布線論理によるタスクスケジューリング時間の平準化技法 *

請　園　智　玲 **
荒　木　光　一 ***

An Equalization Technique of Time for Task Scheduling by Wired Logic

Tomoaki Ukezono**, Koichi Araki***

Software overheads that are caused by task scheduling which is implemented in operating systems may change over time. For this reason, the overheads will be error in estimation of execution time on embedded system. On the other hand, integration degree of transistors on LSI has been continued to dramatically increase by build-out of recent fine processing technology. In this paper, we propose a hard wired mechanism that can achieve equalization of time for task scheduling on embedded systems. The proposed mechanism can reduce and equalize time for real-time scheduling, simplify estimating execution time and verification of real-time performance.

Key Words : Embedded OS, Task Scheduler, Wired Logic, Software Overhead

１．はじめに

リアルタイム組込みシステムでは，システムのピーク性能やスループットよりも，リアルタイム性によってシステムの価値が決まる．リアルタイム性とは個々のタスクが決められた時刻までに完了することを保証する性能である．リアルタイム・オペレーティングシステム(RTOS) は，これをサポートするためのリアルタイム・タスクスケジューリング機能を備える [1]．しかしながら，タスクスケジューリングにもタスク本体の実行と同様に演算時間が必要であり，RTOS が取り扱うタスクの数や種類がスケジューリング処理のタイミングごとに異なる場合，タスクスケジューリングの演算時間が変動する [2]．このため，システム設計時にリアルタイム性を保証するための実行時間の見積もりを複雑にする．

一方，近年の半導体製造技術は進歩を遂げ，2016 年の時点で，14nm プロセス [3] が実現されている．このサイズは，その 20 年前の 0.35m プロセス [4] と比べ，1/25 に減少している．このことから，20 年前に比べ単位面積あたり，およそ 625 倍の回路が実装可能になった．加えて現在では，3 次元集積回路 [5] などの更なる高集積化の技術が研究・開発されており，今後もチップあたりの集積度が向上していくことが予測される．その高集積度を活用した周辺回路のワンチップ化技術（SoC: System on a Chip）は，組込みシステムの高性能化・高機能化に貢献するとともに，組込み製品を低コスト化する基盤技術となっている．

これまで，ハードウェアサイズを小さくする目的で，オペレーティングシステムの機能は，特権モードやメモリ保護，仮想記憶などのサポートを除き，基本的にできるだけソフトウェアで実装されてきた．本研究はこの点に着目し，組込み向け SoC の機能部品の一つとして，RTOS のタスクスケジューリング機能の一部を布線論理で実装する提案を行い，近年の高集積度の半導体製造技術においてその実現可能性を議論するとともに，タスクスケジューリング時間の平準化効果を評価する．

本研究で評価対象とするタスクスケジューラは静的優先度スケジューリングを採用し，プリエンプションを許す．このようなタスクスケジューラを採用した代表的な

* 平成 30 年 11 月 30 日受付
** 電子情報工学科
*** 五大開発株式会社 技術研究所

RTOS として，μITRON 仕様 [7] に準拠した T-Kernel[6] が挙げられる．T-Kernel は小惑星探査機はやぶさや，金星探査機あかつきに Space Cube 2[8] として搭載された実績を持ち，近年では高信頼性が必要とされる組込みシステムを構築する際の選択肢として有望視されている．

提案手法は静的優先度スケジューリングで処理される全て，または，一部のタスクセットを布線論理で実装したレディーキューと呼ばれる実行可能状態であるタスクの実行順序を管理する機構の上で処理させることにより，タスクスケジューリングに関わる演算を最大並列で実行し，以下に分類する処理時間を平準化する．

(a). SELECT：実行可能状態のタスク群から次に実行すべきタスクを選ぶ処理時間
(b). REMOVE：実行可能状態のタスク群から指定のタスクを除去する処理時間
(c). INSERT：実行可能状態のタスク群に新たにタスクを追加する処理時間

(a)SELECTは先行して実行しているタスクが完了し，次に実行すべきタスクを選ぶとき，または，自タスクの優先度を変更したときに，(b)REMOVEは，例えば，wait()／sleep()／receive()などのサービスコールで待ち合わせが発生し，タスクを実行可能状態から除去するときに，(c)INSERTは外部割り込みやサービスコールに関係して，新たにタスクが実行可能状態のタスク群に追加されるときに，RTOSが必要とする処理時間である．これらを平準化することにより，組込みシステム設計時の実行時間の見積もりを容易にすることができる．

リアルタイム処理を実現するには，システム設計時に予定した時間までに各処理が完了することを保証する必要がある．このとき，完全な保証を必要とするか，限定条件をつけたある程度の保証を必要とするかは，デッドラインミスによるシステムの価値の損失程度により異なる．リアルタイム性の保証はタスク実装前の設計時に，タスクの最悪実行時間を見積もることによってなされるが，一般的には，タスクセット設計者を設計・実装していないことから，システム OS コール内部の処理の最悪実行時間の見積もりを立てることが難しい．このため，タスクセット設計者にとってシステムコール処理の実行時間のばらつきが，システム全体の実行時間の見積もりに不明な変数となる．本研究が着目するタスクスケジューリング時間の平準化は，この不明な変数を除外することに貢献する．もし，完全な平準化がなされれば，タスクスケジューリングという，数あるシステムコール処理の一部だけでも，最悪実行時間が静的に決まり，組込みシステム設計時のリアルタイム性の保証につながる．また，タスクスケジューリング処理の時間は実行可能状態のタスクの数と種類，システムコールによるタスク状態操作の種類によって，実行時に変動する．加えて，非周期的な外部割込みによるタスク起動を許す場合，さらにこの問題は複雑となる．このような複雑な処理時間変動要因をもつ変数を一部でも除外できることは，リアルタイム・システム設計者にとって大きな助けとなる．

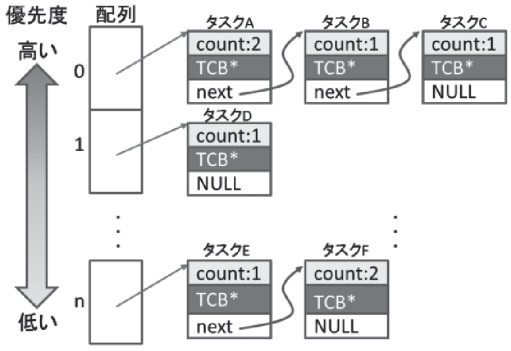

図1 レディーキューのデータ構造

本研究の提案する布線論理で処理を実装することで，最大並列でスケジューリング処理を実行することが可能になる．このことから，後の章で評価する提案手法の回路遅延によってはレディーキューの応答速度の向上が実現し，結果として，タスクの起動や完了などを伴うシステムコールの応答時間が短縮され，システム全体の応答速度を偶発的に高める効果が望める．

しかしながら，リアルタイム性能と応答速度は互いに直交性のある評価軸であることに注意が必要である．本研究において，単なる偶発的なシステム応答速度の向上は，提案する実装の副次的効果である．リアルタイム・システムが保証するリアルタイム性は，与えられたハードウェア資源の中でシステムの設計時に静的に保証するものであり，処理オーバヘッドがシステム設計者の意図とは関係なく，偶発的に短縮し，その結果，必然的にデッドラインミスをする予定のタスクが偶然デッドラインを守れたとしても，それはリアルタイム性が保証されたことにはならない．

本論文は以下の章で構成される．本章では研究の背景及び目的を示した．2 章では，提案手法であるタスクスケジューラの布線論理化に関し，詳細を説明する．3 章では，提案手法の FPGA 上での実装データをもとに，回路規模及び遅延を評価し，タスクスケジューラの布線論理化が現在の半導体製造技術で実現可能かを議論するとともに，タスクスケジューリング時間の平準化について評価する．4 章では，タスクスケジューリングのハードウェア化という観点から，関連研究を紹介し，本研究との相違点を述べる．5 章で本論文をまとめる．

２．布線論理によるタスクスケジューラ

２．１　布線論理化の対象となるレディーキューの構造

図 1 に本研究が布線論理化の対象にした静的優先度スケジューリングを実現するレディーキューのデータ構造を示す．このデータ構造を用いるスケジューラは，優先

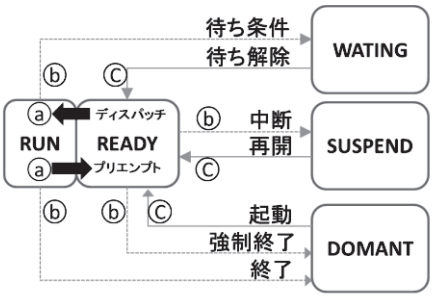

図2 タスクの状態遷移とレディーキューへの操作の関係

度毎に単方向のリスト構造を持ち，同一優先度内では基本的に FCFS(First Come First Served) を実現する．一般的な実装では，各優先度のリストの先頭をポインタ配列で管理し, 優先度の値をその配列の添字として取り扱う．各リストの要素は以下の 3 つの構造体メンバをもつ．

(1) 当該タスクが起動要求された数 (count)
(2) TCB へのポインタ (TCB*)
(3) 次のリスト要素を指すポインタ (next)

TCB とは Task Control Block の略称であり，タスク ID やコンテキスト情報など，タスクの属性が保存されるメモリ領域である．count は実装によって TCB の中に確保されることも考えられるが，図 1 では説明の容易さのために，リスト要素の構造体メンバとした．各優先度では，リストの先頭 (左側) から順にタスクは実行される．タスクが完了したときに，count が 2 以上であるならば，count を 1 減算して，リストの末尾につなぎ直す．もし，タスクの完了時に count が 1 であるならば，リストの要素から除去する．

例えば，タスク A は最も高い優先度 (優先度 0) の先頭に存在するため，現在実行中 (RUN 状態) のタスクである．タスク A が完了したとき，直後に実行すべきタスクとして，タスク A の next によってタスク B が選出される．次にタスク A の count が 2 であることから，count を 1 減算し，タスク C の次につなぎ変えられる．そのままレディーキューの情報に変化がなくタスク B が完了した場合，タスク B の count は 1 であるため，優先度 0 のリストからタスク B に関する要素が除去される．この処理形式に従うと，図 1 の例では，タスクの実行順序はタスク A→B→C→A→D→E→F→F となる．

このように count を導入することで，厳密に FCFS を守ることができなくなる代わりに，TCB* が同一タスクを指すリスト要素がスケジューラ内に複数存在しなくなり，いかなるタスク起動状況においても，リスト要素に必要なメモリ資源の上限をタスクセットの設計段階で静的に決定することができる．

図2にタスクの状態遷移とレディーキューの関係を示す．レディーキューに関係するタスク状態は5つ(RUN, READY，WATING，SUSPEND，DOMANT)ある．図2ではそれぞれ状態を矩形で示している．各タスクがその5つの状態間で遷移するときにレディーキューが変化する．遷移は矢印で示される．図1の例でいえば，RUN状態のタスクがタスクAであり，それ以外のタスクがREADY状態である．READY／RUNから，この2つ以外の状態(WAT-ING／SUSPEND／DOMANT)にタスクが遷移するとき，そのタスクはリスト要素としてレディーキューから外れる．その反対に，WATING／SUSPEND／DOMANTからREADY／RUNにタスク状態遷移するときに，そのタスクはリスト要素としてレディーキューに追加される．RUNとREADYの間でタスク状態が遷移する場合，これをディスパッチ／プリエンプトと呼ぶが，レディーキュー内ではリスト要素数が変化せず，リストのつなぎ変えが行われるのみである．

図 2 のアルファベット a ～ c は, 1. で示したスケジューリングに関わる処理時間の分類，SELECT，REMOVE，INSERT の列挙番号 (a) ～ (c) に対応する．このように，レディーキューの変化に伴う RTOS の演算はタスクセット中の 1 つのタスクを状態遷移させる原因となる全てのシステムコールの実行タイミングで発生する．リアルタイム・システム設計者は SELECT，REMOVE，INSERT の操作に必要な処理時間が異なる場合, または, レディーキューがその時々に扱うリスト要素数によって処理時間が異なる場合，システムの実行時間見積もりが困難になる．

２．２　レディーキューへの操作

図 2 にタスクの状態遷移とレディーキューの関係を示す．レディーキューに関係するタスク状態は 5 つ(RUN, READY，WATING，SUSPEND，DOMANT）ある．図 2 ではそれぞれ状態を矩形で示している．各タスクがその 5 つの状態間で遷移するときにレディーキューが変化する．遷移は矢印で示される．

図1の例でいえば，RUN状態のタスクがタスクAであり，それ以外のタスクがREADY状態である．READY／RUNから，この2つ以外の状態（WAT-ING／SUSPEND／DOMANT）にタスクが遷移するとき，そのタスクはリスト要素としてレディーキューから外れる．その反対に，WATING／SUSPEND／DOMANTからREADY／RUNにタスク状態遷移するときに，そのタスクはリスト要素としてレディーキューに追加される．RUNとREADYの間でタスク状態が遷移する場合，これをディスパッチ／プリエンプトと呼ぶが，レディーキュー内ではリスト要素数が変化せず，リストのつなぎ変えが行われるのみである．

図 2 のアルファベット a ～ c は, 1. で示したスケジューリングに関わる処理時間の分類，SELECT，REMOVE，INSERT の列挙番号 (a) ～ (c) に対応する．このように，レディーキューの変化に伴う RTOS の演算はタスクセッ

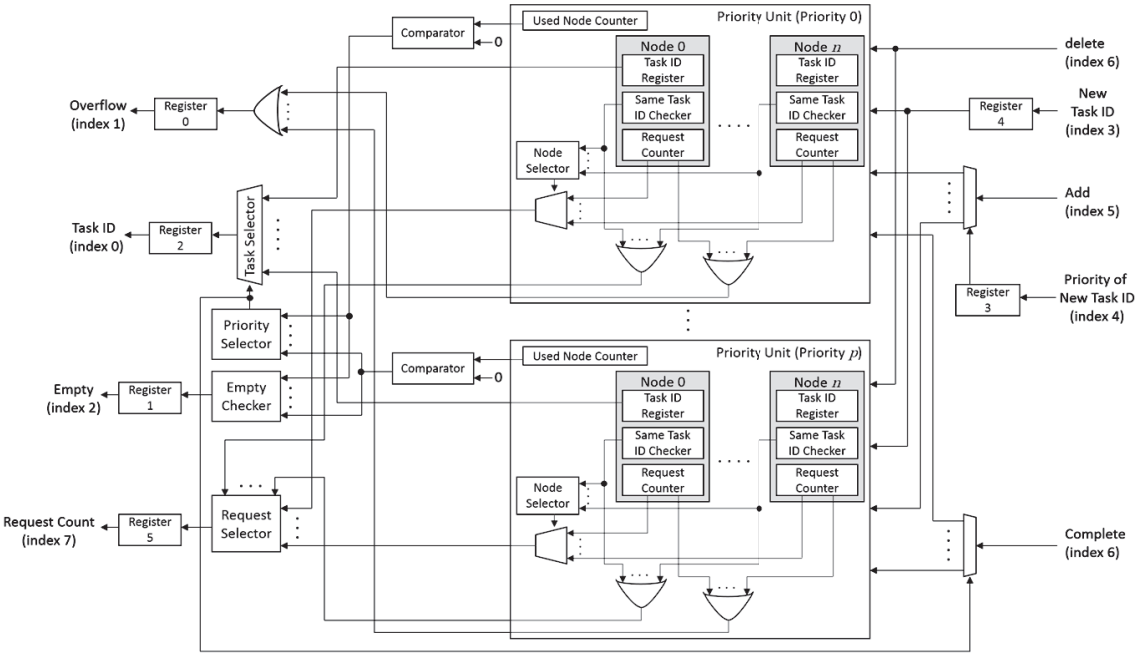

図3　布線論理で実現したレディーキューのブロック図

ト中の 1 つのタスクを状態遷移させる原因となる全てのシステムコールの実行タイミングで発生する．リアルタイム・システム設計者は SELECT，REMOVE，INSERT の操作に必要な処理時間が異なる場合, または, レディーキューがその時々に扱うリスト要素数によって処理時間が異なる場合，システムの実行時間見積もりが困難になる．

２．３　布線論理化したレディーキュー

２．３．１　HWレディーキューの構造

図 3 に布線論理で実現したレディーキュー (HWレディーキュー) のブロック図を示す．図 3 内で表現される回路は Task ID Register，Request Counter，Used Node Counter と入出力時のラッチである Register0 ～ 5 を除き，全て組み合わせ回路で実装されている．HWレディーキューは灰色の矩形で示されるノード (Node) の配列により構成される．各ノードは図 1 におけるリスト要素と同等の意味をもつ．一つのノードは以下の機能をもつ．

- タスク ID レジスタ (Task ID Register)
- 起動要求カウンタ (Request Counter)
- タスク ID チェッカ (Task ID Checker)

タスク ID レジスタは図 1 のリスト要素の TCB* に相当する．RTOS はこのタスク ID をキーとして主記憶を検索し，TCB の配列からタスクのコンテキスト情報を取得する．この検索は RTOS が TCB を ID 順の配列で管理することにより，ポインタでの管理と同程度の演算コストで実現できる．起動要求カウンタはタスクの追加処理時に，HWレディーキュー内に既に同一のタスク ID が存在した場合に値が増加されるカウンタである．起動要求カウンタは図 1 のリスト要素の count に相当する．タスク ID チェッカはタスク ID レジスタに保存されている ID と追加／削除するタスク ID が同一であるかを常に監視する組み合わせ回路である．

一つ以上のノードは優先度ユニット(Priority Unit)としてまとめられる．この優先度ユニットは図1における一つのリスト構造に相当する．図3では，p＋1の優先度ユニットが存在し，各優先度ユニットのノード数はn＋1個となる構成で表現されている．各優先度に必要なノード数は実現する組込みシステムによって異なる．しかしながら，本研究では評価の簡単化のために全ての優先度ユニットにn+1個のノードを備え，評価対象のHWレディーキューの備える全ノード数を(p＋1)(n＋1)とした．また，起動要求カウンタ内のレジスタは5ビット幅に固定し，管理できる起動要求回数を31回までとした．

各優先度ユニット内では，最も先頭 (左側) に位置するノード 0 が，その優先度内で最初に実行すべきタスクである．ゆえに，全ての優先度ユニットのノード 0 を常に監視し，最も優先度が高い優先度ユニットのノード 0 が RUN 状態のタスクとして選ばれるように Task Selector が選択する．優先度ユニットは管理するノード内の情報をノード 0 方向に向かってシフトする機能を備える．シフトするタイミングは図 1 における，リストのつなぎ変えのタイミングと同様である．

RTOSがHWレディーキューのタスクを削除する場合は，削除するタスクのIDが回路の外部から指定される．削除処理はタスクの状態がRTOSのサービスコール

SELECT Operation
Parameter : r1=base address, r2=1

```
l.sw  24(r1), r2 /* send complete signal */
l.lwz r3, 0(r1)   /* get next task ID */
l.lwz r4, 8(r1)   /* get empty flag */
```

REMOVE Operation
Parameter : r1=base address, r2 = task ID,
r3=1

```
l.sw  12(r1), r2  /* set task ID */
l.sw  28(r1), r3  /* send remove signal*/
l.lwz r4, 32(r1)  /* get # of request */
l.lwz r5, 8(r1)    /* get empty flag */
l.lwz r6, 0(r1)    /* get next task ID*/
```

INSERT Operation
Parameter : r1=base address, r2 = task ID,
r3=priority,　　r4=1

```
l.sw  12(r1), r2  /* set task ID */
l.sw  16(r1), r3  /* set priority*/
l.sw  20(r1), r4  /* send insert signal*/
l.lwz r5, 0(r1)    /* get next task */
l.lwz r6, 4(r1)    /* get overflow flag*/
```

図4　レディーキューのソフトウェア制御プログラム

表1　HWレディーキューの制御インターフェース

index	用途	R/W	値範囲
0	RUN状態のタスクID	R	0~(p+1)×(n+1)-1
1	オーバフローフラグ	R	0~1
2	エンプティーフラグ	R	0~1
3	タスクID設定	W	0~(p+1)×(n+1)-1
4	タスク優先度設定	W	0~p
5	タスク追加の設定	W	0~1
6	タスク完了の通知	W	0~1
7	タスク削除の通知	W	0~1
8	削除タスクの起動要求回数	R	$0\sim2^5-1$

によりREADYからWATING／SUSPEND／DOMANTの状態に移行する場合に必要となり，これらの状態遷移では強制的にレディーキューから除去される．最初に，HWレディーキュー内のタスクIDチェッカは，タスクIDが一致するノードを発見する．次にHWレディーキューは発見されたノードより後方(右側)のノードを，発見されたノードを上書きしながら左にシフトする．上書きと同時に，上書きされるノードの起動要求回数は回路外部に公開されるレジスタに一時保存される．シフト処理後，リスト末尾の次のタスクIDは0に初期化される．

RTOS がHWレディーキューにタスクを追加する場合は，追加するタスクの ID と優先度が回路の外部から指定される．HWレディーキュー内に指定されたタスク ID が既に存在していた場合，HWレディーキューはノード内のタスク ID チェッカが一致を検知し，タスクの追加処理を中止するとともに，タスク ID が一致したノードの起動要求回数を 1 増加させる．ノードのタスク ID チェッカが全て不一致の場合，新しいリスト要素として追加処理が行われる．外部から指定された優先度で優先度ユニットを選択し，選択された優先度ユニットが管理するリストの末尾の次のノードに指定されたタスク ID を上書きする．このとき，上書きするノードの起動要求回数は 1 に初期化される．

RTOS が RUN 状態のタスクを完了する場合は，完了するタスクの起動要求回数が 1 のとき，RUN 状態のタスク (ノード 0) を該当するタスクとして，削除処理と同様のシフト処理が行われる．完了するタスクの起動要求回数が 2 以上であれば，ノード 0 のタスクの削除処理の後に，削除対象のタスクの ID でタスクの追加処理と同様の処理が行われる．この処理により，ノード 0 のタスクがリストの末尾に再接続される．このとき，末尾のノードの起動要求回数の初期値は 1 ではなく，完了時の起動要求回数を 1 減算した値となる．

２．３．２　ソフトウェアインターフェース

本研究で設計・評価したHWレディーキューは OpenRISC 命令セットアーキテクチャ [9] を実装した OR1200 CPU[10] を搭載する SoC の ORPSoC[11] の機能部品として接続可能になるよう設計をした．表 1 に ORPSoC 上に実装したHWレディーキューの制御インターフェースを示す．HWレディーキューを制御するためには 9 つの制御レジスタが必要である．ORPSoC では，SoC の機能部品であるペリフェラル回路と CPU 間の通信をメモリマップド I/O で行っている．ORPSoC にHWレディーキューを接続するにあたり，他のペリフェラルが使用してない I/O マップ空間の 0x98000000 番地をベースアドレスとしてHWレディーキューに割り当て，その番地から，4 バイトアラインで 9 つの制御レジスタを割り当てた．つまり，ベースアドレスと表 1 の index を 4 倍した値を足し合わせることで，制御レジスタのアドレスの計算を行う．なお，表 1 の index と図 3 の index は対応している．

SELECT，REMOVE，INSERT 操作のそれぞれの場合において，RTOS がHWレディーキューを操作するためのアセンブリコードを図 4 に示す．図 4 のアセンブリコードは OpenRISC 用のアセンブリ言語で記述されている．OpenRISC は MIPS と同様のアドレッシングモードを持ち，データ転送命令のアセンブリ記法もオペコードを除き MIPS と同一である．図 4 のプログラムでは，l. sw と l. lwz の 2 種類のデータ転送命令オペコードが使用されている．これはそれぞれ，4 バイトストア命令と 4 バイトロード命令である．ストア命令でHWレディーキューへのソフトウェア制御を行い，ロード命令でHWレディーキューから RTOS がタスクスケジューリングに必要な情報を得る. 図 4 に示す通り, SELECT には 3 命令, REMOVE には 5 命令, INSERT には 5 命令が必要である. レディーキュー操作に関する命令実行はこの命令を実行することのみで完結し，実行命令数は変動しない.

表2　回路生成ツール

処理	ツール名	Design Goal	Strategy
論理合成	Xilinx xst14.4	Balanced	Default
マッピング	Xilinx map14.4		
配線	Xilinx par14.4		

表3　Artix-7とSpartan-6の製品品番と最大リソース数

シリーズ	用途	Slice Registers	Slice LUTs	Total Slices
Artix-7	XC7A200T	269,200	134,600	33,650
Spartan-6	XC6SLX45	54,576	27,288	6,822

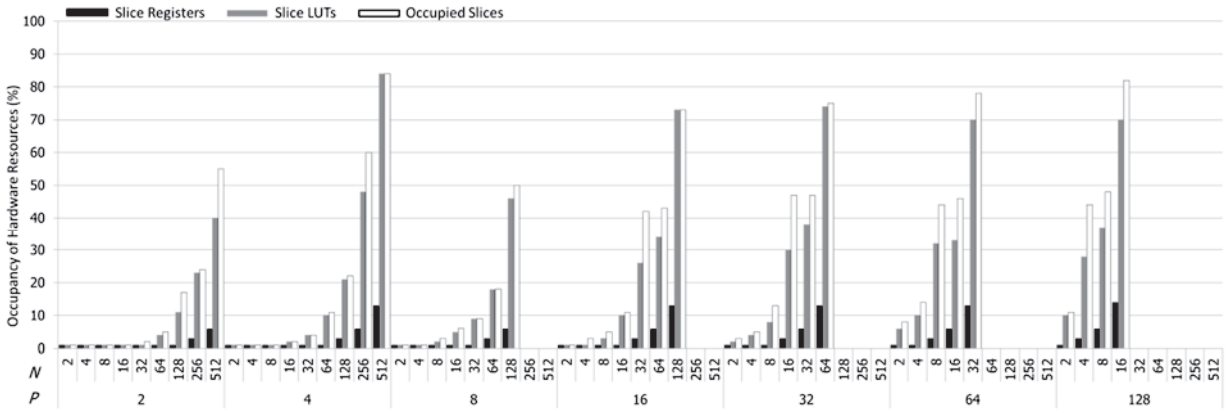

図5　HW レディーキューの構成に必要なハードウェアリソース

３．評価

３．１　回路設計及び評価の環境

提案手法単体での回路規模を評価するために，HWレディーキューを LSI 製造時のプロトタイピングデバイスとして用いられる FPGA 上に実装し，回路規模と遅延の測定を行う．ターゲットの FPGA は Xilinx 社 Artix-7[13] とした．FPGA をターゲットとした回路の生成には表 2 に示す回路生成ツールを使用した．提案するＨＷレディーキュー及びその他の周辺回路は Verilog HDL で実装し，論理合成，配置配線は Linux マシン（CPU：Intel Core i7 38293．6GHz，メモリ：64GB，OS：CentOS 6.6）で行った．

また，提案手法であるＨＷレディーキューへの入出力を含めた SoC 全体の評価を行うために，ORPSoC の IP コアにHWレディーキューを接続し，FPGA 上に実装する評価を行った．IP コアは OpenCores が運営するプロジェクトのサイトより Verilog HDL で記述された RTL ソースを取得し，これに追加・修正をした．評価用ターゲットボードとして，ORPSoC のリファレンスプラットフォームである Digilent 社製 AtlysTM Spartan-6 FPGA 開発ボード [12] を使用した．

本評価で使用した Artix-7 と Spartan-6 の品番とその最大リソース数を表 3 に示す．回路規模の評価は表 3 の最大リソース数を上限とした使用率で示される．

３．２　ＨＷレディーキュー単体の回路規模の評価

図 5 にHWレディーキューの構成に必要なハードウェアリソースを示す．様々な規模の使用リソース数を検証するために，優先度ユニット数 P（＝ p ＋ 1）は 2 から 128 までの 2 の累乗数とした．各 P における優先度ユニット内のノード数 N（＝ n ＋ 1）は 2 から 512 までの 2 の累乗数とした．以降，便宜上ＨＷレディーキューの規模を｛P，N｝と定義する．各規模のＨＷレディーキューで管理できるタスク数は N×P である．

図 5 の縦軸は Artix-7 の最大リソースに対する使用率を示している．FPGA はスライスという単位で，回路構成を記憶する．一つのスライスの中には，FPGA 上で実装する回路のレジスタとして使用される複数の Slice Register と，組み合わせ回路を表現するために使用される，複数の Slice LUTs を備える．図 5 の横軸は P と N を組み合わせた構成を示している．図 5 には黒，灰色，白の 3 本の棒があり，それぞれ表 3 で示される Artix-7 の最大リソース数に対する Slice Register の使用率，Slice LUT の使用率，FPGA 全体のスライス占有率を示している．Slice Register の使用率は論理合成後の必要な記憶素子の規模，Slice LUT の使用率は論理合成後の組み合わせ回路に必要な回路規模，FPGA 全体のスライス占有率は配置配線後の回路全体の規模を示している．

図 5 のデータのない構成は回路量が Artix-7 の容量を超えたため，配置配線ができなかった構成である．Slice Register の使用率が Artix-7 にＨＷレディーキューが実装できる最大構成で 14% 程度あることから，提案手法の支配的な要因は Slice LUTs の使用率であることがわかる．Slice LUTs はマルチプレクサ, コンパレータ, デコーダなどの組み合わせ回路に用いられており，この回路規模が実装できるノード数を決定している．Artix-7 で実装できる最大のノード数は，{4，512}，{16，128}，{32，64}, {64, 32}, {128, 16} の 5 構成で 2, 048 ノードであっ

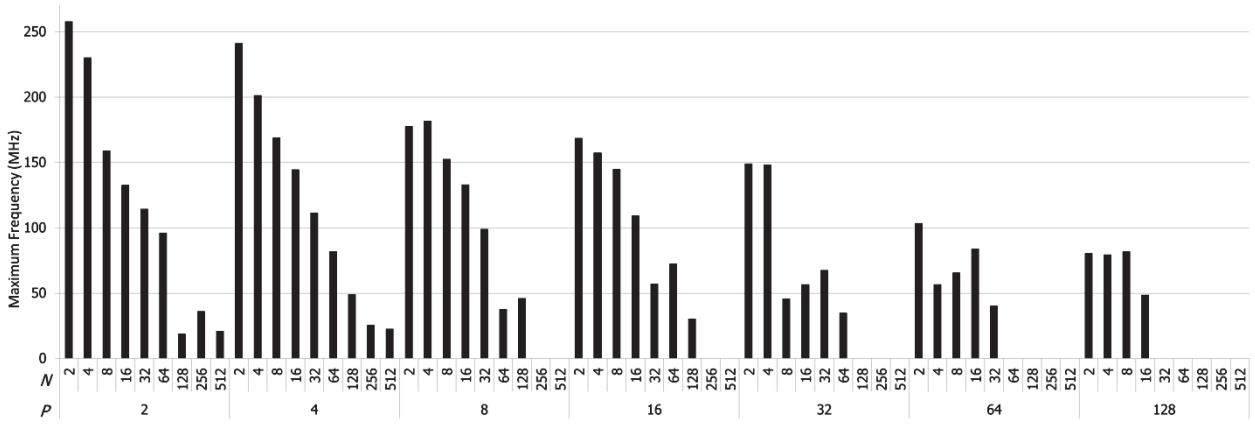

図6　HW レディーキューの最大動作周波数

た．それ以外の構成では 1,024 ノードが最大である．同一数のノードが実装できる構成間で比較すると，一つの優先度ユニット内に多くのノードをもつ構成の方が，回路規模が大きくなる傾向があることがわかる．これは削除に必要な起動要求回数を保存する処理のためのマルチプレクサが支配的であるためと考えられる．

本評価で使用した Artix-7 は FPGA の実装規模で約 22 万ロジックセルを備えるデバイスである．2016 年時点で最新型の FPGA である Xilinx 社の Virtex UltraScale VU440[14] はその 20 倍の回路規模である 440 万ロジックセルを備える．Virtex UltraScale VU440 でHWレディーキューを実装した場合，2,048 ノードの 20 倍である 40,960 ノード規模が実現可能である試算が立つ．40,960 ノードは，商用の RTOS である Spansion 社製 REALOS[17] の最大構成である，32,767 タスク，1,024 優先度の規模に相当することから，FPGA 上の実装で，大規模な組込みシステムの全てのタスクをHWレディーキューで処理することは，現実的に不可能ではないことが明らかとなった．

一方，ASIC による実装を考えた場合，Virtex UltraScale VU440 は ASIC のプロトタイピングデバイスとして，約 5，000 万 ASIC ゲート相当の回路を実装可能である [15]．現在の ASIC による LSI 設計では，45nm プロセスで配線可能なゲート数が 2 億ゲートに達している [16] ことから，提案するHWレディーキューは現在の高集積化技術を用いることで，ASIC 設計時の SoC 機能部品として今後，現実的なコストで実装可能になることが考察できる．

３．３　HWレディーキュー単体の遅延の評価

図 6 にHWレディーキューの最大動作周波数を示す．図 6 の縦軸は最大動作周波数 (MHz)，横軸は図 5 と同様に P と N を組み合わせた構成を示している．表 2 で示した配置配線ツールは発見的な手法で局所解を求めながら回路の最適化を行うため，遅延に関しては構成ごとに

表4　ORPSoC に128 ノードのHWレディーキューを接続した場合の回路規模と動作周波数

	ORPSoC	ORPSoC +提案手法	増加率（%）
Slice Registers 使用率（%）	12	15	20
Slice LUTs 使用率（%）	45	63	28.57
Occupied Slices 使用率（%）	64	81	20.98
動作周波数（MHz）	50	50	0

単純な増減を示さない．しかしながら，図 6 の構成全体を通して，各優先度ユニットのノード数が増加するほど遅延が大きくなる傾向にある．これは，図 5 のルックアップテーブルの使用率の増加傾向の原因と同一の原因であり，それにより，直列に接続する論理回路が構成される結果が増加するためである．

設計したHWレディーキューは 1 クロックサイクルで SELECT，REMOVE，INSERT の全ての操作の演算を完了させる．なお，SoC 実装時に，CPU などの動作周波数に合わせて，HWレディーキューのクロックサイクルを短くするために，組み合わせ回路にパイプラインレジスタを挿入し，パイプライン化することは可能であるが，タスクスケジューリング処理は時系列に前状態に依存してタスク状態の遷移をしなければならない制約があるため，タスクスケジューラの演算は並行実行することができない．このため，パイプライン化によってスループットは向上しない．つまり，RTOS は CPU の動作周波数に関わらず，この遅延時間を必ず待たなければならない．

評価した構成の中で，遅延が最も長い構成が 50ns(動作周波数 20MHz) 程度であった．現在の FPGA の現実的な動作周波数が最高で 100MHz(10ns の遅延) 程度であることを考慮すると，最も遅延が長い 20ns のHWレディーキューの構成を，例えば，100MHz で駆動する CPU で制御した場合，CPU はHWレディーキューの出力を得るために，現在の READY 状態のタスク数やレディーキューへの操作の種類 (SELECT，REMOVE，INSERT) に関係なく，5 クロックサイクルの一定の待ちが必要となる．

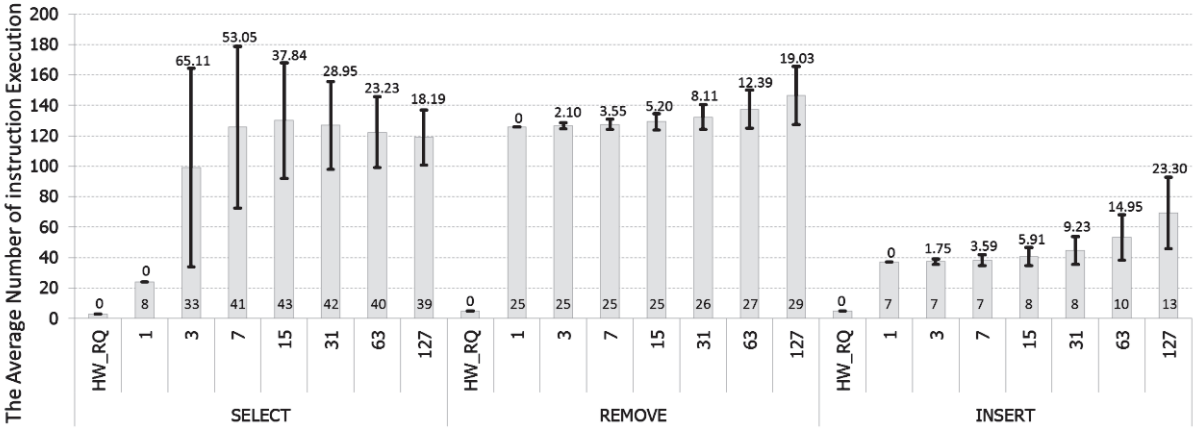

図7　ソフトウェア実装時の平均命令実行数と標準偏差

３．４　ＳｏＣに組み込んだ場合の回路規模と遅延の評価

SoC の機能部品として ORPSoC に 128 ノードを備えるＨＷレディーキューを接続した場合の，回路規模と動作周波数の変化を表 4 に示す．128 ノードは｛16, 8｝で構成される．評価は ORPSoC の実装ターゲットを AtlysTMSpartan-6FPGA 開発ボードとした場合の回路規模と動作周波数を示している．Spartan-6 に ORPSoC のみを実装した状態で，スライスの占有率が 64% に達していたため，残りの 36% で実装可能なＨＷレディーキューの構成を図 5 の結果より選択したことから，本評価では 128 ノードの実装となった．

３．５　ＨＷレディーキューの構造と実行時間平準化の関係

図 3 で示したＨＷレディーキューはすべてのデータパスがバス共有されずに，マルチプレクサにより送信先が制御されている．また，演算用の組み合わせ回路がすべてのノードに備えられ，演算が最大並列化されている．このハードウェア構成でのみ，マイクロアーキテクチャに依存するノード内データの待ち合わせ処理のための待機クロックサイクルがいかなる場合にも存在せず，レディーキュー内にあるタスクの数や種類にかかわらず，必ず 1 クロックサイクルですべてのキュー操作を完了することが可能になる．ＨＷレディーキューの備えるノード数に比例して，ハードウェア構成ごとに 1 クロックサイクルの時間は変化するが，ハードウェア構成はリアルタイム・システムにとって静的な構成要素である．このため，本論文が現在の LSI 製造技術を用いてはじめてスケーラブルにＨＷレディーキューが構成可能であることを示したことは，レディーキュー操作のための処理時間を完全に平準化し，OS のレディーキュー操作に限り最悪実行時間をシステム設計時に静的に決定可能であることを示したことになる．

また，ＨＷレディーキューにより平準化した 1 クロックサイクルの長さは，1k ノードあたり 30ns 程度である．これはシステム運用上，十分に現実的な演算時間であるうえ，論理的に必ずソフトウェア実行より高速に演算できる．この最悪実行時間でシステム実行全体の処理時間を見積もることは，実行時間過剰見積もりによる CPU 資源過多の設計の原因にならない．

３．６　ソフトウェア実行時の実行時間変動との比較

実システムで実行中の RTOS のレディーキュー操作に関する実行時間のみを切り出して計測することは非常に困難である．そのため，本研究ではソフトウェアで実装したレディーキューを or1ksim[18] を用いてシミュレートし，命令実行数レベルで実行時間を見積もる．or1ksim は ORPSoC 上で動作する Linux の動作確認ができる高精度なシミュレータであり，通常は ORPSoC 上で動作する OS やデバイスドライバの開発／デバッグに用いられるフリーソフトである．本研究ではこれを修正し，レディーキュー操作の命令数を計測する機能を追加した．

図 1 で示されるレディーキューのデータ構造に，初期状態でタスクを割り当て，C 言語で記述した SELECT, REMOVE，INSERT の操作を実行したときの命令実行数を計測する．コンパイル環境は OR1000 アーキテクチャ向けの GNU ツールチェインを用いた．コンパイラは gcc のバージョン 4．5．1 を用い，最適化オプションは -O2 を使用した．2n － 1 のタスク数を上限としたランダムな数のタスクを，ランダムな優先度を選びレディーキューに挿入した初期状態を 10,000 パターン生成し，その 10,000 パターンの初期状態に対し SELECT, REMOVE，INSERT の操作を行い，それに要する命令数を計測した．3．4 で示した 128 ノードのＨＷレディーキュー構成 (｛16, 8｝) と比較するため，レディーキューの優先度数は 16 とし，初期状態の生成時にランダムに選んだ優先度内に異なる 8 タスクが既に存在していた場合，当該タスクの挿入は無視されることとした．また，128 ノードを上限とする評価にするため，n は 1 ～ 7 の

整数の範囲内で計測を行った．

図 7 にソフトウェア実装時の平均命令実行数と標準偏差を示す．図 7 の横軸の数字は初期状態のタスク数の上限値を示しており，SELECT，REMOVE，INSERT の操作の分類ごとに分けられている．数字でない HW_RQ は HW レディーキューで実装した場合を示している．HW レディーキューでは初期状態のタスク数に関わらず命令実行数が変動しない．このため各操作に一つ存在する．図 7 の縦軸は平均命令実行数を示している．これは生成した初期状態の 10,000 パターンで操作を実行し，得られた命令実行数の平均値である．灰色の棒の根本の内部にある数字は HW_RQ を基準とした命令数の比を小数点以下で切り捨てて示している．例えば，この数字が 2 ならば，HWレディーキューの実装と比べてソフトウェア実装のレディーキューの応答速度が 2 倍遅いことを示す．棒の最上部の中央から上下方向に黒い線が引かれている．これは計測した命令実行数の標準偏差を示しており，棒の最上部から正方向と負方向に標準偏差の幅で線を引き，誤差として表現している．また，黒い線の最上部の数字は算出した標準偏差を小数点第三位以下で切り捨てて示している．この線の範囲が大きいほど実行時間にばらつきが多い傾向にあり，逆に小さいほど実行時間にばらつきが少ない傾向にある．

SELECT 操作では，タスク上限数が増加するほど実行時間のばらつきが小さくなる傾向が確認された．また，平均命令実行数は上限タスク数が 7 以上の場合に変動が少ないことがわかる．SELECT のアルゴリズムは次に実行すべきタスクが優先度 0(優先度最高) から選ばれる可能性が高いほど，平均命令実行数が変動しない特性をもつため, タスク上限数が大きいほどその確率があがる．SELECT 操作は他の操作 (REMOVE，INSERT) に比べて，ばらつきが大きい．これは次のタスクを選出する際に，完了タスクと同一優先度内に他のタスクがなければ，優先度を管理する配列テーブルをリニアサーチするため，このサーチ時間のばらつきがそのまま命令実行数に反映されている．このばらつきは優先度数が多くなるほど大きくなる．

REMOVE 操作では，タスク上限数が増加するほど平均命令実行数及びばらつきが大きくなる傾向が確認された．REMOVE 操作はタスク ID と優先度をキーとしたサーチを行って，該当するタスクをリストから除外する操作を行う．優先度が外部から指定されるため，サーチは優先度内のリスト要素から行う．タスク上限数が増加するほど，リスト内のタスクが増加する可能性があり，サーチに要する命令実行数がばらつく．このばらつきは，優先度内で管理されるリスト要素が増加するほど大きくなる．

INSERT 操作では，REMOVE 操作と同様に，タスク上限数が増加するほど平均命令実行数及びばらつきが大きくなる傾向が確認された．しかしながら，INSERT は REMOVE と異なり，末尾にリスト要素を追加するのみの単純なリスト操作であるため，スケジューラの応答速度が速い．起動要求回数を増加させるために，INSERT も REMOVE と同様に優先度内のリスト要素をサーチしなければならないことから，REMOVE と同様の命令実行数のばらつきが確認できる．

ＨＷレディーキューは，SELECT，REMOVE，INSERT 操作を行うにあたり，図 4 で示したソフトウェア制御のみが命令実行で必要となる．この制御プログラムには分岐命令が存在しないことから，命令実行数は変化しない．このため, 命令実行数の標準偏差は 0 である．

実行命令のばらつきに関して，特にＨＷレディーキューの効果が見込めるのは SELECT 操作である．最も大きい標準偏差は 65.11 を示している．また，一時的にスケジューラの管理するタスク数が少ない状況はシステムにとって十分に発生する可能性のある状況である上，SELECT の操作はタスクの完了時に必ず実行しなければならないことから, 実行頻度が高い．このことから，SELECT 操作の実行時間が平準化されることで，システム設計者は大きな恩恵を受けることになる．

また，ソフトウェア実装と比べてレディーキュー処理の応答速度は命令実行数レベルで 7 ～ 43 倍に高速化されている．これはレディーキューの処理を布線論理で最大並列化した恩恵である．加えて言えば，最大で 43 倍の高速化は過小評価である．キャッシュミスなどのマイクロアーキテクチャレベルのプロセッサのストール要因により，実際には CPI が 1 より大きくなるため，実システムの時間でスケジューラの応答速度を計測するならば，さらに大きい応答速度の差がある．

４．関連研究

本研究と異なるアプローチでスケジューリング処理を並列演算させる方式にマイクロプログラム方式がある [19]．これは OS の機能であるスケジューリング処理をマイクロプログラムとして抜き出し，メインの CPU とは別の MPU で並列に実行させることで，スケジューリングの演算時間を隠蔽する手法である．この手法は個々のタスクの実行が十分に長く，完全にスケジューリング時間が隠蔽された場合に，タスクスケジューリングのオーバヘッドが隠蔽され，本研究の目的とする平準化が結果的に達成されるが，もし, タスクの実行時間が短く，頻繁にタスク切り替えが発生する場合は，スケジューリング時間の平準化が保証されない．この点が大きく本研究と異なる．

KuacharoenらはFPGAを用いて実装するリアルタイム・システム向けのハードウェアスケジューラを提案した[2]．このハードウェアスケジューラは割り込みコントローラの拡張として布線論理で実装され，外部割り込

みを起点としたタスク生成に対応し，RM／EDF／優先度スケジューリングをFPGAの再構成機能を用いて選択的に利用できる．本研究のHWレディーキューはタスクスケジューラ全体ではなく，レディーキューのみのハードウェア化に焦点を当てていることが大きく異る．このため，本研究のHWレディーキューはタスクの起動が割り込み起点のみである制約が無い．また，実現可能性の検証の面で，Kuacharoenらは適用範囲を極めて少ないタスクセット(4〜64タスク程度)に限定していることに相違点がある．

仲野らは，μITRON の機能を VLSI 上に実装し，システムコール処理を高速化するシリコン TRON を提案した [20]．現在，μITRON 仕様が実装されている T-Kernel は本研究が対象とした静的優先度スケジューリングを採用しており，シリコン TRON の機能である STRON-I は本研究とアプローチが近い．また，STRON-I のレディーキュー操作と比べ，本研究は同様のハードウェア機能を備える．しかしながら，本研究は高速化ではなく，布線論理で実装することによる実行時間の平準化に着目して評価している点と，タスクセットの増加に伴う回路のスケーラビリティについて深く言及している点で異なる．シリコン TRON は約 1 万〜 2 万ゲート程度の回路規模の実現を前提としていたため，開発できるタスク数がハードウェアによって 1 桁程度に限定される上に，1 〜 4 クロックサイクルの間で，システムコール処理時間が変動した．これはシリコン TRON がリアルタイム性の向上を目的としたわけではなく，マルチメディア，通信制御，信号処理，ロボティクスなどの組込みシステム分野で求められる高速な応答時間の実現に応えるために設計さたためである．一方，1 億ゲートを超えるトランジスタを利用可能な現代の LSI 設計は十分な高速処理を実現可能としており，極めて小さいロードであれば，必ずリアルタイム性能を満たすことができる．現在のリアルタイム・システム設計者はシステムの機能面での価値を高めるため，与えられた計算資源でリアルタイム性を損なわないことを保証する範囲で，可能な限り多くのロードをシステムに実行させる．この場合，システムの実行時間見積もりの方法論が重要となる．本研究はこの見積に着目し，OS のレディーキュー処理の実行時間の平準化を行うことで，システム全体の実行時間見積もりの容易さの向上を実現している．加えて，平準化が守れる範囲のレディーキューのスケーラビリティを示している．

また，シリコン TRON は 20 年以上前の半導体製造技術を前提に議論がなされており，近年の半導体製造プロセスで実現可能性を議論したことが，本研究との本質的な相違点であると言える．一般的には，過去の半導体製造プロセス用に設計されていた同期回路を，半導体製造プロセスを新しく変えて同様に実装することはできない．利用する製造プロセスが変われば，そのデザインパラメータを利用した再設計が要求される．例えば，微細化による配線抵抗の増大は大きな設計上の問題となる．レイアウト時の配線長の制約が増えることから，リピータ挿入による回路規模増大も視野に入れて論理設計から行わなければならない．これは，シリコン TRON が研究されていた 20 年前には存在しないデザインパラメータであり，LSI を設計する場合は必ず各製造プロセスに最適化した回路設計が要求される．本研究は近年の製造プロセスで実装された FPGA を用いて，上記のデザインパラメータを考慮した上で，現代の LSI 製造プロセスをもってしてはじめて，図 3 で示したデータパスを共有せずに，シングルサイクルの最大並列化で処理を行うレディーキューのハードウェアがスケーラブルに実装可能であることを詳細な実装データとして示しており，このデータの資料的価値がシリコン TRON と対比した場合の本論文の貢献である．

５．おわりに

本研究は RTOS がタスクの静的優先度スケジューリングに用いるレディーキューを布線論理で実装することで，スケジューリング時間を平準化する手法を提案した．HWレディーキューは，組込み SoC の機能部品として扱われ，RTOS がスケジューリング処理に利用することにより，リアルタイム・システムの実行時間の見積もりを容易にする．

HWレディーキューの評価として，FPGA 上で実装した場合の回路規模と遅延を示し，高集積度を実現する現在の半導体製造プロセスで製造した場合の実現可能性を議論した．また，AtlysTM Spartan-6 FPGA 開発ボード上で動作する組込み SoC である ORPSoC の機能部品としてHWレディーキューを実装し，HWレディーキューによる組込み SoC 全体の性能の変化を計測した．さらに，提案手法の平準化の効果を示すために，ソフトウェアで実装したレディーキューをシミュレータ上で実行し，命令実行数レベルで実行時間のばらつきを標準偏差で示し，HWレディーキューが達成する実行時間の平準化効果を明確にした．また，同じ評価で，レディーキューの平均命令実行数を併せて示し，最大で 43 倍のレディーキューの応答速度の向上が確認されたことを示した．

今後の課題として，HWレディーキューのハード・リアルタイム・システムへの適用が考えられる．EDF や LLF などの，デッドライン時刻が明確に指定された場合の動的なスケジューリングアルゴリズムの布線論理化を含め，本研究の評価と同様に回路規模と性能の観点から，実現可能性を議論する予定である．

また，将来的に本研究の成果は，我々が現在研究を進めているハードウェアのみで実現する組込みシステム向けの仮想化環境のドメインスケジューラに応用される予定である．これは IoT 向けのネットワーク接続型の極小

デバイスで，軽量かつ応答性能・リアルタイム性能が高い仮想化環境を実現するためのスケジューラである．ドメインスケジューラはゲストOS間の資源管理，リアルタイム性を保持するための資源管理を行うことができることから，将来無数に存在することが予測できる極小IoTデバイスのソフトウェアシステムの管理をクラウドの特性をもつシステム管理として容易化できる特性と，リアルタイム性能の両立を可能にする．このドメインスケジューラはOSのタスクスケジューラと構造が酷似していることから，本研究のレディーキューのハードウェア実装におけるスケーラビリティの評価がそのまま適用できる．

謝辞

本研究は公益財団法人　服部報公会平成26年度工学研究奨励援助金研究課題名：「ハードウェアによる組込み仮想化のVM間リアルタイム・タスクスケジューラの開発」の助成により行われた．また，本研究の評価にあたり，福岡大学 佐藤寿倫 教授に貴重なご意見を頂いた．深く感謝する．

参考文献

[1] G.C.Buttazzo, “Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications, Third Edition”, Springer, ISBN 978-1-4614-0675-4, 2011.

[2] D.Tsafrir, “The Context-Switch Overhead Inflicted by Hardware Interrupts (and the Enigma of Do Nothing Loops)”, Proc. of the 2007 workshop on Experimental Computer Science, pp.1--12, 2007.

[3] Jan C.H., et.al. ,”A 14 nm SoC platform technology featuring 2nd generation Tri-Gate transistors, 70nm gate pitch, 52 nm metal pitch, and 0.0499 um2 SRAM cells, optimized for low power, high performance and high density SoC products”,2015 Symposium on Technology, pp.T12--T13, 2015.

[4] H. El-Aawar, “Increasing The Transistor Count by Constructing A Two-Layer Crystal Square on A Single Chip”, International Journal of Computer Science & Information Technology(IJCSIT), Vol. 7, No. 3, pp.97{105, 2015.

[5] R.S.Patti, “Three-Dimensional Integrated Circuits and the Future of System-on-Chip Designs”, Proc. of IEEE, Vol. 94, No. 6, 2006.

[6] T-Engine フォーラム(著), 坂村健(監修), “T-Kernel 標準ハンドブック“, ISBN 978-4893622297, パーソナルメディア株式会社，東京，Jun. 2005.

[7] 坂村健(監修),”μITRON4.0標準ガイドブック”, 社団法人トロン協会(編)，ISBN　978-4-89362-191-7, パーソナルメディア株式会社，東京，Nov. 2001.

[8] T. Takahashi, et.al., “Space Cube 2 - An Onboard Computer Based on Space Cube Architecture”, Proc. of International Space Wire Conference 2007, pp.65--68.

[9] D. Lampret, C. Chen, M. Mlinar, J. Rydberg, M. ZivAv, C. Ziomkowski, G. McGray, B. Gardner, R. Mathur and M. Bolado, OPENCORES.ORG, “OpenRISC 1000 Architecture Manual”, https://www.isy.liu.se/edu/kurs/TSEA44/OpenRISC/openrisc_arch3.pdf, 参照 Nov 20,2018.

[10] D. Lampret, OPENCORES.ORG, “OpenRISC 1200 IP Core Specification (Preliminary Draft)”, https://opencores.org/websvn/filedetails?repname=openrisc&path=%2Fopenrisc%2Ftrunk%2For1200%2Fdoc%2Fopenrisc1200_spec.pdf&rev=645, 参照 Nov 20, 2018.

[11] J. Baxter, OPENCORES.ORG, “ORPSoC User Guide, Issue 4 for ORPSoC”, http://www.rte.se/sites/default/files/Blog/Modesty/orpsoc.pdf, 参照 Nov 20, 2018.

[12] “AtlysTM Board Reference Manual”, Digilent, Inc, https://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPAtlys/documentation/Atlys_rm.pdf, 参照 Nov 20, 2018.

[13] Xilinx Inc., “Xilinx 7 Series FPGAs Overview, Product Speciation”, https://japan.xilinx.com/support/documentation/data_sheets/j_ds180_7Series_Overview.pdf, 参照 Nov 20, 2018.

[14] Xilinx Inc., “Xilinx UltraScale アーキテクチャおよび製品概要”, https://japan.xilinx.com/support/documentation/data_sheets/j_ds890-ultrascale-overview.pdf, 参照 Nov 20, 2018.

[15] Xilinx Inc., “Xilinx プレスリリース”, https://japan.xilinx.com/news/press/2015/shipped-first-4m-logic-cell-fpga.html, 参照 Nov 20, 2018.

[16] IBM Corp., “IBM Cu-45HP”, https://www.globalfoundries.com/sites/default/files/pb-cu-45hp-advanced-soi-based-design-system-for-high-performance-applications.pdf, 参照 Nov 20, 2018.

[17] “ITRON Newsletter No.25”, http://www.ertl.jp/ITRON/Newsletter-J/itronnews25.html 参照 Nov 20, 2018.

[18] J. Bennett, “Or1ksim User Guide - Issue 1 for Or1ksim 0.4.0-“, http://ryo-on.users.sourceforge.net/distrib/or1ksim-0.4.0.pdf, Embecosm Lmited, 参照 Nov 20, 2018.

[19] R. Chattergy, “Micro-programmed implementation of a scheduler”, Proc. of the 9th annual workshop on

Microprogramming (Micro 9), pp.15-19, Sep. 1976.
[20] 仲野巧，アンディウタマ，板橋光義，塩見彰睦，今井雅治，”リアルタイム OS の VLSI 化とその評価”, 信学論 (D)，vol.J78-D，no.8，pp.679-686，Aug. 1995.