第6章

XScaleへのカーネル移植の実際

邑中 雅樹

第5章では，開発環境に関する一般的な注意事項を解説し，今回の移植ターゲットを選定し，最低限のセットアップを行いました．実際に移植を行うためには，もう少し準備が必要です．

まずは，XScale評価キット添付のGCCがもつ問題を解決し，補助ツールの導入を行います．その後，JSP1.4のソース・ツリーを用いた作業を始めます．

1．補助ツールの準備

GNUのツール群を標準開発環境に据えていることから想像できるとおり，TOPPERSカーネルはUNIX系OS上における開発を前提としています．たとえば，ビルド環境を構築するためのconfigureスクリプトはPerlで書かれており，`Makefile`には`rm`などのUNIX系ツールの導入を前提とした記述が見られます．

TOPPERS対応をうたうボードであれば，これらのツールも含まれていると思いますが，今回のようにTOPPERS未対応ボードの場合は，自前で調達することになります．

本章を執筆した2004年9月時点でのTOPPERSプロジェクトの公式見解では，Windows環境についてはCygwinの使用を推奨しています．最近のCygwinは独自インストーラの採用によって，数年前よりインストールしやすくなりました．しかし，それでも必要なパッケージを適切にインストールするには，UNIXのツールに関する知識が要ります．μClinuxなどの普及でUNIXツール類に精通している読者の方も多いと思いますが，UNIXツールの導入をTOPPERSカーネルの敷居の高さとして挙げる人が多いのもまた事実のようです．

今回はボードへの移植が本題なので，環境構築の細かい部分については深入りせず，バイナリ・ディストリビューションで解決することにします．

2004年9月の時点では，ボード添付ではなく，特定のCPUアーキテクチャに特化しておらず，ソフトウェアだけ入手できるTOPPERSに特化したパッケージとして，PizzaFactory2とeFOSiの二つがあります．eFOSiはアドバンスド・データ・コントロールズから配布されている環境です．完成度は高く，無償で配布されているのですが，Green Hills Software社（GHS）の開発環境を前提としています．GHSの開発環境は，気軽に試してみるというわけにはいかない価格です．

そこで今回は，PizzaFactory2の無償配布版であるOvenless Editionを使用します．PizzaFactory2については，**コラム1**を参照してください．

PizzaFactory2のインストールは非常に簡単です（**図1**）．一般的なWindowsアプリケーションと同じように，実行形式のインストーラを立ち上げてウィザードの指示に従うだけで，必要なツールがハード・ディスク上に展開されます．クロス・コンパイラ付きの有償版のようなプロダクトIDの入力も，無償版のOvenless Editionにはありません．

PizzaFactory2には，JSP1.4が含まれているので，あらためてJSPカーネルをダウンロードする必要はあ

図1　PizzaFactory2のインストーラ

第7章

XScaleへのデバイス・ドライバの移植とカーネル移植の完了

邑中 雅樹

本章では，sample1アプリケーションが動作するところまでを解説します．本章の内容を読み進めれば，**図1**のようにJSPカーネルが手元で動作することでしょう．

1. 目標設定

大ざっぱな目標を，ソース・コードと照らし合わせて設定しておきましょう．

第6章では，プログラム・カウンタが関数kernel_start()に到達するところまでを確認しました．本章では，kernel_start()から先を実装することが目標です．後で説明しますが，kernel_start()を抜けた先は，アプリケーション・プログラムの領域になります．もちろん，アプリケーション・プログラムはカーネルを呼び出すのですが，その大半はターゲットに依存しない部分なので，カーネル移植者が気にする必要はありません．

今回はJSPカーネルの移植が主題なので，なるべくPXA250に依存しない話題を中心に話を進めます．しかし，それでもPXA250に対する知識が必要な局面もあります．PXA250に関する文献を章末に示すので，必要であれば適宜参照して読み進めてください．特に参考文献(1)の第9章以降は今回の評価ボードを使った解説になっており，おおいに参考になります．

2. カーネルが動作するまで

① 幹となる関数kernel_start()

リスト1はstartup.cの一部です．startup.cの関数kernel_start()を見ればわかるとおり，この関数を最後まで実行した後，カーネルは動作を開始します．

第5章で述べましたが，TOPPERSカーネルでは，ターゲット依存部と共通部が分離されています．そのため，移植する際に，複数のファイルを参照しながら境界部分をつくろうような作業が多くなります．また，UNIX系OSやWin32にあるような抽象化されたデバイス・モデルはありません．また，慣れないうちは自分が何をしようとしていたのかを見失いがちです．そのような混乱を避けるため，その時々で「今の作業はkernel_start()のどの行から呼ばれているのか？」ということを意識しながら，読み進めてください．

どこに注目しているのかをわかりやすくするために，本章の見出しとkernel_start()内のソース・コードのコメントを一致させてあります．

② カーネル動作の開始

作業開始早々，最後の行を説明するのはちょっと気がひけるのですが，これ以前の処理を理解するために重要なので，あえて最後の行から説明します．

まず，iniflgにTRUEが代入されます．iniflgはカーネルが動作するときにTRUEと定義されています．ここで「カーネルが動作する」というのは，「タス

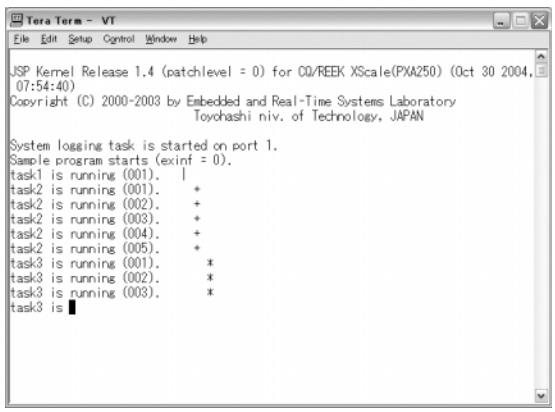

図1　TOPPERSカーネルが起動したようす

第8章

ARM7への移植

中野 俊和，平尾 智也，赤星 博輝

今回，μITRON 4.0規格に準拠した「TOPPERS/JSP 1.4.2」をDesign Wave Magazine 2006年3月号の付属ARM7ボードに移植しました．本章ではARM7にTOPPERS/JSPを移植する方法について解説します．

● TOPPERS/JSPとは？

TOPPERSプロジェクト[(3)]は，ITRON仕様の技術開発成果を出発点とし，組み込みシステム構築の基盤となる各種のソフトウェアを開発し，良質なオープン・ソース・ソフトウェアとして公開することで，組み込みシステム技術と産業の振興を図ることを目的としたプロジェクトです．2003年9月に設立された特定非営利活動法人（NPO法人）を中心に，名古屋大学の高田広章氏をリーダとして活動しています[注]．

今回，ARM7基板に移植したのはTOPPERS/JSPのバージョン1.4.2です．JSPとは，Just Standard Profileの略で，μITRON4.0仕様のスタンダード・プロファイル規定に従って実装されています．このJSPカーネルには，以下のような特徴があります．

- 読みやすく改造しやすいソース・コード
- ほかのターゲットCPUへのポーティング（移植）が容易な構造
- 高い実行性能と小さいRAM使用量
- Linux上およびWindows上でのシミュレーション環境
- 開発環境まで含めてフリー・ソフトウェアのみで構築可能

● OSの作成は難しいが，移植はそれほど難しくない

リアルタイムOSは，ソフトウェアとしては高度なプログラムと言ってよいと思います．理由としては，

1）割り込みや例外処理により複数のタスクを切り替えるため，動作が複雑
2）その割り込みを制御するためにアセンブラによるプログラミングが必須

という点が挙げられます．

まず，割り込みにおいてどのような処理が行われているかを**図1**で見てみます．最初，プログラムAを実行していて100番地の命令を実行しようとしたときに割り込みがかかったとします．そうすると，まずCPUの中にあるデータをメモリに書き出して保存します（そうしないと，別のプログラムの処理を開始したとき，これまでの結果が上書きされてしまう）．それで，CPUでは戻り番地を覚えておき，プログラムBの実行が終わったら，先ほどメモリに書き出したデータをCPUに戻し，プログラム・カウンタ（PC）をリターン・アドレスの内容に変更することで，プログラムAを再実行することができます．

このようなCPUの内部の情報にアクセスするためには，アセンブリ言語を使ったプログラミングを行う必要があります．C言語では，どのレジスタにどの値を入れるかを指定することができませんし，制御レジスタの内容を変更したり，割り込みを禁止したりといった記述を書けません．例えば，今回使用するARMプロセッサにおける内部の情報の書き出し・復帰には，以下のような記述を使っています．

```
stmfd sp!, {r4 - r11,lr}
  /* レジスタR4～R11，LRのスタックへの保存 */
ldmfd sp!, {r4 - r11,lr}
  /* スタックからレジスタR4～R11，LRの復帰 */
mrs   r0, cpsr
          /* ステータス・レジスタをR0に代入 */
msr   cpsr, r0
```

注：今回のARM7基板へのTOPPERS/JSPカーネルの移植に関して，名古屋大学の高田・冨山研究室の方々にご指導，ご協力いただきました．心より御礼申し上げます．

第9章

ゲームボーイアドバンス（ARM7）への移植事例

大西 秀一

任天堂[注1]のゲームボーイアドバンス（以下，GBAと略す）は大ヒットした携帯型ゲーム機で，お持ちの方も多いでしょう．GBAは組み込み開発プラットホームとしても扱いやすいマシンです．今回はオープン・ソースのμITRON仕様OSであるTOPPERS/JSPカーネルを移植して動作させてみます．

1. GBAへ移植することになった経緯

移植の前に，筆者がGBAを扱うようになった経緯から説明しておきます．2003年8月に会社の同僚からGBAのフラッシュ・カートリッジを渡されました．それがGBAとの付き合いの始まりでした．フラッシュ・カートリッジはその名のとおり，GBAのスロットに挿すことのできるフラッシュ・メモリのカートリッジです．

フラッシュ・カードリッジがある

→自作のプログラムが動かせるかも？

→TOPPERS/JSPカーネルが動くかも！

という単純な動機により，GBAのプログラミングを開始しました．くしくもTOPPERSプロジェクトがその年（2003年）の9月にNPO法人として発足するという時期に，筆者は開発環境を手に入れたわけです．TOPPERSプロジェクトの第1回通常総会（2003年9月12日）には展示スペースがあり，その隅のほうで展示したのがGBA用のTOPPERS/JSPカーネルです．このときに筆者に与えられた開発期間はわずか1週間でした．かなり無理のある開発でしたが，なんとか展示に間に合わせることができました．このことは「TOPPERS/JSPカーネルの移植は，条件さえそろえば1週間で可能である」という証明にもなりました．

それ以降，さまざまなGBA用のプログラムを開発しましたが，ベースとなっている技術はカーネル移植で得た知識であり，たいへん役に立っています．

本章は少し「お遊び」の色彩の濃い内容になるかと思います．まずGBAへの移植手順を解説しましょう．

注1：http://www.nintendo.co.jp/

コラム1　GBAのライセンス

GBAのプログラムを続けていくと，RAM実行ではメモリ領域が少なくて物足りなさを感じることがあります．そこで登場するのがフラッシュ・カートリッジです．このフラッシュ・カートリッジは通販サイトなどで入手可能ですが，任天堂のライセンス製品ではありません．したがって，フラッシュ・カートリッジの利用によるGBA本体の破損などについて，任天堂は補償してくれないというわけです．

さて，このフラッシュ・カートリッジですが，自作のプログラムを書き込んでも動作しないのです．なぜでしょう？　理由は，カートリッジ内に一種のライセンス・データが存在するためです．そのライセンスとはGBAの電源を入れたときに出てくる任天堂GBAのロゴです．カートリッジの特定の場所にそのロゴが存在しないとブートしません．

これを回避するにはロゴ・データを自分で書き込む必要がありますが，筆者はライセンス契約を結んでいないので書けません．どうしてもフラッシュ・カートリッジでという方にはPogoShellという逃げ道が存在します．以下にURLを記載しておくので，興味のある方は参照してください．

http://www.nightmode.org/pocket/

第10章

Blackfin DSPへの移植 ――ディスパッチ処理

中村 健真

Analog Devices社のDSP/RISCプロセッサであるBlackfinに，μITRON4.0仕様に準拠したリアルタイムOSのTOPPERS/JSPを移植しました．移植はTOPPERSプロジェクトの公開文書を参考にしながら，m68k向けの実装を手本とし，Blackfinのターゲット依存部を開発する形で行いました．

すでに存在するTOPPERS/JSP実装を，同じCPUを使った別システムに移植する方法の解説はよく見受けられますが，新しいCPUへの移植情報は，あまり見かけません[注1]．そこで，筆者が行った作業などを書き記すことにしました．

1. DSPにこそリアルタイムOSが必要

筆者がDSP関連の職に就いた1990年代初頭は，DSPでリアルタイムOSを走らせることは現実的とは言えませんでした．市場にDSP向けのリアルタイムOSはありましたが，DSP自身が単機能のアクセラレータのように使われることが多かったため，需要がそれほどありませんでした．また，使いたくても，性能の面で余裕がなくて使えなかったとも言えます．

しかし，最近のDSPは数百MHzで動作するものが多くなり，新しく設計されるDSPは高級言語による開発やリアルタイムOSの搭載を意識したものになっています．

性能の面で問題がないとなると，DSPでこそリアルタイムOSを使いたくなってきます．それにはいくつか理由があります（**図1**）．

● DSPのアプリケーションが高機能化

まず，DSPのアプリケーションが単機能でなくなったことが挙げられます．以前なら信号処理だけ行えばよかったのですが，高性能化するにつれ，汎用マイコンを搭載せずにDSPですべての処理を行いたいという要求が出てきました．たとえばスイッチの読み取りやLEDの制御といったユーザ・インターフェース周りの仕事，ファイル・システムの管理，プロトコル・スタックの実行などが挙げられます．

複数の異なる仕事を同時にこなすにはリアルタイムOSが最適です．汎用マイコンの世界では通常，リアルタイムOSを使ってアプリケーションを開発します．DSPでそれらの仕事もこなすのであれば，リアルタイムOSが必要になるでしょう．

● DMAの同期に使いたい

リアルタイムOSを使いたい二つ目の理由は，DMA（Direct Memory Access）との同期です．信号処理の効率を上げるには，プログラムの最適化もさることながら，データの流れの効率化が極めて重要です．いくら信号処理アルゴリズムが高速でも，バスやSDRAMのためにデータの読み書きが遅くなっては性能が出ないからです．そこで，これを解決するためにDSPで

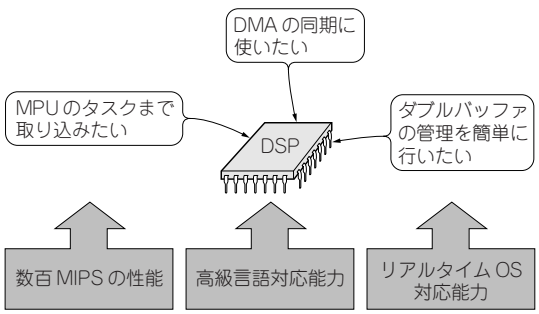

図1　DSPでリアルタイムOSを使いたい理由

注1：CPU依存部の理解や実装において，一番重要な文書はTOPPERS/JSPの配布アーカイブにあるdoc/config.txtである．この文書はCPU依存部の実装に必要なことをほとんど網羅している．プレーン・テキストが苦手な人は，プロジェクト外部でPDF化された文書を読むこともできる．