

Zynq UltraScale+ MPSoC ソフトウェア開発者向けガイド

UG1137 (v2019.1) 2019 年 6 月 26 日

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



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
2019 年 6 月 26 日	v2019.1	<ul style="list-style-type: none"> 第 4 章: <ul style="list-style-type: none"> 表 4-3 を更新。 第 7 章: <ul style="list-style-type: none"> 表 7-6 ～表 7-17 を更新。 第 10 章: <ul style="list-style-type: none"> 「CSU/PMU レジスタへのアクセス」セクションを追加。 表 10-10 を更新。 第 11 章: <ul style="list-style-type: none"> 「コンフィギュレーションオブジェクト」セクションを更新。 「電力管理の初期化」セクションを更新。 付録 A ～付録 L を更新。 新しいライブラリとして、付録 M 「XilMailbox Library v1.0」を追加。
2019 年 1 月 18 日	v9.0	<ul style="list-style-type: none"> 第 2 章: <ul style="list-style-type: none"> 「ブート モード」セクションを更新。 「システム レベル保護」を更新し、UG1085 への参照を追加。 第 3 章: <ul style="list-style-type: none"> 「デバイス ツリーの生成」セクションを追加。 第 4 章: <ul style="list-style-type: none"> XilRSA への参照を削除。 第 8 章: <ul style="list-style-type: none"> 「XMPU レジスタの設定」セクションを更新。 第 10 章: <ul style="list-style-type: none"> PMU ファームウェア ビルド フラグの表に Efuse_Access and PM_LOG_LEVEL を追加。 「電力管理フレームワーク」セクションを更新。 「リスタート トラッカー構造体のメンバー」の表を更新。 「PMU ファームウェアのメトリクス」の表を更新。 第 12 章: <ul style="list-style-type: none"> 「ウォーム リスタート」のセクションにオンチップ メモリ (OCM) に関する注記を追加。 第 16 章: <ul style="list-style-type: none"> 内容を削除し、短い説明を追加。Bootgen ユーザー ガイドへの参照を追加。

日付	バージョン	内容
2018 年 6 月 22 日	v8.0	<ul style="list-style-type: none"> 第 7 章: <ul style="list-style-type: none"> 2019.1 リリースから SHA-2 が非推奨となり、SHA-3 の使用が推奨されることを注記として追加。 第 8 章: <ul style="list-style-type: none"> 「RSA キー取り消しサポートの改善」を追加。 第 10 章: <ul style="list-style-type: none"> 「PLL ロック エラーによって PMU ファームウェアから送信される PS_ERROR_OUT」および「PMU ファームウェアのロード方法」のセクションを更新。
2018 年 5 月 4 日	v7.0	<ul style="list-style-type: none"> 第 8 章: <ul style="list-style-type: none"> 「eFUSE に格納された難読化 (グレー) キーを使用する BIF ファイル」を追加。 SHA-2 認証の非推奨に関する記述を更新。 第 12 章: <ul style="list-style-type: none"> 「ウォーム リスタート」セクションを追加。 第 16 章: <ul style="list-style-type: none"> ブート イメージフォーマットの資料を更新。
2018 年 1 月 19 日	v6.0	<ul style="list-style-type: none"> 第 5 章: <ul style="list-style-type: none"> 「ベアメタル アプリケーションの開発」のセクションに注記を追加。 第 7 章: <ul style="list-style-type: none"> 「ブート フロー」セクションを更新。 「ブート モード」セクションを更新。 第 8 章: <ul style="list-style-type: none"> 「複数の AESKEY ファイルを使用した BIF ファイル」セクションを更新。 第 13 章: <ul style="list-style-type: none"> 「イーサネット フロー」の図を更新。 第 16 章: <ul style="list-style-type: none"> [fsbl_config] パラメーターの例を更新。

日付	バージョン	内容
2017 年 11 月 15 日	v5.0	<ul style="list-style-type: none"> 第 1 章: <ul style="list-style-type: none"> 「前提条件」 セクションを更新。 第 2 章: <ul style="list-style-type: none"> 「ブート プロセス」 セクションを更新。 「セキュリティ」 セクションを更新。 第 4 章: <ul style="list-style-type: none"> 「FreeRTOS ソフトウェア スタック」 セクションを更新。 第 7 章: <ul style="list-style-type: none"> 「FSBL のビルド プロセス」 セクションを追加。 「FSBL コンパイル フラグの設定」 セクションを追加。 「ブート モード」 セクションを更新。 第 8 章: <ul style="list-style-type: none"> 「ブート時のセキュリティ」 セクションを更新。 第 9 章: <ul style="list-style-type: none"> 「PS のプラットフォーム管理」 セクションを更新。 「PMU ファームウェア」 セクションを更新。 第 10 章「プラットフォーム管理ユニット ファームウェア」を追加。 第 11 章: <ul style="list-style-type: none"> 「Zynq UltraScale+ MPSoC の電力管理ソフトウェア アーキテクチャ」 セクションを更新。 「API を使用した電力管理」 セクションを更新。 「コンフィギュレーション オブジェクト」 セクションを更新。 「電力管理の初期化」 セクションを更新。 「API を使用した電力管理」 セクションを更新。 「XiIPM の実装の詳細」 セクションを更新。 第 16 章: <ul style="list-style-type: none"> 「BIF ファイルのパラメーター」 セクションを更新。 「ブート イメージフォーマット」 セクションを更新。 「ブート ヘッダー テーブル」 を更新。 付録 A ～付録 L を更新。 付録 A 「その他のリソースおよび法的通知」 を更新。

日付	バージョン	内容
2017 年 5 月 3 日	v4.0	<ul style="list-style-type: none"> 第 2 章: <ul style="list-style-type: none"> 「ブート プロセス」を追加。 第 4 章: <ul style="list-style-type: none"> 図 4-2 を更新。 Linux ソフトウェア スタックの例外レベル EL0 ～ EL3 に関する情報を追加。 第 5 章: <ul style="list-style-type: none"> 図 5-1 を更新。 第 7 章: <ul style="list-style-type: none"> 「ブート フロー」を第 2 章から第 7 章へ移動。 「QSPI24 および QSPI32 ブート モード」および「eMMC18 ブート モード」を追加。 「JTAG ブート モード」に詳細な情報を追加。 「USB ブート モード」を追加。 図 7-6 を更新。 FSBL_USB_EXCLUDE を表 7-3 に追加。 第 8 章: <ul style="list-style-type: none"> セキュア ブートフローチャートを削除。 「ライブラリ サポート」を削除。このセクションは付録 I 「XilSecure Library v4.0」で説明。 「暗号化」に例を追加。 「認証」に詳細な情報を追加。 「外部メモリを使用するビットストリーム認証」を追加。 「システム メモリ管理ユニット」を追加。 「A53 メモリ管理ユニット」を追加。 「R5 メモリ保護ユニット」を追加。 第 11 章「電力管理フレームワーク」を追加。この内容は以前『Zynq UltraScale+ MPSoC 電力管理フレームワーク ユーザー ガイド』(UG1199)に含まれていた。 第 16 章: <ul style="list-style-type: none"> 表 16-1 に、パラメーターおよび説明を追加。 「ブート イメージのフォーマット」を追加。 表 16-9 に追加ビットの説明を追加。 OS およびライブラリの内容に関する付録を追加 (付録 A ～付録 K)。
2016 年 12 月 15 日	v3.0	<p>第 1 章の「はじめに」に説明を追加。</p> <p>第 7 章の「ブート モード」の説明を修正。</p> <p>『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) への参照リンクを変更。</p> <p>付録 A 「その他のリソースおよび法的通知」を修正してリンクを追加。</p>

日付	バージョン	内容
2016 年 10 月 5 日	v2.0	<p>第 2 章: 図 2-2 から JTAG および MDM を削除。第 2 章のセキュアおよび非セキュア ブートモードの説明を明確化。割り込み機能を削除。</p> <p>第 3 章: 「ハードウェア IDE」機能リストを追加。「Vivado Design Suite」を追加。「ザイリンクス ソフトウェア開発キット (SDK)」のサポートされる機能を変更。SDK_Download へのリンクを追加。PetaLinux の図を表 3-1に差し替え。</p> <p>第 4 章: 図 4-2 を差し替え。「FreeRTOS ソフトウェア スタック」を追加。</p> <p>第 5 章: 「オープン ソースによるソフトウェア開発」を削除。</p> <p>第 6 章: 第 6 章「ソフトウェア デザインのパラダイム」のタイトルを変更。「マルチプロセッサ開発用フレームワーク」セクションを追加。</p> <p>第 7 章: 図 7-3 の SD モードの図を変更。図 7-5 の NAND モードの図を変更。第 7 章「システム ブートおよびコンフィギュレーション」の CSU の主な組織を削除。第 7 章「システム ブートおよびコンフィギュレーション」の復帰メカニズムを削除。第 7 章の「プリブート シーケンス」を追加。</p> <p>第 8 章: 第 8 章「セキュリティ機能」のタイトルを変更し、セクションを再構築。「ザイリンクス ペリフェラル保護ユニット」の説明を修正し、タイトル名を変更。「暗号化」の説明を修正。暗号キーの種類およびキー レジスタの表を削除。『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) への相互参照を変更。「Arm トラステッドファームウェア」を変更。「XMPU によるメモリ保護」から説明を削除。「セキュリティ機能」から「保護チェック」、「エラー処理」、「ペリフェラル保護の使用」セクションを削除。「ライブラリ サポート」を追加。</p> <p>第 9 章: ATF [参照 41] への参照を追加。復帰メカニズムの説明を削除。「電力管理フレームワーク」を追加。「PMU ファームウェア」を変更。</p> <p>第 12 章「DMA」を削除。</p> <p>第 13 章: QEMU 機能表を削除。「ボード/キット」に説明を追加。</p> <p>第 15 章: 「システム コヒーレンシ」を削除。</p> <p>第 16 章「ブート イメージの生成」に付録 A を移動。「BootGen コマンド オプション」から -interface オプションを削除。「仮想化」セクションを削除。『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) への参照を変更。フィールドおよびオフセットテーブルを削除。『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) への参照を変更。ブート アクセスがプログラム可能であることを追加。</p> <p>付録 A「その他のリソースおよび法的通知」から Wiki サイトを削除。</p>
2015 年 11 月 18 日	v1.0	初版

目次

改訂履歴	2
第 1 章: このユーザー ガイドについて	
はじめに	10
対象者および内容	11
前提条件	11
第 2 章: Zynq UltraScale+ MPSoC デバイスのプログラミング ビュー	
はじめに	13
ハードウェア アーキテクチャの概要	14
ブート プロセス	16
仮想化	18
システム レベル リセットの要件	18
セキュリティ	19
安全性と信頼性	22
APU および RPU の実行ファイル用のメモリ概要	25
第 3 章: 開発ツール	
はじめに	27
Vivado Design Suite	28
ザイリンクス ソフトウェア開発キット (SDK)	29
Arm GNU ツール	31
デバイス ツリー ジェネレーター	31
PetaLinux ツール	32
Yocto を使用した Linux ソフトウェア開発	32
第 4 章: ソフトウェア スタック	
はじめに	35
ベアメタル ソフトウェア スタック	35
Linux ソフトウェア スタック	38
サードパーティ ソフトウェア スタック	42
第 5 章: ソフトウェア開発フロー	
ソフトウェア開発フローの概要	43
ベアメタル アプリケーションの開発	44
PetaLinux ツールを使用したアプリケーション開発	46
SDK を使用した Linux アプリケーションの開発	47
第 6 章: ソフトウェア デザインのパラダイム	
はじめに	50
マルチプロセッサ開発用フレームワーク	50

対称型マルチプロセッシング (SMP).....	51
非対称型マルチプロセッシング (AMP).....	52
第 7 章: システム ブートおよびコンフィギュレーション	
はじめに	56
ブート プロセスの概要.....	56
ブート フロー.....	57
ブート イメージの生成.....	58
ブート モード.....	59
ブート フローの詳細.....	64
ブート シーケンスにおける FPD の無効化.....	66
FSBL コンパイル フラグの設定.....	66
FSBL のビルド プロセス.....	70
第 8 章: セキュリティ機能	
はじめに	94
ブート時のセキュリティ	94
外部メモリを使用するビットストリーム認証	105
実行時セキュリティ	107
Arm トラステッド ファームウェア.....	107
FPGA マネージャー ソリューション	111
ザイリンクス メモリ保護ユニット.....	113
ザイリンクス ペリフェラル保護ユニット	113
システム メモリ管理ユニット	113
A53 メモリ管理ユニット	114
R5 メモリ保護ユニット.....	114
第 9 章: プラットフォーム管理	
はじめに	115
PS のプラットフォーム管理.....	115
第 10 章: プラットフォーム管理ユニット ファームウェア	
はじめに	121
特長	121
PMU ファームウェアのアーキテクチャ.....	122
実行フロー	123
PMU ファームウェアにおけるプロセッサ間割り込みの処理.....	124
PMU ファームウェアのモジュール.....	128
エラー管理 (EM) モジュール	131
電力管理 (PM) モジュール	137
スケジューラ	138
セーフティ テスト ライブラリ	138
CSU/PMU レジスタへのアクセス	139
タイマー	140
コンフィギュレーション オブジェクト	142
PMU ファームウェアのロード方法.....	145
PMU ファームウェアの使用法.....	150
PMU ファームウェアのデバッグ.....	157
PMU ファームウェアのメモリ レイアウトとフットプリント	163
依存関係	164

第 11 章: 電力管理フレームワーク

はじめに	165
Zynq UltraScale+ MPSoC の電力管理の概要	167
Zynq UltraScale+ MPSoC の電力管理ソフトウェア アーキテクチャ	169
電力管理フレームワークの概要	170
API を使用した電力管理	180
XilPM の実装の詳細	186
Linux	188
Arm トラストッド ファームウェア (ATF)	201
PMU ファームウェア	205

第 12 章: リセット

はじめに	208
システム レベル リセット	208
ブロック レベル リセット	208
APU のリセット	209
RPU のリセット	209
FPD のリセット	210
ウォーム リスタート	210

第 13 章: 高速バス インターフェイス

はじめに	236
USB 3.0	236
ギガビット イーサネット コントローラー	240
PCI Express	243

第 14 章: クロックおよび周波数管理

はじめに	248
ペリフェラルの周波数変更	248

第 15 章: ターゲット開発プラットフォーム

はじめに	250
QEMU	250
ボード/キット	250

第 16 章: ブート イメージの生成

はじめに	251
------------	-----

付録 A: その他のリソースおよび法的通知

ザイリンクス リソース	252
ソリューション センター	252
Documentation Navigator およびデザイン ハブ	252
参考資料	253
お読みください: 重要な法的通知	255

このユーザー ガイドについて

はじめに

このガイドでは、ザイリンクス Zynq® UltraScale+™ MPSoC デバイス用のシステム ソフトウェアおよびアプリケーションを設計、開発する上で必要なソフトウェア関連情報を提供します。Zynq UltraScale+ MPSoC ファミリには、次のシステム機能に基づいたさまざまな製品があります。

- アプリケーションプロセッシングユニット (APU)
 - デュアル コアまたはクワッド コア Arm Cortex™-A53 MPCore™
 - CPU 周波数は最大 1.5GHz
- リアルタイム プロセッシングユニット (RPU)
 - デュアル コア Arm Cortex-R5F MPCore
 - CPU 周波数は最大 600MHz
- グラフィックス プロセッシングユニット (GPU)
 - Arm Mali™-400 MP2
 - GPU 周波数は最大 667MHz
- ビデオ コーデック ユニット (VCU)
 - 別々のコアを介してエンコードとデコードの同時実行
 - H.264 ハイ プロファイル、レベル 5.2 (4Kx2K-60 レート)
 - H.265 (HEVC)、メイン 10 プロファイル、レベル 5.1、ハイ ティア、4Kx2K-60 の最大レート
 - 8 ビットおよび 10 ビットのエンコーディング
 - 4:2:0 および 4:2:2 のクロマ サンプリング

詳細は、Zynq UltraScale+ MPSoC の製品表 [\[参照 5\]](#) および製品の長所 [\[参照 6\]](#) を参照してください。

対象者および内容

このガイドは、ソフトウェア開発者およびシステム アーキテクトを対象に、次の内容について説明しています。

- ザイリンクスのソフトウェア開発ツール
- 利用可能なプログラミング オプション
- デバイス ドライバー、ミドルウェア スタック、フレームワーク、サンプル アプリケーションなどザイリンクスのソフトウェア コンポーネント
- プラットフォーム管理ユニット ファームウェア (PMUFW)、Arm® トラストッド ファームウェア (ATF)、OpenAMP、PetaLinux ツール、Xen Hypervisor、Zynq UltraScale+ MPSoC デバイス向けに開発されたその他のツール

前提条件

このガイドは、次の経験、知識があることを前提としています。

- エンベデッド ソフトウェアの開発経験
- Armv7 および Armv8 アーキテクチャに関する知識
- Vivado® 統合設計環境 (IDE)、ザイリンクス ソフトウェア開発キット (SDK)、コンパイラ、デバッガー、オペレーティング システムなどザイリンクスの開発ツールに関する知識。

このガイドは次の章で構成されます。

- **第 2 章「Zynq UltraScale+ MPSoC デバイスのプログラミング ビュー」**: Zynq UltraScale+ MPSoC ハードウェアのアーキテクチャについて簡単に説明します。各機能を理解するために、この章をご一読いただくことを推奨します。
- **第 3 章「開発ツール」**: ザイリンクスのソフトウェア開発ツールについて簡単に説明します。ソフトウェア開発ツールの機能に対する理解を深めるのに役立ちます。ソフトウェア開発者はこの章を全体に目を通し、ソフトウェア アプリケーションのビルドおよびデバッグの手順をご確認ください。
- **第 4 章「ソフトウェア スタック」**: Zynq UltraScale+ MPSoC デバイスを使用したシステム開発を支援するためにザイリンクスが提供する各種ソフトウェア スタックとして、ベアメタル ソフトウェア スタック、RTOS ベース ソフトウェア スタック、および完全な Linux スタックについて説明します。
- **第 5 章「ソフトウェア開発フロー」**: ソフトウェア開発プロセスの各手順について説明します。Linux OS およびベアメタルでサポートされる API およびドライバについても簡単に説明します。
- **第 6 章「ソフトウェア デザインのパラダイム」**: ヘテロジニアス プロセッシング システム上でのいくつかのソフトウェア開発アプローチについて説明します。特に対称型マルチプロセッシング (SMP)、非対称形マルチプロセッシング (AMP)、仮想化、および SMP と AMP を組み合わせたハイブリッド モードなど、さまざまなプロセッサ モードでのプログラミングについて詳しく説明します。
- **第 7 章「システム ブートおよびコンフィギュレーション」**: セキュア モードと非セキュア モードの両方において各種ブート デバイスを使用したブート プロセスについて説明します。
- **第 8 章「セキュリティ機能」**: アプリケーションのブート時および実行時のセキュリティを強化するための Zynq UltraScale+ MPSoC デバイスの機能について説明します。

- [第 9 章「プラットフォーム管理」](#): 消費電力の管理に関する機能、およびソフトウェアを使用して各種パワーモードを制御する方法について説明します。
- [第 10 章「プラットフォーム管理ユニット ファームウェア」](#): Zynq UltraScale+ MPSoC デバイス用に開発された PMU ファームウェアの特長と機能について説明します。
- [第 11 章「電力管理フレームワーク」](#): プラットフォーム管理ユニット (PMU) 経由で柔軟な電力管理制御をサポートするザイリンクスの電力管理フレームワーク (PMF) の機能について説明します。
- [第 12 章「リセット」](#): システム レベルおよびモジュールレベルのリセットについて説明します。
- [第 13 章「高速バス インターフェイス」](#): 高速インターフェイス プロトコルのコンフィギュレーション フローについて説明します。
- [第 14 章「クロックおよび周波数管理」](#): Zynq UltraScale+ MPSoC デバイスのペリフェラルのクロックおよび周波数管理について簡単に説明します。
- [第 15 章「ターゲット開発プラットフォーム」](#): Zynq UltraScale+ MPSoC デバイス向けに提供される各種開発プラットフォームとして、Quick Emulator (QEMU) および Zynq UltraScale+ MPSoC ボード/キットについて説明します。
- [第 16 章「ブート イメージの生成」](#): Zynq UltraScale+ MPSoC デバイスのブータブル イメージを作成するためのスタンドアロン ツール Bootgen について説明します。Bootgen は SDK に含まれています。
- [付録 A ～付録 L](#) では、ソフトウェア プラットフォームの開発に役立つ利用可能なライブラリおよびボード サポート パッケージについて説明しています。
- [付録 A「その他のリソースおよび法的通知」](#): このガイドで参考資料として示した文書へのリンクをまとめます。

Zynq UltraScale+ MPSoC デバイスの プログラミング ビュー

はじめに

Zynq® UltraScale+™ MPSoC デバイスは、ヘテロジニアス マルチプロセッシングを必要とするアプリケーションを幅広くサポートします。ヘテロジニアス マルチプロセッシングシステムは、種類の異なるシングルコアおよびマルチコアプロセッサを複数組み合わせることで構成されます。このデバイスは次の機能をサポートしています。

- 複数レベルのセキュリティ
- 高度な安全性
- 高度な電力管理
- 広帯域のプロセッサ、I/O、およびメモリ
- ヘテロジニアス マルチプロセッシングに基づくアプローチで直面する設計には、次のような課題があります。
 - 消費電力の仕様を満たしながらアプリケーション パフォーマンスの要件を満たすこと
 - ヘテロジニアス マルチプロセッシングシステム内でメモリ アクセスを最適化すること
 - プロセッシング エンジン間で低レイテンシかつコヒーレントな通信を行うこと
 - すべての動作モードでシステム消費電力を管理および最適化すること

ザイリンクスは、Zynq UltraScale+ MPSoC デバイスでのハードウェア/ソフトウェア開発やオペレーティング システム、ヘテロジニアス システム ソフトウェア、セキュリティ管理モジュールなどのさまざまなソフトウェア モジュールに対応できる包括的なツールを提供しています。

Zynq UltraScale+ MPSoC デバイスは、高性能で高エネルギー効率の 64 ビット アプリケーションプロセッサである Arm® Cortex™-A53、および 32 ビットの Arm Cortex-R5F デュアルコア リアルタイム プロセッサを搭載するヘテロジニアス デバイスです。

ハードウェア アーキテクチャの概要

Zynq UltraScale+ MPSoC デバイスは、消費電力の削減、プログラマブル アクセラレーション、I/O、およびメモリ帯域幅が強化されており、ヘテロジニアス マルチプロセッシングを必要とするアプリケーションに最適です。

図 2-1 に、セキュリティ、セーフティ、信頼性、およびスケーラビリティを兼ね備える次世代プログラマブル エンジン内蔵の Zynq UltraScale+ MPSoC アーキテクチャを示します。

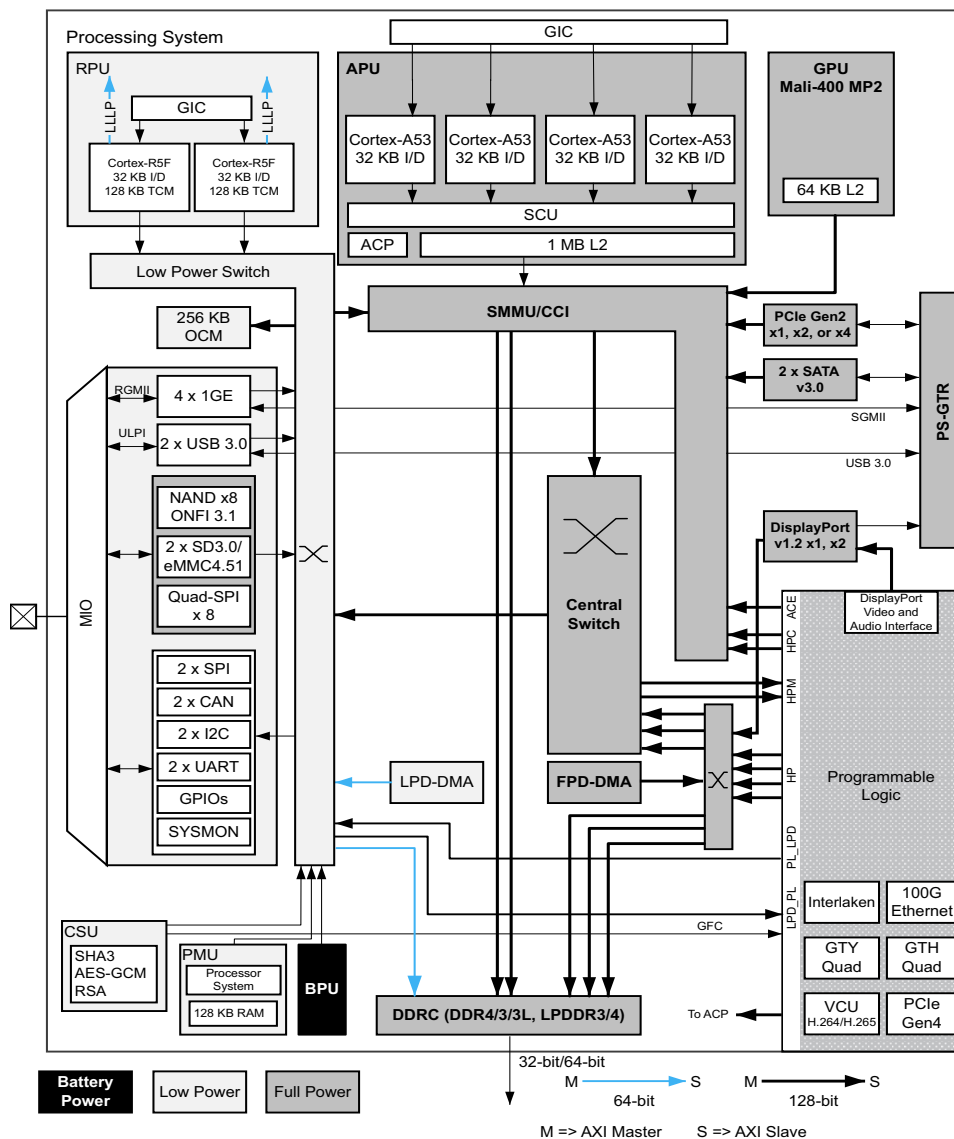


図 2-1: Zynq UltraScale+ MPSoC デバイスのハードウェア アーキテクチャ

Zynq UltraScale+ MPSoC デバイスには次の機能があります。

- Cortex-R5F デュアルコア リアルタイムプロセッシング ユニット (RPU)
- Arm Cortex-A53 64 ビット クワッド/デュアルコア アプリケーション プロセッシング ユニット (APU)
- Mali-400 MP2 グラフィックス プロセッシング ユニット (GPU)
- 外部メモリ インターフェイス: DDR4、LPDDR4、DDR3、DDR3L、LPDDR3、2x クワッド SPI、および NAND
- 汎用コネクティビティ: 2x USB 3.0、2x SD/SDIO、2x UART、2x CAN 2.0B、2x I2C、2x SPI、4x 1GE、および GPIO
- セキュリティ: AES (Advanced Encryption Standard)、RSA 公開キー暗号アルゴリズム、および SHA-3 (Secure Hash Algorithm-3)
- AMS システム モニター: 10 ビット、1 MSPS ADC、温度、電圧、および電流モニター
- プロセッシング システム (PS) には、次のプロトコルをサポートする 5 つの高速シリアル I/O (HSSIO) インターフェイスがあります。
 - PCIe: ベース仕様バージョン 2.1 準拠、Gen2x4
 - SATA 3.0
 - DisplayPort: 最大ビデオ解像度 4k x 2k に対応した DisplayPort インターフェイス (ソースのみ) を実装
 - USB 3.0: USB 3.0 仕様に準拠し、5Gb/s ライン レートを実装
 - シリアル GMII: 1Gb/s SGMII インターフェイスをサポート
- 電源シーケンス、安全性、セキュリティ、およびデバッグの機能を支えるプラットフォーム管理ユニット (PMU)。

詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の次のセクションを参照してください: [APU](#)、[RPU](#)、[PMU](#)、[GPU](#)、およびプロセッサ間割り込み (IPI)。

その他のコンポーネントについては、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] を参照してください。

ブート プロセス

複数ステージからなるブート プロセスは、プラットフォーム管理ユニット (PMU) とコンフィギュレーション セキュリティ ユニット (CSU) で管理および実行されます。デバイスはセキュア モードまたは非セキュア モードでブートできます。「[ブート プロセスの概要](#)」、または『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「ブートおよびコンフィギュレーション」の章を参照してください。

ブート モード

外部デバイスからのブートでは、次のいずれかのブート モードを使用できます。

- クワッド SPI フラッシュ メモリ (QSPI24、QSPI32)
- eMMC18
- NAND
- SD (Secure Digital) インターフェイス メモリ (SD0、SD1)
- JTAG
- USB

bootROM は SATA、イーサネット、および PCI Express (PCIe) からのブートを直接サポートしていません。ブート セキュリティは TrustZone (TZ) に依存しておらず、これらはほぼ独立しています。bootROM はプラットフォーム管理ユニット上で動作し、キー管理などのセキュリティ リソース管理を実行し、信頼のルートを確立します。bootROM は FSBL を認証し、ブート セキュリティ リソースをロックし、トラスト チェーンの制御を APU または RPU 上の FSBL に渡します。

各種ブート モードにおけるブート プロセスの詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「ブートおよびコンフィギュレーション」の章を参照してください。

QSPI24 および QSPI32

QSPI ブート モードは、次の機能をサポートします。

- シングル QSPI フラッシュ メモリ (QSPI24) では x1、x2、および x4 読み出しモード。デュアル QSPI では x8 読み出しモード。
- マルチブートのイメージ検索。
- I/O モードは FSBL でサポートされません。

注記: シングル クワッド SPI メモリ (x1、x2 および x4) は、XIP (eXecute-In-Place) をサポートする唯一のブート モードです。

詳細は、「[QSPI24 および QSPI32 ブート モード](#)」を参照してください。

eMMC18

eMMC18 ブート モードは次の機能をサポートします。

- FAT 16 および FAT 32 ファイル システムによるブート イメージの読み出し。
- マルチブートのイメージ検索。マルチブートで検索可能なイメージ ファイルの最大数は 8,191。

詳細は、「[eMMC18 ブート モード](#)」を参照してください。

NAND

NAND ブート モードは、次の機能をサポートします。

- 8 ビット幅でのブート イメージの読み出し。
- マルチブートのイメージ検索。

詳細は、「[NAND ブート モード](#)」を参照してください。

SD

SD ブートではバージョン 3.0 がサポートされます。このバージョンは、次の機能をサポートします。

- FAT 16/32 ファイル システムによるブート イメージの読み出し。
- マルチブートのイメージ検索。マルチブートで検索可能なイメージ ファイルの最大数は 8,191。

詳細は、「[SD ブート モード](#)」を参照してください。

JTAG

PS に必要なソフトウェア イメージと PL に必要なハードウェア イメージは、JTAG を使用してダウンロードできます。



重要: JTAG モードの場合、Zynq UltraScale+ MPSoC デバイスは非セキュア モードでのみブート可能です。

詳細は、「[JTAG ブート モード](#)」を参照してください。

USB

USB ブート モードは USB 3.0 をサポートします。マルチブート、イメージのフォールバック、XIP はサポートされません。セキュア ブートと非セキュア ブートのどちらのモードもサポートされます。USB ブート モードは DDR なしのシステムではサポートされません。USB ブート モードはデフォルトで無効となっています。詳細は、「[USB ブート モード](#)」を参照してください。

仮想化

仮想化により、1つのプロセッサで複数のソフトウェア スタックを同時実行できるため、Zynq UltraScale+ MPSoC デバイスの生産性が向上します。仮想化の役割はシステムによってそれぞれ異なります。あるシステムでは、仮想化によってプロセッサの使用率を常に高く維持し、消費電力の削減と性能の最大化を実現しています。また別のシステムでは、独立性や冗長性を確保するために各種のソフトウェア スタックを分割する手段として仮想化を利用しています。

詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「システム仮想化」のセクションを参照してください。

仮想化サポートは、Arm 例外レベル 2 (EL2) を含むインプリメンテーションにのみ該当します。Armv8 は仮想化拡張機能によって完全仮想化をサポートしているため、ほぼネイティブなゲスト オペレーティング システムの性能が得られます。仮想化の対象となるハードウェア コンポーネントには、主に次の 3 つがあります。

- CPU 仮想化
- 割り込み仮想化
- I/O 仮想化のためのシステム MMU

システム レベル リセットの要件

「システム レベル リセット」とは、システムまたはサブシステム レベルのリセットを表す用語として使用します。「システム リセット」とは、「システム レベル リセット」の種類の 1 つです。表 2-1 に、システム レベル リセットをまとめます。それぞれの詳細は、この後のセクションで説明します。

表 2-1: システム レベル リセット

リセットの種類	説明
外部 POR	外部 POR リセットは、外部ピンのアサートによってトリガーされます。POR リセットの影響を受けないソフトウェアのみのレジスタが多数存在します。PS と PL で HFT1 が要求される安全システムの場合、2 回目以降の POR では PS のみがリセットされ、PL はリセットされないように最初の POR ブートで設定できます。
内部 POR	内部 POR リセットは、ソフトウェアによるレジスタ書き込み、またはセーフティエラーによってトリガーできます。エラー ステータス レジスタは外部 POR でリセットされますが、内部 POR ではリセットされません。それ以外は、内部 POR と外部リセットでリセットされるものは同じです。内部 POR は、突入電力の懸念があるためシリコン検証を実施しないと保証されません。このため、検証未実施の場合、内部 POR は社内用途に限られます。
システム リセット	システム リセットでは、デバッグ ロジックを除くシステム全体をリセットできます。システム リセットを簡略化するため、このリセットの影響を受けないものはいくつかあります (xBIST、スキャン クリア、パワー ゲーティングなど)。また、ブート モードの情報もシステム リセットではリセットされません。システム リセットは、外部ピン (SRST)、ソフトウェアによるレジスタ書き込み、またはセーフティ エラーによってトリガーできます。

表 2-1: システム レベル リセット (続き)

リセットの種類	説明
PS 専用のリセット	PS 専用のリセットは、PL を動作させたまま PS をリセットします。このリセットは、ハードウェア エラー信号またはソフトウェアによるレジスタ書き込みによってトリガーできます。このリセットは、システム リセットから PL リセットを除いたものです。エラー信号によって PS リセットがトリガーされた場合、そのエラーは PL にも送信されます。
FPD リセット	FPD リセットは、フル電力ドメイン (FPD) 全体をリセットします。このリセットは、エラーまたはソフトウェアによるレジスタ書き込みによってトリガーできます。エラー信号によって FPD リセットがトリガーされた場合、そのエラーは LPD と PL にも送信されます。
RPU リセット	RPU リセットは、エラー発生時に RPU をリセットします。各 R5 コアは個別にリセットできますが、ロックステップ モードでは、R5_0 をリセットすると両方の R5 コアがリセットされます。このリセットは、エラーまたはソフトウェアによるレジスタ書き込みによってトリガーできます。

セキュリティ

ザイリンクス デバイスの採用はあらゆる分野へと広がっており、デバイスで処理するデータを保護するだけでなく、デバイス内の IP を保護することも同じくらい重要になっています。セキュリティの脅威が高まる中、セキュアな製品を運用していく上で考慮すべきセキュリティ脅威と潜在的な脆弱性も増えています。

Zynq UltraScale+ MPSoC には、SoC で動作するアプリケーションのセキュリティを強化する次の機能があります。

- コンフィギュレーション ファイルの暗号化と認証。
- ユーザー アプリケーションで利用できるハード マクロの暗号アクセラレータ。
- 暗号化キーの安全な格納方法。

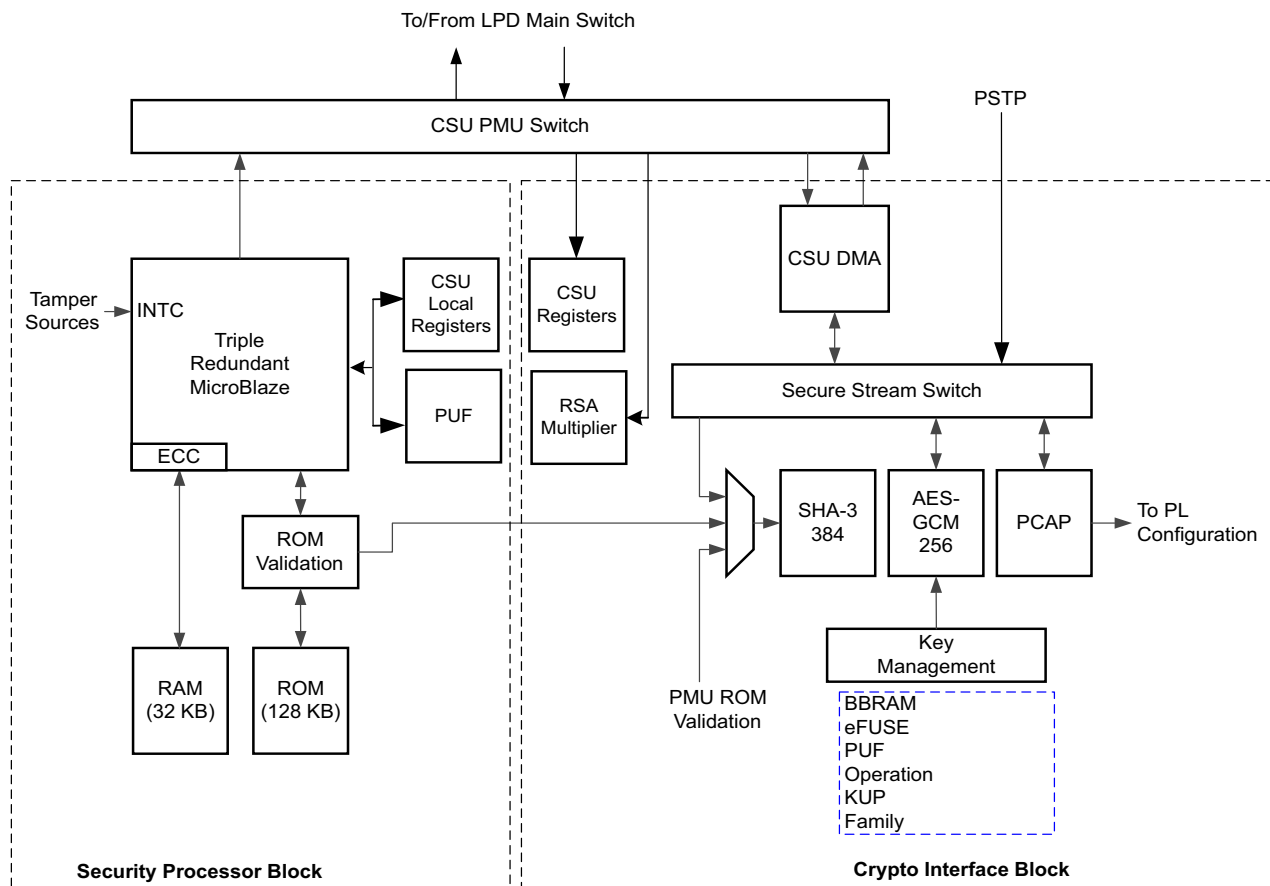
タンパー イベントの検出と応答の方法。詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「セキュリティ」の章を参照してください。

コンフィギュレーション セキュリティ ユニット (CSU)

コンフィギュレーション セキュリティ ユニット (CSU) が担う重要な役割には、次のものが含まれます。

- セキュア ブート。
- タンパーの監視と応答。
- セキュアなキーの格納および管理。
- 暗号化のハードウェア アクセラレーション。

図 2-2 に示すように、CSU は 2 つの主要ブロックで構成されます。左側のブロックは、ブート動作を制御する三重冗長プロセッサを備えているセキュアプロセッサブロックです。また、付属 ROM、小型専用 RAM、およびすべてのセキュア動作をサポートするために必要な制御/ステータスレジスタもあります。右側のブロックは暗号インターフェイスブロック (CIB) で、AES-GCM、DMA、SHA、RSA、および PCAP インターフェイスを備えています。



X15318-032817

図 2-2: コンフィギュレーション セキュリティ ユニットのアーキテクチャ

- ブート後、CSU はタンパーレスポンスの監視を実行します。これらの暗号インターフェイスは動作中に使用できます。これら機能の使用方法は、付録 L 「XilFPGA Library v5.0」を参照してください。詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「セキュリティ」の章を参照してください。
- セキュアプロセッサブロック:** 三重冗長プロセッサアーキテクチャにより、シングルイベントアップセット (SEU) が発生した場合の CSU の動作が強化されます。
- 暗号インターフェイスブロック (CIB):** AES-GCM、DMA、SHA-3/384、RSA、および PCAP インターフェイスで構成されます。
- AES-GCM:** AES-GCM コアは、32 ビットワードベースのデータインターフェイスを備えており、256 ビットキーをサポートします。
- キー管理:** AES を使用するには、AES ブロックにキーを読み込む必要があります。キーは CSU の bootROM で選択します。
- SHA-3/384:** SHA-3/384 エンジン、認証の際に入力イメージのハッシュ値の計算に使用します。
- RSA-4096 アクセラレータ:** RSA 認証を高速化します。

ブート イメージ暗号化または認証の詳細は、次を参照してください。

- [第 7 章「システム ブートおよびコンフィギュレーション」](#)
- [第 16 章「ブート イメージの生成」](#)。
- 『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「セキュリティ」の章。
- 『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「ブートおよびコンフィギュレーション」の章。

システム レベル保護

システム レベル保護メカニズムには、次の領域があります。

- Zynq UltraScale+ MPSoC システム ソフトウェア スタックは、Arm トラステッド ファームウェア (ATF) 上で動作します。XMPU および XPPU を設定してシステム レベルの実行時セキュリティを確保することで、より強力な保護を実現します。
 - 。 バグのある、または悪意のあるソフトウェア (誤ったソフトウェア) によるシステム メモリの破損またはシステム障害の発生を防止します。
 - 。 正しくプログラムされていない、または悪意のあるデバイス (誤ったハードウェア) によるシステム メモリの破損またはシステム障害の発生を防止します。
 - 。 メモリ (DDR、OCM) およびペリフェラル (ペリフェラル制御、SLCR) を誤ったソフトウェアまたはハードウェアによる不正アクセスから保護してシステムを保護します。
- ザイリンクス メモリ保護ユニット (XMPU): メモリを分割し、メモリおよび FPD スレーブに対して TrustZone (TZ) 保護を適用します。XMPU は、1 つまたは複数のマスターからのアクセスを開発者が定義したアドレス範囲に限定するように設定できます。
- ザイリンクス ペリフェラル保護ユニット (XPPU): LPD ペリフェラルを分離し、プロセッサ間割り込み (IPI) を保護します。XPPU は、1 つまたは複数のマスターに対して LPD ペリフェラルへのアクセスを許可するように設定できます。詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「XPPU によるスレーブの保護」のセクションを参照してください。

安全性と信頼性

Zynq UltraScale+ MPSoC アーキテクチャは、安全性が重視されるアプリケーションの信頼性を高める機能を備えており、ユーザーと設計者の双方にとってシステム信頼性が向上します。主な機能には、次のものがあります。

- メモリおよびキャッシュのエラー検出と訂正
- RPU の安全機能
- システム全体の安全機能

これら機能の使用方法は、[第 8 章「セキュリティ機能」](#)を参照してください。

安全機能

Cortex-A53 MPCore プロセッサは、次に示す別々のエレメントを使用して、プロセッサ内のすべての RAM インスタンスに対して ECC によるキャッシュ保護をサポートしています。

- SCU-L2 キャッシュ保護
- CPU キャッシュ保護

これらのエレメントにより、Cortex-A53 MPCore プロセッサはすべての RAM で 1 ビット エラーの検出と訂正、および 2 ビット エラーの検出を可能にしています。

Cortex-A53 MPCore の RAM はシングル イベント アップセット (SEU) から保護され、プロセッサ システムはエラーを検出してもデータ破損なしに処理を継続するためのアクションを実行できます。RAM の種類により、パリティによるシングル エラー検出 (SED) に対応したものと、ECC によるシングル エラー訂正、ダブル エラー検出 (SEDED) に対応したものがあります。

RPU には 2 つの主要な安全機能があります。

- ロックステップ動作 ([図 2-3](#) 参照)。
- ECC ([「ECC \(エラー チェック訂正\)」](#) 参照)。

ロックステップ動作

Cortex-R5F プロセッサは、2 つの RPU CPU が冗長構成で動作するロックステップ動作をサポートしています。これをセーフティ モードと呼びます。

Cortex-R5F プロセッサの動作をロックステップ モードに設定した場合、片方の CPU インターフェイスのみを使用します。

Cortex-R5F プロセッサはスプリット モードとロックステップ モードの動的な変更をサポートしておらず、これらモードの切り替えはプロセッサ グループがパワーオン リセット (POR) 状態に保持されている間のみ許可されます。プロセッサ グループの動作モードは、入力信号 SLCLAMP と SLSPLIT で制御します。

ロックステップ モードでは、これらの信号によって多重化およびクランプ ロジックが制御されます。次の図に示すように、Cortex-R5F プロセッサがロックステップ モードの場合、GIC 内のディストリビューターが CPU0 に対してのみ割り込みを発行するように、リセット ハンドラーにコードを記述しておく必要があります。

RPU は、Cortex™-R5F MPCore プロセッサの専用割り込みコントローラーを備えています。この Arm® PL390 ジェネリック割り込みコントローラー (GIC) は、GICv1 規格に基づいています。

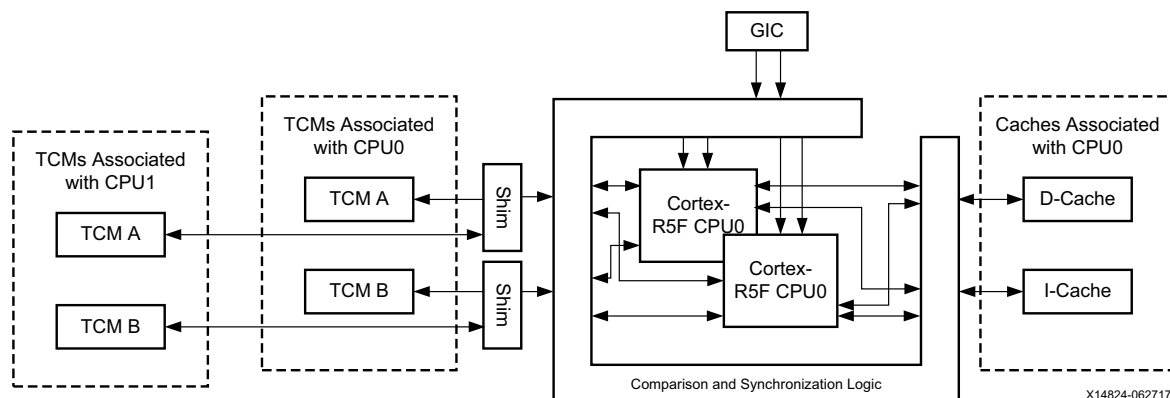


図 2-3: RPU のロックステップ動作

密結合メモリ (TCM) は各 Cortex-R5F プロセッサのローカルアドレス空間にマップされますが、同時にグローバルアドレス空間にもマップされるため、XPPU の設定でアクセスが許可されている場合、すべてのマスターからアクセスできます。

次の表に、RPU から見たアドレス マップを示します。

表 2-2: RPU のアドレス マップ

動作モード	メモリ	R5_0 から見たマップ (開始アドレス)	R5_1 から見たマップ (開始アドレス)	グローバル アドレスから 見たマップ (開始アドレス)
スプリット モード	R5_0 ATCM (64KB)	0x0000_0000	N/A	0xFFE0_0000
	R5_0 BTCM (64KB)	0x0002_0000	N/A	0xFFE2_0000
	R5_0 命令 キャッシュ	I-キャッシュ	N/A	0xFFE4_0000
	R5_0 データ キャッシュ	D-キャッシュ	N/A	0xFFE5_0000
スプリット モード	R5_1 ATCM (64KB)	N/A	0x0000_0000	0xFFE9_0000
	R5_1 BTCM (64KB)	N/A	0x0002_0000	0xFFEB_0000
	R5_1 命令 キャッシュ	I-キャッシュ	N/A	0xFFEC_0000
	R5_1 データ キャッシュ	D-キャッシュ	N/A	0xFFED_0000
ロック ステップ モード	R5_0 ATCM (128KB)	0x0000_0000	N/A	0xFFE0_0000
	R5_0 BTCM (128KB)	0x0002_0000	N/A	0xFFE2_0000
	R5_0 命令 キャッシュ	I-キャッシュ	N/A	0xFFE4_0000
	R5_0 データ キャッシュ	D-キャッシュ	N/A	0xFFE5_0000

ECC (エラー チェック訂正)

Cortex-R5F プロセッサは、ECC (エラー チェック訂正) によるデータ保護をサポートしています。データの特性は同じですが、ECC を適用するデータ チャンクのサイズが異なります。

境界に揃ったデータ チャンクごとにプロセッサが冗長コード ビットの数进行計算し、データと一緒に格納します。これにより、プロセッサはデータ チャンクまたはコード ビット内の最大 2 つのエラーを検出し、データ チャンクまたは関連するコード ビット内の 1 つのエラーを訂正できます。これは SEC-DED (Single-Error Correction、Double-Error Detection) ECC 方式とも呼ばれます。

システム全体の安全機能

システム全体の安全機能は、Zynq UltraScale+ MPSoC の動作でエラーが発生しないように設計されています。

これらの機能を次に示します。

- [プラットフォーム管理ユニット \(PMU\)](#)
- [PMU の三重冗長プロセッサ](#)

以降のセクションでは、これら機能について説明します。

プラットフォーム管理ユニット (PMU)

Zynq UltraScale+ MPSoC デバイスのプラットフォーム管理ユニット (PMU) は、ROM および RAM からロードしたコードをフラットなメモリ空間で実行し、PS の電圧レールに対する改ざん防止を目的とした電源の安全性に関するルーチンを実装し、ロジック BIST (LBIST) の実行やユーザー駆動の電力管理シーケンスへの応答などの処理を担います。

PMU にはデバイスの動作と安全性に関する重要な機能を制御するレジスタもあります。安全に関連するレジスタには、次のものがあります。

- GLOBAL_RESET: 安全関連ブロックに対するリセットを格納します。
- SAFETY_GATE: ハードウェア機能が誤って有効化されないようにゲーティングします。
- SAFETY_CHK: 安全性アプリケーションのターゲット レジスタに対して周期的に書き込みと読み出しを実行してインターコネクト データ ラインのインテグリティをチェックします。

PMU の三重冗長プロセッサ

プラットフォーム管理ユニット (PMU) には三重冗長プロセッサがあり、高度なシステム信頼性と強力な SEU 耐性を実現します。PMU は、システム全体のリソースに対するパワーアップ、パワーダウン、および監視を制御します。PMU は、次のような複数のタスクを実行します。

- ブート時のシステム初期化
- 各種電源ドメインおよび電源アイランドの電力ゲーティングとリテンション ステートの管理
- 外部電源制御デバイスへの電源設定の通信
- ディープスリープ モードを含む各種スリープ ステートの管理および復帰機能の処理

PMU の詳細は、[第 9 章「プラットフォーム管理」](#)を参照してください。

割り込み

割り込みは汎用割り込みコントローラー (GIC) が処理します。APU と RPU には、割り込み処理用にそれぞれ専用の GIC があります。

RPU は、柔軟性と保護機能を考慮して GICv1 仕様に基づく Arm PL390 GIC を実装しています。

APU は GICv2 コントローラーを実装しています。GICv2 は、マルチプロセッサ システムで割り込みをサポートおよび管理するためのリソースを集約しています。また、プロセッサ仮想化をサポートしたシステムでの GIC 実装を可能にする GIC 仮想化拡張をサポートしています。

Zynq UltraScale+ MPSoC デバイスは、ヘテロジニアス プロセッサ間の通信をサポートするプロセッサ間の割り込み (IPI) ブロックを実装しています。PMU は種類の異なるプロセッサと同時に通信できるため、PMU には 4 つの IPI があり、これらは PMU の GIC に接続されます。

各種プロセッサに対する IPI の配線の詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「割り込み」の章を参照してください。

APU および RPU の実行ファイル用のメモリ概要

APU および RPU で設定可能なメモリ領域を次の各表に示します。

次に注意事項を示します。

- ロックステップ モードの RPU (「[ロックステップ動作](#)」) では、R5_0_ATCM_MEM_0 および R5_0_BTCM_MEM_0 メモリ アドレスはシステム アドレス マップの R5_0_ATCM_LSTEP および R5_0_BTCM_LSTEP のメモリ範囲にそれぞれマップされます。
- スプリット モードの RPU では、R5_x_ATCM_MEM_0 および R5_x_BTCM_MEM_0 のメモリ アドレスはシステム アドレス マップの R5_x_ATCM_SPLIT および R5_x_BTCM_SPLIT のメモリ範囲にそれぞれマップされます。
- QSPI コントローラーがリニア モードの場合 QSPI メモリにアクセス可能です。

詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「システム アドレス」の章を参照してください。

RPU、R5 および OCM の詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「リアルタイムプロセッシングユニット (RPU)」および「オンチップ メモリ」の章を参照してください。

表 2-3: APU で設定可能なメモリ領域

メモリ タイプ	開始アドレス	サイズ
DDR Low	0x00000000	2GB
DDR High	0x80000000	2GB
OCM	0xFFFFC000	256KB
QSPI	0xC0000000	512MB

表 2-4: RPU (ロックステップ モード) で設定可能なメモリ領域

メモリ タイプ	開始アドレス	サイズ
DDR Low	0x100000	2047MB
OCM	0xFFFC0000	256KB
QSPI	0xC0000000	512MB
R5_0_ATCM_MEM_0	0x00000	64KB
R5_0_BTCM_MEM_0	0x20000	64KB
R5_TCM_RAM_0_MEM	0x00000	256KB

表 2-5: RPU (スプリット モード) で設定可能なメモリ領域

メモリ タイプ	開始アドレス	サイズ
R5_0		
DDR Low	0x100000	2047MB
OCM	0xFFFC0000	256KB
QSPI	0xC0000000	512MB
R5_0_ATCM_MEM_0	0x00000	64KB
R5_0_BTCM_MEM_0	0x20000	64KB
R5_1		
DDR Low	0x100000	2047MB
OCM	0xFFFC0000	256KB
QSPI	0xC0000000	512MB
R5_1_ATCM_MEM_0	0x00000	64KB
R5_1_BTCM_MEM_0	0x20000	64KB

注記: bootROM は常に 0xFFFC0000 から FSBL (第 1 段階ブートローダー) をコピーします。この設定は変更できません。CSU bootROM (CBR) はブート イメージのパーティション ヘッダーを解析せず、単に固定された OCM メモリ アドレス (0xfffc0000) の FSBL コードをコピーするだけです。このため、これ以外のロード アドレスにコンパイルされている FSBL を Bootgen は拒否します。Bootgen の詳細は、[第 7 章「システム 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。

開発ツール

はじめに

この章では、Zynq® UltraScale+™ MPSoC デバイスソフトウェアのプログラミングに使用するザイリンクス ツールおよびフローについて説明します。ザイリンクス ツールは Eclipse ベース統合開発環境 (IDE) や GNU コンパイラ ツールチェーンなどの一般的なコンポーネントを採用しているため、この章の内容の多くはサードパーティ ツールにも当てはまります。

この章では、Zynq UltraScale+ MPSoC デバイスの各種プロセッサを対象にしたオープン ソース開発に利用できるオープン ソース ツールについても簡単に説明します。

Zynq UltraScale+ MPSoC デバイスで動作するソフトウェア アプリケーションの開発とデバッグには、次のような包括的なツール セットが用意されています。

- ハードウェア IDE
- 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 (例: QEMU)
- モデルおよびバーチャル プロトタイピング ツール (例: エミュレーション ボード プラットフォーム)

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 UltraScale+ MPSoC デバイスの直接サポート レベルがさまざまです。

後続のセクションでは、ザイリンクスの開発ツールの概要を説明します。

Vivado Design Suite

ザイリンクスの Vivado® Design Suite には、Vivado 統合設計環境 (IDE) に含まれるツールがあります。IDE は、優れた機能を備えた直感的なグラフィカル ユーザー インターフェイス (GUI) を提供します。

Vivado Design Suite は、ザイリンクス ISE ソフトウェアに代わる設計環境ですが、システムオンチップ開発や高位合成などの機能が追加されています。この環境は、システム レベルの統合やインプリメンテーションでの生産性を高めるためにザイリンクスが開発した、SoC デバイス対応で IP やシステムを中心とする次世代開発環境です。

Vivado Design Suite 内のすべてのツールおよびツール オプションは、ネイティブ ツール コマンド言語 (Tcl) で記述されているため、Vivado IDE と Vivado Design Suite Tcl シェルの両方で利用可能です。解析や制約の割り当ては、設計プロセス全体で可能です。たとえば、合成後、配置後、配線後のいつでもタイミングや消費電力の見積もりを実行できます。データベースは Tcl を使用してアクセスできるため、インプリメントし直さなくても制約、デザイン構成、およびツール設定をリアルタイムに変更できます。

Vivado IDE では、メモリ内でデザインを開くというコンセプトを導入しています。デザインを開くと、デザイン フローのその特定段階でのネットリストが読み込まれ、制約がデザインに割り当てられ、デザインがターゲット デバイスに適用されます。これにより、デザインを各段階で視覚化して処理できます。



重要: Vivado IDE では、7 シリーズ以降のデバイスをターゲットとするデザインのみがサポートされます。

Vivado Design Suite の次の機能を利用することで、デザインの性能や扱いやすさを向上させることができます。

- IP インテグレーターのプロセッサ コンフィギュレーション ウィザード (PCW) では、グラフィカル ユーザー インターフェイスを使用して IP インテグレーター ブロック デザイン内で PS の作成および変更が可能です。



ビデオ: PCW の詳細は、QuicTake ビデオ: 「[Vivado Processor Configuration Wizard の概要](#)」をご覧ください。

- レジスタ転送レベル (RTL) デザイン (VHDL、Verilog、SystemVerilog)
- Vivado IP インテグレーターでザイリンクスの IP カタログからコアをすばやく統合および設定し、ブロック デザインを作成
- Vivado 合成
- C 言語ベースのソース (C、C++、SystemC)
- 配置および配線を実行する Vivado インプリメンテーション
- デバッグ用の Vivado シリアル I/O およびロジック アナライザー
- Vivado 消費電力解析
- タイミング制約を入力する SDC ベースの XDC (Xilinx Design Constraints)
- スタティック タイミング解析
- 柔軟なフロアプランニング
- 配置および配線の詳細変更
- ビットストリーム生成
- Vivado Tcl Store - Vivado で簡単に機能を追加したり変更が可能

Vivado Design Suite は、ザイリンクスの「Vivado Design Suite – HLx Editions」[\[参照 3\]](#) からダウンロード可能です。

ザイリンクス ソフトウェア開発キット (SDK)

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。この中には、GNU ベースのコンパイラ ツールチェーン、JTAG デバッガー、フラッシュ プログラマ、ミドルウェア ライブラリ、ベアメタル BSP、およびザイリンクス IP 用ドライバが含まれます。また、C/C++ ベアメタルおよび Linux アプリケーション開発とデバッグ用の強力な IDE も含まれます。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。

SDK では、Arm® Cortex™-A53 および Cortex-R5F プロセッサ、さらにはザイリンクス MicroBlaze™ プロセッサ用に統合された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作成できます。また、アプリケーションを作成するための、次のようなさまざまな手段を提供しています。

- MicroBlaze 用のベアメタルおよび FreeRTOS アプリケーション
- APU 用のベアメタル、Linux、および FreeRTOS アプリケーション
- RPU 用のベアメタルおよび FreeRTOS アプリケーション
- PMU ファームウェアのユーザー カスタマイズ
- 次のライブラリ サンプル (すぐにソースをロードして構築可能)
 - OpenCV
 - OpenAMP RPC
 - FreeRTOS “HelloWorld”
 - lwIP
 - パフォーマンス テスト (Dhrystone、メモリ テスト、ペリフェラル テスト)
 - 画像やビットストリームの改ざんや変更を防ぐための RSA 認証
 - APU または RPU 用の FSBL (第1段階ブートローダー)。

Vivado の Project Navigator からブロック デザイン、ハードウェア デザイン ファイル、およびビットストリーム ファイルを SDK のエクスポート ディレクトリに直接エクスポートできます。Vivado Design Suite の詳細は、Vivado Design Suite に関する資料 [\[参照 22\]](#) を参照してください。

このエクスポート プロセスを完了させるために必要なすべてのプロセスは自動で実行されます。ザイリンクス SDK は、ワークスペース内に次のファイルを含む新しいハードウェア プラットフォーム プロジェクトを作成します。

- .project: プロジェクト ファイル
- psu_init.tcl: PS 初期化スクリプト
- psu_init.c、psu_init.h: PS 初期化コード
- psu_init.html: レジスタ サマリ ビューアー
- system.hdf: ハードウェア定義ファイル

コンパイラは、次のように切り替え可能です。

- 32 ビットまたは 64 ビット (Cortex-A53 をターゲットとするアプリケーション)
- 32 ビットのみ (Cortex-R5F、およびザイリンクス MicroBlaze をターゲットとするアプリケーション)

ビルド手順の一覧は、ザイリックス ソフトウェア開発キットのヘルプ [参照 24] を参照してください。SDK を起動した後、[ヘルプ] からさらなる検索が可能です。

また、ザイリックス SDK には、ザイリックス エンベデッド ソフトウェア開発に使用できる次のツールも含まれます。

- 。 **Xilinx System Debugger (XSDB):** システム デバッガー GUI およびコマンド ライン インターフェイス経由でザイリックスの hw_server を利用できます。また、ザイリックス SDK では直接利用できない低レベルのデバッグ機能も各種揃っています。
- 。 **FPGA プログラマ:** ザイリックス デバイスにビットストリームを書き込みます。
- 。 **フラッシュ プログラマ:** ビットストリームおよびソフトウェア アプリケーション イメージを外部パラレル NOR フラッシュ デバイスに書き込みます。
- 。 **リンカー スクリプト ジェネレーター:** アプリケーション イメージをハードウェア メモリ空間にマップします。
- 。 **ブート イメージジェネレーター:** ブートローダー、ビットストリーム、ユーザー アプリケーションを結合してブート イメージを生成します。オプションで認証および暗号化を有効にもできます。
- 。 **ザイリックス ソフトウェア コマンド ライン ツール (XSCT):** ザイリックス ソフトウェア コマンド ライン ツールは Tcl スクリプトをベースとし、SDK のインストール時に提供されます。ユーザーは、このツールで Tcl プロンプトを開き、サポートされているすべてのコマンドを使用できます。

ザイリックス ソフトウェア コマンド ライン ツールは、ザイリックス SDK コマンド、XSDB コマンド、および HSI コマンドを実行するためのスクリプト操作可能なコマンド ライン インターフェイスです。XSCT は、[スタート] メニューから起動できますが、<SDK installation directory>/bin フォルダの xsct.bat ファイルを実行しても起動できます。XSCT でサポートされているすべてのコマンドは、カテゴリ別にグループ化されています。

ザイリックス SDK では、ソフトウェア開発プロセスを容易にするために、各タスクに対応する個別のパーспекティブを提供します。C/C++ 開発で利用可能なパーспекティブには次のものがあります。

- 。 **C/C++ パーспекティブ ビュー:** ソフトウェア C/C++ プロジェクトの表示、作成、ビルドに利用します。デフォルトでは、エディター領域のほか、ザイリックス SDK プロジェクト、C/C++ プロジェクト (ワークスペース内のソフトウェア プロジェクトを表示)、ナビゲーション コンソール、プロパティ、タスク、make ターゲット、アウトライン、検索などの各種ビューで構成されます。
- 。 **システム デバッガー:** ソフトウェア アプリケーションのデバッグに利用します。オープン ソースのシステム デバッガーをカスタマイズしたものを SDK に統合しています。
- 。 **システム パフォーマンス モニター:** MicroBlaze デバイス、および Zynq UltraScale+ MPSoC デバイスの PL と PS のパフォーマンス サマリ ビューを利用して、ハードウェアおよびソフトウェア システムのパフォーマンス特性評価が可能です。
- 。 **リモート システム エクスプローラー:** 各種リモート システムに接続して開発が可能です。

ザイリックス SDK は Linux アプリケーションの開発をサポートしていますが、Linux カーネルの開発とデバッグを明示的にターゲットとしていません。これらのツールと機能は、ザイリックス PetaLinux ツールおよびサードパーティのパートナーによって提供されます。

ザイリックス SDK の機能および SDK デザイン フローの詳細は、ザイリックス ソフトウェア開発キットのヘルプ [参照 24] を参照してください。

SDK ツールは、「エンベデッド開発」ページ [参照 26] からダウンロード可能です。

Arm GNU ツール

ザイリンクス ソフトウェア開発プラットフォームには、Arm GNU オープン ソース ツール チェーンが採用されています。ザイリンクス ソフトウェア開発キット (SDK) には Linux ホスト用 GNU ツールが含まれます。このセクションでは、Zynq UltraScale+ MPSoC デバイスのプロセッシング クラスターに利用可能なオープン ソース GNU ツールおよび Linux ツールについて説明します。

次の表に、APU、RPU、およびエンベデッド MicroBlaze プロセッサのプログラミングに利用可能なザイリンクス Arm GNU ツールの一部を示します。

表 3-1: ザイリンクス Arm GNU ツール

ツール	説明
aarch64-linux-gnu-gcc aarch64-linux-gnu-g++	GNU C/C++ コンパイラ。
aarch64-linux-gnu-as	GNU アセンブラー。
aarch64-linux-gnu-ld	GNU リンカー。
aarch64-linux-gnu-ar	アーカイブの作成、変更、およびアーカイブからのファイル抽出用ユーティリティ。
aarch64-linux-gnu-objcopy	オブジェクト ファイルのコピーと変換を実行。
aarch64-linux-gnu-objdump	オブジェクト ファイルからの情報を表示。
aarch64-linux-gnu-size	オブジェクトまたはアーカイブ ファイルのセクション サイズをリスト表示。
aarch64-linux-gnu-gprof	プロファイリング情報を表示。
aarch64-linux-gnu-gdb	GNU デバッガー。

デバイス ツリー ジェネレーター

デバイス ツリー (DT) データ構造体には、ハードウェアを記述するプロパティを持つノードが格納されます。Linux カーネルは、デバイス ツリーを使用して幅広いハードウェア構成をサポートします。

FPGA では、ペリフェラル ロジックの組み合わせ、および各ペリフェラル ロジックが使用する構成が変化する可能性があります。こうしたすべての組み合わせに対して、デバイス ツリー ジェネレーター (DTG) は .dts/.dtsi デバイス ツリー ファイルを生成します。

デバイス ツリー ジェネレーターによって生成される dts/dtsi ファイルには、次のものがあります。

- `p1.dtsi`: すべてのメモリ マップド ペリフェラル ロジック (PL) IP が含まれます。
- `pcw.dtsi`: PS IP の動的プロパティが含まれます。
- `system-top.dts`: メモリ、ブート引数、およびコマンド ライン パラメーターが含まれます。
- `zynqmp.dtsi`: PS 固有の情報および CPU に関するすべての情報が含まれます。
- `zynqmp-clk-ccf.dtsi`: PS ペリフェラル IP のすべてのクロック情報が含まれます。

詳細は、ザイリンクス Wiki [\[参照 38\]](#) の「[Build Device Tree Blob](#)」のページ (英語) を参照してください。

PetaLinux ツール

PetaLinux ツールには、オープン ソース Linux ソフトウェアをカスタマイズおよびビルドしてデバイスに展開するために必要なものがすべて揃っています。

PetaLinux には次が含まれます。

- GNU、petalinux-build、make などカーネル イメージおよびアプリケーション ソフトウェアをビルドするためのビルド ツール。
- GDB、petalinux-boot、およびプロファイリング用の oprofile を含むデバッグ ツール。

次の表に、サポートされる PetaLinux ツールを示します。

表 3-2: サポートされる PetaLinux ツール

ツール	説明
GNU	Arm GNU ツール。
petalinux-build	ソフトウェア イメージ ファイルの構築に使用。
Make	アプリケーションをコンパイルするための make ビルド。
GDB	デバッグ用の GDB ツール。
petalinux-boot	Linux のブートに使用。
QEMU	Zynq UltraScale+ MPSoC デバイス用のエミュレーター プラットフォーム。
OProfile	プロファイリングに使用。

詳細は、次の資料を参照してください。

- PetaLinux ツール資料 [\[参照 2\]](#)
- 『Zynq UltraScale+ MPSoC: エンベデッド デザイン チュートリアル』(UG1209) [\[参照 13\]](#)
- 『Zynq デバイス用 Libmetal および OpenAMP ユーザー ガイド』(UG1186) [\[参照 16\]](#)

Yocto を使用した Linux ソフトウェア開発

ザイリンクスは、自社の Yocto ビルド システムを使用するカスタマーが Zynq UltraScale+ MPSoC デバイス用に Linux を設定、ビルド、およびデプロイできるように、meta-xilinx Yocto/OpenEmbedded レシピを提供しています。

meta-xilinx レイヤーには、ザイリンクス デバイスを使用した一般的なボードに対する BSP も多数含まれます。

meta-xilinx レイヤーは各種コンポーネントのレシピを追加しており、Yocto/OE をより強力にサポートします。詳細は、meta-xilinx リンク [\[参照 33\]](#) を参照してください。

Cortex-A53 上で動作する Linux ソフトウェアは、オープン ソースの Linux ツールを使用して開発できます。このセクションでは、Linux Yocto ツールと Yocto Project 開発環境について説明します。

次の表に、Yocto ツールを示します。

表 3-3: Yocto ツール

ツール タイプ	名前	説明
Yocto ビルド ツール	Bitbake	タスク間の複雑な依存制約の中でシェルと Python タスクを効率的に並列実行できる汎用タスク実行エンジン。
Yocto プロファイル およびトレース ツール	Perf	Linux カーネルに付属するプロファイリングおよびトレーシングツール。
	Ftrace	ftrace 関数トレーサーを参照します。また、それ以外の関連する多数のトレーサー、およびこれらトレーサーが使用するインフラストラクチャも含まれます。
	Oprofile	コマンドラインアプリケーションとしてターゲットシステム上で動作するシステム全体のプロファイラー。
	Sysprof	システム全体のプロファイラーで、1つのウィンドウに3つのペインと複数のボタンが配置され、ワンストップでプロファイルの開始、停止、および表示が可能。
	Blktrace	低レベル ディスク I/O のトレースおよびレポート用ツール。

Yocto Project 開発環境

Yocto Project 開発環境を設定することにより、Xilinx GIT サーバーで提供される Yocto レシピを使用して Zynq UltraScale+ MPSoC デバイス用 Linux ソフトウェアを開発できるようになります。Yocto Project のコンポーネントを利用すると、Linux を使用したソフトウェアスタックの設計、開発、ビルドが可能です。

図 3-1 に、完全な Yocto Project 開発環境を示します。Yocto Project は幅広い種類のツールを統合しており、最新のザイリンクス カーネルをダウンロードして、ローカルプロジェクト形式で部分的に改良を加えてビルドできます。

また、BSP を利用してビルドおよびハードウェア構成を変更することもできます。

Yocto はコンパイラとその他のツールを組み合わせることでイメージのビルドとテストを実行します。イメージが品質テストに合格し、SDK 生成に必要なパッケージフィードを受け取ると、Yocto ツールはアプリケーション開発用に SDK を起動します。

Yocto Project には次の重要な特長があります。

- 最新の Linux カーネルおよびエンベデッド環境に適したシステム コマンドのセットとライブラリを提供します。
- X11、GTK+、Qt、Clutter、SDL をはじめとする各種システム コンポーネントを利用できるため、ディスプレイ ハードウェアを備えたデバイスで豊富なユーザー体験を実現できます。デバイスにディスプレイがない場合や別の UI フレームワークを使用する場合、これらのコンポーネントはインストール不要です。
- OpenEmbedded プロジェクトと互換性のある安定したコアを、目的に応じて作成します。このコアを使用して、Linux ソフトウェアを簡単かつ確実にビルドして開発できます。
- Quick Emulator (QEMU) により、幅広い種類のハードウェアおよびデバイスのエミュレーションをサポートします。詳細は、『Zynq UltraScale+ MPSoC QEMU ユーザー ガイド』(UG1169) [参照 8] を参照してください。



重要: Yocto の一部の機能は、ザイリンクス QEMU では利用できません。

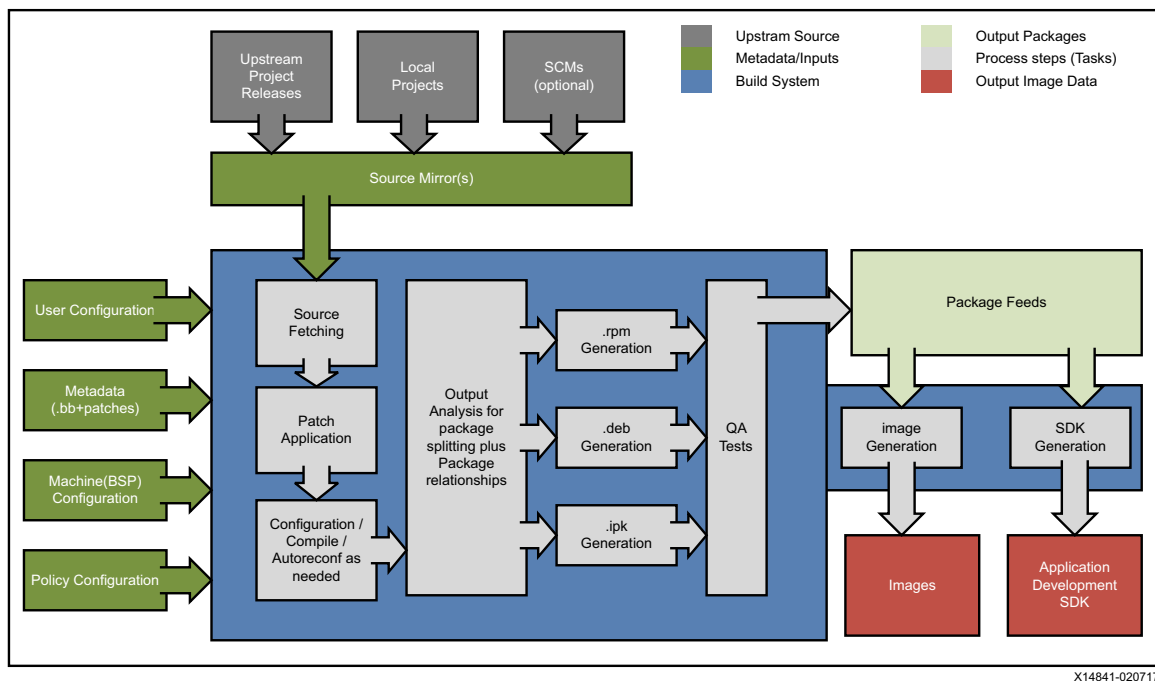


図 3-1: Yocto Project 開発環境

Yocto ツールおよび Yocto Project 開発環境は、Yocto Project のウェブサイト [参照 44] からダウンロードできます。

ザイリンクスが提供する Yocto 機能の詳細は、『PetaLinux ツール資料: リファレンス ガイド』(UG1144) [参照 27] の「Yocto の機能」を参照してください。

ソフトウェア スタック

はじめに

この章では、Zynq® UltraScale+™ MPSoC デバイスで利用可能な各種ソフトウェア スタックの概要を紹介します。

このデバイスで使用する各種ソフトウェア開発ツールの詳細は、[第 3 章「開発ツール」](#)を参照してください。ベアメタルおよび Linux ソフトウェア アプリケーション開発の詳細は、[第 5 章「ソフトウェア開発フロー」](#)を参照してください。

ベアメタル ソフトウェア スタック

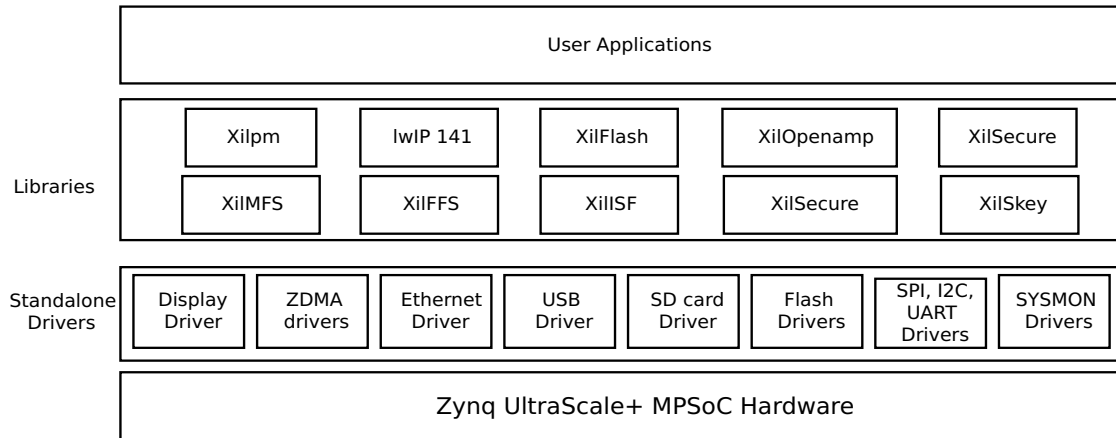
ザイリンクスは、ザイリンクス SDK ツールの一部としてスタンドアロン BSP (ボード サポート パッケージ) と呼ばれるベアメタル ソフトウェア スタックを提供しています。スタンドアロン BSP は、標準入出力やプロセッサ ハードウェア機能へのアクセスなどの基本機能を備えたシンプルなシングル スレッド環境を提供します。この BSP および付属のライブラリは非常に柔軟な設定が可能で、必要な機能を最小限のオーバーヘッドで提供します。詳細は、[第 3 章「開発ツール」](#)の「[ザイリンクス ソフトウェア開発キット \(SDK\)](#)」で説明しています。スタンドアロン ドライバーは、次のパスにあります。

```
<Xilinx Installation Directory>\SDK\<version>\data\embeddedsdsw\XilinxProcessorIPLib\drivers
```

ライブラリは、次のパスにあります。

```
<Xilinx Installation Directory>\SDK\<version>\data\embeddedsdsw\lib\sw_services
```

次の図に、APU のベアメタルソフトウェア スタックを示します。



X17169-11261

図 4-1: ベアメタルソフトウェア開発スタック

注記: RPU のベアメタル用ライブラリおよびドライバーのソフトウェア スタックは、APU と同じです。

ベアメタル スタックは、主に次のコンポーネントで構成されます。

- 。 ペリフェラル用のソフトウェア ドライバー : PS の Arm® Cortex™-A53、Arm Cortex-R5F プロセッサ、および PL のザイリンクス MicroBlaze™ プロセッサを使用する上で必要なコア ルーチンなど。
- 。 PS のペリフェラルおよびオプションの PL ペリフェラル用ベアメタル ドライバー。
- 。 標準 C ライブラリ: オープン ソースの Newlib ライブラリを Arm Cortex-A53、Arm Cortex-R5F、および MicroBlaze プロセッサにポーティングした libc および libm。
- 。 その他のミドルウェア ライブラリ: ネットワーキング、ファイル システム、暗号化をサポート。
- 。 サンプルアプリケーション: FSBL (第 1 段階ブートローダー)、テスト アプリケーションなど。

標準 C ライブラリ (libc)

libc ライブラリにはすべての C プログラムで使用できる標準関数が含まれます。表 4-1 に、libc のモジュールを示します。

表 4-1: libc.a の関数と説明

ヘッダー ファイル	説明
alloca.h	スタック内の空間確保
assert.h	診断コード
ctype.h	文字に関する操作
errno.h	システム エラー
inttypes.h	整数型の変換
math.h	数学関数
setjmp.h	ローカル外の goto コード
stdint.h	標準整数型

表 4-1: libc.a の関数と説明 (続き)

ヘッダー ファイル	説明
stdio.h	標準 I/O 機能
stdlib.h	一般的なユーティリティ関数
time.h	タイム関数

標準 C ライブラリ 数学関数 (libm)

表 4-2 に、libm 数学 C モジュールを示します。

表 4-2: libm.a の関数タイプと関数一覧

関数タイプ	サポートされる関数
代数関数	cbrt、hypot、sqrt
初等超越関数	asin、acos、atan、atan2、asinh、acosh、atanh、exp、expm1、pow、log、log1p、log10、sin、cos、tan、sinh、cosh、tanh
高等超越関数	j0、j1、jn、y0、y1、yn、erf、erfc、gamma、lgamma、gamma_rgamma_r
整数丸め関数	ceil、floor、rint
IEEE 標準推奨関数	copysign、fmod、ilogb、nextafter、remainder、scalbn、fabs
IEEE 分類関数	isnan
浮動小数点	logb、scalb、significand
ユーザー定義のエラー処理ルーチン	matherr

スタンドアロン BSP

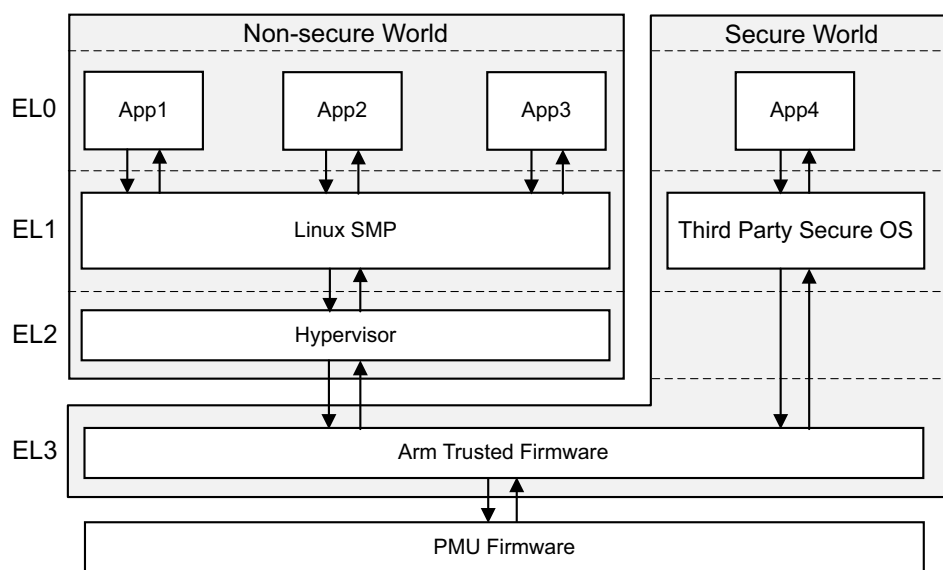
スタンドアロン BSP には、次のライブラリが用意されています。

- 。 XilFatFS: ザイリンクス システム ACE コンパクト フラッシュに格納されたファイルへの読み出し/書き込みアクセスを提供する LibXil FATFile システム。
- 。 XilFFS: 汎用 FAT ファイル システム ライブラリ。
- 。 XilFlash: Intel/AMD CFI 準拠パラレル フラッシュ用ザイリンクス フラッシュ ライブラリ。
- 。 XilHSF: ザイリンクス インシステム フラッシュ ハードウェアをサポートしたインシステム フラッシュ ライブラリ。
- 。 XilMFS: メモリ ファイル システム。
- 。 XilSecure: セキュア ハードウェア (AES、RSA および SHA) エンジンへのアクセスをサポートするザイリンクス セキュア ライブラリ。
- 。 XilKey: ザイリンクス セキュア キー ライブラリ。
- 。 lwIP ライブラリ: コア lwIP スタック、およびこのスタックへの BSD (Berkeley Software Distribution) ソケット スタイル インターフェイスへのアクセスを提供するオープン ソース TCP/IP プロトコルスイート。

付録 B 「Xilinx Standard C Libraries」にこれらのライブラリを示します。

Linux ソフトウェア スタック

Linux OS は Zynq UltraScale+ MPSoC デバイスをサポートしています。唯一の例外である Arm GPU を除いて、ザイリックスは PS のすべてのペリフェラル、および PL の主要ペリフェラルのオープン ソース ドライバーを開発して提供しています。次の図に、Linux およびオプションのハイパーバイザーを含む APU の完全なソフトウェア スタックを示します。



X18968-07121

図 4-2: Linux ソフトウェア開発スタック

Armv8 例外モデルは、次に示す EL0 ~ EL3 の例外レベルを定義しています。

- EL0 は最も低いソフトウェア実行特権レベルです。EL0 での実行は特権なし実行と呼ばれます。
- 例外レベルが 1 から 3 へと大きくなるにつれ、ソフトウェア実行特権レベルが上がります。
- EL2 はプロセッサ仮想化をサポートします。オープン ソースまたは商用のハイパーバイザーをオプションでソフトウェア スタックに含めることができます。
- EL3 はセキュア ステートをサポートします。Cortex-A53 MPCore プロセッサは EL0 ~ EL3 のすべての例外レベルを実装し、各例外レベルで AArch64 と AArch32 の両方の実行ステートをサポートしています。

Zynq UltraScale+ MPSoC デバイスの Linux ソフトウェア スタックはいくつかの方法で利用できます。オプションは次のとおりです。

- 。 **PetaLinux ツール:** PetaLinux ツールには Linux ソース ツリーのブランチ、U-Boot、および Yocto ベース ツールが含まれ、カーネル、root ファイル システム、デバイス ツリー、およびアプリケーションを含む完全なザイリンクス デバイス用 Linux イメージを簡単にビルドできます。詳細は、PetaLinux 製品ページ [\[参照 2\]](#) を参照してください。PetaLinux ツールは、次に示すオープン ソース Linux コンポーネントと組み合わせて使用できます。
- 。 **オープン ソース Linux および U-Boot:** ドライバー、ボード コンフィギュレーション、U-Boot アップデートを含む Zynq UltraScale+ MPSoC デバイス用の Linux カーネル ソースは、ザイリンクスの Github リンク [\[参照 31\]](#) から入手できます。また、メイン Linux カーネルおよび U-Boot ツリーからも継続的に提供されます。Yocto ボード サポート パッケージもメイン Yocto ツリーにあります。
- 。 **商用 Linux ディストリビューション:** 一部の商用ディストリビューションもザイリンクスの UltraScale+ MPSoC デバイスをサポートしており、Linux の設定、最適化、およびデバッグのための高度なツールを備えています。詳細はザイリンクスのエンベデッド コンピューティングのページ [\[参照 32\]](#) を参照してください。

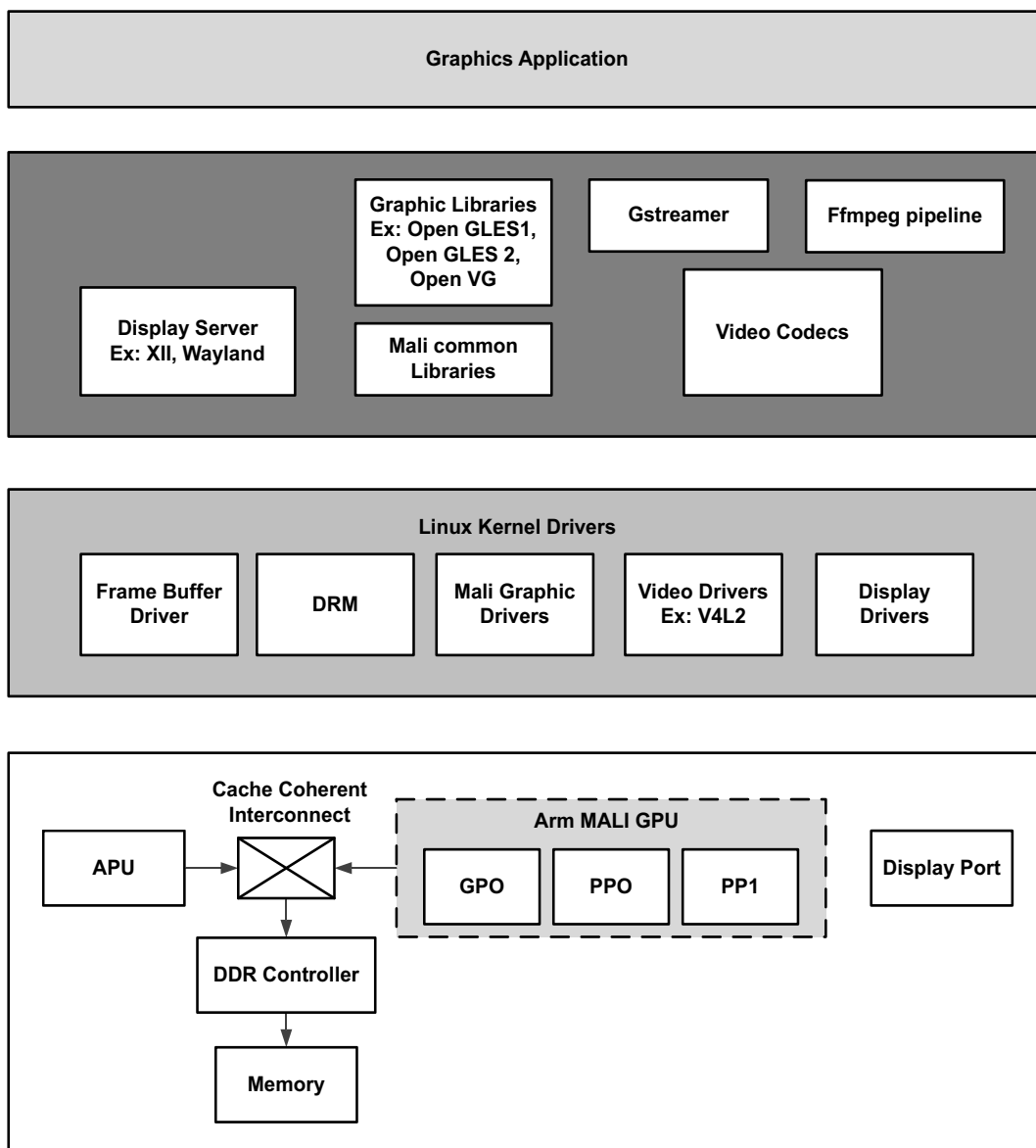
マルチメディア スタックの概要

このセクションでは、Zynq UltraScale+ MPSoC デバイスのマルチメディア ソフトウェア スタックについて説明します。

グラフィックス アプリケーションは、GPU および高性能 DisplayPort によって高速に処理されます。GPU は 2D/3D グラフィックスに対するハードウェア アクセラレーションを提供します。GPU には 1 つのジオメトリ プロセッサ (GP) と 2 つのピクセルプロセッサ (PP0、PP1) があり、それぞれが専用のメモリ管理ユニット (MMU) を備えています。APU と GPU のキャッシュ コヒーレンシは、キャッシュ コヒーレント インターコネクト (CCI) によって確保されます。CCI は ACE (AXI Coherency Extension) のみをサポートします。

APU と GPU は CCI を経由して DDR コントローラーに接続され、ここで DDR アクセスのアービトレーションが実行されます。

次の図に、マルチメディア スタックを示します。



X14795-071317

図 4-3: マルチメディア スタック

プロセッサ上で動作するアプリケーションは、マルチメディア用 Linux カーネルドライバーを利用してハードウェアにアクセスします。

アプリケーションは、表 4-3 に示すライブラリおよびフレームワーク コンポーネントで構成されるミドルウェア スタックを介してマルチメディア ドライバーを利用します。

表 4-3: ライブラリおよびフレームワーク コンポーネント

コンポーネント	説明
ディスプレイ サーバー	アプリケーションからオペレーティング システムに対する入出力を調整します。
グラフィックス ライブラリ	Zynq UltraScale+ MPSoC デバイス アーキテクチャは OpenGL ES 1.1/2.2 および OpenVG 1.1 をサポートしています。
Mali-400 MP2 共通ライブラリ	Mali-400 MP2 グラフィックス ライブラリです。異なる EGL バックエンドの切り替え方法の詳細は、「 Xilinx MALI Driver 」の Wiki ページ (英語) を参照してください。
Gstreamer	プログラマが各種メディア処理コンポーネントを作成するためのフリーウェアのマルチメディア フレームワークです。
ビデオ コーデック	ビデオ エンコーダーおよびデコーダー。

表 4-4 に、Linux カーネル グラフィックス ドライバーを示します。

表 4-4: Linux カーネル ドライバー

ドライバ	説明
フレーム バッファードライバー	/dev/fb* 経由のインターフェイスで外部からアクセス可能なカーネル グラフィックス ドライバーです。このインターフェイスはビデオ モードの設定やリニア フレーム バッファへの描画など、一部の機能のみを実装しています。
ダイレクト レンダリング マネージャ (DRM)	複数のユーザー空間コンポーネント間でハードウェアのレンダリングを実行します。
MALI-400 MP2 グラフィックス ドライバー	GPU ハードウェアへのアクセスを提供します。
ビデオ ドライバー	V4L2 フレームワークに基づくビデオ キャプチャおよび出力デバイス パイプライン ドライバーです。ザイリンクス Linux V4L2 パイプライン ドライバーは、複数のサブデバイスを持つパイプライン全体を指します。パイプラインはメディア ノード経由で設定でき、ストリーム オン/オフなどの制御はビデオ ノード経由で実行できます。 デバイス ノードはパイプライン ドライバーによって作成されます。パイプライン ドライバーには DMA エンジン API のラッパー レイヤーも含まれ、RAM からのフレーム読み出し/書き込みが可能です。
DisplayPort ドライバー	DRM フレームワークに基づいて DisplayPort へのハードウェア アクセスを許可します。

FreeRTOS ソフトウェア スタック

ザイリンクスは、ザイリンクス SDK ツールの一部として FreeRTOS BSP (ボード サポート パッケージ) を提供しています。FreeRTOS BSP は、標準入出力やプロセッサ ハードウェア機能へのアクセスなどの基本機能を備えたシンプルなマルチスレッド環境を提供します。この BSP および付属のライブラリは非常に柔軟な設定が可能で、必要な機能を最小限のオーバーヘッドで提供します。FreeRTOS ソフトウェア スタックは、FreeRTOS ライブラリを含む点を除き、ベアメタル ソフトウェア スタックと同じです。通常、スタンドアロン ライブラリに含まれるザイリンクス デバイス ドライバーは、デバイスへのアクセスを要求するのがシングル スレッドであれば、FreeRTOS 内で使用できます。ザイリンクス ベアメタル ドライバーは、オペレーティング システムを認識しません。クリティカル セクションを保護するためのミューテックスはサポートされず、セマフォを使用した同期メカニズム也没有ありません。FreeRTOS カーネルでドライバー API を使用する際は、これらの点に注意が必要です。

次の図に、RPU の FreeRTOS ソフトウェア スタックを示します。

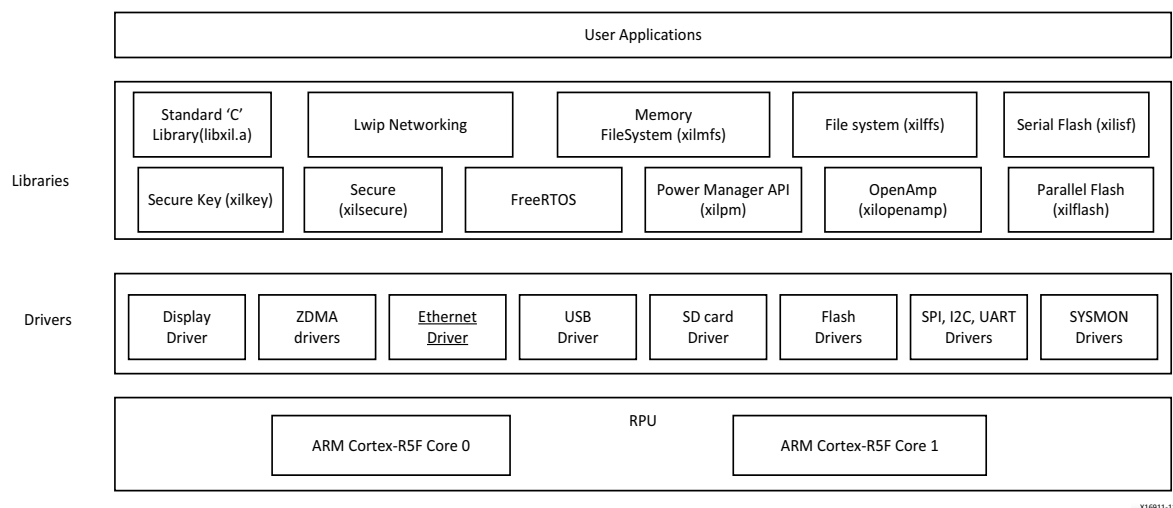


図 4-4: FreeRTOS ソフトウェア スタック

注記: APU の FreeRTOS ソフトウェア スタックは、ライブラリが 32 ビットと 64 ビットの両方をサポートすること以外は RPU のソフトウェア スタックと同じです。

サードパーティ ソフトウェア スタック

既述以外に多くのエンベデッド ソフトウェア ソリューションがザイリンクス パートナー エコシステムから提供されています。詳細は、ザイリンクスのウェブサイト「エンベデッド開発」[参照 32] および「サードパーティ ツール」[参照 4] を参照してください。

ソフトウェア開発フロー

ソフトウェア開発フローの概要

この章では、ザイリンクス ソフトウェア開発キット (SDK) を使用した RPU および APU 用のベアメタル ソフトウェア開発、ならびに PetaLinux ツールおよび SDK ツールを使用した APU 用の Linux ソフトウェア開発について説明します。

次の図に、Zynq® UltraScale+™ MPSoC デバイスの最上位ソフトウェア アーキテクチャを示します。

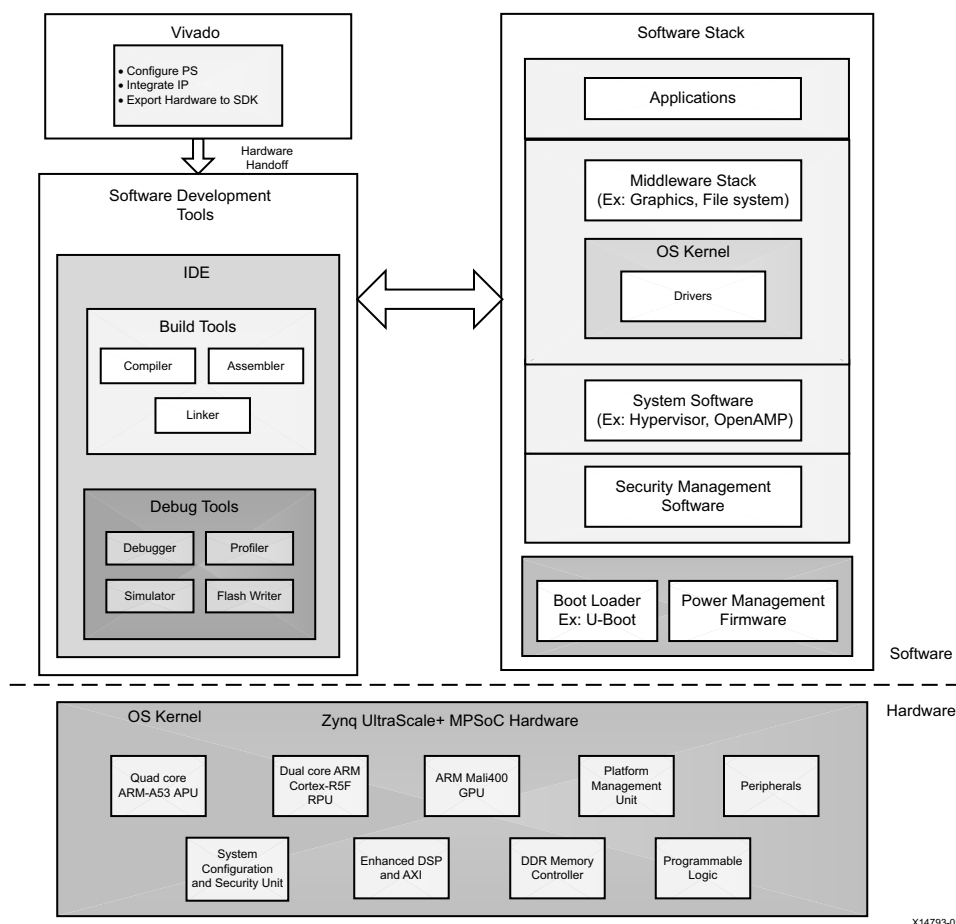
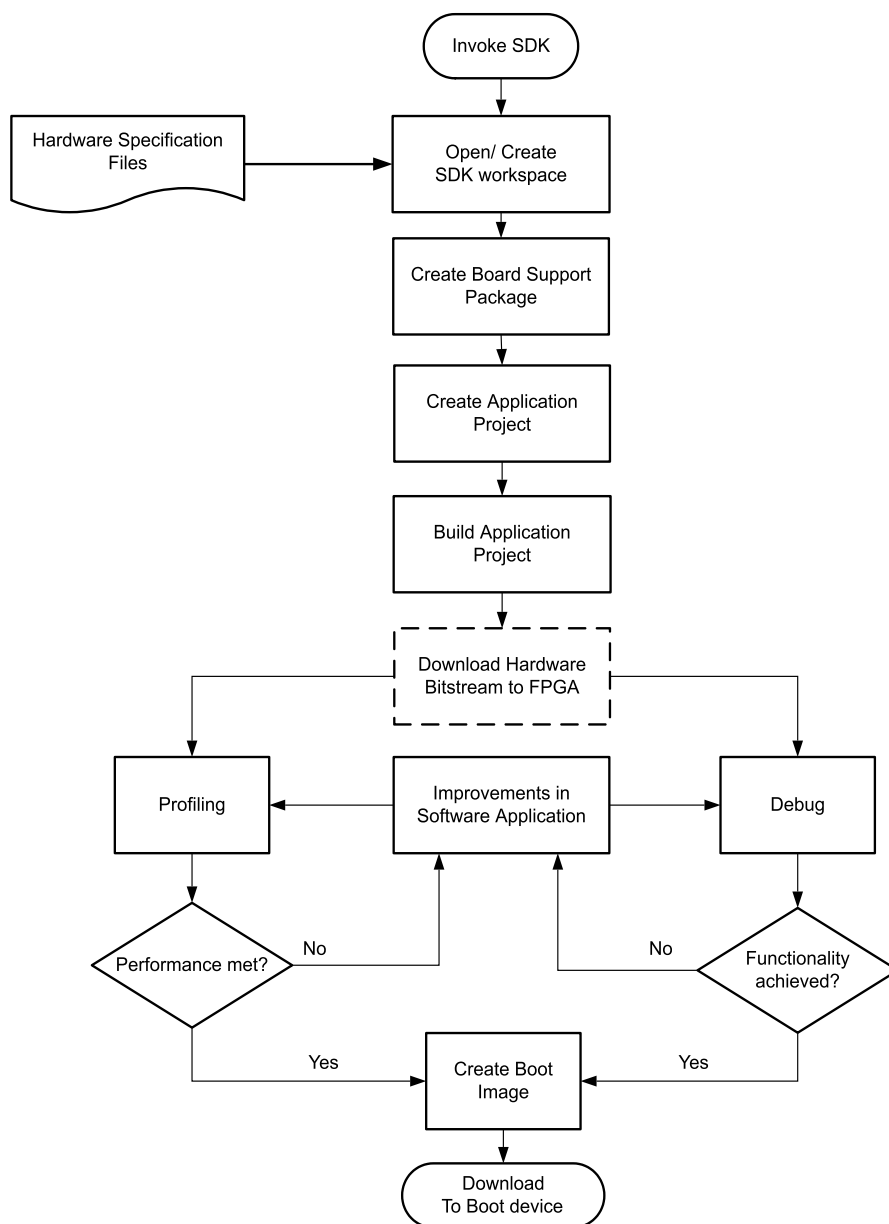


図 5-1: ソフトウェア開発アーキテクチャ

ベアメタル アプリケーションの開発

このセクションでは、APU および RPU 用のベアメタルアプリケーションをザイリンクス SDK を使用して開発する場合のデザイン フローについて説明します。

次の図に、SDK の最上位のデザイン フローを示します。



X14817-071217

図 5-2: ベアメタル アプリケーションの開発フロー

ベアメタル アプリケーションの開発手順は、次のとおりです。

1. ベアメタル アプリケーション用の SDK ワークスペースを開く/作成する。[「Creating a Standalone Application Project」](#) 参照。
2. ハードウェア プラットフォーム情報をインポートする。[「Importing a Hardware Platform Specification File」](#) 参照。
3. ターゲット プロセッサ (Cortex-A53、Cortex-R5F、または PMU MicroBlaze™) を選択する。
4. ボード サポート パッケージ (BSP) を作成する。[「Creating a Board Support Package \(SDK\)」](#) 参照。
5. BSP のコンフィギュレーション設定を変更する (オプション)。[「Changing Build Configuration」](#) 参照。
6. カスタム IP ドライバーのサポートを追加する (オプション)。[「Using the Board Support Package Drivers Page」](#) 参照。
7. アプリケーション プロジェクトを作成する。[「Creating Application Projects」](#) 参照。
8. アプリケーション プロジェクトをビルドする。[「Building Projects」](#) 参照。
9. ユーザー アプリケーションをデバッグする。[「Debugging Projects」](#) 参照。
10. ユーザー アプリケーションを実行する。[「Running Projects」](#) 参照。
11. ユーザー アプリケーションをプロファイリングする。[「Software Profiling」](#) 参照。
12. システム パフォーマンスをモデリングする。[「System Performance Modeling」](#) 参照。
13. ブート イメージを作成する。[「Creating a Boot Image」](#) 参照。

これらの手順の詳細は、「MPSoC PetaLinux Software Development」[\[参照 34\]](#) を参照してください。

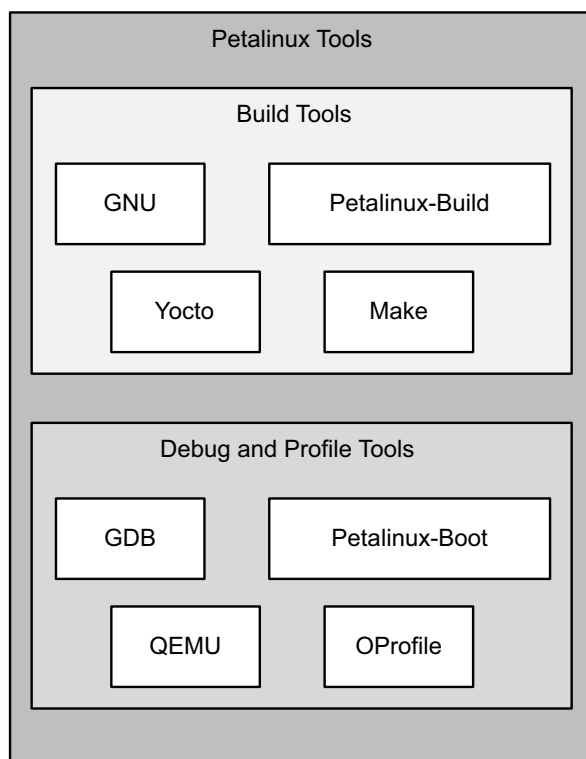
QEMU の詳細は、『Zynq UltraScale+ MPSoC QEMU ユーザー ガイド』(UG1169) [\[参照 8\]](#) を参照してください。

注記: Cortex-R5F および Cortex-A53 の 32 ビット ベアメタル ソフトウェアは、デバイス DMA を使用した 64 ビット アドレスのデータ転送をサポートしていません。

注記: デフォルトでは、スタンドアロン アプリケーションはすべて APU0 上でのみ動作します。その他の APU コアはオフになります。

PetaLinux ツールを使用したアプリケーション開発

PetaLinux ツール環境でのソフトウェア開発フローには多くの工程があります。デザインフローをわかりやすく整理するため、次の図に PetaLinux 環境でのアプリケーション開発のすべての工程を示します。



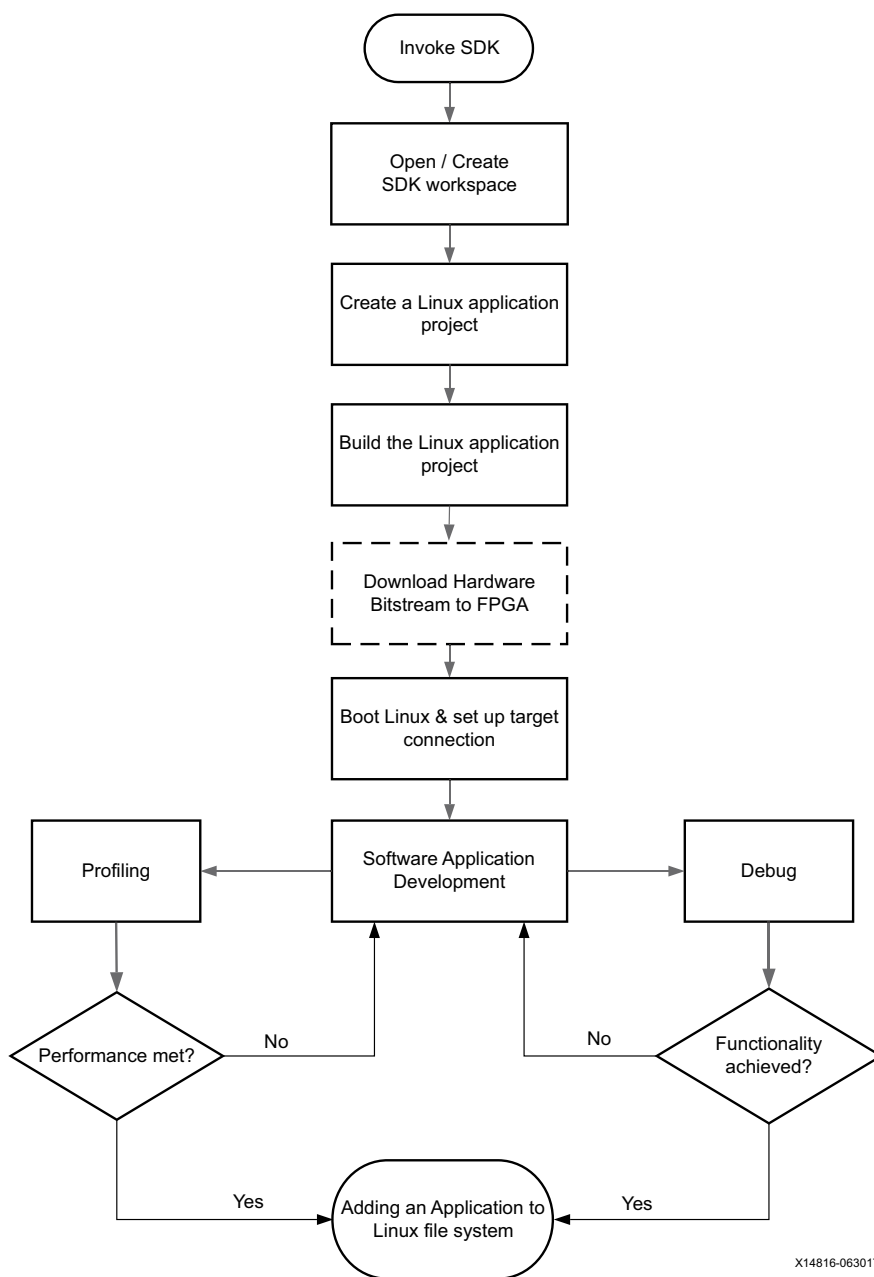
X14815-063017

図 5-3: PetaLinux を使用したソフトウェア開発フロー

SDK を使用した 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、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。

次の図に、SDK を使用した Linux ユーザー アプリケーションの一般的な開発手順を示します。



X14816-063017

図 5-4: Linux アプリケーションの開発フロー

次に、PetaLinux アプリケーションの開発および実行の手順を示します。

1. PetaLinux ツールの作業環境をセットアップします。「[PetaLinux 作業環境のセットアップ](#)」を参照してください。
2. PetaLinux プロジェクトまたはユーザー アプリケーションを作成します。「[新規 PetaLinux プロジェクトの作成](#)」を参照してください。
3. ユーザー プロジェクトに応じて、PetaLinux ツールを設定およびカスタマイズします。
4. SDK Linux C/C++ アプリケーション プロジェクトを PetaLinux ワークスペースにインポートします。「[インストール手順](#)」を参照してください。
5. PetaLinux イメージを構築します。「[システム イメージのビルド](#)」を参照してください。
6. プラットフォーム (QEMU またはボード) で PetaLinux イメージを実行します。「[ユーザー アプリケーションのビルド](#)」を参照してください。
7. PetaLinux イメージをデバッグします。デバッグにはいくつかのオプションがあります。『PetaLinux ツール資料: リファレンス ガイド』(UG1144) [\[参照 27\]](#) を参照してください。

上記手順の詳細は、「MPSoC PetaLinux Software Development」[\[参照 34\]](#) を参照してください。

QEMU の詳細は、『Zynq UltraScale+ MPSoC QEMU ユーザー ガイド』(UG1169) [\[参照 8\]](#) を参照してください。

SDK 機能の詳細および「Hello World」サンプル デザインを使用した SDK デザイン フローは、SDK のヘルプ [\[参照 24\]](#) を参照してください。その他、『ザイリンクス ソフトウェア開発キット (SDK) ユーザー ガイド: システム パフォーマンス解析』(UG1145) [\[参照 28\]](#) も参照してください。

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」や空のアプリケーション、FSBL アプリケーションなど、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 ウィザードで起動します。

空のアプリケーションを作成するか、また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。

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できます。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 makefile 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。

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 UltraScale+ MPSoC デバイス プラットフォームで 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。必要に応じて次のコマンドを使用します。

- 。 `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

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 カーネルはこれらのドライバーをサポートします。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 3 章「開発ツール」](#)を参照してください。

Linux ファイル システムにアプリケーションを追加する

コンパイル済みのユーザー アプリケーション、および必要な共有ライブラリを Linux ファイル システムに追加するには、次の手順を実行します。

- 。 Linux 環境に SSH が含まれている場合は、Zynq UltraScale+ MPSoC デバイス プラットフォーム上で 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。

Linux アプリケーション開発の詳細は、SDK ヘルプの「Linux アプリケーション プロジェクトを作成する」[\[参照 24\]](#)を参照してください。

ソフトウェア デザインのパラダイム

はじめに

ザイリンクス Zynq® UltraScale+™ MPSoC デバイス アーキテクチャは、タスクごとに最適なエンジンを使用するヘテロジニアス マルチプロセッサ エンジンをサポートしています。これらプロセッサをターゲットとしたソフトウェアの主な開発アプローチには、次のものがあります。

- **マルチプロセッサ開発用フレームワーク**: Zynq UltraScale+ MPSoC デバイスでの開発に利用できるフレームワークについて説明します。
- **対称型マルチプロセッシング (SMP)**: PetaLinux で SMP を使用すると、Zynq UltraScale+ MPSoC デバイス向け Linux プラットフォームで SMP を最もシンプルに開発できます。
- **非対称型マルチプロセッシング (AMP)**: AMP モードでは、どのプロセッサで何を実行するかを厳密に制御できるため、複数のプロセッサ エンジンをより効果的に活用できます。SMP とは異なり、AMP にはさまざまな利用方法があります。このセクションでは、複雑度の異なる AMP の 2 つの利用法について説明します。

以降のセクションでは、これらの各開発手法について詳しく説明します。

マルチプロセッサ開発用フレームワーク

ザイリンクスは、ヘテロジニアス プロセッサとザイリンクス 7 シリーズ FPGA を利用したアプリケーション開発を容易にするため、Zynq UltraScale+ MPSoC デバイス向けのフレームワークを複数提供しています。これらのフレームワークの説明は次のとおりです。

- **ハイパーバイザー フレームワーク**: ザイリンクスが提供する Xen ハイパーバイザーは、Zynq UltraScale+ MPSoC デバイスの APU における仮想化をサポートするために欠かせないアイテムです。詳細は、「[ハイパーバイザーの使用](#)」を参照してください。
- **認証フレームワーク**: Zynq UltraScale+ MPSoC デバイスでは、認証フレームワークの一部として、認証機能と暗号化機能をサポートしています。認証フレームワークの詳細は、[第 8 章の「ブート時のセキュリティ」](#)を参照してください。
- **TrustZone フレームワーク**: TrustZone テクノロジは 1 つのシステム内でセキュアなプロセスと非セキュアなプロセスを分離し、維持します。

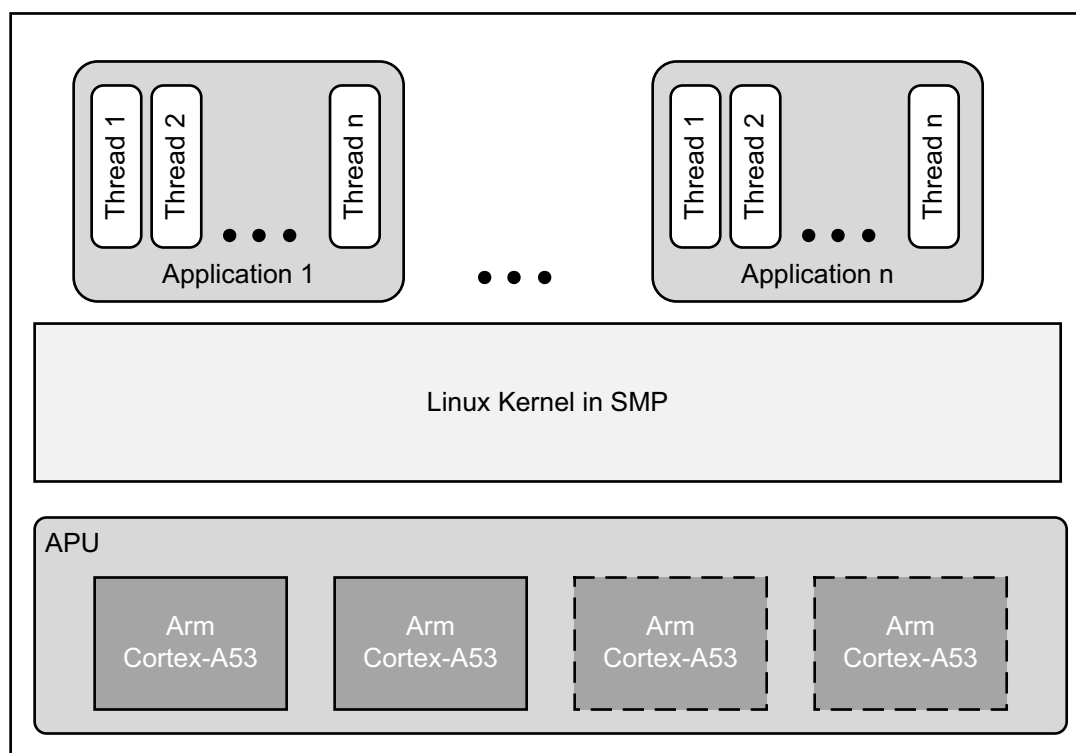
ザイリンクスは、Arm トラステッド ファームウェア (ATF) を使用して TrustZone に対応し、セキュア プロセスと非セキュア プロセスを分離します。ATF の詳細は、[第 8 章の「Arm トラステッド ファームウェア」](#)を参照してください。

- **マルチプロセッサ通信フレームワーク:** ザイリンクスは、異なるプロセッシング ユニット間で互いに通信できるように、Zynq UltraScale+ MPSoC デバイス用の OpenAMP フレームワークを提供しています。詳細は、『ザイリンクス Quick Emulator ユーザー ガイド』(UG1169) [参照 16] を参照してください。
- **電力管理フレームワーク:** 電力管理フレームワークにより、異なるプロセッシング ユニットで駆動されるソフトウェア コンポーネントが電力管理ユニットと通信できるようになります。

対称型マルチプロセッシング (SMP)

SMP は、1 つのオペレーティング システム インスタンスで複数のプロセッサを使用できるようにします。複数のプロセッサ、キャッシュ、ペリフェラル割り込み、および負荷分散などの複雑な管理のほとんどは、オペレーティング システムによって実行されます。

Zynq UltraScale+ MPSoC デバイスの APU にはキャッシュ コヒーレントな 4 つのヘテロジニアス Arm Cortex™-A53 プロセッサがあり、これらは OS (Linux または VxWorks) を使用した SMP モードをサポートしています。SMP モードの APU を簡単に利用できるように、ザイリンクスおよびパートナーはオペレーティング システムを提供しています。次の図は、1 つの OS 上で複数のアプリケーションを実行した Linux SMP の例を示しています。



X14837-063017

図 6-1: Linux を使用した SMP モードの例

この動作モードは、既存のザイリンクス ソフトウェアを使用する開発者が利用可能な Linux アプリケーション コアとの親和性を無視してしまうため、ハード リアルタイム要件が求められる場合は最適ではありません。

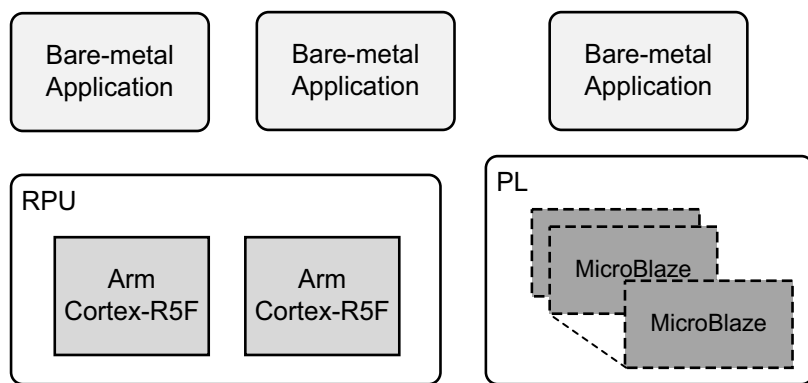
非対称型マルチプロセッシング (AMP)

AMP モードでは、どのプロセッサで何を実行するかを厳密に制御して複数のプロセッサを使用します。SMP とは異なり、AMP にはさまざまな利用方法があります。このセクションでは、複雑度の異なる AMP の 2 つの利用法について説明します。

このモードでは、各プロセッサでどのコードを実行するかをソフトウェア開発者が決定してから、各 CPU のソフトウェア実行ファイルを含むブート イメージをコンパイルして生成する必要があります。RPU の Arm Cortex-R5F プロセッサを AMP モードで使用するにより、ソフト リアルタイムではなくハード リアルタイム要件を満たすことができます (Cortex-R5F では SMP はサポートされない)。

複数のアプリケーションを個別に開発し、プロセッサ間通信 (IPC) オプションを使用してアプリケーションどうしが通信できるようにこれらをプログラムできます。この機能の詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「割り込み」の章を参照してください。

この AMP モードは、PL の MicroBlaze™ プロセッサで動作するアプリケーションや APU のプロセッサで動作するアプリケーションにも適用できます。次の図に、RPU と PL でアプリケーションを実行し、アプリケーション間の通信を使用しない AMP の例を示します。



X19225-071317

図 6-2: RPU と PL でベアメタル アプリケーションを実行した AMP の例

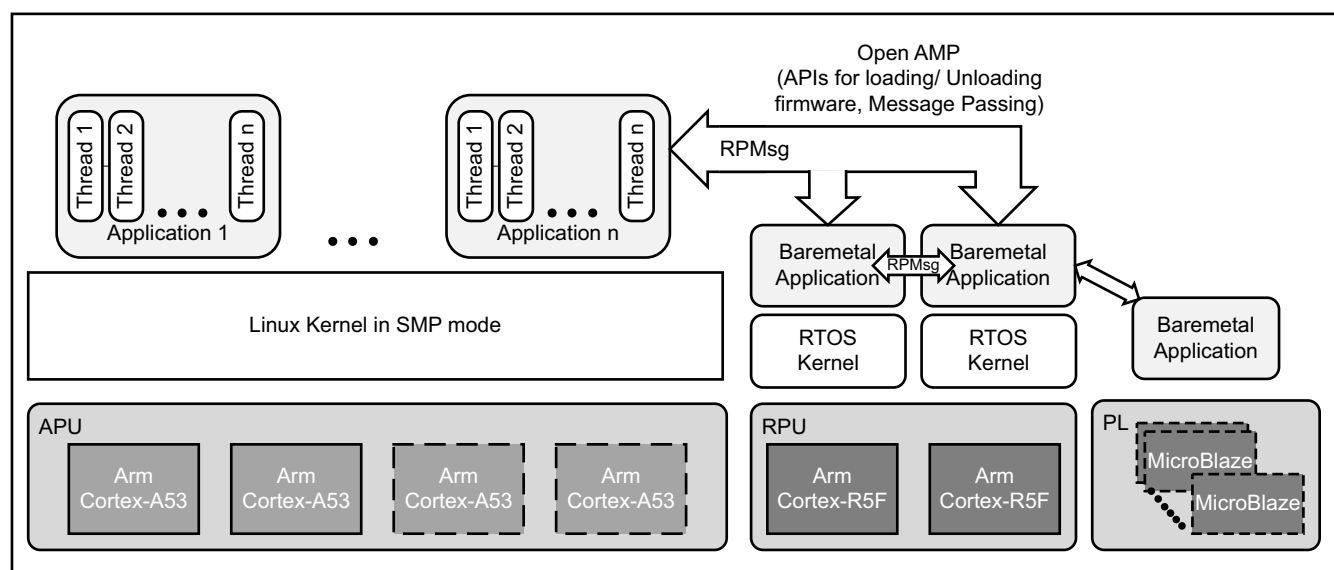
OpenAMP

OpenAMP フレームワークには次のようなメカニズムが用意されています。

- 。 ファームウェアのロードとアンロード
- 。 標準 API を使用したアプリケーション間通信

次の図は、OpenAMP フレームワークを使用した RPU の OpenAMP 機能とハード リアルタイム機能の例を示しています。

この例では、APU で動作する Linux アプリケーションが RPU アプリケーションのロードとアンロードを実行します。これにより、開発者は必要に応じて異なる処理に特化したアルゴリズムを RPU の各プロセッシング エンジンにロードし、非常に確定的な性能を得ることができます。



X14839-063017

図 6-3: OpenAMP フレームワークを使用した SMP + AMP の例

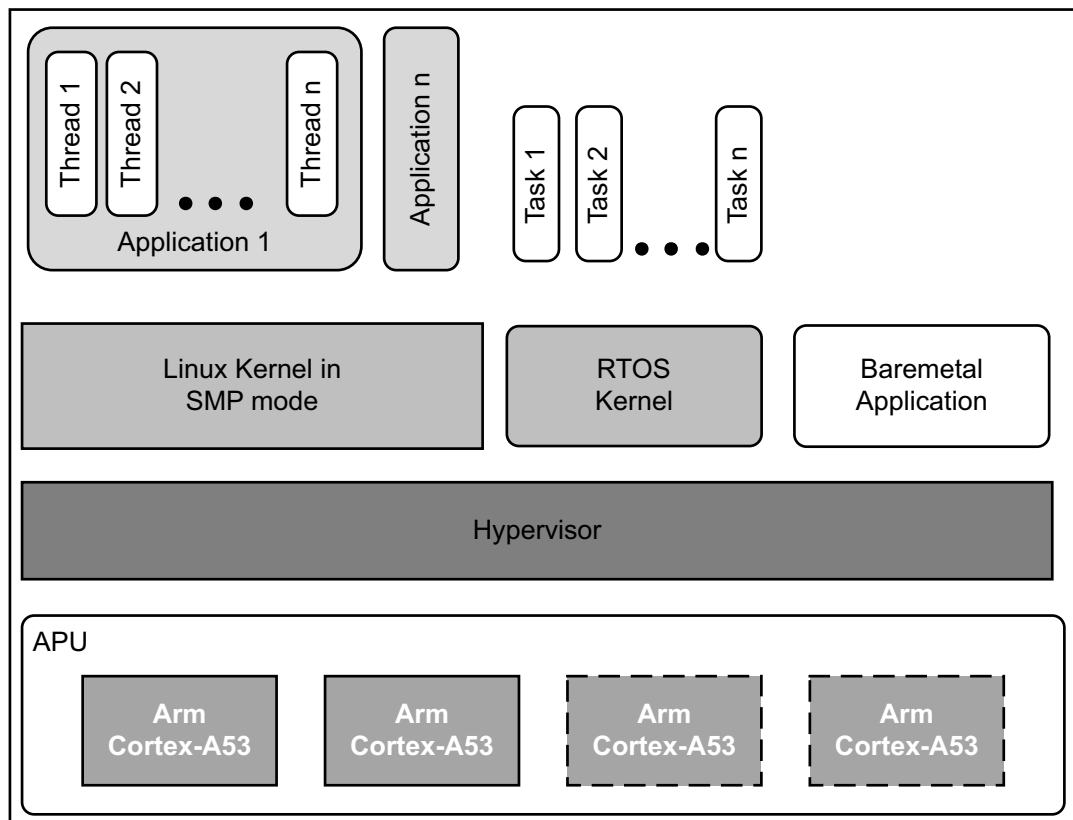
OpenAMP フレームワークの詳細は、『Zynq デバイス用 Libmetal および OpenAMP ユーザー ガイド』(UG1186) [\[参照 16\]](#) を参照してください。

ハイパーバイザーを使用した仮想化

Zynq UltraScale+ MPSoC デバイスは、Arm Cortex-A53 プロセッサ、割り込みコントローラー、および Arm System MMU (SMMU) のハードウェア仮想化拡張機能をサポートしており、APU 内で SMP と AMP など、さまざまなオペレーティング システムを柔軟に組み合わせることができます。

次の図は、1 つのハイパーバイザー上で Linux などの SMP 対応 OS、リアルタイム オペレーティング システム (RTOS)、およびベアメタル アプリケーションが協調動作する例を示しています。

これにより、各アプリケーションはそれぞれの動作モードで個別に開発できます。



X14840-063017

図 6-4: ハイパーバイザーを使用した例

Zynq UltraScale+ MPSoC に含まれるハードウェア仮想化およびハイパーバイザーによって、標準オペレーティング システムとそのアプリケーションは比較的容易に動作させることができますが、ハイパーバイザーを追加すると、電力管理、FPGA ビットストリーム管理、OpenAMP ソフトウェア スタック、およびセキュリティ アクセラレータ アクセスなど、下層のシステム ファームウェアを利用する下位のシステム機能の設計が複雑になります。このため、システム構築および実装の中でも特にこれらの点については早期から念頭において設計を進めることを推奨します。

Zynq UltraScale+ MPSoC での Xen などのハイパーバイザーの使用の詳細は、Xen ハイパーバイザーのウェブサイト [参照 41] を参照してください。

ハイパーバイザーの使用

ザイリンクスは、オープンソースのハイパーバイザー Xen を Zynq UltraScale+ MPSoC デバイス用にポーティングして配布しています。Xen ハイパーバイザーを使用すると、1 つのコンピューティングプラットフォーム上で複数のオペレーティングシステムを実行できます。Xen ハイパーバイザーはハードウェア上で直接動作し、CPU、メモリ、割り込みを管理します。このハイパーバイザー上で複数の OS を実行できます。これらの OS をドメインまたは仮想マシン (VM) と呼びます。

Xen ハイパーバイザーを使用すると、複数のオペレーティングシステムおよびそれらの標準アプリケーションを比較的容易に同時実行できます。ただし、Xen にはゲストオペレーティングシステムにシステム機能へのアクセスを提供する汎用インターフェイスはありません。このため、このセクションで説明する注意事項に従う必要があります。

Xen ハイパーバイザーは、1 つのドメイン (ドメイン 0) と 1 つまたは複数のゲストドメインを制御します。制御ドメインには、次のような特権機能があります。

- 。 ハードウェアへの直接アクセス
- 。 システムの I/O 機能へのアクセス処理
- 。 ほかの仮想マシンとの通信

また、外部からアクセス可能な制御インターフェイスもあり、このインターフェイスを利用してシステムを制御します。各ゲストドメイン上ではそれぞれの OS とアプリケーションを実行します。ゲストドメインはハードウェアから完全に隔離されます。

Xen ハイパーバイザーを使用して複数の OS を実行するには、ホスト OS をセットアップして 1 つまたは複数のゲスト OS を追加します。

注記: Xen ハイパーバイザーは、PetaLinux ツール内で選択可能なコンポーネントとして入手可能ですが、ザイリンクス GIT からダウンロードできます。ザイリンクスが提供する Linux および Xen ソフトウェアを利用することで、独自の Linux ゲストコンフィギュレーションを構築できます。Linux 以外のゲスト OS を構築する場合は、サードパーティのソフトウェアと技術が必要です。詳細は、PetaLinux の製品ページ [\[参照 2\]](#) を参照してください。

システムブートおよびコンフィギュレーション

はじめに

Zynq® UltraScale+™ MPSoC デバイスは、QSPI フラッシュ、SD カード、USB デバイス ファームウェア アップグレード (DFU) ホスト、NAND フラッシュ ドライブなどからのブートをサポートしています。この章では、セキュア モードと非セキュア モードの両方において各種デバイスからのブート プロセスについて説明します。

ブート プロセスの概要

複数ステージからなるブート プロセスは、プラットフォーム管理ユニット (PMU) とコンフィギュレーション セキュリティ ユニット (CSU) で管理および実行されます。デバイスは、認証されたブート イメージを使用するセキュア モード、または認証されていないブート イメージを使用する非セキュア モードでブートできます。ブートの各ステージは次のとおりです。

- プリコンフィギュレーション ステージ: このステージは主に PMU が制御し、PMU ROM を実行してシステムをセットアップします。リセットおよびウェークアップに関するプロセスはすべて PMU が処理します。
- コンフィギュレーション ステージ: このステージでは、PS の FSBL (第 1 段階ブートローダー) コードをオンチップ RAM (OCM) にロードします。セキュア ブートと非セキュア ブートのどちらのモードもサポートされます。ブート ヘッダーを介して、Cortex-R5F プロセッサと Cortex-A53 プロセッサのいずれかに対して FSBL を実行できます。Cortex-R5F プロセッサでは、ロックステップ モードもサポートされています。
- ポストコンフィギュレーション ステージ: FSBL の実行が開始すると、Zynq UltraScale+ MPSoC デバイスはポスト コンフィギュレーション ステージに移行します。

ブート フロー

Zynq UltraScale+ MPSoC アーキテクチャには、セキュア ブートと非セキュア ブートの2つのフローがあります。以降のセクションでは、各種プロセッサを立ち上げて必要なブート タスクを実行するシーケンスについて説明します。

注記: 各セクションの図では、必須コンポーネントとオプション コンポーネントをすべて含むブート フロー全体を示しています。

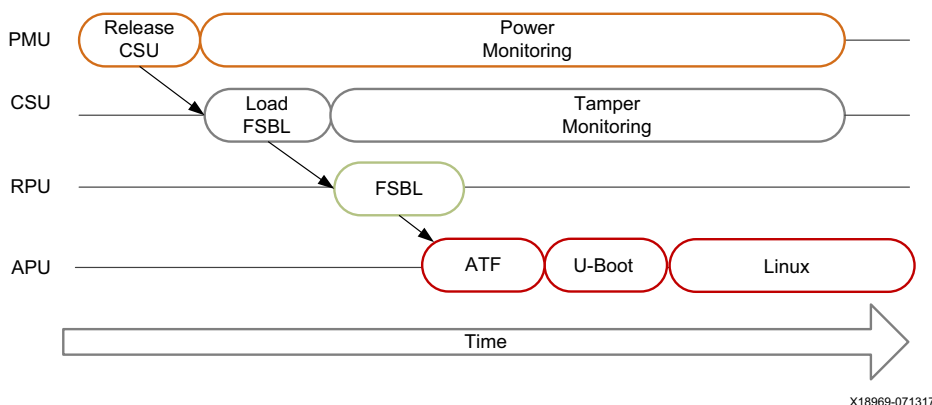


図 7-1: ブート フローの例

非セキュア ブート フロー

非セキュア ブート モードでは、PMU はコンフィギュレーション セキュリティ ユニット (CSU) のリセットを解放後、PMU サーバー モードに移行して電力を監視します。PMU によってリセットから解放されると、CSU は FSBL を OCM にロードします。PMU ファームウェアは PMU RAM にロードされます。OCM の FSBL と並行して、PMU ファームウェアは PMU RAM から実行されます。この例では、FSBL は RPU と ATF にロードされます。U-Boot と Linux は APU にロードされます。FSBL は RPU または APU のいずれかにロードできます。その他のブート コンフィギュレーションでは、RPU は APU と完全に独立して起動および動作させることができます (その逆も同様)。

- APU では、FSBL から ATF へのハンドオフの後に ATF が実行されます。ATF へのハンドオフの後、通常は U-Boot などの SSBL (第2段階ブートローダー) が実行され、Linux などの OS がロードされます。
- RPU では、FSBL の実行はソフトウェア アプリケーションに渡されます。
- その後、Linux が実行可能なソフトウェアをロードします。

注記: 対称型マルチプロセッシング (SMP) モードでは、OS が複数の Cortex™-A53 プロセッサを管理します。

セキュア ブートのフロー

セキュア ブート モードでは、PMU はコンフィギュレーション セキュリティ ユニット (CSU) のリセットを解放後、PMU サーバー モードに移行して電力を監視します。

PMU によってリセットから解放されると、CSU は FSBL またはユーザー アプリケーションによって認証が要求されているかを確認します。

CSU は次を実行します。

- 認証チェックを実行し、このチェックにパスした場合のみ次へ進みます。そして、イメージに暗号化されたパーティションがあるかどうかをチェックします。
- 暗号化されたパーティションを検出した場合、CSU は復号化を実行し、FSBL を OCM にロードします。

CSU の詳細は、「[コンフィギュレーション セキュリティ ユニット \(CSU\)](#)」を参照してください。

APU では、FSBL から ATF へのハンドオフが実行されます。ATF はそれを受けて U-Boot を実行し、Linux などの OS をロードします。その後、Linux が実行可能なソフトウェアをロードします。同様に、FSBL はロードする各パーティションの認証および暗号化をチェックします。認証および、暗号化されていた場合は復号化に問題がないことが確認されたパーティションのみが FSBL によってロードされます。

注記: セキュア ブート モードでは、psu_coresight_0 は標準出力ポートとしてはサポートされません。

ブート イメージの生成

SDK に含まれる Bootgen ユーティリティを使用すると、開発したアプリケーションを Zynq UltraScale+ MPSoC デバイスでブートするのに適した 1 つのブート イメージファイルが生成されます。Bootgen 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対する暗号化、認証およびチェックサムをサポートします。

このユーティリティは、ブート イメージフォーマット (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。



重要: .BIF ファイルには ATF より上位のビットストリームを含める必要があります。

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 イメージをビルドするには、次の手順を実行します。

1. BIF ファイルを作成します。
2. Bootgen ユーティリティを実行してバイナリ ファイルを作成します。
3. (QEMU の場合): バイナリ ファイルをブート デバイスの種類に応じたイメージフォーマットに変換します。

Bootgen の詳細は、[第 16 章「ブート イメージの生成」](#)を参照してください。

ブート モード

利用可能なブート モードの概要は、表 7-4 を参照してください。利用可能なブート モードの一覧は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「ブートおよびコンフィギュレーション」の章を参照してください。

QSPI24 および QSPI32 ブート モード

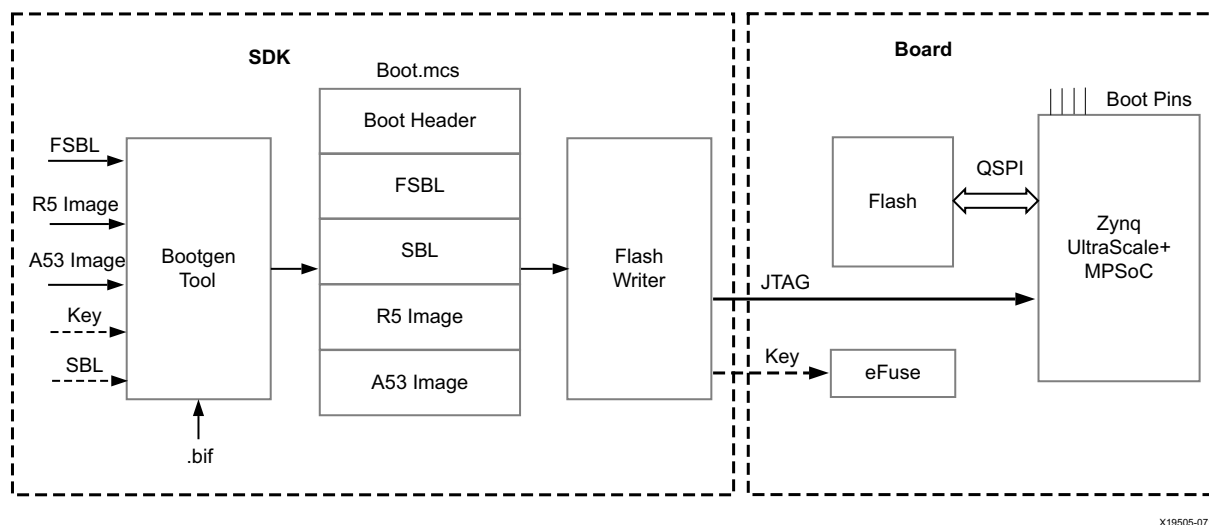
QSPI24 および QSPI32 ブート モードは次の機能をサポートします。

- 。 シングル QSPI フラッシュ メモリ 24 (QSPI24) およびシングル QSPI フラッシュ メモリ 32 (QSPI32) では x1、x2、および x4 読み出しモード
- 。 デュアル QSPI で x8 読み出しモード。
- 。 マルチブートのイメージ検索。
- 。 BSP ドライバー用 I/O モード (FSBL ではサポートされない)。

bootROM が x8 モードで最初の 256Mb を検索します。QSPI24 および QSPI32 ブート モード (QSPI24/32 デバイスが 128Mb 未満) でマルチブートを使用する場合は、128Mb 未満のメモリ アドレスに収まるように複数イメージを配置します。QSPI24 ブート モードのピン設定は 0x1 です。

注記: QSPI デュアル スタック (x8) のブートはサポートされません。QSPI STR (Single Transmission Rate) のみサポートされます。シングル クワッド SPI メモリ (x1、x2 および x4) が、XIP (eXecute-In-Place) をサポートする唯一のブート モードです。

次の図に、QSPI モードでのブート例を示します。



X19505-071317

図 7-2: QSPI モードでのブート

QSPI24/QSPI32 ブート イメージを作成するには、Bootgen に次のファイルを入力します。

- 。 FSBL ELF
- 。 U-Boot などの SSBL (第2段階ブートローダー)、または Cortex-R5F/Cortex-A53 アプリケーション ELF
- 。 認証/暗号化キー (オプション)

認証/暗号化の詳細は、第8章「セキュリティ機能」を参照してください。

Bootgen によって boot.mcs および boot.bin バイナリ ファイルが生成されたら、フラッシュライターを使用してこれらを QSPI24/QSPI32 フラッシュに書き込みます。MCS は Intel HEX フォーマット ファイルで、信頼性を高めるためにチェックサムを含んでいます。

注記: QSPI24 ブート モードのピン設定は 0x2 です。

SD ブート モード

SD ブート (バージョン 3.0) は次の機能をサポートしています。

- 。 FAT 16/32 ファイル システムによるブート イメージの読み出し。
- 。 マルチブートのイメージ検索。マルチブートの最大ファイル数は 8,192。

次の図に、SD モードでの Linux のブート例を示します。

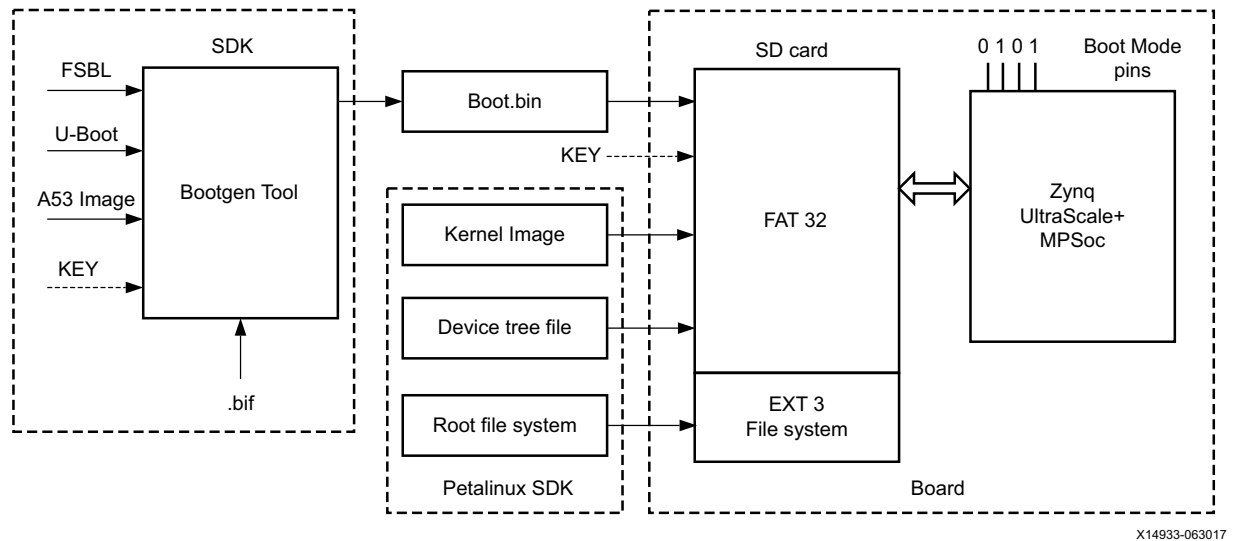


図 7-3: SD モードでのブート

SD ブート イメージを作成するには、Bootgen に次のファイルを入力します。

- FSBL ELF
- Cortex-R5F/Cortex-A53 アプリケーション ELF
- 認証/暗号化キー (オプション)

Bootgen からは boot.bin バイナリ ファイルが生成されます。この boot.bin ファイルを SD カード リーダーを使用して SD カードに書き込みます。

PetaLinux で、次を実行します。

1. Linux カーネル イメージ、デバイス ツリー ファイル、およびルート ファイル システムを構築します。
2. これらのファイルを SD カードにコピーします。

これで、フォーマット済み SD カードの FAT32 パーティションに boot.bin、カーネル イメージ、およびデバイス ツリー ファイルが格納され、EXT 3 パーティションにルート ファイル システムが格納されます。



重要: SD1 からブートする場合はブート ピンを 0x5 に設定する必要があります。SD0 からブートする場合はブート ピンを 0x3 に設定し、レベルシフターを用いた SD からブートする場合はブート ピンを 0xE に設定します。

eMMC18 ブート モード

eMMC18 ブート (バージョン 4.5) は次の機能をサポートしています。

- FAT 16/32 ファイル システムによるブート イメージの読み出し。
- マルチブートのイメージ検索。マルチブートの最大ファイル数は 8,192。

次の図に、eMMC18 モードでの Linux のブート例を示します。

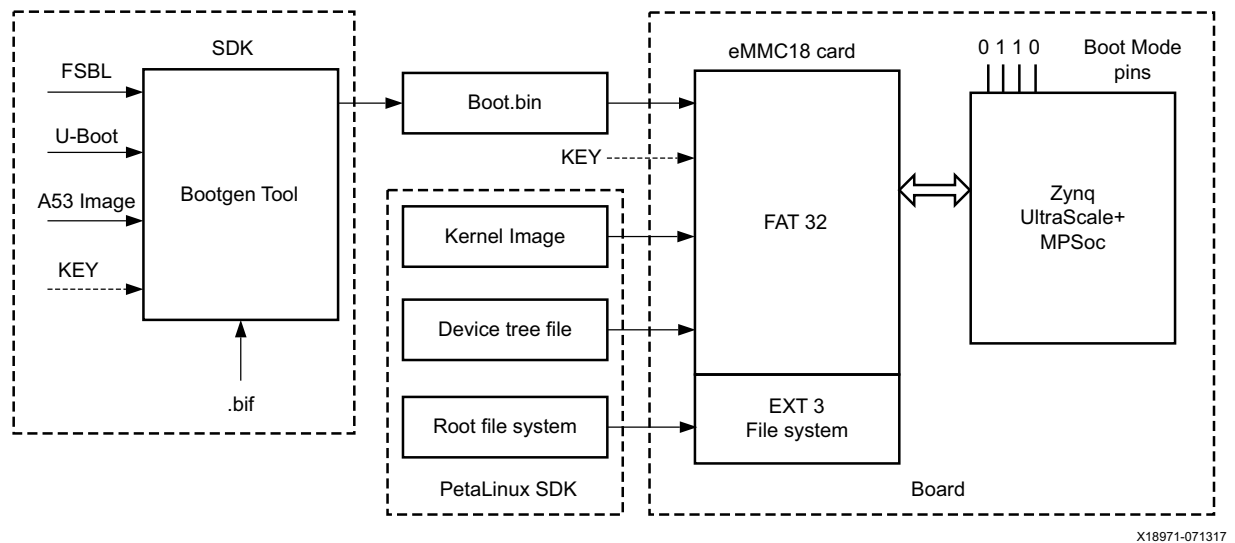


図 7-4: eMMC18 モードでのブート

eMMC18 ブート イメージを作成するには、Bootgen に次のファイルを入力します。

- FSBL ELF
- Cortex-R5F/Cortex-A53 アプリケーション ELF
- 認証/暗号化キー (オプション)

Bootgen からは boot.bin バイナリ ファイルが生成されます。この boot.bin ファイルを eMMC18 カード リーダーを使用して eMMC18 カードに書き込みます。

PetaLinux で、次を実行します。

- Linux カーネル イメージ、デバイス ツリー ファイル、およびルート ファイル システムを構築します。
- これらのファイルを eMMC18 カードにコピーします。

これで、フォーマット済み eMMC18 カードの FAT32 パーティションに boot.bin、カーネル イメージ、およびデバイス ツリー ファイルが格納され、EXT3 パーティションにルート ファイル システムが格納されます。

NAND ブート モード

NAND ブートは、8 ビット幅でのブート イメージ読み出しとマルチブートのイメージ検索をサポートします。次の図に、NAND モードでの Linux のブート例を示します。

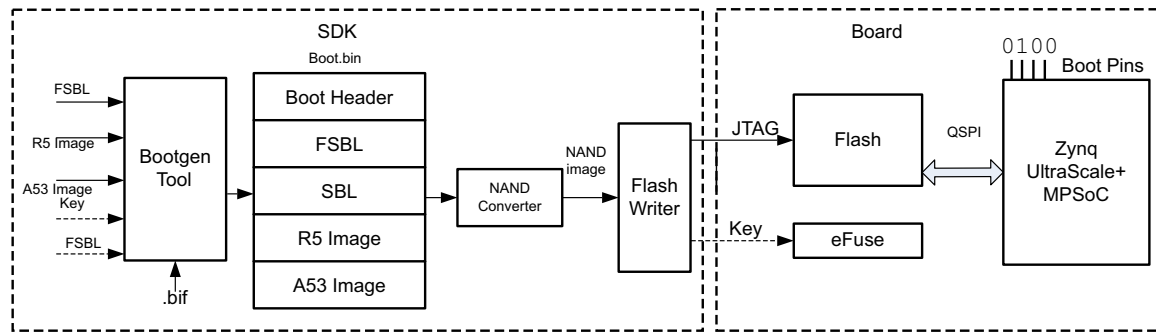


図 7-5: NAND モードでのブート

NAND ブート イメージを作成するには、Bootgen に次のファイルを入力します。

- FSBL ELF
- Cortex-R5F アプリケーション ELF/Cortex-A53 アプリケーション ELF
- 認証/暗号化キー (オプション)

Bootgen からは boot.bin バイナリ ファイルが生成されます。この NAND ブータブル イメージを、フラッシュライターを使用して NAND フラッシュに書き込みます。



重要: NAND からブートするには、ブート ピンを 0x4 に設定する必要があります。

JTAG ブート モード

PS に必要なソフトウェア イメージと PL に必要なハードウェア イメージは、JTAG を使用して個別にダウンロードできます。

JTAG ブート モードの設定は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「ブートおよびコンフィギュレーション」の章を参照してください。



重要: JTAG モードではセキュア ブートはサポートされません。

USB ブート モード

USB ブート モードは USB 2.0 のみ使用できます。USB ブート モードでは、セキュア ブートと非セキュア ブートの両方のモードがサポートされます。ただし DDR を使用しないシステムでは、マルチブート、フォールバック イメージ、および XIP はサポートされません。

注記: FSBL では、USB ブート モードはデフォルトで無効となっています。USB ブート モードを有効にするには、`xfsbl_config.h` ファイルで `FSBL_USB_EXCLUDE_VAL` を 0 に設定します。

表 7-1: USB ブート モードの詳細

モード ピン	0x7
MIO ピン	MIO[63:52]
非セキュア	あり
セキュア	あり
符号	あり
モード	スレーブ

USB ブート モードには、`dfu-utils` がインストールされたホスト PC が必要です。このホストとデバイスを USB デバイス ケーブルで接続します。ホストには、bootROM コード用の `boot.bin` (`fsbl.elf` しか含まれない) が 1 つ、FSBL 用の別の `boot_all.bin` が含まれている必要があります。USB ブート モードでボードの電源を入れたら、次のコマンドを実行します。

- Linux の場合:
 - `dfu-util -D boot.bin`
 ファイルをデバイスにダウンロードし、その後 FSBL を実行します。
 - `dfu-util -D boot_all.bin`
 ファイルをデバイスにダウンロードします。FSBL は必要な処理を実行します。
- Windows の場合:
 - `dfu-util.exe -D boot.bin`
 ファイルをデバイスにダウンロードし、その後 FSBL を実行します。
 - `dfu-util.exe -D boot_all.bin`
 ファイルをデバイスにダウンロードします。FSBL は必要な処理を実行します。

`boot.bin` および `boot_all.bin` のサイズは、それぞれ利用可能な OCM と DDR のサイズに制限されます。

セカンダリ ブート モード

Zynq UltraScale+ MPSoC アーキテクチャでは、2 つのブート デバイスを使用できます。bootROM が FSBL (およびオプションで PMU ファームウェア) をロードするために使用するブート デバイスを、プライマリ ブート モードと呼びます。FSBL がそれ以外のパーティションをロードするために使用するブート デバイスを、セカンダリ ブート モードと呼びます。セカンダリ ブート モードとしては、QSPI24、QSPI32、SD0、eMMC、SD1、SD1-ls、NAND および USB がサポートされます。



重要: セカンダリ ブート モードを使用する場合、プライマリ ブート デバイスと別のデバイスを指定する必要があります。たとえば、プライマリ ブート デバイスが QSPI32 の場合、QSPI24 をセカンダリ ブート モードとすることはできません。この場合、SD0、eMMC、SD1、SD1-Is、NAND、または USB をセカンダリ ブート モードとして使用します。プライマリ ブート デバイスとセカンダリ ブート デバイスは、任意の組み合わせが可能です。

注記: デフォルトでは、セカンダリ ブート モードはプライマリ ブート モードと同じで、ブート イメージは1つのみとなります。

詳細は、FSBL の Wiki ページの「[What is Secondary Boot Mode](#)」(英語) を参照してください。

ブート フローの詳細

Zynq UltraScale+ MPSoC デバイスのプラットフォーム管理ユニット (PMU) は、主要なブート前タスクを処理します。

ブート中にデバイスのデフォルト電源ステートの設定、RAM の初期化、そしてメモリとレジスタのテストを実行するために、PMU ROM は ROM から実行します。PMU は、これらのタスクを実行した後にシステムの制御をコンフィギュレーション セキュリティ ユニット (CSU) に渡して、サービス モードへ遷移します。このモードの PMU は、レジスタ インターフェイスを介してシステムからの割り込み要求に対応したり、専用 I/O を介してハードウェアからの割り込み要求に対応してプラットフォーム管理サービスを実行します。

プリブート シーケンス

次の表に、プリブート シーケンスで PMU が実行するタスクを示します。

表 7-2: プリブート シーケンス

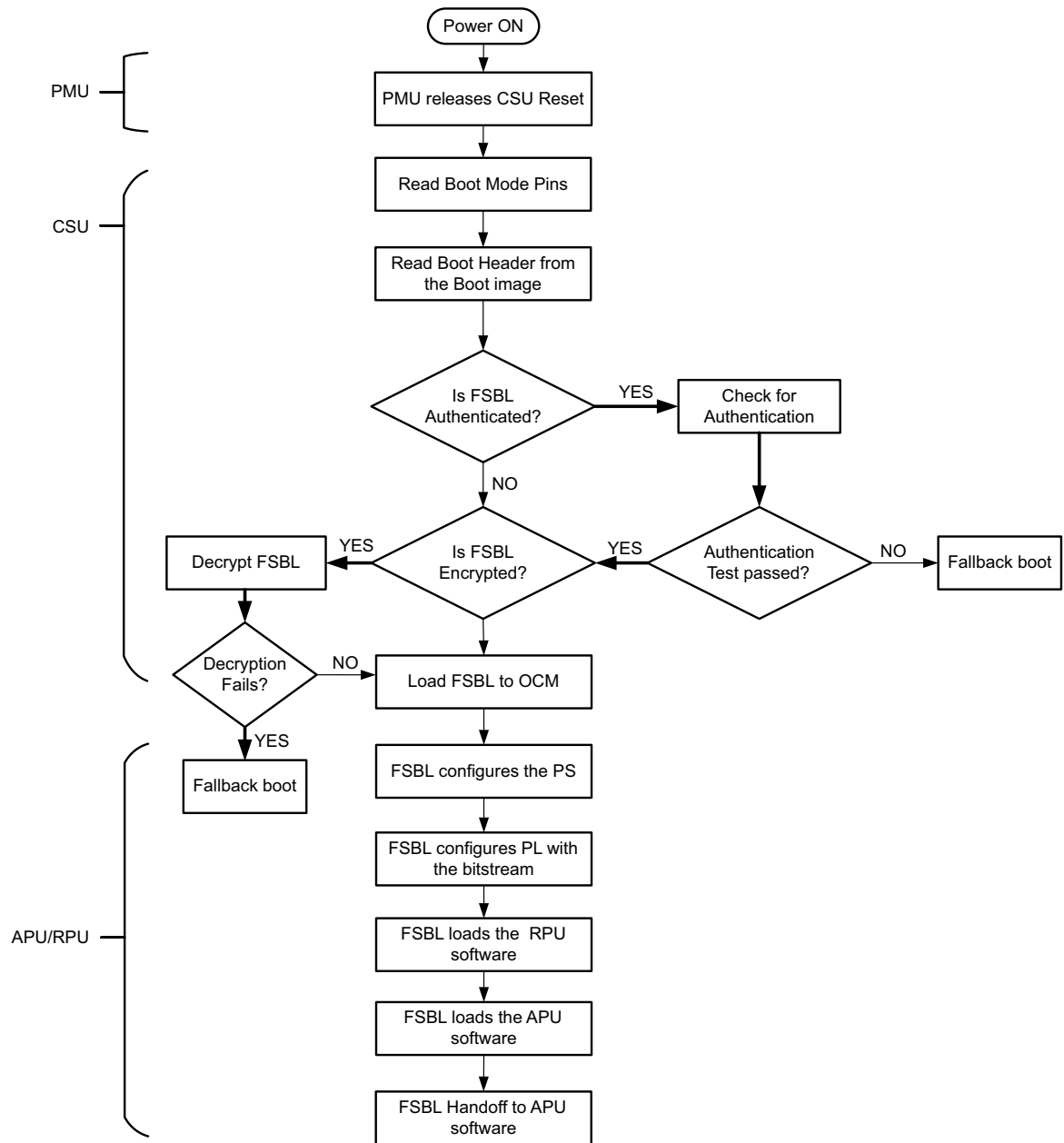
プリブート タスク	説明
0	MicroBlaze™ プロセッサを初期化する。主なステートをキャプチャする。
1	LPD および FPD をスキャンおよびクリアする。
2	システム モニターを初期化する。
3	MBIST クロックに使用される PLL を初期化する。
4	PMU RAM をすべて 0 に設定する。
5	PLL を検証する。MBIST クロックを設定する。
6	電源を検証する。
7	FPD メモリを修復する (必要な場合)。
8	LPD と FPD をすべて 0 に設定し、メモリ セルフテストを初期化する。
9	無効化されたすべての IP の電源を切断する。
10	CSU を解放、またはエラー ステートに遷移する。
11	サービス モードに遷移する。

CSU のリセットが解放されると同時に、CSU ブート ROM を実行し、次のシーケンスを実行します。

1. OCM を初期化します。
2. POR 時にブート モードのピン ストラッピングを取り込んだブート モード レジスタを読み出して、ブート モードを判別します。

3. CSU は、FSBL の読み込みとオプションの PMU ファームウェア コードを続行します。ファームウェア コードは、PMU ユニットが実行できるソフトウェア コードです。このコードは PMU の RAM から実行します。詳細は、第9章「プラットフォーム管理」を参照してください。

図 7-6 に、詳細なブート フローを示します。



X14935-070717

図 7-6: 詳細なブート フローの例

ブート シーケンスにおける FPD の無効化

FPD の電源が瞬間的にオンになることによる FPD ロックアウトを回避するには、次の手順を実行します。

- bootROM の実行が完了するまで電源を供給します。
- FSBL 実行中に FPD の電源を遮断するには、PMU_GLOBAL_REQ_PWRDWN_STATUS レジスタの FPD ビット 22 をセットします。
- ブート プロセスのその後のステージで FPD に再び電源を供給するには、PMU_GLOBAL_REQ_PWRUP_STATUS レジスタのビット 22 をセットします。

FSBL のブート前に FPD の電源が供給されていない場合は、次の手順を実行します。

- R5 の電源をオンにします。
- レジスタがセットされ、FPD がロックしたことが示されます。POR が保留となり、FPD ではリセットまたはクリア シーケンスは実行できません。
- R5 は FPD のロック ステータスを PMU_GLOBAL_REQ_ISO_STATUS レジスタのビット 4 から読み出すことができます。
- これで、PMU_GLOBAL_REQ_PWRUP_STATUS のビット 22 はセットされることはありません。
- FPD ノードを再びプリングアップするには、ノードに電源を供給し、POR を発行する必要があります。

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 7-7 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。

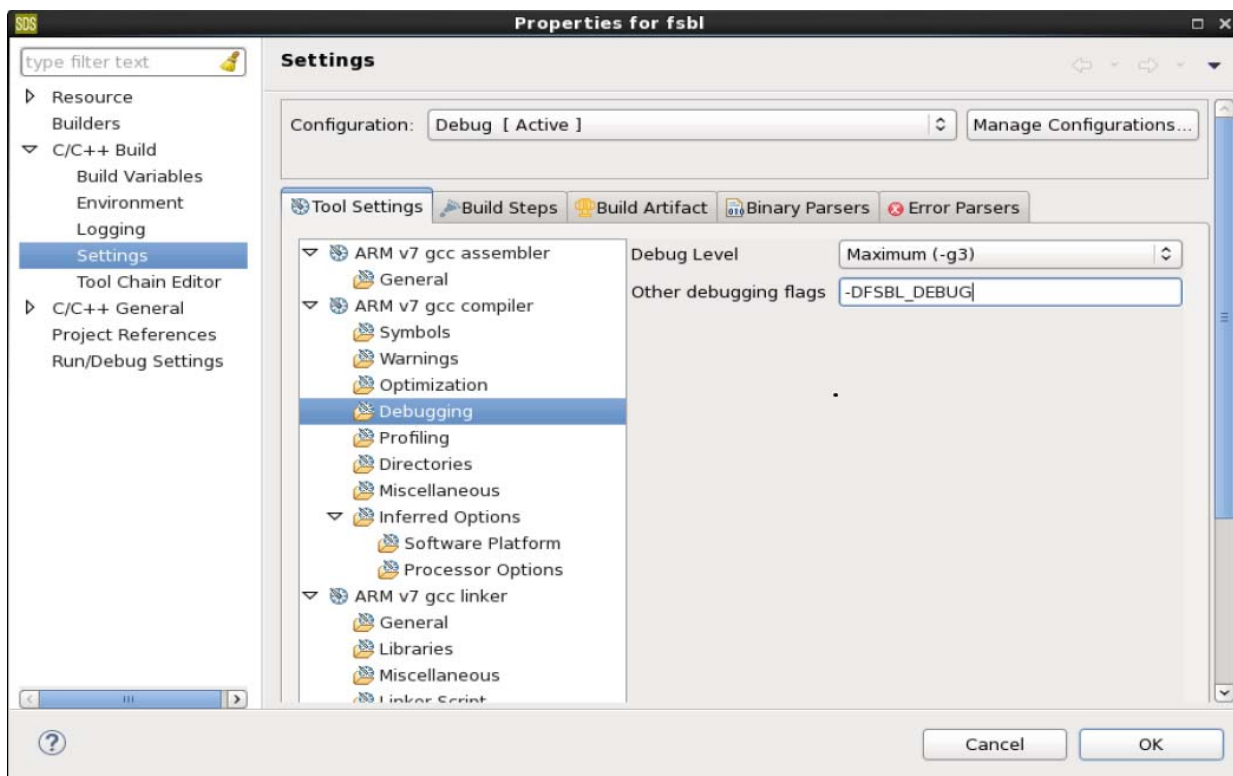


図 7-7: FSBL デバッグ フラグ

次の表に、FSBL コンパイル フラグを示します。

表 7-3: FSBL コンパイル フラグ

フラグ	説明
FSBL_DEBUG	基本情報およびエラーがあればエラー ステータスを出力する。
FSBL_DEBUG_INFO	基本情報のほかに、フォーマット指定子付きの出力を有効化する。
FSBL_DEBUG_DETAILED	転送されたすべてのデータを含む情報を出力する。
FSBL_NAND_EXCLUDE	NAND サポート コードを除外する。
FSBL_QSPI_EXCLUDE	QSPI サポート コードを除外する。
FSBL_SD_EXCLUDE	SD サポート コードを除外する。
FSBL_RSA_EXCLUDE	認証コードを除外する。
FSBL_AES_EXCLUDE	復号化コードを除外する。
FSBL_BS_EXCLUDE	ビットストリーム コードを除外する。
FSBL_SHA2_EXCLUDE	SHA-2 コードを除外する。
FSBL_WDT_EXCLUDE	WDT サポート コードを除外する。

表 7-3: FSBL コンパイル フラグ (続き)

フラグ	説明
FSBL_USB_EXCLUDE	USB コードを除外する。 デフォルトは 1 です。 0 にすると、USB ブート モードが有効になります。
FSBL_FORCE_ENC_EXCLUDE_VAL	ENC_ONLY ヒューズがプログラムされていても、すべてのパーティションの暗号化を強制しない。デフォルトは 0 です。 デフォルトでは、ENC_ONLY がプログラムされている場合、FSBL はすべてのパーティションに対して強制的に暗号化を有効にします。

詳細は、[FSBL の Wiki ページ \(英語\)](#) の「I'm unable to build FSBL due to size issues, how can I reduce its footprint」を参照してください。



重要: SHA-2 は現在のリリースでは非推奨です。SHA-2 ではなく SHA-3 の使用を推奨します。

デバッグ プリントの有効化

デバッグ プリントを有効にするには、次の手順を実行します。

1. FSBL_DEBUG_INFO シンボルを定義します。SDK で [FSBL application project] を右クリックし、[C/C++ Build] の [Settings] をクリックします。次に、[Tool Settings] タブの [Arm A53 gcc compiler] の下にある [Symbols] をクリックします。
2. 追加ボタン (+) をクリックして、「FSBL_DEBUG_INFO」と入力します。
3. [OK] をクリックして [Properties] ダイアログ ボックスを閉じます。

FSBL のデバッグの詳細は、[FSBL の Wiki ページ \(英語\)](#) を参照してください。

フォールバックおよびマルチブート フロー

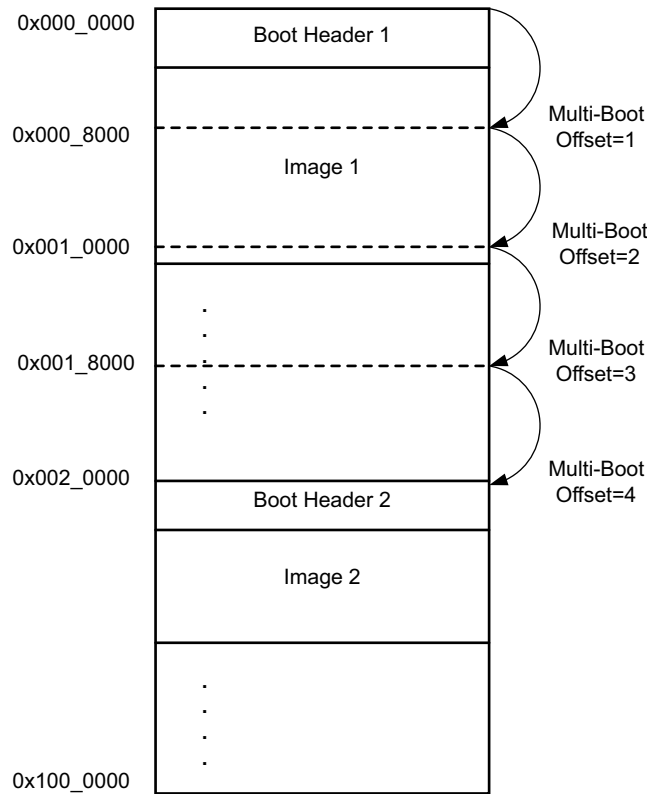
Zynq UltraScale+ MPSoC デバイスでは、CSU bootROM がマルチブートおよびフォールバック ブート イメージ検索をサポートしています。これは、コンフィギュレーション セキュリティ ユニット (CSU) の bootROM がブート デバイス内を検索してロードすべき有効なイメージを検出するものです。このシーケンスは次のとおりです。

- 。 bootROM はフラッシュの 32KB オフセットで有効なイメージ ID 文字列 (イメージ ID XLNX) を検索します。
- 。 有効な ID を検出すると、bootROM はヘッダーのチェックサムを検証します。
- 。 チェックサムが有効なら、bootROM はそのイメージをロードします。これは、フラッシュ内に複数のイメージが含まれることを見込んでいます。

マルチブートの場合:

- 。 FSBL またはユーザー アプリケーションがブート イメージ検索を開始して別のブート イメージを選択する必要があります。
- 。 このイメージ検索を開始するには、FSBL またはユーザー アプリケーションでマルチブート オフセットを更新して目的のブート イメージを指し示すようにし、CRL_APB レジスタに書き込んでソフト リセットを生成します。

次の図に、フォールバックの例およびマルチブート フローを示します。



X14936-071217

図 7-8: マルチブート フロー

注記: セキュア ブートと非セキュア ブートの両方に同じフローを適用できます。

フォールバック ブート フローの例では、次のシーケンスが実行されます。

- CSU bootROM は最初に 0x000_0000 にあるブート イメージをロードします。
- このイメージが破損しているか復号化/認証に失敗した場合、CSU bootROM はマルチブート オフセットを 1 つインクリメントして 0x000_8000 (32KB オフセット) で有効なブート イメージを検索します。
- このオフセットで有効な ID 値を検出できない場合、CSU bootROM はマルチブート オフセットをさらに 1 つインクリメントして次の 32KB 境界アドレスで有効なブート イメージを検索します。
- CSU bootROM は、検索可能な範囲内で有効なブート イメージが検出されるまでこの動作を繰り返します。この例では、次のイメージは 0x002_0000 にあり、マルチブート オフセットは 4 です。
- このマルチブート フローの例でアドレス 0x002_0000 にある 2 番目のイメージをロードするには、FSBL またはユーザー アプリケーションでマルチブート オフセットを 4 に設定します。マルチブート オフセットを更新したら、システムのソフト リセットを生成します。

次の表に、各ブート デバイスのマルチブート イメージ検索範囲を示します。

表 7-4: 各ブート デバイスのマルチブート イメージ検索範囲

ブート デバイス	マルチブート イメージ検索範囲
QSPI シングル (24 ビット)	16MB
QSPI デュアル (24 ビット)	32MB
QSPI シングル (32 ビット)	256MB
QSPI デュアル (32 ビット)	512MB
NAND	128MB
SD/eMMC	8,191 のブート ファイル
USB	N/A

FSBL のビルド プロセス

FSBL (第 1 段階ブートローダー) はビットストリームがあれば FPGA をコンフィギュレーションし、オペレーティングシステム (OS) イメージ、スタンドアロン (SA) イメージ、SSBL (第 2 段階ブートローダー) イメージのいずれかを不揮発性メモリ (NAND/SD/eMMC/QSPI) から RAM (DDR/TCM/OCM) へロードします。リセットを終了するには、Cortex-R5F プロセッサまたは Cortex-A53 プロセッサユニットが必要です。FSBL は複数のパーティションをサポートします。各パーティションには、コード イメージまたはビットストリームを格納できます。各パーティションは、必要に応じて認証/復号化できます。認証/復号化が完了したら、FSBL が OCM にロードされ、CSU bootROM によってハンドオフされます。

注記: カスタム FSBL を作成する場合、OCM はサイズが 256KB であり、CSU bootROM から利用できることを理解しておく必要があります。FSBL のサイズは約 170KB で、OCM に収まります。USB ブート モードを使用する場合、PMU ファームウェアを CSU bootROM ではなく FSBL でロードする必要があります。これは、CSU bootROM でロードする boot.bin は、サイズを 256KB 未満とする必要があるためです。

ソフトウェア開発キットを使用した FSBL の作成

ソフトウェア開発キット (SDK) を使用して FSBL を作成するには、次の手順を実行します。

1. 次のコマンドを実行して SDK を起動します。

```
xsdk
```
2. [File] → [New] → [Application Project] をクリックして、New Project ウィザードを開きます。FSBL プロジェクトの名前を入力します。
3. [Target Hardware] で、ZynqMP の定義済みハードウェア プラットフォーム ([ZCU102_hw_platform] など) を選択します。

または、.hdf ファイルから新規/カスタム プラットフォームを作成する場合は、[New] をクリックして、新規ハードウェアプラットフォームを作成します。次の [New Hardware Project] ダイアログボックスでプロジェクト名を入力し、[Target Hardware Specification] で [Browse] をクリックし、HDF ファイルを選択します。新規ハードウェアプラットフォームが作成されます。

4. New Project ウィザードで、`psu_cortexa53_0` プロセッサまたは `psu_cortexr5_0` プロセッサを選択します。
`psu_cortexa53` を選択した場合、ドロップダウン リストから 64 ビット (デフォルト) または 32 ビットのコンパイラを選択します。
5. [Next] をクリックし、[Zynq MP FSBL] を選択します。
6. [Finish] をクリックすると、A53/R5 FSBL が生成されます。これにより、FSBL コードと BSP がビルドされます。

注記: FSBL は、A53_0 (AArch32 または AArch64)、R5_0 または R5_Lockstep のいずれかでのみ実行できます。

FSBL の詳細は、[FSBL の Wiki ページ \(英語\)](#) を参照してください。

注記: FSBL のデバッグ プリントは、デフォルトでは無効です (FSBL バナーを除く)。詳細は、「[デバッグ プリントの有効化](#)」を参照してください。

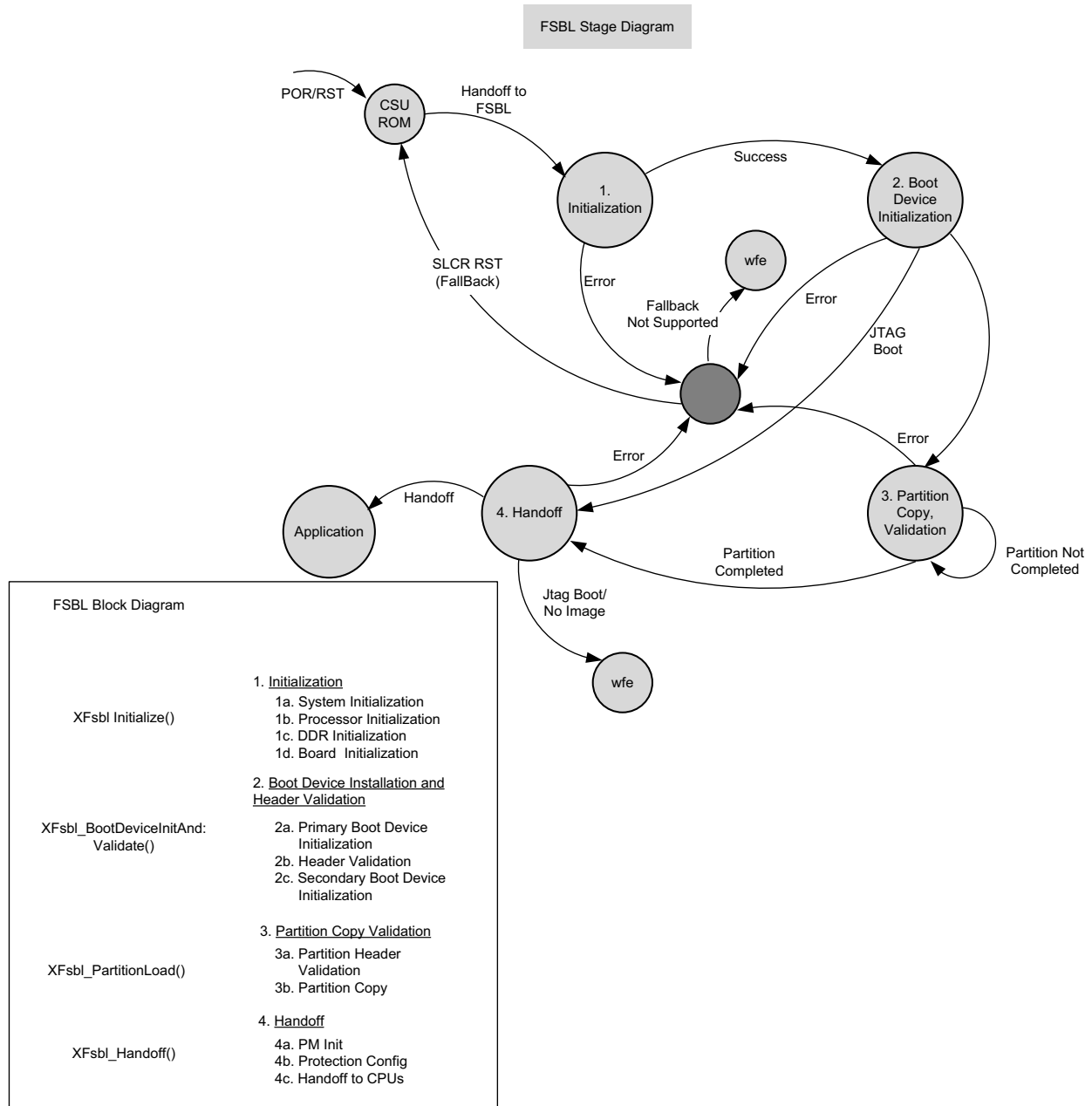
ソース ファイル (FSBL または BSP) を変更するには、ソース ファイルを開いて編集し、保存します。プロジェクトをビルドします。FSBL プロジェクトの Debug/Release フォルダに、.elf ファイルが生成されます。

FSBL の動作フェーズ

FSBL の動作は、次の 4 つのステージで構成されます。

- 初期化
- ブート デバイスの初期化
- パーティションのロード
- ハンドオフ

図 7-9 に、FSBL の動作ステージを示します。



X19962-101917

図 7-9: FSBL の動作ステージ

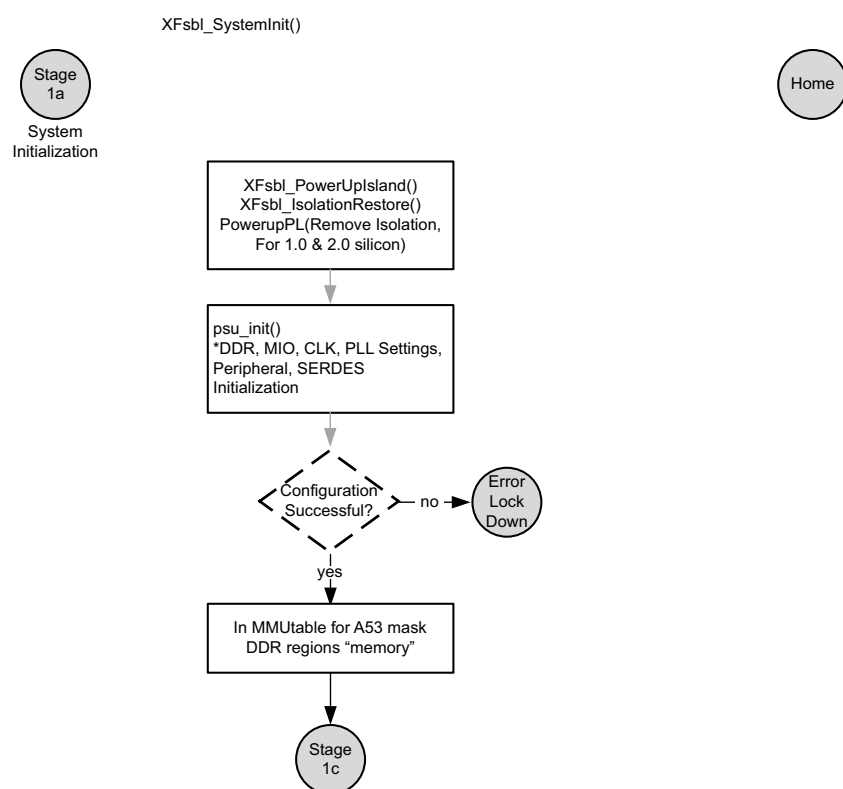
初期化

初期化ステージは、次の4つの内部ステージで構成されます。

1. [XFsbI_SystemInit](#)
2. [XFsbI_ProcessorInit](#)
3. [DDR の初期化](#) (APU 専用のリセットの場合は不要)
4. [XFsbI_BoardInit](#)

XFsbI_SystemInit

この関数は、1.0 および 2.0 シリコンで PL の電源を投入し、PS-PL の分離を解除します。psu-init で指定されたクロックとペリフェラルを初期化します。APU 専用のリセットでは、この関数は呼び出されません。



X19952-101917

図 7-10: FSBL システムの初期化

XFsbI_ProcessorInit

このステージでは、プロセッサの初期化が開始します。A53 の場合は命令およびデータ キャッシュ、L2 キャッシュ、MMU 設定、スタック ポインター、R5F の場合は命令およびデータ キャッシュ、MPU 設定、メモリ 領域、スタック ポインター、および TCM 設定をセットアップします。これら設定のほとんどは BSP コードの初期化で実行されます。IVT ベクターは、A53 の場合は OCM の先頭、R5F の場合は TCM の先頭 (lowvec では 0x0、highvec では 0xffff0000) に変更されます。

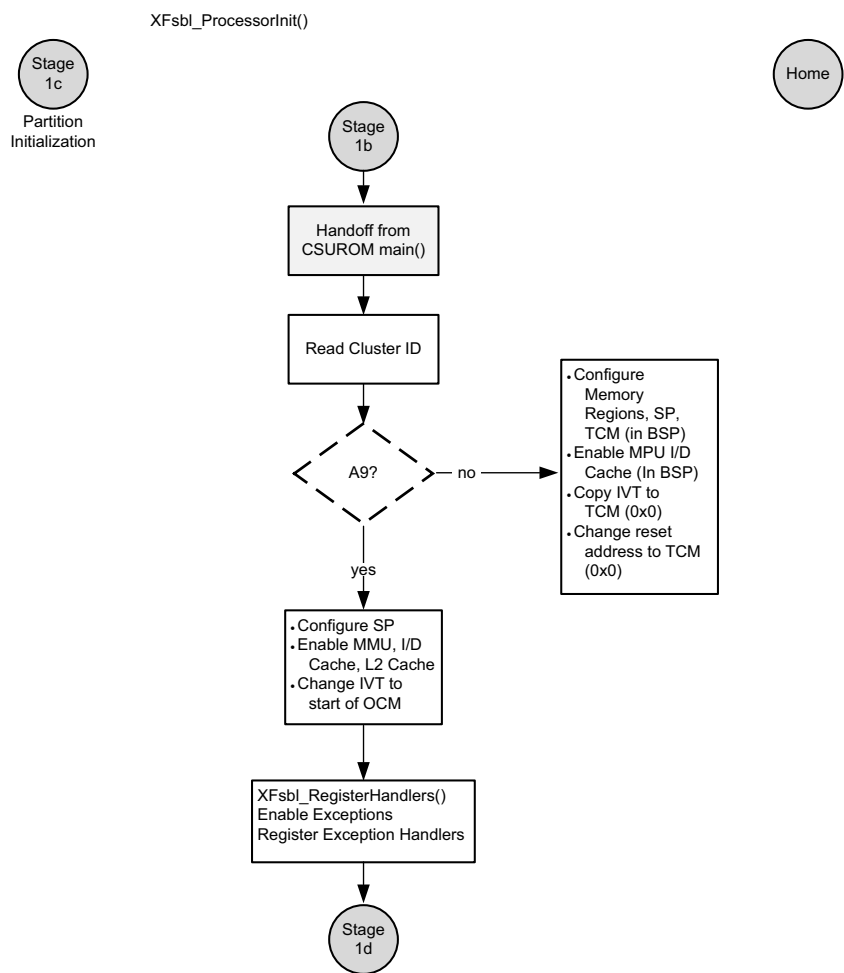
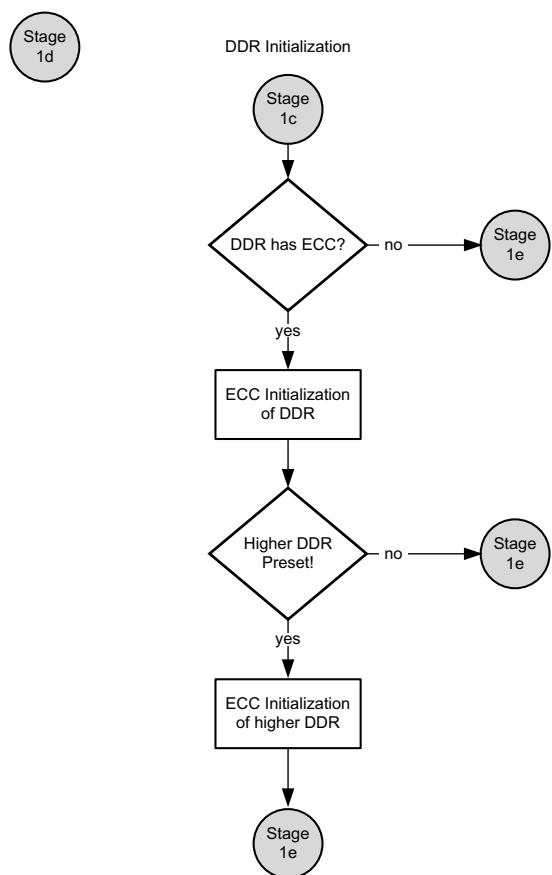


図 7-11: プロセッサの初期化

DDR の初期化

このステージでは、DDR が初期化されます。APU 専用のリセットでは、この関数は呼び出されません。



X19957-101917

図 7-12: DDR の初期化

XFsbI_BoardInit

この関数は、必要に応じてボード固有の初期化を実行します。特に重要な機能として、GT レーンと IIC を設定します。

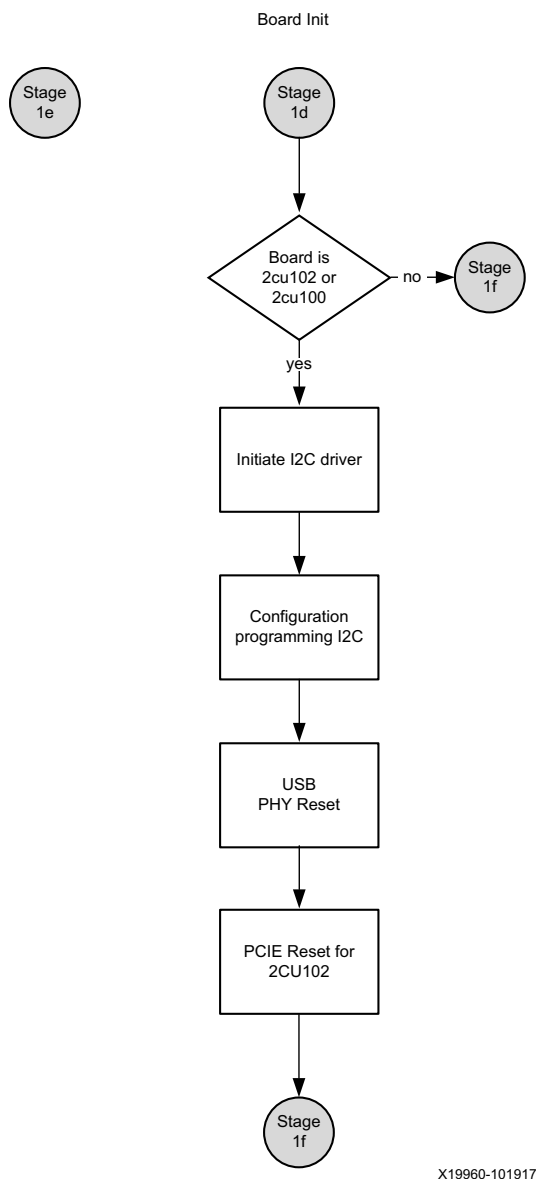


図 7-13: ボードの初期化

XFsbbl_ValidateHeader

コピー機能を使用して、FSBL はブート ヘッダーの属性およびイメージオフセット アドレスを読み出します。EFUSE ビットを読み出して認証が必要かどうかをチェックします。イメージ ヘッダーを読み出してイメージ ヘッダー テーブルを検証します。その後、イメージ ヘッダーの Partition Present Device 属性を読み出します。この値が 0 以外の場合、セカンダリ ブート デバイスであることを示します。0 の場合は、セカンダリ ブート デバイスがプライマリ ブート デバイスと同じであることを示します。

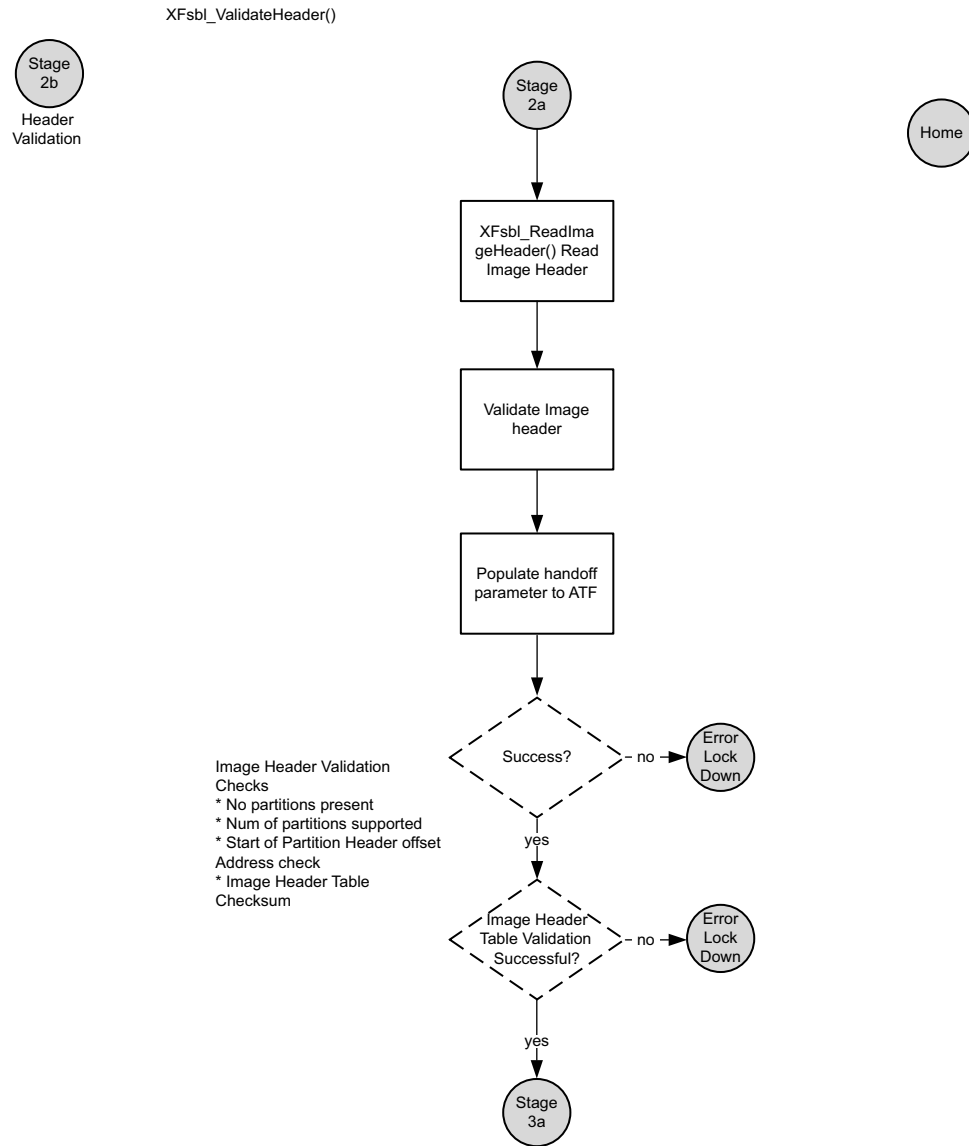
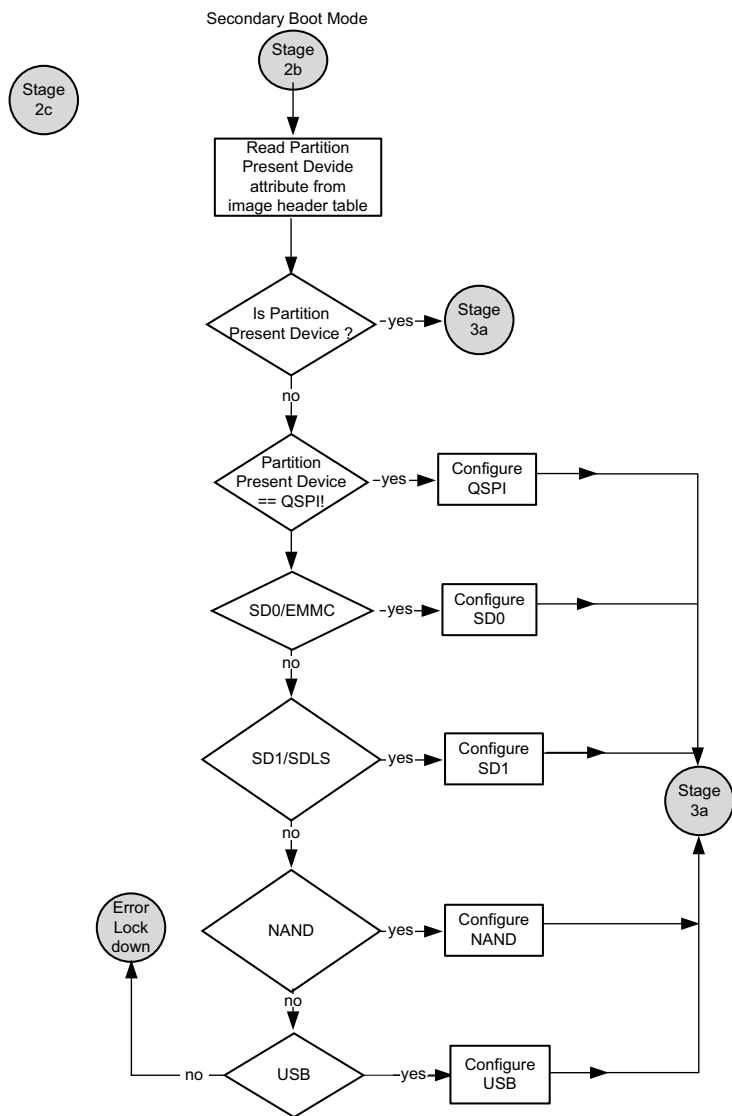


図 7-15: ヘッダーの検証

XFsbI_SecondaryBootDeviceInit

イメージ ヘッダー テーブルの Partition Present Device 属性が 0 以外の値の場合、この関数が呼び出されます。セカンダリ ブート デバイスのドライバーを初期化すると、FSBL はセカンダリ ブート デバイスを使用してすべてのパーティションをロードします。



X19961-101917

図 7-16: セカンダリ ブート モード

XFsbbl_SetATFHandoffParams

FSBL の次にロード可能なパーティションが ATF とします。ATF はその他すべてのパーティションをロードできるため、ハンドオフ構造体が渡されます。

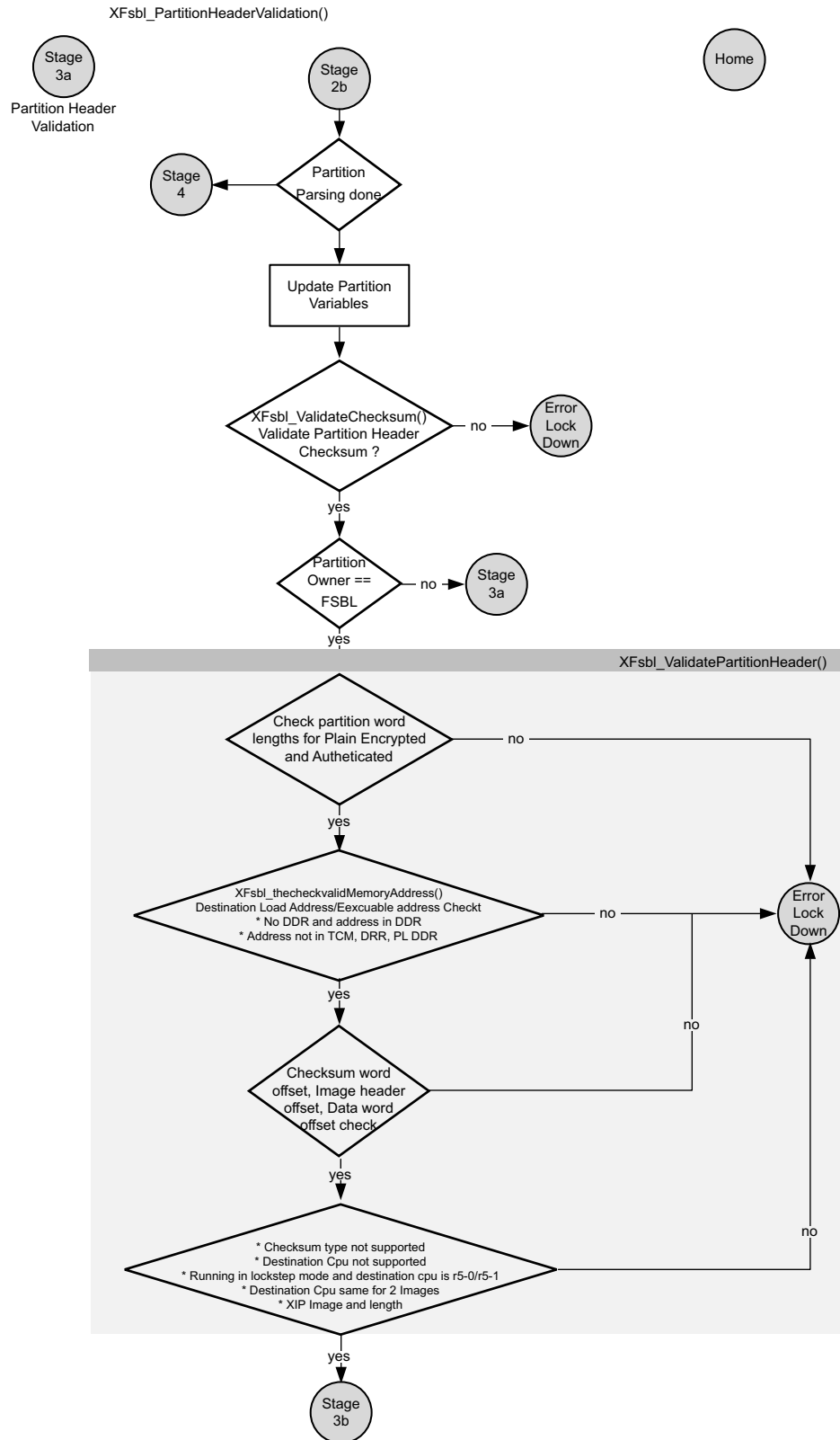
アプリケーションの最初のパーティションには、0 以外の実行アドレスがあります。このアプリケーションのその他のパーティションは、実行アドレスが 0 です。このため、最初のパーティション以外で実行アドレスが 0 でないパーティションを探します。CPU は A53 とします。

この関数は、Arm トラステッド ファームウェア (ATF) へのハンドオフ パラメーターを設定します。最初の引数は FSBL パーティション ヘッダーから取得します。これらのパラメーターを格納しているハンドオフ構造体へのポインターは、PMU_GLOBAL.GLOBAL_GEN_STORAGE6 レジスタに格納されており、これを ATF が読み出します。この構造体にはマジック文字の「X」、「L」、「N」、および「X」が含まれ、その後にパーティションの総数と各パーティションの実行アドレスが続きます。

パーティションのロード

XFsbbl_PartitionHeaderValidation

パーティション ヘッダーをさまざまな基準で検証します。必要なパーティション変数はすべてこのステージで更新されます。パーティション オーナーが FSBL でない場合、パーティションは無視され、次のパーティションに進みます。

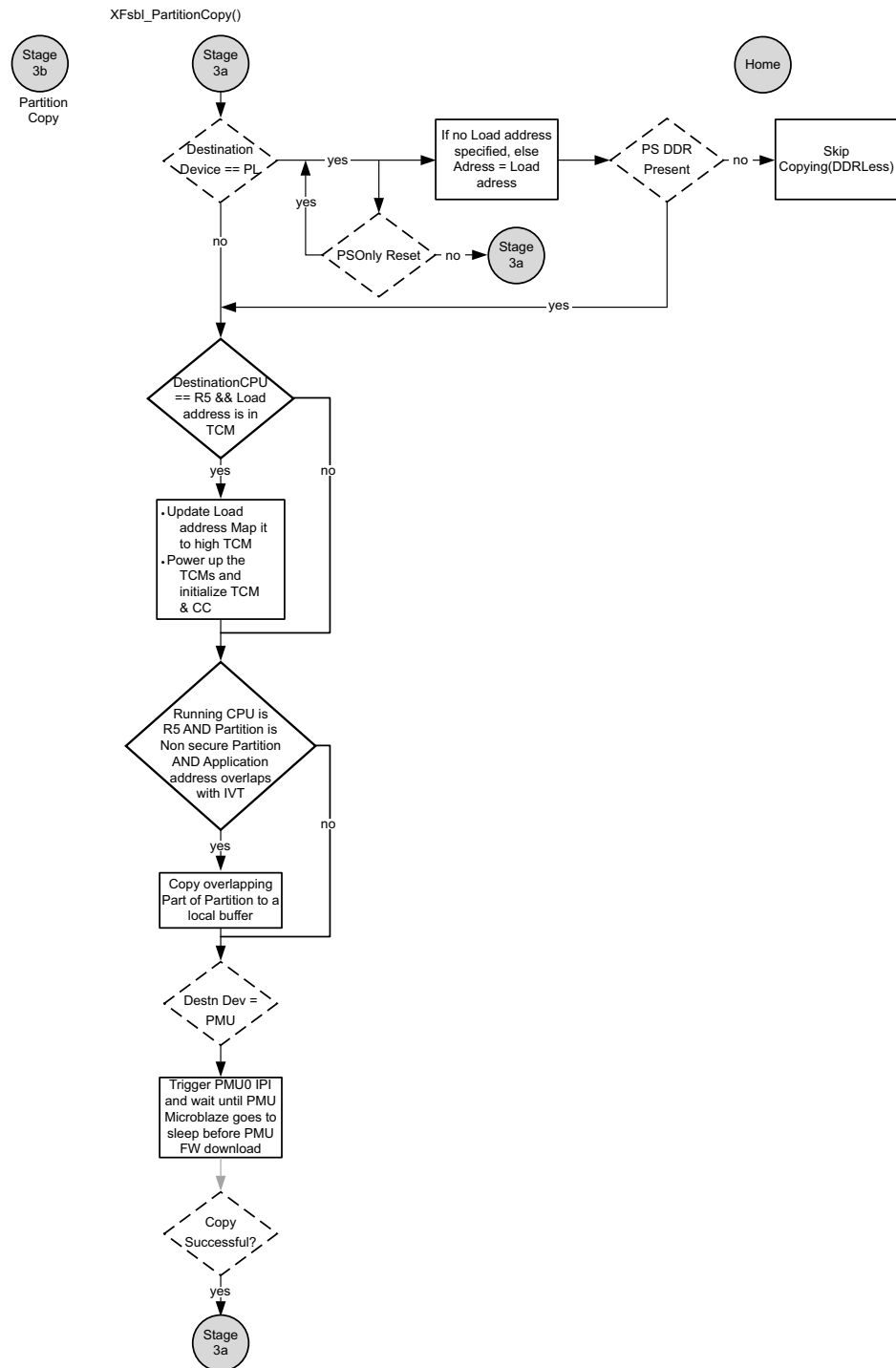


X19951-101917

図 7-17: パーティション ヘッダーの検証

XFsbI_PartitionCopy

パーティションは DDR、TCM、OCM、PMU RAM のいずれかにコピーされます。

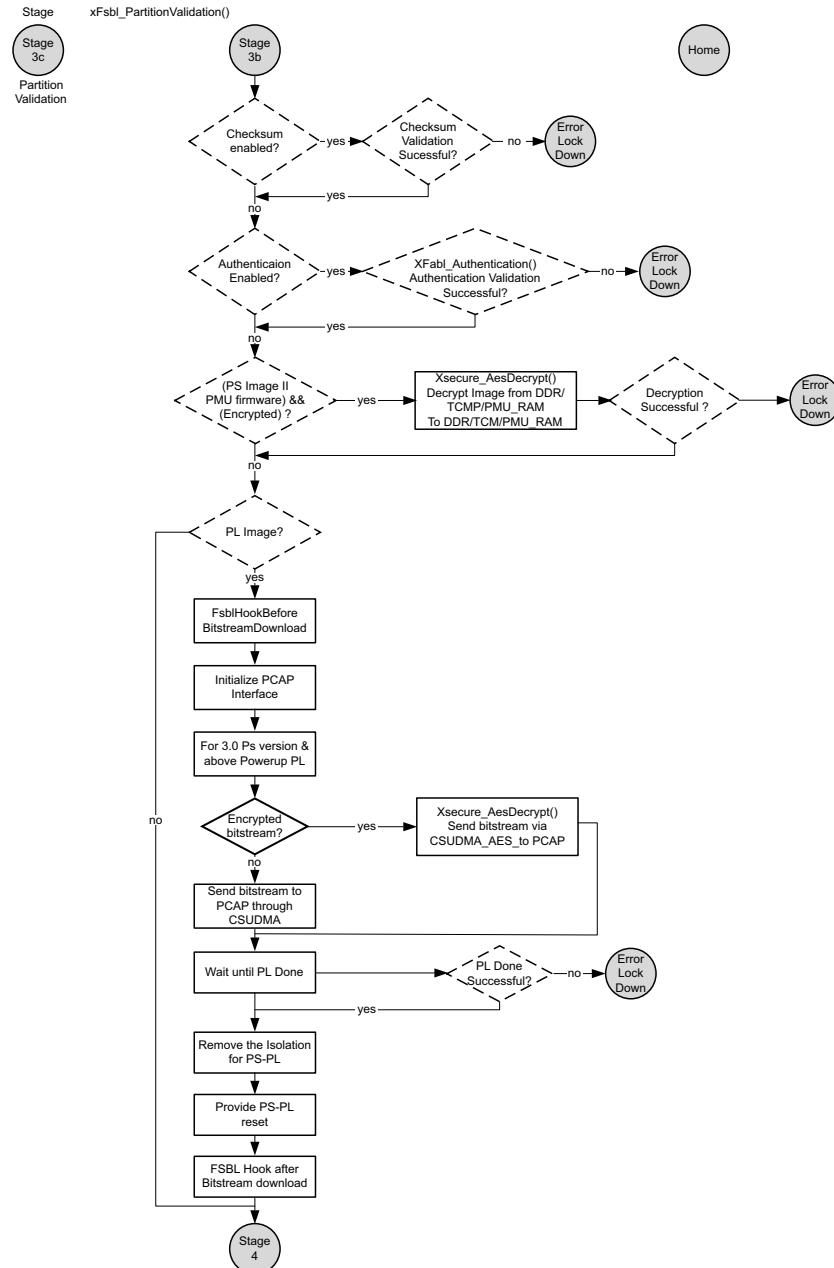


X19950-101917

図 7-18: パーティション コピー

XFsbI_PartitionValidation

パーティションは、パーティション属性に基づいて検証されます。チェックサムビットが有効な場合、まずパーティションのチェックサムを検証し、次に認証フラグに基づいて認証を実行します。暗号化フラグをセットしている場合、パーティションは復号化されてからデスティネーションへコピーされます。



X19949-101917

図 7-19: パーティション検証関数

ハンドオフ

このステージでは、`psu_init` からの `protection_config` 関数を実行した後、ハンドオフ機能が実行されます。また、PS-PL の分離も無条件に解除されます。R5F 上で動作するパーティションが存在する場合は、R5F がリセットから解放されます。ブート イメージのビルド時にユーザーが指定した内容に基づいて、R5F は `lowvec` モードまたは `highvec` モードでブートします。ハンドオフ アドレスは、`lowvec` モードの場合 `0x0` で、`highvec` モードの場合 `0xffff0000` です。`lowvec/highvec` の情報は、ブート イメージのビルド時にユーザーが指定します。ほかのすべての PS イメージが完了したら、実行中の CPU イメージが PC の値を更新してその CPU にハンドオフされます。実行中の CPU のイメージがない場合、CPU は `wfe` ループに入ります。

プロセッサを実行しても、ほかのプロセッサにパラメーターは渡されません。パーティション間の通信は、PMU グローバルレジスタの読み出し/書き込みによって実行します。

実行中のプロセッサへのハンドオフには、APU リセットの場合と同様、実行中のプロセッサのプログラム カウンター (PC) を更新する必要があります。ほかのプロセッサへのハンドオフには、そのプロセッサの PC を更新してリセットから解放する必要があります。

XFsbI_PmInit

この関数は、プロセッサ間割り込み (IPI) を初期化および設定します。次に、PM コンフィギュレーション オブジェクト アドレスを IPI バッファに書き込み、ターゲットへの IPI をトリガーします。すると、PMU ファームウェアがコンフィギュレーション オブジェクトの指定に従ってデバイス ノードを読み出し、設定します。

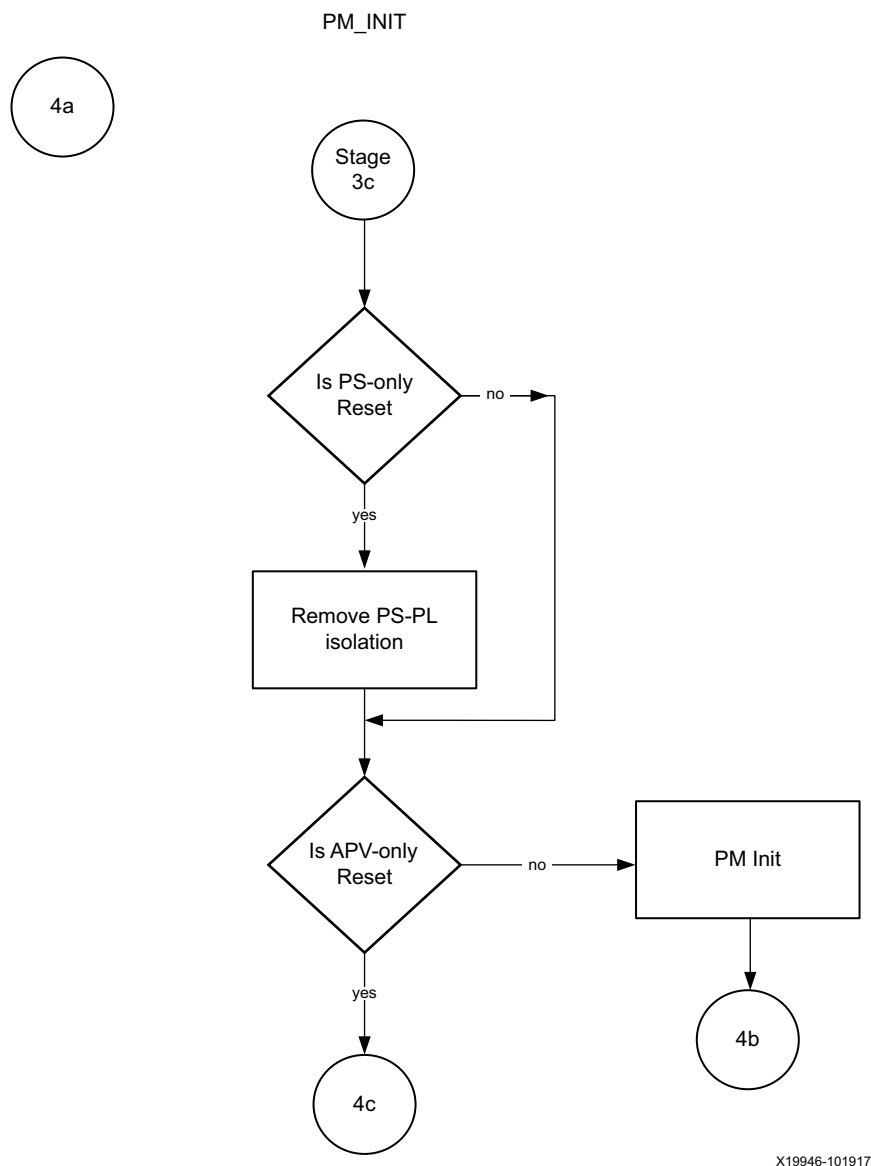
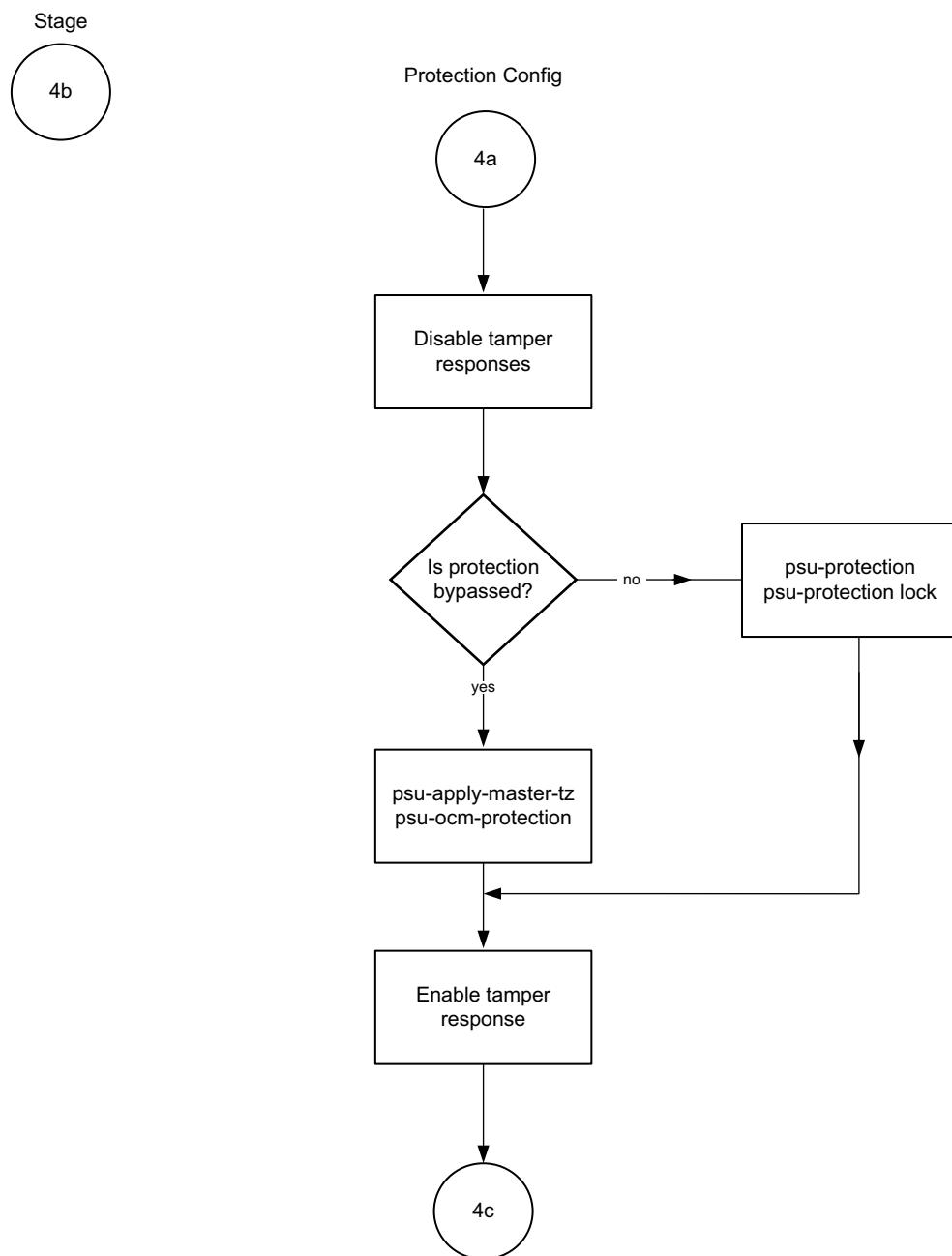


図 7-20: PM の初期化

保護の設定

このステージでは、psu_init からの protection_config 関数が実行されます。現在、保護の設定はデフォルトでバイパスされています。分離は、現在 OCM のみに限定されています。保護のバイパスまたは適用は、このステージで実行されます。



X19947-101917

図 7-21: 保護の設定

ハンドオフ

実行中のプロセッサへのハンドオフには、APU リセットの場合と同様、実行中のプロセッサのプログラム カウンター (PC) を更新する必要があります。ほかのプロセッサへのハンドオフには、そのプロセッサの PC を更新してリセットから解放する必要があります。R5 上で動作するパーティションが存在する場合、A53 の FSBL によって R5 がリセットから解放されます。ブート イメージのビルド時にユーザーが指定した内容に基づいて、R5 は lowvec モードまたは highvec モードでブートします。ハンドオフ アドレスは、lowvec モードの場合 0x0 で、highvec モードの場合 0xffff0000 です。

lowvec/highvec の情報は、ブート イメージのビルド時にユーザーが指定する必要があります。ほかのすべての PS イメージが完了したら、実行中の CPU イメージが PC の値を更新してその CPU にハンドオフされます。実行中の CPU のイメージがない場合、CPU は wfe ループに入ります。

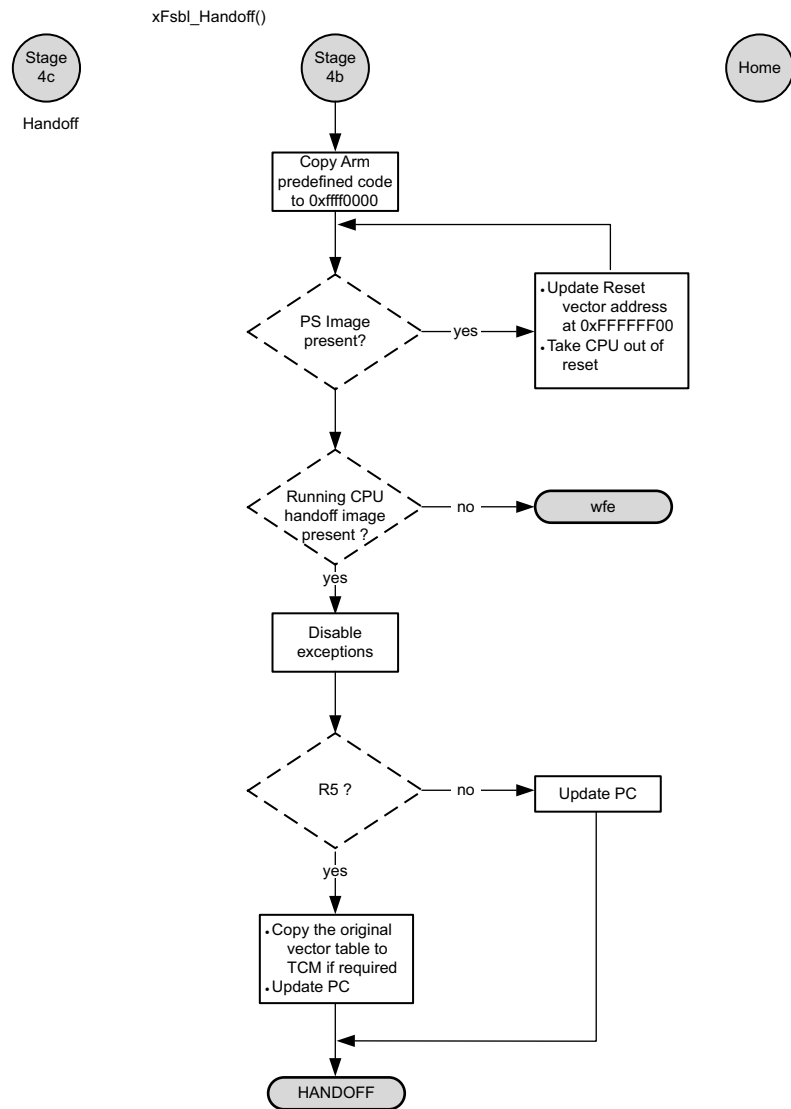


図 7-22: ハンドオフ

サポートされるハンドオフ

表 7-5 に、FSBL でサポートされるハンドオフの組み合わせを示します。

表 7-5: サポートされるハンドオフ

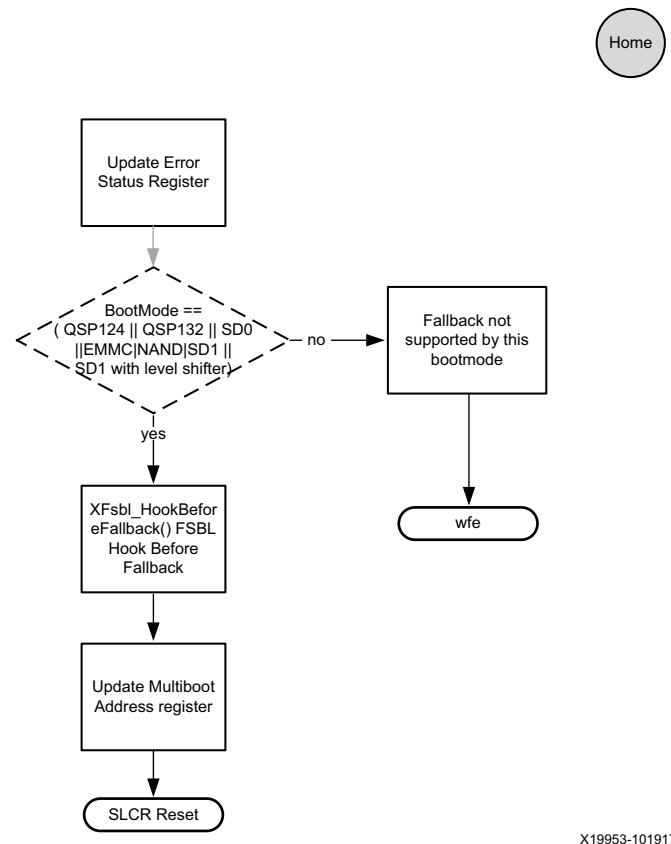
FSBL	アプリケーション	プロセッサ コア	実行アドレス
64 ビット	64 ビット	すべて (A53-0、A53-1、A53-2、A53-3)	任意のアドレス
64 ビット	32 ビット	A53-1、A53-2、A53-3	0x0
32 ビット	32 ビット	A53-0	任意のアドレス
32 ビット	32 ビット	A53-1、A53-2、A53-3	0x0
32 ビット	64 ビット	A53-1、A53-2、A53-3	任意のアドレス

エラー ロックダウン

XFsb1_ErrorLockDown 関数は、FSBL 内のエラーを処理します。関数の戻り値がエラーの発生を示している場合、この関数が常に呼び出されます。この関数はエラー ステータス レジスタを更新し、フォールバックがサポートされていない場合は無限ループに入ります。

ブート モードがフォールバックをサポートしている場合は、マルチブート オフセット レジスタを更新し、WDT リセットを待ちます。リブートされると、bootROM および FSBL はマルチブート オフセットに基づいて計算した新しいアドレスからイメージを読み出してロードします。

XFsb1_ErrorLockDown()



X19953-101917

図 7-23: エラー ロックダウン関数

その他の関数

FSBL では、次の関数を利用できます。

- [XFsbl_PrintArray](#)
- [XFsbl_Strcpy](#)
- [XFsbl_Streat](#)
- [XFsbl_Stremp](#)
- [XFsbl_MemCpy](#)
- [XFsbl_PowerUpIsland](#)
- [XFsbl_IsolationRestore](#)
- [XFsbl_SetTlbAttributes](#)
- [XFsbl_GetSiliconIdName](#)
- [XFsbl_GetProcEng](#)
- [XFsbl_CheckSupportedCpu](#)
- [XFsbl_AdmaCopy](#)
- [XFsbl_GetDrvNumSD](#)
- [XFsbl_MakeSdFileName](#)

XFsbl_PrintArray

DebugType での指定に従い、配列全体をバイト単位で表示します。

```
void XFsbl_PrintArray (u32 DebugType, const u8 Buf[], u32 Len, const char *Str);
```

表 7-6: XFsbl_PrintArray の FSBL のパラメーター

パラメーター	説明
DebugType	ここで定義したデバッグ タイプに従って配列が表示されます。
Buf	表示するバッファーへのポインター
Len	表示するバイトの長さ
Str	表示するデータへのポインター

XFsbl_Strcpy

ソース文字列をデスティネーション文字列へコピーします。

```
char *XFsbl_Strcpy(char *DestPtr, const char *SrcPtr)
```

表 7-7: XFsbl_Strcpy の FSBL のパラメーター

パラメーター	説明
DestPtr	表示するバッファーへのポインター
SrcPtr	ソース文字列を含むバッファーへのポインター

XFsbI_Strcat

1 番目の文字列に 2 番目の文字列を追加します。

```
char* XFsbI_Strcat(char* Str1Ptr, const char* Str2Ptr)
```

表 7-8: XFsbI_Strcat の FSBL のパラメーター

パラメーター	説明
Str1Ptr	1 番目の文字列。この文字列に、Str2Ptr が指し示す文字列が追加されます。
Str2Ptr	2 番目の文字列へのポインター

XFsbI_Strcmp

文字列を比較します。

```
s32 XFsbI_Strcmp( const char* Str1Ptr, const char* Str2Ptr)
```

表 7-9: XFsbI_Strcmp の FSBL のパラメーター

パラメーター	説明
Str1Ptr	1 番目の文字列へのポインター
Str2Ptr	2 番目の文字列へのポインター

XFsbI_MemCpy

SrcPtr が指し示すメモリ内容を、DestPtr が指し示すメモリへコピーします。Len はコピーするバイトの数です。

```
void* XFsbI_MemCpy(void * DestPtr, const void * SrcPtr, u32 Len)
```

表 7-10: XFsbI_MemCpy の FSBL のパラメーター

パラメーター	説明
SrcPtr	コピーするメモリ内容へのポインター
DestPtr	デスティネーションへのポインター
Len	コピーするバイトの長さ

XFsbI_PowerUpIsland

1 つ以上の電源アイランドの電力ステートをチェックし、必要に応じて電源を投入します。

```
u32 XFsbI_PowerUpIsland(u32 PwrIslandMask)
```

表 7-11: XFsbI_PowerUpIsland の FSBL のパラメーター

パラメーター	説明
PwrIslandMask	電源投入が必要なアイランドのマスク

XFsbI_IsolationRestore

PMU ファームウェアを介して分離の回復を要求します。

```
u32 XFsbI_IsolationRestore(u32 IsolationMask);
```

表 7-12: XFsbI_IsolationRestore の FSBL のパラメーター

パラメーター	説明
IsolationMask	分離を回復するエントリのマスク

XFsbI_SetTlbAttributes

変換テーブルのセクションに対するメモリ属性を設定します。

```
void XFsbI_SetTlbAttributes(INTPTR Addr, UINTPTR attrib);
```

表 7-13: XFsbI_SetTlbAttributes の FSBL のパラメーター

パラメーター	説明
Addr	属性を設定するアドレス
Attrib	メモリ領域の属性

XFsbI_GetSiliconIdName

CSU_ID_CODE レジスタを読み出し、デバイスの SvdId を計算します。対応する deviceID 名を返します。

```
const char *XFsbI_GetSiliconIdName(void);
```

XFsbI_GetProcEng

エンジン タイプを判定して返します。現在サポートされているエンジン タイプは、CG、EG、および EV のみです。

```
const char *XFsbI_GetProcEng(void);
```

XFsbI_CheckSupportedCpu

CPU がこのシリコン バリエーションによってサポートされているかどうかを調べます。現在は CG デバイスかどうかをチェックします。A53_2/3 コアへのハンドオフが禁止されています。

```
u32 XFsbI_CheckSupportedCpu(u32 CpuId);
```

表 7-14: XFsbI_CheckSupportedCpu の FSBL のパラメーター

パラメーター	説明
CpuId	プロセッサが A53_2 または A53_3 かどうかを調べます。

XFsbI_AdmaCopy

ADMA を使用してメモリからメモリへデータをコピーします。キャッシュの無効化とフラッシュは、ユーザーが正しく処理する必要があります。この関数を呼び出す前に、ADMA をシンプルな DMA に設定しておく必要があります。

```
u32 XFsbI_AdmaCopy(void * DestPtr, void * SrcPtr, u32 Size);
```

表 7-15: XFsbI_AdmaCopy の FSBL のパラメーター

パラメーター	説明
DestPtr	データのコピー先となるデスティネーション バッファへのポインター
SrcPtr	データのコピー元となるソース バッファへのポインター
Size	コピーするデータのバイト数

XFsbI_GetDrvNumSD

デザインおよびブート モードに基づき、ドライブ番号を取得します。

```
u32 XFsbI_GetDrvNumSD(u32 DeviceFlags);
```

表 7-16: XFsbI_GetDrvNumSD の FSBL のパラメーター

パラメーター	説明
DeviceFlags	ブート モードに関する情報 (SD0、SD1、eMMC、SD1-LS のいずれか) を指定します。

XFsbI_MakeSdFileName

ブート イメージのファイル名を返します。ファイル名は、パラメーターから推論します。

```
void XFsbI_MakeSdFileName(char*XFsbI_SdEmmcFileName, u32 MultiBootReg, u32 DrvNum);
```

表 7-17: XFsbI_MakeSdFileName の FSBL のパラメーター

パラメーター	説明
XFsbI_SdEmmcFileName	最終的なファイル名を格納します。
MultiBootReg	マルチブート レジスタの値が 0 以外の場合、ファイル名に追加されます。
DrvNum	SD0 と SD1 の論理ドライブを区別します。

FSBL のフック

フックとは、ユーザー定義可能な関数です。FSBL にはブランク関数がいくつかあり、これらを戦略的な位置から実行します。次の表に、現在利用可能なフックを示します。

表 7-18: FSBL のフック

フックの目的/位置	フック関数名
PL ビットストリームのロード前	XFsb1_HookBeforeBSDownload()
PL ビットストリームのロード後	XFsb1_HookAfterBSDownload()
いずれかのアプリケーションへの最初のハンドオフ前	XFsb1_HookBeforeHandoff()
フォールバック前	XFsb1_HookBeforeFallback()
psu_init に初期化コードを追加したり、psu_init をカスタム初期化コードで置き換えたりします。	XFsb1_HookPsuInit()

FSBL の詳細は、[FSBL の Wiki ページ \(英語\)](#) を参照してください。

セキュリティ機能

はじめに

この章では、アプリケーションのブート時および実行時のセキュリティ対策として利用できる Zynq® UltraScale+™ MPSoC デバイスの機能について説明します。セキュアブートメカニズムの詳細は、『Zynq UltraScale+ MPSoC テクニカルリファレンスマニュアル』(UG1085) [参照 11] の「セキュリティ」の章を参照してください。

システム保護ユニット (SPU) は、Zynq UltraScale+ MPSoC デバイス上で実行するアプリケーションの実行時のセキュリティを確保するために次のハードウェア機能を提供します。

- [ザイリンクスメモリ保護ユニット \(XMPU\)](#)
- [ザイリンクスペリフェラル保護ユニット \(XPPU\)](#)
- [システムメモリ管理ユニット \(SMMU\)](#)
- [A53 メモリ管理ユニット](#)
- [R5 メモリ保護ユニット](#)

実行時のセキュリティ機能の 1 つに、Linux から PMU および CSU グローバルレジスタへのアクセスを制限する機能があります。これらのレジスタは 2 つのリストに分類されます。1 つはホワイトリスト (デフォルトでいつでもアクセスできるレジスタ) で、もう 1 つはブラックリスト (コンパイル時フラグを設定した場合のみアクセスできるレジスタ) です。詳細は、「[CSU/PMU レジスタへのアクセス](#)」を参照してください。

ブート時のセキュリティ

このセクションでは、認証および暗号化用のさまざまなブートイメージフォーマットについて詳しく説明します。

暗号化

Zynq UltraScale+ MPSoC デバイスにはユーザーブートイメージの機密性をサポートする AES-GCM ハードウェアエンジンがあり、これを使用してブート後にユーザーデータを暗号化/復号化することも可能です。

AES 暗号エンジンは、さまざまなキーソースへアクセスできます。キーソースの詳細は、『Zynq UltraScale+ MPSoC テクニカルリファレンスマニュアル』(UG1085) [参照 11] を参照してください。

イメージを暗号化するにはレッドキーを使用します。ブートファイル (BOOT.bin) の生成時に、レッドキーと初期化ベクター (IV) を .nky ファイル形式で Bootgen に提供する必要があります。

PMU ファームウェアは、CSU bootROM または FSBL によってロードできます。CSU bootROM は FSBL と PMU ファームウェアを別々のパーティションとして扱い、これらを個別に復号化します。FSBL と PMU ファームウェアの両方が暗号化されている場合、規格に違反して AES キー /IV が再利用されます。



重要: FSBL と PMU ファームウェアの両方が暗号化されている場合、AES キー /IV ペアの再利用を防ぐために PMU ファームウェアを CSU bootROM ではなく FSBL でロードする必要があります。詳細は、[ザイリンクス アンサー 70622](#) を参照してください。

次の BIF ファイルは暗号化されたイメージ用で、PMU ファームウェアは FSBL によってロードされます。

```
the_ROM_image:
{
  [aeskeyfile] bbram.nky
  [keysrc_encryption] bbram_red_key
  [bootloader, encryption=aes, destination_cpu=a53-0] ZynqMP_Fsbl.elf
  [destination_cpu = pmu, encryption=aes] pmufw.elf
}
```

BBRAM レッド キーを使用する BIF ファイル

次の BIF ファイルの例は、BBRAM に格納されたレッド キーを示しています。

```
the_ROM_image: {
  [aeskeyfile] bbram.nky
  [keysrc_encryption] bbram_red_key
  [bootloader, encryption=aes, destination_cpu=a53-0] ZynqMP_Fsbl.elf
  [destination_cpu = a53-0, encryption=aes] App_A53_0.elf
}
```

eFUSE レッド キーを使用する BIF ファイル

次の BIF ファイルの例は、eFUSE に格納されたレッド キーを示しています。

```
the_ROM_image: {
  [aeskeyfile] efuse.nky
  [keysrc_encryption] efuse_red_key
  [bootloader, encryption=aes, destination_cpu=a53-0] fsbl.elf
  [destination_cpu = a53-0, encryption=aes] App_A53_0.elf
}
```

操作キーを使用する BIF ファイル

操作キーを使用して Bootgen でブート イメージを生成する場合、ユーザーは操作キーのほかに .nky ファイルのレッド キーと IV をツールに入力する必要があります。Bootgen は、操作キーをヘッダーに配置し、デバイスのレッド キーを使用してこれを暗号化します。その結果、「暗号化された安全なヘッダー」が構築されます。この方法の主なメリットは、デバイス キーの使用を最小限に抑えて攻撃への露出を制限できることです。

詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「AES ブート キーの使用を最小限にとどめる (OP Key オプション)」を参照してください。

```
the_ROM_image:
{
  [aeskeyfile]  bbram.nky
  [fsbl_config] opt_key
  [keysrc_encryption] bbram_red_key
  [bootloader, encryption=aes, destination_cpu=a53-0]  ZynqMP_Fsbl.elf
  [destination_cpu = a53-0, encryption=aes] App_A53_0.elf
}
```

操作キーを使用して開発環境のデバイス キーを保護する

ここでは、秘密のレッド キーを管理するチーム A (セキュア チーム) およびチーム B (非セキュア チーム) の 2 つの開発チームが共同で取り組み、秘密のレッド キーを共有せずに暗号化されたイメージを構築するステップを説明します。チーム A は秘密のレッド キーを管理します。チーム B は、開発およびテスト用に暗号化されたイメージを構築しますが、秘密のレッド キーへはアクセスできません。

チーム A はデバイス キー (Op Key オプションを使用) でブートローダーを暗号化し、それをチーム B に渡します。チーム B は Op Key を使用してその他のパーティションをすべて暗号化します。

チーム B は、Bootgen を使用し、暗号化したパーティションとチーム A から受け取った暗号化されたブートローダーを 1 つの boot.bin にまとめます。

次に、イメージを作成する手順を説明します。

手順 1

まず、チーム A が opt_key を使用してデバイス キーでブートローダーを暗号化し、それをチーム B に渡します。これで、チーム B は操作キーをデバイス キーとして使用し、すべてのパーティションと暗号化されたブートローダーをまとめて完全なイメージを作成できます。

1. ブートローダーをデバイス キーで暗号化します。

```
bootgen -arch zynqmp -image stage1.bif -o fsbl_e.bin -w on -log error
```

stage1.bif の例:

```
stage1:
{
  [aeskeyfile] aes.nky
  [fsbl_config] opt_key
  [keysrc_encryption] bbram_red_key
  [bootloader, destination_cpu=a53-0, encryption=aes] fsbl.elf
}
```


stage1 の aes.nky の例:

```
Device xc7z020clg484;
Key 0 AD00C023E238AC9039EA984D49AA8C819456A98C124AE890ACEF002100128932;
IV 0 F7F8FDE08674A28DC6ED8E37;
Key Opt 229C993D1310DD27B6713749B6D07FCF8D3DCA01EC9C64778CBAF457D613508F;
```

2. 操作キーをデバイス キーとして使用し、暗号化されたブートローダーと残りのパーティションをまとめて完全なイメージを作成します。

```
bootgen -arch zynqmp -image stage2a.bif -o final.bin -w on -log error
```

stage2.bif の例:

```
stage2:
{
  [aeskeyfile] aes-opt.nky
  [bootimage] fsbl_e.bin
  [destination_cpu=a53-0,encryption=aes]hello.elf
  [destination_cpu=a53-1,encryption=aes]hello1.elf
}
```

stage2 の aes-opt.nky の例:

```
Device xc7z020clg484;
Key 0 229C993D1310DD27B6713749B6D07FCF8D3DCA01EC9C64778CBAF457D613508F;
IV 0 F7F8FDE08674A28DC6ED8E37;
```

手順 2

まず、チーム A が `opt_key` オプションを使用してデバイス キーでブートローダーを暗号化し、それをチーム B に渡します。これで、チーム B は、操作キーをデバイス キーとして使用し、パーティションごとに個別に暗号化されたイメージを作成できます。最後に、チーム B は `Bootgen` を使用し、暗号化されたパーティションすべてと暗号化されたブートローダーをまとめて完全なイメージを作成できます。

1. ブートローダーをデバイス キーで暗号化します。

```
bootgen -arch zynqmp -image stage1.bif -o fsbl_e.bin -w on -log error
```

stage1.bif の例:

```
stage1:
{
  [aeskeyfile] aes.nky
  [fsbl_config] opt_key
  [keysrc_encryption] bbram_red_key
  [bootloader,destination_cpu=a53-0,encryption=aes]fsbl.elf
}
```

stage1 の aes.nky の例:

```
Device xc7z020clg484;
Key 0 AD00C023E238AC9039EA984D49AA8C819456A98C124AE890ACEF002100128932;
IV 0 F7F8FDE08674A28DC6ED8E37;
Key Opt 229C993D1310DD27B6713749B6D07FCF8D3DCA01EC9C64778CBAF457D613508F;
```

2. 操作キーをデバイス キーとして使用し、残りのパーティションを暗号化します。

```
bootgen -arch zynqmp -image stage2a.bif -o hello_e.bin -w on -log error
```

stage2a.bif の例:

```
stage2a:
{
  [aeskeyfile] aes-opt.nky
  [destination_cpu=a53-0,encryption=aes]hello.elf
}
bootgen -arch zynqmp -image stage2b.bif -o hello1_e.bin -w on -log error
```

stage2b.bif の例:

```
stage2b:
{
  [aeskeyfile] aes-opt.nky
  [destination_cpu=a53-1,encryption=aes]hello1.elf
}
```

stage2a および stage2b の aes-opt.nky の例:

```
Device xc7z020clg484;
Key 0 229C993D1310DD27B6713749B6D07FCF8D3DCA01EC9C64778CBAF457D613508F;
IV 0 F7F8FDE08674A28DC6ED8E37;
```

3. Bootgen を使用して上記のものをまとめて完全なイメージを作成します。

Use bootgen to stitch the above, to form a complete image.

stage3.bif の例:

```
stage3:
{
  [bootimage]fsbl_e.bin
  [bootimage]hello_e.bin
  [bootimage]hello1_e.bin
}
```

注記: aes.nky の Key Opt は aes-opt.nky の Key 0 と同じものとし、IV 0 は両方の nky ファイルで同じものとする必要があります。

eFUSE に格納されたブラック キーを使用する BIF ファイル

使用しないデバイス キーを暗号化して格納しておく場合には、PUF (Physical Unclonable Function) を利用できます。この場合、実際のレッド キーは PUF で生成された暗号化キー PUF KEK (キー暗号化キー) を使用して暗号化されます。デバイスはブラック キーを復号化して実際のレッド キーを取得します。したがって、ユーザーは必要な入力を Bootgen に与える必要があります。ブラック キーは eFUSE またはブート ヘッダーのいずれかに格納できます。shutter の値は、PUF でオシレーターの値を取り込む時間の長さを示します。この値は、常に 0x100005E とする必要があります。

詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「暗号化(ブラック) キーの格納」を参照してください。

次に示す例では、eFUSE に格納されたブラック キーを示しています。

```
the_ROM_image:
{
  [pskfile] PSK.pem
  [sskfile] SSK.pem
  [aeskeyfile] red.nky
  [keysrc_encryption] efuse_blk_key
  [fsbl_config] shutter=0x0100005E
  [auth_params] ppk_select=0
  [bootloader, encryption = aes, authentication = rsa,
  destination_cpu=a53-0] fsbl.elf
  [bh_key_iv] black_key_iv.txt
}
```

ブート ヘッダーに格納されたブラック キーを使用する BIF ファイル

次の BIF ファイルの例では、ブート ヘッダーに格納されたブラック キーの暗号化を示しています。

```
the_ROM_image:
{
  [aeskeyfile] redkey.nky
  [keysrc_encryption] bh_blk_key
  [bh_keyfile] blackkey.txt
  [bh_key_iv] black_key_iv.txt
  [fsbl_config] pufhd_bh , puf4kmode , shutter=0x0100005E, bh_auth_enable
  [pskfile] PSK.pem
  [sskfile] SSK.pem
  [bootloader, authentication=rsa , encryption=aes, destination_cpu=a53-0] fsbl.elf
  [puf_file] hlprdata4k.txt
}
```

注記: ブラック キーの暗号化を使用する場合は、ブート イメージの認証が必須です。

ブラック キーを使用して eFUSE を生成またはプログラムする方法は、[付録 J「XilSKey Library v6.7」](#) の「Zynq eFUSE PS API」を参照してください。

eFUSE に格納された難読化 (グレー) キーを使用する BIF ファイル

デバイス キーを難読化して格納しておく場合は、実際のレッド キーをファミリ キー (暗号化キーの一種) で暗号化できます。デバイスは難読化キーを復号化して実際のレッド キーを取得します。したがって、ユーザーは必要な入力を Bootgen に与える必要があります。難読化したキーは、eFUSE またはブート ヘッダーのいずれかに格納できます。

詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「難読化 (グレー) キーの格納」を参照してください。

注記: ファミリ キーは、同じ Zynq MPSoC ファミリのデバイスならすべて共通です。この方法を使用すると、実際のユーザー キーを秘密にしたまま委託製造業者に難読化キーを配布できます。

次に示す例は、eFUSE に格納された難読化キーを示しています。

```
the_ROM_image:
{
  [aeskeyfile] red.nky
  [keysrc_encryption] efuse_gry_key
  [bh_key_iv] bhkeyiv.txt
  [bootloader, encryption=aes, destination_cpu=a53-0] fsbl.elf
}
```

次に示す例は、ブート ヘッダーに格納された難読化 (グレー) キーを示しています。

the_ROM_image:

```
{
    [aeskeyfile] red.nky
    [keysrc_encryption] bh_gry_key
    [bh_key_iv] bhkeyiv.txt
    [bh_keyfile] bhkey.txt
    [bootloader, encryption=aes, destination_cpu=a53-0] fsbl.elf
}
```

ファミリ キーを使用して難読化キーを生成する方法

難読化キーは、ザイリンクス ツール (Bootgen) を使用して生成します。ただし、ファミリ キーはザイリンクス開発 ツールには付属しません。これらは別途配布されます。ザイリンクスから提供されたファミリ キーは、次の例に示すように BIF ファイルに含めます。



重要: ファミリ キーの提供を受ける方法は、secure.solutions.dirxilinx.com にお問い合わせください。

難読化キーを生成するための BIF の例:

```
all:
{
    [aeskeyfile] aes.nky
    [familykey] familyKey.cfg
    [bh_key_iv] bhiv.txt
}
```

Bootgen を使用したキーの生成

Bootgen を使用してキーを生成する場合、NIST 認証済みの KDF を使用します (カウンター モードの KDF、PRF として CMAC を使用)。

1 つのキー /IV ペアを使用する場合:

- シードが指定されている場合、シードに基づいてキーを生成します。
- シードが指定されていない場合、Key0 に基づいてキーを生成します。

空のファイルを指定した場合、Bootgen は時間ベースのランダム化を使用してシードを生成します。これは KDF のような標準ではありません。このシードを KDF に入力して、キー /IV ペアを生成します。

複数の AESKEY ファイルを使用した BIF ファイル

次の BIF ファイルの例は、aeskey ファイルを使用した暗号化を示しています。

1 パーティションにつき 1 つの AES キー

複数の nky ファイル (イメージ内の各パーティションに対して 1 つずつ) を指定します。パーティションは、各パーティションの前の行で指定したキーを使用して暗号化されます。

```
sample_bif:
{
    [aeskeyfile] test1.nky
    [bootloader, encryption=aes] fsbl.elf
    [aeskeyfile] test2.nky
    [encryption=aes] hello.elf
    [aeskeyfile] test3.nky
    [encryption=aes] app.elf
}
```

fsbl.elf パーティションは、test1.nky ファイルのキーで暗号化されます。hello.elf に 2 つのロード可能なセクションがあり、2 つのパーティションが存在するとした場合、これらのパーティションは両方とも test2.nky ファイルのキーで暗号化されます。app.elf パーティションは、test3.nky ファイルのキーで暗号化されます。

1 パーティションにつき 1 つの AES キー (ロード可能なセクションが複数ある場合)

複数の nky ファイル (イメージ内の各パーティションに対して 1 つずつ) を指定します。パーティションは、各パーティションの前の行で指定したキーを使用して暗号化されます。ロード可能なセクションが複数あるために複数のパーティションが作成される場合、元のパーティションのキーファイルと同じディレクトリに、ファイル名末尾に「.1」、「.2」、... 「.n」を付けたキーファイルを置くことにより、各パーティションに対して別々のキーファイルを使用できます。

```
sample_bif:
{
    [aeskeyfile] test1.nky
    [bootloader, encryption=aes] fsbl.elf
    [aeskeyfile] test2.nky
    [encryption=aes] hello.elf
    [aeskeyfile] test3.nky
    [encryption=aes] app.elf
}
```

fsbl.elf パーティションは、test1.nky ファイルのキーで暗号化されます。hello.elf ファイルに 3 つのロード可能なセクションがあり、3 つのパーティションが存在するとした場合、hello.elf.0 が test2.nky ファイルからのキーで暗号化されるなら、hello.elf.1 は test2.1.nky ファイルからのキーで暗号化され、hello.elf.2 は test2.2.nky ファイルからのキーで暗号化されます。app.elf パーティションは、test3.nky ファイルのキーで暗号化されます。

同じ .nky を複数のパーティションで使用すると、各パーティションで AES キーと AES キー /IV ペアを再利用することになります。同じ AES キーを複数のパーティションに使用することにより、キーが露出する可能性が高まり、セキュリティ脆弱性となることがあります。複数のパーティションに同じ AES キー /IV ペアを使用することは、規格に違反しています。AES キー /IV ペアの再利用を避けるため、Bootgen は IV をパーティション番号だけインクリメントします。AES キーと AES キー /IV ペアの再利用を避けるため、Bootgen は複数の .nky ファイルに対応して、各パーティションに 1 つずつ使用できます。



重要: キーの再利用を避けるため、今後 1 つの .nky ファイルを複数のパーティションに使用することはできなくなる予定です。



注意: 複数のパーティションに 1 つの .nky ファイルを使用すると、各パーティションで同じキーが使用されるため、セキュリティ脆弱性となることがあります。現在のリリースでは警告が示されますが、将来のリリースではエラーとなる予定です。

注記: Key0/IV0 は、すべての nky ファイルで同じにする必要があります。

複数のキーを指定して、キーの数が暗号化されるブロック数より少ない場合、エラーとなります。

複数のキー /IV ペアを指定する場合は、ペアの数を (ブロック数 + 1) とする必要があります。

余分のキー /IV ペアは SH 用です。たとえば、ブロックの数が 4 の場合、4+1=5 個のキー /IV ペアを指定する必要があります。

認証

Zynq UltraScale+ MPSoC が備える SHA ハードウェア アクセラレータは、SHA-3 アルゴリズムを実装しており、384 ビットのダイジェストを生成します。これを RSA アクセラレータと共に使用することでイメージの認証が可能になり、イメージの復号化には AES-GCM を使用します。これらのブロック (SHA-3/384、RSA、および AES-GCM) は、ハード マクロとして提供されており、暗号インターフェイス ブロック (CIB) の一部となっています。

認証フローでは FSBL を生データとして扱います。イメージが暗号化されているかどうかは無視されます。キーにはプライマリ キー (PK) とセカンダリ キー (SK) の 2 つのレベルがあります。

各キーには、秘密キーと公開キーの 2 つの補足要素があります。

- PK にはプライマリ公開キー (PPK) とプライマリ秘密キー (PSK) があります。
- SK にはセカンダリ公開キー (SPK) とセカンダリ秘密キー (SSK) があります。

CIB 内のハード RSA ブロックは、RSA で求められる大規模な計算を高速化するためのモンゴメリ乗算器です。ハードウェア アクセラレータはシグネチャ生成や検証に使用できます。ROM コードは、シグネチャ生成のみをサポートします。秘密キーは、証明を生成する場合のシグネチャ生成ステージでのみ使用されます。



重要: シグネチャ生成はデバイスで実行されるのではなく、ブート イメージの準備中にソフトウェアで実行されます。

認証証明のフォーマットの詳細は、『Bootgen ユーザー ガイド』(UG1283) [参照 23] を参照してください。

PPK と SPK の 2 つのキーでパーティションを認証します。

PSK と SSK を使用してパーティションを符号化します。

各シグネチャ (SPK、ブート ヘッダー、ブート イメージ) の式を次に示します。

- SPK シグネチャ。次の計算によって 512 バイトの SPK シグネチャが生成されます。
`SPK signature = RSA(PSK, padding || SHA(SPK+ auth_header)).`
- ブート ヘッダー シグネチャ。次の計算によって 512 バイトのブート ヘッダー シグネチャが生成されます。
`Boot header signature = RSA(SSK, padding || SHA(boot header)).`

- ブート イメージシグネチャ。次の計算によって 512 バイトのブート イメージシグネチャが生成されます。

$$BI\ signature = RSA(SSK, padding || SHA(PFW + FSBL + authentication\ certificate)).$$

注記: SHA-3 認証では、ブート ヘッダー、PPK、およびブート イメージのハッシュ値の計算に Keccak SHA3 が常に使用されます。ROM によってロードされないその他すべてのパーティションには、NIST-SHA3 が使用されます。

Bootgen は RSA シグネチャの生成のみをサポートします。係数、累乗、および事前計算された $R^2 \text{ Mod } N$ が必要になります。

ソフトウェアは RSA 暗号の公開キー暗号化でのみサポートされます。シグネチャ RSA エンジンの暗号化には、係数、累乗、および事前計算された $R^2 \text{ Mod } N$ が必要です。これらはすべてキーから抽出されます。

SHA-3 ブート ヘッダー認証と PPK0 を使用する BIF ファイル

次の BIF ファイルの例は、BH RSA オプションをサポートしています。このオプションは、システムがフィールド展開される前の統合およびテストをサポートします。詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「統合およびテストのサポート (BH RSA オプション)」を参照してください。

BIF ファイルは SHA-3 ブート ヘッダー認証用であり、実際の PPK ハッシュ値は eFUSE に格納された値と比較されません。

```
the_ROM_image: {
  [fsbl_config] bh_auth_enable
  [auth_params] ppk_select=0; spk_id=0x00000000
  [pskfile] primary_4096.pem
  [sskfile] secondary_4096.pem
  [bootloader, authentication=rsa, destination_cpu=a53-0] fsbl.elf
  [pmufw_image, authentication=rsa] xpfw.elf
}
```

SHA-3 eFUSE RSA 認証と PPK0 を使用する BIF ファイル

次の BIF ファイルの例は、PPK0 と SHA-3 を使用する eFUSE RSA 認証を示しています。

```
the_ROM_image:
{
  [auth_params]ppk_select=0;spk_id=0x584C4E58
  [pskfile]psk.pem
  [sskfile]ssk.pem
  [bootloader, authentication = rsa, destination_cpu=a53-0]zynqmp_fsbl.elf
  [destination_cpu = a53-0, authentication = rsa]Application.elf
}
```

RSA キー取り消しサポートの改善

RSA キーを使用すると、すべてのパーティションのセカンダリ キーを取り消すことなく、1 つのパーティションのセカンダリ キーを取り消すことができます。

注記: プライマリ キーは、すべてのパーティションで同じものである必要があります。

これは、USER_FUSE0 ~ USER_FUSE7 の eFUSE (すべてのキーを使用する必要がない場合は、最大で 256 個のキーを取り消すことが可能) と、新しい BIF パラメーターの `spk_select` によって可能になっています。

次の BIF ファイルの例は、改善されたユーザー eFUSE による取り消しを示しています。

次に示す `bif` 入力を使用すると、イメージ ヘッダーと FSBL は認証に異なる SSK (それぞれ `ssk1.pem` と `ssk2.pem`) を使用します。

the_ROM_image:

```
{
  [auth_params]ppk_select = 0
  [pskfile]psk.pem
  [sskfile]ssk1.pem
  [bootloader, authentication = rsa, spk_select = spk-efuse, spk_id = 0x12345678,
  sskfile = ssk2.pem]zynqmp_fsbl.elf
  [destination_cpu =a53-0, authentication = rsa, spk_select = user-efuse, spk_id = 200,
  sskfile = ssk3.pem]Application1.elf
  [destination_cpu =a53-0, authentication = rsa, spk_select = spk-efuse, spk_id =
  0x12345678, sskfile = ssk4.pem]Application2.elf
}
```

別々の SSK を指定しない場合、イメージ ヘッダーと FSBL の両方に同じ SSK (`ssk2.pem`) が使用されます。

the_ROM_image:

```
{
  [auth_params]ppk_select = 0
  [pskfile]psk.pem
  [bootloader, authentication = rsa, spk_select = spk-efuse, spk_id = 0x12345678,
  sskfile = ssk2.pem]zynqmp_fsbl.elf
  [destination_cpu =a53-0, authentication = rsa, spk_select = user-efuse, spk_id = 200,
  sskfile = ssk3.pem]Application1.elf
  [destination_cpu =a53-0, authentication = rsa, spk_select = spk-efuse, spk_id =
  0x12345678, sskfile = ssk4.pem]Application2.elf
}
```

`spk_select = spk-efuse` は、そのパーティションに `spk_id` eFuse が使用されることを示しています。

`spk_select = user-efuse` は、そのパーティションにユーザー eFUSE が使用されることを示しています。CSU ROM によってロードされるパーティションは、常に `spk_efuse` を使用します。

注記: `spk_id` eFuse は、どのキーが有効であるかを指定します。したがって、ROM は `spk_id` eFuse のフィールド全体を SPK ID と照合してビットが一致することを確認します。

ユーザー eFUSE は、どのキー ID が有効でない (取り消された) かを指定します。したがって、ファームウェア (ROM 以外) は、SPK ID を表すそのユーザー eFUSE がプログラムされているかどうかをチェックします。

外部メモリを使用するビットストリーム認証

ビットストリームの認証は、その他のパーティションとは異なります。FSBL は、OCM 内にすべて含めることが可能であるため、デバイス内部で認証して復号化できます。ビットストリームの場合は、ファイルサイズが非常に大きいためデバイス内にすべてを含めることができず、外部メモリを使用する必要があります。外部メモリを使用すると、攻撃者がこの外部メモリに侵入する可能性があるため、セキュリティ管理という課題が生じます。

次のセクションでは、外部メモリを使用する場合にビットストリームを安全に認証する方法について説明します。

Bootgen

ビットストリームの認証が要求されると、Bootgen はビットストリーム全体を 8MB ブロックに分割し、各ブロックに認証証明が与えられます。

ビットストリームが 8MB の倍数でない場合は、最後のブロックに残りのビットストリーム データが含まれます。

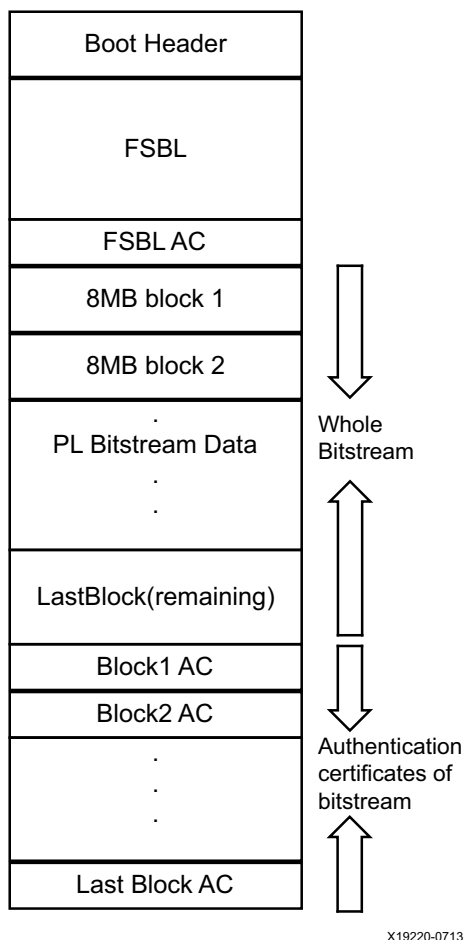


図 8-1: ビットストリームブロック

認証と暗号化の両方が有効の場合は、最初にビットストリームの暗号化が実行されます。その後、Bootgen は暗号化されたデータをブロックに分割して、各ブロックに認証証明書を提供します。

ソフトウェア

ビットストリーム パーティションを安全に認証するために、FSBL は ATF セクションの OCM メモリを使用して、フラッシュまたは DDR からビットストリームをチャンクとしてコピーします。したがって、ブート イメージの作成中、ビットストリーム パーティションは ATF パーティションより前に配置される必要があります。ATF パーティションより後に配置されると、ビットストリーム パーティションが処理されるときに ATF メモリが上書きされてしまいます。

DDR を使用するシステムも使用しないシステムも、このワークフローはほぼ同じです。唯一の違いは、DDR を使用するシステムでは FSBL がすべてのビットストリーム パーティション (ビットストリームと認証証明) をフラッシュデバイスから DDR にコピーするという点です (DDR の方が高速アクセスが可能のため)。その後、FSBL は毎回 DDR からビットストリームのチャンクをコピーします。DDR をサポートしないシステムでは、FSBL はビットストリームのチャンクをフラッシュ デバイスから直接コピーします。

ビットストリーム認証のソフトウェア ワークフローは次のとおりです。

1. FSBL が XFSBL_PS_DDR マクロに基づき、システムで DDR が利用できるかどうかを特定します。FSBL は OCM 内に 2 つのバッファを持ちます。1 つはバッファ サイズ 56KB の ReadBuffer で、もう 1 つは、8MB ブロックの各 56KB 用に計算された中間ハッシュを格納する HashsOfChunks[] です。
 2. FSBL が最初の 8MB ブロックから 56KB チャンク データを ReadBuffer へコピーします。
 3. FSBL が 56KB のハッシュ値を計算して HashsOfChunks に格納します。
 4. 8MB ブロックすべてが完了するまで、FSBL が前の手順を繰り返します。
- 注記:** 性能を考慮して 56KB を使用しますが、実際は任意のサイズを選択可能です。
5. FSBL がビットストリームを認証します。
 6. 認証に成功すると、FSBL が DDR/フラッシュの最初のブロックから ReadBuffer へ 56KB データのコピーを開始し、ハッシュ値を計算して、HashsOfChunks に格納されているハッシュ値と比較します。
 7. ハッシュ値が一致すると、FSBL が DMA (ビットストリームが暗号化されていない場合) または AES (暗号化が有効の場合) を使用してデータを PCAP へ送信します。
 8. 8MB ブロックすべてが完了するまで FSBL が前の 2 つの手順を繰り返します。
 9. ビットストリームのすべてのブロックに対して一連のプロセスを繰り返します。

注記: いかなる段階でもエラーが生じた場合には、PL がリセットされて FSBL は終了します。

セキュア ストリーム スイッチを設定することで、ビットストリームは CSU DMA 経由で PCAP へ直接転送されます。

DDR システムの場合、暗号化されたすべてのビットストリームは DDR へコピーされます。DDR をサポートしていないシステムでは、復号化されたビットストリームは OCM (ATF セクション) にチャンクとしてコピーされます。

注記: ビットストリーム パーティションは、ブート イメージ内で FSBL パーティションの直後に配置することを推奨します。

実行時セキュリティ

実行時セキュリティとは、正しくプログラムされていないデバイス、または悪意のあるデバイスによるシステムメモリの破損やシステム障害からシステムを保護することをいいます。

システムを保護するには、ソフトウェア実行中にメモリとペリフェラルを保護することが重要です。Zynq UltraScale+ MPSoC デバイスには、メモリとペリフェラルを保護する目的で次のブロックが用意されています。

- 。 [Arm トラステッド ファームウェア](#)
- 。 [ザイリンクス メモリ保護ユニット](#)
- 。 [ザイリンクス ペリフェラル保護ユニット](#)
- 。 [システム メモリ管理ユニット](#)
- 。 [A53 メモリ管理ユニット](#)
- 。 [R5 メモリ保護ユニット](#)

実行時のセキュリティ機能の1つに、Linux から PMU および CSU グローバル レジスタへのアクセスを制限する機能があります。これらのレジスタは2つのリストに分類されます。

- 。 デフォルトでいつでもアクセスできるレジスタ (ホワイト リスト レジスタ)。
- 。 コンパイル時フラグを設定した場合のみアクセスできるレジスタ (ブラック リスト レジスタ)。

詳細は、「[CSU/PMU レジスタへのアクセス](#)」を参照してください。

Arm トラステッド ファームウェア

Zynq UltraScale+ MPSoC デバイスには、Arm v8 コア用の標準実行モデルが組み込まれています。このモデルは、低いレベルの権限で通常のオペレーティングシステムを実行するため、セキュリティを重視するハードウェアまたはレジスタへのアクセスを要求する場合は、セキュア モニター コール (SMC) と呼ばれるプロキシ ソフトウェアを使用する必要があります。Linaro Arm トラステッド ファームウェア (ATF) には、ザイリンクスの Zynq UltraScale+ MPSoC デバイス用 SMC が含まれています。これまで説明したように、セキュア ブート機能は CSU と PMU によって実現されるため、ザイリンクスはトラステッド OS を必要とせず、提供もしていません。ただし、ザイリンクスが提供する ATF には、ユーザー独自のトラステッド OS を追加するためのフックが用意されており、トラステッド アプリケーションを追加で組み込むことができます。ATF には、セキュア ワールドと非セキュア ワールドを切り替えるためのセキュア モニターがあります。

ATF の主な目的は、絶対に必要な場合を除き、システム モジュール (ドライバー、アプリケーション) がリソースにアクセスしないようにすることにあります。たとえば、SoC 内の公開キーが格納されている領域に Linux からアクセスすることは禁止する必要があります。同様に、暗号ブロックのドライバーは使用中のセッション キーを識別する必要があります。つまり、キー ネゴシエーション アルゴリズムでセッション キーをプログラムして、暗号ブロック内の安全な場所に格納することも可能です。

PSCI は、非セキュア ソフトウェアから電力管理ユース ケース (セカンダリ CPU ブート、ホットプラグ、アイドルなど) を実装したファームウェアへのインターフェイスです。

例外レベルで動作する監視システムでは、コンテキストの復元やコアの電源ステート切り替えといったアクションが必要になる場合があります。

非セキュア ソフトウェアは、Arm セキュア モニター コール (SMC) 命令を使用して ATF のランタイム サービスにアクセスできます。

Arm アーキテクチャでは、SMC 命令によって生成される SMC 例外を利用した非セキュア ステートからセキュア ステートへの同期制御転送は、セキュア モニターで処理されます。セキュア モニターの動作は、レジスタで渡されるパラメーターで決定します。

次の 2 種類の呼び出しが定義されています。

- 。 不可分なセキュア動作を実行する高速呼び出し
- 。 プリエンプティブなセキュア動作を開始する標準呼び出し

SMC 命令の 2 つの呼び出し規約によって SMC 命令の 2 つの関数識別子を定義します。

- 。 SMC32: 32 ビットおよび 64 ビット クライアント コードのどちらも使用できる 32 ビット インターフェイス。最大 6 つの 32 ビット引数を渡します。
- 。 SMC64: 64 ビット クライアント コードのみが使用する 64 ビット インターフェイス。最大 6 つの 64 ビット引数を渡します。

SMC 関数識別子は、ユーザーが呼び出し規約に基づいて定義します。SMC 関数識別子を定義して、それをすべての SMC 呼び出しのレジスタ R0 または W0 に渡すと、次のものが決まります。

- 。 呼び出しタイプ
- 。 呼び出し規約
- 。 呼び出すセキュア関数

ATF は、いずれかのセキュリティ ステートで生成された割り込みを設定および管理するためのフレームワークを実装しています。実装されているのは、Arm リファレンス プラットフォームの TBBR (Trusted Board Boot Requirements) および PDD (Platform Design Document) の一部です。

コールド ブート パスはプラットフォームをパワーアップして TBBR シーケンスが開始する場所で、このシーケンスは DRAM 内の非セキュア ワールドで動作するファームウェアに制御を渡すまで動作を継続します。コールド ブート パスは、プラットフォームに物理的に電源が供給された時点から開始します。

- 。 リセットから解放された CPU のうち 1 つがプライマリ CPU として選択され、その他の CPU はセカンダリ CPU と見なされます。
- 。 プライマリ CPU の選択方法はプラットフォームにより異なります。コールド ブート パスは主にプライマリ CPU によって実行されますが、必須の CPU 初期化はすべての CPU によって実行されます。
- 。 プライマリ CPU がセカンダリ CPU のブートに必要な初期化を十分に実行するまで、セカンダリ CPU はプラットフォーム固有の安全なステートに保持されます。

ウォーム ブートの場合、CPU はリセットから解放されたときと同じプロセッサ モードでプラットフォーム固有のアドレスへジャンプします。

表 8-1 に、ATF 関数の一覧を示します。

表 8-1: ATF 関数

ATF 関数	説明
<code>bl31_arch_setup();</code>	EL3 からの全般的なアーキテクチャセットアップ。
<code>bl31_platform_setup();</code>	BL1 でのプラットフォーム セットアップ。
<code>bl31_lib_init();</code>	すべての BL31 ヘルパー ライブラリを初期化するシンプルな関数。
<code>cm_init();</code>	コンテキスト管理ライブラリ初期化ルーチン。
<code>dcsw_op_all(DCCSW);</code>	再び非セキュア ソフトウェア ワールドへ入る前にキャッシュをクリーニングします。
<code>(*bl32_init)();</code>	BL32 イメージの初期化に使用する関数ポインター。
<code>runtime_svc_init();</code>	ランタイム サービスによってエクスポートされるディスクリプター内の初期化ルーチンを呼び出します。ディスクリプターの検査が完了したら、ディスクリプターの開始および終了所有エンティティ番号と呼び出しタイプを結合して一意の oen を生成します。この一意の oen を <code>rt_svc_descs_indices</code> 配列へのインデックスとして使用します。ランタイム サービス ディスクリプターのインデックスは、このインデックスに格納されます。
<code>validate_rt_svc_desc();</code>	ランタイム サービス ディスクリプターを使用する前にサニティ チェックするシンプルなルーチンです。
<code>get_unique_oen();</code>	一意の oen を取得します。
<code>bl31_prepare_next_image_entry();</code>	EL3 レジスタをプログラムし、次の ERET で BL31 の次のイメージに入るために必要なセットアップを実行します。
<code>bl31_get_next_image_type();</code>	<code>next_image_type</code> を返します。
<code>bl31_plat_get_next_image_ep_info(image_type);</code>	指定したセキュリティ ステートで動作しているイメージに対応する <code>entry_point_info</code> 構造体への参照を返します。
<code>get_security_state ()</code>	セキュリティ ステートを取得します。
<code>cm_init_context()</code>	現在の CPU によって最初に使用される <code>cpu_context</code> を初期化し、 <code>entry_point_info</code> 構造体で指定された初期エントリ ポイント ステートを設定します。
<code>cm_get_context_by_mpidr()</code>	指定したセキュリティ ステートのコンテキストとして設定した、MPIDR で特定される CPU の最新の <code>cpu_context</code> 構造体へのポインターを返します。そのような構造体が指定されていない場合は NULL が返されます。
<code>get_scr_el3_from_routing_model()</code>	前回の <code>set_routing_model()</code> の呼び出しで保存されたセキュリティ ステートのルーティング モデル (IRQ および FIQ ビットで表現) を含む <code>SCR_EL3</code> のキャッシュ コピーを返します。

表 8-1: ATF 関数 (続き)

ATF 関数	説明
<code>get_el3state_ctx()</code>	ERET が正しいエントリにジャンプするように EL3 ステートに値を入力します。
<code>get_gpregs_ctx()</code>	コンテキストへのエントリ ポイントから X0-X7 の値を保存します。
<code>cm_prepare_el3_exit()</code>	<p>セキュアまたは非セキュア ソフトウェア ワールドへの最初のエントリに対する CPU システム レジスタを準備します。</p> <ul style="list-style-type: none"> EL2 または hyp モードに対して実行が要求される場合、SCTLR_EL2 が初期化されます。 非セキュア EL1 または svc モードに対して実行が要求される場合、CPU は EL2 をサポートします。次に、必要なすべての EL2 レジスタを設定して EL2 を無効にします。 <p>すべてのエントリに関して、EL1 レジスタは <code>cpu_context</code> から初期化されます。</p>
<code>cm_get_context(security_state);</code>	セキュリティ ステートのコンテキストを取得します。
<code>el1_sysregs_context_restore</code>	システム レジスタのコンテキストを復元します。
<code>cm_set_next_context</code>	例外リターンに使用するコンテキストをプログラムします。これにより、SP_EL3 は必要なセキュリティ ステートに設定された <code>cpu_context</code> へのポインターに初期化されます。
<code>bl31_late_platform_setup();</code>	プラットフォームをセットアップします。
<code>bl31_register_bl32_init</code>	BL32 init 関数へのポインターを初期化します。
<code>bl31_set_next_image_type</code>	ランタイム サービスが BL31 の後に実行するイメージを決定するのに使用するアクセサ関数。

ATF に関する詳細は、『Arm トラステッド ファームウェア』[参照 40] を参照してください。

FPGA マネージャー ソリューション

Zynq UltraScale+ MPSoC の FPGA マネージャーは、実行中に Linux 環境から各種ビットストリーム (フル、パーシャル、認証済み、暗号化済みなど) をダウンロードするためのインターフェイスとなります。FPGA マネージャーの主な機能は次のとおりです。

- フルビットストリームの読み込み
- パーシャル リコンフィギュレーション (パーシャル ビットストリームの読み込み)
- 暗号化されたフル/パーシャルビットストリームの読み込み
- 認証済みフル/パーシャルビットストリームの読み込み
- 暗号化された認証済みフル/パーシャルビットストリームの読み込み
- コンフィギュレーションレジスタのリードバック
- ビットストリーム (コンフィギュレーション データ) のリードバック

FPGA マネージャーのアーキテクチャ

次の図に、FPGA マネージャーのアーキテクチャを示します。

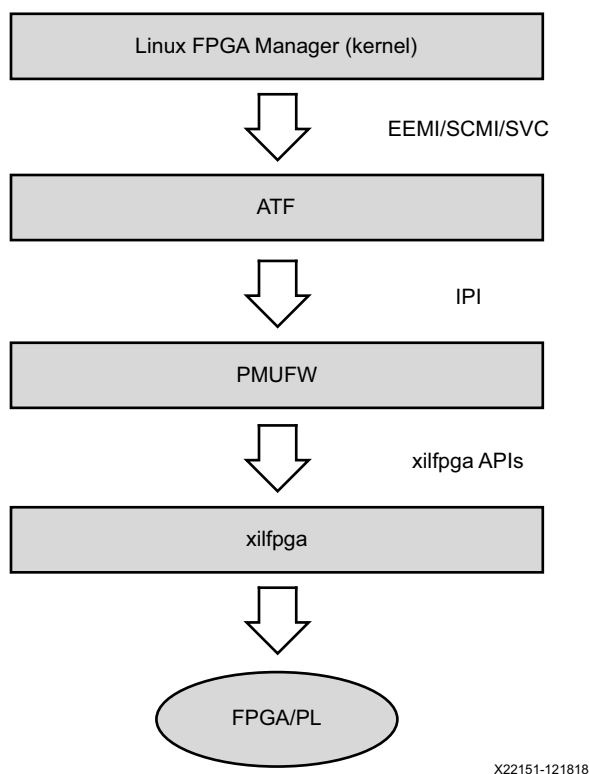
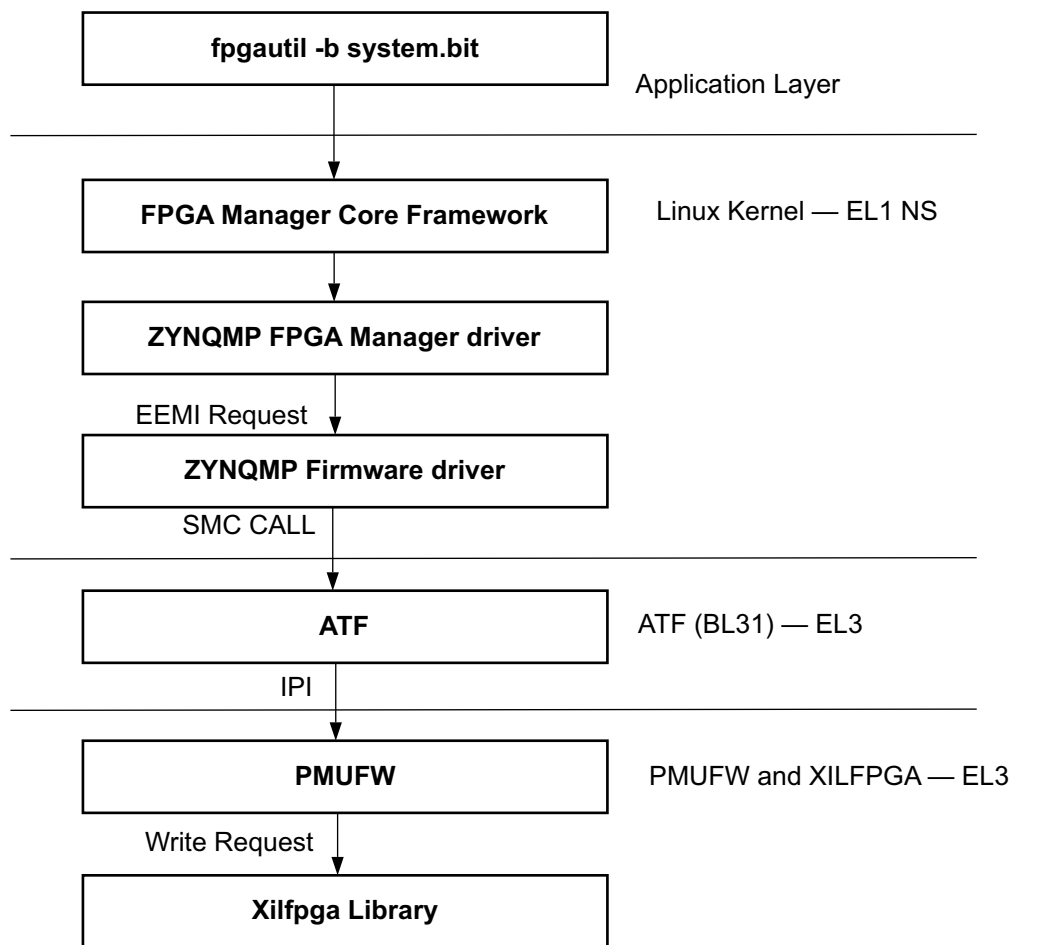


図 8-2: FPGA マネージャーのアーキテクチャ ブロック図

実行フロー

FPGA マネージャーは、ユーザーが Linux を使用してビットストリームを読み込む際の抽象化機能を提供します。xilfpga ライブラリは PCAP、CSUDMA およびその他のハードウェアを初期化します。xilfpga の詳細は、[付録 L「XilFPGA Library v5.0」](#) を参照してください。

ビットストリームを読み込むには、まず FPGA マネージャーが必要なメモリを割り当て、FPGA LOAD API ID を使用して EEMI API を呼び出します。この要求はブロッキング コールです。FPGA マネージャーは ATF からの応答を待ちます。応答は FPGA コア レイヤーに返され、そこからアプリケーションに渡されます。これを、次の図に示します。



X22152-121818

図 8-3: FPGA マネージャーのフロー

ザイリンクス メモリ保護ユニット

ザイリンクス メモリ保護ユニット (XMPU) は領域ベースのメモリ保護ユニットです。詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「システム保護ユニット」の章を参照してください。

XMPU によるメモリ保護

XMPU の機能の詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「システム保護ユニット」の章を参照してください。

XMPU レジスタの設定

XMPU は、ワнтаイム設定、またはセキュア マスター (PMU、APU TrustZone セキュア マスター、またはセキュア マスターとして指定されている場合の RPU) から trust-zone アクセスを介して設定可能です。ブート時に XMPU を設定して、その設定をロックすると、次のパワーオン リセットまで設定を変更できません。設定をロックしない場合、XMPU はセキュア マスター アクセスによって何回でも設定可能です。XMPU を動的に設定する場合、アクティブなデバイスおよび AXI バスをアイドルにするなど、多くの点に注意が必要です。

XMPU の詳細は、『Zynq UltraScale+ MPSoC のアイソレーション手法』(XAPP1320) [\[参照 10\]](#) を参照してください。

ザイリンクス ペリフェラル保護ユニット

ザイリンクス ペリフェラル保護ユニット (XPPU) の機能および動作の詳細は、[このセクション](#) (『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「ザイリンクス ペリフェラル保護ユニット」) を参照してください。

XMPU の詳細は、『Zynq UltraScale+ MPSoC のアイソレーション手法』(XAPP1320) [\[参照 10\]](#) を参照してください。

システム メモリ管理ユニット

システム メモリ管理ユニット (SMMU) は、分離サービスを提供します。SMMU は、I/O デバイスが実際に可能な範囲を超えてアドレス指定できるようにします。メモリを分離しないと、I/O デバイスによってシステム メモリが破壊される可能性があります。SMMU はデバイスを分離して DMA 攻撃を防ぎます。分離とメモリ保護のために、SMMU は DMA 対応 I/O に対するデバイス アクセスを事前に割り当てた物理空間のみに制限します。

SMMU の機能と動作の詳細は、[このセクション](#) (『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「システム メモリ管理ユニット」) を参照してください。

A53 メモリ管理ユニット

メモリ管理ユニット (MMU) はメイン メモリ内の変換テーブルにアクセスするテーブル ウォーク ハードウェアを制御します。また、仮想アドレスを物理アドレスに変換します。MMU は、ページ テーブルに格納された仮想アドレスから物理アドレスへのマップおよびメモリ属性を組み合わせることで細精度のメモリ システム制御を可能にします。これらは、メモリ アドレスにアクセスする際に変換ルックアサイド バッファ (TLB) に読み込まれます。

MMU の機能の詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「メモリ管理ユニット」のセクションを参照してください。

R5 メモリ保護ユニット

メモリ保護ユニット (MPU) を有効にすると、メモリを複数の領域に分割して各領域に対して個別に保護属性を設定できます。MPU を無効にするとアクセス許可のチェックは実行されず、メモリ属性はデフォルトのメモリ マップに従って割り当てられます。MPU には最大 16 の領域があります。

MPU の機能の詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「メモリ保護ユニット」のセクションを参照してください。

プラットフォーム管理

はじめに

Zynq® UltraScale+™ MPSoC デバイスは、高性能と低消費電力の両立が要求される幅広い市場のアプリケーション向けに設計されています。システムの消費電力は、各種サブシステムをソフトウェアでどれだけ高度に管理できるかによって決まります。つまり、必要なときだけサブシステムの電源をオンにしたり、より細かなレベルでパフォーマンスと消費電力のトレードオフを最適化することが求められます。この章では、消費電力の管理に関する機能、およびソフトウェアを使用して各種パワー モードを制御する方法について説明します。

PS のプラットフォーム管理

スケーラブルなプラットフォーム管理ユニット (PMU) を実現するため、Zynq UltraScale+ MPSoC デバイスは次に示す複数の電源ドメインをサポートしています。

- フル電力ドメイン (FPD)
- 低電力ドメイン (LPD)
- バッテリ電源ドメイン
- PL パワードメイン

PMU およびオプションの PMU ファームウェア機能の詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] を参照してください。

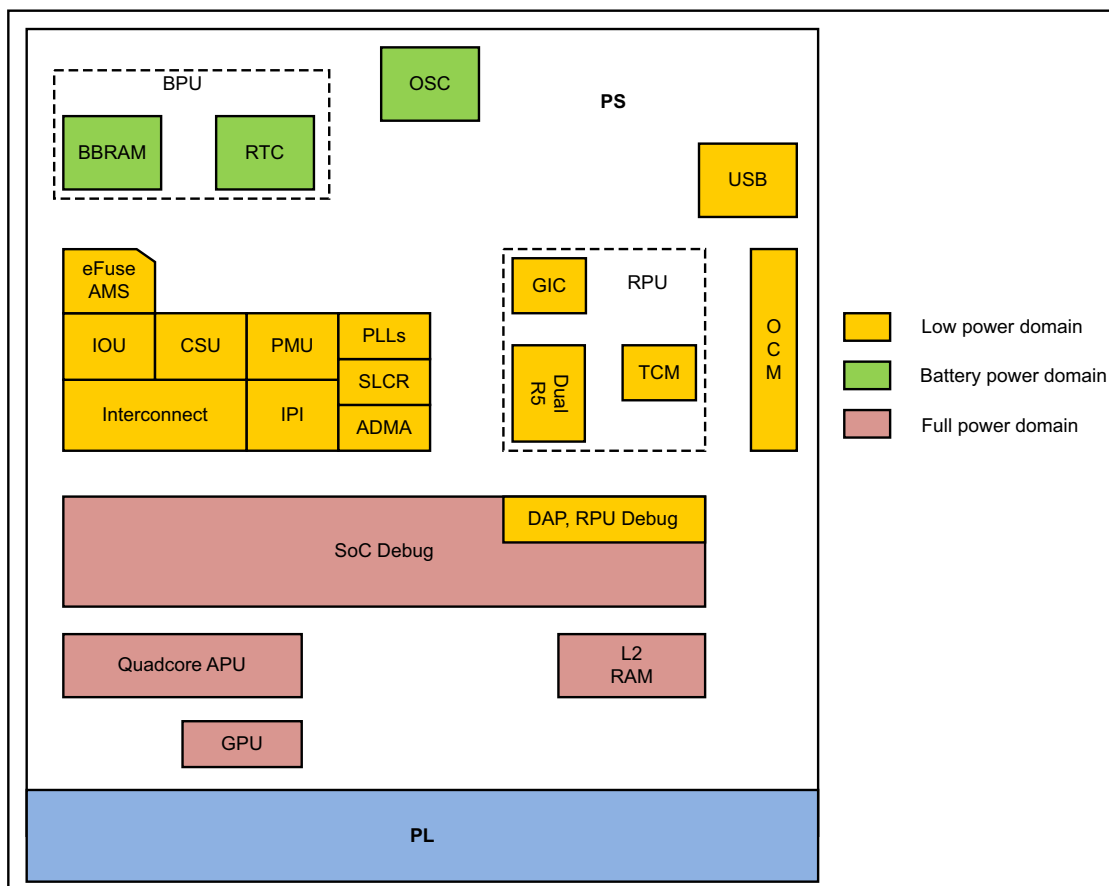
PS のクロックを動的に変更する方法の詳細は、第 14 章「クロックおよび周波数管理」を参照してください。

PS ブロックは高度な機能と性能を備えています。しかし同時に、動作の各段階で必要な機能とパフォーマンスに応じて PS ブロックの消費電力を最適化したいという強い要求も存在します。

Zynq UltraScale+ MPSoC デバイスには複数の電源レールがあります。各レールは個別にオフにすることも、異なる電圧を使用することもできます。また、各電源レールに属するブロックの多くは電力ゲーティングを実装しており、個別にゲートオフが可能です。

これらの電力ゲーティングに対応したドメインの例としては、Arm® Cortex™-A53/Cortex-R5F プロセッサ、GPU ピクセル プロセッサ (PP)、大容量 RAM、個々の USB などがあります。

次に、PS レベルでのプラットフォーム管理のブロック図を示します。



X19226-071317

図 9-1: PS レベルのプラットフォーム管理

消費電力に関して、Zynq UltraScale+ MPSoC デバイスは PS レベルで次の動作モードをサポートしています。

- フル電力動作モード
- 低電力動作モード
- ディープ スリープ モード
- シャットダウン モード
- バッテリー電源モード

以降のセクションでは、これらのモードについて説明します。

フル電力動作モード

フル電力動作モード (図 9-1 に示したフル電力ドメイン) では、システム全体が動作します。全体の消費電力は、動作しているコンポーネントの数 (それらのステートと動作周波数) によって決まります。このモードでは、消費電力全体のほとんどをダイナミック消費電力が占めます。

フル電力動作モードでスタティック消費電力とダイナミック消費電力を最適化するため、大規模なモジュールはすべて専用の電源アイランドを持っており、使用しない間はシャットダウンできます。

フル電力動作モードの詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「プラットフォーム管理ユニット」の章を参照してください。

低電力動作モード

低電力動作モードでは、PMU、RPU、CSU、IOU など PS の一部のブロック (図 9-1 で低電力ドメインとして表示したブロック) への電源が供給されます。

このモードでは、システム周波数の変更によって消費電力を調整できます。CSU は、SEU および改ざんからシステムを保護するために動作を継続する必要があります。

低電力動作モードでは、低電力ドメインに属するすべてのペリフェラルが動作します。低電力動作モードに属するブロックの中で、PLL、デュアル Cortex-R5F、USB、および TCM/OCM ブロック RAM が電力ゲーティングに対応しています。

各ブロックに対する電力ゲーティングは、ソフトウェアで LPD_SLCR レジスタを設定して制御できます。LPD_SLCR レジスタの詳細は、SLCR レジスタのページ [参照 12] を参照してください。

* SATA、PCIe、および DisplayPort ブロックはフル電力ドメイン (FPD) に属します。

ディープスリープ動作モード

ディープスリープ モードは、PS がサスペンド状態に移行してウェークアップ信号を待つ特別なモードです。ウェークアップは、MIO、USB、または RTC でトリガーできます。

ウェークアップ後、PS はブート プロセスを実行する必要がなく、システムのセキュリティ ステートは維持されます。ディープスリープ モードでは、ブートおよびセキュリティ ステートを維持したままデバイスの消費電力を最小限に抑えることができます。

このモードでは、システム モニターや PLL など低電力ドメインに属するブロックを除き、すべてのブロックがパワーダウンされます。LPD では、Cortex-R5F がパワーダウンします。このモードではコンテキストを維持するため、TCM と OCM はリテンション ステートとなります。

シャットダウン モード

シャットダウン モードでは、APU コア全体がパワーダウンします。このモードは、APU にのみ適用可能です。シャットダウン中、キャッシュを含むすべてのプロセッサ ステートが完全に失われます。したがって、PMU に対して APU コアのパワーダウンを要求する前に、ソフトウェアですべてのステートを保存しておく必要があります。

CPU がシャットダウンした後、いずれかのペリフェラルからその CPU に関係する割り込みが発生すると CPU はパワーアップを開始する必要があります。このため、APU コアへの割り込みラインは PMU の割り込みコントローラーにも接続されており、APU コアがパワーダウンするとこの割り込みラインが有効になります。

API のシャットダウン呼び出しについては、『Embedded Energy Management Interface: EEMI API リファレンス ガイド』(UG1200) [参照 17] を参照してください。

詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「PMU のプログラミング モデル」のセクションを参照してください。

バッテリー電源モード

システムがオフの間も、PS の一部の機能はバッテリーで動作を継続する必要があります。次の機能は、バッテリー電源ドメインの PS (図 9-1 参照) で動作します。

- バッテリー バックアップ RAM (BBRAM): セキュア コンフィギュレーション用のキーを格納
- リアルタイム クロック (RTC): クリスタル I/O を含む

Zynq UltraScale+ MPSoC デバイスにはバッテリー電源ドメインが 1 つしかありません。したがって、PS に実装された機能のみバッテリー バックアップ可能です。バッテリー電源ドメインに必要な I/O としては、バッテリー電源パッドおよび RTC クリスタル用 I/O パッドがあります。

ウェークアップ メカニズム

ウェークアップ メカニズムの詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「プラットフォーム管理ユニット」の章の「PMU の動作」を参照してください。

メモリのプラットフォーム管理

Zynq UltraScale+ MPSoC デバイスは、L2 キャッシュ、OCM、TCM などの大容量 RAM を備えています。これらの RAM は、クロック ゲーティング、電力ゲーティング、メモリ リテンション モードなどの電力管理機能をサポートしています。

- TCM と OCM は個別に電力ゲーティングとリテンション モードをサポートしています。
- L2 キャッシュ コントローラーはダイナミック クロック ゲーティング、リテンション、およびシャットダウン モードをサポートしており、細かな消費電力の削減が可能です。

DDR コントローラー

DDR コントローラーには、消費電力を削減する目的で次のメカニズムが実装されています。

- **クロック停止:** この機能を有効にすると、DDR PHY は DRAM へのクロックを停止できます。
 - 。 DDR2 と DDR3 では、この機能はセルフ リフレッシュ モードの場合のみ有効です。
 - 。 LPDDR2 では、この機能はアイドル期間、パワーダウン モード、セルフ リフレッシュ モード、およびディープ パワーダウン モードでも有効です。
- **プリチャージパワーダウン:** この機能を有効にすると、DDRC は動的にプリチャージパワーダウン モードを使用してアイドル期間の消費電力を削減します。コントローラーが新規要求を受信すると、通常動作が再開します。
- **セルフ リフレッシュ:** アイドル期間中、DDR コントローラーは DRAM を動的にセルフ リフレッシュ モードに移行できます。コントローラーが新規要求を受信すると、通常動作が再開します。

このモードでは、DDRC コア ロジックの電源を完全にオフにしても DRAM の内容が維持されます。このため、DDR 終端を制御する DDR3X クロックおよび DCI クロックを停止できます。

インターコネクトのプラットフォーム管理

インターコネクトは、複数の電源レールおよび電源アイランドにまたがっています。これらの電源レールや電源アイランドは、別々にオン/オフが可能です。インプリメンテーションを容易にするため、互いに通信を行う 2 つの電源ドメインに対するクロックは、ほとんどの場合非同期とする必要があります。したがって、これらドメインの相互接続にはシンクロナイザーが必要です。

タイミング要件を緩和するため、電源ドメインはちょうどクロックが交差する場所に配置されます。シンクロナイザーは、接続する 2 つのドメインのそれぞれに実装し、ブリッジを形成する必要があります。

このブリッジはスレーブ インターフェイスとマスター インターフェイスがそれぞれ 1 つずつで構成され、各インターフェイスは 1 つの電源およびクロック ドメイン内に完全に包含されます。このインターフェイスのクロック周波数は互いに独立して変更でき、リセットもそれぞれ独立して実行できます。

多電圧実装またはパワーダウンのためには、ブリッジの両端の間にレベル シフターまたはクランプのいずれか、または両方を実装する必要があります。

また、このブリッジは次の方法でオープンなトランザクションを追跡します。

- PMU からパワーダウン要求を受信すると、ブリッジはその要求をログに記録します。
- 過去の未処理のトランザクションが通常どおり処理されている間、トランザクション カウンターが 0 になるまで新規トランザクションはすべてエラーを返します。カウンターが 0 になるとブリッジは PMU に肯定応答 (ACK) を返し、ブリッジに接続されたマスターまたはスレーブをシャットダウンしても安全であることを通知します。
- インターコネクト内のすべてのブリッジがアイドルになった場合のみ、インターコネクト全体がシャットダウンします。

詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「プラットフォーム管理ユニット」の章の「PMU インターコネクト」のセクションを参照してください。

PMU ファームウェア

ザイリンクスがサポートするシステム コンフィギュレーションには、パワーアップおよびスリープ管理の機能に加え、PMU ファームウェアが必ず含まれています。PMU は高度なシステム監視および電力管理アルゴリズムを実装したユーザー プログラムを実行できます。このモードでは、APU または RPU は電力管理プログラムをインバウンド LPD スイッチ経由で PMU 内部の RAM にコピーします。PMU は、ザイリンクスがサポートするシステム コンフィギュレーションの範囲内で必要なリセット、電力管理、システム監視、および割り込み制御を実装したソフトウェアを実行します。

詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [\[参照 11\]](#) の「PMU のプログラミング モデル」のセクションを参照してください。

カスタム PMU ファームウェアの作成には、ザイリンクス SDK を使用できます。SDK には、PMU ファームウェア テンプレートのソース コードと必要なライブラリ サポートが含まれています。SDK プロジェクトの作成方法は、[第 5 章「ソフトウェア開発フロー」](#)を参照してください。

電力管理フレームワーク

電力管理 API 関数の使用方法は、『Embedded Energy Management Interface: EEMI API リファレンス ガイド』(UG1200) [\[参照 17\]](#) を参照してください。

注記: ザイリンクス EEMI API に関する、ベアメタル、FreeRTOS、Linux の電力管理方法に違いはありません。

プラットフォーム管理ユニット ファームウェア

はじめに

Zynq® UltraScale+™ MPSoC デバイスのプラットフォーム管理ユニット (PMU) は、低電力サブシステムにあります。PMU 内では、MicroBlaze プロセッサが 32KB ROM および 128KB RAM からフラットなメモリ空間へ実行コードをロードします。PMU は、プロセッサ間割り込みや電力管理レジスタなどを使用してシステム内リソースのパワーアップ、リセット、および監視を制御します。ROM には、プリブート タスクを実行してサービス モードに入るための PMU bootROM (PBR) が格納されています。ザイリンクスがサポートする各ユース ケースの高度なシステム機能を実行するには、PMU ファームウェアをロードする必要があります。この章では、Zynq UltraScale+ MPSoC デバイス用に開発された PMU ファームウェアの特長と機能について説明します。

特長

PMU ファームウェアの主な特長は次のとおりです。

- モジュールで機能を提供: PMU ファームウェアは、モジュラー型の設計を採用しています。モジュールの形でユーザーが新しい機能を追加できます。
- モジュールのカスタマイズが容易
- ユーザー アプリケーションに必要な機能のみを含めるように容易に設定可能
- IPI (プロセッサ間割り込み) により、システム内のほかのコンポーネントとの通信をサポート
- EM モジュールを実行時に設定可能
- さまざまな電力管理機能をサポート

PMU ファームウェアのアーキテクチャ

図 10-1 に、PMU ファームウェアのアーキテクチャ ブロック図を示します。PMU ファームウェアはモジュラー型的设计を採用しており、モジュールの形で新しい機能を追加できます。各機能がモジュールとして分離されているため、ユーザー アプリケーションに必要な機能のみを PMU ファームウェアに含めるように設定できます。このようなモジュラー設計とすることで、新しい機能を容易に追加できるほか、使用しないモジュールを無効にしてメモリフットプリントを最小に抑えることができます。

PMU ファームウェアは、ベース ファームウェアとモジュールで構成されます。ベース ファームウェアは、モジュールの初期化や各モジュールに対するイベントの登録を実行し、すべてのモジュールに必要な共通機能を提供します。これらの共通機能は、次の API に分類されます。

1. PMU ファームウェア コア API
 - a. スケジューラ
 - b. イベント マネージャー
 - c. IPI マネージャー
2. PMU ファームウェア一般 API
 - a. BSP/ユーティリティ API
 - b. リセット サービス API
 - c. ROM サービス API

PMU ファームウェアの各モジュールは、必要に応じてこれらの API を使用して、指定されたアクションを実行します。

Platform Management Overview

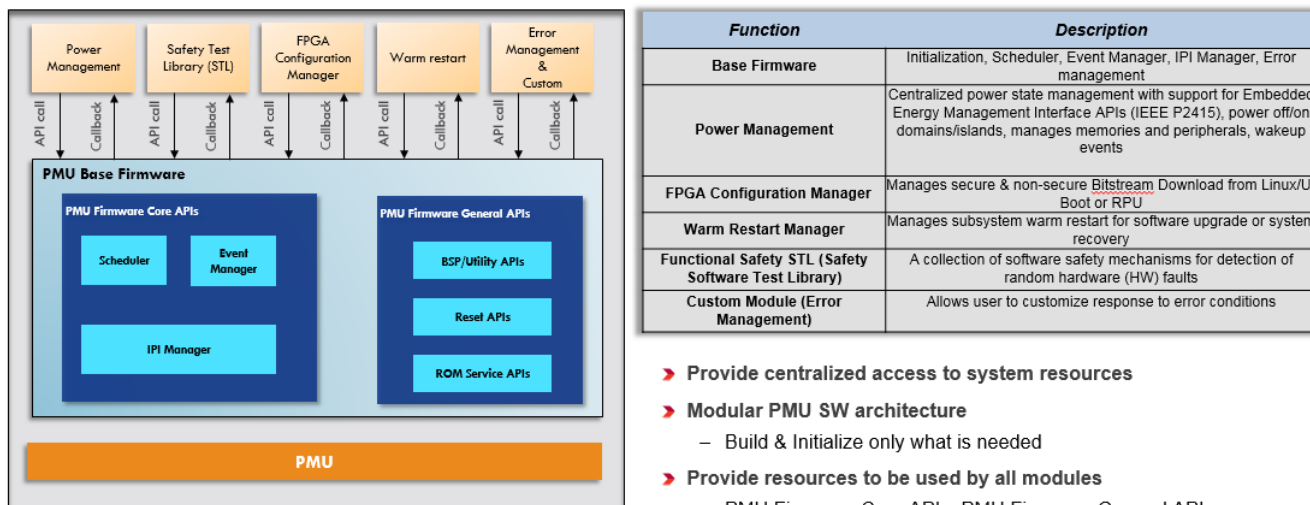


図 10-1: PMU ファームウェアのアーキテクチャ ブロック図

実行フロー

PMU ファームウェアの初期化は、通常のコンテキストで実行されます。意図しない割り込みによって、初期化が完了していないシステム リソースが使用されるのを防ぐため、割り込みは無効にされます。初期化が完了すると、割り込みが有効になり、必要なタスクの実行がスケジュールされます。システムはスリープ ステートに移行します。システムは、何らかのイベントが発生した場合、またはスケジュールしたタスクがトリガーされ、対応するハンドラーが実行される場合のみウェークアップします。図 10-2 に、PMU ファームウェアのステート遷移を示します。

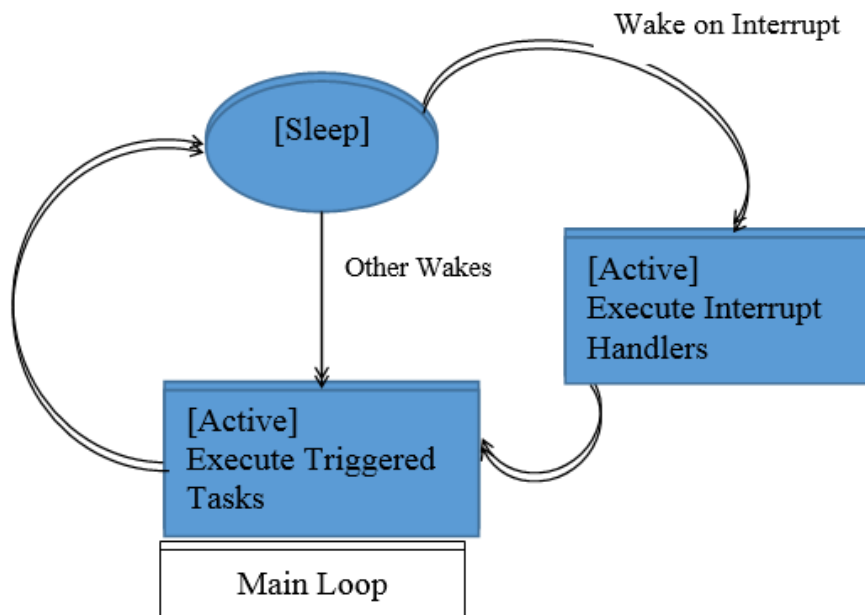


図 10-2: PMU ファームウェアのメイン ループのステート遷移

PMU ファームウェアの実行フローは、次の 3 つのフェーズで構成されます。

1. **初期化フェーズ:** このフェーズには、PMU ファームウェアの起動、セルフテストと検証の実行、ハードウェアの初期化、モジュールの作成と初期化が含まれます。このフェーズでは割り込みが無効にされ、これはフェーズが完了すると有効になります。
2. **初期化後:** このフェーズでは、PMU ファームウェアはスリープに移行して割り込みを待ちます。これをサービスモードと呼びます。
3. **ウェークアップ:** PMU ファームウェアは割り込みコンテキストに入り、割り込みを処理します。このタスクが完了すると、再びスリープに戻ります。

図 10-3 に、PMU ファームウェアの実行フローを示します。

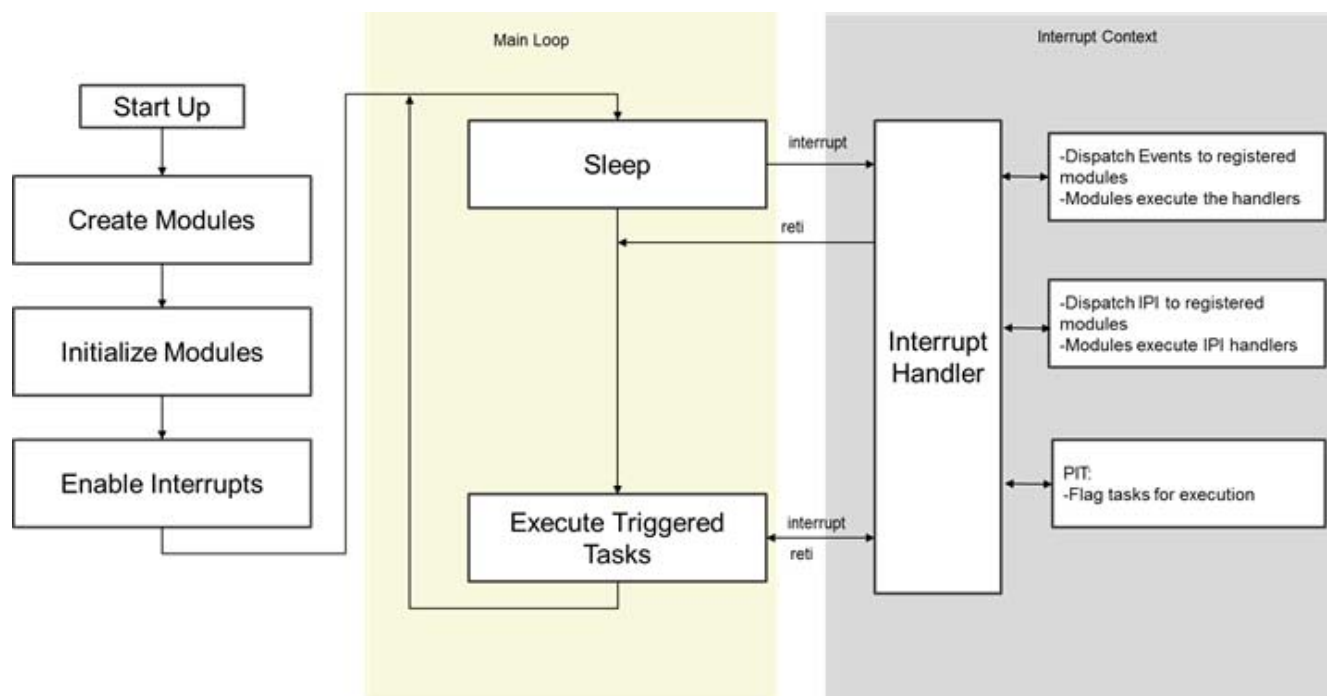


図 10-3: PMU ファームウェアの実行コンテキスト ビュー

PMU ファームウェアにおけるプロセッサ間割り込みの処理

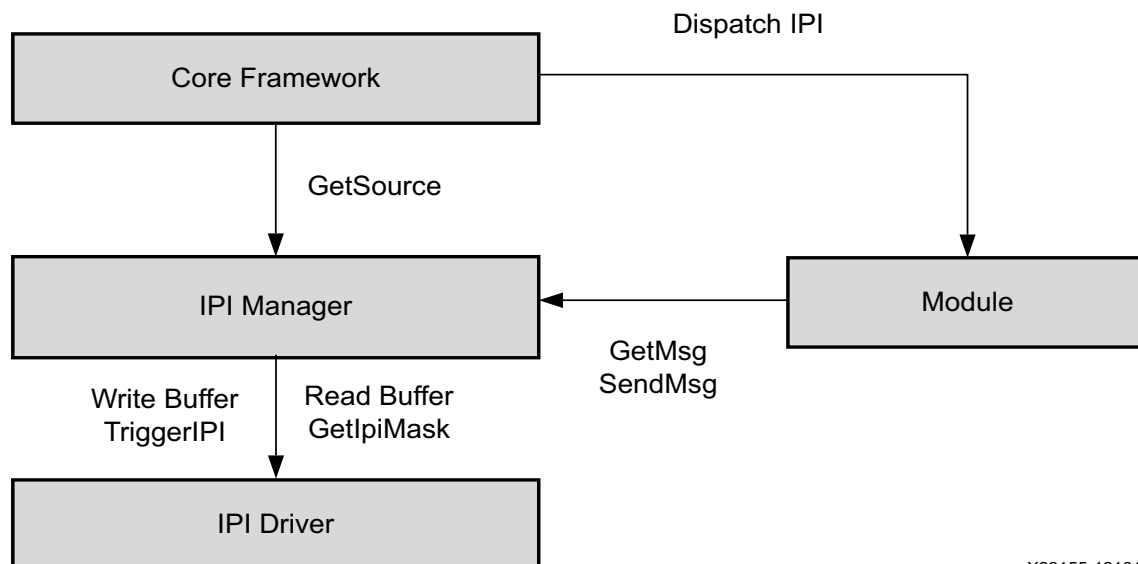
プロセッサ間割り込み (IPI) は、PMU ファームウェアと SoC 上の PMU 以外のエンティティを接続する重要なインターフェイスです。PMU には 4 つの IPI とバッファが割り当てられています。デフォルトでは、PMU ファームウェアは SoC 上のほかのマスターから PMU に対して開始された通信には IPI-0 と関連バッファを使用します。PMU からほかのマスターへのコールバック、および PMU ファームウェアが開始した通信には、IPI-1 と関連バッファを使用します。

図 10-4 に、IPI ハンドラー スタックとこのプロセスに関与する各種コンポーネント間のインターフェイスを示します。PMU ファームウェアは、IPI ドライバーを使用してメッセージを送受信します。ベースファームウェアの IPI ドライバーの上位層に実装された IPI マネージャーは、メッセージの先頭ワードに格納された IPI ID に基づいて、IPI メッセージを登録済みのモジュールハンドラーへ送信します。表 10-1 に、IPI メッセージのフォーマットを示します。

表 10-1: IPI メッセージのフォーマット

ワード	内容	説明
0	ヘッダー	<target_module_id, api_id>
1	ペイロード	モジュール依存のペイロード
2		
3		
4		
5		
6	予約	将来のために予約
7	チェックサム	

PMU からほかのマスターへのコールバック、および PMU ファームウェアが開始した通信には、IPI-1 と関連バッファを使用します。現在、IPI を使用するのは PM および EM モジュールです。IPI メッセージングを必要とするカスタム モジュールを実装する場合は、これらを参考にご覧ください。



X22155-121818

図 10-4: IPI ハンドラー スタックとインターフェイス

PMU ファームウェアには、IPI ドライバー機能に対するラッパー API が用意されており、これを使用して IPI メッセージを送受信できます。初期化フェーズで、PMU ファームウェアは IPI ドライバーを初期化し、IPI で割り当てられたマスターからの IPI 割り込みを有効にします。

IPI メッセージの送信

XPfw_IpiWriteMessage() API は、IPI メッセージをターゲットに送信します。この関数は、内部で IPI ドライバーの write API (バッファ タイプはメッセージ バッファ) を呼び出します。

パラメーター

表 10-2: IPI メッセージの送信

パラメーター	説明
ModPtr	IPI メッセージ送信元のモジュール ポインター。IPI メッセージの target_module_id フィールドは、モジュール ポインターにあるモジュール IPI ID の情報で更新されます。
DestCpuMask	デスティネーション ターゲットの IPI ID
MsgPtr	メッセージ ポインター
MsgLen	メッセージ長さ

戻り値

XST_SUCCESS: メッセージが正しく送信された場合。

XST_FAILURE: メッセージ エラーが発生した場合。

IPI 応答の送信

XPfw_IpiWriteResponse() API は、IPI メッセージ送信元のマスターに応答を送信します。この関数は、内部で IPI ドライバーの write API (バッファ タイプは応答バッファ) を呼び出します。

パラメーター

表 10-3: IPI 応答の送信

パラメーター	説明
ModPtr	この IPI 応答をどのモジュールが受信したかを調べるためのモジュール ポインター
SrcCpuMask	IPI 応答を読み出すソースの IPI ID
MsgPtr	応答メッセージ ポインター
MsgLen	応答メッセージ長さ

戻り値

XST_SUCCESS: IPI 応答が正しく送信された場合。

XST_FAILURE: 応答エラーが発生した場合。

IPI メッセージの読み出し

XPfw_IpiReadMessage() は、IPI 割り込み発生時に受信した IPI メッセージを読み出します。この関数は、内部で IPI ドライバーの read API (バッファ タイプはメッセージバッファ) を呼び出します。

パラメーター

表 10-4: IPI メッセージの読み出し

パラメーター	説明
SrcCpuMask	IPI メッセージを読み出すソースの IPI ID
MsgPtr	メッセージ ポインター
MsgLen	メッセージ長さ

戻り値

XST_SUCCESS: IPI メッセージが正しく読み出された場合。

XST_FAILURE: メッセージ エラーが発生した場合。

IPI 応答の読み出し

XPfw_IpiReadResponse() は、送信したメッセージに対する IPI 応答を読み出します。この関数は、内部で IPI ドライバーの read API (バッファ タイプは応答バッファ) を呼び出します。

パラメーター

表 10-5: IPI 応答の読み出し

パラメーター	説明
ModPtr	この IPI 応答をどのモジュールが受信したかを調べるためのモジュール ポインター
SrcCpuMask	IPI 応答を読み出すソースの IPI ID
MsgPtr	応答メッセージ ポインター
MsgLen	応答メッセージ長さ

戻り値

XST_SUCCESS: IPI 応答が正しく読み出された場合。

XST_FAILURE: 応答エラーが発生した場合。

IPI のトリガー

XPfw_IpiTrigger() は、デスティネーションに対して IPI をトリガーします。この関数は、内部で IPI ドライバーの trigger を呼び出します。この関数は、IPI メッセージが IPI バッファに書き込んだ後に呼び出す必要があります。

パラメーター

表 10-6: IPI のトリガー

パラメーター	説明
DestCpuMask	デスティネーション ターゲットの IPI ID

戻り値

XST_SUCCESS: IPI が正しくトリガーされた場合。

XST_FAILURE: IPI が正しくトリガーされなかった場合。

PMU ファームウェアのモジュール

PMU ファームウェアは、次のモジュールで構成されます。

1. エラー管理 (EM)
2. 電力管理 (PM)
3. スケジューラ
4. セーフティ テスト ライブラリ (STL)

PMU ファームウェアにはモジュール データ構造体 (XPfw_Module_t) があり、この中にモジュールに関する情報が格納されます。作成した各モジュールに対してこのデータ構造体が定義されます。表 10-7 に、この構造体のメンバーを示します。

表 10-7: モジュール データ構造体のメンバー

メンバー	範囲	その他の情報
ModId	0 ~ 31	
CfgInitHandler	初期化ハンドラー関数ポインター	デフォルトは NULL
IpiHandler	IPI マネージャーのハンドラー	デフォルトは NULL
EventHandler	モジュールの登録済みイベント ハンドラー	デフォルトは NULL
IpiId	16 ビット IPI ID	各モジュールに固有

PMU ファームウェアには、すべてのモジュールのリストと詳細を格納したコア データ構造体もあります。表 10-8 に、この構造体のメンバーを示します。

表 10-8: コア データ構造体のメンバー

メンバー	範囲	その他の情報
ModList 配列	0 ~ 31	モジュール構造体のモジュール リストの配列 (32 要素)
スケジューラ	スケジューラ構造体	モジュールがオーナーのスケジューラ タスク
ModCount	0 ~ 31	
IsReady	コアの準備完了/動作なし	
モード	安全診断モード/通常モード	

これらのモジュールが使用する API のいくつかは、PMU のベース ファームウェアでもサポートされます。また、カスタム モジュールを作成する場合は、xpfw_core.h からこれらの API を使用できます。

モジュールの作成

モジュールを作成するには、開始時に `XPfw_CoreCreateMod()` API を呼び出します。PMU ファームウェアには最大 32 個のモジュールを含めることができます。この関数は、モジュールが最大数に達しているかどうかをチェックします。最大数に達していない場合、詳細をコア データ構造体の `ModList` に追加し、このモジュール データ構造体を呼び出し元に返します。それ以外の場合は、`NULL` が返されます。

モジュールのハンドラーのセットアップ

各モジュールは 3 つのハンドラーを利用できます。これらはそれぞれ、次に示すフェーズで呼び出されます。

表 10-9: モジュール ハンドラー

モジュール ハンドラー	目的	ハンドラーを登録するための API	実行コンテキスト
Init	コアの初期化時に呼び出され、モジュールの設定、イベントの登録、またはスケジューラタスクの追加を実行します。必要に応じて、設定データをモジュールに読み込むこともできます。	<code>XPfw_CoreSetCfgHandler(const XPfw_Module_t *ModPtr, XPfwModCfgInitHandler_t CfgHandler);</code>	StartUp
Event Handler	イベント発生時に呼び出されます (イベントは、なるべく初期化フェーズで登録しておきます)。	<code>XPfw_CoreSetEventHandler(const XPfw_Module_t *ModPtr, XPfwModEventHandler_t EventHandler);</code>	割り込み
IPI Handler	該当する <code>module-id</code> を持つ IPI メッセージを受信した場合に呼び出されます。	<code>XPfw_CoreSetIpiHandler(const XPfw_Module_t *ModPtr, XPfwModIpiHandler_t IpiHandler, u16 IpiId);</code>	割り込み

PMU ファームウェアのビルド フラグ

PMU ファームウェアの各モジュールは、必要に応じて有効/無効にできます。これは、ビルド フラグを使用して指定します。表 10-10 に、PMU ファームウェアの主なビルド フラグとその使用法を示します。すべてのビルド フラグの一覧は、PMU ファームウェア ソースの `xpfw_config.h` ファイルを参照してください。

表 10-10: PMU ファームウェアのビルド フラグ

フラグ	説明	必要条件	デフォルト設定
<code>XPFW_DEBUG_DETAILED</code>	PMU ファームウェアで詳細なデバッグプリントを有効にします。 この機能は、2017.3 リリース以降でサポートされます。		無効
<code>PM_LOG_LEVEL</code>	PM モジュールのプリント ベースのデバッグ機能を有効にします。 設定可能な値: <ul style="list-style-type: none"> アラート エラー 警告 情報 大きい値を指定すると、それ以下の値も暗黙的に指定されます。 たとえば 3 (警告) を有効にすると、1 (アラート) と 2 (エラー) も有効になります。		無効
<code>ENABLE_EM</code>	エラー管理 (EM) モジュールを有効にします。	<code>ENABLE_SCHEDULER</code>	無効
<code>ENABLE_ESCALATION</code>	サブシステムのリスタートを正常に実行できなかった場合に、 SRST/PS 専用のリセットへのエスカレーションを有効にします。	<code>ENABLE_RECOVERY</code>	無効
<code>ENABLE_RECOVERY</code>	WDT を使用した APU サブシステムのリスタートを有効にします。	<code>ENABLE_EM</code> 、 <code>ENABLE_PM</code> 、 <code>ENABLE_SCHEDULER</code>	無効
<code>ENABLE_PM</code>	電力管理 (PM) モジュールを有効にします。		有効
<code>ENABLE_NODE_IDLING</code>	サブシステムを強制シャットダウンする前にノードをアイドルにしてリセットします。		無効
<code>ENABLE_SCHEDULER</code>	スケジューラ モジュールを有効にします。		無効
<code>ENABLE_WDT</code>	CSU WDT によるシステム リスタートを PMU が利用できるようにします。	<code>ENABLE_SCHEDULER</code> 、 <code>ENABLE_EM</code>	無効
<code>ENABLE_STL</code>	STL モジュールを有効にします。	なし	無効

表 10-10: PMU ファームウェアのビルド フラグ (続き)

フラグ	説明	必要条件	デフォルト設定
ENABLE_RTC_TEST	RTC イベント ハンドラー テスト モジュールを有効にします。	なし	無効
ENABLE_SAFETY	IPI メッセージに対する CRC 計算を有効にします。	なし	無効
ENABLE_FPGA_LOAD	FPGA ビットストリームの読み込み機能を有効にします。	ENABLE_PM	有効
ENABLE_SECURE	セキュリティ機能を有効にします。	ENABLE_PM	有効
IDLE_PERIPHERALS	PS 専用のリセットまたはシステムリセットの前にペリフェラルをアイドルにします。	ENABLE_PM	無効
ENABLE_POS	パワーオフ サスペンド機能を有効にします。	ENABLE_PM	無効
EFUSE_ACCESS	eFUSE アクセス機能を有効にします。	ENABLE_PM	無効
ENABLE_UNUSED_RPU_PWR_DWN	PmInitFinalize 受信後、動作していない RPU およびスレーブの電源を遮断します。		有効

エラー管理 (EM) モジュール

エラー管理ハードウェア

Zynq UltraScale+ MPSoC には、SoC 全体で致命的なエラーを総括して処理するための専用エラー ハンドラーがあります。詳細は、テクニカル リファレンス マニュアル/アーキテクチャ仕様を参照してください。

エラー マネージャーに転送されるすべての致命的エラーの処理方法は、ハードウェアで処理して SRST/PoR/PS_ERROR_OUT をトリガーするか、PMU への割り込みをトリガーするかを設定できます。

PMU ファームウェアでのエラー管理

エラー管理モジュールはハードウェアで生成されたエラーを初期化および処理します。ハンドラーは、ユーザーがカスタマイズできます。ハードウェアには、発生したエラーの種類を記録するエラー ステータス レジスタが 2 つあります。また、各エラーは PMU MicroBlaze に割り込みを送信するかどうかを設定できます。各エラーに対して、その発生時にどのようなアクションを実行するかはユーザーが選択できます。次に示すいずれか 1 つ、または複数の組み合わせが可能です。

1. デバイスの PS_ERROR_OUT 信号をアサートする。
2. PMU プロセッサへの割り込みを生成する。
3. システム リセット (SRST) を生成する。
4. パワーオン リセット (POR) を生成する。

PMU ファームウェアには、カスタム エラー ハンドラーを登録する (エラーに対するデフォルトのアクション (SRST/PoR/PS_ERROR_OUT) を割り当てる) ための API があります。PMU ファームウェアは、`xpfw_error_manager.c` で定義された `ErrorTable[]` 構造体に従って、起動時にエラー アクションを PMU への割り込み、または `PS_ERROR_OUT` に設定します。

エラー管理 API 呼び出し

このセクションでは、PMU ファームウェアのエラー管理モジュールでサポートされる API について説明します。

エラー アクションのセットアップ

`XPfw_EmSetAction()` は、指定したエラーに対するアクションをセットアップします。

パラメーター

表 10-11: `XPfw_EmSetAction`

パラメーター	説明
ErrorId	表 10-19 で定義したエラー ID を指定します。
ActionId	表 10-20 で定義したいずれか 1 つのアクションを指定します。
ErrorHandler	ErrorHandler は、エラー アクションが PMU への割り込みの場合に呼び出されるハンドラーです。

戻り値

`XST_SUCCESS`: エラー アクションが正しく設定された場合。

`XST_FAILURE`: エラー アクションが正しく実行されなかった場合。

エラー アクションの削除

`XPfw_EmDisable()` は、指定したエラーに対するエラー アクションを削除します。

パラメーター

表 10-12: `XPfw_EmDisable`

パラメーター	説明
ErrorId	エラー アクションを削除するエラーの ID を指定します。

戻り値

`XST_SUCCESS`: 正しく実行された場合。

`XST_FAILURE`: 正しく実行されなかった場合。

エラーの処理

XPfw_EmProcessError() は、発生したエラーを処理します。エラーにエラー ハンドラーが割り当てられている場合は、この関数によってそのハンドラーが呼び出され、適切なアクションが実行されます。

パラメーター

表 10-13: XPfw_EmProcessError

パラメーター	説明
ErrorType	受信したエラーのタイプ (EM_ERR_TYPE_1: PMU GLOBAL ERROR_STATUS_1 のエラー EM_ERR_TYPE_2: PMU GLOBAL ERROR_STATUS_2 のエラー)

戻り値

XST_SUCCESS: 正しく実行された場合。

XST_FAILURE: 正しく実行されなかった場合。

EM モジュールによる IPI の処理

PM モジュール同様、EM モジュールも IPI-0 チャンネルを使用してメッセージを交換します。APU および RPU 0/1 マスターは IPI を使用してこのモジュールと通信ができます。受信したメッセージに基づいてどのモジュールがアクションを実行するかは、IPI メッセージ内の target_module_id で区別します。EM モジュールに登録された IPI ハンドラーの target_module_id は、0xE です。現在、PMU ファームウェアがサポートする IPI メッセージは、[表 10-14](#) に示すもののみです。

表 10-14: PMU ファームウェアでサポートされる IPI メッセージ

S.No	IPI メッセージ	IPI メッセージ ID/API ID
1	エラー アクションを設定	0x1
2	エラー アクションを削除	0x2
3	発生したエラーを送信	0x3

エラー アクションを設定

いずれかのターゲットからこの IPI メッセージを受信すると、PMU ファームウェアはメッセージで受信したエラー ID に対するエラー アクションを設定します。メッセージの処理が正しく完了すると、ターゲットに SUCCESS (0x0) 応答が返されます。それ以外の場合は、FAILURE (0x1) 応答が返されます。メッセージフォーマットは次のとおりです。

表 10-15: 「エラー アクションを設定」のメッセージフォーマット

ワード	説明
0	<target_module_id, api_id>
1	エラー ID。サポートされるエラー ID は、 表 10-19 を参照してください。
2	エラー アクション。サポートされるエラー アクションは、 表 10-20 を参照してください。

エラー アクションを削除

いずれかのターゲットから PMU ファームウェアへのこの IPI メッセージを受信すると、EM モジュールの IPI ハンドラーは、受信したエラー ID に対するエラー アクションを削除します。メッセージの処理が完了すると、ターゲットに SUCCESS または FAILURE 応答が返されます。メッセージフォーマットは次のとおりです。

表 10-16:「エラーアクションを削除」のメッセージフォーマット

ワード	説明
0	<target_module_id, api_id>
1	エラー ID。サポートされるエラー ID は、表 10-19 を参照してください。

発生したエラーを送信

PMU ファームウェアは、システムで発生したエラーを記録し、要求があるとターゲットに送信します。メッセージフォーマットは次のとおりです。

表 10-17:「発生したエラーを送信」のメッセージフォーマット

ワード	説明
0	<target_module_id, api_id>

表 10-18 に、PMU ファームウェアから送信される応答メッセージを示します。

表 10-18: PMU ファームウェアからの応答メッセージ

ワード	説明
0	<target_module_id, Success/Failure>
1	Error_1 (ビットの説明は、PMU グローバルレジスタの ERROR_STATUS_1 レジスタと同じです。ビットが 1 にセットされている場合、ERROR_STATUS_1 の対応するエラーが発生したことを示します。)
2	Error_2 (ビットの説明は、PMU グローバルレジスタの ERROR_STATUS_2 レジスタと同じです。ビットが 1 にセットされている場合、ERROR_STATUS_2 の対応するエラーが発生したことを示します。)
3	PMU RAM の訂正可能 ECC カウント

EM エラー ID 一覧

表 10-19: EM エラー ID 一覧

エラー ID	エラー 番号	エラーの説明	デフォルトの エラー アクション
EM_ERR_ID_CSU_ROM	1	CSU bootROM (CBR) のログに記録されたエラー	PS_ERROR_OUT
EM_ERR_ID_PMU_PB	2	プリブート ステージで PMU bootROM (PBR) のログに記録されたエラー	PS_ERROR_OUT
EM_ERR_ID_PMU_SERVICE	3	サービス モードで PBR のログに記録されたエラー	PS_ERROR_OUT
EM_ERR_ID_PMU_FW	4	PMU ファームウェアのログに記録されたエラー	PS_ERROR_OUT
EM_ERR_ID_PMU_UC	5	PMU HW のログに記録された訂正不能エラー。これには PMU ROM パリデーション エラー、PMU TMR エラー、訂正不能 PMU RAM ECC エラー、および PMU ローカルレジスタ アドレス エラーが含まれます。	PS_ERROR_OUT
EM_ERR_ID_CSU	6	CSU HW 関連エラー	PS_ERROR_OUT
EM_ERR_ID_PLL_LOCK	7	PLL のロックが解除されるとセットされるエラー。PLL がロックした後でのみ有効にする必要があります。	PS_ERROR_OUT
EM_ERR_ID_PL	8	PS に転送される一般的な PL エラー	PS_ERROR_OUT
EM_ERR_ID_TO	9	すべてのタイムアウト エラー [FPS_TO, LPS_TO]	PS_ERROR_OUT
EM_ERR_ID_AUX3	10	補助エラー 3	PS_ERROR_OUT
EM_ERR_ID_AUX2	11	補助エラー 2	PS_ERROR_OUT
EM_ERR_ID_AUX1	12	補助エラー 1	PS_ERROR_OUT
EM_ERR_ID_AUX0	13	補助エラー 0	PS_ERROR_OUT
EM_ERR_ID_DFT	14	CSU システム ウォッチドッグ タイマー エラー	システム リセット
EM_ERR_ID_CLK_MON	15	クロック モニター エラー	PS_ERROR_OUT
EM_ERR_ID_XMPU	16	XPMU エラー [LPS XMPU, FPS XMPU]	PMU への割り込み
EM_ERR_ID_PWR_SUPPLY	17	電源検出障害エラー	PS_ERROR_OUT
EM_ERR_ID_FPD_SWDT	18	FPD システム ウォッチドッグ タイマー エラー	PMUへの割り込み (ENABLE_RECOVERY フラグを定義した場合)。システム リセット (それ以外の場合)
EM_ERR_ID_LPD_SWDT	19	LPD システム ウォッチドッグ タイマー エラー	システム リセット
EM_ERR_ID_RPU_CCF	20	いずれかの RPU CCF エラーが生成された場合にアサートされる	PS_ERROR_OUT
EM_ERR_ID_RPU_LS	21	いずれかの RPU CCF エラーが生成された場合にアサートされる	PMU への割り込み
EM_ERR_ID_FPD_TEMP	22	FPD 温度シャットダウン アラート	PS_ERROR_OUT
EM_ERR_ID_LPD_TEMP	23	LPD 温度シャットダウン アラート	PS_ERROR_OUT
EM_ERR_ID_RPU1	24	RPU1 エラー (訂正可能および訂正不能エラー)	PS_ERROR_OUT

表 10-19: EM エラー ID 一覧 (続き)

エラー ID	エラー番号	エラーの説明	デフォルトのエラーアクション
EM_ERR_ID_RPU0	25	RPU0 エラー (訂正可能および訂正不能エラー)	PS_ERROR_OUT
EM_ERR_ID_OCM_ECC	26	OCM 訂正不能 ECC エラー	PS_ERROR_OUT
EM_ERR_ID_DDR_ECC	27	DDR 訂正不能 ECC エラー	PS_ERROR_OUT

EM エラー アクション一覧

表 10-20: EM エラー アクション一覧

エラー アクション	エラー アクション番号	エラー アクションの説明
EM_ACTION_POR	1	パワーオン リセットをトリガー
EM_ACTION_SRST	2	システム リセットをトリガー
EM_ACTION_CUSTOM	3	ErrorHandler パラメーターで登録したカスタムハンドラーを呼び出す
EM_ACTION_PSERR	4	PS_ERROR_OUT アクションをトリガー

PLL ロック エラーによって PMU ファームウェアから送信される PS_ERROR_OUT

EM モジュールを有効にする場合は、スケジューラも有効にすることを推奨します。FSBL が psu_init を実行している間、PLL ロック エラーが発生することが予想されます。EM モジュールの初期化中にこのエラーが発生するのを避けるため、PMU ファームウェアは PLL ロック エラーを有効にしません。スケジューラ タスクを使用して、FSBL が psu_init を完了するのを待ちます。FSBL が psu_init の実行を完了すると、PMU ファームウェアはすべての PLL ロック エラーを有効にします。

xpfw_error_management.c には、PLL ロック エラーに対する PMU ファームウェアのデフォルトの動作が次のように記述されています。

```
[EM_ERR_ID_PLL_LOCK] = { .Type = EM_ERR_TYPE_2, .RegMask =
PMU_GLOBAL_ERROR_STATUS_2_PLL_LOCK_MASK, .Action = EM_ACTION_NONE, .Handler =
NullHandler},
```

ここで、PMU_GLOBAL_ERROR_STATUS_2_PLL_LOCK_MASK は値が 0x00001F00 と定義されており、すべての PLL ロック エラーが有効になっています。したがって、デザインで未使用の PLL がロックしていない場合も PS_ERROR_OUT 信号がトリガーされます。つまり、PMU_GLOBAL_ERROR_STATUS_2 レジスタ (ビット [12:8]) でロックしていない PLL が 1 つでもある場合、PS_ERROR_OUT 信号がトリガーされます。

これが本当に問題であるかどうかは、デザインで使用している PLL のステータスを調べることでわかります。

たとえば、IO_PLL と DDR_PLL のみを使用するデザインで PMU_GLOBAL_ERROR_STATUS_2 レジスタの値が 0x1600 の場合、RPU_PLL、APU_PLL、および Video_PLL で PLL ロック エラーが発生しています。ほかのレジスタを調べると、PLL の実際のステータスを確認できます。

PLL_STATUS

- PLL_STATUS (CRL_APB) = FF5E0040: 00000019
- PLL_STATUS (CRF_APB) = FD1A0044: 0000003A

表 10-21: PLL_STATUS

PLL のステータス	ERROR_STATUS_2
IOPLL がロックして安定している	ビット [8] (IO_PLL) = 0
RPLL が安定しているがロックしていない (バイパス)	ビット [9] (RPU_PLL) = 1
APPL が安定しているがロックしていない (バイパス)	ビット [10] (APU_PLL) = 1
DPLL がロックして安定している	ビット [11] (DDR_PLL) = 0
VPLL が安定しているがロックしていない (バイパス)	ビット [12] (Video_PLL) = 1

このように、IO_PLL と DDR_PLL しか使用しないデザインでは、RPU_PLL、APU_PLL、および Video_PLL がロックしていてもエラーではありません。

無意味な PS_ERROR_OUT 信号がトリガーされないように、使用しない PLL は PMU_GLOBAL_ERROR_STATUS_2_PLL_LOCK_MASK でマスクすることを推奨します。

例:

```
#define PMU_GLOBAL_ERROR_STATUS_2_PLL_LOCK_MASK ((u32)0X00000900U) will only
signal on PS_ERROR_OUT IO_PLL and DDR_PLL errors.
```

電力管理 (PM) モジュール

Zynq UltraScale+ MPSoC の電力管理フレームワークは、EEMI (Extensible Energy Management Interface) の実装をベースとしています。チップやデバイス上のさまざまなプロセッシング ユニット (PU) 上で動作するソフトウェア コンポーネントは、このフレームワークを利用して電力管理のための要求を発行したり、要求に応えたりします。

電力管理モジュールは、イベント駆動型のモジュールとして PMU ファームウェアに実装されています。電力管理モジュールによって処理されるイベントは、電力管理イベントと呼ばれます。すべての電力管理イベントは、割り込みを利用してトリガーされます。

割り込みを処理する際、PMU ファームウェアは関連するイベントを電力管理モジュールで処理する必要があるかどうかを判断します。そしてイベントが電力管理に関係するものであると判断した場合、電力管理モジュールが有効であれば、PMU ファームウェアはそれをトリガーしてイベントを処理させます。

たとえば、PS および PL の割り込みはすべて GIC プロキシを介して PMU に転送できます。アプリケーション プロセッサ (APU または RPU) が一時的にサスペンドされている場合、PMU が GIC プロキシ割り込みを処理し、アプリケーション プロセッサをウェークアップし、元の割り込みはアプリケーション プロセッサが処理します。実際には PMU ファームウェアがこれらの割り込みを処理するわけではありませんが、アプリケーション プロセッサではなく PMU で処理するようにファームウェアを自由にカスタマイズできます。詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「割り込み」の章を参照してください。

電力管理イベントを処理する際、電力管理コントローラーは PMU ROM ハンドラーを使用してハードウェア リソースのステート制御に関する特定の処理を実行することがあります。

ウォーム リスタートおよび FPGA コンフィギュレーション マネージャーは、電力管理モジュールに含まれます。PMU ファームウェアには、PL FPGA コンフィギュレーションの機能をサポートする XilFPGA ライブラリと、セキュア機能にアクセスするための XilSecure ライブラリが含まれます。詳細は、[第 11 章「電力管理フレームワーク」](#)を参照してください。

注記: 電力管理モジュールは IPI マネージャー/イベント マネージャーなどのベース ファームウェア API を使用するため、PMU ファームウェアがないとスタンドアロンの電力管理機能を実行できません。XilPM を使用したデザインの例は、「[PM Examples](#)」の [Wiki ページ \(英語\)](#) を参照してください。

スケジューラ

STL などのモジュールは、レジスタ カバレッジ、スクラビングなどの周期的タスクをサポートするためにスケジューラを必要とします。PMU ファームウェアは、LPD WDT 機能にもスケジューラを使用します。詳細は、後述します。PMU MicroBlaze には 4 つの PIT (0 ~ 3) があり、スケジューラは PIT1 を使用します。スケジューラは最大 10 個のタスクをサポートします。次の表に、スケジューラのタスク リストデータ構造体のメンバーを示します。

表 10-22: スケジューラのデータ構造体のメンバー

メンバー	値/範囲	その他の情報
Task ID	0 ~ 9	0 ~ 最大優先度
Interval	タスクの実行間隔 (ms)	
OwnerId	0 ~ 9	このタスクのオーナー モジュール
Status	Enabled/Disabled	
Callback	関数ポインター	デフォルトは NULL

注記: デフォルトでは、スケジューラ機能は無効です。有効にするには、ビルド フラグの ENABLE_SCHEDULER を定義する必要があります。

セーフティ テスト ライブラリ

セーフティ テスト ライブラリ (STL) は、ランダム ハードウェア (HW) 障害を検出するためのハードウェア安全機能を補完するソフトウェア安全メカニズムを集めたものです。PMU ファームウェアには、PMU ファームウェア起動時の STL 初期化用にプレースホルダーがあります。これは、ビルド フラグの ENABLE_STL を定義して有効にします。ソフトウェア ライブラリおよび安全に関する資料は、[機能安全ラウンジ](#)にあります。

CSU/PMU レジスタへのアクセス

このセクションでは、CSU および PMU グローバル レジスタの読み出し/書き込みの方法について説明し、ホワイト リスト レジスタとブラック リスト レジスタを示します。

レジスタの書き込み

```
$ echo <address> <mask> <value> > /sys/firmware/zynqmp/config_reg
```

レジスタの読み出し

```
$ echo <address> > /sys/firmware/zynqmp/config_reg
```

```
$ cat /sys/firmware/zynqmp/config_reg
```

CSU および PMU グローバル レジスタは、2 つのリストに分類されます。

- デフォルトでいつでもアクセスできるレジスタ (ホワイト リスト レジスタ)。次に、ホワイト リスト レジスタを示します。
 - CSU モジュール
 - Csu_status
 - Csu_multi_boot
 - Csu_tamper_trig
 - Csu_ft_status
 - Jtag_chain_status
 - Idcode
 - Version
 - Csu_rom_digest(0:11)
 - Aes_status
 - Pcap_status
 - PMU グローバル モジュール
 - Global_control
 - Global_Gen_Storage0-6
 - Pers_Glob_Gen_Storage0-6
 - Req_Iso_Status
 - Req_SwRst_Status
 - Csu_Br_Error
 - Safety_Chk
- ブラック リスト レジスタは、コンパイル時フラグを設定した場合のみアクセスできます。

CSU モジュールと PMU_GLOBAL モジュールのレジスタのうち、ホワイト リスト レジスタでないものはすべてブラック リスト レジスタです。RSA および RSA_CORE モジュールのレジスタもブラック リスト レジスタです。

#define オプション (SECURE_ACCESS_VAL) を使用すると、ブラック リスト レジスタにアクセスできます。ブラック リスト レジスタにアクセスするには、SECURE_ACCESS_VAL フラグをセットして PMU ファームウェアをビルドします。

タイマー

Zynq UltraScale+ MPSoC には、フル電力ドメイン (FPD) 用と低電力ドメイン (LPD) 用に 1 つずつ、合計 2 つのシステムウォッチドッグ タイマー (WDT) があります。各 WDT は、エラー マネージャーにエラー状態情報を送ります。EM モジュールは、FPD または LPD WDT がタイムアウトした場合に特定のエラー アクションを実行するように設定できます。このセクションでは、これら WDT の使用法と、WDT がタイムアウトした場合の PMU ファームウェアの機能について説明します。

FPD WDT

FPD WDT を使用すると APU または FPD をリセットできます。FPD WDT エラーが発生した場合に実行するエラーアクションは、PMU ファームウェアのエラー管理 (EM) モジュールで設定できます。PMU ファームウェアには、FPD WDT エラーに対するリカバリ メカニズムが実装されています。このメカニズムはデフォルトで無効になっています。このメカニズムを有効にするには、ビルド フラグの ENABLE_RECOVERY を定義します。

リカバリ メカニズムが無効の場合、PMU ファームウェアの EM モジュールは FPD WDT に対するエラー アクションを「システム リセット」に設定します。この場合、PMU ファームウェアは FPD WDT を初期化も設定もしません。Linux ドライバーで適宜 WDT を初期化して開始する必要があります。WDT がタイムアウトすると、システム リスタートが実行されます。

ENABLE_RECOVERY フラグを定義した場合、PMU ファームウェアは FPD WDT に対するエラー アクションを「PMU への割り込み」に設定し、このエラーが発生した場合に呼び出すハンドラーを登録します。この場合、PMU ファームウェアが起動すると、WDT を初期化して開始します。また、TTC を初期化してタイマー モードをインターバルに設定します。

PMU ファームウェアは、FPD WDT が 60 秒でタイムアウトするように設定します。WDT エラーが発生すると、PMU ファームウェアは割り込みを受信し、登録したハンドラーを呼び出します。PMU ファームウェアには、マスターのリスタート フェーズおよびその他の情報を追跡するためのリスタート トラッカー構造体があります。現在、この構造体を使用するマスターは APU のみです。次に、この構造体のメンバーを示します。

表 10-23: リスタート トラッカー構造体のメンバー

メンバー	説明
Master	リスタート サイクルを追跡する対象マスター
RestartState	リスタート サイクルのフェーズを追跡する
RestartScope	FPD WDT エラー割り込みが発生した場合のリスタートの範囲
WdtBaseAddress	このマスターに割り当てられた WDT のベース アドレス
WdtTimeout	WDT のタイムアウト値
ErrorId	WDT に対応するエラー ID
WdtPtr	このマスターの WDT へのポインター
TtcDeviceId	TTC タイマーのデバイス ID
TtcPtr	このマスターの TTC へのポインター

表 10-23: リスタート トラッカー構造体のメンバー (続き)

メンバー	説明
TtcTimeout	マスターにイベントを通知するまでのタイムアウト時間
TtcResetId	TTC のリセット ライン ID

WDT エラーが発生すると、WDT エラー ハンドラーが呼び出され、PMU ファームウェアは次の処理を実行します。

1. マスターが APU で、エラー ID が FPD WDT かどうかをチェックします。次に、リスタート ステートが進行中かどうかをチェックします。進行中でない場合、リスタート ステート進行中に変更し、マスターがリスタートした回数を追跡するためにリスタート カウントをインクリメントします。
2. WDT をリスタートして、WDT エラーが APU アプリケーションによるものでない場合は PMU ファームウェアにそのことがわかるようにします。
3. タイマー割り込み TTC3_0 を使用して ATF に IPI を送信して、APU をアイドルにします。

注記: これは Linux の場合のみで、ベアメタルの場合は ATF が存在しないため、この処理は実行しません。

4. ENABLE_ESCALATION フラグが定義されている場合、最初のリスタートに失敗すると、PMU ファームウェアによってシステム リセットまたは PS 専用のリセットにエスカレーションされます。ENABLE_ESCALATION を定義していない場合、PMU ファームウェアは APU をリスタートします。それ以外の場合、PMU ファームウェアは次の処理を実行します。
 - a. PL のコンフィギュレーションが完了しているかどうかを PMU ファームウェアがチェックします。
 - b. PL のコンフィギュレーションが完了している場合、PMU ファームウェアは PS 専用のリセットを開始します。それ以外の場合、システム リセットを開始します。

注記: Linux で WDT ハートビート アプリケーションが動作していることを確認してください。

CSU WDT

CSU WDT は PMU ファームウェアで使用するよう設定されており、何らかの理由で PMU ファームウェアのアプリケーションがハングするとシステムがリスタートします。この機能は、ENABLE_WDT フラグを定義した場合のみ有効です。

EM モジュールは CSU WDT エラー アクションを「システム リセット」として設定します。CSU WDT を初期化するには、FSBL からの psu_init によって WDT をリセットから解放する必要があります。FSBL は、psu_init 完了のステータスを PMU グローバル汎用ストレージレジスタ 5 に書き込みます。PMU ファームウェアは、この完了ステータスを確認してから CSU WDT を初期化します。PMU ファームウェア初期化時に ENABLE_WDT フラグが定義されている場合は、FSBL によって psu_init 完了ステータスが更新されるまで 100ms ごとにトリガーされるタスクをスケジューラに追加します。psu_init が完了すると、このタスクはスケジューラのタスク リストから削除され、PMU ファームウェアは CSU WDT を初期化して 90ms に設定します。また、50ms ごとに WDT をリスタートするスケジューラ タスクを開始します。PMU ファームウェア コードがハングして CSU WDT エラーが発生した場合、このエラーはハードウェアで処理され、「システム リセット」がトリガーされてシステムがリスタートします。

次に、この WDT 機能を使用するための依存関係を示します。

1. ENABLE_EM フラグを定義して EM モジュールを有効にする。
2. ENABLE_WDT フラグを定義して CSU WDT を有効にする。
3. ENABLE_SCHEDULER を定義してスケジューラ モジュールを有効にし、psu_init の完了をチェックして WDT を周期的にリスタートするタスクをスケジューラに追加する。

コンフィギュレーション オブジェクト

コンフィギュレーション オブジェクトは、PMU ファームウェアの電力管理モジュールのデータ構造体をブート時に更新できるようにするためのバイナリ データ オブジェクトです。コンフィギュレーション オブジェクトは、Zynq UltraScale+ MPSoC のプロセッシング ユニットがメモリにコピーする必要があります。つまり、コンフィギュレーション オブジェクトは PMU からアクセスできるメモリ領域に格納する必要があります。

次の API を呼び出すと、PMU がコンフィギュレーション オブジェクトをロードします。

```
XPm_SetConfiguration(address);
```

引数 `address` は、メモリに格納されたコンフィギュレーション オブジェクトの開始アドレスを表します。PMU は、コンフィギュレーション オブジェクトの内容に基づいてそのサイズを特定します。

PMU は、コンフィギュレーション オブジェクトをロードして、PMU はハードウェア リソース (ノード) のステート管理に使用するデータ構造体を更新します。パーシャル コンフィギュレーションはサポートされません。コンフィギュレーション オブジェクトに、このユーザー ガイドで定義した情報が含まれないか、情報の一部しか含まれない場合、PMU ファームウェアの電力管理データの整合性は保証されません。コンフィギュレーション オブジェクトを作成する際に、格納する情報の整合性を確保する必要があります。いったんコンフィギュレーション オブジェクトをロードすると、PMU はノードのステートを変更しません。また、PMU はコンフィギュレーション オブジェクトに格納されているノードの現在のステートに関する情報が、実際のハードウェアの現在のステートと一致しているかどうかをチェックしません。現在のステートとは、PMU がコンフィギュレーション オブジェクトを処理した時点でのハードウェア リソースのステートです。

コンフィギュレーション オブジェクトは、次のものを指定します。

- システムで利用可能なマスターのリスト
- マスターが現在使用中のすべてのスレーブ ノード、およびマスターからスレーブ コンフィギュレーションに対する現在の要件
- マスターが使用可能なすべてのスレーブ ノード、およびマスターからスレーブ コンフィギュレーションに対するデフォルトの要件
- 各電力ノードに対して、どのマスターが要求/リリース/パワーダウンの実行を許可されているか
- 各リセット ラインに対して、どのマスターがリセット ラインの値の変更要求を許可されているか
- どのシャットダウン モードの要求をマスターが許可されているか、およびそのマスターのシャットダウン タイムアウト値
- システム ブート時に FSBL によって既に設定されているコンフィギュレーションに対し、どのマスターがコンフィギュレーションを設定できるか

PM コンフィギュレーション オブジェクトの生成

PM コンフィギュレーション オブジェクトは、次の手順で生成します。

1. PCW ツールを使用してカスタム PM フレームワーク コンフィギュレーションを指定します。
2. PCW で HDF ファイルを生成します。
3. ビルド時に、HDF パーサーが HDF ファイルを解析し、コンフィギュレーション オブジェクトを FSBL コードに挿入します。

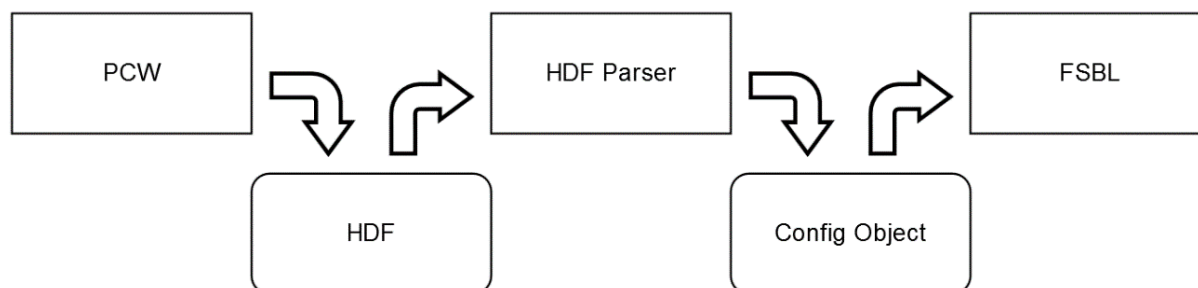


図 10-5: コンフィギュレーション オブジェクトの生成

ブート時の最初のコンフィギュレーション

コンフィギュレーション オブジェクトは、EEMI API を呼び出す前にロードしておく必要があります。ただし、次の API は除きます。

- Get API Version
- Set configuration
- Get Chip ID

最初のコンフィギュレーション オブジェクトがロードされるまで、PM コントローラーは APU または RPU マスターからそれぞれ IPI_APU または IPI_RPU_0 IPI チャンネルを通じて EEMI API 呼び出しを受信するように設定されています。したがって、最初のコンフィギュレーション オブジェクトは APU または RPU によってロードする必要があります。

最初のコンフィギュレーション オブジェクトがロードされたら、次のコンフィギュレーション オブジェクトのロードは特権マスターがトリガーできます。特権マスターは、最後にロードされたコンフィギュレーション オブジェクトで定義されます。

次に、ブート レベルの手順を示します。

1. FSBL が Set Configuration API を使用してコンフィギュレーション オブジェクトを PMU に送信します。
2. PMU がコンフィギュレーション オブジェクトを解析してコンフィギュレーションします。
3. すべてのマスターが初期化を完了すると、PMU は使用されていないすべてのノードの電源をオフにします。

最初に Set Configuration API を呼び出す前の要求はすべて PMU ファームウェアによって拒否されます。

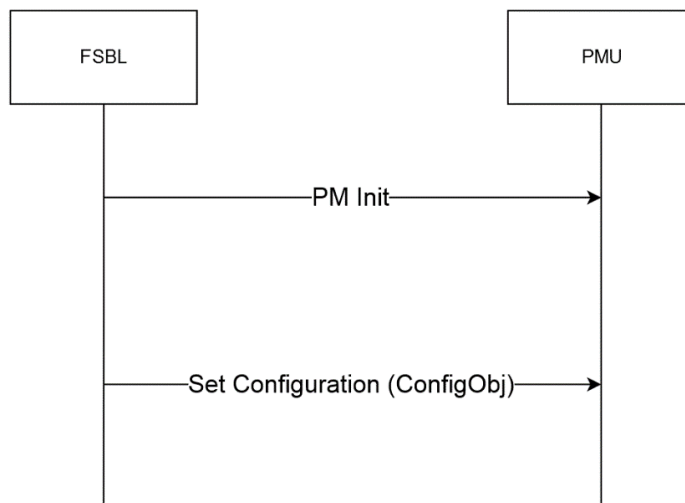


図 10-6: ブート時の最初のコンフィギュレーション

PMU ファームウェアのロード方法

PMU ファームウェアは、FSBL または CSU bootROM (CBR) のいずれかによってロードできます。ザイリンクスはどちらのフローもサポートしています。FSBL を使用して PMU ファームウェアをロードすると、次の利点があります。

- ビットストリームの後に PMU ファームウェアをロードする場合、ブート時間の短縮が期待できる。
- 2 つの BIN ファイル (安定版とアップグレード可能版) が必要な場合、FSBL でアップグレード可能なイメージに PMU ファームウェアを含めることができる。



重要:

FSBL は CBR がロードします。つまり、CBR が PMU ファームウェアもロードする場合、FSBL と PMU ファームウェアの両方のセキュア ヘッダーが同じキー /IV ペアで復号化されることになります。セキュリティの原則として、2 つのパーティションが同じキー /IV ペアを使用することは禁止されているため、これはセキュリティ脆弱性となります。この問題は、CBR ではなく FSBL で対処します。したがって、復号化を使用するセキュアなケースでは、CBR で PMU ファームウェアをロードすることは避けてください。

ウォーム リスタートで DDR セルフリフレッシュを実行する場合、ビットストリームなどのイメージよりも先に FSBL および PMU ファームウェアをロードする必要があります。

パワーオフ サスペンドの場合は、FSBL よりも先に CSU が PMU ファームウェアをロードする必要があります。

JTAG ブート モードでの PMU ファームウェアのロード

2017.1 リリース以降では、PMU の動作には FSBL でコンフィギュレーション オブジェクトをロードすることが必須条件となっています。このため、JTAG ブート モードでは FSBL をロードする前に PMU ファームウェアをロードする必要があります。デバイス ブート モードでは、PMU ファームウェアを CBR でロードする場合も FSBL でロードする場合も、コンフィギュレーション オブジェクトは FSBL で PMU ファームウェアにロードされます。

JTAG モードでブートするには、次の手順を実行します。

1. セキュリティ ゲートを無効にして、PMU MicroBlaze を表示します。PMU MicroBlaze は、シリコン v3.0 以降の xsdb では表示されません。
2. PMU ファームウェアをロードして実行します。
3. FSBL をロードして実行します。
4. U-Boot/Linux/ユーザー アプリケーションをロードして実行します。

次に、完全な Tcl スクリプトを示します。

```
#Disable Security gates to view PMU MB target
targets -set -filter {name =~ "PSU"}

#By default, JTAGsecurity gates are enabled
#This disables security gates for DAP, PLTAP and PMU.
mwr 0xffca0038 0x1fff
after 500

#Load and run PMU FW
targets -set -filter {name =~ "MicroBlaze PMU"}
```

```
dow xpfw.elf
con
after 500

#Reset A53, load and run FSBL
targets -set -filter {name =~ "Cortex-A53 #0"}
rst -processor
dow fsbl_a53.elf
con

#Give FSBL time to run
after 5000
stop

#Other SW...
dow u-boot.elf
dow bl31.elf
con

#Loading bitstream to PL
Targets -set -nocase -filter {name =~ "*PL*"}
fpga download.bit
```

JTAG ブート モード以外での PMU ファームウェアのロード

1.0 シリコンでは、JTAG ブート モード以外で PMU ファームウェアをロードすると、開始時に「Error: Unhandled IPI received」というエラー メッセージが PMU ファームウェアのログに記録されることがありますが、これは無視してかまいません。これは、IPI0 ISR が PMU ROM によってクリアされていないために発生する現象です。この問題は、バージョン 2.0 以降のシリコンでは修正されています。

FSBL を使用して PMU ファームウェアをロードする方法

1. SDK で PMU ファームウェア アプリケーションをビルドします。
2. SDK で A53 用の FSBL をビルドします (R5 も使用可能)。
3. A53 用のサンプルとして `hello_world` を作成します。
4. [Xilinx] → [Create Boot Image] をクリックします。
5. 新規 BIF ファイルを作成します。次のように選択します。
 - a. [Architecture]: ZynqMP
 - b. デフォルトでは、A53 fsbl と `hello_world` のサンプルがパーティションに表示されます。また、PMU ファームウェアも必要です。
 - c. [Add] をクリックして、`pmufw.elf` のパスを指定します。[Partition type] で [datafile] を選択し、[Destination Device] で [PS] を選択し、[Destination CPU] で [PMU] を選択します。
 - d. [OK] をクリックします。

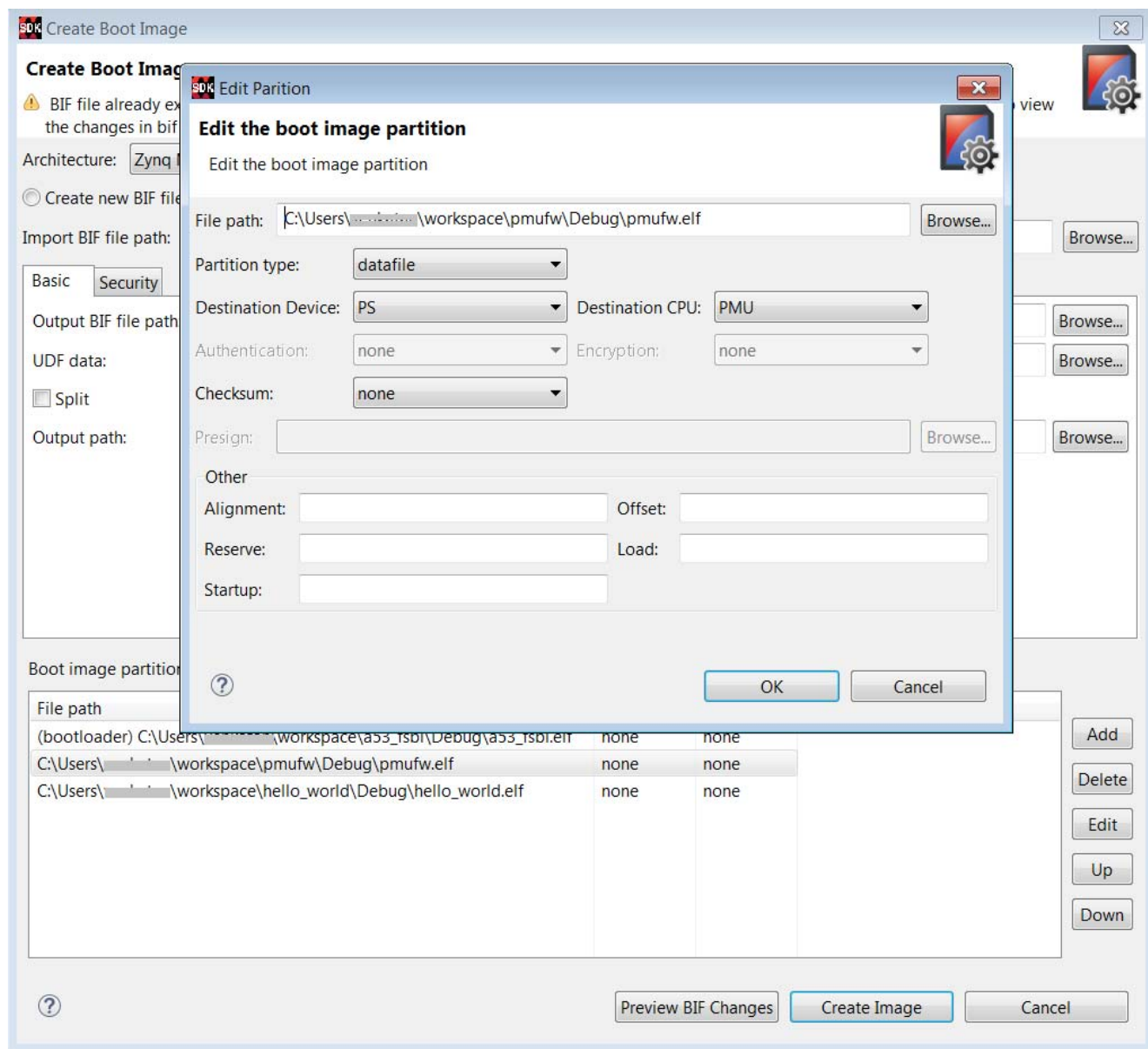


図 10-7: ブート イメージの生成

6. pmufw をパーティションとして追加します。pmufw のパスをクリックして [UP] をクリックします。
7. 次の順にパーティションを選択します。
 - a. A53 FSBL
 - b. PMU ファームウェア
 - c. Hello World アプリケーション
8. [Create Image] をクリックします。サンプルプロジェクトの bootimage フォルダに BOOT.bin が生成されます。
9. BIF ファイルを開いて、パーティション順を確認します。

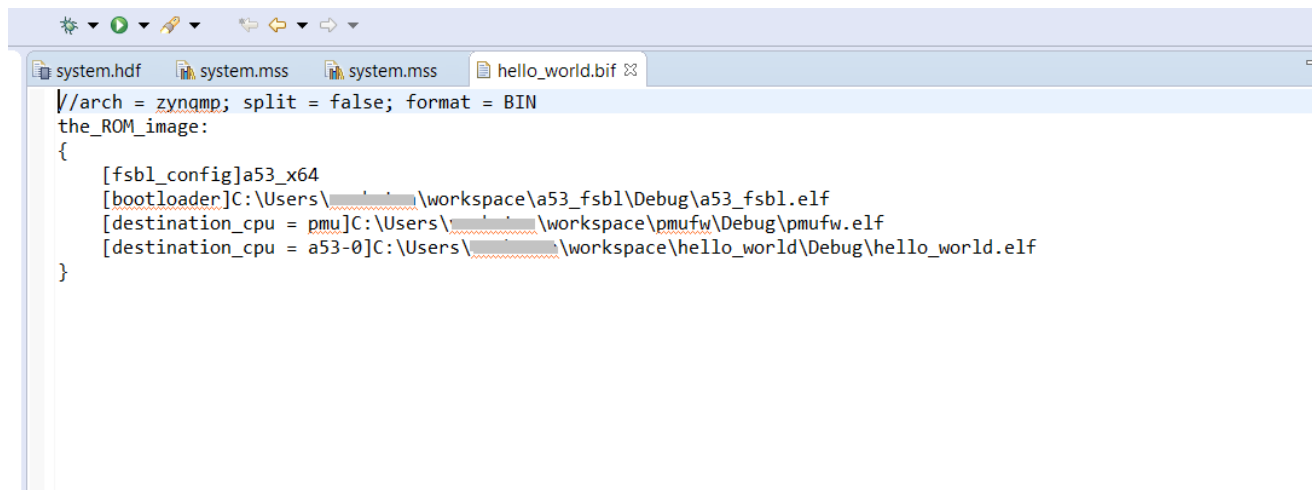


図 10-8: BIF ファイル

10. この BOOT.bin を SD カードにコピーします。
11. ZCU102 ボードを SD ブート モードで起動します。fsbl -> pmufw -> hello_world の順に表示されます。

CBR を使用して PMU ファームウェアをロードする方法

CBR による PMU ファームウェアのロードは、FSBL よりも前に実行されます。このため、MIO、クロックおよびその他の初期化はこの時点では完了していません。したがって、FSBL の実行前は PMU ファームウェアのバナーやその他のプリントが表示されません。FSBL の実行が完了すると、PMU ファームウェア プリントは通常どおり表示されます。

CBR を使用して PMU ファームウェアをロードするには、次の手順を実行します。

1. BOOT.bin のブート パーティションを変更します。
2. 「JTAG ブート モード以外での PMU ファームウェアのロード」に記載の手順を実行します。
3. 新規 BIF ファイルを作成します。次のように選択します。
 - a. [Architecture]: ZynqMP
 - b. デフォルトでは、A53 fsbl と hello_world のサンプルがパーティションに表示されます。また、PMU ファームウェアも必要です。
 - c. [Add] をクリックして、pmufw.elf のパスを指定します。[Partition type] で [pmu] を選択します (bootROM によってロード)。
 - d. [OK] をクリックします。

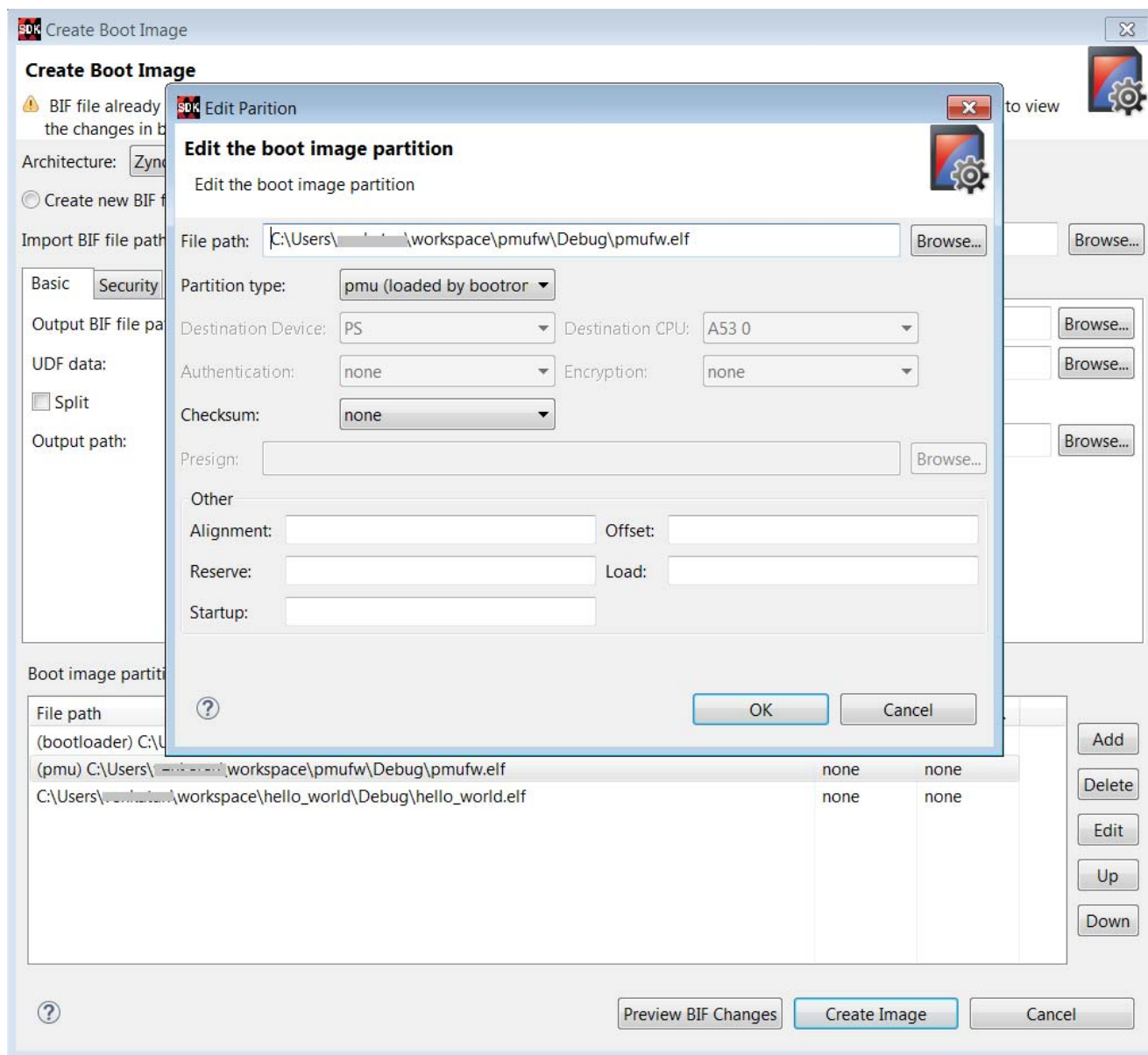


図 10-9: ブート イメージ パーティションの生成

- e. [Create Image] をクリックします。サンプルプロジェクトの bootimage フォルダに BOOT.bin が生成されます。
- f. BIF ファイルを開いてパーティション順を確認することもできます。

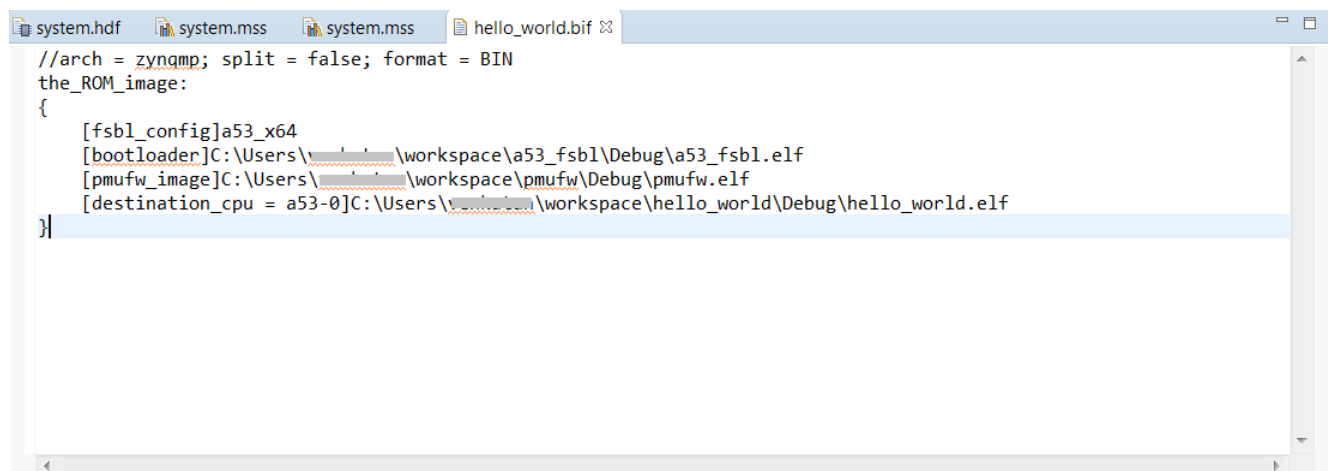


図 10-10: BIF ファイルの確認

- g. この BOOT.bin を SD カードにコピーします。
- h. ZCU102 ボードを SD ブート モードで起動します。pmufw -> fsbl -> hello_world の順に表示されます。

PMU ファームウェアの使用法

このセクションでは、例を挙げながら PMU ファームウェアの使用法について説明します。

モジュールの有効/無効化

このセクションでは、SDK および PetaLinux の両方で PMU ファームウェアのビルド フラグを有効/無効にする方法について説明します。

SDK の場合

1. PM モジュールのビルド フラグは、<pmu firmware application>/src/xpfw_config.h ファイルで定義されます。このファイルを開いて、PMU ファームウェア コードの include options セクションを変更すると、ビルド フラグを有効/無効にできます。
2. その他すべてのビルド フラグを有効にするには、次の手順を実行します。
 - a. PMU ファームウェア プロジェクトを右クリックして、[C/C++ Build Settings] をクリックします。[Properties] ウィンドウが表示されます。
 - b. [C/C++ Build] → [Settings] をクリックします。[Tool Settings] タブの [Microblaze gcc compiler] で [Symbols] をクリックします。
 - c. [Defined symbols (-D)] の下にある [Add] をクリックします。
 - d. 定義するフラグ名を入力して [OK] をクリックします。

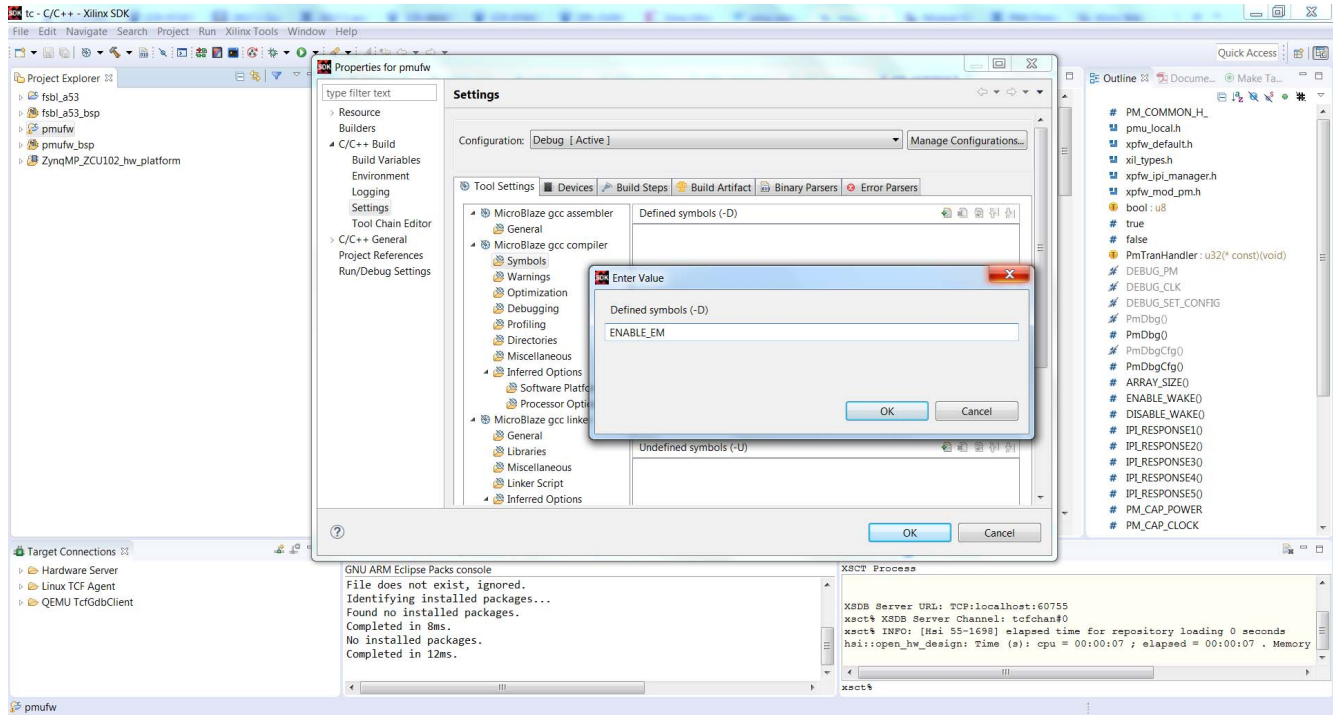


図 10-11: SDK でのビルド フラグの有効化

- e. [Properties] ウィンドウでもう一度 [OK] をクリックします。これで、新しく定義したコンパイルフラグを使用してアプリケーションのビルドが開始します。

PetaLinux の場合

1. PetaLinux プロジェクトを作成します。
2. `<plnx-project-root>/project-spec/meta-user/recipes-bsp/pmu/pmu-firmware_%.bbappend` ファイルを開き、次の行を追加します。

```
YAML_COMPILER_FLAGS_append = -DENABLE_EM
```

この行により、EM モジュールが有効になります。フラグを有効にするには、接頭辞「-D」を付けます。

3. YAML コンパイラフラグを変更した場合は、アプリケーションを再ビルドする前にクリーンステートを強制してください。

カスタム モジュールの使用

ある特定の機能を実行するユーザー定義ルーチンセットは、PMU ファームウェアのモジュールとして設計することを推奨します。これらのファイルは自己完結している必要があります。ただし、これらのファイルは `xpfw_core.h` からの宣言を使用できます。各モジュールには、次のインターフェイスを登録できます。モジュールで不要なハンドラーは登録をスキップできます。

1. Config ハンドラー: 初期化中に呼び出されます。
2. イベント ハンドラー: 登録したイベントがトリガーされると呼び出されます。
3. IPI ハンドラー: 登録した IPI ID を持つ IPI メッセージを受信すると呼び出されます。

カスタム モジュールの作成

カスタム モジュールを作成するには、PMU ファームウェアに次のコードを追加します。

```
/* IPI Handler */
static void CustomIpiHandler(const XPfw_Module_t *ModPtr, u32 IpiNum, u32 SrcMask,
const u32* Payload, u8 Len)
{
    /**
     * Code to handle the IPI message received
     */
}

/* CfgInit Handler */
static void CustomCfgInit(const XPfw_Module_t *ModPtr, const u32 *CfgData,
u32 Len)
{
    /**
     * Code to configure the module, register for events or add scheduler tasks
     */
}

/* Event Handler */
static void CustomEventHandler(const XPfw_Module_t *ModPtr, u32 EventId)
{
    /**
     * Code to handle the events received
     */
}

/*
 * Create a Mod and assign the Handlers. We will call this function
 * from XPfw_UserStartup()
 */
void ModCustomInit(void)
{
    const XPfw_Module_t *CustomModPtr = XPfw_CoreCreateMod();

    (void) XPfw_CoreSetCfgHandler(CustomModPtr, CustomCfgInit);
    (void) XPfw_CoreSetEventHandler(CustomModPtr, CustomEventHandler);
    (void) XPfw_CoreSetIpiHandler(CustomModPtr, CustomIpiHandler, (u16) IPI_ID);
}
```

イベントの登録

PMU に送信されるすべての割り込みは、ユーザーからは `xpfw_events.h` で定義した `EVENTID` を持つイベントとして見ることができます。イベントは、通常 `CfgHandler` を使用して任意のモジュールに登録できます。イベントがトリガーされると、モジュールの `EventHandler` が呼び出されます。

RTC イベントを登録する場合

```
Status = XPfw_CoreRegisterEvent (ModPtr, XPFW_EV_RTC_SECONDS);
```


EventHandler の例

```
void RtcEventHandler(const XPfw_Module_t *ModPtr, u32 EventId)
{
    xil_printf("MOD%d:EVENTID: %d\r\n", ModPtr->ModId, EventId);
    if(XPFW_EV_RTC_SECONDS == EventId){
        /* Ack the Int in RTC Module */
        Xil_Out32(RTC_RTC_INT_STATUS,1U);
        xil_printf("RTC: %d \r\n", Xil_In32(RTC_CURRENT_TIME));
    }
}
```

エラー管理の使用方法

このセクションでは、PMU ファームウェアが受信するエラー (システム内で発生し、PMU MB にマップされたエラー) に対するエラー アクションを EM モジュールを使用して設定する方法について説明します。

エラー管理の例 (カスタム ハンドラー)

ここでは、OCM の訂正不能エラー (EM_ERR_ID_OCM_ECC) を例として考えます。PMU ファームウェアで、このエラーに対するカスタム ハンドラーを登録します。このハンドラーは、エラー メッセージをプリントアウトするだけのものです。実際には、破損したメモリの内容を再度読み込む必要がありますが、この例では単にエラーをクリアしてメッセージを出力するだけにとどめています。

OCM の訂正不能エラーに対するエラー ハンドラーを PMU ファームウェアに追加します。

```
+++ b/lib/sw_apps/zynqmp_pmufw/src/xpaw_mod_em.c
@@ -140,6 +140,14 @@ void FpdSwdtHandler(u8 ErrorId)
    XPfw_RecoveryHandler(ErrorId);
}

+/* OCM Uncorrectable Error Handler */
+static void OcmErrHandler(u8 ErrorId)
+{
+    XPfw_Printf(DEBUG_DETAILED, "EM: OCM ECC error detected\n");
+    /* Clear the Error Status in OCM registers */
+    XPfw_Write32(0xFF960004, 0x80);
+}
+
+/* CfgInit Handler */
+static void EmCfgInit(const XPfw_Module_t *ModPtr, const u32 *CfgData,
+    u32 Len)
@@ -162,6 +170,8 @@ static void EmCfgInit(const XPfw_Module_t *ModPtr, const u32
*CfgData,
    }
}

+    XPfw_EmSetAction(EM_ERR_ID_OCM_ECC, EM_ACTION_CUSTOM, OcmErrHandler);
+

    if (XPfw_RecoveryInit() == XST_SUCCESS) {
        /* This is to enable FPD WDT and enable recovery mechanism when
```

デバッガー (xsdb) を使用して、OCM の訂正不能 ECC エラーを挿入します。

```

;# Enable ECC UE interrupt in OCM_IEN
mwr -force 0xFF96000C [expr 1<<7]

;# Write to Fault Injection Data 0 Register OCM_FI_D0
;# Errors will be injected in the bits which are set, here its bit0, bit1
mwr -force 0xFF96004C 3

;# Enable ECC and Fault Injection
mwr -force 0xFF960014 1
;
;# Clear the Count Register : OCM_FI_CNTR
mwr -force 0xFF960074 0
;# Now write data to OCM for the fault to be injected
;# Since OCM does a RMW for 32-bit transactions, it should trigger error here
mwr -force 0xFFFE0000 0x1234

;# Read back to trigger error again
mrd -force 0xFFFE0000

```

エラー管理の例 (エラーへの応答として POR を使用)

致命的なエラーの場合は、システムをエラーから回復することが困難で、システム全体に対するリセットが必要になることがあります。このような場合は、エラーアクションとして **POR** または **SRST** を使用します。この例では、OCM ECC ダブルビット エラーに対する応答として **POR** リセットを使用しています。

次のコードで、**POR** をエラー アクションとして設定します。

```

@@ -162,6 +162,8 @@ static void EmCfgInit(const XPfw_Module_t *ModPtr, const u32
 *CfgData,
        }
    }

+    XPfw_EmSetAction(EM_ERR_ID_OCM_ECC, EM_ACTION_POR, NULL);
+

    if (XPfw_RecoveryInit() == XST_SUCCESS) {
        /* This is to enable FPD WDT and enable recovery mechanism when

```

OCM ECC エラーを挿入する Tcl スクリプトは、上に示した例と同じです。エラーをトリガーすると **POR** が発生し、すべてのプロセッサが通常のパワーオン ステートと同じリセット ステートになります。PMU RAM も **POR** によってクリアされます。このため、PMU ファームウェアをもう一度読み込む必要があります。

エラー管理の例 (エラーに対して PS_ERROR_OUT を使用)

エラー発生時にシステム外部に通知する必要がある場合は、そのエラーに PS_ERROR_OUT 応答を設定します。これにより、そのエラーが発生すると外部に伝搬し、PS_ERROUT 信号の LED が点灯します。この例では、OCM ECC ダブルビットエラーに対する応答として PS_ERROR_OUT を使用しています。

次のコードで、PS_ERROR_OUT をエラーアクションとして設定します。

```
@@ -162,6 +162,8 @@ static void EmCfgInit(const XPfw_Module_t *ModPtr, const u32
    *CfgData,
    }

+    XPfw_EmSetAction(EM_ERR_ID_OCM_ECC, EM_ACTION_PSERR, NULL);
+

    if (XPfw_RecoveryInit() == XST_SUCCESS) {
        /* This is to enable FPD WDT and enable recovery mechanism when
```

OCM ECC エラーを挿入する Tcl スクリプトは、上に示した例と同じです。エラーをトリガーすると、ボードの PS_ERROUT LED が点灯します。

IPI メッセージングの使用方法

このセクションでは、PMU ファームウェアから RPU0、および RPU0 から PMU ファームウェアへの IPI メッセージの使用方法について説明します。IPI ドライバーの初期化時に、PMU ファームウェアは IPI チャンネルが割り当てられたマスターからの IPI 割り込みを有効にします。

PMU ファームウェアから RPU0

詳細は、「[Zynq UltraScale Plus MPSoC - IPI Messaging Example](#)」の Wiki ページ (英語) を参照してください。

注記: この例を実行するには、PMU ファームウェアの EM モジュールを有効にしておく必要があります。

RPU0 から PMU ファームウェア

RPU から PMU への IPI メッセージの例は、「[Zynq UltraScale Plus MPSoC - IPI Messaging Example](#)」の Wiki ページ (英語) を参照してください。



重要: この Wiki ページの例は、PMU と RPU0 の間で IPI をトリガーする方法を示しているため、APU または RPU1 との間で IPI をトリガーするには、デスティネーション CPU マスクを目的のマスターに合わせて変更する必要があります。

スケジューラへのタスクの追加

タスクは引数なし (void)、戻り値なし (void) の関数です。現在の PMU ファームウェアには、事前に定義した時間内にタスクが完了したことを確認する手段がありません。このため、タスク設計時に何らかの手段を確保しておく必要があります。ここで、次のメッセージを出力するタスクを考えてみます。

```
void TaskPrintMsg(void)
{
    xil_printf("Task has been triggered\r\n");
}
```

このタスクが 500ms ごとに発生するようにスケジュールする場合、次のコードを使用します。TaskModPtr は、タスクをスケジュールするモジュールへのポインターです。

```
Status = XPfw_CoreScheduleTask(TaskModPtr, 500U, TaskPrintMsg);
if(XST_SUCCESS == Status) {
    xil_printf("Task has been added successfully !\r\n");
}
else {
    xil_printf("Error: Failed to add Task !\r\n");
}
```

RPU からの FPD ロック ステータスの読み出し

レジスタ 0xFFD600F0 は PMU ファームウェアのローカルレジスタで、ビット 31 は FPD がロックしているかどうかを示します。ビット 31 が 1 にセットされている場合、FPD はロックしています。これは、POR がアサートされるまで分離されたままです。FPD のロック ステータスは、PMU ファームウェアを使用してこのレジスタを読み出すことで確認できます。そのためには、PMU ファームウェアに対する MMIO 読み出し関数を呼び出します。R5 から FPD ロック ステータスを読み出すには、次の手順を実行します。

1. R5 プロセッサ用の空のアプリケーションを作成します。BSP の設定で xilpm ライブラリを有効にします。
2. プロジェクトで新規 .c ファイルを作成し、次のコードを追加します。

```
#include "xipipsu.h"
#include "pm_api_sys.h"
#define IPI_DEVICE_ID XPAR_XIPIPSU_0_DEVICE_ID
#define IPI_PMU_PM_INT_MASKXPAR_XIPIPS_TARGET_PSU_PMU_0_CH0_MASK

#define MMIO_READ_API_ID20U
#define FPD_LOCK_STATUS_REG0xFFD600F0

int main(void)
{
    XIpiPsu IpiInstance;
    XIpiPsu_Config *Config;
    s32 Status;
    u32 Value;

    /* Initialize IPI peripheral */
    Config = XIpiPsu_LookupConfig(IPI_DEVICE_ID);
    if (Config == NULL) {
        xil_printf("Config Null\r\n");
        goto END;
    }

    Status = XIpiPsu_CfgInitialize(&IpiInstance, Config,
```

```

        Config->BaseAddress);
if (0x0U != Status) {
    xil_printf("Config init failed\r\n");
    goto END;
}

/* Initialize the XilPM library */
Status = XPm_InitXilpm(&IpiInstance);
if (0x0U != Status) {
    xil_printf("XilPM init failed\r\n");
    goto END;
}
/* Read using XPm_MmioRead() */
Status = XPm_MmioRead(FPD_LOCK_STATUS_REG, &Value);
if (0x0U != Status)
{
    xil_printf("XilPM MMIO Read failed\r\n");
    goto END;
}
xil_printf("Value read from 0x%x: 0x%x\r\n", FPD_LOCK_STATUS_REG, Value);

END:
    xil_printf("Exit from main\r\n");
}

```

注記: このアプリケーションは、FSBL (の実行) が正しく完了した後に実行する必要があります。FSBL が PMU ファームウェアへコンフィギュレーション オブジェクトを正しく送信できていなければ、このアプリケーションは動作しません。

PMU ファームウェアのデバッグ

PMU ファームウェアは、デフォルトで -Os および LTO 最適化を使用してビルドされます。アプリケーションをデバッグするには、最適化を無効にする必要があります。

PMU ファームウェアの最適化を無効にする手順

最適化を無効にするとコード サイズが増大し、PMU ファームウェアをビルドできなくなることがあるため、使用しない機能はプロジェクト設定からビルド フラグを削除して無効にしてください。PMU ファームウェアの xpfw_config.h ファイルでは、いくつかの機能が無効になっています。

最適化を無効化/変更するには、[Properties for pmu_fw] ダイアログ ボックスの [Miscellaneous] を開き、[図 10-12](#) にハイライトで示したオプション (-Os -flto -ffat -lto -objects) を削除/変更します。

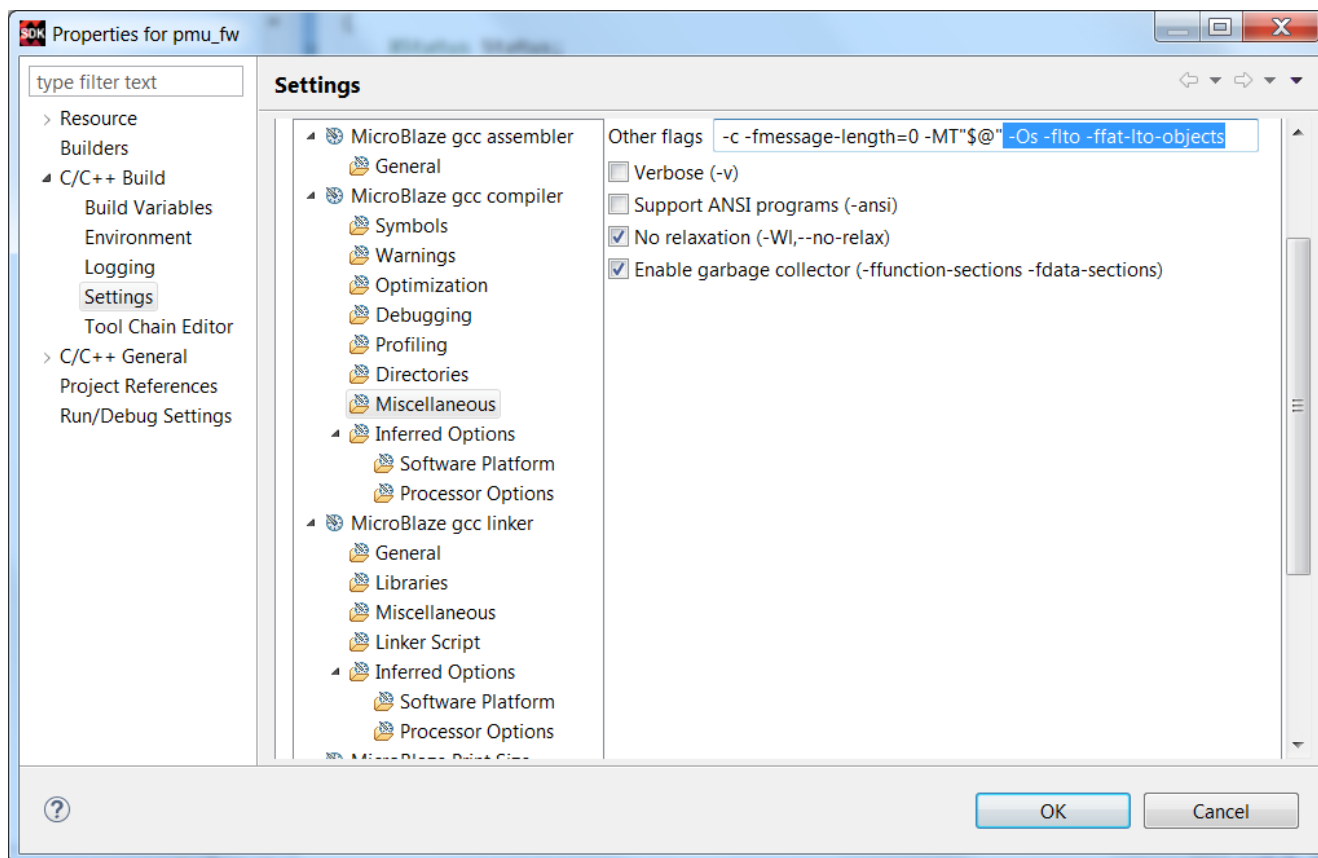


図 10-12: オプションの変更



重要: SDK のコンパイラ設定の [Optimization] には、[Miscellaneous] で変更した最適化オプションが反映されません。[Optimization] の表示は「-O0」のままで (図 10-13 参照)、ユーザーには紛らわしい表示となっています。

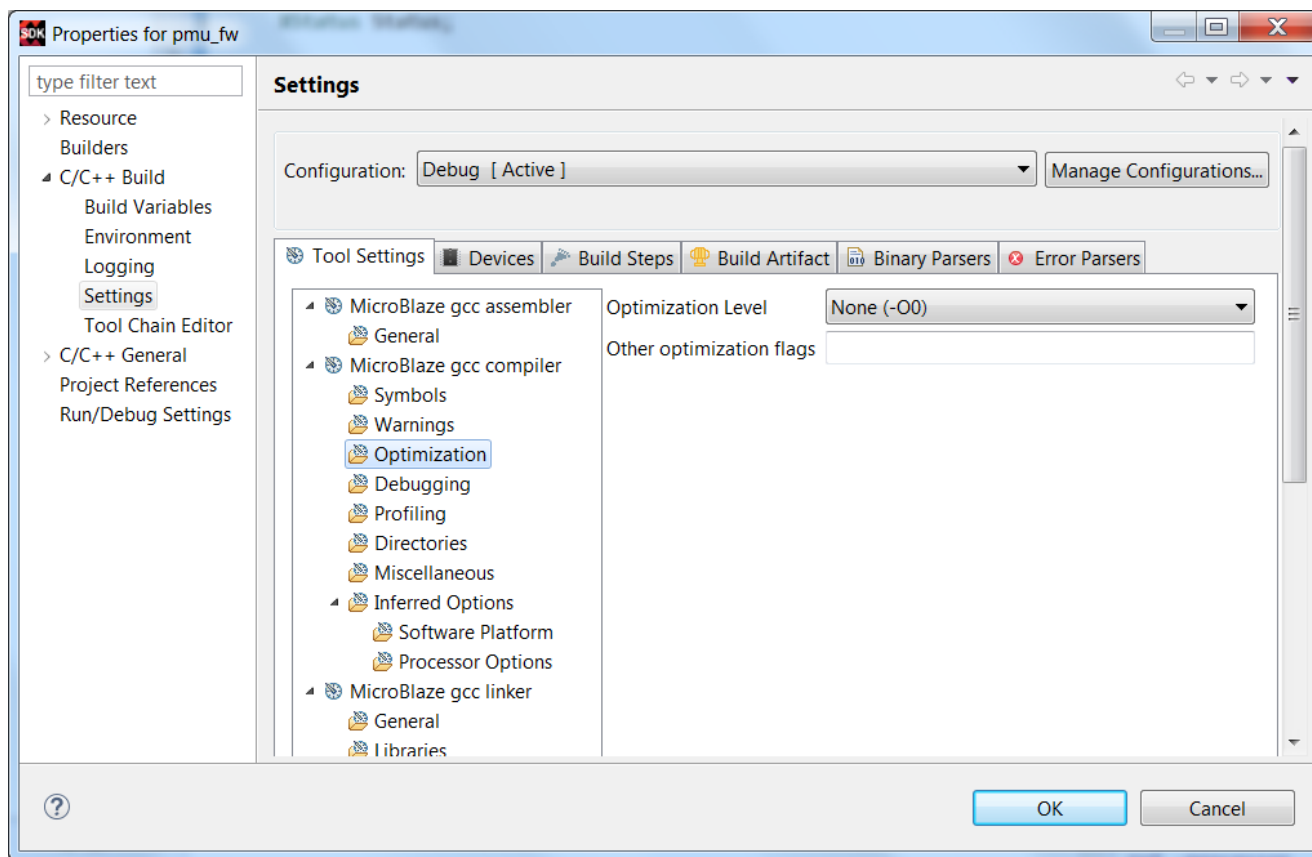


図 10-13: 最適化レベルの設定

SDK を使用した PMU ファームウェア アプリケーションのデバッグ手順

1. アプリケーションを右クリックして、[Debug As] をクリックし、[Debug Configurations] をクリックします。
2. [System Debugger] を右クリックして、[New] をクリックします。新しいコンフィギュレーションが作成されます。[Debug] をクリックします。

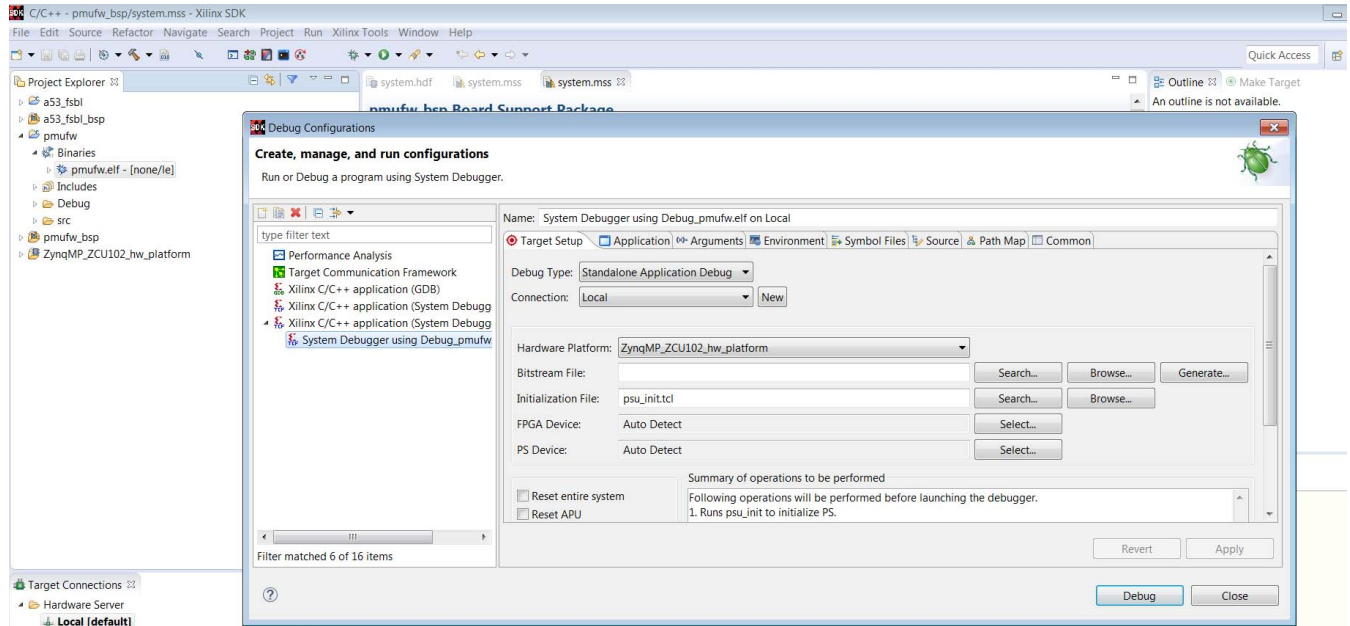


図 10-14: SDK を使用した PMU ファームウェアのデバッグ

3. [Debug] パースペクティブを選択します。[OK] をクリックします。[Confirm Perspective Switch] ダイアログ ボックスで [Yes] をクリックします。

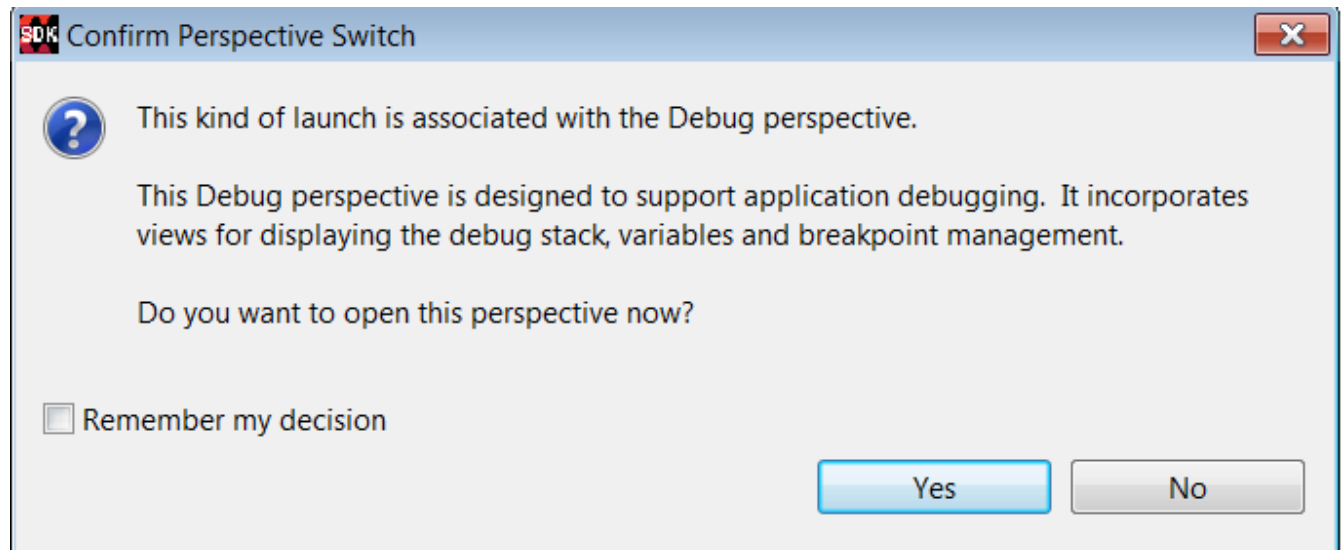


図 10-15: [Confirm Perspective Switch] ダイアログ ボックス

4. [Debug] パースペクティブが表示され、PMU ファームウェアが実行されます。

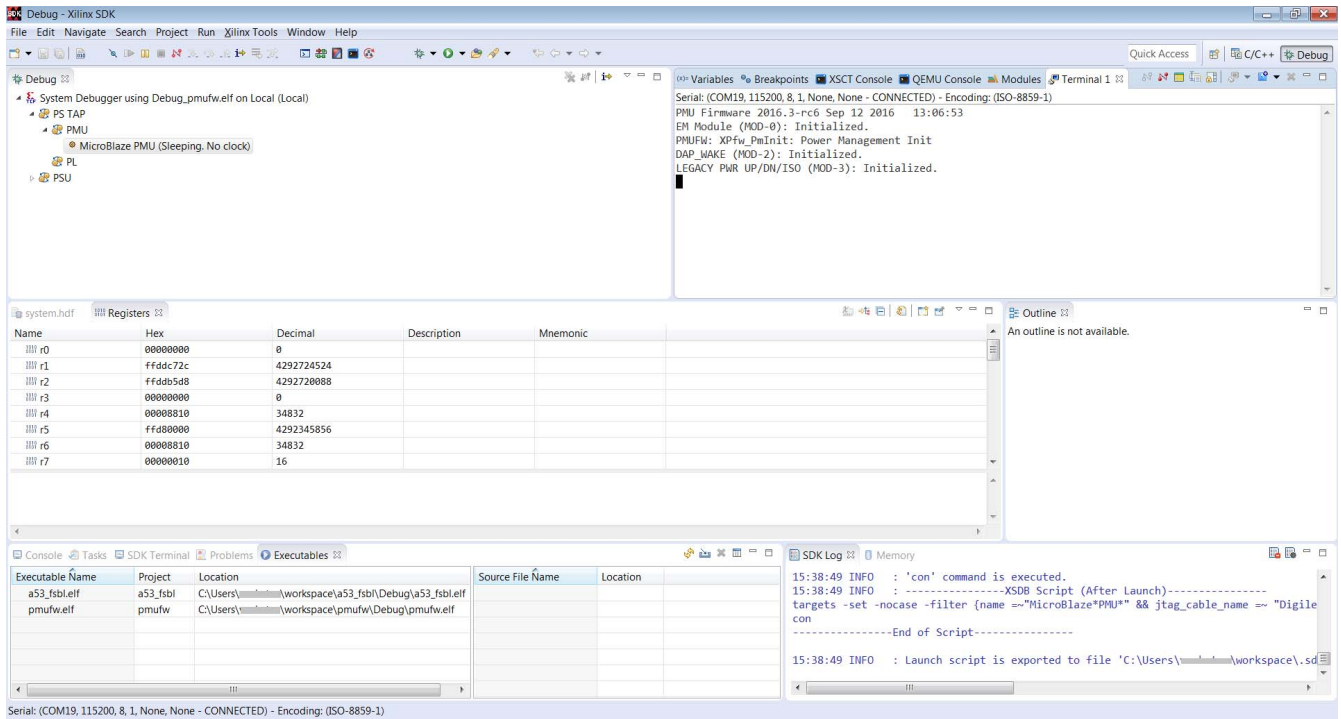


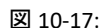
図 10-16: PMU ファームウェアの [Debug] パースペクティブ

5. フロー制御のためのブレークポイントを挿入した後、再度実行してデバッグします。

PMU ファームウェア ログの転送

PMU ファームウェアのログを別の UART に転送するには、次の手順を実行します。

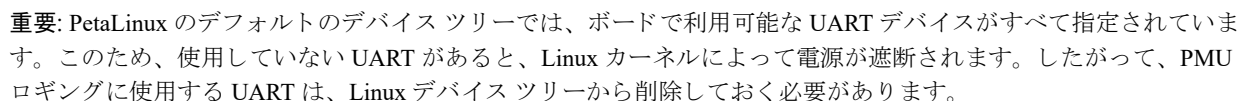
1. PMU ファームウェア アプリケーションを作成したら、[PMU firmware BSP] を右クリックして、[Board Support Package Settings] をクリックします。
2. [Board Support Package Settings] ウィンドウが表示されます。[Overview] の下にある [Standalone] をクリックします。
3. [stdin] と [stdout] の [Value] 欄で、ログの転送先となる UART をドロップダウン リストから選択します。デフォルトでは、[psu_uart_0] が選択されています。



- 注記: ハードウェア デザインに UART が 1 つしかない場合は、複数のターゲットで共有する必要があります。

PMU ロギングには、次のコンソールポートを使用できます。

- PMU ロギングを有効にした場合、PMU がロギングに使用する UART の無効化、クロック ゲーティングの適用、パワーダウンはできません。これらは、PMU がハングする原因となります。この結果、システム全体がハングします。また、PMU ロギングを有効にすると、ディープ スリープ時に UART の PLL およびクロックが無効にならないため、消費電力が増大します。



PMU ロギングに使用している UART の電源ステートが、RPU 上で動作するスタンドアロン アプリケーションによって変更されないように注意する必要があります。

デフォルトでは、PMU ロギングは無効です。PMU ロギングに使用している UART が、APU または RPU の影響を受けないように注意する必要があります。

PMU ファームウェアのメモリレイアウトとフットプリント

このセクションでは、PMU ファームウェアのメモリ レイアウトの詳細と、各種モジュールを有効にした場合のメモリ フットプリントについて説明します。

PMU RAM の一部は PBR 用の予約領域で、約 125.7KB が PMU ファームウェア用です。図 10-18 に、PMU RAM のメモリ レイアウトを示します。

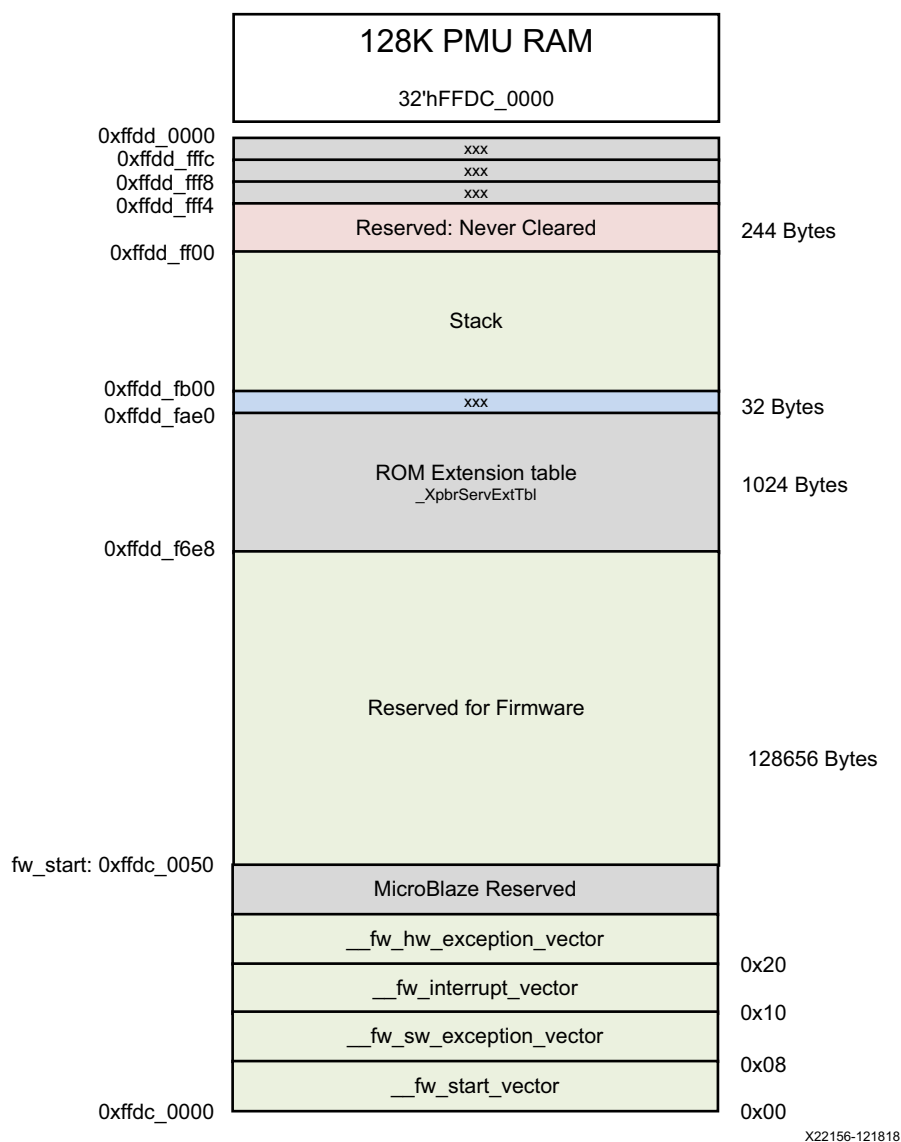


図 10-18: PMU ファームウェアのメモリ レイアウト

PMU ファームウェアのうち、デフォルトで有効なのはベース ファームウェアと PM モジュールのみで、それ以外のモジュールはすべて無効です。モジュールのデフォルト設定は、「[PMU ファームウェアのビルド フラグ](#)」を参照してください。

注記: ここに記載したメトリクスは、コンパイル オプションでサイズ優先の最適化 (-Os) を指定した場合のものです。この最適化設定は、SDK のデフォルトで有効になっています。無効にするには、「[PMU ファームウェアの最適化を無効にする手順](#)」の手順に従ってください。

表 10-24: PMU ファームウェアのメトリクス

S.No	機能/コンポーネント	占有サイズ (KB)	空き領域 (KB)	説明	備考
1	詳細なデバッグプリントを無効にした PMU ファームウェア	112.5	15.5	PMU ベース ファームウェアと PM モジュールを使用した場合です。	
2	詳細なデバッグプリントを有効にした PMU ファームウェア	123	5	詳細なデバッグプリントは、XPFW_DEBUG_DETAILED および DEBUG_MODE フラグを定義して有効にします。	(1) と (2) を足した概算値です。
3	エラー管理モジュールを有効にした PMU ファームウェア	115.4	12.6	エラー管理モジュールは、ENABLE_EM および ENABLE_SCHEDULER フラグを定義して有効にします。	(1) と (3) を足した概算値です。
4	リスタート機能を有効にした PMU ファームウェア	117.5	10.5	リスタート機能は、ENABLE_RECOVERY、ENABLE_ESCALATION、CHECK_HEALTHY_BOOT フラグ、および EMABLE_EM、ENABLE_SCHEDULER フラグを定義して有効にします。	(1) と (4) を足した概算値です。

依存関係



推奨: ソフトウェア コンポーネント (FSBL、PMU ファームウェア、ATF、U-Boot および Linux) は、すべて同じリリース タグ (2017.3 など) のものを使用することを推奨します。

電力管理フレームワーク

はじめに

Zynq® UltraScale+™ MPSoC は、複数のユーザー プログラマブル プロセッサ、FPGA、最先端の電力管理機能を統合した業界初のヘテロジニアス マルチプロセッサ SoC (MPSoC) です。

電力効率に優れた最新デザインでは、消費電力を削減するハードウェア オプションを複数備えた複雑なシステム アーキテクチャを使用し、複数のマスター デバイスからのすべての電力管理要求に対応できる特別な CPU を使用して、各リソースへのパワーアップやパワーダウンおよび電源ステートの遷移を管理する必要があります。この場合の課題は、業界規格 (IEEE P2415) に準拠し、異なるオペレーティング システムを実行する複数 CPU からの要求にすべて応えることができるインテリジェントなソフトウェア フレームワークを提供することです。ザイリンクスは、PMF (電力管理フレームワーク) を作成し、プラットフォーム管理ユニット (PMU) を使用する柔軟な電力管理制御を可能にしました。

この電力管理フレームワークは、さまざまなユース ケースをサポートします。たとえば、Linux は、アイドル、ホットプラグ、サスペンド (一時停止)、レジューム (復帰)、ウェークアップなどの基本的な電力管理機能を提供します。カーネルは API を使用して電力管理制御を実行しますが、ほとんどの RTOS にはこの機能がありません。したがって、ユーザー インプリメンテーションが必要になり、この電力管理フレームワークを使用することで簡単に実装できます。

エンベデッド ビジョン システム、先進運転支援システム、監視システム、医療用ポータブル測定機器、IoT (Internet of Things) などの産業向けアプリケーションでは、高性能なヘテロジニアス SoC へのニーズが高まっている一方で、厳しい電力バジェットが要求されます。これらのアプリケーションの中にはバッテリーで駆動されるものがあり、バッテリー寿命が重視されています。その他、クラウドやデータセンターでは、環境コストの削減はもちろんのこと、冷却コストやエネルギー コストの削減が非常に重視されています。これらすべてのアプリケーションは、柔軟な電力管理ソリューションによって大きなメリットを享受できます。

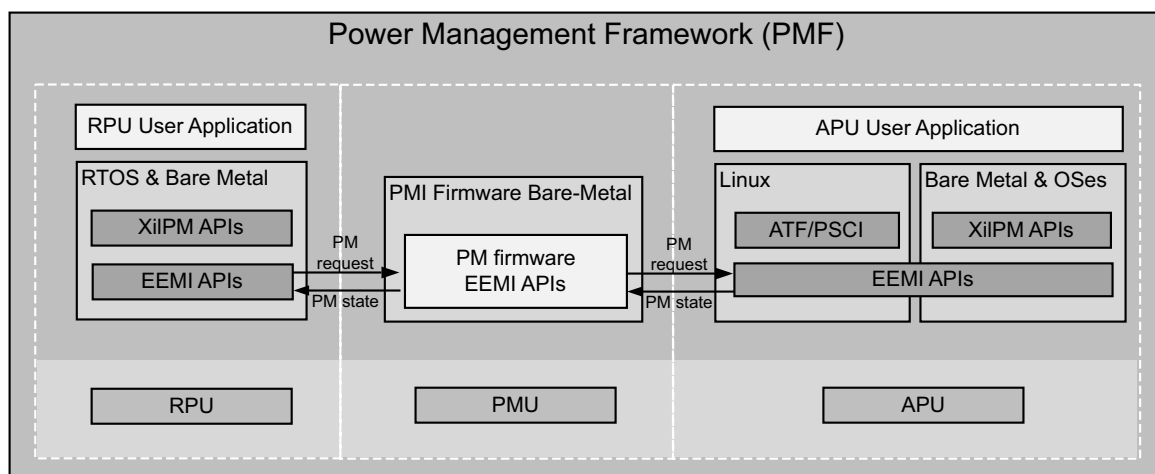
主な機能

電力管理フレームワークの主な機能は次のとおりです。

- プラットフォーム管理ユニット (PMU) の使用によって、集中管理された電源ステート情報を提供
- EEMI (Embedded Energy Management Interface) API (IEEE P2415) をサポート
- すべてのデバイスの電源ステートを監視
- Linux の電力管理機能をサポート
 - Linux デバイス ツリー電力管理
 - ATF/PSCI 電力管理サポート
 - アイドル
 - ホットプラグ
 - サスペンド
 - レジューム (復帰)
 - ウェークアップ プロセス管理
- 24 以上の API を使用して次の電力管理機能を直接制御
 - プロセッシング ユニットのサスペンドとウェークアップの管理
 - メモリとペリフェラルの管理

電力管理ソフトウェアのアーキテクチャ

Zynq UltraScale+ MPSoC アーキテクチャには、すべてのシステム リソースのパワーアップ、パワーダウン、監視、およびウェークアップ機能を制御する専用のプログラマブル ユニット、PMU を備えています。ユーザーは、複数プロセッサを統合したヘテロジニアス システムの電力管理に最適な機能を備えたシステムを利用することで、大きなメリットを享受できます。その一方で、デザインは複雑になります。電力管理フレームワークの目的は、この複雑なデザインを抽象化し、ユーザーが電力バジェットの目標を満たすために必要な API のみを提供することです。



X19504-071317

図 11-1: 電力管理フレームワーク

EEMI の目的は、すべてのソフトウェア コンポーネントがコアやペリフェラルの電力を管理できるように共通の API を提供することです。EEMI によって、複雑プロセッサや単一コアをサスペンドするなどの高度な電力管理が可能になります。この基本アーキテクチャでは、最適な省電力アプローチをユーザーが簡単に実装できるようになります。

Linux デバイス ツリーは、各デバイスに共通する記述フォーマットと電力特性を提供します。また、Linux は基本的な電力管理機能として、アイドル、ホットプラグ、サスペンド、レジューム、ウェークアップなども提供します。カーネルは、実装された API を利用して電力管理を実行します。

ユーザーは、XilPM ライブラリを使用して、24 以上の API を利用する独自の電力管理アプリケーションを作成することも可能です。

Zynq UltraScale+ MPSoC の電力管理の概要

Zynq UltraScale+ MPSoC の電力管理フレームワークは、EEMI (Extensible Energy Management Interface) の実装をベースとする電力管理のオプションです。これによって、チップやデバイス上のさまざまなプロセッシング ユニット (PU) を駆動するソフトウェア コンポーネントが電力管理のための要求を発行したり、要求に応えることが可能になります。

Zynq UltraScale+ MPSoC の電力管理ハードウェア アーキテクチャ

Zynq UltraScale+ MPSoC デバイスは、主に 4 つの電源ドメインに分割されています。

- 。 フル電力ドメイン (FPD): Arm Cortex™-A53 アプリケーションプロセッシング ユニット (APU) のほか、APU が使用する多数のペリフェラルが含まれます。
- 。 低電力ドメイン (LPD): Arm Cortex-R5F リアルタイムプロセッシング ユニット (RPU)、プラットフォーム管理ユニット (PMU)、コンフィギュレーションセキュリティ ユニット (CSU)、その他残りのオンチップペリフェラルが含まれます。
- 。 プログラマブル ロジック (PL) 電源ドメイン: PL が含まれます。
- 。 バッテリ電源ドメイン: リアルタイム クロック (RTC) およびバッテリ バックアップ式 RAM (BBRAM) が含まれます。

次の図にあるその他の電力ドメインは、電力フレームワークによって積極的に管理されません。電力ドメインに対する電力管理スイッチングをデザインで利用する場合は、一部の電源レールを分離しておく必要があります。こうすると、電力ドメイン スwitchング ロジックを使用して電源レールを個別にオフにできます。詳細は、『UltraScale アーキテクチャ PCB デザイン ユーザー ガイド』(UG583) [参照 21] の「UltraScale デバイスの PCB 電源分配システム」を参照してください。

次の図に、Zynq UltraScale+ MPSoC デバイスの電源ドメインと電源アイランドを示します。

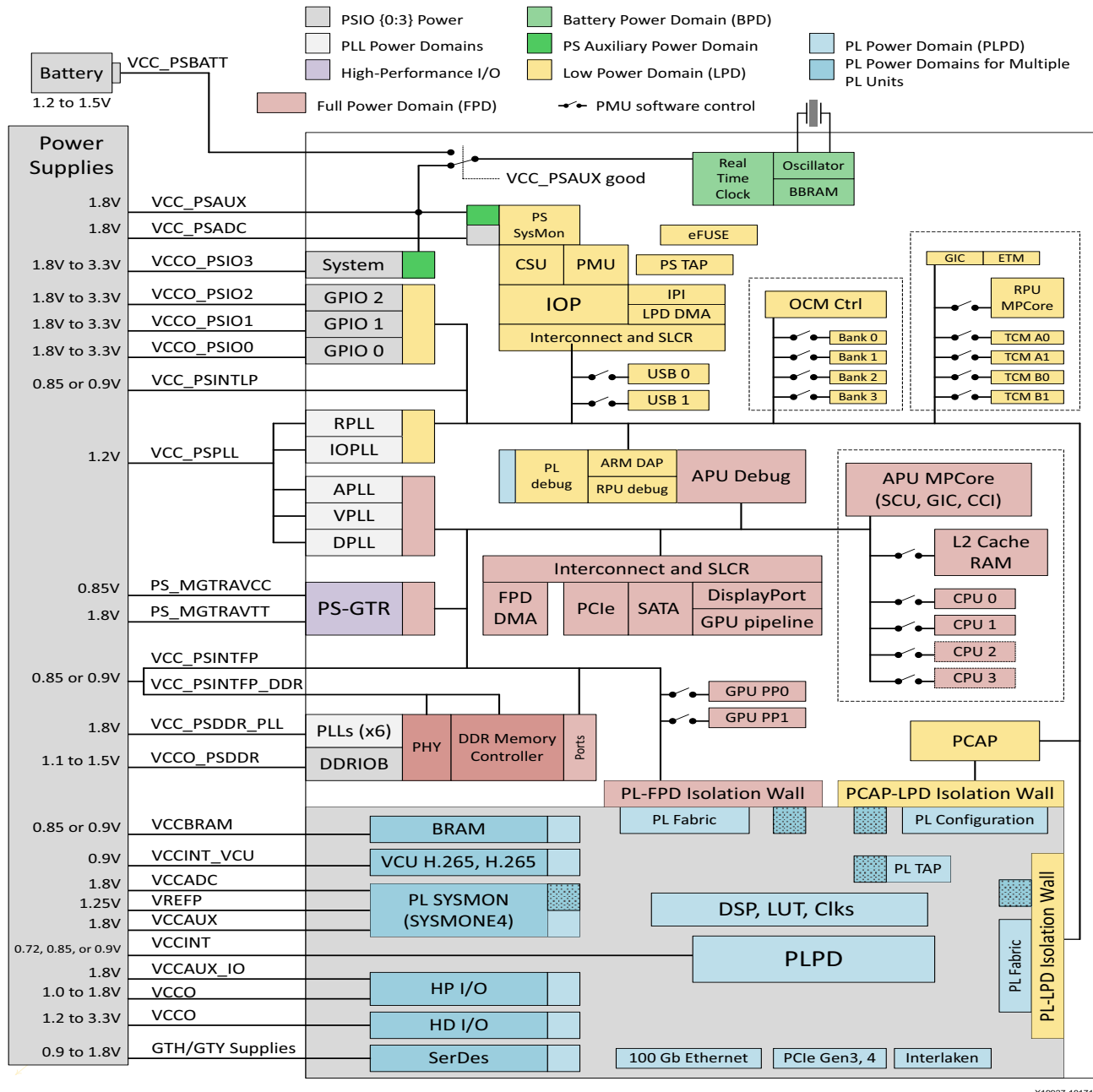


図 11-2: Zynq UltraScale+ MPSoC の電力ドメインと電源アイランド

Zynq UltraScale+ MPSoC デバイスはヘテロジニアスなマルチコア アーキテクチャであるため、単一プロセッサが個々のコンポーネントやサブシステムの電源ステートを独自に判断することは不可能です。

その代わりに、電力管理 API がすべての電力管理制御をプラットフォーム管理ユニット (PMU) に委託する協調アプローチが採用されています。これは、APU または RPU などのその他のプロセッシング ユニット (PU) から受信した電力管理要求に対応したり、電力管理 API を介してほかのプロセッシング ユニットからの要求や実行に対応する重要なコンポーネントです。



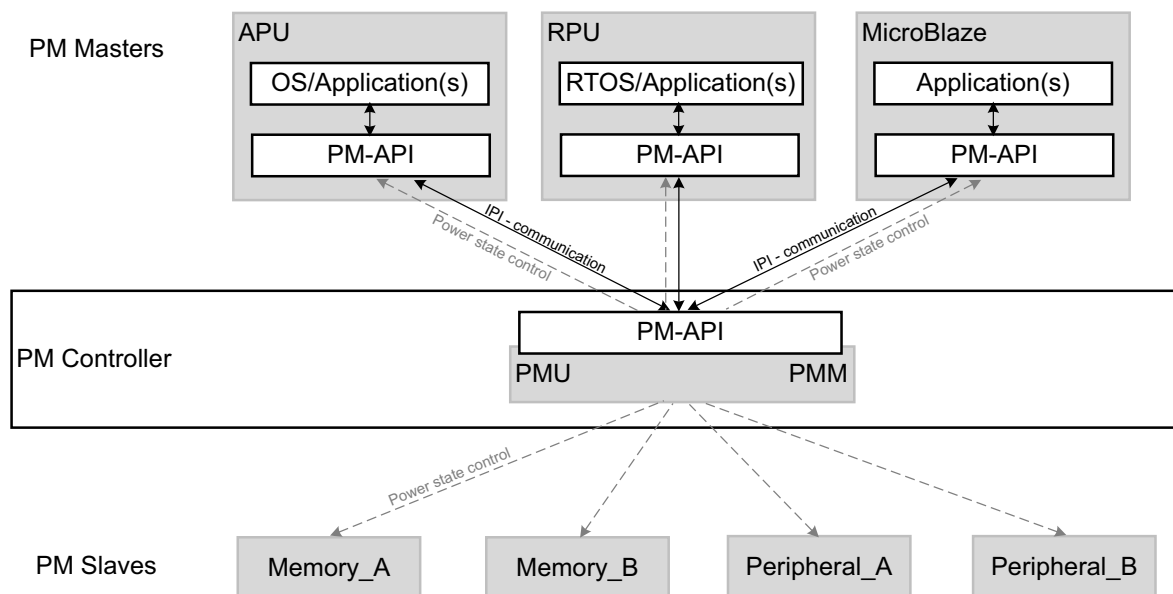
重要: Zynq UltraScale+ MPSoC の EEMI 実装の場合、プラットフォーム管理ユニット (PMU) は、APU や RPU などのプロセッシングユニット (PU) の電力管理コントローラーとして機能します。これらの APU/RPU は、電力管理 (PM) マスター ノードとして機能し、電力管理要求を発行します。これらの要求に基づいて、PMU はすべての PM スレーブ ノードおよび PM マスター ノードの電源ステートを制御します。特に記載のない限り、「PMU」と「電力管理コントローラー」は同じ意味を表します。

Zynq UltraScale+ MPSoC デバイスは、異なるプロセッサ間での電力管理関連の通信手段として使用されるプロセッサ間割り込み (IPI) もサポートしています。詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「割り込み」の章を参照してください。

Zynq UltraScale+ MPSoC の電力管理ソフトウェアアーキテクチャ

電力管理において複数のプロセッシングユニットが協調できるようにするため、Zynq UltraScale+ MPSoC デバイスのソフトウェアフレームワークはヘテロジニアスマルチプロセッシングシステムを管理するための電力管理 API の実装を提供します。

次の図は、API ベースの電力管理ソフトウェアアーキテクチャを示しています。



X19503-071317

図 11-3: API ベースの電力管理ソフトウェアアーキテクチャ

電力管理フレームワークの概要

Zynq UltraScale+ MPSoC デバイスの電力管理フレームワーク (PMF) は、EEMI の実装をベースとしています (『Embedded Energy Management API Specification』(UG1200) [参照 17] 参照)。この実装には、プロセッシングユニット (PU) が電力管理コントローラーにメッセージを送信するための機能や、電力管理コントローラーがメッセージを PU に送信するためのコールバック機能を持つ API が含まれます。API は、次のような機能に分類されます。

- 。 PU のサスペンドとウェークアップ
- 。 メモリやペリフェラルなどのスレーブ デバイスの電力管理
- 。 その他
- 。 直接アクセス

API の呼び出しと応答

IPI を使用する電力管理の通信

Zynq UltraScale+ MPSoC デバイスの場合、電力管理通信層は、IPI ブロックで提供されるプロセッサ間割り込み (IPI) を使用して実装されます。IPI の詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「割り込み」の章を参照してください。

各 PU には、割り込みとペイロード バッファを含む電力管理コントローラーと接続する専用の IPI チャネルが 1 本あります。バッファは API ID (最大 5 つの引数) を渡します。ターゲットへの IPI 割り込みによって、次のような API の処理がトリガーされます。

- 。 API 関数を呼び出すと、PU が PMU に IPI を生成して必要な電力管理動作の実行を促します。
- 。 PMU は、各 PM 動作を分割できないものとして実行します。つまり、この動作は割り込みで中断できません。
- 。 PMU から PU への通知に使用される PM のコールバック機能をサポートするために、各 PU にはこれらのコールバック IPI の処理機能が実装されています。

肯定応答メカニズム

Zynq UltraScale+ MPSoC の電力管理フレームワーク (PMF) は、ブロッキング/ノンブロッキングの肯定応答をサポートしています。肯定応答の引数を提供するほとんどの API 呼び出しでは、呼び出し元で次に示す肯定応答オプションのいずれかを選択できます。

- 。 REQUEST_ACK_NO: 肯定応答なしの要求
- 。 REQUEST_ACK_BLOCKING: ブロッキング肯定応答の要求
- 。 REQUEST_ACK_NON_BLOCKING: コールバック機能を使用する非ブロッキング肯定応答の要求

各プロセッシングユニット (PU) は API 呼び出し用に 1 本の IPI チャネルを使用するため、電力管理 API 呼び出しが複数ある場合はシリアル化されます。1 つの要求が電力管理コントローラーに送信された後、電力管理コントローラーが 1 つ目の要求の処理を完了した場合のみ次の要求を発行できます。したがって、いずれの肯定応答メカニズムを使用した場合でも、後続要求を発行しているときには、呼び出し元がブロックされます。

肯定応答なし

肯定応答なし (REQUEST_ACK_NO)) が要求された場合、電力管理コントローラーは呼び出し元へ肯定応答を返さずに要求を処理します。それ以外は、通常は肯定応答が送信されます。

ブロッキング肯定応答

ブロッキング肯定応答 (REQUEST_ACK_BLOCKING) を指定した PM 要求が開始されると、電力管理コントローラーが肯定応答を提供しない限り、呼び出し元はブロックされたままとなります。

プラットフォーム管理ユニット (PMU) は、IPI 割り込みをクリアする前に IPI バッファの応答部分に肯定応答値を書き込みます。IPI 観察レジスタが割り込みのクリアを示した後 (つまり、PMU が発行された IPI の処理を完了した後) に、呼び出し元は IPI バッファから肯定応答値を読み出します。PMU が次の要求に対応する準備ができるようになるまで、PU の IPI は無効になります。

非ブロッキング肯定応答

非ブロッキング肯定応答 (REQUEST_ACK_NON_BLOCKING) を指定した PM 要求が発行されると、呼び出し元はプラットフォーム管理ユニット (PMU) が要求を処理するまで待機する必要はありません。つまり、呼び出し元は、PMU からの肯定応答を待つ間、別の動作を実行できます。

PMU は、要求の処理を完了すると、IPI バッファに肯定応答値を書き込みます。その後、PMU は呼び出し元 PU に対して IPI をトリガーして、その動作を割り込み、送信された肯定応答に関する情報を通知します。

非ブロッキング肯定応答は、呼び出し元 PU で実行されるコールバック関数を使用して実装されます (XpM_NotifyCb Callback 参照)。

XpM_NotifyCb の詳細は、付録 K 「XilPM Library v2.5」を参照してください。

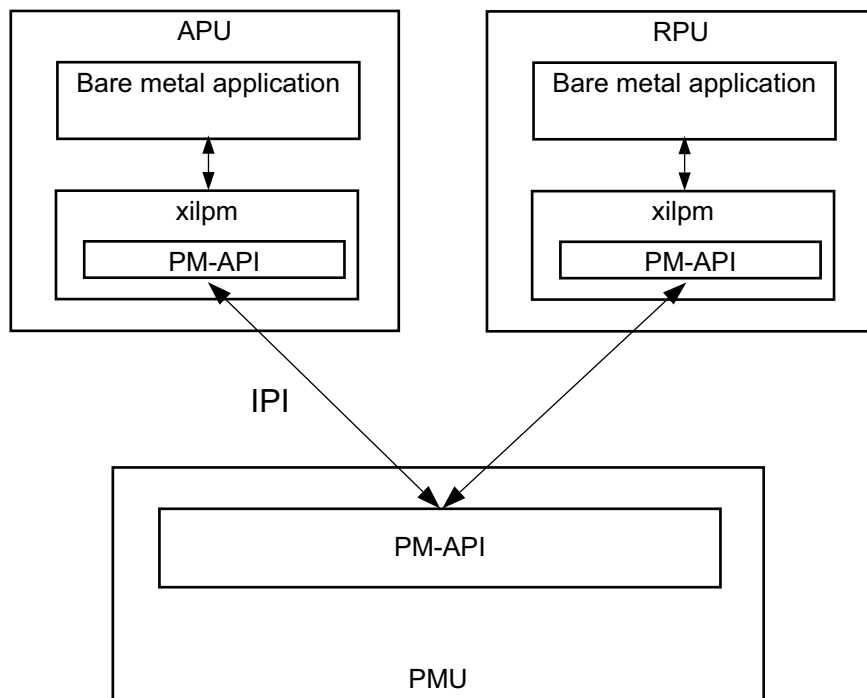
電力管理フレームワークの層

Zynq UltraScale+ MPSoC デバイスの電力管理フレームワーク (PMF) インプリメンテーションには、次に示す API 層があります。

- 。 **Xilpm:** このライブラリ層は、APU や RPU などの異なるプロセッシングユニットのスタンドアロンアプリケーションに使用されます。
- 。 **ATF:** Arm Trusted Firmware (ATF) には、クライアント側の PM フレームワークの独自インプリメンテーションが含まれます。現在、Linux オペレーティングシステムで使用されています。
- 。 **PMU ファームウェア:** プラットフォーム管理ユニット ファームウェア (PMUFW) は、プラットフォーム管理ユニット (PMU) 上で動作し、電力管理 API を実装しています。

詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] のこのセクションを参照してください。

次の図に APU、RPU、PMF 間における API の相互作用を示しています。



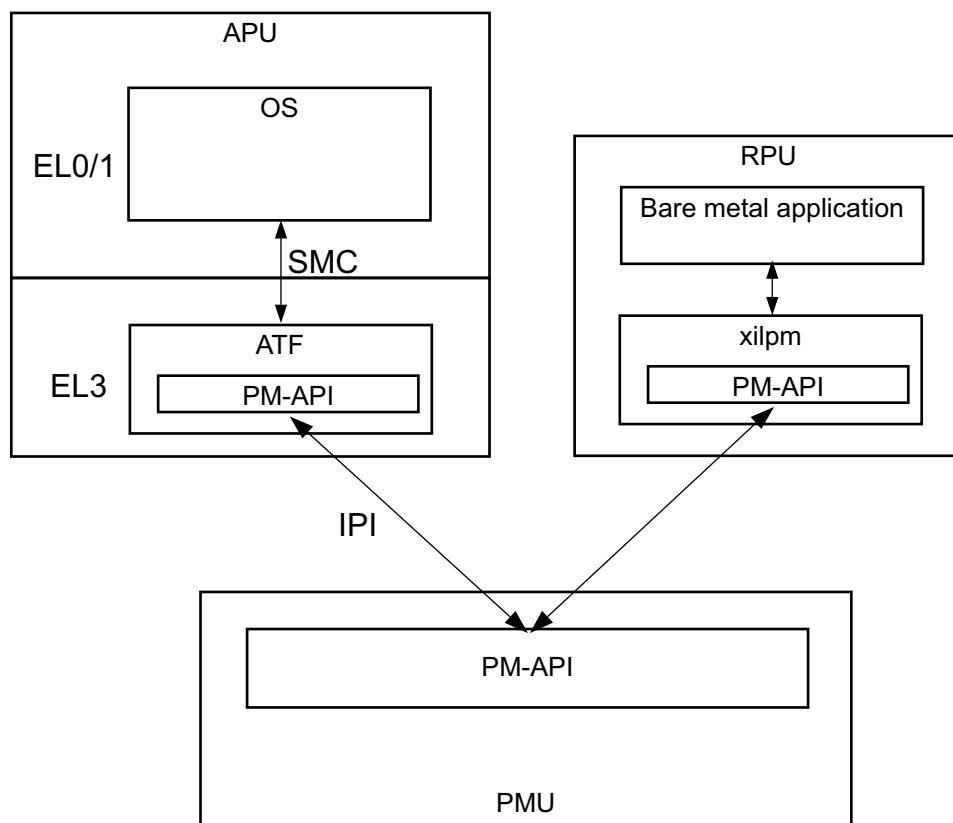
X19094-071317

図 11-4: ベアメタルアプリケーションのみで使用する API 層

APU がオペレーティング システムを使用してフル ソフトウェア スタックを実行する場合、Xilpm ライブラリは使用されません。その代わりに、EL3 上で動作する ATF がクライアント側の電力管理 API を実装し、上位層に対して SMC (セキュア モニター コール) ベースのインターフェイスを提供します。

図 11-5 に、この動作を示します。Armv8 アーキテクチャおよびその異なる実装モードの詳細は、『Power State Coordination Interface』[参照 46] を参照してください。

次の図は、APU 上でフル ソフトウェア スタックが実行される場合の PMF 層を示しています。



X19093-071317

図 11-5: APU 上でフル ソフトウェア スタックが実行される場合の PM フレームワーク層

一般的な電力管理 API 呼び出しフロー

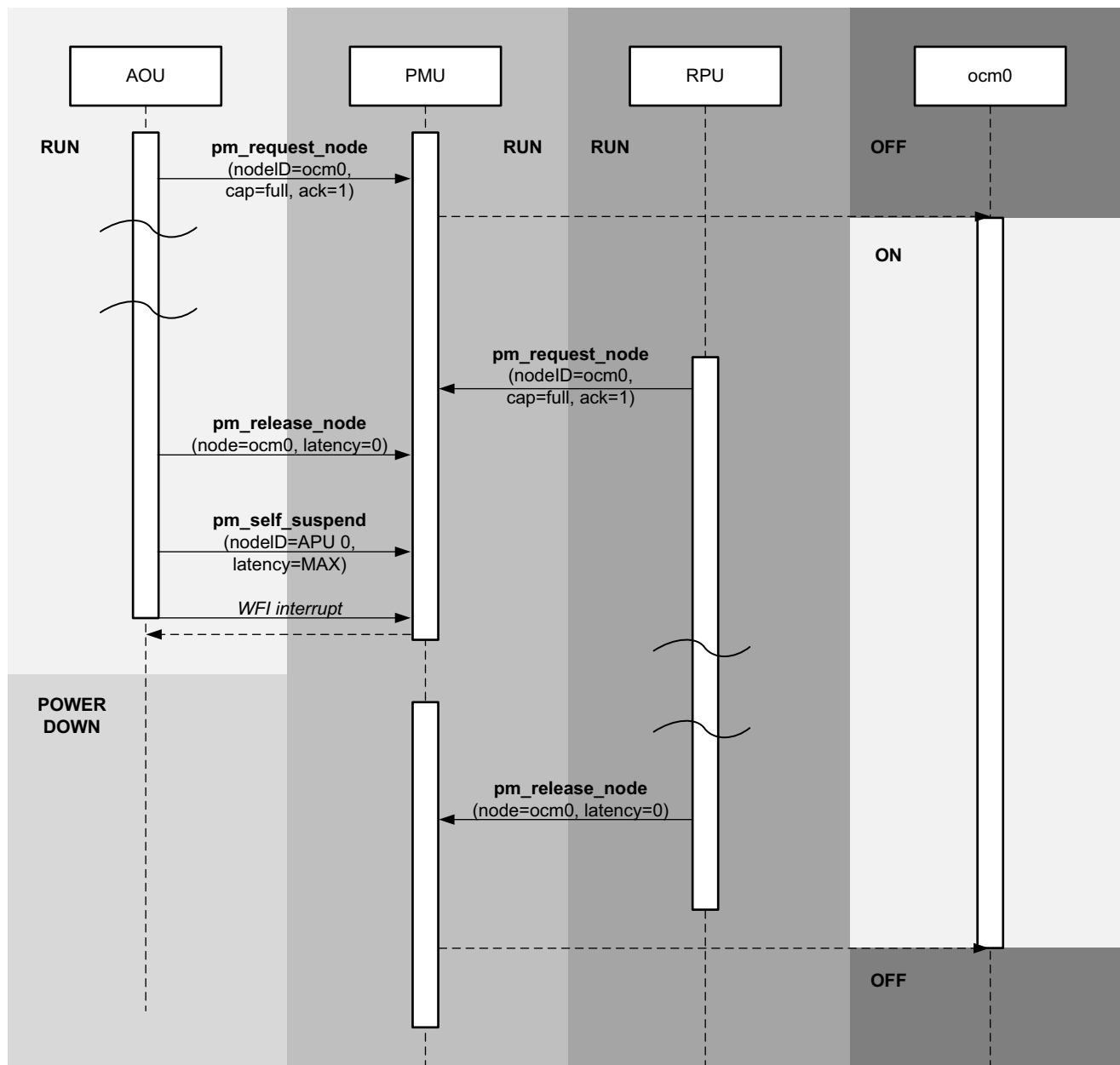
電力管理に関わるすべてのエンティティを「ノード」と呼んでいます。この後のセクションでは、電力管理フレームワーク (PMF) が、APU や RPU に割り当てられたスレーブ ノードを使用してどのように機能するのかを説明します。

一般に、APU または RPU は、電力管理コントローラーに要求を送ることによって、スレーブ ノードを使用することを通知します。次に、スレーブ ノードに対する機能要件を電力管理コントローラーに通知します。この時点で、電力管理コントローラーがスレーブ ノードの電源を投入し、APU または RPU はスレーブ ノードを初期化できるようになります。

スレーブ ノードの要求とリリース

PU がペリフェラルまたはメモリいずれかのスレーブ ノードを必要とする場合は、電力管理 API を使用するスレーブ ノードを要求する必要があります。スレーブ ノードが機能の実行を完了して不要になった場合は、リリースして電源をオフにできます。

次の図は、APU と RPU が 1 つの OCM メモリ バンク (ocm0) を共有している場合の呼び出しフローを示しています。



X20022-110217

図 11-6: APU と RPU が 1 つの OCM メモリ バンクを共有している場合の PM フレームワーク呼び出しシーケンス

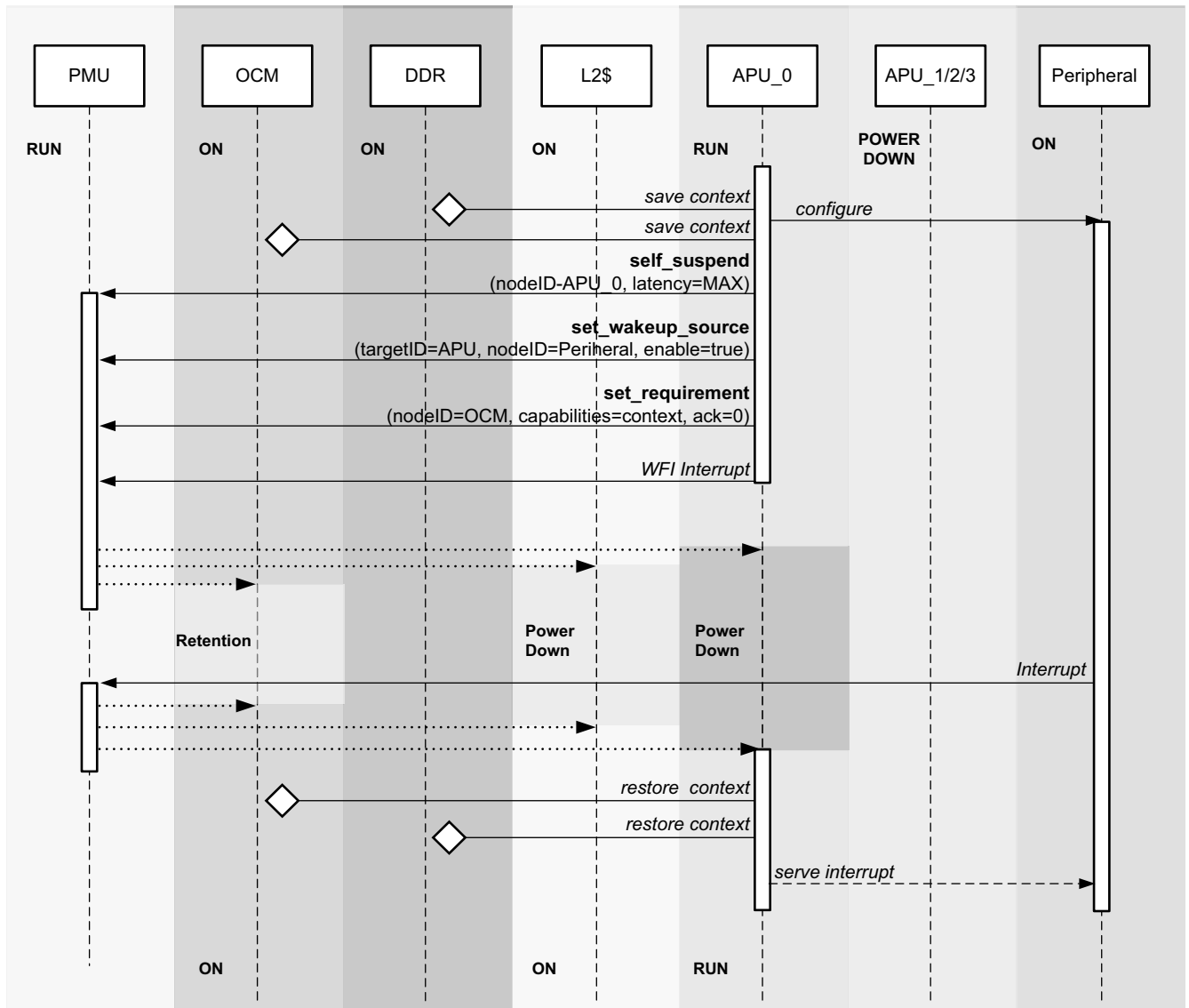
注記: APU が XStatus XPm_ReleaseNode を呼び出した後、RPU も同じスレーブ ノードを要求しているため、ocm0 メモリの電源はオンのままとなります。RPU も ocm0 ノードをリリースした後に、PMU は ocm0 メモリの電源をオフにします。

プロセッシングユニットのサスペンドとレジューム

プロセッシングユニット (PU) の電源をオフにすることは、アイドル状態に移移するのとは異なり、外部エンティティを使用してパワーダウンおよびパワーアップ遷移を処理する必要があります。

Zynq UltraScale+ MPSoC デバイスの場合、プラットフォーム管理ユニット (PMU) がすべての電源ステートの遷移を管理します。

PU が PMU に電源ステートの遷移が要求されていることを通知します。次の図に、このプロセスを示しています。



X20023-110217

図 11-7: APU のサスペンドおよびレジューム プロセス

サスペンドまたはレジューム プロセスの詳細は、「CPU/PU のセルフ サスペンド」で説明しています。通常、1 つの PU を動作させるには、多くのスレーブ ノードが必要です。

サブシステムの電力管理

分離の設定

Zynq UltraScale+ MPSoC は、複数のサブシステムに分割してそれぞれを電力管理フレームワークで個別に管理できます。たとえば、Linux サブシステムとリアルタイム サブシステムを定義できます。Linux サブシステムには、APU (PM マスター) と多数のペリフェラル (PM スレーブ) を含めることが考えられます。リアルタイム サブシステムには、RPU とその他の多数のペリフェラルを含めることが考えられます。各サブシステムは、互いに影響を与えることなくパワーアップ、パワーダウン、リスタート、サスペンドが可能です。サブシステムに含めることのできる PM マスターは 1 つだけで、FPD ペリフェラルと LPD ペリフェラルは両方を含めることができます。

Vivado PCW ツールを使用すると、独自のサブシステムを作成できます。図 11-8 に、PCW の画面例を示します。ここに表示している有効な設定は 1 つの APU サブシステムだけで構成されており、RPU サブシステムは含まれていません。

Please review **Known Limitations** under the **Isolation Configuration** Section of PG201.

☒ Enable Isolation ☒ Enable Secure Debug ☐ Lock Unused Memory

Search:

Name	Start Address	Size	Unit	TZ Settings	Access Sett...	End Address	Type
LINUX							
Masters							
SD1				Secure			
GEM3				NonSecure			
APU							
PCIe				NonSecure			
DP				NonSecure			
GPU				NonSecure			
Coresight							
SATA0				NonSecure			
SATA1				NonSecure			
USB0				NonSecure			
FPD_DMA				NonSecure			
DAP							
QSPI				NonSecure			
Slaves							
Memory							
DDR_LOW	0x0	2	GB	NonSecure	Read/Write	0x7FFFFFFF	DDR
QSPI_Linear...	0xC0000000	524288	KB	NonSecure	Read/Write	0xDFFFFFFF	LPD
Peripherals							
CAN1	0xFF070000	64	KB	NonSecure	Read/Write	0xFF07FFFF	LPD
GEM3	0xFF0E0000	64	KB	NonSecure	Read/Write	0xFF0EFFFF	LPD
GPIO	0xFF0A0000	64	KB	NonSecure	Read/Write	0xFF0AFFFF	LPD
I2C0	0xFF020000	64	KB	NonSecure	Read/Write	0xFF02FFFF	LPD
I2C1	0xFF030000	64	KB	NonSecure	Read/Write	0xFF03FFFF	LPD
SWDT0	0xFF150000	64	KB	NonSecure	Read/Write	0xFF15FFFF	LPD
TTC0	0xFF110000	64	KB	NonSecure	Read/Write	0xFF11FFFF	LPD
UART0	0xFF000000	64	KB	NonSecure	Read/Write	0xFF00FFFF	LPD
UART1	0xFF010000	64	KB	NonSecure	Read/Write	0xFF01FFFF	LPD
TTC1	0xFF120000	64	KB	NonSecure	Read/Write	0xFF12FFFF	LPD
TTC2	0xFF130000	64	KB	NonSecure	Read/Write	0xFF13FFFF	LPD
TTC3	0xFF140000	64	KB	NonSecure	Read/Write	0xFF14FFFF	LPD

> Control and Status Registers

図 11-8: PCW の設定

Name	Start Address	Size	Unit	TZ Settings	Access Setti...	End Address	Type
Linux							
Masters							
Slaves							
Memory							
Peripherals							
Control and Status Registers							
USB3_0	0xFF9D0000	64	KB	NonSecure	Read/Write	0xFF9DFFFF	LPD
USB3_0_XHCI	0xFE200000	1024	KB	NonSecure	Read/Write	0xFE2FFFFF	LPD
Coresight	0xFE800000	8192	KB	NonSecure	Read/Write	0xFEFFFFF	LPD
LPD_DMA_0	0xFFA80000	64	KB	NonSecure	Read/Write	0xFFA8FFFF	LPD
LPD_DMA_1	0xFFA90000	64	KB	NonSecure	Read/Write	0xFFA9FFFF	LPD
LPD_DMA_2	0xFFAA0000	64	KB	NonSecure	Read/Write	0xFFAAFFFF	LPD
LPD_DMA_3	0xFFAB0000	64	KB	NonSecure	Read/Write	0xFFABFFFF	LPD
LPD_DMA_4	0xFFAC0000	64	KB	NonSecure	Read/Write	0xFFACFFFF	LPD
LPD_DMA_5	0xFFAD0000	64	KB	NonSecure	Read/Write	0xFFADFFFF	LPD
LPD_DMA_6	0xFFAE0000	64	KB	NonSecure	Read/Write	0xFFAEFFFF	LPD
LPD_DMA_7	0xFFAF0000	64	KB	NonSecure	Read/Write	0xFFAFFFFF	LPD
QSPI	0xFF0F0000	64	KB	NonSecure	Read/Write	0xFF0FFFFF	LPD
SD1	0xFF170000	64	KB	NonSecure	Read/Write	0xFF17FFFF	LPD
AMS	0xFFA50000	64	KB	NonSecure	Read/Write	0xFFA5FFFF	LPD
APM1	0xFFA00000	64	KB	NonSecure	Read/Write	0xFFA0FFFF	LPD
APM2	0xFFA10000	64	KB	NonSecure	Read/Write	0xFFA1FFFF	LPD
APM_FPD_LPD	0xFFA30000	64	KB	NonSecure	Read/Write	0xFFA3FFFF	LPD
APM_INTC_IUO	0xFFA20000	64	KB	NonSecure	Read/Write	0xFFA2FFFF	LPD
IOU_GPV	0xFE000000	1024	KB	NonSecure	Read/Write	0xFE0FFFFF	LPD
IPI_CTRL	0xFF380000	512	KB	NonSecure	Read/Write	0xFF3FFFFF	LPD
LPD_GPV	0xFE100000	1024	KB	NonSecure	Read/Write	0xFE1FFFFF	LPD
RTC	0xFFA60000	64	KB	NonSecure	Read/Write	0xFFA6FFFF	LPD

図 11-9: PCW の設定 (続き)

APU_secure							
Masters							
SD1				Secure			
APU							
Slaves							
Memory							
OCM	0xFFFFC0000	256	KB	Secure	Read/Write	0xFFFFCFFF	OCM
Control and Status Registers							
CRF_APB	0xFD1A0000	1280	KB	Secure	Read/Write	0xFD20FFFF	FPD
CRL_APB	0xFF5E0000	2560	KB	Secure	Read/Write	0xFF85FFFF	LPD
EFUSE	0xFFCC0000	64	KB	Secure	Read/Write	0xFFCCFFFF	LPD
IOU_SLCR	0xFF180000	768	KB	Secure	Read/Write	0xFF23FFFF	LPD
PMU Firmware							
Masters							
PMU							
Slaves							
Peripherals							
UART0	0xFF000000	64	KB	NonSecure	Read/Write	0xFF00FFFF	LPD
Control and Status Registers							
CRF_APB	0xFD1A0000	1280	KB	Secure	Read/Write	0xFD20FFFF	FPD
DDR_XMPU0...	0xFD000000	64	KB	Secure	Read/Write	0xFD00FFFF	FPD
DDR_XMPU1...	0xFD010000	64	KB	Secure	Read/Write	0xFD01FFFF	FPD
DDR_XMPU2...	0xFD020000	64	KB	Secure	Read/Write	0xFD02FFFF	FPD
DDR_XMPU3...	0xFD030000	64	KB	Secure	Read/Write	0xFD03FFFF	FPD
DDR_XMPU4...	0xFD040000	64	KB	Secure	Read/Write	0xFD04FFFF	FPD
DDR_XMPU5...	0xFD050000	64	KB	Secure	Read/Write	0xFD05FFFF	FPD
FPD_SLCR	0xFD610000	512	KB	Secure	Read/Write	0xFD68FFFF	FPD
FPD_XMPU...	0xFD5D0000	64	KB	Secure	Read/Write	0xFD5DFFFF	FPD
LPD_XPPU	0xFF980000	64	KB	Secure	Read/Write	0xFF98FFFF	LPD
CRL_APB	0xFF5E0000	2560	KB	Secure	Read/Write	0xFF85FFFF	LPD
EFUSE	0xFFCC0000	64	KB	Secure	Read/Write	0xFFCCFFFF	LPD
IOU_SLCR	0xFF180000	768	KB	Secure	Read/Write	0xFF23FFFF	LPD
LPD_SLCR	0xFF410000	640	KB	Secure	Read/Write	0xFF4AFFFF	LPD
OCM_XMPU...	0xFFA70000	64	KB	Secure	Read/Write	0xFFA7FFFF	LPD
RPU	0xFF9A0000	64	KB	Secure	Read/Write	0xFF9AFFFF	LPD

図 11-10: PCW の設定 (続き)

注記: PCW ツールは、セキュリティのためにペリフェラルを分離する目的でも使用できます。ペリフェラルを分離する方法の詳細は、『Zynq UltraScale+ MPSoC: エンベデッド デザイン チュートリアル』(UG1209) [参照 13] および『Zynq UltraScale+ MPSoC Processing System LogiCORE IP 製品ガイド』(PG201) [参照 14] を参照してください。

コンフィギュレーション オブジェクト

サブシステムのコンフィギュレーションは、Vivado および PetaLinux ツールチェーンによって生成されるコンフィギュレーション オブジェクトに記録されます。コンフィギュレーション オブジェクトには、次の情報が含まれます。

- システムに存在する PM マスター (APU/RPU)。コンフィギュレーション オブジェクトで指定されていない PM マスターは、PMU によってパワーダウンされます。
- 各 PM マスターに対する設定可能なパーミッション。
 - どの PM マスターがどの PM スレーブを使用できるか (PM マスターは、同じサブシステムに属するすべての PM スレーブを使用できる)。
 - MMIO アドレス領域へのアクセス。
 - ペリフェラル リセット ラインへのアクセス。
- 事前に割り当てられた PM スレーブ。これらの PM スレーブは、PM マスターからの要求がなくても使用できます。これらの PM スレーブは、PM マスターはブート時にこれらの PM スレーブを必要とします。ツールチェーンによって、APU は要求を出さずに L2 キャッシュと DDR バンクにアクセスできるようになります。RPU の場合も同様に、すべての TCM バンクにアクセスできるようになります。

ブート時に、コンフィギュレーション オブジェクトは FSBL から PMU ファームウェアに渡されます。詳細は、「[コンフィギュレーション オブジェクト](#)」を参照してください。

注記: コンフィギュレーション オブジェクトの作成には、分離は必要ありません。サブシステムを作成してコンフィギュレーション オブジェクトをカスタマイズした後、[Enable Isolation] をオフにできます。

電力管理の初期化

ブート中は電力管理が無効で、すべてのペリフェラルに電源が供給されます。これは、ブートおよび初期化中はペリフェラル間で相互に依存関係 (一時的なものを含む) が存在することが多いためです。ペリフェラルの初期化およびアプリケーション バイナリのロードが完了したら、FSBL はコンフィギュレーション オブジェクトを PMU に渡します。これにより、PMU はすべてのサブシステムおよびそれぞれの PM マスターとスレーブを認識できるようになります。コンフィギュレーション オブジェクトに含まれていない PM マスターと PM スレーブは使用されることがないため、PMU によって電源がオフにされます。

PM マスターがすべての PM スレーブを常時使用することはほとんどありません。したがって、使用する場合のみ電源を供給するようにします。PM マスターは、PM スレーブの使用前と使用後に PMU に通知する必要があります。この機能は、PetaLinux カーネルに実装されています。この要件は、消費電力の最適化よりも機能を重視して新しい RPU アプリケーションの開発を始める際に障壁となります。したがって、PM スレーブを使用する際に通知を送らない PM 非対応のマスターも PMU でサポートした方が便利です。そのためには、PM マスターが PmInitFinalize 要求を PMU に送信するまでサブシステム内のすべての PM スレーブの電源をオンにしておきます。PM 非対応のマスターはこの要求を送信しないため、このマスターが使用する PM スレーブは常に (またはこの PM マスター自身がパワーダウンするまで) 電源が供給されたままとなります。

PM 対応マスターは、サブシステムの初期化が完了したらこの要求を送信します。これにより PMU は、サブシステム内で使用されていない PM スレーブをパワーダウンします。

この結果、システム内にアプリケーションを実行していない RPU マスターが存在する場合、PMU ファームウェアはこれを PM 非対応のマスターと見なし、RPU とそのスレーブをパワーダウンしません。この動作は、2018.3 リリース以降で修正されており、未使用の RPU をパワーダウンできるようになっています。この変更はコンパイルフラグ `ENABLE_UNUSED_RPU_PWR_DWN` で保護されており、デフォルトで有効になっています。このフラグが有効な場合、使用していない RPU と関連スレーブがパワーダウンされます。

注記: デフォルトで RPU の電源をオフにしないようにするには、PMU ファームウェアのコンパイル時に `ENABLE_UNUSED_RPU_PWR_DWN` フラグを 0 に設定します。JTAG ブート モードの場合、`ENABLE_UNUSED_RPU_PWR_DWN` フラグが 1 でも動作の変更に影響はありません。

注記: サブシステムは、互いに重複してもかまいません。つまり、一部の PM スレーブは複数のサブシステム (DDR と OCM など) に属することができます。1 つの PM スレーブが複数のサブシステムに属する場合、この PM スレーブが関連するすべての PM マスターからリリースされるまで、またはこれらすべての PM マスターが自分自身をパワーダウンするまで PMU はこのスレーブをパワーダウンしません。

デフォルトのコンフィギュレーション

デフォルトでは分離設定は無効で、ツールチェーンは 3 つのサブシステムで構成されるコンフィギュレーションを生成します。各サブシステムの PM マスターは、APU、R5-0、および R5-1 です。これら 3 つのサブシステムには、いずれもすべての PM スレーブが属します (つまり、すべてのサブシステムが重複している)。これは、[Enable Isolation] をオフにした場合に PCW によって生成されるデフォルトのコンフィギュレーションです。デフォルトの PetaLinux カーネル コンフィギュレーションは PM に対応していますが、R5-0 と R5-1 も「PM 対応」のアプリケーションを実行する必要があるため、このようにしないとパワーダウンされます。それ以外のケースでは、PMU は PM スレーブをパワーダウンしません。

注記: プロセッサのブートおよび実行を禁止するコンフィギュレーションを作成することもできます。初心者の場合、「分離の設定」で説明した APU のみの構成を使用し、必要に応じてカスタマイズすることを推奨します。

RPU のロックステップ モードとスプリット モード

ツールチェーンは、RPU の動作モードを PCW の分離設定から次のように推論します。

- サブシステムに RPU が含まれない場合: コンフィギュレーション オブジェクトに RPU が含まれない。
- サブシステムに R5-0 のみが存在する場合: コンフィギュレーション オブジェクトにロックステップ モードの R5-0 が含まれる。
- サブシステムに R5-0 と R5-1 の両方が存在する場合: コンフィギュレーション オブジェクトにスプリット モードの R5-0 と R5-1 が含まれる。
- サブシステムに R5-1 のみが存在する場合: コンフィギュレーション オブジェクトにスプリット モードの R5-1 が含まれる。

デフォルトのコンフィギュレーション オブジェクトには 2 つの RPU PM マスター (R5-0、R5-1) が含まれ、PMU は R5-0 と R5-1 がスプリット モードで動作しているものと判断します。ただし、RPU がロックステップ モードとスプリット モードのどちらで動作しているかは、実際にはブート イメージで決定します。ブート イメージの RPU 動作モードと、コンフィギュレーション オブジェクトの RPU PM マスターの数が一致している必要があります。一致しない場合、電力管理フレームワークは正しく動作しません。

注記: R5 をロックステップ モードで使用する場合は、PCW で [Isolation Configuration] をオンにし、サブシステムに R5-0 のみが存在すること (R5-1 が存在しないこと) を確認する必要があります。

デバイスの共有

APU と RPU でデバイスへのアクセスを共有することは可能ですが、十分な注意を払う必要があります。デバイスへのアクセスと動作は、そのクロック (該当する場合のみ)、コンフィギュレーション、および電力ステート (オン、オフ、リテンションなど) によって決まります。PMU はすべての PM マスターの要求を満たす範囲でデバイスの電力ステートを最小に維持しますが、デバイスのクロックとコンフィギュレーションは APU と RPU によってセットアップされます。

Linux を実行している APU と RPU の間でデバイスを共有する場合は、特に注意が必要です。Linux は、いずれかのデバイスを別のエンティティが使用していることを認識せず、自らが使用していないデバイスに対してクロックゲーティング、パワーゲーティング、および無効化を実行します。対処方法は次のとおりです。

- デバイスの Linux 実行時電力管理を無効にする。詳細は、<https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-devices-power> を参照してください。これにより、Linux が使用しない場合もデバイスは動作を継続します。ただし、Linux がスリープに移行すると、デバイスはクロックゲーティングおよび無効化されます。
- 特別なデバイスドライバを実装する。

APU が使用しないデバイスは、デバイス ツリーから削除する必要があります。

API を使用した電力管理

はじめに

この章では、ザイリンクスの電力管理フレームワーク (PMF) API を使用して、一般的な電力管理タスクを実行する手順について詳しく説明します。

プロセッシングユニットに電力管理機能を実装

プロセッサ上で実行されるスタンドアロン アプリケーションは、Xilpm ライブラリが提供する機能を使用して電力管理 API 呼び出しを開始できます。

プロジェクトに Xilpm ライブラリを含める方法は、『ザイリンクス ソフトウェア開発キットのヘルプ』(UG782) [参照 24] を参照してください。

Xilpm ライブラリの初期化

すべての電力管理 API 呼び出しを開始する前には、XpM_InitXilpm を呼び出し、適切に初期化されたプロセッサ間割り込み (IPI) ドライバ インスタンスにポインターを移動させて、Xilpm ライブラリを初期化する必要があります。

IPI の詳細は、このセクション (『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の第 13 章「割り込み」) を参照してください。

XpM_InitXilpm の詳細は、付録 K 「XilPM Library v2.5」を参照してください。

スレーブ デバイスの使用

Zynq® UltraScale+™ MPSoC デバイスの電力管理フレームワーク (PMF) には、メモリやペリフェラルなどのスレーブ デバイス (PM スレーブとも呼ばれている) を管理するための専用機能が含まれています。プロセッシング ユニット (PU) がこれらの機能を使用して、電力管理コントローラーにこれらのデバイスの要件 (機能やウェークアップレイテンシなど) を伝えます。電力管理コントローラーは、各デバイスが該当するすべての PU 要件を満たしながら最小限の消費電力ステートを常に維持できるようにシステムを管理します。

ノードの要求とリリース

PU は、XPm_RequestNode API を使用して、スレーブ デバイスへのアクセスを要求し、そのデバイスの要件を提示します。PU とスレーブが同じサブシステムに属している場合、電力管理コントローラーは要求されたデバイスのパワーオンおよびアクティブ ステートを管理します。

デバイスが使用されなくなると、通常 PU は XPm_ReleaseNode 関数を呼び出します。これにより、PM コントローラーはそのデバイスの電力ステートを再評価し、可能であれば低電力ステートに移行させることができます。また、別の PU がこのデバイスへのアクセスを要求することも可能です。

XPm_ReleaseNode の詳細は、[付録 K 「XilPM Library v2.5」](#) を参照してください。

要件の変更

PU が PM スレーブを使用する場合、スレーブの機能に対する要件が変化することがあります。たとえば、使用していないインターフェイスがある場合、そのインターフェイス ポートを低電力ステートに移行させたり、完全に電源をオフにしたりできます。PM スレーブの機能に対する要件を変更する場合、PU は XPm_SetRequirement を使用します。再び要件が変化する場合がある場合、PU は通常 PM スレーブをリリースしません。

次の呼び出しの例では、ウェークアップ割り込みのみを必要とするように node 引数の要件を変更しています。

```
XPm_SetRequirement(node, PM_CAP_WAKEUP, 0, REQUEST_ACK_NO);
```



重要: ノードの要件を 0 に設定することと、PM スレーブをリリースすることは同じではありません。PM スレーブをリリースすると、ほかの PU がそのデバイスを独占的に使用する可能性があります。

複数の PU が 1 つの PM スレーブ (通常はメモリ) を共有する場合、電力管理コントローラーはその PM スレーブを使用するすべての PU の要件を満たすように PM スレーブの電力ステートを選択します。

PM スレーブに対する要件には、機能要件とレイテンシ要件があります。機能要件には、最高機能ステート、いくつかの中間機能ステート、非アクティブ ステート (ただしコンフィギュレーションは保持)、およびオフ ステートがあります。レイテンシ要件は、その PM スレーブがいずれかのステートから最高機能ステートへ切り替わるまでの最大許容時間を指定します。この時間制限を満たすことができない場合、電力管理コントローラーは機能要件にかかわらず PM スレーブを最高機能ステートのままにします。

XPm_SetRequirement の詳細は、[付録 K 「XilPM Library v2.5」](#) を参照してください。

CPU/PU のセルフ サスペンド

PU は複数の CPU の集合体として構成できます。APU は、4 つの CPU を持つ PU です。RPU は 2 つの CPU を持ち、スプリット モードで動作する場合は 2 つの PU と見なされ、ロックステップ モードで動作する場合は 1 つの PU と見なされます。

セルフ サスペンドを実行するには、CPU が `XPm_SelfSuspend` 関数を呼び出して、電力管理コントローラーへセルフ サスペンドの開始を伝えます。その後、次の動作が実行されます。

- `XPm_SelfSuspend()` 呼び出しが処理された後、その後いかなる割り込みが発生しても CPU は必ずスリープ状態になります。APU および RPU では、このようなビヘイビアを管理するために、`XPm_SelfSuspend()` 呼び出しが完了した後に CPU へのすべての割り込みを GIC ウェークアップ割り込みとして電力管理コントローラーへ送信します。
- その後、電力管理コントローラーは、CPU がサスペンド プロセスを完了するまで待機します。PU は、`XPm_SuspendFinalize` を呼び出して、スリープ ステートへ遷移する準備が整ったことを電力管理コントローラーへ通知します。
- `XPm_SuspendFinalize()` 関数はアーキテクチャに依存します。これにより、未処理の電力管理 API 呼び出しが処理され、アーキテクチャ固有のサスペンド シーケンスが実行されます。また、サスペンドの完了が電力管理コントローラーへ通知されます。
- APU や RPU などの Arm プロセッサの場合、`XPm_SuspendFinalize()` 関数は WFI (Wait for interrupt) 命令を使用します。この命令は CPU をサスペンドし、電力管理コントローラーへの割り込みをトリガーします。
- サスペンド完了が電力管理コントローラーに通知されると、電力管理コントローラーは CPU をリセット状態にし、アイランド内のその他のコンポーネントが現在アクティブでなければ CPU の電源アイランドの電源を切断できます。
- CPU の GIC インターフェイスを介して有効になった割り込みは、その特定の CPU に割り当てられた GIC ウェークアップ割り込みとして電力管理コントローラー (PMC) にリダイレクトされます。割り込みはリダイレクトされるため、CPU は電力管理コントローラーを使用してのみ復帰できます。
- PU をサスペンドするには、すべての CPU を個別にサスペンドする必要があります。

`XPm_SelfSuspend` および `XPm_SuspendFinalize` の詳細は、[付録 K 「XilPM Library v2.5」](#) を参照してください。

レジューム (復帰) の実行

CPU は、ハードウェア リソースでトリガーされるウェークアップ割り込み、または `XPm_RequestWakeup` API を使用する明示的なウェークアップ要求で復帰可能です。

CPU は、`XPm_SelfSuspend` 呼び出しで提供されるレジューム アドレスから実行を開始します。

`XPm_RequestWakeup` および `XPm_SelfSuspend` の詳細は、[付録 K 「XilPM Library v2.5」](#) を参照してください。

ウェークアップ ソースのセットアップ

いずれの FPD デバイスも使用されておらず、既存のレイテンシ要件がパワーダウンを許可する場合は、電力管理コントローラーが FPD 全体の電源をオフにできます。FPD の電源がオフ状態で、LPD のデバイスによってトリガーされる割り込みで APU を起動する場合は、FPD ウェークアップ イベントの伝搬を許可するように GIC プロキシを設定しておく必要があります。ウェークアップ割り込みを発行する必要のあるすべてのデバイスに対して `XPm_SetWakeUpSource` を呼び出すことによって、APU は確実に起動できます。

したがって、サスペンドする前に、APU は `XPm_SetWakeUpSource(NODE_APU, node, 1)` を呼び出して、必要なスレーブをウェイクアップソースとして追加する必要があります。この場合 APU は使用しているすべてのスレーブに対して要件 0 を設定できます。APU がサスペンド処理を終了し、ほかの PU が FPD 内のリソースを使用していない場合、PM コントローラーは FPD 全体の電源をオフにし、GIC プロキシを設定して LPD スレーブのウィーク イベントの伝搬を有効にします。

`XPm_SetWakeUpSource` の詳細は、付録 K 「[XilPM Library v2.5](#)」を参照してください。

サスペンド処理の中断

`XPm_SetSelfSuspend` 関数を呼び出した後に PU がサスペンド処理の中断を決定した場合、PU は、`XPm_AbortSuspend` 関数を呼び出すことによって、電力管理コントローラーに中断したサスペンドについて通知する必要があります。

`XPm_SetSelfSuspend` および `XPm_AbortSuspend` の詳細は、付録 K 「[XilPM Library v2.5](#)」を参照してください。

サスペンド処理中の PM スレーブへの対応

セルフ サスペンドする PU は、使用中のペリフェラルとメモリに関して変更した要件を、電力管理コントローラーに通知する必要があります。PU が電力管理コントローラーに通知しないと、使用されているすべてのデバイスの電源はオンのままとなります。スレーブ デバイスがメモリの場合、通常、次の関数を使用してコンテキストが保持されるようにする必要があります。

```
XPm_SetRequirement(node, PM_CAP_CONTEXT, 0, REQUEST_ACK_NO);
```

`XPm_SetSelfSuspend` を呼び出した後、サスペンド中に PM スレーブの要件を設定する場合、CPU がサスペンドを終了するまで設定は延期されます。これにより、サスペンド処理の完了に必要なデバイスは、呼び出し元の CPU がサスペンド処理を完了した後、確実に低電力ステートへ遷移します。

一般的な例には命令メモリがあります。サスペンド処理が完了するまで CPU はメモリへアクセスできます。CPU がメモリをサスペンドした後、そのメモリのコンテンツは保持されます。各 CPU が復帰する前に、延期されたすべての要件は自動的に置き換えられます。

PU 全体がサスペンドする場合、PU 内の最後に起動した CPU がデバイスの変更を管理する必要があります。

`XPm_SetSelfSuspend` の詳細は、付録 K 「[XilPM Library v2.5](#)」を参照してください。

APU/RPU をサスペンドするためのコード例

APU または RPU をサスペンドするためのソース コードの例を次に示します。

```
/* Base address of vector table (reset-vector) */
extern void *_vector_table;
/* Inform PM controller that APU_0 intends to suspend */
XPm_SetSelfSuspend(NODE_APU_0, MAX_LATENCY, 0,
(u64)&_vector_table);
/**
 * Set requirements for OCM banks to preserve their context.
 * The PM controller will defer putting OCMs into retention
 until the suspend is finalized
 */
XPm_SetRequirement(NODE_OCM_BANK_0, PM_CAP_CONTEXT, 0,
REQUEST_ACK_NO);
XPm_SetRequirement(NODE_OCM_BANK_1, PM_CAP_CONTEXT, 0,
```

```
REQUEST_ACK_NO);  
Xpm_SetRequirement(NODE_OCM_BANK_2, PM_CAP_CONTEXT, 0,  
REQUEST_ACK_NO);  
Xpm_SetRequirement(NODE_OCM_BANK_3, PM_CAP_CONTEXT, 0,  
REQUEST_ACK_NO);  
  
/* Flush data cache */  
Xil_DCacheFlush();  
/* Inform PM controller that suspend procedure is completed */  
Xpm_SuspendFinalize();
```

FPD ドメイン全体をサスペンド

フル電力ドメイン全体の電源をオフにするためには、FPD デバイスが使用されていないときに電力管理コントローラーが APU をサスペンドする必要があります。この条件が満たされると、電力管理コントローラーは FPD の電源を自動的にオフにできます。レイテンシ要件によってこの動作が制限されない限り、電力管理コントローラーは FPD の電源をオフにします。要件によって制限された場合、FPD の電源はオンのままになります。

FPD の強制的パワーダウン

XPM_ForcePowerdown 関数を呼び出すことによって、FPD を強制的にパワーダウンさせるオプションがあります。この場合、呼び出し元の PU が電力管理コントローラーで設定された適切な権限を持っている必要があります。電力管理コントローラーは、APU で使用されているすべての PM スレーブを自動的にリリースします。

注記: 特に APU 上で複雑なオペレーティングシステムを実行している場合、OS が適切にサスペンドされなかったことによってデータやシステムの破損が発生する可能性があるため、この強制的パワーダウンは推奨していません。



重要: Xpm_RequestSuspend API を使用する必要があります。

XPM_ForcePowerdown の詳細は、付録 K 「XilPM Library v2.5」を参照してください。

その他のプロセッシングユニットとの相互作用

PU をサスペンドする

PU は、Xpm_RequestSuspend を呼び出して、引数でターゲット ノード名を渡すことで、別の PU のサスペンドを要求できます。

これを受けて、電力管理コントローラーは Xpm_InitSuspendCb() を呼び出します。これは、ターゲット PU に実装されたコールバック関数です。その後、ターゲット PU はサスペンド処理を開始するか、または Xpm_AbortSuspend を呼び出してサスペンドの中断理由を指定します。たとえば、次のコマンドを使用して APU にサスペンドを要求できます。

```
Xpm_RequestSuspend(NODE_APU, REQUEST_ACK_NON_BLOCKING, MAX_LATENCY, 0);
```


次の図は、XPM_RequestSuspend の呼び出しによってトリガーされる一般的なシーケンスを示しています。

XPm_RequestSuspend、XPm_InitSuspendCb、および XPm_AbortSuspend の詳細は、[付録 K 「XilPM Library v2.5」](#) を参照してください。

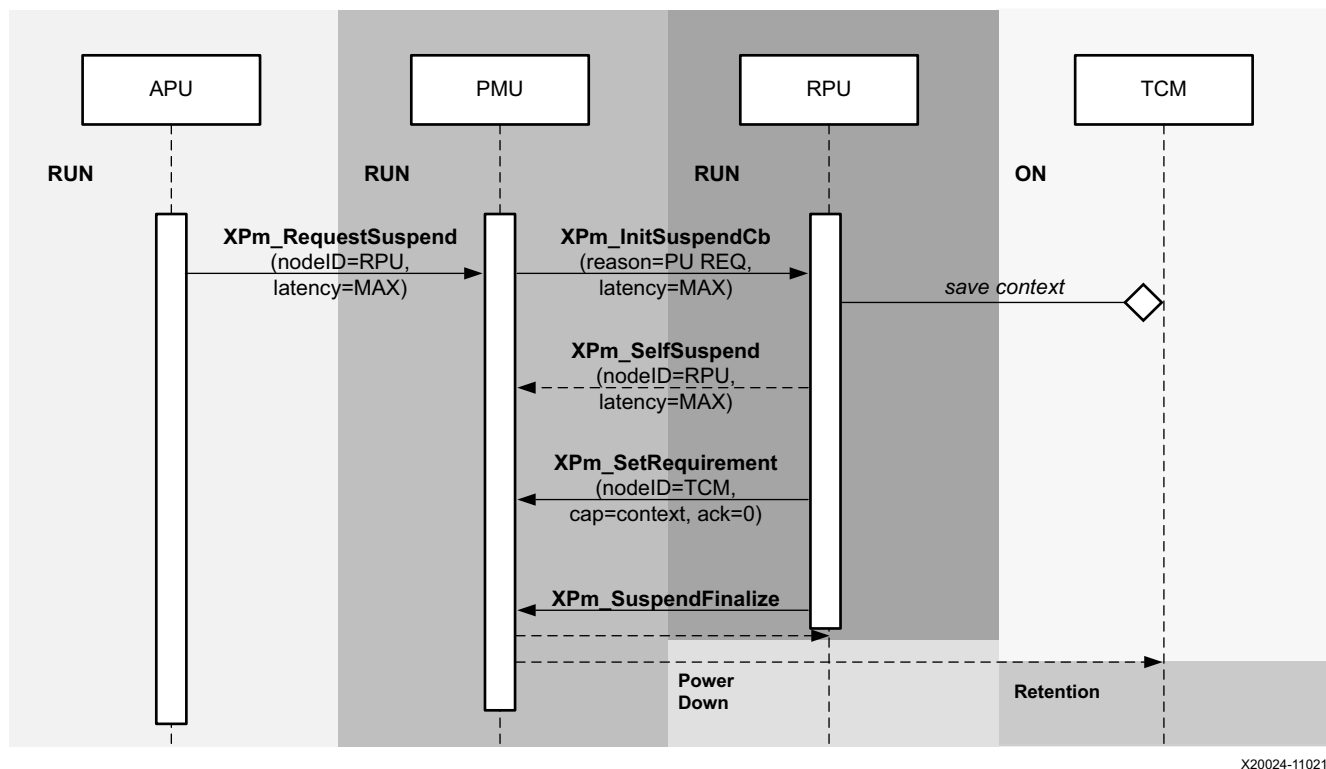


図 11-11: XPm_RequestSuspend の呼び出しで RPU に対して APU がサスペンドを開始

PU をウェークアップさせる

さらに、PU は、XPm_RequestWakeup を呼び出すことによって、その CPU の 1 つまたは別の PU のウェークアップを要求できます。

- 呼び出しを処理するとき、電力管理コントローラーはターゲット CPU または PU を起動します。
- PU がターゲットである場合、この要求によっていずれか 1 つの CPU が起動します。
- 電力管理コントローラーで選択された CPU は、PU 内のプライマリ CPU として見なされます。

次に、ウェークアップ要求の例を示します。

```
XPm_RequestWakeup (NODE_APU_1, REQUEST_ACK_NO);
```

XPm_RequestWakeup の詳細は、[付録 K 「XilPM Library v2.5」](#) を参照してください。

XiIPM の実装の詳細

PM フレームワークのシステム層は、プロセッサ間割り込み (IPI) を使用して Zynq UltraScale+ MPSoC に実装されています。EEMI API 呼び出しを開始する場合、PU は API データ (API ID と引数) を IPI 要求バッファに書き込み、その後、PMU に対して IPI をトリガーします。

PM コントローラーが要求を処理した後、特定の EEMI API および提供された引数に応じて、肯定応答を送信します。

PMU への API 呼び出しのペイロード マッピング

各 EEMI API 呼び出しは、次のデータで識別されます。

- EEMI API 識別子 (ID)
- EEMI API の引数

API 識別子と API 引数の一覧表は、付録 A を参照してください。

PMU への IPI を実行する前に、PU は呼び出しの情報を IPI 要求バッファへ書き込む必要があります。IPI バッファへ書き込まれる各データは 32 ビット ワードです。合計ペイロード サイズは 6 つの 32 ビット ワードで構成されます。EEMI API 識別子用に 1 ワードが予約されており、残りのワードは引数に使用されます。IPI バッファへの書き込みはオフセット 0 から開始します。情報データは、次のようにマッピングされます。

- Word [0] EEMI API ID
- Word [1:5] EEMI API の引数

IPI 応答バッファを使用して、動作ステータスと最大 3 つの値を返します。

- Word [0] サクセス/エラー コード
- Word [1:3] 値 1..3

PMU からの API コールバックのペイロード マッピング

EEMI API には、PM コントローラーによって呼び出され、PU に送信されるコールバック関数が含まれています。

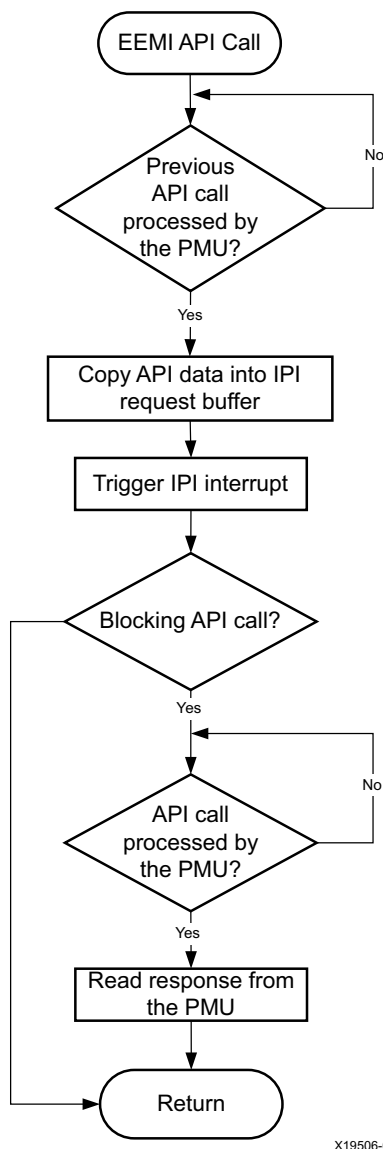
- Word [0] EEMI API Callback ID
- Word [1:5] EEMI API の引数

API 識別子および API の引数の一覧表は、付録 K 「XiIPM Library v2.5」を参照してください。

PMU への EEMI API 呼び出しを実行

PMU への API 呼び出しを実行する前に、PU は前の API 呼び出しが PMU で完了されるまで待機する必要があります。対応する IPI 観察レジスタを読み出すことで、PMU 動作の完了を確認できます。

IPI ペイロード バッファに API データを書き込み、PMU への IPI 割り込みをトリガーすると、API 呼び出しが実行されます。ブロッキング API 呼び出しの場合は、PMU が応答バッファに動作ステータスと最大 3 つの値を書き込んで応答します。PM 動作にエラーが生じた場合に PMU から送信されるすべてのエラーの一覧表は、付録 B を参照してください。応答バッファのデータが有効であることを確認するために、PU は、PMU が API 呼び出し処理を完了するまで待機してから、応答バッファを読み出す必要があります。



X19506-071017

図 11-12: PMU への API 呼び出し実行フローの例

PMU からの API コールバックを処理

PMU は、IPI バッファに API コールバック データを書き込み、PU への IPI 割り込みをトリガーすることによって、PU へコールバック関数を呼び出します。このような割り込み信号を受信するために、PU は IPI ブロックと割り込みコントローラーを適切に初期化する必要があります。すべてのコールバックには 1 つの割り込みがあります。このため、ペイロード バッファの要素 0 には、PU が API コールバックを識別するための API ID が含まれています。したがって、PU はそれぞれの API コールバック関数を呼び出す際には、IPI 要求バッファのロケーション 1 ~ 4 から取得した引数でその関数を渡す必要があります。

このビヘイビアの実装は、XilPM ライブラリにあります。

Linux

Linux は EL1 レベルで実行され、Linux と ATF ソフトウェア層の間の通信は SMC 呼び出しを使用して実行されます。

EEMI API ベースの電力管理機能が Linux カーネルに移植され、Linux で実行される電力管理機能は PMU で提供される EEMI サービスを利用できるようになります。

さらに、デバッグの場合、EEMI API へは `debugfs` を使用して直接アクセス可能です。`debugfs` を使用して EEMI API に直接アクセスすると、カーネルの電力管理操作が妨げられ、予期しない問題が発生する可能性があることに注意してください。

この章で紹介するすべての Linux 電力管理機能は、PetaLinux のデフォルト設定で利用できます。

ユーザー空間の PM インターフェイス

システムの電源ステート

ユーザーは、システムまたはシステム全体の電源ステートを変更できます。PMU によって、システムやサブシステムの新しい電源ステートへの切り替えが容易になります。

シャットダウン

ユーザーは、標準の `shutdown` コマンドで APU サブシステムをシャットダウンできます。

システム全体をシャットダウンするには、APU サブシステムをシャットダウンする前に、その他すべてのサブシステムをシャットダウンする必要があります。たとえば、次のコマンドを実行して PL の電源を遮断します。

```
echo pm_release_node 69 > /sys/kernel/debug/zynqmp-firmware/pm
```

このコマンドを使用して、PL を再びパワーアップします。

```
echo pm_request_node 69 > /sys/kernel/debug/zynqmp-firmware/pm
```

PL サブシステムをシャットダウンする方法は、『Zynq デバイス用 Libmetal および OpenAMP ユーザー ガイド』(UG1186) [参照 16] を参照してください。

リブート (再起動)

ユーザーは `reboot` コマンドを使用して、APU、PS、またはシステムをリセットできます。`reboot` コマンドはデフォルトでシステムをリセットします。

必要に応じて、`reboot` コマンドの範囲を APU または PS に変更できます。

リブート範囲を APU に変更する場合:

```
echo subsystem > /sys/firmware/zynqmp/shutdown_scope
```

リブート範囲を PS に変更する場合:

```
echo ps_only > /sys/firmware/zynqmp/shutdown_scope
```

リブート範囲をシステムに変更する場合:

```
echo system > /sys/firmware/zynqmp/shutdown_scope
```

リセットが完了するとリブート範囲は再びシステムに設定されます。

サスペンド (一時停止)

CPU および大半のペリフェラルがパワーダウンすると、カーネルはサスペンドされます。サスペンド状態から復帰するのに必要なシステム ラン ステートが DRAM に格納されており、セルフリフレッシュ モードに設定されています。

カーネル コンフィギュレーションの要件:

- 電力管理オプション
 - [*] Suspend to RAM and standby
 - [*] User space wakeup sources interface
 - [*] Device power management core functionality
- デバイス ドライバー
 - SoC (システム オンチップ) 固有のドライバー
 - ザイリンクス SoC ドライバー
 - Zynq MPSoC SoC
 - [*] Enable Xilinx Zynq MPSoC Power Management driver
 - [*] Enable Zynq MPSoC generic PM domains
- ファームウェア ドライバー
 - Zynq MPSoC ファームウェア ドライバー
 - *- Enable Xilinx Zynq MPSoC firmware interface

どのデバイスでも、カーネルのサスペンドを防ぐことができます。

https://wiki.archlinux.org/index.php/Power_management/Suspend_and_hibernate も参照してください。

カーネルをサスペンドする場合:

```
$ echo mem > /sys/power/state
```

ウェークアップ ソース

ウェークアップ イベントが生じると、カーネルはサスペンド モードから復帰します。使用可能なウェークアップ ソースを次に示します。

- UART

UART がウェークアップ ソースとして有効になっている場合、UART 入力によってカーネルはサスペンド モードから復帰します。

カーネル コンフィギュレーションの要件:

- 。 「サスペンド (一時停止)」と同じ。

たとえば、UART 入力 で APU を起動する場合:

```
$ echo enabled > /sys/devices/platform/amba/ff000000.serial/tty/ttyPS0/power/wakeup
```

- RTC

RTC がウェークアップ ソースとして有効になっている場合、RTC タイマーが切れたときにカーネルはサスペンド モードから復帰します。RTC ウェークアップ ソースはデフォルトで有効です。

カーネル コンフィギュレーションの要件:

- 。 「サスペンド (一時停止)」と同じ。

たとえば、10 秒後に APU をウェークアップするように RTC を設定するには、次のようにします。

```
$ echo +10 > /sys/class/rtc/rtc0/wakealarm
```

- GPIO

GPIO がウェークアップ ソースとして有効になっている場合、GPIO イベントによってカーネルはサスペンド モードから復帰します。

カーネル コンフィギュレーションの要件:

- 。 デバイス ドライバー

- 入力デバイス サポート、[*]

一般的な入力層 (キーボード、マウスなどに必要)(INPUT [=y])

[*] Keyboards (INPUT_KEYBOARD [=y])

[*] GPIO Buttons (CONFIG_KEYBOARD_GPIO=y)

[*] Polled GPIO buttons

たとえば、GPIO ピンで APU を起動する場合:

```
$ echo enabled > /sys/devices/platform/gpio-keys/power/wakeup
```

CPU の電力管理

CPU ホットプラグ

ユーザーは、CPU ホットプラグ制御インターフェイスを介して、必要に応じて 1 つまたは複数の APU コアをオンラインおよびオフラインで使用できます。

カーネル コンフィギュレーションの要件:

- カーネルの機能
 - [*] Support for hot-pluggable CPUs

関連資料

- <https://www.kernel.org/doc/Documentation/cpu-hotplug.txt>
- <http://lxr.free-electrons.com/source/Documentation/devicetree/bindings/arm/idle-states.txt>

たとえば、CPU3 をオフラインで使用する場合:

```
$ echo 0 > /sys/devices/system/cpu/cpu3/online
```

CPU アイドル

有効にすると、APU コアがアイドル状態の時にカーネルは個々の APU コアの電力を削減できます。

カーネル コンフィギュレーションの要件:

- CPU 電力管理
 - CPU アイドル
 - [*] CPU idle PM support
 - Arm CPU アイドル ドライバー
- [*] Generic Arm/Arm64 CPU idle Driver

関連資料

- <https://www.kernel.org/doc/Documentation/cpuidle/core.txt>
- <https://www.kernel.org/doc/Documentation/cpuidle/driver.txt>
- <https://www.kernel.org/doc/Documentation/cpuidle/governor.txt>
- <https://www.kernel.org/doc/Documentation/cpuidle/sysfs.txt>

cpuidle の sysfs インターフェイスは次のとおりです。

```
$ ls -lR /sys/devices/system/cpu/cpu0/cpuidle/

/sys/devices/system/cpu/cpu0/cpuidle/:
drwxr-xr-x  2 root    root          0 Jun 10 21:55 state0
drwxr-xr-x  2 root    root          0 Jun 10 21:55 state1

/sys/devices/system/cpu/cpu0/cpuidle/state0:
-r--r--r--  1 root    root          4096 Jun 10 21:55 desc
-rw-r--r--  1 root    root          4096 Jun 10 21:55 disable
-r--r--r--  1 root    root          4096 Jun 10 21:55 latency
-r--r--r--  1 root    root          4096 Jun 10 21:55 name
-r--r--r--  1 root    root          4096 Jun 10 21:55 power
-r--r--r--  1 root    root          4096 Jun 10 21:55 residency
-r--r--r--  1 root    root          4096 Jun 10 21:55 time
-r--r--r--  1 root    root          4096 Jun 10 21:55 usage

/sys/devices/system/cpu/cpu0/cpuidle/state1:
-r--r--r--  1 root    root          4096 Jun 10 21:55 desc
-rw-r--r--  1 root    root          4096 Jun 10 21:55 disable
-r--r--r--  1 root    root          4096 Jun 10 21:55 latency
-r--r--r--  1 root    root          4096 Jun 10 21:55 name
-r--r--r--  1 root    root          4096 Jun 10 21:55 power
-r--r--r--  1 root    root          4096 Jun 10 21:55 residency
-r--r--r--  1 root    root          4096 Jun 10 21:55 time
-r--r--r--  1 root    root          4096 Jun 10 21:55 usage
```

説明:

- desc: アイドル ステートに関する簡単な説明 (文字列)
- disable: このアイドル ステートを無効にするためのオプション (ブール型)
- latency: このアイドル ステートを終了するためのレイテンシ (マイクロ秒)
- name: アイドル ステートの名称 (文字列)
- power: このアイドル ステート中の消費電力 (ミリワット)
- time: このアイドル ステートの合計時間 (マイクロ秒)
- usage: このステートに遷移した回数 (カウント)

cpuidle ガバナーの sysfs インターフェイスは次のとおりです。

```
$ ls -lR /sys/devices/system/cpu/cpuidle/

/sys/devices/system/cpu/cpuidle/:
-r--r--r--  1 root    root          4096 Jun 10 21:55 current_driver
-r--r--r--  1 root    root          4096 Jun 10 21:55 current_governor_ro
```


CPU 周波数

有効の場合、CPU コアは異なる動作クロック周波数間で切り替えることができます。

カーネル コンフィギュレーションの要件:

- CPU 周波数スケーリング
 - [*] CPU Frequency scaling
 - デフォルト CPUFreq ガバナー
 - Userspace
- CPU 電力管理
 - [*] CPU Frequency scaling
 - デフォルト CPUFreq ガバナー
 - Userspace
 - <*> Generic DT based cpufreq driver

利用可能な CPU スピードを確認:

```
$ cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_cpu_freq
```

CPU 周波数制御に `userspace` ガバナーを選択:

```
$ echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

現在の CPU スピードを確認 (すべてのコアで同じ):

```
$ cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_cpu_freq
```

CPU スピードを変更 (すべてのコアで同じ):

```
$ echo <freq> > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

CPU 周波数の追加および変更の詳細は、[Linux カーネル資料の「Generic OPP Bindings」\(英語\)](#) を参照してください。

デバイスの電力管理

クロック ゲーティング

使用していないときにデバイスのクロックを停止する (Common Clock Framework と呼ばれている)

カーネル コンフィギュレーションの要件:

- 共通クロック フレームワーク
 - [*] Support for Xilinx ZynqMP Ultrascale+ clock controllers

ランタイム PM

使用していないときにデバイスの電源をオフにする。個々のドライバーは、ランタイム電力管理 (PM) をサポートしていない場合があることに注意してください。

カーネル コンフィギュレーションの要件:

- 電力管理オプション
 - [*] Suspend to RAM and standby
- デバイス ドライバー
 - SoC (システム オンチップ) 固有のドライバー
 - [*] Xilinx Zynq MPSoC driver support

グローバル汎用ストレージ レジスタ

一般的な用途に使用できる 32 ビット ストレージレジスタが 4 つあります。これらの値は、ソフトウェア リブートによって失われます。表 11-1 に、グローバル汎用ストレージレジスタを示します。

表 11-1: グローバル汎用ストレージレジスタ

デバイス ノード	MMIO レジスタ	MMIO アドレス	有効な値の範囲
/sys/firmware/zynqmp/ggs0	GLOBAL_GEN_STORAGE0	0xFFD80030	0x00000000 - 0xFFFFFFFF
/sys/firmware/zynqmp/ggs1	GLOBAL_GEN_STORAGE1	0xFFD80034	0x00000000 - 0xFFFFFFFF
/sys/firmware/zynqmp/ggs2	GLOBAL_GEN_STORAGE2	0xFFD80038	0x00000000 - 0xFFFFFFFF
/sys/firmware/zynqmp/ggs3	GLOBAL_GEN_STORAGE3	0xFFD8003C	0x00000000 - 0xFFFFFFFF

グローバル ストレージ レジスタの値を読み出す場合:

```
$cat /sys/firmware/zynqmp/ggs0
```

グローバル ストレージ レジスタのマスクおよび値を書き込む場合:

```
$echo 0xFFFFFFFF 0x1234ABCD > /sys/firmware/zynqmp/ggs0
```

永続的なグローバル汎用ストレージ レジスタ

一般的な用途に使用できる永続的な 32 ビットグローバル ストレージレジスタが 4 つあります。これらの値は、ソフトウェア リブートの後も失われません。表 11-2 に、永続的なグローバル汎用ストレージレジスタを示します。

表 11-2: 永続的なグローバル汎用ストレージレジスタ

デバイス ノード	MMIO レジスタ	MMIO アドレス	有効な値の範囲
/sys/firmware/zynqmp/pggs0	PERS_GLOB_GEN_STORAGE0	0xFFD80050	0x00000000 - 0xFFFFFFFF
/sys/firmware/zynqmp/pggs1	PERS_GLOB_GEN_STORAGE1	0xFFD80054	0x00000000 - 0xFFFFFFFF
/sys/firmware/zynqmp/pggs2	PERS_GLOB_GEN_STORAGE2	0xFFD80058	0x00000000 - 0xFFFFFFFF
/sys/firmware/zynqmp/pggs3	PERS_GLOB_GEN_STORAGE3	0xFFD8005C	0x00000000 - 0xFFFFFFFF

永続的なグローバル ストレージ レジスタの値を読み出す場合:

```
$cat /sys/firmware/zynqmp/pggs0
```

永続的なグローバル ストレージ レジスタのマスクおよび値を書き込む場合:

```
$echo 0xFFFFFFFF 0x1234ABCD > /sys/firmware/zynqmp/pggs0
```

デモ

PetaLinux のビルド済みイメージには、いくつかの簡単な電力管理タスクを実行するデモ スクリプトが用意されています。

- システム サスペンド
- CPU ホットプラグ
- CPU 周波数
- システム リブート
- システム シャットダウン

デモを開始するには、次のコマンドを実行します。

```
$ hellopm
```

デバッグ インターフェイス

PM プラットフォーム ドライバーは、すべての EEMI サービスへアクセスできるように標準の `debugfs` インターフェイスをエクスポートします。このインターフェイスはテスト専用であり、不適切な使用や、引数の数、タイプ、有効範囲などをチェックする機能はありません。このインターフェイスを使用して EEMI サービスを直接呼び出すと、カーネルの電力管理動作を妨害してしまい、結果として予期しない動作やシステム クラッシュを招く可能性があることに注意してください。デフォルトでは、ZynqMP の `debugfs` インターフェイスは、`defconfig` で無効になっています。これは、次に示す手順で明示的に有効にする必要があります。

カーネル コンフィギュレーションの要件 (この順番):

- カーネル ハッキング
 - コンパイル時間チェックとコンパイラ オプション
 - `[*] Debug Filesystem`
- ファームウェア ドライバー
 - Zynq MPSoC ファームウェア ドライバー
 - `[*] Enable Xilinx Zynq MPSoC firmware interface`
 - `[*] Enable Xilinx Zynq MPSoC firmware debug APIs`

次を除き、任意の EEMI API を呼び出すことができます。

- セルフ サスペンド (Self Suspend)
- システム シャットダウン (System Shutdown)
- APU のパワーダウン強制 (Force Power Down the APU)
- APU のウェークアップ要求 (Request Wake-up the APU)

コマンド ライン入力

ユーザーは、debugfs インターフェイス ノードに EEMI API ID と最大 4 つの引数を書き込むことで、EEMI サービスを呼び出すことができます。

API ID

関数 ID は、EEMI API 関数の名称または ID 番号であり、それぞれに文字列または整数を入力します。

引数

引数の数や型は、選択した API 関数に依存します。すべての引数は整数型で指定し、EEMI 引数リストにあるその特定引数の型の序数になる必要があります。関数の説明、引数の型、および引数の数の詳細は、EEMI API の仕様を参照してください。

例

NODE_USB_0 の request_node API 呼び出しを実行する方法を示しています。

```
$ echo "pm_request_node 22 1 100 1" > /sys/kernel/debug/zynqmp-firmware/pm
```

コマンド リスト

Get API Version

API バージョンを取得します。

```
$ echo pm_get_api_version > /sys/kernel/debug/zynqmp-firmware/pm
```

Request Suspend

別の PU にセルフ サスペンドするように要求します。

```
$ echo pm_request_suspend <node> > /sys/kernel/debug/zynqmp-firmware/pm
```

Self Suspend

この PU がセルフ サスペンドすることを PMU に通知します。

```
$ echo pm_self_suspend <node> > /sys/kernel/debug/zynqmp-firmware/pm
```

Force Power Down

別の PU の電源を強制的にオフにします。

```
$ echo pm_force_powerdown <node> > /sys/kernel/debug/zynqmp-firmware/pm
```

Abort Suspend

サスペンド処理が中断されたことを PMU に通知します。

```
$ echo pm_abort_suspend > /sys/kernel/debug/zynqmp-firmware/pm
```

Request Wake-up

別の PU に対して、サスペンド状態から復帰することを要求します。

```
$ echo pm_request_wakeup <node> <set_address> <address> >
/sys/kernel/debug/zynqmp-firmware/pm
```

Set Wake-up Source

ノードをウェークアップ ソースとして設定します。

```
$ echo pm_set_wakeup_source <target> <wkup_node> <enable> >
/sys/kernel/debug/zynqmp-firmware/pm
```

Request Node

ノードの使用を要求します。

```
$ echo pm_request_node <node> > /sys/kernel/debug/zynqmp-firmware/pm
```

Release Node

使用しないノードをリリースします。

```
$ echo pm_release_node <node> > /sys/kernel/debug/zynqmp-firmware/pm
```

Set Requirement

ノード上の電源要件を設定します。

```
$ echo pm_set_requirement <node> <capabilities> > /sys/kernel/debug/zynqmp-firmware/pm
```

Set Max Latency

ノードの最大ウェークアップ レイテンシ要件を設定します。

```
$ echo pm_set_max_latency <node> <latency> > /sys/kernel/debug/zynqmp-firmware/pm
```

Get Node Status

ノードのステータス情報を取得します。(ノードの割り当てにかかわらず、すべての PU が任意のノードのステータスを確認できます。)

```
$ echo pm_get_node_status <node> > /sys/kernel/debug/zynqmp-firmware/pm
```

Get Operating Characteristic

ノードの動作特性を取得します。

```
$ echo pm_get_operating_characteristic <node> > /sys/kernel/debug/zynqmp-firmware/pm
```

Reset Assert

特定のリセット ラインでアサート/ディアサートします。

```
$ echo pm_reset_assert <reset> <action> > /sys/kernel/debug/zynqmp-firmware/pm
```

Reset Get Status

リセット ラインのステータスを取得します。

```
$ echo pm_reset_get_status <reset> > /sys/kernel/debug/zynqmp-firmware/pm
```

Get Chip ID

チップ ID を取得します。

```
$ echo pm_get_chipid > /sys/kernel/debug/zynqmp-firmware/pm
```

Get pin control functions

指定したピンに対して現在選択している機能を取得します。

```
$ echo pm_pinctrl_get_function <pin-number> > /sys/kernel/debug/zynqmp-firmware/pm
```

Set pin control functions

指定したピンに要求された機能を設定します。

```
$ echo pm_pinctrl_set_function <pin-number> <function-id> >
/sys/kernel/debug/zynqmp-firmware/pm
```

Get configuration parameters for the pin

指定したピンに対して要求された設定パラメーターの値を取得します。

```
$ echo pm_pinctrl_config_param_get <pin-number> <parameter to get> >
/sys/kernel/debug/zynqmp-firmware/pm
```

Set configuration parameters for the pin

指定したピンに対して要求された設定パラメーターの値を設定します。

```
$ echo pm_pinctrl_config_param_set <pin-number> <parameter to set> <param value> >
/sys/kernel/debug/zynqmp-firmware/pm
```

Control device and configurations

デバイスと設定を制御し、設定値を取得します。

```
$ echo pm_ioctl <node id> <ioctl id> <arg1> <arg2> >
/sys/kernel/debug/zynqmp-firmware/pm
```

Query Data

ファームウェアからのデータを要求します。

```
$ echo pm_query_data <query id> <arg1> <arg2> <arg3> >
/sys/kernel/debug/zynqmp-firmware/pm
```

Enable Clock

指定したクロック ノード ID のクロックを有効にします。

```
$ echo pm_clock_enable <clock id> > /sys/kernel/debug/zynqmp-firmware/pm
```

Disable Clock

指定したクロック ノード ID のクロックを無効にします。

```
$ echo pm_clock_disable <clock id> > /sys/kernel/debug/zynqmp-firmware/pm
```

Get Clock State

指定したクロック ノード ID のクロック ステートを取得します。

```
$ echo pm_clock_getstate <clock id> > /sys/kernel/debug/zynqmp-firmware/pm
```

Set Clock Divider

指定したクロック ノード ID のクロック分周値を設定します。

```
$ echo pm_clock_setdivider <clock id> <divider value> >
/sys/kernel/debug/zynqmp-firmware/pm
```

Get Clock Divider

指定したクロック ノード ID のクロック分周値を取得します。

```
$ echo pm_clock_getdivider <clock id> > /sys/kernel/debug/zynqmp-firmware/pm
```

Set Clock Rate

指定したクロック ノード ID のクロック レートを設定します。

```
$ echo pm_clock_setrate <clock id> <clock rate> >
/sys/kernel/debug/zynqmp-firmware/pm
```

Get Clock Rate

指定したクロック ノード ID のクロック レートを取得します。

```
$ echo pm_clock_getrate <clock id> > /sys/kernel/debug/zynqmp-firmware/pm
```

Set Clock Parent

指定したクロック ノード ID の親クロックを設定します。

```
$ echo pm_clock_setparent <clock id> <parent clock id> >
/sys/kernel/debug/zynqmp-firmware/pm
```

Get Clock Parent

指定したクロック ノード ID の親クロックを取得します。

```
$ echo pm_clock_getparent <clock id> > /sys/kernel/debug/zynqmp-firmware/pm
```

注記: クロック ID の定義は、クロック バインド資料の次のテキスト ファイルにあります。

Documentation/devicetree/bindings/clock/xlnx, zynqmp-clk.txt

PM プラットフォーム ドライバー

Linux 用 Zynq UltraScale+ MPSoC 電力管理機能は、電力管理ドライバー、電力ドメイン ドライバーおよびプラットフォーム ファームウェア ドライバーにカプセル化されています。システムレベルの API 関数はエクスポートされるため、GPL 互換ライセンスを使用した別の Linux モジュールで呼び出すことが可能です。関数宣言は、次の場所にあります。

```
include/linux/firmware/xilinx/zynqmp/firmware.h
```

実装した関数は、次の場所にあります。

```
drivers/firmware/xilinx/zynqmp/firmware*.c
```

ドライバーを正しく初期化するには、Linux デバイス ツリーに正確なノードを追加する必要があります。ファームウェア ドライバーは、「firmware」ノードを目印にして PMU ファームウェアの存在を検出し、PM フレームワーク ファームウェア層への呼び出しメソッド (「smc」または「hvc」) を決定し、コールバック割り込み番号をレジスタに格納します。

「firmware」ノードには、次のプロパティが含まれます。

- **Compatible:** 「xlnx, zynqmp-firmware」を含む必要があります。
- **Method:** PM フレームワーク ファームウェアを呼び出すメソッド。「smc」とします。

注記: その他の情報は、Linux 資料の次のテキスト ファイルにあります。

```
Documentation/devicetree/bindings/firmware/xilinx/xlnx, zynqmp-firmware.txt
```

例:

```
firmware {
    zynqmp_firmware: zynqmp-firmware {
        compatible = "xlnx, zynqmp-firmware";
        method = "smc";
    };
};
```

注記: 電力ドメイン ドライバーと電力管理ドライバーのバインドの詳細は、Linux 資料の次のファイルを参照してください。

```
Documentation/devicetree/bindings/soc/xilinx/xlnx, zynqmp-power.txt
```

```
Documentation/devicetree/bindings/power/zynqmp-genpd.txt
```

注記: 次の EEMI API は、xilPM ではサポートされません。現在のリリースでは、これらは Linux から ATF を介してのみ利用できます。

- query_data
- ioctl
- clock_enable
- clock_disable
- clock_getstate
- clock_setdivider
- clock_getdivider
- clock_setrate

- clock_getrate
- clock_setparent
- clock_getparent
- pinctrl_request
- pinctrl_release
- pinctrl_set_function
- pinctrl_get_function
- pinctrl_set_config
- pinctrl_get_config

Arm トラステッド ファームウェア (ATF)

Arm Trusted Firmware (ATF) は EL3 で実行されます。EEMI API をサポートし、IPI ベースの通信手段を介して、PMU へ PM 要求を送信することによってスレーブ ノードの電源ステートを管理します。

ATF アプリケーション バイナリ インターフェイス

APU で実行可能な EL3 以下のすべての層は、ATF を介して PMU と間接的に通信可能です。ATF は、下位層 EL からのすべての呼び出しを受信し、すべての要求を統合して、PMU にそれらを送信します。

Arm の SMC 呼び出し規約に従うため、非セキュア ワールドから ATF への PM 通信は、呼び出し規約で指定されているように事前定義された SMC 関数識別子と SMC サブレンジ所有権を使用して SiP のサービス コールとして実行されます。

APU 用 EEMI API の実装は、SMC64 呼び出し規約にのみ準拠していることに注意してください。

OS またはハイパーバイザー ソフトウェア レベルで作成された EEMI API 呼び出しは、32 ビットの API ID を SMC 関数識別子として渡し、最大 4 つの 32 ビット引数も渡します。PM の引数はすべて 32 ビット値であるため、2 つのペアが結合されて 1 つの 64 ビット値を構成します。

ATF は最大 5 つの 32 ビット戻り値を返します。

- 処理完了またはエラーとその理由のいずれかを示すステータスを返す
- PM コントローラーからのその他の情報

API バージョンの確認

EEMI API を使用してスレーブ ノードを管理する前に、ATF に実装されている EEMI API バージョンが PMU ファームウェアに実装されているバージョンと一致していることを確認する必要があります。EEMI API バージョンは、メジャー バージョンを示す上位 16 ビットとマイナー バージョンを示す下位 16 ビットに分離された 32 ビット値です。これら両方のフィールドが ATF と PMU ファームウェアで同じになる必要があります。

EEMI API バージョンの確認方法

ATF に実装されている EEMI バージョンは、ローカルの EEMI_API_VERSION フラグに定義されています。リッチ OS の場合、PM_GET_API_VERSION 関数を呼び出すことで PMU から EEMI API バージョンを取得できることがあります。これらのバージョンが異なる場合、この呼び出しでエラーがレポートされます。

注記: この EEMI API 呼び出しはバージョンに依存しません。つまり、すべての EEMI バージョンにこの呼び出しが実装されています。

チップ ID の確認

Linux などのリッチ OS は、SMC 経由で PM_GET_CHIPID 関数を呼び出すことで PMU からチップ ID 情報を取得できます。

戻り値は次のとおりです。

1. CSU idcode レジスタ (TRM 参照)。
2. CSU version レジスタ (TRM 参照)。

詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] を参照してください。

PSCI

PSCI (Power State Coordination Interface) は、サスペンド、シャットダウン、リブートなど、Arm プロセッサのシステム電源ステータスを制御するための標準インターフェイスです。PSCI 仕様の詳細は、<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0022c/index.html> を参照してください。

ATF が Linux からの PSCI 要求を処理します。ATF は PSCI v0.2 のみをサポートしています (v0.1 の下位互換性サポートなし)。

Linux カーネルには PSCI の標準サポートが付随します。カーネルと ATF/PSCI のバインドに関する情報は、<https://www.kernel.org/doc/Documentation/devicetree/bindings/arm/psci.txt> を参照してください。

表 11-3: ATF でサポートされる PSCI v0.2 の関数

関数	説明	サポート
PSCI Version	実装された PSCI のバージョンを返す。	あり
CPU Suspend	コアまたは上位レベル トポロジのノードで実行をサスペンド。 この呼び出しは、コアがウェークアップ イベントによって復帰することになっているアイドル状態のサブシステムで使用することを目的とする。	あり

表 11-3: ATF でサポートされる PSCI v0.2 の関数 (続き)

関数	説明	サポート
CPU On	コアをパワーアップする。次のいずれかのコアをパワーアップするために使用。 <ul style="list-style-type: none"> 呼び出し元の監視ソフトウェアをまだ起動していない。 CPU_OFF 呼び出しで、前もって電源オフにされている。 	あり
CPU Off	呼び出し元のコアの電源をオフにする。この呼び出しは、ホットプラグ状態で使用することを目的とする。CPU_OFF によって電源をオフにするコアは、CPU_ON のみに応答して再びパワーアップできる。	あり
Affinity Info	呼び出し元がアフィニティ インスタンスのステータスを要求できる。	あり
Migrate (オプション)	ユニプロセッサのトラステッド OS に、コンテキストを特定コアに移行するように要求する。	あり
Migrate Info Type (オプション)	呼び出し元がトラステッド OS に存在するマルチコア サポートのレベルを識別可能になる。	あり
Migrate Info Up CPU (オプション)	ユニプロセッサのトラステッド OS の場合、この関数は現在の常駐コアを返す。	あり
System Off	システムをシャットダウンする。	あり
System Reset	システムをリセットする。	あり
PSCI Features	PSCI v1.0 で導入。 クエリ API を使用して、特定の PSCI 関数が実装されているかどうか、またその機能情報について API に問い合わせる。	あり
CPU Freeze (オプション)	PSCI v1.0 で導入。 コアを IMPLEMENTATION DEFINED 低電力ステートにする。 CPU_OFF とは異なり、コアに割り込みをターゲットするには有効。ただし、CPU_ON コマンドが発行されるまで、コアは低電力状態を維持する必要がある。	なし
CPU Default Suspend (オプション)	PSCI v1.0 で導入。 コアを IMPLEMENTATION DEFINED 低電力ステートにする。 CPU_SUSPEND とは異なり、呼び出し元が電力ステート パラメーターを指定する必要はない。	なし
Node HW State (オプション)	PSCI v1.0 で導入。 この関数は、システムの電力ドメイン トポロジ内のノードの真の HW ステートを返すことを目的とする。	あり
System Suspend (オプション)	PSCI v1.0 で導入。 RAM にサスペンドを実装する。最も低い電力ステートにする CPU_SUSPEND と同等。	あり
PSCI Set Suspend Mode (オプション)	PSCI v1.0 で導入。 この関数を使用して、CPU_SUSPEND で使用されるモードを設定して電源ステートを調整できる。	なし

表 11-3: ATF でサポートされる PSCI v0.2 の関数 (続き)

関数	説明	サポート
PSCI Stat Residency (オプション)	PSCI v1.0 で導入。 コールド ブート後にプラットフォームが指定された電源ステートで費 やした時間を返す。	あり
PSCI Stat Count (オプション)	PSCI v1.0 で導入。 コールド ブート後にプラットフォームが指定された電源ステートを使 用した回数を返す。	あり

PMU ファームウェア

PMU ファームウェアには、各種サービスを実行するモジュールの 1 つとして、EEMI サービス ハンドラーを実装した PM コントローラー モジュールがあります。詳細は、[第 10 章「プラットフォーム管理ユニット ファームウェア」](#)を参照してください。

電力管理イベント

PM コントローラーはイベント駆動型であるため、すべての操作は次のいずれかのイベントによってトリガーされます。

- IPI0 割り込みでトリガーされる EEMI API イベント。
- GPI1 割り込みでトリガーされるウェークアップ イベント。
- GPI2 割り込みでトリガーされるスリープ イベント。
- PIT2 割り込みでトリガーされるタイマー イベント。

EEMI API イベント

EEMI API イベントは、ソフトウェアで生成されるイベントです。PM マスターが PMU への EEMI API 呼び出しを開始すると、IPI 割り込みによってイベントがトリガーされます。PM コントローラーは EEMI 要求を処理し、肯定応答を返すことができます (要求されている場合)。一部の例外を除き、EEMI 要求は通常、ノードまたはマスターの電源ステートの変更をトリガーします。

ウェークアップ イベント

ウェークアップ イベントは、ハードウェアで生成されるイベントです。これらはペリフェラル信号でトリガーされ、PM マスターを復帰させる必要があることを示します。すべてのウェークアップ イベントは GPI1 割り込みでトリガーされます。

PM コントローラーでサポートされるウェークアップ イベントを次に示します。

- 関連する GIC インターフェイスのハードウェア リソースによってトリガーされる割り込みが発生し、CPU を復帰させる必要があることを通知する GIC ウェークアップ イベント。サポートされる GIC ウェークアップ イベントを次に示します。
 - APU[3:0] 各 APU プロセッサに 1 つのイベント
 - RPU[1:0] 各 RPU プロセッサに 1 つのイベント
- GIC プロキシで指示される FPD ウェークアップ イベント。このウェークアップ イベントは、ウェークアップ ソースがサスペンドされる前に有効になったときにトリガーされます。このイベントの目的は、FPD の電源がオフになっているときに APU マスターのウェークアップをトリガーすることです。FPD の電源がオフになっていない場合は、いずれのウェークアップ 信号も FPD ウェークアップ を使用して伝搬されません。代わりに、GIC での関連する割り込みが適切に有効になっている場合には、GIC ウェークアップ を使用してウェークアップ 信号を伝搬できます。RPU をターゲットとするすべてのウェークアップ 信号は、関連する GIC ウェークアップ を使用して伝搬されます。

スリープ イベント

スリープ イベントは、ソフトウェアで生成されるイベントです。サスペンド処理を完了した後に CPU によってトリガーされるイベントであり、低電力ステートに遷移する準備が整ったことを PMU に通知する目的があります。すべてのスリープ イベントは GPI2 割り込みによってトリガーされます。

サポートされるスリープ イベントを次に示します。

- APU[3:0] 各 APU プロセッサに 1 つのイベント
- RPU[1:0] 各 RPU プロセッサに 1 つのイベント

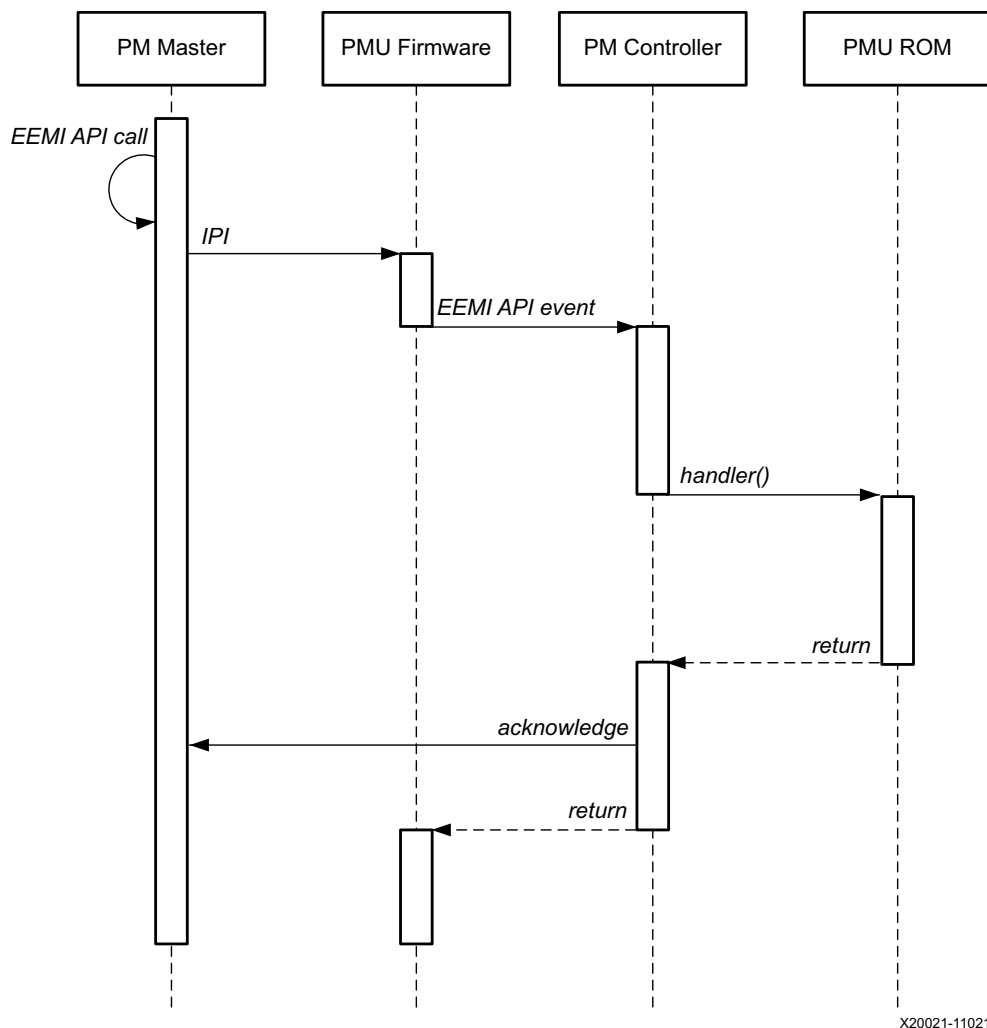
PM コントローラーが特定の CPU のスリープ イベントを受信すると、その CPU は低電力ステートになります。

タイマー イベント

タイマー イベントは、ハードウェアで生成されるイベントです。一定の時間が経過すると、ハードウェア タイマーによってトリガーされます。このイベントは電力管理タイムアウト アカウンティングに使用され、PIT2 割り込みでトリガーされます。

EEMI API 呼び出しの一般的フロー

次の図は、PM マスター (別の PU など) で開始される呼び出しから始まる、標準的な API 呼び出しのシーケンスを示しています。



X20021-110217

図 11-13: EEMI API 呼び出しシーケンス

上記の図には 4 つのアクターがあり、1 番目は PM マスター (RPU、APU、MicroBlaze™ プロセッサ コアなど) です。残りの 3 つのアクターは PMU の異なるソフトウェア層となります。

まず最初に PMU ファームウェアが IPI 割り込みを受信します。この割り込みが電力管理関連の割り込みとして認識されると、電力管理モジュールに IPI 引数が渡されます。その後、PM コントローラーが API 呼び出しを処理します。必要に応じて、PMU ROM を呼び出して、電源アイランドや電力ドメインのパワーオン/パワーオフなどの電力管理動作を実行します。

リセット

はじめに

Zynq® UltraScale+™ MPSoC デバイスのリセット ブロックはシステムに対する外部および内部リセット入力を処理し、すべてのペリフェラル、APU および RPU に対するリセット要件が満たされるようにします。リセット ブロックはデバイスのプログラマブル ロジックに対してリセットを生成し、PS ブロックと PL ブロックに対して個別にリセットのアサートが可能です。

この章では、システム レベル リセットおよびモジュール単位のリセットの仕組みについて説明します。

システム レベル リセット

Zynq UltraScale+ MPSoC デバイスは、APU や RPU などのブロック単位、または FPD や LPD といった電源ドメイン単位での個別リセットが可能です。システム レベル リセットには、次に示す複数の方法があります。

- パワーオン リセット (POR)
- システム リセット (SRST_B)
- デバッグ システム リセット

システム レベル リセット フローの詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「リセット システム」の章を参照してください。

ブロック レベル リセット

PS のみのリセットはシステム リセットのサブセットとして実装できますが、PS のみのリセットが開始される前に PS-PS 間の AXI トランザクションを正しい手順で終了させるためのソフトウェアをユーザーが提供する必要があります。

PS 専用のリセット

PS 専用のリセットは、PL を動作させたまま PL を再起動します。このリセットは、ハードウェアのエラー信号やソフトウェアによるレジスタ書き込みによってトリガーできます。エラー信号によって PS 専用のリセットが発生した場合、PL にもそのエラーを通知できます。これにより、PL は PR の開始準備ができます。

PS 専用のリセット シーケンスは、次の方法で実装できます。

- 。 [ErrorLogic] PS のみのリセットが必要となるようなエラー割り込みをアサートします。この要求は PMU に割り込みとして送信されます。
- 。 [PMU-FW] PMU エラー (=>PS のみのリセット) をセットして PL に通知します。

PS 専用のリセット シーケンスの詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「リセット システム」の章の「PS のみのリセット」を参照してください。

APU のリセット

APU の各 CPU コアは、ソフトウェアで個別にリセットできます。

APU MPCore リセットは FPD、WDT、またはソフトウェアによるレジスタ書き込みでトリガーできます。ただし、APU MPCore は APU への要求および APU からの要求を正しい手順で終了せずにリセットされます。これは、重大な障害が発生した場合は FPD リセットを使用するためです。APU のリセットは、主にソフトウェア デバッグに使用します。

APU リセット シーケンスの詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] を参照してください。

APU 専用のリセット

APU 専用のリセットは、qspi24、qspi32、sd0、sd1、sd-ls ブート モードでサポートされます。ただし、フラッシュ容量が 16MB を超えるシステムの場合、qspi24 モードで APU 専用のリセットはサポートされません。

RPU のリセット

Cortex™-R5F の各コアは、個別にリセットできます。ロックステップ モードでは、Cortex-R5F_0 のみをリセットすると両方の Cortex-R5F コアがリセットされます。このリセットは、エラーまたはソフトウェアによるレジスタ書き込みでトリガーできます。ロックステップ エラーで Cortex-R5F リセットをトリガーして、RPU をリセットおよび再起動できます。

Cortex-R5F のリセットを開始する前に、その Cortex-R5F のイングレス (入力) およびイーグレス (出力) トランザクションを正しい手順で終了する必要があります。

FPD のリセット

FPD リセットは、フル電力ドメイン (FPD) すべてをリセットし、エラーまたはソフトウェアによるレジスタ書き込みによってトリガーできます。エラー信号によって FPD リセットが発生した場合、LPD と PL の両方にエラーを通知する必要があります。

FPD リセットは FPD パワーアップ シーケンスを利用して実装できます。ただし、FPD のリセットを開始する前に FPD のイングレス (入力) およびイグレス (出力) AXI トランザクションを正しい手順で終了する必要があります。FPD リセット シーケンスは PL リセットになります。

Zynq UltraScale+ MPSoC デバイスには PMU ブロックからの汎用出力ピンがあり、これを使用して PL 内のブロックをリセットできます。また、EMIO インターフェイスを使用した GPIO でも PL 内のブロックをリセットできます。リセットフローの詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』[参照 11] の「リセット システム」の章を参照してください。

リセット用ソフトウェア API の詳細は、第 9 章「プラットフォーム管理」の「PMU ファームウェア」を参照してください。

ウォーム リスタート

Zynq UltraScale+ MPSoC のシリコンは非常に複雑で、チップ上で複数のサブシステムを実行できます。このため、Zynq UltraScale+ MPSoC は多くの種類のリセットをサポートしています。これには、最もシンプルなシステム リセットから、非常に複雑なサブシステム リスタートまでさまざまなものがあります。プロセッサ コンポーネントとプログラマブル ロジック コンポーネントを含むシステムまたはサブシステムをリセットする際は、ハードウェアとソフトウェアの両方をリセットする必要があります。ハードウェアに対するリセットでは、次を実行します。

- プロセッサ、およびそのシステム/サブシステムに関連するペリフェラルをリセットする
- 必要に応じてメモリをクリーンアップする
- インターコネクトがクリーンなステートにあり、トラフィックの転送が可能であることを確認する

ソフトウェア リセットを実行すると、プロセッサはリセット ベクターから動作を開始します。ただし、システムをクリーンな実行ステートに戻すには、システム/サブシステムに対する有効でクリーンなコードをリセット ベクターに格納しておく必要があります。

Zynq UltraScale+ MPSoC に対するリセットは、大きく次の 2 つのカテゴリに分類されます。

- システム全体のリセット
- サブシステムのリスタート

システム全体のリセットには、次のものがあります。

- パワーオン リセット (POR)
- システム リセット
- PS 専用のリセット

サブシステムのリスタートには、APU サブシステムのリスタートと RPU サブシステムのリスタートがあります。

システム全体のリセットは、非常に単純です。ハードウェアがリセット ステートに戻され、ソフトウェアが ROM コードの実行を開始します (動作は、リセットの種類によって多少異なる)。PS 専用のリセットについては、いくつかの注意点があるため、後で詳しく説明します。

サブシステムのリスタートは、やや複雑です。Zynq UltraScale+ MPSoC のサブシステムには、特定のオペレーティング システムのコンポーネントがすべて含まれます。図 12-1 は Vivado で PS を表示したもので、OS によって定義されたサブシステムの例を示しています。Vivado のデフォルトの IP コンフィギュレーション メニューでは、利用可能な PS コンポーネントがすべてフラットに表示されます。この例では、これらのコンポーネントは 3 つのサブシステムに分離され、それぞれが独立したオペレーティング システムを実行しています。各サブシステムは 1 個のプロセッサと複数のペリフェラルおよびメモリで構成されます。この例では、次のサブシステムが表示されています。

- uC/OS-II を実行している RPU ベースのサブシステム
- FreeRTOS を実行している RPU ベースのサブシステム
- Linux を実行している APU ベースのサブシステム

Vivado PCW (PS Configuration Wizard) で [Switch to Advanced Mode] をオンにすると、[Isolation Configuration] ページでサブシステムを設定できます。

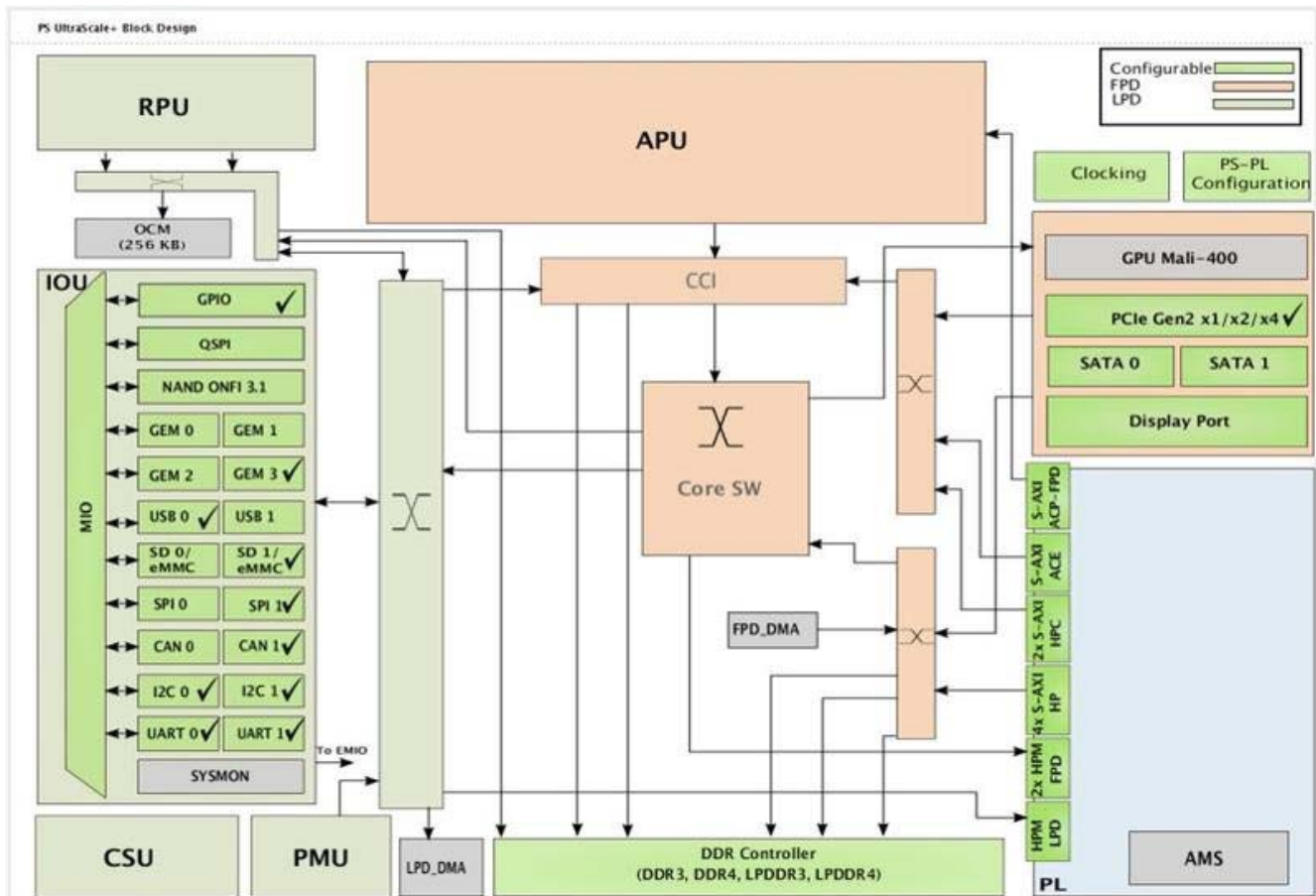


図 12-1: Vivado IP コンフィギュレーション メニュー

サブシステムをリスタートすると、Zynq UltraScale+ MPSoC で定義されたほかのアクティブなサブシステムの動作には影響を与えずに、サブシステム全体がクリーンなステートからリスタートします。たとえば、APU サブシステムのリスタート時に、Linux を実行している APU サブシステムは FSBL に戻ってリスタートしますが、FreeRTOS および uC/OS-II を実行している RPU サブシステムは中断なしに動作を継続します。同様に、RPU サブシステムのリスタートを実行する場合、APU サブシステムは中断なしに動作を継続します。

サブシステムのリスタートは、プラットフォーム管理ユニット (PMU) によって管理されます。各サブシステムをリスタートする際、PMU は実行中の AXI トランザクションがすべて終了し、新しいトランザクションが発行されていないことを確認します。図 12-2 にはサブシステムを示していますが、サブシステムのコンポーネントを接続するインターコネクトは明示していません。各サブシステムには複数のインターコネクトがあり、3 つのサブシステムはすべて同じインターコネクトを使用します。インターコネクトに未完了のトランザクションが残った状態で、PMU ファームウェアがサブシステム内のすべてのコンポーネントをリセットした場合、AXI マスターとスレーブの両方がリセット ステートになることがあります。ただしその場合、未完了の AXI トランザクションはインターコネクト内に残ったままになるため、その後のトラフィックはすべてブロックされます。インターコネクト内でトラフィックが停止すると、これらの接続は共有されているため、システムがフリーズします。このため、プロセッサを含めサブシステムのコンポーネントをリセットする前には、すべてのトランザクションが完全に終了していることを PMU が確認する必要があります。

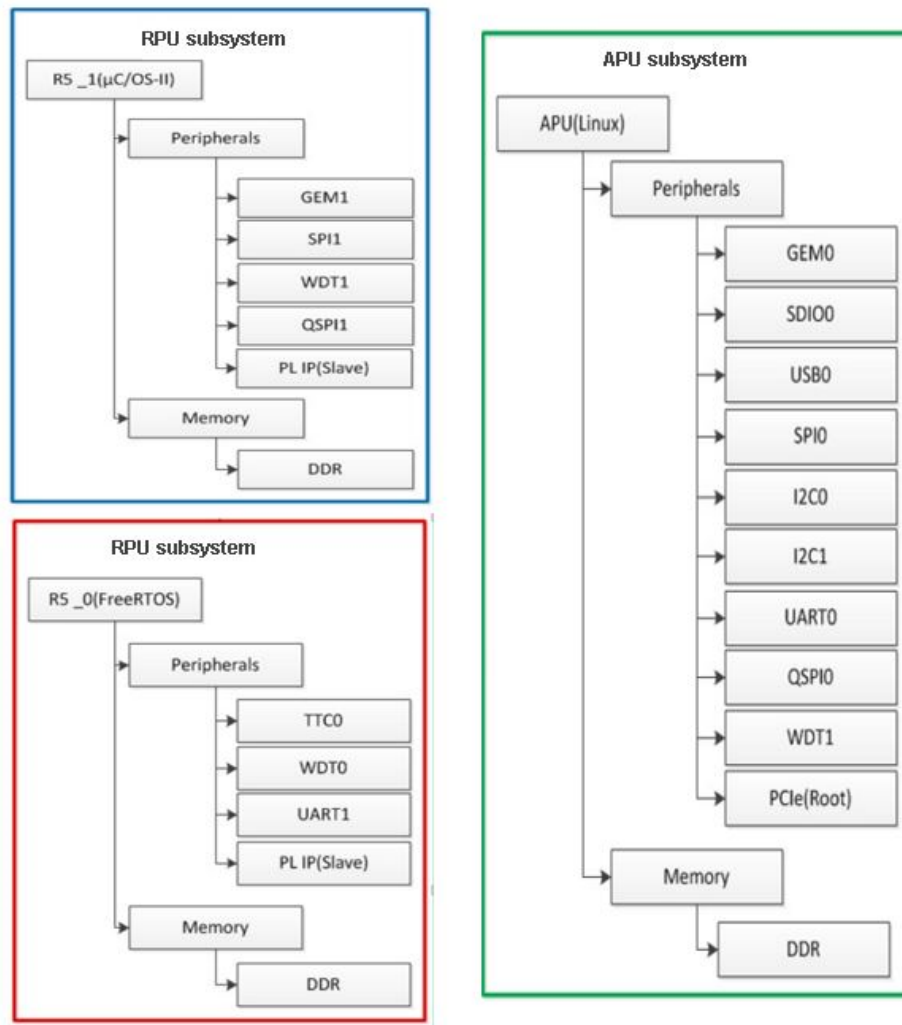


図 12-2: 各種オペレーティング システムを実行するサブシステムのコンポーネント

プロセッサをリセットから解放する前に、PMU はリセット ベクターのコードによってシステムが正常にリスタートすることを確認しておく必要があります。たとえばスタンドアロン アプリケーションを実行している RPU サブシステムの場合、アプリケーションの ELF ファイルのクリーンなコピーをロードするか、またはアプリケーション コードがリエントラントであることを確認する必要があります。Linux を実行している APU サブシステムの場合は、FSBL のリエントラントなコピーから開始する必要があります。

注記: オンチップ メモリ (OCM) モジュールには、0xFFFC0000 から始まる 256KB の RAM があります。OCM は主に FSBL および ATF コンポーネントによって使用されます。FSBL は OCM の 0xFFFC0000 ~ 0xFFFE9FFF の領域を使用します。この領域の最後の 512B は、ATF がハンドオフするアプリケーションに対応したハンドオフ パラメータを共有するために FSBL が使用します。ATF は OCM の残りの領域、すなわち 0xFFFEA000 ~ 0xFFFFFFFF を使用します。

ウォーム リセットの現在の実装では、リスタートなしに PMU ファームウェアから既存の FSBL へハンドオフできるようにするには FSBL を OCM に格納する必要があります。このため、ウォーム リスタートを有効にした場合は OCM が完全に使用され、ほかのアプリケーションは OCM を使用できません。

サポートされるユース ケース

APU サブシステムのリスタート

APU サブシステムのみをリスタートする場合、Vivado デザイン ツールの PCW を使用して APU サブシステムを定義する必要があります。PMU は APU サブシステムをリスタートするための関数を実行します。まず、PMU は APU サブシステムのすべてのコンポーネントをアイドル ステートにします。すべてのコンポーネントがアイドル ステートになったら、PMU は APU プロセッサを含む各コンポーネントをリセットします。リセットが解放されたら、OCM 内の FSBL コードを再実行します。FSBL がリスタート時に実行するタスクは、POR 時に実行するものとはわずかに異なります。

注記: FSBL はリエントラントです。このため、APU は FSBL のクリーンなコピーを再ロードしなくても、FSBL を再実行できます。

図 12-3 に、APU サブシステムのリスタート プロセスを示します。

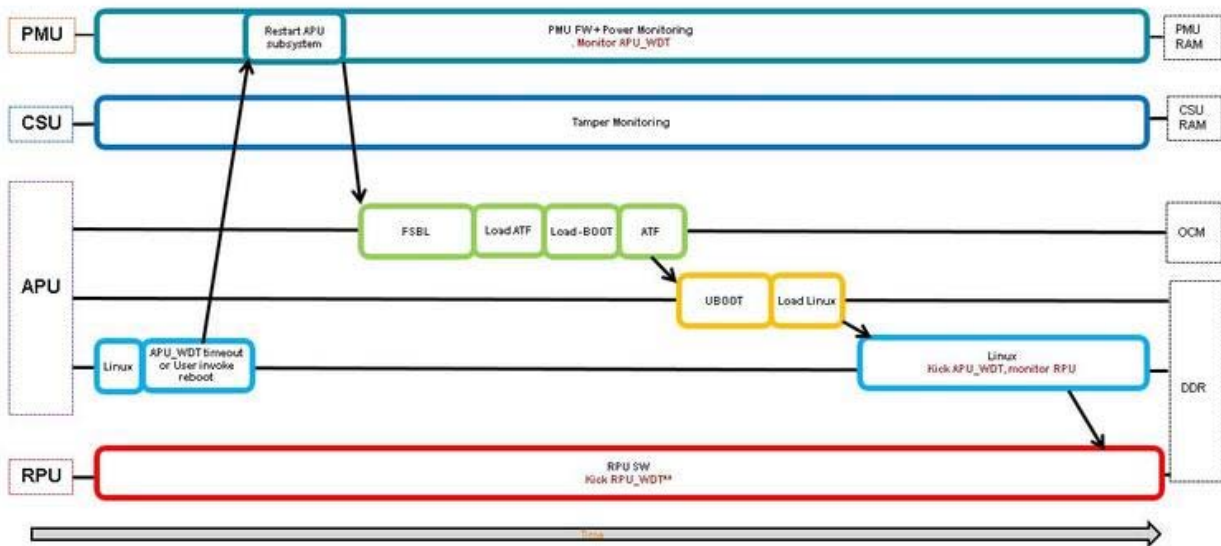


図 12-3: APU サブシステムのリスタート プロセス

このフローチャートは、クリーンな実行ステートから開始しています。Linux、RPU、PMU、および CSU サブシステムはいずれも実行中ステータスです。APU サブシステムの動作状態は、APU ウォッチドッグ タイマー (WDT) で監視します。Linux 上で動作するバックグラウンドアプリケーションが、このウォッチドッグを周期的にブーストすることにより、WDT タイムアウトの発生を防ぎます。APU サブシステムがハングすると、WDT がタイムアウトします。このタイムアウトによって PMU に割り込みが送信され、APU サブシステムがリスタートします。または、Linux で直接呼び出して APU サブシステムをリスタートすることもできます。

インプリメンテーション

サブシステムのリスタートをサポートするには、そのサブシステムを Vivado デザイン ツールの [Isolation Configuration] ページで定義しておく必要があります。

APU サブシステムで Linux を実行する場合、デフォルトの PMU サブシステム以外に次の APU サブシステムが必要です。

1. FSBL および ATF を実行するためのセキュアな APU システム
2. Linux を実行するための非セキュアな APU サブシステム。

サブシステムのコンフィギュレーションの詳細および APU のみのサブシステムの例は、「[分離の設定](#)」を参照してください。



重要: APU サブシステムは PS のコンポーネントだけで構成されますが、PL にインプリメントした IP ペリフェラルが APU サブシステムに含まれることもよくあります。ただし、[Isolation Configuration] ページのメニューには、PL IP を別のサブシステムに割り当てる機能がありません。このため、生成されるデバイス ツリー ソース (DTS) ファイルには Vivado でインスタンス化したすべての IP が追加されます。APU サブシステムを正しく定義するには、APU サブシステムに含まれない PL の IP をすべて手作業で DTS ファイルから削除する必要があります。削除していない場合、すべてのソフト IP に対するドライバーが Linux で有効になり、APU はウォーム リスタート中もすべてのソフト IP を管理しようとしてしまいます。



重要: サブシステムのリスタートを実行する際は、サブシステムに含まれるすべてのコンポーネントをアイドル ステートにしてからリセットする必要があります。この機能は、PS でサポートされるコンポーネントに対して実装されています。サブシステムの PL に含まれる IP が AXI スレーブの場合、これをアイドルにするのに特別なタスクは必要ありません。必要に応じて、これらのスレーブをリセットするためのコードを用意してください。PL の IP が AXI マスターの場合、リセット用のコードに加え、このマスターからのすべての AXI トランザクションを停止および完了するコードも用意する必要があります。アイドルへの移行およびリセットのためのコードを追加する方法の詳細は、「ペリフェラルのアイドル移行とリセット」を参照してください。

PS から PL への `resetn` 信号 (`ps_resetn`) を使用する場合は設計上の問題とガイドラインは、「PL に対する GPIO リセット」を参照してください。オプションでリカバリとエスカレーションの機能を有効にすることもできます。ソフトウェアのビルドの詳細は、「ソフトウェアのビルド」を参照してください。

RPU サブシステムのリスタート

RPU サブシステムのリスタートには、APU サブシステムとロックステップ モードまたはスプリット モードで動作する 1 つ以上の RPU サブシステムが必要です。Linux を実行する APU サブシステムが RPU サブシステムのマスターとなり、OpenAMP の `remoteproc` 機能を使用してサブシステムのライフ サイクルを管理します。APU は `remoteproc` を使用して RPU アプリケーションをロード、開始、および停止します。また、リスタート後の APU サブシステムと RPU サブシステムの再同期も実行します。APU サブシステムは、次のシーケンスによって RPU リスタートをトリガーできます。

1. RPU を停止する。
2. 新しいファームウェアをロードする。
3. RPU をリスタートする。

ユーザー コマンド、RPU ウォッチドッグ タイムアウト、メッセージパイプを使用した RPU から APU へのメッセージなど、多くのイベントによって RPU サブシステムのリスタートがトリガーされます。次に、APU が PMU に対して RPU を開始または停止するための `remoteproc` コマンドを発行すると、PMU が RPU サブシステムのステートを変更します。

図 12-4 に、RPU サブシステムのリスタート プロセスを示します。

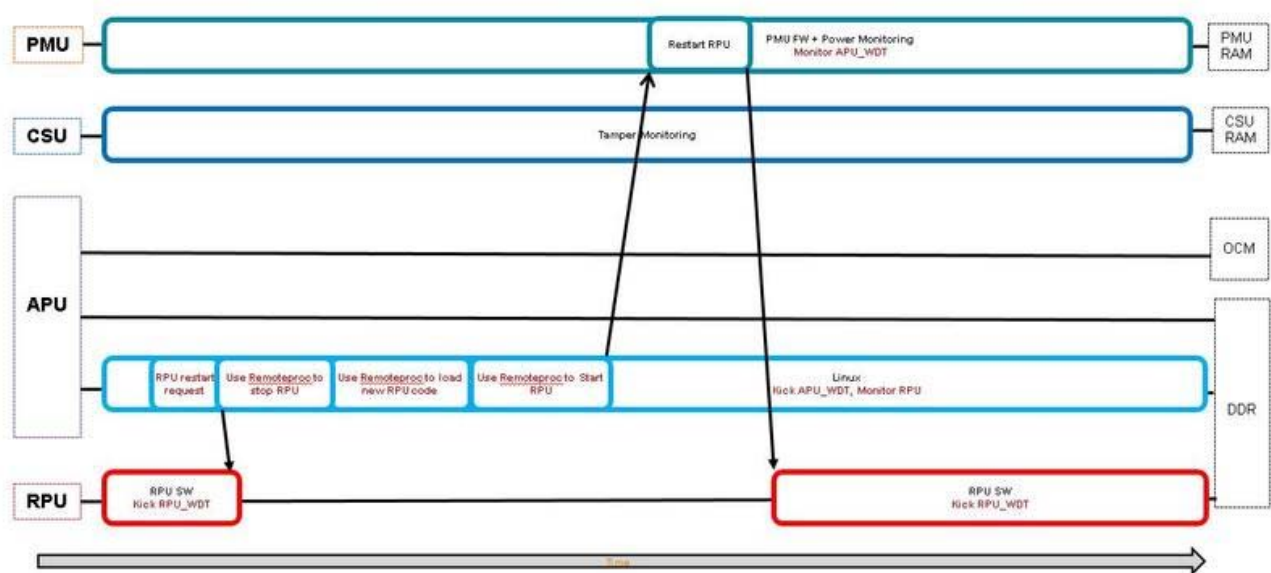


図 12-4: RPU サブシステムのリスタート

上図は、すべてのサブシステム (Linux、RPU、PMU および CSU) がクリーンな実行ステートから開始しています。このフローチャートで、APU は RPU サブシステムのリスタート要求を受信します。リスタート要求を受信すると、APU は `remoteproc` 機能を使用して RPU サブシステムを停止し、新しいファームウェア コードをロードした後、RPU サブシステムをリスタートします。このフローチャートは、RPU WDT を使用した場合を示しています。RPU は周期的にウォッチドッグをクリアします。RPU がハングすると、WDT がタイムアウトします。このタイムアウトを Linux が受信し、RPU サブシステムをリスタートします。

インプリメンテーション

Vivado PCW の [Isolation Configuration] ページで RPU サブシステムを定義しておく必要があります。また、PMU サブシステムと APU サブシステムも必要です。RPU サブシステムのコンフィギュレーションは、ロックステップ モードとスプリット モードの 2 つが可能です。サブシステムのコンフィギュレーションの詳細は、[Isolation Configuration Consideration の Wiki ページ \(英語\)](#) を参照してください。複数のサブシステム間でペリフェラルは共有できません。すべてのサブシステムでペリフェラルが相互排他であることを確認してください。



重要: サブシステムのリスタートを実行する際は、サブシステムに含まれるすべてのコンポーネントをアイドルステートにしてからリセットする必要があります。この機能は、PS でサポートされるコンポーネントに対して実装されています。サブシステムの PL に含まれる IP が AXI スレーブの場合、これをアイドルにするのに特別なタスクは必要ありません。必要に応じて、これらのスレーブをリセットするためのコードを用意してください。PL の IP が AXI マスターの場合、リセット用のコードに加え、このマスターからのすべての AXI トランザクションを停止および完了するコードも用意する必要があります。アイドルへの移行およびリセットのためのコードを追加する方法の詳細は、「ペリフェラルのアイドル移行とリセット」を参照してください。

RPU サブシステムのリスタートは、APU 上の remoteproc の Linux カーネル実装と、RPU 上の OpenAMP ライブラリを組み合わせることでサポートしています。現在、APU 上の Linux ユーザー空間の OpenAMP ライブラリではサポートされません。RPU アプリケーションは、OpenAMP アプリケーションの要件に従って作成する必要があります。詳細は、『Zynq デバイス用 Libmetal および OpenAMP ユーザー ガイド』(UG1186) [参照 16] を参照してください。remoteproc に rpmsg は必要ありません。rpmsg 機能は、2 つのプロセッサ間の通信パイプとして使用できます。この際、remoteproc と rpmsg は独立しています。サブシステムのリスタート時に remoteproc を正しく動作させるには、RPU アプリケーションに静的な共有メモリ割り当てを使用したリソース テーブルを含める必要があります。サブシステムのリスタートでは、動的な共有メモリ割り当てはサポートされません。『Zynq デバイス用 Libmetal および OpenAMP ユーザー ガイド』(UG1186) [参照 16] の「簡単な OpenAMP アプリケーションの作成方法」に記載された手順を実装すると、remoteproc の要件をすべて満たすことができます。

初期化後、RPU アプリケーションは XPm_InitFinalize() を呼び出して、電力管理初期化が完了したことを PMU に通知する必要があります。

注記: XPm_InitFinalize() の呼び出しが早すぎると、初期化が完了していないスレーブの電源がオフになります。これらのスレーブは、次に RPU アプリケーションによって初期化される際に再び電源がオンになるため、パワーアップ時のレイテンシが長くなります。電力管理に対応した RPU アプリケーションの作成方法の詳細は、「ZU+ Example - PM Hello World」の Wiki ページ (英語) を参照してください。

最後に、RPU コード用の予約メモリ アドレスがすべてのレイヤーで同期していることを確認する必要があります。これは、Vivado の [Isolation Configuration] ページで APU サブシステムと RPU サブシステム両方の [Memory] で定義します。Linux の OpenAMP オーバーレイの DTS ファイル、および RPU アプリケーションのリソース テーブルとリンク スクリプトで同じアドレス領域を使用する必要があります。

PS から PL への resetn 信号 (ps_resetn) を使用する場合は設計上の問題とガイドラインは、「PL に対する GPIO リセット」を参照してください。オプションでリカバリとエスカレーションの機能を有効にすることもできます。ソフトウェアのビルドの詳細は、「ソフトウェアのビルド」を参照してください。

PS 専用のリセット

PS 専用のリセットでは、PL が動作を継続したままプロセッシング システム (PS) 全体がリセットされます。PS 専用のリセットを開始する前に、PMU は PS と PL の分離を有効にし、PS と PL の間の信号を十分に定義されたステートにクランプします。PS 専用のリセットが解放されると、PS は PMU ROM、CSU ROM、FSBL から始まる標準のブート プロセスを実行します。FSBL の実行中は PS と PL の分離が解除されます。



重要: PS 専用のリセットによってソフトウェアはリセットされるため、動作を継続している PL のハードウェア IP のステートと、この IP と通信 (または IP を制御) するソフトウェアのステートの同期が失われることがあります。ソフトウェア ステートとハードウェア ステートの同期は、ユーザーが適切に維持する必要があります。PS 専用のリセットでは、ビットストリームを再ダウンロードできません。

PS 専用のリセットは、Linux コマンド、ウォッチドッグ タイムアウト、または PMU エラー管理ブロックのいずれかによって開始できます。

APU/RPU サブシステムをリスタートせずに PS 専用のリセットを実行する場合、サブシステムの分離設定は必要ありません。リブート タイプとリブートを設定する Linux コマンドを、変更なしにそのまま実行できます。

システム リセット

システム リセットでは、PS と PL を含むハードウェア全体がリセットされます。システム リセットが解放されると、PS は PMU ROM、CSU ROM、FSBL から始まる標準のブート プロセスを実行します。次の表に、システム リセットと POR の違いを示します。

表 12-1: POR とシステム リセットの違い

POR	システム リセット
永続レジスタをリセット	永続レジスタはリセットされない
ブート モード ピンを再サンプル	ブート モード ピンは再サンプルされない
デバッグ ステートをリセット	デバッグ ステートはリセットされない
eFuse の値を再サンプル	リフレッシュにはソフトウェアによる明示的な動作が必要
セキュリティ ステートが決定される	セキュリティ ステートはロックされる
タンパー応答がクリアされる	タンパー応答はクリアされない
セキュリティ キー ソースを選択する	セキュリティ キー ソースはロックされる
オプションで LBIST、スキャン/クリアを実行	LBIST、スキャン/クリアは実行されない
MBIST を実行	MBIST の実行にはソフトウェアによる明示的な動作が必要

システム リセットは、Linux コマンド、ウォッチドッグ タイムアウト、または PMU エラー管理ブロックのいずれかによって開始できます。APU/RPU サブシステムをリスタートせずにシステム リセットのみを実行する場合、サブシステムの分離設定は必要ありません。

ペリフェラルのアイドル移行とリセット

サブシステムをリセットする前に、サブシステムの IP またはプロセッサが実行中のトランザクションを停止/完了する必要があります。そうしないとインターコネクトがハングし、その結果システム全体がハングすることがあります。また、リブート後に IP が正しく動作するように、IP をリセットしたら bootROM 後のステートまで移行させることが推奨されます。

サブシステムのリセット中にアイドルへの移行とリセットが可能な PS の IP については、ペリフェラルをアイドルに移行してリセットする機能が PMU ファームウェアに実装されています。どの IP に対してアイドルへの移行とリセットを試みるかは、Vivado の分離設定に基づいて決定します。

サブシステム ノードのアイドルへの移行とリセットを有効にするには、PMU ファームウェアをビルドする際に次のフラグを使用します。

```
ENABLE_NODE_IDLING
IDLE_PERIPHERALS
```

ノードのリセットとアイドル

サブシステムのリスタート時、PMU ファームウェアは関連する PS ペリフェラル ノードがアイドルになり、リセット ステートになっていることを確認します。

リセットされるサブシステムに含まれる PS ペリフェラルのうち、アイドルへの移行/リセットが現在サポートされているのは、次のとおりです。

- TTC
- イーサネット/EMAC
- I2C
- SD
- eMMC
- QSPI
- USB
- DP
- SATA

GPIO リセットの影響の詳細は、「PL に対する GPIO リセット」を参照してください。

注記: PS 専用のリセットおよびシステム リセットのコマンドをユーザーが実行した場合、PS ペリフェラルがアイドルに移行してからリセットが開始します。

カスタムフック

PMU ファームウェアは PL ペリフェラルを管理しません。このため、PMU ファームウェアにはアイドル/リセット機能は実装されていません。ただし、PL ペリフェラルも PS ペリフェラルと同じように扱う必要があります。

idle_hooks.c ファイルに、PL ペリフェラルをアイドルにしてリセットするカスタムフックを追加できます。

これらのフックは、PMU ファームウェアの pm_master.c ファイルにある PmMasterIdleSlaves 関数から呼び出すことができます。

```
lib/sw_apps/zynqmp_pmufw/src/pm_master.c
:dir:dir -769,6 +769,12 :dir:dir static void PmMasterIdleSlaves(PmMaster* const
master)

    PmDbg(DEBUG_DETAILED, "%s\r\n", PmStrNode(master->nid));

+    /*
+     * Custom hook to idle PL peripheral before PS peripheral idle
+     */
+
+    Xpfw_PL_Idle_HookBeforeSlaveIdle(master);
+
    while (NULL != req) {
        u32 usage = PmSlaveGetUsageStatus(req->slave, master);
        Node = &req->slave->node;
```

```

:dir:dir -783,6 +789,11 :dir:dir static void PmMasterIdleSlaves(PmMaster* const
master)
    }
    req = req->nextSlave;
}
+
+ /*
+  * Custom hook to idle PL peripheral after PS peripheral idle
+  */
+ Xpfw_PL_Idle_HookAfterSlaveIdle(master);
#endif
}

```

必要に応じて、Xpfw_PL_Idle_HookBeforeSlaveIdle と Xpfw_PL_Idle_HookAfterSlaveIdle に PL ペリフェラルをアイドルにしてリセットするコードを含めることができます。実装方法は、次の 2 つがあります。

- PL IP の AXI レジスタに書き込んでアイドル ステートにしてリセットする。PL ペリフェラルを適切にアイドルにできるため、この方法を推奨します。
- 信号ベースのハンドシェイクを実装し、PMU ファームウェアが PL に対して PL にあるすべての IP をアイドルにするように通知する。直接制御してトラフィックを適切に停止することができない場合は、この実装を使用します。たとえば、リセット制御がなくファイアウォール IP を経由して接続されている DMA 以外の IP が PL に存在するような場合、この実装を使用できます。また、1 番目の実装では各 IP を個別にアイドルにする必要がありますが、この実装では通過するすべてのトラフィックを停止できます。

注記: これらカスタム フックのインプリメンテーションは、ザイリンクスからは提供されません。

PL に対する GPIO リセット

Vivado での設定により、PS から PL へのファブリック リセットが可能です。図 12-5 に示すように、[Fabric Reset Enable] をオンにして、[Number of Fabric Resets] を「2」に設定すると、Zynq UltraScale+ ブロックの出力信号 pl_resetn0 と pl_resetn1 を使用して PL コンポーネントのリセット ピンを駆動できます。

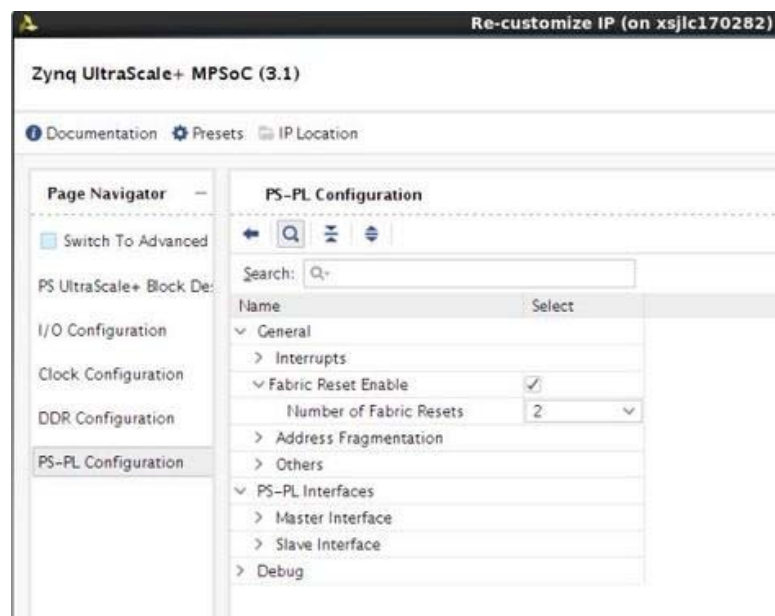


図 12-5: PS から PL へのリセット

p1_resetn 信号は PS GPIO を使用して実装されます。p1_resetn ピンは、psu_ps_pl_reset_config_data 関数を使用してソフトウェアでビットストリーム コンフィギュレーションが完了した後に解放されます。サブシステムがリセット以外の目的でも GPIO を使用している場合、サブシステムの定義に GPIO ブロックが含まれます。次の図に、GPIO をスレーブ ペリフェラルとして含めた APU サブシステムの例を示します。

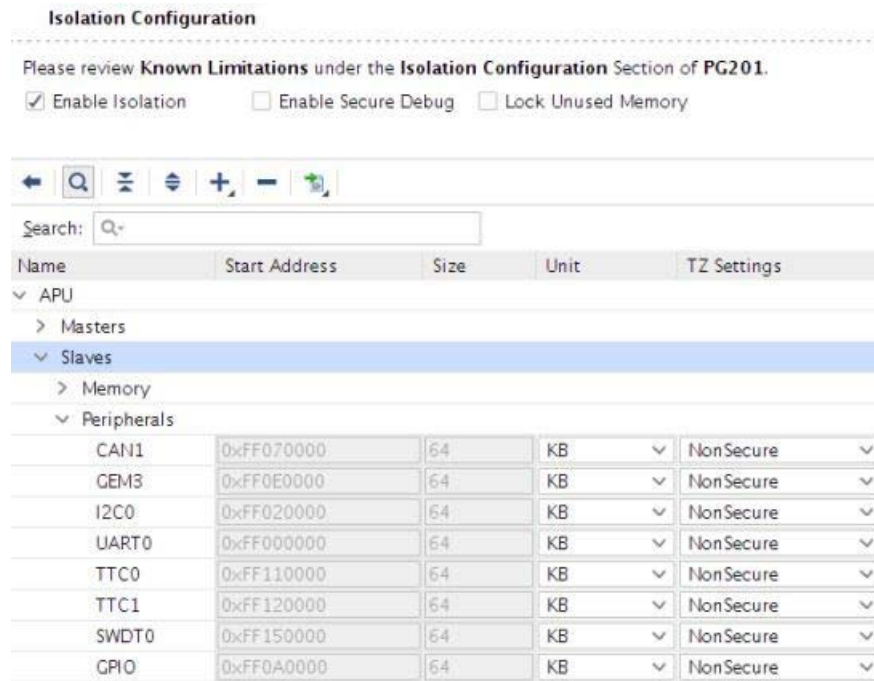


図 12-6: GPIO を含めた APU サブシステム

GPIO がサブシステムのスレーブ ペリフェラルの場合、サブシステムをリスタートすると、リスタートプロセスの一部として GPIO コンポーネント全体がリセットされます。p1_resetn は GPIO で実装されているため、サブシステムのリスタート中、p1_resetn は Low に駆動されます。この動作は、PL にリセット対象の IP 以外にサブシステム内に p1_resent 信号で駆動されている IP が存在する場合、問題になることがあります。たとえば、p1_resetn0 が APU サブシステムの PL IP に対するリセットを駆動し、p1_resetn1 が RPU サブシステムの PL IP を駆動しているとします。APU サブシステムのリスタート中、p1_resetn0 と p1_resent1 はいずれも強制的にリセットステートとなります。したがって、RPU サブシステムの PL IP がリセットされてしまいます。GPIO は p1_resetn 信号を使用して APU サブシステムと RPU サブシステムの間で暗黙的に共有されているため、APU のリスタートは RPU サブシステムに影響しないはずですが、実際の動作はそうになりません。サブシステムのリスタートに関してはペリフェラルの共有がサポートされていないため、p1_resetn によってサブシステムのリセット時に問題が発生します。回避策としては、サブシステムの分離設定で GPIO ペリフェラルが割り当てられていても、サブシステムのリスタート時に GPIO ペリフェラルのアイドルへの移行とリセットをスキップするようにします。

ノードのアイドルへの移行/リセット時に GPIO リセットをスキップするには、PMU ファームウェアをビルドする際に次のフラグを使用します。

REMOVE_GPIO_FROM_NODE_RESET_INFO

注記: GPIO コンポーネントは、PS 専用のリセットでもリセットされます。PMU ファームウェアが PS と PL の分離を有効にしてから PS 専用のリセットを呼び出すことで p1_resetn が High にロックされます。ただし、FSBL が PS と PL の分離を解除するとすぐにリセットが Low に遷移します。次に、FSBL は psu_ps_pl_reset_config_data を呼び出して p1_resetn を再び High にします。この動作が必要なのは、PL コンポーネントをリセットすることによって、リセット後のソフトウェアステートとハードウェアステートを正しく同期できるようにするためです。

ハングしたシステムからのリカバリ

FPD WDT のタイムアウトによってシステムのハングが確認された場合、PMU を使用してイベント シーケンスを実行し、無応答の状態からのリカバリを試みることができます。デフォルトでは、FPD WDT タイムアウトが発生しても PMU ファームウェアはリスタートを開始しません。これは、ユーザーが希望の動作を指定できるようにするためです。ただしザイリンクスは、PMU ファームウェアが FPD WDT を使用して APU サブシステムのステートを監視し、WDT のタイムアウトによって Linux の異常を検出したら APU (Linux) サブシステムをリスタートする一般的なリカバリ方式を提供しています。

RPU サブシステムは `remoteproc` を使用して Linux が管理するため、RPU サブシステムのライフサイクルは、完全に Linux に依存します。PMU は、RPU サブシステムをいつリスタートするか判断には関与しません。RPU のハングからのリカバリは、APU サブシステムと RPU サブシステム間のソフトウェアまたはハードウェア ウォッチドッグを使用して実装することもできます。この場合、ウォッチドッグは Linux によって設定および処理されますが、ハートビートは RPU アプリケーションによって提供されます。RPU をいつリスタートするかを判断する方法は数多く存在し、ウォッチドッグはその 1 つにすぎません。どの方法を使用するかは、ユーザーに委ねられます。リカバリを有効にするには、エラー管理とリカバリを有効にして PMU ファームウェアをビルドする必要があります。リカバリ機能は、次のマクロで有効にします。

```
ENABLE_EM
ENABLE_RECOVERY
```

また、ATF をビルドする際に次のフラグを使用する必要があります (詳細は「[APU のアイドルへの移行](#)」参照)。

```
ZYNQMP_WARM_RESTART=1
```



重要: これらのコンパイル フラグを有効にすると、TTC タイマーの 1 つ (タイマー 9) が PMU 専用となります。

ウォッチドッグ管理

APU のステータスは、FPD WDT を使用して監視します。APU 上で動作するソフトウェアが周期的に FPD WDT をクリアし、タイムアウトしないようにします。APU に予期しない条件が発生し、ソフトウェアが正常に動作しなくなると FPD WDT がタイムアウトし、APU のリスタートが開始されます。FPD WDT は初期化ステージで PMU ファームウェアによって設定されますが、周期的なクリア処理は APU 上で動作するソフトウェアで実行されます。

WDT のタイムアウト時間はデフォルトで 60 秒に設定されており、PMU ファームウェアの `RECOVERY_TIMEOUT` フラグで変更できます。APU サブシステムがリスタート サイクルに入っても、FPD WDT は動作を継続します。これは、リスタート後も APU 上で動作するソフトウェアが再び WDT をクリアして通常の実行ステートを維持できるようにするためです。したがって、リスタート プロセスの途中で WDT がタイムアウトするのを防ぐため、WDT のタイムアウト時間は APU サブシステムのリスタートにかかる時間よりも長く設定する必要があります。Linux からのハートビートは、なるべく早く送信を開始することを推奨します。PetaLinux の BSP には、ウォッチドッグ管理サービスを `init.d` に追加するためのレシピが含まれます。FPD WDT は PMU ファームウェアがオーナーのため、WDT にフル機能 Linux のドライバを使用するのは安全ではありません。WDT のリスタート レジスタ (WDT ベース + 0x8) にリスタート キー (0x1999) を書き込むことによって、ハートビートを実装することを推奨します。これは、C プログラムのデーモンで実行する方法と、`bash` スクリプトのデーモンで実行する方法があります。

これをアイドル スレッドなどの低優先度スレッドに含めておき、これがハングしたらサブシステムのハングと見なすことを推奨します。

次に、コマンド プロンプトから実行する FPD WDT への 1 回分のハートビートのスニペットを示します。これを、周期的に実行する `bash` スクリプトに含めることができます。

```
# devmem 0xFD4D0008 32 0x1999
```

次に示す `wdt-heartbeat` アプリケーションは、FPD WDT に対して周期的にハートビートを送信します。デモのため、このアプリケーションはデーモンとして起動しています。このアプリケーションのコードは、Linux の `Idle` スレッドなど適切な場所に実装できます。

```
#include <stdio.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>

#define WDT_BASE                0xFD4D0000
#define WDT_RESET_OFFSET       0x8
#define WDT_RESET_KEY          0x1999

#define REG_WRITE(addr, off, val) (*(volatile unsigned int*)(addr+off)=(val))
#define REG_READ(addr, off) (*(volatile unsigned int*)(addr+off))

void wdt_heartbeat(void)
{
    char *virt_addr;
    int fd;
    int map_len = getpagesize();

    fd = open("/dev/mem", (O_RDWR | O_SYNC));

    virt_addr = mmap(NULL,
                     map_len, PROT_READ|PROT_WRITE,
                     MAP_SHARED,
                     fd,
                     WDT_BASE);

    if (virt_addr == MAP_FAILED)
        perror("mmap failed");

    close(fd);

    REG_WRITE(virt_addr, WDT_RESET_OFFSET, WDT_RESET_KEY);

    munmap((void *)virt_addr, map_len);
}

int main()
{
    while(1)
    {
        wdt_heartbeat();
        sleep(2);
    }
    return 0;
}
```


ウォッチドッグがタイムアウトすると、PMU ファームウェアは WDT 割り込みを受信し、処理します。PMU ファームウェアはサブシステムのマスター CPU (すべての A53 コア) をアイドルにした後 (「[APU のアイドルへの移行](#)」参照)、APU 専用のリスタート フロー (CPU のリセット、およびサブシステム リセットの影響を受けるペリフェラルのアイドル移行とリセット (「[ペリフェラルのアイドルへの移行](#)」参照)) を実行します。

注記: PMU ファームウェアでエスカレーションを有効にした場合、APU のみのリスタートとは別のリスタート フローが適宜トリガーされます (「[エスカレーション](#)」参照)。

APU のアイドルへの移行

各 A53 は、WFI ステートに遷移することによってアイドルへ移行します。これは、Arm トラステッド ファームウェア (ATF) によって実行されます。CPU をアイドルに移行する場合、PMU ファームウェアが TTC 割り込み (タイマー 9) を ATF に送信することで、すべてのアクティブな A53 コアにソフトウェア割り込みが送信されます。各コアは保留中の SGI をクリアして、WFI ステートに遷移します。

最後のコアは、WFI に移行する直前に PMU ファームウェアに対して `pm_system_shutdown` (PMU ファームウェア API) を発行し、PMU ファームウェアはこれによって APU 専用のリスタート フローを実行します。

リカバリを正しく動作させるには、ATF でこの機能を有効にしておく必要があります。この機能を有効にするには、ATF をビルドする際に `ZYNQMP_WARM_RESTART=1` フラグを使用します。

リカバリ方式の変更

ザイリンクスは、`ENABLE_RECOVERY` がオンの場合に FPD WDT タイムアウトによって APU サブシステムのリスタートが開始するリカバリ方式を実装しています。このコードを変更すると、リカバリの動作を簡単に変更できます。FPD WDT タイムアウトによって PMU ファームウェアがシステム リセットを開始する実装の例は、ザイリンクス アンサー : [69423](#) を参照してください。

エスカレーション

現在実行中のリカバリではシステムを正常な動作ステートに戻すことができない場合、システムは次の WDT タイムアウトでより大規模なリセットを実行し (エスカレーション)、完全なリカバリを試みる必要があります。エスカレーションの方式は、ユーザーが選択できます。一般的には、1 回目のウォッチドッグ タイムアウトで APU リスタートを試み、2 回目のウォッチドッグ タイムアウトで PS 専用のリセットを試み、最後にシステム リセットを実行するという方式がよく使用されます。

エスカレーションを有効にするには、PMU ファームウェアのビルド時に次のフラグを使用します。

```
ENABLE_ESCALATION
Escalation Scheme
```

デフォルトの方式

デフォルトの方式では、APU のみのリスタートに対して ATF から `pm_system_shutdown` が正しく呼び出されたかどうかをチェックします。この API は、ATF がすべてのアクティブな CPU をアイドルに移行できたときに呼び出されます。ATF がアクティブなコアをアイドルにできなかった場合 (図 12-7 の青で示したボックス)、再び WDT タイムアウトが発生し、`WDT_in_Progress` フラグがセットされ、エスカレーションが実行されます。

エスカレーションによってシステム レベル リセットがトリガーされます。システム レベル リセットの定義は、PLL が存在する場合は PS 専用のリセット、PLL が存在しない場合はシステム リセットとなります。

図 12-7 に、デフォルトの方式の制御フローを示します。

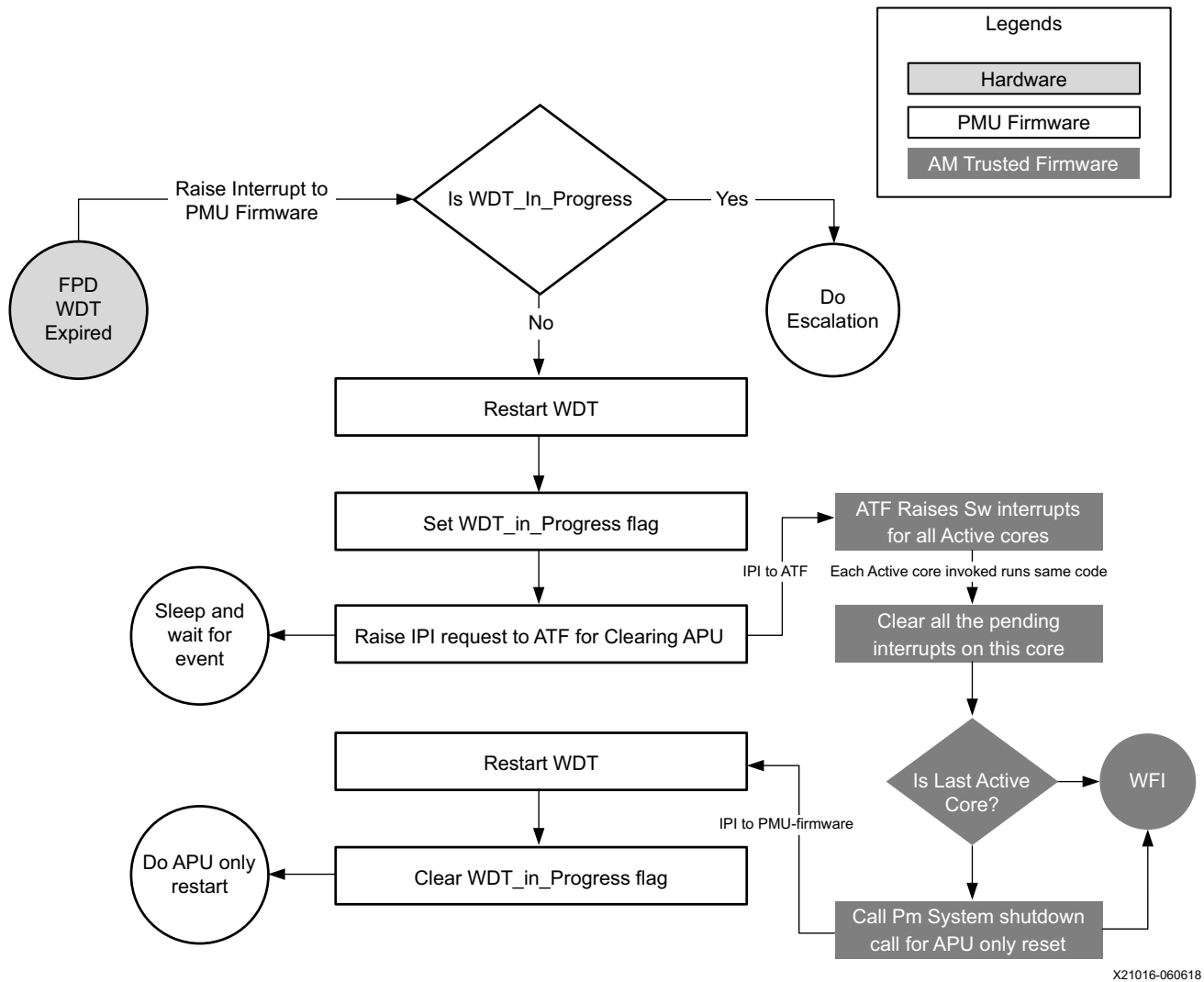


図 12-7:

ヘルシービット方式

デフォルトの方式では、必ずしもシステムのリブートが正しく実行されるとは限りません。リカバリ中に CPU をアイドルにするという ATF の役割が確実に実行されるというだけにすぎません。たとえば、FPD_WDT がタイムアウトして APU サブシステムのリスタートが呼び出される場合を考えてみます。ここで、ATF が pm_system_shutdown を正しく呼び出したとしても、pm_system_shutdown が呼び出された直後の段階では、APU サブシステムのリスタートが完了したとはいえません。FSBL、U-Boot、または Linux 初期化ステートなど別の場所でリスタートプロセスが停止してしまう可能性があります。これらいずれかのタスクでリスタートプロセスが停止すると、再び FPD_WDT タイムアウトが発生し、ATF がロードされて正しく機能していれば、同じサイクルが繰り返されるだけです。このサイクルは無限に続き、システムは何度ブートしても決してクリーンな実行ステートには戻りません。

この問題を解決するのがヘルシービット方式です。ヘルシービットは、正常にブートすると Linux によってセットされるビットで、PMU ファームウェアはデフォルトの方式に加え、ヘルシービットのチェックも実行します。WDT タイムアウトの発生時にヘルシービットがセットされている場合、Linux がクリーンな実行ステートにブート可能なことを示しているため、エスカレーションは必要ありません。ヘルシービットがセットされていない場合は、最後のリスタートで Linux が正しくブートしていないことを示しており、エスカレーションが必要となります。こうすると、同じ種類のリスタートを繰り返す必要がなくなります。PMU ファームウェアはリセットをエスカレーションし、システムレベルリセットを呼び出します。

ヘルシービット方式は、PMU グローバル汎用ストレージレジスタのビット 29

(PMU_GLOBAL_GLOBAL_GEN_STORAGE0[29]) を使用して実装されています。PMU ファームウェアは、リカバリまたは通常のリブートを開始する前にこのビットをクリアします。Linux は、正常にブートしたらこのビットをセットして知らせる必要があります。

Linux から PMU グローバルレジスタへのアクセスには、sysfs インターフェイスを使用します。このため、Linux からヘルシービットをセットするには、次のコマンドを実行するか、コードに記述します。

```
# echo "0x20000000 0x20000000" > "/sys/devices/platform/firmware/ggs0"
```

ヘルシービット方式のエスカレーションを有効にするには、PMU ファームウェアのビルド時に次のフラグを使用します。

```
CHECK_HEALTHY_BOOT
```

図 12-8 に、ヘルシー ビット方式によるエスカレーションの制御フローを示します。

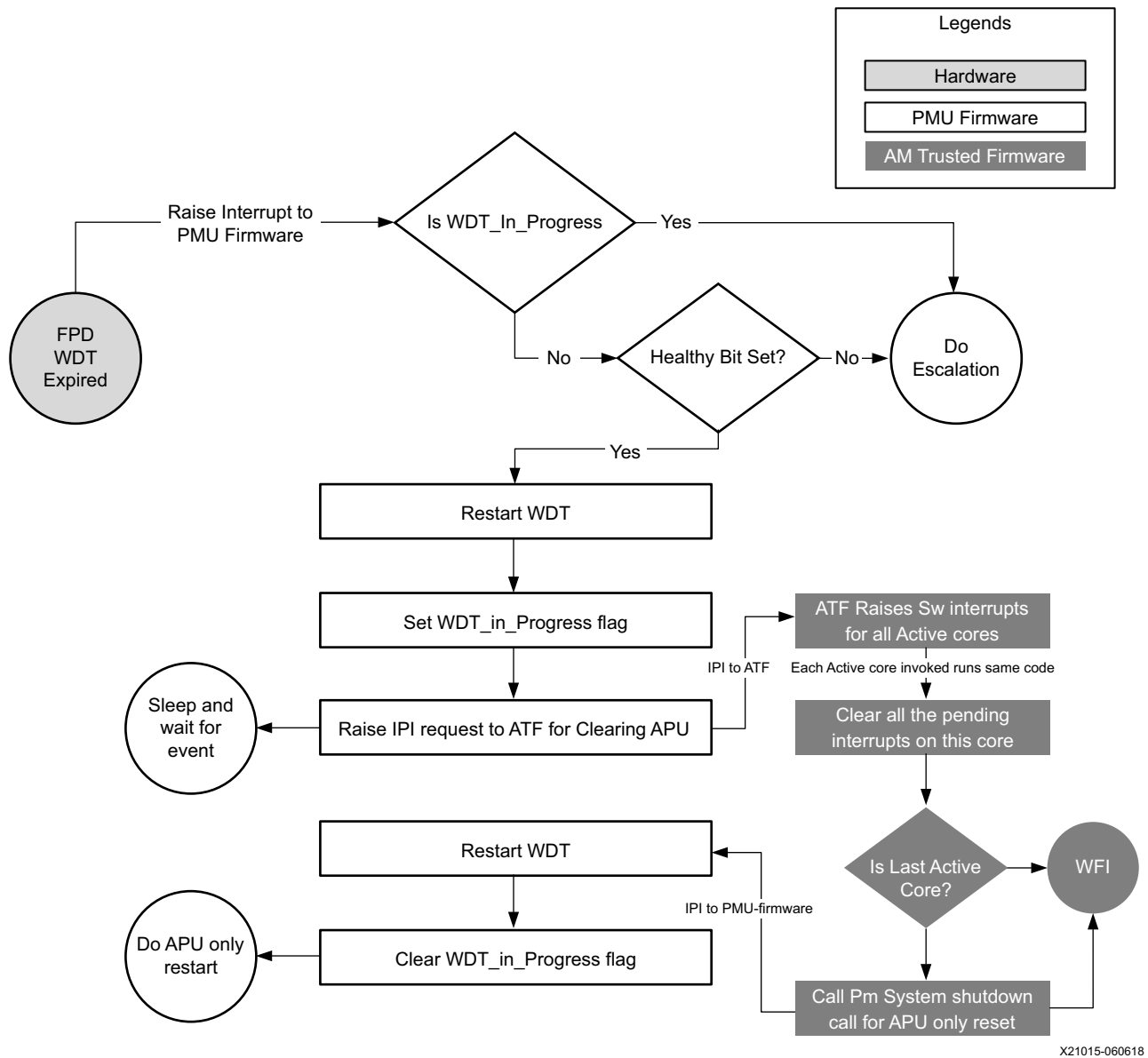


図 12-8: ヘルシー ビット方式によるエスカレーション

リカバリおよびエスカレーション方式のカスタマイズ

デフォルトでは、FPD WDT タイムアウトが発生しても PMU ファームウェアはリスタートを開始しません。ザイリックスはリカバリとエスカレーションに関する動作を定義した実装を提供していますが、これらはユーザーが希望する方式に簡単にカスタマイズできます。

FPD WDT タイムアウトが発生すると、FpdSwdtHandler が呼び出されます。ENABLE_EM が定義されている場合、FpdSwdtHandler は XPfw_recoveryHandler を呼び出します。それ以外の場合は空の関数です。

```
xpfw_mod_em.c:

#ifdef ENABLE_EM
oid FpdSwdtHandler(u8 ErrorId)
{
XPfw_Printf(DEBUG_ERROR, "EM: FPD Watchdog Timer Error (Error ID: %d)\r\n",
ErrorId);
XPfw_RecoveryHandler(ErrorId);
}

#else
void FpdSwdtHandler(u8 ErrorId) { }
```

ENABLE_EM が定義されていない場合は、FPD タイムアウト時に呼び出される FpdSwdtHandler のみを更新します。ENABLE_EM を有効にした場合は、XPfw_recoveryHandler を更新する必要があります。

同様に、RECOVERY を有効にすると XPfw_RecoveryHandler が定義されます (xpfw_restart.c 参照)。RECOVERY を有効にしない限り、XPfw_RecoveryHandler は空の関数のため、FPD_WDT タイムアウトが発生しても何も起こりません。

RecoveryHandler は、基本的に「エスカレーション」のセクションで示したフローチャートに従って動作します。FPD_WDT タイムアウトが発生すると、コードはオレンジで示したボックスに従って進行します。WDT が実行中でなければ、WDT をリスタートして WDT_In_Progress フラグをセットし、ATF に対して TTC (タイマー 9) 割り込みを送信します。この後は、ATF が処理を引き継ぎます。ATF はすべてのアクティブなコアに対してソフトウェア割り込みを送信し、保留中の割り込みをクリアするなどの処理を実行します (青で示したボックス)。基本的に、PMU は WDT をリスタートしてブーストした後、ATF へ要求を送信します。ATF は 4 つの APU をすべて正しい手順でアイドルにし、これらがすべて WFI に遷移したら (Last Active Core = True)、APU サブシステムを引数として PMU システム シャットダウンを PMU に対して発行します。PMU はこのコマンドを受信すると、APU サブシステムのリスタートを開始します。

ENABLE_ESCALATION がセットされていないと、コードが Do Escalation のパスに進むことはありません。たとえば何らかの異常によって APU が 4 つすべての CPU コアを WFI に遷移できない場合など、何らかの理由で RecoveryHandler がハングすると、APU リスタートの再試行を無限に繰り返し、ハングしてしまいます。ENABLE_ESCLATION が有効の場合は、このフローチャートの実行中に何らかの異常が発生しても、WDT_in_progress フラグはフローチャートの最後でクリアされるため WDT は動作中と見なされます。したがって Do Escalation によって SYSTEM_RESET が呼び出されるため、APU リスタートの再試行を繰り返すことはありません。

リカバリとエスカレーションの動作をカスタマイズするには、XPfw_recoveryHandler をテンプレートとして使用し、カスタマイズした XPfw_recoveryHandler 関数を使用してください。

ソフトウェアのビルド

すべてのソフトウェア コンポーネントは、ザイリンクス PetaLinux ツールを使用してビルドおよびパッケージ化します。ソフトウェア コンポーネントのビルドおよびパッケージ化の方法の詳細は、[PetaLinux の Wiki ページ \(英語\)](#) を参照してください。

リスタート ソリューション用のビルド フラグ

Zynq UltraScale+ MPSoC のリスタートの動作は、次に示すビルド時フラグで変更できます。これらは、デフォルトでは有効になっていません。

表 12-2: ビルド時フラグ

コンポーネント	フラグ名	説明	依存関係
PMU ファームウェア	ENABLE_EM	エラー管理を有効にし、WDT 割り込みを処理できるようにします。これはリスタートには直接関係しませんが、リカバリに必要です。	
	ENABLE_RECOVERY	WDT タイムアウト発生時のリカバリを有効にします。	
	ENABLE_ESCALATION	ブートまたはリカバリがエラーになった場合のエスカレーションを有効にします。	
	CHECK_HEALTHY_BOOT	ヘルシー ビットを使用してエスカレーションを決定します。	
	IDLE_PERIPHERALS ENABLE_NODE_IDLING	PMU ファームウェアがペリフェラル ノードをアイドルにしてリスタートするには、これらのフラグを両方使用する必要があります。	
	REMOVE_GPIO_FROM_NODE_RESET_INFO	アイドルへの移行およびリセットの対象となるノードのリストから GPIO を除外します。 これは、現在のサブシステム以外の PL またはその他のペリフェラルに対して GPIO を使用してリセット (または同等の) 信号を供給している場合に必要となります。 このフラグがセットされていると、GPIO はリセットされません。	
ATF	ZYNQMP_WARM_RESTART=1	ATF のウォーム リスタート リカバリ機能を有効にし、PMU ファームウェアからのトリガーによって CPU をアイドルにできるようにします。	
FSBL	FSBL_PROT_BYPASS	DDR および OCM を除き、XMPU/XPPU ベースのシステム コンフィギュレーションをスキップします。	
Linux	CONFIG_SRAM	Remoteproc を使用して RPU イメージを TCM に読み込むために必要です。	

コンポーネント レシピの変更

各コンポーネントのレシピを変更して、ビルド時のコンパイルフラグを含めたり、カスタムコードの変更/追加のためのパッチを含めたりできます。PetaLinux には、ユーザーによる変更をサポートする `meta-user yocto` ベースのレイヤーがあります。このレイヤーは、プロジェクトディレクトリの `project-spec/meta-user/` ディレクトリにあります。

PMU ファームウェア

PMU ファームウェアのユーザー固有レシピは、`dir:project-spec/meta-user/recipes-bsp/pmu/pmu-firmware_%.bbappend` にあります。存在しない場合、このパスに作成してください。

PMU ファームウェアコードは、[embeddedsw](#) GitHub リポジトリへのパッチによって変更できます。ソースコードは、`embeddedsw/tree/master/lib/sw_apps/zynqmp_pmufw` にあります。

これらのパッチを `project-spec/meta-user/recipes-bsp/pmu/files` ディレクトリにコピーし、コピーしたパッチの名前を `pmu-firmware_%.bbappend` ファイルに追加します。

例:

PMU ファームウェアソースに対する `my_changes.patch` を追加し、[表 12-2](#) に示したすべてのフラグをセットして有効にする場合、

`project-spec/meta-user/recipes-bsp/pmu/pmu-firmware_%.bbappend` を次のように変更します。

```
YAML_COMPILER_FLAGS_append = " -O2 -DENABLE_EM -DENABLE_RECOVERY
-DENABLE_ESCALATION -DENABLE_NODE_IDLING -DREMOVE_GPIO_FROM_NODE_RESET_INFO
-DCHECK_HEALTHY_BOOT -DIDLE_PERIPHERALS"

FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI_append = " file://my_changes.patch"
```

FSBL

FSBL のユーザー固有レシピは、`dir:project-spec/meta-user/recipes-bsp/fsbl/fsbl_%.bbappend` にあります。存在しない場合、このパスに作成してください。FSBL コードは、[embeddedsw](#) GitHub リポジトリへのパッチによって変更できます。ソースコードは、`embeddedsw/tree/master/lib/sw_apps/zynqmp_fsbl` にあります。

これらのパッチを `project-spec/meta-user/recipes-bsp/fsbl/files` ディレクトリにコピーし、コピーしたパッチの名前を `fsbl_%.bbappend` ファイルに追加します。

例:

FSBL ソースに対する `my_changes.patch` を追加し、[表 12-2](#) に示したすべてのフラグをセットして有効にする場合、

`project-spec/meta-user/recipes-bsp/fsbl/fsbl_%.bbappend` ファイルを次のように変更します。XPS_BOARD_ZCU102 フラグはデフォルトで存在します。

```
YAML_COMPILER_FLAGS_append = " -DXPS_BOARD_ZCU102 -DFSBL_PROT_BYPASS"
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
SRC_URI_append = " file://my_changes.patch"
```

ATF

ATF のユーザー固有レシピは、
:dirproject-spec/meta-user/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_%.bbappend にあります。存在しない場合、このパスに作成してください。ATF ファイルは、[Arm トラステッド ファームウェア用の Git リポジトリ](#)にあります。

例:

ATF にウォーム リスタート フラグを追加するには、
project-spec/meta-user/recipes-bsp/arm-trusted-firmware/arm-trusted-firmware_%.bbappend を次のように変更します。

```
#
# Enabling warm restart feature
#
EXTRA_OEMAKE_append = " ZYNQMP_WARM_RESTART=1 "
```

Linux

Linux の構成を追加、変更する方法は数多く存在します。詳細は、『PetaLinux ツール資料: リファレンス ガイド』(UG1144) [\[参照 27\]](#) を参照してください。

Linux カーネルのユーザー固有レシピは、project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend にあります。存在しない場合、このパスに作成してください。

Linux ファイルは、[Linux 用の Git リポジトリ](#)にあります。

例:

SRAM 構成を Linux に追加するには、次の bsp.cfg ファイルを作成します。
project-spec/meta-user/recipes-kernel/linux/linux-xlnx/bsp.cfg

```
CONFIG_SRAM=y
```

このファイルを、Linux の次の bbappend ファイルに追加します。
project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend

```
SRC_URI += "file://bsp.cfg"
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
```

デバイス ツリーの変更

デバイス ツリーのユーザー固有レシピは、
project-spec/meta-user/recipes-bsp/device-tree/device-tree-generation_%.bbappend にあります。このファイルの内容は、次のとおりです。

```
SRC_URI_append = "\
    file://system-user.dtsi \
"
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
```

project-spec/meta-user/recipes-bsp/device-tree/files ディレクトリにある system-user.dtsi の内容は、次のとおりです。

```
/include/ "system-conf.dtsi"
/ {
};
```

このファイルで、DTS ノードを追加、削除、変更することにより、デバイス ツリーの機能を拡張できます。

例: DT ノードの追加 [remoteproc RPU スプリット モード]

オーバーレイ `dtsti` をファイル/ディレクトリに追加して、`bbappend` ファイルを更新し、`system-user.dtsi` に含めます。
`remoteproc` 関連のエントリを追加して RPU サブシステムのロード/アンロード/リスタートを有効にするために、
`remoteproc.dtsi` という名前の新しいオーバーレイ ファイルを追加します。

注記: これは、スプリット モードの場合です。ロックステップなど、その他の構成の場合については、OpenAMP の資料を参照してください。

ファイル: `remoteproc.dtsi`

```
/ {
    reserved-memory {
        #address-cells = <2>;

        #size-cells = <2>;

        ranges;

        rproc_0_reserved: rproc:dir3ed000000 {

            no-map;

            reg = <0x0 0x3ed00000 0x0 0x1000000>;

        };
    };

    power-domains {

        pd_r5_0: pd_r5_0 {

            #power-domain-cells = <0x0>;

            pd-id = <0x7>;

        };

        pd_r5_1: pd_r5_1 {

            #power-domain-cells = <0x0>;

            pd-id = <0x8>;

        };

        pd_tcm_0_a: pd_tcm_0_a {

            #power-domain-cells = <0x0>;

            pd-id = <0xf>;

        };
    };
}
```



```

pd_tcm_0_b: pd_tcm_0_b {
    #power-domain-cells = <0x0>;

    pd-id = <0x10>;
};

pd_tcm_1_a: pd_tcm_1_a {
    #power-domain-cells = <0x0>;

    pd-id = <0x11>;
};

pd_tcm_1_b: pd_tcm_1_b {
    #power-domain-cells = <0x0>;

    pd-id = <0x12>;
};
};
amba {

    r5_0_tcm_a: tcm:dirffe00000 {
        compatible = "mmio-sram";

        reg = <0x0 0xFFE00000 0x0 0x10000>;

        pd-handle = <&pd_tcm_0_a>;
    };

    r5_0_tcm_b: tcm:dirffe20000 {
        compatible = "mmio-sram";

        reg = <0x0 0xFFE20000 0x0 0x10000>;

        pd-handle = <&pd_tcm_0_b>;
    };

    r5_1_tcm_a: tcm:dirffe90000 {
        compatible = "mmio-sram";

        reg = <0x0 0xFFE90000 0x0 0x10000>;

        pd-handle = <&pd_tcm_1_a>;
    };

    r5_1_tcm_b: tcm:dirffeb0000 {
        compatible = "mmio-sram";
    };
};

```

```

        reg = <0x0 0xFFEB0000 0x0 0x10000>;

        pd-handle = <&pd_tcm_1_b>;

    };

elf_dds_0: ddr:dir3ed00000 {

    compatible = "mmio-sram";

    reg = <0x0 0x3ed00000 0x0 0x40000>;

};

elf_dds_1: ddr:dir3ed40000 {

    compatible = "mmio-sram";

    reg = <0x0 0x3ed40000 0x0 0x40000>;

};

test_r50: zynqmp_r5_rproc:dir0 {

    compatible = "xlnx,zynqmp-r5-remoteproc-1.0";

    reg = <0x0 0xff9a0100 0x0 0x100>, <0x0 0xff340000 0x0 0x100>, <0x0
0xff9a0000 0x0 0x100>;

    reg-names = "rpu_base", "ipi", "rpu_glbl_base";

    dma-ranges;

    core_conf = "split0";

    sram_0 = <&r5_0_tcm_a>;

    sram_1 = <&r5_0_tcm_b>;

    sram_2 = <&elf_dds_0>;

    pd-handle = <&pd_r5_0>;

    interrupt-parent = <&gic>;

    interrupts = <0 29 4>;

};

test_r51: zynqmp_r5_rproc:dir1 {

    compatible = "xlnx,zynqmp-r5-remoteproc-1.0";

    reg = <0x0 0xff9a0200 0x0 0x100>, <0x0 0xff340000 0x0 0x100>, <0x0 0xff9a0000 0x0
0x100>;

    reg-names = "rpu_base", "ipi", "rpu_glbl_base";

    dma-ranges;

```

```

        core_conf = "split1";

        sram_0 = <&r5_1_tcm_a>;

        sram_1 = <&r5_1_tcm_b>;

        sram_2 = <&elf_dds_1>;

        pd-handle = <&pd_r5_1>;

        interrupt-parent = <&gic>;

        interrupts = <0 29 4>;

    } ;
};
};

```

次に、このノードを `system-user.dtsi` に含めます。

```

/include/ "system-conf.dtsi"
/include/ "remoteproc.dtsi"
/ {
};

```

OpenAMP および `remoteproc` の詳細は、[OpenAmp の Wiki ページ \(英語\)](#) を参照してください。

例: DT ノードの削除 [PL ノード]

アクセスしない PL ノードや、APU サブシステムに依存しない PL ノードはデバイス ツリーから削除する必要があります。ノードまたはプロパティを削除する場合も、`project-spec/meta-user/recipes-bsp/device-tree/files` にある `system-user.dtsi` を編集します。

たとえば、`dtb` から AXI DMA ノードを削除する場合は `system-user.dtsi` を次のように変更します。

```

/include/ "system-conf.dtsi"
/include/ "remoteproc.dtsi"
/ {
/delete-node/axi-dma;
};

```

高速バス インターフェイス

はじめに

Zynq® UltraScale+™ MPSoC デバイスには、高速シリアル インターフェイス用のシリアル入出力ユニット (SIU) が 1 つあります。このユニットは、PCIe™、USB 3.0、DisplayPort、SATA、およびイーサネットなどのプロトコルをサポートします。

- SIU ブロックは PS のフル電力ドメイン (FPD) に属します。
- Zynq UltraScale+ MPSoC デバイスの低電力ドメイン (LPD) に属する USB およびイーサネット コントローラー ブロックも PS-GTR トランシーバーを共有します。
- 4 つの PS-GTR トランシーバーは、インターコネクト マトリックスによって複数のコントローラー ブロック間でさまざまな組み合わせの多重化が可能です。
- SIU 内の信号は、レジスタ ブロックを使用して制御および監視します。

この章では、高速インターフェイスプロトコルのコンフィギュレーションフローについて説明します。

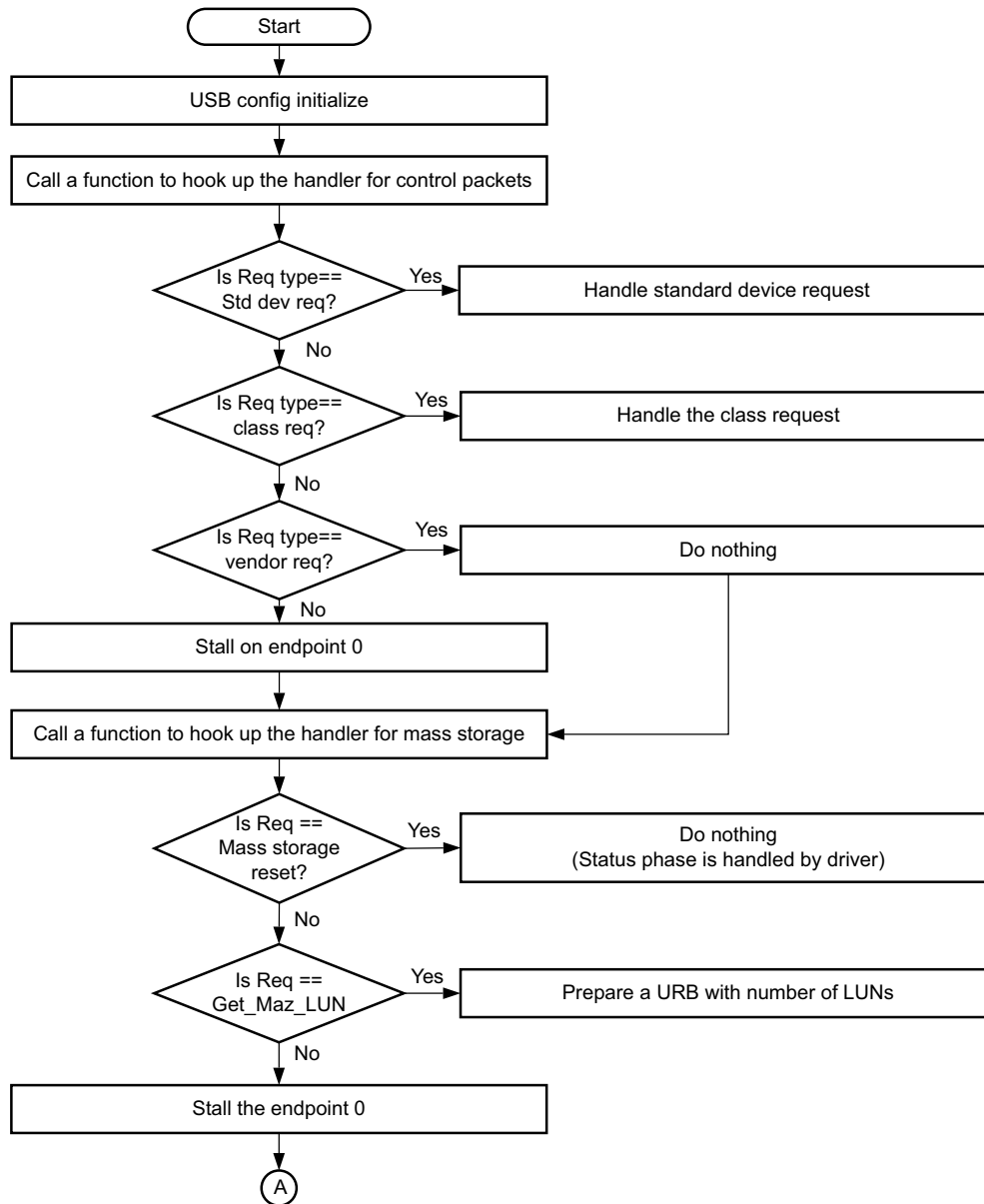
詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「高速 PS-GTR トランシーバー インターフェイス」の章を参照してください。

USB 3.0

Zynq UltraScale+ MPSoC の USB 3.0 コントローラーは、2 つの独立したデュアル ロール デバイス (DRD) コントローラーで構成されます。これらのコントローラーは、いつでもホスト モードまたはデバイス モードに個別に設定できます。USB 3.0 DRD コントローラーからシステム ソフトウェアへは、AXI (Advanced eXtensible Interface (AXI) スレーブ インターフェイスを介して xHCI (eXtensible Host Controller Interface) が提供されます。

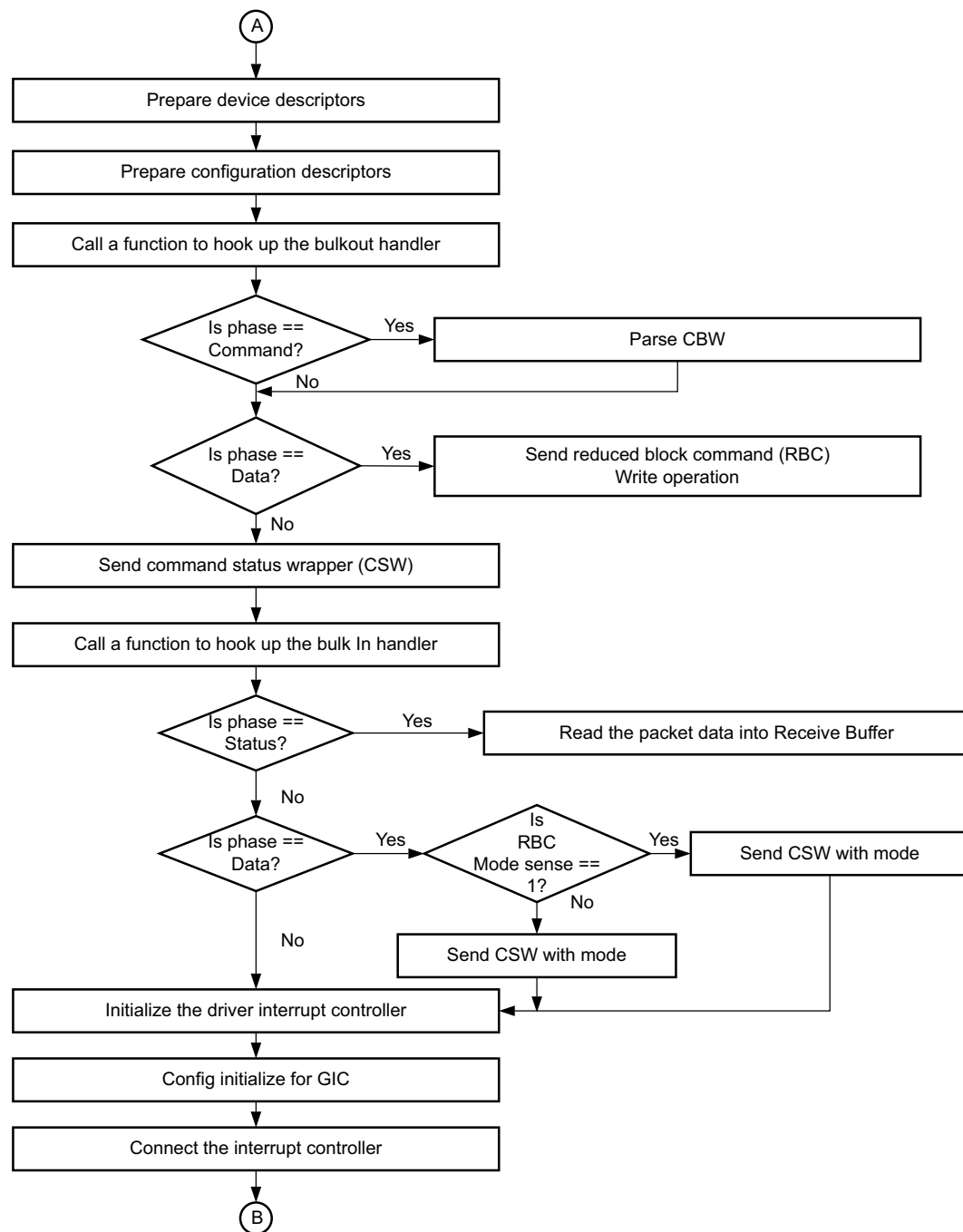
- このコントローラーには内部 DMA エンジンがあり、AXI マスター インターフェイスを使用してデータを転送します。
- 3 つのデュアル ポート RAM コンフィギュレーションは、RX データ FIFO、TX データ FIFO、およびディスクリプター / レジスタ キャッシュを実装します。

次に、USB をマス ストレージ デバイスとして設定する方法をフローチャートで示します。



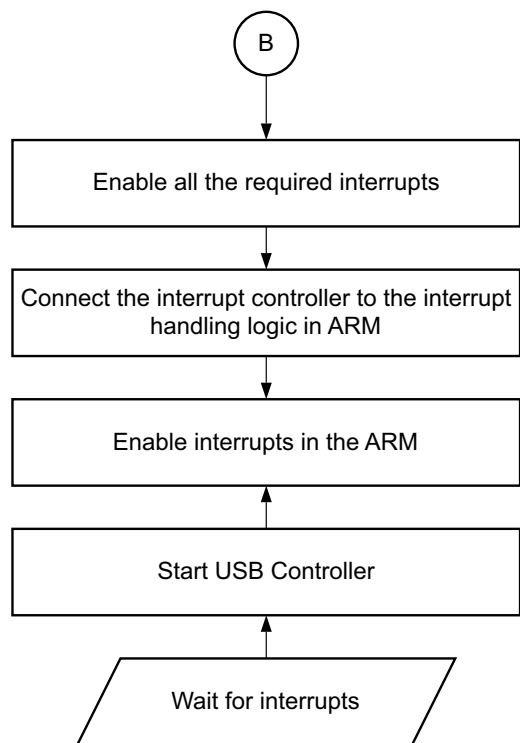
X15463-071017

図 13-1: USB フローの例: USB の初期化



X15477-071017

図 13-2: USB フローの例: バルク IN およびバルク OUT ハンドラーの接続および割り込みコントローラーの初期化



X15478-021317

図 13-3: 割り込みの有効化および USB コントローラーの起動

USB コントローラーの詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「USB 2.0/3.0 ホスト/デバイス/OTG コントローラー」の章を参照してください。

ギガビット イーサネット コントローラー

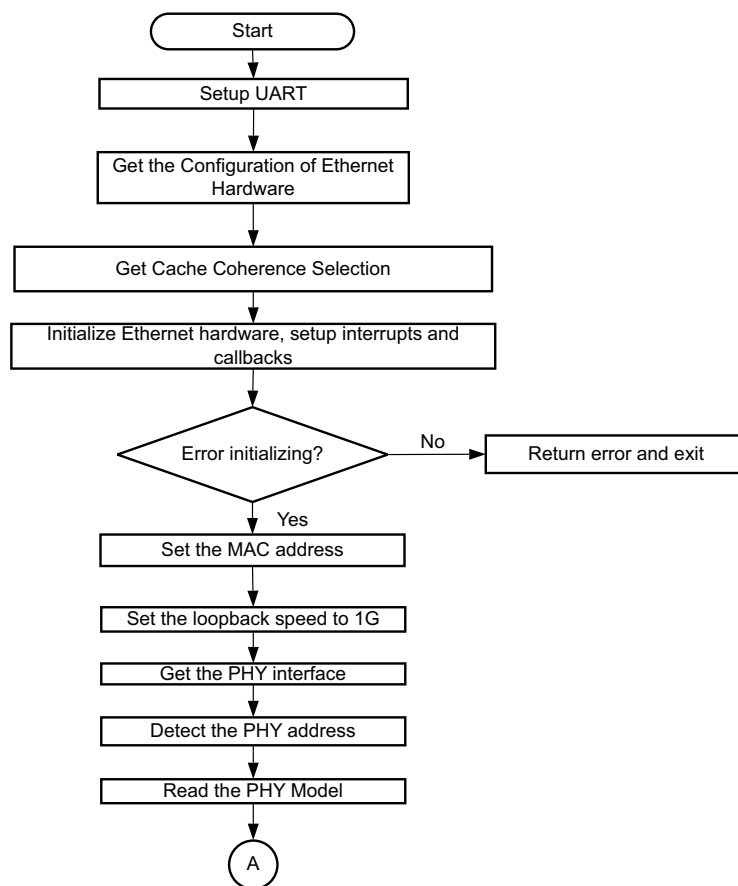
ギガビット イーサネット コントローラー (GEM) は IEEE イーサネット 規格 (IEEE Std 802.3-2008) に準拠した 10/100/1000Mb/s イーサネット MAC を実装しており、半二重または全二重 10/100 モード、および全二重 1000 モードでの動作が可能です。

プロセッシング システム (PS) には 4 つのギガビット イーサネット コントローラーがあります。

MAC の機能および動作モードはレジスタで設定します。

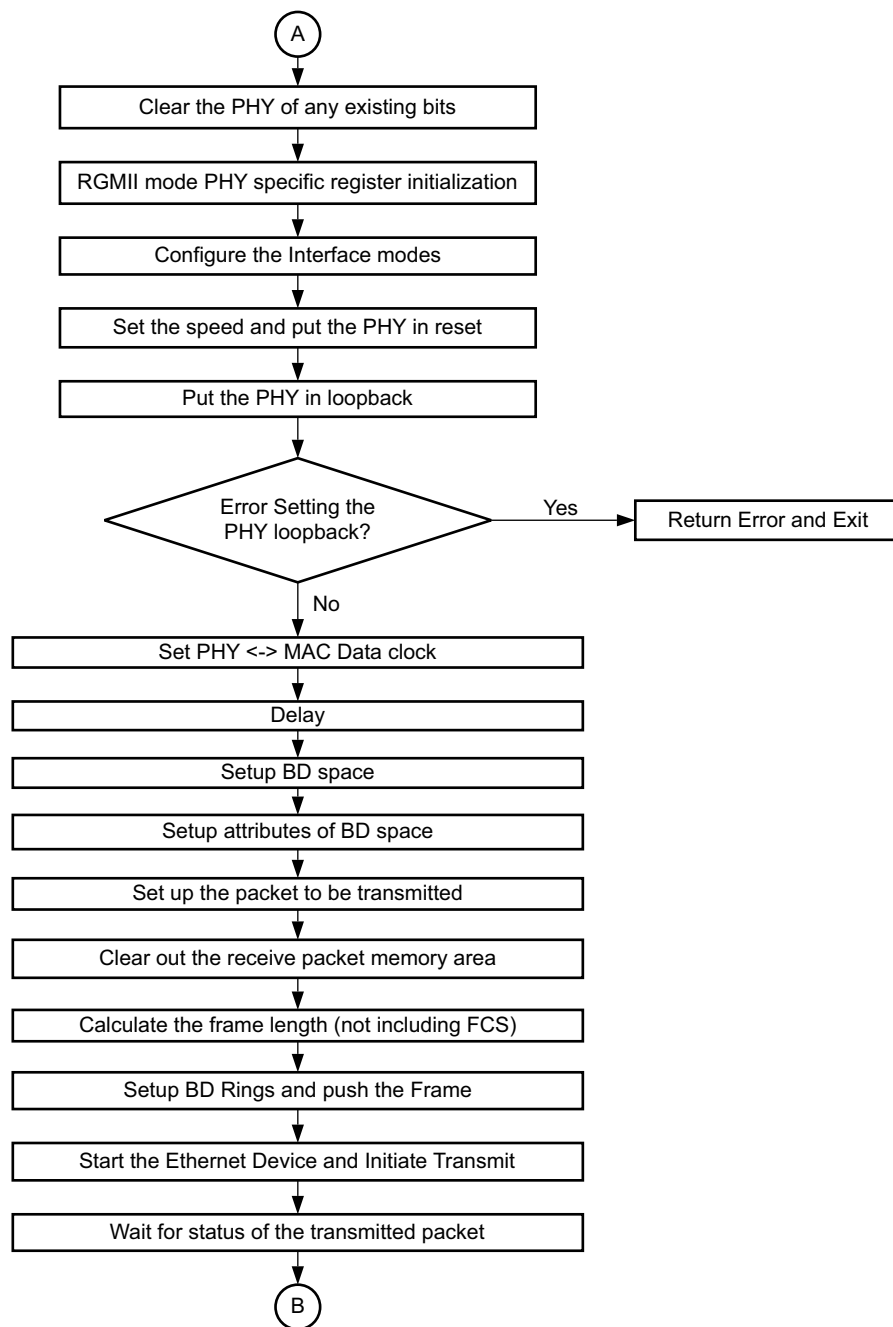
DMA コントローラーは AXI (Advanced eXtensible Interface) 経由でメモリに接続します。このコントローラーは MAC コントローラーの FIFO インターフェイスに接続されており、パケット データをスキャッター ギャザー方式でプロセッシング システムに格納できます。

次に、1 パケットのデータを RGMII モードで送信するようにイーサネット コントローラーを設定する場合の例をフローチャートで示します。



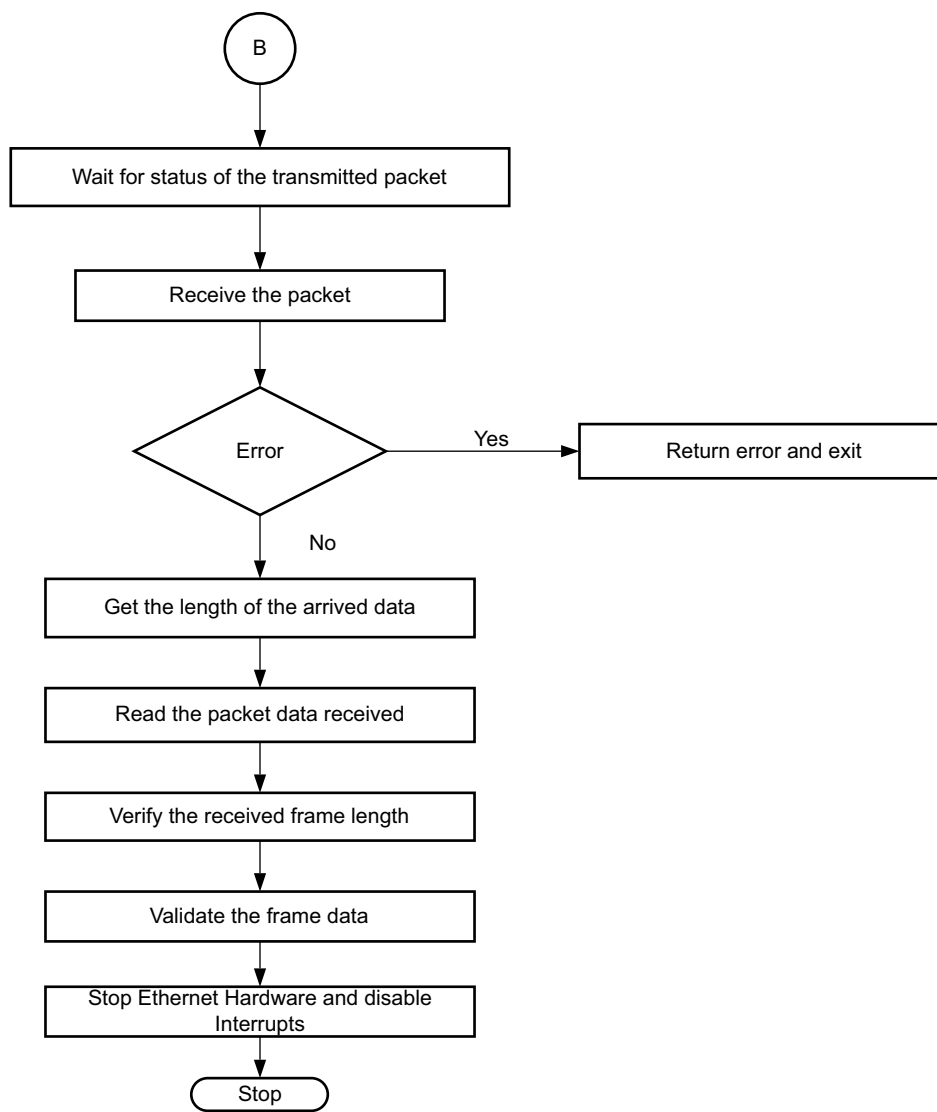
X15462-071017

図 13-4: イーサネット フローの例: イーサネット コントローラーの初期化



X15479-071017

図 13-5: イーサネット フローの例: イーサネット パラメーターの設定および送信の開始



X15480-021317

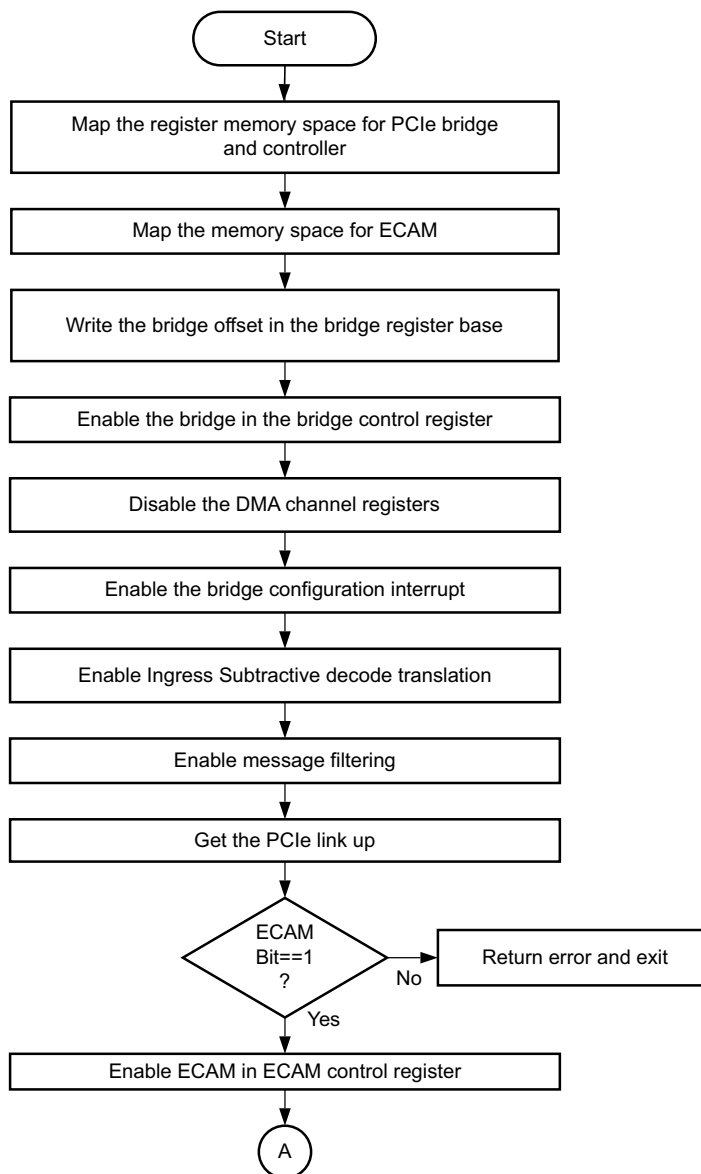
図 13-6: イーサネット フローの例: データの受信および検証

イーサネット コントローラーの詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「ギガビット イーサネット コントローラー」の章を参照してください。

PCI Express

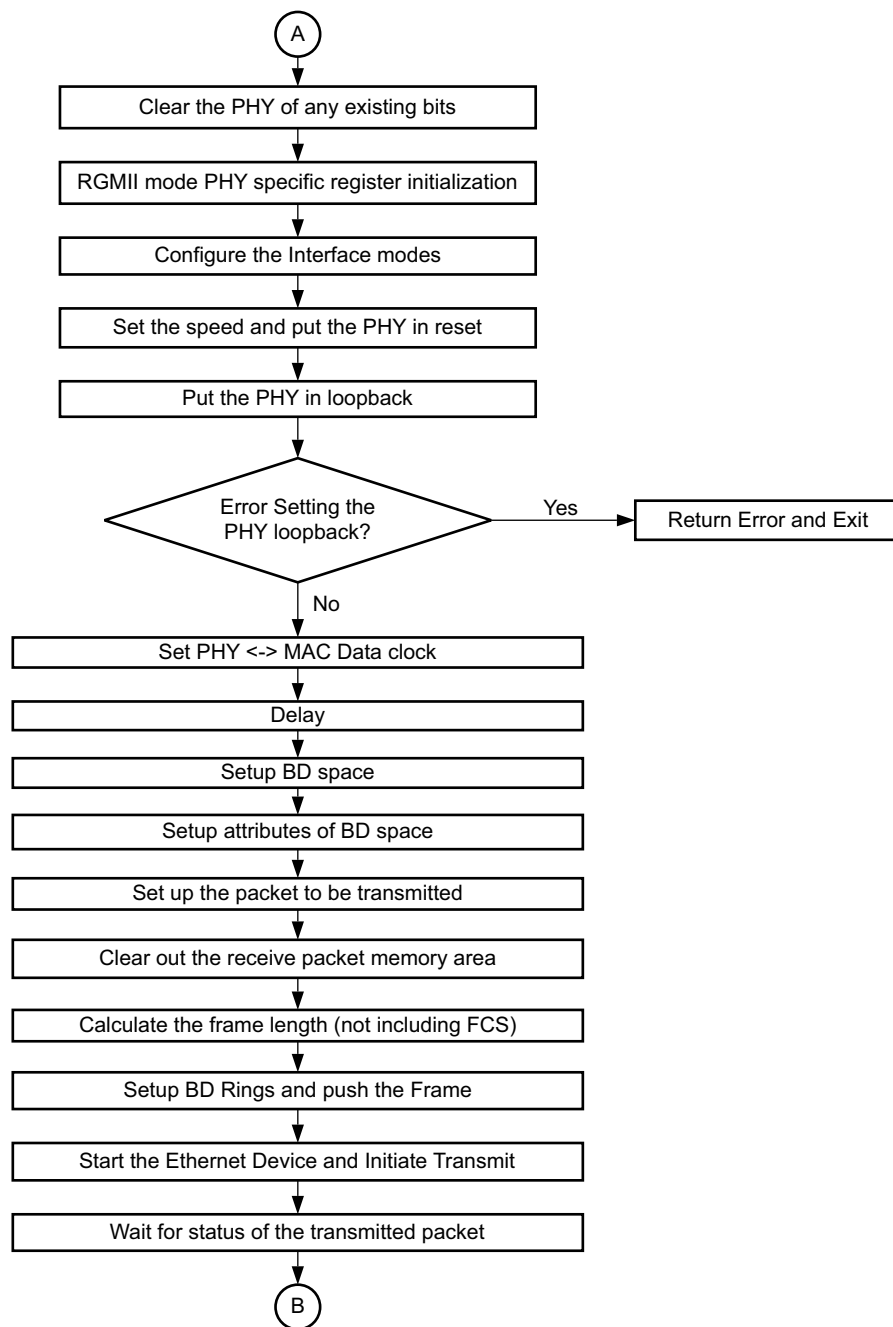
Zynq UltraScale+ MPSoC デバイスには、PCI Express™ v2.1 準拠の統合ブロック、AXI-PCIe ブリッジ、および DMA モジュールで構成される PCIe コントローラーがあります。AXI-PCIe ブリッジは PCIe と AXI を高速にブリッジ接続します。

次に、データ転送用に PCIe ルート コМПレックスを設定する場合の例をフローチャートで示します。



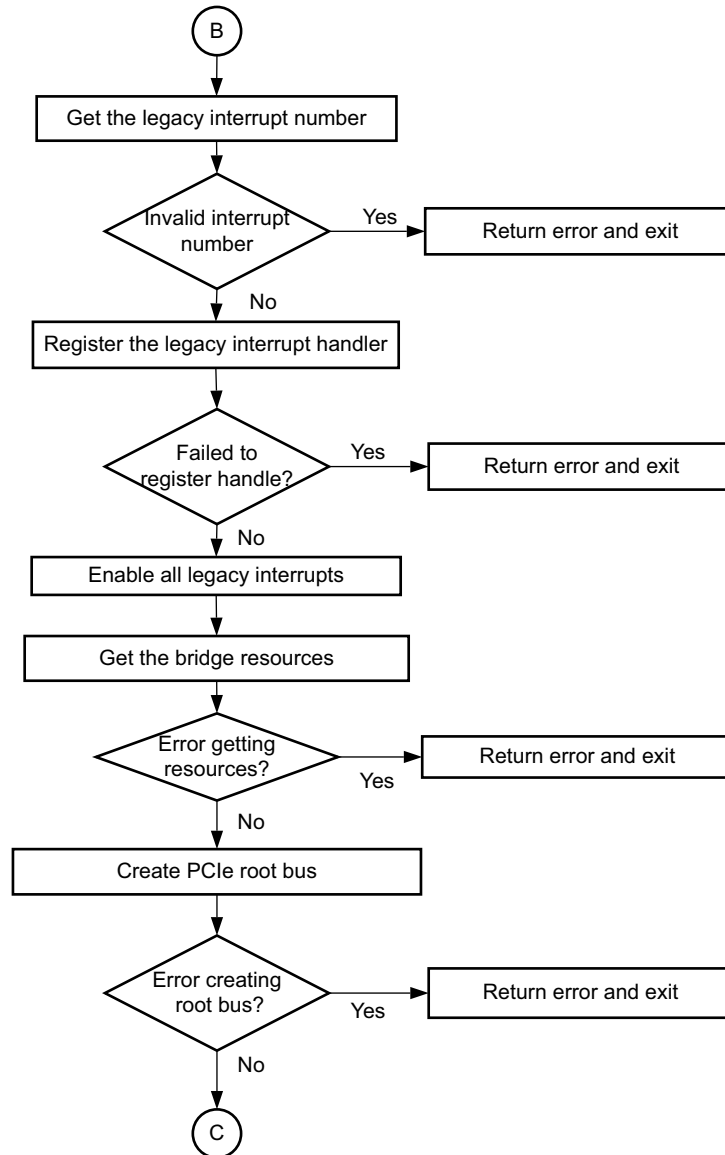
X15481-071217

図 13-7: PCIe フローの例: レガシ割り込みの有効化および PCIe ルート バスの作成



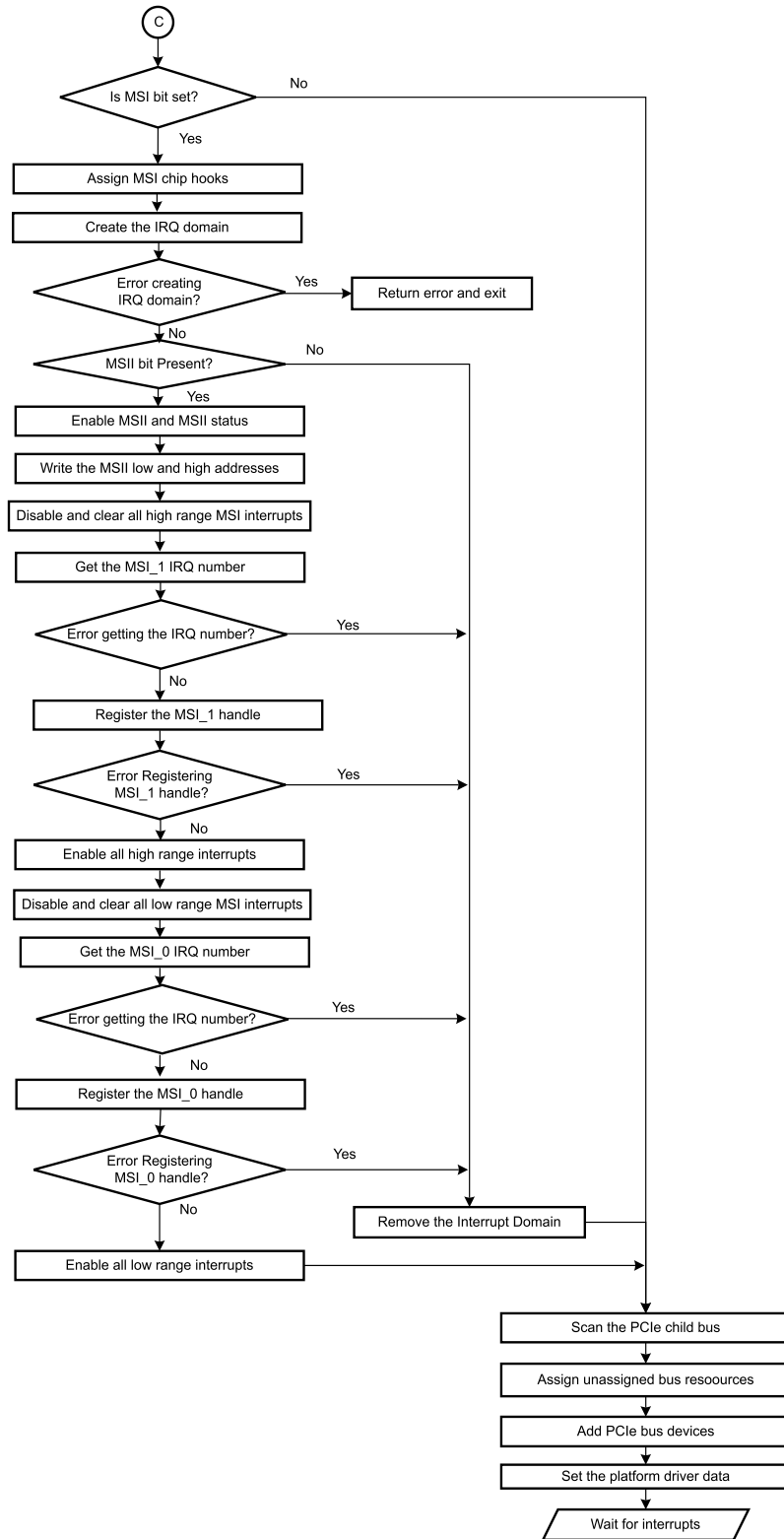
X15479-071017

図 13-8: PCIe フローの例: PCIe パラメーターの設定および送信の開始



X15483-071217

図 13-9: PCIe フローの例: レガシ割り込みの有効化および PCIe ルート バスの作成



X15484-071217

図 13-10: PCIe フローの例: MSI 割り込みの有効化および割り込みの待機

注記: エンドポイントの操作については、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「PCI Express コントローラー」の章を参照してください。

PCIe ブリッジと ECAM のメモリ空間をマップ後、ECAM 変換を実行できるよう ECAM を有効にします。次に、バス範囲を取得してバス番号を設定し、primary、secondary、および subordinate バス番号を書き込みます。

割り込みシステムをセットアップする際は、レガシ割り込みを含むすべての種類の割り込みを有効にする必要があります。PCI ホストブリッジ デバイス ノードの ranges プロパティを解析し、その内容に基づいてリソース マップをセットアップできます。

ルート バスを作成するには、PCIe ルート バスを割り当て、そのバスに初期リソースを追加します。

MSI ビットがセットされている場合、MSI (Message Signalling Interrupt) を有効にする必要があります。

MSI 割り込みを設定したら、PCIe スロットをスキャンして PCIe バス全体をエニユメレートし、スキャンしたバスに対してバス リソースを割り当てます。

これで、PCIe デバイスをシステムに追加できます。

PCI Express の詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「DMA コントローラー」の章および「PCI Express コントローラー」の章を参照してください。

クロックおよび周波数管理

はじめに

Zynq® UltraScale+™ MPSoC デバイス アーキテクチャには、PS の位相ロックループ (PLL) ブロックを使用して、ある一定の入力クロック周波数から複数のクロックを生成するプログラマブルなクロック ジェネレーターがあります。各 PLL からの出力クロックを、PS の各種ペリフェラルの基準クロックとして使用します。

USB およびイーサネット ペリフェラルを除き、UART や SD などのペリフェラルではデバイスの周波数設定を動的に変更できます。

この章では、これらペリフェラルの動作周波数を動的に変更する方法について説明します。クロック周波数の低減および調整の詳細は、「[電力管理フレームワーク](#)」を参照してください。

ペリフェラルの周波数変更

ペリフェラルの動作周波数は、各ペリフェラルのクロック コンフィギュレーション レジスタで周波数を直接設定して変更できます。Zynq UltraScale+ MPSoC の BSP には、ユーザー要件に合わせてペリフェラルのクロック周波数を動的に変更するための API が用意されています。

次の表に、ペリフェラルの周波数を変更するために使用するスタンドアロン API を示します。

表 14-1: スタンドアロン API

API	説明
XSDPS_change_clkfreq	SD のクロック周波数を変更します。
XSPIPS_setclkprescaler XSPIPS_getclkprescaler	SPI 周波数のプリスケアラです。
XRtcPSu_calculatecalibration	オシレーターの周波数を変更します。
XQSPIPSU_setclkprescaler	QSPI のクロック周波数を変更します。

Linux アプリケーションの場合、すべてのペリフェラルの周波数をデバイス ツリー ファイルで設定します。次のコードの一部は、ペリフェラルのクロック設定方法を示しています。

```
ps7_qspi_0: ps7-qspi:dir0xFF0F0000 {
    #address-cells = <0x1>;
    #size-cells = <0x0>;
    #bus-cells = <0x1>;
    clock-names = "ref_clk", "pclk";
    compatible = "xlnx,usmp-gqspi", "cdns,spi-r1p6";
    stream-connected-dma = <0x26>;
    clocks = <0x1e 0x1e>;
    dma = <0xb>;
    interrupts = <0xf>;
    num-chip-select = <0x2>;
    reg = <0x0 0xff0f0000 0x1000 0x0 0xc0000000 0x8000000>;
    speed-hz = <0xbebc200>;
    xlnx,fb-clk = <0x1>;
    xlnx,qspi-clk-freq-hz = <0xbebc200>;
    xlnx,qspi-mode = <0x2>;
}
```

エラー条件の発生を防ぐため、ペリフェラルのクロック周波数を変更する前にそのペリフェラルを停止しておく必要があります。

ペリフェラルのクロック周波数を変更するには、次の手順を実行します。

1. クロック周波数を変更するペリフェラル (IP) に関する遷移を停止し、アイドル状態にします。
2. レジスタの設定を変更して IP を停止します。
3. ペリフェラルのクロック周波数を変更します。
4. IP に対してソフト リセットを発行します。
5. IP を再起動します。

Zynq UltraScale+ MPSoC のクロック ジェネレーターの詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] の「クロッキング」のセクションを参照してください。

ターゲット開発プラットフォーム

はじめに

この章では、Quick Emulator (QEMU) および Zynq UltraScale+ MPSoC ボード/キットなどの Zynq® UltraScale+™ MPSoC デバイス向けに提供される各種開発プラットフォームについて説明します。

QEMU

QEMU は、Intel 互換の Linux ホスト システム上で動作するシステム エミュレーション モデルです。ザイリンクス QEMU は、コマンド ラインから渡されたデバイス ツリーに基づいてカスタム マシン モデルを生成するフレームワークを実装しています。QEMU の詳細は、『Zynq UltraScale+ MPSoC QEMU: ユーザー ガイド』(UG1169) [\[参照 8\]](#) を参照してください。

ボード/キット

ザイリンクスは、開発者向けに Zynq UltraScale+ MPSoC ZCU102 評価キットを提供しています。ZCU102 評価キットの詳細は、ザイリンクス アンサー : 66249 「Zynq UltraScale+ MPSoC、ZCU102 評価キット - ZCU102 入門ガイド」 [\[参照 36\]](#) を参照してください。

各種 Zynq UltraScale+ MPSoC デバイスについては、Zynq UltraScale+ MPSoC の製品ページ [\[参照 7\]](#) を参照してください。

ブート イメージの生成

はじめに

Zynq® UltraScale+™ MPSoC は、セキュア ブートと非セキュア ブートの両方をサポートしています。フィールドで運用中のデバイス上で、認証されていないコードや変更されたコードが実行されないようにすることが重要です。

Zynq UltraScale+ MPSoC は、アプリケーションを安全にホスティングするために必要な機密性、完全性、および認証の機能を備えています。セキュリティ機能の詳細は、『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085) [参照 11] を参照してください。

Zynq UltraScale+ MPSoC には、設計意図に従って機能するようにデバイスをブートするためのハードウェアおよびソフトウェア バイナリが数多く備わっています。このようなバイナリには、FPGA ビットストリーム、ファームウェア、ブートローダー、オペレーティング システム、ユーザー指定のアプリケーションが含まれます。たとえば、FPGA ビットストリーム ファイル、FSBL (第 1 段階ブートローダー)、PMU ファームウェア、ATF、U-Boot、Linux カーネル、Rootfs、デバイス ツリー、スタンドアロンまたは RTOS アプリケーションなどです。ザイリンクスがスタンドアロン ツールとして提供する Bootgen は、これらすべてのバイナリ イメージを 1 つにまとめ、ザイリンクスのローダー プログラムが解釈可能なフォーマットのデバイス ブート イメージを生成します。

Bootgen には多くの属性とコマンドがあり、これらを使用してブート イメージを生成する際の動作を定義します。これにより、セキュア ブート イメージの生成、非セキュア ブート イメージの生成、セキュア キーの生成、HMI モードなどがサポートされます。Bootgen ツールの入手法、インストール手順、Zynq UltraScale+ のブート イメージフォーマット、Bootgen のコマンド、属性、およびブート イメージの生成手順と例などの詳細は、『Bootgen ユーザー ガイド』(UG1283) [参照 23] を参照してください。

その他のリソースおよび法的通知

ザイリンクス リソース

アンサー、資料、ダウンロード、フォーラムなどのサポート リソースは、[ザイリンクス サポート](#) サイトを参照してください。

ソリューション センター

デバイス、ツール、IP のサポートについては、[ザイリンクス ソリューション センター](#)を参照してください。デザイン アシスタント、デザイン アドバイザリ、トラブルシューティングのヒントなどが含まれます。

Documentation Navigator およびデザイン ハブ

ザイリンクス Documentation Navigator (DocNav) では、ザイリンクスの資料、ビデオ、サポート リソースにアクセスでき、特定の情報を取得するためにフィルター機能や検索機能を利用できます。DocNav を開くには、次のいずれかを実行します。

- Vivado IDE で [Help] → [Documentation and Tutorials] をクリックします。
- Windows で [スタート] → [すべてのプログラム] → [Xilinx Design Tools] → [DocNav] をクリックします。
- Linux コマンド プロンプトに「docnav」と入力します。

ザイリンクス デザイン ハブには、資料やビデオへのリンクがデザイン タスクおよびトピックごとにまとめられており、これらを参照することでキー コンセプトを学び、よくある質問 (FAQ) を参考に問題を解決できます。デザイン ハブにアクセスするには、次のいずれかを実行します。

- DocNav で [Design Hubs View] タブをクリックします。
- ザイリンクス ウェブサイトの[デザイン ハブ](#) ページを参照します。

注記: DocNav の詳細は、ザイリンクス ウェブサイトの [Documentation Navigator](#) ページを参照してください。



注意: DocNav からは、日本語版は参照できません。ウェブサイトのデザイン ハブ ページをご利用ください。

参考資料

注記: 日本語版のバージョンは、英語版より古い場合があります。

ザイリンクスのリソース

1. ザイリンクス サードパーティのライセンスに関するソリューション センター
2. [PetaLinux の製品ページ](#)
3. [ザイリンクス Vivado Design Suite – HLx Edition](#)
4. [ザイリンクス サードパーティ ツール](#)
5. [Zynq UltraScale+ MPSoC 製品一覧](#)
6. [Zynq UltraScale+ MPSoC 製品の特長](#)
7. [Zynq UltraScale+ MPSoC の製品ページ](#)

Zynq 関連資料

8. 『Quick EMUlator (QEMU) ユーザー ガイド』([UG1169](#))
9. 『UltraScale アーキテクチャおよび製品データシート: 概要』(DS890: [英語版](#)、[日本語版](#))
10. 『Zynq UltraScale+ MPSoC のアイソレーション手法』([XAPP1320](#))
11. 『Zynq UltraScale+ MPSoC テクニカル リファレンス マニュアル』(UG1085: [英語版](#)、[日本語版](#))
12. 『Zynq UltraScale+ MPSoC レジスタ リファレンス』([UG1087](#))
13. 『Zynq UltraScale+ MPSoC: エンベデッド デザイン チュートリアル』(UG1209: [英語版](#)、[日本語版](#))
14. 『Zynq UltraScale+ MPSoC Processing System LogiCORE IP 製品ガイド』(PG201: [英語版](#)、[日本語版](#))
15. 『UltraScale アーキテクチャ システム モニター ユーザー ガイド』(UG580: [英語版](#)、[日本語版](#))
16. 『Zynq デバイス用 Libmetal および OpenAMP ユーザー ガイド』(UG1186: [英語版](#)、[日本語版](#))
17. 『Embedded Energy Management Interface Specification』([UG1200](#))
18. 『UltraFast エンベデッド デザイン設計手法ガイド』(UG1046: [英語版](#)、[日本語版](#))
19. 『Zynq-7000 SoC: エンベデッド デザイン チュートリアル』(UG1165: [英語版](#)、[日本語版](#))
20. 『Zynq-7000 SoC ソフトウェア開発者向けガイド』(UG821: [英語版](#)、[日本語版](#))
21. 『UltraScale アーキテクチャ PCB デザイン ユーザー ガイド』(UG583: [英語版](#)、[日本語版](#))
22. [Vivado Design Suite に関する資料](#)
23. 『Bootgen ユーザー ガイド』(UG1283: [英語版](#)、[日本語版](#))

SDK および PetaLinux 関連資料

24. 『ザイリンクス ソフトウェア開発キットのヘルプ』([UG782](#))
25. 『OS およびライブラリ資料コレクション』([UG643](#))
26. エンベデッド開発ツール ([ダウンロード](#))
27. 『PetaLinux ツール資料: リファレンス ガイド』(UG1144: [英語版](#)、[日本語版](#))
28. 『ザイリンクス ソフトウェア開発キット (SDK) ユーザー ガイド: システム パフォーマンス解析』(UG1145: [英語版](#)、[日本語版](#))

ザイリンクス IP 関連資料

29. 『LogiCORE IP AXI Central Direct Memory Access 製品ガイド』([PG034](#))
30. 『LogiCORE IP AXI Video Direct Memory Access 製品ガイド』([PG020](#))

その他のリンク

31. [ザイリンクス Github](#)
32. [エンベデッド開発](#)
33. [meta-xilinx](#)
34. [PetaLinux ソフトウェア開発](#)
35. [Zynq UltraScale+ シリコン デバイス ページ](#)
36. ザイリンクス アンサー: [66249](#)
37. [Vivado Quick Take ビデオ: 「Vivado Processor Configuration Wizard の概要」](#)
38. [ザイリンクス Wiki](#)

サードパーティのリソース

39. [Lauterbach テクノロジ](#)
40. [ARM トラストッド ファームウェア](#)
41. [Xen ハイパーバイザー](#)
42. [ARM デベロッパ リソース](#)
43. 『[ARM Cortex-A53 MPCore Processor Technical Reference Manual](#)』
44. [Yocto Project 開発](#)
45. [GNU FTP](#)
46. 『Power State Coordination Interface』– Arm DEN 0022B.b、2013/6/25

お読みください: 重要な法的通知

本通知に基づいて貴殿または貴社 (本通知の被通知者が個人の場合には「貴殿」、法人その他の団体の場合には「貴社」。以下同じ) に開示される情報 (以下「本情報」といいます) は、ザイリンクスの製品を選択および使用することのためにのみ提供されます。適用される法律が許容する最大限の範囲で、(1) 本情報は「現状有姿」、およびすべて受領者の責任で (with all faults) という状態で提供され、ザイリンクスは、本通知をもって、明示、黙示、法定を問わず (商品性、非侵害、特定目的適合性の保証を含みますがこれらに限られません)、すべての保証および条件を負わない (否認する) ものとし、(2) ザイリンクスは、本情報 (貴殿または貴社による本情報の使用を含む) に関し、起因し、関連する、いかなる種類・性質の損失または損害についても、責任を負わない (契約上、不法行為上 (過失の場合を含む)、その他のいかなる責任の法理によるかを問わない) ものとし、当該損失または損害には、直接、間接、特別、付随的、結果的な損失または損害 (第三者が起こした行為の結果被った、データ、利益、業務上の信用の損失、その他あらゆる種類の損失や損害を含みます) が含まれるものとし、それは、たとえ当該損害や損失が合理的に予見可能であったり、ザイリンクスがそれらの可能性について助言を受けていた場合であったとしても同様です。ザイリンクスは、本情報に含まれるいかなる誤りも訂正する義務を負わず、本情報または製品仕様のアップデートを貴殿または貴社に知らせる義務も負いません。事前の書面による同意のない限り、貴殿または貴社は本情報を再生産、変更、頒布、または公に展示してはなりません。一定の製品は、ザイリンクスの限定的保証の諸条件に従うこととなるので、<https://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。IP コアは、ザイリンクスが貴殿または貴社に付与したライセンスに含まれる保証と補助的条件に従うことになります。ザイリンクスの製品は、フェイルセーフとして、または、フェイルセーフの動作を要求するアプリケーションに使用するために、設計されたり意図されたりしていません。そのような重大なアプリケーションにザイリンクスの製品を使用する場合のリスクと責任は、貴殿または貴社が単独で負うものです。<https://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。

自動車用のアプリケーションの免責条項

オートモーティブ製品 (製品番号に「XA」が含まれる) は、ISO 26262 自動車用機能安全規格に従った安全コンセプトまたは余剰性の機能 (「セーフティ設計」) がない限り、エアバッグの展開における使用または車両の制御に影響するアプリケーション (「セーフティアプリケーション」) における使用は保証されていません。顧客は、製品を組み込むすべてのシステムについて、その使用前または提供前に安全を目的として十分なテストを行うものとします。セーフティ設計なしにセーフティアプリケーションで製品を使用するリスクはすべて顧客が負い、製品の責任の制限を規定する適用法令および規則にのみ従うものとします。

© Copyright 2015-2019 Xilinx, Inc. Xilinx、Xilinx のロゴ、Alveo、Artix、Kintex、Spartan、Versal、Virtex、Vivado、Zynq、およびこの文書に含まれるその他の指定されたブランドは、米国およびその他の各国のザイリンクス社の商標です。すべてのその他の商標は、それぞれの所有者に帰属します。AMBA、AMBA Designer、Arm、ARM1176JZ-S、CoreSight、Cortex、PrimeCell、Mali、MPCore は EU およびその他の各国の Arm Limited の登録商標です。MATLAB および Simulink は The MathWorks, Inc. の登録商標です。PCI、PCIe、および PCI Express は PCI-SIG の商標であり、ライセンスに基づいて使用されています。

この資料に関するフィードバックおよびリンクなどの問題につきましては、jpn_trans_feedback@xilinx.com まで、または各ページの右下にある [フィードバック送信] ボタンをクリックすると表示されるフォームからお知らせください。フィードバックは日本語で入力可能です。いただきましたご意見を参考に早急に対応させていただきます。なお、このメールアドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。