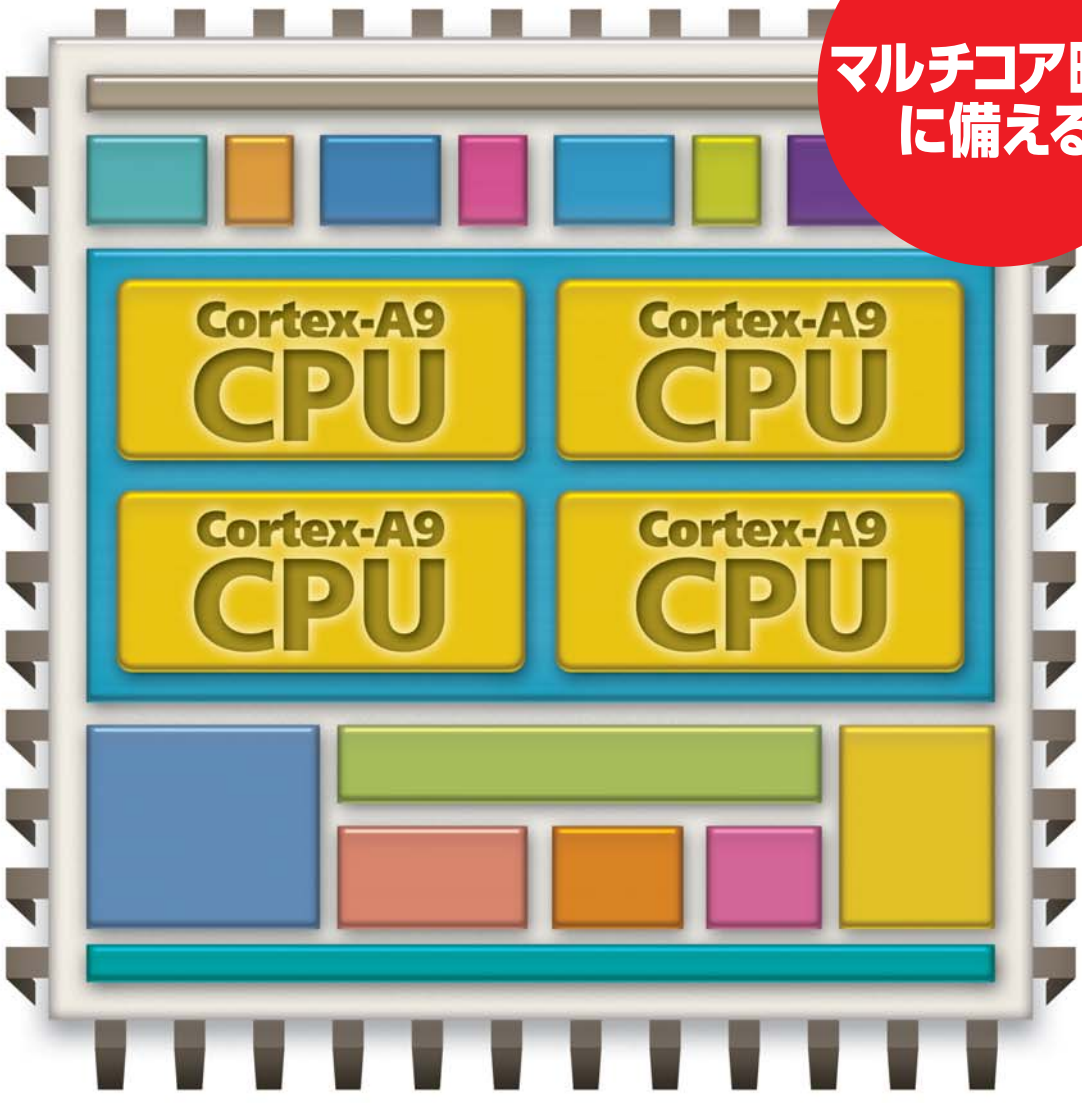
TECH I Embedded Software
Technology Interface
Vol.53

見本

並列＆高速処理のためのマイコン・プログラミング

ロボットもキビキビ動く！ ARMや並列プロセッサXMOSを例にして

インターフェース編集部●編



CQ出版社

並列処理の基本となるマルチタスク・プログラミングの基礎

五十嵐 仁

複数のタスクが同時に動作するマルチタスクOSは，複数の作業を同時に処理したり，リアルタイム性の要求される処理が容易に行えるなどのメリットがある．そのため，組み込み向けOSといえばマルチタスクOSといえるほど普及している．しかし，マルチタスクOSはシングル・タスクOSよりさまざまな要素を考慮しなければならない．資源の排他制御，優先順位，タスクの分割――これらの事項を把握して適切に設計する必要がある．そこで本章では，並列コンピューティングの基本であるマルチタスク・プログラミングについて学んでいく．

組み込みシステムの世界においても，マルチコア・プロセッサ対応の波が押し寄せてきています．その中でも，負荷分散型のSMP（Symmetric Multi-processing；対称型マルチプロセッシング）に注目が集まり，SMPに対応したCPUやOSが続々と登場しています．

しかし，単純にシングル・コア環境で開発してきたソフトウェアをSMP環境で動作させれば，性能が上がるわけではありません．それどころか，正常に動作しない可能性すらあるのです（**コラム1.1**を参照）．

マルチコア・プロセッサ環境では，複数の処理が完全に並列に実行されます．そのため，ソフトウェアを正常に動作させるには，適切な排他・同期制御を行う必要があります．とは言っても，マルチコア対応のために，新しい特別な技術が必要になるわけではありません．基本となるのは「マルチタスク・プログラミング」です．

そこで，本章ではマルチタスク・プログラミングの基礎について説明していきます．

1.1 マルチタスク・プログラミングとは

かつてのパソコン向けOSはシングル・タスクでした．同時に一つの処理しか行えず，複数の処理を行いたい場合は，一つの処理が終了したあとで，次の処理に取り掛かるしかありませんでした．

これに対してマルチタスクとは，1台のコンピュータ・システム（処理装置）で，同時に複数の処理（タスク）を並行して実行する機能のことです．例えば，1台のパソコンでCDの音楽を聴きながら，Webサイトを閲覧して，ワープロ・ソフトウェアでレポートを書くことができます（**図1.1**）．このたとえでは，音楽再生ソフトウェアとWebブラウザ，ワープロ・ソフトウェアを同時に実行させているので，マルチタスク環境を利用して実現していることになります．

近年のOSでは，パソコン用だけでなく，組み込み用のOSでもマルチタスクが標準となっています．複数の処理を同時に行いたいという要求が高まっているためです．

それでは次に，マルチタスク・プログラミングがどのような原理で複数の処理を同時に実行しているかについて説明します．

● マルチタスクを実現する方法

一番簡単にマルチタスクを実現する方法は，「タスクの数だけプロセッサを用意する」ことです．マルチタスクを実現したいコンピュータ・システムに，同時に実行するタ

図1.1 パソコンの環境では，マルチタスク・プログラミングは当たり前

組み込みOSで使われるマルチタスク技術の歴史

藤倉 俊幸

組み込み機器を動かす社会のインフラである組み込みOS．昔から存在するように錯覚するが，実はマイコンが普及した1980年代以降に登場したため，わずか30年の歴史しかない．はじめはモニタ・プログラム的な小さいOSが使われていたが，現在ではLinuxのような大規模なOSまで組み込みOSとして使われている．
本章では，組み込みOSを支えるマルチタスク関連技術を中心に，各OSの技術について見ていく．

2.1 組み込みOSの誕生はマイクロプロセッサとともに

電気で動く最初のコンピュータが作られたのは1940年代である．1960年代から1970年代になるとエアコンの効いた部屋に鎮座するコンピュータを限られた人が操作するようになった．そして，米国Intel社の8080や米国Motorola社の6800などのマイクロプロセッサが1974年に登場して，組み込みシステムの歴史が始まった．このとき，同時に組み込みOSの歴史も始まった．

1980年代に入ると，組み込みOSの種類も増えてマルチタスク環境を提供するための技術基盤が確立された．リアルタイムOS（RTOS）と組み込みOSがほぼ同義語の時代でもある．

この時期のOSは，ほとんどアセンブラで書かれており非常に性能が良かった．しかし，1980年代後半からはCPUの種類が増えて，各OSは新しいCPUに早く対応することが市場から求められるようになり，移植性が重要視され，C言語による書き換えが行われた．この結果，カーネル内の割り込み禁止区間が増えて，基本的な性能は落ちた．しかし一方では，プライオリティ・シーリングなどの複雑なアルゴリズムを実装できるようになった．

● 技術以外の要素が組み込みOSの命運を分けた

自社開発のコンパイラを使用しているOSベンダもあったが，移植性の点ではGCCのようなオープンなコンパイラを利用した方が有利であった．このときの対応により，その後の各OSの運命が決まったような気がする．基本的性能，あるいは自慢の自社技術に固執したOSは，1990年代後半には消えていった．OSの歴史は，技術だけでは語れない．

1990年代に入ると組み込み製品も多様化し，それに対応して新しいタイプの組み込みOSが利用されるようになった．また，組み込みOSはどれも同じように成熟してOSそのものの差がなくなり，OS本体よりも利用できるミドルウェアや開発環境の方が重要視されるようになった．

アプリケーションの幅が広がって他社製品と通信する必要性が出てくると，特定のプロトコルを実装した共通のミドルウェアを利用した方が効率が良いのである．

また，オープン・ソース化やフリーのOSの出現により，従来のロイヤリティ・ベースのビジネス・モデルは成立しなくなり，OSベンダの数は減少していった．ビジネスとしてのOSも，この時代に終わったのかもしれない．1980年代に確立されたマルチタスクやリアルタイム同期の技術は，アプリケーション中心に再構成されて利用された．

● OSはコンポーネントの一つになった

2000年代は多様化がさらに進み，肥大化した情報系組み込み機器向けのものや先祖返りのようなフットプリントの小さな制御用のものなどに分化が進んだ．また，OSはすでにプラットホームを構成するコンポーネントの一つであり，マルチコア環境ではこれらの分化した複数のOSを組み合わせて使用することも行われるようになった．

1980年代にOSとともに登場した，タスクあるいはスレッドという概念は，オブジェクト指向の考え方によってモデル化されたアーキテクチャに取り込まれるようになった．さらに，Androidではタスクはアクティブ化したアク

マルチコア，マルチプロセッサのハードウェア

中森 章

マルチプロセッサを実現するためには，単純にプロセッサを複数搭載すればよいというわけではない．複数プロセッサをバスに接続するだけでは，バスの奪い合いになり，思ったように性能が上がらない．また，共有メモリの排他制御なども問題になる．これらはプロセッサのハードウェアを工夫することで解決することが可能だ．ここでは，マルチコア，マルチプロセッサから，ハイパースレッドまで，ハードウェア面の工夫について解説する．

3.1 性能向上にはマルチコア化が必然

昨今，半導体メーカがマイクロプロセッサやマイクロコントローラを作るために使用するCPUコアの種類が，各企業ごとに淘汰されてきた感があります．また，ARM社やMIPS Technologies社といったCPUコアを提供するIPベンダからCPUコアを買ってきて，そのまま組み込むというケースも増えてきました．

しかし，1980年代には，各半導体メーカがCPUのさまざまなアーキテクチャを開発し，それを業界標準とするべく躍起になっていました．そのような時代においても，CPU単体の性能向上はいつか限界に達すると予測されており，最終的には「マルチプロセッサが主流になっていくのは必然」と考えられていました．

冗談として，一つのプロセッサ内にマルチプロセッサ・システムを構築する「1チップ・マルチプロセッサ」という言葉が話題になったものです．つまり，「マルチプロセッサなのに一つのプロセッサ？」というのが言葉遊びのように矛盾する感覚で興味深かったのです．

しかし，この冗談が今や現実味を帯びてきました．一つのチップに集積する場合は，「マルチプロセッサ」という単語はやはり違和感があるのか，「マルチコア」，「メニイ・コア」という言葉が使われています．マルチコアへの方向性を決定的にしたのはIntel社ですが，最初は性能向上の手段というよりも，性能を維持しつつ消費電力を下げる手段として提案されました．動作周波数が数GHzを超えるCPUコアの開発では，製造プロセスを微細化する必要があり，トランジスタのスレッショルド電圧を低く抑えた結果，リーク電力（スタティック電力）が巨大になって使いものにならなくなるという事情があります．また，当然，動作周波数が上がれば，動作電力（ダイナミック電力）も周波数に比例して上昇していくので，その影響も受けます．

これと同じような傾向は，組み込み制御の世界にも起きています．CPUコアを現実的な消費電力（2W ～ 3W程度）で動作させるには，動作周波数を400MHz ～ 600MHz程度に制限せざるを得ません．そこで，それ以上の性能向上を狙うにはマルチコア化が必然となってきます．

とはいえ，CPUコアを二つ持ったからといって，2倍の性能が得られるとは限りません．各CPU間がお互いに協調して動作する仕組みを作り込むことが必要です．それにも増して，OSの手助けが必須となります．

本章では，マルチコアのハードウェアに求められる最低限の基幹技術を俯瞰（ふかん）していきたいと思います．なお，マルチコアといっても，その実体はマルチプロセッサ・システムを1チップに集積しただけで，基幹技術はマルチプロセッサと同じです．以降では，典型的なマルチプロセッサの基礎知識を紹介していきます．

3.2 マルチプロセッサの構成

● マルチプロセッサの目的は全体的な処理時間の短縮

マルチプロセッサとは，プロセッサが複数存在するシステムのことです．ここで，各プロセッサにどのように処理させるかによって，論点は微妙に異なります．

マルチプロセッサでプログラムを作成するためのアセンブリ命令

中森 章

マルチプロセッサ・システムを構築する際には，資源の排他制御をどのような手法で実現するのかが問題となる．近代的なプロセッサでは，排他制御を1命令で実現できるアセンブリ命令が用意されている．本章では，これらのアセンブリ命令について解説する．

本章では，マルチプロセッサ間で同期を取るために用意されたアセンブリ言語の命令について解説します．また，命令だけにとどまらず，その使い方を知ることにより，資源の排他制御を実現する方法などについても解説します．

4.1 マルチプロセッサ・システムにおける排他制御

マルチプロセッサ・システムを構築する上では，いかにして排他制御を行うかが重要になります．純粋にソフトウェアだけで排他制御を行うことも可能ですが，ハードウェアによる支援があると，高速な排他制御（1クロック～数クロックで実行）が可能です．そして，現在のマルチプロセッサ，マルチコアCPUのほとんどがハードウェア的な排他制御機構を持ち，そのためのアセンブリ命令を備えています．

● 同期のためのソフトウェア処理

マルチプロセッサ・システムにおいては，共有バスがメイン・メモリ（共有メモリ）にアクセスする唯一のアクセス手段です．メモリの排他制御は，バスの使用権を獲得したプロセッサがメモリ・アクセスを独占し，ほかを閉め出せばよいのです．この処理は，不可分（アトミック）なスワップ命令（データ交換命令）があれば簡単に実現できます．

不可分とは，あるプロセスがメモリにアクセスしている間に，ほかのプロセッサからのメモリ・アクセスがないことをいいます．あるいは，ほかのプロセスがアクセスしようとしてもそれが禁止されている状態をいいます．実際には，（バス）ロックという端子を活性化して外部回路に通知し，ハードウェア的にほかのプロセッサのアクセスを禁止します．

話が横にそれましたが，ソフトウェア的には，排他制御のために，各共有資源に対して，1対1に対応するロック変数を用意します．つまり，ハード・ディスクには変数A，ディスプレイには変数B，プリンタには変数Cといった具合です．ここで，ロック変数が0の場合はそれに対応する資源が未使用，0でない場合はそれに対する資源が使用中であることを示すものとします．

プロセスは早い者勝ちで，0以外の値をロック変数に書き込むことで，その資源を使用する権利を得ます．例えば，プリンタを使用したい場合，変数Cが0なら誰も使用していないので使用可能，0以外なら使用中なので待つ（またはあきらめる）といった処理になります．

初期のLinuxでは，共有資源ごとにロック変数を持つのではなく，ただ一つのロック変数で排他制御を実現していたそうです．つまり，ロック変数にアクセスできたプロセスは，すべての共有資源にアクセスできるようになるのです．こういう単純な制御でも，システム的には破たんしない程度の実行性能が得られる（昔は得られていた？）のでしょうね．

● テスト・アンド・セットによる排他制御

排他制御を実現する最も単純な処理が，「テスト・アンド・セット」です．これは，ロック変数を読み込み，その値が0ならそこに1（または0以外の値）を書き込みます．ロック変数が0でなければ，書き込みは行いません．通常，ロック変数の前の値がリードの結果として返されるので，それが0ならロックが成功したことを示します．0でなければロックは失敗したことになります（つまり，ほかのプロセスが使用中であることを意味する）．

図4.1に，テスト・アンド・セットを利用した相互排除

マルチコア環境における タスク動作の検証方法

藤倉 俊幸

マルチタスクOSで動作するタスクは，割り込みなどの外的要因によってタスクの実行順序が変化し，それがバグの要因となることがある．タスクの実行順序を把握するためには，実行順序を全て洗い出し，ステート・マシンとして記述するという方法がある．従来は手動でこの作業を行っていたが，モデル検査ツールLTSAを使用することにより，ステート・マシンを自動出力できる．本章では，形式検証を使ったマルチプロセッサ環境で動作するタスクの検証について解説する．

モデル検査ツールを使用したタスク設計の検証については，参考文献(5.1)の 第22章以降で説明したことがあります．本章では，その内容を簡単に解説し，マルチプロセッサ環境やマルチコア環境にどのように適用させるのかなどについて，モデル検査にあまりなじみのない人でも分かるように説明します．

5.1 モデル検査とは

● 数学的な手法をソフトウェア開発に取り入れる

形式手法(Formal Method)と呼ばれる手法があります．形式手法は，「数学を積極的に使ってソフトウェアを開発していこう！」という心意気に対して付けられた名前です．手法と言うよりは，アプローチあるいは手法群と言った方が良いかもしれません．歴史は古く，コンピュータが使われ始めた頃から研究されていました．研究はされていましたが，「難しすぎる」，「実際に使うにはコストが掛かりすぎる」などの理由から，ずっと日の目を見ない存在でした．しかし，この形式手法と呼ばれる一群の手法の中に，「モデル検査」と呼ばれる分野があり，最近になって開発現場で利用され始めています．

モデル検査では，検査対象になる要求仕様や設計仕様，実際のコードなどから，動きや操作に関する情報を抽出してモデルを作ります．このモデルは，全ての可能な動きのパターンを生成します．全ての可能な動きのパターンが生成できたら，次に，一つ一つの動きを検査します．例えば，イベントの順番や特定のイベントが発生しているかなどです．

モデルと検査でモデル検査になるわけですが，要求や設計を対象とする場合には，要求分析を行う人や設計を行う人が全ての可能な動きを把握できることには重要な意味があります．つまり，検査なしでも有効な使い道があります．考えていることの全体を把握できることが重要なのです．

そこで，ここでは検査にはあまり立ち入らずに，まず「モデルとはどんなものか」，「全ての可能な動きとは何か」について詳しく説明します．

● LTSAを使ってボールを取り出す問題を解く

非常に簡単なサンプルで考えてみましょう．高校などで習う順列組み合わせ問題です．

> つぼの中にAというボールが3個とBというボールが2個入っているとする．ここから3個のボールを取り出すときの取り方は何通りあるか？

これは，順列組み合わせ問題の公式に当てはめれば，7通りであることが分かります．しかし，具体的にどのような取り出し方が可能なのかは，公式を見ても分かりません．これをモデル検査ツールであるLTSA(Labelled Transition System Analyser)を使って解くと，具体的な7通りの動作パターンを作り出すことができます．

図5.1に，実際にLTSAのモデルと生成した動作パターンを示します．動作パターンは，実際はつぼからAかBを取り出す順で示すべきですが，ここではその取り出し順を生成するステート・マシンで示します．モデルと言っているのは，通常このステート・マシンのことです．A=…，

100CPUを用いた大規模マルチプロセッシング・システムの構築

岡澤 幸一

8CPU程度のマルチプロセッサであれば，従来のマルチタスク技術の延長でシステムを構築できた．しかし，それより大規模な，例えば100CPUなどのシステムになると，通常の構成とは異なるアプローチが必要になる．本章では，マイクロカーネル・アーキテクチャを採用したOS「QNX」を例に，分散環境とは何かについて解説する．また，大規模分散環境の構築事例も紹介する．

CPUのクロック速度の向上が限界に達しつつあるにもかかわらず，ユーザからは高性能化や低消費電力化の要求が高まっています．それに対する処方せんとして，ハードウェア・ベンダにより提供されている技術がマルチコアであると筆者は考えます．

マルチコア化したシステムに対して，従来のアプリケーションがそのまま動作することが要求されます．また，マルチコアにより動作するCPUコンテキストが複数あるため，同期制御や排他制御という問題が浮上してきます．マルチコア化とは，ソフトウェアの課題と言っても過言ではありません．

本章では，マルチコアを始めとした分散システムの分類について解説します．ハードウェア・システムは，それぞれのCPUが独立したメモリとI/O資源を持ち，高速通信路で接続された並列プロセッサを用いた構成とします．ターゲットとなるプロセッサの数は，数十個～数百個を想定しています．

6.1 並列と分散

そもそも「並列」と「分散」という用語は同じでしょうか．普段は何とはなしに使っているかもしれません．まず，背景などを整理してみます．

簡単に，二つの言葉が含まれている用語を挙げてみます．

「並列」コンピュータ，「並列」処理，超「並列」，…

「分散」ファイル・システム，「分散」システム，…

これらの例から言うと，走行するハードウェア側から，「複数の環境で同時に走行すること」に対して語られる言葉が「並列」であるといえます．一方，ソフトウェアやシステムの立場から，「実際のソフトウェア・コードを分散させる方法や，その結果できあがるシステム」に対して「分散」システムと名付けられます．

つまり「並列」化したハードウェア・システム環境で，「分散」化したシステムを実現すること，これが本章のテーマになります．

6.2 ハードウェアの視点――「並列」

ソフトウェアの観点から疑似「並列」を実現するOSについて考えてみる必要がありますが，ここではまず，実際に並列性を実現するためのハードウェア・システムの技法から見ていきます．

CPUを複数使い，要求される処理を分割してシステムとして構築する手法は，要求される処理の分割のタイプによって分類されます．

● 専用サブプロセッサ（ジョブ実行型）

専用サブプロセッサは，データ処理の視点が中心となるアーキテクチャといえます．画像処理や信号処理などを行

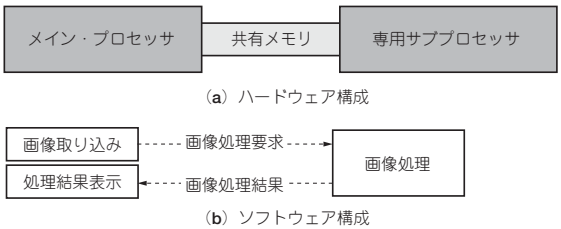


図6.1 専用サブプロセッサの構成

組み込みマルチコアにおけるSMPとAMPの比較

柴下 哲

マルチコア環境には，すべてのCPUに公平に処理（タスク）を割り当てるSMP方式と，特定のCPUに特定の処理を割り当てるAMP方式がある．一般的に，パソコンではSMPが採用されることが多いが，組み込み機器ではAMP方式が多い．本章ではAMP方式を採用した時の動作をツールを用いて解析する．

7.1 組み込みマルチコアの現状

高機能化と低消費電力化，および世界中を視野に入れたグローバルな製品展開に対応する必要性が年々高まっています．最初の二つの要求はハードウェアでも対応可能ですが，最後の要求を満たすためには，それぞれの国や地域ごとに異なる要望に応える必要があるため，ソフトウェアによる対応が必須です．これらを満たす解として，かなり前からマルチコア化が提唱されてきており，実際にさまざまな分野で使われるようになりました．

マルチコアに関する興味深いデータとして，米国VDC Research社の調査結果があります．**図7.1**は，2010年における製品に搭載されているCPUの数を表していますが，シングル・コアのケースが63.6%と大半を占めています．ここで，マルチプロセッサとは複数のCPUチップを，マルチコアとは1チップに複数のコアを搭載しているということです．図を見れば分かるように，マルチコアを使用している比率は9.6%とまだ少数派です．これに対し**図7.2**は，2010年から2年以内の次期製品に搭載予定のCPUの数を表しています．特徴的なのは，シングル・コアの製品が30.1%と半減していることです．マルチプロセッサの比率はほとんど変わらないので，減少した30%は，マルチコアとマルチコアを含む複数のCPUチップへ移動しています．そのためマルチコアを使う予定が合計で約40%となっているのです．

このように，マルチコアの使用が組み込み製品でもポピュラになってきており，本章ではその問題点と注意点を事例を基に説明します．

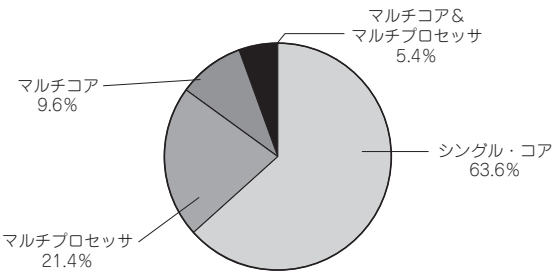


図7.1 2010年時点の製品に搭載されているCPUの数
出展：VDC Research's 2010 Embedded Engineering Survey

7.2 既にマルチスレッド化しているのでOKなのか？

シングル・コアでも，組み込みLinuxやVxWorksのようなリッチなRTOSを使っている場合は，既にプログラムをマルチスレッド化しているケースも多いことでしょう．その場合，OSがマルチコアに対応すれば，それほど大した作業をしなくても十分速くなると思っている方もいますが，

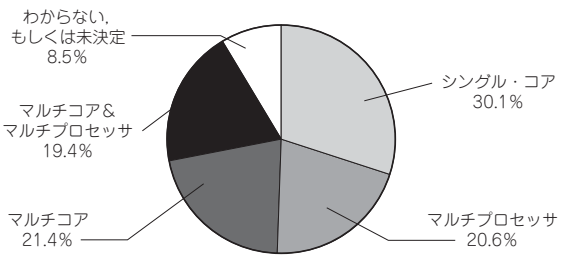


図7.2 2010年から2年以内の次期製品に搭載が予定されているCPUの数

トランスピュータの流れを汲む並列プロセッサXMOSの概要

小山尾 登

現在，並列動作をするプログラムを書くためにはマルチタスクOSを使い，既存のC言語がそのまま使われている．1990年ごろに注目を集めたトランスピュータは，プログラミング言語にOccamを使用することで言語レベルから並列動作をサポートするだけでなく，動作を数学的に検証することが可能であった．そのトランスピュータが形を変え，XMOSとして入手できるようになった．本章では，このXMOSについて解説する．

8.1 トランスピュータの復活

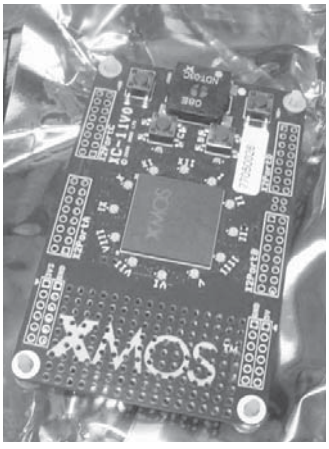
昔々，Inmos社のトランスピュータ（Transputer）というプロセッサがあった．トランスピュータはOccamという言語でプログラミングされた．Occamには，モデル検査で有名になったCSP（Communication Sequential Processes）という数学的背景があり，形式手法の走りでもあった．

当時，「bit」という雑誌があり，トランスピュータの紹介記事が掲載されていた．もちろんInterface誌にも関連記事が紹介されていた．1990年ごろのことである．斬新なコンセプトを持ち超並列マシンなどと呼ばれることもあったが，いつの間にか姿を消してしまった[8]．Inmos社は英国の会社だったが，英国の政権交代の際に政策が変わって売却されてしまったという噂も流れた．また，当時は開発環境などが高価であったのも普及しなかった理由かもしれない．

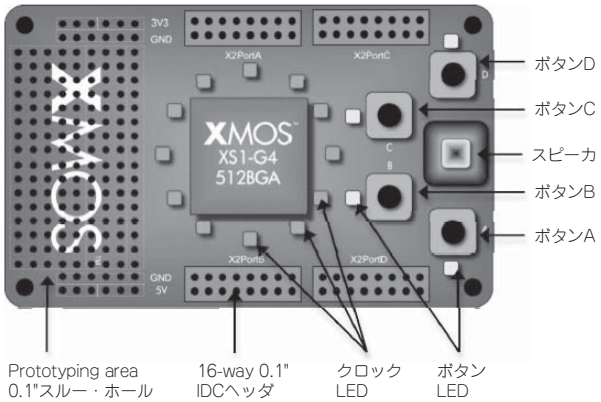
それがXMOS[1]という名前で復活した．シングル・コアのXS1-L1プロセッサは$7.5，コンパイラなどを含めた開発環境は無償である．ちなみに，XMOSは会社の名前であり，XS1-xxはプロセッサの名前，XC-nはボードの名前になっている．例外は，XS1-Gという弁当箱型の開発キットである．

XMOSを使用すれば，FPGA並みの処理をソフトウェアで実現できる．さらに，RTOSなしで並列処理も実現できる．割り込みハンドラという概念もなく，イベント・ドリブンで直接ソフトウェアが動く…などといきなり言われてもイメージできないぐらいに，今でも斬新さは衰えていない．

また，10ns～20nsのオーダでアプリケーション・ソフトウェアが外部イベントを認識できるので，十数μsオーダのコンテキスト・スイッチや割り込み遅延がいつ起こる



◀ 写真8.1 XMOSを搭載したXC-1評価ボードの外観



▶ 図8.1 XC-1評価ボードに搭載されている部品とデバイス

第9章　RTOSなしに並列処理が可能で，しかも検証もできる

並列プロセッサXMOSの検証環境PAT

小山尾 登

第8章では，トランスピュータの流れを汲む並列プロセッサXMOSの概要について解説した．XMOSの特徴の一つに，プログラミング言語XCの言語仕様が検証に使うモデルと近いので，検証が容易であることが挙げられる．そこで本章では，XCに近い検証環境としてPATを取り上げ，検証を行ってみる．

9.1 XMOSの検証環境を検討する

第8章では，XMOSの特徴について説明した．その中の一つは，検証モデルとソース・コードが非常に近い関係にあり，モデルをほぼそのまま実行できることだと述べた．その例として，定番の「哲学者の食事」問題をプログラミングして対応するLTSA（Labelled Transition Analyser）モデルと比較してみた．

しかし，LTSAとXMOSのプログラミング言語であるXCとの間には，まだギャップがある．それは，変数の扱いとチャネルによる通信である．そこで，もう少しXCに近い検証環境としてPAT（Process Analysis ToolKit）[1]を使ってみようと思う．元祖トランスピュータ系の検証ツールとしてFDR2[2]があるが，アカデミックな用途以外は有償である．その点，PATは今のところフリーである．

それから，XMOSの特徴を活かす設計方法についても説明が必要だと思う．現在の組み込みソフトウェアのほとんどは，共有変数に強く依存した構造になっているが，この考え方を改めないとXMOSの特徴を活かした設計はできない．すなわち，並列処理の設計をどうすれば行えるかという問題である．

これは，XMOSに限らずRTOS下のタスク間の関係やRTOSを使用しない場合の割り込みハンドラとmain関数との関係にも当てはまる．

● 設計したら，まず検証したい

設計したら，すぐにソース・コードにするのではなく，まず検証することが並列性の高いアプリケーションを作る場合には必要である（**図9.1**）．それは，デッドロックやライブロックは，ソース・コードになってしまった設計をいくら見ていてもわからないからである．すなわち，設計環境と検証環境は，表裏一体の関係になければならない．

開発環境はIDE（Integrated Development Environment）あるいは統合開発環境を指し，開発全体をサポートするような印象を受けるが，実体は編集・コンパイル・デバッグの狭いサイクルを効率良く回すツール・セットであり，たいそうな名前が付いている割には小さいことしかやっていない．

現状のIDEは，統合実装支援環境と呼ぶのが正しいだろ

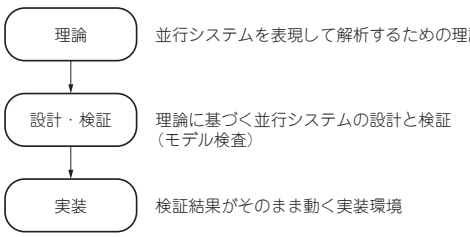


図9.1　開発の流れ

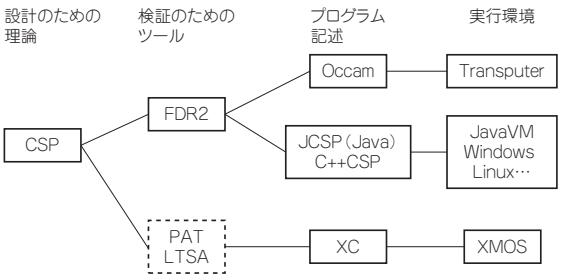


図9.2　設計・検証環境の組み合わせ

PUPPY倒立振子を並列プロセッサXMOSで実現する

小山尾 登，CSP研究会

CSP（Communicating Sequential Processes）によるプログラミングが可能な並列プロセッサXMOSで倒立振子を実現する．
SH-2とリアルタイムOS（RTOS）を用いて実現した既存の倒立振子をXMOSの並列に動作するハードウェア・スレッドへ移植する．

10.1 倒立振子PUPPY

● XMOSで倒立振子を実現する

2009年に開催されたETロボコン[注1]の題材は，倒立振子だった．PUPPY[注2]やBeauto Balancer[注3]のような市販の倒立振子もある．Embedded Technology 2009（ET2009）[注4]では，XMOSブースで北斗電子の倒立振子ロボットPUPPYのCPUをXMOSに載せ替えてファルクウェア[注5]がデモンストレーション（注5のリンク先に動画あり）を行っていた．

倒立振子を実現するには，単なるライン・トレースとは違って制御工学やリアルタイム制御の知識が必要になる．本章では，ET2009でデモンストレーションしていたものとは別に，CSP研究会に参加している北海道職業能力開発大学校の中原 博史氏が開発したXMOS版PUPPYをベースにして，XMOSプログラミングの特徴を説明する．

本章で紹介するソフトウェアは，中原氏が北斗電子の厚意によりH8プロセッサ用のPUPPYプログラムを利用して開発したものである．H8プロセッサ用とXMOSプログラムとの対応がわかりやすいように，基本的な関数名をそのまま使用している．また，PUPPYについてはCPUをH8からSH-2に変更した解説記事[1]がある．公開されているSH-2版のソース・コードをXMOS版のリファレンスとする．

● オリジナルのPUPPY

ETロボコン2009で使用されたMathWorks[2]によるものはRTOSとサーボ・モータを使ったもので，道具立てが大げさな割に倒立制御モジュールはブラック・ボックスになっている．XMOSの特徴は，ハードに近いFPGAで行うような制御もC言語に近いXC言語で記述できる点にある．同じ倒立ロボットでも，ETロボコンはON/OFF制御のモデリングを楽しむようなイベントであり，コンセプトが異なる．そのため，倒立振子の制御は単純なON/OFF制御ではなく，制御工学技術者と組み込みソフトウェア技術者の境界領域が混ざったようなところがある．

PUPPYは，RTOSなしのDCモータ制御である．Beauto BalancerはPUPPYと同じような製品だが，H8に最適化されていてCPUだけをXMOSに変更するのが大変

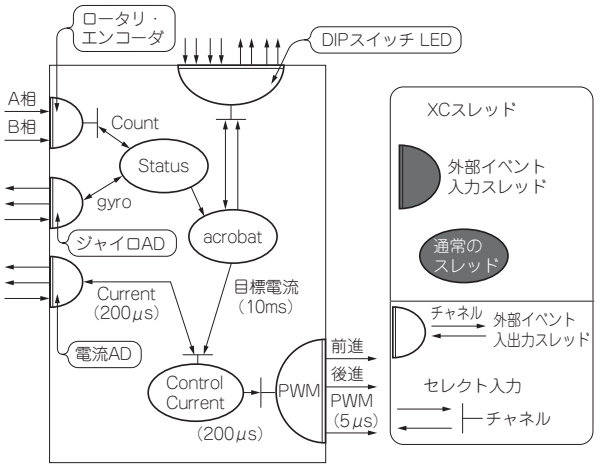


図10.1 XMOS版PUPPYアーキテクチャ

注1：http://www.etrobo.jp/ETROBO2009/
注2：http://www.hokutodenshi.co.jp/7/PUPPY.htm
注3：http://www.vstone.co.jp/top/products/robot/beauto/bindex.html
注4：http://www.jasa.or.jp/et/
注5：http://fulcware.jp/

XMOSによる フル・カラー LED Cube 実験

小山尾 登

本章では，並列処理プロセッサXMOSでLED Cubeを制御する実験を行ってみる．通常のCPUでは，このような実時間・並列処理はマルチタスクOSなどで実現する．これに対してXMOSは，プロセッサ自身が並列処理機能を持ち，ハードウェア並列処理が可能である．最近のXMOSは，タイミング・アナライザやレポート機能が充実しているので，これらについても見ていこう．

筆者が所属するCSP研究会[注1]の第6回で，北海道職業能力開発大学校の中原 博史氏からフル・カラー LED Cube（**写真11.1**）をXMOSで制御する発表があった．本章では，この発表を元に，フル・カラー LED Cubeのドライバ製作を検討してみる．また，その過程でXMOSの並列性と高速性を利用した制御方法およびハードウェア制約をソフトウェアでカバーする方法について説明する．

11.1 XMOSと フル・カラー LED Cube

● XMOSは並列処理をソフトウェアで記述できる

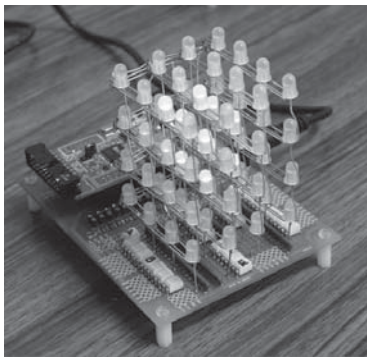
XMOSは，リアルタイムOS（RTOS）なしにFPGA並みの時間領域の並列処理をソフトウェアで記述できるチップである(1)．

ここでは，XMOSの中からシングル・コアのXS-L1(3)というチップを使用する．これは，8スレッドまで同時に利用できる．XMOSのプログラムには割り込みハンドラという概念は存在せず，スレッドが直接イベント・ドリブンで動作するという特徴がある．あるいは逆に，「並列動作するWait可能な割り込みハンドラだけでできている」といってもよいかもしれない．

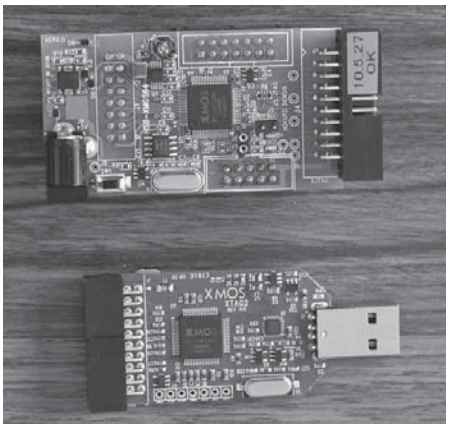
C言語を並列化拡張したXCと呼ばれる言語によってプログラミングを行う．無償の統合開発環境が提供されていてダウンロードすれば，ソフトウェア開発も可能になる(4)．

● フル・カラー LED CubeとJTAGアダプタ

使用するハードウェアは，北海道職業能力開発大学校と北斗電子が開発したXMOS学習キット（XMOS汎用ボードと学習ボードから構成される）を使用している．学習



◀写真11.1
フル・カラー LED Cube
赤，緑，青の3原色LED（フル・カラー LED）を4×4×4の立体格子状に配置・接続する．



▶写真11.2
XMOS汎用ボード（上）とXTAG2（下）

注1：形式手法を含めたCSP（Communicating Sequential Processes）モデルを普及させるために設立された研究会．
http://www.csp-consortium.org/index.html

CQ出版社

見本

このPDFは，CQ出版社発売の「並列 & 高速処理のためのマイコン・プログラミング」の一部見本です.

内容・購入方法などにつきましては以下のホームページをご覧下さい.
内容　　http://shop.cqpub.co.jp/hanbai/books/49/49811.htm
購入方法　　http://www.cqpub.co.jp/order.htm

TECH I Embedded Software
Technology Interface