Vol.2012-EMB-26 No.4
2012/9/10

周期実行システムにおける省電力スケジューリングの初期検討

岡本 和也[1,a)]　薦田 登志矢[1]　中田 尚[1]　三輪 忍[1]
佐藤 洋平[2]　植木 浩[2]　林越 正紀[2]　清水 徹[2]　中村 宏[1]

概要：マイクロプロセッサを備えたセンサであるスマートセンサは，周囲の状況を周期的にサンプリングし，センシングした結果に簡単な処理を施し，その結果をメインのシステムへ送信する，周期的リアルタイムシステムの一種である．ただし，一般的なリアルタイムシステムとは異なり，入力データのサンプリング周期とデータ送信（デッドライン）の周期が必ずしも一致するわけではなく，一般には，後者の周期が前者の周期よりもはるかに大きい．そのため，データの入力間隔に合わせてシステムがデータを処理するのではなく，データを一旦バッファに格納しておき，いくつかのデータがバッファに溜まったらシステムを起動して処理を行い，処理が完了したらシステムをシャットダウンする，という制御が可能である．このような制御を行えば，DVFS や動的電源制御などの従来の制御を行う場合よりも，省電力なシステムを実現できると考えられる．本稿では，上述の制御を行うシステムのモデルを提案し，既存の制御手法と比較する．評価の結果，既存手法と比べて消費エネルギを 79.6% 削減できることがわかった．

1. はじめに

近年，センサとマイクロプロセッサを一体化した，**スマートセンサ**が普及している．例えば，生体埋め込み型のテレメトリ装置やペースメーカ，自動車エンジンのアンチノッキングシステム等，さまざまなシステムにおいてスマートセンサは使用されている．スマートセンサは，今後ますます普及していくものと期待されている．

スマートセンサにおいては，プロセッサの消費する電力は決して無視できるものではない．センサノードにもよるが，プロセッサの消費電力は，ノード全体のそれの数割に達することもある [7]．そのため，スマートセンサのさらなる長寿命化，および，応用範囲拡大のためには，プロセッサの消費電力を如何に減らすかが鍵となる．

スマートセンサは，一種の**周期的なリアルタイムシステム**（周期実行システム）と見なすことができる．スマートセンサは，それに備え付けられたセンサによって，画像，温度，圧力等の周囲の状況を周期的にサンプリングする．そして，サンプリング結果に対し，内蔵するプロセッサによって補正やフィルタリングなどの処理を施す．処理の結果は，センシング結果を処理するメイン・システムへと周期的に送られる．ただし，一般的なリアルタイムシステムとは異なり，入力データのサンプリング周期とデータ送信（デッドライン）の周期が必ずしも一致するわけではない．通常は，後者の周期が前者の周期よりもはるかに大きい（**時間制約がゆるい**）．

一般のリアルタイムシステムにおいては，単純に **DVFS**（Dynamic Voltage and Frequency Scheduling）[3], [8] あるいは **DPM**（Dynamic Power Management）[2] を行うことで，システムの消費電力を削減してきた．DVFS は，プロセッサの負荷が低い時に電圧および周波数を低下させることで，プロセッサの消費電力を減らす方法である．リアルタイムシステムにおいては，タスクがデッドラインまでに完了するように，できるだけ低い電圧と周波数で動作させることによって，低消費電力化が図れる．一方，DPM は，プロセッサがアイドル状態の時にプロセッサ全体の電源を遮断することで，電力を削減する方法である．リアルタイムシステムにおいては，タスクが完了してから次のタスクを処理するまでの間，プロセッサの電源を遮断することで消費電力を削減できる．ただし，前者の方法はリーク電力を削減する効果はなく，また後者の方法は電源の ON/OFF によってエネルギ・オーバヘッドが発生してしまう．

前述のように，スマートセンサの場合は，一般的なリア

[1] 東京大学
The University of Tokyo
[2] ルネサス エレクトロニクス株式会社
Renesas Electronics Corporation
[a)] okamoto@hal.ipc.i.u-tokyo.ac.jp

ルタイムシステムとは異なり，デッドラインの周期はタスクの周期と比べて大きい．そのため，データの入力間隔に合わせてプロセッサがデータを処理するのではなく，データを一旦バッファに格納しておき，いくつかのデータがバッファに溜まったらプロセッサを起動して処理を行い，処理が完了したらプロセッサをシャットダウンする，という制御が可能である．このような制御を行えば，デッドラインを守りつつ電源 ON/OFF の機会を削減できるため，単純に DPM を行う場合よりも消費電力を削減できると考えられる．また，DPM を行うことでリーク電力も削減できるため，DVFS を適用した場合よりも低消費電力になると考えられる．

本稿では，ゆるい時間制約を考慮したタスクスケジューリングを行うことで，DPM によるエネルギオーバヘッドを削減し，より大きなエネルギ削減を達成する省電力スケジューリング手法を提案する．さらに，提案手法の有効性を定量的に論じるため，提案手法の消費エネルギモデルを提案し，これと DVFS，DPM のエネルギモデルを用いて，あるセンサ・ノードに各手法を適用した場合のエネルギ削減率を評価した．その結果，提案手法は既存手法と比べて消費エネルギを 79.6% 削減できることがわかったので報告する．

本稿は次のように構成されている．次章で一般のリアルタイムシステムを対象とした既存の省電力化手法について述べ，続く 3 章でゆるい時間制約のシステムを対象とした新しい省電力化手法を扱う．4 章で実際のアプリケーションを想定したパラメタを用いて評価を行い，提案構成の有効性について考察する．5 章でまとめと今後の展望について述べる．

2. 既存の省電力化手法とその問題点

リアルタイムシステムにおいて，ゆるい時間制約に特化しない既存の省電力化手法である DVFS および DPM の問題点について述べる．まず，タスクモデルおよびプロセッサモデルを導入して，各手法の定量的な比較のため，最悪ケースのエネルギモデルを導出する．その上で，本研究のアプローチについて述べる．

2.1 タスクモデル

タスクは一定時間 T_i で入力される各データに対して，最悪ケースの場合にサイクル数 C を費やす処理を施す．各データに対する処理を**タスクインスタンス**，あるいは単純に**インスタンス**と呼ぶ．インスタンスの実行には時間制約が存在し，T_d の時間以内に処理を終了する必要がある．本稿では $T_i < T_d$ となるアプリケーションを対象とし，この不等式を満たす T_d を**ゆるい時間制約**と呼ぶ．

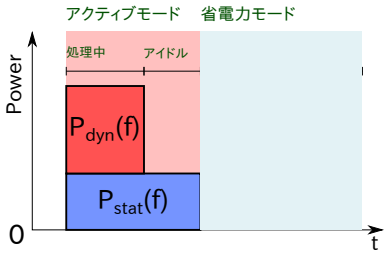

図 1 プロセッサにおける電力消費の概念図

2.2 プロセッサモデル

DVFS が可能で，アクティブモードと省電力モードを持つプロセッサを仮定する．アクティブモードはデータを処理できる状態，省電力モードは処理が行えないがスタティック電力の小さな状態である．処理時には，DVFS によって，$f_{min} \leq f \leq f_{max}$ の範囲で任意の周波数 f を用いることができる．2.3 節での議論により，処理時の周波数をタスク全体で一定とする．各インスタンスの処理時間はサイクル数，および周波数の逆数に比例するとする．f_{max} はプロセッサのとりうる最高周波数である．f_{min} は，プロセッサのとりうる最低周波数 f_{min}^{proc} に対して $f_{min} = \min(f_{min}^{proc}, C/T_i)$ と定義する．C/T_i は処理を T_i でちょうど終わらせることのできる周波数であり，これより小さくすると，制約時間までに終了できないインスタンスが生じてしまう．また，各周波数に対して，対応する電圧が決まっているものとする．

図 1 はプロセッサにおける電力消費の概念図である．横軸に時間，縦軸に電力をとり，電力消費の要因によって色分けされている．ある周波数 f（および対応する電圧）で処理を行なっている時にプロセッサで消費されるダイナミック電力を $P_{dyn}(f)$，スタティック電力を $P_{stat}(f)$ と表記する．なお，$P_{dyn}(f)$，$P_{stat}(f)$ の値は，省電力モードの状態でも消費される電力を 0 とし，それを基準とした値とする．すなわち，タスク実行中には $P_{dyn}(f) + P_{stat}(f)$（+ 省電力モードの電力 = 0）が消費され，アクティブ状態でアイドルしていると $P_{stat}(f)$（+ 省電力モードの電力 = 0）の電力が消費され続ける．

$P_{dyn}(f)$，$P_{stat}(f)$ は増加凸関数または非減少線形関数であるとする．また，$P_{dyn}(0) = 0$ とする．これは以下の理由による．一般に周波数に応じて電圧も変更する際には，ダイナミック電力もスタティック電力も周波数に対して増加凸関数となる．電圧を最大周波数で動作できる値のまま変更しない場合，ダイナミック電力は増加線形関数，スタティック電力は f によらず一定の値をとる．この仮定の下では，タスク実行中に消費される電力の合計 $P_{dyn}(f) + P_{stat}(f)$ は増加凸関数または非減少線形関数となる．

2.3 DVFS

DVFS では，プロセッサの動作中に周波数（およびそれに付随して電圧）を変更することができる．しかし，以降では周波数（および電圧）はタスクの実行中に一定の値を用いる．ある処理を実行中に周波数を変更することは，本稿で対象としている最悪ケースの消費エネルギを増大することになるためである．その理由を以下に示す．処理全体のうち，割合 x の部分を周波数 f_i，残りの部分を f_{i+1} で実行する場合を考える．この時処理全体の実行時間は，その平均周波数 $f_{avr} := xf_i + (1-x)f_{i+1}$ で処理した場合と等しい．ここで，任意のサイクル数の処理について，全体を f_{avr} で実行した時の消費電力が f_i と f_{i+1} を途中で切り替えた時の平均の消費電力よりも少なければ，タスク処理中の周波数変更が適切でないことが示される．ある周波数 f で実行中のプロセッサの消費電力を $P(f)$ とおく．f_i，f_{i+1} を途中で切り替える時の平均の消費電力は，$xP(f_i)+(1-x)P(f_{i+1})$ と表される．f_{avr} で実行中の電力は $P(f_{avr}) = P(xf_i + (1-x)f_{i+1})$ である．ここで電力が周波数に対して増加凸関数であるとき，増加関数の定義から以下の等式が得られる．

$$\begin{aligned} P(f_{avr}) &= P(xf_i + (1-x)f_{i+1}) \\ &\leq xP(f_i) + (1-x)P(f_{i+1}) \end{aligned} \tag{1}$$

すなわち，一定の周波数で処理を行ったほうが，途中で周波数を変更して処理するよりも電力効率が良い．また，電力が周波数に対して非減少線形関数であるとき式 1 において等号が成り立つが，やはり一定の周波数で処理をすべきである．これは周波数の切り替えにエネルギオーバヘッドが生じることによる．

本稿では，プロセッサのダイナミック電力とスタティック電力を，周波数に対して増加凸関数あるいは非減少線形関数であると仮定している．また，実行時に消費される電力の合計 $P_{dyn}(f) + P_{stat}(f)$ も増加凸関数あるいは非減少線形関数となる．以上の議論より，以降では処理の実行中に周波数変更を行わないものとする．

動作周波数を自由に選べるが，DPM やバッファリングができない構成を，構成 F と表記する．構成 F で周波数 f を用いた時に，1 インスタンスあたりに消費されるエネルギ $EPI_F(f)$ は以下のように表される．

$$EPI_F(f) = P_{dyn}(f) \cdot (C/f) + P_{stat}(f) \cdot T_i \tag{2}$$

第 1 項がダイナミックエネルギ，第 2 項がスタティックエネルギに相当する．ダイナミックエネルギは動作中，すなわち C/f の時間のみ生じ，スタティックエネルギはインターバル時間の間生じ続ける．

2.4 DPM

DPM は，パワーマネジャを追加することで，動作中に省電力モードへの移行を可能にする．省電力状態への移行およびアクティブモードへの復帰には合計で E^{dyn}_{DPM} のエネルギオーバヘッドが存在する．またパワーマネジャのリーク電流によるエネルギなど，常に P^{stat}_{DPM} のオーバヘッド電力が生じる．2.2 節において省電力モード中の電力をシステムの消費電力の基準としているので，その間の電力消費は 0 である．省電力モードで待機することで削減されるエネルギよりも E^{dyn}_{DPM} の方が大きい場合には，アクティブな状態のまま待機したほうが消費エネルギの総計は少なくなる．しかし，本稿では常にアクティブモードの構成も比較対象として扱うため，処理の前後で必ず省電力モードを用いることとする．

周波数を選ぶことができ，DPM も可能な構成を，構成 FP と表記する．構成 FP で 1 インスタンスあたりに消費されるエネルギ $EPI_{FP}(f)$ は以下のように表される．

$$\begin{aligned} EPI_{FP}(f) &= (P_{dyn}(f) + P_{stat}(f)) \cdot (C/f) \\ &\quad + E^{dyn}_{DPM} + P^{stat}_{DPM} \cdot T_i \end{aligned} \tag{3}$$

第 1 項がプロセッサのダイナミックエネルギおよびスタティックエネルギに相当する．DPM によって，スタティックエネルギも動作中（C/f の時間）にのみ生じる．第 2 項は DPM の状態遷移に要するエネルギ，第 3 項は DPM の制御を行う追加回路等によるスタティックエネルギである．

2.5 従来手法の問題点と本研究のアプローチ

我々の研究は，プロセッサの省電力モードとアクティブモードを切り替える際に生じるエネルギオーバヘッドを，バッファリングによって削減できるというモチベーションに基づいている．プロセッサのアイドル期間中のスタティック電力は，DPM を採用することで削減することができる．しかし，DPM において，アクティブモードから省電力モードへの移行および省電力モードからアクティブモードへの移行には，回路の負荷容量の充放電等の無視できないエネルギオーバヘッドが生じる．従来の DPM では各インスタンスの処理の前後で，モードの切り替えを逐一行うため，インスタンス数分のオーバヘッドが生じる．ここで，バッファリングを用いることで，各インスタンスの開始時刻を遅らせることができるようになる．インスタンスの実行をあえて遅らせることで，後続する複数のインスタンスと共に，続けて処理を行うことができる．これによって，処理時間は従来の DPM と等しくしたままで，モード遷移の回数を格段に減らすことができる．

図 2 はリアルタイムシステムにおける従来の DPM 制御の様子を示している．横軸が時間経過を表し，縦軸が各インスタンスを表す．横軸に平行な各線分が，その線分に対応するインスタンスの実行可能時間帯を意味する．この線分の終端点までに各インスタンスを終了させる必要がある．隣り合ったインスタンスの線分の開始点間の距離がイ

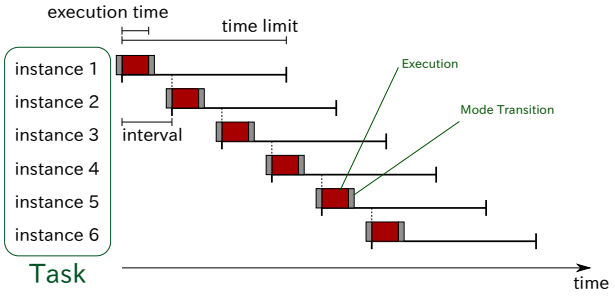

図 2　従来の DPM 制御

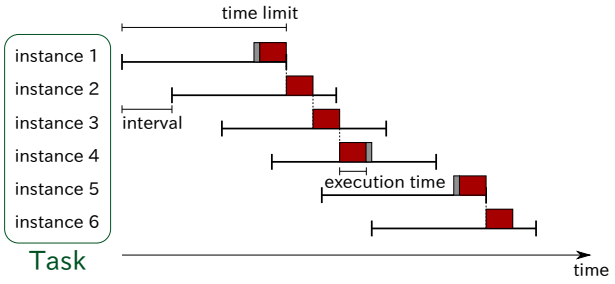

図 3　まとめて処理を行う例

ンスタンスのインターバルに相当する．赤い矩形部分がインスタンスの実行を表す．処理の前後にある灰色の矩形領域が状態の遷移を示している．従来の DPM では，1 インスタンスあたり前後に 1 回の状態遷移（省電力モードへの移行およびアクティブモードへの復帰）が行われる．

図 3 はリアルタイムシステムにおいて複数インスタンスをまとめて実行した様子を示している．バッファリングによって，各インスタンスの実行開始時刻をずらすことができるようになっている．複数インスタンスをまとめるために，インターバルよりも長い，ゆるい時間制約が活用されている．図 3 のように 4 インスタンスをひとまとめにして処理をした場合には，1 インスタンスあたりの状態遷移回数が 1/4 に減少している．これによって，エネルギオーバヘッドも 1/4 に削減される．

本稿の主な貢献を次に挙げる．1 つ目はゆるい時間制約を他のリアルタイムシステムと切り離して扱ったことである．リアルタイムシステムの省電力化に関する多くの研究が，インスタンスの制約時間とインターバル時間を等しいと仮定している [1], [5], [6], [9]．ゆるい時間制約のアプリケーションにも適用できる研究 [4] も少数ながら存在するが，他のリアルタイムシステムと一括して扱っている．ゆるい時間制約のアプリケーションの特性を利用して，最悪ケースのエネルギを削減する手法の提案はこれまで全く行われていない．

2 つ目は単一タスクのインスタンスをまとめて実行するスタティックスケジューリングの検討である．我々の過去の研究 [10] において，**めりはり型実行**が提唱されている．めりはり型実行とは，命令の実行順序を調節することでプロセッサのアイドル時間をまとめる実行方法である．めりはり型実行は本研究と似た概念であるが，[10] ではアウトオブオーダ実行の命令の並べ替えを主に想定している．また，文献 [5] は，マルチタスクのリアルタイムシステムにおいて，タスクをまとめて実行する手法を提案している．しかし，インターバル時間を時間制約と等しいと仮定したタスクモデルを用いており，異なるタスクのインスタンスをまとめて実行している．同一タスクに属するインスタンスをまとめる手法ではない．更に，文献 [5] が提案しているのはダイナミックスケジューリングである．本稿のように，ゆるい時間制約のシステムに特化した，単一タスクのスタティックスケジューリングはこれまでに全く提案されていない．

3 つ目は DVFS のみ行う手法，それに加えて DPM も行う手法，およびバッファリングも行う本手法で消費されるエネルギの定性的・定量的な比較である．各構成についてエネルギモデルを導出し，得失を考察した．また，実アプケーションを想定したパラメタを用いて，既存の 2 つの手法との比較を行い，本手法の有効性を示した．

3. バッファリングによる省電力化

バッファリングを行うことで，ゆるい時間制約のシステムにおける最悪ケースのエネルギを，既存手法よりも更に削減できる可能性がある．本章ではその手法についてエネルギモデルを導出し，検討する．

3.1 バッファモデル

バッファリング回路を挿入することで，各インスタンスの実行開始を遅らせることができるとする．1 インスタンス分のデータをバッファリングするには，読み書き合計で E_{buff}^{dyn} のダイナミックエネルギを消費する．また，最大で N_{buff} 個のインスタンスをバッファリングできる回路を用いた時，常に $P_{buff}^{stat}(N_{buff})$ のスタティックなオーバヘッド電力が生じる．今後，N_{buff} を**バッファ数**と呼ぶ．$P_{buff}^{stat}(N_{buff})$ は N_{buff} に対して増加関数である．実際にまとめて実行するインスタンス数をまとめ度と呼び，N と表記する．この時，明らかに $N \leq N_{buff}$ である．N インスタンスまとめて処理し，その前後でのみ DPM を適用することで，1 インスタンスあたりの状態遷移オーバヘッドは E_{DPM}^{dyn} から E_{DPM}^{dyn}/N となる．4 章の評価では各アプリケーションに対して直接 E_{buff}^{dyn} をパラメタとして与えているが，1 インスタンスあたりの入力データサイズ（単位 [byte]）に比例するとして計算している．$P_{buff}^{stat}(N_{buff})$ は，バッファサイズ （単位 [byte]）に比例する電力の成分と，比例しない成分を想定して算出している．なお，N_{buff} =（バッファサイズ）/（1 インスタンスのデータサイズ）である．

周波数を選ぶことができ，DPM とバッファリングが可能な構成を FPB と表記する．構成 FPB の概念図を図 4 に

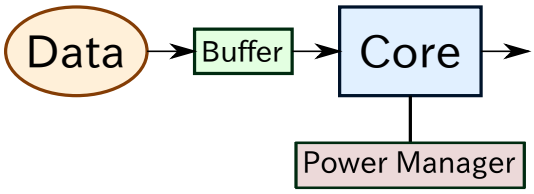

図 4　構成 FPB の概念図

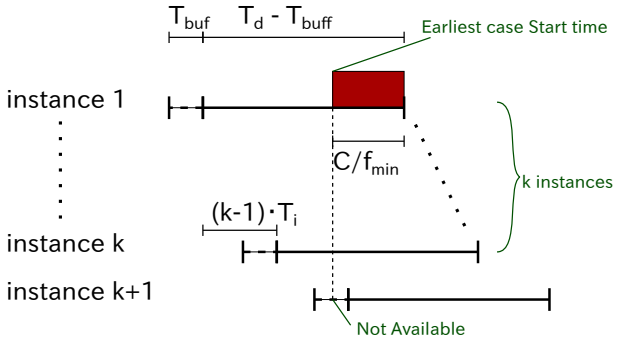

図 5　N_{app} の導出

示した．構成 FPB で消費される 1 インスタンスあたりのエネルギ $EPI_{FPB}(f, N, N_{buff})$ は以下のように表される．

$$\begin{aligned} &EPI_{FPB}(f, N, N_{buff}) \\ &= (P_{dyn}(f) + P_{stat}(f)) \cdot (C/f) \\ &\quad + E^{dyn}_{DPM}/N + P^{stat}_{DPM} \cdot T_i \\ &\quad + E^{dyn}_{buff} + P^{stat}_{buff}(N_{buff}) \cdot T_i \end{aligned} \tag{4}$$

右辺の 1 行目がプロセッサの消費エネルギ，2 行目が DPM に関わるオーバヘッドエネルギ，3 行目がバッファリングによるエネルギである．2 行目の 1 項目はダイナミックエネルギに対応しており，N インスタンスをまとめることで，状態遷移のエネルギが $1/N$ になることを示している．

3.2　最大まとめ度

構成 FPB におけるまとめられるインスタンス数の最大値を導出する．まとめられるインスタンス数の上限を定めるのは，バッファ数と時間制約である．無限のバッファ数を持つときの最大まとめ度を N_{app} とおく．すなわち，時間制約によるまとめ度の制約は $N \leq N_{app}$ である．このとき，最大まとめ度 N_{max} は以下のように表される．

$$N_{max} = \min(N_{buff}, N_{app})$$

バッファ数と時間制約の，制約の厳しい方の値が最大値になることを意味している．

次に N_{app} を導出する．バッファ数が無限である状態で，まとめられるインスタンス数の限界は**図 5** で表される．横方向に伸びる各線分が，各インスタンスの実行可能な時間帯を示している．バッファリングを行うことによって，実際にインスタンスが利用可能になる時刻に T_{buff} 分の遅れが生じる．その利用できない期間を各線分の左端，点線部分で示している．インスタンス 1 の実行開始時刻は，最も開始が早い周波数の場合を考える．本手法とランタイムのスケジューリングを組み合わせる場合，実際の周波数を事前に得られないためである．最も開始が早い時，周波数は 2.2 で定義した f_{min} であり，その実行開始時刻は，インスタンス 1 のデッドラインより C/f_{min} の時間分前の時刻である．インスタンス 1 の実行が開始された時点で，インスタンス 2 からインスタンス k までは，ただちに利用可能であるが，インスタンス $k+1$ は，まだ利用不可能である．さて，本稿で扱っているのは最悪ケースのサイクル数であり，実際にはもっと少ないサイクル数で処理が終わる可能性がある．例えば，インスタンス 1 から k までにおいて，実際に必要とされるサイクル数が非常に小さかった場合，インスタンス $k+1$ のデータが利用不可能である．ゆえに，インスタンス 1 が開始された時点ですでに利用可能なインスタンス数が，N_{app} の最低限保証される値となる．ここで，図 5 中の k は以下の不等式を満たす最大の整数となる．

$$(k-1) \cdot T_i \leq T_d - T_{buff} - C/f_{min}$$

以上の議論によって，ランタイムのリスケジューリングによらず，保証される N_{app} の値は次のように表されることが分かる．

$$N_{app} = k = \left\lfloor \frac{T_d - T_{buff} - C/f_{min}}{T_i} \right\rfloor + 1$$

なお，$\lfloor x \rfloor$ は床関数である．

3.3　最適まとめ度およびバッファ数

バッファ数を選ぶことができない場合，式 4 が N の減少関数であるから，単純に最大のまとめ度 N_{max} を採用すれば良い．そこで，以降ではバッファ数を選べる場合を考える．$P^{stat}_{buff}(N_{buff})$ が N_{buff} に対して増加関数であるため，N_{buff} を小さくする必要がある．しかし，$N \leq N_{buff}$ の仮定と，式 4 が N の減少関数であることから，N_{buff} を小さくし過ぎるとエネルギ消費が増大する．ゆえに $N_{buff} = N$ とし，以下の条件を満たす N, N_{buff} が最適解である．

$$N = N_{buff} = \underset{1 \leq N \leq N_{app}}{\arg\min}\ EPI_{FPB}(f, N, N)$$

なお，$\arg\min_x f(x)$ は，最小の $f(x)$ を与える x を表す．また，バッファの取りうる最大バッファ数が N_{app} よりも小さい場合は，N_{app} の代わりにその値を用いる．

4.　評価

実際のアプリケーションを想定したパラメタを用いて，消費エネルギの比較を行う．本稿では F，FP，FPB の 3 つの構成を比較する．各構成で可能・不可能な機能は**表 3** にまとめた．

アプリケーションとして想定するのは，軽い処理の例と

表 1　評価に用いるパラメタ

	火災報知センサ	イメージセンサ
T_i	100 ms	33 ms
T_d	1 s	1 s
T_{buff}	1.32 μs	10.7 ms
C	1000 cycle	2.8 Mcycle
α	3 mW/MHz	3 mW/MHz
P_{stat}	581 μW	581 μW
E^{dyn}_{DPM}	91 μJ	91 μJ
P^{stat}_{DPM}	0.6 μW	0.6 μW
E^{dyn}_{buff}	4.8 nJ	46 μJ
β	3 μW	3 μW
γ	0.916 nW	8.79μW
$(f^{proc}_{min}, f_{max})$	（4 MHz, 500 MHz）	（4 MHz, 500 MHz）
N_{buff} の範囲	1 - 16000	1 - 3

表 2　パラメタから算出される値

	火災報知センサ	イメージセンサ
C/T_i	10 kHz	85 MHz
(f_{min}, f_{max})	（4 MHz, 500 MHz）	（85 MHz, 500 MHz）
N_{app}	10	29

表 3　本稿で扱う構成

	F	FP	FPB
DVFS	可能	可能	可能
DPM	不可能	可能	可能
バッファリング	不可能	不可能	可能

して火災報知センサ，重い処理の例としてイメージセンサである．それぞれのパラメタおよびそこから算出される値を**表 1**，**表 2** に示した．電圧は周波数によらず一定とし，$P_{dyn}(f) = \alpha f$，$P_{stat}(f) = P_{stat}$（一定）とした．すなわち，ダイナミック電力が周波数に比例し，スタティック電力は周波数によらず一定である．また $P^{stat}_{buff}(N_{buff}) = \beta + \gamma N_{buff}$ とした．バッファリング用の回路で消費されるスタティック電力は，バッファ数に比例する成分（γ）と，バッファ数に比例しない成分（β）に分けられるという仮定である．2 つのアプリケーションにおいて，1 インスタンスあたりのデータサイズがそれぞれ異なるため，バッファ数 N_{buff} の範囲や γ，E^{dyn}_{buff} の値が異なる．

4.1　火災報知センサの場合

EPI_{FPB} について，バッファ数が可変であるときのまとめ度 N および N_{buff} の最適値を導出する．バッファ数が可変である場合には，3.3 節の議論により $N = N_{buff}$ とすればよい．$EPI_{FPB}(f, N, N_{buff})$ における $N(= N_{buff})$ の影響を示したのが**図 6** である．横軸はまとめ度およびバッファ数である．縦軸はそのまとめ度およびバッファ数を採用した際の構成 FPB の 1 入力あたりのエネルギを示している．まとめ度およびバッファ数の最適値は図 6 より，ともに 10 となることが分かる．この値は，時間制約によって定まる最大まとめ度を意味する N_{app} の値である．1 イ

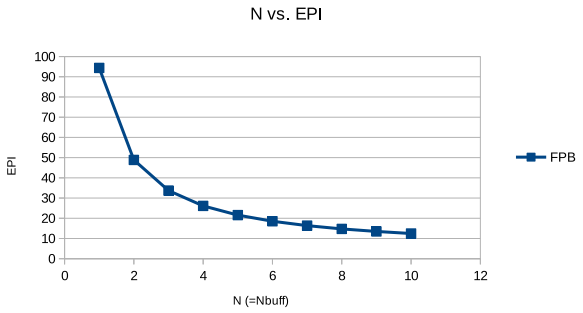

図 6　火災報知センサにおける N，N_{buff} に対する EPI_{FPB} の変化

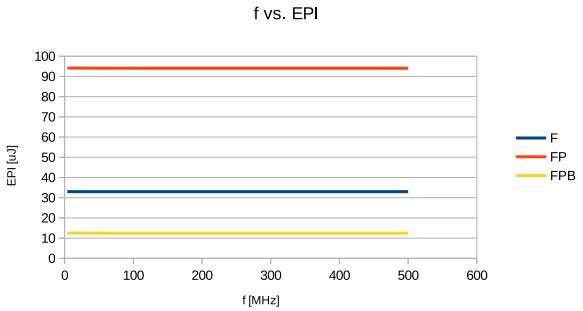

図 7　火災報知センサにおける f に対する EPI_F，EPI_{FP}，EPI_{FPB} の変化

ンスタンスあたりのデータが小さいため，バッファ数が十分に確保できる．ゆえに，バッファ数の上限よりも先に，アプリケーションの時間制約によって最大まとめ度が定まる．構成 FPB において $N = N_{buff} = 10$ としたとき，周波数ごとに構成 F，FP とエネルギを比較した図が**図 7** である．横軸に周波数，縦軸に EPI をとっている．各凡例が各構成に対応する．EPI_F は周波数によらず，周波数の変化に対して一定の値となっている．これは，プロセッサのダイナミック電力が周波数に比例し，スタティック電力が一定であることによる．式 2 の第 1 項は $\alpha f \cdot (C/f) = \alpha C$ より一定値になり，第 2 項も P_{stat} が周波数に依存しないためである．一方で，EPI_{FP}，EPI_{FPB} は周波数に対して減少関数である．しかし，アクティブ時の消費エネルギよりも状態遷移に要するエネルギが非常に大きい．そのため，式 2 の第 1 項の値が第 2 項に対して小さく，周波数の影響がほとんど見えない．DPM の状態遷移に要するエネルギの影響を大きく受ける構成 FP に比べて，状態遷移回数を 1/10 に削減した構成 FPB では効果的に EPI を削減していることが分かる．これらの結果から，状態遷移オーバヘッドを理由に一般の DPM が適用できないようなアプリケーションにも，本提案手法が有効である．バッファリングを用いた手法により，ゆるい時間制約に着目しない既存手法よりも消費エネルギが削減された．既存手法のうち消費エネルギが小さい構成 F の 1 インスタンスあたりの消費エネルギは常に 61.1μJ であるのに対し，構成 FPB は

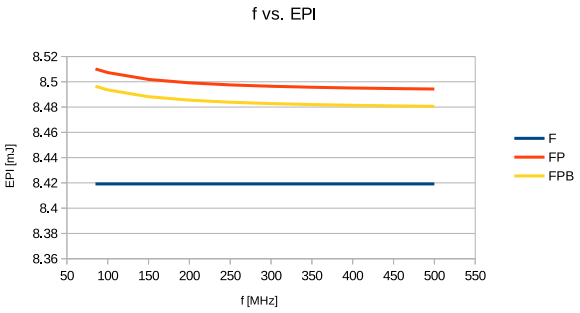

図 8 イメージセンサにおける f に対する EPI_F, EPI_{FP}, EPI_{FPB} の変化

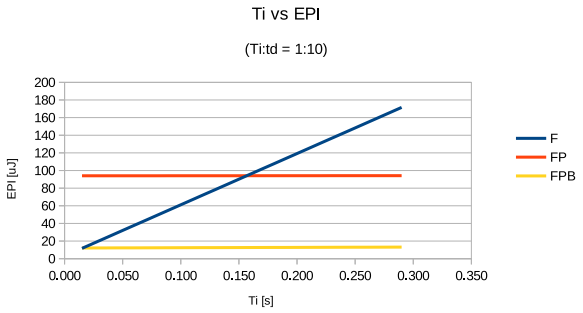

図 9 T_i の変化に対する EPI の変化（火災報知センサ）

500MHz の周波数で 12.5μJ であり，79.6%の消費エネルギが削減された．

4.2 イメージセンサの場合

バッファ数が可変であるときのまとめ度 N および N_{buff} の最適値を導出する．火災報知センサの例と同様に，バッファ数が可変であれば $N = N_{buff}$ とすればよい．$EPI_{FPB}(f, N, N_{buff})$ におけるまとめ度およびバッファ数の最適値は 3 である．時間制約による上限よりも，バッファ数による上限が支配的である．これは，データサイズが大きいアプリケーションを想定しており，少ないバッファ数でも必要とされるバッファの実サイズが膨大になるためである．構成 FPB において $N = N_{buff} = 3$ としたとき，周波数ごとに構成 F，構成 FP とエネルギを比較した図が**図 8** である．横軸に周波数，縦軸に EPI をとっている．各凡例が各構成に対応する．イメージセンサの例では，火災報知センサの例よりも重い処理を想定しているため，1 インスタンスあたりの最悪サイクル数 C が大きい．そのため，式 3 や式 4 において，周波数に反比例する第 1 項の影響が大きくなり，火災報知センサでは見られなかった，周波数に対する変化が見られる．また，重い処理により，アイドル可能な時間が少なくなる．このようなアプリケーションでは，DPM の状態遷移に要するエネルギが，アクティブ状態のままの時に消費されるエネルギよりも大きくなってしまう．更に，まとめ度の上限が 3 であるため，状態遷移のオーバヘッドをあまり削減できない．火災報知センサでは構成 FPB のエネルギ消費が一番小さかったのに対し，イメージセンサでは構成 F の消費エネルギのが一番小さいのはそのためである．以上から，1 インスタンスあたりのデータサイズが大きいアプリケーションや，処理に時間を要するアプリケーションには本手法が向かないことが分かる．

4.3 T_i の影響

T_i の値を変化させることで，構成 F，FP，FPB の各構成のエネルギ消費がどのように変化するか評価する．T_i は

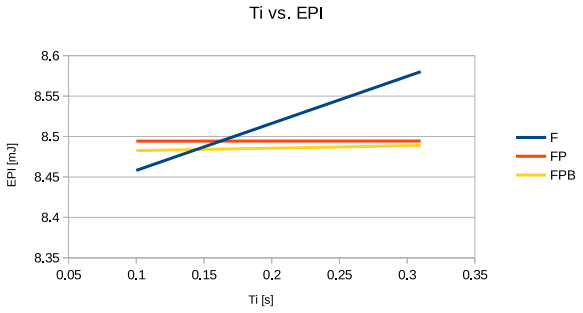

図 10 T_i の変化に対する EPI の変化（イメージセンサ）

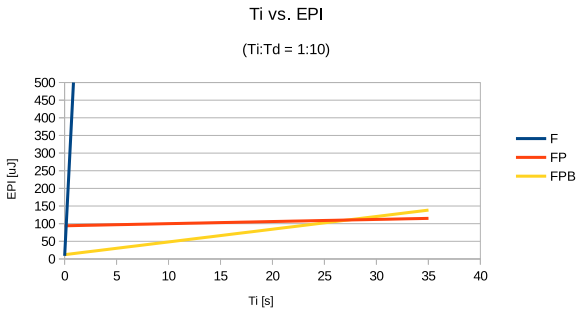

図 11 T_i の変化に対する EPI の変化（火災報知センサ・広域）

アイドル時間に密接に関わっており，T_i が大きいほどアイドル時間が長くなる．アイドル時間が十分に長い時，最もエネルギ消費が少ないのは構成 FP である．構成 F はアイドル期間中にもプロセッサのスタティック電力が消費され続けてしまい，エネルギ消費が大きい．構成 FPB は，構成 FP が持たないバッファリング回路を持つため，そのスタティック電力がアイドル期間中のエネルギに上乗せされてしまう．一方，アイドル時間が短い時には，構成 F が最もエネルギの消費が少ない．構成 FP や構成 FPB の状態遷移時におけるダイナミックエネルギや，バッファリングのためのダイナミックエネルギが，アイドル期間中のプロセッサのスタティックエネルギを上回るためである．ゆえに，提案手法は，既存の構成 F や FP では十分な削減が期待出来なかった，中程度のアイドル時間のアプリケーションに効果があると考えられる．

図 9 が火災報知センサ，**図 10** がイメージセンサに対する結果を表す．横軸が T_i，縦軸が各構成の EPI である．ただし，横軸の T_i の値に応じて T_d/T_i が一定になるように T_d も変化させている．

まず火災報知センサについて考察する．**図 11** は図 9 をより広い T_i についてプロットしたものである．T_i =0.015 〜 27s の広い範囲で構成 FPB の消費エネルギが最小であることが分かる．アプリケーションが扱うデータサイズが比較的小さく，複数インスタンスをまとめて実行するためのコストが小さいためである．特に構成 F と構成 FP が同程度のエネルギを消費する $T_i = 0.16$s 付近では，両構成の 94.1μJ に対して，構成 FPB は 12.7μJ である．既存手法の 86.5%のエネルギ削減が達成されている．

次にイメージセンサについて考察する．元のパラメタの 33ms では構成 F のエネルギ消費が最小であったが，0.13s 〜 0.58s の間で構成 F のエネルギ消費が最小になっている．イメージセンサのような例でも，アイドル時間が長ければ提案手法は有効に作用することがありうる．しかし，火災報知センサの場合に比べて，提案手法が最適になる T_i の範囲はあまり広くない．これは，アプリケーションの扱う 1 インスタンスあたりの入力データサイズが大きいことが原因であると考えられる．データサイズが大きいと，その分大きなバッファが必要になる．そして，1 インスタンスあたりのバッファサイズが大きくなると，バッファが消費するダイナミックエネルギもスタティックエネルギも大きくなる．

以上から，特に扱うデータサイズが小さく，インターバルが中程度のアプリケーションにおいて，提案構成 FPB の採用による大幅なエネルギ削減の可能性が示された．これまで，最悪ケースの消費エネルギに関して行われてきた「インターバルが長ければ DPM，短ければアクティブ状態のまま」という設計手法 [2] に対して，2 手法間のギャップを埋めることのできる強力な手法であることを意味する．

5. まとめと今後の展望

本稿では，**ゆるい時間制約**のリアルタイムシステムを対象に，新しい省電力化手法を提案した．バッファリング回路の挿入により，DPM の状態遷移に伴って生じるエネルギオーバヘッドを削減できることを示した．モデル上で最適なパラメタの選び方を導出し，実アプリケーションを想定したパラメタを用いて既存の構成と比較した．比較によって，本手法が，既存手法では対応出来なかった，中程度のインターバルのアプリケーションに対して有用であることを示した．火災報知センサの例では，既存の手法に対して，79.6%のエネルギ消費を削減した．更に，アイドル時間を変化させることで，最大で 86.5%のエネルギ削減が可能になるアプリケーションの存在も示唆した．

本稿の提案する構成はマルチコアをはじめとする各種システムにも応用可能である．マルチコアシステムに対して，本手法が省電力化にどの程度寄与するのか検討する予定である．また，バッファリング回路に対して，パワーゲーティング，不揮発化等の技術を適用することで，より強力な省電力化が可能である．今後，バッファリング回路のスタティック電力削減による影響についても調査する．

6. 謝辞

本研究の一部は，NEDO「ノーマリオフコンピューティング基盤技術開発」事業による．

参考文献

[1] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Comput.*, Vol. 53, No. 5, pp. 584–600, May 2004.

[2] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system level dynamic power management. *IEEE Transactions On Very Large Scale Integration (VLSI) Systems*, Vol. 8, No. 3, June 2000.

[3] Anantha P. Chandrakasan, Samuel Sheng, and Robert W. Brodersen. Low power cmos digital design. *IEEE Journal of Solid State Circuits*, Vol. 27, pp. 473–484, 1995.

[4] Cheol-Hoon Lee and Kang G. Shin. On-line dynamic voltage scaling for hard real-time systems using the edf algorithm. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium*, RTSS '04, pp. 319–327, Washington, DC, USA, 2004. IEEE Computer Society.

[5] Linwei Niu and Gang Quan. Peripheral-conscious scheduling on energy minimization for weakly hard real-time systems. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pp. 1–6, Apr 2007.

[6] Euiseong Seo, Jinkyu Jeong, Seonyeong Park, and Joonwon Lee. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, pp. 1540–1552, 2008.

[7] Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pp. 188–200, New York, NY, USA, 2004. ACM.

[8] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.

[9] Dakai Zhu, Rami Melhem, and Bruce R. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, pp. 686–700, 2003.

[10] 近藤正章, 中村宏. めりはり型実行モデルに基づくアーキテクチャ. 情報処理学会研究報告, 2005-ARC-165, Vol. 2005, No. 120, pp. 73–74, 2005.