Zynq-7000 EPP ソフトウェア開発者向けガイド

UG821 (v2.0) 2012 年 4 月 24 日

XILINX®

本資料は英語版 (v2.0) 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
2012 年 3 月 26 日	v1.0	初版リリース
2012 年 4 月 24 日	v2.0	表 3-1 に「パーティション データ セクション カウント」を追加。 第 4 章のベアメタル BSP を削除。

目次

XILINX®

第 1 章

はじめに

このガイドでは、ザイリンクスの 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 (EPP) 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を対象としています。

- 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

1.1 内容およびその他のリソース

このガイドは、Zynq-7000 EPP デバイス向けのエンベデッド アプリケーション開発について概説することを目的としており、一般的な設計ガイドラインではありません。詳細については、次の資料を参照してください。公開されている資料の一覧は、付録 A 「その他のリソース」を参照してください。リンクがない資料は、Zynq Beta ラウンジから入手できます。

- Zynq-7000 エクステンシブル プロセッシング プラットフォーム概要 (DS190)
- Zynq-7000 EPP DC 特性および AC スイッチング特性 (DS187)
- Zynq-7000 EPP テクニカル リファレンス マニュアル (UG585)
- Zynq EDK コンセプト、ツール、およびテクニック ガイド (UG873)
- SDK オンライン ヘルプ
- 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)
- OS およびライブラリ ドキュメント集 (UG643)
- USB ケーブル インストール ガイド (UG344)
- プラットフォーム ケーブル USB II データ シート (DS593)
- 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)

このガイドは、次の内容で構成されています。

6 ページの「アーキテクチャの選択」では、EPP デザイン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。

7 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 EPP プロセッサでアプリケーションを実行しながら、Zynq-7000 EPP 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。8 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 EPP 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 EPP 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 3 つのブート モードを紹介し、その後 2 つのブート 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ミング方法の説明も含まれます。

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 「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 コンセプト、ツール、およびテクニック ガイド』(UG873) を参照してください。

1.2 アーキテクチャの選択

Zynq-7000 EPP 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 EPP 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。

1.2.1 マルチプロセッシングの考察

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。各イメージがオペレーティング システムのものであることも可能ですが、一般に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 システムでは、ネットワークやユーザー インターフェイスを介して、比較的簡単に外部と接続できます。
- 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します。

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 (7 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
- 1 つのプロセッサが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があります。

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ています。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 ユーザーに透過的です。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

1.3 オペレーティング システム (OS) の考察

1.3.1 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わずかに消費し、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は受け入れられ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っていることに伴い、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。

1.3.2 オペレーティング システム : Linux

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。

1.3.3 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。

XILINX®

第 2 章

ソフトウェア アプリケーションの開発フロー

Zynq™-7000 EPP 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できます。

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 EPP ソリューションには、Eclipse ベースの統合設計環境 (IDE) や GNU コンパイラー 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。

2.1 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 EPP プロセッサでアプリケーションを実行しながら、Zynq-7000 EPP 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 EPP 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、オペレーティング システムを使用しないベアメタル型アプリケーションとオープンソース Linux オペレーティング システムのアプリケーション両方の開発が可能です。XPS (Xilinx Platform Studio) などのザイリンクスのハードウェア設計ツールを使用すると、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できます。

XPS では、その他のデータ ファイルと共に XML 形式のハードウェア プラットフォーム情報を取り込み、その後、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、コンパイラー オプションの推論、PL のプログラム、JTAG 設定の定義、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。

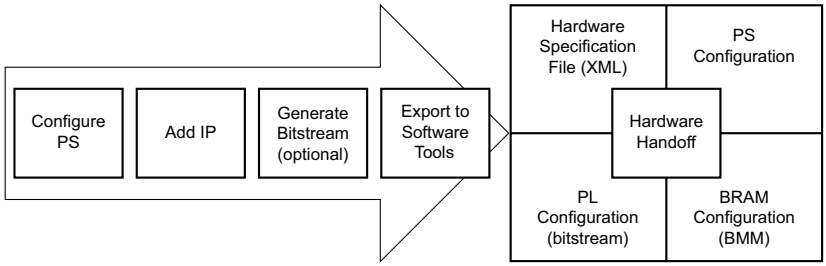

図 2-1 : ハードウェア ツールからソフトウェア ツールへのハンドオフ

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
- コンパイラー 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
- シミュレーター
- 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 EPP デバイスの直接サポート レベルがさまざまです。ザイリンクスでは、カーネルの開発およびデバッグ専用ツールを提供していませんが、サードパーティから提供されているため、それらをご利用ください。

後続のサブセクションでは、ザイリンクスが提供するソフトウェア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。

2.1.1 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ー ツールチェーン (GCC コンパイラー、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 ソフトウェア、アプリケーション固有機能用のミドルウェア 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レーション環境
- プロジェクト管理
- アプリケーションのビルド設定および `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リング用の統合環境
- 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 バージョン コントロール)

SDK は、ザイリンクスの ISE ® Design Suite のインストール パッケージやエンベデッド開発キット (EDK) に含まれていますが、スタンドアロンでインストールして利用することも可能です。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。

2.1.2 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。

2.1.3 ザイリンクス Cortex-A9 コンパイラー 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 コンパイラー 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 コンパイラー 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

2.1.4 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 コアを挿入して、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 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。

2.1.5 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。

2.1.6 ISim Simulator

ISE® Design Suite および PlanAhead™ ツールには、ISim HDL シミュレーター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) テクノロジを使用することによって、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。

2.2 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 (11 ページの図 2-2 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
- プロセッサ階層 1 (デバイス ドライバー)
- 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 : 通常、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。

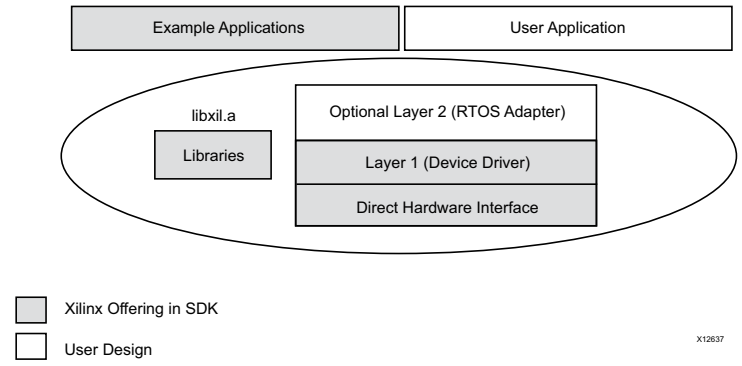

図 2-2 : ベアメタル デバイス ドライバーのアーキテクチャ

2.2.1 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 特有の関数および識別子を参照 (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

2.2.2 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
- デバイス機能の包括的なサポート
- 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

2.2.3 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、およびハードウェア レジスタへのユーザー アクセスを可能にするシンプルなマクロ
- 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、または全くない
- 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

2.3 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て、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) プログラミング 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 EPP 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 2-3 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。

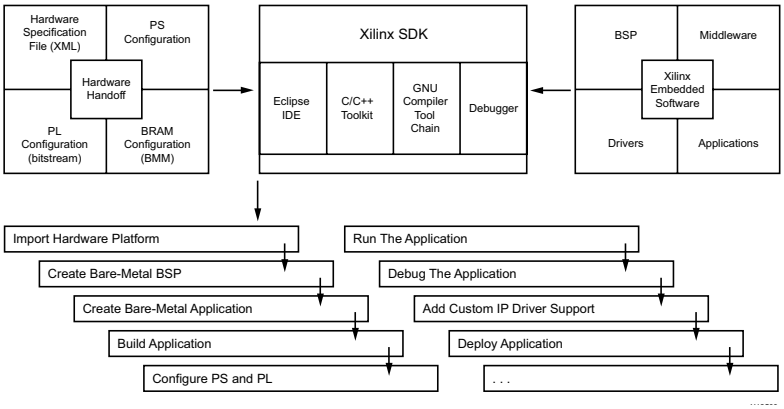

図 2-3 : ベアメタル アプリケーション開発の概要

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) を参照してください。

2.3.1 ハードウェア プラットフォーム情報をインポートする

XPS でハードウェア プラットフォーム データを作成し、SDK にそれらをインポートして、ハードウェア プラットフォーム プロジェクトを作成します。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

2.3.2 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 プロジェクトを作成後、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 では、多重 I/O (MIO) コンフィギュレーションを含む PS で有効化された 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) に含まれる『スタンドアロン BSP (v3.05.a)』(UG652) を参照してください。このユーザー ガイドへのリンクは、付録 A 「その他のリソース」に記載しています。

2.3.3 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。

2.3.4 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ブートローダーや 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。

2.3.5 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ビルド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 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ビルド オプション (コンパイラー、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。

2.3.6 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 EPP 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には、この手順は省略できます。

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。

2.3.7 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。

2.3.8 カスタム IP ドライバーのサポートを追加する

XPS 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て、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て、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) を参照してください。このユーザー ガイドへのリンクは、付録 A 「その他のリソース」に記載しています。

2.3.9 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 プログラミング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。

2.4 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て、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) プログラミング 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 EPP 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) に含まれる『スタンドアロン BSP (v3.05.a)』(UG652) を参照してください。このユーザー ガイドへのリンクは、付録 A 「その他のリソース」に記載しています。

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は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リングを行う
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 2-4 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。

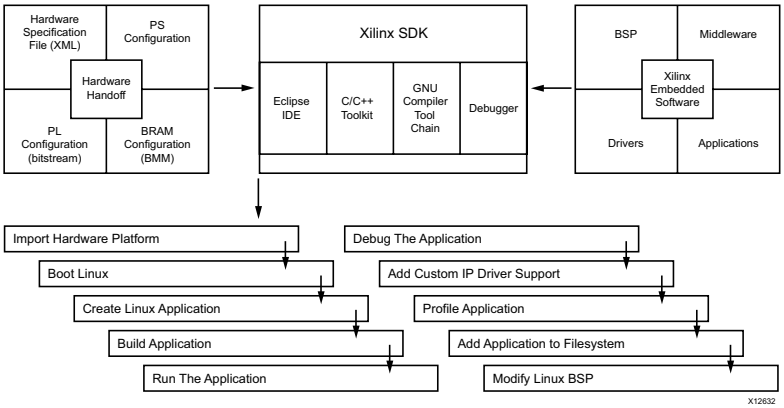

図 2-4 : Linux アプリケーション開発

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

2.4.1 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て、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 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 EPP で Linux を使用する場合、原則的に SDK は EPP 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 NAND 型または NOR 型によって異なり、Quad-SPI のパーティションには、FSBL、U-Boot、Linux カーネル、User、Scratch、および `/root` ファイル システムを含めることができます。

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。

2.4.2 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 ベンチマーク 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。

2.4.3 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 `makefiles` を作成) または自動管理 (SDK が `makefiles` 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ビルドを開始します。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ビルド オプション (コンパイラー、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。

2.4.4 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 EPP で Linux を動作させ、Linux 環境に SSH が含まれる場合は、`sftp` を使用することで、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`** を使用して実行可能ファイルを実行する

2.4.5 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。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 (コール スタック、ソース ビューアー、ディスアセンブリ、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。

2.4.6 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 を生成する場合には、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。

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。

2.4.7 アプリケーションのプロファイリングを行う

Linux ユーザー アプリケーションのプロファイリングには、アプリケーションを構築する際に **`-pg`** プロファイリング 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リングは、**`gprof`** ユーティリティに基づき、お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リングするために、SDK には呼び出しプロファイリング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。

2.4.8 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、次のようにして 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 EPP 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て、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。

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

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) に含まれる『スタンドアロン BSP (v3.05.a)』(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。

2.5 その他の情報

このセクションでは、ベアメタルおよび Linux アプリケーションの開発について解説しながら、ザイリンクスのソフトウェア ソリューションの概要を提供していますが、その他のツールおよびフローに関する情報は十分ではありません。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 EDK コンセプト、ツール、およびテクニック ガイド』(UG873) の次のセクションも参照してください。

- 「EDK デザイン フロー」
- 「Sysgen IP の追加」
- 「シミュレーション (システム レベル - ハードウェア イン ザ ループ)」

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 EPP 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ノンセキュア 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 EPP 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。

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て、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 が実行を開始して、残り 64 キロバイト エリアの予約がなくなると、256 キロバイト全体が利用できます。

FSBL ブート コードは完全にユーザー制御が可能で、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、必要に応じてユーザーが変更可能なサンプル 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) のペリフェラルの初期化コードが含まれます。詳細は、3.3.2 第 1 段階ブートローダー 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まれることができます。

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て、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することができます。

21 ページの図 3-1 に、ブート プロセスに焦点を当てた Zynq-7000 EPP プロセッサ システム (PS) 全体のレイアウトを示します。

注記 : DDR および SCU は BootROM では有効にできません。詳細は、『Zynq-7000 EPP 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。

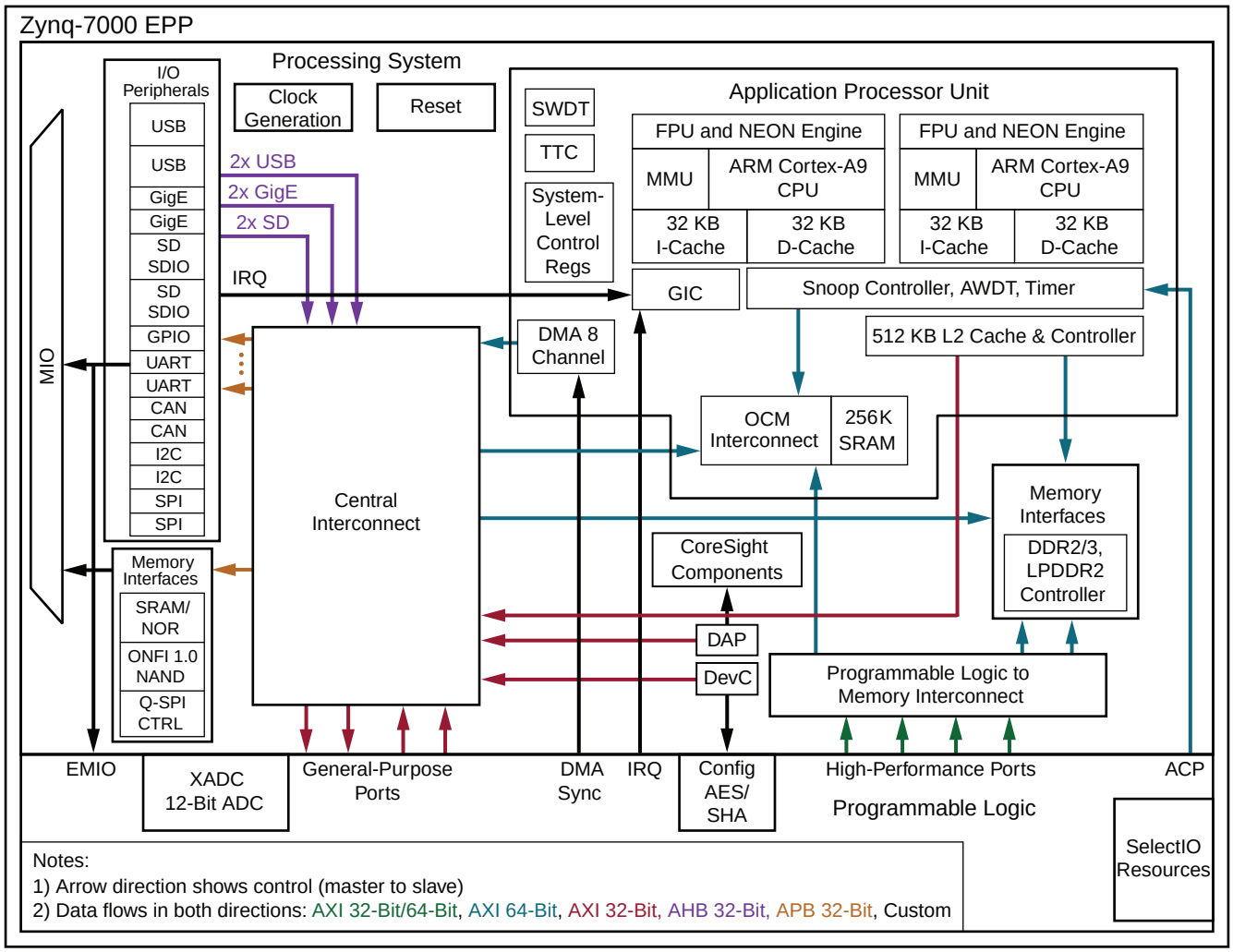

図 3-1 : Zynq-7000 EPP プロセッサ システムのハイレベル図

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 ノンセキュア 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 EPP テクニカル リファレンス マニュアル』(UG585) の「デバイス コンフィギュレーションおよびデバイス コンフィギュレーション ユニット」を参照してください。

3.3 ブート ステージ

Zynq-7000 EPP デバイスは、次に示すセキュア ブート プロセスとノンセキュア 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)

『Zynq-7000 EPP テクニカル リファレンス マニュアル』(UG585) の第 6 章のセクション 6.3 「BootROM」を参照してください。

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 ヘッダーの記述に従ってメモリマップされた NAOR または Quad-SPI のいずれかのフラッシュから OCM (BootROM による) または暗号化されていない XIP (eXecutes In Place) へロードされます。

FSBL の役割は次のとおりです。

- XPS から提供される PS コンフィギュレーション データを使用して初期化を行う (「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
- 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 アプリケーションの実行を開始する

注記 : FSBL は、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進む前に MMU 機能を無効にします。これは、Linux (および、おそらくその他のオペレーティング システム) は、開始時にこの機能が無効であることを前提としているためです。

使用される CPU やペリフェラルが FSBL でどのように初期化されるか、および簡潔な C ランタイム ライブラリをどのように使用するのかの詳細は、SDK で提供される FSBL コードを参照してください。

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) 、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、3.4.1 ブート イメージのフォーマット セクション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て、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、3.4.2 パーティション ヘッダー テーブル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、3.5 ブート イメージの生成を参照してください。

23 ページの図 3-2 に、BootROM が FSBL を OCM へロードするフローを示します。

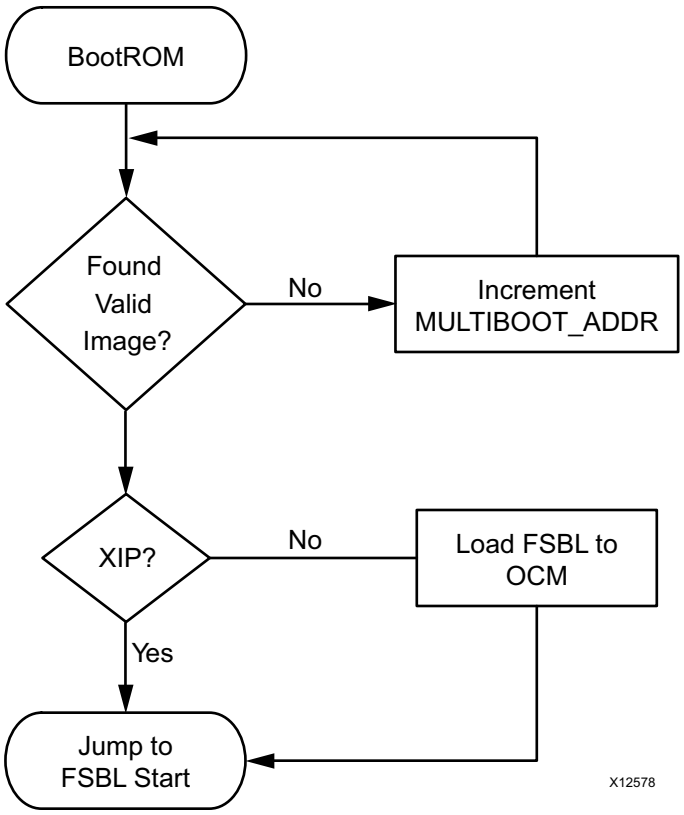

図 3-2 : FSBL フロー

Zynq PS のコンフィギュレーション

XPS の Zynq コンフィギュレーション UI を使用して、MIO および SLCR レジスタの初期化コードを生成します。XPS プロジェクト ディレクトリには、次のような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` で実行される初期化は、`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` で実行される初期化は、`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には、Tcl ファイルが有用です。たとえば、`ps7_init.tcl` ファイルを実行後、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。

重要 : 今後のリリースで PS 初期化データの位置およびフォーマットが変更される可能性があります。

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、4.3 U-Boot を参照してください。

3.4 イメージのフォーマット

3.4.1 ブート イメージのフォーマット

ブート イメージのフォーマットは、次の要素で構成されています。

- BootROM ヘッダー
- FSBL イメージ
- 1 つまたは複数の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 3-3 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。

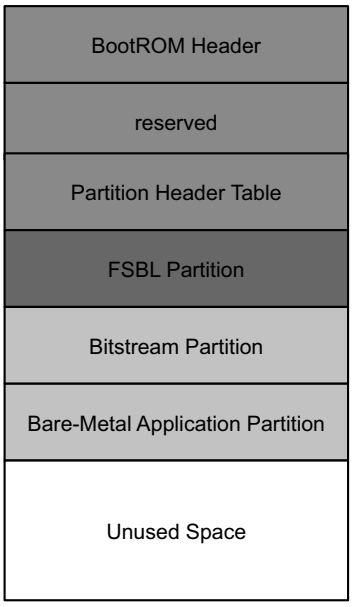

図 3-3 : Zynq-7000 EPP ブート イメージ フォーマット

図 3-4 に、Zynq-7000 EPP Linux のブート イメージ フォーマットの例を示します。

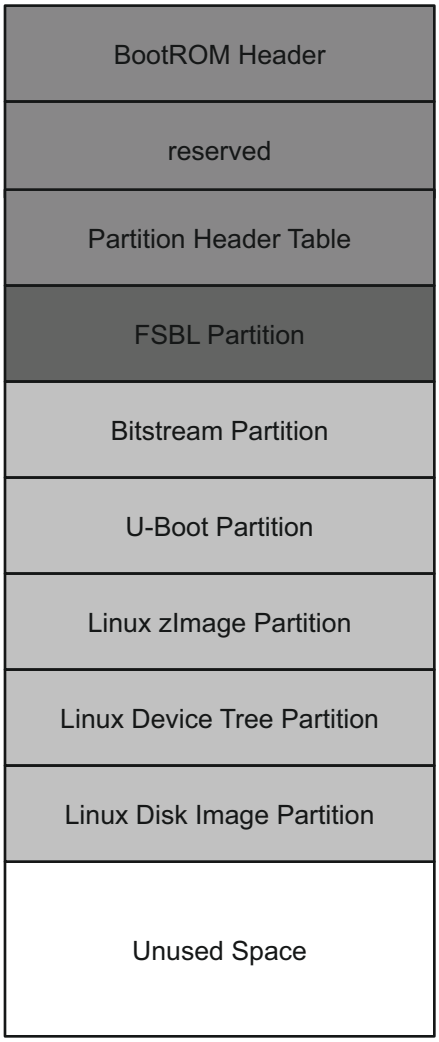

図 3-4 : Zynq-7000 EPP Linux ブート イメージ フォーマットの例

『Zynq-7000 EPP テクニカル リファレンス マニュアル』(UG585) のセクション 6.3.2「BootROM ヘッダーのフォーマット」を参照してください。

3.4.2 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、25 ページの表 3-1 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 3-1 : パーティション ヘッダー テーブル

オフセット	名前	説明
0x00	パーティション データ サイズ	パーティション データの長さ (ワード単位)
0x04	書き込まれたデータの長さ	復号化されたパーティション データの長さ (ワード単位)。この値は、暗号化されていないデータのパーティション データ サイズと同じになる
0x08	パーティション全体のサイズ	拡張用の未使用空間を含むこのパーティション ヘッダー テーブルに相当するブート イメージ パーティションのサイズ。図 3-5 を参照
0x0C	ロード アドレス	イメージをメモリへロードするためのアドレス。このアドレスへ PS データが書き込まれる。(これは、PL ビットストリーム パーティションには適用されない)
0x10	実行アドレス	実行を開始するアドレス。(これは、PL ビットストリーム パーティションには適用されない)
0x14	パーティション オフセット	このパーティション ヘッダー テーブルに対応するパーティション データの入力が開始する位置の、ブート イメージの始めからのワード数
0x18	属性	ビット 5 はビットストリーム パーティション用で、ビット 4 は PS パーティション用 (メモリへコピーされる)。その他のビットはすべて予約ビット
0x1C	パーティション データ セクション カウント	パーティション データ内にある個別にロードされたデータ セクションの数。 ELF ファイルなどのパーティション データをロードするメモリへのみ適用
0x20 ～ 0x3B	予約	
0x3C	チェックサム	ヘッダーの有効性をチェックするために FSBL で使用される。チェックサムは、この構造体にある以前の値をすべて合計したものをビット単位で論理否定演算 (bitwise NOT) する

図 3-5 に、長さフィールドを示します。

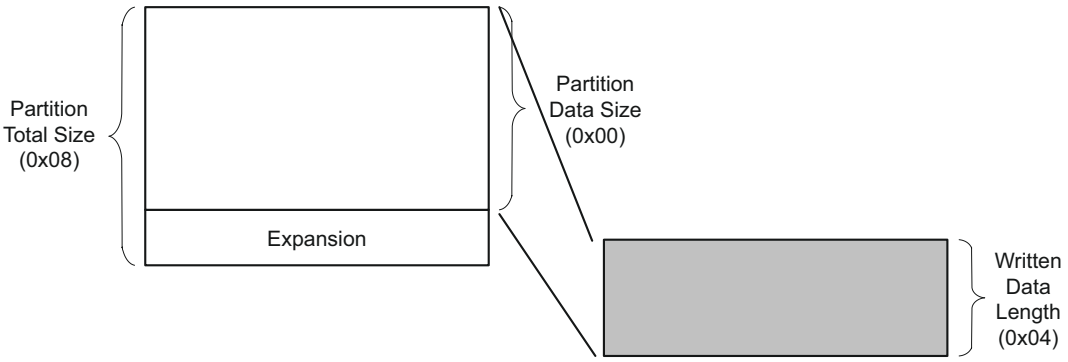

図 3-5 : パーティション ヘッダーの長さフィールド

3.5 ブート イメージの生成

Bootgen は、Zynq デバイスのコンフィギュレーション用のブート イメージを生成するツールです。BIT ファイルと ELF ファイルを 1 つのブート イメージに結合し、ブート イメージ フォーマット (BIF) ファイルで定義されたフォーマットで Zynq フラッシュ デバイスへロードします。PROMGen などの従来ツールの使用法とは異なり、フラッシュのパーティショニングやイメージのワード オフセットを計算する必要はありません。Bootgen がデータを自動的に 1 つのブート イメージに結合します。ブート イメージがフラッシュ ストレージへプログラムされると、Zynq BootROM および FSBL がこの中からユーザー データ イメージを自動抽出して Zynq デバイスをコンフィギュレーションします。

簡単なコマンド ラインの例を示します。

```
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されます。Bootgen のオンライン ヘルプを参照してください (-h コマンド ライン オプションを使用して表示)。

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) を使用して、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。

コードの一部の例を示します。

```
// A simple BIF file example.

the_ROM_image:
   {
   [init]init_data.int
   [bootloader]myDesign.elf
   Partition1.bit
   Partition1.rbt
   Partition2.elf
   Partition3.elf
   }
```

この中には 2 つの属性があります。

- `bootloader` - ELF データ ファイルを FSBL として認識します。ELF ファイルのみこれらの属性を持つことができ、FSBL として指定できるファイルは 1 つのみです。
- `init` - 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。

ELF ファイルには、ユーザーがコンパイルしたプロセッサ コードが含まれています。Bootgen は、ELT ファイルの不要なデータ セクションを削除し、コンパイルした コードをデータ パーティションとして追加します。

BIT および RBT ファイルは、Bitgen で生成されたビットストリーム ファイルです。Bootgen はビットストリームから BIT ファイル ヘッダーを削除して、要求に応じてファイルを暗号化し、ビットストリームをデータ パーティションとして追加します。

INT ファイルは [int] 属性を持つデータ ファイルで、FSBL がロードされて実行される前のブート時にレジスタを初期化するために使用されるオプションの Bootgen 機能です。この機能を利用してボー レートやクロック レートなどの値を設定できます。

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。

- `#ifdef`
- `#ifndef`
- `#define`
- `#undef`
- `#include`
- `#if`
- `#else`
- `#endif`
- `#elif`
- `#error`
- `#pragma`

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括弧を含む) 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。

XILINX

第 4 章

Linux

ザイリンクスが提供する Zynq™ 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。また、Embedded Linux フォーラム (http://forums.xilinx.com) をとおしても、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 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 リストを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 (http://wiki.xilinx.com) を参照してください。

ザイリンクスは、Linux カーネル、ザイリンクス ボード用 BSP、および一部 IP 用のドライバーを保存している公開 Git サーバーを提供しています。この Git サーバーは、サードパーティがザイリンクス ハードウェア向けのエンベデッド Linux ディストリビューションを構築できるようにする目的で設けられています。実際、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のではなく、独自の Linux システムを開発することも可能です。

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。

4.1 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て、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ブラウズできます。

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 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 をインストールして、コマンドラインから `gitk` を使用して実行できます。

31 ページの図 4-1 に、このツールのスクリーン ショットを示します。

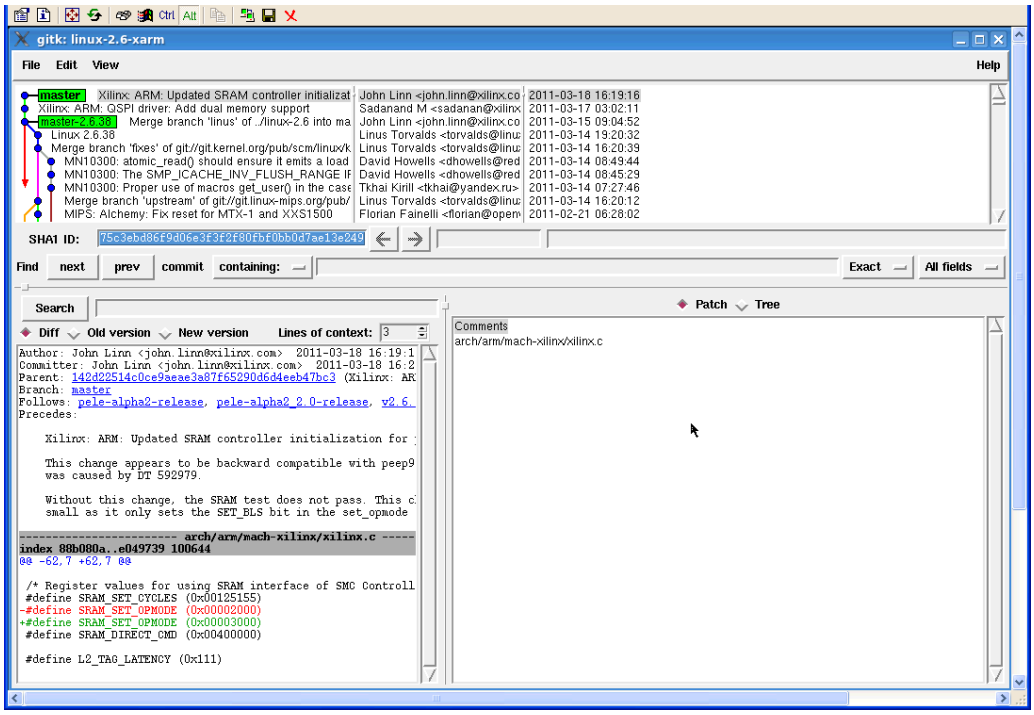

図 4-1 : Gitk

4.2 Linux BSP に含まれるもの

4.2.1 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。

32 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。

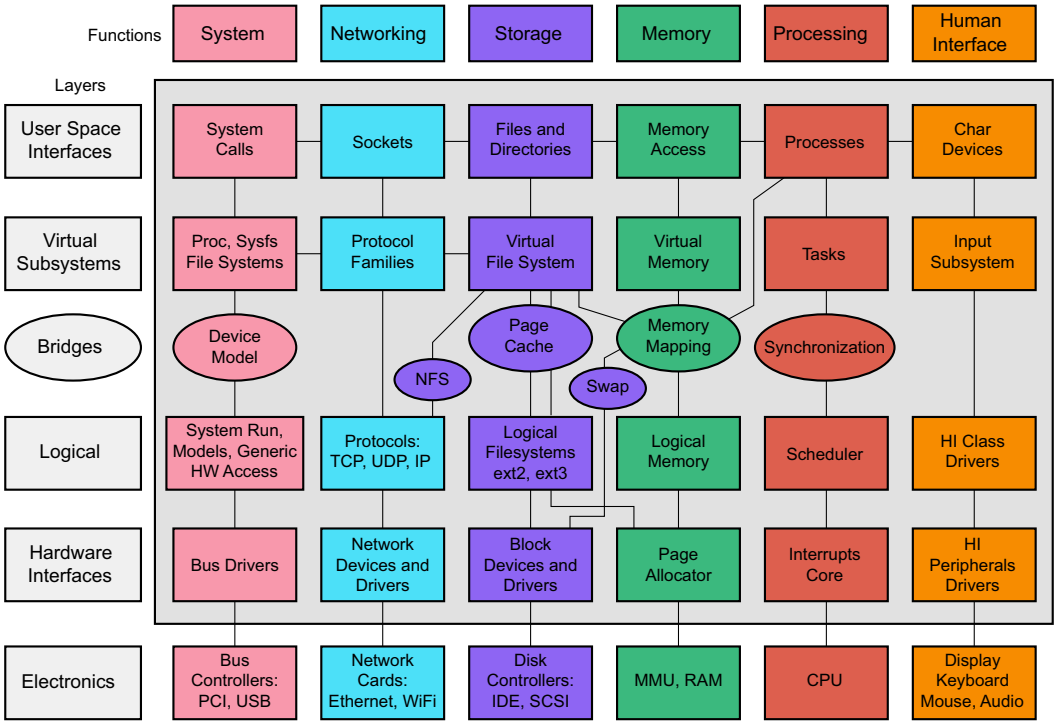

図 4-2 : Linux カーネル

4.2.2 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。

4.3 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 PowerPC® プロセッサおよび MicroBlaze™ 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

その他のリソース

A.1 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
- 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_1/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_1/irn.pdf
- ザイリンクスのフォーラムおよび 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

A.2 ソリューション センター

ザイリンクス ウェブ サイトの「ソリューション センター」ページでは、設計サイクルのあらゆる段階におけるデバイス、ソフトウェア ツール、および IP コアのサポートを提供しています。サポートには、設計のサポート、注意事項、問題解決のヒントなどが含まれます。

A.3 参考資料

A.3.1 Zynq-7000 EPP 関連資料

Zynq-7000 EPP に関する詳細は、次の資料を参照してください。

- Zynq-7000 EPP 製品パンフレット (CS692)
- Zynq-7000 EPP 製品概要 (DS190)
- Zynq-7000 EPP データシート(DS187)
- Zynq-7000 EPP パッケージおよびピン配置の仕様 (UG865)
- Zynq-7000 EPP エラッタ (EN191)

A.3.2 PL 関連資料 - デバイスおよびボード

PL リソースの詳細は、次に示す 7 シリーズ FPGA の資料を参照してください。

- ザイリンクス LogiCORE IP PCI 7 シリーズ FPGA Express 用インテグレイテッド ブロックの製品仕様 (DS821)
- ザイリンクス 7 シリーズ FPGA SelectIO リソース ユーザー ガイド (UG471)
- ザイリンクス 7 シリーズ FPGA クロッキング リソース ユーザー ガイド (UG472)
- ザイリンクス 7 シリーズ FPGA メモリ リソース ユーザー ガイド (UG473)
- ザイリンクス 7 シリーズ FPGA コンフィギャブル ロジック ブロック ユーザー ガイド (UG474)
- ザイリンクス 7 シリーズ FPGA GTX トランシーバー ユーザー ガイド (UG476)
- ザイリンクス 7 シリーズ FPGA PCI Express 用インテグレイテッド ブロック v1.3 ユーザー ガイド (UG477)
- ザイリンクス 7 シリーズ FPGA DSP48E1 ユーザー ガイド (UG479)
- ザイリンクス 7 シリーズ FPGA XADC ユーザー ガイド (UG480)

これらのユーザー ガイドおよびその他の関連情報は、ザイリンクスのウェブサイトから入手できます。

http://japan.xilinx.com/support/documentation/7_series.htm

A.3.3 ソフトウェア関連資料

- Zynq EDK コンセプト、ツール、およびテクニック ガイド (UG873)
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug873-zynq-ctt.pdf

  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 IDE Design Suite Embedded Edition の一部として提供されます。Linux ドライバーは、ザイリンクス 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、オペレーション 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 ウェブサイトの Zynq-7000 ページ (http://japan.xilinx.com/products/silicon-devices/epp/zynq-7000) を参照してください。

A.3.4 git 情報

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

A.3.5 デザイン ツール関連資料

- ザイリンクス コマンドライン ツール ユーザー ガイド (UG628)
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_1/devref.pdf
- ザイリンクス ISE マニュアル : http://japan.xilinx.com/support/documentation/dt_ise.htm
- ザイリンクス XPS/EDK でサポートされる IP のウェブサイト : http://japan.xilinx.com/ise/embedded/edk_ip.htm
- ChipScope Pro ソフトウェアおよびコア ユーザー ガイド (UG029)
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_1/chipscope_pro_sw_cores_ug029.pdf
- PlanAhead チュートリアル : ChipScope を使用したデバッグ (UG677)
  http://japan.xilinx.com/support/documentation/sw_manuals/xilinx14_1/PlanAhead_Tutorial_Debugging_w_ChipScope.pdf
- ザイリンクスのトラブルシュート : http://japan.xilinx.com/support/troubleshoot.htm

EDK 関連資料

EDK 関連のすべての資料は、オンラインで入手できます。
http://japan.xilinx.com/support/documentation/dt_edk_edk14-1.htm

各資料のリンクは、次のとおりです。

- SDK オンライン ヘルプ
- 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/xilinx14_1/est_rm.pdf
- OS およびライブラリのドキュメント コレクション (UG643)
  http://japan.xilinx.com/support/documentation/xilinx14_1/oslib_rm.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/xilinx14_1/psf_rm.pdf
- EDK チュートリアルのウェブサイト
  http://japan.xilinx.com/support/documentation/dt_edk_edk14-1_tutorials.htm
- Platform Studio および EDK のウェブサイト
  http://japan.xilinx.com/ise/embedded_design_prod/platform_studio.htm

A.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』Rev 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 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 (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)、プライベート タイマーとウォッチドッグ タイマー (AWDT)、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』
  - ATB バスおよび認証に関する説明が含まれる
- 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』Rev. 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