JAIST Repository
https://dspace.jaist.ac.jp/

Title	リアルタイムシステムにおけるハードウェアスケジュールに関する研究
Author(s)	大崎，哲弥
Citation	
Issue Date	2004-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1803
Rights	
Description	Supervisor:田中　清史，情報科学研究科，修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

リアルタイムシステムにおける
ハードウェアスケジューラに関する研究

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

大崎 哲弥

2004 年 3 月

修 士 論 文

リアルタイムシステムにおける ハードウェアスケジューラに関する研究

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　堀口進 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

210010 大崎 哲弥

提出年月: 2004 年 2 月

Copyright © 2004 by Osaki Tetsuya

概 要

リアルタイムシステムにおいてリアルタイム性を要求するタスク数の増加は，スケジューラによるプロセッサ占有率が本来実行すべきタスクの実行時間に影響を与え，デッドラインミスとなるタスク数の増加につながる．

本論文ではメインプロセッサと並列に動作するスケジューリング専用ハードウェアを提案し，シミュレーションを用いて評価することにより，小規模なハードウェア量で，メインプロセッサのスケジューリングオーバーヘッド軽減をおこない，平均で 12 パーセント，最高で 65.2 パーセントデッドラインミスとなるタスク数を減少させる手法を示す．

目 次

第 1 章　はじめに **1**
1.1 背景と目的 1
1.2 本論文の構成 1

第 2 章　リアルタイムシステム **3**
2.1 リアルタイム性 3
2.2 タスクの状態と属性 3
2.3 処理時間と価値 3
2.4 リアルタイムスケジューリング 5
2.5 駆動方式 6
2.6 リアルタイムスケジューリングアルゴリズム 7
2.6.1 静的優先度方式 7
2.6.2 RMA(Rate Monotonic Priority Algorithm)[2] 7
2.6.3 EDFA(Earlist Deadline First Algorithm)[2] 9
2.6.4 LLFA(Latest Laxity First Algorithm)[3] 10
2.6.5 ADPA(Adaptive Dynamic Priority Algorithm)[4] 11

第 3 章　ハードウェアスケジューラ **13**
3.1 高機能割り込みコントローラ 13
3.1.1 スケジューリングオーバヘッドの削減 13
3.1.2 デッドラインを越えた場合のハードリアルタイムタスク処理 15
3.2 ハードウェア仕様 15
3.3 スケジューリング方法 17
3.3.1 タスク管理 17
3.3.2 スケジューリングルーチン 19
3.3.3 実行タスクのデッドラインオーバー時の処理 21
3.3.4 実行タスクの切り替え時の処理 23
3.3.5 実行タスクの終了時の処理 23

第 4 章　評価 **27**
4.1 ハードウェア量 27

4.2 シミュレーション手法 27
4.3 シミュレーション結果 28
4.3.1 デッドラインミス数 28
4.3.2 応答時間 28
4.3.3 静的優先度 28

第 5 章 関連研究 **35**

第 6 章 おわりに **36**
6.1 まとめ 36
6.2 今後の課題 36

第 7 章 謝辞 **37**

第1章　はじめに

1.1　背景と目的

近年のLSI技術，マイクロプロセッサ技術の発展と共に，高性能なコンピュータシステムを低コストで実現することが可能となり，組込みシステムの適用分野は工場の生産ライン中心の産業機器から，通信機器，オフィス機器，家電機器等様々な分野へ急速に広がっている．これに伴い，高機能機器を制御するソフトウェアの需要が増大し，組込みソフトウェアは急速に大規模化・複雑化することとなった．そのため，組込みシステムにおいて，複数の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機会が増加している．

組込みシステムでは信頼性の高いリアルタイム性を保証しなければならない．そのため，どのタスクをいつ実行するかを管理するスケジューリング処理がデッドラインミスとなるタスク数に影響する．このことから，リアルタイム用スケジューリングアルゴリズムに関する様々な提案がされてきた．

スケジューリングアルゴリズムをソフトウェアで実装した場合，スケジューリング処理によるプロセッサの占有が本来のタスクの実行に影響を及ぼし，デッドラインミスとなるタスク数に影響を及ぼす場合がある．

一方で，ハードウェアによるスケジューリングアルゴリズム実装の提案がされている[5]．この場合，スケジューリング処理を専用ハードウェアがおこなうことによって，プロセッサが本来のタスク処理に従事できる．しかし，複雑なスケジューリングアルゴリズムの専用ハードウェアを用意した場合, 多大なハードウェア量が問題となる．

本論文では，プロセッサと並列に動作する様々なスケジューリングアルゴリズムが実装可能な小規模スケジューリング専用プロセッサを提案する．これをプロセッサコアの外部に付与することにより，プロセッサの有効実行率を向上させ，さらに，的確なタイミングでプロセッサにタスクの切り替えを通知することにより，デッドラインミスとなるタスク数を最小限に抑えることを目的とする．

1.2　本論文の構成

本論文の構成を以下に示す．

第２章 リアルタイムシステムについて説明する．

第３章 提案したハードウェアスケジューラについて説明する.

第４章 提案機構の評価を示す.

第５章 本論文の関連研究を紹介する.

第６章 本論文のまとめ，及び今後の課題について.

第2章　リアルタイムシステム

2.1　リアルタイム性

リアルタイム性とは，単に計算処理速度が速いことや応答時間が短いことをいうのではなく，定められた時間要件を満たして動作する性質をいう．

リアルタイムシステムにおいて，単一の処理のみを実行すべき場合は，計算機はその処理に専念することが可能なため，制限時間内に処理を完了させることは容易である．しかし，実行すべき処理が複数あり，処理を実行すべきタイミングがそれぞれ外部からあたえられる場合は，リアルタイムシステムはマルチタスクでなければならない．そのため，限られた処理能力の計算機で効率よくリアルタイム・マルチタスク処理をおこなうには，現在どのタスクを実行するかを決定するスケジューリングが重要となる．

2.2　タスクの状態と属性

タスクは３つの状態と様々な時間に関する属性を持つ．図 2.1 と図 2.2 に示す．タスクはブロック状態から，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ステムのスケジュール処理によって，適切な時刻に実行される．処理が終了した場合はブロック状態となり，次回の起動要求時刻を待つ．また，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．終了時間はタスク起動毎に決められている締切時間 (dead line) を守るようにあらかじめスケジューリングされなければならない．締切時間を越えることなく実行が完了する実行開始時の最大遅延時間を余裕時間 (laxity time) という．タスクは周期タスクと非周期タスクの２種類に分別される．周期タスクとはあらかじめ決められた間隔で起動要求が発生するタスクである．それに対し不定期に発生するタスクのことを非周期タスクという．

2.3　処理時間と価値

リアルタイムタスクはタスクによって，処理時間と価値の関係がある．本論文では大きくハードリアルタイムとソフトリアルタイムの２種類に分ける．以下にそれぞれの性質を持つ代表的な機器を示す．

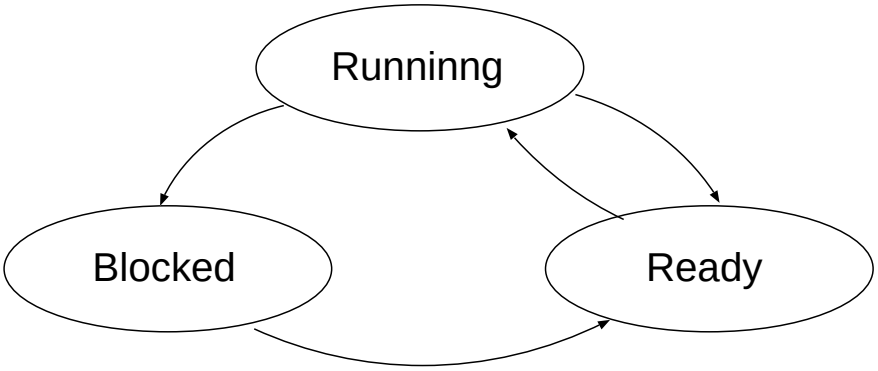


図 2.1: タスクの状態に関する図.

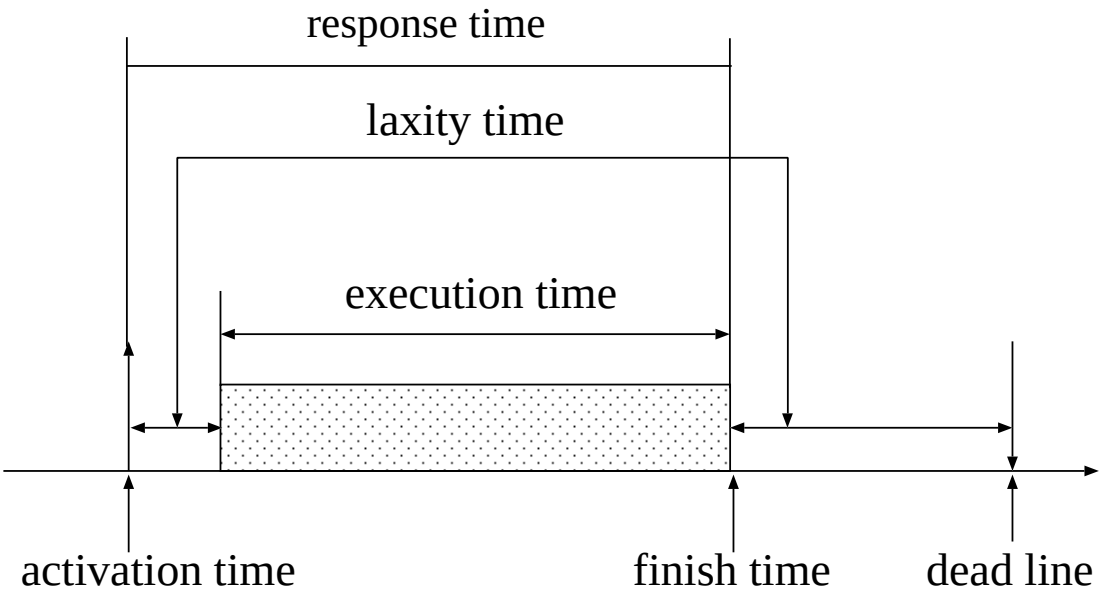


図 2.2: タスクの時間属性に関する図.

ソフトリアルタイム

1. プリンタ
2. エアコン
3. 自動販売機
4. 携帯ゲーム機
5. 電子楽器

ハードリアルタイム

1. ルータ
2. 伝送装置
3. NC 工作機械
4. ATM スイッチ
5. カーナビゲーションシステム

ソフトリアルタイムとは制限時刻を越えた場合にタスクの価値がなだらかに減少する．図 2.3(a) に示す．例にある電子楽器では，入力であるデジタルデータから出力のアナログデータに変換し，音を発生させるが，制限時間内に必ず処理を終了しなくとも，音の遅延はシステムとして重大な問題とはならない．同様に，自動販売機でも，商品選択ボタンによる入力からの応答が制限時刻内におこなわれなくとも，システムとして重大な問題とはならない．

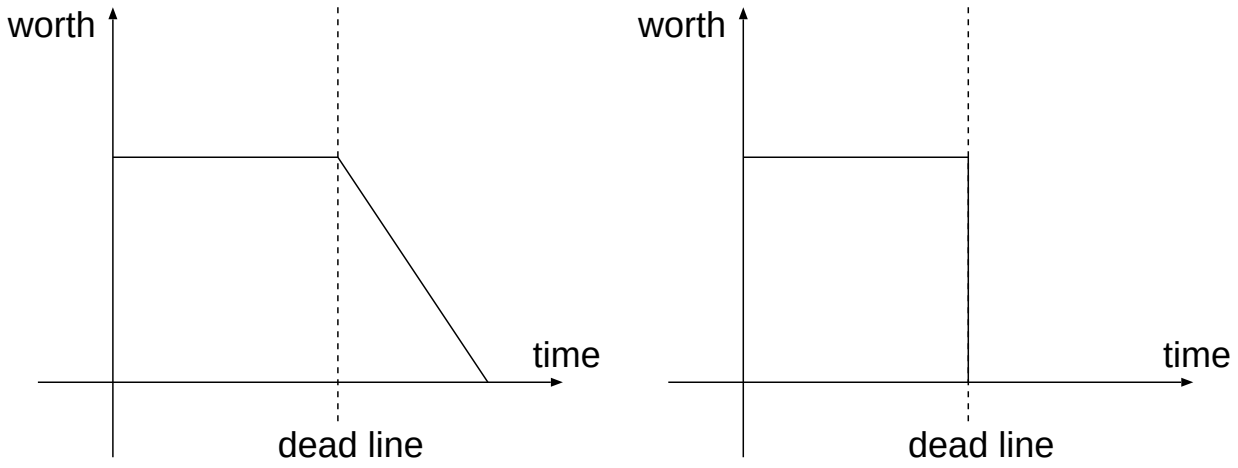
一方，ハードリアルタイムでは制限時刻を越えた場合，価値が急激に減少する．図 2.3(b) に示す．ルータでは，制限時間を越えて処理を行った場合は，パケットの発信元である端末がパケットが届かなかったと解釈し，再びパケットを送信するため，価値がゼロになる．また，カーナビゲーションシステムでは，本来曲がるべきはずの交差点を通過した後に，運転者に指示したとしても価値はゼロになる．

2.4 リアルタイムスケジューリング

複数のタスクを実行しなければならない場合，オペレーティングシステムはどのタスクを先に実行すべきか決定しなければならない．オペレーティングシステムの中でこの決定に関わる部分をスケジューラという．また，そこで用いられ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．

リアルタイムスケジューリングアルゴリズムの代表的な評価基準として以下の３つがある．

1. 応答時間：タスクが起動してから終了するまでの時間
2. 効率：プロセッサの有効実行効率
3. スループット：単位時間に処理可能なタスクの個数



(a) ソフトリアルタイムタスク.　　(b) ハードリアルタイムタスク.

図 2.3: タスクの価値関数に関する図.

2.5 駆動方式

プリエンプティブスケジューリング[1]において,，いつスケジューリングをおこなうかは駆動方式に依存する.

- 事象駆動方式
- 時分割方式

事象駆動方式は，外部事象[2] によってスケジューリング処理を実行する方式である．現在，リアルタイムオペレーティングシステム（以下リアルタイム OS）の大部分はこの方式を採用している．なぜならば，リアルタイム処理は時間制限をもった処理要求を満足しなければならない．このため，事象発生時に直ちにタスクの状態変化を反映したスケジューリングをおこなう必要があるからである.

それに対し時分割方式では処理時間を非常に短い時間に区切り，その単位毎に処理をおこなう．時分割方式では組込みシステムの採用例が少なく，複数のユーザーが端末を共有する UNIX などの汎用 OS で用いられている.

[1]プリエンプティブスケジューリング (preemptive schedluing) は実行タスクを一時中断し，他のタスクの実行をおこなうことが可能な方式をいう，また，それと対照的に処理を完了しないと他の実行可能タスクを実行できない方式をノンプリエンプティブスケジューリング (nonpreemptive schedling) という.

[2]外部事象とはタスクの状態を変化させる事態を指す.

2.6 リアルタイムスケジューリングアルゴリズム

代表的なリアルタイムスケジューリングアルゴリズムを取り上げ，事象駆動方式の例を使って説明する．

2.6.1 静的優先度方式

静的優先度方式ではタスク間の優先順位を基準にしてスケジューリングがおこなわれる．この方式を採用している例としてITORN[1]が挙げられる．タスク間の優先順位はアプリケーション開発者が各タスクの重要度に合わせ，あらかじめ段階的に決定した優先度を基準として，次のように決定される．実行可能なタスクが複数ある場合，その中で最も優先度の高いタスクが実行される．異なる優先度を持つタスクの優先順位は，高い優先度を持つタスクの優先順位が高くなる．等しい優先度のタスクの場合はFIFO(First In First Out)形式が採られる．図2.4に動作例を示す．各タスクの実行時間の先頭に付随する部分はOSがスケジューリング処理をおこなう時間とコンテキストスイッチの時間を合わせたものである．

最も静的優先度が低いタスク１が実行中にタスク２が起動する．スケジューラはタスク１とタスク２の静的優先度を比較し，より静的優先度の高いタスク２に実行を切り替える．さらに，タスク２の実行中にタスク３が起動する．スケジューラはタスク１とタスク２，タスク３の静的優先度を比較し，最も静的優先度の高いタスク３に実行を切り替える．そして，タスク３の実行が終了した後，スケジューラはタスク１とタスク２の静的優先度を比較し，より静的優先度の高いタスク２を実行させる．タスク２の実行が終了した後，スケジューラは残ったタスク１を実行させ，タスク１は終了する．そして，再びタスク２が周期による起動要求となり，タスク２を実行させ，終了する．

2.6.2 RMA(Rate Monotonic Priority Algorithm)[2]

RMAは周期タスクセットに対して起動周期の短いタスクほど優先度を高く設定する．さらに相対デッドラインは周期と同じ，つまり次の周期の起動時刻までに実行が終了することが条件である．RMAの動作例を図2.5に示す．

タスク１，タスク２，タスク３は周期タスクであり，デッドラインは周期と等しいとする．タスク１の実行中にタスク２が起動する．スケジューラはタスク１とタスク２の周期を比較し，より周期の短いタスク２に実行を切り替える．さらに，タスク２の実行中にタスク３の起動が発生する．しかし，タスク３に対し，実行タスク２の方が周期が短いため，実行タスクの切り替えはおこなわれず，タスク２の実行が続けられる．その後実行タスク２が終了し，実行待ち状態のタスク１とタスク３の中から選択されるが，タスク３の方が周期が短いため，タスク３が実行される．そして，タスク３の実行が終了し，残った実行待ち状態のタスク１が実行され，終了する．その後再びタスク２の起動要求時刻とな

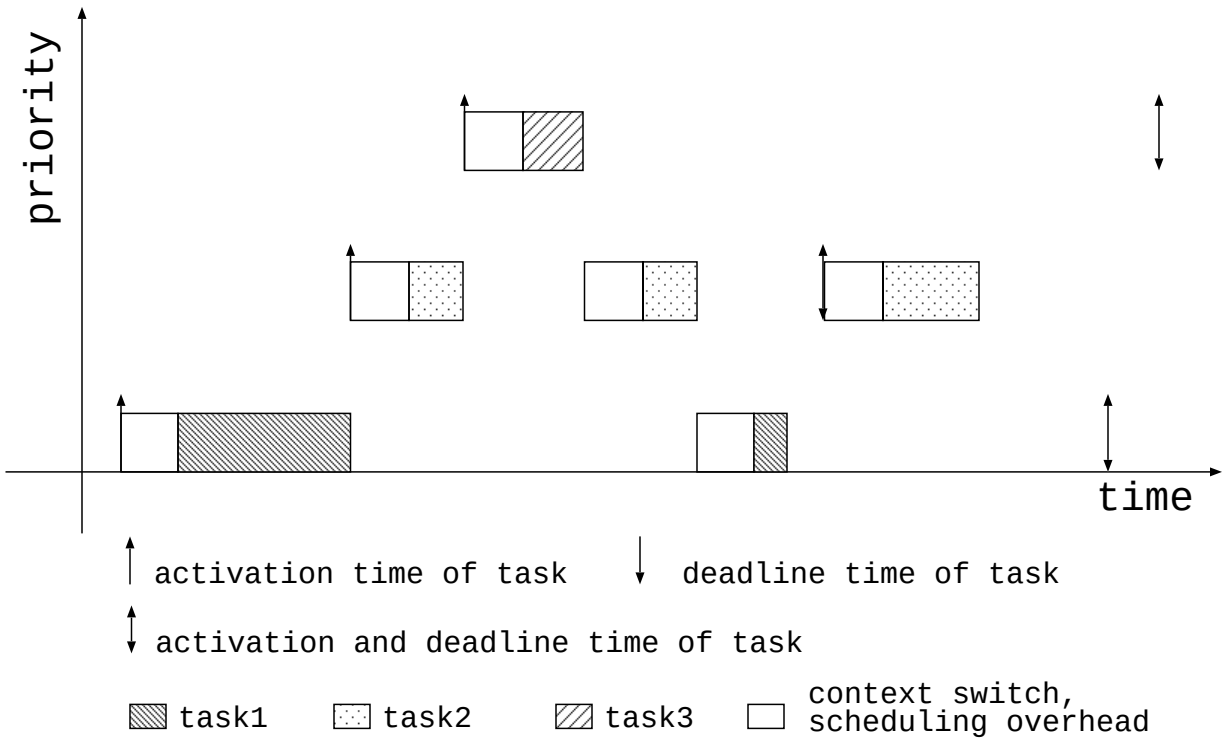


図 2.4: 静的優先度法を用いたスケジューリング図.

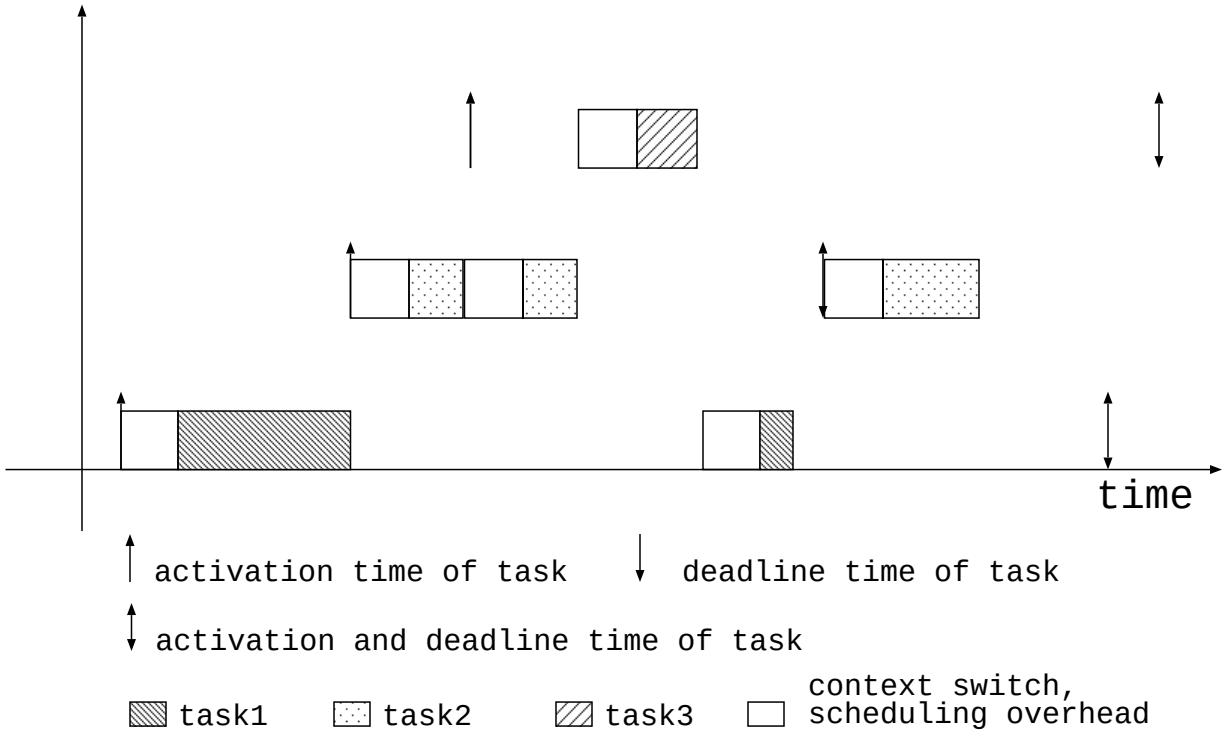


図 2.5: RMA によるスケジューリング.

り，タスク２が実行され，終了する．

2.6.3 EDFA(Earlist Deadline First Algorithm)[2]

EDFA は非周期的に起動されるタスクについてデッドラインの早い順に高い優先度を付ける方法である．EDFA は固定値の優先度に依存せず，デッドライン以外の条件は使用しない．そのため，周期タスクについても適応可能である．EDFA の動作例を図 2.6 に示す．

タスク１が実行中にタスク２の起動要求が発生する．ここで，スケジューラはタスク１とタスク２のデッドラインまでの時間を比較する．そして，タスク２の方がよりデッドラインまでの時間が短いため，タスク２を実行する．タスク２を実行中にタスク３の起動要求が発生する．しかし，タスク３に対し，タスク２の方がよりデッドラインまでの時間が短いため，タスク切り替えはおこなわれず，タスク２の実行を続ける．タスク２の実行が終了した際に，実行可能状態であるタスク１とタスク３のデッドラインまでの時間が比較される．そして，よりタスク１の方がデッドラインまでの時間が短いため，タスク３を実

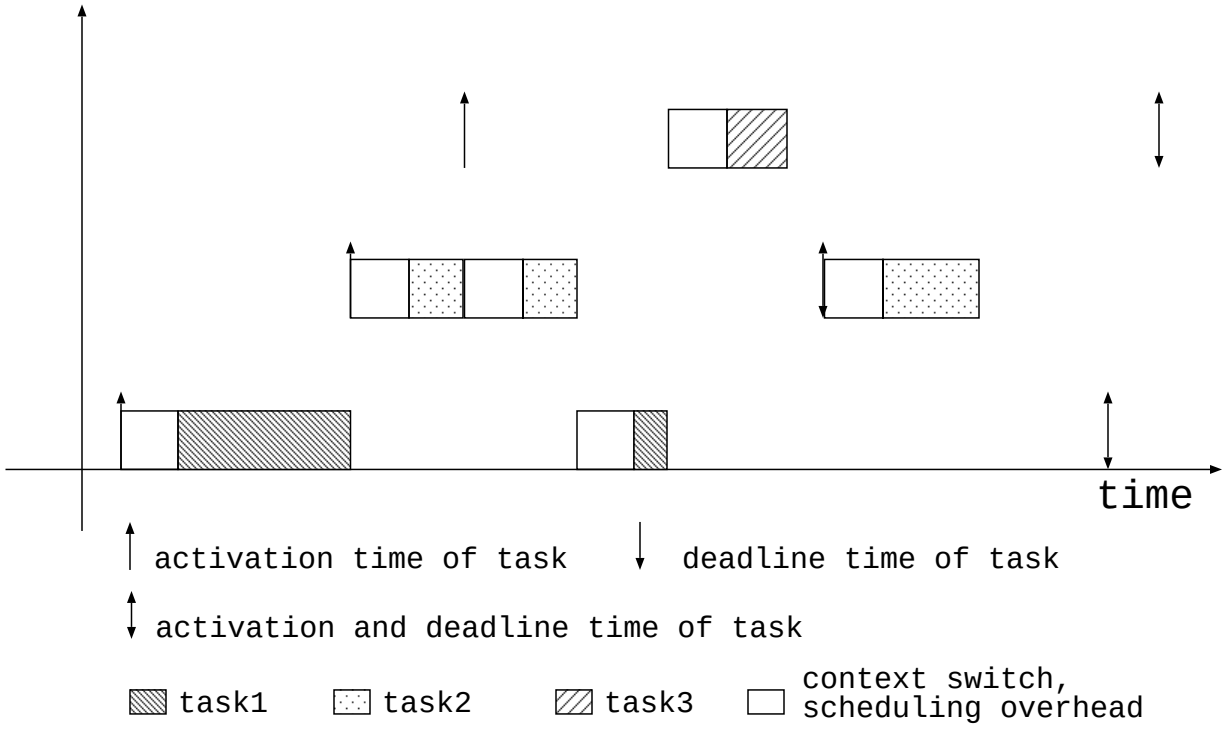


図 2.6: EDFA によるスケジューリング.

行する．そして，タスク 1 の実行が終了し，残った実行可能状態であるタスク 3 が実行される．その後終了し，タスク 2 の起動が発生し，タスク 2 が終了する．

2.6.4 LLFA(Latest Laxity First Algorithm)[3]

LLFA はタスクの余裕時間に着目し，余裕時間の一番小さいタスクに最高の優先度を与える．つまり，余裕のないタスクから先に実行する．LLFA では，タスクの余裕時間を計算するため，タスクの実行時間をタスクの実行前から見積もらなければならない．しかし，タスクの実行時間は常に一定ではないため，タスクの実行前から算出することは不可能である．そのため，タスク毎に WCET(Worst-Cace Execution Time) を見積もる必要がある．LLFA による動作例を図 2.7 に示す．

タスク 1 の実行中にタスク 2 の起動要求が発生する．スケジューラがタスク 1 とタスク 2 の余裕時間を比較した場合，タスク 2 の方が短いため，タスク 2 の実行に切り替わる．さらに，タスク 2 の実行中に，タスク 3 の起動要求が発生する．ここでタスク 2 とタ

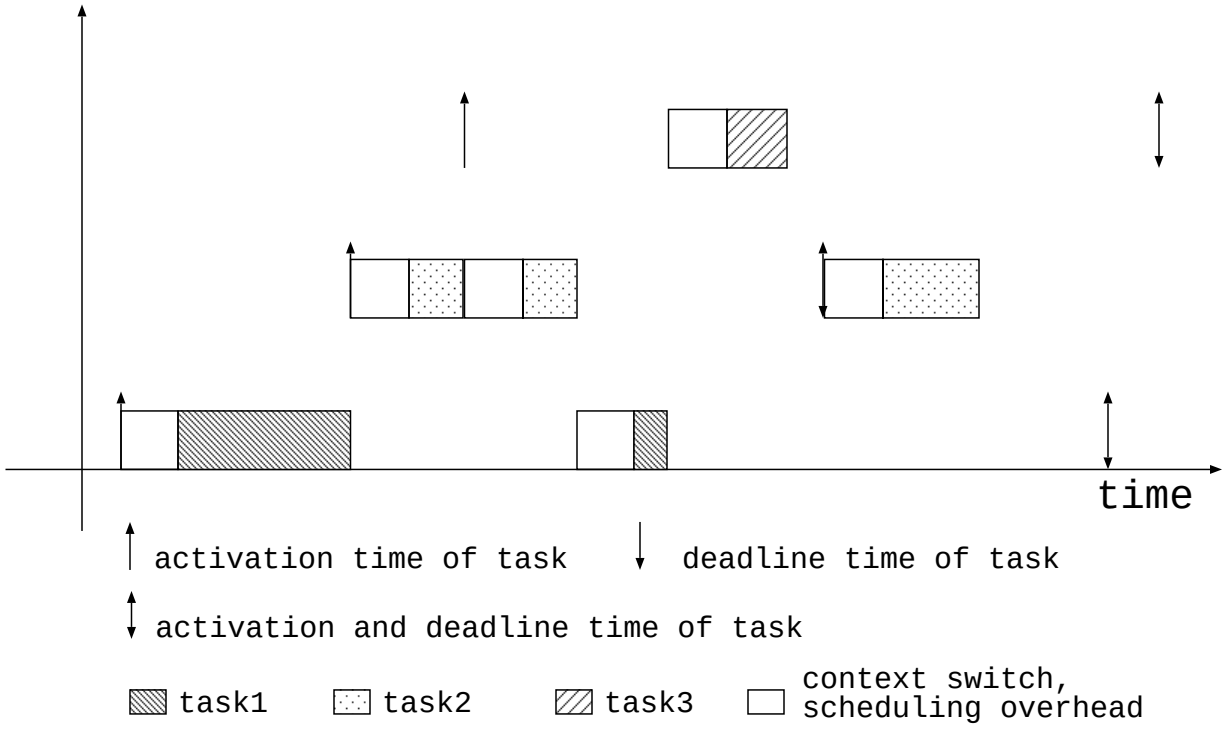


図 2.7: LLFA によるスケジューリング.

スク３，さらにタスク１を比べる．タスク２は余裕時間は変更されないが，タスク１は余裕時間が減少している．そのため，LLFA では一度決定した優先度の相対関係が時間の経緯にしたがって，変更される可能性がある．この場合はタスク２が最も余裕時間が短いため実行される．タスク２が終了すると，スケジューラはタスク１とタスク３の比較をおこなう．この場合タスク１の方がより余裕時間が短いため，タスク１が実行される．そしてタスク１の実行が終了すると，残った実行待ちタスクであるタスク３が実行され，終了する．そして，タスク２が起動し実行を完了する．

2.6.5 ADPA(Adaptive Dynamic Priority Algorithm)[4]

ADPA は静的優先度を基準とし，これに動的要因を導入することによって優先度を更新する．アプリケーション開発者が各タスクの重要度，及び実行順序を考慮して設定した値は意味的に重要であるため，静的優先度を基準と考え，動的要因として，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ている．余裕時間は LLFA に従い，余裕時間に

応じて優先度を高めに変更し，周期タスクに関してはRMAに従い，周期に応じて優先度を高めに変更する．動的優先度 $Pri_{dynamic}$[3]は次式であらわされる．

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

Pri_{static}
T:周期
L:余裕時間
γ , δ :定数

また，ADPAでは余裕時間の見積もりに関し，WCETを用いるのではなく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際の実行時間に基づいてPETの再計算を行い，次回の動的優先度の見積もりの計算に反映させる．この計算はタスクの終了時におこなわれ，終了したタスクの実行時間と前回のPETの値の加重平均から算出する．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．

$$PET_i = \beta \times Pri_{i-1} - (1 - \beta) \times (last\ execution\ time)$$

β :定数

[3] Pri_{static},$Pri_{dynamic}$ 共に値が小さいほど優先度が高い．

第3章　ハードウェアスケジューラ

本章ではスケジューリングオーバヘッド削減とデッドラインを越えたハードリアルタイムタスクの強制終了を可能とする高機能割り込みコントローラの提案をする．

3.1　高機能割り込みコントローラ

3.1.1　スケジューリングオーバヘッドの削減

リアルタイムシステムにおいて，リアルタイム OS がおこなうスケジューリングのオーバヘッドは最小限に抑えなくてはならない．なぜならば，有限であるプロセッサの処理能力は本来の目的であるタスク処理が可能な限り使用し，タスク管理処理であるスケジューラはメインプロセッサの使用を極力抑え，タスクのデッドラインミスを防がなくてはならないからである．

現在，大部分の組込みリアルタイム OS はソフトウェア実装かつ事象駆動方式である．この場合，事象が多発した場合，オペレーティングシステムのスケジューリング処理がメインプロセッサを占有し，本来のタスク実行をおこなうことができない．そのため，デッドラインミスにつながる．

高機能割り込みコントローラを使用した場合，高機能割り込みコントローラはすべての実行可能状態のタスク優先度を常時計算し，次回に実行する予定のタスクを用意する．また，外部要因であるタスクの起動要求に関してもプロセッサに直接通知されることなく，本コントローラ内で対応する処理をおこなう．これにより，メインプロセッサは事象発生時のスケジューリング処理が削減され，タスク処理に専念することが可能となり，デッドラインミスとなるタスク数を減少させることが可能となる．図 3.1 にスケジューリングオーバヘッド削減によるデッドラインミス数が減少する例を示す．

a 図，b 図ともにタスク実行中により優先度の高いタスクの起動要求が発生したため，より優先度が高いタスクへの実行タスクの切り替えがおこなれる．そして，優先度の高い実行タスクが終了した後に，元の優先度の低いタスクの実行に切り替わる．ここで，従来型である a 図ではスケジューリングオーバヘッドが大きいため，優先度の高いタスクはデッドラインミスとなる．さらに，優先度の低いタスクはその優先度の高いタスクの影響とオーバヘッドの大きさからデッドラインミスとなる．しかし，本コントローラを用いた b 図ではオーバヘッドはコンテキストスイッチのみであるため，優先度の高いタスク，優先度の低いタスク共にデッドラインミスは発生しない．

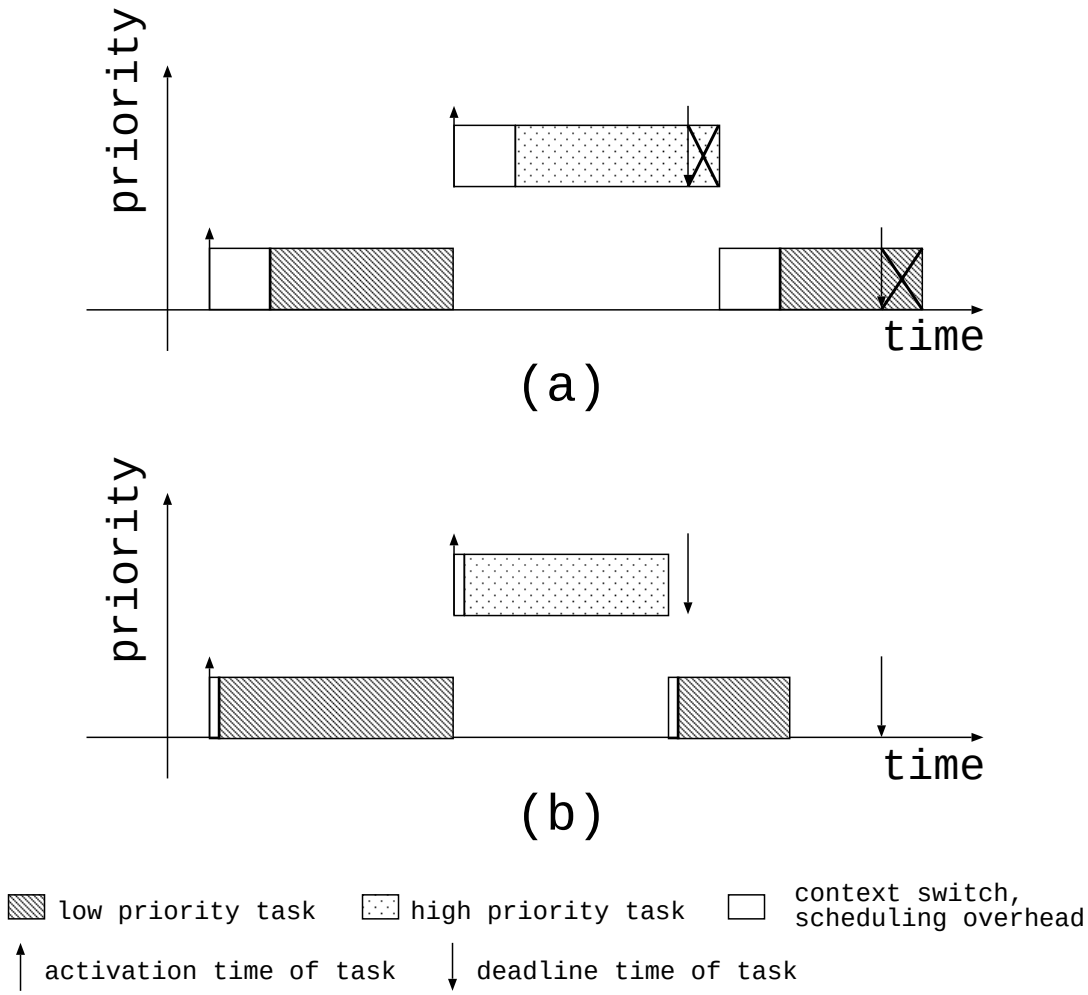


図 3.1: オーバヘッド軽減によるデッドラインミス削減例.

3.1.2 デッドラインを越えた場合のハードリアルタイムタスク処理

ハードリアルタイムタスクは制限時間を越えた場合，価値がゼロとなるため，制限時間を越えた後に実行する必要がない．しかし，従来のソフトウェアかつ事象駆動方式による実装方法では実行タスク，及び実行可能タスクの制限時間を過ぎた後に，次の事象が発生するまで，デッドラインを越えたことを判別することは不可能である．

本コントローラでは全ての実行可能タスクの制限時間を常時監視することにより，終了が必要な制限時間を越えたタスクを強制終了させることが可能である．

図 3.2 にデッドラインミスとなったハードリアルタイムタスクの実行を終了させることによりデッドラインミス数が減少した例を示す．c 図，d 図ともにタスク実行中により優先度の高いタスクの起動要求が発生したため，より優先度が高いタスクへの実行タスクの切り替えがおこなれる．そして，c 図，b 図共に優先度の高い実行タスクはそのままデッドラインミスとなる．ここで，c 図では優先度の高いタスクはデッドラインミス後も実行を続けることによって，優先度の低いタスクはデッドラインミスとなる．しかし，本コントローラを使用した d 図ではデッドラインミスとなったタスクの実行の継続をせず，優先度の低いタスクの実行に切り替えるため，優先度の低いタスクはデッドラインミスとはならない．

3.2 ハードウェア仕様

本節では提案した高機能割り込みコントローラの基本仕様について述べる．

本論文で提案した高機能割り込みコントローラを図 3.3(a)，図 3.3(b) に示す．

<u>高機能割り込みコントローラの基本仕様</u>

- 命令実行型 (MIPS-I)
- 16 ビットレジスタ
- 16 ビットメモリアドレス
- 専用キャッシュ(4K バイト，命令，データ)
- メインプロセッサへの割り込み機構

高機能割り込みコントローラは命令実行型である．これにより特定のスケジューリングアルゴリズムだけでなく，様々な種類のスケジューリングアルゴリズムを実装することが可能である．また，本コントローラはスケジューリング専用プロセッサとして動作する．本コントローラは組込みシステムとして利用されることを前提としているため，できる限り小規模なハードウェア量に抑える必要がある．そのため，本コントローラではスケジューリング実行をおこなう最低限の命令のみを用意することとし，乗算命令，除算命令

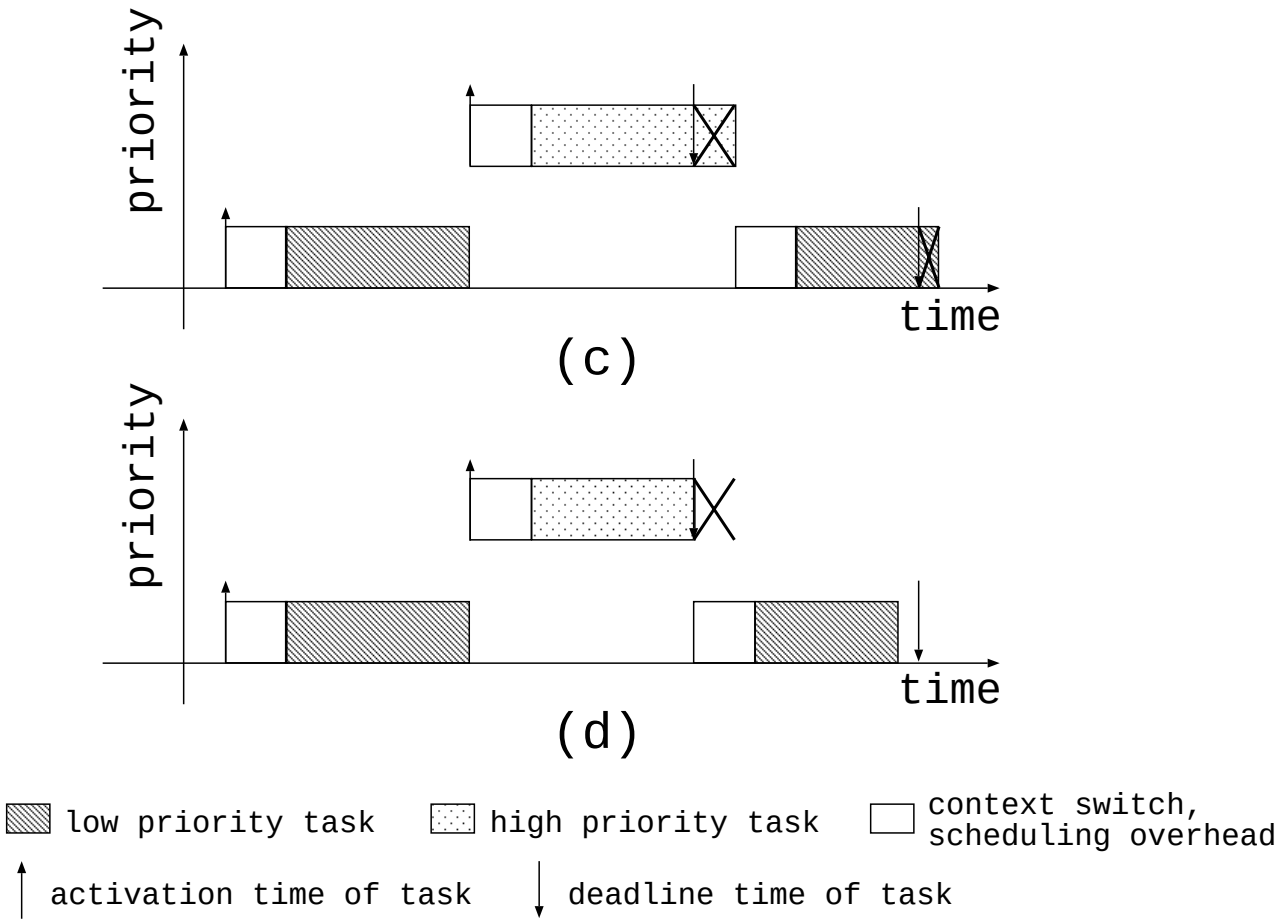


図 3.2: ハードリアルタイムタスクの強制終了によるデッドラインミス削減例.

などの多大なハードウェア量となる命令は用意しない．同様の理由により，スケジューリング実行可能な最小限のハードウェアとして，レジスタは１６ビットとする．

本コントローラでは，１６ビットメモリアドレス空間を使用することとし，メインプロセッサが使用するメモリアドレス空間の一部と共用する．メモリアクセスには専用データキャッシュ，及びインストラクションキャッシュを用いることによって，メインメモリにアクセスする機会を減少させ,．メインプロセッサのタスク実行とのメモリアクセス衝突の機会を抑える．

命令コードは最もコード量の大きい ADPA を実装した場合で，約２０Ｋバイトである．このことから，命令専用キャッシュは４Ｋバイトで十分である．また，各タスク情報のメモリ量は２０バイト以下であるため，タスク情報に関して使用するメモリはタスク数とタスク情報の使用するメモリ量の乗算で求められる．本論文では最大タスク数を１５０とする．タスク数１５０は組込みシステムにおいて十分に過大である．タスク数１５０において使用するメモリ量はスケジューラが演算に使用する領域を除いて最大３Ｋバイトである．したがって，４Ｋバイトデータキャッシュを用いた場合，タスク情報以外に使用するメモリ量が１Ｋバイト未満ではキャッシュミスは発生しない．また，タスク切り替えを通知する手段として，メインプロセッサへの割り込み機構を持つ．

3.3 スケジューリング方法

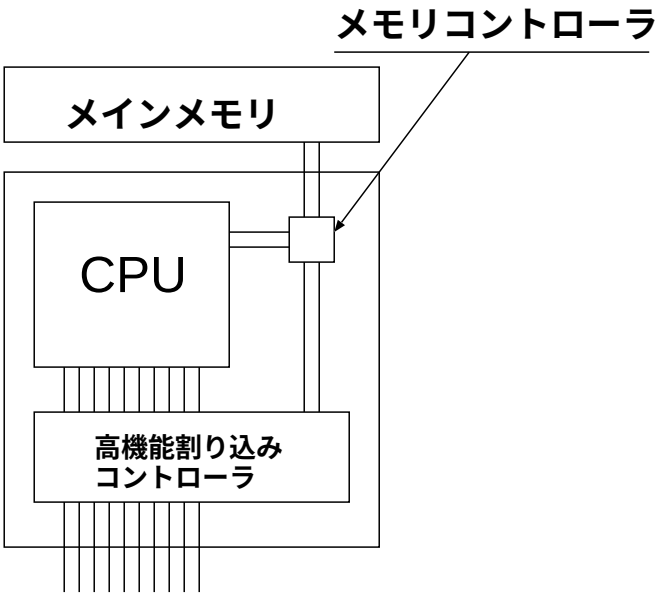
本節では高機能割り込みコントローラを使用したスケジューリング手法について示す．示すスケジューリング手法は全てのスケジューリングアルゴリズムに対応しているが，特に ADPA を対象としている．

3.3.1 タスク管理

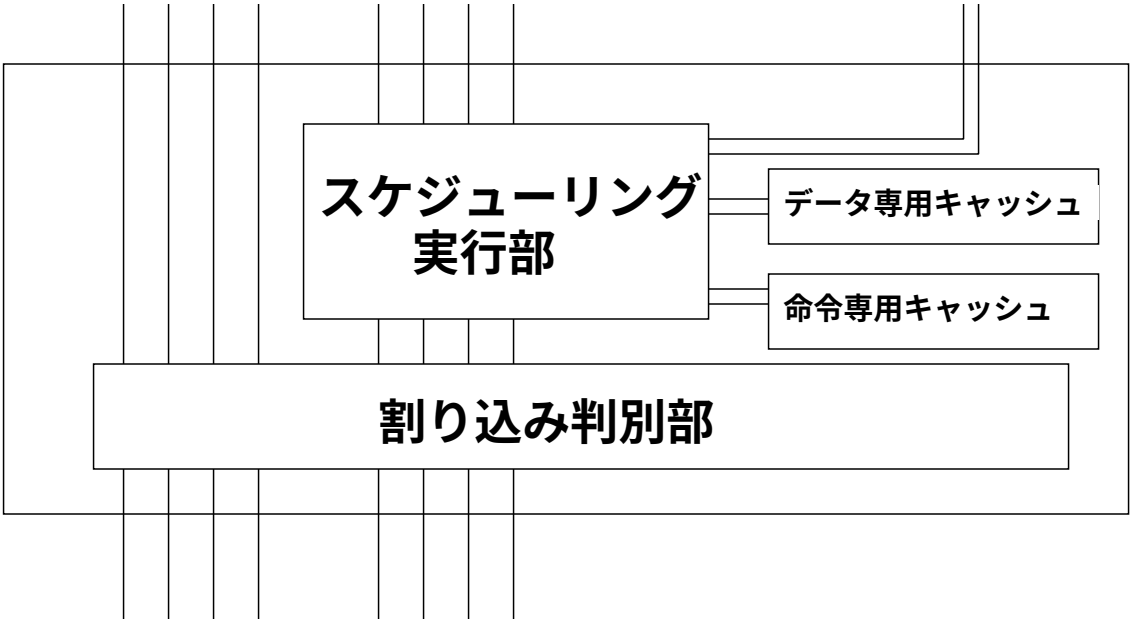
スケジューラはタスクを管理するために，各タスクの状態，属性を保持している．この情報はタスクディスクリプタに格納されている．

<u>タスクディスクリプタ</u>

- タスク ID
- 静的優先度
- ソフトリアルタイムかハードリアルタイムかのフラグ
- WCET
- PET



(a) 全体図.



(b) 高機能割り込みコントローラ詳細図.

図 3.3: 提案するシステム概要図

- 実行時間
- 絶対デッドライン
- 動的優先度
- 余裕時間
- タスクキューで使用する次タスクへのポインタ

メインプロセッサは起動時に実行予定の各タスク情報をタスクディスクリプタテーブルに格納する．ここで格納する情報はタスク ID，静的優先度，ソフトリアルタイムかハードリアルタイムかのフラグ，WCET である．本コントローラはこれらの情報を基に各タスクの PET，絶対デッドライン，動的優先度，余裕時間の計算をおこなう．

スケジューラはタスクディスクリプタを指す実行タスクエントリ，次回実行タスクエントリ，タスクキューによってタスクを図 3.4 に示す実行タスク，実行可能タスク管理リストによって管理する．これを実行タスクエントリはメインプロセッサが現在実行しているタスク，次回実行タスクエントリは次に実行するタスク，タスクキューのエントリはその他の実行可能タスクを示す．

起動要求が発生した場合，高機能割り込みコントローラは割り込みハンドラを起動し，起動要求が発生したタスクの ID を起動要求バッファに格納する．本コントローラは起動要求バッファを一定の間隔で検査し，対応するタスクをタスクキューに加える．そして，キューに加えられたタスクは優先度によって，次回実行タスクエントリ，実行タスクエントリに変更される．次節でスケジューラが常時おこなうルーチンについて説明する．

3.3.2 スケジューリングルーチン

本コントローラは常時，図 3.5 のルーチンを実行することにより，メインプロセッサが次に実行をおこなうタスクを用意する．また，必要に応じメインプロセッサの実行タスク切り替えを通知する．

スケジューリングルーチンは３つのブロックから構成される．タスクキューに関する a ブロック，次回実行タスクエントリに関する b ブロック，実行タスクエントリに関する c ブロックの３つである．コントローラが a ブロックを実行している間はメインプロセッサに制限を設けないが，b,c ブロックを実行している場合，メインプロセッサはタスクの実行終了処理を完了することができない．なぜなら，メインプロセッサは終了処理によって，実行タスクエントリ，及び次回実行タスクエントリを変更するため，実行タスク，実行可能タスク管理リストの整合性が保たれないからである．

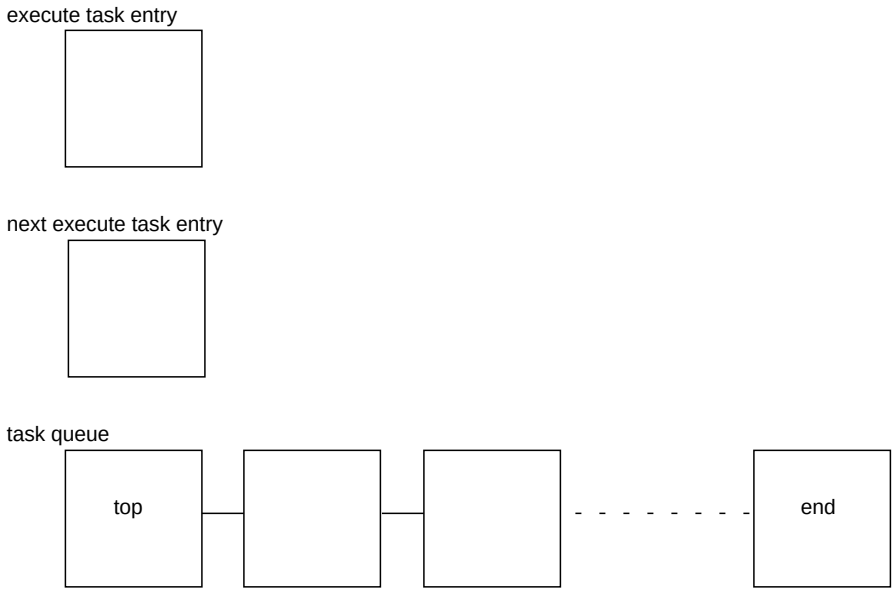


図 3.4: 実行タスク，実行可能タスク管理リスト．

a ブロック

a ブロックではタスクキューに関する処理をおこなう．スケジューリングのルーチンはタスクキューに登録されているすべてのタスクの優先度を更新する．また，このときタスクキュー内のハードリアルタイムタスクがデッドラインを越えていないかを検査し，越えていた場合はタスクキューから削除する．次に実行完了タスクが格納される終了バッファに格納されているタスクの終了処理をおこなう．ここで，ADP アルゴリズム実装の場合，終了したタスクの PET の計算をおこなう．そして，起動要求からの割り込みハンドラによって起動要求バッファに格納されたタスク ID を確認し，存在した場合はタスクキューに加える．そして，タスクキュー内の全てのタスクの優先度を比較し，最も優先度の高いタスクをタスクキューの先頭にする．

b ブロック

b ブロックでは次回実行タスクエントリについての処理をおこなう．最初にロックをかけ，このブロックでのメインプロセッサのタスク実行終了があった場合はロックが解除されるまでタスク終了処理を待たせる．後述するメインプロセッサの終了処理によって既にロックがかかっていた場合は，ロックが解除されるまで待つ．そして，次回実行タスクエントリに登録されているタスクがハードリアルタイムタスクだった場合，デッドラインを

越えていないかどうかの検査をおこない，越えていた場合は次回実行タスクエントリから削除する．次に次回実行タスクの優先度を更新する．そして，次回実行タスクとタスクキューの先頭タスクとの優先度の比較をおこなう．タスクキューの先頭タスクの方が優先度が高かった場合，次回実行タスクエントリにある次回実行タスクをタスクキューの最後に追加し，次回実行タスクエントリにタスクキューの先頭タスクを登録する．そして，タスクキューの先頭のタスクをタスクキューから削除する．タスクキューの先頭タスクの方が優先度が低かった場合は特別な処理はおこなわれない．最後に，このブロックの最初にかけたロックを解除する．

c ブロック

c ブロックでは実行タスクエントリに関する処理をおこなう．最初にロックをかけ，このブロックでのメインプロセッサの実行タスク終了処理を待たせる．既にメインプロセッサのタスク終了処理によってロックがかかっていた場合は，ロックが解除されるまで待つ．実行タスクエントリに登録されている実行タスクがハードリアルタイムタスクであった場合，デッドラインを越えていないかを検査する．越えていた場合は後述する実行タスクがデッドラインを越えていた場合の処理をおこなう．実行タスクの優先度の更新はおこなわない．なぜならば，全てのスケジューリングアルゴリズムでは実行タスクの優先度の変更はおこなわれないからである．そして，最初にかけたロックを解除する．

続いて，ロックをかけ，メインプロセッサのタスク実行終了処理を待たせる．既にメインプロセッサのタスク終了処理によってロックがかかっていた場合は，ロックが解除されるまで待つ．次に実行タスクと次回実行タスクの優先度の比較をおこなう．次回実行タスクの優先度の方が高かった場合，後述する次回実行タスクの優先度の方が高かった処理をおこなう．実行タスクの優先度の方が高かった場合は特別な処理はおこなわれない．そして，ロックを解除する．

3.3.3 実行タスクのデッドラインオーバー時の処理

ハードリアルタイム性を持つ実行タスクがデッドラインオーバーした場合，実行タスクの ID をデッドラインオーバーしたフラグと共に終了バッファに格納する．そして次回実行タスクを実行タスクエントリに登録する．ここで次回実行タスクエントリに次回実行タスクが存在しないことを示す NULL の場合にも同様の処理をおこなう．そして，次回実行タスクエントリを NULL とし，メインプロセッサに対し割り込み信号線を使い実行タスクの終了を通知する．その後，高機能割り込みコントローラはメインプロセッサがロックを解除するまで待つ．割り込みを受けたメインプロセッサは割り込みハンドラを起動する．割り込みハンドラでは，実行タスクエントリが NULL かどうかを判断し，NULL だった場合はロックを解除し，次節で述べるタスク切り替え処理によって実行タスクが決定されるまで待つ．NULL でなかった場合は実行タスクエントリのタスクを割り込みハンドラ

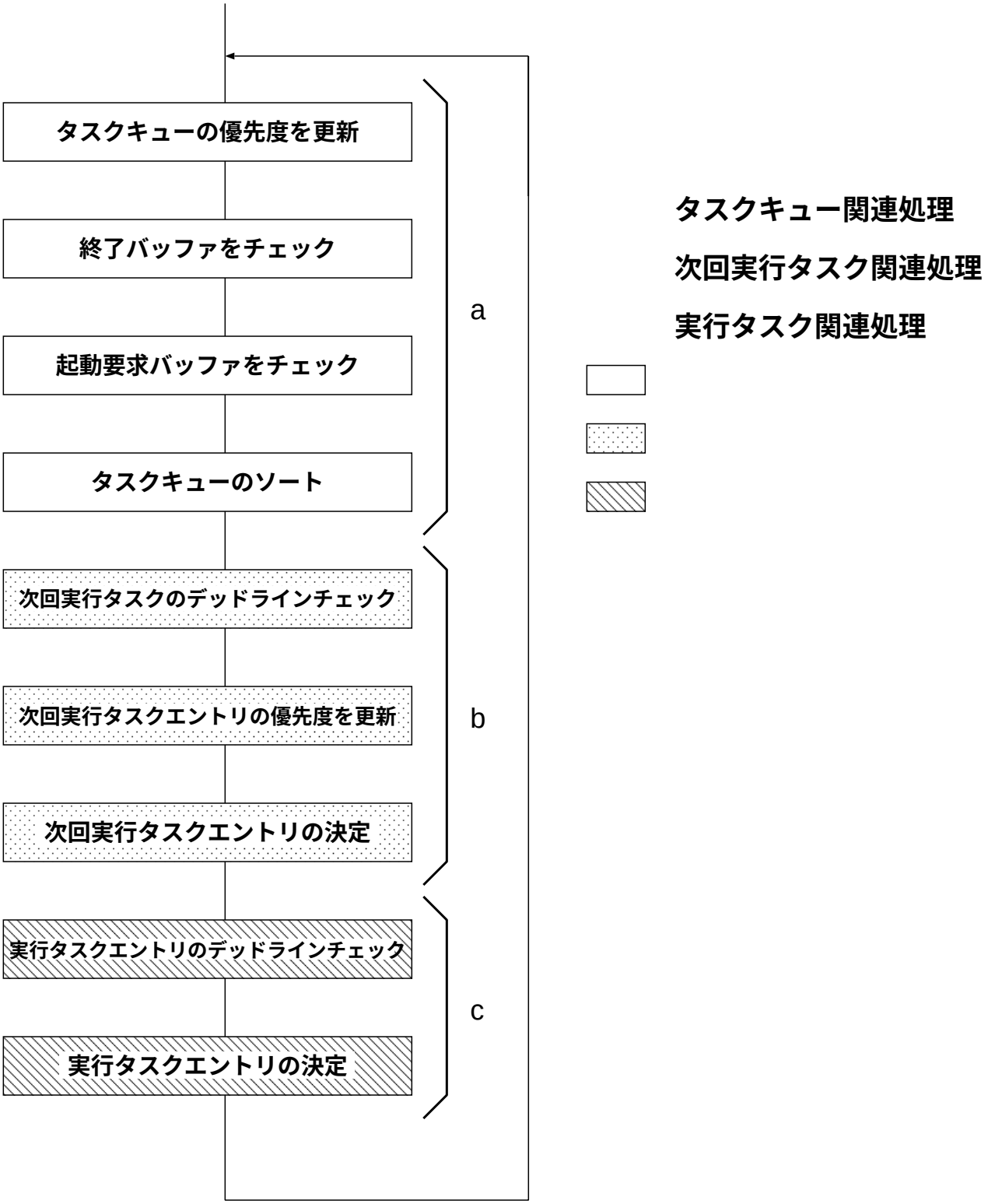


図 3.5: スケジューリングルーチン.

終了後の実行タスクとして準備をする．準備完了後に，ロックを解除しハンドラを終了する．ロックが解除されたことにより，高機能割り込みコントローラは再び元のスケジューリングルーチンの実行を再開する．デッドラインオーバー時の処理の流れを図 3.6 に示す．

3.3.4 実行タスクの切り替え時の処理

次回実行タスクが実行タスクの優先度より高い場合，実行タスクの切り替え処理がおこなわれる．まず，実行タスクエントリと次回実行タスクエントリの登録タスクを入れ替える．そして，メインプロセッサに対し割り込み信号を使用することによりタスク切り替えを通知する．そして，高機能割り込みコントローラはメインプロセッサによるロック解除を待つ．メインプロセッサでは割り込みを受け，割り込みハンドラを起動し，実行タスクエントリのタスクを実行する準備をおこなう．そして，ロックを解除し，割り込みハンドラを終了することにより，切り替えられたタスクを実行する．一方，割り込みコントローラではロックが解除されたことにより，再び元のスケジューリングルーチンの実行を再開する．タスク切り替え時の処理の流れを図 3.7 に示す．

3.3.5 実行タスクの終了時の処理

実行タスクの終了は高機能割り込みコントローラでは検出することができないため，メインプロセッサ側で処理をおこなう．メインプロセッサはタスクの終了を検知した場合，ロックをかける．ここで，メインプロセッサはロックを所得できなかった場合，タスクを終了することはできず，高機能割り込みコントローラのロック解除を待つ．もし，高機能割り込みコントローラによるロック解除がおこなわれる前に前述の実行タスクのデッドラインオーバー，及び実行タスクの切り替えがおこなわれた場合，そのタスクの実行は終了したことにならず，次回そのタスクが実行となったときに終了する．ロックを所得できた場合，終了バッファに実行タスクエントリの ID を格納し，次回実行タスクエントリの値を確認する．確認した値が NULL の場合はロックを解除し，高機能割り込みコントローラによる実行タスク切り替え処理を待つ．NULL でなかった場合は次回実行エントリのタスクを実行タスクエントリに登録し，次回実行エントリは NULL とする．そして，ロックを解除し，実行タスクエントリに登録されているタスクの実行を開始する．実行タスク終了時の流れを図 3.8 に示す．

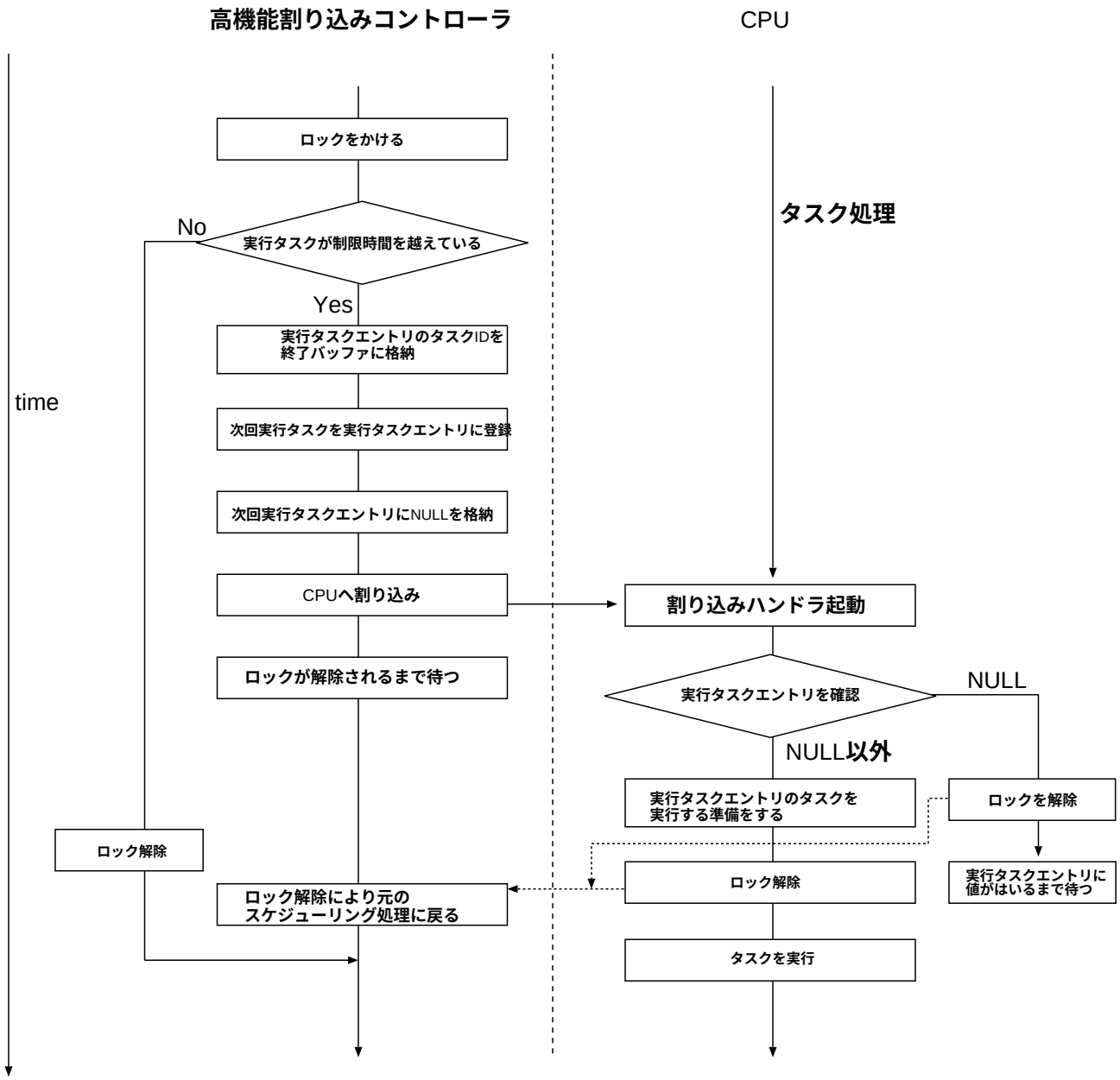


図 3.6: デッドラインオーバー時処理.

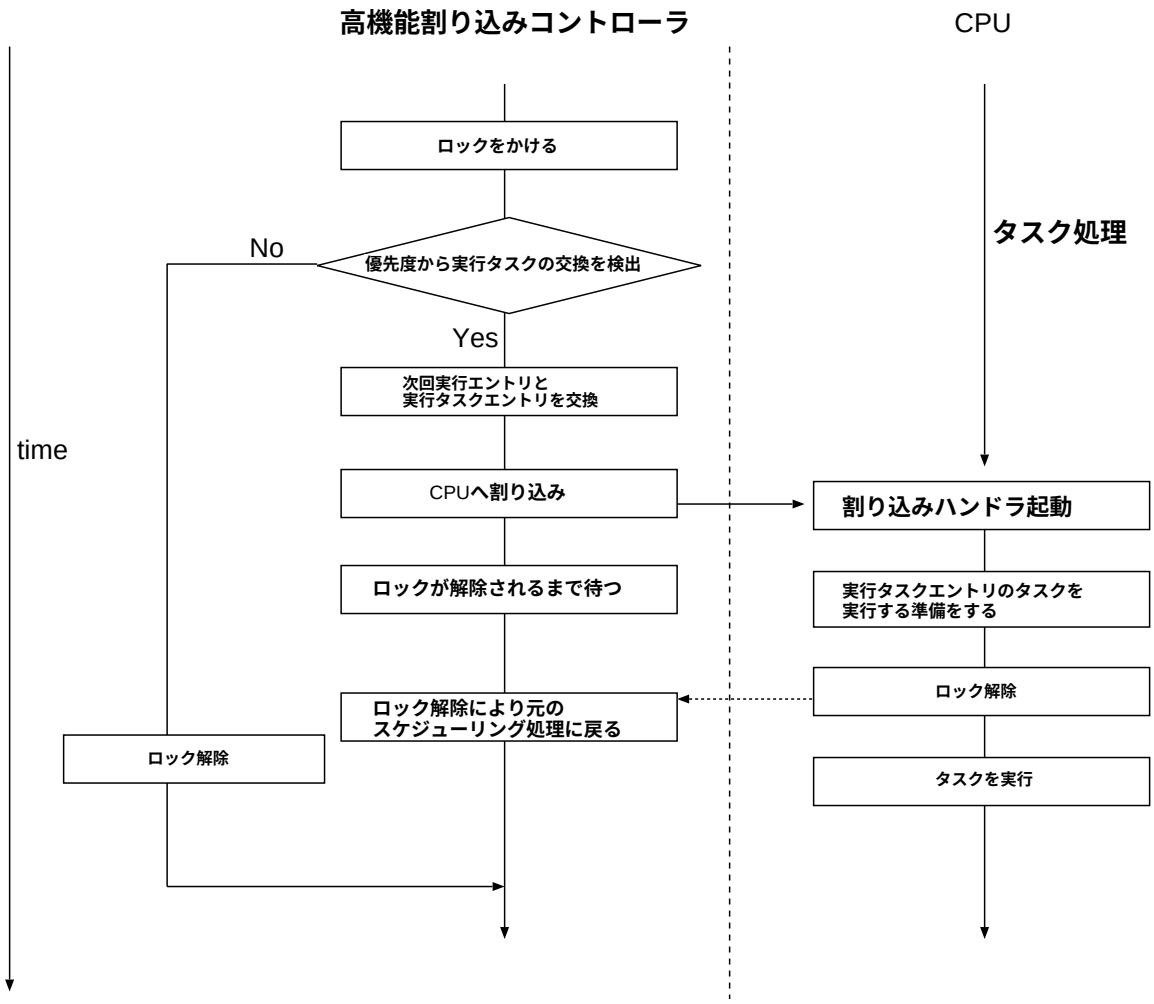


図 3.7: タスク切り替え処理.

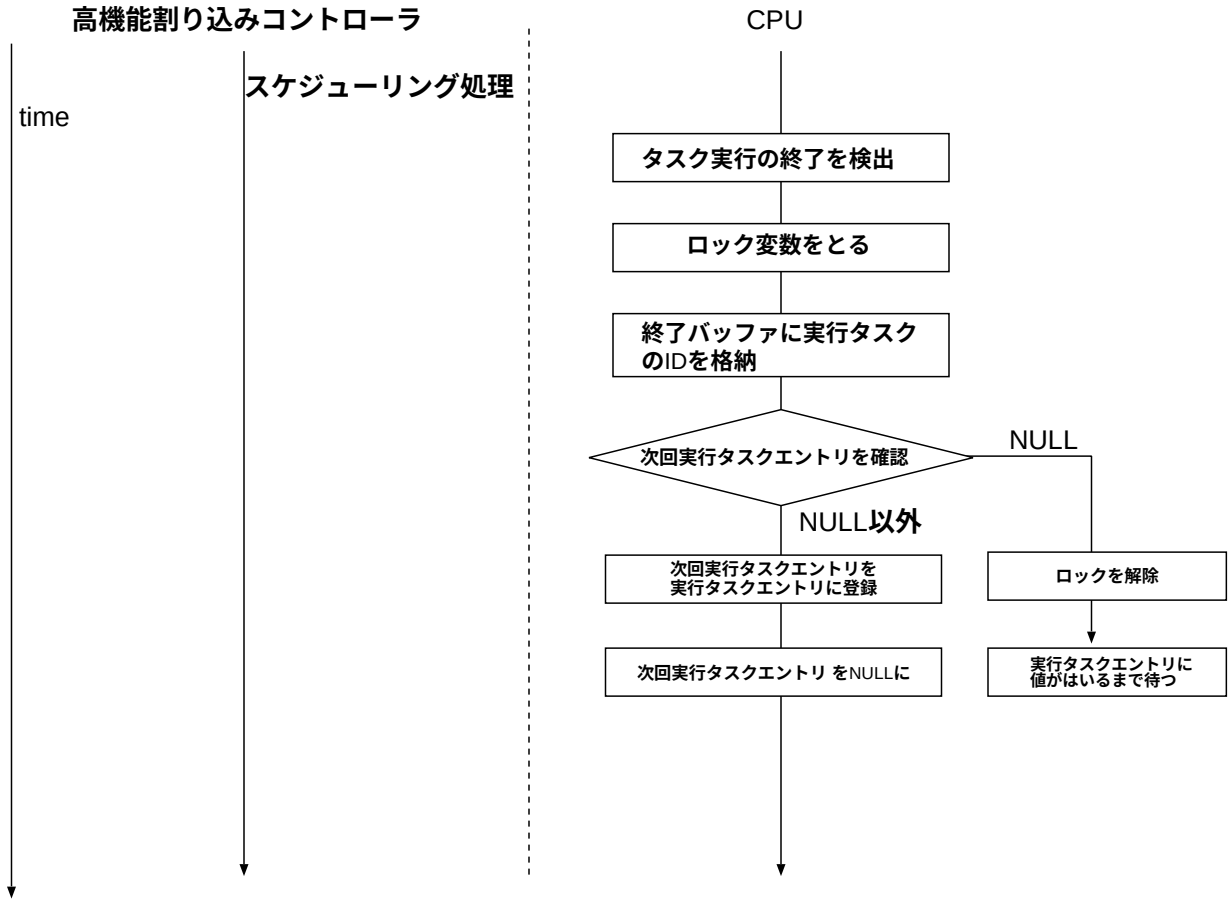

高機能割り込みコントローラ
CPU
time
スケジューリング処理
タスク実行の終了を検出
ロック変数をとる
終了バッファに実行タスク
のIDを格納
次回実行タスクエントリを確認
NULL
NULL以外
次回実行タスクエントリを
実行タスクエントリに登録
ロックを解除
次回実行タスクエントリ をNULLに
実行タスクエントリに
値がはいるまで待つ

図 3.8: タスク終了処理.

第4章　評価

本章では，高機能割り込みコントローラのハードウェア量と，代表的なスケジューリングアルゴリズムを実装した場合の性能を評価する．

4.1　ハードウェア量

前章で説明したハードウェア仕様を基に，高機能割り込みコントローラをハードウェア記述言語を用いて設計をした．ハードウェア量の評価は 0.25 μ m の ASIC ライブラリを用いて，Synopsys 社の Design Compiler を使 p 用した．ゲート規模は NAND 換算で 5022 ゲート相当であり，同一コンパイラを使用した場合，Booth-Wallace 型の乗算機が 11104〜11464 ゲート程度であることから，本コントローラは非常に小規模なハードウェアである．

4.2　シミュレーション手法

ソフトリアルタイム，ハードリアルタイムタスクが混在するシステムにおいて，メインプロセッサがタスク処理をおこなう一方で，高機能割り込みコントローラが並列してスケジューリングをおこなうことを仮定する．

1. 周期タスク：全てハードリアルタイムタスク，周期は５００, ０００〜 １００, ０００クロックの間
2. 非周期タスク：１対１の割合でハードリアルタイムタスク，ソフトリアルタイムタスク
3. デッドライン：周期タスクの場合周期の５０〜３０％
4. WCET： デッドラインの１０〜５０％
5. 静的優先度：８段階
6. 観測時間：１００, ０００, ０００クロック
7. コンテキスト切り替えオーバヘッド：５００クロック
8. CPU 使用率５０％程度

以上の条件でタスクをランダムに発生させ，シミュレーションをおこなった．

4.3 シミュレーション結果

図 4.1〜4.11 において各アルゴリズムの名称の後に H がつくものはハードウェアスケジューラを使用したシステム，つかないものは使用していないシステムをさす．また，各名称の最後に＋がつくものはスケジューリング時にハードリアルタイムタスクのデッドラインミスを検知した場合に実行を終了させるシステムを，つかないものは終了させないシステムを指す．各ミス率はパーセントとして表示してある．また，平均応答時間の単位はクロックである．

4.3.1 デッドラインミス数

図 4.1 に示すデッドラインミス率は各アルゴリズムで比較した場合，ハードウェアスケジューラを使用することによって，平均で 60 %デッドラインミスが減少した．

また，最も効果が高かったのは EDFA を使用した場合で，74 %デッドラインミスが減少する効果が得られた．最も効果が低かったのは RM を使用した場合で，53 %デッドラインミスが減少した．

4.3.2 応答時間

ハードウェアスケジューラを使用することによって，図 4.2 に示す応答時間は平均で 9 %減少した。

ハードウェアスケジューラを使用することによって，最も応答時間が短縮されたのは LLFA を使用した場合で、11 %短縮された。最も応答時間が短縮されなかったのは RMA を使用した場合で、応答時間は 6 %短縮された。

4.3.3 静的優先度

本研究ではアプリケーションエンジニアの与えた静的優先度を重視し，静的優先度の評価に関し，静的優先度に重みを与えた独自の評価方法を使用する．評価方法は８段階の静的優先度に対し，最も優先度の高い静的優先度１を持つタスクのデッドラインミス率に８をかけ，次に優先度が高い静的優先度２を持つタスクのデッドラインミス率は７をかける．同様に優先度８までおこない優先度が低い順に乗算する値を小さくする．そして，それらの総和を評価基準とする．値が低いほど静的優先度の高いタスクのデッドラインミス率が小さい．

図 4.3〜図 4.10 に静的優先度別のタスクのデッドラインミス率を，図 4.11 に重みづけの測定値を示す．ハードウェアスケジューラを使用することによって，測定値は平均で 0.4 %増加した．

最も測定値が減少したのは EDFA を使用する場合であり 1.1 %減少した．最も測定値が

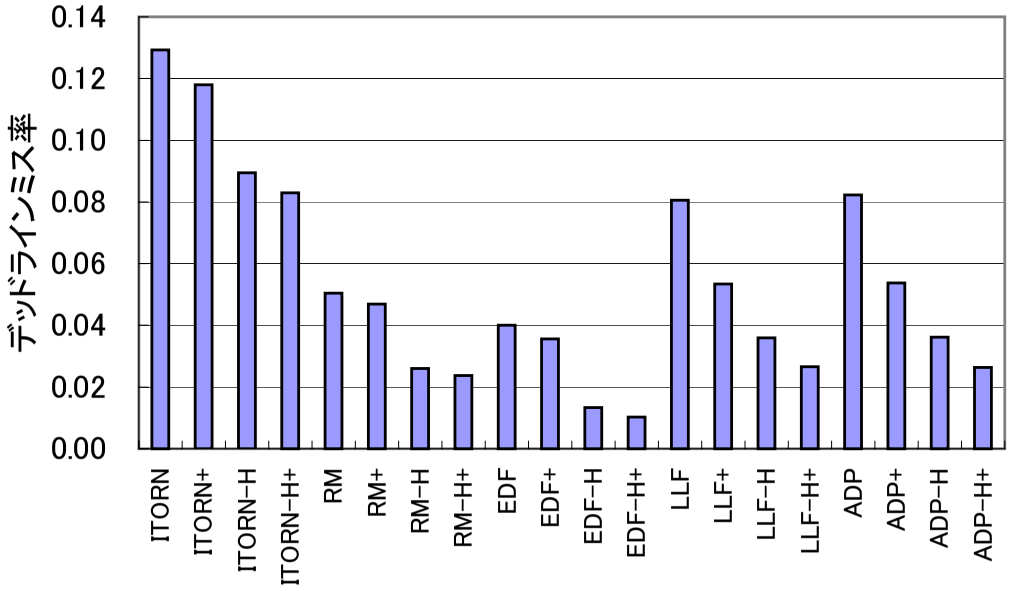


図 4.1: デッドラインミス率.

減少しなかったのは ITORN の場合であり 4.3 %増加した.

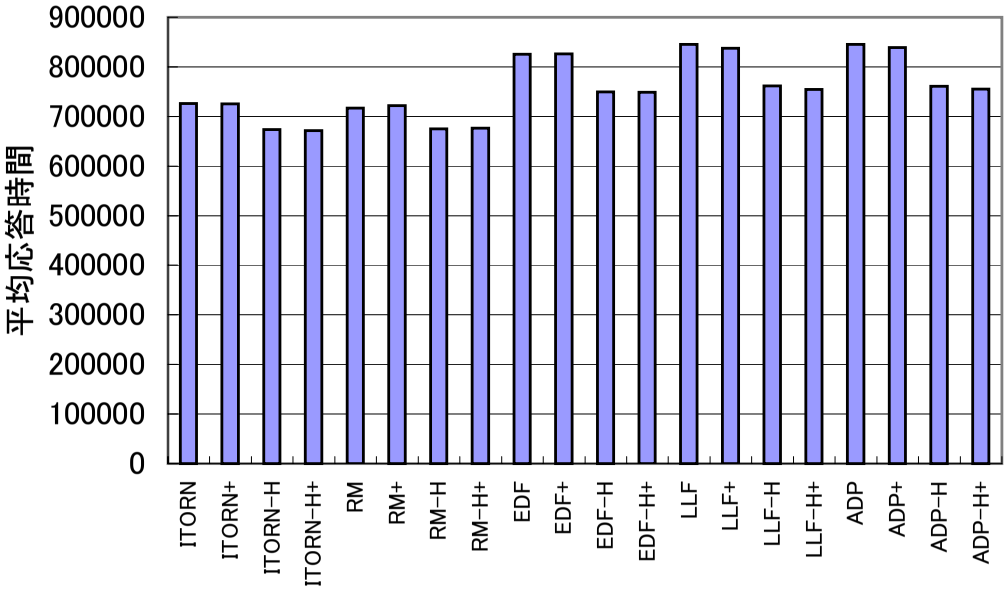


図 4.2: 平均応答時間.

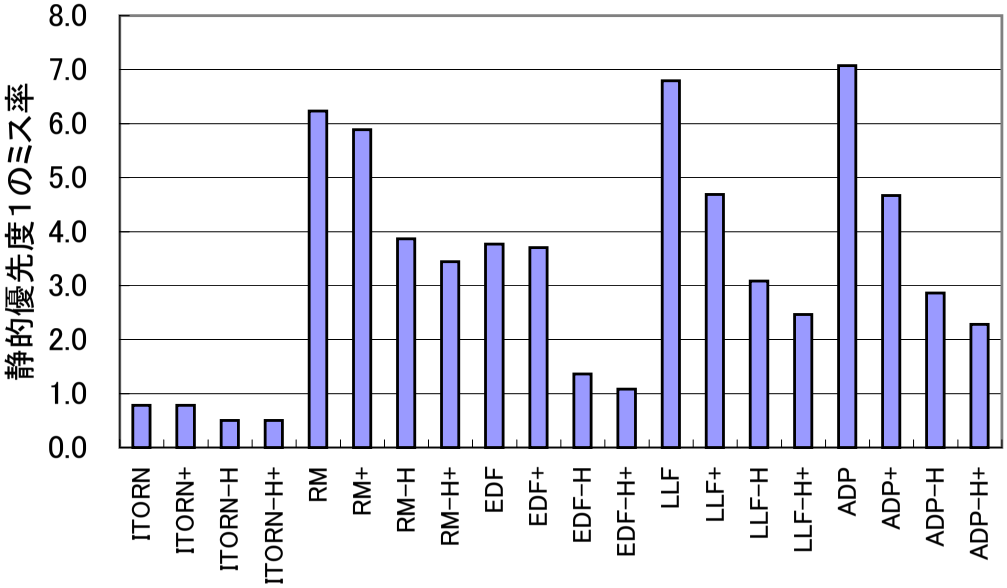


図 4.3: 静的優先度１のタスクにおける平均ミス率.

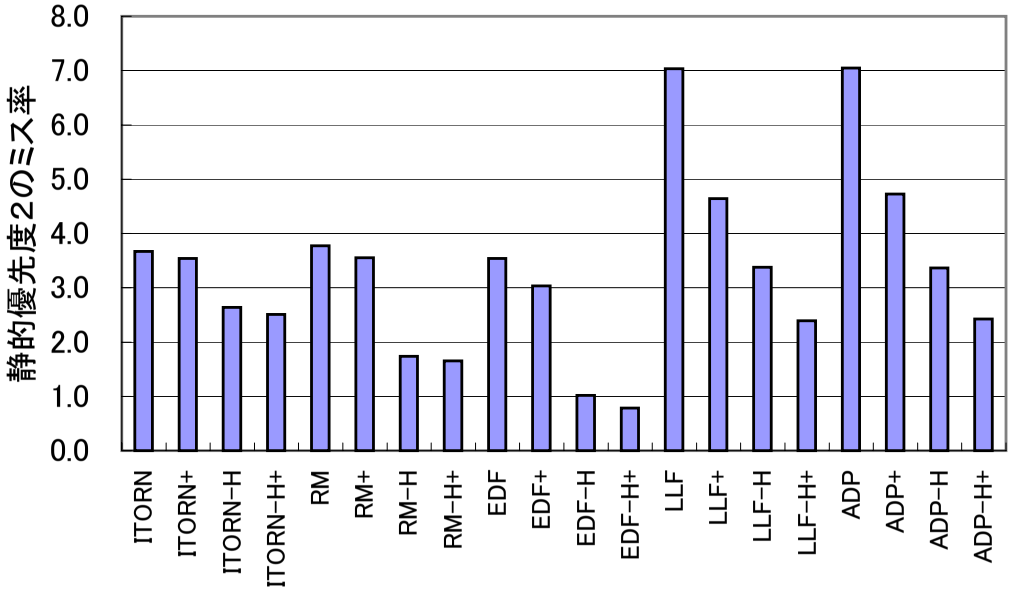


図 4.4: 静的優先度 2 のタスクにおける平均ミス率.

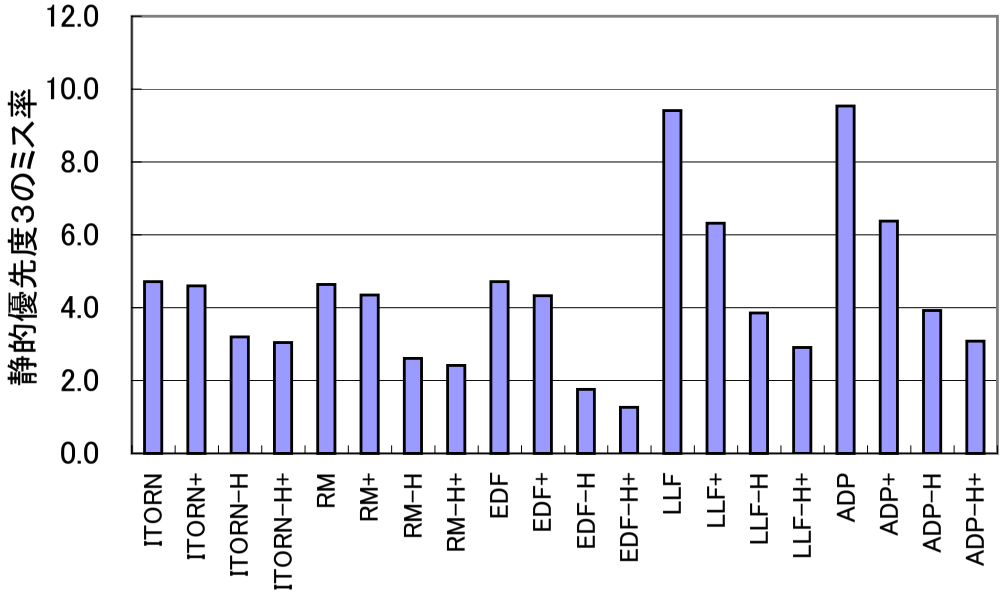


図 4.5: 静的優先度 3 のタスクにおける平均ミス率.

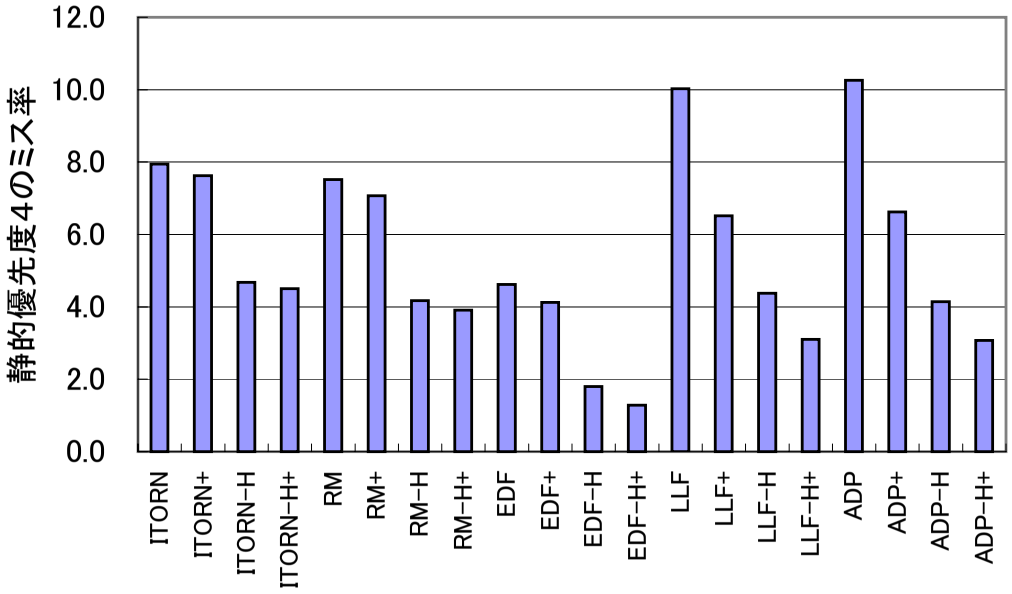


図 4.6: 静的優先度４のタスクにおける平均ミス率.

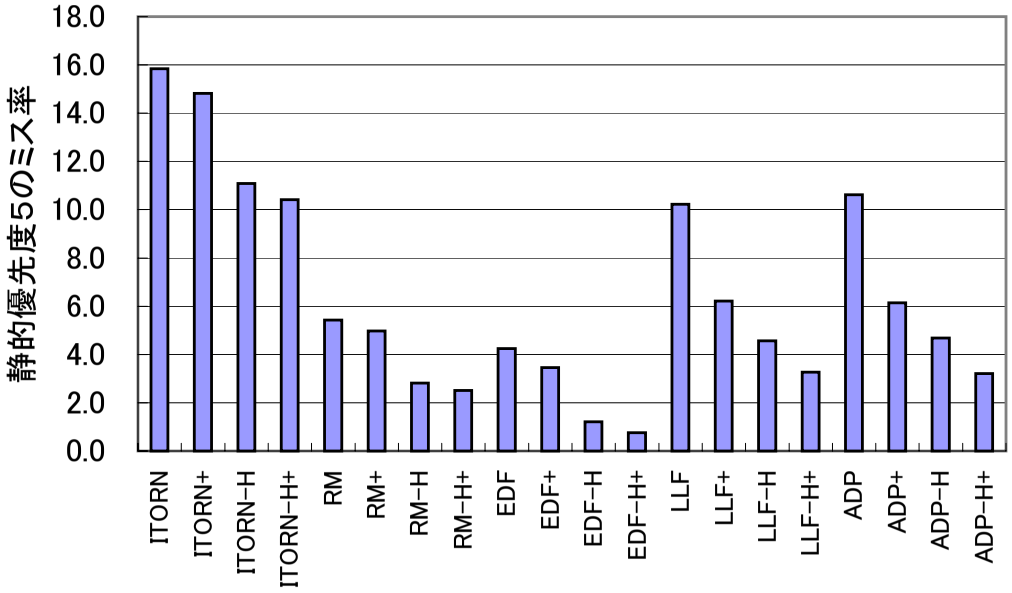


図 4.7: 静的優先度５ののタスクにおける平均ミス率.

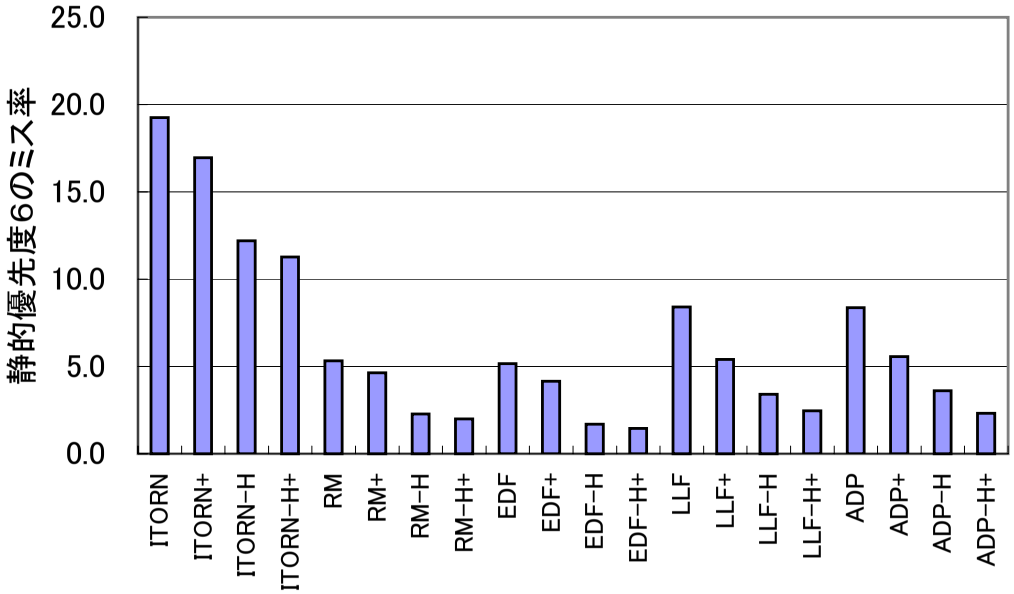


図 4.8: 静的優先度６におけるタスクの平均ミス率.

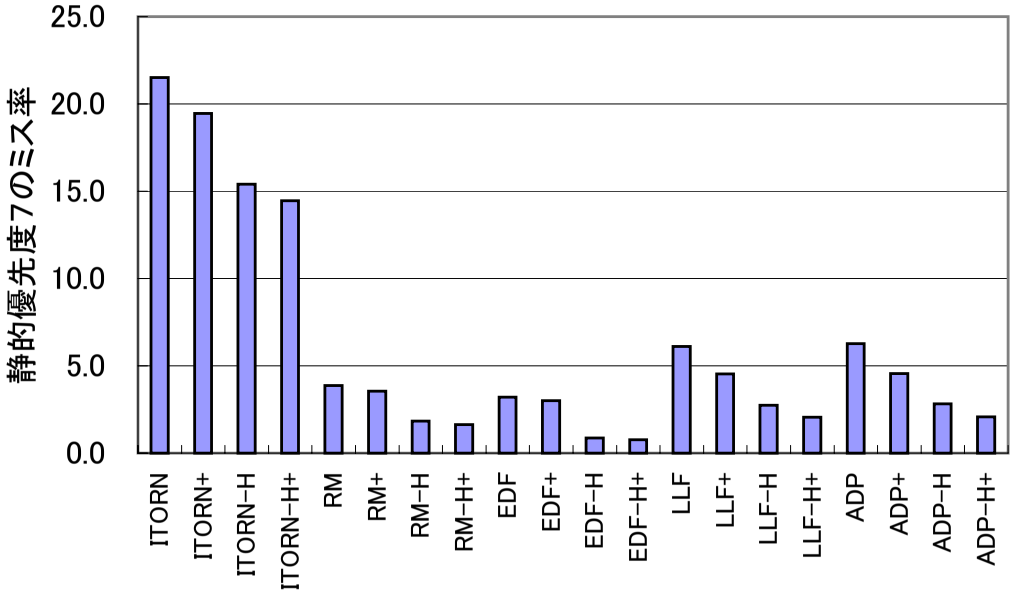


図 4.9: 静的優先度７におけるタスクの平均ミス率.

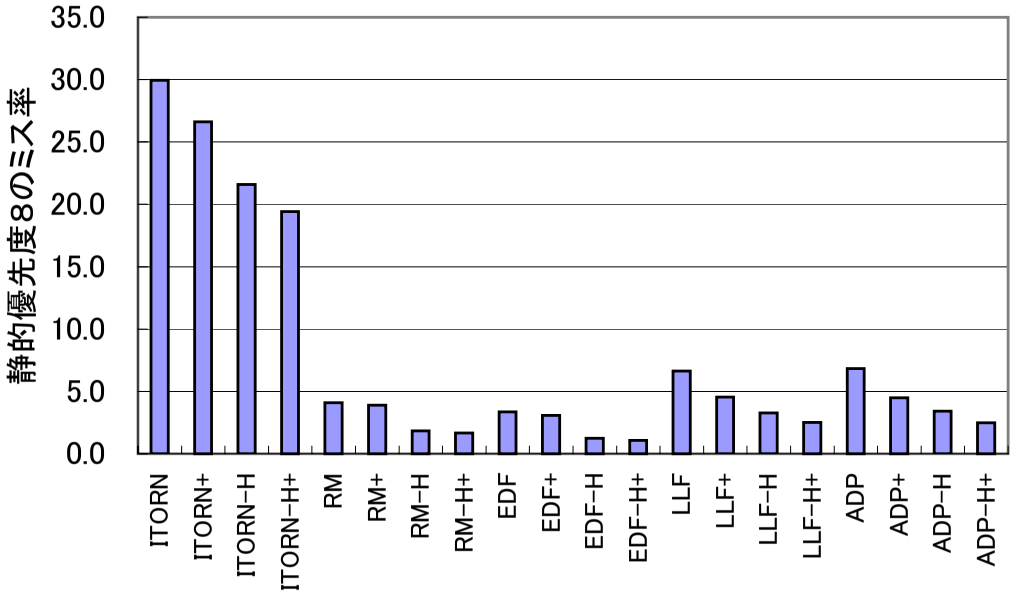


図 4.10: 静的優先度 8 におけるタスクの平均ミス率.

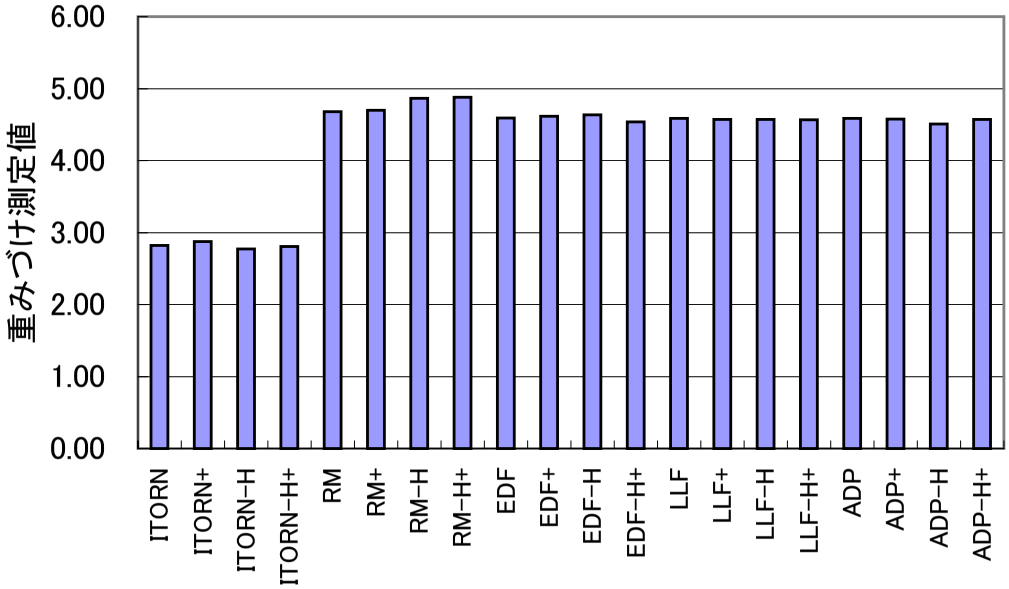


図 4.11: 重みづけによる測定値.

第5章　関連研究

本章ではハードウェアスケジューラに関する研究を二つ報告する．

[5] は周期タスクをハードリアルタイムタスク，非周期タスクをソフトリアルタイムタスクと仮定し，周期タスクのみではデッドラインミスが発生しないシステムにおいて，ハードウェアスケジューラが非周期タスクの起動要求時に実行可能と判断した場合のみ，非周期タスクを起動するシステムである．複雑なシステムとなっているため，ハードウェア量が多く，また，特定のアルゴリズム専用であるため，汎用性がない．

[6] は代表的なリアルタイム OS である μ ITORN を VLSI 化した研究である．スケジューリングアルゴリズムだけではなく，オペレーティングシステム自体を VLSI 化しているため，ハードウェア量が多く，汎用性がない．

本研究では小規模かつ汎用性があるハードウェアを提案し，デッドラインミスを削減している．

第6章　おわりに

6.1　まとめ

ここでは，本研究のまとめおよび考察をおこなう．

- ハードウェアスケジューラの提案をおこない，デッドラインオーバーとなるタスクの削減をおこなった．
- 提案したハードウェア量について評価した．
- 提案したハードウェアを用いたシミュレーションをおこない，様々なアルゴリズムで評価した．

6.2　今後の課題

今後の課題としては，シミュレータの高精度化が挙げられる．今回作成したシミュレータはキャッシュを導入していなかったため，キャッシュミスペナルティを考慮していない．また，コンテキスト切り替えにおけるオーバヘッドに概算値を使用しているため，厳密なタスク切り替えをシミュレートしていない．また，タスク定義に関して，より厳密に現実に使用されているタスクで評価する必要がある．

第7章　謝辞

本研究を遂行にするにあたり，終始熱心にかつ懇切丁寧にご指導を賜りました，田中 清史助教授に心から深く感謝するとともに，ここに御礼申し上げます.

貴重な御助言をしていただきました日比野 靖 教授，堀口 進 教授に深く感謝致します.

そして，本講座におきまして研究のみならず日常生活においてもご指導，御助言を頂きました同講座生の皆様に心から御礼申し上げます.

参考文献

[1] http://www.assoc.tron.org

[2] C.L.Liu and J.W.Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," Jounal of the ACM, 20 (1), 1973

[3] A.K.Mok, "Foundamental Design Problems of Distributed System for the Hard-Real-Time Environment," ph.D.Dissertation, MIT, 1983

[4] 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, 電子情報通信学会技術研究報告 CPSY, Vol.102, NO478, pp.127-132,2002

[5] Sergio Saez, Joan Vila, Alfons Crespo, A Hardware Scheduler for Complex Real-Time Systems, ISIE'99-Bled, Slovenia

[6] 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