論文

組込み制御システム向けリアルタイムOSのハードウェア化

森　久直*1)　坂巻佳壽美*1)　重松宏志*2)

Hardware implementation of a real-time operating system for embedded control systems

Hisanao MORI, Kazumi SAKAMAKI and Hiroshi SHIGEMATSU

Abstract　A real-time operating system is indispensable to present embedded devices from the point of view of the time limitations and portability of an application program. But, application programs change take on a large-scale factor and complexity when making embedded devices high performance oriented. And, the occupation time of a real-time operating system in an application program shows that problems increase. This problem invites a situation wherein the operation of a task cannot be ended within the limited time. And so, a function with high use frequency in a real-time operating system(μ ITRON) was made in combination with hardware on the assumption of recycling an existing application program. A real-time operating system made hardware was designed by VHDL, and implemented on FPGA. As a result, the processing time of the real-time operating system was shortened, and the problem of overhead was solved.

Keywords　Real-time operating system, Hardware design, μ ITRON, VHDL, FPGA

１．はじめに

現在の組込み機器は,IT 関連機器等に見られるように,アプリケーション・プログラムの移植性・拡張性等から,組込み制御用 OS であるリアルタイム OS が必須になっている。しかし,組込み機器の高機能化により，アプリケーション・プログラムが急速に大規模化・複雑化し，リアルタイム OS のオーバーヘッドが増加する問題がある。

この問題に対して,リアルタイム OS の機能の一部をハードウェアとして LSI に実装する研究や，リアルタイム OS の機能の中で必要な部分をマイクロプロセッサ(MPU)の機能の一部として実現する研究[1]が行われている。従来は,リアルタイム OS の機能の一部を,ディジタル回路の書き換えが不可能な LSI として開発され,後からの機能変更が困難である。

そこで,上記の問題を解決するために，リアルタイム OS の機能の一部をディジタル回路の書き換えが可能な FPGA を用いてハードウェア化した。その結果,リアルタイム OS のオーバーヘッドを削減することができた。更に,FPGA ベースとすることで,この FPGA が製品に組込みこまれた状態でも,インターフェースとリアルタイム OS の仕様を柔軟に変更でき,既存 MPU に対応できる。しかし，前者はディジタル回路の書き換えが不可能な LSI を開発するものであり,機能変更が困難である。後者は既存 MPU が使用不能である。

*1) 情報科学グループ　*2) エレクトロニクスグループ

２．リアルタイム OS の構成

μ ITRON[2,3]は,組込みシステムに実装されるリアルタイム OS のうち約 40%を占めている[4]。このため,μ ITRON に準拠したリアルタイム OS を開発する。以下に,リアルタイム OS の構成における基本概要を述べる。

(1)タスクの状態

μ ITRON の仕様書に準拠するために,タスクの状態は READY（実行可能状態）,RUN（実行状態）,WAIT（待ち状態）が必須となっている。

(2)オブジェクト

基本となるオブジェクトは，タスク，イベントフラグ,セマフォである。オブジェクトとしてのタスクは,タスク・コントロール・ブロック（TCB）のことであり,タスクを管理する情報を保存する。イベントフラグは,タスク間で同期をとるためのものである。そして,セマフォはシリアルポートなどのリソースの排他制御をするためのものである。

(3)割込みハンドラ

割込みハンドラは,非タスクであり,処理中はタスクの切り替えがロックされる。

(4)システムコール

システムコールは,タスクやセマフォ，イベントフラグを操作する機能を呼び出すための機構である。

(5)スケジューラ

スケジューラは，タスクのステータスから，タスクの実行順序を決定する。フラグやセマフォを操作するような使用頻度の高いシステムコールは，例外を除いてスケジューラの処理を伴う。スケジューラは，リアルタイム OS の重要な機能である。

３．ハードウェア化における設計方針

本研究では,リアルタイム OS のハードウェア化にあたって,小規模のアプリケーション・プログラムを想定し,リアルタイム OS の仕様を必要最低限のものとする（表１）。更に，タスクを停止させるための DORMANT（休止状態）をタスクの状態に加える。DORMANT では,タスクからハードウェア資源を全て解放したり,レジスタやプログラムカウンタなどの初期化を行う。

表１　開発するリアルタイム OS の仕様

項目	数	性能
タスク	8	優先順位 8 レベル
イベントフラグ	8	16 ビットフラグ
セマフォ	8	16 ビット計数型
割込みハンドラ（外部割込み）	8	優先順位 8 レベル
システムコール	24	－(注)

(注)システムコールは以下の通り。

●タスク管理機能
dis_dsp, ena_dsp, ext_tsk, get_tid, sta_tsk, ter_tsk, chg_pri, rel_wai, cre_tsk, ref_tsk

●付属同期機能
slp_tsk, wup_tsk, can_wup

●同期・通信機能
sig_sem, wai_sem, set_flg, clr_flg, wai_flg, cre_sem, ref_flg, ref_sem

●割込み管理機能
loc_cpu, unl_cpu

●時間管理機能
dly_tsk

リアルタイム OS のオーバーヘッドを削減するためには,従来のソフトウェアによる順次処理をハードウェアに置き換えるだけではなく,最小限のクロック内で処理が完了できるようにする。そのためには,可能な限り,回路を順序回路ではなく,組み合わせ回路で構成する。また,MPU とリアルタイム OS は異なるリソースであるため,並列動作させる。組み合わせ回路は,接続する基本ロジックの長さによって処理時間が決まるため,クロック同期で動作する順序回路に比べて高速である。しかし，リアルタイム OS の全ての機能をハードウェア化すると，既存の MPU との接続が困難になる。そこで,リアルタイム OS の一部をハードウェア化する際に,ハードウェア部とソフトウェア部に分ける（図 1）。ソフトウェア部は,(Ⅰ)インターフェース処理機能,(Ⅱ)タスク切り替え処理機能で構成し、ハードウェア部は,(Ⅰ)マルチタスク制御機能,(Ⅱ)タスク・スケジューリング機能,(Ⅲ)割込み処理機能,(Ⅳ)システムコール機能等で構成する。

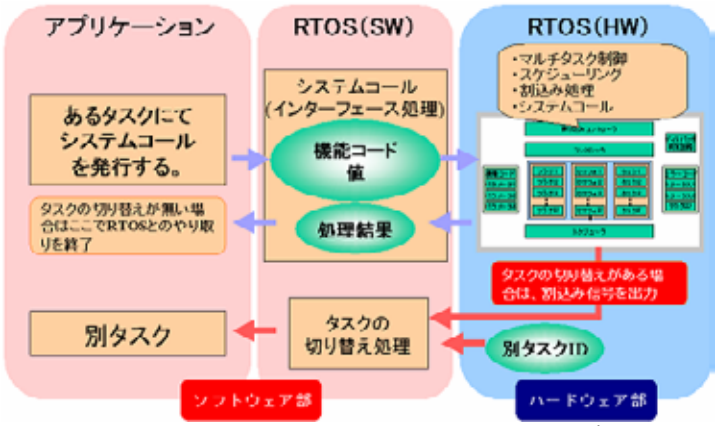

図１　開発するリアルタイム OS のシステム構成

インターフェース処理機能は,システムコールの処理を識別する機能コードと必要な値をリアルタイム OS（ハードウェア部）に渡す部分，リアルタイム OS（ハードウェア部）における処理結果を取得する部分をもつ。このインターフェース処理機能を柔軟にするために,ハードウェア化するリアルタイム OS 内部に入出力レジスタを設置し,この入出力レジスタへのアクセスをアドレスバス,データバス,コントロールバスの 3 つのバスで行う。

以上の設計方針で，オーバーヘッドを削減し,柔軟なインターフェース処理機能をもったリアルタイム OS を開発する。

４．リアルタイム OS のハードウェア部

4.1　ハードウェア部の内部構成

開発するリアルタイム OS(ハードウェア部)の内部構成は,図 2 に示すように，スケジューラ,フラグ,セマフォ,TCB，割込みコントローラ,入出力レジスタと,これらの制御に必要なコントローラで構成する。入出力レジスタにはアドレスの 0 から 7 を割当てて,メモリマップド I/O 方式にし,既存の MPU との接続を可能にする。

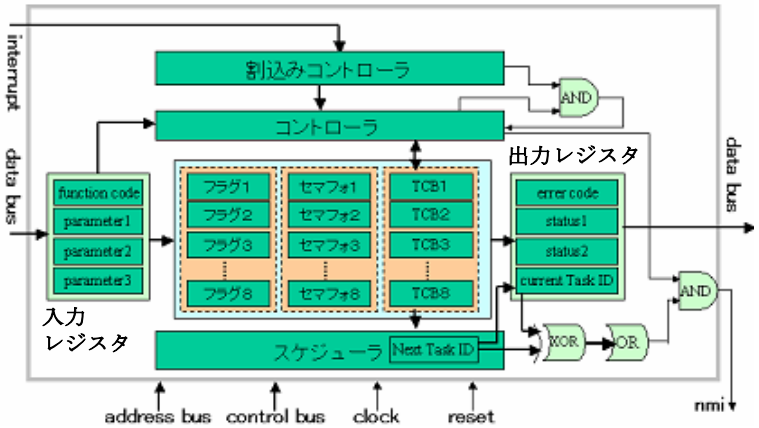

図２　リアルタイム OS（ハードウェア部）の内部構成

4.2　スケジューラ

スケジューラの回路を図３に示す。スケジューラの高速処理を実現するために，①組み合わせ回路のみで設計する，②１つの優先度に１つのタスクをつなげる制約をつける，という２つの条件で設計を行う。この条件で，優先度に基づいてタスクの実行順序を決める方式のスケジューラを設計する。同一優先度のタスクを実装する場合には，ソフトウェアで対応可能である。スケジューラは，常にTCB の PRI（優先度値）,RDY（実行可能フラグ）,ID（タスク ID コード）と接続し,これらの値に変化があった場合に即時処理を行い,次に実行するタスク ID コード(Next Task ID)を出力する。

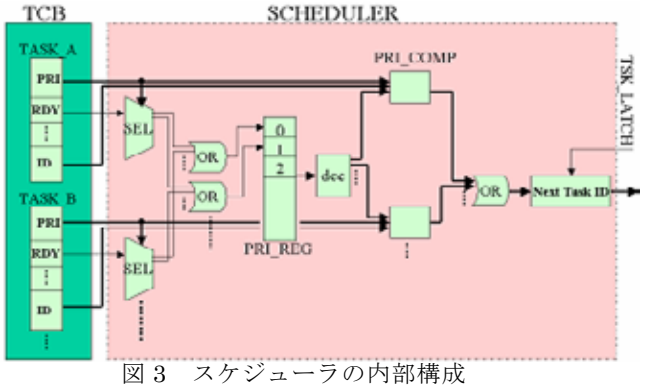

図３　スケジューラの内部構成

4.3　フラグ

フラグの回路を図４に示す。フラグの内部に,フラグの wait パターンと set パターンを保存するレジスタ,待ち条件を保存するレジスタ,その他の組み合わせ回路を基本ユニットとし,このユニットをフラグオブジェクトの数だけ設置する。parameter1 からは使用するフラグオブジェクトを識別する ID コードを入力し,parameter2 からは wait もしくは set フラグパターンを入力する。parameter3 からは,フラグパターンの待ち条件と set パターンの初期化の有無を設定するコードを入力する。

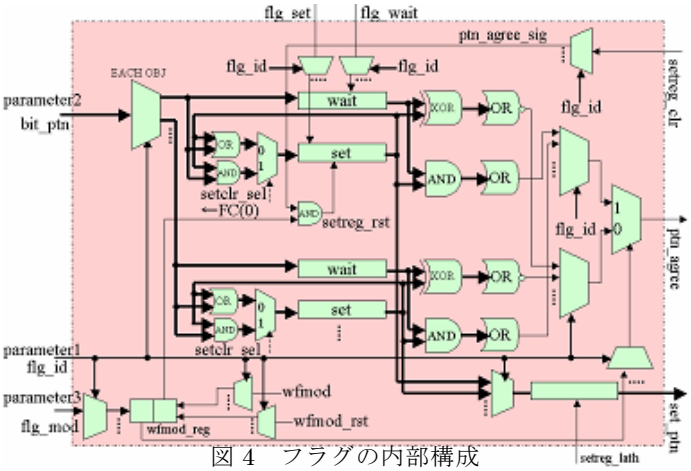

図４　フラグの内部構成

フラグに関連したシステムコールの処理を高速化するために，wait パターンと set パターンを設定した後の処理は組み合わせ回路で行う。これにより,システムコールを発行した後は,すぐに処理結果を出すことができる。更に高速化を実現するために,このフラグに対する複数タスクの待ち行列を処理する機能を外した。その理由は,例えばタスクが８つ定義されている場合に,８つのフラグを定義することで対応できるからである。

4.4　セマフォ

セマフォの回路を図５に示す。セマフォの内部には,アップ/ダウン･カウンタと,FIFO,その他の組み合わせロジックを基本ユニットとした回路であり,セマフォオブジェクトの数だけ設置している。parameter1 からは,使用するセマフォオブジェクトを識別する ID コードを入力し,parameter2 からはカウンタ初期値を入力する。

セマフォに関連したシステムコールの処理を高速化するために,FIFO をハードウェアで実現する。FIFO は一種のメモリであり,フリップフロップを使用するために順序回路で組む。この FIFO は,セマフォに対する複数タスクの待ち行列を処理するために必要不可欠である。

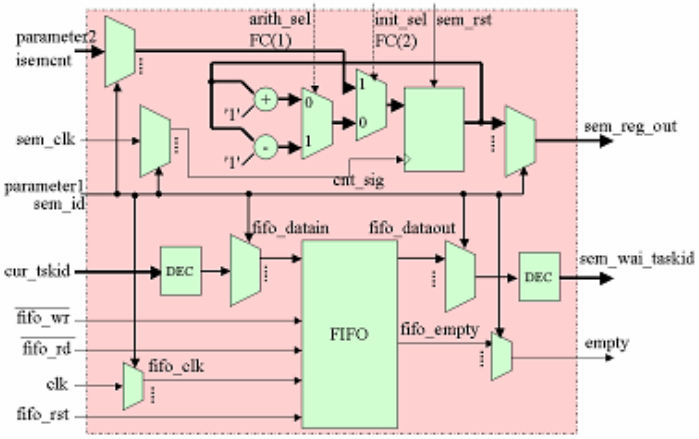

図５　セマフォの内部構成

4.5　インターフェース処理機能

リアルタイム OS の入出力レジスタには,0 から 7 までのアドレスを割当てた（図２）。function code(0 番地),parameter1～3(1～3 番地),errer code(4 番地),status1～2(5～6 番地),current Task ID(7 番地)である。

function code をシステムのメモリマップ上において 200000 番地に割り当てた場合，current Task ID の値を読み出すためには,200007 番地にアクセスする。このことをふまえて,システムコール sta_tsk を C 言語で記述したときのプログラムを図 6 に示す。また,このシステムコールを実行した時のリアルタイム OS(ハードウェア部)の動作を表した波形図を図 7 に示す。

図６のインターフェース処理プログラムでは，最初の３行で function code と parameter1,errer code にアクセスするためのポインタを定義している。５行目以下の sta_tsk では,次のような処理を行っている。

①sta_tsk の中では,まず起動するタスク ID コードをパラメータとして*HWTRON_R1 に代入し,sta_tsk の機能コードを*HWTRON_R0 に代入している。
②次の wait_clk(1)は,リアルタイム OS(ハードウェア部)が処理完了するまでの待ち時間である。
③最後に，*HWTRON_R4 にアクセスし，リアルタイム OS(ハードウェア部)から出力されたデータを取得している。この場合のデータは,エラーコードである。

システム上のメモリマップが変わった場合には,入出力レジスタへのポインタの変更で対応できる。また,システムコールによってはリアルタイム OS に渡すパラメータの数や,リアルタイム OS から取得する処理結果の数が異なるので,インターフェース処理プログラムの関数部分を適切に変更する。

```
#define HWTRON_R0    (volatile unsigned char *)0x200000
#define HWTRON_R1    (volatile unsigned char *)0x200001
#define HWTRON_R4    (volatile unsigned char *)0x200004
int ec;
int sta_tsk(int p1){
    ①  *HWTRON_R1 = p1;   /*タスク ID コードの出力*/
    ②  *HWTRON_R0 = 0xffe9;    /*機能コードの出力*/
        wait_clk(1);                    /*時間待ち*/
    ③   ec = *HWTRON_R4;      /*エラーコードの入力*/
        return ec;
}
```

図 6　インターフェース処理(sta_tsk)プログラム

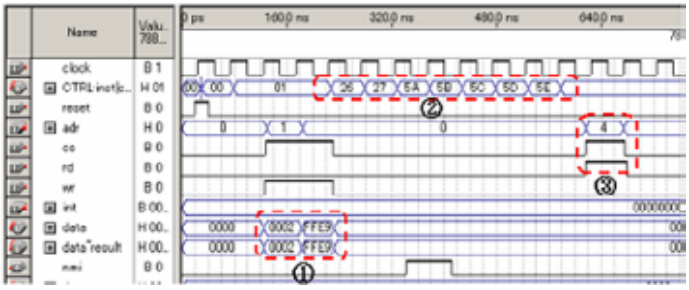

図 7　リアルタイム OS(ハードウェア部)の動作

5．評価

リアルタイム OS の開発には, VHDL(ハードウェア記述言語)を用いた。また,実装先の FPGA は, 安価で比較的大容量のロジックをもつ ALTERA 社の Cyclone を使用した。開発したリアルタイム OS の評価は,システムコールの処理時間(クロック単位)について,既存のソフトウェアで出来たリアルタイム OS と比較することで行った。比較対象とした既存リアルタイム OS は HOS[5]というフリーソフトであり,μITRON に準拠している。ターゲットに用いた MPU は,ルネサステクノロジ社製の H8/3048F（16 ビットマイコン）である。

評価結果の一部を表 2 に示す。表の数値は,システムコールとタスクの切り替えに要した時間である。今回開発したリアルタイム OS は,HOS と比較して 1/2 から 1/15 の処理時間を実現した。これは，MPU のソフトウェア処理と並列してハードウェア化したリアルタイム OS が動作しているからである。これにより,タスクをより多く実装可能になり,処理速度の遅い MPU に対してもリアルタイム OS を容易に利用することができるようになった。

更に,開発したリアルタイム OS の実装先を FPGA にすることで,リアルタイム OS が製品に組込まれた状態であっても,リアルタイム OS の仕様変更が容易にできるようになった。異なる MPU の実装に伴う各種バス幅の変更やメモリマップの変更にも柔軟に対応できる。

表 2　リアルタイム OS の処理時間比較

頻繁に使用するシステムコール例	従来の RTOS 単位:クロック	開発した RTOS 単位:クロック
wai_sem セマフォ資源獲得	5,226	532
sig_sem セマフォ資源返却	5,974	532
wai_flg イベントフラグ待ち	6,233	652
set_flg イベントフラグセット	8,084	592

6．まとめ

本研究では，組込み制御用 OS であるリアルタイム OS の機能の中でも使用頻度の高い部分をハードウェア化した。開発したリアルタイム OS は，ソフトウェアで出来たものと比較して 1/2 から 1/15 の時間で処理可能であり，スケジューラ等の重要な機能が MPU と並列動作するため，オーバーヘッドが大幅に削減されている。その上，インターフェース処理プログラムを介して，既存の MPU との接続を柔軟にしている。それにより，開発したリアルタイム OS を組込む製品開発においては，プログラム開発効率の向上，強いリアルタイム性の実現，低コストかつ低機能の既存 MPU の採用が可能になる等の利点が得られる。

参考文献

1) 荒木 英夫,久津輪 敏郎,原嶋 勝美:電子情報通信学会論文誌,C Vol.J86-C No.8,pp.799-807(2003).
2) (社)トロン協会:ITRON 標準ガイドブック 2,パーソナルメディア㈱(1994).
3) (社)トロン協会:μITRON3.0 標準ハンドブック,パーソナルメディア㈱(1993).
4) (社)日本システムハウス協会,(社)トロン協会:組込みシステムにおけるリアルタイム OS の利用動向に関するアンケート調査報告書,p6(2004).
5) https://sourceforge.jp/projects/hos/

（原稿受付　平成 17 年 8 月 4 日）