論文

任意に構成可能なリアルタイムシステムの開発

佐藤正利*　森　久直*　坂巻佳壽美*

Development of the configurable real-time system

Masatoshi SATO, Hisanao MORI and Kazumi SAKAMAKI

Abstract　In recent years, regarding the MPU,due to the progress made in semiconductor technology, there are many high-performance products on the market. However, when building a low-cost system, use of the high-performance MPU leads to a waste of resources being both expensive and more powerful than is actually necessary. Even if it uses low MPU of performance to carry a suitable realtime OS to a peripheral device, we can make demanded Embedded system. In addition, this becomes a cheap system and the thing which it is easy to utilize. By combining the 'Realtime OS' which developed this time in 2000 from 1999 together, it is possible to construct a 'Realtime System' . This report outlines that system.

Keywords　Embedded system, Realtime OS, Realtime system, Configurable real-time system

１．はじめに

多品種・少量生産の製品開発を行う中小企業にとって，製品の仕様に合わせるために任意に構成可能なリアルタイムシステムを活用することで，所望の組込みシステムをいつでも実現できる様にすることは，開発コストの削減，開発期間の短縮につながる。

リアルタイムシステムにリアルタイムOSを搭載しないことも可能であるが，開発者は終始システム全体のことを考えながら開発を行っていく必要に迫られる。リアルタイム OS を搭載する場合にも，余計な OS 処理が入るため性能が低下する，OS を実行させるためにメモリーを大きくせざるを得ない等の問題を考えることがある。しかし，一つ一つの仕事をタスクという単位に分割することができ，タスクごとの設計が容易になる利点は大きく，生産性，メンテナンス性に優れたシステムとすることが可能である。

近年，MPU の高速・高機能化により，システム構築の規模が大きくなってきているが，比較的小規模な組込みシステムを考えた場合には，市販の多機能なリアルタイム OS を利用するまでもないことが多い。小規模システムに適したリアルタイム OS を開発できれば，より製品の開発効率を上げることが可能となる。ここで開発するリアルタイム OS は，このような要求に応えるものである。使用する MPU，ハードウェア資源にとらわれず，任意に構成可能なリアルタイムシステムが可能となり，多くの小規模システムに組み込むことができる。

*情報システム技術グループ

２．リアルタイムシステムの仕様

本研究で開発したリアルタイム OS は，μITRON 仕様に準拠しながらも，利用できるハードウェア資源の限られた小規模システムに適したものとするために，機能を縮小し任意に構成できるようにしたものである。

多くのアプリケーションプログラムは，OS の元でタスクという単位で実行される。図１に，ここで開発したリアルタイムシステムとしてのタスクと OS の関係を示す。

リアルタイム OS には，独立して登録されたタスクを，MPU やメモリなどタスクを実行するために必要な資源の無駄遣いせずに確保し，実行することが必要である。タスクには，OS のシステムコール機能を利用して，タスク間の連携や，他のタスクの実行制御要求を行う機能が必要となる。

そこで，マルチタスク機能とタスク間連携機能に重点をおいた開発を行った。

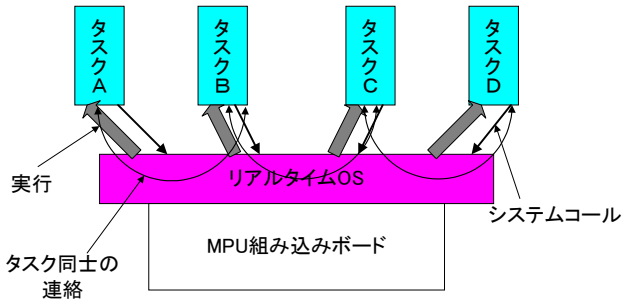

図１　タスクとリアルタイム OS の関係

2.1　マルチタスク機能の開発

OS にタスクを登録するときには，タスクのシステム内での識別子（タスク ID）と，実行に際しての優先度を与える必要がある。この登録されたタスクに対する実行要求の有無などにより，タスクをいくつかの状態に分けて管理する。図 2 に本研究で構成した機能を絞ったタスクの状態遷移図を示す。また，この状態遷移の動作を表 1 に示す。

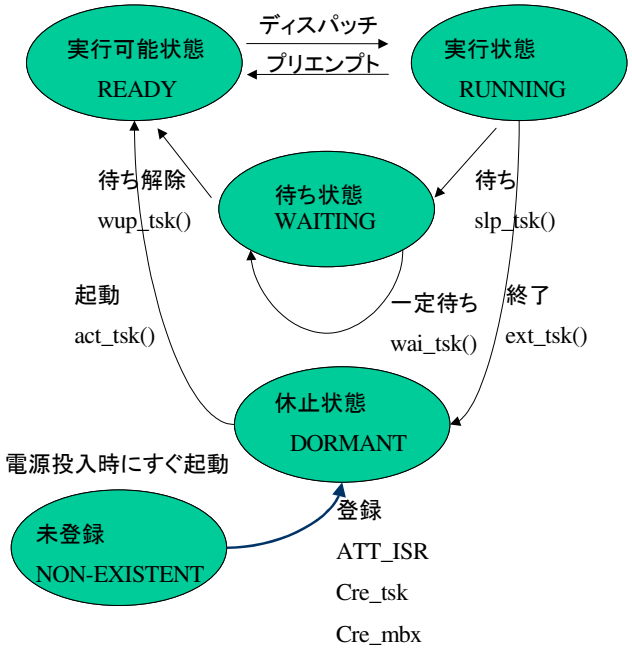

図 2　タスクの状態遷移図

表 1　状態遷移の動作

	タスクの状態説明
1	タスクは，電源投入前には未登録状態（Non-Existent）であるが，電源投入時にすぐ登録状態となり起動し休止状態（DORMANT）となる。休止状態のタスクの中から優先度の高いものから実行可能状態（READY）に移行するようにした。
2	OS は，実行可能状態にあるタスクの中から，最も優先度の高いタスクを実行状態（RUNNING）にする。同一優先度のタスクが存在するときは，先に起動しているタスクを実行後，同一優先度のタスクを実行する。
3	実行中のタスクより優先度の高いタスクが実行可能状態になると，現在実行状態のタスクを中断し，実行可能状態となるようにした。
4	実行中のタスクが終了すると，そのタスクは休止状態となる。
5	実行中のタスクが長い処理を行っている場合，または，他のタスクからの連絡を待つ必要があるとき，タスクは OS に対しその目標が発生するまで待ち状態（WAITING）となるようにした。
6	待ち状態のタスクは，その他のタスクからの連絡により実行可能状態に戻る。

また，複数のタスクがリアルタイム OS に制御されて実行する状態を図 3 に示す。タスクは A，B，C の 3 個を想定し，優先度はタスク A が最も高くタスク C が最も低いものとした。

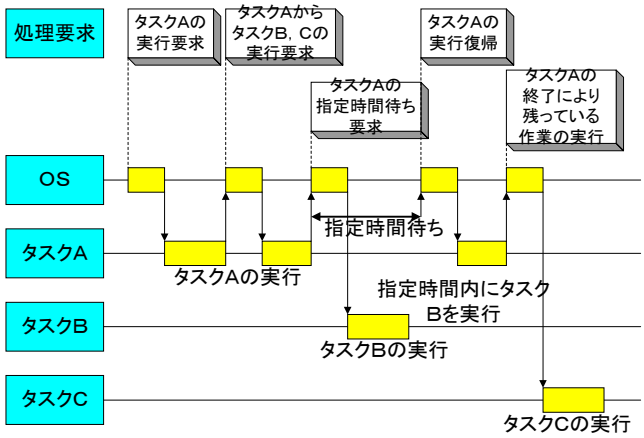

図 3　マルチタスクの実行形態

ここで使うシステムコールを，表 2 に示す。小規模なリアルタイムシステムにするために，必要最小限のシステムコールで構成した。

表 2　タスクに関する主要システムコール

	システムコール	機　　能
タスク管理機能	cre_tsk	タスクを生成する。
	act_tsk	タスクを起動する。
	ext_tsk	自タスクを終了する。
タスク付属同期機能	slp_tsk	タスクを待ち状態に移行する。パラメータはなし。待ち状態は，wup_tsk が発行されたときに解除される。
	wai_tsk	タスクを一定時間待ち状態に移行する。パラメータは，タイムアウト(tmout)。待ち状態は，wup_tsk が発行されるか tmout で指定した時間が経過したときのみ解除される。
	wup_tsk	待ち状態のタスクを起床する。パラメータは，タスク ID(tskid)。slp_tsk または wai_tsk で待ち状態となっていた tskid のタスクを実行可能状態へ移す。tskid のタスクが待ち状態にないときは，この wup_tsk が記憶され，待ち状態になったときに有効となる。

以上の機能により，ここで構成されたリアルタイム OS は，タスクに与えられた優先度やシステムからの要求をもとに，複数のタスクを切り替え，並列処理を行う。

2.2　タスク間の連携機能の開発

タスクは，各々が独立して OS から制御されるが，タスクの処理はお互いに何らかの関係をもって機能する。開発

したリアルタイム OS では，このタスク間の連携を取り合う機能も備えている。タスク間の連携機能には，イベントフラグ，セマフォ，メイルボックスがあるが，ここでは表 3 に示すようにタスク間連携機能としてメイルボックスを選択し，タスク間相互のデータの受け渡しを行うこととした。

表 3　タスク間連携機能に関するシステムコール

	システムコール	機　　能
同期・通信機能	cre_mbx	メイルボックスを生成する。
	snd_msg	メイルボックスへ送信する。パラメータは，メイルボックス ID(mbxid)と送信メッセージの先頭アドレス(pk_msg)。
	rcv_msg	メイルボックスからの受信を待つ。パラメータは，メイルボックス ID(mbxid)と受信メッセージの先頭アドレス(pk_msg)。

3．組込みボードへの適用とシステム設計及び実証

ここでは，開発したリアルタイムシステムの仕様により、アンドールシステムサポート社のエンベデッドトレーナー（Model ET-2）を用いて，実際にボード上のハードウェア資源を利用したシステム設計を行った。

ボード上には，ハードウェア資源として，ステータス出力，パラレル入出力，シリアル入出力，AD・DA コンバータ，スピーカ，DC モータ，ステッピングモータ，ボリューム，スイッチ等が実装されている。各々を動かすプログラムはタスクとして無限ループ状態にし，タスク間のやりとりをリアルタイム OS のシステムコールを利用してリアルタイム動作を行うようにした。どのタスクでも，リアルタイム OS を介して動作できるので，任意にハードウェア資源を組み合わせてリアルタイムシステムを構成することが可能になる。

以下に，ハードウェア資源を組み合わせて，リアルタイム動作を実証した一例を示す。

3.1　システム要求仕様

2 つの 16 桁入出力ポート LED とステッピングモータを同時に動かし、ボリュームにより瞬時にステッピングモータの回転スピードを変更することができるシステムにより実証実験を行った。ハードウエア構成は，図 4 に示す。

・ボリューム

2 つのボリューム（VR1,VR2）のうち VR1 を使用し，ステッピングモータの速度制御を行う。VR1 は AD コンバータの CH0 に接続され，回転させることにより 0～3V の電圧が出る。プログラムからは，AD コンバータに割り当てられたアドレスを読み込むことで，VR1 からの値を参照することができる。

・16 桁パラレル入出力ポート LED

１つのパラレルポートは，16 ビットで構成されている。この信号を LED により捉えることができる。

・ステッピングモータ

1-2 相励滋方式により回転させる。回転軸を 1 回転/秒させるためには，回転制御信号を 10ms で 96 個必要である。一定周期で回転信号を出力するためには，ソフトウェアタイマーを使用する。

・タイマー

エンベデッドトレーナーET-2 には，時間計測，信号数カウント，一定時間間隔で信号を発生することのできるタイマーがある。タイマーを動作させるためには，モードを指定するコントロールワードをコントロールワードレジスタに書き込み，カウントレジスタにカウントをどの程度行うかを設定する。

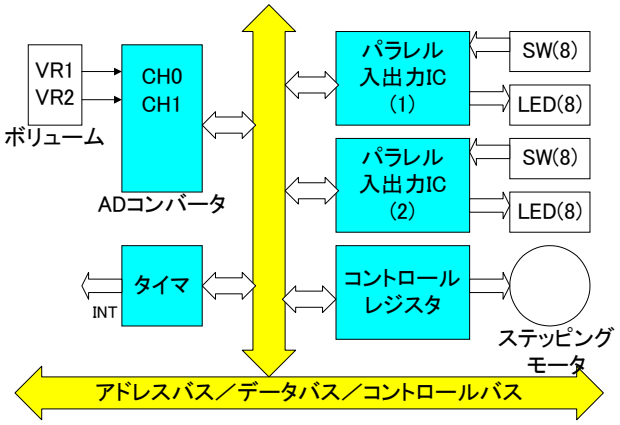

図 4　ハードウェア構成

以上のハードウェア資源を使用したリアルタイムシステムとしての要求仕様を以下に示す。

①ステッピングモータの回転制御

ステッピングモータを，ボリュームの値によりスピードを変化させて回転させる。

②2 つのパラレル入出力ポートへ，ステッピングモータの回転数に応じた信号を送り，その信号出力を 16 桁 LED に表示させる。

以上の要求を元に，簡易なリアルタイム動作検証を，ボリュームの調整制御，ステッピングモータの回転，16 桁 LED 表示という動作で行うこととした。

3.2　ソフトウェア設計

まず，プログラムにより実現する機能を以下に示す。

(1)ボリュームの値を読み込んでくる。

ボリュームの値は，一定周期ごとにその値を読み出す。読み出した値は，メモリに書き込む。このフローチャートを図 5 に示す。図 5 に示されたフローチャートが 1 つのタスクとなる。このタスクは，起動後にボリュームの値を読み出し，その値をメモリに書き込む。その後 wai_tsk シス

テムコールを発行し実行を停止する。このシステムコールの機能により，指定時間経過後に wai_tsk 発行後の次の命令から実行する。

このタスクは，一定周期でボリュームの値をメモリに格納する処理を繰り返す。

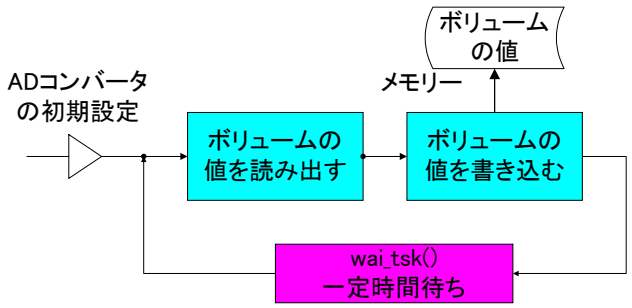

図 5　ボリューム読み出しタスクのフロー

(2)ボリュームからステッピングモータの回転速度を決める。

ボリューム読み出しタスクにより得られた値から，ステッピングモータの回転速度を計算する。図 6 に回転速度の計算フローチャートを示す。メモリ内のボリュームの値から回転速度を計算し，他のタスクが使用できるようにメモリに書き込む。

この処理は，ボリュームの値が変化したときのみ実行するタスクであるから，通常は待ち状態とし，ボリューム値の変更時のみ実行されるようにする。

slp_tsk を発行することで実行が停止し，ボリューム値読み出しタスクから値を読み込んだときに，wup_tsk システムコールを発行してもらう。

このように，回転速度の計算が独立したタスクとなっているため，ボリュームの特性が変更されたときにも，このタスクのみ変更することでシステムとしての機能全体を見直す必要がない。

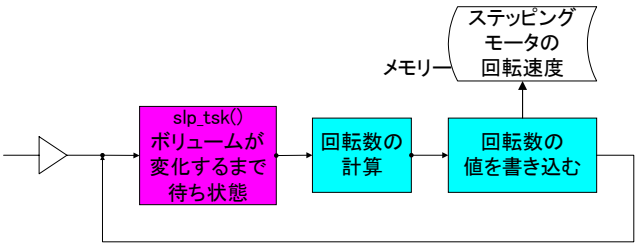

図 6　ステッピングモータ回転数計算タスクのフロー

(3)パラレル入出力ポートにステッピングモータの回転速度信号を出力する。

16 桁パラレル入出力ポート LED を表示するには，ポートに接続された外部機器へデータを送信するタスクが必要である。このとき，ステッピングモータの回転速度を制御するボリュームに変化があった場合は，そのタスクを優先することとした。

図 7 にパラレル入出力ポート LED 表示のフローチャートを示す。LED 表示要求は，出力データをメッセージとしたメイルボックス機能を用いる。ステッピングモータの回転数を確認し，その速度にあった信号を LED に表示するには，rcv_msg システムコールによりメイルボックスに要求メッセージが来るのを待ち，メッセージを受信した後に表示処理をはじめる。表示は，1 出力データを送信用レジスタに取り込む。送信用レジスタに書き込み後，snd_msg システムコールをメイルボックスに返し，処理を終了する。

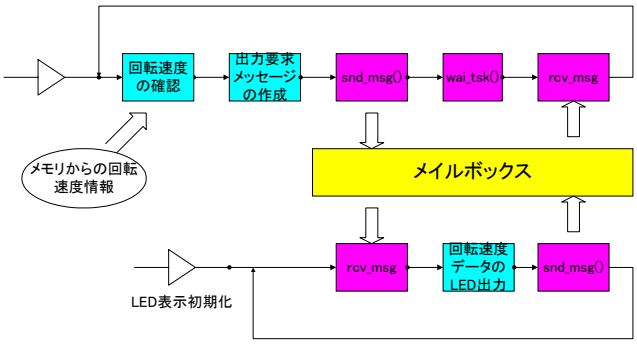

図 7　パラレル入出力ポート LED 表示フロー

4．まとめ

リアルタイムシステム設計においては，以下に示す一定の開発手順を用いることにより，組込みボードのハードウェア資源を任意に構成し活用することが可能となった。

①タスク単位の振り分け
②他のタスク，割り込み同期，ハンドリング方法の検討
③タイミングの検討
④タスク毎のプログラム設計

終始，全体システム構成を考える必要がなく，タスク毎のプログラム設計で任意に構成されたリアルタイムシステムを組むことできた。ここで開発した OS は，休止状態，実行待ち状態，実行可能状態，実行状態の 4 つの状態を遷移する機能を最小限に押さえたものであり，小規模なハードウェア資源のシステムに最も有効に活用できるものである。

参考文献

1) 社団法人トロン協会 ITRON 部会 坂村健，高田広章: μITRON4.0 仕様(2001).
2) 社団法人トロン協会: μITRN4.0 標準ハンドブック(2001).
3) パーソナルメディア:TRONWARE Vol.5, 23-136(1990).
4) パーソナルメディア:TRONWARE Vol.8, 32-43(1991).
5) 森久直，坂巻佳壽美:産技研研究報告第 4 号，89-92(2001).

（原稿受付　平成 14 年 8 月 1 日）