ソフトウェア部品化とハードリアルタイム処理技術

Hard Realtime Scheduling Theory and Trends in Software Components in Embedded Systems

横澤　彰　A. Yokosawa
平尾　繁晴　S. Hirao

組込みシステムにおけるソフトウェア部品化への流れと，ソフトウェア部品の組合せ利用時に発生するリアルタイム処理上の問題点を解決するための組込みシステム開発環境に要求されるハードリアルタイム処理のサポートについて紹介する。近い将来，プロセッサ性能の飛躍的な向上により，ソフトウェア部品を組み合わせて多様な機能を実現することが可能となる。ハードリアルタイム処理のサポートはこのようなシステムのプラットホームとして重要となる。

The growth in performance of embedded microprocessors in the near future will lead to the combination of several available software components as a widespread development strategy. In combining software components that have realtime constraints, there are several problems that must be solved. Hard realtime processing technology will play a major role in solving such problems.

1 まえがき

マイクロプロセッサの性能が飛躍的に伸びたことにより，個人用情報端末などの組込み機器で，これまで専用ハードウェアで実現していた機能をマイクロプロセッサ上のソフトウェアで実現することが可能になりつつある。組込み機器の機能の複合化を支援するために，ファイル システムやモデム機能などが組込み機器向けのパッケージとして入手可能になりつつある。このような機能モジュールをソフトウェア部品と呼ぶ。

ここでは，ソフトウェア部品化への流れについてまず解説し，次にソフトウェア部品の組合せ利用時に重要となるハードリアルタイム スケジューリング理論の概要を紹介する。最後にこの理論を実際に応用する際に必要なプラットホーム技術と，この技術分野の動向を紹介する。

2 組込みシステムにおけるソフトウェア部品化

パソコン上でのマルチメディア機能，例えば動画再生はこれまで専用の拡張ボードを必要としていたが，プロセッサの性能向上とともにソフトウェアだけで実現できるようになってきた。組込みシステムにおいても，組込み用プロセッサの性能向上とともに，これまで外付けの専用ハードウェアで実現していた機能がソフトウェアだけで実現できるようになってきた。特に，高性能マイクロプロセッサの主な応用分野であるマルチメディア，通信機器に必要な機能がソフトウェア部品として半導体メーカなどから供給されている。

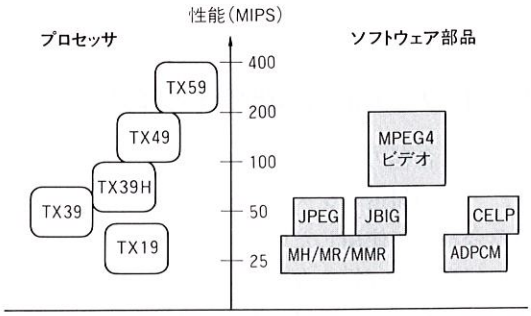

図1．プロセッサ性能とソフトウェア部品　TX シリーズ RISC プロセッサとソフトウェア部品の対応を示す。
Processor performance and software components

このようなソフトウェア部品には，MPEG (Moving Picture Experts Group)，JPEG (Joint Photographic Experts Group)，MH／MR／MMR (Modified Huffman／Modified Read／Modified MR)，JBIG (Joint Bi level Image experts Group) のような画像コーデック (圧縮伸長処理) および ADPCM (Adaptive Differential Pulse Code Modulation)，CELP (Code Excited Linear Prediction) のような音声コーデックがある。

当社製 RISC (縮小命令セットコンピュータ) プロセッサとソフトウェア部品を性能に関連づけて表示すると図1のようになる。

これらの機能を専用ハードウェアではなく，ソフトウェアで実現することにより，応用システムに次のような特長をもたせることができる。

(1)　ソフトウェアの使用によるもの

(a) プロセッサの高速化がシステムの高性能化に直結
(b) システムの柔軟性向上
・国際規格，産業標準の変更や追加に対する柔軟性
・処理アルゴリズムの改良に対する柔軟性
(2) ハードウェアの削減によるもの
(a) コストの低減
(b) 小型・軽量化
(c) 低消費電力化
(d) 信頼性の向上

各機能の代表的な応用分野を**表1**に示す。

表1．ソフトウェア部品の応用分野
Fields of application of software components

機　能	応用分野
MPEG	カラオケ，ビデオCD，ビデオ会議
JPEG	ディジタルスチルカメラ，カラープリンタ
MH/MR/MMR，JBIG	ファクシミリ
ADPCM，CELP	PHS，携帯電話

3 ソフトウェア部品の組合せの問題点

このように各種のソフトウェア部品が入手可能になってくると，一つの組込みシステムに，複数のソフトウェア部品を組み合わせて使用したいという要求が生ずる。ハードウェアの部品ならば組み合わせておのおの並列に動作させることができるが，単一プロセッサ上のソフトウェア部品の場合は，単独で動作する部品を組み合わせた場合に動作するかどうかは自明ではない。

単純な判定方法としては，ソフトウェア部品の要求性能の合計が，使用するプロセッサ性能を超えないことが必要条件である。しかし，これは十分条件ではない。

問題は，おのおののソフトウェア部品がなんらかのリアルタイム制約，すなわちある入力を受けてから応答するまでの時間的な制約をもつ場合に，単純に組み合わせてそれらの部品が正しく動作するとは限らないことである。組み合わせて動作可能かどうか判定するためには，おのおののソフトウェア部品が単なる要求性能ではなく，以下に述べるようなスケジューリング可能性判定のために，リアルタイム制約を示すパラメータを明記しなければならない。

4 ハードリアルタイム スケジューリング理論

まず，“ハードリアルタイム”という言葉の定義を明確にしておく。リアルタイム処理は，時間制約の性質によって分類される。ハードリアルタイムは，その分類の一つを指す用語である。人によってその定義は一律ではないが，ここでは，絶対に守られなければならない時間制約をもつリアルタイム処理という定義に従う。ハードリアルタイムな処理が守らなければならない時間制約のことを“デッドライン”と呼ぶ。これに対して，時間制約が平均時間で与えられる場合のように，時間制約の限界値が決まっていない処理を“ソフトリアルタイム”と呼ぶ。

複数のタスクを組み合わせて実行した場合に，それぞれの時間制約が満たされるかどうかを判定することは，スケジューリング理論の対象として研究されてきた。いくつかのアプローチがあるが，ここではそのなかでもっとも実用に近いとされるRMA（Rate Monotonic Analysis）と呼ばれる解析手法について紹介する。

RMAの起源は，LiuとLaylandの論文にさかのぼる[1]。これは，シングルプロセッサのシステムで，複数のタスクに静的に優先度を割り当てる場合に適用可能な理論である。ただし，扱えるタスクの集合に強い制約を課していた（**表2**）。

表2．初期のスケジューリング理論の制約
Restrictions in earlier theory of scheduling

(1) 周期タスクだけを扱う
(2) タスクの最大実行時間は既知
(3) タスクは周期の始めに実行可となる
(4) デッドラインは周期に一致する
(5) タスク間に同期はない
(6) タスクはみずから実行を中断しない
(7) タスク切換えのオーバヘッドを無視
(8) シングルプロセッサシステム

そのうえで，周期タスクの優先度を，その周期の短いものから順に割り当てるというスケジューリング方式（RMS：Rate Monotonic Scheduling）が最適なアルゴリズムであることを示した。ここで，最適とはほかのアルゴリズムでスケジューリング可能ならばRMSでもスケジューリング可能であるという意味である。

スケジューリング可能性は，各タスクのCPU使用率の合計が式(1)に示す関係を満たすかどうかで判定する。

$$\sum_{j=1}^{n}\frac{C_j}{T_j}\leqq n(2^{1/n}-1) \qquad (1)$$

ここで，C_jはタスクjの実行時間，T_jはタスクjの周期である。右辺の値は，タスク数nが2のとき0.83，3のとき0.78，nが大きくなると0.69に収束する。

例として，**表3**に示したタスクの集合のスケジューリング可能性を判定してみる。このタスクの集合の場合，CPU使用率の合計は0.96で，タスク数が三つの場合のRMAでの上限値0.78を超えており，この方法ではスケジュール可能性の判定はできない。このような場合は，各タスクの応答時間を網羅的に計算するアルゴリズムが知られており，

表3．タスク集合の例
Example of task set

	周期＝デッドライン（ms）	実行時間（ms）	CPU使用率
タスク1	9	3	0.333
タスク2	15	6	0.4
タスク3	110	25	0.227
計			0.960

定量的に判定することができる(2)。そのアルゴリズムを図2に示す。

このアルゴリズムに従って表3の各タスクの応答時間を求めると表4のようになり，すべてのタスクの応答時間がデッドラインに間に合うことがわかる。この場合はスケジュール可能である。求めた応答時間がデッドラインを超えてしまう場合はスケジューリングできない。

さて，この理論を現実の問題に適用しようとする場合，表2に示したタスクの集合に課される制約条件が足かせとなる。そこで，これらの制約を緩めて，現実的な応用に適用可能にするための研究が進められ，現在では実用の域に達した(2)。以下に，表2の制約のなかで実用上特に問題となる(1)，(4)，(5)の制約を回避する方法を紹介する。

タスク i の応答時間を求める。

$$a_0 = \sum_{j=1}^{i} C_j$$

以下，a_{n+1} が a_n と等しくなるまで繰り返し

$$a_{n+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_n}{T_j} \right\rceil C_j$$

$\lceil\ \rceil$ は，括弧内の値を下回らない最小の自然数

図2．応答時間を求めるアルゴリズム　タスクの応答時間を網羅的に計算で求める方法を示す。
Algorithm for calculating task response time

表4．タスク集合の例（表3）の応答時間
Response time of task set in Table 3

	優先度	周期＝デッドライン（ms）	実行時間（ms）	応答時間（ms）
タスク1	高	9	3	3
タスク2	中	15	6	9
タスク3	低	110	25	103

4.1　非周期タスクの扱い

非周期タスクでも，(1)起動間隔に下限があるものと，(2)ある時間間隔の起動回数に上限があるものとについては，周期タスクとみなしてスケジューリング可能性を判定することができる。起動間隔に下限がある場合はその下限時間を周期に読み替えて判定する。時間間隔の起動回数に上限がある場合は，その時間間隔を周期とし，上限回数分の実行時間を実行時間と読み替えて判定する。

いずれの限界もない非周期タスクの場合，ハードなデッドラインを設定することはできない。

4.2　周期よりも短いデッドライン

デッドラインが周期に一致しない場合は，デッドラインの短い順に優先度を割り当てるDeadline Monotonic Schedulingが最適である。そのうえで，図2に示したアルゴリズムによって各タスクの応答時間を求め，デッドラインを満たすかどうか判定する。

4.3　タスク間の同期の扱い

あるタスクが，同期によって優先度の低いタスクに実行権を取られてしまう（優先度逆転）時間の上限を決めることができれば，その時間をタスクの実行時間に加えて計算することで判定することができる。

優先度逆転時間の上限を決めることができるかどうかは，用いる同期プロトコルに依存する。そのため，ハードリアルタイム処理が必要な場合は適切な同期プロトコルを選択することが必要である。

同期プロトコルについては，文献(2)を参照していただきたい。

5　ハードリアルタイム処理に必要な技術

5.1　リアルタイム オペレーティング システム

リアルタイム制約をもつソフトウェア部品の流通という観点では，標準のプラットホームとしてのリアルタイム オペレーティング システム（OS）の普及が望ましい。わが国においてはμITRON仕様(注1)が組み込み用リアルタイムOSとして業界標準の位置を占めているが，製品による実装依存の幅が大きい。さらに標準化の度合いを強めた仕様が求められており，ITRON専門委員会の呼びかけで検討が進められている。その検討項目の中にハードリアルタイムのサポートも含まれている。

ここでは，μITRON仕様に限定せずに，リアルタイムOS一般に対して要求される機能をまとめる。

RMSもしくは，Deadline Monotonic Schedulingは，タスクの優先度を静的に割り当てるため，リアルタイムOSのスケジューリング方式としては，実行可能なタスクのなかで

(注1)　TRONは，The Realtime Operating system Nucleusの略称。ITRONは，Industrial TRONの略称。

もっとも優先度の高いタスクに実行権を与える優先度ベースのスケジューリングがあればよい。

優先度ベース スケジューリングは，多くのリアルタイムOSで標準的に使用できるスケジューリング方式であり，その意味では新たな機能が要求されるわけではない。ただし，原則どおりに優先度を割り当てると，タスクの数だけ優先度を必要とするので，組込みシステムにおいては制約の厳しいメモリ容量を圧迫することになる。これを避けるためには，いくつかのタスクに共通な優先度を割り当てるなどのくふうが必要となる。そうすると，純粋なRMSではなくなり(1)式による優先度判定はできなくなるが，必要十分条件を判定するアルゴリズム（図2）を使えば判定が可能である。

タスク間の同期機構については，4.3節に述べたように優先度逆転の上限時間を決めることが可能な同期プロトコルのサポートが求められる。また，OS自体に上限のない優先度逆転を引き起こすような処理が存在しないことも要求される。

5.2 実行時間の見積り

RMA理論では，タスクの実行時間は既知のものとしてスケジューリング可能性を判定する。しかし，ソフトウェアの実行時間はCPUの品種と動作クロックだけでは決まらず，使用するメモリの速度にも依存する。また，32ビットクラスのRISC系組込みマイコンでは，キャッシュメモリの搭載が一般的であるが，リアルタイム処理に対するキャッシュメモリの影響を評価する方法は確立していない。

リアルタイム処理のなかでは割込み処理が頻繁に使われるが，キャッシュメモリが効果的なプログラムの処理中に割込み処理が入ると，割込み処理のために，割り込まれたプログラムが使用していたキャッシュメモリの一部が追い出されてしまう。このため，割込み処理の後で割り込まれたプログラムの実行が再開されると，しばらくはキャッシュミスが多発してオーバヘッドがかかるという現象がみられる（図3）。

この現象を考慮に入れてスケジューリング可能性を判定するためには，優先度の高いタスクの実行時間に，それが他のタスクに割り込む際に他のタスクに与えるオーバヘッドの時間を加えた値を使えばよい。しかし，そのオーバヘッドの見積りは困難である。

また，スループット重視のシステムで，このような割込みによるキャッシュミスのオーバヘッドを避けたい場合には，割込みに対する応答がやや鈍くなることが許容できるならば，割込みではなくポーリング主体のプログラムとすることも可能である。

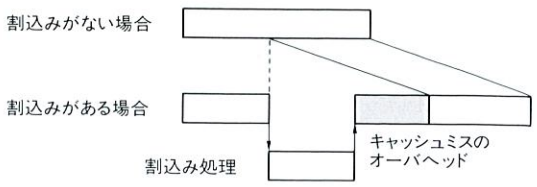

図3．実行時間に対する割込みとキャッシュメモリの影響　割込み処理の後，割り込まれた処理にキャッシュミスによるオーバヘッドが発生する。

Cache memory effect on program execution at time of interruption

将来的には，LSIの集積度向上を，実行時間に不確定な影響を与えるキャッシュメモリとして利用するのではなく，高速で確実なオンチップメモリとして利用することによって実行時間の見積りを容易にすることが一つの手法として普及すると考える。

6 あとがき

近い将来，組込みシステム分野でソフトウェア部品の組合せ利用を，ソリューションの一つとして利用可能としなければならない。その前提の一つとして，ここに述べたハードリアルタイム処理技術を，実際の製品開発の舞台へ適用することが必要である。ソフトウェア部品の設計手法，リアルタイムOS，ソフトウェア実行時間の評価手法といった広い範囲の技術について，その研究と普及活動を進めていく所存である。

文　献

(1) C. L. Liu and J. W. Layland: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, J. Assoc. Comput. Mach. 20, 1, pp.40-62 (1973)

(2) M. H. Klein, et al: A Practitioner's Handbook for Real-Time Analysis, Kluwer Academic Publications (1993)

横澤　彰　Akira Yokosawa

マイクロエレクトロニクス技術研究所 システムLSI技術研究所主務。リアルタイムOSおよびソフトウェア部品化技術の開発に従事。

Microelectronics Engineering Lab.

平尾　繁晴　Shigeharu Hirao

情報・通信システム技術研究所 COS開発センター主査。マイコン応用ソフトウェアの開発に従事。

Information & Communications Systems Lab.