Vol.2010-SE-168 No.2
Vol.2010-EMB-17 No.2
2010/6/1

DUOS:車載ECU統合向けRTOSフレームワーク

長尾卓哉[†1]　山田真大[†1]　石谷健[†1]
松原豊[†1]　山崎二三雄[†1]
本田晋也[†1]　高田広章[†1]

近年，自動車制御の高機能化，高性能化に伴い，車載制御システムに搭載される電子制御ユニット（ECU）の数が増加し続けている．搭載される ECU の数を減らすために，別々の ECU で個別に動作していた複数のリアルタイムアプリケーションを，単一の高性能な ECU に統合して動作させる手法が検討されている．本論文では，ECU 統合におけるアプリケーション統合の負荷をできる限り低減することを目的に，車載 ECU 統合向け RTOS フレームワークを提案する．提案フレームワークは，アプリケーション間の時間保護を実現する階層型スケジューラをベースに，μITRON 仕様と OSEK OS 仕様に準拠する API を両方備えたデュアル API をもつ．プロトタイプシステム DUOS を実装し，API 実行時間とタスク切替え時間を評価した結果，実用上許容できるオーバヘッドであることを確認した．

DUOS: A Real-Time OS Framework for Integrating Electronic Control Units in Automotive Control Systems

TAKUYA NAGAO,[†1] MASAHIRO YAMADA,[†1]
TAKESHI ISHITANI,[†1] YUTAKA MATSUBARA,[†1]
FUMIO YAMAZAKI,[†1] SHINYA HONDA[†1]
and HIROAKI TAKADA [†1]

The number of Electronic Control Units (ECUs) in automotive control systems has been continuously increasing due to the requirement from the advanced electronic control functions. For the purpose of reducing this number, several approaches has been studied to integrate ECUs into a high performance one and unite applications on it. This paper presents a new RTOS framework for integrating ECUs to reduce the amount of comprehensive verification works at the integration of ECUs. The RTOS framework includes a hierarchical scheduler, and API layers for μITRON OS and OSEK OS specification. The results of the evaluation of the prototype system, called DUOS, indicate that the overhead of execution time of API and task switching are reasonably acceptable for the real-time applications.

1. はじめに

車載制御システムは，ECU 間をネットワーク（ワイヤハーネス）で接続し，分散するアプリケーションを連動させることで，より高度で複雑な制御を実現している．自動車制御の高機能化，高性能化に伴い，車載制御システムに搭載される電子制御ユニット（ECU）の数が増加し続けており，リアルタイム性を要求される制御ソフトウェア（アプリケーション）も大規模，複雑化する傾向にある．自動車に搭載される ECU の数とワイヤハーネスが増加する中で，今後のさらなる高機能化を考えると，ECU やワイヤハーネスを設置するためのスペースが不足することに加えて，自動車制御システムのハードウェアコストの増加，アプリケーションの設計と検証の負荷の急激な増加が大きな問題となっている．このような問題を解決するため，車載制御システムに搭載される ECU の数を減らす車載 ECU 統合の必要性が高まっている．

車載 ECU 統合においては，出来る限り既存の機能を維持したまま，複数の ECU を単一の ECU へ集約する技術が必要となる．これまで，車載 ECU 統合に適用可能な技術が数多く提案されているが，我々は，別々の ECU で個別に動作していた複数のリアルタイムアプリケーションを，単一の高性能な ECU に統合して動作させるためのアプリケーション統合手法を検討している．

アプリケーション統合においては，統合前に正常動作しているアプリケーションが，統合後も正常に動作することを検証する必要がある．特に，リアルタイム性を要求されるアプリケーションの場合は，機能的な動作が正しいことに加えて，時間制約を満たして動作が完了することを検証する必要があるため，複数のアプリケーションが同一の ECU で動作するようなる統合時の検証（統合検証）の負荷が高いという問題がある．また，異なる RTOS で動作するアプリケーションを統合する場合には，統合後のアプリケーション動作環境に対応させる必要がある．具体的には，使用する API を始めとして，アプリケーション内部のタ

†1 名古屋大学 大学院情報科学研究科 附属組込みシステム研究センター
Center for Embedded Computing Systems, Nagoya Univ.

Vol.2010-SE-168 No.2
Vol.2010-EMB-17 No.2
2010/6/1

スク設計，実装方法を修正する必要があり，場合によっては，アプリケーションを再設計し直すような大きな修正が必要となる．

本論文では，車載 ECU 統合において，アプリケーションをできる限り修正することなく統合するための RTOS フレームワークを提案する．提案フレームワークは，アプリケーション単位のプロセッサ時間保護を実現することで，統合検証の負荷を低減する階層型スケジューラ[1),2)] をベースに，統合後の動作環境へのアプリケーションの移行を助けるために，μITRON 仕様[3)] と OSEK OS 仕様[4)] に準拠する API を呼び出すことができる複数 API 機能をもつ．さらに，提案フレームワークを適用したプロトタイプシステムである DUOS（Dual API Real-time OS）を実装して，評価する．

以下，本論文の構成を述べる．まず，２章で車載 ECU 統合向け RTOS の要件を整理する．３章では，その要件を満たす RTOS フレームワークについて述べる．４章では，DUOS の実装について述べ，５章でオブジェクトサイズと API 実行性能を評価する．６章で関連研究について述べる．最後に，７章で結論と今後の課題を述べる．

2. 車載 ECU 統合向け RTOS の要件

2.1 想定する ECU 統合手法

本論文で想定している ECU 統合手法を説明する．**図 1** は，二つの ECU を単一の ECU に統合する例である．統合前に，ECU-A では動作周波数 30MHz のシングルプロセッサ上に RTOS-A を載せて，アプリケーション A を動作させている．同様に，ECU-B では 40MHz のシングルプロセッサ上に RTOS-B を載せて，アプリケーション B を動作させているとする．この二つの ECU を 100MHz のシングルプロセッサをもつ ECU-C（統合 ECU）に統合する．ここで，統合前の個別の ECU で正常に動作していたアプリケーション A とアプリケーション B のみを統合 ECU である ECU-C に移行することに注意されたい．ECU 統合に際して，RTOS-A と RTOS-B は，ECU-C に移行しないことから，我々はこの統合手法をアプリケーション統合と呼んでいる．また，RTOS-A と RTOS-B は，異なる種類の RTOS であるものとする．提案手法は，ECU-C で動作する RTOS-C に適用することを想定している．なお，本論文では，アプリケーション間のメッセージ通信，メモリやデバイスの共有はないものとする．

2.2 アプリケーション統合における課題

車載 ECU で動作するアプリケーションは，リアルタイム性を要求される制御を実現するために時間制約をもつ場合が多い．個別の ECU で時間制約を満たして動作するアプリケーションを容易に統合するためには，統合の前後において，アプリケーションの動作が同じように振る舞うことが望ましい．しかし，複数のアプリケーションを統合 ECU で動作させる状況において，あるアプリケーションに不具合が発生した場合に，想定よりも，そのアプリケーションが多くのプロセッサ時間を使用してしまう可能性がある．その結果，別のアプリケーションが動作するための時間がなくなり，時間制約を満たせなくなってしまうという問題がある．そこで，アプリケーションごとのプロセッサ時間を保護することが課題となる．また，異なる種類の RTOS で動作するアプリケーションを統合する場合には，使用する API やタスク優先度割当てなどを再設計する必要がある．高信頼性が求められるアプリケーションにおいては，動作検証が完了している既存の実装（ソースコード）を大きく修正すると，再度動作検証を実施する必要があるため，統合に際して，出来る限り既存アプリケーションの修正を少なくすることが望ましい．

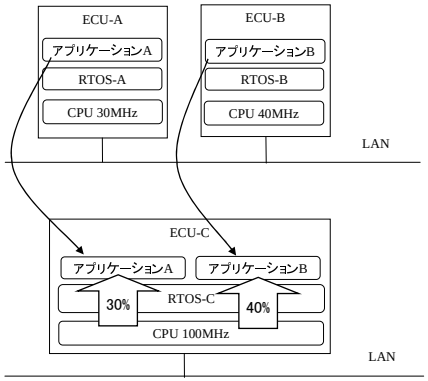

図 1 ECU 統合
Fig. 1 Integration of ECUs

2.3 RTOS の要件

ECU 統合を目的としたアプリケーション統合の課題を議論した結果，本論文では，統合 ECU で動作する RTOS が備えるべき機能要件を次の二つに整理する．

- アプリケーション単位のプロセッサ時間保護機能をもつこと
  統合 ECU で動作するアプリケーション間で，時間的な不具合が波及することを防ぐために，アプリケーション単位の時間保護機能をもつことを要件とする．この要件によ

Vol.2010-SE-168 No.2
Vol.2010-EMB-17 No.2
2010/6/1

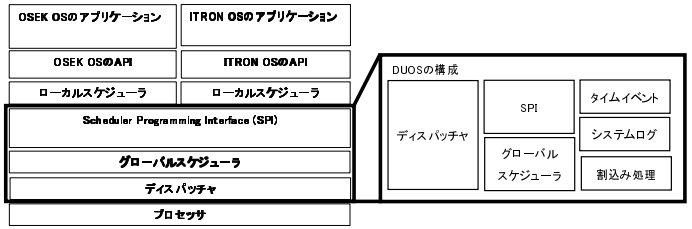

図 2 提案フレームワークの構成
Fig. 2 Composition of the proposed RTOS framework.

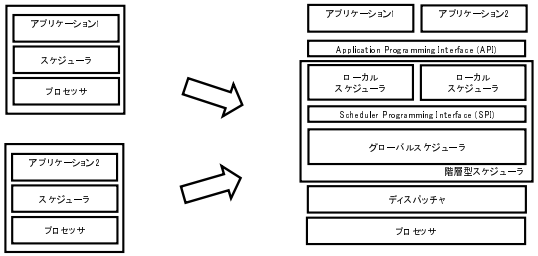

図 3 階層型スケジューラの構成
Fig. 3 Composition of the hierarchical scheduler

り，統合 ECU で発生する不具合の原因を，アプリケーションごとに切り分けられるため，統合検証の負荷を減らすことができると考える．

- 複数種類の API をサポートすること
  アプリケーション内部のタスクの構成，優先度の割当て設計，呼び出す API を変更することなく統合するために，複数種類の OS の API をサポートすることを要件とする．サポートすべき API は統合するアプリケーションによって異なるため，システム構築時にアプリケーションが使用する API を選択できるものとする．この要件により，アプリケーションを統合 ECU に移行する負荷を減らすことができると考える．

3. 車載 ECU 統合向け RTOS フレームワーク

3.1 概　　要

提案フレームワークは，図 2 に示すように，RTOS のタスクスケジューラを階層化した階層型スケジューラ，複数 API をサポートするための API レイヤ，さらに，OS 共通機能であるシステムログ，タイムイベント処理機能等をもつ．本章では，各機能について詳細に説明していく．

3.2 階層型スケジューラ

アプリケーション単位の時間保護機能を実現するために，提案フレームワークでは，階層型スケジューラを採用する．階層型スケジューラは，**図 3** に示すように，アプリケーションをスケジュールするグローバルスケジューラと，アプリケーションのタスクをスケジュールするローカルスケジューラを階層的にもつ．

ローカルスケジューラとは，アプリケーション内の処理単位（タスク，割込みサービスルーチンなど）をスケジューリングする機構である．各アプリケーションのタスクのスケジューリングのアルゴリズムは，単一のプロセッサへ統合する前と同じである．グローバルスケジューラは，実行すべきアプリケーションを決定するためのスケジューリング機構と，アプリケーションに対して設計段階で設定されるプロセッサの利用率に応じてプロセッサ時間（これをバジェットと呼ぶ）を補充する機能をもつ．

グローバルスケジューラの代表的なアルゴリズムとして，重み付きラウンドロビン方式，固定デッドライン EDF 方式，変動デッドライン EDF 方式など（**図 4**）がある．

- 重み付きラウンドロビン方式
  システム全体で，アプリケーションの起動周期を同一にして，周期ごとにすべてのアプリケーションを順番に実行する．各アプリケーションの実行時間は，それぞれの重み（プロセッサ利用率）に比例して分配される．
- 固定デッドライン EDF
  アプリケーションごとに，起動周期（デッドライン）を設定し，デッドラインの早い順番に，アプリケーションを実行する．実行中に実行周期を変更することはできない．
- 変動デッドライン EDF
  アプリケーションごとに，実行中にデッドラインを計算し，デッドラインの早い順番にアプリケーションを実行する．

本研究で実装したプロトタイプシステム DUOS では，スケジューリング処理が単純である固定デッドライン EDF 方式を採用している．アプリケーションのタスク構成がハーモニックタスクセットで，統合前に個別の ECU で時間制約を満たしていれば，単一の ECU へ統合した後も時間制約で動作することを保証できる[1)]．

Vol.2010-SE-168 No.2
Vol.2010-EMB-17 No.2
2010/6/1

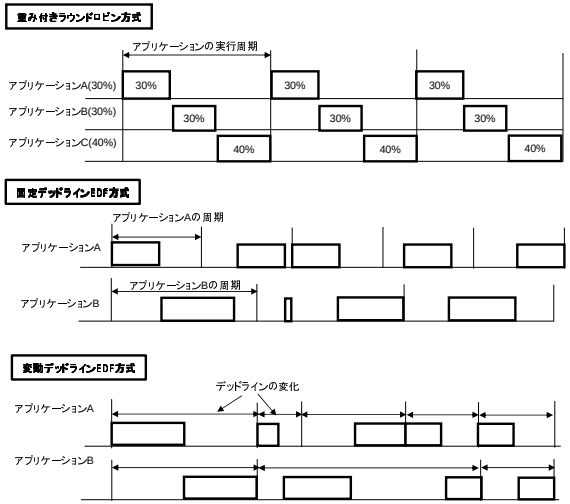

図 4 グローバルスケジューラのスケジューリング方式
Fig. 4 Three types of scheduling method for global scheduler

ローカルスケジューラとグローバルスケジューラの間で情報をやり取りするためのインタフェースとして SPI（Scheduler Programming Interface）がある．SPI には，ローカルスケジューラがグローバルスケジューラに対して情報を通知するための情報通知機能と，ローカルスケジューラがグローバルスケジューラの管理する情報を参照するための情報参照機能がある．SPI のリストを**表 1** に示す．

3.3 API レイヤ

提案フレームワークでは，複数種類の API をサポートするために，API ごとに API レイヤをもつ．本論文では，サポートする API として，μITRON 仕様[3] と OSEK OS 仕様[4] の 2 種類を選択する．μITRON 仕様は，高機能で比較的規模の大きいリアルタイムアプリケーションを統合対象に，一方，OSEK OS 仕様は，車載制御システムの小規模な制御アプリケーションを統合対象と想定している．

API レイヤは，**図 2** に示したように，アプリケーションとローカルスケジューラの間に位置しており，RTOS の基本機能を実現するものである．アプリケーションのタスクをスケジューリングする場合には，ローカルスケジューラと一体になり SPI を呼び出すことで，タスク切替えを実現する．API レイヤが実現すべき機能は，サポートする API の種別によって異なるが，およそ以下の機能をアプリケーションに対して提供する．具体的な例は 4 章で述べる．

表 1 SPI の機能一覧
Table 1 List of SPI functions

機能名	詳細
情報通知機能	実行すべきタスクの通知
	アプリケーションのデッドラインの通知
	タスクディスパッチ（コンテキスト保存あり）
	ディスパッチ（コンテキスト保存なし）
	ディスパッチ要求のセット
	ディスパッチ可能状態のセット
情報参照機能	実行中タスクの取得
	実行中割込みハンドラの取得

- 共通データ型定義
- タスク管理機能
- 割込み管理機能
- タスク間同期機能
- タイムイベント機能

3.4 共通機能

提案フレームワークでは，アプリケーション，階層型スケジューラ，API レイヤが，共通で利用できる機能を共通機能としてもつ．以下では，共通機能について述べる．

- システムログ

アプリケーションが任意の文字を出力する機能や，システムの動作履歴（ログ）を出力する機能を，システムログ機能としてもつ．

- タイムイベント処理

システム時刻の管理に加えて，システム時刻に応じて処理をするタイムイベント処理機能をもつ．

- 割込み管理処理

割込み要求を受付けて，割込み要因に対応する処理を実行する機能（割込み入り口処理）と，割込み処理から，割込み発生時の処理に戻る機能（割込み出口処理）である．

- ディスパッチャ

実行中タスクを切替える機能である．実行中タスクのコンテキスト（レジスタ，スタック

Vol.2010-SE-168 No.2
Vol.2010-EMB-17 No.2
2010/6/1

ポインタなど）を保存し，実行を開始するタスクのコンテキストを復帰する．ローカルスケジューラから SPI を呼び出すことで，ディスパッチャに対してディスパッチを要求できる．

- システムコンフィギュレーション

システム開発者は，アプリケーションを構成するタスク，セマフォ，フラグなどの OS 資源を，システム構築時にシステムコンフィギュレーションファイルに記載する．コンフィギュレーションツールは，システムコンフィギュレーションファイルから，OS 資源の定義，テータ領域等が含まれるカーネルソースコードを出力する．通常，コンフィギュレーションファイルとツールの仕様は RTOS ごとに異なるが，提案フレームワークでは，拡張可能なコンフィギュレーションツールを用いることで，すべてのアプリケーションのコンフィギュレーション情報を，同一のコンフィギュレーションツールで処理できる．具体的な例は次章で述べる．

4. 実　　装

4.1 DUOS の概要

はじめに，用語を定義する．アプリケーションに関する情報を管理するための管理ブロックをアプリケーション管理ブロックと呼ぶ．タスクのスケジューリングに関する情報を管理する管理ブロックをローカルスケジューラ管理ブロックと呼ぶ．これらの管理ブロックと SPI をまとめて API 共通部と呼ぶ．

DUOS の実装は，TOPPERS/ASP カーネルを階層型スケジューラへ拡張したシステムをベースとして行った．実装の方針としては，ベースとしたカーネルへの処理の共通化をできる限り増やして，API 実行性能を優先させた．

4.2 DUOS の構成

DUOS は，API 共通部，API レイヤ，ターゲット依存部から構成される．それぞれについて説明する．

API 共通部は，アプリケーション管理ブロック，ローカルスケジューラ管理ブロック，SPI から構成される．API レイヤは，統合前のサービスコールを同様に提供するシステムサービス群で，アプリケーションに対して 1 対 1 で用意される．ターゲット依存部は，割込み出入り口処理，割込みハンドラ，ディスパッチャなどで構成される．

4.3 API 共通部

以下に API 共通部の実装について説明する．

4.3.1 アプリケーション管理ブロック

アプリケーション管理ブロックは，アプリケーションに関する情報を管理するための管理ブロックとしてアプリケーション毎に生成する．主な情報として，アプリケーションの静的情報（プロセッサ利用率，起動周期），そのアプリケーションで実行中のタスクのポインタ，最高優先順位のタスクへのポインタ，アプリケーションの残りのバジェットを持つ．

4.3.2 ローカルスケジューラ管理ブロック

ローカルスケジューラ管理ブロックは，ローカルスケジューラがアプリケーション内の処理単位をスケジュールする際に必要な情報を管理するための管理ブロックで，ローカルスケジューラ毎に生成する．主な情報として，ローカルスケジューラ種別，実行状態のタスクを優先度毎に管理するタスクレディキュー，実行可能なタスクのうち最高優先順位のタスクのポインタをもつ．また，ローカルスケジューラ毎の固有の情報として，μITRON 仕様のディスパッチフラグなどもこのテーブルで管理する．

4.3.3 SPI

SPI の実装について，表 1 の実行すべきタスクの通知を例に説明する．この機能は，API レイヤで，タスクの状態が変更された時などに呼ばれるインターフェースである．具体的には，アプリケーション管理ブロックへ最高優先度順位のタスクを登録し，登録後は，必要に応じてアプリケーションの状態を移行させる処理を呼び出す処理として実装した．

4.4 API レイヤ

ITRON OS のタスク起動の API(*act_tsk*) を例に説明する．従来の *act_tsk* では，タスクを起動させる時，最高優先順位のタスクをグローバル変数へ設定していた．DUOS では，API レイヤへ最高優先順位のタスクを SPI の機能（実行すべきタスクの通知）と連携してアプリケーションへ通知する処理を実装した．

4.5 ターゲット依存部

ターゲット依存部の割込み出入口処理，ディスパッチャ，タイムイベント処理の実装について説明する．

割込み出入り口処理は，ITRON OS，OSEK OS で共通処理とした．しかし，OSEK OS では，固有の情報を退避/復帰する処理がある．この処理は，ITRON OS には無い処理である．そのため，共通の割込みハンドラの出入り口処理へアプリケーションを意識した処理のための復帰/退避を実装した．これにより，ITRON OS は冗長な処理が追加されたことになるが，動作としては影響でない実装（参照はするが，使用しない）とした．

ディスパッチャも同様に，ITRON OS，OSEK OS で共通処理とした．ディスパッチャ

Vol.2010-SE-168 No.2
Vol.2010-EMB-17 No.2
2010/6/1

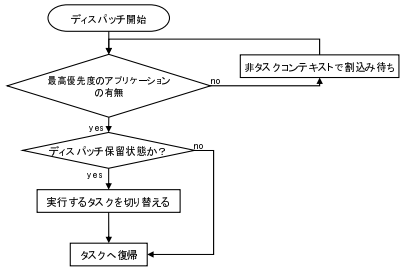

図 5 DUOS のディスパチャ
Fig. 5 Dispatcher of DUOS

は，タスクの切替えのルーチンであるため，最高優先度のアプリケーションのチェックと，アプリケーション内のタスクの実行処理が必要である．そこで，**図 5** で示すフローで設計を行い，ディスパッチ処理へアプリケーションのための判断処理と実行すべきタスクの切替え処理を実装した．

各アプリケーションでバジェットを使い切ったことを監視するためのタイマとして，専用のハードウェアタイマを割り当てた（バジェットタイマ）．バジェットタイマに各アプリケーションのバジェットを設定し，バジェットタイマが満了するとアプリケーション切替え処理を行う．

4.6 システムコンフィギュレーション方法

コンフィギュレータとは，カーネルやシステムサービスの構成や初期化情報を持ったファイルなどを生成するツールである．DUOS では，システムコンフィギュレーションは TOPPERS 新世代カーネル統合仕様[5] の静的 API を利用する．

このコンフィギュレータのテンプレートファイルを追加することで，自由に静的 API を追加可能である．DUOS では，このコンフィギュレーションを利用して，アプリケーションの定義を行えるように静的 API を追加した．以下に追加した静的 API の説明をする．

- アプリケーションの定義

以下に示すように，アプリケーションを定義し，アプリケーションの括りの中に属するカーネルオブジェクトを定義する．

```
/*
* 周期実行アプリケーション（ITRON カーネル用アプリケーション）
*/
APPLICATION(MAINAPP) {
  CFG_APP({ TA_ASP, TA_CYC, 30, 100 });
  CRE_TSK(TASK1,{TA_NULL,1,task1,MID_PRIORITY,TA_SCHFULL,STACK_SIZE,NULL,
            TA_NULL,0,0});
  CRE_TSK(TASK2,{TA_NULL,2,task2,MID_PRIORITY,TA_SCHFULL,STACK_SIZE,NULL,
            TA_NULL,0,0});
  CRE_TSK(MAIN_TASK,{TA_ACT,0,main_task,MAIN_PRIORITY,TA_SCHFULL,
            STACK_SIZE,NULL,TA_NULL,0,0});
}
```

図 6 コンフィギュレーションファイル
Fig. 6 configuration file

```
APPLICATION(”アプリケーション名”) {
    CFG_APP(...)
    CRE_TSK(...)
    ...
}
```

- アプリケーションの属性設定

以下に示すようにアプリケーションの属性は CFG_APP で定義を行う．第一引数で使用する API を指定し，第二引数でグローバルスケジューラのアルゴリズムを指定する．第三引数でプロセッサ利用率を指定し，第四引数でアプリケーションの起動周期（ms）を指定する．

```
CFG_APP(”API の指定”, ”スケジューラのアルゴリズム”, ”CPU 利用率”, ”周期”);
```

図 6 にコンフィギュレーションの例を示す．ここでは，TA_ASP の API を使用して，CPU 利用率 30%，周期 100ms の周期実行アルゴリズムを用いたアプリケーションの定義で，このアプリケーションには，main_task，task1，task2 が属していることを表す．

5. 評　　価

本章では，実装した DUOS の性能を評価する．評価用のターゲットボードは，ARM Cortex-A9 MPCore の 1 コアモデルで，コアクロックは 400MHz，RAM を 512MB（動

Vol.2010-SE-168 No.2
Vol.2010-EMB-17 No.2
2010/6/1

表 2 カーネルのオブジェクトサイズ (byte)
Table 2 Size of object files(byte)

	text	data	bss	total
ITRON OS	35,579	0	4	35,583
OSEK OS	23,532	0	2	23,534
DUOS	59,954	0	6	59,960

表 3 API 実行性能 (μs)
Table 3 API execute performance

	Native		DUOS	
	ディスパッチ無し	ディスパッチ有り	ディスパッチ無し	ディスパッチ有り
ITRON API	8.0	11.4	10.5	22.6
OSEK API	10.3	14.2	12.1	24.4

作周波数 250MHz）搭載の評価ボードである．コンパイラは，ARM 用 GCC コンパイラ 4.3.3(arm-none-eabi-gcc) を使用した．システムコンフィギュレーションは，ITRON OS のアプリケーションの起動周期を 100ms，CPU 利用率を 30%と指定し，OSEK OS のアプリケーションの起動周期を 200ms，CPU 利用率を 30%とした．評価は，DUOS でサポート対象とした ITRON OS と OSEK OS と DUOS のカーネルオブジェクトサイズと API 実行性能について実施した．API 実行性能は，タスクを起動する API の *act_tsk*(ITRON OS) と，*ActivateTask*(OSEK OS) を測定対象とした．API 実行性能は 10 万回測定した結果の平均値とした．性能測定に利用したツールのオーバーヘッドは，10 万回の測定の平均値で 708ns である．

5.1 カーネルのオブジェクトサイズ

DUOS の実装によるカーネルのオブジェクトサイズを評価する．DUOS でサポート対象とした ITRON OS と OSEK OS と DUOS のカーネルオブジェクトのサイズを**表 2** に示す．DUOS の text 領域のサイズは ITRON OS と比べて 24KB の増加，OSEK OS と比べて 36KB と増加している．それぞれの OS から text 領域が増加しているのは，DUOS で API を 2 つもったことと，スケジューラを階層化したことなどによるソースコードの増加が原因である．しかし，全体のカーネルオブジェクトサイズは，ITRON OS と OSEK OS の合計サイズ（59117byte）と比較して 2%の増加に留まっている．これは，VM 方式のように複数の OS を VM 上にそのまま載せた場合と比べて優位性はあると考えられる．

5.2 実 行 性 能

5.2.1 API の平均性能

表 3 に API の実行時間について測定した結果を示す．ここで，表 3 に示す Native とは，DUOS でサポート対象としたオリジナルの ITRON OS と OSEK OS のことを指す．測定した内容は，API を実行した時にタスクの切替え処理が動作しない場合の性能とタスク切替え処理が動作する場合の性能を測定した．表 3 ではそれぞれをディスパッチ無し，ディスパッチ有りと表記した．ディスパッチ有りの性能は，タスク起動の API の実行から，タスクが起動するまでの間を測定した．

はじめに，ITRON OS と OSEK OS の性能差について説明する．ITRON OS と OSEK OS のディスパッチしない時の性能差 (8.0μs と 10.3μs) が生じているのは，それぞれの API の実装の違いから，実行ステップ数が異なることに起因している．その他の性能でも，同様の傾向がみられるのは同じ理由からである．

DUOS の各 API と Native の各 API のディスパッチを伴わない API 実行性能は，DUOS が 2μs 遅くなっている．これは，階層化のための処理が追加されていることに起因する．また，ディスパッチを伴う場合には，10μs 遅くなっている点も，同様に階層化のための処理が追加されたことに起因している．次に DUOS の各 API のディスパッチ無しの場合の性能について比較する．ディスパッチ有りの場合と比べて，2.1 倍となった．これは，ディスパッチ処理と，階層化のために実装した処理（ローカルスケジューラ，API レイヤ，SPI）のオーバーヘッドに起因している．

Native の ITRON API，OSEK API と DUOS の API の性能を比較して，ディスパッチ無しの場合は，2μs のオーバーヘッドが生じた．ディスパッチ有りの場合でも，10μs のオーバーヘッドである．このオーバーヘッドは数十 ms もしくは数百 ms で動作するアプリケーションへの影響は 1%以下であるため，実用上許容できると考える．

5.2.2 システムタイマ割込み時の API 実行性能

次に DUOS のディスパッチが発生しない時の測定結果の分布を**図 7** に示す．以下，DUOS の ITRON API の実行性能を例に 52μs 付近の分布について説明する．10 万回の測定で，割込み処理などが入らない場合，API の実行の測定にかかる総時間はおよそ 1.0s である．一方で，システムタイマは 1ms 毎に割込みを発生させているため，1.0s 間に，およそ 1000 回の割込みが発生する．測定結果からは，52μs 付近の遅延が発生した API は 1084 回であった．これはシステムタイマの割込みの回数にほぼ一致する．これらの結果から，52μs 付近の API 実行性能はシステムタイマの割込み処理により，割り込まれた結果，遅延していることがわかる．また，DUOS の OSEK API では，1289 回の遅延が発生した．DUOS の

Vol.2010-SE-168 No.2
Vol.2010-EMB-17 No.2
2010/6/1

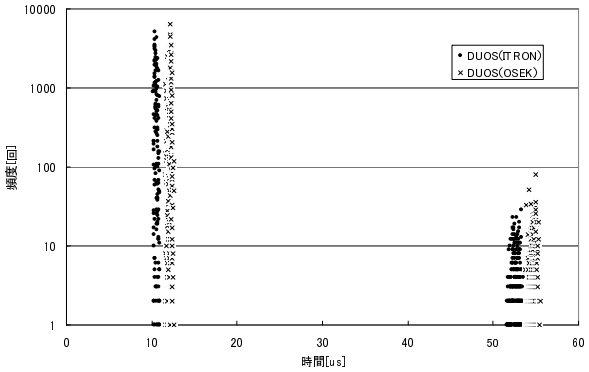

図 7 API 実行性能 (ディスパッチ無し)-1
Fig. 7 API execution performance(Not execute dispatch)-1

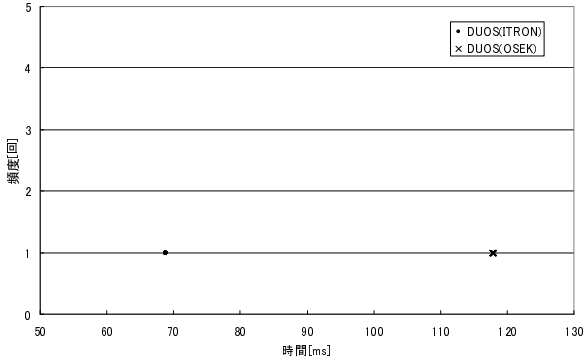

図 8 API 実行性能 (ディスパッチ無し)-2
Fig. 8 API execution performance(Not execute dispatch)-2

OSEK API では，ディスパッチ無しの条件での API 性能は 12μs である．このことから，DUOS の OSEK API の遅延もシステムタイマの割込み処理によるものである．

DUOS では，システムタイマの割込み処理の時，バジェットタイマを停止しているため，この間はバジェットを消費しない．そのため，アプリーションの CPU 利用率に対する影響はないと考える．

5.2.3 アプリケーションの切替り時の API 実行性能

次に，**図 8** について説明する．図 8 は，測定した結果が，図 7 の分布から大きく外れている部分をクローズアップした図である．本結果は，測定結果のオーダーが大きかったため，表示単位を ms とした．

図 8 の API の実行性能は，68ms と 120ms 付近に集中している．前述したとおり，割込みなどが発生しない条件では，測定に 1.0s かかる．一方で，ITRON OS のアプリケーションは 100ms 周期で CPU 利用率を 30%とした．そのため，アプリケーションが 1 回のスケジューリングで動作可能な時間は 30ms である．この時，30ms の間に測定可能な回数は，3000 回である．残りは，次回以降のアプリケーションの CPU 時間の割り当ての時に実行する．つまり，10 万回の測定では，33 回程度のアプリケーション切替えが発生する．アプリケーションの切替えのための割込み処理が動作すると，次回の割り当てまでは，70ms の間隔が空くことなる．今回の測定から，68ms の遅延を 30 回計測した．これは，上記の計算からアプリケーションの切り替わりが 33 回程度発生することに一致する．これらの結果から，68ms 付近の API 実行性能は CPU 利用率を使い切った結果，アプリケーションが切り替ったことによる遅延であることがわかる．また，測定結果から，120ms 付近での遅延は 25 回計測した．これは，68ms の時と同じ計算方法から 20 回程度は発生する．これにより，120ms の遅延原因は，68ms の場合と同じことがわかる．

アプリケーションの処理で考えた場合，統合前のプロセッサ時間を割り当てて動作するため，本事象は影響ないと考える．

5.2.4 アプリケーション切替え時間

DUOS における，アプリケーション切替えにかかるオーバーヘッドを測定した．測定は 100 回行い，その平均値を求めた．測定した結果，アプリケーション切替えのオーバーヘッドの平均値は 30.6μs となった．5.3.1 で述べた，ディスパッチを伴う API の処理時間が 22.6μs で，その 1.3 倍程度でアプリケーションの切替えが行われているため，実用上許容できると考える．

6. 関 連 研 究

ソフトウェアによるハードウェア仮想化技術を利用し，シングルプロセッサ上で複数の OS を動作させる手法[6),7)] は，アプリケーションごとに専用の OS が動作するため，アプリケーションの独立性が高く，統合に際して設計や実装を変更する必要がないという特徴をもつ．しかしながら，統合するアプリケーションの数だけ OS を動作させる必要があり，OS

Vol.2010-SE-168 No.2
Vol.2010-EMB-17 No.2
2010/6/1

切替え処理のオーバヘッドが大きいことから，リアルタイム性を要求されるアプリケーションの統合に適用するのは困難である．類似の技術として，マイクロカーネル上に複数の OS もしくは API レイヤをもたせる手法[8)–10)] もある．アプリケーションをユーザ空間で動作させることでメモリ領域を保護できる利点があるが，OS のメモリ使用量や処理オーバヘッドが大きいという課題がある．

RTOS の API を組み合わせて，異なる OS の API を実現する API ラッパ方式を採用しているソフトウェアとしては，eCos[11)]，xenomai[12)] がある．この方式は，RTOS が本来備える API の上に，API レイヤを重ねる方式であるため，ベースのソフトウェアの構成を変更することなく，複数の API を提供できるという特徴がある．しかしながら，ラッパとして実現された API の実行性能は，ベースとなるソフトウェア（例えば，RTOS）の API 性能に依存するため，本論文の提案手法のように，アプリケーションが直接 API を呼び出す手法に比べると，実行性能は落ちてしまうと考えられる．

7. まとめと今後の課題

本論文では，車載 ECU 統合において，統合前に個別の ECU で動作していたアプリケーションのみを統合するアプリケーション統合を想定し，統合におけるアプリケーション開発と検証の負荷をできる限り低減することを目的に，車載 ECU 統合向け RTOS フレームワークを提案した．提案フレームワークは，アプリケーション間の時間保護を実現する階層型スケジューラをベースに，μITRON 仕様と OSEK OS 仕様に準拠する API を両方備えたデュアル API をもつ．プロトタイプシステムである DUOS を実装し，API 実行時間とタスク切替え時間を評価した結果，実用上許容できるオーバヘッドであることを確認した．

今後は，μITRON 仕様，OSEK OS 仕様以外の API をサポートすることを想定して，API 実行性能と OS の拡張性を考慮した OS 構成を検討する必要がある．また，統合 ECU でアプリケーション間のメッセージ通信の手法や，メモリ，デバイス等の共有手法を検討していく．

参 考 文 献

1) 松原 豊, 本田 晋也, 冨山 宏之, 高田 広章: リアルタイムアプリケーション統合のための柔軟なスケジューリングフレームワーク, 情報処理学会論文誌, Vol.49，No.10，pp. 3508–3519 (2008)
2) 松原 豊, 本田 晋也, 冨山 宏之, 高田 広章: 時間保護のためのリアルタイムスケジューリングアルゴリズム, 情報処理学会論文誌 コンピューティングシステム, Vol.48，No.SIG 8(ACS 18)，pp.192-202 (2007)
3) TRON ASSOCIATION: μITRON 4.0 Specification Ver.4.02.00
4) OSEK/VDX Group.: OSEK/VDX Operating System Specification 2.2.1 (2003)
5) TOPPERS プロジェクト: TOPPERS 新世代カーネル統合仕様書 Release 1.1.0 (2009)
6) S.Devine, E.Bugnion, and M.Rosenblum: Virtualization system including a virtual machine monitor for a computer with a segmented architecture, US Patent (1998)
7) P.Barham, B.Dragovic, K.Fraser, S.Hand, T.Harris, A.Ho, R.Neugebar, I.Pratt and A.Warfield: Xen and the Art of Virtualization, In Proc. of the ACM Symposium on Operating Systems Principles (2003)
8) K.Robert and W.Stephan: The Pike OS Concept - History and Design, SYSGO AG : White paper (2007)
9) H.Härtig, M.Hohmuth, J.Liedtke, S.Schönberg and J.Wolter:e The Performance of μ-Kernel-Based Systems, 16th ACM Symposium on Operating Systems Principles (1997)
10) S.Oikawa, H.Ishikawa, M.Iwasaki and T.Nakajima: Providing Protected Execution Environments for Embedded Operating Systems Using a u-Kernel, In Proc. of International Conference on Embedded and Ubiquitous Computing, 153-163 (2004)
11) http://ecos.sourceware.org
12) http://www.xenomai.org