Zynq-7000 All Programmable SoC ソフトウェア開発者向けガイド

UG821 (v5.0) 2013 年 6 月 19 日

XILINX®

本資料は英語版 (v5.0) 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	内容
2013 年 3 月 20 日	v4.0	「ハードウェア デザイン ツール」を追加。IP インテグレーターのアーリー アクセス版に関する注釈を追加。 「RSA_SUPPORT」を追加。「FSBL マルチブート」を追加。定義内容を変更。 「FSBL フック」を追加。「DDR ECC イネーブル」を追加。 「セキュア ブート サポート」を追加。「認証証明」を追加。 表 3-1 および表 3-2 を追加。図 3-14 のブート イメージ フォーマットを変更 図 3-15 を変更。 「ブート イメージの生成」を変更。 Zynq の相互参照を追加し、U-Boot の PPC プロセッサを削除。 「付録 A : Bootgen の使用」へ Bootgen および BIT ファイルの説明を移動。

日付	バージョン	内容
2013 年 6 月 19 日	v5.0	29 ページの図 3-2を追加。 33 ページの「FSBL フォールバック機能」にフォールバックの説明を追加。 31 ページの「eMMC フラッシュ デバイス」を追加。 32 ページの「FSBL コンパイル フラグの設定」にコンパイル フラグを追加。 42 ページの「NAND ブート モード」を追加。 42 ページの「QSPI ブート モード」を追加。 46 ページの表 3-1 および 46 ページの表 3-2 の情報を変更。 付録 B に Zynq-7000 AP SoC 用 eFUSE を追加。 付録 B から eFUSE 情報を削除 ー この内容は『OS およびライブラリ資料コレクション』へ移動 eFUSE ライブラリへ参照を追加。 Zynq-7000 quick take ビデオへの参照を追加。 52 ページの「BIF ファイルの属性」の表に BIF オプションを追加。 Bootgen コマンド オプション (-intstyle、-split) を削除。UDP オプションに <filename> を追加。 52 ページの「BIF ファイルの属性」の図を変更。

目次

付録 B : その他のリソース

XILINX

第 1 章

はじめに

1.1 概要

このガイドでは、ザイリンクスの Zynq®-7000 All Programmable SoC デバイスを使用するデザインに役立つ情報をソフトウェアを中心に提供しています。主に次のような設計者と対象としています。

- エンベデッド ソフトウェア デザイン経験者
- ARM® 開発ツールの使用経験者
- ザイリンクスの FPGA デバイス、IP、開発ツール、およびツール環境について理解している設計者

1.2 はじめに

この文書は、次の内容で構成されています。

7 ページの「アーキテクチャの選択」では、All Programmable SoC の設計を開始する前に判断すべきアーキテクチャ情報について説明しています。

9 ページの「オペレーティング システム (OS) の考察」では、「ベアメタル」ソフトウェア システム (OS を使用しない)、Linux OS、およびリアルタイム OS について簡単に説明しています。

ハードウェアとソフトウェアのインターフェイスにハードウェアのプログラマビリティが追加されることで、デザイン フローに新たな要件が生じます。

ハードウェア協調シミュレーションや協調デバッグ機能など、一部のハードウェア機能はザイリンクス独自のものです。このような機能を利用することで、物理的ボードやエミュレーター上の Zynq-7000 AP SoC プロセッサでアプリケーションを実行しながら、Zynq-7000 AP SoC デバイスあるいはロジック シミュレーション環境にインプリメントされたカスタム ロジックを検証できます。11 ページの「ソフトウェア ツールの概要」を参照してください。

第 2 章「ソフトウェア アプリケーションの 開発フロー」では、Zynq-7000 AP SoC デバイス向けのアプリケーション開発およびデバッグに使用されるザイリンクス ツールの概要から始まり、ソフトウェア アプリケーション開発全体について説明しています。また、ベアメタル アプリケーション (ザイリンクス SDK ツールを使用) 開発およびエンベデッド Linux アプリケーション開発の一般的な手順も説明しています。

第 3 章「ブートおよびコンフィギュレーション」では、Zynq-7000 AP SoC デバイスのブート プロセスについて説明しています。設定可能なブート モードを紹介し、その後ブート ステージについて解説しています。この章には、ブート イメージの生成方法およびフラッシュ デバイスのプログラム方法も含まれます。

第 4 章「Linux」では、Git およびザイリンクスの公開 Git サーバーの使用法、Linux カーネルの図、U-Boot について説明し、これらに関連するその他の情報へのリンクを提供しています。

付録 A 「Bootgen の使用」では、Bootgen のユーティリティについて説明しています。

付録 B 「その他のリソース」では、関連するすべての資料とリンク (ある場合) を提供しています。

EDK を使用して Zynq ベースのエンベデッド システムを設計する際の詳細手順は、『Zynq-7000 All Programmable SoC コンセプト、ツール、およびテクニック ガイド』(UG873) [参照 7] を参照してください。

Vivado® Design Suite を使用して Zynq ベースのエンベデッド システムを設計する際の詳細手順は、『Vivado Design Suite チュートリアル : エンベデッド プロセッサ ハードウェア デザイン』(UG940) [参照 12] を参照してください。また、エンベデッド ハードウェア デザインのプロセスについては、『Vivado Design Suite ユーザー ガイド : エンベデッド ハードウェア デザイン』(UG898) [参照 11] を参照してください。

重要 : Vivado IP インテグレーターは、Zynq-7000 デバイスや MicroBlaze プロセッサをターゲットとするエンベデッド プロセッサ デザイン用の Xilinx Platform Studio (XPS) に代わるものです。XPS では MicroBlaze ™ プロセッサをターゲットとするデザインはサポートされますが、Zynq-7000 デバイスはサポートされません。IP インテグレーターと XPS は共に Vivado IDE から使用できます。

Zynq-7000 AP SoC デバイスに関するビデオは、
http://japan.xilinx.com/products/silicon-devices/soc/zynq-7000/smarter-system.html からご覧ください。

1.3 アーキテクチャの選択

Zynq-7000 AP SoC 上で実行するアプリケーションのエンベデッド開発を始める前に、アーキテクチャに関してユーザーが決定すべき事項がいくつかあります。

Zynq-7000 AP SoC デバイスには、デュアル コア ARM Cortex™-A9 プロセッサが搭載されているため、非対称型マルチプロセッシング (AMP) または対称型マルチプロセッシング (SMP) のいずれかを選択する必要があります。

すべてのエンベデッド ソフトウェア プロジェクトで、使用するオペレーティング システムを決定しておく必要があります (使用しない場合は不要)。この章では、AMP と SMP の両方について説明し、トレードオフやそれぞれについての注意点について解説しています。

1.3.1 マルチプロセッシングの考察

このセクションでは、マルチプロセッシングに関する 2 つの考慮すべき事項について説明します。

非対称型マルチプロセッシング

非対称型マルチプロセッシング (AMP) とは、マルチプロセッサ システムでそれぞれのプロセッサが物理的メモリを共有しながら、異なるオペレーティング システム イメージを実行するプロセッシング モデルのことをいいます。各イメージは同じオペレーティング システムのものであることも可能ですが、通常はそれぞれが異なるオペレーティング システムで、異なる特性を持つ別の OS を補完します。

- Linux などのフル機能を備えたオペレーティング システムは、ネットワークやユーザー インターフェイスを介して簡単に外部と接続できる
- 小規模で軽量なオペレーティング システムは、メモリやリアルタイム動作に関してより高い効率を発揮する

一般的な例として、Linux をプライマリ オペレーティング システムとして実行し、FreeRTOS またはベア メタル システム (9 ページの「ベア メタル システム」で説明) などの比較的小規模で軽量なオペレーティング システムをセカンダリ オペレーティング システムとして使用する場合が挙げられます。

プロセッサ間におけるシステム デバイス (UART、タイマー カウンター、イーサネットなど) の役割分担は、システム設計で最も重要な要素です。一般的には次のようになります。

- ほとんどのデバイスは、指定されたプロセッサ専用のデバイスとなる
- 割り込みコントローラーは、複数プロセッサ間で共有されるように設計する
- 割り込みコントローラーを初期化する割り込みコントローラー マスターとして 1 つのプロセッサを指定する

プロセッサ間の通信は、両方のオペレーティング システムを効率よく動作させるための重要な要素です。これを実現するには、プロセッサ間での割り込み、メモリの共有、メッセージの送信などさまざまな方法を用います。

対称型マルチプロセッシング

対称型マルチプロセッシング (SMP) とは、マルチプロセッサ システムでそれぞれのプロセッサが 1 つのオペレーティング システム イメージを実行するプロセッシング モデルのことをいいます。オペレーティング システムのスケジューラーによって、各プロセッサが実行するプロセスがスケジューリングされます。

効率良いプロセッシング モデルでは、選択した 1 つのオペレーティング システムがシステム要件を満たします。オペレーティング システムは自動的に複数のプロセッサの処理能力を活用し、その処理はエンドユーザーに透過的に行われます。ユーザーが可能な操作は次のとおりです。

- プロセスを実行する特定のプロセッサを指定する
- いずれかのプロセッサを使用して割り込みを処理する
- 1 つのプロセッサをシステムの初期化やほかのプロセッサを起動するマスターとして指定する

1.4 オペレーティング システム (OS) の考察

1.4.1 ベアメタル システム

ベアメタルとは、オペレーティング システムを使用しないソフトウェア システムのことをいいます。通常、このソフトウェア システムは、オペレーティング システムで提供される多くの機能 (ネットワークなど) を必要としません。オペレーティング システムはプロセッサの処理能力を幾分消費し、シンプルなソフトウェア システムと比較した場合、確定性が劣る傾向にあります。システム デザインによっては、このオペレーティング システムのオーバーヘッドや確定性の低さを許容できない場合があります。エンベデッド プロセッシングの処理速度が高くなるに伴い、システム デザインの多くでオペレーティング システムのオーバーヘッドは無視できるレベルとなりましたが、システムの複雑化を避けるために、オペレーティング システムを使用しない場合もあります。

1.4.2 オペレーティング システム : Linux の場合

Linux は、多くのエンベデッド デザインで使用されているオープンソースのオペレーティング システムです。さまざまなベンダーからディストリビューションとして提供されており、オープンソース リポジトリからの構築も可能です。本来、Linux はリアルタイム オペレーティング システムではありませんが、よりリアルタイム性に対応できるようになっています。

Linux は、プロセッサのメモリ管理ユニット (MMU) を利用するフル機能を備えたオペレーティング システムであり、安全性の高いオペレーティング システムとして評価されています。また、複数プロセッサを使用する際の SMP 機能もあります。

1.4.3 リアルタイム オペレーティング システム

ザイリンクスのサードパーティ パートナーが提供するリアルタイム オペレーティング システム (RTOS) を使用するシステム設計者もいます。

RTOS は、タイミングを重視するアプリケーションやシステムで求められる確定性や予測可能な応答性を備えています。

最新のサードパーティ ツールに関する情報は、ザイリンクスへお問い合わせください。

1.4.4 パートナー企業の Zynq-7000 オペレーティング システム

設計者は、独自の経験、最新規格、特定要件、レガシ デザイン、企業間契約などに応じてエンベデッド ソリューションを選択できます。

Zynq-7000 デバイスでサポートされているパートナー企業のオペレーティング システムの一覧は、次のリンクを参照してください。

http://japan.xilinx.com/products/silicon-devices/soc/zynq-7000/ecosystem/index.htm

XILINX

第 2 章

ソフトウェア アプリケーションの開発フロー

2.1 概要

Zynq®-7000 All Programmable (AP) SoC ソフトウェア アプリケーション開発フローでは、統合されたザイリンクスのツール セットを使用してソフトウェア アプリケーションを構築できます。さらに、ARM® Cortex™-A9 プロセッサ用にサードパーティが提供するさまざまなツールも利用可能です。

この章では、ザイリンクスのツールおよびフローを中心に説明していますが、その概念は広くサードパーティ ツールに応用できます。Zynq-7000 AP SoC デバイス ソリューションには、Eclipse ベースの統合設計環境 (IDE) や GNU コンパイラ ツールチェーンなどの使い慣れたコンポーネントが含まれています。

さらに、ザイリンクス ツールを使用する、ベアメタルおよび Linux のソフトウェア アプリケーション開発フローについても簡単に説明しています。ザイリンクス ツールのサポートは、その他のザイリンクス エンベデッド プロセッサと同様ですが、異なる部分については注記しています。また、アプリケーション開発フローに含まれるブート、デバイス コンフィギュレーション、および OS 使用法についても説明しています。これらについては、ほかの章および資料で詳しく解説しています。

11 ページの図 2-1 に、Zynq-7000 AP SoC プロセッサのブロック図を示します。

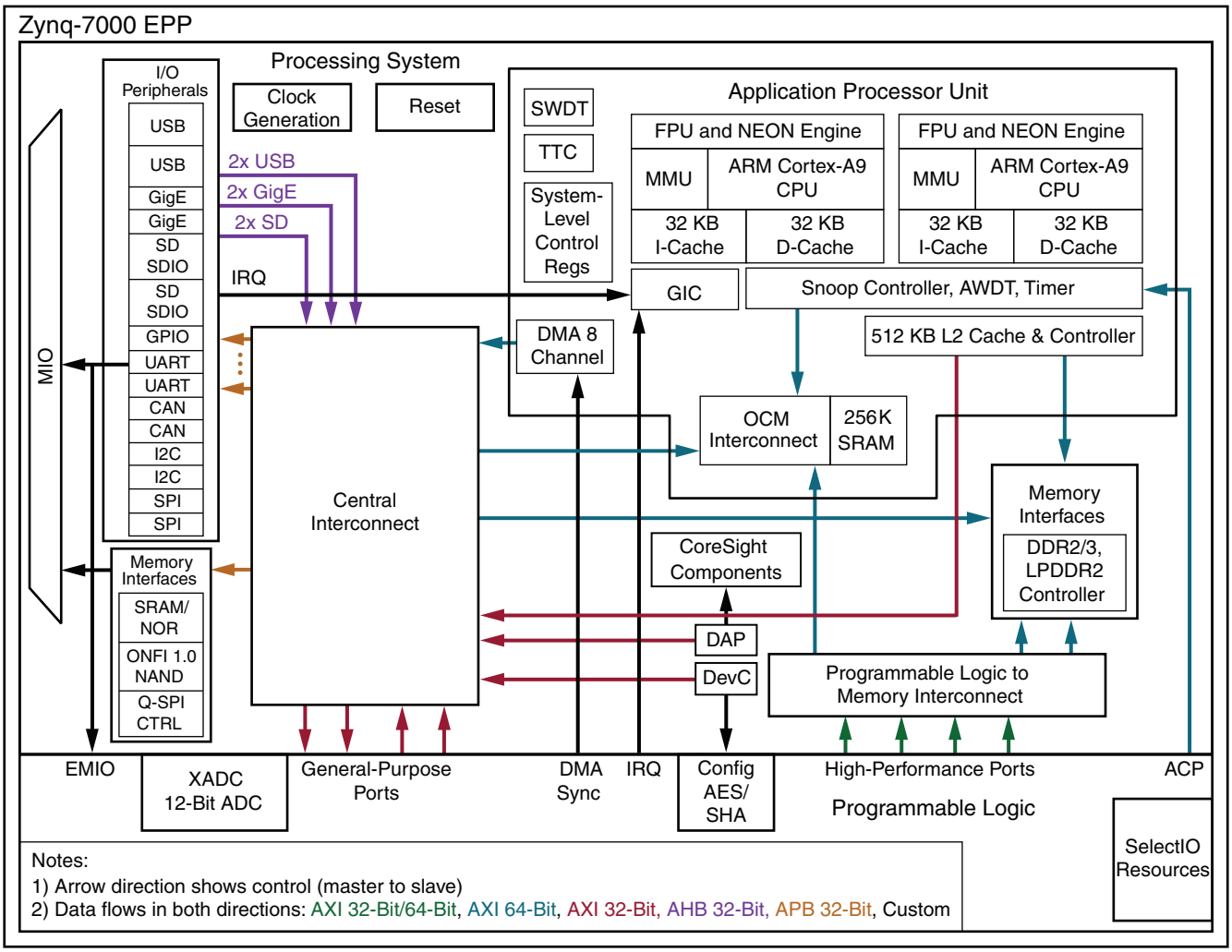

図 2-1 : Zynq-7000 AP SoC プロセッサ システムの概略図

2.2 ソフトウェア ツールの概要

ARM ベースのプロセッシング システム (PS) とプログラマブル ロジック (PL) が結合されたことによって、カスタムのペリフェラルやコプロセッサが追加できるようになり、より独自性のある開発が可能になりました。PL にインプリメントされたカスタム ロジックによって、タイミングが重要なソフトウェア機能の強化、アプリケーション レイテンシの削減、システムの消費電力削減、あるいはソリューション固有のハードウェア機能追加などが実現します。

ハードウェアとソフトウェアのインターフェイスにハードウェアのプログラマビリティが追加されることで、デザイン フローに新たな要件が生じます。ハードウェア協調シミュレーションや協調デバッグ機能など、一部のハードウェア機能はザイリンクス独自のものです。このような機能を利用することで、物理的ボードやエミュレーター上の Zynq-7000 AP SoC プロセッサでアプリケーションを実行しながら、Zynq-7000 AP SoC デバイスあるいはロジック シミュレーション環境にインプリメントされたカスタム ロジックを検証できます。

ザイリンクスは、Zynq-7000 AP SoC デバイスのソフトウェア アプリケーションの開発およびデバッグ向けの設計ツールを提供しています。

- ソフトウェア IDE

- GNU ベースのコンパイラ ツールチェーン
- JTAG デバッガー
- 関連するユーティリティ

これらのツールでは、次のアプリケーション開発が可能です。

- OS を使用しないベアメタル型アプリケーション
- オープン ソース Linux OS のアプリケーション

カスタム ロジックとユーザー ソフトウェアは、物理的ハードウェアやシミュレーションのさまざまな組み合わせで動作でき、ハードウェアのイベントを監視する機能も備えています。次に例を示します。

- ハードウェアまたはシミュレーション ツールで動作するカスタム ロジック
- ターゲット上またはソフトウェア エミュレーターで動作するユーザー ソフトウェア
- イベント時の PL およびプロセッサのクロストリガ

Cortex-A9 プロセッサをサポートするサードパーティのソフトウェア ソリューションも利用できます。一部を次に示します。

- ソフトウェア IDE
- コンパイラ ツールチェーン
- デバッグおよびトレース ポート
- エンベデッド OS とソフトウェア ライブラリ
- シミュレータ
- モデルおよび仮想プロトタイピング ツール

サードパーティ ツール ソリューションは、統合レベルおよび Zynq-7000 AP SoC デバイスの直接サポート レベルがさまざまです。カーネルの開発およびデバッグ専用ツールをザイリンクスは提供していませんが、サードパーティから入手可能なツールがあります。それらをご利用ください。

後続のサブセクションでは、ザイリンクスの開発ツールの概要を説明します。ツールは、32 ビットおよび 64 ビット Windows および x86 Linux ホスト コンピューティング プラットフォームで使用できます。

2.2.1 ハードウェア設計ツール

ザイリンクスは、次のハードウェア設計ツールを提供しています。

- ISE® Design Suite エンベデッド開発キット (EDK) に含まれる Xilinx Platform Studio (XPS) : コンフィギュレーション設定、レジスタ メモリ マップ、PL 初期化用ビットストリームなどの PS およびペリフェラルに関する情報を管理します。
- Vivado IP インテグレーター : ブロック図を使用しながら、PL や Zynq-7000 プロセッサと関連している IP を設定します。

重要 : Vivado IP インテグレーターは、Zynq-7000 デバイスや MicroBlaze プロセッサをターゲットとするエンベデッド プロセッサ デザイン用の Xilinx Platform Studio (XPS) に代わるものです。XPS では MicroBlaze プロセッサをターゲットとするデザインはサポートされますが、Zynq-7000 デバイスはサポートされません。IP インテグレーターと XPS は共に Vivado IDE から利用できます。

重要 : このユーザー ガイドでは、XPS または IP インテグレーターのいずれかを利用する場合、ザイリンクスの「ハードウェア設計ツール」として言及しています。

Xilinx Platform Studio

XPS では、XML 形式のハードウェア プラットフォーム情報とその他のデータ ファイルを取り込んだ後、ソフトウェア設計ツールがそれらの情報を使用してボード サポート パッケージ (BSP) ライブラリの作成および設定、コンパイラ オプションの推論、PL のプログラム、JTAG 設定の定義、そしてハードウェア情報を必要とするその他の動作の自動化を行います。

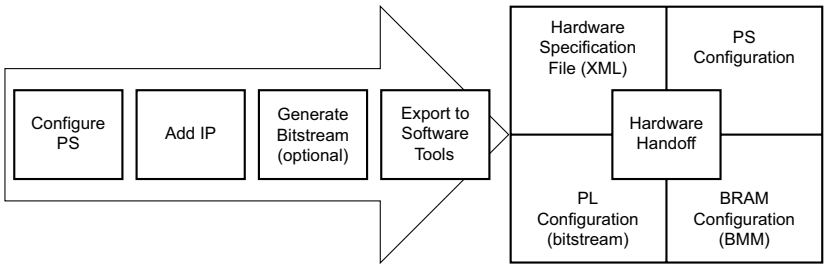

図 2-2 : XPS ハードウェアからソフトウェア ツールへのハンドオフ

Vivado IP インテグレーター

Vivado Design Suite IP インテグレーターは Zynq-7000 AP SoC 用のブロック図を提供するため、ユーザーは XML ファイル形式や INIT ファイル形式 (`.h`、`.c` および `.tcl`) でプログラマブル ロジック (PL) 情報を設定できます。ソフトウェア設計ツールはこれらの情報を使用してボード サポート パッケージ (BSP) ライブラリの作成および変更、コンパイラ オプションの推論、JTAG 設定の定義、そしてハードウェア情報を必要とするその他の動作の自動化を行います。

詳細は、次の資料を参照してください。

- 『Vivado Design Suite ユーザー ガイド : エンベデッド ハードウェア デザイン』(UG898) [参照 11]
- 『Vivado Design Suite チュートリアル : エンベデッド ハードウェア デザイン』(UG940) [参照 12]
- 『Vivado Design Suite チュートリアル : IP インテグレーターを使用して IP サブシステムを設計』(UG994) [参照 14]
- 『Vivado Design Suite ユーザー ガイド : Vivado IDE の使用』(UG893) [参照 13]

2.2.2 ソフトウェア開発キット

ザイリンクス ソフトウェア開発キット (SDK) は、ザイリンクス エンベデッド プロセッサをターゲットとするソフトウェア アプリケーションの開発に必要なすべてが揃った完全な環境を提供します。このキットには、GNU ベースのコンパイラ ツールチェーン (GCC コンパイラ、GDB デバッガー、ユーティリティ、およびライブラリ)、JTAG デバッガー、フラッシュ プログラマー、ザイリンクス IP 用ドライバー、ベアメタル ボード サポート パッケージ、アプリケーション固有機能用ミドルウェア ライブラリ、C/C++ ベアメタルおよび Linux アプリケーション開発/デバッグ用 IDE が含まれています。オープンソースの Eclipse プラットフォームがベースとなる SDK には、C/C++ 開発ツールキット (CDT) が統合されています。CDT の特徴は次のとおりです。

- C/C++ コード エディターおよびコンパイル環境
- プロジェクト管理
- アプリケーションの構築設定および `makefile` の自動生成
- エラー ナビゲーション
- エンベデッド ターゲットのデバッグおよびプロファイル用の統合環境
- サードパーティのプラグインを使用して利用可能な追加機能 (例 : ソース コードのバージョン管理)

SDK の利用

SDK は、ハードウェア情報を渡すことができるすべてのザイリンクス設計ツール (ISE® Design Suite インストール パッケージ、ザイリンクスのエンベデッド開発キット (EDK)、および Vivado IDE) で利用できます。また、スタンドアロンのアプリケーションとしても利用可能です。SDK には、第 1 段階ブートローダー (FSBL) 作成用のアプリケーション テンプレートおよびブート イメージ構築用のグラフィカル インターフェイスも含まれています。その他に、コンセプト、タスク、参照情報を提供するヘルプもあります。

ハードウェアの定義をエクスポートする場合、図 2-3 のように Vivado から SDK を起動できます。

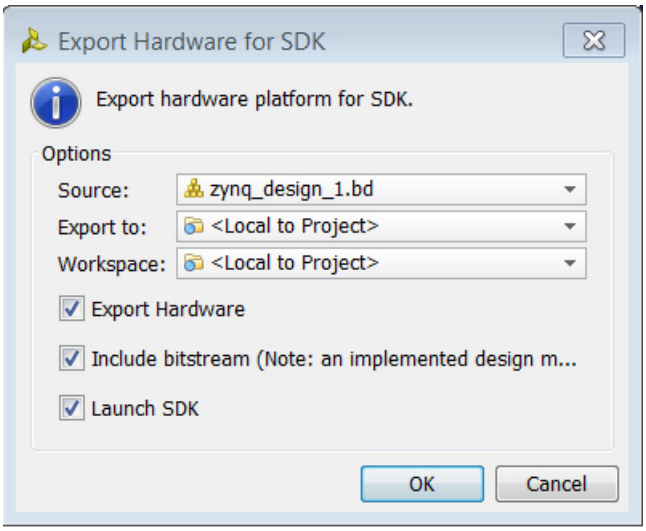

図 2-3 : [Export Hardware for SDK] ダイアログ ボックス

2.2.3 マイクロプロセッサ デバッガー

ザイリンクス マイクロプロセッサ デバッガー (XMD) は、コマンド ラインから起動してプログラムをダウンロード、デバッグ、検証するための JTAG デバッガーであり、繰り返し実行するタスクや複雑なタスクのスクリプト化 (自動化) をサポートする Tcl (ツール コマンド言語) インターフェイスを含みます。XMD は、ソースレベルのデバッガーではなく、ベアメタル アプリケーションをデバッグする際の GDB および SDK の GDB サーバーとして機能します。

Linux アプリケーションをデバッグする場合は、ターゲット上で動作している GDB サーバーと SDK が相互作用します。デバッガーは、同じホスト コンピューター上、またはネットワーク上のリモート コンピューター上で動作している XMD へ接続できます。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』(UG111) [参照 16] を参照してください。

2.2.4 ザイリンクス Cortex-A9 コンパイラ ツールチェーン用の Sourcery CodeBench Lite Edition

SDK には、ベアメタル EABI (エンベデッド アプリケーション バイナリ インターフェイス) および Linux アプリケーション開発向けのザイリンクス Cortex-A9 コンパイラ ツールチェーン用 Sourcery CodeBench Lite Edition が含まれています。

SDK のザイリンクス Sourcery CodeBench Lite ツールチェーンには、標準の Sourcery CodeBench Lite Edition EABI や Linux コンパイラ ツールチェーンと同じ GNU ツール、ライブラリ、資料が含まれていますが、その他に次が追加されています。

- ザイリンクス Cortex-A9 プロセッサ用のデフォルト ツールチェーン設定

- ザイリンクス Cortex-A9 プロセッサ用のベアメタル (EABI) スタートアップ サポートおよびデフォルト リンカー スクリプト
- ベクター浮動小数点 (VFP) と NEON™ に最適化されたライブラリ

2.2.5 解析ツール

ChipScope Pro Analyzer

ChipScope™ Pro Analyzer は、PL デザインのカスタム ロジックにロジック アナライザー、システム アナライザー、および仮想 I/O コアを挿入し、内部信号やノードの様子を視覚的に確認できます。

解析する信号は動作速度でキャプチャされ、ChipScope デバッグ ツールで表示して解析できます。カスタム ロジックでのイベント (変化) がソフトウェア デバッガーのブレークポイントをトリガーし、プロセッサを実行するプログラムを停止します (その逆も可能)。SDK、ChipScope Pro Analyzer、およびその他のザイリンクス ツールを使用したさまざまな協調デバッグ フローがサポートされています。ISE Design Suite プロジェクトをデバッグする場合の詳細説明は、『ChipScope Pro ソフトウェアおよびコア ユーザー ガイド』(UG029) を参照してください。

Vivado ラボ ツール

Vivado IDE にはデバッグ機能が統合されています。詳細は、『Vivado Design Suite ユーザー ガイド : プログラムおよびデバッグ』(UG908) [参照 17] を参照してください。

2.2.6 System Generator for DSP

System Generator™ for DSP ツールは、DSP およびデータ フローを中心とするハードウェア ベースのコプロセッサ開発に使用可能で、MATLAB®/Simulink® 環境内で動作します。

このツールは、DSP ハードウェアの高速シミュレーションをサポートしているため、全体的な開発時間を短縮するだけでなく、PS へ接続できるコプロセッサの生成を自動化します。SDK の協調デバッグ機能を使用することで、System Generator で開発中のハードウェアを確認および制御しながら、プロセッサ上で動作しているプログラムを SDK 内で実行およびデバッグできます。

2.2.7 ISim Simulator

ISE® および PlanAhead™ ツールには ISim HDL シミュレータが含まれているため、物理的なボードを使用しなくても、PL にインプリメントされているカスタム ハードウェア ロジックを検証およびデバッグできます。ISim ハードウェア協調シミュレーション (HWCoSim) テクノロジを用いることによって、ISE simulator でカスタム ロジックをデバッグしながら、同時にエミュレーターでターゲット プロセッサ上で動作するプログラム、あるいは SDK でターゲット デバイス上で動作するプログラムのデバッグが可能です。

2.3 ベアメタル デバイス ドライバーのアーキテクチャ

ベアメタル デバイス ドライバーは、階層構造になっています (16 ページの図 2-4 参照)。この階層アーキテクチャは、デバイス ドライバーのさまざまな使用事例に対応できると同時に、オペレーティング システム、ツールセット、プロセッサ間の移植性にも優れています。

階層アーキテクチャは、次のコンポーネントとシームレスに統合します。

- 「階層 2 (RTOS アダプター)」- フル機能でオペレーティング システム間の移植が可能な抽象的デバイス ドライバー インターフェイス
- 「階層 1 (デバイス ドライバー)」- プロセッサ
- シンプルな使用事例またはカスタム デバイス ドライバー開発向けの直接ハードウェア インターフェイス

以降のサブセクションで各階層について説明します。

重要 : 直接ハードウェア インターフェイスは通常、記号定数 (manifest constant) とマクロとして実装されているため、デバイス ドライバーの関数呼び出しにオーバーヘッドを追加しません。

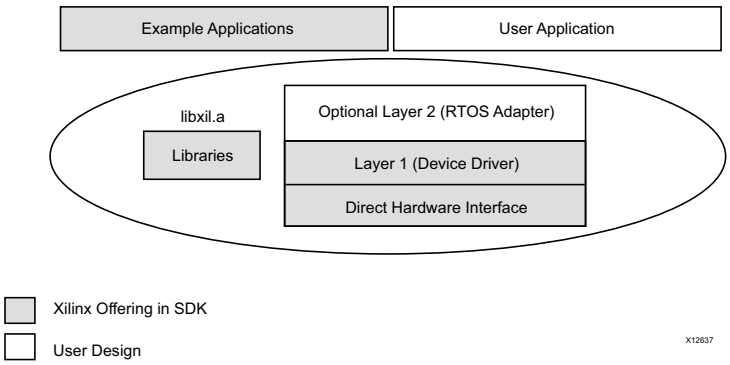

図 2-4 : ベアメタル デバイス ドライバーのアーキテクチャ

2.3.1 階層 2 (RTOS アダプター)

階層 2 は、RTOS とデバイス ドライバー間のアダプターです。これは、階層 1 のデバイス ドライバーを RTOS 用ドライバー モデルの要件に合うインターフェイスへ変換します。各 RTOS に固有のアダプターが必要になることがあります。

アダプターの一般的な特徴は次のとおりです

- RTOS およびデバイス ドライバーの階層 1 インターフェイスと直接通信
- RTOS に固有のファンクションおよび識別子を参照 (したがって、この階層はオペレーティング システム間で移植不可)
- メモリ管理を使用できる
- スレッディングやタスク間通信などの RTOS サービスを使用できる
- RTOS インターフェイスおよびデバイス ドライバー要件によって、単純または複雑な構造になる

2.3.2 階層 1 (デバイス ドライバー)

階層 1 は、下層ハードウェアへの潜在的な変更からユーザー アプリケーションを保護する抽象化デバイス ドライバー インターフェイスです。マクロおよび関数で実装されており、開発者がデバイスの機能すべてを有効活用できるように設計されています。デバイス ドライバーは、オペレーティング システムとプロセッサから独立しているため、移植性に優れています。このインターフェイスの一般的な特徴は次のとおりです。

- ユーザーがソリューションをすぐに利用できるようにする確実な API。抽象化された API は、ハードウェアの変更からユーザー アプリケーションを保護する
- RTOS やプロセッサに依存しないため、デバイス ドライバーは移植性に優れている
- 入力引数のアサートなどのランタイム エラー チェック、コンパイルによってエラーを削除
- デバイス機能のサポート
- デバイス コンフィギュレーションのパラメーターをサポートし、FPGA ベースのハードウェア デバイスのパラメーター化
- デバイスの複数インスタンスをサポートし、各インスタンスに固有の特性を管理する
- ポーリングおよび割り込みで駆動される I/O
- 複雑なアプリケーションをサポートするノンブロッキング関数呼び出し
- 大規模なメモリ フットプリント
- バイト　インターフェイスと対照的なデータ転送用バッファー　インターフェイス。複雑なアプリケーションで API の使用が容易になる
- 上位階層との通信に非同期コールバックを使用しているため、階層 2 アダプターやアプリケーション ソフトウェアと直接通信しない

2.3.3 直接ハードウェア インターフェイス

階層 1 デバイス ドライバー内に含まれるインターフェイスは、直接ハードウェア インターフェイスです。通常、このインターフェイスはマクロや明示定数として実装され、小規模アプリケーションやカスタム デバイス ドライバーを作成できるよう設計されています。このインターフェイスの一般的な特徴は次のとおりです。

- デバイス レジスタのオフセットやビット フィールドを定義する定数
- ハードウェア レジスタへアクセスするためのシンプルなマクロ
- 小規模なメモリ フットプリント
- エラー チェック機能はほとんどない、またはまったくない
- 最低限の抽象レベルであるため、通常 API はデバイス レジスタと一致する。API はハードウェア デバイスの変更からの保護レベルが低い

- デバイス コンフィギュレーション パラメーターをサポートしない
- API へのベース アドレス入力による、デバイスの複数インスタンスのサポート
- ステートなし、または最小限
- ポーリング I/O のみ
- シンプルな使用事例向けのブロッキング機能
- 一般的にバイト インターフェイスが提供される

2.4 ベアメタル アプリケーションの開発

ザイリンクスのソフトウェア設計ツールは、さまざまなランタイム環境でのエンベデッド ソフトウェア アプリケーション開発を容易にします。

ザイリンクスのエンベデッド設計ツールを使用し、次を含むハードウェア プラットフォームのデータ ファイルを作成します。

- XML 形式のハードウェア記述ファイル - プロセッサ、ペリフェラル、メモリ マップ、その他のシステム データ情報が含まれている
- ビットストリーム ファイル - プログラマブル ロジック (PL) のプログラム データが含まれている (オプション)
- ブロック RAM メモリ マップ (BMM) ファイル
- Zynq-7000 AP SoC の FSBL (第 1 段階ブートローダー) で使用される PL コンフィギュレーション データ

ベアメタル BSP (ボード サポート パッケージ) は、ユーザー アプリケーションの下位層を形成するライブラリとドライバーを集めたものです。

ランタイム環境は、基本的な機能を提供する単純なセミホスト型のシングル スレッド環境で、ブート コード、キャッシュ機能、例外処理、基本のファイル I/O、メモリ割り当てやその他の呼び出し用の C ライブラリ サポート、プロセッサ ハードウェア アクセス マクロ、タイマー機能、ベアメタル アプリケーションをサポートする機能などが多数含まれています。

ハードウェア プラットフォームのデータとベアメタル BSP を使用することで、SDK でベアメタル アプリケーションの開発、デバッグ、展開が可能です。

図 2-5 に、ベアメタル アプリケーション開発のフロー チャートを示します。

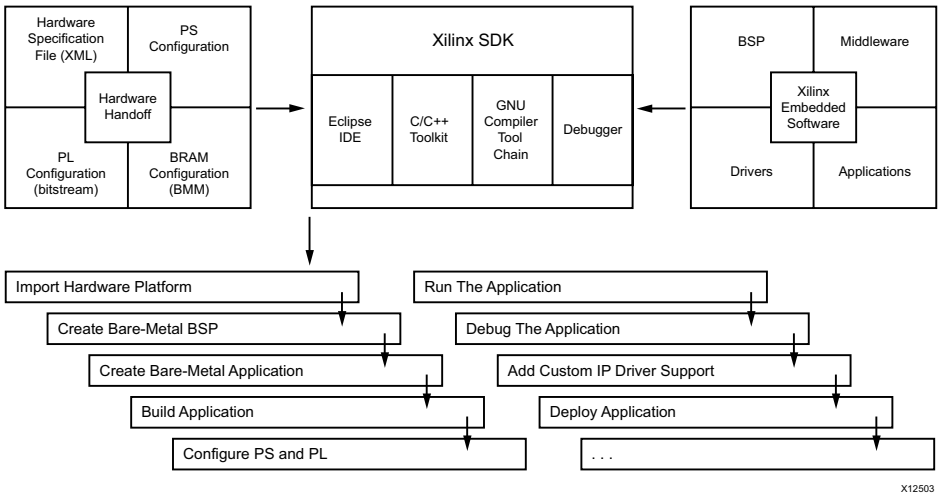

図 2-5 : ベアメタル アプリケーション開発の概要

SDK でのベアメタル アプリケーション開発の一般的な手順は次のとおりです。

1. 「ハードウェア プラットフォーム情報をインポートする」
2. 「ベアメタル BSP を作成する」
3. 「ベアメタル アプリケーションを作成する」
4. 「アプリケーション プロジェクトを構築する」
5. 「デバイスおよび実行アプリケーションをプログラムする」
6. 「アプリケーションをデバッグする」
7. 「カスタム IP ドライバーのサポートを追加する」
8. 「アプリケーションを展開する」

以降のサブセクションでは、これらの手順について説明しています。SDL ツールの詳細および使用例については、SDK オンライン ヘルプ、または『Zynq EDK コンセプト、ツール、およびテクニック ガイド』(UG873) [参照 7] を参照してください。

2.4.1 ハードウェア プラットフォーム情報をインポートする

ザイリンクスのハードウェア設計ツールは、ハードウェア プラットフォーム プロジェクトを作成するために、SDK へエクスポートできるハードウェア プラットフォーム データを作成します。SDK では、プロジェクト内でハードウェア システムに関する次のような情報を管理しますが、これに限ったものではありません。

- BSP 生成用のプロセッサおよびペリフェラル情報
- リンカー スクリプト生成用のメモリ マップ情報
- カスタム ロジックを含む PL をプログラムする際に使用するビットストリーム データ
- FSBL およびデバッガーで使用される PS コンフィギュレーション データ

2.4.2 ベアメタル BSP を作成する

ハードウェア プラットフォーム プロジェクトの作成後、SDK でベアメタル BSP プロジェクトを作成します。ドライバーおよびライブラリのソース ファイルは段階ごとに分類され、ハードウェア プラットフォーム (プロセッサ、IP、機能セット、ハードウェア コンフィギュレーション設定) に基づいてパラメーター化されてヘッダー ファイル パラ

メーターを定義し、その後コンパイルされます。BPS は、多重 I/O (MIO) コンフィギュレーションを含む PS で有効にされた IP や PL 内のカスタム ロジックを反映します。BSP の設定は変更/再生成できます。詳細は、『OS およびライブラリ資料コレクション』(UG643) [参照 8] に含まれる『スタンドアロン BSP』(UG652) [参照 7] を参照してください。

2.4.3 サードパーティ ツールを使用してベアメタル BSP を作成する

SDK では、関連するドライバーとライブラリを設定および構築するためのソース ファイルやメタデータ ファイルを含むソフトウェア リポジトリへのパスを指定することによって、その他のエンベデッド OS 環境およびツールの BSP を生成できます。

2.4.4 ベアメタル アプリケーションを作成する

SDK では、基本の「Hello World」や Dhrystone ベンチマーク アプリケーションから FSBL や TCP/IP Echo サーバーまで、キットに含まれるサンプル プログラム用のテンプレート ベースのアプリケーション ジェネレーターを提供しています。これらのアプリケーションのデフォルトのリンカー スクリプトが作成されます。

アプリケーション ジェネレーターは、ザイリンクスの C または C++ アプリケーション ウィザードで起動されます。空のアプリケーションを作成するか、あるいは既存アプリケーションをインポートしてベアメタル BSP へ移植することも可能です。各アプリケーション プロジェクトは 1 つの BSP プロジェクトと関連付けられます。

コード開発ツールには、エディター、検索、リファクタリングなどのツールのほかに、ベースの Eclipse プラットフォームや CDT プラグインで利用できる機能があります。

2.4.5 アプリケーション プロジェクトを構築する

SDK アプリケーション プロジェクトは、ユーザー管理 (ユーザーが `makefile` を作成) または自動管理 (SDK が `makefile` を作成) が可能です。ユーザー管理のプロジェクトでは、ユーザーが `makefile` を管理してアプリケーションの構築を開始する必要があります。

自動管理のプロジェクトでは、ソース ファイルが追加または削除されたときに SDK が必要に応じて `makefile` を更新し、変更が保存されて ELF が自動生成されると、ソース ファイルがコンパイルされます。Eclipse CDT の用語では、アプリケーション プロジェクトは Managed Make プロジェクトと呼ばれています。

可能な場合は、SDK が使用するハードウェア プラットフォームおよび BSP に基づいてデフォルトの構築オプション (コンパイラ、リンカー、ライブラリ パス オプションなど) を推論して設定します。

2.4.6 デバイスおよび実行アプリケーションをプログラムする

ベアメタル アプリケーションの構築後、SDK を使用して PS をコンフィギュレーションし、PL をプログラムしてアプリケーションを実行します。SDK は、FSBL で使用されるコンフィギュレーション データを含むシステム レベル コンフィギュレーション レジスタ (SLCR) を用いて PS をコンフィギュレーションします。

ビットストリーム (BIT) およびブロック メモリ マップ (BMM) データが Zynq-7000 AP SoC へダウンロードされると、すべてのカスタム デザイン ロジックが PL へロード可能になりますが、PS のみ必要なアプリケーションを実行する場合はこの手順を省略できます。

SDK で実行コンフィギュレーションを作成し、アプリケーションの ELT ファイルをダウンロードして実行します。アプリケーションとの相互通信には、`STDIN` および `STDOUT` を使用して端末表示が可能です。

2.4.7 アプリケーションをデバッグする

SDK でアプリケーションをデバッグする手順は、アプリケーションを実行する場合とほとんど同じですが、実行コンフィギュレーションを作成する代わりにデバッグ コンフィギュレーションを作成します。一連のウィンドウ (ビュー) によって、完全なデバッグ環境が提供されます。このデバッグ方法は、CDT プラグインを追加する Eclipse

ベースの IDE を使用した経験のあるユーザーにとっては使い慣れた環境となり、セッションのステートを示すデバッグ ウィンドウ (呼び出しスタック、ソース ビューアー、逆アセンブリ、メモリ、レジスタ、その他のビュー、コンソール) が統合されています。標準的なデバッガー コマンドでブレークポイントを設定し、実行を制御できます。

2.4.8 カスタム IP ドライバーのサポートを追加する

ザイリンクスのハードウェア設計ツールで作成されるハードウェア プラットフォームのデータには、PL エリアで使用されるザイリンクス IP ブロックが取り込まれているため、これらのブロックのドライバーのサポートは、自動的にベアメタル BSP に含まれます。ハードウェア記述メタデータ ファイルを含むカスタム IP ブロックも、SDK へインポートされるハードウェア プラットフォームのデータに取り込み可能です。

SDK では、カスタム ドライバーおよびメタデータを含むソフトウェア リポジトリへのパスを指定し、ベアメタル BSP にそれらを含めることも可能です。

カスタム ドライバー ソース ファイルを管理および構築するためのライブラリ プロジェクトを作成し、ベアメタル BSP と共にライブラリ プロジェクトを使用するアプリケーションを構築することもできます。

ハードウェア プラットフォームが変更されると、カスタム IP ドライバーの設定が必要な場合があります。ソフトウェア ドライバーをカスタマイズするには、Tcl ファイルのほかにマイクロプロセッサ ドライバー定義 (MDD) ファイルを使用します。

設定が必要なドライバー パラメーターは、MDD ファイルで指定されます。`.h` または `.c` ファイルの生成手順は、Tcl ファイルに記述されます。詳細は、『プラットフォーム仕様フォーマット リファレンス マニュアル』(UG642) [参照 9] を参照してください。

2.4.9 アプリケーションを展開する

SDK でベアメタル アプリケーションを開発およびデバッグした後、ボード上で展開するアプリケーションのブート イメージを作成します。SDK には変更可能な FSBL 用のアプリケーション テンプレートが含まれているため、これらを使用して最終的な FSBL を生成します。FSBL、ベアメタル アプリケーション、および PL プログラム用ビットストリーム (オプション) が一緒になってブート イメージが生成されます。このブート イメージは、SDK Flash Writer でサポートするデバイスへプログラムできます。

ブート イメージ フォーマットの詳細は、第 3 章「ブートおよびコンフィギュレーション」を参照してください。

2.5 Linux アプリケーションの開発

ザイリンクス ソフトウェア設計ツールは、ベアメタル アプリケーションのほかに、Linux ユーザー アプリケーションの開発をサポートします。このセクションでは、Linux アプリケーション開発のフローについて簡単に説明します。

ザイリンクスのエンベデッド設計ツールを使用し、次を含むハードウェア プラットフォームのデータ ファイルを作成します。

- XML 形式のハードウェア記述ファイル - プロセッサ、ペリフェラル、メモリ マップ、その他のシステム データ情報が含まれている
- ビットストリーム ファイル - プログラマブル ロジック (PL) のプログラム データが含まれている (オプション)
- ブロック RAM メモリ (BMM) ファイル
- Zynq-7000 AP SoC の FSBL (第 1 段階ブートローダー) で使用される PL コンフィギュレーション データ

Linux は、オープンソースのオペレーティング システムです。ザイリンクスのオープン ソース ソリューションは、シングル プロセッサおよび対称型マルチプロセッシング (SMP) をサポートしています。ザイリンクスは、プロセッサ システム (PS) のペリフェラル用のドライバーを提供しています。PL 内にあるカスタム ロジック用のドライバーを追加できます。

ベアメタル ボード サポート パッケージの詳細は、『OS およびライブラリ資料コレクション』(UG643) [参照 8] に含まれる『スタンドアロン BSP』(UG652) を参照してください。

Linux の U-Boot ブートローダーに関しては、第 4 章「Linux」を参照してください。第 4 章には、より多くの情報を提供するザイリンクス オープン ソース Wiki へのリンク情報も記載されています。

ハードウェア プラットフォームのデータと Linux カーネルを使用することで、SDK で Linux ユーザー アプリケーションの開発、デバッグ、展開が可能です。SDK は、Linux カーネルのデバッグをサポートしていません。このセクションでは、Linux カーネルのコンフィギュレーションおよび構築プロセスについて説明していません。

SDK での Linux ユーザー アプリケーション開発の一般的な手順は次のとおりです。

1. 「Linux を起動する」
2. 「アプリケーション プロジェクトを作成する」
3. 「アプリケーションを構築する」
4. 「アプリケーションを実行する」
5. 「アプリケーションをデバッグする」
6. 「PL 内のカスタム IP ドライバーのサポートを追加する」
7. 「アプリケーションのプロファイルを行う」
8. 「Linux ファイル システムにアプリケーションを追加する」
9. 「Linux BSP (カーネルまたはファイル システム) を変更する」

図 2-6 に、Linux アプリケーション開発のフロー チャートを示します。

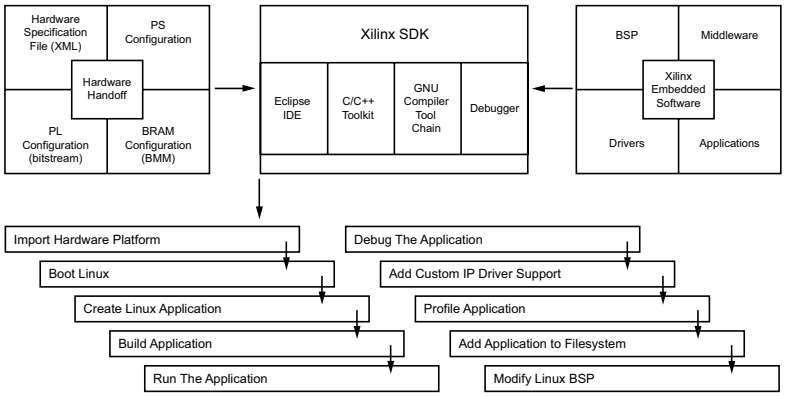

図 2-6 : Linux アプリケーション開発

以降のサブセクションでは、これらの手順について説明しています。SDK ツールの詳細および使用例は、『Zynq EDK コンセプト、ツール、およびテクニック ガイド』(UG873) [参照 7] を参照してください。

2.5.1 Linux を起動する

Linux は、ワーク フローに基づいていくつかの方法で起動できます。

- ブート イメージをフラッシュにプログラムし、ボードに電源投入またはリセットする
- FSBL をダウンロードして実行し、その後 U-Boot と Linux カーネルを起動する
- U-Boot を使用してイメージをロードし、実行する

Zynq-7000 AP SoC で Linux を使用する場合、SDK は AP プラットフォームをリモート Linux ホストとして見なし、その機能はファイル システム内のコンポーネントによって異なります。

フラッシュ メモリのオフセットは、メモリ タイプ (NAND、NOR、Quad-SPI) によって異なります。パーティションには、FSBL、U-boot、Linux カーネル、デバイス ツリー、RAMdisk、およびユーザー アプリケーションを含めることが可能です。

ブート プロセス中に FSBL が実行されて PS を設定し、その後 U-Boot が起動します。U-Boot Linux カーネル イメージをロードし、Linux を起動できます。実際のブート シーケンスやフラッシュ イメージの生成プロセスは、フラッシュの種類やその他の要件によって異なります。たとえば、FSBL を使用してカスタム ロジックを含む PL をコンフィギュレーションでき、U-Boot イメージ内に FSBL を含めることが可能です。

2.5.2 アプリケーション プロジェクトを作成する

SDK では、基本の「Hello World」や Dhrystone ブートローダー、FSBL アプリケーションからベンチマーキング アプリケーションまで、キットに含まれるサンプル プログラム用のテンプレート ベースのアプリケーション ジェネレーターを提供しています。アプリケーション ジェネレーターは、ザイリンクスの C または C++ アプリケーション ウィザードで起動されます。

空のアプリケーションを作成するか、あるいは既存アプリケーションをインポートして移植することも可能です。コード開発ツールには、エディター、検索、リファクタリングなどのツールのほかに、ベースの Eclipse プラットフォームや CDT プラグインで利用できる機能があります。

SDK には、ブート可能なイメージ (.bin/.mcs) を生成するユーティリティ (bootgen) があります。ブート イメージを作成するには、ユーザーが bootgen ツールへすべてのイメージおよびロード アドレスを提供する必要があります。

SDK ではフラッシュ デバイスでイメージを管理することも可能です。

2.5.3 アプリケーションを構築する

SDK アプリケーション プロジェクトは、ユーザー管理 (ユーザーが makefile を作成) または自動管理 (SDK が makefile を作成) が可能です。ユーザー管理のプロジェクトでは、ユーザーが makefile を管理してアプリケーションの構築を開始します。自動管理のプロジェクトでは、ソース ファイルが追加または削除されたときに SDK が必要に応じて makefile を更新し、変更が保存されて ELF が自動生成されると、ソース ファイルがコンパイルされます。Eclipse CDT の用語では、アプリケーション プロジェクトは Managed Make プロジェクトと呼ばれています。可能な場合は、SDK が使用するハードウェア プラットフォームおよび BSP に基づいてデフォルトの構築オプション (コンパイラ、リンカー、ライブラリ パス オプションなど) を推論して設定します。

2.5.4 アプリケーションを実行する

コンパイルされたアプリケーションをファイル システムへコピーしてアプリケーションを実行するには、SDK で実行コンフィギュレーションを作成します。Zynq-7000 AP SoC で Linux を動作させ、Linux 環境に SSH が含まれる場合は、sftp を使用して実行コンフィギュレーションが実行可能ファイルをファイル システムへコピーします。アプリケーションとの相互通信には STDIN および STDOUT を使用して端末表示が可能です。

次のコマンド シェルを使用してアプリケーションを実行することも可能です。

- sftp で実行可能ファイルをコピーする
- Linux で ssh を用いて実行可能ファイルを実行する

2.5.5 アプリケーションをデバッグする

SDK でアプリケーションをデバッグできます。SDK でデバッガー セッションのオプションを定義するデバッグ コンフィギュレーションを作成します。gdbserver が Linux 上でアプリケーションを実行し、SDK デバッガーが TCP を使用してアプリケーションと通信します。ウィンドウ (ビュー) によって、完全なデバッグ環境が提供されます。

このデバッグ方法は、CDT プラグインを追加する Eclipse ベースの IDE を使用した経験のあるユーザーにとっては使い慣れた環境となり、セッションのステートを示すデバッグ ウィンドウ (コール スタック、ソース ビューアー、逆アセンブリ、メモリ、レジスタ、その他のビュー、コンソール) が統合されています。標準的なデバッガー コマンドでブレークポイントを設定し、実行を制御できます。

2.5.6 PL 内のカスタム IP ドライバーのサポートを追加する

SDK では、PS 内のペリフェラル用の Linux BSP だけでなく、PL 内のカスタム IP 用の Linux BSP も生成できます。Linux BSP を生成する場合は、SDK がデバイス ツリーを生成します。これは、ブート時にカーネルへ渡されるハードウェア システムに関する情報を含むデータ構造です。デバイス ドライバーはカーネルの一部として、または個別モジュールとして提供され、利用できるハードウェア機能および有効な機能はデバイス ツリーで定義されます。

さらに、ユーザーは動的にロード可能なドライバーを追加できます。Linux ドライバーはこれらのドライバーをサポートします。詳細は、『Zynq-7000 All Programmable Soc コンセプト、ツール、およびテクニック ガイド』(UG873) [参照 7] を参照してください。

PL 内のカスタム IP は柔軟に設定可能であり、デバイス ツリーのパラメーターによって、システム内で利用できる IP と各 IP で有効なハードウェア機能の両方が定義されます。

Linux カーネルおよびブート シーケンスの詳細は、第 4 章「Linux」を参照してください。

2.5.7 アプリケーションのプロファイルを行う

Linux ユーザー アプリケーションのプロファイルには、アプリケーションを構築する際に -pg プロファイル オプションを使用します。ユーザー アプリケーションのプロファイルは、gprof ユーティリティに基づき、付随するビューアーに呼び出しグラフなどのデータを表示します。

ユーザー アプリケーション内のすべての実行コード、カーネル、割り込みハンドラー、およびその他のモジュールをプロファイルするために、SDK には呼び出しプロファイル機能の可視化をサポートする `OProfile` プラグインが備わっています。`OProfile` は、システム全体を監視する Linux 用のオープンソース プロファイラーです。サンプル データの収集には、カーネル ドライバーおよびデーモン (DAEMON) が必要です。

2.5.8 Linux ファイル システムにアプリケーションを追加する

コンパイルされたユーザー アプリケーションおよび必要となる共有ライブラリは、次のように Linux ファイル システムへ追加できます。

- Linux 環境に SSH が含まれている場合は、Zynq-7000 AP SoC 上で Linux が動作している間に `sftp` を使用してファイルをコピーできます。
- SDK の場合、リモート システム エクスプローラー (RSE) プラグインを利用し、ドラッグ アンド ドロップ操作でファイルをコピーできます。
- SDK 以外の場合、ファイル システム イメージを作成してフラッシュへ書き込む前に、ファイル システム フォルダーへアプリケーションとライブラリを追加します。

2.5.9 Linux BSP (カーネルまたはファイル システム) を変更する

第 4 章「Linux」に、Linux U-Boot ブートローダーの説明、およびその他の情報を提供するザイリンクス オープン ソース Wiki [参照 1] へのリンク情報が記載されています。

2.6 その他の情報

この章で説明した内容に関連する詳細は、第 1 章「はじめに」に記載している資料リストの中から適切な資料を参照してください。『Zynq-7000 All Programmable SoC コンセプト、ツール、およびテクニック ガイド』(UG873) [参照 7] の次のセクションも参照してください。

- 「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」
- 「ファブリックでの Zynq PS への IP 追加」

XILINX

第 3 章

ブートおよびコンフィギュレーション

3.1 概要

Zynq®-7000 All Programmable SoC デバイスのブートとコンフィギュレーションには、スタティック メモリのみを使用 (JTAG は無効) するセキュア モードと、JTAG またはスタティック メモリのいずれかを使用する非セキュア モードがあります。

- 開発およびデバッグには、主に JTAG が使用されます。
- デバイスのブートには、NAND、パラレル NOR、シリアル NOR (Quad-SPI)、およびセキュア デジタル (SD) フラッシュ メモリが使用されます。これらのブート モードの詳細は、『Zynq-7000 AP SoC テクニカル リファレンス マニュアル』(UG585) [参照 6] を参照してください。

プロセッサ システムは、2 段階のプロセスでブートされます。

- 内部 BootROM にステージ 0 ブート コードが格納されており、これによって一方の ARM® プロセッサおよびペリフェラルをコンフィギュレーションし、ブート デバイスの 1 つから第 1 段階ブート ローダー (FSBL) のブート コードのフェッチを開始します。プログラマブル ロジック (PL) は、BootROM でコンフィギュレーションされません。BootROM へは書き込みできません。
- FSBL ブート コードは、通常 1 つのフラッシュ メモリに格納されていますが、JTAG を介してダウンロードすることも可能です。BootROM コードが、選択したフラッシュ メモリからオンチップ メモリ (OCM) へ FSBL ブート コードをコピーします。OCM へロードされる FSBL のサイズは、最大 192 キロバイトに制限されています。FSBL が実行を開始した後、256 キロバイト全体が利用可能になります。
- FSBL でサポートされているもう 1 つのブート モードは、eMMC ブート モードです。このモードは、プライマリ ブート モード (ブート モード ピンで設定) が QSPI の場合にのみ有効です。このオプションは、小規模 QSPI フラッシュがあり、eMMC のような大規模フラッシュにその他すべてのパーティションを格納することを望む場合に使用します。この場合、QSPI フラッシュに FSBL を格納し、eMMC フラッシュにはその他すべてのパーティションを格納します。

FSBL ブート コードは完全にユーザーが制御でき、ユーザー ブート コードと呼ばれています。このため、ユーザー システムで求められるあらゆるブート コードを柔軟にインプリメントできます。

ザイリンクスは、ユーザーがカスタム変更できるサンプル FSBL ブート コードを提供しています。FSBL ブート コードには、プロセッシング システム (PS) 内のペリフェラルを初期化するコードが含まれます。FSBL 初期化シーケンスの詳細は、SDK で提供される FSBL コードを参照してください。ブート イメージには、プログラマブル ロジック (PL) のビットストリームを含めることができます。

PL がコンフィギュレーションされていなくても、PS は正常動作できる状態であるため、この段階での PL のコンフィギュレーションは必須ではありません。PL をブート/コンフィギュレーションするために FSBL ブート コードをカスタマイズし、イーサネット、USB、または STDIO などその他の PS ペリフェラルを使用できます。

注記 : DDR および SCU は BootROM では有効にできません。詳細は、『Zynq-7000 AP SoC テクニカル リファレンス マニュアル』(UG585) [参照 6] を参照してください。

3.2 ブート モード

次に示すブート モードを利用できます。

- PS マスター非セキュア ブート
- PS マスター セキュア ブート
- JTAG/PJTAG ブート

これらのブート モードの詳細は、『Zynq-7000 AP SoC テクニカル リファレンス マニュアル』(UG585) [参照 6] の「ブートおよびコンフィギュレーション」を参照してください。

3.3 ブート ステージ

Zynq-7000 AP SoC デバイスは、次に示すセキュア ブート プロセスと非セキュア ブート プロセスをサポートしています。

- 「ステージ 0 ブート (BootROM)」
- 「第 1 段階ブートローダー」
- 「第 2 段階ブートローダー (オプション)」

3.3.1 ステージ 0 ブート (BootROM)

詳細は、『Zynq-7000 AP SoC テクニカル リファレンス マニュアル』(UG585) [参照 6] の「BootROM」を参照してください。

27 ページの図 3-1 に、BootROM コードで FSBL を OCM へロードするフローを示します。

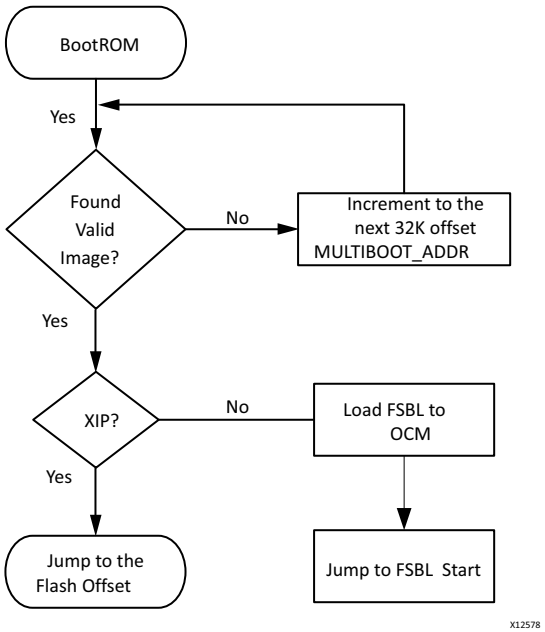

図 3-1 : ブート フロー

33 ページの「FSBL フォールバック機能」では、有効なイメージが検出されない場合の BootROM フローに関する詳細情報を提供しています。

3.3.2 第 1 段階ブートローダー

第 1 段階ブートローダー (FSBL) は、ブート後に開始します。FSBL は BootROM によって OCM へロードされます。

FSBL の役割は次のとおりです。

- ザイリンクスのハードウェア設計ツールで提供される PS コンフィギュレーション データを使用して初期化を実行する (43 ページの「Zynq PS のコンフィギュレーション」参照)
- ビットストリームを使用して PL をプログラムする (ビットストリームがある場合)
- 第 2 段階ブートローダーまたはベアメタル アプリケーション コードを DDR メモリへロードする
- 第 2 段階ブートローダーまたはベアメタル アプリケーションへ渡す

注記 : 第 2 段階ブートローダーまたはベアメタル アプリケーションへ渡す前に、FSBL は命令キャッシュを無効にしてキャッシュと MMU を利用不可にします (U-Boot は、これらが無効であることを前提として起動する)。

FSBL の初期化シーケンスの詳細は、SDK で提供される FSBL コードを参照してください。

29 ページの図 3-2 に FSBL フローの例を示します。

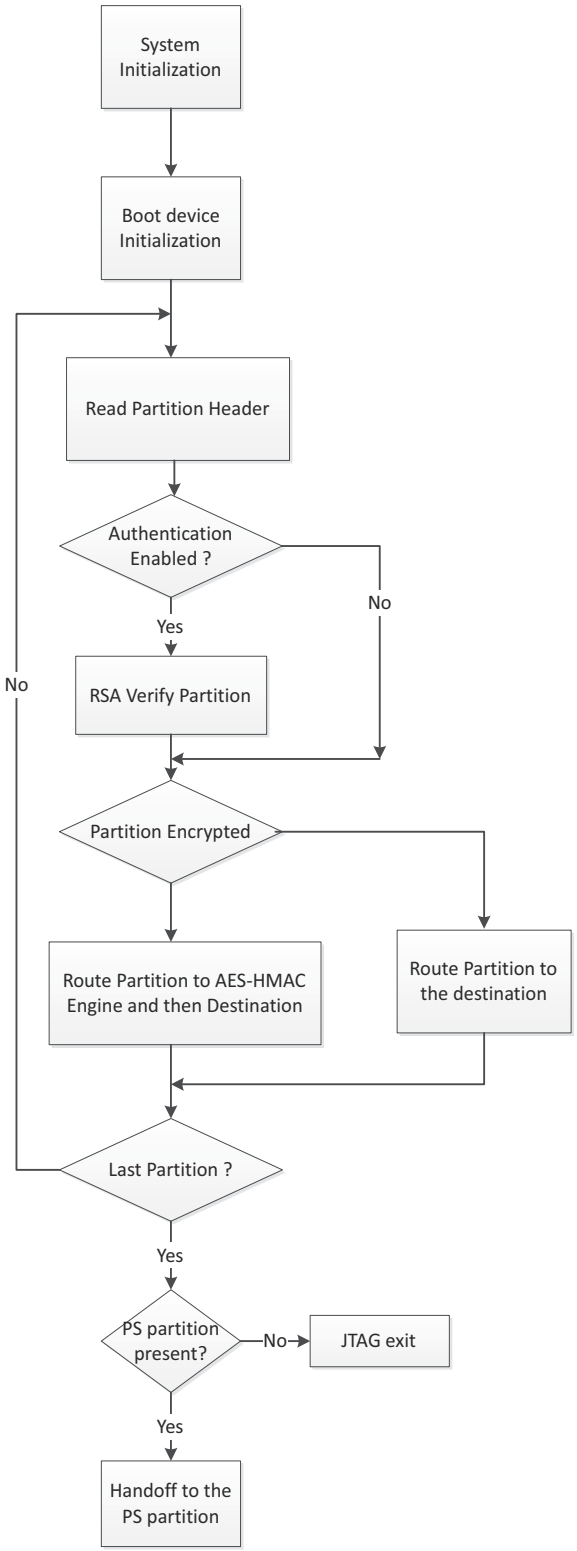

図 3-2 : FSBL フローの例

PL 用ビットストリーム、第 2 段階ブートローダーまたはベアメタル アプリケーションのデータ、および第 2 段階ブートローダー、Linux (またはその他のオペレーティング システム) 、あるいはベアメタル アプリケーションで使用されるその他のコードとデータは、フラッシュ イメージの各パーティションに分類されます。これらの構成図は、「3.4.2 ブート イメージのフォーマット」を参照してください。

FSBL は、パーティション ヘッダー テーブル内を検索し、ビットストリームおよび第 2 段階ブートローダー (またはベアメタル アプリケーション) のパーティションを見つけます。詳細は、付録 A 「Bootgen の使用」を参照してください。

これらのパーティションを含むブートイメージの生成方法の詳細は、「3.4 ブート イメージの生成」を参照してください。

Bootgen プログラムを使用し、FSBL にビットストリームとアプリケーションを結合します。SDK には [Create Zynq Boot Image] ウィザード オプション (図 3-3 を参照) があり、パーティション イメージの追加とブート可能なイメージの生成を行い、その後フラッシュへ格納できます。

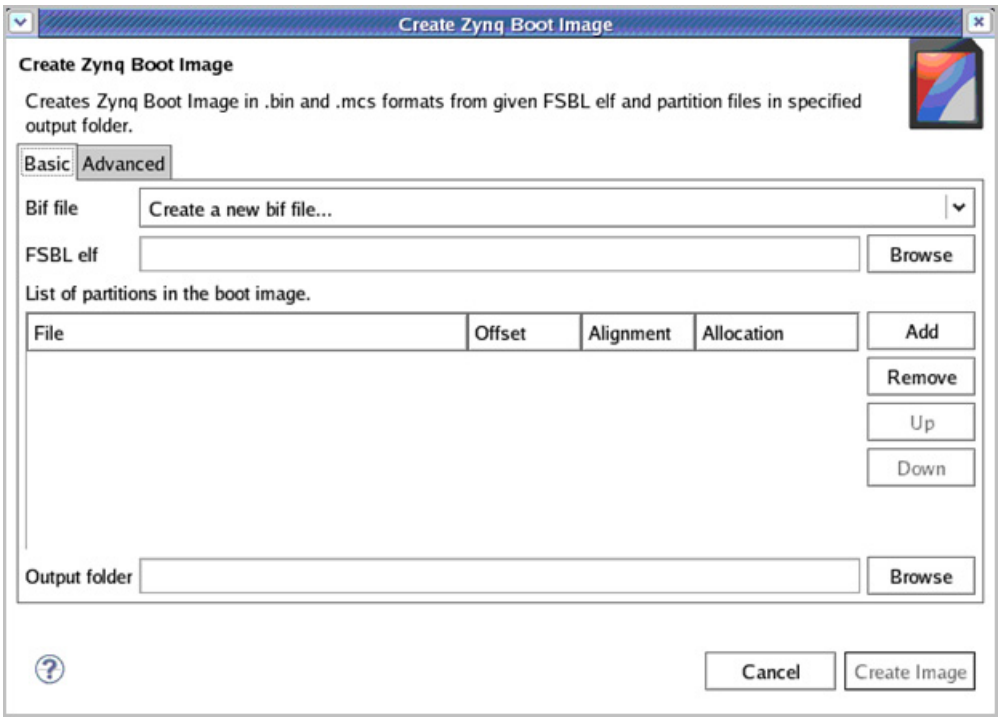

図 3-3 : [Create Zynq Boot Image] ウィザード

次の規則があります。

- 最初のパーティションは必ず FSBL ELF になり、次にビットストリーム パーティション、その次にアプリケーション ELF となります。
- ビットストリームはオプション選択です。FSBL は、BIF 内の順序で最初のアプリケーションに渡されます。

重要 : BIF ファイル内の順序は重要です。ビットストリームは、必ず FSBL の後のパーティションに含まれる必要があります。ビットストリームは必須ではなく、PL をプログラムする必要がある場合にのみ必要です。

FSBL は DDR をリマップしません。したがって、1Mb 以下の DDR は使用できません。

重要 : アプリケーション ELF には 1Mb 以上の実行アドレスがあります。

図 3-4 に FSBL フローの概略図を示します。

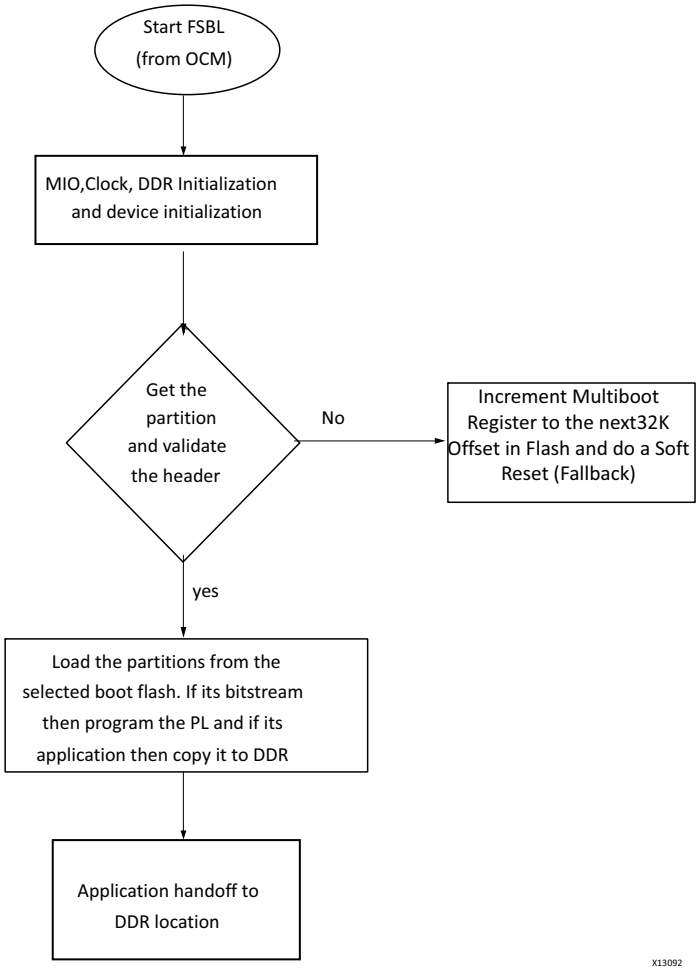

図 3-4 : FSBL フロー

eMMC フラッシュ デバイス

Zynq-7000 デバイスは、セカンダリ ブート ソースとして MLC および SLC コンフィギュレーションの eMMC フラッシュをサポートしています。FSBL は、eMMC からパーティションをロードできます。ただし、これはプライマリ ブート モード (ブート モード ピンで設定) が QSPI の場合にのみ可能です。このオプションは、小規模 QSPI フラッシュがあり、eMMC のような大規模フラッシュにその他すべてのパーティションを格納することを望む場合に使用します。この場合、QSPI フラッシュに FSBL を格納し、eMMC フラッシュにはその他すべてのパーティションを格納します。

このブート モードを有効にする手順は次のとおりです。

1. SDK で MMC_SUPPORT フラグを有効にし、FSBL を構築します。

   現在、FSBL イメージの構築 (`fsbl.elf`) には、eMMC サポートがあります。

2. 唯一のパーティションとして FSBL を含めたブート イメージを生成します (Bootgen を使用)。
3. QSPI フラッシュにブート イメージを格納します。
4. Bootgen を使用し、必要となるその他のパーティション (ビットストリームや U-Boot など) を含めたブート イメージを生成して eMMC フラッシュに格納します。
5. ブート モードを QSPI に設定します。

6. ボードの電源を一旦切って再度入れます。

BootROM が起動し、QSPI フラッシュから OCM へ FSBL がロードされて FSBL へ制御が渡されます。そして、FSBL が eMMC デバイスからその他すべてのパーティションを DDR へロードし、制御がアプリケーションへ渡されます。

この場合、FSBL は設定されているプライマリ ブート モード QSPI (ボード上のブート モード ピンで設定) を無視し、eMMC からその他のパーティションをロードします。

QSPI フラッシュに FSBL と U-Boot をロードする場合、FSBL で `MMC_SUPPORT` フラグが有効になる必要はありませんが、U-Boot 自動コンフィギュレーション ファイルを更新し、U-Boot に対して eMMC フラッシュから残りのパーティションをロードするように示す必要があります。

この場合、FSBL が U-Boot を DDR へロードし、制御を U-Boot へ渡します。

U-Boot は eMMC フラッシュから残りのパーティションをロードします。この場合の留意点として、U-Boot は RSA 認証をサポートしていないため、eMMC フラッシュにあるパーティションは RSA によって認証されません。

FSBL コンパイル フラグの設定

コンパイル フラグは、SDK FSBL プロジェクトで [C/C++ Build] の [Settings] を使用して設定できます (図 3-5 を参照)。

注記 : これらのフラグを含めるために FSBL ソース ファイルやヘッダー ファイルを変更する必要はありません。

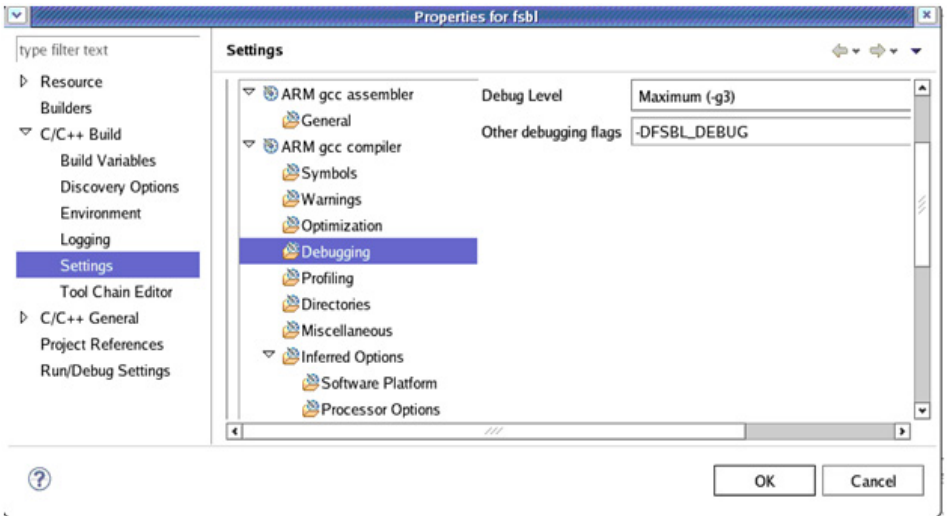

図 3-5 : SDK FSBL のプロパティ設定

FSBL コンパイル フラグは次のとおりです。

FSBL_DEBUG	ログやメッセージの プリントを可能にします。
FSBL_DEBUG_INFO	レジスタやパーティション ヘッダー ダンプなどの詳細ログを取得します。
NON_PS_INSTANTIATED_BITSTREAM	ビットストリームに PS コンポーネントが含まれていない場合にこのフラグを設定します。これにより、FSBL はレベル シフタを有効にしません。
RSA_SUPPORT	FSBL の認証機能を有効にします。
MMC_SUPPORT	FSBL の MMC サポートを有効にします。このフラグが設定されている場合、(ブート モード ピンで設定される) プライマリ ブート デバイスの代わりに eMMC デバイスからすべてのパーティションを読み出します。

FSBL フォールバック機能

エラー状態から回復するために、FSBL はフォールバック機能を使用します。つまり、別のブート可能なイメージ (最初に生成された正常なゴールデン イメージ) がフラッシュ メモリ内にある場合は、BootROM がそれらをロードできるようになります。FSBL は、BootROM が次の有効なイメージをロードできるように、マルチブート レジスタを更新してソフト リセットします。

eFUSE 内の AES キーを使用するセキュア ブートの場合は、ソフト リセットを行わずに FSBL によってフォールバック プロセスが実行されます。以降のサブセクションで詳細を説明します。

eFUSE の詳細は、『OS およびライブラリ資料コレクション』(UG643) [参照 8] に含まれる『LibXil SKey for Zynq-7000 AP SoC Devices』(UG996) を参照してください。

非セキュアなフォールバック プロセス

FSBL の非セキュア フローでは、次の動作が行われます。

- パワー オン リセット (POR) 後、BootROM が起動して Image 1 Boot ヘッダーを検証します。
  - エラーがない場合、BootROM は FSBL へ制御を渡します。そして FSBL がイメージ内のその他のパーティションをロードします。
  - ブート ヘッダーにエラーがある場合、BootROM はフォールバック機能を利用して次の有効なイメージを検索します。図 3-6 の例では、BootROM が Image 2 ブート ヘッダーを検証し、エラーがない場合には Image 2 を FSBL へ渡し、FSBL が Image 2 の残りのパーティションを処理します。
  - 非セキュア イメージの場合、FSBL やその他のイメージのエラーは認識されません。

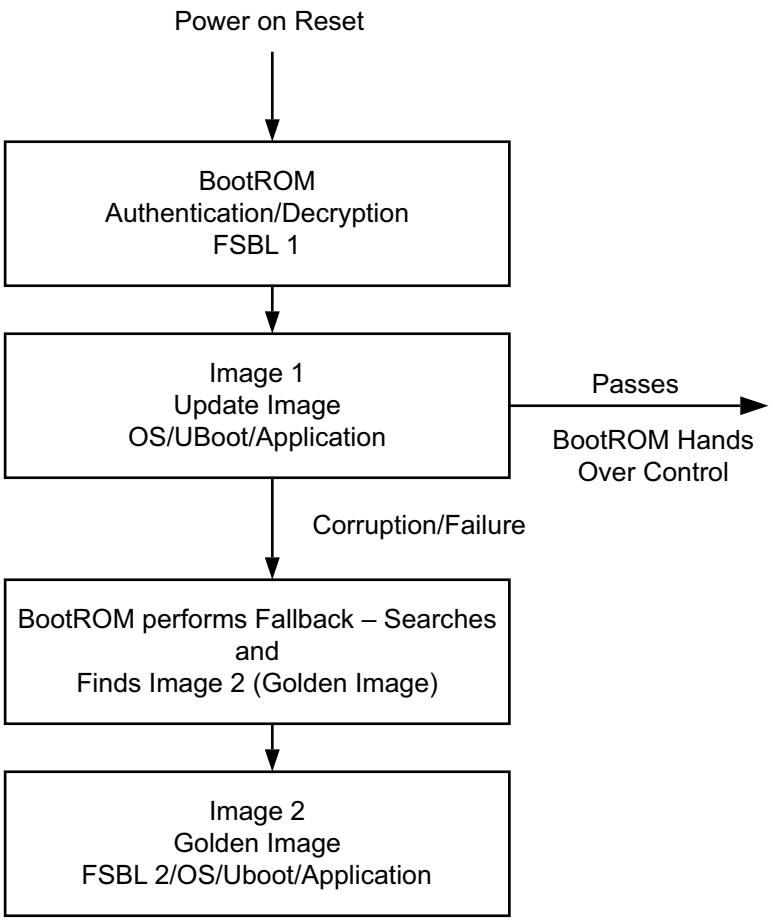

図 3-6 : POR フォールバック

図 3-7 に、非セキュアなフラッシュ イメージ フォーマットを示します。

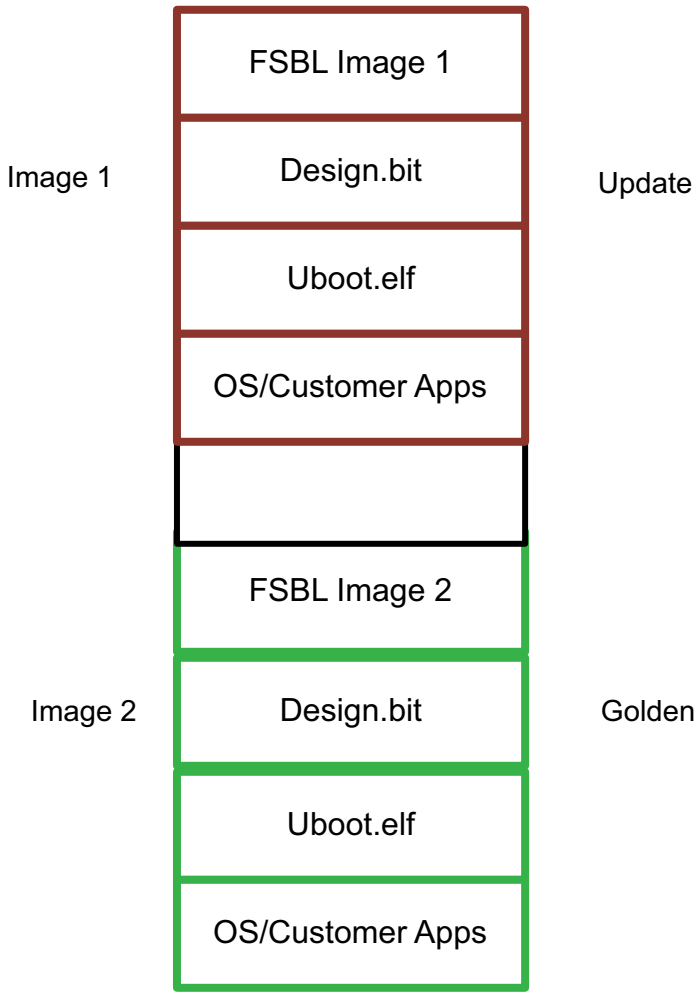

図 3-7 : 非セキュア フォールバック イメージのフォーマット

RSA のみを使用したフォールバック フロー

RSA 認証機能が有効な非セキュア フォールバックでは、次の動作が行われます。

- POR 後、BootROM が起動して Image 1 の Boot Header を検証します。
- Boot Header にエラーがない場合、BootROM は制御を FSBL へ渡します。そして FSBL が残りのパーティションを認証してロードします。
- Boot Header または FSBL イメージにエラーがある場合、BootROM はフォールバック機能を使用して次の有効なイメージを検索します。この例の Image 2 では、BootROM が Image 2 のブート ヘッダーを検証します。ブート ヘッダーの検証が完了すると、BootROM は Image 2 の FSBL を認証して FSBL へ制御を渡します。
- このとき、ビットストリーム、U-Boot、または OS にエラーがあると、FSBL 認証が失敗となり、システムのソフト リセットによってフォールバックが行われ、BootROM がゴールデン イメージを検索します。

図 3-8 に、RSA のみを使用したフォールバック フローを示します。

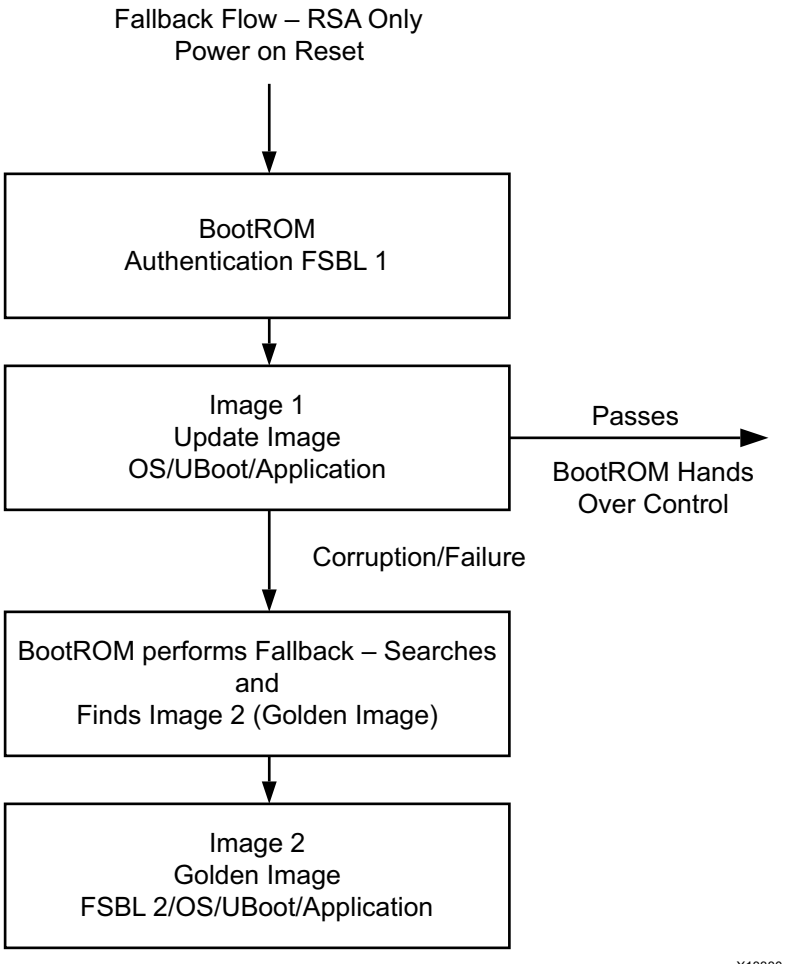

図 3-8 : RSA のみを使用したフォールバック フロー

図 3-9 に、非セキュアなフラッシュ イメージ フォーマットを示します。

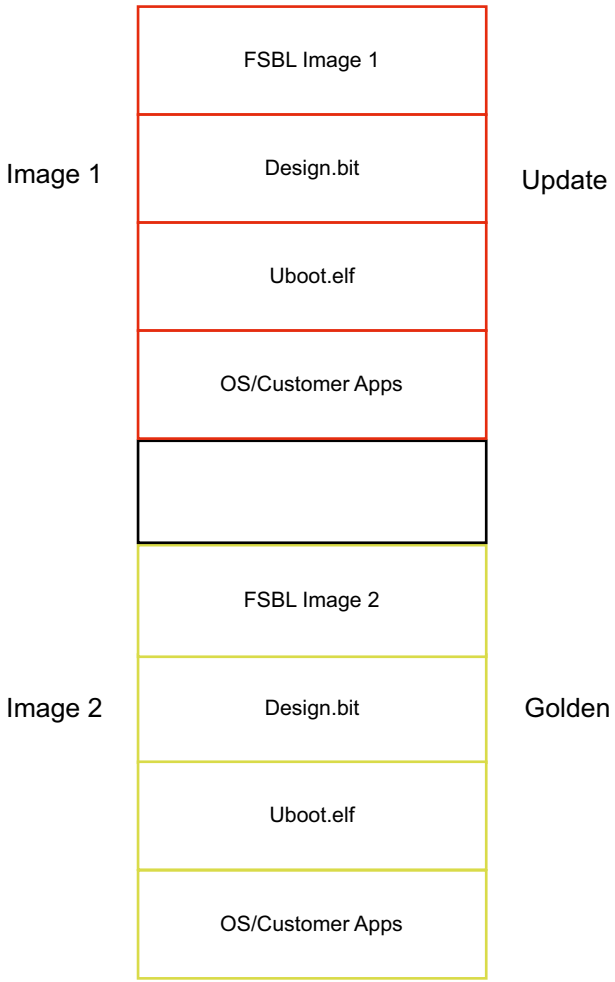

図 3-9 : RSA のみを使用するフォールバック パーティション

BBRAM を使用するセキュアなフォールバック フロー

BBRAM を使用するセキュアなフォールバック フローでは、次の動作が行われます。

- BootROM が起動し、RSA が有効な場合には FSBL1 を復号化して認証します。
  - 認証成功後、BootROM は制御を FSBL へ渡します。そして FSBL がその他のパーティションをロード、復号化、および認証 (有効の場合) し、制御を OS、U-Boot、および/またはアプリケーションへ渡します。
  - Image1 のブート ヘッダーにエラーがある場合、BootROM は Image2 を検索して FSBL を復号化し、Image2 の FSBL へ渡します。そして FSBL が残りのパーティションの復号化および認証 (有効な場合) をすべて行い、U-Boot、OS、またはスタンドアロン アプリケーションへ渡します。このプロセスでは、FSBL がエラーを含むイメージを検出すると、フォールバックを開始します。

図 3-10 に、BBRAM を使用するセキュアなフォールバック フローを示します。

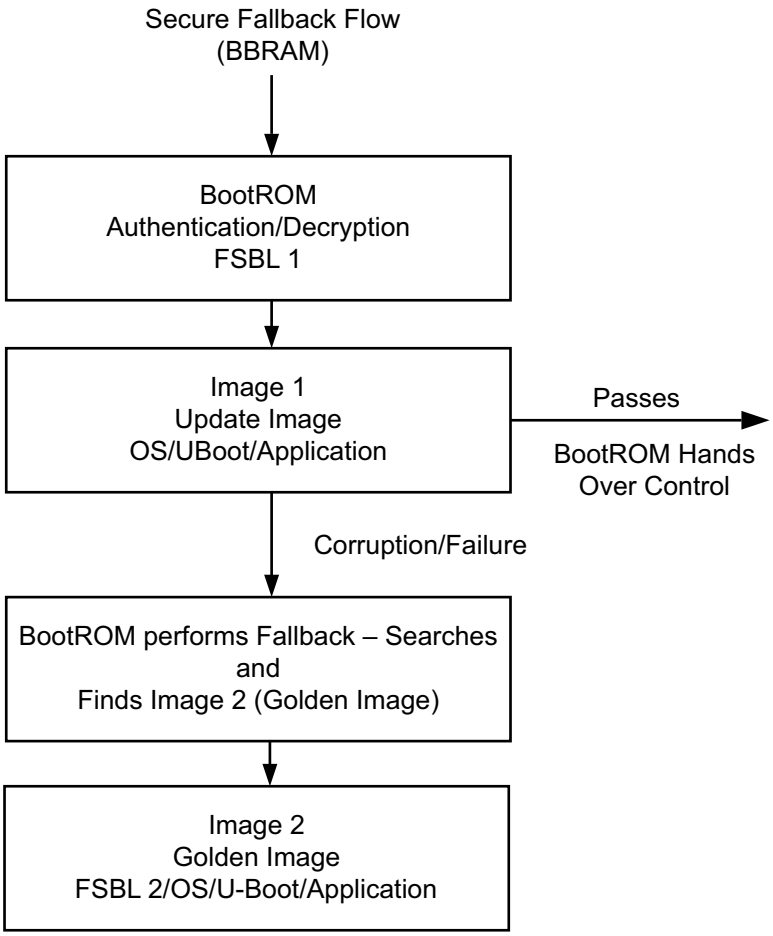

図 3-10 : BBRAM を使用するセキュアなフォールバック

図 3-11 に、BBRAM を使用するセキュア ブートのフラッシュ パーティションを示します。

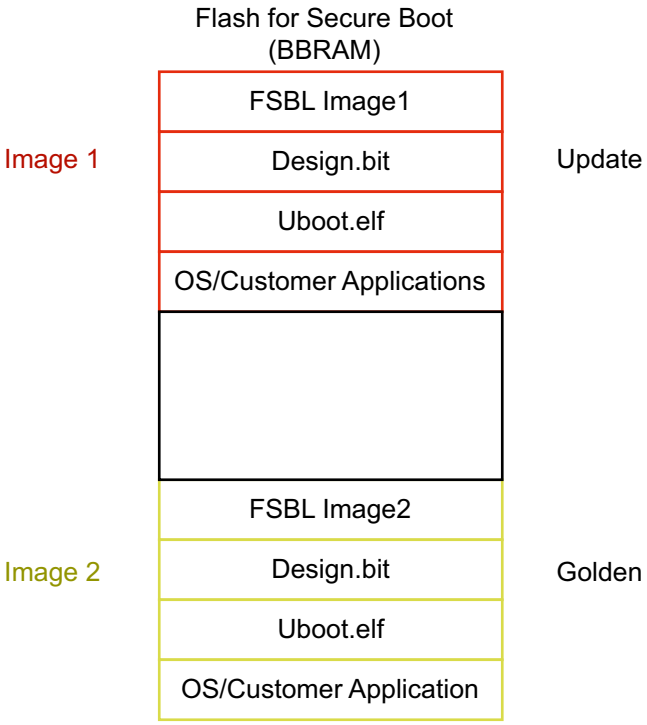

図 3-11 : フラッシュ BBRAM のパーティション

eFUSE を使用するセキュアなフォールバック フロー

パワー オン リセット (POR) 時に eFUSE を使用するセキュアなフォールバック フローは次のとおりです。この場合、FSBL はソフト リセットを行わずにフォールバックを実行します。

- BootROM が起動し、FSBL* を復号化して認証 (有効な場合) を行い、FSBL へ制御を渡します。
- その後 FSBL* が次を実行します。
  - 暗号化されたフォールバックのシナリオに従います。
  - その他のパーティションが見つからない場合は、フォールバックを行い、次の有効なイメージを検索します。
    - Image 2 を検索して Image 2 のブート ヘッダーを検証します。
    - 有効な場合は、Image 2 の FSBL を飛ばして Image 2 内のその他のパーティションを処理し、その後 Image 2 のアプリケーションに制御を渡します。

Image 2 にヘッダー エラーがある場合は、次を実行します。

- FSBL* がエラー メッセージを出して Image 2 にエラーがあることを示します。(Image 1 の FSBL がファイルを制御し、フォールバックを実行して次のイメージを検索します。)
- FSBL* が Image 3 を検索した後、次を実行します。
  - ブート ローダーを検証します。
  - RSA が有効の場合にはファイルを認証します。
  - Image 3 の FSBL を飛ばして Image 3 の残りのパーティションを処理します。

推奨 : セキュア イメージには認証機能を使用してください。

図 3-12 に、eFuse を使用するセキュアなフォールバック フローを示します。

これは、暗号化を使用するフォールバックに使用される FSBL です。

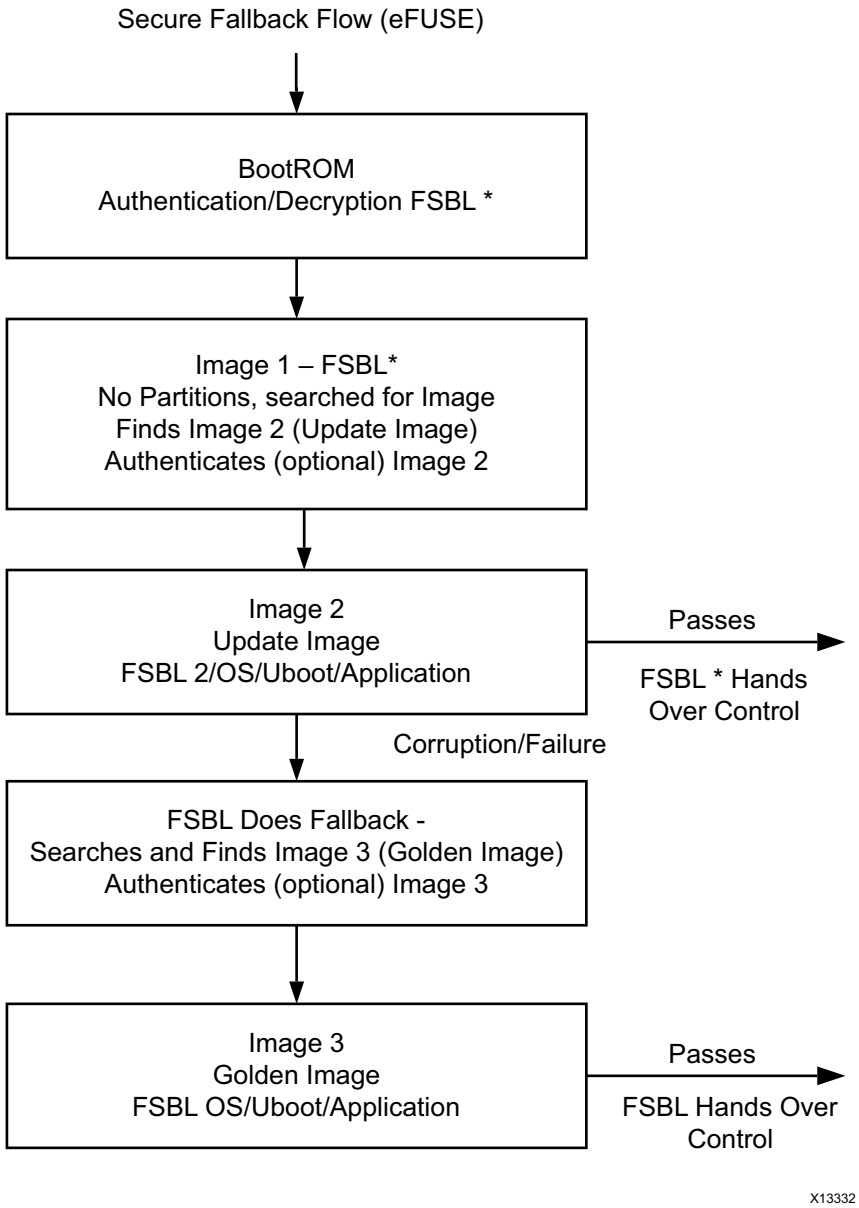

図 3-12 : eFUSE を使用するセキュアなフォールバック フロー

注記 : PL の eFUSE に AES キーが格納されている場合のセキュアなフォールバック フローは、その他のフローと異なります。RSA 認証はオプションです。

- FSBL* (Image 1 の FABL) が認証に失敗した場合、BootROM はセキュア ロックダウン モードになります。したがって、Image 1 にエラーがないことをユーザーが確認する必要があります。
- Image 1 のブート ヘッダーが有効でない場合、BootROM は Image 2 へ移動し、Image 2 の FSBL が起動します。

推奨 : セキュア モードでは、SRST ではなく、割り込み用にウォッチドッグ タイマーを設定することを推奨しています。GPIO を介して POR を行うようにウォッチドッグ割り込みを配線できます。

図 3-13 に、eFUSE を使用するセキュア ブートの FSBL* パーティションを示します。

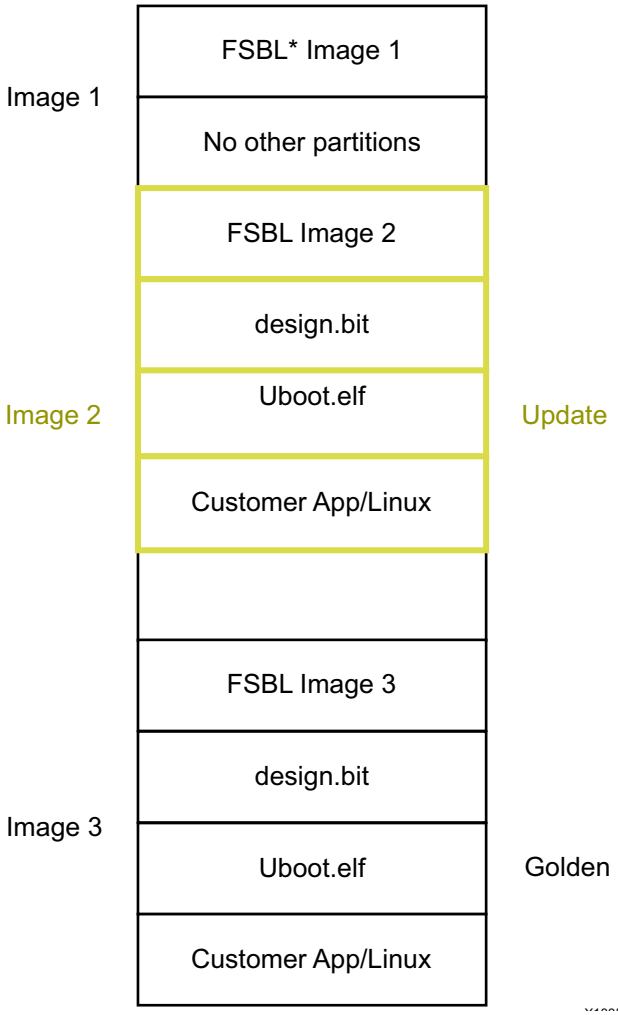

図 3-13 : eFUSE を使用するセキュア ブートのフラッシュ パーティション

FSBL マルチブート

マルチブートは、現在実行中のバージョンとは異なる FSBL をロードすることを望む場合に使用します。たとえば、FSBL のあるバージョン (セルフテストや診断を実行する任意のイメージ) を実行した後、実際のアプリケーションへ移動します。

この場合、診断を実行するイメージを実行した後、ユーザーは実際のアプリケーションを含むロード ファイルのシーケンス番号を使用してマルチブート レジスタをアップデートし、ソフト リセットを発行できます。

マルチブートのシナリオ :

- 複数のイメージを使用して 1 つのデバイスの機能をセットアップできる
- 指定時間にデバイスが実行しなければならない機能に基づいて、ユーザーがイメージを選択できる

システムがソフト リセットによってブート アップすると同時に、BootROM は最初のロード可能なイメージの代わりにマルチブート レジスタを読み出し、そのロード可能なイメージへ移動します。

eFUSE 内の AES キーを使用するセキュア ブートの場合、ユーザーがマルチブートのシナリオを実行する必要があります (ソフト リセットに依存しない)。

NAND ブート モード

NAND ブート モードでマルチブートを使用する場合は、FSBL に提供されている `calculate_multiboot()` API を呼び出します。この API がマルチブート アドレスを計算します。

シーケンスは次のとおりです。

1. ブートストラップ ピンを使用してブート モードを NAND に設定します。
2. FSBL に提供されている API を、マルチブートを起動するアプリケーションに追加します。
3. マルチブートを起動するアプリケーションから calculate_multiboot() API を呼び出し、マルチブート アドレスを計算します。
4. マルチブート アドレス レジスタのマルチブート アドレスをアップデート (『Zynq-7000 AP SoC テクニカル リファレンス マニュアル』(UG585) [参照 6] のセクション 6.4.6 を参照) し、ソフト リセットをトリガーします。

QSPI ブート モード

QSPI ブート モード (QSPI デバイスが 128Mb 以上) でマルチブートを使用する場合は、128Mb 以下のメモリに収まるように複数イメージを配置します。

このモードを有効にするには、128Mb 以下のメモリに収める必要があるため、(FSBL+U-Boot) のみをイメージに含めます。そして、128Mb を超えるその他のパーティションは、U-Boot で管理してください。この場合、`zynq_common.h` をアップデートし、必要なパーティションをロードするためのコマンドを追加します。詳しい使用法および例については、ザイリンクスの Wilki ページ [参照 1] を参照してください。

FSBL のフック

FSBL のフックは、定義済み関数 (たとえば、ビットストリームをロードした後に PI IP を初期化) を簡単にプラグインできるようにします。FSBL のフック用関数は、`fsbl_hook.c` ファイル内にあります。

`fsbl_hook.c` ファイルには次の関数が含まれています。

- `FsblHookBeforeBitstreamDload` : この関数は、PL ビットストリームがダウンロードされる前に呼び出されます。これは、カスタマイズした任意のコードです。この関数で、ビットストリームをダウンロードする前にカスタム コードを追加できます。
- `FsblHookAfterBitstreamDload` : この関数は、アプリケーションへ渡す前に呼び出されます。アプリケーションへ渡す前に実行する必要があるカスタム動作を追加できます。
- `FsblHookBeforeHandoff` : この関数は、FSBL がアプリケーションへ渡す前に呼び出すフックです。このルーティンへ渡す前に実行すべきカスタム コードを追加できます。
- `FsblHookFallback` : この関数は、FSBL がフォールバックするときに呼び出されます。フォールバックが発生するときに、メッセージ表示、エラー ログ、あるいは任意の動作を行うなど、いずれかのカスタム コードを追加できます。

これらのフック機能を使用することによって、FSBL のフロー シーケンスにアプリケーション特有のカスタム コードを簡単にプラグインできます。

DDR ECC の有効化

DDR の ECC サポートを有効にする機能です。

- XPS で、この機能を有効にします。
- Vivado IP インテグレーター Zynq-7000 ブロック図の場合、DDR コンフィギュレーション ページを使用します。

この機能が有効に設定されると、EEC 機能を有効にするために FSBL が DDR を初期化します。

FSBL は、ECC エラーの管理をサポートしていません。したがって、ユーザーがプログラム内にエラー対応措置を講じる必要があります。

FSBL は DDR をリマップしないため、DDR は 1Mb から開始します。このため、アプリケーション プログラムは 1Mb 以上の DDR の使用を考慮する必要があります。1Mb より小規模な DDR を使用しなければならない場合は、ECC 機能をサポートするために、DRR を初期化する必要があります。

セキュア ブートのサポート

FSBL は、次のセキュア ブート機能をサポートします。

- AES (Advanced Encryption Standard) 暗号
  - 256 ビットのキーを使用する AES-CBC
  - チップ上の eFuse または BBRAM (Battery-Backed RAM) のいずれかに格納される暗号化キーを使用
- HMAC (Keyed-Hashed Message Authentication Code)
  - SHA-256 認証エンジン (FIPS180-2)
- RSA 公開キー認証
  - 2048 ビットの公開キー

FSBL は、ユーザーが有効にしたセキュア機能に基づくセキュア モードで動作します。

RSA 認証が有効の場合、FSBL は復号化または実行される前に公開キーを使用して認証されます。RSA 認証を有効に設定するには、ブート可能なイメージを生成する際に Bootgen にオプションとしてこの機能を与えます。パーティション ヘッダーに提供されているコンフィギュレーションに基づいて、FSBL はイメージを認証して復号化します。

RSA 認証の詳細は、『Zynq-7000 AP SoC テクニカル リファレンス マニュアル』(UG585) [参照 6] を参照してください。

Zynq PS のコンフィギュレーション

Zynq-7000 コンフィギュレーション インターフェイスを使用し、ザイリンクスのハードウェア設計ツールが MIO および SLCR レジスタの初期化用コードを生成します。プロジェクト ディレクトリには、次のファイルがあります。

- `ps7_init.c` および `ps7_init.h` ― CLK、DDR、および MIO を初期化する場合に使用できます。`ps7_init.tcl`による初期化は、`ps7_init.c` 内のコードで実行される初期化と同じです。
- `ps7_init.tcl` ― CLK、DDR、および MIO を初期化する場合に使用できます。`ps7_init.tcl` による初期化は、`ps7_init.c` 内のコードで実行される初期化と同じです。

  注記 : XMD を使用してアプリケーションをデバッグする場合は、Tcl ファイルが有用です。たとえば、ps7_init.tcl ファイルの実行後、ユーザー アプリケーションを DDR へロードしてデバッグを行うことができます。この場合、FSBL を実行する必要はありません。
- `ps7_init.html` ― 初期化データが含まれています。

注記 : PL ビットストリームと初期化データの同期は XPS で維持されます。これらの設定を手動で変更することは、推奨していません。

3.3.3 第 2 段階ブートローダー (オプション)

第 2 段階ブートローダーは、ユーザーが設計するオプションです。第 2 段階ブートローダーの例は、「4.4 U-Boot」を参照してください。

3.4 ブート イメージの生成

ROM やフラッシュ メモリのプログラムに最適な 1 つのブート イメージ ファイルを作成するためのユーティリティ プログラム (Bootgen) があります。このプログラムは、必要なブート ヘッダーを構築して後続パーティションを定義するテーブルを追加し、パーティションの入力データ ファイル (ELF ファイル、FPGA ビットストリーム、その他のバイナリ ファイル) を処理することによってブート イメージを生成します。各パーティションに対して特定のデスティネーション メモリ アドレスを割り当てたり、アライメント要件を与える機能があります。また、各パーティションにおける暗号化、認証またはチェックサムをサポートします。

このユーティリティは、Boot Image Format (BIF) ファイルというコンフィギュレーション ファイル (*.bif) で駆動されます。BIF ファイルには、ブート イメージの入力ファイルのほかに、アドレス指定する際のオプションの属性やオプションの暗号化、認証、またはチェックサムなどが含まれます。BIF ファイルのフォーマットについては、この後説明します。

上級認証フローでは、Bootgen を使用してオフラインで符号化できる中間のハッシュ ファイルを出力できます。それ以外の場合、Bootgen は提供されたプライベート キーを使用し、ブート イメージ内に含まれる認証証明 (Authentication certificates) を符号化します。

ブート イメージのフォーマットは、ハードウェア要件とソフトウェア要件の両方に対応します。ブート イメージ ヘッダーは、シングル パーティション (通常は FSBL) をロードする Zynq-7000 BootROM ローダーで必要です。その他のブート イメージは、FSBL によってロードされて処理されます。

ユーティリティの詳細は、付録 A 「Bootgen の使用」を参照してください。

3.4.1 Bootgen コマンドの例

簡単なコマンド ライン例を示します。

```
bootgen -image myDesign.bif -o i myDesignImage.bin
```

この例では、Bootgen が、ブート ヘッダーとそれに続いて `myDesign.bif` に記述されたデータ ファイルから作成されたデータ パーティションを含む `myDesignImage.bin` というファイルを生成します。

3.4.2 ブート イメージのフォーマット

ブート イメージの構成内容は次のとおりです。

- BootROM ヘッダー
- FSBL イメージ
- 1 つまたは複数パーティションのイメージ
- 未使用空間 (該当する場合のみ)

図 3-14 にブート イメージ フォーマットのレイアウトを示します。

注記 : FSBL の暗号機能はオプションです。

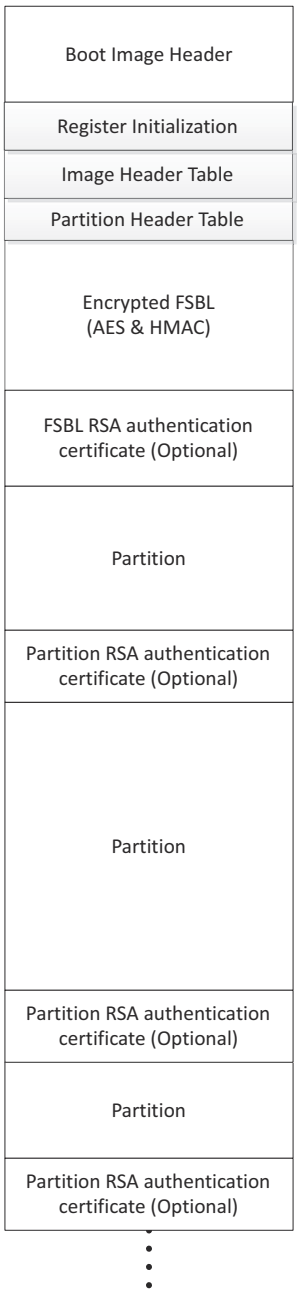

図 3-14 : Zynq セキュア ブート イメージのフォーマット

3.4.3 認証証明

認証証明は、認証済みの各パーティションの末尾に追加されます。すべての整数は、リトルエンディアン方式で格納され、2048 ビットの係数が含まれています。

表 3-1 に、認証証明のオフセット値、サイズ、フィールド、および注記を示します。

表 3-1 : 認証証明

オフセット	長さ	フィールド	注記
0x0	0x4	認証証明のヘッダー	表 3-2 を参照
0x04	0x4	認証証明の長さ	
0x008	0x3C	ユーザー定義のフィールド	56 バイト
0x44	0x100	PPK 係数	640 バイトのリトルエンディアン
0x144	0x100	PPK 係数の拡張	
0x244	0x04	PPK 指数	
0x248	0x3C	パディング	480 の 0
0x284	0x100	SPK 指数	640 バイトのリトルエンディアン
0x384	0x100	SPK 係数指数	
0x484	0x04	SPF 係数の拡張	
0x488	0x	パディング	480 0s
0x4C4	0x	SPK シグナチャ	
0x5C4		パーティション シグナチャ	

FSBL の負荷を軽減するために、BootGen はべき乗剰余を求めるためのモンゴメリ リダクションで使用される係数の拡張をあらかじめ計算します。これらの値は、係数フィールドの後の認証証明の中に格納されます。表 3-2 に、認証証明のビット、フィールド、および値を示します。

表 3-2 : ビット認証証明ヘッダー

ビット	フィールド	値
31:16	予約	0s
15:14	認証証明のフォーマット	00 : PKCS #1 v1.5
13:12	認証証明のバージョン	00 : 現在の AC
11	PPK キー タイプ	0 : ハッシュ キー
10:9	PPK キー ソース	0 : eFUSE
8	SPK 有効	1 : SPK イネーブル
7:4	公開強度	0 : 2048
3:2	ハッシュ アルゴリズム	0 : SHA256
1:0	公開アルゴリズム	1 : RSA

図 3-15 に、Zynq-7000 AP SoC Linux のブート イメージ パーティションの例を示します。

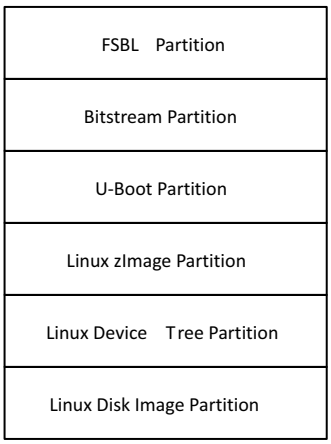

図 3-15 : Zynq-7000 AP SoC Linux のブート イメージ パーティションの例

3.5 BootRom ヘッダーのフォーマット

BootROM ヘッダーのフォーマットは、『Zynq-7000 AP SoC テクニカル リファレンス マニュアル』(UG585) [参照 6] の表 6-3 を参照してください。

XILINX

第 4 章

Linux

4.1 概要

ザイリンクスが提供する Zynq®-7000 AP SoC Linux は、オープン ソース ソフトウェア (kernel.org のカーネル) をベースにしています。ザイリンクスは、Linux カーネルのザイリンクス固有部分 (ドライバーおよびボード サポート パッケージ BSP) のサポートを提供しています。

Embedded Linux フォーラム ([参照 1]) でも、Linux のサポートを提供しています。多くのオープン ソース プロジェクトと同様に、ザイリンクス Zynq-7000 に特定されない部分の Linux に関しては、オープン ソース メーリング リストの利用を推奨しています。

ザイリンクス Zynq Linux およびザイリンクス オープン ソース プロジェクトの詳細は、ザイリンクスの オープン ソース Wiki サイト ([参照 1]) を参照してください。

ザイリンクスは、公開 Git サーバーを使用して Linux カーネル、ザイリンクス ボード用 BSP、および特定 IP 用ドライバーを提供しているため、サードパーティ企業はザイリンクス ハードウェア用のエンベデッド Linux ディストリビューションを構築できます。Git サーバーを利用することによって、ディストリビューションを購入するというよりも独自の Linux システムを開発することが可能です。

注記 : すべてのザイリンクス IP がサポートされているわけではありません。

4.2 Git サーバーと Gitk コマンド

ザイリンクスは、Git を使用することで Linux オープン ソース コミュニティとの相互作用を容易にしています。たとえば、パッチは自動的にカーネルのメインラインへプッシュされます。また、ユーザーが Git ツリーからパッチを取得し、それをカーネルへ適用することも可能です。さらに、Git はコンフィギュレーションの管理も行うため、カーネルの変更はユーザーがすべて把握できます。

- Git ツリーは http://git.xilinx.com にあり、リポジトリのスナップショット取得方法も含まれています。ウェブサイトでコードを参照できます。

公開リポジトリのメイン ブランチがマスターとなります。ここには、最も信頼性が高く、ザイリンクスによる検証が完了したコードのみが含まれます。

- Git の概要 : http://git-scm.com
- Git に関する資料 : http://git-scm.com/documentation
- Git のダウンロード : http://git-scm.com/download

gitk ツールは、Git ツリーをグラフィカルに表示するためのツールです。これは、ツリー内のブランチ検索に有効です。Git をインストールし、コマンドラインから gitk を使用して実行できます。

図 4-1 にこのツールのスクリーン ショットを示します。

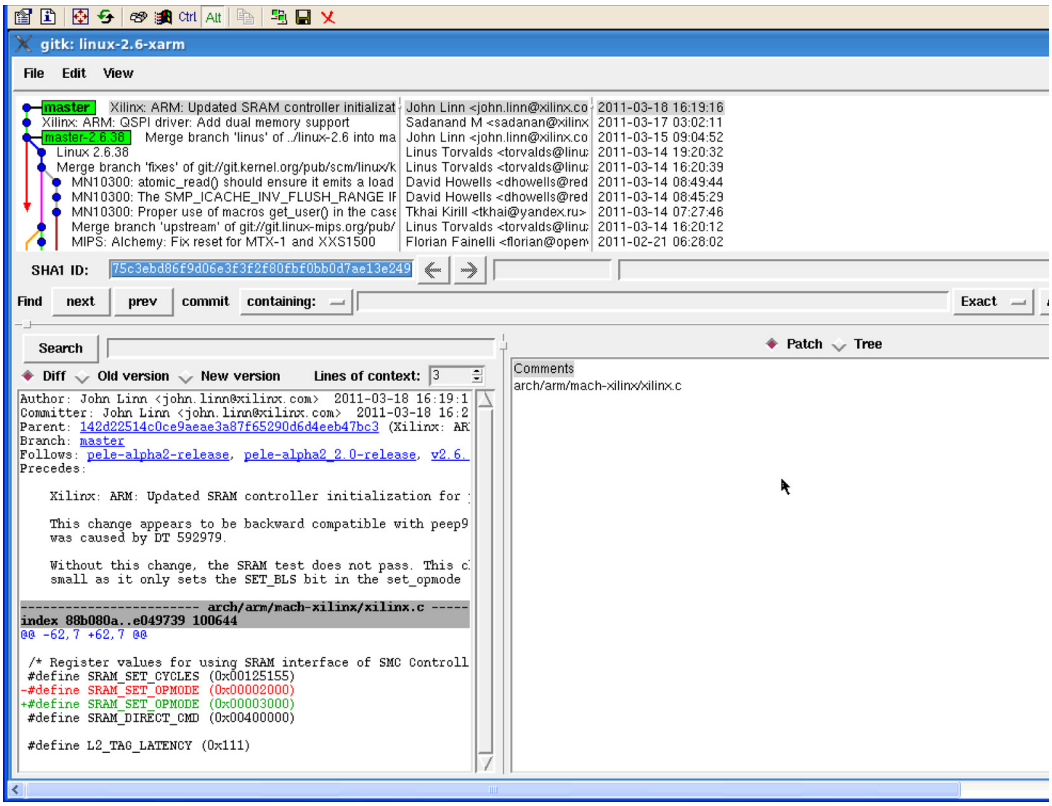

図 4-1 : Gitk

4.3 Linux BSP に含まれるもの

4.3.1 カーネル

Linux カーネルは、システムのボードとドライバー向けに生成されたボード サポート パッケージ (BSP) と共に含まれます。カーネルにはファイル システムが必要となるため、カーネルを起動するためのファイル システムを作成する必要があります。

注記 : カーネルが含まれるディレクトリをカーネル ツリーといいます。ここでは、Linux カーネルのディレクトリ構造を理解していることを前提に説明しています。

50 ページの図 4-2 には、さまざまな機能が異なる層とどのように関連しているかをわかりやすく示すために、Linux カーネルの上位図を示しています。

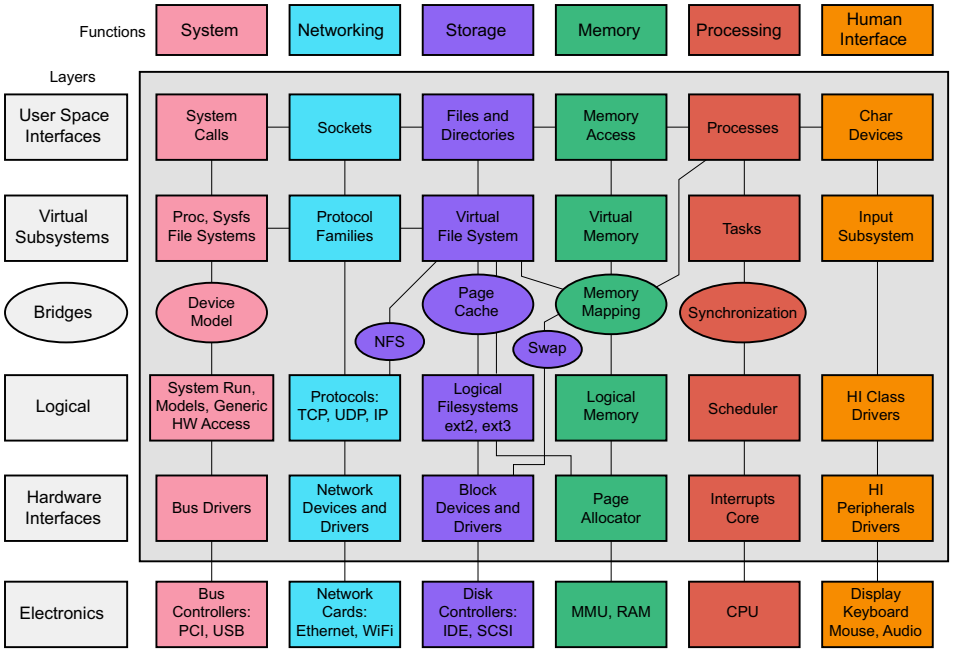

図 4-2 : Linux カーネル

4.3.2 ドライバー

ザイリンクス SDK ドライバーについては、オンラインの資料を参照してください。

4.4 U-Boot

マイクロプロセッサはメモリ内にあるコードを実行できる一方で、オペレーティング システムは通常、ハードディスク、CD-ROM、USB ディスク、ネットワーク サーバー、その他の永久記憶媒体などの大容量デバイス内にあります。プロセッサに電源が投入されるときに、メモリ内にオペレーティング システムがない場合は、別の媒体にある OS をメモリ内に移動させるための特別なソフトウェアが必要です。このソフトウェアがブートローダーと呼ばれる小さなコードです。

U-Boot は、Linux コミュニティで頻繁に使用されているオープン ソースのブートローダーであり、ザイリンクスも Linux 用の MicroBlaze™ プロセッサや Zynq-7000 AP プロセッサで使用しています。ブートローダーは、Linux カーネルが必ずしも初期化するわけではないハードウェア (シリアル ポートおよび DDR など) を初期化します。システム プロバイダーは、通常、フラッシュ メモリ内に U-Boot を置いています。U-Boot は、3.3.3 第 2 段階ブートローダー (オプション) で説明した第 2 段階ブートローダーの代表的な例です。U-Boot は、便利な機能を数多く提供します。

たとえば、イーサネット、フラッシュ メモリ、および USB からイメージをロードして実行できるだけでなく、メモリからカーネル イメージを開始できます。また、メモリの読み出しや書き込み動作などさまざまなコマンドを使用するコマンド インタープリターの利用が可能になり、`ping` コマンドを用いたネットワーク動作も可能になります。

最新情報は、http://wiki.xilinx.com/zynq-uboot を参照してください。

XILINX

付録 A

Bootgen の使用

A.1 概要

Bootgen はブート可能なイメージを作成するスタンドアロン ツールで、Zynq®-7000 AP SoC プロセッサに最適です。このプログラムがパーティション リストの先頭にヘッダー ブロックを付けてブート イメージを構築します。オプションで、各パーティションの暗号化や認証も可能です。出力は、システムのブート フラッシュ メモリへ直接プログラムできるシングル ファイルです。認証や暗号化をサポートするその他のペリフェラル ファイルも、このツールで生成できます。

このツールは、SDK へ統合されてイメージ生成を自動化したり、あるいはコマンドライン ベースのスクリプトで使用できます。

A.2 BIF ファイルの構文

BIF ファイルは、ブート イメージの各コンポーネントをブート順に指定し、オプションで各イメージ コンポーネントに属性を適用できるようにします。各イメージ コンポーネントは、通常 1 つのパーティションにマップされますが、1 つのイメージ コンポーネントがメモリ内で連続する場合は、1 つのイメージ コンポーネントが複数パーティションにマップされます。

BIF ファイルの構文は次の形式となります。

```
name ":"{" "["attributes"]" datafile..."}"
```

- name (ファイル名) と {...} (グループ化) を使用し、ROM 内のパーティションに分類するファイルをまとめます。{...} には 1 つまたは複数のデータ ファイルがリストされます。
- ファイルの拡張子からイメージ データの種類 (ELF、BIT、RBT、または INT - `[init]` 属性を持つデータ ファイル) が推論され、ファイルの種類に基づいて必要となる特別な前処理が行われます。
- データ ファイルは、ファイル名の前に構文 `["attributes"]` を使用することで、オプションで属性を持つことができます。
- 属性によって、データ ファイルに何らかの特性が適用されます。
- 複数の属性がある場合は、カンマ (,) で区切って並べることができます。その場合の順序は重要ではありません。属性は、1 つあるいは複数のキーワードを引用符で囲みます。
- 現在のディレクトリにファイルがない場合は、ファイル名を含むファイル パスを追加できます。ファイルのリスト方法は自由形式で、1 行にすべてをリスト (スペースで区切る、最低 1 スペースは必要) しても、1 つずつと改行してもかまいません。
- スペースは無視されるため、読みやすいように追加しても問題ありません。
- C 形式のブロック コメント /*...*/、または C++ 形式の行コメント //... を使用できます。

BIF ファイルの例

次に、例としてシンプルな BIF ファイルの一部を示します。

```
// A simple BIF file example.

the_ROM_image:
   {
   [init]init_data.int
   [bootloader]myDesign.elf
   Partition1.bit
   Partition1.rbt
   Partition2.elf
   }
```

次に、パーティションが暗号化および認証される BIF ファイルの例を示します。

```
image {
[aeskeyfile]secretkey.nky /* this is the key file used for AES */
[pskfile]primarykey.pem /* primary secret key file for authen.*/
[sskfile]secondarykey.pem /* secondary secret key file for authen.*/
[bootloader,authentication=rsa] fsbl.elf /*first stage bootloader */
[authentication=rsa]uboot.elf /* second stage bootloader */
linux.gz /* OS image (compressed)*/
}
```

BIF ファイルの属性

BIF ファイルには、次の 2 種類の属性があります。

- `bootloader` ー ELF データ ファイルを FSBL として認識します。
  - これらの属性は ELF ファイルのみに適用可能
  - 1 ファイルのみ FSBL に指定可能
- `init` ー INT (`[init]` 属性を持つデータ ファイル) をレジスタ初期化ファイルとして認識します。

次の表に、BIF ファイルの属性および属性タイプを示します。

表 A-1 : BIF ファイルの属性

識別子	説明
init	レジスタ初期化ブロック
booloader	FSBL を含むパーティション
alignment = <value>	バイト アライメントを設定
offset = <value	絶対オフセットを設定
checksum =<value>	チェックサムを指定 (`md5`、`sha1`、`sha2`)
encryption = <value>	暗号機能を指定 (`none` または `aes`)
authentication = <value>	このパーティション用の総メモリ容量を確保します。パーティションは、この領域までパディングされます。
pskfile	パーティションを符号化するために使用するプライマリ非公開キー (PSK)
psksignature	PSK を使用して作成された SPK シグナチャ
sskfile	パーティションを符号化するために使用するセカンダリ非公開キー (SSK)

表 A-1 : BIF ファイルの属性 (続き)

識別子	説明
ppkfile	パーティションを認証するために使用するプライマリ公開キー (PPK)
spkfile	パーティションを認証するために使用するセカンダリ公開キー (PPK)
aeskeyfile	AES キー ファイル
presign =<filename>	符号付きパーティションをインポート
udf_data =<filename>	認証証明の User Defined Field (ユーザー定義フィールド) へコピーされる最大 56 バイトのデータを含むファイルをインポート

次の表に、Bootgen でサポートされるファイルを示します。

表 A-2 : Bootgen でサポートされるファイル

拡張子	説明	注記
bin	バイナリ	未処理のバイナリ ファイル
.bit/.rbt	ビットストリーム	BIT ファイル ヘッダーを除く
.dtb	バイナリ	未処理のバイナリ ファイル
image.gz	バイナリ	未処理のバイナリ ファイル
.elf	ELF	シンボルとヘッダーを除く
.int	レジスタ初期化	
.nky	AES キー	
.pk1	RSA キー	

A.3 初期化ペアおよび INIT ファイルの属性

ブート イメージ ヘッダーの固定部分の最後に 256 の初期化ペアがあります。1 つのペアは、32 ビットのアドレス値と 32 ビットのデータ値で構成されるため、初期化ペアはこれにしたがって指定されます。初期化が実行されない場合、アドレス値はすべて `0xFFFFFFFF` となり、データ値はすべて `0x00000000` となります。

これらの初期化ペアは、デフォルトでファイル拡張子が `.int` のテキスト ファイルで設定されますが、拡張子は変更できます。

このファイルは、ファイル名の前に [init] ファイル属性を使用することによって、.bif 内で INIT ファイルとして認識されます。

データ フォーマットでは、動作指示子の後に次の情報が続きます。

- アドレス値
- = 文字
- データ値

ライン (行) は、セミコロン (;) で終わります。次に、「`.set.`」動作指示子の例を示します。

```
.set.0xE0000018 = 0x00000411;   // This is the 9600 uart setting.
```

Bootgen は、最大で 256 ペアまで、INT ファイルからのブート ヘッダー初期化データを埋めていきます。BootROM が実行されると、アドレス値を参照します。その値が `0xFFFFFFFF` でなければ、BootROM はアドレス値に続く次の 32

ビット値を使用してアドレスの値を書き込みます。BootROM は、アドレス値が 0xFFFFFFFF になるまで、または 256 番目の初期化ペアへ到達するまで、初期化ペアを順に設定し続けます。

Bootgen は、C/C++ に完全準拠したプリプロセッサとすべての指示子をサポートしています。さらに、パラメーター付きの拡張マクロもサポートしています。パラメーターは、GCC に準拠する -D オプションを使用して Bootgen コマンド ラインへ渡すことができ、#define のように機能します。

Bootgen では、次に示す演算子を含む式の評価 (優先を強調するネストされたかっこを含む) を実行できます。

```
*   = 乗算
/   = 除算
%   = モジュロ除算
+   = 加算
-   = 減算
~   = 論理否定
>>   = 右シフト
<<   = 左シフト
&   = バイナリ AND
|   = バイナリ OR
^   = バイナリ NOR
```

数値は 16 進数 (0x)、8 進数 (0o)、または 10 進数が可能です。数表現は、128 ビットの固定小数点整数として保持されます。読みやすいように、表現演算子の前後にスペースを入れることができます。

プリプロセッサでは、BIF および INT ファイルのパラメーター指定が可能ですが、コマンド ラインから選択できる複数コンフィギュレーションを持つ BIF および INTファイルのパラメーター指定も可能です。INT ファイルと一緒にインクルードファイルを用いると、そのままの値ではなく、記号化された値を使用できるようになるため便利にになります。

例 :

```
#include "register_defs.h"

.set. kBAUD_RATE_REG = ( k9600BAUD | kDOUBLE_RATE ) << BAUD_BITS;
```

-D オプションを使用して Bootgen コマンド ラインで値を設定することも可能です。-D オプションは、インクルードファイルの #define コマンドと同様に機能します。例 :

```
-D kCURRENT_RATE = 10
```

これによって、Bootgen がシェル スクリプトから実行されるとき、あるいは、INT ファイルを繰り返し変更せずにさまざまな値を試すときに、コマンド ラインから直接 INT 値を設定できます。

#if などの指示子を使用して異なるコンフィギュレーションを選択し、BIT ファイルや INT ファイルで使用される値を渡すことも可能です。

A.4 暗号化の概要

暗号化のプライベート キーは、eFUSE またはブロック RAM メモリ内に格納されます。BootROM は暗号プライベート キーを使用し、最初の FSBL パーティション ブート イメージをデコードします。実際、暗号化は Zynq-7000 ハードウェアの AES/HMAC エンジンで行われます。

パーティションを暗号化するには、次の手順を実行します。

1. コマンドラインに -encrypt オプションを適用し、変数は efuse または bbram を使用します。
2. BIT ファイルに [aeskeyfile] 属性を使用するキー ファイルをリストします。
3. 暗号化されるべき BIT ファイル内の各イメージ ファイルに [encryption=aes] 属性が適用されていることを確認します。

次にコマンド ラインの例を示します。

```
Bootgen ….-encrypt efuse

Example BIF file:

image:{
   [aeskeyfile]secretkey.nky
   [bootloader,encryption=aes] fsbl.elf
   [encryption=aes]uboot.elf
   linux.gz
}
```

A.5 認証の概要

Zynq-7000 RSA 認証では、プライマリ キーとセカンダリ キーを使用します。プライマリ キーがセカンダリ キーを認証し、セカンダリ キーはパーティションを認証します。

キーを示す頭文字の最初のアルファベットは、プライマリの場合 P、セカンダリの場合 S となります。キーを示す頭文字の 2 番目のアルファベットは、公開の場合 P、非公開の場合 S となります。キーの種類は次の 4 つです。

- PPK = プライマリ公開キー
- PSK = プライマリ非公開キー
- SPK = セカンダリ公開キー
- SSK = セカンダリ非公開キー

BootGen は、次の 2 つの方法で認証証明を作成できます。

- PSK と SSK を与えます。これらの 2 つの入力を使用し、SPK シグナチャがオンザフライで計算されます。
- PPK と SSK、そして SPK シグナチャを入力として与えます。この方法は、PSK が分からない場合に使用されます。

プライマリ キーはハッシュ化されて eFUSE 内に格納されます。このハッシュ値は、FSBL によってブート イメージに格納されたプライマリ キーのハッシュ値と比較されます。

BIT ファイルの例を次に示します。

```
image {
   [aeskeyfile]secretkey.nky
   [pskfile]primarykey.pem
```

```
    [sskfile]secondarykey.pem
    [bootloader,authentication=rsa] fsbl.elf
    [authentication=rsa]uboot.elf
    linux.gz
}
```

コマンド ラインで Bootgen を使用する例

```
bootgen -image bootimage.bif -o i my.mcs -efuseppkbits efuseppkbits.txt -encrypt
bbram developer.nky -p xc7z020clg484 -w on
```

A.6 Bootgen コマンドのオプション

次の表に Boootgen コマンド ライン オプションを示します。

表 A-3 : Bootgen コマンド ラインのオプション

引数	説明
-efuseppkbits <filename>	PPK ハッシュを含めるために書き込みを行う efuse ファイルの名前を指定します。
-encrypt [bbram | efuse] [StartCBC=<hex_string>] [Key0=<hex_string>] [HMAC=<hex_string>] [<filename>[.nky]]	暗号化方法を指定します。 **注記** : 重要な情報は BIF ファイルに指定されているため、新規デザインでは変数をイタリック体で表すことは推奨していません。
-f <filename>[.opt]	オプション ファイル名。 ファイル拡張子が分からない場合は、.opt と仮定します。 これらのオプションは、コマンド ライン オプションと同じです。 改行はスペースとして見なされます。
-fill [<hex_byte>]	フィル用に使用するバイトを指定します。
-generate_hashes	SHA256 ハッシュ ファイルを生成します。
-h	ヘルプ サマリをプリントアウトします。
-image <filename>[.bif]	入力するブート イメージ ファイル (*bif) を指定します。
-log <filename>[.log	stdout や stderr の代わりに、人間が読めるテキストをファイルとして出力します。拡張子が指定されていない場合は、.log 拡張子が追加されます。
-o i <filename>	出力ファイルを指定します。 サポートされる出力ファイルは、次のとおりです。 • bin • mcs 拡張子が指定されていない場合は、bin が追加されます。
-p <partname>	ザイリンクスのデバイス名を指定します。この名前は暗号キーの生成で必要となり、NKY の「Device」行に正確にコピーされます。それ以外は、Bootgen で使用されません。

表 A-3 : Bootgen コマンド ラインのオプション (続き)

引数	説明
-q [s|e|w|i	メッセージ出力を無効にします。オプションの後に続く文字列は、無効にするメッセージ タイプを示します。スペースでは、メッセージ タイプの文字を分離させることはできませんが、どのような順序で表しても問題ありません。 一度に使用するメッセージ タイプ数は制限されていません。メッセージ タイプの文字列はオプションです。 メッセージ タイプをブランクにすることは、`-q s` を使用した場合と同じです。 大文字を使用すると、メッセージ タイプは有効になります。 メッセージ タイプの文字は次のとおりです。 `s` = STATUS メッセージを無効にする `i` = INFO メッセージを無効にする `w` = WARNING メッセージを無効にする `e` = ERROR メッセージを無効にする
-spksignature <filename>	書き込みを行う spk シグナチャ ファイルの名前を指定します。BIT ファイルで pskfile および spkfile オプションを使用して指定する必要があります。
-w [on|off]	出力ファイルが存在する場合の動作を指定します。`-w on` は、上書きを意味します。

A.7 イメージ ヘッダー テーブル

通常、イメージ ヘッダー テーブルは、固定サイズのブート ヘッダーとレジスタ初期化テーブルのすぐ後ろに現れます。つまり、アドレス 0x000008A0 に現れます。イメージ ヘッダー テーブルは、ヘッダーの後に一連のイメージ ヘッダーが続いて構成されています。

イメージ ヘッダー テーブルは連続する必要はありませんが、bootgen によって継続的に生成されます。各イメージ ヘッダーは、`NextEntryOffset` によって次のイメージ ヘッダーとリンクしています。

注記 : オフセットはワード (バイト オフセットではない) で指定されます。変換する場合は、ワード オフセット値に 4 を掛けるとバイト オフセットになります。

表 A-4 : イメージ ヘッダー テーブルのヘッダー

オフセット	名称	注記
0x0	バージョン	0x01010000
0x4	イメージ ヘッダー数	
0x8	パーティション ヘッダーのワード オフセット	
0xC	最初のイメージ ヘッダーのワード オフセット	
0x10	パディング	64 バイト バウンダリまで 0xFFFFFFFF で埋められている

ブート イメージの規定では、イメージ ヘッダーはブート イメージ全体に散在できますが、イメージ ヘッダー テーブルのヘッダーの後には、bootgen で生成された一連のイメージ ヘッダー が続きます。

A.7.1 パーティション ヘッダー テーブル

パーティション ヘッダー テーブルは、次の表に示すデータを含む構造体の配列 (AOS) です。FSBL パーティションを含め、各パーティションに 1 つの構造体があります。テーブル内の最後の構造体は、すべて NULL 値となります (チェックサムを除く)。

表 A-5 : パーティション ヘッダー テーブル

オフセット	名称	説明
0x0	パーティション データ ワード長	暗号化されていないパーティション データの長さ
0x4	抽出されたデータ ワード長	暗号化されたデータの長さ
0x8	合計パーティション ワード長 (認証証明を含む)	暗号化、パディング、拡張、認証をすべて合計した長さ
0x0C	デスティネーション ロード アドレス	このパーティションがロードされる RAM アドレス
0x10	デスティネーション実行アドレス	ロード後、このパーティションの実行可能アドレス
0x14	イメージのデータ ワード オフセット	ブート イメージの開始に対するパーティション データの位置
0x18	属性ビット	58 ページの表 A- 6 を参照
0x1C	セクション数	シングル パーティション内のセクション数
0x20	チェックサム ワード オフセット	ブート イメージ内のチェックサム ワードの位置
0x24	イメージ ヘッダー ワード オフセット	ブート イメージ内のイメージ ヘッダーの位置
0x28	認証証明ワード オフセット	ブート イメージ内の認証証明の位置
0x2C	未使用	0x00000000 にする
0x30	未使用	0x00000000 にする
0x34	未使用	0x00000000 にする
0x38	未使用	0x00000000 にする
0x3C	ヘッダー チェックサム	パーティション ヘッダーの以前のワードの合計値

A.7.2 パーティション属性ビット

表 A-6 : パーティション属性ビット

ビット フィールド	説明	注記
31:16	データ属性	インプリメントされていない
15	RSA シグナチャの有無	0 – RSA 認証証明なし 1 – RSA 認証証明あり

表 A-6 : パーティション属性ビット (続き)

ビット フィールド	説明	注記
14:12	チェックサム タイプ	b000 = 0 = チェックサムなし b001 = 1 = RFU (将来的な使用のために予約) b010 = 2 = RFU b011 = 3 = RFU b100 = 4 = RFU b101 = 5 = RFU b110 = 6 = RFU b111 = 7 = RFU
11:8	デスティネーション インスタンス	インプリメントされていない
7:4	デスティネーション デバイス	0 - なし 1 - PS 2 - PL 3 - INT 4 ～ 15 - 予約
3:2	ヘッド アライメント	
1:0	テール アライメント	

A.8 イメージ ヘッダー

表 A-7 : イメージ ヘッダー

オフセット	名称	注記
0x0	次のイメージ ヘッダーのワード オフセット	次のイメージ ヘッダーと関連します。最後のイメージ ヘッダーの場合は 0 となります。
0x4	最初のパーティション ヘッダーのワード オフセット	最初のパーティション ヘッダーと関連します。
0x8	パーティション数	常に 0
0xC	イメージ名の長さ	実際のパーティション数の値
0x10 ～ N	イメージ名	ビッグエンディアン順に圧縮されます。文字列を再構築するため、一度に 4 バイトを解凍し、順序を逆にして連結されます。 たとえば、文字列「FSBL10.ELF」は次のように圧縮されます。 0x10 : 'L','B','S','F', 0x14 : 'E','.','0','1', 0x18 : '\0','\0','F','L' 圧縮されたイメージ名は 4 の倍数バイトです。
varies	0x00000000	ストリング ターミネーター (終端名) です。
varies	0xFFFFFFFF	64 バイト境界まで繰り返しパディングされます。

XILINX

付録 B

その他のリソース

B.1 ソリューション センター

すべての設計段階におけるデバイス、ソフトウェア ツール、および IP に関するサポートは、ザイリンクスのソリューション センターを参照してください。設計に関するアシスタンス、アドバイス、問題解決のヒントなどを記載しています。

B.2 ザイリンクスの資料

- 製品サポートおよび資料 : http://japan.xilinx.com/support
- ザイリンクス用語集 : http://japan.xilinx.com/company/terms.htm
- デバイス ユーザー ガイド : http://japan.xilinx.com/support/documentation/user_guides.htm

1. ザイリンクスのフォーラム Wiki リンク :
   - http://forums.xilinx.com
   - http://wiki.xilinx.com
   - http://wiki.xilinx.com/zynq-linux
   - http://wiki.xilinx.com/zynq-uboot

- ザイリンクス Git のウェブサイト : http://git.xilinx.com

2. ザイリンクス エコシステム パートナー企業のウェブサイト : http://japan.xilinx.com/products/silicon-devices/soc/zynq-7000/ecosystem/index.htm
3. 『Vivado Design Suite ユーザーガイド リリース ノート、インストールおよびライセンス』(UG973)
4. 『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)
5. 『ザイリンクス デザイン ツール : リリース ノート ガイド』(UG631)

B.3 参考資料

- Zynq 関連ページ : http://japan.xilinx.com/products/silicon-devices/soc/zynq-7000/

6. 『Zynq-7000 AP SoC テクニカル リファレンス マニュアル』(UG585)

B.3.1 PL 関連資料 - デバイスおよびボード

- ザイリンクス 7 シリーズのサポート ページ : http://japan.xilinx.com/support/documentation/7_series.htm

PL リソースの詳細は、7 シリーズ FPGA のユーザー ガイドを参照してください。

B.3.2 ソフトウェア関連資料

7. 『Zynq-7000 All Programmable SoC コンセプト、ツール、およびテクニック ガイド』(UG873)

   スタンドアロンおよび FSBL のソース ドライバーは、ザイリンクスの IDE Design Suite Embedded Edition の一部として提供されています。Linux ドライバーは、ザイリンクスのオープン ソース Wiki (http://wiki.xilinx.com) で提供しています。

   IP、ミドルウェア、オペレーティング システムなどに関するシステム ソフトウェア ソリューションは、ザイリンクス アライアンス プログラム パートナーが提供しています。最新情報は、ザイリンクスのウェブサイトの Zynq-7000 (http://japan.xilinx.com/products/silicon-devices/soc/zynq-7000/) を参照してください。

B.3.3 git 情報

- http://git-scm.com
- http://git-scm.com/documentation
- http://git-scm.com/download

B.3.4 デザイン ツール関連資料

- 『ChipScope Pro ソフトウェアおよびコア ユーザー ガイド』(UG029)
- EDK 関連資料 : http://japan.xilinx.com/support/documentation/dt_edk_edk14-6.htm

8. 『OS およびライブラリ資料コレクション』(UG643)
9. 『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)
10. 『Vivado Design Suite ユーザー ガイド : リリース ノート、インストール、およびライセンス』(UG973)
11. 『Vivado Design Suite ユーザー ガイド : エンベデッド ハードウェア デザイン』(UG898)
12. 『Vivado Design Suite チュートリアル : エンベデッド ハードウェア デザイン』(UG940)
13. 『Vivado Design Suite ユーザー ガイド : Vivado IDE の使用』(UG893)
14. 『Vivado Design Suite チュートリアル : IP インテグレーターを使用して IP サブシステムを設計』(UG995)
15. 『Vivado Design Suite ユーザー ガイド : IP インテグレーターを使用して IP サブシステムを設計』(UG994)
16. 『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)
17. 『Vivado Design Suite ユーザー ガイド : プログラムおよびデバッグ』(UG908)

B.4 サードパーティが提供する資料

Zynq-7000 デバイスに含まれているベンダー IP あるいは関連する国際的なインターフェイス規格の機能については、次の資料を参照してください。

注記 : ARM 社が提供する資料は、http://infocenter.arm.com/help/index.jsp から入手できます。

- ARM 社 ― 『AMBA Level 2 Cache Controller (L2C-310) Technical Reference Manual』(または PL310)
- ARM 社 ― 『AMBA Specification』Revision 2.0、1999 年 (IHI 0011A)
- ARM 社 ― 『Architecture Reference Manual』(ARM 社のサイト登録が必要)
- ARM 社 ― 『Cortex-A Series Programmer's Guide』

- ARM 社 － 『Cortex-A9 Technical Reference Manual』
- ARM 社 － 『Cortex-A9 MPCore Technical Reference Manual』(DDI0407F) : アクセラレーター コヒーレンシ ポート (ACP)、CPU プライベート タイマーとウォッチドッグ タイマー (AWDT)、イベント バス、汎用割り込みコントローラー (GIC)、グローバル タイマー (GTC)、およびスヌープ制御ユニット (SCU) に関する説明が含まれる
- ARM 社 － 『Cortex-A9 NEON Media Processing Engine Technical Reference Manual』
- ARM 社 － 『Cortex-A9 Floating-Point Unit Technical Reference Manual』
- ARM 社 － 『CoreSight v1.0 Architecture Specification: Includes descriptions for ATB Bus, and Authentication』
- ARM 社 － 『CoreSight Program Flow Trace Architecture Specification』
- ARM 社 － 『Debug Interface v5.1Architecture Specification』
- ARM 社 － 『Debug Interface v5.1 Architecture Specification Supplement』
- ARM 社 － 『CoreSight Components Technical Reference Manual』: エンベデッド クロス トリガー (ECT)、エンベデッド トレース バッファー (ETB)、インスツルメンテーション トレース マクロセル (ITM)、デバッグ アクセス ポート (DAP)、およびトレース ポート インターフェイス ユニット (TPIU) に関する説明が含まれる
- ARM 社 － 『CoreSight PTM-A9 Technical Reference Manual』
- ARM 社 － 『CoreSight Trace Memory Controller Technical Reference Manual』
- ARM 社 － 『Generic Interrupt Controller v1.0 Architecture Specification』(IHI 0048B)
- ARM 社 － 『Generic Interrupt Controller PL390 Technical Reference Manual』(DDI0416B)
- ARM 社 － 『PrimeCell DMA Controller (PL330) Technical Reference Manual』
- ARM 社 － アプリケーション ノート 239 : 『Example programs for CoreLink DMA Controller DMA-330』
- ARM 社 － 『PrimeCell Static Memory Controller (PL350 シリーズ) Technical Reference Manual』Revision r2p1、2007 年 10 月 12 日 (ARM DDI 0380G)
- BOSCH 社 － 『CAN Specification Version 2.0 PART A and PART B』1991 年
- Cadence 社 － 『Watchdog Timer (SWDT) Specification』
- IEEE 802.3-2008 － 『IEEE Standard for Information technology-Specific requirements - Part 3 : Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications』2008 年
- インテル コーポレーション (Intel Corp.) －『Enhanced Host Controller Interface Specification for Universal Serial Bus』v1.0、2002 年
- ISO 11898 Standard USB Association － 『USB 2.0 Specification』
- Multimedia Card Association － 『MMC-System-Specification』v3.31
- SD Association － 『Part A2 SD Host Controller Standard Specification』Ver2.00 Final 070130
- SD Association － 『Part E1 SDIO Specification』Ver2.00 Final 070130
- SD Group － 『Part 1 Physical Layer Specification』Ver2.00 Final 060509