Vol.2017-OS-139 No.9
2017/3/1

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

本田 晋也[1,a)]　鈴木 均[2]　樋口 正雄[3]　福井 昭也[3]

概要：近年，車載システムの高機能化や ECU 統合により，単一 ECU 内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てシステムを構築する必要がある．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，ECU 統合では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，本論文では車載システム向けのハードウェア仮想化支援機能を用いた仮想マシンモニタ（VMM）を提案する．本 VMM は，ホスト型として，RTOS と一体に動作することにより，車載システムで要求される高いリアルタイム性を少ないハードウェアリソースやソフトウェア変更量で実現した．

RTOS Integrated Virtual Machine Monitor with Hardware Virtualization Support for Automotive System

Shinya Honda[1,a)]　Hitoshi Suzuki[2]　Masao Higuchi[3]　Akiya Fukui[3]

Abstract:
The integration of different safety integrity level software in the same ecu needs a partitioning mechanism such as memory and time protection. The AUTOSAR standard gives the specification of RTOS (AUTOSAR-OS) with memory and timing protection mechanisms to ensure these requirements. The AUTOSAR-OS has some problems such as a high OS API execution overhead, an increasing complexity of OS, and a fine grained protection unit. To solve these problems, this work gives a Virtual Machine Monitor (VMM) with the hardware-assisted virtualization for automotive system. The proposed VMM runs along with the host OS to achieve high interrupt response time required in automotive system with small hardware resources and software modification.

1. はじめに

近年，車載システムの高機能化に伴い，ECU 上で実行されるソフトウェア（車載アプリケーション）の複雑化・大規模化が進んでいる．車載システムは高信頼性が要求されるため，ECU を開発しているサプライヤがその ECU 上の全てのプログラムを開発するのが通常であった．しかしながら，車載アプリケーションの複雑化・大規模化に伴い，外部で開発されたライブラリやソフトウェアプラットホームを使用する必要が出てきている．また，これまではサプライヤが開発する部品毎に ECU を用意するのが通常であったが，車載システムの高機能化に伴い ECU 数の増加が問題となっている．そのため，これまで別の ECU で実行していた複数の車載アプリケーションを単一の ECU で実行したいという要望がある．

前述の様にある ECU 内のプログラムは単一のサプライヤにより開発されていたため，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った．そのため，プログラム間をパーティショニングする必

[1] 名古屋大学 大学院情報科学研究科
愛知県名古屋市千種区不老町
[2] ルネサス エレクトロニクス株式会社
東京都江東区豊洲三丁目 2 番 24 号
[3] ルネサス システムデザイン株式会社
東京都小平市上水本町五丁目 22 番 1 号
[a)] honda@ertl.jp

要がなく，メモリ保護や時間保護といった保護機構を持つRTOS（保護 RTOS）は必要無かった．一方，外部ソフトウェアの導入や ECU 統合を行うと，ECU 内に安全度水準の異なるソフトウェアが共存することになるため，保護 RTOS によるパーティショニングを行わない場合，全てのアプリケーションを最も高い安全度水準のアプリケーション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．また，パーティショニングを行わない ECU 統合は，サプライヤ間の調整が必要となるが，実現は事実上不可能と言われる．よって今後の ECU 開発においては，高い安全水準のアプリケーションを保護すると共に，効率的なソフトウェア開発を行うため，保護 RTOS 等による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，時間保護仕様である SC2，メモリ保護仕様である SC3，両方をサポートした SC4 が策定され公開されている [1]．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（OS の API 呼び出し）が大きいという問題がある．また，現状の AUTOSAR 仕様の保護 RTOS ではタスクや割込みハンドラという細かい単位で保護を行うため，安全水準の異なる車載アプリケーションを分離することはできない．例えば，ある車載アプリケーション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アプリケーション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，仮想マシンモニタ（VMM）と仮想マシン（VM）を用いて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 VMM 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 [2], [3], [4]．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 VMM が提案されている [5], [6]．

VMM と 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がある．汎用システムやスマートフォン向けには，Intel の Intel-Vt や ARM の Virtualization Extensions があり，それらを用いた VMM が実用化されている．車載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．

本研究では，車載システム向けプロセッサのハードウェア仮想化支援機能を用いた VMM を提案する．本 VMM は車載システムの要件である，リソースの使用量，実行オーバヘッドの低減，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．安全度水準が高い車載アプリケーションは，ホストの RTOS で実行し，安全度水準が低い車載アプリケーションは VM 内で実行する．

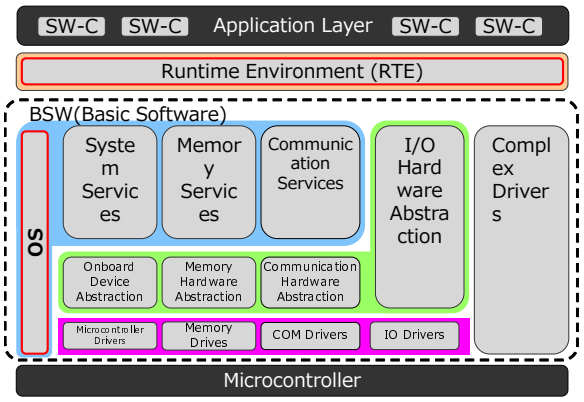

図 1 AUTOSAR プラットホーム
Fig. 1 AUTOSAR Platform

本論文の構成は次の通りである．まず，2 章で車載システムにおけるパーティショニングに対する要件について述べ，3 章で保護機能付き AUTOSAR OS の機能とパーティショニングに関する問題点について述べる．4 章で仮想化による実現方法の概要について述べ，5 章で車載システム向けのハードウェア仮想化支援機能について説明する．次に，6 章で車載システム向け VMM の設計と実装について説明した後，7 章で評価について述べる．

2. 車載システムにおけるパーティショニングに対する要件

本章では車載システムにおけるパーティショニングに対する要件について述べる．

車載システムおいてパーティショニングが必要なユースケースとして，AUTOSAR プラットホーム及び関連する外部ソフトウェアの導入，ASIL デコンポジション，ECU 統合等が挙げられる．これらの要求のうち本研究では，AUTOSAR プラットホーム及び関連する外部ソフトウェアの導入を対象とする．AUTOSAR プラットホームは，車載システムのディファクトスタンダートとなっているソフトウェアプラットホームであり，欧州の OEM に ECU を納入する際には，AUTOSAR プラットホーム及び対応した通信スタック等を用いることが納入要件であることが多い．AUTOSAR プラットホームの構成を**図 1** に示す．アプリケーションは SW-C と呼ばれるコンポーネントとして記述され，RTE を介して BSW と呼ばれる各種サービスを呼び出す．これまでの ECU 開発においては AUTOSAR プラットホームのうち，OS のみを用いて[*1]システムを構築しているケースが一般的であった．

この様に AUTOSAR OS のみを用いていたシステム（既存システム）に対して，ネットワーク接続や外部モジュールを実行するために AUTOSAR プラットホームを用いる

[*1] 正確には OSEK/VDX 仕様の OS

システム（フル AUTOSAR）を追加する場合，システムとしては既存システムの安全度を維持したいという要求がある．一方，フル AUTOSAR は BSW 等の規模が大きいため，システム全体の安全度水準を高めることは困難である．

ここで，既存システムを信頼系，追加するフル AUTOSAR 上のシステムを通常系と定義して，保護要件，機能要件，非機能要件，前提をまとめると次のようになる．

保護要件

- **保護要件 1**：信頼系のメモリ領域が保護されること．
- **保護要件 2**：信頼系の割込み応答性が悪化しないこと．
- **保護要件 3**：通常系の動作により，信頼系の振る舞いが変化しないこと．

機能要件

- **機能要件 1**：信頼系は AUTOSAR OS 上で動作すること．
- **機能要件 2**：通常系の実行を信頼系で容易に制御できること．

非機能要件

- **非機能要件 1**：通常系及び信頼系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**：通常系及び信頼系のソースコードの変更が少ないこと．
- **非機能要件 3**：ハードウェアコストは可能な限り低くすること．

前提

- **前提 1**：フル AUTOSAR 全体の安全度水準を高めることは困難である．

3. 保護機能付き RTOS による実現

3.1 保護機能付き AUTOSAR OS

AUTOSAR プラットホームにおいてパーティショニングを実現するためには，保護機能付きの AUTOSAR OS（保護付き A-OS）を使用する必要がある．保護付き A-OS の構成を図 **2** に示す．保護機能としては，メモリ保護機能と時間保護機能を持つ．保護付き AUTOSAR-OS では，タスク等の OS オブジェクトを OS アプリケーション（OSAP）と呼ばれるグループに分ける．OSAP には保護が有効な非信頼 OSAP と，無効な信頼 OSAP がある．非信頼 OSAP はユーザーモードで動作させ，信頼 OSAP と BSW は特権モードで動作させる．

メモリ保護に関しては，非信頼 OSAP からアクセス可能なメモリや I/O を制限する．そのため，非信頼 OSAP から BSW の機能（例えば OS の API 呼び出し）を使用するためには，ソフトウェア割込みによる呼び出しが必要となる．

時間保護に関しては，タスクや割込みハンドラ（ISR）毎に実行時間や起動間隔を監視する機構を提供する．

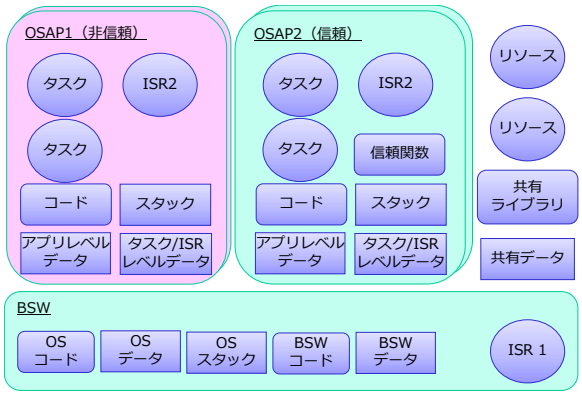

図 2 保護機能付き AUTOSAR OS
Fig. 2 AUTOSAR OS with Protection

3.2 パーティショニングに関する問題点

保護付き A-OS を用いて車載システムを構築する場合，通常系を非信頼 OSAP で，信頼系を信頼 OSAP とする．この構成では，2 章で述べた要件に関して以下の問題が発生する．

まず，保護要件 3 に関して，現在の保護付き A-OS では，時間保護の単位がタスクであるため，通常系のタスク設計や優先度を変化させた場合，信頼系のスケジューリングに影響を及ぼしてしまい，満たすことが出来ない．同様に機能要件 2 に関しても実現することが出来ない．

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 では非特権 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 2 倍近くの実行時間が必要となり，実行オーバヘッドが増加し非機能要件 1 を満たせない．

また前提より，保護付き A-OS においては OS を含む BSW 全体を特権モードで実行するため，信頼系が要求する安全度水準よりフル AUTOSAR で実現可能な安全度水準が低い場合は，保護要件 1,2,3 を満たすことが出来ない．

4. 仮想化による実現

前述のように保護付き A-OS では，各要件を満たすことが出来ない．この問題を解決する方法として，仮想化を用いる方法が考えられる．具体的には，**図 3** に示す様に通常系と信頼系を別の仮想マシンで動作させる．AUTOSAR OS に関しては，VM レベルで保護が適用されるため，保護無し A-OS を使用する．

この構成により，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，通常系と信頼系が OS を含む大きな単位でパーティショニングされるため，通常系のタスク設計や優先度を変更したとして

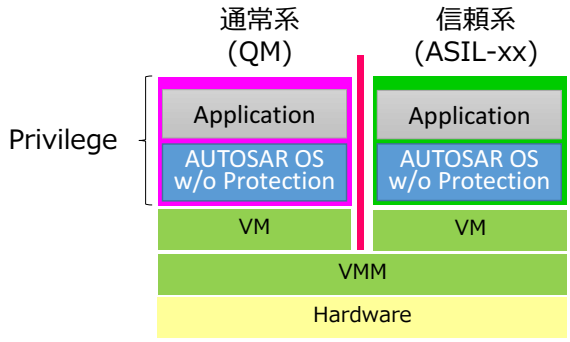

図 3 仮想化によるパーティショニング
Fig. 3 Partitioning with Virtualization

も，信頼系に影響を及ぼさない．実行オーバヘッドに関しては，OS に保護無し A-OS を使用するため BSW の呼び出し時間は増加しない．次に，通常系を実行する OS を含む BSW 全体が VM 内で実行されるため保護の対象となり，通常系の安全度水準よりフル AUTOSAR で実現可能な安全度水準が低い場合であっても，それぞれを独立した安全度水準で開発することが可能である．

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車載システムにおけるパーティショニングに対する要件を満たせる可能性が高い．車載システムに仮想化を適用するには，以下の点を満たす様に VMM を設計及び実装する必要がある．

- **仮想化要件 1**：仮想化による実行オーバヘッドが小さいこと．
- **仮想化要件 2**：仮想化のための機構のコードサイズが小さいこと．
- **仮想化要件 3**：仮想化のために既存のプログラムの変更量が小さいこと．

仮想化要件 1 は非機能要件 1 から来る要求であり，仮想化の導入により，信頼系や通常系や割込み応答時間の増加量や，信頼系と通常系間の切り替え時間が小さいことが必要である．仮想化要件 2 は，仮想化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．仮想化要件 3 は，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．

5. 車載システム向けのハードウェア仮想化支援機能

5.1 ハードウェア仮想化支援機能

4 章で述べた各仮想化要件を満たす VMM を実現するには，ハードウェア仮想化支援機能の利用が必須である．組込みシステムにおいても，スマートフォンやカーナビゲーションシステムのような非リアルタイムシステム向けに，ARM-A Profile の TrustZone や Virtualization Extensions といったハードウェア仮想化支援機能があり，それらを用いた VMM が存在する．一方，リアルタイム性が求められる車載システム向けのプロセッサにおいても，RH850 や ARM-R Profile においてハードウェア仮想化支援機能が用意されている．

5.2 RH850 評価コアにおけるハードウェア仮想化支援機能

本論文では RH850 ベースの評価用コアに搭載されているハードウェア仮想化支援機能（HAV:Hardware-asisted Virtualization）[8] を用いて VMM を実現する．本章では HAV の概要について説明する．

RH850 は車載システム向けのプロセッサであり，パワートレイン等の高い性能とリアルタイム性が要求される ECU で用いられている．

HAV は RH850 評価コアに搭載されたハードウェア仮想化支援機能である．HAV では既存のプロセッサが持つ動作モードに加えて新たな動作モードを追加している．既存の動作モードが，OS と OS 上で動作するアプリケーションとを区別するのに対し，新しい動作モードは，CPU 本来の機能をすべて利用できるネイティブマシン (NM) と，一部の機能だけを利用できる仮想マシン（VM）とを区別する．**図 4** に 2 つの動作モードの関係を示す．

VMM は仮想マシンを動作させる場合には，コアの動作モードを VM に変更する．動作モードが VM 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 NM のリソースではなく，VM 用に独立した異なるリソース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．VM コンテキストは、プロセッサの用途に応じて 1 組以上の数が搭載される．本研究に用いた評価コアは，1 組の VM コンテキストを搭載している．HAV では，NM と VM の切替え時に，VMM による実行コンテキスト入れ替えが小規模で済み、かつ VM が NM の実行資源を破壊しないように、VM コンテキストとして扱う機能レジスタを定義している．また，VMM は MPU 設定の一部を用いて，仮想マシンが利用可能なメモリ資源を制限することが可能である．

NM 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 NM 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，NM 呼び出し命令実行時にのみ遷移する．

割込みはチャネル毎に NM ないし任意の VM に割り付け，優先度制御は VM 毎に行う．割込みが発生した場合の振る舞いを**表 1** に示す．VM で実行中に NM 割り付けの割込みが発生すると即座に NM に遷移する．また VM 実行中は NM 割り付けの割込みは禁止出来ないため，NM 側で

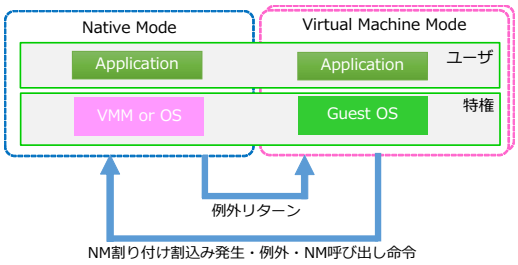

図 4 RH850 評価コアにおけるハードウェア仮想化支援機能
Fig. 4 Hardware-Assisted Virtualization of RH850 evaluation core

表 1 HAV の割込み発生時の振る舞い
Table 1 HAV Interrupt Behavior

実行中のコンテキスト	発生した割込みの割り付け先	振る舞い
NM	VM	受け付けない
NM	NM	NM で受け付ける
VM	VM	VM で受け付ける
VM	NM	NM に遷移して受け付ける

実行するプログラムの時間保護を実現することが出来る．

このようにして，HAV を用いることで車載システムに搭載する異なるアプリケーションを空間的にも時間的にも分離することが可能となる．

6. 車載システム向け VMM

6.1 設計

前述の HAV を用いた RH850 評価コア向け VMM を設計する．VMM の実現方法としてネイティブ型とホスト型がある．ネイティブ型はホスト OS が必要ないが，信頼系と通常系それぞれに VM が必要となる．一方，ホスト型では，ホスト OS 上で信頼系を実行可能であるため，1 個の VM コンテキストがあればよい．非機能要件 3 から，車載システム向けの VMM ではホスト型の方が必要なハードウエアリソースが少ないため適していると言える．また，機能要件 1 からホスト OS には AUTOSAR OS を用いる．信頼系は仮想化導入前の同じ構成で実行されるため，非機能要件 1,2 も満たすことが出来る．保護要件についても，通常系を VM 上で実行することにより満たすことが可能である．今回用いる RH850 評価コアは，1 組の VM コンテキストを搭載しており，ホスト型の VMM を構成するのに適している．

ホスト型の車載システム向け VMM の構成を**図 5** に示す．通常系を実行する VM は，機能要件 2 を満たすため，ホスト OS 上に VM を実行するタスク（VM タスク）を導入する．VM タスクがスケジューリングされると VM が実行される機構とする．この機構により，AUTOSAR OS の機能により VM タスクのスケジューリングを制御することで，VM の実行を制御することが可能である．VM タスク

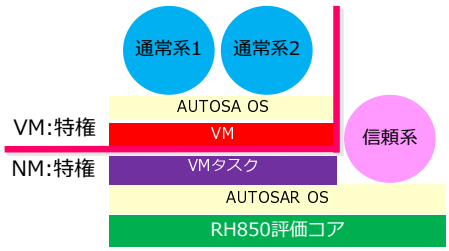

図 5 車載システム向け VMM の概要
Fig. 5 Overipew of Automotive System VMM

自身は信頼系のタスクとなるため，機能要件 2 を満たすことが出来る．VM タスクの詳細については次節で説明する．

6.2 実装

AUTOSAR OS として，TOPPERS プロジェクトから公開されているオープンソースの ATK2-SC1[7] を用いて，前述の設計に従って車載システム向け VMM の実装を行った．

実装の概要図を**図 6** に示す．NM 上でホスト OS として使用する OS を ATK2-SC1-VMM と呼ぶ．VM 内で実行するゲスト OS は通常の ATK2-SC1 からソースコードを変更しない．ホスト OS とゲスト OS は使用するメモリや周辺回路等のハードウェアリソースが衝突しないようにコンフィギュレーションを調整する．

VM のコンテキストは専用のコンテキスト保存領域を用意して保存する．ATK2-SC1-VMM の ATK2-SC1 に対する変更点は，割込みの入口処理となる図中の点線で囲まれた箇所である．VM から NM への切り替えは VM 動作中に NM に割り付けた割込み（NM 割込み）が発生した場合に行われる．まず，割込み発生時に NM を実行していたのか，VM を実行していたのかを判断する（図中の From VM）．VM を実行していた場合には，VM のコンテキストを保存して VM タスクの callee レジスタを復帰する（図中 a）．次に，通常の割込み処理を実施し割り込んだタスクの caller レジスタを保存し（図中 c）て割込みハンドラを実行する．その後，ディスパッチャが呼び出され，実行するべきタスクが存在すればディスパッチする．

VM タスクは他のタスクと同様にスケジューリングされる．VM タスクのコードを**図 7** に示す．vm_init() は VM の初期化処理で起動時に 1 回のみ呼び出される．その後無限ループ内でユーザーコードを実行して必要に応じて VM に遷移する関数（vm_start()）を呼び出す．vm_start() は，図 6 の g/h に相当する．VM への遷移後，前述のように NM 割込みが発生して VM から NM に処理が移った後，VM タスクに再びディスパッチすると vm_start() からリターンし，VM タスク中のユーザーコードが実行される．そのため，VM の実行をユーザーコードにより制御可能である．例えば，VM を周期的に実行したい場合は，周期イベントを作成してそのイベントに対してユーザーコードで

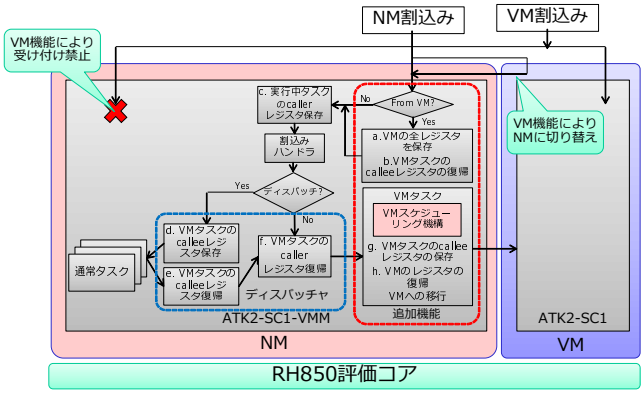

図 6 車載システム向け VMM の実装
Fig. 6 Implementaion of Automotive System VMM

```
1:  TASK(VMTask) {
2:      vm_init();
3:      while(1) {
4:          /* User Code */
5:
6:          vm_start(p_vminib->rbase); /* Call VM */
7:      }
8:  }
```

図 7 VM タスク
Fig. 7 VM Task

待つコード用意することによって，VM の周期実行が可能である．

7. 評価

前述の車載システム向け VMM に対して，前述の仮想化要件を満たしているか評価を行う．まず，仮想化要件 2,3 への適合を確認するためコードの追加量について評価する．次に仮想化要件 1 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．評価に用いた環境は，プロセッサ:RH850 評価コア 120Mhz，コンパイラ:GHS 2016.5.3 である．

7.1 コード量評価

既存のプログラムの変更箇所としては，6.2 章で述べた通りホスト OS の割込みハンドラの入口処理を変更する必要がある．この箇所はアセンブラで記述されており，内容は割込み発生のコンテキストの判定と VM 実行時に発生した場合の VM のコンテキスト，具体的には汎用レジスタの保存である．変更により追加されたコード量は 35 行と小さく，またコア部分を変更していないため，仮想化要件 3 を満たしていると言える．

OS 以外の仮想化のための機構としては，VM タスクが必要となる．VM タスク本体はユーザーがどの様に VM を実行するかによって記述量は変化する．VM を呼び出す

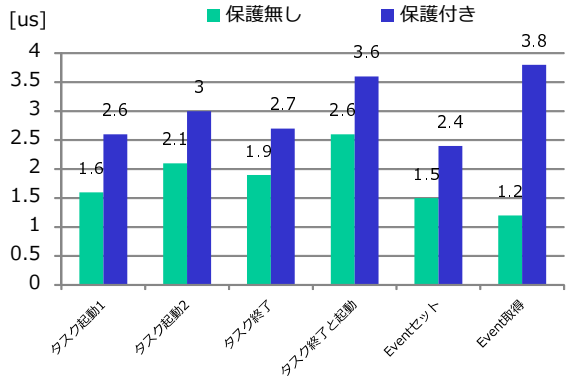

図 8 保護無し/付き AUTOSAR OS の実行オーバヘッド
Fig. 8 Execution Overhead of AUTOSAR OS with Protection

vm_start() に関してはアセンブラで記述されており，その行数は 29 行と少なく，仮想化要件 2 を満たしていると言える．

また，データ量の増大に関しては，VM 毎に 128 バイトの保存領域が必要となる．

7.2 実行オーバヘッド評価

まず，3 章で述べた保護付き A-OS について評価する．**図 8** に，保護付き A-OS（SC3）と保護無し A-OS（SC1）の代表的な API の実行時間の測定結果を示す．タスク起動 1 はディスパッチを伴わないタスク起動 API の実行時間，タスク起動 2 はディスパッチを伴うタスク起動 API の実行時間である．測定結果より，保護付き A-OS の実行時間は保護無し A-OS と比較して大きいことが分かる．特に Event 取得の増加率が最も大きく 3 倍近く増加している．この API は現在の Event の値を取得する API であり，引数に Event の値を格納する変数へのポインタを指定するため，API 内でポインタの正当性のチェックが必要なためである．

次に車載システム向け VMM の各種オーバヘッドを**表 2** に示す．(1)(2) の系の切り替え時間を図 8 の API の実行時間と比較すると十分小さいと言える．これは，HAV により VM コンテキストはハードウェア的に NM と別になっており，切り替えの際，保存・復帰の必要がないためである．次に (3)(4) の割込み応答時間については，VMM を用いない場合の ATK2-SC1 の割込み応答時間が 1.30μsec であるため，増加分はそれぞれ 0.66μsec と 0.13μsec であり，大きな値ではない．

以上の評価より，仮想化要件 1 を満たしていると言える．

8. おわりに

本研究では，車載システムにおいて，安全度水準が異なる複数のアプリケーションを同じ ECU で元の安全度水準

表 2 VMM の実行オーバヘッド
Table 2 VMM Execution Overhead

番号	パターン	実行時間
(1)	信頼系 → 通常系切り替え時間	1.18μsec
(2)	通常系 → 信頼系切り替え時間	0.53μsec
(3)	通常系の実行時の信頼系の割り込み応答時間	1.96μsec
(4)	信頼系の実行時の信頼系の割り込み応答時間	1.43μsec

で開発・実行するための車載システム向けのハードウェア仮想化支援機能を用いた VMM を提案した．提案した機構を元に設計及び実装を行い評価を行った．評価の結果，提案機能は要件を満たしていることを確認した．

今後の課題としては，複数の ECU を統合するために必要な，複数の VM を実現する機構や，時間保護の仕組みを入れることなどが挙げられる．

謝辞 本研究の一部は JSPS 科研費 JP26330062 の助成を受けたものです。

参考文献

[1] Specification of Operating System (online), available from ⟨http://www.autosar.org/fileadmin/files/releases/4-0/software-architecture/system-services/standard/AUTOSAR_SWS_OS.pdf⟩ (accessed 2017-01-19).
[2] A. Hergenhan, and G. Heiser, "Operating systems technology for converged ECUs." 6th Emb. Security in Cars Conf.(escar). Hamburg, Germany: ISITS. 2008.
[3] 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.
[4] G. Heiser, "Virtualizing embedded systems: why bother?." Proceedings of the 48th Design Automation Conference. ACM, 2011.
[5] 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.
[6] D. Haworth,"An AUTOSAR-compatible microkernel for systems with safety-relevant components", Informatik aktuell, Volume "Herausforderungen durch Echtzeitbetrieb", Pages 11-20, 2012.
[7] TOPPERS/ATK2 (online), available from ⟨http://www.toppers.jp/atk2.html⟩ (accessed 2017-01-19).
[8] リアルタイム制御システムに適した V850 CPU 向け仮想化技術 (online), available from ⟨https://www.renesas.com/ja-jp/about/press-center/news/2010/news20100929.html⟩ (accessed 2017-01-19).