# BIOS に代わるプラットフォーム・ファームウェアを あらゆるインテル® シリコンに

(日本語参考訳)

Vincent Zimmer
Staff Engineer
Enterprise Platforms Group
Intel Corporation

# 目次

(ページ番号をクリックすると、該当のセクションにジャンプします。)

# BIOS に代わるプラットフォーム・ファームウェアをあらゆるインテル® シリコンに

**Vincent Zimmer**
Staff Engineer
Enterprise Platforms Group
Intel Corporation

## 概要：BIOS に代わるもの

1980年代から PC プラットフォームは急速に進歩してきました。今や、パフォーマンス、使いやすさ、ストレージ容量、接続性などは当時に比べ飛躍的に向上しています。しかし PC の中でこの23年間まったく変わっていない部分が1つあります。それが、BIOS (Basic Input/Output System) です。

インテル® Platform Innovation Framework for EFI (Extensible Firmware Interface) は、従来の BIOS に代わるものとして、ブートの高速化をはじめ、マネージャビリティやその他の機能性の向上をもたらします。

## 背景知識

BIOS であれ、この新しいフレームワークをベースにしたファームウェアであれ、ブート・ファームウェアの役割とは、ブート前には単なるハードウェアの集合体として存在しているシステムを、ブート後に完全なシステムとして機能させることにあります。チップやボードは電源投入時に初期化されていない状態で起動しますが、製造コストを考えるとこれは今後も予測可能な将来にわたって変わらないものと考えられます。このため、これらのコンポーネントを使って構築したシステムは、リセット時には全体的にプリミティブな状態にならざるをえません。

こうしたシステムにおいてオペレーティング・システムを立ち上げたり、(特にブート・プロセスの初期段階において) オペレーティング・システムにサービスを提供したり、システムのマネージャビリティに関するデータを提供したりできるようにシステムの準備を整えるには、ブート・ファームウェアが大きな役割を果たします。

## 現在の問題点

まず最初に、現在のシステムで使われている BIOS の役割について見てみましょう。BIOS はプラットフォーム上の不揮発性ストレージに格納されており、システムのリスタート時に実行を開始します。ここで、BIOS はシステムの初期化を行いますが、これは一般に POST (Power-On Self Test) と呼ばれます。

通常、BIOS の POST 処理は静的にリンクされたモノリシックな16ビット、リアル・モードのアセンブリ言語で作成されており、コード実行用にごくわずかな領域しか確保されていません。アセンブリ言語で記述しなければならないこと、近代的なメモリ・マネージャのような一貫したシステム・サービスが存在しないこと、そして実行スペースが限られていることなどが理由となって、アルゴリズムや機能の開発は困難な作業となっています。

POST が完了すると、オペレーティング・システム (OS) が起動し、OS にサービスを提供できるようになります。ここでは、OS サービスは16ビットのソフトウェア割り込みによって提供されます。ソフトウェア割り込みの例としては、ディスク・アクセスの INT13h、ビデオ・アクセスの INT10h、キーボード・アクセスの INT16h などがあります。

オペレーティング・システムをロードするには、これらのサービスの存在が不可欠です。しかし、BIOS サービスには大きな制約があります。それは、新しいサービスを拡張しにくい、レジスタ間でのパラメータの受け渡しに制約がある、リアル・モードの制約がある、といった点です。EFI では、IA-32 やインテル® Itanium® プロセッサ・ベースのプラットフォームなど異なるプラットフォーム・アーキテクチャ間で共通の OS ローダを利用することができます。現在のレガシー OS ローダは IA-32 ベースの PC でしか使用されません。

## 新しいテクノロジ

インテル Platform Innovation Framework for EFI (以下、フレームワーク) では、システムの初期化および OS へのサービス提供を、一連のフェーズによってサポートするというアーキテクチャを採用しています。それぞれのフェーズは、利用可能なリソース、フェーズ内でコードが従うべき規則、フェーズの結果などがすべて異なります。図1はこれらのフェーズを示したものです。これを見ると分かるように、各フェーズは一定の順番で実行されるようになっています。

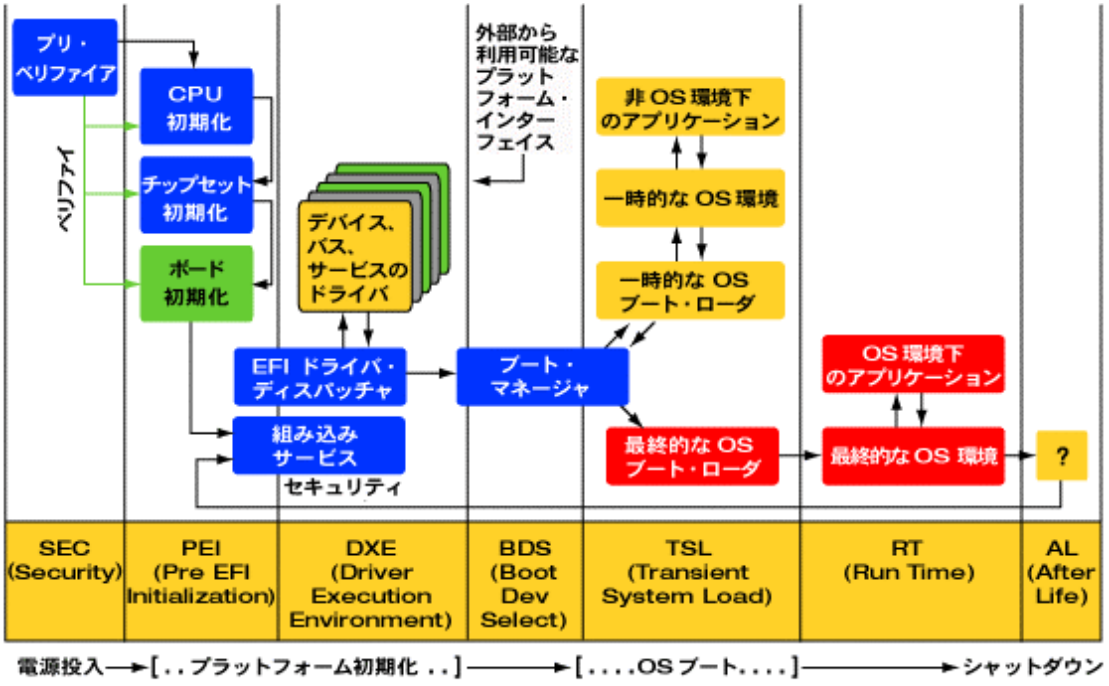


図1: ファームウェアのフロー

各フェーズで利用可能なインフラストラクチャは中心となるフレームワークによって提供され、プラットフォーム固有の機能は相互通信可能なモジュールを使って実装されます。PEI (Pre-EFI Initialization) フェーズのモジュールは PEI モジュール (PEIM) と呼びます。DXE (Driver eXecution Environment) のモジュールには DXE ドライバと EFI ドライバがあります。PEI と DXE の関係を示したのが図2です。ファウンデーションとモジュールのほとんどは、移植性の高い C コードで記述されます。

EFI ドライバは OS のデバイス・ドライバとよく似た役割を果たします。EFI ドライバによってフレームワーク・アーキテクチャには高い拡張性が備わるため、以下のような利点が生まれます。

- 幅広いプラットフォームの要求に対応できる。
- 新しいイニシアティブやフィックスを組み込んだり、新しいハードウェアへの対応が容易。
- モジュラー型ソフトウェア・アーキテクチャをサポート。

EFI ドライバはさまざまなメーカがさまざまな時期に開発することが考えられます。このため、従来のモノリシックな BIOS にはなかった問題も発生します。そこで、この新しいフレームワークでは、EFI ドライバの実行順序の管理、EFI ドライバ・インターフェイスの抽象化、共有リソースの管理に関する強力なソリューションを定義しています。また、このフレームワークと EFI ドライバは使用前にオプションとして暗号化による検証を行うこともでき、電源投入から OS のブート後に至るまで、全体的なトラスト・チェーンを保証できるようになっています。

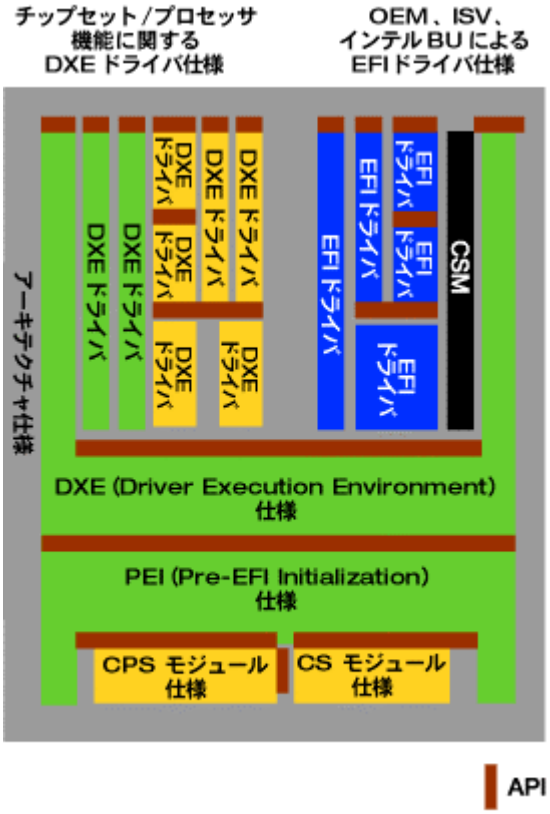


図2: フレームワークのバイナリ・モジュール

PEIM とドライバは、1つ1つが独立して構築されたバイナリ・モジュールとして導入することができます。これらのモジュールは、ファームウェア・ボリュームと呼ばれる1つの抽象化されたストレージにまとめられます。ファームウェア・ボリュームは、さまざまなテクノロジの中でも特にプラットフォームの不揮発ストレージを記述するために用いられます。

モジュールとシステム、およびモジュール間の通信は呼び出し可能なインターフェイスによって行われ、このインターフェイスには GUID (Globally Unique IDdentifier) によって名前がつけられています。GUID は静的に一意であることが保証された128ビットの値です。このように一意の値を与えることによって、標準サービスやプラットフォーム固有のサービスどうしの競合などの制約なしに、拡張可能なサービスを開発できるようになっています。

EFI とこのフレームワークはインターフェイス主体の設計を採用しており、特定のマイクロアーキテクチャやプラットフォーム・テクノロジからソフトウェアを抽象化して切り離しています。このため、このフレームワークはすでに IA-32 ベースのデスクトップ、サーバ、エンベデッド機器、モバイル・システムに幅広く移植されており、Itanium® プロセッサ・ベースのサーバや Intel XScale® テクノロジ・ベースのプラットフォームへの導入も進んでいます。なお、Intel XScale® テクノロジ・ベースのプラットフォームへの導入に関しては、フレームワーク・コンポーネントをクロスコンパイルし、その他のモジュールを開発または再利用すればよいため、スムーズな導入が可能となっています。

PEI および DXE フェーズは、プラットフォームの初期化を行います。これは、メモリの初期化、I/O バスのリソース管理、I/O デバイスの検出など、レガシー BIOS の POST フェーズで行われていたものです。バイナリ・モジュールと動的なサービス検出を採用しているため、コンポーネントの再利用やプラットフォーム機能の区別が容易に行え、モノリシックな BIOS のようにすべてのコードを再リンクする必要がありません。

POST に相当するプラットフォーム初期化機能に加え、このフレームワークは OS インターフェイスも提供します。DXE フェーズの初期段階で、いくつかの EFI インターフェイスが開始されます。このため、EFI に準拠したインターフェイス群をきわめて小さいスペースで実装することが可能となっています。また、このフレームワークではいくつかのドライバを利用することによってレガシー OS インターフェイスもサポートします。**図3**は、このデュアル OS インターフェイスのサポートを示しています。

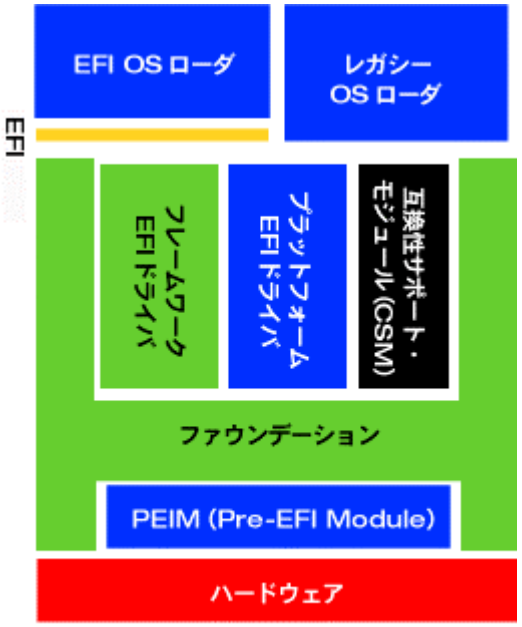


**図3：ソフトウェアのレイヤ**

## 参考システム例

このフレームワークを利用すれば、プラットフォームをサポートすることができます。**図4**は、現在の典型的な PC プラットフォームの参考システム例を示しています。主なコンポーネントとしては、シリコン・コンポーネント (青)、メモリ・テクノロジ (緑)、I/O バス (紫)、CPU テクノロジ (オレンジ) があります。このフレームワークを利用してプラットフォームを構築するには、メイン・メモリの初期化、基本的な CPU およびチップセットの状態など、基本的なプラットフォーム・ファブリックの初期化を行う PEIM が必要となります。

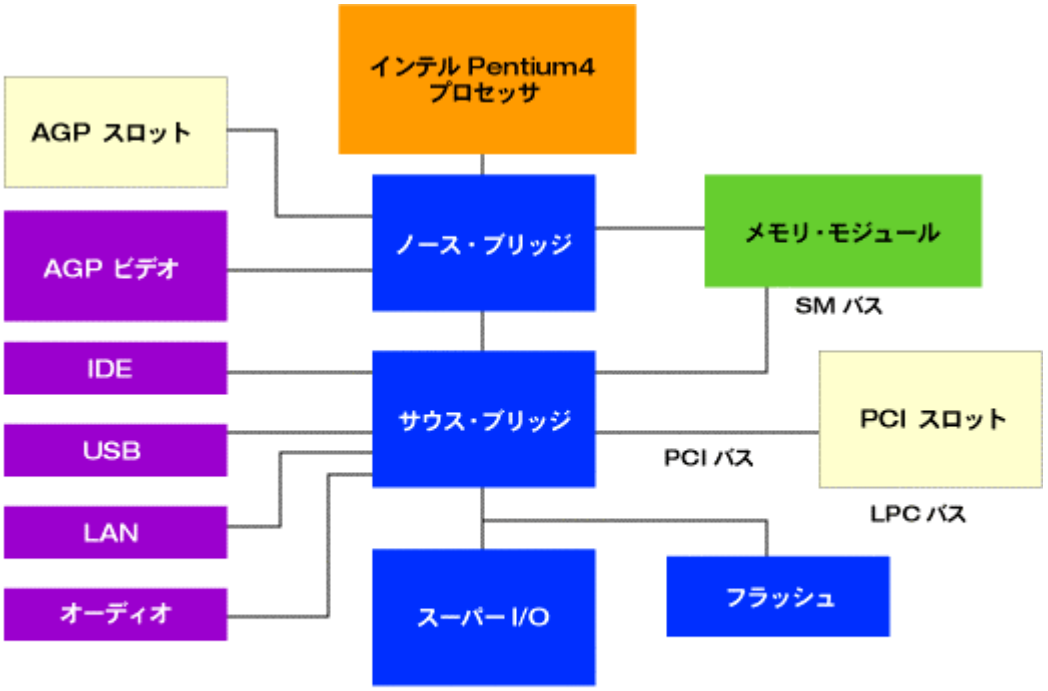


図4：システム・ブロック図

システム初期化の PEI フェーズは、PEI ファウンデーションがいくつもの PEIM を次々と呼び出しながら進行します。PEIM には、ノース・ブリッジ初期化のための PEIM、メモリ・テクノロジ初期化のための PEIM、プラットフォーム PEIM、CPU PEIM などのモジュールがあります。このようにモジュール単位で PEIM が分離されているため、例えば、設計やチップセットの異なるプラットフォームに対して同一の CPU PEIM を再利用できるといった利点が生まれます。

基本的なプラットフォームの状態が確立したら、DXE フェーズが開始します。プラットフォーム初期化の大半はこの DXE フェーズで行われます。ここでは、PCI バスのリソース割り当てのためのドライバ、フラッシュへの耐障害性を備えたアクセスを抽象化するためのドライバ、コンソール・アクセス (ビデオ、USB キーボード) のためのドライバなどが使用されます。また、レガシー OS と EFI OS のブートのサポートもこの DXE フェーズで行われます。

PEI の場合と同じく、DXE も一部のドライバを交換するだけで異なるプラットフォームをサポートすることができます。例えばフラッシュ・コンポーネントやビデオ・デバイスを変更した場合でも、これらのドライバを1つ変更するだけで対応できます。また、ブート・ポリシを記述したドライバを1つ変更するだけで、高可用性が求められるサーバにも、構成が固定されたエンベデッド・システムにも対応できるといったケースも考えられます。

## まとめ

インテル Platform Innovation Framework for the EFI (Extensible Firmware Interface) は EFI を製品レベルで実装したものです。このプラットフォームは従来の BIOS に代わるものとして、ブートの高速化をはじめ、マネージャビリティやその他の機能性の向上をもたらします。このフレームワークは C 言語で記述された強力なアーキテクチャ・インターフェイスで構成されているため、モジュラ型のコンポーネントを組み合わせることでプラットフォーム・ファームウェアを構築できるようになっています。このフレームワークを利用することで、BIOS 業界やインテルの顧客企業は差別化を図った画期的なプラットフォームを短期間で開発できるようになります。

## 関連情報

- インテル® Platform Innovation Framework for EFI に関するページ
- EFI (Extensible Firmware Interface) に関するページ

## 著者紹介

Vincent Zimmer、エンタープライズ・プラットフォーム事業本部所属のスタッフ・エンジニア。インテル勤務歴7年。エンベデッド・ソフトウェア開発に12年以上携わり、現在100件以上の米国特許を持つ (一部、出願中のものも含む)。コーネル大学にて電気工学の学士号、ワシントン大学シアトル校にて計算機科学の修士号を取得。

— Technology@Intel マガジンの記事 終わり —