修　士　論　文

無線センサノード向け
機能分散型マルチコアCPU

**A Functionally-Distributed Multi-Core CPU
for Wireless Sensor Nodes**

指導教員　**森川 博之 教授**

**東京大学大学院工学系研究科
電気系工学専攻**

氏　名　**37-086470　大原 壮太郎**

提 出 日　**2010年2月8日**

目次

図目次

表目次

第１章

序論

1.1 研究の背景と目的

無線センサネットワークのコンセプトが Smart Dust[1] により示されて 10 年以上の歳月が経過した．J. M. Kahn らが提唱した Smart Dust は，MEMS（Micro Electro Mechanical System）技術により数ミリ四方に収められたセンサノードを環境上に散布し，センサ情報を無線通信により収集するというコンセプトを持つ．以降，Smart Dust の斬新なコンセプトを実現すべく，ネットワーク技術，ハードウェア構成技術，オペレーティングシステム等の基盤技術の研究が進められてきた．現在ではこれらの基盤技術の組み合わせにより実験レベルのアプリケーションを構築することが可能となり，アプリケーションの研究が多く行われるようになってきた．

アプリケーション構築に際して，センサノードは以下の 2 つの要件を満たさなければならない．1 つ目の要件は，すべてのタスクの時間制約を満たすことである．無線センサノードがタスクの時間制約を満たせない場合，処理落ちによるパケットロスや，サンプリングジッタといった問題が発生し，有用性を著しく損ねる．2 つ目の要件は省電力性である．無線センサノードは多くの場合バッテリなどの独立電源で駆動するため，消費電力の削減は避けることのできない課題である．

しかしながら，現在のシングルコア CPU のシステムで省電力性を確保したまま，タスクの時間制約を満たすことは困難である．タスクの時間制約を満たすためには，タスク分割や適切な割り込み処理等の実装を行った上で，高い駆動周波数で駆動することが必要となり，開発者の負担と消費電力の増大を招く．また，シングルコア CPU で CPU 資源を複数の処理に割り当て，処理の並列性を保つためにはスケジューラが必要である．無線センサノードでは，スケジューラに起因する電力消費とタスクの実行遅延の発生が問題となる．

このような問題に対して筆者らは，時間制約のあるタスクの実装を容易とするとともに低消費電力での動作を実現する，無線センサノード向けマルチコア CPU「Mulco」の研究開発を進めている．無線センサノードでは，処理するタスクが固定的かつ周期的に実行されることから，事前にタスクをコアに割り振ることが可能となる．Mulco ではタスク処理を行うコアを決定する際に，1 つの CPU コア内で同時に複数の処理要求が発生しないようにすることでリアルタイム性を確保する．この際，割り振られたタスクの時間制約を満たす最低の駆動周波数で各コアを動作させることで，低消費電力での動作が可能となる．さらに，コア内で同時に複数のタスクの処理要求が発生しないため，スケジューラを排除することができる．

本論文では，Mulco の有効性を示すために以下の 3 つの評価を行う．第 1 に，シングルコア CPU の問題点を明確にするために，汎用センサノード上でアプリケーションを動作させ，スケジューリングに要する電力とタスクの実行遅延を計測した．結果として，スケジューラが CPU 全体の 10 % 程度の電力を消費していることや，処理コストの大きなタスクの存在により，無視できない大きさの実行遅延が生じることが示された．

第 2 に，CPU のマルチコア化で駆動周波数を低減させることによる，消費電力削減の効果を確認する評価を行った．マルチコア CPU を模した評価用回路として，3 つの PIC18LF2321 と 1 つのリアルタイムクロックからなるマルチコア CPU システムを実装し，アプリケーションを模したベンチマークを実行して，コア数ごとの消費電力を測定した．その結果，トリプルコア CPU の場合，シングルコア CPU と比較して，タスクの処理要求の偏りの大きなベンチマークの実行時に 70 % の消費電力が削減された．

第 3 に，マルチコア化による実行サイクル数削減による消費電力削減の効果を確認する評価を行った．複数のアプリケーションについて，シングルコア CPU を用いた場合とマルチコア CPU を用いた場合の実行サイクル数の比較を行った．スケジューラ排除により削減された実行サイクルよりもコア間通信のオーバーヘッドが小さい場合，マルチコア化により実行サイクルを削減することができる．結果として，測定対象とした全てのアプリケーションにおいて，CPU のマルチコア化により，実行サイクル数を削減できることがわかった．

本論文での 3 つの評価により，無線センサノード向け CPU としてマルチコア CPU が適していることが示され，今後の具体的なアーキテクチャ決定に際する一助となろう．

1.2　本論文の構成

本論文では，第 2 章で無線センサネットワークのこれまでの研究を概観し，無線センサノードの要件を整理する．また，既存のシステムと無線センサノード向け CPU の研究における問題点を述べる．さらに，既存のマルチコア CPU は，タスクの性質の相違により無線センサノードにそのまま適用できないことを説明する．第 3 章では我々の提案する無線センサノード向けマルチコア CPU「Mulco」について述べ，その特徴を説明する．第 4 章では Mulco の有用性を明らかにするための 3 つの評価を示す．まず，既存アーキテクチャの問題点を示す評価を行い，CPU のマルチコア化が必要であることを示す．続いて，Mulco がシングルコア CPU と比較して低消費電力での動作を実現できることを，電源電圧低減，実行サイクル数削減の 2 つの側面から示す．第 5 章では今後の研究方針と，無線センサノード向け CPU の開発プラットフォームである Mulco ノードに関して述べる．第 6 章でまとめとする．

第 2 章

現在の研究と問題点

2.1 はじめに

本章では，既存の無線センサネットワークの研究を説明し，それぞれの特徴をまとめ，問題点を指摘する．2.2 では，無線センサノードの要件を整理する．そのもとで，現在研究に使用されているプラットフォームをソフトウェアとハードウェアに分けて説明し，課題を抽出する．2.3 では，既存のシングルコア CPU を用いたシステムの問題点を指摘する．2.4 で関連研究として無線センサノード向けの CPU の研究がリアルタイム性に関しての考慮が欠如していることを指摘する．また既存のマルチコア CPU を紹介し，無線センサネットワークのタスクの特異性ゆえにそのまま無線センサノードに用いることができないことを説明する．

2.2 無線センサネットワーク

2.2.1 無線センサノードの要件

無線センサネットーワークはセンシング機能を有するコンピュータを無線通信によりネットワーク接続し，実空間の情報を収集する技術である．その「容易に導入可能」という利点を生かし，軍事やヘルスモニタリング，環境モニタリングなど様々な分野への応用が期待されている [2][3][4]．これらのアプリケーションの実行の際には，各センサノードにおいてサンプリング，計算処理，無線通信，デバイス制御といった複数のタスクが，1 つの安価な CPU 上で並列に実行される．

無線センサノードは以下の 2 つの要件を満たさなければならない．1 つ目の要件は，タスク処理の時間制約を満たすことである．センサノード上で処理されるタスクは固定性，周期性，ハードリアルタイム性の 3 つの特徴を持つ．例えば加速度センサを用いてユーザのコンテキストを推定するアプリケーション [5] では，10 ms 毎にセンサ値を取得し，計算処理を行ってユーザのコンテキストを推定する．無線通信においてビットレートが 38.4 kbps の場合，26 μs 毎にデッドライン 26 μs のタスクが発生する．もしこの物理層の処理が 26 μs のデッドラインを超えてしまった場合にはデータの送受信が失敗したことになり，サービスの質（QoS）は直ちにゼロになる．このようなタスクをハードリアルタイムタスクと呼ぶ．無線センサノードがタスクの時間制約を満たせない場合，処理落ちによるパケットロスや，サンプリングジッタといった問題が発生し，有用性を著しく損ねる．

2 つ目の要件は省電力性である．センサノードは有線での電源供給を受けられないこと

から，電力資源に制約をうける．センサノードの低消費電力化は大きな課題の 1 つであり，オペレーティングシステム，MAC プロトコル，ルーティングプロトコルなど，様々な視点から検討がなされている [6][7][8]．センサノード全体の消費電力のうち，CPU が占める割合は大きい．Shnayder らは，汎用センサノードである Mica2 ノードにおいて，実際のアプリケーションを実行した際の CPU，無線モジュール，センサ等の構成要素ごとの消費電力を示した [9]．[9] によると，CPU での消費電力はノード全体の消費電力の 28 % から 86 %，平均で約 50 % を占める．すなわち，センサノードの電力を削減するためには，低消費電力な無線センサノード向けの CPU が求められる．

しかしながら，既存のシステムは，リアルタイム性，省電力性ともに未だ多くのアプリケーションとって十分であるとは言えない．例えば山岳地帯にセンサノードを配置し，山火事を監視する FireWxNet [10] では，消費電力の観点から既存のルーティングプロトコルを用いることができず，ノードの配置時に位置情報を入力することでルーティングを行う．これにより，人間がヘリコプタで立ち入ることのできない場所への配置は不可能となる．また，橋梁に加速度センサを設置し，揺れを観測する研究 [3] では，オペレーティングシステムのソフトウェアスケジューラにより生じる不確定なサンプリングジッタによって，サンプリング値の有用性が著しく損なわれることが報告されている．

以上のことから，幅広いアプリケーションに適用可能な汎用性を持ったセンサノードの実現に向けては，タスクの時間制約を満たしたまま，いかに消費電力を削減するかが重要な課題となる．

2.2.2 センサノード向けソフトウェアプラットフォーム

無線センサノード向けのオペレーティングシステムとしては, TinyOS [7][11], SOS [12], Contiki [13]，protothreads [14]，MANTIS OS [15]，t-kernel [16]，PAVENET OS [6] があげられる．これらはイベント駆動型オペレーティングシステムとスレッドを用いたオペレーティングシステムに大分される．

TinyOS，SOS，Contiki，protothreads はイベント駆動型オペレーティングシステムに分類される．イベント駆動型オペレーティングシステムでは，タスクをイベントによって起動し，run-to-completion で実行することで，省資源性と高い安全性を提供する．一方で並列性の確保のために，ユーザーが処理を細かいタスクに分割して実装する必要があり開発が困難であることや，タスクが non-preemptive に実行されるため，リアルタイム処理をサポートできないという問題がある．

MANTIS OS，t-kernel，PAVENET OS はスレッドを用いたオペレーティングシステ

表 2.1 モデルによるオペレーティングシステムの比較

	省資源性	安全性	開発の容易さ	リアルタイム性
イベントモデル	○	○	×	×
スレッドモデル	×	×	○	○

ムに分類される．スレッドを用いたオペレーティングシステムでは，ユーザーは一連の処理をスレッドとして記述するため，容易な実装が可能となる．また，各スレッドには優先度が設定されており，優先度が高いスレッドは優先度が低いスレッドを preempt することにより，リアルタイム処理をサポートできる．一方，複数のスレッドが存在するために，省資源性や安全性ではイベント駆動型オペレーティングシステムに及ばない．

イベントモデルとスレッドを用いたオペレーティングシステムの比較を表 2.1 に示す．表 2.1 の通り，イベントモデルとスレッドモデルそれぞれに問題がある．protothreads のようにイベントモデルのオペレーティングシステムでありながらも，プログラムの書きやすさを実現した研究や，Pavenet OS のように，スレッドモデルのオペレーティングシステムでありながらも，特定のハードウェアに特化した実装を行うことで，イベントモデルと同等の省資源性を実現している研究も存在する．しかしながら，省資源性の消失やハードウェア依存により移植性を欠くという別の問題が生じている．現状ではオペレーティングシステムに求められる全ての要件を満たす研究は存在せず，省電力性の観点から TinyOS がデファクトスタンダードとして広く用いられている．

TinyOS は，カリフォルニア大学バークレー校を中心とするグループにより開発されたイベント駆動型のオペレーティングシステムであり，nesC [17] と呼ばれるイベントモデルに最適化された言語で構築され，全てのオペレーティングシステムの中で，最も省電力，省資源，低オーバーヘッドでの動作を実現している．

TinyOS が，デフォルトのスケジューラである SchedulerBasicP を用いてタスクを実行する様子を図 2.1 に示す．TinyOS では，`event` と `task` の 2 つの処理形式が用意されており，通常の処理を `task` として実装することで並列処理を実現している．`event` はハードウェア割り込み，他の `event`，`task` から呼び出されて実行され，割り込みから呼び出された場合は `task` を preemption できる．`task` は `event` や他の `task` から実行要求を受けてスケジューラを通して non-preemptive に実行される．具体的には，CPU はタイマや外部機器からのハードウェア割り込みを受けた時，対応する `event` が実行され，必要であればその `event` 内で `task` をタスクキューに挿入する（`post`）．最後にスケジューラがタスクキューから FIFO で `task` を取り出し，順番に実行（`run`）する．スケジュー

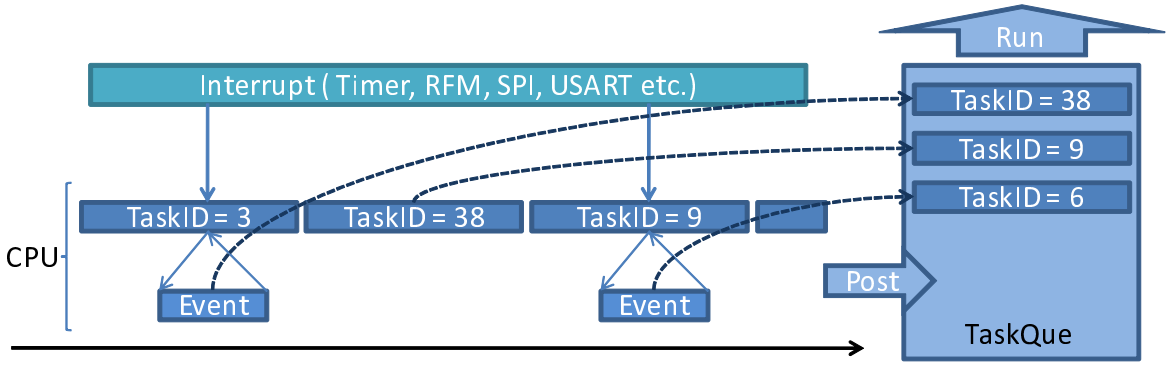

図 2.1　TinyOS のタスク実行の様子

ラは，タスクキューに実行待ちのタスクが存在しなくなった場合，CPU をスリープさせる．SchedulerBasicP では `task` を `post` および `run` する処理に 103 cycle を要する．また，タスクキューに実行待ちの `task` が存在しないことを確認して CPU をスリープさせる処理に 20 cycle を要する．MicaZ センサノード [18] の駆動周波数である 7.37 MHz ではそれぞれ 14.0 μs，2.7 μs に相当する．

TinyOS は，イベントモデルのオペレーティングシステムの短所である，リアルタイム性の欠如を，2 つの処理形式と FIFO のタスクスケジューラで解決しようとした．しかしながらリアルタイム性の確保はいまだ不十分であり，タスクスケジューラ自体が電力を消費するという問題も生じる．現状ではデファクトスタンダードとして扱われている TinyOS がバランスがとれた設計となっているものの，無線センサノードの要件を全て満たしているとは言いがたい．現在のオペレーティングシステムの研究は，ハードウェア，特に CPU は変更できないものとして扱っている．オペレーティングシステムの短所を補完しうる無線センサノード向け CPU が求められる．

2.2.3　センサノード向けハードウェアプラットフォーム

センサノードは主に CPU，無線モジュール，Flash メモリから構成される．汎用センサノードは，市販されている電子部品を組み合わせて実現されており，汎用性と消費電力のバランスを考慮して設計されている．例えば，カリフォルニア大学のバークレイ校が開発し，Crossbow Technology 社が販売している MicaZ ノード [18]（図 2.2）は，CPU として ATMEL 社の ATmega128L [19]，無線モジュールとして Texas Instruments 社の CC2420[20]，Flash メモリとして ATMEL 社の AT45DB [21] を具備している．

図 2.2　MicaZ ノードの外観

表 2.2　汎用のセンサノードと具備している CPU

Node	Developing Group	CPU
Micaz	UC Berkeley	ATMEL ATmega128
BT node	ETH Zurich	
TerosB	UC Berkeley	Texas Instrumens MSP430
BSN node	ICL	
PAVENET	University of Tokyo	Microchip PIC18F4620

表 2.3　汎用 CPU の電流値での比較

	ATmega	PIC18	MSP	8051
Word Size	8 bit	8 bit	16 bit	8 bit
f_{MAX} at 3V	8 MHz	20 MHz	6 MHz	6.3 MHz
Power Off	8 μA	2.6 μA	1.8 μA	21 μA
8 MHz	8 mA	2.4 mA	1.9 mA	13.3 mA

代表的な汎用センサノードとそれぞれが具備する CPU を表 2.2 に示す．またこれらの CPU の電流値での比較を表 2.3 に示す．表 2.2 と表 2.3 から分かるように，汎用性と消費電力の観点から，全ての汎用センサノードがシングルコアの 8 bit もしくは 16 bit CPU を具備している．性能も類似しており，多くの無線センサネットワークのアプリケーションが求める CPU 性能がこの範囲にあることがわかる．汎用センサノードでアプリケーションの要件を満たせない場合は，専用のセンサノードを設計してアプリケーションを構築する．例えば，Park らの構造モニタリングの研究 [22] では，サンプリングのリアルタイム性を確保するために 2 つの PIC18 マイコンを具備したセンサノードを構築している．

現在，無線センサネットワークは一部を除き研究段階にとどまっているため，性能の近いと思われる汎用の CPU で処理をおこなっている．しかしながら，電源資源や計算資源に大きな制約をうける無線センサネットワークでは，CPU は極限まで資源を節約すると同時に幅広いアプリケーションに適応するすることが求められる．今後実現可能なアプリケーションが増えることに伴ない，無線センサネットワークのアプリケーションの共通点を十分に考慮したうえで，無線センサノード専用の CPU を設計する必要があろう．

2.3 既存システムの問題点

現在アプリケーションを構築する際には，多くの場合オペレーティングシステムとして TinyOS，CPU としてシングルコア CPU が用いられる．しかし TinyOS をシングルコア CPU 上で動作させアプリケーションを実行すると，同時に解決することが困難な 2 つの問題が発生する．1 つ目の問題は，タスク実行の際に発生する遅延である．遅延の発生は以下の 2 つに因る．1 つ目の遅延要因は，スケジューラ自身の処理によって発生する 14.0 μs の遅延である．この遅延はすべてのタスクに対して等しく発生する．2 つ目の遅延要因は，あるタスクを `post` する際に実行中のタスク及びタスクキューに入っているタスクの処理によって発生する遅延である．2 つ目の遅延は先発タスクによって決定されるため，不確定な大きさの遅延となる．

2 つ目の問題は，電力消費である．無線センサノードの CPU で処理されるタスクの多くは計算量が小さく，103 cycle というタスクスケジューラのオーバーヘッドを無視できない．CPU での電力消費がノード全体での電力消費に占める割合は大きく，CPU で実行されるスケジューラのオーバーヘッドが大きい場合，その電力消費はノード全体にも大きな影響を及ぼす．また，タスクの実行遅延が不確定な大きさで発生するため，余裕を持った駆動周波数の選定が必要となり消費電力の増大につながる。

一般的に CPU が高い周波数で駆動する場合には，計算量当たりの消費電力が増大す

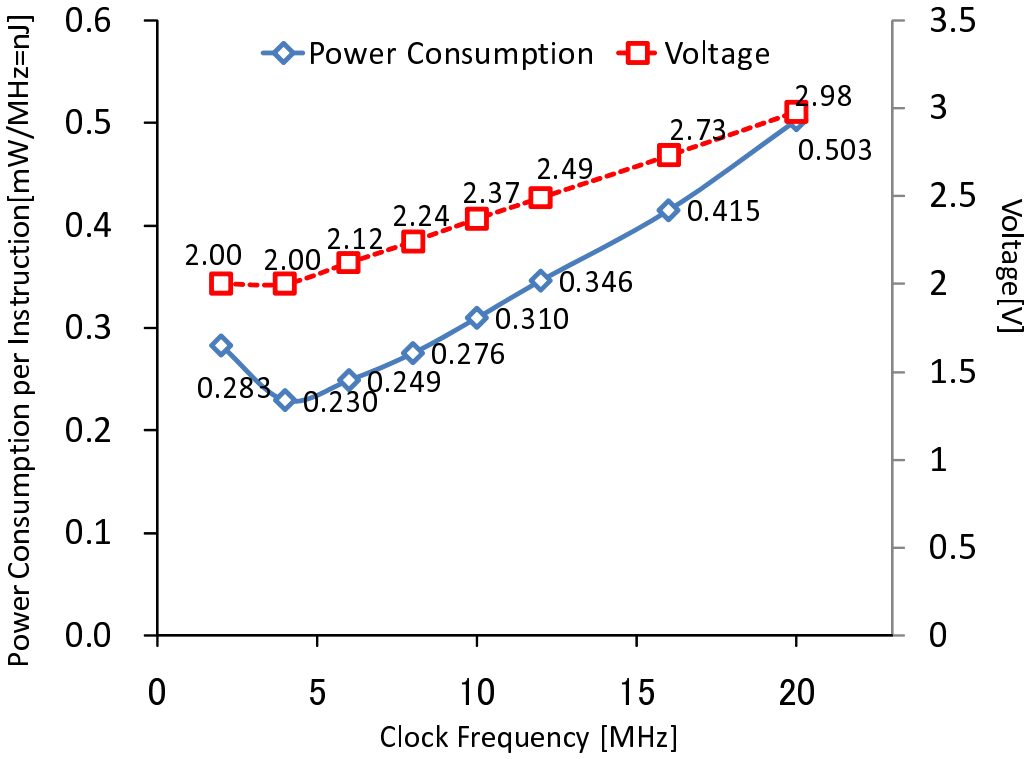

図 2.3　計算量あたりの消費電力と駆動周波数の関係

る [23]．図 2.3 に，PIC18LF2321 [24] を用いて，複数の駆動周波数での計算量あたりの消費電力を測定した結果を示す．また，各駆動周波数における最低駆動電圧を示す駆動周波数 f_{max} と最低駆動電圧 V の関係は次の式で与えられる [23]．

$$f_{max} = 16.36 \times (V - 2.0) + 4 \quad (V \geq 2.0) \tag{2.1}$$

図 2.3 と式 (2.1) から分かるように，駆動周波数を増加させると，計算量当たりの消費電力が増大する．例えば，20 MHz では，4 MHz で CPU を駆動させた場合と比べて，220 % の電力を消費している．逆に駆動周波数を低減させ，同時に電源電圧を低く抑えることで，計算量当たりの消費電力を削減できる．なお，2 MHz で 4 MHz よりも多くの電力を消費しているのは，4 MHz より駆動周波数を低減させても，電源電圧を下げることができない PIC18LF2321 の仕様に起因する．

現在のシングルコア CPU のシステム上でスケジューリングによる電力消費とタスクの実行遅延の発生という問題を同時に解決し，省電力性とリアルタイム性を同時に実現することは困難である．スケジューリングのオーバーヘッドを軽減するためには，分割されて

実装されているタスクをまとめて 1 つのタスクとし，一度のスケジューリングで多くの処理を行うことが考えられる．しかし，1 つのタスクの計算量が大きくなると，前発タスクの処理による後発タスクの実行遅延が増大する．またシングルコア CPU において実行遅延を低減するためには，駆動周波数をあげ処理性能を向上させるか，同時に処理要求が発生しないように実装を行う必要があり，前者は消費電力の増大を招き，後者は実装を複雑化する．

以上の観点から，シングルコア CPU を用いたシステムでは，リアルタイム性と省電力性を両立することは困難である．この問題に対し，CPU のマルチコア化が有効な手段となる．無線センサノード向けマルチコア CPU では，コア内で同時に複数の処理要求が発生しないようにすることで，低消費電力を維持したままタスク処理の時間制約を容易に満たすことができる．このような観点から筆者らは，無線センサノード向けマルチコア CPU「Mulco」の研究開発を進めている．

2.4 関連研究

2.4.1 無線センサノード向け CPU の研究

無線センサノード向け CPU の研究としては，YUPPIE [25]，SNAP/LE [26]，Subliminal [29][30][31] が代表的である．これらの研究は CPU の消費電力削減に主眼を置いており，MicaZ ノードに搭載されている ATMEL 社 ATmega の 4 MIPS，1500 pJ/ins での動作と比較して，低消費電力な動作を実現している．しかしながら，現状の無線センサノード向け CPU の研究は，リアルタイム処理に関する考慮にかけていると言える．

YUPPIE

YUPPIE [25] は ATMEL 社の ATmega を非同期化した CPU であり，0.51 V で 48 MIPS，2.7 pJ/ins での動作が可能であるとしている．非同期 CPU は動作部分でのみ電力を消費するため低消費電力化が実現できることに加え，スリープからの復帰の際にクロック素子の安定を待たなくてもよいため，低遅延での復帰が可能となるなど，無線センサネットワークにおいて多くのメリットを持つ．

しかしながら，非同期 CPU にはクロックが存在しないため遅延保証がない．また，性能が電源電圧により決定するため，リアルタイム処理を保証することが困難である．

SNAP/LE

SNAP/LE [26] は無線センサノードに特化して設計された非同期 CPU である．SNAP/LE では，ソフトウェアスケジューラを排除し，FIFO のハードウェアタスクスケジューラにより低消費電力で処理の並列性を実現している．命令セットは，SNAP ISA [27] と呼ばれ，RISC の標準的な命令の他に，Timer Coprocessor 制御用の命令，Message Coprocessor 制御用の命令，ネットワークプロトコル用の命令，ハードウェアスケジューラ制御用の命令を追加したものである．これらにより，SNAP/LE は 0.6 V で 28 MIPS，24 pJ/ins での動作，数十 ns でのスリープからの復帰を実現した．これは ATmega128 の 7 倍の処理能力を 1/60 の消費電力で実現していることに相当する．また，センサネットワークのアプリケーションが高処理能力を要求しないことから，SNAP/LE の低消費電力版である BitSNAP [28] も設計された．BitSNAP では，SNAP/LE における電力消費の約 60 % がデータパスにおけるものであることに着目し，そのシリアル化と，データの圧縮により 0.6 V で 6 MIPS，17 pJ/ins での動作を実現している．

しかしながら，YUPPIE と同様に非同期 CPU であるために遅延保証がない．また，FIFO でタスクを処理するため，タスクの処理要求が同時に複数発生した場合，タスクの実行遅延が問題となる．

Subliminal

Subliminal [29][30][31] は無線センサネットワーク向けのサブスレッショルドプロセッサを開発しているグループである．Subliminal グループが設計した第 2 世代のプロセッサ [30] では，サブスレッショルドプロセッサの電力消費に影響を与える命令セットとして，トランジスタの利用効率の高い 12 bitRISC を採用している．8 bit データパス，独立のデータメモリと命令メモリ，3 段のパイプライン処理の場合に，0.1 MIPS で 600 fJ/ins を実現している．同じグループの [31] では，[30] と比べてやや高性能に最適化されており，温度等の環境変化に対する耐久性を高め，細やかな消費電力制御を可能とする，body biasing の有効性を示している．また，[30]，[31] ともに，無線センサネットワークのタスクの周期性に着目し，ハードウェアのタスクスケジューラを具備している．

しかしながら，性能が限られており適用範囲が狭い．また，サブスレッショルド領域での動作は温度変化に対して性能が大きく変動するため，ハードリアルタイム処理を保証するためには，余裕を持った駆動周波数の選定が必要となる．

2.4.2 マルチコア CPU

パーソナルコンピュータや組み込み機器向けの CPU においても，消費電力削減の必要性や処理の汎用性確保の観点からマルチコア CPU が広く用いられている．しかしながら，タスク特性の差異のために，既存のマルチコア CPU のアーキテクチャは，無線センサノード向けマルチコア CPU にとって最適なアーキテクチャとはならず，そのまま用いることは適当ではない．つまり，無線センサノード向けマルチコア CPU を開発する際には，タスクの特性を考慮して無線センサノード向けのアーキテクチャの検討を行う必要がある．

パーソナルコンピュータ向けマルチコア CPU

パーソナルコンピュータ向け CPU は，性能向上のために駆動周波数をあげるにつれ発熱が増加し，近年になり周波数増加による性能向上の限界が指摘されている．これはリーク電流やトランジスタ数の増加に伴い，プロセスの微細化による消費電力削減が期待できなくなったためである．パーソナルコンピュータ向け CPU における最も重要な要件は，性能の向上であるため，パーソナルコンピュータ向けマルチコア CPU は，高性能と低消費電力の実現の要求に基づき，負荷分散を行うことに主眼を置く．このため，ホモジニアスで対称な構成となる．

パーソナルコンピュータ上のタスクは，多様性に富み，予測不可能であり，ソフトリアルタイムタスクであるという 3 つの特徴を持つ．パーソナルコンピュータ上では常に数十個のタスクが並列に起動されており，ユーザの要求に応じて，多様で予測不可能なタスクが実行される．またパーソナルコンピュータ上のタスクは，デッドラインを過ぎると QoS が徐々に低下するソフトリアルタイムタスクである．一方，無線センサノード向け CPU において最も重要な要件は，消費電力の削減である．また，無線センサノードの上のタスクは固定的で周期的に実行され，パーソナルコンピュータと比べ少数で負荷の小さいタスクを処理する．

このようなタスクの特性の相違から，パーソナルコンピュータ向けマルチコア CPU のアーキテクチャは，無線センサノード向けマルチコア CPU にとって最適なアーキテクチャとはならない．パーソナルコンピュータ向けマルチコア CPU では，タスクを実行時にコアに割り振ることで，各コアの駆動周波数を落としつつも QoS を維持している．無線センサノードでは，パーソナルコンピュータに対して負荷の小さいタスクを処理するため，実行時にタスクを割り振る処理の負荷が相対的に大きくなってしまうと考えられる．

また，パーソナルコンピュータのマルチコア CPU に用いられているメモリ共有アーキテクチャを，無線センサノードにそのまま適用した場合，メモリアクセスの排他制御のための待ち時間が，タスク処理にかかる時間に比べて無視できないほど大きくなり，電力を無駄に消費する可能性がある．

組み込み機器向けマルチコア CPU

組み込み機器上で処理されるタスクは，無線センサノード上で処理されるタスクと類似した特徴を有しており，あらかじめ決められたことを決められた手順で，かつ最小限の消費電力で処理することが求められる．これまで信号処理はおもに専用のハードワイヤード論理で実現されてきた．しかし，携帯電話やカーナビゲーションシステムなど，一部の組み込み機器では，機能の複合化に伴ってシステムが複雑かつ大規模になってきている．また，携帯電話などでは，市場の急激な立ち上がりに対応するため，非常に短い期間での開発が求められている．このため，汎用性に優れた CPU や DSP によって処理を行うことが，カスタマイズやデバッグの観点から必要となってきている．

組み込み機器では，さまざまな処理が分散・協調しながら実行される．つまり，各入出力毎に CPU や DSP などのプロセッサが処理を行う構造となる．このような機能を 1 つのプロセッサとして実現する際には，性能が必要な部分を専用ハードウェアや DSP で処理させ，残りの処理を小さな CPU で行うというような機能分散型のマルチプロセッサ形態となる．つまり，組み込み機器向けマルチコア CPU では，ヘテロジニアスで非対称な構造が重要となる．

組み込み機器上で処理されるタスクは，周期性，固定性，リアルタイム性の観点から，無線センサノード上で処理されるタスクと類似している．しかし，組み込み機器向けマルチコア CPU のアーキテクチャは，機能（タスク）間の通信頻度やその負荷の影響から，無線センサノード向けマルチコア CPU にとって最適なアーキテクチャとはならない．ヘテロジニアスで非対称なマルチコア CPU では，多くの場合共有メモリを持たず，コア間通信により発生するオーバーヘッドが性能に影響を与える．組み込み機器では，コア間の通信，つまり機能間での通信が疎な場合に，特にマルチコア化によるメリットが大きい．無線センサノードでは，コア間通信によるオーバーヘッドが，負荷の小さなタスク処理に対して相対的に大きくなり，消費電力の増大を招く可能性がある．

2.5　おわりに

本章では，リアルタイム性と省電力性という無線センサノードの要件を，シングルコア CPU のシステムでは満たせないことを説明した．既存の無線センサノード向け CPU の研究では，省電力性に関する検討は行っているものの，リアルタイム性に対する考慮にかけている．今後，無線センサネットワークが広く普及するためには，センサノード向け CPU は多くのアプリケーションの要求を満たす必要があり，リアルタイム性に関する検討が必要である．

第 3 章

Mulco

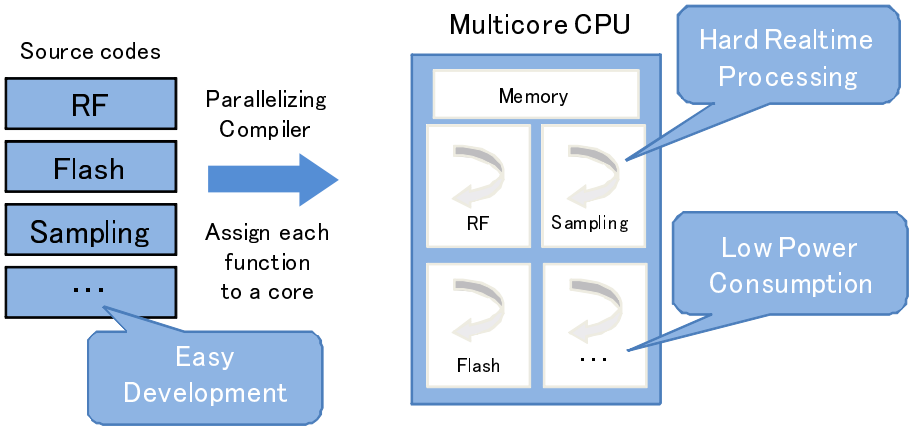

図 3.1　Mulco の概念図

3.1　はじめに

本章では我々が研究開発を進めている, 無線センサノード向けマルチコア CPU「Mulco」について述べる．3.2 で無線センサノード向けのマルチコア CPU を設計する際に考慮すべき事項について述べ，3.3 で Mulco の特徴とそれによってもたらされる恩恵に関して説明する．3.4 では Mulco で必要となるコア数に関する検討をおこない，無線センサネットワームでは数個のコアでアプリケーションを構成できることを示す．

3.2　Mulco の設計

筆者らは，無線センサノード向けマルチコア CPU「Mulco」の研究開発を進めている．図 3.1 に示す通り，Mulco の目的は CPU の消費電力を削減すると同時にリアルタイムタスクの実装を容易とすることである．2.4 で述べたように，既存のマルチコア CPU のタスクの割り当て方，消費電力削減手法，アーキテクチャは無線センサノードでは必ずしも最適とはならない．これは無線センサネットワークで処理されるタスクの特異性に起因する．

例えば無線センサノード向けマルチコア CPU では，実行前にタスクをコアへ静的に割り当て，コアごとに性能を決定するべきである．無線センサノードの CPU 上では，タスクが固定的，周期的に実行される．また，各タスクの計算量はパーソナルコンピュータや組み込みシステムと比較して小さい．このため，複数のコアを用意し負荷分散を行う既存

の消費電力削減の手法は効果的であるとは言えない．

また，CPU のマルチコア化によって生じるデメリットに関しても考慮しなければならない．マルチコア化によるデメリットとしては以下の 2 点が考えられる．

回路規模の増加 複数のコアが存在することで，回路規模が増加する．無線センサノードに搭載される CPU コアの回路規模は，SRAM やノードの他の構成要素と比較して小さいものの，コア数が増えすぎるとノード全体の回路規模増加につながる恐れがある．

コア間通信の発生 コア間でデータを共有する場合，何らかの方法でコア間でデータを受け渡す必要がある．その際，コア間通信のために要する時間や，共有メモリへのアクセスの際の排他制御による待ち時間がオーバーヘッドとして発生する．オーバーヘッドが大きくなると，マルチコア化による消費電力削減の効果を打ち消し，逆に消費電力が増加する恐れがある．

例えばメモリ配置に関しては，共有メモリと局所メモリの使い分けを，各コアがメモリにアクセスするタイミングと，コア間での共有データ量を考慮して注意深く決定する必要がある．次章では，コア間で共有されるデータの量を評価し，考察を加える．

以上のように，Mulco の設計に際しては，無線センサネットワークのタスクの特異性を十分に考慮しながら，タスクの割り当て方からハードウェア構成までを含めた，総合的な検討が必要となる．

3.3 Mulco の特徴

Mulco は以下の 2 つの特徴を持つ．1 つ目の特徴は，割り込み要求を受けて実行を移すべきコアを判断する割り込みディスパッチャを具備することである．2 つ目の特徴は，機能ごとに 1 つのコアを割り当てることである．無線センサノードでは，タスクが固定的かつ周期的に実行されることから，各コアへのタスクの割り当ては，コンパイラや開発者により事前に行うことが可能である．各タスクは，図 3.3 に示すように，割り込みディスパッチャからの処理要求を受け逐次的に実行され，サンプリングや無線送信などのまとまった機能を実現する．本稿ではこのタスク群をタスクフローと呼ぶ．図 3.3 は，割り込みディスパッチャがタイマ割り込みを受けてサンプリングと無線送信のタスクフローに実行要求を振り分ける例を示している．表 3.1 に無線センサネットワークのアプリケーションを構成する機能を示す．

このように各タスクを複数のコアに分割することによって，タスク同士がコア内で影響

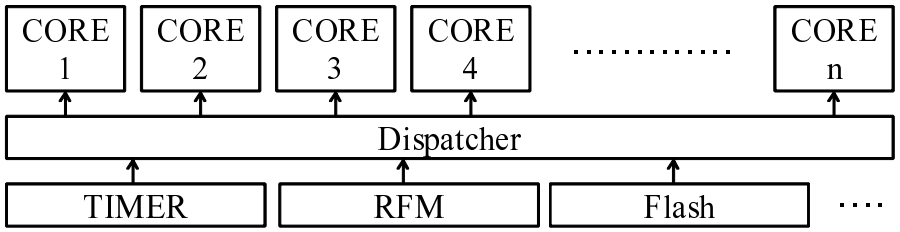

図 3.2 Mulco のアーキテクチャ

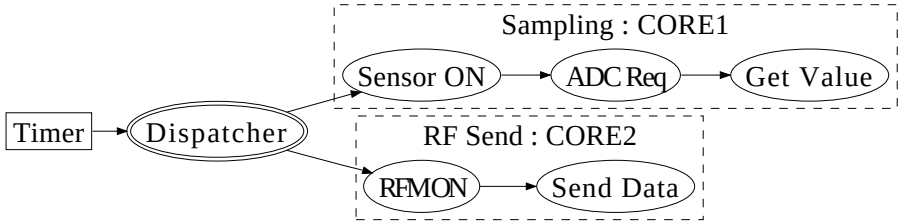

図 3.3 タスクフローの例

表 3.1 CPU で行われる処理の分類

タスクフロー	含まれる処理
Sampling	センサ制御，ADC 制御
RFM Driver	RFM 制御，RFM ステータス管理，LPL，CRC
Routing	テーブル管理，ルート計算
FTSP	時刻同期
Flash	ブロック書き込み，読み出し
Caluculate	加算，減算，乗算，除算，平均，RMS

を与え合うことを防ぐことができる．つまりタスクフロー内ではタスクの実行中に他のタスクの処理要求が発生しないため，1 つのコア内で複数のタスクの処理要求が同時に発生することはない．タスクフローごとにコアを用意することで，同時に処理要求が発生しうるタスクが別のコアに割り振られるため，処理のリアルタイム性を確保することができる．また，シングルコア CPU において CPU 資源を複数の処理に分配するために必要であったスケジューラを排除することができるため，スケジューリングに要する電力を削減できる．さらに，不確定な大きさの遅延を排除できるため，タスク処理の終了時刻を正確に把握できる．開発者は各コアに割り振られたタスクをデッドラインまでに処理するため

に必要な最低の駆動周波数を与え，またその駆動周波数での動作を保証された最低の電源電圧を与える．各タスクをシングルコア CPU での処理と比較して低い駆動周波数で処理することが可能となり，消費電力を削減することができる．

Mulco では，コア内でのスケジューリングは不要となるが，コア間でのスケジューリングは依然として必要である．例えば，各コアが共通のタイマを利用する場合には，発生した割り込みがどのタスクフローに対応するものなのかを判断する必要がある．この条件判断は柔軟に記述できることが好ましい．Protothreads [14] ではイベントモデルの TinyOS に疑似的にスレッドを導入し，条件ブロックを可能とすることでソフトウェアの記述性を大きく向上させている．Mulco ではこの条件判断を Protothreads のようにソフトウェアで行うことも可能ではあるが，効率的かつ柔軟に行うことができるハードウェアを設計することができれば，さらに効率化することができる．

3.3.1 タスクフロー解析

タスクフローの初期的な検証として，TinyOS のサンプルアプリケーションである MultihopOscilliscope のタスクフローの解析を行った．MultihopOscilliscope は定期的にサンプリングを行い，センサデータをマルチホップでシンクノードにリアルタイム転送するアプリケーションであり，サンプリング，無線送受信，ルーティング等，センサネットワークのアプリケーションに共通して必要とされるタスクの多くが含まれている．

表 3.2 に MultihopOscilliscope に含まれるタスクと測定時のタスクの実行回数，表 3.3 に割り込み要因および割り込み処理の実行回数を示す．表 3.2 と表 3.3 の各実行回数より，各処理がどのタスクフローに所属しているかが分かる．例えば，表 3.2 における実行回数が 381 回となっているタスク `ArbiterP/0/grantedTask` とタスク `PowerManagerP/0/startTask` は同じタスクフローに属する．また，表 3.2 における実行回数が 844 回であるタスク `AlarmToTimerC/0/fired` と表 3.3 における実行回数が 844 回である割り込み `TIMER0 Compare Match` は同じタスクフローに属する．

表 3.2 および表 3.3 の実行回数と，ソースコード内でのタスクの依存関係から解析したタスクフローを図 3.4 に示す．図 3.4 は `event` および `task` 間の要求関係と，次のタイマの発火設定など割り込みを発生させる `task`，`event` の因果関係から決定される．図 3.4 の四角は割り込みハンドラの処理，二重円はソフトウェアタイマなどの割り込みを受け付けシグナルを送信すべき適切なタスクフローを選択する処理，楕円は `task` と `event` での処理を表している．また，上から下への矢印では，`task` が `post` されており，横方向の矢印ではハードウェア割り込みは `event` として処理が遷移することを示している．

表 3.2 MultihopOscilloscope のタスク

TaskID	Task	Number of Times
1	AlarmToTimerC/0/fired	844
2	VirtualizeTimerC/0/updateFromTimer	1298
3	ArbiterP/0/grantedTask	381
4	ArbiterP/1/grantedTask	381
6	PowerManagerP/0/startTask	381
7	PowerManagerP/0/stopTask	381
10	PhotoTempControlP/0/stopDone	380
12	ArbiterP/3/grantedTask	381
13	AdcP/acquiredData	381
16	CC2420CsmaP/sendDonetask	78
23	CC2420ReceiveP/receiveDonetask	78
24	CtpForwardingEngineP/0/sendTask	153
27	CtpRoutingEngineP/0/updateRouteTask	6
28	CtpRoutingEngineP/0/sendBeaconTask	1
29	SerialP/RunTx	307
33	SerialDispatcherP/0/signalSendDone	153
37	UARTDebugSenderP/sendTask	154

図 3.4(a) を例にタスクフローの説明を行う．このタスクフローでは 8 つのタスクと 2 つの割り込み要求発生の待機から構成される．サンプリング周期ごとにタイマが発火し，センサの電源を ON にしてセンサの電源が安定するまでのタイマを設定する．設定したタイマが発火した後，AD コンバータに対して読み取り要求を発行し，変換完了割り込みが発生するまで待機する．AD 変換が完了したら割り込みが発生し，センサの電源を OFF としてタスクフローを終了する．

MultihopOscilloscope では，サンプリングに関連するタスクフロー（図 3.4(a)），ルーティングに関するタスクフロー（図 3.4(b)）および無線通信のデータ受信に関連するタスクフロー（図 3.4(c)）の 3 つの非同期なタスクフローが存在する．それぞれのタスクフローは独立した関係にあるため，このタスクフローごとにコアを割り当てると，コア内で非同期なタスクの実行要求が発生しない．

このタスクフロー図から，MultihopOscilloscope を Mulco で実装する際には，3 つのコアに割り振れば非同期な実行要求は発生せずに，実行遅延とスケジューラのオーバーヘッドの 2 つの問題を解決できると言える．ただし，図 3.4(a) においてはサンプリング

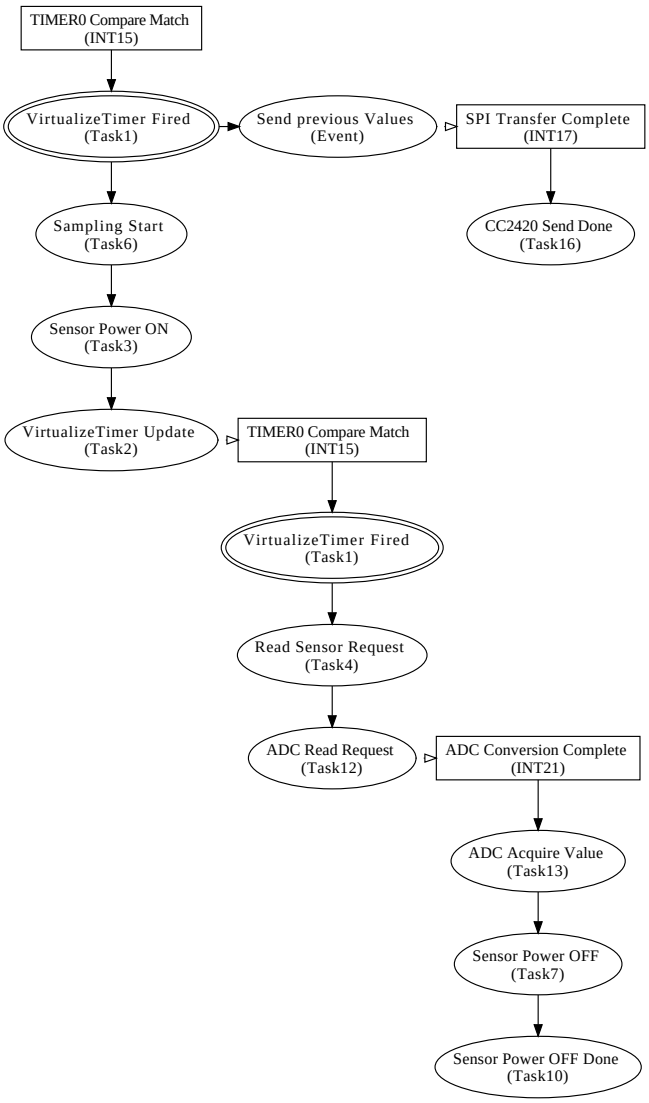

図 3.4(a) サンプリングとデータ送信

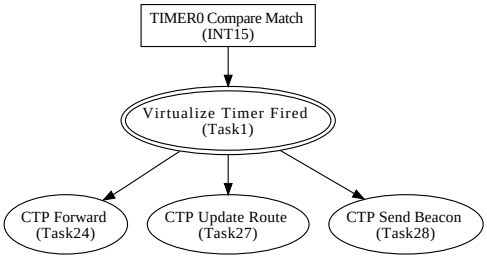

図 3.4(b) ルーティング

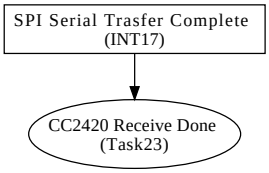

図 3.4(c) データ受信

図 3.4 MultihopOscilloscope のタスクフロー

表 3.3 MultihopOcsilloscope の割り込み

InterruptID	Interrupt	Number of Times
7	External Interrupt 6	231
11	TIMER2 Overflow	535
12	TIMER1 Compare Match A	159
14	TIMER1 Overflow	16
15	TIMER0 Compare Match	844
17	SPI Serial Transfer Complete	1081
20	USART0 TX Complete	3382
21	ADC Conversion Complete	381
29	TIMER3 Overflow	522

のタスクフローとセンサ値送信のタスクフローが同期して起動する．2 つのフローは同期しているため同時に処理要求が生じることはないが，並列に実行される可能性があるため別のコアを割り当てることで消費電力を削減できる可能性が高い．

3.4 Mulco のコア数に関する検討

タスクフローごとにコアを用意するというアプローチでは，アプリケーションに含まれるタスクフロー数によっては必要となるコア数が膨大となってしまう．コア数が増えすぎた場合，回路規模の肥大化や，コア間通信のオーバヘッドの増大を招く．このような観点から，以下で必要となるコア数を把握するために，センサノードで実行されるタスクの実行フローを解析し，タスクフロー数がどの程度なのかを示す．多くの無線センサネットワークのアプリケーションでは，サンプリングにより得られたセンサ値を解析し，無線通信によってデータをシンクノードに収集する．これらの 3 つのアプリケーションには，サンプリング，マルチホップ通信，時刻同期，省電力制御などの処理が含まれており，これらの処理を組み合わせることで多くの無線センサネットワークのアプリケーションを構築することができる．つまり，3 つのアプリケーションを構成するタスクフロー数を知ることで，アプリケーションを構築する際に必要となるタスクフロー数を把握することができる．解析は，実行されたタスクの ID を汎用 I/O に出力してロジックアナライザで計測し，各タスクの実行回数とソースコードでのタスクの依存関係をあわせることにより行った．

解析の結果得られたタスクフローの例として，解析を行った 3 つのアプリケーションの

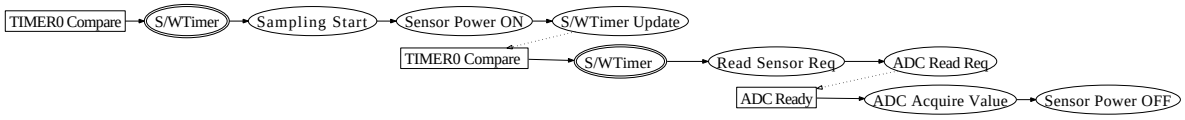

図 3.5 Sampling のタスクフロー．図中の四角は割り込みハンドラ内の処理，二重円はソフトウェアタイマの処理，楕円はタスクでの処理を表す．

表 3.4 アプリケーションに含まれるタスクフローと必要なコア数

アプリケーション	含まれるタスクフロー	コア数
MultihopOscilloscope	Sampling，Routing，RF Receive，RF Send	4
LowPowerSensing	Sampling, LPL, Flash Write	3
TestFTSP	FTSP Receive, FTSP Send	2

うち，MultihopOscilloscope と LowPowerSensing に共通して含まれるサンプリングタスクフローを図 3.5 に示す．タイマにより起動されたサンプリングタスクフローが，センサの電源 ON，ADC への処理要求，センサ値の取得，センサの電源 OFF という順番で処理されていることが分かる．この際，タスクフロー内ではタスクの実行中に他のタスクの処理要求が発生しないことから，サンプリングタスクフローが 1 つのコアを占有すれば，コア内で同時に複数の処理要求が発生することがない．

表 3.4 にそれぞれのアプリケーションに含まれるタスクフローと，タスクフローごとにコアを割り当てた際に必要となるコア数を示す．例えば MultihopOsilloscope は，前節で示したように Sampling, RF Receive, RF Send, Routing のタスクフローにより構成されるため 4 つのコアが必要となる．表 3.4 より，無線センサネットワークのアプリケーションではタスクフローごとにコアを用意しても数個のコアで十分であることが分かる．無線センサノードでは 1 コアのコストが低く，特に現在のセンサノードでは CPU ダイの面積はほぼ SRAM によって占められている．また CPU コアは数千ゲートで構成されているため，他のセンサノードの構成要素と比較すると，数個のコアであればマルチコア化によるコストの増大を抑えることが可能である．このことから，タスクフローごとにコアを用意する Mulco のタスク分割手法が現実的であると考えている．

3.5　おわりに

本章では我々が研究開発を進めている, 無線センサノード向けマルチコア CPU「Mulco」を紹介した．その上で Mulco の設計に際しては，無線センサネットワークのタスクの特異性を十分に考慮しながら，タスクの割り当て方からハードウェア構成までを含めた，総合的な検討が必要となることを説明した．また，現状での Mulco のデザインに関して述べ，機能ごとにタスクを割り当てることによる消費電力削減手法に関して述べた．

第 4 章

評価

4.1 はじめに

本章では，Mulco の有効性を示す 3 つの評価に関して述べる．4.2 では現在のシングルコア CPU を用いたシステムで発生する，スケジューリングに起因する電力消費とタスク処理の実行遅延を定量化し，これらが許容できない問題であることを示す．その上で，4.3 では CPU をマルチコア化することで電源電圧を低減させることにより削減される電力に関する評価を示す．また 4.4 ではマルチコア化により削減される CPU の実行サイクル数に関する評価を示す．

4.2 スケジューラに起因する電力消費と遅延の定量化

本節では現在のシングルコア CPU における問題点を明確にするために，タスクスケジューラに起因する電力消費と遅延の定量化を行う．本節の評価により，これらの問題が許容できず，無線センサノード向けのマルチコア CPU が必要であることが示される．

4.2.1 スケジューリングによる電力消費

スケジューリングによる CPU での電力消費の測定のために，TinyOS の複数のサンプルアプリケーションにおけるタスクの実行要求／実行開始／実行終了のタイミング，割り込みルーチンの実行開始／実行終了のタイミングを記録した．

具体的にはセンサノードとして MicaZ ノード [18] を利用し，タスクの実行開始時および実行終了時に汎用 I/O にタスク ID，割り込み ID を出力する測定用のコードをスケジューラおよび割り込みルーチンに挿入する．サンプリング周波数が 100 MHz のロジックアナライザを利用してタスクおよび割り込みごとに実行の開始および終了のタイミングを記録した．実験に使用した MicaZ ノードとロジックアナライザを接続した様子を図 4.1 に示す．

例として，図 4.2 に TinyOS のデフォルトのスケジューラである SchedulerBasicP のソースコードと測定用コードの挿入部を示す．SchedulerBasicP では，(1) でタスクキューの先頭を取り出し，タスクキューに実行待ちのタスクが存在しない場合は (2) で CPU をスリープさせ，タスクキューに実行待ちのタスクがある場合は (4) で実際に実行する．(3) と (5) が計測用に用意した関数であり，それぞれ実行開始時と実行終了時にタスク ID を汎用 I/O に出力する．タスクが `post` される時刻および割り込み処理に関し

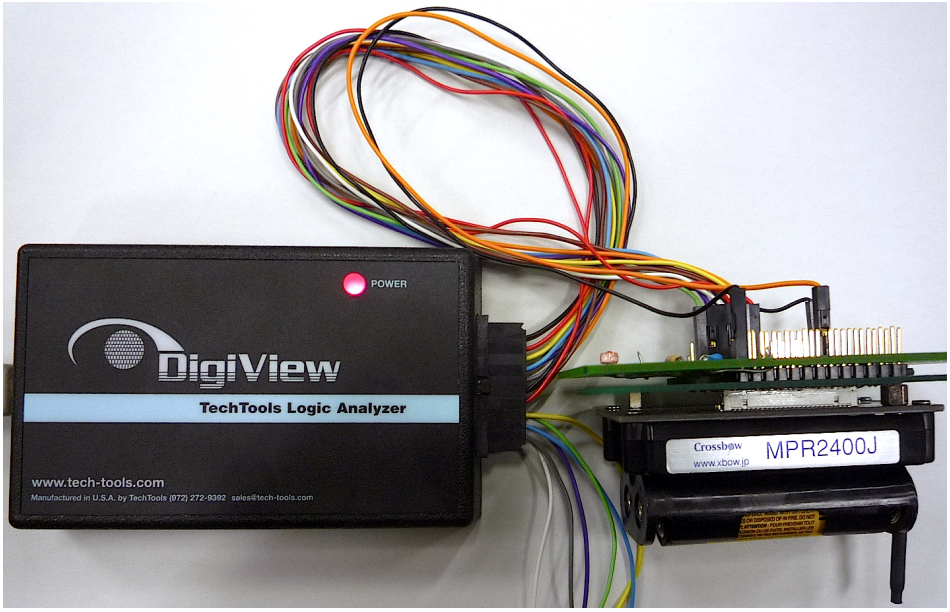

図 4.1　MicaZ ノードとロジックアナライザ

```
command void Scheduler.taskLoop(){
  for (;;){
    uint8_t nextTask;
    atomic{
      while ((nextTask = popTask() == NO_TASK){・・・(1)
        call McuSleep.sleep();・・・(2)
      }
    }
    setTaskIDStart(nextTask);・・・(3)
    signal TaskBasic.runTask[nextTask]();・・・(4)
    setTaskIDStop(nextTask);・・・(5)
  }
}
```

図 4.2　SchedulerBasicP

表 4.1　Sense のタスク

TaskID	Task	Number of Times	Cost（μs）
0	AlarmToTimerC/0/fired	194	59.1
1	VirtualizeTimerC/0/updateFromTimer	291	57.0
2	ArbiterP/0/grantedTask	97	37.1
3	ArbiterP/1/grantedTask	97	30.0
4	ArbiterP/2/grantedTask	0	—
5	PowerManagerP/0/start	97	31.3
6	PowerManagerP/0/stop	97	21.7
7	PowerManagerP/1/startTask	0	—
8	PowerManagerP/1/stopTask	0	—
9	PhotoTempControlP/0/stopDone	97	23.7
10	PhotoTempControlP/1/stopDone	0	—
11	ArbiterP/3/granted	97	28.0
12	AdcP/acquiredData	97	42.3
Average			42.0

ても同様の方法で計測を行った．

評価を行ったサンプルプログラムの 1 つである Sense のタスク一覧と，各タスクの処理にかかる時間（Cost）を表 4.1 に示す．Sense は定期的にセンサ値をサンプリングし，下位 3bit を LED に表示するアプリケーションである．表 4.1 のタスク ID は TinyOS でタスクに付与される ID，Number of Times は測定期間内に当該タスクが実行された回数，Cost は当該タスクの実行に要した時間の平均である．最下段の Average は実行されたすべてのタスクの回数によって重みを付けた実行時間の重み付け平均である．Sense では，`AlarmToTimerC/0/fired` の 1 回の実行時間が 59.1 μs と最も大きく，全てのタスクの実行時間の平均が 42.0 μs となる．

同様の計測を Oscilloscope，Blink，RadioSenseToLeds，MultihopOscilloscope，RadioCountToLeds，LowPowerSensing のアプリケーションについて行い，タスク処理の平均コストを算出した．これらのアプリケーションは TinyOS のソースコードツリーに含まれているサンプルアプリケーションであり，実用的なセンサネットワークを動作させるうえで必要不可欠なタスクが含まれている．さらに，アプリケーション実行時に CPU が動作している時間とタスクの処理に要した時間から，全タスクに対するスケジューラのオーバヘッドの割合を算出した．

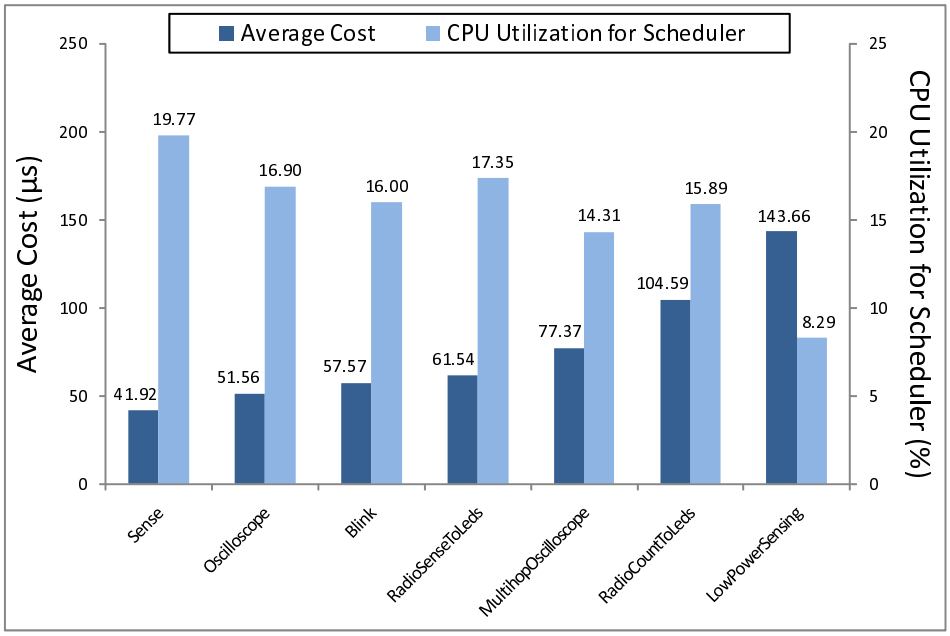

図 4.3　アプリケーション間のスケジューラのオーバヘッドとタスクの平均コスト

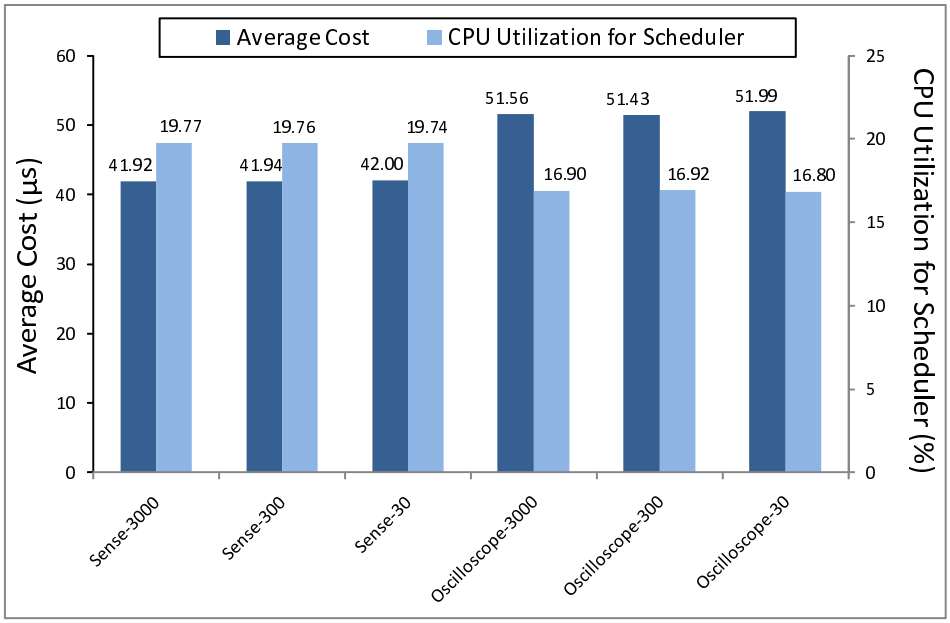

図 4.4　サンプリング間隔を変えた際のアプリケーション間のスケジューラのオーバヘッドとタスクの平均コスト

異なるアプリケーションにおけるタスク処理の平均コストとスケジューラのオーバヘッドを図 4.3 に示す．図 4.3 より，タスク処理のコストの平均が小さくなると，スケジューラ処理のオーバーヘッドの影響が大きくなることが分かる．例えば図 4.3 では Sense のタスクの平均コストが 41.92 μs と比較したアプリケーションの中で最も小さいものの，スケジューラのオーバヘッドは 19.77 % と最も大きくなる．タスクの粒度を小さくすればするほどスケジューラのオーバヘッドが増大することが示された．

同一アプリケーションでタスク実行の周期を変化させた際の比較を図 4.4 に示す．横軸のアプリケーション名の末尾の数字はサンプリングタスクの実行周期を ms 単位で表している．図 4.4 から，サンプリングの周期を変化させてもスケジューラのオーバヘッドの割合が変化しないことから，スケジューリングのオーバーヘッドは CPU の使用率ではなく，アプリケーションが含むタスクによって決まることがわかる．例えば Sense においてサンプリング間隔を 3000 ms，300 ms，30 ms と変化させてもスケジューラのオーバヘッドの割合は約 19.7 % と変化しない．

4.2.2　タスクの実行遅延

次に，計算量の大きなタスクを含めた際に発生する遅延の測定評価について示す．アプリケーションとして LowPowerSensing を用い，無線モジュールの電源管理のタスクの遅延時間の測定を行った．LowPowerSensing では，定期的にセンサ値を取得し，取得したセンサ値をフラッシュメモリに保存する．また，LowPowerSensing はロングプリアンブル型の MAC プロトコル [8] を用いており，LPL（Low Power Listening）によりシンクノードからのセンサ値送信のリクエストを検知する．センサノードはシンクノードからのリクエストを受け取ると，フラッシュメモリに蓄積したセンサ値をシンクノードに送信する．

LowPowerSensing の LPL 実行時の CPU と無線モジュールの動作を図 4.5 に示す．RF が ON となった後，シンクノードからの要求検知を行う `getCca` のタスクが起動する．`getCca` では，シンクノードからの送信要求が検出されない場合は終了時に RF OFF のタスクを `post` する．`getCca` の実行中にタイマイベントが発生し，ソフトウェアタイマのタスクが `post` されると，`getCca` 終了後にソフトウェアタイマ内に `event` として実装されたサンプリングのタスクが実行される．この時，タイマイベントである `event` の終了後に RF OFF のタスクが実行されるため，実行遅延が生じる．

ここで，サンプリングのタスク内に，0 ms，2 ms，4 ms，8 ms の大きさの計算処理を挿入し，タスクの実行開始の遅延を測定した．実験ではタスクの計算時間がタスク実行の

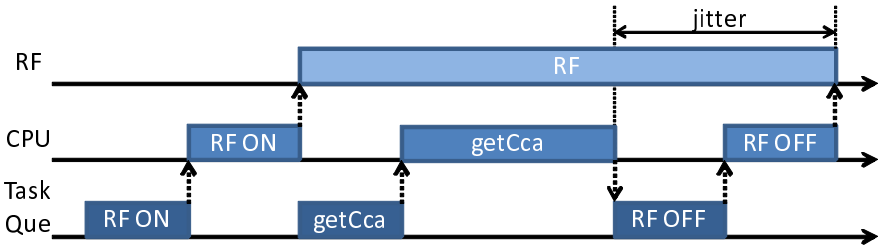

図 4.5　RF モジュールの電源管理

遅延に与える影響を明らかにするため，サンプリングの周期と LPL の周期を遅延の発生しやすい 50 ms，13 ms に固定した．実行遅延の最大値と平均値を図 4.6 に示す．図 4.6 によれば，最大遅延は計算処理に費やす時間よりも多く発生している．例えば，タスクの処理時間が 8 ms の場合，最大遅延は約 8.78 ms となる．この遅延の増加は，タスクが長い場合にはタスクの実行中に他の割り込みが入る確率が増加することに起因する．このようなタスクの実行遅延の不確定性により，RF OFF の実行に予測不能な遅延が生じる．平均値に関しても，計算処理が大きくなるに伴い増加しているが，計算処理が 2 ms の場合と 4 ms の場合で平均値の大きさが逆転している．これは RF OFF のタスクの実行遅延により LPL の周期に変化が生じていることに起因する．

このように，計算量の大きいタスクが存在すると，他のタスクの実行タイミングに不確定性が発生してしまう．例えば，無線通信モジュールの電力制御タスクの場合には，電源オフのタイミングが遅延することによる消費電力の増大や，電源オンのタイミングが遅延することによる予期せぬパケットロスなどが発生する．図 4.6 に示した 2 ms の場合と 4 ms の場合の平均値の大きさが逆転していることからも分かるとおり，タスクの遅延はあらかじめ予想できるものではなく，アプリケーションの構築において大きな問題となる．

4.3　駆動周波数低減による消費電力削減の効果に関する評価

前述のように，パーソナルコンピュータと無線センサネットワークのタスクは特性が異なる．また，タスク特性の相違により，最適な低消費電力化の手法にも違いが生じる．さらに，図 2.3 に示した PIC18LF2321 の例のように，一般的な CPU では動作が保証される電源電圧には下限が存在し，駆動周波数を下げても電源電圧を下げられない場合がある．このような理由から，無線センサネットワークのタスクモデルに沿って，Mulco の消

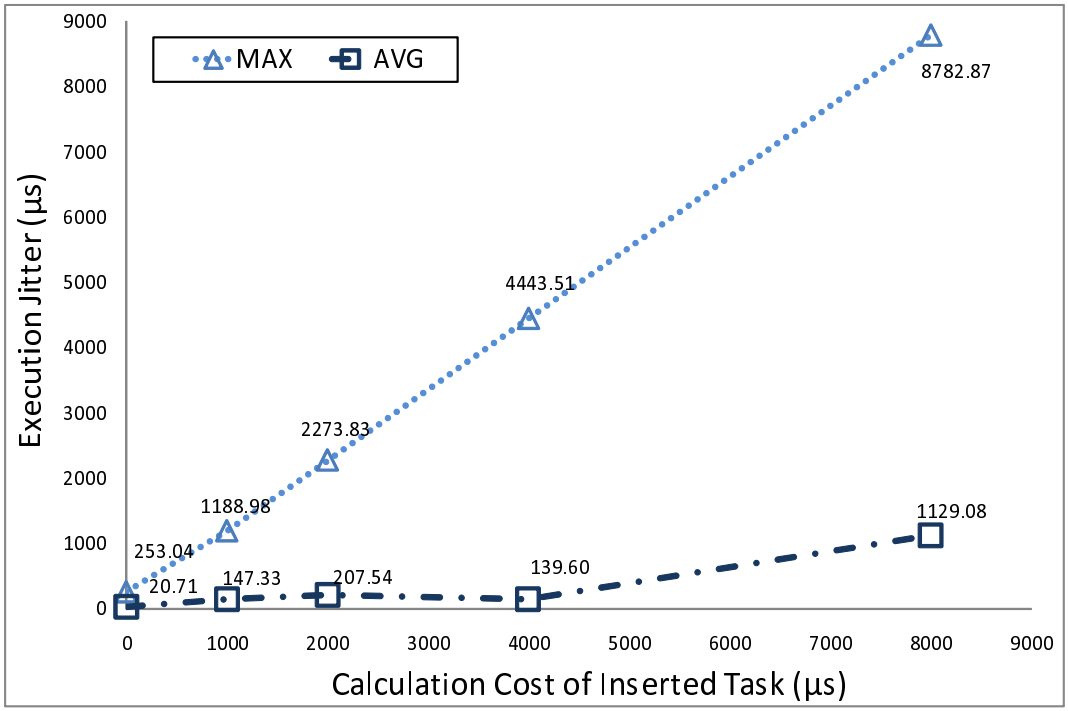

図 4.6　計算量を変化させた時のタスク実行遅延

費電力削減手法の有効性を確認する必要がある．本稿では，評価用のマルチコア CPU システムを実装し，無線センサネットワークのアプリケーションを模した 3 つのベンチマークを実行して，コア数による消費電力の変化を測定することで Mulco の消費電力削減手法の有効性を確認する．

4.3.1　マルチ CPU システム

評価用回路の外観を図 4.7 に，ブロック図を図 4.8 に示す．電源電圧，抵抗値，電圧計の値から，シングルコア，ダブルコア，トリプルコアそれぞれの場合の消費電力を算出することができる．評価用回路は 3 つの PIC18LF2321 と 1 つのリアルタイムクロックから構成される．各コアのセラミック振動子を交換することで，コア毎に駆動周波数を変えることができる．利用可能なセラミック振動子は，2 MHz，4 MHz，6 MHz，8 MHz，10 MHz，12 MHz，16 MHz，20 MHz の周波数の振動子である．電源電圧は安定化電源より供給し，電圧値は可変であるが，各コアの電圧は共通である．各コアの I/O ピンは，他のコアの割り込みピンに接続されており，相互にタスク処理要求が可能である．

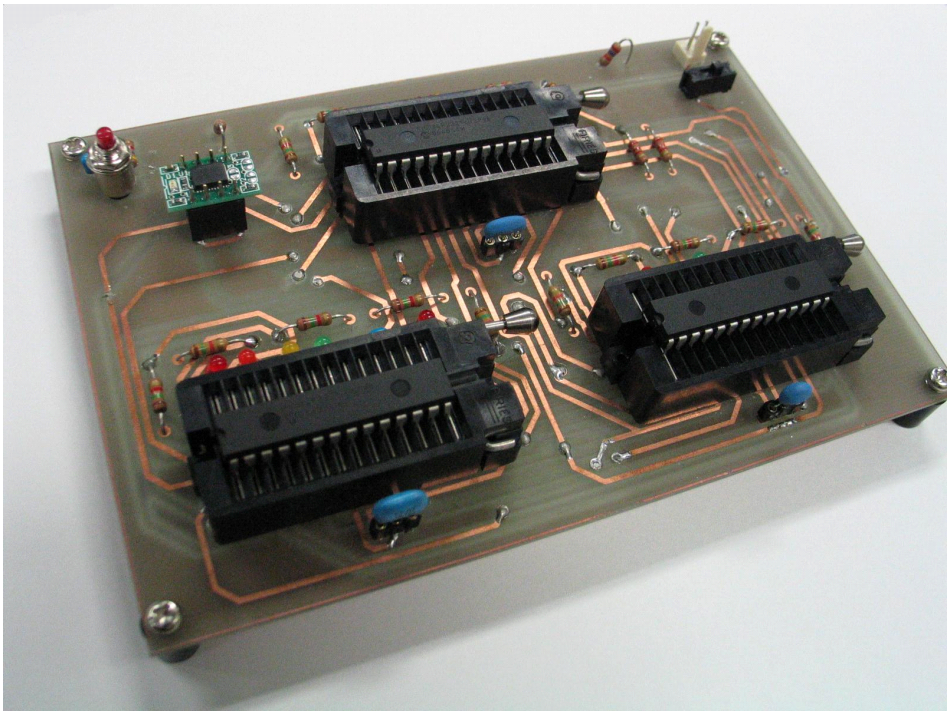

図 4.7　評価用マルチコア CPU システム

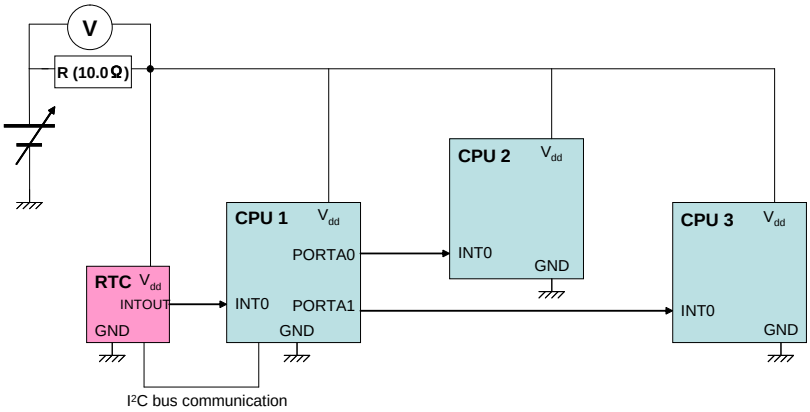

図 4.8　マルチコアシステムのブロック図

表 4.2 タスクの特性

Task	T（実行周期）	D（Deadline）	C（cost）
Sampling	10 msec	17 μsec	17 cycle
RMS	10 msec	10 msec	5957 cycle
PHY (RX)	26 μsec	26 μsec	39 cycle
CS	50 msec	416 μsec	465 cycle
CRC	10 msec	4.2 msec	503 cycle

4.3.2 ベンチマーク

無線センサネットワークのアプリケーション実行時のマルチコア CPU の消費電力を評価するために，各種サンプリング，計算処理，無線通信処理を含むタスク処理にかかる CPU のサイクル数を PIC18 シリーズの有するタイマ機能を用いて計測した．計測には無線センサネットワーク向けのハードウェアプラットフォームである PAVENET Module [6] を用いた．PAVENET Module は，評価用マルチ CPU システムと同じく PIC18 シリーズの CPU である，PIC18LF4620 [32] を具備する．また無線モジュールとして，Texas Instrumests 社の CC1000 [33] を具備する．測定したタスクのうち，ベンチマークが模したアプリケーションに含まれるタスクを以下に示す．

- Sampling：加速度データを 100 Hz で取得
- RMS：データの RMS 値 (Root Mean Square) を計算
- PHY (RX)：無線モジュールから送られる 1bit のデータの処理
- CS (Carrier Sense)：RSSI 値 (Received Signal Strength Indicator) を取得し，他のノードの送信の有無を検査
- CRC：16bit の CRC を計算し誤り検出を行う

表 4.2 に各タスクの実行周期（T），デッドライン（D），処理にかかる CPU のサイクル数（C）を示す．各タスクの時間制約を満たすために必要な駆動周波数 F_{min} は，PIC18LF2321 が 4 クロックで 1 命令を実行するため以下の式で求められる．

$$F_{min} = \frac{4 \times C}{D} \tag{4.1}$$

これらのタスク評価を用いて，実際のアプリケーションの実行時のタスクモデルを模した 3 つのベンチマーク（CSSR，SRuL，SRRC）に分けて評価を行った．

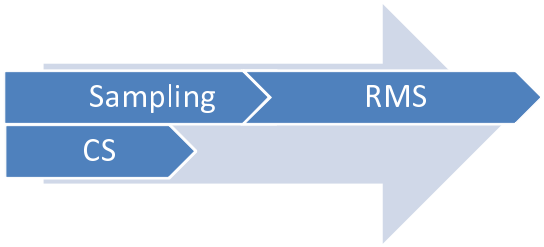

図 4.9　CSSR 実行の流れ

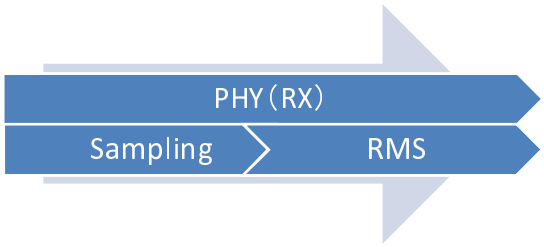

図 4.10　SRuL 実行の流れ

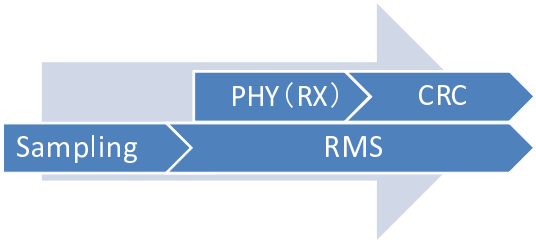

図 4.11　SRRC 実行の流れ

CSSR（Carrier Sense，Sampling，RMS）と SRuL(Sampling，RMS under Listening) は，加速度センサを用いてユーザのコンテキストを推定するアプリケーション [5] を模して実装した．[5] は，周期的に加速度を測定し，RMS 値を計算する．RMS 値が閾値を越えた場合，イベント通知を行う．パケットはマルチホップ通信によってイベントパケットを伝送する．MAC プロトコルとして，ロングプリアンブル型の MAC プロトコル [8] を採用し，プリアンブル長は 50 ms に設定されている．CSSR の実行の流れを図 4.9 に示す．

通常時は 50 ms の周期でキャリアセンスを行い，10 ms の周期で加速度データを取得し RMS 値の計算を行う．SRuL の実行の流れを図 4.10 に示す．CSSR の CS において，イベントパケットの受信を検知すると SRuL の動作に入る．SRuL では，イベントパケットを受信しつつ，サンプリングと RMS 値の計算を行う．

SRRC は地震モニタリング [34] のアプリケーションを模して実装した．SRRC の実行の流れを図 4.11 に示す．地震モニタリングでは正確に時刻同期のとれた 100 Hz のサンプリングを行う必要がある．このため，タスクは厳しい時間制約を持つ．ノード間で時刻同期をとるために，各ノードは 10 ms に一度時刻同期パケットを受信し CRC により誤り検出を行う．また，無線通信が加速度データの取得に影響を与えないようにサンプリング

表 4.3 タスクの割り当て (CSSR，SRuL)

CPU Number	Clock Frequency	task(s)
1 CPU	20 MHz	Sampling，RMS，CS，PHY
2 CPU	12 MHz	CS，PHY
	4 MHz	Sampling，RMS
3 CPU	6 MHz	PHY
	6 MHz	CS
	4 MHz	Sampling，RMS

表 4.4 タスクの割り当て (SRRC)

CPU Number	Clock Frequency	task(s)
1 CPU	8 MHz	Sampling，RMS，PHY，CRC
2 CPU	6 MHz	PHY，CRC
	4 MHz	Sampling，RMS
3 CPU	6 MHz	PHY
	4 MHz	CRC
	4 MHz	Sampling，RMS

中は無線通信をオフとする MAC プロトコルを採用している．

4.3.3 結果と考察

評価用回路を用いて 3 つのベンチマークのそれぞれについて，コア数を変化させ，消費電力を測定した．各コアへのタスクの割り当てと駆動周波数を，CSSR と SRuL は表 4.3 に，SRRC は表 4.4 に示す．SRRC と SRuL は同一のアプリケーション内の動作であるため，マルチコア CPU でのタスク割り当ては同じものとなる．電源電圧は，各コアの中で最も駆動周波数が高いもので動作が保証されている最低電圧を用いた．

図 4.12，4.13，4.14 に 3 つのベンチマークについて，時系列での電流値の変化をシングルコア CPU，ダブルコア CPU，トリプルコア CPU それぞれの場合について示す．シングルコア CPU がマルチコア CPU と比較して多くの電力を消費していることが分かる．また，図 4.13 よりシングルコア CPU では，PHY タスクの動作中はタスク処理要求間の時間が短いためにスリープすることができないことが分かる．

図 4.15 に 3 つのベンチマークそれぞれのシングルコア CPU，ダブルコア CPU，トリ

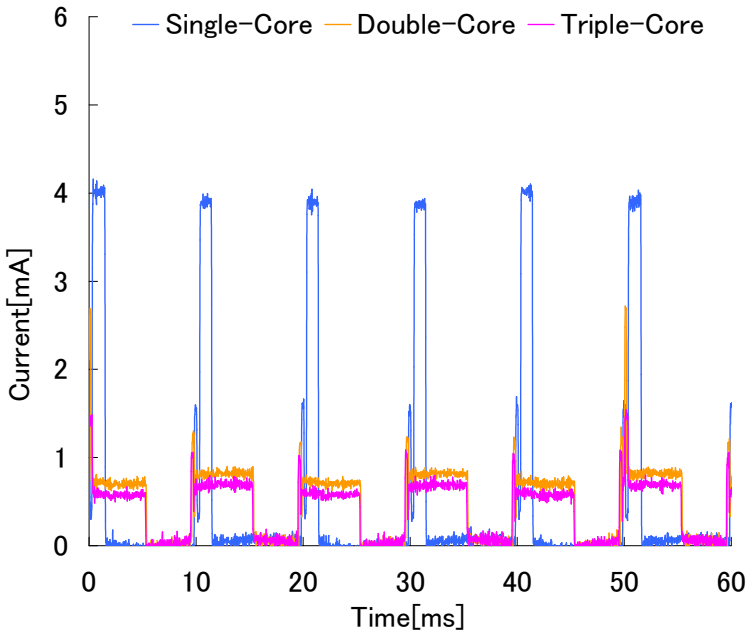

図 4.12　CSSR 実行時の電流値

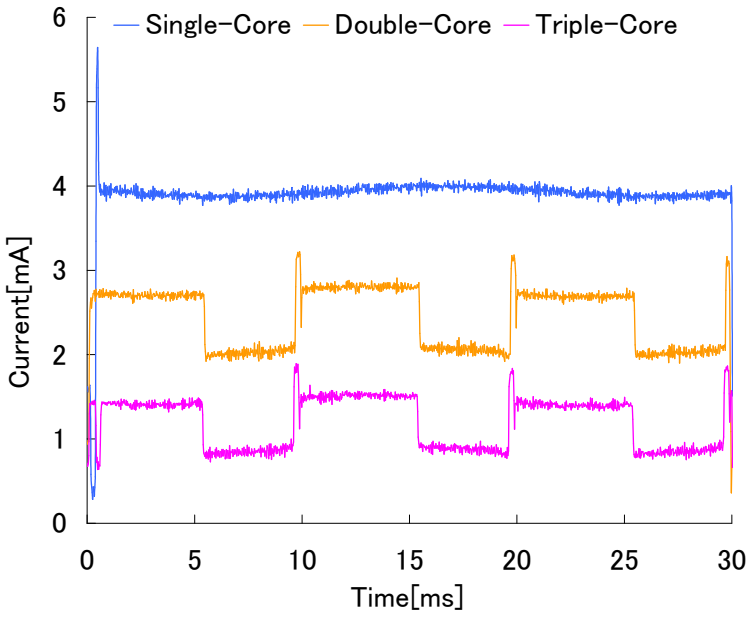

図 4.13　SRuL 実行時の電流値

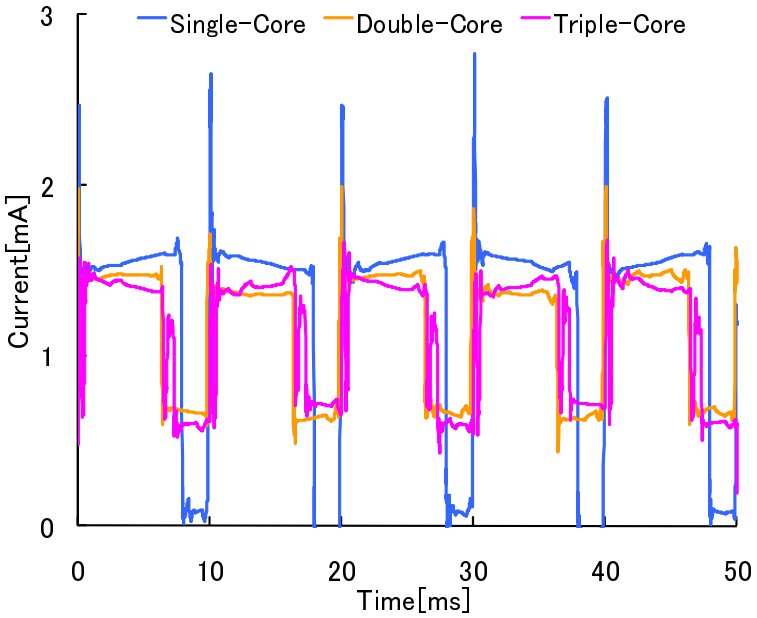

図 4.14　SRRC 実行時の電流値

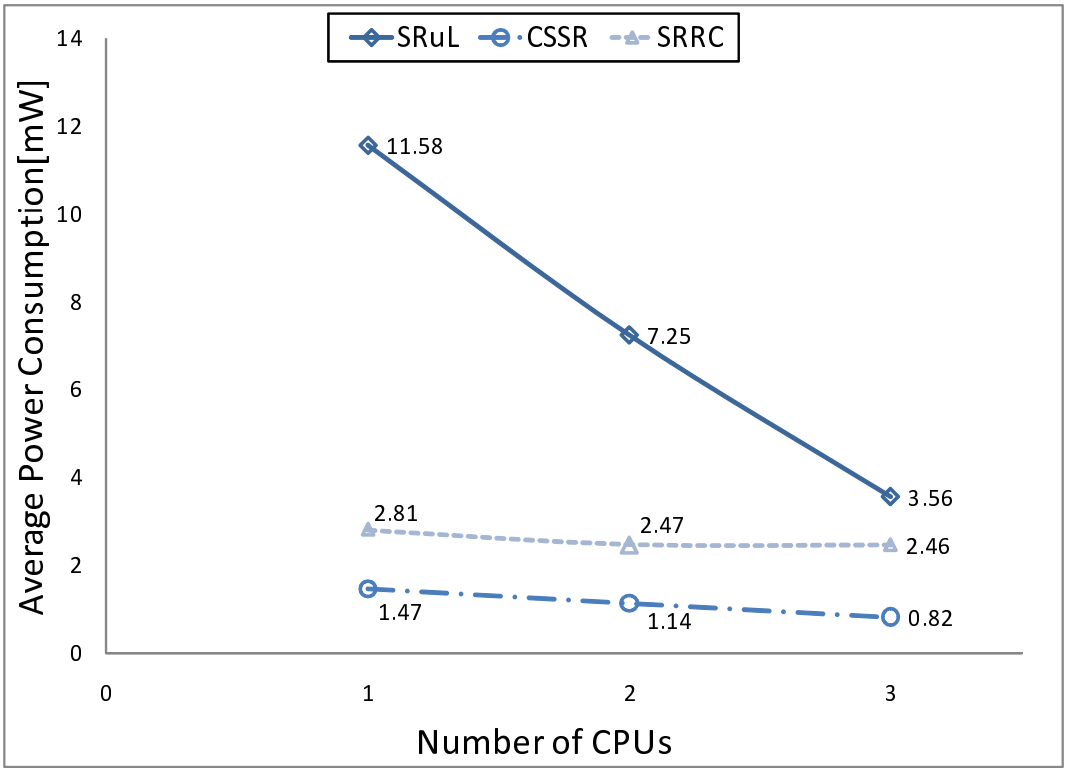

図 4.15　コア数による消費電力の変化

プルコア CPU の場合の平均消費電力を示す．図 4.15 によれば，CPU のマルチコア化により消費電力が削減されていることが分かる．本稿の実験においては各コアに与えられる電源電圧は共通であるが，コア毎に駆動周波数に合わせた適切な電源電圧を選択することができれば，さらなる消費電力の削減が見込まれる．また，SRuL はマルチコア化により各コアの駆動周波数を大きく削減することができ，その結果，消費電力を大幅に削減することに成功している．本論文の実験では，シングルコア CPU とトリプルコア CPU を比較して 70 % の消費電力が削減された．一方で，SRRC では消費電力は削減されているものの，大幅な削減は確認されなかった．マルチコア化により各コアの駆動周波数を大きく下げることができるため，タスク処理要求の偏りが大きいアプリケーションほどマルチコア化による大きな消費電力の削減が見込まれる．

4.4　サイクル数低減による消費電力削減の効果に関する評価

コア間での通信量が大きい場合には，コア間通信のオーバーヘッドがスケジューラ排除による実行サイクル数削減の効果を打ち消し，消費電力が増加してしまう恐れがある．このような観点から，以下ではマルチコア化により CPU の実行サイクル数がどのように変化するかを解析し，マルチコア化による実行サイクル数削減の効果を確認する．また，アプリケーション間でのコア間通信量を比較することで，アプリケーションが含むタスクフローによる通信量の変化を検討する．

4.4.1　評価方法

複数のアプリケーションについて，シングルコア CPU を用いた場合とマルチコア CPU を用いた場合の実行サイクル数の比較を行い，マルチコア化により実行サイクル数を削減できることを示す．スケジューラ排除により削減された実行サイクルよりもコア間通信のオーバーヘッドが小さい場合，マルチコア化により実行サイクルを削減することができる．

測定対象として，TinyOS のサンプルアプリケーションである MultihopOscilloscope, LowPowerSensing，Oscilloscope を用いる．これら 3 つのアプリケーションは，サンプリング，マルチホップ通信，省電力制御などの無線センサネットワークに共通して含まれる機能で構成されており，実際のアプリケーションでマルチコア化によって実行サイクル数がどのように変化するのかを把握することができる．

シングルコア CPU では，汎用センサノードである MicaZ ノードを用いて実行サイク

ル数を実測する．具体的には，TinyOS でタスクごとに割り当てられる ID をタスクの実行開始時および終了時に汎用 I/O に出力しロジックアナライザで計測することで，実行サイクル数と各タスクの実行周期を求める．

マルチコア CPU では，通信量のサイクル数に与える影響を明確にするために，各コアが局所メモリと他コアとの通信用のメモリを有するアーキテクチャを想定する．通信用のメモリは自分以外のコアの数だけ用意し，通信用メモリ間をフルメッシュ接続する．つまりノードが N 個のコアで構成されてるとすると，$N \times (N-1)$ 個の通信用のメモリが必要となる．これにより，コア間でのデータ共有を全てコア間通信として見積もるとともに，通信メモリは容量が小さなメモリで構成されるため，回路規模増大の影響を無視できる程度に抑えて評価をおこなうことができる．タスク処理のコスト，コア間の通信量および頻度から実行サイクル数を算出する．コア数およびコアへのタスクの割り当ては，タスクの依存関係から，3.3 に示した手法に従って決定する．コア間でやりとりされるデータのサイズと実行周期の情報に基づいて，通信量が CPU での実行サイクル数に及ぼす影響を見積もるために，コア間通信としては比較的低速な 2 Mbps の通信でのオーバーヘッドを算出する．

4.4.2　結果と考察

表 4.5 に各アプリケーションが含む機能，シングルコア CPU とマルチコア CPU に共通して含まれる，タスクの処理に要する単位時間あたりの実行サイクル数を示す．例えば MultihopOscilloscope をマルチコア CPU で実現する場合には，Sampling，Receive，Send, Routing の 4 つの機能に分割し，それぞれの機能にコアを割り当てる．また，表 4.6 に，シングルコア CPU での各アプリケーションごとのスケージューリングのオーバーヘッドと，オーバヘッドを加味した単位あたりの実行サイクル数を示す．

マルチコア CPU では，コア間通信として，5 回のサンプリングごとに，Sampling のコアから Send のコアへ 18 Byte のコア間通信や，他のノードからのパケットを Forward する際に，Receive のコアから Send のコアへ 18 Byte のコア間通信等が生じる．これら生じる全てのコア間通信の平均通信量（Byte/s）を MicaZ の駆動周波数である 7.3728 MHz での単位時間あたりのサイクル数とみなしたものを，コア間通信のオーバーヘッド（Cycle/s）とする．表 4.7 に，マルチコア CPU での単位時間あたりのコア間通信量とサイクル数の見積もり，オーバーヘッドを加味した単位時間あたりの実行サイクル数を示す．

表 4.5，表 4.6，表 4.7 に示すとおり，シングルコア CPU とマルチコア CPU の単位時間あたりの実行サイクル数を比較すると，マルチコア化により単位時間当たり 1.8 % か

表 4.5 各アプリケーションのタスク処理コスト

アプリケーション	アプリケーションを構成する機能	タスク処理のコスト（Cycle/s）
MultihopOscilloscope	Sampling, Receive, Send, Routing	23,854
LowPowerSensing	Sampling, LPL, FlashWrite	40,835
Oscilloscope	Sampling，Send	28,661

表 4.6 シングルコア CPU におけるスケジューリングオーバーヘッド

アプリケーション	スケジューラオーバーヘッド（Cycle/s）	実行サイクル数（Cycle/s）
MultihopOscilloscope	2,071	25,952
LowPowerSensing	1,954	42,789
Oscilloscope	5,242	28,661

表 4.7 マルチコア CPU におけるコア間通信オーバーヘッド

アプリケーション	コア間通信量（Byte/s）	コア間通信オーバーヘッド（Cycle/s）	実行サイクル数サイクル数（Cycle/s）
MultihopOscilloscope	53.766	1,586	25,442
LowPowerSensing	0.688	20	40,856
Oscilloscope	11.221	331	23,750

ら 17.0 % の処理を削減することができる．Mulco ではシングルコアの場合と比較して駆動周波数を低減させ，電源電圧を低く設定することができるため，コア間通信を考慮しても Mulco は消費電力の削減に有効であると考えられる．

測定対象としたアプリケーションを比較すると，Oscilloscope においてはタスクスケジューラを除去することによるサイクル数削減の効果がコア間通信によるオーバーヘッドに対して相対的に大きいため，マルチコア化によりサイクル数を大きく削減できる．一方，MultihopOscilloscope では，Routing の機能を処理するコアが他のコアと頻繁にコア間通信を行うため，実行サイクル数削減の効果が小さい．このような場合，単純にコア間通信によりデータを共有するのではなく，共有メモリ等の仕組みを利用することで，データ共有のオーバーヘッドを削減することができると考えている．

4.5 おわりに

本章では Mulco の有効性を示すための 3 つの評価を行った．まず，シングルコア CPU の問題を明らかにするために，スケジューリングに要する電力と実行遅延の定量化を行った．スケジューリングはノード全体の 10 % 程度の消費しており，また，不確定な大きさの実行遅延が発生することも確認された．

また，CPU のマルチコア化により，駆動周波数を低減させ，電源電圧を下げることによる，消費電力削減の効果を測定した．結果としてタスク処理の偏りの大きなアプリケーションにおいて 70 % の電力が削減された．

さらに，CPU のマルチコア化によりスケジューラを排除することで，コア間通信を加味しても CPU の実行サイクル数を削減できることを示した．同時に，アプリケーションによっては，局所メモリのみを有するアーキテクチャは最適とはならず，共有メモリ等の機構を検討する必要があることを明らかにした．

第 5 章

実装評価に向けた検討

5.1 はじめに

現在，Mulco の実装を行いタスク割り当てやアーキテクチャに関して，詳細な検討を進めている．5.2 では Mulco のタスク割り当てやアーキテクチャに関する検討を行うためには実装した Mulco 上でアプリケーションを動作させて検討を進めることが必要であることを述べ，我々の検討方針を説明する．5.3 では，無線センサノード向けの CPU 開発プラットフォームとして我々が実装した Mulco ノードに関して述べる．

5.2 Mulco のデザイン方針

現在我々は，Mulco の実装を進めながらタスク割り当てやアーキテクチャに関して詳細な検討を行っている．これまで述べてきたように，無線センサノード向けのマルチコア CPU を設計する際には，無線センサネットワークのタスクの特異性を十分に考慮する必要がある．無線センサネットワークの適用先は多岐にわたり，様々なアプリケーションが検討されていることを考えると，前章で行ったような評価で無線センサネットワークを網羅するのは限界がある．

例えば，3.3.1 で述べたように，コア内で同時に複数のタスクの実行要求が発生しないようにコアへタスクを割り当てる Mulco のタスク割り当ての手法では，コアが処理する機能がアプリケーションによって一意に定まらない．タスク割り当てを決定する際には，コア間通信量や駆動周波数などにより左右される消費電力を基準に検討を行う必要がある．つまり，Mulco 上でアプリケーションを動作させて検討を行わなければならない．また 3.1 で述べた，メモリ配置や，共有メモリと局所メモリの使い分けや，ディスパッチャのハードウェア構成を決定する際にも，同様にアプリケーションを動作させての検討が必要となる．

以上のような観点から，現在 FPGA 上に Mulco の実装を行っている．FPGA 上へ実装することで，短期間でハードウェア構成を変更することが可能となり，試行錯誤的にアーキテクチャを決定することや，アーキテクチャ間での比較が容易となる．

現在，Mulco のコアとして，OpenCores.org [35] において開発された IP（Intellectual Property）コアである，AVR Core を用いて実装を行っている．AVR Core は MicaZ ノードに搭載されている ATmega128 と互換性のある ATmega103 を模して実装されている．ただし AVR Core には，ATmega128 の以下の機能が搭載されていない．

表 5.1 Mulco ノードの構成

モジュール	Supplier	型番	備考
CPU	ATMEL	ATmega128L [19]	取り外すことで FPGA と接続可能
RFM	Texas Instruments	CC2420DK [20]	電源回路付き開発ツール
Flash メモリ	ATMEL	AT45DB [21]	4 Mbit
シリアル IC	MAXIM	DS2401 [36]	一意の ID をシリアル出力
振動子	EPSON	SG-531G [37]	7.37 MHz
51 pin コネクタ	Hirose	DF9B-51S [38]	MicaZ のプログラマに接続可能

- SPI 通信
- ADC
- 16 bit タイマ

これらの機能を実装することで，FPGA 上で TinyOS を動作させて評価を行うことが可能となる．

5.3 Mulco ノード

前節で述べたように，Mulco のタスク割り当てやアーキテクチャに関して検討を進める際には，アプリケーションを動作させての検討が必要となる．そこで我々は，センサノード向けの CPU 開発用プラットフォームとして，Mulco ノードの実装を行った．Mulco ノードの上面の外観を図 5.1 に，下面の外観を図 5.2 に示す．また Mulco ノードが具備している主なモジュールを表 5.1 に示す．Mulco ノードは，CPU 部を取り外すことが可能で，64 pin コネクタを通じて FPGA と接続することができる．これにより，FPGA 上で設計した CPU をセンサノード上で動作させることが可能となる．

Mulco ノードの設計に際しては，既存の汎用センサノードとの互換性を保つことを念頭に置いた．具体的には，Mulco ノードは MicaZ ノードとの互換性を持つ．例えば，表 5.1 に示したモジュールは，一部を除き MicaZ と共通のものを使用している．これにより，MicaZ ノードとの比較による動作検証が可能となると同時に，既存のドライバやプログラマ等の資源を活用することで CPU の開発が容易となる．また，FPGA 上で動作するシングルコア CPU との比較を行う際に，TinyOS をそのまま利用できる．これまでに，MicaZ ノード用のプログラマを通じて Mulco ノードに搭載した ATmega128 にプログラ

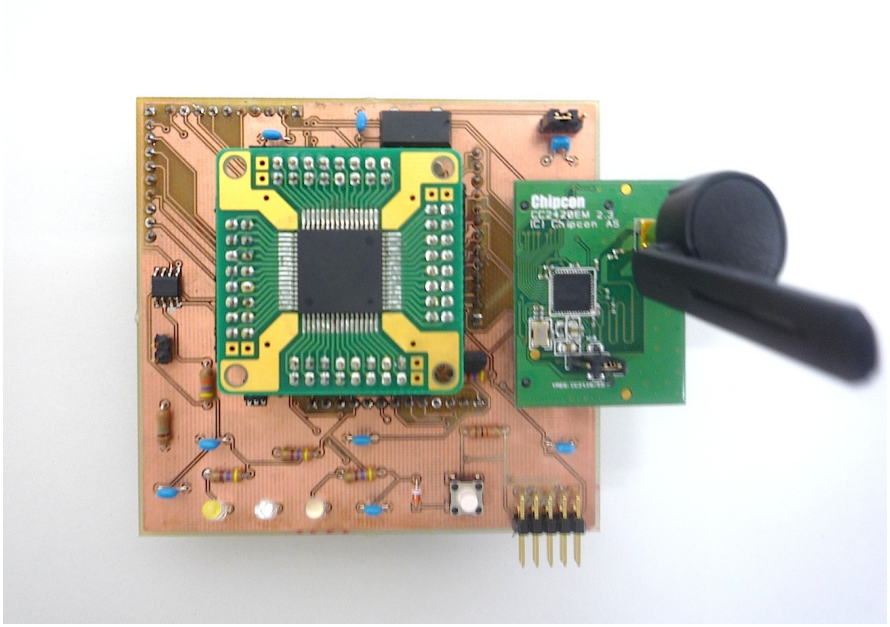

図 5.1 Mulco ノード（上面）

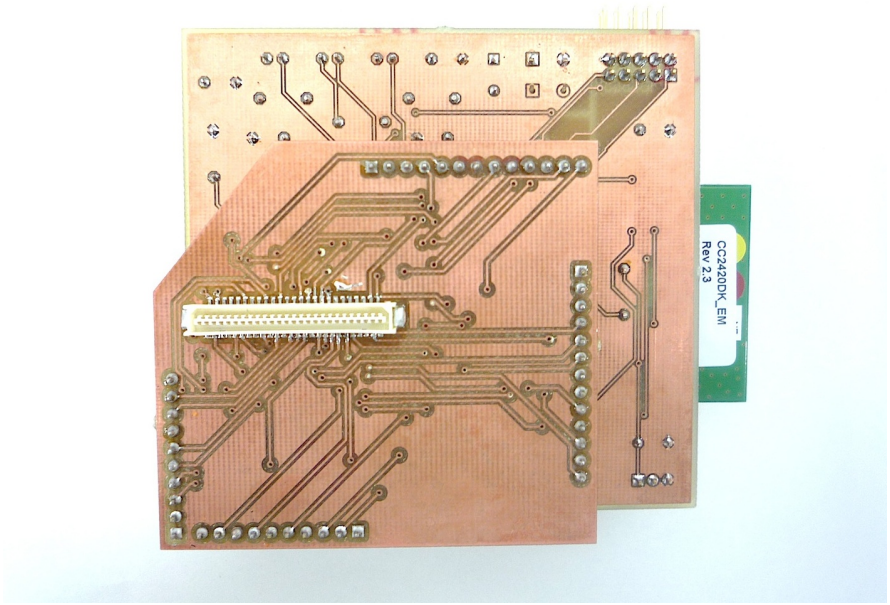

図 5.2 Mulco ノード（下面）

ミングを行えることや，TinyOS が Mulco ノード上で動作することを確認した.

5.4 おわりに

本章では，現在我々が進めている Mulco の研究方針に関して述べるとともに，無線センサノード向け CPU の開発プラットフォームとして Mulco ノードを紹介した.

今後は，Mulco の実装を行いタスク割り当てやアーキテクチャに関して，詳細な検討を進め，Mulco ノードに接続した FPGA 上でシングルコア CPU とマルチコア CPU の比較評価を行うことで，Mulco の実現を目指す.

第 6 章

結論

6.1 本研究の主たる成果

無線センサノードには，タスク処理の時間制約を満たすと同時に省電力で動作することが求められる．

本論文では，時間制約を満たすための実装を容易とし，消費電力を削減する，無線センサノード向けマルチコア CPU「Mulco」に関して述べた．その上で以下の 2 点を評価により示した．1 つ目は，現在無線センサノードに搭載されているシングルコア CPU では無線センサノードの要件を満たせないことである．スケジューリングに要する電力とタスク処理の実行遅延を実際のアプリケーションを用いた計測から定量化し，シングルコア CPU でアプリケーションを処理すると，許容できない問題が発生することを示した．2 つ目は，無線センサノードの CPU をマルチコア化することで，消費電力を削減できることである．駆動周波数を低減させ電源電圧を下げることによる消費電力削減効果と，スケジューラを排除することによる実行サイクル数削減による省電力効果の 2 点から評価を行った．以上により，無線センサノード向けの CPU としてマルチコア CPU が適していることが示された．

また本論文では上記評価に加えて，今後の FPGA を用いた Mulco の開発方針に関して述べるとともに，無線センサノード向け CPU の開発プラットフォームとして Mulco ノードを紹介した．多種多様なアプリケーションへの適用が期待される無線センサネットワークでは，アーキテクチャ決定に際する比較のために，開発期間の短縮が重要となる．アーキテクチャの変更が容易な FPGA 上に実装した CPU を，開発プラットフォーム上で実際のセンサノードとして動作させることで，より幅広いアプリケーションの要件を満たす CPU の開発が可能となる．

以上，評価による 2 点の結論と開発環境の提案をもって本論文の成果とする．これらの知見は，今後無線センサネットワークが多岐分野に渡って利用され，市場規模の拡大によりセンサノード向け CPU が開発される際の一助となるであろう．

謝辞

本研究を進めるにあたり，研究内容に関してご指導いただいたのみならず，かっこいい生き方，日本人としての誇りをご教授下さいました森川博之教授に深く感謝いたします．他の研究室と比較してキツイというイメージが強い森川研究室ですが，その一員として 3 年間過ごせて本当に良かったと思っております．研究環境の整備にご尽力下さった南正輝准教授，翁長久准教授に感謝いたします．お二人の存在は学生の立場からも頼もしく，安心して研究活動に専念できました．講師の今泉英明さんには，自由闊達な生き様というものを教えていただきました．安定を求めず自由を追求する生き方そのものが私のあこがれです．ありがとうございました．猿渡俊介助教には，真剣に取り組むということを教えて頂きました．私と意見がぶつかり言い合いになった時に，逃げずに真剣に考えて下さったことは，今でも忘れられません．また，何事も相対評価をしてしまい，自分に甘いところのある私に，その欠点を指摘し絶対的な価値基準を持つようにアドバイス下さったことは，何よりも真剣に私のことを考えて下さっていた証拠であり，正直驚きを隠せませんでした．感謝の念に堪えません．また，研究室の癒しであり続け，絶対的な安心感を与えてくれた秘書の川北敦子さん，出来るビジネスウーマンであり続け, 時には辛口な意見で私を鼓舞して下さった秘書の石崎智子さんに感謝いたします．

研究を進める上では，博士課程の鈴木誠さんに 3 年間ご指導頂きました．細かな点まで妥協を許さず真実を追求する姿は私にはとても真似のできないものです．研究を進めることができたことだけではなく，人間としての自分の弱さが少しはわかってきた気がします．ありがとうございました．

社会人博士の荒木さん，上條さん，山田さん，諸橋さん，山本さん，浅井さん，黒岩さんには，たくさんの刺激をいただきました．学生でありながらこんなにも社会を感じられる研究室はめったにないと思います．ありがとうございました．博士課程のスーさん，劉さん，石田さんの研究に対する誠意を見て，私は最後なんとか甘えずに耐え切れたと思います．ありがとうございました．博士課程のキムさんには主に飲み会でお世話になりました．実力主義の研究の世界で，時には儒教の教えに従うのも悪くないですね．

同期の林敏樹君と李慧さんには卒論配属から 3 年間お世話になりました．圧倒的に頭が

いいけれど優しすぎる故に損をしてしまう林君，圧倒的に頭が良い上に実はしたたかな李さん．2 人と同じ研究室に来られてよかった．ありがとう．同じく同期の西村亨輔君は修士課程から森川研にやってきて 2 年間を共に過ごしました．最初はそのプレゼンスの高さに圧倒されたけれど，なんとか西村君に負けたくないと思うなかで，人にはそれぞれに長所があり，それを武器に戦えば良いことを教えてもらいました．ありがとうございました．デラジャトエコリアント君，ティモティシトラスローレンス君，パンジャイタンフェルナンド君の同期インドネシア 3 人衆には，国民性の違いをまざまざと見せつけられると共に，そんな違いもすぐに乗り越え，分かり合えることを教えてもらいました．3 人と知り合えたことは私の人生にとって大きなプラスとなりました，ありがとう．

修士過程 1 年の岡村君，瀧口君，高木君，長縄君，陳君，ベクさん，君たちの学年は 10 年に一度の逸材揃いで，僕にとてつもないプレッシャーを与えてくれました．修士課程での成果は，全部君たちのプレッシャーに負けじと出したものです．ありがとう．そしてこれからの森川研究室をよろしく．卒論配属でやってきた岩本君，中嶋君，角田君は三者三様の個性の持ち主で，見ているだけで面白かった．君たちが研究に打ち込む若々しい姿は，私に初心を思い出させてくれました．ありがとう．

パベルプピレフさん，川西直さん，渡部克弥さん，町田啓太さんをはじめ，多くの OB の方々に大変お世話になりました．OB との交流が絶えることのない森川研究室は本当に良い研究室だと思います．この研究室に来られて良かった．皆様本当ににありがとうございました．

最後に，遠く三重の地から支え続けてくれた家族と，私の学生時代を見守り続けてくれた松尾玲子さんに感謝の意を表し，謝辞といたします．

2010 年 2 月 8 日

参考文献

[1] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for "smart dust". In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking (MobiCom'99)*, Washington, USA, 1999.

[2] P. Volgyesi, G. Balogh, A. Nadas, C. B. Nash, and A. Ledeczi. Shooter localization and weapon classification with soldier-wearable networked sensors. In *Proceedings of the 5th International Conference on Mobile Systems (MobiSys'07)*, San Juan, Puerto Rico, 2007.

[3] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN'07)*, Massachusetts, USA, 2007.

[4] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, Georgia, USA, 2002.

[5] Y. Kawahara, H. Kurasawa, and H. Morikawa. Recognizing user context using mobile handsets with acceleration sensors. In *International Conference on Portable Information Devices (PORTABLE'07)*, Florida, USA, 2007.

[6] 猿渡 俊介, 鈴木 誠, 水野 浩太郎, and 森川 博之. 無線センサノード向けハードリアルタイムオペレーティングシステムの設計. 情報処学会研究報告, ユビキタスコンピューティングシステム研究会 (UBI-13-29). 2007.

[7] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors *ACM SIGPLAN Notices*, 35(11):93–104, 2000.

[8] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wire-

less sensor networks. In *Proceedings of the 2nd ACM Conference on Embedded Network Sensor Systems (SenSys'04)*, MaryLand, USA, 2004.

[9] V. Shnayder, M. Hempstead, B. Chen, GW. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd ACM Conference on Embedded Network Sensor Systems (SenSys'04)*, Maryland, USA, 2004.

[10] C. Hartung, R. Han, C. Seielstad, and S. Holbrook. Firewxnet: a multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *Proceedings of the 4th international conference on Mobile systems, applications and services (MobiSys'06)*, Uppsala, Sweden, 2006.

[11] P. Levis, D. Gay, V. Handziski, J. Hauer, B. Greenstein, M. Turon, J. Hui, K. Klues, C. Sharp, R. Szewczyk, J. Polastre, P. Buonadonna, L. Nachman, G. Tolle, D. Culler, and A. Wolisz. T2: A second generation os for embedded sensor networks. In *Technical Report TKN-05-007, Telecommunication Network Group, Technische Universitat Berlin*, 2007.

[12] C. C. Han, R. Kumar, R. Shea, E. Kohler, and M. B. Srivastava. A dynamic operating system for sensor nodes. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys '05)*, Washington, USA, 2005.

[13] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN '04)*, Florida, USA, 2004.

[14] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys '06)*, Colorado, USA, 2006.

[15] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. *ACM Mobile Networks Applications (MONET), Special Issue on Wireless Sensor Networks*, 10(4):536–579, 2005.

[16] L. Gu and J. A. Stankovic. t-kernel: Provide reliable os support for wireless sensor

networks. In *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys ' 06)*, Colorado, USA, 2006.

[17] D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, California, USA, 2003.

[18] *MicaZ Data Sheet*:
`http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf`

[19] *ATmega128L Data Sheet*:
`http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`

[20] *CC2420 Data Sheet*:
`http://focus.ti.com/lit/ds/symlink/cc2420.pdf`

[21] *AT45DB Data Sheet*:
`http://www.atmel.com/dyn/resources/prod_documents/doc3595.pdf`

[22] C. Park, Q. Xie, P. Chou, and M. Shinozuka. Duranode: wireless networked sensor for structural health monitoring. In *Proceedings of the 4th IEEE International Conference on Sensors*, California, USA, 2005.

[23] D. A. Patterson and J. L. Hennessy. Computer Architecture: A Quantitative Approach. *Morgan Kaufmann Pub*, 2006.

[24] *PIC18F2331 Data Sheet*:
`http://ww1.microchip.com/downloads/DeviceDoc/39616C.pdf`

[25] L. Necchi, L. Lavagno, D. Pandini, and L. Vanzago. An ultra-low energy asynchronous processor for wireless sensor networks. In *Proceedings of 12th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'06)*, Grenoble, France, 2006.

[26] V. Ekanayake, C. Kelly IV, and R. Manohar. An ultra low-power processor for sensor networks. *ACM SIGOPS Operating Systems Review*, 38(5):27–36, 2004.

[27] C. Kelly IV, V. Ekanayake, and R. Manohar. Snap: a sensor-network asynchronous processor. In *Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems*, BC, Canada, 2003.

[28] V. Ekanayake, C. Kelly IV, and R. Manohar. Bitsnap: Dynamic significance compression for a low-energy sensor network asynchronous processor. In *Proceedings*

of the 11th International Symposium on Asynchronous Circuits and Systems, New York, USA, 2005.

[29] J. Kao, S. Narendra, and A. Chandrakasan. Subthreshold leakage modeling and reduction techniques. In *Proceedings of Conference on Computer-Aided Design*, California, USA, 2002.

[30] L. Nazhandali, M. Minuth, B. Zhai, J. Olson aud J. Austin, and D. Blaauw. A second-generation sensor network processor with application-driven memory optimizations and out-of-order execution. In *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, California, USA, 2005.

[31] S. Hanson, B. Zhai, M. Seok, B. ClineK. Zhou, M. Singhal, M. MinuthJ. Olson, L. Nazhandali, T. Austin, D. Sylvester, and D. Blaauw. Exploring variability and performance in a sub-200-mv processor. *IEEE Journal of Solid-State Circuits*, 43(4):881, 2008.

[32] *PIC18LF4620 Data Sheet*:
`http://ww1.microchip.com/downloads/en/DeviceDoc/39626e.pdf`

[33] *CC1000 Data Sheet*:
`http://focus.tij.co.jp/jp/lit/ds/symlink/cc1000.pdf`

[34] M. Suzuki, N. Kurata, S. Saruwatari, and H. Morikawa. Demo abstract: A high-density earthquake monitoring system using wireless sensor networks. In *Proceedings of the 2nd ACM Conference on Embedded Network Sensor system (Sensys'07)*, Sydney, Australia, 2007.

[35] *OpenCores.org*:
`http://www.opencores.org`

[36] *DS2401 Data Sheet*:
`http://datasheets.maxim-ic.com/en/ds/DS2401.pdf`

[37] *SG-531P Data Sheet*:
`http://www.datasheetcatalog.org/datasheet/epson/SG-531P.pdf`

[38] *DF9 Data Series Sheet*:
`http://www.hirose.co.jp/catalogj_hp/j54000041.pdf`

発表文献

[1] 大原 壮太郎, 鈴木 誠, 猿渡 俊介, 森川 博之. “無線センサノード向けマルチコア CPU のコンセプト検証”, 電子情報通信学会総合大会, B-20-33, 2008.

[2] S. Ohara, M. Suzuki, S. Saruwatari, and H. Morikawa. “A Prototype of A Multi-Core Wireless Sensor Node for Reducing Power Consumption“ in Proceedings of The 2008 International Symposium on Applications and the Internet (SAINT 2008), Turku, Finland, 2008.

[3] 大原 壮太郎, 鈴木 誠, 猿渡 俊介, 南 正輝, 森川 博之. “無線センサノード向けマルチコア CPU の低消費電力性に関する初期的検討”, 電子情報通信学会技術研究報告, ユビキタスセンサネットワーク研究会, USN2008-59, 2008.

[4] 大原 壮太郎, 鈴木 誠, 猿渡 俊介, 南 正輝, 森川 博之. “無線センサノードにおけるシングルコア CPU の問題点に関する定量的評価”, 情報処理学会研究報告, 情報処理学会研究報告, UBI-23-6, 2009.

[5] 鈴木 誠, 大原 壮太郎, 猿渡 俊介, 倉田 成人, 南 正輝, 森川 博之. “無線センサネットワークにおける時刻同期誤差伝播の解析”, 電子情報通信学会技術研究報告, ユビキタスセンサネットワーク研究会, USN2009-15, 2009.

[6] S. Ohara. “A Multi-Core CPU for Reducing Scheduling Overhead in Wireless Sensor Nodes”, Asian Workshop on Ubiquitous and Embedded Computing (AWUEC) 2009, Beijing, China, 2009.

[7] Y. Chen, S. Ohara, M. Suzuki, S. Saruwatari, T. Higata, M. Minami, and H. Morikawa. “Initial Discussion of Deer Detecting System for Railways using Wireless Sensor Network”, The 2nd Asia-Europe Workshop on Ubiquitous Computing 2009 (AEWUC’09), Shizuoka, Japan, 2009.

[8] 大原 壮太郎, 鈴木 誠, 猿渡 俊介, 南 正輝, 森川 博之. “無線センサノード向けマルチコア CPU におけるタスク分割手法”, 電子情報通信学会ソサイエティ大会, B-20-20, 2009.

[9] 鈴木 誠, 大原 壮太郎, 猿渡 俊介, 今泉 英明, 倉田 成人, 南 正輝, 森川 博之. “無線

センサネットワークにおける時刻同期の誤差分布に関する検討”，電子情報通信学会ソサイエティ大会，B-20-6，2009.

[10] 大原 壮太郎, 鈴木 誠, 猿渡 俊介, 森川 博之．“無線センサノート向けマルチコアCPU におけるコア間通信に関する検討”，電子情報通信学会総合大会，B-20-58，2010.（発表予定）