

EDK コンセプト、ツール、 テクニック

効率的なエンベデッドシステム構築を サポートするハンディ ガイド

UG683 EDK 11

本資料は英語版 (v11) を翻訳したものです。英語版の最新バージョンが
リリースされている場合には、そちらを必ずご参照ください。



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

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

© 2009 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Xilinx is providing this product documentation, hereinafter "information," to you "AS IS" with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice.

XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.

© 2009 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	改訂内容
2007 年 1 月 1 日	9.1i	EDK 9.1i のマニュアル リリース
2007 年 9 月 5 日	9.2i	EDK 9.2i のマニュアル リリース
2007 年 11 月 5 日	10.1	ISE Design Suite 10.1 のマニュアル リリース
2008 年 9 月 18 日	10.1	ISE Design Suite 10.1 サービス パック 3 のマニュアル リリース
2009 年 5 月 11 日	11	ISE Design Suite 11 のマニュアル リリース

目次

第 1 章: 入門	
このマニュアルについて	5
その他の資料	5
添付ファイル	6
EDK によるエンベデッド プロセッサ デザインの簡略化	6
ISE Design Suite : Embedded Edition	6
エンベデッド開発キット (EDK)	6
ツールによるデザイン プロセスの短縮	7
設計開始の準備	8
第 2 章: 新規プロジェクトの作成	
Base System Builder	11
BSB ウィザードを使用する理由	11
BSB ウィザードで可能な操作	11
BSB ウィザードと ISE Design Suite	13
BSB およびカスタム ボードに関するメモ	16
次の操作	16
第 3 章: Xilinx Platform Studio の使用	
XPS とは	17
XPS ソフトウェア	17
プロジェクト情報エリア	19
[System Assembly View]	21
コンソール ウィンドウ	22
[Start Up Page]	23
XPS ツール	23
XPS のディレクトリ構造	24
ディレクトリ	25
次の操作	26
第 4 章: エンベデッド プラットフォームの操作	
ハードウェア プラットフォームの概要	27
XPS でのハードウェア プラットフォーム開発	27
[System Assembly View] でのハードウェア プラットフォーム	28
ハードウェア プラットフォームの生成	28
ハードウェア プラットフォームのエクスポート	28
次の操作	31
第 5 章: ソフトウェア開発キット	
SDK について	33
実行された処理	35
次の操作	40
第 6 章: SDK での編集およびデバッグ	
ドライバ	41
SDK のパースペクティブとウィンドウのタイプ	41
次の操作	52

第 7 章: 独自の IP の作成

CIP ウィザードの使用.....	53
IP 作成の概要.....	54
CIP ウィザードを使用したカスタム IP の作成.....	54
CIP ウィザードの実行前に知っておくべきこと.....	55
サンプル デザインの説明.....	63
ファイルの内容の確認.....	66
プロセッサ システムへのカスタム IP の追加.....	67
次の操作.....	75

第 8 章: デュアル プロセッサ デザインの作成とデバッグ

BSB を使用したデュアル プロセッサ システムの作成.....	77
----------------------------------	----

付録 A: Project Navigator での ModelSim を使用したシミュレーション

シミュレーション.....	85
出力の観察.....	87
実行された処理.....	88

付録 B: IP バス ファンクション モデル シミュレーション

BFM についてと使用する理由.....	89
----------------------	----

付録 C: 用語集

EDK で使用される用語.....	95
-------------------	----

入門

このマニュアルについて

ザイリンクス エンベデッド開発キット (EDK) は、ザイリンクス FPGA (フィールド プログラマブル ゲート アレイ) デバイスにインプリメントするエンベデッド プロセッサ システムを設計するツールおよび IP (Intellectual Property) のセットです。

このマニュアルでは、EDK を使用してカスタム エンベデッド プロセッサ システムを開発するデザイン フローを説明します。基礎的な情報も含まれますが、EDK の機能とその使用方法の説明が主な内容です。

このマニュアルは、次のような場合に適しています。

- EDK とそのユーティリティに関する基礎的な入門情報が必要な場合
- エンベデッド プロセッサ システムの開発からしばらく離れていた場合
- ザイリンクス EDK ツールをインストールする場合
- プロセッサ システムの設計中に簡単に参照できる資料が必要な場合

メモ：このマニュアルは、Windows オペレーティング システムでの動作に基づいて記述されています。Linux システムでは、ツールの動作およびグラフィカル ユーザー インターフェイス (GUI) が異なる場合があります。

チュートリアル

ソフトウェア ツールについて学ぶには、使用してみるのが最良の方法です。このマニュアルには説明したツールを実際に使用してみるチュートリアルがあり、手順に従ってサンプル プロジェクトを作成できます。このセクションでは、自動機能を使用した場合に実行される処理についても説明しています。

その他の資料

EDK に関するその他の資料は、次のサイトから参照できます。

http://japan.xilinx.com/ise/embedded/edk_docs.htm

ISE (Integrated Software Environment) に関する資料は、次のサイトから参照できます。

http://japan.xilinx.com/support/documentation/sw_manuals/xilinx11/manuals.pdf

添付ファイル

このマニュアルには、チュートリアルを実行するためのサンプルプロジェクト ファイルが添付されています。Adobe Acrobat Reader の左下にあるクリップ アイコンをクリックすると、これらのファイルを参照できます。

添付されているファイルは、次のとおりです。

- bus_transaction_bfl_code.txt
- leds.c
- pn.do file
- pwm_light.vhd
- user_logic.vhd

EDK によるエンベデッド プロセッサ デザインの簡略化

エンベデッド システムは複雑です。エンベデッド デザインのハードウェア部分とソフトウェア部分を機能させるだけでも課題であり、2 つのデザイン コンポーネントが 1 つのシステムとして機能するよう統合するという作業も加わります。これに FPGA の設計が加われば、さらに複雑さが増します。

設計プロセスを簡略化するため、ザイリンクスでは複数のツール セットを提供しています。これらのツール名、プロジェクト ファイル名、略称を覚えておくとうりです。このマニュアルの最後に EDK で使用される用語をリストした付録 C「用語集」があるので、参考になしてください。

ISE Design Suite : Embedded Edition

ザイリンクスでは、ISE Design Suite によりさまざまな開発システム ツールを提供しています。用途に応じた異なるエディションがあり、エンベデッド システムの開発用には Embedded Edition が提供されています。Embedded Edition には、次のツールが含まれます。

- ISE (Integrated Software Environment)
- PlanAhead デザイン開発ツール、ChipScope™ Pro (FPGA デザインのデバッグに有益)
- エンベデッド開発キット (EDK)

FPGA デザインでの ISE ツールの使用法は、次のページから ISE のヘルプおよびマニュアルを参照してください。

http://japan.xilinx.com/support/documentation/sw_manuals/xilinx11/manuals.pdf

エンベデッド開発キット (EDK)

EDK は、ザイリンクス FPGA デバイスにインプリメントするエンベデッド プロセッサ システムを設計するツールおよび IP のセットです。

Xilinx Platform Studio (XPS)

XPS を実行するには、ISE がインストールされている必要があります。ISE は、エンベデッド プロセッサ システムおよびそのデザインに関するすべてを包括するツールです。

XPS は、エンベデッド プロセッサ システムのハードウェア部分を定義するのに使用する開発環境です。XPS は、bash シェル コマンド ライン、バッチ モード、GUI を使用して実行できます。このマニュアルでは、GUI を使用する方法を示します。

ソフトウェア開発キット (SDK)

SDK は XPS に補足的に使用する統合開発環境で、C/C++ エンベデッド ソフトウェア アプリケーションの作成および検証に使用します。SDK は Eclipse™ オープン ソース フレームワークに基づいて構築されているので、GUI にはなじみがあるかもしれません。

その他の EDK コンポーネント

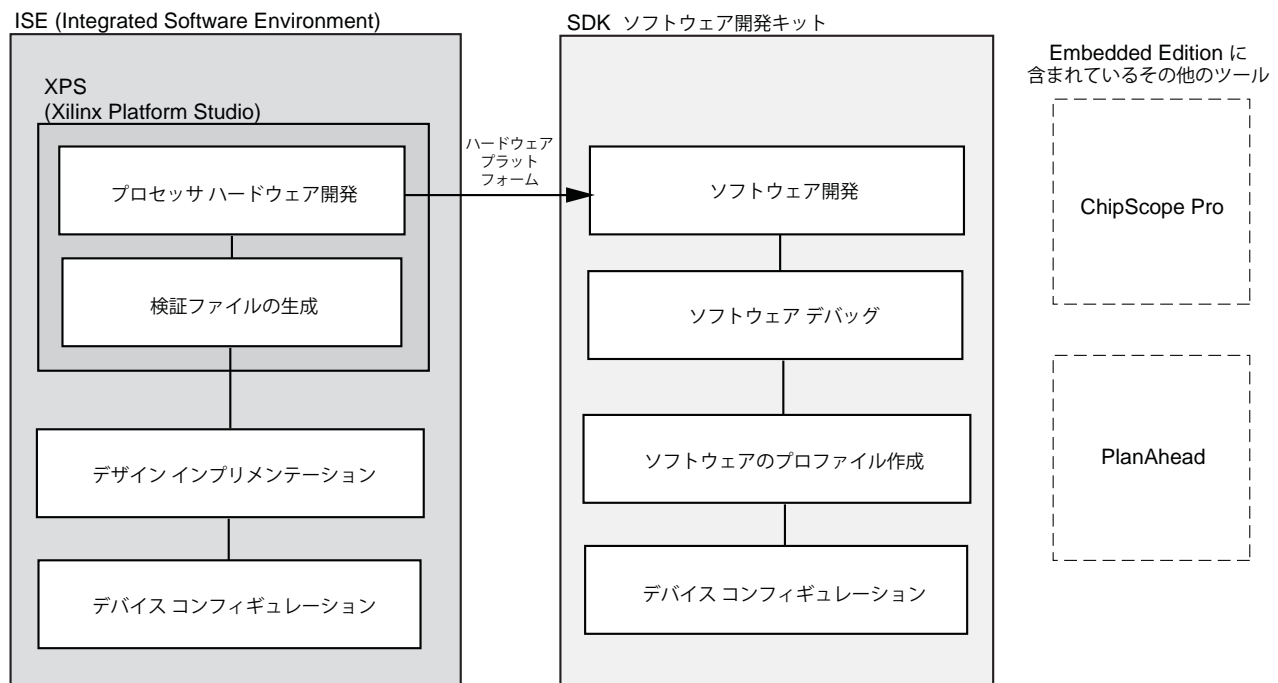
EDK には、上記のほかに次のものが含まれています。

- ザイリンクス エンベデッド プロセッサ用のハードウェア IP
- エンベデッド ソフトウェア開発用のドライバおよびライブラリ
- MicroBlaze™ および PowerPC® プロセッサをターゲットとした C/C++ ソフトウェア開発用の GNU コンパイラおよびデバッグ
- マニュアル
- サンプル プロジェクト

EDK に含まれるユーティリティを使用すると、エンベデッド システムの設計フローを最初から最後まで実行できます。

ツールによるデザイン プロセスの短縮

次に、エンベデッド デザインのフローを示します。



X11124

図 1-1: 基本的なエンベデッド システム設計プロセスのフロー

通常は、まず ISE 開発ソフトウェアを使用してエンベデッド プロセッサ ソースを作成し、これを ISE プロジェクトに追加します。

- XPS は、主にエンベデッド プロセッサ ハードウェア システムの開発に使用します。マイクロ プロセッサおよびペリフェラルの仕様、これらのコンポーネントの接続およびプロパティの設定に、XPS を使用します。
- ソフトウェアの開発には、SDK を使用します。ISE Design Suite: Embedded Edition のこのリリースから、SDK はスタンドアロン アプリケーションとして入手することも可能です。SKD は個別の実行ファイルであり、ほかのツールなしで実行できます。
- ハードウェア プラットフォームの機能を検証するには、デザインを HDL シミュレータでシミュレーションします。

XPS では、次の 3 つのシミュレーションを実行できます。

- ◆ ビヘイビア
- ◆ 構造
- ◆ タイミング

XPS では、シミュレーションに使用する HDL ファイルも含め、検証プロセス構造が自動的に設定されるので、クロック タイミングおよびリセット ステイミュラス情報を入力し、アプリケーション コードを供給するだけで済みます。

- XPS でデザインを設計したら、ISE に戻って FPGA コンフィギュレーション ファイルを生成し、ターゲット デバイスにプログラムします。
- エンベデッド デザインを含むビットストリームを FPGA にコンフィギュレーションしたら、SDK でソフトウェア プロジェクトの ELF (Executable and Linkable Format) ファイルをダウンロードしてデバッグします。

XPS に関連したエンベデッド デザイン設計プロセスの詳細は、http://japan.xilinx.com/ise/embedded/edk_docs.htm から『エンベデッド システム ツール リファレンス マニュアル』の「設計プロセスの概要」を参照してください。

設計開始の準備

ツールについて詳細に説明する前に、ツールが正しくインストールされ、チュートリアルの手順を実行できるように環境が設定されているかを確認します。

インストール要件：EDK ツールの実行に必要なもの

ISE および EDK

ISE と EDK は、ISE Design Suite: Embedded Edition に含まれています。ソフトウェアおよび最新のアップデート がインストールされていることを確認してください。<http://japan.xilinx.com/support> で最新のソフトウェア バージョンを確認できます。

EDK のインストール要件

Linux の bash シェル

Linux プラットフォームで EDK を実行する場合は、bash シェルが必要です。[『ISE Design Suite 11 : インストール、ライセンス、リリース ノート』](#)に掲載されているサポートされるプラットフォームも確認してください。

ソフトウェア ライセンス

ISE 11.1 リリースから、ザイリンクス ソフトウェアに FLEXnet ライセンスが使用されるようになりました。ソフトウェアを初めて実行する際、ライセンス検証プロセスが実行されます。有効なライセンスが存在しない場合、Xilinx License Configuration Manager を使用してライセンスを取得できます。ソフトウェアを試用する場合は、評価ライセンスを生成できます。

ザイリンクス ソフトウェアのライセンスの詳細は、[『ISE Design Suite 11 : インストール、ライセンス、リリース ノート』](#)を参照してください。

シミュレーションのインストール要件

EDK ツールを使用してシミュレーションを実行するには、適切なシミュレータをインストールし、シミュレーション ライブラリをコンパイルする必要があります。

1. SecureIP をサポートする混合言語シミュレータ (ModelSim PE/SE v6.4b または NCSim 8.1-s009 以降) が必要です。MXE は混合言語および SecureIP をサポートしていないので、エンベデッド デザインには使用できません。

オプションで、CoreConnect™ ツールキットをインストールします。CoreConnect ツールキットは、バス ファンクション モデル (BFM) シミュレーションを実行する場合にのみ必要です。BFM シミュレーションを実行しない場合は、CoreConnect ツールキットをインストールする必要はありません。CoreConnect は IBM から提供されている無償のユーティリティで、ザイリンクスの次の Web サイトからダウンロードできます。

http://japan.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=dr_pcentral_coreconnect

この Web ページで適切な選択をして注文および登録すると、ダウンロードできるようになります。

シミュレーションのインストール要件

2. シミュレーション ライブラリをコンパイルしていない場合は、XPS ヘルプに説明されている手順に従ってコンパイルしてください。
 - a. XPS ヘルプは、[Help] → [Help Topics] をクリックすると表示されます。また、http://japan.xilinx.com/support/documentation/sw_manuals/xilinx11/manuals.pdf から参照できます。
 - b. 「エンベデッド プロセッサの設計手順」→「シミュレーション」→「シミュレーション ライブラリのコンパイル」→「XPS でのシミュレーション ライブラリのコンパイル」トピックを参照してください。

インストール プロセスの詳細は、[『ISE Design Suite 11 : インストール、ライセンス、リリース ノート』](#)を参照してください。

新規プロジェクトの作成

この章では、ザイリンクス エンベデッド開発キット (EDK) を使用してエンベデッド システムの開発を開始します。

Base System Builder

BSB について

Base System Builder (BSB) は、機能するデザインを短時間で効率的に作成するウィザードで、XPS (Xilinx Platform Studio) に含まれています。BSB ウィザードで作成したデザインは、要件に応じてカスタマイズできます。

このセクションの最後に、BSB ウィザードを使用してプロジェクトを作成する最初のチュートリアルがあります。

BSB ウィザードを使用する理由

ザイリンクスでは、新規エンベデッド デザイン プロジェクトの基本デザインを作成するのに BSB ウィザードを使用することをお勧めします。BSB ウィザードで必要なデザインを作成できる場合もありますが、カスタマイズが必要な場合でも、BSB ウィザードで基礎となるハードウェアおよびソフトウェア プラットフォームを自動的にコンフィギュレーションできるので、設計時間を大幅に短縮できます。ウィザードを実行すると、必要な基本要素をすべて含むプロジェクトが作成されるので、このプロジェクトをカスタマイズしてより複雑なシステムを作成できます。

BSB ウィザードで可能な操作

BSB ウィザードでは、プロジェクト ファイルの作成、ボードの選択、プロセッサおよび I/O インターフェイスの選択と設定、内部ペリフェラルの追加、ソフトウェアのセットアップ、およびシステム サマリ レポートの生成が可能です。

選択したボードのシステム コンポーネントおよびコンフィギュレーションが認識され、適切なオプションが表示されます。

XMP ファイルの作成時に、BSB ウィザードで作成した別のプロジェクトの設定を適用することも可能です。

ボード タイプの選択

BSB ウィザードでは、リストからボード タイプを選択するか、カスタム ボードを作成できます。

サポートされるボード

ザイリンクスまたはザイリンクス パート ナーから 提供されているエンベデッド プロセッサ開発ボードを使用する場合、BSB ウィザードでボードで使用可能なペリフェラルを選択し、自動的に FPGA のピン配置をボード に一致させ、ボード にダウンロードして実行できる完全なプラットフォームとテスト アプリケーションを作成できます。各オプションには、デフォルト 値があらかじめ選択されています。この基本プロジェクトは、XPS でさらにカスタマイズするか、ISE のインプリメンテーション ツールを使用してインプリメント できます。

ボード タイプの選択

EDK をインストールすると、ザイリンクス ボード ファイルのみがインストールされます。サードパーティ ボードを使用する場合、ボード サポート ファイルを追加する必要があります。BSB ウィザードの [Board Selection] ページに、サードパーティのボード サポート ファイルを見つけるのに役立つページへのリンクがあります。ファイルをインストールすると、BSB ウィザードのドロップダウン メニューにこれらのボードが表示されるようになります。

カスタム ボード

カスタム ボードのデザインを開発している場合、BSB ウィザードで使用可能なプロセッサ コア (ターゲット FPGA デバイスに応じて MicroBlaze™ または PowerPC® プロセッサ) を選択し、IP ライブラリに含まれている互換性のある、頻繁に使用されるペリフェラル コアと接続できます。生成されたハードウェア システムを開始点として使用し、必要に応じてさらにプロセッサおよびペリフェラルを追加できます。カスタム ペリフェラルの作成も含むこの作業には、XPS に含まれるユーティリティを使用できます。

プロセッサの選択とコンフィギュレーション

MicroBlaze または PowerPC のどちらかを選択し、次のオプションを指定できます。

- リファレンス クロックの周波数
- プロセッサ バス クロックの周波数
- リセットの極性
- デバッグ用のプロセッサのコンフィギュレーション
- キャッシュ設定
- 浮動小数点ユニット (FPU) 設定

複数の I/O インターフェイスの選択とコンフィギュレーション

BSB ウィザードでは、定義済みのボードで使用可能な外部メモリおよび I/O デバイスが認識され、デバイスに応じて次を選択できます。

- 使用するデバイス
- ボーレート
- ペリフェラル
- データ ビット数
- パリティ
- 割り込みの使用/不使用

外部メモリおよび I/O デバイスのデータシートを BSB ウィザードから開くことができます。

内部ペリフェラルの追加

BSB ウィザードで、ペリフェラルを追加できます。ペリフェラルは、選択したボードおよび FPGA デバイス アーキテクチャによりサポートされていることが必要です。カスタム ボードでは、一部のペリフェラルしか通常の選択および自動システム接続できません。

ソフトウェアの設定

標準の入力および出力デバイスは BSB ウィザードで指定でき、サンプル C アプリケーションを生成できます。BSB ウィザードで生成したサンプル C アプリケーションは、[付録 A「Project Navigator での ModelSim を使用したシミュレーション」](#)のシミュレーション例で使用されます。ソフトウェアの開発には、ソフトウェア開発キット (SDK) を使用することをお勧めします。このマニュアルには、SDK を使用するチュートリアルが含まれています。ソフトウェア デバッグ チュートリアルで使用するサンプル C アプリケーションは、SDK で生成したものです。

システム サマリ

BSB ウィザードですべてのオプションを選択すると、システム サマリが表示されます。この時点で、プロジェクトを生成するか、前のダイアログ ボックスに戻って設定を変更できます。

チュートリアルで 使用されるデバイスと ボード

このマニュアルでは、Spartan®-3A DSP 1800A スタータ ボードを使用し、MicroBlaze プロセッサをターゲットとしています。選択するオプションは、[14 ページの「チュートリアル：新規エンベデッド プロジェクトの作成」](#)にリストされています。

PowerPC 405 (Virtex®-4 FX) または PowerPC 440 (Virtex-5 FXT) プロセッサを含む FPGA を搭載したボードを使用する場合は、MicroBlaze または該当する PowerPC プロセッサを使用できます。ほとんどの場合、ツールの動作は同じです。

BSB ウィザードと ISE Design Suite

ISE ソフトウェアで新規プロジェクトを作成します。プロジェクトの作成には、Project Navigator の New Project Wizard を使用します。プロジェクトを作成し、エンベデッド プロセッサ ソースを作成すると、自動的に XPS が開き、BSB を使用してエンベデッド プロセッサ プロジェクトを作成できます。XPS が起動するのに多少の時間がかかります。

XMP ファイル

XMP (Xilinx Microprocessor Project) ファイルは、開発するエンベデッド システムの最上位ファイル記述です。すべてのプロジェクト情報は、XMP ファイルに保存されます。

XMP ファイルは、HDL コードや制約ファイルなどのその他のソース ファイルと同様に、ISE で作成され管理されます。これらのプロセスを次のチュートリアルで学びます。

チュートリアル：新規エンベデッド プロジェクトの作成

このチュートリアルでは、ISE Project Navigator を起動して、エンベデッド プロセッサ システムを最上位ソースとするプロジェクトを作成します。

1. ISE Project Navigator を起動します。
2. [File] → [New Project] をクリックし、New Project Wizard を開きます。
3. 次の表の指示に従って、ウィザードの各ページでオプションを設定します。

ウィザードのページ	システム特性	設定または使用するコマンド
[Create New Project]	<ul style="list-style-type: none"> プロジェクト名 ([Name]) プロジェクトの保存ディレクトリ ([Location]) および説明 ([Description]) 最上位ソース タイプ ([Top-level source type]) 	<ul style="list-style-type: none"> プロジェクト名を入力します (スペースは含めない)。 プロジェクトの保存ディレクトリを選択します (スペースは含めない)。説明を記述することもできます。 [HDL] を選択します (デフォルト)。
[Device Properties]	<ul style="list-style-type: none"> 製品カテゴリ ([Product Category]) デバイス ファミリー ([Family]) デバイス ([Device]) パッケージ ([Package]) スピード グレード ([Speed]) 合成ツール ([Synthesis Tool]) シミュレータ ([Simulator]) 優先する言語 ([Preferred Language]) 	<ul style="list-style-type: none"> [All] [Spartan-3A DSP] [XC3SD1800A] [FG676] [-4] [XST (VHDL/Verilog)] 使用するシミュレータを選択します。* [VHDL] <p>その他の設定はデフォルトのままにします。</p> <p>* サポートされているシミュレータは、8 ページの「インストール要件：EDK ツールの実行に必要なもの」にリストされています。</p>
[Create New Source]		[New Source] をクリックします。New Source Wizard が開きます。
New Source Wizard の [Select Source Type]	<p>ソース タイプとして [Embedded Processor] を選択し、次の値を指定します。</p> <ul style="list-style-type: none"> ファイル名 ([File name]) 保存ディレクトリ ([Location]) 	<ul style="list-style-type: none"> 「system」と入力します。 デフォルトのディレクトリを使用します。 <p>[Add to project] をオンのままにします。</p> <p>[Next] をクリックし、次のダイアログ ボックスで [Finish] をクリックします。</p>
[Create New Source]	これ以外に新規ソースは追加しません。	[Next] をクリックします。
[Add Existing Sources]	何も追加しません。	[Next] をクリックします。
[Project Summary]		[Finish] をクリックします。

New Project Wizard が終了すると、ISE でエンベデッド プロセッサ システムが含まれることが認識され、XPS が起動します。

4. 「This project appears to be a blank project. Do you want to create a Base System using the BSB Wizard?」(このプロジェクトは空です。BSB ウィザードを使用して基本システムを作成しますか) というメッセージが表示されます。これには少し時間がかかる場合があります。[Yes] をクリックします。

BSB ウィザードが起動します。次の表の指示に従ってプロジェクトを作成します。

メモ：表に設定またはコマンドがない場合は、デフォルト値をそのまま使用します。

ウィザードのページ	システム特性	設定または使用するコマンド
[Welcome to the Base System Builder]	プロジェクト タイプ オプション	[I would like to create a new design] をオンにします。
[Board Selection]	<ul style="list-style-type: none"> ボード ベンダー ([Board Vendor]) ボード名 ([Board Name]) ボードのリビジョン ([Board Revision]) 	<ul style="list-style-type: none"> [Xilinx] を選択します。 [Spartan-3A DSP 1800A Starter Board] を選択します。 1800A ボードには Spartan-3A DSP デバイスが含まれており、MicroBlaze プロセッサを 1 つまたは複数コンフィギュレーションできます。 [1] を選択します (デフォルト)。
[System Configuration]	システムのタイプ	[Single-Processor System] をオンにします。
[Processor Configuration]	<ul style="list-style-type: none"> プロセッサのタイプ ([Processor Type]) システム クロック周波数 ([System Clock Frequency]) ローカル メモリ ([Local Memory]) 浮動小数点ユニットの使用 ([Enable Floating Point Unit]) 	<ul style="list-style-type: none"> [MicroBlaze] を選択します。 デフォルトのシステム クロック周波数 (62.5 MHz) をそのまま使用します。 [16 KB] を選択します。 浮動小数点ユニットは無効にします (チェック ボックスをオフ)。
[Peripheral Configuration]	プロセッサ 1 (MicroBlaze) の ペリフェラルのリスト ([Processor 1 (MicroBlaze) Peripherals])	デフォルトのリストから次のペリフェラルを削除します。 <ul style="list-style-type: none"> Ethernet_MAC SPI_FLASH その他のコアはそのままにします。
[Cache Configuration]	命令キャッシュ ([Instruction Cache]) およびデータ キャッ シュ ([Data Cache])	オフにします。

ウィザードのページ	システム特性	設定または使用するコマンド
[Application Configuration]	サンプル アプリケーションのオプション ([Example Applications])	デフォルト値をそのまま使用します。
[Summary]	システム サマリ ページ	<p>すべてのシステム コンポーネントを選択し、コンフィギュレーションすると、システムのサマリが表示されます。このページで選択を確認します。</p> <p>次のコンポーネントを含む MicroBlaze プロセッサが作成されているはずです。</p> <ul style="list-style-type: none"> • 複数ポート メモリ コントローラ (mpmc) • XPS GPIO (3 インスタンス) • XPS UartLite (xps_uartlite) • LMB BRAM IF コントローラ (2 インスタンス) <p>前のページに戻って設定を変更できます。</p> <p>BSB ウィザードによりデフォルトのメモリ マップが作成されます。このメモリ マップは BSB ウィザードからは変更できませんが、BSB ウィザードを閉じた後に変更できます。</p>

5. システム サマリを確認したら、[Finish] をクリックします。

BSB およびカスタム ボードに関するメモ

カスタム ボードを含むプロジェクトを作成する場合、カスタム ボード ライブラリ用にザイリンクス ボード記述ファイル (*.xbd) を作成し、\$XILINX_EDK\board に配置する必要があります。詳細は、次のサイトから『Platform Specification Format Reference Manual』の「Xilinx Board Description (XBD) Format」を参照してください。

http://japan.xilinx.com/ise/embedded/edk_docs.htm

次の操作

この後、ハードウェアの基礎を説明します。

- 第 3 章「[Xilinx Platform Studio の使用](#)」では、XPS (Xilinx Platform Studio) ソフトウェアを使用します。
- 第 4 章「[エンベデッド プラットフォームの操作](#)」では、XPS でプロジェクトを表示し、変更する方法を学びます。

Xilinx Platform Studio の使用

この章では、Xilinx Platform Studio (XPS) の概要を説明します。XPS を使用すると、BSB で作成したプロジェクトをカスタマイズできます。この章で XPS の概要を説明し、その後の章で XPS を使用してデザインを変更する方法を説明します。

メモ：この章を読むと、このマニュアルのこの後の章および XPS のその他の資料を理解しやすくなるので、目を通されることをお勧めします。

XPS とは

XPS は、プロジェクトの設計を支援するツールセットを含むグラフィカル ユーザー インターフェイスです。この章では、XPS ソフトウェアと最も頻繁に使用されるツールについて説明します。

XPS ソフトウェア

XPS ソフトウェアを使用すると、ザイリンクス FPGA デバイスにインプリメントする完全なエンベデッド プロセッサ システムを設計できます。次の図に、XPS のメイン ウィンドウを示します。

この章のチュートリアルでは、XPS のメイン ウィンドウの各部分にある情報およびツールを見てみます。

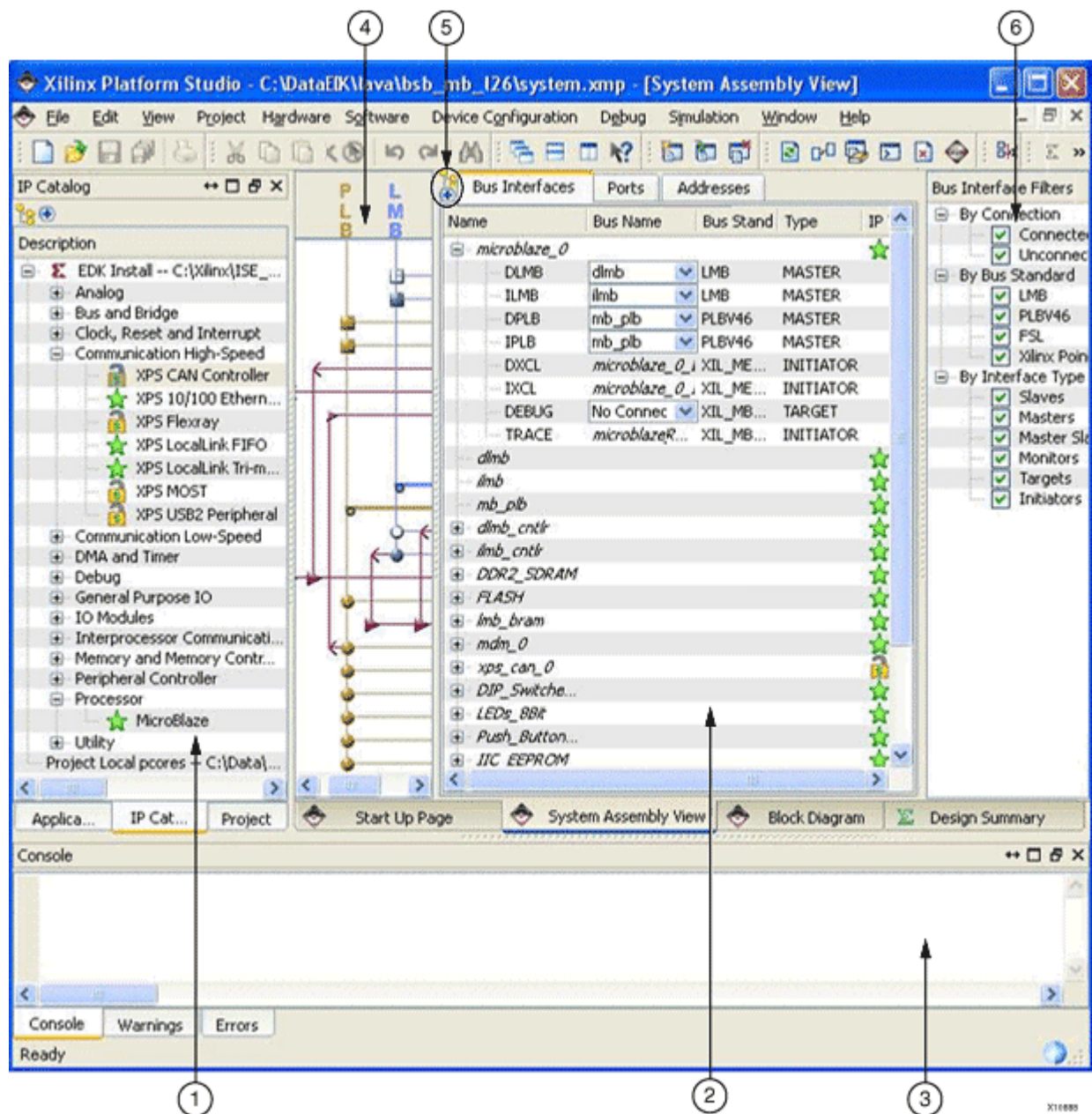


図 3-1 : XPS のメイン ウィンドウ

XPS ユーザー インターフェイスの使用

XPS のメイン ウィンドウは、次の 3 つの部分から構成されています。

- プロジェクト情報エリア (1)
- [System Assembly View] (2)
- コンソール ウィンドウ (3)

XPS のメイン ウィンドウには、次のエリアもあります。

- バス接続パネル (4)
- 表示切り替えボタン (5)
- フィルタ パネル (6)

プロジェクト情報エリア

プロジェクトの情報を表示します。ここからプロジェクトを管理します。[Project]、[Applications]、および [IP Catalog] タブがあります。

[Project] タブ

プロジェクトに関連するファイルおよび情報の一覧を表示します。ファイルおよび情報は、次のカテゴリに分類されています。

- [Project Files]

MHS (Microprocessor Hardware Specification) ファイル、MSS (Microprocessor Software Specification) ファイル、ユーザー制約ファイル (UCF)、iMPACT コマンド ファイル、インプリメンテーション オプション ファイル、BitGen オプション ファイルなど、プロジェクト特定のファイルをリストします。

- [Project Options]

デバイス、ネットリスト、インプリメンテーション、HDL、シミュレーション モデルなどのプロジェクト特定のオプションをリストします。

- [Design Summary]

エンベデッド デザインのステートを表示し、システム ファイルに簡単にアクセスできるようにします。

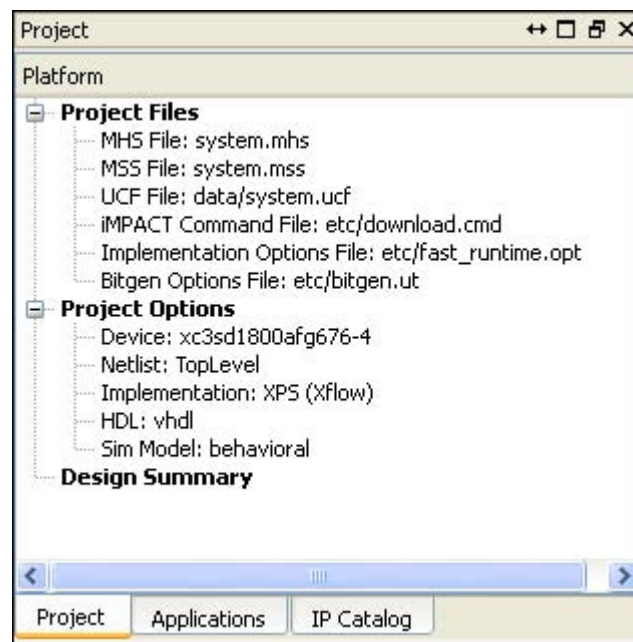


図 3-2 : プロジェクト情報エリアの [Project] タブ

[Applications] タブ

ソフトウェア アプリケーションのオプション設定と、各アプリケーション プロジェクトに関連付けられているヘッダ ファイルおよびソース ファイルの一覧をリストします (図 3-3)。このタブを選択していると、次の操作を実行できます。

- ソフトウェア アプリケーション プロジェクトの追加、プロジェクトの作成、およびブロック RAM への読み込み
- コンパイラ オプションの設定
- プロジェクトへのソース ファイルおよびヘッダ ファイルの追加

メモ： XPS でもソフトウェア プロジェクトを作成および管理できますが、ソフトウェアの開発には SDK を使用することをお勧めします。

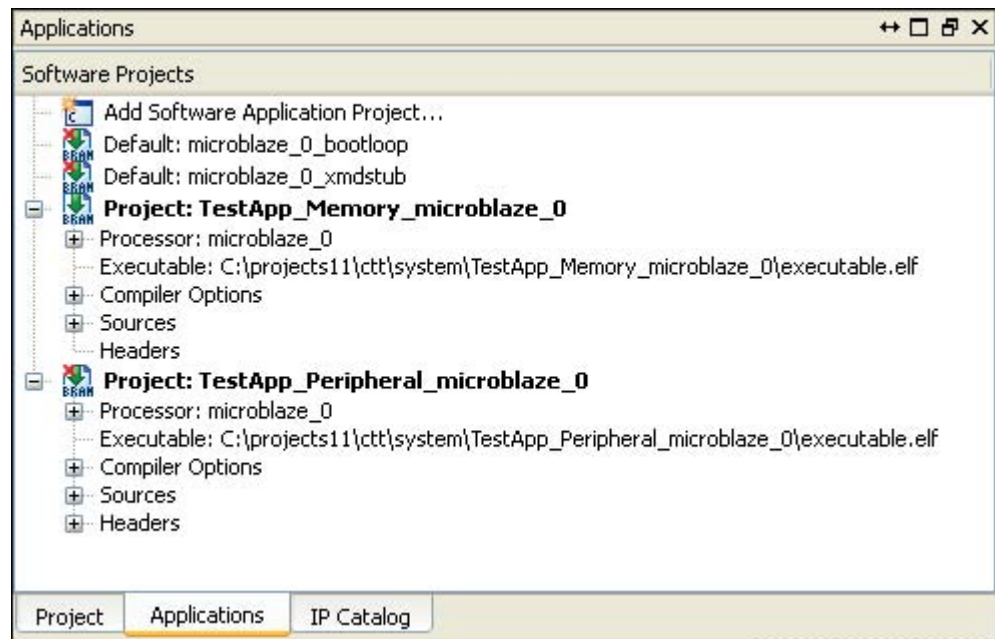


図 3-3：プロジェクト情報エリアの [Applications] タブ

[IP Catalog] タブ

IP コアの一覧を表示します。次の情報が示されます。

- コア名とライセンスのステータス
- リリース バージョンとステータス (アクティブ、早期アクセス、廃止予定)
- サポートされるプロセッサ
- 分類

バージョン変更履歴、データシート、MPD (Microprocessor Peripheral Definition) ファイルなどの IP コアの詳細情報は、[IP Catalog] タブで IP コアを右クリックして表示されるメニューからアクセスできます。デフォルトでは、IP コアはファンクションごとにグループ化されています。

メモ： IP の詳細をすべて表示するには、タブのサイズを広げる必要があります。

チュートリアル：プロジェクト情報エリア

1. XPS でプロジェクトを開き、[Project] タブをクリックします。
2. [Project Files] の下にあるアイテムのいずれかを右クリックし、[Open] をクリックします。この後のチュートリアルで、これらのファイルの一部を編集します。
3. [File] → [Close] をクリックしてファイルを閉じます。
4. [Project Options] の下にあるアイテムのいずれかを右クリックして [Project Options] をクリックし、[Project Options] ダイアログ ボックスを開きます。[Project] → [Project Options] をクリックしても、同じ操作を実行できます。
5. [Project Options] ダイアログ ボックスを閉じます。
6. [IP Catalog] タブをクリックします。
7. [IP Catalog] タブの左上に 2 つのボタンがあります。これらをクリックして、[IP Catalog] タブの表示がどのように変化するかを観察します。
8. [IP Catalog] タブに表示されているアイテムのいずれかを右クリックして、表示されるコマンドを確認します。特に、次のコマンドに注目してください。
 - ◆ [Add IP] : 選択した IP をデザインに追加します。
 - ◆ [View PDF Datasheet] : IP のデータシートを開きます。
 - ◆ [View IP Modifications (Change Log)] : IP の変更履歴を開きます。
9. [Communication Low-Speed] の横にあるプラス記号をクリックして展開表示します。
10. [XPS_UART (Lite)] ペリフェラルを右クリックし、[View PDF Datasheet] をクリックしてデータシートを開きます。

[System Assembly View]

[System Assembly View] では、システム ブロック エlement を表示およびコンフィギュレーションします。[System Assembly View] がメイン ウィンドウに表示されていない場合は、ペインの下部に表示されている [System Assembly View] タブをクリックします。

[Bus Interfaces]、[Ports]、および [Addresses] タブ

[System Assembly View] には 3 つのペインがあり、上部のタブをクリックすることにより表示を切り替えます。

- [Bus Interfaces] : デザインのバスを表示します。このビューから各バスの情報および接続を変更できます。
- [Ports] : デザインのポートを表示します。このビューから各ポートの詳細を変更できます。
- [Addresses] : デザインの各 IP インスタンスのアドレス範囲を表示します。[Generate Addresses] をクリックすると、システム アドレス マップを自動的に生成できます。

バス接続パネル

[Bus Interface] タブを選択していると、その左側にバス接続パネルが表示されます (18 ページの図 3-1 の 4)。これは、ハードウェア プラットフォーム接続のグラフィカル表示です。

縦の線はバスを表し、横の線は IP コアへのバス インターフェイスを表します。接続が可能な場合、バスと IP コアのバス インターフェイスの交差点にコネクタ シンボルが表示されます。

線とコネクタは、互換性を示すために色分けされています。コネクタ シンボルの形により、IP ブロックがバス マスタであるかバス スレーブであるかが示されます。塗りつぶされていないコネクタは接続可能であることを示し、塗りつぶされているコネクタは接続されていることを示します。コネクタ シンボルをクリックすると、接続/接続解除を切り替えることができます。

フィルタ パネル

XPS には、[System Assembly View] での [Bus Interfaces] および [Ports] タブの表示を制御するフィルタがあります。これらのフィルタは、[Bus Interfaces] または [Ports] タブを選択したときにフィルタ パネル (18 ページの図 3-1 の 6) にリストされます。これらのフィルタを使用すると、多数のバスを含むデザインで接続パネルの表示を簡潔にすることができます。

表示切り替えボタン

デザイン情報を並べ替え、デザインを編集しやすくするため、[System Assembly View] にはデータの表示方法を変更する 2 つのボタンがあります (18 ページの図 3-1 の 5)。

- [Change to Hierarchical View]/[Change to Flat View] ボタン
 - ◆ Hierarchy View では、デザイン情報がハードウェア プラットフォーム上の IP コア インスタンスごとにグループ化され、ツリー形式で表示されます。これがデフォルト表示です。
 - ◆ Flat View では、いずれかの列のアルファベット順に情報を並べ替えることができます。
- [Expand All Tree Nodes]/[Collapse All Tree Nodes] ボタン

IP に関連付けられている ネット またはバスをすべて一度に展開表示または非表示にします。

コンソール ウィンドウ

コンソール ウィンドウ (18 ページの図 3-1 の 3) には、起動したツールからのランタイムの出力が表示されます。[Console]、[Warnings]、および [Errors] の 3 つのタブがあります。

チュートリアル：[System Assembly View]

1. [System Assembly View] 上部の [Ports] タブをクリックします。
2. [External Ports] の横にあるプラス記号をクリックして展開表示し、FPGA デバイス外部の信号を表示します。
3. 信号名は [Net] 列に表示されています。RS232_Uart_1 ポートに関連する信号を見つけます。必要に応じて [Net] 列ヘッダの右端を右にドラッグして列幅を広げ、内容がすべて表示されるようにしてください。これらの信号を次の手順で参照します。[External Ports] の横にあるマイナス記号をクリックして信号を非表示にします。
4. [RS232_Uart_1] ペリフェラルの横にあるプラス記号をクリックして展開表示します。

ネット名が外部信号の名前と対応していることを確認します。UART からの RX および TX ネットは、外部ポートと名前により関連付けられています。

5. [RS232_Uart_1] ペリフェラルを右クリックし、[Configure IP] をクリックして IP コンフィギュレーション ダイアログ ボックスを開きます。システムのすべてのペリフェラルに対して同様のコンフィギュレーション ダイアログ ボックスを開くことができます。
 - a. パラメータ名の上にカーソルを置くと何が起こるかを確認します。
 - b. ダイアログ ボックスの上部に、3 つのボタンと 3 つのタブがあることを確認します。
 - c. 終了したら、このダイアログ ボックスを閉じます。
6. [Change to Hierarchical View]/[Change to Flat View] ボタンをクリックして表示がどのように変化するかを観察します。

[Start Up Page]

[Start Up Page] には、リリース情報へのリンク、ソフトウェアおよびハードウェア デザイン フローへのリンクなど、XPS のご使用のバージョンに関する情報が表示されます。また、EDK のマニュアルにアクセスするためのタブもあります。少し時間をとって [Start Up Page] の内容を確認してください。

チュートリアル： [Start Up Page] の内容の確認

このチュートリアルでは、[Start Up Page] の内容を確認します。

1. XPS のメイン ウィンドウで [Start Up Page] タブをクリックします。[Start Up Page] タブが表示されていない場合は、[Help] → [View Start Up Page] をクリックします。
2. [New This Release] タブで [Redesigned interface improves project handoff and portability between hardware and software teams] リンクをクリックします。以前のバージョンの XPS を使用していた場合、新機能について学ぶことができます。ほとんどの新機能は、このマニュアルのチュートリアルで扱います。

XPS ツール

XPS には、ソフトウェア インターフェイスに加え、エンベデッド プロセッサ システムのハードウェアおよびソフトウェアのコンポーネントを開発するのに必要なツールも含まれています。

- **Base System Builder (BSB) ウィザード**：新規プロジェクトを作成します。BSB ウィザードは XPS の起動時に表示される [Xilinx Platform Studio] ダイアログボックスから起動できます。このダイアログ ボックスは、[File] → [New Project] をクリックしても開きます。
- **Platform Generator (Platgen)**：エンベデッド プロセッサ システムのハードウェア プラットフォームを生成します。Platgen を起動するには、[Hardware] → [Generate Netlist] をクリックします。
- **Simulation Model Generator (Simgen)**：オリジナルのエンベデッド ハードウェア デザイン (ビヘイビア レベル) または完成した FPGA インプリメンテーション (タイミング レベル) のいずれかに基づいて、エンベデッド ハードウェア システムのシミュレーション モデルを生成します。Simgen を起動するには、[Simulation] → [Generate Simulation HDL Files] をクリックします。
- **Create and Import Peripheral (CIP) Wizard**：独自のペリフェラルを作成し、EDK 準拠のレポジトリまたは XPS プロジェクトにインポートします。このウィザードを起動するには、[Hardware] → [Create or Import Peripheral] をクリックします。

- **Library Generator (Libgen)** : エンベデッド プロセッサ システム用にライブラリ、デバイスドライバ、ファイル システム、および割り込みハンドラを設定します。**Libgen** を起動するには、[Software] → [Generate Libraries and BSPs] をクリックします。

チュートリアル : XPS ツールにアクセスするメニュー コマンド

上記の XPS ツールには、[Hardware]、[Software]、[Simulation] メニューからアクセスできます。各メニューをクリックして、どのようなコマンドがあるかを確認してください。

XPS のディレクトリ構造

前の章で開始したチュートリアル デザインでは、**BSB** ウィザードによりプロジェクト ディレクトリ構造が自動的に設定され、単純で完全なプロジェクトが作成されました。ただし、ツールでどのような処理が実行されているのを理解しなければ、**BSB** ウィザードによるプラットフォームのコンフィギュレーションで短縮された時間が無駄になります。**BSB** ウィザードで作成されたディレクトリ構造が、プロジェクトの開発プロセスでどのように有益であるかを見えます。

メモ : ファイルは、プロジェクト ファイルを作成したディレクトリに保存されています。

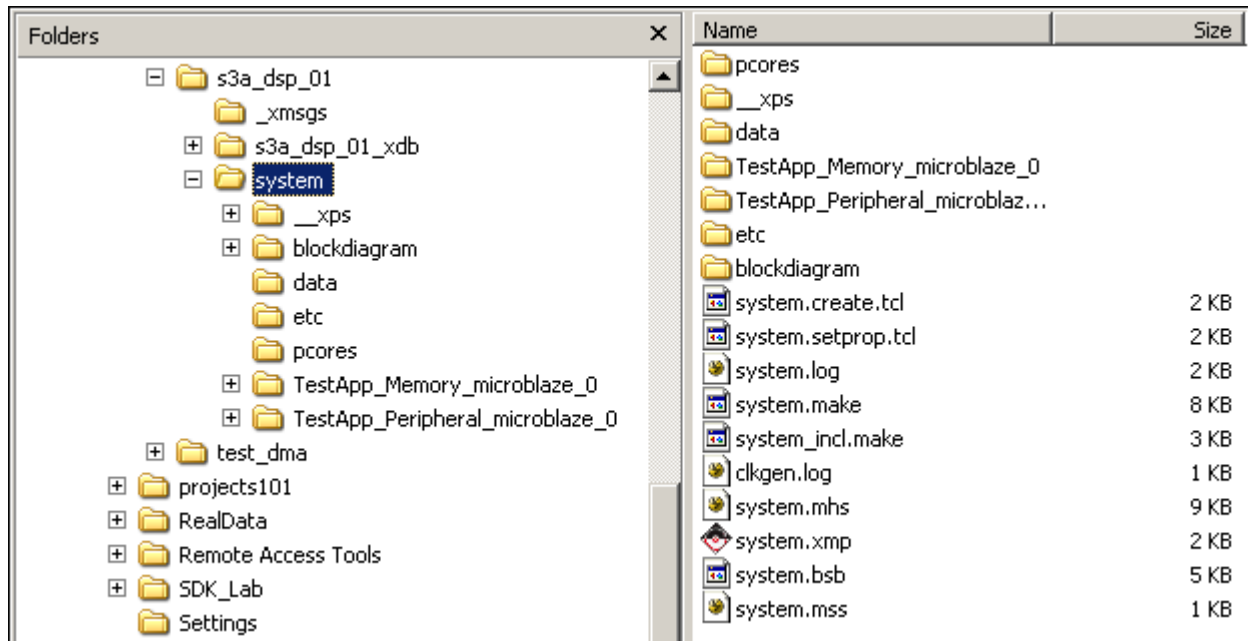


図 3-4 : ファイル ディレクトリ構造

ディレクトリ

BSB ウィザードでは、レポジトリ検索パスにいくつかのディレクトリが自動的に作成されます。エンベデッド システム ソースの名前でプロジェクト ディレクトリが作成され、そのディレクトリにこれらのサブディレクトリが含まれます。これらのディレクトリは、次のとおりです (図 3-4)。

__xps	XPS およびその他のツールで内部プロジェクト管理用に生成される中間ファイルが含まれます。このディレクトリは使用できません。
blockdiagram	ブロック ダイアグラムに関連するファイルが含まれます。
data	ユーザー制約ファイル (UCF) が含まれます。このファイルの詳細と使用方法は、次のサイトから ISE の UCF に関する資料を参照してください。 http://japan.xilinx.com/support/documentation/sw_manuals/xilinx11/manuals.pdf
etc	ツールの実行に使用されるオプションを設定するファイルが含まれます。BSB ウィザード以外で操作を実行していないので、このディレクトリは空です。
pcores	カスタム ハードウェア ペリフェラルで使用されます。

次の 2 つのディレクトリには、BSB ウィザードで生成されたファイルが含まれます。

- TestApp_Memory_microblaze_0
- TestApp_Peripheral_microblaze_1

これらのディレクトリには、テスト アプリケーション C ソース コード、ヘッダ ファイル、およびリンカ スクリプトが含まれます。これらのファイルを使用することもできますが、このマニュアルでは SDK で作成したサンプル アプリケーションを使用します。これについては、この後の章で説明します。

メイン プロジェクト ディレクトリには、次のファイルも含まれています。

system.xmp	最上位プロジェクト ファイルです。XPS はこのファイルを読み込み、ユーザー インターフェイスにその内容を図で表示します。
system.mhs	マイクロプロセッサ ハードウェア仕様で、システム エlement、パラメータ、および接続をテキスト形式で記述します。MHS ファイルは、プロジェクトのハードウェア部分の基盤となります。
system.mss	デザインのソフトウェア部分を表すマイクロプロセッサ ソフトウェア仕様で、システム エlement、ペリフェラルと関連付けられているソフトウェア パラメータをテキスト形式で記述します。MSS ファイルは、プロジェクトのソフトウェア部分の基盤となります。

MHS と MSS ファイルは XPS で生成される主なファイルで、ハードウェアおよびソフトウェア システム全体がこれら 2 つのファイルで表されます。

チュートリアル：ディレクトリ構造

このチュートリアルでは、XPS のディレクトリ構造を見てみます。

1. Windows エクスプローラなどのファイル エクスプローラ ユーティリティで、プロジェクトの最上位ディレクトリに移動します。
2. さまざまなサブディレクトリを開き、どのようなファイルがあるかを確認します。

次の操作

これで、XPS の GUI のナビゲーション方法を理解でき、前の章で開始したプロジェクトを操作する準備ができました。第 4 章「[エンベデッド プラットフォームの操作](#)」に進みます。

エンベデッド プラットフォームの操作

ハードウェア プラットフォームの概要

エンベデッド ハードウェア プラットフォームは、1 つまたは複数のプロセッサ、さまざまなペリフェラル、およびメモリ ブロックで構成されています。これらの IP ブロックは、インターコネクト ネットワークを使用して接続されます。また、システム外部と接続するために追加ポートが使用されます。各プロセッサまたはペリフェラル コアの動作は、カスタマイズできます。インプリメンテーション パラメータはオプションの機能を制御し、FPGA に最終的に何をインプリメントするかを指定します。インプリメンテーション パラメータは、システムのアドレスも定義します。

XPS でのハードウェア プラットフォーム開発

MHS ファイルについて

Xilinx® Platform Studio (XPS) では、対話型の開発環境を使用してハードウェア プラットフォームを詳細に設定できます。ハードウェア プラットフォーム記述は、MHS (Microprocessor Hardware Specification) ファイルで管理されます。MHS ファイルは、編集が簡単なテキスト ファイルで、エンベデッド システムのハードウェア コンポーネントを記述した主要ソース ファイルです。XPS では MHS ソース ファイルが HDL ネットリストに合成され、このネットリストがインプリメンテーション プロセスで処理されます。

MHS ファイルは、デザイン プロセスに必要不可欠です。すべてのペリフェラル インスタンスとそのパラメータが含まれます。バス アーキテクチャ、ペリフェラル、プロセッサ、接続、アドレス空間などを含む、エンベデッド プロセッサ システムのコンフィギュレーションを定義するファイルです。MHS ファイルの詳細は、次のサイトから『Platform Specification Format Reference Manual』の「Microprocessor Hardware Specification (MHS)」の章を参照してください。

http://japan.xilinx.com/ise/embedded/edk_docs.htm

チュートリアル：MHS ファイル

このチュートリアルでは、BSB ウィザードを実行したときに作成された MHS ファイルを見ます。

1. XPS ソフトウェアのプロジェクト情報エリアで、[Project] タブをクリックします。
2. [Project Files] の下にある [MHS File: system.mhs] をダブルクリックして開きます。
3. [Edit] → [Find] をクリックし、表示された検索ツールバーを使用して `system.mhs` で「`xps_uartlite`」を検索します。

MHS ファイルでペリフェラル、ポート、およびパラメータがどのように設定されているかを見ます。

4. デザインのその他の IP コアも見えます。終了したら、`system.mhs` ファイルを閉じます。

[System Assembly View] でのハードウェア プラットフォーム

[System Assembly View] には、すべてのハードウェア プラットフォーム IP インスタンスがツリービューの表形式で表示されます。

この表示は、カスタマイズしたり、並び替えたり、フィルタをかけたりして、見やすくできます。IP エlement、ポート、プロパティ、およびパラメータは [System Assembly View] で設定でき、MHS ファイルに直接記述されます。

ポート名の変更およびパラメータの設定は、Enter キーを押すか [OK] をクリックすると反映され、MHS ファイルのハードウェア データベースにシステム変更が自動的に記述されます。MHS ファイルを編集するには、[System Assembly View] を使用することをお勧めします。

メモ：IP の追加、削除、カスタマイズについては、第 7 章「独自の IP の作成」で説明します。

ハードウェア プラットフォームの生成

ハードウェア プラットフォームの生成には、3 つの操作があります。まず XPS でネットリストを生成し、ISE® ツールでデザインをインプリメント (FPGA ロジックにマップ) した後、最後にインプリメントされたデザインを FPGA にダウンロード可能なビットストリームに変換します。

ネットリストの生成

XPS でネットリストを生成すると、プラットフォーム構築ツールである Platgen が起動し、次の処理が実行されます。

- ◆ デザインのプラットフォーム コンフィギュレーションである MHS ファイルが読み込まれます。
- ◆ MHS ファイルの HDL 記述が生成され、system.[vhd|v] および system_stub.[vhd|v] に記述されます。system ファイルは、MHS 記述が HDL フォーマットで記述されたものです。system_stub ファイルは最上位 HDL テンプレート ファイルで、プロセッサシステムをコンポーネントとして HDL ベース デザインにインスタンスエートする場合に使用します。
- ◆ XST (Xilinx Synthesis Technology) を使用してデザインが合成されます。
- ◆ ネットリスト ファイルが生成されます。

Platgen の詳細は、次のサイトから『エンベデッド システム ツール リファレンス マニュアル』の「Platform Generator (Platgen)」の章を参照してください。

http://japan.xilinx.com/ise/embedded/edk_docs.htm

ハードウェア プラットフォームのエクスポート

エンベデッド 開発キット (EDK) では、ハードウェア プラットフォームは XPS で設計し、ソフトウェアはソフトウェア開発キット (SDK) で開発およびデバッグします。ハードウェア プラットフォームに関する情報は SDK で必要なので、system.xml という ファイルをエクスポートします。

system.xml ファイル

system.xml ファイルには、設計したハードウェア プラットフォームでソフトウェア開発およびデバッグを行うために SDK で必要な情報が含まれます。

チュートリアル：ハードウェア プラットフォームの SDK へのエクスポート

1. XPS ソフトウェアで、[Project] → [Export Hardware Design to SDK] をクリックします。
2. デフォルトの保存ディレクトリを使用することをお勧めします。デフォルトの保存ディレクトリを使用する場合、system.xml ファイルへのリポジトリ検索パスはプロジェクト ディレクトリの system\SDK\SDK_Export\hw\... になります。

メモ：XPS で使用される XML ファイルはほかにもあるので、使用する XML ファイルの場所を知っておくことが重要です。

3. [Export Only] をクリックします。この後の章のチュートリアルで、SDK を実行します。

実行された処理

エクスポート処理を詳細に理解しておくことは、特に複数のハードウェア バージョンを管理している場合に重要です。

[Export Only] をクリックすると、SDK で使用される多数のファイルが作成されます。XML ファイルに加え、ソフトウェア ドライバおよびハードウェア IP の資料も含まれており、SDK から必要な情報にアクセスできます。

もう 1 つのボタン [Export & Launch SDK] をクリックすると、既存の XML ファイルが上書きされます。エクスポート ディレクトリにビットストリーム (BIT) ファイルおよびブロック メモリ マップ (BMM) ファイルが存在する場合はそれらは消去され、エクスポートするプロジェクトに BIT および BMM が含まれる場合は、これらがエクスポート ディレクトリに保存されます。これにより、エクスポート ディレクトリに最新のハードウェア ファイルのみが含まれるようになります。

チュートリアル：ビットストリームの生成

XPS での処理が正常に完了したら、ISE Project Navigator を使用してデザインをインプリメントし、ビットストリームを生成できます。

ISE Project
Navigator を使用
したデザインの
インプリメント

Project Navigator は生成されたネットリストとユーザー制約ファイル (UCF) を読み込み、ハードウェア デザインを含む BIT ファイルを生成します。コンパイルされた C コードはビットストリームには含まれず、後で SDK を使用して追加します。

1. Project Navigator のメイン ウィンドウを見ます。[Design] パネルの表示は、次のようになっているはずです。

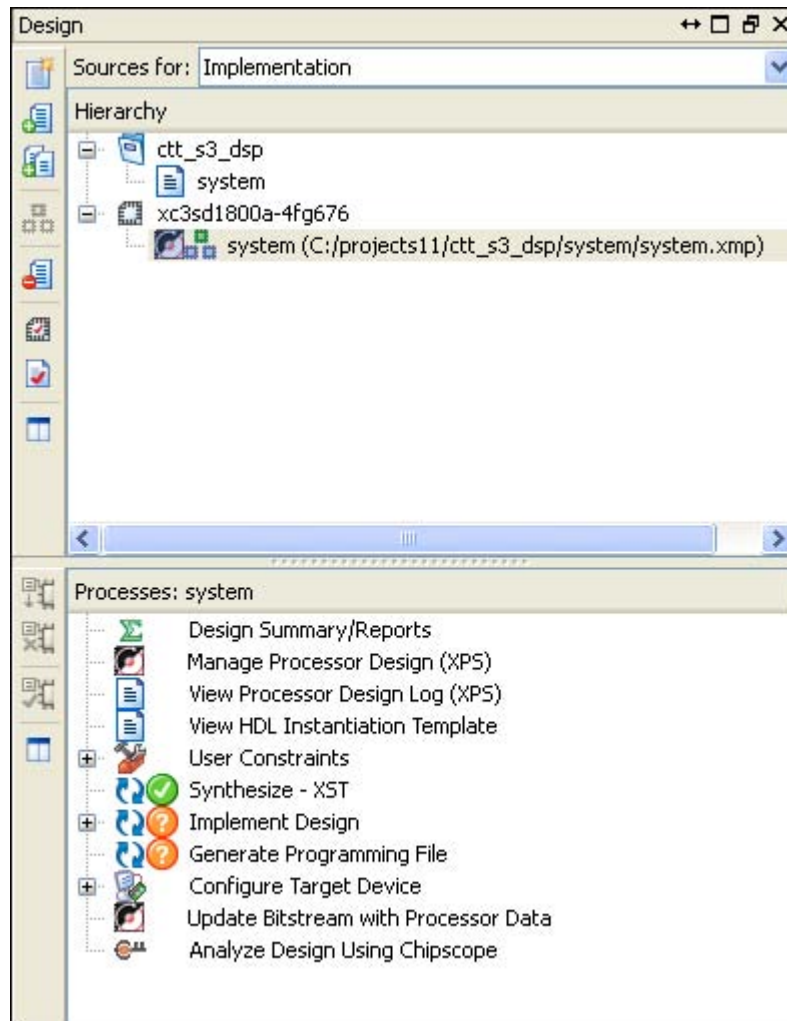


図 4-1 : ISE Project Navigator の [Design] パネル

ビットストリームの
生成および UCF
ファイルの作成

ビットストリームを生成する前に、ピン配置や実行速度など、ISE 配置配線ツール (PAR) で使用されるデザインに関する情報を追加する必要があります。

これらの情報は、UCF ファイルに含まれます。BSB を実行すると、UCF ファイルが生成されます。

2. Project Navigator で [Project] → [Add Source] をクリックし、プロジェクト ディレクトリの system\data にある system.ucf ファイルを追加します。
この手順は、XMP ソースが ISE プロジェクトの最上位デザインである場合にのみ必要です。XMP を VHDL または Verilog ファイルにインスタンス化の場合は、ISE により EDK UCF ファイルが管理されます。
3. [Adding Source Files] ダイアログ ボックスが表示され、UCF ファイル処理の進行状況が表示されます。ファイル処理が完了したら、[OK] をクリックします。

[Design] パネルの [Hierarchy] ペインの表示は、次のようになります。

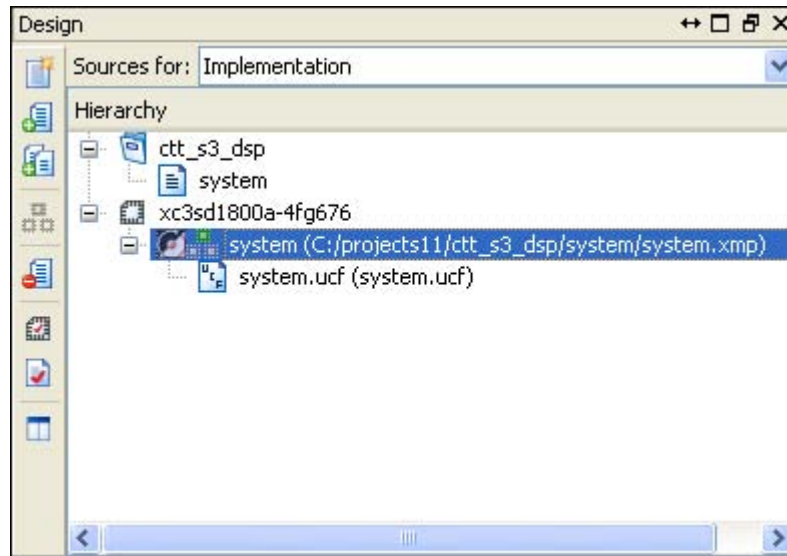


図 4-2 : system.xmp と system.ucf ファイル

system.ucf ファイルが追加され、エンベデッド システム デザイン (system.xmp) に関連付けられています。

EDK system.bit およ
び BmmFile_bd.bmm
ファイル

4. [Design] パネルの [Processes] ペインで [Generate Programming File] をダブルクリックします。この処理には数分かかります。処理が完了すると、[Console] パネルに「Process "Generate Programming File" completed successfully」というメッセージが表示されます。

生成されたビット ストリームの名前は system.bit です。これ以外に edkBmmFile_bd.bmm というファイルも生成されます。このファイルは、SDK でターゲット ボードにメモリを読み込むために使用されます。

これらのファイルとそのディレクトリ (ハードウェア プロジェクトのルート ディレクトリ) を覚えておきます。これらのファイルは、この後の章で使用します。

次の操作

SDK を使用して、プロジェクトのソフトウェアを開発します。次の 2 章で、エンベデッド ソフトウェア設計の基礎を説明します。

ソフトウェア開発キット

ザイリックス ソフトウェア開発キット (SDK) は、エンベデッド ソフトウェア アプリケーション プロジェクトを開発するためのツールで、Eclipse オープン ソース標準に基づく独自のソフトウェア です。SDK は XPS に補足的に使用するプログラムで、XPS でデザインに組み込んだペリフェラル およびプロセッサ エレメントで使用するソフトウェアを開発します。

SDK について

ハードウェア プラットフォーム、ソフトウェア プラットフォーム、ソフトウェア プロジェクト、パースペクティブ、ビューなどの SDK で使用される用語を理解しておく必要があります。

SDK の用語

ハードウェア プラットフォームとは、XPS で作成され、XML ファイルの形でエクスポートされた エンベデッド ハードウェア デザインです。XML ファイルを SDK にインポートするということは、ハードウェア プラットフォームをインポートするということです。1 つの SDK プロジェクトに対して、ハードウェア プラットフォームを 1 つのみ含むことができます。

ハードウェア プラットフォームをインポートしたら、ソフトウェア プラットフォームを作成します。ソフトウェア プラットフォームは、アプリケーション ソフトウェア スタックの最下位層を構成するソフトウェア ドライバおよび OS をまとめたものです。ソフトウェア アプリケーションは、アプリケーション プログラム インターフェイス (API) を使用して、指定のソフトウェア プラットフォームにリンクするか、ソフトウェア プラットフォーム上で実行する必要があります。そのため、SDK でソフトウェア アプリケーションを作成および使用する前に、ソフトウェア プラットフォーム プロジェクトを作成する必要があります。SDK には、次の 2 つのソフトウェア プラットフォーム タイプが含まれます。

SDK のソフトウェア プラットフォーム タイプ

- **standalone** : 標準入力/出力、プロセッサ ハードウェア機能へのアクセスなどの基本的な機能を提供する単純なセミホスト型シングル スレッド環境
- **xilkernel** : スケジューリング、スレッド、同期化、メッセージ パッシング、タイマなどの POSIX 型サービスを提供する単純な軽量カーネル

1 つの SDK プロジェクトには、複数のソフトウェア プラットフォームを含めることができます。たとえば、standalone 用にセットアップされたソフトウェア プラットフォームと xilkernel 用にセットアップされたソフトウェア プラットフォームを含めることができます。

ソフトウェア プラットフォームには、ソフトウェア プロジェクトが含まれます。ソフトウェア プロジェクトはアプリケーションです。1 つのソフトウェア プラットフォームに複数のソフトウェア プロジェクトを含むことができます。

パースペクティブとビュー

SDK は、実行する操作によってソフトウェアのインターフェイスが変化します。C または C++ コードを開発する場合はコード開発用のウィンドウが表示され、ハードウェア上でコードをデバッグする場合はデバッグ用のウィンドウが表示されます。コードをプロファイルする際は、また別のウィンドウが表示されます (このマニュアルでは説明しません)。EDK でのプロファイルの詳細は、『[EDK Profiling User Guide](#)』を参照してください。これらの異なるウィンドウ セットの表示をパースペクティブと呼び、各パースペクティブの各ウィンドウをビューと呼びます。

ワークスペースの左上にあるタブをクリックするだけで、簡単にパースペクティブを切り替えることができます。このパースペクティブが、SDK にさらなる機能と柔軟性を与えています。

チュートリアル：ハードウェア プラットフォームのインポート

1. デスクトップで [Xilinx SDK 11] アイコンをダブルクリックするか、[スタート] → [プログラム] → [Xilinx ISE Design Suite 11] → [EDK] → [Xilinx Software Development Kit] をクリックします。
2. ワークスペースを指定します。場所はどこでもかまいません。プロジェクトのルート ディレクトリか SDK での作業を実行するディレクトリを指定してください。

注意：パスにスペースが含まれていないことを確認してください。

3. 前の章でエクスポートしたハードウェア デザインを指定します。これは XML ファイル (system.xml) で、コードが実行されるエンベデッド プラットフォームが記述されています。
4. 次のメッセージが表示されます。[OK] をクリックする前に、このメッセージの内容を確認してください。SDK でのプロジェクト操作について説明されています。メッセージの内容は、次のとおりです。

ハードウェア デザインが読み込まれたので、ソフトウェア プロジェクトを作成できます。C または C++ アプリケーション プロジェクトを作成する前に、ソフトウェア プラットフォーム プロジェクトを作成する必要があります。ソフトウェア プラットフォーム プロジェクトを作成するには、[File] メニューの下にある [New] ボタンをクリックし、[Software Platform] を選択します。

ザイリンクス ソフトウェア開発キット (SDK) のチュートリアルは、[Help] → [Cheat Sheets] をクリックしてください。SDK のオンライン ヘルプは、[Help] → [Help Contents] をクリックするか、[Welcome] ページで [SDK] アイコンをクリックしてください。



図 5-1 : [Next Steps] ダイアログ ボックス

SDK が開き、次の図に示すように [C/C++ Projects] ビューが表示されます。microblaze_0 は、ハードウェア プラットフォームの MicroBlaze™ プロセッサを表します。

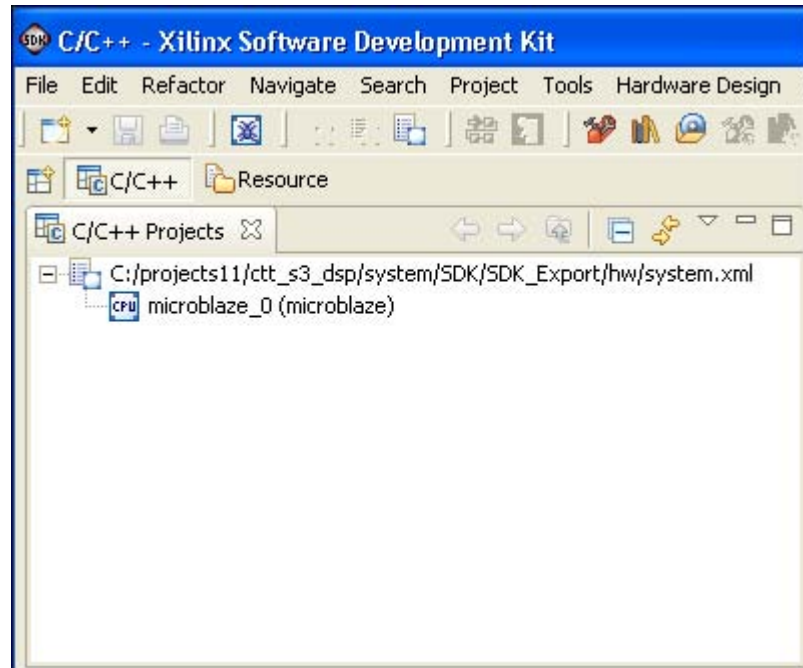


図 5-2 : ハードウェア プラットフォームを含む [C/C++ Projects] ビュー

チュートリアル：ソフトウェア プラットフォームの作成

作成したハードウェア プラットフォームに対応するソフトウェア プラットフォームを作成します。

1. [File] → [New] → [Project] をクリックし、[Software Platform] を選択します。

1 つのエンベデッド デザインに、複数のソフトウェア プラットフォームを含めることができます。このチュートリアルで作成するのは、standalone プロジェクトです。

メモ：ハードウェア プラットフォームに複数のマイクロプロセッサが含まれる場合、各マイクロプロセッサが [Processor] ドロップダウン リストに表示されます。

2. 次のように設定します。

- ◆ [Project name] : SW_Platform_1
- ◆ [Processor] : [microblaze_0 (microblaze)]
- ◆ [Platform Type] : [standalone]
- ◆ [Project Location] : [Use default] をオン

3. [Finish] をクリックします。

実行された処理

SDK により、ハードウェア仕様ファイル (system.xml) と選択したソフトウェア プラットフォームのタイプが読み込まれ、ハードウェア プラットフォームのコンポーネントに対応するライブラリがコンパイルされます。このプロセスのログを [Console] ビューで確認できます。

ソフトウェア プラットフォームのドライバとライブラリのリストを表示するには、[SW_Platform_1] を右クリックして [Software Platform Settings] をクリックします。左側のボックスで [Software Platform]、[OS and Libraries]、[Driver] を選択すると、それぞれソフトウェア プラットフォーム、OS とライブラリ、ドライバが表示されます。このダイアログ ボックスでコンフィギュレーションを変更できます。

次の図に示すように、[C/C++ Projects] タブで [SW_Platform_1] の下にある [microblaze_0] を展開表示します。code、include、lib、および libsrc フォルダに、エンベデッド デザインのハードウェアのライブラリすべてが含まれます。

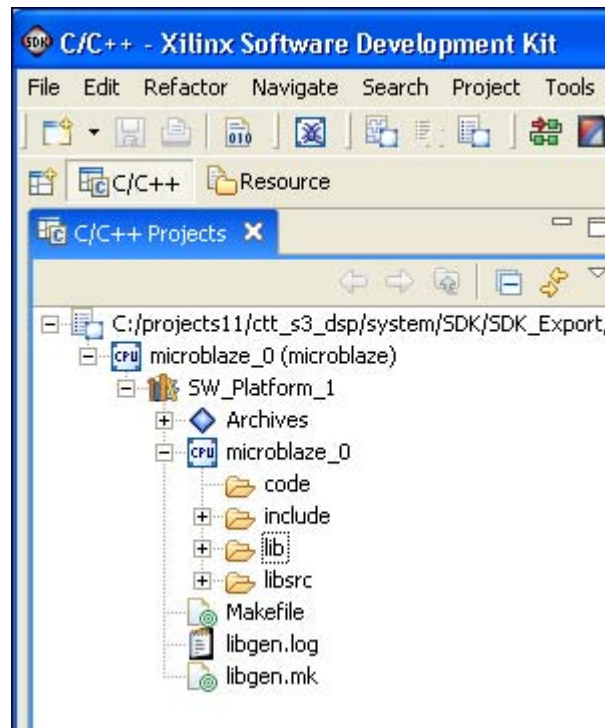


図 5-3：ソフトウェア ディレクトリ ツリー

いずれかのファイルをダブルクリックすると、SDK エディタ エリアに表示されます。

チュートリアル：ソフトウェア環境の設定

前の章で作成したエンベデッド プロセッサ デザインをターゲットとするコードを記述する環境を設定しました。

このチュートリアルでは、Software_Platform_1 用の Managed C アプリケーション プロジェクトを作成します。

1. [File] → [New] → [Managed Make C Application Project] をクリックします。
使用可能なサンプルアプリケーションが複数あります。まず **Hello World** サンプルアプリケーションを使用します。
2. [Create a new Managed Make C project] ページで次のように設定します。
 - ◆ [Project Name] : hello_1
 - ◆ [Software Platform] : [SW_Platform_1]
 - ◆ [Project Location] : [Use Default Location for Project] をオン
 - ◆ [Sample Applications] : [Hello World] を選択
3. [Next] をクリックします。
4. [Select a build configuration] ページで次のように設定します。
 - ◆ [Project Type] : [Xilinx MicroBlaze Executable]
 - ◆ [Configuration] : すべてのチェック ボックス ([Debug]、[Release]、および [Profile]) をオン
5. [Finish] をクリックします。

hello_1 サンプルアプリケーションが自動的に構築され、ターゲット ハードウェアにダウンロード可能な ELF ファイルが生成されます。

[C/C++ Projects] ビューに、ソフトウェアプラットフォームとソフトウェアプロジェクトに関する情報が表示されます。どのような情報が表示されているかを確認してください。このビューに、関連するプロジェクトの管理情報が表示されます。

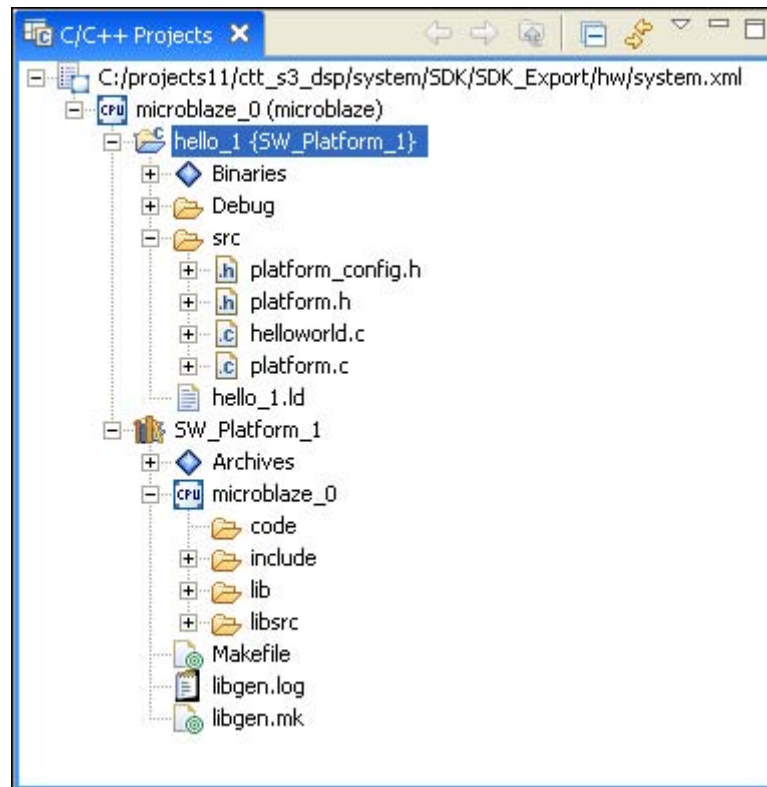


図 5-4 : [C/C++ Projects] ビューに表示されるプロジェクト

6. [hello_1] ソフトウェア プロジェクトの下にある [src] フォルダを展開表示します。
[hello_1 {SW_Platform_1}] という表示により、このソフトウェア プロジェクトが SW_Platform_1 ソフトウェア プラットフォーム用に構築されていることがわかります。
7. [helloworld.c] をダブルクリックします。ファイルが **SDK** エディタ ウィンドウに開きます。必要に応じて、サンプルコードを変更したり、独自のコードを作成したりできます。

このプロジェクト用に hello_1.ld リンカ スクリプトも生成されています。リンカ スクリプトは、ソフトウェアコードを読み込むハードウェア システム メモリ の位置を指定するために必要です。

これで、ソフトウェア プロジェクトを編集、コンパイル、構築するフレームワークが完了しました。次のチュートリアルでデバッグを実行します。

チュートリアル：SDK でのデバッグ

デバッグとは、C コードをターゲット ハードウェアにダウンロードして実行し、コードが正しく実行されるかどうかを評価するプロセスです。FPGA では、FPGA にデザインを読み込むため、FPGA をビットストリームでコンフィギュレーションする手順が追加で必要です。この場合、デザインはエンベデッド プロセッサ システムです。

1. システムをデバッグ用に設定します。
 - a. USB プログラム ケーブルを接続し、RS232 ケーブルでデモ ボードとコンピュータを接続します。デモ ボードがオンになるので、USB ケーブルの LED は緑色に点灯するはずです。
 - b. PC でハイパーターミナル (またはその他のターミナル エミュレーション プログラム) を開き、表示を 9600 ボー、8 ビット データ、1 ストップ ビットに設定します。
2. [Tools] → [Program FPGA] をクリックします。
3. 開いたダイアログ ボックスで、次のように設定します。
 - ◆ [Bit File]: プロジェクト フォルダに含まれる system.bit ファイルを選択
 - ◆ [Bmm File]: プロジェクト フォルダに含まれる edkBmmFile_bd.bmm ファイルを選択
 - ◆ [Specify the ELF file to be initialized to each processor's BRAM memory]: [Processor] が [microblaze_0]、[Type] が [microblaze]、[Initialization ELF] が [BootLoop]

メモ：ハードウェア プロジェクトの設定方法によって、ファイル名が異なる場合があります。

FPGA をビットストリームでコンフィギュレーション

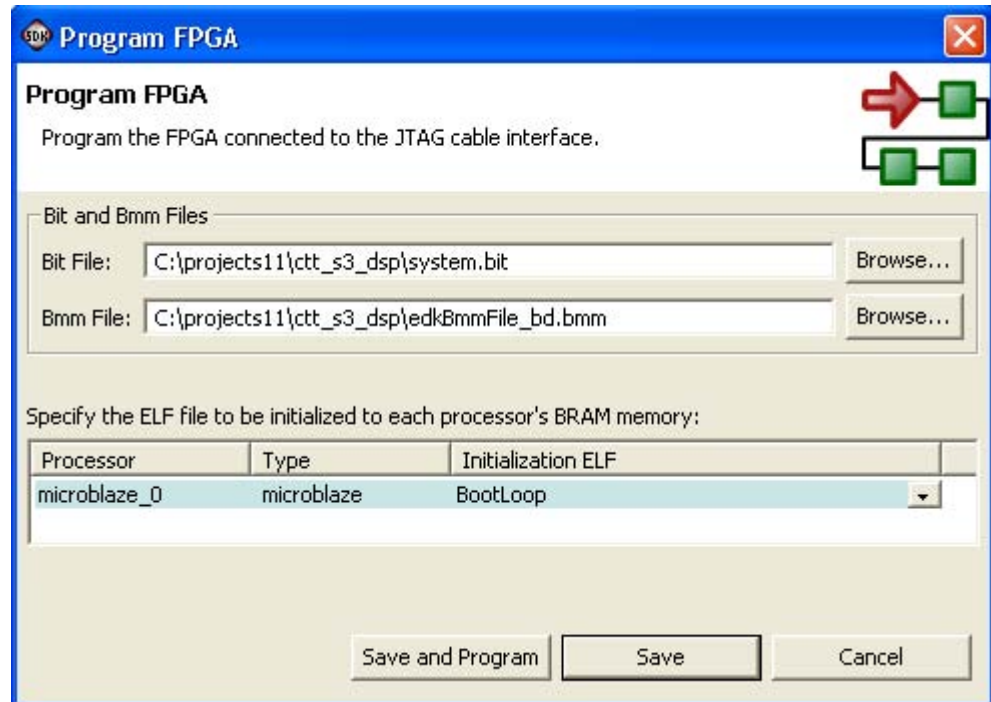



図 5-5 : [Program FPGA] ダイアログ ボックス

4. [Save and Program] をクリックします。FPGA がビットストリームで正しくプログラムされたことを示すメッセージが表示されます。


FPGA ビットストリームとブートループのダウンロード

ビットストリームが FPGA にダウンロードされ、マイクロプロセッサが **bootloop** という自身に分岐する命令で初期化されます。ブートループはプロセッサを既知のステートに保持し、別のプログラムがダウンロードされて実行されるかデバッグされるのを待ちます。

FPGA のオンチップ RAM に ELF ファイルを含める場合は、[Program FPGA] ダイアログボックスの [Initialization ELF] フィールドで ELF ファイルを指定します。ここでは、ELF ファイルを別の手順でダウンロードします。

5. [hello_1.elf] を右クリックし、[Debug As] → [Debug on Hardware] をクリックします。パースペクティブが [Debug] に変更されました。
6. [Debug] パースペクティブで、C コードの最初の実行行がハイライトされ、[Debug] ビューに Thread[0] で main() 関数が自動的に挿入されたブレークポイントのため 28 行目で停止していることが示されます。
7. [Resume] ボタン  をクリックしてコードを実行します。

[Debug] パースペクティブ

8. [Terminate] ボタン  をクリックしてデバッグ セッションを停止します。
9. ターミナル ウィンドウで出力を確認します。終了したら、SDK を閉じます。

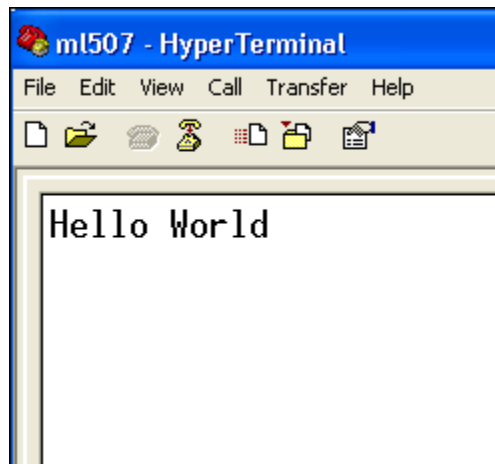


図 5-6：ターミナルの表示

実行された処理

SDK で実行したコードによりターミナル ウィンドウに「Hello World」と表示され、SDK を使用してソフトウェアが簡単に実行できたことがわかります。

次の操作

この章では、SDK プロジェクトを設定し、ターゲット ボードにビットストリームをダウンロードしてコードを実行しました。

次の章ではさらに SDK を使用して、新規ソフトウェア プロジェクトを作成、ソース コード管理を使用、およびデバッグを実行します。

SDK での編集およびデバッグ

ザイリンクス ソフトウェア開発キット (SDK) は、ソフトウェア開発プロセス全体で使用できます。ソフトウェア プロジェクトの作成、編集、および構築、ソフトウェアのターゲット ハードウェア上でのデバッグ、ターゲット ハードウェア上でのソフトウェアのプロファイル、ソフトウェアのリリース、ソフトウェアのフラッシュ メモリへのプログラムなど、ソフトウェア開発の作業はすべて SDK で実行できます。

ドライバ

SDK での作業を開始する前に、ザイリンクスが提供する低レベルのソフトウェア ドライバについて知っておく必要があります。これらのドライバは、次のディレクトリにあります。

```
<Xilinx Install>\EDK\sw\XilinxProcessorIPLib\drivers
```

各ペリフェラルドライバに 1 つずつディレクトリがあり、ハードウェアの各部分に対応するドライバを EDK で使用できます。このディレクトリには、少なくとも次のものが含まれています。

- ドライバのソース コード
- ドライバに関する HTML 資料
- ドライバの使用例

次に進む前に、これらの重要な情報を確認してください。

SDK のパースペクティブとウィンドウのタイプ

前の章で示したように、SDK にはパースペクティブという異なる画面表示があります。

前の章では、[C/C++] パースペクティブと [Debug] パースペクティブで作業を実行しました。これ以外に、もう 1 つ [Profiling] パースペクティブがあります。どのパースペクティブで作業していても、SDK のウィンドウ システムが優れていることがわかります。パースペクティブには、編集ウィンドウと情報ウィンドウの 2 種類のウィンドウがあります。

C または C++ ソース コードを含む編集ウィンドウは言語特有であり、構文が認識されます。編集ウィンドウでアイテムを右クリックすると、そのアイテムに対して実行可能な操作のリストが表示されます。

情報ウィンドウは特に柔軟性があり、必要に応じていくつでも開くことができます。情報ウィンドウには複数のビューがあることがあり、ウィンドウ上部のタブをクリックすることにより切り替えることができます。

[Debug] パースペクティブのビューには [Disassembly]、[Registers]、[Memory]、および [Breakpoints] などがあります。これらのビューは、自由に移動、ドラッグ、および組み合わせることができます。

[Debug] パースペクティブのビュー

[C/C++] または [Debug] パースペクティブで、いずれかのビューをクリックして別のウィンドウに移動してみてください。ビューが移動先のウィンドウで表示されます。選択したパースペクティブで使用可能なビューを表示するには、[Window] → [Show View] をクリックします。

ウィンドウをさまざまに移動してみてください。ワークスペースを自在にカスタマイズできるのは、SDK の優れた機能の 1 つです。

チュートリアル：ソフトウェアの編集

前の章で、サンプル ソフトウェア モジュールをコンパイルし、デバッグしました。このチュートリアルでは、さらに 2 つのサンプル モジュールを実行し、これら 2 つのルーチンを呼び出すソフトウェア モジュールを作成します。複数のソース ファイルを含むソフトウェア プロジェクトの管理方法を学ぶのが目的です。

1. SDK を閉じている場合は、[スタート] → [プログラム] → [Xilinx ISE Design Suite 11] → [EDK] → [Xilinx Software Development Kit] をクリックして再起動します。
2. 新規ワークスペースを作成します。
3. 第 5 章の「チュートリアル：ハードウェア プラットフォームのインポート」の手順に従って、同じハードウェア仕様ファイルを使用してハードウェア プラットフォームをインポートします。
4. 第 5 章の「チュートリアル：ソフトウェア プラットフォームの作成」の手順に従って新しい standalone ソフトウェア プラットフォームを作成します。ソフトウェア プラットフォームの名前は editor_exercise_platform にします。

チュートリアル：Managed Make C アプリケーション プロジェクトの作成

次に、異なるサンプル アプリケーションを関連付けた Managed Make C アプリケーション プロジェクトを 2 つ作成します。その後、空の C アプリケーション プロジェクトを作成してこれら 2 つのファイルを含める方法を学びます。

このような基本的なファイル管理は、大型のプロジェクトで必要です。手順が複雑だと感じられる場合は、第 5 章のチュートリアルに戻ってプロジェクトの管理方法を復習してください。

1. 第 5 章を参照して、Managed Make C アプリケーション プロジェクトを作成します。[Memory Test] サンプル アプリケーションを選択します。
2. Managed Make C アプリケーション プロジェクトをもう 1 つ作成します。このプロジェクトでは、[Peripheral Tests] サンプル アプリケーションを選択します。

3. [C/C++ Projects] ビューの表示は、次のようになっているはずです。両方の C プロジェクト (memory_tests および peripheral_tests) が editor_exercise_platform 用に作成されています。

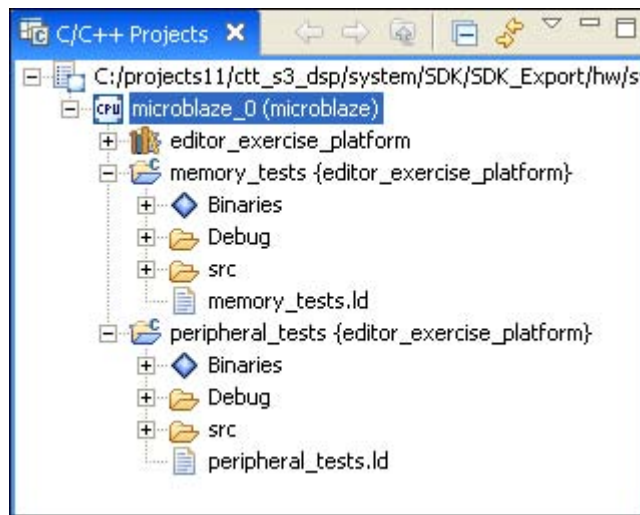


図 6-1 : 2 つの C プロジェクト

これらの 2 つのアプリケーションを実行するには、前の章と同様に、FPGA ビットストリームをボードにダウンロードする必要があります。

4. [Tools] → [Program FPGA] をクリックします。
5. 開いたダイアログ ボックスで、次のように設定します。
- ◆ [Bit File] : プロジェクト フォルダに含まれる system.bit ファイルを選択
 - ◆ [Bmm File] : プロジェクト フォルダに含まれる edkBmmFile_bd.bmm ファイルを選択
 - ◆ [Specify the ELF file to be initialized to each processor's BRAM memory] : [Processor] が [microblaze_0]、[Type] が [microblaze]、[Initialization ELF] が [BootLoop]

メモ : ハードウェア プロジェクトの設定方法によって、ファイル名が異なる場合があります。

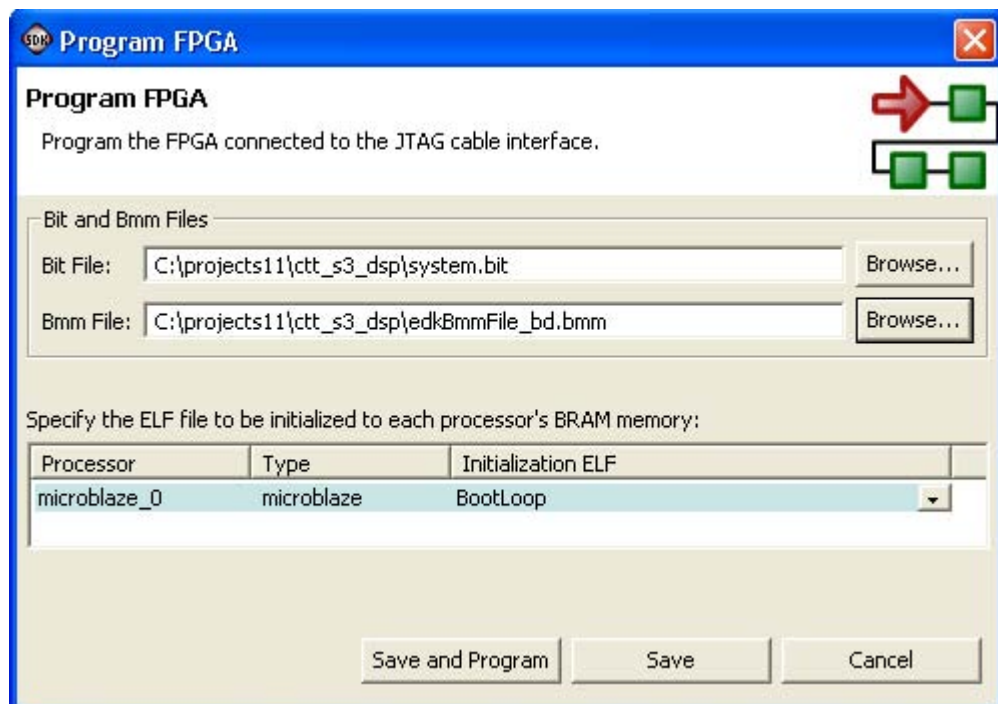


図 6-2 : [Program FPGA] ダイアログ ボックス

2 つのサンプル プログラムで何が実行されるかを確認するため、まず `memory_tests` プロジェクトを実行し、その後 `peripheral_tests` プロジェクトを実行します。

6. 次の手順に従って、`memory_tests` プロジェクトを実行します。
 - a. [C/C++ Projects] タブで `memory_tests/Binaries` の下にある `memory_tests.elf` ファイルを見つけます。
 - b. [`memory_tests.elf`] を右クリックし、[Debug As] → [Debug on Hardware] をクリックします。
 - c. [Debug] パースペクティブが開いたら、[Run] → [Resume] をクリックしてプログラムを実行します。ターミナル ウィンドウでプログラムの出力を確認します。
 - d. [Run] → [Terminate] をクリックしてデバッグ セッションを終了します。

7. [C/C++] パースペクティブをクリックし、`peripheral_tests` C プロジェクトに対して手順 6 の操作を実行します。

`memory_tests` および `peripheral_tests` アプリケーションを実行すると、ターミナルウィンドウの出力は次のようになります。

```
--Starting Memory Test Application--
Testing memory region: DDR2_SDRAM
  Memory Controller: mpmc
    Base Address: 0x88000000
      Size: 0x08000000 bytes
        32-bit test: PASSED!
        16-bit test: PASSED!
        8-bit test: PASSED!
--Memory Test Application Complete--
---Entering main---

Running GpioInputExample() for DIP_Switches_8Bit...
GpioInputExample PASSED. Read data:0x0

Running GpioOutputExample() for LEDs_8Bit...
GpioOutputExample PASSED.

Running GpioInputExample() for Push_Buttons...
GpioInputExample PASSED. Read data:0x0

Running UartLiteSelfTestExample() for mdm_0...
UartLiteSelfTestExample PASSED
---Exiting main---
```

図 6-3 : ターミナル ウィンドウの出力

8. [Run] → [Terminate] をクリックしてデバッグ セッションを終了します。

2 つのアプリケーションが正しく実行されました。この後、これらの 2 つのアプリケーションを呼び出す 3 つ目のアプリケーションを作成します。SDK では、これを既存のアプリケーションをインポートすることにより実行します。次のチュートリアルで、アプリケーションをインポートします。

チュートリアル：複数のソース ファイルでの作業

既存の 2 つのソフトウェア アプリケーションを、3 つ目のアプリケーションで呼び出せるように変更する必要があります。各アプリケーションの `main()` を変更し、新しい `main()` 関数で呼び出せるようにします。

1. [C/C++] パースペクティブで、`memorytest.c` および `testperiph.c` ファイルをダブルクリックして開きます。

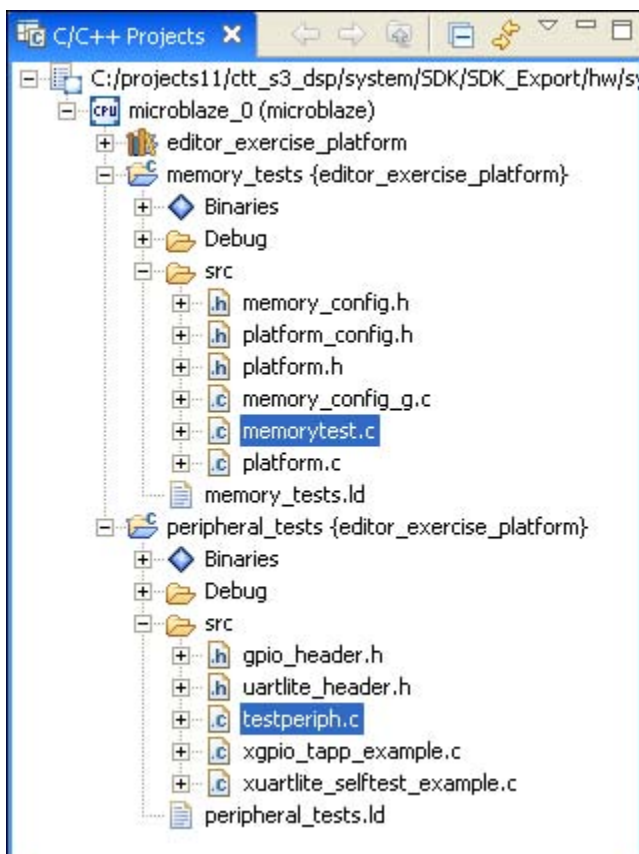


図 6-4 : [C/C++ Projects] タブの 2 つのプロジェクト

2. `memorytest.c` で、`main()` を `memorytest_main()` に変更します。53 行目付近にあります。
3. `testperiph.c` で、`main()` を `peripheraltest_main()` に変更します。46 行目付近にあります。

4. ファイルを保存します。

保存すると、ファイルが自動的に構築されます。main 関数がないので、この処理でエラーが発生します。関数名を main() に戻せば、エラーはなくなります。

次に、memorytest_main() および peripheraltest_main() 関数を呼び出すモジュールを作成します。

5. 第 5 章の「チュートリアル: ソフトウェア環境の設定」を参照して、Managed Make C アプリケーション プロジェクトを作成します。[Empty Application] サンプル アプリケーションを選択し、プロジェクト名を「top_test」とします。
6. [File] → [New] → [File] をクリックします。[New File] ダイアログ ボックスで、[Enter or select the parent folder] で [top_test] を選択し、[File name] に「top_test.c」と入力してファイルを作成します。
7. top_test.c ファイルを開き、次の図に示すコードを入力します。

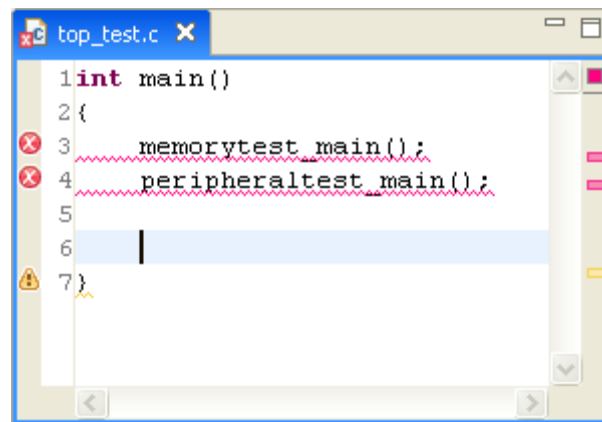


図 6-5 : top_test.c

8. ファイルを保存します。SDK によりファイルが自動的に構築されます。この自動構築機能は、[Project] → [Build Automatically] をクリックしてオン/オフを切り替えることができます。

SDK で memorytest_main() および peripheraltest_main() を見つけることができないので、エラーが発生します。

次に、memorytest_main() および peripheraltest_main() 関数を top_test プロジェクトにインポートし、top_test ソフトウェア プロジェクトでアクセスできるようにします。

1. [File] → [Import] をクリックし、[File system] を選択します。
2. [File system] ページで次の図に示すように設定します。

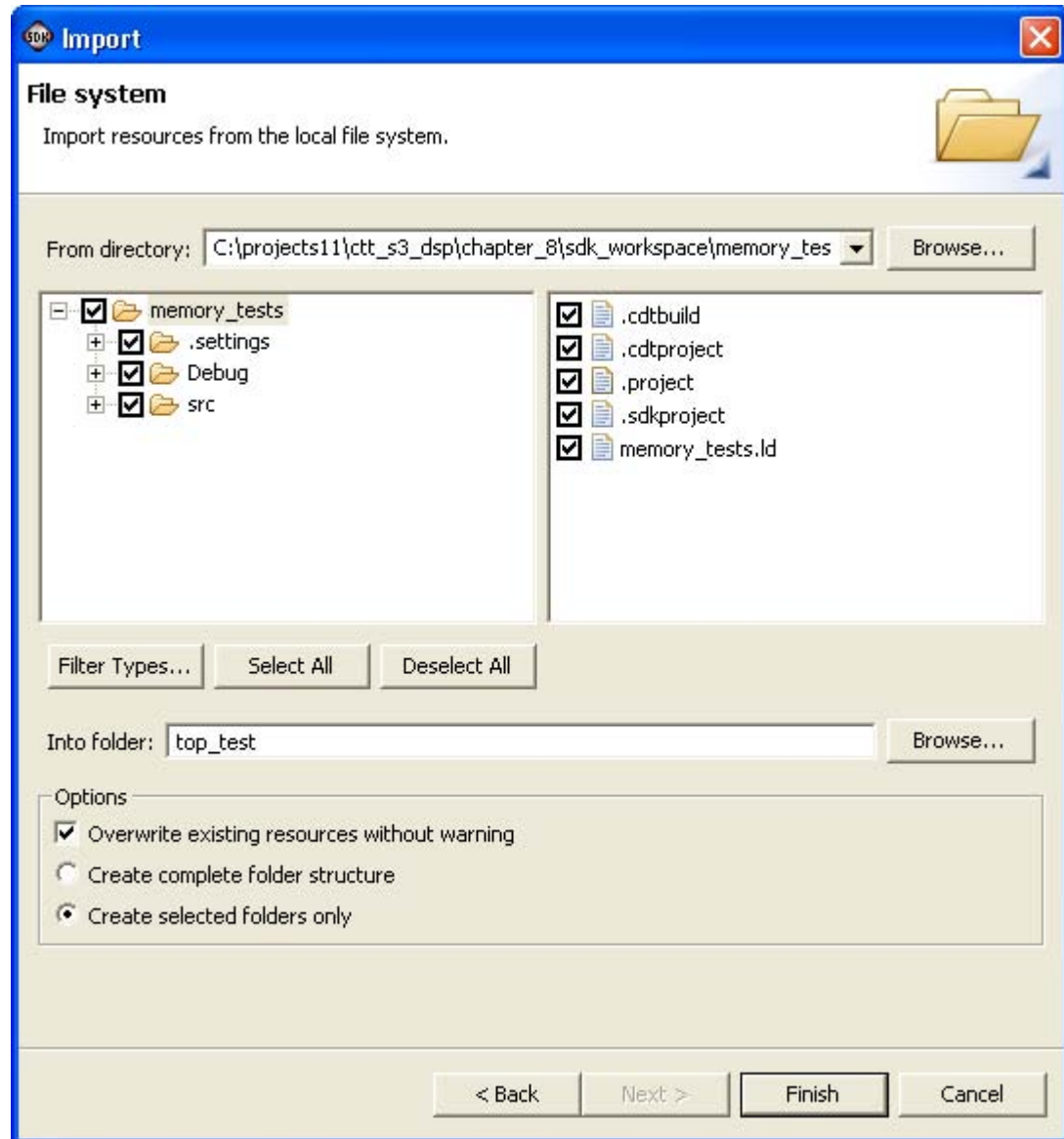


図 6-6：ファイル システムのインポート

上図のダイアログ ボックスでは、memory_tests ファイル システムのインポート方法を指定しています。この例では、インポートする C ファイルが各ディレクトリに 1 つあるので、ファイル階層全体をインポートする必要はありません。より複雑なプロジェクトをインポートする場合は、[Create Complete Folder Structure] をオンにします。ここでもこのオプションをオンにできますが、結果は同じになります。

3. peripheral_tests ファイル システムに対して同じ操作を実行します。

これら 2 つのファイル システムをインポート すると、`top_test.c` のエラーがなくなり、[C/C++ Projects] タブの表示は次の図に示すようになります。エラーがなくなる場合は、42 ページのチュートリアルの手順 2 で作成したファイルがインポート されているかどうかを確認してください。

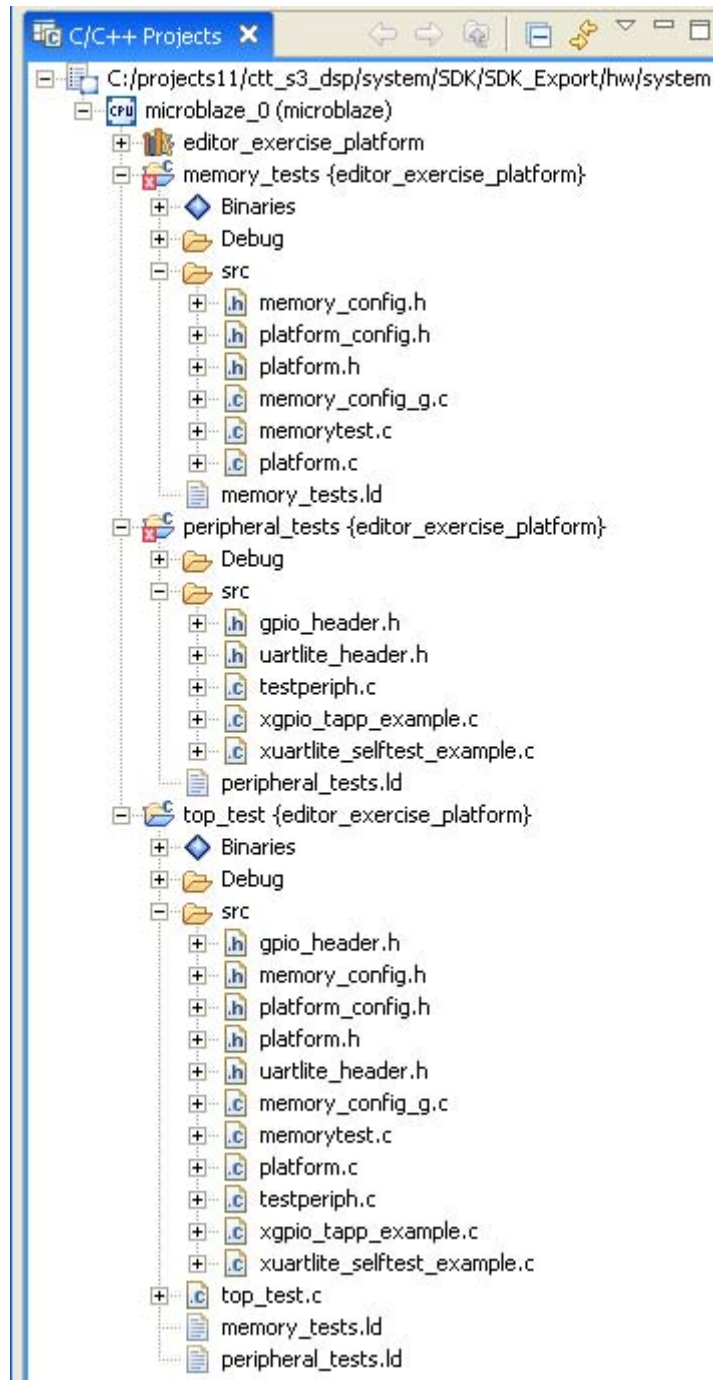


図 6-7: プロジェクト ディレクトリ ツリー

4. test_top を右クリックして [Generate Linker Script] をクリックし、アプリケーションのカスタム リンカ スクリプトを作成します。
5. top_test.elf ファイルを右クリックし、[Debug As] → [Debug on Hardware] をクリックしてアプリケーションをダウンロードおよび実行し、正しく実行されることを確認します。ターミナル ウィンドウに、memory_tests および peripheral_tests の両方が正しく機能していることが示されます。

チュートリアル：デバッガの使用

このチュートリアルでは、デバッガの機能を見えます。SDK には、ソース レベル デバッグ機能が含まれています。ほかのデバッガを使用したことがある場合、SDK デバッガに通常の機能のほとんどが含まれていることがわかります。

[Debug] パースペクティブの [Debug] ビューには、デバッグ セッションの状態に関する詳細な情報が表示されます。

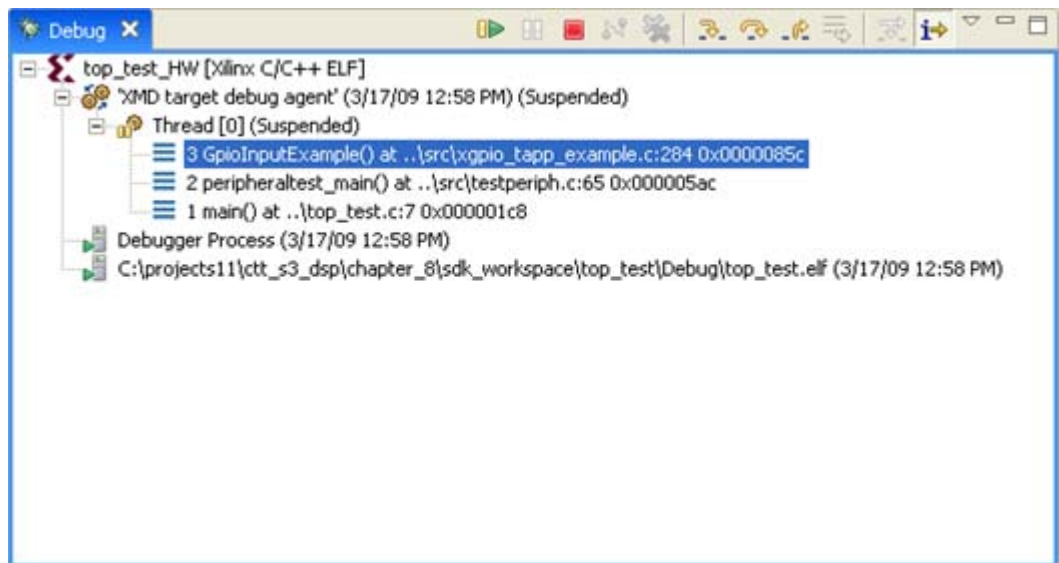


図 6-8 : [Debug] ビュー

上図では、呼び出しスタックが 3 レベルであることがわかります。アドレス 0x000001c8 の main() によりアドレス 0x000005ac の peripheraltest_main() が呼び出され、そこからアドレス 0x0000085c の GpioInputExample() が呼び出されています。

プログラムは現在停止されており、ブレークポイントに達したことを意味します。呼び出しスタックの各アイテムには、呼び出しルーチンのあるコード行も示されています。

ソフトウェアの実行も [Debug] ビューから制御できます。[Debug] パースペクティブで [Debug] ビューの上部にあるボタンの上にマウスを配置すると、各ボタンの機能が表示されます。

チュートリアル：デバッグ出力の観察

このチュートリアルを開始する前に、50 ページの「チュートリアル：デバッガの使用」のチュートリアルを終了してください。

1. [C/C++ Projects] タブで `top_test.elf` ファイルを選択します。[Run] → [Debug As] → [Debug on Hardware] をクリックし、`top_test.elf` ファイルをターゲット ボードにダウンロードします。ダウンロードが正常に完了すると、[Debug] パースペクティブが開きます。
2. [Disassembly] や [Memory] など、必要なビューが表示されていない場合は、[Window] → [Show View] をクリックしてビューを選択してください。

ビューの位置は、ドラッグ アンド ドロップで自由に移動できます。

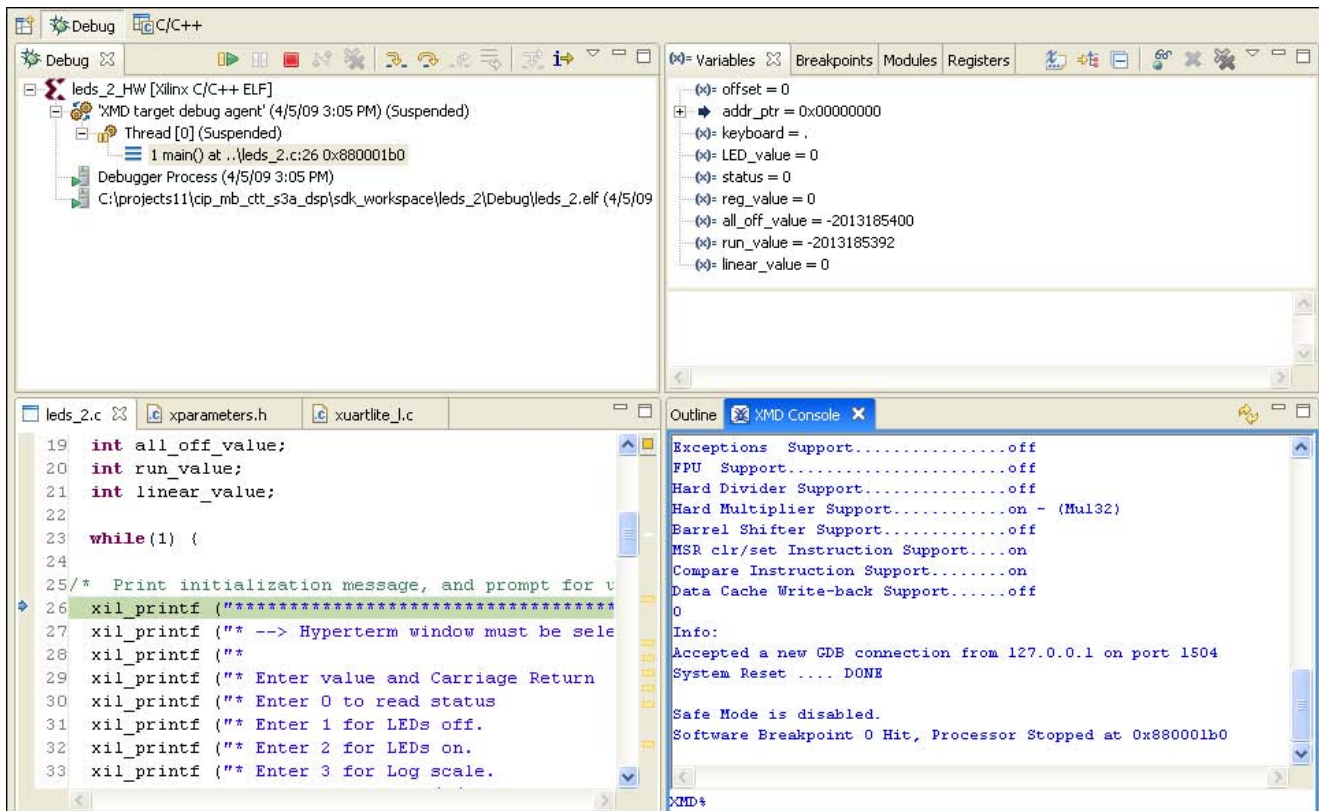


図 6-9 : [Debug] パースペクティブ

上図では、プロセッサ コードは `main()` の最初にあり、プログラムの実行が 26 行目で停止しています。

[Disassembly] ビューを見ると、アセンブリ レベルのプログラムの実行が `0x000001b8` で停止していることがわかります。

[Registers] ビューを開き、RPC レジスタ (プログラム カウンタ) に `0x000001b8` が含まれていることを確認します。[Registers] ビューが表示されていない場合は、[Window] → [Show View] → [Registers] をクリックします。

3. `top_test.c` ファイルの `peripheraltest_main()` 行の左側のマージンをダブルクリックします。`peripheraltest_main()` にブレークポイントが設定されます。

4. [Breakpoints] ビューでブレークポイント が設定されていることを確認します。[Breakpoints] ビューが表示されていない場合は、[Window] → [Show View] → [Breakpoints] をクリックします。
5. [Run] → [Resume] をクリックしてプログラムの実行を再開し、ブレークポイント まで実行します。[Debug] ビューおよび [Disassembly] ビューから、プログラムの実行が `peripheraltest_main()` の行、アドレス `0x000001c0` で停止したことがわかります。
6. [Run] → [Step Into] をクリックし、`peripheraltest_main()` ルーチンに移動します。
`peripheraltest_main()` ルーチンが実行され、`0x00000580` で停止します。呼び出しスタックは 2 レベルになります。
7. [Run] → [Resume] をクリックして、プログラムを最後まで実行します。
プログラムの実行が終了すると、[Debug] ビューにプログラムが `exit` ルーチンで停止したことが示されます。デバッガでプログラムを実行すると、このようになります。
8. ターミナル ウィンドウで `peripheraltest_main()` および `memorytest_main()` の両方が実行されたことを確認します。
9. コードを数回実行します。コードを 1 行ずつ実行、メモリやブレークポイントを確認、コードを変更、`print` 文を追加するなど、いろいろ試してみてください。また、ビューを追加したり移動したりしてみてください。

これで、複数のファイルを含む C プログラムを機能させることができました。また、デバッガの使用方法、SDK のカスタマイズ方法も学びました。

次の操作

次の章では、独自の IP を作成します。その後の章では、デュアル プロセッサ デザインを作成し、デバッグする方法を説明します。

独自の IP の作成

XPS (Xilinx® Platform Studio) ではデザイン作成のほとんどの操作が自動的に実行されるので、エンベデッド プロセッサ システムを簡単に作成できます。Base System Builder (BSB) ウィザードを使用すると、設計にかかる時間と労力を削減できます。

XPS および BSB ウィザードの利点

BSB ウィザードを使用して、ほとんどのエンベデッド プロセッサ デザインを作成できます。BSB ウィザードで作成したデザインは、XPS でカスタマイズできます。 デザインのカスタマイズは、UART Lite のボー レートを変更するなど既存の IP コアの一部のパラメータを変更するだけで済む場合から、カスタム IP を設計して既存のデザインに組み込む必要がある場合があります。

CIP ウィザードの利点

必要なカスタム IP の機能のほかに、CoreConnect™ バス プロトコル、XPS に必要な pcores ディレクトリ構造、バス ファンクション モデル シミュレーション フレームワークの作成などを理解する必要があります。この章では、これらのシステムの特徴を説明し、Create and Import Peripheral (CIP) Wizard を使用してカスタム IP を作成する手順を示します。

CIP ウィザードの使用

CIP ウィザードには、BSB ウィザードと同様の利点があります。バス インターフェイス ロジックなどのデザインのフレームワークを作成し、カスタム ロジックを統合するための HDL テンプレートを提供します。カスタム ペリフェラル コア (pcore) をエンベデッド デザインに含めるために必要なファイルは、すべて CIP ウィザードで生成されます。

カスタム IP の作成は、XPS の最も理解されていない機能の 1 つです。CIP ウィザードを使用すれば pcore フレームワークを簡単に作成できますが、どのような処理がなぜ実行されるのかを理解するのは重要です。この章では基本を説明し、初期プロセスを示します。また、参照および解析用に完成した pcore デザインも含まれています。

IP 作成の概要

XPS の [System Assembly View] (18 ページの図 3-1) に、バス、プロセッサ、および IP の接続が表示されます。作成する IP は、設計しているシステムに互換している必要があります。

互換性のある IP を作成するには、次の手順に従います。

1. IP に必要なインターフェイスを特定します。カスタム ペリフェラルを接続するバスを特定する必要があります。たとえば、次のインターフェイスを選択できます。
 - ◆ プロセッサ ローカル バス (PLB) バージョン 4.6：プロセッサと高パフォーマンス ペリフェラル間的高速インターフェイスです。PowerPC® および MicroBlaze™ プロセッサシステムの両方で使用されます。
 - ◆ 高速シンプレックス リンク (FSL)：ポイント ツー ポイントの FIFO のようなインターフェイスです。MicroBlaze プロセッサ デザインで使用でき、PowerPC プロセッサシステムでは通常使用されません。
2. 機能をインプリメントし、検証します。EDK ペリフェラル ライブラリで使用可能な共通機能を再利用できることを念頭においてください。
3. スタンドアロン コアを検証します。コアを分離することにより、デバッグが簡単になります。
4. IP を EDK にインポートします。ペリフェラルを EDK のリポジトリ検索パスにコピーする必要があります。プラットフォーム仕様フォーマット (PSF) の MPD (Microprocessor Peripheral Definition) および PAO (Peripheral Analyze Order) ファイルを作成し、EDK ツールでペリフェラルが認識されるようにします。
5. ペリフェラルを XPS で作成したプロセッサ システムに追加します。

CIP ウィザードを使用したカスタム IP の作成

CIP ウィザードは、カスタム IP を作成、検証、インプリメントするのに必要な手順の実行を支援します。また、XPS でサポートされるバスをサポートします。

カスタム ロジックを直接 PLBv46 バスに接続するのが最も良くあるケースです。CIP ウィザードを使用すると、バス プロトコルの詳細を理解していなくても簡単にバスを接続できます。スレーブ接続とマスタ接続の両方を使用できます。

CIP ウィザードで IP の HDL テンプレートと BFM シミュレーションを作成

CIP ウィザードは、IP 作成プロセスを順を追って示すことにより、デザインのインプリメンテーションおよび検証を支援します。多数のテンプレートが設定され、これを IP ロジック用に編集できます。

HDL テンプレートの作成に加え、BFM (バス ファンクション モデル) 検証用に pcore 検証プロジェクトも作成されます。テンプレートと BFM プロジェクトの作成により、IP 開発を即座に開始でき、IP が作成中のシステムに互換することを確実にできます。BFM シミュレーションの詳細は、[付録 B 「IP バス ファンクション モデル シミュレーション」](#) を参照してください。

CIP ウィザードの実行前に知っておくべきこと

CIP ウィザードでプロジェクトを作成する前に、次の情報を確認します。

CIP ウィザードでサポートされるペリフェラル

CIP ウィザードでは、定義済みの IP インターフェイス (IPIF) ライブラリを使用して、4 種類の PLB v4.6 ペリフェラルを作成できます。

- シングル データ ビート転送用 PLB v4.6 スレーブ
- バースト データ転送用 PLB v4.6 スレーブ
- シングル データ ビート転送用 PLB v4.6 マスタ
- バースト データ転送用 PLB v4.6 マスタ

CIP ウィザードでは、高速シンプレックス リンク (FSL) ペリフェラルの作成がサポートされます。ウィザードの [Bus Interface] ページの下部にある [Enable OPB and PLB v3.4 bus interfaces] をオンにすると、以前の PLB v3.4 および OPB バスも使用できるようになります。

メモ：OPB および PLBv3.4 IP のサポートは、XPS 12 で削除される予定です。

資料

CIP ウィザードを起動する前に、使用するバス インターフェイスの資料を参照してください。資料に含まれる情報に目を通すことにより、バス システム インターフェイスの詳細を理解し、混乱を避けることができます。CIP ウィザードの詳細は、[Help] → [Help Topics] をクリックし、XPS ヘルプで「エンベデッド プロセッサの設計手順」→「ペリフェラルの作成とインポート」のセクションを参照してください。このヘルプに、CIP ウィザードを実行する際に必要な基本的な情報が含まれています。

IP データシートへのアクセス

XPS には、システムの IP に関連するデータシートが含まれています。データシートにアクセスするには、[Help] → [View Start Up Page] をクリックして [Start Up Page] を開き、[Documentation] タブをクリックして [IP Reference and Device Drivers Documentation] を展開表示し、[Processor IP Catalog] をクリックします。

PLBv4.6 ペリフェラルを作成する場合は、カスタム ペリフェラルがスレーブかマスタか、シングル データ アクセスかバースト データ アクセスかによって、次のいずれかのデータシートを参照してください。

- plbv46_slave_single
- plbv46_master_single
- plbv46_slave_burst
- plbv46_master_burst

IP インターコネクト (IPIC) 信号に関する説明は、カスタム ロジックに接続する IPIF 信号を特定するのに役立ちます。

メモ：次のチュートリアルで説明するように、CIP ウィザードは通常 XPS から起動しますが、XPS の環境外で実行することも可能です。

チュートリアル：テンプレートの生成と保存

このチュートリアルでは、CIP ウィザードを使用してカスタム ペリフェラル用のテンプレートを作成します。簡単にするため、ほとんどの手順ではデフォルト値を使用しますが、設定可能なオプションをすべて見ていくことができます。

注意：アドバンス ユーザー以外は、このチュートリアルを実行する前に第 4 章「エンベデッド プラットフォームの操作」および第 5 章「ソフトウェア開発キット」のチュートリアルを完了してください。

1. CIP ウィザードを起動し、カスタム ペリフェラル ファイルの保存先を指定します。
 - a. ISE® Project Navigator を起動し、プロジェクトを読み込みます。system.xmp を選択し、[Manage Processor Design (XPS)] プロセスをダブルクリックして XPS を起動します。
 - b. XPS で、[Hardware] → [Create or Import Peripheral] をクリックします。

最初のページで [Next] をクリックすると、[Peripheral Flow] ページが開きます。ここで、新規ペリフェラルを作成するか、既存のペリフェラルをインポートするかを指定します。

2. [Create templates for a new peripheral] をオンにします。次のページに進む前に、このページに記載されている情報に目を通してください。

メモ：CIP ウィザードの各ページには、有益な情報が記載されています。[More Info] をクリックすると、関連の XPS ヘルプトピックが表示されます。

3. [Repository or Project] ページで、カスタム ペリフェラル ファイルの保存先を指定します。このペリフェラルは 1 つのエンベデッド プロジェクトで使用するので、[To an XPS project] をオンにします。

CIP ウィザードを XPS から起動したので、ディレクトリの場所は既に入力されています。

メモ：カスタム pcore を複数のエンベデッド プロジェクトで使用する場合は、ファイルを EDK レポジトリに保存できます。

4. [Next] をクリックします。[Name and Version] ページが表示されます。

Create Peripheral

Name and Version
Indicate the name and version of your peripheral.

Enter the name of the peripheral (upper case characters are not allowed). This name will be used as the top HDL design entity.

Name:

Version: 1.00.a

Major revision: Minor revision: Hardware/Software compatibility revision:

Description:

Logical library name: pwm_lights_v1_00_a

All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compiled into the logical library name above. Any other referred logical libraries in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.

[More Info](#) [< Back](#) [Next >](#) [Cancel](#)

図 7-1 : [Name and Version] ページ

このページでは、ペリフェラルの名前とバージョンを指定します。

- ◆ このチュートリアルでは、「pwm_lights」という名前を使用します。
 - ◆ バージョン番号は自動的に入力されますが、必要に応じて変更できます。プロジェクトの説明も記述できます。
5. [Bus Interface] ページでは、ペリフェラルをエンベデッド デザインに接続するプロセッサ バスを選択します。このチュートリアルでは、[Processor Local Bus (PLB v4.6)] をオンにします。
- メモ : [Bus Interface] ページから、関連するデータシートにアクセスできます。
6. [Next] をクリックします。[IPIF (IP Interface) Services] ページが表示されます。

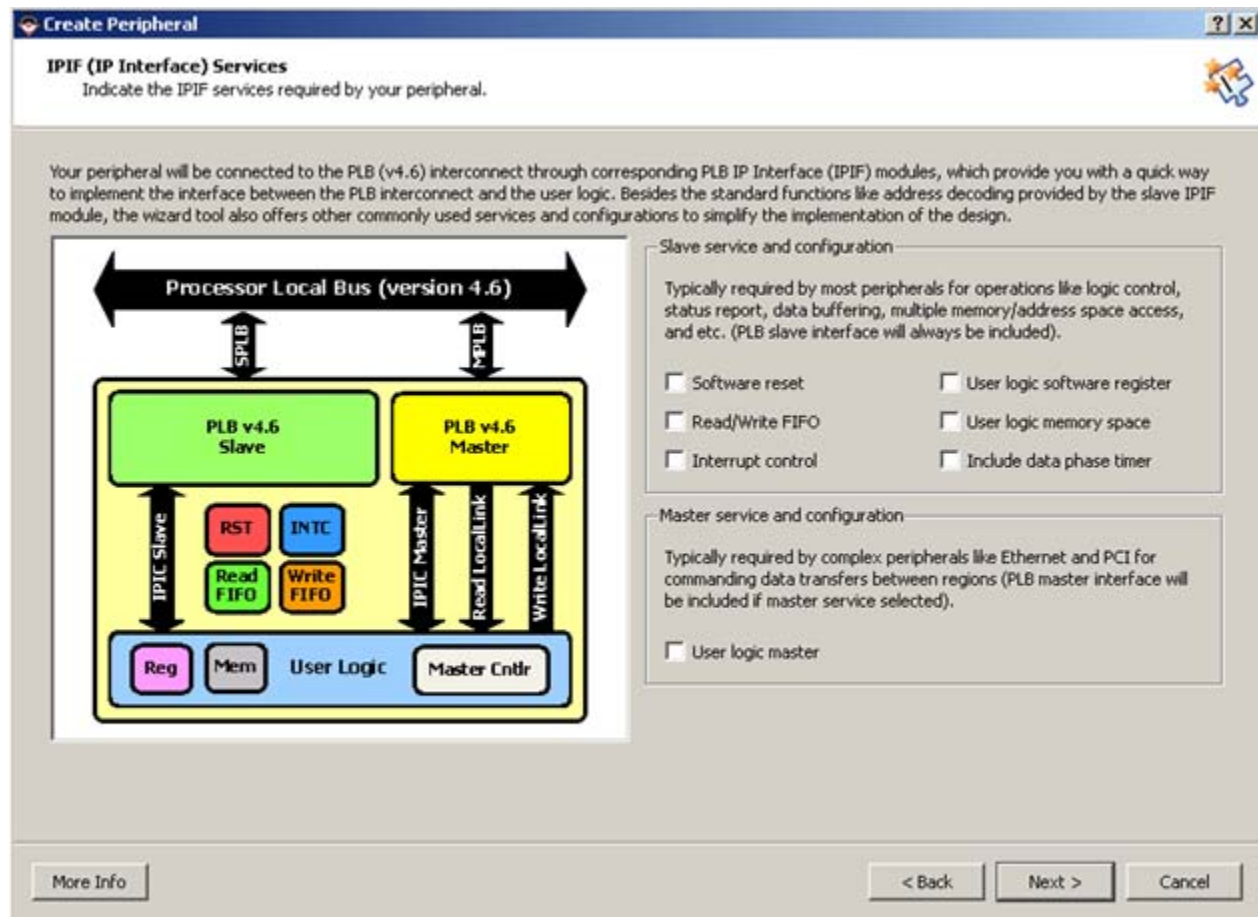


図 7-2 : [IPIF (IP Interface) Services] ページ

このページでは、CIP ウィザードにより自動的に次のものが作成されます。

- ◆ PLB バスへのマスタまたはスレーブ接続
- ◆ 必要なバス プロトコル ロジック
- ◆ カスタム HDL コードを接続するのに使用する信号セット

これらの基本的な機能以外に、オプションのサービスも追加できます。

[More Info] をクリックし、開いたヘルプ トピックで [IPIF の機能] リンクをクリックしてください。各サービスの詳細を参照して、その機能が IP で必要かどうかを判断できます。

7. すべてのチェック ボックスをオフにします。このチュートリアルでは、これらのサービスは必要ありません。

次の [Slave Interface] ページでは、バーストおよびキャッシュ ライン サポートを設定します。このチュートリアルではこのサポートは使用しませんが、スレーブ パリフェラルおよびデータ幅に関する記述に目を通してから次のページに進んでください。

8. [Next] をクリックします。[IP Interconnect (IPIC)] ページが表示されます。

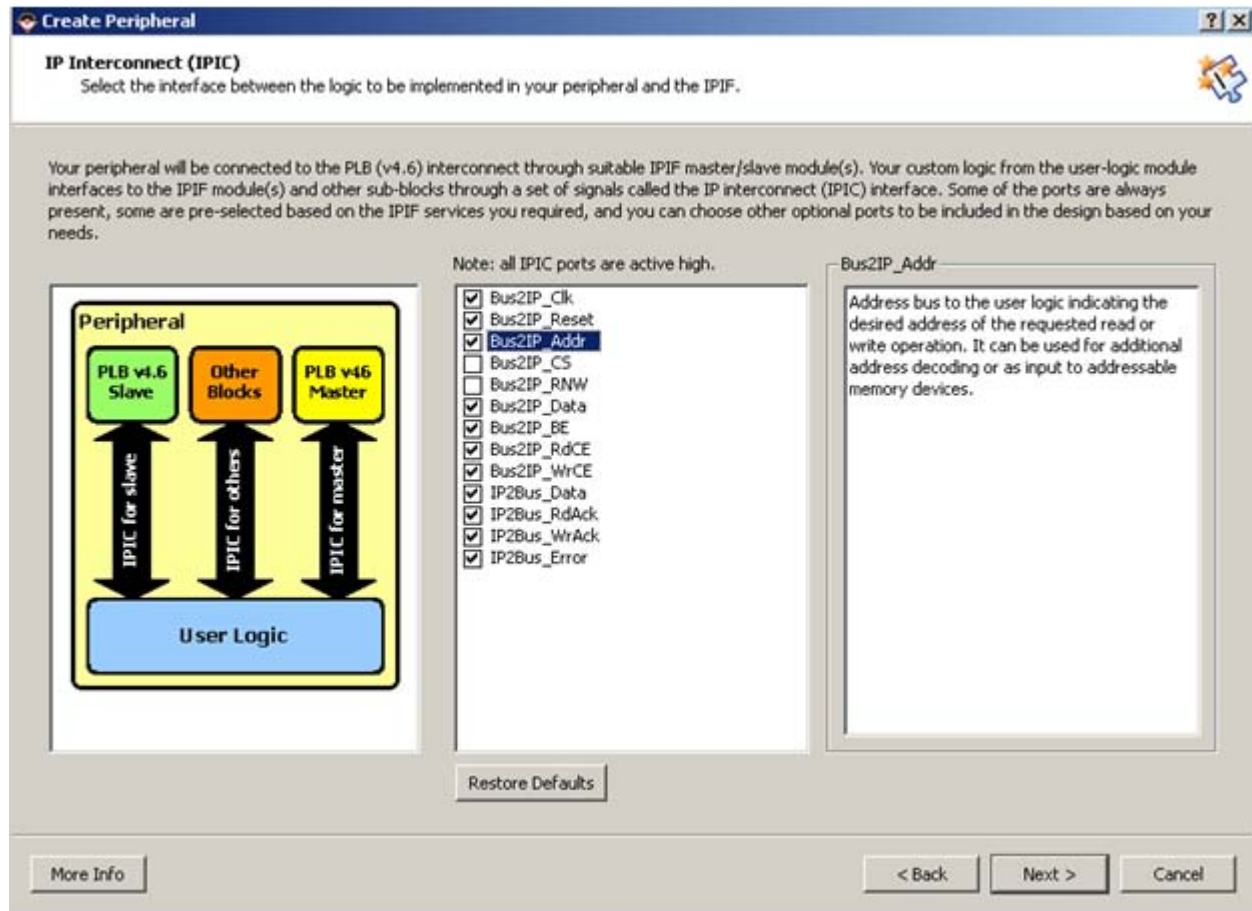


図 7-3 : [IP Interconnect (IPIC)] ページ

このページには、カスタム ペリフェラルで使用可能な **IPIC** 信号が表示されます。これらの信号の機能がわからない場合は、前のページに戻って該当する仕様を参照してください。選択されている信号で、ほとんどのカスタム ペリフェラルを適切に接続できます。

このチュートリアルでは、異なるアドレスへの複数の書き込みをデコードする必要があるので、HDL 内にデコード ロジックを作成するのに必要な [Bus2IP_Addr] 信号を追加します。

[IPIF (IP Interface) Services] ページで [User logic memory space] をオンにした場合、ユーザー メモリ空間を管理するページが開きます。

9. [Bus2IP_Addr] をオンにします。その他のチェック ボックスは変更しません。
10. [Next] をクリックします。[Peripheral Simulation Support] ページが表示されます。

このページでは、プロジェクトの BFM シミュレーション プラットフォームを生成するかどうかを指定します。BFM シミュレーション プラットフォームを生成するには、次が必要です。

- ◆ EDK 用の BFM シミュレーション パッケージがダウンロードおよびインストールされている。
- ◆ ModelSim SE または PE がインストールされている。

[BFM Package Installation Instructions] リンクをクリックすると、BFM のライセンスを取得し、ダウンロードおよびインストールできます。BFM シミュレーションについては、付録 B「IP バス ファンクション モデル シミュレーション」を参照してください。このチュートリアルで作成している IP で BFM シミュレーションを実行する予定の場合は、BFM プラットフォームを生成してください。

CIP ウィザードでは、pcore フレームワークをインプリメントする 2 つの HDL テンプレートが作成されます。

- ◆ `pwm_lights.vhd` ファイル：PLBv4.6 バス インターフェイス ロジックが含まれます。ペリフェラルにシステム外部へのポートが含まれる場合は、適切なポート名を追加する必要があります。このファイルには説明が記述されており、ポート情報をどこに追加したらよいのかははっきりわかります。

Verilog デザインの場合は、HDL 構文を使用してポート名を記述する必要があることに注意してください。次のチュートリアルソースコードを、今後の pcore 作成のテンプレートとして使用できます。

- ◆ `user_logic.vhd` ファイル：ペリフェラルを定義するカスタム RTL を追加するテンプレート ファイルです。追加のソース ファイルを作成することもできますが、ここで必要なのは `user_logic.vhd` ファイルのみです。

11. [Next] をクリックします。[Peripheral Implementation Support] ページが表示されます。

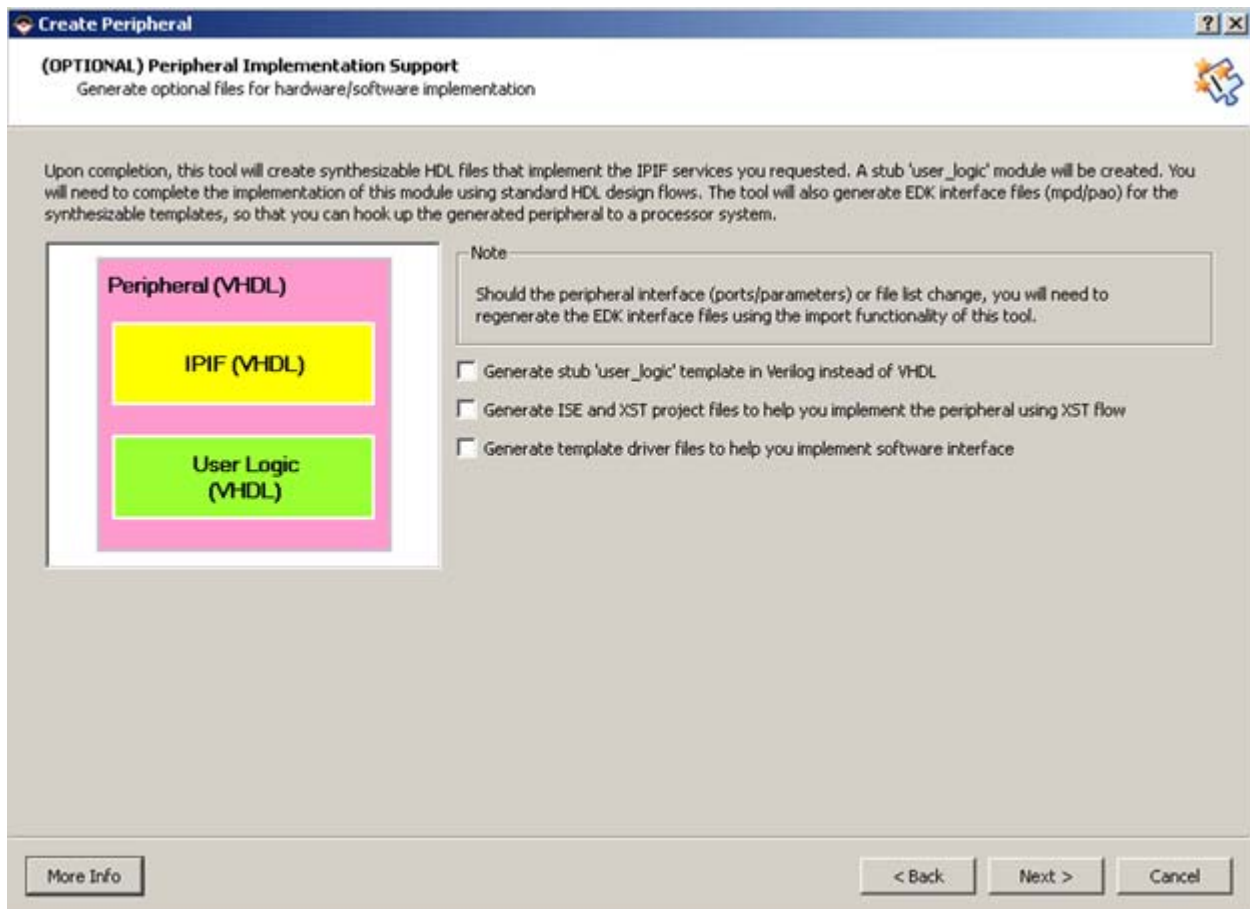


図 7-4 : [Peripheral Implementation Support] ページ

Verilog サポート

[Peripheral Implementation Support] ページには、ハードウェアおよびソフトウェアのインプリメンテーション用のファイルを生成するオプションがリストされています。

- ◆ `user_logic` テンプレートを VHDL の代わりに Verilog で生成する場合は、[Generate stub 'user_logic' template in Verilog instead of VHDL] をオンにします。
- ◆ `pcore` デザインをタイミング解析またはタイミング シミュレーション用にインプリメントする場合は、[Generate ISE and XST project files to help you implement the peripheral using XST flow] をオンにします。必要な ISE プロジェクト ファイルが生成されます。ペリフェラルが低速で単純な場合は不要です。
- ◆ ペリフェラルが複雑で複雑なソフトウェアドライバを必要とする場合は、[Generate template driver files to help you implement software interface] をオンにすると、選択したサービスに基づいて必要なドライバ構造およびプロトタイプドライバが作成されます。

このチュートリアルでは、これらのチェック ボックスはオフのままにします。最後のページに、CIP ウィザードで作成されるファイルとその保存先が表示されます。

サマリ情報

12. この情報を確認し、[Finish] をクリックします。ファイル生成ステータスが [Console] ウィンドウに表示されます。

実行された処理

CIP ウィザードでどのような処理が実行されたのでしょうか。ここで、IP 作成に関する概念とウィザードで作成された出力を検証してみます。

EDK では、PLB スレーブおよびバースト ペリフェラルを使用して、さまざまなプロセッサ ペリフェラルの一般的な機能をインプリメントします。PLB スレーブとバースト ペリフェラルは、バス マスタまたはバス スレーブとして機能します。

CIP ウィザードの [Bus Interface] および [IPIF (IP Interface) Services] ページでは、ターゲット バスと IP で使用するサービスを選択します。これにより、IP で必要な PLB スレーブとバースト ペリフェラル エlement を指定します。

PLBv4.6 スレーブ
およびバースト
ペリフェラル

PLB スレーブおよびバースト ペリフェラルは検証および最適化されており、詳細にパラメータ指定可能なインターフェイスです。また、簡略化されたバス プロトコルのセットも提供します。カスタム RTL は IPIC 信号に接続されるので、PLB または FSL バス プロトコルを直接操作するよりも非常に簡単です。PLB スレーブおよびバースト ペリフェラルを要件に合わせてパラメータ指定することにより、すべてを一から作成する必要がないので、設計およびテストにかかる労力を大幅に削減できます。

図 7-5 に、バス、単純な PLB スレーブ ペリフェラル、IPIC、およびユーザー ロジックの関係を示します。

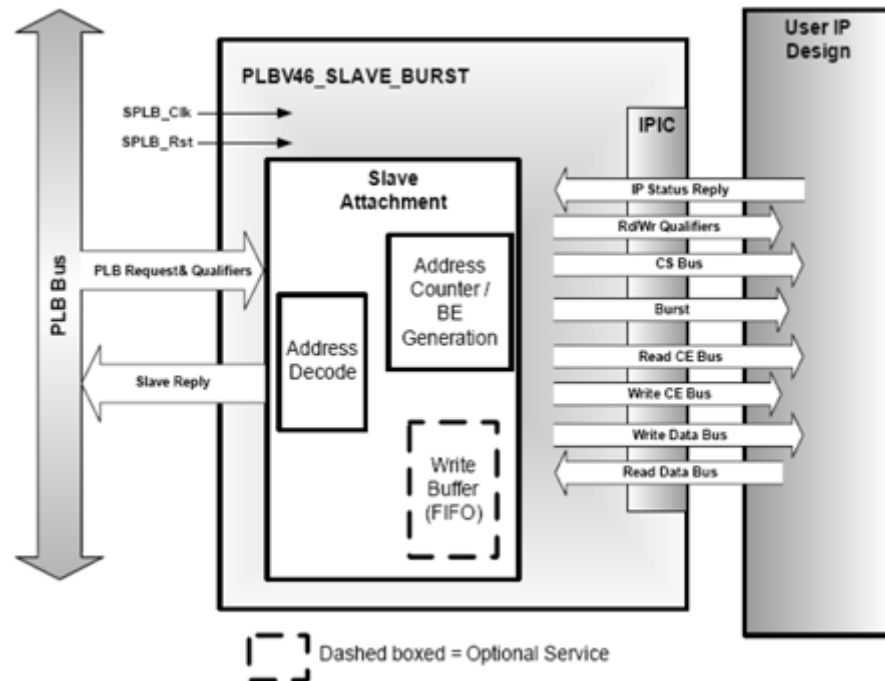


図 7-5：カスタム ペリフェラルの PLB スレーブ/バースト モジュール

次の図に、CIP ウィザードで作成されたディレクトリ構造と主要なファイルを示します。これらのファイルは、プロジェクト ディレクトリの pcores サブディレクトリに保存されています。

メモ：この図には、すべてのファイルは表示されていません。

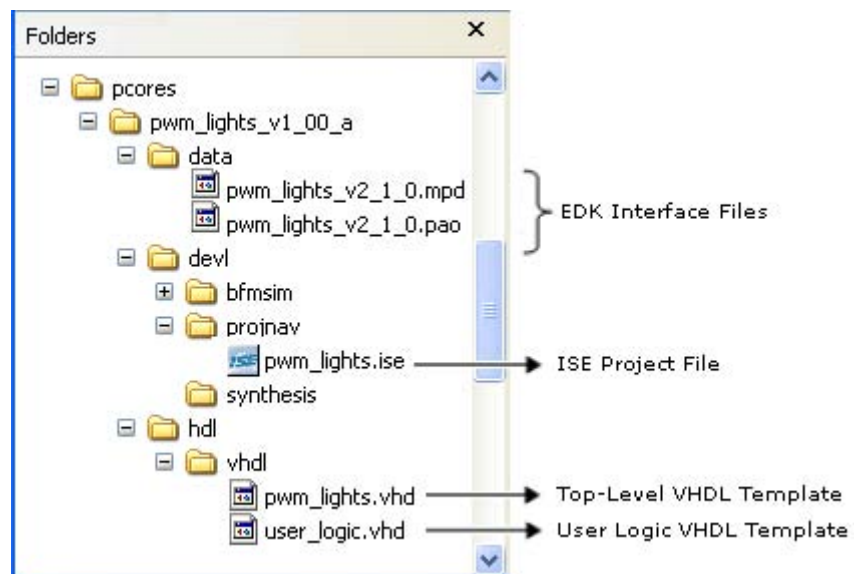


図 7-6：CIP ウィザードで生成されるディレクトリ構造

CIP ウィザードで生成されるファイルに関して、次の点に注意してください。

- `pwm_lights.vhd` および `user_logic.vhd` という 2 つの HDL テンプレート ファイルが生成されます。
- `user_logic` ファイルは、`pwm_lights.vhd` でコンフィギュレーションされた PLB スレーブ/バースト コアを使用して PLB v4.6 バスに接続します。
 - ◆ `user_logic.vhd` ファイルは、カスタム機能ブロックと同等です。
 - ◆ `pwm_lights.vhd` ファイルは、PLBv.46 スレーブ/バースト ブロックと同等です。
- カスタム ロジックは IPIC 信号を使用して接続されます。

次の図に、図 7-5 で示されるブロック図と図 7-6 に示されるファイルの関係を示します。

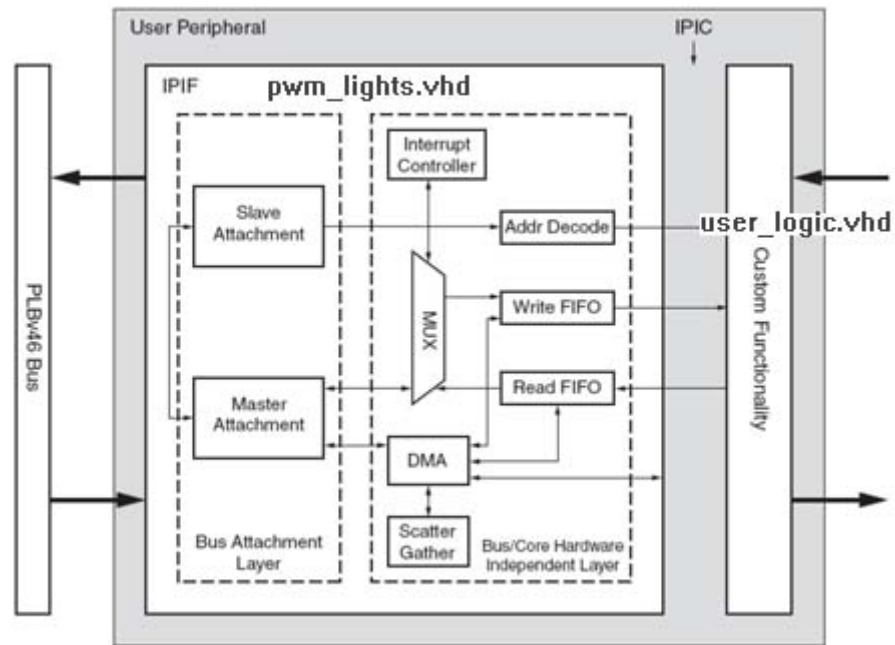


図 7-7 : IP モジュールと生成されたファイルの関係

デザインを完了するには、カスタム ロジックを 2 つのファイルに追加する必要があります。

サンプル デザインの説明

CIP ウィザードを使用すると、読み出しレジスタと書き込みレジスタで必要な機能が実行できる場合は、完全に機能するペリフェラルを作成できます。単純なペリフェラルはこのように作成できますが、実際に機能するサンプル デザインを変更する方法を知っておくと有益なので、個々では単純な PLBv4.6 ペリフェラルを定義します。

次のチュートリアルで、このペリフェラルのソース コードを開き、変更します。これらのファイルは、プロジェクト ディレクトリの `pcores` サブディレクトリに保存されています。このマニュアルに添付されているサンプル ファイルを使用することもできます。Adobe Acrobat Reader の左下にあるクリップ アイコンをクリックし、添付ファイルを確認してください。チュートリアルでこれらのファイルを開きます。

このカスタム ペリフェラルは、評価ボードの 8 個の LED を制御します。pwm_lights 回路は、次のように動作します。

- オフセット 0 に書き込むと LED がオフになります。
- オフセット 4 に書き込むと LED がオンになります。
- 単純な PWM 回路を使用してライトの輝度を制御します。輝度は 16 個の値により指定します。
 - ◆ オフセット 8 に書き込むと対数強度駆動スケールが使用されます。
 - ◆ オフセット 12 に書き込むと線形強度対数スケールが使用されます。
- 制御回路ステータスをリードバックします。

ハードウェア デザインに加え、単純なソフトウェア アプリケーションによりさまざまな設定とリードバック ステータスを制御します。

チュートリアル：CIP ウィザードで生成されたテンプレート ファイルの変更

このチュートリアルでは、CIP ウィザードで生成されたテンプレート ファイルを開き、変更します。

概念的には、このチュートリアルでの操作は簡単です。CIP ウィザードで作成した pcore を制御する C コードを読み込んで実行するだけです。チュートリアルを開始する前に、ソフトウェアの次の機能にも注目してください。

- ワークスペースで使用する system.xml ファイルは、SDK でモニタされます。このファイルは、第 4 章「[エンベデッド プラットフォームの操作](#)」で使用しました。このファイルが変更されると、その変更が SDK で認識されます。このチュートリアルでファイルにハードウェアを追加したときに、この機能の例を見ることができます。
- デフォルトでは、すべての C コードはブロック RAM にマップされます。このチュートリアル の C コードはこれまでのチュートリアルで使用したものよりもサイズが大きいため、SDK でマップされたブロック RAM では小さすぎます。そのため、リンカ スクリプトを変更する必要があります。SDK には、リンカ スクリプトの変更を簡略化する GUI があります。

1. XPS で [File] → [Open] をクリックし、pcores\pwm_lights_v1_00_a\hdl\vhdl ディレクトリに移動します。このディレクトリに pwm_lights.vhd ファイルと user_logic.vhd ファイルがあります (62 ページの図 7-6 を参照)。

メモ：これらのファイルを表示するには、[ファイルの種類] ドロップダウン リストから [VHDL] を選択する必要があります。

2. pwm_lights.vhd ファイルを開きます。

このファイルに外部ポート名を追加します。外部ポート名は、次の 2 箇所に追加します。

- ◆ 最上位エンティティ ポート宣言
- ◆ user_logic インスタンス化のポート マップ

3. 160 行目付近にスクロールします。次の図に示すように、最上位エンティティに LED ポート宣言を追加します。

```

port
(
  -- ADD USER PORTS BELOW THIS LINE -----
  LEDs                                     : out std_logic_vector(0 to 7);
  -- ADD USER PORTS ABOVE THIS LINE -----

  -- DO NOT EDIT BELOW THIS LINE -----
  -- Bus protocol ports, do not add to or delete
  SPLB_Clk                               : in  std_logic;
  SPLB_Rst                               : in  std_logic;

```

図 7-8 : LED ポートのコード

4. 390 行目付近にスクロールします。次の図に示すように、user_logic ポート マップに LED ポート宣言を追加します。

```

    C_SLV_DWIDTH                        => USER_SLV_DWIDTH,
    C_NUM_REG                           => USER_NUM_REG
  )
  port map
  (
    -- MAP USER PORTS BELOW THIS LINE -----
    LEDs                                => LEDs,|
    -- MAP USER PORTS ABOVE THIS LINE -----

    Bus2IP_Clk                         => ipif_Bus2IP_Clk,
    Bus2IP_Reset                       => ipif_Bus2IP_Reset,

```

図 7-9 : ポート宣言の追加

5. ファイルを保存します。
2 つのテンプレート ファイル (<ip core name>.vhd および user_logic.vhd) でユーザーが情報を入力する必要がある部分には、必要な情報とその場所を示すコメントが記述されています。
ほとんどの場合 <ip core name>.vhd に必要な変更は、最上位エンティティにポートを追加し、user_logic インスタンスにこれらのポートをマップすることだけです。
6. XPS で [File] → [Open] をクリックし、pcores\pwm_lights_v1_00_a\hdl\vhd1 ディレクトリに移動します。
7. user_logic.vhd ファイルを開きます。
8. 完成した user_logic.vhd ファイルは、このマニュアルに添付されています。添付ファイルは、Adobe Acrobat Reader の左下にあるクリップ アイコンをクリックすると表示されます。現在開いている user_logic.vhd ファイルの内容を、このマニュアルに添付されている user_logic.vhd ファイルの内容で置換し、ファイルを保存します。

ファイルの内容の確認

VHDL を理解していれば、pwm_lights のコードを理解するのは簡単です。

user_logic.vhd には、最上位 pwm_lights.vhd ファイルと同様に、カスタム RTL を追加する位置を示すコメントが含まれています。CIP ウィザードを使用していない場合は、コメントを参照し、インターフェイス信号のリストおよび RTL を追加する位置を確認してください。

自動生成されたジェネリックおよびポートは変更しないでください。指定された位置にカスタムジェネリックおよびポートのみを追加してください。

100 行目付近に、ユーザー ポート LEDs (0 to 7) が追加されています。この出力ベクタは、評価ボードの 8 個の LED を駆動します。デザインに特定の信号を追加した場合は、この位置にそれらのポートを追加します。これらのポートは、最上位ファイルにも追加し、user_logic にマップする必要があります。

architecture 宣言の後のコードは、ほとんどカスタム コードです。

必要な内部信号および定数を宣言した後、デザインの最初のブロックで単純なカウンタを駆動します。このカウンタから、次の 2 つの出力信号が取り出されます。

- PWM アップデート クロック (約 1KHz)
- LED アップデート クロック (約 4Hz)

クロック レートの変更

これらのクロック レートを変更するには、定数 PWM_tap および slow_clock_tap を変更します。

decode プロセスは、IPIC からのインターフェイス信号から適切な関数を選択します。カスタム ブロックへの書き込みは、Bus2IP_WrCE(0) がアクティブ (High) のときに実行されます。デコードロジックにアドレス信号を追加することにより、次の動作をインプリメントします。

- オフセット 0x00 に書き込み：すべての LED をオフ
- オフセット 0x04 に書き込み：すべての LED をオン
- オフセット 0x08 に書き込み：LED の輝度は 2 乗関数駆動信号を使用して調整
- オフセット 0x0c に書き込み：LED の輝度は線形関数駆動信号を使用して調整
- オフセット 0x1x に書き込み：LED の輝度を定数 (0 ~ 0xFF) に設定

PWM プロセスは、slow_clock のアップデート レートに基づいて駆動信号をアップデートします。

- 最初の case 文は、2 乗関数を使用して駆動値をアップデートします。
- 2 番目の case 文は、線形に駆動値をアップデートします。

自由に駆動値を変更してみてください。16 個の駆動値が使用されます。

LED は、PWM で生成された駆動信号で駆動されます。駆動信号のデューティ サイクルは、0% (駆動なし) からほぼ 100% (0xFF または 255) です。

247 行目付近の LEDs(0 to 7) の代入は、回路に最後に記述された命令に基づいて LED を制御します。

説明したコードはどれも単純なので、自由に変更して試してみてください。ただし、256 行目以降のインターフェイス信号には、特定の動作が必要です。これらの信号を不正なロジックで駆動すると、カスタム pcore がバスの動作を阻害し、デバッグ中に予測されない結果が得られます。

IP2Bus_Data は、読み出し動作でプロセッサにより読み出されるバスです。PLB を正しく動作させるには、カスタム pcore から読み出しが実行されている場合を除き、このバスをすべて 0 で駆動する必要があります。リセットが 0 で Bus2IP_RdCE が 1 であれば、読み出しは正しくデコードされます。この条件が満たされた場合、カスタム回路で指定の値がバスに駆動されます。条件が満たされない場合は、0 が駆動されます。

このサンプル デザインでは、ペリフェラル アドレス マップ内のアドレスを読み出すと、次のような 32 ビット値が返されます。

ビット 0 ~ 15 : 0xF0F0
ビット 16 ~ 23 : LED 駆動レジスタに記述される 1 バイト値
ビット 24 ~ 27 : 0x0
ビット 28 ~ 31 : all_off (1 ビット)、run (1 ビット)、linear (2 ビット)

最後の信号 IP2Bus_WrAck および Bus2IP_WrCE(0) も重要です。IP2Bus_WrAck は書き込み確認信号で、カスタム ロジックから送信する必要があります。IP2Bus_WrAck は 1 サイクル間のみ High にします。カスタム ロジックで応答に wait ステートを追加する必要がある場合は、遅延させることができます。この例では wait ステート は不要なので、IP2Bus_WrAck を直接 Bus2IP_WrCE(0) に接続し、単純な wait ステート なしの応答を生成します。読み出し 確認信号のロジックも 同一で、必要に応じて wait ステートを追加できます。

pwm_lights pcore には C_INCLUDE_DPHASE_TIMER パラメータが含まれており、これを 1 に設定すると、ペリフェラルがバス要求に応答しない場合に自動バス タイムアウト 信号が生成されます。

この例では、データ位相タイマは含まれていません。このロジックを追加する場合は、pwm_lights ペリフェラルに C_INCLUDE_DPHASE_TIMER パラメータを追加し、値を 1 に設定します。このロジックを追加すると、未確認のバス転送は PLB クロックの 128 サイクル後にタイムアウトします。

最後に、IP2Bus_Error が定数ロジック 0 で駆動されており、エラーが発生していないことを示します。カスタム ペリフェラルが別の外部ロジックを待機する必要がある、タイムアウトする可能性がある場合は、ロジックで IP2Bus_Error を駆動してバス転送を終了できます。

プロセッサ システムへのカスタム IP の追加

pwm_lights.vhd および user_logic.vhd を変更したときに、テンプレート デザインに新しいポートを追加しました。MPD ファイルでポートまたはパラメータを変更した場合、CIP ウィザードをインポート モードで実行する必要があります。これにより、EDK へのインターフェイス ファイルである PSF ファイル (MPD および PAO) が再生成されます。インポート フローを完了したら、カスタム pcore をエンベデッド デザインに追加できます。

このチュートリアルを開始する前に、現在 IP 作成プロセスのどの段階であるかを確認します。

- CIP ウィザードを実行して pwm_lights ペリフェラルを作成し、バス インターフェイスを設定してテンプレート ファイルを生成しました。
- 次に、再び CIP ウィザードを使用してプロジェクトに pwm_lights を追加します。このプロセスで、pwm_lights が XPS の適切なディレクトリにインポートされ、MPD および PAO ファイルが生成されます。PSF ファイルの詳細は、次のサイトから『Platform Specification Format Reference Manual』を参照してください。

http://japan.xilinx.com/ise/embedded/edk_docs.htm

チュートリアル：CIP ウィザードを使用した XPS プロジェクトへの変更したファイルのインポート

1. CIP ウィザードを起動し、[Peripheral Flow] ページで [Import existing peripheral] をオンにし、XPS プロジェクトに既存のペリフェラルをインポートすることを選択します。
2. [Name and Version] ページで、[Name] ドロップダウン リストから [pwm_lights] を選択します。バージョンは必要ありませんが、[Use version] をオンにしてデフォルト値またはカスタムバージョン番号を使用します。
3. この名前の既存のペリフェラルを上書きするかどうかを確認する [Overwrite Existing Peripheral] ダイアログ ボックスが表示されたら、[Yes] をクリックします。
4. [Source File Types] ページで [HDL source files] をオンにします。RTL または既存のネットリストを使用して pcore を作成することもできます。カスタム ペリフェラルに資料を含めることもできます。

ここまでは簡単でしたが、この後インポート フローは複雑になります。

CIP ウィザードでは、さまざまな方法で作成された pcore をインポートできます。pcore を CIP ウィザードを使用して作成した場合は、PAO (Peripheral Analysis Order) ファイルを使用してソース ファイルを特定するのが最も簡単な方法です。

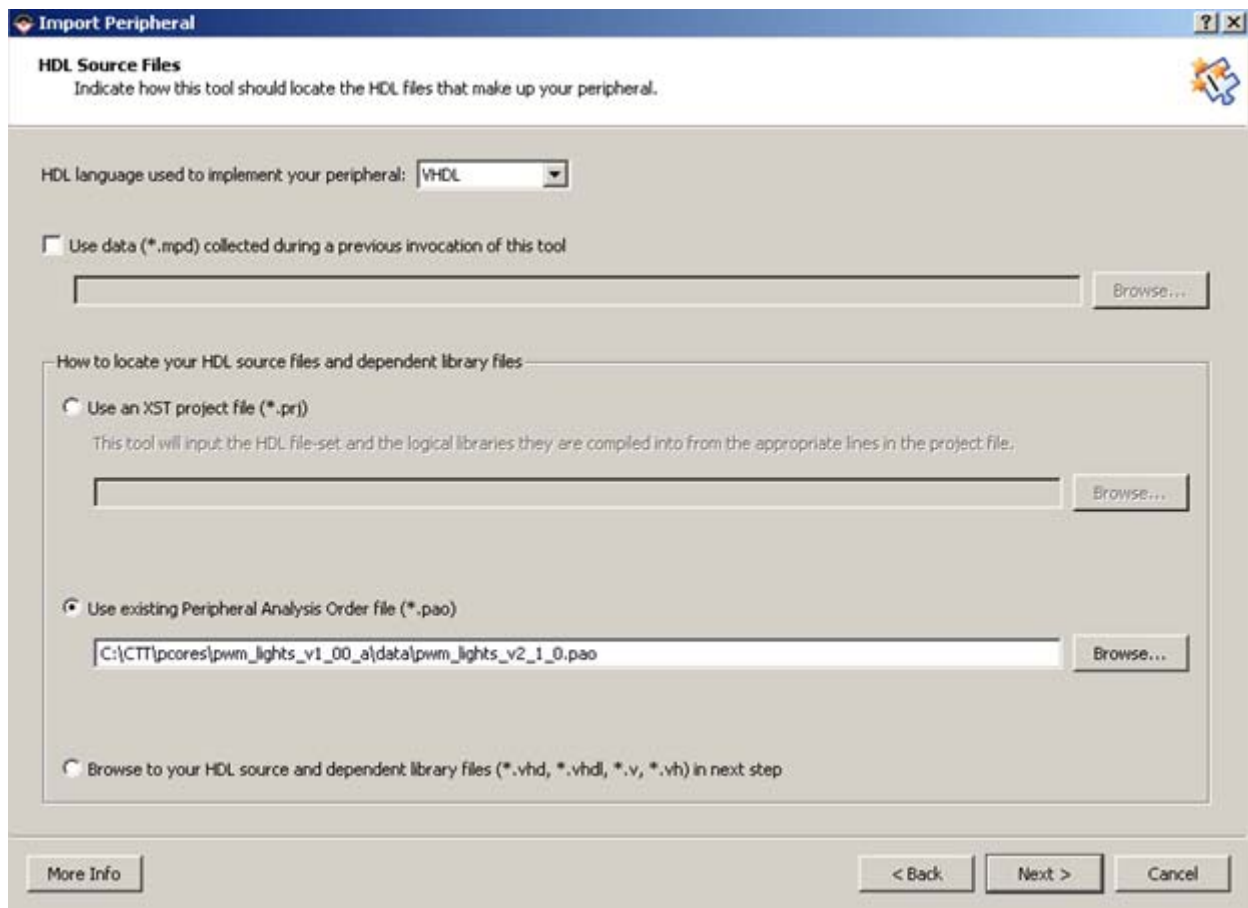


図 7-10 : [HDL Source Files] ページ

5. [HDL Source Files] ページで、[Use existing Peripheral Analysis Order file (*.pao)] をオンにします。
6. PAO ファイルを選択します。デフォルト では、[Browse] ボタンをクリックすると最上位 pcores ディレクトリ が開きます。PAO ファイルは、pwmLights_v1_00_a\data サブディレクトリ にあります。

PAO ファイルを使用して下位ライブラリを含む必要なソース ファイルを指定する場合は、さらにファイルやライブラリを追加する必要はありません。

多数のファイルで構成される複雑なペリフェラルをインポートする場合は、ライブラリパスおよび HDL ソース ファイル パスに必要なファイルおよびライブラリが含まれていることを確認してください。

メモ：ファイルを表示するには、[ファイルの種類] ドロップダウン リストでファイルの種類を変更する必要がある場合があります。

7. [HDL Analysis Information] ページで、リストの最後に user_logic.vhd および pwmLights.vhd が含まれていることを確認します。

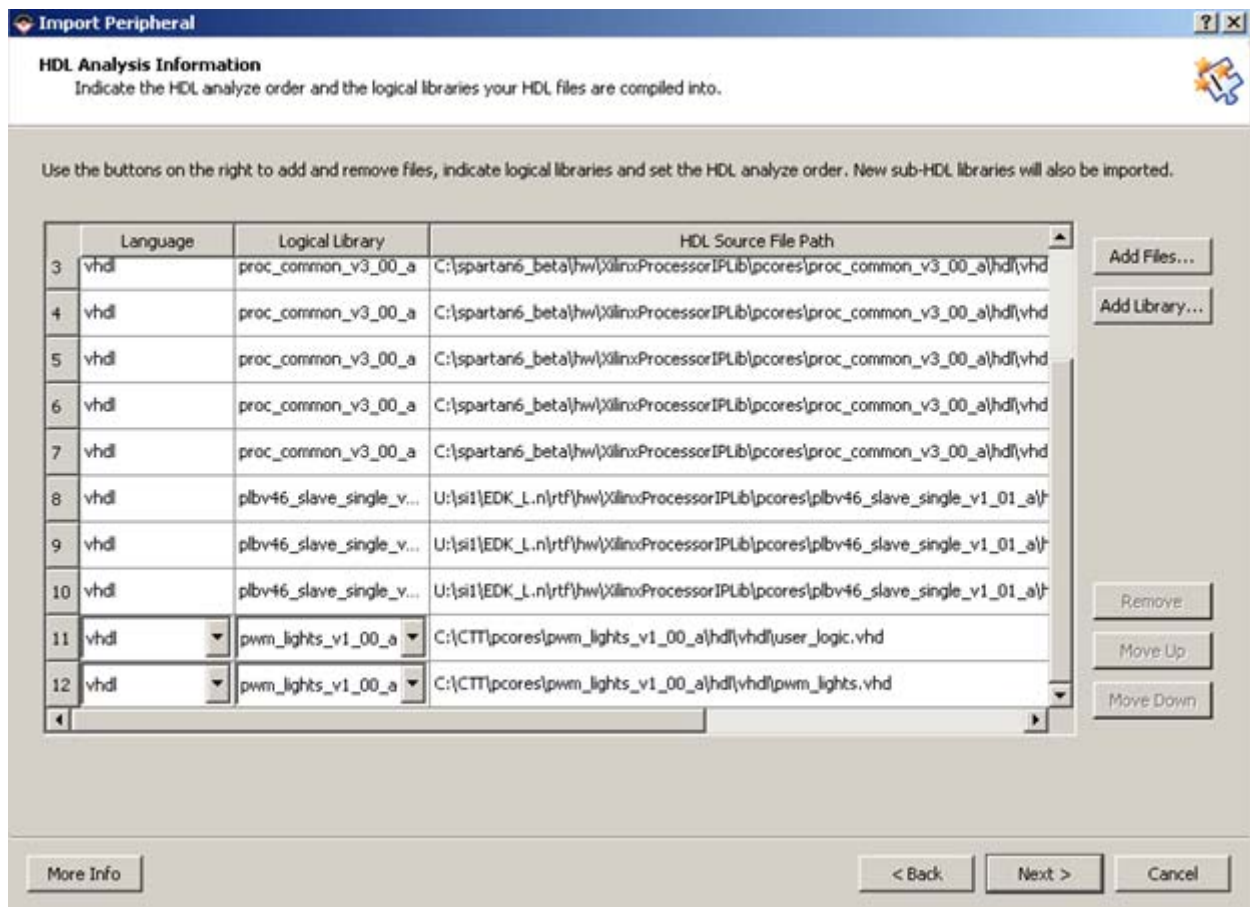


図 7-11 : [HDL Analysis Information] ページ

8. [Current Logical Library] ダイアログ ボックスが表示され、追加した VHDL ファイルがコンパイルされていないことが示される場合があります。[Next] をクリックし、ウィザードでこれらのファイルがコンパイルされるようにします。

9. [Bus Interface] ページで、適切なバスを選択します。pwm_lights ペリフェラルは PLBv46 スレーブ (SPLB) なので、[PLBV46 Slave (SPLB)] をオンにします。

[SPLB : Port] ページに、このデザインで使用される PLBv46 バス信号がリストされます。これらの信号および関連付けられているバス プロトコルは、すべて CIP ウィザードで自動的に処理されます。

ほとんどの場合、特に CIP ウィザードを使用してコアを作成している場合は、必要な信号がすべて含まれているかを解析するのに時間を費やす必要はありません。

別の方法でコアを作成した場合、複雑なバス インターフェイスやカスタム バス インターフェイスが含まれる場合は、バス信号を解析するのにこのページが有益です。

pwm_lights ペリフェラルは、pcore をデザインに含めるときに、XPS が割り当てる 1 つのアドレス範囲にマップされます。複雑なペリフェラルでは、メモリ ブロックやデコード範囲が含まれる場合もあります。

10. [SPLB : Parameter] ページで、デフォルト値をそのまま使用します。

pwm_lights ペリフェラルには、割り込みソースは含まれていません。

11. [Identify Interrupt Signals] ページで、[Select and configure interrupt(s)] をオフにします。

複雑なペリフェラルでは、多数のパラメータを使用し、注意深く PLB バスの動作を制御する必要があります場合があります。[Parameter Attributes] ページでは、パラメータ設定を表示および変更できます。

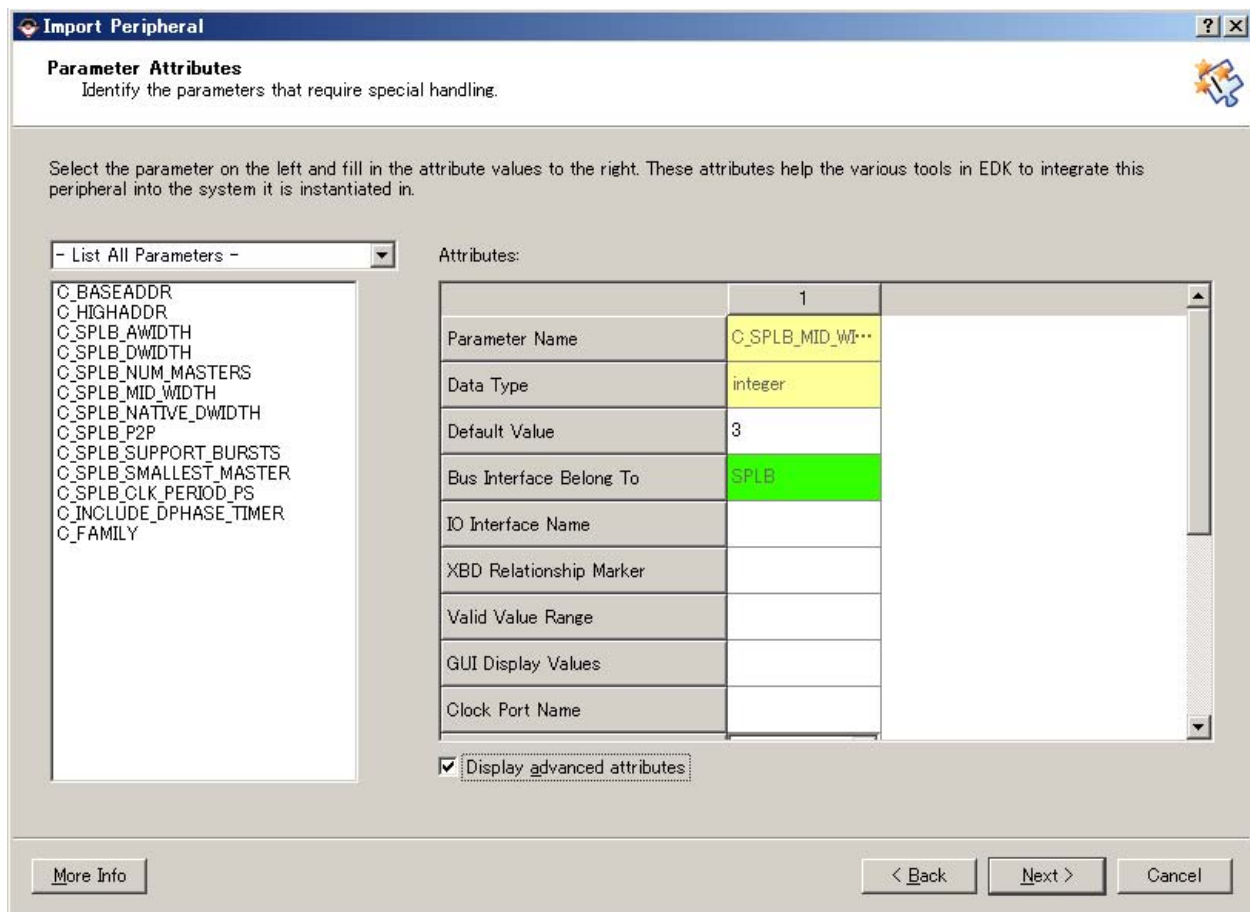


図 7-12 : [Parameter Attributes] ページ

ドロップダウン リストから、カスタム pcore のパラメータのみを表示するか ([List User Parameters only])、選択したバス インターフェイスのパラメータのみを表示するか ([List Bus Interface Parameters only])、両方を表示するか ([List All Parameters]) を選択できます。

pcore パラメータは、CIP ウィザードのこれまでのページで生成されています。バス インターフェイス パラメータは自動的に生成されています。

このペリフェラルでは、変更は不要です。

12. [Port Attributes] ページで [LEDs] をクリックし、[Display advanced attributes] をオンにして属性をすべて表示します。このようにすると、ポート属性をより詳細に制御できます。属性は、MPD ファイルに記述されます。

ドロップダウン リストで [List All Ports] を選択すると、カスタム pcore とエンベデッド プロセッサ サブシステムの接続に使用されるすべてのポートが表示されます。PLB 信号が多数ありますが、CIP ウィザードですべてのポートの属性が自動的に設定されます。

13. このページを参照した後、[Next] をクリックして最後のページに進みます。[Finish] をクリックすると、インポート操作は完了です。

**pwm_lights pcore を
プロジェクトに追加**

[IP Catalog] タブの [Project Local pcores] の下に、カスタム ペリフェラルがリストされています。

pwm_lights をデザインに追加する前に、既存のデザインに 1 つだけ変更を加える必要があります。評価ボードの 8 個の LED は、GPIO 出力に接続されています。pwm_lights でこれらの LED を駆動するようにするので、LEDs_8Bit pcore をデザインから削除する必要があります。

14. [System Assembly View] で [LEDs_8Bit] を右クリックし、[Delete Instance] をクリックします。[Delete IP Instance] ダイアログ ボックスが表示されます。

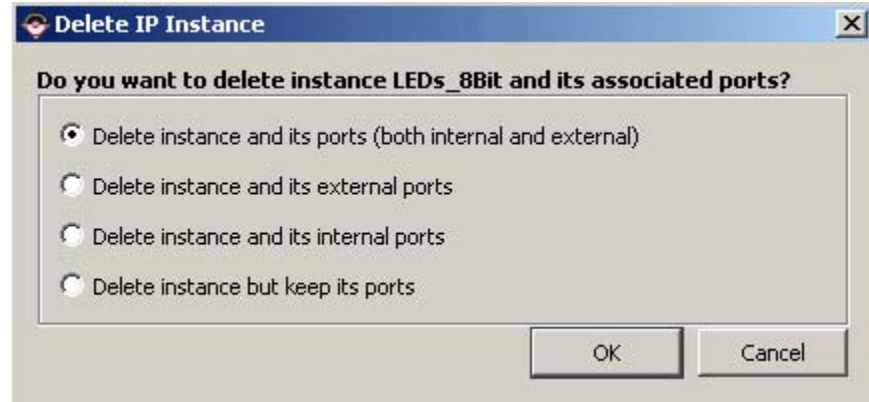


図 7-13 : [Delete IP Instance] ダイアログ ボックス

15. ここでは、デフォルト設定を使用します。外部ポートは、デザインに手動で追加します。
16. [IP Catalog] タブで [PWM_LIGHTS] を右クリックし、[Add IP] をクリックします。
IP が [System Assembly View] に追加されます。[Bus Interfaces] タブで確認できます。

17. バス接続パネルで、PLB バスをクリックしてバスを接続します。

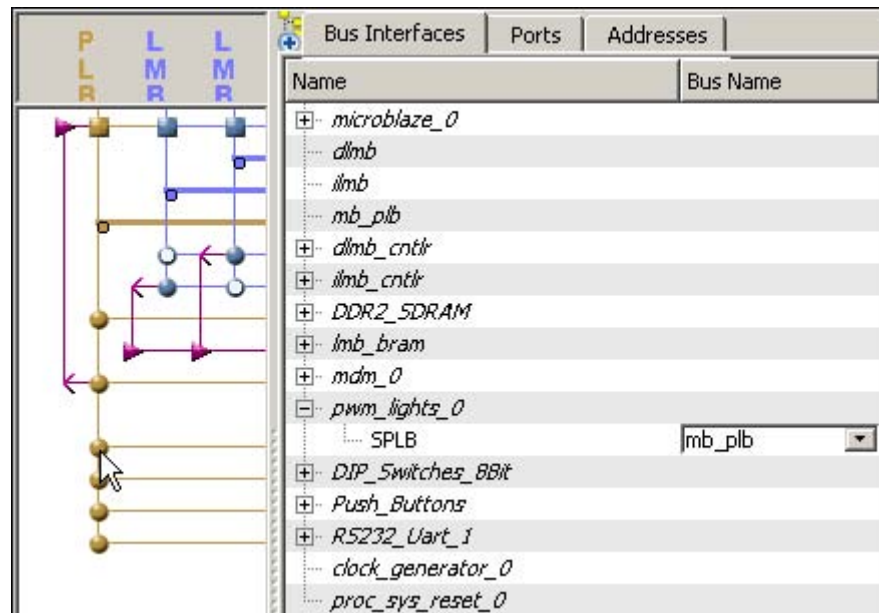


図 7-14：新規 IP のバスを接続

これで、pwm_lights コアがエンベデッド システムに追加されました。次に、pwm_lights と評価ボード上の LED の外部接続を追加する必要があります。

18. [Ports] タブをクリックし、[pwm_lights_0] を展開表示して、[Net] 列のドロップダウン リストから [Make External] を選択します。

デフォルト名である pwm_lights_0_LEDs[0:7] がネット名として使用されます。[Ports] タブの 1 番上にある [External Ports] を展開表示すると、pwm_lights_0_LEDs_pin[0:7] というポート名が表示されます。

19. pwm_lights_0 のドロップダウン ボックスをクリックすると、割り当てられたネットおよびピン名を変更できます。また、MHS ファイルを手動で編集することも可能です。ここでは、デフォルト名をそのまま使用します。

次に、pwm_lights pcore のアドレスを生成します。

20. [Addresses] タブをクリックします。[Unmapped Addresses] の下に pwm_lights_0 が表示されていない場合は、[Project] → [Rescan User Repositories] をクリックします。

21. [Addresses] タブで [Generate Addresses] をクリックします。pwm_lights pcore が 0xC4600000 ~ 0xC460FFFF のアドレス範囲に割り当てられます。

22. DDR2_SDRAM のアドレス範囲が 0x88000000 ~ 0x8fffffff であることを確認します。このアドレスが変更されている場合は、元の値に戻します。

単純なペリフェラルが 64KB のアドレス空間に割り当てられるのはおかしいと思われるかもしれませんが、大きいアドレス空間ではデコードするアドレス ラインが少なくて済みます。FPGA では、多入力のデコーダはカスケードされたルックアップ テーブルとしてインプリメントされます。

カスケード 段が増えると、動作周波数は低くなります。広いアドレス範囲に割り当てると、FPGA インプリメンテーションの動作が速くなります。

23. 最後に、UCF ファイルで LED 出力を適切な FPGA ピンに割り当てます。

24. [Project] タブをクリックし、[UCF File] をダブルクリックして開きます。
25. fpga_0_LEDs_8Bit_GPIO_IO_O を検索します。GPIO pcore を削除しましたが、UCF ファイルにはこれらのピン割り当てが残っています。pcore を削除しても、UCF ファイルは自動的にアップデートされないので注意してください。
26. 8 個すべての fpga_0_LEDs_8Bit_GPIO_IO_O_pin を pwm_lights_0_LEDs_pin に置き換え、UCF ファイルを保存します。

これで、カスタム pcore が完成しました。

デザインのエクスポート とビットストリームの 生成

次に、ハードウェア デザインをエクスポートして新しいビットストリームを生成し、ハードウェアでこの pcore をテストします。

1. XPS で、[Project] → [Export Hardware Design to SDK] をクリックします。デフォルトのディレクトリをそのまま使用し、[Export Only] をクリックします。
2. エクスポートが完了したら、XPS を閉じて ISE に戻り、[Generate Programming File] をダブルクリックします。
3. SDK を起動します。起動するのに少し時間がかかる場合があります。ハードウェア プラットフォームの変更が認識されると、[Hardware Design Changes Detected] ダイアログ ボックスが表示されます。
4. [Show Hardware Changes] をクリックします。

次の図に示すファイルがブラウザで開きます。

Module Level Changes		
PROCESSOR: microblaze_0		
PROCESSOR PARAMETERS		
NAME	OLD	NEW
VERSION	7.20.a	7.20.a
C_SCO	0	0
C_DATA_SIZE	32	32
C_DYNAMIC_BUS_SIZING	1	1
...
PROCESSOR PERIPHERALS		
INSTANCE	TYPE	STATUS
dlmb_cntlr	lmb_bram_if_cntlr	(no change)
ilmb_cntlr	lmb_bram_if_cntlr	(no change)
RS232_Uart_1	xps_uartlite	(no change)
Push_Buttons	xps_gpio	(no change)
DIP_Switches_8Bit	xps_gpio	(no change)
DDR2_SDRAM	mpmc	(no change)
mdm_0	mdm	(no change)
- LEDs_8Bit	xps_gpio	(removed)
+ pwm_lights_0	pwm_lights	(added)
PROCESSOR MEMORY MAP		
PERIPHERAL	OLD	NEW
dlmb_cntlr	0x00000000-0x00003fff (rw-)	0x00000000-0x00003fff (rw-)
ilmb_cntlr	0x00000000-0x00003fff (--x)	0x00000000-0x00003fff (--x)
RS232_Uart_1	0x84000000-0x8400ffff (rwx)	0x84000000-0x8400ffff (rwx)
Push_Buttons	0x81400000-0x8140ffff (rwx)	0x81400000-0x8140ffff (rwx)
DIP_Switches_8Bit	0x81440000-0x8144ffff (rwx)	0x81440000-0x8144ffff (rwx)
* DDR2_SDRAM	0x88000000-0x8fffffff (rwx)	0xc8000000-0xcfffffff (rwx)
mdm_0	0x84400000-0x8440ffff (rwx)	0x84400000-0x8440ffff (rwx)
- LEDs_8Bit	0x81420000-0x8142ffff (rwx)	(removed)
+ pwm_lights_0	-	0xc4600000-0xc460ffff (rwx)

図 7-15 : ハードウェア デザイン変更サマリ

LED を駆動する `xps_gpio pcore` が削除され、`pwm_lights` が追加されています。SDK で `system.xml` ファイルに加えたハードウェア プラットフォームの変更が認識され、表示されます。

5. ブラウザ ウィンドウを閉じます。

[Hardware Design Changes Detected] ダイアログ ボックスで [SDK Actions] タブをクリックすると、`gpio` ドライバが削除され、`pwm_lights` ドライバが追加されたことが表示されます。

6. [Hardware Design Changes Detected] ダイアログ ボックスを閉じます。

SDK に [C/C++] パースペクティブが表示されます。

7. 第 5 章および第 6 章で使ったワークスペースを使用する場合は、`standalone` プラットフォームが既に存在しています。

新しいワークスペースを使用して SDK を起動した場合は、`cip_wizard_platform` という `standalone` ソフトウェア プラットフォームを作成します。

8. LEDS という Managed Make C アプリケーション プロジェクトを、[Empty Application] サンプル アプリケーションを選択して作成します。

このマニュアルの添付ファイルは、Adobe® Acrobat Reader の左下にあるクリップ アイコンをクリックすると表示されます。

9. `leds.c` という名前の新規ソース ファイルを追加します。このマニュアルに添付されている `leds.c` ファイルの C コードをコピーして貼り付け、保存します。コンパイル エラーが発生します。このエラーは、オブジェクト コードが選択されたメモリ空間よりも大きいことが原因で発生します。これを調べてみます。

10. [Projects] タブで [`leds.c`] を右クリックし、[Generate Linker Script] をクリックします。

[Linker Script Generator] ダイアログ ボックスが表示されます。テキスト、ヒープ、スタックなど、すべてがブロック RAM (`ilmb_cntl1r_d1mb_cntl1r`) に割り当てられています。この割り当てを、メモリ空間が大きい DDR2 RAM に変更します。

11. [Linker Script Generator] ダイアログ ボックスで、すべてのコード セクションに対してドロップダウン リストから [DDR2_SDRAM_MPMC_BASEADDR] を選択します。ヒープおよびスタックのサイズが 0x1000 に設定されていることを確認します。

これで、コードが正しくコンパイルされます。作成された ELF ファイルは約 93K で、16K のブロック RAM よりかなり大きくなっています。

12. [Initialization ELF] を [BootLoop] に設定してビットストリームをダウンロードします。

13. `leds.elf` をデバッグします。

14. ターミナル ウィンドウが開いて `leds.elf` を実行します。アプリケーション コードを実行すると、ターミナル ウィンドウにデバッグ オプションが表示されます。

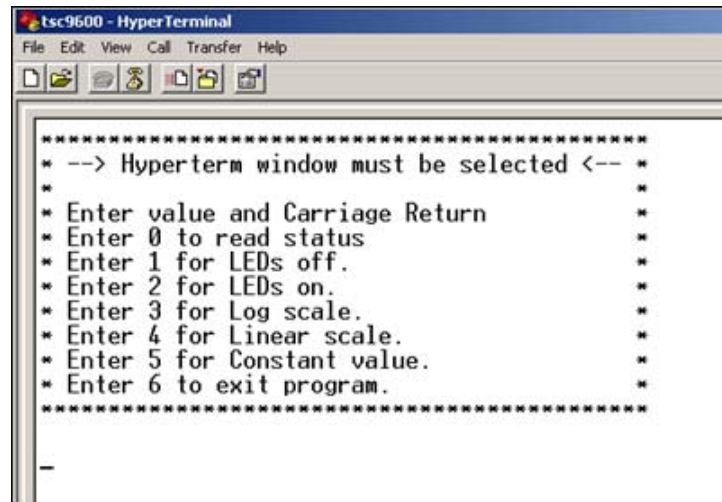


図 7-16 : HyperTerminal に表示されたデバッグ オプション

値を入力します。

15. さまざまな値を入力し、LED が予測どおりに動作するかどうかを確認します。

実行された処理

CIP ウィザードを使用してカスタム IP を作成しました。タスクを完了するのに多数の手順を実行する必要がありましたが、これで手順を理解でき、今後 CIP ウィザードを効率的に使用できるようになったはずです。必要に応じて、この章を参照してください。

次の操作

次の章では、デュアル プロセッサ デザインを作成し、EDK でデバッグします。

デュアル プロセッサ デザインの作成とデバッグ

MicroBlaze™ プロセッサはソフト マイクロプロセッサであり、FPGA に収まるだけの数の MicroBlaze プロセッサを含めることができます。XPS (Xilinx® Platform Studio) の Base System Builder (BSB) でデュアルプロセッサシステムを作成するのは、1つの MicroBlaze プロセッサを含むシステムを作成するのとはほとんど同じです。また、ソフトウェア開発キット (SDK) を使用すると、1 つまたは複数の MicroBlaze プロセッサを含むエンベデッド システムを簡単にデバッグできます。

BSB を使用したデュアル プロセッサ システムの作成

PowerPC® プロセッサを含む FPGA を使用する 場合、同じ概念を適用できます。BSB ウィザードを使用して、デュアル PowerPC デザイン (FPGA に PowerPC プロセッサが 2 つ含まれている 場合)、PowerPC プロセッサと MicroBlaze プロセッサを 1 つずつ含むデザインを作成できます。Spartan®-3A DSP ボードを使用する場合、BSB ウィザードでデュアル MicroBlaze プロセッサ デザインを作成し、XPS でさらに MicroBlaze プロセッサを追加できます。

チュートリアル：2 つの MicroBlaze プロセッサを含むエンベデッド システムの作成

このチュートリアルでは、Base System Builder ウィザードおよび ISE® Project Navigator を使用して、11 ページの「新規プロジェクトの作成」と同様の方法でデュアル プロセッサ システムを作成し、インプリメントします。

メモ：1 つのエンベデッド プロジェクトに複数のプロセッサを含むことができます。このチュートリアルでは、エンベデッド プロジェクトに 2 つの MicroBlaze プロセッサを含めます。

1. 1 つのエンベデッド プロセッサ ソースを含む ISE プロジェクトを作成します。

XPS が開くのを待ちます。これには少し時間がかかる場合があります。

2. 「This project appears to be a blank project. Do you want to create a Base System using the BSB Wizard?」(このプロジェクトは空です。BSB ウィザードを使用して基本システムを作成しますか) というメッセージが表示されたら、[Yes] をクリックします。

3. BSB で、次の表の指示に従ってプロジェクトを作成します。表に設定またはコマンドがない場合は、デフォルト値をそのまま使用します。

ウィザードのページ	システム特性	設定または使用するコマンド
[System Configuration]	システムのタイプ	[Dual-Processor System] をオンにします。
[Processor Configuration]	ローカル メモリ ([Local Memory])	両方のプロセッサに対して [8 KB] を選択します。
[Peripheral Configuration]	<ul style="list-style-type: none"> プロセッサ 1 ペリフェラル ([Processor 1 (MicroBlaze) Peripherals]) 共有ペリフェラル ([Shared Peripherals]) プロセッサ 2 ペリフェラル ([Processor 2 (MicroBlaze) Peripherals]) 	<ul style="list-style-type: none"> デフォルトのリストから次のペリフェラルを削除します。 - DIP_Switches_8Bit - Ethernet_MAC - LEDs_8Bit プロセッサ 1 の残りのペリフェラルは、DDR2_SDRAM、RS232_Uart_1、dlmb_cntlr、ilmb_cntlr です。 xps_mutex_0 を削除します。残りのペリフェラルは xps_mailbox_0 です。 デフォルトのリストから次のペリフェラルを削除します。 - Push_Buttons - SPI_FLASH プロセッサ 2 の残りのペリフェラルは、dlmb_cntlr_1 および ilmb_cntlr_1 です。

4. デザインが完成したら、XPS の [System Assembly View] でシステムを見てみます。
- 2 つのエンベデッド プロセッサ システムは完全に独立しており、それぞれ独自のメモリ マップがあります。例外は xps_mailbox_0 ペリフェラルです。このペリフェラルはデュアル ポート RAM で、1 つのポートは 1 つのプロセッサの PLBv46 バスに接続されており、もう 1 つのポートはもう 1 つのプロセッサの PLBv46 バスに接続されています。[System Assembly View] で xps_mailbox_0 ペリフェラルを右クリックし、[View PDF Datasheet] をクリックすると、このペリフェラルのデータシートが表示されます。
5. [Project] → [Export Hardware Design to SDK] をクリックし、system.xml ファイルを SDK にエクスポートします。
6. デフォルトのディレクトリをそのまま使用し、[Export Only] をクリックします。
7. ISE Project Navigator に戻り、ISE プロジェクトに UCF ファイルを追加します。
- メモ：ISE プロジェクトへの UCF ファイルの追加に関する詳細は、第 4 章「エンベデッド プラットフォームの操作」を参照してください。
8. [Generate Programming File] をダブルクリックしてデザインをインプリメントし、ビットストリームを生成します。

これで、ターゲット ハードウェアにダウンロードするビットストリームと SDK で使用する system.xml ファイルの準備ができました。

チュートリアル：SDK を使用したデュアル プロセッサ システム用のソフトウェアの開発

このチュートリアルでは、2つのエンベデッド プロセッサ用のソフトウェアの開発およびデバッグに使用する SDK プロジェクトを作成します。

SDK を使用してデュアル プロセッサ システムをデバッグする手順は、第 5 章と第 6 章のチュートリアルで示したシングル プロセッサ システムのデバッグとほぼ同じです。

これまでに説明したとおり、SDK でソフトウェアを開発するには、ソフトウェア プラットフォームを作成し、C または C++ アプリケーション プロジェクトを作成します。この手順は、システムにプロセッサがいくつ含まれていても同じです。ソフトウェア プラットフォームを作成する際、プラットフォームで使用するプロセッサを指定する必要があります。

1. SDK を起動します。
2. [Workspace Launcher] ダイアログ ボックスが表示されたら、Dual_Processor_Workspace という名前のワークスペースを作成し、任意のディレクトリに保存します。
3. [New Hardware Specification File] ダイアログ ボックスで、エクスポートした system.xml ファイルを指定します。デフォルトのプロジェクト ロケーションを使用した場合は、このファイルは <ISE Project Name>\system\SDK\SDK_Export\hw\system.xml にあります。

[C/C++] パースペクティブが表示されます。SDK でエンベデッド システムに 2 つの MicroBlaze プロセッサがあることが認識され、次の図のように表示されます。

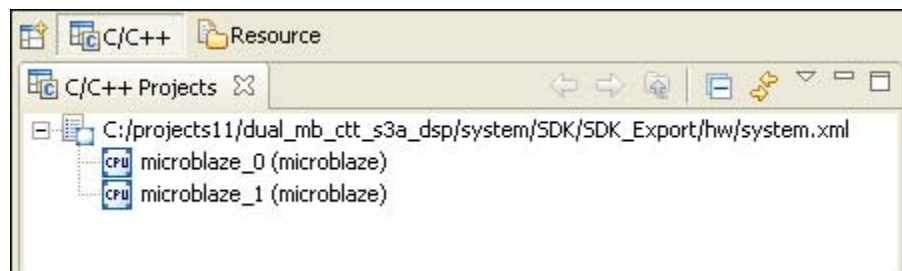


図 8-1：エンベデッド システムに表示される 2 つの MicroBlaze プロセッサ

4. [File] → [New] → [Project] をクリックし、[Software Platform] を選択して、次の設定を使用してソフトウェア プラットフォーム プロジェクトを作成します。
 - ◆ [Project Name]：MicroBlaze_Platform_0
 - ◆ [Processor]：[microblaze_0 (microblaze)]
 - ◆ [Platform Type]：[standalone]
 - ◆ [Project Location]：[Use default] をオン
5. 同じ手順を使用して、2 つ目のソフトウェア プラットフォーム プロジェクトを次の設定で作成します。
 - ◆ [Project Name]：MicroBlaze_Platform_1
 - ◆ [Processor]：[microblaze_1 (microblaze)]
 - ◆ [Platform Type]：[standalone]
 - ◆ [Project Location]：[Use default] をオン

次の図に示すように、各 MicroBlaze プロセッサに 1 つずつソフトウェア プラットフォームが表示されます。

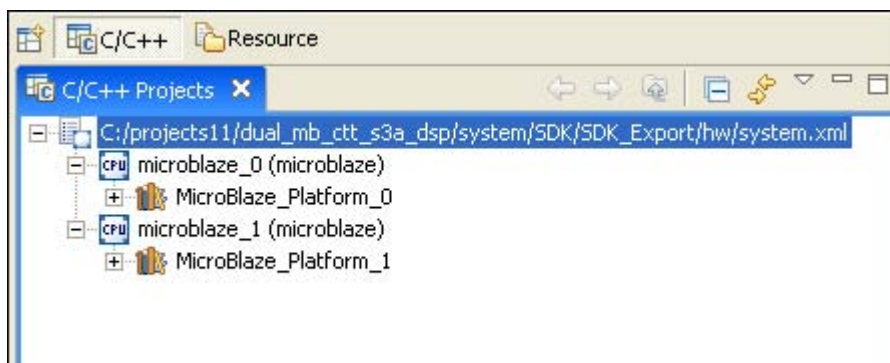


図 8-2 : MicroBlaze プロセッサと関連付けられたソフトウェア プラットフォーム

次に、各プロセッサに対して Managed C アプリケーション プロジェクトを作成します。この例では、各プロセッサに対して Hello World プロジェクトを作成し、変更します。

6. [File] → [New] → [Managed Make C Application Project] をクリックし、次の設定を使用して Managed Make C アプリケーション プロジェクトを作成します。
 - ◆ [Project Name] : hello_world_0
 - ◆ [Software Platform] : [MicroBlaze_Platform_0]
 - ◆ [Project Location] : [Use Default Location for Project] をオン
 - ◆ [Sample Applications] : [Hello World] を選択
7. 同じ手順を使用して、2 つ目の Managed Make C アプリケーション プロジェクトを次の設定で作成します。
 - ◆ [Project Name] : hello_world_1
 - ◆ [Software Platform] : [MicroBlaze_Platform_1]
 - ◆ [Project Location] : [Use Default Location for Project] をオン
 - ◆ [Sample Applications] : [Hello World] を選択

次の図に示すように、各プロセッサに 1 つずつサンプル C アプリケーションが作成されました。

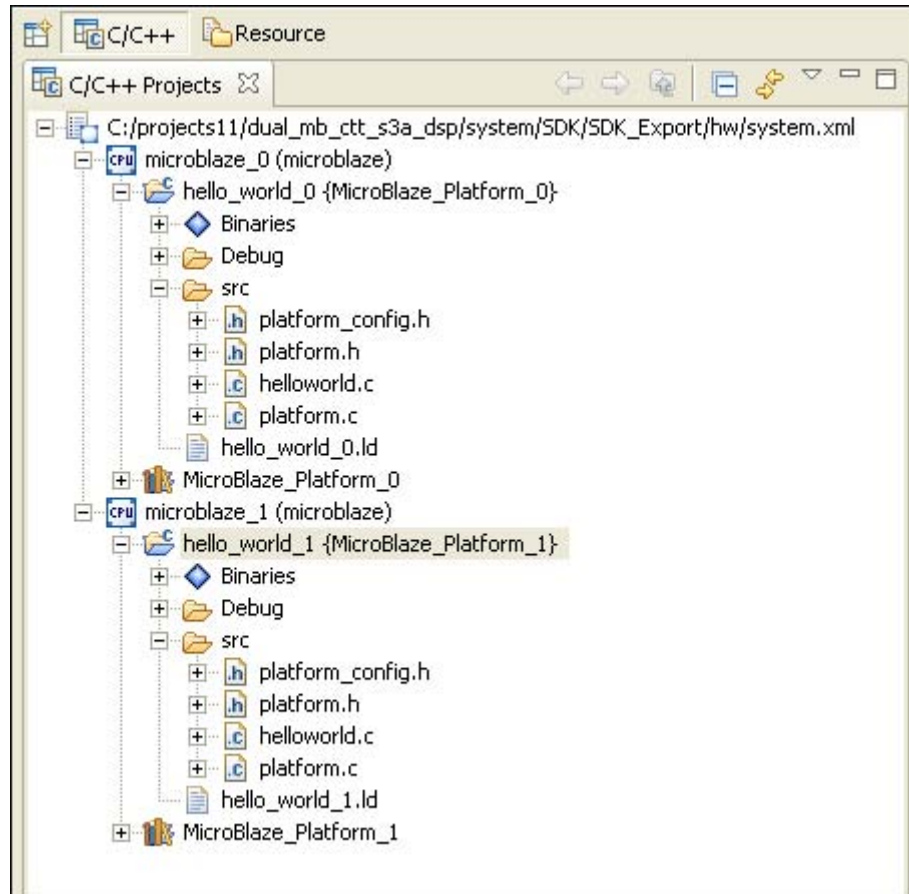


図 8-3 : MicroBlaze プロセッサと関連付けられた Managed C アプリケーション プロジェクト

8. helloworld.c ファイルを変更して、どちらのプロセッサが実行されているかが示されるようにします。MicroBlaze_Platform_0 プロジェクトの helloworld.c ファイルを開き、コードを次のように変更します。

```
print("Hello World\n\r");
```

上記のコードを次のコードで置換します。

```
print("Hello From Processor 0!\n\r");
```

9. MicroBlaze_Platform_1 プロジェクトの helloworld.c ファイルも同様に変更します。
10. 各ファイルを保存します。保存すると、SDK でファイルが自動的に構築されます。[Console] ビューの出力を確認してください。

```
***** Determining Size of ELF File *****
```

```
mb-size hello_world_1.elf
text data bss dec hexfilename
1958 296 2090 4344 10f8hello_world_1.elf
```

```
Build complete for project hello_world_1
```

プログラムには、アプリケーションを実行するのに十分なメモリが必要です。

このサンプル デザインでは、MicroBlaze_Platform_0 のみが外部 DDR2 メモリにアクセス可能で、MicroBlaze_Platform_1 は 8KB のオンチップ ブロック RAM にしかアクセスできません。
[Console] ビューの出力からわかるように、hello_world_1.elf のサイズは 4.344KB であり 8KB 以下なので、メモリのサイズは十分です。

チュートリアル：ソフトウェア プラットフォーム設定の変更

MicroBlaze_Platform_0 プロセッサは、stdin および stdout ペリフェラルに UART を使用しますが、MicroBlaze_Platform_1 プロセッサには UART がないので、stdin および stdout ペリフェラルに XMD と MDM-UART を使用する必要があります。そのため、MicroBlaze_Platform_1 のソフトウェア プラットフォーム設定の変更が必要な場合があります。

1. [Tools] → [Software Platform Settings] をクリックし、MicroBlaze_Platform_1 プロセッサの [Software Platform Setting] ダイアログ ボックスを開きます。
2. [OS and Libraries] ページで、stdin および stdout の設定を確認します。XMD を実行するハードウェアを MDM にする必要があります。デフォルト値は mdm_0 であり、正しい設定になっていることがわかります。

MicroBlaze_Platform_0 プロセッサの設定を確認すると、stdin および stdout が xps_uartlite に設定されているはずです。

チュートリアル：1 つの SDK [Debug] パースペクティブを使用した複数のプロセッサのデバッグ

デザインの各エンベデッド プロセッサには、個別のバイナリ ELF ファイルが必要です。ファイルには、プロセッサ名に基づいて自動的に名前が付けられます。たとえば、MicroBlaze_Platform_0 プロセッサのバイナリ ファイルの名前は hello_world_0.elf であり、MicroBlaze_Platform_1 プロセッサのバイナリ ファイルの名前は hello_world_1.elf です。

各バイナリ ファイルは、個別にダウンロードできます。その前に、デュアル プロセッサ システム デザインのビットストリームをターゲット ハードウェアにダウンロードする必要があります。

1. [Tools] → [Program FPGA] をクリックし、次の場所からビットストリームとブロック メモリ マップ ファイルを選択します。
 - ◆ <ISE Project Name>\system.bit
 - ◆ <ISE Project Name>\edkBmmFile_bd.bmm
2. 各プロセッサの [Initialization EFL] が [BootLoop] に設定されていることを確認し、[Save and Program] をクリックします。

ターゲット ハードウェアがデュアル プロセッサ デザインのビットストリームでプログラムされます。

次に、各プロセッサに ELF ファイルをダウンロードします。
3. [C/C++ Projects] ビューの [hello_world_0 {MicroBlaze_Platform_0}] フォルダにある [hello_world_0.elf] を右クリックし、[Debug As] → [Debug on Hardware] をクリックします。

MicroBlaze_Platform_0 プロセッサの [Debug] パースペクティブが開きます。

4. [C/C++] パースペクティブに戻り、同じ手順を使用して `hello_world_1.elf` ファイルを `MicroBlaze_Platform_1` にダウンロードします。

再び [Debug] パースペクティブが開きます。[Debug] ビューに 2 つのデバッグ タスクがあることを確認します。

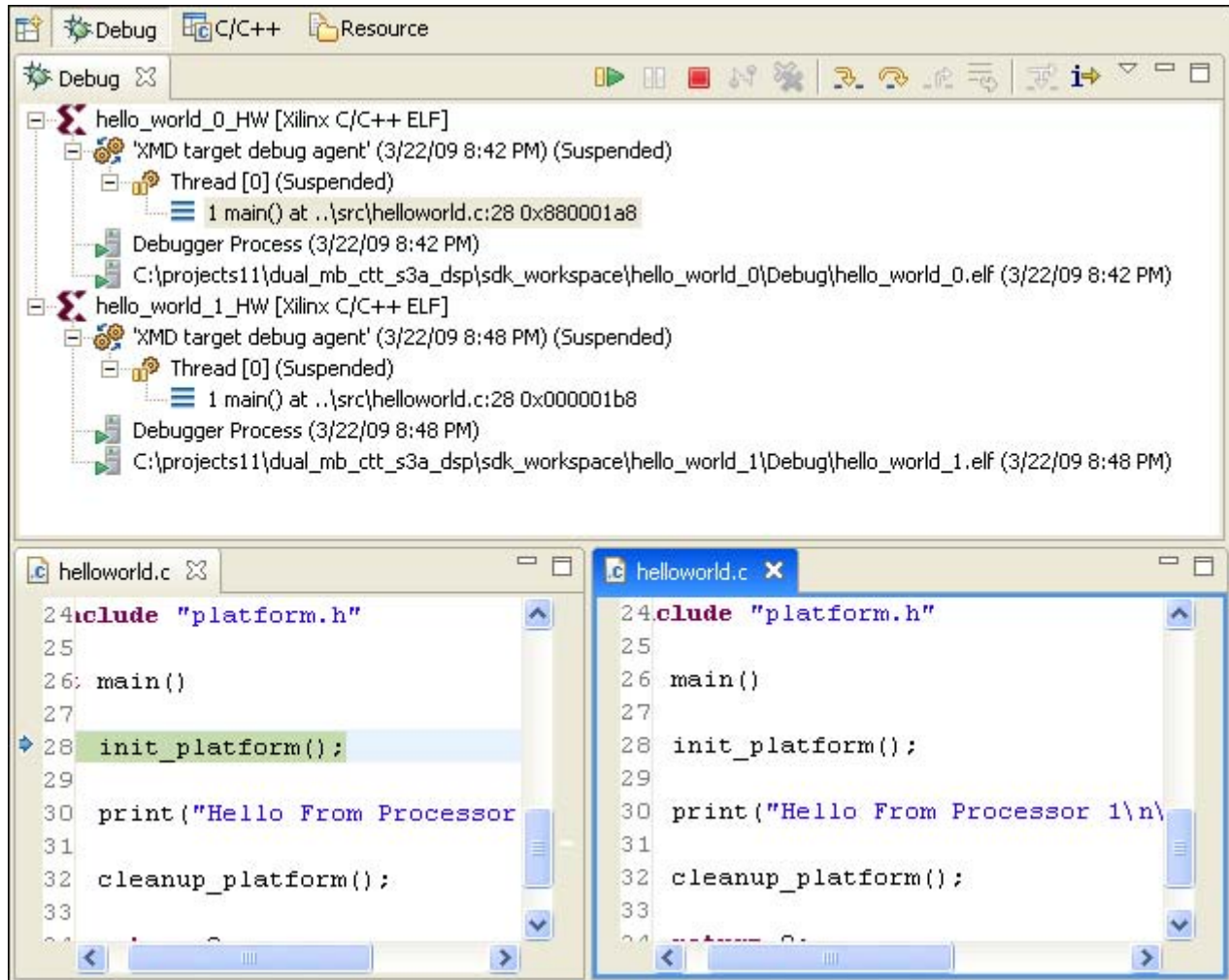


図 8-4 : [Debug] パースペクティブ

5. これらのプログラムをデバッグする前に、RS232 ケーブルでコンピュータとターゲット ボードを接続し、ターミナル ウィンドウでコンソール I/O を確認します。
6. [XMD Console] ビューで「**terminal**」と入力し、ターミナル I/O を MDM に送信します。このウィンドウを使用して、`MicroBlaze_Platform_1` の出力を観察できます。

メモ : [XMD Console] ビューが表示されていない場合は、[Window] → [Show View] → [Other] をクリックし、[Xilinx] → [XMD Console] を選択します。

7. [Debug] ビューで `hello_world_0` または `hello_world_1` の呼び出しスタックをハイライトし、[Run] → [Resume] をクリックします。

メモ : どちらの呼び出しスタックも「1 main() at...」となっているはずです。

ターミナル ウィンドウまたは [XMD Console] ビューのプロセッサ出力表示は、次の図のようになります。

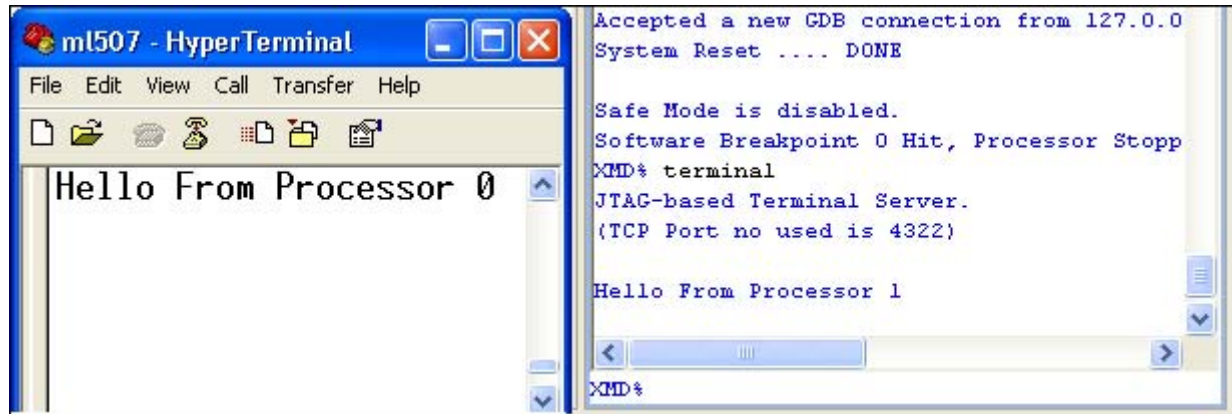


図 8-5：ターミナル ウィンドウおよび [XMD Console] ビューのプロセッサ出力

SDK での複数プロセッサ デザインのデバッグは、1 つのプロセッサ デザインのデバッグと同様に実行できます。これは単純な例でした。コードの 1 行ずつの実行、ブレークポイントの設定、レジスタおよびメモリの確認など、その他のソフトウェア開発タスクも SDK で実行できます。

Project Navigator での ModelSim を使用したシミュレーション

このマニュアルのチュートリアルでは、すべての操作を ISE® Project Navigator から開始しました。エンベデッド デザインのシミュレーションも Project Navigator で実行できます。

シミュレーション

第 7 章「独自の IP の作成」のチュートリアルで作成したデザインを使用します。

チュートリアル：エンベデッド デザインのシミュレーション

このチュートリアルでは、XPS で生成された VHDL ファイルに変更を加えます。この VHDL ファイルをエンベデッド システム (system.xmp ファイル) をインスタンスエートする最上位ファイルとして使用します。また、デザインのブロック RAM に読み込む ELF ファイルを指定します。

このチュートリアルでは、Project Navigator でエンベデッド デザインをシミュレーションする方法を示すと共に、エンベデッド システムをインスタンスエートする最上位 VHDL ファイルを作成する方法も示します。この例は、VHDL のみです。

Project Navigator でエンベデッド デザインをシミュレーションするには、まずシミュレーション モデルの生成オプションを設定します。

1. 第 7 章「独自の IP の作成」で作成した ISE プロジェクトを開きます。
2. XMP ソースをダブルクリックして XPS を開きます。
3. [Project] → [Project Options] をクリックします。
4. [HDL and Simulation] タブをクリックします。
5. 次のオプションを設定します。
 - ◆ [HDL] : [VHDL]
 - ◆ [Simulation Test Bench] : [Generate test bench template] をオフ
 - ◆ [Simulation Models] : [Behavioral]

次に、シミュレーションする実行ファイルを選択します。

6. [Applications] タブで [Project: TestApp_Peripheral_microblaze_0] を右クリックし、[Mark to Initialize BRAMs] をクリックします。
7. [Project: TestApp_Peripheral_microblaze_0] を右クリックし、[Build Project] をクリックします。
8. XPS で [Simulation] → [Compile Simulation Libraries] をクリックし、ウィザードの指示に従ってシミュレーション ライブラリをコンパイルまたはシミュレーション ライブラリの場所を指定します。これには数分かかります。

9. [Simulation] → [Generate Simulation HDL Files] をクリックします。

これで、ブロック RAM から実行される C コードも含め、エンベデッド システム全体を表す VHDL ファイルが生成されました。

シミュレーション モデルの生成オプションを設定したので、最上位 VHDL ファイルを作成し、プロセス サブシステムをインスタンス化します。

1. ISE で、<project name>/system/hdl ディレクトリにある system_stub.vhd ソース ファイルを開きます。このファイルは、第 7 章でエンベデッド システムのネットリストを作成したときに自動的に生成されたものです。
2. [File] → [Save As] をクリックし、ファイルを system_top.vhd という名前で保存します。ファイルはツールを実行すると上書きされるので、異なる名前で保存することは重要です。
3. [Project] → [Add Source] をクリックします。
4. 先ほど作成したプロジェクトの system\hdl サブディレクトリにある system_top.vhd ファイルを選択します。[Adding Source Files] ダイアログ ボックスで [Association] を [All] に設定します。
5. [Design] パネルの [Sources for] ドロップダウン リストで [Behavioral Simulation] を選択します。
6. system_top.vhd ファイルの最後 (#end architecture STRUCTURE 文の後) に次のコードを追加します。

```
-- synthesis translate_off
configuration system_conf_top of system_stub is
  for STRUCTURE
    for system_i : system
      use configuration work.system_conf;
    end for;
  end for;
end system_conf_top;
-- synthesis translate_on
```

このコードは、データで初期化する必要のあるブロック RAM を含む任意の VHDL デザインに追加できます。

7. ファイルを保存します。

次に、シミュレーションを実行するよう Project Navigator を設定します。

1. ISE プロジェクトの最上位ディレクトリに pn.do ファイル (このマニュアルの添付ファイル) をコピーします。
メモ : 添付ファイルは、Adobe Acrobat Reader の左下にあるクリップ アイコンをクリックすると表示されます。
2. [Design] パネルの [Processes] ペインで [ModelSim Simulator] → [Simulate Behavioral Model] を右クリックし、[Process Properties] をクリックします。

3. 次の手順に従って、コピーした pn.do ファイルを指定します。
 - ◆ [Use Automatic Do File] をオフにします。
 - ◆ [Use Custom Do File] をオンにし、[Custom Do File] で pn.do を指定します。

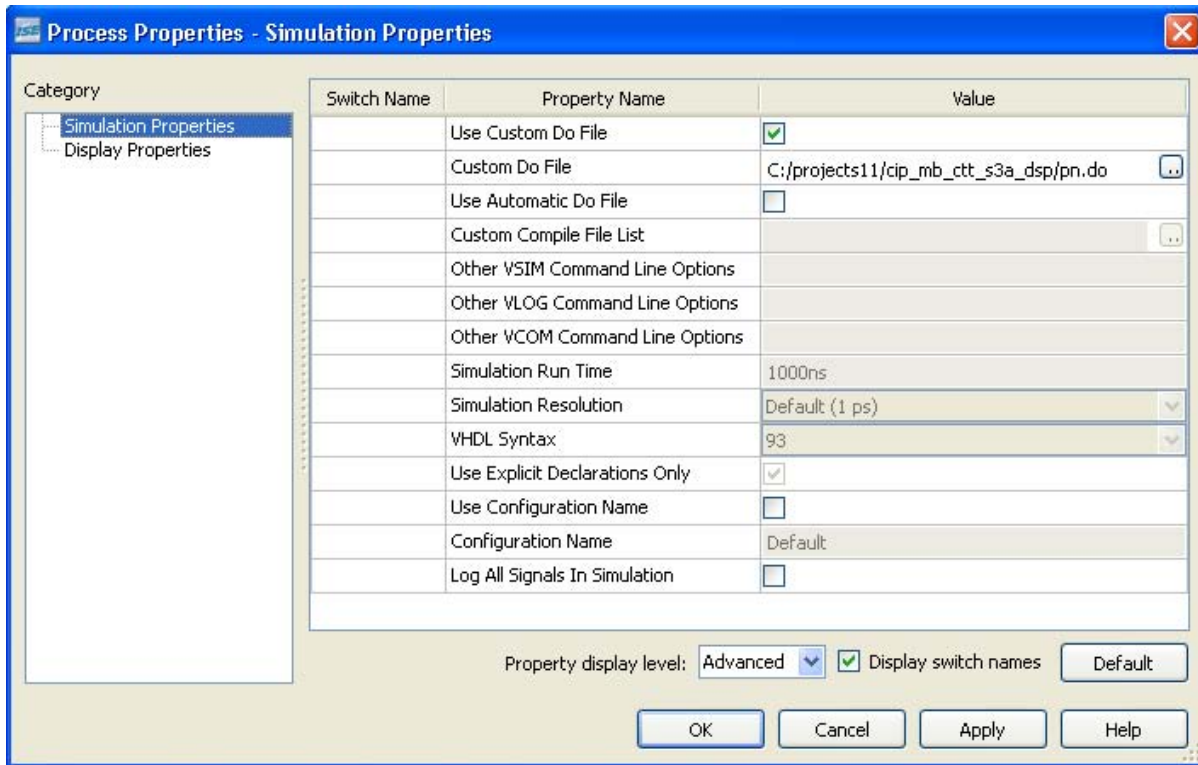


図 A-1 : [Process Properties] ダイアログ ボックスの [Simulation Properties] ページ

4. [OK] をクリックします。
5. [Simulate Behavioral Model] をダブルクリックして ModelSim を実行します。

出力の観察

シミュレーションが読み込まれ、実行されると、ModelSim の [Wave] ウィンドウにエンベデッド デザインの最上位信号が表示されます。

MicroBlaze™ プロセッサのコードは、ブロック RAM のロケーション 0x00000000 から開始します。これを表示するには、[Wave] ウィンドウで signal/system/ilmb_lmb_abus (MicroBlaze ローカル メモリ アドレス バス) の 1400000ps 付近にスクロールします。アドレス 0、4、8... が表示されます。これらのアドレスでフェッチされる命令は /system/ilmb_lmb_readdbus ファイルにあります。

実行された処理

XPS で構築および管理し、SDK で実行した TestApp_Memory_Microblaze ソフトウェア プロジェクトを ModelSim でシミュレーションしました。シミュレーション モデル生成ツールである Simgen により、XPS の [Applications] タブで [Mark to Initialize BRAMs] をオンにしたプログラムが認識され、シミュレーション モデルが生成されます。

シミュレーションされた ELF ファイル (executable.elf) は、プロジェクト ディレクトリの system\TestApp_Memory_Microblaze_0 サブディレクトリにあります。

SDK で開発したアプリケーションをシミュレーションする場合は、ELF ファイルを \system\TestApp_Memory_Microblaze_0 にコピーし、シミュレーション HDL ファイルを再生成してシミュレーションを再実行できます。

ここに示した方法は、オンチップ ブロック RAM から実行されるソフトウェア用です。外部 DDR2 メモリから実行されるコードのシミュレーションについては、このマニュアルでは説明していません。外部 DDR2 メモリからのシミュレーション方法は、http://japan.xilinx.com/support/documentation/sw_manuals/xilinx11/manuals.pdf から『合成/シミュレーション デザイン ガイド』を参照してください。

エンベデッド システムのシミュレーション環境を構築するには、必要な手順が多数あります。シミュレーション環境を一度設定すると、エンベデッド システムと FPGA のそれ以外の部分を両方シミュレーションするために使用できます。

IP バス ファンクション モデル シミュレーション

この付録では、第 7 章「独自の IP の作成」の `pwm_lights` デザインおよびバス ファンクション モデル (BFM) シミュレーション プラットフォームを使用します。第 7 章のチュートリアルを実行していない場合は、戻ってデザインを完了させる必要があります。

また、EDK シミュレーション ライブラリがコンパイルされていることが必要です。

BFM についてと使用する理由

バス ファンクション モデルは、バス トランザクション (この場合は PLBv46) の動作をモデリングするために使用するシミュレーション モデルです。BFM は、通常カスタム IP の動作をモデリングする目的でのみ使用されます。また、BFM シミュレーションは合成後のシミュレーションまたはタイミング シミュレーションよりも高速なので、複雑なトランザクションのシミュレーションを高速化するためにも使用されます。

PLBv46 の仕様は長くて複雑です。カスタム IP をバスに接続して、IP がバス仕様を満たすかどうかを検証する必要がある場合があります。テストベンチを作成して適切なスティミュラスを作成し、正しく動作するかどうかを確認するのは非常に困難であり、テストベンチが正しく記述されているかどうかを確認する手段也没有ありません。

これを支援するため、既知の適切なスティミュラス モデルとして使用可能なバス ファンクション モデル (BFM) が作成されました。PLBv46 マスタおよびスレーブ デバイスのモデルだけでなく、トランザクションをキャプチャし、正しく動作しているかどうかを検証するために使用可能なモニタ モジュールも含まれています。

BFM コンパイラ (BFC) も含まれており、特定のデザイン言語を使用して、一連の読み出しおよび書き込みバス トランザクションと予測される値を記述できます。

概念的には簡単ですが、バス ファンクション シミュレーションを実行するシミュレーション環境を手動で設定するのは困難です。CIP ウィザードでは、必要な BFM シミュレーション モデルをテスト中の IP に自動的に接続できます。

メモ： `bfm_system` プロジェクトを起動したときに Version Management ウィザードが開いた場合は、必要なコアをアップデートしてください。

チュートリアル：BFM の実行

チュートリアルを開始する前に、開いている XPS プロジェクトを閉じます。BFM を作成するよう指定した場合、pcores\pwm_lights_v1_00_a\dev1 ディレクトリに bfmsim というサブディレクトリが作成され、bfm_system.xmp という XPS BFM シミュレーションプロジェクトが保存されています。

1. XPS で bfm_system.xmp プロジェクト ファイルを開きます。[Bus Interfaces] タブの表示は、次のようになります。

Bus Interfaces		Ports	Addresses		
Name	Bus Name	IP Type	IP Version	IP Classification	
plb_bus		★ plb_v46	1.04.a	PLBV46 Bus	
[-] bfm_processor	MPLB	★ plbv46_mast...	1.00.a	Peripheral	
[-] bfm_monitor	MON_PLB	★ plbv46_moni...	1.00.a	Peripheral	
[-] bfm_memory	SPLB	★ plbv46_slav...	1.00.a	Peripheral	
[-] my_core	SPLB	★ pwm_lights_tb	1.00.a	Peripheral	
synch_bus		★ bfm_synch	1.00.a	IP	

図 B-1：XPS BFM ユーザー pcore シミュレーション プロジェクト

2. [Project] → [Project Options] → [HDL and Simulation] タブをクリックします。
3. シミュレーションする HDL 言語を選択します。ここでは、デフォルトの [VHDL] をオンにします。
4. BFM ではビヘイビア シミュレーションのみが可能なので、[Simulation Models] はデフォルトの [Behavioral] をオンのままにします。
5. シミュレーション オプションの設定が終了したら、[OK] をクリックします。

6. [Simulation] → [Generate Simulation HDL Files] をクリックし、Simgen (Simulation Model Generator) を実行します。

Simgen を実行すると、bfmsim ディレクトリに simulation ディレクトリが作成されます。このディレクトリには、HDL ラップ ファイルとビヘイビア シミュレーションの実行に必要な DO スクリプト ファイルが含まれています。

7. ツールバーの [Custom Button 1] をクリックします。CIP ウィザードにより、BFM シミュレーションプロジェクトが作成されたときにこのツールバー ボタンが設定されています。

[Custom Button 1] をクリックすると、次の処理が実行されます。

- ◆ Bash シェルが起動し、makefile が実行されます。
- ◆ 設定したシミュレーション オプションを使用して、sample.bfl ファイルに対して CoreConnect™ ツールキット バス ファンクション コンパイラ (BFC) が実行されます (詳細は <project name>\pcores\pwm_lights_v1_00_a\dev1\bfmsim\scripts\sample.bfl を参照)。
- ◆ BFC 出力コマンド ファイル (シミュレータによって INCLUDE または DO ファイル) を使用してシミュレータが実行され、sample.bfl ファイルのコマンドが実行されます。

次に示すようなシミュレータ波形が表示されます。

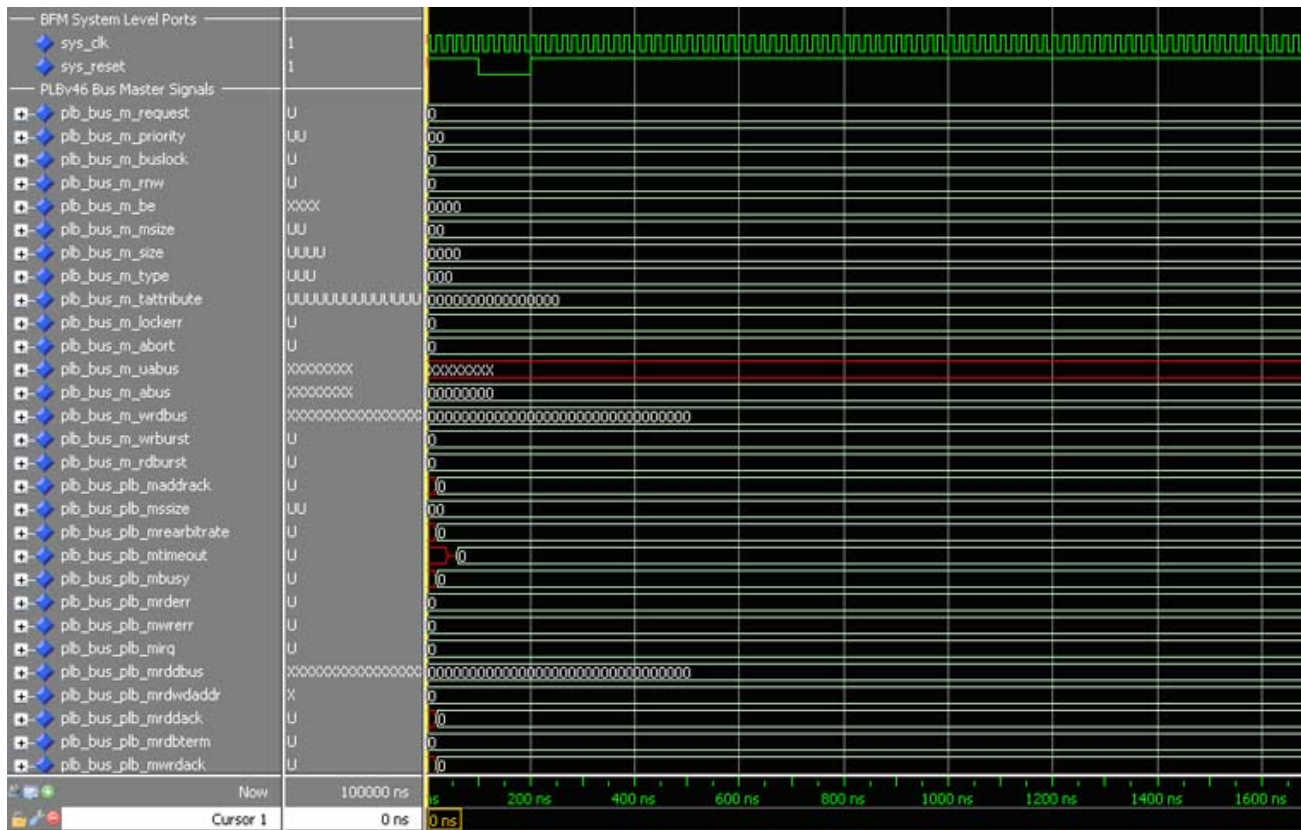


図 B-2 : sample.bfm の BFM 波形シミュレーション結果

実行された処理

CIP ウィザードは、シミュレーションを実行する前に次の処理を実行します。

CIP ウィザードでテストプロジェクトを作成

- HDL テンプレート ファイルを生成します。このテンプレートを変更すると、機能する pcore を作成できます。
- テスト プロジェクトを生成します。pcore が独立したものになるので、システムに統合する前にバスとの動作を検証できます。

このプロジェクトは、<project_name>\pcores\pwm_lights_v1_00_a\dev1\bfmsim ディレクトリに作成されます。

このテスト プロジェクトでは、CoreConnect ツールキットで提供される複数の BFM が使用されます。ここでは、プロセッサ、バス、メモリ、およびバス モニタのモデルがあり、すべてコアに接続されます。

XPS ツールの利点

XPS を使用すると、これらのモデルを手動で作成する必要がなく、すべて自動的に正しく接続されます。

シミュレーション プラットフォームを生成した後、[Custom Button 1] を使用することにより、シミュレーション プロセスのいくつかの面倒な手順が自動化されます。CoreConnect バス ファンクション コンパイラを使用して sample.bfm が実行され、シミュレータで使用するコマンド ファイルが生成されます。

カスタム ボタンの詳細は、[Project] → [Customize Buttons] をクリックし、F1 キーを押してください。使用する makefile の場所は、次のチュートリアルで示します。

[Custom Button 1] により実行される makefile は、BFL のコンパイルに加え、コマンド ファイルを使用してシミュレータを起動し、シミュレーションを開始します。シミュレーションの開始とコンパイルプロセスを 1 つのボタンのクリックで実行できます。

チュートリアル: バス トランザクションを実行するスクリプトの記述

BFM スクリプトの構文は直感的に理解できるものではありませんが、完成したスクリプトを参照することにより、簡単に機能を拡張できます。

1. XPS で [File] → [Open] をクリックし、`<project name>\pcores\wm_lights_v1_00_a\dev1\bfmsim\scripts` ディレクトリに移動します。
2. `sample.bfl` ファイルを開きます。

このファイルをこのマニュアルに添付されている `sample.bfl` ファイルで置換することもできます。その場合は、手順 5 に進んでください。

コードの最初の約 160 行では、コマンド ラインを解読しやすくするためのコマンド エイリアスが設定されています。これらのコマンドはソースおよびデスティネーションメモリに自動的に値を設定し、さまざまなコア機能をテストします。コアの要件に応じてコマンドを追加または削除したり、まったく新しい BFL コマンド ファイルを作成できます。

BFL コマンドの情報

メモ: 新しい BFL ファイルを作成する場合は、`bfmsim` ディレクトリにある `bfm_sim_xps.make` ファイルを変更する必要があります。BFL コマンドの詳細は、`$XILINX_EDK\third_party\doc` の `PLBToolkit.pdf` ファイルを参照してください。

次に、BFM コードを変更して `pwm_lights` の実際のシミュレーションを実行し、バス ファンクション シミュレーションの高度な機能を見えます。

完成した `sample.bfl` ファイルは、このマニュアルに添付されています。この添付ファイルは、このチュートリアルで変更したファイルと比較するのに使用するか、またはここで作成する `sample.bfl` と置換できます。このチュートリアルでのコピー作業を簡単にするため、`sample.bfl` ファイルに追加するコマンドが、このマニュアルに添付されている `bus_transaction_bfl_code.txt` というテキスト ファイルに含まれています。

メモ: 添付ファイルは、Adobe Acrobat Reader の左下にあるクリップ アイコンをクリックすると表示されます。

`sample.bfl` にコマンドを追加

3. `sample.bfl` テンプレートで、175 行目付近にある「configure」で開始する行を次のように変更します。
`configure(msize = 01)`

4. ファイルの最後の方にある「Start Testing」セクションの後に、
bus_transaction_bfl_code.txt から次のコードを追加します。これらのコマンドで、バス トランザクションが生成されます。

```
--
-- Define several bus transactions for pwm_lights
-- Memory updates are 64 bits write, bus transactions are 32 bits wide.
--
--
-- Write value of 22 hex to LED register
--
mem_update(addr=30000010,data=22222222_22222222)
write (addr=30000010,size=0000,be=11110000)
-- Read status register, expect to get F0F02207
read (addr=30000000,size=0000,be=11110000)
-- Write to offset 0, then read status, expect to get F0F02208
mem_update(addr=30000000,data=00000000_00000000)
write (addr=30000000,size=0000,be=11110000)
read (addr=30000000,size=0000,be=11110000)
-- Write to offset 4, then read status, expect to get F0F02200
--mem_update(addr=30000000,data=00000000_00000000)
write (addr=30000004,size=0000,be=00001111)
read (addr=30000000,size=0000,be=00001111)
-- Write to offset 8, then read status, expect to get F0F02208
mem_update(addr=30000008,data=00000000_00000000)
write (addr=30000008,size=0000,be=11110000)
read (addr=30000000,size=0000,be=11110000)
-- Write to offset C, then read status, expect to get F0F02200
--mem_update(addr=30000000,data=00000000_00000000)
write (addr=3000000C,size=0000,be=00001111)
read (addr=30000000,size=0000,be=00001111)
001111)
```

このコードで実行される処理は、次のとおりです。

- ◆ mem_update は、書き込むアドレスに書き込みデータ値を設定します。
 - ◆ write コマンドは PLB 書き込みを開始し、read コマンドは PLB 読み出しを開始します。
 - ◆ size=0000 は、シングル トランザクションであることを示します。
 - ◆ be (バイト イネーブル) 設定は、64 ビット バスに対応します。
 - ◆ 0、8、などに揃えられているアドレスは、バイト イネーブルを 11110000 に設定します。
 - ◆ 4、c、などに揃えられているアドレスは、バイト イネーブルを 00001111 に設定します。
5. マニュアルに添付されている sample.bfl ファイルを開きます。
 6. ここで作成した sample.bfl ファイルが添付ファイルと同じであるかどうかを確認し、ファイルを保存します。
 7. [Custom Button 1] をクリックします。
 8. ModelSim の波形ウィンドウで、信号リストの最後にスクロールします。最後にリストされている信号は user_logic の信号で、pwm_logic pcore のカスタム信号です。

実行された処理

BFM スクリプトは、**pcore** に対して書き込みと読み出しを交互に 5 回実行します。BFM スクリプトを参照し、書き込みが実行される順序を確認してください。

- ◆ **ip2bus_rdash** が **High** の場合、**ip2bus_data** 信号には **pwm_lights** からリードバックされた値が含まれます。
- ◆ シミュレーションでは、コアへの書き込みの順序は **F0F02207**、**F0F02208**、**FF002200**、**FF002204**、**FF002205** と示されます。

この情報をシミュレーションで確認してください。最初の値のリードバックは、**450ns** で発生します。これは、これらの特定の値がリードバックされる理由を理解するのに役立ちます。

CIP ウィザードでは、**BFL** ファイルに加え、**bfmsim** プロジェクト ディレクトリの下に対応する **pcores** ディレクトリも作成されます。このディレクトリには、**BFM** テストベンチのテンプレートが含まれます。

コア ロジックの要件に応じて、テンプレート テストベンチに追加できます。

バス ファンクション シミュレーションの高度な機能と、**XPS** で自動的に実行される処理を活用する方法を学びました。カスタム IP を作成する際に **BFM** シミュレーションを実行すると、テスト時間を短縮し、IP が予測どおりに機能することを確実にできます。

用語集

EDK で使用される用語

B

BBD ファイル

ブラック ボックス定義ファイル。ペリフェラルで使用されるネットリスト ファイルをリストします。

BFL

Bus Functional Language (バス ファンクション言語) の略。

BFM

Bus Functional Model (バス ファンクション モデル) の略。

BIT ファイル

ビットストリーム ファイル。

BitInit

Bitstream Initializer の略。FPGA 上のプロセッサの命令メモリを初期化し、ブロック RAM の命令メモリに格納するツールです。

BMM ファイル

ブロック RAM のマップ ファイル。各ブロック RAM がどのように連続した論理データ空間を構成するかが記述されています。Data2MEM は、このファイルを使用してデータを最適な初期化形式に変換します。BMM ファイルはテキスト ファイルなので、直接編集可能です。

BSB

Base System Builder の略。EDK デザインを作成するためのウィザードです。Base System Builder で使用されるファイル タイプも BSB です。

BSP

[「Standalone BSP」](#) を参照してください。

C

CFI

Common Flash Interface (共通フラッシュ インターフェイス) の略。

D

DCM

Digital Clock Manager (デジタル クロック マネージャ) の略。

DCR

デバイス コントロール レジスタ。

DLMB

データ側のローカル メモリ バス。「LMB」も参照してください。

DMA

Direct Memory Access の略。

DOPB

データ側のオンチップ ペリフェラル バス。「OPB」も参照してください。

DRC

デザイン ルール チェック。

DSPLB

データ側のプロセッサ ローカル バス。「ISPLB」も参照してください。

E

EDIF ファイル

Electronic Data Interchange Format の略。業界標準のネットリスト ファイル形式です。

EDK

ザイリンクス エンベデッド開発キット

ELF ファイル

Executable and Linked Format の略。CPU 実行コードのイメージを含む、CPU 上で実行可能なバイナリ データ ファイル。コンパイラ/リンカによって作成されます。ELF ファイルは基本的なデータ入力形式であり、Data2MEM で使用されます。Data2MEM には、ELF ファイルの内容を確認する機能も備わっています。

EMC

External Memory Controller の略。外部メモリ コントローラ。

EST

エンベデッド システム ツール。

F

FATfs (XilFATfs)

LibXil FATFile システム。XilFATfs ファイル システムのアクセス ライブラリを使用すると、ザイリンクス System ACE のコンパクト フラッシュまたは IBM のマイクロドライブ デバイスに保存されたファイルに対して読み出し/書き込みを実行できます。

Flat View

[IP Catalog] タブおよび [System Assembly View] で、項目をグループ化せずにすべてフラットに表示するビューです。

FPGA

Field Programmable Gate Array の略。ザイリンクスが開発した集積回路で、IC を製造、出荷した後、エンド ユーザーがザイリンクスの開発システム ソフトウェアを使用し、ロジック ファンクションを定義します。ザイリンクスでは、RAM ベースの FPGA デバイスを提供しています。

FSL

MicroBlaze™ の高速シンプレックス リンク。単一方向のポイント トゥ ポイント (PPP) データ ストリーミング インターフェイスで、ハードウェア間でのデータ送信を高速にします。MicroBlaze プロセッサには、FSL インターフェイスが直接接続されています。

G

GDB

GNU デバッガ。

GPIO

汎用入出力。オンチップ ペリフェラル バスに接続される 32 ビットのペリフェラルです。

H

HDL 言語

ハードウェア記述言語。

Hierarchical View

[IP Catalog] タブおよび [System Assembly View] タブの両方でのデフォルトの表示方法で、IP インスタンスごとにグループ化されています。IP インスタンスは、分類条件 (上からプロセッサ、バス、バスブリッジ、ペリフェラル、汎用 IP) に基づいてリストされています。同じ種類の IP インスタンスは、アルファベット順に表示されます。IP 別に表示すると、IP インスタンスに関連するデータすべてを簡単に確認できます。この表示は、ハードウェアプラットフォームに IP インスタンスを追加する場合に特に便利です。

I

IBA

Integrated Bus Analyzer の略。

IDE

Integrated Design Environment の略。

ILA

Integrated Logic Analyzer の略。

ILMB

命令側のローカル メモリ バス。「LMB」も参照してください。

IOPB

命令側のオンチップ ペリフェラル バス。「OPB」も参照してください。

IPIC

IP インターコネクト。

IPIF

IP インターフェイス。

ISA

命令セット アーキテクチャ。プロセッサの命令セット、レジスタ、割り込み、例外、アドレスなどをプログラマに対してどのように表示するかが記述されています。

ISC

割り込みソース コントローラ。

ISE ファイル

ザイリンクス ISE® (Integrated Software Environment) の Project Navigator プロジェクト ファイル。

ISOCM

命令側のオンチップ メモリ。

ISPLB

命令側のプロセッサ ローカル バス。「[DSPLB](#)」も参照してください。

ISS

命令セット シミュレータ。

J

JTAG

Joint Test Action Group の略。

L

Libgen

XPS (Xilinx® Platform Studio™) に含まれる Library Generator の略。

LibXil 標準 C ライブラリ

標準 C ライブラリ関数、およびペリフェラルにアクセスするための関数を含む EDK ライブラリとデバイスドライバ。EDK ライブラリは、MSS ファイルの情報を基に Libgen で自動的にコンフィギュレーションされます。

LMB

ローカル メモリ バス。主にオンチップ ブロック RAM にアクセスするのに使用するレイテンシの低い同期バスです。MicroBlaze プロセッサには、ILMB バスと DLMB バスが含まれます。

M

MDD ファイル

Microprocessor Driver Definition の略。

MDM

Microprocessor Debug Module の略。

MFS ファイル

LibXil Memory File System の略。ファイル ハンドルの形式で、ユーザーがプログラム メモリを管理できます。

MHS ファイル

Microprocessor Hardware Specification の略。バス、ペリフェラル、プロセッサ、接続、アドレス空間などのエンベデッド プロセッサ システムのコンフィギュレーションを定義するファイルです。

MLD ファイル

Microprocessor Library Definition の略。

MOST[®]

Media Oriented Systems Transport の略。オートモーティブ ネットワーク デバイスの開発規格です。

MPD ファイル

Microprocessor Peripheral Definition の略。ペリフェラルで使用可能なポートおよびハードウェア パラメータがすべて含まれます。

MSS ファイル

Microprocessor Software Specification の略。

MVS ファイル

Microprocessor Verification Specification の略。

N

NCF ファイル

ネットリスト制約ファイル。

NGC ファイル

論理デザイン データと制約の情報を含むネットリスト ファイル。EDIF および NCF ファイルからの情報がこのファイルに含まれています。

NGD ファイル

Native Generic Database の略。ザイリンクス プリミティブで記述された論理的デザイン ファイル。

NGO ファイル

ザイリンクス 特定のフォーマットのバイナリ ファイルで、オリジナルのコンポーネントと階層によるデザインの論理記述が含まれます。

NPI

Native Port Interface の略。ネイティブ ポート インターフェイス。

ISE ファイル

ザイリンクス ISE Project Navigator の以前のプロジェクト ファイル。

O

OCM

オンチップ メモリ。

OPB

オンチップ ペリフェラル バス。

P

PACE

Pinout and Area Constraints Editor の略。

PAO ファイル

Peripheral Analyze Order ファイル。合成およびシミュレーションに必要な HDL ファイルの解析順を定義します。

PBD ファイル

Processor Block Diagram ファイル。

Platgen

XPS に含まれる Hardware Platform Generator の略。

PLB

プロセッサ ローカル バス。

PROM

Programmable Read-Only Memory の略で、プログラム可能な読み取り専用のメモリです。

PSF

Platform Specification Format の略。EDK ツールを駆動するデータ ファイルの形式です。

S

SDF ファイル

Standard Data Format ファイル。プログラム間のデータを転送するために固定長フィールドを使用するデータ形式です。

SDK

ソフトウェア開発キット。

SDMA

Soft Direct Memory Access の略。

Simgen

XPS に含まれる Simulation Generator の略。

SPI

Serial Peripheral Interface (シリアル ペリフェラル インターフェイス) の略。

Standalone BSP

スタンドアロン ボード サポート パッケージ。プロセッサに特有の関数にアクセスするソフトウェア モジュールのセットで、アプリケーションがボードまたはプロセッサ機能に直接 (OS レイヤなしで) アクセスする際に使用されます。

SVF ファイル

Serial Vector Format ファイル。スティミュラスで表現されたテスト パターン、予測される応答、IEEE 1149.1 ベースのテスト用のマスク データを記述する ASCII 形式のファイル。さまざまなシミュレーション ソフトウェア、テスト装置で使用できます。

U

UART

Universal Asynchronous Receiver-Transmitter の略。

UCF

ユーザー制約ファイル。

V

VHDL

VHSIC ハードウェア記述言語。

X

XBD ファイル

ザイリンクス ボード定義ファイル。

XCL

Xilinx CacheLink の略。MicroBlaze プロセッサで使用可能な高パフォーマンスの外部メモリ キャッシュ インターフェイスです。

Xilkernel

EDK に含まれるザイリンクス エンベデッド カーネル。ザイリンクス エンベデッド ソフトウェア プラットフォーム用のコンフィギュレーション可能なモジュール型の小型 RTOS です。

XMD

Xilinx Microprocessor Debugger の略。

XMP ファイル

Xilinx Microprocessor Project の略。EDK デザインの最上位のプロジェクトファイルです。

XPS

Xilinx Platform Studio の略。エンベデッド デザインを開発する GUI 環境です。

XST

Xilinx Synthesis Technology の略。

Z

ZBT

Zero Bus Turnaround™ の略。

さ

ソフトウェア プラットフォーム

ソフトウェア プラットフォームは、アプリケーションを構築するソフトウェアドライバおよび OS をまとめたものです。ハードウェアプラットフォームの流動性およびザイリンクスおよびサードパーティ パートナーのサポートにより、各ハードウェア プラットフォームに対して複数のソフトウェア プラットフォームを作成できます。

は

ブロック RAM (BRAM)

デバイスに組み込まれた LUT ベースのランダム アクセス メモリ (RAM) のブロック。分散 RAM とは区別されます。

ハードウェア プラットフォーム

ザイリンクスの FPGA 技術を使用すると、プロセッサ サブシステムのハードウェア ロジックをカスタマイズ可能です。標準的なマイクロプロセッサチップまたはコントローラチップでは、このようなカスタマイズは不可能です。ハードウェア プラットフォームは、ザイリンクスの技術を使用して作成するエンベデッド プロセッサ サブシステムです。