解説

新世代 BIOS "UEFI" の構造と実装

菅原清文

インテル（株）
ソフトウェア&サービス統括部
ソフトウェア技術部

はじめに

BIOS（Basic Input/Output System）はマザーボード上に搭載されるさまざまな H/W コンポーネント（キーボード，フロッピーディスク，ハードディスク，ビデオなど）を制御するプログラムであり，マザーボード上のフラッシュメモリに搭載されている．PC の電源が投入されると BIOS は自動的に起動され H/W の初期化を行い，インストールされている OS を呼び出す．PC の利用者は BIOS 設定画面で操作することはできるが，その役割はあまり知られることはない．しかしながら，現在の PC は BIOS なくして起動することすら不可能なのである．

PC BIOS はおよそ 27 年前に設計された IBM PC のファームウェアが基になっており，ROM-BASIC や DOS を起動する前のハードウェア設定と，DOS 起動後プログラムからの H/W へのアクセスをサポートすることを主な役割としてきた．Plug & Play が一般的になった今では Add-in カードの入れ替えや追加にさして問題は起こらないが，かつての PC-AT バスのカードでは Add-in カードごとに割込みや，I/O ポート，メモリアドレスの割り当てを手動で行う必要があり，状況に応じ BIOS 設定も変更する必要があった．また，プログラミングの面から見ても DOS はすべてのハードウェアへのアクセスを提供しておらず，ソフトウェアは BIOS サービスと呼ばれる割込みベースの呼び出しによって，直接ハードウェアにアクセスしなければならなかった．

[BIOS の限界]

PC 向けに設計された BIOS であるが，IA アーキテクチャを採用するワークステーションやサーバでもそれに変わるファームウェアは採用されず，21 世紀を迎えても旧体質の BIOS が使われ続けたが，さすがにいろいろな面で制限と限界が見えてきた．能力的および機能的な面で BIOS の限界をあげると下記になる．

①アーキテクチャ上の制限がある
16 ビットアセンブリで記述された BIOS では利用可能なコードとメモリ空間，そして CMOS エリアに限りがある．

②信頼性に乏しい
OS が停止した場合に BIOS モニタに復帰するなど，信頼性を高める機能がない．

③新たな技術を容易に実装できない
パーティショニングやセキュリティなど最新の技術革新への余地がない，また新しいタイプのブートデバイスが登場するたびに OS ブートローダを変更しなければならない．

④明確な仕様が定められていない
OS のブートローダと BIOS の間に明確な仕様が定められていない．そのため複数の OS のインストールとブートが煩雑である．

BIOS からこれらの制限を取り除き，OS ブート前の環境として新たなファームウェアが求められてきた．

[業界は何を考えたか]

1999 年，米インテル社はファームウェアと OS のブートの仕様を明確にするため，EFI（Extensible Firmware Interface）仕様を発表した．当初の EFI は 64 ビットマシンである Itanium プラットフォームのファームウェアとして利用された．当時 OpenFirmware のようなブートの標準化仕様があったが，過去の IA 資産を活かすには不都合であった．インテルは IA-64 だけでなく IA-32 の標準ファームウェアを見据えて EFI の仕様改良と実装を進め，2003 年 11 月に最終仕様であるバージョン 1.10 を公開した．Pheonix，Award，AMI，Insyde などの BIOS

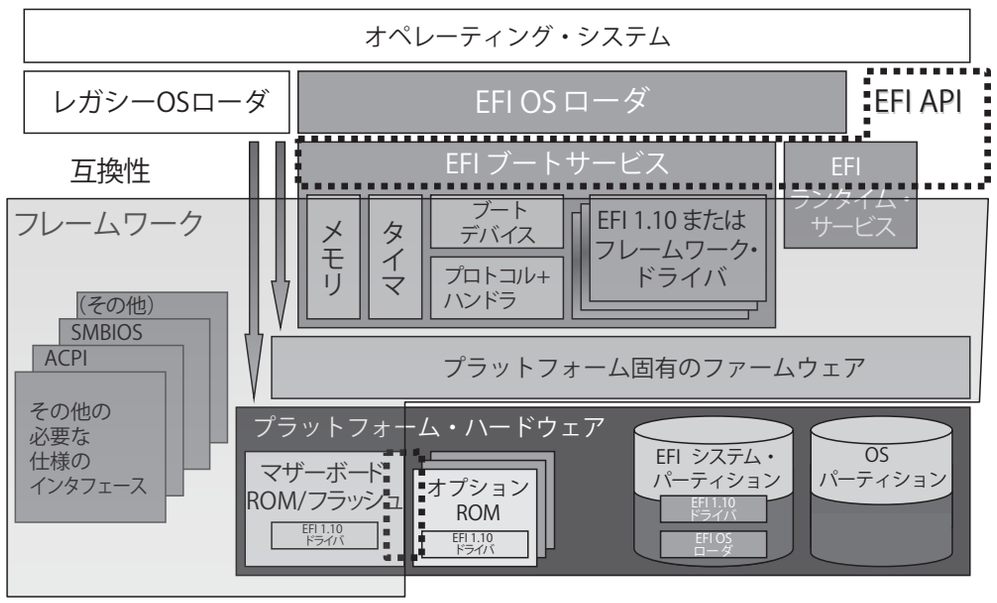


図 -1
EFI のシステムトポロジ

ベンダもハードウェアの進化に対し BIOS を独自に実装する限界を感じ，新たなファームウェアの開発に積極的であった．そして OS ベンダが EFI 対応のブートローダ開発を表明したことで，今日に続く新たなファームウェアの革新が始まった．

［UEFI（Unified Extensible Firmware Interface）とは］

2005 年 7 月，非営利法人 UEFI フォーラムが設立された．インテルが提唱した EFI を拡張し，レガシー BIOS との互換性を高め，ユーザインタフェースの充実を目指し，2005 年以降は UEFI フォーラムによって，EFI 仕様の管理が行われている．2006 年 1 月 31 日に UEFI 2.0 仕様が公開され，2007 年 7 月には最新の仕様となる UEFI 2.1 仕様が公開された（2008 年 6 月に 2.1 仕様のバグフィックス版がリリースされている）．現在 UEFI のボードメンバには，AMD，American Megatrends Inc（AMI），Apple，HP，IBM，Insyde，Intel，Lenovo，Microsoft，そして Phoenix Technologies が参加している．

EFI 概要

本章では UEFI の基礎となる EFI の概要について説明する．

［EFI を構成する基本要素］

EFI インタフェースの目的は，EFI ドライバ，EFI アプリケーション，および EFI OS ローダを含む EFI イメージによって使用される共通のブート環境を定義することであり，呼び出し規則は，拡張性を考慮して完全な 64 ビットインタフェースが定義されている．プラットフォームと OS を EFI インタフェースによって分離することで，プラットフォームと OS がお互いに独立して進化することが可能となる．

EFI プラットフォームインタフェースは，すでにあるブートサービスで使用される Plug & Play オプション ROM などの使用を可能にする．PC 業界ではインテルアーキテクチャのオプション ROM に多大な投資を行っており，すでに採用されているそれらの技術を排除することは，最初の世代の EFI 準拠システムでは現実的ではないと判断した．オプション ROM とは，Add-in カードに搭載される BIOS とは別のファームウェアであり，PC の起動時に BIOS とやりとりをして，Add-in カードを利用可能にするプログラムである．Plug & Play オプション ROM では，複数の Add-in カードが利用するリソース（割込みや，I/O ポート，メモリ）が BIOS によって競合することなく自動設定される．

EFI プラットフォームインタフェースはプラットフォーム上で行うブートプロセスにおける，OS とプラットフォーム間のインタフェースを明確にする．また，EFI 仕様では診断およびユーティリティプログラムとプラットフォーム間の API を定義している．

図 -1 に示す EFI のシステムトポロジで，EFI 仕様が定義するインタフェースは破線で囲まれた部分である．インタフェースは定義されているが，実際の H/W に対する実装方法は明確にされていない．ベンダは仕様に従って，独自に既存の BIOS コードの機能を実装する必要があっ

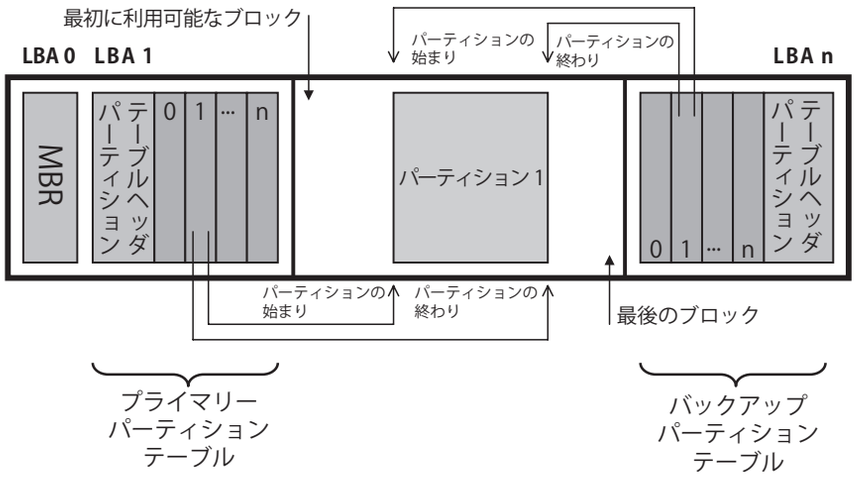


図 -2
Global Partition Table（GPT）

た．そのため後述する EFI フレームワークでは，フレークワークと記される部分の実装方法が追加定義された．

ブートサービス

ブートサービスは OS のブート時に使用されるデバイスとプラットフォームに対するインタフェースを提供する．デバイスは「ハンドル」と「プロトコル」を通してアクセスされ，デバイスへアクセスする OS ローダへ負担を負わせることなく，既存の BIOS コードに投資されている資源を再利用することを可能にする．デフォルトでは最小限のブートデバイス（HDD）と VGA がサポートされるが，OS ブート前にそれ以外のデバイスの EFI ドライバをロードすることで利用可能となる．

EFI のブートフェーズにはブートマネージャが用意されており，複数 OS のブートを容易に管理できる．また，OS を起動することなく EFI コマンドラインシェルを利用することができる．

レガシーリソースの再利用

OS やファームウェアにおける既存の環境をサポートするコードを活かすため，インテルアーキテクチャ上で一般的に実装されている既存の規格（ACPI や WfM など）は，EFI 仕様を満たすプラットフォームに実装される必要があり，そのためのインタフェースを定義する．

システムパーティション

システムパーティションは，複数のベンダおよび異なる目的で安全に共有されるパーティションとファイルシステムを定義する．共有可能なシステムパーティションを分離することで，NVRAM に対する負荷なしでプラットフォームに新たな機能を加えることを可能とする．レガシー BIOS では Master Boot Record（MBR）というパーティションフォーマットを使用するが，EFI は**図 -2** に示す Global Partition Table（GPT）と呼ばれるパーティションフォーマットを利用し，EFI はその中に EFI システムパーティションを持つ（100MB 程度の FAT32 でフォーマットされた領域）．EFI システムパーティションには，OS のブートローダや OS ベンダが提供するツール，または EFI ドライバ，プリブート環境で利用するツール（H/W の診断ツールなど）が格納される．

GPT は 2 つのパーティションテーブルを持ち冗長性を高めている．パーティションテーブルヘッダは 128 エントリーのパーティション情報を保持することができる．つまり 128 個のブート可能なパーティションが作成できる．各パーティションは Global Unique IDentity（GUID）と呼ばれる 128 ビットの識別情報を持ち，OS ごとに個別の識別子が割り当てられる．レガシーのディスク管理ツールとの互換性を保持するため，LBA0 にはこれまでどおり MBR を持つ．この情報がないと HDD は空ディスクであると判断されてしまう．

ランタイムサービス

OS がブート中と起動後に必要とするハードウェアリソースの参照をサポートするため，最小限のランタイムサービスを提供する．提供されるサービスは GetTime（リアルタイムクロック値を得る），SetTime（リアルタイムクロック値を設定），GetWakeupTime（ウエイクアップアラーム時間を得る），SetWakeupTime（ウエイクアップアラーム時間を設定 ），ResetSystem（プロセッサとデバイスをリセットする）などである．

これらの EFI ランタイムサービスで使用されるすべてのメモリ領域は，OS が予約領域もしくは未使用領域として定義しなければならない．EFI ランタイムサービスが使用するメモリ領域は EFI 関数にのみ提供され，OS やそのコンポーネントが直接操作できない．EFI はランタイムサービスで使用されるハードウェアリソースを定

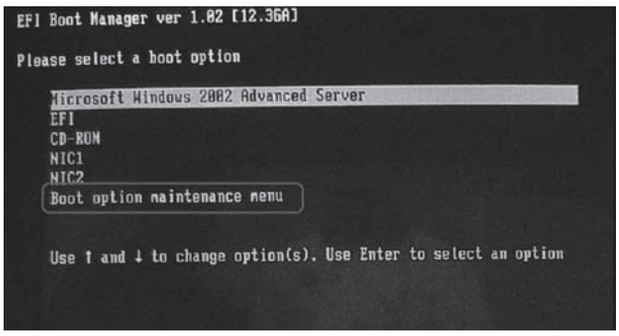


図-3　EFI コンソールのブートマネージャ

義することに責任があり，OS はランタイムサービスを呼び出すことでそれらのリソースと同期することができる．もしくはそれらのリソースが OS から参照されないことを保証する．

［ブートマネージャ］

EFI は EFI アプリケーション（OS のファーストステージのブートローダを含む）や EFI ドライバを，EFI システムパーティションから EFI で定義されたイメージロードサービスを使用して読み込むことを管理するブートマネージャを備えている．**図-3** に EFI コンソールのブートマネージャの画面例を示す．EFI は読み込むファイルの位置を知るため NVRAM 変数を使用する．これらの変数には，EFI アプリケーションに直接引き渡される詳細データが含まれる．変数は，メニューで表示される可読な Unicode 文字列を含むことができる．

EFI で定義される変数は，システムファームウェアにすべてのオペレーティングシステムの格納場所をブートメニューに含むことを許し，さらに同じオペレーティングシステムの複数のバージョンを含むことを可能にする．EFI 設計の目的は，プラットフォームファームウェアで 1 つのブートメニューを持つことであり，EFI は選択されたオペレーティングシステムが使用する変数のみを NVRAM から読み出す．EFI は付加価値を加える用途として，システムメニューの変更を許可する．これにより，利用者は自由にブートメニューをカスタマイズできる．

EFI は PC BIOS にブートの柔軟性を持たせた構造を持つ．今日の PC BIOS システムでは，ブートはプライマリーのフロッピー，ハードディスク，CD-ROM そしてネットワークに制限されており，共通のハードディスクからブートすることは，異なるベンダの OS 間や，同じベンダのバージョンの異なる OS で，相互互換の問題が発生する可能性を秘めている．

一例として，Windows と Linux をデュアルブートする場合，現在の BIOS と MBR の構造では Linux インストール後に Windows をインストールすると，Windows のブートマネージャが Linux の GRUB を上書きして Linux がブートできなくなったり，正しく双方の OS をブートするには設定に気を使わなければいけない場面があり得る．また，MBR では OS がブート可能なプライマリーパーティションは 4 つまでしか確保できないが，EFI における GPT ではこの制限はない．

［EFI コンソール］

EFI コンソールは，シンプルな入出力プロトコルで構成されており，これらの 2 つのプロトコルは，プラットフォームファームウェア，EFI アプリケーションおよび EFI OS ローダが情報を表示することを可能にし，システム管理者から入力を受け取るための簡単なテキストベースのコンソールを提供する．

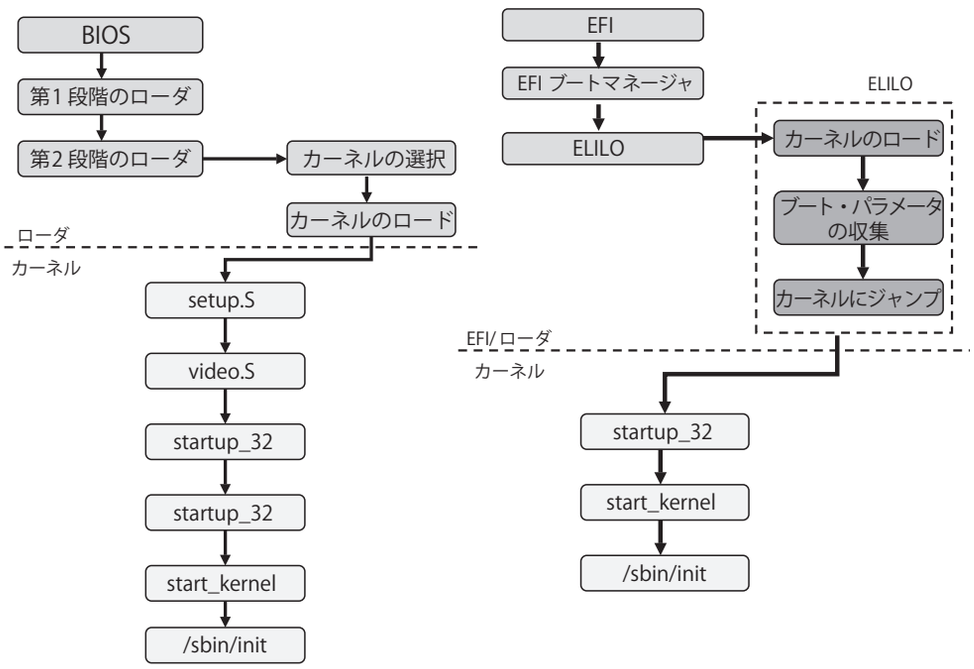
EFI コンソールは，16 ビットの Unicode 文字，簡単な入力制御文字，そしてインテリジェントなターミナルに等しい機能を提供するためのプログラマティックなインタフェースから構成されるが，最初の仕様ではマウスなどのポインティングデバイスやビットマップ出力をサポートしていなかった．

EFI 仕様では，シンプルな入力プロトコルが対応する同じ言語をサポートする出力プロトコルが必要であり，テキスト出力プロトコルは，少なくとも標準ターミナルエミュレーションソフトウェアが EFI コンソールを使用できるように Unicode の ISO Latin1 文字セットをサポートすることが推奨されている．ISO Latin1 文字セットは，16 ビット文字へ拡張された ASCII のスーパーセットを実装しており，その他の Unicode ページはオプションでのサポートとなる．

EFI コンソールからは EFI SHELL 環境へ移行することができ，SHELL 環境では基本的な EFI 内部コマンド群と Python ライクなスクリプト機能が用意される．サポートされるコマンドは以下のものがある．

help，cp，exit，guid，comp，dblk，set，mkdir，pci，alias，rm，attrib，dh，memmap，load，type，unload，dmpstore，map，ver，mount，time，cd，date，echo，reset，pause，vol，ls，cls，mode，edit

内部コマンドは，Linux のシェルや Windows のバッチファイルで利用可能なコマンドに似た名称が多く存在する．BIOS 環境での一番の問題は少なくても DOS をブートしなければ，メディアのコピーどころか H/W の診断もできないことである．EFI のコンソール環境では，上記のコマンドを利用してメディアの操作を行ったり，外部ベンダが提供する EFI 外部コマンドを利用することが可能になる．



レガシーブート　　EFI ブート

図 -4
Linux 環境でのブートプロセスの違い

レガシー BIOS の環境ではプログラム開発者が OS ブート前にそのプロセスに介在することは容易ではなかったが，EFI 環境では EFI 外部コマンド（EFI イメージと呼ばれる）プログラムを実行することができる．プログラムは「.efi」の拡張子を持つ PE32+ フォーマットの実行形式バイナリであり，マイクロソフトが定義する 64 ビット対応バイナリである Windows アプリケーションと同じ形式を持つ．言い換えれば，Visual Studio 等の Windows 開発環境がそのまま使用できるということである．Linux 環境でビルドする場合 objcopy ユーティリティを利用して，ELF から PE32+ へフォーマット変換する必要がある．

現在 EFI イメージや EFI ドライバの開発に必要なツールキットは，
https://www.tianocore.org/
で入手可能である．

ベンダの役割

EFI のインタフェースにより，各ベンダの役割が明確になり，それぞれの開発が分担されるようになった．

[OS ベンダの役割]

OS ベンダが OS を EFI 対応にするには，まずブートローダを変更しなければならない．レガシー BIOS 環境でのブートプロセスは，**図 -4** の左のようなプロセスになる．HDD の LBR0 に格納される MBR の Initial Program Loader（IPL：図 -4 では第 1 段階のローダ）が OS ローダを呼び出し，OS ローダはカーネルを読み込み，カーネルに制御を渡す．ここでローダは役割を終え，カーネルが利用するデバイス（ビデオなど）の初期化を行う．

これに対し EFI のブート環境（図 -4 の右のプロセス）では，EFI ブートマネージャが OS ローダを呼び出し，OS ローダがカーネルを読み込んで制御を渡す．一見同じプロセスに見えるが，EFI 環境では VGA などブートに最小限必要なデバイスのセットアップは OS ローダ起動前に終了しているため，カーネルの初期化プロセスを簡略化できる．ELILO は EFI Linux Loader の略であり，HP 社の Eranian 氏と Mosberger 氏によって開発され，EFI 未対応のカーネルブートも可能である．

EFI 環境での OS ローダは EFI システムパーティションに格納されるため，初期化されていない HDD に初めて OS をインストールする場合，OS インストールの過程で EFI システムパーティションを作成しなければならない．EFI に対応する Linux や Windows では，インストール時にディスクパーティションを作成する過程で，EFI システムパーティション（約 100MB ）の作成とフォーマットを行うようになっている．OS ローダは EFI システムパーティションの EFI ディレクトリ下に配置される．OS ベンダによっては EFI ユーティリティ（ディスクフォーマットやパーティション作成コマンド）が提供される．

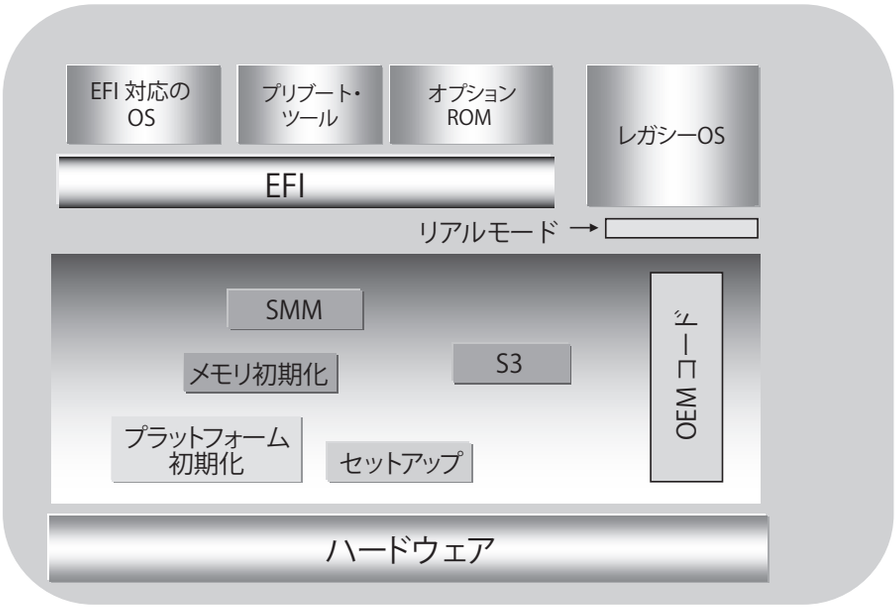


図 -5
EFI における実装の問題点

[ハードウェアベンダの役割]

Add-in カードを開発するハードウェアベンダは EFI 対応に際し 2 つのことを行わなければならない．1 つはそのハードウェアがブートセクションで利用される可能性がある場合，EFI ドライバを提供すること，もう 1 つは Add-in カード上に搭載されるオプション ROM の構造を変えることである．レガシーのオプション ROM は 16 ビットもしくは 32 ビットの IA-32 コードであるが，EFI 仕様では EFI バイトコード（EBC）と呼ばれる新たな方式を推奨している．EBC を搭載する Add-in カードは，EFI ベースの IA-32，Intel64，IA-64 そして他のアーキテクチャベースのシステムでもオプション ROM 機能を利用できる．当面はレガシーコードと EBC を同時搭載することですべてのシステムに対応する．

EFI の最初の実装での問題点

OS と BIOS を分離してインタフェースを定義した EFI の最初の仕様では，解決されていないいくつかの技術的問題が残っていた．EFI ではファームウェアと OS ブート環境のインタフェース（図 -1 の破線の部分）を定めただけで，**図 -5** の中段にあたるファームウェアレイヤ（もとの BIOS にあたる部分）の実装仕様を明確に定めていなかった．プラットフォームがパワーオンされ OS をブートするまでにはさまざまなプロセスが存在する．EFI はファームウェア内部をモジュラー構造にしているが，H/W の初期化と BIOS セットアップ，OEM や BIOS ベンダ固有のサービスや機能のセットアップなどそれぞれを明確な仕様によって区分しているわけではない．そのため BIOS ベンダ，OEM メーカは各種 BIOS のコンポーネントを個別に EFI 化しなければならなかった．

UEFI で拡張された仕様

レガシー BIOS に EFI を構造的に上乗せするだけでは不十分であり，BIOS が持つ同じ課題や問題を継承してしまう．この問題を解決するため，2003 年，インテル社は，各ベンダ間の垣根を越えたモジュラー構造を持ち，クリーンでスケーラブルなファームウェアのアーキテクチャとして，EFI フレームワーク（正式には，Intel Platform Innovation Framework for EFI）を発表した．この EFI フレームワークが UEFI の実装の基となっている．

図 -6 に EFI フレームワークの構造を示す．EFI フレームワークは Foundation と呼ばれる基本コンポーネントを骨格とし，フレームワークドライバ，プラットフォームドライバ，EFI ドライバ，アーキテクチャプロトコルそして互換性サポートモジュール（CSM）を実装する．フレームワークが定義されたことにより，各ベンダは仕様に従ってさまざまなコンポーネントを標準化できるようになった．

[Foundation と制御の移行]

Foundation は Pre-EFI（PEI）Foundation と DXE（Driver Execution Environment）Foundation の 2 つの階層で構成される．

Pre-EFI Foundation では，CPU，チップセットそして

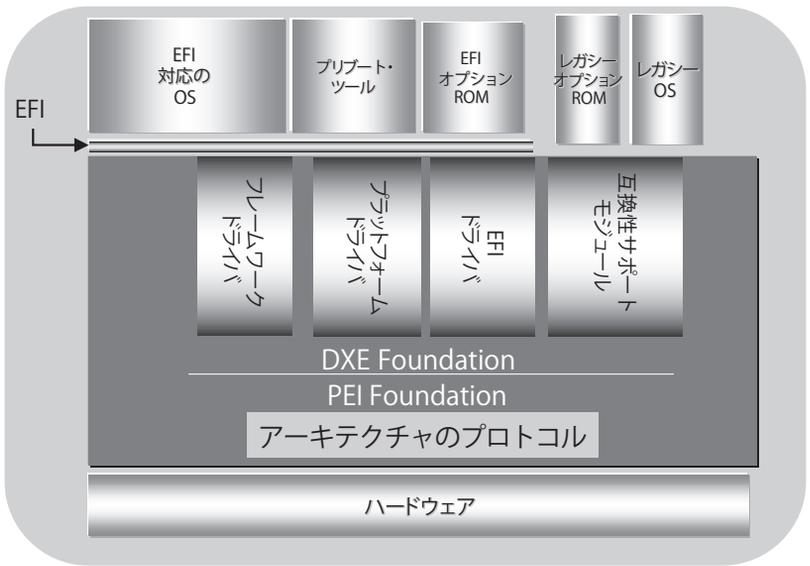


図 -6
EFI フレームワーク

ボードの初期化が行われるが，主な目的はブートモードの検索，メインメモリ初期化，DXE コアの検索と起動である．PEI フェーズではメモリが初期化中であるため利用できない，そのためメモリに代わってプロセッサキャッシュを RAM として使用する．モジュラー化された PEI コードは，CPU やチップセットの初期化を簡略化する．PEI は BIOS の POST に相当し，DXE Foundation が参照するマザーボード上の H/W デバイスを利用可能な状態に初期化するフェーズであり，H/W に依存するコードはここに実装される．PEI で初期化された H/W は DXE 以降のフェーズで利用可能となる．

DXE Foundation では，デバイス，バス，そしてさまざまなサービスで利用されるドライバの初期化を行い，利用可能な状態にする．DXE フェーズ中に実行される機能は DXE ドライバ（**図 -7** のフレームワークドライバやプラットフォームドライバ）として実装する．DXE Foundation 自身にはハードウェア固有の機能はない．そのためハードウェアへのアクセスはアーキテクチャプロトコル（AP）によって抽象化された各 DXE ドライバによって行われる．AP（= ドライバ）は，CPU，チップセット，ボードに固有の機能をカプセル化する．システム起動に必要なドライバがすべてロードされると，Boot Device Selection（BDS）として実装されるブートマネージャを起動する．DXE では必要に応じてキーボードやビデオなど起動に必要な EFI ドライバを接続する．

ドライバの仕様はチップセットとプロセッサ機能に関するプラットフォームドライバ（DXE ドライバとも呼ばれる）とサービスを提供する EFI ドライバの 2 つに分けられる．フレームワークドライバは Foundation の制御に関するドライバである．

BDS は DXE から制御を受け取り OS ブートマネージャや EFI コンソールに制御を渡すまでのプロセスを管理する．デバイス管理とユーザインタフェースはこのフェーズに集中され，BIOS ベンダや OEM によるカスタマイズもここで実装される．

レガシー BIOS で提供される BIOS セットアップ画面も EFI イメージとして BDS から呼び出される．また，追加したドライバや OS ブートの情報は NVRAM に格納される．OS ブートの失敗を監視するウォッチドッグを搭載し，ブートが失敗すると BDS フェーズに戻ることができる．

［下位互換サポートモジュール（CSM）］

下位互換サポートモジュール CSM（Compatibility Support Module）はレガシー BIOS の機能を提供する．CSM は BIOS ベンダによって供給され EFI ファームウェアのモジュールの 1 つとして実装される．CSM の実装にはレガシーの 16 ビットコードをベースにする方法と，レガシー BIOS をフレームワークモジュールで包み込む 2 つの方法がある．CSM は単独もしくは EFI インタフェースとともに動作し，BIOS と完全互換を要求する OS のブートや，レガシーオプション ROM のみを提供する Add-in カードを使用して EFI 対応 OS をブートすることを可能にする．

OS ブート以外にも，レガシー BIOS が提供するランタイムインタフェースも CSM によって実装される．たと

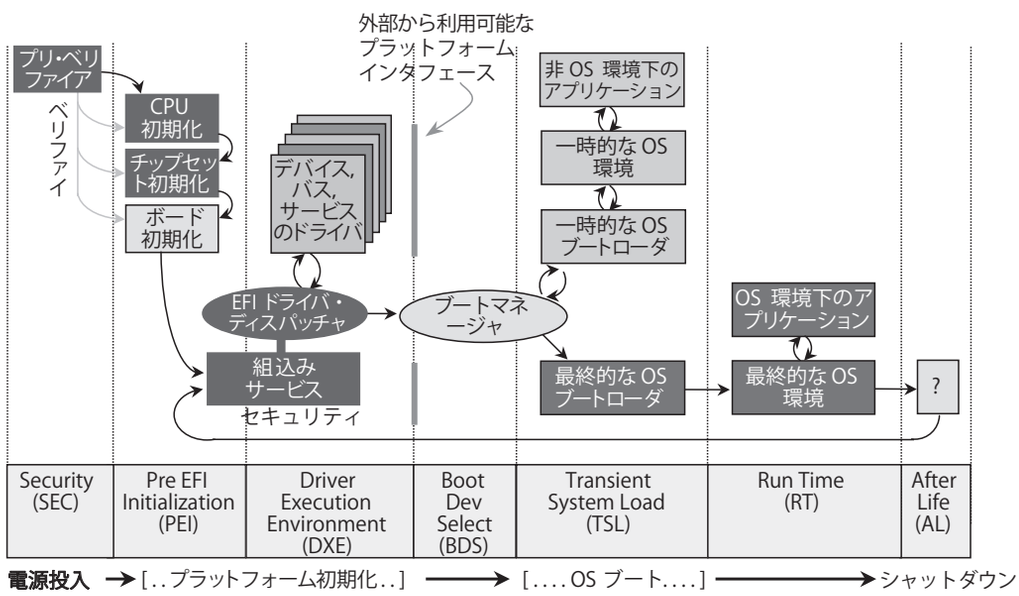


図-7
EFI フレームワーク環境でのブートまでの流れ

えばレガシー BIOS 環境では割込み（int x）によって BIOS サービスを駆動できるが，EFI フレームワークでは同様のレガシーサービスを提供しない．

[その他の機能]

ヒューマンインタフェースは EFI フレームワーク以降大幅に改善された領域である．初期の EFI コンソールはテキストモードのみを提供し，特殊な I/O デバイス固有の機能は利用できなかった．特に，日本語キーボードを搭載したシステムで EFI コンソールを利用するのは不便であったし，フォントの拡張実装が行われていなかったため，日本語を利用することもできなかった．

現在の UEFI 仕様をベースにしているファームウェアは，言語が異なる場合の文字コード，フォント，入力デバイスに関する問題を解決し，ファームウェアのローカライズを前提に設計されている．文字コードは Unicode を利用し，簡易であるがビットマップフォントが利用できる．これらの機能もまた BIOS ベンダによって提供される．

EFI フレームワークでは，Internal Forms Representation（IFR）を実装する．IFR はブラウザ上でページレイアウトや表示オペコードおよび文字列トークンを記述するための言語である Visual Forms Representation（VFR）によって作成する．これにより OS ブート前の表示制御のパフォーマンスを向上させている．また，XHTML や Java スクリプトなどへの変換も容易となる．多くの BIOS ベンダはセットアップを Web モデルにマッピングしブラウザを実装することで容易なルックアンドフィールの設計を実現している．

TCP/IP ドライバを組み込み，ネットワークアダプタを利用可能にすることで，OS ブート前にインターネットのブラウズができたり，サポートセンタへの接続を可能にする．OEM や BIOS ベンダから見ると，OS ブート前に問題を特定することは，サポートリソースの観点から見て非常に有益である．

[フレームワークのカスタマイズ]

フレームワークのカスタマイズは BIOS ベンダによって提供され，マザーボードベンダや OEM によって機能や見た目上のカスタマイズが行われる．特にルックアンドフィールは BIOS ベンダごとの工夫が凝らされている．

EFI 採用の主導権を握るのは PC を製造する OEM である．現在ほとんどのプラットフォームでは，レガシー BIOS と UEFI の選択が可能であるが，FLASH ROM の容量の問題で両者をダイナミックに選択可能なものはない．また，EFI を搭載する製品でもレガシー BIOS と同じ操作とルックアンドフィールを実装しているため，使用するシステムがどちらか判別することは容易ではない．

近年 EFI フレームワークを最も効果的に実装したのは Apple 社である．以前のオープンファームウェアベース

の見た目や操作感をまったく変えず，EFI フレームワークを実装している．EFI フレームワークを搭載した Mac は以前よりブートスピードが 2 倍はやくなり，OS 起動前に WiFi デバイスからのブートや Bluetooth デバイスの利用が可能になった．Apple 社はファームウェアの移植を 1 カ月で行ったという．唯一の不満は開発ベースが Windows であるということである．

EFI フレームワークの移植とカスタマイズは PC ベースのハードウェアだけではなく，組込み機器の分野にも革新をもたらしつつある．組込み機器も電源パワーオンからサービスを開始するまでの間に，ハードウェアの初期化を行い，サービスコードの起動をしなければいけない．これまでは独自のモニタを開発していたメーカが EFI フレームワークを実装し始めている．見えないところでも着実にこの愛すべきファームウェアは普及しつつある．

まとめ

エンドユーザに EFI やフレームワークの恩恵や利点を説明しても理解されないケースが多い．ユーザの観点から見れば，BIOS から EFI フレームワークに変わっても使い方は同じであってほしいものであり，ルックアンドフィールは変更せずに EFI フレームワークによる利便性が実装されることが望まれる．

この新たなファームウェアの恩恵を得るのは OEM，BIOS ベンダそしてハードウェアベンダなどの開発メーカである．特に熟練エンジニアの確保を要求される BIOS ベンダにはさまざまな面で影響を与えている．また，まったく新しくビジネスチャンスを期待できるのはソフトウェアベンダである．これまで，OS ブート前の環境にソフトウェアベンダが入り込む余地はなかったが，EFI ベースのアプリケーションを開発するという新たなカテゴリが現れたわけである．多くの EFI アプリケーションが登場することを期待したい．

参考文献

1）Extensible Firmware Interface Specification 1.10（Dec. 1, 2002）.
2）Unified Extensible Firmware Interface Specification 2.1（June 5, 2008）.
3）UEFI 2.1 and UEFI Platform Initialization（PI）1.0 Details and Differences（IDF 2007）.
4）http://www.uefi.org
5）https://www.tianocore.org/
6）http://www.intel.com/technology/efi/

（平成 20 年 10 月 24 日受付）

菅原清文
kiyofumi.sugawara@intel.com

インテル（株）ソフトウェア開発製品部門のシニアエンジニア．これまで i860 C コンパイラをはじめとして各種ツールの開発，Plug & Play の日本語環境仕様開発，インテルソフトウェア開発製品の開発者支援業務などに従事．