システムＬＳＩ設計技術　97－2
（２０００．９．７）

低消費エネルギー化を目的とする可変電源電圧プロセッサを用いたITRONによる電圧制御アルゴリズム

大隈 孝憲　　門前 淳　　安浦 寛人

九州大学 大学院 システム情報科学府 情報工学専攻
〒816-8580 福岡県春日市春日公園 6-1
E-mail:{okuma, m_atsusi, yasuura}@c.csce.kyushu-u.ac.jp

あらまし:
動的に電源電圧を変化させ，その性能と消費電力を制御できる可変電源電圧プロセッサをリアルタイムシステムに用いる場合，OSによってタスク毎に電源電圧を制御することで低消費エネルギー化が図れる．電源電圧を下げると消費エネルギーを下げることができるが，電圧低下に伴ってプロセッサの動作周波数も低減するため，タスクの実行時間は長くなる．そのため，電源電圧を下げるときはリアルタイム制約を損なわないよう行う必要がある．

本稿では，タスクスケジューリングの際に，次に実行されるタスクを決定するだけでなく，そのタスクを実行するプロセッサの電圧も決定するアルゴリズムを提案する．このとき，すべてのタスクが時間制約を満たすという条件下で，システムの消費エネルギーを最小化する電圧が選ばれる．さらに，提案するアルゴリズムは，ITRON仕様のOSに実装可能なアルゴリズムである．

キーワード：　低消費電力設計，可変電源電圧プロセッサ，実時間処理，タスクスケジューリング，μITRON

Voltage Scheduling Applied to the μITRON Operating System for a Variable Voltage Processor

Takanori OKUMA　　Atsushi MONZEN　　Hiroto YASUURA

Department of Computer Science and Communication Engineering, Kyushu University
6–1 Kasuga-koen, Kasuga, 816–8580, Japan
E-mail: {okuma, m_atsusi, yasuura}@c.csce.kyushu-u.ac.jp

Abstract:
This paper presents a voltage scheduling applied to the μITRON OS for a variable voltage processor which can vary its supply voltage dynamically. On the variable voltage processor, task processing with lower supply voltage is able to reduce power consumption dramatically. But, lower voltage may violate time constraints of the real-time process. In this paper, we propose technique by which CPU time and supply voltage are simultaneously assigned each task to minimize power consumption. It is possible that our technique is implemented in μITRON OS.

Key Words: Low Power Design, Variable Voltage Processor, Real-Time Scheduling, μITRON OS

1 はじめに

組み込みシステムの多くは実時間処理を伴い，リアルタイム性への要求が厳しく高速化が望まれる．またその一方で，軽量のエネルギー源で多様な処理を長時間実行するシステムの実現に対する要求も高まっている．システムにリアルタイム性を持たせるためには，高い速度性能を持つ汎用のプロセッサを利用することが考えられるが，多くの場合，低消費エネルギー化に対する要求が満足されない．なぜなら，一般的にプロセッサのエネルギー消費とプロセッサの動作周波数との間にはトレードオフの関係があるためである．つまり，高速化と低エネルギー化を同時に満たすことは困難である．この問題を解決するために，文献 [4] において，電源電圧を動的に変更できる可変電源電圧プロセッサが提案されている．可変電源電圧プロセッサは以下の性質を持つ．

- プロセッサは電圧制御命令を持ち，アプリケーションやオペレーティングシステムがこの命令を使うことにより，プロセッサの電源電圧とクロック周波数を任意の実行ステップで変更できる．
- プロセッサは電源電圧の変更による回路遅延の変化に合わせたクロック周波数を発生する．

可変電源電圧プロセッサにより，制約時間が厳しく，短い時間で処理しなければならないタスクには，高い電圧で高速に処理を行ない，制約時間が緩く，高い処理能力を必要としないタスクには，低い電圧で処理し，消費エネルギーを抑えることができる．一般に消費エネルギーは電源電圧の 2 乗に比例する [8, 9] ので，低い電圧で処理を行なうことはエネルギー削減に大きく貢献できる．図 1 に例を示す．この例において電圧，周波数，エネルギーの関係を図中の表のように仮定する．この仮定の下で実行サイクル数が 1000M サイクルのタスクを制約時間 25 秒以内に完了しなければならないとき，5.0V で実行した場合と 4.0V で実行した場合とでは，両方とも制約時間を満たしているが，4.0V で実行した方がエネルギー消費が少なくなる．

電源電圧を下げると消費エネルギーを下げることができるが，電圧低下に伴ってプロセッサの動作周波数も低減するため，タスクの実行時間は長くなる．そのため，電源電圧を下げるときはリアルタイム制約を損なわないよう行う必要がある．プロセッサの電源電圧の制御はアプリケーションやオペレーティングシステムによって行なわれる．よって，アプリケーションが

Supply Voltage	4.0V	5.0V
Clock Frequency	40MHz	50MHz
Energy/Cycle	25nJ	40nJ

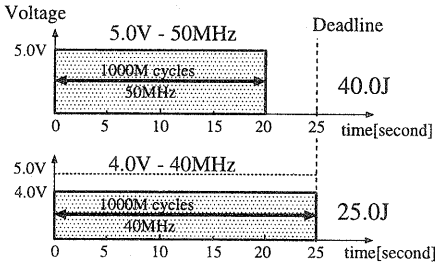

図 1: 電力削減の例

リアルタイム制約を満たすように電源電圧の制御を行なうためのコンパイラ技術や，オペレーティングシステム技術の確立が重要である．

文献 [6] において，筆者らは可変電源電圧プロセッサを用いた場合のオペレーティングシステムにおけるタスクスケジューリングのアルゴリズムを提案した．このアルゴリズムはタスクスケジューリングの際に，次に実行されるタスクを決定するだけでなく，そのタスクを実行するプロセッサの電圧も決定する．このとき，すべてのタスクが時間制約を満たすという条件下で，システムの消費エネルギーを最小化する電圧が選ばれる．電圧を動的に決定するアルゴリズムとして，あらかじめ到着時刻が分かっているタスクセットに対しては SD(Static order and Dynamic voltage allocation) アルゴリズムと，到着時刻があらかじめ分かっていないタスクセットに対しては DD(Dynamic order and Dynamic voltage allocation) アルゴリズムを提案した．

上記アルゴリズムは，電圧を低減させずに実行させた場合にリアルタイム制約が満たされているならば，電圧を低減させてもリアルタイム制約が満たされることを保証している．しかし，まだ実際の OS に対しての実装は行なわれていない．実際の OS として ITRON 仕様の OS を選択し，上記アルゴリズムを実装しようとした場合，アルゴリズムに対する仮定にいくつか問題があり，実装するのは困難であった．そこで本稿では，それらの問題点を明らかにし，ITRON 仕様の OS に実装し易いように改良した電圧制御アルゴリズムを

提案する.

本稿の構成は以下の通りである. 第2章で以前提案した動的な電圧決定アルゴリズムについて簡単に述べる. 第3章では, そのアルゴリズムを ITRON 仕様の OS に実装する上での問題点を挙げ, その解決策を議論する. 第4章では, ITRON 仕様の OS に実装可能な電圧制御アルゴリズムを提案し, 第5章で, その考察を行なう. 最後に第6章でまとめと今後の課題を述べる.

2 動的な電圧決定アルゴリズム

本章では, 文献 [6] で提案した, 動的な電圧決定アルゴリズムについて説明する. このアルゴリズムは, 対象とするシステムに対して SD アルゴリズムと DD アルゴリズムに別れるが, 基本的な考え方は同じである. タスクの実行が, あるタスクから別のタスクへと実行が切り替わる際に, 今から実行されるタスクの電圧を決定し, 電圧をその値に変更する. その際, タスクの総実行時間は最悪のケースを考えて電圧を決定している. しかし実際にそのタスクを実行させた場合, 予想終了時刻よりも早く終了する可能性が高いため, その分 時間的な余裕 (余裕時間) が生じる. 今まで実行されていたタスクも, 最悪ケースの実行時間を考慮にいれて決定された電圧で実行されていたが, 上記の理由で余裕時間が生じるため, この余裕時間を次に実行されるタスクの実行時間として利用することで, そのタスクを実行させる電圧の値の低減を図っている. もちろんこのときも, 今から実行されるタスクの実行時間は最悪ケースであることを考慮にいれて電圧が決定される. そうすれば, また余裕時間が生じ, タスクの実行が切り替わるごとに電圧の低減を図ることが可能となる. さらに, SD アルゴリズムでは, すべてのタスクの到着時刻があらかじめ分かっているため, 今から実行されるタスクよりも後で実行されるタスクの予定終了時刻がそのデッドライン時刻と比較して余裕があれば, その時間も先に利用して電圧の低減を図っている.

2.1 アルゴリズムの構成

電圧決定アルゴリズムでは, 以下の3つのステップで電圧を割り当てるタスクの開始時刻と終了予定時刻を求め, その時間内に終了するような電圧をタスクに割り当てている.

Step1: CPU Time Allocation

すべてのタスクを最大電圧で実行すると仮定して CPU 時間を割り当てる. この時, 各タスクの実行サイクル数は最悪ケースで考える. このときの決定された各タスクの start time, end time を保持しておき, 後のステップで利用する. また, 各タスクに対してデッドライン時刻と end time との差分も保持しておく. これは SD アルゴリズムにのみ利用される.

Step2: End Time Prediction

次に実行されるタスクの終了予定時刻を決定する. DD アルゴリズムの場合, Step1 で決定されたこのタスクの end time が終了予定時刻である. さらに, SD アルゴリズムの場合は, 後に実行されるタスクに対して, Step1 で求めたデッドライン時刻と end time との差分の最小値を終了予定時刻に加算する.

Step3: Start Time Assignment

次に実行されるタスクの開始時刻は, 前に実行されたタスクの終了時刻によって変動する. タスクの実際の実行サイクル数は入力データによってバラツキがあり, 最悪の場合より早く終了することの方が多い. つまり前に実行されたタスクが早く終了すればするほど, 次に実行するタスクの開始時刻を早めることが可能である.

3 ITRONに実装する上での問題点

本章では, 上記アルゴリズムを ITRON 仕様の OS に実装する上で, 問題となる点をあげ, その解決策を述べる.

3.1 待ちの問題

上記アルゴリズムにおいて, タスクの順序を決定するアルゴリズムとして, Earliest Deadline First(EDF)[3, 7] が採用されている. EDF とはタスクをデッドラインの早い順に実行するアルゴリズムである. つまり, デッドラインの早いタスクが優先度の高いタスクとなる. 現在実行されているタスクよりも高い優先度を持つタスクが実行可能になった場合, その実行はより高

い優先度をもつタスクへと切り替わる．このとき，実行を中断されたタスクは，また，いつでも実行が再開できるように，レディーキューに入れられる．つまり，上記アルゴリズムの対象とするタスクの状態は，実行可能状態と実行状態の2状態しかない．

一方，ITRON仕様でのタスクの状態は，実行可能状態と実行状態の2状態に加え，待ち状態という重要な状態が加わる．待ち状態とは，IO待ちや資源の確保待ちなどで，実行が待たされる状態である．待ち状態の時は，優先度が一番高くても実行することができない．

上記アルゴリズムでは，待ち状態の概念がないため，ITRON仕様のOSに実装することは非常に困難である．つまり，待ち状態を実行可能状態と同様に扱って上記アルゴリズムを適用した場合，待たされているタスクのリアルタイム性が保証できないのである．

3.2 デッドラインの問題

上記アルゴリズムにおけるタスクのモデルでは，各タスクにデッドライン時刻が与えられる．このデッドライン時刻が早いか遅いかで，優先度が決定される．一方ITRON仕様では，各タスクにデッドライン時刻を指定することができず，優先度を直接与えなければならない．

優先度に関しては問題はないが，SDアルゴリズムではデッドライン時刻までの時間的な余裕も利用して電圧の低減を図っているため，デッドライン時刻の指定できないITRON仕様のOSでは，その時間の利用が難しい．それと比較して，DDアルゴリズムでは，その時間を利用していないので，デッドライン時刻の指定はまったく意味がない．つまり，ITRON仕様のOSにDDアルゴリズムを実装する場合は，この問題は考える必要はない．しかし，デッドライン時刻までに明らかに余裕がある場合，その時間をうまく利用できる仕組みがあれば，電圧をさらに低減することが可能であろう．

3.3 解決策

3.3.1 待ちの問題に対して

待たされているタスクのリアルタイム性を保証するために，次に実行されるタスクよりも優先度の高いタスクが待ち状態にあるならば，電圧の低減は行なわないようにする．あるタスクが待ち状態に存在しても，次に実行されるタスクよりも優先度が低ければ，アルゴリズムにしたがって，電圧の低減を行なう．これは，電圧を低減せずに実行させた場合にリアルタイム制約が満たされているならば，アルゴリズムにしたがって電圧を低減させてもリアルタイム制約は満たされるというポリシーに違反しない．

3.3.2 デッドラインの問題に対して

SDアルゴリズムでは，CPU Time Allocationで求めたend timeからデッドライン時刻までの空き時間を利用して電圧の低減を図っている．つまり，次に実行されるタスクに電圧を割り当てる際に，そのタスク以降に実行されるタスクに対してその空き時間の最小値を毎回計算して，その最小空き時間を次に実行されるタスクのために利用している．この値は静的に求めることが可能な値で，この値をタスクのパラメータとして与えることも可能である．デッドラインの問題の解決策として，はじめからこのパラメータの値を与えることで問題を解消できる．つまり，このパラメータをmargin timeとすると，SDアルゴリズムにおいては，今まで通り計算していた値を，そのまま使えば良いし，さらにDDアルゴリズムにおいても，margin timeを利用することで，電圧のさらなる低減を図ることが可能である．しかし，DDアルゴリズムの場合，その値の決定を慎重に行なう必要がある．SDと同様にデッドライン時刻までの時間を一杯に使ってその値を決定してしまうと，リアルタイム性が保証できなくなるので，プログラマの責任においてこの値を決定することにする．プログラマがmargin timeを設定する時，デッドライン時刻までの空き時間を一杯に使う必要はなく，タスクが実行される時の負荷に応じて，自由に設定することが可能である．しかし，自由に設定できるとは言っても，ある規則に従う必要がある．この規則に従わないとリアルタイム性を保証できない．その規則とは以下の規則である．

margin timeに関する規則

$Priority(T_i) \geq Priority(T_j)$ ならば，
$T_i.mtime \leq T_j.mtime$ である．

ここで $Priority(T)$ はタスク T の優先度を表し，$T.mtime$ はタスク T のmargin timeを表す．

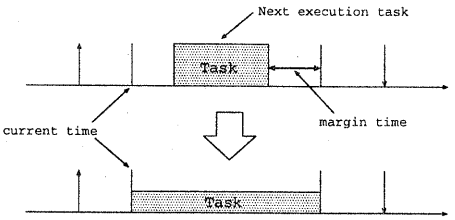

図 2: margin time

わる．OS ではリアルタイムな時間を扱う必要があるため，時間の経過を表す別の変数が必要になる．この変数をリアルクロックと呼ぶことにする．クロック周波数が $\frac{1}{m}$ になると，システムクロックがカウントアップされる時間間隔は m 倍になる．よって，インターバルタイマハンドラによってカウントアップされる値が，システムクロックが 1 であるのに対してリアルクロックは m カウントアップされるようにすれば，実際の時間経過を反映することができる．

4 ITRONによる電圧制御アルゴリズム

4.1 プロセッサに対する仮定

本稿で提案する ITRON による電圧制御アルゴリズムにおけるプロセッサに対する仮定を以下に述べる．

- 以下の m 種類のクロック周波数を切替えて動作させることができる．

$$f, \frac{f}{2}, \frac{f}{3}, \cdots, \frac{f}{m} \ [\mathrm{Hz}]$$

- 電圧はクロック周波数の変更に合わせて自動的に変更される．
- 周波数を切替える命令が用意されており，プログラムによってプロセッサのクロック周波数を変えることが可能である．

提案するアルゴリズムは電圧を制御するアルゴリズムであるが，実際はプロセッサを動作させるクロック周波数 $\frac{f}{m}$ の m の値を動的に決定するアルゴリズムである．

4.2 時間の扱い

OS の扱うことのできる時間は，インターバルタイマハンドラによってカウントアップされる変数によって管理される．この変数をシステムクロックと呼び，カウントアップされる間隔が OS の扱うことのできる時間の最小単位である．インターバルタイマハンドラはプロセッサで N 命令実行される毎に割り込みが入り実行される．よってプロセッサの周波数が変わるとシステムクロックがカウントアップされる時間間隔も変

T_{pre}: 今まで実行されていたタスク
T_{next}: 次に実行されるタスク
T_{idle}: アイドルタスク
$finish?(T)$: タスク T が終了していたら真
$T.xmax$: タスク T の最悪実行時間
$T.stime$: タスク T の実行開始時刻
$T.mtime$: タスク T の margin time
$real\_clock$: リアルクロック
$Priority(T)$: タスク T の優先度
$MPT(\mathbf{J})$: 集合 $\mathbf{J}$ の中で，最も優先度の高いタスク．
$\mathbf{W}$: 待ち状態にあるタスクの集合
$\mathbf{R}$: 実行可能状態にあるタスクの集合

Algorithm:

if $T_{pre} = T_{idle}$ then
  $static\_stime = real\_clock$
else if $finish?(T_{pre})$ then
  $static\_stime \mathrel{+}= T_{pre}.xmax$
else
  $$static\_stime \mathrel{+}= \frac{real\_clock - T_{pre}.stime}{m}$$
  $$T_{pre}.xmax \mathrel{-}= \frac{real\_clock - T_{pre}.stime}{m}$$
end if

$T_{next} = MPT(\mathbf{R})$

if $Priority(T_{next}) \geq Priority(MPT(\mathbf{W}))$ then
  $etime = static\_stime + T_{next}.xmax + T_{next}.mtime$
  $$m = \left\lfloor \frac{etime - real\_clock}{T_{next}.xmax} \right\rfloor$$
else
  $m = 1$
end if

$T_{next}.stime = real\_clock$

図 3: ITRON に実装可能な電圧制御アルゴリズム

4.3 電圧制御アルゴリズム

スケジューラはタスクスイッチ時, 次に実行するタスクを決定し, そのタスクを実行させるプロセッサの動作周波数 $\frac{f}{m}$ の m の値を決定する. そして, プロセッサの動作周波数を変更する命令を発行し, 次に実行するタスクへと実行を移す. この時の m を決定するアルゴリズムを図 3 に示す. 各タスクの最悪実行時間 ($T_i.xmax$) と margin time ($T_i.mtime$) は, あらかじめ静的に求めておく. このときそれぞれの値は, OS の扱うことのできる時間の最小単位であるシステムクロックを基準にして求める. つまり, それぞれの時間を求めたら, その時間内での $m = 1$ におけるシステムクロックのカウント数を設定する.

static_stime は静的開始時刻を表す. 静的開始時刻とは, 今まで実行してきたタスク全てが電圧を低減させずに実行され, 最悪ケースで終了したと仮定した時の次に実行されるタスクが開始し得る時刻である. この時刻に最悪実行時間と margin time をプラスした値がそのタスクの終了予定時刻になる. 最初の if 文では *static_stime* の更新を行なっている. *static_stime* は, 前に実行されていたタスク (T_{pre}) の静的開始時刻になっているので, タスクスイッチの時に値を更新する. タスクスイッチが起こるの以下の 3 つのケースが考えられる.

1. 何も実行されていない時 (アイドルタスクが実行されている時) に, 新しいタスクが実行可能状態になった.
2. 実行中のタスクの処理が終了した.
3. 実行中のタスクの優先度よりも高い優先度を持つタスクが実行可能状態になった.

次に実行されるタスク (T_{next}) の静的開始時刻は, それぞれ以下のようになる. まず (1) の場合は, その時の時刻 (*real_clock*) になる. (2) の場合は, T_{pre} の静的開始時刻に, T_{pre} の最悪実行時間を加えた時刻になる. また, (3) の場合は, T_{pre} の静的開始時刻に, T_{pre} の実行が開始されてから中断されるまでの処理を $m = 1$ の状態で実行させた時の経過時間を加えた時刻になる. (3) の場合, 中断されたタスクの実行が再開された時に備えて, 最悪実行時間から, 中断されるまでの $m = 1$ における実行時間を引いておく.

レディーキューの中から, 一番優先度の高いタスクを選び (T_{next}), そのタスクの優先度と, 待ち状態にあるタスクのなかで一番優先度の高いタスクの優先度を比較する. もし, T_{next} の優先度が大きければ, 現時刻から実行を開始して, 終了予定時刻 (*etime*) までに終了するような最大の m を与える. またそうでない時は $m = 1$ として, 電圧の低減を行なわない. 最後に次のタスクスイッチに備えて T_{next} の実行開始時刻を現時刻に更新しておく.

5 考察

動的に m の値を決定するために, オーバヘッドを考えなければならないが, 提案するアルゴリズムは数サイクルの処理であるため, スケジューリングのオーバーヘッドはほとんどない.

OS の扱うことのできる時間の最小単位はシステムクロックであることは述べた. よって, クロック周波数の切替えをシステムクロックの時間単位で行なうようにアルゴリズムを設計したが, タスクスイッチはシステムクロックを基準にして起こるわではなく, 任意の時間で起こる. それゆえ, 微妙な時間のずれが生じるかも知れない. このことがリアルタイム性にどのように影響するかは, 今後の課題とする.

6 おわりに

本稿では, 可変電源電圧プロセッサの電源電圧を OS により制御するためのアルゴリズムを提案した. 提案するアルゴリズムは, ITRON 仕様の OS に実装可能なアルゴリズムである. このアルゴリズムにより電源電圧をタスク毎に制御することによって, リアルタイム制約を満たしつつ, エネルギー消費を大幅に削減することが可能である. 今後は, アルゴリズムをより最適なものへと改良できるように, 可変電源電圧のシミュレーションもしくはエミュレーションができ, エネルギー消費の評価がとれる環境を構築したい.

謝辞

本研究は, 一部STARCプロジェクト番号:PJ-No.987「コアプロセッサベースシステム LSI の最適化設計技術に関する研究」の支援による.

参考文献

[1] Giorgio C. Buttazzo. *"HARD REAL-TIME COMPUTING SYSTEM"*. Kluwer Academic Publishers, 1985.

[2] T. Ishihara and H. Yasuura. "Voltage Scheduling Problem for Dynamically Variable Voltage Processors". In *Proc. of International Symposium on Low Power Electronics and Design(ISPLED'98)*, pages 197–202, August 1998.

[3] C. L. Liu and J. Layland. "Scheduling algorithms for multiprogramming in a hard real-time environment". *Journal of the ACM*, 20(1):46–61, 1973.

[4] T. Ishihara and H. Yasuura. "Optimization of Supply Voltage Assignment for Power Reduction on Processor-Based Systems". In *Proc. of SASIMI '97*, pages 51–58, 1997.

[5] T. Ishihara and H. Yasuura. "Programmable Power Management Architecture for Power Reduction". *IEICE Transaction on Electronics*, E81-C(9), September 1998.

[6] T. Okuma, T. Ishihara, and H. Yasuura. "Real-Time Task Scheduling for a Variable Voltage Processor". In *Proc. of ISSS99*, pages 25–29, 1999.

[7] W. Horn. "Some simple scheduling algorithms". *Naval Research Logistics Quarterly*, 21, 1974.

[8] 管野 卓雄. *ULSI* 設計技術. 電子情報通信学会, 1993.

[9] 榎本 忠儀. *CMOS* 集積回路. 培風館, 1996.