修 士 論 文

タスクスケジューリング情報を利用するDVSアルゴリズムに関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

今井 聡

2006年3月

修 士 論 文

タスクスケジューリング情報を利用するDVSアルゴリズムに関する研究

指導教官　田中清史 助教授

審査委員主査　田中清史 助教授
審査委員　日比野靖 教授
審査委員　井口寧 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

410012 今井 聡

提出年月: 2006 年 2 月

概 要

DVS はプロセッサは駆動電圧と周波数を動作中に変動させる技術である. 一般にシステムがプロセッサに要求する処理能力には変動幅がある. そこで、時々刻々のプロセッサに求められる処理能力に見合った周波数で動作させることで, 必要な時間当たり処理能力が小さい状況では電力消費を抑制する一方, 高速な処理を必要とする状況への対応を両立することが可能となる.

周波数の変更は、実行状況をモニタした上で一定のアルゴリズムに従って駆動周波数を決定する. 実行状況の情報として,OS の持つタスクスケジューリング情報を利用することで, 現時点のみならず今後実行するプログラムも考慮した上で周波数を決定することが可能となる.

本稿では,DVS プロセッサの制御に, タスクスケジューリング情報を利用することで, デッドラインミスの防止と, 電力消費の削減の両立を企図するシステムを提案する.

目 次

第1章　はじめに

1.1　背景と目的

近年,PDA, 携帯電話, ポータブル AV 機器などに代表される, 持ち歩きながら使用可能な機器, 可搬機器の高機能化が著しく進展している. 可搬機器は, 個人による占有的な使用を目的として製造され, 家庭内やオフィスでの使用にとどまらず, 電源の確保できない外部での使用も可能である. 特に, 携帯電話や PDA では, 個人ユーザーの嗜好にあわせた機能の充実が, 機器の販売戦略上重要な要素となっており, たとえば, 携帯電話端末では, 音声による通話や電子メールによる通信といった基本機能と同時に,Web ブラウザによるインターネット接続, 音楽の再生録音, デジタルカメラによる静止画や動画録画・再生, テレビ放送の受信, 日時に連動するスケジューラなどの付加機能が搭載されている.

これらの実現のために, 可搬機器の開発では, 汎用のパーソナルコンピュータ (PC) やワークステーション (WS) で一般的な要素技術が可搬機器向けに利用される機会が増大している. 具体的には, 動画や音楽に関する機能は, 計算量が大きいために, その実現のために高性能なプロセッサが使用され, これら多数の機能を一つの機器で実現するために, それぞれの機能を実現するアプリケーションを統合的に管理するオペレーティングシステムを利用することが一般化している.

一方, 高機能化に伴ってプロセッサの消費電力増大が重大な問題となっている. 可搬機器では, 軽量小型であることと, バッテリー駆動時間の長いことが特に重要視される. しかし, 計算量の大きい処理を行うための高性能プロセッサは消費電力が増大傾向にあり, 単に処理能力の高いプロセッサを搭載しただけでは, バッテリ駆動時間への悪影響あるいは, バッテリの大型化による本体サイズの大型化が避けられない状況にある.

本研究では, プロセッサの電力消費削減を目的とした,OS によるタスクスケジューラ機能と連携する DVS アルゴリズム, およびハードウェアを提案する.

1.2　本論文の構成

第 2 章 リアルタイムシステムについて説明する.
第 3 章 DVS(Dynamic Voltage Scaling) について説明する.
第 4 章 提案した DVFS コントローラ統合型ハードウェアスケジューラについて説明する.
第 5 章 提案機構の評価を示す.
第 6 章 本論分の関連研究を紹介する.

第 7 章 本論文のまとめおよび今後の課題について.

第2章　リアルタイムシステム

タスク[1] の実行に際して, あらかじめ決められた一定の時刻までに実行完了しなければならないシステムをリアルタイムシステムと呼ぶ. この, タスクを完了しなければならない時刻のことをデッドラインと呼び, デッドラインまでに完了できない事態のことを, デッドラインミスと呼ぶ. 一方, リアルタイムシステムでは多くの場合, デッドラインまでにタスク実行が完了すれば, それよりも早く実行が完了することに価値は無い. この点では, 単により高速であれば価値が高いとするシステムとは異なる. 組込みシステムでは, 多くの場合でリアルタイム性が求められる. 本章では, リアルタイムシステムのタスクとそのスケジューリングについて紹介する.

2.1　リアルタイム性

リアルタイム性は次の 2 種類がある.

1. ハードリアルタイム (Hard Real Time) ： ハードリアルタイム (HRT) では, デッドラインまでにタスクが完了しない場合, タスク実行が完全に無価値となる.HRT の例として, ガソリンエンジンの燃料噴射を制御するタスクがあげられる. ガソリンエンジンでは, 燃料噴射はピストンが特定の位置にある瞬間に行う必要があり, それ以外のピストン位置で噴射を行ってはならない. したがって, 燃料噴射のタスクは, 噴射タイミング以前に必要な計算を完了することが求められる.(図 2.1(a)).
2. ソフトリアルタイム (Soft Real Time) ： ソフトリアルタイム (SRT) では, タスクがデッドラインまでに処理を完了できない場合であっても, デッドラインを超えて実行を完了することに価値がある. 言い換えると, 過負荷状態ではデッドラインミスもやむをえないとするシステムである. また, デッドラインミスを起こした場合も, できる限りデッドラインに近いうちにタスク実行を完了することが望ましい. このような処理の例として, 音声や動画の再生や電話交換機などがあげられる.(図 2.1(b)).

[1]本稿のタスクとは, プログラム内の並行実行単位のことである.

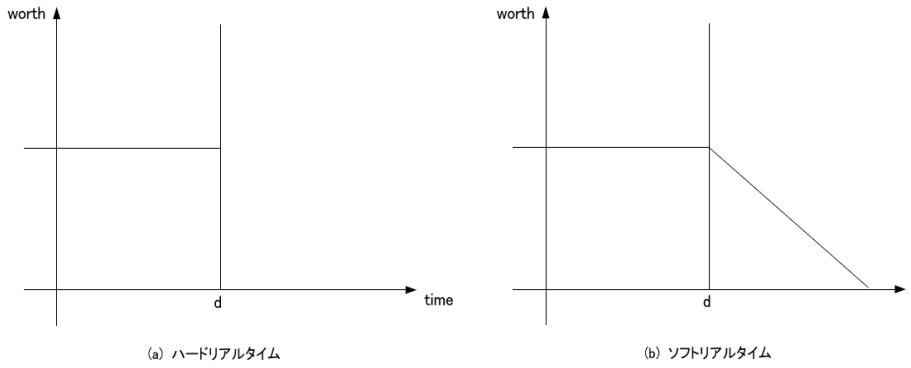


図 2.1: (a) ハードリアルタイム (HRT),(b) ソフトリアルタイム (SRT)

2.2 タスクの状態と属性

本節では, 本論文で使用する用語について ITRON4.0[2] の仕様に基づいて定義を行う.

2.2.1 タスクの状態

タスクの状態は次の 5 つに分類される. このうち広義の待ち状態は, さらに 3 つの状態に分類される. また, 実行状態と実行可能状態を総称して, 実行できる状態と呼ぶ.

- 実行状態 (RUNNING)

  対象のタスクが現在実行中である状態. ただし, 非タスクコンテキスト[3]の実行中は, 直前に実行されていたタスクを実行状態とする. 実行できる状態 (=実行状態+実行可能状態) のタスク中で最高優先度のもの.

- 実行可能状態 (READY)

  対象タスクを実行するための準備が整っているが, より優先度の高いタスクを実行中のため, 実行を待っている状態. 実行できる状態 (=実行状態+実行可能状態) のタスク中で最高優先度以外のもの.

- 広義の待ち状態

  そのタスクを実行できる条件が整わないために, 実行できない状態. 何かの条件が揃うのを待っている状態. タスクが広義の待ち状態にある間, プログラムカウンタやレジスタなどのプログラムの実行状態 (タスクコンテキスト) は保存されている. タスクを広義の待ち状態から実行再開する時には, タスクコンテキストを広義の待ち状

[2] http://www.assoc.tron.org/spec/itron/mitron-402j.pdf
[3] タスク以外の処理が利用する環境のこと. 割込みハンドラが利用する環境など

態に入る直前の値に復元する. 広義の待ち状態は, 待ちの原因によって, 次の 3 つに分類される.

- 待ち状態 (WAITING)
  何かの条件が整うまで自タスクの実行を中断するシステムコールを呼び出したことにより, 実行が中断された状態. 整うべき条件として以下のようなものがある.
  - 起床待ち (slp_tsk,tslp_tsk) [4]
  - 時間待ち (dly_tsk)
  - イベントフラグ成立待ち (wai_flg,twai_flg)
  - セマフォ獲得待ち (wai_sem,twai_sem)
  - メールボックスでのメッセージ受信待ち (rcv_mbx,trcv_mbx)
  - 固定長メモリブロック獲得待ち (get_mpf,tget_mpf)
- 強制待ち状態 (SUSPENDED)
  他のタスクによって, 強制的に実行を中断させられた状態. ただし, μ ITRON4.0 仕様では自タスクを強制待ち状態にすることもできる.
- 二重待ち状態 (WAITING-SUSPENDED)
  待ち状態と強制待ち状態が重なった状態. 待ち状態にあるタスクに対して強制待ち状態への移行が要求されると, 二重待ち状態に移行させる.

- 休止状態 (DORMANT)
  タスクがまだ起動されていないか, 実行を終了した後の状態. タスクが休止状態にある間は, コンテキストは保存されない. タスクを休止状態から起動するときには, タスクの起動番地から実行を開始する.

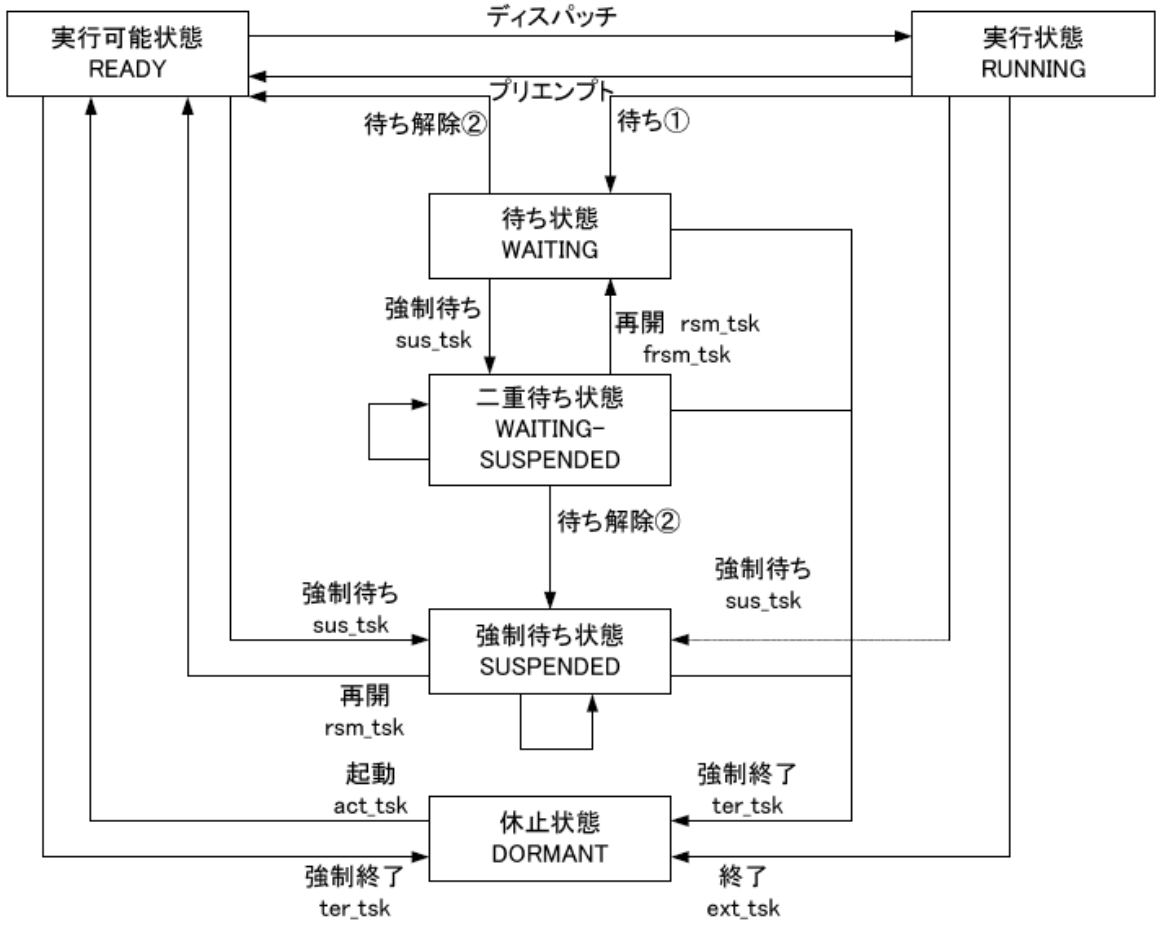
- 未登録状態 (NON-EXISTENT) タスクがまだ生成されていないか, 削除された後の状態. システムに登録されていない仮想的な状態.

実行可能状態のタスクや, 新たに実行可能状態となったタスクの優先度が, 実行状態のタスクより高くなったときには, ディスパッチが発生し, 実行タスクが切り替わる. このとき, それまで実行状態だったタスクは実行可能状態に移行し, このことをプリエンプトされたという. (図 2.2)

2.2.2 タスクの属性

タスクは様々な時間に関する属性を持つ.(図 2.3) に示す. タスクは起動要求時刻 (activation time) に発生した外部事象を受け, 実行可能状態となる. そして, オペレーティングシ

[4]括弧内は対応するシステムコール名



① slp_tsk, tsl_tsk, wai_sem, twai_sem, wai_flg, twai_flg, rcv_mbx, trcv_mbx, get_mpf, tget_mpf, dly_tsk
② rel_wai, wup_tsk, sig_sem, set_flg, snd_mbx, rel_mpf, ter_tsk

図 2.2: タスクの状態遷移図

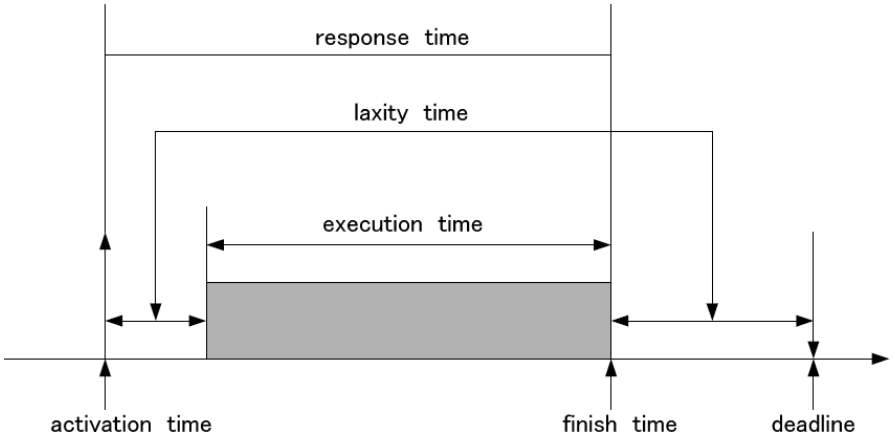


図 2.3: タスクの属性

ステムのスケジューリンラによって順序付けされ, 実行権を得た時に実行される. タスクの処理が終了した場合には休止状態となり, 次回の起動要求がくるまで実行されない.

タスクの終了時刻 (finish time) から起動時刻を引いたものを応答時間 (response time) という. 終了時刻がタスク起動ごとに決められている締切り時刻 (deadline) を守るようにタスクスケジューリング, 実行されなければならない. デッドラインを超えることなく実行を完了できる最大遅延時間を余裕時間 (laxity time) という.

一般に, タスクは起動の規則性により, 周期タスクと非周期タスクの 2 種類に分類できる. 周期タスクは一定間隔で起動要求が発生するタスクであり, 非周期タスクは不定期な起動要求が発生するタスクである.

2.3 リアルタイムスケジューリング

2.3.1 優先度

優先度は, タスク毎に決定される値であって, タスクは優先度が高いものから順番に実行される. 優先度を決定する作業がタスクの実行順序を決定することになる. 優先度決定の作業をタスクスケジューリングと呼ぶ. タスクスケジューリングによって決定された優先度に基づいて, 実行するタスクを準備する機能がスケジューラであり,OS の機能の一つである.

2.3.2 WCET

リアルタイムシステムに限らず, プログラムの実行時間は入力データなどの条件の違いにより変動する. しかし, タスクの実行時間がわからなければ, デッドラインを保証するタスクスケジューリングが不可能である. そこで, 最長の実行時間を見積もることが行われる. この時間を最悪実行時間 (Worst Case Execute Time) と呼ぶ.

2.3.3 リアルタイムタスクスケジューリングアルゴリズム

タスクの実行順序を決定するアルゴリズムをスケジューリングアルゴリズムという. スケジューリングアルゴリズムの分類法はいくつかあるが, 本節では静的スケジューリングと動的スケジューリングの分類を紹介する.

- 静的 (static) スケジューリング

  プログラマがあらかじめ設定した優先度を使ってタスク起動の順序付けを行う. 静的優先度, あるいは固定優先度 (fixed priority) とも呼ばれ, 一度決められたら優先度が変化しない.

- 動的 (dynamic) スケジューリング

  タスクの時間属性などによって, 優先度が変化するタスクスケジューリングアルゴリズムである.

静的スケジューリングはプログラマによって決定されるため議論の余地は少ない. 一方, 動的スケジューリングは様々な提案がされている. ここで, 代表的なタスクスケジューリングアルゴリズムを紹介する.

- 静的優先度方式

  タスク間の優先度順位を基準にしてタスクスケジューリングが行われる. 優先度はあらかじめ段階的に決められていて, 実行可能なタスクが複数ある場合は, 実行可能タスクの中で, 最も優先度が高いタスクが実行される.

  (ITRON では優先度が等しいタスクの場合には FIFO(First In First Out) 形式で順序付けを行う)

- RM(Rate Monotonic)[4]

  周期タスク用のアルゴリズムであり, 周期と相対デッドラインが等しいという条件の下, 周期の短いタスクの優先度を高く設定する. なお,RM 法では他の条件として, タスクが互いに独立していて順序制約が無いこと, 資源制約が無いこと, プリエンプティブであることが必要である.

RM 法を用いてスケジュールできない周期タスクセットは, 他の静的スケジューリングアルゴリズムを用いてもスケジュールできないことが証明されている. また, 周期はあらかじめ設定されているものであることから,RM 法は静的優先度方式の一つである.

- EDF(Earlest Deadline First)[4]

  絶対デッドラインの早いタスクに高い優先度を与える方式. 周期タスク, 非周期タスクともに扱える. 前提条件として, タスクが互いに独立で順序制約が無いこと, 資源制約が無いこと, プリエンプティブであることが挙げられる.

  スケジュール可能なタスクセットであれば, 必ず EDF でスケジュール可能であり, 最大遅延時間を最小にすることが証明されている. ただし, 各タスクの実行時間を考慮に入れないため, 将棋倒し的にデッドラインミスを誘発する可能性がある.

- LLF(Least Laxity First)[5]

  余裕時間の短いタスクから優先度付けを行う方法. 前提条件は EDF と同様である.

  LLF を厳密に実現しようとした場合, 実行状態タスクの余裕時間は減らないが, 実行可能状態のタスクの余裕時間が減少する. この結果, 実行可能状態タスクと実行状態タスクの入替えが発生するが, その直後には再び, 余裕時間の逆転が発生して, タスクの入替えが発生する. そのため, コンテキスト切替が頻発して, オーバヘッドが大きくなるという欠点がある.

- 適応型動的優先度方式 [2]

  タスクスケジューリングのパラメータの中で, 静的優先度方式の場合の優先度は設計者が決定した値であり, 意味的に重要であると考えられる. これに, 実行の最中の状況を一部導入することによって, 優先度を決定するスケジューリングアルゴリズムが, 適応型動的優先度方式 (ADP Adaptive Dynamic Priority) である. 静的優先度を基準として, 実行中に, タスクの余裕時間と周期タスクの場合の周期を取り入れた上で, 優先度を決定している. LLF に従い, 余裕時間に応じて優先度を高く変更する. 動的優先度 $Pri_{dynamic}$ は次式で表される.

$$Pri_{dynamic} = Pri_{static} - (\frac{\gamma}{T} + \frac{\delta}{L})$$

$Pri_{dynamic}$: 動的優先度

Pri_{static} : 静的優先度

T : 周期

L : 余裕時間

γ, δ : 定数

最高優先度の値は 1 であり, 値が大きいほど優先度は低い

ADP では余裕時間の見積もりに最悪実行時間 WCET ではなく,PET(Predictive Execution time) を用いる. PET とは, タスクの実際の実行時間に基づき動的に変動する見積もり時間である. 各タスクはシステム起動後に複数回実行される可能性があるため, 実行毎にタスクの実行時間に基づいて PET の再計算を行い, 次回の実行時間見積もりを見直す. この計算はタスク終了時に行い, 終了したタスクの実行時間と前回の PET の値を加重平均して算出する. これにより, 経験的に実際の実行時間を反映する PET の値を得る. あるタスクの i 回目の実行終了時の PET の再計算は以下の式で与えられる. ここで,β は定数である.

$$PET_i = \beta \times PET_{i-1} + (1 - \beta) \times (measurad\ exe.time)$$

第3章　DVS

本章では, プロセッサ動作中に駆動電圧・クロック周波数を変更する技術である DVS [1]動的電圧変更[2]と,DVS の制御アルゴリズムについて紹介する. なお, 本研究で提案する DVS アルゴリズムは, タスクスケジューリング情報を利用する方法である.

3.1　プロセッサと消費電力

3.1.1　電力消費の成分

プロセッサの消費電力には次の式に示される関係が知られている.

$$P = N_{active} f C V^2 + N_{total} I V$$

P：消費電力
N_{active}：動作中のノード数
f：クロック周波数
C：容量 (定数)
V：駆動電圧
N_{total}：全ノード数
I：1 ノードあたりリーク電流 (定数)

- 第 1 項が消費電力の動的要素

  クロック周波数の関数になっていることからもわかるように, プロセッサの動作による電力消費成分である. 第 1 項では, 消費電力が電圧の 2 乗に比例し, クロック周波数に正比例する.

- 第 2 項が消費電力の動的要素

  第 2 項は消費電力の静的要素で, 電源が ON である限り発生する電力消費である. 第 2 項では, クロック周波数は関係せず, 駆動電圧に正比例する. 現在の半導体製造プロセスでは, 消費電力に占める第 1 項と第 2 項の割合は 1：1 程度と言われるが, 半導体製造プロセスの微細化と共に, 第 2 項の割合が増大する傾向にある.

[1]Dynamic Voltage Scaling
[2]DVFS (Dynamic Voltage Frequency Scaling) とも呼ばれる

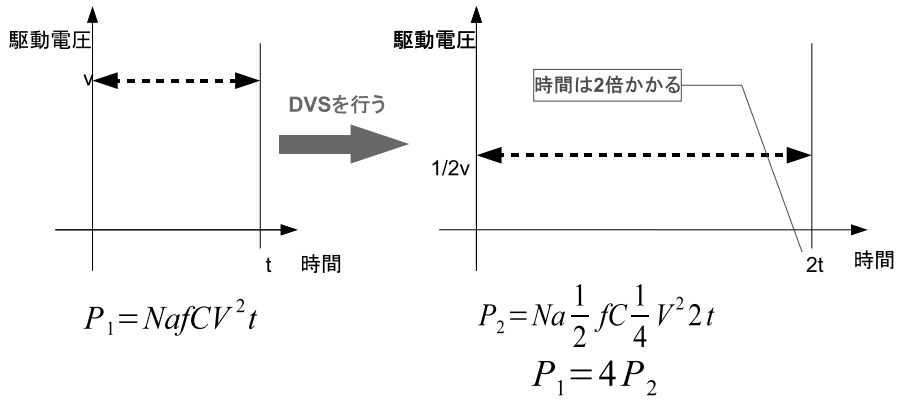


図 3.1: 同一計算を周波数を変えた場合の電力比較

3.1.2 駆動周波数と電力消費

プロセッサの駆動周波数を決定する遅延を支配する要素がトランジスタ間の電界であることから, プロセッサの駆動周波数と駆動電圧は概ね比例関係[3]になる. ただし, 動作可能な電圧には最低と最大とが規定され, 最低電圧以下では動作不能であり, 最大電圧以上ではプロセッサが破壊される. このため, たとえクロック周波数をゼロとしても最低の電圧を供給する必要がある.

一方, 前項に述べたように消費電力の第 1 項 (動的成分) は駆動電圧の 2 乗に比例し, 同時にクロック周波数に比例する. 例えば, 駆動周波数と駆動電圧を 2 分の 1 とした場合, 計算に掛かる時間が 2 倍になるものの, 計算完了までの消費電力量は第 1 項に関しては 4 分の 1 に減少する. 同一のプロセッサを異なる電圧・周波数で動作させて, 同一量の計算を完了するまでの消費電力を図式化したものが (図 3.1) である.

さらに, 消費電力の第 2 項 (静的成分) についても, 高速で動作中の消費電力が電圧に比例して大きい上, 計算完了後にクロックを停止し状態でも, 最低電圧分の電力を消費してしまう.

3.2 DVS 対応プロセッサ

3.2.1 動的な周波数変更

前項に述べた関係から, 一定量の計算を完了するまでの消費電力はプロセッサが許容する最低電圧とそのときのクロック周波数で動作させた場合がもっとも少ない. 最低のクロック周波数で動作させるため計算完了までの処理時間はもっとも長くなる.

多くのシステムで, プロセッサに求められる時間あたり処理能力は一定ではなく, そのと

[3]シュムープロットと呼ばれる電圧と周波数のグラフがプロセッサ製造業者によって公表されている

きのアプリケーションプログラムの実行状況によって異なる. 例えば本研究で対象の 1 つである近年の携帯電話において, 通信の待ち受け状態でほとんどのアプリケーションが休止状態にある状況と, 動画アプリケーションを含むテレビ電話の実行中では, 要求されるプロセッサの処理能力に大差がある.

そこで, プロセッサを必要に応じて低速と高速に切り替え可能とすることで, このような変動する負荷に対応する手法として,DVS(Dynamic Voltage Freqency Scaling) が考案された.DVS では, プロセッサが動作中にクロック周波数と駆動電圧を変更することができる. このため, 前述の例にあげた携帯電話の場合, 要求される処理能力が小さい通信待ち受け状態では駆動電圧を低下させてバッテリーライフの増大を図る一方, 動画アプリケーションや高速通信のアプリケーションが動作するテレビ電話の実行中には, クロック周波数と駆動電圧を上昇させ, 必要な処理能力を得ることができる.DVS 対応プロセッサとしては,Intel 社の PentiumM, 同 XScale,Transmeta 社の Crusoe,Efficeon, ルネサステクノロジ社の SH-Mobile など, バッテリー駆動機器への搭載を考慮する多くの製品が製造, 販売されている.

3.2.2 周波数遷移時の挙動

DVS において, 周波数と駆動電圧を変化させるために遷移時間が必要になる. 特に, クロック周波数を上昇させたい場合には, まず供給する電圧を上昇, プロセッサ全体に必要な電圧が供給されてから, クロック周波数を変更する必要がある. 電圧の上昇幅が大きいほど, クロック周波数の変更が可能になるまでのタイムラグが長くなる. そこで,DVS 対応のプロセッサでは図 3.2 に示すように, 段階的にクロック周波数と供給電圧を上昇させている. クロック周波数の下降時には, このような段階的な遷移は不要で, クロック周波数を下げてから電圧を下降させればよい.

3.2.3 DVS クロック周波数の段階と遷移時間

DVS 対応プロセッサでは, クロック周波数と駆動電圧の組み合わせを複数定義している. この組み合わせはプロセッサベンダーによって決定されるものであるが, その値は段階的になっている. 最高速と低速の 2 段階のみの製品もあるが, 多くは 4 段階程度から,20 段階程度である.

3.3 DVS アルゴリズム

本節では,DVS の制御方法について述べる.DVS を備えたプロセッサに対して, クロック周波数を切り替える指示を行うポリシーが必要となるが, このポリシーを DVS アルゴリズムと呼ぶ. その上で, 現在利用されている DVS アルゴリズムとして,AC 電源判別法,ACPI

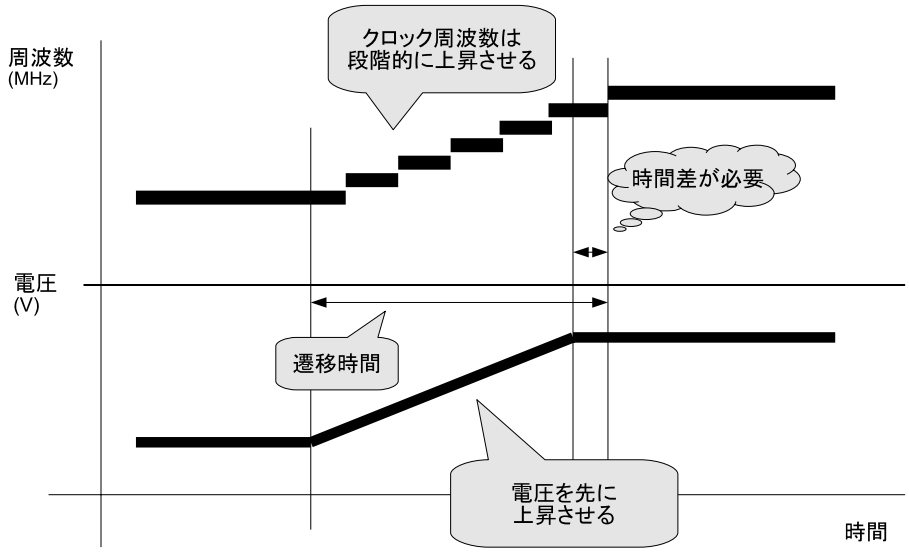


図 3.2: DVS のクロック上昇時遷移

電源ステートデューティー比による制御法および, 本研究で対象とする DVS アルゴリズムとしてタスクスケジューリング情報利用法を紹介する.

3.3.1 AC 電源判別による制御

本方式は,AC(商用交流) 電源が確保できる場合には, 最高のクロック周波数でプロセッサを駆動し, バッテリーからの電源供給時のみクロック周波数を低下させる手法である. 実際の使用例としてはノート型 PC に適用し, 電源が確保できる環境ではデスクトップ型 PC と同等の処理能力を発揮する一方, バッテリー駆動の場合には処理能力の低下を受忍することで, バッテリー駆動時間の増大を図る.

DVS の制御方法としてもっとも単純な方式であり,Intel 社の Pentium3(Mobile) [4]で採用された. 現在では後述の ACPI 電源ステートデューティー比による制御に移行しているが, 電源が確保できる場合には最高のクロック周波数で駆動する手法は引き続き使用されている.

3.3.2 ACPI 電源ステートデューティー比による制御

PC の電源規格である ACPI [5] では, プロセッサ動作中のクロック信号供給について定義している．DVS に対応しないプロセッサを対象する場合，本方式では，オペレーティ

[4]SpeedStep という名称で 2000 年に発表された
[5]Advanced Configuration and Power Interface

ングシステムは, 全タスク中の System Idle Task [6]が占める実行時間の割合を計測することで, 現在のプロセッサ使用率を求め，この使用率からクロック供給を ON にする時間の割合を決定し，プロセッサに通知している．クロック供給の ON/OFF は 1 秒間に 100 回程度実施する．DVS 対応の PC 用プロセッサ (Transmeta 社のノート型 PC 用プロセッサ Crusoe 等) で，オペレーティングシステムが通知したクロック供給の ON 比率を，クロック周波数と電圧の制御に利用する方式が採用されている．

3.3.3 タスクスケジューリング情報利用法

これまでに述べた 2 つの DVS アルゴリズムは, 現時点におけるシステムの状況を情報源とする手法である. 非リアルタイムシステムでは, 電力消費とのトレードオフとして, タスク処理時間が増大する. しかし, リアルタイム性をもつシステムに DVS を適用した場合, クロック周波数低下によるデッドラインミスが発生するかどうかが問題であり,DVS アルゴリズムの工夫によって, デッドラインミスを発生させずに処理が完了できる可能性がある. この場合, クロック周波数の低下による消費電力の削減を実現しつつ, システムの性能には影響を与えなかったと言える.

そこで, 本研究ではデッドラインミスの防止を念頭に,DVS によるプロセッサのクロック周波数を低下させるアルゴリズムとして, タスクスケジューリング情報を利用する手法を考案した.

前章で述べたように, オペレーティングシステムが持つタスクスケジューリング情報には, 各タスクのデッドライン,WCET,PET, 優先度が含まれる. スケジューリングされたタスクの実行順序と PET, それにデッドラインの関係から, 各タスクの余裕時間が算出できる.

ここではタスクスケジューリング情報を利用した DVS アルゴリズムとして Critcal-Deadline 法と保険 DVS について紹介する.

- Critical デッドライン

  リアルタイムシステムでは, デッドラインまでにタスク実行を完了できればよい, 逆に早く完了することは消費電力の観点からは無駄であると言える. そこで, 予測される余裕時間をクロック周波数の低下により増大する処理時間に充当する.(図 3.3) ただし, この図 (図 3.3) のような単一のタスクを DVS の対象とすると, 後続のタスクの余裕時間が少ない場合に, 後続タスクがデッドラインミスを起こしてしまう.(図 3.4) したがって, 後続タスクの余裕時間についても考慮した DVS アルゴリズムとする必要がある.

  そこで, 実行状態 (RUNNING) と実行可能状態 (READY) のタスクのうち先頭の一定数のタスクを対象に, スケジューリング後の余裕時間が最も短いデッドラインを検索する. このデッドラインを”Critical デッドライン”と呼ぶ. そして, スケジューリングされた最高優先度のタスクから,Critical デッドラインまでの区間全体に対して,

[6]アイドルタスク, 優先度が最低のタスクで, プロセッサに他の仕事が無い状態の時実行される

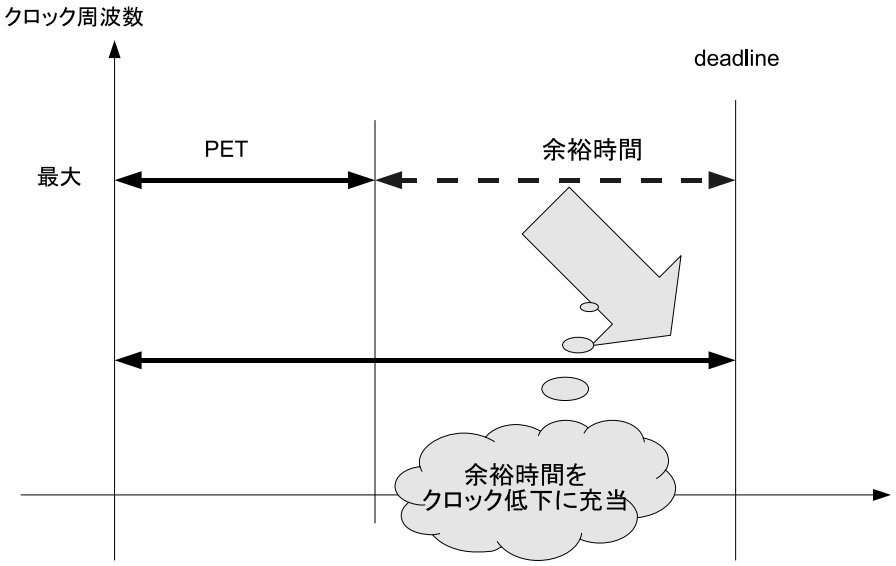


図 3.3: 余裕時間をクロック周波数の低下分に充当

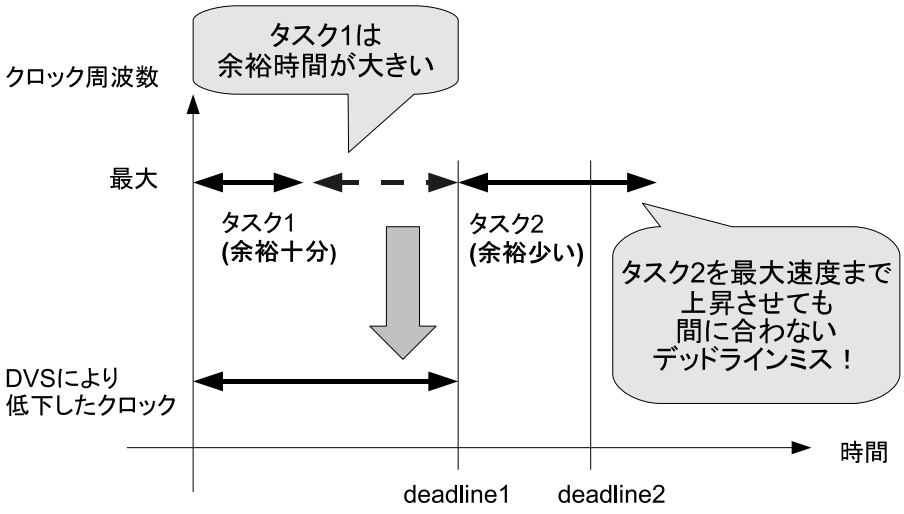


図 3.4: 後続タスクのデッドラインミス

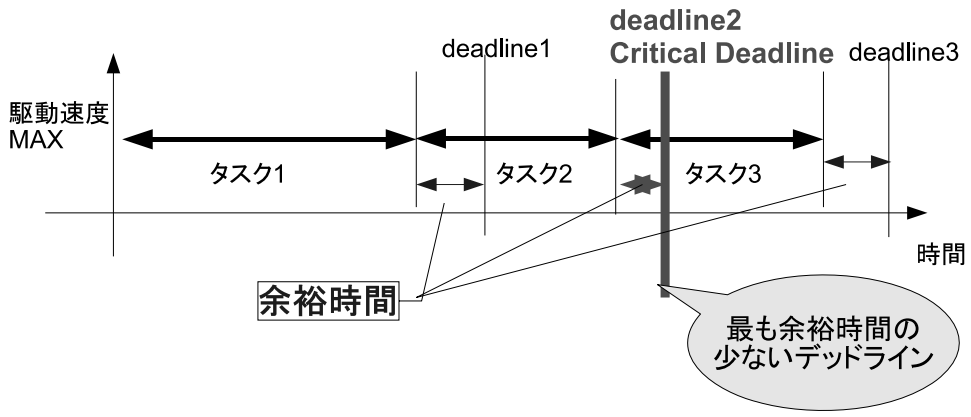


図 3.5: Critical デッドライン

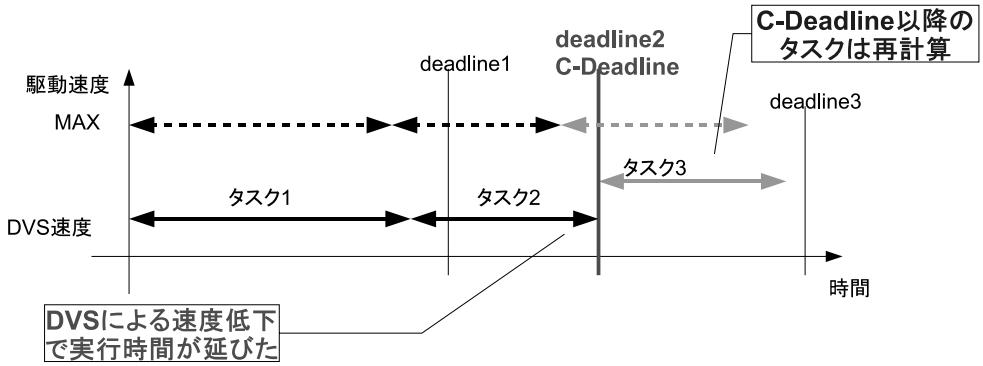


図 3.6: Critical デッドラインによる DVS

余裕時間がゼロになるように DVS レシオを算出する.(図 3.5)Critical デッドラインより後方のタスクについては,Crtical デッドラインを再度算出し,DVS レシオもここで切り替わる.(図 3.6)

実行可能状態になるタスク数はシステムの状況によって変化するため, その中の Critical デッドラインの検索に掛かる計算量は限定できない. 先頭から一定のタスクのみを対象とする理由は,Critical デッドラインの検索に掛かる計算量を限定するためである.

- 保険 DVS

  Critical デッドラインによる DVS アルゴリズムでは, 実行可能タスクの先頭から一定数のタスクを対象として, 余裕時間の予測を行っているため,DVS の検討対象外となっている後方のタスクでデッドラインミスを誘発してしまう可能性がある. そこで, 実行状態, 実行可能状態の全タスクを対象として,DVS によるクロック周波数低下の可否を決定するアルゴリズムとして,”保険 DVS”を考案した.

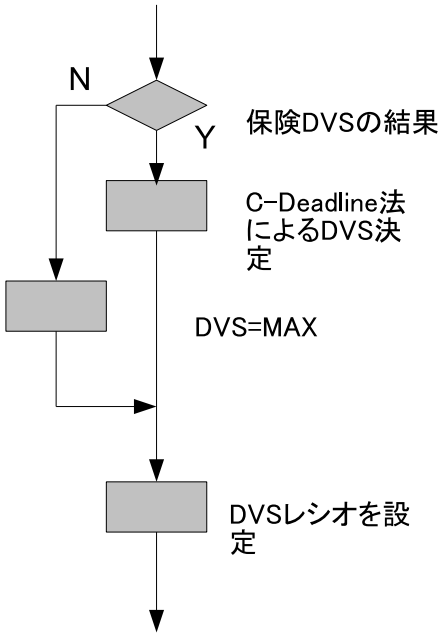


図 3.7: DVS 組合せのフローチャート

保険 DVS では, 実行状態, 実行可能状態タスクの PET を総計した値と, 最遠デッドラインを比較する. ここで, 最遠デッドラインとは, 対象全タスク中の最も遠いデッドラインのことである. 保険 DVS は簡単かつ線形な計算量の DVS アルゴリズムである. タスクが新たに実行可能状態となったときに必要な計算は次の 3 つに限られる.

- それまでの PET 総計に新着タスクの PET を加算して PET 総計を更新
- 最遠デッドラインと新着タスクのデッドライン比較, 最遠デッドラインの入替判断
- 最遠デッドラインと PET 総計の比較

このアルゴリズムで最後に比較した, 最遠デッドラインと PET 総計の差が一定以下となったときに,DVS によるクロック周波数の低下を不可と判断する.

- 2 種の DVS アルゴリズムの組合わせ方

  Critical デッドラインと保険 DVS の組合わせを,(図 3.7) 示す. 保険 DVS によるクロック周波数低下可否判断で, クロック周波数低下が可能と判断された場合に,Critical デッドラインを求め, 周波数の制御を指示する.

第4章　DVSコントローラ統合型ハードウェアスケジューラ

本章では,DVS の制御にタスクスケジューリング情報を利用するために,DVS コントローラを統合したハードウェアスケジューラについて紹介する.

4.1　ハードウェアスケジューラ

タスクの順序を決定するタスクスケジューラは OS の代表的な機能の一つであり, 通常はソフトウェアとして実装される. しかし, デッドラインを考慮して複雑なスケジューリングアルゴリズムを採用した場合, スケジューリングオーバヘッドによって, 却ってデッドラインミスを誘発する可能性がある. 例えば,2 章で述べた適応型動的優先度法は, 比較的計算量の大きいスケジューリングアルゴリズムである. そこで, ハードウェアによるタスクスケジューリングの支援が考えられている [3]. 本研究では, ハードウェアスケジューラの特性を DVS の制御に利用することで, デッドラインミスを防止しつつ, プロセッサ消費電力の削減を行う.

4.1.1　DVS クロック低下時スケジューラ補償

DVS によりプロセッサ駆動周波数が低下している場合, 通常のソフトウェア実装によるタスクスケジューラでは, スケジューリングオーバヘッドが大きくる. ハードウェアによりタスクスケジューラを実装し, プロセッサのクロック周波数と関係なく, 常時一定の周波数で駆動することで,DVS による低クロック状況でもスケジューリングオーバヘッドの増大を抑えることが可能である.

4.1.2　DVS の電圧遷移時間対応

タスクスケジューラを通常のソフトウェアにより実装し, タスクスケジューリングの瞬間のみプロセッサのクロック周波数を最高周波数に上昇させる手法では，DVS 対応プロセッサはクロック周波数上昇時の遷移時間が大きいことによりスケジューリングオーバヘッドを削減することができない.

4.1.3 スケジューリングアルゴリズムの選択

本研究ではスケジューリングアルゴリズムに,ADP[2] を採用した. DVS の制御に利用するタスクスケジューリング情報は, 予測実行時間であり, そこから算出される余裕時間でもある. このため余裕時間を使うアルゴリズムであれば, タスクスケジューリングのために必要な情報を流用するだけで DVS レシオを決定可能となり,DVS のために専用の情報を収集する必要が無い.

ADP は余裕時間を利用するだけなく, 予測実行時間に PET を利用している. WCET に比べて柔軟なため, 過去の実績で余裕時間が大きい場合には,DVS の消費電力削減の効果が期待できる.

4.1.4 スケジューリングオーバヘッドの削減

ADP は比較的計算量のが大きいため, ハードウェアによって実装することで, スケジューリングオーバヘッドを隠蔽できる.[3]

4.2 本研究で提案するハードウェア・ソフトフェア

本節では, 本研究で提案するハードウェアとソフトウェアについて紹介する

4.2.1 通信レジスタによる通信

プロセッサとハードウェアスケジューラの間では, タスク情報を通信する必要がある. タスクスケジューリングに必要なタスクの情報は, ハードウェアスケジューラが保持しており, プロセッサはコンテキスト切替に必要な情報を, 別途保持している. 両者が通信するタスクの情報はタスク ID と呼ばれるタスク識別のための ID である.

タスク ID の通信は, ハードウェアスケジューラとプロセッサの間に設けた, 通信レジスタへの書込み, 読出しで行う. (図 4.1)

新たに起動要求があったタスクや, 中断して待ち状態に入っていたタスクが実行可能状態になった場合, タスクスケジューリングの対象として, また新たに追加する. ここで，起動・再開要求が現在実行中のタスク中で発生した場合, (図 4.2) 非タスクコンテキスト (割込み) から発生した場合,(図 4.3) それぞれにレジスタを用意して処理している．非タスクコンテキストからの起動要求では，現在実行中のプロセッサは関与せず，直接ハードウェアスケジューラが処理を行う.

実行可能状態のタスクが待ち状態になり, スケジュール対象から取り除く処理を依頼する場合も同様である. (図 4.4) これらは, プロセッサからハードウェアスケジューラに, 対象となるタスク ID を渡す.

一方, ハードウェアスケジューラからプロセッサに対してタスク ID を渡すのは, 実行状

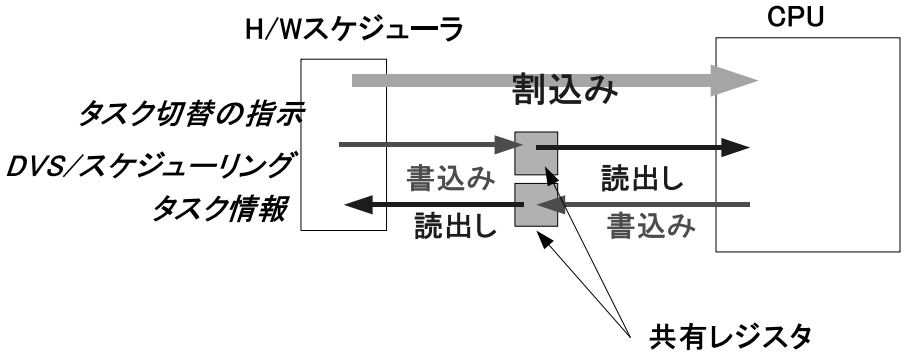


図 4.1: 通信レジスタ

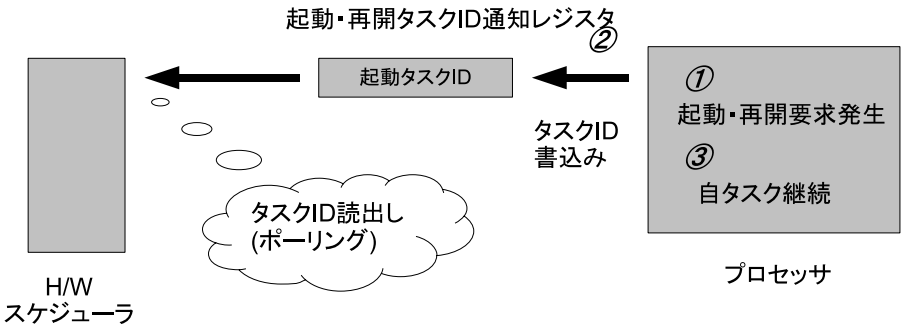


図 4.2: 起動・再開タスク ID の通知 (タスクから)

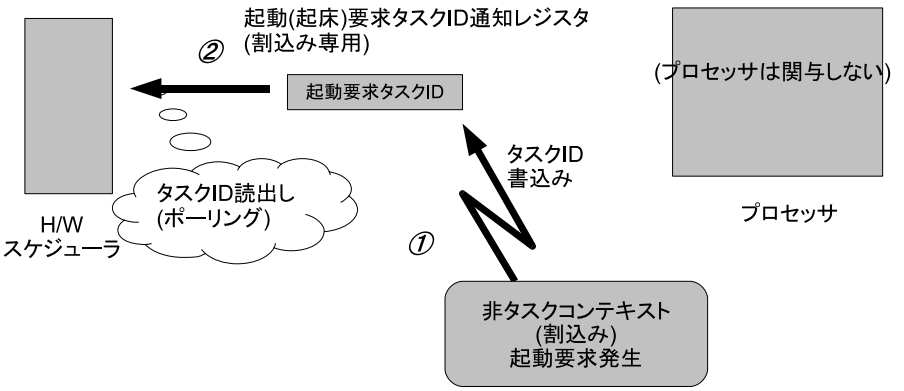


図 4.3: 起動・再開タスク ID の通知 (割込みから)

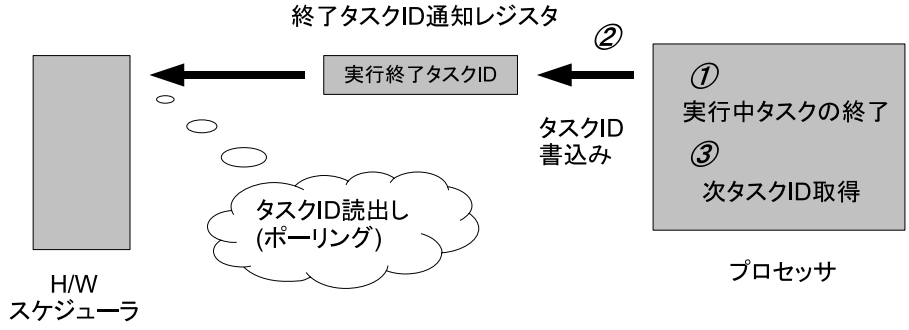


図 4.4: 終了タスクの通知

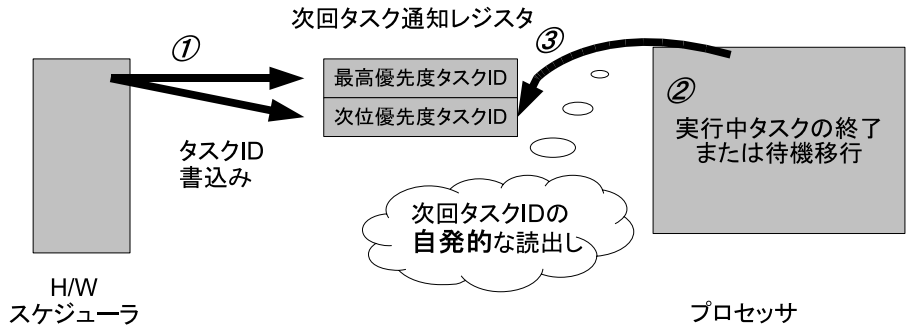


図 4.5: 次回タスク ID の通知 (ディスパッチなし)

態タスクのタスク ID である. 現在実行状態にあるタスクを継続して実行する場合[1]には, 現在のタスクの終了後にプロセッサが自発的に次回実行タスクのタスク ID を取得する. このため，ハードウェアスケジューラは，次回実行タスク ID を用意した上，次回実行タスク通知レジスタに書込む．(図 4.5) 一方，現在実行中のタスクに代わって，新たに実行するタスクを変更する必要がある場合[2]には，ハードウェアスケジューラは次回実行タスク ID を通信レジスタに書込んだ上で，割込みを使用してプロセッサに通知する．(図 4.6) よりも優先度の高いタスクが発生し, 実行状態タスクが切り替わる場合には, 切り替えの通知は, 割込みによって行う.

4.2.2 タスクスケジューリング情報の DVS 制御への流用

ADP によるタスクスケジューリングでは, タスク毎の実行時間の実績を PET として保持している. また，タスクスケジューリングでの必要からデッドラインも保持している. また，タスクの起動，再開，中断の時刻についてもハードウェアスケジューラが，通信レジ

[1]ディスパッチが発生しない場合
[2]ディスパッチが発生する場合

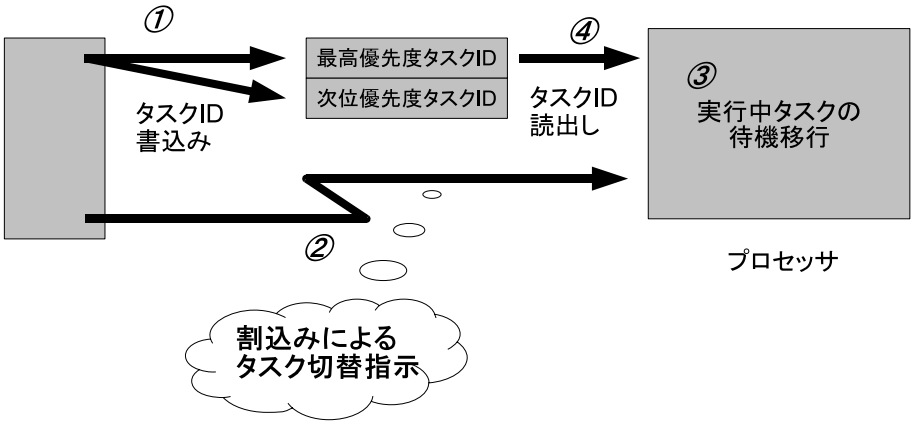


図 4.6: 次回タスク ID の通知 (ディスパッチ)

スタからの読出し時刻を用いることで利用可能である.

これらの情報は，Critcal-Deadline 法と保険 DVS 双方の DVS 制御に必要な情報のすべてをまかなうことが可能である.

4.2.3 ハードウェアスケジューラ

ハードウェアスケジューラは命令実行型となっている. このハードウェアに, ファームウェアとしてタスクスケジューラ機能部分と,DVS 制御部分を搭載し, 無限ループで実行させる. 前節で述べた，各通信レジスタの監視を行い，それぞれのレジスタで得たタスク ID に基づいてタスクスケジューリングを行う．その際，ADP によるタスクスケジューリングのために，各タスクの PET とデッドラインが必要となる．この情報と，タスクスケジューリングの結果を流用して DVS の制御を行う.

第5章　評価

本章ではシミュレータによる実験の結果について評価を示す.

5.1　評価環境

評価に用いた環境は，ハードウェアスケジューラのシミュレータを含むプロセッサのシミュレータ上で，μ ITRON4.0 インターフェース互換の OS を動作させ，仮想タスクセット用いてクロック周波数ごとの動作時間を計測した.

消費電力のモデルには Intel 社の PDA 向けプロセッサ PXA270 の Electrical Specifications[6] を参照した．PXA270 は 13MHz 毎にクロック周波数を変更可能である．本研究ではクロック周波数を 20 段階として，シミュレーションの DVS モード毎に実行時間を計測し，消費電力を計測したものである．次の 3 条件により実験を行った.

- タスクスケジューリングは ADP DVS 実施

  タスクスケジューリングに ADP を用い，スケジューリング情報を利用した DVS 制御を行う.

- タスクスケジューリングは静的優先度 DVS 実施

  タスクスケジューリングは，あらかじめ決定した優先度によって行う．DVS 制御のために，スケジューリングとは別にデッドライン，予測実行時間を用いる.

- タスクスケジューリングは静的優先度 DVS 非実施 (比較用)

  消費電力，実行時間の比較のため，DVS を実施しない条件も実験を行った.

5.2　DVS 適用時の消費電力

プロセッサの消費電は駆動周波数ごとに mW 単位で公表されている．消費電力を算出するために，シミュレータ上でプロセッサが動作した時間をクロック周波数ごとに保存し算出した．本研究で参照した駆動周波数ごとの消費電力の表 (5.1) を示す.

実験結果から消費電力を算出するために，クロック周波数の最大クロック周波数の設定を 520MHz とした．このため，計測された時間を 520000000 で除算した上で，消費電力を

クロック周波数 (MHz)	消費電力 (mW)
13	44
104	116
208	279
312	375
416	570
520	747

表 5.1: クロック周波数毎の消費電力

計算した. なお，消費エネルギーの単位は (mJ) である．[1]

タスク実行時間は，評価環境のタスクセットを実行に要した時間である．単位 clk は，プロセッサが最高速度で駆動したときの 1 サイクルの時間である.

5.2.1 ADP による DVS 適用時の消費電力

タスクスケジューリングに ADP を適用した場合のエネルギー消費とタスク実行時間を表 (5.2) に示す.

	消費電力 (mJ)	タスク実行時間 (clk)
DVS+ADP	665	1314748
DVS 不使用時 (参考)	1060	9752123
差分	-37%	+33%

表 5.2: ADP(DVS 実施)

5.2.2 静的優先度による DVS 適用時の消費電力

タスクスケジューリングに静的優先度を用い，DVS アルゴリズムを適用した場合の，エネルギー消費とタスク実行時間を表 (5.3) に示す.

5.2.3 スケジューリングアルゴリズムによるクロック周波数分布

(図 5.1) に各スケジューリングアルゴリズム別のクロック周波数分布を示す．ADP と DVS を併用した場合の分布が中央付近に集中しているの対して，静的優先度によるタス

[1] 1mJ = 1mW * 1sec.

	消費電力 (mJ)	タスク実行時間 (clk)
DVS+静的優先度	884	12052484
DVS 不使用時 (参考)	1060	9752123
差分	-16.6%	+24%

表 5.3: 静的優先度 (DVS 実施)

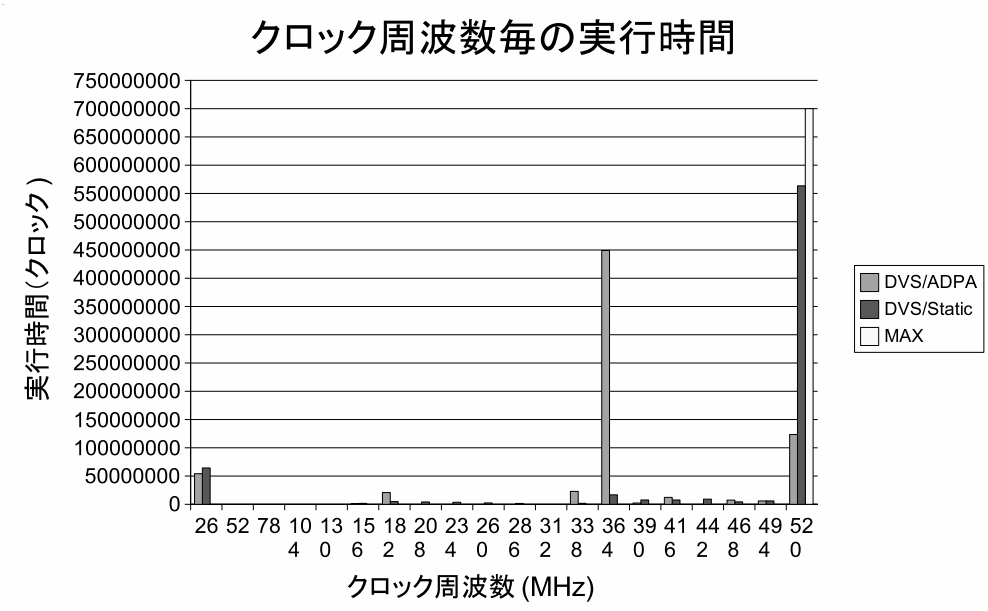


図 5.1: アルゴリズム別クロック周波数分布

クスケジューリングと DVS を併用した場合には，最大クロック周波数で実行された時間が長いことがわかる．この部分が両者の消費電力の差分となって現れている．

5.2.4 考察

DVS によってクロック周波数を最大速度よりも低下させるため，処理時間が増加する．ただし，同一の処理を完了するまでの消費電力が削減を削減可能である．

前項の結果から，ADP と DVS の組合せによる結果が最も電力削減効果が大きい．特に，処理時間の増分との関係では，ADP と DVS の組合せでは，33%の処理時間の増分に対して，37%の消費電力削減を実現した．一方静的優先度によるタスクスケジューリングと DVS の組合せでは，処理時間の増分 24%に対して，電力削減量が 17%に止まっている．この理由は，ADP の特性によって，デッドライン間際のタスクが優先的に実行され，タスクキューから早期に取り除かれるからである．例えば，静的優先度では，現在 2 番目で

実行待ちをしているタスクが Critical-Deadline 対象のデッドラインになっている場合，先頭タスクの余裕時間が大きくても，追い越しは発生せず，先頭タスクと 2 番目のタスクを高速に動作させる．しかし，ADP によって，2 番目のタスクが先頭に繰り上がって実行された場合，このタスクが終了した時点で，新たに Critical-Deadline を計算しなおす．このため，静的優先度では，高速で実行されたタスクが，より低速で実行できる場合がある．今回の実験では，評価タスクのデッドラインミス数は ADP/静的優先度/DVS なし共に同数であった．しかし，デッドラインミスの大きさを比較すると，表 (5.4) に示すように，ADP による場合は静的優先度に比べてミス量が小さく，全体の処理時間の伸長量に比較して，デッドラインミスを避ける方向となっていることがわかる．

	全タスク処理時間の伸長 (%)	デッドラインミスタスクの伸長 (%)
DVS+ADP	+33%	+9%
DVS+静的優先度	+24%	27%
DVS 不使用時	0%	0%

表 5.4: デッドラインミスタスクの処理時間伸長量

第6章　関連研究

本章では，ハードウェアスケジューラについての研究とμITRONのハードウェア実装に関する研究を報告する．[3]は，タスクスケジューラのハードウェアによる実装についての研究である．外部ハードウェアによるタスクスケジューラの実装にあたり，プロセッサとのスケジューリング情報の通信を確実に行いつつ，並行動作可能な処理を実施している．さらに，タスクスケジューリング方式にADPを利用し，外部ハードウェアの特性を活用して，ハードリアルタイムタスクのデッドラインミス時に，当該タスクの処理を強制的に中断することにより，静的優先度に比べて，同一タスクセットで2.5[7]はμITRONをVLSI化した研究である．タスクスケジューラだけではなく，オペレーティングシステム自体をVLSI化した研究である．OS全体をハードウェア化することでタスク切替のオーバヘッドを削減し，近年要求される数μsの高速応答性能を目指した研究である．この研究では，対象とする機器が特に強いリアルタイム性と信頼性が求められる自動車，航空機，宇宙機などとなっており，ソフトリアルタイムの機器をターゲットととする本研究とは異なる．なお，専用ハードウェアによる実装で，スケジューリング処理時間の削減を実現しており，7個のタスクを対象としたスケジューリング処理時間を0.40 μsで完了している．

第7章　まとめ

本研究では，タスクスケジューラが持つ，タスクのデッドライン，予測実行時間の情報を利用して，プロセッサのクロック周波数を制御するハードウェアを提案した．本ハードウェアは，2 種類の DVS アルゴリズムを組み合わせることで，DVS によるプロセッサの性能低下がデッドラインミスを発生させることを防止しつつ，消費電力の削減を図る．さらに，タスクスケジューラと統合した外部ハードウェアとして実装することで，タスクスケジューリング情報を DVS 制御とタスクスケジューリングの双方で利用することにより，DVS によるプロセッサの性能低下時にも，比較的複雑なスケジューリングアルゴリズムである ADP を利用することができる．

ハードウェアによるタスクスケジューラが，スケジューリング情報を利用した DVS 制御を行うこと，ADP により Critical-Deadline タスクを早期に実行し，レディーキューから排除することによって，DVS による処理時間の増大を上回る消費電力削減を実現した．

7.1　課題

消費電力削減を目的としてハードウェアを追加するために，追加したハードウェアが消費する電力が追加される．本研究ではプロセッサと同一のハードウェアを想定した上で，電圧・周波数を固定した状態でシミュレーションを行った．そのため，ハードウェアスケジューラの消費電力を過大に見積もられている．ハードウェアスケジューラが必要とする命令は限定的なため，プロセッサよりも小規模なハードウェアで実現可能であることが知られている [3]．従って，ハードウェア量の測定を行うことで，より正確な消費電力削減の効果を確認可能であると考えられる．

謝辞

本研究を行うにあたり，終始ご指導を頂いた北陸先端科学技術大学院大学情報科学研究科田中清史助教授に心から深く感謝いたします.

適切なご助言を頂いた，本学の日比野靖教授，井口寧助教授に深く感謝いたします.

そして，本学における研究，生活において常にご指導，ご助言頂いた，本学計算機アーキテクチャ講座の皆様に心から御礼申し上げます.

参考文献

[1] Thomas D. Burd, Trevor A. Pering, Anthony J. Stratakos, and Robert W. Brodersen “A Dynamic Voltage Scaled Microprocessor System” IEEE Journal of Solid-State Circuits, Vol. 35, No. 11, 2000

[2] 栗谷一路, 田中清史 “リアルタイム OS における適応型スケジューリング方式” 電子情報通信学会研究報告 CPSY， Vol.102, No.478, pp.127-132. 2002

[3] 大崎哲弥, 田中清史 “リアルタイム支援高機能割り込みコントローラの提案” 電気関係学会 北陸支部連合大会講演論文集, pp.208 2003

[4] C.L.Liu and J.W.Layland, ”Scheduling Algorithms for Multiprograming in a Hard Real Time Environment,” Journal of the ACM, Vol.20,No.1,pp.46-61,1973

[5] A.K.Mok, ”Fundamental Design Problems of Distributed Systems for the Hard Real Time Environment,” Ph.D.thesis,MIT Cambridge MA, 1983.

[6] Intel Corporation, ”Intel PXA270 Processor Electrical, Mechanical, and Thermal Specification Data Sheet” Intel Corporation, 2005.

[7] 仲野巧, アンディ ウタマ, 板橋 光義, 塩見 彰睦, 今井 正治, “リアルタイム OS の VLSI 化とその評価” 電子情報通信学会論文誌 NO.8 pp.679-686 1995.8