Vol.2019-SLDM-189 No.16
Vol.2019-EMB52 No.16
2019/11/14

車載制御システム向け次世代プロセッサの仮想化支援機能を用いたハイパーバイザー

本田 晋也[1]　山本 椋太[1]

概要：近年，車載制御システムに搭載する ECU の数を減らす ECU 統合への要求が高まっている．ECU 統合では，単一の ECU 上に安全度水準が異なる複数のアプリケーションを実行するため，メモリ保護や時間保護といったパーティショニング機構を用いる必要がある．パーティショニング機構を持つ RTOS を用いた場合，API の実行オーバヘッドが大きいこと，RTOS 自身が複雑になり信頼性を確保するのが困難であること，アプリケーション間での優先度調整が必要であるといった問題がある．これらの問題を解決するため，本論文では次世代の車載制御向けプロセッサに導入された仮想化支援機能を用いたハイパーバイザーを提案する．本ハイパーバイザーは，TDMA スケジューリングにより時間保護を実現し，仮想化用の MPU 機構によりメモリ保護を実現する．仮想化支援機能を用いることにより，統合前と同等の実行オーバヘッドでアプリケーションを実行可能であることを確認した．

1. はじめに

近年，車載システムの高機能化に伴い車両に搭載する ECU 数の増加が問題となっている．これは，サプライヤが開発する機能（部品）毎に ECU を用意するためである．車両に搭載する ECU の数を減らすため，機能毎に独立した ECU で実行していた複数の車載制御アプリケーション（車載制御アプリ）を単一の ECU で実行する ECU 統合を実現したいという要望がある．

現状の ECU は単一のサプライヤにより開発されているため，ECU 内の全てのプログラムの安全度水準を同じレベルで開発することが可能であり，プログラム間をパーティショニングする必要がなかった．一方，ECU 統合を行うと，ECU 内に安全度水準の異なる車載制御アプリが共存することになるため，パーティショニングを行わない場合，全ての車載制御アプリを最も高い安全度水準の車載制御アプリに合わせて開発する必要があり開発コストの増大を招く．よって ECU 統合においては，高い安全水準の車載制御アプリを保護すると共に，効率的なソフトウェア開発を行うため，何らかのパーティショニング機構が必要になる．

車載システム向けの RTOS の標準仕様である AUTOSAR 仕様では，保護を持つ RTOS 仕様を定めている．保護 RTOS では保護対象のソフトウェアをプロセッサのユーザーモードで動作させ，OS を特権モードで動作させるため，OS の実行オーバヘッド（アプリケーションから OS の API 呼び出し時間）が大きいという問題がある．また，AUTOSAR 仕様の保護 RTOS（保護付き A-OS）ではタスクや割込みハンドラという細かい単位で保護する．そのため，安全水準の異なる車載制御アプリを分離することはできない．例えば，ある車載制御アプリのタスクの優先度を変更すると，システム上の全ての車載制御アプリがその影響を受けるという問題がある．

これらの問題を解決する手法として，ハイパーバイザー（HV）や仮想マシンモニタ（VMM）により仮想マシン（VM）を実現し，OS も含めた大きな単位でパーティニングを行う手法が挙げられる．車載システムにおいても，情報系の汎用 OS と AUTOSAR プラットホームを HV を用いて同じシステムで動作させる構成が提案されている [1], [2], [3]．また，車載システム向けの既存プロセッサを用いて準仮想化による HV が提案されている [4], [5]．

VM を効率的に実行する手法として，プロセッサのハードウェア仮想化支援機能を用いる手法が挙げられる．車載制御システム向けのプロセッサにおいても前述の要求から，ハードウェア仮想化支援機能を持つアーキテクチャが発表されている．

これまで我々は，車載制御システム向けプロセッサの試作チップに搭載されたハードウェア仮想化支援機能を用いた VMM を提案している [6]．提案した VMM は車載システムの要件である，リソースの使用量と実行オーバヘッドの低減及び高いリアルタイム性を実現するために，VMM とホストの RTOS を一体化した構成としている．提案した VMM はハードウェアの制約により，低い実行オーバヘッドで実現可能な VM は 1 個であった．この問題に対して，低い実行オーバヘッドで複数の VM を実現可能なよう拡張がなされたハードウェア仮想化支援機能を持つ車載制御システム向けプロセッサが開発された [7], [8]．

本研究では，この新たなハードウェア仮想化支援機能を用いた車載制御システム向けの HV を提案する．本 HV は，複数の VM をサポートし，ECU 統合を容易化するた

[1] 名古屋大学 大学院情報学研究科

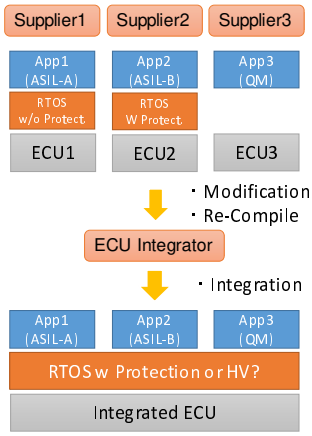

図 1　ECU 統合の設計フロー

め，一定周期内をタイムウィンドウと呼ぶ時間幅に分割して各 VM に割り付け，繰り返し順に実行していく TDMA（Time Division Multiple Access）スケジューリングを用いることにより時間保護を実現する [9]．前述のように，HV ではソフトウェアを VM 単位で保護するため，VM 内の車載制御アプリでは保護機能を持たない RTOS（保護無し A-OS）を使用出来る．さらに，処理の切り換えは，VM という大きな単位で保護して切り換える．以上より，保護付き A-OS を用いた統合と比較して，処理の切り換え時間や OS の実行オーバヘッドが軽減されると予想される．

2. ECU 統合

本節では，想定される設計フローや前提について説明し，プラットホームに求められる要件について述べる．

2.1 ECU 統合の設計フローの想定

前述の ECU 等に対する要求や現状の ECU 開発を考慮して，ECU 統合を実施する場合に想定される設計フローを図 **1** に示す．

統合前の車載制御アプリは別々のサプライヤが開発しており，アプリ毎に機能安全水準が異なる．ソフトウェアプラットホームは，A-OS を用いているものや，OS レスのものがある．また，OS ベンダもサプライヤ毎に異なる．

前述の複数の性質を持つ車載アプリを最終的に，ECU インテグレータがが 1 つの ECU に統合する．各サプライヤは，ECU インテグレータに対して，統合する車載アプリを提供するが，この際に車載アプリのソースコードやタスク構成といった設計情報は開示したくないという要望がある．一方，ECU 統合のために車載アプリのソースコードの軽微な変更と再コンパイルは許容する．

2.2 プラットホームに求められる要件

ECU 統合を実現するためには，プラットホームには次のような要件が求められると考えられる．

- **保護要件 1**：各車載制御アプリが使用するメモリ領域（ROM/RAM/周辺回路）がお互い保護されること．
- **保護要件 2**：ある車載制御アプリが，他の車載制御アプリの時間的振る舞いに影響を与えないこと．

これらの保護は統合対象の各車載制御アプリがそれぞれ異なる安全度水準を持つ場合に必須であるのは明確である．安全度水準が同じ場合であっても，開発するサプライヤが異なると保護がなければ，ある車載制御アプリを変更した場合に他の車載制御アプリの再検証が必要となる．そのため，サプライヤ間の並行開発が出来ず開発効率が悪くなる．この問題を解決するためには，保護機能が必要となる．

- **非機能要件 1**：車載制御アプリの実行オーバヘッドが大きく増加しないこと．
- **非機能要件 2**：車載制御アプリのソースコードの変更が少ないこと．
- **非機能要件 3**：サプライヤは ECU インテグレータに対して設計情報を開示せずに統合が可能なこと．

非機能要件 1 は，統合 ECU は統合前の ECU と比較して高速なものを用いるが，複数の車載アプリで CPU 時間を共有するため，実行オーバヘッドが増加しないことが求められる．

非機能要件 2 に関しては，ECU 統合向けに変更量が多いと，再検証のコストが大きくなるため，変更量が少ないことが求められる．

3. ECU 統合の手法の比較

ECU 統合の手法について, 保護機能付き RTOS を使う方法とその問題点ついて述べ，それらの問題が仮想化により解決可能な理由と求められる要件について説明する．

3.1 保護機能付き RTOS による課題

保護付き A-OS を用いて車載統合を実現するには，各車載制御アプリを個別の OSAP として実行する．この構成では，保護要件 1 は満たすが，他の要件に関して以下の問題が発生する．

まず，保護要件 2 に関して，保護付き A-OS では，時間保護は備えるが保護単位がタスクであり，全てのタスクは OSAP を区別せず優先度ベースでスケジューリングされる．そのため，ある OSAP のタスク設計（実行時間）や優先度を変化させた場合，他の OSAP のタスクのスケジューリングに影響を及して実行タイミングが変わるため，保護要件 2 を満たすことが出来ない．

次に，非機能要件 1 に関しては，前述のように保護付き A-OS では，OSAP から OS を含む BSW の機構を呼び出す場合にはソフトウェア割込みより呼び出す必要がある．保護機能が無い AUTOSAR OS（保護無し A-OS）を用いた場合では，関数呼び出しで実現できるため，比較すると 3 倍近くの実行時間が必要となる．

非機能要件 2 に関しては，統合前に AUTOSAR プラットホームを導入している場合は，統合前の前後で異なる OS ベンダの AUTOSAR プラットホームとなる可能性がある．この際には，ソフトウェアの変更が必要となる．それに加えて，ライセンス購入等のために追加の費用が必要となる．一方，AUTOSAR プラットホームを導入していない場合は，その対応のための変更が必要となる．

非機能要件 3 に関しては，A-OS では，OSAP 内のタス

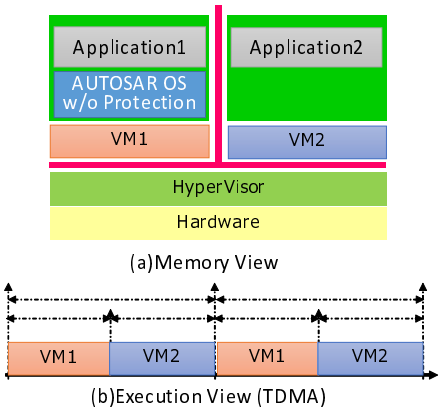

図 2　仮想化によるパーティショニング

クやその他のカーネルオブジェクトの構成は XML 形式のコンフィギュレーションファイルとして記述して提供する必要がある．そのため，タスク構成といった設計情報を開示することになり満たせない．

3.2　仮想化による実現

仮想化では，**図 2**（a）に示す様に，各車載制御アプリを VM 内で動作させる．各 VM には特定のメモリ空間を割り付け，それらはお互い保護される．時間保護に関しては，図 2（b）に示す様に VM 単位で TDMA スケジューリングで実行することで，保護付き A-OS で問題となる保護の単位が小さいという問題が解決される．具体的には，各車載制御アプリが OS を含む大きな単位でスケジューリングされるため，ある OSAP のタスク設計や優先度を変更したとしても，他の OSAP の実行に影響を及ぼさない．また，各車載アプリは独立したバイナリであり，OS 使用の有無や使用する OS を個別に選択可能であるため，ソースコードの変更や追加のライセンスの問題も発生しない．実行オーバヘッドに関しては，後述するハードウェア仮想化支援機能を用いることにより解決する．

以上の通り，仮想化を用いることにより，コンセプトレベルでは ECU 統合に求められる要件を満たせる可能性が高い．非機能要件に関しては，仮想化を前提とすると次の様に詳細化される．

- **仮想化要件 1**：仮想化による車載アプリの実行オーバヘッドの増加が小さいこと．
- **仮想化要件 2**：TDMA スケジューリングの実行オーバヘッドが小さいこと．
- **仮想化要件 3**：車載制御アプリのソースコードの変更量が少ないこと．
- **仮想化要件 4**：HV が使用するメモリサイズが小さいこと．

仮想化要件 1 は，非機能要件 1 から来る要求であり，統合対象の車載制御アプリ自身の実行時間が VM 内で実行することにより増加しないことが求められる．仮想化要件 2 も同様に非機能要件 1 から来る要求であり，TDMA スケジューリングの導入により，VM の切り換えに大きな実行オーバヘッドが発生しないことを確認する．仮想化要件 3 は，非機能要件 2 と同等であり，既存のプログラムの変更，特に OS のディスパッチャのようなコア部分の変更があると仮想化向けにプログラムの再検証や認証が必要となるため，変更量が少ないことが求められる．仮想化要件 4 は，HV 自体のコードサイズが大きいと，高い安全度水準が要求される場合に，開発コストが大きくなるという問題があるためである．

4.　仮想化支援ハードウェア

本論文で提案する HV は，ルネサスエレクトロニクス社の車載制御システム向けのプロセッサである RH850 に搭載されたハードウェア仮想化支援機能（HAV:Hardware-asisted Virtualization）[7], [8] を前提に設計及び実装を行った．

HV 特権モードと VM 特権モード

HAV では，既存の動作モードである特権モードとユーザーモードと直行した，CPU 本来の機能をすべて利用できる HV 特権モードと，一部の機能だけを利用できる VM 特権モードを追加している．

動作モードが VM 特権モードの間は，特定の機能の設定レジスタなどのプロセッサリソースをプロセッサ本来のリソースではなく，VM 用に用意されたものを参照して動作する．これらのリソースを VM コンテキストと呼ぶ．複数の VM を実行する際には，VM を切り換えるタイミングでその内容を HV で入れ替える必要がある．VM の切り替えを高速化するため，1 命令で VM コンテキストのメモリへの退避や復帰が可能な VM レジスタ退避・復帰命令を持つ．

HV から VM への遷移は，ステータスレジスタに VM に遷移することを示すビットをセットして例外リターン命令を実行することで行う．VM から HV への遷移は，コールゲートとして、後述する割込みや，例外発生時，HV 呼び出し命令実行時にのみ遷移する．

VM 識別子と割込み

GPID と呼ぶ ID を保持するレジスタを持つ．このレジスタは，HV 特権モードでのみ変更可能である．VM のインスタンス毎に GPID を割り付け，VM 切り換え時にこれから実行する VM に対応する GPID を HV によりセットすることを想定している．

割込みは要因毎に HV に割り付けるか，GPID を指定して特定の VM に割り付けることが可能である．前者を HV 割込み，後者を VM 割込みと呼ぶ．割込みが発生した場合の振る舞いを**表 1** に示す．VM 実行中は HV 割込みは禁止出来ないため，HV 割込みを発生させるタイマにより TDMA スケジューリングを実現することにより，VM 内の状態がどの様な状態であっても，HV に制御を移すことが可能であり，時間保護が実現可能となる．

メモリ保護

MPU 領域が追加されており，各 MPU 領域に対して，VM で使用する MPU 領域（VM-MPU 領域）と HV で使用可能な MPU 領域（HV-MPU 領域）を指定することが可能である．HV-MPU 領域は VM 内からは設定出来ず，VM-MPU 領域で許可されたメモリ空間であっても，HV-MPU 領域で禁止されていれば，VM 内からのアクセスはエラーとな

表 1 HAV の割込み発生時の振る舞い

実行中の特権モード	発生した割込みの 割り付け先	振る舞い
HV 特権モード	VM 割込み	受け付けない
HV 特権モード	HV 割込み	HV 特権モードで割込みが許可されていれば受け付ける.
VM 特権モード	HV 割込み	常に受け付けて HV 特権モードへ遷移
VM 特権モード	VM 割込み（同一 GPID）	VM 内で割込みが許可されていれば受け付ける.
VM 特権モード	VM 割込み（異 GPID）	受け付けない

る．VM の切り換えを高速化するために，MPU の複数の領域の設定を 1 命令で退避・復帰する MPU 退避・復帰命令が用意されている．

一方，周辺回路は，メモリと異なり，不連続な領域を各 VM に割り付けるため，周辺回路保護機能によりメモリ保護を実現する．

5. ハイパーバイザー（HV）の仕様と実装

ECU 統合向けの HV の仕様と実装について説明する．

5.1 仕様

時間保護

時間保護の概要を**図 3** に示す．時間保護を実現するため，スケジューリングは TDMA を用いる．TDMA ではシステム周期を指定し，システム周期内の時間を複数のタイムウィンドウに分割する．各タイムウィンドウは任意の時間長（タイムウィンドウ長）と実行する処理を指定することが可能である．タイムウィンドウで実行可能な処理は大きく次の 2 種類がある．1 つ目は，VM タイムウィンドウであり，ソフトウェアは VM 特権で実行される．統合前の各 ECU 上で実行していたソフトウェアを実行することを想定している． 1 つの VM に 1 つの車載制御アプリを割り付ける．もう 1 つは，HV タイムウィンドウであり，ソフトウェアは HV 特権で実行される．HV とバイナリを共有するハンドラが実行される．例えばネットワーク処理や ECU 全体の監視処理等を実行することを想定している．

システム周期の最後にはタイムウィンドウを割り付けない区間を設ける（アイドル区間）．この区間では HV アイドル処理が実行される．HV アイドル処理では，HV とバイナリを共有するハンドラが実行される．HV アイドル処理は後述する HV 割込みによりシステム周期内の時間が消費された場合でも，システム周期内で各タイムウィンドウの実行時間を保証するための機構である．そのため，HV 割込みの発生の有無によって実行時間が変動する．

メモリ保護

ROM・RAM の領域及び周辺回路を各 VM に ECU 統合時に割り付ける．各 VM 実行時はその VM に割り付けた ROM・RAM の領域及び周辺回路のみアクセス可能とし，それ以外をアクセスした場合はエラーとする．

割込み

各割込みは，特定の車載制御アプリないし HV に静的に割り付ける．車載制御アプリに割り付けた割込みを VM 割込み，HV に割り付けた割込みを HV 割込みと呼ぶ．

VM 割込みは割り付けた VM が実行されている間のみ受け付けられる．HV 割込みはどの状態でも受け付けられ，割り込まれた時間分，タイムウィンドウの切り換え時間は遅延する（時間時間を保証）．その分 HV アイドル処理の実行時間が短くなる．アイドル区間より HV 割込みの処理時間が長くなった場合はエラーなる．アイドル区間を超過しないよう HV 割込みを調整するのは，ECU インテグレータの責任とする．

5.2 実装

時間保護

タイマを 2 個用いる．それぞれタイムウィンドウタイマとシステム周期タイマと呼ぶ．

システム周期タイマは，周期タイマとしてシステム周期を周期として設定して HV の初期化終了時にスタートさせる．満了時の割込みは HV 割込みとして，HV のシステム周期割込みハンドラを呼び出す様に設定する．

システム周期割込みハンドラでは，割り込んだ処理が HV アイドル処理か判断して HV アイドル処理でなければエラー処理を実行する．HV アイドル処理であれば，次のシステム周期を開始するため，システム周期の最初のタイムウィンドウの情報を取得して，後述するタイムウィンドウハンドラと同様にその処理への切り換えを実施する．

タイムウィンドウタイマは，単発タイマとして用い，次に実行するタイムウィンドウのウィンドウ時間を設定してスタートさせる．割込みは HV 割込みとし，HV のタイムウィンドウハンドラを呼び出す様に設定する．

タイムウィンドウハンドラでは，割込み発生時のコンテキストを判定する．VM が実行されていれば，VM のコンテキストを VM レジスタ退避命令と汎用レジスタ退避命令により退避する．HV 特権の処理であれば，HV タイムウィンドウか HV アイドル処理か判断して，汎用レジスタをそれぞれのコンテキスト保存領域に退避する．その後，次に実行するタイムウィンドウの情報を取得し，VM や HV タイムウィンドウへの遷移ならタイムウィンドウタイマの設定を行う．最後にコンテキスト復帰を行う．遷移先が VM であれば VM レジスタ復帰命令と汎用レジスタ復帰命令及び MPU 復帰命令によりコンテキストを復帰して VM の処理を開始する．HV 特権の処理であれば，対象の処理の汎用レジスタを汎用レジスタ復帰命令により復帰して対象の処理へ遷移する．

メモリ保護

ROM/RAM の保護は HV-MPU 領域を設定して各 VM 毎にアクセス可能な領域を設定する．MPU の領域設定は，VM 切り換え時に MPU 設定命令により設定する．

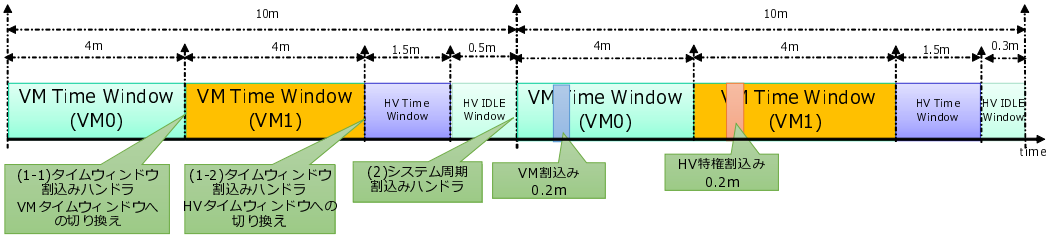

図 3　HV の TDMA スケジューリング

周辺回路に関しては，周辺回路保護機能を用いて，各 VM が使用する周辺回路のみをアクセス可能となるよう設定する．この設定は HV の初期化時に実施すればよい．

割込み

VM 割込みは，割込み割り付け機能により，割込み毎にその割込みを割り付ける VM の GPID にバインドする．

HV 割込みが発生すると HV の割込みハンドラが呼び出される．ハンドラでは，タイムウィンドウタイマを一時停止させたのち，割込み発生時のコンテキストを判定する．VM 特権すなわち VM が実行されていれば，VM のコンテキストを VM レジスタ退避命令と汎用レジスタ退避命令により退避する．HV 特権であれば，汎用レジスタの一部を汎用レジスタ退避命令により退避する．次に割込み要因を判定してハンドラを実行する．ハンドラの実行終了後は，タイムウィンドウタイマを再開し，戻り先のコンテキストにより復帰処理を選択して復帰する．

VM 割込み発生時は，割り付けた VM が実行されていなければ，割込みの受け付けは保留される．割り付けた VM が実行されていれば，VM 内の割込みハンドラが VM 特権で呼び出される．

6. 評価

実現した HV に対して，2.2 節で述べた保護用件と 3.2 節で述べた仮想化要件を満たしているか評価を行う．

仮想化要件に関しては，まず，仮想化要件 1 及び，3 節で述べた保護付き A-OS の保護無し A-OS に対する実行オーバヘッドを示すため，各 RTOS の実行オーバヘッドについて計測する．続いて，仮想化要件 2 への適合を確認するため，各種実行オーバヘッドを評価する．次に，仮想化要件 3 への適合を確認するため VM 内で動作させる車載制御アプリの変更量として，AUTOSAR OS の変更量を評価する．そして，仮想化要件 4 への適合を確認するため HV の規模について評価する．

評価には，前述の仮想化支援技術を搭載した研究用試作プロセッサを用いた．動作周波数は，400MHz である．評価に用いた RTOS は AUTOSAR 仕様の TOPPRES/ATK2 カーネルを用いた．

6.1 保護要件の充足の評価

保護要件 1 に関しては，VM 毎にメモリ領域と周辺回路を割り当て，VM 切り換え時に，HV により HAV の提供する HV-MPU 領域の機能と周辺回路保護機能を設定することにより，各 VM の ROM/RAM/周辺回路が保護される．

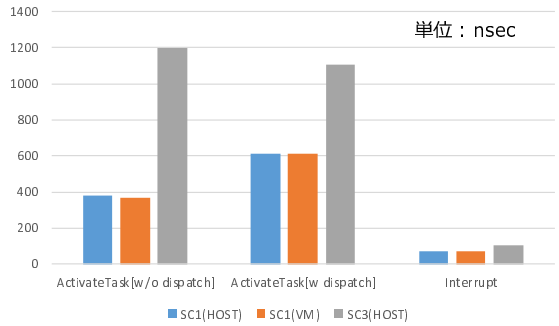

図 4　各 OS の実行時間

そのため保護要件 1 を満たしていると言える．

保護要件 2 に関しては，VM の切り替えは，HV が管理するタイマにより発生し，このタイマは VM から保護され，割込みに関しても HV 特権割込みとすることにより，VM の振る舞いや状態に関係なく，VM に割り当てた実行時間経過後は割込みが入り VM の切り換えを実施可能となる．以上より保護要件 2 を満たしていると言える．

6.2 車載アプリの実行オーバヘッド評価（仮想化要件 1）

各種条件で計測した OS の実行時間を図 **4** に示す．SC1（HOST）と SC3（HOST）は，直接プロセッサ上で保護無し/保護付き A-OS を実行した結果であり，SC1（VM）は VM 上で保護無し A-OS を実行した結果ある．

測定項目は，タスク起動の API を呼び出してリターンするまでの時間と，ディスパッチしてタスクが起動するまでの時間，及び割込み応答時間を計測した．

SC1（HOST）と SC1（VM）を比較すると実行時間はほぼ同等であり，仮想化要件 1 を満たしていると言える．

SC1（HOST）と SC3（HOST）を比較すると SC3（HOST）は最大 3 倍程度低速となっている，3 節で述べたように保護付き A-OS を用いると，実行時間が増加することが分かる．

6.3 HV の実行オーバヘッド評価（仮想化要件 2）

図 3 に示すタイムウィンドウとシステム周期の切り換え時間を計測した測定結果を**表 2** に示す．なお，測定において HV-MPU の数は 4 枚である．

タイムウィンドウの切り換え時間及びシステム周期割込みハンドラに関しては，SC1（HOST）と SC3（HOST）のディスパッチを伴うタスク起動と比較する．SC1（HOST）のタスク切り替え時間は，図 4 から 600ns 程度であり，VM 切り換えと同程度の実行オーバヘッドとなっている．一方，

表 2 HV の各実行オーバヘッド（nsec）

項目	実行時間
(1-1) タイムウィンドウ割込みハンドラ : VM タイムウィンドウへの切り替え	830
(1-2) タイムウィンドウ割込みハンドラ HV タイムウィンドウへの切り替え	352
(2) システム周期割込みハンドラ	625

表 3 HV と各 RTOS のコードサイズ（byte）

対象ソフト	ROM	RAM	RAM(w/o stack)
HV	8,310	2,616	1,592
保護無し A-OS	48,116	9,342	2,830
保護付き A-OS	90,765	20,614	5,510

SC3（HOST）は，1,100ns 程度であり，VM 切り換え時間より大きい．これは，HV が機械的に VM コンテキストを切り換えしているのに対して，SC3（HOST）は，エラーチェックやタスク状態変更と行った細かい処理が必要となるため，実行オーバヘッドが大きくなったことで生じたと言える．スケジューリング方法にも依存するが，VM はタスクより処理の粒度が大きいため，切り換え頻度はタスク切り替えの方が多い．この様に，統合前に用いる A-OS と比較してほぼ同等の時間で VM 切り換えを実現出来るため，仮想化要件 2 を満たしていると言える．

6.4 RTOS の変更量評価（仮想化要件 3）

OS の VM 対応のため，以下の変更が必要となった．

- システム全体に影響するハードウェアの操作の削除．クロックや IO ポートの初期化処理が該当し，これらの処理は無効とし，HV で実施するよう変更した．
- 使用するハードウェアリソースの変更 ROM/RAM やタイマや UART のチャネルを実行する VM に割り付けられたものに変更する．ROM/RAM の変更はリンカスクリプトを変更することで対応でき，タイマや UART は OS のデバイスドライバのコンフィギュレーションを変更することで対応可能である．

以上のように，OS の基本的な箇所の変更は必要なくコンフィギュレーションレベルの変更で VM 上で実行でき，仮想化要件 3 を満たしていると言える．

6.5 HV の規模評価（仮想化要件 4）

統合前に使用する保護付き A-OS 及び，統合に用いる可能性がある保護付き A-OS と比較する．それぞれの ROM サイズと RAM サイズを**表 3** に示す．RAM サイズはタスクや HV タイムウィンドウで使用するスタックサイズを含まない場合も示す．RAM サイズは VM やタスクの数で変わる．HV は 2 個の VM，保護無し A-OS では 11 個保護付き A-OS では 21 個のタスクの構成で計測した．

RAM サイズは各 OS と比較して 1 桁小規模であり，検証が容易であると言える．なお，VM のコンテキストの保存領域は，VM 毎に 280byte 必要である．

このように，RTOS と比較して規模が小さいため，仮想化要件 4 を満たしていると言える．

7. おわりに

本論文では次世代の車載制御向けプロセッサに導入された HAV を用いた ECU 統合向けの HV を提案した．ECU 統合で求められる要件について説明し，仮想化及び HAV を用いることにより，定性的にそれらの要件を満たせることを示した．更に HAV を用いた HV を実現し評価することにより，定量的にも要件を満たしていることを示した．今後の課題としては，本 HAV を用いた ECU 統合の設計フローを確立し，その際に必要となるツールや HV の機能を検討し実現することが挙げられる．

謝辞 本研究を進めるにあたりご協力いただいたルネサスエレクトロニクス株式会社に深く御礼申し上げます．
本研究の一部は JSPS 科研費 JP17K00075 の助成を受けたものです．

参考文献

[1] A. Hergenhan, and G. Heiser, ”Operating systems technology for converged ECUs.” 6th Emb. Security in Cars Conf.(escar). Hamburg, Germany: ISITS. 2008.
[2] D. Reinhardt, D Kaule, and M. Kucera. ”Achieving a scalable e/e-architecture using autosar and virtualization.” SAE International Journal of Passenger Cars-Electronic and Electrical Systems 6.2013-01-1399 (2013): 489-497.
[3] G. Heiser, ”Virtualizing embedded systems: why bother?.” Proceedings of the 48th Design Automation Conference. ACM, 2011.
[4] D. Reinhardt, and G. Morgan, ”An embedded hypervisor for safety-relevant automotive E/E-systems,” Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014), Pisa, pp. 189-198 2014.
[5] D. Haworth, “An AUTOSAR-compatible microkernel for systems with safety-relevant components”, Informatik aktuell, Volume “Herausforderungen durch Echtzeitbetrieb”, Pages 11-20, 2012.
[6] 本田晋也, 鈴木均, 樋口正雄, 福井昭也, ”車載システム向けハードウェア仮想化支援機能による RTOS 一体型仮想マシンモニタ”, 情報処理学会研究報告, Vol.2017-OS-139,No.9, pp. 1-7, 福岡, Mar 2017.
[7] S. Otani et al., ”2.7 A 28nm 600MHz Automotive Flash Microcontroller with Virtualization-Assisted Processor for Next-Generation Automotive Architecture Complying with ISO26262 ASIL-D,” 2019 IEEE International Solid- State Circuits Conference - (ISSCC), San Francisco, CA, USA, 2019, pp. 54-56.
[8] RH850/U2A16 (online), available from ⟨https://www.renesas.com/jp/ja/products/microcontrollers-microprocessors/rh850/rh850u2x/rh850u2a16.html⟩ (accessed 2019-09-26).
[9] S. Saidi, S. Steinhorst, A. Hamann, D. Ziegenbein and M. Wolf, ”Special Session: Future Automotive Systems Design: Research Challenges and Opportunities,” 2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Turin, 2018, pp. 1-7.