

Versal ACAP システム ソフトウェア開発者向けガイド

UG1304 (v2020.2) 2020 年 11 月 24 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。

| セクション | 改訂内容 |
|--------------------------------|---|
| 2020 年 11 月 24 日 バージョン 2020.2 | |
| PLM ブートおよびコンフィギュレーション | JTAG の情報を更新し、PLM エラー コードを追加。 PLM インターフェイス (XilPLMI) に PLM ウォッチドッグ タイマーの詳細を追加。 XilLoader でイメージの遅延ロードとイメージの遅延ハンドオフの詳細を更新。 PLM の使用 で PLM のビルド フラグと PLM 用の予約メモリおよびレジスタを更新。 |
| 第 9 章: セキュリティ | ECDSA-RSA ハードウェア アクセラレータおよび AES-GCM ハードウェア暗号化ブロックの詳細を更新。 |
| 第 10 章: Versal ACAP プラットフォーム管理 | XPm_DevIoctl EEMI API を更新。 |
| 2020 年 7 月 16 日 バージョン 2020.1 | |
| 初版。 | N/A |

目次

| | |
|--|----|
| 改訂履歴..... | 2 |
| 第 1 章: 概要..... | 5 |
| Versal ACAP の概要..... | 5 |
| 設計プロセス別のコンテンツ ガイド..... | 6 |
| このユーザー ガイドについて..... | 6 |
| 第 2 章: Versal ACAP のプログラミング..... | 8 |
| ハードウェアの概要..... | 8 |
| 第 3 章: 開発ツール..... | 17 |
| Vivado Design Suite..... | 17 |
| Vitis ソフトウェア プラットフォーム..... | 18 |
| PetaLinux ツール..... | 21 |
| デバイス ツリー ジェネレーター..... | 23 |
| オープンソース..... | 23 |
| Yocto ツールを使用した Linux ソフトウェア開発..... | 24 |
| QEMU..... | 26 |
| AI エンジン開発環境..... | 28 |
| 第 4 章: ソフトウェア スタック..... | 31 |
| ベアメタル ソフトウェア スタック..... | 31 |
| Linux ソフトウェア スタック..... | 33 |
| サードパーティ ソフトウェア スタック..... | 36 |
| 第 5 章: ソフトウェア開発フロー..... | 37 |
| Vitis 環境でのベアメタル アプリケーション開発..... | 38 |
| PetaLinux ツールを使用した Linux アプリケーション開発..... | 39 |
| Vitis ソフトウェア プラットフォームを使用した Linux アプリケーション開発..... | 40 |
| 第 6 章: ソフトウェア デザインのパラダイム..... | 43 |
| マルチプロセッサ開発用フレームワーク..... | 43 |
| 対称型マルチプロセッシング..... | 44 |
| 非対称型マルチプロセッシング..... | 46 |
| 第 7 章: ブートおよびコンフィギュレーション..... | 51 |
| Versal ACAP のブート プロセス..... | 51 |
| ブート フロー..... | 56 |

| | |
|---|------------|
| セカンダリ ブート プロセスとデバイスの選択..... | 59 |
| フォールバック ブートとマルチブート..... | 60 |
| プログラマブル デバイス イメージ..... | 62 |
| コンフィギュレーション データ オブジェクト..... | 62 |
| ブート イメージ (PDI) の作成..... | 63 |
| 第 8 章: プラットフォーム ローダーおよびマネージャー..... | 66 |
| PLM ブートおよびコンフィギュレーション..... | 66 |
| PLM ソフトウェアの詳細..... | 70 |
| PLM インターフェイス (XilPLMI)..... | 79 |
| XilLoader..... | 88 |
| XilPM..... | 90 |
| XilSecure..... | 92 |
| XilSEM..... | 92 |
| PLM の使用..... | 92 |
| サービス フロー..... | 95 |
| 例外処理..... | 95 |
| 第 9 章: セキュリティ..... | 96 |
| セキュリティ 機能..... | 96 |
| 非対称型のハードウェアによる信頼のルート (A-HWRoT) (認証が必要)..... | 98 |
| 暗号化..... | 99 |
| 対称型のハードウェアによる信頼のルート (S-HWRot) ブート モード (暗号化が必要)..... | 100 |
| 第 10 章: Versal ACAP プラットフォーム管理..... | 101 |
| Versal ACAP プラットフォーム管理の概要..... | 102 |
| Versal ACAP の電源ドメイン..... | 102 |
| Versal ACAP のプラットフォーム管理ソフトウェア アーキテクチャ..... | 104 |
| 第 11 章: ターゲット開発プラットフォーム..... | 145 |
| ボードおよびキット..... | 145 |
| 付録 A: ライブラリ..... | 146 |
| 付録 B: その他のリソースおよび法的通知..... | 147 |
| ザイリンクス リソース..... | 147 |
| Documentation Navigator およびデザイン ハブ..... | 147 |
| 参考資料..... | 147 |
| お読みください: 重要な法的通知..... | 149 |

概要

Versal ACAP の概要

Versal™ ACAP (Adaptive Compute Acceleration Platform) はスカラー エンジン、適応型エンジン、およびインテリジェント エンジンを中心に、最先端のメモリおよびインターフェイス テクノロジーを組み合わせることによって、あらゆるアプリケーションで強力なヘテロジニアス アクセラレーションを実現します。Versal ACAP で最も重要な点は、ソフトウェア開発者やデータ サイエнтиストがハードウェア開発者と同様にハードウェアとソフトウェアをプログラムおよび最適化できることにあります。Versal ACAP は、さまざまなツール、ソフトウェア、ライブラリ、IP、ミドルウェア、およびフレームワークでサポートされ、業界標準のデザイン フローを活用できます。

TSMC 社の 7nm FinFET プロセス テクノロジーを採用した Versal ポートフォリオは、ソフトウェア プログラマビリティと特定分野に向けたハードウェア アクセラレーションに適応性を兼ね備え、現代の急速なイノベーションに対応できるようにした初のプラットフォームです。6つのシリーズで構成されるこのデバイス ポートフォリオは、独自のアーキテクチャによりクラウド、ネットワーク、無線通信、エッジ コンピューティング、エンドポイントなど、幅広い市場における多くのアプリケーションで優れたスケーラビリティと AI 推論能力を発揮します。

Versal アーキテクチャは、異なるタイプのエンジン、さまざまなコネクティビティおよび通信機能、ネットワーク オン チップ (NoC) を組み合わせており、デバイスの高さおよび幅全体にスムーズなメモリ マップド アクセスが可能です。インテリジェント エンジンは、適応型の推論および高度な信号処理演算向けの SIMD VLIW AI エンジンと、固定小数点、浮動小数点、および複素 MAC 演算向けの DSP エンジンです。適応型エンジンは、高い演算密度を達成できるよう設計されたプログラマブル ロジック ブロックとメモリです。スカラー エンジンには Arm® Cortex™-A72 および Cortex-R5F プロセッサが含まれ、計算負荷の高いタスクを可能にします。

VersalAI コア シリーズは、現在のサーバー クラス CPU の 100 倍以上の演算性能を達成する AI エンジンを備え、AI 推論を飛躍的に高速化します。このシリーズは、動的ワークロードに対応したクラウドや、超高帯域ネットワークなど幅広いアプリケーションをサポートすると同時に、最先端の安全性とセキュリティ機能を提供します。ソフトウェアおよびハードウェア開発者だけでなく、AI およびデータ サイエнтиストも高い演算密度を活用してあらゆるアプリケーションの性能を高速化できます。

Versal プライム シリーズは、Versal プラットフォームの基盤となるミッドレンジ デバイスで、さまざまな市場で幅広く使用できます。具体的なアプリケーションとしては、100G ~ 200G ネットワーク機器、データセンターのネットワークおよびストレージ アクセラレーション、通信テスト装置、放送機器、航空宇宙/防衛機器などがあります。このシリーズはメインストリームの 58G トランシーバーおよび最適化された I/O および DDR コネクティビティを統合し、幅広いワークロードにおいて低レイテンシの高速化と性能を達成します。

Versal プレミアム シリーズ は、消費電力とフットプリントを最小限に抑えた適応型プラットフォームで、画期的なヘテロジニアス統合、超高性能演算、コネクティビティ、セキュリティを実現します。このシリーズは、ワイヤード通信、データセンター、テスト/計測などの広帯域幅で演算負荷の高いアプリケーションにおける要件を十分に満たすよう設計されています。Versal プレミアム シリーズ ACAP には、112G PAM4 トランシーバー、600G イーサネット用の統合ブロック、600G Interlaken、PCI Express® Gen5、および高速暗号化エンジンが含まれます。

Versal アーキテクチャのすべての 資料は、<https://japan.xilinx.com/versal> から参照できます。

設計プロセス別のコンテンツ ガイド

ザイリンクスの資料は、開発タスクに関連する内容を見つけやすいように、標準設計プロセスに基づいて構成されています。この資料では、次の設計プロセスについて説明します。

- システム/ソリューション プランニング: システム レベルのコンポーネント、パフォーマンス、I/O、およびデータ転送要件を特定します。ソリューションの PS、PL、および AI エンジン へのアプリケーション マップも含まれます。
 - [第 5 章: ソフトウェア開発フロー](#)
 - [第 6 章: ソフトウェア デザインのパラダイム](#)
 - [第 7 章: ブートおよびコンフィギュレーション](#)
 - [第 8 章: プラットフォーム ローダーおよびマネージャー](#)
 - [第 9 章: セキュリティ](#)
 - [第 10 章: Versal ACAP プラットフォーム管理](#)
 - [第 11 章: ターゲット開発プラットフォーム](#)
- エンベデッド ソフトウェア開発: ハードウェア プラットフォームからソフトウェア プラットフォームを作成し、エンベデッド CPU を使用してアプリケーションを開発します。XRT および Graph API も含まれます。
 - [第 3 章: 開発ツール](#)
 - [第 4 章: ソフトウェア スタック](#)
 - [第 6 章: ソフトウェア デザインのパラダイム](#)

このユーザー ガイドについて

主に Versal ACAP のシステム ソフトウェア開発環境に焦点を当てており、ソフトウェア開発者向けのスタートアップガイドとなります。このガイドは次の章で構成されています。

- 第 2 章: Versal ACAP のプログラミング: システム ソフトウェアの概要および Versal ACAP ハードウェアの関連事項について説明します。
- 第 3 章: 開発ツール: Versal デバイスのプログラムに使用するザイリンクス ツールおよびプログラミング フローについて説明します。
- 第 4 章: ソフトウェア スタック: Versal 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。
- 第 5 章: ソフトウェア開発フロー: Vitis™ IDE を使用したリアルタイム プロセッシング ユニット (RPU) およびアプリケーション プロセッシング ユニット (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 Vitis 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。
- 第 6 章: ソフトウェア デザインのパラダイム: 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たザイリンクス Versal デバイス アーキテクチャについて説明します。
- 第 7 章: ブートおよびコンフィギュレーション: Versal ACAP でサポートされるブート モード タイプを示し、セキュア ブート モードと非セキュア ブート モードにおけるブートおよびコンフィギュレーション プロセスについて説明します。

- 第 8 章: プラットフォーム ローダーおよびマネージャー: プラットフォーム管理コントローラー (PMC) のプラットフォーム プロセッシング ユニット (PPU) で動作するプラットフォーム ローダーおよびマネージャー (PLM) の役割について説明します。PLM は Versal ACAP のブートおよびコンフィギュレーションを実行した後、Versal デバイス上でのサービスを継続的に監視します。
- 第 9 章: セキュリティ: 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に利用できる Versal デバイスの機能について説明します。
- 第 10 章: Versal ACAP プラットフォーム管理: 電力、クロック、リセット、およびピンなどのリソースを最適に管理するプラットフォーム管理の役割について説明します。
- 第 11 章: ターゲット開発プラットフォーム: Versal ACAP で利用可能な各種ボードおよびキットについて説明します。
- 付録 A: ライブラリ: Versal ACAP で利用可能な各種ライブラリおよび API について説明します。

Versal ACAP のプログラミング

Versal™ ACAP デバイスには、5 種類のプログラマブル プロセッサがあります。各プロセッサは、システム全体のさまざまな要件を満たすために異なる演算能力を提供します。

- プロセッシング システム (PS) 内の Arm® Cortex™-A72 デュアルコア プロセッサ: 通常、制御プレーン アプリケーション、オペレーティング システム、通信インターフェイス、および下位または複雑な演算に使用されます。
- PS 内の Arm Cortex-R5F デュアルコア プロセッサ: 通常、安全性と確定性を重視するアプリケーションに使用されます。
- プログラマブル ロジック (PL) 内の MicroBlaze™ プロセッサ: (オプション) 通常、データの操作と移動、ベクトルベース以外の演算、および Versal ACAP の PS やその他コンポーネントとのインターフェイスに使用されます。
- PMC 内の RCU および PPU ユニット: 通常、プラットフォーム ロダーおよびマネージャー (PLM) コードの実行に使用されます。
- AI エンジン: 通常、ベクトル実装で演算負荷の高い機能に使用されます。

Versal ACAP のもう 1 つの主要ブロックは、プラットフォーム管理コントローラー (PMC) です。通常、PLM コードの実行に使用されます。このブロックは、ブート、コンフィギュレーション、Dynamic Function eXchange のほか、セキュリティなどのライフサイクル管理タスクを実行します。PMC によるブートおよび Dynamic Function eXchange の詳細は、[第 7 章: ブートおよびコンフィギュレーション](#) を参照してください。

この章では、次の項目について説明します。

- デュアルコア Cortex-A72 およびデュアルコア Cortex-R5F プロセッサを搭載した PS
- PL 内の MicroBlaze プロセッサ
- プロセッサで使用される Linux およびベアメタル ソフトウェア スタック
- ブートおよびコンフィギュレーション
- ソフトウェア エンジニアに関するその他の機能

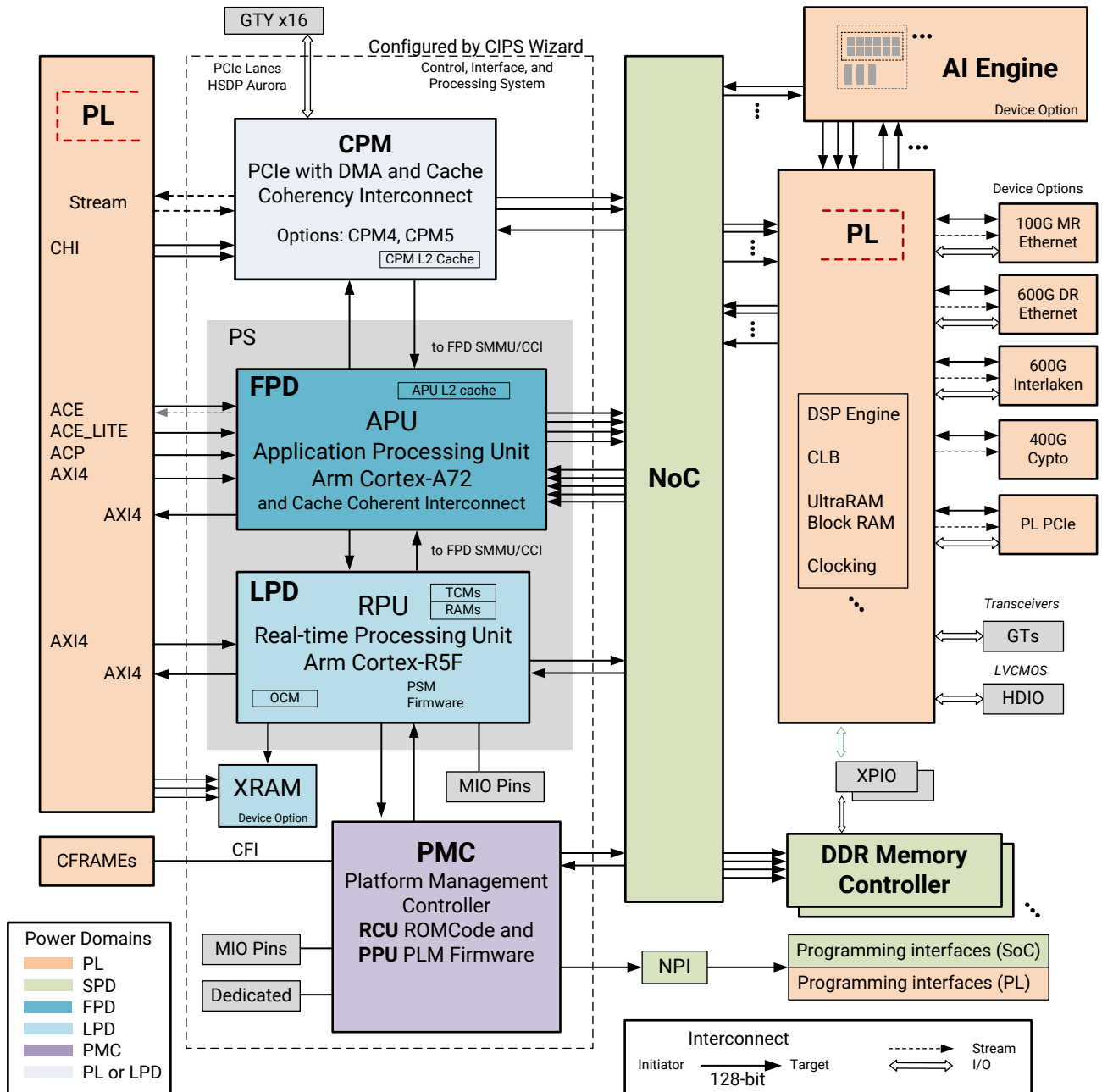
PMC、DDR のバス幅、DDR コントローラーの数、CCIX PCIe (CPM) インターコネクト、PCI Express® など、その他の機能の詳細は、『Versal アーキテクチャおよび製品データシート: 概要』(DS950: [英語版](#)、[日本語版](#)) を参照してください。『Versal ACAP テクニカル リファレンス マニュアル』([AM011](#)) には、PMC/PS を中心にした説明とハードウェア アーキテクチャのセクションがあり、CPM、DDR、AI エンジン、PL など多くの統合ハードウェアおよびペリフェラルに関する資料へのリンクも記載されています。

ハードウェアの概要

このセクションでは、Versal ACAP のハードウェア コンポーネントについて説明します。

注記: Versal ACAP ハードウェアの詳細は、『Versal ACAP テクニカル リファレンス マニュアル』([AM011](#)) を参照してください。

図 1: システム レベルのインターコネクト アーキテクチャ



X21692-111320

主要なハードウェア コンポーネント

概略図で大きく示されているハードコンポーネントは次のとおりです。

- AI エンジン: AI エンジンは、カスタム アクセラレーションや計算エンジンに使用されます。
- APU: アプリケーション プロセッシング ユニット (APU) は Cortex-A72 プロセッサ コア、L1/L2 キャッシュ、および関連機能で構成されます。Cortex-A72 コアおよびキャッシュは、Arm MPCore IP の一部です。

Versal ACAP は 1MB の L2 キャッシュを統合したデュアル コア Cortex-A72 プロセッサ システムを使用します。Cortex-A72 コアは Armv8 64 ビット アーキテクチャを実装します。Cortex-A72 MPCore は汎用割り込みコントローラー (GIC) を内蔵していないため、外部 GIC IP を使用します。詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 (AM011) の「APU プロセッサの機能」を参照してください。

- AXI インターコネク: AXI (Advanced eXtensible Interface) インターコネクは、1 つ以上のメモリマップ方式 AXI マスター デバイスと、1 つ以上のメモリマップ方式のスレーブ デバイスを接続します。AXI インターフェイスは、Arm の AMBA® AXI 仕様バージョン 4 に準拠しています。この仕様には AXI4-Lite 制御レジスタ インターフェイスのサブセットも含まれます。
- CPM: CCIX (Cache Coherent Interconnect for Accelerators) インターコネクおよび PCIe® (CPM) モジュールが、プロセッシング システムの主要な PCIe インターフェイスとなります。CPM には 2 つの PCIe 統合ブロックがあり、最大 Gen4 x16 をサポートします。両方の PCIe 統合ブロックをエンドポイントとして構成できます。また、各統合ブロックをダイレクト メモリ アクセス (DMA) ロジックを含むルート ポートとして構成できます。CPM には CCIX の機能もあり、PL アクセラレータを CCIX 準拠アクセラレータとして動作させることができます。
- PL: プログラマブル ロジック (PL) は適応型エンジンとインテリジェント エンジンを含むスケーラブルな構造をしており、アクセラレータやプロセッサなど、複雑な機能のほとんどすべてを構築できます。コンフィギュレーションには、Vivado® ツールを使用します。PL デザインに含めるコンポーネントは設計者が決定します。たとえば、MicroBlaze プロセッサは IP コアであるため、必要に応じて MicroBlaze プロセッサをデザインに追加できます。詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 (AM011) の「プログラマブル ロジック」を参照してください。
- PMC: プラットフォーム管理コントローラー (PMC) は、デバイスのリセット シーケンス、初期化、ブート、コンフィギュレーション、セキュリティ、パワー マネージメント、Dynamic Function eXchange (DFX)、健全性モニター、エラー管理などのデバイス管理制御機能を実行します。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。詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 (AM011) の「プラットフォーム管理コントローラー」を参照してください。
- NoC インターコネク: NoC が主なインターコネクであり、垂直コンポーネント (VNoC) と水平コンポーネント (HNoC) が含まれています。
 - HNoC は、HSR (Horizontal Super Row/Region) に統合されています。HSR には XPIO、ハード DDR メモリ コントローラー、PLL、システム モニター サテライト、HBM、ME などのブロックが含まれます。
 - VNoC には、グローバル クロック カラムとシステム モニター サテライトが統合されています。SSI テクノロジでは、VNoC は SLR (Super Logic Region) 境界を越えて接続されます。Thin-HNoC にはこのためのマイクロバンプとバッファがあります。コンフィギュレーション データは、SSI テクノロジのマスターとスレーブ間で NoC を介して移動します。
- RPU: リアルタイム プロセッシング ユニット (RPU) は、Armv7-R アーキテクチャに基づくデュアル コア Cortex-R5F プロセッサで、2 つの独立したコアとして、またはロックステップ構成での動作が可能です。詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 (AM011) の「プラットフォーム管理」を参照してください。

その他のハードウェア コンポーネント

- ペリフェラル コントローラー: PS モジュール内の I/O ユニット (IOP) には、低電力ドメイン (LPD) の低速ペリフェラルとフル電力ドメイン (FPD) の高速ペリフェラルが含まれます。一部の IOP は、PMC 電源ドメイン (PPD) のペリフェラルにあります。

詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 (AM011) の「I/O ペリフェラル」を参照してください。

- インターコネクとバス: Versal ACAP には、次に示すインターコネクとバスもあります。
 - NPI: NoC プログラミング インターフェイス。NoC およびその他の接続されたユニットへの 32 ビットのプログラミング インターフェイスです。

詳細は、『Versal ACAP Programmable Network on Chip および Integrated Memory Controller LogiCORE IP 製品ガイド』(PG313)を参照してください。

- APB: APB (Advanced Peripheral Bus) は、32 ビットの単一ワード読み出し/書き込みバス インターフェイスです。このバスは、ファンクションユニット (サブシステム ユニット) の制御レジスタへのアクセスに使用されます。この制御レジスタを使用して、ファンクション ユニットのプログラムします。次の 4 つのエリアでは、APB スイッチがメイン スイッチとして使用されます。
 - PMC
 - LPD
 - FPD
 - CPM
- CFI: コンフィギュレーション フレーム インターフェイス (CFI) は、ブート イメージに含まれるコンフィギュレーション情報を PMC から Versal デバイス内のデスティネーションへ転送します。CFI は PL に対する専用の広帯域 128 ビット バスで、これを使用してコンフィギュレーションとリードバックを実行します。詳細は、『Versal ACAP テクニカル リファレンス マニュアル』(AM011)の「CFI」を参照してください。
- システム ウォッチドッグ タイマー: システム ウォッチドッグ タイマー (SWDT) は、さまざまな異常動作を検出して回復するために使用します。SWDT を使用すると、ソフトウェアがデッドロック状態になってシステムがロックアップするのを防ぐことができます。詳細は、『Versal ACAP テクニカル リファレンス マニュアル』(AM011)の「システム ウォッチドッグ タイマー」を参照してください。
- クロック: Versal ACAP には、次のクロックがあります。
 - PMC および PS クロック
 - CPM クロック
 - NoC、AI エンジン、および DDR メモリ コントローラー クロック
 - PL クロック

詳細は、『Versal ACAP テクニカル リファレンス マニュアル』(AM011)の「クロック」の章を参照してください。

- メモリ: Versal デバイスには、次のメモリがあります。
 - DDR メモリ: 最大 12GB の RAM がサポートされます。この DDR メモリはデバイス外部にあります。
 - PS 内のオンチップ メモリ (OCM): このメモリは容量が 256KB で、RPU からアクセスできます。
 - RPU 内の密統合メモリ (TCM): このメモリは 256KB で、主に RPU が使用しますが、APU からアクセスできます。
 - バックアップ バッテリー付き RAM (BBRAM): このメモリは、AES (Advanced Encryption Standard) 256 ビット キーを格納します。
 - アクセラレータ RAM (XRAM): 一部の Versal デバイスに存在する 4MB のメモリです。
 - eFUSE: 2,048 ビットのユーザー メモリで、複数のキー、およびセキュリティ コンフィギュレーション設定を格納します。
- リセット: Versal ACAP にはいくつかのリセット階層があり、それぞれのリセットの範囲には重複があります。最上位のリセット階層は、電源ドメインとほぼ一致しています。次の階層は電源アイランドをリセットし、一番下の階層は個々のファンクションユニットをリセットします。一部のファンクションユニットには、ブロックの一部のみを範囲とするローカル リセットがあります。リセット階層は次のとおりです。
 - サブシステム リセット (電源ドメイン)
 - 電源アイランド リセット

- ファンクション ユニット (ブロック) リセット
- ブロックの一部リセット (場合による)

詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 (AM011) の「メモリ仮想化」を参照してください。

- 仮想化: Versal デバイスには、仮想化に関する次の 3 つのハードウェア コンポーネントがあります。
 - CPU 仮想化
 - メモリ仮想化
 - 割り込み仮想化

詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 (AM011) の「メモリ仮想化」を参照してください。

- セキュリティと安全: Versal デバイスには、次に示すセキュリティ管理および安全機能があります。
 - セキュアなキーの格納および管理
 - タンパーの監視と応答
 - ユーザー アクセス可能なザイリンクス ハードウェア暗号アクセラレータ
 - ザイリンクス メモリ保護ユニット (XMPU) とザイリンクス ペリフェラル保護ユニット (XPPU) は、ハードウェアによる分離機能を提供します。
 - TrustZone

詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 (AM011) の「プラットフォーム管理コントローラー」および第 9 章: セキュリティ を参照してください。XMPU と XPPU の詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 (AM011) の「メモリ保護」を参照してください。

プロセッシング システム

プロセッシング システム (PS) は、次のコンポーネントで構成されています。

- デュアル コア Arm Cortex-A72 プロセッサ。エラー訂正符号 (ECC) 付き 1MB L2 キャッシュ内蔵
- デュアル 32 ビット Cortex-R5F プロセッサ コア。Arm® v7-R アーキテクチャに基づき、エラー訂正符号 (ECC) 対応 128KB TCM を備えた RPU と ECC 対応 128KB TCM を備えた単一のロックステップ R5 の両方をサポート。
- エラー訂正符号 (ECC) 付き 256KB オンチップ メモリ
- TMC、STM、ATM、および APM を備えた Arm CoreSight™ デバッグおよびトレース (DAP)
- システム メモリ管理ユニット (SMMU)
- CCI
- 1 つの USB (ユニバーサル シリアル バス) 2.0
- CPM に PCIe RP/EP (デバイスによる)
- TSN をサポートした 2 つのギガビット イーサネット MAC
- 1 つの低電力ドメイン DMA (LPD-DMA)
- 高性能 I/O
- 2 つの CAN-FD (コントローラー エリア ネットワーク フレキシブル データ レート)、2 つの SPI (シリアル ペリフェラル インターフェイス)、2 つの I2C、および 2 つの UART コントローラー
- PS 管理コントローラー (PSM)

注記: PS は外部の共有 DDR にアクセスできます。

アプリケーション プロセッシング ユニット

アプリケーション プロセッシング ユニット (APU) は Arm Cortex-A72 プロセッサ、L1/L2 キャッシュ、および関連機能で構成されます。Cortex-A72 コアとキャッシュは Arm プロセッサ MPCore IP の一部で、L1 および L2 キャッシュを統合しています。

Versal デバイスは 1MB の L2 キャッシュを統合したデュアル コア Cortex-A72 プロセッサを使用します。

Cortex-A72 コアは Armv8 64 ビット アーキテクチャを実装します。Cortex-A72 MPCore プロセッサは汎用割り込みコントローラー (GIC) を内蔵していないため、外部 GIC IP を使用します。

APU には次の機能があります。

- エラー訂正符号 (ECC) 付き 1MB L2 キャッシュを内蔵したデュアル コア Cortex-A72 コア クラス。L1 キャッシュは容量 48KB の I キャッシュと 32KB の D キャッシュで構成されます。L1 キャッシュはエラー訂正符号 (ECC) を含みます。
- GIC-500 割り込みコントローラー
- コアごとのパワー ゲーティングをサポート
- TrustZone をサポート

リアルタイム プロセッシング ユニット

Versal ACAP の APU は安全レベルと性能がいずれも改善されています。ただし、より高いレベルの安全性 (ASIL-C/SIL3 など)、信頼性、および確実性が要求されるリアルタイム アプリケーションには、リアルタイム プロセッシング ユニット (RPU) をロックステップ プロセッサ サブシステムと組み合わせて使用します。

RPU のアーキテクチャ仕様は、RPU コア、TCM、およびオンチップ メモリで構成されます。次に、RPU の主な機能を示します。

- デュアル 32 ビット Cortex-R5F コア。Arm v7-R アーキテクチャに基づき、ロックステップまたはスプリット モードのオプションをサポート
- スプリット モードでは各 Cortex-R5F プロセッサに 128KB TCM
- ロックステップ モードでは TCM を 256KB に結合可能
- RPU と APU からアクセス可能なエラー訂正符号 (ECC) 付き 256KB オンチップ メモリ
- エラー訂正符号 (ECC) またはパリティ付き 32KB L1 命令キャッシュとエラー訂正符号 (ECC) 付き 32KB L1 データキャッシュ
- GIC アーキテクチャをサポートする汎用割り込みコントローラー (GIC)
- ロックステップごとのパワー ゲーティングをサポート
- TCM および OCM のパワー ゲーティング
- TrustZone 対応

密統合メモリ (TCM) インターフェイス

コアごとの TCM は、次のモードに設定できます。

- スプリット (高性能) モード: スプリット モードで動作中、RPU の各コアはそれぞれ 128KB の TCM にしかアクセスできません。

- ロックステップ (セーフティ) モード: ロックステップ モードで動作中、RPU は 256KB の TCM 全体にアクセスできます。

RPU のコンフィギュレーション オプション

次の表と図に、RPU の 4 つのコンフィギュレーション オプションを示します。

表 1: RPU のコンフィギュレーション オプション

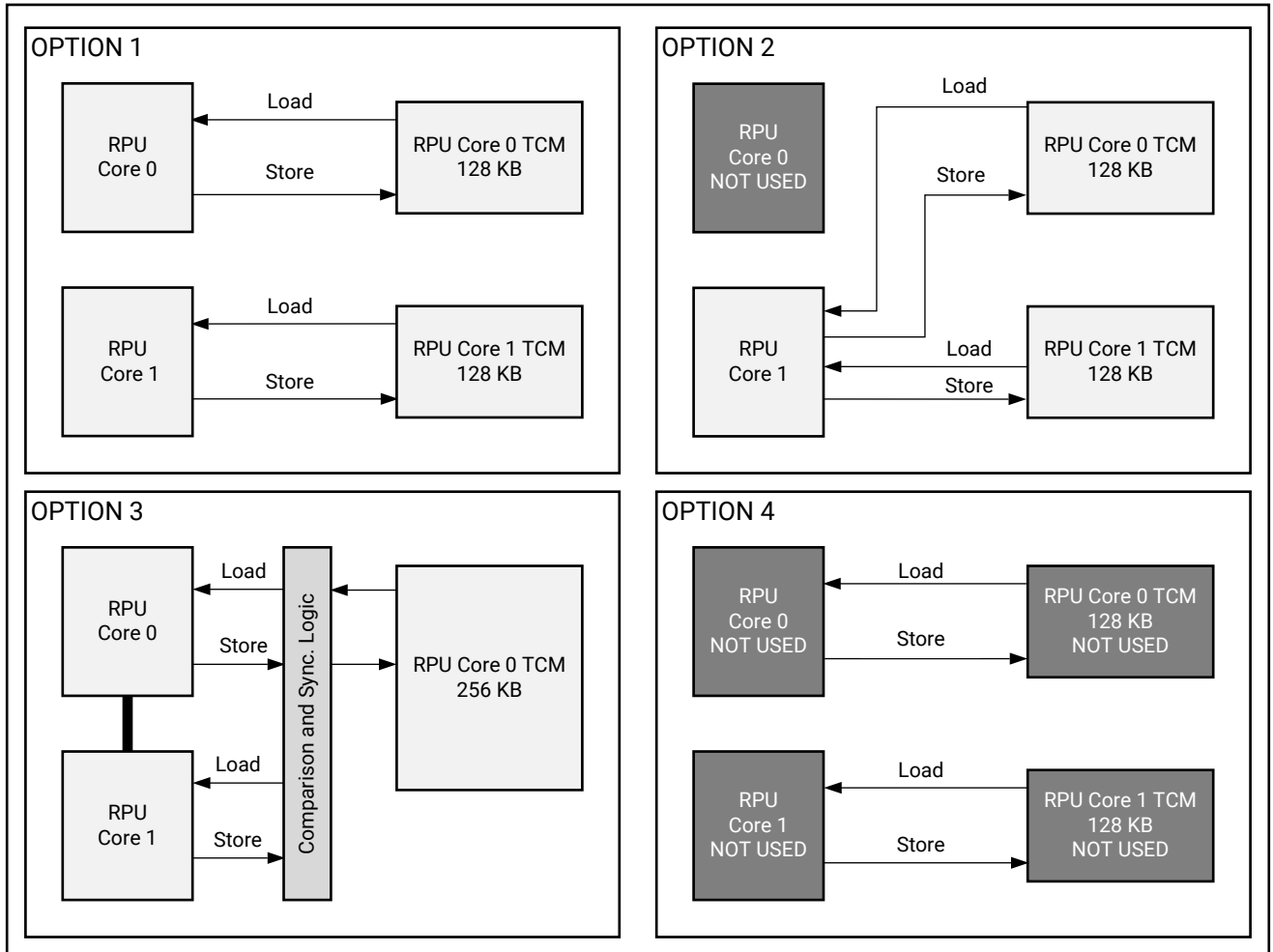
| コンフィギュレーション オプション | 説明 | 動作するコア |
|--------------------------------|--|---------------------|
| オプション 1: スプリット モード | 高性能モード。 2 つのリアルタイム コアが独立して動作し、それぞれが別の TCM を使用します。 | RPU コア 0 と RPU コア 1 |
| オプション 2: スプリット モードで、1 つのコアのみ使用 | 高性能モード。 RPU コア 0 をリセット状態にし、RPU コア 1 が 256KB TCM をすべて使用して単独で動作します。 | RPU コア 1 のみ |
| オプション 3: ロックステップ モード | セーフティ モード。 注記: 通常このモードは、安全性重視の確定的アプリケーションに使用されます。 2 つのコアが内蔵のコンパレータ ロジックを使用して同時に動作します。RPU コア 0 がマスター、RPU コア 1 がチェッカーとなります。 TCM は結合し、RPU コア 0 の方に大きな TCM が割り当てられます。 2 つのコアは同じコードを実行します。これらのコアの入力と出力が比較され、一致しない場合にコンパレータがエラーを検出します。 2 つのコアが使用されますが、処理量は 1 つ分です。 | RPU コア 0 のみ |
| オプション 4: なし | RPU は使用されません。 | なし |

注記:

1. RPU コア 0 の TCM は、スプリット モードのときに Cortex-R5F コア 0 に関連付けられる密結合メモリです。RPU コア 1 の TCM は、スプリット モードのときに RPU_0 コアに関連付けられる密結合メモリです。
2. RPU コア 1 の TCM は、RPU コア 0 の TCM よりもコアから少し離れているため、コアが RPU コア 1 の TCM へアクセスする際には、RPU コア 0 の TCM へアクセスするときよりレイテンシがわずかに長くなる可能性があります。

RPU コアは、システム ウォッチドッグ タイマー (SWDT) を使用してタイマー レジスタへの定期的な書き込みを実行することで、機能の監視とパフォーマンスのチェックを実行します。

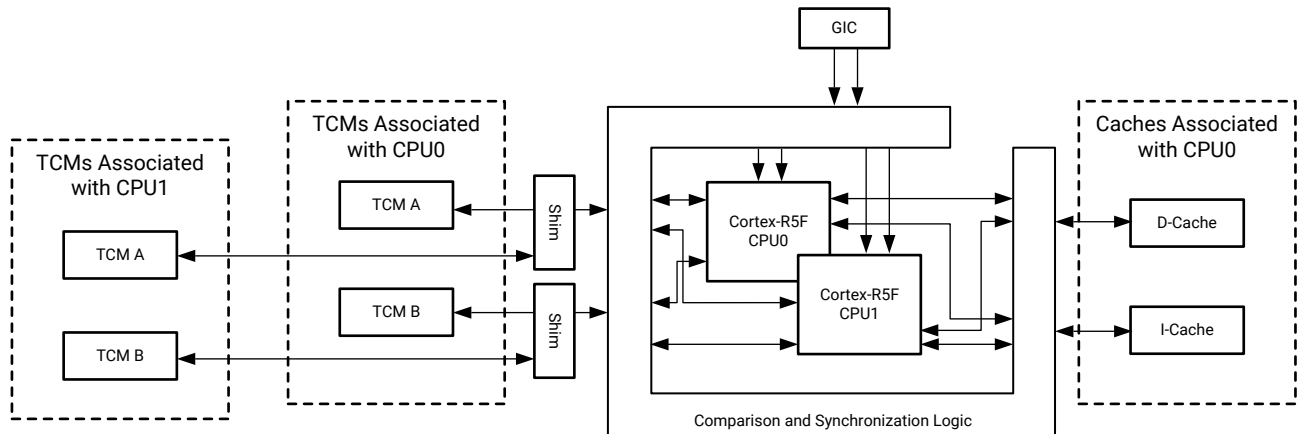
図 2: RPU のコンフィギュレーション オプション



X22497-041719

次の図は、ロックステップ モードの RPU のリソース共有を示しています。

図 3: RPU Cortex-R5F プロセッサ (ロックステップ モード)



X15295-050919

プログラマブル ロジック

Versal ACAP のプログラマブル ロジック (PL) は、Vivado IP インテグレーターを使用してコンフィギュレーションできます。PL は柔軟性に優れており、構築ブロックや統合コンポーネントを使用してコンフィギュレーションすることで、カスタム デザインを作成できます。

PL は、統合されインスタンス化されたハードウェア アクセラレータ、コントローラー、メモリ、やその他のファンクション ユニットの含む複雑な構造を持ちます。

- 統合ファンクション ユニットには、マルチギガビット イーサネット MAC などがあります。
- 構築ブロックは、ファンクション ユニットをインスタンス化し、統合されたユニットをインターコネクトや I/O 構造に接続するために使用します。構築ブロックには、DSP、ブロック RAM、UltraRAM、およびクロック構造が含まれます。
- PL の統合された構築ブロックを使用して、次のようなインスタンス化されたファンクション ユニットが構築されます。
 - インターコネクト: AXI、NoC インターコネクト
 - プラットフォーム管理コンポーネント: PS のコンフィギュレーションとリセット
 - デジタル ファンクション ユニット: 加算器、カウンタ、浮動小数点ユニット (FPU)、およびビデオ
 - 無線周波数 (RF) 指向のファンクション ユニット: RF を使用するファンクション ユニット、LTE (Long Term Evolution)、および無線

プログラマブル ロジック内の MicroBlaze プロセッサ

オプションで、MicroBlaze プロセッサを PL デザインに追加できます。この MicroBlaze プロセッサは、Linux、ベアメタル アプリケーション、FreeRTOS、またはその他のカスタム ソフトウェアなど、さまざまなソフトウェアを実行できます。

開発ツール

この章では、Versal™ ACAP 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、Versal ACAP 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。

Versal 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 ツール
- シミュレータ (Quick Emulator (QEMU) など)
- 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)

注記: サードパーティのツール ソリューションを使用した場合の統合レベルおよび Versal ACAP の直接サポート レベルはさまざまに異なります。

後続のセクションでは、ザイリンクス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 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 GUI を提供します。

これらのツールは、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開発環境です。

Vivado Design Suite 内のすべてのコマンドおよびコマンド オプションは、ネイティブ ツール コマンド言語 (Tcl) を使用しているため、Vivado IDE と Vivado Design Suite Tcl シェルの両方で実行できます。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。

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。

- グラフィカル ユーザー インターフェイスを備えた IP インテグレーター内で CIPS (Control, Interfaces, And Processing System) IP をコンフィギュレーションすることにより、IP インテグレーター ブロック デザイン内で CIPS の作成および変更が可能。
- 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 (Synopsys Design Constraints) ベースの XDC (ザイリンクス 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
- Vivado Tcl Store - Vivado ツール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](#) からダウンロード可能です。

Vitis ソフトウェア プラットフォーム

Versal ACAP デザインは、Vitis™ ツール、ライブラリ、IP を使用して完成します。Vitis IDE では、AI エンジン カーネルとグラフ、PL、高位合成 (HLS) IP、RTL IP、PS アプリケーションなど、Versal ACAP AI エンジン アプリケーションのさまざまなエレメントをプログラム、実行、デバッグできます。Vitis IDE ツール フローの詳細は、『Versal ACAP デザイン ガイド』(UG1273: [英語版](#)、[日本語版](#)) の「第 4 章: デザイン フロー」を参照してください。

Vitis ソフトウェア プラットフォームは、次のフローによるソフトウェア開発をサポートしています。

- [アクセラレーション フロー](#)
- [エンベデッド フロー](#)

アクセラレーション フロー

アクセラレーションが必要な場合、Vitis 開発環境で OpenCL™ またはオープンソースのザイリンクス ランタイム (XRT) ネイティブ API を使用してソフトウェア アプリケーションをビルドし、ザイリンクス Alveo™ データセンター アクセラレータ カードなどのアクセラレータ カードでハードウェア カーネルを実行させることができます。Vitis コア開発キットは、Versal ACAP などの Linux エンベデッド プロセッサ プラットフォーム上でのソフトウェア アプリケーションの実行もサポートします。

エンベデッド プロセッサ プラットフォームの場合も、Vitis 実行モデルは OpenCL™ API と Linux ベースの XRT を使用してハードウェア カーネルをスケジュールし、データ移動を制御します。

Vitis ツールは、Alveo PCIe ベース カード、ZCU102 ベース、ZCU102 ベース + Dynamic Function eXchange (DFX)、ZCU104 ベース、ZC702 ベース、ZC706 ベース エンベデッド プロセッサ プラットフォーム、および Versal ACAP VCK190 ベース、VMK180 ベース ボードをサポートしています。

これらのすぐに使用可能なプラットフォームに加え、Vitis ツールはカスタム ツールもサポートします。

Vitis 開発環境はデータセンター プラットフォームとエンベデッド プロセッサ プラットフォームの両方のアプリケーション開発をサポートしており、データセンター アプリケーションをエンベデッド プラットフォームへ移行することもできます。Vitis ツールには、すべてのプラットフォームのハードウェア カーネル用の `v++` コンパイラ、x86 ホスト上で動作するアプリケーションをコンパイルするための `g++` コンパイラ、およびザイリンクス デバイスのエンベデッド プロセッサ上で動作するアプリケーションをクロス コンパイルするための Arm® コンパイラが含まれます。

注記: `v++` コンパイラには、コンパイル (Vitis HLS に対する `v++ -c`)、システム リンキング (`v++ -l`)、およびパッケージング (`-p`) の 2 つのオプションがあります。

詳細は、『Vitis 統合ソフトウェア プラットフォームの資料』 (UG1416) の [Vitis アプリケーション アクセラレーション開発フロー](#) を参照してください。

エンベデッド フロー

オープンソースの Eclipse プラットフォームをベースにした Vitis 開発環境は、ザイリンクスのエンベデッド プロセッサ向けのソフトウェア アプリケーションを構築するための包括的な環境を提供します。この環境には、GNU ベースのコンパイラ ツールチェーン、C/C++ 開発ツールキット (CDT)、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 用ドライバーが含まれます。また、C/C++ ベアメタルおよび Linux アプリケーション開発やデバッグに有効な IDE もあります。

この開発環境では、Arm Cortex™-A72 および Cortex™-R5F プロセッサ、さらにはザイリンクスの MicroBlaze™ プロセッサ用に統合されたザイリンクス 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。この環境では、次の方法でアプリケーションを作成できます。

- MicroBlaze プロセッサ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
- ユーザーによる PLM のカスタマイズ (主に Versal ACAP のブートに使用)
- Vitis には次のサンプル ライブラリが付属しており、すぐにソースをロードしてビルドできます。
 - 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)
 - 暗号ハードウェア エンジンへのアクセス
 - eFUSE および BBRAM のプログラミング
 - PLM

Vivado IDE でハードウェア デザインを作成したら、Vivado の Project Navigator からブロック デザイン、ハードウェア デザインおよびビットストリーム ファイルを Vitis ツールのエクスポート ディレクトリに直接エクスポートできます。

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。Vitis ツールのプロセスは、次のファイルを Vitis ツールのディレクトリにエクスポートします。

- `.xpr`: Vivado プロジェクト ファイル
- `psv_init.tcl`、`psv_init_gpl.c`、`psv_init.c`、`psv_init.h`: PLM 作成時に必要な情報を含む
- `psv_init.html`: Versal ACAP レジスタ サマリ ビューアー
- `.xsa`: デザインのザイリンクス [サポート アーカイブ](#)。

Vitis 環境では次のファイルも生成できます。

- 次のプロセッサのセキュアおよび非セキュア ブートに使用するプログラマブル デバイス イメージ (PDI)。
 - Arm Cortex-A72
 - Cortex-R5F
 - MicroBlaze

Vitis 環境は、Linux アプリケーションの開発とデバッグをサポートしています。

詳細は、『Vitis 統合ソフトウェア プラットフォームの資料』 (UG1416) の [Vitis エンベデッド ソフトウェア開発フローの資料](#) を参照してください。

Vitis ツール

Vitis 開発環境には、ザイリンクスのエンベデッド ソフトウェア開発で使用する次のツールが用意されています。

- ザイリンクス システム デバッガー (XSDB): ザイリンクス `hw_server` へのコマンド ライン インターフェイスを提供します。`hw_server` は、デバイス/プロセッサ レベルのデバッグ機能を XSDB などのフロントエンド ツールや Vitis IDE に提供するバックエンド ツールです。また、Vitis 開発環境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。
- フラッシュ プログラマ: ソフトウェア アプリケーション イメージを QSPI、OSPI、または eMMC などの外部フラッシュ 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: ペアメタル/FreeRTOS アプリケーションの ELF セクションをハードウェア メモリ空間にマップします。
- GNU コンパイラ ツールスイート: ザイリンクスの全プロセッサでコンパイルに使用される GNU Compiler Collection (GCC)、AS、LD、binutils などのツールを含みます。
- Bootgen: PDI と呼ばれるブート イメージの生成に使用します。
- パフォーマンス解析ツール: GPROF、TCF プロファイラー、および OProfile を含みます。
- デバッグ/ダウンロード ツール: GNU デバッガー (GDB)、XSDB、フラッシュ ライターを含みます。
- シミュレータ: QEMU

- ザイリンクス ソフトウェア コマンド ライン ツール (XSCT): ザイリンクス ソフトウェア コマンド ライン ツール (XSCT) は、Vitis IDE への対話型コマンド ライン インターフェイスで、スクリプトの実行が可能です。ほかのザイリンクス ツール同様、XSCT のスクリプト言語はツール コマンド言語 (Tel) に基づいています。これらのツールは細部を抽象化し、一般的な機能をわかりやすいウィザード形式にまとめてあるため、初心者でも簡単に使えます。

ただし、ツールの機能を柔軟に拡張するには、スクリプトへの対応も欠かせません。この機能は、夜間に行うリグレッション テストの作成、または開発者がよく使用するコマンド セットの実行において特に役立ちます。

XSCT のコマンドは対話形式で実行することも、スクリプトを作成して自動化することもできます。XSCT は次のアクションをサポートしています。

- ハードウェア、ドメイン、プラットフォーム プロジェクト、システム プロジェクト、およびアプリケーション プロジェクトを作成する
- リポジトリを管理する
- ツールチェーンの設定をカスタマイズする
- ドメイン/BSP およびアプリケーションを設定およびビルドする
- アプリケーションをターゲット ハードウェアにダウンロードして実行する
- Bootgen および program_flash ツールを実行してブート イメージを作成し、フラッシュ メモリに書き込む

このリファレンス コンテンツは、ザイリンクス プロセッサをターゲットにしたソフトウェアの開発とデバッグのためのスクリプトを作成するのに必要な情報を提供することを目的としています。

詳細は、『Vitis 統合ソフトウェア プラットフォームの資料』(UG1416) のエンベデッド ソフトウェア開発フローの[ザイリンクス ソフトウェア コマンド ライン ツール](#)を参照してください。

Vitis ツールでは、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スペクティブを提供します。C/C++ 開発で利用可能なパースペクティブには次のものがあります。

- デザイン パースペクティブ 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、Vitis 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。
- デバッグ ビュー: 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。デバッグ パースペクティブからオープンソース アプリケーションをカスタマイズして、Vitis 環境に統合できます。
- リモート システム エクスプローラー: Versal デバイス上で動作する Linux など、各種リモート システムに接続して動作します。

PetaLinux ツール

PetaLinux ツールには、ザイリンクスのプロセッシング システム向けに組み込み Linux ソリューションをカスタマイズ、ビルド、および運用するために必要なものがすべて揃っています。SoC デバイスの設計生産性向上を目指して最適化されたこのソリューションは、ザイリンクスのハードウェア設計ツールと協調動作して Versal デバイス向けのオープン ソース Linux システムの開発を支援します。

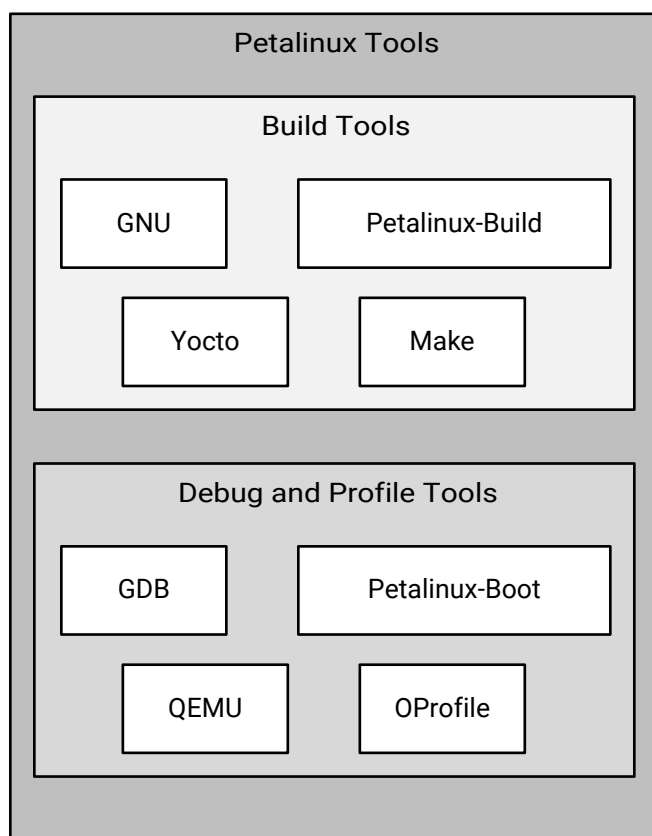
PetaLinux には次が含まれます。

- 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。

- GNU デバッガー (GDB)、petalinux-boot、およびプロファイリング用の Oprofile などのデバッグ ツール。次に、サポートされる PetaLinux ツールチェーンの一部を示します。
 - GNU gcc/g++ ツールチェーン: ザイリンクスの 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-package: ブート イメージの作成に使用。
 - petalinux-boot: Linux のブートに使用。
 - QEMU: Versal 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。

次の図に、PetaLinux ツール フローを示します。

図 4: PetaLinux ツール ベースのフロー



X23441-112719

詳細は、次を参照してください。

- 『PetaLinux ツール資料: リファレンス ガイド』 (UG1144: [英語版](#)、[日本語版](#))
- <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842250/PetaLinux> (ザイリンクス Wiki)。

デバイス ツリー ジェネレーター

デバイス ツリー (DT) は、ハードウェア デバイスを記述するプロパティを持つノードを格納したデータ構造体です。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。

また、それぞれに異なる構成を使用するペリフェラル ロジックをさまざまに組み合わせることも可能です。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。

- `pl.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。
- `versal.dtsi`: PS 固有の情報および CPU の情報が含まれます。
- `versal-clk.dts`: PS ペリフェラルのクロックおよび電源ドメインの情報が含まれます。詳細は、<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842279/Build+Device+Tree+Blob> (ザイリンクス Wiki) を参照してください。

オープンソース

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 Linaro GCC ツールチェーンが採用されています。Vitis 環境には Linux ホスト用 GNU ツールが含まれます。このセクションでは、Versal ACAP 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。

ザイリンクス Arm GNU ツール

次に、APU、RPU、およびエンベデッド MicroBlaze プロセッサ用ソフトウェアのコンパイルに利用可能なザイリンクス Arm GNU ツールを示します。

- `arm-linux-gnueabi-hf-gcc`: Armv8 C コードをハード浮動小数点命令を使用した 32 ビット Linux アプリケーションにコンパイルします。
- `arm-linux-gnueabi-hf-g++`: Armv8 C++ コードを 32 ビット Linux アプリケーションにコンパイルします。
- `aarch64-linux-gnu-gcc`: Armv8 C コードを 64 ビット Linux アプリケーションにコンパイルします。
- `aarch64-linux-gnu-g++`: Armv8 C++ コードを 64 ビット Linux アプリケーションにコンパイルします。
- `aarch64-none-elf-gcc`: Armv8 C コードを 64 ビット RTOS およびベアメタル アプリケーションにコンパイルします。
- `aarch64-none-elf-g++`: Armv8 C++ コードを 64 ビット RTOS およびベアメタル アプリケーションにコンパイルします。
- `armv7-none-eabi-gcc`: Armv7 C コードを 32 ビット RTOS およびベアメタル アプリケーションにコンパイルします。

- `armr5-none-eabi-g++`: Armv7 C++ コードを 32 ビット RTOS およびベアメタル アプリケーションにコンパイルします。
- `microblaze-xilinx-elf-gcc`: MicroBlaze™ C コードをコンパイルします。
- `microblaze-xilinx-elf-g++`: MicroBlaze C++ コードをコンパイルします。

注記: いずれの GNU コンパイラにも、関連するアセンブラーやリンカーなどが付属します。

Yocto ツールを使用した Linux ソフトウェア開発

ザイリンクスは、ユーザーが自社の Yocto ビルド システムを使用して Versal デバイス用に Linux を設定、ビルド、および運用できるように、オープンソースの meta-xilinx Yocto/OpenEmbedded レシピを <https://github.com/Xilinx> で提供しています。

また、meta-xilinx レイヤーには、ZC702、ZCU102、Ultra96、Zedboard、VCK190、VMK180 などのザイリンクス リファレンス ボードおよびベンダー ボード向けの BSP も多数含まれます。

meta-xilinx レイヤーは Yocto/OpenEmbedded と互換性があり、各種コンポーネントのレシピを追加しています。詳細は、<https://git.yoctoproject.org/cgi/cgit.cgi/meta-xilinx/> を参照してください。

Arm Cortex-A72 プロセッサ上で動作する 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 ディストリビューションを示します。

表 2: Yocto ディストリビューション

| ディストリビューション タイプ | 名前 | 説明 |
|---------------------------|----------|--|
| Yocto ビルド システム | 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 パッケージ | 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 開発環境は、<https://github.com/Xilinx> で提供される Yocto レシピを通じ、Versal ACAP 用 Linux ソフトウェアの開発をサポートします。Yocto Project のコンポーネントを利用すると、オープンソース ベースの Linux ソフトウェア スタックを設計、開発、構築できます。

次の図に、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、最新のザイリックス 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。

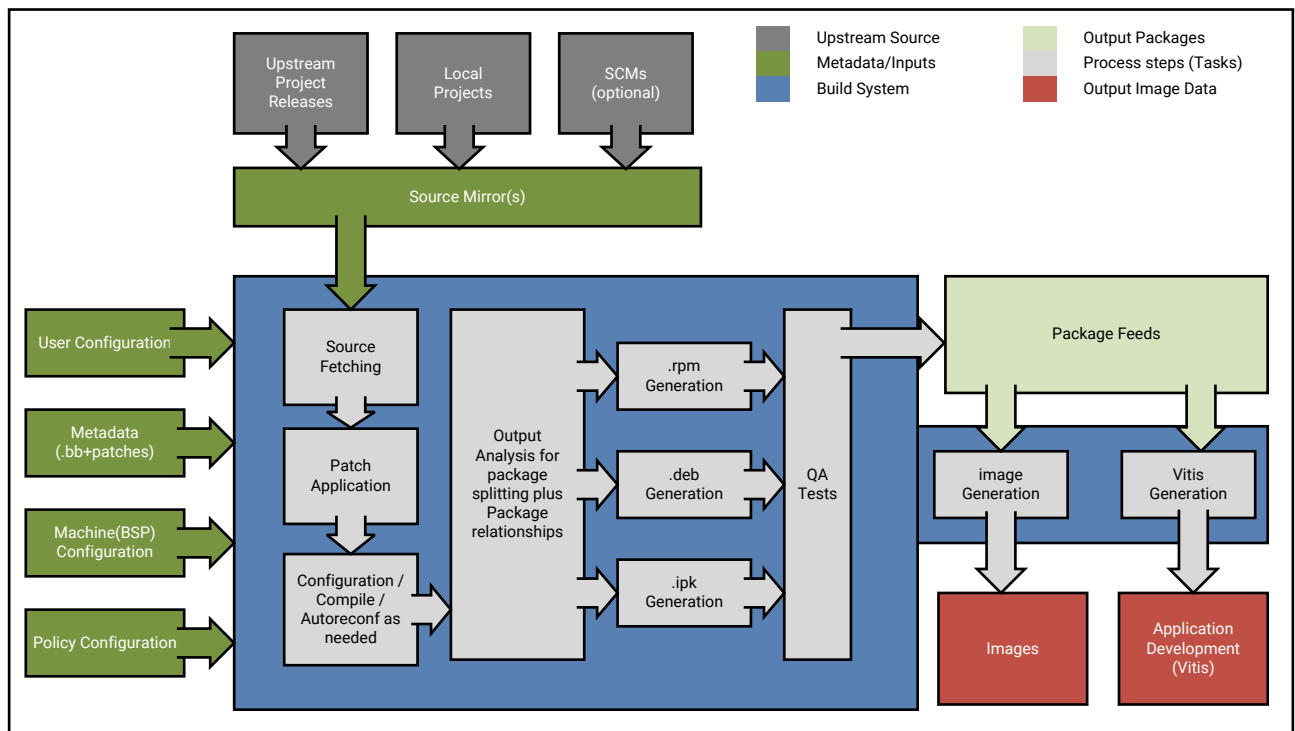
また、BSP を利用してビルドおよびハードウェア構成を変更することもできます。

Yocto は高機能なコンパイラと高品質な解析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、Vitis ツール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 Vitis 環境を起動します。

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、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。
- 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。

図 5: Yocto Project 開発環境



X14841-062420

Yocto ツールおよび Yocto Project 開発環境は、<https://www.yoctoproject.org/software-overview/downloads/> からダウンロードできます。

ザイリンクスがサポートする Yocto 機能の詳細は、『PetaLinux ツール資料: リファレンス ガイド』(UG1144) の「Yocto の機能」および <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841883/Yocto> (ザイリンクス Wiki) を参照してください。

QEMU

QEMU は高速動作する命令精度の機能エミュレーターで、Linux および Windows ホストのいずれにもインストールできます。ザイリンクス QEMU はザイリンクス シリコンと同じペリフェラル モデルを多数備えています。ホストからの実際のデータをエミュレーション (イーサネット、UART、ストレージ、CAN-FD など) に挿入して、アプリケーションのステミュレーションに使用できます。ザイリンクスは、Versal ACAP、Zynq UltraScale+ MPSoC、Zynq-7000、および MicroBlaze プロセッサなど、何世代にもわたり開発プラットフォームとして QEMU を提供してきました。

ザイリンクス QEMU は、すべてのザイリンクス リリースにおいて Vitis、PetaLinux、および Yocto に統合されています。最新の開発ブランチについては、ソースからザイリンクス QEMU をダウンロードして、ビルドとインストールを実行できます。

このエミュレーション プラットフォームを使用するにあたり、次に関する知識が必要です。

- デバイス アーキテクチャ
- リモート デバッグ用の GNU デバッガー (GDB)
- Yocto、ザイリンクス PetaLinux、および Vitis ツールでのゲスト ソフトウェア アプリケーションの生成
- Linux デバイス ツリー

この文書では、ザイリンクス QEMU で Versal ACAP のエミュレーション用にサポートされる機能の一覧を示します。このセクションでは、ソフトウェア アプリケーション開発者、システム ソフトウェア開発者、システム ハードウェア設計者、およびバリデーション/ベリフィケーション設計者を対象に、QEMU を使用したソフトウェア デバッグに関する基本情報を提供します。

Versal ACAP の QEMU モデル

ザイリンクス Versal ACAP の QEMU モデルは、次のリソースをサポートしています。

- CPU (中央処理装置)
 - アプリケーション プロセッシング ユニット (APU)
 - 2 x Arm Cortex-A72 プロセッサ
 - リアルタイム プロセッシング ユニット (RPU)
 - 2 x Arm Cortex-R5F プロセッサ
 - プラットフォーム管理コントローラー (PMC)
 - プラットフォーム プロセッシング ユニット (PPU) 0 (MicroBlaze プロセッサ)
 - プラットフォーム プロセッシング ユニット (PPU) 1 (MicroBlaze プロセッサ)
 - PS 管理コントローラー (PSM)
 - 1 x MicroBlaze プロセッサ

- メモリ
 - オンチップ メモリ (OCM)
 - 密統合メモリ (TCM)
 - DDR
- セキュリティ モジュール
 - デバイス キー ストレージ
 - バッテリ バックアップ RAM (BBRAM)
 - eFUSE (Electronic Fuse)
 - PUF (Physical Unclonable Function) (API のみ)
 - 暗号プリミティブ
 - AES-GCM (Advanced Encryption Standard - Galois/Counter Mode)
 - SHA-3 (Secure Hash Algorithm 3)
 - RSA-4096
 - 楕円曲線 DSA (ECDSA)
- ペリフェラルおよびコントローラー
 - 2 x シリアル ペリフェラル インターフェイス (SPI)
 - Octal SPI (OSPI)
 - OSPI DMA
 - 2 x SD
 - eMMC
 - 2 x CANFD
 - 2 x UART
 - 3 x I2C (Inter-Integrated Circuit)
 - PS に 2
 - PMC に 1
 - 2 x ギガビット イーサネット
 - 4 x トリプル タイマー/カウンタ (TTC)
 - プロセッサ間割り込み (IPI)
 - 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)
 - 汎用割り込みコントローラー (GIC) v3
 - ダイレクト メモリ アクセス (DMA)
 - リアルタイム クロック (RTC)

- 。 USB (ホスト モードのみ)

次のステップ

次のステップは、ザイリンクス Wiki の「Getting Started」を参照してください。

AI エンジン開発環境

Versal AI コア シリーズは、現在のサーバー クラス CPU の 100 倍以上の演算性能を発揮する AI エンジンを用意し、AI 推論を飛躍的に高速化します。このシリーズは、動的ワークロードに対応したクラウドや、超高帯域ネットワークなど幅広いアプリケーションをサポートすると同時に、最先端の安全性とセキュリティ機能を提供します。ソフトウェアおよびハードウェア開発者だけでなく、AI およびデータサイエンティストも高い演算密度を利用してあらゆるアプリケーションの性能を加速できます。先進の信号処理演算性能を備えた AI エンジンは、無線、5G、バックホールなどの高度に最適化されたワイヤレス アプリケーションや、その他の高性能 DSP アプリケーションに最適です。

AI エンジンは SIMD (Single Instruction Multiple Data) ベクター ユニットの備えた VLIW (Very-Long Instruction Word) プロセッサをアレイ状に配置しており、特にデジタル信号処理 (DSP)、5G ワイヤレス アプリケーション、機械学習 (ML) を含む人工知能 (AI) テクノロジーなど、演算負荷の高いアプリケーションに高度に最適化されています。

AI エンジンの並列性には、命令レベルの並列性とデータレベルの並列性があります。命令レベルの並列性には、1 つのスカラ演算、最大 2 つのムーブ、2 つのベクター読み出し (ロード)、1 つのベクター書き込み (ストア)、および 1 つのベクター命令が含まれ、1 クロック サイクルで合計 7 ウェイの VLIW 命令が実行されます。データレベルの並列性は、1 クロック サイクルで複数のデータセットに処理を実行するベクター レベルの演算によって実現されます。各 AI エンジンにはベクター プロセッサとスカラ プロセッサ、専用のプログラム メモリ、ローカル 32KB データ メモリがあり、3 方向にある隣接するローカル メモリのいずれにもアクセスできます。また、DMA エンジンおよび AXI4 インターコネクト スイッチにアクセスして、ストリーム経由でほかの AI エンジンやプログラマブル ロジック (PL) または DMA と通信することもできます。AI エンジン アレイおよびインターフェイスの詳細は、『Versal ACAP AI エンジン アーキテクチャ マニュアル』 (AM009: [英語版](#)、[日本語版](#)) を参照してください。

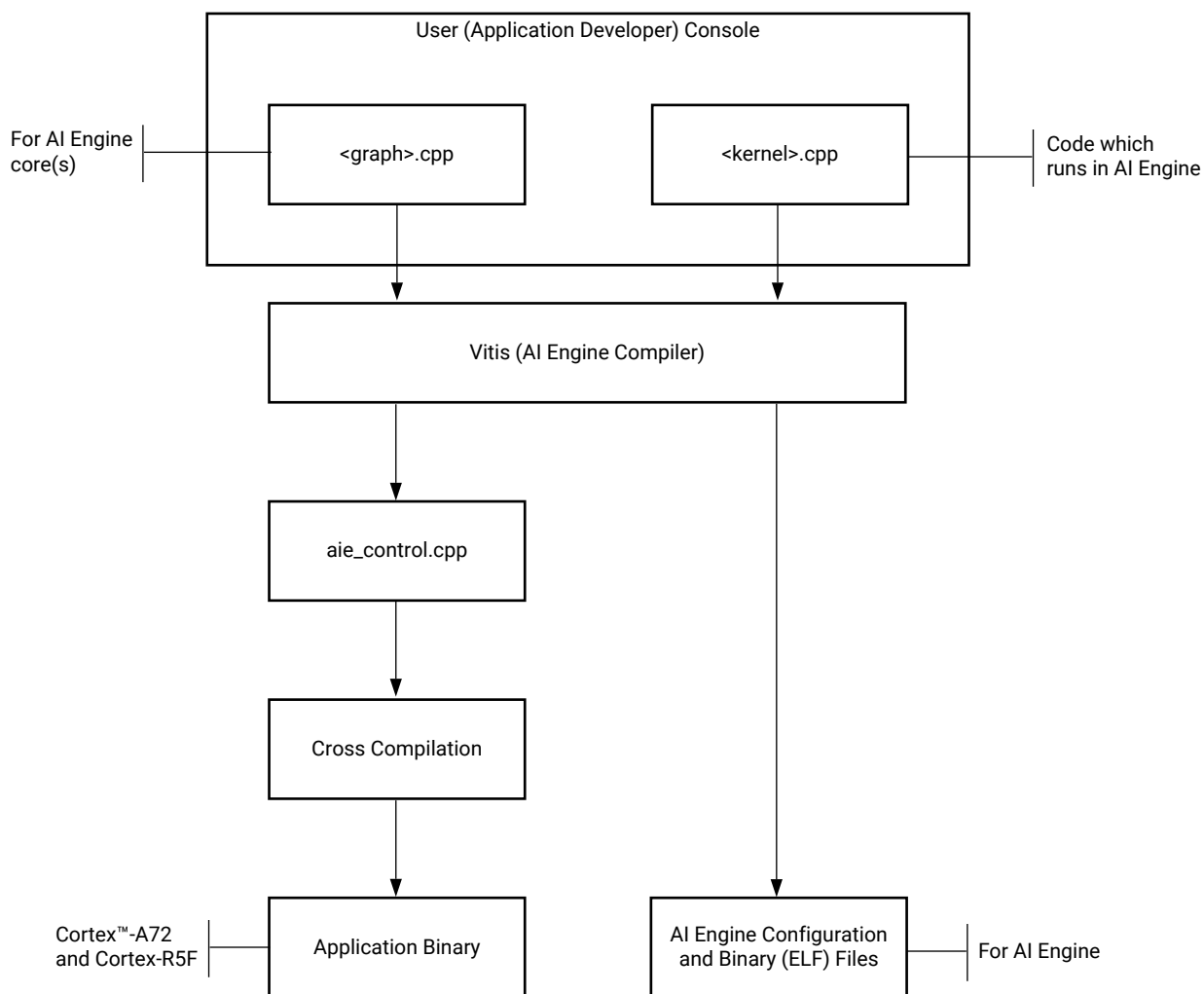
ザイリンクスは、ユーザー アプリケーションが Versal™ AI エンジンにアクセスするためのデバイス ドライバーとライブラリを提供しています。ザイリンクス AI エンジン プログラム環境の使用の詳細は、『Versal ACAP AI エンジン プログラミング環境ユーザー ガイド』 ([UG1076](#)) を参照してください。Versal ACAP AI エンジン エンジンの各種エレメントのコンパイル、シミュレーション、およびデバッグには Vitis IDE を使用します。Vitis IDE ツール フローの詳細は、『Versal ACAP デザイン ガイド』 (UG1273: [英語版](#)、[日本語版](#)) を参照してください。Vitis AI エンジン ツールおよびフローの詳細は、『Versal ACAP AI エンジン プログラミング環境ユーザー ガイド』 ([UG1076](#)) を参照してください。

以降のセクションで、AI エンジンのソフトウェア スタックについて説明します。

AI エンジン ソフトウェア開発フロー

AI エンジン アプリケーションは、AI エンジン タイル上で動作するカーネルと、CPU 上で動作する制御アプリケーションで構成されます。次の図に、AI エンジン制御アプリケーションと ELF の生成フローを示します。

図 6: AI エンジン ソフトウェア開発フロー



X23551-051920

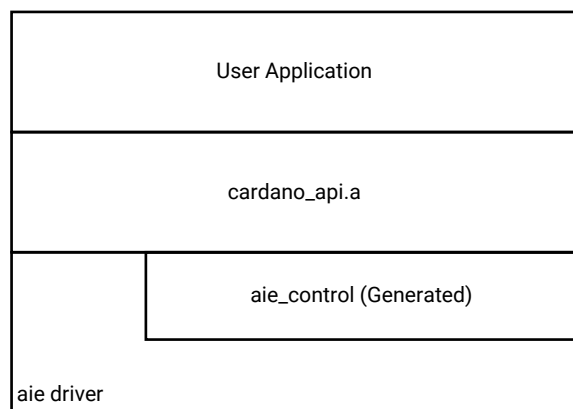
`<graph>.cpp` と `<kernels>.cpp` をユーザーが定義します。これらを AI エンジン コンパイラに入力すると、APU または RPU 上で動作して AI エンジンを設定および監視する `aie_control.cpp` と、AI エンジン タイル上で動作する ELF ファイルが生成されます。

AI エンジンのランタイム スタック

PS アプリケーションは、システム ライブラリを呼び出して AI エンジンのレジスタにアクセスし、カーネル ELF ファイルをロードできます。

- ベアメタル: アプリケーションは `cardano_api.a` ライブラリを使用して AI エンジンのレジスタにアクセスできます。`cardano_api.a` ライブラリは `aie_control` と AI エンジン ドライバーを呼び出し、AI エンジンのレジスタとデバイス メモリにアクセスします。

図 7: AI エンジンのベアメタル ランタイム スタック



X23552-112119

- Linux: Linux では、アプリケーションは AI エンジン ドライバー API または XRT API を使用して AI エンジンと相互に通信できます。

ソフトウェア スタック

この章では、Versal™ デバイスで利用可能なベアメタル、Linux、および FreeRTOS ソフトウェア スタックの概要を紹介します。Versal デバイスでは、多くのサードパーティ ソフトウェア スタックも使用できます。詳細は、[エンベデッド ソフトウェア/エコシステム](#) のページを参照してください。

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

ザイリックスは Vitis™ ツールの一部としてベアメタル 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 ソフトウェアには、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プロジェクト ドメインを備えたシンプルなシングル スレッド環境が含まれます。付属のボード サポート パッケージ (BSP) および付属のライブラリを設定することで、必要な機能が最小限のオーバーヘッドで得られます。スタンドアロン ドライバーは、次のパスにあります。

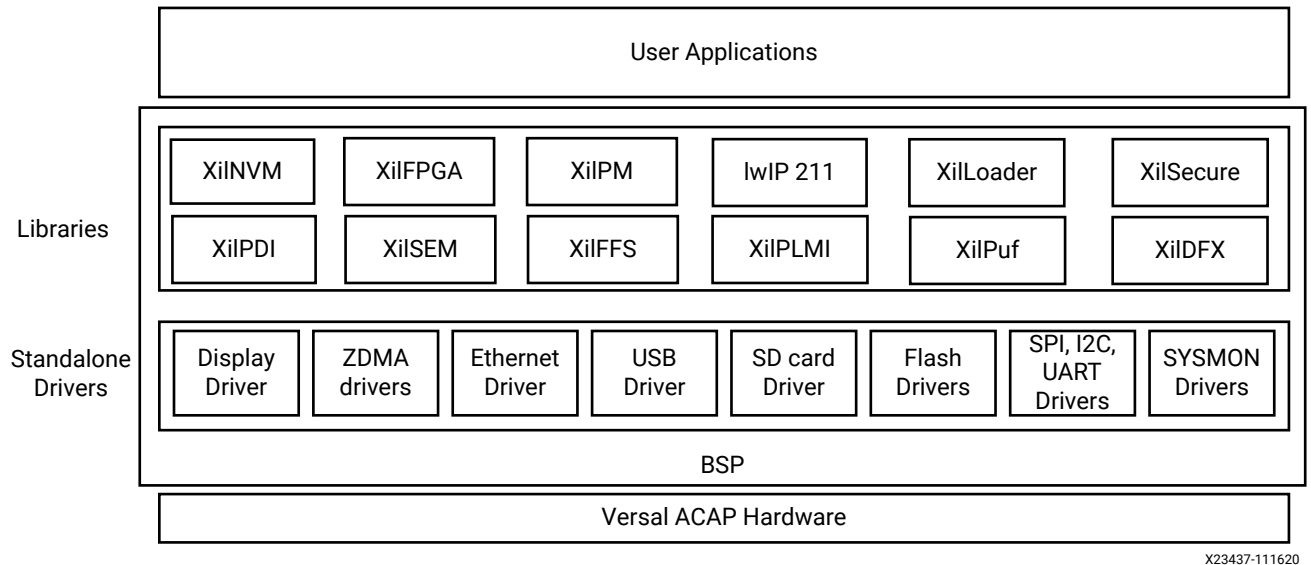
```
<Xilinx Installation Directory>\Vitis\<version>\data\embeddedsdsw  
\XilinxProcessorIPLib\drivers
```

ライブラリは、次のパスにあります。

```
<Xilinx Installation Directory>\Vitis\<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。

図 8: 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 レイヤーのソフトウェア スタックは、APU と同じです。

ベアメタル スタックの主なコンポーネントには、次のものがあります。

- ペリフェラル用のソフトウェア ドライバー: PS の Arm® Cortex™-A72、Cortex-R5F プロセッサ、および PL の 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 Cortex-A72、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。
- テスト アプリケーションを含むアプリケーション サンプル。

標準 C ライブラリ (libc)

libc ライブラリには C プログラムで使用できる標準関数が含まれます。libc ライブラリには、次のヘッダー ファイルが含まれます。

- `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

- `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

C 標準ライブラリ数学関数 (libm)

次の表に、libm 数学 C モジュールを示します。

表 3: 関数タイプ別の libm モジュールとその内容

| 関数タイプ | サポートされる関数 |
|------------------|---|
| 代数関数 | cbirt、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 | eil、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 はシンプルな下位ソフトウェア層です。この層は、キャッシュ、割り込み、例外などの基本的なプロセッサ機能、および標準入出力、プロファイリング、アボート/終了などホスト環境の基本的な機能へのアクセスを提供します。

次に、スタンドアロン BSP で利用可能なライブラリの一部を示します。

- XilFFS (ザイリンクス Fat File System) ライブラリ
- XilMailbox ライブラリ
- XilPM (Xilinx Power Management)
- XilSEM (ザイリンクス Soft Error Mitigation) ライブラリ
- lwIP (Lightweight IP)

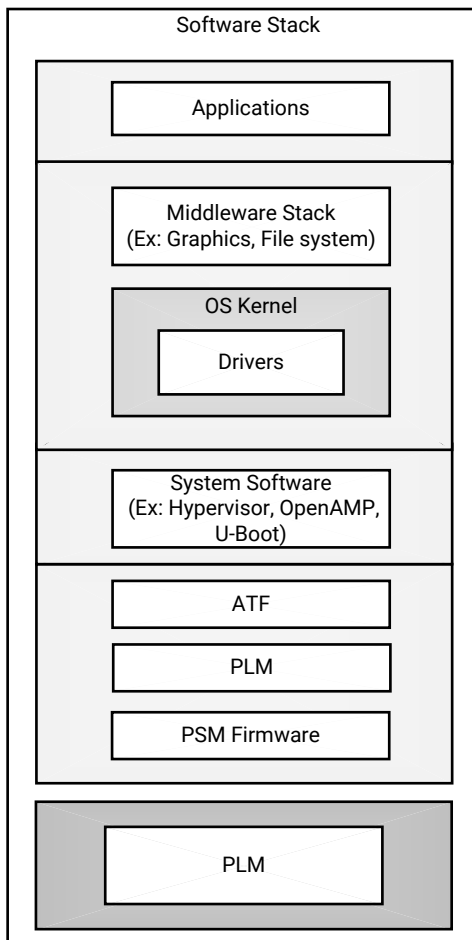
利用可能なライブラリの詳細は、[付録 A: ライブラリ](#) および『OS およびライブラリ資料コレクション』([UG643](#))を参照してください。

Linux ソフトウェア スタック

注記: Arm AArch64 アーキテクチャは Zynq UltraScale+ MPSoC APU と Versal ACAP APU で共通です。Linux ソースにあるアーキテクチャ参照表記「ZynqMP」は、Versal デバイスにも適用されます。

Linux OS は Versal デバイスをサポートしています。ザイリンクスは 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。

図 9: Linux ソフトウェア開発スタック



X23439-111620

ザイリンクスは、Versal デバイス向けのカスタム Linux ディストリビューションをビルドおよび運用するためのツールとして、PetaLinux ツールとオープンソース コラボレーション プロジェクトである Yocto Project の 2 つを提供しています。詳細は、<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841996/Linux> (ザイリンクス Wiki) を参照してください。

- PetaLinux ツール: PetaLinux ツールには、組み込み Linux を短時間でビルドするための簡単なコマンド セットが用意されています。このツールには 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。PetaLinux ツールは、次に示すすべてのオープン ソース Linux コンポーネントと組み合わせて使用できます。PetaLinux を使用してカスタム ディストリビューションをビルドする方法の詳細は、[PetaLinux ツール](https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842475/PetaLinux+Yocto+Tips) のページおよび <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842475/PetaLinux+Yocto+Tips> (ザイリンクス Wiki) を参照してください。

- Yocto Project: 経験のあるユーザーは、Yocto Project を使用すると個々のボードに合わせて組み込み Linux を高度にカスタマイズできます。Yocto Project に関心のあるユーザーのために、ザイリンクス Wiki にはザイリンクスデバイス上で Yocto を使用して Linux をビルドするための情報や記事が掲載されています。初めての方は、ザイリンクス Wiki の <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841883/Yocto> を最初に参照することを推奨します。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。

Versal 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。

- オープンソース Linux および U-Boot: ザイリンクスは、[Versal ACAP ボード、キットおよびモジュール](#) のページで Versal ACAP キット、VMK180 および VCK190 用のビルド済みイメージをリリースごとに提供しています。ドライバ、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 Versal デバイス用の Linux カーネルソースは、<https://github.com/Xilinx/linux-xlnx/> およびメイン Linux カーネルおよび U-Boot リポジトリから入手できます。詳細は、<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/115048462/Release+Notes+for+Open+Source+Components> (ザイリンクス Wiki) を参照してください。
- 商用 Linux ディストリビューション: オープンソース Linux 以外にも、ザイリンクスはサードパーティ数社と協力して Linux ソリューションを提供しています。一部の商用ディストリビューションは Versal デバイスもサポートしており、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、[エンベデッドソフトウェア/エコシステム](#) のページを参照してください。

FreeRTOS ソフトウェア スタック

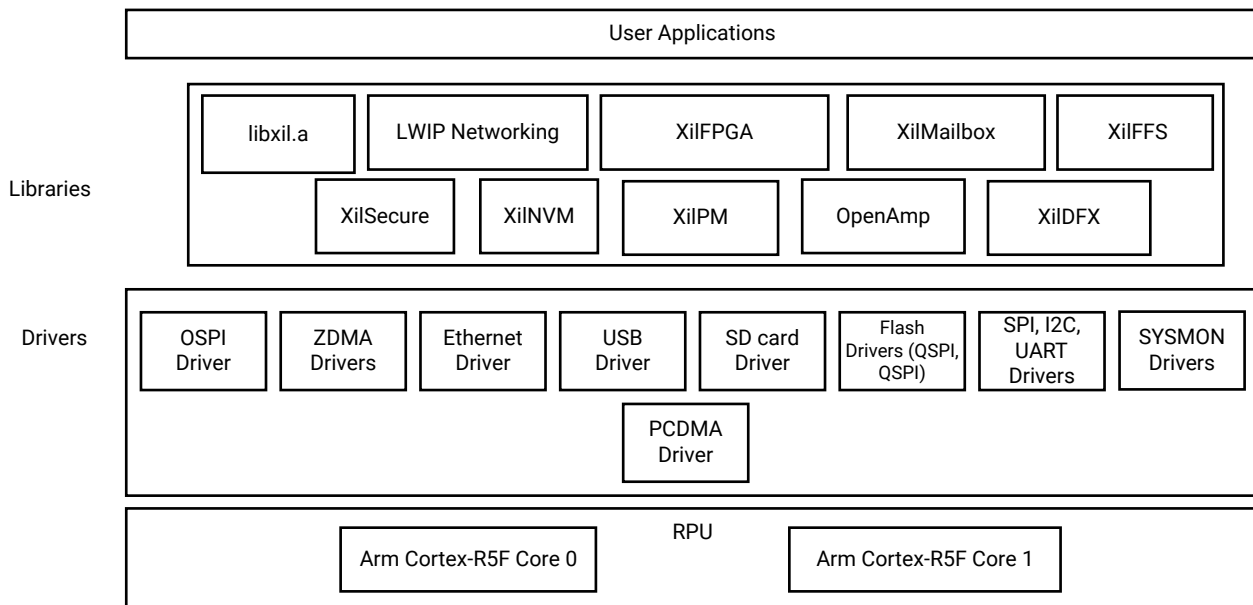
比較的シンプルな RTOS では、RTOS 自体が同じドメインで動作するアプリケーションと RTOS をリンクするライブラリのような役割を果たします。より高度な RTOS では、オプションのプロセス モード (VxWorks リアルタイム プロセス、Nucleus プロセス モデル、および QNX プロセスなど) を使用して Linux と同様のアプリケーションとカーネルのデカップリングが可能です。ザイリンクスは、Vitis ソフトウェア プラットフォームの一部として FreeRTOS BSP を提供しています。FreeRTOS BSP は、標準入出力やプロセッサ ハードウェア機能へのアクセス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 ライブラリを含むこと以外は、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。



重要: ザイリンクス ベアメタル ドライバは、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、同期中に使用するセマフォのメカニズムもありません。FreeRTOS カーネルでドライバ 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 10: FreeRTOS ソフトウェア スタック



X16911-111620

注記: APU の FreeRTOS ソフトウェア スタックは、ライブラリが APU 上で 32 ビットと 64 ビットの両方の動作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、次を参照してください。

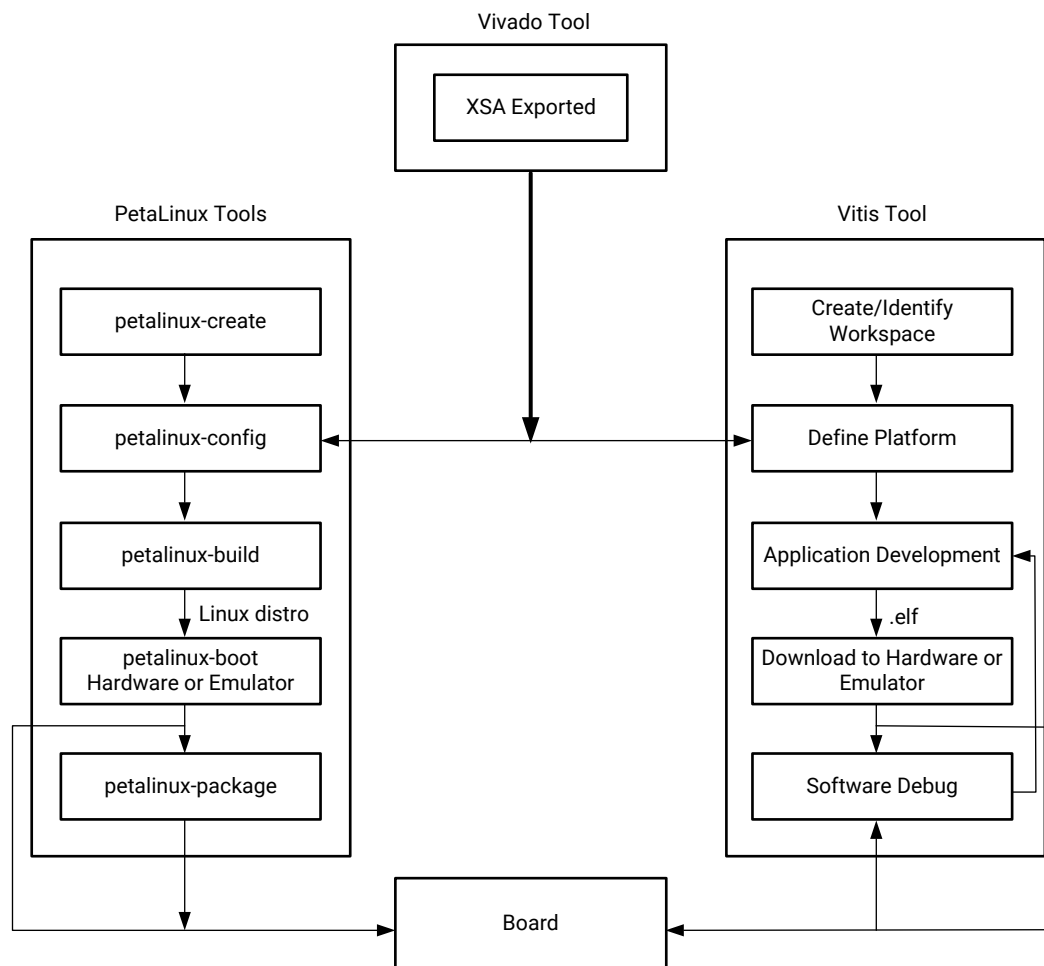
- [エンベデッド ソフトウェア/エコシステム のページ](#)
- [ザイリンクス サードパーティ ツール のページ](#)
- <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842463/3rd+Party+Operating+Systems> (ザイリンクス Wiki)

ソフトウェア開発フロー

この章では、Vitis™ IDE 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、基本的なソフトウェア開発フローを示します。

図 11: ソフトウェア開発フロー



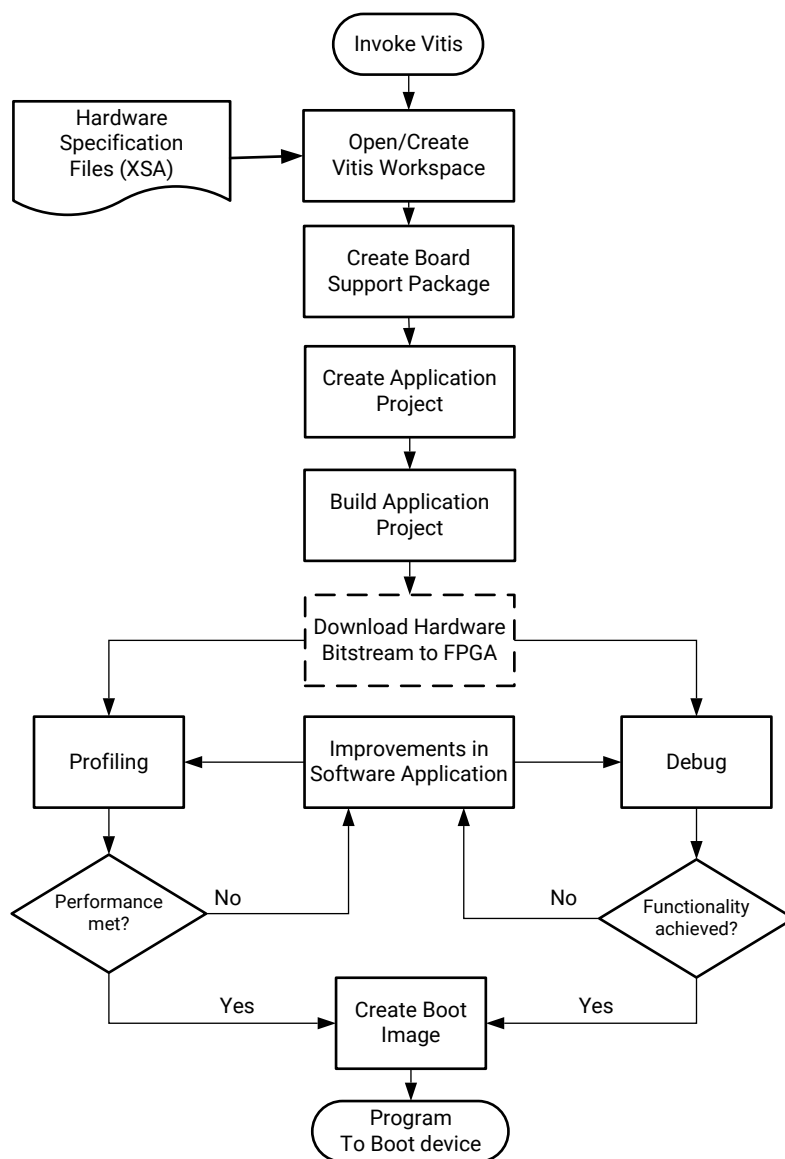
X24151-070920

詳細は、『Vitis 統合ソフトウェア プラットフォームの資料』 (UG1416) の [Vitis エンベデッド ソフトウェア開発フローの資料](#) を参照してください。

Vitis 環境でのベアメタル アプリケーション開発

次の図に、ベアメタル アプリケーション開発フローを示します。

図 12: ベアメタル アプリケーション開発フロー

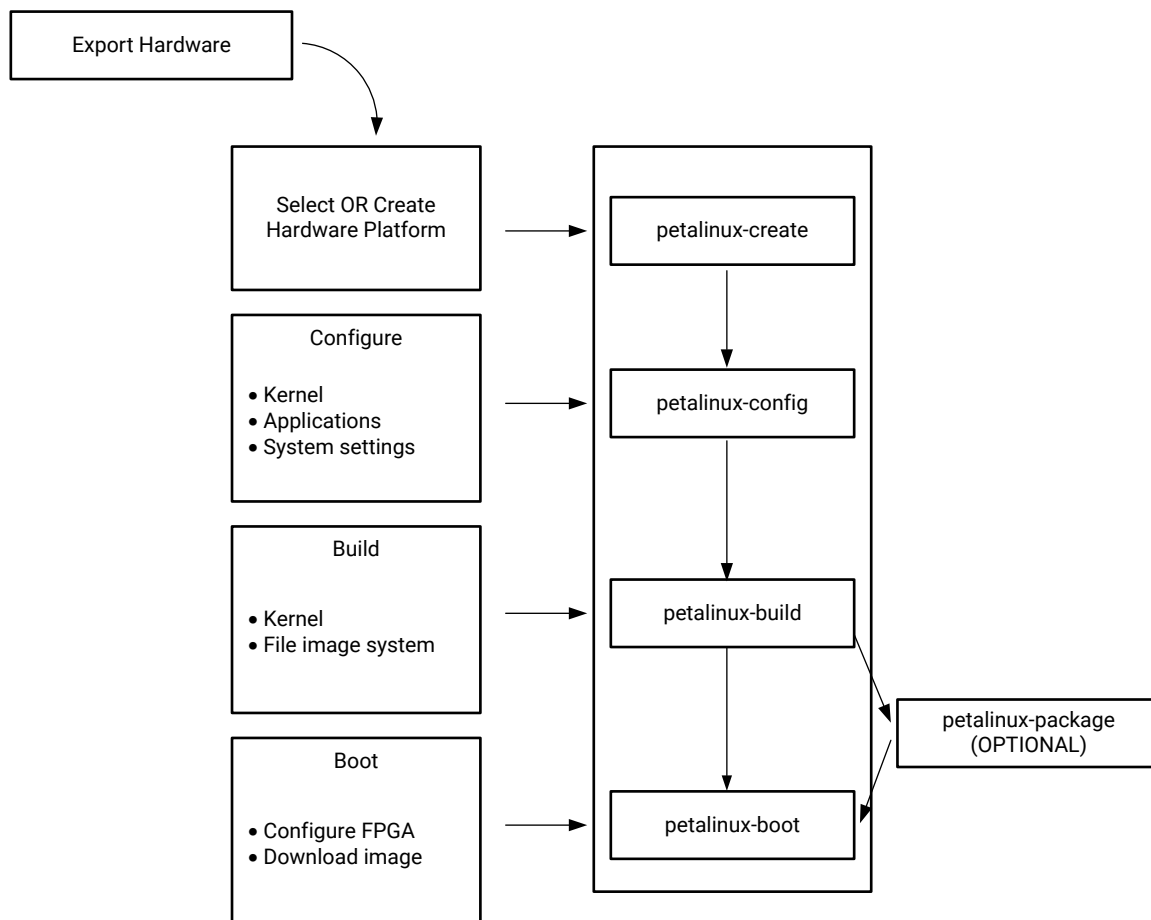


X14817-112420

PetaLinux ツールを使用した Linux 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。次の図に、PetaLinux ツール環境におけるアプリケーション開発の主要な工程を示します。

図 13: PetaLinux ツールを使用したソフトウェア開発フローの概略



X24150-062420

PetaLinux ツールとボード固有 BSP を使用して Versal デバイスの Arm® Cortex™-A72 コア ベース APU 向けに Linux OS プラットフォームを設定およびビルドする方法は、『ザイリンクス エンベデッド デザイン チュートリアル: Versal Adaptive Compute Acceleration Platform』(UG1305) の「PetaLinux を使用した Linux イメージの作成」を参照してください。

詳細は、『PetaLinux ツール資料: リファレンス ガイド』(UG1144: [英語版](#)、[日本語版](#)) の「プロジェクトの作成」、「設定およびビルド」、「ブートおよびパッケージ」を参照してください。

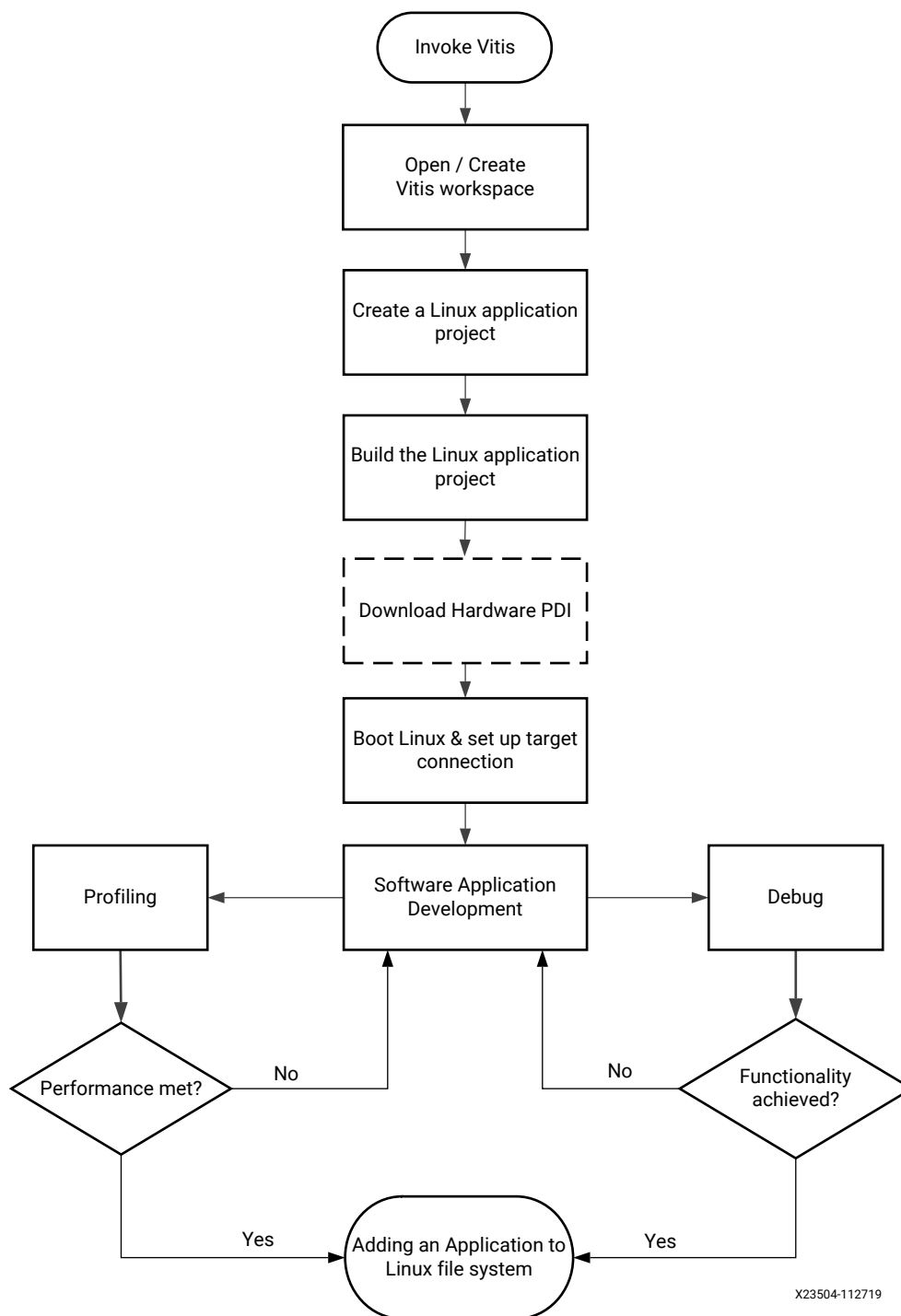
Vitis ソフトウェア プラットフォームを使用した Linux アプリケーション開発

Vitis 統合設計環境 (IDE) 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、Vitis ツールを使用した Linux アプリケーション開発フローの概要について説明します。

次の図に、Vitis プラットフォームを使用した 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。

注記: これらの手順は、Linux ドメインが利用可能な状態で Vitis プラットフォームをビルドした場合のみ有効です。

図 14: Linux ユーザー アプリケーションの開発



ソフトウェア アプリケーションの作成、サンプル プロジェクト アプリケーションの作成とビルド、およびアプリケーションのデバッグなど、Linux アプリケーション開発フローを Vitis ツールから完了する方法は、『ザイリンクス エンベデッド デザイン チュートリアル: Versal Adaptive Compute Acceleration Platform』 ([UG1305](#)) を参照してください。

- 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。
- 詳細は、『PetaLinux ツール資料: リファレンス ガイド』([UG1144](#)) の「カスタム モジュールの作成と追加」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

Versal[™] 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**: Versal 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**: PetaLinux で SMP を使用すると、Versal デバイス向け Linux オペレーティング システムで SMP デザイン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 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。

マルチプロセッサ開発用フレームワーク

Versal ACAP でのアプリケーション開発を容易にするため、ザイリンクスは次のような各種フレームワークを提供しています。

- **ハイパーバイザー フレームワーク**: ザイリンクスは、Versal ACAP で仮想化をサポートするために欠かせない Xen ハイパーバイザーをサポートしています。詳細は、[ハイパーバイザーの使用](#) を参照してください。
- **セキュリティ フレームワーク**: Versal デバイスは、セキュリティ フレームワークの一部として認証、暗号化、およびその他の暗号機能をサポートしています。セキュリティ フレームワークの詳細は、[第 9 章: 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ハードウェア/ソフトウェアと非セキュアなハードウェア/ソフトウェアに分離し、その分離を維持します。

ザイリンクスは、Arm[®] トラストド ファームウェア (ATF) を使用して TrustZone をサポートし、セキュア ワールドと非セキュア ワールドの分離を維持します。Versal デバイスに TEE (Trusted Execution Environment) を実装する場合、ATF は TEE の主要コンポーネントの 1 つとなります。TEE アーキテクチャの概要は、[このホワイトペーパー](#) を参照してください。

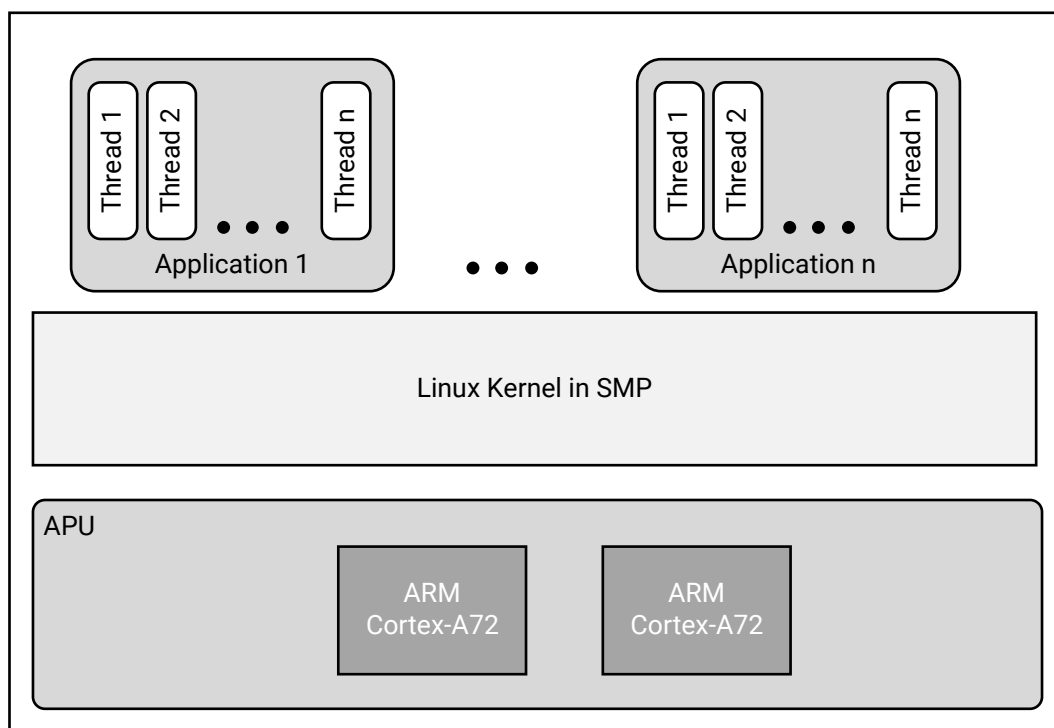
- **マルチプロセッサ通信フレームワーク**: ザイリンクスは、異なるプロセッシング ユニット間で互いに通信できるように、Versal デバイス用の OpenAMP フレームワークを提供しています。
- **電力管理フレームワーク**: 電力管理フレームワークにより、異なるプロセッシング ユニットで駆動されるソフトウェア コンポーネントが電力管理ユニットと通信できるようになります。

対称型マルチプロセッシング

SMP 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。

Versal デバイスの APU にはキャッシュ コヒーレントな 2 つのホモジニアス Arm Cortex™-A72 プロセッサがあり、これらは OS (Linux や VxWorks® など) を使用した SMP モードの動作をサポートしています。APU 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多くのオペレーティング システムを提供しています。次の図に、1 つの OS 上で複数のアプリケーションを実行した Linux SMP の例を示します。

図 15: Linux を使用した SMP モードの例



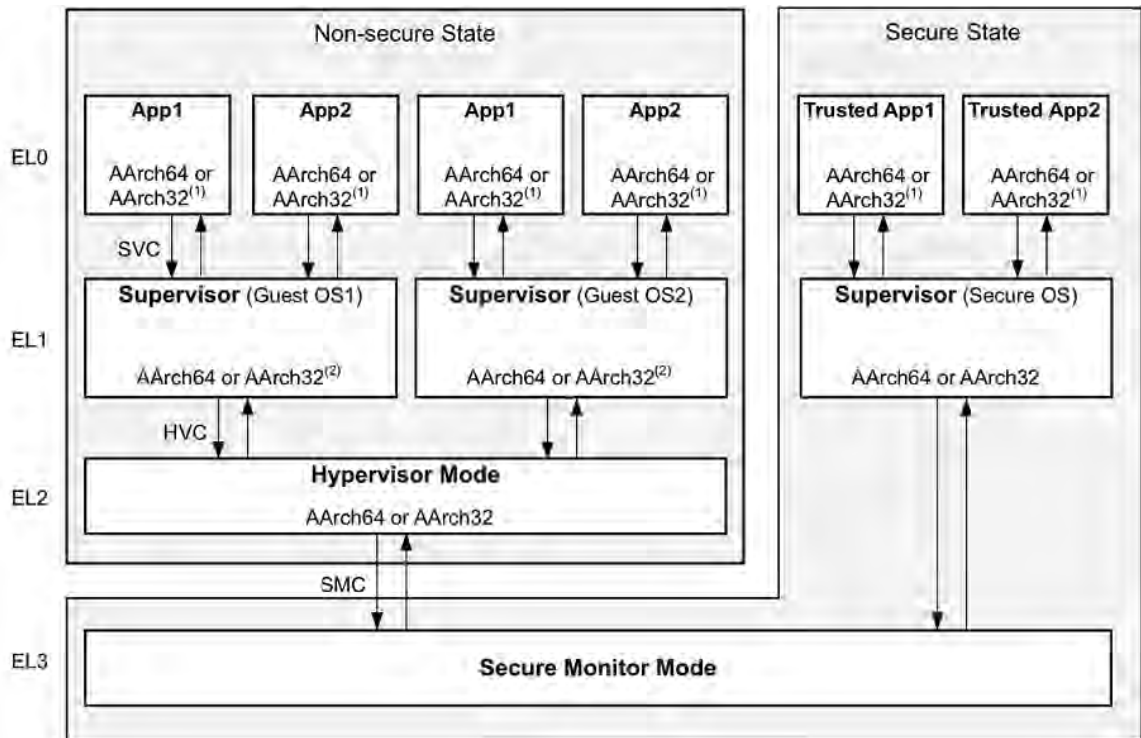
X23442-112719

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ないことがあります。

実行ドメインとイメージ

ドメインとは、それぞれ専用のアドレス空間を持つ独立した実行環境で、デバイスが接続されていることもあります。メモリ管理ユニット (MMU) を持たない最小構成の MicroBlaze プロセッサなど、シンプルな CPU にはドメインが 1 つしかありません。APU など、より高度な CPU には複数のドメインがあり、これらは次の図に示すように例外レベル 0 ~ 3 (EL0 ~ 3) に分割されます。また、その一部はセキュア ドメイン セグメント (SELO ~ 1) を持つことができます。

図 16: APU のセキュア ドメイン セグメント



次の表に、APU の各種ドメイン上で実行可能なものの例を示します。

表 4: ドメインの例

| ドメイン | 非セキュア | セキュア (TrustZone) |
|------|---|------------------------|
| EL0 | Linux プロセス/アプリケーション RTOS アプリケーション プロセス モデル | セキュアまたはトラステッド アプリケーション |
| EL1 | Linux カーネル RTOS カーネル | OP-TEE OS |
| EL2 | オプションのハイパーバイザー (Xen)、U-Boot | 将来の Arm プロセッサでサポート予定 |
| EL3 | ATF | |

CPU 型の実行エンジン (APU、RPU、MicroBlaze プロセッサ、AI エンジンなど) では、ある 1 つのドメインで動作するコンパイル済みプログラムをイメージと呼びます。通常は標準の ELF フォーマットを使用しますが、場合によっては PDI などのよりコンパクトなフォーマットに変換することもあります。

イメージという用語は、PL/FPGA で動作する「コード/ロジック」という意味でも使用されます。PL イメージ (ビットストリーム) も広い意味で使われる用語で、コンフィギュレーション情報 (デバイスの開始など) を指すこともあれば、PL ファブリック コンフィギュレーションに変換されるカスタマー コードを指すこともあります。

イメージがドメインにロードされる方法には、いくつかあります。

- ブート中:

- QSPI、SD、eMMC、SMAP、または OSPI にある PDI ファイル
 - bootROM が PLM をロードし、PLM がイメージの残りの部分をロードします。ハイパーバイザーを使用する場合は、U-Boot によってロードされます。U-Boot は Linux などの OS もロードします。
- JTAG デバッガーを使用してイメージをメモリに置くことができます。
- 動作中:
 - オペレーティング システムは、ファイル システム (rootfs、リモートファイルシステム、SD カードなど) からイメージをロードできます。
 - 別のプロセス (fork/exec)
 - 別の CPU (OpenAMP を使用)
 - ハイパーバイザーが別の仮想マシン (VM) および関連するソフトウェア セットをロードすることもできます。
 - ファイル システムにアクセスできるより高度な RTOS は、Linux と同様にイメージをロードできます。
 - PLM ファームウェアは、リスタート中またはクライアントの要求に応じてイメージを動的にロードできます。

ほとんどの場合、イメージがドメインにロードされたら、イメージ内のデータは変化しますが、コードはライフサイクルの最後まで変化しません。ただしこれには例外があります。代表的なのは、Linux がドライバーをカーネル モジュールとして動的にロードする場合、DFX 領域が PL にロードされる場合、そしてファームウェアが更新される場合です。

1 つのアプリケーションに複数のイメージを接続できます。これらの複数イメージ アプリケーションは、たとえば通常の Linux プロセスがアクセラレータを使用しており、それと同時にそのアクセラレータをロードする必要があるような場合に使用します。

非対称型マルチプロセッシング

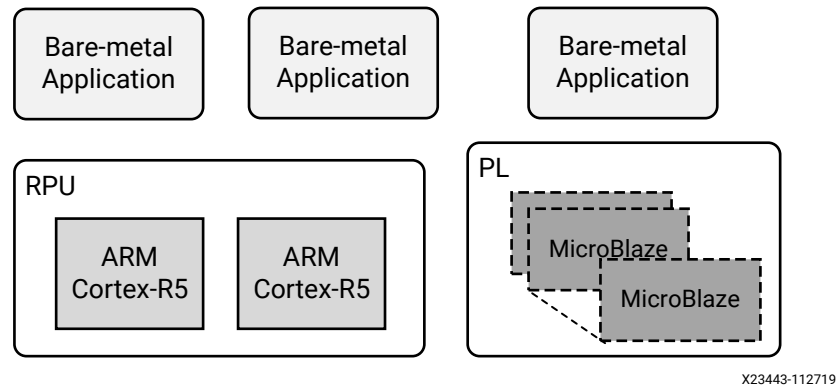
AMP モードでは、どのプロセッサで何を実行するかを厳密に制御して複数のプロセッサを使用します。SMP とは異なり、AMP にはいくつかの利用方法があります。このセクションでは、複雑度の異なる AMP の 2 つの利用法について説明します。

AMP 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、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。これには、APU および RPU プロセッサ クラスターが含まれます。たとえば、RPU の Arm Cortex-R5F プロセッサで AMP モードを使用すると (SMP はサポートされない)、非常に厳しいハード リアルタイム要件を満たすことができます。

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 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。

図 17: 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

ザイリンクスはオープンソース プロジェクトの OpenAMP に参加しています。OpenAMP は、APU と RPU の通信をサポートします。この通信パスは、Versal ACAP 全体を使用するために必要な基本機能を提供します。たとえば、OpenAMP を使用することで APU は必要に応じて RPU ソフトウェアをロードまたはアンロードしたり、RPU をリセットできます。

OpenAMP フレームワークは、AMP システム用ソフトウェア アプリケーションの開発に必要なソフトウェア コンポーネントを提供します。このフレームワークの主な機能は、次のとおりです。

- リモート コンピューティング リソースおよび関連するソフトウェア コンテキストの管理に使用される、ライフサイクル管理機能とプロセッサ間通信機能。
- RTOS およびベアメタル ソフトウェア環境で利用可能なスタンドアロン ライブラリ。
- Linux remoteproc および RPMsg アップストリーム コンポーネントとの互換性。

OpenAMP は、次の構成をサポートします。

- Linux マスター/汎用 (ベアメタル) リモート
- 汎用 (ベアメタル) マスター/Linux リモート
- 汎用 (ベアメタル) マスター/汎用 (ベアメタル) リモート

プロキシ インフラストラクチャおよび付属のデモでは、ベアメタル ベースのリモート コンテキストからの printf、scanf、open、close、read、および write 呼び出しをマスター上のプロキシで処理する方法を示しています。

強化されたシステム管理や高レベルの通信 API など、より高度な機能が必要な場合は、OpenAMP コミュニティ プロジェクトでザイリンクス SoC などをターゲットとした有用なコンテンツが見つかることがあります。

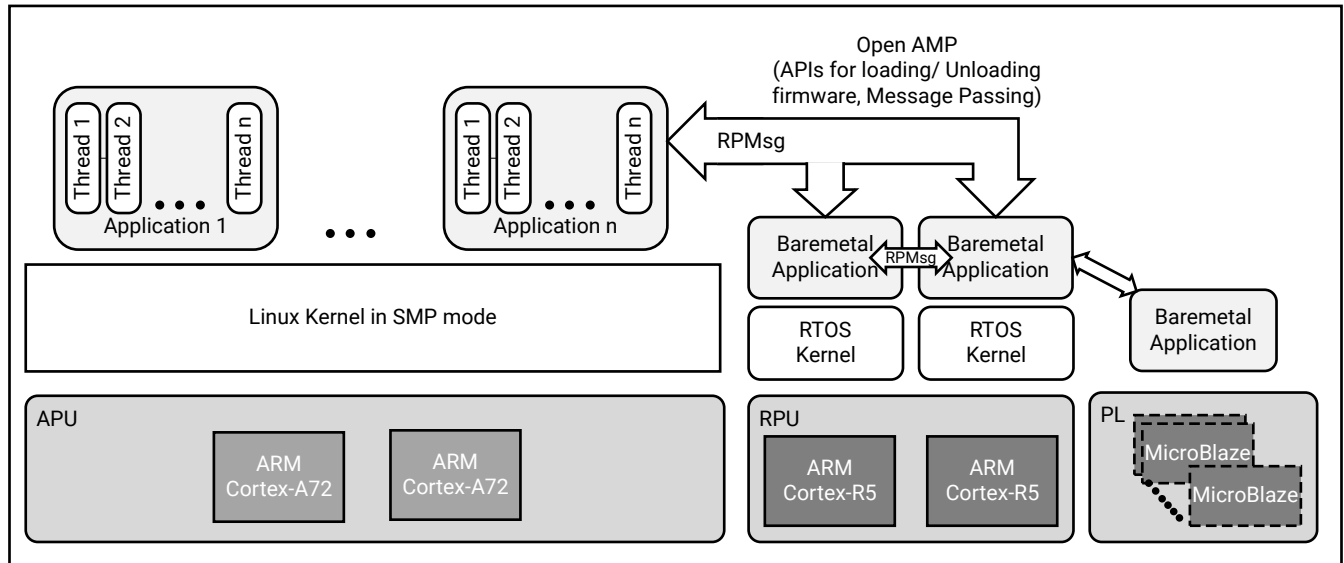
OpenAMP フレームワークには次のようなメカニズムが用意されています。

- 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

次の図に、Versal デバイス上の OpenAMP アーキテクチャの例を示します。

この例では、APU 上で動作する Linux アプリケーションが RPMsg プロトコルを利用して RPU と通信します。Linux アプリケーションは、Remoteproc フレームワークにより RPU に対してアプリケーションをロードおよびアンロードできます。これにより、開発者は必要に応じて各種の専用アルゴリズムを 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。このアーキテクチャは SMP モードと AMP モードを組み合わせて使用していることに注意してください。

図 18: OpenAMP フレームワークを使用した SMP + AMP の例



X23479-112719

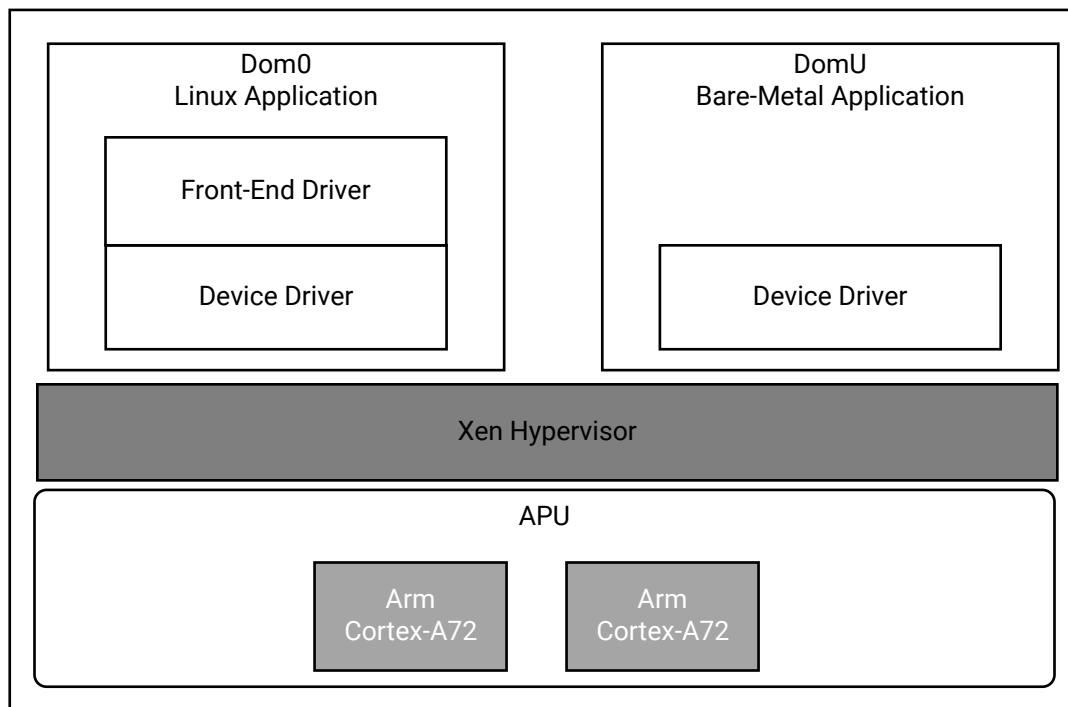
詳細は、<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841718/OpenAMP> (ザイリンクス Wiki) を参照してください。

ハイパーバイザーを使用した仮想化

Versal デバイスの Arm Cortex-A72 プロセッサ、Arm GIC-500 割り込みコントローラー、および Arm システム MMU (SMMU) はハードウェア仮想化の拡張機能をサポートしており、ハイパーバイザーを使用して高い性能を達成できます。

次の図に、Versal デバイスで動作するハイパーバイザー アーキテクチャの例を示します。この例では、ハイパーバイザーが Linux などの SMP 対応 OS、RTOS、またはベアメタル アプリケーションを実行しています。

図 19: ハイパーバイザー アーキテクチャの例



X23480-071020

ハイパーバイザーを追加すると、パワー マネージメント、PL 管理、OpenAMP ソフトウェア スタック、およびセキュリティ アクセラレータがアクセスするシステム ファームウェアの下層レイヤーが増えるため、低レベル システム機能の設計が複雑になります。ザイリンクスは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、早期から考慮して設計を進めることを推奨しています。

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 Versal 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。ハイパーバイザー上では複数のオペレーティング システムを実行できます。この OS のことを、ドメイン、仮想マシン (VM)、またはゲスト OS 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 OS 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

制御ドメインには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。Xen ハイパーバイザーの詳細は、<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842530/XEN+Hypervisor> (ザイリンクス Wiki) を参照してください。

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 ツール](#) のページを参照してください。

Xen 以外にも、いくつかのハイパーバイザーがザイリンクス エコシステム パートナー各社から提供されています。詳細は、[エンベデッド ソフトウェア/エコシステム](#) のページを参照してください。

ブートおよびコンフィギュレーション

この章では、Versal™ ACAP のブートおよびコンフィギュレーション プロセスの概要を紹介します。主にブート デバイスのオプション、およびプラットフォームのブートと各種コンポーネントのコンフィギュレーションの各ステージに関するソフトウェアについて説明します。

プラットフォーム管理コントローラー (PMC) は、ブートとコンフィギュレーション、およびポストブートのタスクを実行します。Versal ACAP のブートとコンフィギュレーションには、プログラマブル デバイス イメージ (PDI) と呼ばれるブート イメージを使用します。PDI はプラットフォーム ローダーおよびマネージャー (PLM) 実行ファイル、イメージ、およびコンフィギュレーション データで構成されます。PDI は、SD カード、eMMC、OSPI、QSPI などのブート デバイスから、または JTAG や SelectMAP などのスレーブ インターフェイス経由でロードできます。bootROM がブート プロセスを開始して PLM ソフトウェアをロードした後、この PLM ソフトウェアがイメージをロードし、イメージまたは PDI 内のコンフィギュレーション データに基づいてシステムをコンフィギュレーションします。

注記: ハードウェア関連の情報は、『Versal ACAP テクニカル リファレンス マニュアル』 ([AM011](#)) を参照してください。

Versal ACAP のブート プロセス

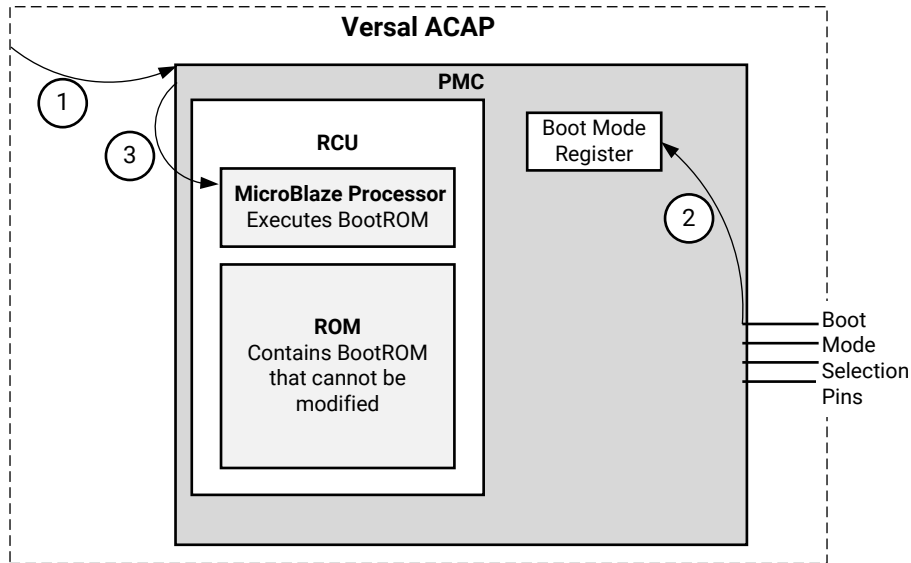
ブート プロセスは、ブート モードに依存しない 4 つのフェーズで構成されています。

- フェーズ 1: プリブート (パワーアップとリセット)
- フェーズ 2: ブートのセットアップ (初期化とブート ヘッダー処理)
- フェーズ 3: プラットフォームヘロード (ブート イメージの処理とコンフィギュレーション)
- フェーズ 4: ポストブート (プラットフォームの管理とサービスの監視)

以降のセクションでは、これら 4 つのフェーズの概要を簡単に説明します。

フェーズ 1: プリブート

図 20: フェーズ 1: プリブート (パワーアップとリセット)



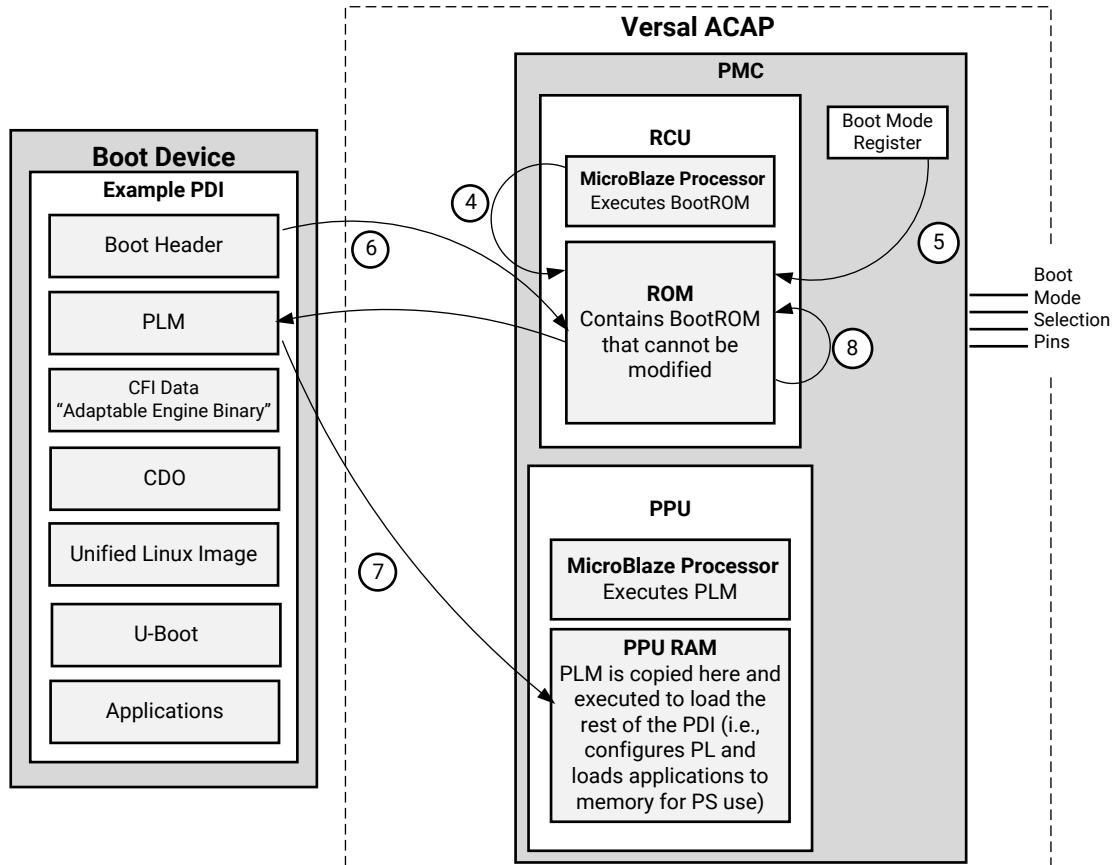
X22201-050119

- 1: プリブート フェーズは、PMC が PMC 電源を検知し、外部 `POR_B` ピンが解放されると開始されます。
- 2: PMC がブート モード ピンを読み出し、ブート モード レジスタに値を保存します。
- 3: PMC が RCU にリセット信号を送信します。

注記: その他すべてのプロセッサとコントローラー ユニットは、リセット状態を保持します。

フェーズ 2: ブート セットアップ

図 21: フェーズ 2: ブート セットアップ



X22200-071020

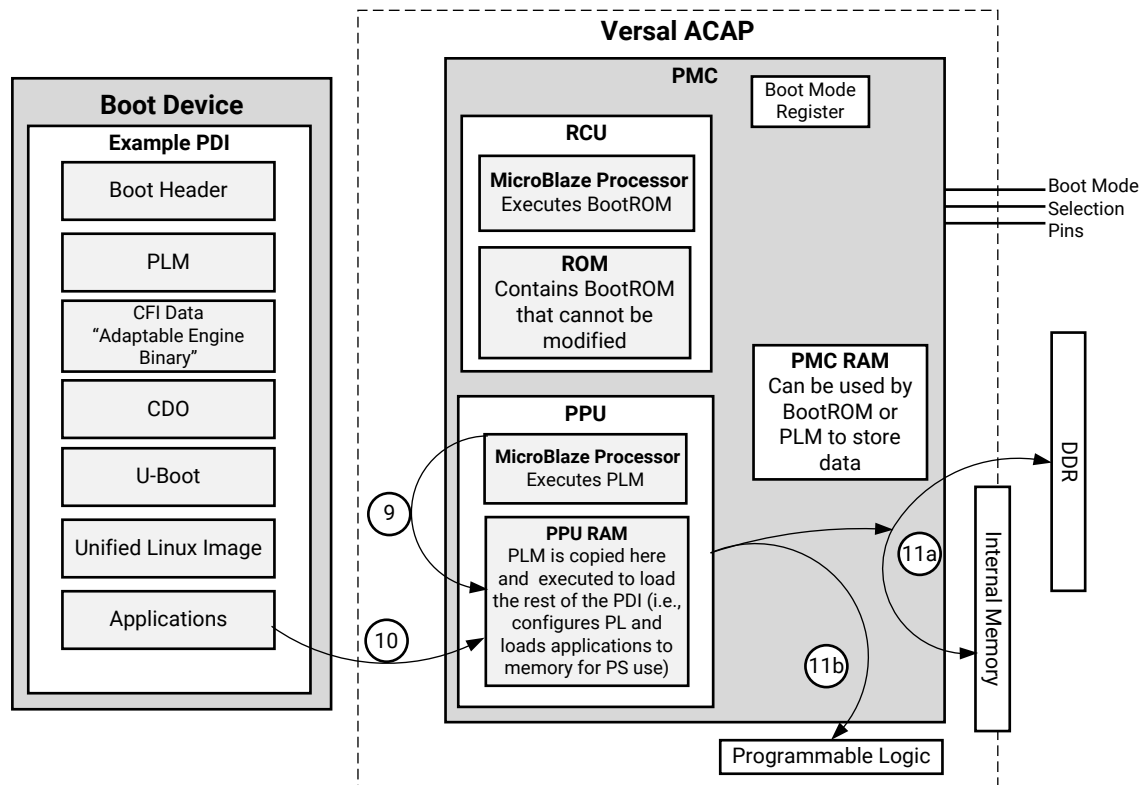
- 4: RCU が、RCU ROM から bootROM の実行を開始します。
- 5: bootROM がブート モード レジスタを読み出し、ブート デバイスを選択します。
- 6: bootROM がブート デバイスから PDI ブート ヘッダーを読み出し、検証します。
 - ブート ヘッダーが有効な場合は、bootROM がブート ヘッダーのデータに基づいてブート パラメーターを設定した後、ブート プロセスを継続します。
 - ブート ヘッダーが有効でない場合は、通常ブート プロセスがフォールバック ブート プロセスに移行します。
- 7: bootROM が PPU に対するリセットを解放し、PLM を PDI から PPU RAM にロードし、検証します。検証後、PPU がウェイクアップします。この時点で、PLM ソフトウェアが実行を開始します (フェーズ 3: プラットフォームヘロード の「9」を参照)。
- 8: bootROM 実行ファイルがスリープ状態に遷移します。bootROM 実行ファイルは、次のパワーオン リセット (POR) またはシステム リセットが実行されるまで動作を継続し、ブート後のプラットフォーム タスクを実行します。

関連情報

フェーズ 4: ポストブート

フェーズ 3: プラットフォームヘロード

図 22: フェーズ 3: プラットフォームヘロード



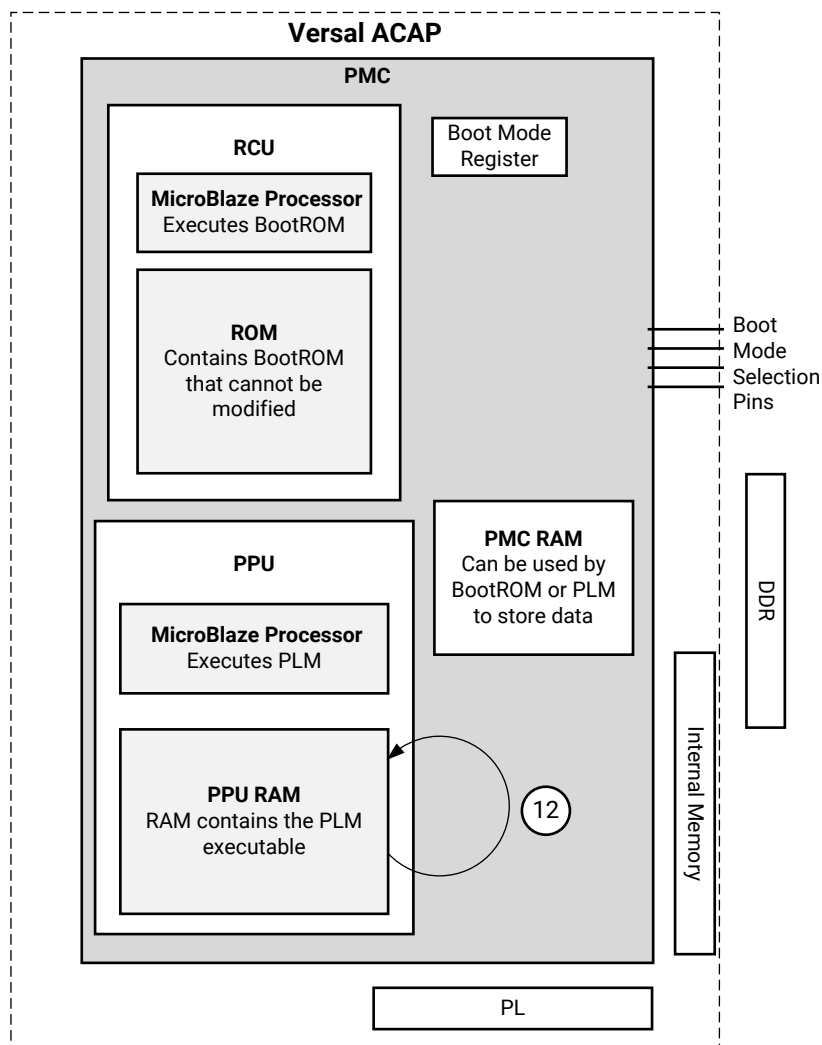
X22197-070620

- 9: PPU が PPU RAM から PLM の実行を開始します。
- 10: PLM が PDI コンポーネントを読み出して処理します。
- 11: PLM が PDI の内容に基づいて Versal デバイスの各種コンポーネントを設定します。
 - 11a: PLM は、次に示す Versal ACAP のコンポーネントにコンフィギュレーション データを適用します。
 - PMC、PS ブロック (CDO ファイル)
 - MIO (Multiplexed I/O)、クロック、リセットなど。
 - NoC 初期化および NPI コンポーネント (NPI ファイル)
 - DDR、NoC、GT、XPIPE、I/O、クロッキング、およびその他の NPI コンポーネント
 - 適応型エンジン (PL) データ (CFI ファイル)
 - AI エンジン コンフィギュレーション (AI エンジン CDO)
 - 11b: PLM は、個々のプログラミング制御/ステータス レジスタのスキャン クリアをトリガーします。

- 11b: PLM が、ELF ファイルで指定されたさまざまなメモリに Arm Cortex-A72 および Cortex-R5F プロセッサのアプリケーションとデータをロードします。これらのメモリには、オンボード DDR と内部メモリ (OCM、TCM など) が含まれます。

フェーズ 4: ポストブート

図 23: フェーズ 4: ポストブート



X22089-071020

- 12: PLM は、次の POR またはシステム リセットが実行されるまで動作を継続し、ブート後のプラットフォーム管理タスクを実行します。ポストブート サービスには、DFX リコンフィギュレーション、パワー マネージメント、サブシステムのリスタート、エラー管理、安全およびセキュリティ サービスなどが含まれます。

セカンダリ ブート デバイスの場合、PLM は次の動作を実行します。

1. PDI メタ ヘッダーのフィールドを読み出し、指定されたセカンダリ ブートデバイスを決定します。
2. 指定されたセカンダリ ブート デバイスを使用して、指定された PDI イメージをロードします。

ブート フロー

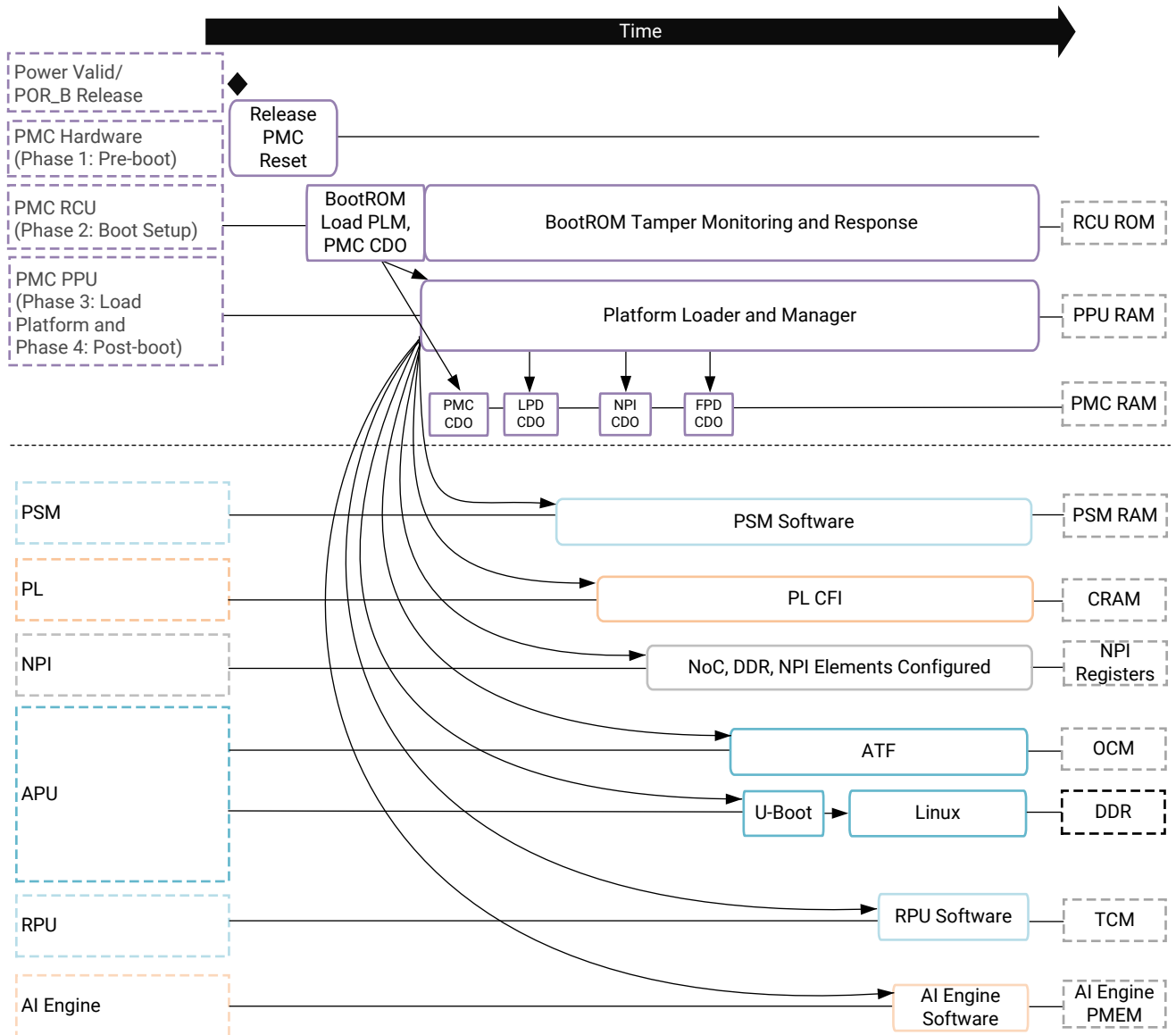
システムを起動するには、Versal ACAP はサポートされるブート ソースから正しく初期化、ブートおよびコンフィギュレーションを実行する必要があります。Versal ACAP アーキテクチャには、セキュア ブートと非セキュア ブートの 2 つのブート フローがあります。

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 タスクを実行するブート シーケンスの例について説明します。

非セキュア ブート フロー

次の図に、ブートおよびコンフィギュレーション シーケンスの例を示します。この図には、PLM が Versal ACAP ソフトウェア スタックの主要なパーティション コンポーネントをどのようにロードするかが示してあります。

図 24: 標準ブート フローのプロセッシング エンジンとメモリ ソースの例



X23595-110820

非セキュア ブート モードでは、bootROM が PLM を PPU RAM にロードし、PPU をリリースすると PLM の実行が開始します。PLM は PDI からイメージのロードを継続します。上記のブート フロー例では、PLM が CDO ファイルを使用して LPD、FPD および DDR を初期化しています。LPD イメージの一部として、PLM が PSM ファームウェアをロードして実行を開始します。PLM が RCDO ファイルを使用して PL CFI をロードします。PLM が ATF と U-Boot をロードして、APU 上で ATF (EL3-S) を開始します。ATF が U-Boot (EL2-NS) を開始します。次に、U-Boot が Linux をロードして処理を渡します。PLM は RPU と AI エンジン イメージもロードして実行を開始できます。

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-A72 プロセッサを管理します。

セキュア ブート フロー

セキュア ブート フローの詳細は、次を参照してください。

- 第 9 章: セキュリティ
- 『Versal ACAP テクニカル リファレンス マニュアル』 (AM011)

PDI コンテンツの整合性とセキュア ブートのサポート

PDI コンテンツの整合性は、チェックサムを使用して検証されます。PDI 内の実行ファイルとデータ オブジェクトは、ユース ケースに応じて暗号化または認証 (あるいはその両方) が可能です。PDI の作成に必要な中間生成物については、[ブート イメージ \(PDI\) の作成](#) を参照してください。

注記: ブート プロセスの説明で、「検証」という用語はチェックサム検証と認証の意味を含んでいます。

ブート デバイス モード

次の表に、プライマリ ブートに使用できるブート デバイス、およびブート デバイス モードを示します。詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 (AM011) を参照してください。

表 5: プライマリ ブート デバイス

| ブート モード | Mode[3.0] ピンの設定 | データ バス幅 | セキュア ブート | フォールバックブートとマルチブート | オフセット検索制限 |
|--|-----------------|----------|----------|-------------------|---------------|
| eMMC1 (4.51) | 0110 | x1、x4、x8 | ○ | ○ | 8191 FAT ファイル |
| JTAG | 0000 | x1 | × | × | N/A |
| Octal SPI シングル、またはデュアルスタック ⁵ | 1000 | x8 | ○ | ○ | 8Gb |
| Quad SPI24 シングル、またはデュアルスタック ⁵ | 0001 | x1、x2、x4 | ○ | ○ | 128Mb |
| Quad SPI24 デュアルパラレル | 0001 | x8 | ○ | ○ | 256Mb |
| Quad SPI32 シングル、またはデュアルスタック ⁵ | 0010 | x1、x2、x4 | ○ | ○ | 4Gb |
| Quad SPI32 デュアルパラレル | 0010 | x8 | ○ | ○ | 8Gb |
| SD0 (3.0) | 0011 | x4 | ○ | ○ | 8191 FAT ファイル |
| SD1 (2.0) | 0101 | x4 | ○ | ○ | 8191 FAT ファイル |
| SD1 (3.0) | 1110 | x4 | ○ | ○ | 8191 FAT ファイル |

表 5: プライマリ ブート デバイス (続き)

| ブート モード | Mode[3.0] ピンの 設定 | データ バス幅 | セキュア ブート | フォールバックブ ートとマルチブート | オフセット検索制限 |
|-----------|---------------------|------------|----------|-----------------------|-----------|
| SelectMAP | 1010 | x8、x16、x32 | ○ | × | N/A |

注記:

1. XIP (eXecute In Place) は Versal でサポートされません。
2. LQSPI (Legacy mode linear Quad SPI) は Versal でサポートされません。
3. オフセット検索制限機能は、bootROM 実行ファイルが、ブート ヘッダーと PLM を使用して PDI のブート デバイスを検索する場合に使用されます。これは、フォールバックやマルチブートで使用されます。
4. JTAG と SelectMAP はスレーブ ブート モードです。その他のデバイスはすべてマスター ブート モードです。
5. デュアル スタック QSPI の場合、bootROM ステージでアクセスできるのは最初のフラッシュ デバイスのみです。

ボード デザインに実装するブート デバイスを選択する際には、ブート後の共有 MIO (Multiplexed I/O) ピンの使用法と各ブート モードの電圧要件を考慮する必要があります。詳細は、『Versal ACAP テクニカル リファレンス マニュアル』(AM011) の「プラットフォーム管理コントローラー」の章を参照してください。

セカンダリ ブート プロセスとデバイスの選択

ブート プロセスには、SD、eMMC、QSPI、OSPI などのプライマリ ブート デバイスを使用することも、JTAG や SelectMAP などのスレーブ インターフェイスを使用することもできます。必要に応じて、ブート プロセスはプライマリ ブート デバイスを使用して開始し、セカンダリ ブート デバイスを使用して完了します。この場合、PLM はプライマリ ブート デバイスからイメージをロードし、セカンダリ ブート デバイスを設定します。セカンダリ ブート デバイスには PDI が格納され、この中にはセカンダリ ブート デバイスからロードする必要のあるイメージとコンフィギュレーション データが含まれます。このセカンダリ ブート デバイスには、PLM やブート ヘッダーは含まれません。セカンダリ ブート プロセスは、PDI メタ ヘッダーのフィールドにセカンダリ ブート デバイスが指定されている場合に実行されます。POR/システム リセットが実行されると、ブート プロセスはプライマリ ブート から開始します。

- bootROM:
 - ブート モード レジスタを読み出し、プライマリ ブート デバイスを決定する
 - 指定されたプライマリ ブート デバイスから PPU RAM に PLM をロードする
 - PPU のリセットを解除して PLM を実行する
- PLM:
 - PDI メタ ヘッダーのフィールドを読み出し、指定されたセカンダリ ブート デバイスを決定する
 - 指定されたセカンダリ ブート デバイスを使用して、PDI コンテンツの残りをロードする

注記: セカンダリ ブート デバイスとプライマリ ブート デバイスを同じにすることはできません。

セカンダリ ブート デバイスを次に示します。

- eMMC0、eMMC1

注記: コントローラーは、eMMC0 と eMMC1 の 2 つがあります。各コントローラーに 2 つの異なる MIO セットを割り当てるか、1 つの EMIO を割り当てることができます。1 番目のスロット (eMMC0) でのプライマリ ブートはサポートされません。2 番目のスロット (eMMC1) は、QSPI MIO ピンと競合するため使用できません。1 番目のスロットの eMMC1 (MIO26-36) はセカンダリ ブートに使用できます。

- イーサネット

注記: セカンダリ ブート デバイスとしてイーサネットを使用する場合、Versal ACAP はまずプライマリ ブート デバイスを使用して U-Boot まで起動します。その後、U-Boot がイーサネットを使用してブート プロセスを完了できます。

- OSPI

- PCIe® インターフェイス

1. まず、指定されたプライマリ ブート デバイスから PPU RAM に PLM がロードされます。
2. 次に、PLM が実行されて PCIe Endpoint (EP) モードの CCIX (Cache Coherent Interconnect for Accelerators) PCIe Gen4 モジュール (CPM) ブロックを初期化します。
3. 最後に、PCIe ホストがセカンダリ ブート デバイスとして、残りのイメージをロードします。

注記: PCIe インターフェイスはセカンダリ ブート デバイスとしてのみサポートされます。

- QSPI

- SD0、SD1

- USB

セカンダリ PDI は dfu_util を使用して固定 DDR アドレス (0x50000000) にダウンロードされます。dfu_util は、Windows および Linux 用に提供されているオープンソース ユーティリティです。その後、PLM が PDI を処理します。

USB をセカンダリ ブート モードとして示すには、BIF ファイルで usb を boot_device 属性として指定します。

- SelectMAP (SMAP)

SelectMAP をセカンダリ ブート モードにするには、BIF ファイルで smap を boot_device 属性として指定します。

フォールバック ブートとマルチブート

フォールバック ブートは、最初の PDI のブートに失敗した場合、Versal ACAP が同じプライマリ ブート デバイスのもう 1 つの PDI を自動的にブートできるようにするものです。PDI ロードのブート中にエラーが発生すると、PLM は bootROM が次の正常なイメージを見つけられるように、マルチブート レジスタの値を 1 つインクリメントしてデバイスをリセットします。

マルチブートは、次の Versal ACAP ブート プロセスで使用する PDI (最初の PDI とは異なる) を同じプライマリ ブート デバイスから指定できるようにします。

フォールバック ブートまたはマルチブートを使用する場合は、同じプライマリ ブート デバイスの検索範囲内に複数の PDI を格納しておく必要があります。詳細は、[ブート デバイス モード](#) のプライマリ ブート デバイスを参照してください。

Octal-SPI および Quad-SPI ブート デバイス

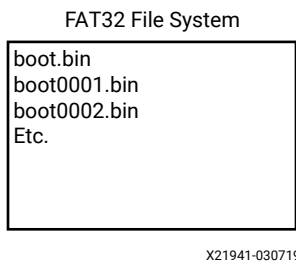
bootROM 実行ファイルは、Octal-SPI および Quad-SPI ブート デバイスを使用するフォールバック ブートとマルチブートをサポートしています。bootROM は 32k オフセットで QSPI および OSPI デバイスを検索します。たとえばマルチブート レジスタの値が 2 の場合、オフセット 0x10000 (2 * 32k) からイメージの検索を開始します。

SD/eMMC ブート デバイス

bootROM 実行ファイルは、SD カードと eMMC フラッシュ デバイスを使用するフォールバック ブートおよびマルチブートをサポートしています。SD カードまたは eMMC フラッシュは、最初のパーティションが FAT 16/32 ファイルシステムとなるように分割する必要があります。Bootgen を使用して、PDI ファイルを作成します。ファイル名は、boot.bin、boot0001.bin、boot0002.bin などとなります。

PMC_MULTI_BOOT の値が 0 の場合を除いて、まず文字列 boot の後に PMC_MULTI_BOOT の値が連結されます。これに拡張子 .bin を付けて、個々の PDI ファイル名が作成されます。たとえば、PMC_MULTI_BOOT=2 の場合、PDI ファイル名は boot0002.bin となります。コマンドライン ユーザーの場合、PDI ファイル名は Bootgen コマンドラインで指定します。その後、PDI ファイルは、ブート デバイス上の FAT16/32 ファイルシステムにコピーされます。デバイスに指定された検索制限は、boot8190.bin などのファイル名の最大数に相当します。

図 25: フォールバック ブートおよびマルチブート対応の SD および eMMC FAT16/32 ファイル



SD/eMMC RAW ブート モード

RAW パーティションとは、フォーマットされていない eMMC 上のパーティションを指します。bootROM と PLM は、従来のファイルシステム モードに加え、RAW フォーマットの SD/eMMC もサポートしています。

bootROM はブート イメージを検索して、ファイルシステムまたは RAW に存在するブート イメージを検出します。イメージが検出されると、bootROM はマルチブート レジスタにイメージの情報を追加します。PLM は、マルチブート レジスタの情報を使用してファイルシステムまたは RAW からブートを継続します。

マルチブート レジスタには、イメージがファイルシステムにあるか RAW にあるか、イメージのファイル番号またはオフセットなどの情報が記録されます。

セカンダリ ブートの組み合わせ

- SD/eMMC 以外の任意のプライマリ ブートと、セカンダリ ブート モードの SD/eMMC
- eMMC ブート パーティション 1/2 のプライマリ ブートと、eMMC ユーザー領域のセカンダリ ブート
- プライマリ ブート モードの eMMC ブート パーティション 1、ブート パーティション 2、およびユーザー領域

プログラマブル デバイス イメージ

Versal ACAP のブート イメージは PDI と呼ばれます。PDI は、Versal ACAP ブート プロセスまたはパーシャル コンフィギュレーション プロセスの一部として、PLM によって処理されるザイリンクス ファイル フォーマットです。PDI データは、設計要件に依存する固有のもので、通常、Versal ACAP のブート イメージはプラットフォームのブートとコンフィギュレーションに使用するバイナリで構成されます。これらのバイナリには、ブートローダー、ファームウェア、ハードウェア コンフィギュレーション データ、OS およびユーザー アプリケーションが含まれます。具体的には、適応型エンジン (PL) コンフィギュレーション ファイル、PLM イメージ、ATF、U-Boot、Linux カーネル、rootfs、デバイス ツリー、スタンドアロン/RTOS アプリケーションなどです。

PDI ファイルは複数のイメージを集めたもので、各イメージは 1 つ以上のパーティションで構成されます。各パーティションには次のものが含まれます。

- 順番に実行される一連のコマンドをリストにした CDO ファイル
- ロード可能データ (APU/RPU ELF やデータなど)

PDI ファイルは、ユース ケースによって異なります。

- Versal ACAP スタートアップ用 PDI: POR (パワーオン リセット) 時に使用されます。

この PDI には、Versal ACAP のブート、コンフィギュレーション、管理に必要な次の情報が含まれます。

- ブート ヘッダー
- PLM

注記: PLM は bootROM 実行ファイルによって読み出されます。

- 特定のサブシステムまたはエンジンのコンフィギュレーションに使用されるその他のサブシステム イメージとイメージ パーティション。
- これらイメージのいくつかは、セカンダリ ブートに使用する別の PDF の一部に含めることもできます。
- サブシステムのリスタートまたは DFX: 特定の電源管理条件、ウォーム リスタート、およびパーシャル リコンフィギュレーションを実行する際に使用されます。

このタイプの PDI には Versal ACAP のリコンフィギュレーションに必要な情報が含まれ、ブート ヘッダーと PLM は含まれません。

コンフィギュレーション データ オブジェクト

コンフィギュレーション データ オブジェクト (CDO) ファイルは、ツールで生成したコンフィギュレーション情報をプラットフォーム ロードерおよびマネージャー (PLM) に渡すために使用します。PDI ロード シーケンスの一部として、PLM は CDO から情報 (CDO コマンド) を各種コンポーネントに渡してシステムをコンフィギュレーションします。

コンフィギュレーションは複数の CDO ファイルとして生成でき、プログラマブル デバイス イメージ (PDI) ファイルのパーティションとして各 CDO ファイルを格納します。Versal ACAP の CDO は複数のコマンドをリストにしたもので、任意のサブシステムのプリロードまたはプリブート要件を含めることを目的としています。

適応型エンジン (PL) のコンフィギュレーション データは RAW ファイル (CFI ファイル) として生成され、これを Bootgen に入力します。Bootgen はこのコンフィギュレーション データを PDI にパーティションとして追加し、PDI イメージを作成します。

ブート イメージ (PDI) の作成

Versal ACAP のブートおよびコンフィギュレーションに使用するイメージ ファイル (PDI) は、ザイリンクスのブート イメージ生成ツール Bootgen を使用して作成する必要があります。Bootgen は、必要なブート ヘッダーを構築してイメージと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、PL コンフィギュレーション データ、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 (Versal ACAP の場合は PDI) を生成します。Bootgen には、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、Bootgen は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計算もサポートします。

Bootgen には、ブート イメージ フォーマット (BIF) ファイルと呼ばれるファイル (*.bif) を入力します。

ブート イメージのビルドの詳細は、『Bootgen ユーザー ガイド』 (UG1283: [英語版](#)、[日本語版](#)) を参照してください。

Vivado ツール フローは、バックエンドで Bootgen を使用して Versal ACAP 用の PDI を生成します。この PDI には、ハードウェア固有のユーザー コンフィギュレーション データ、適応型コンフィギュレーション データ、NPI データ、および PLM が含まれます。通常、Vivado ツールから出力された PDI は、Vitis ソフトウェア プラットフォームや PetaLinux などのソフトウェア ツールにエクスポートされます。これらのツールは、PDI コンポーネントを再利用するか、または PDI、追加のソフトウェア実行ファイル、またはイメージを入力してより大規模な PDI を作成します。Vitis ツールの PDI には、PLM (ユーザーが変更した場合を含む)、Vivado ツールで生成したコンフィギュレーション データ、そして一般的には APU/RPU コアで動作するアプリケーションが含まれます。

Bootgen は、Versal ACAP のブートおよびコンフィギュレーションに使用されるイメージ ファイル (PDI) の作成に不可欠です。Bootgen はこれらすべてのバイナリ イメージを配置して、イメージのロード時に PLM が解釈できるフォーマットでブータブル イメージを生成します。

Vivado ツールで生成された PDI には、次のイメージまたはコンフィギュレーション データが含まれます。

- ブート ヘッダー (PDI の一部、BIF の属性に基づく)
- PLM (プラットフォームに依存しない)
- PMC CDO
- メタ ヘッダー
- CFI データ
- LPD CDO および FPD CDO (CIPS コンフィギュレーションに基づく)
- PSM (CIPS コンフィギュレーションによる)

PL のみのフローなど、PMC および I/O の CIPS コンフィギュレーションが一部のみに限定されるようなユース ケースでは、LPD/FPD CDO や PSM など一部のデータ ファイルやイメージが不要なため、作成されないことがあります。

Vivado Design Suite で生成した PDI の拡張

Vitis ツールで作成し、Bootgen に入力されるファイルの例:

- PLM

注記: Bootgen は、Vivado で生成した PDI からの PLM および PSM ELF ファイルの置き換えをサポートしています。ただしこの機能を使用するのは、これらのファイルを特定用途向けに変更する必要がある場合のみです。ほとんどのユース ケースでは、Vivado ツールで生成したデフォルトの PLM および PSM ELF で正しく動作します。

- Arm Cortex-A72/Cortex-R5F アプリケーション
- AI エンジン コンフィギュレーション (DMA など)
- AI エンジン ELF
- U-Boot

Bootgen は、次の種類の入力を使用して PDI を作成します。

- Vivado ツールで生成した PDI。
- Vitis ツールから出力されるその他のファイル (任意で変更した PLM を含む)。カスタム PLM はデフォルトの PLM を置き換えます。ただし、ほとんどの場合はデフォルトの PLM をそのまま使用できます。
- ユーザーが作成したブート イメージ フォーマット (BIF) ファイル (オプション)。Vitis ツール PDI の作成に必要な命令を Bootgen に提供します。ごく一部のユース ケースにおいて、Vivado で生成した一部のファイルが更新された場合、Vivado で生成した BIF を必要に応じて再利用して、新しい PDI を作成/更新することもできます。

PDI をプライマリ ブート デバイスにコピーする方法

マスター ブート モードを使用するシステム コンフィギュレーションの場合、PDI ファイルをブート デバイスにコピーする必要があります。次の方法を使用して、PDI ファイルをマスター ブート モード デバイスにコピーできます。

- Vivado ツールのハードウェア デバイス マネージャー
- Vitis ツールのプログラム フラッシュ ユーティリティ
- SD カードなどの取り外し可能なデバイスは、個別にプログラムしてからボードに追加できます。
- QSPI などのソケット式やはんだ式のデバイスは、オフボードでプログラムしてからボードに装着できます。

注記: Vivado ツールのハードウェア デバイス マネージャーまたは Vitis ツールのプログラム フラッシュ ユーティリティを使用して PDI を ブート デバイスにコピーする場合は、ホスト PC から PMC JTAG ポートにケーブルを接続します。マスター ブート デバイスは、JTAG インターフェイスを介してプログラムされます。JTAG は U-Boot で Versal ACAP を起動する際にも使用され、U-Boot はマスター ブート デバイスのプログラムに使用できます。

BIF ファイルを使用して PDI ファイルの作成を制御 (ソフトウェア 開発者用)

PDI 内のサブシステム イメージやサブシステム イメージのパーティションを指定するには、ザイリンクス ブート イメージ (BIF) ファイルが使用されます。サブシステム イメージ内にある各ファイルは、サブシステム イメージ パーティションにそれぞれ格納されています。BIF ファイルは、ASCII 形式のデータ ファイルです。Bootgen は、BIF ファイルの指示に従って各入力ファイルを処理し、PDI を作成します。

注記: PLM は bootROM で処理されるため、サブシステム イメージまたはパーティションとしてフォーマットされません。

コマンド ラインを使用する場合は、Vivado で生成した BIF ファイルを編集し、Bootgen を実行して BIF ファイルの命令に従って PDI を作成する必要があります。ユーザー自身で BIF ファイルを記述し、Vivado で生成した PDI とその他の入力ファイルを指定して、PDI を拡張することもできます。Vitis IDE を使用する場合は、ウィザードを使用して BIF ファイルに必要な入力を指定します。その後、Vitis IDE で BIF ファイルを作成し、Bootgen を実行して PDI を作成します。

注記: Bootgen ウィザードは、Versal ACAP にすべて対応しているわけではありません。今のところ、暗号化や認証などのセキュリティ機能を備えない PS および AI エンジンのパーティションのみに対応しています。

プラットフォーム ロードおよびマネージャー

プラットフォーム ロードおよびマネージャー (PLM) は、プラットフォーム管理コントローラー (PMC) のプラットフォーム プロセッシング ユニット (PPU) 上で動作します。PLM は Versal ACAP のブートおよびコンフィギュレーションを実行し、最初のブートおよびコンフィギュレーションの後、サービスを継続的に提供します。

最初のブート時に、bootROM がプログラマブル デバイス イメージ (PDI) をデコードし、PLM を PPU RAM にロードします。PPU PLM は PDI を処理し、PDI に存在するパーティションをロードしてシステム全体をブートアップします。PLM は、実行時にパーシャル PDI をロードできます。詳細は、『Versal ACAP テクニカル リファレンス マニュアル』(AM011) を参照してください。

PLM ブートおよびコンフィギュレーション

bootROM、PLM ハンドオフ ステート

bootROM はブート デバイスから PLM を PPU RAM へロードした後、PPU をリセットから解放し、PPU が PLM の実行を開始します。

PLM ELF は PPU RAM にロードされます。PMC RAM を使用して PMC DATA CDO ファイルをロードします。

bootROM ハンドオフ時点のシステム ステートは、次のとおりです。

- 通常の JTAG ブート モードの場合、リセットが解放されて PPU はスリープ状態。
- PPU RAM と PMC RAM はエラー訂正符号 (ECC) を使用して初期化される。
- JTAG IDCODE 命令は、ブート モードにかかわらず常に利用できます。必要に応じて eFUSE をプログラムすると、JTAG IDCODE 命令以外のすべての JTAG 命令を無効にできます。eFUSE をプログラムしておらず、セキュア ブートが実行された場合、基本 JTAG 命令のみがサポートされます。セキュア ブート モードで AUTH_JTAG イネーブル命令が Versal ACAP に送信された場合、基本命令と拡張命令を合わせたすべての JTAG 命令を有効にできません。
- ブート デバイスはリセットから解放され、初期化された状態。
- マスター PLM がスレーブ PMC デバイスに接続できるように、スレーブ SLR (Super Logic Region) では NoC のデフォルト コンフィギュレーションが有効。

PLM サブシステム

表 6: PLM サブシステムのコンポーネント

| ファイル | 内容 |
|---------|--------------|
| PLM ELF | PLM ELF ファイル |

表 6: PLM サブシステムのコンポーネント (続き)

| ファイル | 内容 |
|---------|--|
| PMC CDO | PMC CDO ファイル <ul style="list-style-type: none"> デバイス トポロジ PMC のコンフィギュレーション: MIO、クロック、リセットに関するレジスタの書き込み/ポーリング |

bootROM は PLM ELF ファイルと PMC CDO ファイルをそれぞれ PPU RAM と PMC RAM にロードします。

bootROM から PLM へのハンドオフ後:

- PPU およびレジスタ割り込みを初期化します。
- モジュールを初期化して CDO/IPI コマンドおよびハンドラーを登録します。
- PMC CDO ファイルを設定します。
 - デバイス トポロジ: PMC CDO コマンドでノードを登録。
 - PMC と LPD MIO、クロックなどを初期化するための汎用/プラットフォーム管理呼び出し。
 - クロック、MIO、リセットに関する PMC 初期化。

LPD のコンフィギュレーション

表 7: LPD のコンフィギュレーション

| ファイル | 内容 |
|---------|---|
| LPD CDO | <ul style="list-style-type: none"> • PS LPD PM 初期化ノード コマンド (SC、LBIST、BISR、MBIST) • LPD のコンフィギュレーション: レジスタ書き込み/ポーリング |
| PSM ELF | PSM ELF ファイル |

PMC CDO の初期化後:

- PLM がブート デバイスを初期化し、ブート デバイスから LPD CDO ファイルをロードします。
- LPD CDO ファイルを設定します。
 - 必要に応じて、LPD に対してスキャン クリア、BISR、MBIST を開始します。
 - XiIPM がリセットを解放し、CDO 要件に基づいてノードをパワーアップします。
- PLM が PSM ELF ファイルをロードし、初期化が完了するのを待ちます。

LPD のコンフィギュレーションをロードする前に、PMC のコンフィギュレーションが完了していることを確認してください。

PL のコンフィギュレーション

表 8: PL のコンフィギュレーション

| ファイル | 内容 |
|-----------------|---|
| PL CDO <.rcdo> | <ul style="list-style-type: none"> PL ドメインのスキャン クリア、ハウス クリーニング、BISR のための PM 初期化ノード コマンド CFU へのレジスタ書き込み CFI データをロードするための DMA キーホール転送コマンド CFU へのレジスタ書き込み/ポーリング NPI が存在しない場合: <ul style="list-style-type: none"> グローバル信号 (GMC_B、GRESTORE、GHIGH_B..): レジスタ書き込み/ポーリング グローバル信号 (GWE、EOS、EN_GLOB): レジスタ書き込み/ポーリング |
| NPI CDO <.rnpi> | <p>NPI データ</p> <ul style="list-style-type: none"> PM 初期化ノード コマンド (SC、BISR、MBIST) NPI データのロード: DMA 書き込み/レジスタ書き込み CFI が存在する場合: <ul style="list-style-type: none"> グローバル信号 (GMC_B、GRESTORE、GHIGH_B..): レジスタ書き込み/ポーリング NPI シーケンス: レジスタ書き込み/ポーリング CFI が存在する場合: <ul style="list-style-type: none"> グローバル信号 (GWE、EOS、EN_GLOB): レジスタ書き込み/ポーリング 分離および PL リセット コマンド |

NPI データは、各種 NPI ブロック用に Vivado ツールによって生成されます。Versal ACAP に存在する NPI ブロックとしては、NoC、DDR、XPHY、XPIO、GTy、MMCM などがあります。

PL のコンフィギュレーションをロードする前に、PMC のコンフィギュレーションが完了していることを確認してください。

FPD のコンフィギュレーション

表 9: FPD のコンフィギュレーション

| ファイル | 内容 |
|---------|---|
| FPD CDO | <ul style="list-style-type: none"> PS FPD PM 初期化ノード コマンド (SC、BISR、MBIST) FPD のコンフィギュレーション: レジスタ書き込み/ポーリング |

FPD CDO をロードする前に、PMC と LPD のコンフィギュレーションが完了していることを確認してください。

DFX のコンフィギュレーション

Dynamic Function eXchange (DFX) コンフィギュレーションは、デバイスの 1 つ以上のサブ領域を新しいコンフィギュレーション データで独立して再プログラムする機能で、その他の領域 (スタティックまたはリコンフィギュレーション可能領域) は影響を受けずに動作を継続します。DFX PDI は、PCIe、DDR、またはプライマリ ブート デバイスから取得できます。DFX PDI のロードには、IPI インターフェイスで XilLoader CDO コマンドを実行します。

CPM のコンフィギュレーション

- CPM: レジスタ書き込みを使用した CPM コンフィギュレーション CDO

CPM CDO をロードする前に、PMC、LPD およびステージ 1 PL コンフィギュレーションが完了していることを確認してください。ステージ 1 PL コンフィギュレーションには XPIPE、GT、および NPI ファイルの NoC コンフィギュレーション データを含めます。

プロセッサ サブシステムのコンフィギュレーション

APU と RPU はプロセッサ ベースのサブシステムに属します。プロセッサ ベースのサブシステムには、いずれもイメージの一部として ELF ファイル/CDO があります。プロセッサの詳細はイメージ ヘッダーから読み出し、プロセッサは XilPM コマンドを使用して初期化します。

このコンフィギュレーションは、次のファイルで構成されます。

表 10: プロセッサ サブシステムのコンフィギュレーション

| ファイル | 内容 |
|----------------------|--|
| PSM/RPU/APU CDO ファイル | <ul style="list-style-type: none"> • (オプション) ノードと要件を指定した PM コマンド セット |
| PSM/RPU/APU ELF ファイル | <ul style="list-style-type: none"> • Cortex-R5F プロセッサ アプリケーション: ベアメタル/RTOS • Cortex-A72 プロセッサ アプリケーション: ATF/U-Boot/Linux/ベアメタル |

- Cortex-R5F プロセッサ アプリケーションをロードする場合は、LPD コンフィギュレーションが完了していることを確認してください。
- Cortex-A72 プロセッサ アプリケーションをロードする場合は、FPD コンフィギュレーションが完了していることを確認してください。
- DDR メモリを使用して Cortex-R5F/Cortex-A72 プロセッサ アプリケーションをロードする場合は、PL コンフィギュレーション (DDR を含む NPI コンフィギュレーション) が完了していることを確認してください。
- Cortex-R5F/Cortex-A72 プロセッサ アプリケーションを DDR にロードする場合は、デザインで PMC から DDR への NoC パスを有効にしてください。

AI エンジンのコンフィギュレーション

AI エンジンのコンフィギュレーションは、次のファイルで構成されます。

表 11: AI エンジンのコンフィギュレーション

| ファイル | 内容 |
|-----------------|---|
| AI エンジン NPI CDO | NPI を使用した AI エンジン グローバル コンフィギュレーション <ul style="list-style-type: none"> • PLL コンフィギュレーション • PM 初期化ノード コマンドを使用した AI エンジンのスキャン クリアおよびメモリ クリア |
| AI エンジン ELF | AI エンジン タイルのプログラムおよびデータ メモリ |

表 11: AI エンジンのコンフィギュレーション (続き)

| ファイル | 内容 |
|-------------|--|
| AI エンジン CDO | AI エンジン アレイ コンフィギュレーション <ul style="list-style-type: none"> プログラム メモリ コンフィギュレーション データ メモリ コンフィギュレーション DMA、ロック、ストリーム スイッチ コンフィギュレーション AI エンジン モジュール レジスタ コンフィギュレーション |

AI エンジン NPI CDO をロードする前に、PLM、LPD および PL (NPI ファイルの NoC コンフィギュレーションを含む) が完了していることを確認してください。また、PLM が AI エンジン データ メモリをクリアできるように、デザインの PMC から AI エンジンへの NoC パスを有効にしてください。

PLM ソフトウェアの詳細

このセクションでは、PLM の役割、アーキテクチャ、および実行の詳細について説明します。

PLM の役割

PLM は、bootROM がブートした後に PMC の PPU 上で動作を開始し、bootROM のポストブートからシステムの動作が終了するまで常にアクティブな状態を維持します。

PLM はシステムの初期化、Versal ACAP のサブシステム (APU、PL、および AI エンジンを含む) のブートおよびコンフィギュレーションを実行します。また、[第 9 章: セキュリティ](#) で説明するように、セキュア ブートの場合は PLM が認証と復号化を実行します。PLM はパワー マネージメント、パーシャル リコンフィギュレーション、エラー管理、サブシステム リスタート、およびヘルス モニターも実行します。

PLM の役割には、次のものがあります。

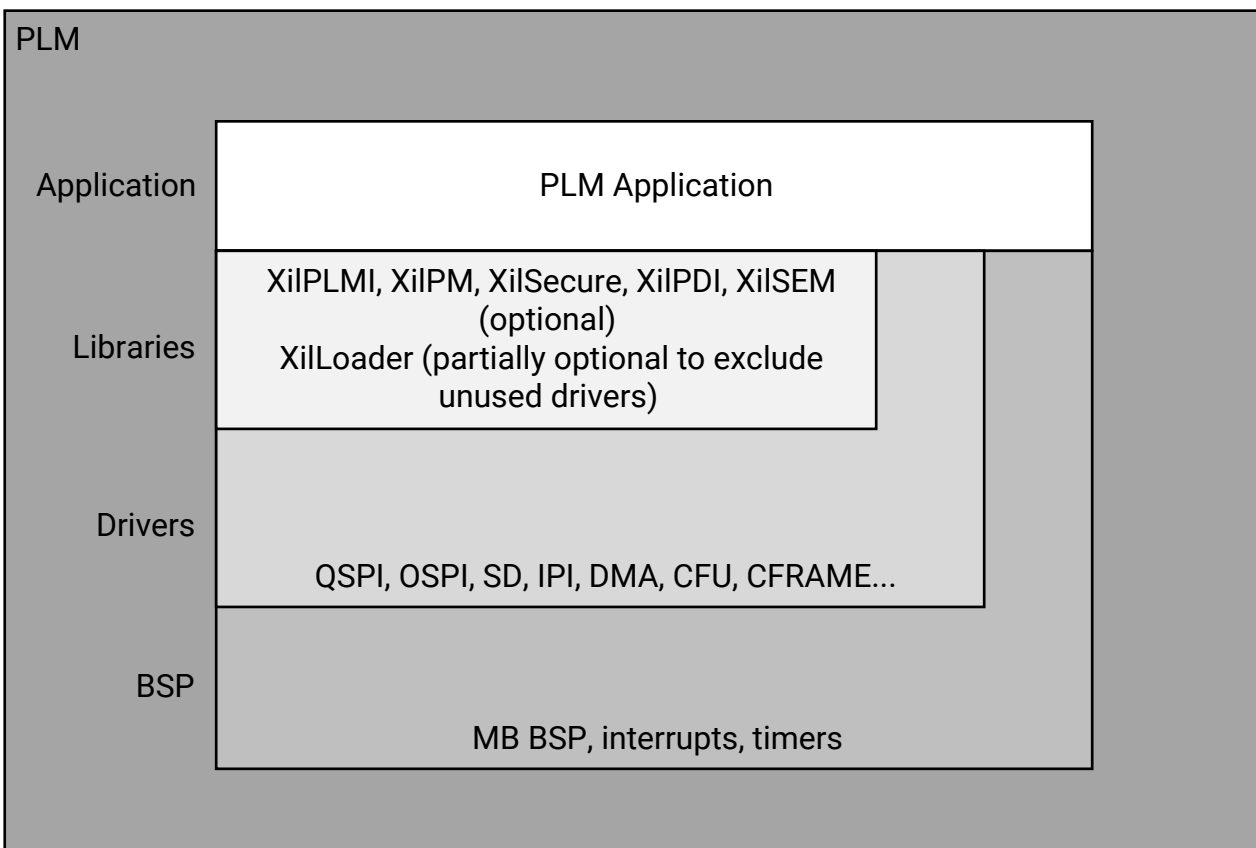
- セキュア/非セキュア ブート
 - システムの初期化
 - NoC の初期化、NPI (NoC Programming Interface)、DDR、および CPM (オプション) の設定
 - AI エンジンの設定
 - APU (Arm® Cortex-A72 プロセッサ) および RPU (Cortex-R5F プロセッサ) への PS イメージのロード
- プラットフォーム管理タスクには、次のものがあります。
 - Dynamic Function eXchange (DFX)
 - エラー管理
 - パワー マネージメント
 - サブシステムのリスタート
 - ステートの監視
 - ソフト エラーの緩和 (SEM)

PLM のアーキテクチャ

PLM はモジュール形式のサブシステム ベース構成とタスク ベースのサービスにより設計されています。各機能は、独立したモジュールとして設計されています。PLM アプリケーション層には、モジュール初期化およびスタートアップ タスクが含まれます。コンフィギュレーションで選択したモジュールに応じて、初期化されるモジュールが異なります。各モジュールは、PLM インターフェイス層を使用してイベント ハンドラー、IPI ハンドラー、CDO コマンド、およびスケジューラ イベントを登録できます。

PLM はブート デバイスから PDI を読み出し、PDI 内のペイロードを使用してシステムをコンフィギュレーションします。次の図に、PLM ELF ファイルに含まれるコンポーネントを示します。

図 26: PLM ELF のコンポーネント



X23875-042720

注記: モジュール性、およびほかのシステム ソフトウェア モジュールによる機能の再利用性を考慮して、次の PLM アプリケーション モジュールは別々のライブラリに配置されています。現在、これらのライブラリは内部での使用のみを想定しており、これらライブラリの API を直接呼び出すことはできません。

- XilPLMI
- XilPDI
- XilLoader

次の表に、各コンポーネントの説明を示します。

表 12: PLM ELF のコンポーネント

| ソフトウェア コンポーネント | 範囲 | 使用するライブラリ/ドライバー |
|----------------|---|-------------------------------------|
| PLM アプリケーション | <ul style="list-style-type: none"> スタートアップ イベントの登録 モジュールおよびデバッグ レベルを含めたコンフィギュレーション キューを使用したイベント スケジューリング | この表に示されているすべてのライブラリ。 |
| XilPLMI | <ul style="list-style-type: none"> PLM インターフェイス層 CDO ファイル解析用のインターフェイスを提供 セクション データに対するハンドラー登録用のインターフェイスを提供 IPI イベント登録用のインターフェイスを提供 割り込みイベント登録用のインターフェイスを提供 エラー応答の送信およびエラー ハンドラーの登録用のインターフェイスを提供 PL への CFI データ書き込みおよび PL からの CFI データ読み出し用のインターフェイスを提供 | BSP、PMC DMA、IPI、IOMODULE |
| XilLoader | <ul style="list-style-type: none"> ブート PDI およびサブシステム イメージのロードを実行 PDI からサブシステム イメージをロードするためのインターフェイスを提供 XilPLMI、XilSecure、XilPM およびフラッシュ ドライバーと接続するためのインターフェイス | SD/eMMC、QSPI、OSPI、USB、XilPLMI、XilPM |
| XilPM | サブシステム、MIO、クロック、電源、およびリセット ノードの設定を作成および管理するためのインターフェイスを提供。 | XilPLMI |
| XilSecure | <ul style="list-style-type: none"> セキュア モジュールと接続するためのインターフェイス。ザイリンクス イメージの認証と復号化のためのインターフェイスを提供 | PMC DMA |
| XilSEM | <ul style="list-style-type: none"> コンフィギュレーション RAM および NPI レジスタにおけるシングル イベント アップセット (SEU) イベントのためのハンドラーを提供 SEU 検出スキャンのためのハンドラーをスケジューリング | XilSecure、XilPLMI、CFRAME、CFU |

PLM ソフトウェア

PLM アプリケーションは、プロセッサの初期化、モジュールの初期化、およびスタートアップ タスクを実行します。

- プロセッサの初期化: スタックおよび対応するハードウェアの保護機能を初期化し、必要な割り込みと例外を有効にします。
- スタートアップ タスク: スタートアップ タスクには、モジュールの初期化、PMC CDO の実行、ブート PDI のロード、および事前に定義した場所でのユーザー フックの実行などがあります。

- モジュール: XilPLMI を使用してコマンド、割り込みハンドラー、およびスケジューラ タスクを登録するための、PLM に含まれるモジュール初期化関数で構成されます。モジュールは次のことを実行できます。
 - XilPLMI と登録されたモジュール ID を使用して各種コマンドを登録する。コマンドは PDI に含めることも、IPI メッセージとして送信することもできます。
 - PLMI を使用してタスクをスケジューリングする。
 - 任意の PLM 割り込みに対する割り込みハンドラーを登録する。
- PLM のフック: PLM アプリケーションには、ユーザーが独自のコードを追加できるように、事前に定義された場所にフック関数があります。事前に定義されたユーザー フックは、次の場所にあります。

表 13: PLM のフック

| 関数 | 説明 |
|-----------------------|----------------------------|
| XPlm_HookBeforePlmCdo | PMC CDO ファイルを処理する前に実行されます。 |
| XPlm_HookAfterPlmCdo | PMC CDO ファイルを処理した後に実行されます。 |
| XPlm_HookAfterBootPdi | ブート PDI をロードした後に実行されます。 |

- 例外処理: PPU で例外が発生すると、例外ハンドラーが呼び出されます。このハンドラーは、例外のログを記録します。

PLM の実行フロー

PLM はタスクに基づいて実行されます。プロセッサおよびプログラマブル インターバル タイマー (MB 内部) を初期化した後、最初のスタートアップ タスクが PLM タスク キューに追加されます。スタートアップ タスクには、モジュールの初期化、PMC CDO の実行、ブート PDI のロード、および事前に定義した場所でのユーザー フックの実行などが含まれます。スタートアップ イベントの後、PLM はサービス モード (PLM がスリープに移行し、イベントを待機する状態) に入ります。割り込みによってウェークアップすると、PLM は割り込みコンテキストに入り、割り込みを処理した後、タスク キューに戻ってタスクの有無を確認します。タスク キューが空の場合はスリープに移行します。

PLM では次のイベント シーケンスが発生します。

- プロセッサを初期化し、割り込みハンドラーを登録し、割り込みを有効にする
- スタートアップ タスクを実行する
 - モジュールを初期化する
 - XilPLMI、XilPM、および XilLoader などのモジュールを初期化する
 - すべてのモジュールに関して: CDO コマンドおよび割り込みハンドラーを登録する
 - PMC RAM に格納された PMC CDO を処理する
 - ブート PDI に含まれるその他のイメージをロードする
 - ユーザー フックを実行する
- タスク ディスパッチ ループ (イベントを待機)
 - タスク キューに追加されたタスクを実行する

PLM エラー

PDI のロード中にエラーが検出されると、PLM はエラー コードを PLM_FW_ERROR レジスタに書き込みます。

または、JTAG エラー ステータス コマンドから PLM エラーを読み出すこともできます。エラーは、次のフォーマットで記録されます。

```
Error code: 0xXXXXXXXX
```

- XXXX: メジャー エラー コード。PLM/XilLoader/XPLMI のエラー コードは、`xplmi_status.h` に定義されています。
- YYYY: マイナー エラー コード。各モジュールで定義されたライブラリ/ドライバのエラー コードです。

PLM エラー コード

次の表に、PLMI、PLM、XilLoader、および XilSecure のメジャー エラー コード一覧を示します。

表 14: PLM エラー コード

| 値 | 説明 |
|-------|--|
| 0x0 | XPLM_SUCCESS: 成功 |
| 0x1 | XPLM_FAILURE: 一部の関数において内部で使用。 |
| 0x2 | XPLMI_TASK_INPROGRESS: タスクが進行中であることを示すために内部で使用。 |
| 0x100 | XPLMI_ERR_DMA_LOOKUP: DMA ドライバーのルックアップ エラー。 |
| 0x101 | XPLMI_ERR_DMA_CFG: DMA ドライバーのコンフィギュレーション エラー。 |
| 0x102 | XPLMI_ERR_DMA_SELFTEST: DMA のセルフ テスト エラー。 DMA がリセット中に PLM が DMA を初期化しようとした場合に発生します。 |
| 0x103 | XPLMI_ERR_IOMOD_INIT: IOModule ドライバーのルックアップ エラー。 |
| 0x104 | XPLMI_ERR_IOMOD_START: IOModule ドライバーのスタートアップ エラー。 |
| 0x105 | XPLMI_ERR_IOMOD_CONNECT: IOModule ドライバーの接続エラー。 |
| 0x106 | XPLMI_ERR_MODULE_MAX: PLMI モジュールが登録されていない場合のエラー。 XilPLMI が無効な CDO CMD を処理した場合に発生します。 |
| 0x107 | XPLMI_ERR_CMD_APIID: XilPLMI が有効なモジュールと未登録の CMD ID を処理した場合のエラー。 |
| 0x108 | XPLMI_ERR_CMD_HANDLER_NULL: CDO CMD に対するコマンド ハンドラーがモジュールによって登録されていない場合のエラー。 |
| 0x109 | XPLMI_ERR_CMD_HANDLER: CDO CMD ハンドラーから返されるエラー。 CMD から返されるエラーについては、PLM マイナー コードを確認してください。 |
| 0x10A | XPLMI_ERR_RESUME_HANDLER: CDO CMD レジューム ハンドラーから返されるエラー。 CMD から返されるエラーについては、PLM マイナー コードを確認してください。 |
| 0x10B | XPLMI_ERR_CDO_HDR_ID: CDO ヘッダーに有効な CDO ヘッダー ID がない場合のエラー。 異なるパーティション タイプを CDO として処理した場合に発生します。 |
| 0x10C | XPLMI_ERR_CDO_CHECKSUM: CDO ヘッダー チェックサムが誤っている場合のエラー。 CDO ヘッダーが発生している場合に発生します。 |
| 0x10D | XPLMI_ERR_UART_DEV_PM_REQ: UART に対する XilPM のデバイス要求のエラー。 PM エラー コードは PLM マイナー コードにあります。 |
| 0x10E | XPLMI_ERR_UART_LOOKUP: UART ドライバーのルックアップ エラー。 |
| 0x10F | XPLMI_ERR_UART_CFG: UART ドライバーのコンフィギュレーション エラー。 |
| 0x110 | XPLMI_ERR_SSI_MASTER_SYNC: SSI テクノロジのスレーブとマスターの同期エラー。 |
| 0x111 | XPLMI_ERR_SSI_SLAVE_SYNC: SSI テクノロジのマスターがスレーブの同期ポイントを待機中にタイムアウトになった場合のエラー。 |
| 0x112 | XPLMI_ERR_INVALID_LOG_LEVEL: ログ コマンドで無効なログ レベルを受信した場合のエラー。 |

表 14: PLM エラー コード (続き)

| 値 | 説明 |
|-------|--|
| 0x113 | XPLMI_ERR_INVALID_LOG_BUF_ADDR: ログ コマンドで無効なログ バッファ アドレスを受信した場合のエラー。 |
| 0x114 | XPLMI_ERR_INVALID_LOG_BUF_LEN: ログ コマンドで無効なログ バッファ 長を受信した場合のエラー。 |
| 0x115 | XPLMI_ERR_IPI_CMD: IPI を介したコマンド実行がサポートされない場合のエラー。 |
| 0x116 | XPLMI_ERR_REGISTER_IOMOD_HANDLER: IOModule ハンドラーを登録する際のエラー。 |
| 0x117 | XPLMI_ERR_WDT_PERIODICITY: SetWDT コマンドの Periodicity パラメーターが無効。 |
| 0x118 | XPLMI_ERR_WDT_NODE_ID: SetWDT コマンドの Node ID パラメーターが無効。 |
| 0x119 | XPLMI_ERR_WDT_LPD_NOT_INITIALIZED: WDT に LPD MIO を使用しているが、LPD が初期化されていない。 |
| 0x200 | XPLM_ERR_TASK_CREATE: タスクを作成できなかった場合のエラー。 最大数のタスクが作成されている場合に発生します。 |
| 0x201 | XPLM_ERR_PM_MOD: PM モジュール初期化のエラー。 |
| 0x202 | XPLM_ERR_LPD_MOD: LPD モジュール初期化のエラー。 |
| 0x203 | XPLM_ERR_EXCEPTION: PLM 初期化中に例外が発生。 有効な場合、EAR と ESR が UART コンソールに出力されます。 |
| 0x204 | XPLM_ERR_NPLL_LOCK: マスター SLR デバイスに対して NOC PLL をロックできない。 |
| 0x205 | XPLM_ERR_STL_MOD: STL モジュール初期化のエラー。 |
| 0x300 | XLOADER_UNSUPPORTED_BOOT_MODE: ブート モードがサポートされない場合のエラー。 無効なブート モードを選択した場合、またはブート モード ベリフェラルを CIPS Wizard で選択していない場合に発生します。 |
| 0x302 | XLOADER_ERR_IMGHDR_TBL: このエラーは、いくつかの条件で発生します。 <ul style="list-style-type: none"> イメージ ヘッダー テーブルのチェックサムが誤っている場合。 PLM がイメージ ヘッダー テーブルを読み出せない場合。 |
| 0x303 | XLOADER_ERR_IMGHDR: イメージ ヘッダー チェックサム エラー。 |
| 0x304 | XLOADER_ERR_PRTNHDR: パーティション ヘッダー チェックサム エラー。 |
| 0x305 | XLOADER_ERR_WAKEUP_A72_0: ハンドオフ時の A72-0 ウェークアップ エラー。 PM エラー コードは PLM マイナー コードを確認してください。 |
| 0x306 | XLOADER_ERR_WAKEUP_A72_1: ハンドオフ時の A72-1 ウェークアップ エラー。 PM エラー コードは PLM マイナー コードを確認してください。 |
| 0x307 | XLOADER_ERR_WAKEUP_R5_0: ハンドオフ時の R5-0 ウェークアップ エラー。 PM エラー コードは PLM マイナー コードを確認してください。 |
| 0x308 | XLOADER_ERR_WAKEUP_R5_1: ハンドオフ時の R5-1 ウェークアップ エラー。 PM エラー コードは PLM マイナー コードを確認してください。 |
| 0x309 | XLOADER_ERR_WAKEUP_R5_L: ハンドオフ時の R5-L ウェークアップ エラー。 PM エラー コードは PLM マイナー コードを確認してください。 |
| 0x30A | XLOADER_ERR_WAKEUP_PSM: ハンドオフ時の PSM ウェークアップ エラー。 PM エラー コードは PLM マイナー コードを確認してください。 |
| 0x30B | XLOADER_ERR_PL_NOT_PWRUP: PL のパワーアップ エラー。 |
| 0x30C | XLOADER_ERR_UNSUPPORTED_OSPI: OSPI フラッシュがサポートされない場合のエラー。 |
| 0x30D | XLOADER_ERR_UNSUPPORTED_OSPI_SIZE: OSPI フラッシュ サイズがサポートされない場合のエラー。 |
| 0x30E | XLOADER_ERR_OSPI_INIT: OSPI ドライバーのルックアップ エラー。 OSPI を CIPS で選択していない場合に発生します。 |

表 14: PLM エラー コード (続き)

| 値 | 説明 |
|-------|---|
| 0x30F | XLOADER_ERR_OSPI_CFG: OSPI ドライバーのコンフィギュレーション エラー。 |
| 0x310 | XLOADER_ERR_OSPI_SEL_FLASH: OSPI ドライバーがフラッシュを選択できない場合のエラー。 OSPI ドライバー エラー コードはマイナー コードを確認してください。 |
| 0x311 | XLOADER_ERR_OSPI_READID: OSPI ReadID のエラー。 |
| 0x312 | XLOADER_ERR_OSPI_READ: OSPI ドライバーの読み出しエラー。 OSPI ドライバー エラー コードはマイナー コードを確認してください。 |
| 0x313 | XLOADER_ERR_OSPI_4BMODE: OSPI が 4B モードを開始/終了できない場合のエラー。 |
| 0x314 | XLOADER_ERR_QSPI_READ_ID: QSPI 読み出しエラー。 |
| 0x315 | XLOADER_ERR_UNSUPPORTED_QSPI: QSPI フラッシュがサポートされない場合のエラー。 |
| 0x316 | XLOADER_ERR_QSPI_INIT: QSPI ドライバーのルックアップまたはコンフィギュレーション エラー。 |
| 0x317 | XLOADER_ERR_QSPI_MANUAL_START: QSPI ドライバーのマニュアル スタート エラー。 |
| 0x318 | XLOADER_ERR_QSPI_PRESCALER_CLK: QSPI ドライバーのプリスケラ設定エラー。 |
| 0x319 | XLOADER_ERR_QSPI_CONNECTION: シングル、デュアル、またはスタックド以外の無効な QSPI 接続がリストされた場合のエラー。 |
| 0x31A | XLOADER_ERR_QSPI_READ: QSPI ドライバーの読み出しエラー。 |
| 0x31B | XLOADER_ERR_QSPI_LENGTH: QSPI 読み出し長さがフラッシュ サイズより大きい場合のエラー。 |
| 0x31C | XLOADER_ERR_SD_INIT: SD のマウント エラー。 |
| 0x31D | XLOADER_ERR_SD_F_OPEN: SD のファイル オープン エラー。 ファイルが存在しない場合、または SD からの読み出しができなかった場合に発生します。ファイル システム エラー コードは PLM マイナー コードにあります。 |
| 0x31E | XLOADER_ERR_SD_F_LSEEK: SD カードから読み出し中の f_seek のエラー。 |
| 0x31F | XLOADER_ERR_SD_F_READ: SD カードからの読み出し中のエラー。 |
| 0x320 | XLOADER_ERR_IMG_ID_NOT_FOUND: イメージを再ロードする際にイメージ ID がサブシステムに見つからない場合のエラー。 |
| 0x321 | XLOADER_ERR_TCM_ADDR_OUTOF_RANGE: TCM へのロード時のアドレス範囲外エラー。 |
| 0x322 | XLOADER_ERR_CFRAME_LOOKUP: CFRAME ドライバーのルックアップ エラー。 |
| 0x323 | XLOADER_ERR_CFRAME_CFG: CFRAME ドライバーのコンフィギュレーション エラー。 |
| 0x324 | XLOADER_ERR_UNSUPPORTED_SEC_BOOT_MODE: セカンダリ ブート モードがサポートされない場合のエラー。 |
| 0x325 | XLOADER_ERR_SECURE_METAHDR: メタ ヘッダーのセキュア バリデーション エラー。 |
| 0x326 | XLOADER_ERR_GEN_IDCODE: IDCODE の不一致によるエラー。 |
| 0x327 | XLOADER_ERR_USB_LOOKUP: USB のルックアップ エラー。 |
| 0x328 | XLOADER_ERR_USB_CFG: USB のコンフィギュレーション 初期化エラー。 |
| 0x329 | XLOADER_ERR_USB_START: USB 開始エラー。 |
| 0x32A | XLOADER_ERR_DFU_DWNLD: PDI ダウンロード エラー。 |
| 0x32B | XLOADER_ERR_DEFERRED_CDO_PROCESS: CDO 処理中に発生したエラーが、CDO 全体の処理が完了するまで延期された。 |
| 0x32C | XLOADER_ERR_SD_LOOKUP: SD のルックアップ エラー。 |
| 0x32D | XLOADER_ERR_SD_CFG: SD のコンフィギュレーション エラー。 |
| 0x32E | XLOADER_ERR_SD_CARD_INIT: SD カードの初期化エラー。 |
| 0x32F | XLOADER_ERR_MMC_PART_CONFIG: RAW ブート モードで MMC がユーザー領域に切り替わる際に発生したエラー。 |

表 14: PLM エラー コード (続き)

| 値 | 説明 |
|-------|---|
| 0x330 | XLOADER_ERR_SEM_STOP_SCAN: XilSEM スキャン停止時のエラー。 |
| 0x331 | XLOADER_ERR_SEM_CFR_INIT: XilSEM スキャン開始時のエラー。 |
| 0x332 | XLOADER_ERR_DELAY_ATTRB: delay_handoff 属性と copy 属性の両方を指定した場合のイメージのエラー。 |
| 0x333 | XLOADER_ERR_NUM_HANDOFF_CPUS: CPU の数が最大数を越えた場合のエラー。 |
| 0x334 | XLOADER_ERR_OSPI_CONN_MODE: OSPI モードがサポートされない場合のエラー。 |
| 0x335 | XLOADER_ERR_OSPI_SEL_FLASH_CS1: OSPI ドライバーがフラッシュ CS1 を選択できない場合のエラー。OSPI ドライバー エラー コードはマイナー コードを確認してください。 |
| 0x336 | XLOADER_ERR_OSPI_SDR_NON_PHY: OSPI ドライバーがコントローラーを SDR NON PHY モードに設定できない場合のエラー。 |
| 0x337 | XLOADER_ERR_OSPI_COPY_OVERFLOW: OSPI コピーのソース アドレスがフラッシュ サイズを超過した場合のエラー。 |
| 0x338 | XLOADER_ERR_SD_F_CLOSE: SD ファイル システム モードでのファイル クローズ エラー。 |
| 0x339 | XLOADER_ERR_SD_UMOUNT: ファイル システムのマウント解除エラー。 |
| 0x33A | XLOADER_ERR_DMA_XFER: DMA 転送エラー。 |
| 0x33B | XLOADER_ERR_DMA_XFER_SD_RAW: SD RAW の DMA 転送エラー。 |
| 0x33C | XLOADER_ERR_CONFIG_SUBSYSTEM: サブシステム コンフィギュレーション中のエラー。 |
| 0x33D | XLOADER_ERR_COPY_TO_MEM: copy 属性が有効な場合にイメージを DDR にコピーする際のエラー。 |
| 0x33E | XLOADER_ERR_DELAY_LOAD: イメージの delay_load 属性が有効で、ブート ソースが SMAP、SBI、PCIE または JTAG の場合、SBI バッファの容量を空けるためにイメージが PMC RAM にコピーされます。このような PMC RAM へのコピーでエラーが発生すると、このエラー コードで示されます。 |
| 0x33F | XLOADER_ERR_ADD_TASK_SCHEDULER: タスクをスケジューラに追加する際のエラー。 |
| 0x340 | XLOADER_ERR_SD_MAX_BOOT_FILES_LIMIT: ブート可能ファイルの検索時に上限値に達すると返されるエラー コード。 |
| 0x341 | XLOADER_ERR_UNSUPPORTED_QSPI_FLASH_SIZE: QSPI フラッシュ サイズがサポートされない場合のエラー。 |
| 0x342 | XLOADER_ERR_PM_DEV_PSM_PROC: PM_DEV_PSM_PROC に対する XPM のデバイス要求のエラー。 |
| 0x343 | XLOADER_ERR_PM_DEV_IOCTL_RPU0_SPLIT: スプリット モードの RPU0_0 に対する XPM のデバイス Ioctl のエラー。 |
| 0x344 | XLOADER_ERR_PM_DEV_IOCTL_RPU1_SPLIT: スプリット モードの RPU0_1 に対する XPM のデバイス Ioctl のエラー。 |
| 0x345 | XLOADER_ERR_PM_DEV_IOCTL_RPU0_LOCKSTEP: ロックステップ モードの RPU0_0 に対する XPM のデバイス Ioctl のエラー。 |
| 0x346 | XLOADER_ERR_PM_DEV_IOCTL_RPU1_LOCKSTEP: ロックステップ モードの RPU0_1 に対する XPM のデバイス Ioctl のエラー。 |
| 0x347 | XLOADER_ERR_PM_DEV_TCM_0_A: PM_DEV_TCM_0_A に対する XPM のデバイス要求のエラー。 |
| 0x348 | XLOADER_ERR_PM_DEV_TCM_0_B: PM_DEV_TCM_0_B に対する XPM のデバイス要求のエラー。 |
| 0x349 | XLOADER_ERR_PM_DEV_TCM_1_A: PM_DEV_TCM_1_A に対する XPM のデバイス要求のエラー。 |
| 0x34A | XLOADER_ERR_PM_DEV_TCM_1_B: PM_DEV_TCM_1_B に対する XPM のデバイス要求のエラー。 |
| 0x34B | XLOADER_ERR_PM_DEV_DDR_0: PM_DEV_DDR_0 に対する XPM のデバイス要求のエラー。 |
| 0x34C | XLOADER_ERR_PM_DEV_QSPI: PM_DEV_QSPI に対する XPM のデバイス要求のエラー。 |
| 0x34D | XLOADER_ERR_PM_DEV_SDIO_0: PM_DEV_SDIO_0 に対する XPM のデバイス要求のエラー。 |
| 0x34E | XLOADER_ERR_PM_DEV_SDIO_1: PM_DEV_SDIO_1 に対する XPM のデバイス要求のエラー。 |
| 0x34F | XLOADER_ERR_PM_DEV_USB_0: PM_DEV_USB_0 に対する XPM のデバイス要求のエラー。 |

表 14: PLM エラー コード (続き)

| 値 | 説明 |
|-------|--|
| 0x350 | XLOADER_ERR_PM_DEV_OSPI: PM_DEV_OSPI に対する XPM のデバイス要求のエラー。 |
| 0x600 | XLOADER_ERR_INIT_GET_DMA: 初期化時に DMA インスタンスを取得できなかった場合のエラー。 |
| 0x601 | XLOADER_ERR_INIT_INVALID_CHECKSUM_TYPE: SHA3 チェックサムのみがサポートされます。 |
| 0x602 | XLOADER_ERR_INIT_CHECKSUM_COPY_FAIL: フラッシュ デバイスからチェックサムをコピーする際のエラー。 |
| 0x603 | XLOADER_ERR_INIT_AC_COPY_FAIL: フラッシュ デバイスから AC をコピーする際のエラー。 |
| 0x604 | XLOADER_ERR_INIT_CHECKSUM_INVLD_WITH_AUTHDEC: 認証と暗号化が有効な状態でチェックサムを有効にした場合のエラー。 |
| 0x605 | XLOADER_ERR_DMA_TRANSFER: コピー中の DMA 転送エラー。 |
| 0x606 | XLOADER_ERR_IHT_AUTH_DISABLED: イメージ ヘッダー テーブルに対する認証が有効でない。 |
| 0x607 | XLOADER_ERR_IHT_GET_DMA: IHT 認証における DMA インスタンス取得エラー。 |
| 0x608 | XLOADER_ERR_IHT_COPY_FAIL: フラッシュ デバイスから IHT AC をコピーする際のエラー。 |
| 0x609 | XLOADER_ERR_IHT_HASH_CALC_FAIL: IHT 認証用のハッシュ計算エラー。 |
| 0x60A | XLOADER_ERR_IHT_AUTH_FAIL: IHT の認証エラー。 |
| 0x60B | XLOADER_ERR_HDR_COPY_FAIL: フラッシュ デバイスから IH/PH をコピーする際のエラー。 |
| 0x60C | XLOADER_ERR_HDR_AES_OP_FAIL: AES 初期化、復号化の初期化、またはキー選択のエラー。 |
| 0x60D | XLOADER_ERR_HDR_DEC_FAIL: イメージ ヘッダー/パーティションの復号化エラー。 |
| 0x60E | XLOADER_ERR_HDR_AUTH_FAIL: イメージ ヘッダー/パーティションの認証エラー。 |
| 0x60F | XLOADER_ERR_HDR_NOT_SECURE: イメージ ヘッダー/パーティションに対して認証も暗号化も有効でない。 |
| 0x610 | XLOADER_ERR_HDR_GET_DMA: イメージ ヘッダー/パーティション認証/復号化用の DMA インスタンス取得エラー。 |
| 0x611 | XLOADER_ERR_HDR_HASH_CALC_FAIL: イメージ ヘッダー/パーティション認証用のハッシュ計算エラー。 |
| 0x612 | XLOADER_ERR_HDR_NOT_ENCRYPTED: イメージ ヘッダー/パーティション ヘッダーが暗号化されていない。 |
| 0x613 | XLOADER_ERR_HDR_AUTH_DISABLED: イメージ ヘッダー/パーティション ヘッダーの認証が無効。 |
| 0x614 | XLOADER_ERR_SEC_IH_READ_VERIFY_FAIL: イメージ ヘッダー読み出しおよびチェックサム検証のエラー。 |
| 0x615 | XLOADER_ERR_SEC_PH_READ_VERIFY_FAIL: パーティション ヘッダー読み出しおよびチェックサム検証のエラー。 |
| 0x616 | XLOADER_ERR_PRTN_HASH_CALC_FAIL: パーティション認証のハッシュ計算エラー。 |
| 0x617 | XLOADER_ERR_PRTN_AUTH_FAIL: パーティション認証エラー。 |
| 0x618 | XLOADER_ERR_PRTN_HASH_COMPARE_FAIL: パーティション ハッシュ照合エラー。 |
| 0x619 | XLOADER_ERR_PRTN_DECRYPT_FAIL: パーティション復号化エラー。 |
| 0x61A | XLOADER_ERR_AHWROT_EFUSE_AUTH_COMPULSORY: PPK をプログラムしたが eFUSE 認証が無効。 |
| 0x61B | XLOADER_ERR_AHWROT_BH_AUTH_NOT_ALLOWED: PPK がプログラムされており、BH 認証が有効。 |
| 0x61C | XLOADER_ERR_AUTH_EN_PPK_HASH_ZERO: PPK がプログラムされておらず、認証が有効。 |
| 0x61D | XLOADER_ERR_SHWROT_ENC_COMPULSORY: 暗号化が無効。 |
| 0x61E | XLOADER_ERR_KAT_FAILED: 既知解テスト (KAT) エラー。 |
| 0x61F | XLOADER_ERR_DATA_COPY_FAIL: 内部メモリへのデータ コピーのエラー。 |
| 0x620 | XLOADER_ERR_METAHDR_LEN_OVERFLOW: 合計サイズが Metahdr の長さを超えている場合のエラー。 |
| 0x621 | XLOADER_ERR_AUTH_JTAG_EFUSE_AUTH_COMPULSORY: PPK がプログラムされていない場合の JTAG 認証エラー。 |

表 14: PLM エラー コード (続き)

| 値 | 説明 |
|--------|--|
| 0x622 | XLOADER_ERR_AUTH_JTAG_DISABLED: JTAG 認証無効 eFUSE ビットがセットされている。 |
| 0x623 | XLOADER_ERR_AUTH_JTAG_PPK_VERIFY_FAIL: PPK 検証時の JTAG 認証エラー。 |
| 0x624 | XLOADER_ERR_AUTH_JTAG_SIGN_VERIFY_FAIL: シグネチャ検証時の JTAG 認証エラー。 |
| 0x625 | XLOADER_ERR_AUTH_JTAG_EXCEED_ATTEMPTS: 複数回の JTAG 認証エラー。 |
| 0x626 | XLOADER_ERR_AUTH_JTAG_GET_DMA: JTAG 認証における DMA インスタンス取得エラー。 |
| 0x627 | XLOADER_ERR_AUTH_JTAG_HASH_CALCULATION_FAIL: シグネチャ検証前のハッシュ値計算エラー。 |
| 0x628 | XLOADER_ERR_AUTH_JTAG_DMA_XFR: DMA 転送による認証 JTAG データ取得エラー。 |
| 0x2XYZ | <ul style="list-style-type: none"> 0x2 は CDO コマンドのエラーを示す。 X はコマンド モジュールを示す。 <ul style="list-style-type: none"> 1 - PLM 2 - PM 3 - XilSEM 7 - ローダー 8 - エラー YZ は CDO コマンドのハンドラー ID を示す。 |

PLM インターフェイス (XilPLMI)

PLM インターフェイス (XilPLMI) は、PLM メイン アプリケーションおよびその他の PLM モジュール用の低レベル インターフェイス層です。XilPLMI は PLM で動作する各モジュールが必要とする一般的な機能を提供します。新規モジュールは、サポートされているコマンド ハンドラーを使用して登録できます。これらのコマンド ハンドラーは、ほかのモジュールまたはサブシステムから受信した要求に基づいて実行されます。

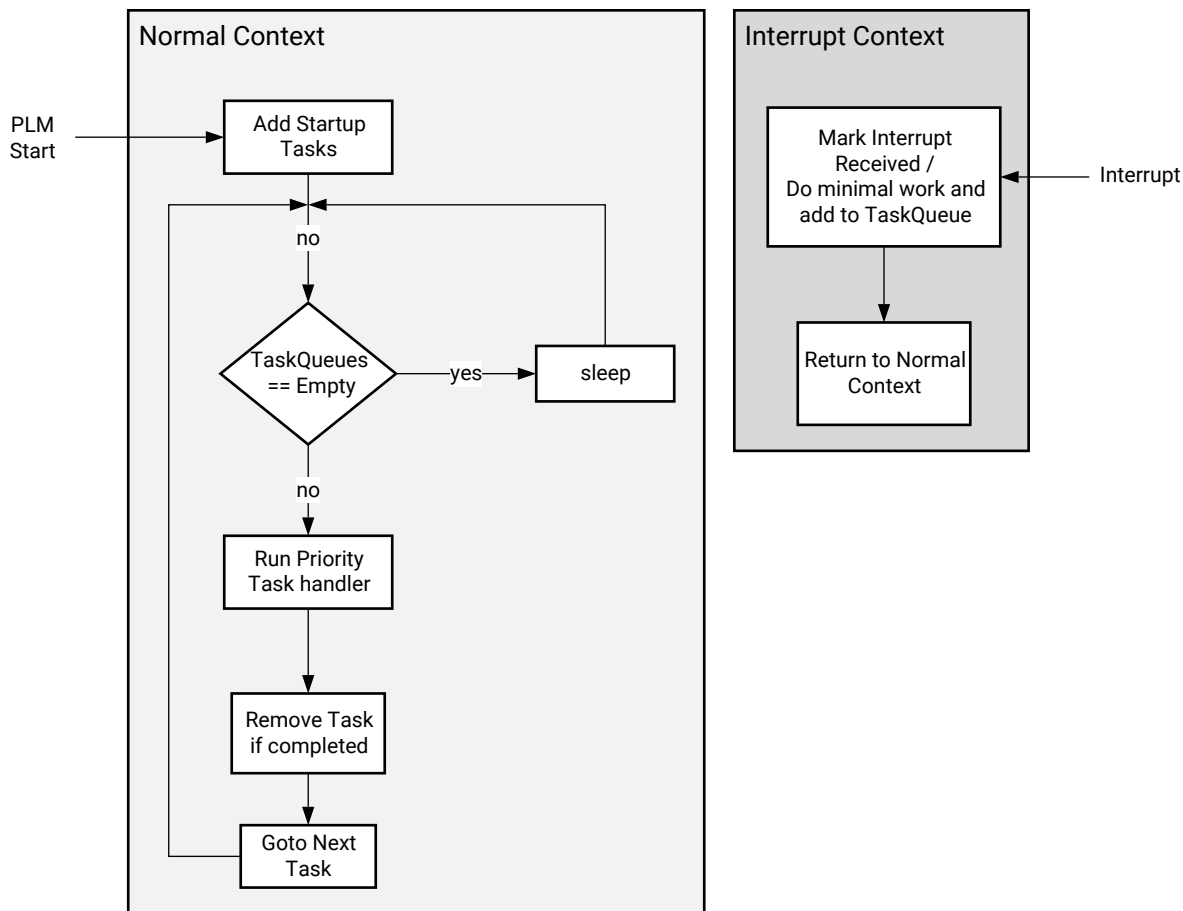
XilPLMI には、任意の CDO コマンドを解析して実行する CDO パーサーも含まれます。XilPLMI は汎用モジュールを実装しており、この中にはほかのすべてのモジュールが使用する汎用コマンドが用意されています。XilPLMI レイヤーは次のものを提供します。

- CDO ファイル解析用のインターフェイス
- 汎用 PLM CDO コマンドのインプリメンテーション
- CDO または IPI に含めることのできるコマンドのハンドラーを登録するためのインターフェイス
- エラー イベントに対するエラー アクションを設定し、通知を登録するためのインターフェイス
- タイマー イベントをスケジュールするためのインターフェイス
- デバッグ プリント レベルと共通ユーティリティ
- タスク ディスパッチャー ループ: XilPLMI は、リアルタイム動作を確保するために、実行開始から終了まで中断されることのない (Run-to-Completion)、時間制限付きのシンプルで優先タスク ループ モデルを採用しています。メイン プログラムは、タスク キューから次のタスクを参照し、そのタスクを実行するための関数を呼び出すだけの簡単なループです。

このモデルは、すべてのタスクが実行完了まで中断されないため、クリティカル領域もなく、ロックが不要なため、非常にシンプルです。PLM 内のコードからグローバル データにアクセスする際も、別のタスクがデータの更新中である可能性を考慮する必要がありません。

この Run-to-Completion モデルは、特定のタスクを最大許容時間 (GTL) を超えて実行しようとする複雑になります。その場合は、タスクを複数のイベントに分割してください。

図 27: タスク ディスパッチ ループ



X23876-042720

- モジュール: XilPLMI は、各モジュールが PLM にコマンドを登録するためのインターフェイス層を提供します。コマンドは、CDO または IPI から取得できます。
- コンフィギュレーション可能パラメーター:
 - モジュール: コンフィギュレーションにより、必要なモジュールやブート ドライバーのみを含めることができます。
 - デバッグ出力: コンフィギュレーションにより、複数のレベルの print 文を含めることができます。詳細は、[PLM のビルド フラグ](#) を参照してください。
- CDO: CDO ファイルには、ツールで生成された CDO コマンド形式のコンフィギュレーション情報が格納されています。CDO をサポートしているモジュールは、モジュール初期化フェーズで自分自身を PLM に登録します。XilPLMI は、CDO ファイルを解析してコマンドとそのペイロードを各モジュールに伝搬するための API を提供します。
- IPI 処理: PLM は、PLM と Versal デバイス上のほかのプロセッサとの間でメッセージを交換できるように、IPI 割り込みを処理します。IPI 経由で送信されるデータは、CDO フォーマットに従います。

- スケジューラ: これは、周期的タスクの実行をサポートするシンプルなタイマー ベースのファンクション スケジューラです。XilSEM などのモジュールが SEU 検出スキャンなどの周期的タスクをサポートするためにはスケジューラが必要です。

注記: スケジューラの精度は、PMC I/O クロック精度の影響を受けます。

- PLM ウォッチドッグ タイマー: PLM には、PLM の健全性を周期的に更新するためのフレームワークがあります。この目的のため、外部ウォッチドッグ タイマー (WDT) を使用する場合は、PLM が MIO (Multiplexed I/O) をトグルします。PMC MIO の使用が推奨されますが、LPD MIO を使用する場合は、PS がパワーダウンすると WDT が無効になります。

PLM WDT を有効にするには、SetWdt CDO コマンドにパラメーター MIO PIN と periodicity を指定して実行します。SetWdt CDO コマンドを実行する前に、MIO を GPIO として設定し、対応する PMC/LPD CDO をロードしておく必要があります。PDI のロードはコンフィギュレーション モードと見なされ、2020.2 リリースではコンフィギュレーション モード中は WDT はサポートされません。これ以外に、ブート、パーシャル PDI のロード、およびサブシステム リスタート/レジュームもコンフィギュレーション モードと見なされます。スケジューラの最小周期は 15ms です。

実装は次のとおりです。

- 通常コンテキスト:
 - PLM は、タスクとタスクの間で毎回、ALIVE を示す変数をセットします。
 - PDI のロード中、PLM のモードはコンフィギュレーション モードに設定されます。PDI のロードが完了すると、動作中モードにリセットされます。
 - LPD MIO を使用する場合、LPD シャットダウン中は WDT が無効になります。
 - いずれかのタスクの実行時間が WDT の周期より長い場合、ALIVE ビットがセットされ、外部 WDT がデバイスをリセットできます。
- 割り込みコンテキスト:
 - 現在のスケジューラの周期は 10ms です。つまり、10ms ごとの周期で PLM がタイマー割り込みを受信し、タスクをスケジューリングします。
 - WDT ハンドラーがスケジューラ内で呼び出されます。
 - WDT を有効にすると、設定した周期に基づいて WDT ハンドラーが次の動作を周期的に実行します (たとえば周期 100ms の場合、次のタスクが周期タイムアウトの約 10ms 前まで実行されます)。
 - コンフィギュレーション モードでは、PLM の ALIVE ステータスにかかわらず WDT ハンドラーが MIO ピンをトグルします。
 - 動作中モードでは、PLM の ALIVE ステータスがセットされている場合のみ WDT ハンドラーは MIO ピンをトグルし、PLM の ALIVE ステータスをクリアします。

| コマンド: Set PLM WDT | | | |
|----------------------------------|--------------|---------------------------------|--------------------|
| 予約 [31:24]=0 | 長さ [23:16]=2 | PLM=1 | CMD_SET_PLM_WDT=22 |
| NODE Idx - PMC MIO ピン/LPD MIO ピン | | | |
| [31:16] 予約 | | [15:0] 周期 (単位 ms)。デフォルト = 100ms | |

PLM イベント ログ機能

イベント ログ機能には、大きく分けて次のものがあります。

- PLM ターミナル出力のログ機能

- トレース イベントのログ機能

PLM ターミナル出力のログ機能

PLM は、PLM ターミナル出力のログをメモリに記録する機能をサポートしています。この機能は、UART が存在しない場合のデバッグに役立ちます。このログは、ユーザーが有効にしたログ レベルに基づいて記録されます。各出力メッセージのログには、PLM タイムスタンプが付けられます。ログ作成の一部として、次の機能がサポートされます。

- ログ出力レベルの設定: PLM をビルドする際に、使用するログ レベルを決定できます。PLM に基づき、ELF ファイルが生成されます。PLM に IPI コマンドを送信してログ レベルを変更できる機能があります。IPI コマンドにより、ログ レベルを上下に変更できます。
- デバッグ ログ バッファ メモリの設定: デフォルトでは、ログ バッファは 16K の PMC RAM 領域で、PLM は出力ログを PMC RAM メモリに記録します。この機能を使用すると、デバッグ ログ メモリを別のメモリに変更できます。
- デバッグ ログ バッファの取得: 出力ログを任意のメモリに取得できます。
- デバッグ ログ バッファ情報の取得: PLM 出力ログの情報を取得できます。
- ターミナル出力の取得: 次のコマンドを使用して、ターミナル出力を取得できます。`xsdb > plm log`

トレース イベントのログ機能

PLM はトレース イベントのログをメモリに記録する機能をサポートしています。このメモリは、PLM 出力のログに使用するメモリとは異なります。現在、PLM は PDI ロードのトレース イベントをタイムスタンプ付きでログに記録する機能のみをサポートしています。トレース機能の一部として、次の機能が PLM によってサポートされます。

- トレース ログ バッファ メモリの設定: デフォルトでは、PLM はトレース イベントのログを PMC RAM メモリに記録します。この機能を使用すると、トレース ログ メモリを別のメモリに変更できます。
- トレース ログ バッファの取得: トレース イベント ログを任意のメモリに取得します。
- トレース ログ バッファ情報の取得: トレース イベント ログの情報を取得します。

トレース イベントのログ機能の使用例は、[イベント トレース ログ コマンドの例](#) を参照してください。

トレース イベント ログのフォーマット

表 15: トレース イベント ログのフォーマット

| 構造 | |
|----------------------|---------|
| タイムスタンプを含むトレース イベント長 | トレース ID |
| タイムスタンプ (ミリ秒) | |
| タイムスタンプ (小数) | |
| ペイロード | |
| ... | |

現在、PLM はトレース イベント ログの一部としてロードされるイメージのリストのみをログに記録します。

表 16: トレース イベント テーブル

| トレース イベント ID | トレース イベント長 | ペイロード |
|----------------------------------|------------|---------|
| XPLMI_TRACE_LOG_LOAD_IMAGE = 0x1 | 3 | イメージ ID |

イベント トレース ログ コマンドの例

次に、イベント トレース ログの構造の例を示します。

例: ログ レベルをデバッグ情報に変更

| 構造 |
|--------------------------------------|
| CMD - 0x020113 (イベント ログ コマンド) |
| SubCmd - 0x1 (ログ レベルを変更するためのサブ コマンド) |
| Arg1 - 0x4 (デバッグ情報に変更) |

例: デバッグ ログ バッファのアドレスを変更

次のコマンド構造は、ログ バッファの開始アドレスを 1M (サイズ 512K) に変更します。

| 構造 |
|--|
| CMD - 0x040113 (イベント ログ コマンド) |
| SubCmd - 0x2 (ログ バッファ アドレスを変更するためのサブ コマンド) |
| Arg1 - 0x0 (上位アドレス) |
| Arg2 - 0x100000 (下位アドレス 1M) |
| Arg3 - 0x80000 (ログ バッファ サイズ 512K) |

例: ログ データをコピー

| 構造 |
|--|
| CMD - 0x030113 (イベント ログ コマンド) |
| SubCmd - 0x3 (ログ バッファを下記のアドレスにコピーするためのサブ コマンド) |
| Arg1 - 0x0 (上位アドレス) |
| Arg2 - 0x100000 (下位アドレス 1M) |

例: ログ バッファの詳細を取得

| 構造 |
|---|
| CMD - 0x010113 (イベント ログ コマンド) |
| SubCmd - 0x7 (イベント ログ バッファの詳細を取得するためのサブ コマンド) |
| 応答ペイロード 0: コマンド ステータス |
| 応答ペイロード 1: 上位アドレス |
| 応答ペイロード 2: 下位アドレス |
| 応答ペイロード 3: ログがどこまで有効かを示すバッファ オフセット |
| 応答ペイロード 4: ログ バッファ長 |

| 構造 |
|---------------------------------|
| 応答ペイロード 5: ログ バッファがフルかどうかのステータス |

イベント ログ IPI コマンド

表 17: イベント ログ コマンド

| 構造 | | | |
|--------------|--------------|-------|----------------------|
| 予約 [31:24]=0 | 長さ [23:16]=4 | PLM=1 | CMD_EVENT_LOGGING=19 |
| サブ コマンド | | | |
| 引数 1 | | | |
| 引数 2 | | | |
| 引数 3 | | | |

PLM は、PLM ターミナル出力とトレース イベントのログを別々のバッファに記録する機能をサポートしています。このコマンドは、ログを記録するためのバッファを設定し、バッファ情報を取得します。

サブ コマンド情報は次のとおりです。

- 1U: このサブ コマンドは、PLM ターミナルのログ レベルを動作中に設定するために使用します。このログ レベルは、コンパイル時に設定したログ レベル以下の値にする必要があります。
 - 引数 1: ログ レベル。この引数は、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。
 - 0x1U: 無条件メッセージ (DEBUG_PRINT_ALWAYS)
 - 0x2U: 一般的なデバッグ メッセージ (DEBUG_GENERAL)
 - 0x4U: より多くのデバッグ情報 (DEBUG_INFO)
 - 0x8U: 詳細なデバッグ情報 (DEBUG_DETAILED)
- 2U: このサブ コマンドは、デバッグ ログ バッファのメモリを設定するために使用します。デフォルトでは、このメモリは PMC RAM に設定されます。必要に応じて、このメモリを変更できます。
 - 引数 1: PLM ターミナル出力のログを記録するメモリの上位アドレス
 - 引数 2: PLM ターミナル出力のログを記録するメモリの下位アドレス
 - 引数 3: デバッグ ログ バッファの長さ
- 3U: このサブ コマンドは、デバッグ ログ バッファに記録した PLM 出力ログを任意の場所にコピーするために使用します。
 - 引数 1: ログ バッファ データのコピー先上位アドレス
 - 引数 2: ログ バッファ データのコピー先下位アドレス
- 4U: このサブ コマンドは、PLM 出力のログを記録するデバッグ ログ バッファ メモリの詳細を取得するために使用します。応答は次のとおりです。
 - ペイロード 0: コマンド ステータス
 - ペイロード 1: ログが記録されるデバッグ ログ バッファの上位アドレス

- ペイロード 2: ログが記録されるデバッグ ログ バッファの上位アドレス
- ペイロード 3: ログがどこまで記録されるかを示すデバッグ ログ バッファ オフセット
- ペイロード 4: デバッグ ログ バッファの長さ
- ペイロード 5: デバッグ ログ バッファがフルかどうかのステータス
- 5U: このサブ コマンドは、トレース ログ バッファのメモリを設定するために使用します。デフォルトでは、このメモリは PMC RAM に設定されます。必要に応じて、このメモリを変更できます。
 - 引数 1: トレース イベントのログを記録するメモリの上位アドレス
 - 引数 2: トレース イベントのログを記録するメモリの下位アドレス。デバッグ ログ バッファの長さ
- 6U: このサブ コマンドは、トレース ログ バッファに記録したトレース イベント ログを任意の場所にコピーするために使用します。
 - 引数 1: トレース ログ バッファ データのコピー先上位アドレス
 - 引数 2: トレース ログ バッファ データのコピー先下位アドレス
- 7U: このサブ コマンドは、トレース イベントのログを記録するトレース ログ バッファ メモリの詳細を取得するために使用します。応答は次のとおりです。
 - ペイロード 0: コマンド ステータス
 - ペイロード 1: ログが記録されるトレース ログ バッファの上位アドレス
 - ペイロード 2: ログが記録されるトレース ログ バッファの下位アドレス
 - ペイロード 3: ログがどこまで記録されるかを示すトレース ログ バッファ オフセット
 - ペイロード 4: トレース ログ バッファの長さ
 - ペイロード 5: トレース ログ バッファがフルかどうかのステータス

エラー マネージャー

PLM のエラー管理モジュールは、Versal プラットフォーム全体の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。また、これらのエラー アクションをカスタマイズするオプションもあります。エラー管理の機能は、PLM でサポートされる重要な実行時プラットフォーム管理機能の 1 つです。ハードウェアには PMC 用と PSM 用にそれぞれ 2 つのエラー ステータス レジスタがあり、ここには発生したエラーのタイプが記録されます。個々のエラーが PMC/PSM MicroBlaze プロセッサに割り込みを送信するかどうかはユーザーが設定できます。PMC および PSM に送信される各エラーについて、ハンドラー ポインター、エラー アクションおよびサブシステム (エラー アクション SUBSYSTEM SHUTDOWN または SUBSYSTEM RESTART の場合にシャットダウンまたはリスタートするサブシステム) の情報を格納したエラー テーブルは、PLMI にあるエラー マネージャー コードによって管理されます。各エラーに対して、サポートされる任意のエラー アクションを設定しておく、そのエラーが発生した場合に適切なアクションを実行できます。

エラー アクションには、次のものがあります。

- パワーオン リセット (POR) を生成する。
- システム リセットを生成する。
- デバイスのエラー出力信号をアサートする。
- アクションなし (エラーに対するすべてのアクションを無効にし、対応するステータス ビットをクリアする)。

PLM エラー マネージャーには、発生したエラーに対するデフォルトのエラー アクションを割り当てるための API があります。PLM モジュールの初期化時に、PLMI はエラー マネージャーを初期化し、エラーを有効にし、`xplmi_err.c` ファイルで定義したエラー テーブル構造体に従って、各エラーに対するエラー アクションを設定します。

エラー管理ハードウェア

Versal デバイスには、SoC 全体で発生した致命的エラーを集約して処理する専用のエラー ハンドラーがあります。エラー マネージャーは、このハードウェアを使用して致命的エラーを処理し、SRST/PoR/エラー出力、または PMC/PSM への割り込みをトリガーします。

詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 ([AM011](#)) を参照してください。

エラー管理 API 呼び出し

PLM エラー マネージャーは、次の API をサポートしています。

- `XPlmi_EmSetAction`: エラー アクションをセットアップするための API です。この API は、エラー ノード ID、エラー マスク、アクション ID (およびアクション ID が PMC への割り込みの場合はエラー ハンドラー) を入力引数にとります。この関数を呼び出すと、それまでのエラー アクションが無効になり、指定したエラーに対してアクション ID で指定したアクションが設定されます。
- `XPlmi_EmDisable`: 特定のエラー ID に対するすべてのエラー アクションを無効にするための API です。この API は、エラー ID を入力引数にとります。

エラー管理 CDO コマンド

PLM エラー管理モジュールは、次の CDO コマンドをサポートしています。

EM アクションを設定

表 18: コマンド: Set EM Action

| 構造 | | | |
|--------------|--------------|----------|---------------------|
| 予約 [31:24]=0 | 長さ [23:16]=3 | EM=8 | CMD_SET_EM_ACTION=1 |
| エラー イベント ID | | | |
| 予約 | | アクション ID | |
| エラー マスク | | | |

このコマンドは、指定したエラー イベント ID およびエラー マスクに対するエラー アクションを設定するために使用します。サポートされるエラー イベント ID およびエラー マスクの一覧は、次に示すエラー イベント ID の表を参照してください。現在、エラー管理 API は IPI ではサポートされません。

- アクション:
 - パワーオン リセット: 0x1
 - システム リセット: 0x2
 - エラー出力: 0x4
 - なし: 0x7。イベントに対するすべてのアクションを 無効にし、エラー ステータスをクリアする

注記: PSM エラー イベントの場合、LPD が初期化されていないと、このコマンドはエラーを返します。

EM イベントに対する通知を登録

表 19: コマンド: Register Notifier

| 構造 | | | |
|----------------------|--------------|-------------|----------------------------|
| 予約 [31:24]=0 | 長さ [23:16]=4 | PMC_XILPM=2 | CMD_PM_REGISTER_NOTIFIER=5 |
| ノード ID (エラー イベント ID) | | | |
| イベント マスク (エラー マスク) | | | |
| 引数 1 | | | |
| 引数 2 | | | |

EM には、登録済みのエラーが発生した場合に、XilPM でサポートされる Register Notifier API を使用してサブシステムに通知を送る機能があります。このコマンドは、登録されたエラーが発生した場合の通知を登録するために使用します。サポートされるエラー イベント ID およびエラー マスクの一覧は、エラー イベント ID の表を参照してください。

- ノード ID: デバイス ID またはエラー イベント ID のいずれか。エラー イベントの登録には、エラー イベント ID を使用します。
- イベント マスク
 - デバイス ID の場合: イベント タイプ
 - エラー イベント ID の場合: エラー マスク
- 引数 1
 - デバイス ID の場合: ウェークアップ
- 引数 2
 - デバイス ID の場合: イネーブル

エラー イベント ID で指定したイベントに対して Register Notifier コマンドを実行すると、対応する PMC/PSM_ERR#N_STATUS ビットがクリアされ、対応する PMC/PSM_IRQ#N_EN に書き込みが実行されて、エラー イベントが有効になります。イベントが発生すると、イベント インデックス (Notify Callback でセットされたビット) が返されます。

Register Notifier コマンドは Notify Callback コマンドと連携して動作します。

たとえば、エラー ノード GT_CR エラー イベントに対して Register Notifier コマンドを実行すると、PMC_ERR1_STATUS.GT_CR がクリアされ、PMC_IRQ1_EN.GT_CR が有効になり、数値 (たとえば 5) が返されます。この場合、Notify Callback はイベント ステータスのビット 5 をセットして、GT_CR エラーが発生したことを示します。

コールバックを通知

表 20: コマンド: Notify Callback

| 構造 | | | |
|----------------------|--------------|-------------|------------------------|
| 予約 [31:24]=0 | 長さ [23:16]=4 | PMC_LIBPM=2 | CMD_PM_NOTIFY_CALLBACK |
| ノード ID (エラー イベント ID) | | | |
| イベント ステータス (エラー マスク) | | | |

エラー イベント ID で指定したイベントに対して Notify Callback コマンドを実行すると、エラー ノードが無効になります。たとえば、エラー ノード GT_CR エラー イベントに対する通知は、PMC_IRQ1_DIS.GT_CR に書き込むことによってエラーを無効にします。再び通知を受けるには、登録し直す必要があります。

エラー イベント

利用可能なエラー イベントの一覧は、『Versal ACAP テクニカル リファレンス マニュアル』(AM011)を参照してください。

XilLoader

PDI には、システムにロードする必要のあるすべてのイメージが含まれます。PLM はブート デバイスから PDI を読み出し、これらのイメージをイメージ ヘッダーに基づいて適切なメモリにロードします。

注記: PDI フォーマットの詳細は、[第 7 章: ブートおよびコンフィギュレーション](#) の「PDI」のセクションを参照してください。

XilLoader は、各モジュールが PDI 内のイメージをロード/開始/参照するために必要なインターフェイスを提供します。XilLoader は XilSecure (セキュア ブート時)、XilPM (サブシステムの立ち上げ時)、およびブート ドライバー (ブート時)とも相互に通信します。

XilLoader には、次の関数があります。

- `XLoader_LoadAndStartSubSystemPdi`: この関数は、PDI ポインターを入力パラメーターにとります。この API を呼び出すと、PDI イメージ内のイメージがロードされ、PDI 内のハンドオフ情報に基づいてサブシステムが開始されます。この API は、PDI 内のイメージおよびパーティション情報に基づいて PDI ソースから各イメージ パーティションを読み出してロードします。イメージがハンドオフを必要とするかどうかをチェックし、必要なら XilPM API を呼び出してハンドオフを実行します。

イメージは、CDO パーティションまたはサブシステム イメージとすることができます。イメージが CDO パーティションの場合、この API は PLMI 経由で CDO パーサー API を呼び出します。イメージがサブシステムの実行ファイルパーティションの場合、この API はパーティションをそれぞれのサブシステムのメモリへロードします。イメージをロードしたら、この API は PDI から CPI ID とハンドオフ アドレスを読み出し、これらの情報を使用して XilPM API を呼び出します。イメージのロードまたは開始中にエラーが発生した場合、この API は該当するエラー コードを返します。それ以外の場合は、SUCCESS が返されます。

- `XLoader_RestartImage`: PDI の各イメージには、一意の ID があります。この関数はイメージ ID を入力にとり、PDI 内のイメージを特定してそのイメージをリスタートします。この API は、格納されたサブシステム情報をイメージ ID に基づいてブート時に解析し、PDI 内のイメージ番号とパーティション番号を取得します。次に、この API はイメージ パーティションを読み出し、これらをロードします。また、この API はイメージがハンドオフを必要とするかどうかをチェックし、必要なら XilPM API を呼び出します。イメージのリスタート中にエラーが発生した場合、この API は該当するエラー コードを返します。それ以外の場合は、SUCCESS が返されます。

動作シーケンス

ローダーとその他のコンポーネント間で発生する動作シーケンスは次のとおりです。

1. ブート モードに応じたフラッシュ ドライバーを使用して PDI を読み出します。
2. イメージ ヘッダーに基づき、ローダーがイメージに関する CPU の詳細を取得します。
3. ローダーが XilPlmi API を使用して CDO をロードし、XilPM API を使用してプロセッサを開始/停止し、メモリを初期化します。

4. ローダーが XilSecure を使用して、イメージに対してチェックサム計算、認証、または復号化を実行します。

ブート ドライバー

XilLoader は、次の高レベル ブート デバイス インターフェイス API を実装します。これら呼び出して、ブート デバイスを初期化し、ブート デバイスから適切なサブシステム メモリヘイメージをロードします。これらの API は、必要に応じて内部でドライバー API を呼び出します。

- フラッシュ ドライバー: QSPI、OSPI、SD/eMMC ドライバーを使用してフラッシュ デバイスからイメージを読み出します。
- PMC DMA: PMC DMA 関連機能のための API。

XilLoader/IPI CDO のコマンド

次の表に、XilLoader がサポートするコマンドの一覧を示します。

表 21: XilLoader/IPI CDO のコマンド

| コマンド | API ID |
|---------------------|--------|
| 予約 | 0 |
| Load Partial PDI | 1 |
| Load DDR Copy Image | 2 |

Load Partial PDI

Load Partial PDI は、DDR およびフラッシュ デバイスからのパーシャル リコンフィギュレーションをサポートするために IPI インターフェイスで使用する CDO コマンドです。パーシャル リコンフィギュレーションは、IPI コマンドの PdiSrc フィールドに DDR/QSPI/OSPI フラッシュを指定して要求します。PdiSrc フィールドの値は、プライマリ ブート デバイスとして使用する場合のブート モードの値と同じになります。

表 22: Load Partial PDI の構造

| 構造 | | | |
|---|--------------|-------------|-----------------------------|
| 予約 [31:24]=0 | 長さ [23:16]=3 | XilLoader=7 | CMD_XILLOADER_LOAD_PPDI = 1 |
| PdiSrc - 0x1=QSPI24、0x2=QSPI32、0x8=OSPI、0xF=DDR | | | |
| 上位 PDI アドレス | | | |
| 下位 PDI アドレス | | | |

Load DDR Copy Image

表 23: Load DDR Copy Image の構造

| 構造 | | | |
|--------------|--------------|-------------|------------------------------------|
| 予約 [31:24]=0 | 長さ [23:16]=1 | XilLoader=7 | CMD_XILLOADER_LOAD_DDR_CPY_IMG = 2 |
| イメージ ID | | | |
| ファンクション ID | | | |

イメージの遅延ロード

PLM は、メモリへのコピーとイメージの遅延ハンドオフの機能をサポートしています。シナリオによっては、PDI に含まれる特定のサブシステム イメージのロードを特定のブート時点まで遅らせたり、特定のサブシステム イメージへのハンドオフを遅らせたりすることが必要な場合があります。このようにして特定のブート順を確立することにより、ブート時間を短縮できることがあります。

メモリへのコピー

BIF ファイルで `copy = <DDR address>` の属性を指定したイメージは、DDR の指定アドレスに格納された後、ブート PDI ロード時にロードされます。`copy` 属性を指定すると、イメージをロードする際にイメージのバックアップが DDR に作成されます。BIF ファイルで `delay_load` を指定したイメージは、ロードがスキップされます。

BIF ファイルで `copy = <DDR address>`, `delay_load` と指定すると、DDR にバックアップ イメージが作成され、DDR イメージのロードがスキップされます。

イメージをロードするには、IPI コマンド `0x10702` の引数に `image ID`, `function ID` を指定します。この IPI コマンドを実行すると、ほかのすべてのイメージが動作を開始した後に、これらのイメージがロードされ、実行されます。1 つのイメージに対して `delay_load` と `delay_handoff` を同時に指定することはできません。

ただし、`copy` と `delay_handoff` は 1 つのイメージに対して同時に指定できます。

パーシャル PDI では、メモリへのコピー機能はサポートされません。

イメージの遅延ハンドオフ

Bootgen に入力する BIF ファイル内の特定のイメージ ID に、`delay_handoff` 属性を指定できます。PLM は、`delay_handoff` 属性を読み出すと、ブート PDI のロードが完了するまでそのイメージのハンドオフを遅らせます。つまり、`delay_handoff` 属性を指定したイメージは、`delay_handoff` および `copy = 1` 属性が指定されていないイメージの実行が開始するまで実行を開始しません。イメージの遅延ハンドオフは、フル PDI とパーシャル PDI の両方でサポートされます。

XilPM

プラットフォーム管理 (XilPM) は、ノードのサブシステム、MIO、クロック、電源、およびリセット設定を作成および管理するためのインターフェイスを提供するライブラリです。次の表に、このモジュールがサポートするコマンドの一覧を示します。プラットフォーム管理の詳細は、[第 10 章: Versal ACAP プラットフォーム管理](#) を参照してください。

表 24: プラットフォーム管理モジュール

| コマンド名 | API ID |
|----------------------|--------|
| ノード関連コマンド | |
| PM_GET_NODE_STATUS | 3 |
| PM_REQUEST_SUSPEND | 6 |
| PM_SELF_SUSPEND | 7 |
| PM_ABORT_SUSPEND | 9 |
| PM_REQUEST_WAKEUP | 10 |
| PM_SET_WAKEUP_SOURCE | 11 |

表 24: プラットフォーム管理モジュール (続き)

| コマンド名 | API ID |
|-----------------------------|--------|
| PM_REQUEST_NODE | 13 |
| PM_RELEASE_NODE | 14 |
| PM_SET_REQUIREMENT | 15 |
| PM_SET_MAX_LATENCY | 16 |
| リセット制御コマンド | |
| PM_RESET_ASSERT | 17 |
| PM_RESET_GET_STATUS | 18 |
| ピン制御コマンド | |
| PM_PINCTRL_REQUEST | 28 |
| PM_PINCTRL_RELEASE | 29 |
| PM_PINCTRL_GET_FUNCTION | 30 |
| PM_PINCTRL_SET_FUNCTION | 31 |
| PM_PINCTRL_CONFIG_PARAM_GET | 32 |
| PM_PINCTRL_CONFIG_PARAM_SET | 33 |
| 汎用コマンド | |
| PM_GET_API_VERSION | 1 |
| PM_REGISTER_NOTIFIER | 5 |
| PM_FORCE_POWERDOWN | 8 |
| PM_SYSTEM_SHUTDOWN | 12 |
| PM_INIT_FINALIZE | 21 |
| PM_GET_CHIPID | 24 |
| PM_QUERY_DATA | 35 |
| PM_IOCTL | 34 |
| クロック制御コマンド | |
| PM_CLOCK_ENABLE | 36 |
| PM_CLOCK_DISABLE | 37 |
| PM_CLOCK_GETSTATE | 38 |
| PM_CLOCK_SETRATE | 39 |
| PM_CLOCK_GETRATE | 40 |
| PM_CLOCK_SETDIVIDER | 41 |
| PM_CLOCK_GETDIVIDER | 42 |
| PM_CLOCK_SETPARENT | 43 |
| PM_CLOCK_GETPARENT | 44 |
| PM_PLL_SET_PARAMETER | 48 |
| PM_PLL_GET_PARAMETER | 49 |
| PM_PLL_SET_MODE | 50 |
| PM_PLL_GET_MODE | 51 |

XilSecure

XilSecure ライブラリは、ハード暗号化コアにアクセスして AES-GCM 256 ビット/128 ビット エンジン、RSA/ECC エンジン (RSA-4096、RSA-3076、RSA-2048、および ECDSA NIST P-384、NIST P-521 をサポート)、および SHA3/384 エンジンをサポートするためのセキュリティ ドライバーのライブラリです。

詳細は、[第 9 章: セキュリティ](#) を参照してください。

XilSEM

XilSEM (ザイリンクス Soft Error Mitigation) ライブラリは、Versal ACAP のコンフィギュレーション メモリにおけるソフト エラーを検出および訂正 (オプション) するためのソリューションで、コンフィギュレーションと検証が事前に完了しています。

詳細は、『OS およびライブラリ資料コレクション』 ([UG643](#)) を参照してください。

PLM の使用

このセクションでは、Vitis™ ツールを使用した PLM ソフトウェアのビルドについて説明します。これには、使用可能なビルド フラグ、PLM メモリのレイアウト、および PLM 用の予約メモリ/レジスタについての説明が含まれます。Vitis ツールを使用して PLM のビルドを実行するには、『ザイリンクス エンベデッド デザイン チュートリアル: Versal Adaptive Compute Acceleration Platform』 ([UG1305](#)) を参照してください。

PLM のビルド フラグ

次の表に、PLM の重要なビルド フラグとその使用方法を示します。全ビルド フラグの一覧は、XilPLMI ライブラリの `xplmi_config.h` ファイルを参照してください。ビルド フラグに変更を適用するには、BSP と PLM アプリケーションをビルドし直してください。

表 25: PLM のビルド フラグ

| フラグ | 説明 | 必要条件 | デフォルト設定 |
|--------------------|--------------------------------------|----------------------------|---------|
| PLM_PRINT | 有効にすると、PLM ヘッダーおよび必須のプリントを出力する。 | なし | 無効 |
| PLM_DEBUG | 基本情報とすべてのエラー メッセージを出力する。 | なし | 有効 |
| PLM_DEBUG_INFO | 基本情報のほかに、フォーマット指定子を付けて出力する。 | なし | 無効 |
| PLM_DEBUG_DETAILED | 詳細情報を出力する。 | なし | 無効 |
| PLM_PRINT_PERF | パーティション、イメージ、およびタスクのロードにかかった時間を出力する。 | 上記のデバッグ出力フラグのいずれかが有効であること。 | 有効 |
| PLM_QSPI_EXCLUDE | 有効にすると、QSPI コードが除外される。 | なし | 無効 |

表 25: PLM のビルド フラグ (続き)

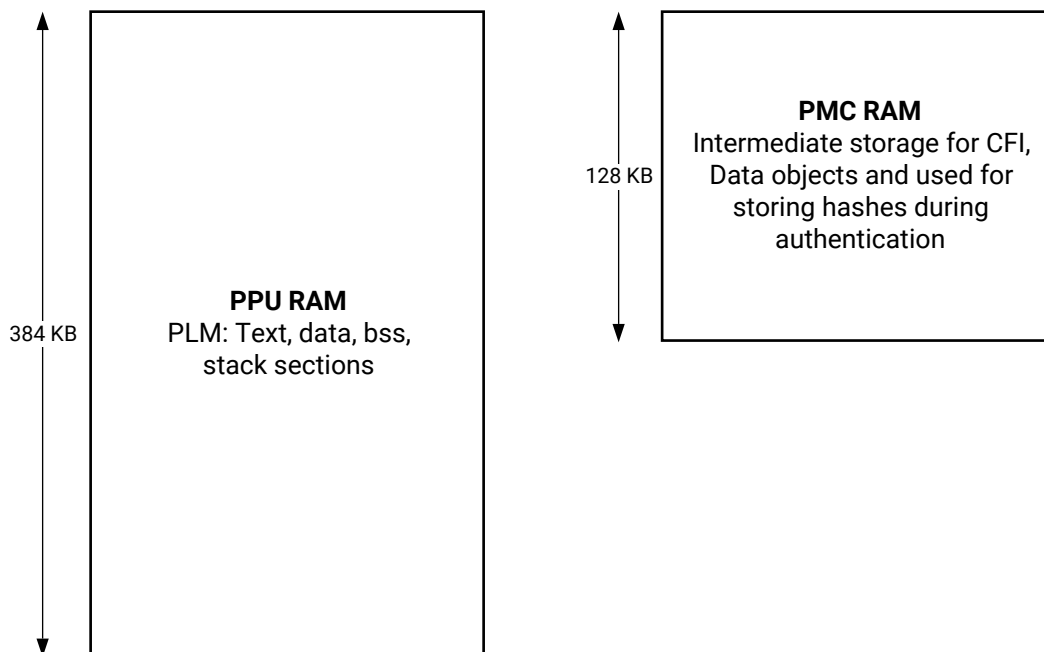
| フラグ | 説明 | 必要条件 | デフォルト設定 |
|----------------------------|---|------|---------|
| PLM_SD_EXCLUDE | 有効にすると、SD コードが除外される。 | なし | 無効 |
| PLM_SEM_EXCLUDE | 有効にすると、xilSEM コードが除外される。 | なし | 無効 |
| PLM_DEBUG_MODE | ベース PDI のロード中にエラーが発生した場合、ソフトリセットを無効にする。デバッグ用にシステム ステートを維持するのに役立つ。 | なし | 無効 |
| PLM_PRINT_PERF_POLL | MASK_POLL コマンドのポーリングにかかった時間を出力する。 | なし | 無効 |
| PLM_PRINT_PERF_DMA | PMC DMA、QSPI、OSPI にかかった時間を出力する。 | なし | 無効 |
| PLM_PRINT_PERF_CDO_PROCESS | CDO ファイルの処理にかかった時間を出力する。 | なし | 無効 |
| PLM_PRINT_PERF_KEYHOLE | キーホール コマンドの処理にかかった時間を出力する。キーホール コマンドは Cframe およびスレーブ SLR イメージのロードに使用する。 | なし | 無効 |
| PLM_PRINT_PERF_PL | PL の電源ステータスとハウスクリーニング ステータスを出力する。 | なし | 無効 |

PMC メモリ レイアウトおよび PLM フットプリント

このセクションでは、PLM のビルド オプションによって PMC メモリ レイアウトと PLM メモリ フットプリントがどのように変わるかを示します。

次の図に、PMC メモリ レイアウトを示します。

図 28: PMC メモリ レイアウト



X23867-042020

PLM では、PLM_DEBUG および PLM_PRINT_PERF ビルド フラグとすべてのモジュールがデフォルトで有効です。

表 26: PLM フットプリント

| S. No | 機能/コンポーネント | 占有サイズ (KB) | 説明 | 備考 |
|-------|------------------------|------------|---|------------------------------|
| 1 | PLM のデフォルト ビルド | 348KB | デフォルトの PLM にはすべての PLM モジュールと基本的な PLM プリントが含まれ、タイムスタンプが有効です。(PLM_DEBUG、PLM_PRINT_PERF) | |
| 2 | 詳細なデバッグ情報の出力を有効にした PLM | 360 KB | PLM_DEBUG_INFO フラグを有効にすると、詳細なデバッグ情報が出力されます。 | この見積もり値は、(1) と (2) を足したものです。 |

PLM 用の予約メモリおよびレジスタ

次のレジスタとメモリ領域は PLM 用に予約されています。

- PPU RAM の 384KB 領域: PLM の実行用に予約されています。このメモリ領域は、PLM ELF ファイルのコード、データ、および BSS セクションに使用します。
- PMC RAM の 128 KB 領域: ツールで生成したデータ、認証/復号チャンク、スレーブ ブート モード、およびイベント ロギングに使用します。

- PMC_GLOBAL_GLOBAL_GEN_STORAGE0、および PMC_GLOBAL_GLOBAL_GEN_STORAGE1: ROM 実行タイムスタンプを格納するための ROM 用予約レジスタ。PLM がアクティブな場合、これら 2 つのレジスタを読み出して ROM 実行時間を取得します。
- PMC_GLOBAL_GLOBAL_GEN_STORAGE2:
デバイスのセキュリティ ステータスを格納し、ROM によって更新されます。PLM はこのレジスタを使用して KAT を実行する必要があるかどうかを判断します。
- PMC_GLOBAL_GLOBAL_GEN_STORAGE4:
PLM は、このレジスタを使用して ATF ハンドオフ パラメーターのアドレス ポインターを格納します。
- PMC_GLOBAL_PERS_GLOB_GEN_STORAGE0: XilPM が各電源ドメインの初期化ステータスを保存するための予約レジスタ。
- PMC_GLOBAL_PERS_GLOB_GEN_STORAGE1: 予約
- PMC_GLOBAL_PERS_GLOB_GEN_STORAGE2: XilPM がリセット要因を保存するために使用します。

サービス フロー

割り込みはすべてイベントとして追加され、通常コンテキストで実行されます。PLM は、受信した割り込みまたはプロセッサ間割り込み (IPI) に基づいて適切なハンドラーを呼び出します。

例外処理

PPU で例外が発生すると、例外ハンドラーが呼び出されます。このハンドラーは例外データのログを記録し、エラーマネージャーのファームウェア エラー ビットをセットします。

ブート PDI のプログラムが完了する前に例外が発生すると、PLM エラー マネージャーは PMC マルチブート レジスタを更新し、システム リセット (SRST) をトリガーします。ブート PDI のプログラムが完了した後に例外が発生した場合、PLM アプリケーションは `while` 文の無限ループで動作します。

セキュリティ

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に利用できる Versal™ デバイスの機能について説明します。

必要なセキュリティ機能のプロダクション対応状況、およびその使用法の詳細は、『Versal ACAP セキュリティ マニュアル』(UG1508)を参照してください。この資料は、[デザイン セキュリティ ラウンジ](#) (要登録) からダウンロードできます。

セキュリティ機能

Versal デバイスには、いくつかのセキュリティ関連機能があります。Versal ACAP の最大のセキュリティ機能の 1 つであるハード暗号化エンジンは、次のものをサポートしています。

- AES-GCM (Advanced Encryption Standard Galois Counter Mode) 128 ビットおよび 256 ビット
- RSA 2048、3072、および 4096
- 複数の曲線をサポートした楕円曲線暗号 (ECC) エンジン
- SHA3/384 ハッシュ生成

Versal ACAP にはハード暗号化エンジンが内蔵されているため、ザイリンクスはセキュア ブート中または実行時に暗号化エンジンを使用するセキュリティ関連ドライバーを提供しています。セキュア ブート中、ROM、PLM、および U-Boot はこれらの暗号機能を使用できます。実行時には、ベアメタル アプリケーションは直接または間接的に (アーキテクチャ コンフィギュレーションによる) これらのドライバーにアクセスできます。これには、OS、ハイパーバイザー、トラステッド実行環境などの使用が含まれます。たとえば Linux アプリケーションでは、アプリケーションが Linux カーネルを呼び出し、Linux カーネルがセキュリティ ライブラリの動作する PLM に IPI 要求を送信するという動作が可能です。これは、実行時にセキュリティ ライブラリにアクセスする方法の一例に過ぎません。Versal ACAP は非常に柔軟なコンフィギュレーションが可能のため、これ以外にも多くのオプションがあります。

必要なセキュリティ機能がザイリンクスから提供されていない場合、PL を使用して追加のセキュリティ機能を実装できるほか、Arm Cortex-A72 プロセッサに内蔵の Arm®v8 暗号拡張命令や Arm NEON 拡張命令を使用することもできます。

既知解テスト

Versal ACAP には、暗号化エンジンの使用前に既知解テスト (KAT) を実行する機能があります。KAT は、ハード暗号化エンジンでデータを処理する前に、その完全性をチェックします。

KAT には次のテストが含まれます。

- SHA3/384
- RSA-4096

- NIST P-384 曲線を使用した ECDSA
- AES-GCM 256 ビット (DPA 対策あり/なし)

KAT はブート中に実行できます。これらの API は、Cortex-A72 または Cortex-R5F プロセッサ上で動作するアプリケーションから明示的に呼び出すことができます。

セキュア ブート

Versal デバイスでは、デバイスにロードされるファームウェアとソフトウェアの機密性、完全性、および認証をセキュア ブートによって確保しています。信頼のルートは bootROM から始まり、bootROM が PLM ソフトウェアを認証/復号化します。このようにして PLM ソフトウェアの信頼を確保した後、PLM がノリノファームウェアとソフトウェアをセキュアな方法で処理します。セキュア ブートがきわめて重要である理由は 2 つあります。

1. デバイスにロードされるソフトウェアが、ロードしてよいものであることを確認する。これにより、悪意のあるコードがデバイス上で実行されるのを防ぎます。
2. ソフトウェアは暗号化された状態で格納されるため、OEM IP が保護される。これにより、OEM の知的財産が盗まれるのを防ぎます。

セキュア ブートを必要としない場合も、ソフトウェアは少なくともシンプルなチェックサムで検証されます。ただし、このようなブート方法では、上記の保護機能は適用されないことに注意してください。次の表に、セキュア ブートの可能な構成を示します。

表 27: 累加的なセキュア ブート動作

| ブート タイプ | 動作 | | | ハードウェア暗号化エンジン |
|---|---------|---|----------------|------------------------------|
| | 認証 | 復号化 | 完全性 (チェックサム検証) | |
| 非セキュア ブート | なし | なし | なし | なし |
| 非対称のハードウェアによる信頼のルート (A-HWRoT) | はい (必須) | なし | なし | RSA/ECDSA と SHA3 |
| 対称のハードウェアによる信頼のルート (S-HWRoT) (PDI の復号化には強制的に eFUSE ブラック キーを使用) | なし | あり (PLM とメタ ヘッダーは eFUSE KEK で暗号化する必要あり) | なし | AES-GCM |
| A-HWRoT + S-HWRoT | はい (必須) | はい (必須) | なし | RSA/ECDSA と SHA3 および AES-GCM |
| PDI の認証と復号化 | あり | Yes (キー ソースは BBRAM または eFUSE) | なし | RSA/ECDSA と SHA3 および AES-GCM |
| 復号化 (ユーザーが選択したキーを使用。キー ソースは、BBRAM/ BHDR または eFUSE など任意のものを使用可能) | なし | あり | なし | AES-GCM |
| チェックサム検査 | なし | なし | Yes | SHA3 |

セキュア ブート プロセスでは、Versal ACAP システムは次のハードウェア暗号化ブロックを使用します。

- SHA ハードウェア アクセラレータ: イメージに対する SHA3/384 ハッシュを計算します。署名用の RSA または ECC (楕円曲線暗号) と組み合わせて使用します。

- ECDSA-RSA ハードウェア アクセラレータ: 公開非対称キーを使用してイメージを認証します。RSA-4096 または ECDSA (NIST P-384 曲線) を使用できます。

ほかのイメージに対しては、PLM は NIST-P384 以外に NIST-P521 曲線も使用できます。MetaHeader、PMC CDO、および PLM には P-381 が必要です。その他すべてのパーティションには、P-521 を使用できます。

- AES-GCM ハード暗号化ブロック: 256 ビット キーを使用してイメージを復号化し、復号化したイメージの完全性を GCM タグを使用して検証します。

ほかのイメージに対しては、PLM は AES-GCM 256 ビット以外に AES-GCM 128 ビットも使用できます。MetaHeader、PMC CDO、および PLM には AES-GCM 256 が必要です。その他すべてのパーティションには、AES-GCM 128 ビットを使用します。

チェックサム検査

Versal ACAP は、SHA-3 ダイジェストを使用して Versal デバイス イメージのデータ完全性を検証します。PDI でチェックサム検証が有効になっている場合、PLM は SHA-3 ダイジェストを計算し、PDI に格納されている SHA-3 ダイジェストと比較します。

A-HWRoT/S-HWRoT にはデータ完全性チェックの機能が含まれているため、これらとチェックサム オプションを組み合わせることはできません。

非対称型のハードウェアによる信頼のルート (A-HWRoT) (認証が必要)

Versal デバイス イメージの SHA3 ハッシュは RSA/ECDSA 秘密キーを使用して署名され、生成されたシグネチャは Versal デバイス イメージに格納されます。ブート時にイメージの SHA3 ハッシュが計算され、イメージに格納されているシグネチャが公開キーを使用して RSA/ECDSA エンジンに渡されます。計算で求めた SHA3 ハッシュと検証済みシグネチャが一致すると、イメージは有効です。

Versal ACAP では、PPK (Primary Public Key) と SPK (Secondary Public Key) の 2 種類の公開キーを使用します。各イメージには、専用または共通の SPK が割り当てられます。たとえば PLM で SPK0 を使用するよう割り当てた場合、Cortex-A72 には同じ SPK0 を割り当てることも、SPK1 など専用の SPK を割り当てることもできます。

Versal デバイスでは、PPK ハッシュの格納場所が eFUSE メモリに 3 つあります (PPK0、PPK1、PPK2)。これら PPK eFUSE のいずれかのビットをプログラムすると、ブート時に A-HWRoT が強制されます。したがって、すべてのソフトウェアを認証してから Versal デバイスにロードする必要があります。非対称キー ペアは、RSA 4096 または ECDSA-P384 楕円曲線を使用できます。3 つの PPK には、RSA と ECDSA ハッシュ値の組み合わせをプログラムできます。

RSA エンジン

ブート中は、RSA-4096 のみがサポートされます。ただしブート後は RSA エンジンにアクセスでき、サイズ 2048、3072、および 4096 ビットの秘密キーおよび公開キーの動作がサポートされます。

ECC (Elliptic Curve Cryptography) エンジン

ブート中は、PLM に対しては NIST P-384 曲線を使用した楕円曲線 DSA (ECDSA) のみがサポートされますが、その他のイメージは NIST P-384 曲線または NIST P-521 曲線を使用して署名できます。ただしブート後は ECC (Elliptic Curve Cryptography) エンジンを利用できるため、NIST P-384 曲線以外にもさまざまな曲線がサポートされます。サポートされる曲線は NIST P-384 と NIST P-521 です。

キーの無効化

eFUSE には PPK のハッシュ値を格納できる領域が 3 つしかなく、これらの値を無効にできるのは 2 回までです。もう 1 回無効化されると、デバイスはブートできなくなります。PPK が漏洩した場合、eFUSE の対応する PPK 無効化ビットをセットすることによって、その公開キーを無効化できます。

SPK を無効化するには、無効化 ID の対応する eFUSE ビットをプログラムします。このビットは全部で 256 個 [0-255] あるため、SPK は 255 回まで無効化できます。無効化 ID のビット 0 が KEY 0 を表し、ビット 32 が KEY 32 を表します (その他も同様)。

暗号化

Versal デバイスは、機密性、認証、および完全性をサポートした AES-GCM ハードウェア エンジンを内蔵しています。GCM は、認証および完全性チェックを支援します。このエンジンは、ブート中またはブート後に対称キーと初期化ベクター (IV) ペアを使用して、データを暗号化/復号化できます。このブート フローでは、PLM をロードして 256 ビットのキーと 96 ビットの初期化ベクターを使用する必要があります。その他のファームウェアおよびソフトウェアには、256 ビットまたは 128 ビットのキー サイズを使用できます。ブート後は、このエンジンは 256 ビットと 128 ビットの両方のキー サイズをサポートします。

DPA 対策

Versal デバイスの AES エンジンには、DPA 攻撃からデバイスを保護する対策の機能があります。オートモーティブデバイス (XA) は、デフォルトで DPA 対策を備えています。その他のデバイスでは、DPA を有効にしたデバイス専用の注文コードを使用する必要があります。

DPA 攻撃に対するもう 1 つの対策として、ブート イメージはキー ローリングをサポートしており、1 つの AES キーの使用を最小に抑えることができます。DPA 対策は、ブート中およびブート後の両方で使用できます。

キー ソース

レッド キー

レッド キーは、BBRAM、eFUSE、ブート ヘッダーのいずれかに平文として格納されます。

ブラック キー

ブラック キーは、PUF で生成されたキー暗号化キー (KEY) を使用してレッド キーを暗号化した後に生成されます。

注記: KEK は Versal デバイスごとに固有で、デバイス外部に読み出すことはできません。

ブラック キーは eFUSE、BBRAM、またはセキュア ブート時のブート ヘッダーのいずれかに格納できます。暗号化のみのブート モードを選択した場合、ブラック キーは eFUSE にのみ格納できます。

ユーザー キー

ブート プロセスの後、BBRAM または eFUSE に格納されたキー、およびメモリに格納されたオペレーティング システムまたはアプリケーション固有のキーを AES エンジンにロードすることもできます。

無効化

無効化 ID が指し示す eFUSE がプログラムされていない場合、無効化 ID を使用したキーは有効です。この eFUSE をプログラムすると、対応するキーが無効化されます。無効化 ID は A-HWRoT ブート モードと共有されるため、キーを無効化すると SPK も無効化されます。

対称型のハードウェアによる信頼のルート (S-HWRot) ブート モード (暗号化が必要)

eFUSE には、S-HWRoT ブート モードに対応する専用ビットがあります。これらビットのいずれかがセットされていると、ブート イメージは暗号化されたフォーマットとする必要があり、キー ソースは eFUSE ブラック キーに限定されます。その他のパーティションについては、暗号化してもしなくてもよく、任意の有効なキー ソースを使用できます。

Versal ACAP プラットフォーム管理

Versal™ ACAP は、複数のユーザー プログラマブル プロセッサ、FPGA、および最先端の電力管理機能を統合したヘテロジニアス マルチプロセッサ SoC 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。電力だけでなく、クロック、リセット、ピンなどのリソースも同様の管理が必要です。この場合の課題は、業界規格 (IEEE P2415) に準拠し、異なるソフトウェアを実行する複数 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。ザイリックスは、プラットフォーム管理コントローラー (PMC) を介して柔軟な管理制御をサポートするプラットフォーム管理機能を開発しました。

このプラットフォーム管理機能は、さまざまな使用状況に対応します。たとえば、Linux は CPU 周波数スケーリングや CPU ホットプラグ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、この機能はユーザーが実装する必要がありますが、プラットフォーム管理フレームワークを使用するとその作業が容易になります。

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、柔軟なプラットフォーム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

プラットフォーム管理の主な機能は次のとおりです。

- PMC を使用することにより、電源ステート、クロック、リセット、およびピンの設定を一元化
- EEMI (Embedded Energy Management Interface) API (IEEE P2415) をサポート
- Linux の共通クロック フレームワークをサポート
- Linux のリセット フレームワークをサポート
- すべてのデバイスの電源ステート、クロック、およびリセットを管理
- Linux の電力管理機能をサポート
 - Linu デバイス ツリー電力管理
 - ATF/PSCI 電力管理サポート
 - CPU 周波数スケーリング
 - CPU 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
- 。 アイドル
- 24 以上の API を使用して次の電力管理機能を直接制御
 - 。 コア管理
 - 。 エラー管理
 - 。 メモリおよびペリフェラル管理
 - 。 電源、クロック、リセット
 - 。 プロセッシング ユニットのサスペンドまたはウェークアップの管理

Versal ACAP プラットフォーム管理の概要

Versal ACAP プラットフォーム管理は EEMI を実装します。Versal ACAP の各種サブシステム上で動作するソフトウェア コンポーネントは、プラットフォーム管理を利用してプラットフォーム管理要求を発行したり、それに応答したりします。また、このプラットフォーム管理は EEMI API を使用してさまざまなプロセッシング ユニットがクロック特性 (イネーブル/ディスエーブル ステート、分周値、および使用する PLL など) を設定できるようにしています。リセット ラインのアサート/ディアサートや、各種ファンクション ユニットで使用するピンの設定変更も、EEMI API 経由でサポートします。

Versal ACAP の電源ドメイン

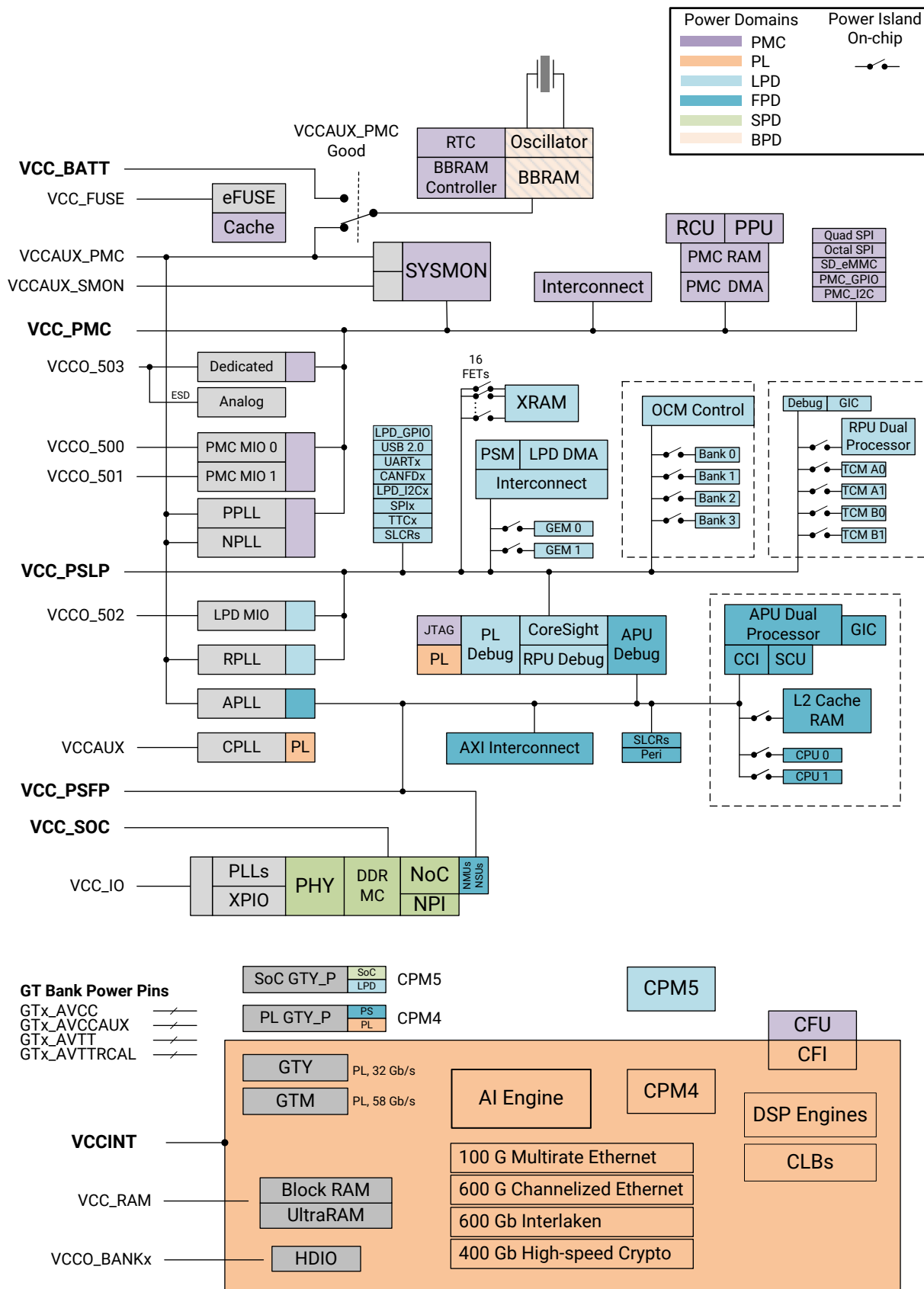
Versal デバイスは、次の電源ドメインで構成されます。

- フル電力ドメイン (FPD): Arm® Cortex-A72 アプリケーション プロセッサ ユニット (APU) が含まれます。
- 低電力ドメイン (LPD): Arm Cortex-R5F リアルタイム プロセッサ ユニット (RPU) やオンチップ ペリフェラルが含まれます。
- システム電源ドメイン (SPD): DDR コントローラーおよび NoC が含まれます。
- PL 電源ドメイン: PL および AI エンジンが含まれます。
- バッテリー電源ドメイン: リアルタイム クロック (RTC) およびバッテリー バックアップ式 RAM (BBRAM) が含まれます。
- PMC 電源ドメイン: プラットフォーム管理コントローラーが含まれます。

バッテリー電源ドメインと PMC 電源ドメインは、フレームワークによって管理されません。プラットフォーム管理による電源ドメインの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。

次の図に、Versal 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 29: 電源ドメインおよび電源アイランドの図



Versal ACAP 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、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コントローラー (PMC) に委託する協調アプローチが採用されています。PMC は、APU または RPU などのその他のプロセッシング ユニット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。

Versal ACAP は、異なるプロセッサ間でのプラットフォーム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、『Versal ACAP テクニカル リファレンス マニュアル』(AM011) の割り込みの情報を参照してください。

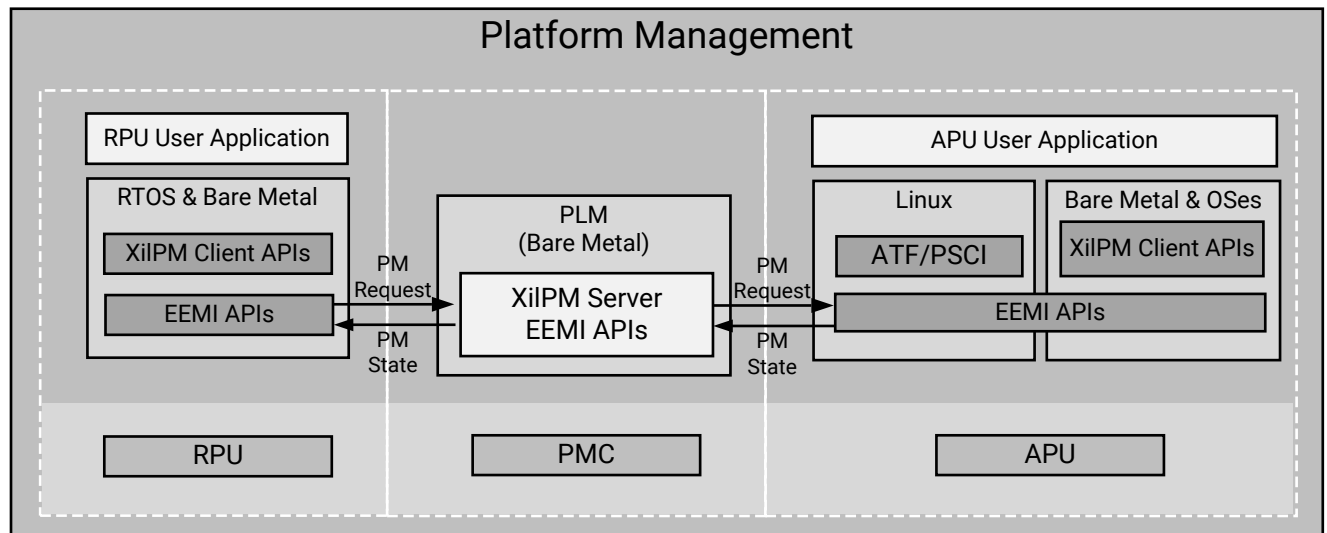
Versal ACAP のプラットフォーム管理ソフトウェア アーキテクチャ

Versal ACAP 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、および監視を制御する専用のプログラマブル ユニットがあります。複数のプロセッサを統合したヘテロジニアス システムのプラットフォーム管理に最適な機能を備えたシステムには、大きなメリットがあります。その一方で、システムの動作は非常に複雑になります。プラットフォーム管理フレームワークはこの複雑さを抽象化し、ユーザーが消費電力バジェットの目標を満たし、リソースを効率よく管理する上で必要な API のみを提供します。

PMC に含まれる PPU が PLM を実行します。PLM は、いくつかのモジュールで構成されます。そのうちの 1 つである XilPM サーバーが、プラットフォーム管理フレームワーク層を実装します。もう 1 つの PPU MicroBlaze プロセッサが、PS 管理コントローラー (PSM) を制御します。PSM ファームウェアは、電源アイランドおよび電源ドメインのパワーアップとパワーダウンを制御します。

PLM は、CDO を介してシステム内のリソースのトポロジをプラットフォーム管理フレームワークに渡します。プラットフォーム管理フレームワークは電源ドメイン、電源アイランド、クロック、リセット、ピンなどのリソース、およびこれらリソースと CPU コア、メモリ、およびペリフェラル デバイスとの関係を管理します。

図 30: プラットフォーム管理フレームワーク



X23401-051220

Versal ACAP のプラットフォーム管理フレームワークは、EEMI の実装をベースにしています (詳細は、『エンベデッド エネルギー管理インターフェイス EEMI API リファレンス ガイド』 (UG1200) 参照)。プロセッシング ユニットが PLM にメッセージを送信するための API や、PLM がプロセッシング ユニットにメッセージを送信するためのコールバック関数があります。

EEMI が提供する共通の API を使用して、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を管理できます。電力管理に関しては、複雑なプロセッサ クラスターや単一コアをサスペンドするなどのハイレベルな管理目標を EEMI で指定できます。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。

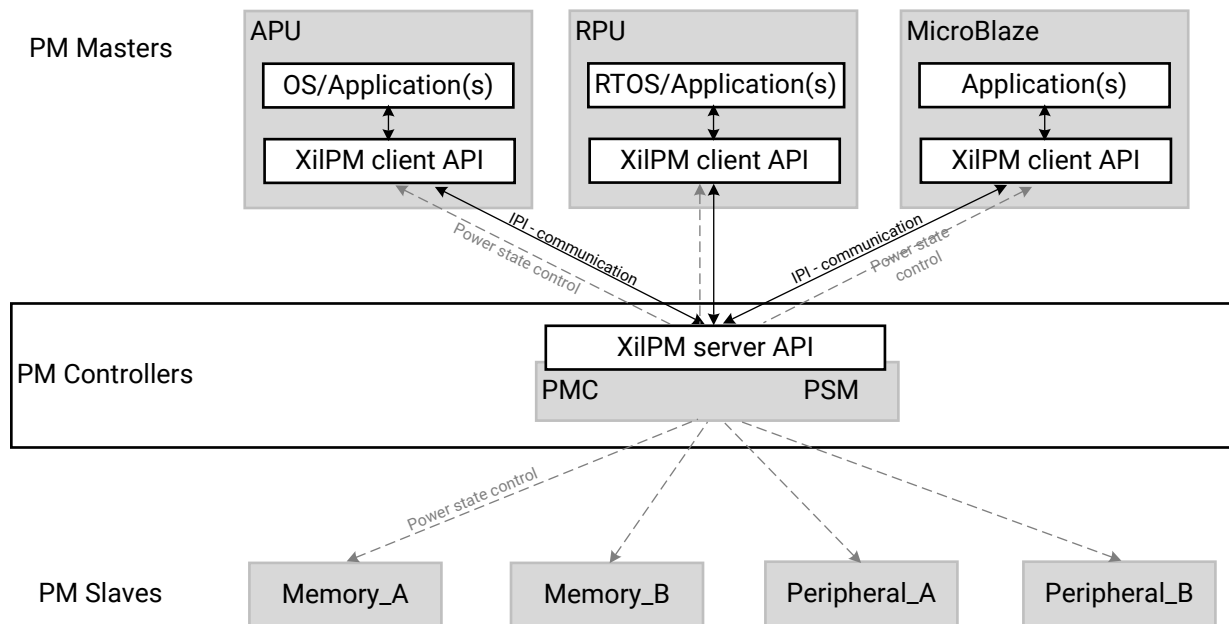
Linux デバイス ツリーは、各デバイスに共通する記述フォーマットと電力特性を提供します。また、Linux は CPU およびクロック周波数スケーリングや CPU ホットプラグなどの基本的な電力管理機能も提供します。カーネルは、実装された API を利用して電力管理を実行します。

24 以上の API にアクセス可能な XilPM クライアント ライブラリを使用して、独自のプラットフォーム管理アプリケーションを作成することも可能です。API は、次のような機能に分類されます。

- 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
- クロック管理
- リセット管理
- ピン管理
- その他

次の図に、API ベースのプラットフォーム管理ソフトウェア アーキテクチャを示します。

図 31: API ベースのプラットフォーム管理ソフトウェア アーキテクチャ



X23402-051220

関連情報

プラットフォーム ロードーおよびマネージャー

API の呼び出しと応答

IPI を使用するプラットフォーム管理の通信

Versal デバイスのプラットフォーム管理通信層は、プロセッサ間割り込み (IPI) ブロックで提供される IPI を使用して実装されます。IPI の詳細は、『Versal ACAP テクニカル リファレンス マニュアル』(AM011) の「割り込み」の章を参照してください。

各プロセッシング ユニットの、割り込みとペイロード バッファで構成される専用の IPI チャネルを使用して PMC と接続します。バッファは API ID と最大 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、プロセッシング ユニットの PMC に IPI を生成して必要なプラットフォーム管理動作の実行を促します。
- PMC は、各要求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。
- PMC からプロセッシング ユニットの通知に使用されるプラットフォーム管理コールバックをサポートするために、各プロセッシング ユニットの IPI の処理機能が実装されています。

プラットフォーム管理層

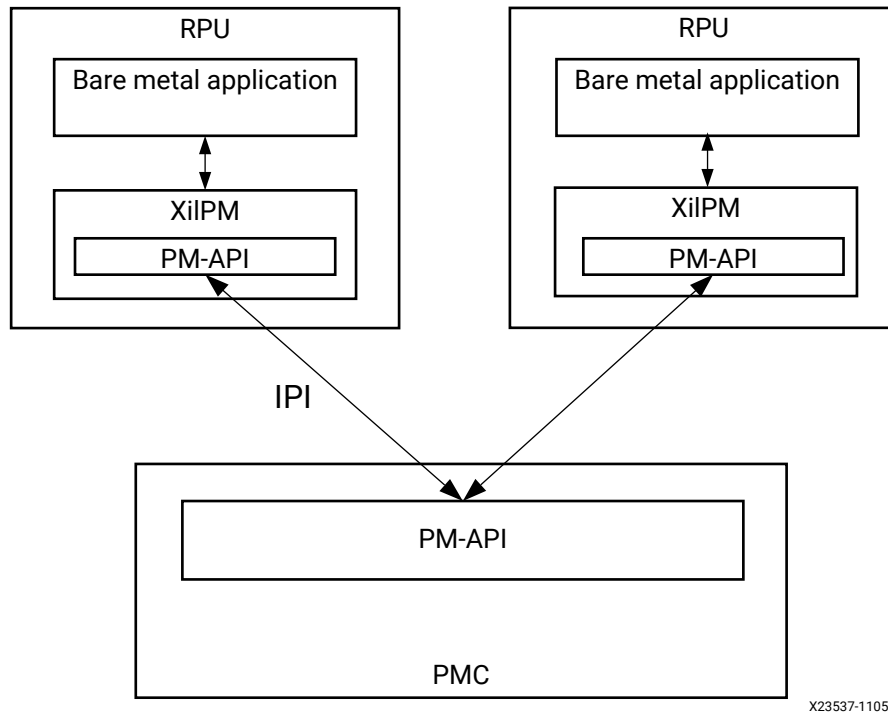
Versal デバイスに実装されたプラットフォーム管理には、次の API 層が含まれます。

- XilPM (クライアント): 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 アプリケーションが使用するライブラリ層です。
- XilPM (サーバー): PLM ファームウェアを構成するライブラリ層で、IPI を介して XilPM クライアント層から渡される要求を処理します。
- ATF: Arm トラステッド ファームウェア (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。ATF は現在、Linux OS で使用されています。
- PLM: IPI パケットを受信し、XilPM サーバーへプラットフォーム管理要求を渡します。
 - XilPM (サーバー): PLM ファームウェアを構成するライブラリ層で、IPI を介して XilPM クライアント層から渡される要求を処理します。
- PSM ファームウェア: XilPM サーバーから呼び出され、電源アイランドおよび電源ドメインを制御します。

PMC、PSM、および電源ドメイン ハードウェアの詳細は、『Versal ACAP テクニカル リファレンス マニュアル』(AM011) を参照してください。

次の図に APU、RPU、およびプラットフォーム管理 API の相互作用を示します。

図 32: ベアメタル アプリケーションのみで使用される API 層

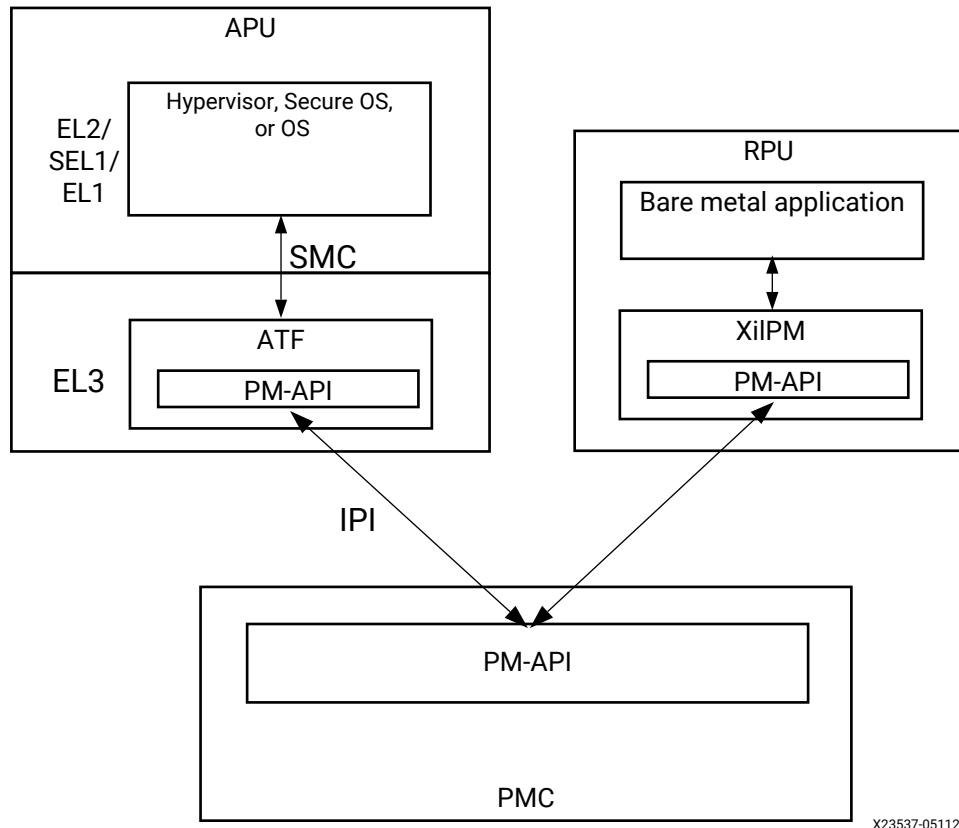


APU がオペレーティング システムを使用してフル ソフトウェア スタックを実行する場合、XilPM ライブラリは使用しません。その代わりに、EL3 で動作する ATF がクライアント側の XilPM API を実装し、EL2、SEL1、または EL1 (Versal ACAP デバイスのシステム アーキテクチャによる) で動作するソフトウェアに対するセキュア モニター コール (SMC) ベースのインターフェイスを提供します。

Armv8 アーキテクチャおよびその各種実装モードの詳細は、<https://developer.arm.com/docs/ddi0487/latest/arm-architecture-reference-manual-armv8-for-armv8-a-architecture-profile> を参照してください。

次の図に、APU 上でフル ソフトウェア スタックが実行される場合のプラットフォーム管理層を示します。

図 33: APU 上でフル ソフトウェア スタックが実行される場合のプラットフォーム管理層



一般的なプラットフォーム管理 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」と呼びます。以降のセクションでは、APU や RPU ではなく各サブシステムに割り当てられたスレーブ ノードとプラットフォーム管理がどのように通信するかについて説明します。

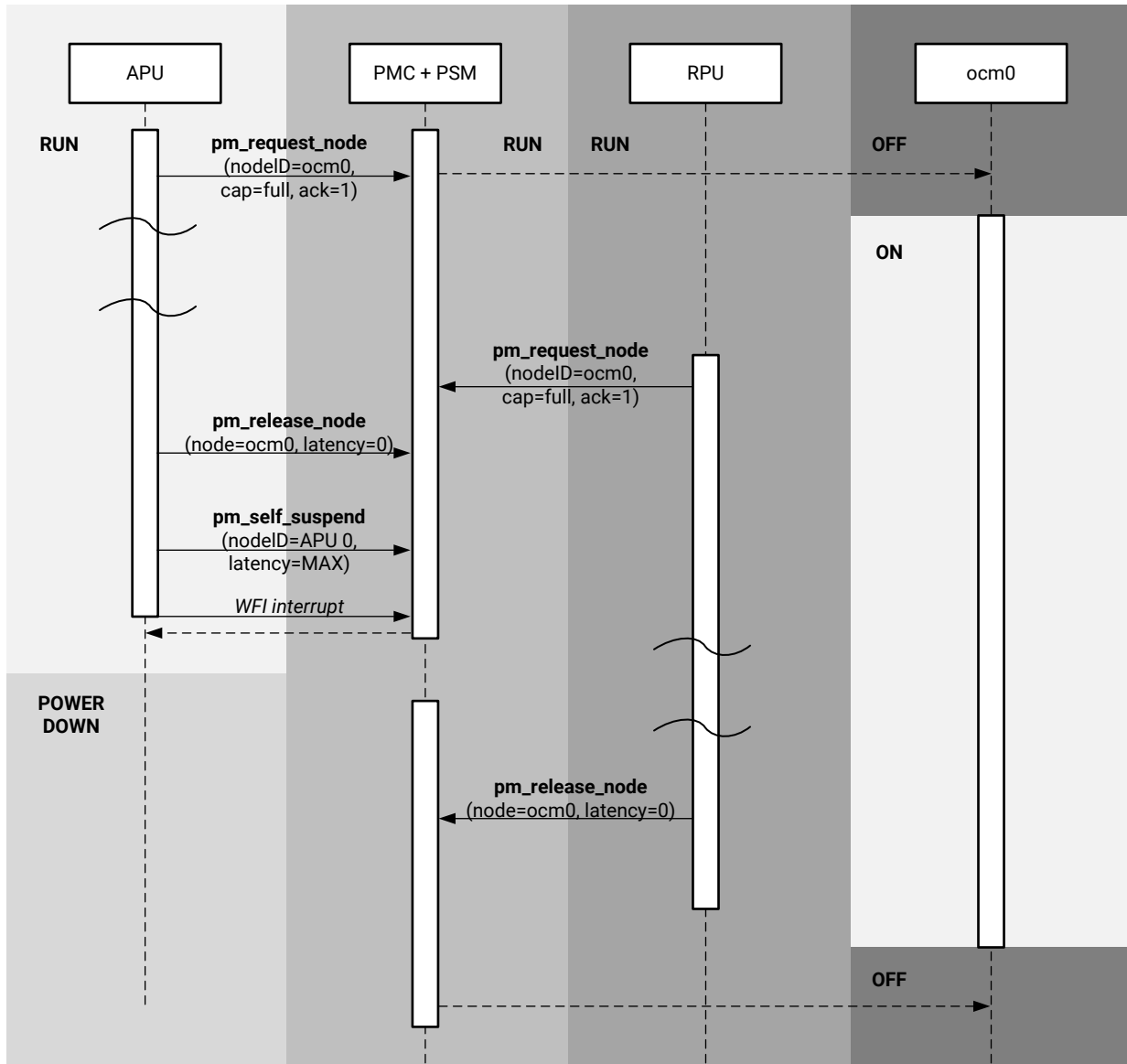
一般に、APU または RPU は、PMC に要求を送ることによって、スレーブ ノードを使用することを通知します。この後、PMC にはスレーブ ノードに求められる機能要件が通知されます。この時点で、PMC がスレーブ ノードの電源を投入し、APU または RPU はスレーブ ノードを初期化できるようになります。

スレーブ ノードの要求とリリース

プロセッシング ユニットがペリフェラルまたはメモリ スレーブ ノードを必要とする場合、電力管理 API を使用してスレーブ 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。

次の図に、APU と RPU が 1 つのオンチップ メモリ メモリ バンク (ocm0) を共有するユース ケースにおける呼び出しフローを示します。

図 34: APU と RPU が 1 つのオンチップ メモリ バンクを共有している場合のプラットフォーム管理フレームワーク呼び出しシーケンス



X20022-111020

注記: APU が `XStatus XPm_ReleaseNode` を呼び出した後も、RPU が同じスレーブ 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` ノードをリリースすると、PMC は PSM に対して `ocm0` メモリの電源をオフにするよう要求します。

プラットフォーム管理のデフォルト サブシステム

現時点では分離コンフィギュレーションはサポートされておらず、すべての PM マスターがデフォルト サブシステムに含まれます。

この構成では、両方の PM マスター (APU と RPU) がすべてのデバイスにアクセスできるため、PM スレーブ デバイスが別のマスターで使用中である可能性を考慮する必要があります。

サブシステムの有効化

サブシステムを有効化 するとは、`XPm_RequestNode` を使用してそのサブシステムに事前に割り当てられたすべてのデバイスを要求することにより、サブシステムを動作状態にすることです。サブシステムの有効化は、サブシステム イメージの実行を開始する前に 1 回だけ必要です。サブシステムをリスタート/シャットダウンするたびに、新しいサブシステム イメージの実行前に有効化が実行されます。

アプリケーション バイナリ (ELF) をプログラマブル デバイス イメージ (PDI) を使用してサブシステム イメージとしてダウンロードすると、PLM ファームウェアは BIF に存在するサブシステム イメージ ID を使用して、対応するサブシステム イメージを自動的に有効化します。現時点では、カスタム サブシステムはサポートされないため、これがデフォルトのサブシステムです。ただしデバッグのユース ケースでは、PDI フローではなくザイリンクス システム デバッガー (XSDB) を使用して、アプリケーション バイナリ (ELF) を直接マスターにダウンロードして実行することが必要になる場合があります。この場合、PLM は ELF に対応するサブシステム イメージの ID を認識できません。このため、サブシステムの動作に必要な事前割り当て済みデバイスが要求されず、予見できない問題が発生することがあります。このため、XSDB を使用してアプリケーション バイナリ (ELF) をダウンロードする前に、サブシステムの有効化を 1 回だけ実行しておく必要があります。

現在、XSDB は特定のマスターにアプリケーション バイナリ (ELF) をダウンロードする前に、デフォルトのサブシステムの有効化を PLM ファームウェアに自動で要求します。将来、複数の (カスタム) サブシステムが存在するようになった場合は、このようなデバッグのユース ケースで目的のサブシステムを明示的に有効化する必要があります。

API を使用した電力管理

このセクションでは、ザイリンクスのプラットフォーム管理 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。これらの API は、XilPM クライアント ライブラリをインポートする任意のベアメタルアプリケーションで使用できます。

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 クライアント ライブラリが提供する機能を利用して電力管理 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、『Zynq UltraScale+ MPSoC: ソフトウェア開発者向けガイド』(UG1137) を参照してください。

XilPM ライブラリの初期化

電力管理 API 呼び出しを開始する前には、`XPm_InitXilpm` API を呼び出して XilPM ライブラリを初期化する必要があります。この API は、適切な IPI ドライバー インスタンスへのポインターを引数にとるため、PM マスターが PMC と通信できるように IPI チャネルを PM マスターに割り当てておく必要があります。

IPI の詳細は、『Versal ACAP テクニカル リファレンス マニュアル』(AM011) の「割り込み」の情報を参照してください。

`XPm_InitXilpm` の詳細は、『OS およびライブラリ資料コレクション』(UG643) を参照してください。

XPm_InitFinalize を使用した電力管理

サブシステムのコンフィギュレーションを完了し、使用されていないノードをパワーダウンして消費電力を最小にするため、サブシステムは `XPm_InitFinalize(void)` 要求を PLM ファームウェアに送ります。

ベアメタル アプリケーションの場合、アプリケーション開発者はアプリケーションから `XPm_InitFinalize(void)` を呼び出す必要があります。

Linux アプリケーションの場合、プラットフォーム管理ドライバが `XPm_InitFinalize()` を呼び出します。

プラットフォーム管理の機能を持たないサブシステムがこの要求を送ることはありません。したがって、そのプラットフォーム管理デバイスは常時、または PM サブシステム自体がパワーダウンするまでパワーアップしたままとります。

`XPm_InitFinalize()` が呼び出されない場合、PLM ファームウェアはどのデバイスもパワーダウンしません。
`XPm_InitFinalize()` の目的は、`XPm_InitFinalize()` の呼び出し元サブシステムがプラットフォーム管理機能に対応している (すなわち、デバイスを必要とする場合にはプラットフォーム管理 API を使用できる) ことをファームウェアに認識させることにあります。`XPm_InitFinalize()` が呼び出されていなくても、`XPm_RequestNode()` によってデバイスはパワーアップされます。また、`XPm_InitFinalize()` が呼び出されていなくても、`XPm_ReleaseNode()` によってノードはリリースされます。この場合も、`XPm_ReleaseNode()` が適用されます。この場合、使用カウントのみがデクリメントし、パワーダウンは実行されません。

XPm_InitFinalize を呼び出さない場合に PLM ファームウェアがデバイスをパワーダウンしない理由

ここでは、PM API を使用する PM 対応の APU サブシステムと、PM API を使用しない PM 非対応の RPU サブシステムの場合を考えます。どちらのサブシステムも TCM を使用するものとします。

APU サブシステムは PM に対応しているため、`XPm_RequestNode(TCM)` と `XPm_InitFinalize()` の両方を呼び出します。すると、PLM ファームウェアは APU サブシステムが PM 対応であることを認識し、デバイスを要求する場合は `XPm_RequestNode()` を呼び出します。

これに対し、RPU は PM に対応しておらず、また、ザイリンクスは `XiIPM` ライブラリを使用しないでアプリケーションを実行することを認めているため、アプリケーションが PM API を要求せずにデバイスを使用している可能性があります。PLM ファームウェアは RPU サブシステムが動作していることは認識しますが、RPU が使用しているデバイスについては認識しません。したがって、PLM ファームウェアは各サブシステムが `XPm_InitFinalize()` を呼び出すまで、どのデバイスもパワーダウンしません。

パワー マネージメントを適切に実行し、必要な電力値を取得するために `XPm_InitFinalize()` を呼び出します。PLM ファームウェアは、各サブシステムから `XPm_InitFinalize()` が呼び出されると、使用されていないすべてのノードをパワーダウンします。また、`XPm_ReleaseNode()` API を呼び出すと、PLM ファームウェアは任意のデバイスのパワーダウンを許可します。

PM 対応サブシステムは、サブシステムの初期化が完了したら `XPm_InitFinalize()` 要求を送信します。これにより PLM ファームウェアは、サブシステム内で使用されていない PM デバイスのパワーダウンを開始します。

スレーブ デバイスの使用

Versal ACAP のプラットフォーム管理には、メモリやペリフェラルなどのスレーブ デバイス (PM スレーブとも呼ぶ) を管理するための専用機能が含まれています。サブシステムはこれらの機能を使用して、PMC にこれらデバイスの要件 (機能やウェークアップ レイテンシなど) を伝えます。PMC は、各デバイスが該当するすべてのサブシステム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

サブシステムがペリフェラルまたはメモリいずれかのデバイス ノードを必要とする場合、電力管理 API を使用してデバイスを要求する必要があります。デバイス ノードが機能の実行を完了して不要になった場合は、デバイスの電源をオフにするか、ほかのサブシステムで使用できるようにデバイスをリリースする必要があります。

`XPm_InitFinalize()` API を呼び出すと、プラットフォーム管理ファームウェアはいずれのサブシステムからも要求されていないデバイス (ペリフェラルまたはメモリ) の電源をオフにします。

特定のペリフェラル/メモリにアクセスするには、`REQUEST_NODE` API に `PM_CAP_ACCESS` 引数を渡して電源をオンにする必要があります。

デバイスが要求されていない場合、デバイス、クロック、およびリセット動作は許可されません。

デバイス要求のパーミッション

PLM ファームウェアは、デバイスを要求するサブシステムに対してパーミッションをチェックします。Vivado ツールでサブシステムを作成する際には、1 つのペリフェラルを複数のサブシステムに割り当てることも、1 つのサブシステムに割り当てることもできます。この情報は PMC CDO にエクスポートされ、デバイスの要求に対するアクセスは、CDO のこの情報に基づいて許可されます。デフォルトでは、このパーミッションはすべてのサブシステムで共通のため、1 つのデバイスを複数のサブシステムが要求できます。例外はクロック、リセットおよび電源操作で、これらはリソースを要求するサブシステムが 1 つしか存在しない場合のみ許可されます。

注記: いずれのサブシステムも、自分自身のデバイス、および `XPm_RequestNode()` を使用して要求したデバイスの初期化がすべて完了したら、`XPm_InitFinalize()` API を呼び出す必要があります。そうしないと、PLM ファームウェアはどのデバイスもパワーダウンしません。

次に、PM API を使用してデバイスを管理する場合の理想的な呼び出しシーケンスを示します。

1. 必要なすべてのデバイスに対して `XPm_RequestNode()` を呼び出す。
2. デバイスを初期化する。
3. `XPm_InitFinalize()` を呼び出して、必要なすべてのデバイスが要求され、初期化されたことを PLM ファームウェアに通知する。

`XPm_InitFinalize()` を呼び出す前に `XPm_RequestNode()` を呼び出すことは必須ではありません。

`XPm_RequestNode()` の前に `XPm_InitFinalize()` を呼び出した場合、PLM ファームウェアはそのデバイスをパワーダウンします (初期状態はパワーオン)。 `XPm_InitFinalize()` の後に `XPm_RequestNode()` を呼び出した場合、PLM ファームウェアはそのデバイスをパワーアップします。

一部のデバイス初期化は、CDO によって実行されます。したがって、`XPm_RequestNode()` の前に `XPm_InitFinalize()` を呼び出した場合、デバイスがまずパワーダウンした後に再びパワーアップするため、初期化が失われる可能性があります。必要なデバイスをすべて要求したら、`XPm_InitFinalize()` を呼び出すことを推奨します。そうしないと、もう一度初期化が必要になります。

`XPm_ReleaseNode()` を呼び出す際は、デバイスがパワーダウンして初期化コンフィギュレーションが失われることがある点に注意してください。

スタンドアロン アプリケーションからのデバイスの要求とリリース

アプリケーションがペリフェラル/デバイスの使用を要求する場合は、`XPm_RequestNode()` を呼び出す必要があります。次に例を示します。

```
XStatus XPm_RequestNode (const u32 DeviceId, const u32 Capabilities, const
u32 QoS, const u32 Ack);
```

引数は次のとおりです。

- Device ID: 要求する PM デバイスのデバイス ID。
- Capabilities: 必要なデバイス固有の機能。複数の組み合わせが可能。機能には次のものがあります。
 - PM_CAP_ACCESS: フル アクセス/機能
 - PM_CAP_CONTEXT: コンテキストを保存
 - PM_CAP_WAKEUP: ウェークアップ割り込みを送信

- PM_CAP_UNUSABLE: 実行時サスペンド (サブシステムに対してデバイスが要求されたが、デバイスのクロックが無効)

詳細は、『OS およびライブラリ資料コレクション』 (UG643) を参照してください。

- QoS: 必要なサービス品質 (0 ~ 100)。

注記: 現在、この引数は利用できません。

- Ack: 要求された肯定応答 (ACK) タイプ。

注記: この引数は、ZynqMP でのみ使用します。Versal デバイスの場合、この引数の値は常に blocking に設定されます。

デバイスが既にいずれかのサブシステムによって要求されている場合、XPm_SetRequirement() を呼び出して要件を変更できます。次に例を示します。

```
XStatus XPm_SetRequirement (const u32 DeviceId, const u32 Capabilities,
                             const u32 QoS, const u32 Ack);
```

アプリケーションで不要になったデバイスは、リリースする必要があります。デバイスをリリースするには、XPm_ReleaseNode() を呼び出します。次に例を示します。

```
XStatus XPm_ReleaseNode (const u32 DeviceId);
```

要件の変更

サブシステムが 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わることがあります。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することがあります。XPm_SetRequirement を使用すると、サブシステムが PM スレーブの機能に対する要件を変更できます。再び要件を変更する可能性がある場合、サブシステムは通常 PM スレーブをリリースしません。

次に示す呼び出しの例は、電源を投入して PM マスターからアクセスできるように node 引数の要件を変更しています。

```
XPm_SetRequirement(node, PM_CAP_ACCESS, 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、ほかのサブシステムがそのデバイスを独占的に使用する可能性があります。

複数のサブシステムが 1 つの PM スレーブ (通常はメモリ) を共有する場合、PMC はその PM スレーブを使用するすべてのサブシステム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、PMC はほかの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、『OS およびライブラリ資料コレクション』 (UG643) を参照してください。

セルフ サスペンド

プロセッシング ユニットは複数の CPU の集合体として構成できます。APU はデュアル コア Arm Cortex-A72 CPU で、RPU はデュアル コア Arm Cortex-R5F CPU です。RPU はスプリット モードで独立動作とすることも、フォールトトレラント モードでロックステップ動作とすることもできます。現在、これらのプロセッシング ユニットはデフォルト サブシステムに含まれます。

いずれのプロセッシング ユニットも、XPm_SelfSuspend API を呼び出して PMC にその意図を通知することにより、自分自身をサスペンドできます。また、プロセッシング ユニットはこの呼び出しの一部としてターゲット ステートも通知する必要があります。現在、PU サスペンドのターゲット ステートとしては、CPU アイドルがサポートされます。

ターゲット ステートには、CPU アイドルとサスペンド (Suspend to RAM) の 2 つがあります。

それぞれの場合に、プロセッシング ユニット、PMC および PSM は次のアクションを実行します。

- CPU アイドル:
 - Linux などのリッチ OS には、どのプロセスでも使用されていないコアをアイドルにした後、パワーダウンする機能があります。これは消費電力の削減に役立ちます。ワークロードが増えた場合、パワーダウンしたコアを OS がウェークアップできます。プラットフォーム管理は ATF および XilPM (クライアント) ライブラリの一部として API 呼び出しを提供します。
 - ベアメタル APU アプリケーションのユース ケースでは、プロセッシング ユニットが XPm_SelfSuspend を使用し、ターゲット ステートを PM_SUSPEND_STATE_CPU_IDLE として設定することにより、CPU アイドルへの移行を呼び出すことができます。
- サスペンド (Suspend to RAM):
 - プラットフォーム管理には、サブシステム全体をサスペンドする機能があります。サブシステムが DDR を使用しており、ほかのサブシステムが DDR を使用していない場合、XilPM は DDR をセルフ リフレッシュ モードに移行します。このモードでは、DDRMCMC は DRAM をリフレッシュする必要がありません。
 - サスペンドしようとするサブシステムに属する PM マスターは、ターゲット ステートを PM_SUSPEND_STATE_SUSPEND_TO_RAM として設定して PMC に対して XPm_SelfSuspend を実行することにより、サブシステム サスペンドへの移行を呼び出すことができます。

XPm_SelfSuspend および XPm_SuspendFinalize の詳細は、『OS およびライブラリ資料コレクション』(UG643)を参照してください。

ウェークアップ ソースの設定

プラットフォーム管理には、PU またはサブシステムをパワーダウンするオプションがあります。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、プラットフォーム管理が FPD 全体の電源をオフにすることもできます。FPD の電源がオフ状態で、LPD または PMC ドメイン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。

サスペンドする前に、PU は XPm_SetWakeUpSource API を呼び出して、スレーブをウェークアップ ソースとして追加する必要があります。その後、PU は要件を 0 に設定できます。ただし、要件を 0 に設定するには次の条件を満たしている必要があります。

1. ほかのサブシステムがデバイスを共有していない。現時点ではデフォルトのサブシステムのみがサポートされているため、この条件は考慮する必要がありません。
2. 同じサブシステム内に、実行中の PM マスターが存在しない (セルフ サスペンドする PU がスレーブ デバイスを使用するため)。

要件を 0 に設定すると、サブシステムがこれらスレーブの動作を必要としないことをプラットフォーム管理に通知することになります。PU がサスペンド処理を終了した後、FPD に属するデバイスが使用されていなければ、PMC は FPD 全体の電源をオンにし、LPD または PMC ドメインのスレーブのウェークアップ イベントを伝搬するように GIC プロキシを設定します。

`XPm_SetWakeupSource` の詳細は、『OS およびライブラリ資料コレクション』 ([UG643](#)) を参照してください。

レジューム (復帰) の実行

CPU は、次の方法でウェークアップできます。

- ハードウェア リソースによってトリガーされるウェークアップ割り込み
- `XPm_RequestWakeup` クライアント API を使用した明示的なウェークアップ要求

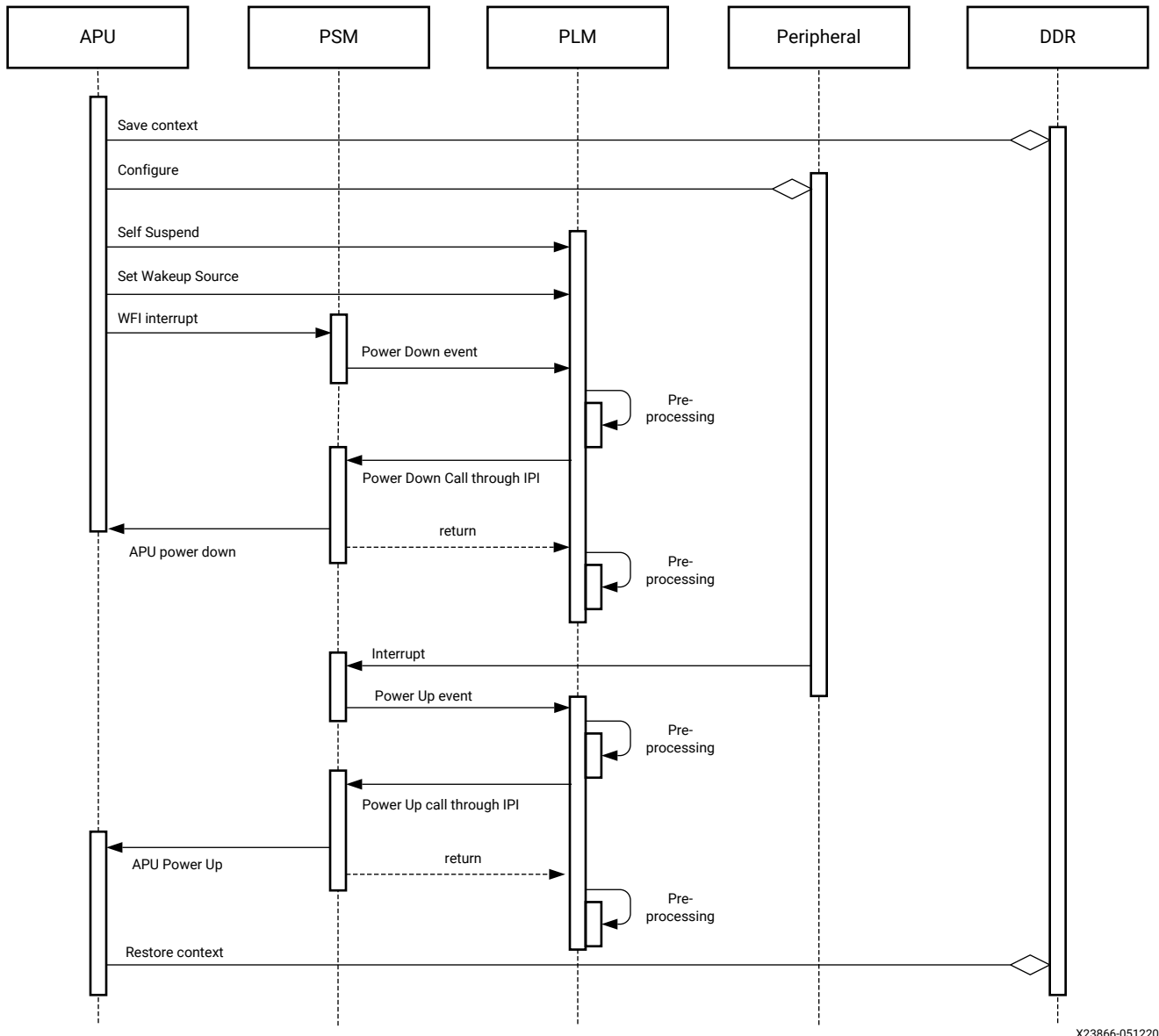
CPU は、`XPm_SelfSuspend` 呼び出しで提供されるレジューム アドレスから実行を開始します。

`XPm_RequestWakeup` および `XPm_SelfSuspend` の詳細は、『OS およびライブラリ資料コレクション』 ([UG643](#)) を参照してください。

サスペンド/レジューム フロー

次の図に、サスペンド/レジュームのユース ケースにおける各種プロセッサ間の相互通信の詳細を示します。

図 35: Versal ACAP APU のサスペンド/レジューム フロー



X23866-051220

この例では、ペリフェラルをウェイクアップソースとして設定しています。次に、各ステップについて説明します。

• サスペンド フロー:

1. APU が CPU コンテキストを DDR に保存します。
2. ペリフェラルをウェイクアップソースとして使用するように設定します。
3. APU が ATF または XilPM クライアント (APU ベアメタル アプリケーションを実行している場合) を使用して、PMC にサスペンドの意図を通知します。PLM が WFI 割り込みを有効にします。
4. PLM にウェイクアップソースを通知します。
5. APU が WFI ステートに移行して、PSM に割り込みを送信します。APU がベアメタル アプリケーションを実行している場合、XPm_SuspendFinalize API を使用して PSM に割り込みを送信できます。

6. 割り込みを受信すると、PSM と PLM が IPI を利用してハンドシェイクを実行します。PSM は、パワーゲーティングを実行し、APU へのリセットをアサートし、APU をクロックゲーティングしてパワーダウンします。
- ウェークアップフロー:
 1. APU がウェークアップソースに指定したペリフェラルが PSM に割り込みを送信します。
 2. PSM と PLM が APU のパワーアップ開始のためのハンドシェイクを実行します。
 3. ハンドシェイクの後、PSM が APU をパワーアップします。APU に対するパワーゲーティングを無効にし、リセットをディアサートし、クロックゲーティングを無効にします。
 4. APU が DDR に保存されたコンテキストを復元します。

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 1 つも使用されていないときに PMC 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、PMC 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、PMC 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 のパワーダウンの詳細は、[セルフサスペンド](#) の Suspend to RAM を参照してください。

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 PMC で設定された適切な権限を持っている必要があります。PMC 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。



重要: 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。

XPM_ForcePowerdown の詳細は、『OS およびライブラリ資料コレクション』([UG643](#)) を参照してください。

API を使用したクロック管理

プロセッシングユニットは、EEMI API を使用してシステム内のクロックを管理できます。各クロックは、一意の ClockId で識別されます。PM マスターは、そのプロセッシングユニットからアクセス可能な ClockId の一覧を XPm_Query API を使用して取得できます。PM マスターは、クロックの属性を変更する前に、そのクロックに関連付けられたスレーブデバイスを要求する必要があります。詳細は、『OS およびライブラリ資料コレクション』([UG643](#)) の XPm_RequestNode API を参照してください。

クロックのステータスを取得および設定するには、次の API を使用します。

- XStatus XPm_ClockGetStatus(const u32 ClockId, u32 *const State)
- XStatus XPm_ClockEnable(const u32 ClockId)
- XStatus XPm_ClockDisable(const u32 ClockId)

クロック分周値の設定を変更するには、次の API を使用します。

- XStatus XPm_ClockGetDivider(const u32 ClockId, u32 *const Divider)
- XStatus XPm_ClockSetDivider(const u32 ClockId, const u32 Divider)

クロックを別の親クロックで駆動するように設定するには、次の API を使用します。

- `XStatus XPm_ClockGetParent(const u32 ClockId, u32 *const ParentId)`
- `XStatus XPm_ClockSetParent(const u32 ClockId, const u32 ParentId)`



重要: `ParentId` は、`ClockId` の親として設定可能なクロックへのインデックスです。Zynq UltraScale+ MPSoC では、Linux とスタンドアロン アプリケーションで `ParentId` の定義が異なっていました。Versal ACAP では、これらの EEMI API はいずれも `ParentId` が親として設定可能なクロックへのインデックス値に統一されています。

基準クロックのレートを設定するには、次の API を使用します。

- `int XPm_ClockGetRate(const u32 ClockId, u32 *const Rate)`
- `int XPm_ClockSetRate(const u32 ClockId, const u32 Rate)`



重要: `XPm_ClockSetRate()` は、CDO ロード中のみ有効です。XiIPM クライアント API からクロック レートを設定することはできません。

`ClockId` で識別される PLL を設定するには、次の API を使用します。

- `XStatus XPm_PllGetMode(const u32 ClockId, u32 *const Value)`
- `XStatus XPm_PllSetMode(const u32 ClockId, const u32 Value)`
- `XStatus XPm_PllGetParameter(const u32 ClockId, const enum XPm_PllConfigParams ParamId, u32 *const Value)`
- `XStatus XPm_PllSetParameter(const u32 ClockId, const enum XPm_PllConfigParams ParamId, const u32 Value)`

これら API の詳細は、『OS およびライブラリ資料コレクション』(UG643) を参照してください。

API を使用したリセット管理

PU は、EEMI API を使用してシステム内のリセット ラインを管理できます。各リセットは、一意の `ResetId` で識別されます。Linux が PM マスターの場合、利用可能な `ResetId` はデバイス ツリーから取得します。PM マスターは、リセット ラインのステートを変更する前に、そのリセットに関連付けられたスレーブ デバイスを要求する必要があります。詳細は、『OS およびライブラリ資料コレクション』(UG643) の `XPm_RequestNode` API を参照してください。

リセットを管理するには、次の API を使用します。

- `XStatus XPm_ResetGetStatus(const u32 ResetId, u32 *const State)`
 - 。ステートは 1 (asserted) または 2 (released) のいずれかです。
- `XStatus XPm_ResetAssert(const u32 ResetId, const u32 Action)`
 - 。アクションは 0 (reset_release)、1 (reset_assert)、または 2 (reset_pulse) のいずれかです。

API を使用したピン管理

PU は、EEMI API を使用してシステム内のピンを管理できます。各ピンは、一意の `PinId` で識別されます。PM マスターは、その PU からアクセス可能な `PinId` の一覧を `XPm_Query` API を使用して取得できます。

ピンの制御を要求または解放するには、次の API を使用します。

- XStatus XPm_PinCtrlRequest(const u32 PinId)
- XStatus XPm_PinCtrlRelease(const u32 PinId)

1つのピンを複数のファンクションユニットで使用するように設定できます。特定のピンにアクセス可能なファンクションユニット (FunctionId で識別) の一覧を取得するには、XPm_Query API を使用します。ピンにファンクションユニットを割り当てる前に、ピンを要求する必要があります。ほとんどのファンクションユニットは、デバイスに関連付けられています。デバイスをピンに割り当てる前に、XPm_RequestNode API を使用してデバイスを要求します。

- XStatus XPm_PinCtrlGetFunction(const u32 PinId, u32 *const FunctionId)
- XStatus XPm_PinCtrlSetFunction(const u32 PinId, const u32 FunctionId)

ピンに異なる動作特性を設定するには、次の API を使用します。利用可能な ParamId および ParamVal の一覧は、『OS およびライブラリ資料コレクション』 ([UG643](#)) を参照してください。

- XStatus XPm_PinCtrlGetParameter(const u32 PinId, const u32 ParamId, u32 *const ParamVal)
- XStatus XPm_PinCtrlSetParameter(const u32 PinId, const u32 ParamId, const u32 ParamVal)

各種 API の使用

XPm_DevIoctl EEMI API

XPm_DevIoctl EEMI API は、プラットフォーム管理マスターが指定したデバイスに対して特定の動作を実行するために使用します。

次の表に、Versal ACAP でサポートされる動作を示します。

表 28: XPm_DevIoctl の動作

| ID | 名前 | 説明 | 引数 | | | |
|----|----------------------------|------------------------------|--------------------------|---|------|-----------------------------------|
| | | | ノード ID | Arg1 | Arg2 | 戻り値 |
| 0 | IOCTL_GET_RPU_OPER_MODE | 現在の RPU の動作モードを返します。 | NODE_RPU_0 NODE_RPU_1 | - | - | 動作モード: 0: LOCKSTEP 1: SPLIT |
| 1 | IOCTL_SET_RPU_OPER_MODE | RPU の動作モードを設定します。 | NODE_RPU_0 NODE_RPU_1 | 動作モードの値 0: LOCKSTEP 1: SPLIT | - | - |
| 2 | IOCTL_RPU_BOOT_ADDR_CONFIG | RPU のブートアドレスを設定します。 | NODE_RPU_0 NODE_RPU_1 | ブートアドレスの設定値 0: LOVEC/TCM 1: HIVEC/OCM | - | - |
| 3 | IOCTL_TCM_COMB_CONFIG | TCM をスプリットモードまたは結合モードに設定します。 | NODE_RPU_0 NODE_RPU_1 | 設定値 (スプリット/結合) 0: SPLIT 1: COMB | - | - |

表 28: XPm_DevIoctl の動作 (続き)

| ID | 名前 | 説明 | 引数 | | | |
|----|------------------------------|--------------------------------------|-------------------------|---|---|-------|
| | | | ノード ID | Arg1 | Arg2 | 戻り値 |
| 4 | IOCTL_SET_TAPDELAY_BYPASS | タップ遅延バイパスを有効/無効にします。 | NODE_QSPI | タップ遅延のタイプ 2: QSPI | タップ遅延イネーブル、ディスエーブル 0: DISABLE 1: ENABLE | - |
| 6 | IOCTL_SD_DLL_RESET | SD デバイスの DLL ロジックをリセットします。 | NODE_SD_0、 NODE_SD_1 | SD DLL リセットタイプ 0: ASSERT 1: RELEASE 2: PULSE | - | - |
| 7 | IOCTL_SET_SD_TAPDELAY | SD デバイスの入力/出力タップ遅延を設定します。 | NODE_SD_0、 NODE_SD_1 | 設定するタップ遅延のタイプ 0: INPUT 1: OUTPUT | タップ遅延の設定値 | - |
| 12 | IOCTL_WRITE_GGS | GGs レジスタに値を書き込みます。 | - | GGs レジスタインデックス (0/1/2/3) | レジスタに書き込む値 | - |
| 13 | IOCTL_READ_GGS | GGs レジスタの値を返します。 | - | GGs レジスタインデックス (0/1/2/3) | - | レジスタ値 |
| 14 | IOCTL_WRITE_PGGS | PGGS レジスタに値を書き込みます。 | - | PGGS レジスタインデックス (0/1/2/3) | レジスタに書き込む値 | - |
| 15 | IOCTL_READ_PGGS | PGGS レジスタの値を返します。 | - | PGGS レジスタインデックス (0/1/2/3) | - | レジスタ値 |
| 17 | IOCTL_SET_BOOT_HEALTH_STATUS | ファームウェアにブートの健全性を通知するヘルシービットの値を設定します。 | - | ヘルシービットの値 | - | - |

表 28: XPm_DevIoctl の動作 (続き)

| ID | 名前 | 説明 | 引数 | | | |
|----|--------------------------|----------------------------------|---|---|------|-------|
| | | | ノード ID | Arg1 | Arg2 | 戻り値 |
| 19 | IOCTL_PROBE_COUNTER_READ | LPD/FPD のプローブ カウンター レジスタを読み出します。 | FPD/LPD 電源ドメイン ID LPD: 0x4210002U FPD: 0x420C003U | レジスタ設定 - カウンター番号 (ビット 0 ~ 7) - レジスタタイプ (ビット 8 ~ 15) 0 - LAR_LSR アクセス (要求タイプとカウンター番号を無視) 1 - メイン制御 (カウンター番号を無視) 2 - コンフィギュレーション制御 (カウンター番号を無視) 3 - ステート周期 (カウンター番号を無視) 4 - PortSel 5 - Src 6 - Val - 要求タイプ (ビット 16 ~ 23) 0 - 読み出し要求 1 - 読み出し応答 2 - 書き込み要求 3 - 書き込み応答 4 - lpd 読み出し要求 (LPD のみ) 5 - lpd 読み出し応答 (LPD のみ) 6 - lpd 書き込み要求 (LPD のみ) 7 - lpd 書き込み応答 (LPD のみ) | - | レジスタ値 |

表 28: XPm_DevIoctl の動作 (続き)

| ID | 名前 | 説明 | 引数 | | | |
|----|---------------------------|----------------------------------|---|--|------------|----------------------------|
| | | | ノード ID | Arg1 | Arg2 | 戻り値 |
| 20 | IOCTL_PROBE_COUNTER_WRITE | LPD/FPD のプローブ カウンター レジスタに書き込みます。 | FPD/LPD 電源ドメイン ID LPD: 0x4210002U FPD: 0x420C003U | レジスタ設定 - カウンター番号 (ビット 0 ~ 7) - レジスタタイプ (ビット 8 ~ 15) 0 - LAR_LSR アクセス (要求タイプとカウンタ番号を無視) 1 - メイン制御 (カウンタ番号を無視) 2 - コンフィギュレーション制御 (カウンタ番号を無視) 3 - ステート周期 (カウンタ番号を無視) 4 - PortSel 5 - Src - 要求タイプ (ビット 16 ~ 23) 0 - 読み出し要求 1 - 読み出し応答 2 - 書き込み要求 3 - 書き込み応答 4 - lpd 読み出し要求 (LPD のみ) 5 - lpd 読み出し応答 (LPD のみ) 6 - LPD 書き込み要求 (LPD のみ) 7 - LPD 書き込み応答 (LPD のみ) | レジスタに書き込む値 | - |
| 21 | IOCTL_OSPI_MUX_SELECT | OSPI AXI マルチプレクサーを選択します。 | NODE_OSPI | 動作モード 0: DMA を選択 1: リニアを選択 2: モードを取得 | - | モードを取得 0: DMA 1: リニア |

表 28: XPm_DevIoctl の動作 (続き)

| ID | 名前 | 説明 | 引数 | | | |
|----|-----------------------------|-------------------------------|------------------------|---|------|--|
| | | | ノード ID | Arg1 | Arg2 | 戻り値 |
| 22 | IOCTL_USB_SET_STATE | USB コントローラーのデバイス電力ステートを設定します。 | NODE_USB_0 | 要求された電力ステート 0: D0 1: D1 2: D2 3: D3 | - | - |
| 23 | IOCTL_GET_LAST_RESET_REASON | 最後のシステムリセットの要因を取得します。 | - | - | - | 0 - システム外部で POR ボタンが押された 1 - ソフトウェアによる内部 POR が発生した 2 - ほかのいずれか 1 つの SSIT スライスによって POR が発生した 3 - エラーにより POR が発生した 7 - JTAG TAP によりシステムリセットが開始された 8 - エラーによりシステムリセットが開始された 9 - ソフトウェアによりシステムリセットが開始された 10 - ほかのいずれか 1 つの SSIT スライスによってシステムリセットが発生した 15 - 無効なリセット要因 |
| 24 | IOCTL_AIE_ISR_CLEAR | AIE NPI 割り込みをクリアします。 | DEV_AIE (0x18224072 U) | 4 ビット NPI 割り込みクリアマスク (wtc) ビット <3-0> が割り込み <3-0> に対応 | - | - |

XPm_GetOpCharacteristic EEMI API

XPm_GetOpCharacteristic EEMI API は、PM マスターが特定のコンポーネントの動作特性に関する情報を返すように PMC に要求するために使用します。現時点では、次のデバイス特性を要求できます。

- SoC の温度
- コア、電源アイランド/レール、および PLL ロック時間のウェークアップ レイテンシ

この API には、アプリケーションが `XPm_RegisterNotifier` API を使用して通知を登録できるようにする機能もあります。この API を使用すると、PU は特定の条件を満たすイベントが発生した場合に必ず `Notify Callback` を呼び出すように PMC に要求できます。これにより、特定のノードに関連する特定の、またはすべてのイベントについて通知を受けるように要求できます。コールバック関数は、次の 2 つのイベントが発生すると呼び出されます。

- コールバックを登録したノードのステートが変化した
- コールバックを登録したノードのユーザーが 0 になった

また、コールバックを登録したノードに関連するイベントについて通知を受信する必要がなくなった場合は、登録を解除できます。

注記: 現時点では、イベント ノードとデバイス ノードのみがサポートされます。

詳細は、『OS およびライブラリ資料コレクション』 ([UG643](#)) を参照してください。

XPm_Query EEMI API

`XPm_Query` EEMI API は、プラットフォーム管理マスターが PLM に対して特定のコンフィギュレーション情報を要求するために使用します。

次の表に、Versal™ ACAP でサポートされる動作を示します。

表 29: `XPm_Query` の動作

| ID | 名前 | 説明 | 引数 | | |
|----|---|-----------------------------------|---------|-----------------|---|
| | | | Arg1 | Arg2 | 戻り値 |
| 1 | <code>XPM_QID_CLOCK_GET_NAME</code> | 指定したクロック ID に関連する名前 (文字列) を取得します。 | クロック ID | - | データ [0]: 成功/エラー データ [1-4]: 名前 (文字列) |
| 2 | <code>XPM_QID_CLOCK_GET_TOPOLOGY</code> | クロックのトポロジを取得します。 | クロック ID | トポロジ ノード インデックス | データ [0]: 成功/エラー データ [1-3]: 引数 2 で指定したノード インデックスを始点にした 3 ノードのトポロジ |
| 3 | <code>XPM_QID_CLOCK_GET_FIXEDFACTOR_PARAMS</code> | 固定係数の値を取得します。 | クロック ID | - | データ [0]: 成功/エラー データ [1]: 固定係数の値 |

表 29: XPM_Query の動作 (続き)

| ID | 名前 | 説明 | 引数 | | |
|----|---|----------------------------------|------------|-------------|---|
| | | | Arg1 | Arg2 | 戻り値 |
| 4 | XPM_QID_CLOCK_GET_MUXSOURCES | クロックのマルチプレクサーソースを取得します。 | クロック ID | 親ノード インデックス | データ [0]: 成功/エラー データ [1-3]: 引数 2 で指定した親ノード インデックスを始点にしたクロックの親 ID |
| 5 | XPM_QID_CLOCK_GET_ATTRIBUTES | クロックの属性を取得します。 | クロック ID | - | データ [0]: 成功/エラー データ [1] ビット (0): 有効/無効クロック データ [1] ビット (1): 最初のイネーブル要件 データ [1] ビット (2): クロックタイプ (出力/外部) データ [1] ビット (14:19): クロック ノード タイプ データ [1] ビット (20-25): クロック ノード サブクラス データ [1] ビット (26:31): クロック ノード クラス |
| 6 | XPM_QID_PINCTRL_GET_NUM_PINS | コンフィギュレーションに利用可能なピンの数を取得します。 | - | - | データ [0]: 成功/エラー データ [1]: ピンの数 |
| 7 | XPM_QID_PINCTRL_GET_NUM_FUNCTIONS | 利用可能なファンクションユニットの総数を取得します。 | - | - | データ [0]: 成功/エラー データ [1]: ファンクションの数 |
| 8 | XPM_QID_PINCTRL_GET_NUM_FUNCTION_GROUPS | 指定したファンクション ID が属するグループの数を取得します。 | ファンクション ID | - | データ [0]: 成功/エラー データ [1]: グループの数 |

表 29: XPM_Query の動作 (続き)

| ID | 名前 | 説明 | 引数 | | |
|----|-------------------------------------|-------------------------------------|------------|--------|---|
| | | | Arg1 | Arg2 | 戻り値 |
| 9 | XPM_QID_PINCTRL_GET_FUNCTION_NAME | ファンクションユニットに関連する名前 (文字列) を取得します。 | ファンクション ID | - | データ [0]: 成功/エラー データ [1-3]: 名前 (文字列) |
| 10 | XPM_QID_PINCTRL_GET_FUNCTION_GROUPS | 指定したファンクション ID が属するグループ ID を取得します。 | ファンクション ID | インデックス | データ [0]: 成功/エラー データ [1-3]: 引数 2 で指定したインデックスを始点にした 6 つのグループ (各グループは 16 ビット) |
| 11 | XPM_QID_PINCTRL_GET_PIN_GROUPS | 指定したピン ID が属することのできるグループ ID を取得します。 | ピン ID | インデックス | データ [0]: 成功/エラー データ [1-3]: 引数 2 で指定したインデックスを始点にした 6 つのグループ (各グループは 16 ビット) |
| 12 | XPM_QID_CLOCK_GET_NUM_CLOCKS | クロックの数を取得します。 | - | - | データ [0]: 成功/エラー データ [1]: クロックの数 |
| 13 | XPM_QID_CLOCK_GET_MAX_DIVISOR | 指定したクロックの最大分周値を取得します。 | クロック ID | - | データ [0]: 成功/エラー データ [1]: 最大分周値 |
| 14 | XPM_QID_PLD_GET_PARENT | PL デバイス ノードの親を取得します。 | PL デバイス ID | - | データ [0]: 成功/エラー データ [1]: PL デバイスの親 |

XPM_ActivateSubsystem API

XSDB マスターは、XPM_ActivateSubsystem API を使用してサブシステムの有効化を PMC に要求します。詳細は、[サブシステムの有効化](#) を参照してください。

この API は、あるサブシステムの動作に必要なすべての事前割り当て済みデバイスを要求することにより、サブシステムを有効化します。この API は、有効化する必要のあるターゲット サブシステムの ID を入力引数にとります。サブシステム有効化のフォーマットは、次のとおりです。

```
XPm_ActivateSubsystem(u32 Subsystem ID)
```

このコマンドは、XSDB マスターからのみ実行できます。現在はデフォルトのサブシステムしかサポートされていないため、特定のマスターにアプリケーション バイナリをダウンロードする前に、デフォルトのサブシステムが自動的に有効化されます。

XiIPM クライアント実装の詳細

プラットフォーム管理のシステム層は、IPI を使用して Versal ACAP に実装します。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、PMC に対して IPI をトリガーします。

PMC は要求を処理した後、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。

PMC への API 呼び出しのペイロード マップ

EEMI API 呼び出しは、次のデータで一意的に識別されます。

- EEMI API 識別子 (ID)
- EEMI API の引数

API 識別子と API 引数の値の一覧は、『OS およびライブラリ資料コレクション』 ([UG643](#)) を参照してください。

PMC への IPI を開始する前に、PU は呼び出しに関する情報を IPI 要求バッファへ書き込みます。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

PMC からの API コールバックのペイロード マップ

EEMI API には、PM 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 コールバック ID
- Word [1:5] EEMI API の引数

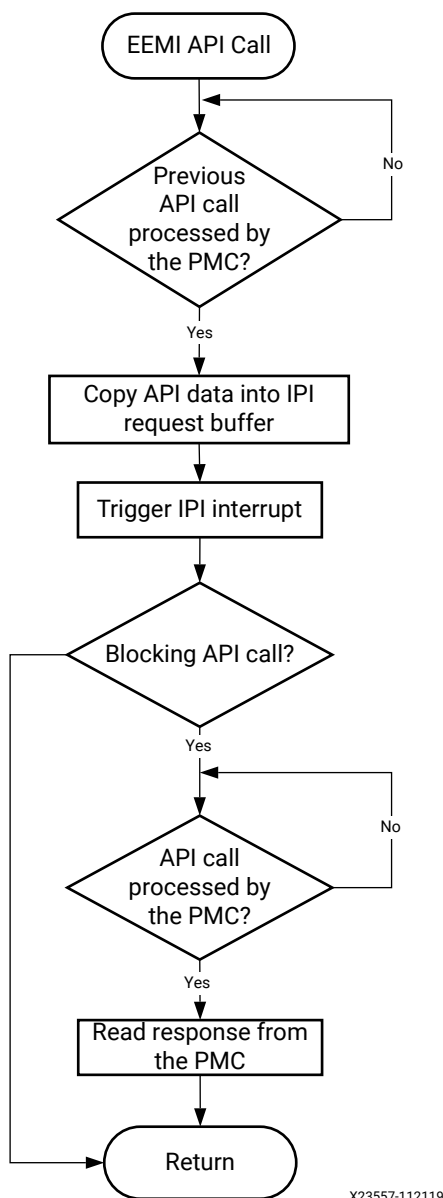
API 識別子と API 引数の値の一覧は、『OS およびライブラリ資料コレクション』 ([UG643](#)) を参照してください。

PMC への EEMI API 呼び出しの実行

PMC への API 呼び出しを実行する前に、PU は前の API 呼び出しが PMC で完了されるまで待機する必要があります。PMC の動作完了を確認するには、対応する IPI 観察レジスタを読み出します。

IPI ペイロード バッファに API データを書き込み、PMC への IPI 割り込みをトリガーして、API 呼び出しを実行します。ブロッキング API 呼び出しの場合は、PMC 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 PMC 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、PMC 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。

図 36: PMC への API 呼び出し実行フローの例



PMC からの API コールバックの処理

PMC 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対して割り込みは 1 つです。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。

XiIPM ライブラリには、この動作のインプリメンテーションが含まれます。

Linux における PM 機能

Linux は EL1 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ており、Linux で実行される電力管理機能は PMC によって提供される EEMI サービスを利用できます。

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。

ユーザー空間のプラットフォーム管理インターフェイス

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の変更を要求できます。プラットフォーム管理によって、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。

サスペンド

CPU およびほとんど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。

- 必要なカーネル設定:
 - 電力管理オプション
 - [*] Suspend to RAM and standby
 - [*] User space wakeup sources interface 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
 - 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 ザイリンクス Zynq MPSoC firmware interface

注記: 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。

- コード例: カーネルをサスペンドする場合:

```
$ echo mem > /sys/power/state
```

https://wiki.archlinux.org/index.php/Power_management/Suspend_and_hibernate も参照してください。

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。

- 必要なカーネル設定: [サスペンド](#) と同じ
- コード例: 10 秒後に APU をウェークアップするように RTC を設定する場合:

```
$ echo enabled > /sys/devices/platform/amba/ff000000.serial/tty/ttyAMA0/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。

- 必要なカーネル設定: [サスペンド](#) と同じ
- コード例: UART 入力で APU を起動する場合:

```
$ echo enabled > /sys/devices/platform/amba/ff000000.serial/tty/ttyAMA0/power/wakeup
```

- 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。

- 必要なカーネル設定:
 - 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

- コード例: 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

CPU ホットプラグ制御インターフェイスを使用すると、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。次のカーネル設定が必要です。

- カーネルの機能
 - [*] Support for hot-pluggable CPUs

たとえば CPU1 をオフラインにするには、次のコマンドを実行します。

```
echo 0 > /sys/devices/system/cpu/cpu1/online
```

詳細は、次の資料を参照してください。

- <https://www.kernel.org/doc/Documentation/cpu-hotplug.txt>
- <http://lxr.free-electrons.com/source/Documentation/devicetree/bindings/arm/idle-states.txt>

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が個々の APU コアの電力を削減することがあります。次のカーネル設定が必要です。

- CPU 電力管理
- CPU アイドル
- [*] CPU idle PM support
- [*] Ladder governor (for periodic timer tick)
- -* Menu governor (for tickless system)
- Arm CPU アイドル ドライバー
 - [*] Generic Arm/Arm64 CPU idle Driver
 - [*] PSCI CPU idle Driver
- 例 1: 次のコード例は、cpuidle の sysfs インターフェイスです。

```
$ 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)
- 例 2: 次のコード例は、cpuidle ガバナーの sysfs インターフェイスです。

```
$ 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 */
```

詳細は、次の資料を参照してください。

- <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>

CPU 周波数

この機能を有効にすると、CPU コアは動作クロック周波数を切り替えることができます。

- 必要なカーネル設定:
 - CPU 周波数スケールリング
 - [*] 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

- コマンド例: 次に、CPU 周波数スケーリングに関するコマンドをいくつか示します。

- 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 周波数の追加と変更についての詳細は、[汎用 OPP に関する Linux カーネル資料](#) 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します (共通クロック フレームワーク)。

次のカーネル設定が必要です。

- 共通クロック フレームワーク
 - [*] Support for ザイリンクス Zynq MP 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します。

注記: 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ります。

次のカーネル設定が必要です。

- 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
 - [*] ザイリンクス Zynq MPSoC driver support

グローバル汎用ストレージ レジスタ

汎用の 32 ビット ストレージ レジスタは、4 つあります。これらレジスタの値は、リブートによって失われます。次の表に、グローバル汎用ストレージ レジスタを示します。

表 30: グローバル汎用ストレージ レジスタ

| デバイス ノード | 有効な値の範囲 |
|---------------------------|-------------------------|
| /sys/firmware/zynqmp/ggs0 | 0x00000000 ~ 0xFFFFFFFF |

表 30: グローバル汎用ストレージ レジスタ (続き)

| デバイス ノード | 有効な値の範囲 |
|---------------------------|-------------------------|
| /sys/firmware/zynqmp/ggs1 | 0x00000000 ~ 0xFFFFFFFF |
| /sys/firmware/zynqmp/ggs2 | 0x00000000 ~ 0xFFFFFFFF |
| /sys/firmware/zynqmp/ggs3 | 0x00000000 ~ 0xFFFFFFFF |

グローバル ストレージ レジスタの値は、次の方法で読み出します。

```
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は、次の方法で書き込みます。

```
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 ビットグローバル汎用ストレージ レジスタが 4 つあります。これらレジスタの値は、ソフトウェア再起動の後も維持されます。次の表に、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 31: 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 | 有効な値の範囲 |
|----------------------------|------------------------------------|---------------------------------|
| /sys/firmware/zynqmp/pggs0 | PMC_GLOBAL_PERS_GLOB_GEN_STORA GE3 | 0x00000000 を 0xFFFFFFFF に変更します。 |
| /sys/firmware/zynqmp/pggs1 | PMC_GLOBAL_PERS_GLOB_GEN_STORA GE4 | 0x00000000 を 0xFFFFFFFF に変更します。 |
| /sys/firmware/zynqmp/pggs2 | PSM_GLOBAL_PERS_GLOB_GEN_STORA GE0 | 0x00000000 を 0xFFFFFFFF に変更します。 |
| /sys/firmware/zynqmp/pggs3 | PSM_GLOBAL_PERS_GLOB_GEN_STORA GE1 | 0x00000000 を 0xFFFFFFFF に変更します。 |

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方法:

```
cat /sys/firmware/zynqmp/pggs0
```

永続的なグローバル ストレージ レジスタのマスクと値を書き込む方法:

```
echo 0xFFFFFFFF 0x1234ABCD > /sys/firmware/zynqmp/pggs0
```

次のレジスタは予約です。



重要: ザイリンクスは、予約済みレジスタの使用を推奨していません。

表 32: 予約済みのストレージ レジスタ

| レジスタ | 説明 |
|---|---|
| PMC_GLOBAL_GLOBAL_GEN_STORAGE0、 PMC_GLOBAL_GLOBAL_GEN_STORAGE1 | ROM 実行タイムスタンプを格納します。PLM がアクティブな場合、これら 2 つのレジスタを読み出して ROM の実行時間を取得します。ブート PDI をロード後、レジスタを使用できます。 |
| PMC_GLOBAL_GLOBAL_GEN_STORAGE2 | デバイスのセキュリティ ステータスを格納し、ROM によって更新されます。PLM はこのレジスタを使用して KAT を実行する必要があるかどうかを判断します。 |
| PMC_GLOBAL_GLOBAL_GEN_STORAGE4 | PLM は、このレジスタを使用して ATF ハンドオフ パラメーターのアドレス ポインターを格納します。 |
| PMC_GLOBAL_PERS_GLOB_GEN_STORAGE0 | XilPM が各電源ドメインの初期化ステータスを保存するための予約レジスタ。 |
| PMC_GLOBAL_PERS_GLOB_GEN_STORAGE1 | PLM とデバッガーでデータを共有することを目的としたレジスタですが、現在の PLM では使用しません。 |

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。Zynq MP debugfs インターフェイスはデフォルトで有効になっています。

- 必要なカーネル設定:
 - 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 ザイリンクス Zynq MPSoC firmware interface
 - [*] Enable ザイリンクス 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)
- APU のウェークアップ要求 (Request Wake-up the APU)
- 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)

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 (string 型) または ID 番号 (integer 型) のいずれかです。

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、『エンベデッド エネルギー管理インターフェイス EEMI API リファレンス ガイド』(UG1200)を参照してください。

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
```

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
```

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。

```
echo pm_release_node <node> > /sys/kernel/debug/zynqmp-firmware/pm
```

注記: 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。

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

指定したピンに現在設定されているファンクションを取得します。

```
echo pm_pinctrl_get_function <pin-number> > /sys/kernel/debug/zynqmp-firmware/pm
```

Set Pin Control Functions

指定したピンに特定のファンクションを設定します。

```
echo pm_pinctrl_set_function <pin-number> <function-id> > /sys/kernel/debug/zynqmp-firmware/pm
```

Get Configuration Parameters for the Pin

指定したピンの特定のコンフィギュレーション 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

指定したピンの特定のコンフィギュレーション 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

デバイスとコンフィギュレーションを制御し、コンフィギュレーションの値を取得します。

```
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 のクロックの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 のクロックの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 のクロックの分周値を取得します。

```
echo pm_clock_setdivider <clock id> <divider value> >
/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_setrate <clock id> <clock rate> >
/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,versal-clk.txt

Self Suspend

この PU がセルフ サスペンドすることを PMC に通知します。

```
$ echo pm_self_suspend <node> > /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
```

PM Linux ドライバー

Linux 用 Versal ACAP 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。

PM ファームウェア ドライバー

関数宣言は、次の場所にあります。include/linux/firmware/xlnx-zynqmp.h

実装した関数は、次の場所にあります。drivers/firmware/xilinx/zynqmp/zynqmp*.c

ドライバーを適切に初期化するには、Linux デバイス ツリーで正しいノードを指定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 PLM 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,versal-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 {
    versal_firmware: versal-firmware {
        compatible = "xlnx,versal-firmware";
        method = "smc";
    };
};
```

注記: 電力管理ドライバーと電力ドメイン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/power/reset/xilinx/xlnx,zynqmp-power.txt
- Documentation/devicetree/bindings/power/reset/xilinx/xlnx,zynqmp-genpd.txt

Linux からのデバイスの要求とリリース

Linux ZynqMP (https://github.com/torvalds/linux/blob/master/drivers/soc/xilinx/zynqmp_pm_domains.c 参照) は、デバイス使用の要求、要件の設定、およびデバイスのリリースを管理します。

各デバイス ノードの電源ドメイン ID としてデバイス ID を指定する必要があります。zynqmp-firmware ノードは電源ドメイン プロバイダーとして動作します。次に例を示します。

```
firmware {
    zynqmp_firmware: zynqmp-firmware {
        ...
        #power-domain-cells = <1>;
        ...
    };
};
usb0 {
    ...
    power-domains = <&zynqmp_firmware 0x18224018U>;
    ...
};
```

デバイスが使用状態にある場合、Linux ZynqMP 電源ドメイン ドライバーは PM_CAP_ACCESS をセットします。実行時サスペンドの間、電源ドメイン ドライバーは要件を PM_CAP_UNUSABLE (クロックのみ停止) に設定します。デバイスが使用されていないと、電源ドメイン ドライバーはデバイスをリリースします。この場合、ほかのサブシステムがこのデバイスを使用していなければ、プラットフォーム管理ファームウェアはパワー ゲーティングを実行してクロックを停止し、デバイスをリセットします。

Arm トラステッド ファームウェア

Arm トラステッド ファームウェア (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、PMC 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进行管理します。

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 PMC と間接的に通信することがあります。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、PMC 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、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ます。ハイパーバイザー、セキュア OS、または OS 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 ノード进行管理する前に、ATF に実装されている EEMI API バージョンが PMC 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 PMC 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 PMC から EEMI API バージョンを取得できます。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 関数はバージョンに依存しません。つまり、すべての EEMI バージョンにこの関数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 を使用して PM_GET_CHIPID 関数を呼び出すことで、PMC 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。

- TAP idcode レジスタ
- TAP version レジスタ

詳細は、『Versal ACAP テクニカル リファレンス マニュアル』 ([AM011](#)) を参照してください。

PSCI (Power State Coordination Interface)

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。

次の表に、ATF でサポートされる PSCI v0.2 の関数を示します。

表 33: ATF でサポートされる PSCI v0.2 の関数

| 機能 | 説明 | サポート |
|-----------------------------|---|------|
| PSCI Version | 実装されている 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る実行状態への復帰を待機しているアイドル状態のサブシステムで使用。 | あり |
| 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 |

表 33: ATF でサポートされる PSCI v0.2 の関数 (続き)

| 機能 | 説明 | サポート |
|-------------------------------|---|------|
| 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ポロジ内のノードの真のハードウェア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 |
| PSCI Stat Residency (オプション) | PSCI v1.0 で導入。 コールド ブートにより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 ブートにより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 |

PS 管理コントローラー ファームウェア

PS 管理コントローラー (PSM) は 1 つの独立した MicroBlaze プロセッサで、PS ドメインに三重モジュール式冗長 MicroBlaze として実装されています。PSM 上で動作する PSM ファームウェアが PS の電源アイランドおよび電源ドメインを管理します。PMC は、PSM ファームウェアに実装された関数を利用して PS 電源アイランドおよびドメインをパワーアップまたはパワーダウンします。これらの関数は PLM からのみアクセスできます。

EEMI 要求によって電源アイランドまたは電源ドメインのステートが変化した場合、PLM は PSM の割り込みハンドラーをトリガーして、その動作を処理します。PSM には、動作完了時に PLM と通信するための IPI チャンネルも割り当てられています。

PLM ファームウェアとの関係

PLM には、PM コントローラーと呼ばれるモジュールの 1 つとして EEMI サービス ハンドラーが実装されています (PLM では、これ以外にもいくつかのモジュールが動作し、種類の異なるサービスを実行しています)。詳細は、[第 7 章: ブートおよびコンフィギュレーション](#) を参照してください。

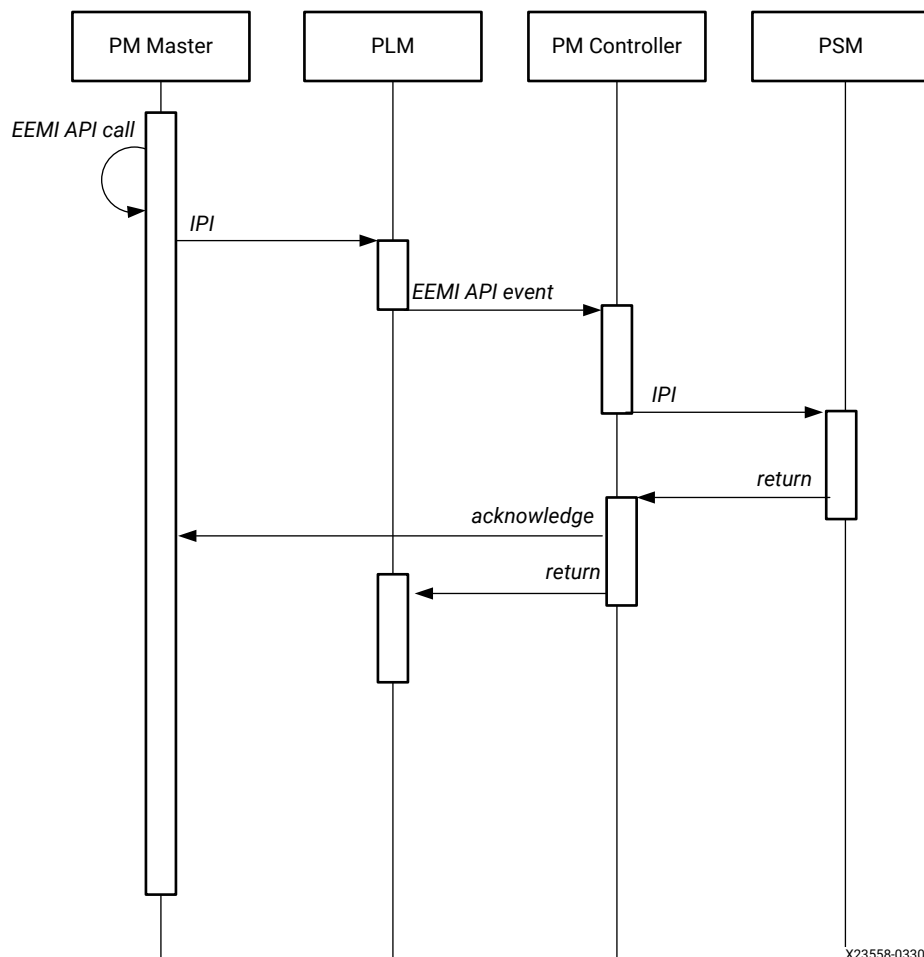
Embedded Energy Management Interface (EEMI)

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 PMC 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。EEMI 要求は、PM コントローラーが処理します。多くの場合、EEMI 要求によって電源ステートの変更、またはリソース設定の変更がトリガーされます。

EEMI API 呼び出しの一般的フロー

次の図に、一般的な API 呼び出しシーケンスを示します。最初に、PM マスター (別のプロセッシング ユニットなど) が呼び出しを開始します。

図 37: EEMI API 呼び出しシーケンス



X23558-033020

この図には 4 つのアクターが示されています。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、PLM ファームウェア、PMC、および PSM です。

まず最初に PLM 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、XilPM サーバーに IPI 引数が渡されます。次に、XilPM サーバーが API 呼び出しを処理します。必要に応じて、XilPM サーバーが PSM ファームウェアを呼び出して、電源アイランドや電源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できます。

ターゲット開発プラットフォーム

この章では、Versal ACAP 向けに提供されるボードやキットなどの各種開発プラットフォームについて説明します。

次のプラットフォームが提供されています。

- AI コア シリーズ: 中集積度のプログラマブル ロジック (PL)、コネクティビティ 機能、および AI/DSP アクセラレーション エンジンを組み合わせた高性能演算向けシリーズです。
- プライム シリーズ: 中集積度の PL、信号処理、およびコネクティビティ 機能を備えたミッドレンジ シリーズです。
- プレミアム シリーズ: 充実したネットワーク インターフェイス、セキュリティ エンジン、および高い演算密度を備えた広帯域のハイエンド シリーズです。

ボードおよびキット

ザイリンクスは、開発者向けに Versal ACAP デバイス評価キットを提供しています。

現在、Versal ACAP デバイスの評価キットには次の 2 つがあります。

- VCK190: AI コア シリーズの評価キット。VCK190 ボードの詳細は、『VCK190 評価ボード ユーザー ガイド』([UG1366](#))を参照してください。
- VMK180: プライム シリーズの評価キット。VMK180 ボードの詳細は、『VMK180 評価ボード ユーザー ガイド』([UG1411](#))を参照してください。

ライブラリ

次に示す Versal ACAP ライブラリの API リファレンスについての情報は、『OS およびライブラリ資料コレクション』([UG643](#)) を参照してください。

- XilFFS
- XilMailbox
- XilPM
- XilSEM

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。

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 Hub View] タブをクリックします。
- ザイリンクス ウェブサイトで[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。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

次の文書は、このユーザー ガイドの補足資料として役立ちます。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。

ザイリンクス参考資料

1. [PetaLinux ツール](#)

2. [ザイリンクス Vivado Design Suite – HLx Edition](#)
3. [ザイリンクス サードパーティ ツール](#)
4. [エンベデッド ソフトウェア/エコシステム](#)

デバイス資料

1. 『Versal ACAP AI エンジン アーキテクチャ マニュアル』 (AM009: [英語版](#)、[日本語版](#))
2. 『Versal ACAP テクニカル リファレンス マニュアル』 ([AM011](#))
3. 『Versal アーキテクチャおよび製品データシート: 概要』 (DS950: [英語版](#)、[日本語版](#))
4. 『Versal ACAP Programmable Network on Chip および Integrated Memory Controller LogiCORE IP 製品ガイド』 ([PG313](#))
5. 『OS およびライブラリ資料コレクション』 ([UG643](#))
6. 『Versal ACAP AI エンジン プログラミング環境ユーザー ガイド』 ([UG1076](#))
7. 『エンベデッド エネルギー管理インターフェイス EEMI API リファレンス ガイド』 ([UG1200](#))
8. 『Versal ACAP デザイン ガイド』 (UG1273: [英語版](#)、[日本語版](#))
9. 『Bootgen ユーザー ガイド』 (UG1283: [英語版](#)、[日本語版](#))
10. 『ザイリンクス エンベデッド デザイン チュートリアル: Versal Adaptive Compute Acceleration Platform』 ([UG1305](#))
11. 『VCK190 評価ボード ユーザー ガイド』 ([UG1366](#))
12. 『Versal ACAP QEMU ユーザー ガイド』 (UG1372)
13. 『VMK180 評価ボード ユーザー ガイド』 (UG1411)

ツールおよび PetaLinux 資料

1. 『Zynq UltraScale+ MPSoC: ソフトウェア開発者向けガイド』 ([UG1137](#))
2. 『PetaLinux ツール資料: リファレンス ガイド』 (UG1144: [英語版](#)、[日本語版](#))
3. 『Bootgen ユーザー ガイド』 (UG1283: [英語版](#)、[日本語版](#))
4. 『Vitis 統合ソフトウェア プラットフォームの資料』 ([UG1416](#))

その他のリンク

1. <https://github.com/Xilinx/linux-xlnx/>
2. https://github.com/torvalds/linux/blob/master/drivers/soc/xilinx/zynqmp_pm_domains.c
3. <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842279/Build+Device+Tree+Blob>
4. <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841996/Linux#Linux-XilinxLinux>
5. <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842250/PetaLinux>
6. <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842316/Linux+Prebuilt+Images>
7. <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841718/OpenAMP>
8. <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842530/XEN+Hypervisor>
9. <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841883/Yocto>

サードパーティのリソース

1. <https://developer.arm.com/docs/ddi0487/latest/arm-architecture-reference-manual-armv8-for-armv8-a-architecture-profile>
2. <https://www.yoctoproject.org/software-overview/downloads/>
3. <https://git.yoctoproject.org/cgi/cgit.cgi/meta-xilinx/>
4. <https://www.kernel.org/doc/Documentation/devicetree/bindings/arm/psci.txt>
5. <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0022c/index.html>
6. <https://www.kernel.org/doc/Documentation/cpuidle/core.txt>
7. <https://www.kernel.org/doc/Documentation/cpuidle/driver.txt>
8. <https://www.kernel.org/doc/Documentation/cpuidle/governor.txt>
9. <https://www.kernel.org/doc/Documentation/cpuidle/sysfs.txt>
10. <https://www.kernel.org/doc/Documentation/cpuidle/sysfs.txt>
11. <https://www.kernel.org/doc/Documentation/devicetree/bindings/opp/opp.txt>
12. <http://lxr.free-electrons.com/source/Documentation/devicetree/bindings/arm/idlestates.txt>
13. <https://www.kernel.org/doc/Documentation/cpu-hotplug.txt>
14. https://wiki.archlinux.org/index.php/Power_management/Suspend_and_hibernate

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) ものとします。また、(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) に関係し、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、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。

この文書は暫定的な情報を含むものであり、通知なしに内容が変更されることがあります。この文書に記述される情報は、販売前の製品・サービスに関するもので、情報目的としてのみ提供されており、この文書で参照されている製品・サービスの販売申込みまたは製品の商品化を試みたものとしては意図されておらず、また解釈されるものでもありません。

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、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、

Copyright

© Copyright 2020 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。AMBA、AMBA Designer、Arm、ARM1176JZ-S、CoreSight、Cortex、PrimeCell、Mali、および MPCore は、EU およびその他の各国の Arm Limited の商標です。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。