OSを動かすまでの技術を総覧する

ブートローダの役割と開発の流れ

日高 亜友

「boot」という語は，コンピュータを起動すること，あるいは利用可能になるまでの一連の手順を意味するが，もともとは「bootstrap」からきている．また再起動の「reboot」という語は「boot」から派生した語である．最初に，ブートに関連する用語について整理する．（筆者）

ミュンヒハウゼン男爵は，18世紀のドイツの軍人で，若いころにロシアのフリードリッヒ大王の騎兵隊長を務めた後，ヨーロッパ各地を渡り歩き，狩猟家，冒険家としても知られています．しかし彼を有名にしたのは，軍役を終えて故郷に帰ってから彼が周囲に聞かせた物語です．

この数々の話は「ほらふき男爵の冒険」として後に出版され，また映画化もされました．また彼の名を取ったミュンヒハウゼン症候群は，健康であるにもかかわらずに自分があたかも病気であるかのように訴えたり，病院めぐりをしたりする精神病の症状として知られています．

この物語では，男爵は砲弾に乗るという驚異的な経験をし，それで月にも旅行し，あるときには自分で自分の髪の毛を引っ張って沼から這い出たとあります．そして後年出たこの話の別版では，湖から自分を引っぱり出すために，自分のブートストラップ（ブーツを履く時に引っ張り上げる「つまみ革」）を使用したとあります．「bootstrap」は，「自力で成し遂げる」という英語の意味がありますが，このほらふき男爵の冒険の話から出たことばであるという説が有力なようです．

ブートローダを理解するための各種用語

● IPL，ブートローダ，MBR

コンピュータに電源を投入したときに，最初に起動して，ほかのプログラムやオペレーティング・システム（以下OS）を動かすプログラムをIPL（Initial Program Loader）と呼びました．文献によっては使い分けている例もありますが，多くの場合はこのIPLを「bootstrap loader」とも呼んでいます．ブートローダ（bootloader）という用語はこれの短縮形で，一般にはIPLと同意に用いられています．

また，IPLやブートローダという語は，PC（IBM PC互換機）の環境においては，ディスク・ドライブの最外周にあるMBR（Maser Boot Record）部分に書かれている，BIOSが起動して最初に実行するディスク上のプログラム（別名マスタ・ブート・コード）のことを指す場合もありますが，もともとは前述のより広い意味で使用されるものでした．

PCでは電源投入時に最初に，ROMに搭載されたBIOSが実行されます．PCでさまざまな種類のOSを起動したり，あるい

Column 1 BIOSとは

古い時代のシステムでは，デバイス・ドライバの一部とブートローダに相当するものをBIOS（Basic Input Output System）としてROMに組み込み，システムの抽象度を上げることにより，アプリケーションや各種OSをより汎用的な環境で動作させてきました．

IBM PCとその互換機にもこのBIOSが採用され，初代IBM機とは異なるハードウェア構成でも，BIOSが機器構成の違いを吸収することにより，同じMS-DOSやWindowsを起動することができるという互換性を維持してきています．さらにBIOSは，ACPIによるパワー・マネジメント機構やPnPなどのOSに依存しないシステム固有の制御コードを組み込むためにも不可欠です．

また，近年では，USBキーボードをBIOS内でPS/2キーボードと同等に処理するというように，旧来（レガシ）のインターフェースに新技術をBIOSでマップすることで，進化し続けてきています．

GNUプロジェクトの推進者であるFSF（Free Software Foundation）のリチャード・ストールマン氏は今年3月に，フリーBIOSの開発を呼びかけました．現在でもサーバ機向けのLinuxBIOS（http://www.linuxbios.org/）のようなフリーBIOSはいくつか存在します（**表1**，p.55）が，彼はデスクトップやノートPC向けのフリーBIOS開発の必要性を主張して，準備しています．この理由は，BIOS開発メーカやPCメーカによってBIOSに実装されているパワー・マネジメントやシステム管理などの最新技術のコードが公開されないため，Linuxをはじめとするオープン・ソース系OSの開発に支障があるという問題のためです．

さらに，現状のBIOSには，以下のような問題があります．

- 周辺機器が多い場合はとくに初期化コードの実行に時間がかかる
- リアル・モードで動作するため実行するプログラムに制限がある
- システム起動時やBIOS更新時にMS-DOS互換OSまたは機能を必要とする
- BIOSが管理するパラメータのインターフェースが公開されていない
- 開発が一部のBIOSメーカとPCメーカに限られていて導入コストが高くつく

AMD社は，LinuxBIOSのようなフリーBIOS開発を積極的にサポートしているため，ストールマン氏もAMD社製マシンの使用を推奨しています．

これに対してIntel社では，IA-64（Itanium）で採用しているEFI（Extensible Firmware Interface）というBIOSに代わる新しい規格の普及をめざしていて，AMD社やMicrosoft社なども賛同しています．EFIは，既存のBIOSとの完全な互換性はなく，MS-DOSや既存のWindowsは起動することができないため，次期のWindows VistaやインテルのアーキテクチャのMacintoshが対応するのではないかとうわさされています．

また，最近ではこの高価なメーカ製のBIOSを導入することなく，BIOSを使わないでフル機能のPCアーキテクチャ・システムをブートするという技術も開発されています．たとえばLinuxのKEXECは，BIOSを介すことなく迅速にシステムの再起動時を行うため，システム再構成やカーネル更新時に，エンタープライズ・サーバの停止時間を短縮することができます．そのほか，一部の組み込み系x86 CPU専用のブートローダも開発されています．

はインストールしてある複数のOSを選択起動することができるのは，BIOSから呼び出されて動作する，MBRに書いてあるこのIPLと，それに続いて起動される一連のプログラムの働きによります．このように，Windows 2000/XPの起動時に使用されるNTLDRや，Linuxの起動で利用されるLILOやGRUBは，ブートローダとも呼ばれるので注意が必要です．また，システムコマンダーなどの各種OSの切り替え起動を目的とするツールは，ブート・マネージャと呼ぶことがありますが，同様にMBRのIPL起動のしくみを利用して実装しています．

いずれの場合もブートローダはシステムの起動時に一度しか実行されないプログラムですが，OSをもつコンピュータ・システムには不可欠なものです．

● ブートROM，ファームウェア

PCのBIOSを含めた広い意味のブートローダは，不揮発性メモリ（ROM）に格納されていることから，このROMをブートROMと呼びます．組み込みシステムでは，ROMの中にブートローダだけでなく動作プログラム（アプリケーション）やOSまで格納する場合があります．この場合も，システム全体を制御するプログラムが入っているROMをブートROMと呼びます．

ファームウェアは，特定のハードウェアを制御するために機器や周辺コントローラに組み込んで使用するソフトウェアです．ファームウェアは一般的にはROMに格納されていますが，中には機器の起動時に通信手段を経由して，外部から転送して利用する場合もあります．

● デバッガ，モニタ，モニタ・デバッガ，ROMモニタ

組み込み用のCPUメーカやボード・メーカ，リアルタイム・オペレーティング・システム（以下RTOS）メーカでは一般に，ソフトウェアの開発や移植を行いやすくするために，実機システムのROMに組み込むための，ユーザ・プログラムのロードやデバッグをサポートするプログラム（ROMモニタ）を提供しています．デバッガはとくにブレークポイントやトレースなどのデバッグ機能までをもつものを指し，そこまでの機能がないものをモニタと呼びます．実際上は明確な使い分けはなく，総称してモニタ・デバッガと呼ぶ場合もあります．

これらは，多くの場合，ブートローダの機能ももつため，そのままか，または改造してブートローダとして使用する場合があります．

デバッガやモニタは，無料で配布されたり，各メーカのWebサイトなど，インターネット上で公開されている場合が多いのですが，製品に組み込んで使用する場合には，利用条件やライ

Column 2　以前はROM，最近はフラッシュ・メモリに搭載

ブートローダは，マスクROMやPROM，紫外線消去タイプのEPROM（Erasable and Programmable ROM，別名UV-EPROM）に代わり，最近では一般的に，ボード上に実装したまま書き換えができるフラッシュ・メモリに搭載されています．フラッシュ・メモリは，電気的にメモリの一部または全部を消去して再書き換えできるEEPROM（Electronically Erasable and Programmable ROM）の一種で，高速にアクセスし，大容量化しやすく，おもにNOR型とNAND型に分類できます．

NOR型はランダム・アクセス性能に優れ，バイト単位で書き込みが可能なため，ブートローダやファームウェアといったプログラムや，システム・パラメータの保管に使用されます．

NAND型は，ブロック単位での読み書きしかできませんが，ビットあたりの単価が安く，消去や書き込みの速度も速く大容量化に向いているので，ディジタル・カメラなどの携帯機器の可搬型外部記憶デバイスとして使用されています．ただし，NAND型は，読み出し時にビット・エラーが発生する場合があるため，冗長ビットを付加してエラー訂正を行い，利用セルの再配置を行うなどのくふうをして信頼性や寿命を向上させる必要があるため，制御用のハードウェア回路やソフトウェアが必要になります．

最近のLinuxでは，MTD（Memory Technology Device）として，ROMを扱い，ファイル・システムとしてマウントして使用することができるようになりました．NAND型フラッシュ・メモリもサポートされていますが，これを利用するブートローダやファームウェアの開発では注意が必要です．

そのほか，16ビット程度までの比較的小規模の組み込み系CPUでは，CPUチップにフラッシュ・メモリやRAMまでを内蔵し，外付けメモリがなくてもシステムを構築できるように配慮しているものも普及してきています．また，大容量の不揮発性メモリを必要とするシステムの開発では，比較的コストを安価にできるバッテリ・バックアップRAMを使用する場合があります．

センス内容について確認する必要があります．

● OSのブート・コード

OSの種類によっても異なりますが，起動時にOSのすべてのコードをメモリに展開して実行するのではなく，段階的に起動する場合があります．OSのコードの中でも，次のようなカーネル本体の起動時だけに実行する部分をOSのブート・コードと呼びます．組み込みLinuxなどでは，カーネルのロード方法や動作環境に合わせてOSのブート・コードを修正または作成する必要があり，この項目とブートローダが移植の際には技術的なポイントとなります．

a）カーネルの動作に必要なメモリ領域，CPU，RAM，割り込みベクタの設定
b）起動に必要なファイルをメモリ上に展開するRAMDISKを作成
c）圧縮したカーネルやRAMDISKイメージを読み込んでメモリに展開
d）ファイル・システムのマウントに必要な外部記憶装置やネットワークとUIの初期化

● リセット，ハードウェア・リセット，ソフトウェア・リセット

電源ON時は，CPUごとに実行する命令のアドレスが決まっています．ハードウェア・リセットは，リセット・ボタンなどの外部からの要求で，システムを構成するCPUをはじめとする各種コントローラに電気信号的にリセット信号を送ることにより，電源ON直後と同様の状況を作り，起動動作から再実行することを言います．

これに対して，ソフトウェア・リセットは，リセット信号の助けを借りずにソフトウェアでCPUをはじめとする各種コントローラを初期化して，電源ON時と同じアドレスから起動し直すことを言います．このときにどの程度の初期化を行うかについては，ソフトウェア・リセットを実行するコードに依存しますが，通常はソフトウェア・リセットを行っても問題なく動作するようにシステムを開発します．

一般にリセットというと，ハードウェア・リセットを指します．本章では，以降リセットと電源ONが同じ動作をすると仮定して解説します．

コラム1にBIOSに関する簡単な解説を，コラム2にはROMとフラッシュ・メモリの解説を載せました．以降は，PCアーキテクチャにはあまりとらわれずに，より広い意味でのブートローダ，とくに組み込み系システムでのブートローダを中心にして説明していきます．

ブートローダの種類と現状

それでは実際に，どのようにしてブートローダは開発されているのでしょうか．その前に，以下に一般的にブートローダと呼ばれるものや，ブートローダの機能を実現しているものを例示して，コードの入手方法の観点から種類別に整理します．まず，**表1**におもなブートローダをまとめて整理しました．

● 開発，配布，流通形態による分類

▶ オープン・ソース系（例：RedBoot，Das U-boot）

おもにボランティアの手によって開発され，インターネットで公開されているブートローダです．もともとブートローダとして開発されているため，完成度は高いのですが，対応するアーキテクチャはある程度限定されています．

New Products　ハード・リアルタイム機能を搭載したMontaVista Linux Professional Edition 4.0がリリース
米MontaVista Software社は，組み込み向けLinux「MontaVista Linux Professional Edition 4.0」をリリースした．ハード・リアルタイム機能を搭載している．Linux Kernel2.6.10に対応．

表1　公開されているおもなブートローダの一覧

名　前	説　明	動作環境	URL
LILO	「LInux LOader」の略	x86 PC	http://www.ibiblio.org/pub/Linux/system/boot/lilo/!INDEX.html
GRUB	「GRand Unified Bootloader」の略	x86 PC	http://www.gnu.org/software/grub/
syslinux	FD ブート用 Linux ローダ	x86 PC	http://syslinux.zytor.com/
loadlin	MS-DOS 用 Linux ローダ	x86 PC	http://elserv.ffm.fgan.de/~lermen/
TCCBOOT	ソースからコンパイルする Linux ローダ	x86 PC	http://fabrice.bellard.free.fr/tcc/tccboot.html
MBM	マルチブート・マネージャ	x86 PC	http://elm-chan.org/fsw/mbm/mbm.html
Booteasy	BSD 系 OS 用ブートセレクタ	x86 PC	http://www.openbsd.org/ja/ http://www.jp.freebsd.org/
Etherboot	ネットワーク・ブートツール	x86 PC	http://etherboot.sourceforge.net/
NILO	Network Interface Loader	x86 PC	http://nilo.sourceforge.net/
tinyBIOS	x86 系組み込み用 BIOS	x86 Embedded	http://www.pcengines.ch/tinybios.htm
ROLO	SYSGO ElinOS 用のブートローダ	x86Embedded, PPC, ARM, MIPS, SH	http://www.elinos.com/index.php?id=116&L=1 ftp://ftp.sysgo.de/pub/elinos/rolo/
LinuxBIOS	Linux 専用オープン・ソース BIOS	x86 BIOS	http://www.linuxlabs.com/linuxbios.html
OpenBIOS	OpenBIOS	x86 PC ほか	http://www.openbios.info/
IBM SLOF	JS20 blade server 用 Open Firmware	PowerPC	http://www-128.ibm.com/developerworks/power/library/pa-dw-slof.html
RedBoot	eCos ベースの汎用ブートローダ	ARM, 68K, PPC, MIPS, H8, SH, ほか	http://sources.redhat.com/redboot/
Yaboot	オープン・ソース Open Firmware ブートローダ	PPC	http://penguinppc.org/bootloaders/yaboot/
Das U-boot	汎用ブートローダ	PPC, MIPS, ARM ほか	http://sourceforge.net/projects/u-boot
YAMON	MIPS 用ブートローダ	MIPS	http://www.mips.com/content/Products/SoftwareTools
sh-boot, ipl+g	SH 用ブートローダ	SH	http://cvs.linux-sh.org/viewcvs.py/linuxsh/sh-boot/
Hermit	ARM Linux 用ブートローダ	ARM	http://www.bluemug.com/~miket/arm/arm.html
Lucent MicroMonitor	組み込み開発用ファームウェア・ソース・コード・パッケージ	PPC, SH-2, 68K	http://www.bell-labs.com/topic/swdist http://www-out.bell-labs.com:80/project/micromonitor/
Blob	「Boot Loader OBject」の略	ARM	http://www.lart.tudelft.nl/lartware/blob/
OS Loader	Windows CE 用 OS ローダ	iPAQ	http://www.handhelds.org/Compaq/iPAQH3600/OSloader.html

▶CPU メーカ提供（例：ルネサス H8 モニタ，Motorola BUG）

CPU やボード・メーカがおもに製品の評価目的に提供する ROM モニタやデバッガです．無償または比較的安価で入手できますが，機能や拡張性が乏しい場合があります．

▶OS メーカ製提供（例： VxWorks，Windows CE など）

組み込み系 OS メーカでは，自社の OS を起動するためのブート ROM に搭載するブートローダを供給しています．実行・デバッグ環境との相性も良く，サポートも受けられますが，それなりの費用がかかります．

▶ ボード・メーカ提供（例： E!Kit-1100 など）

マイコン評価ボードを購入すると，サポートする OS を起動するためのブート ROM が付属します．オープン・ソース系 CPU メーカ提供のものに，独自に手を入れたものが多いのですが，中には独自にブート ROM を開発しているメーカもあります．

▶ 専用メーカが提供（例： PC の BIOS，BASIC）

今では CPU，ボードや OS を作らずにブートローダ（BIOS）を開発，販売しているのは，Award 社を買収した Phoenix 社と AMI 社の PC の BIOS メーカ 2 社に絞られています．かつては Microsoft 社が開発した MS Basic の ROM 版も，PC アプリケーションのブートローダとして利用されていました．

▶ 独自開発

最初から独自に開発する場合，社内や外注先で以前作ったものの流用や改造，本来はブートローダではなかったファームウェアや OS を流用して改造する場合があります．

▶ 明確なブートローダがない場合

ファームウェアのようにシステム構成が比較的簡単で，システムを起動するコードと主要な制御を行うコードと明確に分かれてない場合もあります．

● OpenBoot，Open Firmware と OpenBIOS

前述のカテゴリに区分けしにくいブートローダとして Open Boot，Open Firmware，OpenBIOS の系統があります．

Open Firmware はもともと，Motorola 社などが PowerPC プラットホームの普及のための規格を制定した際に，起動する OS に依存しない BIOS 兼ブートローダの仕様をまとめたものでした．

これは，Sun Microsystems 社の SPARC 系マシンで使われていた OpenBoot というブートローダを元にして，IEEE 1275-

1994（http://playground.sun.com/1275/home.html）として規格化したものです．現在はPowerPCのMacintoshとIBM社のPowerPCサーバで採用され，Sun Microsystems社のマシンにも規格に準拠したものが搭載されています．Forth言語インタプリタをAPIとして搭載して，デバイスや構成情報の設定をすべてForthで行うのが特徴です．

OpenBIOS（http://www.openbios.info/）は，PowerPCプラットホームとそれ以外（具体的にはx86PC，AMD64など）で動作するIEEE 1275準拠のOpen Firmwareの実装コードを開発し，動作させようというプロジェクトで，すでに開発済みのソース・コードが公開されています．これらは，コラム1に示したPC用フリーBIOSの開発や後継機能開発とは，別の動きとなっています．

ブートローダの役割と動作

ブートローダの開発の際にチェックしておくべきブートローダのおもな役割と動作について以下に整理します．ブートローダでは，自分自身で入出力制御ルーチンや，メモリなどのリソース管理をもち，これらの機能をOSの助けを借りずに実装することになります．

● ブートローダのおもな役割

▶ 初期化

CPUの初期化

メモリ設定（ROM，RAMの設定，必要であればRAMにROM内容をコピー，メモリのコンフィグレーション・レジスタとチップ・セレクト設定，プログラム中に使用するメモリをすべて使えるようにする）

通信，UI，周辺コントローラ（ブート動作で使用するデバイスを初期化，設定）

▶ ユーザ・インターフェース（システム構成に依存）

起動メッセージやプロンプトの表示

エラー・メッセージ，ログ・メッセージの出力先設定

ユーザ指示の入力，メニュー機能，ヒストリ機能，設定環境の記憶

デバッグ機能の呼び出し

▶ 通信（システム構成に依存）

シリアル通信（フロー制御，速度，パリティ，バイナリ・データ通信）

TCP/IP（自ホスト，ネットワーク，相手先，プロトコルの設定）

ほかの通信方式の場合はそれらに依存した初期設定

▶ デバッグ（システム構成，CPUの機能，周辺コントローラの機能に依存）

メモリ，周辺I/O，通信，タイマ，UI，プログラム・ロード

レジスタ参照，メモリ参照，ステップ実行，ブレークポイント

▶ フラッシュ・メモリ管理（必要な場合だけ）

メモリ領域管理（特定領域の保護，消去，書き出し，コピー）

▶ プログラムのロードと起動

ファイル形式の変換（Sレコード，HEXフォーマット，バイナリ形式）

実行時パラメータの設定と受け渡し，プログラムのロードと起動

● ブートローダの動作

それでは実際に，どのようにしてブートローダは実装されて，動作しているのでしょうか．**図1**に，一般的なPCでLinuxが起動するまでの制御が渡る順序と，動作内容の概略を示します．ここでBIOSはブートROMに，ディスク・ブートローダはブートディスクのMBRとそれに続く領域に，Linuxカーネルはディスク上のファイルとして格納されていて，それぞれが別々に開発されている，独立したプログラム・コードであることに注意してください．

バケツ・リレーのように，各モジュールがCPUの制御を受け渡しているようにも見えますが，**図2**のように，システムの起動が進むにつれてプログラムが動作する環境を作り直して，住み替えているといった表現が近いと思います．

ここで大切なことは，使用する必要最低限のデバイス（コントローラ）を制御するための，必要最小限度のコードだけが必要だということです．四畳半のアパート暮らしのブート初期段階から，乗る予定がないベンツを買って用意しておく必要はないということです．そのためには，ブート時に利用するデバイスや機能を洗い出してチェックしておく必要があります．

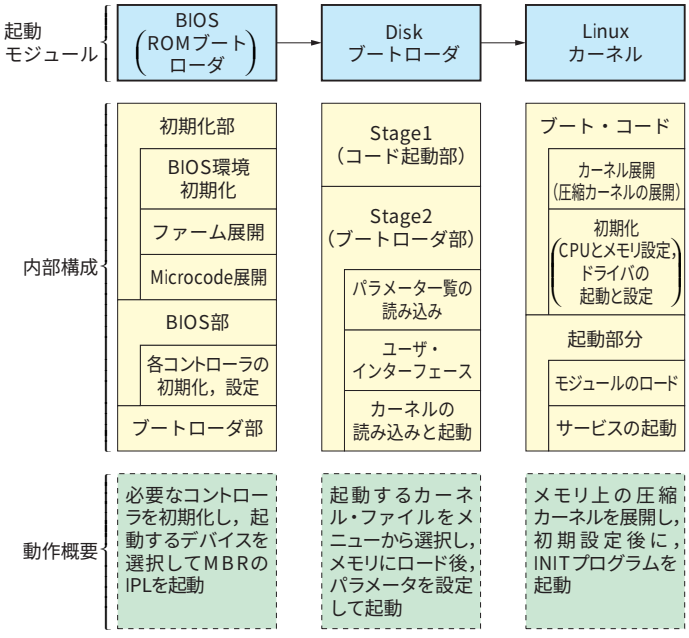

図1　PCでのLinuxの起動シーケンス

図2　CPU君の住みかえ――システムの起動が進むにつれてプログラムが動作する環境を作り直していく

たとえば，ストレージからOSをブートする場合には，そのストレージやファイル・システムの読み出し機能だけを実装すればよく，tftpでOSを転送する場合には，UDPとtftpを実装すればよいことになります．

しかし開発時には，結果的に同様の機能をもつコードを環境別に2種類も3種類も作ってしまうという状況はありえます．評価プログラムやブートローダを開発して，その後にOSの移植をするという場合は，先に目的のデバイスをOSなしで利用する評価プログラムを開発し，その後でOS用のデバイス・ドライバを開発することになります．一見むだに見える開発手順ですが，このような開発の積み重ねで，信頼性のあるシステムが開発できるようになります．

ブートローダ開発のガイドライン

それでは一般的なブートローダ（ブートROM）の開発はどのように行われるのでしょうか．以下に開発に必要な環境とツールをカテゴリ別に示します．

● 開発に必要な環境

ブートローダの開発には一般に以下のようなものが必要です．

▶ クロス開発用ツールチェイン

OSの元で実行されるプログラムは，実行環境のOSにプログラムの情報を与えるためのヘッダを持ち，OSからロードされるときに実行するアドレスが動的に決定されます．ブートローダの開発とは，OSなしで動作するプログラムを開発するということなので，クロス開発となります．クロス開発に必要なクロス・コンパイラ，クロス・アセンブラ，リンカ，ライブラリをツールチェインと呼び，専用のコンパイル，アセンブル環境を用意する必要があります．

これらのツールチェインは，ブートローダの配布と同様に，各メーカが開発環境として有償/無償で配布していますが，そのほかにGNUなどのオープン・ソース・ツールを利用することも可能です．

ブートローダの開発で重要なのは，固定番地に配置して，OSなしで直接単体で実行できるオブジェクト・モジュールを作成するリンク操作（アドレスとシンボルの解決）が可能であることです．このためOSのシステム・コールに依存しないライブラリを自作したり，必要に応じて改造したりする場合もありますし，リンカを特別なパラメータを付けて使用する場合もあります．たとえばGCCを使用する場合には，リンカ・スクリプトの書き方を調べておきます．

▶ ターゲット環境

ブートローダの開発では，実験で使用するターゲット環境が必要です．開発の状況によっては最初から実機が使用することができないために，同じCPUのほかのボードを使用したり，エミュレーション環境を利用する場合もありますが，最終的には実機でテストをします．

▶ プログラム転送手段とデバッガ

開発したプログラムをテストするために，ターゲット用の実行環境に転送する手段が必要です．最初からブートROMにROMライタで書き込んでテストをすることも可能ですが，通常はすでに同様の環境下で稼動しているモニタやデバッガを利用して，RAMにブートローダ・プログラムの全部，または一部を転送してテストを行います．

ROMに書き込んだ内容は，RAMのように通常の手順で読み書きができないため，開発の途中では，ROMのプログラムを頻繁に書き換えたり，一時的に一部分を修正して実行したり，ROMデバッガでブレークポイントを設定してデバッグすることができません．そのため，そのようなブートROMの開発では，JTAGデバッガやROMエミュレータなどのデバッグ機器を使用します．

動作確認やデバッグのためには，ステップ実行やアセンブラ命令を記述して実行できるデバッガは有効で，とくにプログラムがうまく動作しないときに，ハードウェアが正しく動作して

Information――「bo:heap」によるバッファ・オーバフロー攻撃が多発
マカフィー（株）の8月のコンピュータ・ウイルスの検知状況によると，マシン数および企業数の両方で，ウィルスではないものの，ヒープ・オーバフローを引き起こす「bo:heap」が1位となった．2位はExploit-URLSpoof.gen（トロイの木馬型）だった．

いるのかを検証することができます．また，JTAGデバッガは実機上のブートROMに新規にプログラムを転送する場合や，内容を書き換える場合にも使用します．

● 開発時やデータ・シートを読むために必要な知識

ブートローダを開発することは，システムが電源ON時に，何を行うべきかをすべて把握しておく必要があります．そのためには以下に挙げるような点に注意し，メーカが提供するCPUやコントローラ，ターゲット・ボードのプログラミング・マニュアルなどのデータ・シートを読むことが必要です．

- 各マニュアルの記述内容と役割
- CPUやコントローラの内部構造，制御レジスタと設定方法
- サンプルのプログラム・コード

そのほかに必要と思われる知識を以下にまとめます．

▶ アセンブラの読み書き

ハードウェアが実行している命令やデータを直接読み，また，初期化のプログラムを書くために，アセンブラを読み書きする知識は必要です．初心者には抵抗があるかもしれませんが，すべての命令やテクニックを身に付ける必要はなく，単純な記法や命令，疑似命令は，CPUアーキテクチャによる違いも少なく，習うよりは慣れるつもりでいたほうがよいと思います．

アセンブラでのコーディングは必要最低限でよいのですが，C言語で書いたコードがどのようなアセンブラ命令に変換されるか，C言語のマクロで直接アセンブラを記述するにはどうしたらよいかという点は，押さえておく必要があります．そのほかに実機での，デバッガのステップ実行などを経験することで，徐々に習得していくと思います．

▶移植時の問題：ポートI/Oとメモリ・マップドI/O，エンディアン

CPUのアーキテクチャに依存する方式の違いなので，プログラムの移植をするときには注意が必要です．ポートI/Oとメモリ・マップドI/Oは，入出力に使用する命令とアドレスの指定方法が異なるだけなので，アクセスのワード幅に気をつければ，ある程度機械的に書き換えができます．エンディアンの違いについては，アクセスするビット数，コントローラごとの設定や，回路構成によっても操作が異なるため，注意が必要です．

▶ コントローラの制御とレジスタのアクセス

各種コントローラの操作は，コントローラのインターフェースに相当する各レジスタの各ビットが“H”(1を書いたとき)でアクティブなのか，“L”(0を書いたとき)でアクティブなのかに注意します．制御をするときにはビット操作命令を使用してコーディングしますが，書き込み時には，関係ない制御ビットの値を変えてしまわないように，OR命令やマスク・パターンを使用したAND命令を使用します．

コントローラによっては，通常に読み書き可能なもののほか，読み出し専用，書き込み専用，読み出しと書き込みでレジスタの実体(動作)が異なるもの，読み書きに一定の手順が必要なものがあります．

▶ チップ・セレクト

ROMやRAMなどのメモリは通常，物理的にアドレス信号で接続してCPUのメモリ空間上に配置されますが，配線を簡便にして消費電力を抑えるために，チップ・セレクト信号で接続する場合があります．

たとえば，特定のROMが“L”でアクティブなチップ・セレクト信号CS#1で接続された場合，そのメモリ空間にアクセスするためには，CS#1にゼロを設定する必要があります．しかしアクセスのたびにCS#1にゼロを設定(“L”を入力)するのはたいへんなので，CPUや回路構成によっては，アドレス・デコーダの出力ラインをチップ・セレクト信号に接続することで，特定メモリにアクセスすると自動的にCS#1にゼロを設定して，目的のメモリにアクセスできるようにするものがあります．そのような機構を利用するためには，チップ・セレクトとメモリ・アドレスの関連付けを初期設定する必要があります．

▶ 割り込みの動作，例外処理と例外処理ベクタ

CPUは通常，プログラム・カウンタを増やしながら，メモリ中にある命令(機械語プログラム)を逐次読み込んで動作します．そのほかに，CPUでは外部割り込みやゼロ除算などの例外が発生した場合，特定のメモリ(要因別に飛び先メモリが配列状に並んでいることからベクタと呼ぶ)からアドレス値を読み

Column 3

eCos，Newlib，uClibc，OSKit

eCos(http://sourceware.org/ecos/)はRedBootとともにRedHat社が開発，配布しているμITRON Ver.3系のOSです．実はRedBootはeCosそのものです．このようにブートローダとして，ほかのOSを流用する例は最近では少なくありません．

Newlib(http://sources.redhat.com/newlib/)はOSなしで動作することを想定した，組み込みクロス開発環境用のソース・コード・ライブラリでRedHat社が配布しています．uClibc(http://www.uclibc.org/)はOSの元での利用を想定している小規模ライブラリですが，ライブラリをOSなしで利用するように構築することも可能です．これらをうまく利用すれことができれば，開発に必要なライブラリを，一から作る必要はありません．

OSKit(http://www.cs.utah.edu/flux/oskit/)はRedHat社が開発，配布しているOSに依存しない主要なデバイス・ドライバやサービスを含む，OS開発用のライブラリ・キットです．OSKitはカーネルローダ，TCP/IPプロトコル・スタック，ファイル・システム・インターフェースやサンプル・カーネルを含み，簡単なアプリケーションの実行環境として利用するようになっているため，ファームウェアやブートローダの開発にも利用できます．実装例としては，OSKit-Machが有名です．

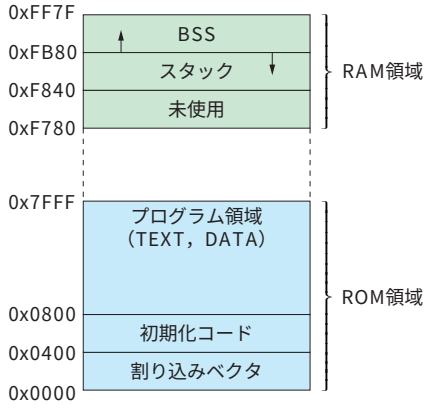

図3　H8/3694Fのメモリ・マップ

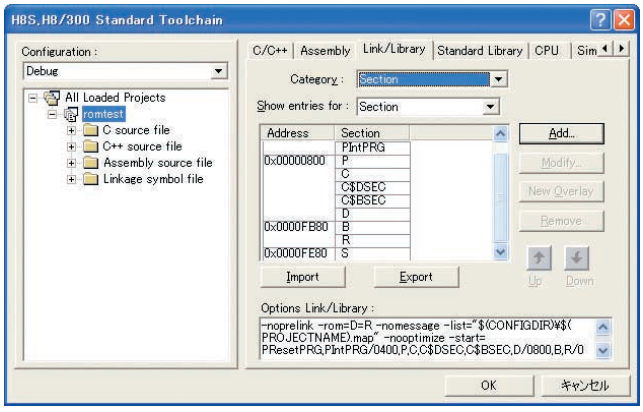

図4　H8/3694Fリンカの設定

込んで，そのアドレスにジャンプするという動作を行います．これがCPUの割り込み動作です．

割り込み動作は，コンディション・レジスタで割り込みが禁止されていない場合には，いつでも発生します．しかし，割り込み動作が終了したときに元の位置に戻って動作が継続できるように，割り込み前の状態をスタック・ポインタが指し示しているスタック・メモリに一時的に保管します．このようなCPUの動作から，電源ON時に実行するブートローダには，次のような処理を行う必要があることがわかります．

- スタック・ポインタの設定
- コンディション・レジスタの設定
- 例外処理ルーチンの用意
- 例外処理ベクタ・アドレスの設定

▶ メモリ・マップとROM化方法

システム全体のメモリ・マップを把握しておく必要があります．とくにブートローダに必要なROM化を行うときには，BSS（初期化されないデータ），スタック（自動変数割り当てやレジスタの退避領域），HEAP（動的メモリ確保領域）はRAM領域に設定して，TEXT（プログラムのコード）とDATA（初期値付きデータ）をROMに設定する必要があります．リンカの設定で，これらに固定的にアドレスに割り付ける必要があるため，注意が必要です．

トランジスタ技術の2004年4月号のH8/3694Fマイコン・ボードを使用して，試験的に作成した簡易ブートローダもどきプログラムのメモリ・マップを**図3**に，HEWを使用したリンカでのメモリ割り当て設定を**図4**に示します．

＊　　＊

簡単ですがここまで駆け足で，ブートローダの種類と現状，実装すべき機能と，開発方法について解説してきました．従来は独自にデータ・シートを見ながら，ブートローダやファームウェアを開発してきた事例も多かったのですが，最近の傾向としては，多数公開されているオープン・ソース系のプログラムを参考にするか，改造して利用する例が増えていると思います．また，コラム3のように，組み込み系システムの開発に必要なプログラム・モジュールをOSやライブラリの形にして公開している例もあるので，これらもブートローダに限らずファームウェアや組み込みシステムの開発に利用されています．

参考文献

(1) 特集 組み込みシステムの世界へようこそ！，Interface，2004年5月号，CQ出版社．
(2) 特集 フリーソフトウェア活用組み込みプログラミング，Interface，2005年1月号，CQ出版社．
(3) 特集 付属基板で始めるマイコン入門，トランジスタ技術，2004年4月号，CQ出版社．

ひだか・あとむ　（株）デバイスドライバーズ