第1章

ARM組み込みシステム

ARMプロセッサ・コアは多くの32ビット組み込みシステムのキー・コンポーネントとなっています．お気づきでないかもしれませんが，だれでも身の回りにARMプロセッサを一つは持っています．ARMコアは携帯電話，電子手帳など，日常的に使用するさまざまな携帯用機器に使用されています．

ARMの開発は，1985年の最初のARM1プロトタイプの設計から長い道程を経てきました．2001年末には，全世界におけるARMプロセッサの出荷個数が1億個を突破しました[注1.1]．会社としてのARMに成功をもたらした当初のシンプルで高性能な設計は，現在でも着実な技術革新によって改良が続けられています．ARMコアは実は一つのコアだけではありません．同じ設計理念と共通の命令セットをもつファミリ全体を指します．

たとえば，ARM製品の中でもっとも売れているコアはARM7TDMIです．ARM7TDMIは最大120 Dhrystone MIPS[注1.2]の性能があり，携帯組み込み機器に最適な高コード効率と低消費電力で知られています．

第1章では，ARMがどのような経緯でRISC（Reduced Instruction Set Computer）の設計理念を採用し，柔軟性に優れた組み込みプロセッサを開発したかを説明します．その後，組み込みデバイスの例を挙げ，ARMプロセッサを取り巻く典型的なハードウェアとソフトウェアを紹介します．

1.1 RISCの設計理念

ARMコアはRISCアーキテクチャを採用しています．RISCとは，高速なクロックでも1サイクルで実行する，単純で強力な命令を使用するという設計思想です．つまり，ハードウェアが実行する命令の複雑さを軽減することに集約されます．なぜならばそのほうが，ソフトウェアの柔軟性と理解力を高めるのが容易になるからです．このため，RISC設計ではコンパイラに課される条件が厳しくなります．逆に，従来のCISC（Complex Instruction Set Computer）は，おもにハードウェアで命令の機能を果たそうとするために，命令が複雑になります．**図1.1**にRISCとCISCのおもな違いを示します．

RISCの考え方はおもに次の四つの設計規則で実現されています．

1. 命令 — RISCプロセッサは一般に命令クラスの数が少ない．各クラスは1サイクルで実行される単純な

注1.1：2006年度の出荷個数は24億5000万個．

注1.2：Dhrystone MIPSバージョン2.1は小型のベンチマーク・プログラム．

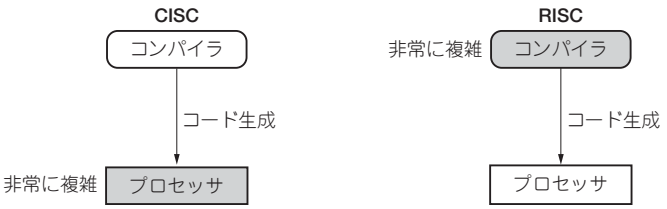

図 1.1
CISC と RISC の比較．CISC はハードウェアが複雑．RISC はコンパイラが複雑．

演算命令で構成される．コンパイラまたはプログラマは複数の単純な命令を組み合わせて複雑な演算（除算など）命令を合成する．各命令の長さは一定のため，パイプラインは現在の命令をデコードする前に次の命令をフェッチできる．CISC プロセッサの場合は，命令のサイズが可変であることが多く，実行に多くのサイクルが必要である．

2. **パイプライン** — 命令の処理は小さな単位に分割され，パイプラインで並行して実行される．理想的にはパイプラインが各サイクルで 1 段階ずつ処理し，最大のスループットを達成する．命令はパイプラインの 1 段でデコードされる．CISC プロセッサと異なり，命令をマイクロコードと呼ばれるミニプログラムで実行する必要はない．
3. **レジスタ** — RISC マシンはおもに多数の汎用レジスタ群で構成される．各レジスタはデータかアドレスのいずれかを保持することができる．レジスタはすべてのデータ処理命令において高速ローカル・メモリとして機能する．CISC プロセッサはおもに目的の決まった専用レジスタで構成される．
4. **ロード・ストア・アーキテクチャ** — プロセッサはレジスタの保持しているデータの演算を行う．独立したロード命令やストア命令はレジスタ・バンクと外部メモリ間でデータを転送する．メモリ・アクセスには負荷がかかるため，データ処理とメモリ・アクセスを分けたほうが有利である．そうすれば，1 回のメモリ・アクセスでレジスタ・バンクに保持されたデータ・アイテムを何度も使用できるからである．対照的に，CISC 設計ではデータ処理命令が直接メモリにアクセスを行う．

このような設計規則により，RISC プロセッサは構造が単純なので，コアは高いクロック周波数で動作可能です．それにひきかえ，従来の CISC プロセッサは複雑で，低いクロック周波数でしか動作できません．しかし，20 年の間に CISC プロセッサも RISC の設計概念を多く採用し，RISC と CISC の区別があいまいになってきています．

1.2 ARM の設計理念

ARM プロセッサの設計を推進してきたのは，さまざまな物理的特性です．まず，携帯用の組み込みシステムには何らかの電池電源が必要です．ARM プロセッサは小型化によって消費電力を削減し，バッテリ寿命を伸ばすことをとくに重視して設計されています．これは携帯電話や PDA（携帯情報端末）などのアプリケーションにとっては不可欠なことです．

高いコード効率も重要な条件の一つです．なぜなら，コストや物理的な大きさの制限を受ける組み込みシステムには，限られたメモリしか搭載できません．高いコード効率は携帯電話や大容量記憶装置などのオンボード・メモリの少ないアプリケーションに有利です．

さらに，組み込みシステムには価格の制約があるため，低コストの低速メモリ・デバイスを使用しています．デジタル・カメラなどの量産製品の場合は1円でもコストを削減したいところです．低コストのメモリ・デバイスを使用できれば，大幅な節約になります．

もう一つ重要な条件は，ダイの上で組み込みプロセッサが占める面積を縮小することです．シングル・チップの場合は組み込みプロセッサの占める面積が小さければ小さいほど，特別な用途の周辺回路に使える面積が広くなります．その結果，最終製品に必要なディスクリート・チップが少なくなり，設計と製造にかかるコストが下がります．

ARMはプロセッサ内にハードウェア・デバッグ技術を組み込んでいるため，ソフトウェア・エンジニアはプロセッサがコードを実行する際に何が起こっているかを見ることができます．この視認性の向上により，ソフトウェア・エンジニアは問題を速く解決できます．これは製品化期間に直接影響を与え，全体的な開発コストを削減します．

ARMコアはおもな用途が組み込みシステムであるという制約から，純粋なRISCアーキテクチャではありません．ある意味で，ARMコアの強みはRISCのコンセプトにおいて行き過ぎていないところかもしれません．現在のシステムでもっとも重要なのは純粋なプロセッサ速度ではなく，有効な全体のシステム性能と消費電力だからです．

1.2.1 組み込みシステム向けの命令セット

ARM命令セットは純粋なRISCの定義とは複数の点で異なります．これがARM命令セットが組み込みアプリケーションに適している理由です．

■ **一部の命令の実行サイクル数が変化する** ― すべてのARM命令が1サイクルで実行されるわけではない．たとえば，複数ロード・ストア命令の実行サイクル数は転送されるレジスタ数によって変化する．転送はシーケンシャル・メモリ・アドレスで発生する．シーケンシャル・アクセスはランダム・アクセスより速いことが多いため，性能が高まる．複数のレジスタ転送は関数の最初と最後によく起こる処理なので，コード密度も高くなる．

■ **複雑な命令につながるインライン・バレル・シフタ** ― インライン・バレル・シフタは，命令で使用される前に入力レジスタを前処理するハードウェア・コンポーネントである．これは命令の機能を拡大し，コア性能とコード密度を改善する．この特徴については，第2章，第3章，第4章で詳しく説明する．

■ **Thumb 16ビット命令セット** ― ARMはプロセッサ・コアを拡張し，Thumb[注1.3]と呼ばれる第2の16ビット命令セットを追加した．このためARMコアは16ビット命令も32ビット命令も実行できる．16ビット命令は32ビットの固定長命令に比べ，コード密度を約30%高める．

■ **条件実行** ― 特定の条件を満たした場合にだけ命令が実行される．これにより，分岐命令が減り，性能とコード密度が高まる．

■ **拡張命令** ― 高速16ビット×16ビット乗算演算と飽和演算に対応する拡張ディジタル信号処理（DSP）命令が標準的なARM命令セットに追加された．これらの命令はプロセッサにDSPを加えた従来の組み合わせを置き換えることにより，ARMプロセッサの性能の向上を可能にする．

注1.3：アーキテクチャv6でThumb2命令セットを追加．

以上の追加機能により，ARM プロセッサは一般的に使用される 32 ビット組み込みプロセッサ・コアの一つとなっています．世界の大手半導体企業の多くが ARM プロセッサを搭載した製品を生産しています．

1.3 組み込みシステムのハードウェア

組み込みシステムは生産ライン上の小さなセンサから NASA の宇宙探査機に使用されるリアルタイム制御システムまで，多くのデバイスを制御できます．このようなデバイスはすべてソフトウェア・コンポーネントとハードウェア・コンポーネントの組み合わせで構成されます．各コンポーネントは効率性を考慮して選択され，適宜，将来的な拡張にも対応するように設計されています．

図 1.2 に ARM コアを搭載した典型的な組み込み機器の構成を示します．各項目は特徴や機能を示しています．項目をつなぐ線はデータを転送するバスです．機器の構成は四つのおもなハードウェア・コンポーネントに分けることができます．

- **ARM プロセッサ**は組み込み機器を制御する．必要な動作に合わせて，さまざまなバージョンの ARM プロセッサが提供されている．ARM プロセッサは，コア（命令を処理し，データを操作する実行エンジン）と，コアとバスを接続する周辺コンポーネントで構成される．周辺コンポーネントにはメモリ管理装置やキャッシュも含まれることがある．
- **コントローラ**はシステムの重要な機能ブロックの動きを調整する．割り込みコントローラ，メモリ・コントローラの二つのコントローラが一般的．
- **ペリフェラル**はチップ外部のすべての入出力機能を提供し，その組み込み機器の独自性を決定する．
- **バス**はデバイスの各部分が通信するために使用される．

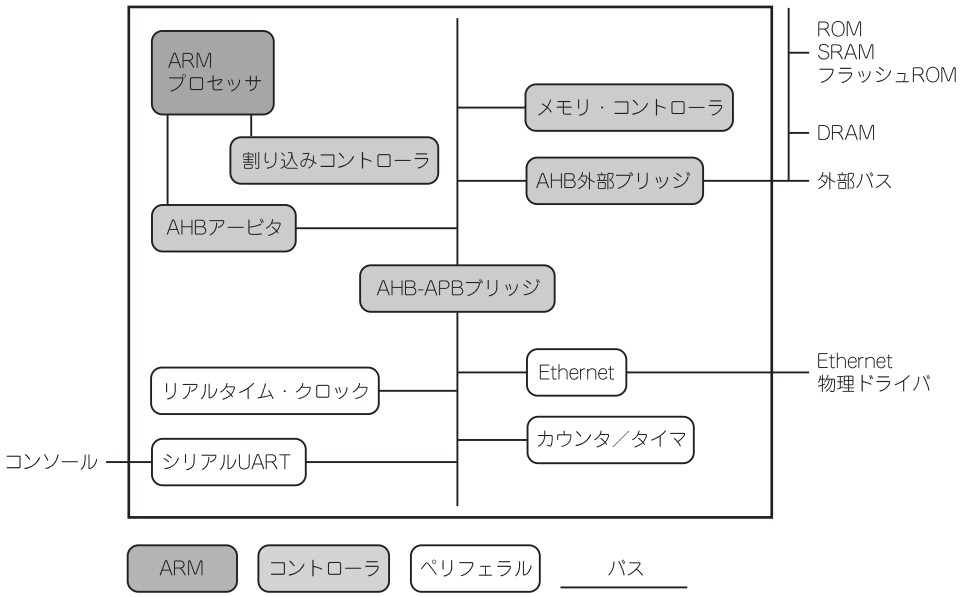

図 1.2　ARM を搭載した組み込みデバイスであるマイクロコントローラの例

1.3.1 ARMのバス技術

組み込みシステムはx86パソコン用に設計されたものだけでなく，さまざまなバス技術を使用します．もっとも一般的なパソコン用バス技術であるPCI（Peripheral Component Interconnect）は，ビデオ・カード，ハード・ディスク・コントローラなどのデバイスをx86プロセッサ・バスに接続します．この種類のものは周辺技術，すなわちオフチップ（バスが，チップの外にあるデバイスに機械的，電気的に接続する）であり，パソコンのマザーボードに組み込まれています．

しかし，組み込みデバイスはチップの内部にあるオンチップ・バスを使用します．オンチップ・バスはさまざまなペリフェラル・デバイスとARMコアの相互接続をします．

バスに接続するデバイスには二つのクラスがあります．ARMプロセッサ・コアは**バス・マスタ**，すなわち同じバス上にある別のデバイスとのデータ転送をする論理デバイスです．ペリフェラルは多くの場合，**バス・スレーブ**，すなわちバス・マスタからの転送要求に応答することしかできない論理デバイスです．

バスには二つのアーキテクチャ・レベルがあります．第1のレベルは電気的な特性とバス幅（16，32，64ビットのいずれか）に関係する物理レベルです．第2のレベルは**プロトコル**，すなわちプロセッサとペリフェラルの通信を規定する論理規則に関係します．

ARMは基本的に設計会社です．バスの電気的特性を規定することはほとんどありませんが，バス・プロトコルを指定することはよくあります．

1.3.2 AMBAバス・プロトコル

AMBA（Advanced Microcontroller Bus Architecture）は1996年に発表され，ARMプロセッサ向けのオンチップ・バス・アーキテクチャとして普及しています．最初に発表されたAMBAバスはARMシステム・バス（ASB）とARMペリフェラル・バス（APB）でした．のちにARMは，AHB（ARM High Performance Bus）[注1.4]と呼ばれるバス設計も発表しました．ペリフェラル設計者はAMBAを使用し，同じ設計を複数のプロジェクトに再使用することができます．AMBAインタフェースを使って開発されたペリフェラルは数多くあるため，ハードウェア設計者は実証されたペリフェラルを豊富な選択肢から選んで使用できます．ペリフェラルはオンチップ・バス上に固定するだけで済み，プロセッサ・アーキテクチャごとにインターフェースを再設計する必要はありません．このハードウェア開発者向けのプラグ・アンド・プレイ・インターフェースにより，納期と製品化期間が短縮されます．

集中型マルチプレクス・バス方式を採用したAHBは，ASBの双方向バス設計よりデータ転送容量において優れています．この変更により，AHBバスは高いクロック速度で動作できるようになり，64ビット，128ビット幅に対応する初のARMバスとなりました．ARMは，マルチレイヤAHB，AHB-Liteの2種類のAHBバスを発表しました．元のAHBはバス上で1回に一つのバス・マスタしかアクティブになれませんでしたが，マルチレイヤAHBバスは複数のバス・マスタをアクティブにできます．AHB-LiteはAHBバスのサブセットで，バス・マスタは一つに限られます．このバスは，標準的なAHBバスの全部の機能を必要としない設計向けに開発されました．

注1.4：2006年現在，AXIバスも発表されている．

AHB とマルチレイヤ AHB はマスタとスレーブについて同じプロトコルを使用しますが，相互接続の方法は異なります．マルチレイヤ AHB の新しい相互接続の方法は，複数のプロセッサを搭載したシステムに適しています．この相互接続は並列処理を可能にし，高いスループットを実現します．

図 1.2 に示したデバイスには三つのバスがあります．高性能ペリフェラル用の AHB バス，低速ペリフェラル用の APB バス，このデバイス特有の外部ペリフェラル用の第 3 のバスです．この外部バスには AHB バスと接続するための専用ブリッジが必要です．

1.3.3 メモリ

組み込みシステムにはコードを保存し，実行するための何らかのメモリ機能が必要です．階層，幅，タイプなど，メモリの特性を決める際には価格，性能，消費電力を比較することになります．必要な帯域幅を維持するためにメモリが 2 倍の速度で動作しなければならないような場合は，メモリに多くの電力が必要になります．

1.3.3.1 階層

すべてのコンピュータ・システムには，何らかの階層をなすメモリがあります．**図 1.2** は外部オフチップ・メモリをサポートするデバイスを示しています．プロセッサ内部にはメモリの性能を高めるキャッシュというオプション（**図 1.2** には示されていない）があります．

図 1.3 はメモリのトレードオフを示しています．もっとも速いメモリ・キャッシュが物理的に ARM プロセッサ・コアの近くに，もっとも遅い二次メモリが遠くに配置されています．一般にメモリがプロセッサ・コアに近いほどコストが高く，容量が小さくなります．

キャッシュはメイン・メモリとプロセッサ・コアの間に配置され，プロセッサとメイン・メモリの間のデータ転送を高速化します．キャッシュは，全体的な性能を高めますが，実行時間が予測不能になります．また，システムの全般的な性能は高くなりますが，リアルタイム・システムの応答性には貢献しません．多くの小型の組み込みシステムには，キャッシュの性能的利点が不要であることに注意してください．

メイン・メモリは大容量です．アプリケーションによっては約 256 K バイト～256 M バイト（もしくはそれ以上）に及び，一般に独立したチップに保存されます．ロード命令とストア命令は，データが高速アクセス用のキャッシュに保存されていない限り，メイン・メモリにアクセスします．二次ストレー

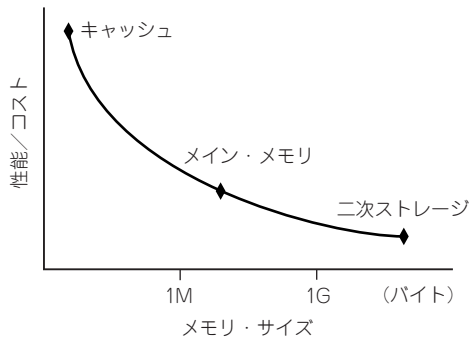

図 1.3
ストレージのトレード・オフ

ジはもっとも大きく，かつ低速のメモリです．二次ストレージにはハードディスク・ドライブやCD-ROMドライブがあります．近年，二次ストレージは600 Mバイト～60 Gバイトの容量があります．

1.3.3.2　幅

メモリ幅とはアクセスごとにメモリが戻すビット数です．一般的には8，16，32，64ビットのいずれかです．メモリ幅は全体的な性能と費用比率に直接効いてきます．

キャッシュなしのシステムで32ビットのARM命令と16ビット幅のメモリ・チップを使用する場合，プロセッサは1命令当たり二回のメモリ・フェッチを実行する必要があります．フェッチ1回には2回の16ビット・ロードが必要です．これでは，明らかにシステム性能が下がりますが，16ビット・メモリは低価格であるという利点があります．

しかし，プロセッサ・コアが16ビットThumb命令を実行する場合は16ビット・メモリのほうが性能が高くなります．これは，一つの命令をロードするのに，コアがメモリに対して1回しかフェッチを実行しないからです．したがって，16ビット幅のメモリ・デバイスでThumb命令を使用すると，性能向上とコスト削減の両方が実現します．

表1.1はさまざまなメモリ幅のデバイスを使用した場合のARMプロセッサにおける理論的なサイクル数をまとめたものです．

1.3.3.3　タイプ

メモリには多くの種類があります．ここでは，ARMベースの組み込みシステムにおいてもっともよく使用されるメモリ・デバイスについて説明します．

ROM（Read-Only Memory）は変更を必要としないプログラムされたイメージであり，再プログラムできないため，メモリ素子の中でもっとも柔軟性に欠けます．ROMは更新や修正が不要の大容量デバイスです．多くの機器はブート・コードの保持にもROMを使用します．

フラッシュROMは読み出しも書き込みもできますが，書き込みは遅いため，動的データの保持に使用するべきではありません．おもな用途はファームウェアの保持や，電源を切った後でも維持する必要のある長期的なデータの保存です．フラッシュROMの消去と書き込みは完全にソフトウェア制御で行え，ハードウェア回路の追加が必要ないため，製造コストが下がります．フラッシュROMは現在，大容量ストレージまたは二次ストレージの代用として，一般的に使用されている読み出し専用メモリです．

DRAM（Dynamic Random Access Memory）はもっとも一般的に使用されるRAMです．DRAMはほかのRAMにくらべ，Mバイト当たりのコストは低くなります．DRAMはダイナミック動作をします．すなわち，数msごとにストレージ・セルをリフレッシュし，新しい電荷を注入する必要があるため，メモリを使用する前にDRAMコントローラをセットアップする必要があります．

SRAM（Static Random Access Memory）は従来のDRAMより高速ですが，チップ面積が大きくなり

表1.1　メモリからの命令フェッチ

命令サイズ	8ビット・メモリ	16ビット・メモリ	32ビット・メモリ
ARM32ビット	4サイクル	2サイクル	1サイクル
Thumb 16ビット	2サイクル	1サイクル	1サイクル

ます．SRAMはスタティック動作です．すなわち，リフレッシュは不要です．SRAMはデータ・アクセス間のポーズ期間が不要のため，アクセス時間は同等のDRAMより大幅に短くなります．コストが高いため，高速メモリ，キャッシュなどの小さな高速タスクに使用されます．

SDRAM（Synchronous Dynamic Random Access Memory）はDRAMの下位区分の一つに分類され，従来のメモリよりも大幅に高速なクロックで動作します．SDRAMはプロセッサ・バスのクロックに同期化します．内部的には，データがメモリ・セルからフェッチされ，パイプライン処理され，最後にバーストでバス上に送られます．旧式のDRAMは非同期なので，SDRAMほど効率的にバーストしません．

1.3.4 ペリフェラル

組み込みシステムが外部の機器とデータの交換を行うには何らかのペリフェラル・デバイスが必要です．ペリフェラル・デバイスは外部のデバイスやセンサに接続することにより，入出力機能を果たします．各ペリフェラル・デバイスは通常，一つの機能をもち，内部に配置されていることもあります．ペリフェラルには単純なシリアル通信デバイスから複雑なIEEE802.11対応ワイヤレス・デバイスまで，多様なものがあります．

ARMペリフェラルはすべて**メモリに割り付け**られています．つまり，プログラムではメモリ・アドレスを指定したレジスタ群を指定します．これらのレジスタのアドレスは特定のペリフェラル・ベース・アドレスを基準にしたオフセットです．

コントローラは組み込みシステム内で高いレベルの機能性を実現するための専用ペリフェラルです．重要なコントローラとして，メモリ・コントローラと割り込みコントローラの二つがあります．

1.3.4.1 メモリ・コントローラ

メモリ・コントローラはさまざまなタイプのメモリをプロセッサ・バスに接続します．起動時には，ハードウェアでメモリ・コントローラのコンフィギュレーションが行われ，特定のメモリ・デバイスがアクティブになります．そして，これらのメモリ・デバイスが初期化コードを実行させます．一部のメモリ・デバイスはソフトウェアでセットアップする必要があります．たとえば，DRAMを使用する場合，まずメモリのタイミングとリフレッシュ・レートを設定しなければアクセスが可能になりません．

1.3.4.2 割り込みコントローラ

ペリフェラルやデバイスは必要に応じて，プロセッサに割り込みをかけます．割り込みコントローラはプログラム可能な管理規定があるため，通常ソフトウェアが割り込みコントローラ・レジスタの該当ビットをセットし，現在どのペリフェラルやデバイスがプロセッサに割り込めるかを決定しています．

ARMプロセッサでは標準割り込みコントローラとベクタ割り込みコントローラ（VIC）の2タイプの割り込みコントローラが使用できます．

標準割り込みコントローラは外部デバイスがサービスを要求すると，プロセッサ・コアに割り込み信号を送ります．個別のデバイスあるいはデバイス群をマスクしたり，無視したりするように，割り込みコントローラをプログラムすることも可能です．割り込みハンドラは割り込みコントローラのデバイス・ビットマップ・レジスタを読むことで，どのデバイスがサービスを要求しているかを判断します．

VICは割り込みに優先度を設定し，どのデバイスが割り込みを引き起こしたかの判断をシンプルに行うため，標準の割り込みコントローラより強力です．VICはまず各割り込みに優先度とハンドラのアドレスを結びつけ，新しい割り込みの優先度が現在実行中の割り込みハンドラより高い場合にだけ，コアへの割り込み信号をアサートします．そして，VICのタイプにより標準割り込み例外ハンドラを呼び出し，デバイスに対応するハンドラのアドレスをVICからロードするか，デバイスに対応するハンドラに直接ジャンプさせるかをします．

1.4　組み込みシステムのソフトウェア

組み込みシステムは動作するためのソフトウェアを必要とします．**図1.4**は組み込みデバイスの制御に必要な四つの標準的なソフトウェア・コンポーネントを示しています．スタックにある各ソフトウェア・コンポーネントは，高レベルの抽象化を行い，コードをハードウェア・デバイスから独立させます．

初期化コードはボード上で最初に実行されるコードであり，特定のターゲットまたはターゲット・グループに固有のものです．初期化コードはボードのうち最小限の部分をセットアップしてから，オペレーティング・システムに制御を渡します．

オペレーティング・システムはアプリケーションを制御し，ハードウェア・システム・リソースを管理するためのインフラを提供します．多くの組み込みシステムは完全なオペレーティング・システムではなく，イベント駆動またはポーリング駆動の単純なタスク・スケジューラで十分です．

デバイス・ドライバは**図1.4**の3番目のコンポーネントです．デバイス・ドライバはハードウェア・デバイス上のペリフェラルに整合するインターフェースを提供します．

最後に，アプリケーションは機器に必要なタスクの一つを実行します．たとえば，携帯電話のダイアリ・アプリケーションです．オペレーティング・システムによって制御されている機器上で，複数のアプリケーションが実行されることもあります．

ソフトウェア・コンポーネントはROMから動作することも，RAMから動作することもできます．デバイス上で固定されているROMコード（初期化コードなど）は**ファームウェア**と呼ばれます．

1.4.1　初期化（ブート）コード

初期化コード（ブート・コード）はプロセッサをリセット状態からオペレーティング・システムが動作できる状態にします．初期化コードは通常，メモリ・コントローラとプロセッサ・キャッシュの設定を行い，一部のデバイスを初期化します．単純なシステムではオペレーティング・システムの代わりに簡単なスケジューラやデバッグ・モニタが使用されることもあります．

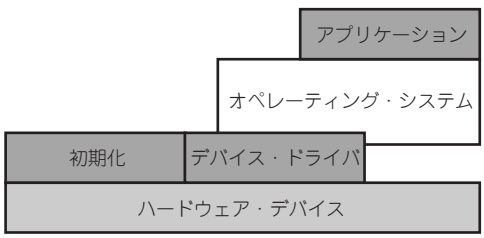

図1.4
ハードウェア上で実行されるソフトウェア抽象化レイヤ

初期化コードは多数の管理タスクを処理した上で，オペレーティング・システム・イメージに制御を渡します．これらのタスクは初期ハードウェア・コンフィギュレーション，システム・チェック，ブートの3段階に分けることができます．

初期ハードウェア・コンフィギュレーションではターゲット・プラットフォームをセットアップしてイメージをブートできるようにします．ターゲット・プラットフォーム自体にも標準コンフィギュレーションが準備されていますが，このコンフィギュレーションは通常，ブートするイメージの条件に合わせて変更する必要があります．たとえば，メモリ・システムは通常，**例1.1**のようなメモリ・マップの再編成を必要とします．

システム・チェックは多くの場合，初期化コードに組み込まれています．システム・チェックのコードはハードウェア・ターゲット上で，ターゲットが正常に動作する状態にあるかどうかを確認することにより，システムをテストします．また，標準的なシステム関連の不具合を見つけ出します．このようなテストはソフトウェア製品の完成後に行われるため，製造時において非常に重要です．システム・チェックのおもな目的は障害の特定と分離です．

ブートにはイメージのロードと，そのイメージへの制御の受け渡しが含まれます．システムが複数の異なるオペレーティング・システム，あるいは同じオペレーティング・システムの複数のバージョンをブートしなければならない場合，ブート処理自体が複雑となることがあります．

イメージのブートは最後の段階であり，まずはイメージをロードする必要があります．イメージのロードには，コードとデータを含むプログラム全体をRAMにコピーすることから，揮発性の変数を含むデータ領域だけをRAMにコピーすることまでのすべてが含まれます．いったんブートされると，システムはプログラム・カウンタをイメージの最初のアドレスを指し示すように変更し，制御を渡します．

イメージ・サイズを縮小するために，イメージが圧縮されることがあります．このイメージはロードされるとき，または制御が渡されるときに展開されます．

例 1.1　メモリの初期化つまりメモリ・マップを構成することは，初期化コードの重要な部分です．なぜなら，多くのオペレーティング・システムが動作を開始する前に既知のメモリ・レイアウトを要求するからです．

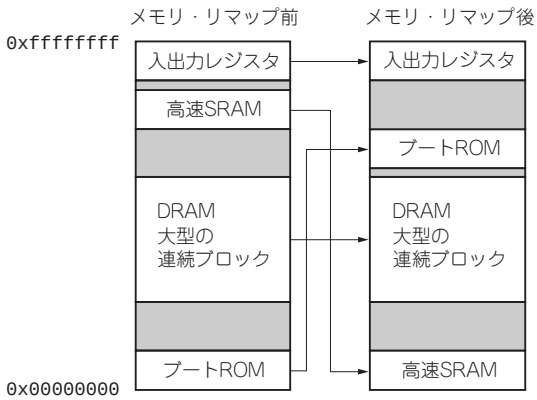

図1.5
メモリのリマップ

図1.5に再編成前後のメモリ・マップを示します．ARMベースの組み込みシステムは，通常，メモリ・リマップ機能を備えています．これにより，システムは起動時にROMから初期化コードを読み込み初期化を開始します．その後，初期化コードはメモリ・マップを再定義またはリマップし，RAMをアドレス0x00000000に置きます．これは重要な手順です．つまり，これにより例外ベクタ・テーブルがRAMに置かれ，リプログラム可能となるからです．ベクタ・テーブルの詳細は，2.4節で説明します．

1.4.2　オペレーティング・システム

初期化処理はオペレーティング・システムの制御に備えて，ハードウェアの準備を整えます．オペレーティング・システムはペリフェラル，メモリ，処理時間などのシステム・リソースを準備します．このようなリソースは，オペレーティング・システムの元で実行される多様なアプリケーションから効率的に使用されます．

ARMプロセッサは50以上のオペレーティング・システムに対応します．オペレーティング・システムはリアルタイム・オペレーティング・システム（RTOS）とプラットフォーム・オペレーティング・システムの二つの種類に分けられます．

RTOSはイベントに対する応答時間を保証します．システム応答時間をどれだけ制御できるかはオペレーティング・システムによって異なります．ハード・リアルタイム・アプリケーションの場合，応答が保証されていなければまったく動作できません．ソフト・リアルタイム・アプリケーションの場合は速い応答時間が必要ですが，応答時間が超過しても性能が緩やかに低下するだけです．RTOSが動作するシステムは一般に二次ストレージをもちません．

プラットフォーム・オペレーティング・システムは大型の非リアルタイム・アプリケーションを管理するためのメモリ管理ユニットを必要とし，多くは二次ストレージをもちます．Linuxオペレーティング・システムは典型的なプラットフォーム・オペレーティング・システムです．

この二つの種類のオペレーティング・システムは相互に排他的ではありません．メモリ管理ユニットを備えたARMコアを使用し，リアルタイム特性をもつオペレーティング・システムもあります．そして，各カテゴリ専用のプロセッサ・コア群が開発されています．

1.4.3　アプリケーション

オペレーティング・システムはアプリケーション，すなわち特定のタスクを処理する専用コードのスケジュールを決めます．アプリケーションは処理タスクを実行し，オペレーティング・システムは環境を制御します．組み込みシステムは一つのアプリケーションをアクティブにすることも，同時に複数のアプリケーションを実行することもできます．

ARMプロセッサはネットワーク機器，車載機器，モバイル機器，コンシューマ機器，大容量記憶装置，イメージング機器など，多数のマーケットで採用されています．これらのマーケットでも，ARMプロセッサは複数のアプリケーションに使用されています．

たとえば，ARMプロセッサはホーム・ゲートウェイ，高速インターネット通信用DSLモデム，IEEE802.11対応ワイヤレス通信などのネットワーク・アプリケーションに使用されています．モバイル機器市場は携帯電話のおかげでARMプロセッサにとって最大の応用分野となっています．ARMプロセッサはハードディスク・ドライブなどの大容量記憶装置，インクジェット・プリンタなどのイメージ

ング機器にも使用されています．これらはコスト意識の強い量産アプリケーションです．

逆に ARM プロセッサはトップクラスの高性能を必要とするアプリケーションには採用されていません[注1.5]．ARM は，そのような一般に生産量の少ない高コストのアプリケーション向けの設計に対してはそれほど注力してはいません．

1.5 まとめ

純粋な RISC プロセッサは高性能を重視していますが，ARM は，RISC の設計理念を一部変更し，優れたコード密度と低消費電力を目標としています．組み込みシステムはプロセッサ・コアとその周辺にあるキャッシュ，メモリ，ペリフェラルで構成されます．システムはアプリケーション・タスクを管理するオペレーティング・システムによって制御されます．

RISC プロセッサ設計のポイントは命令の簡素化による性能向上と，パイプラインによる命令処理速度の向上，コアの近くにデータを保存する大型のレジスタ・セットの装備，ロード・ストア・アーキテクチャの使用です．

ARM の設計理念には RISC に反する考え方も採用されています．

■ 一部の命令について可変サイクルでの実行を認め，消費電力，実装面積，コード・サイズを節約する．
■ 一部の命令の機能を拡大するためにバレル・シフタを追加する．
■ Thumb 16 ビット命令セットを使用してコード密度を高める．
■ 命令の条件実行によってコード密度と性能を改善する．
■ ディジタル信号処理などの機能を果たす拡張命令を追加する．

組み込みシステムには，ハードウェア・コンポーネントが含まれます．ARM プロセッサはチップに組み込まれています．プログラマはメモリ・マップ・レジスタ経由で**ペリフェラル**にアクセスします．コントローラと呼ばれる特別なタイプのペリフェラルもあります．組み込みシステムはコントローラを使用し，メモリ，割り込みなどの高レベルの機能のコンフィギュレーションを行います．**AMBA オンチップ・バス**はプロセッサとペリフェラルを接続します．

また，組み込みシステムにはソフトウェア・コンポーネントもあります．**初期化コード**はコンフィギュレーションによってハードウェアを既知の状態にします．コンフィギュレーションが終わると，**オペレーティング・システム**がロードされ，実行されます．オペレーティング・システムはハードウェア・リソースとインフラを使用するための共通のプログラミング環境を提供します．**デバイス・ドライバ**はペリフェラルへの標準インタフェースとなります．**アプリケーション**は組み込みシステムの特定タスク向け処理を実行します．

注 1.5：2007 年現在は，Cortex-A8 などの高性能プロセッサ・コアを発表するなど，トップ・クラスの性能を必要とするアプリケーションにも採用されている．