2K-2

# リアルタイムシステム向け細粒度パワーゲーティング制御のためのスケジューリング手法の研究

嶋田　裕巳[†1]　坂本　龍一[†1]　塚本　潤[†1]　和田　基[†1]　佐藤　未来子[†1]　並木　美太郎[†2]

[†1] 東京農工大学大学院工学府　　[†2] 東京農工大学大学院工学研究院

## 1. 緒言

近年，組み込みシステムにおいて LSI の微細化に伴い増加するリーク電力の削減が求められている．また，リアルタイムシステムの必要性が高まっており，性能制約が厳しい中で省電力性が求められる．筆者らは細粒度 PG(Power Gating)技術を搭載したプロセッサ Geyser の研究においてリーク電力の削減に取り組んでいる．Geyser は細粒度 PG で使用していないユニットへの電源供給を遮断し，スリープ状態にさせることでリーク電力を削減するプロセッサである[1]．本研究ではこの細粒度 PG を RTOS のスケジューラで制御することでリアルタイム性を保証しつつリーク電力の削減を目指す．本稿ではリアルタイムタスク動作時に発生する余裕時間を振り分けてユニットのスリープ期間を意図的に延ばす PG 制御手法を提案し，Geyser の自律的な PG よりさらにリーク電力を削減することを目標とする．本稿では RTOS での余裕時間の振り分け手法について述べ，電力評価を行い検証する．

## 2. システムモデル

### 2.1 Geyser の PG 制御

Geyser の PG は四つのユニット(ALU, Shift, Mult, Div)において PG 効果の指標となるサイクル数である BEP (Break Even Point)を持つ．PG 時のスリープ期間が BEP を下回るとオーバヘッド電力により消費電力が増加する．一方で PG 時のスリープがある一定の期間を超えると消費電力が最小の定常状態となる．これらの特徴を踏まえ，Geyser では各ユニットの PG 動作モードをソフトウェアから制御して省電力化を図るために PG 制御レジスタと PG 制御命令を提供している．PG 制御レジスタでは，ハードウェア PG を行わせない「アクティブ」，ハードウェア PG を行わせる「動的 PG」，指定したサイクル数まで各ユニットをスリープさせる「強制スリープ PG」の三つを設定できる．

### 2.2 PG 制御を行う Geyser RTOS

本稿で扱うタスクは Geyser 上で動作する RTOS により周期の短いものから高い優先度を与える RM(Rate Monotonic)アルゴリズムでスケジューリングされる．Geyser RTOS ではタスク処理がデッドラインまでに終了したときに発生する余裕時間を活用し，図 1 に示すように BEP 以上の比較的長いスリープが定常状態以上になるようにスリープを延ばすことでリアルタイム性を保証しつつリーク電力を低減する．

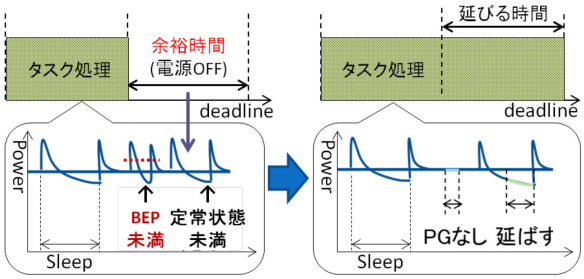


図 1　PG 制御適用の例

BEP 未満のスリープによる電力増加は PG 制御命令を適用して常にアクティブにすることで抑制する．

## 3. 余裕時間振り分け手法

### 3.1 余裕時間振り分けのアルゴリズム

PG 時のスリープを延ばすためには RTOS から PG 制御レジスタを操作し，強制スリープ PG を適用する．そのためにまず RTOS 上でタスクごとに事前実行を行い，各タスクの処理時間や，スリープ頻度情報から得られる動的 PG と強制スリープ PG を適用したときの消費リーク電力を比較した電力削減率を取得する．マルチタスク実行時に本情報に基づき，以下のアルゴリズムにより余裕時間を振り分ける．図 2 にアルゴリズムの各処理とタスクの関係を示す．

(a) 動作中のどのタスクへ余裕時間を振り分けるかを決定する．タスクの選択方式は 3.2 節に示す．図 2 ではタスク 1 を選択している．

(b) 処理(a)で選択されたタスクへ振り分けるための余裕時間を算出する期間を決定する．余裕時間算出期間はタスクの周期とする．どのタスクの周期を選ぶかはタスク選択方式によって異なる．図 2 ではタスク 1 の周期を選択している．

(c) RM スケジューリングにおいてデッドラインを保証するために，タスクに振り分けられる余裕時間の最大値を算出する．デッドラインを保証できるかは CPU 使用率により判断でき，各タスクの周期・処理時間をもとに算出する．この処理はタスク初回起動時に行われる．

(d) 余裕時間算出期間における余裕時間をタスクの周期や処理時間をもとに見積もる．

(e) 強制スリープ PG を適用し，タスクにおける定常状態未満のスリープから順に，定常状態以上になるように処理(d)で求めた余裕時間を処理(c)で求めた最大値を超えないように振り分ける．

処理(d)，(e)は余裕時間算出期間ごとに行われる．アルゴリズムの各処理を RM スケジューラに追加す

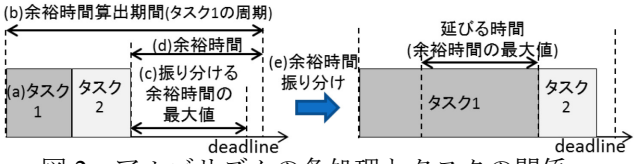


図 2　アルゴリズムの各処理とタスクの関係

ることで，マルチタスク実行時に余裕時間を振り分けてスリープ期間を延ばしリーク電力を削減する．

### 3.2 余裕時間を振り分けるタスクの選択方式

余裕時間を振り分けるタスクの選択方式として以下の三つの方式を提案する．各方式によって 3.1 節で述べたアルゴリズムの処理(a)(b)の内容が異なる．

(1) 電力削減率の高い一つのタスクを選択：処理(a)はタスク初回起動時に行われ，各タスクの事前実行で求めておいた電力削減率を比較し，最も高いものを振り分けるタスク$T_e$として決定する．$T_e$の決定後，処理(b)において$T_e$の周期を余裕時間算出期間として決定する．事前実行時に削減効果のあるタスクを選択することでマルチタスク実行時においても電力削減が見込める．

(2) ある期間ごとに一つのタスクを選択：処理(a)は処理(b)で決められた任意の余裕時間算出期間ごとに行われ，最初に動作するタスクに余裕時間を振り分ける．余裕時間算出期間で最初に動作するタスクはそのときに起動リストの先頭にあるタスクとなる．タスク単体での事前実行時とマルチタスク実行時では余裕時間が異なるため，ある期間ごとにタスクを選択することで事前実行では予測できない電力削減が見込める．

(3) すべてのタスクを選択：ある期間において動作する$N$個のタスクに余裕時間を用いて省電力化を図る手法が文献[2]で提案されている．本稿では余裕時間を$N$で割り，全タスクに振り分ける．処理(a)はタスク初回起動時に行われ，全タスクを対象とする．まとまった余裕時間を振り分けるために処理(b)では余裕時間算出期間として全タスクの周期の中で最も長いものを選択する．方式(1)，(2)に比べて与える余裕時間は少なくなるが，各タスクからの電力削減が見込める．

これらの方式(1)～(3)を適用し，PG 制御を行ったときの電力評価を次章に示す．

## 4. 評価

評価では，周期タスクを動的 PG のみで動作させた場合と，提案した PG 制御方式をそれぞれ適用し動作させた場合において，各ユニットの平均リーク電力の合計を机上で算出した．この電力を動的 PG と選択方式(1)～(3)において比較し，削減効果を示す．また，どの選択方式が効果的であったかを示す．評価タスクとして周期を持つタスクを三つ組み合わせた内容の異なるタスクセット i～iii を動作させたとした．平均リーク電力は Geyser を FPGA 上に実装した Geyser on FPGA のパフォーマンスカウンタ

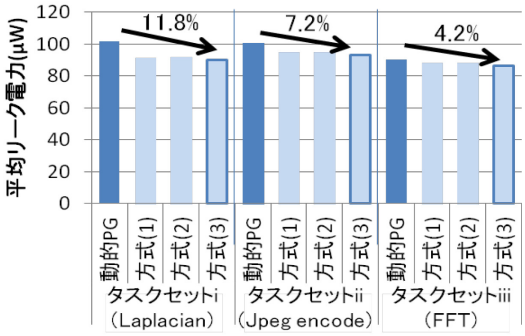


図 3　平均リーク電力

から取得できる各ユニットのサイクル情報をもとに，各方式を適用したときのタスクの動作をシミュレーションして算出した．

図 2 に動的 PG，各方式の平均リーク電力を示し，動的 PG と比較して最も高かった削減率を示す．タスクセット i～iii においてどの方式においても従来の動的 PG よりもリーク電力を削減することができ，削減率は平均で 6.5%，最大で 11.8%となった．また，どのタスクセットにおいても削減率は方式(3)が最も高くなった．方式(1)，(2)ではある期間で一つのタスクを選択するため，実行中の振り分けそのものの回数が方式(3)に比べて少なくなる．このとき方式(1)，(2)は方式(3)と比べて使える余裕時間が長くなるが，本評価の処理内容では余裕時間が多いと定常状態未満のスリープに余裕時間を振り分けきってしまい余ってしまった．そのため方式(3)のように少ない余裕時間でも多くのタスクに振り分けることで削減率が高くなったと考えられる．

## 5. 結言

本稿ではリアルタイムシステム特有の余裕時間を用い，定常状態を超えるようユニットをスリープ状態にさせることで消費リーク電力を削減する PG 制御手法を提案した．評価の結果，すべてのタスクに余裕時間を振り分ける方式で動的 PG に比べて最大で 11.8%のリーク電力を削減することができた．このことから余裕時間を余らせないようタスクに振り分けることで，電力削減の効果が出ることがわかった．今後の課題としては提案した PG 制御手法を RTOS のスケジューラに実装することが挙げられる．現在はシミュレーションで評価を行っているため，実際にタスクを動作させたときの評価を行い，電力削減効果を確認する．

## 参 考 文 献

[1] N. Seki, et al. “A Fine-grain Dynamic Sleep Control Scheme in MIPS R3000”, Proceeding of the 26th IEEE International Conference on Computer Design, pp.612-617, 2008.

[2] Pillai, P. et al. “Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems”, 18th ACM Symposium on Operating Systems Principles, pp.89-102, 2001.