Zynq-7000 EPP コンセプト、ツール、テクニック ガイド

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

UG873 (v14.2) 2012 年 7 月 27 日

XILINX®

本資料は英語版 (v14.2) を翻訳したもので、内容に相違が生じる場合には原文を優先します。
資料によっては英語版の更新に対応していないものがあります。
日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

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

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	改訂内容
2011 年 12 月 21 日	ベータ 1 アップデート 2	Zynq™-7000 EPP ベータ プログラム向け初版リリース
2012 年 3 月 7 日	ベータ 2 アップデート	ハードウェア デザイン フロー、ソフトウェア アプリケーション開発フロー、SDK を使用したデバッグ、ChipScope™ を使用したデバッグに関連する情報を追加。
2012 年 4 月 24 日	14.1	Zynq™-7000 EPP デバイスをサポートする 14.1 リリース
2012 年 5 月 28 日	14.1	14.1 アップデート バージョン。主なアップデーは次のとおり。 • タイトルをアップデート • ボードを Zynq-7000 リビジョン C にアップデート • 第 5 章「SDK を使用した Linux のブートおよびアプリケーションのデバッグ」を追加 • 付録 A 「アプリケーション ソフトウェア」を追加 • このガイドと併用するデザイン ファイルを含む `ug873_design_files.zip` を追加
2012 年 5 月 31 日	14.1	21 ページ および 47 ページ の LED の表記を D23 から DS23 に修正。

日付	バージョン	改訂内容
2012 年 7 月 25 日	14.2	ISE Design Suite 14.2 向けアップデート。主なアップデートは次のとおり。 • 図のアップデート • 14.2 ソフトウェア向けにデザイン ファイルをアップデート • アプリケーション ソフトウェアに関する情報を 第 3 章 の終わりに移動 • 第 6 章「プロセッシング システムの高性能スレーブ ポートを使用したシステム デザイン」を追加 • 第 7 章「SDK を使用したソフトウェア プロファイリング」を追加 • 第 8 章「アクセラレータ コヒーレンシ ポート (ACP)」を追加
2012 年 7 月 27 日	14.2	Zynq シリコン 1.0 の情報を削除。

XILINX

第 1 章 : 概要

第 2 章 : Zynq プロセッシング システムを使用するエンベデッド システム デザイン

第 3 章 : Zynq プロセッシング システムおよびプログラマブル ロジックを使用するエンベデッド システム デザイン

第 4 章 : SDK および ChipScope を使用したデバッグ

第 5 章 : SDK を使用した Linux のブートおよびアプリケーションのデバッグ

第 6 章 : プロセッシング システムの高性能スレーブ ポートを使用したシステム デザイン

XILINX

第 1 章

概要

1.1 このガイドについて

このガイドは、Zynq™-7000 EPP を使用する場合のザイリンクスの ISE® Design Suite フローについて説明します。ここに挙げるサンプル プロジェクトは、ザイリンクスの ZC702 Rev C 評価ボードおよびツール バージョン 14.2 をターゲットとしています。

注記 : テスト ドライブは、64 ビットの Windows 7 オペレーティング システムで作成されたものです。ほかのバージョンの Windows では結果が異なる場合があります。

このガイドは、次の各章で構成されています。

- 第 1 章では概要を説明します。
- 第 2 章「Zynq プロセッシング システムを使用する エンベデッド システム デザイン」では、Zynq プロセッシング システム (PS) を用いたシステムの作成、および簡単な Hello World アプリケーションの実行について説明します。
- 第 3 章「Zynq プロセッシング システムおよびプログラマブル ロジックを使用するエンベデッド システム デザイン」では、Zynq のプロセッシング システム (PS) とプログラマブル ロジック (PL、つまりファブリック) を使用してシステムを作成する方法に加えて、PS および PL を実行する簡単なアプリケーションの使用方法について説明します。
- 第 4 章「SDK および ChipScope を使用したデバッグ」では、ソフトウェア (SDK デバッグを使用) およびハードウェア (ChipScope™ ソフトウェアを使用) の観点からデバッグ情報を提供します。
- 第 5 章「SDK を使用した Linux のブートおよびアプリケーションのデバッグ」では、Zynq™-7000 EPP ボードにおける Linux OS のブート、およびアプリケーションのデバッグについて情報を提供します。
- 第 6 章「プロセッシング システムの高性能スレーブ ポートを使用したシステム デザイン」では、ファブリックで AXI CDMA IP をインスタンシエートする際の情報を提供し、この IP を PS の高性能 (HP) 64 ビット スレーブ ポートと結合させる方法について説明します。
- 第 7 章「SDK を使用したソフトウェア プロファイリング」では、スタンドアロン BSP のプロファイリング機能、および第 6 章 で作成した AXI CDMA 関連のアプリケーションについて説明します。
- 第 8 章「アクセラレータ コヒーレンシ ポート (ACP)」では、ACP コヒーレントおよび ACP 非コヒーレントの場合の読み出し要求と書き込み要求に関する情報を提供します。
- 付録 A 「その他のリソース」では、このガイドに関連するその他の資料へのリンクを提供します。

1.1.1  テスト ドライブの使用

ソフトウェア ツールの使用方法を習得する最善の方法は、まず使用してみることです。このガイドでは、実際にツールを操作する機会を提供します。テスト ドライブ セクションでは、サンプル プロジェクトの仕様のほかに、背景で何が起こっているかやそれを実行しなければならない理由が示されます。

テスト ドライブには、上記タイトルの横にあるクルマのアイコンが示されています。

1.1.2 その他の資料

その他の資料一覧は、付録 A 「その他のリソース」に記載されています。

1.1.3 トレーニング演習

テスト ドライブの中には関連するタスクをさらに実践するために利用可能なトレーニング演習が提供されているものがあります。演習がある場合、テスト ドライブ セクションの最後にその説明が記載されています。

トレーニング演習へのリンクは、付録 A 「その他のリソース」に記載されています。

1.2 Zynq EPP および EDK でエンベデッド プロセッサ デザインを簡略化

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とソフトウェアに該当する部分はそれぞれがプロジェクトとなります。2 つのデザイン コンポーネントを 1 つのシステムとして機能するように統合することには、さらに課題が伴います。その統合されたものに FPGA デザイン プロジェクトを追加すると、状況は非常に複雑になると考えられます。

Zynq エクステンシブル プロセッシング プラットフォーム (EPP) ソリューションでは、ハード IP として ARM Cortex A9 デュアル コアが提供され、さらに 1 つの SoC 上でこのコアと共にプログラマブル ロジックを使用できるため、複雑性が緩和されます。Zynq EPP はこの種としては業界で初めてのソリューションであり、完全なシステムとして大きな可能性を持っています。

設計プロセスを単純にするために、ザイリンクスはいくつかのツールを提供しています。基本的なツールおよびプロジェクト ファイルの名前、これらツールの略称を把握しておくことを推奨します。EDK 固有の用語は、次のザイリンクス用語集に記載されています。http://japan.xilinx.com/support/documentation/sw_manuals/glossary.pdf

エンベデッド開発キット (EDK) は、Xilinx Platform Studio (XPS) とソフトウェア開発キット (SDK) が組み合わされたものです。ハードウェアとソフトウェアのアプリケーションの設計、デバッグ、実行が可能になるほか、検証や評価を目的としてデザインを実際のボードで動作させるのにも役立ちます。

1.2.1 ISE Design Suite : Embedded Edition

ザイリンクスは、ISE Design Suite と総称される開発システム ツールを複数提供しています。エンベデッド システム開発向けに提供されているのが ISE Design Suite : Embedded Edition です。Embedded Edition は次のツールで構成されています。

- ISE (Integrated Software Environment)
- PlanAhead™ デザイン解析ツール
- FPGA デザインのオンチップ デバッグに有用な ChipScope Pro
- エンベデッド開発キット (EDK)

EDK は DSP デザイン向けの ISE Design Suite : System Edition にも含まれます。

FPGA デザインに ISE ツールを使用する方法の詳細は、付録 A 「その他のリソース」に記載されている資料を参照してください。

1.2.2 エンベデッド開発キット

EDK はザイリンクスの FPGA デバイスに実装する完全なエンベデッド プロセッサ システムを設計する際に使用可能なツール セットおよび IP コアです。

Xilinx Platform Studio

Xilinx Platform Studio (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の設計に使用する開発環境です。XPS はバッチ モードまたは GUI を使用して実行できます。このガイドでは、この実行例が示されています。

ソフトウェア開発キット

ソフトウェア開発キット (SDK) は XPS を補完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 フレームワークで構築されているため、ソフトウェア設計者や設計チームにとって使い慣れた環境です。Eclipse 開発環境の詳細は、http://www.eclipse.org を参照してください。

その他の EDK コンポーネント

EDK のその他のコンポーネントは次のとおりです。

- 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ーおよびライブラリ
- Zynq プロセッシング システムの ARM Cortex-A9MP プロセッサをターゲットにした C/C++ ソフトウェア開発向けの GNU コンパイラおよびデバッガー
- 資料
- サンプル プロジェクト

1.3 ISE ツールでデザイン プロセスを加速

PlanAhead デザイン解析ツールではデザイン ソースをハードウェアに追加できます。このハードウェア システムは XPS を用いて作成可能です。XPS を使用することで、必要な IP を簡単に既存のデザイン ソースへ追加してクロックやリセットなどのポート接続を作成できます。

- XPS は主に、エンベデッド プロセッサ システムの開発に使用します。マイクロプロセッサ、ペリフェラル、コンポーネントの相互接続やそれぞれに対する詳細な設定は XPS で可能です。
- ソフトウェア開発には SDK を使用します。SDK はスタンドアロンのアプリケーションとして提供されています。このツールは、購入後、ロード先のマシンにほかのザイリンクスのツールがインストールされていなくても使用可能です。また、ソフトウェア アプリケーションのデバッグに使用できます。

Zynq プロセッシング システム (PS) は、FPGA プログラマブル ロジック (PL) に何もプログラムされていなくても、ブートして動作させることができます。ただし、ファブリックでソフト IP を使用したり、EMIO を用いて PS ペリフェラルを接続するには、PL をプログラムする必要があります。この手順は SDK から実行できます。

XPS に関するエンベデッド デザイン プロセスの詳細は、『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「デザイン プロセスの概要」を参照してください。この資料へのリンクは、付録 A 「その他のリソース」に記載されています。

注記 : Zynq 開発ツールの初期バージョンでは、プロセッシング システムの直接シミュレーションは利用できません。

1.4 最初に必要なセットアップ

ツールについて詳しく説明する前に、ツールが適切にインストールされ、セットアップした環境がこのガイドのテスト ドライブ セクションの記載要件に一致するかを確認します。

1.4.1 インストール要件 : EDK ツールの実行条件

PlanAhead ツールおよび EDK

PlanAhead デザイン ツールおよび EDK はいずれも ISE Design Suite : Embedded Edition ソフトウェアに含まれています。ソフトウェアと最新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。最新のソフトウェア バージョンであるかどうかは、http://japan.xilinx.com/support にアクセスして確認できます。

ソフトウェアのライセンス

ザイリンクスのソフトウェアには、FLEXnet ライセンスが使用されています。ソフトウェアを初めて起動する際、ライセンスの検証プロセスが実行されます。有効なライセンスが検出されない場合、ライセンス ウィザードに従ってライセンスを取得し、ザイリンクス ツールがそのライセンスを使用できるようにします。ソフトウェアの評価のみが目的であれば、評価ライセンスを取得できます。

ザイリンクス ソフトウェアのライセンスの詳細は、『ザイリンクス デザイン ツール : インストールおよびライセンス ガイド』(UG798) を参照してください。この資料へのリンクは、付録 A 「その他のリソース」に記載されています。

1.4.2 ハードウェア要件

このガイドでは、Zynq ZC702 Rev C 評価ボードをターゲットとしています。このガイドを活用するにあたって、評価ボードに含まれている次のものを用意してください。

- ZC702 評価ボード
- AC 電源アダプター (12 VDC)
- USB Type-A/Mini-B ケーブル (UART 通信用)
- JTAG を介するプログラムおよびデバッグ用のザイリンクス プラットフォーム ケーブルおよび Digilent 社製ケーブル
- Linux ブート用の SD-MMC フラッシュ カード
- ターゲット ボードとホスト マシンの接続用のイーサネット ケーブル

XILINX®

第 2 章

Zynq プロセッシング システムを使用するエンベデッド システム デザイン

ザイリンクスのエンベデッド開発キット (EDK) の概要に引き続き、このツールを活用して Zynq™-7000 EPP プロセッシング システムを使用するエンベデッド システムを開発する方法を説明します。

Zynq EPP は、ARM Cortex A9 ハード IP とプログラマブル ロジックで構成されています。これらは、次の 2 つの方法で使用できます。

1. Zynq PS は、ファブリックからの追加 IP なしでスタンドアロン モードで使用できます。
2. IP は、ファブリックでインスタンシエートして Zynq PS で使用可能です。PS と PL を組み合わせて使用することで、1 つの SOC で複雑かつ高効率なデザインを実現できます。

2.1 エンベデッド システムの構築

Zynq システム デザインを作成するには、ブート デバイスおよびペリフェラルを適切に選択するために PS をコンフィギュレーションします。PS ペリフェラルと利用可能な MIO の接続がデザイン要件を満たしている限り、ビットストリームは必要ありません。この章では、このようなデザインの作成方法を説明します。

2.1.1  テスト ドライブ : Zynq プロセッシング システムの新規エンベデッド プロジェクトを作成する

このテスト ドライブでは、ISE® の PlanAhead™ デザイン解析ツールを起動し、エンベデッド プロセッサ システムのプロジェクトを最上位として作成します。

PlanAhead デザイン ツールでデザインを開始する

1. PlanAhead ツールを起動します。
2. **[Create New Project]** をクリックして New Project ウィザードを開きます。

3. 次の表の情報に基づいて、ウィザード画面で選択操作を行います。

ウィザード画面	システム プロパティ	使用する設定コマンド
Project Name	Project Name	プロジェクト名を指定する。
	Project Location	プロジェクト ファイルを格納するディレクトリを指定する。
	Create Project Subdirectory	チェック ボックスをオンのままにする。
Project Type	デザインのソースのタイプを指定する。RTL または合成済み EDIF から開始できる。	デフォルトの **[RTL Project]** を使用する。
Add Sources	この画面では変更しない。	
Add Existing IP	この画面では変更しない。	
Add Constraints	この画面では変更しない。	
Default Part	Specify	**[Boards]** を選択する。
	Board	**[Zynq-7 ZC702 Evaluation Board]** を選択する。
New Project Summary	プロジェクト サマリ	**[Finish]** をクリックする前にプロジェクト サマリを再確認し、プロジェクトを作成する。

[Finish] をクリックすると New Project ウィザードが閉じ、作成したプロジェクトが PlanAhead デザイン ツールで開きます。

ここからは Add Sources ウィザードを使用して、エンベデッド プロセッサ プロジェクトを作成します。

1. [Project Manager] で **[Add Sources]** をクリックします。

   Add Sources ウィザードが開きます。

2. **[Add or Create Embedded Sources]** をオンにして、**[Next]** をクリックします。
3. [Add or Create Embedded Source] ウィンドウで **[Create Sub-Design]** をクリックします。
4. モジュール名を入力して **[OK]** をクリックします。この例では、「`system`」という名前を入力します。

   作成したモジュールがソース リストに表示されます。

5. **[Finish]** をクリックします。

   PlanAhead デザイン ツールでエンベデッド デザイン ソース プロジェクトが作成されます。このツールはエンベデッド プロセッサ システムの存在を認識し、XPS を起動します。

XPS で設計を継続する

XPS では、次の 2 つの方法で新しいエンベデッド システムを設計できます。

- Base System Builder (BSB) ウィザードを使用する

  BSB ウィザードでは、プロセッシング システムの I/O インターフェイスを選択してコンフィギュレーションし、デフォルトのペリフェラルをファブリックに追加できます。ザイリンクスは、常に BSB ウィザードを使用して新しいエンベデッド デザイン プロジェクトの基礎を作成することを推奨します。

- 空のプロジェクトを作成する

  このオプションでは、プロセッシング システムを手動でデザインへ追加し、I/O インターフェイスをコンフィギュレーションする必要があります。

BSB ウィザードを使用して新しいエンベデッド システムを設計する

1. BSB ウィザードを使用してベース システムを作成するかどうかを確認するダイアログ ボックスが表示されたら、[Yes] をクリックします。

   BSB の最初のウィンドウでは、システムを AXI ベースにするか PLB ベースにするかの選択が求められます。

2. AXI システムを選択して [OK] をクリックします。
3. Base System Builder ウィザードで、次の表に記載されている設定を使用してプロジェクトを作成します。設定またはコマンドが表に記載されていない場合は、デフォルトの値に従います。

ウィザード画面	システム プロパティ	使用する設定コマンド
Board and System Selection	Board	デフォルトのオプションを使用して、Zynq ZC702 評価プラットフォーム リビジョン C 向けのシステムを作成する。 **注記**：ここでは、PlanAhead ツールで選択したボードが自動入力される。
	Board Configuration	ボードの選択に基づいて情報が自動入力される。
	Select a System	Zynq プロセッシング システム 7
Peripheral Configuration	Select and Configure Peripherals	Included Peripherals リストから次を削除する。 • GPIO_SW • LEDs_4Bits

4. デザインを生成するには、**[Finish]** をクリックします。
5. XPS ウィンドウを閉じます。アクティブな PlanAhead ツール セッションが、プロジェクトの設定でアップデートされます。

   ソフトウェア開発用にシステム情報を SDK にエクスポートする方法は、「2.1.2 テスト ドライブ : SDK へエクスポートする」を参照してください。

空のプロジェクトを使用して新しいエンベデッド システムを設計する

BSB ウィザードでデフォルトのエンベデッド システムを作成済みの場合は、このセクションを省略して「2.1.2 テスト ドライブ : SDK へエクスポートする」へ進んでください。

1. BSB ウィザードを使用してベース システムを作成するかどうかを確認するダイアログ ボックスが表示されたら、**[No]** をクリックします。

   ここでは、プロセッサをシステムに手動で追加します。

2. IP カタログで **[Processor]** → **[Processing System]** をクリックし、プロセッサをシステムに追加します。

   processing_system7 4.01.a インスタンスを 1 つデザインに追加するかどうかを確認するダイアログ ボックスが表示されます。

3. **[Yes]** をクリックしてプロセッサ インスタンスを追加します。
4. **[Bus Interfaces]** タブをクリックします。`processing_system7` が追加されていることを確認します。

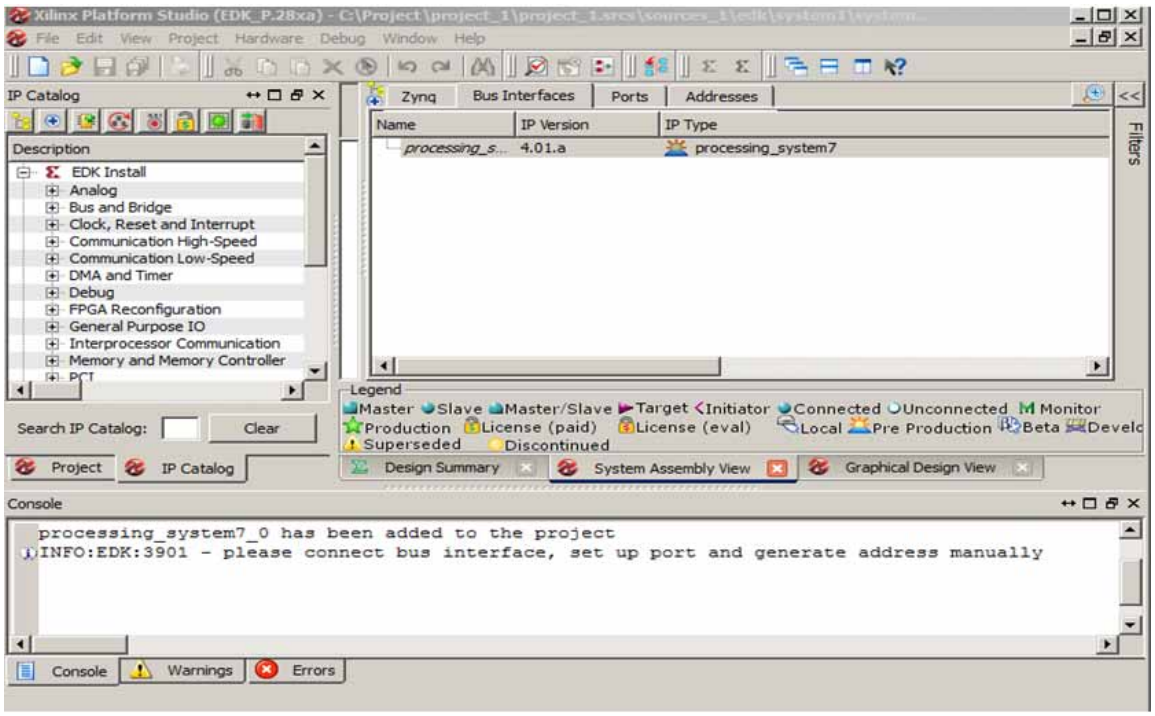

図 2-1 : XPS の [System Assembly] ビュー

5. [System Assembly] ビューで **[Zynq]** タブをクリックし、Zynq プロセッシング システムのブロック図を開きます。

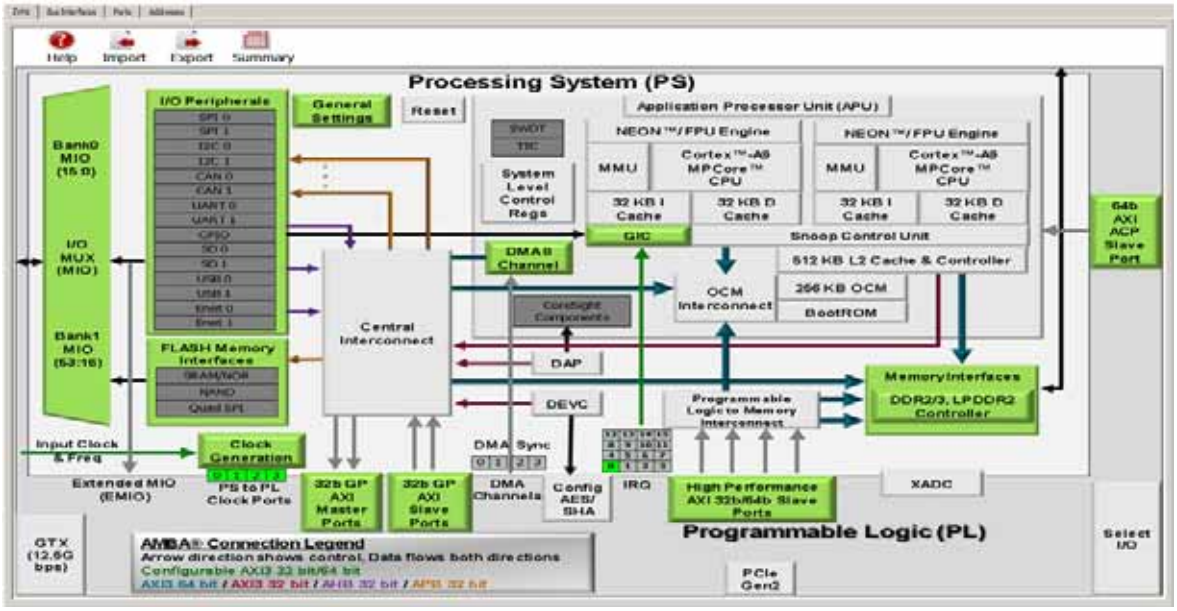

図 2-2 : Zynq **プロセッシング システム**

ブロック図の構成を確認します。Zynq プロセッシング システム ブロック図の緑色のブロックは、コンフィギュレーション可能です。ブロックをクリックすると、そのコンフィギュレーション ウィンドウが開きます。

6. **[Import Zynq Configurations]** をクリックします。

[Import Zynq Configurations] ダイアログ ボックスが開きます。

7. コンフィギュレーション テンプレート ファイルを選択します。デフォルトで選択されるテンプレートは、ローカル マシンの ZC702 ボードに対応するインストール パスにあるものです。

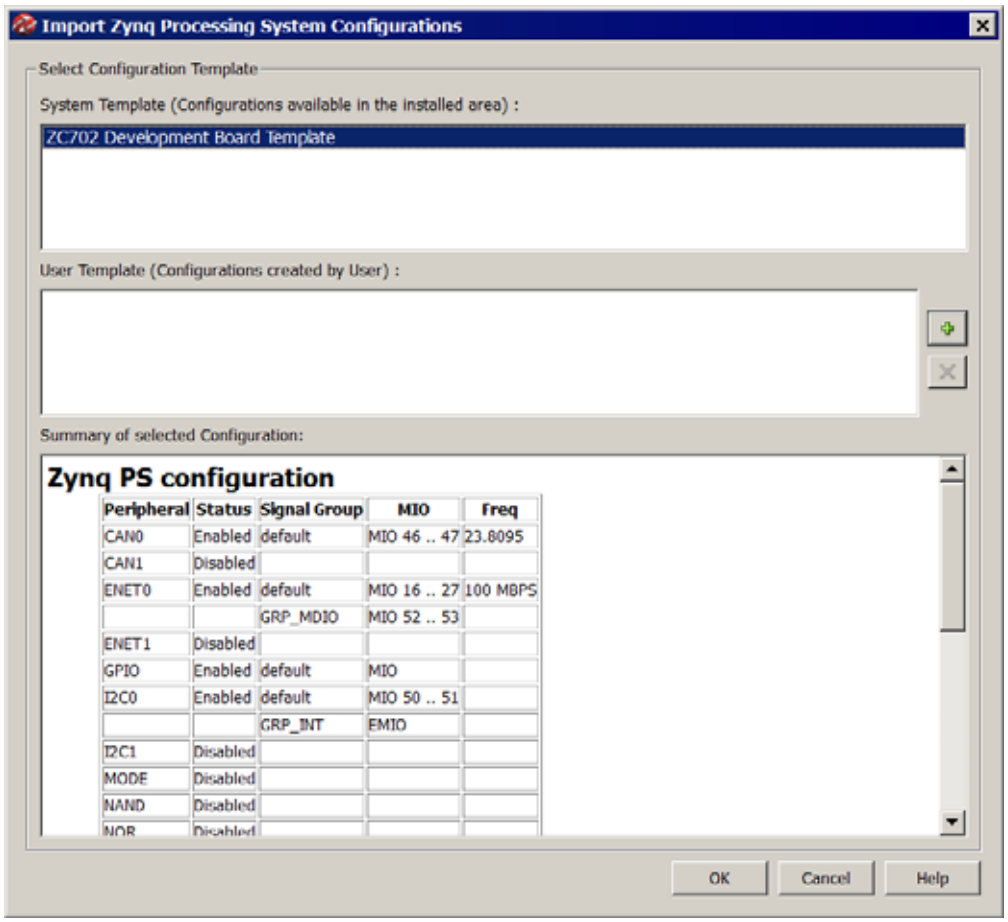

図 2-3 : [Import Zynq Processing System Configurations] **ダイアログ ボックス**

8. **[OK]** をクリックします。
9. Zynq の MIO コンフィギュレーションとデザインがアップデートされることを確認するウィンドウが開いたら、**[Yes]** をクリックします。

10. Zynq のブロック図が変更されていることを確認します。I/O ペリフェラルがアクティブになります。

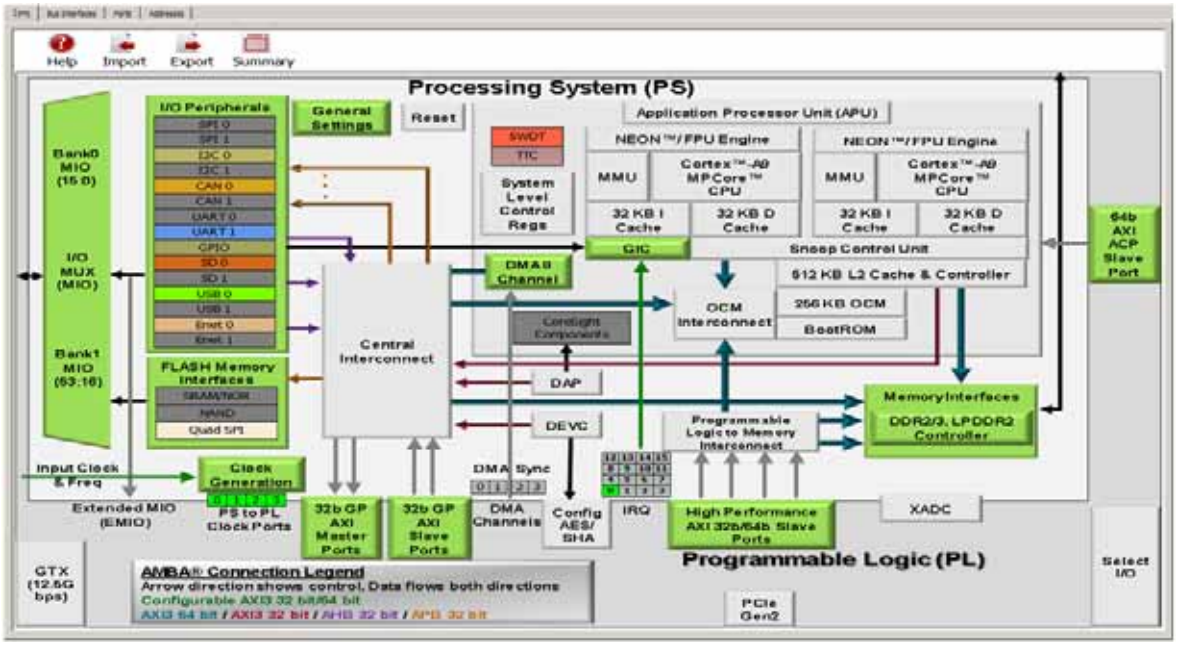

図 2-4 : アップデート後の Zynq のブロック図

11. ブロック図で緑色の **I/O Peripherals** ボックスをクリックします。

    ZC702 のボード レイアウトに従っていくつかの MIO ピンが複数のペリフェラルに割り当てられており、これらのペリフェラルはプロセッシング システムで有効となっています。たとえば、UART1 は有効で、UART0 は無効です。これは、UART1 が UART を経由して USB - UART コネクタを介し、ZC702 ボードの USB コンバーター チップへ接続されているためです。

12. [Zynq PS MIO Configurations] ダイアログ ボックスを閉じます。
13. XPS ウィンドウを閉じます。アクティブな PlanAhead ツール セッションが、プロジェクトの設定でアップデートされます。

    ソフトウェア開発用にシステム情報を SDK にエクスポートする方法は、「2.1.2 テスト ドライブ : SDK へエクスポートする」を参照してください。

2.1.2 テスト ドライブ : SDK へエクスポートする

このテスト ドライブでは、PlanAhead ツールから SDK を起動します。

1. [Sources] ペインの **[Design Sources]** で、**[system(system.xmp)]** を右クリックして **[Create Top HDL]** をクリックします。

   PlanAhead がデザインの最上位モジュールである `system_stub.v` を生成します。

2. PlanAhead ツールで **[File]** → **[Export]** → **[Export Hardware]** をクリックします。

   [Export Hardware] ダイアログ ボックスが開きます。デフォルトでは、[Export Hardware] チェック ボックスがオンになっています。

3. **[Launch SDK]** チェック ボックスをオンにします。
4. **[OK]** をクリックすると、SDK が開きます。

SDK が起動すると、ハードウェア記述ファイルが自動的に読み込まれます。[system.xml] タブは、プロセッシング システム全体のアドレス マップを表示します。

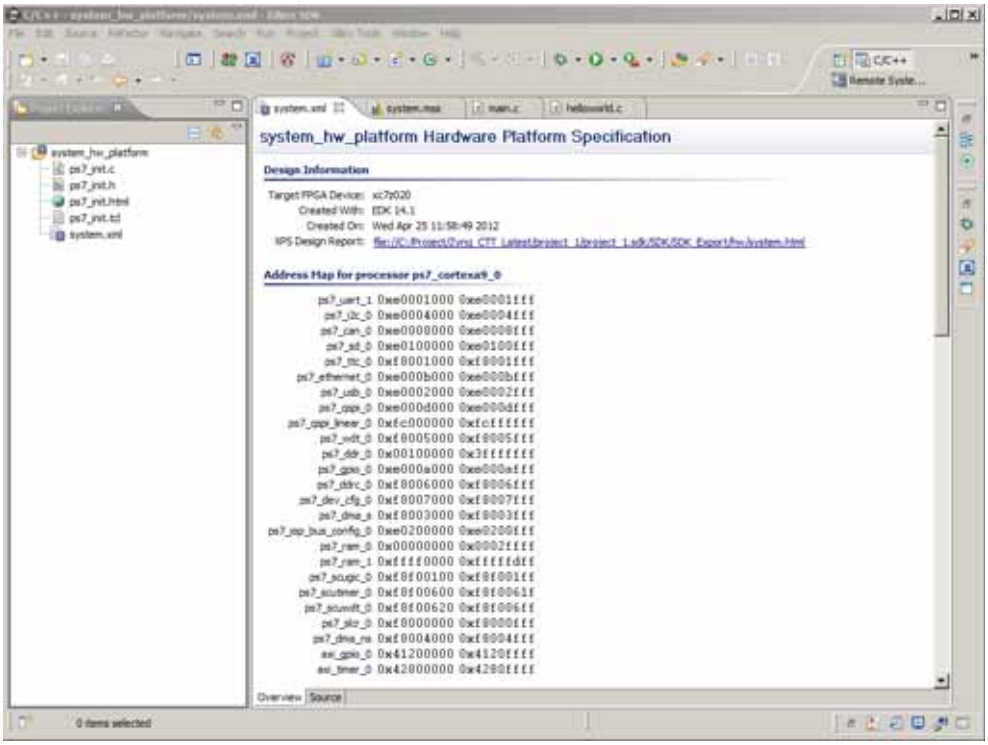

図 2-5 : SDK の [system.xml] タブのアドレス マップ

ここまでの結果

PlanAhead デザイン ツールによって、デザインのハードウェア プラットフォーム仕様 (この例では `system.xml`) が SDK にエクスポートされました。`system.xml` に加えて、さらにファイル (`ps7_init.c`、`ps7_init.h`、`ps7_init.tcl`、`ps7_init.html`) が SDK にエクスポートされています。

SDK が起動すると、system.xml ファイルがデフォルトで開きます。このファイルから読み出されるシステムのアドレス マップがデフォルトで SDK ウィンドウに表示されます。

`ps7_init.c` および `ps7_init.h` ファイルには、Zynq プロセッシング システムの初期化コードのほかに、DDR、クロック、PLL、および MIO の初期化設定が含まれます。SDK は、アプリケーションがプロセッシング システムの最上位で実行可能となるように、これらの設定をプロセッシング システムの初期化時に使用します。プロセッシング システムの設定の中には ZC702 評価ボード向けに固定されているものがあります。

次に実行すること

これで、SDK を使用してプロジェクトのソフトウェア開発を開始できます。次のセクションでは、ハードウェア プラットフォーム向けのソフトウェア アプリケーションの作成に役立つ情報を提供します。

2.1.3  テスト ドライブ : Hello World アプリケーションを実行する

1. 電源ケーブルをボードに接続します。
2. ザイリンクスのプラットフォーム ケーブル USB II または Digilent 社製ケーブルを使用して Windows ホスト マシンとターゲット ボードを次のように接続します。

   ザイリンクス プラットフォーム ケーブル USB II を使用する場合は、スイッチ SW10 を次のように設定します。

   a. ビット 1 の位置を ON 側に設定する
   b. ビット 2 の位置を ON 側の反対方向に設定する

   Digilent 社製ケーブルを使用する場合は、スイッチ SW10 を次のように設定します。

   a. ビット 1 の位置を ON 側の反対方向に設定する
   b. ビット 2 の位置を ON 側に設定する
3. ザイリンクスのプラットフォーム ケーブル USB II を使用して Windows ホスト マシンとターゲット ボードを接続します。
4. USB ケーブルを Windows ホスト マシンと接続しているターゲット ボードのコネクタ J17 につなぎます。シリアル転送に USB を使用する際、このように接続します。
5. 図 2-6 に示すスイッチを使用して ZC702 ボードに電源を投入します。

重要 : ジャンパー J27 および J28 は SD カード スロットから離れた側に付け、同じラインにあるほかのジャンパーは SD カード スロットの方を向くようにします。

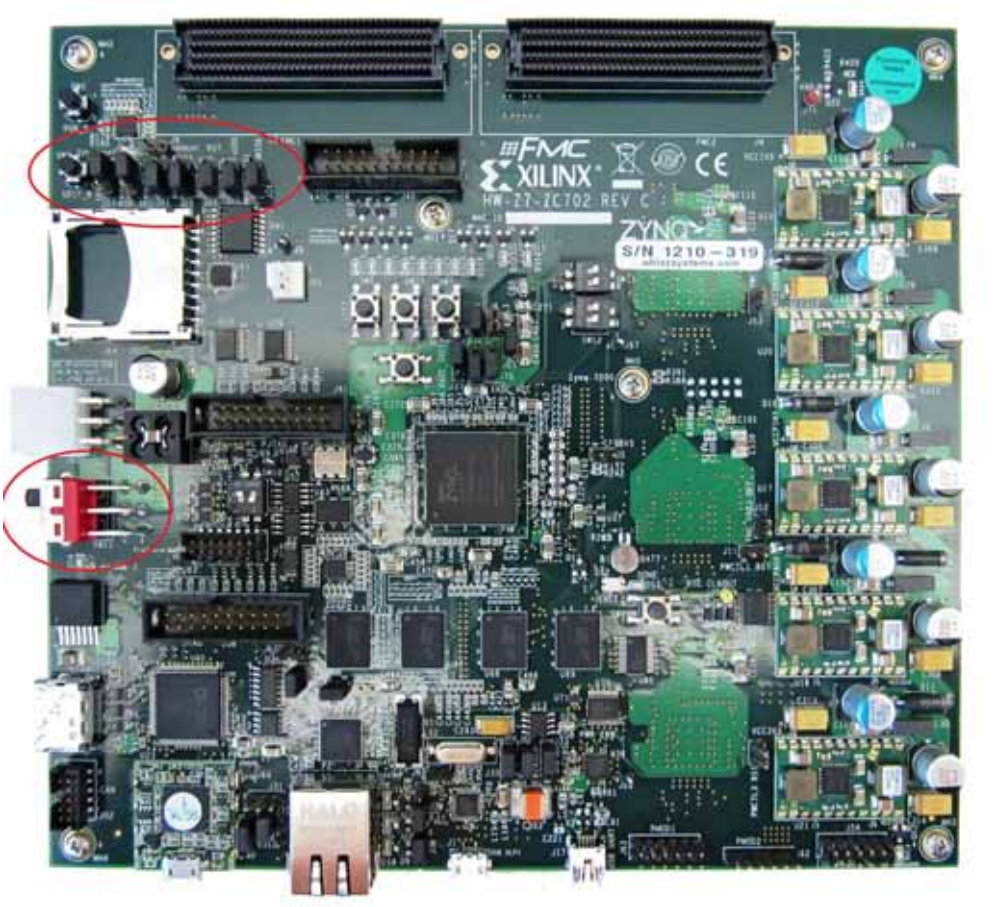

図 2-6 : ZC702 **ボードの電源スイッチ**

6. システムで割り当てられている COM ポートのシリアル通信ユーティリティを開きます。SDK ではシリアル ターミナル ユーティリティが提供されています。**[Window]** → **[Show View]** → **[Terminal]** をクリックしてこのユーティリティを開きます。

   注記 : Zynq プロセッシング システムの標準的なコンフィギュレーションは、「ボー レート 115200、8 ビット、パリティ : なし、停止 : 1 ビット、フロー制御 : なし」です。

7. **[File]** → **[New]** → **[Xilinx C Project]** をクリックします。
8. テンプレート リストで **[Hello World]** を選択し、残りのオプションはデフォルトのままにします。プロジェクトの位置、使用するハードウェア プラットフォーム、およびプロセッサがこのウィンドウに表示されます。ここから使用するプロセッサは `ps7_cortexa9_0` です。
9. **[Next]** をクリックします。
10. 次のページで、このプロジェクトの BSP を選択します。**[Finish]** をクリックして Hello World アプリケーションの BSP を生成します。
11. Hello World アプリケーションとその BSP はいずれもコンパイルされ、.elf ファイルが生成されます。
12. **[hello_world_0]** を右クリックして **[Run as]** → **[Run Configurations]** をクリックします。
13. **[Xilinx C/C++ ELF]** を右クリックして **[New]** をクリックします。
14. 新しい実行コンフィギュレーションが `hello_world_0 Debug` という名前で作成されます。

    アプリケーションに関連するコンフィギュレーションは、起動コンフィギュレーションの [Main] タブに自動入力されます。

15. 起動コンフィギュレーションの **[Device Initialization]** タブをクリックし、設定を確認します。

初期化 TCL ファイルへのコンフィギュレーション パスがあることに着目してください。ここでは ps7_init.tcl のパスです。このファイルは、デザインを SDK にエクスポートした際にエクスポートされたもので、プロセッシング システムの初期化情報を含みます。

16. [STDIO Connection] タブは、起動コンフィギュレーションの設定で使用できます。このタブを使用して STDIO をコンソールに接続させることができます。ここからは、シリアル通信ユーティリティがすでに起動しているため、このタブを使用することはありません。起動コンフィギュレーションにはほかにもオプションがありますが、これらについては後半で説明します。
17. **[Run]** をクリックします。
18. シリアル通信ユーティリティに Hello World が表示されます。

注記 : Zynq 評価ボードで実行されるこのソフトウェア アプリケーション用にビットストリームをダウンロードする必要はありませんでした。ARM Cortex A9 デュアル コアはすでにボードに実装されています。簡単なアプリケーションを実行することを目的としたこのシステムの基本的な初期化は、デバイス初期化 TCL スクリプトで実行されます。

ここまでの結果

アプリケーション ソフトウェアによって、Hello World の文字列が PS の UART1 ペリフェラルに送信されました。

UART1 からホスト マシンで動作しているシリアル ターミナル アプリケーションへ、Hello World の文字列がバイトごとに送信され、文字列として表示されます。

関連するトレーニング演習

このテスト ドライブに対応するトレーニング演習は、「EDK : ソフトウェアの追加とダウンロード」です。この演習では、SDK ツールでソフトウェア ボード サポート パッケージおよびサンプル アプリケーションを作成します。次にデバイスをコンフィギュレーションし、アプリケーションをダウンロードしてテストします。

トレーニング演習へのリンクは、付録 A 「その他のリソース」 に記載されています。

2.1.4 その他の情報

ボード サポート パッケージ

ボード サポート パッケージ (BSP) は、電源投入時の基本的な初期化やソフトウェア アプリケーションがハードウェア プラットフォームまたはボードの最上位で実行されるようにサポートする、ハードウェア プラットフォームまたはボードのサポート コードです。これは、ブートローダーおよびデバイス ドライバーを備えるオペレーティング システム固有のものにできます。

スタンドアロンの OS

スタンドアロンは、シンプルな下位ソフトウェア層です。これは、キャッシュ、割り込み、例外などの基本的なプロセッサ機能およびホスト環境の基本的なプロセッサ機能へのアクセスを提供します。これらの基本的な機能には、標準の入力/出力、プロファイリング、停止、終了が含まれます。これはセミホスト型のシングル スレッド環境です。

この章で実行したアプリケーションが、スタンドアロン OS の最上位に作成されました。

XILINX

第 3 章

Zynq プロセッシング システムおよびプログラマブル ロジックを使用するエンベデッド システム デザイン

ザイリンクスの Zynq™ EPP をエンベデッド デザインのプラットフォームとして使用する場合、独自の特徴の 1 つとして ARM Cortex A9 デュアル コア プロセッシング システムに Zynq のプロセッシング システム (PS) を使用できるだけでなく、プログラマブル ロジック (PL) も使用できます。

この章では、次を備えるデザインを作成します。

- AXI GPIO およびファブリックから PS への割り込みを持つ AXI タイマー
- PL で インスタンシエートされる ChipScope™ IP
- EMIO インターフェイスを介して PL 側のピンへ接続される Zynq PS GPIO ピン

この章のフローは、第 2 章で説明した内容と類似しています。ここでは、第 2 章の内容について繰り返し言及しているため、前章を省略した場合はその内容を確認する必要があります。

3.1 ファブリックでの Zynq PS への IP 追加

ファブリックで追加可能な Zynq PS と密に結合される IP は、どんなに複雑なものでも制限はありません。ここでは、AXI GPIO、割り込みを持つ AXI タイマー、EMIO インターフェイスを介して PL 側のピンへ接続される PS の GPIO ピン、そしてコンセプトを実証するための ChipScope インスタンシエーションを用いて簡単な例を説明します。

ここでは、デザインを作成して、AXI GPIO、ファブリックでインスタンシエートされた割り込み付き AXI タイマー、および EMIO インターフェイスを持つ PS の GPIO の機能を確認します。図 3-1 に、システムのブロック図を示します。

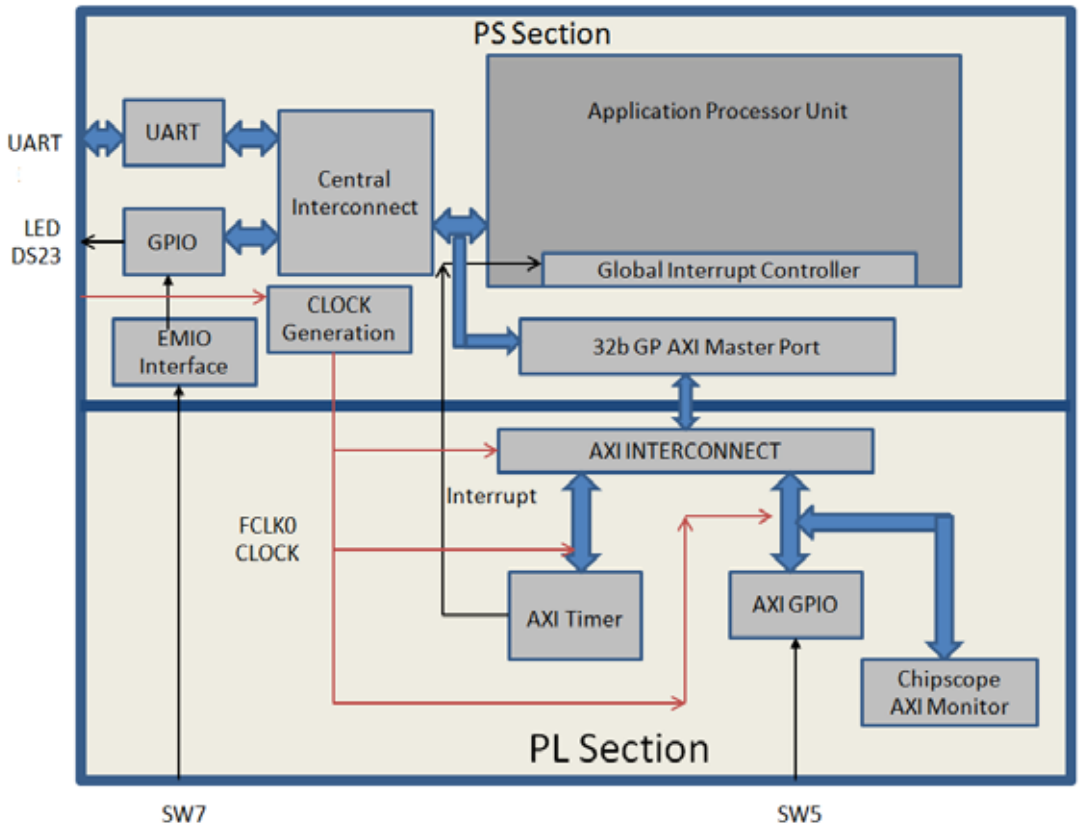

図 3-1 : ブロック図

このシステムは、次のように接続されています。

- ファブリック側の AXI GPIO は 1 ビット幅のチャネルのみを備え、ZC702 ボード上のプッシュボタン スイッチ SW5 へ接続されています。
- PS の GPIO も、EMIO インターフェイスを介してファブリック ピンへ配線される 1 ビット幅のインターフェイスを備えており、ボード上のプッシュボタン スイッチ SW7 へ接続されています。
- PS では、別の 1 ビット GPIO が MIO ポートにあるボード上の LED DS23 へ接続されています。
- AXI タイマーの割り込みはファブリックから PS の割り込みコントローラーへ接続されます。このタイマーは、ボード上の指定されたプッシュ ボタンを押すと開始します。タイマーが終了すると、その割り込みがトリガーされます。

アプリケーション ソフトウェアのコードを書き込みます。コードを実行する際、シリアル ターミナルにメッセージが表示され、ボードで使用するプッシュ ボタン スイッチ (SW7 または SW5 のいずれか) の選択が求められます。そのボタンを押すと、タイマーは動作を自動的に開始して LED DS23 をオフにし、タイマーの割り込みが発生するのを待機します。タイマーの割り込みが発生すると、LED DS23 がオンになり、タイマーは実行を再開してシリアル ターミナルでプッシュ ボタン スイッチが再度選択されるのを待機します。

後半のセクションで AXI モニターを使用したハードウェアのデバッグ方法について説明するため、ICON (ChipScope Integrated Controller) IP および AXI Monitor IP をデザインに追加します。

第 2 章の該当セクションは、このデザイン フローにも有効です。第 2 章で作成したシステムを使用し、「2.1.1 テスト ドライブ : Zynq プロセッシング システムの新規エンベデッド プロジェクトを作成する」に従います。

3.1.1  テスト ドライブ : ファブリックでインスタンシエートされる IP の機能を確認する

このテスト ドライブでは、AXI GPIO、ファブリックでインスタンシエートされる割り込み付き AXI タイマー、および EMIO インターフェイスの機能を確認します。

1. PlanAhead ツールの [Sources] ペインで、[system_i-system(system.xmp)] をダブルクリックして XPS を起動します。これは、11 ページの「テスト ドライブ : Zynq プロセッシング システムの新規エンベデッド プロジェクトを作成する」で作成したエンベデッド ソースです。
2. [XPS System Assembly] ビューで **[Bus Interfaces]** タブをクリックします。
3. IP カタログで **[General Purpose IO]** を展開し、**[AXI General Purpose IO]** をダブルクリックして追加します。

   axi_gpio 1.01.b IP インスタンスをデザインへ追加するかどうかを確認するメッセージが表示されます。
4. **[Yes]** をクリックします。

   GPIO のコンフィギュレーション ウィンドウが開きます。
5. [Channel 1] を展開してチャネル 1 のコンフィギュレーション パラメーターを表示させます。
6. [GPIO Data Channel Width] の値が 32 となっていることを確認します。デザインの動作には 1 ビットの入力のみが必要なため、この値を 1 に変更します。ほかのパラメーターは変更せずにそのままにします。
7. **[OK]** をクリックします。

   「axi_gpio IP with version number 1.01.b is instantiated with name axi_gpio_0」と表示されたメッセージ ウィンドウが開きます。接続先のプロセッサを決定するように求められます。ここでは、デュアル コア ARM プロセッサを使用して設計しています。また、このメッセージには、XPS がバス インターフェイスの接続およびアドレスの割り当てを実行し、IO ポートを外部にすることが表示されます。

   プロセッサのデフォルト選択は processing_system7_0 です。このままこれを使用します。
8. **[OK]** をクリックします。

   一部自動ではなく手動で接続すべきものがあります。

   注記 : AXI インターコネクトは、ファブリックの IP および PS のインターコネクト間で自動的にインスタンシエートされます。この例では、AXI GPIO が AXI インターコネクトを介して PS へ接続されます。
9. IP カタログで **[DMA and Timer]** を展開し、**[AXI Timer/Counter IP]** をダブルクリックして追加します。

   axi_timer_1.03.a IP インスタンスをデザインに追加するかどうかを確認するダイアログ ボックスが表示されます。
10. **[Yes]** をクリックします。

    TIMER のコンフィギュレーション ウィンドウが開きます。ほかのパラメーターは変更せずにそのままにします。

11. **[OK]** をクリックします。

    「axi_timer IP with version number 1.03.a is instantiated with name axi_timer_0.」と表示されたメッセージ ウィンドウが開きます。接続先のプロセッサを決定するように求められます。ここでは、デュアル コア ARM プロセッサを使用して設計しています。また、このメッセージには、XPS がバス インターフェイスの接続およびアドレスの割り当てを実行し、IO ポートを外部にすることが表示されます。

    プロセッサのデフォルト選択は processing_system7_0 です。このままこれを使用します。

12. **[OK]** をクリックします。

    このセクションの後半では、AXI タイマーの割り込みを PS の割り込みに手動で接続します。

13. IP カタログで **[Debug]** を展開し、2 つの IP (**ChipScope AXI Monitor** と **ChipScope Integrated Controller**) をデザインに追加します。いずれの IP のコンフィギュレーションも変更しません。

14. IP およびそれらのポートがリストされている **[Ports]** タブをクリックします。[axi_interconnect_1]、[axi_gpio_0、axi_timer_0]、[chipscope_axi_monitor_0]、および [chipscope_icon_0] を展開します。

15. 次の IP の接続を確認します。接続されていないものがあれば、ここで接続します。

IP	ポート	接続
axi_interconnect_1	INTERCONNECT_ACLK	processing_system7_0 :FCLK_CLK0
	INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N
axi_gpio_0	(BUS_IF) S_AXI::S_AXI_ACLK	processing_system7_0:FCLK_CLK0
	(IO_IF) gpio_0::GPIO_IO	External Port ::axi_gpio_0_GPIO_IO_pin
axi_timer_0	(BUS_IF) S_AXI_::S_AXI_ACLK	processing_system7_0 :FCLK_CLK0
Chipscope_axi_monitor_0	CHIPSCOPE_ICON_CONTROL	Chipscope_icon_0 ::control0
	(BUS_IF) MON_AXI::MON_AXI_ACLK	processing_system7_0 :FCLK_CLK0
Chipscope_icon_0	Control0	Chipscope_axi_monitor0::CHIPSCOPE_ICON_CONTROL

[Ports] タブの内容が図 3-2 のようになるようにします。

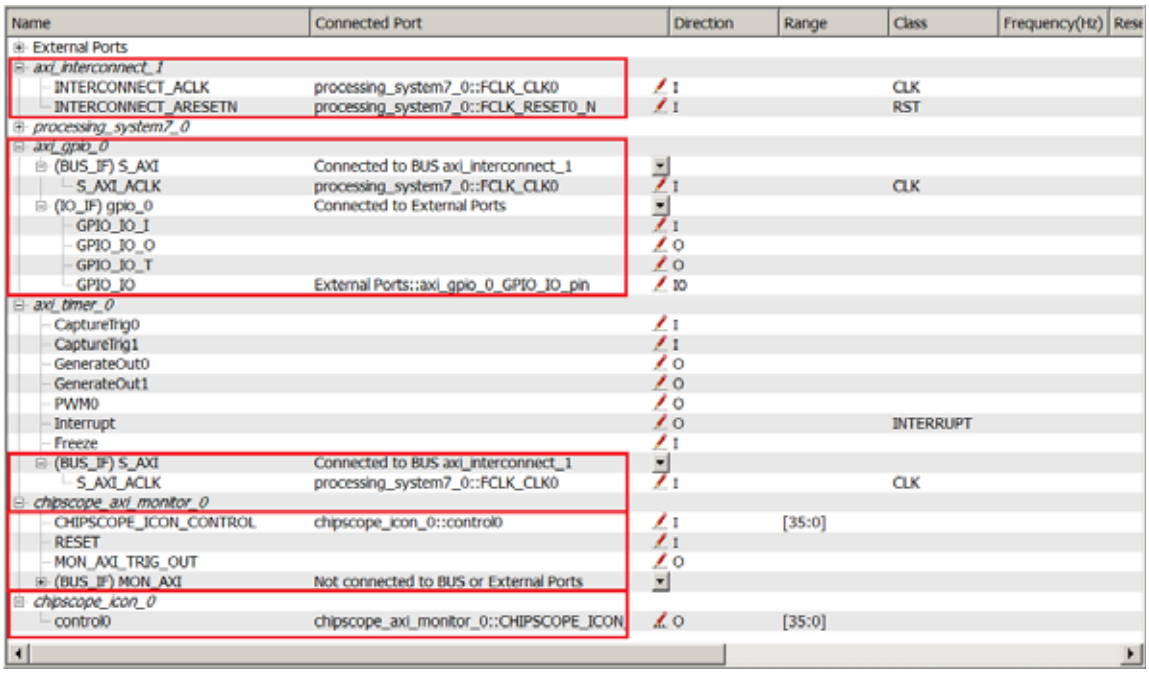

図 3-2 : 完成したポート接続

16. すべての IP を非展開にし、[processing_system7_0] を展開します。次のようにポートが接続されていない場合は、ここで接続します。図 3-3 のように接続します。

IP	ポート	接続
Processing_system7_0	(BUS_IF) M_AXI_GP0:: M_AXI_GPO_ACLK	processing_system7_0 ::FCLK_CLK0

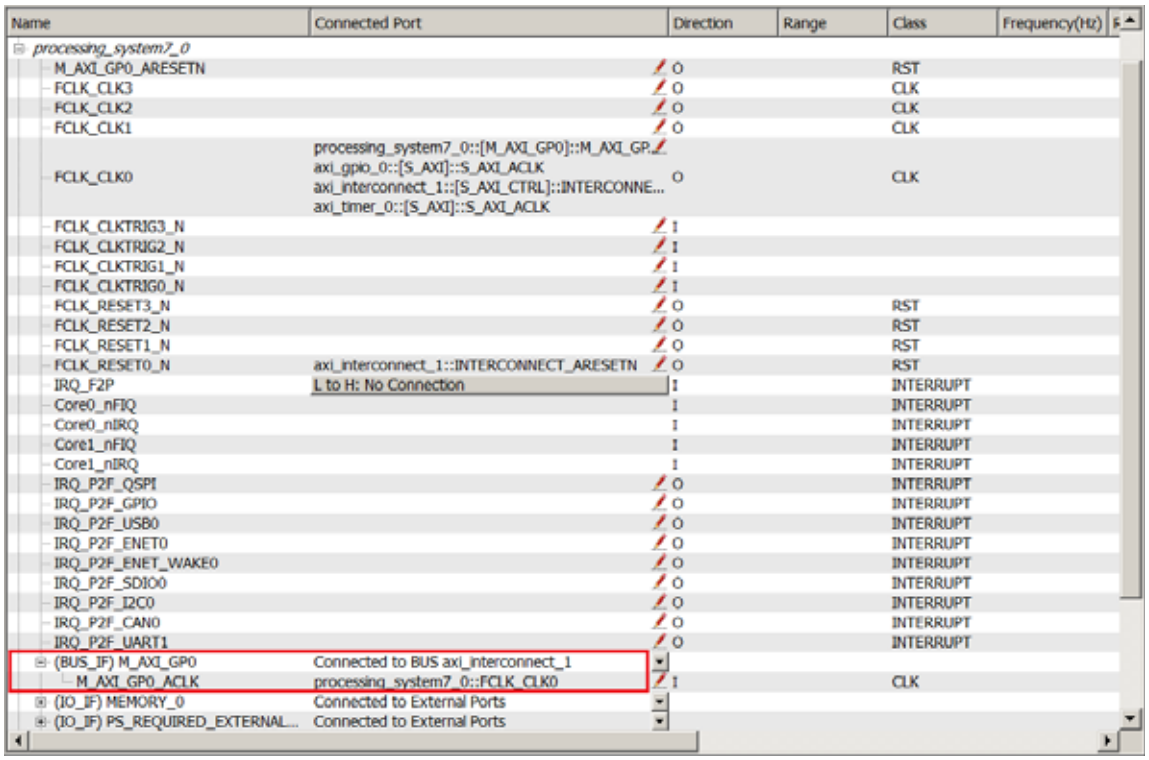

図 3-3 : 展開した processing_system7_0 と M_AXI_GP0_ACLK の接続を示す [Ports] タブ

17. 次の手順に従って、ファブリック側のタイマー割り込みを PS 側の割り込みコントローラーへ接続します。
    a. `[Processing_System7_0]` の [Connected Port] 列で **[L to H:No Connection]** をクリックします。

    [Interrupt Connection] ダイアログ ボックスが開きます。

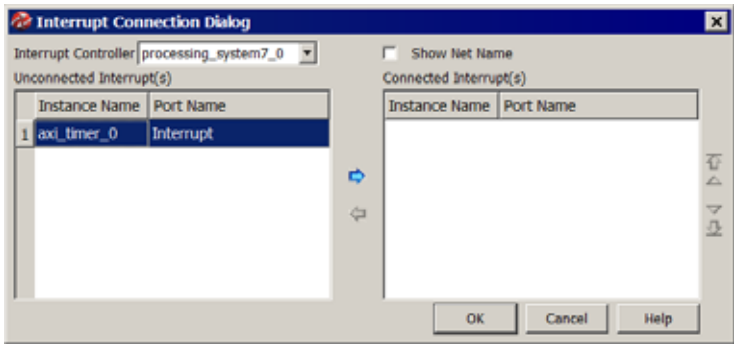

図 3-4 : [Interrupt Connection] ダイアログ ボックス

    b. [Unconnected Interrupts] リストで axi_timer_0 を選択し、矢印をクリックして選択したアイテムを [Connected Interrupts] リストへ移動させます。

図 3-5 に、割り込み ID 91 に接続される axi_timer_0 割り込みインスタンスを示します。

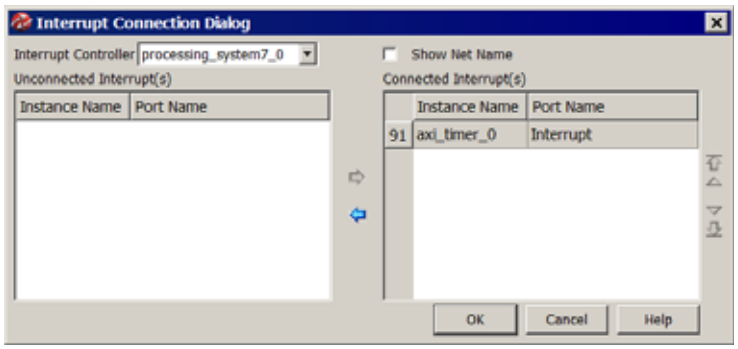

図 3-5 : 接続された割り込みを示す [Interrupt Connection] ダイアログ ボックス

c. **[OK]** をクリックします。

XPS はファブリック側のタイマー割り込みを PS の割り込みコントローラーへ接続します。

図 3-6 : ファブリック側で接続されたタイマー割り込み

18. **[Bus Interfaces]** タブをクリックし、[chipscope_axi_monitor_0] を展開します。
19. **[Bus Name]** 列で **[No Connection]** をクリックします。表示されるドロップダウン リストで chipscope_axi_monitor を axi_gpio_0.S_AXI に接続します。

このように接続することで、axi_gpio_0 スレーブ AXI バス上のあらゆる AXI 関連のトランザクションを ChipScope Analyzer でモニタリングできます。

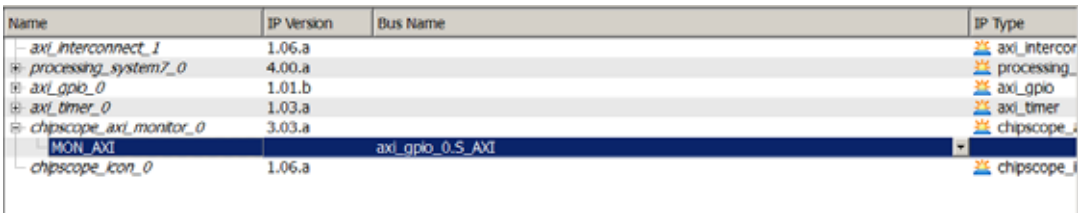

図 3-7 : 接続された chipscope_axi_monitor

20. 次の手順に従い、EMIO インターフェイスを使用して PS の GPIO を PL 側のパッドへ配線します。
    a. [XPS System Assembly] ビューで **[Zynq]** タブをクリックします。
    b. 緑色の **[General]** をクリックして [XPS Core Config] ダイアログ ボックスを開きます。
    c. **[User]** タブで **[General]** のアイテムを展開します。
    d. **[Enable GPIO on EMIO Interface]** チェックボックスをオンにします。

ヒント : [XPS Core Config] ダイアログ ボックスのアイテムに対応するチェック ボックスが表示されない場合は、ウィンドウの右側をクリックしてドラッグし、展開させます。

[Width of GPIO on EMIO interface] の設定は、次の行で有効になります。デフォルトの設定は 64 です。

    e. GPIO の幅を **1** に設定して **[OK]** をクリックします。
    f. [System Assembly] ビューで [Ports] タブをクリックし、[processing_system7_0] を展開します。GPIO ポートが外部ポートに接続されていないことがわかります。

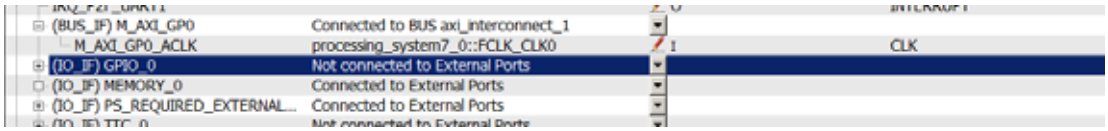

図 3-8 : 外部ポートに接続されていない GPIO ポート

21. [(IO_IF)GPIO_0] を展開し、**[GPIO]** をクリックします。
22. **[Connected Port]** 列でドロップダウン リストの矢印をクリックし、**[External Ports]** を選択します。

    この接続によって、この章の後半では PL のピン位置がユーザー制約ファイル (UCF) で PS GPIO に割り当てられます。

23. デザイン ルール チェックの実行コンソールにエラーがないことを確認します。

    注記 : エラーがある場合は、実行した手順を再度確認します。

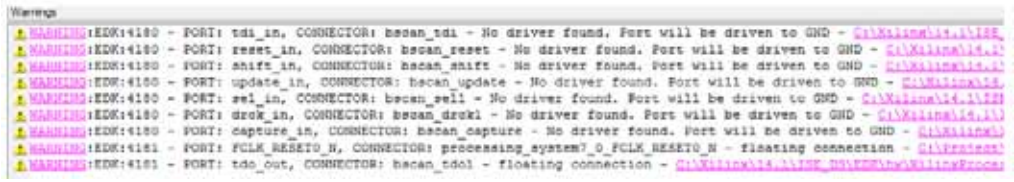

図 3-9 : デザイン ルール チェックの警告メッセージ

24. XPS を閉じます。PlanAhead™ デザイン ツール ウィンドウが再度アクティブになります。
25. [Design Sources] でエンベデッド ソースをクリックしてから右クリックし、**[Create Top HDL]** をクリックします。PlanAhead ツールによって system_stub.v ファイルが生成されます。
26. Flow Navigator の [Project Manager] リストで **[Add Sources]** を選択します。
27. ダイアログ ボックスが開くので、**[Add or Create Constraints]** をクリックしてから **[Next]** をクリックします。
28. **[Create File]** をクリックします。[Create Constraints File] ダイアログ ボックスが開くので、ファイル名に「**system**」と入力して **[OK]** をクリックします。
29. **[Finish]** をクリックします。

30. [Sources] ペインで **[Constraints]** フォルダーを展開します。空の system.ucf ファイルが constrs_1 に追加されていること確認します。

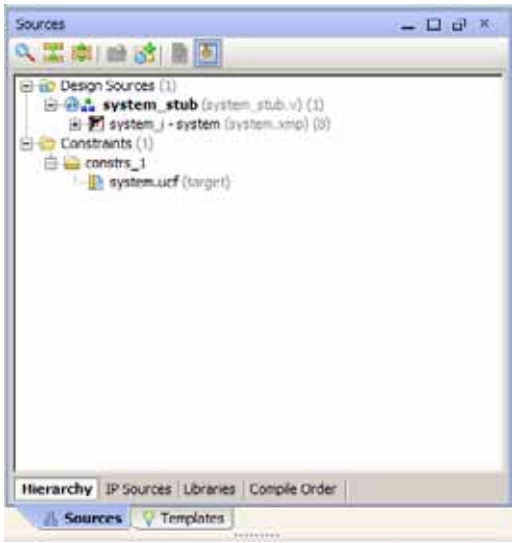

図 3-10 : system.ucf ファイルの追加

31. UCF ファイルで次のテキストを入力します。

```
# Connect to Push Button "SW5"
NET axi_gpio_0_GPIO_IO_pin IOSTANDARD=LVCMOS25 | LOC=G19;
# Connect to Push Button "SW7"
NET processing_system7_0_GPIO_pin IOSTANDARD=LVCMOS25 | LOC=F19;
```

次のように設定されます。

- NET 「axi_gpio_0_IO_pin」 の LOC 制約によって、AXI GPIO ピンが PL の G19 ピンに接続され、さらにそのピンがボード上の SW5 プッシュ ボタンに物理的に接続されます。
- NET 「processing_system7_0 GPIO pin」 の LOC 制約によって、PS の GPIO が PL の F19 ピンに接続され、さらにそのピンがボード上の SW7 プッシュ ボタンに物理的に接続されます。
- IOSTANDARD=LVCMOS25 制約によって、いずれのピンも LVCMOS 2.5V I/O 規格に設定されます。

32. 変更したファイルをすべて保存します。
33. Flow Navigator の [Program and Debug] リストで **[Generate Bitstream]** をクリックします。表示される警告メッセージはすべて無視します。
34. ビットストリームの生成が完了したら、第 2 章の説明に従ってハードウェアをエクスポートして SDK を起動します。このデザインでは、PL ファブリック用に生成されたビットストリームがあるため、これも SDK へエクスポートされます。

関連するトレーニング演習

このテスト ドライブに対応するトレーニング演習は、「SDK : 基本システムのインプリメンテーション」 です。この演習では、Processing System Configuration ウィザード (Zynq EPP) を使用してハードウェア デザインを作成します。次に、基本となるソフトウェア プラットフォームを指定してシステムにソフトウェア アプリケーションを追加します。

トレーニング演習へのリンクは、付録 A 「その他のリソース」 に記載されています。

3.1.2 テスト ドライブ : SDK を使用する

1. SDK は、第 2 章でスタンドアロン PS を用いて作成した Hello World プロジェクトで起動します。
2. **[Project]** → **[Clean]** をクリックし、プロジェクトを消去して再度構築します。
3. `helloworld.c` ファイルを開き、31 ページの「デザイン用のスタンドアロン アプリケーション ソフトウェア」の説明に従ってアプリケーション ソフトウェア コードを変更します。
4. ボー レートが **115200** に設定されているシリアル通信ユーティリティを開きます。
5. ボードを接続します。
6. PL ファブリックのビットストリームがすでにあるため、これをダウンロードします。まず、**[Xilinx Tools]** → **[Program FPGA]** をクリックします。図 3-11 に示す [Program FPGA] ダイアログ ボックスが開きます。PlanAhead からエクスポートされたビットストリームが表示されます。

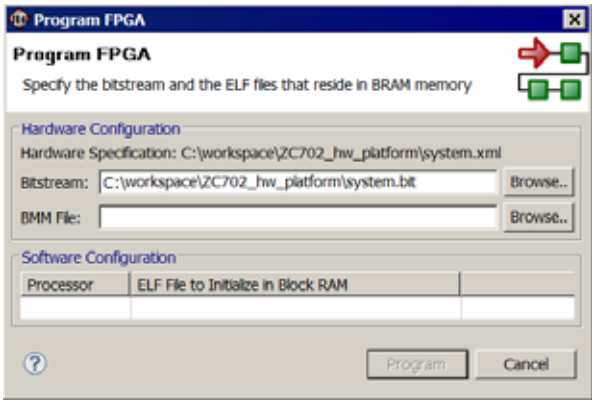

図 3-11 : [Program FPGA] ダイアログ ボックス

7. **[Program]** をクリックし、ビットストリームをダウンロードして PL ファブリックをプログラムします。
8. 19 ページの「テスト ドライブ : Hello World アプリケーションを実行する」と同じ手順でプロジェクトを実行します。
9. システムでは、AXI GPIO ピンがボード上のプッシュ ボタン SW5 に接続され、PS の GPIO ピンが EMIO インターフェイスを介してボード上のプッシュ ボタン SW7 に接続されます。
10. シリアル ターミナルの手順に従って、アプリケーションを実行します。

3.2 デザイン用のスタンドアロン アプリケーション ソフトウェア

この章で設計したシステムをボードで実行するには、アプリケーション ソフトウェアが必要です。ここでは、アプリケーション ソフトウェアについて詳細に説明します。

アプリケーション ソフトウェアの `main()` 関数は、実行のエントリ ポイントです。この関数は、初期化およびシステムで接続されるすべてのペリフェラルに必要な設定を含みます。さらに、AXI GPIO や EMIO インターフェイスを使用する PS GPIO などのユース ケースを選択するための手順が含まれます。シリアル ターミナルの手順に従ってさまざまなユース ケースを選択できます。

3.2.1 アプリケーション ソフトウェアの構成手順

アプリケーション ソフトウェアは次の手順で構成されます。

1. AXI GPIO モジュールを初期化します。
2. AXI GPIO ピンが入力ピンとなるように方向を設定します。このピンは、ボード上の SW5 プッシュ ボタンに接続されます。この位置はシステムの構築中にユーザー制約ファイル (UCF) の LOC 制約で固定します。
3. AXI TIMER モジュールをデバイス ID 0 で初期化します。
4. タイマー コールバック関数を AXI timer ISR と関連付けます。

   この関数は、タイマーの割り込みが発生するたびに呼び出されます。このコールバックにより、ボード上の LED DS23 がオンになり、割り込みフラグがセットされます。

   `main()` 関数は割り込みフラグを用いて実行を停止し、タイマー割り込みの発生を待機した後に実行を再開します。
5. タイマーのリセット値を設定します。この値は、リセットおよびタイマーの開始時にタイマーへロードされます。
6. Interrupt モードや Auto Reload モードなどのタイマー オプションを設定します。
7. PS の GPIO を初期化します。
8. PS の GPIO (チャネル 0、ピン番号 10) を出力ピンに設定します。このピンは MIO ピンにマップされ、ボード上の LED DS23 に物理的に接続されます。
9. PS の GPIO (チャネル 2、ピン番号 0) を入力ピンに設定します。このピンは、EMIO インターフェイスを介して PL 側のピンにマップされ、SW7 プッシュ ボタン スイッチに物理的に接続されます。
10. スヌープ制御ユニットであるグローバル割り込みコントローラーを初期化します。タイマー割り込みルーチンを割り込み ID 91 に設定し、例外ハンドラーを登録して割り込みを有効にします。
11. ループの中でシーケンスを実行し、シリアル ターミナルで AXI GPIO または PS GPIO いずれかのユース ケースを選択します。

    ソフトウェアは、選択したユース ケースをシリアル ターミナルから受け取り、適宜手順を実行します。

    シリアル ターミナルでユース ケースを選択した後、ターミナルの手順に従ってボード上のプッシュ ボタンを押します。LED DS23 をオフにすると、タイマーが開始してタイマー割り込みが発生するまで関数を無限に待機させます。タイマー割り込みの発生後、LED DS23 がオンになって実行が再開されます。

API に関連するデバイス ドライバーの詳細は、『Zynq-7000 ソフトウェア開発者向けガイド』(UG821) を参照してください。この資料へのリンクは、付録 A 「その他のリソース」に記載されています。

3.2.2 アプリケーション ソフトウェアのコード

システムのアプリケーション ソフトウェアは、このガイドで提供されている `ug873_design_files.zip` ファイルから入手可能な `helloworld.c` に含まれています。この ZIP ファイルのへのリンクは、付録 A 「その他のリソース」に記載されています。

XILINX

第 4 章

SDK および ChipScope を使用したデバッグ

この章では、これまで説明してきたデザイン フローで可能な 2 通りのデバッグについて説明します。1 つ目は、SDK を使用してソフトウェアでデバッグする方法です。2 つ目の方法は、ChipScope™ ソフトウェアでサポートされるハードウェア デバッグです。

4.1  テスト ドライブ : SDK を使用してソフトウェアをデバッグする

まずは、SDK を使用してソフトウェアのデバッグを実行します。

1. [C/C++] パースペクティブでは、Hello_world_0 プロジェクト上で右クリックして **[Debug As]** → **[Debug Configurations]** をクリックします。設定がデバッグに適切であるかどうかを確認します。
2. **[Debug]** をクリックします。

   システムのリセット プロパティに関して確認を求めるダイアログ ボックスが表示されます。
3. **[OK]** をクリックします。

   別のダイアログ ボックスが表示され、このような起動は一時停止時に [Debug] パースペクティブを開くように設定されていることを知らせます。
4. **[Yes]** をクリックします。[Debug] パースペクティブが開きます。

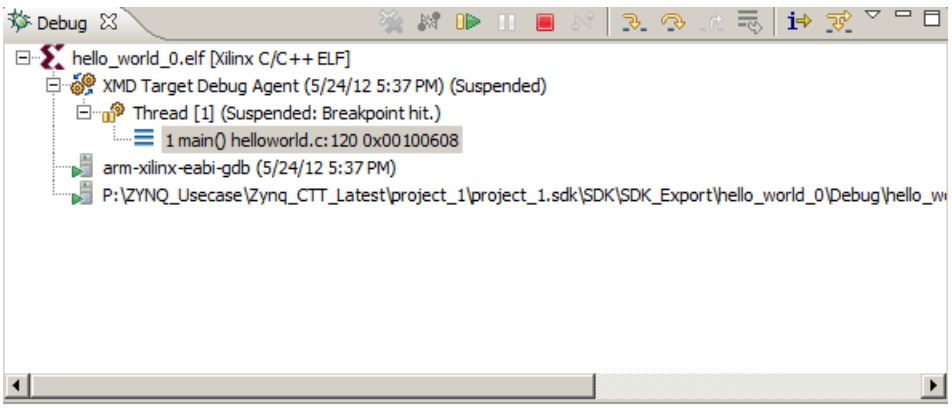

図 4-1 : [Debug] パースペクティブの一時停止

注記 : このページに示されているアドレスは、システムで表示されているものと若干異なる場合があります。

プロセッサが `main()` の始めの部分にあり、プログラム実行が 0x00100608 で停止していることがわかります。この情報は [Disassembly] ビューで確認できます。ここでは、アセンブリ レベルのプログラム実行についても 0x00100608 で一時停止されていることがわかります。

注記 : [Disassembly] ビューが表示されない場合は、**[Window]** → **[Show view]** → **[Disassembly]** をクリックします。

helloworld.c ウィンドウにも、実行が C コードの最初の実行可能な行で一時停止していることが表示されます。[Registers] ビューを選択し、プログラム カウンターである pc レジスタが 0x00100608 を含むことを確認します。

注記 : [Registers] ウィンドウが表示されない場合は、**[Window]** → **[Show View]** → **[Registers]** をクリックします。

5. `init_platform ()` を読み出すコードの行の横にある、helloworld.c ウィンドウの空白をダブルクリックします。これにより、`init_platform ()` でブレークポイントが設定されます。ブレークポイントの確認には、[Breakpoints] ウィンドウを参照します。

   注記 : [Breakpoints] ウィンドウが表示されない場合は、**[Window]** → **[Show View]** → **[Breakpoints]** をクリックします。

6. **[Run]** → **[Resume]** をクリックし、ブレークポイントまでプログラムの実行を再開させます。

   プログラムの実行は、`init_platform ()` を含むコードの行で停止します。[Disassembly] および [Debug] の両ウィンドウで、プログラムの実行が 0x00100630 で停止していることが示されます。

7. **[Run]** → **[Step Into]** をクリックし、`init_platform ()` ルーチンへステップインします。

   プログラムの実行が 0x00100C44 の位置で一時停止します。コール スタックの深さはここで 2 レベルです。

8. **[Run]** → **[Resume]** を再度クリックし、プログラムを終了まで実行します。

   プログラムの実行が完了すると、[Debug] ウィンドウはプログラムが `exit` というルーチンで一時停止していることを示します。これは、デバッガーの制御下で実行しているときに発生します。

9. コードを複数回再実行します。シングル ステップ、メモリの検査、ブレークポイント、コードの変更、および print 文の追加を試してみます。ビューの追加および移動を試します。
10. SDK を閉じます。

関連するトレーニング演習

このテスト ドライブに対応するトレーニング演習は、「SDK : デバッグ」です。この演習では、SDK の [Debug] パースペクティブおよびストップウォッチ アプリケーションを起動します。これらを使用して、デバッグの演習、ブレークポイントの設定、割り込みレイテンシの計算、およびプログラム操作を実践できます。

トレーニング演習へのリンクは、付録 A 「その他のリソース」に記載されています。

4.2  テスト ドライブ : ChipScope ソフトウェアを使用してハードウェアをデバッグする

次に、「3.1.2 テスト ドライブ : SDK を使用する」で作成した同じアプリケーション使用し、ChipScope ソフトウェアでハードウェアのデバッグを実行します。

1. 「3.1.2 テスト ドライブ : SDK を使用する」の記載に従って、ビットストリームとアプリケーションを ZC702 に再度ダウンロードします。
2. アプリケーションを実行して SDK を閉じます。
3. ChipScope Pro™ Analyzer を開きます。
4. ハードウェアが、ザイリンクスのプラットフォーム ケーブルと Digilent 社製ケーブル経由でコンピューターの USB ポートへ接続されていることを確認します。ザイリンクスのプラットフォーム ケーブル ドライバーもインストールしておく必要があります。
5. **[Open/Search JTAG Cable]**  をクリックします。
6. **[OK]** をクリックします。
7. `*.cdc` ファイルを ChipScope でインポートし、次を実行します。
   a. **[Dev 1 Mydevice1(XC7020)]** を選択します。
   b. **[File]** → **[Import]** をクリックします。
   c. **[Select New File]** をクリックして `Chipscope_axi_monitor_0.cdc` ファイルを `<project_path>\<project_name>.srcs\sources_1\edk\system\implementation\chipscope_axi_monitor_0_wrapper` から選択します。
   d. **[OK]** をクリックします。
8. 次のように、ARVALID 信号にトリガーを設定します。
   a. [Trigger Setup] ウィンドウを展開します。
   b. M1:MON_AXI_ARADDRCONTROL ユニットで、axi_gpio_0_S_AXI/MON_AXI_AVALID の値をデフォルトの **X** から **1** に変更します。これにより、この信号での有効なトランザクションによって波形が生成されます。

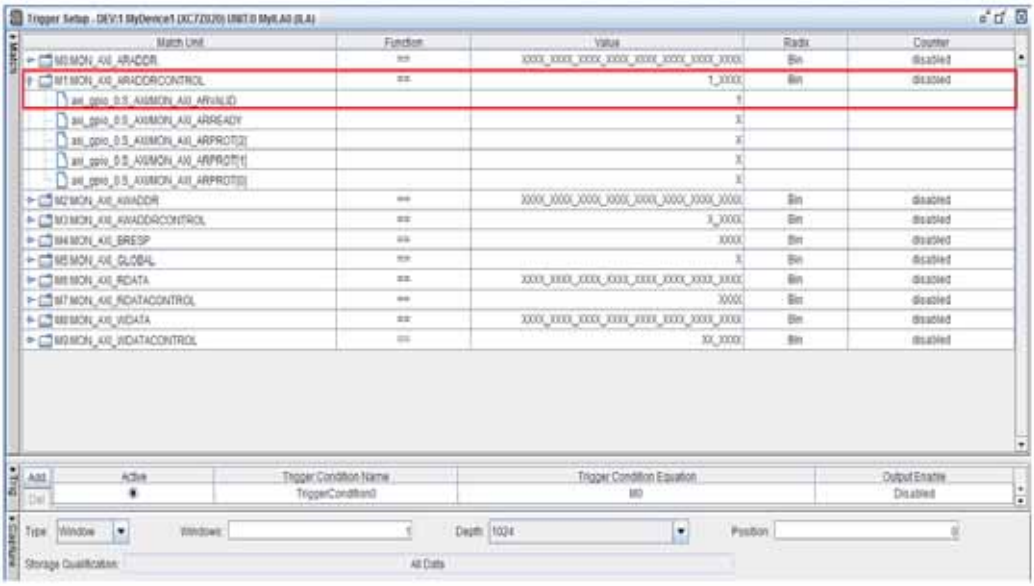

図 4-2 : [Trigger Setup] **ウィンドウ** : MON_AXI_AVALID **の設定**

c. [Trigger Setup] ウィンドウの [Trig] セクションで、**[Trigger Condition Equation]** 列の **[M0]** をクリックします。[Trigger Condition] ダイアログ ボックスが開きます。

d. **[Enable]** 列で **[M0]** をオフにして **[M1]** をオンにします。

トリガー チャネルが M0 から M1 に変更され、ARVALID 信号が M1 チャネル上に現れます。

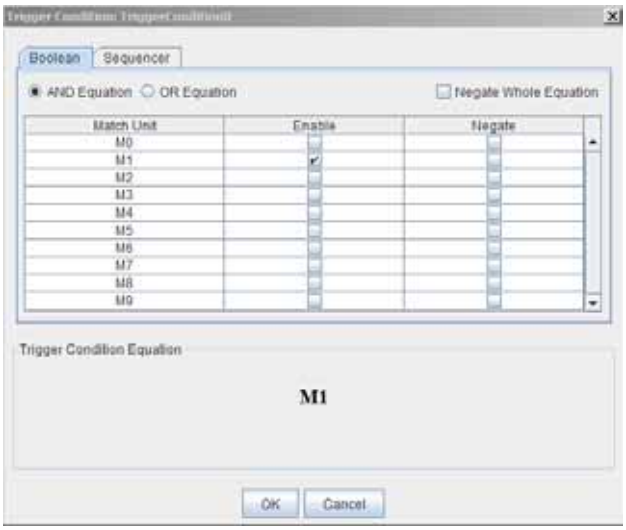

図 4-3 : [Trigger Condition] **ダイアログ ボックス**

9. **[OK]** をクリックします。
10. [Trigger Setup] ウィンドウの [Capture] セクションで、**[Position]** の設定を **0** から **512** に変更します。

    サンプルのワード数が 1024 に変更されるので、トリガー ポイントが波形の中央に移動します。
11. **[Run]**  をクリックします。

    ChipScope Analyzer はトリガー イベントを待機します。
12. シリアル ターミナルの手順に従って、AXI GPIO ユース ケースを選択します。これにより、波形が生成されます。

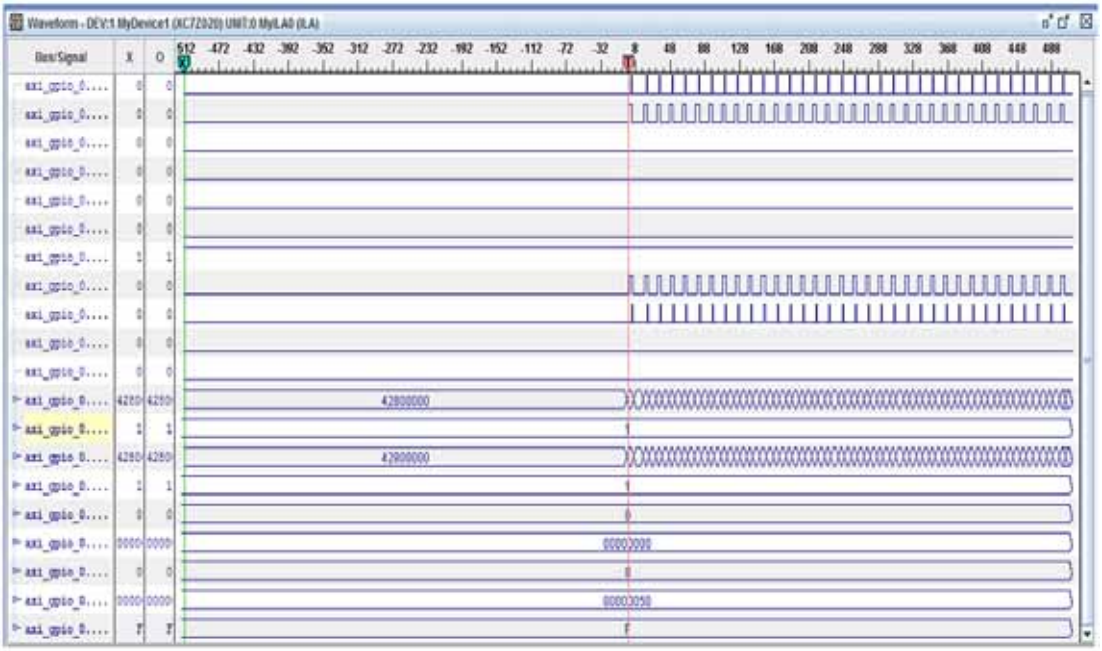

図 4-4 : 波形

関連するトレーニング演習

このテスト ドライブに対応する演習は、「エンベデッド システム開発 - 実践編 : ChipScope Pro Analyzer を使用したデバッグ」です。この演習では、ChipScope Pro Analyzer ソフトウェア、SDK Debug Perspective (GDP)、および XMD を用いてハードウェアとソフトウェアのデバッグを同時に実行します。

トレーニング演習へのリンクは、付録 A 「その他のリソース」に記載されています。

XILINX

第 5 章

SDK を使用した Linux のブートおよびアプリケーションのデバッグ

この章では、Zynq™-7000 EPP ボード上で Linux OS をブートする手順を説明します。また、JTAG インターフェイスを使用してターゲット メモリ上で Linux によってコンパイル済みのイメージをダウンロードする方法も提供します。この章の後半では、Linux でコンパイル済みのイメージがロードされている次の不揮発性メモリをプログラムする方法について説明します。これらのイメージは、ボードをオンにした後に Linux の自動ブートに使用されます。

- オンボード QSPI フラッシュ
- SD カード

SDK のリモート デバッグ機能を用いて、ターゲット ボードで実行している Linux アプリケーションをデバッグする方法についても解説します。SDK ツール ソフトウェアは Windows ホスト マシンで実行します。アプリケーションのデバッグ用に、Linux OS をすでに実行しているターゲット ボードへのイーサネット接続が SDK によって確立されます。

5.1 必要な環境

この章では、ターゲット プラットフォームとは Zynq ボードを指します。ホスト プラットフォームは ISE® Design Suite ツールを実行する Windows マシンのことです。

注記 : この章のチュートリアルには、一般的なブートローダーである Das U-Boot が必要です。これは、次にダウンロードするコンパイル済みのイメージに含まれます。

ザイリンクスの資料サイトから、`ug873_design_files.zip` ファイルをダウンロードします。この資料へのリンクは、付録 A 「その他のリソース」に記載されています。この ZIP ファイルには、次のファイルが含まれています。

- `BOOT.bin` : bootgen で生成された FSBL および U-Boot イメージを含むバイナリ イメージ
- `boot.bif` : BOOT.BIN の作成中に bootgen を制御するファイル
- `cdma_app` : 第 6 章で作成するシステム用のスタンドアロン アプリケーション ソフトウェア
- `devicetree.dtb` : Linux が使用するデバイス ツリーのバイナリ ラージ オブジェクト (blob)、U-Boot でメモリにロードされる
- `helloworld.c` : 第 3 章で作成したシステム用のスタンドアロン アプリケーション ソフトウェア
- `linux_cdma_app` : 第 6 章で作成するシステム用の Linux OS ベースのアプリケーション ソフトウェア
- `ramdisk8M.image.gz` : Linux が使用する Ramdisk イメージ、U-Boot でメモリにロードされる
- `README.txt` : リリースの説明
- `u-boot.elf` : BOOT.BIN イメージの作成に使用する U-Boot ファイル
- `zImage` : Linux カーネル イメージ、U-Boot でメモリにロードされる
- `zynq_fsbl_0.elf` : BOOT.BIN イメージの作成に使用する FSBL イメージ
- `stub.tcl` : 「5.2.3 テスト ドライブ : JTAG モードで Linux をブートする」の手順で使用するスクリプト ファイル

5.2 Zynq ボード上で Linux をブートする

このセクションでは、「5.1 必要な環境」でダウンロードしたコンパイル済みのイメージを使用し、ターゲット ボード上で Linux をブートするフローについて説明します。

注記 : カーネル イメージ、U-Boot、デバイス ツリー、ルート ファイルなどのさまざまなイメージのコンパイルについては、このガイドでは割愛します。

5.2.1 ブート方法

次の 2 通りの方法でブート可能です。

- 「マスター ブート方法」
- 「スレーブ ブート方法」

マスター ブート方法

マスター ブート方法では、QSPI、NAND、NOR フラッシュ、および SD カードなどの異なる種類の不揮発性メモリにブート イメージを保存できます。この方法で、CPU は外部のブート イメージを不揮発性メモリからプロセッサ システム (PS) にロードして実行します。マスター ブート方法には、さらにセキュア モードと非セキュア モードがあります。詳細は、『Zynq-7000 エクステンシブル プロセッシング プラットフォーム テクニカル リファレンス マニュアル』(UG585) を参照してください。この資料へのリンクは、付録 A 「その他のリソース」に記載されています。

ブート プロセスは、プロセッシング システム (PS) にある ARM Cortex-A9 CPU のうちの 1 つで開始され、オンチップ ROM コードを実行します。このオンチップ ROM コードが、FSBL (第 1 段階ブートローダー) をロードします。FSBL は次を実行します。

- ハードウェア ビットストリームがある場合は、これで FPGA をコンフィギュレーションする
- MIO インターフェイスをコンフィギュレーションする
- DDR コントローラーを初期化する
- クロック PLL を初期化する
- Linux の U-Boot イメージを不揮発性メモリから DDR へロードして実行する

U-Boot はカーネル イメージ、ルート ファイル システムおよびデバイス ツリーを不揮発性 RAM から DDR へロードし、これらの実行を開始します。ターゲット プラットフォームでの Linux のブートを終了させます。

スレーブ ブート方法

JTAG は、スレーブ ブート モードでしか使用できません。外部のホスト コンピューターがマスターとして動作し、JTAG 接続を使用してブート イメージを OCM へロードします。

注記 : ブート イメージのロード中、PS CPU はアイドル モードのままです。スレーブ ブート方法でのブートは常に非セキュア モードです。

JTAG ブート モードでは、CPU はセキュリティ関連のすべてのアイテムへのアクセスを無効にし、JTAG ポートを有効にした直後に停止モードに遷移します。CPU の実行を再開させる前に、ブート イメージを DDR メモリにダウンロードする必要があります。

5.2.2 JTAG から Linux をブートする

次のフロー図は、ターゲット プラットフォームで Linux をブートするためのプロセスを示しています。

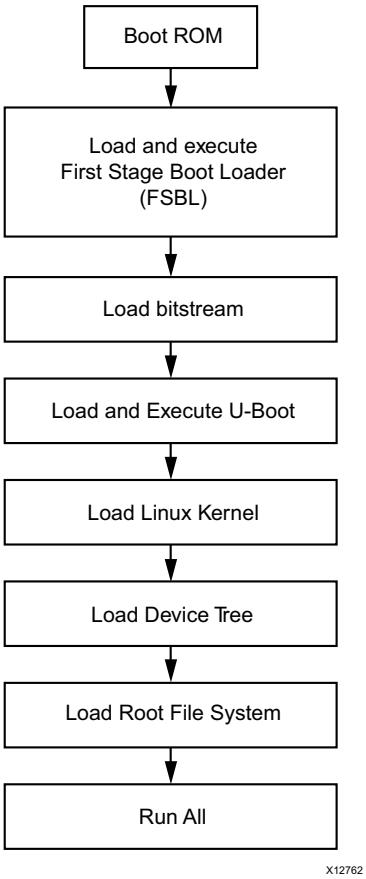

図 5-1 : ターゲット プラットフォームでの Linux ブート プロセス

5.2.3  テスト ドライブ : JTAG モードで Linux をブートする

1. JTAG モードを使用して Linux をブートする場合は、次のボード接続および設定を確認してください。
   - 19 ページの「テスト ドライブ : Hello World アプリケーションを実行する」の説明に従って、ジャンパー J27 および J28 が設定されていることを確認します。
   - Zynq ボードからのイーサネット ケーブルを Windows ホスト マシンに接続します。
   - 電源ケーブルをボードに接続します。
2. ザイリンクスのプラットフォーム ケーブル USB II または Digilent 社製ケーブルを用いて Windows ホスト マシンとターゲット ボードを次のように接続します。

   ザイリンクス プラットフォーム ケーブル USB II を使用する場合は、スイッチ SW10 を次のように設定します。

   a. ビット 1 の位置を ON 側に設定する
   b. ビット 2 の位置を ON 側の反対方向に設定する

   Digilent 社製ケーブルを使用する場合は、スイッチ SW10 を次のように設定します。

   a. ビット 1 の位置を ON 側の反対方向に設定する
   b. ビット 2 の位置を ON 側に設定する
3. USB ケーブルを Windows ホスト マシンと接続しているターゲット ボードのコネクタ J17 につなぎます。シリアル転送に USB を使用する際、このように接続します。
4. 図 5-2 のようにイーサネット ジャンパー J30 および J43 を変更します。

図 5-2 : ジャンパー J30 および J43 の変更

5. ターゲット ボードに電源を投入します。
6. SDK を起動して、第 2 章および第 3 章と同じワークスペースを開きます。
7. シリアル ターミナルが開かない場合は、シリアル通信ユーティリティを 115200 に設定されているボー レートに接続します。
8. **[Xilinx Tools] → [Program FPGA]** をクリックしてから **[Program]** をクリックし、ビットストリームをダウンロードします。
9. **[Xilinx Tools] → [XMD console]** をクリックして XMD ツールを開きます。
10. XMD プロンプトで次を実行します。
    a. 「**connect arm hw**」と入力して PS の CPU1 に接続します。
    b. 「**source <Project Dir>/project_1/project_1.sdk/SDK_Export/system_hw_platform/ps7_init.tcl**」と入力した後、「**ps7_init**」と入力して PS を初期化します (クロック PLL、MIO、DDR などの初期化)。
    c. 「**source directory/stub.tcl**」と入力します。

       CPU2 は CPU1 からのイベントを継続的に待機します。

       注記 : `stub.tcl` ファイルは、このガイドで提供されている `ug873_design_files.zip` から入手できます。この ZIP ファイルへのリンクは、付録 A 「その他のリソース」に記載されています。
    d. 「**target 64**」と入力して実行制御を CPU1 に与えます。

e. 「**dow directory/u-boot.elf**」と入力して Linux U-Boot をダウンロードします。

f. 「**con**」と入力して U-Boot の実行を開始します。

シリアル ターミナルに、次のような自動ブート カウントダウン メッセージが表示されます。

「Hit any key to stop autoboot:3」

g. **[Enter]** をクリックします。

U-Boot からの自動ブートが停止し、シリアル ターミナルにコマンド プロンプトが表示されます。

h. XMD プロンプトに「**stop**」と入力します。

U-Boot の実行が停止します。

i. 「**dow -data directory/zImage 0x8000**」と入力して Linux カーネル イメージ (zImage) を 0x8000 にダウンロードします。

j. 「**dow -data directory/ramdisk8M.image.gz 0x800000**」と入力して Linux のルート ファイル システムのイメージを 0x800000 にダウンロードします。

k. 「**dow -data directory/devicetree.dtb 0x1000000**」と入力して Linux のデバイス ツリーを 0x1000000 にダウンロードします。

l. 「**con**」と入力して U-Boot の実行を開始します。

11. シリアル ターミナルのコマンド プロンプトに「**go 0x8000**」と入力します。

Linux OS が起動します。ブートが完了すると、Zynq> プロンプトがシリアル ターミナルに表示されます。

12. Zynq> プロンプトで次を実行します。

a. Zynq> プロンプトで次のコマンドを入力してボードの IP アドレスを設定します。
```
ifconfig eth0 10.10.70.120 netmask 255.255.255.0
```

このコマンドによって、ボードの IP アドレスが 10.10.70.120 に設定されます。

b. 「**ping 10.10.70.120**」と入力してボードとの接続を確認します。次のような ping 応答が連続するループ内に表示されます。
```
64 bytes from 10.10.70.120: seq=2 ttl=64 time=0.074 ms
```

この応答は、Windows ホスト マシンとターゲット ボード間の接続が確立されていることを意味します。

c. **Ctrl+C** キーを押して、ping 応答の表示を停止させます。

ターゲット ボードでの Linux のブートが終了し、ホスト マシンとターゲット ボード間の接続が完了します。次のテスト ドライブでは、SDK を使用して Linux アプリケーションをデバッグする方法を説明します。

5.2.4  テスト ドライブ : SDK リモート デバッグを使用して Linux アプリケーションをデバッグする

このセクションでは、SDK を使用してデフォルトの Linux `hello world` アプリケーションを作成し、Windows ホスト マシンから Linux アプリケーションをデバッグする手順を実行します。

1. 次を実行して、Windows マシンをホストとしてセットアップします。
   a. **［スタート］ → ［すべてのプログラム］ → ［アクセサリ］**をクリックします。
   b. **[Command Prompt]** を右クリックして **[Run as administrator]** をクリックします。DOS シェルが開きます。
   c. DOS シェルで次を入力して、Windows マシンの IP アドレスを設定します。

      ```
      netsh interface ip set address name="Local Area Connection" static 10.10.70.101
      255.255.255.0 192.168.0.1 1
      ```

2. SDK で **[File] → [New] → [Xilinx C Project]** をクリックします。
3. New Project ウィザードが開きます。ターゲット ソフトウェアで **[Linux]** オプションをオンにします。
4. テンプレート リストで **[Linux Hello World]** を選択し、残りのオプションはデフォルトのままにします。プロセッサは `ps7_cortexa9_0` です。

   プロジェクトの位置、使用するハードウェア プラットフォーム、およびプロセッサがこのダイアログ ボックスに表示されます。

5. **[Finish]** をクリックしてアプリケーションを生成します。

   Hello World アプリケーションがコンパイルされ、.elf ファイルが生成されます。

6. [`linux_hello_world_0`] を右クリックして **[Debug as] → [Debug Configurations]** をクリックします。

   Debug Configuration ウィザードが開きます。

7. Debug Configuration ウィザードで、**[Remote ARM Linux Application]** を右クリックして **[New]** をクリックします。
8. [Connection] ドロップダウン リストで **[New]** をクリックします。

   New Connection ウィザードが開きます。

9. **[SSH Only]** タブをクリックして **[Next]** をクリックします。
10. **[Host Name]** タブでターゲット ボード IP を入力します。

    注記 :「5.2.3 テスト ドライブ : JTAG モードで Linux をブートする」の手順 12a で入力した、**`10.10.70.120`** と類似のターゲット IP を使用します。

11. 接続の名前および説明を、それぞれに対応するタブで設定します。
12. **[Finish]** をクリックして接続を作成します。
13. Debug Configuration ウィザードで、[Remote] の「Absolute File Path for C/C++ Application」の下にある **[Browse]**  をクリックします。Select Remote C/C++ Application File ウィザードが開きます。

14. 次を実行します。
    a. ルート ディレクトリを展開します。ここで Enter Password ウィザードが開きます。
    b. ユーザー ID とパスワード (**root**/**root**) を入力し、**[Save ID]** および **[Save Password]** オプションをオンにします。
    c. **[OK]** をクリックします。

       Windows ホスト マシンとターゲット ボード間の接続がすでに確立されているため、ウィンドウにルート ディレクトリの内容が表示されます。
    d. パス名の「/」で右クリックして新しいディレクトリを作成し、「Apps」と名前を付けます。
    e. [Apps] ディレクトリで、linux_hello_world_0.elf という新しいファイルを作成します。
    f. /Apps/linux_hello_world_0.elf などのアプリケーションへの絶対パスを指定します。
15. **[Apply]** をクリックします。
16. **[Debug]** をクリックします。

    [Debug] パースペクティブが開きます。
17. 33 ページの「テスト ドライブ : SDK を使用してソフトウェアをデバッグする」に従ってデバッグを実行します。

    注記 : Linux アプリケーションの出力は、Linux を実行するために開いたターミナル アプリケーションではなく、SDK コンソールに表示されます。
18. Linux アプリケーションのデバッグ終了後、SDK を閉じます。
19. 次を実行して、Windows マシン IP に戻ります。
    a. Windows で **[スタート] → [すべてのプログラム] → [アクセサリ]** とクリックします。コマンド プロンプトを右クリックして **[Run as administrator]** をクリックします。

       DOS シェルが開きます。
    b. DOS シェルに 「**netsh interface ip set address "Local Area Connection" dhcp**」と入力します。

5.2.5  テスト ドライブ : QSPI フラッシュから Linux をブートする

このテスト ドライブでは次を実行します。

1. 「第 1 段階ブート ローダーの実行ファイルを作成する」
2. 「QSPI フラッシュ用に Linux のブート可能なイメージを作成する」
3. 「JTAG および U-Boot コマンドを使用し、ブート イメージで QSPI フラッシュをプログラムする」
4. 「QSPI フラッシュから Linux をブートする」

第 1 段階ブート ローダーの実行ファイルを作成する

注記 : ダウンロードしたコンパイル済みのイメージで提供される `zynq_fsbl_0.elf` を使用すると、この手順を省略できます。

1. SDK で **[File]** → **[New]** → **[Xilinx C Project]** をクリックします。

   New Project ウィザードが開きます。

2. テンプレート リストで **[Zynq FSBL]** を選択し、残りのオプションはデフォルトのままにします。プロジェクトの位置、使用するハードウェア プラットフォーム、およびプロセッサがこのウィンドウに表示されます。プロセッサは `ps7_cortexa9_0` です。
3. **[Finish]** をクリックして FSBL を生成します。

   Zynq FSBL がコンパイルされ、.elf ファイルが生成されます。

QSPI フラッシュ用に Linux のブート可能なイメージを作成する

1. SDK で **[Xilinx Tools]** → **[Create Boot Image]** をクリックします。

   Create Zynq Boot Image ウィザードが開きます。

2. [FSBL ELF] タブで `zynq_fsbl_0.elf` へのパスを提供します。

   注記 : `zynq_fsbl_0.elf` は、
   `<project dir>/project_1/project_1.sdk/SDK/SDK_Export/zynq_fsbl_0/Debug` にあります。

   「5.1 必要な環境」でダウンロードしたファイルから `zynq_fsbl_0.elf` を使用することもできます。

3. U-Boot イメージを追加します。
4. `zImage.bin` などの Linux カーネル イメージを追加し、**`0x100000`** オフセットに指定します。

重要 : Bootgen コマンドには既知の問題があります。このコマンドはファイル拡張子なしのファイルを受け入れません。この問題を回避するには、ダウンロードした `zImage` ファイルを `zImage.bin` に変更してください。

5. デバイス ツリー イメージ (`devicetree.dtb`) を追加し、**`0x600000`** オフセットに指定します。

6. ルート ファイル システム イメージ (`ramdisk8M.image.gz`) を追加し、 **`0x800000`** オフセットに指定します。

   提供したオフセットは U-Boot で定義済みのものです。U-Boot は、QSPI フラッシュからのブート時にこれらのアドレスを予測します。オフセットを変更する場合は、U-Boot を変更して再構築する必要があります。

7. **[Output Folder]** タブで出力フォルダー名を指定します。

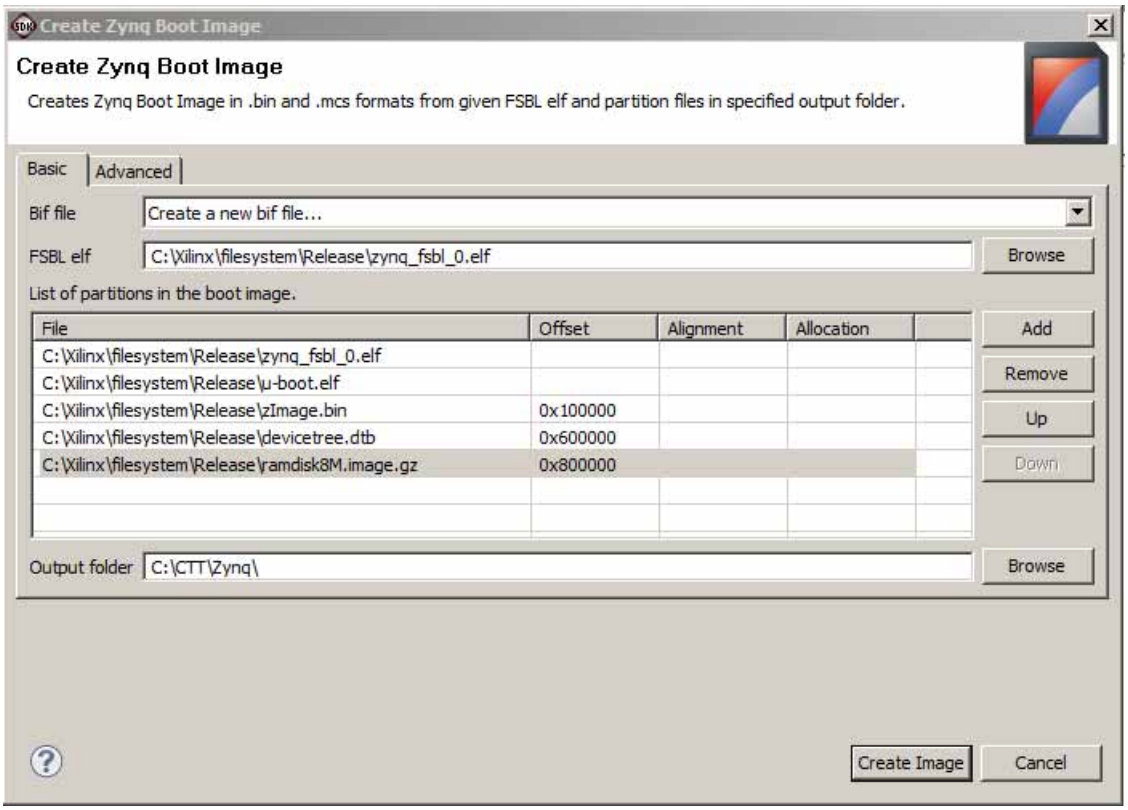

図 5-3 : Zynq **ブート イメージの作成**

8. **[Create Image]** をクリックします。

   Create Zynq Boot Image ウィンドウで、指定した出力フォルダーに次のファイルが作成されます。

   - bootimage.bif
   - u-boot.bin
   - u-boot.mcs

JTAG および U-Boot コマンドを使用し、ブート イメージで QSPI フラッシュをプログラムする

JTAG および U-Boot コマンドを使用し、ブート イメージで QSPI フラッシュをプログラムできます。

1. ZC702 ボードに電源を投入します。
2. シリアル ターミナルが開かない場合は、これを 115200 に設定されているボー レートに接続します。
3. **[Xilinx Tools]** → **[XMD Console]** とクリックして XMD ツールを開きます。
4. XMD プロンプトで次を実行します。
   a. 「**connect arm hw**」と入力して PS の CPU に接続します。
   b. 「**source <Project Dir>/project_1/project_1.sdk/SDK_Export/system_hw_platform/ps7_init.tcl**」と入力した後、「**ps7_init**」と入力して PS を初期化します。
   c. 「**dow directory/u-boot.elf**」と入力して Linux U-Boot を QSPI フラッシュにダウンロードします。
   d. 「**dow -data qspi_boot.bin 0x08000000**」と入力し、Linux のブート可能なイメージを 0x08000000 のターゲット メモリにダウンロードします。

      注記 : 2 進数の実行可能ファイルを DDR メモリにダウンロードするだけです。この 2 進数の実行可能ファイルは、DDR メモリのどのアドレスにもダウンロードできますが、0x04000000 にロードされる U-Boot の実行可能ファイルは変更しないでください。このファイルは、`qspi_boot.bin` データ ファイルのロード後に実行します。
   e. 「**con**」と入力して、U-Boot の実行を開始します。

   シリアル ターミナルに、次のような自動ブート カウントダウン メッセージが表示されます。

   「Hit any key to stop autoboot:3」
5. **[Enter]** をクリックします。

   U-Boot からの自動ブートが停止し、シリアル ターミナルに U-Boot コマンド プロンプトが表示されます。
6. 次の手順に従って、U-Boot をブート可能なイメージでプログラムします。
   a. プロンプトに「**sf probe 0 0 0**」と入力して QSPI フラッシュを選択します。
   b. 「**sf erase 0 0x01000000**」と入力してフラッシュ データを消去します。

      このコマンドにより、16MB のオンボード QSPI フラッシュ メモリが完全に消去されます。
   c. 「**sf write 0x08000000 0 0xFFFFFF**」と入力して QSPI フラッシュにブート イメージを書き込みます。

      ブート可能なイメージがすでに DDR の 0x08000000 に書き込まれていることを確認します。このコマンドにより、ブート可能なイメージと同じサイズのデータが、DDR から `0x0` の QSPI にコピーされました。

      16MB のフラッシュ メモリが対象のこの例では、16MB のデータをコピーしたことになります。ブート可能なイメージのサイズは引数を変更して調整できます。
7. ボードへの電源を切断します。

QSPI フラッシュから Linux をブートする

1. QSPI フラッシュのプログラム後、ボードのジャンパーを 図 5-4 のように設定します。ジャンパー (J20、J21、J22、J25、J26、J27、J28) の設定が必要です。

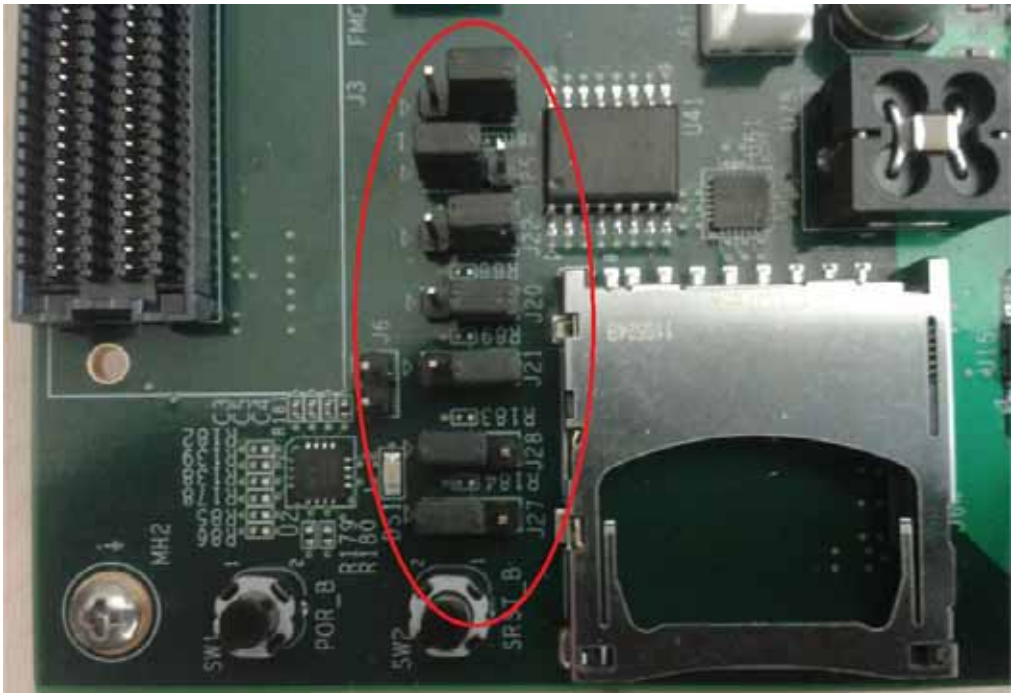

図 5-4 : QSPI フラッシュから Linux をブートする際に必要なジャンパーの設定

2. シリアル ターミナルを 115200 に設定されているボー レートへ接続します。
3. ボード電源をオンにします。

   Linux ブート メッセージがシリアル ターミナルに表示されます。ブート完了後、Zynq> プロンプトが表示されます。

4. 41 ページの「テスト ドライブ : JTAG モードで Linux をブートする」の説明に従ってボード IP アドレスを設定し、コネクティビティを確認します。

Linux アプリケーションの作成およびデバッグは、44 ページの 「テスト ドライブ : SDK リモート デバッグを使用して Linux アプリケーションをデバッグする」を参照してください。

5.2.6 テスト ドライブ : SD カードから Linux をブートする

1. ジャンパー (J20、J21、J22、J25、J26、J27、および J28) を図 5-5 のように設定します。

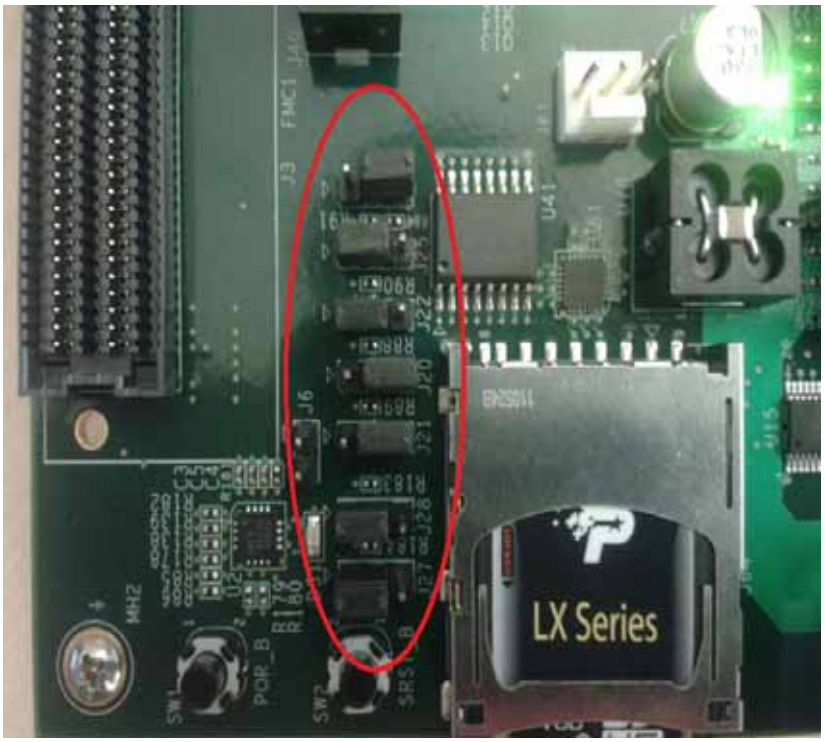

図 5-5 : SD カードから Linux をブートする際のジャンパーの設定

2. 41 ページの 「テスト ドライブ : JTAG モードで Linux をブートする」 の説明に従ってボードを設定します。
3. 46 ページの 「第 1 段階ブート ローダーの実行ファイルを作成する」 の説明に従ってデザインの FSBL を作成します。

   注記 : デフォルトの FSBL イメージを変更する必要がない場合、このガイドの .zip ファイルに含まれる `zynq_fsbl_.elf` ファイルをダウンロードして使用できます。
4. SDK で **[Xilinx Tools] → [Create Boot Image]** をクリックし、Create Zynq Boot Image ウィザードを開きます。

ヒント : ダウンロードした `zynq_fsbl_0.elf` と `u-boot.bin` ファイルに変更がない場合は、ダウンロード済みの `BOOT.bin` ファイルをブート可能なイメージとして使用でき、手順 5、6、および 7 を省略できます。

5. `zynq_fsbl_0.elf` および `u-boot.elf` を追加します。

6. 出力フォルダー パスを **[Output folder]** に指定します。

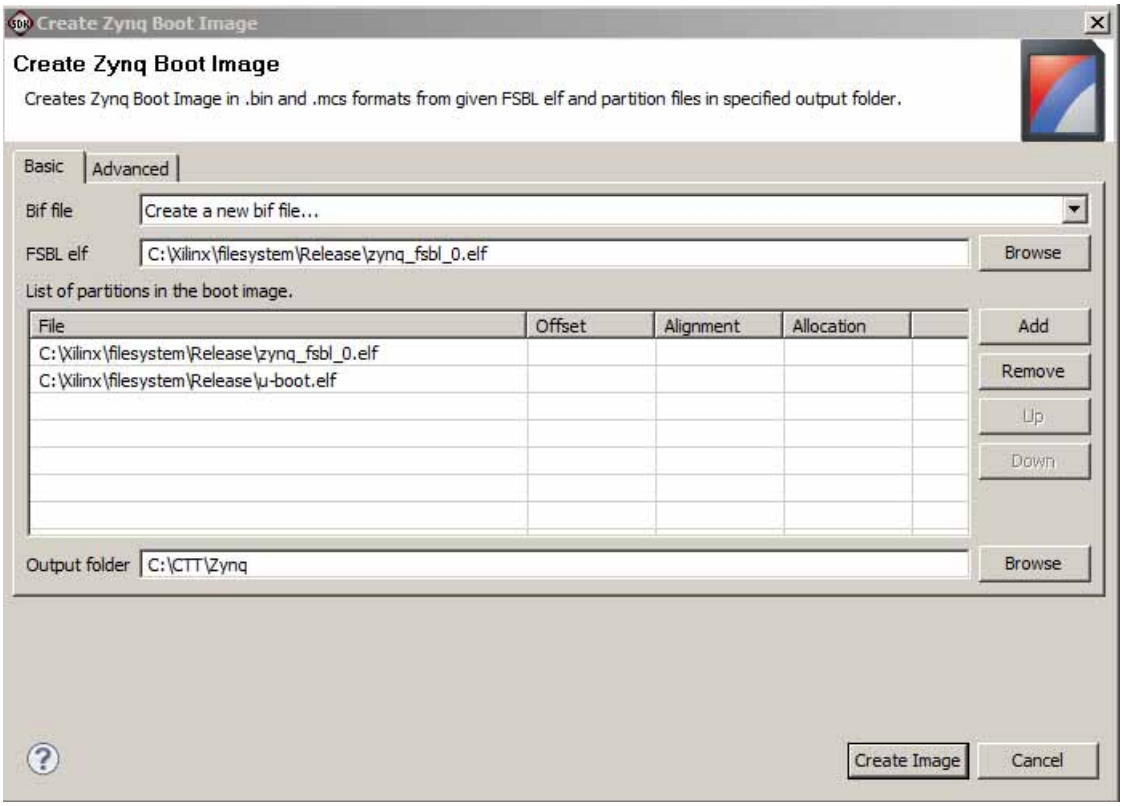

図 5-6 : Zynq **ブート イメージを作成する**

7. **[Create Image]** をクリックします。SDK によって、`u-boot.bin` ファイルが指定のフォルダーに生成されます。
8. `u-boot.bin`、`zImage`、`devicetree.dtb` および `ramdisk8M.image.gz` を SD カードにコピーします。

重要 : ファイル名は変更できません。U-Boot は、システムのブート中に SD カード内でこれらの名前でファイルを検索します。

9. ボード電源を投入し、シリアル ターミナルに表示されるメッセージを確認します。ターゲット ボードで Linux のブートが完了すると、`Zynq>` プロンプトが表示されます。
10. 41 ページの「テスト ドライブ : JTAG モードで Linux をブートする」の説明に従ってボード IP アドレスを設定し、接続を確認します。

Linux アプリケーションの作成およびデバッグは、44 ページの 「テスト ドライブ : SDK リモート デバッグを使用して Linux アプリケーションをデバッグする」 を参照してください。

XILINX

第 6 章

プロセッシング システムの高性能スレーブ ポートを使用したシステム デザイン

この章では、プロセッシング システムの高性能 (HP) 64 ビット スレーブ ポートを用いて AXI CDMA IP をファブリックでインスタンシエートします。このシステムでは、AXI CDMA はマスター デバイスとして機能し、多くのデータを転送元バッファーから DDR システム メモリの転送先バッファーへコピーします。AXI CDMA は、プロセッシング システムの HP スレーブ ポート経由で DDR システム メモリに対する読み出し/書き込みアクセスを実行します。

`mmap()` を用いて、スタンドアロンのアプリケーション ソフトウェアおよび Linux OS ベースのアプリケーション ソフトウェアを記述し、AXI CDMA ブロックを使用するデータ転送を実現します。また、スタンドアロンおよび Linux ベースの両アプリケーション ソフトウェアを ZC702 ボードで個別に実行します。

6.1 AXI CDMA を Zynq PS の HP スレーブ ポートと統合する

ザイリンクスの Zynq™-7000 EPP デバイスは、4 つの高性能 (HP) AXI スレーブ インターフェイスを備えています。これらを使用し、広帯域幅データパスを用いるプログラマブル ロジック (PL) のバス マスターをダブル データ レート (DDR) のオンチップ メモリに接続します。

4 つの高性能 (HP) スレーブ AXI インターフェイスを用いてプログラマブル ロジック (PL) をプロセッシング システム (PS) の非同期 FIFO インターフェイス (AFI) ブロックに接続します。このようなモジュールを使用する目的は、PL の AXI マスターと PS のメモリ システム (DDR およびオンチップ メモリ) 間で高スループットのデータ パスを実現することです。HP スレーブ ポートは、64 ビットまたは 32 ビットのインターフェイスにコンフィギュレーション可能です。

ここでは、AXI CDMA IP をファブリックのマスターとして使用してデザインを作成し、これを PS の HP 64 ビット スレーブ ポートに統合します。図 6-1 に、システムのブロック図を示します。

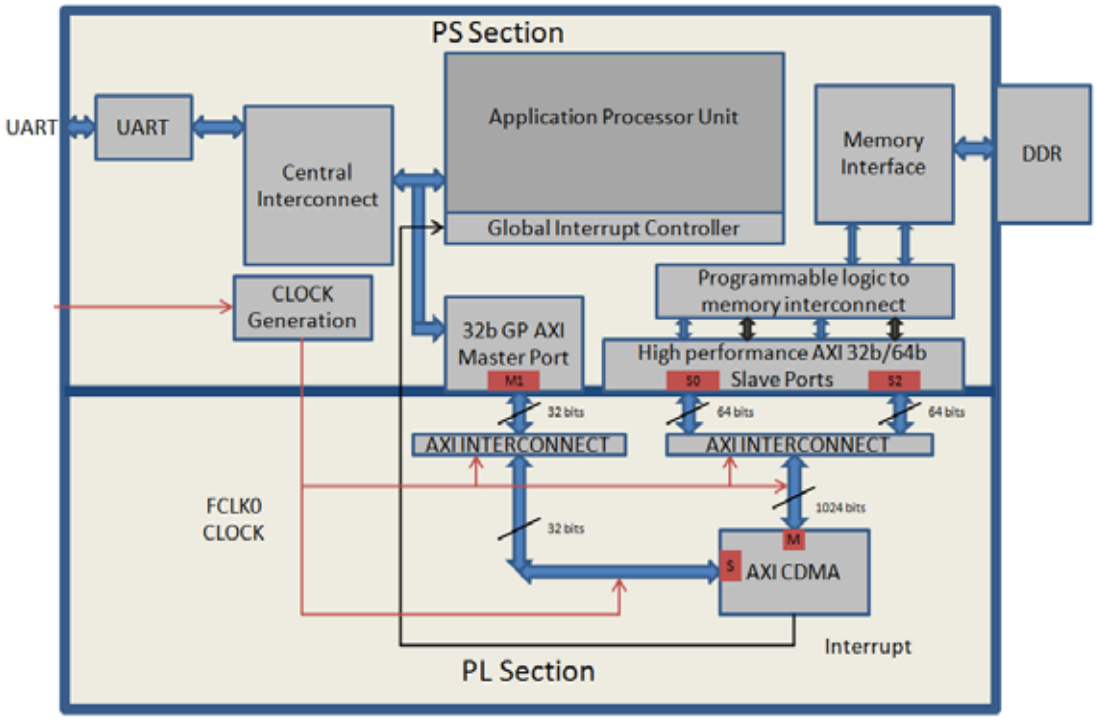

図 6-1 : ブロック図

このシステムは、次のように接続されています。

1. AXI CDMA スレーブ ポートは、PS の汎用マスター ポート 1 (M_AXI_GP1) に接続されます。PS CPU はこれを利用して、データ転送用の AXI CDMA レジスタ セットをコンフィギュレーションし、そのステータスを確認します。
2. AXI CDMA マスター ポートは、PS の高性能 (HP) スレーブ ポート 0 (S_AXI_HP0) に接続されます。AXI CDMA はこれを利用して、DDR システム メモリからデータを読み出します。これは、データ転送中は CDMA の転送元バッファー位置となります。
3. AXI CDMA マスター ポートは PS の高性能 (HP) スレーブ ポート 2 (S_AXI_HP2) に接続されます。AXI CDMA はこれを利用して、DDR システム メモリにデータを書き込みます。これは、データ転送中は CDMA の転送先バッファー位置となります。
4. AXI CDMA の割り込みは、ファブリックから PS の割り込みコントローラーへ接続されます。データの転送後、またはデータ トランザクション中のエラー発生時に、AXI CDMA の割り込みが開始します。

このシステムでは、HP のスレーブ ポート 0 を 0x20000000 ～ 0x2FFFFFFF 範囲の DDR メモリ位置にアクセスするようにコンフィギュレーションします。DDR システムのこのメモリ位置は、CDMA のデータ読み出し時における転送元バッファー位置となります。

さらにユーザーは HP スレーブ ポート 2 を 0x30000000 ～ 0x3FFFFFFF 範囲の DDR メモリ位置にアクセスするようにコンフィギュレーションします。DDR システムのこのメモリ位置は、CDMA のデータ書き込み時における転送先の位置となります。

データ転送チャネルの AXI CDMA IP データ幅を、最大バースト長を 256 とした 1024 ビットにコンフィギュレーションします。この設定により、CDMA の最大転送サイズは、各トランザクションにつき 1024x256 ビットとなります。

上記のシステム向けにアプリケーション ソフトウェア コードを記述します。このコードを実行すると、指定したデータ パターンでソース バッファー メモリが初期化され、転送先バッファー メモリはすべてゼロが書き込まれてクリアされます。次に、DMA 転送向けに CDMA レジスタのコンフィギュレーションを開始します。転送元バッファー位置、転送先バッファー位置、および CDMA レジスタへ転送されるバイトの数を書き込んで CDMA 割り込みを待機します。割り込みが発生すると、DMA 転送のステータスを確認します。

データ転送のステータスが正常の場合、転送元バッファー データと転送先バッファー データを比較し、シリアル ターミナルにその結果を表示します。

データ転送のステータスがエラーの場合、シリアル ターミナルにエラー ステータスを表示し、実行を停止します。

6.1.1 テスト ドライブ : AXI CDMA を PS の HP スレーブ ポートと統合する

1. 次のいずれかから開始します。
   - 24 ページの「テスト ドライブ : ファブリックでインスタンシエートされる IP の機能を確認する」で作成したシステムを使用する
   - 11 ページの「テスト ドライブ : Zynq プロセッシング システムの新規エンベデッド プロジェクトを作成する」 の説明に従って新しいプロジェクトを作成する
2. PlanAhead ツールの [Sources] ペインで **[system_i-system(system.xmp)]** をダブルクリックし、XPS を起動します。
3. [XPS System Assembly] ビューで **[Bus Interfaces]** タブをクリックします。
4. IP カタログで **[DMA and Timer]** を展開し、**[AXI Central DMA]** をダブルクリックして追加します。

   IP インスタンス `axi_cdma 3.03.a` をデザインに追加するかどうかを確認するメッセージが表示されます。
5. **[Yes]** をクリックします。

   CDMA のコンフィギュレーション ウィンドウが開きます。
6. [CDMA Configuration] ウィンドウで [User] タブをクリックします。データ転送チャネル オプションのデータ幅を **`1024`** に、オプションを使用するための最大バースト長を **`256`** に変更します。
7. **[OK]** をクリックします。

   「axi_cdma IP with version number 3.03.a is instantiated with name axi_cdma_0」と表示されたメッセージ ウィンドウが開きます。XPS によって、接続先のプロセッサを決定するように求められます。
8. **[User will make necessary connections and settings]** を選択します。

   注記 : この設定では、AXI CDMA を手動で接続することが必要になります。

9. **[OK]** をクリックします。

   未接続の IP axi_cdma_0 がデザインでインスタンシエートされます。クロスチェックするために、[XPS System Assembly] ビューの **[Bus Interfaces]** タブをクリックします。リストで未接続の axi_cdma_0 が検出されます。

10. IP カタログで **[Bus and Bridge]** を展開し、**[AXI Interconnect IP]** をダブルクリックして追加します。

    axi_interconnect 1.06.a IP インスタンスをデザインに追加するかどうかを確認するダイアログ ボックスが開きます。

11. **[Yes]** をクリックします。

    AXI インターコネクトのコンフィギュレーション ウィンドウが開きます。コンポーネントのインスタンス名として「**axi_interconnect_gp1**」と入力します。

12. **[OK]** をクリックします。

    XPS によって未接続の AXI インターコネクト IP、axi_interconnect_gp1 がシステムへ追加されます。クロスチェックするために、[XPS System Assemblyify] ビューの **[Bus Interface]** タブをクリックします。

    注記 : この章の後半では、このインターコネクトを用いて AXI CDMA スレーブ ポートと PS の GP マスター ポート 1 を接続します。このインターフェイスを利用して、PS CPU から設定された CDMA レジスタをコンフィギュレーションします。

13. 先述の 3 つの手順を繰り返し、さらに axi_interconnect_hp というインスタンス名の AXI インターコネクト IP をもう 1 つ追加します。

    注記 : この章の後半では、このインターコネクトを用いて AXI CDMA マスター ポートおよび PS HP スレーブ ポート 0 とポート 2 を接続します。そして、このインターフェイスを使用してデータを転送元から転送先 DDR メモリ位置へコピーします。

14. [XPS System Assembly] ビューで **[Zynq]** タブをクリックし、Zynq プロセッシング システムのブロック図を開きます。

15. 緑色の **[32b GP AXI Master Port]** ブロックをクリックして processing_system7_0 Configuration ウィザードを開きます。

16. [User] タブで **[General Purpose Master AXI Interface]** を展開し、**[Enable M_AXI_GP1 Interface]** オプションをオンにします。

17. **[OK]** をクリックし、PS とファブリック間の GP マスター ポート 1 インターフェイスを有効にします。

18. Zynq プロセッシング システムのブロック図で、緑色の **[high performance AXI 32/64b Slave Ports]** ブロックをクリックして processing_system7_0 Configuration ウィザードを開きます。

19. [User] タブで **[High Performance Slave AXI Interface]** を展開し、次のように設定します。

    a. **[Enable S_AXI_HP0 interface]** チェック ボックスをオンにして PS とファブリック間の HP スレーブ ポート 0 インターフェイスを有効にします。

    b. HP0 ベース アドレスを 0x20000000 に、HP0 上位アドレスを 0x2FFFFFFF に変更します。

       これにより、HP スレーブ ポート 0 が 0x20000000 ～ 0x2FFFFFFF の DDR システム メモリへアクセスするように設定されます。これは、CDMA がデータを読み出す転送元バッファー位置となります。

c. **[Enable S_AXI_HP2 interface]** チェック ボックスをオンにして、PS とファブリック間の HP スレーブ ポート 2 インターフェイスを有効にします。

d. HP2 ベース アドレスを 0x30000000 に、HP2 上位アドレスを 0x3FFFFFFF に変更します。

これにより、HP スレーブ ポート 2 が 0x30000000 ～ 0x3FFFFFFF の DDR システム メモリへアクセスするように設定されます。これは、CDMA がデータを書き込む転送先の位置となります。

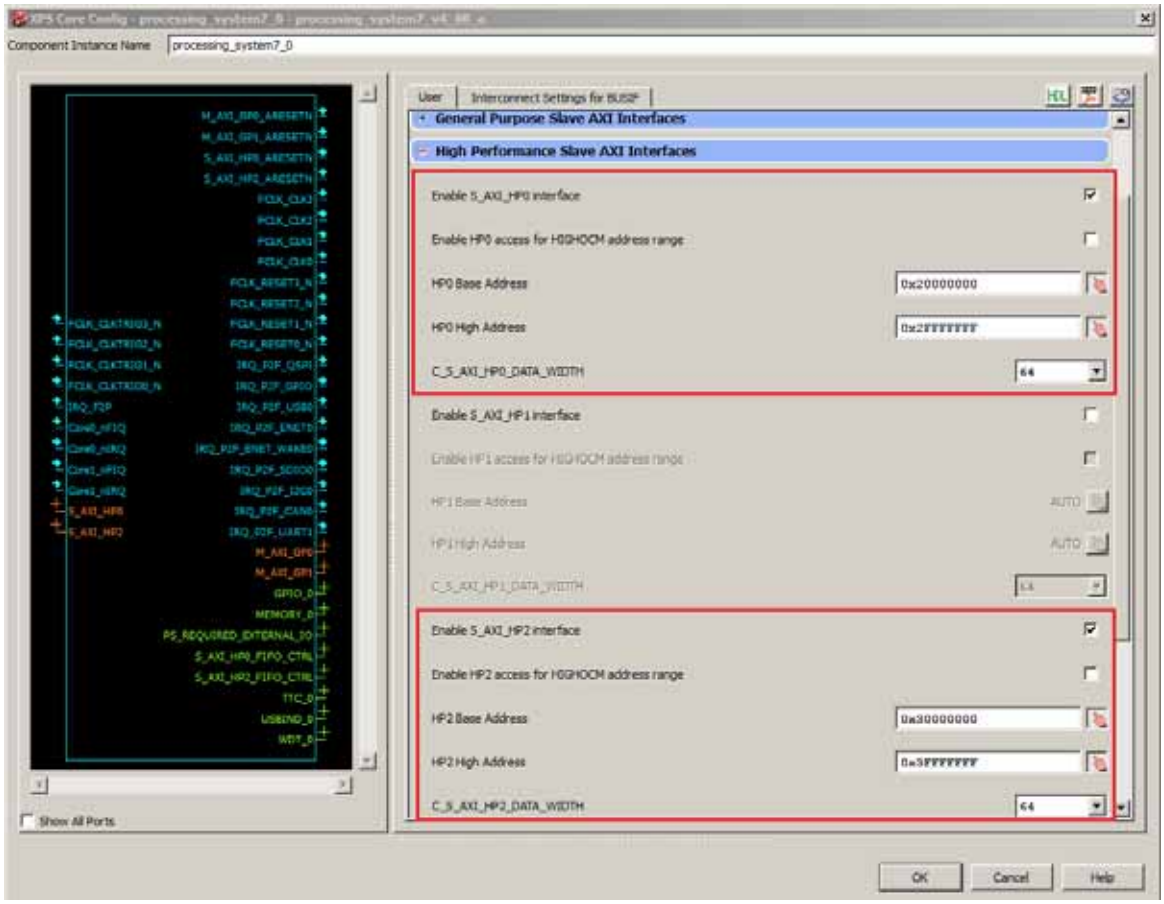

図 6-2 : [XPS Core Configuration] ウィンドウの設定

注記 : デフォルトでは、HP スレーブ ポートは 64 ビット インターフェイスにコンフィギュレーションされています。

e. **[OK]** をクリックします。

20. **[Bus Interfaces]** タブをクリックして、次のように接続します。

a. [processing_system7_0] の下で **[M_AXI_GP1]** をクリックし、[Bus Name] 列で **[No Connection]** をクリックします。表示されるドロップダウン リストで [M_AXI_GP1] を [axi_interconnect_gp1] に接続します。

b. [processing_system7_0] を非展開にします。

c. [axi_cdma_0] の下で **[S_AXI_LITE]** をクリックし、[Bus Name] 列で **[No Connection]** をクリックして [Connection] ダイアログ ボックスを開きます。

d. [Select AXI Interconnect] リストで [axi_interconnect_gp1] をクリックします。

[processing_system7_0.M_AXI_GP1] が [Select Master(s)] リストに表示されていることを確認します。[processing_system7_0.M_AXI_GP1] チェック ボックスをオンにします。

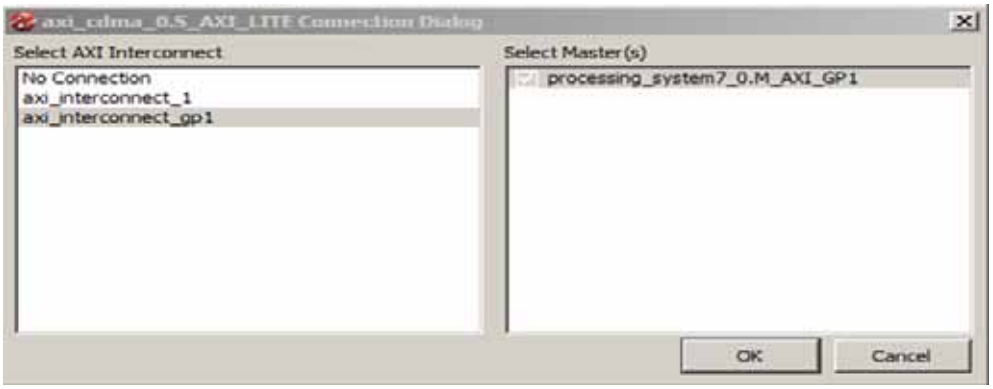

図 6-3 : [AXI Connection] **ダイアログ ボックス**

e. **[OK]** をクリックします。

XPS によって、axi_cdma_0 スレーブ ポート インターフェイスは axi_interconnect_gp1 を介して processing_system7_0 GP マスター ポート 1 インターフェイスへ接続されます。

f. [axi_cdma_0] で **[M_AXI]** を選択し、[Bus Name] 列では **[No Connection]** を選択します。表示されるドロップダウン リストで [M_AXI] を [axi_interconnect_hp] に接続します。

g. [processing_system7_0] を展開して **[S_AXI_HP0]** をクリックします。[Bus Name] 列で **[No Connection]** をクリックし、[Connection] ダイアログ ボックスを開きます。

h. **[Select AXI Interconnect]** リストで [axi_interconnect_hp] をクリックします。

[axi_cdma_0.M_AXI] が [Select Master(s)] リストに表示されていることを確認します。[axi_cdma_0.M_AXI] チェック ボックスをオンにします。

i. **[OK]** をクリックします。

XPS によって、axi_cdma_0 マスター ポートは axi_interconnect_hp を介して processing_system7_0 HP スレーブ ポート 0 へ接続されます。

j. 先述の 3 つの手順を繰り返し、axi_cdma_0 マスター ポートを axi_interconnect_hp を介して processing_system7_0 HP スレーブ ポート 2 へ接続します。

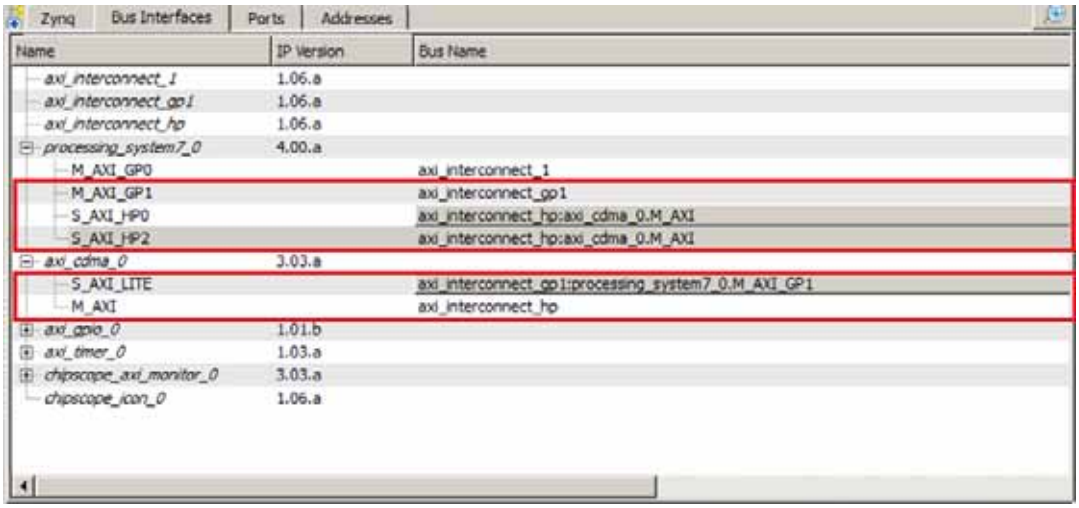

図 6-4 : [Ports] **タブ**

21. IP およびそれらのポートがリストされている [Ports] タブをクリックします。[axi_interconnect_gp1]、[axi_interconnect_hp]、および [axi_cdma_0] を展開します。
22. 次の IP の接続を確認します。これらの中で接続されていないものがあれば、すぐに接続します。

表 6-1 : IP 接続

IP	ポート	接続
axi_interconnect_gp1	INTERCONNECT_ACLK	processing_system7_0 :FCLK_CLK0
	INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N
axi_interconnect_hp	INTERCONNECT_ACLK	processing_system7_0 :FCLK_CLK0
	INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N
axi_cdma_0	(BUS_IF) S_AXI_LITE::s_axi_lite_aclk	processing_system7_0 :FCLK_CLK0
	(BUS_IF) M_AXI:: m_axi_aclk	processing_system7_0 :FCLK_CLK0

[Ports] タブの内容が図 6-5 のようになるようにします。

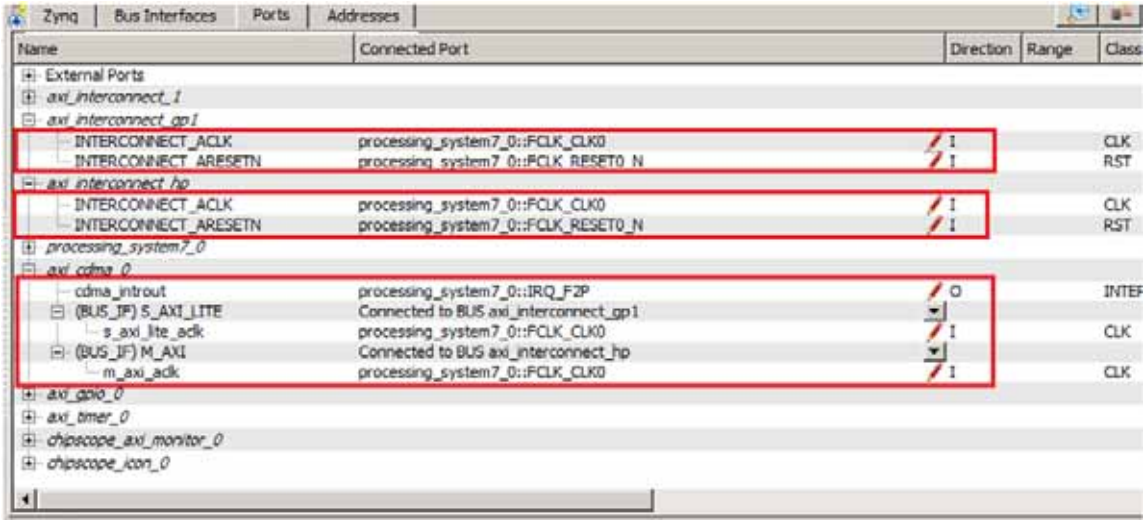

図 6-5 : 完成したポート接続

23. すべての IP を非展開にし、[processing_system7_0] を展開します。次のようにポートが接続されていない場合は、ここで接続します。図 6-6 のように接続します。

表 6-2 : Processing_system7_0 ポートの接続

IP	ポート	接続
Processing_system7_0	(BUS_IF) M_AXI_GP1:: M_AXI_GP1_ACLK	processing_system7_0 :FCLK_CLK0
	(BUS_IF) S_AXI_HP0:: S_AXI_HP0_ACLK	processing_system7_0 ::FCLK_CLK0
	(BUS_IF) S_AXI_HP2:: S_AXI_HP2_ACLK	processing_system7_0 ::FCLK_CLK0

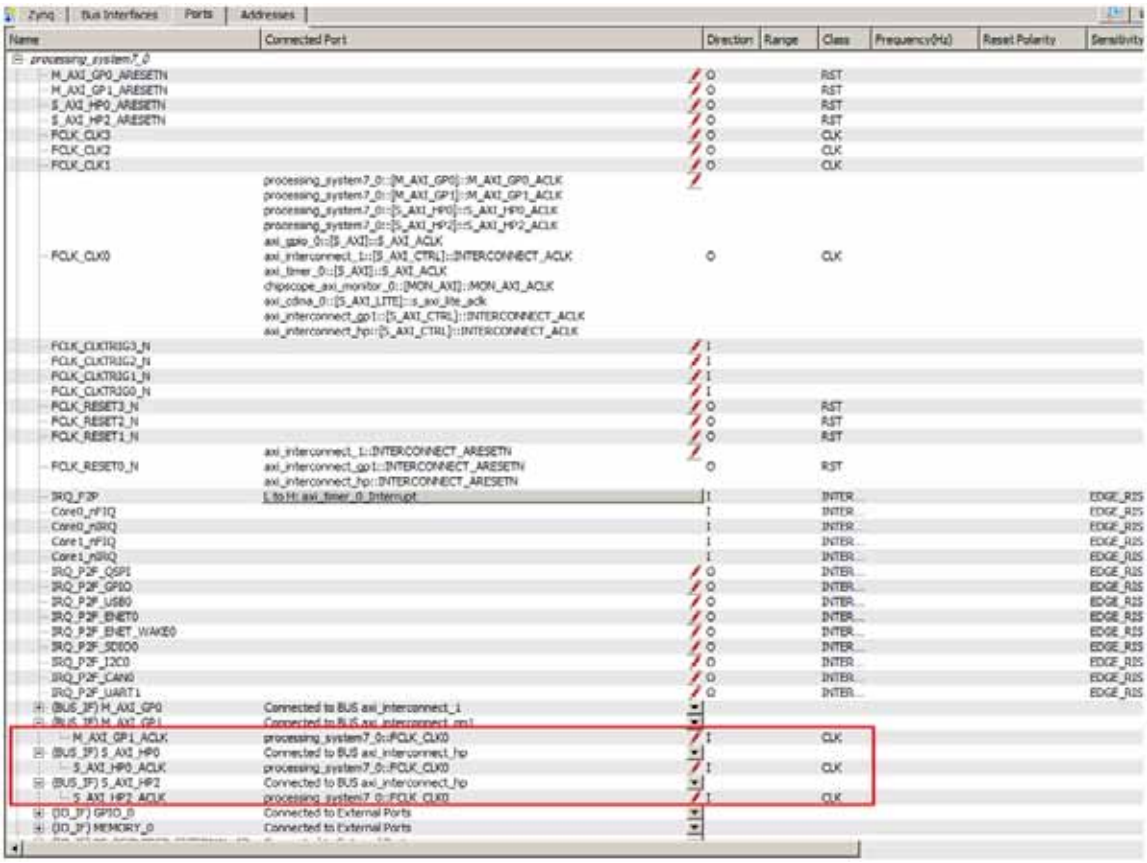

図 6-6 : Processing_system7_0 **の接続**

24. 次の手順に従って、ファブリック側の CDMA 割り込みを PS 側の割り込みコントローラーへ接続します。
    a. `[Processing_System7_0]` の [Connected Port] 列で、**[1L to H:No Connection]** をクリックします。

       注記 : 第 3 章で作成したシステムを変更する場合、[Connected Port] タブの **[L to H : axi_timer_0_interrupt]** を選択します。

       [Interrupt Connection] ダイアログ ボックスが開きます。

    b. [Unconnected Interrupts] リストで **[axi_cdma_0]** を選択し、矢印をクリックして選択したアイテムを [Connected Interrupts] リストへ移動させます。
    c. **[OK]** をクリックします。

       XPS はファブリック側の CDMA 割り込みを PS の割り込みコントローラーへ接続します。

25. **[Addresses]** タブをクリックし、**[Generate Addresses]**  をクリックしてマップされていないデバイスのアドレスを生成します。

    `[axi_cdma_0]` が `[processing_system7_0]` アドレス範囲内のアドレスに割り当てられていることを確認します。

26. [processing_system7_0] のアドレス マップで、`axi_cdma_0` ベース アドレスを `0x80200000` に、上位アドレスを `0x8020FFFF` に変更します。

    これにより、`axi_cdma_0` のアドレス範囲が確実に `processing_system7_0` アドレス空間内の `0x80200000` ～ `0x8020FFFF` に固定されます。

27. デザイン ルール チェックの実行コンソールにエラーがないことを確認します。

28. XPS を閉じます。PlanAhead™ デザイン ツール ウィンドウが再度アクティブになります。
29. [Design Sources] でエンベデッド ソースをクリックしてから右クリックし、**[Create Top HDL]** をクリックします。PlanAhead ツールによって `system_stub.v` ファイルが生成されます。
30. 変更したファイルをすべて保存します。
31. Flow Navigator の [Program and Debug] リストで **[Generate Bitstream]** をクリックします。PlanAhead が生成するすべての警告メッセージを無視します。
32. ビットストリームの生成が完了したら、第 2 章の説明に従ってハードウェアをエクスポートして SDK を起動します。このデザインでは、PL ファブリック用に生成されたビットストリームがあるため、これも SDK へエクスポートされます。

6.2 デザイン用のスタンドアロン アプリケーション ソフトウェア

この章で設計した CDMA ベースのシステムをボードで実行するには、アプリケーション ソフトウェアが必要です。ここでは、CDMA ベースのスタンドアロン アプリケーション ソフトウェアについて詳細に説明します。

アプリケーション ソフトウェアの `main()` 関数は、実行のエントリ ポイントです。これは、指定したテスト パターンで転送元メモリ バッファーを初期化し、転送先メモリ バッファーにすべてゼロを書き込むことでこれをクリアします。

次に、アプリケーション ソフトウェアは、転送元バッファーおよび転送先バッファーの開始位置を提供して CDMA レジスタ セットをコンフィギュレーションします。DMA 転送を開始するために、CDMA レジスタで転送されるバイト数を書き込み、CDMA の割り込みが発生するのを待機します。割り込み後、DMA 転送のステータスを確認し、転送元バッファーを転送先バッファーと比較します。最後に、シリアル ターミナルにその比較結果を出力し、実行を停止します。

6.2.1 アプリケーション ソフトウェアのフロー

アプリケーション ソフトウェアは次を実行します。

1. 転送元バッファーを指定したテスト パターンで初期化します。転送元バッファー位置は `0x20000000` ～ `0x2FFFFFFF` の範囲です。

   転送先バッファーにすべてゼロを書き込むことでこれをクリアします。転送先バッファー位置は、`0x30000000` ～ `0x3FFFFFFF` の範囲です。

2. AXI CDMA IP を初期化して次を実行します。
   a. CDMA コールバック関数を AXI CDMA ISR と関連付け、割り込みを有効にします。

      このコールバック関数は、CDMA の割り込み中に実行されます。これは、DMA 転送のステータスに応じて割り込み Done/Error フラグを設定します。

      アプリケーション ソフトウェアは、コールバック関数がこれらのフラグを設定するまで待機し、そのステータス フラグに応じて実行します。

   b. CDMA をシンプル モードでコンフィギュレーションします。
   c. CDMA IP のステータス レジスタを確認して、CDMA アイドル ステータスを検証します。
   d. 転送元のバッファー開始位置 `0x20000000` を CDMA レジスタに設定します。
   e. 転送先のバッファー開始位置 `0x30000000` を CDMA レジスタに設定します。

f. CDMA レジスタへ転送するバイト数を設定します。アプリケーション ソフトウェアは DMA 転送を開始します。

3. CDMA 割り込みがトリガーされた後、DMA 転送のステータスを確認します。

   転送のステータスが正常の場合、アプリケーション ソフトウェアは転送元バッファー位置と転送先バッファー位置を比較し、シリアル ターミナルにその結果を表示して実行を終了します。

   転送のステータスがエラーの場合、ソフトウェアはシリアル ターミナルにエラー ステータスを出力し、実行を停止します。

6.2.2  テスト ドライブ : SDK を使用してスタンドアロンの CDMA アプリケーションを実行する

1. SDK で **[File]** → **[New]** → **[Xilinx C Project]** をクリックします。
2. テンプレート リストで **[Empty Application]** を選択します。
3. **[Project Name]** に 「`cdma_app`」 と入力し、残りのオプションはデフォルトのままにします。

   プロジェクトの位置、使用するハードウェア プラットフォーム、およびプロセッサがこのウィンドウに表示されます。ここからのプロセッサは `ps7_cortexa9_0` です。

4. **[Next]** をクリックして次のページを開きます。このページではプロジェクト用の BSP を作成します。
5. **[Project Name]** に 「`cdma_app_bsp`」 と入力します。
6. **[Finish]** をクリックして BSP を生成します。
7. [Project Explorer] タブで `[cdma_app]` プロジェクトを展開して `[src]` ディレクトリを右クリックし、**[Import]** を選択して [Import] ダイアログ ボックスを開きます。
8. [Import] ダイアログ ボックスで **[General]** を展開し、**[File System]** をクリックします。
9. **[Next]** をクリックします。
10. `cdma_app.c` ファイルを追加して **[Finish]** をクリックします。

    SDK は自動的にアプリケーションを構築し、コンソール ウィンドウにステータスを表示します。

    注記 : システムのアプリケーション ソフトウェアのファイル名は `cdma_app.c` です。このファイルは、このガイドで提供されている `ug873_design_files.zip` から入手できます。この ZIP ファイルへのリンクは、付録 A 「その他のリソース」に記載されています。

11. ボー レートが `115200` に設定されているシリアル通信ユーティリティを開きます。
12. ハードウェア ボードがセットアップされ、オンになっていることを確認します。

    注記 : ボードのセット アップは、「2.1.3 テスト ドライブ : Hello World アプリケーションを実行する」を参照してください。

13. SDK で **[Xilinx Tools]** → **[Program FPGA]** とクリックし、[Program FPGA] ダイアログ ボックスを開きます。ダイアログ ボックスにはビットストリーム パスが表示されます。
14. **[Program]** をクリックし、ビットストリームをダウンロードして PL ファブリックをプログラムします。
15. 「2.1.3 テスト ドライブ : Hello World アプリケーションを実行する」の手順と同じようにプロジェクトを実行します。
16. シリアル ターミナルで CDMA 転送のステータスを確認します。転送が正常の場合、「DMA Transfer is Successful」というメッセージが表示されます。エラーの場合、シリアル ターミナルにエラー メッセージが表示されます。

6.3 CDMA システム向けの Linux OS ベースのアプリケーション ソフトウェア

このセクションでは、Linux で提供される mmap() システム コールを使用して CDMA 向けの Linux ベース アプリケーション ソフトウェアを作成し、これをハードウェアで実行して CDMA IP の機能を確認します。

メモリのマップ中に提供される属性に応じて読み出しまたは書き込みができるように、mmap() システム コールを使用して指定した Kernel メモリ領域をユーザー層に割り当てます。mmap() システム コールについての詳細は、このガイドでは割愛します。

注意：mmap() コールは、Linux アプリケーションの書き込みに推奨する方法ではありません。Kernel の一部の制限領域または共有リソースへ間違ってアクセスすると、Kernel がクラッシュする可能性があります。

アプリケーション ソフトウェアの main() 関数は、実行のエントリ ポイントです。これは、指定したテスト パターンで転送元アレイを初期化し、転送先アレイをクリアします。その後、転送元アレイの内容を 0x20000000 から開始する DDR メモリへコピーし、転送先への DMA 転送を開始するように DMA レジスタを設定します。DMA 転送後、アプリケーションは転送のステータスを読み出し、その結果をシリアル ターミナルに表示します。

6.3.1 アプリケーション ソフトウェアの構成手順

アプリケーション ソフトウェアは次の手順で構成されます。

1. 0xA5A5A5A5 値を持つユーザー層にある転送元アレイ全体を初期化します。

   ユーザー層にある転送先バッファー全体にすべてゼロを書き込むことでこれをクリアします。

2. 0x20000000 から開始する Kernel のメモリ位置を、mmap() システム コールを使用して書き込み許可を持つユーザー層にマップします。

   このようにマッピングすることで、指定した Kernel メモリへの書き込みが可能になります。

3. 転送元アレイの内容をマップ済みの Kernel メモリへコピーします。
4. ユーザー層から Kernel メモリのマップを解除します。

5. AXI CDMA レジスタのメモリ位置を、mmap() システム コールを使用して読み出し許可および書き込み許可を持つユーザー層にマップします。ユーザー層から CDMA レジスタを次のように設定します。
   a. DMA をリセットして前の通信を停止します。
   b. 割り込みを有効にして、DMA 転送のステータスを取得します。
   c. CDMA をシンプル モードで設定します。
   d. CDMA がアイドル状態であることを確認します。
   e. 転送元バッファーの開始位置 0x20000000 を CDMA レジスタに設定します。
   f. 転送先バッファーの開始位置 0x30000000 を CDMA レジスタに設定します。
   g. CDMA レジスタで転送されるバイト数を設定します。このレジスタへの書き込みにより DMA 転送を開始します。
6. DMA 転送のステータスを、転送が終了するまで継続して読み出します。
7. CDMA 転送の終了後、mumap() システム コールを使用するユーザー層からの変更に備え、CDMA レジスタ メモリのマップを解除します。
8. 0x30000000 から開始する Kernel メモリ位置を、読み出し属性と書き込み属性を備えるユーザー層にマップします。
9. 0x30000000 から開始する Kernel メモリの内容をユーザー層の転送先アレイにコピーします。
10. ユーザー層から Kernel メモリのマップを解除します。
11. 転送元アレイを転送先アレイと比較します。
12. シリアル ターミナルに比較結果が表示されます。比較結果が正常の場合、「DATA Transfer is Successful」というメッセージが表示されます。エラーの場合、シリアル ターミナルにエラー メッセージが表示されます。

6.4  テスト ドライブ : SDK を使用して Linux CDMA アプリケーションを実行する

Linux OS ベースのアプリケーションを実行する手順は次のとおりです。

1. 「ターゲット ボートで Linux をブートする」
2. 「SDK を使用して、アプリケーションを構築してそれをターゲット ボードで実行する」

6.4.1 ターゲット ボートで Linux をブートする

第 5 章「SDK を使用した Linux のブートおよびアプリケーションのデバッグ」の説明に従って、Zynq ZC702 ターゲット ボードで Linux をブートします。

ブートが完了すると、zynq> プロンプトがシリアル ターミナルに表示されます。「5.2.3 テスト ドライブ : JTAG モードで Linux をブートする」の手順 12 の説明に従って、ボードの IP アドレスを設定します。

6.4.2 SDK を使用して、アプリケーションを構築してそれをターゲット ボードで実行する

1. 「5.2.4 テスト ドライブ : SDK リモート デバッグを使用して Linux アプリケーションをデバッグする」の手順 1 の説明に従って、Windows マシンをホストとしてセット アップします。
2. SDK で **[File]** → **[New]** → **[Xilinx C Project]** をクリックします。
3. New Project ウィザードが開きます。ターゲット ソフトウェア リストで **[Linux]** オプションをクリックします。

4. テンプレート リストで **[Linux Empty Application]** を選択します。
5. プロジェクト名に 「**linux_cdma_app**」 と入力し、残りのオプションはデフォルトのままにしておきます。プロセッサは ps7_cortexa9_0 です。

   プロジェクトの位置、使用するハードウェア プラットフォーム、およびプロセッサがこのダイアログ ボックスに表示されます。
6. **[Finish]** をクリックしてアプリケーションを生成します。
7. [Project Explorer] タブで [linux_cdma_app] プロジェクトを展開して [src] ディレクトリを右クリックし、**[Import]** を選択して [Import] ダイアログ ボックスを開きます。
8. [Import] ダイアログ ボックスで **[General]** を展開し、**[File System]** をクリックします。
9. **[Next]** をクリックします。
10. linux_cdma_app.c ファイルを追加して **[Finish]** をクリックします。

    SDK は自動的にアプリケーションを構築し、linux_cdma_app.elf ファイルを生成します。コンソール ウィンドウでステータスを確認します。

    注記 : システムのアプリケーション ソフトウェア ファイル名は linux_cdma_app.c です。このファイルは、このガイドで提供されている ug873_design_files.zip ファイルから入手できます。この ZIP ファイルへのリンクは、付録 A 「その他のリソース」に記載されています。
11. [linux_cdma_app] を右クリックし、**[Run As]** → **[Run Configurations]** とクリックして [Debug Configuration] ウィザードを開きます。
12. Run Configuration ウィザードで **[Remote ARM Linux Application]** を右クリックし、**[New]** をクリックします。
13. **[Connection]** ドロップダウン リストで **[New]** をクリックし、New Connection ウィザードを開きます。
14. **[SSH Only]** タブをクリックして **[Next]** をクリックします。
15. [Host Name] タブでターゲット ボード IP を入力します。

    注記 :「5.2.3 テスト ドライブ : JTAG モードで Linux をブートする」の手順 12a で事前入力したターゲット IP を使用します。
16. 接続の名前および説明を、それぞれに対応するタブで設定します。
17. **[Finish]** をクリックして接続を作成します。
18. Run Configuration ウィザードでは、「**Remote Absolute File Path for C/C++ Application**」の下にある **[Browse]**  をクリックします。Select Remote C/C++ Application File ウィザードが開きます。

19. 次を実行します。
    a. ルート ディレクトリを展開します。Enter Password ウィザードが開きます。
    b. ユーザー ID およびパスワード (**root**/**root**) を入力し、**[Save ID]** および **[Save Password]** オプションをオンにします。
    c. **[OK]** をクリックします。

       Windows ホスト マシンとターゲット ボード間の接続がすでに確立されているため、ウィンドウにルート ディレクトリの内容が表示されます。

    d. パス名の「/」で右クリックし、新しいディレクトリを「**Apps**」という名前で作成します。
    e. `/Apps/linux_cdma_app.elf` などのアプリケーションへの絶対パスを提供します。
20. **[Apply]** をクリックします。
21. **[Run]** をクリックしてアプリケーションの実行を開始します。
22. シリアル ターミナルで CDMA 転送のステータスを確認します。転送が正常の場合、「DMA Transfer is Successful」というメッセージが表示されます。エラーの場合、シリアル ターミナルにエラー メッセージが表示されます。
23. Linux アプリケーションのデバッグ終了後、SDK を閉じます。
24. 「5.2.4 テスト ドライブ : SDK リモート デバッグを使用して Linux アプリケーションをデバッグする」の手順 19 を参照して Windows マシンに戻ります。

XILINX

第 7 章

SDK を使用したソフトウェア プロファイリング

この章では、第 6 章で作成したスタンドアロン BSP と AXI CDMA 関連のアプリケーションに対応するプロファイリング機能を有効にします。

7.1 SDK を使用したアプリケーションのプロファイリング

プロファイリングは、各ルーチンにおけるソフトウェアの実行時間を決定する方法です。この情報を用いて、コードの重要な部分、およびデザインでの最適なコード配置を判断できます。頻繁に呼び出されるルーチンは、キャッシュメモリなど高速メモリに配置するのが最適です。プロファイリング情報を用いることで、コードをハードウェアに配置可能かどうかも判断でき、その結果全体的な性能が向上します。

7.1.1  テスト ドライブ : SDK でアプリケーション ソフトウェアをプロファイリングする

1. ザイリンクスの SDK を起動し、第 2 章および第 3 章で使用したワークスペースと同じものを開きます。
2. シリアル ターミナルが開かない場合は、シリアル通信ユーティリティを 115200 に設定されているボー レートに接続します。
3. SDK で **[File]** → **[New]** → **[Xilinx C Project]** をクリックします。
4. テンプレート リストで **[Empty Application]** を選択します。[Project Name] に「**cdma_app_profile**」と入力し、残りのオプションはデフォルトのままにします。プロジェクトの位置、使用するハードウェア プラットフォーム、およびプロセッサがこのウィンドウに表示されます。使用するプロセッサは ps7_cortexa9_0 です。
5. **[Next]** をクリックします。
6. 次のページで [Project Name] に「**cdma_app_profile_bsp**」と入力し、**[Finish]** をクリックして cdma_app アプリケーション用 BSP を生成します。
7. [Project Explorer] タブで [cdma_app] プロジェクトを展開して [src] ディレクトリを右クリックし、**[Import]** をクリックして [Import] ダイアログ ボックスを開きます。
8. [Import] ダイアログ ボックスで **[General]** を展開し、**[File System]** をクリックします。
9. **[Next]** をクリックします。
10. cdma_app.c ファイルを追加して **[Finish]** をクリックします。

    SDK は自動的にアプリケーションを構築し、コンソール ウィンドウにステータスを表示します。

    注記 : システムのアプリケーション ソフトウェアのファイル名は cdma_app.c です。このファイルは、このガイドで提供されている ug873_design_files.zip ファイルから入手できます。この ZIP ファイルへのリンクは、付録 A 「その他のリソース」で提供されています。

11. [Project Explorer] タブで [cdma_app_profile_bsp] を右クリックし、**[Board Support Package Settings]** をクリックして [Board Support Package Settings] ダイアログ ボックスを開きます。
12. [Board Support Package Settings] ダイアログ ボックスで、左側のナビゲーション ペインにある **[standalone]** をクリックします。リストの **[enable_sw_intrusive_profiling]** オプションを選択し、その値を **true** に設定します。

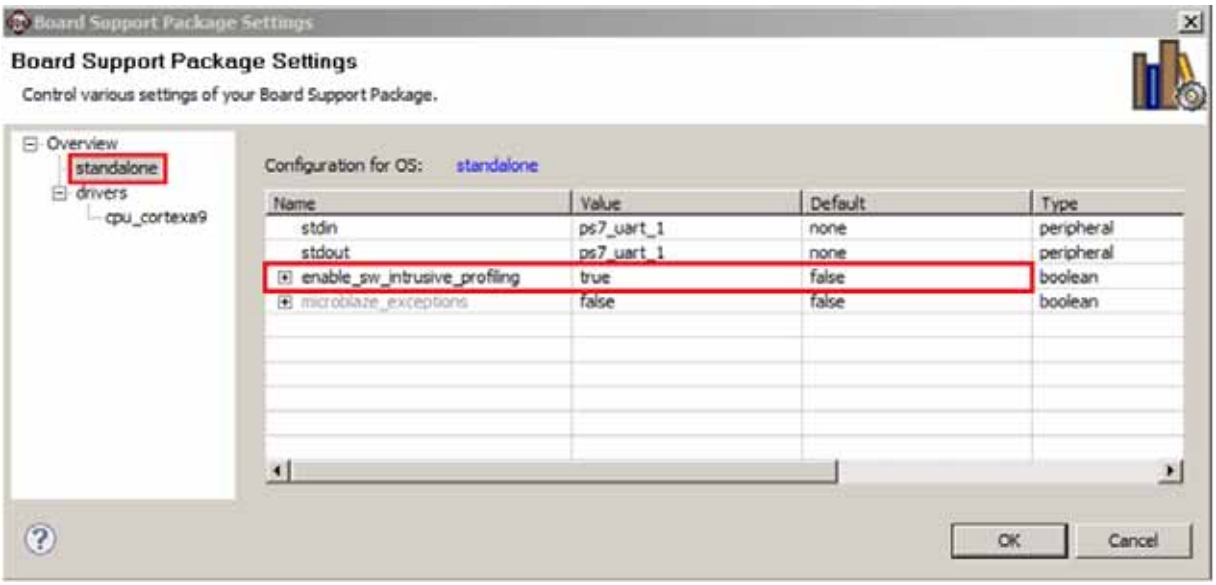

図 7-1 : BSP **設定** : **スタンドアロン コンフィギュレーション**

この設定により、ソフトウェアの即効型 (intrusive) プロファイリングが有効になります。このプロファイリング方法は評価対象のプログラムにコードを埋め込むことで実現するため、即効型ととらえらています。

13. [Board Support Package Settings] ダイアログ ボックスの左側のナビゲーション ペインで、**[drivers]** の下にある **[cpu_cortexa9]** を選択します。[extra_compiler_flags] の設定で、次のように [Value] 列に「`-pg`」と入力します。

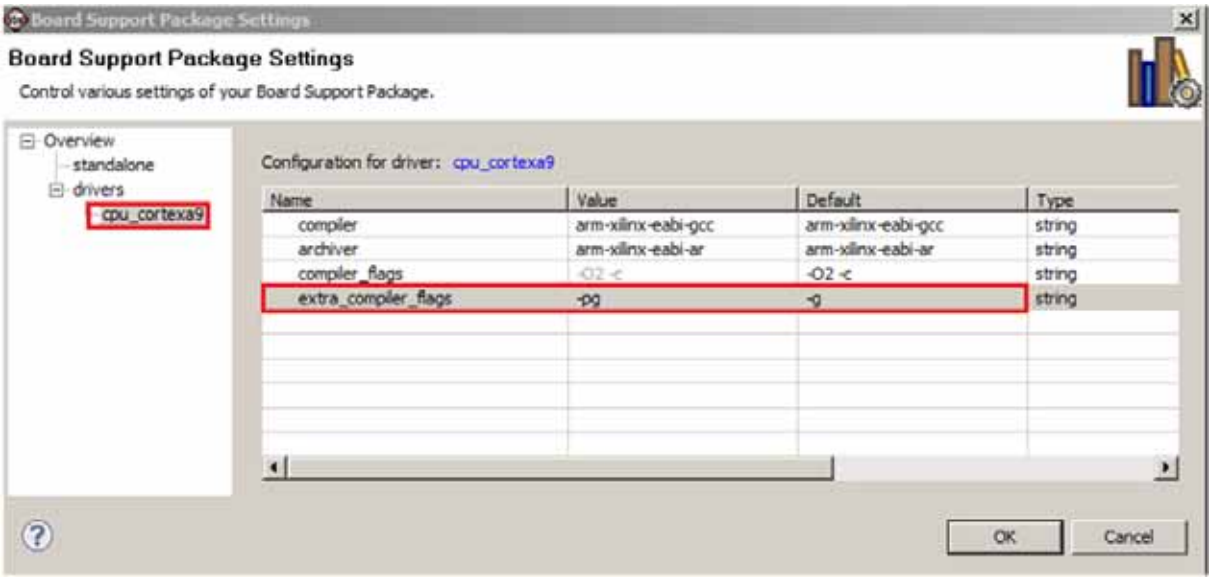

図 7-2 : BSP **設定** : cup_cortexa9 **の設定**

14. **[OK]** をクリックします。

SDK は、新しいプロファイリング オプションで再度 BSP の構築を開始します。

15. [Project Explorer] タブで **[cdma_app_profile]** を右クリックし、**[C/C++ Build]** をクリックして [Properties] ダイアログ ボックスを開きます。
16. [Properties] ダイアログ ボックスが表示されたら、[Tool Settings] タブをクリックします。**[ARM gcc compiler]** を展開し、**[Profiling]** をクリックしてから **[Enable Profiling (-pg)]** オプションを選択します。
17. プロパティ リストで **[Optimization]** を選択し、最適化レベルを **None (-O0)** に設定します。
18. プロパティ リストで **[Debugging]** を選択し、デバッグ レベルを **Default (-g)** に設定します。
19. **[OK]** をクリックします。

SDK はアプリケーションを再構築し、`cdma_app_profile.elf` ファイルを生成します。コンソール ウィンドウでステータスを確認します。

7.1.2  テスト ドライブ : アプリケーションにプロファイリング オプションを適用する

1. 「2.1.3 テスト ドライブ : Hello World アプリケーションを実行する」 の説明に従ってハードウェア ボードがセットアップされ、電源が投入されていることを確認してください。
2. SDK で **[Xilinx Tools]** → **[Program FPGA]** をクリックし、ビットストリーム パスが表示される [Program FPGA] ダイアログ ボックスを開きます。
3. **[Program]** をクリックし、ビットストリームをダウンロードして PL ファブリックをプログラムします。
4. [Project Explorer] タブで **[cdma_app_profile]** プロジェクトを右クリックし、**[Run As]** → **[Run Configurations]** をクリックして [Run Configuration] ダイアログ ボックスを開きます。
5. [Run Configuration] ダイアログ ボックスで、**[Xilinx C/C++ ELF]** をダブルクリックして新しいコンフィギュレーションを作成します。
6. **[Profile Options]** タブをクリックし、69 ページの図 7-3 のようにパラメーターを設定します。
   a. **[Enable Profiling]** チェック ボックスをオンにします。
   b. [Sampling Frequency (Hz)] を **10000** に設定します。

      サンプリング周波数は、どの関数がその時点で実行されているかをプロファイリング ルーチンが周期的に確認するために使用する割り込み間隔です。このルーチンは、割り込みごとにプログラム カウンターを確認し、サンプリングを実行します。

   c. [Histogram Bin Size (Words)] を **4** に設定します。

      ヒストグラムのビンのサイズは、現時点で実行されている関数を判断するために使用されるプログラム カウンター位置を確認する際の精度です。

   d. スクラッチ メモリ アドレスを **0x00300000** に設定します。

      スクラッチ メモリ アドレスは、BSP プロファイリングがデータ収集に使用可能な DDR メモリの位置を示します。アプリケーション プログラムはこの空間を使用できません。

図 7-3 : **プロファイリングを有効にする**

7. **[Run]** をクリックしてコンフィギュレーションを作成し、実行します。
8. [Reset Status] ダイアログ ボックスで **[OK]** をクリックします。

   プロファイリングが完了すると、ダイアログ ボックスが開いて結果ファイルの位置が表示されます。
9. **[OK]** をクリックします。
10. [Project Explorer] タブで `[cdma_app_profile]` の下にある **[Debug]** フォルダーを展開し、**[gmon.out]** をダブルクリックして [Gmon File Viewer] ダイアログ ボックスを開きます。

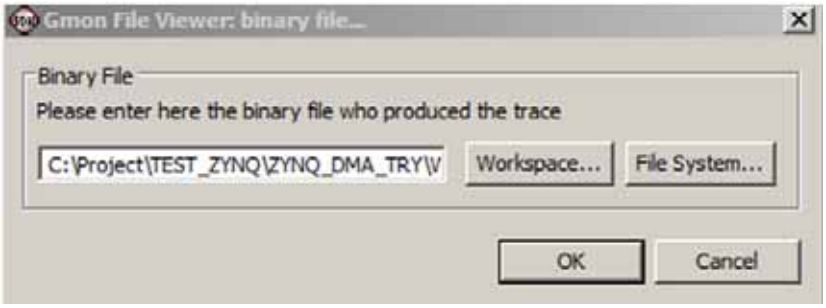

図 7-4 : [Gmon File Viewer] **ダイアログ ボックス**

11. **[OK]** をクリックし、ウィンドウの右側にある [gprof] タブでプロファイリング結果 (gmon.out) を開きます。図 7-5 と類似した表示になります。

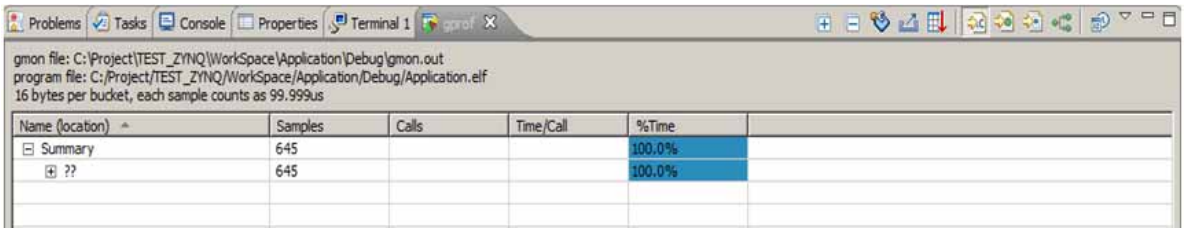

図 7-5 : gmon.out **ファイルの表示**

XILINX

第 8 章

アクセラレータ コヒーレンシ ポート (ACP)

プロセッシング システムの ACP はスヌープ制御ユニット (SCU) の AXI 準拠の 64 ビット スレーブ ポート インターフェイスで、プロセッサ ロジック (PL) とプロセシング システム (PS) の Cortex-A9 MP-Core プロセッサ システムを直接接続し、キャッシュ コヒーレントな非同期アクセス ポイントとして機能します。ACP ポートのトランザクションは、コヒーレントまたは非コヒーレントとマーク付けれらます。ファブリック側の AXI マスターは、AXI バスに関連する信号を用いてコヒーレントな読み出しトランザクションを ARUSERS[0] と ARCACHES[1] で、書き込みトランザクションを AWUSERS[0] と AWCACHES[1] で示します。デバイスまたは厳しく順序付けられたトランザクションは常に非コヒーレントとして処理されます。

注記 : デバイスまたは厳しく順序付けられたトランザクションは、AXI マスター インターフェイスで生成される転送タイプです。これらは、非コヒーレント、かつバッファリング不可のトランザクションです。固定サイズで決まった数のトランザクションが常に生成されます。

非コヒーレントなトランザクションは SCU を通過し、変更されずに AXI マスター インターフェイスへ渡されます。ACP スレーブのコヒーレントな AXI トランザクションは、一部の属性が適宜変更されて AXI マスター インターフェイスで増加または減少します。

コヒーレントな書き込み要求が外部のマスターから ACP で受信されると、SCU はコアの L1 データ キャッシュでアドレスを確認します。存在する場合は、コヒーレンシ プロトコルがキャッシュの該当するラインをクリーニングして無効化し、クリーニング後のデータを書き込み要求と結合させます。

外部マスターからコヒーレントなメモリ領域への読み出し要求は SCU と協調し、必要なデータがプロセッサの L1 キャッシュに格納されているかどうかを確認します。L1 キャッシュに格納されている場合、SCU はそのデータを要求発行元のコンポーネントに直接返します。格納されていない場合、L2 キャッシュ内を検索して最後にメイン メモリにアクセスします。

図 8-1 に、AXI インターコネクトを活用した、AXI-4 準拠のマスターと ACP 64 ビット ポート間の接続を示します。

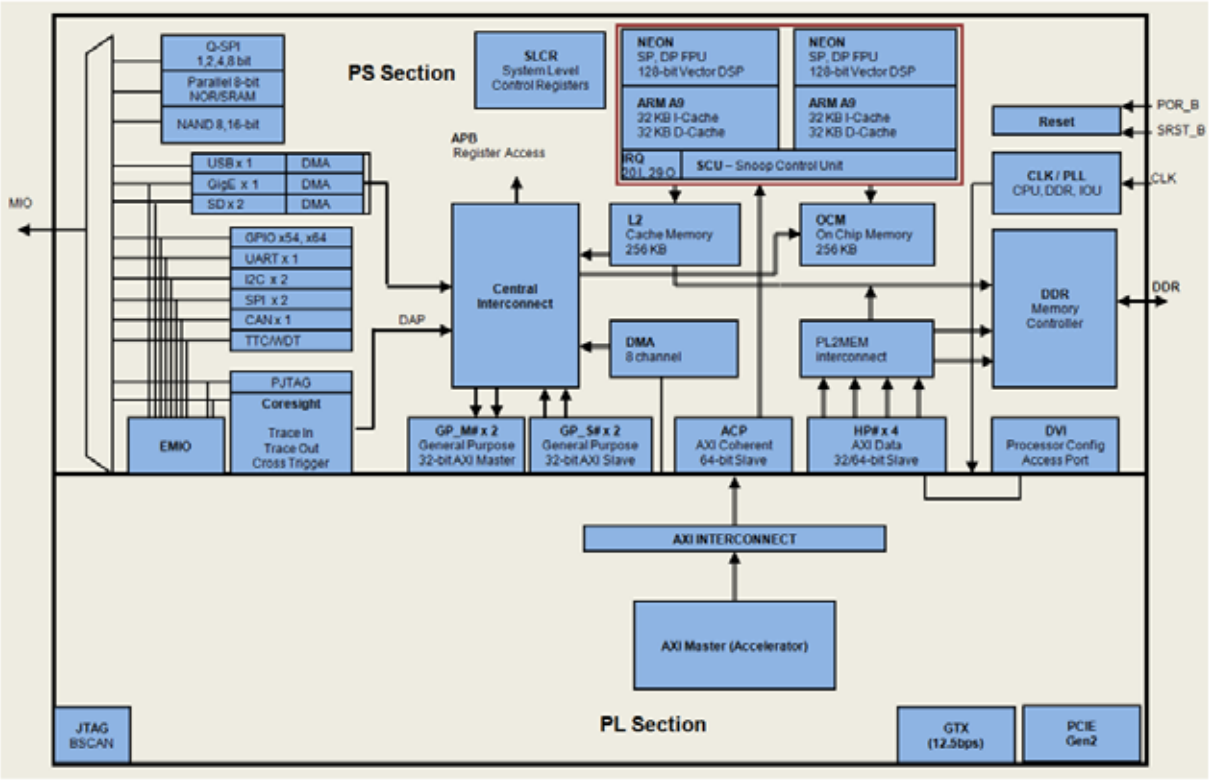

図 8-1 : AXI-4 準拠のマスターと ACP 64 ビット ポート間の接続

8.1 ACP の要求

ACP に対して読み出しおよび書き込み要求を実行した場合の動作は、その要求がコヒーレントかどうかによって次のように異なります。

8.1.1 コヒーレントな ACP 読み出し要求

ARVALID と一緒に ARUSER[0] = 1 および ARCACHE[1] = 1 が送信された場合、ACP の読み出し要求はコヒーレントです。この場合、SCU はコヒーレンシを維持します。データがいずれかの Cortex-A9 プロセッサにある場合、PS コアはそのプロセッサからデータを直接読み出して ACP に返します。データがいずれの Cortex-A9 プロセッサにもない場合、ロックされている属性を除くすべての AXI のパラメーターと共に読み出し要求が L2 キャッシュまたはいずれの Cortex-A9 プロセッサ AXI マスター ポートのメイン メモリへ発行されます。

8.1.2 非コヒーレントな ACP 読み出し要求

ARVALID と共に ARUSER[0] = 0 または ARCACHE[1] =0 が送信された場合、ACP の読み出し要求は非コヒーレントです。この場合、SCU はコヒーレンシを維持せず、読み出し要求はいずれか 1 つの SCU AXI マスター ポート L2 キャッシュまたは OCM へ直接転送されます。

8.1.3 コヒーレントな ACP 書き込み要求

AWVALID と一緒に AWUSER[0] = 1 および AWCACHE[1] =1 が送信された場合、ACP の書き込み要求はコヒーレントです。この場合、SCU はコヒーレンシを維持します。データがいずれかの Cortex-A9 プロセッサの L1 キャッシュにある場合、PS のロジックは最初にその CPU のデータをクリーニングして無効化します。データがいずれの Cortex-A9 プロセッサにもない場合、または既にクリーニングと無効化が実行されている場合、ロックされている属性を除くすべての AXI パラメーターと共に書き込み要求が L2 キャッシュまたはいずれの Cortex-A9 プロセッサ AXI マスター ポートのメイン メモリへ発行されます。

注記 : 書き込みパラメーターの設定によっては、トランザクションを L2 キャッシュに割り当てることもできます。

8.1.4 非コヒーレントな ACP 書き込み要求

AWVALID と共に AWUSER[0] = 1 または AWCACHE[1] = 0 が送信された場合、ACP の書き込み要求は非コヒーレントです。この場合、SCU はコヒーレンシを維持せず、書き込み要求はいずれか 1 つの SCU AXI マスター ポートに直接転送されます。

8.2 ACP の制約

アクセラレータ コヒーレンシ ポート (ACP) には次の制約があります。

- コヒーレントなメモリへの排他的アクセスはできない
- コヒーレントなメモリへは、ロック属性を有効にした アクセスはできない
- 長さ = 3、サイズ = 3、書き込みストローブ ‡ 11111111 の書き込みトランザクションを実行すると、CPU のキャッシュ ラインが破損する
- ACP から OCM へのアクセスを連続すると、ほかの AXI マスターからのアクセスができなくなることがある

ほかのマスターからアクセスできるようにするには、ACP から OCM への帯域幅を OCM の最大帯域幅よりも小さく設定する必要があります。具体的には、バースト サイズを 64 ビット ワード x8 未満に抑えることで可能です。

注記 : PS プロセッサ コアを使用すると、3 番目の制約 (キャッシュ ラインの破損) を検出できます。

この IP を有効にすると、キャッシュを破損する可能性のあるトランザクションをザイリンクス ACP アダプターが監視し、書き込み要求発行元のマスターにエラー応答を返します。書き込みトランザクションは ACP インターフェイスに送信されるため、キャッシュ破損の可能性は残りますが、問題発生の可能性がマスターに通知されるので適切な対処をとることができます。ACP アダプターには CPU への割り込み信号を生成する機能もあり、この信号をソフトウェアで使用して問題発生の可能性を検出できます。

ACP ポートの詳細は、次の資料を参照してください。これらの資料へのリンクは、付録 A 「その他のリソース」に記載されています。

- 『Zynq™-7000 EPP テクニカル リファレンス マニュアル』(UG585)
- 『The Effect and Technique of System Coherence in ARM Multicore Technology』、著者 : John Goodacre (Senior Program Manager、 ARM Processor Division)
- 『ARM Cortex-A9 MPCore Technical Reference Manual』、セクション 2.4 「Accelerator Coherency Port」

その他のリソース

A.1 このガイドのリソース

- この資料に関連する .zip ファイルには、第 5 章のチュートリアルのデザイン ファイルが含まれます。 ug873_design_files.zip は次のサイトから入手できます。 http://japan.xilinx.com/support/documentation/zynq-7000_user_guides.htm
- 『EDK コンセプト、ツール、テクニック ガイド』、第 6 章 「独自の IP を作成する」 http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_2/edk_ctt.pdf
- 『The Effect and Technique of System Coherence in ARM Multicore Technology』、著者 : John Goodacre (ARM Processor Division、 Senior Program Manager) (www.mpsoc-forum.org/previous/2008/slides/8-6%20Goodacre.pdf)
- 『ARM Cortex-A9 MPCore Technical Reference Manual』、セクション 2.4 「Accelerator Coherency Port」 (http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0407e/CACGGBCF.html)

A.2 トレーニング演習

このガイドのテスト ドライブに関連するトレーニング演習は次のサイトで提供されています。 http://japan.xilinx.com/training/embedded/embedded-design-tutorials.htm

A.3 ザイリンクスのリソース

- 『ザイリンクス デザイン ツール : インストールおよびライセンス ガイド』(UG798) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_2/iil.pdf
- 『ザイリンクス デザイン ツール : リリース ノート ガイド』 (UG631) : http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_2/irn.pdf
- **ザイリンクス®資料** : http://japan.xilinx.com/support/documentation
- **ザイリンクス用語集** : http://japan.xilinx.com/support/documentation/sw_manuals/glossary.pdf
- **ザイリンクス サポート サイト** : http://japan.xilinx.com/support/

A.4 EDK の資料

EDK に関する一連の資料は次のサイトから入手できます。 http://japan.xilinx.com/support/documentation/dt_edk_edk14-2.htm

- 『EDK のコンセプト、ツール、テクニック』(UG683) :
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_2/edk_ctt.pdf
- 『エンベデッド システム ツール リファレンス マニュアル』(UG111) :
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_2/est_rm.pdf
- 『MicroBlaze™ プロセッサ リファレンス ガイド』(UG081) :
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_2/mb_ref_guide.pdf
- 『プラットフォーム仕様フォーマットのリファレンス マニュアル』(UG642) :
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_2/psf_rm.pdf
- **『PowerPC 405 プロセッサ ブロック リファレンス ガイド』(UG018) :**
  http://japan.xilinx.com/support/documentation/user_guides/ug018.pdf
- **『PowerPC プロセッサ リファレンス ガイド』(UG011) :**
  http://japan.xilinx.com/support/documentation/user_guides/ug011.pdf
- 『Virtex®-5 FPGA の PowerPC 440 エンベデッド プロセッサ ブロック』(UG200) :
  http://japan.xilinx.com/support/documentation/user_guides/j_ug200.pdf
- 『Zynq-7000 ソフトウェア開発者向けガイド』(UG821) :
  http://japan.xilinx.com/support/documentation/user_guides/j_ug821-zynq-7000-swdev.pdf
- 『Zynq-7000 エクステンシブル プロセッシング プラットフォーム テクニカル リファレンス マニュアル』(UG585) : http://japan.xilinx.com/support/documentation/zynq-7000_user_guides.htm
- Zynq™-7000 関連の資料一覧 : http://japan.xilinx.com/support/documentation/zynq-7000.htm

A.5 EDK 関連のその他のリソース

- Xilinx Platform Studio および EDK のウェブサイト :
  http://japan.xilinx.com/ise/embedded_design_prod/platform_studio.htm
- Xilinx Platform Studio および EDK の資料一覧 :
  http://japan.xilinx.com/ise/embedded/edk_docs.htm
- ザイリンクス XPS/EDK がサポートする IP のウェブサイト :
  http://japan.xilinx.com/ise/embedded/edk_ip.htm
- ザイリンクス チュートリアルのウェブサイト :
  http://japan.xilinx.com/support/documentation/dt_edk_edk14-2_tutorials.htm
- ザイリンクス データシートの一覧 :
  http://japan.xilinx.com/support/documentation/data_sheets.htm
- ザイリンクス トラブルシュート :
  http://japan.xilinx.com/support/troubleshoot/psolvers.htm
- ザイリンクス ISE® Design Suite の資料一覧 :
  http://japan.xilinx.com/support/software_manuals.htm
- GNU の資料一覧 :
  http://www.gnu.org/manual