エンジン制御システムのためのリアルタイム性検証手法

飯山真一[†] 高田広章[†] 菅沼英明[††]

自動車のエンジン制御システムは，燃費向上や排気ガスのクリーン化の要求からソフトウェアの大規模化・複雑化が著しく，体系的なソフトウェア設計手法の導入が求められている．本論文では，エンジン制御システムの設計や時間制約の解析に適用することを目的として，RMA に基づいた体系的な手法を提案する．また，その手法を実際のエンジン制御システムへ適用した事例について述べる．

A Schedulability Analyzing Method for Engine Management System

Shinichi Iiyama,[†] Hiroaki Takada[†] and Hideaki Suganuma[††]

The software of an engine management system is getting larger and more complex rapidly in order to satisfy requirements for high fuel efficiency and low exhaust. Therefore, a systematic software design approach is required. This paper proposes a systematic approach extending the RMA for the design of an engine management system. And we describe an application of the method to an engine management system.

1. はじめに

近年，組込みシステムのソフトウェアの複雑化・大規模化が著しいが，エンジン制御システムもその例外ではない．低燃費化，走行性の向上，排気ガスや安全性に関する厳しい規制をクリアするために，エンジン制御システムのソフトウェアは複雑化・大規模化している．

また，近年の新しい動きとして，エンジン制御を含めた複数の制御を 1 つのプロセッサで実現する複合化の動きや，自動車制御の高度化にともない，複数のプロセッサを自動車内のネットワークで接続した分散制御システムが採用されるケースが増えつつある．このような自動車制御の高度化への要求は，今後もさらに続くものと考えられる．

このように，エンジン制御を行うソフトウェアが大規模化・複雑化・複合化する中で，ソフトウェアの品質と生産性を高く保つために，ソフトウェア開発の効率化を支援するツールの採用や，体系的なリアルタイム性検証手法の導入が必要不可欠となる．

従来のエンジン制御システムのソフトウェア開発においては，システムが時間制約を満たすかどうかを事前に確かめる方法がアドホックであり，ソフトウェアが完成し動作テストをするまで時間制約を満たせるかどうかが分からないという状況があった．また，時間制約が満たせないと分かった場合の対処もアドホックに行われている．ソフトウェアに多くの修正を加えた結果，一度システムが時間制約を満たす状態になれば，そのソフトウェアを容易に修正を加えることができないという状況もあった．体系的な手法を導入したからといって，徹底したテストが不要ということにはならないと思われるが，このようなアドホックな検証手法を用いた開発が，特にソフトウェアの保守性や再利用性の面で非効率であることは疑いの余地がない．

そこで本研究では，エンジン制御システムの時間制約の解析を目的として，代表的なリアルタイムスケジューリング理論である Rate Monotonic Analysis（RMA）に基づいた体系的な手法を提案し，実際のシステムへ適用することを目指している．また，システムが時間制約を満たせるかどうかを事前に確かめ，時間制約が満たされなかった場合にも体系的な対処を可能にする方向で研究を行っている．システムの時間的振舞いを解析する中で，従来の理論だけでは扱えない部分が見つかったため，それを解決するための理論の

† 豊橋技術科学大学情報工学系
Department of Information and Computer Sciences, Toyohashi University of Technology
†† トヨタ自動車第 2 電子技術部
Electronics Engineering Div. II, Toyota Motor Corporation

拡張も行った．

本論文では，エンジン制御システムへの RMA を適用するための手法と，その適用事例について紹介する．以下，2 章において，本研究で扱うエンジン制御システムの概略を紹介した後，3 章において，エンジン制御システムへ適用するために理論的な拡張について述べる．4 章では，その適用事例について述べる．

2. エンジン制御システム

2.1 ハードウェア構成

自動車のエンジン制御は，エンジン回転数やアクセル開度などのセンサから得られる入力信号を処理し，燃料の噴射量，噴射時間や点火タイミングなどを決定し，各アクチュエータを駆動することで実現される．これらの処理は，ECU（Electronic Control Unit）と呼ばれるコンピュータを用いて実現される．ECU は，1 つ以上のプロセッサと RAM，ROM のほかに，センサやアクチュエータとのインタフェース回路で構成されている（**図 1**）．

多くのエンジン制御システムには，RAM と ROM を内蔵し外部にバスを出していないワンチップマイコンが用いられているため，システムで使用できるメモリ量には厳しい制限がある．また，自動車の場合，コスト，耐環境性能や信頼性の視点から，簡単にマイコンの性能を上げて対応という方法をとることができない．つまり，与えられたマイコンにいかに必要な機能を盛り込むかを考えなければならない．

2.2 ソフトウェア構成

エンジン制御を実現するために非常に多くのセンサやアクチュエータが必要となり，これにともなって制御を行う個々の処理の数もきわめて多くなる．各処理を別々のタスクで実現することは，限られたメモリ領域に収めることができなくなるばかりでなく，タスク切替えにともなうオーバヘッドの増加を招く．そこで，各処理を起動周期や相対デッドラインごとにタスク分割を行い，タスクの数を小さく抑えている．

エンジン制御は，一定の時間周期で起動される時間同期型の処理とエンジンの回転に同調して一定の回転角周期で起動される回転角同期型の処理に分けられる．前者には冷却水の温度センサからの信号から各種の補正パラメータを計算する処理などが含まれ，後者には燃料の噴射タイミングを決定する処理などが含まれる．これらの処理は，静的優先度割付けされた優先度ベーススケジューリングを行うリアルタイム OS により，プリエンプティブなマルチタスク環境下で実行される．車両やエンジンの形式によって様々であるが，今回の検討用エンジンに代表される標準的なエンジン制御ソフトウェアは，10〜20 個程度のタスクで構成されている．

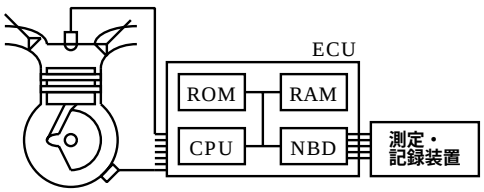

図 1 エンジン制御システムの概要
Fig. 1 Engine management system.

ソフトウェアを構成する処理の中には，必ず時間制約を満たすように実行しなければならないハードリアルタイム処理と，システムが高負荷になった場合には，たまにその実行が遅れても近い将来に実行すればよいソフトリアルタイム処理がある．前者には，燃料の噴射時間や点火タイミングなど，システムの品質に大きく影響する処理が含まれ，後者には，時定数の長い補正パラメータの決定など，比較的影響の小さい処理が含まれる．前述のように，自動車のコスト制約は厳しく，簡単にはプロセッサを代表とするハードウェアの性能を上げることはできないため，システムの高負荷時にソフトリアルタイムな処理が時間制約を満たせなくなることは避けらない．よって，システム全体のスケジュール可能性を考えた場合，ハードリアルタイム処理のスケジュール可能性をいくらかの余裕を持って保証できれば，システムとしてのリアルタイム性を保証できると考えている．

エンジン制御システムのいくつかのタスクは，その内部でサブスケジューリングを行っている．サブスケジューリングとは，特定の周期で起動するタスク内で，2 回起動されるごとに 1 回実行される処理，4 回に 1 回実行される処理など，同一のタスクであっても起動されるたびに実行される処理が異なることである．

エンジン制御システムでサブスケジューリングを行う主な理由は，タスクの数を減らしてメモリの使用量を小さく抑えることである．たとえば，回転同期型タスクの処理の中には特定の角度のときのみ実行すべき処理があるが，それ専用にタスクを用意するのではなく一定の回転角周期で起動されるタスクの中で何回かに 1 回実行する方法がとられる．また，たとえば，周期が 8 ミリ秒でデッドラインが 4 ミリ秒の処理を 4 ミリ秒の周期で起動されるタスクで，2 回に 1 回に実行する方法もある．

3. リアルタイム性検証手法

本章では，前章で述べたエンジン制御システムのリアルタイム性を検証するために，RMA に基づいた手法を提案する．

エンジン制御システムに RMA を適用する場合，まず考慮しなくてはならないのは，サブスケジューリングを行うタスクの取扱いである．サブスケジューリングを行うタスクの実行時間は周期的に変動するため，それを考慮せずに行ったプロセッサ使用率やスケジュール可能性解析の結果は，実際よりも悲観的となる．このような状況を扱うために，我々は，サブスケジューリングを行うタスクをマルチフレームタスク[1]としてモデル化した．

もう 1 つ考慮しなければならないのは，今回取り扱ったエンジン制御システムには，相対デッドラインが周期よりも長いタスクがある点である．マルチフレームタスクモデルでは，タスクの相対デッドラインが周期と一致するものとしているため，ここでは，それを拡張した一般化されたマルチフレームタスク[2]としてモデル化する．

しかしながら，一般化されたマルチフレームタスクからなるタスクセットに対しては，タスクの相対デッドラインが周期以下の場合のスケジュール可能性解析手法しか提案されておらず[3]，そのままでは，エンジン制御システムに適用することはできない．

そこで，本研究では，あらたにタスクの相対デッドラインが周期よりも長い場合にも適用できる一般化されたマルチフレームタスクのスケジュール可能性解析手法を提案する．タスクの相対デッドラインが周期より長い場合には，同じタスクが複数，同時に実行可能となりうる．この場合を取り扱えるように level-i busy period [4]の考え方を用いる．

また，我々は，マルチフレームタスクのスケジュール可能性の必要十分条件を効率的にチェックする方法として maximum interference function（MIF）を提案しているが[3),5]，スケジュール可能性解析における MIF の正当性の証明が課題として残っていた．本論文ではその正当性の証明を行う．

3.1 マルチフレームタスク

マルチフレームタスクモデル[1]は，タスクの実行時間があるパターンに従って大きく変動する状況を効率的に扱うための枠組みである．マルチフレームタスクモデルでは，Liu らにより提案された古典的な周期タスクモデル[6]におけるタスクの各周期をフレームと呼び，各フレームの最大実行時間が大きい周期で変動するものとして扱われる．いい換えると，タスクの（大きい）周期をいくつかの同じ長さのフレームに分割し，タスクの最大実行時間をそれぞれのフレームに対して与える．タスクは各フレームの始めに起動され，そのフレームの終わりをデッドラインとする．

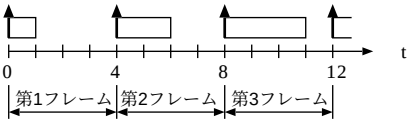

図 2 マルチフレームタスク
Fig. 2 A mutliframe task.

例として，3 つのフレームを持つマルチフレームタスクが実行される様子を**図 2** に示す．このマルチフレームタスクのフレームの長さは 4 で，フレームの実行時間はそれぞれ 1，2，3 である．この場合，3 つのフレームは周期 12 で繰り返して実行される．

一般化されたマルチフレームタスク（Generalized Multiframe Task，GMF タスク）は，マルチフレームタスクの前提を，(1) 各フレームの長さが違う場合，(2) 各フレームにおけるデッドラインがそのフレームの終わりに一致しない場合を許すという 2 つの点で緩めたものである[2]．以下では，単にマルチフレームタスクといった場合，一般化されたマルチフレームタスクのことを指すものとする．N_i 個のフレームを持つマルチフレームタスク τ_i は，j 番目のフレームを 3 つ組 (C_i^j, D_i^j, P_i^j) で表すことにすると，長さ N_i の 3 つ組の列 $((C_i^0, D_i^0, P_i^0), \cdots, (C_i^{N_i-1}, D_i^{N_i-1}, P_i^{N_i-1}))$ によって特徴づけられる．ここで，i はタスクを優先度の高い順に並べたときのタスクの番号，C_i^j は τ_i の j 番目のフレームにおける最大実行時間，D_i^j は j 番目のフレームの（相対的な）デッドライン，P_i^j は j 番目と $j+1$ 番目のフレームの最小間隔（minimum separation）を示す．τ_i の（通算して）k 番目のフレームの起動時刻を a^k，(絶対）デッドラインを $a^k + d^k$，実行時間を c^k とすると，次の関係が成立する．

$$a^0 \geq 0,\ a^{k+1} \geq a^k + P_i^{k \bmod N_i}$$
$$d^k = D_i^{k \bmod N_i}$$
$$c^k \leq C_i^{k \bmod N_i}$$

3.2 Critical Instant 定理

あるタスク（ないしはフレーム）に対する critical instant とは，そのタスク（ないしはフレーム）の実行が起動されてから終了するまでの時間が最も長くなるような状況をいう．プリエンプティブな静的優先度スケジューリングのもとで，GMF タスクセットに対する critical instant 定理は，次の critical instant

candidatesの定義を用いれば，次のように記述できる．

定義 1（Critical Instant Candidates） あるGMF タスクに対する critical instant candidates とは，そのタスクのあるフレームがすべてのより優先度の高いタスクのいずれかのフレームと同時に起動した状況で，以降続くフレームを最小間隔で起動し，いずれのフレームにおいても最大実行時間を使い切る場合をいう．

定理 2（Critical Instant 定理） あるGMF タスクのあるフレームに対する critical instant は，そのタスクに対する critical instant candidates のいずれかである．

critical instant 定理により，critical instant から実行を開始したときの各タスク（ないしはフレーム）のデッドラインまでのスケジュールをチェックするだけで，スケジュール可能性を必要十分にチェックできる．また同時に，上に示した定理は，GMF タスクセットの場合，あるタスクのあるフレームのスケジュール可能性をチェックするために，すべてのより優先度の高いタスクのどのフレームと同時に起動されたかを考える必要がある．つまり，すべてのより優先度の高いタスクのすべてのフレームとの組合せに対してチェックを行う必要がある．

3.3 Maximum Interference Function

前節で述べたすべての組合せに対して，マルチフレームタスクを含むタスクセットのスケジュール可能性解析を効率良く行う方法として，Maximum Interference Function（MIF）を用いた方法を提案する．MIF は，あるタスクが長さ t の期間にそれより優先度の低いタスクの実行を邪魔する最大時間（の関数）である．タスク τ_i の MIF $M_i(t)$，すなわち τ_i が長さ t の期間にそれより優先度の低いタスクの実行を邪魔する最大時間は，次の式で求めることができる．

$$M_i(t) = \max_{0 \le k \le N_i - 1} I_i^k(t)$$

ここで $I_i^k(t)$ は，タスク τ_i が k 番目のフレームから実行開始してから長さ t の期間にそれより優先度の低いタスクの実行を邪魔する最大時間の関数（これを，interference function と呼び，以下 IF と略す）で，次の式で表すことができる．

$$I_i^k(t) = \sum_{h=k}^{k+J-1} C_i^{h \bmod N_i} + \min\left(C_i^{(k+J) \bmod N_i}, t - \sum_{h=k}^{k+J-1} P_i^{h \bmod N_i} \right)$$

ここで J は次の式を満たす最大の整数である．

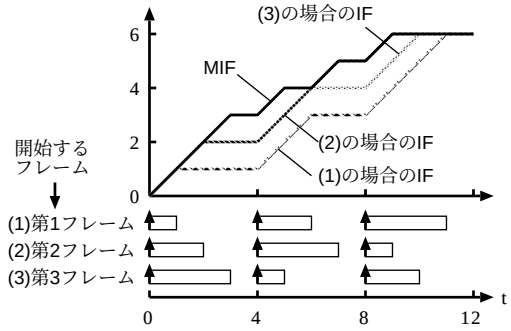

図 3 GMF タスクの MIF
Fig. 3 The MIF of a GMF task.

$$\sum_{h=k}^{k+J-1} P_i^{h \bmod N_i} \le t$$

マルチフレームタスクの MIF は，各フレームから実行を開始した場合の IF の最大値である．例として，マルチフレームタスク $\tau_i = ((1,4,4),(2,4,4),(3,4,4))$ の各フレームが critical instant から実行を開始した場合のタスクの実行の様子と IF を**図 3** に示す．タスクが実行されている間は，IF の傾きは 1 となり，実行されていない間は 0 となる．このように，IF は傾き 1 の区間と傾き 0 の区間が交互に繰り返される広義の単調増加をする連続関数として表される．τ_i の MIF は，3 つの IF の中で最も上の線をつないだものである．

MIF をこのような連続関数で定義することで，時刻 t におけるタスク τ_i の余裕時間 $t - M_i(t)$ も単調増加する．つまり，これは余裕時間がデッドラインで最大となることを意味している．

MIF の定義より，マルチフレームタスク τ_i の k 番目のフレームがスケジュール可能となる必要十分条件は，次のように書くことができる．

$$\exists\, t,\ 0 < t \le D_i^k\ ,\ \sum_{j=1}^{i-1} M_j(t) + C_i^k \le t \qquad (1)$$

このことは定理 4 において証明する．この条件を用いると，タスク τ_i の k 番目のフレームに対して，すべての critical instant candidates におけるスケジュール可能性を効率良くチェックすることができる．

3.4 MIF の飽和加算

あるタスクのスケジュール可能性を考えた場合，そのタスクにとって，どのタスクにどれだけ邪魔されたかではなく，すべての高優先度タスクに（まとめて）どれだけ邪魔されたかが問題である．そのタスクがどれだけ邪魔されたかは，すべての高優先度タスクの MIF を加算することで知ることができる．しかし，通

常の算術加算で得られたものは実際に邪魔されうる時間より長くなるため，MIF のための加算を新たに定義する必要がある．そこで，まず飽和変換を定義する．

定義 3（飽和変換） 非負の整数の傾きを持つ時間の関数 $F(t)$ に対して，飽和変換により得られた新たな関数を $F_s(t)$ とすると，$F(t)$ と $F_s(t)$ の関係は次式のようになる．

$$F_s(t) = t - \max_{0 \le \tau \le t} (\tau - F(\tau))$$

ちなみに，$F(t)$ があるタスクの request function（タスクの要求する計算時間をその到着時に累積する時間関数）であれば，上式の右辺第 2 項は $[0, t]$ における余裕時間の最大値となり，$F_s(t)$ は IF になる．つまり，飽和変換を用いると，IF $I_i^k(t)$ は次のように定義できる．

$$I_i^k(t) = t - \max_{0 \le \tau \le t} \left(\tau - \sum_{h=k}^{k+J-1} C_i^{h \bmod N_i} \right)$$

飽和変換の定義を用いると，MIF $M_j(t)$ を単純加算したものと，それをさらに飽和変換したものとの関係は次のように記述できる．以降，飽和変換を用いて加算することを飽和加算と呼び，次式の左辺のように記述する．

$$\sum_{j=1}^{i-1}{\!\!\!\!\!\!\sum} M_j(t) = t - \max_{0 \le \tau \le t} \left(\tau - \sum_{j=1}^{i-1} M_j(\tau) \right) \quad (2)$$

飽和加算の例として，2 つのマルチフレームタスク $\tau_1 = ((1, 8, 8), (2, 8, 8))$，$\tau_2 = ((3, 8, 8), (2, 8, 8))$ を考える．τ_1，τ_2 の MIF をそれぞれ $M_1(t)$，$M_2(t)$ と表すと**図 4** のようになる．また，$M_1(t)$，$M_2(t)$ の単純な加算 $M(t)$ $(= M_1(t) + M_2(t))$ は傾きが 2 の区間を持ち，この区間では時刻 t までに実際に邪魔されうる時間よりも長く邪魔することになる．$M(t)$ を飽和変換することで，実際に邪魔されうる時間を表す関数 $M_s(t)$ を得ることができる．

つまり，複数の MIF を単純加算した場合でも，さらに飽和変換することにより，最大の傾きが 1 となるという MIF の性質を維持することができる．

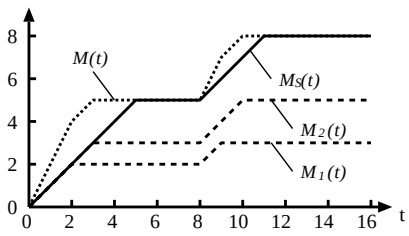

図 4 飽和加算
Fig. 4 The saturate summation.

3.5 MIF を用いた検証手法

MIF の飽和加算を用いると，式 (1) を次のように書き換えることができる．

$$\exists\, t,\ 0 < t \le D_i^k\ ,\ \sum_{j=1}^{i-1}{\!\!\!\!\!\!\sum} M_j(t) + C_i^k \le t \quad (3)$$

MIF の飽和加算を用いると，マルチフレームタスク τ_i の余裕時間は単調増加し，デッドライン D_i^k において最大となるから，上式が成立するかをすべての t ではなくデッドラインでのみ調べればよい．よって，マルチフレームタスクがスケジュール可能となる必要十分条件は，次の定理のように記述することができる．

定理 4 マルチフレームタスク τ_i の k 番目のフレームがのスケジュール可能となる必要十分条件は，次の式で表すことができる．

$$\sum_{j=1}^{i-1}{\!\!\!\!\!\!\sum} M_j(D_i^k) + C_i^k \le D_i^k$$

この条件式は，MIF と飽和加算の定義を用いることで，マルチフレームタスク τ_i の k 番目のフレームのスケジュール可能性をそのデッドラインで調べるだけで判定できることを示している．

定理 4 を証明するために，以降に示す 3 つの補題を証明し，式 (1) が成立すること，式 (3) が成立すること，マルチフレームタスク τ_i の k 番目のフレームがスケジュール可能であることの 3 つの条件が同値であることを示す．

補題 1 式 (1) が満たされるとき，タスク τ_i の k 番目のフレームがスケジュール可能である．

証明 1 式 (1) が満たされるとき，$(0, D_i^k]$ の範囲に等号が成り立つ時刻 t_f が存在する．t_f は，対象となるタスク τ_i より高い優先度を持つすべてのタスクが，$(0, t_f]$ の時間に起動され完了した実行に要した時間の総和と，τ_i の k 番目のフレームの実行時間である．つまり，t_f は τ_i の k 番目のフレームの最大応答時間となる．最大応答時間が相対デッドラインよりも小さいので，スケジュール可能である． □

補題 2 タスク τ_i の k 番目のフレームがスケジュール可能であれば，式 (3) を満たす．

証明 2 対偶を示す．つまり，次式が満たされるとき，タスク τ_i の k 番目のフレームがスケジュール可能でないことを示す．

$$\forall\, t,\ \ 0 < t \le D_i^k,\ \ \sum_{j=1}^{i-1} M_j(t) + C_i^k > t$$

また，上式を変形すれば，次式のようになる．

$$\forall\, t,\ \ 0 < t \le D_i^k,\ \ t - \sum_{j=1}^{i-1} M_j(t) < C_i^k$$

MIF の定義と飽和加算の定義より，上式における左辺は広義の単調増加となるから，上式をすべての t ではなく，デッドラインにおいて成立するかを調べればよい．つまり，

$$D_i^k - \sum_{j=1}^{i-1} M_j(D_i^k) < C_i^k$$

ここで，各 j に対して，

$$\exists\, l_j\ \ M_j(D_i^k) = I_j^{l_j}(D_i^k)$$

であるから，一般性を失うことなく次のように書き換えることができる．

$$D_i^k - \sum_{j=1}^{i-1} I_j^{l_j}(D_i^k) < C_i^k$$

上式において，左辺はデッドラインにおける余裕時間を表しており，必要な計算時間よりも小さいため，スケジュール不可能であると示せる．よって，補題は証明された． □

補題 3 式 (3) が満たされれば，式 (1) も満たされる．

証明 3 式 (3) が満たされるとき，$(0, D_i^k]$ の範囲に等号が成り立つ時刻 t_f が存在する．時刻 t_f は，必要な計算時間 C_i^k が消費されるだけの余裕時間を持つ最初の時刻であるから，式 (2) の右辺第 2 項は $[0, t_f]$ では，時刻 t_f で最大となる．

$$\sum_{j=1}^{i-1} M_j(t_f) = t_f - \left(t_f - \sum_{j=1}^{i-1} M_j(t_f) \right)$$

つまり，時刻 t_f においては，次式が成立する．

$$\sum_{j=1}^{i-1} M_j(t_f) = \sum_{j=1}^{i-1} M_j(t_f)$$

よって，式 (3) を満たすような時刻 t_f は，式 (1) においても存在し，補題は証明された． □

3.6 任意の相対デッドラインへの拡張

マルチフレームタスクモデルにおいて，タスクの相対デッドラインが周期と等しいか短いという前提が課せられている．しかし，本研究で対象とするエンジン制御システムの中には，この前提を満たさないタスクも存在するため，このようなタスクも扱うための枠組みが必要となる．タスクの（相対的な）デッドラインが周期より長い場合は，同じタスクが複数，同時に実行可能となる可能性がある．つまり，前に起動されたタスクが，起動周期よりも長い応答時間を持つ場合，次に起動される同じタスクの実行を妨害する．タスクの最大応答時間を求めるためには，前に起動されたタスクが次の起動時刻からはみ出して実行する時間を考慮する必要がある．

このようなタスクをマルチフレームタスクとして取り扱うために，GMF タスクモデルに level-i busy period [4] の概念を新たに適用した．level-i busy period とは，簡単には，優先度 i よりも高いかまたは等しい優先度を持つ，すべてのタスクが同時に起動された時刻 t_0（t_0 でどのタスクも実行されていないという条件もつく）から，そのすべてが終了する時刻 t までの区間 $(t_0, t]$ のことである．

GMF タスクに対して，level-i busy period を適用すると，タスクの最大応答時間に関して次の定理が成立する．この定理の証明は文献 7) で行っているので，こちらを参考にされたい．

定理 5 GMF タスク τ_i の k 番目のフレームの最悪応答時間は，いずれかの critical instnat candidates から始まる level-i busy period の中にある．

具体的に，あるタスクの level-i busy period を求めるには，そのタスクの優先度よりも高いかまたは等しい優先度を持つ，すべてのタスクの MIF を飽和加算することで求めることができる．このとき，相対デッドラインを超えた場合は，明らかにスケジュール可能ではないので，以降の解析をとりやめる．そうでない場合は，level-i busy period の中に対象となるタスクが何回起動されるかを求め，それぞれに対応する応答時間を計算し，最大のものを最大応答時間として決定する．そして，最大応答時間が相対デッドラインより小さい場合はスケジュール可能であり，そうでない場合はスケジュール不可能であると判定できる．

4. 適　　用

本章では，提案した検証手法のエンジン制御システムへの適用事例について述べる．

4.1 システム構成

本研究で対象としているシステムには，ワンチップマイコン内に NBD（Non-Break Debug block）と呼ばれる機構を持たせている．NBD は，マイコン外部から，プロセッサの動作に影響を与えずにマイコン内部のメモリを読み書きするための機構である．システ

ムの振舞いを調べるために，この NBD の機構と，この機構を用いてあらかじめ指定したメモリ領域を周期的に読み込み，記録する装置（以下，記録装置）を用いた．NBD と記録装置は，元来，エンジン制御システムの動作状態をエンジンを動かしながら監視することで各種の制御パラメータの最適化を行うための機構であり，本研究で取得したいタスク切替えなどの離散的なイベントを記録するためのものではない．そのため，これらの機構を用いて取得・記録できるデータ量はきわめて限られている．

4.2 適用環境

本研究では，シミュレータを用いた環境と実車を用いた環境に対して，エンジン制御システムのリアルタイム性の検証を行った．

シミュレータ環境では，ECU とシミュレータ，ECU と記録装置をそれぞれ接続し，イベント情報を取得した．ここでのシミュレータは，ECU に与える入力信号を静的に発生するだけの装置である．このシミュレータを用いることで，回転数をパラメータとしたエンジンの定常状態のデータを比較的簡単に取得することができる．しかし，一定の動作条件に関するデータしかとれないというデメリットがある．

実車環境では，記録装置を搭載した実車を走行させてイベント情報を取得した．実際の走行で発生する生のデータをとることができるが，シミュレータ環境と比較すると非常に大きな時間と手間を要し，簡単にデータを得ることができないというデメリットがある．

シミュレータ環境でも実車環境と変わらず，本研究で提案する手法を適用し，同様の解析結果を得ることができれば，設計段階においてシミュレータを用いてシステムのリアルタイム性に対して簡単な検証を行うことで，テストを簡略化することができると考えている．

4.3 最大実行時間の決定

RMA を適用するためには，各処理の起動タイミングと最大実行時間が分かっていることが前提となる．このうち，起動タイミングはシステムの設計段階に要求される仕様として決定される．一方，正確な最大実行時間は実装されたコードとターゲットに大きく依存し，簡単に見積もることはできない．

最大実行時間を決定する方法は，大きく 2 つに分けられる．実際に実行されるコードから解析する方法と実際に実行して計測する方法である．また，後者の実際に実行して計測する方法として，2 つの方法が考えられる．ある処理単位ごとに全入力を試す方法と，一定時間実行した結果から計測する方法である．全入力を試すことができれば，完全な結果が得られるが，エンジン制御を行う処理は細やかな制御を行うために分岐が多く，そのすべてを試すことは非常に煩雑となり，現実的にはほとんど不可能となる．

そこで，本研究では簡便さの面から，一定時間実行した結果から最大実行時間を計測する方法を採用した．

4.4 イベント情報の取得

システムの時間的振舞いを調べるためには，(1) タスク切替え，(2) タスク起動/終了，(3) タスクの起床/起床待ち，(4) 割込みハンドラの起動/終了に関する情報をそれぞれ区別して記録することが望ましい．しかし，現在の記録装置には取得できるデータ量に制約があるため，取得する情報を制限せざるをえない．そこで，タスク切替えと割込みハンドラの起動に関する情報に的を絞り，これらの情報をメモリ上に記録するように，リアルタイム OS の改造を行った．タスク切替えに関しては，その発生時刻とタスク切替え後のタスク ID を記録し，割込みハンドラに関しては，その起動時刻と割込み要因番号を記録した．

このように，リアルタイム OS に改造を加えることで，わずかにオーバヘッドが増えるが，制御の内容に影響を与えずに，システムの時間的な振舞いを簡単かつ比較的正確に測定することができる．

4.5 タスクの実行時間の算出

タスクの実行時間を算出するために，タスク切替え時に記録された発生時刻を用いた．タスクの 1 回分の実行時間は，タスクの実行が終了した時刻から起動された時刻を引くことで求めることができる．ただし，測定対象のタスクが，より高い優先度を持つタスクによって実行を中断された時間は除く必要がある．

より正確なタスクの実行時間を求めるためには，タスク切替えにともなうオーバヘッドを考慮する必要がある．タスク切替えにともなうオーバヘッドは，低優先度タスクの実行が中断されてから高優先度タスクの実行が開始されるまでの時間と，高優先度タスクの実行が終了してから低優先度タスクの実行が再開されるまでの時間の合計である．タスク切替えは，高優先度タスクの起動により発生するため，これにともなうオーバヘッドは，高優先度タスクの実行時間に含めるべきである．タスク切替えのパターンから低優先度タスクに含まれる（本来含まれるべきでない）オーバヘッドの時間を推定し，各タスクの実行時間からそれを減算した．つまり，タスクの実行時間は，タスクの実行に要した（正味の）時間にタスク切替えにともなうオーバヘッド時間を加えたものとなる．

4.6 割込みハンドラの実行時間の推定

エンジン制御の中でも特に高い応答性が要求される処理は，割込みハンドラの中で実行される．

割込みハンドラの実行時間はタスクの実行時間に含まれ，さらに割込みハンドラは頻繁に起動されるため，スケジュール可能性の精度に悪影響を与える．システムの時間的振舞いをより正確に知るためには，タスクと同様に割込みハンドラの実行時間を知る必要がある．

割込みハンドラの実行時間は，割込みハンドラの終了時刻と起動時刻との差により求めることができる．しかし，割込みハンドラの実行時間はきわめて短いため，現在の記録装置では正確に記録できない．また，単位時間あたりに取得できるデータ量に制限があるため，割込みハンドラに関しては，その起動時刻のみを記録している．そこで，割込みハンドラの実行時間を求めるために，タスクの実行時間の分布からの推定を行った．基本的な考え方は，次のとおりである．ある割込みハンドラが 1 回だけ実行された場合のタスクの実行時間と，どの割込みハンドラもまったく実行されなかった場合のタスクの実行時間との差を，その割込みハンドラの実行時間とする．種類の異なる複数の割込みハンドラが実行された場合でも，同様の方法を繰り返し行うことで求めることができるが，推定の信頼性を高く保つために，起動された割込みハンドラの種類が少ないときのデータを優先して用いている．

この方法で求めた割込みハンドラの時間は，割込みハンドラのみの実行時間ではなく，割込みハンドラの出入口処理（割込みハンドラの実行の前後に必ず行われる処理）も含まれている．

4.7 MIF を用いたスケジュール可能性解析

前述の MIF を用いて最大実行時間によるエンジン制御ソフトウェアのリアルタイム性を検証を行った．

スケジュール可能性解析の具体的な例として，4 ミリ秒タスクと呼ばれるタスクをとりあげる．このタスクは，起動周期と相対デッドラインが 4 ミリ秒である．今回使用した検討用のシステムにおいて，4 ミリ秒タスクはクリティカルなタスクの中でも，低い優先度を持つタスクとして位置付けられている．最も優先度の高いタスクから順に，この 4 ミリ秒タスクまでのリアルタイム性を保証することができれば，クリティカルなタスクのリアルタイム性を保証することになり，システムの性能を保証するための明確な判断基準の 1 つとなる．

図 5 に，エンジン回転数が高回転時（5000 rpm）における，4 ミリ秒タスクよりも高い優先度を持つすべてのタスクの MIF を飽和加算したものを示す．このMIF に 4 ミリ秒タスクの最大実行時間を足し合わせて最大応答時間を求めた．検証の結果，4 ミリ秒タスクの最大応答時間も約 2,700 μ 秒となり，このタスクはスケジュール可能であると判定できる．

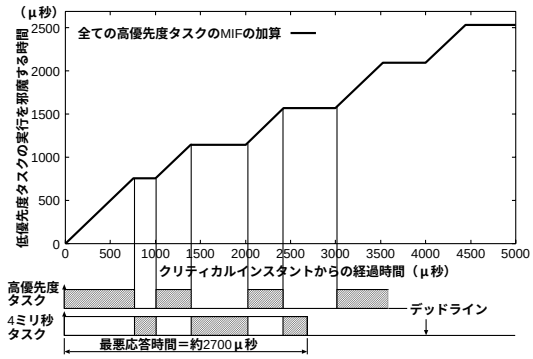

図 5 スケジュール可能性解析の例
Fig. 5 A example of schedulability analysis.

同様に，すべてのタスクについてスケジュール可能性の評価を行った結果，エンジン回転数が 2000 rpm で最低優先度タスクから 1 つ，3000 rpm から 5000 rpm で 2 つ，6000 rpm では優先度の低い方から 3 つのタスクがデッドラインに間に合わずに，そのリアルタイム性を保証できない結果が得られた☆．しかし，これらの低優先度タスクは，エンジン回転全域において時間制約を満たす必要はなく，2000 rpm 付近においておおよその時間制約を満たせば，エンジンの性能も満足できるような処理で構成されている．エンジン回転全域においてデッドラインを満たすことが要求されるクリティカルなタスクについては，現在の最大実行時間が 3 割程度伸ばした場合においても，そのリアルタイム性を保証することが確認できた．最大実行時間を実測で求めるため保証の限界はあるものの，クリティカルなタスクのリアルタイム性については，余裕を持って保証できるという結果が得られた．

4.8 プロセッサ使用率による評価

本研究では，スケジュール可能性解析を行うための前段階として，動作ログより求めたタスクの実行時間と割込みハンドラの実行時間を用いて，すべてのタスクと割込みハンドラのプロセッサ使用率の総和を求めた．この評価を，最大実行時間を用いた場合の評価と実際のシステムの挙動とのずれを見積もるとともに，サブスケジューリングを考慮した場合の解析の精度を見積もる方法の 1 つであると位置付けている．

☆ 実際のエンジンでは，最低優先度タスクは 4000 rpm まで正常に動作していることが確認されている．

まず，サブスケジューリングを考慮しない場合と考慮した場合，それぞれに対してエンジン回転数に対するシステムのプロセッサ使用率を求めた．その結果，サブスケジューリングを考慮した場合はサブスケジューリングを考慮しない場合に比べ，プロセッサ使用率の評価の正確さが 20%程度の向上することが分かった．しかしそれでも，実際のプロセッサ使用率にくらべて，算出したプロセッサ使用率は 30%程度高めであり正確ではない．正確でない理由として，最大実行時間による評価を行い，非周期タスクの周期に設計上の最小起動周期を用いるなど，評価を安全側に倒していることがあげられる．最悪の状況で検証されるリアルタイム性の保証の面から，これらの問題を本質的に回避することはできないと考えている．

4.9 適用結果の評価と課題

今回，実車を用いた環境とシミュレータを用いた環境の両方を用いて，タスクの実行時間の計測を行った．あるタスクの各フレームの平均実行時間，最大実行時間および標準偏差 σ を**表 1** に示す．ただし，各実行時間はシミュレータ環境でのタスクの平均実行時間を 1 として正規化している．

表 1 から分かるように，両者の環境の関係を実車環境で得られたタスクの実行時間の分散（および標準偏差）は，シミュレータ環境で得られたそれより大きいと予想していたが，一概にそうともいいきれない結果が得られた．このような結果が得られた理由として，今回，実車環境で得たデータは可能な限りエンジン回転数を一定に保ち定常状態となるようにして測定したものであり，エンジンの定常状態を再現するシミュレータ環境とほぼ同一の条件になったことがあげられる．また，比較するためのデータが十分に得られていないことも理由の 1 つである．両者の環境の比較を明らかにすることが必要であると考えている．

本研究では，最大実行時間を決定するために，実際に一定時間の実行を行い測定する方法を採用した．この方法では，真の最大実行時間を知ることができないため，リアルタイム性の保証に限界はある．しかしながら，クリティカルなタスクについてのリアルタイム性を余裕を持って保証できれば，テストを簡略化することができるという点において意味があると考えている．より正確な最大実行時間を求める方法は今後の課題としてあげられる．

表 1 タスクの実行時間と標準偏差
Table 1 The execution time and the standard deviation of a task.

	シミュレータ環境			実車環境		
フレーム	平均	最大	σ	平均	最大	σ
1	1.00	1.35	0.09	0.98	1.20	0.04
2	1.00	1.26	0.08	0.99	1.28	0.06
3	1.00	1.27	0.07	0.94	1.05	0.03
4	1.00	1.47	0.09	0.99	1.29	0.06

最大実行時間を決定するもう 1 つの方法として，実際に実行されるコードから解析する方法がある．この方法は Worst-Case Execution Time（WCET）解析と呼ばれていて，与えられたコード片の実行時間の上限を計算する方法である[8]．具体的には，実行されうるコードの中にあるすべてのパスについて実行時間を求め，その最大を最大実行時間とする方法をとる．当然ながら，この手法はハードウェア依存であり，ターゲットとなるハードウェアごとに，その特徴をモデル化することが必要となる．つまり，パイプラインやキャッシュを考慮する必要があるが，パイプラインストールやキャッシュミスまでも考えた場合，WCET 解析により得られた結果は非常に悲観的となる．また，解析を効率化するためには，意味論的なコードの解釈も必要できわめて難しくなる．解析するためのツールも研究段階としてはあるが，きわめて限定されたターゲットのみで使用可能で実用段階にはない．解析ツールの汎用化が課題の 1 つとしてあげられる．

5. 結　　論

本論文では，エンジン制御システムのリアルタイム性を検証するために，既存の RMA を拡張した手法を提案し，その適用事例を紹介した．本検証手法を用いて実際のエンジン制御システムのスケジュール可能性を検証した結果，クリティカルな処理のリアルタイム性を保証することができた．本研究で提案するリアルタイム性検証手法を用いることで，設計段階においてエンジン制御システムが時間的制約を満たすかどうかを検証することができる．これにより，クリティカルな処理のリアルタイム性を保証でき，定量的な指標によるシステムの品質についての議論が可能となるだけでなく，エンジン制御システムの検証プロセスの効率化を期待できる．

本研究は，エンジン制御システムを題材とした RMA の拡張と応用に関する研究であると位置付けることができる．本論文で行った RMA の理論的な拡張は，エンジン制御システムに特化して行ったものではなく，より一般的なシステムに適用できるようしたもので，タスクモデル自体の一般化である．いい換えると，本研究で行った拡張によって，従来の RMA で取り扱うことのできたシステムだけでなく，エンジン制御システムを含めた，より広範なシステムを RMA の枠組み

で扱うことができるようになる．よって，本研究ではエンジン制御システムを対象として取り扱っているが，本研究で提案しているリアルタイム性検証手法は，エンジン制御システムだけでなく，他のシステムへ適用できる可能性が十分にあると期待できる．

参 考 文 献

1) Mok, A.K. and Chen, D.: A multiframe model for real-time tasks, *Proc. Real-Time Systems Symposium*, pp.22–29 (1996).
2) Baruah, S., Chen, D., Gorinsky, S. and Mok, A.: Generalized multiframe tasks, *Real-Time Systems*, Vol.17, No.1, pp.5–22 (1999).
3) 高田広章，坂村　健：マルチフレームタスクセットの静的優先度スケジューリングによるスケジュール可能性，信学技報（1997 年実時間処理に関するワークショップ RTP '97），Vol.96, No.596, pp.29–35 (1997).
4) Lehoczky, J.P.: Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines, *Proc. Real-Time Systems Symposium*, pp.201–209 (1990).
5) Takada, H. and Sakamura, K.: Schedulability of generalized multiframe task sets under static priority assignment, *Proc. Real-Time Computing Systems and Applications*, pp.80–86 (1997).
6) Liu, C.L. and Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *J. ACM*, Vol.20, No.1, pp.46–61 (1973).
7) 飯山真一，遠藤友悟，高田広章，菅沼英明：RMA 手法のエンジン制御システムへの適用に関する研究，信学技報（2001 年実時間処理に関するワークショップ RTP2001），Vol.100, No.655, pp.7–14 (2001).
8) Puschner, P. and Burns, A.: A Review of Worst-Case Execution-Time Analysis (editorial), *Real-Time Systems*, Vol.18, pp.115–128 (2000).

(平成 13 年 12 月 17 日受付)
(平成 14 年 4 月 16 日採録)

飯山　真一

2002 年豊橋技術科学大学大学院情報工学専攻修士課程修了．現在，同大学院電子・情報工学専攻に在学．リアルタイムスケジューリング理論とその応用に関する研究に従事．修士（工学）．

高田　広章（正会員）

豊橋技術科学大学情報工学系助教授．1988 年東京大学大学院理学系研究科情報科学専攻修士課程修了．同学科の助手等を経て，1997 年 12 月より現職．リアルタイム OS，リアルタイムスケジューリング理論，組込みシステム開発技術等の研究に従事．ITRON 仕様の標準化活動に，中心的メンバとして参加．博士（理学）．IEEE，ACM，電子情報通信学会，日本ソフトウェア科学会各会員．

菅沼　英明

トヨタ自動車（株）第 2 電子技術部（東富士研究所）．1994 年東京農工大学工学部機械システム工学修士課程修了．同年トヨタ自動車（株）入社．以来，主にエンジン制御用ソフトウェアの開発および開発環境の研究開発に従事．