社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－SLDM－139 (10)
2009－ＥＭＢ－ 12 (10)
2009／3／6

Responsive Multithreaded Processor における スレッドスケジューリング機構の設計と実装

梅尾 寛之† 水頭 一壽†† 武田 瑛†† 加藤 真平†† 山崎 信行††

† 慶應義塾大学理工学部情報工学科
†† 慶應義塾大学大学院理工学研究科開放環境科学専攻
〒 223–8522 横浜市港北区日吉 3–14–1
E-mail: †{ume,suito,takeda,shinpei,yamasaki}@ny.ics.keio.ac.jp

あらまし リアルタイム処理用プロセッサ Responsive Multithreaded Processor は，スレッド数が 8 スレッド以内であればコンテキストスイッチを行わずに優先度順に同時実行可能な RMT 実行機構を持つ．しかしながら，9 スレッド以上を実行する場合，ソフトウェアスケジューラによってコンテキストスイッチを行わなければならない．また周期タスクのリリースの為にはソフトウェアスケジューラを定期的に呼び出し，リリース時間をチェックしなければならない．本論文では，RMT Processor を対象としたハードウェアによるスレッドスケジューリング機構の設計と実装について述べる．本スレッドスケジューリング機構では，RMT Processor のプロセッシングコアである RMT PU が全スレッドの周期を保持し，周期スレッドをハードウェアで起床させる．更に，コンテキストキャッシュ内のスレッドと実行スレッドを比較し，ハードウェアでコンテキストスイッチを行う．本スレッドスケジューリング機構によってソフトウェアによるスケジューリングを不要とし，スケジューリングオーバヘッドを大幅に削減する．
キーワード RMT Processor，スケジューリングオーバヘッド，スレッドスケジューリング

Design and Implementation of the Thread Scheduling Scheme for Responsive Multithreaded Processor

Hiroyuki UMEO†, Kazutoshi SUITO††, Akira TAKEDA††, Shinpei KATO††, and Nobuyuki YAMASAKI††

† Department of Information and Computer Science, Faculty of Science and Technology, Keio University
†† Department of Computer Science, Graduate School of Science and Technology, Keio University
3–14–1 Hiyoshi, Kouhoku-ku, Yokohama, Kanagawa 223–8522 Japan
E-mail: †{ume,suito,takeda,shinpei,yamasaki}@ny.ics.keio.ac.jp

Abstract Responsive Multithreaded Processor for real-time processing can execute eight threads simultaneously in priority order without context switching. When over nine threads are executed, context switching is required. A real-time scheduler should be called periodically and release times of all tasks are checked. This paper proposes thread scheduling scheme for RMT Processor. RMT PU, which is processing core of RMT processor, holds the periods of all threads and starts threads by hardware without periodic calls of the scheduler. In addition, threads in context cache are compared with threads in execution, and context switching will be realized by hardware. Our thread scheduling scheme reduces scheduling overheads so that traditional software scheduling can be unnecessary.
Key words RMT Processor, Scheduling Overhead, Thread Scheduling

1. はじめに

リアルタイムシステムでは，スケジューラが各タスクの実行周期やデッドラインといった時間制約を基に優先度を割り当て，実行順序を決定する．スケジューラが優先度に従ってタスクを切り替えながら実行することにより，各タスクの時間制約を守る．実行するタスクを切り替える場合，コンテキストスイッチを行う必要がある．つまり，現在実行しているタスクのコンテ

キストをメモリに退避し，次に実行するタスクのコンテキストをプロセッサ内に復帰させる．ソフトウェアによるコンテキストスイッチは大量のメモリアクセスが発生するため，オーバーヘッドが大きい．

より高度なリアルタイムシステムを構築する場合，システム内で実行する処理の数が増加することが考えられる．また，より細かい制御を行うために，より短かい周期で処理を行う必要性が出てくると考えられる．例えばロボットのモータ制御では，その周期は $100\mu sec$〜$10msec$ と非常に短くなっている [1]．このような場合，タスクの実行時間に対するスケジューリングオーバヘッドの割合はさらに増大する．また，スケジューラの実行によるハードウェアリソースの利用は別のタスクの処理時間にも大きな影響を与える．このため，今後はソフトウェアだけでなくハードウェアレベルからリアルタイム処理を行う機構が必要であると考えられる．

Responsive Multithreaded Processor [2] は，リアルタイム処理用プロセッサとして設計，実装が行われている．RMT Processor は，Simultaneous Multithreading(SMT) [3] アーキテクチャに優先度制御を取り入れることにより，コンテキストスイッチを行わずに複数のスレッドを優先度順に同時実行する機能を持つ．しかし，RMT Processor が持つハードウェアコンテキスト数以上のスレッドを実行する場合は従来と同様にソフトウェアによるコンテキストスイッチが発生してしまう．

本論文では，ハードウェアコンテキストの数以上のスレッドのスケジューリングをハードウェアで行う機構を提案する．

2. Responsive Multithreaded Processor

RMT Processor はプロセッシングユニットである RMT PU，リアルタイム通信用規格 Responsive Link [4]，DDR SDRAM I/F，DMA コントローラ，PCI64 I/F，IEEE 1394 I/F，シリアル I/F，外部 I/F といったコンピュータ用 I/O，PWM ジェネレータ，パルスカウンタといった制御用 I/O を 1 チップに集積した System-on-a-Chip（SoC）である．

2.1 RMT 実行

RMT Processor は SMT 実行をベースとし，それに優先度を導入した RMT 実行機構を用いている．従来のソフトウェアによるコンテキストスイッチを RMT 実行に置き換えることにより，コンテキストスイッチを行わずに優先度の高いスレッドから順に同時実行することが可能である．また複数スレッドを同時に実行することで，プロセッサ全体の性能をスーパースカラと比較して大幅に高めることができ，一方で単一スレッドのみを動作させた場合でも，スーパースカラと同等の性能を発揮できる．図 1 に RMT PU によるリアルタイム処理のシーケンスを示す．優先度の高い Task 0 ～ Task 2 の実行は，その他の低優先度スレッドの実行に影響を受けておらず，優先度の高いスレッドの実行効率の変動及び低下を抑制できていることが分かる．RMT Processor はアクティブスレッドの優先度を RM スケジューリングアルゴリズム [5] を効果的に適用するために必要とされる 256 段階で設定可能である．

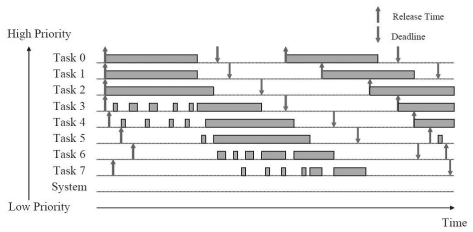

図 1　RMT Processor による RMT 実行の様子

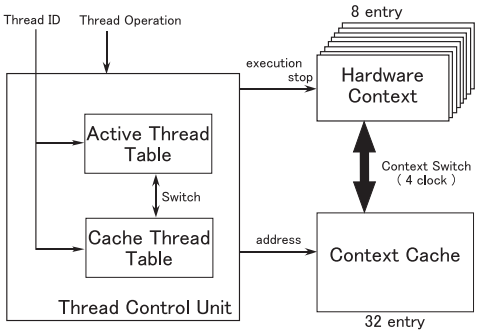

図 2　コンテキストキャッシュを利用したコンテキストスイッチの概略図

2.2 コンテキストキャッシュ

Komodo Microcontroller [6] は RMT Processor と同様にマルチスレッドアーキテクチャを利用し，プロセッサ内に複数のスレッドを保持することで，ソフトウェアによるコンテキストスイッチを行わずにスレッドを切り替えることができる．しかし，ハードウェアコンテキスト数以上のスレッドを実行する場合，ソフトウェアによるコンテキストスイッチによってスレッドを入れ替える必要がある．RMT Processor では，コンテキストスイッチのオーバーヘッドを削減するために，オンチップで 32 スレッドのコンテキスト情報を保持できる専用キャッシュを搭載している．コンテキストスイッチで入れ替えるデータは汎用レジスタ，浮動小数点レジスタ及び制御レジスタであり，コンテキストキャッシュとハードウェアコンテキストとの間で専用バスを介してデータ転送を行う．以降，ハードウェアコンテキストが割り当てられているスレッドをアクティブスレッド，コンテキストキャッシュに格納されているスレッドをキャッシュスレッドと呼ぶ．RMT PU で保持されているスレッドは Thread Control Unit 内で管理されている．また，スレッドを識別するため各スレッドに Thread ID (TID) が付与されている．アクティブスレッド、コンテキストスイッチ間のコンテキストスイッチは，RMT Processor 専用のスレッド制御命令及び入れ替えるスレッドの TID を Thread Control Unit に伝えることで行われる (図 2)．コンテキストスイッチは 4 サイクルで完了する．

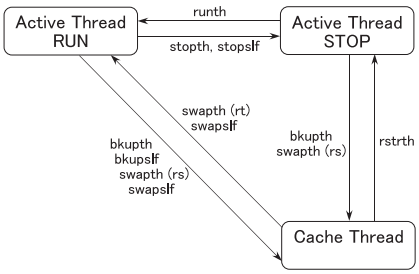

図 3　RMT Processor におけるスレッドの状態遷移図

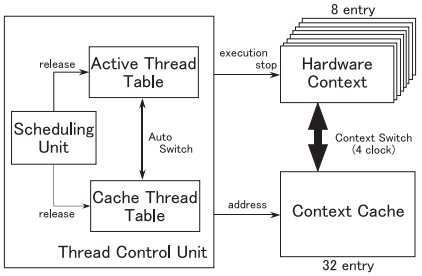

図 4　スレッドスケジューリング機構の概要図

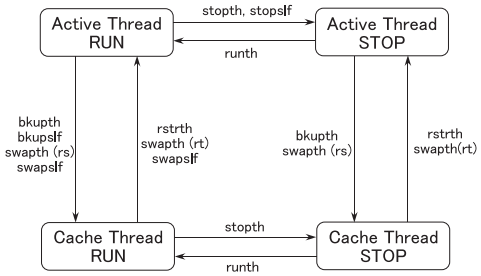

図 5　スレッドの状態遷移図

2.3　スレッドの制御

RMT Processor はスレッドの状態を図 3 のように定義している．プロセッサが実行するスレッドはレジスタファイルやプログラムカウンタなどの資源が確保されておりかつ実行状態にあるスレッド（図 3 の Active Thread RUN）のみである．スレッドの状態は図 2 の Active Thread Table，Cache Thread Table と対応しており，図 3 に示されているスレッド制御命令によって状態遷移が行われる．スレッド制御命令について以下に示す．

runth　指定したスレッドを Active Thread RUN 状態に遷移させる

stopth　指定したスレッドを Active Thread STOP 状態に遷移させる

stopslf　自身を Active Thread STOP 状態に遷移させる

bkupth　指定したアクティブスレッドをコンテキストキャッシュへ退避する

bkupslf　自身をコンテキストキャッシュへ退避する

rstrth　指定したキャッシュスレッドを復帰させる

swapth　指定したアクティブスレッドとキャッシュスレッドをスワップする

swapslf　自身とキャッシュスレッドをスワップする

chgpr　指定したスレッドの優先度を設定する

chgpr 命令に関してはスレッドの状態遷移は行わない．RMT Processor は chgpr 命令で設定された優先度を基に RMT 実行を行う．

2.4　課　　題

RMT Processor は 8 組のハードウェアコンテキストを持っているため，最大 8 スレッドまで並列実行が可能である．しかしながら，9 スレッド以上を同時に実行したい場合にはソフトウェアスケジューラを用いてコンテキストスイッチを行う必要がある．解決策の一つとしてハードウェアコンテキスト数を増やすという方法が存在するが，ハードウェアのコストが高くなってしまう．またこれ以上ハードウェアコンテキスト数を増やすとプロセッサが複雑になってしまい，コストに見合った性能の向上が望めないため，この策は得策ではない．

9 スレッド以上を同時に実行する場合，周期スレッドの管理をするために，ソフトウェアスケジューラを周期的に呼び出す必要がある．ソフトウェアスケジューラを呼び出す周期が短い場合，スケジューリング処理によるプロセッサの占有率が高くなってしまい，実際に処理すべきタスクの実行に影響を及ぼしてしまう可能性があると考えられる．RMT Processor はコンテキストキャッシュを利用することでコンテキストスイッチ自体は 4 クロックで行うことができる．一方，スケジューリング処理全体では高速化を図れていない．

3.　設計及び実装

本研究では，RMT Processor の Thread Control Unit 内にスレッドスケジューリング機構を設計，実装する．

図 4 に提案するスレッドスケジューリング機構の概要を示す．Scheduling Unit は RMT PU に保持されている全周期スレッドの周期情報を持ち，周期情報に従って周期スレッドのリリース時間が来たことを release 信号によってスレッドテーブルに伝える．さらに，アクティブスレッドとキャッシュスレッドを比較し，ハードウェアでコンテキストスイッチを行う．

3.1　スレッドの状態の管理

本研究では，図 3 の Cache Thread 状態を更に分割し，図 5 に示す状態遷移を行うようにする．このように状態を定義することで，プロセッサはどのスレッドが実行可能状態であるかをハードウェアのみで把握することが可能となる．周期スレッドの場合，リリース時間が来た時点で Active Thread RUN 状態もしくは Cache Thread RUN 状態に遷移する．その周期で行うべき処理が終了するとスレッドは stopslf 命令を実行し，Active Thread STOP 状態に遷移する．

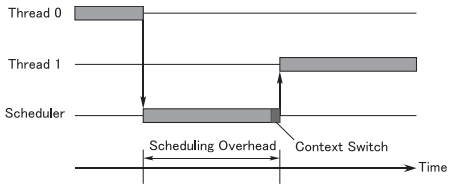

図 6　スケジューリングオーバヘッド

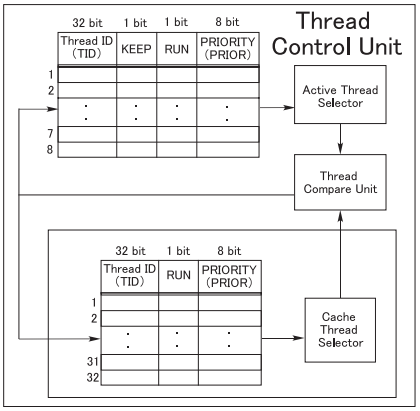

図 7　ハードウェアコンテキストスイッチ機構

3.2　スレッド選択機構

コンテキストスイッチを行う場合は，入れ替えるスレッドを指定する必要がある．図 6 は Thread 0 の実行が終了し，Thread 1 に実行が移るときの概略図である．コンテキストキャッシュを用いることで，コンテキストスイッチ自体は非常に短い時間で完了する．しかしながら，スケジューリングをソフトウェアによって行う場合，コンテキストスイッチにかかる時間と比較して大きなオーバヘッドが伴う．スケジューリングオーバヘッドには，スケジューラの起動，タスクの管理のための処理の他に，次に実行するスレッドの選択が含まれる．本研究では入れ替えるスレッドをソフトウェアスケジューラを用いずにハードウェアによって選択することで，スケジューラによるオーバヘッドの削減を行う．

RMT Processor は，アクティブスレッドとキャッシュスレッドを，Thread Control Unit 内のアクティブスレッドテーブル，キャッシュスレッドテーブルによって管理している．RMT Processor はこれらのスレッドテーブルに保持されている TID を入れ替え，入れ替えた情報をハードウェアコンテキスト及びコンテキストキャッシュへ伝えることで，コンテキストスイッチを行っている (図 7)．

3.2.1　Active Thread Selector

RMT Processor は Thread Control Unit 内に TID および図 8 のようなスレッドテーブルを保持している．アクティブスレッドテーブルのエントリはアクティブスレッドの数，すなわち 8 エントリ存在する．アクティブスレッドテーブルの各ビットは以下の役割を持つ．

図 8　アクティブスレッドテーブル

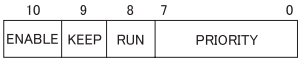

図 9　比較用ビット列

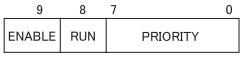

図 10　キャッシュスレッドテーブル

ENABLE　該当エントリにスレッドが割り当てられているかを示す
STATE　アクティブスレッドの状態を示す
KEEP　アクティブスレッドに常駐させたいスレッドであることを示す
PRIORITY　スレッドの優先度 (256 レベル)

本研究では図 8 のスレッドテーブルをほぼそのまま利用し，スワップ候補を選択する．具体的には図 9 のような比較用ビット列をスレッドごとに生成して数値比較を行い，最も低い数値を持つスレッドをスワップアウトされる候補として選択する．選択された TID 及び優先度を Thread Compare Unit に送る．

8 スレッド全てが使用されていない場合は，エントリが空いていることを Thread Compare Unit に送る．また，KEEP ビットが立っているスレッドがスワップアウトされる候補として選択された場合はコンテキストスイッチを行わない．

3.2.2　Cache Thread Selector

従来，キャッシュスレッドに関する情報はスレッド命令 (rstrth, swapth など) で利用するための TID だけが保持されていた．本研究では，ハードウェアによるスレッドの選択を行うために，コンテキストキャッシュ制御ユニットに新しくキャッシュスレッドテーブル (図 10) を作成する．テーブルのエントリ数は，コンテキストキャッシュのエントリ数と同じ 32 エントリである．キャッシュスレッドテーブルの各ビットは以下の役割を持つ．

ENABLE　該当エントリにスレッドが割り当てられているかを示す
RUN　スレッドが実行可能状態であるかを示す
PRIORITY　スレッドの優先度 (256 レベル)

キャッシュスレッドテーブルの情報を基に，実行可能であり，かつ優先度の最も高いスレッドの TID とその優先度をスワップインする候補として Thread Compare Unit に送る．実行可能状態のスレッドが存在しない場合や，コンテキストキャッシュにスレッドが存在しない場合は，スワップインする候補がないことを Thread Compare Unit に送る．

3.2.3　Thread Compare Unit

Thread Compare Unit では，それぞれ送られてきた情報を元に，最終的な動作を決定する．Thread Compare Unit がと

る動作としては以下の三種類である．

NOP　実行可能なキャッシュスレッドが存在しない場合及び，存在しても優先度がアクティブスレッドより低い場合は何も行わない

RESTORE　アクティブスレッドが 8 個全て割り当てられておらず，かつ実行可能なキャッシュスレッドが存在する場合は，無条件でスレッドの復帰を行う

SWAP　選択されたアクティブスレッドが停止状態かつ実行可能なキャッシュスレッドが存在する場合及び，実行可能なキャッシュスレッドの優先度がアクティブスレッドより高い場合はスレッドの入れ替えを行う

決定された動作を基に，アクティブスレッドテーブルとキャッシュスレッドテーブルを書き換える．書き換えられたスレッドテーブルの情報は次のクロックでコンテキストキャッシュに渡り，コンテキストスイッチが行われる．

ソフトウェアによるスレッド命令が Thread Control Unit 内に存在する場合は，Thread Compare Unit のスレッド選択結果は破棄される．これはソフトウェアによるスレッド命令はハードウェアによるスレッドの選択結果よりも優先されるべきと考えたためである．また，他のスレッドがコンテキストスイッチを行っている状態でのスレッド選択結果も破棄される．これはコンテキストキャッシュが複数スレッドを同時に入れ替えることができないためである．

3.3　周期スレッドの制御

Saez ら [7] は，スレッドの周期情報を持つテーブル，スレッドの起動を行うためのコントローラ及びスレッドの状態を管理するためのキューをハードウェアで実装することによりスレッドのハードウェアによるスケジューリングを行う手法を提案した．本研究ではこれとほぼ同様のテーブルを用意し，周期スレッドの制御を行う．

3.3.1　Scheduling Unit

Scheduling Unit は，ハードウェアによってあらかじめ周期の最小単位 (Sampling Period) を定義しておき，定義した Sampling Period ごとにカウントアップするカウンタと，周期スレッド管理用のテーブルを持つ．テーブルのエントリ数は，ハードウェアコンテキストのエントリ数とコンテキストキャッシュのエントリ数の合計である 40 エントリとする．テーブルでは以下の情報を管理する．

Thread ID　スレッド生成時にテーブルに TID が登録される．リリース時間に到達したスレッドの TID を Thread Control Unit に送り，一致するスレッドを RUN 状態に遷移させる

Periodic Mode　スレッドが周期タスクかどうかを示す

Timer　周期実行を行うかどうかを示す

Period　指定した周期が格納される

Release Time　リリース時間を示す．リリース時間に到達したとき，この値を次のリリース時間に更新する

Release Bit　リリース時間に到達したことを示すビットであり，このビットが 1 となったエントリの TID が Thread Control Unit に送られ，実際にスレッドが RUN 状態になるとこのビットはクリアされる．

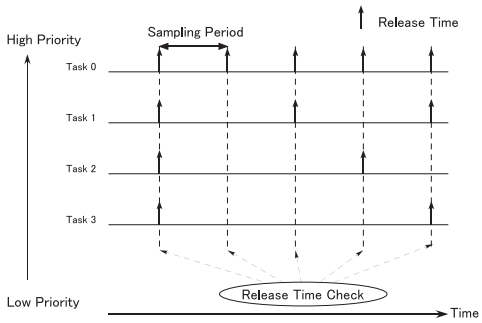

図 11　周期スレッドの設定及び起動

タスクの周期は Sampling Period の倍数で指定し，Sampling Period 毎に各スレッドのリリース時間を調べることで周期の制御を行う (図 11)．RUN 状態に遷移したスレッドがコンテキストキャッシュ内に保持されている（ハードウェアコンテキストを割り当てられていない）場合は，先に述べたスレッド選択機構により，優先度に従ってコンテキストスイッチが行われる．起動したスレッドの優先度が RMT PU で実行しているスレッドの優先度よりも高い場合，スレッド選択機構により起動した直後にコンテキストスイッチが行われ，起動したスレッドにハードウェアコンテキストが割り当てられる．起動したスレッドの優先度が RMT PU で実行しているスレッドの優先度よりも低い場合，優先度の高いスレッドの実行が終了した後にコンテキストスイッチが行われる．実行すべき処理が完了したスレッドは最後に stopslf 命令を実行することによって Active Thread RUN 状態から Active Thread STOP 状態に遷移し，次の実行周期を待つ．

3.3.2　スレッド制御命令の追加

周期スレッドの制御のために，新しく以下のスレッド制御命令を追加する．

chgpd　指定したスレッドの周期を設定する．周期は上で述べた Sampling Period の倍数で指定する．

tmon　指定したスレッドの周期実行を有効にする

tmoff　指定したスレッドの周期実行を無効にする

これらのスレッド制御命令は Scheduling Unit が持つテーブルに情報を書き込む．chgpd 命令は Period に指定した周期を書き込み，Periodic Mode ビットを 1 にセットする．tmon, tmoff 命令は Timer ビットのセット及びリセットを行う．

4.　評　　価

設計したスレッドスケジューリング機構を Verilog-HDL を用いて RMT Processor に実装した．本節では NC-Verilog を用いた RTL シミュレーションにより評価を行い，提案手法であるスレッドスケジューリング機構が，ソフトウェアによるスケジューリングと比べてオーバヘッドを削減できることを示す．

表 1　タスク切り替えによるスケジューリングオーバヘッド

スケジューリング手法	オーバヘッド
ソフトウェアスケジューラ	3178 クロック
スレッドスケジューリング機構	10 クロック

表 2　ハードウェア面積の評価

ユニット	面積
従来 Thread Control Unit	0.33 mm^2
スレッドスケジューリング機構実装	0.64 mm^2

4.1 評価環境

実装に用いた RMT Processor の各種パラメータは文献 [2] と同様である．動作周波数は 100MHz とした．評価では，スレッドスケジューリング機構と現在研究室で開発が進められている RMT Processor 用の OS に実装されているスケジューラとの比較を行った．

4.2 スケジューリングオーバヘッドの評価

提案したスレッドスケジューリング機構とソフトウェアスケジューラとのスケジューリングオーバヘッドに関する評価結果を表 1 に示す．スケジューリングオーバヘッドとして，高優先度タスクがその周期の処理を終了してから，実行可能状態の次のタスクに切り替わり，実行が開始するまでの時間を計測した．この間に行われる処理は具体的には以下のように分けることができる．

- タスクの終了時に必要な処理
- スケジューラの呼び出し
- 次に実行するスレッドの選択
- タスクの開始時に必要な処理
- コンテキストスイッチ

比較の結果，タスク切り替えによるスケジューリングオーバヘッドは約 0.31%まで短縮することができた．

ソフトウェアスケジューラを用いる場合，まずスケジューラの呼び出しによるオーバヘッドが存在する．さらに，コンテキストスイッチを行う前にスレッドを 1 つずつ調べ，次に起動するタスクを選択する必要がある．そのため，ソフトウェアによるスレッドの入れ替えはスレッド数の増加に伴いオーバヘッドも増加していく．一方，スレッドスケジューリング機構を用いる場合，スレッドの選択はハードウェアで 1 クロックで行われるため，必要クロック数はスレッド数に関わらず一定である．

表 1 の結果を動作周波数 100MHz として実時間に変換した場合，ソフトウェアスケジューラによるオーバヘッドはおよそ $30\mu s$ となる．周期が $100\mu s$ のタスクの場合，スケジューラによる処理は周期全体の約 3 割を占めることになる．一方で，スレッドスケジューリング機構を用いた場合，オーバヘッドはおよそ $100ns$ となる．これはタスクの周期全体の 1 割以下であり，スレッドスケジューリング機構が有効であることを示している．

4.3 ハードウェア面積の評価

スレッドスケジューリング機構を実装した Thread Control Unit を，Synopsys 社の Design Compiler 2008.12 を使用して論理合成し，ハードウェアの面積についての評価を行った．テクノロジライブラリはオリジナルの RMT Processor に用いられている TSMC 社の $130nm$ のプロセスを使用した．論理合成の結果を表 2 に示す．

表 2 より，従来の Thread Control Unit の約 2 倍の面積となった．スレッドスケジューリング機構を実装していない状態でのチップ全体の面積は $47.54mm^2$ であったため，全体に対してはおよそ 1%のハードウェア面積増加となった．RMT Processor のダイサイズは $10mm$ 角であるため，スレッドスケジューリング機構は十分に実装可能といえる．

5. ま と め

本論文では，ソフトウェアスケジューラによって行われていたスレッドの起動およびコンテキストスイッチをハードウェアによって行うスレッドスケジューリング機構を提案した．スレッドスケジューリング機構を用いることにより，ソフトウェアスケジューラによるスレッドのリリース時間の周期的な確認及びコンテキストスイッチを不要とし，スケジューリングオーバヘッドを約 0.31% まで削減することができた．スレッドスケジューリング機構の実装による Thread Control Unit の面積の割合は RMT Processor の約 1%であり，十分実装可能であることを示した．

謝辞　本研究の一部は科学技術振興機構 CREST の支援によるものであることを記し，謝意を表す．また，本研究の一部は文部科学省グローバル COE プログラム「環境共生・安全システムデザインの先導拠点」によるものであることを記し，謝意を表す．

文　献

[1] T. Matsui, H. Hirukawa, Y. Ishikawa, N. Yamasaki, S. Kagami, F. Kanehiro, H. Saito and T. Inamura: “Distributed real-time proceeding for humanoid robots”, 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 205–210 (2005).

[2] N. Yamasaki: “Responsive multithreaded processor for distributed real-time systems”, Journal of Robotics and Mechatronics, **17**, 2, pp. 130–141 (2005).

[3] D. M. Tullsen, S. J. Eggers and H. M. Levy: “Simultaneous Multithreading: Maximizing On-Chip Parallelism”, Proc. of the 22nd Annual International Symposium on Computer Architecture, pp. 392–403 (1995).

[4] N. Yamasaki: “Design and implementation of responsive processor for parallel/distributed control and its development environments”, Journal of Robotics and Mechatronics, pp. 125–133 (2001).

[5] J. P. Lehoczky, L. Sha and Y. Ding: “The rate monotonic scheduling algorithm: exact characterization and average case behavio”, Proc. IEEE 10th Real-Time Systems Symp, pp. 166–171 (1989).

[6] U. Brinkschulte, C. Karowski, J. Kreuzinger and T. Ungerer: “A multithreaded java microcontroller for thread-oriented real-time event-handling”, International Conference on Parallel Architectures and Compilation Techniques, pp. 34–39 (1999).

[7] S. Saez, J. Vila, A. Crespo and A. Garcia: “A hardware scheduler for complex real-time systems”, IEEE International Symposium on Industrial Electronics, **1**, pp. 43–48 (1999).