RC-011

仮想キューによる高性能ハードウエア RTOS の実現

Exploiting Virtual Queue for Implementing a High Performance RTOS in Hardware

丸山　修孝† 石原　亨‡ 安浦　寛人‡
Naotaka Maruyama Tohru Ishihara Hiroto Yasuura

あらまし

ネットワーク処理のように極めて高いリアルタイム性を求められるアプリケーションでリアルタイム OS（RTOS）を利用すると性能が大幅に劣化する。これは全体の処理時間に対する RTOS 処理の割合が大きいためである。このような環境において、従来ソフトウエアで実現されていた RTOS をハードウエア化することにより大幅な性能改善が見込まれる。しかし、RTOS が使用する待ち行列をそのままハードウエア化すると大量のハードウエア資源を必要とする。本稿では、RTOS 内部の待ち行列を極めて効率良く実現する「仮想キュー」の概念を提案し、その有効性について述べる。

1. はじめに

近年組込系システムにおいて高い性能を必要とするアプリケーションが増えている。このような分野の一つとして、ネットワーク技術を挙げることができる。有線ネットワークでは 1Gbps の Ethernet がすでに一般的に使用されており、高性能サーバでは 10Gbps Ethernet に移ろうとしている。無線ネットワークでは、構内網において最大 600Mbps 可能な 802.11n 規格が拡大している。公衆網においても Wi-Max や LTE など数十 Mbps から 100Mbps を越える仕様が現れてきている。

上記のようにネットワークの高速化において PHY および MAC 部は着実に高速化されている。一方でネットワーク機能を実現するために必要なネットワークプロトコル、すなわち TCP/IP の高速化に関してはあまり重視されてこなかった。これは従来ネットワークがパソコン中心に実装されてきており、パソコンの CPU 性能が相対的に高かったためこの問題がクローズアップされなかったからである。しかしネットワークがコンシューマ機器に急速に拡大している。コンシューマ機器をはじめとする組込みシステムにおいて高速な TCP/IP を実現することは容易ではない。

たとえば動作クロック 50MHz の ARM9 で TCP/IP を実現した場合、最大スループットの実測値は高々11Mbpsであった。ARM のような組込み型プロセッサでは動作クロックはせいぜい 400MHz が上限であり、これは TCP/IP スループットに換算すると 88Mbps である。

ネットワークの高速化に対する要望に応える方法としては一般的には 2 つの方法がある。一つは高性能プロセッサを導入することであり、もう一つはアプリケーションの一部をハードウエア化することである。一つ目の方法には、2 つの問題点がある。一つは価格、もう一つは消費電力である。一般的に組み込みシステムはコンシューマ向け機器をはじめとし、価格設定の低いものが多い。このためたとえ性能が高くてもコストが高い場合導入できないことが多い。また多くのコンシューマー機器ではシステムの信頼性を高めるためできるだけ強制冷却システムを導入することは避けなければならない。さらに世界的な環境意識の高まりに呼応して、消費電力が高くなることに対する抵抗は高い。

2 つめの方法の課題は柔軟性の欠如である。組み込みシステムにマイクロプロセッサを用いる一つの理由は、バグ対応や機能拡張である。アプリケーションをハードウエア化することにより柔軟性が失われる。

筆者らは第 3 の方法として RTOS をハードウエア化する方法を提案した [14]。この方法は CPU の処理量が減るため、CPU への負荷が軽減されシステムの消費電力が大幅に低減される。また、アプリケーションをハードウエア化するのと異なり、アプリケーションの柔軟性が維持される。したがって、LSI 開発後のアプリケーションの変更、単一 LSI による複数アプリケーションへの対応、製品出荷後のバグ対応等が可能である。

一般に組み込みシステムにおける RTOS は多くのキューを必要とする。その数は数百、多い場合数千個にも昇る。本論文では少ない回路規模で多量のキューを実現できる「仮想キュー」という技術を提案している。さらに「仮想キュー」は 1クロック時間でオペレーションを完了する。

以下、2 章で筆者らの関連研究の一部を紹介し、3 章において従来技術の問題点を指摘、4 章で仮想キューについて詳細に説明し、5 章でこの性能について言及する。

2. ハードウエア RTOS

文献 [14] では、ハードウェア RTOS をベースとした ARTESSO (Advanced Real Time Embedded SiliconSystem Operator) というプロセッサを提案した。本節ではこの概要を簡単に述べる。

TCP/IP は一般にファームウエアで実現されているが、TCP/IP が実行されているときどのような処理が実行されているのか、CPU 占有率を測定した。結果は表 2-1 の「従来」に示すとおりである。同表に示すよう、純粋なプロトコル処理に割かれている時間は高々 10% 程度である。ARTESSO ではプロトコル以外の処理を CPU から切り離し、CPU がプロトコル処理に専念することができるようにした。これにより同表の「ARTESSO」で示すように、CPU は 9 割以上の時間をプロトコル処理に時間を費やすことができるようになった。このため、同じ CPU を使えば TCP/IP 性能は 10 倍以上になること

表 2-1 TCP/IP ファームウエアの分析

処理の種類	CPU 占有時間 (%)	
	従来	ARTESSO
プロトコル処理	10.5	92.7
RTOS	32.1	2.8
ヘッダ並び替え	15.3	0.0
TCP チェックサム	32.2	0.0
メモリコピー	9.9	4.5
合計	100.0	100.0

†カーネロンシリコン（株）
‡九州大学

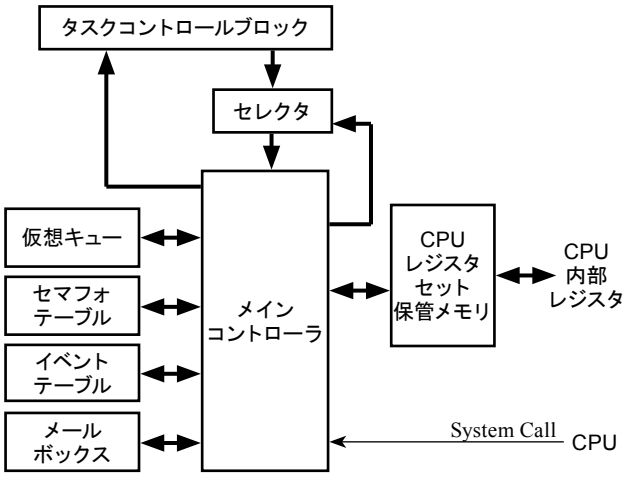

図 2-1　ARTESSO RTOS の内部構造

が確認できた。

CPU から切り離した処理のうち、ヘッダ並び替え、TCP チェックサム、メモリコピーの各処理は CPU 以外で実行することは容易である。しかし RTOS 処理のハードウエア化は困難である。ARTESSO は RTOS のスケジューラ部を完全にハードウエア化した。

図 2-1 は ARTESSO ハードウエア RTOS の構成図である。ARTESSO ハードウエア RTOS は ITRON コンパチブルで、組み込みシステムでよく使用される 30 のシステムコールを実装している。例えばフラグ待ち・フラグセット、セマフォ獲得・解放、メッセージ送信・受信、優先順位の変更、ウエイト状態からの解放などである。ARTESSO RTOS は 256 以上のタスク、イベント、セマフォ、メールボックスを実装することが可能であり、16 の優先順位が設定可能である。

CPU はシステムコール命令を実行すると「メインコントローラ」に対し、"System Call" 信号を発行し、ハードウエア RTOS に対しシステムコールの実行を要求する。「メインコントローラ」はステートマシンで構成され、全てのシステムコールの順序処理を行っている。「タスクコントロールブロック」は各タスクの状態、優先順位、タイマー値など、タスクごとに必要な情報がまとめられており、「メインコントローラ」はこの情報を選択、参照しながら順序処理を実行する。「CPU レジスタセット保管メモリ」にはタスクごとの CPU 内部レジスタセットが保管されており、タスクスイッチのたびに CPU 内部レジスタから実行中のタスクのレジスタセットを待避し、新しく実行するタスクのレジスタセットを CPU 内部レジスタにロードする。

ARTESSO ハードウエア RTOS により CPU は定型的な処理から解放され、プロトコル処理に専念することができるようになった。ARTESSO のメリットは、共通部分をハードウエア化し、アプリケーションをソフトウエアとして残すことにより、アプリケーションの柔軟性を確保したと言うことである。一方、ハードウエア TCP/IP オフロードエンジンはアプリケーションをハードウエア化している。したがって、アプリケーションのバグフィックスや機能拡張に対する柔軟性が極めて低い。ARTESOS では RTOS 上の処理はソフトウエアとして実現されている。したがって、バグフィックスや機能拡張に優れたプラットフォームを提供していると言うことが言える。

ROTS のハードウエア化には様々な課題がある。本論文では RTOS のハードウエア化に対する問題点について検討し、対策を示す。

3　RTOS ハードウエア化の課題

3.1　キュー構造の課題

過去において、RTOS の性能を向上させるために様々な研究がなされている。いくつかの研究では RTOS の機能をハードウエアとソフトウエアに分割し性能向上を図っている [8-13]。またいくつかの研究では大部分の RTOS 機能をハードウエア化している [1-7]。

RTOS をハードウエア化する上で大きな障害になるのが、キューの数である。実用的な RTOS においては各タスクが少なくとも STOP、RUN、READY、WAIT の 4 つの状態を有する。このなかで WAIT 状態にあるタスクは何かのトリガを待っている。たとえばタイムアウト、セマフォの解放、イベント通知、メールボックスへのメッセージの受信などである。同じトリガを待っているタスクが複数存在する場合、優先順位の高いタスクが選択され、このタスクは WAIT 状態から READY 状態に遷移する。優先順位が同じ場合は、先に WAIT 状態に入った方が選択される。この動作はプライオリティベースの FCFS(first-come, first-serve) と呼ばれており、各優先順位ごとにキューが必要になる。本研究で試作した ASIC は、セマフォ 32 個、イベント 32 個、メールボックス 192 個を実装している。優先順位の数は 16 であり、この場合必要とされるキューの数は (32 + 32 + 192) × 16 = 4,096 になる。これだけのキューをハードウエアで実現すると巨大な回路になる。

RTOS のハードウエア化は過去において RTU[1-3] や Silicon TRON [6] で事例が存在する。これらにおいてはキューの数は限定的である。たとえば Silicon TRON ではイベント識別子、セマフォ識別子共に 3 個しか実装されていない。一方、仮想キューは後述するように膨大な量のキューを少ないハードウエアで実現する。

3.2　キュー機能の課題

キューの基本機能は「入った順序で取り出すことができる機能」である。しかし RTOS で利用されるキューはこの基本機能だけでは不十分である。キューの途中から取り出す機能が必要である。

たとえば、あるタスクがあるセマフォを必要としたため「Get Semaphoe」というシステムコールを発行したとする。もしこのセマフォを現在他のタスクが使用している場合、システムコールを発行したタスクは、このセマフォの WAIT キューの最後部に接続される。RTOS ではトリガ待ちをする場合、タイムアウト値を設定することができる。もし WAIT キューに入っているタスクがタイムアウトすると、このタスクがキューの先頭でなくてもこのキューから取り出さなければならない。

また RTOS ではシステムコールにより直接キューからタスクを取り出すこともできる。例えば "Release Wait" は Wait 中のタスクを指定しキューから取り出すシステムコールであるし、"Terminate Task" は指定したタスクを強制的に終了するシステムコールである。

上述のように RTOS ではキューの途中からある指定したタスクを探し出し、取り出す機能が必要である。「検索・取り出し」機能は RTOS 内部では頻繁に利用され、かつ重要な機能である。

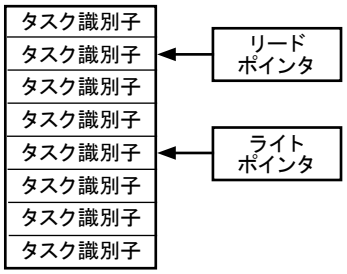

図 3-1 ハードウエア FIFO の構造

RTOS をハードウエア化しようと考えたとき、当然キューシステムもハードウエア化することを検討するが、一般的にキューをハードウエア化しようとしたときに検討されるのが、ハードウエアの FIFO である。図 3-1 にハードウエア FIFO の構造を示す。キューからの読み出しはリードポインタにしたがって読み出される。「検索・取り出し」機能をハードウエア FIFO に実装しようとすると、ライトポインタとリードポインタの間を検索し、目的とするタスクを探し出し、それを消去する機能が必要になる。また消去後、データの前詰め作業を行う必要がある。このような機能を全ての FIFO に持たせることはハードウエア量をさらに増加させる原因となる。またもしこの機能を実装したとしても、「検索・取り出し」処理に多くの時間を消費することになり RTOS の性能に大きな影響を与える。

3.3 キュー性能の課題

従来のソフトウエア RTOS はリスト構造のキューを使っている。これを図 3-2 に示す。キューへの書き込みはこのリストの最後部にエレメントを付け加えることにより行われる。同様にキューからの読み出しは先頭のエレメントを取り外すことにより達成される。これらの処理はソフトウエアで行われ、処理に多くの時間を消費している。

一番時間を消費するのは、「検索・取り出し」機能である。たとえばタスク 7 を取り出そうとしたとき、ソフトウエアはこのリスト構造に沿ってタスク 7 を探し出し、そしてポインタを書き換えることによってタスク 7 をキューから外す。このような処理は多くの時間を消費しするだけでなく、キューの長さにより処理時間が大きく変化し、リアルタイム性の高い処理には影響を与える。仮想キューでは後述するようにこの問題も解決している。

4. 仮想キュー

仮想キューは従来になかった全く新しい概念である。まず従来の FIFO の構造と仮想キューの構造を比較することにより少ない回路規模で構成できることを説明する。図 3-3 は従来のハードウエア FIFO を使った RTOS のキューイングシステムの構造を示している。この例の RTOS は、タスクの数が 8 個、セマフォが 8 個、タスクの優先順位は 4 つであると仮定する。

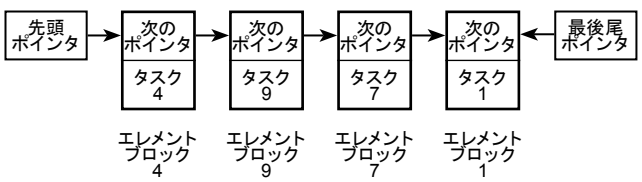

図 3-2 ソフトウエアでのキューの実装方法

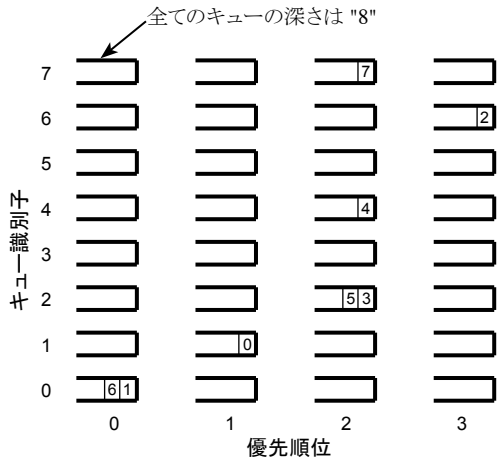

図 3-3　従来のハードウエア FIFO の Wait キュー

したがってセマフォ待ちの FIFO の数は 8 × 4 で 32 個必要である。また各 FIFO の深さはタスクの最大数、すなわち 8 必要である。各タスクはセマフォを獲得するために図のようにキューの中に入っていく。タスクの数は 8 個しかないため図のようにキューは常に閑散としている。一方、一つのキューに待ちが集中することも考えられ、キューごとに最大数のタスクが入っても問題ないよう、最大数の深さを用意しておく必要がある。つまり蓄積部分の最大使用率は 1 / 32 になる。これはタスクの数には関係がない。一般的な言い方をすると、RTOS で n 個のキューが存在すれば、FIFO の蓄積部分の最大使用率は 1 / n になり、極めて使用効率が悪いことがわかる。

前記のように、本研究において試作した ASIC では 4096 個のキューが内在する。したがってもし従来の FIFO で構成したとするとこの蓄積部分の最大使用率は 1 / 4096 となり、ほとんどが使用されていないということが言える。

図 3-4 は仮想キューのデータ構造を示している。従来のキューイングシステムでは、各キュー毎にタスクを管理していたが、仮想キューでは逆に各タスク毎にキューを管理する。すなわちタスク識別子毎に、現在属している「キュー識別子」、「優先順位」、「キューに入った順序」を管理する。維持するデータはこれだけであるため、図 3-3 に示す従来の FIFO に比べ無駄な蓄積領域を確保する必要が無い。また従来の FIFO 型ではキュー識別子の数が 2 倍になると必要になるリソースも 2 倍になったが、仮想キューではキュー識別子が 2 倍になってもキュー識別子を維持するメモリエリアが 1 ビット増えるだけである。したがってキューの数が大きくなればなるほど効果は大きい。

図 3-5 は仮想キューシステムの構成図である。仮想キューは「キュー制御部」、「キュー制御レジスタ部」、「タスク選択部」の 3 つの部分から構成されている。「キュー制御レジスタ部」は図 3-4 で示したタスク毎のキュー管理情報を記憶してるレジスタである。「タスク選択部」は指定されたキュー識別子

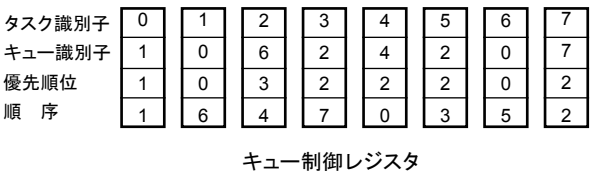

図 3-4　仮想キューのデータ構造

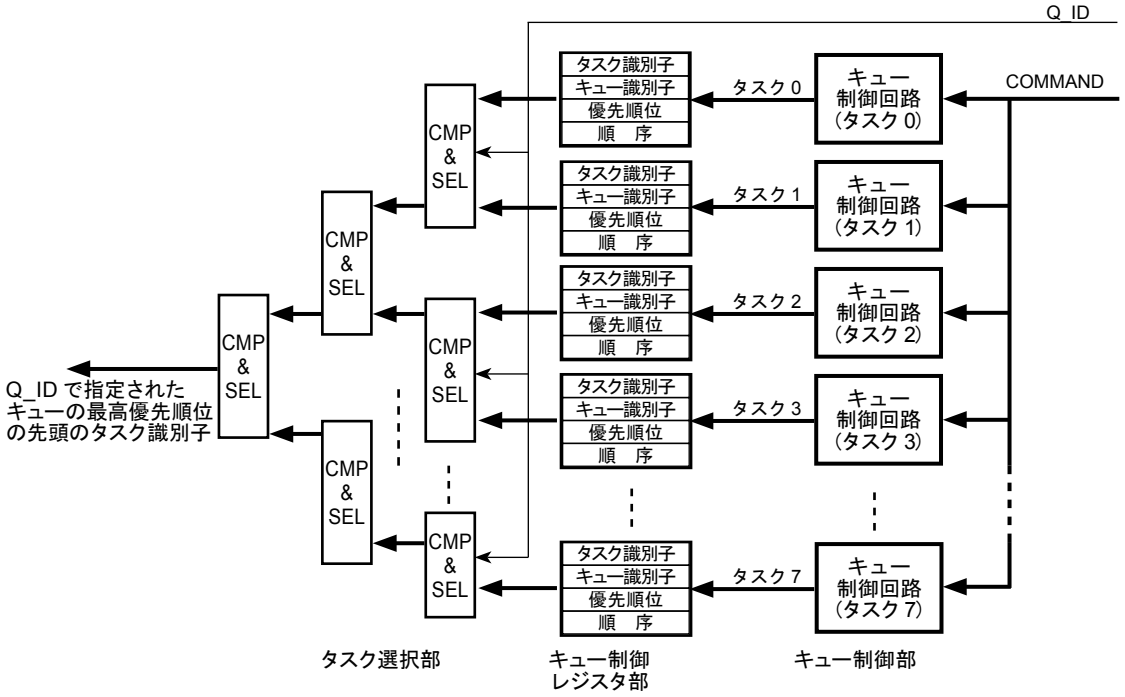

図　3-5　仮想キューの構造

に属するタスクを FCFS にしたがって選択し出力する。キュー識別子の指定は "Q_ID" 信号によってメインコントローラから指定される。「タスク選択部」は複数の "CMP & SEL" モジュールが図のように接続されている。この接続によりトーナメント回路を実現している。各 "CMP & SEL" モジュールでは優先順位の高い方のタスク識別子が後段に送られる。優先順位が同じ場合は「キューに入った順序」が先であるタスクが選択され、後段に送られる。この結果最終段から得られるタスク識別子はプライオリティベース FCFS に基づいている。

図 3-6 および表 3-1 はキューの一例およびそのときの各レジスタの内容を示している。簡単のため、この例ではキュー識別子が 2 つ、優先順位が 2 の例を示している。優先順位は 0 が最優先であると仮定する。このレジスタの内容が "CMP & SEL" モジュールに入力されると、Q_ID=0 と指定したとき最終段から現れるのは「キュー識別子レジスタ」が 0 で、「最優先順位レジスタ」が 0、かつ「順序レジスタ」が最小のもの、すなわちタスク 7 が現れる。Q_ID=1 と指定したときは同様にタスク 3 が最終段から現れる。

図 3-7 及び表 3-2 はタスクの検索・取り出し処理を行ったときのレジスタの変化を示している。この例ではタスク２をキューから取り外している。まずタスク２の「順序レジスタ」と「キュー識別子レジスタ」の値を「未使用」に設定し、同時に４より大きな値を持つ順序レジスタの値を1減算する。以上でタスク２の検索・取り出し処理は終了する。レジスタの書き換えはキュー制御回路が各タスク後と独立に行うため、この処理は 1 クロック時間で完了する。「エンキュー」、「デキュー」もこれらのレジスタを書き換えることにより同様に行うことができる。これらの処理も 1 クロック時間で完了する。

表　3-1　各レジスタの設定

タスク識別子レジスタ	順序レジスタ	キュー識別子レジスタ	優先順位レジスタ
0	4	1	0
1	1	0	0
2	3	0	1
3	6	1	0
4	Non	Non	0
5	0	0	1
6	5	0	1
7	2	0	0

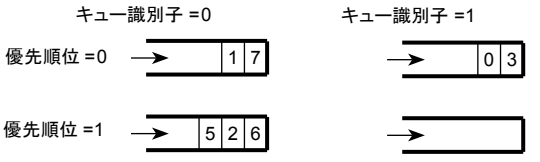

図　3-6　キューの一例

表　3-2　検索・取り出し時の動作

タスク識別子レジスタ	順序レジスタ	キュー識別子レジスタ	優先順位レジスタ
0	5 → 4	1	0
1	2	0	0
2	4 → Non	0 → Non	1
3	7 → 6	1	0
4	0	1	0
5	1	0	1
6	6 → 5	0	1
7	3	0	0

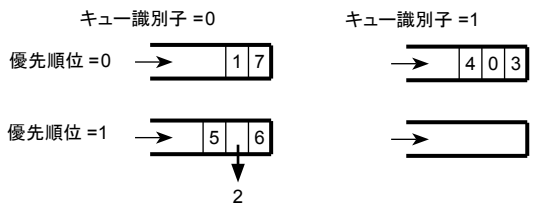

図　3-7　タスクの取り出し

5. 評価

5.1 仮想キュー

本節では仮想キューのゲート数について言及する。ARTESSO は 90nm プロセスで ASIC 化した。この機能仕様は以下の通りである。

- ARTESSO
  - CPU・・・・32bit RISC
  - RTOS
    - タスク数 32
    - イベントフラグ 32
    - セマフォ 32
    - メールボックス 192
    - 優先順位 16
- インターフェース
  - USB 2.0
  - Giga bit Ethernet MAC
  - PCI / PCI Express
- メモリ
  - 1MByte

上記仕様からこの RTOS に実装されている待ち行列の数は

$$(32 + 32 + 192) \times 16 = 4{,}096 \text{ 個}$$

になる。

まず、これだけの数のキューをハードウエア図 3-3 に示す FIFO で実現したらどのくらいのハードウエア規模になるかを算出してみる。タスク数が 32 であるから FIFO の幅は 5 ビット、深さは 32、すなわち 160 ビットのメモリが必要である。このメモリをフリップフロップで実現したとすると概算で 960 ゲートになる。FIFO にはライトポインタとリードポインタ、それにエンプティ、フル、等を検出する回路が必要であり、これらに約 190 ゲート必要である。従って FIFO 一個あたり 1,150 ゲートの回路となり、上記のように 4,096 個のキューが必要であればその回路規模は

$$4{,}096 \times 1{,}150 = 4{,}710{,}400 \text{ ゲート}$$

となる。これは面積で表すと、$13{,}293{,}000 \mu m^2$ (90nm プロセス) である。

試作した ASIC の各モジュールの面積の詳細を表 4-1 に示す。この表にあるよう、仮想キューの面積は $30{,}213 \mu m^2$ である。これは FIFO での実現に比べ 440 分の 1 の面積である。これによりハードウエア量として仮想キューは圧倒的に有利であることが実証された。また RTOS 全体でも $540{,}170 \mu m^2$ であり、大変コンパクトな回路で実現できていることが確認できる。以下、5.2 および 5.3 において仮想キューを使った ARTESSO の性能について記述する。

5.2 RTOS 性能

本節では ARTESSO ハードウエア RTOS の処理時間について言及する。ARTESSO ハードウエア RTOS の性能は Verilog™ シミュレータにより各システムコールに費やされるクロックサイクル数を求めた。一方従来の RTOS として NORTi を選択した。NORTi は ITRON 仕様の商用 RTOS であり、RTOS カーネルと TCP/IP モジュールを含んでいる。NORTi を ARM926 評価ボード上で走らせ、各システムコールに要するクロックサイクル数を測定した。

表 4-2 にこの結果を示す。キュー操作の欄は、"W" はキュー書き込み操作、"R" はキュー読み出し操作、"S" は検索・取り出し操作を示している。先にも述べたように、ARTESSO RTOS のキュー操作は全て 1 クロック時間で完了する。たとえば "Terminate Task" システムコールにおいて、ソフトウエアの RTOS の処理時間は取り出すタスクのキュー内の位置によって処理時間が大きく変わってくる。しかし ARTESSO の場合、このシステムコールの処理時間は必ず 6 クロックで完了する。ARTESSO RTOS の場合、キュー操作がシステムコール処理の負荷になっていないことがこの表からも確認することができる。

表 4-1 ASIC の各要素の面積

	Area (μm^2)
全面積	27,601,284
命令メモリ , ワークメモリ , バッファメモリ	14,851,332
ARTESSO	2,919,025
RTOS	540,170
Logic	372,964
仮想キュー	30,213
その他	342,751
RAM	167,206
その他 (CPU, ヘッダ並び替え , バスアービター / スイッチ , DMA, 暗号エンジン , etc.)	2,378,855
その他 (Ethernet, USB, バスインターフェース)	9,830,927

5.3 TCP/IP 性能

5.1 節で述べたように「仮想キュー」を導入することにより、極めて高性能かつ少ない回路量でハードウエア RTOS を実現することができた。[14] ですでに詳細に説明したが、最後に ARTESSO ハードウエア RTOS を利用した TCP/IP 性能の改善について簡単に言及する。

前記 90nm プロセスで試作した ASIC で TCP/IP スループットを計測した。この ASIC は最大動作周波数 150MHz で、このとき 515Mbps の TCP/IP スループット達成を測定により確

表 4-2 RTOS 性能比較

システムコール	ディスパッチ	キュー操作	NORTi (ARM 926)	ARTESSO RTOS
Put task to sleep	有り	W, R	628	10
Wake up a task	有り	W, R	496	10
Lock CPU	無し	-	120	6
Un Lock CPU	無し	-	232	6
Disable Dispatch	無し	-	214	6
Enable Dispatch	有り	W, R	580	9
Terminate Task	無し	S	368	6
Release Wait	有り	S, W, R	494	10
Change Priority	有り	W, R	541	11
Receive from Mailbox	無し	-	224	7
Receive from Mailbox	有り	W, R	591	11
Send to Mailbox	無し	R	360	8
Send to Mailbox	有り	R, W, R	541	11
Obtain Semaphore	無し	-	216	6
Obtain Semaphore	有り	W, R	558	9
Release Semaphore	無し	R	344	7
Release Semaphore	有り	R, W, R	536	11

Note : "-" は未実装またはデータ無し

認した。また消費電力は約 300mW であった。一方 ARM の場合、1.で示したとおり 50MHz で 11Mbps であるから、150MHz 換算では 33Mbps であり、圧倒的な性能改善があったことが確認できた。

6. 結論

TCP/IP のようなネットワークプロトコル処理は RTOS の処理のために多くの CPU 時間を占有する。したがって RTOS のハードウエア化等により TCP/IP 性能を大幅に向上させることができる。

一方で RTOS のハードウエア化には多くのキューを実装しなければならないという壁がある。「仮想キュー」という新しい発想を導入することにより大量のキューを少ないハードウエアで実現することができた。これにより従来のソフトウエアと同等の機能を有する RTOS をハードウエアで実現することができた。

ARTESSO RTOS が極めてリアルタイム性が高いマルチタスク環境を構築可能であることから、今後 ARTESSO のアプリケーションをネットワーク処理以外、たとえばロボットや自動車等の制御への適用を検討してゆく。

7. 参考文献

[1] Lindh, L.,“Fastchart-a fast time deterministic CPU and hardware basedreal-time-kernel,” in Proc. of Euromicro Workshop on Real Time Systems, pp.36-40, Jun, 1991

[2] Lindh, L. Starner, J. Furunas, J. ,“From single to multiprocessor real-time kernels in hardware,” in Proc. of the Real-Time Technology and Applications Symposium, pp. 42-43, May 1995.

[3] Adomat, J.; Furunas, J.; Lindh, L.; Starner, J.“Real-time kernel in hardware RTU: a step towards deterministic andhigh-performance real-time systems,” in proc. of the 8th Euromicro Workshop, pp.164-168, Jun 1996

[4] Nordstrom, S. Lindh, L. Johansson, L. Skoglund, T.,“Application specific real-time microkernel in hardware,” in Proc. of Real Time Conference, 2005.

[5] Samuelsson, T. Åkerholm, M. Nygren, P. Johan Stärner, J. Lindh, L. “A Comparison of Multiprocessor Real-Time Operating Systems Implemented in Hardware and Software,” in proc. of Int’l Workshop on Advanced Real-Time Operating System Services (ARTOSS’03), 2003.

[6] Nakano T., Utama A., Itabashi M., Shiomi A., and Imai M.,“Hardware implementation of a real-time operating system,” in Proc. of 12th TRON Project International Symposium (TRON’95), pp. 34–42, 1995

[7] Kohout, P. Ganesh, B. Jacob, B.“Hardware support for real-time operating systems,” in Proc. of the 1st International Conference on Hardware/Software Codesign and System Synthesis, pp.45-51, Oct. 2003

[8] Chandra, S. Regazzoni, F. Lajolo, M.,“Hardware/software partitioning of operating systems: a behavioral synthesis approach,” in Proc. of the 16th ACM Great Lakes symposium on VLSI, pp. 324-329, 2006

[9] Parisoto, A. Souza, A., Jr. Carro, L. Pontremoli, M. Pereira, C. Suzim, A.,“F-Timer: dedicated FPGA to real-time systems design support,” in Proc. of 9th Euromicro Workshop on Real-Time Systems, pp. 35-40, 1997.

[10] V.Mooney III, J. Lee, A. Daleby, K. Ingstrom, T. Klevin, and L. Lindth,,“A comparison of the RTU hardware RTOS with a hardware/software RTOS,” in Proc. of Design Automation Conference (DAC’03), 2003, pp. 683–688.

[11] Mooney, V.J., III Blough, D.M.“A hardware-software real-time operating system framework for SoCs,” IEEE Design & Test of Computers, pp. 44 – 51, 2002

[12] V. Mooney III.,“Hardware/software partitioning of operating systems,” in Proc. of Design, Automation and Test in Europe Conference (DATE’03), 2003, pp. 338–339.

[13] TRON ASSOCIATION,“ μITRON4.0 Specification,” 1999.

[14] 丸山 , 石原 , 安浦 : "RTOS のハードウエア化によるソフトウエアベース TCP/IP 処理の高速化と低消費電力化 ", 第 23 回 回路とシステム軽井沢ワークショップ, pp. 370 - 375 (2010)