Vol.2013-EMB-30 No.6
2013/9/17

マルチコア向け組込みリアルタイムシステムの省電力機構

大橋　孝輔[1]　本田　晋也[1]　舘　伸幸[2]　高田　広章[1]

概要：近年，組込みシステムの高性能化に対応するためにプロセッサのマルチコア化が進んでいる．しかし，これに伴う消費電力の増加により，組込みシステムにおける省電力化が課題となっている．汎用システムでは既に省電力化のためのハードウェア機構やアルゴリズム等が提案されており，特にソフトウェアによる省電力機構に関しては，汎用システム向けの OS（汎用 OS）の省電力機能として既にサポートされているものが複数存在する．一方で，組込みリアルタイムシステムで使用される RTOS において，これらの機構をサポートしているものはない．組込みリアルタイムシステムでは汎用システムと比べ，リアルタイム性等の要件が存在し，汎用 OS で使用される機構をそのまま適用することはできない．本研究では，マルチコア向け組込みリアルタイムシステムに適用可能な，RTOS がサポートする省電力機構の仕様を提案する．

1. はじめに

組込みシステムに対するニーズは日々増加する傾向にあり，グラフィック性能やネットワーク通信等の機能の追加のみならず，高性能化のため，高周波数のプロセッサの採用やマルチコア技術の導入が進んでいる．しかし，これらに伴い消費電力の増大が重要な課題となっている．特にモバイル端末では，バッテリー消費の増大につながり，使用継続時間の短縮等の製品価値に大きく影響している [1].

これに対処するために，SoC の精密化やメモリ割り当て方法の改良 [2] 等，既に多くのハードウェア機構やアルゴリズムが提案されている．また既存のハードウェアを利用したソフトウェアによる省電力手法も数多く提案されており，代表的なものとして，動的電源電圧・周波数制御 (DVFS) とマルチコアシステム向けの CPU コアの電源の切断・投入を動的に制御する CPU hotplug がある．汎用システムでも使用されている Linux においても，これらの機構をサポートしている．具体的には CPU hotplug 機能として Linux CPU hotplug が既にサポートしている．

一方で組込みリアルタイムシステムの分野においても，マルチコア化等の高性能化の技術が適用されており，将来的に省電力化の問題に直面すると見込まれる．しかしながら，組込みリアルタイムシステムで使用される RTOS において，これらの機構をサポートしているものは我々の知る限りはない．省電力機構を OS がサポートする利点として，タスクやスケジューラ等の状態を直接把握することができる点が挙げられる．これにより，パフォーマンスと消費電力のバランスをより制御しやすくなるため，ハードウェアやミドルウェアによる省電力機構以上に効率の良い省電力機構を構築できると考えられる．また，汎用 OS である Linux の機構をそのまま使用すると，リアルタイム性[*1]が保証できない等の問題が生じる．

そこで本研究では RTOS 向けに適した省電力機構の仕様を提案する．本研究では，組込みシステムで使用されている CPU hotplug 機能をマルチコア RTOS の一種である TOPPERS/FMP カーネル [3] 向けに最適化した FMP-hotplug の仕様を策定する．例として，RTOS のリアルタイム性を保証するために，CPU 電源遮断要求の受け付け時にできるだけ即座に CPU 電源の遮断を試みる．もし電源遮断対象となっているプロセッサ上でタスク等の処理を実行している場合には，処理の終了を待たず，CPU 電源遮断要求を取り下げ，エラーとする方法が挙げられる．この方法ついては，遮断要求を取り下げ，エラーとする方法以外にも遮断要求を受け付けたまま終了する方法も考えられるため，以降の章にて検討する．

またマルチコアには同等の性能のプロセッサを並列実行する SMP 型と，異なる性能のプロセッサを複数搭載する AMP 型の大きく２つの方式が存在するが，本研究では普及率の高い SMP 型マルチコアシステムを対象とする．

[1] 名古屋大学 大学院情報科学研究科
Graduate School of Information Science, Nagoya University
[2] 名古屋大学 大学院情報科学研究科 附属組込みシステム研究センター
Center for Embedded Computing Systems, Nagoya University

[*1] 定められた上限時間内にタスクの実行を完了しなければならないという，組込みリアルタイムシステムの要件の１つ

本論文の構成は，以下の通りである．まず，第 2 章にて本研究で使用する省電力手法である CPU hotplug について述べ，第 3 章にて本研究で使用する RTOS である TOPPERS/FMP カーネルについて述べる．第 4 章では TOPPERS/FMP カーネルと FMP-hotplug の要件を確認する．第 5 章では，本研究で策定する FMP-hotplug の API 仕様設計について述べ，第 6 章にて実装する．第 6 章では，まず本研究で使用する評価ボードについて述べ，対応している CPU 電源制御方法を比較し，採用する制御方法の根拠を説明する．その後，実際に実装した処理について述べる．第 7 章では，FMP-hotplug の RTOS の性能要件を満たしているかを評価する．第 8 章で，本論文のまとめと今後の展望を述べる．

2. 対象とする省電力手法

本研究で RTOS に構築する省電力手法である，CPU hotplug について述べる．

CPU hotplug とは，システム実行中の動的な CPU の追加と取り外しを可能にした機能である．従来はミッションクリティカルなデータセンター等での信頼性，可用性，保守性の維持を目的とした機能である．しかし，近年のシステムの省電力化の傾向から，CPU hotplug の CPU 電源制御を用いて，システム負荷率の低い時に一部の CPU 電源遮断を実施する等の SMP 型マルチコアシステム向けの省電力機能として使用されるケースがある．

組込みリアルタイムシステムで使用されるプロセッサは汎用コンピュータと比較して低性能であり，構築可能なソフトウェアによる省電力手法には限りがある．そのため，本研究では CPU hotplug をベースとしたマルチコアリアルタイムシステム向けの省電力機構を構築する．

3. TOPPERS/FMP カーネル

TOPPERS/FMP カーネルとは，TOPPERS プロジェクトにおいて ITRON 仕様をベースとして開発しているリアルタイムカーネルのうち，マルチコア対応したものである．

API について，TOPPERS/FMP カーネルでは，システムインタフェースレイヤ，カーネル，各種のシステムサービスを，上位階層のソフトウェアから使うためのインタフェースとしている．

またプロセッサについて，システムの初期化時と終了時に特別な役割を果たすものを，マスタプロセッサと呼び，システムに 1 つ存在する．マスタプロセッサ以外のプロセッサを，スレーブプロセッサと呼ぶ．カーネル実行中では，マスタプロセッサとスレーブプロセッサの動作に違いはない．

TOPPERS/FMP カーネルは，処理単位を登録時に 1 つのプロセッサに割り付けて実行するが，マイグレーションにより処理単位登録後に，割付けプロセッサを変更することが可能である．マイグレーションとは，処理単位の登録後に，割付けプロセッサを変更することである．TOPPERS/FMP カーネルでは，システムの特性を柔軟に変更できるようにするため，汎用 OS である Linux が持つようなシステム負荷の平準化機構を持たず，自動的なタスク・マイグレーションは実施されない．代わりにタスクを実行するプロセッサを変更するマイグレーション API を提供することで，ユーザによる呼び出しでタスクのマイグレーションを可能としている [5]．

4. OS と FMP-hotplug の要件

本節では，FMP-hotplug の仕様を策定するにあたり，考慮すべき要件・要求を確認する．

4.1 TOPPERS/FMP カーネルの要件

TOPPERS/FMP カーネルに対する性能要件を以下に示す．

- リアルタイム性
  - (1) 最悪実行時間
    RTOS では，API の処理の実行時間に上限が定まることが必要である．
  - (2) 予測可能性
    タスクのスケジューリングや実行時間がアプリケーション開発者から予測可能であることが求められる．また，OS の最悪実行時間や割込み応答時間の上限時間が定まらない場合も，予測可能性が低くなるため十分考慮する必要がある．

TOPPERS/FMP カーネルに対する機能要件を以下に示す．

- CPU hotplug 機能
  - (1) 電源制御対象
    システムの実行には最低限 1 つのプロセッサが必要なことから，マルチプロセッサのうち，マスタプロセッサ以外のスレーブプロセッサを CPU hotplug 機能による電源制御対象とする．
  - (2) CPU 電源制御開始トリガの変更
    CPU 電源の遮断処理や投入処理を開始するトリガは，システムの特性により大きく異なる．そのため，hotplug 機能の柔軟性を向上させるために，システム特性を把握している開発者任意の方式に変更できるよう対応する必要がある．また性能要件で述べた予測可能性を保証するため，タスク・イベントのマイグレーション等の，最悪実行時間が予測できない CPU 電源遮断関連の処理は API として実装せず，開発者によるアプリケーションレベルでの実現を求める．

4.2 FMP-hotplug の要件

FMP-hotplug の設計要件を以下にまとめる．

(1) FMP-hotplug の追加による既存の TOPPERS/FMP カーネルの仕様と機能への変更は最小限にし，TOPPERS/FMP カーネルの要件に影響を与えない

(2) FMP-hotplug の仕様は既存の TOPPERS/FMP カーネル用 API の仕様策定方針に従う
仕様策定方針を以下に示す．
- (a) 非タスクからの API 発行においては, 対象タスクか実行状態と実行可能状態で動作を変えない．
- (b) タスクと非タスクで呼び出す API に対称性を持たせる．

(3) アプリケーションレベルで全ての CPU 電源制御処理が実現可能となる機能を提供する

(3) に関しては，4.1 節の機能要件 (2) を満たすために定めた．このためカーネルは，直交性を持つ最低限の機能を提供し，開発者がそれらを組み合わせることで機能を実現できるようにする．

FMP-hotplug の要求機能を以下に述べる．

(1) CPU 電源遮断機能
4.1 節の性能要件を満たすために，CPU 電源遮断を実行することで指定された CPU 電源を可能な限り即座に遮断する．即座に CPU 電源を遮断できない場合はエラーとする．またユーザからの明示的な電源復帰命令がない限り，可能な限り電源を遮断した状態を維持する．

(2) CPU 電源投入機能
4.1 節の性能要件を満たすために，CPU 電源投入を実行することで指定された CPU 電源を可能な限り即座に投入する．即座に CPU 電源が投入できない場合はエラーとする．

(3) CPU 電源状態参照機能
プロセッサの電源の状態（遮断状態・投入状態）を参照する．

5. API 仕様設計

FMP-hotplug の設計方針として，4.1 節で述べた API 処理の実行時間の制約と予測可能性の保証のため，4.2 節の要求機能で述べた最低限の機能のみを API として設計する．以下に設計する API の仕様について述べる．

5.1 電源管理方法

はじめに，CPU 電源の遮断を実現するために，OS の管理を必要とするかどうかを考慮する必要がある．CPU 電源の遮断を OS が管理しない場合，実装が容易であるが，電源管理のトリガとなるタスクや割込みの動作は OS のスケジューラやディスパッチャが担うため，タスクや割込みの開始・終了を検知することができず，無意味な電源操作が増加する可能性がある．またタスクの状態も把握することができないため，タスクが実行状態のまま CPU 電源を遮断し，タスク状態を不安定にすることも考えられることもあり，OS による電源管理を採用する．

5.2 CPU 電源遮断タイミング

5.1 節に関連して，以下の 2 通りの電源遮断タイミングが考えられる．

(1) 即時遮断方式

(2) 遮断待機方式

(1) は，電源遮断 API 呼び出し後，プロセッサ上で実行中の処理の存在に関わらず，即座に電源を遮断する方式である．この方式では，タスクの終了処理等を実行する必要がなく，簡易的な制御で実現することができるが，電源遮断後のシステムの不具合を発生させる可能性がある．

(2) は，ユーザによるタスクや割込み等の終了やマイグレーション処理後に CPU 電源を遮断する方法である．この方式では，処理の終了を待つため，電源遮断要求から実際の電源遮断までにかかる時間は予測不能であり，4 章の性能要件と要求機能に違反する．

(2) については，電源遮断時に対象プロセッサ上で実行中の処理が存在しない状態にする必要がある．そのため，その状態にするためのレイテンシを極力最小化するために，新たな処理の開始を禁止するプロセッサ状態を定義し，電源遮断要求受付後，即座にその状態へ移行する方法や，そもそも対象プロセッサ上でタスク等の処理の実行中の場合，エラーとなり，遮断しないようにする方法が考えられる．一方，実行状態の処理以外にも，セマフォ待ちのタスク等の待ち状態の処理の取り扱いついても検討する必要がある．このことは，電源遮断ポリシーに関わる．つまり，可能な限りプロセッサ電源を遮断したいのか，それとも空き時間にのみプロセッサ電源を遮断したいのか等を明確にしなければならない．

本研究で構築した FMP-hotplug では，4.2 節の要求機能に従い，また OS の動作を安定させるために (2) の方法を採用した．ただし，4.1 節の要件を満たすために，実行中の処理が存在する場合は，エラーとなり，電源遮断を実施しないようにした．また電源遮断要求を受け付けたことを示すための新たなプロセッサ状態を定義する．これをオフライン状態と呼び，この対となる状態をオンライン状態と呼ぶ．これにより実行中の処理が存在せず，CPU 電源の遮断が可能と判断された場合は，プロセッサをオフライン状態に切り替え，CPU 電源を遮断する．

5.3 API 呼び出し元

電源制御 API を実行するプロセッサにより，処理は大きく変わる．この場合以下の 2 通りが考えられる．

(1) 電源制御対象プロセッサ実行方式
(2) マスタプロセッサ実行方式

電源復帰処理については，対象となるプロセッサの電源は遮断されているため，(2) の方式のみ使用可能となる．電源遮断処理に関しては，いずれも対応可能なため，以下より電源遮断処理について検討する．

(1) では，すべての処理を電源遮断対象のプロセッサで完結することができるというメリットがある．この方式を採用する場合は，API を呼び出すタスク（以降，電源遮断タスクと呼ぶ）の状態に注意する必要がある．電源遮断タスクは CPU 電源遮断と同時に強制的に消滅するため，TCB [*2] 上では実行状態であるが，実際には実行されていないという矛盾が発生し，TOPPERS/FMP カーネルの要件に違反する．これを回避するために API 実行中に電源遮断タスクを終了する必要がある．

(2) では，電源遮断処理完了後，タスクを終了することができるため上記の違反は発生しない．しかしタスク・マイグレーションを実施した際に問題が発生する．[*3] 解決策として，電源遮断タスク自体をマイグレーションする方法が挙げられる．具体的には，電源遮断対象のプロセッサ上で電源遮断タスクを実行し，タスク・マイグレーションを実行する．その後，他のプロセッサへ自身をマイグレーションし，電源遮断処理を実施するというものである．しかし，オーバヘッドが確実に大きくなり，更に直交性が悪いという問題があり適さない．またローカルタイマ停止やキャッシュ書き戻し等のプロセッサ固有のハードウェア処理も実施する必要があるため，電源遮断タスクのマイグレーションは必至となる．

このため，FMP-hotplug では，CPU 電源遮断機能を電源遮断対象プロセッサが実行し，CPU 電源復帰機能をマスタプロセッサが実行するよう設計する．

また API の呼び出し元について，呼び出すタスクについても考慮する必要があり，以下の 2 通りが考えられる．

(1) 電源遮断タスクが電源遮断 API を実行する (通常タスク方式)
(2) 最低優先度タスクに電源遮断 API を実行させる (アイドルタスク方式)

(2) を使用する場合には，最低優先度タスクをアイドルタスクとして用意する必要がある．この方式は Linux CPU hotplug で適用されている方式であるが，TOPPERS/FMP カーネルにはアイドルタスクの概念がないことから現在のカーネルの設計に違反する．

FMP-hotplug では，(1) を使用した設計を行う．

[*2] TOPPERS/FMP カーネルには，タスクの内部状態や優先度等を管理するためのタスク管理ブロック (TCB) が存在する [4]
[*3] 現在の TOPPERS/FMP カーネルのタスク・マイグレーションの仕様では，タスクをマイグレーションさせることができるのは，そのタスクと同じプロセッサに割り付けられたタスクのみとなっている [4]

5.4 プロセッサ状態の明示方法

Linux CPU hotplug では，電源遮断要求を電源遮断フラグを用いて管理している．プロセッサがアイドル状態になったときに電源遮断フラグを確認し，フラグがセットされている場合に電源遮断処理を開始する．この方法は，5.2 節の遮断待機方式と 5.3 節のアイドルタスク方式に類似する箇所があるが，それぞれの方式では TOPPERS/FMP カーネルの制約に合致しないため，Linux CPU hotplug と同様の制御方法は採用しない．しかし，CPU 電源状態を参照したタスクのスケジューリング等を開発者が設計することが考えられるため，5.2 節で定義したオフライン状態を使用して，CPU 電源状態を参照できるようにする．

5.5 電源制御における過渡状態

プロセッサ状態を追加することにより，電源制御における過渡状態の扱いを検討する必要がある．電源投入時においては，電源投入要求の受け付けから即座に電源投入処理を開始できるため，過渡状態については考慮しない．そのため，以降は電源遮断時の過渡状態について述べる．

電源遮断時の過渡状態は，「電源遮断要求を受け付け，遮断処理を開始し，実際に遮断されるまでの間」のことを指す．電源遮断時においては，使用する 5.2 節の方式により，オフライン状態に切り替えるタイミングと切り替え後の処理が変わる．5.2 節の遮断待機方式では，電源遮断要求を受け付けてから，実際に遮断処理を開始するまでのレイテンシが実行中のタスク等の状態によるため不定である．このため，オフライン状態に切り替えることで，電源遮断要求受け付けを明示的なものにできるだけでなく，状態切り替え後の実行可能な処理を制限することで，遮断待機方式で述べたレイテンシの最小化方法を実現することができる．この時，オフライン状態へ切り替えることで，「実行すべき処理がない場合に電源遮断する」状態に変更されるとする．状態切り替え後の制限としては，遮断待機方式で述べたように，現在のプロセッサでの処理量を減らし，可能な限り早く電源を遮断するために，他タスク起動と他プロセッサからのタスク・マイグレーション等を禁止することが考えられる．割込みに関しては，外部へのタスク・マイグレーションができなくし，また実行優先度の関係から，禁止にせず受け付け可能にする．一方で CPU ロックは取得する．

検討事項としては，状態切り替え後に割込みを受け付けた場合，また受け付けた割込みにより新たにタスクが実行される場合，実行前にプロセッサ状態を切り替えるかどうかということである．オフライン状態のまま実行する場合は，オフライン状態にも関わらずタスクが実行し続けることになり，プロセッサ状態の一貫性を維持できない．このことから別の処理に切り換える際には，オフライン状態からオンライン状態に切り替えてから実行するべきである．

以下に例を紹介する．

(1) 電源遮断要求を受け付ける前に，対象プロセッサ上で他のタスクが実行しているケース
実行可能タスクがあるのでエラーとなり，オフライン状態に切り替わる前なのでオンライン状態でタスクは実行する
(2) 電源遮断要求を受け付けた後，かつ電源遮断処理を開始する前に他のタスクの実行を開始するケース
電源遮断タスクがCPUロックを取得するため，リリース後にそのタスクが実行される．よって一度電源は遮断され，再度CPU電源が復帰した後 (CPUロックリリース時) のオンライン状態でタスクは実行する

このことから電源遮断要求を受け付けた段階で実際に電源が遮断できるかはわからないため，電源遮断フラグを変更するのは，電源遮断処理の開始直前が最善である．

6. 実装

上記で策定したAPI仕様を基にTOPPERS/FMPカーネルへ実装する．

6.1 実装環境

本研究で使用した実装環境は，Renesas R-Car H1(Marzen) である（以降，評価ボードをMarzenを呼ぶ）．Marzenに搭載されているプロセッサはCortex-A9 MPCore(Quad Core) であり，各プロセッサの動作周波数は1.0GHzである．

Marzenで対応しているCPU電源制御方法には，次の2通り存在する
(1) SYSC方式
(2) WFI方式
それぞれについて説明する．

6.1.1 SYSC方式

SYSC方式とは，Marzenのシステム・コントローラ・レジスタ (SYSC) の一種である，ARMコア電源制御レジスタを使用するものである．使用する主要なレジスタを以下に示す．

- 電源ステータスレジスタ
  CPUコア毎の電源状態を示すレジスタ
- 電源遮断制御レジスタ
  指定するCPUコアに対応するビットに1を書込むことで，そのCPUコアの電源遮断を開始させるレジスタ
- 電源復帰制御レジスタ
  指定するCPUコアに対応するビットに1を書込むことで，そのCPUコアの電源投入を開始させるレジスタ

SYSC方式は，レジスタの変更後即座に電源制御を開始するという特徴がある．そのため，即時性に優れ，電源制御処理の実装が容易である．一方で電源遮断時には，プロセッサにて実行中もしくは実行待ちの処理の有無に関わらず電源を遮断できるため，OSやタスクの状態等を正確に把握しなければシステムの不具合を誘発する可能性がある．

6.1.2 WFI方式

WFI方式とは，プロセッサのアイドル状態にCPUクロックを停止するWFI(Wait For Interrupt) と呼ばれるARM命令を利用し，スヌープ・コントロール・ユニット (SCU) を変更することで，WFI実行時にCPUの電源遮断を実現するものである．

6.1.3 採用方式

WFI方式は，SYSC方式と比べ，少ないコード変更量とコードの汎用性の点において優位である．今回使用するTOPPERS/FMPカーネルではアイドル状態にWFIを実行するため，タスク等の制御を考慮する必要がなく，コードの変更量を少なくする事ができる．また，組込みシステムとして広く普及しているARMプロセッサの固有命令であるWFIを応用していることから，コードの汎用性が高いといえる．しかし一方で，ユーザによる電源遮断要求を受け付けた後，アイドル状態になり実際に電源が遮断されるまでにかかる時間は，電源遮断対象のプロセッサ上のタスクの数や状態等により定まらない．このことは4.1節の性能要件に違反するため，WFI方式を使用する場合においてもSYSC方式と同様に，予測可能性を高めるためにタスクの状態等を把握する必要がある．また，CPU電源の投入方法については，いずれも割込みによって行うことができるため，条件は共に同じである．

以上より，本研究では，電源制御方法として4.1節の性能要件を優先するためにSYSC方式を使用した．

6.2 CPU電源遮断処理

まず，電源遮断時のAPIのTOPPERS/FMPカーネルへの実装について述べる．なお，APIはTOPPERS/FMPカーネルの既存のAPIに倣い，CPUロックとタスクロックを取得する．

本研究では，実質的なCPU電源遮断処理以外のAPI処理として，①指定されたプロセッサのCPU電源遮断が可能かどうかのチェック，②オフライン状態への切り替え，③電源遮断タスクの実行状態の変更，④ローカルタイマの無効化，⑤データキャッシュの書き戻しを実装した．

①については，プロセッサ状態の確認，指定されたプロセッサ番号の確認，実行中のタスクの確認を実行する．なお，これらの確認をするためにプロセッサ管理ブロック (PCB) へアクセスするため，タスクロックとPCBロックを取得する．②については，5.4節でプロセッサ状態の明示方法として導入したオフライン状態に切り替えるための処理である．③については，5.3節で検討した問題を解決するためである．具体的には，電源遮断タスクが実行中であるが，後にCPU電源遮断により強制的に消滅するため，実行中に予めTCBを操作し，終了状態に変更する．④については，4.2のFMP-hotplugの要求機能を実現するため

のものである．TOPPERS/FMP カーネルでは，プロセッサがそれぞれローカルタイマを動作させているが，動作中のまま CPU 電源遮断を試みると，タイマティックが電源投入のトリガとなってしまい，ユーザの意図に反して CPU 電源の投入処理が開始されてしまう．そのため電源遮断処理の一部としてローカルタイマを無効化する必要がある．⑤については，OS 実行中に一部のプロセッサを取り外すため，キャッシュの一貫性について考慮する必要がある．そのため，CPU 電源遮断時にキャッシュの無効化とメモリへの書き戻しを実施する．本研究で実装した CPU 電源遮断処理のアクティビティ図を以下に示す．

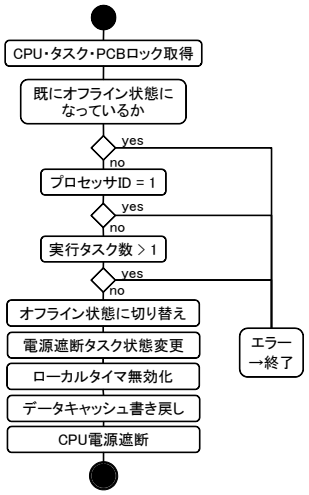

図 1 CPU 電源遮断実装例

6.3 CPU 電源投入処理

本実装では，実質的な CPU 電源投入処理以外の API 処理として，①キャッシュ等のターゲットの初期化，②ローカルタイマのグローバルタイマとの同期，③ローカルオブジェクトの初期化のみとした．

7. 評価実験

構築した省電力機構を評価するために，電力測定とオーバヘッド測定を実施した．電力測定に関しては，マルチプロセッサ環境において，プロセッサにかける負荷状態および電源状態を変更することで観測されるプロセッサ全体の消費電力の変化を測定する．オーバヘッド測定に関しては，プロセッサの電源状態を変更するのにかかるオーバヘッドを測定する．今回の評価では，前述の ARM Cortex-A9 MPCore を搭載したルネサスエレクトロニクス社製評価ボード R-Car H1(Marzen) を使用する．評価では，Marzen の４つのプロセッサのうち，マスタプロセッサとスレーブプロセッサのうちの第 2 プロセッサ（以降，プロセッサ２と呼ぶ）の２つのプロセッサを使用した．残りの２つのプロセッサは停止させている．

またリアルタイム OS として，TOPPERS/FMP カーネルを Marzen 上で実行した．本研究では，リアルタイム OS の実行中の電力とオーバヘッドの測定を実施するが，同時に TOPPERS/FMP カーネルを，電力測定において，プロセッサの負荷状態と電源状態の動的変更処理，およびオーバヘッド測定においては，測定に使用するパフォーマンスカウンタの初期化や起動処理と測定開始時のフラグの投入処理を実行するために使用する．

7.1 電力測定

電力測定は，Marzen 上には電力測定用回路が存在しないことから，代用として電流測定用回路を応用することでプロセッサの消費電力を概算する方法を採用した．Marzen は，DC12.0V の単一出力電源で動作し，スイッチングレギュレータを使用することで，12V 電源から DDR3 メモリやプロセッサ等の動作電圧の異なるペリフェラルに適した電源を生成する仕様となっている．更に，スイッチングレギュレータにより生成されたペリフェラルの電源回路にはそれぞれ，電流測定用回路として回路内電流を測定するための抵抗が存在する．抵抗の抵抗値を R，両端電圧を V_R とすると，求める回路内電流 (I_R) は以下のように計算することができる．

$$I_R = V_R / R \tag{1}$$

以上より，ペリフェラル電源回路電圧を V_P とすると，ペリフェラル電源回路の消費電力 W_P は (1) 式を使用すると以下のように計算できる．

$$W_P = V_P * I_R \tag{2}$$

Marzen における H1 プロセッサのペリフェラル電源回路では，R=10(mΩ) , V_P=1.2(V) であるため，求める消費電力 W_{H1} は (1) 式，(2) 式を利用して以下のように概算することができる．

$$W_{H1} = 1.2 * V_R / 10^3 = 1.2 * 10^{-3} * V_R \tag{3}$$

本研究での電力測定において，回路内電流の測定抵抗にピンをハンダ付けし，更にノイズを削減するために自作のフィルタを装着し，デジタルメータ IWATSU VOAC 22 を用いて測定した．プロセッサの負荷状態として，ビジーループ実行状態とアイドル状態の 2 パターン用意した．ビジーループ実行状態は，ビジーループを実行するタスクを特定のプロセッサ内で実行している状態を指し，アイドル状態は，プロセッサに割り当てられた実行可能状態のタスクが存在せず，WFI モードへ遷移した状態を指す．またマスタプロセッサに関して，プロセッサ状態を変更するためにユーザー入力受付タスクを常に実行している．そのため，マスタプロセッサに限り，上記の負荷状態に，入力受付タスク分の負荷を加算する必要がある．

表 1 H1 プロセッサの消費電力

マスタプロセッサ	プロセッサ 2	電力値
メインループ	アイドル状態	0.922 [W]
ビジー+メイン	アイドル状態	1.023 [W]
メインループ	ビジーループ	1.038 [W]
ビジー+メイン	ビジーループ	1.132 [W]
メインループ	電源遮断状態	0.912 [W]
ビジー+メイン	電源遮断状態	1.013 [W]

表 2 H1 プロセッサの電力削減量

状態変化前	状態変化後	電力削減量・削減率
メイン+ビジー	メインループ	≈ 0.100 [W] (≈ 10.1%)
ビジーループ	アイドル状態	≈ 0.113 [W] (≈ 11.6%)
ビジーループ	電源遮断状態	≈ 0.122 [W] (≈ 12.8%)
アイドル状態	電源遮断状態	≈ 0.010 [W] (≈ 1.04%)

プロセッサの電源状態として，通常状態と電源遮断状態を用意した．通常状態とは，プロセッサの電源が投入された状態を指し，電源遮断状態は，任意でプロセッサの電源を遮断し，電源が投入されていない状態を指す．またマスタプロセッサに関して，Marzen の仕様により，電源を遮断することが不可能なため，マスタプロセッサに限り，通常状態のみとする．

電力測定の結果を**表 1** に示す．測定の結果，プロセッサ状態変化による消費電力の削減量を**表 2** に示す．

この結果より，アイドル状態と電源遮断状態の比較ではほとんど消費電力に差が生じないことがわかった．原因として，プロセッサ内部には CPU コア以外にデータキャッシュや命令キャッシュのみならず，高機能化に伴う浮動小数点演算ユニット等の高機能化要求に応じるための処理装置が搭載されているため，相対的にプロセッサ中に占める CPU コアの消費電力比が減少していることが挙げられる．

7.2 オーバヘッド測定

プロセッサの電源遮断処理と電源投入処理の両方についてオーバヘッドの測定を実施する．対象となるプロセッサはいずれの場合においても測定にパフォーマンスカウンタを使用する．電源操作の対象となるのはプロセッサ 2 であり，マスタプロセッサはパフォーマンスカウンタを起動と終了を担う．確実な時間測定には，測定対象のプロセッサ 2 でパフォーマンスカウンタの操作をするのが最適である．しかし，プロセッサの電源操作にかかるオーバヘッドを操作対象のプロセッサ内部のカウンタで測定するのは不可能である．そのため本評価では，パフォーマンスカウンタ開始・終了のトリガとなるフラグを用意した．パフォーマンスカウンタを操作するマスタプロセッサ上のタスクは電源操作の開始前に実行状態にし，上記のフラグ状態をループで常時確認する．またプロセッサ 2 が電源遮断状態もしくは電源投入状態となった場合をカウンタ終了のトリガとする場合には，フラグを使用せずに，プロセッサ 2 の電源状態レジスタをループで参照することで実現する．

表 3 電源遮断処理のオーバヘッド

処理番号	平均値	最大値
①	2.43 [us]	2.88 [us]
②	301 [us]	302 [us]

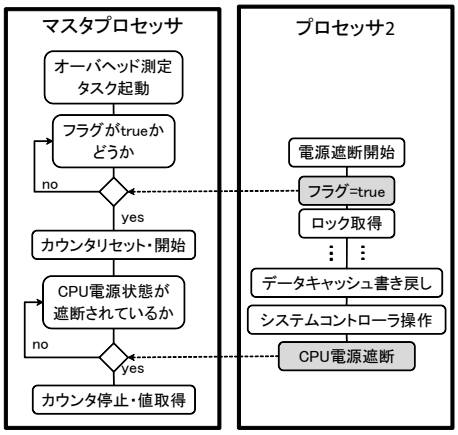

図 2 CPU 電源遮断時オーバヘッド測定方法例

オーバヘッド測定は各パターンにおいて 20 回試行し，最大値と平均値を出す．それぞれについて以下に詳細を述べる．

7.2.1 電源遮断処理のオーバヘッド

電源遮断処理に関しては，6 節で述べたように，TOPPERS/FMP カーネルの API として実装してある実質的な CPU 電源遮断処理と，アプリケーションとして実装してある，ローカルタイマの無効化等の電源遮断の前処理の大きく 2 つの処理系に分かれている．そのため，電源遮断処理のオーバヘッド測定として，①API にて実現している実質的なプロセッサ電源遮断処理単体のオーバヘッド，②プロセッサ電源遮断に関わる全処理のオーバヘッドの 2 通りの測定を実施する．今回の評価で使用した②の測定方法について**図 2** に示す．

電源遮断処理のオーバヘッドの測定結果を**図 3** に示す．実行時間はサイクル数とプロセッサ周波数の商で計算することができ，その結果を**表 3** に示す．①と②において，平均値と最大値の差はそれぞれ，0.45us と 1us である．RTOS

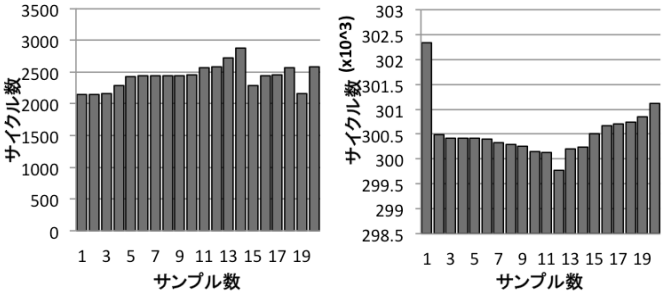

図 3 電源遮断処理のオーバヘッド
左:①の評価結果　右:②の評価結果

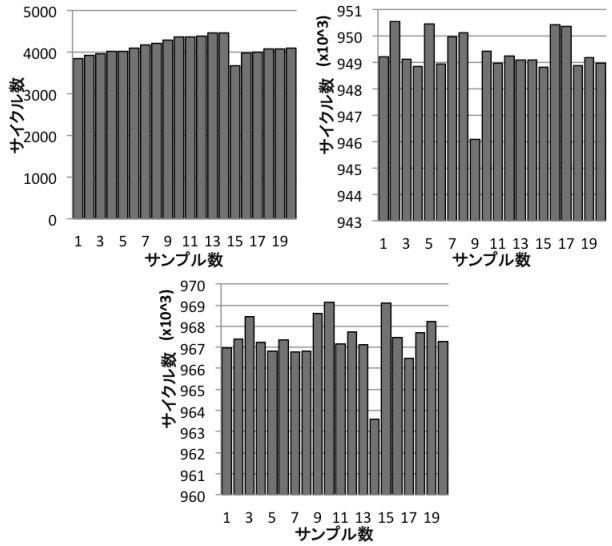

図 4 電源投入処理のオーバヘッド
左上:①の評価結果　右上:②の評価結果　下:③の評価結果

表 4 電源投入処理のオーバヘッド

処理番号	平均値	最大値
①	4.12 [us]	4.46 [us]
②	949 [us]	951 [us]
③	967 [us]	969 [us]

における実行時間の上限は実装するシステムにより大きく異なるため，一概にこれらの結果が RTOS に適切であることを示すことが非常に困難である．しかし，本研究で使用している TOPPERS/FMP カーネルで使用されるシステム時刻粒度が 1ms であり，それ以下の実行時間が測定できたことから，本研究で構築した電源遮断処理は RTOS の一種である TOPPERS/FMP カーネルに適することが言える．

7.2.2 電源投入処理のオーバヘッド

電源投入処理に関しては，7.2.1 節と同様に，API 実装の実質的なプロセッサ電源投入処理と，アプリケーション実装の，新たに電源投入された CPU での OS の起動処理等の，２つの処理系に分かれる．そのため，電源投入処理のオーバヘッド測定として，①実質的なプロセッサ電源投入処理単体のオーバヘッド，②プロセッサ電源投入開始から OS の起動処理までのオーバヘッド，②プロセッサ電源投入開始からディスパッチが開始され，実行待ちとなっていたタスクが起動するまでのオーバヘッドの 3 通りの測定を実施する．

電源投入処理のオーバヘッドの測定結果を**図 4** に示す．実行時間の結果を**表 4** に示す．①，②，③の，平均値と最大値の差はそれぞれ，0.34us，2us，2us である．この結果について，7.2.1 節と同様に，TOPPERS/FMP カーネルで使用されるシステム時刻粒度が 1ms であり，それ以下の実行時間が測定できたことから，本研究で構築した電源遮断処理は RTOS の一種である TOPPERS/FMP カーネルに適することが言える．

8. おわりに

本研究では，マルチコア向け組込みリアルタイムシステムに適用可能な，RTOS がサポートする省電力機構の仕様を提案した．本研究で策定した仕様を基に構築した省電力機構をマルチコア RTOS の一種である TOPPERS/FMP カーネルへ実装し，その評価を実施した．本研究の省電力機構として，汎用システムで既に実用化されている CPU hotplug を使用した．CPU hotplug を採用した理由として，OS がサポートする省電力機構であること，そして，組込みシステムで使用できるプロセッサの性能に見合う省電力機構であると判断したためである．評価に関して，CPU 電源制御のオーバヘッドは，電源遮断処理と電源復帰処理のいずれにおいても，TOPPERS/FMP カーネルに適する結果を得ることができたが，消費電力の削減量は，プロセッサ中の CPU の消費電力比が低いことから，アイドル状態とほとんど変わらない結果となった．今後の展望は，パフォーマンスと消費電力のバランスを考慮した，性能が異なるプロセッサを複数搭載したヘテロジニアスマルチコアを使用した省電力機構の仕様を提案する予定である．

参考文献

[1] Kim, S., Kim, H., Kim, J., Lee, J. and Seo, E.: Empirical Analysis of Power Management Schemes for Multicore Smartphone, *ICUIMC '13 Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*, Vol. 2013, No. 109, p. 7 (2013).
[2] 小林さとみ，中西恒夫，　福田晃：組み込みシステムにおけるオンチップ/オフチップメモリアーキテクチャを対象とした省電力ページングアルゴリズム，情報処理学会研究報告，Vol. 2002, No. 13(OS-89 EVA-2), pp. 155–161 (2002).
[3] TOPPERS プロジェクト：http://toppers.jp/.
[4] TOPPERS 新世代カーネル統合仕様書：http://toppers.jp/docs/tech/ngki_spec-150.pdf.
[5] 石田利永子，本田晋也，高田広章，福井昭也，小川敏行，田原康宏：TOPPERS/FMP カーネル:リアルタイム性と高スループットを実現可能な組込みシステム向けマルチプロセッサ用 RTOS，コンピュータソフトウェア，Vol. 29, No. 4, pp. 219–243 (2012).