Vol.2010-SLDM-144 No.61
Vol.2010-EMB-16 No.61
Vol.2010-MBL-53 No.61
Vol.2010-UBI-25 No.61
2010/3/28

中断可能なキューイングスピンロックのハードウェア実装と評価

一場利幸[†1]　松原豊[†2]
本田晋也[†2]　高田広章[†1,†2]

マルチコアプロセッサシステムでは，コア間で共有するデータの排他制御が必須であり，スピンロックと呼ばれる排他制御方式が広く用いられている．リアルタイムシステムにおいては，最悪実行時間が予測可能であることと，割込みに高速に応答することが求められており，スピンロックも例外ではない．これらの要求を満たす，中断可能なキューイングスピンロックアルゴリズムが提案されているが，CAS 命令や LL/SC 命令を前提としているため，これらの命令を備えていないプロセッサでは実装することができない．また，ソフトウェアによる実装のためロック取得の実行オーバヘッドが大きいという問題もある．本研究では，CAS 命令や LL/SC 命令を用いない排他制御を実現するため，ハードウェアサポートによる中断可能なキューイングスピンロックアルゴリズムを提案する．提案手法と従来のスピンロックアルゴリズムとの比較評価により，要求される性質を満たしつつ，実行オーバヘッドを低減できることを示す．

Hardware Implementation and Evaluation of Queueing Spin Lock with Preemption

Toshiyuki Ichiba,[†1] Yutaka Matsubara,[†2]
Shinya Honda[†2] and Hiroaki Takada [†1,†2]

An exclusive access method for sharing data among cores is needed in multi-core processor systems. Spin lock algorithms are often used.In real-time embedded systems, a spin lock algorithm need to satisfy both of predictable worst-case execution time and fast interrupt response. Queueing spin lock algorithms that satisfy both requirements have been proposed. However, these algorithms cannot be implemented on processors without CAS or LL/SC instructions. Moreover, in the algorithms, execution overhead of acquiring and releasing a lock is large because of software implementation. This paper proposes a new queueing spin lock algorithm with hardware support. The result of evaluation shows the proposed method satisfies real-time requirements and generates low overhead compared by queueing spin lock algorithms implemented in software.

1. はじめに

近年，大規模化・複雑化が進む組込みシステムにおいても，マルチコアシステムの重要性が増している．その背景には，消費電力の増大を抑えつつ処理性能の向上を図るためには，クロック周波数を上げるよりも，プロセッサ数を増やしたほうが有利であるという状況がある．特に，複数のプロセッサを 1 つの LSI 上に集積したマルチコアプロセッサは，処理性能面からも消費電力面からも利点が大きく，広範な組込みシステムへの適用が期待される．

マルチコアプロセッサシステムでは，システム全体で共有するデータへのアクセスには，排他制御が必須である．クリティカルセクションが短い排他制御には，ロックを取得できたかどうかを繰返しチェックするスピンロックと呼ばれる方式が用いられている．スピンロックは TAS（Test and Set）命令を用いた実装が一般的であるが，ロック取得で競合が発生した場合，ロックを取得する順序がランダムで決まるためコアが 3 つ以上のシステムではロック取得までの最悪実行時間の上限が定まらないという問題がある．

ロックを取得するまでの時間に上限をもたせるために，ロック待ちの状態をキューで管理するキューイングスピンロックのソフトウェア実装が提案されている．代表的なキューイングスピンロックアルゴリズムである MCS ロック[1] は，割込み応答性が悪くリアルタイムシステム向けではない．MCS ロックをベースに割込み応答性を向上させたキューイングスピンロックアルゴリズムが提案されている[2)3)]．

これらのキューイングスピンロックアルゴリズムは，CAS（Compare and Swap）命令もしくは LL/SC（Load Linked / Store Conditional）命令の存在を前提としている．これらの命令を実装するためには，バスを複雑化したりコヒーレントキャッシュを使う必要がある．しかしながら，組込みシステムでは，消費電力やコストの観点からこれらのハードウェアを用いるのが難しいため，CAS 命令や LL/SC 命令をサポートしていないプロセッサを用いる場合がある．このようなプロセッサでは，ペリフェラルの追加によって排他制御を実

†1 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University
†2 名古屋大学大学院情報科学研究科附属組込みシステム研究センター
Center for Embedded Computing Systems, Nagoya University

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.61
Vol.2010-EMB-16 No.61
Vol.2010-MBL-53 No.61
Vol.2010-UBI-25 No.61
2010/3/28

現する方法が必要となる．また，ソフトウェア実装によるキューイングスピンロックアルゴリズムは実行オーバヘッドが大きいという問題があるため，ハードウェアサポートによる実行オーバヘッドの低減も期待できる．

本研究では，CAS 命令や LL/SC 命令を用いない排他制御を実現するため，ハードウェアサポートによるキューイングスピンロックの実現方法を提案する．提案手法と従来のスピンロックアルゴリズムとの比較評価により，実行オーバヘッドを低減できることを示す．

本論文の構成は次の通りである．2 章でキューイングスピンロックのソフトウェアアルゴリズムについて述べる．3 章では，提案するハードウェアのアルゴリズムについて述べ，4 章で他のアルゴリズムとの比較評価を行う．5 章でスピンロックの実現に関する研究について述べる，6 章で，まとめと今後の提案ハードウェアの拡張について述べる．

2. キューイングスピンロック

CAS 命令や LL/SC 命令を用いる，ソフトウェア実装のキューイングスピンロックアルゴリズムについて述べる．

2.1 MCS ロックと割込み応答性

MCS ロックは，ロック取得待ちの順序をキューで管理する．ロックを解放する時に次のコアにロックを渡すため，ロック取得の順番は FIFO である．コア 1 の次にコア 2 がロック取得を要求したときのロック待ちキューを**図 1** に示す．MCS ロックでは，ロック取得の順番が決まっているため，取得までの時間に上限をもたせることができる．

ロックを保持している間は，他コアのロック取得の最大実行時間を抑えるために，割込みを禁止する必要がある．ロックを保持すると同時に割込みを禁止するのが理想であるが，そのような命令を備えた一般的なプロセッサはない．このため，どちらかを先に実行する必要がある．ロックを保持している間は割込みを禁止する必要があるため，割込みを禁止してからロックを保持する．MCS ロックでは，割込み処理を考慮していないため，ロック取得を要求するときに割込みを禁止する必要がある．ロック取得待ち時間が割込み禁止になってしまうため，割込み応答性が悪くなる．

MCS ロックは割込み応答性が悪いという問題があるため，割込み応答性を向上させたアルゴリズムが提案されている．このようなアルゴリズムを中断可能なキューイングスピンロックアルゴリズムと呼び，以下に述べる QLPD と QLPR の 2 つが該当する．

2.2 QLPD

文献 2) では，ロック取得待ちの途中で割込みが入った場合にキューから外す方法をアルゴリズム 1 として提案している．このアルゴリズムを本稿では，QLPD（Queueing spin lock with preemption/dequeueing）と呼ぶことにする．

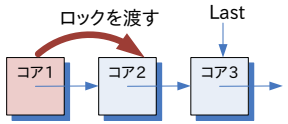

図 1 MCS ロックのロック待ちキュー
Fig. 1 Lock wait queue of MCS lock

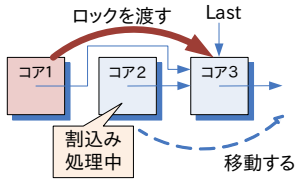

図 2 QLPD のロック待ちキュー
Fig. 2 Lock wait queue of QLPD

ロック取得に失敗したとき，割込み要求が発生しているかをチェックし，割込み要求が発生していたら割込み処理の実行に移り，割込み処理終了後にロック取得待ちに戻る．ここで，割込み処理実行中にロックを取得できる順番になったら，ロックを解放するコアが割込み処理実行中のコアをロック待ちキューから外す．割込み処理実行中にロック待ちキューから外されたコアは，割込み終了後に再度キューに並びなおす．

例を**図 2** に示す．ロックを次のコアに渡そうとしているコア 1 は，コア 2 が割込み処理中のためコア 2 をロック待ちキューから外して，次のコア 3 にロックを渡す．コア 2 は，割込み処理から復帰したときにロック待ちキューから外されたために再度ロック待ちキューに並ぶ．

このアルゴリズムでは，割込み処理の途中でロック取得できる順番になった場合に，ロック待ちキューから外され最後尾に並びなおす．このため，割込み発生頻度が高いとロック取得までの最大待ち時間が長くなるという問題がある．

2.3 QLPR

文献 3) では，待ち途中で割込みが入ってもキューから外さない方法を提案している．このアルゴリズムを本稿では，QLPR（Queueing spin lock with preemption/remaining）と呼ぶことにする．

QLPR は，QLPD と異なりロック取得中に割込みが入った場合も割込み処理中のコアをロック待ちキューから外さないようにすることで，ロック取得までの最大待ち時間を抑える．ロックを解放するコアは，ロック待ちキューから割込み処理を実行していないコアを探し，ロック待ちキューの先頭に移動させてロックを渡す．これにより割込み処理を終了後もロック待ちキューのもとの位置に戻るため，待ち時間の増加を抑えることができる．

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.61
Vol.2010-EMB-16 No.61
Vol.2010-MBL-53 No.61
Vol.2010-UBI-25 No.61
2010/3/28

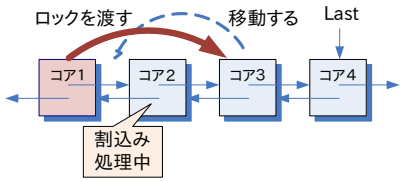

図 3 QLPR のロック待ちキュー
Fig. 3 Lock wait queue of QLPR

例を**図 3** に示す．コア 1 は，コア 2 が割込み処理中のため，コア 3 をロック待ちキューの先頭に移動させてロックを渡す．コア 2 は割込み処理から復帰後，コア 3 からロックを渡される．コア 4 の後に並びなおさないため，QLPD と比較して割込み処理復帰後の待ち時間の増加を抑えることができる．

QLPD と比較して，キューの操作が複雑になり，平均処理時間が長くなるという問題がある．また，ロック待ちキューに並んでいるコアが割込み処理を実行している場合は，そのコアをスキップする処理が必要となる．そのため，割込み処理の実行時間が長い場合はスキップする回数が多くなるため，QLPD と比べてロック取得の実行オーバヘッドが大きくなる．

3. 中断可能なキューイングスピンロックのハードウェア実装

CAS 命令や LL/SC 命令を利用しない，ペリフェラルによる排他制御を実現する手法として，中断可能なキューイングスピンロックのハードウェア実装について述べる．

3.1 ハードウェア実装の要件

ソフトウェア実装によるキューイングスピンロックは，CAS 命令や LL/SC 命令の存在を前提としている．CAS 命令を実装するためにはバスを複雑化したりコヒーレントキャッシュを使う必要がある．しかしながら，組込みシステムでは，消費電力やコストの観点からこれらのハードウェアを用いるのが難しいため，CAS 命令や LL/SC 命令をサポートしていないプロセッサがある．このようなプロセッサでは，ペリフェラルによって排他制御を実現する方法が必要となる．ハードウェア実装によって，実行オーバヘッドを低減できるだけでなく，ヘテロジニアスなマルチコアプロセッサシステムにおいても排他制御を利用することができる．

ペリフェラルによる排他制御を実現するため，中断可能なキューイングスピンロックをハードウェア実装する．ハードウェア実装にあたり，以下の要件が挙げられる．

(1) プロセッサに依存した命令やバスを利用しない
排他制御を実現するにあたり，プロセッサに特別な命令を追加したり，特別なバスの存在を前提とすることはリソース制約の厳しい組込みシステムにおいて大きなデメリットとなる．そのため，プロセッサに依存せず，ペリフェラルの追加によって排他制御を実現することが望ましい．

(2) 特定の割込みアーキテクチャに依存しない
優れた割込み応答性を実現するには，ロック取得待ちの間に割込み処理を実行する方法が必要である．割込みアーキテクチャはプロセッサごとに異なっているため，特定の割込みアーキテクチャに依存しないことが望ましい．

(3) スピンによる他コアのメモリアクセスへの影響が少ない
ロックを取得できたかどうかを繰返しチェックするとバスアクセスが頻繁に発生する．バスが浪費されることにより，メモリへのアクセス時間が長くなるため，スピンロックのバスアクセスを少なくする必要がある．

(4) ハードウェア規模が小さい
ハードウェア規模は製造コストに直接影響を与えるため，小さいことが望ましい．

3.2 ハードウェアの詳細

提案するハードウェア（以降，QLP-HW と呼ぶ）は，優先度発行ユニットと優先度順スピンロックユニットで構成される．優先度発行ユニットはシステムに 1 つ必要で，優先度順スピンロックユニットはロック個数と同数だけ必要である．**図 4** に 2 つのロックを QLP-HW を用いて実現するシステムの例を示す．2 つのユニットを用いてロックを取得する手順は次のようになる．ロックを取得するコアは，優先度発行ユニットから優先度を取得し，優先度順スピンロックユニットに設定する．この優先度はロックを取得する順序を決めるもので，優先度順スピンロックユニットは設定された優先度の中で最も優先度の高いコアにロックを与える．

3.2.1 優 先 度

優先度は，優先度発行ユニットと優先度順スピンロックユニットの両方で利用する値である．優先度発行ユニットから読み出した優先度を優先度順スピンロックユニットに設定する．優先度には，有効値と無効値があり，優先度発行ユニットからは有効値のみを読み出す．優先度順スピンロックユニットには，有効値もしくは無効値を設定することができる．無効値が設定されているコアはロック取得を要求していないことを表す．有効値は，値の小さいものほど高い優先度を持つことを表す．

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.61
Vol.2010-EMB-16 No.61
Vol.2010-MBL-53 No.61
Vol.2010-UBI-25 No.61
2010/3/28

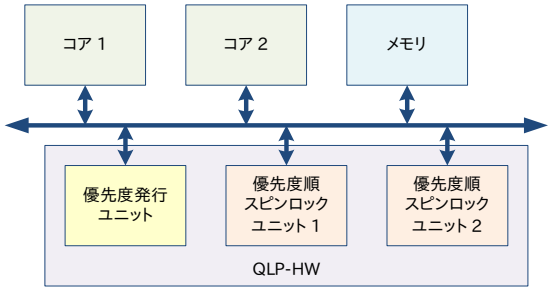

図 4 提案ハードウェアを利用したシステムの例
Fig. 4 An example system with proposed hardware

3.2.2 優先度発行ユニット

優先度発行ユニットは，ロックを取得しようとしているコアに対し，ロック取得の優先度を与える．優先度はユニット内部のカウンタにより保持されており，値を読み出すたびにインクリメントされる．複数のコアが同時にアクセスした際は，バスによる調停が行われるため，複数のコアはそれぞれ異なる値を読み出す．値が小さいほど高い優先度のため，値を先に読み出したコアが高い優先度を持つ．優先度発行ユニットにより，FIFO 順にロックを取得するという原則が与えられる．

3.2.3 優先度順スピンロックユニット

優先度順スピンロックユニットは，ロック取得可能なコアを決定するユニットである．内部には，コアごとに優先度レジスタとロック取得フラグの組を持つ．例としてコア数 4 のときの優先度順スピンロックユニットを**図 5** に示す．

優先度順スピンロックユニットの状態遷移を**図 6** に示す．状態はアンロック状態とロック状態の 2 状態を持ち，リセット時にはアンロック状態となる．アンロック状態の時は，有効値を保持している優先度レジスタの値を比較し，最も高い優先度を持つ優先度レジスタに対応するロック取得フラグをセットし，ロック状態に遷移する．全ての優先度レジスタの値が無効値である場合はアンロック状態のまま遷移しない．ロック状態の時は，セットされているロック取得フラグに対応する優先度レジスタに無効値が書込まれると，ロック取得フラグをクリアして，アンロック状態に遷移する．

ロックを取得したいコアは，まず，優先度の有効値を優先度レジスタに設定する．ロック取得フラグをチェックし，セットされていれば，ロック取得に成功する．ロック取得フラグ

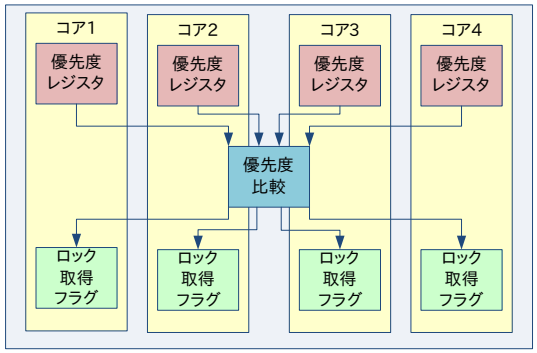

図 5 優先度順スピンロックユニット
Fig. 5 Priority ordering spin lock unit

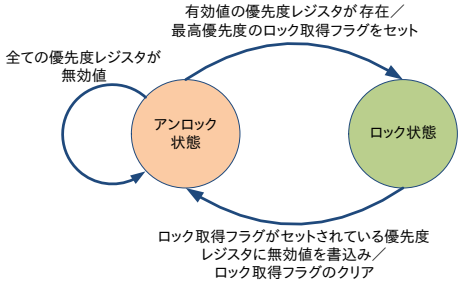

図 6 優先度順スピンロックユニットの状態遷移図
Fig. 6 State transition diagram of the priority ordering spin lock unit

がセットされていないときは，ロック取得に成功するまでロック取得フラグをチェックする．クリティカルセクション実行後は，優先度レジスタに無効値を書込むことでロック解放を行う．

優先度の比較について，優先度のオーバフローを考慮する必要がある．優先度発行ユニットのカウンタは有限ビット長のため，優先度はオーバフローが発生する可能性があり，オーバフローが発生すると，単純な大小比較ではうまくいかない．そのため，比較は ICTOH[4] と同じアルゴリズムを用いている．これは，周期 P の N ビット長カウンタについて，2 つの値の差をとった結果の最上位ビット（$N-1$ ビット目）を比較結果とするものである．こ

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.61
Vol.2010-EMB-16 No.61
Vol.2010-MBL-53 No.61
Vol.2010-UBI-25 No.61
2010/3/28

表 1 ドライバの低レベル関数
Table 1 Lower-level functions of driver

関数名	概要	引数
get_priority()	優先度発行ユニットから優先度を取得する	なし
enter_spinlock()	優先度レジスタに優先度を設定する	qlpid, prcid, priority
try_spinlock()	ロック取得フラグをチェックする	qlpid, prcid
release_spinlock()	優先度レジスタに無効値を書込む	qlpid, prcid
suspend_spinlock()*1	優先度レジスタに無効値を書込む	qlpid, prcid

のアルゴリズムでは比較する値の差は $P/2$ 未満である必要がある．提案手法は，割込みによる中断がなければ優先度の高いコアから順にロックが渡される．P の大きさにもよるが，割込みによる中断が頻発することにより，優先度の差が $P/2$ を超える可能性は低いと思われる．

3.3 ドライバ

QLP-HW を利用するドライバについて述べる．

ロック取得・解放処理について述べる前に，これらの処理が利用する関数を**表 1** に示す．関数の引数として，prcid はコアの識別子を，qlpid は優先度順スピンロックユニットの識別子を，priority は優先度をそれぞれ表す．

3.3.1 ロック取得・解放

QLP-HW を利用したロック取得・解放のルーチンを**図 7**，**図 8** に示す．

ロックを取得する際には，priority 配列には無効値（INVALID）が格納されているため，優先度発行ユニットから get_priority() 関数により優先度を読み出す．読み出した優先度は enter_spinlock() 関数により優先度レジスタに設定する．優先度順スピンロックユニットは，有効値の優先度を比較し，最高優先度のコアに対応するロック取得フラグをセットする．ロックが取得できたかどうかを try_spinlock() 関数によりチェックを行い，取得できていれば，クリティカルセクションの実行に移る．取得できていないときは，割込み処理を行うことで割込み応答性を向上させる．割込み応答性に関しては後述する．

ロックの解放は，優先度順スピンロックユニットに対し無効値を設定し，配列に無効値を設定する．

3.3.2 ロック取得待ちの中断

ロック取得待ち中の割込み応答性を向上させるために，ロック取得できなかった場合は

```
 1:  /* 割込み禁止 */
 2:  disable_interrupt();
 3: retry:
 4:  /* 初回のみ優先度を取得する */
 5:  if( priority[prcid] == INVALID ) {
 6:    priority[prcid] = get_priority();
 7:  }
 8:  spinning[prcid] = true;
 9:  /* 優先度を設定する */
10:  enter_spinlock(qlpid, prcid, priority[prcid]);
11:  while( try_spinlock(qlpid, prcid) ) {
12:    enable_interrupt();
13:    /* ここで割込み処理に移る */
14:    disable_interrupt();
15:    /* 割込み処理をしていたらリトライする */
16:    if (spinning[prcid] == false) {
17:     goto retry;
18:   }
19:  }
```

図 7 ロック取得ルーチン
Fig. 7 Routine of acquiring a lock

割込み処理を行う．具体的には，try_spinlock() 関数でロック取得できていないことがわかった場合，一旦割込みを許可する．割込み要求がある場合はここで割込み処理が実行される．割込みハンドラの実行に移る前の，割込み禁止区間で**図 9** を実行する．図 9 の処理により，優先度レジスタに無効値が書込まれるため優先度順スピンロックユニットによる優先度比較によって，割込み処理を実行するコアにロックが渡されることがない．割込み処理の実行後は，get_priority() 関数を呼ばずに，割込み処理前と同じ優先度を優先度レジスタに設定する．

優先度を再度取得することがないため，割込み処理前後ではロック待ちキューの位置が変化しない．このようにすることで，割込み処理によってロック待ちキューの最後尾に並びなおすことがない．ロック待ちキューから外す処理がない点は QLPR と同じであるが，ロック取得可能なコアを決めるのはハードウェアで行っているため，QLPR で割込み処理中のコアをスキップする処理が必要ない．

*1 release_spinlock() 関数と suspend_spinlock() 関数は同じ処理内容であるが，関数を呼ぶ意味が異なる

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.61
Vol.2010-EMB-16 No.61
Vol.2010-MBL-53 No.61
Vol.2010-UBI-25 No.61
2010/3/28

```
1:  /* 無効値を設定する */
2:  release_spinlock( qlpid, prcid );
3:  spinning[prcid] = false;
4:  priority[prcid] = INVALID;
5:  enable_interrupt();
```

図 8 ロック解放ルーチン
Fig. 8 Routine of releasing a lock

```
1:  /* 無効値を設定する */
2:  suspend_spinlock( qlpid, prcid );
3:  spinning[prcid] = false;
```

図 9 割込み処理入口ルーチン
Fig. 9 Interrupt service routine

4. 評　　価

QLP-HW を Altera 社の FPGA 上で実装し，その他のスピンロックアルゴリズムとの比較を行う．

4.1 評 価 環 境

提案ハードウェアを VHDL で記述し，Altera 社の QuartusII version 8.1 を用いて合成を行った．優先度は 16 ビットで，優先度発行ユニットのカウンタと優先度レジスタを 16 ビットにした．FPGA として Stratix 1S40 を，コアとして周波数 50MHz の NiosII プロセッサを 4 つ用いて評価を行った．NiosII プロセッサの命令キャッシュは 4KB で，データキャッシュはない．プログラム領域およびデータ領域はすべてオンチップメモリに配置した．コンパイラは GNU gcc version 3.4.6 を，RTOS に TOPPERS/FMP カーネル Release 1.1.0[5] を用いた．

コアが複数あるシステムでは，バス衝突により実行時間が大きく変動する．初回の計測結果は命令をメモリから読み込む際にバス衝突が発生するが，2 回目以降は命令キャッシュにヒットするため命令に関するバス衝突の影響が無くなる．このため，初回の実行時間を評価に含めない．

NiosII アーキテクチャでは，従来手法のキューイングスピンロックをソフトウェアで実装するために必要な命令を持っていない．提案手法と従来手法との比較を行うため，LL/SC 命令を実現するカスタム命令を作成した．

表 2 ロック取得・解放処理時間（競合なし）
Table 2 Execution time of acquiring and releasing a lock without lock contention

アルゴリズム	TAS	MCS	QLPD	QLPR	QLP-HW
実行時間 [μs]	1.86	1.46	1.38	1.34	1.02

4.2 評価項目と評価方法

以下の項目について評価を行った．

- ロックの取得・解放処理時間
  - ロック取得の競合がない場合
  - ロック取得の競合がある場合
- 割込み応答時間
- ハードウェアサイズ

ロックの取得・解放処理時間の評価は，ロック取得，クリティカルセクション，ロック解放，という処理を 100 万回繰り返し行い，ロック取得前からロック解放後までの時間の分布を計測することで行った．各コアには，1ms 周期でタイマ割込みが発生し，割込みによるロック取得待ちの中断が発生するようにした．タイマの割込みハンドラの処理時間は約 13μs である．クリティカルセクションの処理は，グローバル変数のインクリメントと空ループであり，コア間の競合が発生しない状況での実行時間は約 65μs である．ロック解放とロック取得の間に遅延時間を挿入することで，競合しないケースも分布に含めている．遅延は一様分布に従った乱数による空ループで，実行時間は 2μs から 162μs である．

比較対象としたアルゴリズムは，TAS 命令に相当する Mutex 回路を用いた TAS スピンロック（TAS），キューイングスピンロックである MCS ロック（MCS），ソフトウェア実装の中断可能なキューイングスピンロックアルゴリズム（QLPD, QLPR）である．

4.3 ロックの取得・解放処理時間

4.3.1 ロック取得の競合がない場合

コア 1 のみがロック取得と解放処理を行い，他のコアはコア 1 に対して影響を与えない状況で，コア 1 のロック取得と解放処理の実行時間を計測した．計測した結果を**表 2** に示す．QLP-HW は，他のアルゴリズムよりもロック取得と解放処理時間が短くなった．

4.3.2 ロック取得の競合がある場合

4 つのコアがそれぞれロック取得，クリティカルセクション，ロック解放処理を行う状況で，コア 1 のロック取得・解放にかかる時間と割込み応答時間について計測を行った．ロッ

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.61
Vol.2010-EMB-16 No.61
Vol.2010-MBL-53 No.61
Vol.2010-UBI-25 No.61
2010/3/28

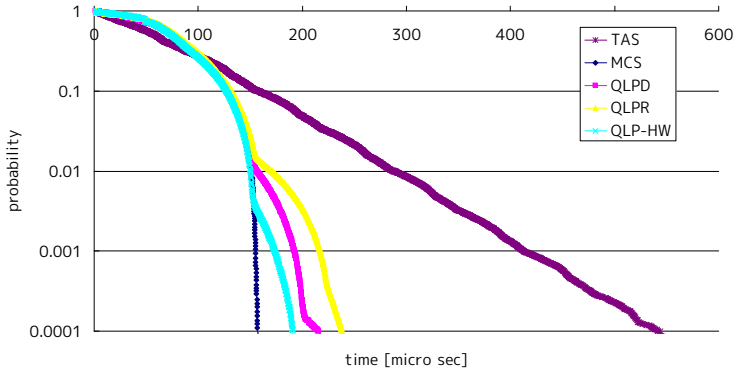

図 10 スピンロックの取得・解放処理時間の分布（競合がある場合）
Fig. 10 Execution time of acquiring and releasing a spin lock

ク取得・解放処理の実行時間を**図 10** に示す．この実行時間には自コアのクリティカルセクション実行時間は除いてあるが，ロック取得を待っている間に実行される他コアのクリティカルセクションの実行時間が含まれている．また，縦軸はその時間までに処理が終わらなかった確率を表す．

途中で割込み処理を行った場合のロック取得・解放処理の実行時間を**図 11** に示す．QLPD では，途中で割込み処理を行うとキューの最後尾になるため，QLPR に比べて実行時間が長くなる．QLP-HW は，ハードウェアサポートにより実行時間が短くなっている．

4.4 割込み応答時間

ロック取得待ちの間における割込み応答時間を**図 12** に示す．MCS ロックと TAS スピンロックは割込み処理を受け付けないため，応答時間が長くなっている．中断可能なキューイングスピンロックの 3 つは応答時間がほぼ同じになっている．

4.5 ハードウェアサイズ

優先度順スピンロックユニットは，コアの数だけ優先度レジスタとロック取得フラグが必要である．16 ビットの優先度レジスタについて 4 コアと 8 コアの実装を FPGA で行った結果，ロジックエレメントは**表 3** のようになった．比較のために，TAS スピンロックに用いた Mutex 回路，1 チャネルの 32 ビットタイマ，1 コアと 4 コアのシステム全体のハードウェアサイズを計測した．システムには，コアごとに，タイマが 1 個ずつ，JTAG UART

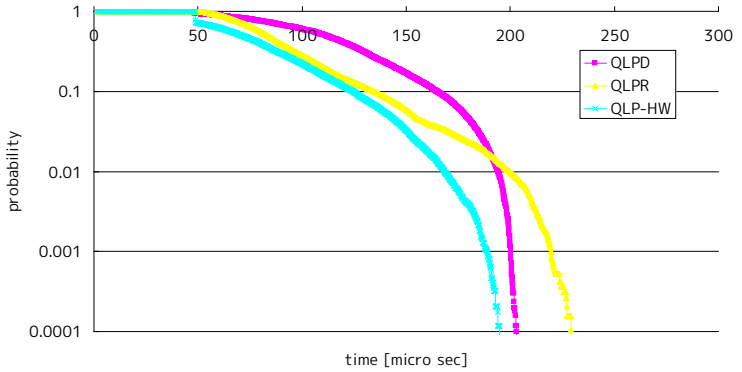

図 11 割込み処理を行った場合のスピンロックの取得・解放処理時間の分布
Fig. 11 Execution time of acquiring and releasing a spin lock with interrupt service routine

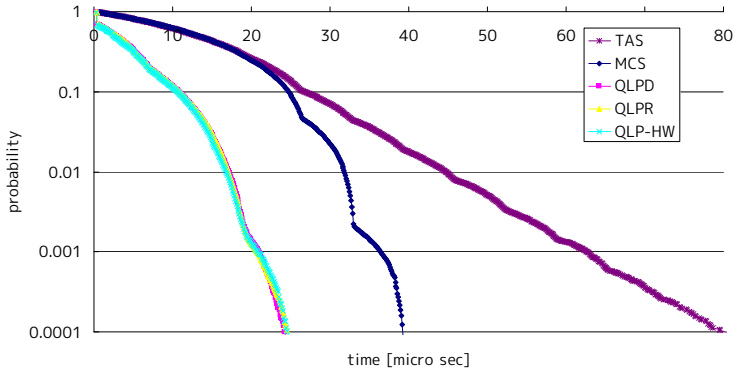

図 12 割込み応答時間の分布
Fig. 12 The interrupt latency times

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.61
Vol.2010-EMB-16 No.61
Vol.2010-MBL-53 No.61
Vol.2010-UBI-25 No.61
2010/3/28

表 3 ハードウェアのサイズ
Table 3 Hardware size

	Mutex	タイマ	システム全体		QLP-HW	
			1 コア	4 コア	4 コア	8 コア
サイズ [LE]	50	157	6431	22961	275	927

が 1 個ずつ，Mutex 回路が 2 つあり，他にオンチップメモリが 260KB，LL/SC 専用メモリが 4KB ある．優先度比較はコア間でそれぞれ行っているため，コア間の組み合わせが増える．具体的には 4 コア時には $6(={}_4C_2)$ 個だった比較器が，8 コア時には $28(={}_8C_2)$ 個になった．

4.6 考　　察

TAS スピンロックは，ロック取得の順序がランダムで決まるため，実行時間が長くなる可能性が他の手法に比べて高い．MCS ロックは，FIFO により取得順序が決まるが，割込み処理による待ち状態の中断をしないため，割込み応答時間が悪くなっている．また，比較に用いた TAS スピンロックも取得待ち中に割込み処理を実行しない．TAS スピンロックはロック取得・解放処理時間が長いため，割込み応答時間は MCS ロックより悪くなっている．

提案手法は，ソフトウェア実装のキューイングスピンロックアルゴリズムと比べて割込み応答性を悪化させず，ロック取得・解放処理の最大実行時間が短くなった．また，競合がない場合においても，他のアルゴリズムと比べてロック取得・解放の実行時間が短い．このことから，ソフトウェア実装のキューイングスピンロックアルゴリズムと比べて，平均実行時間および最大実行時間の両方が短くなると考えられる．最大実行時間が短くなるため，リアルタイム性が向上していると言える．

ハードウェア実装の要件のうち，プロセッサに依存した命令やバスを利用しないという項目と特定の割込みアーキテクチャに依存しないという項目は満たせた．ハードウェア規模については，表 3 に示したように，4 コアのシステムでは全体の約 1%と小さいため，許容できると考えられる．他コアへの影響については，バス衝突を低減するためにメモリへのバスと QLP-HW へのバスが異なることが必要である．本研究で用いた NiosII アーキテクチャでは異なるバスとなるため，スピンロックによる他コアのメモリアクセスへの影響がないと考えられるが，他のアーキテクチャでは検討が必要である．

5. 関 連 研 究

ハードウェアサポートによるスピンロックアルゴリズムが Saglam らによって提案されている[6]．このアルゴリズムは，ロックを取得できなかった時にロック状況を繰返しチェックするのではなく，ロックを解放するコアが次にロックを取得するコアに対して割込みを発生させる機構をハードウェアで実現することにより，ロック取得待ちのバスアクセスを低減している．ロック取得の順序は FIFO を選択可能なため，ロック取得までの時間に上限があるが，割込み応答性については述べていない．

6. おわりに

本研究では，実行オーバヘッドを低減させる，割込みによる待ち状態の中断が可能なキューイングスピンロックハードウェアを提案した．提案ハードウェアをペリフェラルとしてシステムに組込むことで，プロセッサに依存しない排他制御の実現が可能となる．

今後の課題として，2 つの拡張が考えられる．優先度レジスタに設定された値をそのまま比較していたが，あるコアの優先度を上げることで優先的にロックを渡すことが可能となる．これは，特定のコアに所属するロックについては，そのコアに優先的にロックを渡すというものである．また，優先度比較において最高優先度だけでなく次に高い優先度を求め，保持するレジスタを用意することで優先度継承アルゴリズムを実装可能となる．また，RTOS 内部で利用しているスピンロックと置換えて評価することも考えられる．

参 考 文 献

1) Mellor-Crummey, J.M. and Scott, M.L.: Algorithms for scalable synchronization on shared-memory multiprocessors, *ACM Trans. Comput. Syst.*, Vol.9, No.1, pp. 21–65 (1991).
2) 高田広章，坂村健：中断可能なキューイングスピンロックアルゴリズム，電子情報通信学会論文誌 D-I， Vol.J78-D-I, No.8, pp.661–669 (1995).
3) Takada, H. and Sakamura, K.: Predictable Spin Lock Algorithms with Preemption, *In Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software*, IEEE Computer Society Press, pp.2–6 (1994).
4) Carlini, A. and Buttazzo, G.C.: An efficient time representation for real-time embedded systems, *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, ACM, pp.705–712 (2003).
5) TOPPERS プロジェクト：“http://www.toppers.jp/”.
6) Saglam, B.E. and III, V. J.M.: System-on-a-Chip Processor Synchronization Support in Hardware, *Design, Automation and Test in Europe Conference and Exhibition*, pp.633 – 641 (2001).

© 2010 Information Processing Society of Japan