修士論文

題目

ハードウェアスケジューラを用いた
省電力リアルタイムマルチコアプロセッサの提案

指導教員

近藤 利夫 教授

平成 25 年度

三重大学大学院 工学研究科 情報工学専攻
計算機アーキテクチャ研究室

瀬戸　勇介 (412M512)

内容梗概

近年，プロセッサ性能の向上に伴う消費電力の増加が問題視されており，電力面で厳しい制約を持つ組み込みシステムではこれまで以上に高い省電力性能が求められている．組み込み分野には性能や電力以外にも信頼性に関わる多くの要求が存在し，それらがシステムの省電力性能に与える影響についても考慮が必要となる．本論文では，信頼性要求の 1 つであるリアルタイム性に関する消費電力面での課題の解決を図る．

リアルタイム性を保証するためには厳密な時間管理が求められ，処理完了時間の正確さがシステムの信頼性に大きく影響する．リアルタイムシステムは全てのタスクが WCET(worst case execution time：最悪実行時間) で実行された場合にも正確な動作を保証するため，極めて悲観的な想定の下にアーキテクチャを用意する必要がある．この余剰なアーキテクチャの動作による消費電力の増加は省電力性能の改善を行う上での障害となる．本論文では，単一 ISA(instruction set architecture：命令セットアーキテクチャ) の HMP(heterogeneous multi-core processor：ヘテロジニアスマルチコアプロセッサ) 構成を用いることでこの問題を解決する．

単一 ISA の HMP は同命令セットで異性能のコアを組み合わせたマルチコア構成となっており，タスク毎に必要なリソース量に合わせて選択的にコアを切り替えることで高い電力効率を実現する．商用製品への搭載例として ARM の big.LITTLE があり，省電力アプローチとしての有用性が証明されている．しかしながら，単一 ISAHMP のコア切り替えがタスクの処理完了時間に及ぼす影響が問題となっている．RTOS(real-time OS：リアルタイム処理向け OS) の提供するソフトウェアでのタスクスケジューリングでは正確な時間情報管理が困難となるため，既存の単一 ISAHMP を搭載したシステムはリアルタイム処理向けに最適化されていない．

そこで本論文では，ハードウェアスケジューラを用いたタスク管理の効率化手法を新たに提案する．提案手法により単一 ISA HMP をリアルタイムシステムへ適用することで，演算リソースの高効率利用が可能な省電力リアルタイムマルチコアプロセッサを実現する．対象の OS として Linux ベースの RTOS である RTAI を採用した．提案するハードウェアスケジューラのエミュレーション機構を QEMU 仮想環境上に C 言語で実装し，提案手法の有用性を証明した．エミュレーションの結果により，タスク実行時で約 14%の消費電力削減の達成を確認した．

Abstract

Embedded systems must ensure many varieties of reliability, "real-time" is one of them. Real-time systems require to meet the deadline of running task for correct working. Deadline limits execution time of a task. For example, the computational delay of braking of automobile system may cause a serious traffic accident. Real-time systems must satisfy the strict time constraint in any cases, even if all real-time tasks have run in the worst-case execution time (WCET). From this reason, a processor in real-time system equips sufficient hardware resources for the WCET running. In an average execution case, the hardware resources are excessive and cause an increase in power consumption. To solve this problem, a novel approach for power-efficient real-time systems is necessary.

This paper focuses on single instruction set architecture (single-ISA) heterogeneous multi-core processor (HMP). Single-ISA HMP is known as highly power-efficient technique of processor architecture. A single-ISA HMP consists of the same ISA but different performance cores. When a task runs on this architecture, the task is assigned to a suitable core according to the task feature. The single-ISA HMP can efficiently use the hardware resources and achieve low-energy processing. However, no scheduler in current OS support both the single-ISA HMP and strict real-time execution sufficiently. Therefore, to apply single-ISA HMP into real-time systems, this paper proposes a novel task scheduling method.

The proposed scheduling algorithm uses a hardware scheduler. The dedicated hardware achieves strict and efficient time management between HMP architectures and a software scheduler for power-efficient real-time multi-core processor. The target RTOS is RTAI based LinuxOS. For verification, this paper used an emulation system of proposed hardware scheduler implemented in C language on QEMU emulator. According to the evaluation result, the proposed scheduling method can improve approximately 28% of power consumption.

目 次

図 目 次

表 目 次

1　はじめに

近年，スマートフォンの普及や自動車内部の電子制御ユニットの統合等により，汎用システムだけでなく組み込みシステムにおいても単体で高い処理性能を持ったプロセッサの搭載が求められている．しかしその一方で，組み込み分野では電力面で厳しい制約を持つシステムも多く，プロセッサの処理性能の向上に伴う消費電力の増加が問題となっている．そのため，現在の組み込み分野ではプロセッサの省電力性能のさらなる向上が重要な課題であり，様々な観点から消費電力削減のためのアプローチが提案されている．組み込み分野では性能や電力以外の面でも要求・制約が存在し，それらがシステムの省電力性能へ与える影響も無視することは出来ない．本論文では，この要求・制約の内のリアルタイム性の保証に伴って生じる消費電力面での問題に着目し，新たな省電力化手法を提案することで問題の解決を図る．

リアルタイム性は組み込みシステムの時間管理性能に関する信頼性の指標であり，定められた処理完了時間を確実に守ることを保証する性質のことである．例えば，自動車のブレーキ制御システムでは僅かな処理の遅延が重大な事故につながるため，処理結果の正当性に加えて時間管理の厳密性も必要となる．以上より，リアルタイム性を満たす必要のあるリアルタイムシステムでは，タスクの処理完了に掛かる時間の正確さがシステム全体の信頼性に直結する．リアルタイムシステムはシステム上の全てのタスクが想定される最悪の処理完了時間で実行された場合にも正確な動作を保証しなくてはならない．この条件を満足するためには，開発の際に極めて悲観的な想定の下にアーキテクチャを用意する必要がある．平均的な実行時間でタスクが動作した場合，この余剰なアーキテクチャが消費電力の増加を招き，省電力性能の改善を図る上での障害となる．そこで本論文では，単一 ISA の HMP を用いることでリアルタイムシステムの省電力性能の向上を目指す．　現在の組み込み分野では，ARM の開発する big.LITTLE[1] を代表とするような単一 ISA の HMP 構成の利用が新たな省電力アプローチの 1 つとして注目されている．単一 ISA の HMP は同命令セットで且つ異性能のコアを組み合わせたマルチコア構成となっており，現行のタスクの実行により消費するリソース量に合わせて最適なコアを動的に切り替えることで高い電力効率を実現することが出来る．実際の商用組み込み製品への搭載実績から，省電力面での有用性も証明されている．しかしながら，単一 ISAHMP のコア切り替えがタスクの処理完了時間に及ぼす影響が問題となっている．RTOS の提

供するソフトウェアでのタスクスケジューリングでは正確な時間情報管理が困難となるため，既存の単一ISAHMP を搭載したシステムはリアルタイム処理向けに最適化されていない．そこで本論文では，ハードウェアスケジューラを用いたタスク管理の効率化手法を新たに提案し，単一ISA HMP のリアルタイムシステムへの適用を図る．単一ISA HMP 構成のアーキテクチャと RTOS との間に専用ハードウェアを介することでスケジューリングを最適化し，演算リソースの高効率利用が可能な省電力リアルタイムマルチコアプロセッサを実現する．

以下，2 章でリアルタイムシステムに関する概要とその問題点について述べ，3 章で対象となる単一ISA HMP の基本的な構成について述べる．4 章より本論文における提案手法の詳細な説明を行い，5 章で提案手法の有用性の検証のための実装とその結果について述べる．最後に，6 章で本論文のまとめと今後の展望について述べる．

2　リアルタイムシステム

2.1　組み込みシステムにおける要求

パーソナルコンピュータのような汎用情報処理目的のシステムに対して，携帯端末機器や家電機器，医療機器，輸送機械，工場機械，ロボット機械等の製品の制御を主目的として内部に組み込まれているコンピュータを組み込みシステムと呼ぶ．汎用システムと組み込みシステムは，それぞれの用途の違いから，求められる性能の指標や開発の際の制約についても差異を持つ．一般的にプロセッサの性能の指標となる処理速度と消費電力はトレードオフの関係となっており，ハイエンドからローエンドに至る種々の需要に合わせて様々な速度性能と省電力性能を持った商用プロセッサが現在提供されている．個々の製品毎で仕様の差が小さい汎用システムでは，システム規模に対する最適な速度/省電力性能のコアを搭載することで要求を満たすことが出来る．一方の組み込みシステムでは，速度性能や省電力性能の考慮だけでなく，高い耐故障・復旧性能やセキュリティ性，電源制約，省リソース及び省スペース設計要求等の製品毎の特徴に合わせた仕様に特化する必要がある．また，これらの組み込み分野における専用的な要求仕様の中でも，特にシステムの価値や性能を決定する指標としてシステムの信頼性が重要視される．汎用システムと同様に，処理結果の正当性の保証は組み込みの分野においてもシステムの信頼性を満たす上で重要である．加えて，組み込み環境を想定したノイズ耐性や耐熱性能，前述の耐故障・復旧性能やセキュリティ性も代表的な例である．これは医療機器や輸送機械への搭載を考えれば分かる通り，組み込み事例によっては予期しない動作の停止や誤動作に対するリスクが汎用システムに比べてはるかに大きくなるためである．本章では，組み込み分野で求められる信頼性の内のリアルタイム性と呼ばれる指標に関する概要とリアルタイム性の保証に伴って生じる消費電力面での問題について述べる．

2.2　リアルタイム性

リアルタイム性は組み込みシステムの時間管理性能に関する信頼性の指標であり，定められた処理完了時間を確実に守ることを保証する性質のことである．リアルタイム性を満たすシステムは厳密な時間資源の管理が必要であり，処理完了にかかる時間の正確さがシステムの信頼性に

直結する．自動車走行システムを例にとると，走行中のある時点でのブレーキ動作において，通常時と比較して著しい遅延が生じた場合に運転者に危険の及ぶ重大な事故を引き起こす可能性がある．これは，自動車内部の一部の動作タスクが規定の処理完了時間をオーバーしたためにシステム自体の破綻を引き起こした一例である．上の例のようにリアルタイム性を要求するタスクには規定の処理完了時間であるデッドラインが設定されており，リアルタイムシステムでは，システム上で実行される全てのリアルタイムタスクが各々のデッドラインを満たして正確に実行される必要がある．

リアルタイム性は時間制約の厳密性によりソフトリアルタイムとハードリアルタイムに分類される．

- ソフトリアルタイム
  ソフトリアルタイムでの処理完了を要求するタスクがデッドラインオーバーを起こした場合，その処理の完了によって実現される機能及びシステム全体の価値の低下に繋がる．一例として，携帯電話で電話帳の表示を行う際にボタン入力から画面表示までに規定以上の長いラグが生じた場合について考える．この例では，電話帳の表示の遅延により機能提供の円滑さの面でシステムの利便性が損なわれている．デッドラインオーバーにより処理完了までの遅延時間が長くなるに従ってユーザから見た機能の価値の低下は深刻化し，システム全体の価値の低下に繋がる．しかしながら機能の価値が低下する一方で，最終的に電話帳の表示が達成される場合にはシステム自体の破綻は起こらないため，時間制約の厳密性の点から見ればある程度寛容である．

- ハードリアルタイム
  ハードリアルタイムでの処理完了を必要とするタスクがデッドラインオーバーを起こした場合，システムの価値が完全に損なわれ，システム自体の破綻を引き起こす．前述の自動車のブレーキ制御システムがその一例である．ブレーキ動作に遅れが生じることで安全に関する信頼性を完全に損うため，一度の遅延でシステム全体の破綻を招く．以上により，ハードリアルタイムタスクはソフトリアルタイムタスクに比べて極めて高い厳密性の下でデッドラインを満たさなくてはならない．

本論文では，ハードリアルタイムタスクが動作するシステムを想定する．リアルタイム性の保証に必要な機能をシステムに導入する手段として，リアルタイム処理向けに最適化された OS である RTOS を用いることが一般的である．

2.3 組み込み分野での OS 利用

これまで複雑なコンピュータリソースの管理を必要としなかった組み込みシステムにおいても，近年ではデジタル化が進み，OS の搭載によるタスクやコンテキストの管理が一般化している．汎用システムで Windows や Linux 等の汎用 OS が用いられているのに対して，組み込みシステムでは iTRON 系や VxWorks，Android といった組み込み環境向けの OS が用いられる．これらの組み込み環境向けの OS の内，特にリアルタイムシステム上での動作を目的として作られた OS を RTOS と呼び，汎用 OS と比べて機能面で以下の差異を持つ．

- OS 機能の最小化
  組み込みシステムはメモリ容量を始めとする各種リソース面で厳しい制約を持つことから，組み込み環境で OS を動作させることを考えた場合，処理の負荷軽減や OS フットプリントの削減のために OS の機能についても取捨選択する必要がある．汎用 OS は様々な特徴を持つアプリケーションがシステム上で動作することを想定するため，OS 処理の実現に必要と考えられる機能やドライバを多目的に含んでいることが望ましい．一方の組み込みシステムは，アーキテクチャやその上で動作するソフトウェアに至るまで製品毎に特化されるため，実行されるタスクについてもあらかじめ想定されている場合が多い．そのため，動作する可能性のないタスクについては対応の必要がなく，OS の機能や搭載するドライバを絞り込むことが出来る．この機能削減により，RTOS は厳しいリソース制約を持つ組み込み環境に合わせて最小化される．加えて，RTOS は組み込みシステムの中でもリアルタイムシステムでの動作を提供する必要があるため，特有のタスク管理機能を提供する．

- タスクの周期実行への対応
  リアルタイムシステムにおけるタスク実行の形態は汎用システム

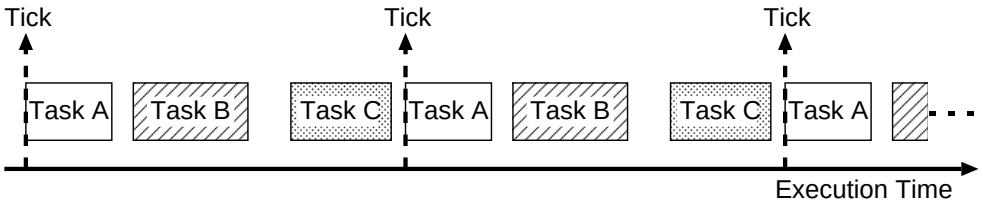

図 2.1: タスクの周期実行の例

とは異なる．リアルタイムシステム上で実行されるリアルタイムタスクは一般的に一定周期で実行されるという特徴を持つ．タスクの周期実行の例を図 2.1 に示す．周期タスクはタスク毎にあらかじめ設定されたタスク固有の時間周期毎に呼び出され，その都度デッドラインを満たして実行される．図 2.1 では，TaskA，B，C の 3 つの周期タスクが実行されている．図 2.1 中の Tick はリアルタイムシステム毎にシステム規模に合わせて設定されている時間単位である．システム全体が Tick 周期の下で動作し，各 Tick 周期の開始時にタイマ割り込みによって RTOS が呼び出される．RTOS は，呼び出される度に RTOS の持つタスク管理機能を実行し，次の Tick までのシステムのリアルタイム処理を保証する．本章ではスケジューリング処理の説明の容易性の観点から，Tick 周期に比べてタスク実行の時間スケールがそれぞれ十分に短い TaskA，B，C の動作を例に取る．

- 時間管理機能の提供
  リアルタイムシステムが時間情報を考慮した厳密なタスク管理を達成するためには，次の Tick までにシステムがリアルタイム性を満足するか否かを予測出来る必要がある．リアルタイムシステムは，システム上の全てのタスクが想定される最悪の処理完了時間 (以下，WCET) で実行された場合にもデッドラインを満たして正確に動作することを保証する．まず WCET での実行を考慮するため，RTOS は各タスクがあらかじめ持つ WCET 情報を元にタスクの実行順や実行タイミングを的確にスケジューリングする仕組みを持たなくてはならない．そのため，RTOS は時間情報の考慮が可能なタスクスケジューラをシステムに提供する．また，全てのタスクについてリ

アルタイム性を保証するため，tick 周期以外のタイミングであっても現行タスクの実行終了時や割り込みイベントの発生時等のシステム上のタスクの時間管理に変化が生じる場合にはタスクスケジューリング処理が行われる必要がある．

以上の特徴から RTOS はリアルタイムシステム向けに最適化される．ここで，RTOS によって提供されるタスク管理の具体的な処理内容について説明する．

2.4 リアルタイムシステムにおけるタスク管理

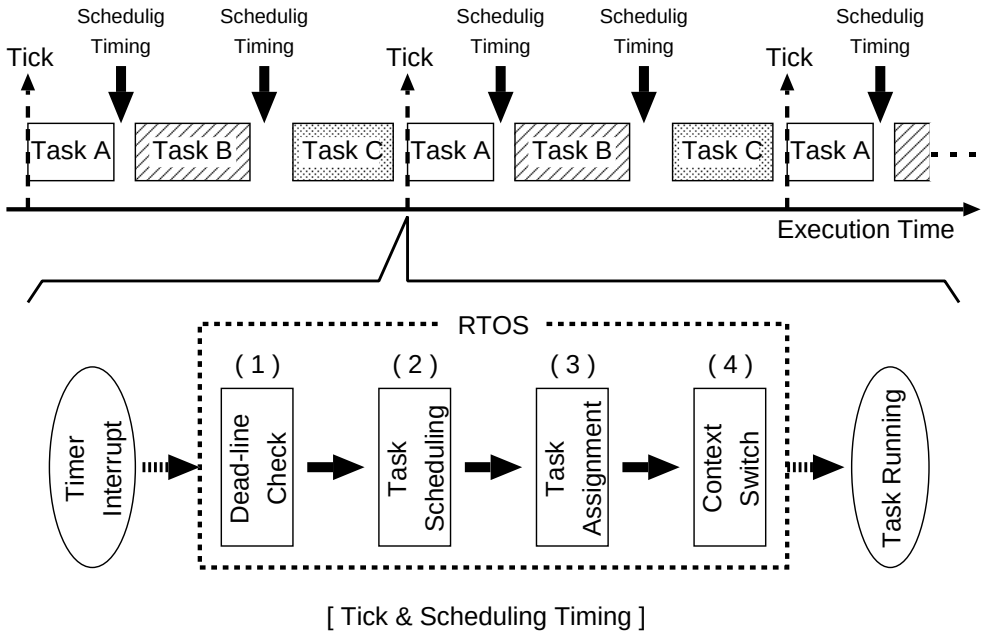

図 2.2: Tick 周期開始時の RTOS の処理ステップ

RTOS 呼び出し～タスク実行開始までの処理ステップを図 2.2 に示す．図 2.2 の（1）～（4）での処理内容は以下の通りである．

(1) デッドラインオーバーの判定

RTOS が割り込みイベントにより呼び出された際，まず始めに，次の Tick までにシステムがリアルタイム性を満足するか否かを判定する．各タスクの持つデッドライン及び WCET 情報を用いること

で予測を行う．

(2) タスクの優先度決定
リアルタイムシステムは一般的に優先度スケジューリングを行う．タスクの重要度に関わる指標を元に個々のタスクの優先度を設定し，その優先度に従ってコアへの割り当て順を決定する．リアルタイムタスクはデッドラインを満たすことが最優先されるため，優先度決定の際にはデッドライン情報が用いられる．

(3) プロセッサコアへのタスク割り当て
前ステップで決定した各タスクの優先度情報から，最高優先度のタスクをプロセッサコアに割り当てる．優先度スケジューリングを利用する RTOS は，優先度を元に格納情報の順番の入れ替えが可能な優先度 FIFO を持つことが多い．

(4) コンテキストスイッチ
スケジューラによってコアへ割り当てられたタスクの実行を開始する際，それまで割り当て先のコアで処理されていたタスクとの入れ替えを行う．このタスクの入れ替えに伴い，それぞれのタスクがコア内で利用していたデータコンテキストについても退避・復元処理が必要となる．これらの処理をまとめて実現する仕組みをコンテキストスイッチと呼び，タスクスケジューリングと同様に RTOS により機能が提供される．

2.5 タスクの時間情報の解析

リアルタイムシステムにおける時間予測性を満足するためには，システムへの RTOS の搭載だけでは不十分である．前節では，デッドラインオーバーの判定や時間情報を元にした優先度決定について述べる際，各リアルタイムタスクが自身の時間情報を持っていることを前提としていた．タスク毎のデッドラインに関してはシステム開発者の裁量により設定されるが，WCET 等の実行時間情報は事前にシステムに合わせた解析作業が必要となる．汎用システムではシステム上で実行されるタスクを動作以前に確定することが難しく，事前のタスク情報解析という手段は

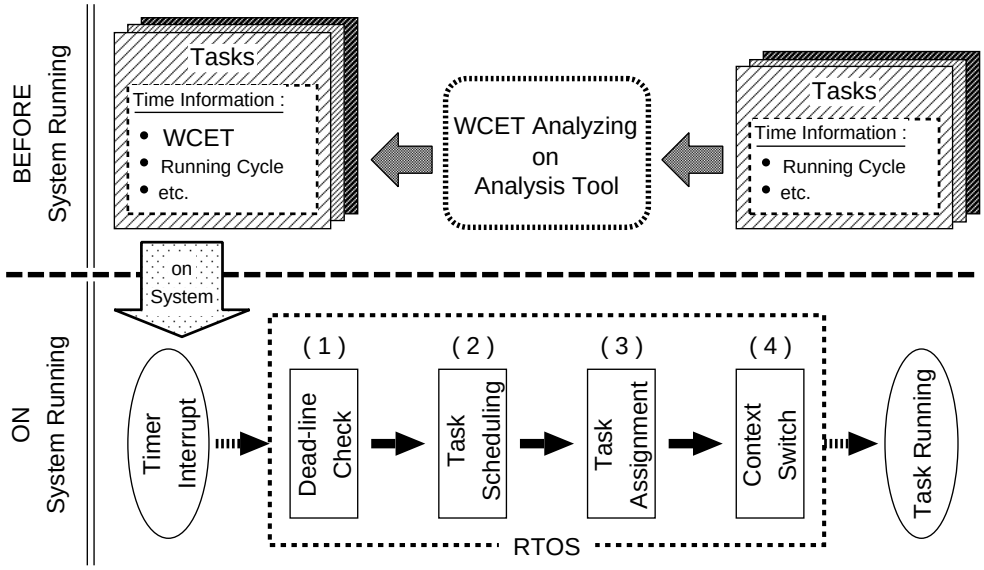

図 2.3: 時間解析による WCET 情報の提供

自然ではない．しかしながらリアルタイムシステムでは，前に述べたように動作するタスクがあらかじめ想定されている．そのため一般的には，図 2.3 に示す様にシステムを目的の環境で動作させる以前に解析を行い，WCET 情報を取得する．これにより，前節で述べたリアルタイム性の保証に伴う RTOS での時間管理処理が可能となる．解析の際には，同様のアーキテクチャ環境で且つ専用の解析ツールを利用する．単純な実行情報情報の一例では無く WCET を見積もる必要があることから，タスクの特徴やコアアーキテクチャの動作以外の要因 (メモリアクセス時間等) も含めた最悪の実行時間を算出する．また，処理時間が不定なアーキテクチャの動作が含まれる場合には時間予測が困難となるため，リアルタイムシステムの開発においてはハードウェア仕様の決定の際にも考慮が必要となる．

2.6 リアルタイムシステムの消費電力面での問題

以上により，RTOS を用いたシステム上では図 2.4 に示すような WCET 情報を考慮したタスクスケジューリングが実現される．図 2.4 は，あるリアルタイムシステムの 1 つのコアで，且つ 1 つの Tick 周期内で TaskA, B，C の 3 つのタスクが実行された例である．図 2.4 の (a) は RTOS によ

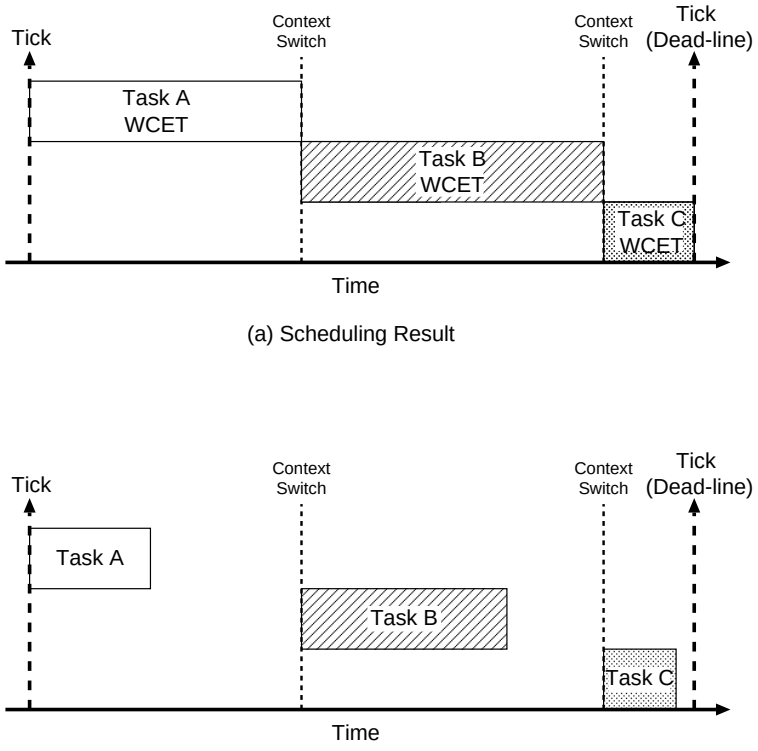

図 2.4: RTOS のスケジューリング例

るスケジューリングの結果であり，図 2.4 の (b) が実際に動作した際のタスクの状態を表している．3 タスク共に次の Tick 周期の開始タイミングがデッドラインとなっており，図 2.4(b) においてそれぞれデッドラインを満たしている．リアルタイム性の満足という点では図のスケジューリング結果は十分であるが，ここで，リソース利用の観点から図のスケジューリング結果を再度考察する．

図 2.4 の (a) と (b) の結果を比べると，Task A と B においてスケジューリング結果で想定している実行時間と実際のタスクの動作した時間との間に大きな開きがあることが分かる．これは先に述べたように，リアルタイム性を保証する上で必要となる WCET の見積もりにおいてメモリアクセス等の要因も考慮していることから，RTOS による時間予測が極めて悲観的になっているためである．リアルタイムシステムの目的は高速処理ではなく指定された時間内での処理完了であり，通常の組み込みシステムと同様に，ハードウェア仕様はシステム規模や要求レベルに合わせ

て最適化されることが望ましい．しかし，リアルタイムシステムのハードウェア仕様を決定する際には当然ながら悲観的な想定の下にアーキテクチャを用意する必要があるため，平均実行時間でシステムが動作する場合に図 2.4(b) のようにリソースに大幅な余剰が発生する．ハイエンドプロセッサの導入を始めとする高性能化への取り組みにより今まで以上に省電力性能が求められる現在の組み込みシステムにとって，この余剰なアーキテクチャの動作に伴う消費電力の増加は大きな問題であると考えられる．

そこで本論文では，この問題を解決するため，省電力アプローチの 1 つとして現在需要の高まっている単一 ISA のヘテロジニアスマルチコアプロセッサ (以下，単一 ISA HMP) に着目する．

3　単一ISAヘテロジニアスマルチコアプロセッサ

3.1　組み込み分野におけるマルチコアプロセッサ

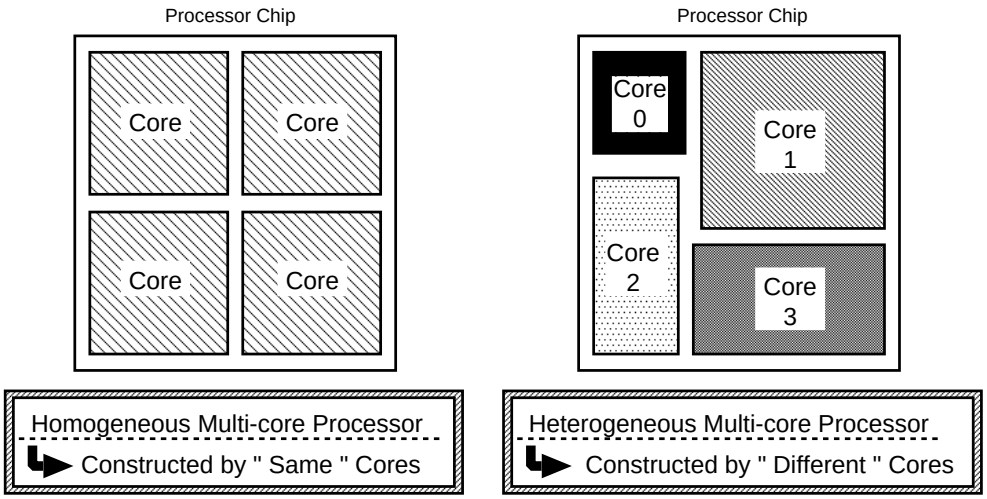

図 3.5: ホモジニアス構成とヘテロジニアス構成のマルチコアプロセッサ

近年の組み込みシステムの高機能化に伴い，組み込み分野においてマルチコアプロセッサの導入が進められている．マルチコア化はプロセッサの速度性能向上の手段として用いられ，汎用システムでは既にマルチコアプロセッサの搭載が一般的となっている．マルチコアプロセッサの構成方法は図 3.5 に示すホモジニアス構成とヘテロジニアス構成に大別される．

汎用システムに搭載されている Intel の Core i7 等のプロセッサはホモジニアス構成に分類される．ホモジニアスマルチコアプロセッサは一様なコアを並列に用いることで処理の高速化を実現する．ホモジニアス構成が汎用システムで採用されている理由として，様々なアプリケーションに対してそれぞれ十分な処理性能を発揮出来ることや OS を含むソフトウェア面で製品毎に専用的な実装を必要としない点が挙げられる．一方で，組み込みシステムの中でもリアルタイムシステムでは，プロセッサコアの並列動作によって生じる処理の非決定性等の要因から時間予測の観点で相性が悪い．また，個々のアプリケーションに対する最適化が行われていないため，アプリケーション毎の特徴や規模の違いに対応することが難しいというデメリットも持つ．

ヘテロジニアス構成は異なる特徴を持ったプロセッサコアを組み合わせることで構成され，各アプリケーションの特徴に合わせて最適なプロセッサコアを選択することで高効率なコンピュータリソース利用を実現することが出来る．デメリットとして，各コアアーキテクチャの命令セットや特徴の違いを考慮するために専用的なソフトウェア実装を必要とする点が挙げられる．汎用システムと比べてソフトウェアの特化や動作アプリケーションの特徴の絞り込みが容易な組み込みシステムでは，このヘテロジニアス構成が主に用いられている．これまで開発されてきた HMP は異なる ISA のプロセッサコアを組み合わせた構成が一般的であったが，現在，同一の ISA のプロセッサコアを用いた HMP が省電力アプローチの 1 つとして注目されている．画像処理や波形処理等の様々な用途に合わせて提供されている商用の組み込み向けプロセッサコアは，HMP を開発する上でが想定する．異種 ISA の HMP は実行されるタスクの特徴に合わせてアーキテクチャが特化されているため，タスク毎に実行されるプロセッサコアが完全に区別されている．一方，単一 ISA の HMP は同一の ISA で異なる性能を持つコアで構成され，現行タスクの負荷に合わせて動作させるコアを選択的に切り替えることでリソースの高効率利用を図る．以上より，現状で実用化されている異種 ISA と単一 ISA の HMP はそれぞれで省電力化のアプローチが異なる．そのため，本論文ではこれらの構成を明確に区別し，この内の単一 ISA の HMP を用いて前章で述べたリアルタイムシステムにおける問題の解決を図る．ここで，単一 ISA HMP の省電力化の原理について詳しく述べる．

3.2 単一 ISA HMP の省電力化の原理

図 3.6 にハイ/ローエンドの 2 つのコアを用いた単一 ISA ヘテロジニアスマルチコア構成のシステムの一例を示す．Processor Architecrute は High-end Core と Low-end Core の 2 つのコアと Memory により構成されている．実行される Task のコアへの負荷によって最適なコアを判断しアクティブにする．Task は OS のスケジューラから図の Switcher(一般的には Kernel や Hyperviser 上に実装される) を介してアクティブなコアに割り当てられる．図 3.6 の (a)，(b) はそれぞれ扱う Task 数が異なる場合のコアアーキテクチャの利用方法の違いを示している．標準的な処理量を扱う (b) の場合に比べて (a) はより多くの Task を実行することが要求されるため，コアへの負荷が大きくなると考えられる．円滑に処理を進めるた

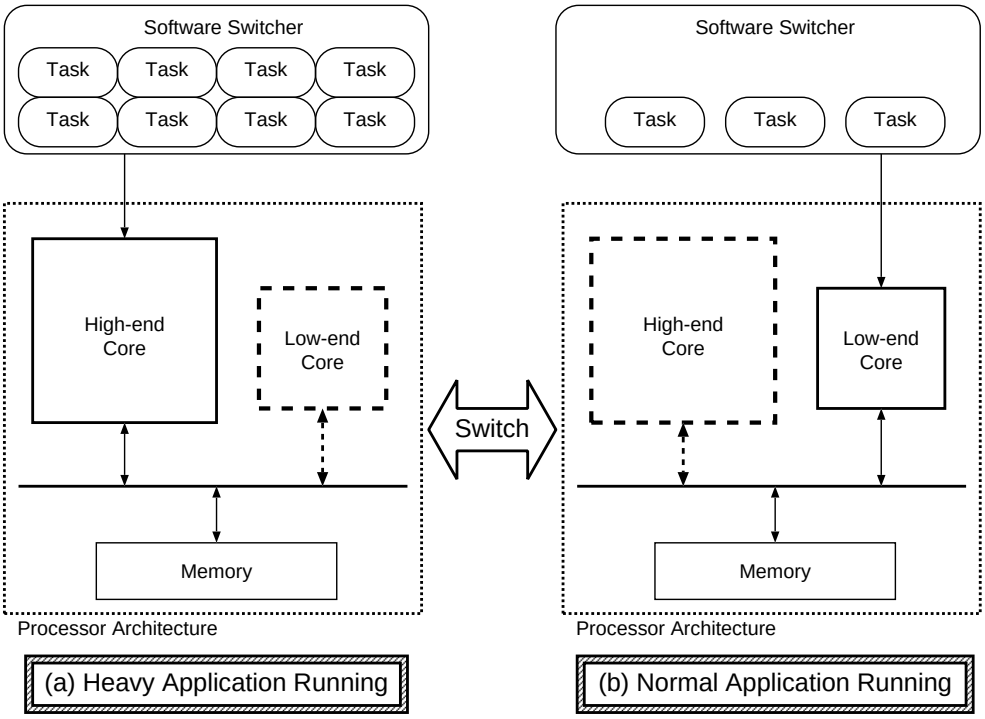

図 3.6: 単一 ISA ヘテロジニアスマルチコア構成の一例

めには十分なリソースを確保する必要があることから High-end Core がアクティブとなり，Switcher が現在アクティブの High-end Core へ Task を送ることで処理が実現される．一方で (b) の場合には，アクティブなコアと Swither のタスクの割り当て先が Low-end Core になっている．これにより，High-end Core に期待される処理性能を維持した上で省電力化を実現している．代表的な商用アーキテクチャとして ARM の big.LITTLE プロセッサが実製品に搭載されており，省電力アプローチとしての有用性が証明されている．また，図のようにタスクの最終的な割り当てを OS ではなく Switcher が行うことでハードウェア構造を隠蔽し，OS やユーザは (a)，(b) それぞれのアーキテクチャの変化を意識することなくシステムを利用することが出来る．

以上の省電力化の方法から，単一 ISA HMP がホモジニアス構成の利点であるソフトウェア開発の容易性とヘテロジニアス構成の利点である高い電力効率を両立していることが分かる．そのため，現状の用途としては，高い省電力性能を求められる組み込みシステムの中でも汎用的な

タスク処理を必要とする高性能携帯端末の分野が主である．しかしながら，最近のスマートフォンの普及や自動車内部の電子制御ユニットの統合等の話題から，組み込み分野において，動作アプリケーションのソフトウェア規模の増大や多様化が加速する傾向にある．この変化に伴うソフトウェア開発工程の負荷の増加から，今後，組み込みシステム開発の分野で単一 ISA HMP の需要がさらに増加すると考えられる．そこで本論文では，この単一 ISA HMP の利用によって，2.6 節で述べたリアルタイムシステムの消費電力に関する問題の解決を図る．

4 関連研究

可変性能プロセッサの利用によるリアルタイムシステムの省電力化を実現している研究として，負荷変動に瞬時適応可能なマルチパフォーマンスプロセッサ [2] が石原らによって提案されている．マルチパフォーマンスプロセッサは 1 つのプロセッサアーキテクチャ内に複数の演算要素 (以下 PE) を搭載し，それぞれ同一命令セットで異なる性能を持つ PE を選択的に利用することで処理性能の可変化を実現している．同時に動作する PE は 1 つだけであり，単一 ISA ヘテロジニアスマルチコアプロセッサにおいて現在主流の構成と同様に，ソフトウェア側からは 1 つの可変性能の演算処理ユニットが動作しているように見える．一方で，本論文で扱うヘテロジニアスマルチコア構成と異なる点として，マルチパフォーマンスプロセッサでは選択的に利用されるのは PE 部分のみであるため，キャッシュを共用により可変性能プロセッサにおける円滑なキャッシュ利用を可能としている．また，この共用のキャッシュについても連想度を動的に変更可能な構造となっており，コア毎にキャッシュサイズが最適化されているヘテロジニアスマルチコア構成と比べて遜色の無い省電力性能を実現出来る．

本研究の提案手法は，現状ではキャッシュ利用については考慮していない．また，扱うプロセッサアーキテクチャも完全に専用の実装が施された環境は想定していないため，上で述べたマルチパフォーマンスプロセッサと比較した場合にコアの性能可変化に伴うオーバーヘッドや省電力性能の面での優位性は多くない．しかしながら本研究では，今後需要が高まると考えられる単一 ISA HMP の中でも，さらに，既存の製品に搭載されているアーキテクチャ構成に近い仕組みのプロセッサを想定している．また，対象 RTOS の選定等の点も含めてシステムのソフトウェア開発における負荷に配慮しており，実システムへの導入における開発コストの面で本論文の提案手法は利点を持つ．また，コア切り替えのオーバーヘッドによる速度性能や省電力面での影響についてもコンテキスト管理用に搭載する専用ハードウェアにより緩和することが出来る．

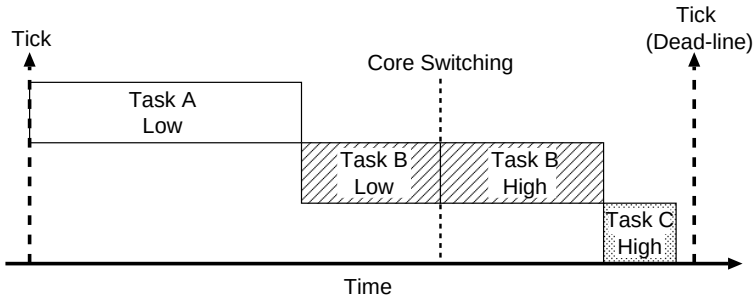

図 5.7: コア切り替えによるタスクスケジューリングの効率化例

5 提案手法

5.1 単一 ISA HMP のリアルタイムシステムへの適用

リアルタイム性の保証に伴う消費電力の増加を緩和するためにはより電力効率の高いプロセッサコアを利用してタスク処理を行うことが望ましい．しかしその一方で，リアルタイムシステムは WCET での動作に対しても対応が必要であることから，搭載するアーキテクチャの最大性能を下げることは出来ない．そのため本論文では，単一 ISA HMP のコア切り替えによる省電力化手法をリアルタイムシステムのタスクスケジューリングに適用することを考える．

図 5.7 は，図 2.4 の例と同性能のコアをハイエンドコアとして利用し，また，より小規模ながら高い電力効率を持つコアをローエンドコアとして搭載したハイ/ローエンドの 2 つのコアから成る単一 ISA HMP 上でのタスクの動作を示している．プロセッサアーキテクチャ以外の条件は図 2.4(b) と同様である．提案するシステムでは図 5.7 の実行結果例のように，タスク A，B はリアルタイム性の保証が可能な限りローエンドコア上で実行される．その後，処理の進行状況に合わせて最適なタイミング (図 5.7 ではタスク B の処理途中) でコア切り替えが行われることにより，リアルタイム性を維持した上で，図 2.4(b) と比較して高効率なリソース利用を可能にしている．

以上より，単一 ISA HMP のコア切り替えの仕組みをリアルタイムシステムに適用することで，図 5.7 に示したスケジューリングの最適化が理想的には可能である．しかしながら，現状で商用化されている単一 ISA HMP を搭載したシステムはリアルタイム処理向けに最適化されていない．

これは，単一 ISA HMP におけるコア切り替えの発生がリアルタイム性の保証に影響を及ぼすためである．本章では，単一 ISA HMP をリアルタイム処理向けに最適化する際の課題について述べた後，その解決案を示すことで単一 ISA HMP を利用したリアルタイムシステムの省電力化手法を新たに提案する．

5.2 提案手法の実現における課題

本節ではまず，単一 ISA HMP をリアルタイムシステムへ適用する際に想定される課題について詳細に説明する．本提案手法の実現の障害となる主な原因は単一 ISA HMP におけるコア切り替えの発生にある．前述の通り，リアルタイムシステムの保証に用いられる時間情報は基本的に固定時間で提供される必要がある．しかしながら，コア切り替えのもたらす時間的な要因がタスクの実行時間に影響を及ぼすことで，単一 ISA HMP 環境上ではこの条件を満たすことが難しくなる．本論文では以下の (1)〜(3) の課題について解決を図る．

(1) コアの切り替えに伴う WCET の変動への対応
リアルタイムタスクの持つ時間情報の中でも，WCET は図 2.3 で示した通り専用ツールを用いた解析によってシステムの実動作以前に取得される．システム稼働前にタスク毎の WCET を確定するという前提から，リアルタイムシステムでは実行時間予測が可能な範囲の処理でタスク実行を完了する必要がある．ここで，図 5.7 のシステムを例にとって考える．タスク A，C は処理完了までハイ/ローエンドコアのいずれかのプロセッサコア上でのみ動作しているため，それぞれ WCET 情報を取得可能である．これは，実行される全てのタスクに対してハイ/ローエンドの両プロセッサコア上での実行分の WCET を解析することで実現出来る．しかし他方のタスク B については，タスクの実行途中で動作コアの切り替えが発生していることから，ハイ/ローエンドどちらのプロセッサコアに合わせた WCET 情報を利用しても正確な WCET を求めることは出来ない．また，最適なスケジューリングの実現を考えた場合，事前にコア切り替えの発生タイミングを定めることも一方で困難である．以上より，本提案手法では，コア切り替えがタスクの実行途中に発生した際にもタスクの正確な WCET 情報を取得するための仕組みを提供

する必要がある．

(2) タスクの実行状態を考慮した動的な時間管理機構の必要性

上の (1) の問題が解決され，コア切り替えの発生タイミングに合わせて最適な WCET 情報が提供されたとしても，既存の RTOS のタスクスケジューラでは提案するスケジューリング最適化は実現出来ない．理由として，最適なコア切り替えタイミングの決定に動的な時間情報が必要となる点が挙げられる．ここで言う ”動的な ”時間情報とは，実際のシステムの動作状況に応じて値の変化する時間情報を指している．図 2.3 で示したシステムの稼働以前に決定されている時間情報が実動作の影響を受けない静的な特徴を持つことから，本論文では区別の方法として静的/動的という表現を用いる．動的な情報が必要となる場面として，タスクの実行状態に応じたコア切り替えの判定処理について述べる．提案手法では，より高効率なリソース利用を実現する上で，図 5.7 の例の様に可能な限りローエンドコアの利用時間を延長してシステムを動作させることが望ましい．この動作においてコア切り替えの延長が可能か否かを判定するためには，現行タスクの実行状態に関連する動的な時間情報を逐一加味し，且つ一定の時間間隔でタスク実行中に判定処理を行わなくてはならない．リアルタイムタスクの平均的な処理完了時間は RTOS が呼び出される Tick 周期と比べて十分に小さいため，処理に掛かるオーバーヘッドの観点から，これらの処理に RTOS の提供するソフトウェアのスケジューラを利用することは現実的でない．以上より，本提案手法では，最適なコア切り替えタイミングの決定に際して低オーバーヘッドの動的な時間管理機構を用意する必要がある．

(3) 固定時間且つ低オーバーヘッドでのコア切り替えの実現

プロセッサコアの切り替えの仕組みをリアルタイムシステムに導入する際の前提として，コア切り替えに掛かる時間についても固定時間でシステム開発者及び利用者に提示出来る必要がある．コア切り替えに伴うオーバーヘッドには切り替え先のコアへの移動が必要なデータコンテキスト量やデータ転送時間等が影響するため，処理に掛かる時間の見積もりは容易ではない．また，リアルタイムシステムの処理時間単位は汎用システムに比べて極めて微細であることか

ら，コア切り替えのオーバーヘッドがシステムのリアルタイム性能の低下を及ぼす可能性があることも考慮しなくてはならない．以上より，本提案手法では，固定時間且つ低オーバーヘッドでのコア切り替えを提供する必要がある．

5.3 解決案

5.3.1 解決方法の概要

本提案手法では，前節で挙げた 3 つの課題をそれぞれ以下の方法により解決する．

(1) コアの切り替えに伴う WCET の変動に柔軟に対応するため，タスク内部に命令数単位でチェックポイントを設定し処理単位を微細化

(2) タスクの実行状態を考慮した動的な時間管理機構を実現するため，タスクの実行状態の監視とその情報を元にした動的なスケジューリング処理を専用ハードウェア化

(3) 固定時間且つ低オーバーヘッドでのコア切り替えを可能にするため，(2) の解決案で示すハードウェアスケジューラ上にコンテキスト管理用に専用ハードウェアを搭載

以上の解決案により，本論文では，リアルタイムシステム上での最適なコア切り替えを可能とする動的スケジューリング機構を提案する．実行タスクの切り替えの際に用いる RTOS のタスクスケジューリングが現行タスクの実行状態を加味しない静的な特徴を持つことから，これに対して，提案するスケジューリング手法を” 動的 “スケジューリングとして以下では区別する．本論文で提案する動的スケジューリングを実現するための上の解決案の内，まず，(1) の解決案であるチェックポイントの設定について述べる．

5.3.2 チェックポイントの設定

コアの切り替えに伴う WCET の変動に柔軟に対処するためには，タスクの途中でコア切り替えが発生した場合の WCET もあらかじめ解析さ

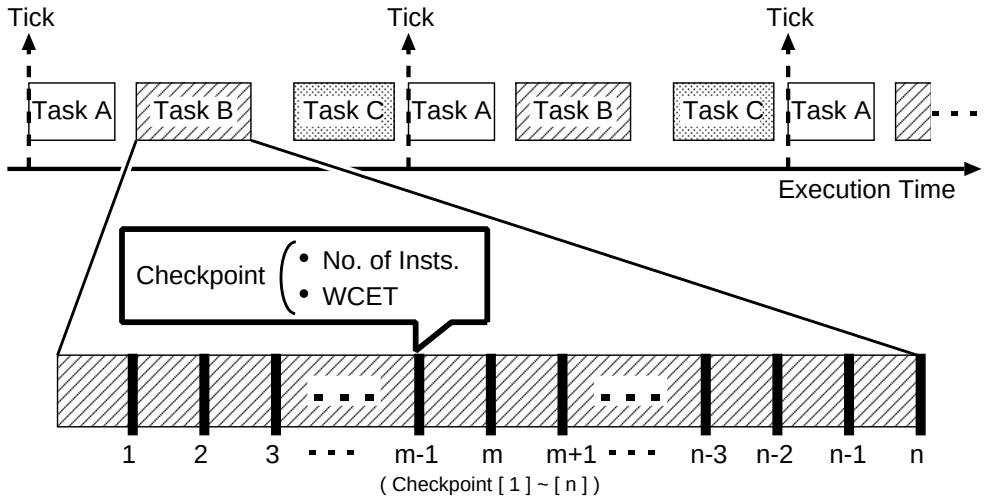

図 5.8: チェックポイント

れている必要がある．しかしながら，タスクの実行中に想定される全てのタイミングでのコア切り替えを予期して WCET を取得する方法は現実的ではない．そこで本論文では，図 5.8 に示す独自のチェックポイントをタスク中に設定する．これにより，リアルタイムタスクの処理時間単位を解析可能な範囲で細分化し，各チェックポイント時点での実行命令数 (No. of Insts.) と WCET に関する情報の提供する．コア切り替えをこのチェックポイント上で行うことで固定的な時間情報を取得出来るため，4.3 節 (1) の WCET の変動に関する問題が解決される．以上より，チェックポイントの設定は本提案手法の実現に有効であるが，2.5 節で述べた動作以前の時間情報解析の時点でタスクに独自の解析方法を適用する必要がある．WCET 解析ツールとして RapiTime や OTAWA 等があり，本提案手法では，これらの既存ツールの改良によってシステムでのチェックポイントの考慮を達成する．また，このチェックポイントの考慮の達成によって可能となる，動的スケジューリング処理を用いたコア切り替えタイミングの最適化について以下で述べる．

図 5.8 はタスク中に n 個のチェックポイントを設定した場合の例である．ここで，チェックポイントの導入によって実現される本提案手法の動的スケジューリングアルゴリズムについて説明する．1 から n 番目までのチェックポイントの内，m 番目のチェックポイントで行われる動的スケジューリングのスケジューリングアルゴリズムは図 5.9 の通りである．図 5.9 のア

ルゴリズム開始時点でタスクはローエンドコア上で動作している．現在のチェックポイント [m] から最悪の実行時間で次のチェックポイント [m+1] まで動作した場合に，チェックポイント [m+1] でデッドラインを満たすか否かを判定している．チェックポイント [m+1] からデッドラインまでの残り時間を算出し，次のチェックポイント [m+1] でハイエンドコアに切り替えたとしても最悪実行時間の動作でデッドラインをオーバーする場合には現在のチェックポイント [m] でコア切り替えを行う．

Dynamic Scheduling Algorithm

```
# 始めに，現行タスクの NoI(実行命令数: number of instructions) が
# Checkpoint[m](m 番目のチェックポイント) に到達しているか否か判定

if ((NoI of Running Task(※ 1)) == (NoI of Checkpoint[m])) {


  # Δ t[m+1]: Checkpoint[m] から Checkpoint[m+1] までの最悪実行時間

  Δ t[m+1] = (WCET of Checkpoint[m]) - (WCET of Checkpoint[m+1]);


  # T[m+1]: Checkpoint[m+1] まで実行した際の予測総実行時間

  T[m+1] = (Execution Time of Running Task(※ 1)) + Δ t[m+1];


  # R[m+1]: Checkpoint[m+1] からデッドラインまでの予測残り時間

  R[m+1] = (Deadline of Running Task) - T[m+1];


  # R_high[m+1]: ハイエンドコア (HC) 上で現行タスクを実行した場合の
  #            : Checkpoint[m+1] から処理完了までの最悪実行時間

  R_high[m+1] = (WCET on HC) - (WCET of Checkpoint[m] on HC)

                             + Core Switching Overhead(※ 2) ;
                             # コア切り替えに掛かる時間の考慮

  # コア切り替えの判定

  if (R[m+1] < R_high[m+1]) {
    Flag of Core Switching = 1;
  } else {
    Flag of Core Switching = 0;
  }

  m ＝ m ＋ 1;
}
```

図 5.9: 動的スケジューリングアルゴリズム

以上のアルゴリズムにより，最適なコア切り替えタイミングの決定が可能となる．しかしながら，図 5.9 中の (※ 1) の値は現行タスクの実行状態に関する情報であり，RTOS によるソフトウェアでの静的なスケジューリングでは 4.3 節 (2) の問題が発生すると考えられる．また，図 5.9 中の (※ 2) のコア切り替えのオーバーヘッドについても考慮がなされていないため，提案スケジューリング手法の実現に際して 4.3 節 (3) の問題も伴うと考えられる．これらの問題を解決するため，本論文ではチェックポイントの導入に加えて，動的スケジューリング処理のための専用ハードウェアの搭載を提案する．

5.3.3 ハードウェアスケジューラの機能概要

図 5.10 は本論文で提案するハードウェアスケジューラの各機能ユニットの概略図である．Dynamic Scheduling Unit は，前節で示したチェックポイントを考慮した動的スケジューリング処理と最高優先度タスクのコアへの割り当てをハードウェアで行う．Monitoring Unit はコアアーキテクチャ上の現行タスクの実行状態を逐一監視するために用いられる．Core Switching Unit はコア切り替え処理を専用的に実現するハードウェアであり，Dynamic Scheduling Unit〜コアアーキテクチャ間のインターフェースも担う．Scheduler Bus はメモリ用の共有バスとは別にスケジューラ専用に用いられるバスである．スケジューラ専用バスのプロトコルは AMBA[3] ベースでの実装を考えているが，コアスイッチの際にコアアーキテクチャ間の通信が発生するため，実ハードウェアでの実装に際しては最適なプロトコルの選定が必要である．以下より，ハードウェアによるスケジューリングの主な機能を実現している Dynamic Scheduling Unit と Core Switching Unit の 2 つの機能ユニットの内部処理について説明する．

まず，図 5.11 に示す Dynamic Scheduling Unit について述べる．Dynamic Scheduling Unit は 2 つのインターフェースを持つことにより，それぞれを介して動的スケジューリングに必要な情報を取得する．RTOS とのデータ転送のためのインターフェースは，ソフトウェアスケジューラで決定された情報を提供する．タスクの優先度情報を元に，ハードウェアで実装された優先度 FIFO を用いて最適なプロセッサコアへ最高優先度タスクを割り当てる．動的スケジューリングに必要な静的情報も同様にして RTOS から受け取り，専用のレジスタ (Task Info. Reg.) に格納する．また，他方のインターフェースは，スケジューラ専用バスを利用した円滑な通信を各コアに付随する Core Switching Unit との間で実現するために

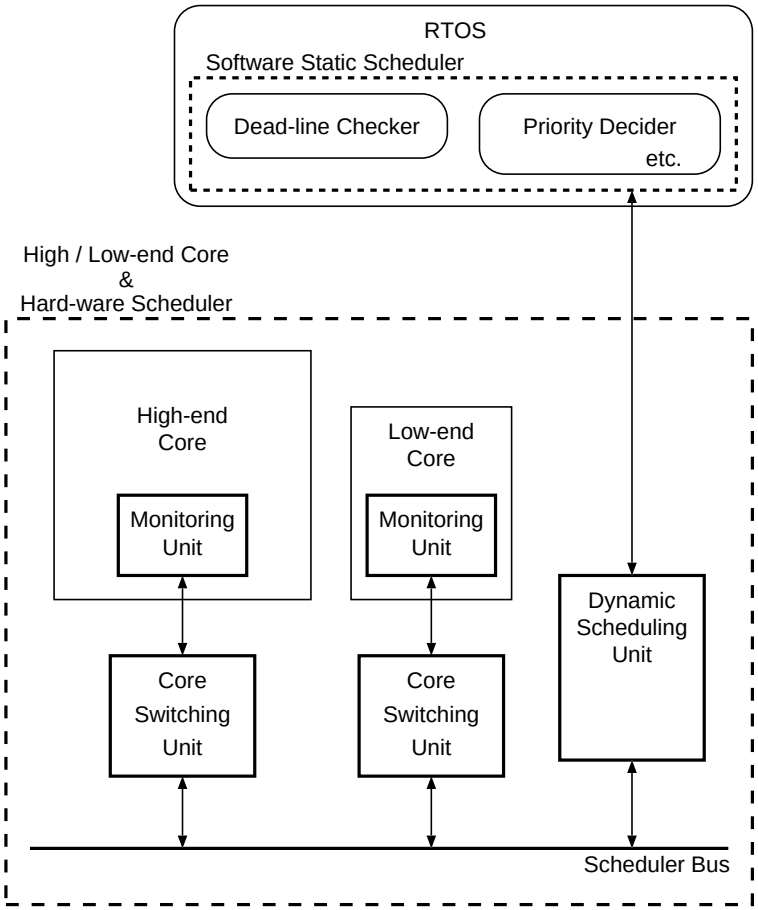

図 5.10: ハードウェアスケジューラの各機能ユニット

用いられる．Monitoring Unit が取得した現行タスクの実行命令数や実行時間等の動的な時間情報は Core Switching Unit 内部のカウンタに保持されていることから，Dynamic Scheduling Unit はこのインターフェースを利用することで動的な時間情報を受け取ることが出来る．その後，図 5.9 で示した動的スケジューリングアルゴリズムを組み合わせ回路で実現した Dynamic Scheduler ユニットに以上で取得した時間情報を渡すことで，目的の動的スケジューリングを実現する．動的スケジューリングの結果は，コア切り替え及び優先度制御のため，Core Switching Unit と RTOS，タスク優先度の制御を担う Priority Controler に送られる．4.3 節 (2) の問題の主な原因は，前述の通り，動的スケジューリング処理やそのために必要なハードウェア情報の監視がソフトウェアによるスケジューリング

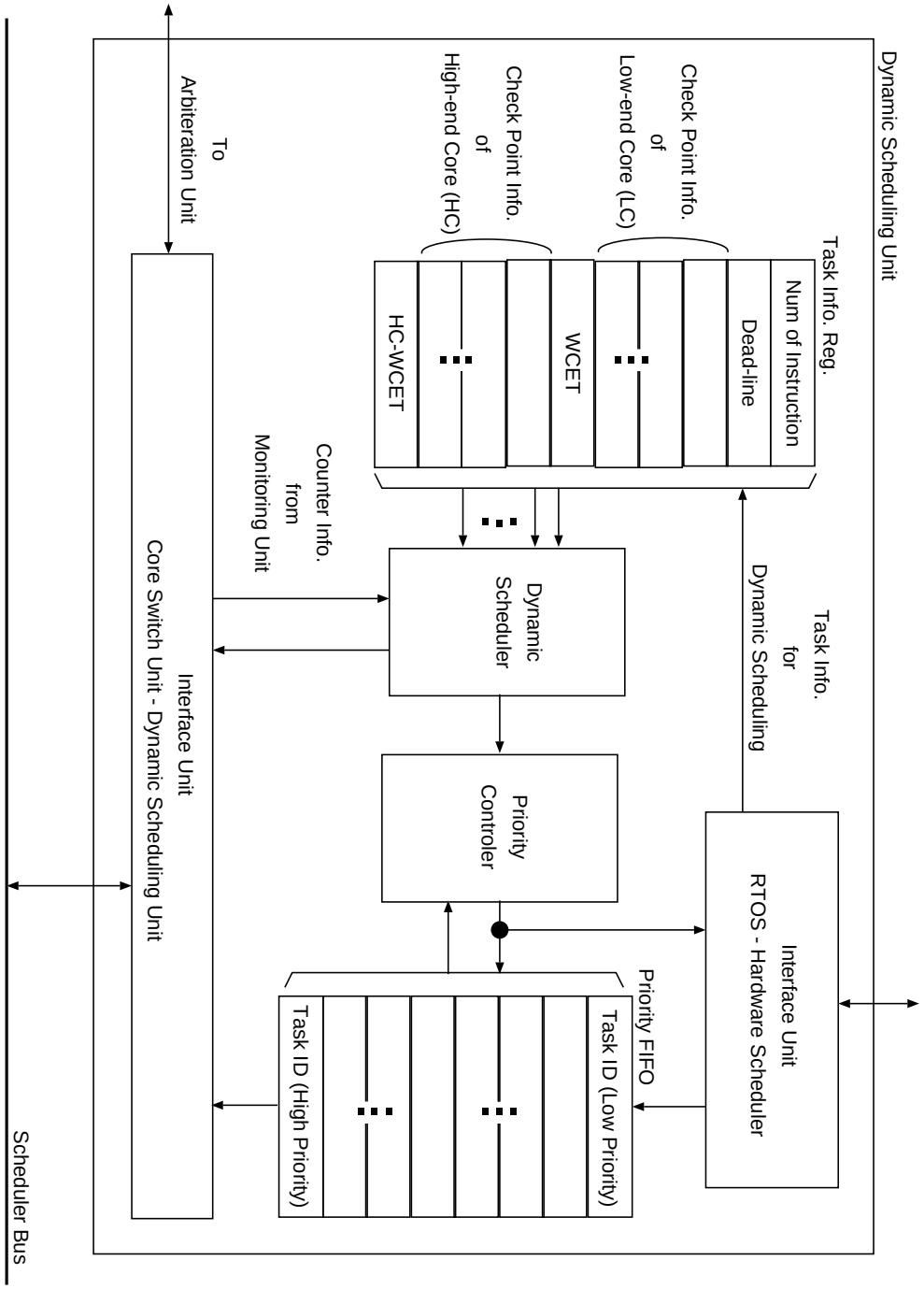

図 5.11: 動的スケジューリングユニット

では困難な点であった．以上より，本論文で提案するスケジューラ機構はそれらの処理に掛かるオーバーヘッドを専用ハードウェアで隠蔽することが出来るため，ソフトウェアスケジューラで懸念されていた動的スケジューリングのオーバーヘッドに伴う 4.3 節 (2) の問題を解決する．また，同様に 4.3 節 (2) において問題視されていたタスクの実行状態の監視についても Monitoring Unit によって実現される．

次に図 5.12 に示す Core Switching Unit について述べる．Core Switching Unit は，4.3 節で挙げた課題の内，残りの (3) に該当するコア切り替えのオーバーヘッドの問題を解決するための機構である．また，コア切り替えの発生時以外は，Dynamic Scheduling Unit と Monitor Unit 及びコアアーキテクチャ間の仲介の役割を担っている．タスクとそれに付随するコンテキストをコアへ割り当てる際に必要となるレジスタを持ち，一方で Monitoring Unit で取得した動的情報を保持するためのカウンタも持つことで，Dynamic Scheduling Unit 側とコアアーキテクチャ側のそれぞれに対して円滑なデータ供給を実現する．加えて，Core Switching Unit は他のコアアーキテクチャの Core Switching Unit へのインターフェースも持つ，これにより，コア切り替えの際にコアアーキテクチャ間での直接のデータ送受信が可能となるため．先で述べたコア切り替えのオーバーヘッドに関する問題を解決することが出来る．コア切り替えの制御は Core Switching Controler が行う．メモリバスと用途が分離され，且つ転送するコンテキスト量を想定して十分な帯域で実装されたスケジューラ専用バスを利用することで，固定時間で低オーバーヘッドなコア切り替えを提供する．

本章では，リアルタイムシステムへのコア切り替え機構の導入における課題に対してそれぞれ解決案を示した．これらの解決案により，提案手法は単一 ISA HMP のリアルタイム処理への適用を達成出来ると考えられる．本研究では最終的に，提案したアプローチを実システム上に実装し，高効率なコンピュータリソース利用を可能にする省電力リアルタイムマルチコアプロセッサを実現する．本アプローチの有用性の証明のため，本論文では，同等の機能を持つソフトウェアエミュレータを C 言語で実装した．以下より，実装エミュレータを用いた有用性検証の方法とその結果について述べる．

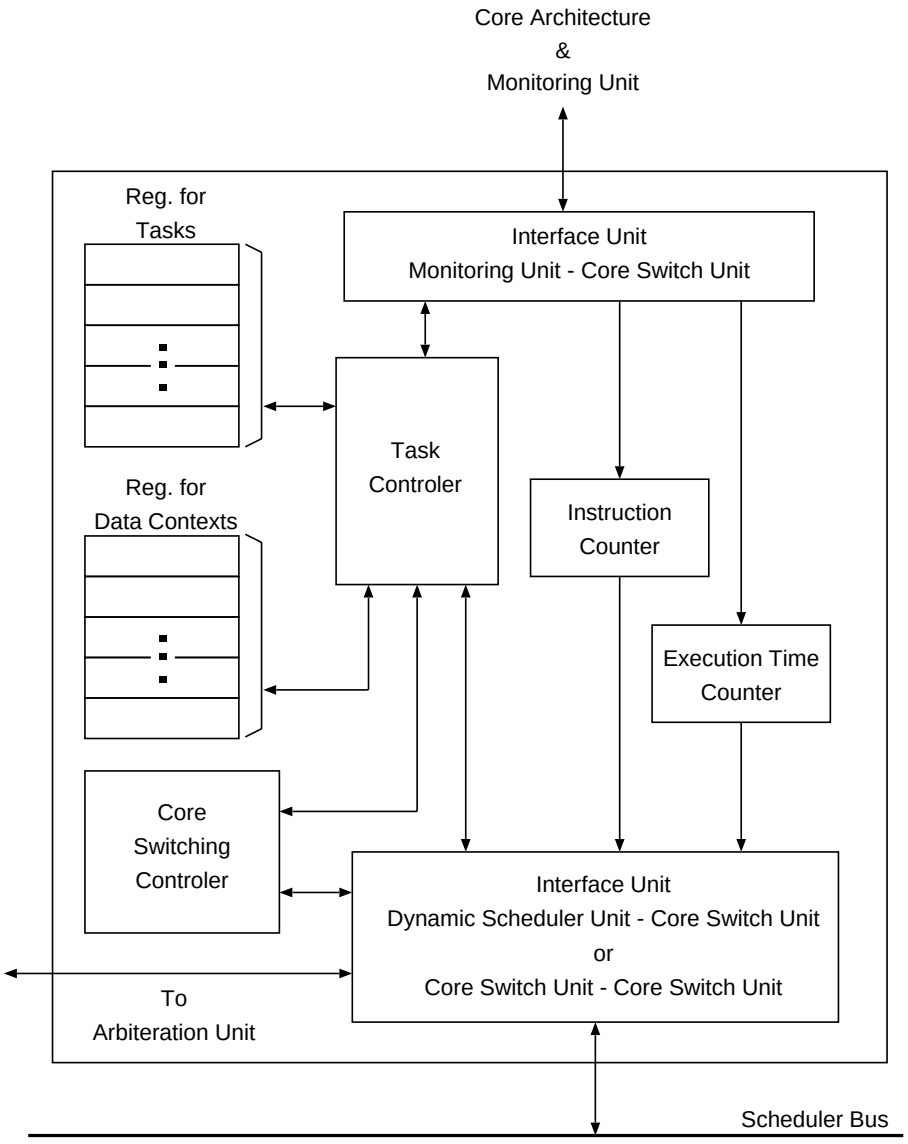

図 5.12: コアスイッチングユニット

6　提案手法の有用性の検証

6.1　評価環境の構築

本提案手法の有用性を検証するために，プロセッサシステムエミュレータである QEMU[4] 上に提案スケジューラのエミュレーション環境を C コードで実装した．QEMU はプロセッサシステム全体の機能をエミュレーション可能な高機能エミュレータである．Bellard 氏らによって開発されており，現在オープンソースで公開されている．本論文で構築したマルチコアプロセッサエミュレーション環境の仕様を表 6.1 に示す．表 6.1 の環境上で独自に実装したベンチマークパターンを実行することで，提案スケジューリング手法の省電力効果について検証を行った．

表 6.1: 構築した検証用エミュレーション環境の仕様

Test Processor	
Instruction Set Architecture	PowerPC
Processor Model	MPC8544DS
Memory	1024 [MB]
RTOS	RTAI
Number of Cores	2
Core0 (Low-end Core)	
CPU	e500v2
Clock Frequency	500 [MHz]
Cache	Nothing
Core1 (High-end Core)	
CPU	e500v2
Clock Frequency	2.0 [GHz]
Cache	Nothing

本検証において単一 ISA HMP 環境を構築する際に，QEMU がヘテロジニアスな構成に未対応であるという点が問題であった．本論文では表 6.1 に示すように，各コアの動作周波数に差をつけることで異性能コアを持つ単一 ISA HMP の動作をエミュレーションした．表 6.1 の Core0 の動作周波数は高い電力効率持つコアアーキテクチャを想定しており，ロー

エンドコアとしての利用に適した値である．また，一方の Core1 は，リアルタイムシステムの WCET での動作に必要な最高性能を持ったコアアーキテクチャを想定しており，ハイエンドコアとしての利用に適した値である．

以下では，RTOS へのリアルタイムクロック供給や上で述べたコアアーキテクチャの動作周波数の考慮のために必要となる QEMU 上での擬似的なクロック供給の方法について説明する．その後，本検証における対象 RTOS である RTAI[5] の選定理由と実装したベンチマークパターンの仕様についてそれぞれ記述し，最終的に得られた検証結果について考察する．

6.2 エミュレータ上でのクロック供給

本検証で用いる QEMU はエミュレーションする環境上での正確なクロック動作を提供していない．ここで言う正確なクロック動作とは，エミュレーションしているプロセッサが自身で固有の時間単位を持ち，その時間単位を元にしたクロック供給により動作することである．QEMU がエミュレーションするプロセッサシステムは QEMU が動作するホスト環境のクロックを元にした時間単位で動作するため，同環境に本論文の提案スケジューラのエミュレーション機構を実装した場合，ホスト環境の仕様や負荷状態の違いによる検証結果への影響が問題となる．RTOS へのリアルタイムクロック供給やコアアーキテクチャの動作周波数の考慮の必要性から，本論文の検証に用いる環境は正確な時間単位の元に動作することが望ましい．

そこで本論文では，提案スケジューラのエミュレーション環境の実装に加えて，正確なクロック供給を行うための対応も行った．実装したエミュレータ内の時間単位は，QEMU がエミュレーションするプロセッサコアの実行状態を元に設定した．空ループ処理の実行開始から終了までを 1 時間単位とし，この時間単位を元に RTOS へのリアルタイムクロック供給やコアアーキテクチャの動作周波数の設定を行った．プロセッサコアの実行状態の監視と時間単位の管理機構は，図 5.10 に示した Monitoring Unit を持つ提案スケジューラのエミュレータ機構内部に実装した．(以下，本検証環境で設定した時間単位から換算される 1 秒を現実の時間単位と同様に ”s ”と表記する．)

6.3 RTAI

本検証では，以下の 3 つの理由から対象の RTOS として RTAI を採用した．

- Linux の開発資源の利用可能性
  単一 ISA HMP を搭載したシステムは異種 ISA の同等の構成のシステムと比べてソフトウェア開発におけるコストの面で優位性を持つ．本論文では，この単一 ISA HMP のシステムの利点に加えて Linux ベースの RTOS である RTAI を対象の RTOS として採用する．将来的に本提案手法を導入したシステムを利用するソフトウェア開発者に対して，現行の LinuxOS の持つ豊富なソフトウェア資源や開発環境の提供を実現出来る．これにより，単一 ISA HMP 上で動作するソフトウェアの開発の余地を広げ，近年の組み込みシステムにおける動作アプリケーションの規模の増大や多様化への需要に対応する．

- マイクロカーネル方式での動作
  RTAI は図 6.13 に示すマイクロカーネル方式で構成され，汎用の LinuxOS にマイクロカーネルと呼ばれる機能を絞ったカーネルを提供することで RTOS の機能を実現する．RTAI では本論文で実装を行うリアルタイムタスク管理に関する機能が全てマイクロカーネル部に集約されており，汎用的機能を持つ LinuxOS 部分と明確に分離されている．そのため実装範囲の切り分けの面で優れており，且つ主にソフトウェア開発者やシステムユーザの触れる LinuxOS 部分については特殊な改良を必要としないという利点を持つ．図 6.13 中では，RTAI の機能の内，本研究で専用ハードウェア化する機能部分についても記述しており，LinuxOS まで実装範囲が及んでいないことが分かる．

- オープンソースである
  上記の 2 つの利点が合致する RTOS として RTAI 以外に RTLinux[6] が挙げられる．RTLinux は商用製品への採用も行われたことから十分なリアルタイム性能を持った RTOS であると言える．しかしながら，実製品採用の際に有償ライセンス版が主流となったため，オー

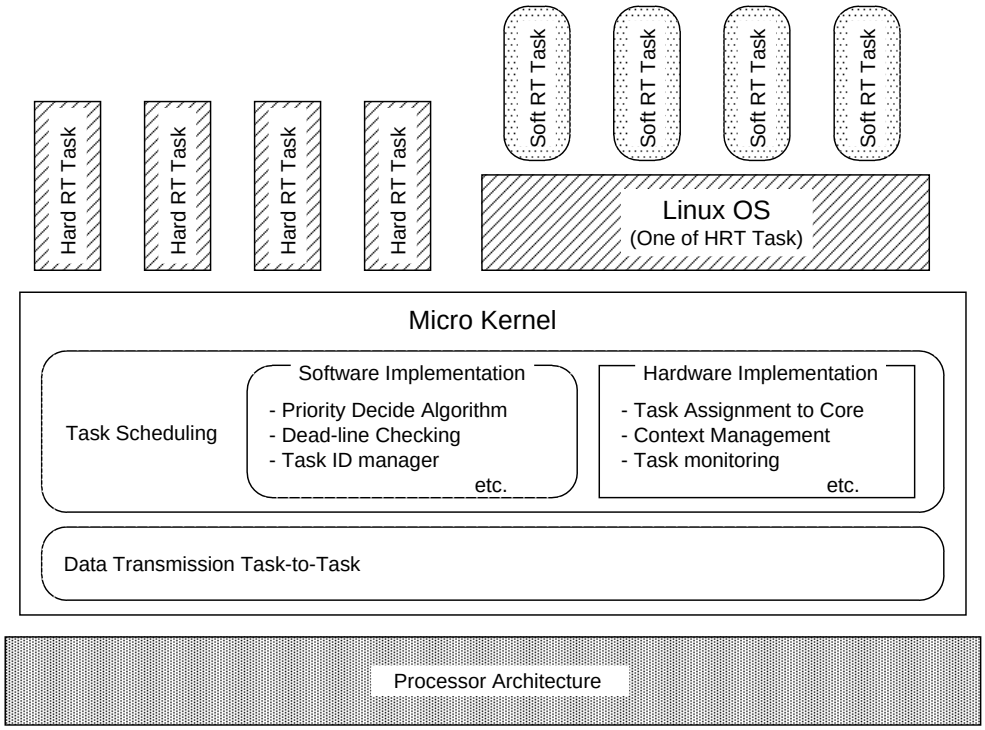

図 6.13: RTAI の構造

プンソースで提供されている FreeRTLinux では近年のコンピュータシステムの変化への対応が十分でない．本研究で対象としているプロセッサ環境である単一 ISA HMP はコンピュータ分野における需要の中でも目新しい方策であるので，オープンソースで且つ最近のシステム環境への対応の点で優位であった RTAI を採用した．

6.4 評価用ベンチマークの実装

本節では，検証の際に利用したベンチマークプログラムの特徴と実装方法について説明する．ベンチマークを用いた動作検証では，一般的に，利用するベンチマークがタスクの様々な動作パターンや用途別の特徴に対して網羅的に対応していることが望ましい．しかしながら，本論文で想定している単一 ISA HMP の環境はコンピュータの分野の中でも比較的新しいプロセッサ構成方法である．加えて，リアルタイムシステムへの適用は本研究による新規のアプローチとなるため，本提案手法の有用

性検証に適したベンチマークは現状の組み込みシステム開発の分野では提供されていない．そのため本論文では，検証に必要なベンチマークソフトウェアについても独自に実装している．

本論文で実装したベンチマークは，実装に伴う作業量の観点から簡単化されている．元々，組み込みシステムで動作するタスクは汎用システムで動作するタスクに比べて小規模であったため，利用されているベンチマークにおいても数千命令から，極めて小規模なものでは数百命令レベルのタスクを含んでいた．しかしながら現在では，70 万〜100 万命令を超えるタスクを想定したベンチマークも一般的に用いられている．提案するスケジューリング手法では最近のプロセッサシステムを対象としているため，動作させるベンチマークについてもある程度の規模を想定する必要がある．また，本論文における実装内容はハードウェアスケジューラに関する部分が主であることから，チェックポイントについても各タスク毎に手作業での設定が必要となる．本論文では，以下の方策によりベンチマーク実装を簡単化した．

本検証のために実装したベンチマークプログラムは空ループと明示的なデータアクセス処理のみで構成されている．システム毎で設定した単位時間の間に実行される空ループ回数やデータアクセス回数を元にベンチマークのタスクの負荷量を設定し，また，呼び出し周期や実行時間の設定時に他のタスクと差異をつけることでタスク毎の処理内容の差別化を行っている．実際の組み込み環境での処理を想定してベンチマークを実装をした場合，個人の実装ではあらゆるタスクのパターンに対処することが難しく，一方で，少量のパターンのベンチマークでは検証結果に偏りが生じる可能性が考えられる．その点，本検証のために実装したベンチマークプログラムは空ループ回数やデータアクセス回数，呼び出し周期，実行時間の考慮のみで容易にベンチマークプログラムを実装出来る．また，本検証では提案スケジューリング手法によるリソース利用の効率化が実現されていることが結果の時間情報として確認出来れば良いため，検証結果の面から考えても，上記の空ループ回数，データアクセス回数，呼び出し周期，実行時間の要素の考慮のみで十分である．加えて，処理時間を決定する要素が絞られていることから，プログラム実装時点での処理時間見積もりやチェックポイントの設定の面からも有用な方法であると言える．

本検証で実装したベンチマークパターンは表 6.2 の 3 パターンである．タスク数は 1ms の Tick 周期で周期実行可能な量を想定している．表 6.2

の項目はそれぞれ各パターンで実行されるタスク数，タスク毎の命令数，タスク毎の実行時間，全体の実行命令数の内の明示的なデータアクセス命令の割合いである．(a) のパターンは，1 つのタスク辺りでそれぞれ中程度の負荷のタスクの集合を想定している．また，タスク毎の時間スケールの差も少なく，最大で 10 倍に収まっている．(b) のパターンは，低負荷で且つ短い実行時間のタスクを大量に処理するシステムを想定した．(c) のパターンは，データアクセスが多く，時間周期に対して規模の大きいタスクが動作する場合を想定している．

表 6.2: 実装したベンチマークパターンの仕様

	Pattern (a)	Pattern (b)	Pattern (c)
No. of Cyckle Tasks	30	120	12
No. of Insts.	10k〜100k	0.5k〜10k	10k〜300k
Execution Time	10 μ〜100 μ [s]	1 μ〜10 μ [s]	50 μ〜1m[s]
Data Access	1 %	0.05 %	5 %

6.5 検証結果

前節のベンチマーク (a)〜(c) を利用して，評価用プロセッサエミュレータ環境上で本論文で提案したスケジューリング手法が正確に行われていることを確認した．Tick 周期は 1ms で，コア切り替えのオーバーヘッドは理想的に Tick 周期の 1000 分の 1 の 1 μ s を想定している．スケジューリング結果の一例を図 6.14 に示す．

表 6.3 は，RTOS によるスケジューリングと提案スケジューリング手法を用いた際のそれぞれのハイ/ローエンドコアの稼働時間の比較である．また，各プロセッサコアの稼働時間の変化による消費電力の変化の見積もり結果も示している．ここで示している消費電力は，各コアで設定した動作周波数に合わせて同等の規模のプロセッサの平均消費電力 [7] をそれぞれのコアの稼働時間に掛け合わせることで見積もった．この結果から，本提案手法を用いたスケジューリング効率化により，(a) のベンチマークパターンにおいて，タスク実行時のみで約 28%の消費電力削減を達成していることが確認出来た．(b) のベンチマークパターンはタスク辺りの実行

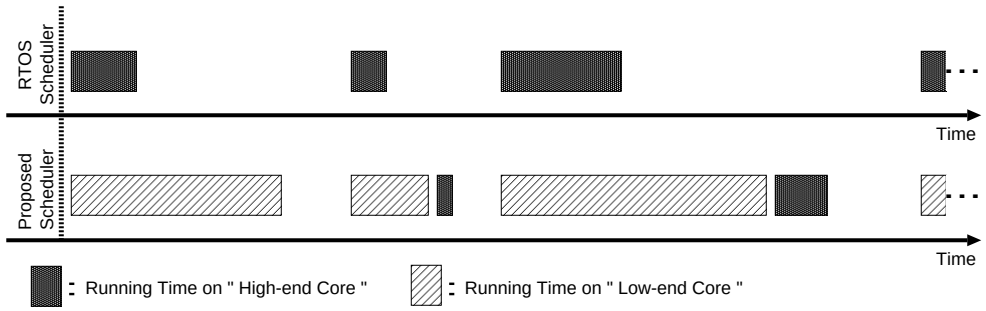

図 6.14: 検証用エミュレータ環境上での提案スケジューラの動作

表 6.3: 各ベンチマークパターンでのハイ/ローエンドコアの動作時間と消費電力の比率の変化

	High-end	Low-end	Power Consumption
RTOS	100 %	- %	100 %
(a)	9 %	91 %	72 %
(b)	13 %	87 %	87 %
(c)	7 %	93 %	69 %

時間が短いことから (a) と比較してデッドラインに余裕が無くなるため，ローエンドコアをコア切り替えの発生が早まり，電力効率が下がっている．一方で，(c) のベンチマークパターンは (a) と比較してデッドラインに大きな差はないが電力効率が上がっている．これは，(c) の方がタスク辺りのデータアクセス命令が多いことから，メモリアクセスのオーバーヘッドにより 2 コア間の動作周波数による性能差が緩和されたためだと考えられる．

7　おわりに

7.1　まとめ

本論文では，ハードウェアスケジューラを利用した省電力リアルタイムマルチコアプロセッサを提案した．前章の検証結果から，提案したタスクスケジューリング手法を利用することにより，中程度の負荷のタスク実行時で約 28%の消費電力の削減が見込めることが分かった．また，(a)，(b)，(c) の各ベンチマークパターンによる省電力性能の変化も期待通りの動きであることから，目的としたアーキテクチャ動作が達成出来ていることも確認出来た．本論文における検証ではアイドル状態時のプロセッサの消費電力等の考慮がされていないため，実ハードウェアでの実装においてさらなる省電力性能の達成が期待出来ると考えられる．

7.2　今後の展望

今後の課題として以下のことが挙げられる．

- 提案スケジューラのハードウェア記述言語による実装
  本研究では，提案したハードウェアスケジューラを最終的に実チップ上に実装することを考えている．そのため，将来的には本提案スケジューラをハードウェア記述言語 (以下，HDL) を利用して実装する必要がある．HDL で実装することで実ハードウェアに近い環境での詳細な電力評価を可能にし，省電力アプローチとしての本提案スケジューリング手法の有用性をより信頼性の高い環境で評価することが出来る．この HDL による実装のためには，ハードウェアスケジューラの詳細設計に加え，評価用ベンチマークの充実やチェックポイント設定のための既存の WCET 解析ツールの改良等の課題がある．また，HDL で記述した提案スケジューラを評価するための環境として，同様に HDL で実装されたプロセッサシステムを用意する必要がある．本研究では，実際の組み込みシステムを想定した評価環境として，既存のプロセッサである FabScalar[8] と SakuraProcessor[9] 用いて単一 ISA ヘテロジニアスマルチコアプロセッサを構築することを考えている．FabScalar は，現在ハイエンドなコンピュータシステムで広く用いられているスーパースカラ構成のプロセッサを SystemVerilog(HDL) コードで自動設計するツー

ルセットである．HMP の開発における設計・検証の時間的なネックの解消を目的として N.K.Choudhary らによってノースカロライナ州立大学で開発されている．FabScalar により，今後組み込み機器に搭載される可能性のある様々なスーパースカラ構成のハイエンドプロセッサを想定した評価が可能となる．SakuraProcessor は一般的な組み込み環境での利用を想定して実装されたプロセッサである．5 段パイプラインのシンプルな構成となっており，本稿で想定している省電力用のローエンドコアとして最適な性能となっている．また，FabScalar と同様に SystemVerilog(HDL) コードで記述されていることから，本研究で想定するローエンドコアに適していると考えられる．

- よりハイエンドなプロセッサ構成への対応
  組み込み分野においてハイエンドプロセッサの搭載が進んでいることは先に述べたが，本論文で扱った単一 ISA ヘテロジニアスマルチコア構成と同様に対称型のマルチコアプロセッサや高性能キャッシュの要求も高まっている．これらの機構の導入はタスクの実行完了順の非決定性や共有バスの競合，キャッシュミス等の要因からリアルタイム性の保証の面で大きな障害となる．反面，リアルタイム性の保証を優先した場合には処理性能や省電力性能の悪化を避けることが出来ず，これらの問題についても解決が必要であると考えられる．よりハイエンドなプロセッサ構成を対象にした省電力化手法の実現のため，将来的には本論文で提案したハードウェアスケジューラをさらに拡張し，コアだけでなくバスやキャッシュ等の部分の改良についても視野に入れる必要がある．

謝辞

本研究の機会を与えて頂いた近藤利夫教授，並びにご指導，ご助言頂いた佐々木敬泰助教に深く感謝いたします．また，様々な局面でご助力頂いた計算機アーキテクチャ研究室の皆様にも心より感謝いたします．

参考文献

[1] ARM Cortex-A15 MPCore Processor Technical Reference Manual
http://www.arm.com/

[2] T. Ishihara, S. Yamaguchi, Y. Ishitobi, T. Matsumura, Y. Kunitake, Y. Oyama, Y. Kaneda, M. Muroyama, T. Sato. AMPLE: An Adaptive Multi-Performance Processor for Low-Energy Embedded Applications. the 6th IEEE Symposium on Application Specific Processors (SASP 2008), pp.83-88, June 2008.

[3] ARM: AMBA Specification(Rev2.0)
http://www.arm.com/

[4] QEMU User / Technical Documentation
http://wiki.qemu.org/Manual

[5] RTAI: Real Time Application Interface
https://www.rtai.org/

[6] V. Yodaiken. The RTLinux Manifesto. In the Proceedings of the 5th Linux Expo, March 1999, in Raleigh North Carolina.

[7] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, D. M. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, p.81, December 03-05, 2003.

[8] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, E. Rotenberg. FabScalar: Composing Synthesizable RTL Designs of Arbitrary

Cores within a Canonical Superscalar Template. Proceedings of the 38th IEEE/ACM International Symposium on Computer Architecture (ISCA-38), pp.11-22, June 2011.

[9] T. Sugiyama, T. Sasaki, T. Nakabayashi, T. Kondo. Development of C++/RTL Co-simulation Environment for Accelerating VLSI Design of An Embedded Processor. The 28th international Technical Conference on Circhuits/Systems, Computers and Communications (ITC-CSCC 2013), Yoesu, Korea, July 1, 2013