Vol.2021-ARC-244 No.9
Vol.2021-SLDM-194 No.9
Vol.2021-EMB-56 No.9
2021/3/25

ハイパーバイザにおけるハードウェアの共有分析手法

鈴木 健太[1]　松原 豊[1]　守谷 友和[2,a)]　本谷 謙治[2]　岩切 英之[2]　高田 広章[1]

概要：コネクティッドカーや先進運転支援システム（ADAS）などにより多機能化する車載システムにおいて，複数の機能を同一の車載電子制御装置（ECU）上で実現することが求められている．機能間での悪影響を防ぐパーティショニング技術として，ハイパーバイザの使用が検討されているが，共有リソースの見落としや分離機能の不足により，機能間のパーティショニングが不十分な場合がある．これに対し，過去の研究では，主に脆弱性の報告をもとに事例ベースでの分離度分析を行ってきたが，そのような分析では未知の悪影響に対処できない．本論文では，システムの設計情報をもとに共有リソースを洗い出し，それぞれに対して分離機能が十分であるか分析を行うハードウェアの共有分析手法を提案する．また，提案手法を仮想の車載システムに適用し，分離が不十分なハードウェアを特定した．

An analysis method for shared hardware resources in hypervisors

Kenta Suzuki[1]　Yutaka Matsubara[1]　Tomokazu Moriya[2,a)]　Kenji Hontani[2]
Hideyuki Iwakiri[2]　Hiroaki Takada[1]

1. はじめに

近年，コネクティッドカーや，先進運転支援システム（ADAS）の普及により，車載組込みシステムの多機能化が進んでいる．その一方で，自動車の製造コスト削減のため，車載電子制御装置（ECU）の統合も進められている．これら 2 つの流れにより，求められる信頼度の異なる機能が混在するシステム（ミックスドクリティカルシステム）が増加している．信頼度の異なる機能間には，影響を防ぐために，パーティショニングが必要となる．そこで，パーティショニング技術として，ハイパーバイザの使用が検討されている．

仮想マシン（VM）間の共有リソースに対する DoS 攻撃により，他の VM の性能劣化が発生したケース [1] がある．車載システムにおいてパーティショニングが崩壊した場合，他の機能に影響を与える可能性がある．従って，パーティショニングが十分堅牢であるか事前に評価する必要がある．ハイパーバイザのセキュリティに関する研究は，大きく 2 つに分類される．1 つ目は，報告済みの脆弱性とその対策に注目しているものである．2 つ目は，特定の種類のデバイスに注目し，その仮想化手法のパフォーマンス分離やセキュリティ分離の改善を試みているものである．上記のアプローチはいずれもハイパーバイザのセキュリティの向上に有効である．しかし，過去に報告された脆弱性のみを分析していては，システムに起こりうる未知の攻撃や不具合に対し，事前に対策がとれない．また，特定のデバイスの分離度を改善したとしても，他に使用されているデバイスの分離度が劣悪であった場合，それを介した VM 間の攻撃や干渉が発生し，パーティショニングが崩壊する可能性がある．従って，システムを可動させる以前に，VM が共有するデバイスの分離度を網羅的に分析し，評価する必要がある．

そこで，本研究では，パーティショニングが崩れる要因を整理し，システムの設計情報から事前にパーティショニングの妥当性を分析することを目的とする．提案手法を仮想的な車載システムに適用し，結果を示す．

[1] 名古屋大学 大学院情報学研究科
Graduate School of Informatics, Nagoya University
[2] トヨタ自動車株式会社 制御電子プラットフォーム開発部
Vehicle Software Development Dept.,
TOYOTA MOTOR CORPORATION
[a)] 現在，WovenAlpha に所属

Vol.2021-ARC-244 No.9
Vol.2021-SLDM-194 No.9
Vol.2021-EMB-56 No.9
2021/3/25

2. 用語の定義

2.1 パーティショニング

パーティショニングの定義は規格によって様々であり，例えば自動車用機能安全規格 ISO 26262 では，「ある設計を実現するための，機能もしくはエレメントの分離」と定義されており，故障の伝播を避ける，もしくは障害を閉じ込めるために使用されるものとされている．本研究では，文献 [2] の「リソース X を共有するコンポーネント間で，リソース X を介して故障が伝播しないようリソース X を分離すること」という定義を参考に，「リソースを共有するシステム（VM）間で，共有リソースを介した影響が発生しないようリソースを分離すること」と定義する．

2.2 ハードウェアリソースの分類

VM 間のパーティショニングが妥当か判断するには，各 VM が使用するリソースを把握する必要がある．しかし，VM が使用するリソースはハイパーバイザによって割り当てられるエンドデバイス（e.g. GPU，ストレージデバイス，ネットワークデバイス）だけではなく，CPU やメモリと割り当てられたデバイスの間にあるバスやキャッシュなど，間接的に使用されるリソースも考慮しなければならない [3]．これらを区別するため，本研究では VM からアクセス可能なハードウェアリソースを明示的リソースと非明示的リソースの 2 つに分類する．

2.2.1 明示的リソース

VM から識別可能なリソースを明示的リソースと定義する．具体的には，完全仮想化，準仮想化，パススルーによって，VM からアクセスされるハードウェアリソースを明示的リソースとする．VM 間の分離度を分析するには，まず明示的リソースが他の VM と共有されていないか，されている場合は分離度を分析する．

2.2.2 非明示的リソース

特定の明示的リソースにアクセスした際に，間接的にアクセスが発生するハードウェアリソースを非明示的リソースと定義する．具体的な例としては，デバイスへのアクセスやデバイスからのアクセスに使用されるバスやキャッシュなどが挙げられる．明示的リソースの使用で発生するアクセスは CPU とデバイス間と，デバイスとメモリ間のアクセスに大別できる．これら 2 種類のアクセスが発生した際に使用されるリソースについても，他の VM と共有されていないか，されている場合は分離が十分であるか（以降，分離度）を分析する．

2.3 共有リソースの分離

リソースの分離手法は時間的分離と空間的分離の 2 つに分類できる [3]．空間的分離は，共有リソース内の領域を空間分割し，VM ごとに異なる領域を割り当てて使用する分離手法である．時間的分離は，共有リソースを時分割により，VM ごとにリソースにアクセスできる時間を制御することで，時間的に分離して使用する手法である．デバイスによっては，片方の実装だけでは分離が不十分となる場合がある．例えば，メインメモリは MMU によって，VM ごとにアクセス領域を制限できるため，空間的分離が実現されている．一方，時間的分離はなく，VM 間でのメモリ帯域の奪い合いにより，パーティショニングが崩壊する可能性がある．

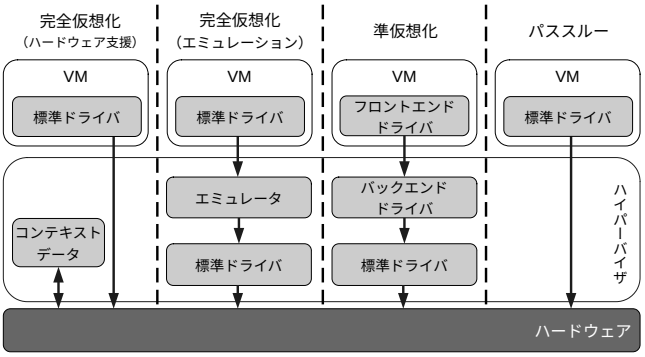


図 1: VM からのハードウェアへのアクセス方法（文献 [4] を基に作成）

本研究では，明示的リソースとそれに関連する非明示的リソースに対して，時間的分離と空間的分離の両方が存在することを十分条件として，各リソースにおけるパーティショニングの妥当性を判断する．一方の分離機能しか存在しない場合は，現状でデバイスの分離度が妥当か分析し，必要に応じて欠けている分離機能を実装する．

3. VM からハードウェアへのアクセス方法

VM からハードウェアへアクセスする方法は完全仮想化，準仮想化，パススルーの 3 つに大別できる．

3.1 完全仮想化

完全仮想化とは，実在するハードウェアの動作をハイパーバイザが完全に模倣し，VM に提供する方式である．ゲスト OS は標準のデバイスドライバで，仮想化されたデバイスを使用する．従って，移植のためにゲスト OS を書き換える必要がない．完全仮想化の主な方式としては，ハードウェア仮想化支援機能を用いる方法とエミュレーションを行う方法がある．

ハードウェア仮想化支援機能を使用する場合，VM はデバイスを直接使用し，デバイスを使用する VM が切り替わる際に，ハイパーバイザがコンテキストデータを切り替える．

エミュレーションを使用する場合は，デバイスを模倣する仮想レジスタや I/O ポートをハイパーバイザが VM に提供し，VM はこれらに対して標準のデバイスドライバを

用いてアクセスする．アクセスがあった際にはハイパーバイザがアクセスをトラップし，デバイス制御情報を受け取る．制御情報を受け取ったハイパーバイザは，制御情報を物理デバイス用に変換し，ハイパーバイザ内の標準ドライバを使用して物理デバイスを制御する．

3.2 準仮想化

準仮想化とは，ゲスト OS やドライバに仮想化を認識した処理を行うための修正を加え，ゲスト OS からデバイスに対する処理の実行をハイパーバイザに依頼する方式である．デバイスの制御情報をハイパーバイザへ受け渡すフロントエンドドライバをゲスト OS に追加し，制御情報を受け取るためバックエンドドライバをハイパーバイザに追加する．ゲスト OS からのデバイス制御情報は，フロントエンドドライバからハイパーバイザの内部通信機構を介し，ハイパーバイザ内のバックエンドドライバに渡される．その後，バックエンドドライバが標準ドライバ用に制御データを変換し，標準ドライバを使用して物理デバイスを制御する．

3.3 パススルー

パススルーとは，物理デバイスのレジスタやメモリ，I/O ポートへのアクセスを VM に許可し，ゲスト OS から物理デバイスに直接アクセスする方式である．ゲスト OS は，標準ドライバを使用して物理デバイスを直接操作する．

従来のパススルー（以降，専有パススルー）では，物理デバイスが 1 つの VM に専有されてしまう欠点が存在する．これを克服するため，いくつかの技術が考案されている．

1 つ目は，パススルーするハードウェアの仮想化支援機能を使用し，ハードウェアの機能や性能を分割して個別にパススルーを可能とする技術である．このような技術の代表としては，PCI-e の SR-IOV がある．1 つの物理デバイスを隔離された仮想機能（Virtual Function）に分割し，個別に VM にパススルーできる．ただし，仮想機能間での性能の分離は，ハードウェアの実装に依存するため，複数の VM からの同時利用により，競合が発生する可能性がある．

2 つ目は，仮想パススルー（Virtual Passthrough）[5] や媒介パススルー（Mediated Passthrough）[6] と呼ばれる方式である．パススルー先の VM を時分割で切り替えることにより，複数の VM が同じデバイスをパススルーで使用可能にする．割り当てる先の VM を切り替える際にはハイパーバイザがデバイスのコンテキストデータの切り替える．

上記のいずれの方式においても，デバイスを介した，他の VM が所有するメモリ領域やデバイスへの不正なメモリアクセスに注意する必要がある．次節では，このようなデバイスを経由した不正なアクセスについても触れる．

4. 関連研究

ハイパーバイザのセキュリティに関連したものについて詳細に述べる．これらの関連研究は，主に「ハイパーバイザの脆弱性対策」と「特定デバイスの分離度改善」の 2 種類に大別される．

4.1 ハイパーバイザの脆弱性対策

ハイパーバイザをパーティショニング技術として使用するには，ハイパーバイザの正常な動作が前提となる．一方，ハイパーバイザは複雑なソフトウェアであり，規模が大きいため，数多くの脆弱性が National Vulnerability Database（NVD）などの脆弱性データベースに報告されている．これらの脆弱性を分析し，対策を講じることで，ハイパーバイザのセキュリティを向上が図られている．

Botero ら [7] は，Xen と KVM の脆弱性を分析し，関連するハイパーバイザの機能，脆弱性を用いた攻撃の攻撃元と攻撃先の 3 つの観点で分類し，その対策を議論した．Patil ら [8] は，Xen と KVM の脆弱性を関連するハイパーバイザのコンポーネントと，脆弱性が作用する VM の実行段階を基に分類し，その対策を議論した．

4.2 特定デバイスの分離度改善

ハイパーバイザはハードウェアを抽象化し，時に多重化も行うことで，複数の VM を同一のハードウェア上で動作させる．この時，共有されるハードウェアが十分に分離されていなければ，VM 間で影響が発生し，パーティションが崩れる原因となりうる．従って，特定のデバイスに注目し，ハイパーバイザによる分離を改善することで，セキュリティ向上が図られている．

Ling ら [9] は，Xen における仮想化されたストレージデバイスに着目し，性能分離を改善した．デフォルトの Xen の I/O スケジューラでは，CFQ を用いて各 VM に均等にストレージデバイスのアクセス帯域を割り当てる．Ling らは I/O スケジューラを改良し，VM ごとにアクセス帯域の上限設定を可能にすることで，VM 間でのストレージへのアクセス競合の影響を軽減した．

Ben ら [10] は，Xen における DMA デバイスに注目し，IOMMU を用いてそのアクセス可能な領域を制限することで，分離度を改善した．DMA デバイスが VM にパススルーされた場合，VM は任意のアドレスを指定した DMA 要求を発行できる．悪意ある VM は，他の VM のメモリ領域をアクセス先に指定した DMA 要求を発行することで，他の VM のメモリ内容の書き換えや読み取りが行えてしまう．そこで，Ben らは IOMMU を用いて，DMA デバイスがアクセス可能なメモリ領域を VM が許可された領域のみに制限している．IOMMU とは，デバイスからのメモリア

Vol.2021-ARC-244 No.9
Vol.2021-SLDM-194 No.9
Vol.2021-EMB-56 No.9
2021/3/25

クセスに対し，仮想アドレス ⇔ 物理アドレスの変換を行うメモリ管理ユニット（MMU）である．この IOMMU のページテーブルはハイパーバイザのみが操作可能なため，デバイスからのアクセスが許可されるメモリ領域のエントリのみをページテーブルに追加することで，デバイスからアクセスできるメモリ領域を制限する．

デバイス間での不正な通信を防ぐ技術の一例として，PCI-e の Access Control Services（ACS）がある．通常，PCI-e デバイス間での DMA は IOMMU を経由しないため，デバイスごとにアクセスできる範囲が制限できず，不正なデバイス間通信が発生する可能性がある．異なる VM に割り当てられたデバイス間で不正な通信が発生した場合，VM 間のパーティションの崩壊に繋がる．そこで，ACS を使用し，PCI-e デバイス間の通信も IOMMU を経由させることで，PCI-e デバイス間での不正な通信を防ぐ [11].

5. 提案手法

本研究では，システム設計情報からパーティショニングの妥当性の分析する手法を提案する．SoC，ハイパーバイザ，OS，アプリケーションの設計情報の使用を想定する．それぞれの設計情報からは，以下の情報が得られることを前提とする．

SoC の設計情報

- SoC に搭載されているハードウェア部品（以降，ハードウェアモジュールと呼ぶ）
- ハードウェアモジュールの接続関係

ハイパーバイザの設計情報

- VM が使用するハードウェアモジュール（以降，ハードウェアリソースと呼ぶ）の分離機能の有無
- 完全仮想化/準仮想化/パススルー可能なハードウェアリソース

OS の設計情報

- 動作に必要なハードウェアリソース

アプリケーションの設計情報

- 動作に必要なハードウェアリソース

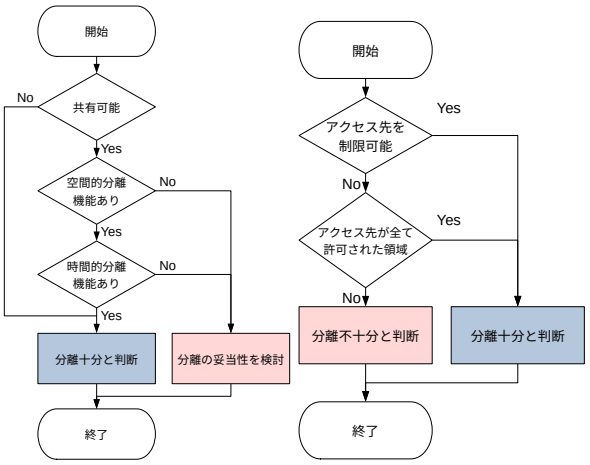
本手法の実行手順は以下の通りである．

（1）SoC 設計情報を基にハードウェアモジュールを列挙する

分析が必要なリソースの漏れを防ぐため，SoC の設計情報を基に SoC に搭載されているハードウェアモジュールを網羅的に列挙する．

（2）手順 1 で列挙したハードウェアモジュールのうち，明示的リソースを列挙する

ハイパーバイザの設計情報を用いて，手順 1 で列挙したハードウェアハードウェアモジュールから仮想化，準仮想化，パススルーによってアクセス可能なハードウェアリソースを VM ごとに列挙する．これらを明示的リソースとする．



(a) 全てのリソースを対象とする分析　(b) DMA デバイスを対象とする分析

図 2: ハードウェアリソースの分離度分析

（3）手順 2 で列挙した各明示的リソースのアクセスについて，アクセス時に間接的に使用される非明示的リソースを列挙する

手順 2 で特定した各明示的リソースについて，SoC の設計情報にあるモジュール間の接続関係の情報から，その明示的リソースへのアクセス時に間接的に使用されるリソースを列挙する．

具体的には，CPU と明示的リソース X 間のアクセス時に使用されるハードウェアリソースを列挙する．それらを明示的リソース X のアクセス時に使用される非明示的リソースとする．また，明示的リソース X が DMA を行うハードウェアリソースであれば，明示的リソース X とメモリ間のアクセス時に使用されるハードウェアリソースも非明示的リソースとする．

（4）手順 2 で列挙したリソースのうち，デバイス間通信を行うリソースを列挙する

デバイス間通信を行う 2 つのハードウェアリソースを，それぞれ異なる VM にパススルーした場合，ハードウェアリソース間で不正な通信が行われ，VM 間のパーティショニングの崩壊に繋がる可能性がある．よって，手順 1 で列挙したハードウェアリソースの内，デバイス間通信を行うものを列挙し，手順 5 で割り当ての妥当性を分析する．

（5）各 VM へのリソース割り当ての妥当性を分析する

OS/アプリケーションの設計情報を基に，各 VM が使用するハードウェアリソースを特定し，図 2 に基づいて分析する．

Vol.2021-ARC-244 No.9
Vol.2021-SLDM-194 No.9
Vol.2021-EMB-56 No.9
2021/3/25

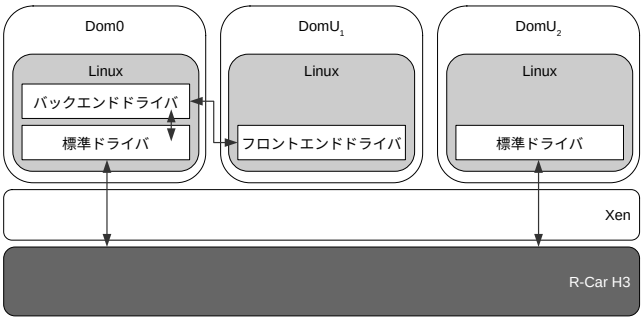


図 3: 対象システムのアーキテクチャ（文献 [12] を基に作成）

まず，各ハードウェアリソースに対し，図 2（a）に従い，分析する．VM 間で共有される可能性がある場合，それらに時間的分離機能と空間的分離機能がそれぞれ存在するかハイパーバイザの設計情報を基に分析する．

DMA やデバイス間通信を行うハードウェアリソースがパススルーされる場合，追加で図 2（b）に従い分析する．まず，SoC の設計情報を基にアクセス先を制限するハードウェア機構が存在するか，ハイパーバイザの設計情報を基に正しく設定可能か分析する．アクセス先を制限できない場合は，アクセス先となりうる全てのハードウェアリソースを同じ VM に割り当てる．

6. 仮想システムへの適用

本節では，5 節で述べた手法の適用例を示す．実験では，実際に自動車メーカーで先行検討されているシステム（図 3）を使用する．

6.1 対象としたシステム

6.1.1 SoC

本実験では SoC として R-Car H3 を使用する．R-Car H3 は Renesas Electronics が開発した車載情報システム向け SoC である．自動運転などの ECU として利用できるような処理性能をもつ．ISO 26262 の ASIL-B 対応や，セキュリティ機能が強化されている．ネットワークインタフェースとして CAN（Controller Area Network）や Ethernet，ビデオ機能としてビデオ表示やビデオ入力を搭載している．

6.1.2 ハイパーバイザ

本実験では，ハイパーバイザとして Xen ハイパーバイザを使用する．使用したバージョンは 4.12.0 の Xen on Arm[12] である．Xen は，オープンソースにより開発されている Type 1 ハイパーバイザで，CPU や I/O デバイスなどの計算機資源を抽象化し，ドメイン（VM と同義）と呼ばれる仮想計算機環境を提供する．Xen 自身の制御や他のドメインの管理を行うためのドメインを Domain0（Dom0）, ゲスト OS が動作するドメインを DomainU（DomU）と呼ぶ．デバイスのエミュレーション用のエミュレータや，準仮想化用のバックエンドドライバは，Dom0 で動作し，

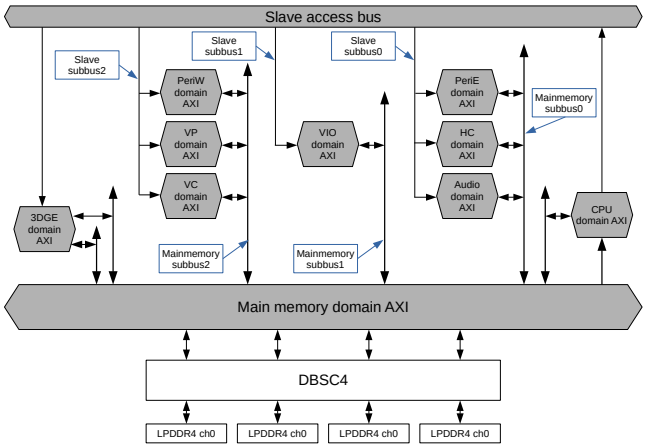


図 4: AXI − bus の接続関係

DomU からの操作に応じて物理デバイスにアクセスする．

6.1.3 OS

本実験では，ゲスト OS として Linux を想定する．カーネルバージョン 4.14.75 のものを使用する．OS の設計情報としては，Linux のソースコード [13] を使用する．

6.2 提案手法の適用

（1）SoC 設計情報を基にハードウェアモジュールを列挙する

SoC の設計情報として R-Car のハードウェアマニュアル [14] を使用し，ハードウェアモジュールを合計 103 種列挙した．

（2）手順 1 で列挙したハードウェアモジュールのうち，明示的リソースを列挙する

ハイパーバイザの設計情報として，Xen ハイパーバイザのソースコード [15] とマニュアル [16] を使用し，手順 1 で列挙したハードウェアモジュールのうち，明示的リソースを合計 34 種列挙した．対象システムにおける明示的リソースとそのアクセス方法を表 1 に示す．なお，Xen on Arm ではエミュレーションは行われないため，完全仮想化されるハードウェアリソースは，全てハードウェア仮想化支援機能によって仮想化される．また，Xen のパススルーはすべて専有パススルーである．

（3）手順 2 で列挙した各明示的リソースについて，アクセス時に間接的に使用される非明示的リソースを列挙する

SoC の設計情報を基に，手順 2 で列挙した各明示的リソースについて，CPU と明示的リソース間のアクセスに使用される非明示的リソースを列挙した． SoC 内の AXI-bus は，複数のサブドメインバスに分かれており，それらを相互接続するバスを「subbus」として図 4 のように識別し，分析した．列挙した結果の一部をリスト 1 に示す．リスト 1 は，CPU Core を根ノードとする木構造になっており，葉ノー

Vol.2021-ARC-244 No.9
Vol.2021-SLDM-194 No.9
Vol.2021-EMB-56 No.9
2021/3/25

リスト 1: 各明示的リソースへのアクセスに使用される非明示的リソース

```
CPU Core
 *--L1 Cache
     *--L2 Cache
         *--CPU domain AXI
             |--Slave access bus
             |   |--(Slave subbus0)
             |   |   |--HC domain AXI
             |   |   |   |--PCIE Controller
             |   |   |   |--SD/MMC
             |   |   |   |--SATA
             |   |   |   |--EHCI/OHCI
             |   |   |   *--USB3.0
```

リスト 2: DMA 時に使用される非明示的リソース

```
LPDDR4
 *--DBSC4
     *--Main memory domain AXI
         |--(Mainmemory subbus0)
         |   |--PeriE domain AXI
         |   |   *--I2C
         |   |--HC domain AXI
         |   |   |--PCIE Controller
         |   |   |--SD/MMC
         |   |   |--SATA
         |   |   |--EHCI/OHCI
         |   |   *--USB3.0
```

ドが明示的リソース，葉ノードから根ノードの間にある先祖ノードがアクセスに使用される非明示的リソースを示している．例えば，USB3.0 ポートへのアクセスには，CPU Core⇒L1 Cache⇒L2 Cache⇒CPU domain AXI⇒Slave access bus⇒(Slave subbus0)⇒HC domain AXI⇒USB3.0 という経路でアクセスされるので，L1 Cache∼HC domain AXI が USB3.0 へのアクセス時に使用される非明示的リソースとなる．さらに，DMA を行う明示的リソースに対して，メモリと明示的リソース間のアクセスに使用される非明示的リソースを列挙した．結果の一部をリスト 2 に示す．リスト 2 はメインメモリ（LPDDR4）を根ノードとする木構造になっており，葉ノードが DMA を行う明示的リソース，葉ノードから根ノードの間にある先祖ノードが DMA に使用される非明示的リソースを示している．例えば，I2C インタフェースへのアクセスには，I2C⇒PeriE domain AXI⇒(Mainmemory subbus0)⇒Main memory domain AXI⇒DMSC4⇒LPDDR4 という経路で DMA が行われるので，PeriE domain AXI∼DMSC4 が I2C からの DMA で使用される非明示的リソースとなる．

（4）手順 2 で列挙したリソースのうち，デバイス間通信を行うリソースを列挙する

SoC の設計情報を基に，デバイス間通信を行うデバイスを列挙した．R-Car H3 には，Audio 関連のデバイスの間で DMA を行うための DMA コントローラ（以降，Audio-DMACpp）が搭載されている．従って，この Audio-DMACpp を使用するデバイスは，相互通信を行う可能性がある．Audio-DMACpp を使用するデバイスは以下のデバイスである[*1]．

- Serial Sound Interface (SSI)
- ADSP
- Sampling Rate Converter (SRC)
- CTU + MIX + DVC (CMD)
- MediaLB I/F Local Memory (MLM)

これらのうち，パススルー可能なものは，SSI，ADSP，SRC である．

[*1] NDA により，一部省略する

（5）各 VM へのリソース割り当ての妥当性を分析する

本研究では，具体的なアプリケーションを想定しておらず，VM に対するリソースの割り当てが決定されていないため，すべての明示的リソースに対して共有の可能性と空間的/時間的分離機能の有無を分析する．さらに，同様の分析を一部の非明示的リソースに対しても行う．

6.3 分析結果

6.3.1 明示的リソースの分析

明示的リソースの共有の可能性と空間的/時間的分離機能の分析結果を表 1 に示す．

Xen のパススルーは全て専有パススルーであるため，パススルーされた明示的リソースが複数の VM 間で共有されることはない．従って，空間的/時間的分離は十分であると言える．ただし，DMA を行うデバイスをパススルーする場合は，IOMMU によってデバイスがアクセスできるメモリ空間が制限可能である必要がある．R-Car H3 には，Renesas Electronics 社の IOMMU の独自実装である IPMMU が搭載されている．そこで，パススルー可能な DMA を行うデバイスが全て IPMMU に接続されているか分析した． 分析時に作成したブロック図のイメージを図 5 に示す．網掛けのデバイスがパススルー可能なデバイスであり，各接続に振られた数値は，親となるデバイスが子を識別するために使用する識別子である．分析の結果，DMA を行うデバイスはすべて IPMMU に接続されていた．従って，不正な DMA を防ぐことができる．ただし，一部のデバイスは図 5 中のデバイス 0∼ デバイス 3 のように，IPMMU から振られた識別子が同じであった．そのよ

Vol.2021-ARC-244 No.9
Vol.2021-SLDM-194 No.9
Vol.2021-EMB-56 No.9
2021/3/25

表 1: 明示的リソースの分析結果（NDA により，一部結果を省略）

No.	Explicit Hardware Resource	Access Type	Shareable	Isolation		Isolation Technology	Isolation between VMs
				Space	Time		
(1)	Arm Cortex-A57	HV	✓	✓	✓	Scheduler + VHEs	✓
(2)	Arm Cortex-A53	HV	✓	✓	✓	Scheduler + VHEs	✓
(3)	External Bus Controller for LPDDR4/DDR3/DDR3L SDRAM (DBSC4)	HV	✓	✓	–	MMU	–
(4)	Serial-ATA Gen3	PV	✓	✓	–	VIRTIO	–
(5)	SD Card Host Interface (SDHI)	PV	✓	✓	–	VIRTIO	–
		PT	–	✓	✓	VFIO/IOMMU	✓
(6)	Multimedia Card Interface (MMC)	PV	✓	✓	–	VIRTIO	–
		PT	–	✓	✓	VFIO/IOMMU	✓
(7)	Ethernet AVB-IF	PV	✓	✓	✓	VIRTIO	✓
		PT	–	✓	✓	VFIO/IOMMU	✓
(8)	JTAG/SWD I/F	PT	–	✓	✓	VFIO	✓
(9)	General-purpose I/O (GPIO)	PT	–	✓	✓	VFIO	✓
(10)	Thermal sensor / Chip Internal Voltage Monitor (THS/CIVM*)	PT	–	✓	✓	VFIO	✓
(11)	Direct Memory Access Controller for System (SYS-DMAC)	PT	–	✓	✓	VFIO/IOMMU	✓
(12)	Direct Memory Access Controller for Audio (Audio-DMAC)	PT	–	✓	✓	VFIO/IOMMU	✓
(13)	Interrupt Controller (INTC)	PT	–	✓	✓	VFIO	✓
(14)	Multifunctional Interface (MFIS)	PT	–	✓	✓	VFIO	✓
(15)	3D Graphics Engine (3DGE)	PT	–	✓	✓	VFIO/IOMMU	✓
(16)	Display Unit (DU)	PT	–	✓	✓	VFIO/IOMMU	✓
(17)	Video Input Module (VIN) MIPI-CSI2 interface	PT	–	✓	✓	VFIO	✓
(18)	Video Input Module (VIN) digital interface	PT	–	✓	✓	VFIO/IOMMU	✓
(19)	[redacted]	PT	–	✓	✓	VFIO/IOMMU	✓
(20)	Video Signal Processor (VSPI)	PT	–	✓	✓	VFIO	✓
(21)	Video Signal Processor (VSPB)	PT	–	✓	✓	VFIO	✓
(22)	Video Signal Processor (VSPD)	PT	–	✓	✓	VFIO, VFIO/IOMMU	✓
(23)	Video Signal Processor (VSPDL)	PT	–	✓	✓	VFIO, VFIO/IOMMU	✓
(24)	Video Codec Processor (VCP4)	PT	–	✓	✓	VFIO	✓
(25)	Video Decoding Processor for inter-device video transfer (iVDP1C)	PT	–	✓	✓	VFIO	✓
(26)	Fine Display Processor (FDP1)	PT	–	✓	✓	VFIO	✓
(27)	Sampling Rate Converter Unit (SCU)	PT	–	✓	✓	VFIO	✓
(28)	Serial Sound Interface Unit (SSIU)	PT	–	✓	✓	VFIO	✓
(29)	USB2.0 Host (EHCI/OHCI)	PT	–	✓	✓	VFIO/IOMMU	✓
(30)	USB3.0 Host Controller	PT	–	✓	✓	VFIO/IOMMU	✓
(31)	PCI-e Controller	PT	–	✓	✓	VFIO/IOMMU	✓
(32)	IIC Bus Interface for DVFS (IIC for DVFS)	PT	–	✓	✓	VFIO/IOMMU	✓
(33)	I2C Bus Interface (I2C)	PT	–	✓	✓	VFIO/IOMMU	✓
(34)	Serial communication interface with FIFO (SCIF)	PT	–	✓	✓	VFIO/IOMMU	✓

HV: ハードウェア仮想化支援（Hardware Virtualized）, PV: 準仮想化（Paravirtualized）, PT: パススルー（Passthrough）

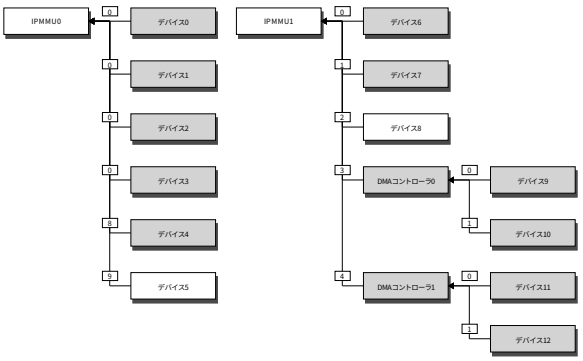


図 5: ipmmu の接続関係（NDA により，イメージ図を使用する）

うなデバイスに関しては，1 つでも VM にパススルーした場合，同じ識別を持つ他のデバイスも全て同じ VM に割り当てる必要がある．

ハードウェア仮想化支援による完全仮想化デバイスについては，複数の VM に共有される可能性があるため，空間的/時間的分離が十分であるか分析した．CPU の場合，仮想 CPU のスケジューリングによって時間的分離が，仮想 CPU の物理 CPU コアへの排他的割り当てによって空間的分離が可能であるため，分離は十分であると言える．メモリの場合は，MMU により VM がアクセスできるメモリ領域を制限可能であるため，空間的分離は十分である．ただし，メモリへのアクセス時間を VM ごとに制御する機能は Xen にはないため，時間的分離は不十分となる．

準仮想化デバイスも複数の VM に共有される可能性があるため，空間的/時間的分離が十分であるか分析する．例えば，(4) Serial-ATA Gen3，(5) SDHI，(6) MMC 等のストレージデバイスに関しては，VM ごとにアクセスできるパーティションを制限可能であるため，空間的分離は十分であると言える．一方，ストレージデバイスに対するアクセス時間を制御する機能は Xen にはないため，時間的分離は不十分となる．(7) Ethernet AVB-IF に関しては，仮想ブリッジを作成し，これに物理ネットワークインタフェース（物理 NIC）を接続する．各 VM には，仮想ブリッジに接続した個別の仮想ネットワークインタフェース（仮想

NIC）を割り当てるため，空間的分離は十分である．また，各仮想化 NIC から単位時間あたりに送信可能なデータ量を制限できるため，時間的分離も十分である．

手順 4 で列挙したデバイス間通信を行うリソースに関しては，IOMMU によるアクセス先の制限ができない．よって，図 2（b）に従い，アクセス先が全て許可された領域となるようにしなければならない．具体的には，AudioDMACpp を使用するデバイスの内，1 つでも VM にパススルーする場合，他のデバイスも同じ VM に割り当てを行う．

6.3.2 非明示的リソースの分析

非明示的リソースの共有されるかは，VM に対する明示的リソースが決定した後，リスト 1 とリスト 2 を基に判断する．異なる VM に割り当てた明示的リソース同士が共有の先祖を持つ場合は，それらが非明示的な共有リソースとなる．ここでは，多くのデバイスが共有する L2 Cache と AXI-bus ついて，空間的/時間的分離機能の有無を分析する．Xen には，VM ごとに使用する L2 Cache のキャッシュラインを制限する機能がなく，VM ごとに L2 Cache へのアクセス時間を制限する機能もない．よって，L2 Cache は空間的分離と時間的分離の両方が不十分となる．また，Xen には VM ごとに AXI-bus の使用時間を制限する機能がない．AXI-bus を空間的に分離するには，VM ごとに専用のバスを用意する必要があるが，それには新たなハードウェアを追加する必要となるため，空間的機能は存在しない．以上より，AXI-bus も空間的分離と時間的分離の両方が不十分となる．

6.4 分析結果の考察

本小節では，分離機能が不十分と判断したリソースの一部に対して，分離度を改善する方法について考察する．

6.4.1 メモリ

先程の分析では，VM ごとにメモリへのアクセス時間を制御する機能がないため，メモリの時間的分離が不十分であるとした．VM ごとにメモリへのアクセス時間を制御する方法としては，VM が使用している CPU コアでパフォーマンスカウンタを用いてキャッシュミスを監視する方法が考えられる．単位時間あたりの発生したキャッシュミス数が，あらかじめ設定した上限に達した場合，VM の仮想 CPU の実行を停止させる．これにより，単位時間あたりのメモリへのアクセス数を制限することができ，時間的分離を実現できる．実際にこのような機能を OS レベルで実現した研究として，Yun らの Memguard[17] がある．

6.4.2 ストレージデバイス

先程の分析では，VM ごとにストレージデバイスに対するアクセス時間を制御する機能がないため，時間的分離は不十分とした．Xen のデフォルトの I/O スケジューラでは，すべての VM に対して等しくアクセス時間を割り当てる．そこで，I/O スケジューラを拡張し，VM ごとにストレージデバイスへのアクセスのスループットを制限することで，時間的分離を実現できる [18]．

6.4.3 L2 Cache

先程の分析では，L2 Cache の空間的分離と時間的分離の両方が不十分とした．空間的分離に関しては，VM のメモリに対してキャッシュカラーリングを適用し，各 VM のメモリが異なるキャッシュラインに割り当てられるよう制御することで，空間的分離を実現できる [19]．時間的分離に関しては，VM が使用している CPU コアの L1 キャッシュミス数を監視する方法が考えられる．単位時間あたりのキャッシュミス数が，あらかじめ設定した上限に達した場合，VM の仮想 CPU の実行を停止させる．これにより，単位時間あたりの L2 Cache へのアクセス数を制限することができ，時間的分離を実現できる．この機能を OS レベルで実現した研究として，Bechtel らの研究 [1] がある．

6.4.4 AXI-bus

先程の分析では，AXI-bus も空間的分離と時間的分離の両方が不十分とした．空間的分離を実現するには，VM ごとに専用のバスを用意する必要があるが，それには新たなハードウェアの追加が必要であるため，ハイパーバイザの機能拡張のみでは対応できない．時間的分離に関しては，AXI-bus の QoS 機能 [20] の使用による実現が考えられる．AXI-bus の QoS 機能ではバスマスタごとに使用可能な帯域幅を制限できる．この制限を VM のスケジューリングに合わせて変更することで，VM ごとに使用できる AXI-bus の帯域幅を制御し，時間的分離を実現できると考えられる．

7. おわりに

本研究では，パーティショニングが崩れる要因の整理を行い，システム設計情報から事前にパーティショニングの妥当性を分析する手法の提案した．また，提案手法を仮想的な車載システムに適用した結果，明示的リソース 4 種と，CPU のキャッシュ, AXI-bus を含む, 複数の分離不十分なハードウェアリソースを発見した．

今後の課題としては，分析の対象にソフトウェアリソースも含むよう，手法を拡張をすることが挙げられる．また，類似する研究や整理された手法が存在しないため，他の手法との網羅性の比較ができていない．従って，実際の開発において使用される手法との比較も今後の課題として考えられる．

参考文献

[1] Bechtel, M. and Yun, H.: Denial-of-service attacks on shared cache in multicore: Analysis and prevention, RTAS, IEEE, pp. 357–367 (2019).
[2] 松原豊，高田広章：組込みシステムにおけるソフトウェア障害の安全対策，コンピュータソフトウェア，Vol. 30, No. 1, pp. 1_119–1_129（オンライン），DOI: 10.11309/jssst.30.1_119 (2013).

[3] VanderLeest, S. H.: ARINC 653 hypervisor, 29th Digital Avionics Systems Conference, IEEE, pp. 5–E (2010).
[4] Xen Project: Xen Project Software Overview: Guest Types, available from〈https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview#Guest_Types〉(accessed 2021-01-13)
[5] Xia, L., Lange, J., Dinda, P. and Bae, C.: Investigating virtual passthrough I/O on commodity devices, ACM SIGOPS Operating Systems Review, Vol. 43, No. 3, pp. 83–94 (2009).
[6] Tian, K., Dong, Y. and Cowperthwaite, D.: A Full {GPU} Virtualization Solution with Mediated Pass-Through, {USENIX}{ATC} 14, pp. 121–132 (2014).
[7] Perez-Botero, D., Szefer, J. and Lee, R. B.: Characterizing hypervisor vulnerabilities in cloud computing servers, Proc. *the 2013 international workshop on Security in cloud computing* pp. 3–10 (2013).
[8] Patil, R. and Modi, C.: An exhaustive survey on security concerns and solutions at different components of virtualization, Concurrency and Computation: Practice and Experience, Vol. 52, No. 1, pp. 1–38 (2019).
[9] Ling, X., Jin, H., Ibrahim, S., et al.: Efficient disk I/O scheduling with qos guarantee for xen-based hosting platforms, CCGRID 2012, IEEE, pp. 81–89 (2012).
[10] Ben-Yehuda, M., Mason, J., Xenidis, J., Krieger, O., Van Doorn, L., Nakajima, J., Mallick, A. and Wahlig, E.: Utilizing IOMMUs for virtualization in Linux and Xen, OLS' 06: The 2006 Ottawa Linux Symposium, Citeseer, pp. 71–86 (2006).
[11] Zhou, Z., Gligor, V. D., Newsome, J. and McCune, J. M.: Building verifiable trusted path on commodity x86 computers, 2012 IEEE symposium on security and privacy, IEEE, pp. 616–630 (2012).
[12] Xen Project: Xen ARM with Virtualization Extensions whitepaper, available from 〈https://wiki.xenproject.org/wiki/Xen_ARM_with_Virtualization_Extensions_whitepaper〉(accessed 2021-01-13)
[13] Linus Torvalds: Linux kernel source tree, available from 〈https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git〉(accessed 2021-01-13)
[14] Renesas Electronics: R-Car H3/M3 User's Manual: Hardware For H3/M3 Starter Kit, Rev.1.50
[15] Xen Project: Xen Project repository, available from 〈http://xenbits.xen.org/gitweb/?p=xen.git;a=summary*〉(accessed 2021-01-13)
[16] Xen Project: Xen Man Pages, available from 〈https://wiki.xenproject.org/wiki/Xen_Man_Pages〉(accessed 2021-01-13)
[17] Yun, H., Yao, G., Pellizzoni, et al.: Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms, RTAS, IEEE, pp. 55–64 (2013).
[18] Wu, S., Tao, S., Ling, X., et al.: iShare: Balancing I/O performance isolation and disk I/O efficiency in virtualized environments, Concurrency and Computation: Practice and Experience, Vol. 28, No. 2, pp. 386–399 (2016).
[19] Stabellini, S.: Cache Coloring: Interference-free Real-time Virtualization., available from 〈https://xenproject.org/2020/09/03/cache-coloring-interference-free-real-time-virtualization/〉(accessed 2021-01-13)
[20] Stevens, A.: Quality of Service (QoS) in ARM, Systems: An Overview, Technical report (2014).