

組込み向けチップマルチプロセッサ上の複数 OS 実行基盤

菅井 尚人† 近藤 弘郁† 落合 真一†

†三菱電機(株) 情報技術総合研究所

‡(株)ルネサステクノロジ システムコア技術統括部

高性能と低消費電力を両立させるチップマルチプロセッサの適用が拡大している。組込み向けのチップマルチプロセッサでは、SMP 構成のみでなくプロセッサごとに機能を分担させた利用形態も想定され、チップ上のそれぞれのプロセッサを使用して機能に適した複数の OS を動作させる必要がある。このような用途に対応し、チップマルチプロセッサ上で複数の OS を同時に実行するための SW 基盤を提案する。これによりプロセッサ間連携機能の実装を OS と分離し、複数 OS 実行環境の構築を容易にすることを狙う。本稿では、チップマルチプロセッサ上にこの SW 基盤を実装して各種の OS を動作させた結果を示し、本方式の有効性を確認する。

A Software Platform for Multiple OS Execution on Embedded Chip-Multiprocessor

Naoto SUGAI† Hiroyuki KONDO† Shinichi OCHIAI†

†Mitsubishi Electric Corp., Information Technology R&D Center

‡Renesas Technology Corp., System Core Technology Div.

Recently single-chip multiprocessors have been widely used, because they can archive both high performance and low power consumption. For embedded systems, not only SMP but also asymmetric type approach is important on utilization of multiprocessors. We propose a software platform to execute multiple OS on each processor in a chip suitable for such systems. In this paper, we describe the design and implementation of the software platform. We also show evaluation results of effectiveness on this software platform.

1. はじめに

高性能と低消費電力を両立できることから、チップマルチプロセッサの適用が拡大している。サーバ用途などでは、複数の同一プロセッサ構成による SMP の技術は既にある程度確立されており、これらの技術は基本的にはチップマルチプロセッサに対しても適用可能である。

一方、組込み向けのチップマルチプロセッサでは、SMP 構成によるスループットの向上のみでなく、機器の制御と情報処理のようにプロセッサごとに機能を分担させた利用形態も想定される。この場合、各機能に適した OS が同時に協調して動作する必要がある。

本稿では、組込み向けチップマルチプロセッサのそれぞれのプロセッサを使用して、複数 OS を同時に実行するための SW 基盤について記述する。

2. 目的

これまで筆者らは、チップマルチプロセッサ上でリアルタイム OS と汎用 OS を同時に動作させるマルチ OS 環境の開発を行ってきた。[1][2][3] ここでは、プロセッサ間の連携にかかわる機能は各 OS 内に実装していた。

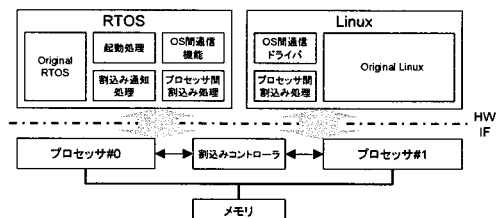


図 1 マルチ OS 環境

この方式では、新たな OS を使用する場合には、プロセッサ間の連携機能をその OS に組み込む必要がある。また、プロセッサ間の連携機能は、プロセッサの

接続形態や、プロセッサ間での割り込み方式などの HW 差異の影響を受ける。このため、同一の OS であっても新たな HW 構成に対しては OS 内の機能の改修が必要となる。

この課題を解決するために、プロセッサ間の連携や協調にかかわる機能を OS に提供する SW プラットフォーム(以下 SWPF)を設けて、HW の差異を吸収することを考える。各 OS は SWPF で提供される機能を利用することで、個々の HW に対応した移植の必要をなくすることができる。

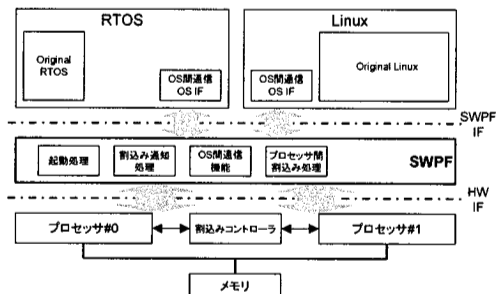


図 3 連携機能の OS からの分離

これにより、複数のプロセッサ上で複数 OS が協調して動作する環境の構築が容易となり、SW の開発期間短縮、コスト削減につながるものと考えられる。

3. 構成と機能

3.1 構成

本 SWPF が対象とするチップマルチプロセッサ HW として以下を前提とする。

- 複数のプロセッサコア
- 全てのプロセッサからアクセス可能なメモリ
- プロセッサ間の割り込み

プロセッサコアは、同一種だけでなく異種のプロセッサコアを含むヘテロジニアスな構成への対応も考慮する。

SWPF は、各プロセッサ上で動作する OS 毎にモジュールを持ち、そのプロセッサ上の OS に機能を提供する。これにより、ヘテロジニアスな HW 攻勢への対応を可能とする。また、1つの OS が複数のプロセッサを使用する SMP-OS の場合は、対応する SWPF のモジュールも複数のプロセッサ上で動作する。

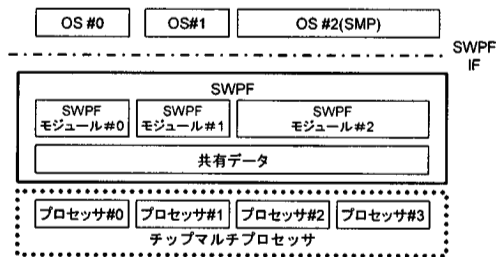


図 2 SWPF の SW 構成

3.2 機能概要

SWPF は、プロセッサ間の連携や協調にかかわる機能を提供するものであり、それ以外に関しては OS がプロセッサ上で直接動作する。SWPF が OS に提供する機能を以下に示す。

- 起動: 複数のプロセッサにおける OS の起動と、その起動順序、起動タイミングの制御
- 通信・同期: 複数のプロセッサ上で動作する OS 間での通信および同期機能
- 割り込み管理: IO など外部からの割り込みのプロセッサに対する配分

この他に必要な機能としては、メモリや IO などの資源の配分・調停が考えられるが、現状の SWPF では対応していない。これらの資源は、競合が発生しないようあらかじめ OS 間で分配されていることを想定している。

3.3 起動処理

起動処理は、複数のプロセッサにおける OS の起動を制御する。プロセッサと OS の対応、起動順序およびタイミングは、SWPF の生成時のコンフィギュレーションとして指定する。

SWPF は、自身の動作に必要な HW 初期化処理を行った後、指定されたプロセッサで、指定されたエントリポイントからの実行を開始する。

SWPF は、OS のブートロード処理は行わない。従って、ブートロード処理が必要な場合は、ここでのエントリポイントの指定を OS 自身ではなく、ブートローダのエントリポイントとする必要がある。

起動タイミングの制御は、コンフィギュレーション中で待ちを行うシーケンス ID を指定し、すでに起動している OS から対応するシーケンス ID を通知することで行う。

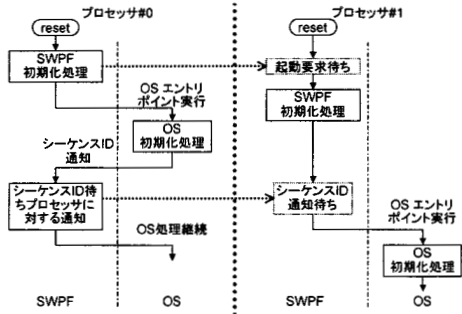


図 4 起動処理

リセット解除後のプロセッサの状態はターゲット HW に依存するが、以下の 2 つの場合が考えられる。

- a) 1 つのプロセッサのみが動作を開始し、他のプロセッサは停止状態に置かれる。動作を開始したプロセッサの指示により、他のプロセッサが動作を開始する。
- b) すべてのプロセッサが同時に動作を開始する。

SWPF では、すべてのプロセッサが同時に動作を開始する場合でも、SWPF の起動処理で同期を取り、指定されたプロセッサからの起動要求を受けてから、処理を継続する。

3.4 OS 間通信

OS 間での通信および同期を行う以下の機能を提供する。

- パイプ(単方向バイトストリーム)による通信
- セマフォによる同期

(1) パイプ

パイプは、単方向のバイトストリームとして OS 間で 1 対 1 の通信を行う。SWPF 上には、システム生成時に決定された個数 N のパイプが存在し、それらは 0 から $N-1$ までのパイプ ID を持つ。

パイプに書き込まれたデータは、SWPF の共有データ領域内のパイプバッファを介して、相手 OS により読み出される。

SWPF のサービスコールは、すべてノンブロッキング動作である。このため、OS に対するイベント通知はコールバックによる方式とする。

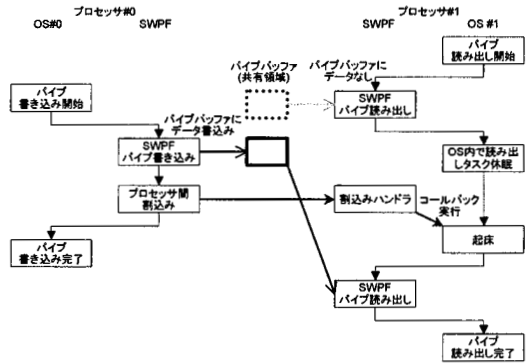


図 5 パイプによる OS 間通信

(2) セマフォ

セマフォは OS 間で同期をとる機能である。SWPF 上には、システム生成時に決定された個数 N のバイナリセマフォが存在し、それらは 0 から $N-1$ の ID を持つ。

セマフォの管理データは共有領域内に置かれ、各プロセッサ上の SWPF モジュールにより操作される。

3.5 割込み処理

複数 OS が動作する場合、IO など外部からの割込みは、その割込みを使用している OS が動作するプロセッサにのみ通知される必要がある。また、HW 上の制約により複数の IO がプロセッサに対する 1 つの割込み要求を共有している場合もある。

これらに対応するため、SWPF による割込み処理として、あるプロセッサが受けた割込みを他のプロセッサに通知する機能を持つ。また、SWPF の OS 間通信で使用するプロセッサ間割込みは、OS は介在せずに SWPF 自身で処理する。

割込みの要因とそれを処理する OS が動作するプロセッサを対応付ける割込み管理テーブルを共有領域に置く。SWPF は割込みを受けるとテーブルを参照し、

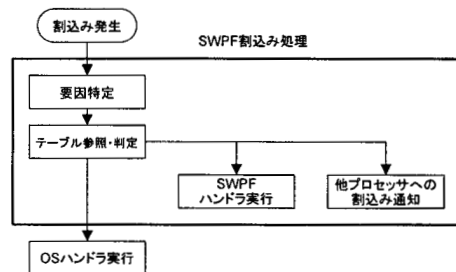


図 6 割込み処理

自プロセッサ上の OS が処理する割り込みかを判定し、そうであれば、OS の割り込みハンドラを呼び出す。自プロセッサ上の OS が処理する割り込みでなく、また SWPF 自身が処理する割り込みでもなければ、その割り込みを処理すべき OS が動作しているプロセッサに対し、プロセッサ間割り込みにより通知する。

4. 試作

ターゲット HW として、M32R コアを 2 個内蔵した M32700 プロセッサ⁴搭載のボードコンピュータ (ルネサステクノロジ製 M3T-M32700UT) を使用し、SWPF の試作を行った。

4.1 SWPF の実装

(1) メモリマップ

ターゲット HW はメモリとして、8MB フラッシュメモリ、512KB SRAM、32MB SDRAM を持っている。これらを、SWPF モジュールおよび 2 つの OS の領域として下図のように分割して使用した。

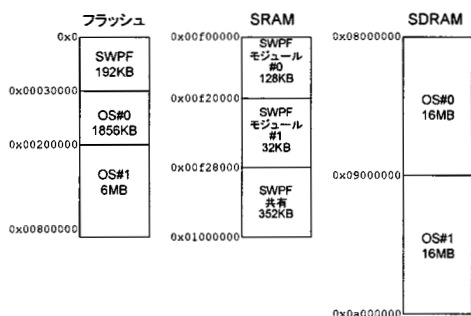


図 7 メモリマップ

(2) 起動処理

試作ターゲット HW の起動シーケンスでは、リセット後にプロセッサ#0 のみが起動し、プロセッサ#1 は停止状態に置かれる。プロセッサ#1 は、プロセッサ#0 からのプロセッサ間割り込みにより起動する。

今回の試作における実装の概略を以下に示す。

a) RESET 起動

プロセッサ#0 がフラッシュメモリ上のリセットベクタより実行を開始する。フラッシュメモリ上の SWPF モジュールを SRAM にコピーし、SRAM 上の SWPF モジュール#0 エントリポイントに実行を移す。

b) HW 初期化処理

HW 初期化処理は SWPF モジュール#0 のみで実行され、プロセッサで共有される HW の初期化を行う。処理内容は、クロック設定、バスコントローラ設定、SDRAM コントローラ設定などである。

c) SWPF 初期化

SWPF 内部のデータ初期化を行う。初期化処理は以下の 3 種類に分けて行う。

- 最初に起動したプロセッサのみが実行するもの
- 全プロセッサで共通に実行もの
- プロセッサ毎に異なる処理を実行するもの

d) プロセッサ#1 起動

プロセッサ#1 起動のためのプロセッサ間割り込みを発行する。プロセッサ#1 は SRAM 上の SWPF モジュール#1 のエントリポイントから実行を開始し、c) の SWPF 初期化を行う。

e) OS 起動

フラッシュメモリ上の各 OS 領域の先頭に実行を移す。

(3) OS 間通信

パイプは、プロセッサ間の共通データとしてパイプ管理データとパイプバッファを持つ。現実装では 1 つのパイプ当たり 8KB のバッファを割り当てている。セマフォでは、セマフォ管理データをプロセッサ間の共通データとして持つ。

パイプ、セマフォのいずれも、SWPF 内の共通処理としてイベント機構を使用しており、このイベント機構がプロセッサ間割り込みを使用してプロセッサ間での通知を行っている。

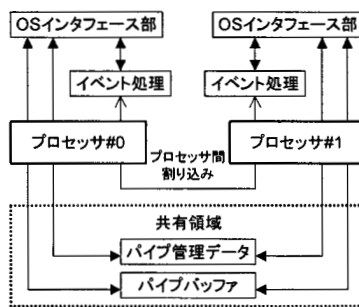


図 8 パイプ実装の概要

(4) 割り込み処理

割り込み処理は、OS 側の変更を最小化するため、フック方式での実装を行った。

OS がプロセッサに対して割り込みベクタアドレスの設定を行う際に、SWPF へのサービスコールでその値を通知する。SWPF では、SWPF が管理するベクタのアドレスをプロセッサに設定する。

SWPF の割り込みハンドラで OS のハンドラを呼び出す場合には、OS から通知されたベクタアドレスを使用して OS 側ハンドラのアドレスを決定する。

4.2 SWPF 上で動作する OS

本 SWPF 上で以下の 3 種類の OS を動作させた。

- Linux(2.6.18)
- TKernel(1.01)
- TOPPERS/JSP カーネル(1.4.2)

原理的には任意の組み合わせによる動作が可能であるが、以下の組み合わせでの動作を確認している。

表 1 SWPF 上 OS の組み合わせ

プロセッサ#0	プロセッサ#1
Linux	Linux
TKernel	TKernel
TKernel	Linux
TOPPERS	TOPPERS
TOPPERS	Linux

SWPF 上で動作させるための OS の変更点は次の 3 点である。

- 初期化処理：プロセッサ共有の HW については SWPF により初期化が行われる。OS がこの初期化処理を行っている場合、削除する。
- OS 間通信 API：SWPF の OS 間通信機能を利用した API として、Emblinx のハイブリッド OS 仕様例に基づく OS 間通信 API を実装している。FIFO 方式については SWPF のパイプ機能を使用した実装を行っている。また、共有メモリブロック方式については、ロック・アンロック操作を SWPF のセマフォを使用して実装している。
- 割り込み：OS が使用する割り込み要因の SWPF の割り込み管理テーブルへの登録処理を追加する。また、割り込みベクタアドレスの SWPF への通知処理を追加する。

5. 評価

本 SW 基盤の評価を以下の観点から行う。

- プロセッサ間連携機能を OS と分離したことによる SW 量の低減
- SWPF 上で動作する OS への影響
 - 実行性能
 - SWPF 対応のための変更

5.1 SW 量の低減

TKernel と Linux の OS 間通信(Emblinx ハイブリッド仕様 FIFO 方式)のソースコードについて、以前開発したマルチ OS 環境の場合と、今回の SWPF 使用の場合とで比較した。結果を下表に示す。

表 2 OS 間通信機能ソースコードライン数

	OS 個別 実装		SWPF	
	TKernel	Linux	TKernel	Linux
OS	1174	1285	464	569
SWPF				739

この結果からは、TKernel の場合ソースコードライン数の 61%、Linux の場合は 56%が SWPF として実装されている。この部分については OS によらずに同一のコードが使用できることから、対応 OS の種類が増えるほど、SW 量削減の効果が高まる。

また、SWPF を使用した場合、OS 側のコードは HW に依存しないものとなる。したがって、プロセッサ間割り込み等の HW が変更された場合でも SWPF のみを対応させればよく、OS 側を修正する必要はなくなる。

5.2 実行性能

実行性能の測定として TOPPERS の割り込み応答性能の測定を行った。測定条件は以下の 4 つの場合である。

- (1) TOPPERS のみが SWPF を使用せずに動作
- (2) SWPF 上で TOPPERS のみが動作
- (3) SWPF 上で TOPPERS と Linux が同時に動作
 - a) Linux が Idle 状態
 - b) Linux が NFS でファイルコピーを実行

測定は TOPPERS のタイマ割り込みを利用して行った。タイマ割り込みハンドラでタイマカウンタの値を読み出し、割り込み発生時のカウンタ値との差をとること

で、ハンドラ実行までの遅延時間を求めている。タイム割込み周期は1ms、タイマの分解能は40nsである。

表 3 TOPPERS 割込み応答時間(単位 μ s)

	最悪値	平均値
(1) TOPPERS 単体動作	0.80	0.52
(2) SWPF 上で TOPPERS のみ動作	1.40	0.96
(3-a) TOPPERS+Linux (Idle)	7.80	0.97
(3-b) TOPPRER+Linux (ファイル操作)	7.88	1.23

SWPF による割込み処理時のフックの影響が(1)と(2)の差であり、最悪値で 0.6μ s となっている。(3-a) および(3-b)ではさらに遅延の増大がみられるが、これは複数 OS 動作時のメモリアクセス競合によるものと考えられる。

(1)の TOPPERS 単体動作時と、(2)の SWPF 上での動作について、割込み発生からハンドラ実行までの実行命令数とその内容を比較した。

表 4 割込み処理の命令数比較

	(1) TOPPERS 単体動作	(2) SWPF 上で TOPPERS 動作
メモリ読み出し	1	7
メモリ書込み	4	9
分岐	3	12
その他演算等	8	34
計	16	62

命令数として約4倍になっており、メモリアクセスについては11命令増加している。これらは、SWPFで実装している割込みフックの構造上やむを得ないものと考えているが、さらに最適化を行うなどして、SWPFによるオーバーヘッド低減を図っていく必要がある。

5.3 SWPF 対応の OS 変更

SWPF 対応のための OS 変更は、4.2 節で記述したうちの OS 間通信 API の追加を除いた部分である。

TKernel, TOPPERS, Linux のそれぞれの変更ソースコードライン数を以下に示す。

表 5 OS 変更ソースコードライン数

OS	変更ライン数
TKernel	135
TOPPERS	144
Linux	191

変更箇所はすべて HW 依存部分であり、OS のソースコード全体に対して十分少ない。5.1 節で示した効果と比較しても、問題のない変更量と考えられる。

6. おわりに

本稿では、チップマルチプロセッサ上で複数の OS を同時に実行するための SW 基盤を提案した。そして、チップマルチプロセッサ上にこの SW 基盤を実装して各種の OS を動作させた。この結果、プロセッサ間連携の機能を OS と分離して実装することにより、OS 個別に実装した場合よりも SW 量が削減されたことを確認した。

今後の課題としては、SWPF による共有資源の調停・管理機能の実現、SWPF 割込み処理でのオーバーヘッド低減等があげられる。また、より多数のプロセッサを持つ HW への対応を行い、SWPF 上での SMP-OS の動作にも取り組んでいく予定である。

参考文献

- [1] 遠藤ほか、「シングルチップマルチプロセッサ上のハイブリッド OS 環境の実現 -システムアーキテクチャー」, 情報処理学会第 66 回全国大会 2D-5, 2004 年 3 月.
- [2] 菅井ほか、「シングルチップマルチプロセッサ上のハイブリッド OS 環境の実現 -OS 間インタフェースの実装」, 情報処理学会第 66 回全国大会 2D-6, 2004 年 3 月.
- [3] 遠藤ほか、「マルチコア上の異種 OS 間通信機能の設計と評価」, 情報処理学会第 68 回全国大会 5A-4, 2006 年 3 月.
- [4] Satoshi KANEKO, et al., "600MHz Single-Chip Multiprocessor with 4.8GB/s Internal Shared Pipelined Bus and 512kB Internal Memory", ISSCC2003
- [5] 日本エンベデッド・リナックス・コンソーシアム, 「Linux における RTOS とのハイブリッド構成に関する仕様(第 1 版)」, 2002 年 8 月.