オープンソースのITRON仕様OS

TOPPERSで学ぶRTOS技術

第16回 Blackfin DSPへのTOPPERS/JSPの移植

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である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向けの実装を手本に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90年代初頭は，DSPでリアルタイムOS（以下RTOS）を走らせることは現実的とは言えませんでした．市場にはDSP向けのRTOSもありましたが，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，それほど需要がありませんでした．また，使いたいとしても，性能の面で余裕がなかったことも考えられます．

しかし，最近のDSPは数百MHzで動作するものが多くあります．また，新しく設計されるDSPは高級言語での開発やRTOSの搭載を意識したものになっています．

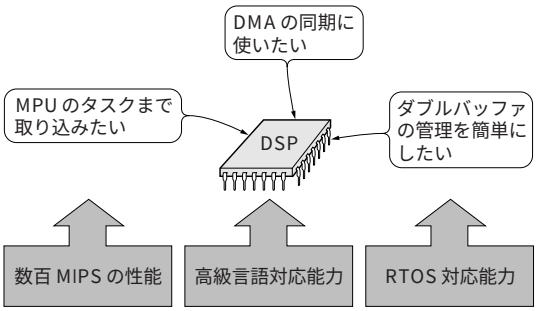

図1 DSPでRTOSを使いたい理由

性能の面で問題がないとなると，DSPでこそRTOS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・インターフェースまわりの仕事，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RTOSが最適です．汎用マイコンの世界では通常，RTOSを使ってアプリケーションを開発します．DSPでそれらの仕事もするのであれば，RTOSが必要になるでしょう．

● DMAの同期に使いたい

RTOSを使いたい二つ目の理由は，DMA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たので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ではDMAを多用しています．

DMAを使用することで，プロセッサ・コアはデータ転送から開放されます．ところが，今度はDMAによるデータ転送とプロセッサが実行するプログラムの間の同期が重要になってきます．DMAが終了するまでポーリングで待つのは何と言っても時間のむだです．そこで，DMAを待つ間は別のタスクを走らせて，DMAが終了したら割り込みによって待ち状態のタスクを起こすといった処理にしたくなります．こういった処理はRTOSの独壇場です．

● ダブルバッファの管理を簡単に

RTOSを使いたい三つ目の理由は，ダブルバッファの管理です．オーディオ信号やビデオ信号のような連続する信号を扱う場合，データの処理を行いながら，次のデータを連続的に入力

注1：CPU依存部の理解や実装において，一番重要な文章は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．プレイン・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．

しなければ，データの流れが途切れてしまうことになります．

ダブルバッファを使うことによって，入出力とデータの処理の両立を実現できます．しかし，そのためには連続的なDMAと二つのバッファを使ってお手玉のような制御を行わなければなりません．こういった処理も入出力ポートが一つだけなら何とかなるものの，二つ以上になると手に負えなくなります．

DMA転送，割り込み，二つのバッファの処理を効率よく行うには，RTOSが便利です．RTOSを使用することで，ダブルバッファの処理を専用のタスクの中に閉じ込めることができるため，処理タスクからは複雑な流れを見ずに済みます(**図2**)．また，すでに説明したようにDMAとの同期が簡単であるほか，バッファ管理タスクとデータ処理タスク間の通信といったことまで管理するRTOSもあります．

このように，DSPに汎用マイコンの処理を行わせたり，もともとDMAやダブルバッファを多用するといった理由から，DSPにこそRTOSが必要になるのです．

2. TOPPERS/JSPのBlackfinへの移植は初めて

TOPPERS/JSPは，TOPPERSプロジェクトが開発したオープンソースのμITRON4.0に準拠したRTOSです．TOPPERS/JSPはすべてのサービス・コールではなく，必須な一部のサービス・コール(スタンダード・プロファイル)にのみ対応することによってコンパクトになっています．たとえば，Blackfin向けの実装では，セマフォと割り込みハンドラだけを使う場合には全体で8Kバイト程度のオブジェクト・サイズに収まります．

また，実装が軽いことからOSのオーバヘッドも小さくなっています．たとえば，600MHzのADSP-BF533の場合，セマフォによるタスク切り替え時間は480ns，タスク切り替えのうち，実際のコンテキスト切り替えにかかる時間は125nsと高速です．

ところで，TOPPERS/JSPは多くのアーキテクチャに対応していますが，この中にBlackfinの名前はありません．プロジェクトが対応を謳(うた)っているのは，公式リリースに含まれているもののみで，これはプロジェクト会員の手によって保守されています．一方，Blackfin向けの実装は筆者が共同作業者の助けを得て独自に実装したものであり，プロジェクトとしてはサポート対象外となります[注2]．

TOPPERSプロジェクトからの成果物はJSPカーネルのほか，μITRON4.0仕様に完全準拠したFI4カーネル，組み込みTCP/IPプロトコル・スタックであるTINETをはじめとして，高信頼性カーネルや新しいASPカーネルなど広がりを見せています．

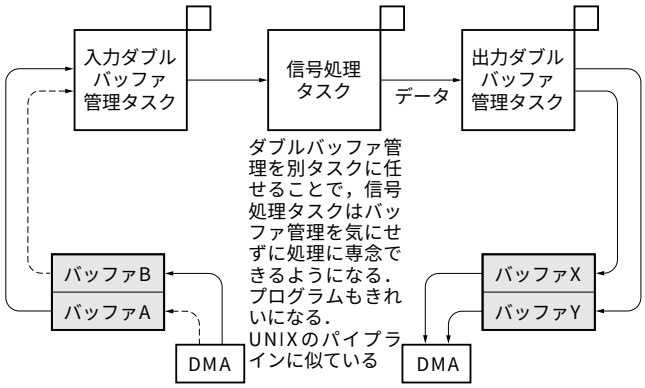

図2　タスクによるダブルバッファ処理

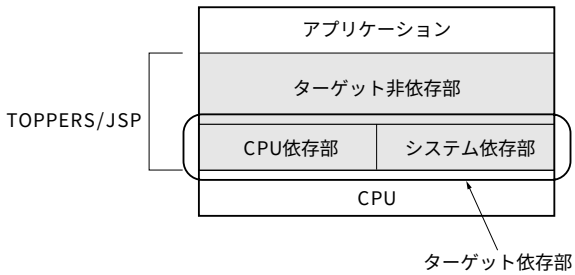

図3　TOPPERS/JSPの構造

3. カーネルの構造

さて，お待ちかねの内部構造に踏み込んでみましょう[注3]．TOPPERS/JSPは大きく分けて二つの部分に分かれます．一つはターゲット依存部で，もう一つはターゲット非依存部です．ターゲット依存部は，さらにCPU依存部とシステム依存部に分かれます(**図3**)．

● ターゲット非依存部

ターゲット非依存部は，その名のとおりCPUに依存しない部分であり，セマフォやデータ・キューなどの通信機構，タスク同期機構などのアプリケーションへのサービスのほとんどがこれに当たります．このターゲット非依存部はすべてC言語で書かれており，どのCPUに対しても再コンパイルを行うだけで利用できます．非常に移植性が高いのが特徴です(**図4**)．

もちろん，C言語を使えば移植性が自動的に高くなるなどというようなうまい話はありません．C言語には実装依存の部分があります．たとえば，型に割り当てるビット数などは処理系によって異なります．また，64ビット整数型なども，処理系によって実装していたり，していなかったりといろいろです．こういったことから，とくに組み込みの分野では，型サイズに気を付け

注2：厳密に言えば，公式リリース自身にもプロジェクトによる公式サポートはない．オープンソースの特徴といえる．
注3：宮城産業技術総合センターのWebサイトに役立つ情報がある．http://www.mit.pref.miyagi.jp/embedded/TOPPERS/