Vol.2014-SLDM-165 No.32
Vol.2014-EMB-32 No.32
2014/3/16

マルチコア向けリアルタイムOSにおける省電力機構

大橋　孝輔[1,a)]　本田　晋也[1]　舘　伸幸[2]　高田　広章[1]

概要：近年，組込みシステム分野において，高性能化に伴うシステムの消費電力が増加しており，省電力化が課題となっている．汎用 OS に関しては，様々な省電力手法が提案されているが，制御系で使用される組込みリアルタイム OS(RTOS) においては，標準的な仕様が提案されていない．本研究では，性能ヘテロジニアス・マルチコア向け組込み RTOS に適用可能なクラスタ・スイッチングをベースとした省電力機構の仕様を提案する．そして策定した仕様を元に実装した省電力機構を FMP-クラスタ・スイッチングとし，その省電力性能を評価する．

キーワード：リアルタイム OS，省電力機構，性能ヘテロジニアス・マルチコア，クラスタ・スイッチング

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，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，使用継続時間が短縮する等，製品の価値を低下させてしまう [1].

現状として，スマートフォンやタブレット等のモバイル端末においては，グラフィック性能やネットワーク通信等の機能追加への要求が急速に増大しており，それに見合うバッテリー容量や半導体の低消費電力技術が追いつかない状態となっている．一方で，同時にユーザからはより長いバッテリー寿命を要求しており，これらの相反した要求を実現するために，新しい SoC の設計手法の開発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ソフトウェアによる省電力手法が数多く提案されている．代表的なものとして，動的電源電圧・周波数制御 (Dynamic Voltage and Frequency Scaling,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[3] 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つつあり，将来的に省電力化の問題に直面すると予測される．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 における省電力機構には，標準的な仕様がなく，現状では各 OS ベンダが個別にサポートしている．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．具体的に，Linux CPU hotplug では，ユーザからの CPU 電源遮断要求を受け付けると，即座に遮断するのではなく，CPU の次のアイドル時に電源遮断処理が実行される．このことから，要求の受け付けから実際に CPU 電源が遮断されるまでの時間が予測できないため，上記の RTOS のリアルタイム性の要件に違反することになる．

そこで本研究では組込み RTOS に適した省電力機構の仕様を提案した．

現在，マルチコア・プロセッサの構成のうち，同種のコアを複数実装したホモジニアス・マルチコアと，性能やアーキテクチャ等が異なるコアを複数実装したヘテロジニアス・マルチコアの 2 種類が存在し，様々な製品に利用されている．

[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
[a)] ohashi@ertl.jp

[*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

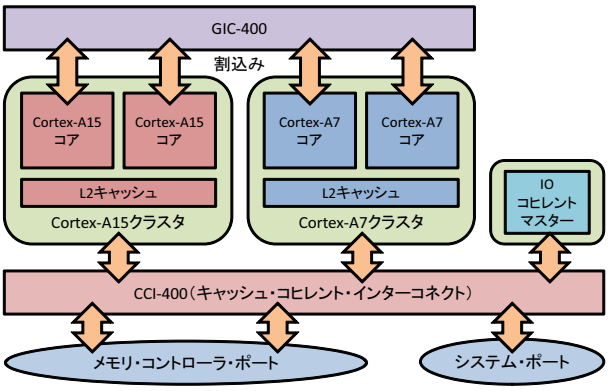

図 1 big.LITTLE システムモデル図
Fig. 1 big.LITTLE system model

本研究では，ヘテロジニアス・マルチコアのうち，アーキテクチャは同じであるが，性能が異なるコアを複数実装した性能ヘテロジニアス・マルチコア構成を対象とした．そして，新しい省電力技術として注目されている ARM 社の big.LITTLE Processing[4] を使用し，そのユースケースの 1 つとして提案されているクラスタ・スイッチング機能をマルチコア RTOS の一種である TOPPERS/FMP カーネル向けに最適化した FMP-クラスタ・スイッチングの仕様を策定する．

本論文の構成は，以下の通りである．まず第 2 章にて，適用する省電力機構を説明し，第 3 章で，省電力機構の仕様を策定する．第 4 章で，仕様を元に実装した FMP-クラスタ・スイッチングの評価を実施し，最後に第 5 章で本論文をまとめる．

なお，本研究は実践的情報教育協働ネットワーク enPiT[5] の一環として実施した．

2. 対象とする省電力手法

性能ヘテロジニアス・マルチコア向け RTOS へは big.LITTLE Processing を用いた省電力手法を構築する．

2.1 big.LITTLE Processing

big.LITTLE Processing（**図 1**）では，1 つの SoC へ搭載した，2 種類の性能が異なる，同一アーキテクチャのプロセッサを組み合わせることで，要求される幅広いパフォーマンスを効率よく実行できるよう考慮されている．搭載されるプロセッサについて，モバイル端末が出力できる最大のパフォーマンスを実行できる高性能なプロセッサと，ある程度のタスクを実行するために十分なパフォーマンスを持ち，更に消費電力との効率を最大限に調整されたプロセッサの 2 種類であり，それぞれを big プロセッサ，LITTLE プロセッサと呼ぶ．つまり，高負荷タスクを big プロセッサ上で，低負荷タスクを LITTLE プロセッサ上で実行することで，SoC での消費電力効率を最大限にすることが可能である．

図 2 ユースケース：クラスタ・スイッチング
Fig. 2 Usecase:cluster switching

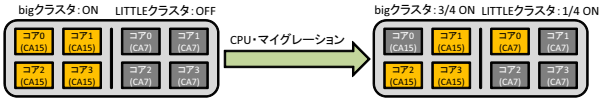

図 3 ユースケース：CPU マイグレーション
Fig. 3 Usecase:CPU migration

ARM が提唱している big.LITTLE Processing に対応した SoC では，big プロセッサとして，ARM Cortex-A15（以下，CA15 とする），LITTLE プロセッサとして同 Cortex-A7（CA7）を使用している．また CCI-400（Cache Coherent Interconnect）を呼ばれる外部デバイスを搭載し，それを経由することで，CA15 と CA7 のプロセッサ間でハードウェアによるキャッシュ等のデータ同期を可能としており，ソフトウェアの実行形態に適した様々な big.LITTLE ユースモデルに対応できるように設計されている．

2.2 ユースケース

big.LITTLE Processing には，システムの消費電力削減のためのユースケースとして，クラスタ・スイッチングと CPU マイグレーションの 2 種類のユースケースが提案されている．

クラスタ・スイッチング（**図 2**）は，LITTLE クラスタ内の CPU コアで実行されていた全てのコンテキストを同数の big クラスタへマイグレーション，もしくはその逆をするものである．しかしながら，CPU クラスタ毎の消費電力には大きな違いがあるため，マイグレーション前後で電力効率が悪化するケースも存在する．

一方で CPU マイグレーション（**図 3**）は，他の CPU クラスタ内の物理的に対応付けられた CPU コアへマイグレーションする方法である．

3. 仕様設計

FMP-クラスタ・スイッチング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．FMP-hotplug で考慮吸う必要のある要件・要求は，実装対象となる TOPPERS/FMP カーネルの性能要件と機能要件，そして FMP-hotplug への設計要件と要求機能である．

3.1 省電力機構のアプローチ

本研究での性能ヘテロジニアス・マルチコア向け省電力機構は，RTOS 上のソフトウェアとして実現可能なものであることが前提である．RTOS には，カーネル・スケ

ジューラのような複雑なスケジューリング機構や DVFS 等の省電力機能を持たず，OS として最低限の機能しか実装されていない．またリアルタイム性の保証等，多くの制約が RTOS には伴うため，RTOS のソフトウェアとして実現する省電力機構も機能は最低限であることが望ましい．このことから，RTOS へ実装する省電力機構は，スケジューラや負荷率監視機構による自動実行をせず，ユーザが任意のタイミングで実行できるシステムとするのが適していると考えられる．また基本的な消費電力の考え方に基づくと，CPU クラスタ内の使用している CPU コアの個数に関わらず，2 つの CPU クラスタを同時に実行することは，両 CPU クラスタ内のキャッシュ等のペリフェラル電源を投入し続ける必要があるため，電力効率は悪いと言える．

このため，本研究では，性能ヘテロジニアス・マルチコア向け省電力機構として 2.2 節のユースケースの内，クラスタ・スイッチングを採用する．本研究では，RTOS として TOPPERS/FMP カーネルを使用することから，構築するクラスタ・スイッチングを FMP-クラスタ・スイッチングと呼ぶ．

3.2 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

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．

- クラスタ・スイッチング機能

(1) 電源制御対象

システムの実行には最低限 1 つの CPU コアが必要なことから，クラスタ・スイッチング実行中において，最低 1 以上の CPU クラスタの電源が投入されている．

(2) クラスタ・スイッチング対象

クラスタ・スイッチングを実行することでタスクや OS 本来の要件が満たせなくなるイベント，もしくはシステム全体に重大な障害を与えるイベントが実行されていないものをクラスタ・スイッチングの対象とする．

(3) クラスタ・スイッチング開始トリガの変更

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．また最悪実行時間が予測できない処理は API として実装せず，上記のクラスタ・スイッチング開始トリガとともに，既存の API を組み合わせる等の手順で開発者によるアプリケーションレベルで実現する．

3.3 FMP-クラスタ・スイッチングの要件

FMP-クラスタ・スイッチングの設計要件を以下にまとめる．

(1) FMP-クラスタ・スイッチング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-クラスタ・スイッチングの仕様は既存の 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, 対象タスクが実行状態と実行可能状態とで動作を変えない．

(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 FMP-クラスタ・スイッチングの基本処理が実現可能となる機能を提供する．基本処理には，CPU クラスタ電源復帰処理，CPU クラスタ電源遮断準備処理，CPU クラスタ電源遮断処理が含まれる．

(3) に関しては，3.2 節の機能要件 (3) 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-クラスタ・スイッチングの要求機能を以下に述べる．

(1) CPU クラスタ電源投入機能

CPU クラスタ電源投入を実行することで指定された CPU クラスタ電源を可能な限り即座に投入する．即座に CPU クラスタ電源が投入できない場合はエラーとする．

(2) CPU クラスタ電源遮断準備機能

クラスタ・スイッチング実行後に OS の正常動作を保証するために，最低限のタスクや割込み等のイベントをマイグレーションする．マイグレーションができない場合はエラーとする．

(3) CPU クラスタ電源遮断機能

CPU クラスタ電源遮断を実行することで指定された CPU クラスタ電源を可能な限り即座に遮断する．即座に CPU クラスタ電源を遮断できない場合はエラーとする．

3.4 API 仕様設計

FMP-クラスタ・スイッチングの設計方針として，3.2 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，3.3 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．

3.4.1 マスタプロセッサのマイグレーション

TOPPERS/FMP カーネルでは，マスタプロセッサとスレーブプロセッサを定義している．マスタプロセッサはシステムの起動時・終了時処理のみならず割込みハンドラの実行等の，マスタプロセッサのみに割り当てられた，システム実行中の処理が複数ある．このため，FMP-クラスタ・スイッチングでは，各 CPU クラスタ内でのプロセッサ ID が 0 の CPU コアをマスタプロセッサとして定義し，マスタプロセッサのイベントを，他 CPU クラスタのマスタプロセッサへマイグレーションできるようにする．

また，システム動作に必要な最低限のイベントのマイグレーションは，FMP-クラスタ・スイッチングを実現するための重要な処理である．システム動作に必要なイベント数は静的であり，オーバヘッドの規模は予測可能であるため，性能要件には違反しないと言える．このことから，システム動作に必要なイベントのマイグレーションに限り，FMP-クラスタ・スイッチングの機能の一部として，CPU クラスタ電源遮断準備機能として設計する．

3.4.2 クラスタ・スイッチングの実現方法

FMP-クラスタ・スイッチングにおいて，CPU クラスタのマイグレーションの間，2 つの CPU クラスタが同時に実行されるタイミングがある．そのため，このタイミングにおいて，ユーザタスクのマイグレーションを実行することは物理的に可能である．このことから，FMP-クラスタ・スイッチングの実現方法として，以下の 2 種類が考えられる．

(1) 一括ロック方式

FMP-クラスタ・スイッチング実行開始から終了まで両方の CPU クラスタをロックし，他のユーザタスクや割込みが実行できないようにする方法

(2) 機能分割方式

実行する処理を 3.3 節で述べた，CPU クラスタ電源投入処理，CPU クラスタ電源遮断準備処理，CPU クラスタ電源遮断処理の 3 つの処理を分割し，処理間でユーザイベントの実行を許可する方法

本研究では，可能な限り消費電力を抑えるためと，全 FMP-クラスタ・スイッチングの実行時間を予測可能にするために，(1) の方式を採用する．

3.4.3 CPU クラスタ・CPU コア電源遮断タイミング

CPU クラスタ内の CPU コアの電源遮断のタイミングについて，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

電源遮断 API 呼び出し後，CPU コア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

(2) 遮断待機方式

実行中のユーザイベントの終了やマイグレーション処理後に CPU 電源を遮断する方式

本研究では，OS の動作を安定させるために (2) の方法を採用するが，予測可能性の要件から，CPU コア上で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．

3.4.4 API 呼び出し元

CPU クラスタ電源制御 API を実行する CPU クラスタについて，以下の 2 通りが考えられる．

(1) マイグレーション先 CPU クラスタ実行方式

(2) マイグレーション元 CPU クラスタ実行方式

まず，CPU クラスタ電源復帰処理については，対象となる CPU クラスタの電源は遮断されているため，(2) の方式のみ採用できる．CPU クラスタ電源遮断処理に関しては，いずれも対応可能であるが，適用対象の TOPPERS/FMP カーネルのタスク・マイグレーション仕様[*2] と，CPU クラスタの電源制御方法 [7] の点から，(1) の方法を採用する．

また，API を実行する CPU コアについては，マスタプロセッサかスレーブプロセッサが実行するかを検討することもできるが，特に違いが無いため，CPU クラスタ内のマスタプロセッサが実行するよう設計する．

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) 通常タスク方式

通常タスクが API を実行する方式

(2) アイドルタスク方式

アイドルタスクに API を実行させる方式

(2) を使用する場合には，アイドルタスクとして常時実行している最低優先度タスクを用意する必要がある．この方式は Linux 等の汎用 OS で一般的に使用されている方法であるが，TOPPPERS/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ことから，(1) の方式を採用する．

4. 評価実験

4.1 評価環境

本章の研究では，ARM が提供する開発プラットフォームである Versatile Express ファミリを使用した．具体的には，マザーボードとして Motherboard Express uATX（V2M-P1）を使用し，メインシステム CPU として big.LITTLE 環境が構築可能な ARM TC2 CoreTile Express A15x2 A7x3（V2P-CA15_A7）ドータボードを使用した（以降，TC2 を呼ぶ）．ドータボード上には，CA15 クラスタとして，Cortex-A15 MPCore（1.0GHz）を 2 コア，L1 キャッシュ

[*2] 現在の仕様では，(1) 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 CPU コアに割り付けられたタスクのみである，(2) マイグレーションを実行する CPU コアに割り当てられたタスクの他の CPU コアへのマイグレーションのみをサポートする，となっている [6]

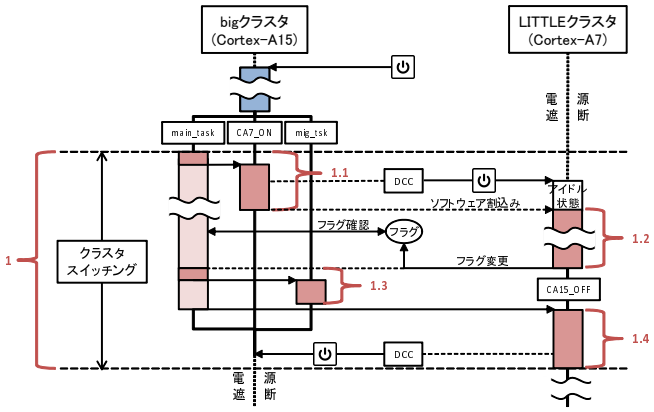

図 4 オーバヘッド測定箇所・範囲
Fig. 4 Points/Ranges of the overhead measurement

1. クラスタ・スイッチング
  1.1. CA7 クラスタ電源投入処理
    1.1.1. ソフトウェア処理（レジスタスイッチの切替等）
    1.1.2. アイドル復帰処理（アイドル状態から通常状態へ移行する処理）
  1.2. CA7 クラスタ FMP 起動処理
    1.2.1. ~ secondary_sta_ker() 呼出前
    1.2.2. secondary_sta_ker() ~ start_dispatch() 呼出前
    1.2.3. start_dispatch() ~ act_tsk()
  1.3. CA15 クラスタ電源遮断準備処理
  1.4. CA15 クラスタ電源遮断処理

図 5 オーバヘッド測定箇所・範囲一覧
Fig. 5 List of points/ranges of the overhead measurement

を 32kB，L2 キャッシュを 1MB 搭載している．CA7 クラスタしては，Cortex-A7 MPCore（800MHz）を 3 コア，L1 キャッシュを 32kB，L2 キャッシュを 512kB 搭載している．

4.2 評価内容

構築した省電力機構を評価するために，オーバヘッド，消費電力，及び消費電力量を測定した．消費電力測定に関しては，big.LITTLE 環境において，CPU にかける負荷状態を変更することで観測される各 CPU クラスタ毎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，FMP-クラスタ・スイッチングを実行するのにかかるオーバヘッドを測定する．消費電力量測定に関しては，FMP-クラスタ・スイッチングを実行するのに消費される電力量を測定する．今回の評価では，RTOS として TOPPERS/FMP カーネルを TC2 上で実行した．

オーバヘッドの測定に関して，CCI-400 のパフォーマンスカウンタとして利用し，電力測定と消費電力測定については，レジスタ・インターフェースからアクセス可能な TC2 上の電力モニタ機能と，オンボード・エネルギー・メータ機能を利用する．

表 1 FMP-クラスタ・スイッチングのオーバヘッド
Table 1 Overhead of FMP-cluster switching

測定箇所	平均値 (ms)	最大値 (ms)
1.	**1.649**	**1.652**
1.1.	**0.810**	**0.811**
1.1.1.	0.646	0.647
1.1.2.	0.165	0.165
1.2.	**0.480**	**0.491**
1.2.1.	0.001	0.001
1.2.2.	0.463	0.472
1.2.3.	0.015	0.015
1.3.	**0.023**	**0.024**
1.4.	**0.279**	**0.284**

表 2 TOPPERS/FMP カーネルの初回起動時のオーバヘッド
Table 2 Overhead of the first boot of TOPPERS/FMP kernel

測定箇所	平均値 (ms)	最大値 (ms)
2.	1.139	1.144

4.3 オーバヘッド測定

オーバヘッドの測定範囲について，**図 4** と**図 5** に示す．図 4 と図 5 の番号は対応している．また図 4 中の測定箇所を細分化した測定範囲を，図 5 では ~.~.1 , ~.~.2 等の番号で示す．

オーバヘッド測定は各パターンにおいて 10 回試行し，その平均値を出す．

FMP-クラスタ・スイッチングのオーバヘッドの測定結果を**表 1** に示す．また参考のために，TOPPERS/FMP カーネルの初回起動時のオーバヘッド測定結果も**表 2** に示す．この結果から，FMP-クラスタ・スイッチングの実行オーバヘッド時間は，TOPPERS/FMP カーネルの初回起動にかかる時間に近いことがわかった．このため，実際にクラスタ・スイッチングを実行する際には，開発者自身が実行中のシステムに影響が発生しないタイミングを計算する等十分な配慮が必要となる．またこの測定結果より，全体に占める CA7 クラスタ電源投入処理（1.1）にかかるオーバヘッドの割合が最大であることがわかった．更に調査したところ，1.1.2. の処理中の，レジスタ変更後に CA7 クラスタの電源投入完了の待ち処理（0.640ms）が大きく占めていることがわかった．この時間は，CA7 クラスタのハードウェアに依存し，支配的であることから，劇的な速度改善は難しいと言える．

4.4 電力測定

CPU コアの負荷状態として，高負荷状態とアイドル状態の 2 パターン用意した．高負荷状態は，ビジーループを実行するタスクを CPU コア上で実行することで再現しており，アイドル状態は，割り当てられた実行可能状態のタスクが存在せず，CPU コアが割込み待ち状態へ移行した

表 3 TC2 の消費電力 (CA15 クラスタ)
Table 3 Power consumption of TC2(CA15 cluster)

ビジーループ実行中の CPU コア数	0	1	2
消費電力 [W]	0.665	0.854	1.04
増加量 [W]	-	0.189	0.183

表 4 TC2 の消費電力 (CA7 クラスタ)
Table 4 Power consumption of TC2(CA7 cluster)

ビジーループ実行中の CPU コア数	0	1	2
消費電力 [W]	0.246	0.298	0.351
増加量 [W]	-	0.052	0.053

状態のことを指す．また TC2 の仕様では，CPU クラスタ内の CPU 単位での電源制御に対応していないため，電源遮断時の電力の測定は実施しなかった．

電力測定の結果を CPU クラスタ毎にそれぞれ**表 3** と**表 4** に示す．この結果から，CA15 CPU コアは CA7 CPU コアの約 3 倍の消費電力であることがわかり，TOPPERS/FMP カーネルにおけるクラスタ・スイッチングを用いた省電力機構による電力削減に期待ができる結果となった．

4.5 電力量測定

FMP-クラスタ・スイッチングによる CPU クラスタ毎の平均電力消費量を**表 5** に示す．また，**表 6** には，FMP-クラスタ・スイッチング実行中の CPU クラスタ毎の平均電力と平均負荷率を示す．

平均電力は，電力消費量の平均値（表 5）と，FMP-クラスタ・スイッチング実行中の，CPU クラスタの電源投入時間との商により算出した．CA15 クラスタは，FMP-クラスタ・スイッチングの実行時間分だけ電源が投入されているため，表 1 で示した平均値（1.65ms）を採用した．CA7 クラスタの投入時間については，CA7 クラスタ電源投入処理中のアイドル状態の時間を考慮し，1.00ms とした．

平均負荷率には，表 3 と表 4 の消費電力値を使用した．CPU クラスタ内の全 CPU コアがビジーループ実行時を 1 とした．

この結果から，FMP-クラスタ・マイグレーションを実行中，CA15 クラスタは平均約 73.5%，CA7 クラスタは平均約 96.6%の負荷がかかることが分かった．

全ての CPU クラスタに関して，FMP-クラスタ・マイグレーションを実行中の負荷率が高い結果となったため，FMP-クラスタ・マイグレーション実行時には予め割込み禁止処理を実行する等，更なる負荷率上昇を避ける手段を取るべきである．しかしながら，この結果に関しては，実装対象のシステムに依存するため一概には良し悪しを判断することはできない．そのためシステム開発者は，4.3 節～4.4 節までの評価結果と共に総合的に判断し，最適な実行タイミングを検討する必要がある．

表 5 CPU クラスタ毎の電力消費量
Table 5 Energy consumption of each CPU cluster

	CA15 クラスタ	CA7 クラスタ
平均消費量 [uJ]	1253.1	339.3

表 6 CPU クラスタ毎の平均電力・平均負荷率
Table 6 Average power consumption/load factor of each CPU cluster

	CA15 クラスタ	CA7 クラスタ
平均電力 [mW]	764.1	339.3
平均負荷率	0.735	0.966

5. おわりに

本研究では，性能ヘテロジニアス・マルチコア向け RTOS に適用可能な省電力機構の仕様を提案した．アーキテクチャとして，big.LITTLE Processing を使用し，そのユースケースの 1 つであるクラスタ・スイッチングを基本とした省電力機構を提案した．また仕様を元にした実装を行い，オーバヘッド，消費電力，消費電力量に関する評価を行った．

今後の課題として，FMP-クラスタ・スイッチングの実行オーバヘッドの改善が挙げられる．特に，CPU クラスタの電源投入処理における待ち時間に関しては，ハードウェア依存であるものの，時間確保方法の見直しによる改善の余地があると思われる．また，実際に使用されているシステムへの適用を考え，実システム環境に近いタスクセットを用いた，FMP-クラスタ・スイッチングのシステムへの影響を評価することも予定している．

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] : CPU hotplug Support in Linux Kernel, `https://www.kernel.org/doc/Documentation/cpu-hotplug.txt`.
[4] : big.LITTLE Processing - ARM, `http://www.arm.com/products/processors/technologies/biglittleprocessing.php`.
[5] 克郎井上，真二楠本，厚宏後藤，尚靖鵜林，博之北川：ぺた語義：実践的情報教育協働ネットワーク enPiT，情報処理，Vol. 55, No. 2, pp. 194–197 (2014).
[6] : TOPPERS 新世代カーネル統合仕様書，`http://toppers.jp/docs/tech/ngki_spec-150.pdf`.
[7] ARMLtd: Application Note 318 CoreTile Express A15x2 A7x3 Power Management, Technical Report DAI0318 (2000).