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 -

菅井 尚人† 遠藤 幸典† 山口 義一† 近藤 弘郁‡
三菱電機(株)† (株)ルネサス テクノロジ‡

1. はじめに

携帯電話やカーナビ、デジタル AV 機器等の高機能化が急速に進んでいるが、これらの機器では、GUI やマルチメディア処理などの情報処理機能とリアルタイム性の両立が重要な課題となっている。この課題に対し、Linux などの汎用 OS が持つ豊富なソフトウェア資産の活用と、リアルタイム OS によるリアルタイム処理を同時に実現することができるマルチ OS 化は有効なソリューションの 1 つと考えられる。

今回、チップ内に 2 つの M32R CPU コアを搭載するシングルチップマルチプロセッサ[1] を採用した H/W プラットフォーム上で、μITRON 仕様のリアルタイム OS と Linux を同時に動作させる新しいハイブリッド OS 環境を実現した。

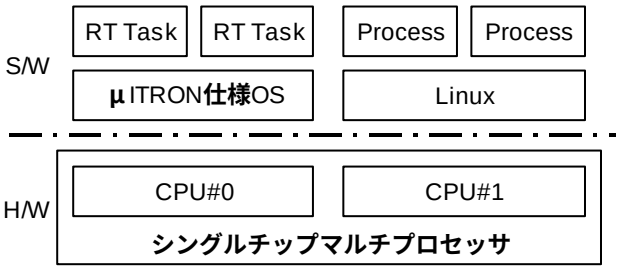

図 1 本システムの概要

本稿では、システムアーキテクチャ[2] に基づき、ハイブリッド OS 環境実現のための OS 間でのインタフェースの実装について記述する。

2. 実装方針

本方式では、μITRON 仕様 OS(以下、RTOS)と Linux がそれぞれ個別の CPU を使用して動作する。このため、OS 相互のやりとりを最小にし、それぞれの CPU 上の OS ができるだけ独立して動作することが望ましい。この点を念頭に置き実装を行った。

本方式の実現に際して検討すべき項目は、以下の 4 点である。

(1) 使用ハードウェア資源の分割
(2) OS 起動処理
(3) 割込み処理
(4) OS 間通信機能

ハードウェア資源の分割に関しては、それぞれの OS が使用する資源を下表のように定義した。H/W 構成上、共用せざるを得ないものを除き、どちらか一方の OS のみが使用するよう分割している。

表 1 H/W 資源の分割

H/W 資源	使用 OS
メモリコントローラ 割込みコントローラ	RTOS・Linux で共有
SDRAM、 FlashROM	アドレスにより分割
タイマ 1、パラレル IO	RTOS
タイマ 2、Ethernet	Linux

以下では、OS 起動処理、割込み処理、OS 間通信機能の実装について記述する。

3. OS 起動処理

今回使用したシングルチップマルチプロセッサの起動シーケンスは次の通りである。

(1) リセット解除により CPU#0 が実行を開始する。この時点では CPU#1 は停止したままである。
(2) CPU#0 が CPU#1 に対し、プロセッサ間割込みを発行する。
(3) プロセッサ間割込みを受信した CPU#1 は外部割込みベクタから実行を開始する。

このシーケンスに従い、CPU#0 で RTOS を、CPU#1 で Linux を起動する処理を実装した。

まず CPU#0 が実行を開始すると双方の CPU が共有する H/W の初期化処理を行う。続いて、RTOS の初期化を行い、それが完了すると、CPU#1 に対するプロセッサ間割込みを発行する。

CPU#1 が実行を開始する時点では CPU#0 により共有する H/W の初期化は完了している。従って、CPU#1 は通常の Linux のブートロード処理と同様に FlashROM からカーネルイメージを SDRAM 上にロードし、Linux カーネルの実行を開始する。

Linux の初期化完了時に、CPU#0 の RTOS に対してプロセッサ間割込みによる通知を行う。これにより、RTOS 側では、一定時間内に CPU#1 の Linux からの通知がない場合、異常処理などの

A hybrid OS environment on Single-Chip Multi- Processor - Implementation of interface between OS -
† Naoto SUGAI, Yukinori ENDO, Yoshikazu YAMAGUCHI Mitsubishi Electric Corp.
‡ Hiroyuki KONDO Renesas Technology Corp.

対処を行うことが可能になっている。

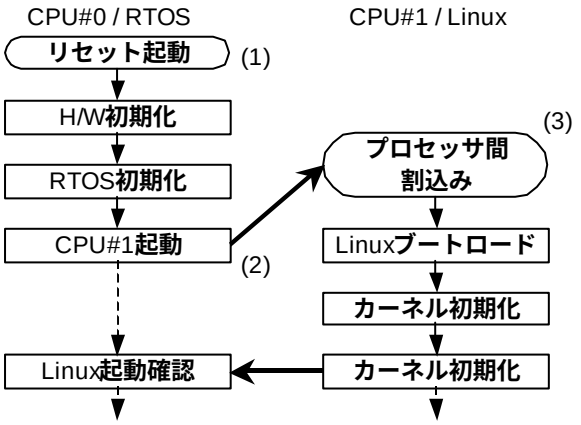

図 2 OS 起動処理

4. 割込み処理

H/W 資源の分割で示した通り、本アーキテクチャにおいては、IO デバイスを OS 間で共有することはなく、各デバイスに対応する割込みもそれぞれの OS を実行する CPU に割り振り、処理を行う。しかし、今回使用した割り込みコントローラでは個別の割込み毎に特定の CPU のみが受理するよう指定することはできない。

本実装では、RTOS におけるリアルタイム性の確保のため、RTOS が動作する CPU#0 で全ての割込みを受け付け、Linux が処理すべき割込みについては、プロセッサ間割込みにより CPU#1 に通知する方式を取った。

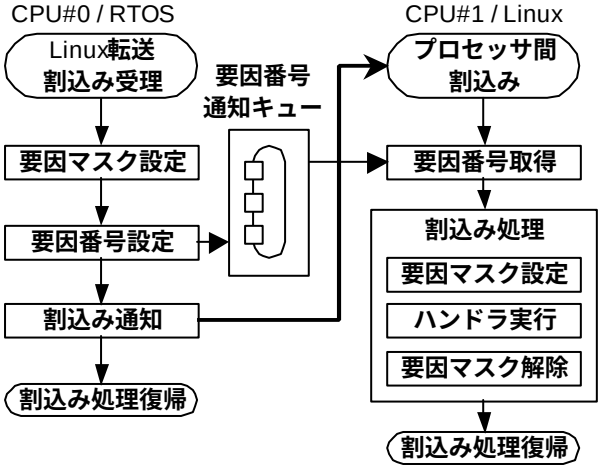

図 3 割込み処理

CPU#0 が、CPU#1 で処理すべき割込みを受理した場合、CPU#0 において対応する割込み要因のマスクを行う。これにより、CPU#0 は実際の割り込み処理の完了を待たず、CPU#1 への割込み通知後に割込み処理から復帰することができる。この場合、その後に別の要因による割込みを受理する可能性があるため、割込み転送時には複数の割込み要因をキューイング可能な機構を実装している。これにより、CPU 間での動作の独立性を高めている。

一方、CPU#1 では CPU#0 からのプロセッサ間割込みを受けると、キューの先頭の割込み要因を取り出して対応する処理を行う。CPU#1 での割込み処理が完了し、要因のマスクを解除した時点で、その割込みを CPU#0 が受理可能となる。

5. OS 間通信機能

OS 間通信としては、Emblix による Linux とリアルタイム OS との通信 API 仕様 [3] に従い、FIFO 方式と共有メモリブロック方式の 2 つの通信方式を実装した。

Emblix による仕様は、本来、1 つの CPU 上で動作するハイブリッド構成を想定したものである。しかし、アプリケーションプログラムから見た場合、RTOS と Linux との通信という点では本方式と同様である。また、1CPU 上でのハイブリッド構成とのアプリケーションプログラムの移行性の点からも、この仕様を踏襲している。

本実装においては、SDRAM 上の共有メモリ領域と、プロセッサ間割込みを使用した実装を行った。双方の OS からアクセスする箇所については、マルチプロセッサでの動作となるため、プロセッサの同期命令を使用した排他制御を行っている。

6. おわりに

本稿で述べた実装により、シングルチップマルチプロセッサ上で μITRON 仕様のリアルタイム OS と Linux がそれぞれ個別の CPU を使用し、可能な限り独立して動作するハイブリッド OS 環境を実現できたものと考える。

今後、割込み転送処理の性能への影響や、OS 間通信性能などの評価を行っていく予定である。

参考文献

[1] Satoshi Kaneko, et al.: “A 600MHz Single-Chip Multiprocessor with 4.8GB/s Internal Shared Pipelined Bus and 512kB Internal Memory”, Proceedings of 2003 Intern. Solid-State Circuits Conf., 14.5

[2] 遠藤、菅井、山口、近藤「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-」、情報処理学会第 66 回全国大会、2004 年 3 月

[3] 「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)」、日本エンベデッド・リナックス・コンソーシアム(Emblix) ハイブリッドアーキテクチャワーキンググループ活動報告書、2002 年 8 月