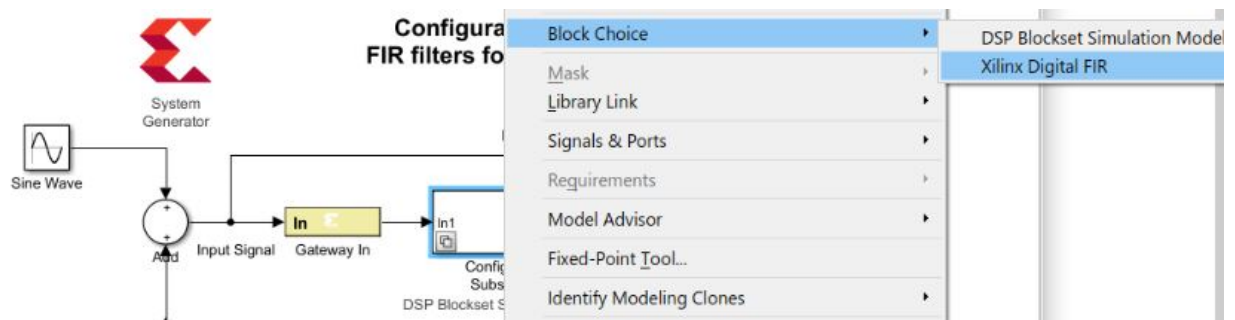
1. コンフィギュラブル サブシステムを定義するライブラリを作成します。
2. ライブラリを開きます。
3. ライブラリからテンプレートをデザインの適切な位置にドラッグします。
4. ドラッグしたテンプレートが、コンフィギュラブル サブシステムのインスタンスになります。

図 48: コンフィギュラブル サブシステム



5. インスタンスを右クリックして [Block Choice] をクリックし、インスタンスのインプリメンテーションとして使用するブロックをクリックします。

図 49: ブロックの選択

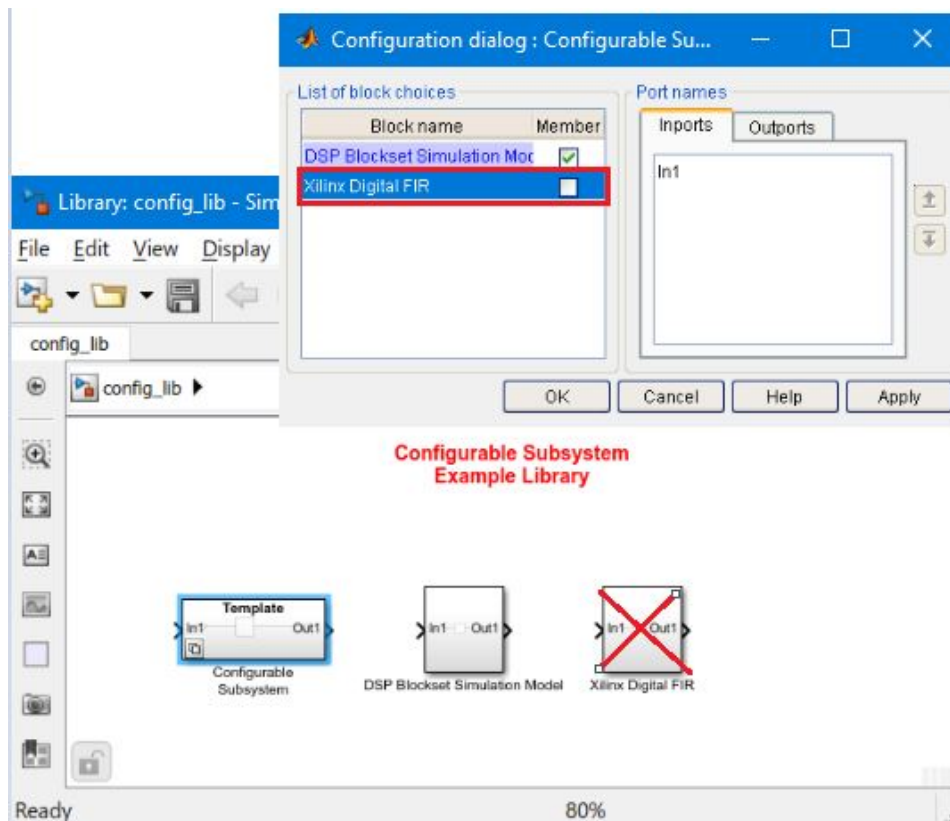


コンフィギュラブル サブシステムからのブロックの削除

コンフィギュラブル サブシステムからブロックを削除するには、次の手順に従います。

1. コンフィギュラブル サブシステムのライブラリを開き、ライブラリのロックを解除します。
2. テンプレートをダブルクリックし、削除するブロックのチェック ボックスをオフにします。
3. [OK] をクリックし、その後ブロックを削除します。

図 50: ブロックの削除



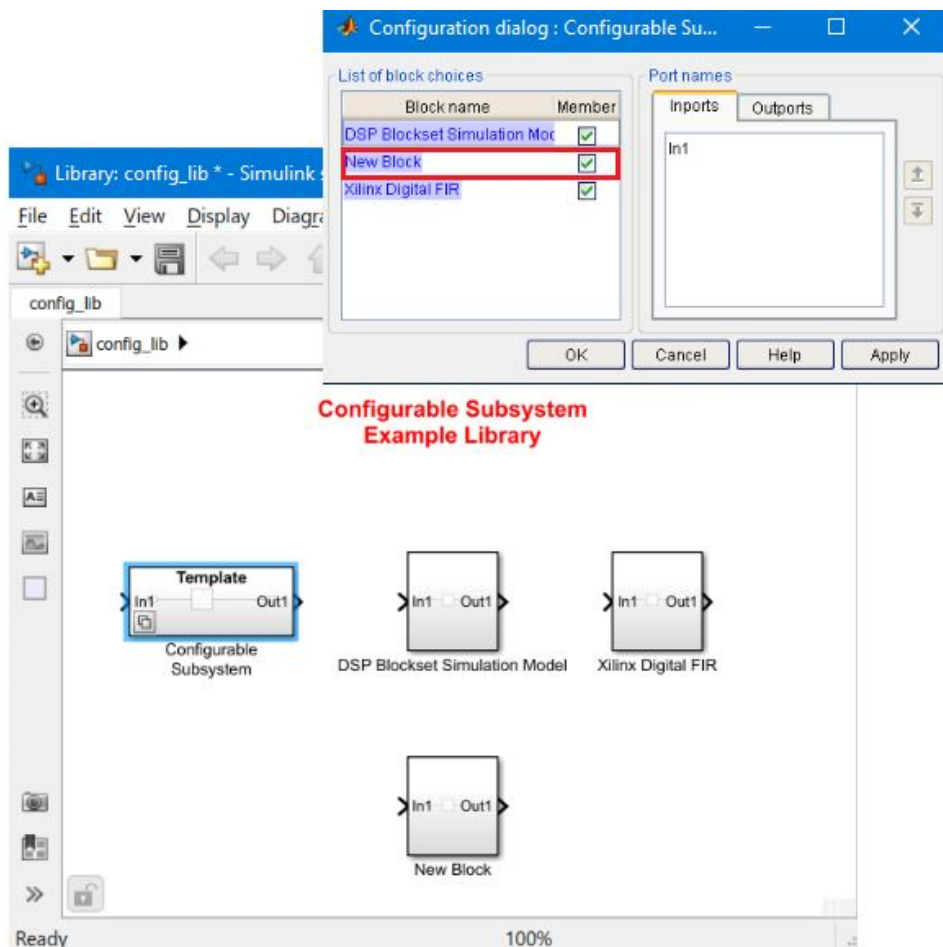
4. ライブラリを保存します。
5. [Ctrl + D] キーを押し、デザインをコンパイルします。
6. 必要に応じて、コンフィギュラブル サブシステムの各インスタンスの設定をアップデートします。

コンフィギュラブル サブシステムへのブロックの追加

コンフィギュラブル サブシステムにブロックを追加するには、次の手順に従います。

1. コンフィギュラブル サブシステムのライブラリを開き、ライブラリのロックを解除します。
2. ブロックをライブラリにドラッグします。
3. テンプレートをダブルクリックし、追加したブロックのチェック ボックスをオンにします。

図 51: ブロックの追加



4. [OK] をクリックし、その後ライブラリを保存します。
5. [Ctrl-D] キーを押し、デザインをコンパイルします。
6. 必要に応じて、コンフィギュラブル サブシステムの各インスタンスの設定をアップデートします。

FPGA デザインのパフォーマンスを向上するためのヒント

バックエンド インプリメンテーション ツールですべての最適化を実行しようとする、次の理由からタイミング クロージャを達成できない可能性があります。

- System Generator デザインに FIR Compiler および FFT のようなより複雑な IP ブロックが生成されます。これらは高度に最適化されたネットリストとして合成ツールおよびインプリメンテーション ツールに供給されるので、それ以上の最適化は実行できないことがあります。
- System Generator ネットリストで多数のプリミティブ (レジスタ、BRAM、DSP48E1 など) がインスタンス化された HDL コードが生成されます。これらのエレメントは、合成ツールではそれほど最適化できません。

次に、インプリメンテーション プロセスを開始する前に System Generator でデザインのパフォーマンスを向上するために可能な操作を示します。

- 各ブロックのパラメーター ダイアログ ボックスに含まれている「Hardware Notes」
- デザインの入力と出力にレジスタを付ける
- パイプライン レジスタを挿入する
- [Saturate] および [Round] オプションは必要な場合以外は使用しない
- すべての Gateway ブロックでデータ レート オプションを設定
- 最高のパフォーマンスを得るためのパイプライン処理
- その他の方法

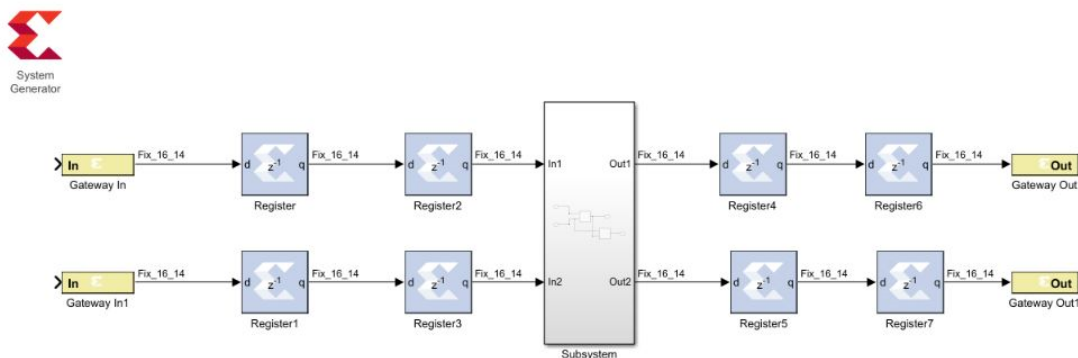
各ブロックのパラメーター ダイアログ ボックスに含まれている「Hardware Notes」

ブロックのパラメーター ダイアログ ボックスに含まれている「Hardware Notes」を読むようにしてください。サイリンクス ブロックセット ライブラリの多くのブロックでは、最も効率の良いハードウェア インプリメンテーションを達成する方法が記載されています。たとえば、Scale ブロックにはこのブロックにハードウェア コストがかからないことが記述されていますが、同じ目的で使用するのある Shift ブロックでは場合によってハードウェアが使用されることが記述されています。

デザインの入力と出力にレジスタを付ける

デザインの入力と出力にレジスタを付けます。次の図に示すように、レジスタを付けるには、Gateway In ブロックの後および Gateway Out ブロックの前に、レイテンシ 1 の Delay ブロックまたは Register ブロックを 1 つまたは複数配置します。Register ブロックの機能のいずれかを追加すると、追加のハードウェア リソースが必要になります。

図 52: レジスタの入力および出力

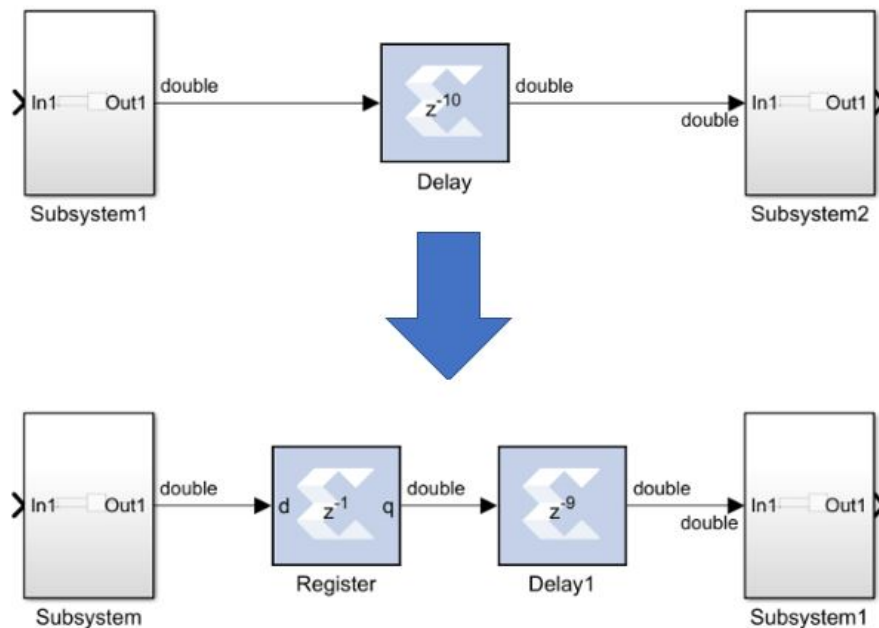


I/O に 2 つのレジスタを付けると、有益な場合があります。この場合、Register ブロックを 2 つインスタンス化するか、レイテンシが 1 の Delay ブロックを 2 つインスタンス化します。このようにすると、1 つのレジスタが IOB 内に配置され、もう 1 つのレジスタが FPGA のロジックの横に配置されます。Delay ブロックのレイテンシを 2 にしても、レイテンシ 2 のブロックが SRL32 を使用してインプリメントされ、IOB 内に配置されないため、同じ結果にはなりません。

パイプライン レジスタを挿入する

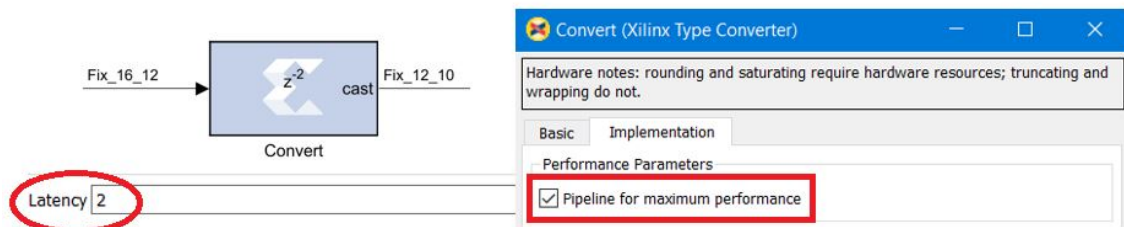
可能な限り、パイプライン レジスタを挿入してください。パイプラインは、Delay ブロックを使用して効率的にインプリメントできます (SRL32 プリミティブが使用される)。レジスタに初期値を指定する必要がある場合は、Register ブロックを使用してください。また、SRL32 の入力パスがタイミングを満たさない場合は、関連する Delay ブロックの前に Register ブロックを配置し、Delay ブロックのレイテンシを 1 に削減します。これにより、配線ツールの柔軟性がさらに増し、Register および Delay ブロック (SRL + Register) を離して配置して、このパスの配線遅延のマージンを最大にできます。

図 53: パイプライン レジスタ



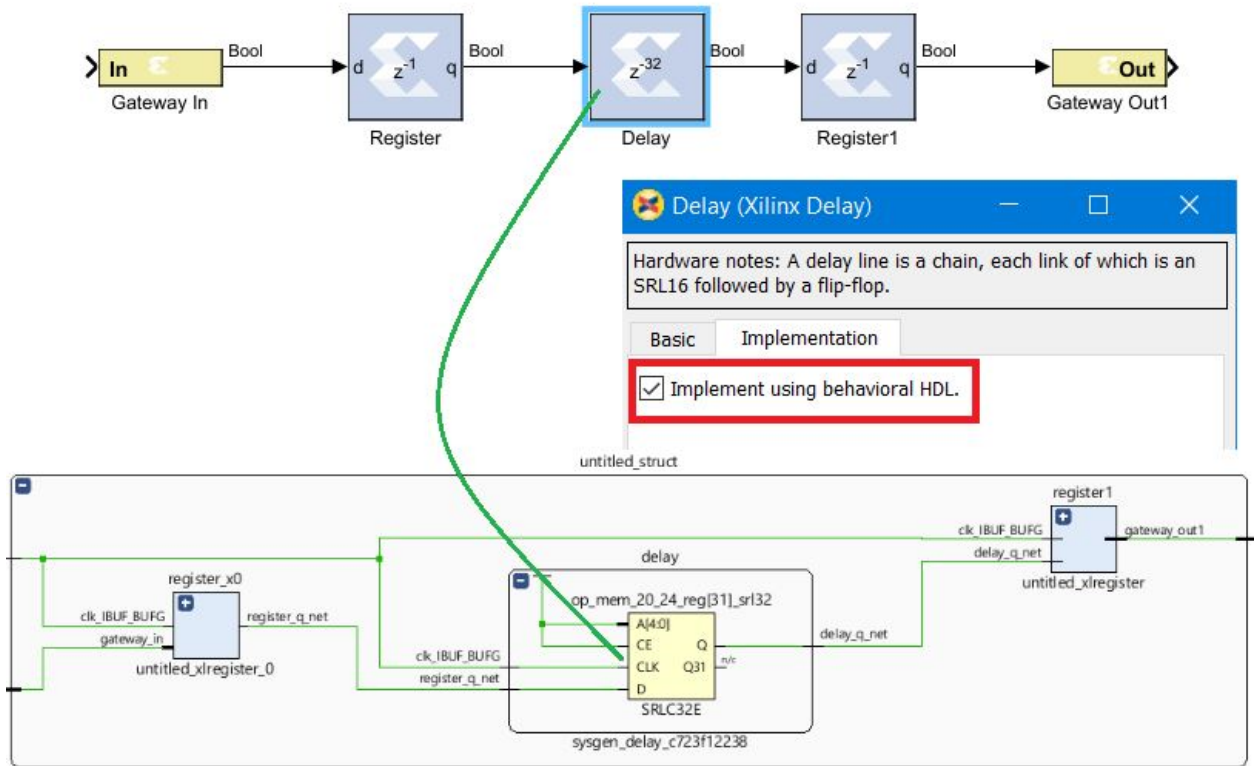
次の図に示すように、Convert ブロックをエンベデッド レジスタ段とパイプライン接続すると、最大パフォーマンスを達成できます。

図 54: Convert ブロック



ザイリンクス ブロックでより効率的なインプリメンテーションを達成するには、[Implement using behavioral HDL] オプションをオンにします。次に示すように、Delay ブロックの遅延が 32 以上の場合、ザイリンクス合成により 1 つの LUT にマップされる SRLC32E (32 ビットのシフト レジスタ) が推論されます。

図 55: ビヘイビア HDL を使用したインプリメント



ブロック RAM (BRAM) には、内部出力レジスタを使用してください。これはレイテンシを 1 (デフォルト) から 2 に設定すると使用できます。これにより、ブロック RAM 出力レジスタがイネーブルになります。

DSP48E1 を使用する場合は、入力、出力、および内部レジスタを、FIFO を使用する場合は、エンベデッド レジスタ オプションを使用します。また、すべての高位 IP ブロックのパイプライン オプションを確認します。

[Saturate] および [Round] オプションは必要な場合以外は使用しない

これらのオプションを使用すると、リソースが多く使用され、パフォーマンスが低下します。必要な場合にのみ使用してください。たとえば、Reinterpret ブロックの場合、どのロジックも失われることはありません。Convert (cast) ブロックの場合、量子化 (Quantization) が切り捨て (Truncate) でオーバーフロー (Overflow) が折り返し (Wrap) に設定されていれば、どのロジックも失うことはありません。データ型に丸め (Rounding) および飽和 (Saturation) オプションを使用する必要がある場合は、Convert ブロックをエンベデッド レジスタ段とパイプライン接続します。DSP48E1 を使用する場合、丸めは DSP48E1 内で実行できます。

すべての Gateway ブロックでデータ レート オプションを設定

[Gateway In] ブロックと [Gateway Out] ブロックのパラメーター ダイアログ ボックスで、[IOB timing constraint] に対して [Data Rate] をオンにします。[Data Rate] をオンにすると、IOB が動作するデータ レートに制約されます。このレートは、System Generator トークンの [Simulink system period (sec)] の値と、デザイン内のその他のサンプル周期に対する Gateway ブロックのサンプル レートによって決定されます。

最高のパフォーマンスを得るためのパイプライン処理

ザイリンクス LogiCORE™ IP を使用する System Generator ブロックでは、コアの外に少なくとも 1 つのレジスタを配置するのがツールのデフォルト動作です。レイテンシ値がコアの最適な値よりも大きい場合は、コアの内部に最適な数のパイプライン レジスタが配置され、残りのレジスタはコアの外に配置されます。

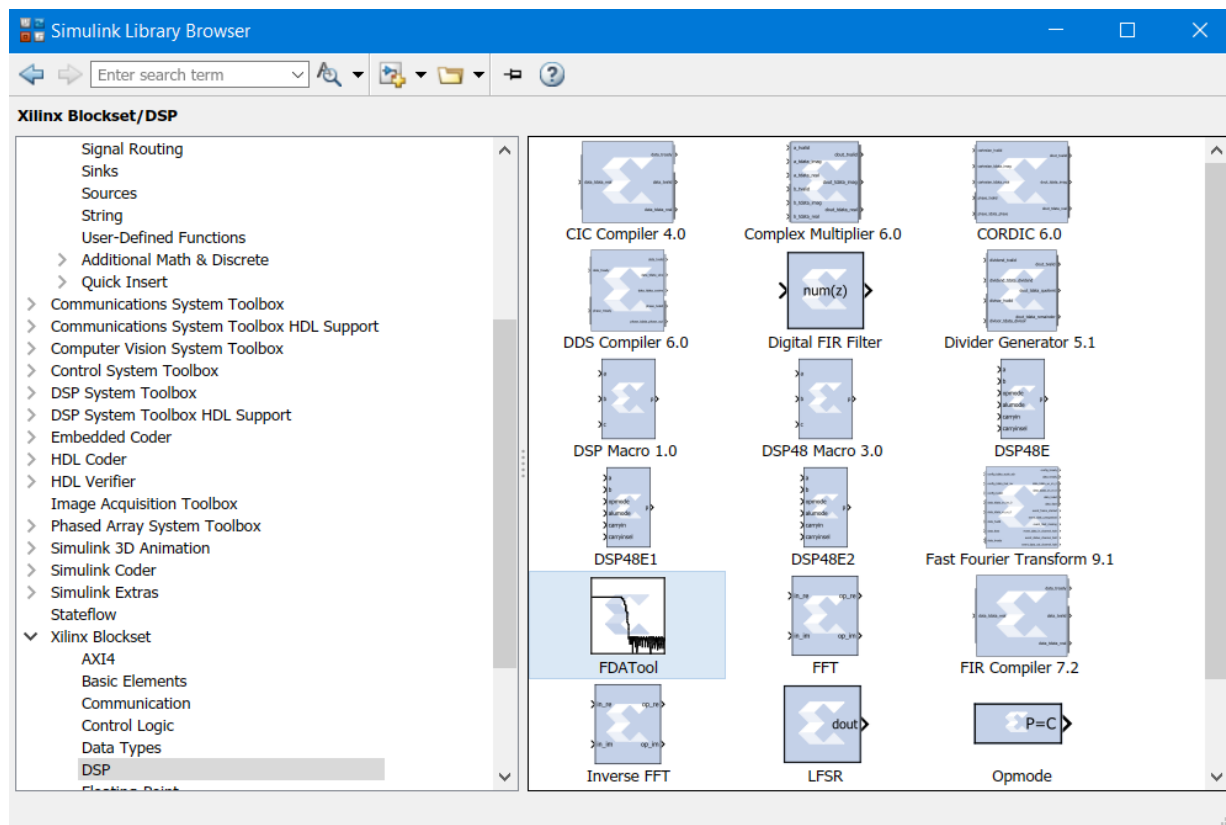
その他の方法

- ソース デザインを変更する
 - 追加のパイプラインを使用する
ブロック RAM および DSP48 内の出力レジスタおよびパイプライン レジスタを使用します。
 - 関数を並列実行する
関数を遅いクロック レートで並列実行します。
 - リタイミング手法を使用する
既存のレジスタを組み合わせロジックを介して移動します。
 - 可能な場合にハード コアを使用する
分散 RAM の代わりにブロック RAM を使用します。
 - 関数に異なる設計手法を使用する
- デザインを過剰に制約しない
デザインを過剰に制約せず、適宜 Up/Down Sample ブロックを使用します。
- クリティカルなデザイン モジュールの周波数を低減する
- インプリメンテーション ツールで調整する
 - さまざまな合成オプションを試してみる
 - クリティカル モジュールをフロアプランする

FDATool を使用したデジタル フィルター アプリケーション

FDATool ブロックを使用してフィルターの次数および係数を定義し、ザイリンクス ブロックセットを使用してフィルターをインプリメントします。ザイリンクス ブロックセットの DSP ライブラリには、FDATool ブロックが含まれます。

図 56: FDATool ブロック



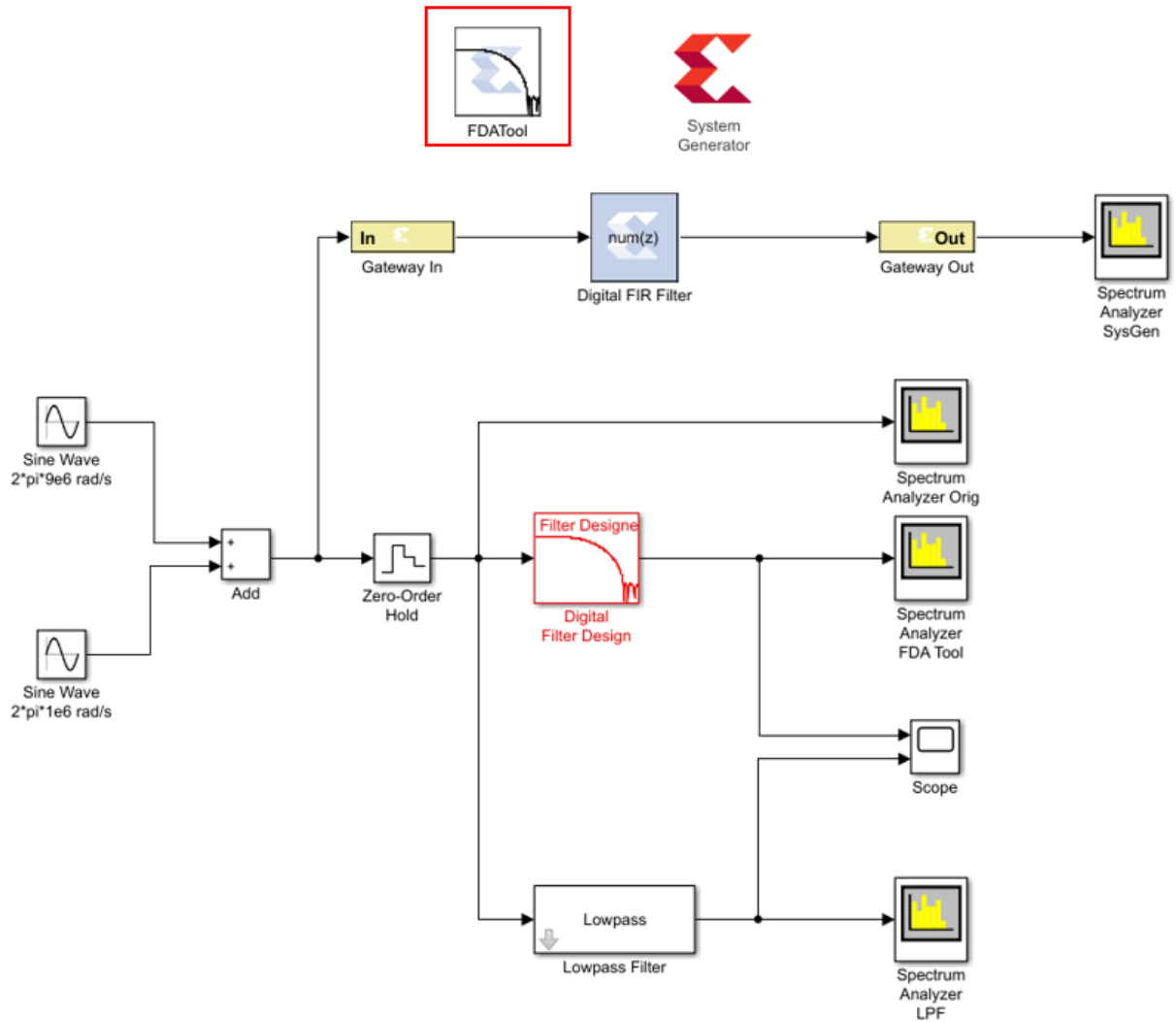
次に示す単純な System Generator モデルは、FDATool および Digital FIR Filter ブロックを使用した標準 FIR フィルターです。

このデザインは 2 つのサイン波ソースを使用し、これらを加算して 2 つのローパス フィルターを介して個別に渡されます。

- 最初のフィルターは、ザイリンクス ブロックセットを使用してインプリメントできます。これは、Digital FIR Filter ブロックを使用してインプリメントされるローパス フィルターです。
- 2 つ目のフィルターは参照フィルターで、Direct Form FIR 構造を使用してインプリメントされるローパス フィルターです。

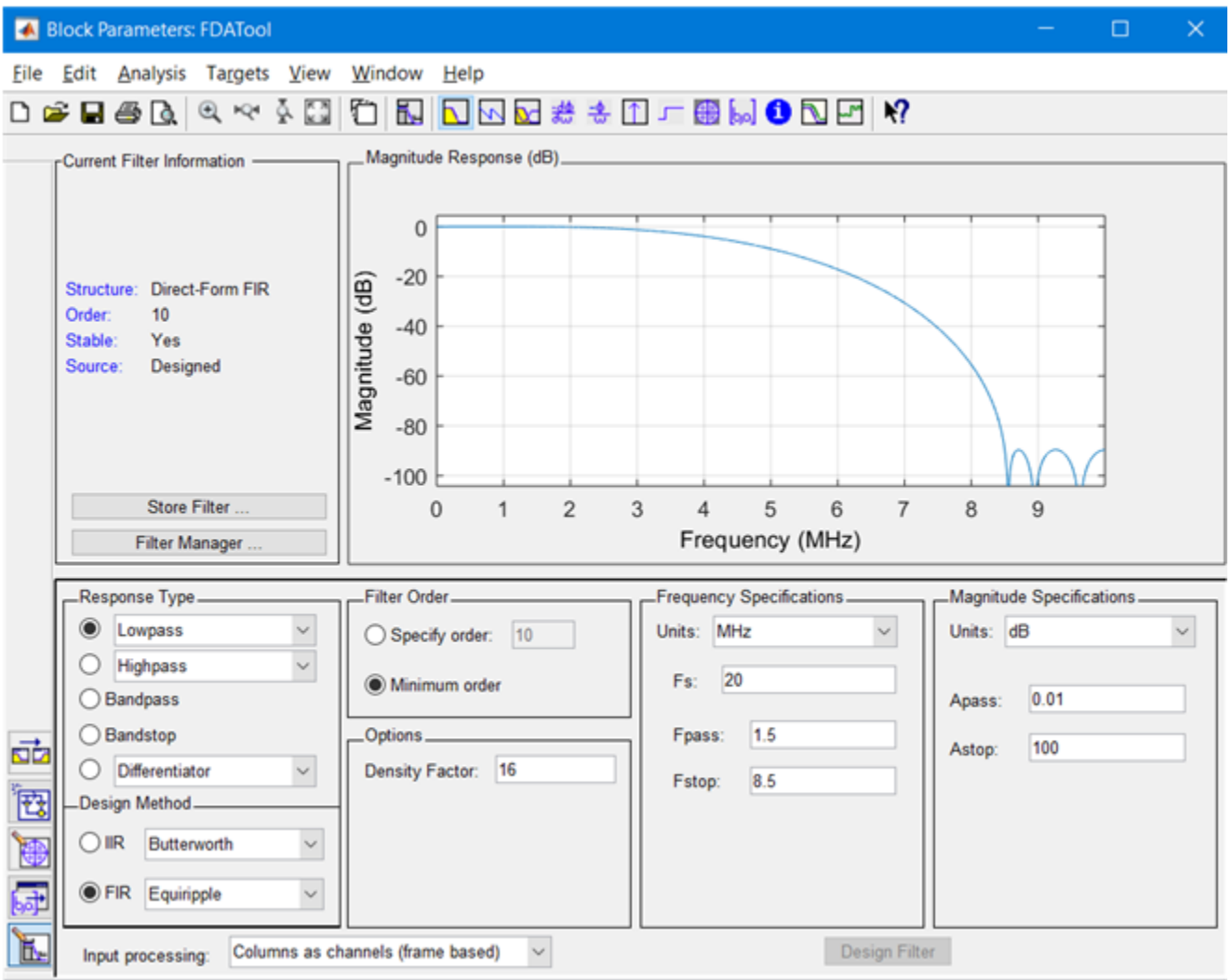
両方のフィルターの周波数応答は、Spectrum Analyzer ブロックで表示します。

図 57: Spectrum Analyzer ブロック



サイリンクスの FDAtool を使用して、ローパス フィルターの係数を定義して高周波数ノイズを除去できます。次の図に示すように、FDAtool のパラメーター ウィンドウで [Response Type]、[Filter Order]、[Frequency Specification]、および [Magnitude Specification] などのフィルター設定パラメーターを変更できます。

図 58: フィルターの設定



ウィンドウ下部の [Design Filter] をクリックすると、フィルター次数と振幅応答を確認できます。また、ツールバーボタンをクリックすると、位相応答、インパルス応答、フィルター係数などを表示できます。フィルター係数を表示するには、MATLAB® ワークスペースに次のように入力します。

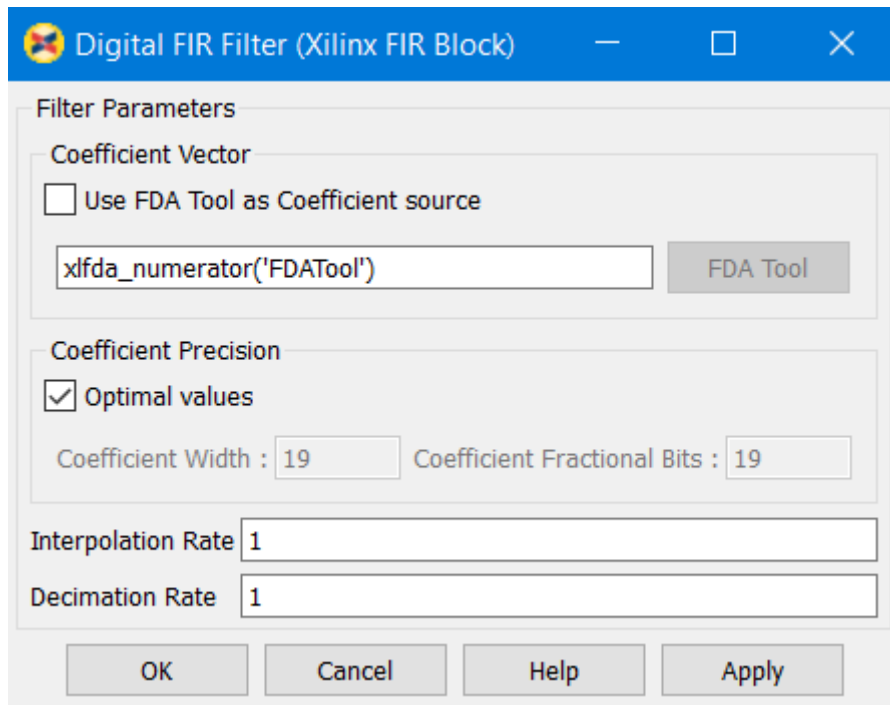
```
>> xlfda_numerator('FDATool')
```

次の関数を使用すると、係数幅と 2 進小数点を正しく指定するための最大係数値および最小係数値がわかります。

```
>> max(xlfda_numerator('FDATool'))
>> min(xlfda_numerator('FDATool'))
```

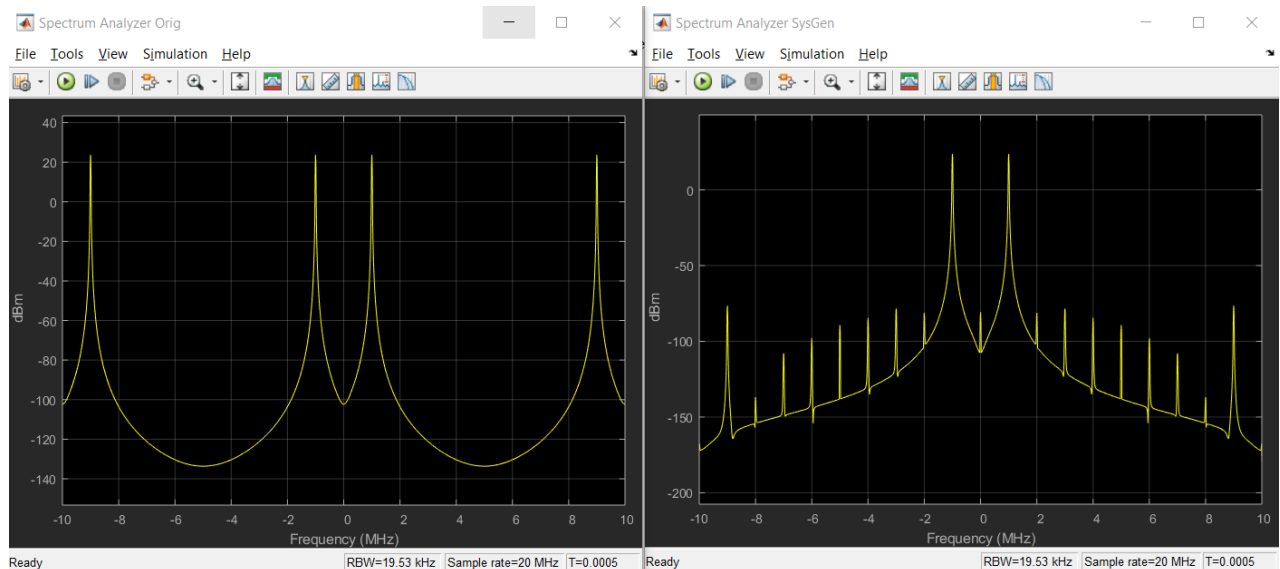
FDATool インスタンスのフィルター パラメーターは、Digital FIR Filter インスタンスに関連付けることができます。

図 59: Digital FIR Filter



Spectrum Analyzer を使用して、ザイリンクスのフィルター応答を表示し、Simulink® 応答と比較できます。

図 60: Spectrum Analyzer



注記: 上の図に示されている System Generator の周波数応答 (右) は、元のデザイン (左) と少し違います。これは、連続時間システムを離散時間ハードウェアで記述する際に発生する量子化およびサンプリングの影響によるものです。

完全な例および FDATool の使用法は、『Vivado Design Suite チュートリアル: System Generator を使用したモデルベースの DSP デザイン』 ([UG948](#)) を参照してください。

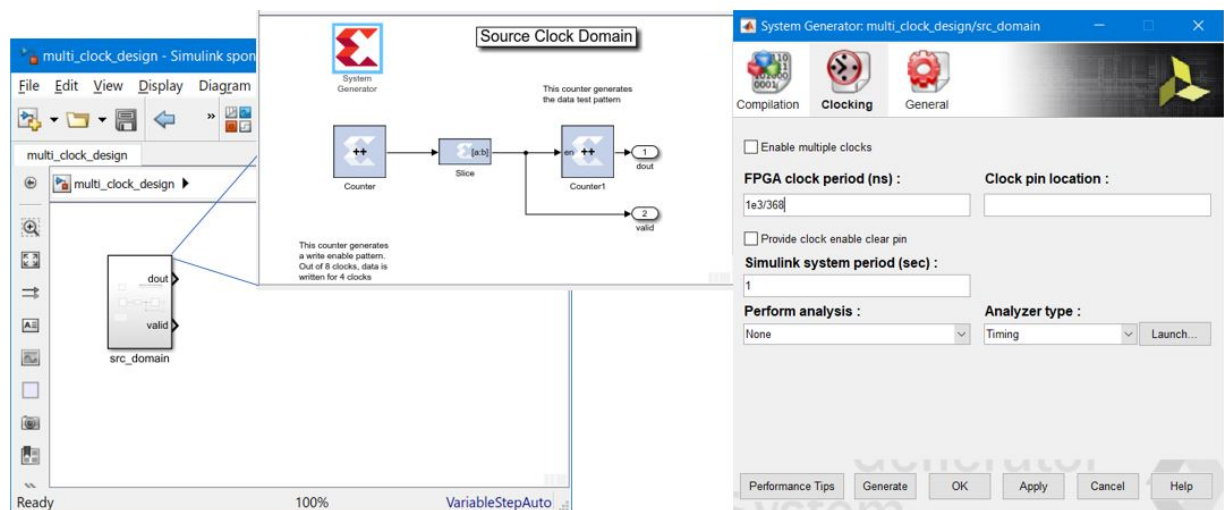
複数の独立クロックのハードウェア デザイン

System Generator for DSP は、サイクル精度の高度なハードウェア モデリングを使用するインプリメンテーション ツールで、そのサイクルの概念はハードウェアのクロックの概念と類似しています。デザインをサブシステム ブロックのグループに分割し、各サブシステムにほかのサブシステムのサイクル周期から独立した共通サイクル周期を使用できます。このセクションでは、ブロックを 1 つのサイクルまたはクロック ドメインにグループ化する方法と、これらのサイクル ドメイン間でデータを転送する方法を説明します。このセクションでは、「サイクル」と「クロック」を同じ意味で使用します。

クロック ドメイン内のブロックのグループ化

ブロックは、System Generator でサブシステムとしてグループ化されます。1 つのクロック ドメイン内でブロックをグループ分けする場合は、System Generator トークンをクロック ドメインとしてマークするサブシステム内に配置する必要があります。これを次の図に示します。

図 61: ソース クロック ドメイン



この図では、src_domain というクロック ドメインのサブシステムが作成されて、System Generator トークンが追加され、System Generator トークンの [Clocking] タブが選択されています。このタブでは、FPGA クロック周期が (1000/368) ns (368 MHz) に、Simulink システム周期が 1 に設定されており、1 Simulink 秒の前進が FPGA クロックの (1000/368) ns に該当することを意味します。

同様に、次の図に示すように、別のクロック ドメインを示す別のブロック グループが dest_domain というサブシステムに含まれます。

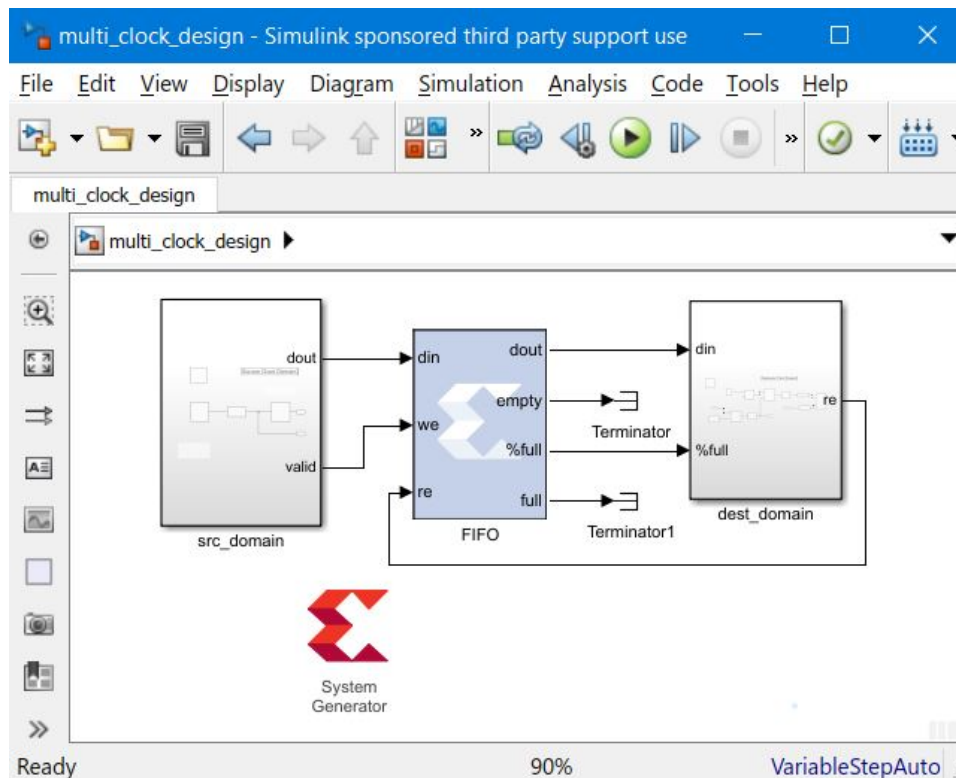
The screenshot shows the Simulink environment with a model named 'multi_clock_design'. The 'src_domain' block is on the left, and the 'dest_domain' block is on the right. The 'dest_domain' block is expanded, showing its internal structure. The status bar at the bottom indicates 'Ready' and '100%' zoom level.

非同期クロック ドメインを作成するために使用する System Generator ブロック

1. FIFO ブロック
2. Dual Port RAM ブロック
3. Register ブロック
4. Black Box ブロック (既存の VHDL、Verilog、および EDIF をデザインに追加できます)。Black Box ユーティリティの詳細は、[第 6 章: HDL モジュールのインポート](#)を参照してください。

UG897 (v2020.1) 2020 年 6 月 3 日
System Generator を使用したデザイン

図 63: FIFO ブロックを使用したドメイン乗せ換え

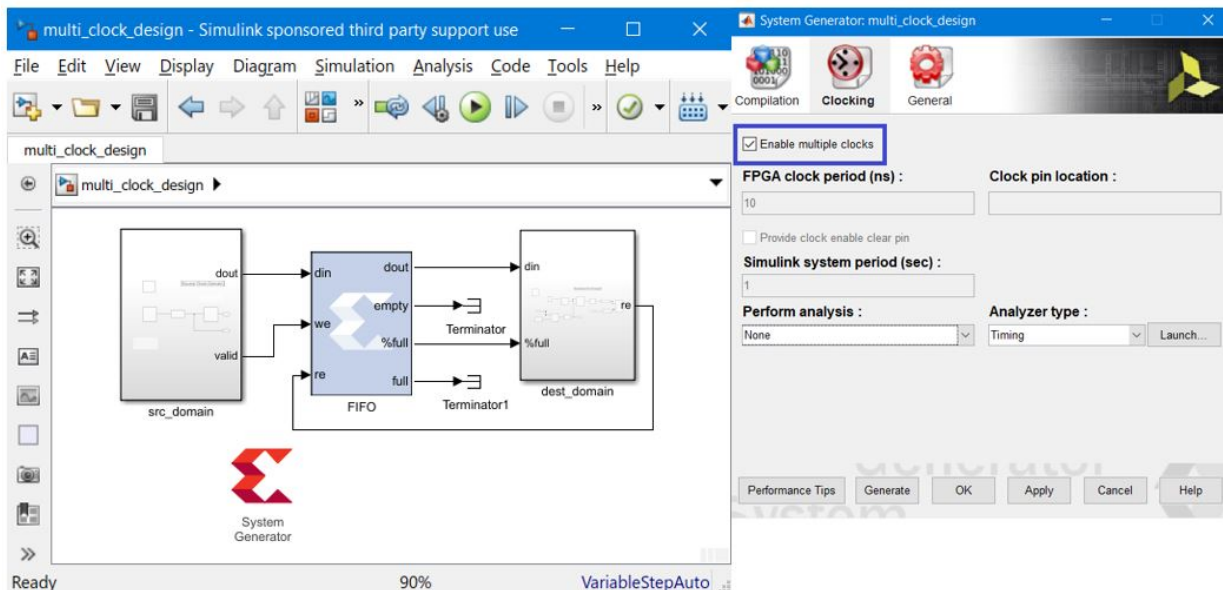


デザインを完了するには、デザインの最上位に FIFO ブロックと追加の System Generator ブロックを含めて、コード生成をイネーブルにします。

最上位 System Generator トークンの設定

複数クロック デザインでは、コードをそれ用に生成するため、最上位 System Generator トークンを設定する必要があります。これには、最上位 System Generator トークンで [Enable multiple clocks] チェック ボックスをオンにします。オンにすると、コード生成エンジンにより `src_domain` および `dest_domain` サブシステムのクロック情報がこれらのサブシステムに含まれる System Generator トークンから取得されます。オフの場合、デザインはシングルクロック デザインとして扱われ、すべてのクロック情報が最上位 System Generator ブロックから継承されます。

図 64: [Enable multiple clocks] チェック ボックス



クロック伝搬アルゴリズム

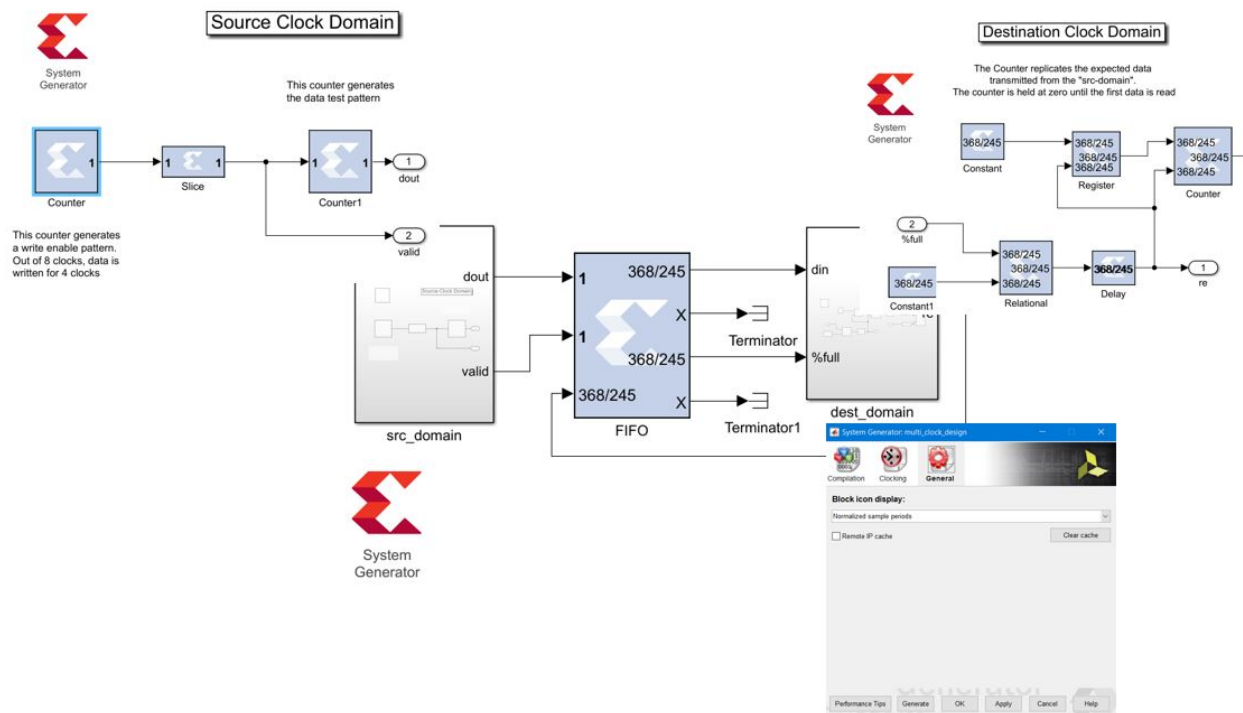
src_domain 内のすべての System Generator ブロックでは、クロッキングが src_domain サブシステムの System Generator トークンで管理されます。dest_domain サブシステムも同様です。FIFO ブロックの場合、クロックはデザインのそのコンテキストから派生されます。we と din ポートは、src_domain サブシステムから出る信号で駆動されるので、FIFO の wr_clk は src_domain クロックに接続されます。dout、full および re ポートは、dest_domain からの信号を駆動するか読み込むので、FIFO の rd_clk は dest_domain クロックに接続されます。クロックドメインを横切ってこれらの信号を混合したり、一致させたり、またはクロックドメインを超えてその他のブロックを使用すると、DRC エラーになります。

クロック伝搬のデバッグ

最上位 System Generator トークンを使用すると、[General] タブの [Block icon display] を使用してすべての System Generator ブロック アイコンの表示を制御できます。このタブから、[Normalized sample periods] または [Sample frequencies] のいずれかを選択すると、デザインでのクロック伝搬を確認できます。

マルチクロック デザインでは、[Normalized sample periods] を選択すると、デザイン内のすべてのサンプル周期を正規化するのに [Simulink system period] の最小値が使用されます。

図 65: クロック伝搬のデバッグ



上の図のように表示するには、最上位 System Generator トークンの [Block icon display] を [Normalized Sample Periods] に設定し、[Apply] をクリックします。

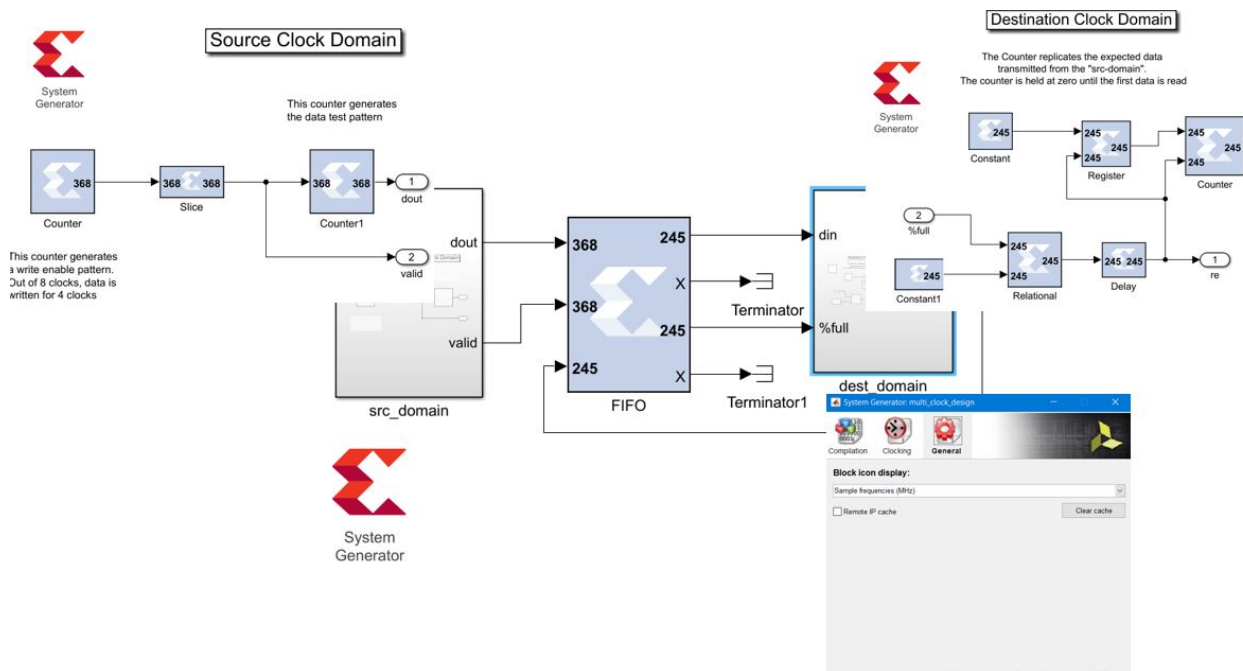
[Sample Frequencies] の場合、次の計算結果ポート アイコンのテキストに表示されます。

$(1e6/\text{FPGA クロック周期}) * \text{Simulink システム周期} / \text{ポート サンプル周期}$

FPGA クロック周期とは、ドメインの System Generator トークンで指定した FPGA クロック周期 (ns) のことで、Simulink システム周期とは、ドメインの System Generator トークンで指定した Simulink システム周期 (秒) のことです。

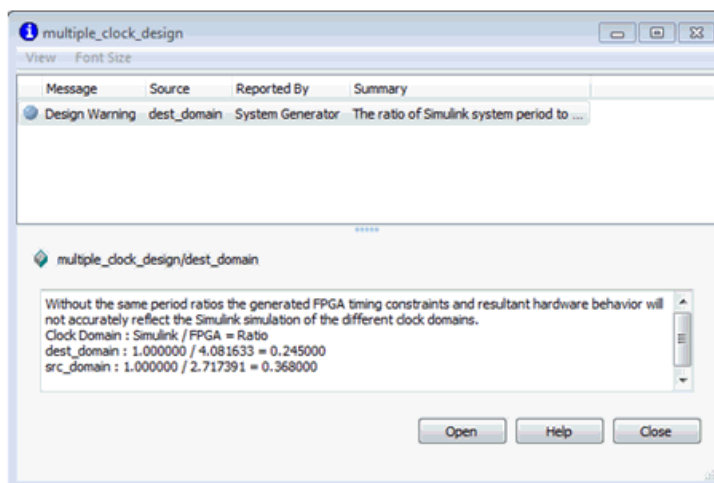
[Sample Frequencies] を使用すると、クロック伝搬が正しいかを確認することもできます。

図 66: 周波数の例



シミュレーションでハードウェアの動作がクロックに対してモデリングされるようにするには、FPGA クロック周期に対する Simulink システム周期の比率が各ドメインで同じである必要があります。この関係が正しい比率でコンパイルされないと、次の図のような警告メッセージが表示されます。

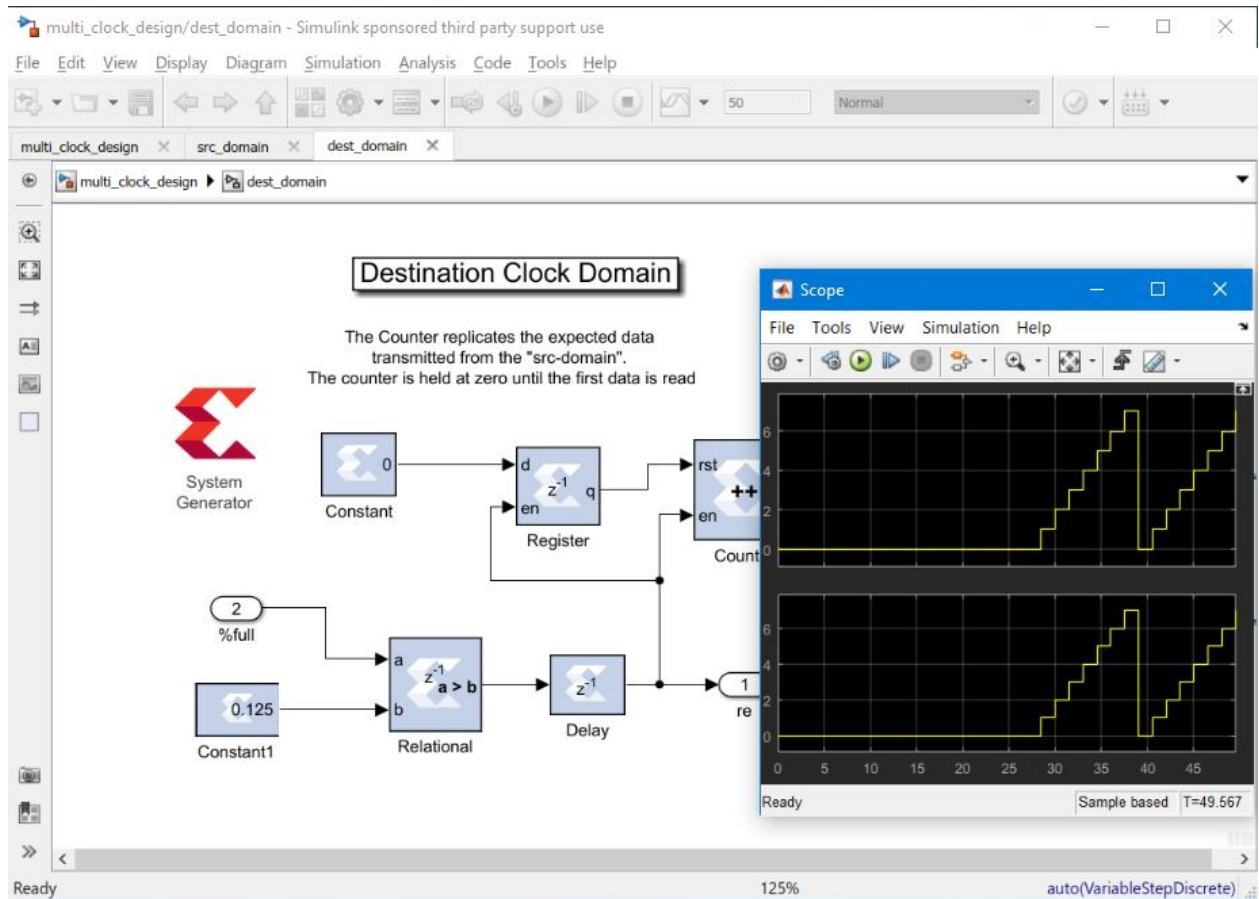
図 67: 警告メッセージ



シミュレーション

シミュレーションを実行すると、次の図の `dest_domain` スコープに示すように結果が表示されます。

図 68: シミュレーション



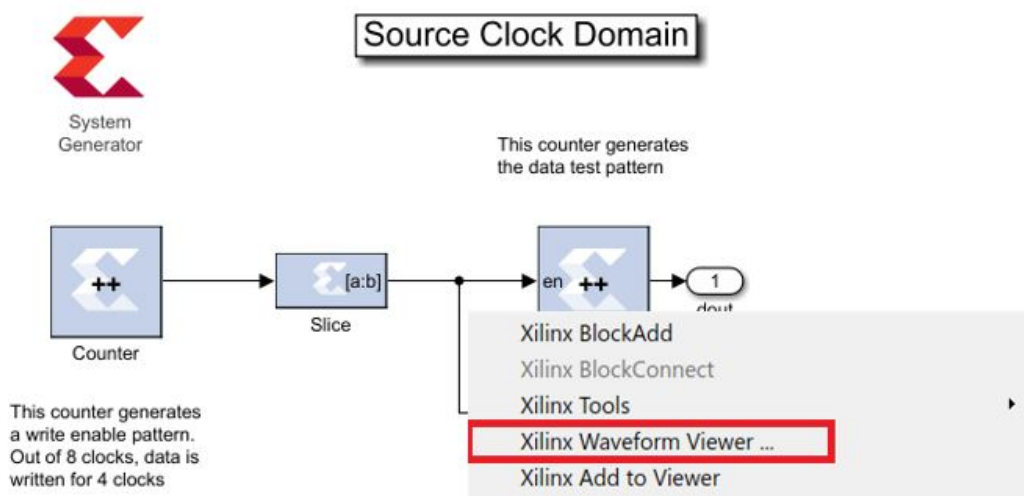
この図のシミュレーション結果では、取得されたデータが予測されたデータと一致していることが示されています。

注記: このクロック乗せ換えでのシミュレーションは、サイクル精度ではありません。

複数のクロック ドメイン信号のデバッグ

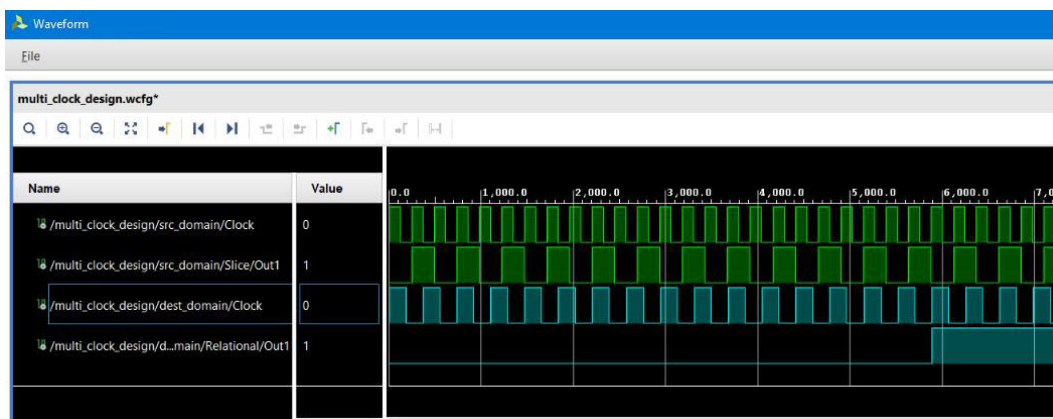
System Generator では、サイリンクス波形ビューアーと Simulink® 図の間のクロスプローブを使用できるので、デバッグしやすくなっています。

図 69: ソース クロック ドメイン



波形ビューアーに信号を追加するには、モデルを信号を右クリックして [Xilinx Add To Viewer] をクリックします。デザインをシミュレーションすると、次の図のように波形ビューアーが起動します。

図 70: 波形ビューアー



同じクロック ドメインの信号すべてが同様に色分けされます。上の図では、src_domain/Slice/Out1 と dest_domain/Relational/Out1 は異なるクロック ドメインにあります。

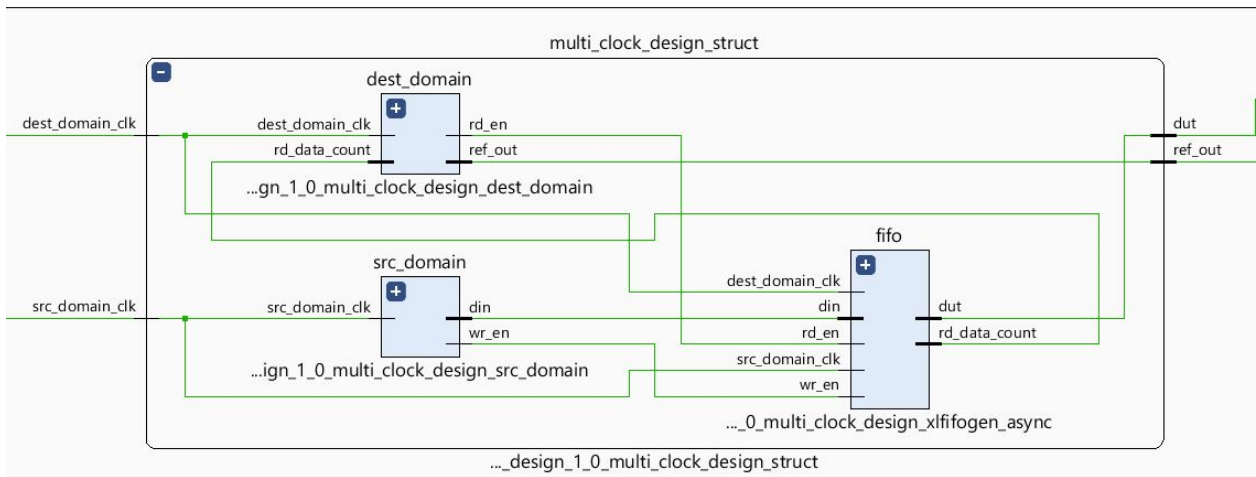
コード生成

マルチクロック デザインのコード生成では、次のコンパイル ターゲットがサポートされます。

- [HDL Netlist]
- [IP Catalog]
- [Synthesized Checkpoint]

次に、最上位ハードウェアのスクリーンショットを示します。

図 71: 最上位ハードウェア



クロック ドメインと同じ数のクロック ポートを最上位で使用でき、MMCM や PLL などのさまざまなザイリンクス クロッキング構造で駆動できます。これらのクロックは完全に非同期で、次の PERIOD 制約が作成されます。

```
1 | create_clock -name src_domain_clk -period 2.717 [get_ports src_domain_clk]
2 | create_clock -name dest_domain_clk -period 4.082 [get_ports dest_domain_clk]
```

IP に追加のクロック ドメイン制約が埋め込まれている FIFO または Dual Port RAM だけが使用できるので、必要な制約はこれだけです。

既知の問題

次に、既知の問題の一部を示します。

- HWCosim コンパイル ターゲットは複数クロック デザインではサポートされません。
- 複数クロックを使用する場合、FIFO および Dual Port RAM ブロックのみを最上位にできます。
- クロック乗せ換えを支援するブロックの動作はサイクル精度ではありません。
- 未接続または終端の出力ポートは波形ビューアーで表示できません。

AXI インターフェイス

AMBA AXI4 (Advanced eXtensible Interface 4) は、Arm® で定義され制御される 4 世代目の AMBA インターフェイスで、ザイリンクスでは FPGA デザインの次世代のインターコネクトとして採用しています。ザイリンクスおよび Arm は緊密に連携し、AXI4 仕様が FPGA のニーズを満たすようにしています。

AXI はオープン インターフェイス規格で、公開されていて、使用料無料、業界標準であるため、多くのサードパーティ IP ベンダーにより広く使用されています。

AMBA AXI4 インターフェイスの接続はポイント ツー ポイントで、AXI4、AXI4-Lite スレーブ、AXI4-Stream の 3 つがあります。

- AXI4 はバースト トランザクションをサポートするメモリ マップのインターフェイスです。

- AXI4-Lite スレーブは AXI4 の簡易バージョンで、インターフェイスはバースト対応ではありません。
- AXI4-Stream は一方向のデータ転送 (マスターからスレーブ) のハイ パフォーマンスのストリーミング インターフェイスで、AXI4 よりも信号要件が少なくなっています。AXI4-Stream では、同じワイヤ セットの複数チャネルのデータがサポートされます。

次の資料では、AXI4 は AXI4 メモリ マップ インターフェイスを、AXI4-Lite スレーブおよび AXI4-Stream はそれぞれ AMBA AXI4 インターフェイスの該当するタイプを示します。インターフェイスのコレクションを示す場合は、「AMBA AXI4」が使用されます。

このセクションでは、AMBA AXI4 の概要を示し、System Generator に関する AMBA AXI4 の詳細を説明します。AMBA AXI4 仕様の詳細は、ザイリンクス ウェブサイトの [AMBA AXI4 インターフェイス プロトコル](#) ページにあるザイリンクス AMBA AXI4 の資料を参照してください。

System Generator での AXI4-Stream のサポート

最もよく使用される AXI4-Stream 信号は、TVALID、TREADY、TDATA の 3 つです。AXI4-Stream 信号すべての中で必須なのは TVALID だけで、それ以外はオプションです。情報を伝達する信号はすべて TVALID と同じ方向に伝搬されますが、TREADY だけは反対方向に伝搬されます。

AXI4-Stream はポイント ツー ポイント インターフェイスなので、マスターおよびスレーブ インターフェイスの概念はデータフローの方向性を示す上で重要です。マスターがデータを生成し、スレーブがデータを消費します。

命名規則

AXI4-Stream 信号は、次のように命名されます。

```
<Role>_<ClassName>[_<BusName>][_<ChannelName>]<SignalName>
```

次に例を示します。

```
m_axis_tvalid
```

この場合、`m` は Role (マスター)、`axis` は ClassName (AXI4-Stream)、`tvalid` は SignalName です。

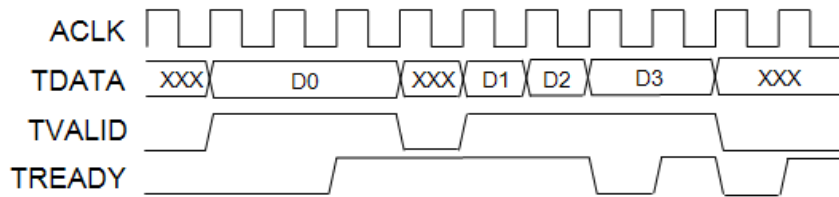
```
s_axis_control_tdata
```

この場合、`s` は Role (スレーブ)、`axis` は ClassName、`control` は特定 IP の同じクラスの複数インスタンス間を区別する BusName、`tdata` は SignalName です。

TREADY/TVALID ハンドシェイクに関する注意事項

TREADY/TVALID ハンドシェイクは、双方向フローを制御できるようにマスターとスレーブ間のデータ交換を制御する AXI の基本的概念です。TDATA およびその他すべての AXI4-Stream 信号 (TSTRB、TUSER、TLAST、TID、TDEST) はすべて TREADY/TVALID ハンドシェイクで認証されます。マスターは TVALID がアサートされるとデータの有効なビートを示し、TREADY がアサートされるまでデータ ビートを保持する必要があります。TVALID はアサートされると、TREADY がアサートされるまでディアサートできません。AXI では、TREADY を TVALID に依存させることはできても、TVALID のアサートは TREADY に依存させることができないという規則も追加されています。この規則により、タイミング ループが回避されます。次のタイミング図に、TREADY/TVALID ハンドシェイクの例を示します。

図 72: TREADY/TVALID ハンドシェイクのタイミング図



ハンドシェイクの例

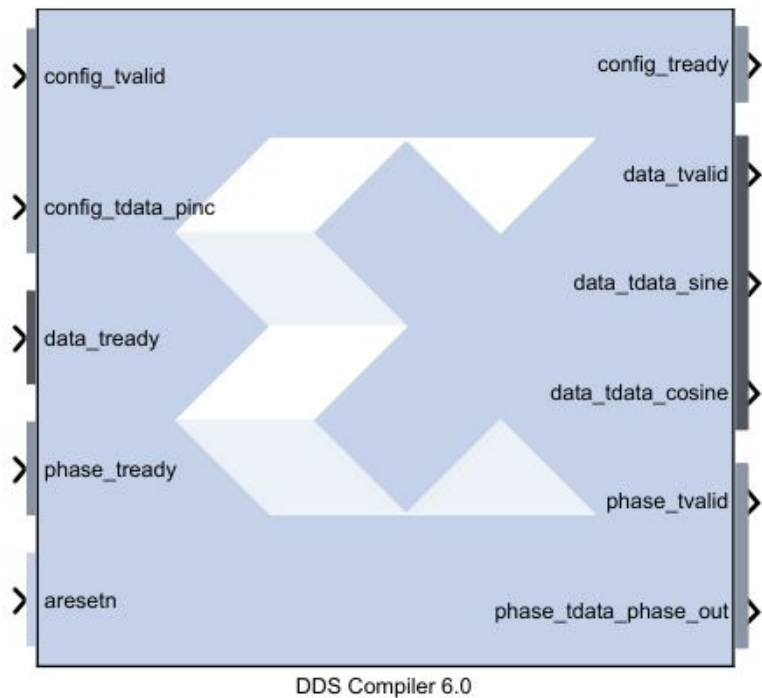
- TREADY および TVALID の両方が同じサイクルで High になると、指定したチャンネルで転送が発生します。
- TVALID がアサートされると、転送が終了するまでディアサートできません (TREADY は High)。転送は取り消したり、中止したりできない可能性があります。
- TVALID がアサートされると、同じチャンネルのそれ以外の信号 (TREADY 以外) の値は転送が終了する (TREADY のアサート後のサイクル) まで変化しません。
- TREADY は TVALID のアサート前、アサート中またはアサート後のサイクルでアサートできます。
- TVALID のアサートは TREADY の値に依存させることができませんが、TREADY のアサートは TVALID の値に依存させることができます。
- マスターおよびスレーブ インターフェイスの両方で入力信号と出力信号の間に組み合わせパスがないようにする必要があります。
 - AXI4-Stream IP に適用されます。つまり、TREADY スレーブ出力は TVALID スレーブ入力から組み合わせで生成できません。TVALID で認証されたデータを即時受信できるスレーブは、データを受信するまでその TREADY 信号をプリアサートする必要があります。または、TREADY にレジスタを付けて、TVALID アサートの次のサイクルを駆動することもできます。
 - デフォルトのデザイン規則では、スレーブは TREADY を独立して駆動するか、TREADY をプリアサートして、レイテンシを最小限に抑える必要があります。
 - 入力信号と出力信号間の組み合わせパスは別々の AXI4-Stream チャンネル間で使用できますが、同じインターフェイス (一緒に動作するチャンネルの関連グループ) に属する複数チャンネルには、使用しないでください。
- 該当するチャンネルでは、TREADY を除いてすべての信号がソース (通常はマスター) からデスティネーション (通常はスレーブ) まで伝搬されます。それ以外の反対方向へ伝搬する必要のある情報伝達信号または制御信号は、すべて別のチャンネルの一部 (別の TREADY/TVALID ハンドシェイクを使用したバックチャンネル) であるか、アウト オフ バンド信号 (ハンドシェイク以外) のいずれかである必要があります。スレーブからマスターへの反対方向への情報の転送には、TREADY を使用しないでください。
- AXI4-Stream では、TREADY を省略できます (デフォルトの値は 1)。これにより、TREADY を生成する IP での相互運用に制限が出る場合があります。AXI4-Stream マスターは前方向のフロー制御 (TVALID) にのみ接続できます。

System Generator での AXI4-Stream ブロック

AXI4-Stream インターフェイスを含む System Generator ブロックは、AXI4 というザイリンクス ブロックセット ライブラリにあります。このライブラリのブロックの図は、通常の AXI4-Stream ではないブロックとは多少異なります。

ポート グループ

図 73: DDS Compiler 6.0



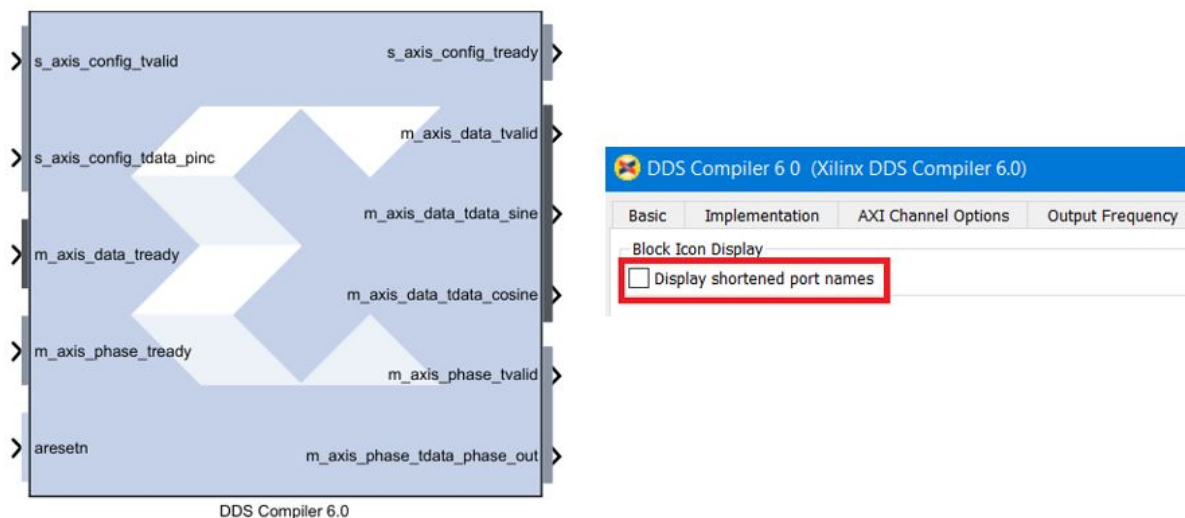
AXI4-Stream インターフェイスを提供するブロックでは、AXI4-Stream チャンネルがグループ化および色分けされています。たとえば、上記の DDS Compiler 6.0 ブロックでは、入力ポート `data_tready` と 3 つの出力ポート `data_tvalid`、`data_tdata_sine`、および `data_tdata_cosine` が同じ AXI4-Stream チャンネルにあります。同様に、入力ポート `config_tvalid`、`config_tdata_pinc`、および出力ポート `config_tready` が同じ AXI4-Stream チャンネルに、`phase_tready`、`phase_tvalid`、および `phase_tdata_phase_out` が同じチャンネルにあります。

AXI4-Stream チャンネルには含まれない信号は、ブロックと同じ背景色で表示されます。`aresetn` はその例です。

ポート名の省略

次の例では、ブロックに表示される AXI4-Stream 信号の名前が省略されていて読みやすくなっています。名前の省略はあくまで表記上のもので、ネットリストでは完全な AXI4-Stream 名が使用されます。名前の省略はデフォルトでオンになりますが、ブロックのパラメーター ダイアログ ボックスの [Display shortened port names] をオフにしてその完全名を表示させることもできます。

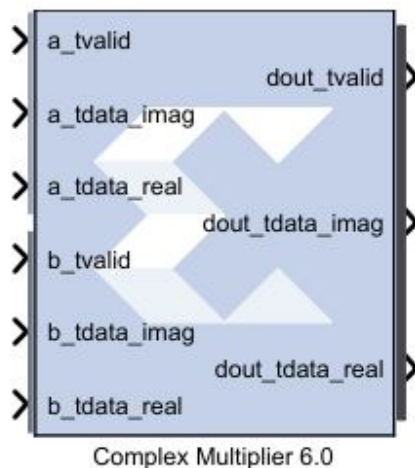
図 74: DDS Compiler 6.0



マルチチャネル TDATA の分割

AXI4-Stream では、TDATA に複数チャネルのデータを含めることができます。System Generator では、TDATA のチャネルが分割されます。たとえば、次の dout ポートの TDATA には、現実と仮想のコンポーネントの両方が含まれます。

図 75: Complex Multiplier 6.0



マルチチャネル TDATA を分割してもデザインにロジックは追加されないの、ユーザー使いやすいように System Generator で分割されます。分割された TDATA ポートのデータも正しくバイト アライメントされます。

AXI4-Lite インターフェイスの生成

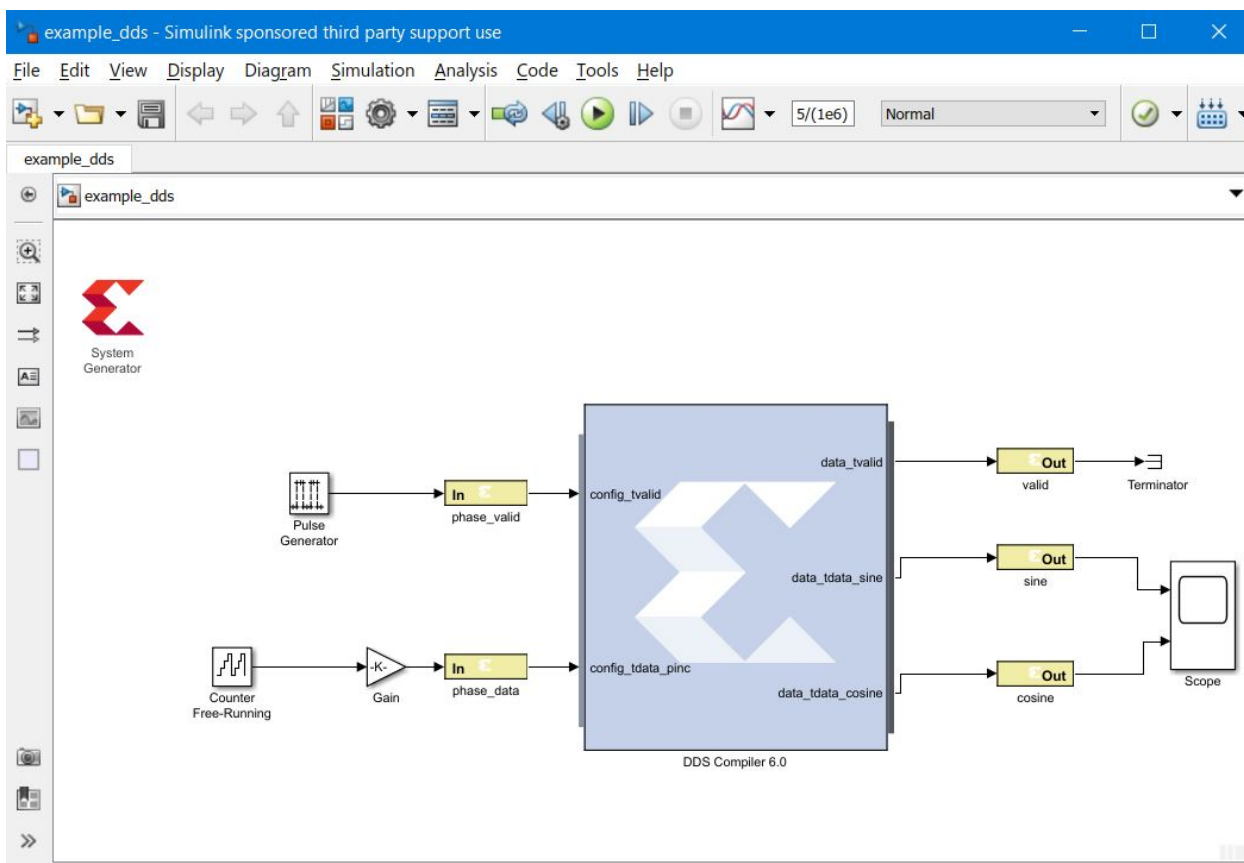
System Generator を使用して開発したデザイン モジュールは、通常より大きい DSP またはビデオ システムのサブシステムを形成します。これらの System Generator モジュールでは、アルゴリズムおよびデータパスが多用されるのが一般的で、MATLAB/Simulink のような視覚化に優れた環境で作成するのがベストです。より大型のシステムは、通常 Vivado® IP カタログの IP からアセンブルされます。これらの IP では、標準ストリームと AXI4-Lite スレーブのような制御インターフェイスが使用されます。大型システムのアセンブルには、Vivado IP インテグレーターのようなツールが使用されます。

System Generator モジュール用に標準 AXI4-Lite インターフェイスを作成し、IP インテグレーターを使用して大型デザインに含めるためにそのモジュールを Vivado® IP カタログにエクスポートする System Generator の機能を説明します。System Generator では、複数のクロック ドメインで複数の AXI4-Lite インターフェイスを作成することもできます。

System Generator での AXI4-Lite インターフェイスの合成

デザインの作成および検証は、AXI4-Lite インターフェイスを含まない System Generator デザインと同じです。次の `example_dds` デザインについて説明します。

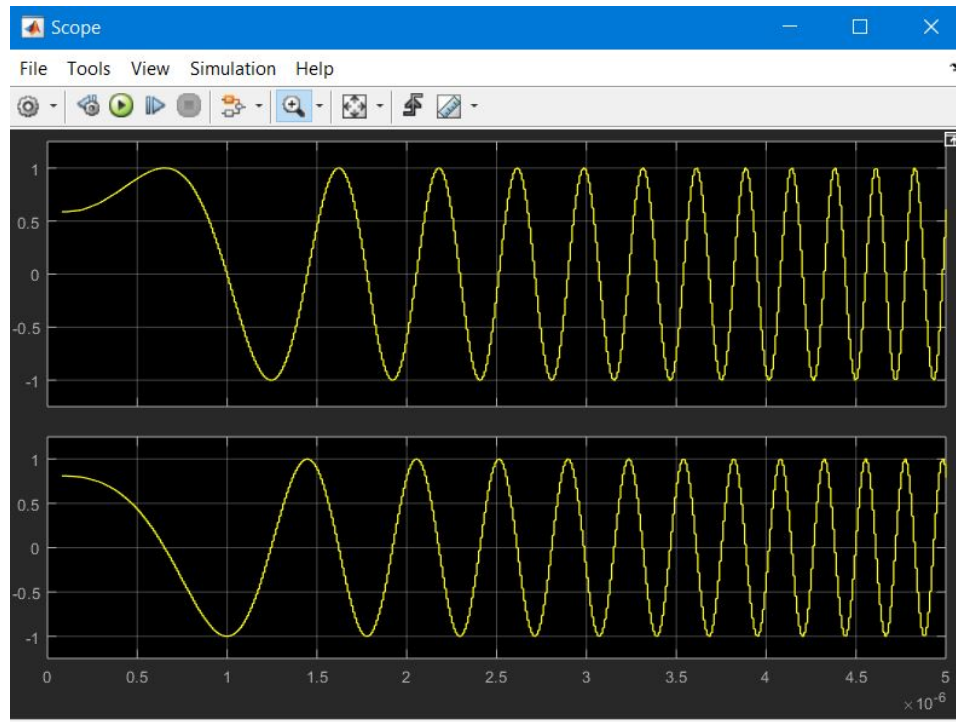
図 76: サンプル DDS デザイン



このデザインには DDS Compiler が含まれ、出力周波数を制御するために 2 つの入力、`config_tvalid` および `config_tdata_pinc` が使用されています。

次はこのデザインのシミュレーション結果で、サイン出力とコサイン出力が別に示されています。

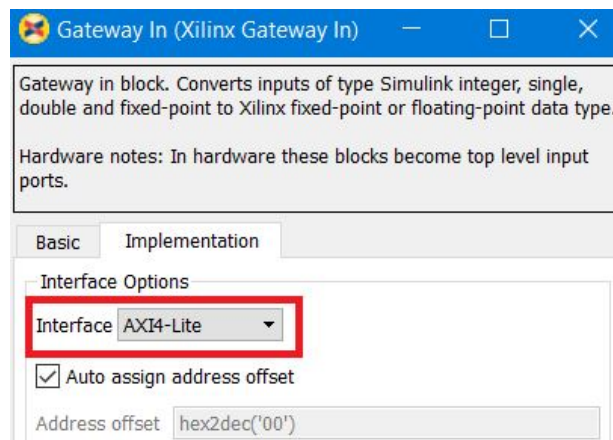
図 77: シミュレーション結果



AXI4-Lite インターフェイス用デザイン設定

example_dds デザインでは、Gateway In および Gateway Out ブロックにより、Simulink デザインのサイクルおよびビット精度の FPGA 部分の境界がマークされます。DDS Compiler の周波数は、Gateway In の出力ポート (phase_valid および phase_data) に接続された信号に正しい値を挿入することで制御できます。これには、次の phase_valid ブロックの例で示すように [Interface] オプションを変更します。

図 78: [Interface] オプション



この例では、System Generator で [Interface] にスレーブ AXI4-Lite インターフェイスが指定されているので、最上位の AXI4-Lite インターフェイスに変換されます。

次のオプションを使用することもできます。

[Auto assign address offset] (オン): 各 Gateway が AXI4-Lite インターフェイス内のレジスタに接続され、AXI4-Lite インターフェイスにマップされた異なる Gateway In の番号に基づいて、アドレス オフセットの自動割り当てが実行されるように指定します。アドレスは、32 ビット データ幅にバイトでアライメントされます。

[Address offset] (オフ): このオプションは、[Auto assign address offset] がオフの場合にのみオンにできるようになります。これにより、ユーザーがアドレス オフセットを手動で上書きできるようになります。

[Interface Name]: インターフェイスに名前を付けます。デザインに複数の AXI4-Lite インターフェイスがある場合は、この名前ですべてのインターフェイスが識別されます。



重要: 名前には、英数文字 (アルファベットは小文字)、またはアンダースコア () のみが使用できます。また、名前の最初の文字は小文字のアルファベットにする必要があります。たとえば、「axi4_lite1」という名前は使用できますが、「1Axi4-Lite」は使用できません。

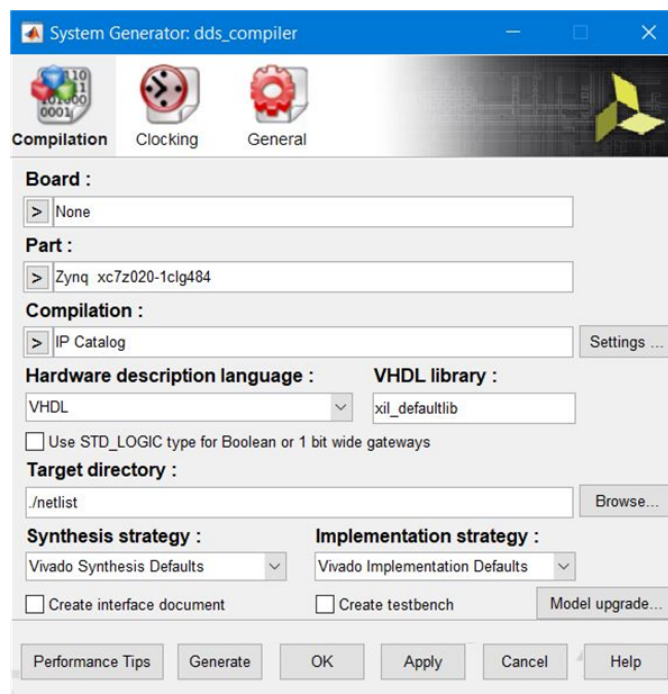
[Description]: ここに入力した情報は、デザインが Vivado IP カタログにエクスポートされるときに自動的に作成されるインターフェイスの資料に含まれます。

その他の Gateway も同じ方法で設定します。

Vivado IP インテグレーターで使用するためのデザインのパッケージ

System Generator での検証が終了したら、デザインを IP インテグレーターでできるようにパッケージできます。

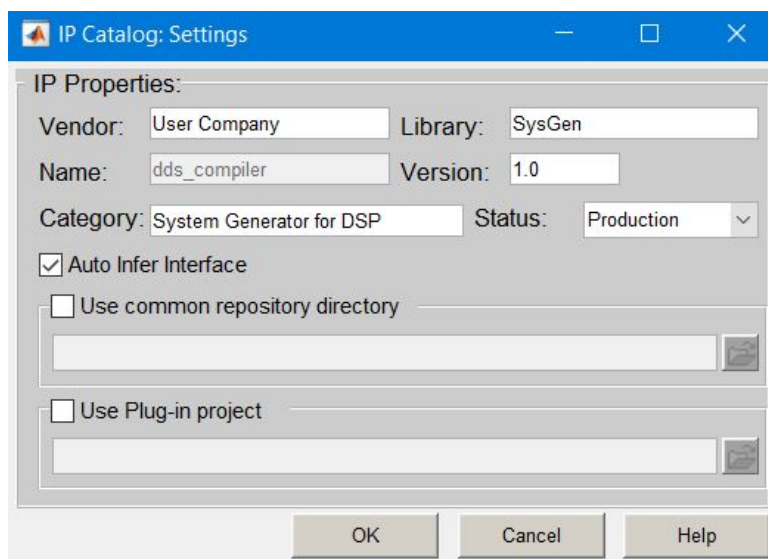
図 79: System Generator 検証



まず、[Compilation] を [IP Catalog] (デフォルトの生成ターゲット) に設定します。このオプションでは、System Generator から作成されたすべてのハードウェア ソース (RTL + IP + 制約) が IP にまとめられます。

選択されるパーツは、ザイリンクス Zynq-7000 ZC702 評価ボードで使用可能なパーツと同じです。また、System Generator トークンの [Settings] ボタンを使用しても IP に含まれる情報を変更できます。この場合、次に示すデフォルト値が使用されます。

図 80: IP カタログの設定



System Generator トークンの GUI で [Generate] ボタンをクリックすると、RTL、IP、および制約がまとめられてパッケージが作成されます。

生成結果の詳細

前述の System Generator 設定に基づいて、次のフォルダーおよびファイルが作成されます。

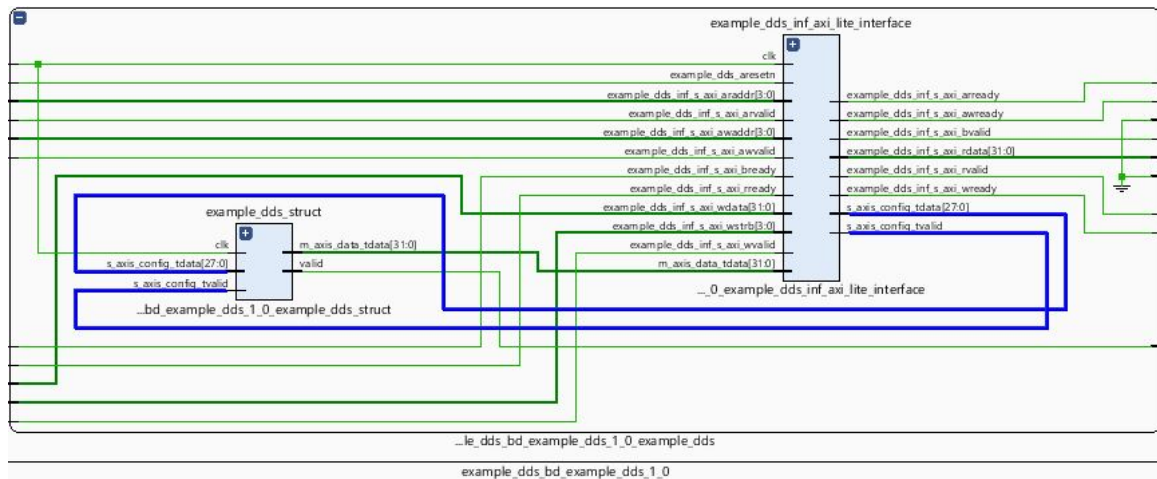
1. `<target_directory>/ip`: このディレクトリには、IP 関連のハードウェア ファイルおよびソフトウェア ドライバーが含まれます。IP カタログに追加する必要があるのはこのディレクトリです。
2. `<target_directory>/ip_catalog`: このディレクトリには `example_dds.xpr` という Vivado IDE プロジェクトのサンプルが含まれます。

AXI4-Lite インターフェイスへのマップ

AXI4-Lite レジスタとしてタグ付けされた Gateway Ins および Gateway Outs は、次の回路図に示すように、メモリ マップ内の異なる 32 ビット レジスタにマップされます。

次の回路図は、1 つの AXI4-Lite インターフェイスへのマップ例で、すべてのゲートウェイに同じインターフェイス名が指定されているものとします。複数の AXI4-Lite インターフェイスがある回路図では、同じインターフェイス名のあるゲートウェイのグループごとに、それぞれ AXI4-Lite インターフェイスがあります。

図 81: 1 つの AXI4-Lite インターフェイス



図に示すように、example_dds_inf_axi_lite_interface というモジュールが RTL デザインに挿入され、このモジュールにより System Generator デザインの config_tvalid および config_tdata ポートが駆動されます。最上位では、スレーブ AXI4-Lite インターフェイスが使用されます。このモジュールによりアドレス デコードが実行され、プロセッサから取得したアドレスに基づいて config_tvalid または config_tdata ポートが駆動されます。

アドレス指定 (s_axi_araddr および s_axi_awaddr) に必要なビット数は、AXI4-Lite インターフェイス レジスタの数と各 AXI4-Lite レジスタのオフセット仕様によって決まります。モジュール生成中には各レジスタを別々にデコードするのに十分なビットが提供されます。この例には phase_data と phase_valid という 2 つのゲートウェイがあり、各ポートにアドレス オフセット 0x0000、0x0004 が割り当てられます。この結果、3 つのアドレスビットが割り当てられます。

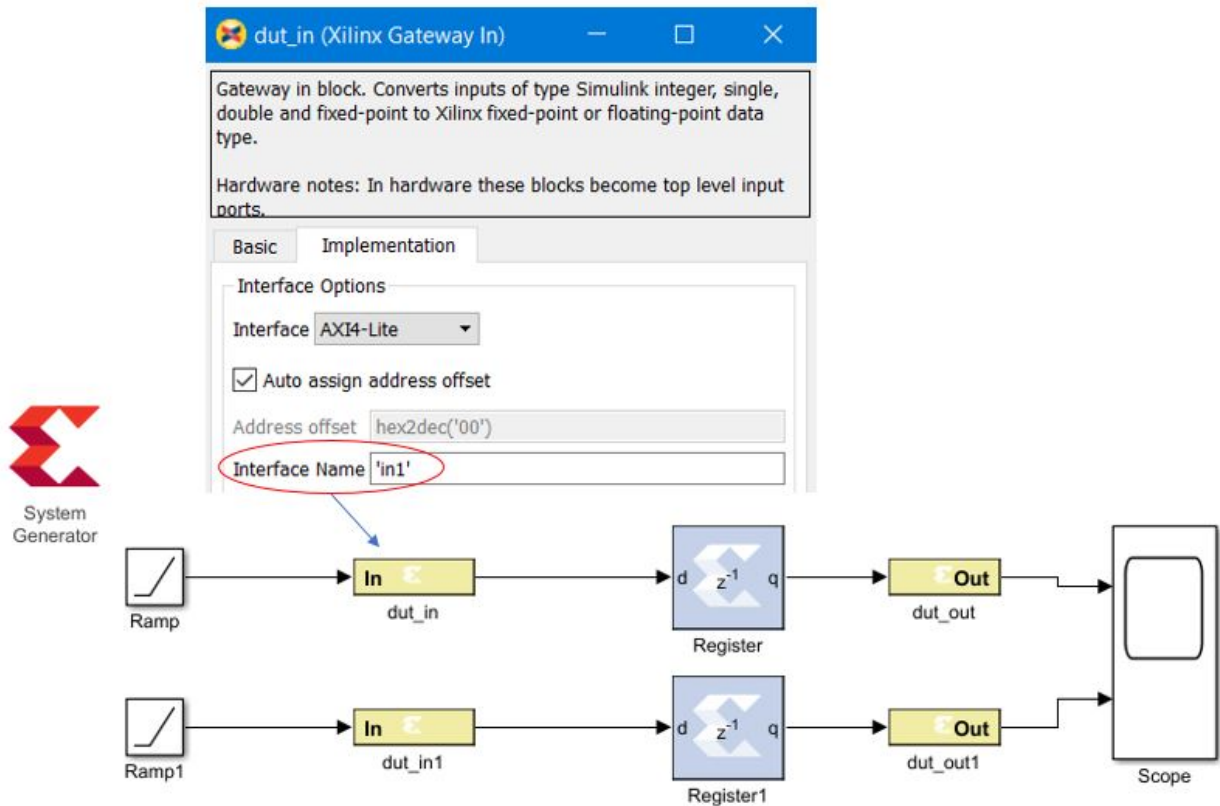
複数の AXI4-Lite インターフェイスの管理

System Generator では、複数の AXI4-Lite インターフェイスを含む IP の作成がサポートされます。Gateway In および Gateway Out ブロックを、異なる AXI4-Lite インターフェイスにまとめることができます。この機能は、複数クロック デザインでも使用できます。ソフトウェア ドライバーも提供されます。

AXI4-Lite インターフェイスに名前を付けるには、インターフェイスに関連付けられている Gateway In および Gateway Out ブロックの [Interface Name] にインターフェイス名を入力します。

[Interface Name] が同じ Gateway In および Gateway Out ブロックは、1 つの AXI4-Lite インターフェイスにまとめられます。[Interface Name] で指定するインターフェイス名は、小文字のアルファベットで開始し、英数字 (アルファベットは小文字) およびアンダースコア () のみを含めることができます。複数のクロック ドメインで [Interface Name] に同じインターフェイス名を指定することはできません。

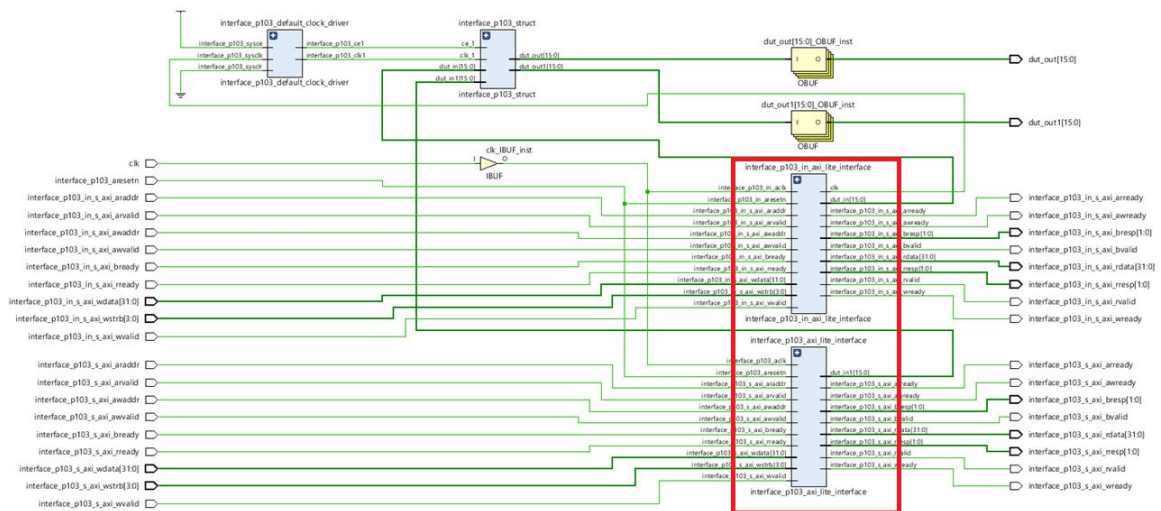
図 82: [Interface Name]



ネットリストを生成するには、コンパイル ターゲットを [IP Catalog] または [HDL Netlist] に設定します。

System Generator トークンでコンパイル ターゲットを [HDL Netlist] に設定して Vivado でデザインをエラポレートすると、次の図の赤いボックスで示すように、AXI4-Lite デコーダーが 2 つ作成されます。

図 83: AXI4-Lite デコーダー

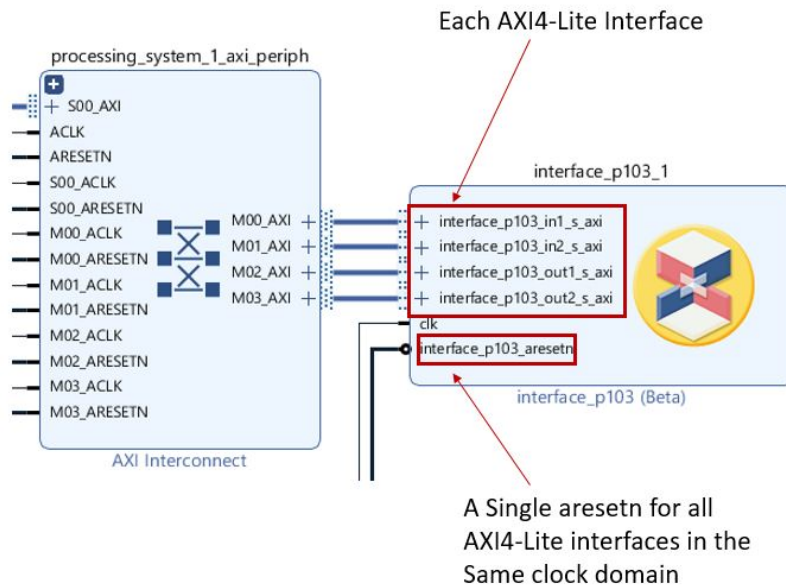


System Generator トークンでコンパイル ターゲットを [IP Catalog] に設定すると、複数の AXI4-Lite インターフェイスと `aresetn` 信号を含むサンプル BD も生成されます。

インターフェイスの命名規則は次のとおりです。

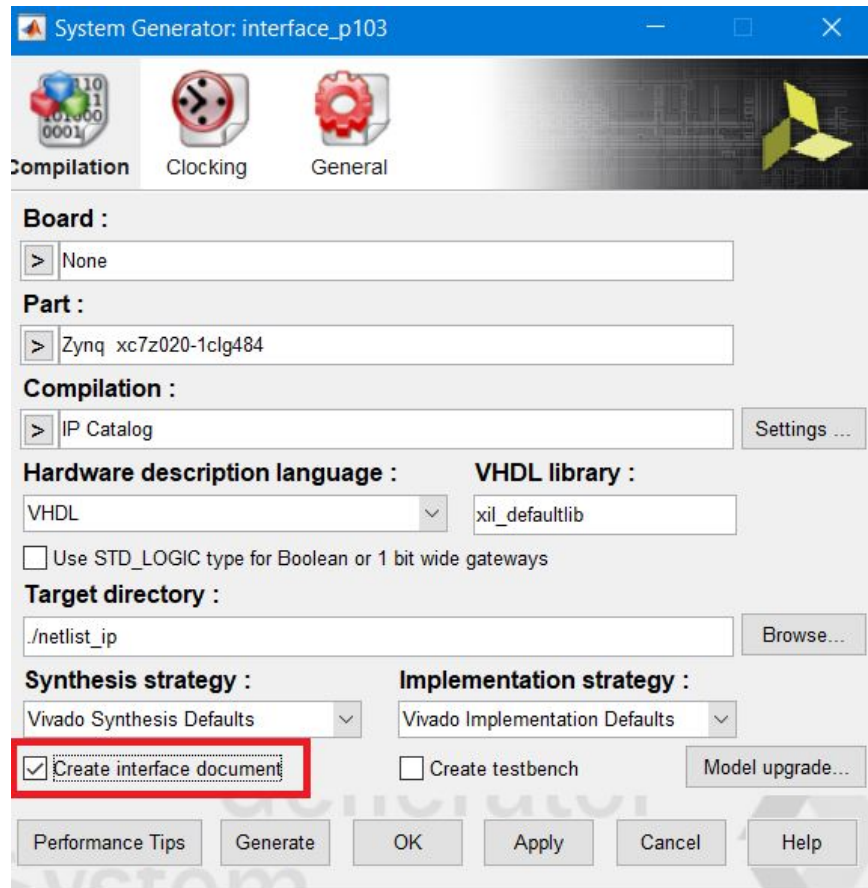
```
<clock domain name/design name>_<interface name>_s_axi
```

図 84: サンプル BD



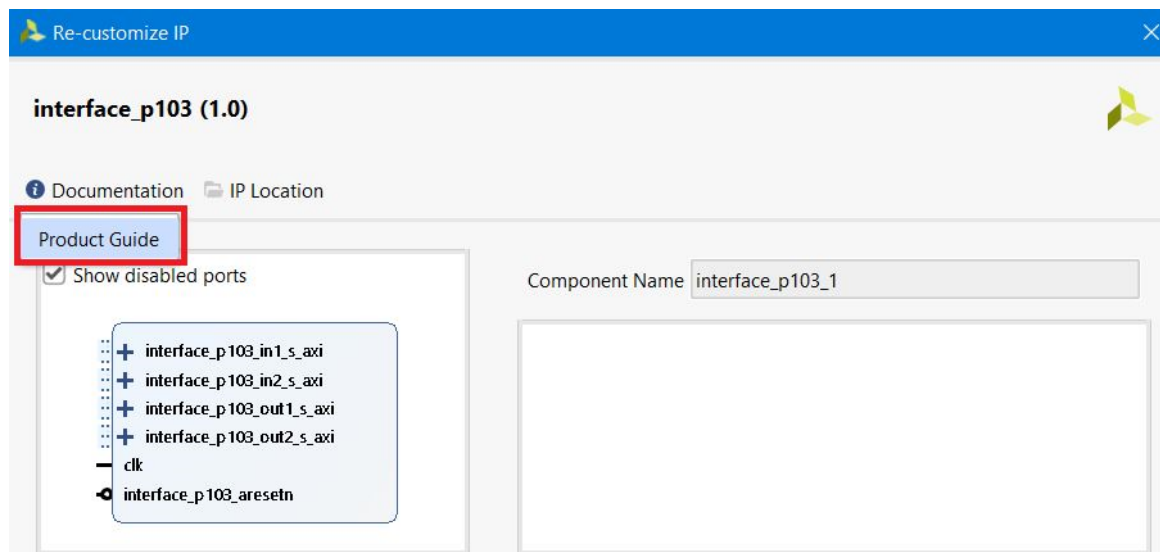
IP を説明するファイルを生成するには、コンパイルを実行する前に、System Generator トークンの [Compilation] タブで [Create interface document] をオンにします。

図 85: [Create interface document] チェック ボックス



資料には、ほかの Vivado IP の資料と同じようにアクセスできます。Vivado 回路図で IP をダブルクリックし、[Documentation]→[Product Guide] をクリックします。

図 86: 資料へのアクセス



次のようなファイル (HTML) が開きます。

図 87: 資料の例



interface_p103

02-Feb-2019

Introduction

This document is generated from a **Xilinx System Generator for DSP** (System Generator) design. The purpose of the document is to specify the interface of this design. Each of the subsequent sections provides details on the port interface, signal timing, design files, design statistics and design environment.

Port Interface

This section documents the port interface of interface_p103. All the *Gateway In* and *Gateway Out* blocks in a System Generator design are translated to top-level input and output ports. *System Generator Type* refers to the type of signals emanating from Gateway Ins and driving Gateway Outs. *Type* refers to one of the following -

- Data - Signals that are synchronized to Clock
- Clock - Clock signal for the design. All operations of the core are synchronized to the rising edge of the Clock signal
- Clock Enable - Clock Enable signal is attached to the clock enable pins of flip-flops. A valid clock signal occurs only when Clock Enable Signal attached to CE pin of flip-flops is high on a rising clock edge. If CE is Low, the flip-flops are held in their current state.

Period refers to the sampling period of a particular signal. Please refer to the section below on Multi-rate Realization for more details.

Name	Direction	HDL Type	Type	System Generator Type	Period	Description
dut_out	out	std_logic_vector(15 downto 0)	data	Fix_16_14	1	
dut_out1	out	std_logic_vector(15 downto 0)	data	Fix_16_14	1	
interface_p103_aresetn	in	std_logic	data	-	1	
interface_p103_in1_s_axi_awaddr	in	std_logic	data	-	1	

この資料には、IP のメモリ マップに関するセクションが含まれています。AXI4-Lite インターフェイスの Gateway In または Gateway Out ポートで [Auto assign address offset] をオンにした場合は、インターフェイスがマップされているアドレスを確認できます。

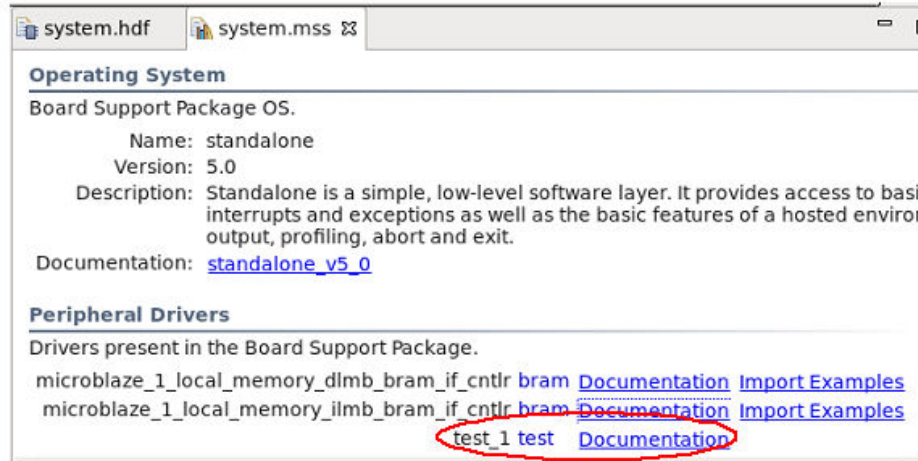
Memory Map

The table below documents the memory map for System Generator design :

Name	Type	Interface Name	Address Offset	Description
dut_in1	Fix_32_2	in	00000000	
dut_in	Fix_32_2	in	00000004	
dut_out	Fix_32_2	out	00000000	
dut_out1	Fix_32_2	out	00000004	

ソフトウェア ドライバーは、自動的に生成され、Vitis™ ソフトウェア プラットフォームにパッケージされます。ソフトウェア ドライバーの資料は、Vitis 環境に含まれます。

図 88: ソフトウェア ドライバーの資料



アドレス生成

自動アドレス生成プロセスでは、次が前提となります。

1. 各 AXI4-Lite ゲートウェイはそれぞれ独自のアドレス オフセットに関連付けられ、32 ビット ワード境界 (4 の倍数) に揃えられます。
2. アドレスは 0 から開始します。
3. アドレスは、ゲートウェイの辞書順にインクリメンタルに割り当てられます。2 つのゲートウェイに同じ名前が付いている場合、任意の名前が付きます。
4. すべての AXI4-Lite ゲートウェイの幅は 32 ビット未満にする必要があります。32 ビット以上の場合はエラーが発生します。
5. AXI4-Lite ゲートウェイの幅が 32 ビット未満であれば、内部レジスタから LSB がテスト対象デバイス (DUT) 内に割り当てられます。
6. 次に、ユーザー指定のオフセット アドレスを管理するための条件を示します。
 - a. ユーザー指定のアドレスはすべて、自動割り当ての前に AXI4-Lite ゲートウェイに割り当てられます。
 - b. 2 つのユーザー指定のアドレスが同じ場合、生成中にのみエラーが発生し、それ以外では無視されます。
 - c. 残りの AXI4-Lite ゲートウェイでアドレスが自動的に割り当てられるように設定されている場合、System Generator でユーザー指定アドレスで残ったアドレスが埋められます。

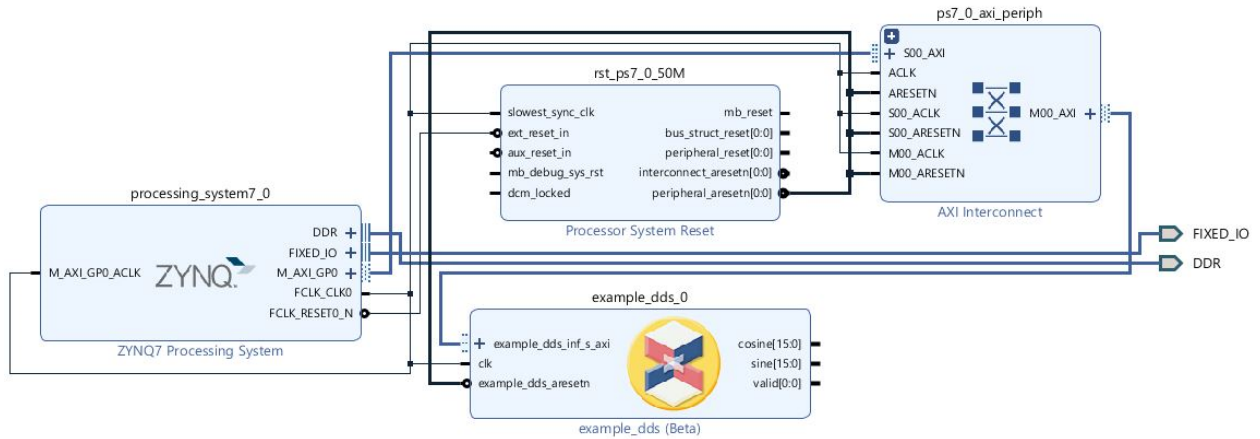
Vivado IDE サンプル プロジェクトの特徴

Vivado® IDE サンプル プロジェクト (example_dds.xpr) を使用すると、System Generator から作成された IP を使用する方法を簡単に理解できます。このプロジェクトは次のようになっています。

1. System Generator から生成された IP がプロジェクトに関連付けられた IP カタログに追加されており、RTL フローおよび IP インテグレーター ベース フローで使用できます。
2. デザインには、example_dds_stub に example_dds_0 という IP の RTL インスタンスが含まれており、IP を RTL フローでどのようにインスタンス化するかを示されています。

3. デザインには、同じ IP を RTL フローにインスタンスエートする example_dds_tb というテストベンチが含まれます。
4. このサンプル プロジェクトで選択されているパーツは Zynq®-7000 SoC で、デザインには Zynq®-7000 サブシステムを含むサンプル IP インテグレーター図が含まれます。その他すべてのパーツには、MicroBlaze ベースのサブシステムが作成されます。

図 89: IP インテグレーターの図



5. 選択されたパーツがサポートされるボードの 1 つと同じ場合は、プロジェクトは同じパーツ設定の最初のボードに設定されます。
6. ブロック デザインをインスタンスエートするラッパーが作成され、最上位として設定されます。



ヒント: IP に関連付けられているインターフェイスの資料には各ブロックの GUI からアクセスできます。資料にアクセスするには、GUI で System Generator IP をダブルクリックし、ダイアログ ボックスで [Documentation] をクリックします。

ソフトウェア ドライバー

ベアメタル ソフトウェア ドライバーは、ゲートウェイに割り当てられたアドレス オフセットに基づいて作成されます。これらのドライバーは、<target_directory>/ip/drivers というフォルダーに保存されます。これらのドライバーを使用するには、Vitis™ 環境の検索パスに <target_directory>/ip を追加する必要があります。

AXI4-Lite インターフェイスにマップされた各 Gateway In に対して、次の 2 つの API が作成されます。

```
/**
 * Write to <Gateway In id> of <design name>. Assignments are LSB-justified.
 *
 * @param InstancePtr is the <Gateway In id> instance to operate on.
 * @param Data is value to be written to gateway <Gateway In id>.
 *
 * @return None.
 *
 * @note <Text from Description control of the Gateway In GUI>
 */
void <Gateway In id>_write(example_dds *InstancePtr, u32 Data);

/**
 * Read from <Gateway In id> of <design name>. Assignments are LSB-justified.
```

```

*
* @param InstancePtr is the phase_valid instance to operate on.
*
* @return u32
*
* @note      Phase Valid Port That Must Be Asserted.
*
*/
u32 <Gateway In id>_read(example_dds *InstancePtr);

```

<Gateway In id>:<design_name>_<gateway_name> で、<design_name> はデザインの VHDL/Verilog 最上位名、<gateway_name> はゲートウェイの省略名です。

Gateway Out でも同様のドライバーが生成されますが、読み出し専用です。

AXI4-Lite インターフェイス生成での既知の問題

ゲートウェイ (Gateway In または Gateway Out) が AXI4-Lite インターフェイスとして設定されているデザインに対しては、テストベンチ生成はサポートされません。

System Generator でのプラットフォーム ベースのアクセラレータの調整

プラットフォーム ベースのアクセラレータは、大型システムを開発しやすくするため、ボトムアップの設計手法を使用します。ボードレベルのインターフェイスをプロセッシング システムに接続する接続プラットフォームと、SoC 内部のデータパスで、接続プラットフォーム デザインによって信号が制御、供給される別個のロジック アクセラレータの 2 つのセクションが作成されます。DSP データパスまたはアクセラレータは、接続プラットフォームと、その外部デバイスへのインターフェイスの自動接続機能を利用できます。

Vivado IP インテグレーターでのデザインの作成をスピードアップさせるには (デザインのアクセラレータ部分が System Generator で開発されるようにするには)、次の手順に従います。

1. Vivado IP インテグレーターでデザインのブロック図 (BD) を作成します。これが接続プラットフォームになります。
2. System Generator に接続プラットフォームをインポートします。
3. System Generator で、デザインのアクセラレータ部分を入力します。
4. System Generator で、IP カタログ フローを使用してアクセラレータ モデルをコンパイルし、元のデザイン (Vivado BD ファイルから) および System Generator モデルの回路を含む Vivado プロジェクトを作成します。

手順 1: Vivado で IP インテグレーター ブロック図 (.bd) として接続プラットフォームを作成

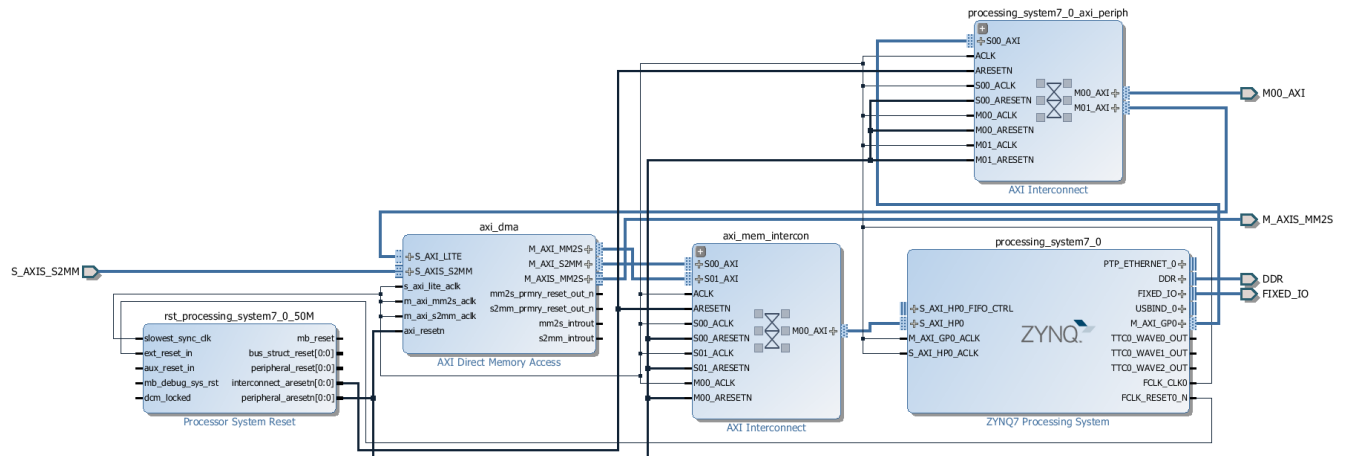
まず、Vivado® IP インテグレーターでプラットフォーム デザインを含むブロック図を作成する必要があります。デザインのアクセラレータ部分を含むプラットフォーム ベース システムとして、コンフィギュラブル サンプル デザイン、リファレンス デザイン、またはカスタム デザインを使用することが可能です。

次の例では、プラットフォーム デザインには Zynq®-7000 プロセッシング システムおよび AXI DMA が含まれています。接続プラットフォーム デザインは、DMA を使用して DDR メモリとデータを送受信し、DDR から受信したデータに DES 暗号化を実行して、暗号化されたデータを DDR に戻します。AXI4-Stream ポート M_AXIS_MM2S および S_AXIS_S2MM (データパス) は、ブロック図 (BD) の外部ポートです。System Generator BD のインポート時に、これらのインターフェイスを System Generator が使用できるようになっています。AXI4-Lite インターフェイス M00_AXI も外部インターフェイスになっていて、アクセラレータ IP に制御インターフェイスがあることを示しています。

IP インテグレーターでのデザインの要件は次のとおりです。

- このシステムは、特定のポートまたはパーツ用にビルドする必要があります。これにより、一部のポートおよびインターフェイスにロケーション属性が割り当てられます。
- デザインのアクセラレータ部分に設定する AXI インターフェイスは、外部インターフェイスにする必要があります。

図 90: AXI インターフェイス



現時点では、プラットフォーム フレームワークに関しては、次のインターフェイスがサポートされています。

表 4: サポートされる AXI インターフェイス

インターフェイス	マスター	スレーブ
AXI4	○	×
AXI4-Lite	○	×
AXI4-Stream	○	○

手順 2: BD ファイルを解析し、ロケーションが割り当てられていないポートおよびインターフェイスを System Generator にインポート

Vivado® IP インテグレーターで作成された BD (ブロック図) ファイルをインポートするため、System Generator で `xilinx.utilities.importBD` ユーティリティを使用します。

このユーティリティはプラットフォーム フレームワークの Vivado プロジェクトおよび System Generator で作成される新しいモデルの名前を読み込みます。System Generator のポートおよび外部インターフェイス (ボード コネクティビティおよび自動化に基づいて、ポートにロケーション属性が設定されていないインターフェイス) のために、プラットフォーム デザインを処理し、デザインのアクセラレータ部分となるサンプル スタブを System Generator で作成します。

コマンドの使用方法:

`xilinx.utilities.importBD` は、プラットフォーム フレームワークの Vivado プロジェクトおよび作成される新しいモデルの名前を読み込みます。プラットフォームを解析して、System Generator ポートやインターフェイスがないか確認し、開発が簡単になるようにサンプルのスタブを作成します。新しいモデル名が指定されていない場合は、名前のないモデルが開きます。

このコマンドに入力ファイルとして指定できるのは、Vivado プロジェクトと `model_name` (オプション) です。

使用方法:

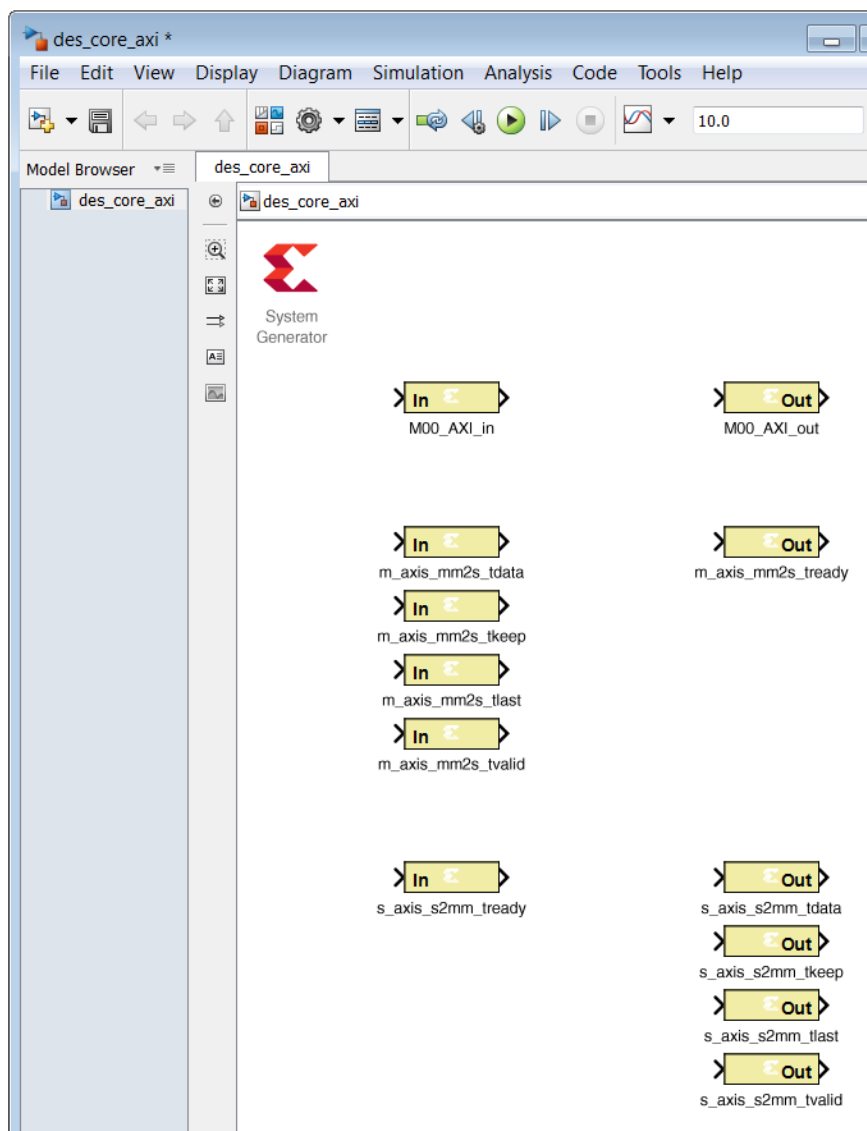
```
xilinx.utilities.importBD('<full_or_relative_path_to_vivado_project_directory>/
<project_name>.xpr', 'mynewmodel')
```

例:

```
xilinx.utilities.importBD('C:\test_importBD\platform.xpr', 'mynewmodel')
xilinx.utilities.importBD('C:\test_importBD\platform.xpr')
```

System Generator で作成されたモデルは次のようになります。

図 91: System Generator モデル



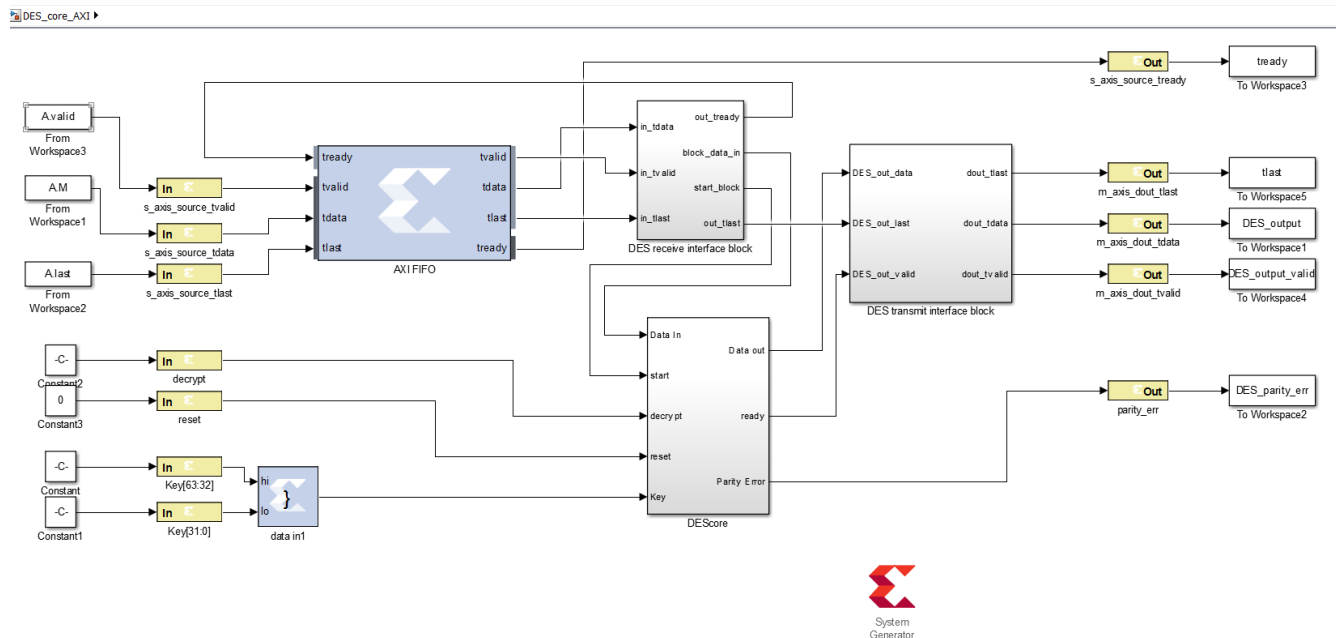
System Generator のモデルには、次の機能があります。

- 各 AXI4-Lite インターフェイスに対し、Gateway In および Gateway Out ブロックが表示されます。この後、AXI4-Lite ゲートウェイをデザインに必要な数だけ複製し追加できます。
- AXI4-Stream インターフェイスに対しては、TDATA、TVALID、TREADY、などの AXI4-Stream ポートが表示されます。
- モデルの System Generator トークンは、[Compilation] ターゲットに [IP Catalog] が設定され、[Part] または [Board] には Vivado プロジェクトと同じサイリンクス デバイスまたはボードが設定されます。

手順 3: System Generator でロジックを BD ソケットに接続

この段階で、System Generator でアクセラレータを作成できます。次の例では、ほかのロジックに接続されていて、ゲートウェイの名前を変更してあります。

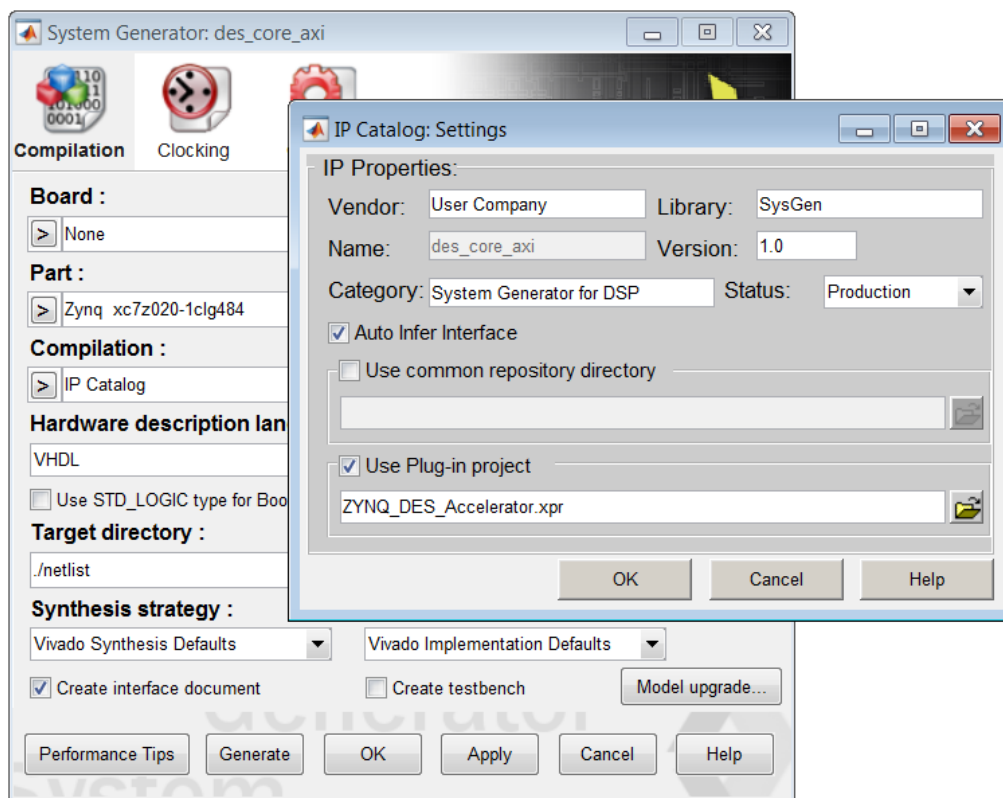
図 92: BD ソケットへの接続



手順 4: アクセラレータ モデル (IP カタログ フロー) をコンパイルして、デザインを作成

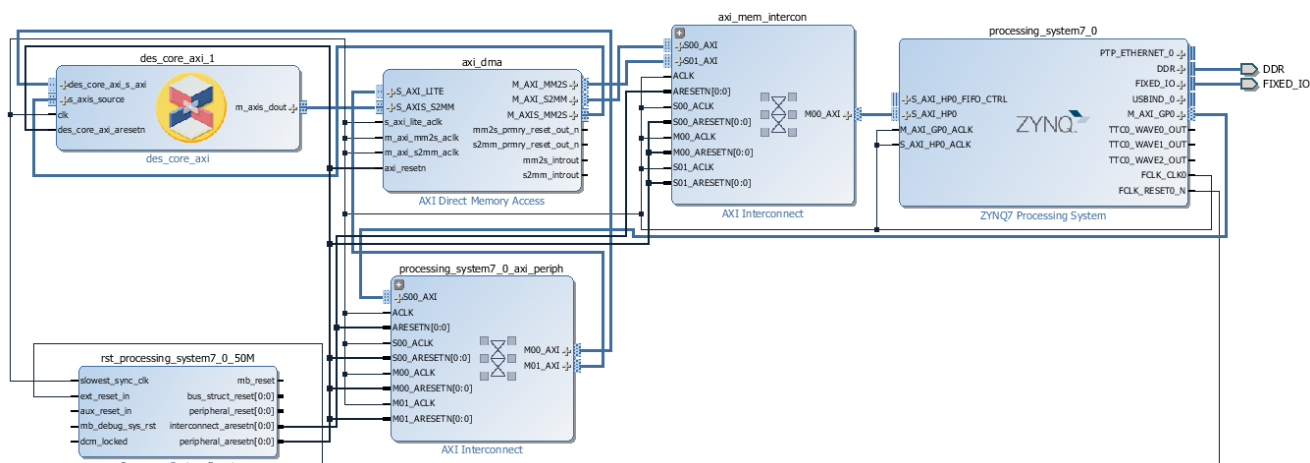
IP カタログ コンパイル フローを使用して、デザインを作成します。IP カタログ フローで System Generator トークンをダブルクリックし、[Settings] ボタンをクリックして、[Use Plug-in project] をオンにしてデザインのインポート元の Vivado® IP インテグレーター プロジェクトに指定します (下の図を参照)。[Generate] をクリックすると、元の Vivado プラットフォーム フレームワーク/システムと、System Generator で作成されたアクセラレータ IP に基づいて、新しい Vivado プロジェクトが、ソフトウェア ドライバーと共に作成されます。このプロジェクトは、System Generator トークンの [Target directory] で設定したディレクトリの下にある ip_catalog というディレクトリに保存され、共通 IP リポジトリにも保存されます。

図 93: ターゲットディレクトリ



この新しいプロジェクトを Vivado で開き、次の図に示すようにデザインをインプリメントできます。System Generator シンボルの付いたブロックは、System Generator で作成されたブロックであることを示しています。

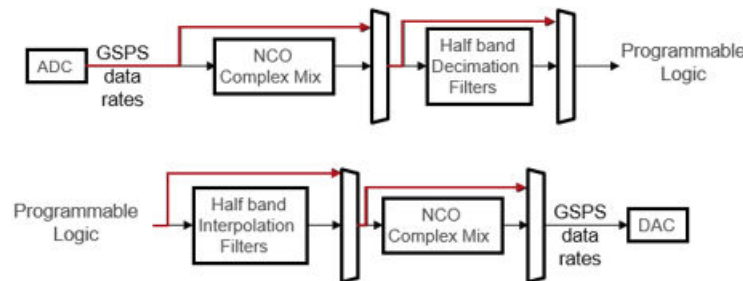
図 94: 新しいプロジェクト



System Generator のスーパー サンプル レート (SSR) ブロックの使用

このセクションで説明するスーパー サンプル レート (SSR) 機能は、すべての ザイリンクス デバイスに適用可能ですが、このセクションはザイリンクス RFSoc デバイス用に記述されています。ダイレクト RF サンプルング データ コンバーターをザイリンクスのテクノロジーに統合したことにより、無線通信、ケーブル アクセス、テストおよび計測、レーダーなどの高性能 RF アプリケーション用の柔軟でフットプリントの小さい低消費電力のソリューションが可能になりました。RFSoc デバイスには、デジタル アップ コンバーター (DUC) およびデジタル ダウン コンバーター (DDC) が組み込まれています。NCO、コンプレックス ミキサー、およびフィルタはハード マクロであり、フィルタの特性は一般的な民生アプリケーション用に最適化されています。

図 95: RFSoc デバイス



RFSoc デバイスは、何が必要かに応じて、2 つの方法で使用できます。

- デバイスに組み込まれている NCO、コンプレックス ミキサー、およびハーフバンド間引き/補間フィルタを使用する。
- デバイスに組み込まれているブロックではデザイン要件が満たされない場合は、上の図に示すようにこれらのブロックをバイパスする。

ブロックをバイパスする場合、デザイン要件を満たすため、System Generator for DSP を使用して、ファブリックに NCO、コンプレックス ミキサー、および DDC ブロックをインプリメントする必要がある場合があります。これには、組み込まれているブロックをバイパスし、System Generator IP をプログラマブル ロジック (PL) のクロック周波数で動作させます。ADC のサンプル レートが GSPS の単位で、PL で MSPS 単位のデータしか処理できない場合は、各データ チャネルに対して各クロック サイクルで複数のサンプルを並列に計算する必要があります。並列サンプルの数は、サンプル周波数とプログラマブル ロジックのクロック周波数の比を計算して求め、SSR パラメーターとして定義します。

SSR とは

SSR は、各クロック サイクルで受け入れる並列サンプルの数を指定するパラメーターです。

SSR の機能

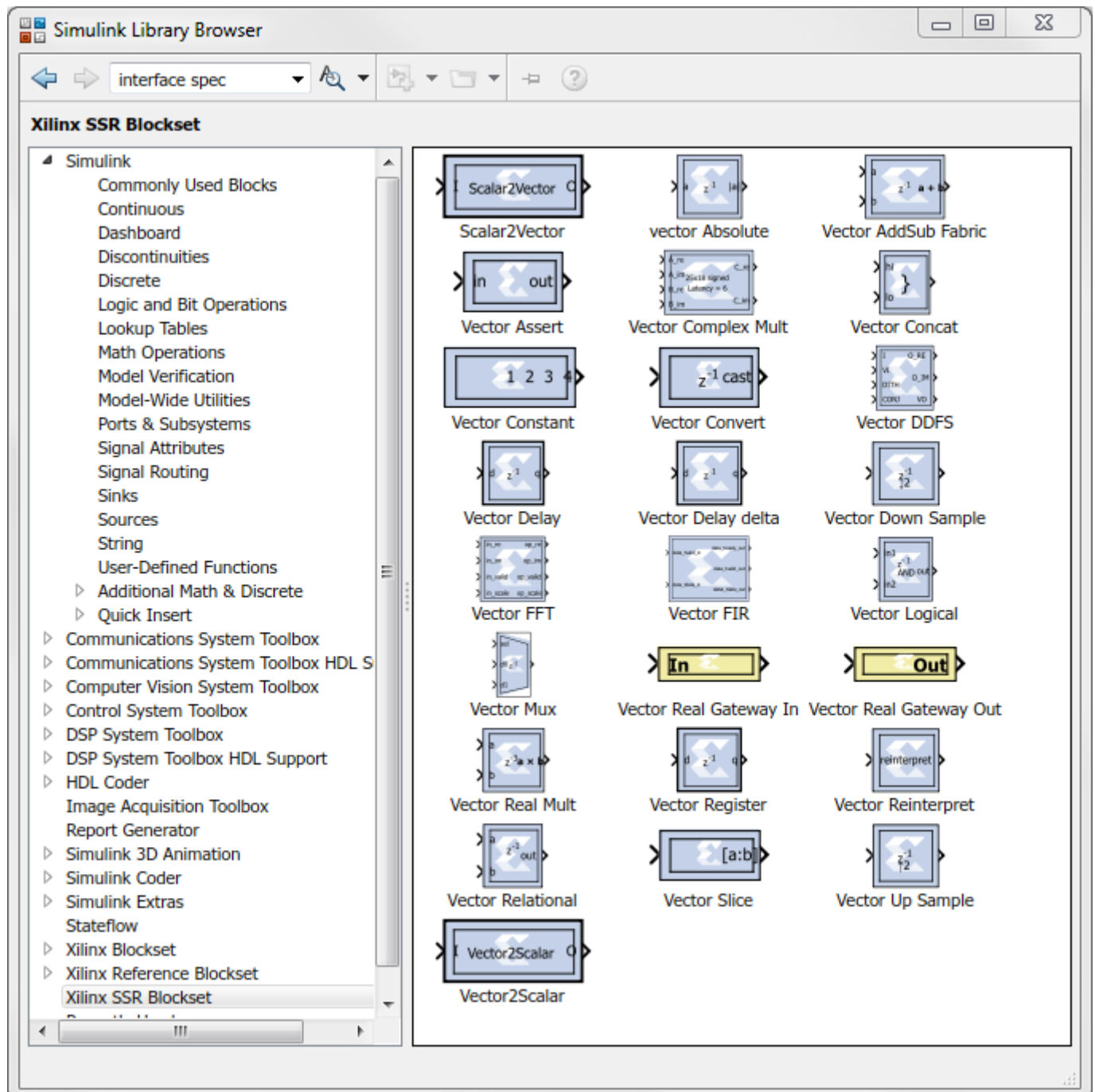
- SSR は、デバイスに組み込まれている RFSoc DUC および DDC を使用できない場合に有益です。
- SSR は、NCO やコンプレックス ミキサーなどのプログラム可能なサブシステムを提供します。ユーザーがブロック マスクにパラメーターを入力すると、System Generator により複数の DDS ブロックを含むサブシステムが構築されます。

- SSR を使用すると、デザインに手動で構造的な変更を加える必要はなくなり、デザイン サイクルを高速化します。

SSR ライブラリ

System Generator には、SSR を処理するためのライブラリ ブロックが別に含まれています。現在のところ、System Generator で 25 個のベクター ブロックがサポートされており、MATLAB® ライブラリ ブラウザーからアクセスできます。

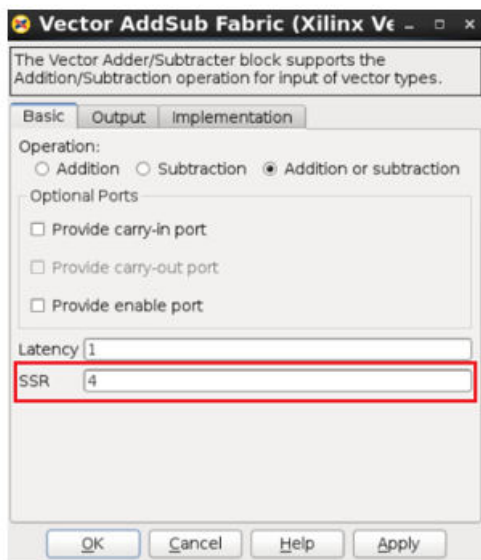
図 96: MATLAB の SSR ブロックセット



SSR パラメーターは、SSR ブロックセットに含まれるすべてのブロックで定義できます。ライブラリからブロックを追加すると、SSR のデフォルト値は 4、SSR の最大値は 256 です。

SSR ブロックセットは、『Vivado Design Suite リファレンス ガイド: System Generator を使用したモデル ベースの DSP デザイン』 ([UG958](#)) に定義されています。

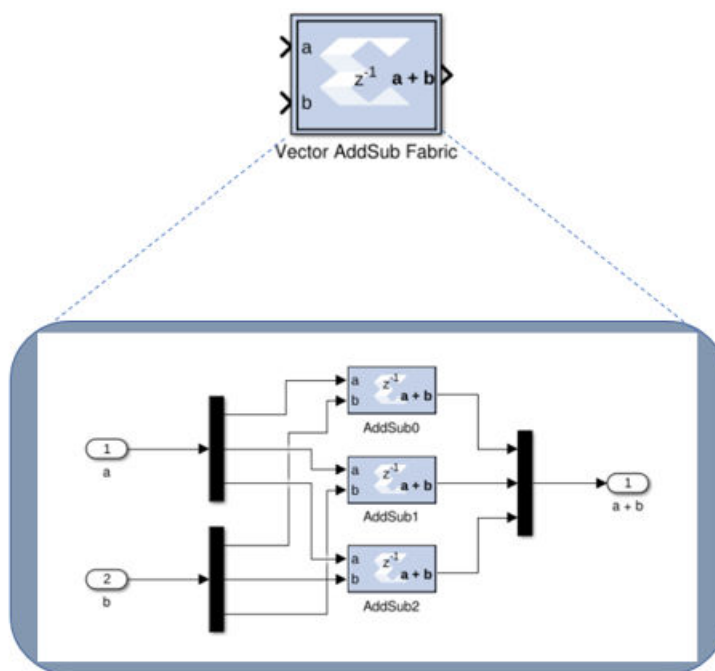
図 97: デフォルト SSR 値



SSR レートにかかわらず、通常の IP ブロックと同様に、限られた数の信号接続を指定する必要があるだけです。SSR ブロック内の並列パス接続はすべて、指定した SSR パラメーターに応じて、System Generator で自動的に処理されます。

たとえば、Vector AddSub ブロックの場合、SSR パラメーターを 3 に変更すると、自動的に次のような内部接続が作成されます。これにより計算用に 3 つの並列パスが作成され、1 つの出力に接続されます。

図 98: Vector AddSub Fabric の例



System Generator での解析の実行

System Generator はビット精度およびサイクル精度のモデリング ツールです。Simulink® でデザインをシミュレーションすることで、その機能を検証できます。ただし、System Generator デザインがターゲットのザイリンクス デバイスにインプリメントされたときに正しく機能することを確認するための解析ツールが System Generator に統合されています。

- タイミング解析: System Generator で生成された HDL ファイルがハードウェアで正しく動作することを確認するには、タイミング クロージャを達成する必要があります。このプロセスを迅速に実行できるようにするため、System Generator にタイミング解析が統合されています。
- リソース解析: System Generator で生成された HDL ファイルがターゲット デバイスに収まることを確認するため、使用されているリソースを解析する必要がある場合があります。このプロセスを迅速に実行できるようにするため、System Generator にリソース解析が統合されています。

System Generator でのタイミング解析	System Generator でのタイミング解析の概要を説明します。
タイミング解析の実行	モデルでタイミング解析を実行する方法を説明します。
タイミング解析結果からモデルへのクロスプローブ	Timing Analyzer の表の行から、バス内の System Generator ブロックをハイライトしながら、Simulink モデルにクロスプローブする方法を説明します。
既存のタイミング解析結果へのアクセス	Timing Analyzer の表に既存のタイミング解析結果を開く方法を説明します。
タイミング違反のトラブルシューティングに関する推奨事項	デザインに発生したタイミング違反の原因を究明する方法を説明します。
System Generator でのリソース解析	System Generator でのリソース解析の概要を説明します。
リソース解析の実行	モデルでリソース解析を実行する方法を説明します。
リソース解析結果からモデルへのクロスプローブ	[Resource Analyzer] ダイアログ ボックスの表から Simulink モデルにクロスプローブし、デザイン内の該当するブロックまたはサブシステムをハイライトする方法を説明します。
既存のリソース解析結果の表示	[Resource Analyzer] ダイアログ ボックスの表に既存のリソース解析結果を開く方法を説明します。
リソース解析を使用した最適化に関する推奨事項	リソース解析結果を使用してデザインのリソース使用率を最適化する方法を説明します。

System Generator でのタイミング解析

System Generator で生成された HDL ファイルがハードウェアで正しく動作することを確認するには、タイミング クロージャを達成する必要があります。このプロセスを迅速に実行できるようにするため、System Generator にタイミング解析が統合されています。

このタイミング解析を使用すると、合成後またはインプリメンテーション後に System Generator で生成された HDL ファイルに対してスタティック タイミング解析を実行できます。合成後またはインプリメンテーション後のネットリストに Vivado® タイミング エンジンを実行した結果と、Simulink® の System Generator モデルを比較する機能も備えているため、Simulink® モデリング環境内でデザインの DSP サブモジュールのタイミング クロージャを達成できます。

コンパイル ターゲット (HDL ネットリストなど) でタイミング解析を実行すると、パスが表に表示され、表の列にタイミング スラックやパス遅延などの情報が表示されます。これがタイミング解析の表です。この表の内容は、列を使用してスラックなどのメトリクスに基づいて並べ替えることができます。また、表のエントリと Simulink モデル間をクロスプローブできるようになっており、モデルのタイミング エラーを検出および修正しやすくなっています。タイミング解析の表から Simulink モデルにクロスプローブするには、表の行を選択します。モデル内の対応するパスがハイライトされます。パスにタイミング違反がある場合は赤でハイライトされ、違反がない場合は緑色でハイライトされます。

タイミング解析の実行

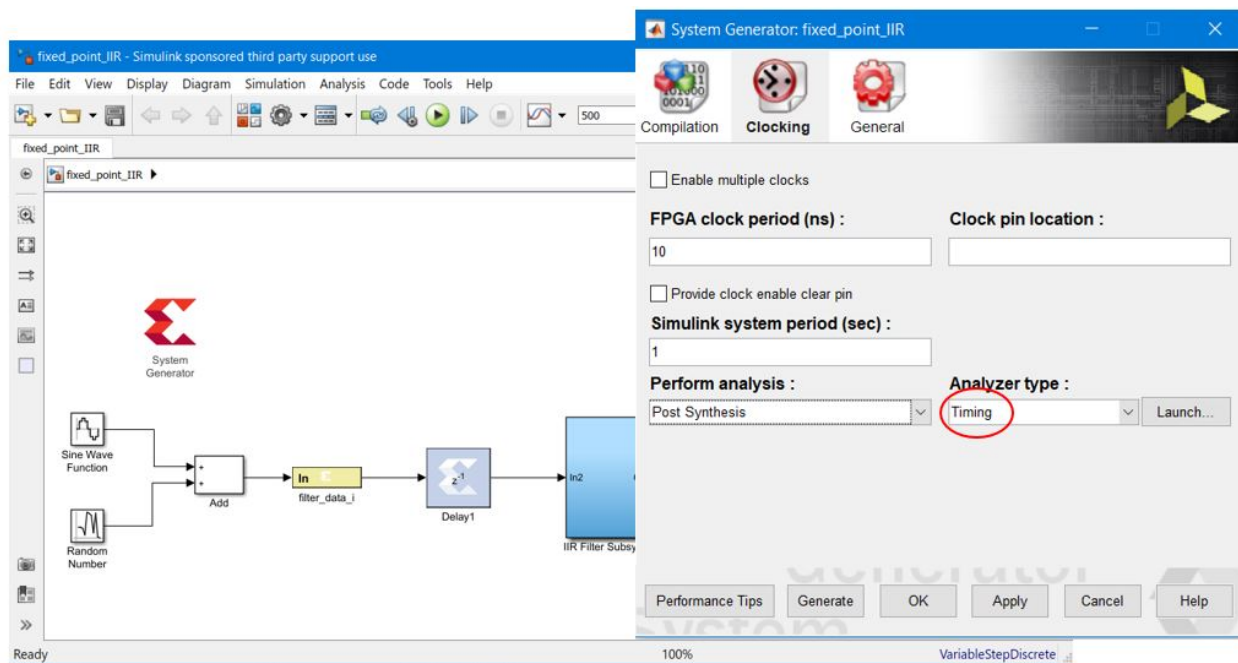
次のコンパイル ターゲットのいずれかを生成するたびに、タイミング解析を実行できます。

- IP カタログ
- ハードウェア協調シミュレーション
- 合成済みチェックポイント
- HDL ネットリスト

System Generator でタイミング解析を実行するには、次の手順に従います。

1. Simulink モデルで System Generator トークンをダブルクリックします。
2. System Generator トークンのダイアログ ボックスで次のように設定します。
 - [Compilation] タブで [Target Directory] を指定します。
 - [Clocking] タブで、ランタイムを重視するか、正確さを重視するかによって、[Perform Analysis] フィールドを [Post Synthesis] または [Post Implementation] に設定します。
 - [Clocking] タブで [Analyzer Type] フィールドを [Timing] に設定します。

図 99: タイミング解析の実行

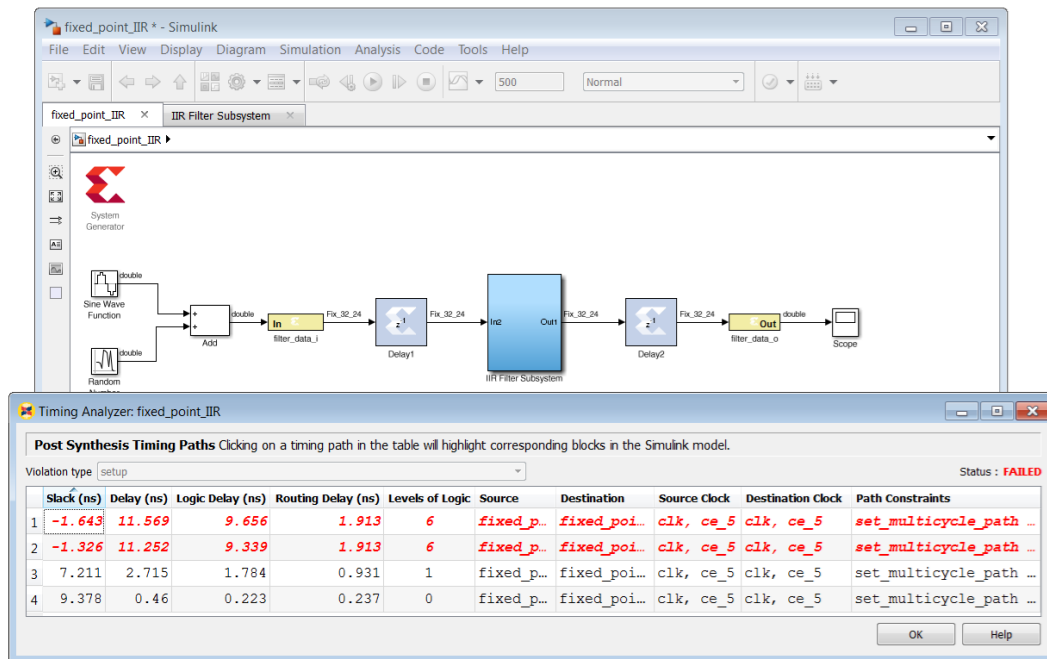


3. System Generator トークンで [Generate] をクリックします。

[Generate] をクリックすると、次が実行されます。

- a. System Generator で選択されているコンパイル ターゲットに必要なファイルが生成されます。タイミング解析では、System Generator によりデザイン プロジェクト用に Vivado がバックグラウンドで起動され、デザインのタイミング制約が Vivado に渡されます。
- b. [Perform Analysis] で [Post Synthesis] を選択したか、[Post Implementation] を選択したかによって、デザインは Vivado で合成またはインプリメンテーションまで実行されます。
- c. Vivado ツールの実行が完了すると、タイミング パスの情報が収集され、Vivado タイミング データベースから指定のファイル フォーマットで保存されます。タイミング パス データが収集されると、Vivado プロジェクトが閉じ、MATLAB®/System Generator プロセスに戻ります。
- d. System Generator はタイミング情報を処理し、タイミング解析の表にタイミング パス情報を表示します (次の図を参照)。

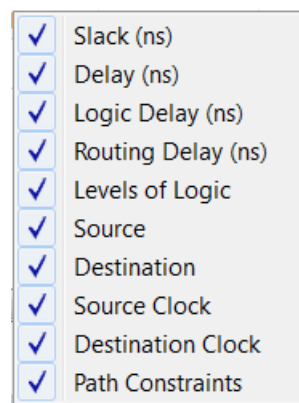
図 100: タイミング解析の表



タイミング解析の表には、次のように表示されます。

- Simulink モデルからのパス (重複しないもののみ) がレポートされます。
- スラック値が最も低いパスから 50 個表示されます。ワースト スラックがのものが 1 番上に表示され、低いものから順に表示されます。
- タイミング違反が発生しているパスのスラック値は負の値であり、赤で表示されます。
- 表示順序は、列見出しをクリックして列の値順に並べ替えることができます。
- 非表示にする列がある場合は、列見出しを右クリックし、非表示にする列のチェック ボックスをオフにします。

図 101: 列の表示/非表示の指定



- マルチサイクル パス制約のあるデザインでは、マルチサイクル パス制約が特定され、[Path Constraints] 列に表示されます。その場合、異なるサンプリング レートを反映させるため、[Source Clock] および [Destination Clock] 列にクロック イネーブル信号が表示されます。

図 102: クロック イネーブル信号

Source Clock	Destination Clock	Path Constraints
clk, ce_3	clk, ce_3	set_multicycle_path -setup 3 -hold 2
clk, ce_4	clk, ce_4	set_multicycle_path -setup 4 -hold 3
clk, ce_6	clk, ce_6	set_multicycle_path -setup 6 -hold 5
clk, ce_12	clk, ce_12	set_multicycle_path -setup 12 -hold 11
clk, ce_12	clk, ce_12	set_multicycle_path -setup 12 -hold 11
clk, ce_12	clk, ce_12	set_multicycle_path -setup 12 -hold 11

- 表の中でパスを選択すると Simulink モデルにクロスプローブでき、Simulink モデルの対応する System Generator ブロックがハイライトされます。[タイミング解析結果からモデルへのクロスプローブ](#) を参照してください。

タイミング解析結果からモデルへのクロスプローブ

タイミング解析の表でパス (行) をクリックすると、Simulink モデルにクロスプローブでき、モデル内の対応する System Generator ブロックがハイライトされます。これにより、タイミング違反が発生しているパスを解析してトラブルシューティングできます。

図 103: タイミング解析の表

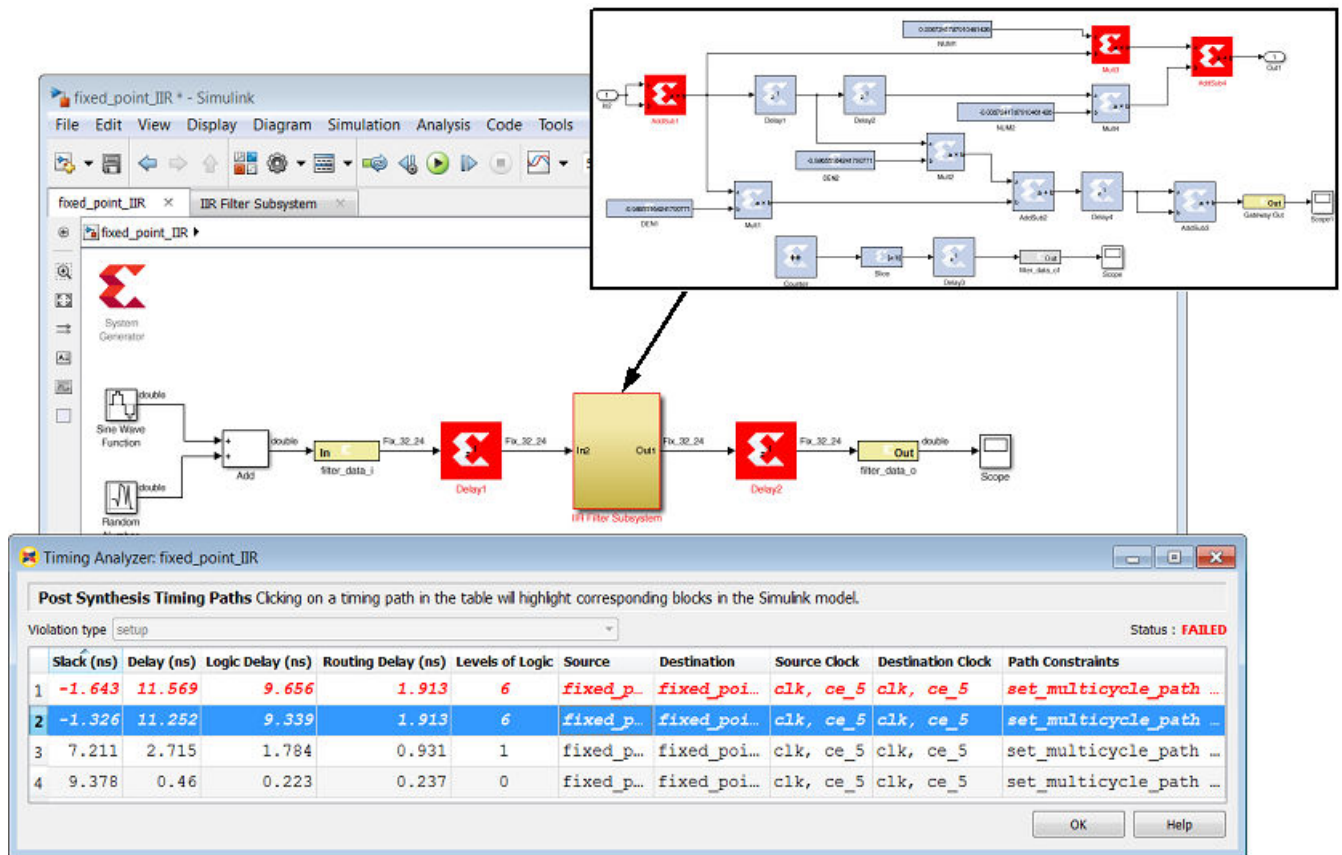
The screenshot shows the Simulink Timing Analyzer window for a model named 'fixed_point_IIR'. The top part displays a Simulink block diagram with various blocks including 'Sine Wave Function', 'Add', 'filter_data_i', 'Delay1', 'IIR Filter Subsystem', 'Delay2', 'filter_data_o', and 'Scope'. The bottom part shows the 'Post Synthesis Timing Paths' table, which lists timing violations. The table has columns for Slack (ns), Delay (ns), Logic Delay (ns), Routing Delay (ns), Levels of Logic, Source, Destination, Source Clock, Destination Clock, and Path Constraints. The first two rows are highlighted in blue, indicating violations.

	Slack (ns)	Delay (ns)	Logic Delay (ns)	Routing Delay (ns)	Levels of Logic	Source	Destination	Source Clock	Destination Clock	Path Constraints
1	-1.643	11.569	9.656	1.913	6	fixed_p...	fixed_poi...	clk, ce_5	clk, ce_5	set_multicycle_path ...
2	-1.326	11.252	9.339	1.913	6	fixed_p...	fixed_poi...	clk, ce_5	clk, ce_5	set_multicycle_path ...
3	7.211	2.715	1.784	0.931	1	fixed_p...	fixed_poi...	clk, ce_5	clk, ce_5	set_multicycle_path ...
4	9.378	0.46	0.223	0.237	0	fixed_p...	fixed_poi...	clk, ce_5	clk, ce_5	set_multicycle_path ...

クロスプローブすると、モデルに次のものが表示されます。

- モデル内でタイミング違反のあるパスのブロックは赤でハイライトされ、タイミング違反のないパス ([Slack] の値が正のパス) のクロックは緑色でハイライトされます。
- ハイライトされているパスのブロックがサブシステム内にある場合は、そのサブシステムが赤でハイライトされるので、サブシステムを展開してその中のブロックを調べることができます。

図 104: クロスプローブ



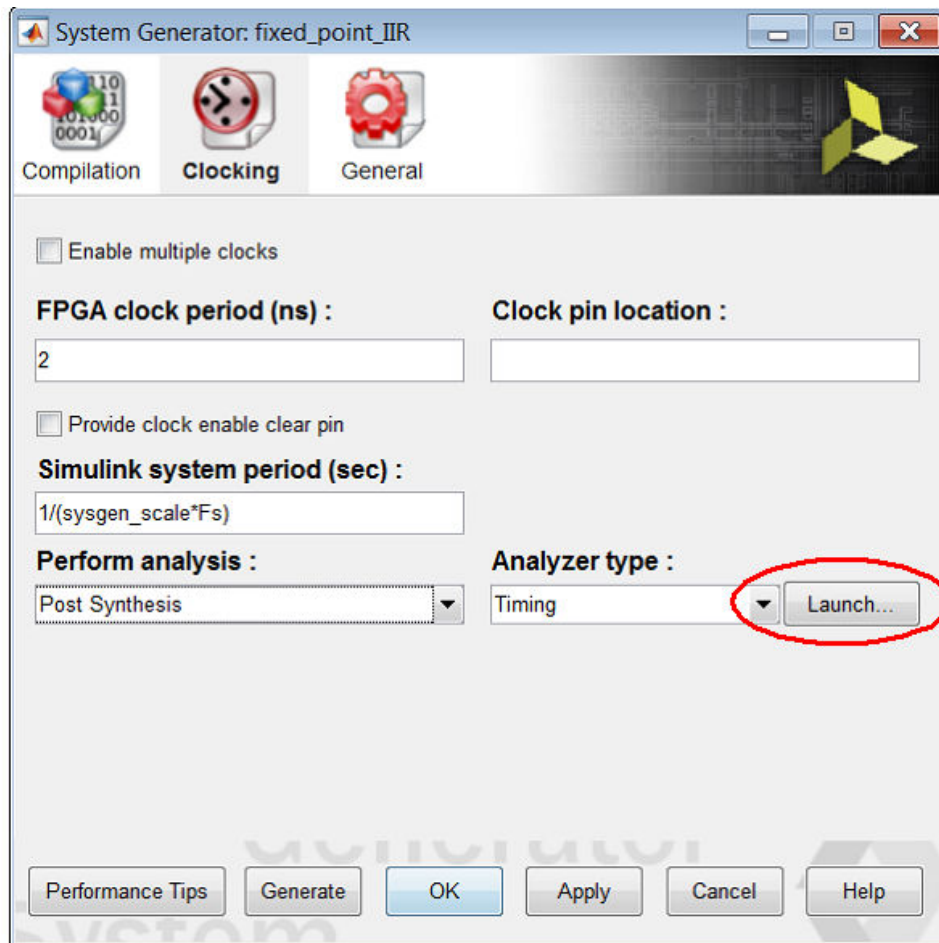
- クロスプローブするパス (表の行) を選択すると、通常はパスの終点にあるデスティネーション ブロックがハイライトされ、そのデスティネーション ブロックを含むサブシステムがモデルの前面に表示されます。このため、ソース ブロックが別のサブシステムにある場合は、ハイライトされたソース ブロックが隠れてしまうことがあります。このような場合にソース ブロックを表示するには、表の [Source] 列のパスをクリックします。これで、そのソース ブロックを含むサブシステムがモデルの前面に表示されます。ほかの列のパスを選択すると、デスティネーション ブロックを含むサブシステムが前面に表示されます。

既存のタイミング解析結果へのアクセス

System Generator トークンのダイアログ ボックスの [Launch] タブにある [Clocking] ボタンをクリックすると、モデルの既存のタイミング解析結果が表形式で表示されます。ダイアログ ボックスの [Target directory] タブの [Compilation] で指定されているディレクトリが Timing Analyzer で読み出し可能で、[Analyzer Type] が [Timing] に設定されていることを確認してください。Simulink モデルでタイミング解析を既に実行していて、Simulink モデルを前回の実行から変更していない場合にのみ、この操作が可能です。

[Launch] をクリックすると、[Target directory] のオプション設定 ([Perform analysis] または [Post Synthesis]) にかかわらず、[Post Implementation] で指定されたディレクトリに保存されているタイミング結果が Timing Analyzer の表に表示されます。

図 105: [Launch] ボタン



次のコマンドを MATLAB® コマンド プロンプトに入力して、Timing Analyzer の表に既存のタイミング解析を表示することもできます。

```
xlAnalyzeTiming(<mdl_hdl>, <netlist_dir>)
```

<mdl_hdl> は Simulink® モデル ハンドル (最上位デザイン ハンドル)、<netlist_dir> は System Generator トークンのダイアログ ボックスの [Target directory] で指定されているディレクトリです。

タイミング違反のトラブルシュートに関する推奨事項

タイミング違反のトラブルシュートには次の点が推奨されます。

- タイミング解析を繰り返して実行する場合、インプリメンテーション後の解析よりも、合成後の解析の方が時間が掛かりません。

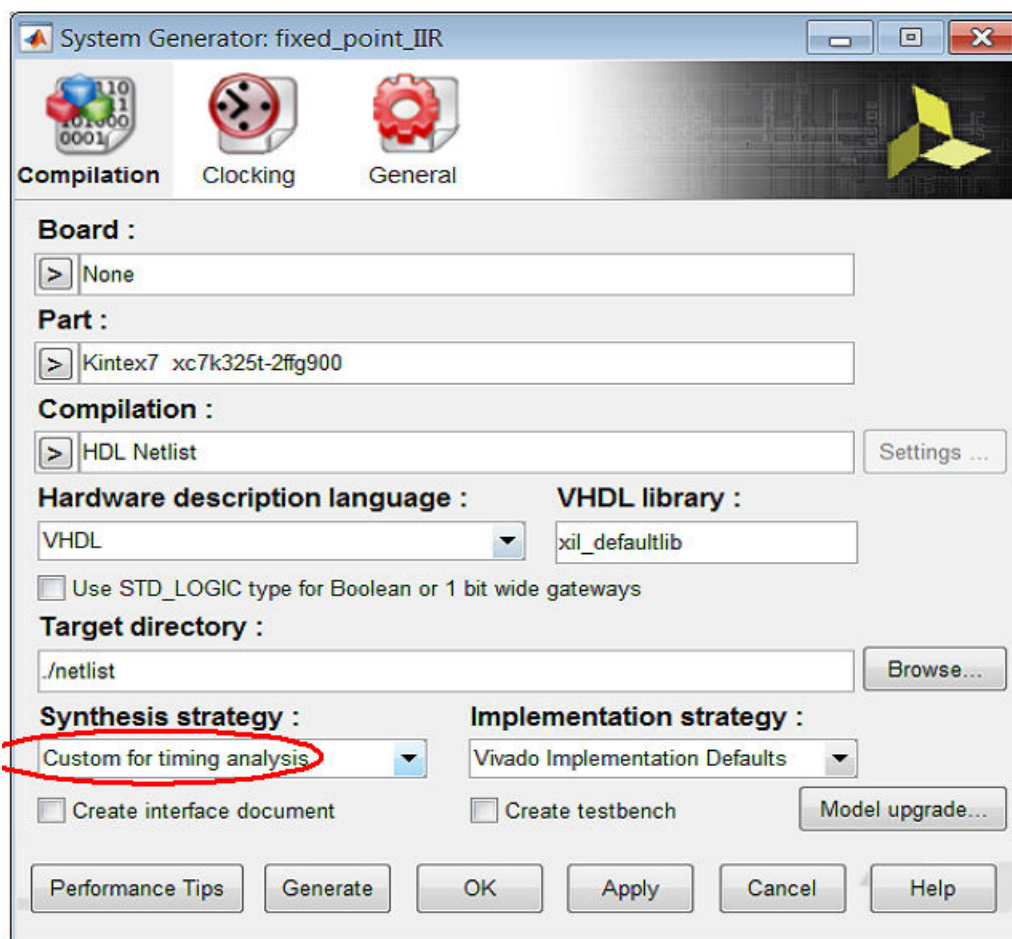
- Vivado 合成プロセス中にロジックが最適化された後は、Vivado データベースでマージされたロジックに関する情報はツールでは保存されません。ロジックがマージされていたり共有されていると、Vivado タイミング パスから Simulink モデルに正確にクロスプローブするのが困難になる場合があります。このため、マージされたロジックおよび共有ロジックを制御するには、カスタムの Vivado 合成ストラテジを作成することをお勧めします。

Vivado でのカスタム合成ストラテジの作成方法は、『Vivado Design Suite ユーザー ガイド: Vivado IDE の使用』(UG893)の[このセクション](#)を参照してください。

マージおよび共有されたロジックを Vivado IDE で制御するには、デフォルトの Vivado 合成ストラテジに次の変更を加える必要があります。

1. Vivado IDE で次の合成オプションを設定します。
 - 合成オプション `-keep_equivalent_registers` を選択します。
 - 合成オプションで `-resource_sharing` を `off` に設定します。
2. 新しい合成ストラテジを保存して Vivado IDE を終了します。
3. System Generator でデザインを生成する前に、System Generator トークンのダイアログ ボックスで [Synthesis strategy] でカスタム合成ストラテジ ([Custom for timing analysis]) を選択します。

図 106: [Synthesis strategy] に [Custom for timing analysis] を選択



System Generator でのリソース解析

System Generator で生成された HDL ファイルがターゲット デバイスに収まることを確認するため、使用されているリソースを解析する必要がある場合があります。このプロセスを迅速に実行できるようにするため、System Generator にリソース解析が統合されています。

リソース解析を実行すると、モデルで使用されるルックアップ テーブル (LUT)、レジスタ、DSP48 (DSP)、およびブロック RAM (BRAM) の数を判断できます。解析は合成後またはインプリメンテーション後に実行され、Vivado® ツールで使用されるリソースと、Simulink® の System Generator モデルを比較できます。このため、デザインでリソースが過剰に使用されているエリアがないかを調べるために Simulink モデリング環境を離れる必要がありません。

コンパイル ターゲット (IP カタログなど) でリソース解析を実行すると、ブロックの LUT、レジスタ、DSP、およびブロック RAM リソースの使用率が表形式で表示されます。これはリソース解析の表です。この表の内容は、列を使用して DSP などのメトリクスに基づいて並べ替えることができます。また、表のエントリと Simulink モデル間をクロスプロブできるようになっており、モデルのタイミング エラーを検出および修正しやすくなっています。リソース解析の表から Simulink モデルにクロスプロブするには、表の行を選択します。モデルの対応するブロックまたは階層が黄色くハイライトされます。

リソース解析の実行

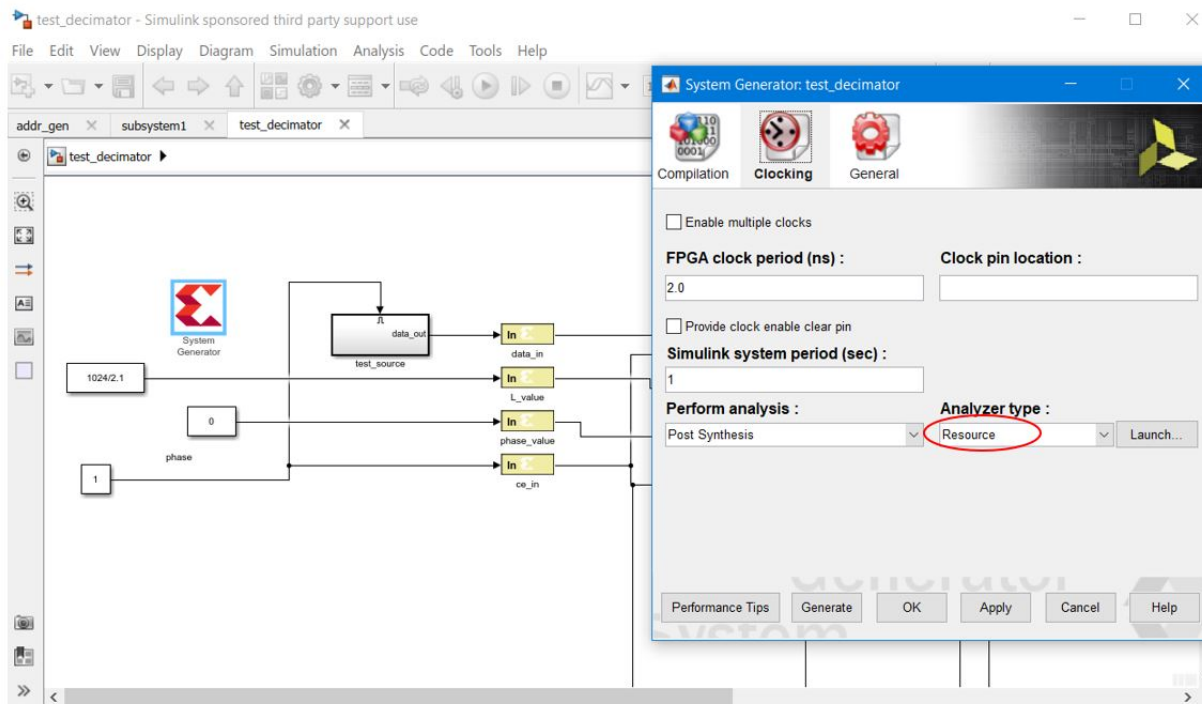
次のコンパイル ターゲットのいずれかを生成すると、いつでもリソース解析を実行できます。

- IP カタログ
- ハードウェア協調シミュレーション
- 合成済みチェックポイント
- HDL ネットリスト

System Generator でリソース解析を実行するには、次の手順に従います。

1. Simulink モデルで System Generator トークンをダブルクリックします。
2. System Generator トークンのダイアログ ボックスで次のように設定します。
 - a. [Compilation] タブ:
 - デザインをインプリメントする [Part] を指定します。
注記: [Board] ではなく [Part] を選択する場合は、[Part] フィールドに [Board] で選択したパーツ名を入力します。
 - [Compilation] でターゲットの 1 つを選択します。
System Generator では、[Compilation] で選択したどのターゲットに対してもリソース解析を実行できます。
 - [Target Directory] を指定します。
 - b. [Clocking] タブ:
 - ランタイムを重視するか、正確さを重視するかによって、[Perform Analysis] フィールドを [Post Synthesis] または [Post Implementation] に設定します。
 - [Analyzer type] を [Resource] に設定します。

図 107: [Resource Analyzer] ダイアログ ボックス

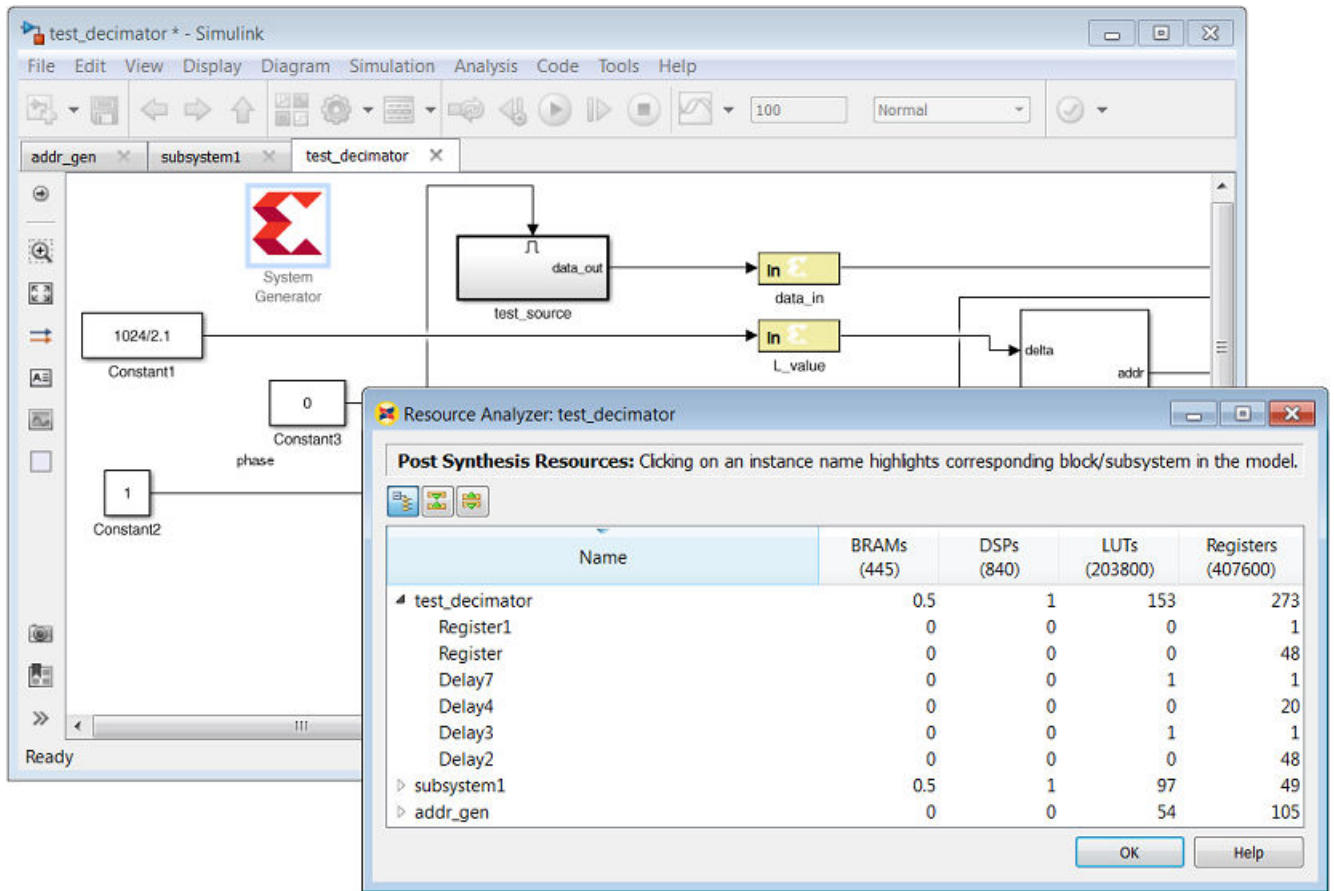


3. System Generator トークンで [Generate] をクリックします。

[Generate] をクリックすると、次が実行されます。

- a. System Generator で選択されているコンパイル ターゲットに必要なファイルが生成されます。System Generator では、リソース解析のために Vivado がバックグラウンドで起動されます。
- b. [Perform analysis] で [Post Synthesis] を選択したか、[Post Implementation] を選択したかによりますが、デザインは Vivado で合成またはインプリメンテーションまで実行されます。
- c. Vivado ツールの実行が完了すると、Vivado データベースからリソース使用率データが収集され、指定ファイル フォーマットでターゲット ディレクトリに保存されます。リソース使用率データが収集されると、Vivado プロジェクトが閉じ、MATLAB/System Generator プロセスに戻ります。
- d. System Generator でリソース使用率データが処理され、[Resource Analyzer] ダイアログ ボックスの表にその情報が表示されます (次の図を参照)。

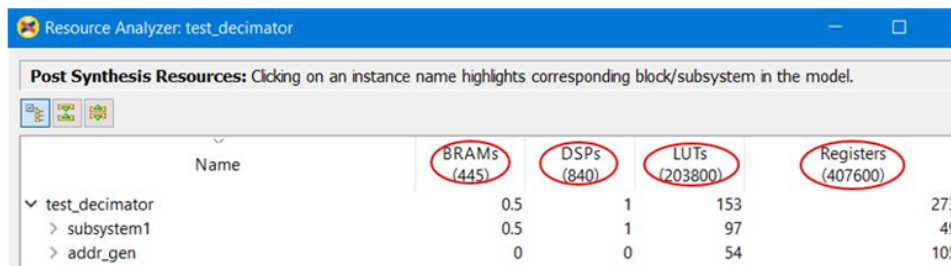
図 108: [Resource Analyzer] ダイアログ ボックス



[Resource Analyzer] ダイアログ ボックスの表には、次のように表示されます。

- ダイアログ ボックスのヘッダー部分に、Vivado からリソース使用率データが収集された Vivado デザイン段階が表示されます。これは [Post Synthesis] または [Post Implementation] です。
- 表内のツールバーには、リソース数の表示方法を変更するボタンがあります。
 - [Hierarchical/Flat Display]: 階層表示またはフラットなリスト表示に切り替えます。
 - [Collapse All]: デザイン階層を非展開にして最上位オブジェクトのみを表示します。
 - [Expand All]: 階層をすべて展開し、デザインの各サブシステムおよび各ブロックで使用されるリソースを表示します。
- 表の各列の見出しには、デザインのターゲットに指定されているサイリンクス デバイスで使用可能なリソースの合計数が表示されます。次の例では Kintex-7 FPGA がターゲットになっています。

図 109: Kintex-7 のリソース解析レポート



Name	BRAMs (445)	DSPs (840)	LUTs (203800)	Registers (407600)
test_decimator	0.5	1	153	273
> subsystem1	0.5	1	97	49
> addr_gen	0	0	54	105

- 表には、デザインの各サブシステムおよびブロックとが階層別にリストされ、リソース タイプ別の数が示されます。
- BRAM: ブロック RAM および FIFO プリミティブ。ブロック RAM (BRAM) は、次のようにカウントされます。

表 5: BRAM 数

プリミティブ タイプ	ブロック RAM 数
RAMB36E	1
FIFO36E	1
RAMB18E	0.5
FIFO18E	0.5

プリミティブのバリエーション (RAM36E1 や RAM36E2 など) もみな同じ方法でカウントされます。

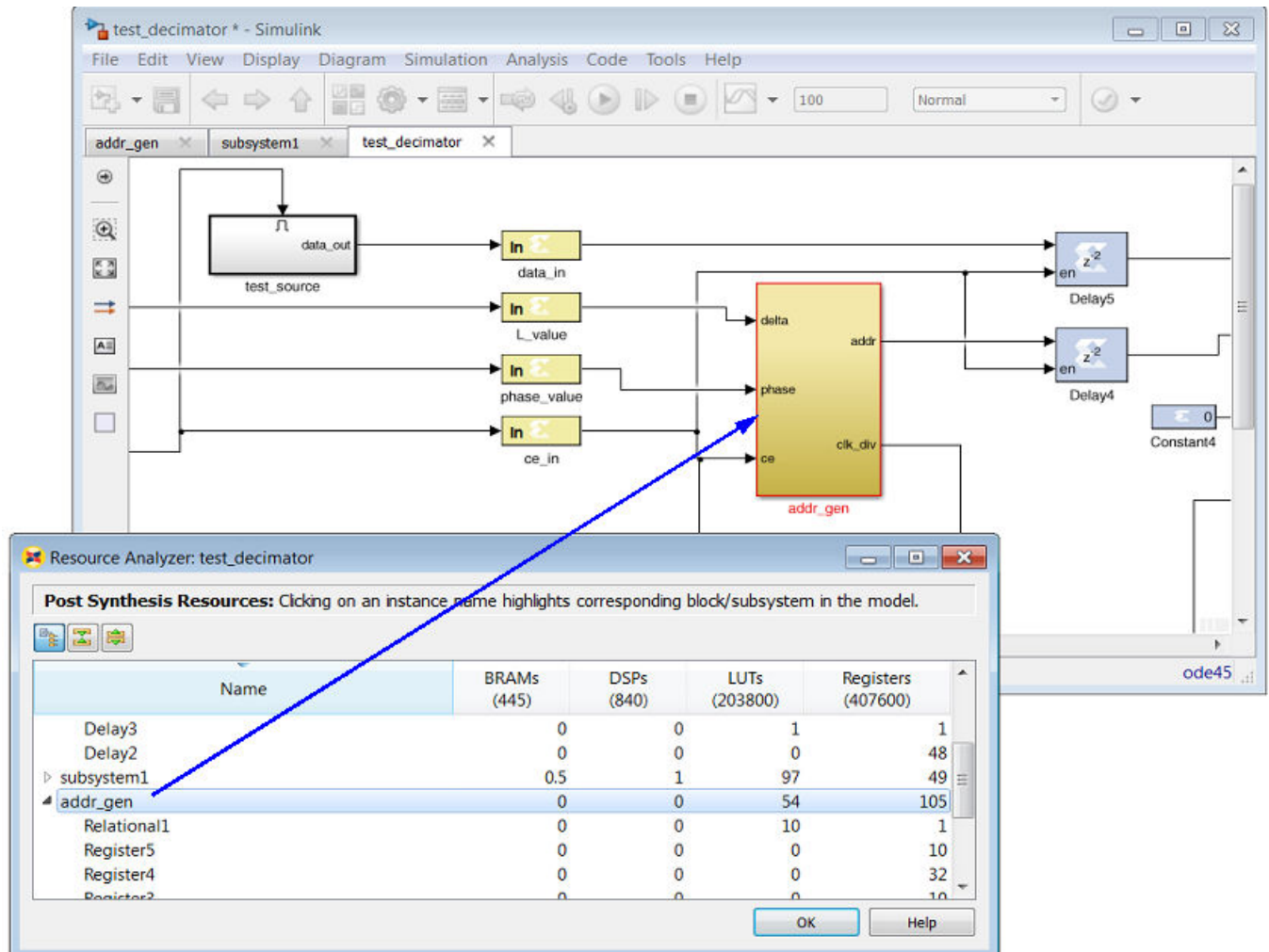
BRAM の総数 = (RAMB36E の数) + (FIFO36E の数) + 0.5 (RAMB18E の数 + FIFO18E の数)

- [DSPs]: DSP48 プリミティブ (DSP48E、DSP48E1、または DSP48E2)。
- [Registers]: レジスタおよびフリップフロップ。「FD*」 (FDCE、FDPE、FDRE、FDSE など) および「LD*」 (LDCE、LDPE など) で始まるプリミティブは、すべて [Registers] に含まれます。
- [LUTs]: すべて LUT タイプ。
- 表示順序は、列見出しをクリックして列の値順に並べ替えることができます。
- 表の中で行を選択すると Simulink モデルにクロスプローブでき、Simulink モデルの対応する System Generator ブロックがハイライトされます。[リソース解析結果からモデルへのクロスプローブ](#) を参照してください。

リソース解析結果からモデルへのクロスプローブ

[Resource Analyzer] ダイアログ ボックスの表でブロックまたはサブシステムをクリックすると、Simulink® モデルにクロスプローブでき、モデル内の対応する System Generator ブロックまたはサブシステムがハイライトされます。クロスプローブは、特定タイプのリソースを使用してインプリメントされたブロックやサブシステムを特定するのに便利です。

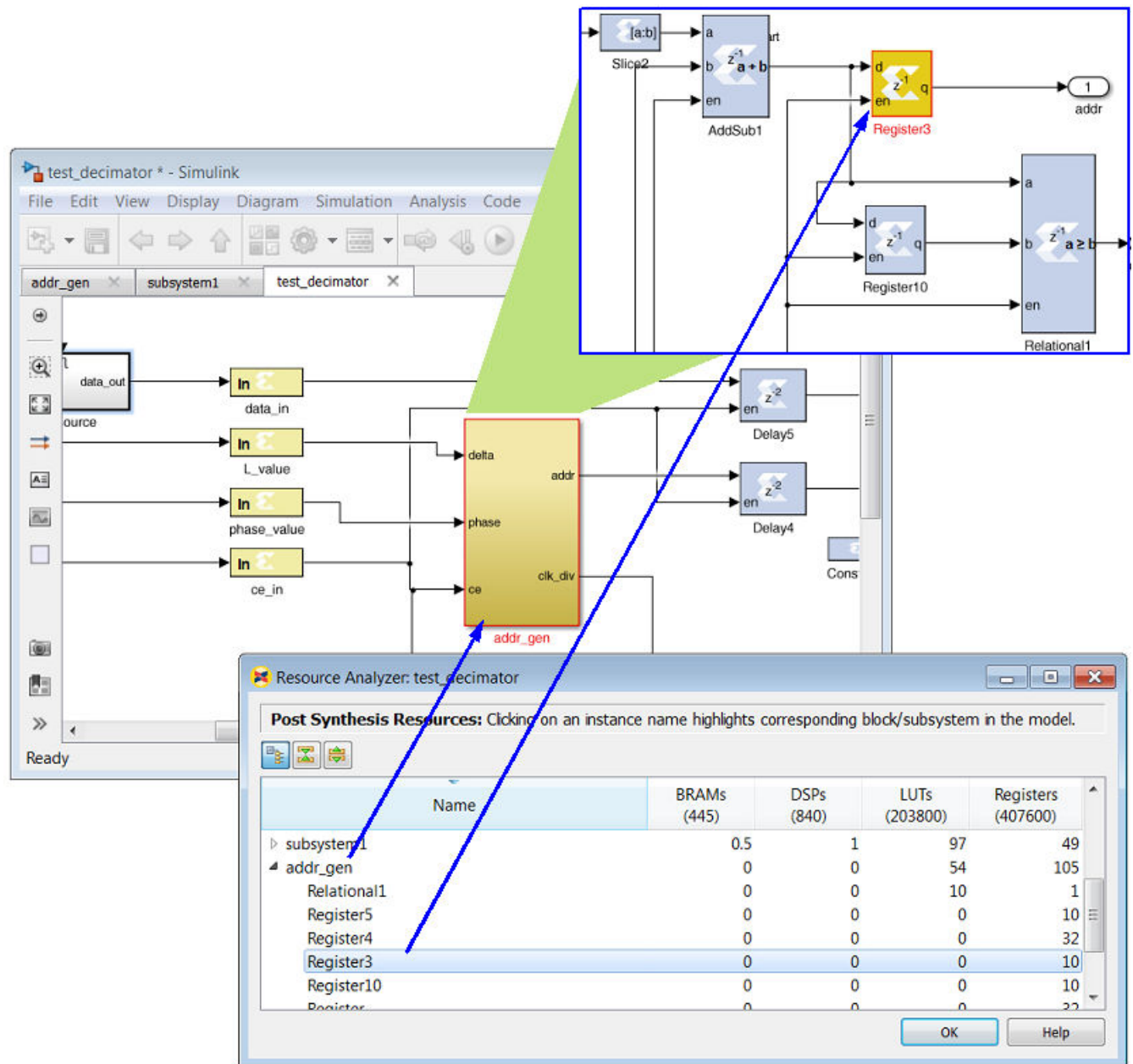
図 110: [Resource Analyzer] ダイアログ ボックス



クロスプローブすると、モデルに次のものが表示されます。

- 表で選択されたブロックが黄色くハイライトされ、赤く囲われます。
- 表で選択されたブロックまたはサブシステムが上位のサブシステム内にある場合、ブロックだけでなくその上位サブシステムも赤色でハイライトされます。

図 111: 選択したサブシステム

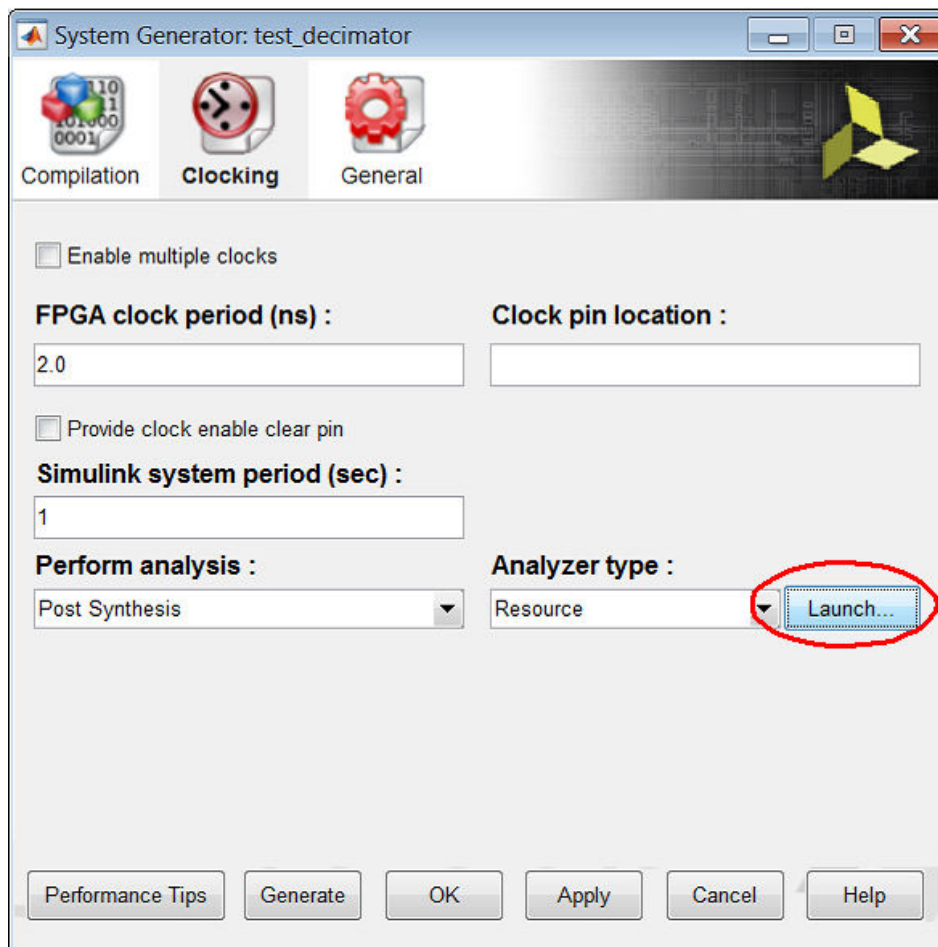


既存のリソース解析結果の表示

System Generator トークンのダイアログ ボックスの [Clocking] タブにある [Launch] ボタンをクリックすると、モデルの既存のリソース使用率結果を表形式で表示する [Resource Analyzer] ダイアログ ボックスが開きます。System Generator トークンのダイアログ ボックスの [Compilation] タブで [Target directory] に指定されているディレクトリが Resource Analyzer で読み出し可能であること、[Analyzer type] が [Resource] に設定されていることを確認してください。Simulink モデルで解析を既に実行していて、Simulink モデルを前回の実行から変更していない場合にのみ、この操作が可能です。

[Launch] をクリックすると、[Target directory] の設定 ([Compilation] または [Perform analysis]) にかかわらず、[Post Synthesis] タブの [Post Implementation] で指定されたディレクトリに保存されているリソース使用率が [Resource Analyzer] ダイアログ ボックスに表示されます。

図 112: [Launch] ボタン



[Resource Analyzer] ダイアログ ボックスに既存のリソース使用率結果を表示するには、MATLAB® コマンド プロンプトに次のコマンドを入力します。

```
>> xlAnalyzeResource(get_param('model_name','handle'), './netlist')
```

- `get_param('model_name','handle')` はモデル ハンドルを取得します。

注記: `model_name` はモデル名です。

- `netlist` ディレクトリへのパスには、絶対パスを使用するか、`netlist` ディレクトリがあるディレクトリからこの API を使用する場合は相対パス (`./netlist` など) を使用できます。

リソース解析を使用した最適化に関する推奨事項

リソース解析を使用してデザインのリソース使用率を最適化する際は、次のことを推奨します。

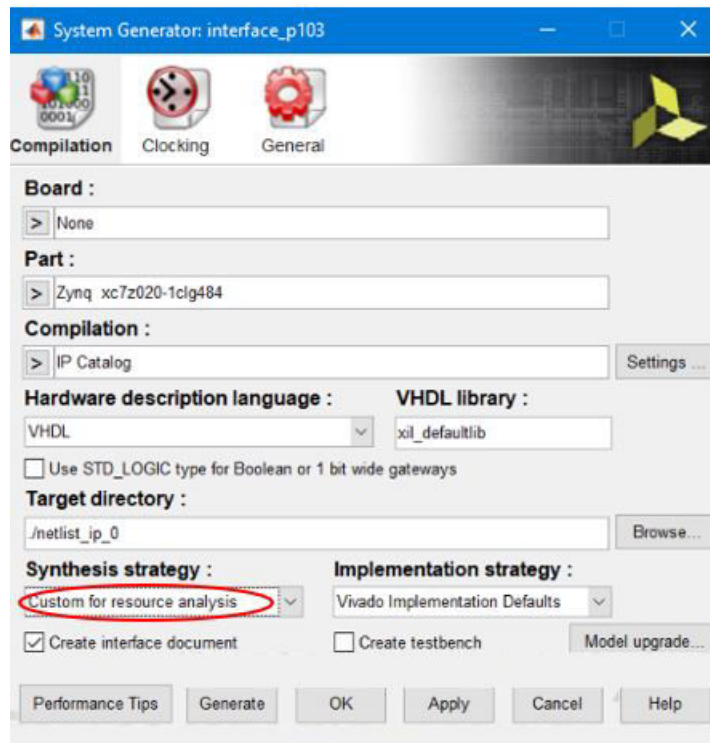
- リソース解析をすばやく実行するには、インプリメンテーション後の解析よりも、合成後の解析が適しています。
- Vivado 合成プロセス中にロジックが最適化された後は、Vivado データベースでマージされたロジックに関する情報はツールでは保存されません。ロジックがマージされていたり共有されていると、Vivado リソース データから Simulink モデルへ正確にクロスプローブするのが難しい場合があります。このため、マージされたロジックおよび共有ロジックを制御するには、カスタムの Vivado 合成ストラテジを作成することをお勧めします。

Vivado IDE でのカスタム合成ストラテジの作成方法は、『Vivado Design Suite ユーザー ガイド: Vivado IDE の使用』 (UG893) の[このセクション](#)を参照してください。

マージおよび共有されたロジックを Vivado IDE で制御するには、デフォルトの Vivado 合成ストラテジに次の変更を加える必要があります。

- Vivado IDE で次の合成オプションを設定します。
 - 合成オプション -keep_equivalent_registers を選択します。
 - 合成オプションで -resource_sharing を off に設定します。
- 新しい合成ストラテジを保存して Vivado IDE を終了します。
- System Generator でデザインを生成する前に、System Generator トークンのダイアログ ボックスで [Synthesis strategy] でカスタム合成ストラテジ ([Custom for timing analysis]) を選択します。

図 113: 合成ストラテジ



ハードウェア協調シミュレーションの使用

System Generator には、ハードウェア協調シミュレーションの機能が含まれ、FPGA で実行されるデザインを直接 Simulink® シミュレーションに読み込むことができます。これで、順次ソフトウェアとして Simulink でシミュレーションしていた System Generator デザイン全体 (または一部) を FPGA でパラレルで実行できるようになり、シミュレーション時間を大幅に短縮できます。このフローを使用すると、より大きなデータ セット、またはより多くのテスト ベクターを送信し、インプリメントされたロジックの機能を詳細にテストできます。コードのテスト範囲を広げることで、より多くのコーナー ケースを検証でき、ロジックのデザイン バグを特定するのに役立ちます。Simulink モデルのコンパイル済みの協調シミュレーション ブロックへの入力データは、ターゲットの FPGA に送信され、1 つのトランザクションまたはトランザクションのバーストとして、指定クロック サイクル間パラレルに実行され、モデルの協調シミュレーション出力にリードバックされます。

ハードウェア協調シミュレーションには、バーストと非バースト (標準) という 2 つのコンパイル タイプがあります。バースト モードの場合は、パフォーマンスがかなり速くなります。コンパイル済みの協調シミュレーション ターゲットの各入力へのチャネルが開き、データの packets がそのオープン チャネルに送信され、それに続いて残りの入力のすべてにバーストします。FPGA デザインは、データを取り込むのに十分なクロック サイクル間並列に実行され、ターゲット出力はチャネル化されてバースト読み出しされます。バーストだと FPGA からの大量のデータを送受信するためのオーバーヘッドが少なく済みます。ただし、バースト モードは、ハードウェア協調シミュレーション ターゲットの MATLAB® スクリプト ベースのハードウェア協調シミュレーションを介してのみサポートされ、Simulink 内では使用されません。大量のデータ ベクターは、協調シミュレーション ターゲットの機能をテストするため、スクリプトにできます。サンプル スクリプトはコンパイルの一部として返されます。非バースト モードの場合はパフォーマンスは低くなりますが、元の System Generator デザイン階層の代わりにコンパイル済みの協調シミュレーション ブロックを Simulink 内で使用できます。

注記: ハードウェア協調シミュレーションでは、複数のクロックを含むデザインはサポートされません。

JTAG およびイーサネットの 2 タイプの物理インターフェイスを使用して協調シミュレーションと通信できるよう、ボードはサポートされています。Vivado® ツールでプロジェクト ターゲットになっている JTAG を認識可能なほとんどのボードで、JTAG ベースの通信が可能です。サイリンクス パートナーのボードの場合は、各パートナーのウェブ サイトからダウンロードし、Vivado Design Suite の一部としてインストールできます。またカスタム ボードも作成できます。詳細は、『Vivado Design Suite ユーザー ガイド: システム レベル デザイン入力』(UG895) の付録 A 「ボード インターフェイス ファイル」を参照してください。System Generator でボードを認識させる方法、board.xml ファイルに必要な最小限のタグを設定する方法については、[System Generator でのボード サポートの指定](#)を参照してください。

[Hardware Co-Simulation] コンパイル オプションを選択してコンパイルすると、選択した通信インターフェイスに基づいてビットストリームが自動的に作成され、ブロックに関連付けられます。

現在のところ、System Generator では、次のボードでハードウェア協調シミュレーションがサポートされています。

- JTAG ハードウェア協調シミュレーション - ボードで JTAG ハードウェア協調シミュレーションがサポートされている場合、[ハードウェア協調シミュレーション用のモデルのコンパイル](#)で説明されている手順に従い、System Generator トークン ダイアログ ボックスの [Hardware Co-Simulation] で [Compilation] オプションをイネーブルにします。[Hardware Co-Simulation] オプションが灰色表示されていて使用できない場合、そのボードでは JTAG ハードウェア協調シミュレーションは実行できません。

このサポートは、次のタイプのボードに適用されます。

- 。 Vivado Design Suite の一部としてインストールされているザイリンクス ボード
- 。 パートナー ウェブサイトからダウンロードして Vivado Design Suite の一部としてインストール可能なパートナー ボード
- 。 カスタム ボード。『Vivado Design Suite ユーザー ガイド: システム レベル デザイン入力』 (UG895) の付録 A 「ボード インターフェイス ファイル」に説明されているように、Vivado Design Suite で作成できます。
- ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション - 現在のところ、次の 2 つのボードでのみサポートされています。
 - 。 Kintex®-7 KC705 評価プラットフォーム
 - 。 Virtex®-7 VC707 評価プラットフォーム

ハードウェア協調シミュレーション用のモデルのコンパイル

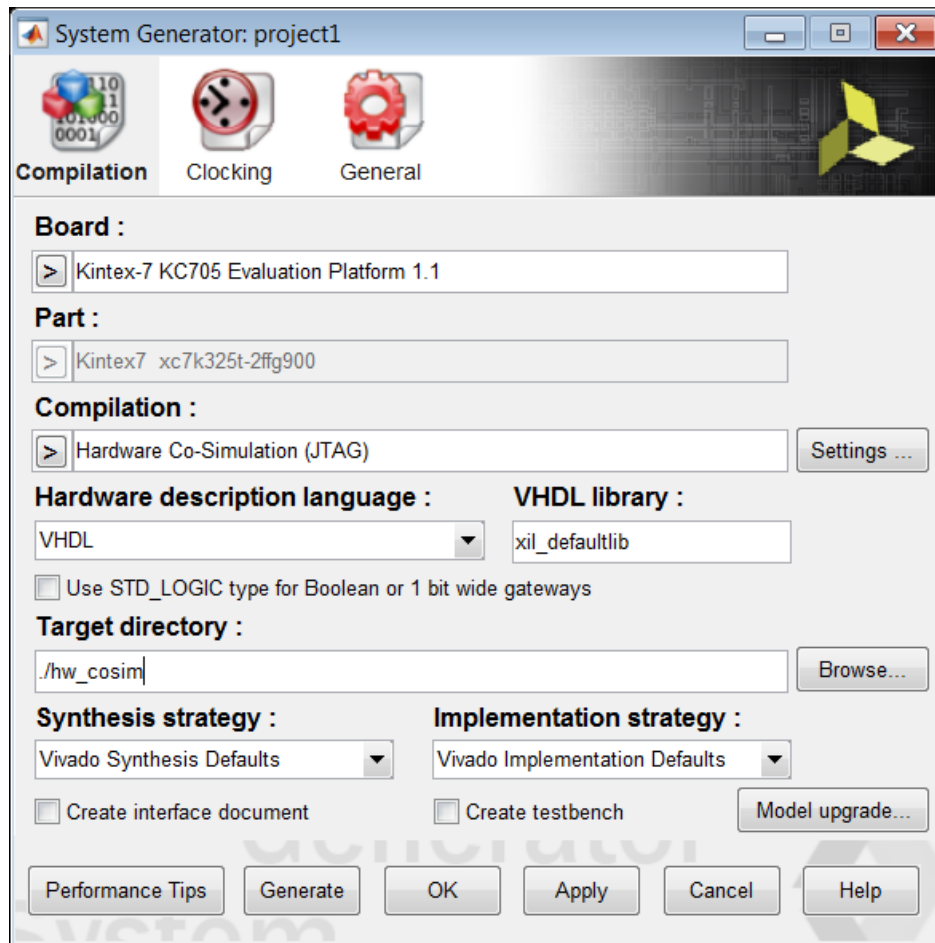
ハードウェア協調シミュレーションは、ハードウェアで実行する System Generator モデルまたはサブシステムから始めます。基本的なハードウェア ボードの要件を満たしているモデルであれば、どのモデルでも協調シミュレーションが可能です。モデルには System Generator トークンを含める必要があります。このブロックは、モデルのハードウェアへのコンパイル方法を定義します。

System Generator トークンの使用方法は、[System Generator トークンを使用したコンパイルおよびシミュレーション](#)を参照してください。

ハードウェア協調シミュレーション用に System Generator モデルをコンパイルするには、次を実行します。

1. System Generator トークンをダブルクリックして、System Generator トークンのパラメーター ダイアログ ボックスを開きます。

図 114: System Generator トークンのパラメーター ダイアログ ボックス



2. [Compilation] タブの [Board] でボードおよびそのバージョンを選択します。

[Board] リストに表示されるボードは次のとおりです。

- Vivado Design Suite の一部としてインストールされているボードすべて。
- Vivado Design Suite で作成されたカスタム ボード。
- 購入し、Vivado Design Suite でイネーブルになっているパートナー ボード。

パートナー ボードまたはカスタム ボードが [Board] リストに表示されるようにするには、ボードを記述するボード ファイルにアクセスするよう System Generator を設定する必要があります。System Generator でのボード サポートについては、[System Generator でのボード サポートの指定](#)を参照してください。

ハードウェア協調シミュレーション用にコンパイルするには、[Board] でボードを選択する必要があります。[Board] を [None] に設定したり、[Part] の代わりに [Board] を選択することはできません。

[Board] を選択すると、[Part] フィールドに [Board] で選択したサイリンクス デバイスの名前が表示されます。[Part] の設定は変更できません。

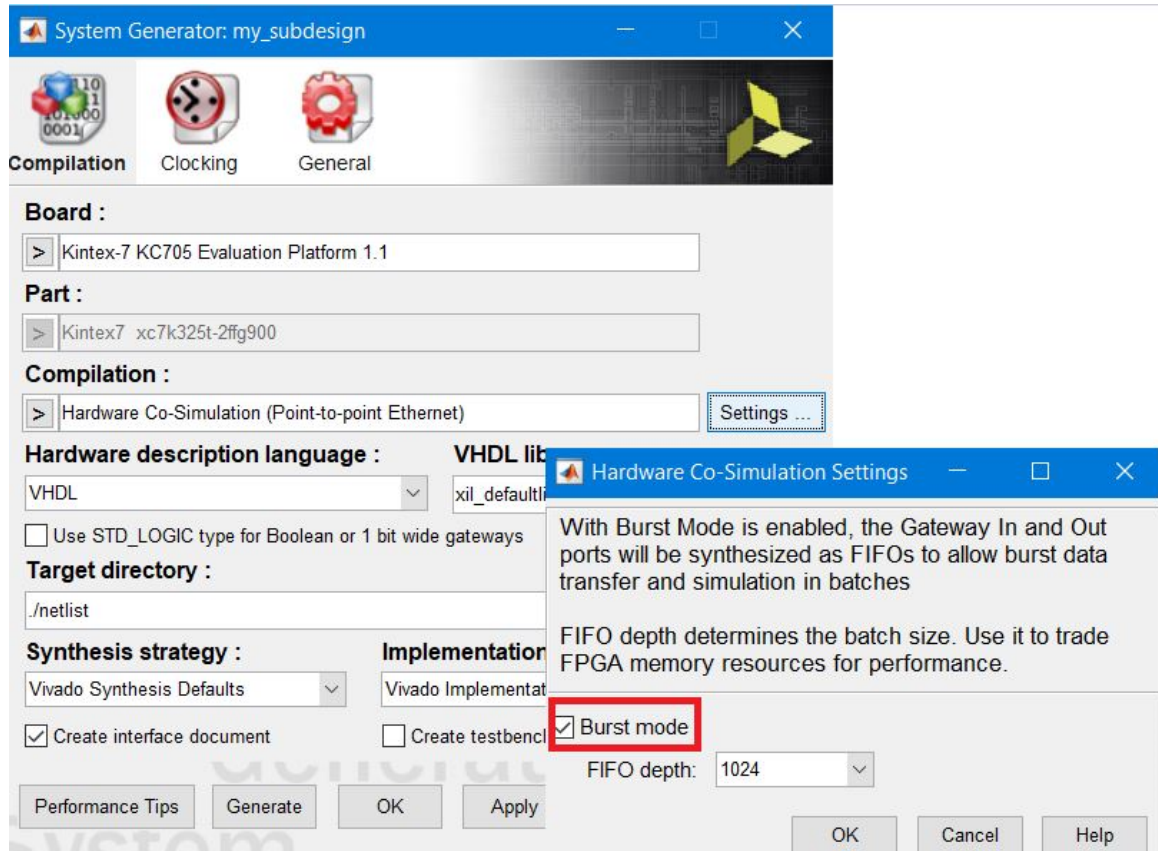
3. [Compilation] で [Hardware Co-Simulation] を選択し、必要に応じて [JTAG] または [Point-to-Point Ethernet] インターフェイスのいずれかを選択します。

[Hardware Co-Simulation] オプションが淡色表示されていて選択できない場合は、そのボードでは JTAG ハードウェア協調シミュレーションは実行できません。現時点では、ザイリンクスの KC705 および VC707 評価ボードでのみポイント ツー ポイント イーサネットを使用したハードウェア協調シミュレーションを実行できます。

4. ハードウェア協調シミュレーションをより高速に実行するためバースト モードを使用する場合は、[Settings] の横にある [Compilation] ボタンをクリックし、[FIFO depth] をオンにして [Burst mode] を指定します。[OK] をクリックして [Hardware Co-Simulation Settings] ダイアログ ボックスを閉じます。

バースト モードの詳細は、[ハードウェア協調シミュレーションのバースト データ転送](#)を参照してください。

図 115: バースト モードの設定



重要: バースト モードのハードウェア協調シミュレーションを実行するには、System Generator トークンのダイアログ ボックスで [Create Testbench] をオンにしてテストベンチを作成する必要があります。

5. コンパイルの一部としてテストベンチを作成する場合は、[Create Testbench] をオンにします。

[Create Testbench] をオンにする場合、コンパイル プロセスでサンプル テストベンチが自動的に作成されます。ハードウェア協調シミュレーション用の独自のテストベンチをユーザーが作成することもできます ([ハードウェア協調シミュレーションへの M コード アクセス](#)を参照)。

6. [Generate] をクリックします。

ハードウェア協調シミュレーション用にデザインの FPGA コンフィギュレーション ビットストリームが生成されます。System Generator では、コンパイル プロセス中にモデルの HDL およびネットリスト ファイルが生成されるだけでなく、FPGA コンフィギュレーション ファイルを生成するのに必要なダウンストリーム ツールも実行されます。

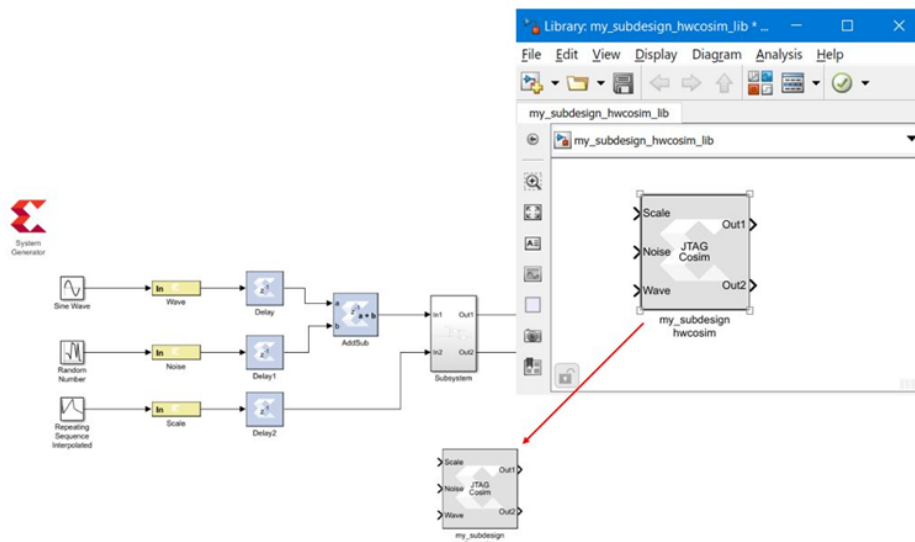
コンフィギュレーション ビットストリームには、モデルに関連するハードウェアと、System Generator がボードと PC 間の物理的インターフェイスを使用してデザインと通信できるようにする追加インターフェイス ロジックが含まれます。このロジックにはメモリ マップ インターフェイスが含まれており、System Generator はこれを介してデザインの入力ポートおよび出力ポートに対して値の読み出しおよび書き込みを実行できます。また、ターゲットの FPGA ボードを正しく機能させるために必要なボード特定の回路も含まれます。

コンパイルが終了すると、次のような結果になります。

- 手順 4 で [Burst mode] をオンにしていない場合は、[JTAG Cosim] または [Point-to-Point Ethernet] ハードウェア協調シミュレーション ブロックが別のウィンドウに表示されます。ハードウェア協調シミュレーション ブロックを Simulink モデルにドラッグ (またはコピーして貼り付け) します。ハードウェア協調シミュレーション ブロックを使用すると、Simulink ウィンドウ内でハードウェア協調シミュレーションを実行できます。

ハードウェア協調シミュレーション ブロックの詳細は、[ハードウェア協調シミュレーション ブロック](#) を参照してください。

図 116: ハードウェア協調シミュレーション ライブラリ ブロック



コンパイルで [Create Testbench] をオンした場合は、コンパイルにより M コードハードウェア協調シミュレーションのサンプル テストベンチも生成されます ([ハードウェア協調シミュレーションへの M コード アクセス](#) を参照)。このテストベンチを使用してハードウェア協調シミュレーションを実行するか、またはこれをカスタマイズしてユーザーのテストベンチを作成することもできます。

- 手順 4 で [Burst mode] をオンにした場合は、ハードウェア協調シミュレーション ブロックは表示されません。バースト モードの協調シミュレーションを実行する場合は、コンパイル中にターゲット ディレクトリに配置された MATLAB® の M コード テストベンチを使用します。

- 最上位デザインをコンパイルすると、テストベンチの名前が次のように指定されます。

```
<design_name>_hwcosim_test.m
```

- デザインのサブシステムをコンパイルすると、テストベンチの名前が次のように指定されます。

```
<design_name>_<sub_system>_hwcosim_test.m
```

コンパイルにより、Simulink モデルでハードウェア協調シミュレーションを実行する準備が整います。

ハードウェア協調シミュレーションを実行するには、次に進みます。

- 。標準 (非バースト モード) を実行する場合は、[標準ハードウェア協調シミュレーションの実行](#)を参照してください。
- 。バースト モードを実行する場合は、[バースト モードのハードウェア協調シミュレーションの実行](#)を参照してください。

標準ハードウェア協調シミュレーションの実行

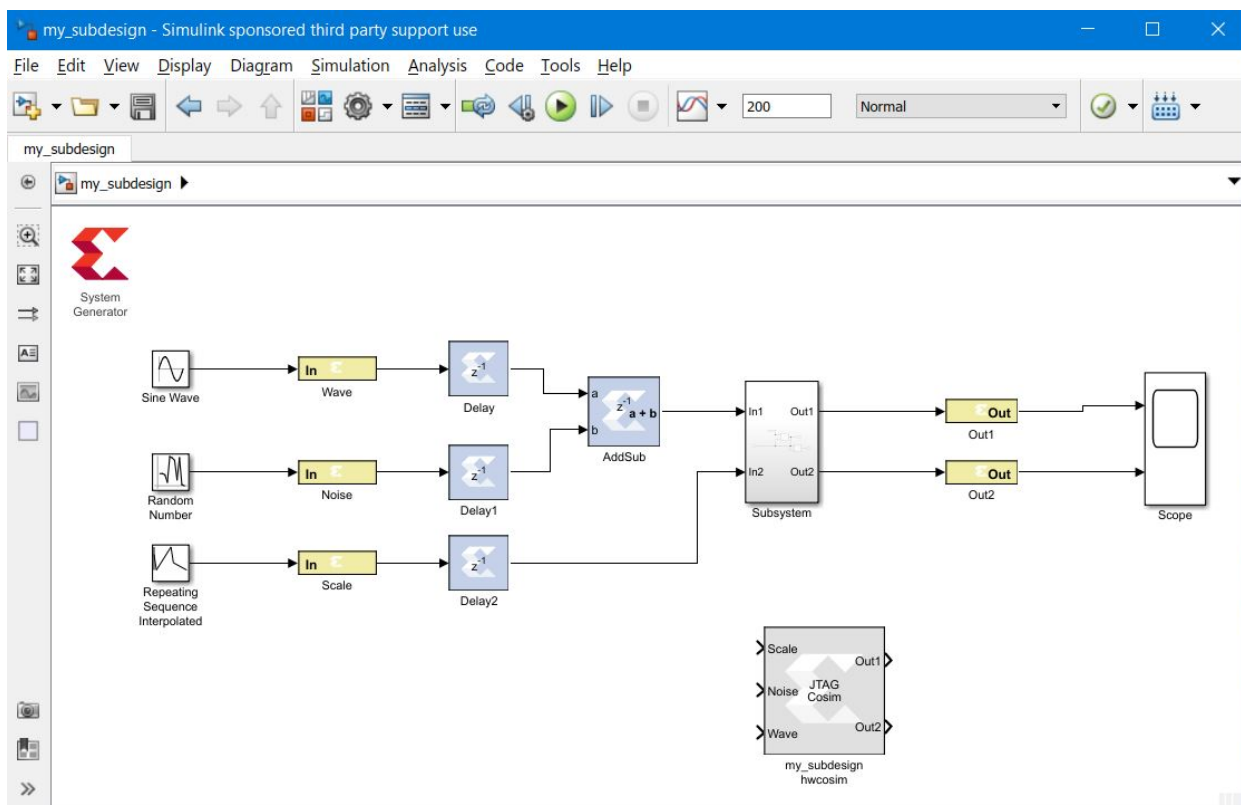
標準ハードウェア協調シミュレーション (非バースト モード) を実行している場合、Simulink モデルには JTAG またはポイント ツー ポイント イーサネットのハードウェア協調シミュレーション ブロックが含まれます。System Generator によりデザインが FPGA ビットストリームにコンパイルされるときに、このブロックは自動的に作成されます ([ハードウェア協調シミュレーション用のモデルのコンパイル](#)を参照)。ブロックは Simulink ライブラリに次のファイル名で保存されます。

```
<design_name>_hwcosim_lib.slx
```

ハードウェア協調シミュレーション ブロックは、コンパイル プロセスの最後に Simulink モデルに移動されます。この後の手順で、ハードウェア協調シミュレーションを実行するために Simulink モデルでこのブロックを接続する必要があります。

注記: ボードに Zynq® SoC デバイスが含まれている場合、ハードウェア協調シミュレーションを実行できるようにするには、Vitis™ 統合ソフトウェア プラットフォームを Vivado® Design Suite と共にインストールする必要があります。

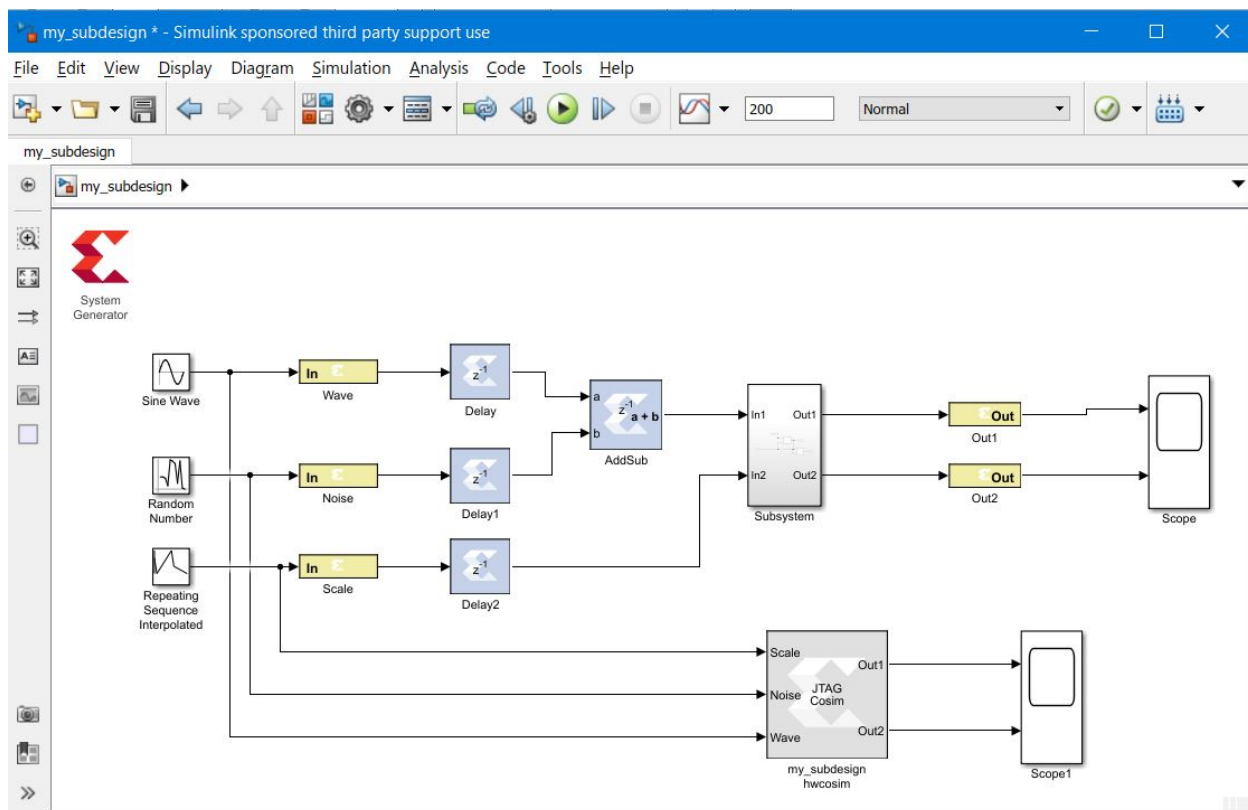
図 117: ハードウェア協調シミュレーション ブロック



標準のハードウェア協調シミュレーションを実行するには、次の手順に従います。

1. ハードウェア協調シミュレーション ブロックを、入力を供給し、出力を受信する Simulink ブロックに接続します。

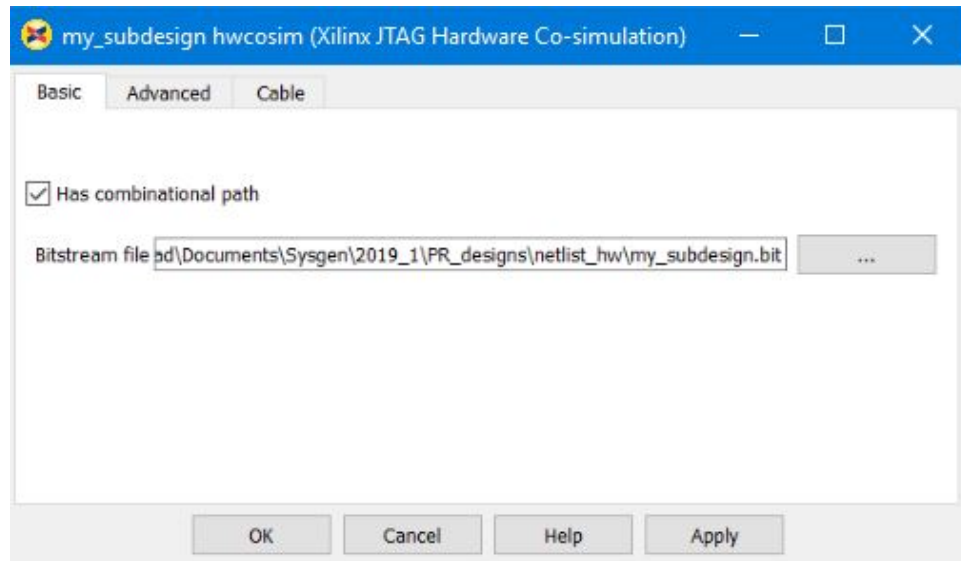
図 118: Simulink ブロックへの接続



2. ハードウェア協調シミュレーション ブロックをダブルクリックし、ブロックのプロパティ ダイアログ ボックスを表示します。

注記: JTAG ハードウェア協調シミュレーションとポイント ツー ポイント イーサネット ハードウェア協調シミュレーションには、それぞれ異なるブロック プロパティがあります。

図 119: ハードウェア協調シミュレーション ライブラリ ブロックのプロパティ



3. プロパティ ダイアログ ボックスでブロック パラメーターを設定します。

プロパティの詳細は、[JTAG ハードウェア協調シミュレーション ブロックのブロック パラメーター](#)または[イーサネット ハードウェア協調シミュレーション ブロックのブロック パラメーター](#)を参照してください。

4. ハードウェア協調シミュレーション実行用にボードを設定します。

- JTAG ハードウェア協調シミュレーションの場合は、ケーブルをボードの JTAG ポートに接続します。

KC705 ボードを使用した JTAG ハードウェア協調シミュレーションの設定手順は、[JTAG ハードウェア協調シミュレーション用の KC705 ボードの設定](#)を参照してください。

- ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションの場合は、ケーブルをボードの JTAG ポートに接続し、別のケーブルをボードのイーサネット ポートに接続します。ハードウェア協調シミュレーションを実行するときは、ボード上のザイリンクス デバイスは JTAG ポートを使用してプログラムし、プログラムしたデバイスをイーサネット ポートを使用してシミュレーションします。

KC705 または VC707 ボードを使用したポイント ツー ポイント イーサネット ハードウェア協調シミュレーションのボード設定手順は、[ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション用の KC705 ボードの設定](#)または[ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション用の VC707 ボードの設定](#)を参照してください。

5. ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションを実行している場合は、次の手順に従います。

- a. PC のローカル エリア ネットワークを設定し、ハードウェア協調シミュレーションを実行できるようにします。

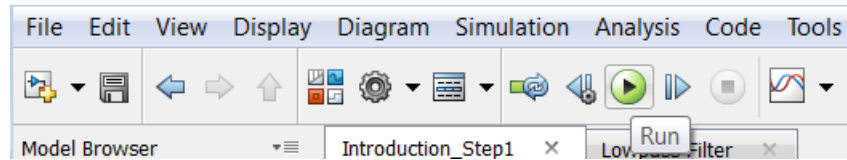
この手順は、[PC でのローカル エリア ネットワークの設定](#)を参照してください。

- b. PC がファイアウォール内で動作している場合は、ハードウェア協調シミュレーション実行中はファイアウォールをディスエーブルにしてください。

- c. オプションで、ハードウェア協調シミュレーション実行中は、PC でアンチウィルス プログラムをディスエーブルにします。

6. Simulink モデルで、[Simulation]→[Run] をクリックするか、またはツールバーの [Run] ボタンをクリックして、モデルおよびハードウェアをシミュレーションします。

図 120: [Run] ボタン



シミュレーションを実行すると、Simulink モデルの System Generator デザイン (またはサブシステム) と、ターゲット ボード上のザイリンクス デバイスの両方がシミュレーションされます。2 つのシミュレーション結果を比較して、ハードウェアにインプリメントされたデザインが予測どおりに動作するかどうかを判断できます。

バースト モードのハードウェア協調シミュレーションの実行

バースト モードのハードウェア協調シミュレーションを実行するには、コンパイル中に自動生成された MATLAB の M コード テストベンチを実行します ([ハードウェア協調シミュレーション用のモデルのコンパイル](#)を参照)。

このテストベンチは、ハードウェア協調シミュレーション用にデザインをコンパイルしたときに [Target directory] で指定したディレクトリにあります。

テストベンチの名前は次のようになります。

- 最上位デザインをコンパイルすると、テストベンチの名前が次のように指定されます。

```
<design_name>_hwcosim_test.m
```

- デザインのサブシステムをコンパイルすると、テストベンチの名前が次のように指定されます。

```
<design_name>_<sub_system>_hwcosim_test.m
```

注記: ボードに Zynq® SoC デバイスが含まれている場合、ハードウェア協調シミュレーションを実行できるようにするには、Vitis™ 統合ソフトウェア プラットフォームを Vivado® Design Suite と共にインストールする必要があります。

バースト モードのハードウェア協調シミュレーションを実行するには、次の手順に従います。

1. ハードウェア協調シミュレーション実行用にボードを設定します。

- JTAG ハードウェア協調シミュレーションの場合は、ケーブルをボードの JTAG ポートに接続します。

KC705 ボードを使用した JTAG ハードウェア協調シミュレーションの設定手順は、[JTAG ハードウェア協調シミュレーション用の KC705 ボードの設定](#)を参照してください。

- ポイント ツー ポイント イーサネットの場合は、ケーブルをボードの JTAG ポートに接続し、別のケーブルをボードのイーサネット ポートに接続します。ハードウェア協調シミュレーションを実行する際は、ボード上のザイリンクス デバイスは JTAG ポートを使用してプログラムし、プログラムしたデバイスをイーサネット ポートを使用してシミュレーションします。

2. ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションを実行している場合は、次の手順に従います。
 - a. PC のローカル エリア ネットワークを設定し、ハードウェア協調シミュレーションを実行できるようにします。

この手順は、[PC でのローカル エリア ネットワークの設定](#) を参照してください。

この手順の中で、PC のイーサネット アダプターでデータ伝送にジャンボ フレーム (1500 バイトより大きいフレーム) が使用されるように指定できます。ジャンボ フレームを使用すると、ポイント ツー ポイント イーサネットのハードウェア協調シミュレーションがスピードアップします。ジャンボ フレームの詳細は、[ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションにジャンボ フレームを使用](#)を参照してください。

- b. PC がファイアウォール内で動作している場合は、ハードウェア協調シミュレーション実行中はファイアウォールをディスエーブルにしてください。
 - c. オプションで、ハードウェア協調シミュレーション実行中は、PC でアンチウィルス プログラムをディスエーブルにします。
3. MATLAB のコンソールからテストベンチのスクリプトを実行します。テストベンチを実行するには、MATLAB のコンソールを開き、[Target Directory] で指定したディレクトリに移動して、スクリプトの名前を指定して実行します。

このスクリプトは、Simulink モデルを実行してほかの Simulink ソース ブロックまたは MATLAB 変数からザイリンクス Gateway In ブロックに駆動されるスティミュラス データを決定し、ザイリンクス ブロック デザイン (BD) により生成される予測出力を取り込み、このデータを別のデータ ファイルとして [Target directory] で指定されているターゲット ディレクトリにエクスポートします。

```
<design_name>_<sub_system>_<port_name>.dat
```

その後、テストベンチを予測出力と比較します。

テストにエラーが発生すると、コンソールにエラーが出力され、次のファイルにエラーが発生した箇所が比較されて出力されます。

```
<design_name>_<sub_system>_hwcosim_test.result
```

ハードウェア協調シミュレーションへの M コード アクセス

MATLAB M コード (M-Hwcosim) を使用すると、System Generator ハードウェア協調シミュレーション フローで作成されたハードウェアをプログラム制御できます。M-Hwcosim インターフェイスでは、Simulink のフレームワークとは別に、ハードウェアに対応する MATLAB オブジェクトを純粋な M コードで作成できます。これにより、これらのオブジェクトを使用して、ハードウェアに対して読み出しおよび書き込みを実行できるようになります。この機能を使用して、ハードウェア協調シミュレーション用にスクリプト インターフェイスを含めることができ、ハードウェアをスクリプト記述されたテストベンチで使用したり、M コードでのハードウェア アクセラレーションとして運用できるようになります。

詳細は、『Vivado Design Suite リファレンス ガイド: System Generator を使用したモデル ベースの DSP デザイン』(UG958) の[ハードウェア協調シミュレーションへの M コード アクセス](#)を参照してください。

ハードウェア ボードの設定

ハードウェア協調シミュレーションを実行するには、まずハードウェア ボードを設定する必要があります。JTAG およびポイント ツー ポイント イーサネットのハードウェア協調シミュレーション用にハードウェアを設定するには、次の手順に従います。

- JTAG ハードウェア協調シミュレーション: ボードの JTAG ポートにケーブルを接続します。

ボードの JTAG ポートの位置に関しては、ボードの資料を参照してください。ザイリンクス ボードは、ザイリンクス ウェブサイトの[ボードおよびキット](#) ページからダウンロードできます。

KC705 ボードを使用した JTAG ハードウェア協調シミュレーションの設定に関しては、[JTAG ハードウェア協調シミュレーション用の KC705 ボードの設定](#)を参照してください。

- ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション: ケーブルをボードの JTAG ポートに接続し、別のケーブルをボードのイーサネット ポートに接続します。ハードウェア協調シミュレーションを実行するときは、ボードのザイリンクス デバイスは JTAG ポートを使用してプログラムされ、プログラムされたデバイスはその後イーサネット ポートを使用してシミュレーションされます。

ボードの JTAG およびイーサネットのポートの位置に関しては、ボードの資料を参照してください。ザイリンクス ボードは、ザイリンクス ウェブサイトの[ボードおよびキット](#) ページからダウンロードできます。

KC705 または VC707 ボードを使用したポイント ツー ポイント イーサネット ハードウェア協調シミュレーションのボード設定手順は、[ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション用の KC705 ボードの設定](#) または [ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション用の VC707 ボードの設定](#) を参照してください。

JTAG ハードウェア協調シミュレーション用の KC705 ボードの設定

次のセクションでは、KC705 ボードで JTAG ハードウェア協調シミュレーションを実行するのに必要なハードウェアの設定方法を説明します。

KC705 ボードの詳細は、『Kintex-7 KC705 評価ボード ユーザー ガイド』([UG810](#)) を参照してください。

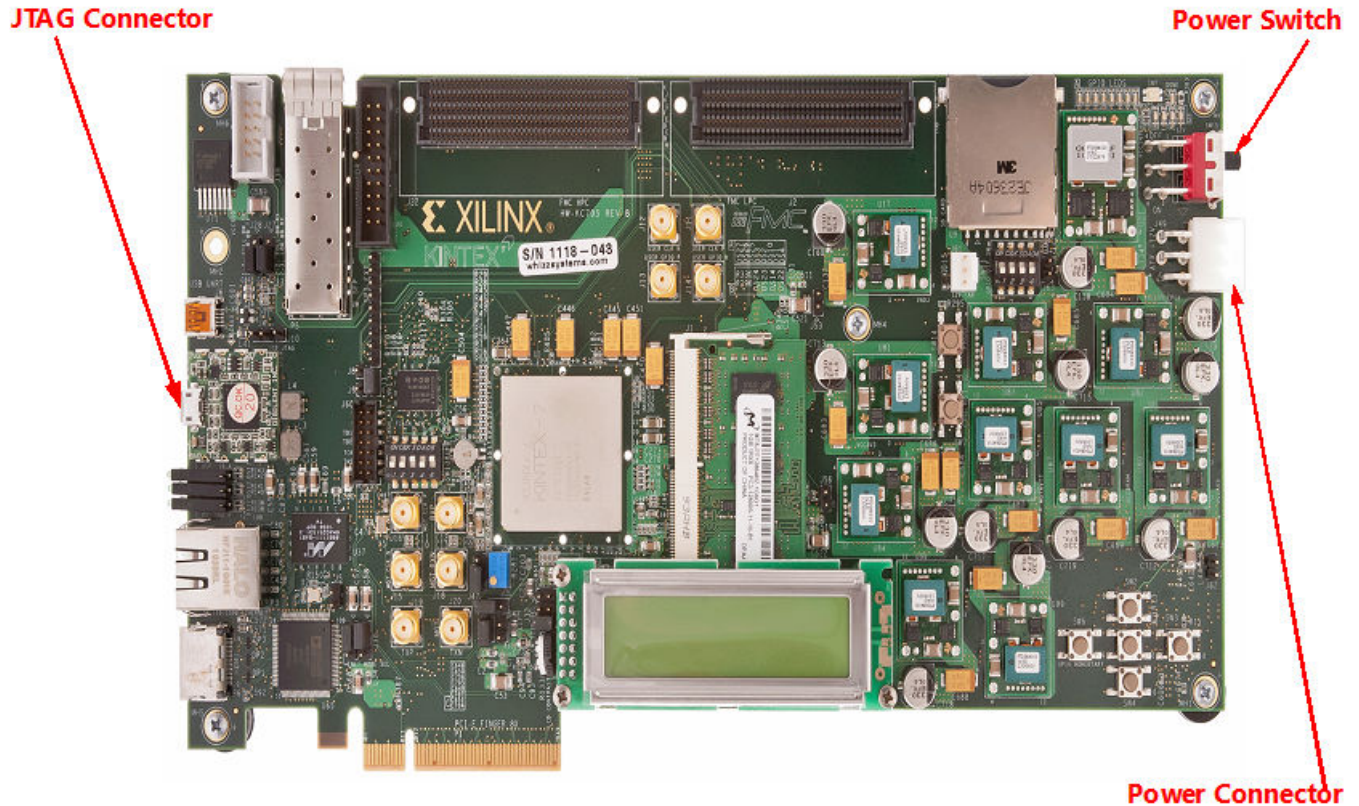
必要なハードウェア

1. ザイリンクス Kintex-7 KC705 ボードには、次のものが含まれています。
 - a. Kintex-7 KC705 ボード
 - b. KC705 キットに含まれる 12V 電源
 - c. Micro USB-JTAG ケーブル

KC705 ボードの設定

次の図に、この JTAG 設定に必要な KC705 コンポーネントを示します。

図 121: KC705 ボード



1. KC705 ボードを図のように配置します。
2. ボードの右上の電源スイッチがオフになっていることを確認します。
3. AC 電源コードを電源に接続します。電源アダプター ケーブルを KC705 ボードに差し込みます。電源ケーブルを AC 電源に差し込みます。
4. Micro USB-JTAG ケーブルの小さいほう端を JTAG ソケットに接続します。
5. Micro USB-JTAG ケーブルの大きいほうの端を PC の USB ソケットに接続します。
6. KC705 ボードの電源スイッチをオンにします。

ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション用の KC705 ボードの設定

次に、KC705 ボードのポイント ツー ポイント イーサネット ハードウェア協調シミュレーションを実行するのに必要なハードウェアをインストールする方法を説明します。

KC705 ボードの詳細は、『Kintex®-7 FPGA 用 KC705 評価ボード ユーザー ガイド』(UG810)を参照してください。

注記: ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションには、オート ネゴシエーションの使用を含む全二重イーサネットが必要です。ネットワーク インターフェイス カード (NIC) または USB-to-Ethernet アダプターを使用してポイント ツー ポイント イーサネット ハードウェア協調シミュレーションを実行する場合、この接続は次の条件下でのみ機能します。

- NIC または USB-to-Ethernet アダプターがボードに直接接続されている。

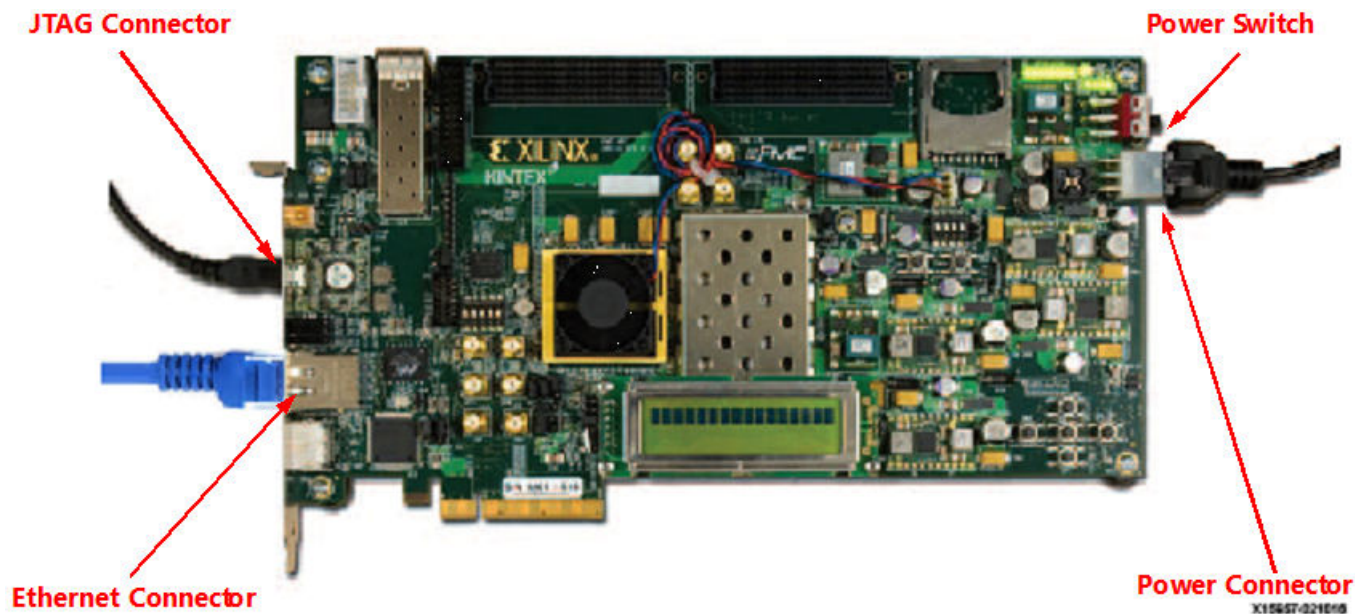
- NIC または USB-to-Ethernet アダプターで IEEE 802.3ab ギガビット イーサネット規格がサポートされている。
- NIC または USB-to-Ethernet アダプターでオート ネゴシエーションを使用した全二重イーサネット動作がサポートされている。オート ネゴシエーションなしで速度を直接設定すると、ポイント ツー ポイント イーサネット接続は機能しません。

必要なハードウェア

- ザイリンクス KC705 ボード
- ボードの電源
- ホスト PC 用のイーサネット ネットワーク インターフェイス カード (NIC)
- イーサネット RJ45 オス/メス ケーブル (ネットワークまたはクロスオーバー ケーブルなど)
- ビットストリーム ダウンロード用の Digilent USB ケーブルまたはプラットフォーム USB ケーブル

KC705 ボードの設定

図 122: KC705 ボードのイーサネット接続



ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション用に KC705 ボードを設定するには、次を実行します。

1. KC705 ボードを図のように配置します。
2. ボードの右上の電源スイッチがオフになっていることを確認します。
3. 電源ケーブルを右側に接続します。電源ケーブルを AC 電源に差し込みます。
4. Digilent USB ケーブルを左上に接続し、もう一端をホスト PC に接続します。
5. イーサネット ケーブルを KC705 ボードの左下に接続し、もう一端をホスト PC に接続します。
6. KC705 ボードの電源スイッチをオンにします。

ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション用の VC707 ボードの設定

次に、VC707 ボードでポイント ツー ポイント イーサネット ハードウェア協調シミュレーションを実行するのに必要なハードウェアをインストールする方法を説明します。

VC707 ボードの詳細は、『Virtex-7 FPGA 用 VC707 評価ボード ユーザー ガイド』(UG885) を参照してください。

注記: ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションには、オート ネゴシエーションの使用を含む全二重イーサネットが必要です。ネットワーク インターフェイス カード (NIC) または USB-to-Ethernet アダプターを使用してポイント ツー ポイント イーサネット ハードウェア協調シミュレーションを実行する場合は、接続は次の条件下でのみ機能します。

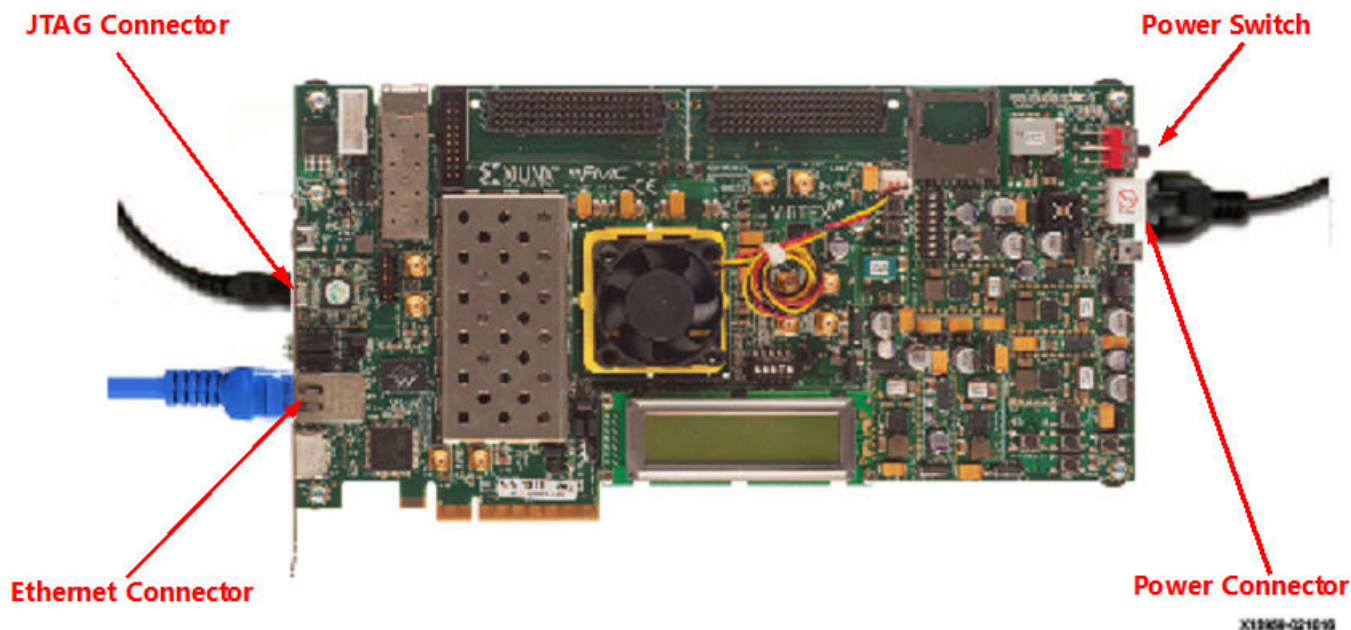
- NIC または USB-to-Ethernet アダプターがボードに直接接続されている。
- NIC または USB-to-Ethernet アダプターで IEEE 802.3ab ギガビット イーサネット規格がサポートされている。
- NIC または USB-to-Ethernet アダプターでオート ネゴシエーションを使用した全二重イーサネット動作がサポートされている。オート ネゴシエーションなしで速度を直接設定すると、ポイント ツー ポイント イーサネット接続は機能しません。

必要なハードウェア

- ザイリンクス VC707 ボード
- ボードの電源
- ホスト PC 用のイーサネット ネットワーク インターフェイス カード (NIC)
- イーサネット RJ45 オス/メス ケーブル (ネットワークまたはクロスオーバー ケーブルなど)
- ビットストリーム ダウンロード用の Digilent USB ケーブルまたはプラットフォーム USB ケーブル

VC707 ボードの設定

図 123: VC705 ボードのイーサネット接続



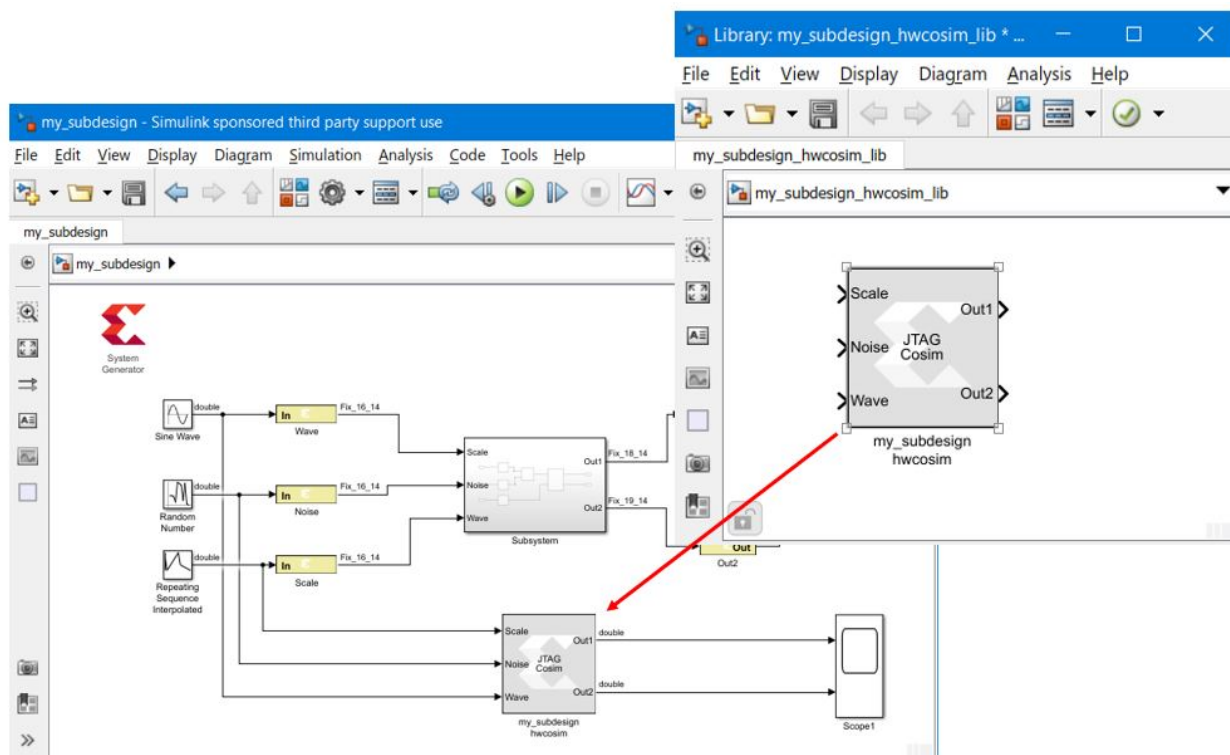
ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション用に VC707 ボードを設定するには、次を実行します。

1. VC707 ボードを図のように配置します。
2. ボードの右上の電源スイッチがオフになっていることを確認します。
3. 電源ケーブルを右側に接続します。電源ケーブルを AC 電源に差し込みます。
4. Digilent USB ケーブルを左上に接続し、もう一端をホスト PC に接続します。
5. イーサネット ケーブルを VC705 ボードの左下に接続し、もう一端をホスト PC に接続します。
6. VC707 ボードの電源スイッチをオンにします。

ハードウェア協調シミュレーション ブロック

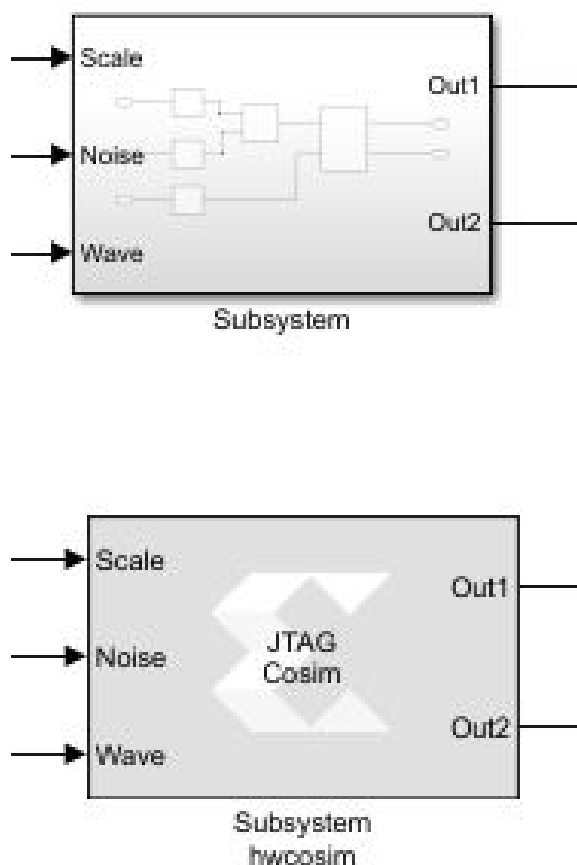
System Generator では、デザインの FPGA ビットストリームへのコンパイルが完了すると、新しいハードウェア協調シミュレーション ブロックが自動的に作成されます。ハードウェア協調シミュレーション ブロックを格納するための Simulink ライブラリも作成されます。この時点で、ライブラリからこのブロックをコピーして、ほかの Simulink または System Generator ブロックと同様に、System Generator デザインで使用できるようになります。

図 124: ハードウェア協調シミュレーション ブロック



ハードウェア協調シミュレーション ブロックは、元になっているモデルまたはサブシステムの外部インターフェイスがあるものと想定します。ハードウェア協調シミュレーション ブロックのポート名は、元のサブシステムのポート名と同じになります。ポート タイプおよびレートも元のデザインと同じになります。

図 125: ポート名



ハードウェア協調シミュレーション ブロックは、ほかのブロックと同じように Simulink デザインで使用されます。シミュレーション中、ハードウェア協調シミュレーション ブロックは FPGA ボードと通信して、デバイス コンフィギュレーション、データ転送、クロック供給などのタスクを自動化します。ハードウェア協調シミュレーション ブロックは、ほかの System Generator ブロックが使用する信号の型と同じものを使用および生成します。値がブロックの入力ポートの 1 つに書き込まれると、ブロックは対応するデータをハードウェアの該当箇所へ送信します。同様に、出力ポートにイベントがあると、ブロックはハードウェアからデータを取り出します。

ハードウェア協調シミュレーション ブロックは、ザイリンクスの固定小数点型、Simulink の固定小数点型、または Simulink の倍精度 (double) 型の信号で駆動可能です。出力ポートは、信号型が駆動するブロックに適したものであるとみなします。出力ポートが System Generator ブロックに接続されると、出力ポートではザイリンクスの固定小数点型の信号が出力されます。また、出力ポートで Simulink ブロックが直接駆動される場合は、Simulink データ型の信号が出力されます。

注記: Simulink データ型がブロック信号型として使用されると、入力データの量子化は丸めによって処理され、オーバーフローは飽和によって処理されます。

ハードウェア協調シミュレーション ブロックには、その他の System Generator ブロックと同様にパラメーター ダイアログ ボックスがあり、さまざまな設定を指定できます。ハードウェア協調シミュレーション ブロックのパラメーターは、ブロックがインプリメントされる FPGA ボードによって異なり、FPGA ボードごとにカスタマイズされたハードウェア協調シミュレーション ブロックが提供されています。

JTAG ハードウェア協調シミュレーション ブロックのブロック パラメーター

Simulink モデルでブロックのアイコンをダブルクリックすると、JTAG ハードウェア協調シミュレーション ブロックのブロック パラメーター ダイアログ ボックスが開きます。

ブロックに特化したパラメーターには次のものがあります。

[Basic] タブ

[Has combinational path]: 回路に組み合わせパスがある場合は、オンにします。クロック イベントなしに入力から出力へ変化が伝搬するのが組み合わせパスです。このパスには、ラッチ、フリップフロップ、またはレジスタはありません。オンにすると、System Generator でデザインへのクロック供給前、および入力の書き込み直後に、出力が読み出されます。これにより、ハードウェア協調シミュレーション ブロックから Simulink モデルまでの組み合わせパスでの値の変更が正しく伝搬されるようになります。

[Bitstream file]: FPGA コンフィギュレーション ビットストリームを指定します。デフォルトで、このフィールドには、System Generator トークンから前回実行された [Generate] 中に System Generator により生成されたビットストリームへのパスが含まれます。

[Advanced] タブ

[Skip device configuration]: オンにすると、コンフィギュレーション ビットストリームが FPGA または SoC には読み込まれません。別のプログラム (Vivado ハードウェア マネージャーや Vivado ロジック解析など) がデバイスを設定している場合に、このオプションを使用できます。

[Display Part Information]: ハードウェア協調シミュレーション ブロックの中央のデバイスのパーツ情報の表示を切り替えます (たとえば、Kintex デバイスの場合は「xc7k325tffg900-2」と表示)。

[Cable] タブ

ケーブル設定

- [Type]: 現在のところ、このパラメーターは [Auto Detect] にのみ設定できます。System Generator は自動的にケーブル タイプを検出します。

イーサネット ハードウェア協調シミュレーション ブロックのブロック パラメーター

Simulink モデルでブロックのアイコンをダブルクリックすると、イーサネット ハードウェア協調シミュレーション ブロックのブロック パラメーター ダイアログ ボックスが開きます。

ブロックに特化したパラメーターには次のものがあります。

[Basic] タブ

[Clocking]

- クロック ソース: System Generator ハードウェア協調シミュレーション ブロックを関連 FPGA または SoC ハードウェアと同期化させるのに使用するクロッキング モード (シングル ステップまたはフリーランニング) を指定します。この 2 つのクロック ソースの詳細は、[クロッキング モード](#)を参照してください。

[Has combinational path]: 回路に組み合わせパスがある場合は、オンにします。クロック イベントなしに入力から出力へ変化が伝搬するのが組み合わせパスです。このパスには、順序ロジック (ラッチ、フリップフロップ、またはレジスタ) はありません。オンにすると、System Generator でデザインへのクロック供給前、および入力の書き込み直後に、出力が読み出されます。これにより、ハードウェア協調シミュレーション ブロックから Simulink モデルまでの組み合わせパスでの値の変更が正しく伝搬されるようになります。

[Bitstream file]: FPGA コンフィギュレーション ビットストリームを指定します。デフォルトで、このフィールドには、System Generator トークンから前回実行された [Generate] 中に System Generator により生成されたビットストリームへのパスが含まれます。

[Advanced] タブ

[Skip device configuration]: オンにすると、コンフィギュレーション ビットストリームが FPGA または SoC には読み込まれません。別のプログラム (Vivado ハードウェア マネージャーや Vivado ロジック解析など) がデバイスを設定している場合に、このオプションを使用できます。

[Display Part Information]: ハードウェア協調シミュレーション ブロックの中央のデバイスのパーツ情報の表示を切り替えます (たとえば、Kintex デバイスの場合は「xc7k325tffg900-2」と表示)。

[Ethernet] タブ

[Host Interface]

[Ethernet Interface]: このドロップダウン リストには、ホスト コンピューターで検出されたイーサネット インターフェイスがすべて含まれます。ターゲット ボードに接続されているインターフェイスを選択してください。選択されたインターフェイスは、ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションを実行できるよう、正しく設定しておく必要があります。ホスト インターフェイス設定の詳細は、[PC でのローカル エリア ネットワークの設定](#)を参照してください。

[Refresh] ボタン:[Refresh]をクリックすると、表示が更新され、使用可能なイーサネット インターフェイスが読み込まれます。ホットプラグ可能なイーサネット インターフェイスを表示させたり、またはブロック パラメーター ダイアログ ボックスを開いたときにはオフになっていたインターフェイスを後でオンにした場合などに、このボタンを使用します。

[FPGA Interface]

[MAC Address]: ターゲット ボードに割り当てられているイーサネット MAC アドレスです。空白のままになっている場合、デフォルト値は da:02:03:04:05:06 です。この値が、ホストの MAC アドレスと同じになることはありません。

[Configuration] タブ

[Cable]

- データ型: 現在のところ、このパラメーターは [Auto Detect] にのみ設定できます。System Generator は自動的にケーブル タイプを検出します。

[Configuration timeout (ms)]: コンフィギュレーション後最初のイーサネット ハンドシェイクのタイムアウト値を指定します。

ハードウェア協調シミュレーションのクロック

標準ハードウェア協調シミュレーションを実行している場合、協調シミュレーション ブロックを Simulink モデルに含めるよう設定するとき、クロッキング モードを選択する必要があります。

クロッキング モード

System Generator ハードウェア協調シミュレーション ブロックを関連付けられている FPGA ハードウェアと同期させる方法は複数あります。シングル ステップ クロック モードでは、FPGA に Simulink からクロックが供給されます。フリーランニング クロック モードでは、内部クロックが使用され、Simulink がハードウェア協調シミュレーション ブロックをウェークアップしたときに非同期にサンプリングされます。

シングル ステップ クロック

シングル ステップ クロック モードでは、ハードウェアをソフトウェア シミュレーションとロックステップで動作させます。これは、各シミュレーション サイクルでハードウェアに 1 つのクロック パルスを供給する方法です (FPGA が入力/出力レートよりも速くクロック供給を受ける場合は複数クロック パルスを供給)。このモードの場合、ハードウェア協調シミュレーション ブロックは元のモデルに対してビットごとまたはサイクルごとに比較されます。

ハードウェア協調シミュレーション ブロックは、Simulink がウェークアップしたときにのみ FPGA ハードウェアのクロック信号を生成するので、Simulink モデルのシミュレーションの残りの部分からのオーバーヘッドや、Simulink と FPGA ボード間の通信オーバーヘッド (バス レイテンシなど) により、ハードウェアで達成されるパフォーマンスが著しく制限されます。通信のオーバーヘッド (ロジック数が多い、ハードウェアが大幅にオーバークロックされるなど) に対して FPGA 内の計算能力が高ければ、ハードウェアでシミュレーション時間を大幅に短縮できます。

フリーランニング クロック

フリーランニング クロック モードでは、ハードウェアはソフトウェア シミュレーションに対して非同期に動作します。Simulink により FPGA クロックが生成されるシングル ステップ クロック モードとは異なり、フリーランニング モードではハードウェア クロックが FPGA 内で継続的に動作します。このモードでは、Simulink がハードウェア協調シミュレーションブロックをウェークアップしたときにのみ Simulink がハードウェアの内部ステートをサンプリングするので、シミュレーションは元のモデルとビット精度およびサイクル精度で動作しません。FPGA ポート I/O は Simulink のイベントとは同期しなくなります。Simulink ポートでイベントが発生すると、その時点で値がハードウェアの対応するポートから読み出されるか、そのポートに書き込まれます。ただし、ポートのイベント間にハードウェアで経過したクロック サイクル数はわからないので、ハードウェアの現在のステートを元の System Generator モデルと照合することはできません。ストリーミング アプリケーションでは、FPGA はフルスピードで動作し、Simulink とは定期的にしか同期しないため、実際にはこれは非常に好ましい状況です。

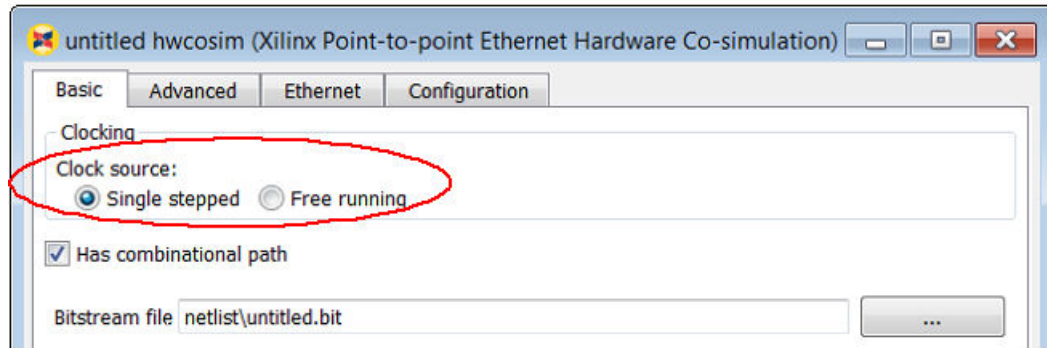
フリーランニング クロック モードでは、System Generator モデルに同期メカニズムを明示的に構築する必要があります。ハードウェア協調シミュレーション ブロックの出力ポートとして使用されるステータス レジスタがこの例で、条件が満たされたときにハードウェアで設定されます。System Generator モデルの残りの部分は、ステータス レジスタをポーリングして、ハードウェアのステートを判断できます。

クロック モードの選択

すべてのハードウェア ボードでフリーランニング クロックがサポートされるわけではありませんが、サポートされる場合は、ハードウェア協調シミュレーション ブロックのパラメーター ダイアログ ボックスでクロッキング モードを選択できます。協調シミュレーションのクロッキング モードを変更するには、シミュレーションの開始前に、パラメーター ダイアログ ボックスの [Clock Source] で [Single stepped] または [Free running] をオンにします。

注記: ハードウェア協調シミュレーション ブロックで使用可能なクロッキング オプションは、使用する FPGA ボードによって異なり、フリーランニング クロック ソースがサポートされないボードの場合は、ダイアログ ボックスにオプションは表示されません。

図 126: [Single stepped] をオン



M-Hwcosim を使用してフリーランニング クロックのオン/オフをプログラミングで切り替える方法については、『Vivado Design Suite リファレンス ガイド: System Generator を使用したモデル ベースの DSP デザイン』(UG958) (UG958) の「M-Hwcosim MATLAB クラス」を参照してください。

ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション

次のものは、ポイント ツー ポイント イーサネット インターフェイスを介して実行しているハードウェア協調シミュレーションに影響します。

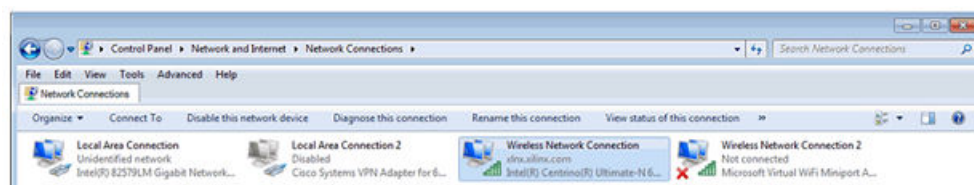
- [PC でのローカル エリア ネットワークの設定](#)
- [Linux でのポイント ツー ポイント イーサネット ハードウェア協調シミュレーション](#)
- [ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションにジャンボ フレームを使用](#)

PC でのローカル エリア ネットワークの設定

ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションには、PC に 10/100 高速イーサネットまたはギガビット イーサネット アダプターが必要です。設定するには、次の手順に従います。

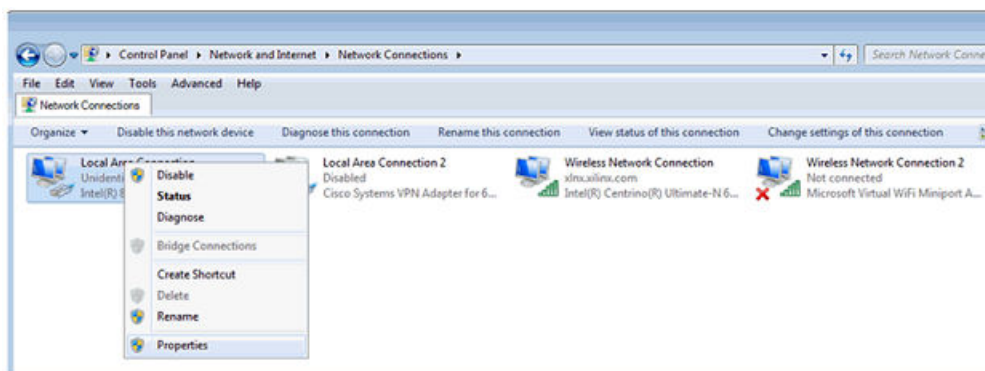
1. [Start] メニューから [Control Panel] (コントロール パネル) を開き、[Network and Internet] (ネットワークとインターネット) → [View network status and tasks] (ネットワークの状態とタスクの表示) をクリックします。左側のペインで [Change Adapter settings] (アダプターの設定の変更) をクリックします。

図 127: アダプターの設定の変更



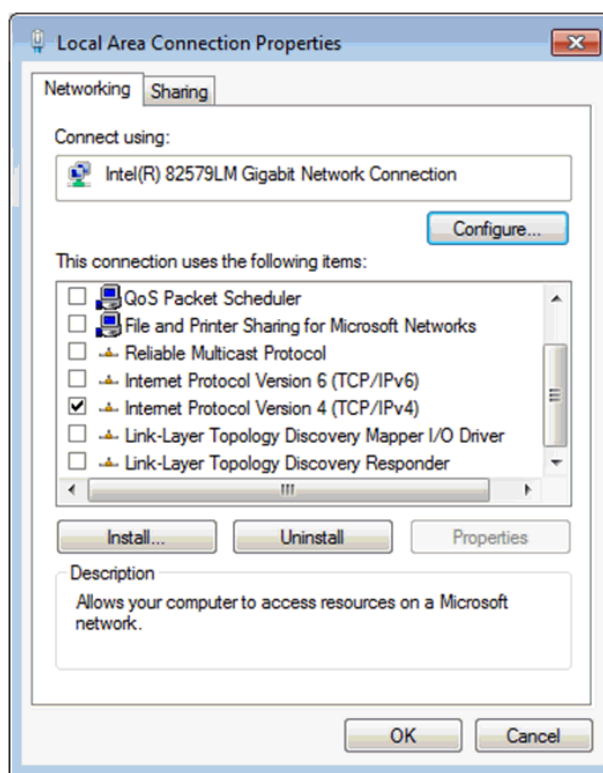
2. [Local Area Connection] (ローカル エリア接続) を右クリックして [Properties] (プロパティ) をクリックします。

図 128: ローカル エリア接続のプロパティ



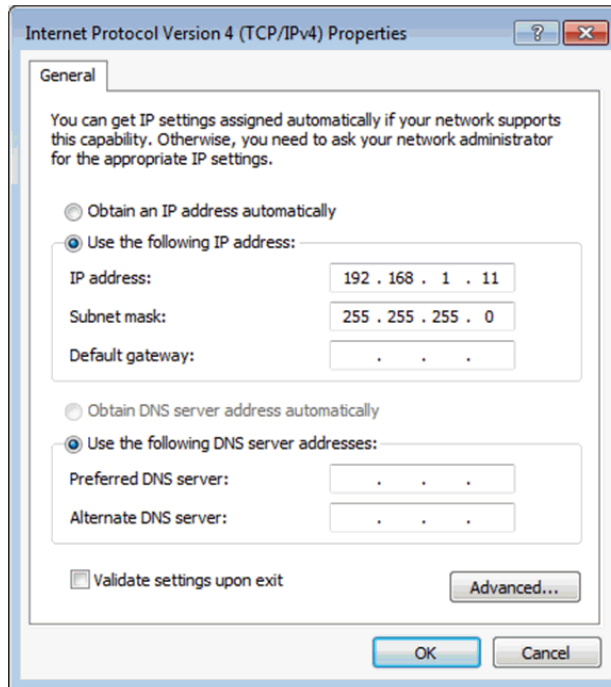
3. [プロパティ] ダイアログ ボックスで [Internet Protocol Version 4 (TCP/IPv4)] (インターネット プロトコル バージョン 4 (TCP/IPv4)) をオンにします。それ以外はすべてオフにします。

図 129: インターネット プロトコル



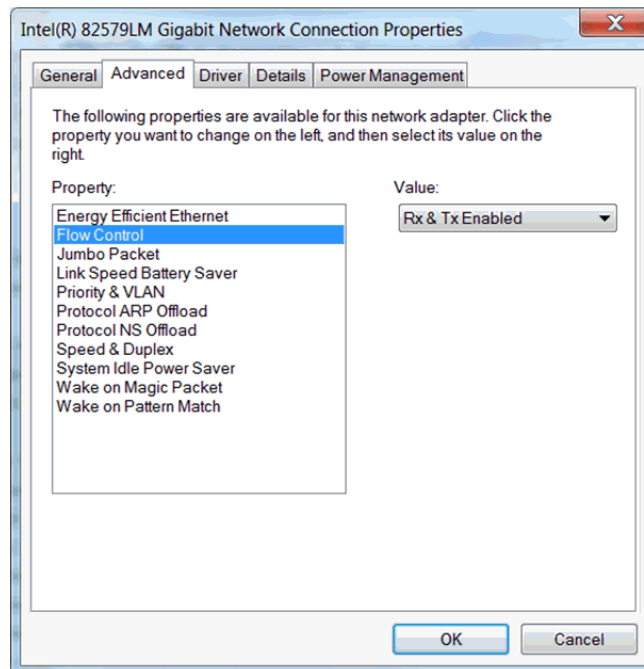
4. [Internet Protocol Version 4 (TCP/IPv4)] (インターネット プロトコル バージョン 4 (TCP/IPv4)) を選択し、[Properties] (プロパティ) をクリックします。[インターネット プロトコル バージョン 4 (TCP/IPv4) のプロパティ] ダイアログ ボックスで、[Use the following IP address] (次の IP アドレスを使う) をオンにし、[IP Address] (IP アドレス) を 192.168.1.11 に、[Subnet mask] (サブネット マスク) を 255.255.255.0 に設定します。[OK] をクリックします。

図 130: IP アドレスの設定



5. [ローカル エリア接続のプロパティ] ダイアログ ボックスで [Configure] (構成) をクリックします。[Advanced] (詳細設定) タブをクリックします。[Flow Control] をクリックします。[Value] (値) を [Rx & Tx Enabled] (Rx および Tx 有効) に設定します。

図 131: フロー コントロール



6. ジャンボ フレーム (1500 バイトを超えるパケット) を使用してポイント ツー ポイント イーサネット ハードウェア協調シミュレーションを高速化する場合は、[Jumbo Packet] (ジャンボ パケット) を選択し、[Value] (値) を必要なフレーム サイズに設定します。ジャンボ フレームについては、[ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションにジャンボ フレームを使用](#)を参照してください。
7. [OK] をクリックしてダイアログ ボックスを閉じます。

ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションにジャンボ フレームを使用

ジャンボ フレームは 1500 バイトを超える大きさのイーサネット フレームです。イーサネット アダプターでデータ転送にジャンボ フレームを使用できるように指定すると、ポイント ツー ポイント イーサネット ハードウェア協調シミュレーションに必要なデータ転送を高速化できます。

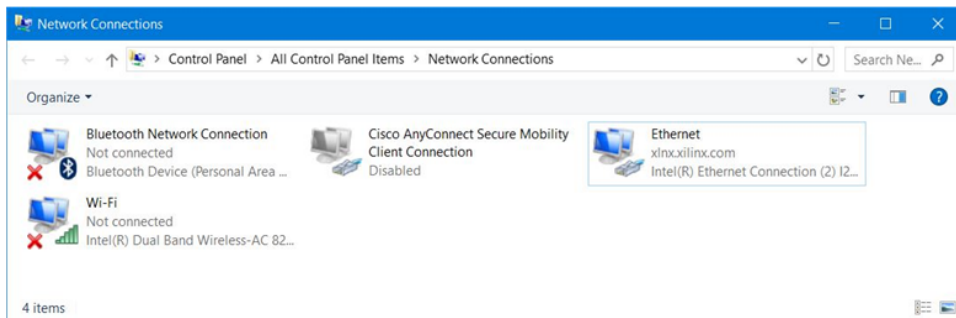
ジャンボ フレームは、ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション用にローカル エリア ネットワークを設定するときにイネーブルできます ([PC でのローカル エリア ネットワークの設定](#)を参照)。

ローカル エリア ネットワークを既に設定している場合は、次のように、後からジャンボ フレームを使用できるように設定できます。

1. Windows の [コントロール パネル] で、[Network and Internet] → [Network and Sharing Center] → [Change Adapter Settings] をクリックします。

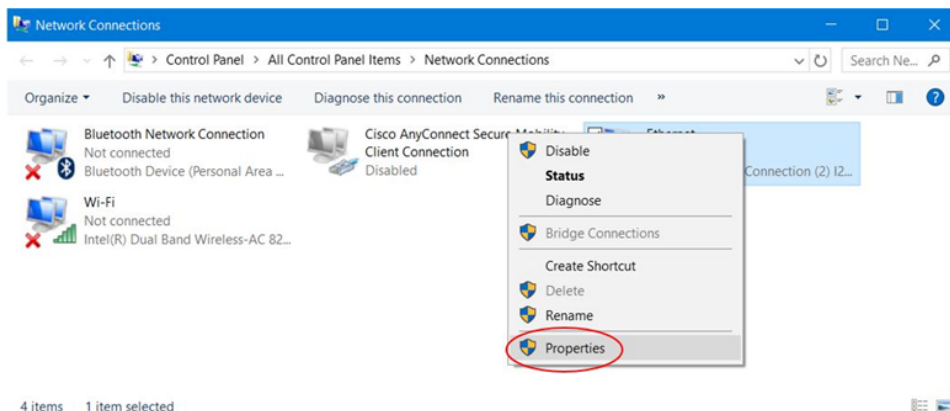
[ネットワーク接続] フォルダーが開きます。

図 132: [ネットワーク接続] フォルダー



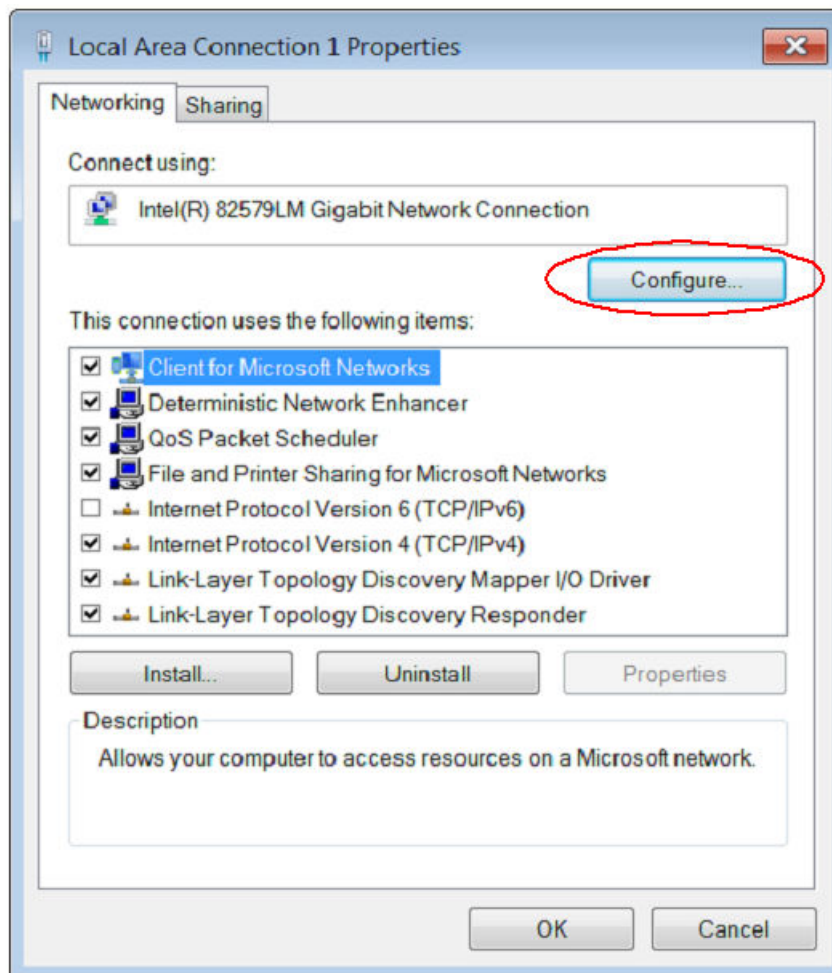
2. イーサネット ハードウェア協調シミュレーションに使用する [Local Area Connection] (ローカル エリア接続) を右クリックし、[Properties] (プロパティ) をクリックします。

図 133: [プロパティ] をクリック



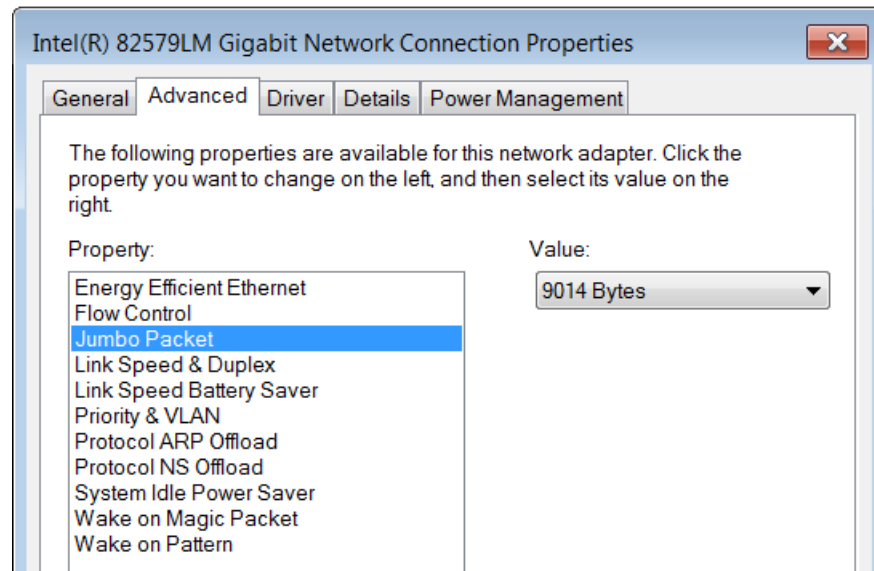
3. [プロパティ] ダイアログ ボックスで [Configure] (構成) をクリックします。

図 134: [構成] ボタン



4. アダプターの [プロパティ] ダイアログ ボックスの [詳細設定] タブで、[Jumbo Packet] (ジャンボ パケット) を選択し、[Value] (値) を目的のフレーム サイズに設定して、ジャンボ パケットを使用できるようにします。

図 135: ジャンボ パケットの設定



5. [OK] をクリックして、アダプターの [プロパティ] ダイアログ ボックスを閉じます。
6. [OK] をクリックして、ネットワーク接続のダイアログ ボックスを閉じます。

Linux でのポイント ツー ポイント イーサネット ハードウェア協調シミュレーション

Linux でポイント ツー ポイント イーサネット ハードウェア協調シミュレーションを実行するには、Linux コンピューターの sudo アクセスが必要です。sudo ユーザーとして System Generator を起動する必要があります。複数のネットワーク インターフェイス カードがコンピューターにない場合は、ネットワーク スイッチを使用できます。

ハードウェア協調シミュレーションのバースト データ転送

ハードウェア協調シミュレーション (HWCosim) は、モデルの中で最も計算負荷が高い箇所の一部またはすべてを、実際のターゲット FPGA プラットフォームで実行する手法です。ホストシステムは、協調シミュレーション インターフェイス (通常は JTAG またはポイント ツー ポイント イーサネット) を介してモデルにスティミュラスを供給し、その応答を処理します。この手法は、生成されたハードウェア デザインをターゲット プラットフォームで検証し、またハードウェア協調検証でモデル検証中のシミュレーションを高速化するのに便利です。

MATLAB/Simulink と System Generator for DSP を併用する場合、GUI ベースおよび MATLAB の M スクリプト ベースのハードウェア協調シミュレーションがサポートされています。GUI ベースのシミュレーションは、Simulink スケジューラで制御して実行され、モデルにフィードバック ループがある可能性があるため、1 クロック サイクルずつ進行します。

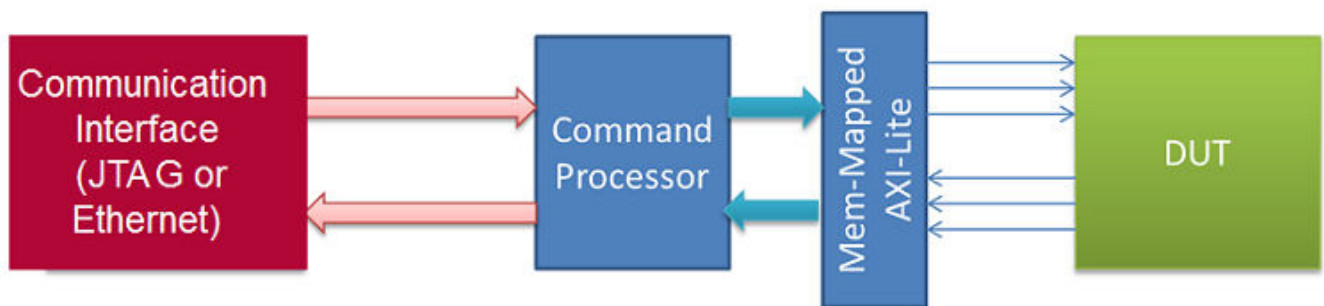
MATLAB の M スクリプト ベースのシミュレーションは、System Generator (M-HWCosim) で制御して実行され、System Generator トークンからのビットストリーム生成中に生成されたテストベンチで使用されるのが一般的です。これらのテストベンチには通常フィードバックはなく、既知の入力が使用されるので、大きなバッチでデバイスに転送できます。

以前のバージョンの System Generator for DSP (Vivado) では、基本的なハードウェア協調シミュレーションのみインプリメントされ、インターフェースのパフォーマンスを最大限に活かすことができませんでした。コマンドおよび応答のパケットは、1 サイクルごとに送信され、帯域幅の一部しか利用できませんでした。最新版の System Generator for DSP では、これらを改善してパフォーマンスを向上しています。

ハードウェア協調シミュレーションの概要

次の図に、ハードウェア協調シミュレーション (HWCosim) の概要を示します。この中心となるのがテスト対象デバイス (DUT) です。DUT は通常、Simulink テスト フレームワークで開発およびテストされる IP で、スティミュラスを供給し、応答を受信 (および可能であれば評価) します。Simulink が DUT と通信できるようにするには、DUT を次のコンポーネントから構成される HWCosim ラッパーに組み込む必要があります。

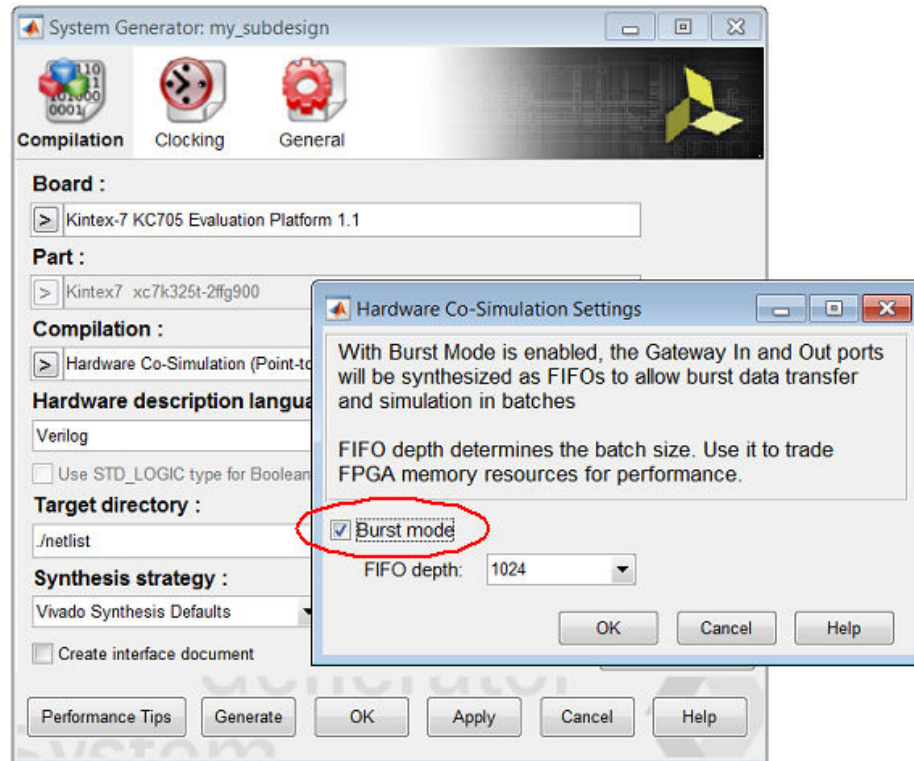
図 136: ハードウェア協調シミュレーション フロー



バースト データ転送モード

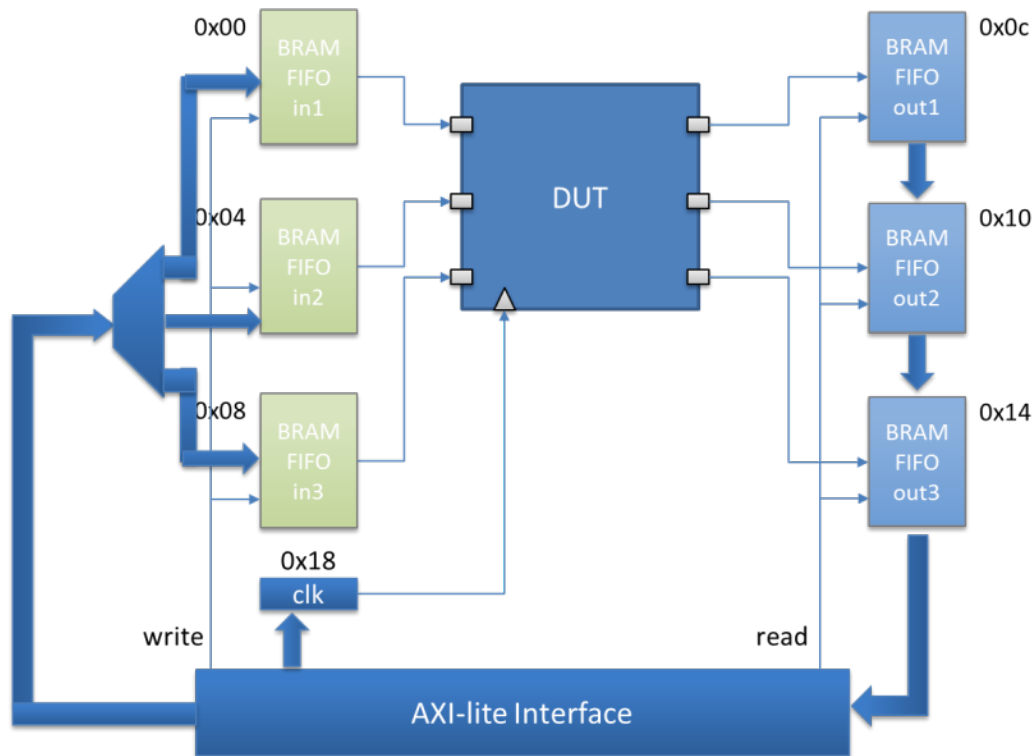
System Generator トークンでバースト データ転送モードをイネーブルにすると ([Compilation] → [Settings] → [Burst mode])、非クロック入力および出力レジスタがエントリが `n` 個の FIFO に置き換えられます。`n` の値は [FIFO depth] で選択できます。この設定は、パフォーマンスと FPGA のブロック RAM リソースの使用率とのトレードオフを調整するのに便利です。

図 137: バースト モード



[Burst mode] をオンにすると、M-HWCosim スケジューラが n 個の値のタイム シーケンスを各入力 FIFO にバーストで書き込み、入力/出力ポートのレートおよび FIFO の深さで決定されるサイクル数間クロックを実行して、出力 FIFO に出力結果を取り込むことができます。この後、スケジューラは出力 FIFO の内容を MATLAB アレイにバーストで読み出します。ここで予測データと比較できます。

図 138: バースト モード フロー



タイム サンプルのこのバッチ処理により、最大ジャンボ フレーム サイズまでの JTAG シーケンスまたはポイント ツー ポイント イーサネット フレームにデータをまとめることができるので、オーバーヘッドを大幅に削減できます。

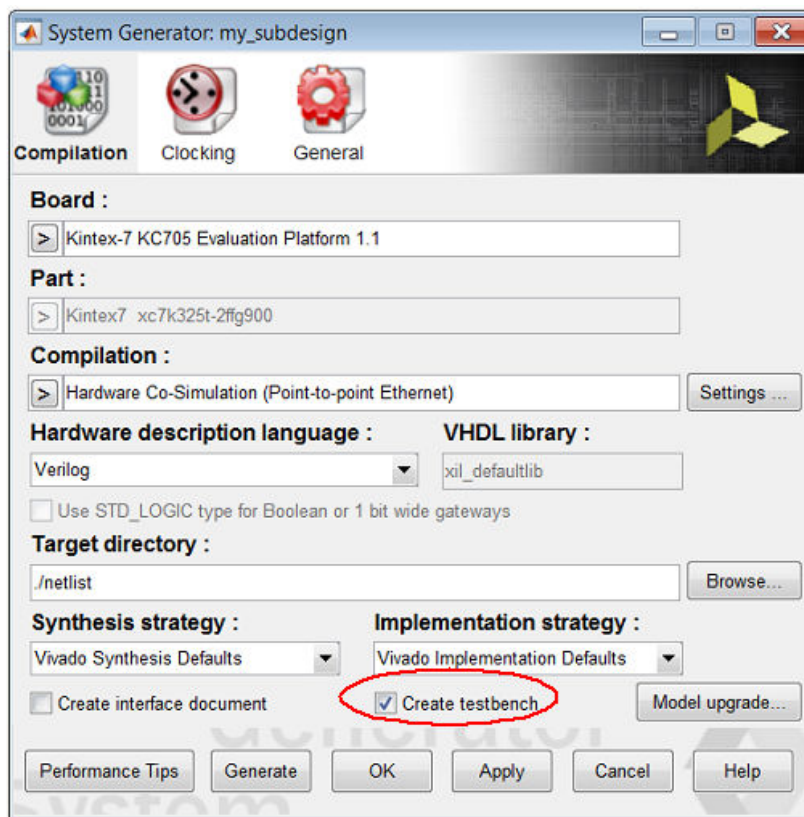
バースト データ転送モードの使用法

バースト データ転送モードを開始するには、自動生成されたテストベンチ スクリプトを使用するのが最も簡単な方法です。上級ユーザーの場合は、System Generator for DSP に含まれている MATLAB Hwcosim オブジェクトを使用した HWCosim API を利用できます。

自動テストベンチ生成

テストベンチ生成は、ハードウェア協調シミュレーション コンパイル フローで実行されます。Simulink モデルで System Generator トークンを開き、ダイアログ ボックスが開くのを待ちます。最初に表示されるタブでは [Compilation] オプションを設定できます。ドロップダウン リストには使用可能なコンパイル ターゲットが表示されます。2 つのハードウェア協調シミュレーション フローのいずれかを選択した後 (使用できるフローは選択したボードによって異なる)、[Settings] ボタンがアクティブになり、クリックするとバースト モードおよび FIFO の深さを選択するダイアログ ボックスが開きます。バースト モードをオンにすると、[Compilation] タブの下部にある [Create testbench] チェック ボックスをオンにして、M-HWCosim テストベンチ スクリプトが自動的に生成されるようになります。

図 139: テストベンチの作成



テスト ジェネレーターは次のターゲット ディレクトリに M スクリプトを生成します。

```
<design_name>_<sub_system>_hwcosim_test.m
```

MATLAB® コンソールでこのスクリプトを実行できます。このスクリプトは、ほかの Simulink ソース ブロックまたは MATLAB 変数から ザイリンクス Gateway In ブロックに駆動されるスティミュラス データを判断するため Simulink モデルも実行し、ザイリンクス ブロック デザイン (BD) により生成された予測される出力を取り込み、データを個別のデータ ファイルとしてターゲット ディレクトリ ([Target directory] で指定) にエクスポートします。

```
<design_name>_<sub_system>_<port_name>.dat.
```

テストベンチを実行するには、MATLAB のコンソールを開き、[Target Directory] で指定したディレクトリに移動して、スクリプトの名前を指定して実行します。テストにエラーが発生すると、コンソールにエラーが出力され、次のファイルにエラーが発生した箇所が比較されて出力されます。

```
<design_name>_<sub_system>_hwcosim_test.result.
```

バースト モードのテストベンチ スクリプト

次は、コンパイル フローの一部としてサンプル デザイン用に生成されたテストベンチです。



ヒント: ボードに複数のイーサネット アダプターが接続されている場合、M-Hwcosim を使用して、ポイント ツー ポイント イーサネット ハードウェア協調シミュレーション用のイーサネット インターフェイスを選択できます。インターフェイスの選択方法については、『Vivado Design Suite リファレンス ガイド: System Generator を使用したモデル ベースの DSP デザイン』(UG958) の [M-Hwcosim を使用したポイント ツー ポイント イーサネット ハードウェア協調シミュレーションのアダプターの選択](#)を参照してください。

```
%% project3_burst_hwcosim_test
% project3_burst_hwcosim_test is an automatically generated example MCode
% function that can be used to open a hardware co-simulation (hwcosim)
% target,
% load the bitstream, write data to the hwcosim target's input blocks, fetch
% the returned data, and verify that the test passed. The returned value of
% the test is the amount of time required to run the test in seconds.
% Fail / Pass is indicated as an error or displayed in the command window.

%%
% PLEASE NOTE that this file is automatically generated and gets re-created
% every time the Hardware Co-Simulation flow is run. If you modify any part
% of this script, please make sure you save it under a new name or in a
% different location.

%%
% The following sections exist in the example test function:
% Initialize Bursts
% Initialize Input Data & Golden Vectors
% Open and Simulate Target
% Release Target on Error
% Test Pass / Fail

function eta = project3_burst_hwcosim_test
eta = 0;

%%
% ncycles is the number of cycles to simulate for and should be adjusted if
% the generated testbench simulation vectors are substituted by user data.
ncycles = 10;

%%
% Initialize Input Data & Golden Vectors
% xlHwcosimTestbench is a utility function that reformats fixed-point HDL
Netlist
% testbench data vectors into a double-precision floating-point MATLAB
binary
% data array.
xlHwcosimTestbench('.', 'project3_burst');

%%
% The testbench data vectors are both stimulus data for each input port, as
% well as expected (golden) data for each output port, recorded during the
% Simulink simulation portion of the Hardware Co-Simulation flow.
% Data gets loaded from the data file ('<name>_<port>_hwcosim_test.dat')
% into the corresponding 'testdata_<port>' workspace variables using
% 'getfield(load('<name>_<port>_hwcosim_test.dat' ... ' commands.
%
% Alternatively, the workspace variables holding the stimulus and / or
golden
% data can be assigned other data (including dynamically generated data) to
% test the design with. If using alternative data assignment, please make
% sure to adjust the "ncycles" variable to the proper number of cycles, as
% well as to disable the "Test Pass / Fail" section if unused.
testdata_noise_x0 =
getfield(load('project3_burst_noise_x0_hwcosim_test.dat', '-mat'),
```

```

'values');
testdata_scale = getfield(load('project3_burst_scale_hwcossim_test.dat', '-
mat'), 'values');
testdata_wave = getfield(load('project3_burst_wave_hwcossim_test.dat', '-
mat'), 'values');
testdata_intout = getfield(load('project3_burst_intout_hwcossim_test.dat', '-
mat'), 'values');
testdata_sigout = getfield(load('project3_burst_sigout_hwcossim_test.dat', '-
mat'), 'values');

%%
% The 'result_<port>' workspace variables are arrays to receive the actual
results
% of a Hardware Co-Simulation read from the FPGA. They will be compared to
the
% expected (golden) data at the end of the Co-Simulation.
result_intout = zeros(size(testdata_intout));
result_sigout = zeros(size(testdata_sigout));

%%
% project3_burst.hwc is the data structure containing the Hardware Co-
Simulation
% design information returned after netlisting the Simulink / System
% Generator model.
% Hwcossim(project) instantiates and returns a handle to the API shared
library object.
project = 'project3_burst.hwc';
h = Hwcossim(project);
try
    %% Open the Hardware Co-Simulation target and co-simulate the design
    open(h);
    cosim_t_start = tic;
    h('noise_x0') = testdata_noise_x0;
    h('scale') = testdata_scale;
    h('wave') = testdata_wave;
    run(h, ncycles);
    result_intout = h('intout');
    result_sigout = h('sigout');
    eta = toc(cosim_t_start);
    % Release the handle for the Hardware Co-Simulation target
    release(h);

%% Release Target on Error
catch err
    release(h);
    rethrow(err);
    error('Error running hardware co-simulation testbench. Please refer to
hwcossim.log for
details.');
```

```

end

%% Test Pass / Fail
logfile = 'project3_burst_hwcossim_test.results';
logfd = fopen(logfile, 'w');
sim_ok = true;
sim_ok = sim_ok & xlHwcossimCheckResult(logfd, 'intout', testdata_intout,
result_intout);
sim_ok = sim_ok & xlHwcossimCheckResult(logfd, 'sigout', testdata_sigout,
result_sigout);
fclose(logfd);
if ~sim_ok
    error('Found errors in the simulation results. Please refer to

```

```
project3_burst_hwcossim_test.results for details. ');
end
disp(['Hardware Co-Simulation successful. Data matches the Simulink
simulation and completed in
' num2str(eta) ' seconds.']) ;
```

このスクリプトはまず、シミュレーションで実行するサイクル数 (`ncycles`) を定義し、テストベンチを準備し、ステイミュラス データおよび予測出力を MATLAB 配列に読み込みます。その後、HWCosim API 共有ライブラリを読み込むハンドル (`h`) を使用して、`xlHwcossim` オブジェクト インスタンスを作成します。try-catch ブロック内でインスタンスを開き、FPGA を初期化して、FPGA への接続を開きます。

セットアップが完了したら、`tic` と `toc` のタイミング コマンド間のコードが書き込み/実行/読み出しのコマンドを実行します。古いバージョンの HWCosim とは異なり、このテストベンチには、各クロック サイクルを通して実行する for-loop は不要です。これは、(`run(h, ncycles)` コマンドの実行中に) 細かいサイクルで書き込み/実行/読み出しのバッチ コマンドをハードウェアに実行する前に、ホスト メモリにほぼ任意サイズの書き込みコマンドを格納しておくことができる、新しいスマート キャッシュ レイヤーがあるからです。

実行フェーズの最後で、HWCosim インスタンスは開放され、テストベンチは実際の出力と予測出力と比較します。

テストベンチのコードのコメントは、ハードウェア協調シミュレーションのフローを理解し、ユーザー デザイン用にカスタマイズされたテストベンチ スクリプトを作成するのに役立ちます。

HDL モジュールのインポート

System Generator デザインに既存の HDL モジュールを 1 つまたは複数追加する場合があります。System Generator のブラック ボックス ブロックを使用すると、VHDL および Verilog をデザインに含めることができます。ブラック ボックス ブロックは、ほかの System Generator ブロックと同様に動作し、デザインに含められ、シミュレーションで使用され、ハードウェアにコンパイルされます。System Generator でブラック ボックス ブロックがコンパイルされる際、ブラック ボックスのポートは自動的に残りのデザインに接続されます。ブラック ボックスは、その周辺および System Generator トークンの設定に基づいて、同期クロック デザインまたは複数のハードウェア クロック デザインをサポートするように設定できます。

ブラック ボックス インターフェイス	
ブラック ボックス HDL の要件および制限事項	ブラック ボックスに関連付ける VHDL、Verilog、EDIF の要件および制限事項を説明します。
第 7 章: Black Box Configuration ウィザード	Black Box Configuration ウィザードの使用方法を説明します。
ブラック ボックス コンフィギュレーション M 関数	ブラック ボックス コンフィギュレーションの M 関数の作成方法を説明します。

HDL 協調シミュレーション	
HDL シミュレータの設定	ブラック ボックス ブロックで HDL を協調シミュレーションする Vivado® シミュレータまたは ModelSim の設定方法を説明します。
複数のブラック ボックスの協調シミュレーション	1 つの HDL シミュレータ セッションで複数のブラック ボックス ブロックを協調シミュレーションする方法を説明します。

ブラック ボックス HDL の要件および制限事項

ブラック ボックスに関連付けられている HDL コンポーネントは、次の System Generator の要件および制限事項に従っている必要があります。

- エンティティ 名が、デザインのほかのエンティティ 名と重複しないこと。
- 双方向ポートは HDL ブラック ボックスでサポートされていますが、System Generator ではポートとして表示されず、ネットリスト後に生成された HDL にのみ表示されます。
- Verilog ブラック ボックスの場合、モジュールおよびポートの名前は、標準の HDL 命名規則に従う必要があります。
- クロックまたはクロック イネーブルのポートは std_logic 型である必要があります。Verilog ブラック ボックスの場合は、ポートは入力 clk など非ベクター入力である必要があります。

- ブラック ボックスの HDL のクロックおよびクロック イネーブル ポートの場合、クロックおよびクロック イネーブルがペアになっている必要があります (すべてのクロックに対応するクロック イネーブルがあり、すべてのクロック イネーブルに対応するクロックがある)。ブラック ボックスには複数のクロック ポートがある場合があり、その動作はデザインの前後関係によって変わります。
 - 同期シングル クロック デザインであれば、各クロック ポートを駆動するのに 1 つのクロック ソースが使用されます。クロック イネーブルのレートのみが異なります。
 - 複数の独立したハードウェア クロックが存在するデザインであれば、クロックおよびクロック イネーブル ピンを駆動するのに 2 つの異なるクロック ソースが使用されます。
- クロック名には、「clk」という文字列を含めます (例: my_clk_1、my_ce_1)。
- クロック イネーブル名には、対応するクロック名と同じ名前にし、「clk」の部分を「ce」に置き換えます。たとえば、クロックの名前が src_clk_1 である場合、そのクロック イネーブル名は src_ce_1 にします。
- 立ち下がりエッジでトリガーされる出力データは使用できません。



重要: System Generator では、ブラック ボックス フローで IP として .dcp ファイルはインポートされません。

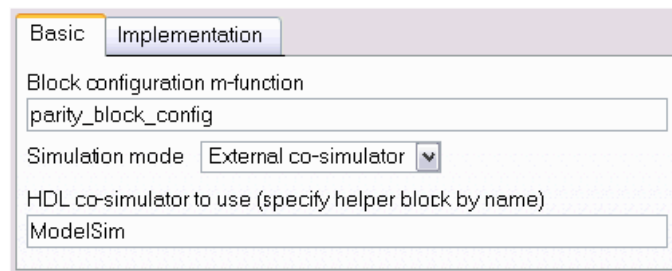
ブラック ボックス コンフィギュレーション M 関数

インポートされたモジュールは、System Generator ではブラック ボックス ブロックで表されます。インポートされたモジュールに関する情報は、コンフィギュレーション M 関数によってブラック ボックスに含められます。この関数では、ブラック ボックス ブロックのインターフェイス、インプリメンテーション、シミュレーション動作が定義されます。コンフィギュレーション M 関数では、次の情報が定義されます。

- モジュールの最上位エンティティの名前
- 言語選択 (VHDL または Verilog)
- ポートの説明
- モジュールに必要なジェネリック
- 同期シングル クロックまたは複数の非同期クロック設定
- クロッキングおよびサンプル レート
- モジュールに関連付けられているファイル
- モジュールに組み合わせパスがあるかどうか

ブラック ボックスに関連付けられているコンフィギュレーション M 関数の名前は、ブラック ボックスのパラメータ ダイアログ ボックスでパラメータとして指定します (次の図では parity_block_config.m)。

図 140: ブラック ボックスのパラメータ ダイアログ ボックス



コンフィギュレーション M 関数では、ブラック ボックス情報を指定するのにオブジェクト ベースのインターフェイスが使用されます。このインターフェイスでは、SysgenBlockDescriptor と SysgenPortDescriptor という 2 つのオブジェクトが定義されます。System Generator がコンフィギュレーション M 関数を呼び出すと、関数はブロック記述子に渡されます。

```
function sample_block_config(this_block)
```

SysgenBlockDescriptor オブジェクトには、ブラック ボックスに関する情報を指定するためのメソッドがあります。ブロック記述子のポートは、ポート記述子を使用してそれぞれ定義されます。

言語の選択

ブラック ボックスは VHDL および Verilog のモジュールをインポートできます。SysgenBlockDescriptor には setTopLevelLanguage というメソッドがあり、どのタイプのモジュールをインポートするかをブラック ボックスに伝えます。このメソッドは、コンフィギュレーション M 関数で一度呼び出す必要があります。次のコードは、VHDL および Verilog をそれぞれ選択する方法を示しています。

VHDL モジュール:

```
this_block.setTopLevelLanguage('VHDL');
```

Verilog モジュール:

```
this_block.setTopLevelLanguage('Verilog');
```

注記: ウィザードでは、コンフィギュレーション M 関数を生成するときに、適切な言語が自動的に選択されます。

最上位エンティティの指定

ブラック ボックスに関連付けられている最上位エンティティの名前は、ブラック ボックスで認識しておく必要があります。SysgenBlockDescriptor には setEntityName というメソッドがあり、最上位エンティティの名前を指定できます。

注記: エンティティ名を指定するには、小文字を使用してください。

たとえば、次のコードでは最上位エンティティに foo という名前を指定しています。

```
this_block.setEntityName('foo');
```

注記: ウィザードでは、コンフィギュレーション M 関数を生成するときに、最上位エンティティ名が自動的に設定されます。

ポート ブロックの定義

ブラック ボックスのポート インターフェイスは、そのブロックのコンフィギュレーション M 関数によって定義されます。ブラック ボックスのポートは、ポート記述子を使用して定義されます。ポート記述子には、ポート幅、データ型、2 進小数点、サンプル レートなど、さまざまなポートの属性を設定するためのメソッドがあります。

新しいポートの追加

ブラック ボックスのポート インターフェイスを定義するとき、ブロック記述子に入力および出力ポートを追加する必要があります。これらのポートはインポートするモジュールのポートに対応します。モデルでは、ブロック記述子オブジェクトで宣言されるポート名によって、ブラック ボックス ブロックのポート インターフェイスが決まります。SysgenBlockDescriptor には入力および出力ポートを追加するためのメソッドがあります。

入力ポートの追加

```
this_block.addSimulinkInport('din');
```

出力ポートの追加

```
this_block.addSimulinkOutport('dout');
```

ポート名は addSimulinkInport および addSimulinkOutport というメソッドに渡される文字列パラメーターにより指定されます。これらの名前は、インポートされたモジュールの対応ポート名と一致している必要があります。

注記: ポート名は小文字で指定してください。

双方向ポートの追加

```
config_phase = this_block.getConfigPhaseString;
if (strcmpi(config_phase,'config_netlist_interface'))
    this_block.addInoutport('bidi');
    % Rate and type info should be added here as well
end
```

双方向ポートはデザインのネットリスト生成中にのみサポートされ、System Generator の図には表示されません。生成された HDL にのみ表示されます。このため、System Generator で HDL を生成するときのみ双方向ポートを追加してください。if-end の条件文により、双方向ポートを追加するコードが実行されます。

1 つのメソッドを呼び出して、入力および出力両方のポートを定義することも可能です。setSimulinkPorts メソッドにはパラメーターを 2 つ指定できます。1 つ目のパラメーターは、ブロックの入力ポート名を定義する文字列のセル アレイです。2 つ目のパラメーターは、ブロックの出力ポート名を定義する文字列のセル アレイです。

注記: Black Box Configuration ウィザードでは、コンフィギュレーション M 関数を生成するときに、ポート名が自動的に設定されます。

ポート オブジェクトの取得

ブロック記述子にポートが追加されると、そのポートに個々の属性を設定する必要がある場合があります。ポートを設定する前に、設定しようとしているポートの記述子を取得する必要があります。SysgenBlockDescriptor には、ポートに関連付けられているポート オブジェクトにアクセスするためのメソッドがあります。たとえば、次のメソッドは this_block 記述子の din という名前のポートを取得します。

SysgenPortDescriptor オブジェクトへのアクセス

```
din = this_block.port('din');
```

上記の例では、din というオブジェクトが作成され、port 関数コールで返される記述子に割り当てられます。

SysgenBlockDescriptor には、`inport` および `outport` というメソッドもあり、ポート インデックスのポート オブジェクトを返します。ポート インデックス (ブロック インターフェイスに表示される順番) の値は、1 から始まり、ブロックの入力/出力ポートの数までになります。ブロックのポートを繰り返す必要がある場合 (エラー チェックなど) に、これらのメソッドは便利です。

ポート タイプの設定

SysgenPortDescriptor には、個々のポートを設定するメソッドがあります。たとえば、符号なし、12 ビット、位置 8 が 2 進小数点になっている `dout` というポートがあるとします。まずは、次のようなコードでこのタイプを定義できます。

```
dout = this_block.port('dout');
dout.setWidth(12);
dout.setBinPt(8);
dout.makeUnsigned();
```

同じことを次のコードでも定義できます。

```
dout = this_block.port('dout');
dout.setType('Ufix_12_8');
```

最初のコードは、個々のメソッド コールを使用してポートの属性を設定しています。2 番目のコードは、文字列で信号タイプを定義しています。どちらのコードも機能的には同じです。

ブラック ボックスでは、シングル ビット ポート (`std_logic` など) またはベクター (`std_logic_vector(0 downto 0)` など) を使用して宣言された 1 ビット ポートの HDL モジュールをサポートします。デフォルトでは、System Generator は、ポートがベクターとして宣言されます。このデフォルト設定を変更するには、記述子の `useHDLVector` メソッドを使用します。このメソッドを `true` に設定している場合は、System Generator でポートがベクターとして処理されます。`false` に設定している場合は、System Generator でポートがシングル ビットとして処理されます。

```
dout.useHDLVector(true); % std_logic_vector
dout.useHDLVector(false); % std_logic
```

注記: ウィザードでは、コンフィギュレーション M 関数を生成するときに、自動的にポート タイプが設定されます。

シミュレーション用の双方向ポートの設定

双方向ポート (入出力ポート) は、HDL ネットリスト生成中にのみサポートされます。つまり、双方向ポートは System Generator の図には表示されません。デフォルトでは、双方向ポートはシミュレーション中 X で駆動されます。ポートにデータ ファイルを関連付けると、この動作を変更できます。ブロックに双方向ポートを追加できるのは `config_netlist_interface` フェーズのみなので、このコードを必ず保護してください。

```
if (strcmpi(this_block.getConfigPhaseString,'config_netlist_interface'))
    bidi_port = this_block.port('bidi');
    bidi_port.setGatewayFileName('bidi.dat');
end
```

上記の例では、シミュレーション中に `bid1.dat` というテキスト形式のファイルが使用されます。データ ファイルはテキスト形式である必要があり、ファイルの各行は各シミュレーション サイクルでポートに駆動する信号を表します。たとえば、3 ビットの双方向ポートを 4 サイクル間シミュレーションする場合、データ ファイルは次のようになります。

```
ZZZ
110
011
XXX
```

指定のデータ ファイルが見つからない場合は、シミュレーションでエラーになります。

ポート サンプル レートの設定

ブラック ボックス ブロックでは、異なるサンプル レートのポートがサポートされます。デフォルトでは、出力ポートのサンプル レートは、入力ポート (または複数の入力と同じサンプル レートで実行されている場合は複数の入力ポート) から継承されたサンプル レートです。出力ポートのレートがブロックの入力サンプル レートとは異なる場合など、場合によってポートのサンプル レートを明示的に指定する必要があります。

注記: ブラック ボックスへの入力のサンプル レートが異なる場合は、各出力ポートのサンプル レートを指定する必要があります。

`SysgenPortDescriptor` には、ポートのレートを明示的に設定する `setRate` というメソッドがあります。

注記: `setRate` メソッドに渡されるレート パラメーターは、そのポートが動作している Simulink® のサンプル レートではなく、System Generator トークンのダイアログ ボックスで定義されている Simulink® のシステム クロック周期と、設定するポートのサンプル周期との比を定義する正の整数です。

Simulink のシステム周期が 2 秒に定義されているモデルがあり、`dout` ポートのレートが次のように `setRate` メソッドを使用して 3 に設定されているとします。

```
dout.setRate(3);
```

このレート 3 は、Simulink の 3 システム周期ごとに `dout` ポートで新しいサンプルが生成されるという意味です。Simulink のシステム周期は 2 秒なので、ポートの Simulink のサンプル レートは $3 \times 2 = 6$ 秒になります。

注記: ポートがサンプリングされない定数の場合は、`SysgenPortDescriptor` の `setConstant` メソッドを使用してコンフィギュレーション M 関数で定義できます。または、`setRate` メソッドで `Inf` を渡して定数を定義できます。

ダイナミック出力ポート

ブラック ボックスには、ダイナミック出力ポートのタイプとレートをサポートする便利な機能があります。たとえば、入力ポートの幅に基づいて出力ポート幅を設定しなければならないことがよくあります。`SysgenPortDescriptor` には、ポートの設定を決めることができるメンバー変数があります。ブロックの入力ポートでこれらのメンバー変数を確認して、出力ポートのタイプおよびレートを設定できます。

たとえば、次のようにポート (この場合は `din`) の幅およびレートを取得できます。

```
input_width = this_block.port('din').width;
input_rate = this_block.port('din').rate;
```

注記: ブラック ボックスのコンフィギュレーション M 関数は、モデルをコンパイルするときに数回呼び出されます。このコンフィギュレーション関数は、データ型およびレートがブラック ボックスに渡される前に、呼び出すことができます。

SysgenBlockDescriptor オブジェクトには、inputTypesKnown と inputRatesKnown というブール メンバー変数があり、ポート タイプおよびレートがブロックに渡されているかどうかを伝えます。入力ポート コンフィギュレーションに基づいてダイナミック出力ポート タイプまたはレートを設定している場合、inputTypesKnown および inputRatesKnown の値をチェックする条件文の中にコンフィギュレーション コールを入れ子にする必要があります。

次に、ダイナミック出力ポート dout の幅を、入力ポート din と同じ幅に設定するコード例を示します。

```
if (this_block.inputTypesKnown)
    dout.setWidth(this_block.port('din').width);
end
```

ダイナミック レートの設定も同じようになります。次のコードは、出力ポート dout のサンプル レートを入力ポート din のサンプル レートより 2 倍低速に設定します。

```
if (this_block.inputRatesKnown)
    dout.setRate(this_block.port('din').rate*2);
end
```

ブラック ボックスのクロッキング

マルチレートのモジュールをインポートするには、そのモジュールのクロッキングに関する情報をコンフィギュレーション M 関数で指定して System Generator に示す必要があります。System Generator では、クロックおよびクロック イネーブルはほかのポート タイプとは異なる方法で処理されます。インポートされたモジュールのクロックポートには、常にクロック イネーブル ポートが付随している必要があります(その逆も同様)。つまり、クロックとクロック イネーブルはペアで定義する必要があり、インポートされたモジュールでペアとして存在する必要があります。これは、シングル レート デザインでもマルチレート デザインでも同じです。

クロックとクロック イネーブルがペアになっている必要がありますが、System Generator ではインポートされたモジュールのすべてのクロック ポートが FPGA システム クロックで駆動されます。クロック イネーブル ポートは、FPGA システム クロックから派生したクロック イネーブル信号により駆動されます。

SysgenBlockDescriptor には、ブラック ボックスのクロックおよびクロック イネーブル情報を定義するための addClkCEPair があります。このメソッドには、3 つのパラメーターを指定できます。最初のパラメーターは、モジュールに表示されるクロック ポートの名前を定義します。2 番目のパラメーターは、モジュールに表示されるクロック イネーブル ポートの名前を定義します。

クロックとクロック イネーブルのペアのポート名は、次の命名規則に従う必要があります。

- クロック ポートには clk という文字列を含めます。
- クロック イネーブル ポートには ce という文字列を含めます。
- clk および ce を含む文字列を同じにします (例: my_clk_1 と my_ce_1)。

3 番目のパラメーターは、クロックとクロック イネーブル ポートのレート関係を定義します。このレート パラメーターは、Simulink サンプル レートではなく、クロックのサンプル周期と必要なクロック イネーブルのサンプル周期の関係を System Generator に示すものです。レート パラメーターの値は、クロック レートとそれに対応するクロック イネーブル レートの比を定義する整数値です。

たとえば、ce_3 という名前のクロック イネーブル ポートがあり、システム クロック周期の 3 倍の周期に設定するとします。このクロック イネーブル ポートは、次の関数読み出しで定義されます。

```
addClkCEPair('clk_3','ce_3',3);
```

System Generator でブラック ボックスがハードウェアにコンパイルされるときに、モジュールに適切なクロック イネーブル信号が生成され、適切なクロック イネーブル ポートに自動的に接続されます。

組み合わせパス

クロック イベントなしに、入力での変化が出力ポートに影響するような組み合わせパスがインポートするモジュールに少なくとも 1 つ含まれる場合、そのことをコンフィギュレーション M 関数で示す必要があります。

SysgenBlockDescriptor オブジェクトには、モジュールに組み合わせパスがあることを示す

tagAsCombinational メソッドがあります。これは、コンフィギュレーション M 関数で次のように呼び出します。

```
this_block.tagAsCombinational;
```

VHDL ジェネリックおよび Verilog パラメーターの指定

System Generator でモデルを HDL にコンパイルする際に、モジュールに渡すジェネリックのリストを指定できます。ジェネリックに割り当てられている値は、マスク パラメーターおよび伝搬されたポート情報 (ポート幅、タイプ、レートなど) から抽出できます。ジェネリックは柔軟に割り当てることができるので、ブラック ボックス周辺の Simulink 環境に基づいて詳細にカスタマイズされるモジュールをサポートできます。

addGeneric メソッドを使用して、デザインがハードウェアにコンパイルされるときにモジュールに渡す必要のあるジェネリックを定義できます。次に、VHDL の整数ジェネリック dout_width を 12 に設定する例を示します。

```
addGeneric('dout_width','Integer','12');
```

伝搬された入力ポートの情報に基づいて出力ポートのジェネリック値を設定することも可能です (ダイナミック出力ポートの幅を指定するジェネリックなど)。

ブラック ボックスのコンフィギュレーション M 関数は、モデルがコンパイルされるときに何度か異なるタイミングで呼び出されるため、データ型 (またはレート) がブラック ボックスに伝搬される前に、コンフィギュレーション関数を呼び出すことが可能です。入力ポートのタイプまたはレートに基づいてジェネリック値を設定する場合、inputTypesKnown または inputRatesKnown 変数の値をチェックする条件文内に addGeneric 呼び出しを含める必要があります。たとえば、次のように dout ポートの幅を din の値に基づいて設定できます。

```
if (this_block.inputTypesKnown)
    % set generics that depend on input port types
    this_block.addGeneric('dout_width', ...
        this_block.port('din').width);
end
```

ジェネリック値は、ブラック ボックスに関連付けられているマスク パラメーターに基づいて設定できます。SysgenBlockDescriptor には、Simulink でのブラック ボックス名を文字列で表した blockName というメンバー変数があります。この変数を使用して、特定のコンフィギュレーション M 関数に関連付けられているブラック ボックスにアクセスできます。たとえば、ブラック ボックスで init_value というパラメーターを定義するとします。init_value というジェネリックは、次のように設定できます。

```
simulink_block = this_block.blockName;
init_value = get_param(simulink_block,'init_value');
this_block.addGeneric('init_value','String',init_value);
```

注記: 次を実行すると、ユーザー指定のパラメーター (ジェネリック値を指定する値など) をブラック ボックスに追加できます。

- ブラック ボックスを Simulink ライブラリまたはモデルにコピーします。

- ブラック ボックスのリンクを解除します。
- 必要なパラメーターをブラック ボックスのダイアログ ボックスに追加します。

ブラック ボックスの VHDL ライブラリ サポート

このブラック ボックス機能を使用すると、ライブラリの依存関係があらかじめ定義されている VHDL モジュールをインポートできます。このインポート方法を例をあげて説明します。

次の VHDL モジュールは、非同期クリアを持つ 4 ビットのアップ カウンターです (async_counter.vhd)。これは、async_counter_lib というライブラリにコンパイルされます。

```

1  -- 4-bit, Up counter, with asynchronous clear
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5  entity async_counter is
6  port (clk, clr : in std_logic;
7        ce: in std_logic := '1'; |
8        q : out std_logic_vector(3 downto 0));
9  end async_counter;
10 architecture archi of async_counter is
11     signal tmp: std_logic_vector(3 downto 0);
12 begin
13     process (clk, clr)
14     begin
15         if (clr='1') then
16             tmp <= "0000";
17         elsif (clk'event and clk='1') then
18             tmp <= tmp + 1;
19         end if;
20     end process;
21     q <= tmp;
22 end archi;

```

次の VHDL モジュールは、同期クリアを持つ 4 ビットのアップ カウンターです (sync_counter.vhd)。これは、sync_counter_lib というライブラリにコンパイルされます。

```

1  -- 4-bit, Up counter, with synchronous clear
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity sync_counter is
7  port (clk, clr : in std_logic;
8        ce: in std_logic := '1'; |
9        q : out std_logic_vector(3 downto 0));
10 end sync_counter;
11 architecture archi of sync_counter is
12     signal tmp: std_logic_vector(3 downto 0);
13 begin
14     process (clk)
15     begin
16         if (clk'event and clk='1') then
17             if (clr='1') then
18                 tmp <= "0000";
19             else
20                 tmp <= tmp + 1;
21             end if;
22         end if;
23     end process;
24     q <= tmp;
25 end archi;

```

次の VHDL モジュールは、前の 2 つのモジュールをインスタンス化するために使用される最上位モジュールです。System Generator モデルにブラック ボックスを追加する際は、このモジュールを指定する必要があります。

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  library sync_counter_lib;
5  use sync_counter_lib.all;
6  library async_counter_lib;
7  use async_counter_lib.all;
8
9
10 entity top_level is
11 port (clk, clr : in std_logic;
12       ce: in std_logic := '1';
13       q_sync : out std_logic_vector(3 downto 0);
14       q_async : out std_logic_vector(3 downto 0)
15 );
16 end top_level;
17
18 architecture structural of top_level is
19 component async_counter
20 port (
21     clk, clr, ce: in std_logic;
22     q: out std_logic_vector(3 downto 0));
23 end component;
24
25 component sync_counter
26 port (
27     clk, clr, ce: in std_logic;
28     q: out std_logic_vector(3 downto 0));
29 end component;
30
31 begin
32 counter_0: entity async_counter_lib.async_counter
33 port map (
34     ce => ce,
35     q  => q_async,
36     clk => clk,
37     clr => clr
38 );
39 counter_1: entity sync_counter_lib.sync_counter
40 port map (
41     ce => ce,
42     q  => q_sync,
43     clk => clk,
44     clr => clr
45 );
46 end structural;

```


Define libraries using "library" and "use" clauses

この VHDL をインポートするには、ブラック ボックスを使用して、まず最上位エンティティ `top_level` をインポートします。

ファイルがインポートされたら、関連付けられているブラック ボックスのコンフィギュレーション M ファイルを次のように変更する必要があります。

```
% Add additional source files as needed.
% |-----
% | Add files in the order in which they should be compiled.
% | If two files "a.vhd" and "b.vhd" contain the entities
% | entity_a and entity_b, and entity_a contains a
% | component of type entity_b, the correct sequence of
% | addFile() calls would be:
% |   this_block.addFile('b.vhd');
% |   this_block.addFile('a.vhd');
% |-----
%   this_block.addFile('');
%   this_block.addFile('');
this_block.addFile('top.vhd');
this_block.addFileToLibrary('async_counter.vhd','async_counter_lib');
this_block.addFileToLibrary('sync_counter.vhd','sync_counter_lib');
```

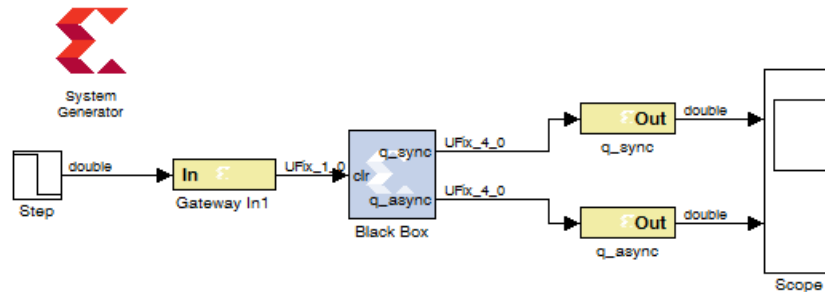
Specifying library names by using
"addFileToLibrary" command



インターフェイス関数 `addFileToLibrary` は、ライブラリ名を `work` 以外のものに指定し、関連付けられている HDL ソースを指定ライブラリにコンパイルするよう指示するために使用されます。

System Generator モデルは次の図のようになります。

図 141: ブラック ボックス サポートを含む System Generator モデル



次に、System Generator トークンをダブルクリックし、[Generate] をクリックして HDL ネットリストを生成します。

この生成プロセスで Vivado IDE プロジェクト (.xpr) が作成され、`netlist` フォルダーの下にある `hdl_netlist` フォルダーに保存されます。Vivado IDE プロジェクトをダブルクリックし、[Source] ウィンドウの [Libraries] ビューを選択すると、`work` ライブラリと共に、`async_counter_lib` ライブラリおよび `sync_counter_lib` ライブラリが表示されます。

エラー チェック

ブラック ボックスのポート タイプ、レート、マスク パラメーターに対してエラー チェックを実行する必要があることが頻繁にあります。SysgenBlockDescriptor には `setError` というメソッドがあり、ユーザーに表示されるエラー メッセージを指定できます。`setError` に渡される文字列パラメーターが、ユーザーに表示されるエラー メッセージとなります。

ブラック ボックス API

SysgenBlockDescriptor のメンバー変数

データ型	メンバー	説明
文字列	entityName	エンティティまたはモジュールの名前。
文字列	blockName	ブラック ボックス ブロックの名前。
整数	numSimulinkInports	ブラック ボックスの入力ポートの名前。
整数	numSimulinkOutports	ブラック ボックスの出力ポートの名前。
ブール型	inputTypesKnown	すべての入力タイプが定義されている場合は true、そうでない場合は false。
ブール型	inputRatesKnown	すべての入力レートが定義されている場合は true、そうでない場合は false。
浮動小数点の配列	inputRates	入力ポートのサンプル周期配列 (inport(indx) でインデックス化)。サンプル周期の値は、マスターの System Generator トークンで指定されている Simulink システム周期の整数の倍数として表されます。
ブール型	error	エラーが検出された場合は true、それ以外は false。
文字列の配列	errorMessages	このブロックのすべてのエラー メッセージ。

SysgenBlockDescriptor のメソッド

メソッド	説明
setTopLevelLanguage(language)	ブラック ボックスの最上位エンティティ (またはモジュール) の言語を宣言します。language には VHDL または Verilog を指定します。
setEntityName(name)	エンティティまたはモジュールの名前を設定します。
addSimulinkInport(pname)	ブラック ボックスに入力ポートを追加します。pname にはポート名を指定します。
addSimulinkOutport(pname)	ブラック ボックスに出力ポートを追加します。pname にはポート名を指定します。
setSimulinkPorts(in,out)	ブラック ボックスに入力および出力ポートを追加します。in には入力ポート名、out には出力ポート名を指定します。
addInoutport(pname)	ブラック ボックスに双方向ポートを追加します。pname にはポート名を指定します。双方向ポートは、コンフィギュレーションの config_netlist_interface フェーズでのみ追加可能です。
tagAsCombinational()	ブロックに、入力ポートから出力ポートまでの組み合わせパス (直接フィードスルー) があることを示します。
addClkCEPair(clkPname, cePname, rate)	ブロックのクロック/クロック イネーブル ポートのペアを定義します。clkPname にはクロック ポート名、cePname にはクロック イネーブル ポート名、rate (double 型) にはポートのペアが実行されるレートを指定します。レートは正の整数である必要があります。クロック名には clk、クロック イネーブル名には ce という文字列を含める必要があります。クロック名とクロック イネーブル名はベース名は同じにし、それに clk または ce を付けたものにしてください。
port(name)	指定した名前に一致する SysgenPortDescriptor を返します。

メソッド	説明
inport(indx)	指定した入力ポートを記述する SysgenPortDescriptor を返します。indx は検索するポートのインデックスを指定し、1 から numInputPorts までの値を指定します。
outport(indx)	指定した出力ポートを記述する SysgenPortDescriptor を返します。indx は検索するポートのインデックスを指定し、1 から numOutputPorts までの値を指定します。
addGeneric(identifier, value)	ブロックのジェネリック (Verilog の場合はパラメーター) を定義します。identifier にはジェネリック名を指定し、value には double 型または string 型の値を指定します。ジェネリックのデータ型は値のデータ型から推論されます。値が integral double (4.0 など) の場合は、ジェネリックのデータ型は integer に設定されます。integral double 以外の値の場合は、real 型に設定されます。値が 0101 など 0 および 1 のみを含む文字列の場合は、データ型は bit_vector に設定されます。それ以外の文字列値の場合は、string 型に設定されます。
addGeneric(identifier, type, value)	ブロックのジェネリック (Verilog の場合はパラメーター) の名前、データ型、値を明示的に指定します。3 つの引数は文字列で、identifier は名前、type はデータ型、value は値です。
addFile(fn)	ブラック ボックスに関連付けられているファイルのリストにファイル名を追加します。fn はファイル名です。通常、HDL ファイルがブラック ボックスに関連付けられていますが、どのタイプのファイルでも追加できます。VHDL ファイル名は .vhd、Verilog ファイル名は .v で終わる必要があります。ファイルを追加した順序は保持され、HDL ファイルはその順にコンパイルされます。ファイル名は絶対名または相対名にできます。相対ファイル名は、MDL ファイルまたはライブラリのディレクトリに相対して処理されます。
getDeviceFamilyName()	ブラック ボックスに対応する FPGA デバイスの名前を取得します。
getConfigPhaseString	現在のコンフィギュレーション フェーズを文字列として返します。有効な戻り値 (文字列) は、config_interface、config_rate_and_type、config_post_rate_and_type、config_simulation、config_netlist_interface および config_netlist です。
setSimulatorCompilationScript(script)	ブラック ボックスが生成するデフォルトの HDL 協調シミュレーション コンパイル スクリプトを上書きします。script には使用するスクリプト名を指定します。たとえば、ブラック ボックスの HDL が変更されていない状態でシミュレーションを繰り返す場合、コンパイル フェーズを省くために、このメソッドを使用できます。
setError(message)	エラーが発生したことを示し、エラー メッセージを記録します。message にはエラー メッセージを指定します。

SysgenPortDescriptor のメンバー変数

データ型	メンバー	説明
文字列	name	ポートの名前を指定します。
整数	simulinkPortNumber	Simulink® でこのポートのインデックスを指定します。インデックスは 1 から始まります (Simulink と同様)。
ブール型	typeKnown	ポート タイプがわかっている場合は true、そうでない場合は false です。
文字列	type	ポートのタイプ (UFix_<n>_、Fix_<n>_、Bool など)。

データ型	メンバー	説明
ブール型	isBool	ポート タイプがブール型の場合は true、そうでない場合は false です。
ブール型	isSigned	タイプが符号付きの場合は true、そうでない場合は false です。
ブール型	isConstant	ポートが定数の場合は true、そうでない場合は false です。
整数	width	ポート幅を指定します。
整数	binpt	2 進小数点の位置を 0 ~ width の整数で指定します。
ブール型	rateKnown	レートがわかっている場合は true、そうでない場合は false です。
double	rate	ポートのサンプル レートを指定します。レートは MATLAB® double として表した正の整数です。レートを無限にして、ポートが定数出力することを示すこともできます。

SysgenPortDescriptor のメソッド

メソッド	説明
setName(name)	このポートに使用する HDL 名を設定します。
setSimulinkPortNumber(num)	Simulink® でこのポートに関連付けられているインデックスを設定します。num は割り当てるインデックスを指定します。インデックスは 1 から始まります (Simulink と同様)。
setType(typeName)	このポートのタイプを設定します。タイプには、ブール、UFix_<n>_、Fix_<n>_、符号付き、または符号なしのいずれかを指定します。最後の 2 タイプを選択しても、幅および 2 進小数点の位置は変わりません。 XFloat_<exponent_bit_width>_fraction_bit_width> もサポートされています。例: ap_return_port = this_block.port('ap_return'); ap_return_port.setType('XFloat_30_2');
setWidth(w)	このポートの幅を w に設定します。
setBinpt(bp)	このポートの 2 進小数点の位置を bp に設定します。
makeBool()	このポートをブールにします。
makeSigned()	このポートを符号付きにします。
makeUnsigned()	このポートを符号なしにします。
setConstant()	このポートを定数にします。
setGatewayFileName(filename)	このポートのシミュレーションおよびテストベンチ生成で使用する DAT ファイル名を設定します。シミュレーション中にユーザーのデータ ファイルが使用できるように、双方向ポートにのみ使用する関数です。このパラメーターを入力および出力ポートに設定するのは無効で、無視されます。
setRate(rate)	このポートのレートを指定します。rate には正の整数で指定します (MATLAB® の double または定数の場合は Inf)
useHDLVector(s)	1 ビット ポートを、シングル ビット (std_logic など) またはベクター (std_logic_vector(0 downto 0) など) に指定します。
HDLTypeIsVector()	1 ビット ポートを std_logic_vector(0 downto 0) に設定します。

ブラック ボックスでの複数の独立クロックのサポート

ポート接続のデザイン ルール チェック

複数の独立したハードウェア クロックのあるデザインでブラック ボックスが使用される場合、ポート接続をチェックするデザイン ルール チェック (DRC) をコンフィギュレーション M 関数に追加する必要があります。これは、異なるクロック ソースを持つポート接続が無効でないか、間違っていないかをチェックするためのものです。すべてのポート信号が、正しくクロック供給されているサブシステム インターフェイスに接続されていることを確認する必要があります。

特定のクロック ドメインからのポートのリストを指定し、それをグループにまとめるには、`checkPortsOfSameClockDomain()` というユーティリティを使用する必要があります。この API (アプリケーション プログラミング インターフェイス) の入力引数は、まず `SysgenBlockDescriptor` オブジェクトを指定し、その後特定のクロック ドメインに関連付けられているポート名のリストを指定します。

次の例では、API がエラー チェックを実行し、4 つのポートが同じサブシステムのクロック ドメインに接続されていることを確認しています。

```
checkPortsOfSameClockDomain (<block_descriptor>, '<port_name_1>',  
'<port_name_2>',  
'<port_name_3>', '<port_name_4>');
```

ポート サンプル レートの設定

複数のクロックを使用するハードウェア デザインでは、ポート インターフェイスのクロック周期は、接続されたクロックが供給されるサブシステム ドメインを使用して計算する必要があります。デフォルトでは、同期システム クロック ソースはすべてのポートに使用されますが、非同期クロック ハードウェア デザインでは、各ポートのクロック ソースを明示的に指定する必要があります (出力ポート クロックがブロックの入力ポート クロックとは異なる場合など)。

注記: 独立したクロックを複数使用するブラック ボックス デザインの出力ポートに対しては、すべてサンプル レートを 1.0 に設定する必要があります。そうすると、出力ポートはデスティネーション クロック サブシステム周期に自動的に設定されます。

`SysgenPortDescriptor` には、ポートのレートの明示的な設定に使用可能な `setRate` というメソッドがあります。

例:

```
port('<port_name>').setRate(1.0)
```

ブラック ボックスのクロッキング

同期または非同期のブラック ボックス モジュールをインポートするには、そのモジュールのクロッキングに関する System Generator 情報をコンフィギュレーション M 関数に通知する必要があります。System Generator では、クロックおよびクロック イネーブルはほかのポート タイプとは異なる方法で処理されます。インポートされたモジュールのクロックポートには、常に対応するクロック イネーブル ポートが必要です (その逆も同様)。つまり、クロックとクロック イネーブルはペアで定義する必要があり、インポートされたモジュールでペアとして存在する必要があります。これは、同期クロックが 1 つのデザインでも複数の独立クロックを持つデザインでも同じです。

SysgenBlockDescriptor には addClkCEPair というメソッドがあり、クロック サブシステム ドメインを使用してクロック、クロック イネーブル、およびクロック周期を定義するのに使用できます。クロック ドメイン情報は、同期シングル クロック デザインには不要です。

最初のパラメーターは、モジュールに表示されるクロック ポートの名前を定義します。2 番目のパラメーターは、モジュールに表示されるクロック イネーブル ポートの名前を定義します。

クロックとクロック イネーブルのペアのポート名は、次の命名規則に従う必要があります。

- クロック ポートには `clk` という文字列を含めます。
- クロック イネーブル ポートには `ce` という文字列を含めます。
- `clk` および `ce` 以外の文字列を同じにします (例: `my_clk_1` と `my_ce_1`)。

3 番目のパラメーターは、クロックとクロック イネーブル ポートのレートの関係を定義します。このレート パラメーターは、Simulink® サンプル レートではなく、クロックのサンプル周期と必要なクロック イネーブルのサンプル周期の関係を System Generator に示すものです。レート パラメーターの値は、クロック レートとそれに対応するクロック イネーブル レートの比を定義する整数値です。

複数の独立したクロックを持つデザインでは、4 番目および 5 番目のパラメーターは必須です。

4 番目のパラメーターは、クロックとクロック イネーブルのペアをグラウンドに接続するかどうかをブール値で定義します。`true` に設定すると、クロックおよびクロック イネーブルの両方がシミュレーション中にグラウンドに接続されます。`false` に接続すると、クロックおよびクロック イネーブルのレートが変わります。

5 番目のパラメーターは、クロック/クロック イネーブルのペアのクロック周期を定義します。複数の独立クロックを持つデザインでは、クロック周期を設定するのに、ブラック ボックス SysgenPortDescriptor の clockDomain プロパティを使用する必要があります。

例:

```
rate_data = this_block.port('<port_name>').rate;
clkDomain_data = this_block.port('<port_name>').clockDomain;
this_block.addClkCEPair('clk','ce',rate_data, false, clkDomain_data);
```

HDL 協調シミュレーション

ここでは、ザイリンクス ブロック、HDL モジュールと Simulink ブロック デザインを含む混合言語/混合フローのデザインをシミュレーションする方法を説明します。

System Generator では、ブラック ボックスのシミュレーションを実行するとき、HDL シミュレータが自動的に起動して、随時追加 HDL を生成し (HDL テストベンチのようなもの)、HDL をコンパイルし、シミュレーション イベントをスケジュールし、Simulink と HDL シミュレータ間のデータ通信が処理されます。これを HDL 協調シミュレーションと呼びます。

HDL シミュレータの設定

ブラック ボックス HDL は、Vivado® シミュレータまたは Model Technology 社の ModelSim シミュレータへの System Generator インターフェイスを使用して、Simulink® で協調シミュレーションできます。

ザイリンクス シミュレータ

ブラック ボックスに関連付けられている HDL の協調シミュレーションにザイリンクス シミュレータを使用するには、ブラック ボックスの [Simulation mode] で [Vivado Simulator] を選択します。これでモデルをシミュレーションできるようになり、HDL 協調シミュレーションが自動的に実行されます。

ModelSim シミュレータ

Model Technology 社の ModelSim シミュレータを使用するには、ザイリンクス ブロックセットのツール ライブラリに表示される ModelSim ブロックを Simulink 図に追加する必要があります。

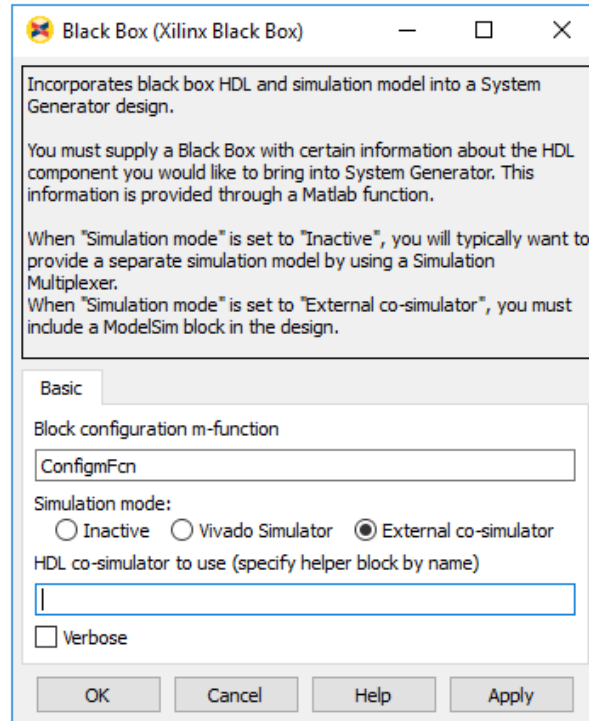
図 142: **ModelSim** ブロック



ModelSim シミュレータを使用して協調シミュレーションする各ブラック ボックスのパラメーター設定ダイアログ ボックスを開き、追加したブラック ボックスの ModelSim セッションを使用するように設定します。次の 2 つの設定を変更します。

1. [Simulation Mode] を [Inactive] から [External co-simulator] に変更します。
2. [HDL co-simulator to use] に ModelSim ブロックの名前 (ModelSim など) を入力します。

図 143: ブラック ボックスのパラメーター



ModelSim ブロックのパラメーター設定ダイアログ ボックスには、ModelSim セッションのさまざまなオプションを設定するためのパラメーターがいくつかあります。詳細は、ブロックのヘルプ ページを参照してください。モデルはこれらのオプション設定でシミュレーションできるようになり、HDL 協調シミュレーションが自動的に実行されます。

複数のブラック ボックスの協調シミュレーション

System Generator では、多くのブラック ボックスで共通の ModelSim 協調シミュレーション セッションを共有できます。たとえば、複数のブラック ボックスを、同じ ModelSim ブロックを使用するように設定できます。この場合、System Generator によりすべてのブラック ボックス HDL コンポーネントが 1 つの共通の最上位協調シミュレーション コンポーネントにまとめられます。ただし、Simulink® シミュレーションで複数のブラック ボックスを協調シミュレーションするのに必要な ModelSim シミュレーション ライセンスは 1 つだけです。

複数のブラック ボックスは、各ブラック ボックスの [Simulation mode] のオプションに [Vivado Simulator] を選択することにより、Vivado シミュレータで協調シミュレーションすることもできます。

Black Box Configuration ウィザード

System Generator には、VHDL または Verilog モジュールをブラック ボックス ブロックに簡単に関連付けるためのウィザードがあります。このウィザードは、インポートしようとしている VHDL または Verilog モジュールを解析し、その結果に基づいてコンフィギュレーション M 関数を作成して、この M 関数をモデルのブラック ボックス ブロックに関連付けます。コンフィギュレーション M 関数をそのまま使用できるかどうかは、インポートする HDL がどれくらい複雑であるかによります。コンフィギュレーション M 関数は、ウィザードでは指定できない詳細を手動で指定するため、カスタマイズする必要がある場合があります。コンフィギュレーション M 関数の作成の詳細は、[ブラック ボックス コンフィギュレーション M 関数](#)を参照してください。

ウィザードの使用

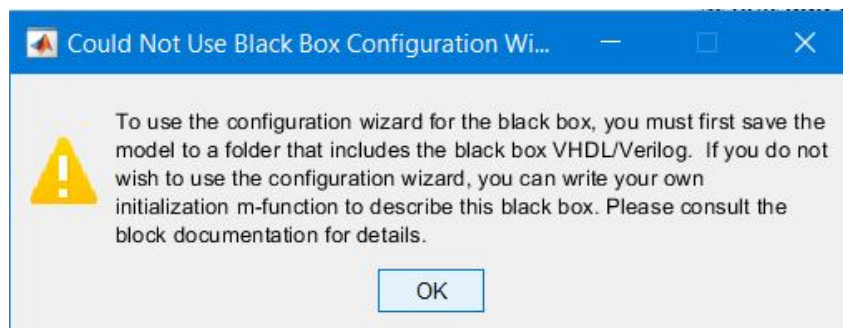
Black Box Configuration ウィザードは、新しいブラック ボックス ブロックがモデルに追加されると自動的に開きます。

注記: ウィザードを実行する前に、インポートしている VHDL または Verilog が[ブラック ボックス HDL の要件および制限事項](#)を満たしていることを確認してください。

モジュールがウィザードで検出されるようにするには、インポートしようとしているモジュールと同じディレクトリに保存する必要があります。

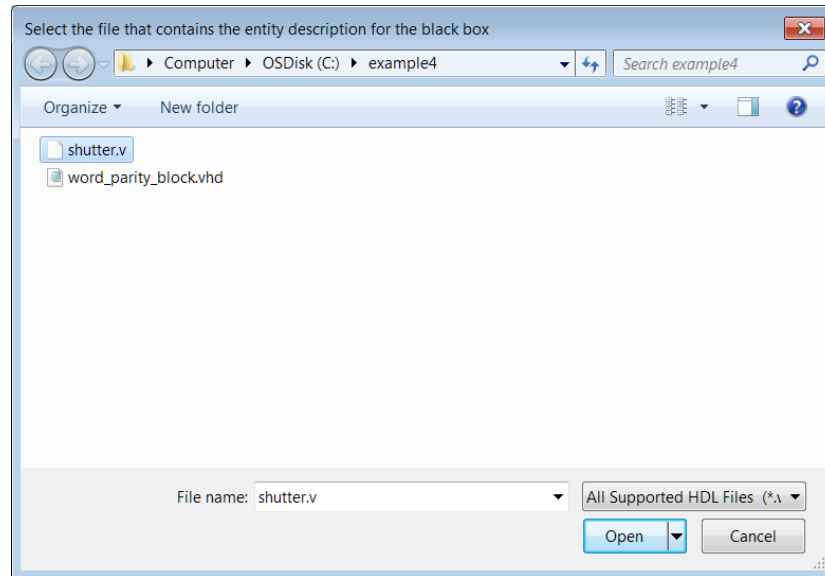
注記: ウィザードでは、モデルと同じディレクトリにある `.vhd` および `.v` ファイルのみが検索されます。ウィザードでファイルが検出されない場合、警告メッセージが表示され、ブラック ボックスは自動的に設定されません。表示される警告メッセージは次のようになります。

図 144: 警告メッセージ



モデルのディレクトリで `.vhd` および `.v` ファイルが検索され、インポート可能なファイルがリストされた新しいウィンドウが開きます。次に、そのウィンドウの例を示します。

図 145: インポートするファイル



インポートするファイルを選択し、[Open] をクリックします。これで、コンフィギュレーション M 関数が生成され、ブラック ボックス ブロックにこれが関連付けられます。

注記: コンフィギュレーション M 関数は、`<module>_config.m` が保存されているモデルのディレクトリに保存されます。`<module>` は、インポートしているモジュールの名前です。

Black Box Configuration ウィザードの詳細設定

Black Box Configuration ウィザードを実行すると、インポートされたモジュールからある程度の情報が自動的に抽出されますが、手動で指定する必要がある部分もあります。手動設定が必要なのは次のものです。

注記: コンフィギュレーション関数には、手動変更が必要な部分を示すコメントが含まれています。

- モデルに組み合わせパスがある場合、ブロックの `SysgenBlockDescriptor` オブジェクトの `tagAsCombinational` メソッドを呼び出す必要があります。複数の独立したハードウェア クロックが存在するデザインでは、組み合わせパスはサポートされません。
- Black Box Configuration ウィザードでは、インポートされる最上位エンティティのみが認識されます。通常、このエンティティに伴うほかのファイルがあります。ファイルごとに `addFile` メソッドを呼び出して、これらのファイルをコンフィギュレーション M 関数に手動で追加する必要があります。
- Black Box Configuration ウィザードでは、1 つの同期クロックのブラック ボックス記述子または複数の非同期クロックのブラック ボックス記述子が自動的に作成されます。
 - シングル レートのブラック ボックスの場合、ブラック ボックスの各ポートは同じレートで動作します。ほとんどの場合はこれで問題はありませんが、ポート レートを明示的に設定すると、シミュレーション時間が短縮される可能性があります。
 - 複数クロック ブラック ボックスの場合、入力ポート レートはソース クロック サブシステムから派生している必要があり、出力ポート レートはデスティネーション クロック サブシステムを基に設定する必要があります。場合によっては、必要なコンフィギュレーションに対してポート レートを明示的に設定するのが望ましい場合があります。

System Generator のコンパイル タイプ

System Generator では、複数の方法でデザインを同等の下位レベルの表記にコンパイルできます。デザインのコンパイル方法は、[System Generator] ダイアログ ボックスの設定によって異なります。さまざまなコンパイル タイプがサポートされているので、デザイン環境に合った記述を選択できます。たとえば、そのデザインが大規模システムのコンポーネントとして使用される場合は、HDL ネットリストまたは IP カタログが適しています。

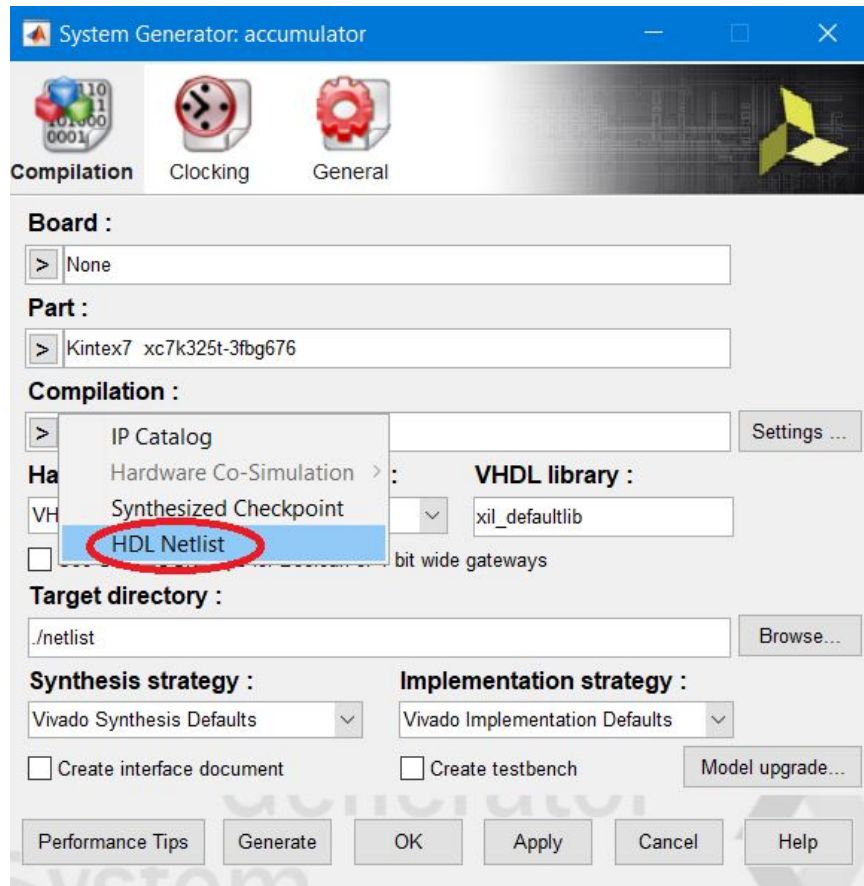
HDL ネットリスト コンパイル	デザインをインプリメントする HDL ファイルの生成方法を説明します。
ハードウェア協調シミュレーション コンパイル	Simulink® および ModelSim で使用可能な FPGA ハードウェアにデザインをコンパイルするための、System Generator 設定方法を説明します。
IP カタログのコンパイル	System Generator デザインを Vivado® IP カタログに追加可能な IP コアとしてパッケージする方法を説明します。 System Generator はデフォルトの生成ターゲットとして IP カタログのコンパイル タイプを使用します。
合成済みチェックポイントのコンパイル	Vivado 統合設計環境 (IDE) プロジェクトで利用できる合成済みチェックポイント ファイル (synth_1.dcp) の生成方法を説明します。

HDL ネットリスト コンパイル

コンパイル タイプを [HDL Netlist] に設定すると、デザインをインプリメントする HDL ファイルが生成されます。HDL ネットリスト コンパイル フローに関する詳細は、[コンパイル結果](#)を参照してください。

[HDL Netlist] コンパイル ターゲットを選択するには、次の図に示すように、System Generator トークン ダイアログ ボックスの [Compilation] をクリックして [HDL Netlist] をクリックします。

図 146: HDL ネットリスト



[Board] および [Part] フィールドでは、HDL ネットリスト [HDL Netlist] コンパイルでターゲットにするボードまたはパーツを指定できます。[Board] を選択すると、[Part] フィールドに [Board] で選択したボード上にあるサイリンクスデバイスの名前が自動的に表示されます。このパーツ名は変更できません。

HDL ネットリスト コンパイルは、Vivado ツールでサポートされるどのボードまたはパーツに対しても実行できます。Vivado のインストールに含まれるサイリンクス開発ボードだけでなく、パートナー ボードまたはカスタム ボードも指定できます ([System Generator でのボード サポートの指定](#) を参照)。

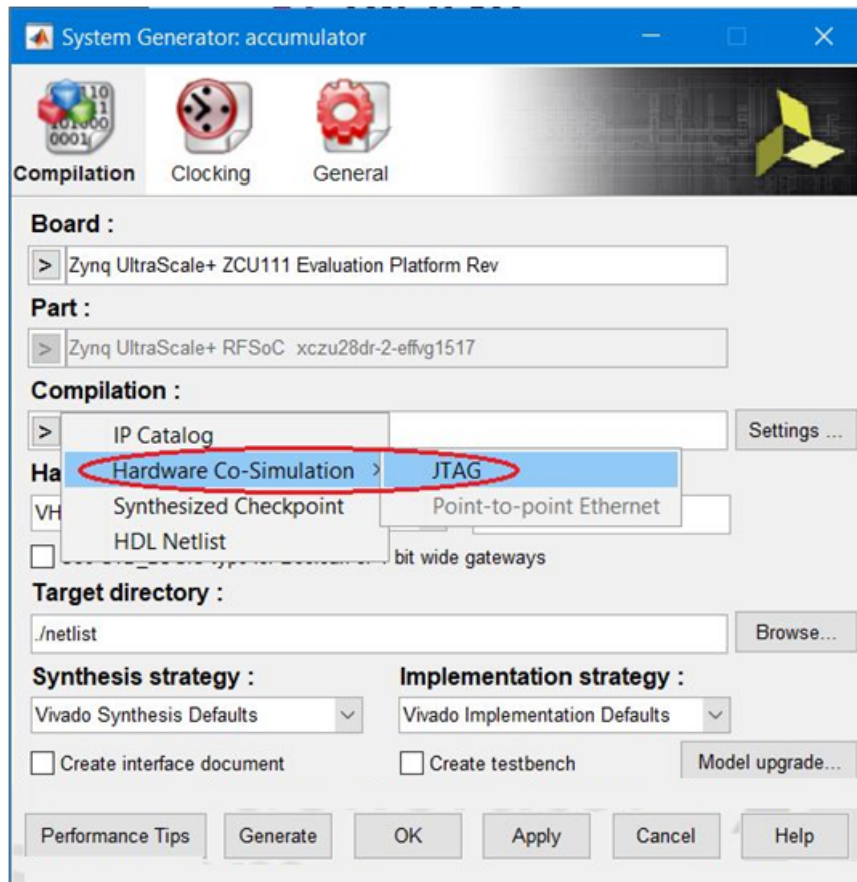
HDL ネットリスト コンパイルで生成されたファイルは、[Target directory] フィールドで指定したディレクトリの下に `hdl_netlist` ディレクトリに含まれます。これらのファイルについては、[コンパイル結果](#) を参照してください。

ハードウェア協調シミュレーション コンパイル

System Generator では、Simulink® シミュレーションとのループに使用可能な FPGA ハードウェアにデザインをコンパイルできます。この機能については、[第 5 章: ハードウェア協調シミュレーションの使用](#) を参照してください。

[Hardware Co-Simulation] コンパイル ターゲットを選択するには、次の図に示すように、System Generator トークンダイアログ ボックスの [Compilation] をクリックして [Hardware Co-Simulation] をクリックします。

図 147: ハードウェア協調シミュレーション



[Board] フィールドには、ハードウェア協調シミュレーション コンパイルを実行する際にターゲットとする開発ボードを指定します。ハードウェア協調シミュレーションに選択できるのは [Board] のみで、[Part] は選択できません。[Board] を選択すると、[Part] フィールドに [Board] で選択したボード上にあるザイリンクス デバイスの名前が自動的に表示されます。このパーツ名は変更できません。

JTAG ハードウェア協調シミュレーションは、すべてのザイリンクス開発ボードでサポートされています。[Hardware Co-Simulation] → [Point-to-Point Ethernet] は KC705 または VC707 ボードでのみサポートされます。

ハードウェア協調シミュレーション コンパイルの一部として生成される Simulink ライブラリ (<design_name>-hwcosim_lib.slx) は、[Target directory] フィールドに指定したディレクトリに含まれます。このライブラリ、およびライブラリに保存されたハードウェア協調シミュレーション ブロックについては、[ハードウェア協調シミュレーション ブロック](#)を参照してください。

IP カタログのコンパイル

System Generator はデフォルトの生成ターゲットとして [IP カタログ]のコンパイル タイプを使用します。

IP カタログ コンパイル ターゲットを選択すると、System Generator デザインを Vivado IP カタログに含めることができるように IP モジュールにパッケージできます。生成された IP は、別の Vivado ユーザー デザインにサブモジュールとして追加できます。

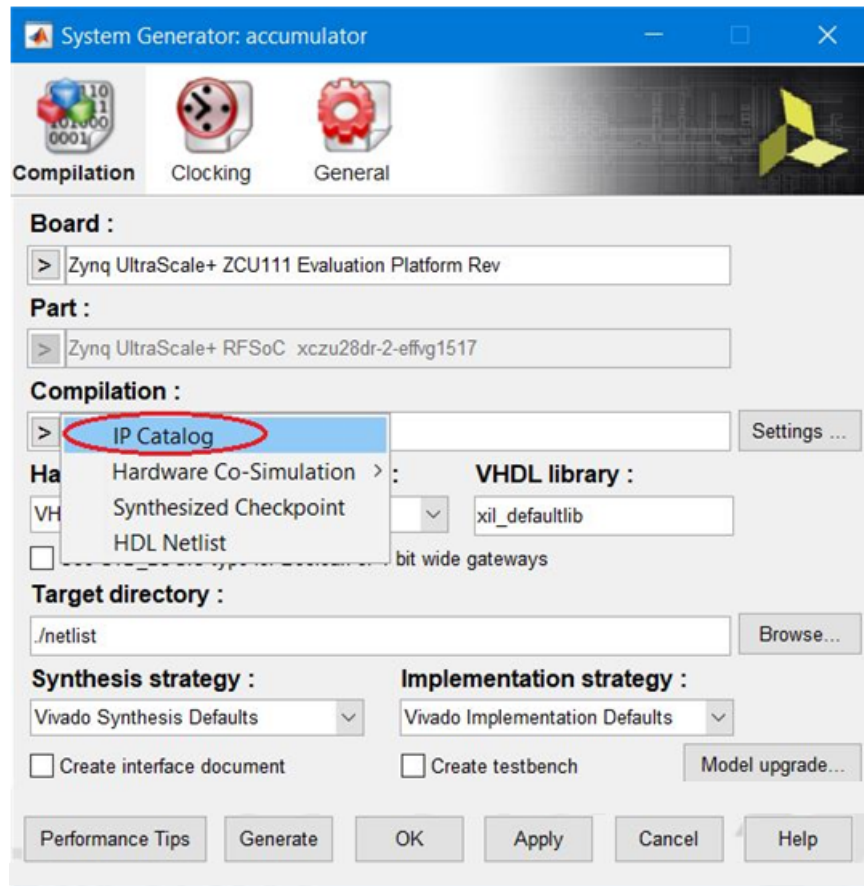
System Generator でまずブロック デザインに基づいて HDL ネットリストが生成されます。デザインに Vivado IP モジュールが含まれる場合は、必要な IP ファイルがすべて IP というサブフォルダーにコピーされます。最後に、RTL デザイン ファイルと Vivado IP デザイン ファイルがすべて ZIP ファイルに圧縮され、ip_catalog というサブフォルダーに保存されます。

IP カタログ フロー

System Generator デザインで System Generator トークンをダブルクリックします。

[Compilation] フィールドの [>] ボタンをクリックし、[IP Catalog]を選択します。

図 148: IP カタログ



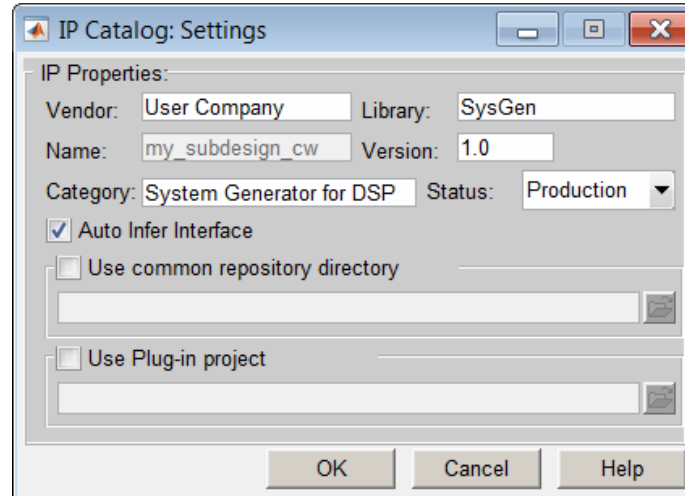
[Board] および [Part] フィールドでは、[IP Catalog]のコンパイルでターゲットにするボードまたはパーツを指定できます。[Board] を選択すると、[Part] フィールドに [Board] で選択したボード上にあるサイリンクス デバイスの名前が自動的に表示されます。このパーツ名は変更できません。

[IP Catalog] コンパイルは、Vivado ツールでサポートされるどのボードまたはパーツに対しても実行できます。Vivado のインストールに含まれるサイリンクス開発ボードだけでなく、パートナー ボードまたはカスタム ボードも指定できます (System Generator でのボード サポートの指定 を参照)。

[Target directory] フィールドで生成ファイルのディレクトリを指定します。

[Settings] ボタンをクリックすると、表示される次のダイアログ ボックスでモジュールに関する情報を入力できます。この情報が Vivado IP カタログに表示されます。

図 149: IP カタログの設定

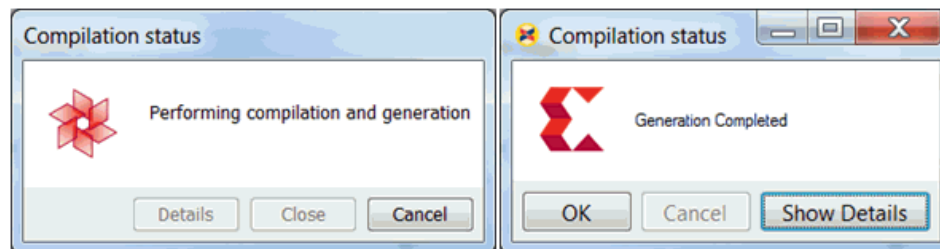


[Use common repository directory] フィールドには、共通リポジトリと呼ばれるディレクトリを指定します。IP カタログ コンパイルでは、作成された IP がこのリポジトリにコピーされます。Vivado ユーザーが Vivado プロジェクトの IP 設定でこのパスをユーザー リポジトリとして追加すると、System Generator ユーザーがこの共通リポジトリに保存した IP が Vivado にすべて自動的に読み込まれ、IP インテグレーターまたは RTL フローで使えるようになります。

[Use Plug-in project] フィールドには、System Generator にインポートされた IP インテグレーターのブロック図 (BD) を含む Vivado プロジェクトを指定します。このフィールドで Vivado プロジェクトを指定する必要がある手順の例は、[System Generator でのプラットフォーム ベースのアクセラレータの調整](#)を参照してください。

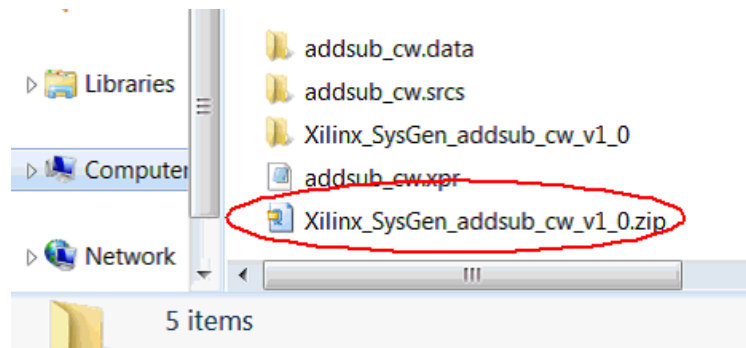
[Generate] ボタンをクリックすると、IP カタログ フローが開始します。次のような [Compilation status] ウィンドウが表示され、フローの進捗状況が示されます。IP カタログ フローが終了すると、[Generation Completed] と表示されます。[Show Details] をクリックすると、詳細な情報が表示されます。

図 150: [Compilation Status] ダイアログ ボックス



指定のターゲット ディレクトリを見ると、ip_catalog というフォルダーがあります。このフォルダーには、System Generator デザインから IP を作成するための必要なファイルがすべて含まれています。次の図に丸で囲まれている ZIP ファイルには、System Generator デザインを IP として Vivado IP カタログに含めるために必要なファイルがすべて含まれています。

図 151: ZIP ファイル



AXI4 インターフェイスの使用

[IP Catalog: Settings] ダイアログ ボックスで [Auto Infer Interface] オプションをオンにすると、デザインの Gateway In および Gateway Out ポートから AXI4 インターフェイスが自動推論されます。[Auto Infer Interface] オプションにより、信号がポート名に基づいて AXI4-Stream、AXI4-Lite、AXI4 インターフェイスにまとめられます。

[Auto Infer Interface] オプションをオンにすると、次の条件に基づいてインターフェイスが推論されます。

- Gateway In および Gateway Out ポート名の接尾語は AXI4 インターフェイス規格の信号名と完全に一致している必要があります。
- デザインに有効な AXI4 インターフェイスに必要な最低限の数の信号が含まれている必要があります。

たとえば、デザインに PortName_tdata および PortName_tvalid という 2 つの Gateway In ポートと、PortName_tready という Gateway Out ポートがある場合、[Auto Infer Interface] オプションでこれら 3 つのポートが自動推論され、PortName という名前の AXI4-Stream ポートにまとめられます。この例の場合、次のようになります。

- ポート名の接尾語は AXI4-Stream インターフェイスの信号 (TDATA、TREADY、および TVALID) と完全に一致しています。
- これらの 3 つの信号は、AXI4-Stream インターフェイスに最低限必要な信号です。

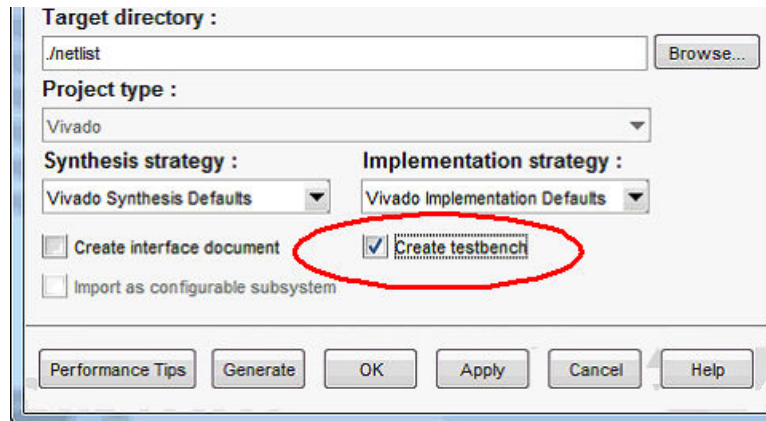
オプションの AXI4 側帯波信号 (AXI4-Stream 規格でオプションの TUSER 信号など) があり、その名前が同じ命名規則に従って付けられている場合は (PortName_tuser など)、同じ AXI4 インターフェイスにまとめられます。

AXI4 インターフェイスの信号名や、AXI4 インターフェイスに最低限必要な信号など、AXI4 インターフェイスの詳細は、『Vivado Design Suite: AXI リファレンス ガイド』 (UG1037: [英語版](#)、[日本語版](#)) を参照してください。

IP モジュールにテストベンチを含める

新しく生成した IP の機能を検証するには、テストベンチを含める必要があります。次の図に示すように [Create testbench] をオンにすると、[Generate] ボタンをクリックしたときにテストベンチが自動的に作成されます。

図 152: テストベンチの作成

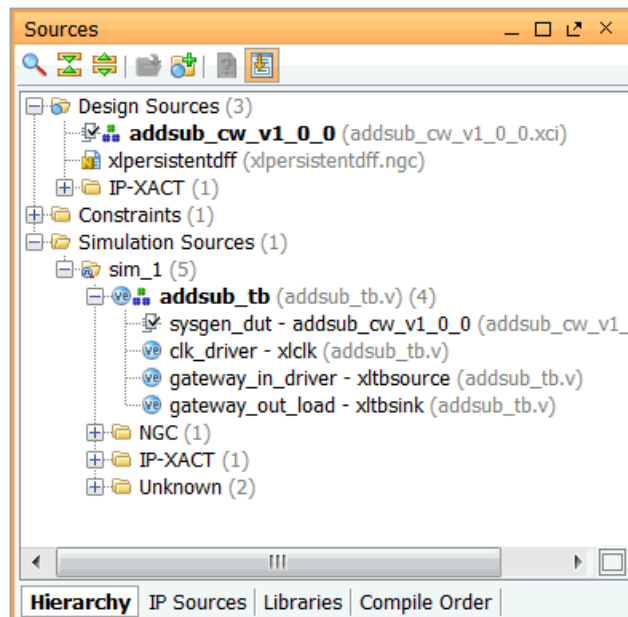


テストベンチを含めた場合、次の 3 つの手順をフローに追加すると、IP の機能を検証できます。

- 手順 1: 新しい IP を Vivado IP カタログに追加します。『Vivado Design Suite ユーザー ガイド: IP を使用した設計』(UG896) を参照してください。
- 手順 2: 新しい Vivado IDE プロジェクトを作成し、IP を最上位ソースとして追加します。
- 手順 3: シミュレーション、合成、インプリメンテーションを実行し、生成した IP の機能を検証します。

次の図に、新しく作成した IP を最上位ソースとして追加した Vivado IDE プロジェクトを示します。

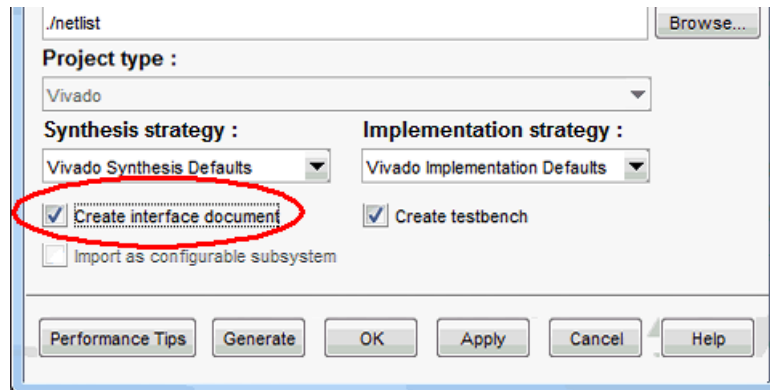
図 153: 新規 IP



IP モジュールへのインターフェイスに関する資料の追加

[Create interface document] をオンにして [Generate] をクリックすると、System Generator で IP のインターフェイスに関する資料が HTML 形式で生成され、その HTML ファイルが IP と共にパッケージされます。

図 154: [Create interface document] チェック ボックス



netlist フォルダの下に documentation フォルダが新しく作成されます。この新規 IP を Vivado IDE で右クリックして [Product guide] をクリックすると、IP のインターフェイス情報を含む HTML ファイルが開きます。

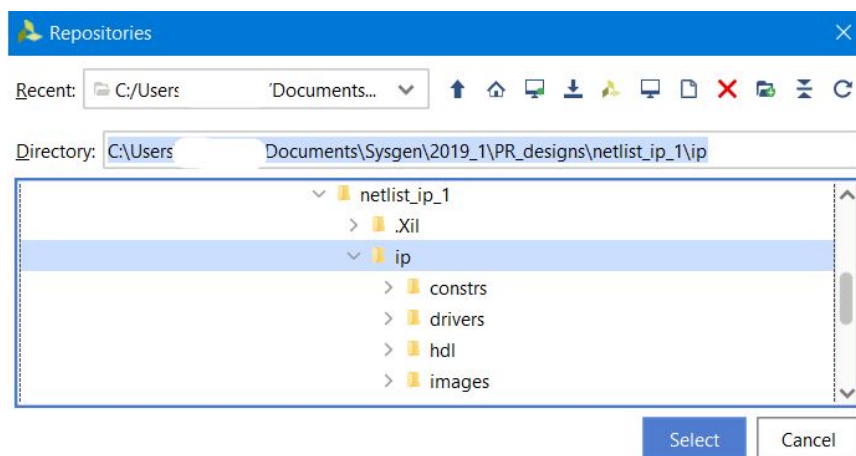
Vivado IP カタログへの生成された IP の追加

System Generator で生成した IP を使用するには、新しいプロジェクトを作成するか、IP を作成したときに System Generator で指定したのと同じデバイスをターゲットにしている既存のプロジェクトを開きます。

注記: IP にはこのプロジェクトのみアクセス可能です。新しいプロジェクトでこの IP を使用するたびに、同じ手順を実行する必要があります。

Flow Navigator で [Project Manager] → [IP Catalog] をクリックし、[IP Catalog] ウィンドウの空のエリアを右クリックします。[Add Repository] をクリックし、新規 IP を含むディレクトリを追加します。

図 155: IP カタログ



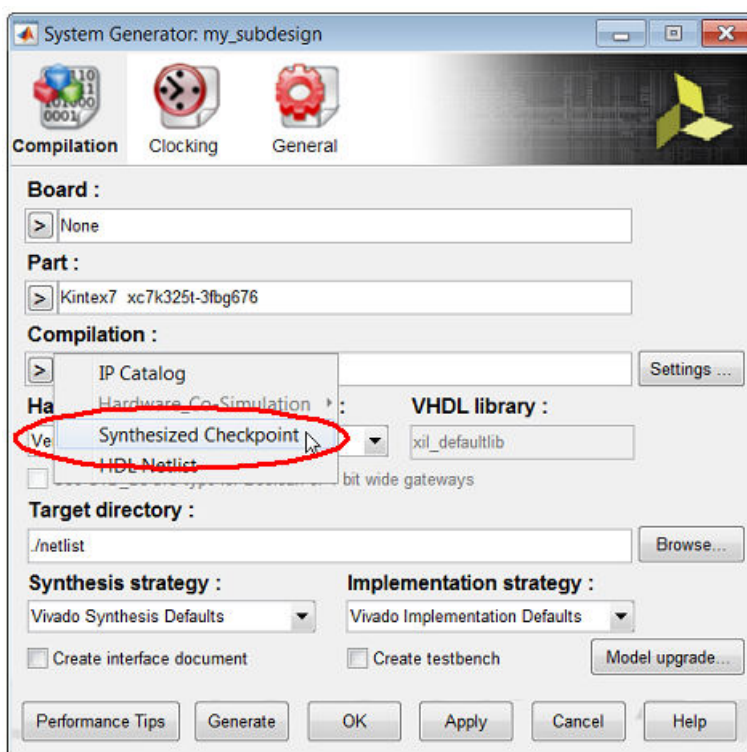
IP が IP カタログに追加されたら、IP カタログのその他の IP と同様、ほかのデザインに含めることができます。

合成済みチェックポイントのコンパイル

Vivado ツールでは、デザイン フローの主要な段階でデザインを保存および復元するためのメカニズムとして、デザイン チェックポイント ファイル (.dcp) が使用されます。チェックポイントは、フローの特定の地点におけるデザインのスナップショットです。合成済みチェックポイントとは、デザインが合成されてから、アウト オブ コンテキスト (OOC) モードで作成されたチェックポイント ファイルです。

[Synthesized Checkpoint] コンパイル ターゲットを選択すると (次の図を参照)、<design_name>.dcp という名前の合成済みチェックポイント ターゲット ファイルが作成され、[Target directory] で指定したターゲット ディレクトリに保存されます。この <design_name>.dcp ファイルは、この後どの Vivado IDE プロジェクトでも使用できます。

図 156: 合成済みチェックポイント



[Board] および [Part] フィールドでは、合成済みチェックポイントのコンパイルでターゲットにするボードまたはパーツを指定できます。[Board] を選択すると、[Part] フィールドに [Board] で選択したボード上にあるサイリンクス デバイスの名前が自動的に表示されます。このパーツ名は変更できません。

合成済みチェックポイントのコンパイルは、Vivado ツールでサポートされるどのボードまたはパーツに対しても実行できます。Vivado のインストールに含まれるサイリンクス開発ボードだけでなく、パートナー ボードまたはカスタム ボードも指定できます (System Generator でのボード サポートの指定 を参照)。

カスタム コンパイル ターゲットの作成

System Generator には、ユーザーが独自のコンパイル ターゲットを作成するカスタム コンパイル インフラストラクチャが含まれています。System Generator デザインから HDL を生成できるだけでなく、HDL 生成前後両方の段階を自動化するコンパイル ターゲット プラグインも作成できます。カスタム コンパイル ターゲットの作成に関する詳細は、[第 9 章: カスタム コンパイル ターゲットの作成](#) を参照してください。

カスタム コンパイル ターゲットの作成

System Generator には、ユーザーが独自のコンパイル ターゲットを作成できるカスタム コンパイル インフラストラクチャが含まれています。System Generator デザインから HDL を生成できるだけでなく、Vivado® 統合設計環境 (IDE) プロジェクトを作成した前後の手順を自動化するコンパイル ターゲット プラグインも作成できます。カスタム コンパイル ターゲットを作成するには、MATLAB® 環境でオブジェクト志向プログラミングのコンセプトに精通している必要があります。

xilinx_compilation ベース クラス

カスタム コンパイル インフラストラクチャには `xilinx_compilation` という名前のベース クラスがあります。このクラスからサブクラスを作成し、そのプロパティを使用して、メンバー関数を上書きして、ユーザー機能をインプリメントします。

図 157: ベース クラス



新しいコンパイル ターゲットの作成

次に、新しいコンパイル ターゲットを作成する一般的な手順と例を示します。

ヘルパー関数の実行

次のヘルパー関数を実行し、新しいカスタム コンパイル ターゲットを作成します。

```
xilinx.environment.addCompilationTarget(target_name, directory_name)
```

たとえば、次のコマンドを例にとってみます。

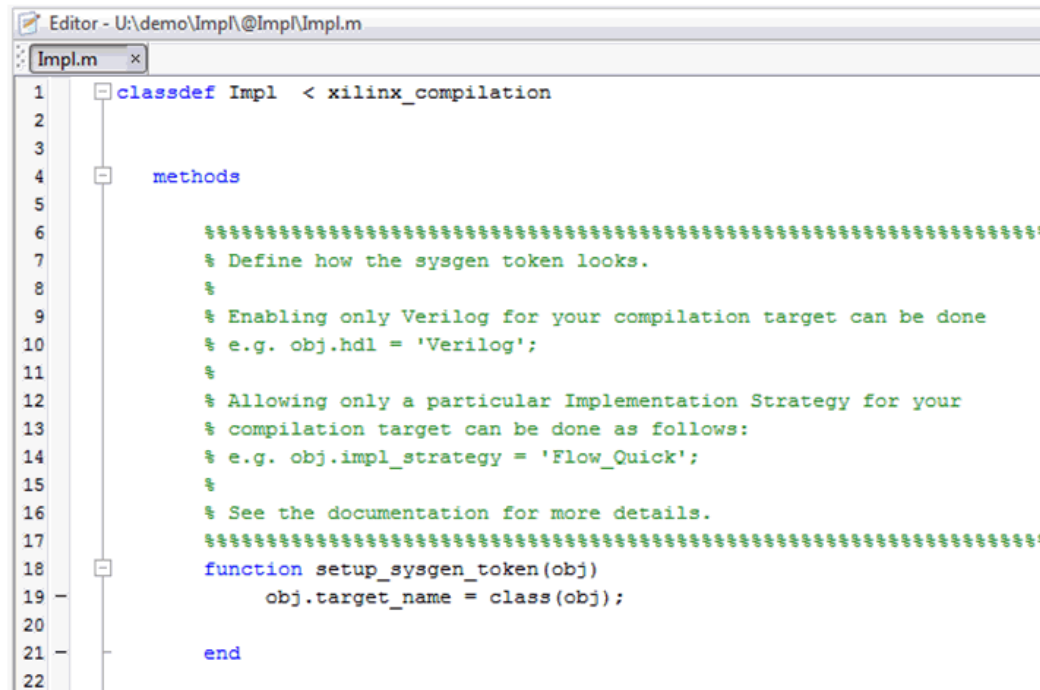
```
xilinx.environment.addCompilationTarget('Impl', 'U:\demo')
```

図 158: ヘルパー関数



先ほど示したように、MATLAB のコマンド ウィンドウにこのコマンドを入力すると、次のようになります。

1. U:\demo に Impl/@Impl という名前のフォルダーが作成されます。
2. このフォルダーに、Impl というテンプレート クラス ファイル (Impl.m) が作成されます。これは、ベース クラス `xilinx_compilation` から派生したものです。この時点で、ファイルに何も変更がなければ、新しく作成された Impl コンパイル ターゲットは、[HDL Netlist] コンパイル ターゲットと動作が同じになります。次の図に、Impl.m ファイルの内容を示します。



3. この新しいクラス `Impl` が MATLAB で検出されるよう、ヘルパー関数で MATLAB パスに `U:\demo\Impl` が追加されます。

注記: `target_name` にはスペースを含めることはできません。クラスが作成された後、クラスの `target_name` プロパティにスペースを追加することはできません。

コンパイル ターゲットの変更

コンパイル ターゲットのクラス ファイルを変更する場合、次のヘルパー関数を呼び出す必要があります。このヘルパー関数は、System Generator で新しいクラス定義が検出されるようにするためのものです。

```
>> xilinx.environment.rehashCompilationTarget
```

既存のコンパイル ターゲットの追加

カスタム コンパイル ターゲットのフォルダーを含むパスを追加する必要があります。次に示すように、MATLAB® で提供されている `addpath` 機能を使用します。

```
>>addpath('U:\demo\Impl');
```

`addpath` を使用する場合は、相対パスではなく、絶対パスを指定する必要があります。

カスタム コンパイル ターゲットの保存

カスタム コンパイル ターゲットを保存するには、MATLAB® の `savepath` 機能を使用します。保存するには、MATLAB のインストール エリアへの書き込み権が必要な場合があります。

カスタム コンパイル ターゲットの削除

カスタム コンパイル ターゲットを削除するには、MATLAB® の検索パスからターゲットへのパスを削除します。

ベース クラス プロパティおよび API

`xilinx_compilation` ベース クラスは、次のディレクトリにあります。

```
<Vivado Install Path>/scripts/sysgen/matlab/@xilinx_compilation
```

System Generator トークン関連プロパティおよび API

`setup_sysgen_token()`

この関数は、カスタム コンパイル インフラストラクチャにより System Generator トークン情報を取得するために呼び出されます。カスタム ターゲットを選択したときのトークンのデフォルト表示を設定するには、System Generator トークンに関連する次の関数を使用します。フィールド、デフォルト値、フィールドのオン/オフは、次の System Generator トークン API 関数で設定できます。

`add_part(family, device, speed, package, temperature)`

たとえば、「`add_part('Kintex7', 'xc7k325t', '-1', 'fbg676', '')`」のように使用します。パーツ関連の API が使用されていない場合は、エンド ユーザーはリストから任意のデバイスを選択できます。

string target_name

setup_sysgen_token() 関数で設定する必要のある必須フィールドです。

string hdl

デフォルト値は空の文字列です。有効なオプションは `verilog` または `vhdl` です。このフィールドの値が設定されると、このフィールドはディスエーブルになりユーザーは選択できなくなります。

string synth_strategy

デフォルト値は空の文字列です。このフィールドの値が設定されると、このフィールドはディスエーブルになりユーザーは選択できなくなります。この API が使用されている場合、指定のストラテジが存在することを確認する必要があります。ストラテジがない場合は、エラーになります。

string impl_strategy

デフォルト値は空の文字列です。このフィールドの値が設定されると、このフィールドはディスエーブルになりユーザーは選択できなくなります。この API が使用されている場合、指定のストラテジが存在することを確認する必要があります。ストラテジがない場合は、エラーになります。

string create_tb

デフォルト値は空の文字列です。有効な値は `on` または `off` です。このフィールドの値が設定されると、このフィールドはディスエーブルになりユーザーは選択できなくなります。

string create_iface_doc

デフォルト値は空の文字列です。有効な値は `on` または `off` です。このフィールドの値が設定されると、このフィールドはディスエーブルになりユーザーは選択できなくなります。

Vivado プロジェクト関連のプロパティ

top_level_module

このプロパティを使用して最上位の名前を設定できます。このパラメーターには MATLAB® の文字列を指定します。

Vivado IDE プロジェクト生成関連の関数

pre_project_creation(design_info)

この関数は、Vivado® IDE プロジェクト作成の最後に呼び出す必要があります。System Generator インフラストラクチャでプロジェクトを作成する前に、Vivado® IDE プロジェクトに追加する必要があるファイルと、実行する必要がある追加 Tcl コマンドを指定しておく必要があります。System Generator デザインの最上位ポート インターフェイスによって、プロジェクトにいくつかのファイルを追加する必要がある場合があります。このため、ポート インターフェイスを記述するストラクチャが `design_info` という関数に渡されます。`design_info` については後ほど詳しく説明します。

post_project_creation(design_info)

Vivado IDE プロジェクト作成の最後に、この関数を呼び出す必要があります。プロジェクト生成スクリプトが実行された後、最後に呼び出されるのがこの関数です。これは、エラー処理、レポート生成、Vivado IDE プロジェクトを開く場合などに便利な関数です。design_info と呼ばれるこの関数に、ポート インターフェイスを記述するストラクチャが渡されます。design_info については後ほど詳しく説明します。

add_tcl_command(string)

この関数は、追加 Tcl コマンドを文字列として追加します。これらの Tcl コマンドは、Vivado IDE プロジェクトが作成された後に発行されます。このコマンドを使用して、プロジェクトが作成された後にビットストリームを作成します。また、この Tcl コマンドは特定の Tcl ファイルを呼び出すのにも使用できます。コマンドは受信された順に実行されます。

add_file(string)

この関数は、Vivado IDE プロジェクトにユーザー定義のファイルを追加します。この アプリケーション プログラミング インターフェイス (API) 関数は、Vivado IDE プロジェクトに XDC 制約ファイルを追加するためにも使用できます。add_file が呼び出される順序は階層的である点に注意してください。最上位ファイルは最後に追加する必要があります。

run_synthesis()

この関数は、Vivado IDE プロジェクトで合成を実行します。

run_implementation()

この関数は、Vivado IDE プロジェクトでインプリメンテーションを実行します。

generate_bitstream()

この関数は、Vivado IDE プロジェクトでビットストリームを生成します。

デザイン情報

design_info は MATLAB® の構造体で、その内容は次のとおりです。

図 159: デザイン情報

design_info				
design_info.ports				
design_info.ports.gateway_in <1x1 struct>				
Field	Value	Min	Max	
ArithmeticType	'xIsSigned'			
BinaryPoint	14	14	14	
DatFile	'adder_gateway_i...			
Direction	'in'			
IconText	'Gateway In'			
IsClock	0	0	0	
Name	'gateway_in'			
Period	1	1	1	
Type	'Fix_16_14'			
Width	16	16	16	

design_info				
design_info.ports				
design_info <1x1 struct>				
Field	Value	Min	Max	
ports	<1x1 struct>			
sim_time	10	10	10	
target_dir	'/group/dspuser...			
testbench	'on'			
top_level	'adder'			

カスタム コンパイル ターゲットの作成例

さまざまタイプのカスタム ターゲットを作成する方法を例を示しながら説明します。

例 1: インプリメンテーション ターゲットの作成

1. System Generator モデルを開き、その後 System Generator トークンを開きます。これで使用可能なコンパイル ターゲットすべてがトークンに読み込まれます。
2. MATLAB® のコマンド ウィンドウで要件に従ってパスを変更し、次のコマンドを入力します。

```
xilinx.environment.addCompilationTarget('Impl', 'U:\demo')
```

テンプレートから作成された編集可能なクラスが開きます。

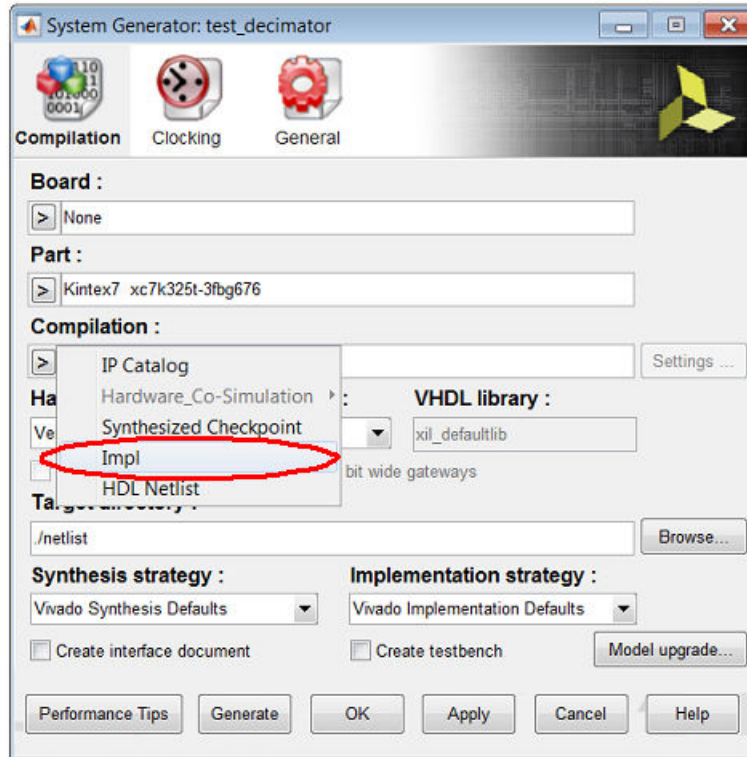
3. MATLAB のコマンド ウィンドウに次のコマンドを入力します。

```
xilinx.environment.rehashCompilationTarget
```

これで、System Generator トークンにより新しいコンパイル ターゲットが選択されます。

4. System Generator トークンをいったん閉じて開き直します。次の図に示すように、トークンに [Impl] というコンパイル ターゲットが表示されます。

図 160: [Impl] の選択



5. この時点では、System Generator トークンで [Impl] を選択しても、カスタマイズされた操作は実行されません。これは HDL ネットリスト コンパイル ターゲットと同等です。
6. MATLAB エディターで `U:\demo\Impl\@Impl\Impl.m` を開きます。
7. `setup_sysgen_token()` 関数の記述を要件に従って変更します。この方法を使用すると、ユーザー定義のカスタム コンパイルを選択したときのフィールドのオン/オフなど、System Generator トークンの表示を制御できます。

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define how the sysgen token looks.
%
% Enabling only Verilog for your compilation target can be done
% e.g. obj.hdl = 'Verilog';
%
% Allowing only a particular Implementation Strategy for your
% compilation target can be done as follows:
% e.g. obj.impl_strategy = 'Flow_Quick';
%
% See the documentation for more details.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function setup_sysgen_token(obj)
    obj.target_name = class(obj);
    obj.hdl = 'Verilog';
    obj.impl_strategy = 'Flow_Quick';
end

```

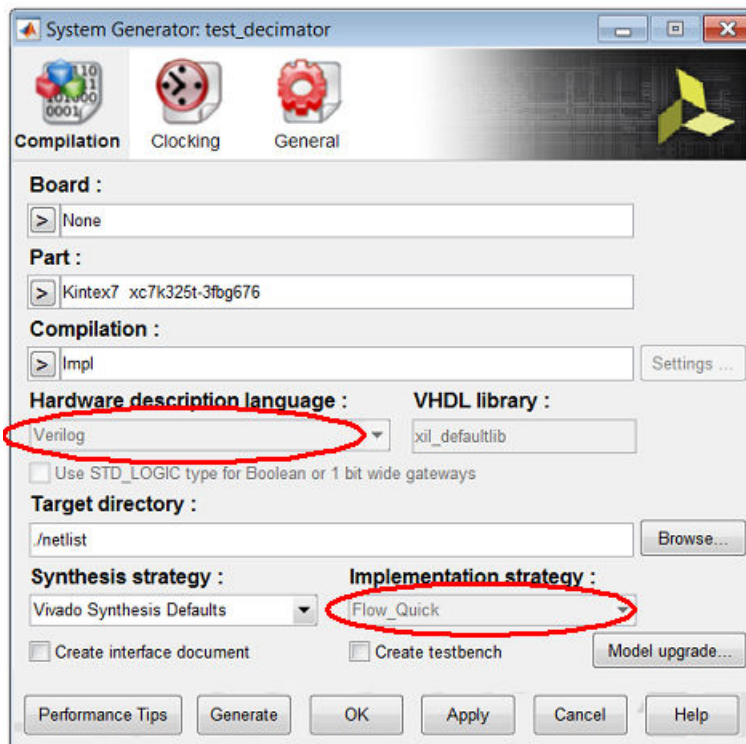
8. MATLAB コマンド ウィンドウに次のコマンドを入力します。

```
xilinx.environment.rehashCompilationTarget
```

これで、[Impl] のアップデートされたクラス定義が使用されるようになります。

9. System Generator トークンをいったん閉じて開き直します。[Compilation] のリストから [Impl] を選択します。
10. System Generator トークンの表示は次のようになります。

図 161: [Verilog] および [Flow_Quick] の選択



11. [Hardware description language] フィールドおよび [Implementation strategy] フィールドが [Impl] クラスに設定した値に固定され、ユーザーが変更できないようになっていることを確認します。
12. ユーザー指定フィールドおよび実行すべき追加の Tcl コマンドは、Vivado® IDE プロジェクトを作成する前にわかっています。pre_project_creation() 関数の記述を次のように変更します。

```
#####
% Define how the Project should be generated. Adding tcl commands,
% files etc. should be done here.
%
% e.g. obj.add_tcl_command('launch_runs synth_1');
% e.g. obj.add_file('C:\work\myconstraints.xdc');
% e.g. obj.run_implementation()
%
% design_info is the struct that contains the information about the
% design and its interface. See documentation for more details
#####
function pre_project_creation(obj, design_info)
    obj.add_tcl_command('launch_runs synth_1');
    obj.add_tcl_command('wait_on_run synth_1');
    obj.run_implementation();
end
```

13. MATLAB のコマンド ウィンドウに次のコマンドを入力します。

```
xilinx.environment.rehashCompilationTarget
```

これで、[Impl] のアップデートされたクラス定義が使用されるようになります。

14. System Generator トークンをいったん閉じて開き直します。[Compilation] のリストから [Impl] を選択します。
15. [Generate] をクリックします。プロセスが完了したら、Vivado IDE プロジェクトを開いてインプリメンテーション結果を確認できます。

例 2: ビットストリーム ターゲットの作成

1. System Generator デザインを開きます。
2. MATLAB のコマンド ウィンドウで、ユーザー要件に従い、最初の例と同様にパスを変更し、それから次のコマンドを入力します。

```
xilinx.environment.addCompilationTarget('Bitstream', '.')
```

ユーザーが編集するテンプレート クラスが開きます。最後のフィールドは、board.xml ファイルを含むディレクトリに対応しています。

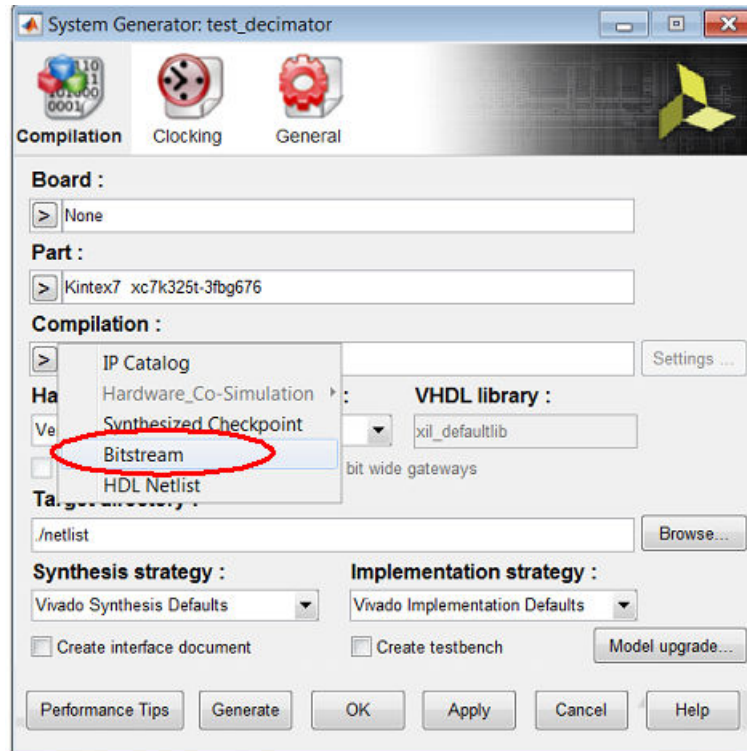
3. MATLAB のコマンド ウィンドウに次のコマンドを入力します。

```
xilinx.environment.rehashCompilationTarget
```

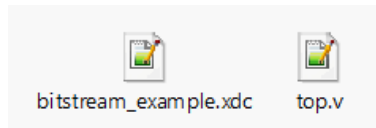
これで、System Generator トークンにより新しいコンパイル ターゲットが選択されます。

4. System Generator トークンをいったん閉じて開き直します。
5. 次の図に示すように、System Generator トークンに [Bitstream] というコンパイル ターゲットが表示されるようになります。

図 162: ビットストリーム



6. ./Bitstream/@Bitstream/Bitstream.m に作成した Bitstream.m を開きます。
7. 次の 2 つのファイルをダウンロードします。



8. pre_project_creation() 内で、次を実行するための行を追加します。
 - a. ボードを KC705 ボードに設定。
 - b. KC705 の差動クロック ポートを使用するため、新しい最上位ファイル (top.v) を追加。
 - c. クロック、DIP、LED ポートのロケーション制約を指定するため、新しい XDC ファイルを追加。
 - d. 新しく追加したモジュール top を最上位に設定。
 - e. 合成を実行。
 - f. インプリメンテーションを実行。
 - g. ビットストリームを生成。

使用のコンピューターのディレクトリにファイルを保存した後、add_file API にファイルへの完全パスを指定する必要があります。

```
add_tcl_command(obj, 'set_property board xilinx.com:kintex7:kc705:1.1
[current_project]');
add_file(obj,
'/group/dspusers-xsj/umangp/rel/2013.4/cust_comp_test/
bitstream_example.xdc');
add_file(obj, '/group/dspusers-xsj/umangp/rel/2013.4/cust_comp_test/
```

```
top.v');
obj.top_level_module = 'top';
run_synthesis(obj);
run_implementation(obj);
generate_bitstream(obj);
```

9. MATLAB のコマンド ウィンドウに次のコマンドを入力します。

```
xilinx.environment.rehashCompilationTarget
```

これで、System Generator トークンにより新しいコンパイル ターゲットが選択されます。

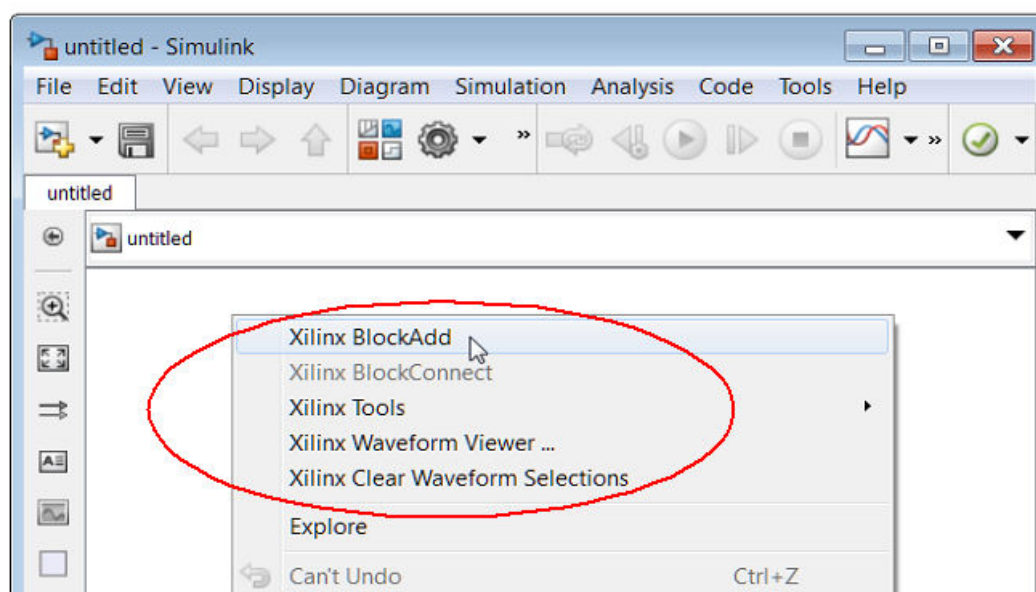
10. System Generator トークンをいったん閉じて開き直します。
11. [Bitstream] コンパイル ターゲットを選択します。
12. [Generate] をクリックします。
13. 生成が完了したら、次のディレクトリに BIT ファイルが保存されます。

```
./<Target_directory>/Bitstream/bitstream_example.runs/impl_1/top.bit
```

System Generator の GUI ユーティリティ

System Generator デザインをすばやく作成および解析できるよう、ザイリンクスでは Simulink® モデルの右クリックメニューにグラフィック コマンドを追加しました。これらのコマンドには、次の図に示すように、Simulink モデルのキャンバスを右クリックし、適切なザイリンクス コマンドをクリックするとアクセスできます。

図 163: ザイリンクス コマンド



次に、GUI に追加されたザイリンクス コマンドの説明を示します。

Xilinx BlockAdd	Simulink モデルにザイリンクス ブロック (および一部の Simulink ブロック) をすばやく追加します。
[Xilinx Tools] → [Save as blockAdd default]	ブロックをあらかじめ設定しておき、そのブロックの複数のコピーを BlockAdd 機能を使用して追加できます。
[Xilinx BlockConnect]	Simulink モデル内でブロックをすばやく接続します。
[Xilinx Tools] → [Terminate]	未接続の出力ポートに Simulink 終端ブロックを、または未接続の入力ポートにザイリンクス Constant ブロックをすばやく追加します。
ザイリンクス波形ビューアー	ザイリンクス波形ビューアーは、System Generator デザインで選択された信号の波形図を表示します。波形は、Simulink シミュレーションを実行した後に、波形ビューアーで表示できます。ザイリンクス ブロックセットのブロックの入力および出力は、波形ビューアーに表示されます。

[\[Xilinx Clear Waveform Selections\]](#)

波形ビューアーで現在表示されている波形すべてを削除し、波形ビューアーを閉じます。

Xilinx BlockAdd

Simulink モデルにザイリンクス ブロック (および一部の Simulink® ブロック) をすばやく追加します。

起動方法

方法 1:

Simulink キャンバスを右クリックして [Xilinx BlockAdd] をクリックします。

方法 2:

Ctrl + 1 キーを押します。

方法 3:

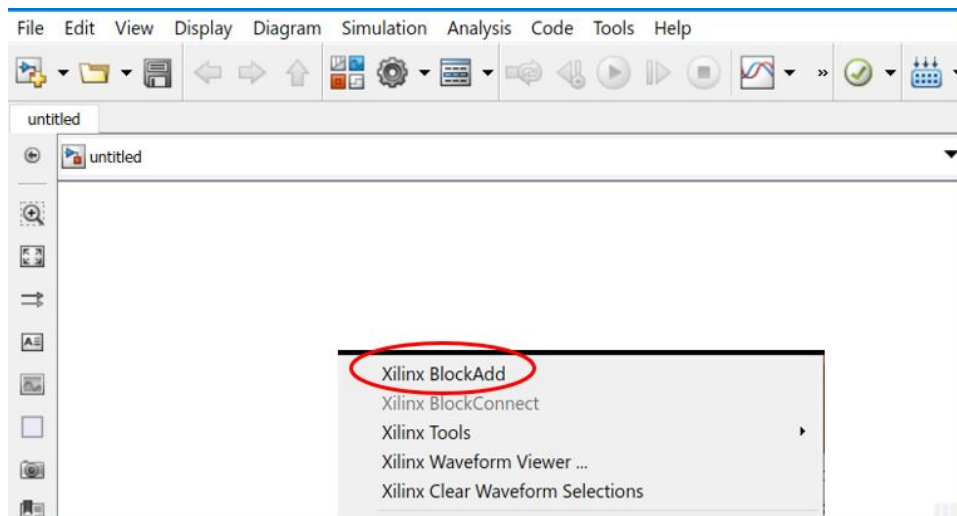
Simulink モデルのプルダウン メニューから次をクリックします。

[Tools] → [Xilinx] → [BlockAdd]

使用方法

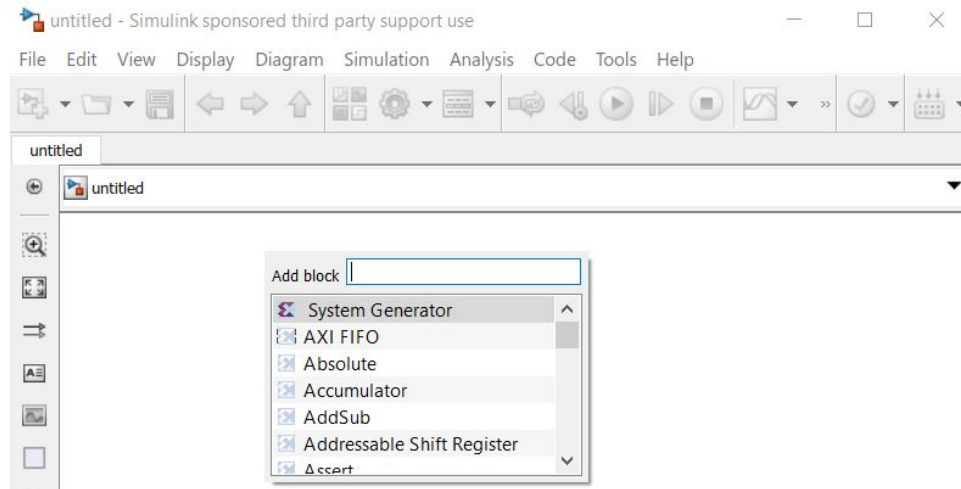
1. Simulink キャンバスを右クリックして [Xilinx BlockAdd] をクリックします。

図 164: [Xilinx BlockAdd] コマンド



2. [AddSub] をダブルクリックします。

図 165: AddSub



3. 同じブロックのコピーを複数追加するには、ブロックを 1 つ追加した後、そのブロックを選択して [Ctrl + C] キーを押し、[Ctrl + V]、[Ctrl + V] のように必要な回数だけ押します。
4. [Add block] ウィンドウを閉じるには、[Esc] キーを押します。

[Xilinx Tools] → [Save as blockAdd default]

ブロックをあらかじめ設定しておき、そのブロックの複数のコピーを BlockAdd 機能を使用して追加できます。

使用方法

モデルに [Boolean] 型の Gateway In ブロックを複数追加する必要があるとします。

1. モデルに 1 つの Gateway In ブロックを追加します。
2. [Gateway In] ブロックをダブルクリックし、[Output type] を [Boolean] に変更して [OK] をクリックします。
3. 変更した [Gateway In] ブロックを右クリックし、[Xilinx Tools]→[Save as blockAdd default] をクリックします。
4. これにより、次回から BlockAdd 機能を使用してモデルに Gateway In ブロックを追加したときに、ブロックの出力型が常にブール型になります。

ブロック デフォルトの復元方法

1. デフォルトを変更したブロックを選択します。
2. 右クリックし、[Xilinx Tools]→[Clear blockAdd defaults] をクリックします。

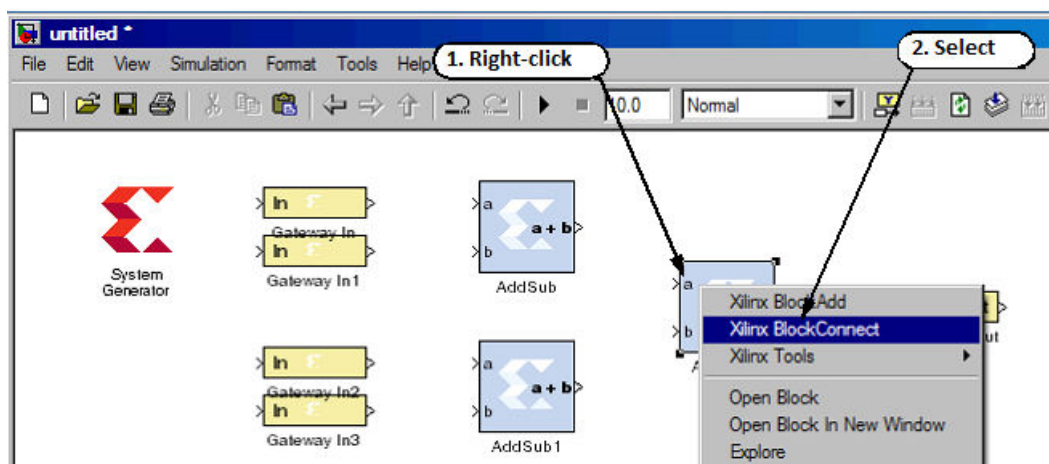
[Xilinx BlockConnect]

Simulink® モデル内でブロックをすばやく接続します。

単純な接続

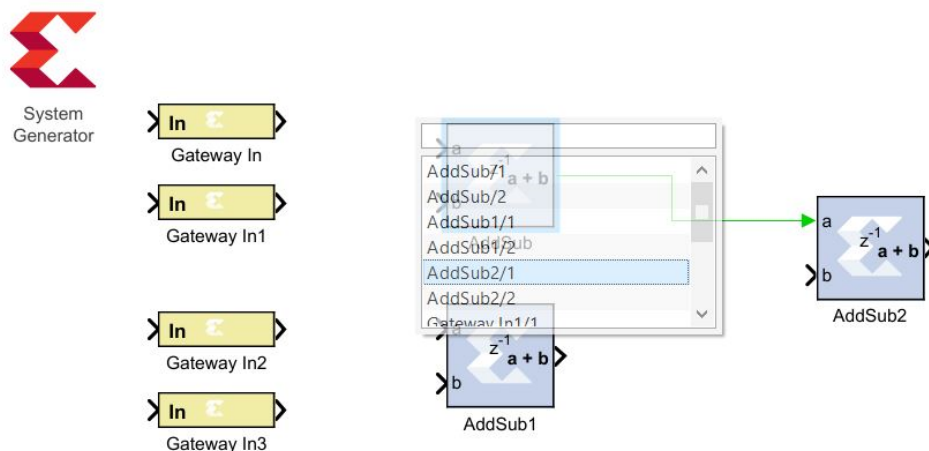
- 次に示すように、ブロックの接続されていないポートを右クリックして [Xilinx BlockConnect] をクリックし、そのポートに可能な接続のリストを表示します。[Xilinx Block Connect] コマンドでは、一度に 1 つのポートの接続のみを表示可能です。

図 166: [Xilinx BlockConnect]



- BlockConnect により、推奨される最も近い接続が緑の線で示されます。その接続を使用するには、表内で接続をダブルクリックします。接続されると、線の色が黒に変わります。その接続を使用しない場合は、表内で別の接続をクリックし、新しく推奨された緑の接続線が正しいかどうか確認します。

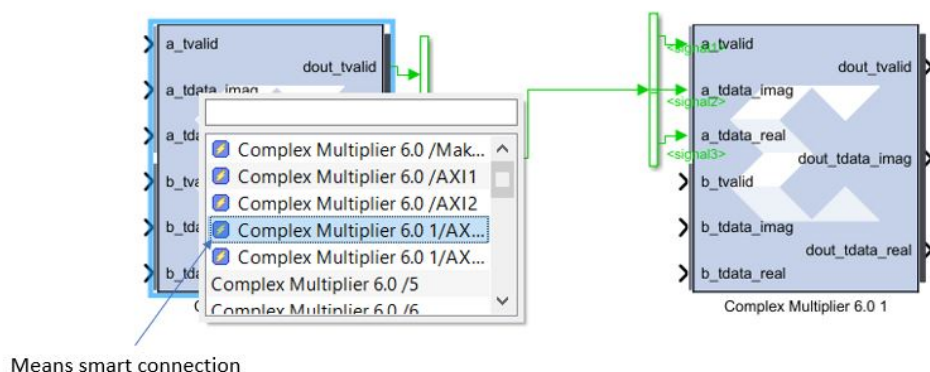
図 167: ブロックの接続



スマート接続

次の図に示す稲妻アイコンは、スマート接続を示します。スマート接続には、接続の管理に役立つ機能が組み込まれています。たとえば、AXI インターフェイスを含むブロックを右クリックすると、AXI 信号をバスにまとめたり、バスを個々の信号に分割したり、同じ数の AXI 接続を持つほかのポートに接続したりできます。

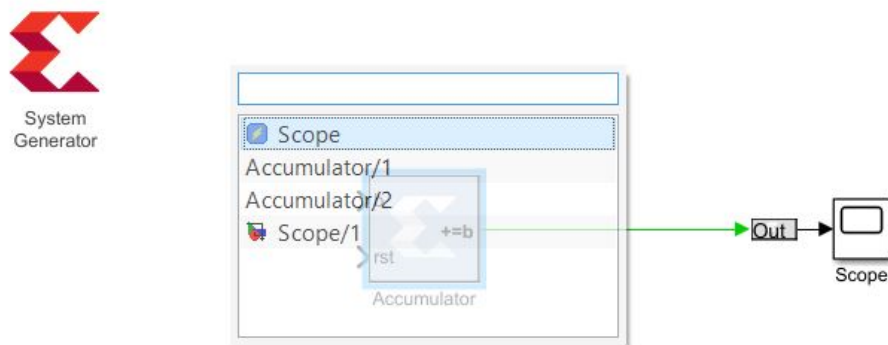
図 168: スマート接続



ポートのデータ型のチェックは実行されず、ポート数が同じの AXI ポートであれば、どれにでも接続できます。

次の別のスマート接続の例では、[Accumulator] ブロック出力を右クリックして [BlockConnect] をクリックし、[Scope] をダブルクリックして Scope ブロックへのスマート接続を作成しています。Gateway Out ブロックは自動的に追加されます。

図 169: Scope ブロックの接続



この Scope ブロックへ 2 つ目の接続があると、2 つ目のポートが Scope ブロックに自動的に追加されます。その駆動信号名が Scope を駆動する信号の名前にも使用されます。

[Xilinx Tools] → [Terminate]

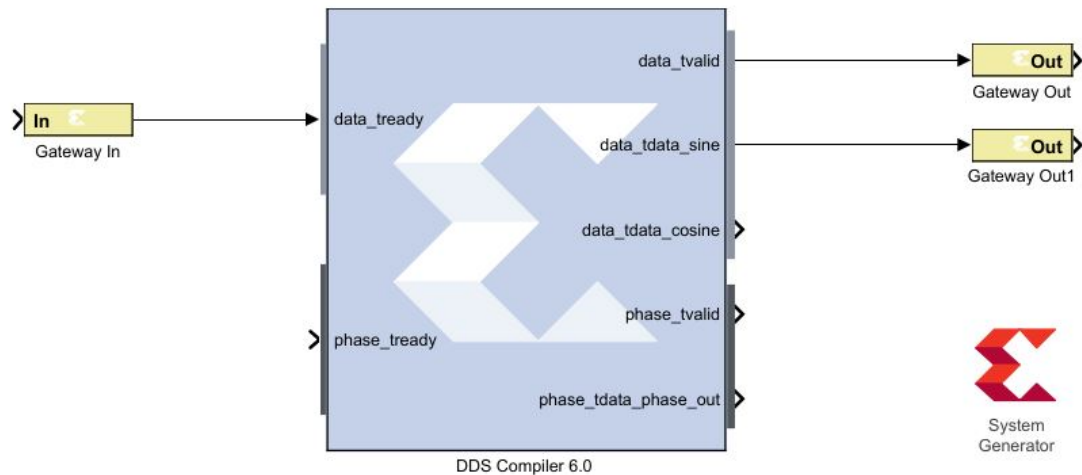
未接続の出力ポートに Simulink® 終端ブロックを、または未接続の入力ポートにザイリンクス Constant ブロックをすばやく追加します。

使用方法

未接続出力の終端

未接続の入力および出力ポートがある次のようなモデルがあるとします。

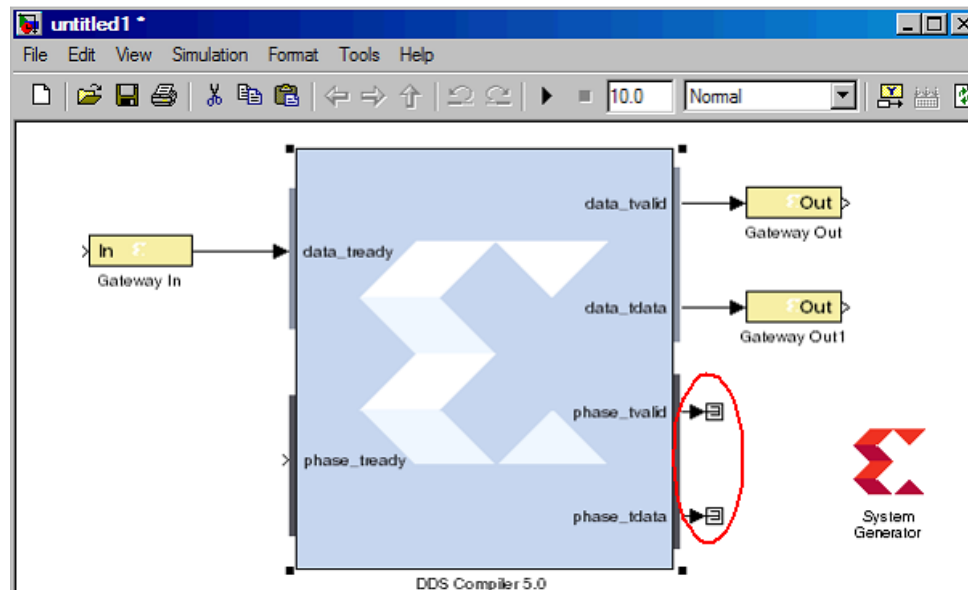
図 170: 未接続の入力および出力ポートがあるモデル



この場合、DDS Compiler 5.0 ブロックを右クリックし、[Xilinx Tools]→[Terminate]→[Outputs] をクリックします。

次の図に、このコマンドを実行した後の終端処理された出力を示します。

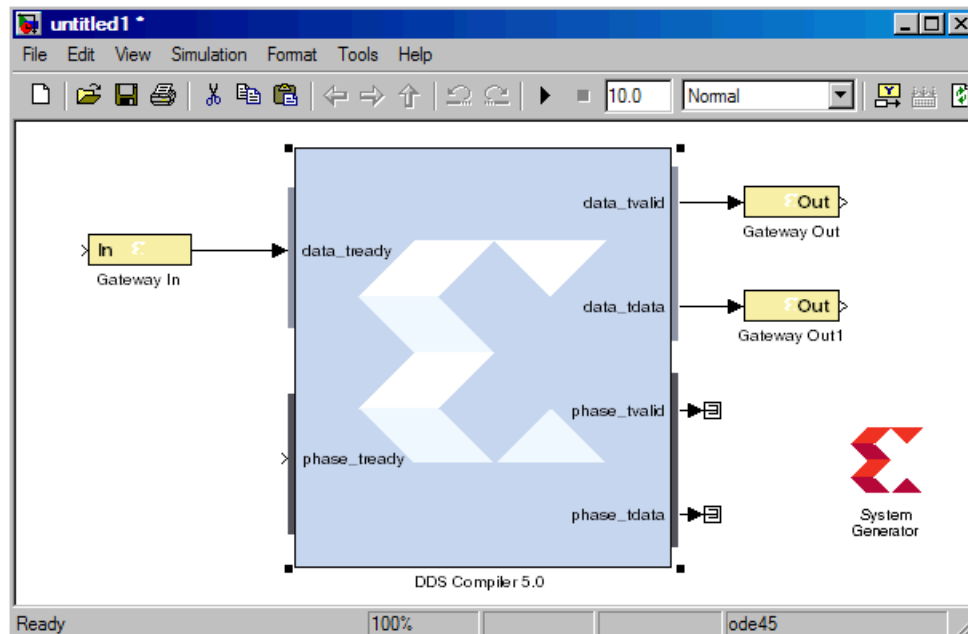
図 171: 終端された出力ポート



未接続入力端子の終端

未接続の入力ポートがある次のようなモデルがあるとします。

図 172: 未接続の入力ポートがあるモデル

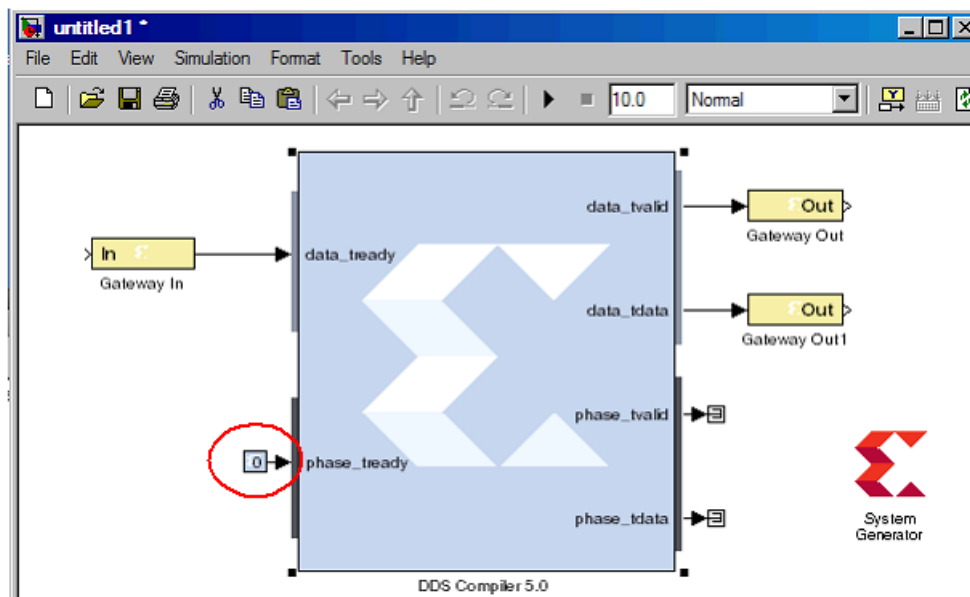


この場合、DDS Compiler 5.0 ブロックを右クリックし、次の順にクリックします。

[Xilinx Tools] → [Terminate] → [Inputs]

次の図に、このコマンドを実行した後の終端処理された入力を示します。

図 173: 終端された入力ポート



入力ポート データ型要件の確認

System Generator では、未接続の入力ポートはそれぞれザイリンクス Constant ブロックに接続されます。新しい Constant ブロックは、次のデフォルト値に設定されます。

[Type]: [Signed] (2 の補数)

[Constant value]: 0

[Number of bits]: 16

[Binary point]: 14

この終端ツールでは、入力ポートのデータ型はチェックされません。未接続のポートに、ブール型などの別のデータ型が必要な場合は、Constant ブロックをダブルクリックして [Output Type] を [Boolean] に変更する必要があります。

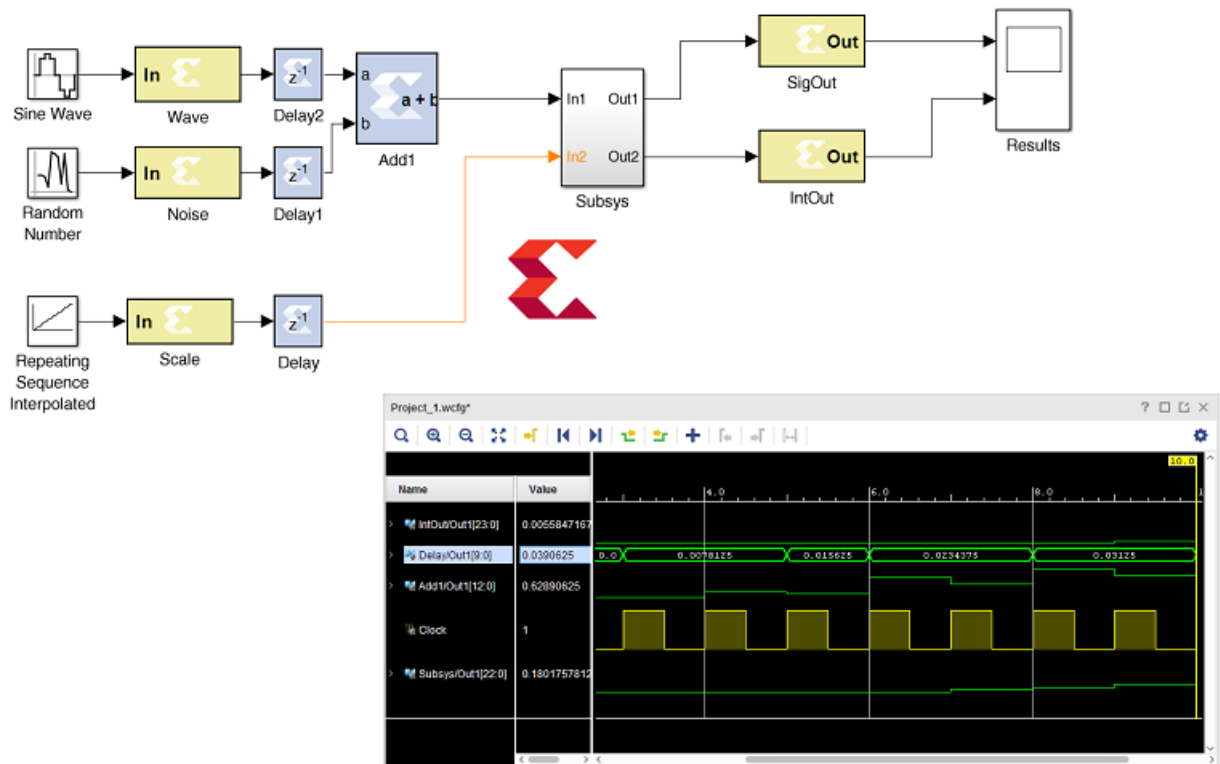
データ型の不一致がないかをチェックするには、Simulink® モデル キャンバスをクリックして、Ctrl + D キーを押します。データ型の不一致があれば、System Generator によりすべてレポートされます。

ザイリンクス波形ビューアー

ザイリンクス波形ビューアーは、System Generator デザインで選択された信号の波形図を表示します。波形は、Simulink シミュレーションを実行した後に、波形ビューアーで表示できます。ザイリンクス ブロックセットのブロックの入力および出力は、波形ビューアーに表示されます。

波形ビューアーで監視する信号をデザインで選択します。デザインの開発およびトラブルシュート時、モデルをシミュレーションするたびに、監視している信号の波形は波形ビューアーでアップデートされます。

図 174: 波形ビューアーでのサンプル デザイン



System Generator で使用するザイリンクス波形ビューアーは、Vivado® ツールセットのほかのツールでも使用されます。波形ビューアーは、Vivado シミュレータでデザインを解析してコードをデバッグするときや、インシステム デバッグで ILA (Integrated Logic Analyzer) でキャプチャされたデータを表示するときにも使用されます。

デザインの開発およびトラブルシュートに波形ビューアーを使用する方法は、『Vivado Design Suite ユーザー ガイド: ロジック シミュレーション』 (UG900) の[このセクション](#)を参照してください。

波形ビューアーのファイル

Simulink モデルに対して波形ビューアーを初めて開くと、System Generator により Simulink モデルを含むディレクトリに `wavedata` というディレクトリが作成されます。

注記: Simulink モデルを含むディレクトリに書き込み権が必要です。

波形ビューアーでの表示を記述するデータは、`wavedata` ディレクトリ内の次のファイルに保存されています。

- `<design_name>.wcfg` - 波形設定ファイル。デザインで監視する信号名と、波形ビューアーでのこれらの信号の波形表示方法を指定するデータが含まれています。
- `<design_name>.wdb` - 波形データベース ファイル。波形ビューアーで波形を描画するのに必要なデータが含まれます。

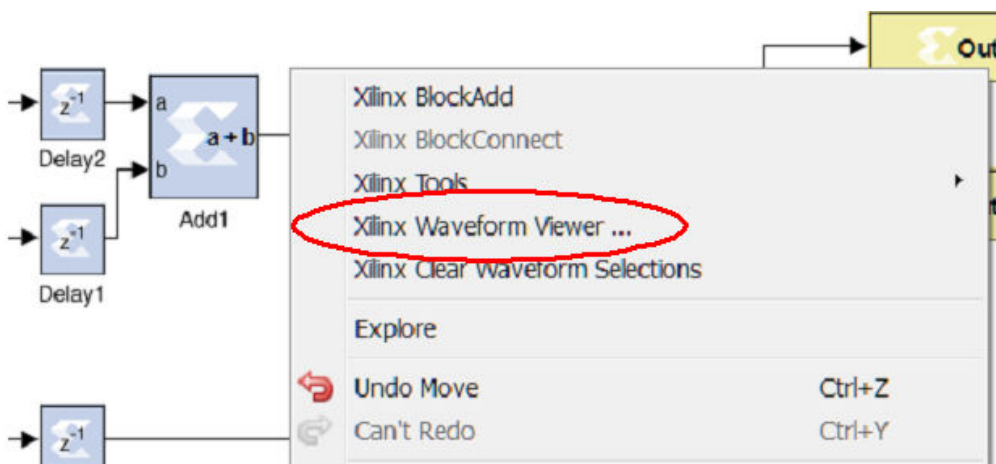
監視する信号の名前は、Simulink モデル (SLX ファイル) に保存されています。モデルの SLX ファイルを別のディレクトリに移動して、移動先のディレクトリでそのファイルを開いた場合など、Simulink モデルが `wavedata` ディレクトリにあるデータにアクセスできない場合、波形ビューアーを開いてデザインをシミュレーションすると、監視する信号を表示できます。これで、監視する信号の波形が波形ビューアーに表示されるようになります。

波形ビューアーの起動

波形ビューアーを開くには、次のいずれかの方法を使用します。

- 右クリック メニューを使用して開く:
モデルを右クリックし、[Xilinx Waveform Viewer] をクリックします。

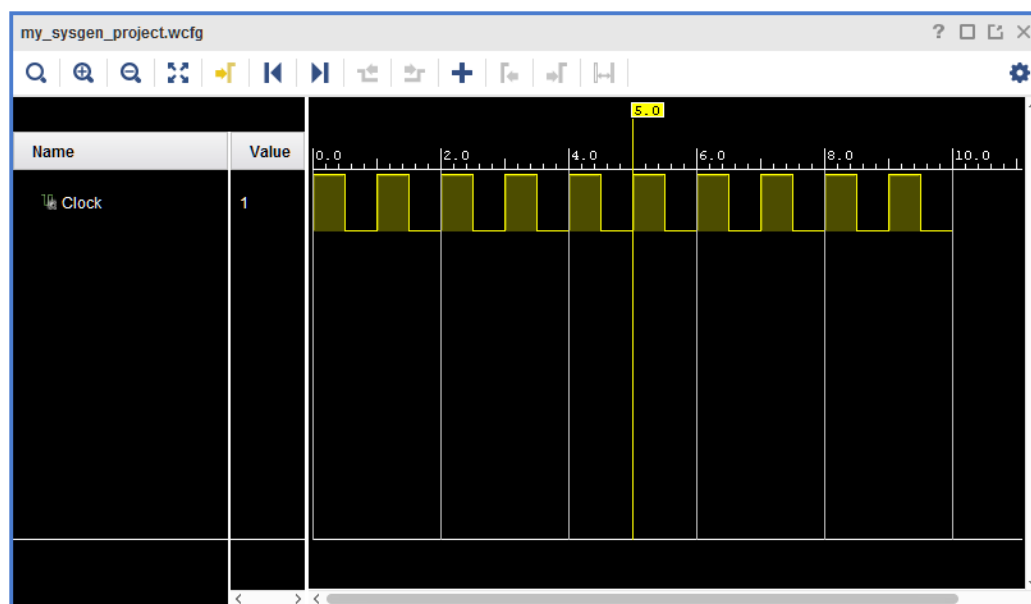
図 175: ザイリンクス波形ビューアー



右クリック メニューから開くと、波形ビューアーは次のような表示になります。

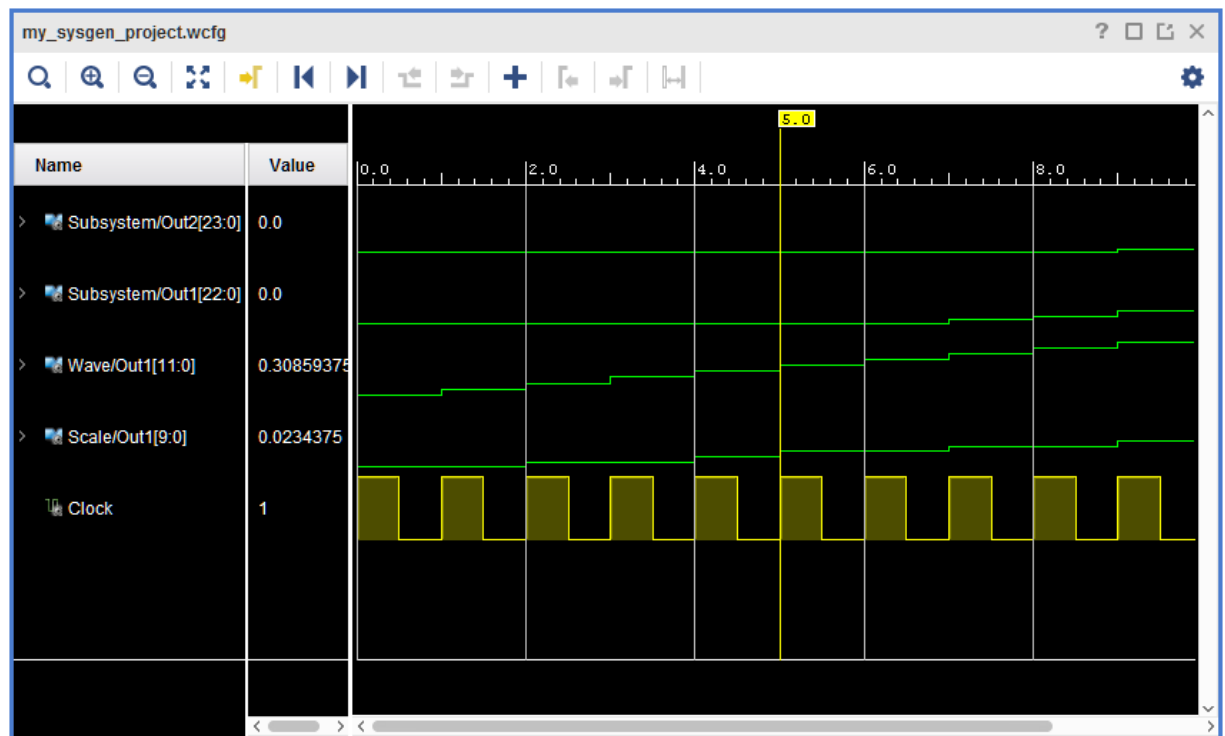
- 。 デザインに対し波形ビューアーを初めて開く場合、デザインのクロック信号の波形のみが表示され、ほかの波形は表示されません。この後、監視する信号を波形ビューアーに追加します ([波形ビューアーへの信号の追加](#)を参照)。

図 176: 波形ビューアーの表示



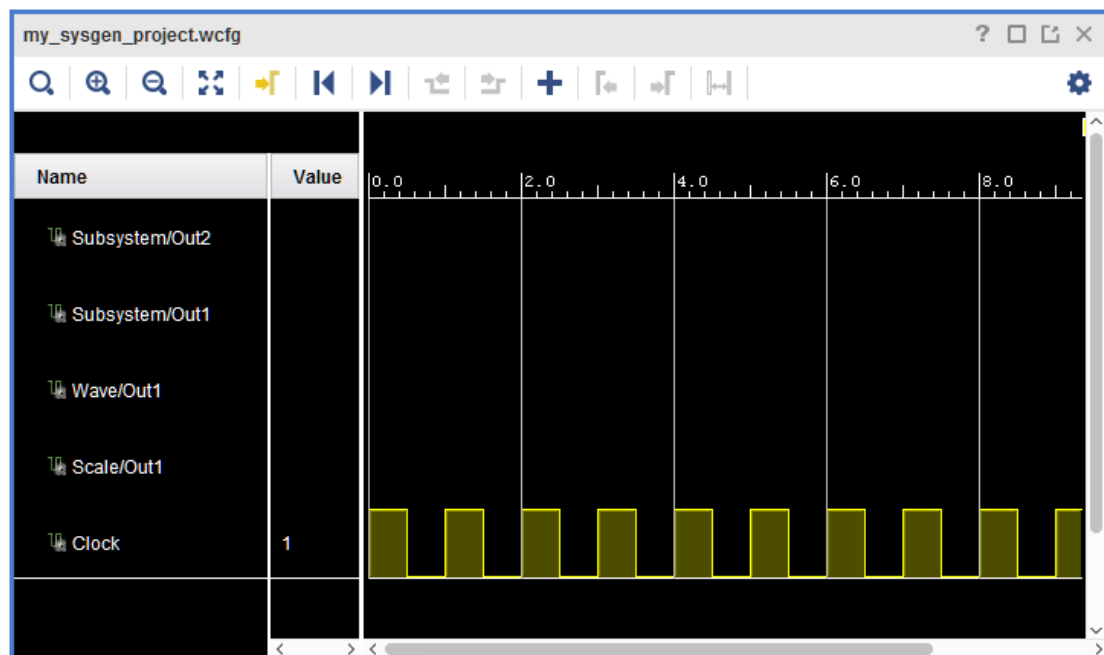
- 。 デザインに対し波形ビューアーを開いたことがあり、データを保存している場合、前回波形ビューアーを閉じたときに表示していた信号名および波形が波形ビューアーで開きます。

図 177: 波形ビューアーの信号



- 。 デザインに対し、波形ビューアーで以前信号を監視したけれど、保存したデータにアクセスできない場合 (モデルの SLX ファイルを別のディレクトリに移動させて、移動先でそのファイルを開いた場合など)、前回モデルを保存したときに監視した信号の名前が波形ビューアーに表示されます。モデルを再シミュレーションするまで、波形ビューアーには監視された信号の波形は表示されません。

図 178: 波形ビューアーの新しい信号



- 。 シミュレーション後に開く:

波形ビューアーで以前デザインの信号を監視したことがある場合は、モデルをシミュレーションすると波形ビューアーが自動的に開きます。

波形ビューアーへの信号の追加

ザイリンクス ブロックセットのブロックの入力および出力は、波形ビューアーに表示されます。各信号の波形を描画するのに必要なデータは、デザインと共に保存されておらず、シミュレーションで生成されます。信号波形は、波形ビューアーに信号を追加してモデルをシミュレーションすると初めて表示されます。

波形ビューアーに信号を追加するには、次の手順に従います。

1. 波形ビューアーを開いた状態で、System Generator モデルで信号を選択します。

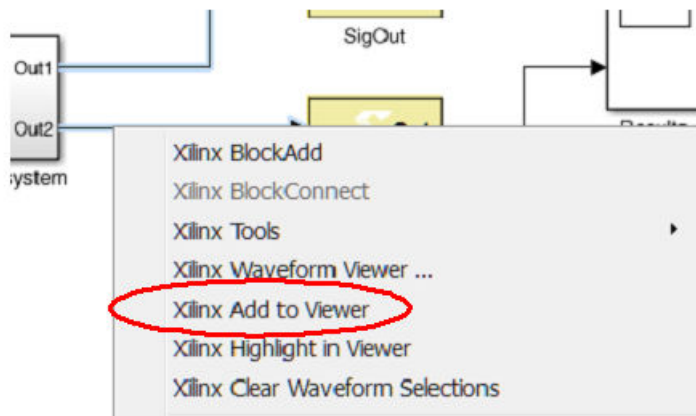
[Shift+] キーを押しながら信号を選択すると、複数の信号を選択できます。

注記: Gateway In ブロックの場合は、出力信号のみが波形ビューアーに表示されます。

2. System Generator モデルで選択した信号の 1 つを右クリックし、[Xilinx Add to Viewer] をクリックします。

注記: 波形ビューアーに現在表示されている信号を選択した場合は、[Xilinx Add to Viewer] は右クリック メニューに表示されません。

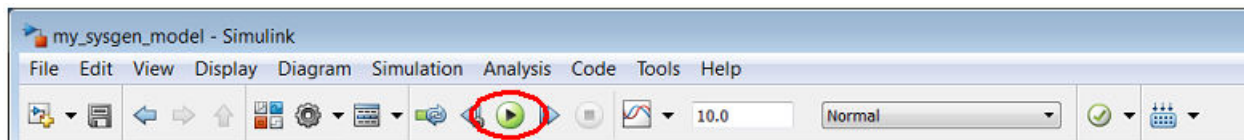
図 179: [Xilinx Add to Viewer] コマンド



選択した信号の名前が波形ビューアーに表示されます。

3. デザインをシミュレーションするまでは、信号の波形を描画するためのデータがないので、波形ビューアーには追加された信号の名前だけが表示されます。
4. モデルをシミュレーションします。

図 180: [Run] ボタン



シミュレーションが完了すると、追加された信号の波形が波形ビューアーに表示されます。

波形ビューアーから信号を削除

1. 波形ビューアーで削除する信号を選択します。

[Shift+] キーまたは [Ctrl+] キーを押しながら、複数の信号名を選択します (すべてを選択するには [Ctrl+A] を押します)。

2. 選択した信号名の 1 つで右クリックし、[Delete] をクリックします。

または

[Delete] キーを押します。

波形ビューアーから波形が削除されます。削除された波形は監視されなくなります。モデルを再シミュレーションしても、削除された波形は波形ビューアーには表示されません。

波形ビューアーとモデル間のクロスプローブ

ビューアーの波形と System Generator モデル内のワイヤを確認するには、クロスプローブ機能を使用すると便利です。

波形ビューアーとモデル間の信号は、次の方法でクロスプローブできます。

- 波形ビューアーから System Generator モデルへ信号をクロスプローブするには、波形ビューアーで信号名を 1 つまたは複数選択します。[Shift+] キーまたは [Ctrl+] キーを押しながら、複数の信号名を選択します。すべてを選択するには [Ctrl+A] を押します。

選択された信号は、System Generator モデルでオレンジ色にハイライトされます。

System Generator モデルでハイライトした信号のハイライトを解除するには、波形ビューアーで [Ctrl+] キーを押しながら信号名をクリックします。信号は System Generator モデルでハイライトされなくなります。

- System Generator モデルから波形ビューアーに信号をクロスプローブするには、次を実行します。

1. 波形ビューアーを開いた状態で、System Generator モデルで信号を選択します。

[Shift+] キーを押しながら信号を選択すると、複数の信号を選択できます。

2. System Generator モデルで選択した信号の 1 つを右クリックし、[Xilinx Highlight in Viewer] をクリックします。

注記: 波形ビューアーに現在表示されていない信号を選択した場合は、[Xilinx Highlight in Viewer] は右クリックメニューには表示されません。

図 181: [Xilinx Highlight in Viewer] コマンド



3. 波形ビューアーで選択した信号の名前がハイライトされていることを確認します。

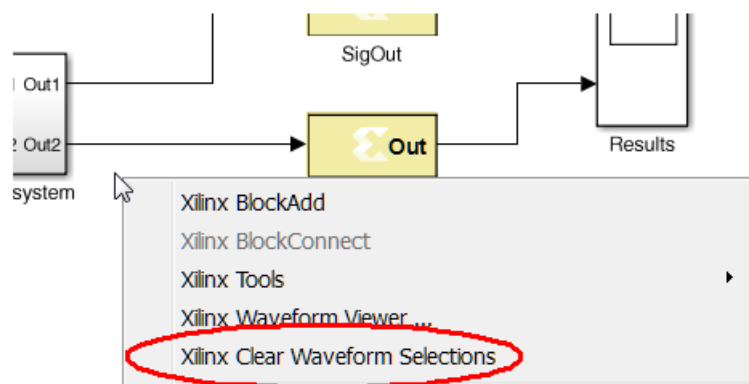
波形表示の削除

波形ビューアーで現在表示されている波形すべてを削除するには、次の手順を実行します。

1. System Generator モデルを右クリックします。
2. [Xilinx Clear Waveform Selections] をクリックします。

現在波形ビューアーで表示されている信号すべてが波形表示から削除され、波形ビューアーが閉じます。削除された波形は監視されなくなり、`wavedata` ディレクトリ (現在の波形ビューアーの表示を記述したデータを含む) が Simulink モデルを含むディレクトリから削除されます。

図 182: [Xilinx Clear Waveform Selections] コマンド



波形ビューアーを再び開くには、モデルを右クリックして [Xilinx Waveform Viewer] をクリックします。波形ビューアーが開いて、デザインのクロック信号の波形が表示され、その他の波形は表示されなくなります。

波形表示のカスタマイズおよび波形の解析

波形ビューアーで波形の表示方法をカスタマイズし、波形を解析するための方法が多数あります。デザインの開発およびトラブルシュートに波形ビューアーを使用する方法は、『Vivado Design Suite ユーザー ガイド: ロジック シミュレーション』 (UG900) の[このセクション](#)を参照してください。

波形ビューアーを使用するときのヒント

次のヒントは、System Generator モデルおよび波形ビューアーを使用して波形を解析するときに役立ちます。

- System Generator セッション中は波形ビューアーを開いたままにします。シミュレーションごとに波形ビューアーを閉じないでください。
- 波形ビューアーで信号のグループを選択する場合、そのグループのすべての信号を波形ビューアーから System Generator モデルにクロスプローブできます。
- System Generator モデルの複数の信号を波形ビューアーに追加するには、左マウス ボタンを押しながらマウスで信号を囲むように四角を描いて、信号を選択します。選択した信号名の 1 つで右クリックし、[Xilinx Add to Viewer] をクリックします。選択した信号が波形ビューアーに追加されます。
- System Generator モデルのブロックの出力信号に名前を付ける場合は、次の表に示す文字の使用を避けてください。これらの文字は VHDL または Verilog で予約されています。予約文字が名前に使用されている信号がモデルに含まれていると、その名前は、次の表に基づき、波形ビューアーで変更されます。

表 6: 予約文字

予約文字	変換後の文字
(#1
)	#2
[#3
]	#4
.	#5
,	#6
:	#7
\	#8

波形ビューアーを閉じる

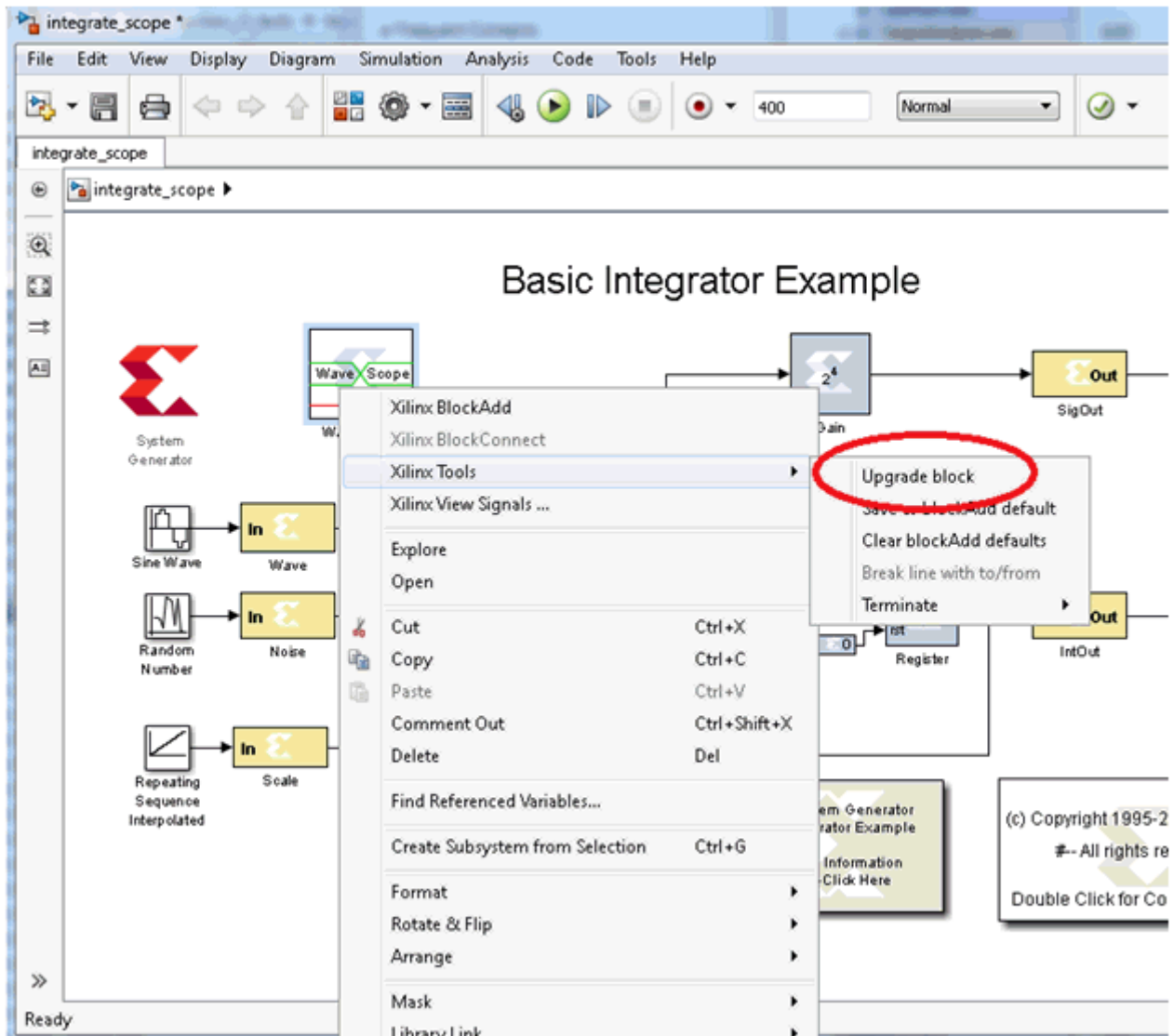
波形ビューアーを閉じるには、[File]→[Exit] をクリックします。波形データをまだ保存していない場合は、波形ビューアーを閉じる前にデータを保存するよう、メッセージが表示されます。

サポートされなくなった WaveScope ブロックの信号名の移行方法

サポートされなくなった WaveScope ブロックがデザインに含まれている場合、このブロックの既存のモニター信号名を、アップグレードしたブロックに移行する必要があります。

1. WaveScope ブロックを右クリックします。
2. [Xilinx Tools]→[Upgrade block] をクリックします。

図 183: ブロックのアップグレード



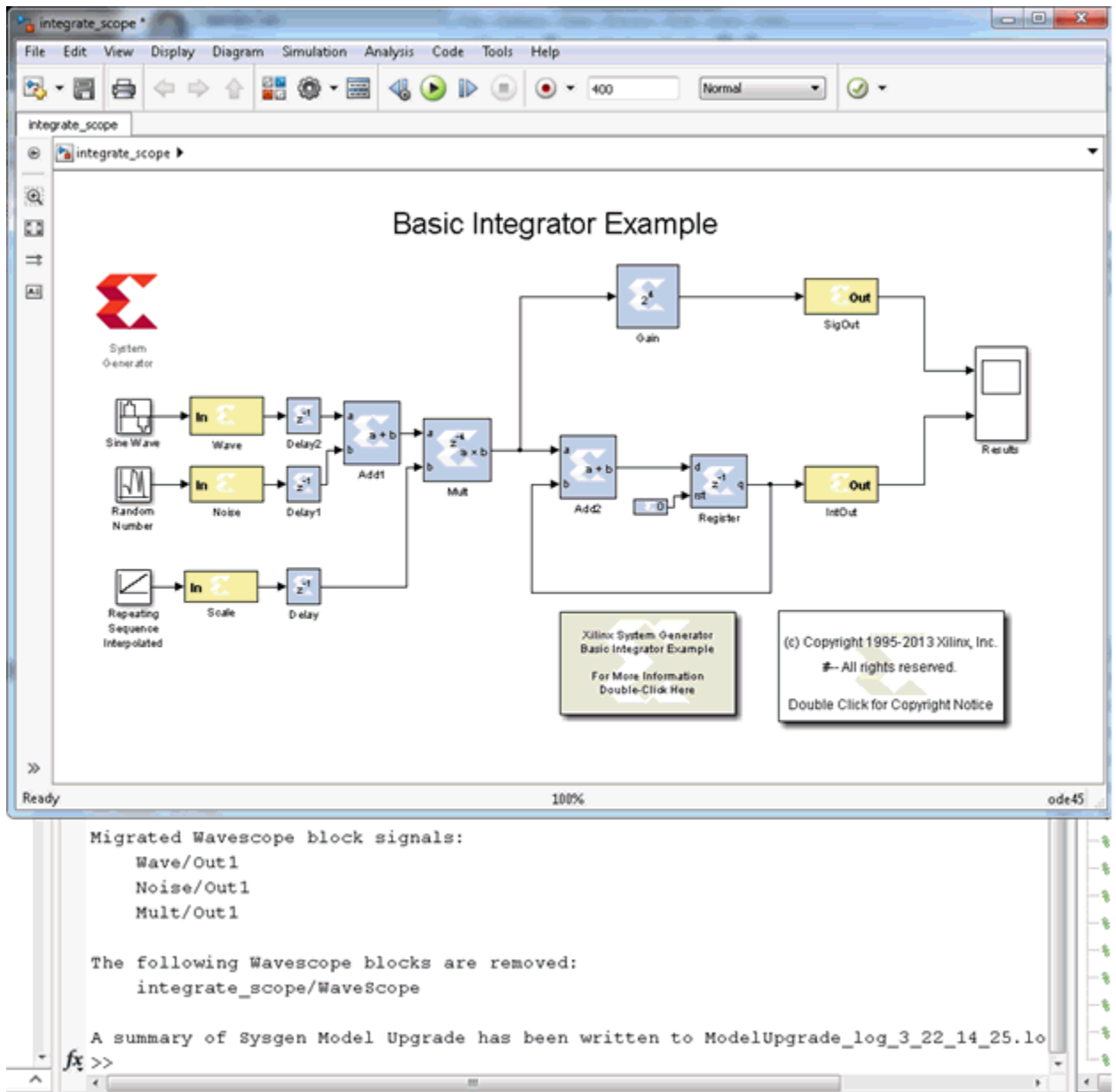
- アップグレードが実行されます。

図 184: [Upgrade Status] ダイアログ ボックス



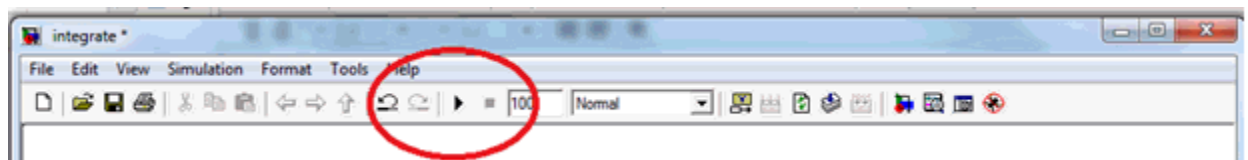
- アップグレードが実行されたら、次の図に示すように、サポートされなくなった WaveScope ブロックがモデルから削除され、MATLAB® コンソールにサマリが表示されます。

図 185: MATLAB コンソール



5. シミュレーション ボタンをクリックしてシミュレーションを実行します。

図 186: シミュレーション ボタン



シミュレーションが終了すると、サポートされなくなった WaveScope ブロックの信号名が波形ビューアーに表示されます。

その他のリソースおよび法的通知

ザイリンクス リソース

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

ソリューション センター

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

Documentation Navigator およびデザイン ハブ

ザイリンクス Documentation Navigator (DocNav) では、ザイリンクスの資料、ビデオ、サポート リソースにアクセスでき、特定の情報を取得するためにフィルター機能や検索機能を利用できます。DocNav を開くには、次のいずれかを実行します。

- Vivado[®] IDE で [Help] → [Documentation and Tutorials] をクリックします。
- Windows で [スタート] → [すべてのプログラム] → [Xilinx Design Tools] → [DocNav] をクリックします。
- Linux コマンド プロンプトに「docnav」と入力します。

ザイリンクス デザイン ハブには、資料やビデオへのリンクがデザイン タスクおよびトピックごとにまとめられており、これらを参照することでキー コンセプトを学び、よくある質問 (FAQ) を参考に問題を解決できます。デザイン ハブにアクセスするには、次のいずれかを実行します。

- DocNav で [Design Hub View] タブをクリックします。
- ザイリンクス ウェブサイトで[デザイン ハブ](#) ページを参照します。

注記: DocNav の詳細は、ザイリンクス ウェブサイトの [Documentation Navigator](#) ページを参照してください。DocNav からは、日本語版は参照できません。ウェブサイトのデザイン ハブ ページをご利用ください。

参考資料

- 『Vivado Design Suite リファレンス ガイド: System Generator を使用したモデル ベースの DSP デザイン』 (UG958)
- 『Vivado Design Suite チュートリアル: System Generator を使用したモデル ベースの DSP デザイン』 (UG948)
- 『Vivado Design Suite ユーザー ガイド: Vivado IDE の使用』 (UG893)
- 『Vivado Design Suite ユーザー ガイド: デザイン フローの概要』 (UG892)
- 『ISE から Vivado Design Suite への移行ガイド』 (UG911)
- 『Vivado Design Suite ユーザー ガイド: IP を使用した設計』 (UG896)
- 『Vivado Design Suite ユーザー ガイド: 制約の使用』 (UG903)
- 『Vivado Design Suite ユーザー ガイド: Tcl スクリプト機能の使用』 (UG894)
- 『Vivado Design Suite チュートリアル: デザイン フローの概要』 (UG888)
- 『Vivado Design Suite ユーザー ガイド: システム レベル デザイン入力』 (UG895)
- 『Vivado Design Suite ユーザー ガイド: リリース ノート、インストール、およびライセンス』 (UG973)
- 『UltraFast 設計手法ガイド (Vivado Design Suite 用)』 (UG949)

お読みください: 重要な法的通知

本通知に基づいて貴殿または貴社 (本通知の被通知者が個人の場合には「貴殿」、法人その他の団体の場合には「貴社」)。以下同じ) に開示される情報 (以下「本情報」といいます) は、ザイリンクスの製品を選択および使用することのためにのみ提供されます。適用される法律が許容する最大限の範囲で、(1) 本情報は「現状有姿」、およびすべて受領者の責任で (with all faults) という状態で提供され、ザイリンクスは、本通知をもって、明示、黙示、法定を問わず (商品性、非侵害、特定目的適合性の保証を含みますがこれらに限られません)、すべての保証および条件を負わない (否認する) ものとします。また、(2) ザイリンクスは、本情報 (貴殿または貴社による本情報の使用を含む) に関係し、起因し、関連する、いかなる種類・性質の損失または損害についても、責任を負わない (契約上、不法行為上 (過失の場合を含む)、その他のいかなる責任の法理によるかを問わない) ものと、当該損失または損害には、直接、間接、特別、付随的、結果的な損失または損害 (第三者が起こした行為の結果被った、データ、利益、業務上の信用の損失、その他あらゆる種類の損失や損害を含みます) が含まれるものとし、それは、たとえ当該損害や損失が合理的に予見可能であったり、ザイリンクスがそれらの可能性について助言を受けていた場合であったとしても同様です。ザイリンクスは、本情報に含まれるいかなる誤りも訂正する義務を負わず、本情報または製品仕様のアップデートを貴殿または貴社に知らせる義務も負いません。事前の書面による同意のない限り、貴殿または貴社は本情報を再生産、変更、頒布、または公に展示してはなりません。一定の製品は、ザイリンクスの限定的保証の諸条件に従うこととなるので、<https://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。IP コアは、ザイリンクスが貴殿または貴社に付与したライセンスに含まれる保証と補助的条件に従うことになります。ザイリンクスの製品は、フェイルセーフとして、または、フェイルセーフの動作を要求するアプリケーションに使用するために、設計されたり意図されたりしていません。そのような重大なアプリケーションにザイリンクスの製品を使用する場合のリスクと責任は、貴殿または貴社が単独で負うものです。<https://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。

自動車用のアプリケーションの免責条項

オートモーティブ製品 (製品番号に「XA」が含まれる) は、ISO 26262 自動車用機能安全規格に従った安全コンセプトまたは余剰性の機能 (「セーフティ 設計」) がない限り、エアバッグの展開における使用または車両の制御に影響するアプリケーション (「セーフティ アプリケーション」) における使用は保証されていません。顧客は、製品を組み込むすべてのシステムについて、その使用前または提供前に安全を目的として十分なテストを行うものとし、セーフティ設計なしにセーフティ アプリケーションで製品を使用するリスクはすべて顧客が負い、製品責任の制限を規定する適用法令および規則にのみ従うものとし、

商標

© Copyright 2016-2020 Xilinx, Inc. Xilinx、Xilinx のロゴ、Alveo、Artix、Kintex、Spartan、Versal、Virtex、Vivado、Zynq、およびこの文書に含まれるその他の指定されたブランドは、米国およびその他の国のザイリンクス社の商標です。AMBA、AMBA Designer、Arm、ARM1176JZ-S、CoreSight、Cortex、PrimeCell、Mali、および MPCore は、EU およびその他の国の Arm Limited の商標です。すべてのその他の商標は、それぞれの所有者に帰属します。

この資料に関するフィードバックおよびリンクなどの問題につきましては、jpn_trans_feedback@xilinx.com まで、または各ページの右下にある [フィードバック送信] ボタンをクリックすると表示されるフォームからお知らせください。フィードバックは日本語で入力可能です。いただきましたご意見を参考に早急に対応させていただきます。なお、このメール アドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。