情報処理学会論文誌 Vol.55 No.8 1830–1840 (Aug. 2014)

# 組込み仮想化のハードウェアI/Oアドレスマップ変換

請園 智玲[1,a)]

受付日 2013年11月13日, 採録日 2014年5月17日

**概要**：近年，組込みシステムの設計に仮想化を用いることが注目されてきている．組込み仮想化は，ソフトウェアポータビリティ，堅牢性，信頼性を向上させる．加えて，組込み仮想化はセキュアシステムの構築に貢献する．しかしながら，通常，仮想化はソフトウェアで実現されるため，ソフトウェアオーバヘッドが存在する．本研究では，特に組込み仮想化のI/Oアドレスマップ変換部のソフトウェアオーバヘッドに着目し，ハードウェアI/Oアドレスマップ変換器を提案する．

**キーワード**：仮想化，組込みシステム，I/Oポートアドレス

# Hardware Translation of I/O Address Map in Embedded Virtualization

Tomoaki Ukezono[1,a)]

Received: November 13, 2013, Accepted: May 17, 2014

***Abstract:*** Recently, exploiting virtualization to design of embedded system is getting a lot more attention lately. Embedded virtualizations can improve software portability, robustness and dependability. In addition, Embedded virtualizations can contribute building secure system. However, In generally, virtualizations have software overheads because the virtualizations is implemented by software. In this paper, especially, we focus on software overheads for translation of I/O address map and propose the hardware translation of I/O address map.

***Keywords:*** virtualization, embedded system, I/O port address

## 1. はじめに

### 1.1 組込み仮想化

2000年代後半ごろから，組込みシステムの開発で仮想化ソフトウェア（VMM: Virtual Machine Monitor）を利用すること（組込み仮想化）が注目を浴びている．本研究では，この組込み向け仮想化ソフトウェアを組込みVMMと呼ぶ．組込みVMMは，汎用とは異なり，VM（ゲストシステム）間のリソース共有を目的として利用されず，仮想化による副次的要因を目的として利用される．組込み仮想化がもたらす4つの恩恵を以下に列挙する．

- ソフトウェアポータビリティの向上
- 堅牢性の向上
- 信頼性の向上
- セキュアシステムの構築

ソフトウェアポータビリティの向上は組込みシステムの製品間でソフトウェアを移植する際の手間の軽減を意味する．汎用システムと異なり，組込みシステムでは，特にI/O周りのハードウェア資源制御の抽象化が成されないままソフトウェア開発が進み，組込みアプリケーションのソースコードレベルでのハードウェア依存が強い場合が多い．このような場合でも，VMMが移植前のハードウェアの構成を完全にエミュレート（完全仮想化）できれば，移植対象のソフトウェア自体の修正は要しない．これが実現できれば，ソフトウェア修正とそれにともなうテストの工数を削減できるとともに，修正による不具合発生を抑止する効果がある．

堅牢性の向上はVMMのライブマイグレーション [2], [3] の特性から得られるものである．組込みシステムにはその

[1] 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology (JAIST), Nomi, Ishikawa 923–1211, Japan
[a)] t-ukezo@jaist.ac.jp

アプリケーションの重要性から，絶対に止めてはいけないシステムが存在する．そのようなシステムでも，実行しながらのソフトウェア更新やシステム fault 時の実行の巻き戻し（チェックポイント回帰）が VMM のライブマイグレーションの特性により容易に実現できる．

信頼性の向上は，システムの failure が VM を超えて影響しにくい VMM の特性を利用したもので，本当に止めてはいけない機能とそれ以外の機能を VM を分けて配置することで，システム設計者が論理的に failure の影響範囲を制御することが可能となり，より信頼性の高いシステムの構築が可能となる．

セキュアシステムの構築は VM に CPU 特権を与えないことで実現できる．VMM に CPU の特権動作を集約し，特権を要するコードサイズを相対的に小さくすることで，外部からの攻撃の対象を小さくすることができる．また，VM は実行イメージとしてカプセル化される特性を持つ．これもセキュリティ性能の向上に貢献するとともに，新しい形のソフトウェア部品のライセンス供給の手段となりうる．

このように，VMM は組込みシステムの開発に大きな利点をもたらす一方，VMM がソフトウェアで実装されることから，その命令実行オーバヘッドが問題となり，実時間処理が重要な組込みシステムへの適用を妨げている．

### 1.2 組込み仮想化の全体像と I/O 制御の関係

本研究は 1.1 節で述べた 4 つの組込み仮想化の恩恵のうち，ソフトウェアポータビリティの向上に着目し，完全仮想化による実装を対象とする．本節では，まず，完全仮想化とソフトウェアのポータビリティ向上に関して議論し，次に仮想化の全体像として，どのような機能が仮想化に必要かを議論する．その後に，本研究が対象とする仮想化の I/O 制御の詳細を示す．

#### 1.2.1 完全仮想化と組込みソフトウェアのポータビリティ向上

一般的に組込みシステム開発は，汎用システムに比べソフトウェアの製造および検証が難しい．このため，既存の製品で作られたプログラムコードを新しい製品で流用する重要性が汎用システムよりも高い．しかしながら，組込みプログラムの開発では，往々にしてハードウェアリソースの抽象化がなされず，移植性の低いソフトウェア資産が多い．たとえば，I/O ハンドルでは，ハードコーディングの定数を I/O アドレスとして扱い，アプリケーションレベルで操作する場合がある．このようなソフトウェアを新規開発のハードウェアに移植するためには，相応の再コーディングと検証のための時間を必要とする．このことから，組み込みシステム開発で，ソフトウェアの移植性を高めるために仮想化を導入することは，システムの開発工数を短縮するために有用である．組込みシステムにおいて完全仮想化が実現すれば，移植前のソースコードおよびバイナリコードに手をつけることなく，ソフトウェアのポータビリティを確保できる．

#### 1.2.2 仮想化の全体像

仮想化において，各 VM（ゲスト OS）は駆動するハードウェア上で唯一動作していると思い込み，動作するが，実際には複数の VM が同一ハードウェア上で動作することになる．その際に，共有/割り当てされなければならないハードウェア資源は競合を起こさないように管理される必要がある．このハードウェア管理機能を持つソフトウェアを VMM と呼ぶ．VMM には大きく分けて 3 つの機能がある．

- CPU コアの資源管理
- 主記憶の資源管理
- I/O の資源管理

CPU コアの資源管理は VM 間で CPU コアの資源共有を行う機能で，あるときに CPU コアの資源をどの VM に割り当てるかを決め，コア上で実行可能な VM を切り替える動作を行う．この機能の実現には複数の VM の実行スケジューリングと，VM スイッチ機能が必要となる．VM スイッチ機能は OS のコンテキストスイッチ機能と同様の機能であり，VM 切替時に CPU コアで実行中の VM のアーキテクチャルステートを主記憶にセーブし，次に実行する VM のアーキテクチャルステートをリストアする．主記憶の資源管理は各 VM がメモリのどの領域を使用するか決定し，管理する機能である．これは OS の主記憶の管理機能（ページング）と同様の機能である．この主記憶の管理機能のサポートハードウェアとして，複数の VM が持つページテーブルを統合的に管理できる Intel VT-x [14] などが存在する．I/O の資源管理はシステムに接続されるペリフェラル機器を複数の VM に対し共有/割り当てする機能である．VMM が I/O ペリフェラル機器に対して権限制御を行うサポートハードウェアとして Intel VT-d [13] などが存在する．

#### 1.2.3 仮想化の I/O 制御

本研究は 1.2.2 項で示した 3 つの機能の中で，I/O の資源管理機能に着目し，この機能のハードウェア化を行うことでソフトウェアオーバヘッドを削減する．

一般的な VMM は Virtualized I/O [21] と Directed I/O [22] の 2 種類の I/O の資源管理手法を持つ．**図 1** に Virtualized I/O と Directed I/O と本研究の提案手法の概念図を示す．Virtualized I/O は VM が物理 I/O ポートを直接制御せず，仮想ハードウェアへの制御を行う．それにともない，VMM がその仮想ハードウェアをソフトウェアエミュレーションし，それに対応する物理ハードウェアのドライバ（Host Driver）を VMM 内で駆動させ，意味的に同じデバイスの動作に変換する手法である．この手法を採用することで，I/O デバイスの VM 間共有が可能となり，

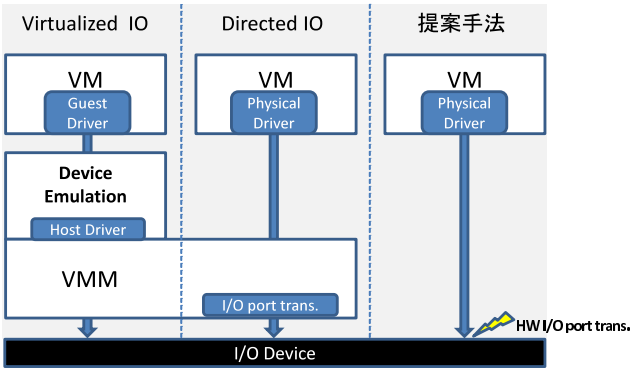


図 1 仮想化における I/O 制御手法と提案手法
**Fig. 1** I/O control technique and proposed technique in virtualization.

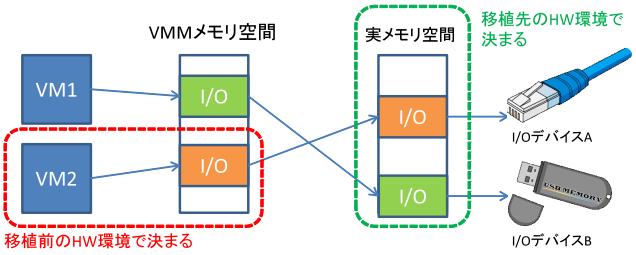


図 2 I/O ポートアドレス変換の概念図
**Fig. 2** Overview of I/O port address translation.

VM の I/O ハンドリングに柔軟性を与える．その一方で，仮想デバイスエミュレーションによる多大なソフトウェアオーバヘッドを必要とし，汎用システムであっても，ネットワークインターフェースなどの速度重視の I/O デバイスでは，利用が避けられる傾向にある．

Directed I/O は VM が直接 I/O デバイスのポートを制御する手法である．汎用システムでは，Intel VT-d が持つ IOMMU 制御機構のパススルー保護モデルがこれにあたる．Directed I/O は VM 間の I/O デバイス共有ができず，VM がドライバを駆動させることが条件となるが，VMM が I/O ポートの管理（I/O ポートアドレス変換）のみを行うことで，Virtualized I/O に比べ格段に性能が向上する．組込み仮想化では Directed I/O 方式を採用し，I/O デバイスの応答性を高めることが重視される．

組込みソフトウェアのポータビリティ向上の観点から，移植前のプラットフォームの I/O デバイスと移植後のプラットフォームの I/O デバイスが同じであれば，その制御を行うソフトウェア一式を VM 実行イメージ単位で，修正を必要とせずに移植できる（完全仮想化できる）ことから，Directed I/O 方式は有効な手法となる．

#### 1.2.4 I/O ポートアドレス変換

図 2 に Directed I/O における I/O ポートアドレス変換の概念図を示す．VM1 と VM2 は新しいプラットフォームに移植されてきたゲストシステム（VM）であるとする．図では，VM1 が I/O デバイス B を，VM2 が I/O デバイス A を制御するシステム構成である．VM の実行イメージをそのまま移植してきただけでは，移植前の I/O ポートアドレスが移植後のプラットフォームと異なる場合に，各 VM は対応する I/O デバイスを直接制御できない．組込みシステムで完全仮想化を実現する場合，このポートアドレスの違いが問題となる．アプリケーションレベルで I/O ポート制御を行う VM の場合はソースコードの修正が必要になり，ドライバで制御する場合はシステムソフトウェア環境に修正が必要となる．しかしながら，完全仮想化を実現する場合，VM を修正することができないため，VM は移植前のプラットフォームの I/O ポートを参照し続けることになる．そこで Directed I/O 方式を採用する場合，VMM が I/O ポートアドレスの変換を行う必要がある．VMM は移植前と移植後のプラットフォームの I/O ポートアドレスマップの対応を知っていることから，実行時にソフトウェアで動的にアドレスを変換することで，各 VM に対応した I/O デバイスを直接動作をさせることができる．

### 1.3 本研究の狙い

1.2.2 項で示した VMM の 3 つの資源管理機能は一般的にはソフトウェアによって実現されることから，すべてにソフトウェアオーバヘッドが存在する．このため，VMM はシステムのリアルタイム性を損ねるとともに，要求仕様を満たすハードウェアは高周波数駆動が必要となり，消費電力や製品単価を増大させる可能性を持つ．本研究は 1.2.4 項で示した完全仮想化を実現するための I/O ポートアドレス変換処理に着目する．Directed I/O 方式を採用する VMM の I/O ポートアドレス変換処理は VM 内で当該ポートを参照するデータ転送命令の命令エミュレーションによる処理を必要とするため，オーバヘッドの割合が大きい．このことから，本研究ではこの I/O ポートアドレス変換処理をハードウェア化する．これにより，組込み仮想化が I/O ポートアドレス変換のために必要とするソフトウェアオーバヘッドを排除できる．また，アドレス変換はハードウェア化に適した処理であり，複雑な制御を含まずに比較的平易な回路で実現できる．本研究はこの I/O ポートアドレス変換ハードウェアを System-on-Chip（SoC）の機能の 1 つとして位置づけることで，新規開発するワンチップ構成の組込み向けプラットフォームの IP コア（ソフトマクロ）の選択肢の 1 つとして提案する．

## 2. ハードウェア I/O ポートアドレス変換

### 2.1 VMM（ソフトウェア）の処理概要

図 3 に，VM と VMM の階層構造とその間の実行遷移を示す．図 3 は Intel VT-x [14] に代表される仮想化サポートハードウェアが搭載されない CPU での完全ソフトウェア実装の例である．VMM が 1 つ以上の VM を管理し，VM は 1 つ以上のユーザプロセスを管理する形となっている．

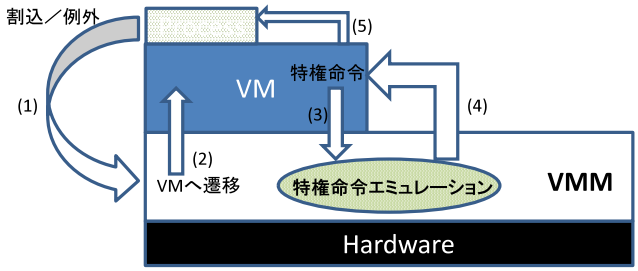


図 3 VM と VMM の階層構造と実行遷移
**Fig. 3** Layered Structure and excecution transition between VM and VMM.

図中の番号に従い VM と VMM の動作遷移を説明する．

(1) ユーザプロセスの実行中に共有資源への参照が必要な割込みまたは例外が発生すると，まず VMM に動作が遷移する．

(2) その後，VMM は VM の適切なハンドラを呼び出す[*1]．

(3) ハンドラ内で特権が必要な命令が実行された場合，再度 VMM が呼ばれ，その命令のエミュレーションを行う．

(4) VMM はエミュレーションが完了した後，VM のハンドラに復帰する[*2]．

(5) VM はハンドラが終了したあとユーザプロセスに復帰する．

この構造の設計を採用する場合，VM は VMM の存在を知らずに（VMM を視野に入れて修正をしなくとも）直接ハードウェア上で動作していてるつもりで処理することが可能となる．この構造の設計を実装するためのキーポイントはトラップベクトルの制御にある．通常，OS はハードウェアが指定する番地にトラップベクトルを配置し，そのトラップベクトルを起点に OS の各機能を実装したプログラムへ遷移する．VMM はトラップベクトルを乗っ取る形で実装される．**図 4** に VM と VMM のトラップベクトルの構造の概要を示す．

ハードウェアが指定するトラップベクトルを乗っ取ることにより，割込み/例外発生時に必ず VMM に制御が移る．VMM は必要な処理をした後で，VM のトラップベクトルの適切なエントリをコールする．コールの後，VM がハードウェア資源を要求した場合は例外としてハンドリングし，再び VMM に制御が移る．VMM は資源を要求した VM 内の命令を解析し，VMM が VM に変わりハードウェア管理（命令エミュレーション）を行う．

### 2.2 VMM による I/O アドレスマップ変換

通常，VM の I/O ポートへの参照は，特権を要するデータ転送命令のエミュレーションによって実現する方式が採

[*1] 特に (1) と (2) の動作は Intel VT-x など仮想化サポートハードウェアで省略することが可能である．
[*2] VMM のエミュレーション内容によっては直接ユーザプロセスに復帰する場合もある．

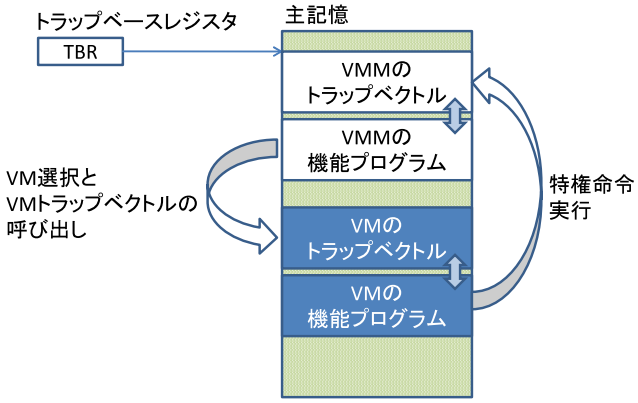


図 4 VM と VMM のトラップベクトルの構造の概要
**Fig. 4** Overview of structure of trap vector in VM and VMM.

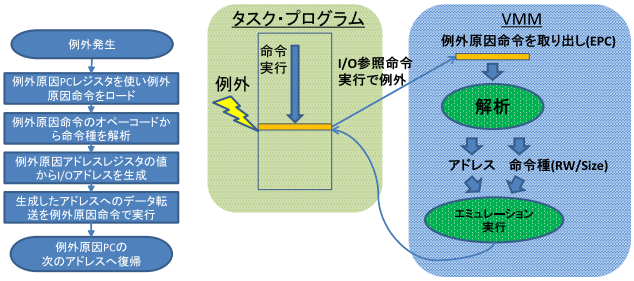


図 5 ソフトウェア I/O アドレスマップ変換処理のフローチャートと対応する処理概要
**Fig. 5** Flow chart of software I/O address map translation and overview of individual process.

用される．VMM が命令単位の制御を行えば，どの複数ある VM がどの I/O ポートを参照するか（または，参照してはいけないか）を厳密に制御できる．命令単位のソフトウェアによる I/O アドレスマップ変換処理のフローチャートとそれに対応する処理概要を**図 5** に示す．

図 5 で示す処理は図 3 の処理概要の (3) と (4) に相当する．VMM は VM が移植前のプラットフォームの I/O ポートを参照する場合に，例外を発生させるように，あらかじめ MMU を制御する．VMM がすべての I/O アドレスマップを参照不可にして例外を発生させることは既存の MMU のアーキテクチャで可能である．一般的な CPU には Precise Exception をサポートするために，例外原因命令のプログラムカウンタを保存する例外原因 PC レジスタと，MMU 例外をハンドルするために，MMU 例外を起こしたアドレスを保持する例外アドレスレジスタが存在する．まず，例外原因 PC レジスタからアドレスを取得し，例外原因命令をロードする．次にその命令のオペコードをもとに命令を解析し，データ参照命令の種類を判別する．このとき，例外アドレスレジスタには移植前のプラットフォームの I/O アドレスマップ内のアドレスが保持されていることから，そのアドレスを変換し，移植後のプラットフォームの I/O ポートのアドレスを生成する．最後に，解析した命令種と同じデータ転送命令を変換したアドレスで実行する．これ

ら動作を本研究では命令エミュレーションと呼ぶ．例外から復帰するときは，例外を起こした命令はすでに命令エミュレーションを終えているため，例外を起こした命令の次に実行予定の命令に復帰する．

これら一連のソフトウェア処理が組込み仮想化におけるI/Oポート参照の動作を遅くしている．本研究はこれらのソフトウェア処理を代替するハードウェアを提案し，I/Oデバイス参照速度を向上させる．

### 2.3 提案ハードウェア

**図 6** に本研究で提案するI/Oアドレスマップ変換ハードウェアを示す．提案ハードウェアの構造はTLBと酷似している．ただし，TLBとは大きく用途が異なり，TLBが仮想–物理アドレスの変換に用いるのに対し，提案ハードウェアはI/Oアドレスマップ領域のみに限定した物理–物理アドレス変換を行う．SoC設計時にハードウェア設計者がI/Oポートアドレスマップを任意に設定できる状況であっても，複数のVMが違う機器を同じI/Oアドレスとして参照する可能性があるため，本提案ハードウェアは必要となる．

図6のハードウェアは変換前アドレスと変換後アドレスが，対で登録されている連想メモリを用いる．これを変換前アドレスで参照し，変換後アドレスを得る．テーブルのエントリには，格納された変換情報が有効なVMの番号が格納されており，このVMの番号はVMM管理している．また，このハードウェアは現在実行中のVMの番号を保存するレジスタを保有し，参照時にはそのレジスタとエントリのVMの番号が比較される．これは複数のVMが同時に実行される場合，たとえば，VM1とVM2が同一のI/Oアドレスを参照するが，システム上は別のI/Oアドレスを参照する場合に意味がある．提案ハードウェアはSoCの設計上，MMUとI/Oコントローラの間のアドレスラインに接続される．

本研究で性能評価を行ったイーサネットデバイスをI/Oデバイスに持つ評価ボードを例にあげれば，イーサネットの通信制御に必要な基本のI/Oポートは18個であり，各ポートのサイズは4バイトである．これに加え，送受信バッファの個数に応じポート数が増加する．本研究の実験環境では，バッファディスクリプタ数を256に設定した．このバッファディスクリプタのポートも同様に各4バイトである．そのため，イーサネットに関してのみの制御で，274ポート必要となる．この274ポートを4バイト粒度で変換した場合，274エントリのテーブルエントリが必要とされるが，アドレス的に連続する場合は2のべき乗単位で粒度を上げ，エントリを結合することでエントリ数を削減することができる．

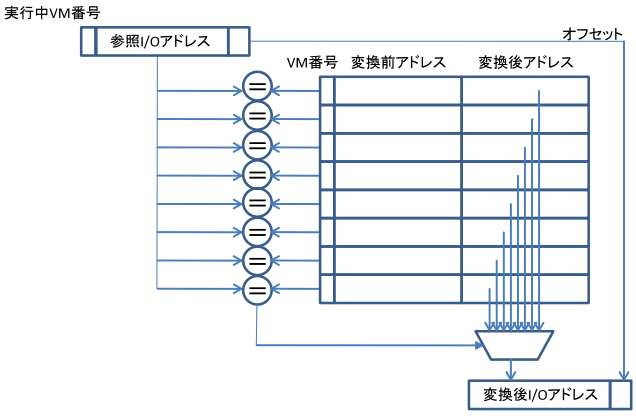


**図 6** I/Oアドレスマップ変換ハードウェア
**Fig. 6** Hardware of I/O address map translation.

評価のイーサネット通信実験はOSのLinuxとLinux用イーサネットドライバを使用した．Linuxのイーサネットドライバのソースコードから調査したところ，すべてのI/Oポートアドレスがアドレス的に連続していた．組込みシステムにおいて，I/Oポートのアドレス的連続性はハードウェア設計者が決める．単一のI/Oデバイスにおいて，データ転送命令の転送サイズが規定するアライメントの問題を除き，アドレス空間にI/Oポートを散在させる意味はなく，連続させるのはプログラマビリティの観点（C言語の構造体の利用など）からも妥当である．このため，イーサネットのみを扱う場合は32ビットアドレス空間の2Kバイト粒度（アドレス上位21ビット）を変換するエントリを用意すれば，1エントリで提案ハードウェアが実現可能である．

また，同時に複数のデバイスを扱う場合を考察すると，イーサネットは複雑な制御を持ち，他のI/Oと比べてもポート数が多いことから，2Kバイトの粒度は十分に大きい設定であると考えられる．このままでは，保護粒度の観点から，セキュアなシステム構成を構築できない恐れがある．しかしながら，提案手法と既存のCPUを利用した完全なソフトウェア実装のVMMとの違いは，SoC設計において，本提案手法を導入するのも，I/Oポートの物理アドレス空間への配置を決定するのも同一のハードウェア設計者である点である．ハードウェア設計者が仮想化のためのI/Oアドレス変換が2Kバイトの粒度であることを考慮に入れ，物理アドレス空間に各デバイスのI/Oポートを2Kバイトアラインで連続して配置すれば，VMが割り当てられていないI/Oデバイスに干渉する問題は避けられる．また，十分に大きい粒度の変換を提供することで，同時にアクティベイトする必要があるI/Oデバイスの上限数分のエントリを用意すれば，各システム構成に対応できることになり，提案ハードウェアのエントリ数の設計も単純になる．

上記のことから，本研究は，本稿の評価におけるI/Oアドレスマップ変換ハードウェアのエントリは同時にアクティブにするI/Oデバイスの上限数分，変換粒度は2Kバイトで提供することが妥当であると結論づける．しかしな

がら，この粒度は本稿の評価環境における，エントリ数と変換粒度の妥当な決定であり，すべての組込み SoC に適用できるわけではない．本稿の考察と同様にシステムに接続する I/O デバイスの数と種類によって妥当なエントリ数と変換粒度を，組込み SoC 設計時に考える必要がある．

### 2.4 提案ハードウェアと汎用システムにおける既存手法の差異の議論

提案ハードウェアは I/O ポートアドレスに対しアドレス変換を行うハードウェアである．通常，仮想記憶をサポートする CPU にはアドレス変換を行うハードウェアである TLB が存在する．本章では，TLB を応用する汎用システム向けの既存手法を対象に，提案手法との差異を議論する．

I/O アドレスを管理する方法として，既存の TLB を用いて VMM がソフトウェア制御で I/O アドレス変換をする方法が考えられる．汎用システムでは，Intel VT-x などの仮想化サポートを実現した MMU が EPT（Extended Page Table）を用いて VMM が各 VM のページ管理に介入し VM の管理する仮想アドレス空間から直接物理アドレスに変換する．この仕組みを用いれば，I/O ポートアドレス変換を行うことが論理的には可能である．

この汎用システムにおける既存手法との差異を論じるにあたり，まず，本研究の最も重要な前提が，提案ハードウェアが組込み SoC 向けの機能部品（ソフトマクロ）として位置づけられる点である．汎用システムにおいては，ハードウェアは様々な利用形態で使用されることを想定し，様々な機能でデータパスやメモリなどを極力共有化させる設計を行う．一方，組込み向け SoC では，実行されるシステムソフトウェアおよびアプリケーションが限定されるため，用途特化に最適なハードウェアが設計可能である．本研究はこの点に着目し，組込み仮想化を SoC 機能として実装することを出発点としている．

TLB は，本来，OS のページテーブルに限定したキャッシュメモリであり，その機能を維持しながら I/O ポートアドレス変換に流用するためには，VMM が各 VM のページ管理方針に介入しなければならない．これは組込み向けであるにもかかわらず，VMM のメモリ管理を複雑にし，メモリ管理のためのソフトウェアオーバヘッドを増大させる．また，組込みソフトウェアのポータビリティ向上のためには，既存の仮想アドレスを用いない（OS のない）VM の移植を行う必要があり，それらの混在を許さなければならず，さらに VMM のメモリ管理は複雑となる．加えて，TLB は OS のメモリ管理のページ粒度で最小変換粒度が 4 K バイト〜8 K バイトと I/O ポート制御の単位と比べ大きく設定されており，VM 間の I/O 機器制御の独立性を維持する場合，I/O アドレスマップのハードウェア設定が OS のページサイズに依存する不自然な状態が発生する．

本研究の本質は組込み SoC と仮想化の組合せを対象にした場合に，単純かつ妥当なサイズの専用 SoC 部品を導入することで，VMM 制御の複雑さを解消するという点にある．およそ 40 年間の仮想化の歴史 [1] で，仮想化は当時，とても高価な汎用システムを複数のユーザで共有する目的から導入された．時を経て現在，組込み仮想化はハードウェアリソースの共有ではなく，1.1 節で示した副次的要因に主眼をおいて利用されるため，汎用システムの設計マナーでシステムを設計することは，必ずしも最適ではない．本研究はこの主張を持ち，同時にこれが汎用システムの既存手法との差異であると位置づける．

## 3. 評価

### 3.1 評価環境

#### 3.1.1 FPGA ボード

評価環境として Digilent 社製 Atlys$^{TM}$ Spartan-6 FPGA 開発ボード [15] の FPGA 上に OpenRISC 1000 [16] 命令セットを採用した OR1200 プロセッサ [17] と ORPSoC [18] を実装し，構築した．Spartan-6 FPGA 開発ボードには DDR2 SDRAM が 128 MB，1 Gbit ether phy，HDMI 入出力，AC-97 audio, SPI Flash メモリが 16 MB，USB UART，USB HID Host，LED，スライドスイッチ，プッシュボタンの I/O が用意されている．本研究の評価では，I/O 性能を評価する対象としてイーサネットと USB UART に着目した．イーサネットは他の I/O の中でも，とりわけ高い動作速度が要求される入出力デバイスであり，VMM の I/O アドレスマップ変換時のソフトウェアオーバヘッドの影響が大きいと判断したためである．USB UART はシステムのシリアルコンソールやペリフェラルデバイスの制御に使用されることから，汎用性の高い I/O デバイスであると判断したためである．

#### 3.1.2 SoC 設計環境

OR1200 プロセッサは OpenCores プロジェクトのホームページより Verilog HDL で記述された RTL ソースを取得し，使用した．当該 FPGA 開発ボードは ORPSoC のリファレンスプラットフォームであるため，各種 I/O を制御するためのコントローラが Verilog HDL の RTL ソースで用意されている．本稿の評価で使用するイーサネットコントローラと USB UART コントローラはこの ORPSoC 内の RTL ソースで提供されるものを使用し，Spartan-6 上に実装した．

図 6 で示した提案ハードウェアは ORPSoC の機能の 1 つとして，Verilog HDL で記述した．イーサネットと USB UART に限った性能評価では 1 エントリのみが必要であり，この実装ではエントリが増加した場合のハードウェア量と遅延量のスケーラビリティが分からないため，複数デバイスをハンドルする場合のハードウェアの指標として，32 エントリの場合の提案ハードウェアを別途実装しハードウェア量と遅延を測定した．この実装は図 6 の白矩形部

表 1 ハードウェア設計環境
**Table 1** Environment for hardware design.

| | |
|---|---|
| 論理合成ツール | Xilinx xst14.4 |
| マッピングツール | Xilinx map14.4 |
| 配線ツール | Xilinx par14.4 |
| ロジック・アナライザ | Xilinx ChipScope Pro 14.4 |

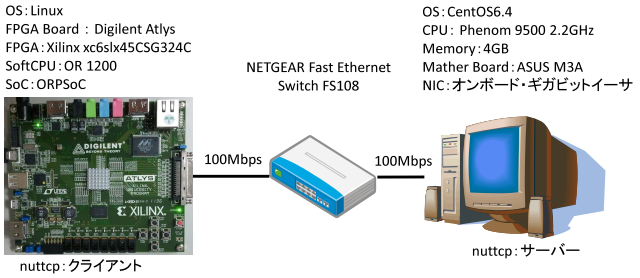


図 7 イーサネットの I/O 性能評価環境
**Fig. 7** Environment for performance evaluation of ethernet I/O.

分のメモリを定数記述した場合と BRAM 実装した場合の 2 種類を用意した．

本研究の評価で使用したハードウェア設計環境を**表 1** に示す．

### 3.1.3 システムソフトウェア環境

VM となる OS には Linux を選択した．これは性能評価で一般的に普及・利用されているアプリケーションをベンチマークとして採用できるためである．I/O アドレスマップ変換の効果を確かめるために，Linux 上のイーサネットドライバと USB UART ドライバの I/O ポートアドレスは ORPSoC 内で定義されているイーサネットコントローラと USB UART コントローラの I/O アドレスマップと異なるアドレスを設定した．

VMM は評価対象となる部分のみを OpenRISC のアセンブリ言語を用いて実装した．図 4 で示すトラップベクトルを実装し，I/O アドレスマップ変換に関係する部分以外はすべて Linux のトラップベクトルにリダイレクトする構造とした．I/O アドレスマップ変換部は図 5 で示すフローチャートに沿って実装している．

### 3.1.4 I/O 機器接続環境とベンチマーク

イーサネットは**図 7** で示す接続を行い通信させた．通信性能を計測するためにアプリケーションソフトウェア nuttcp [19] をベンチマークとして使用した．nuttcp は TCP/UDP のネットワークテストツールであり，インターネット上のホスト間のネットワーク状態の調査・検証に一般的に用いられるツールである．nuttcp はクライアント–サーバ間でのイーサネット接続の 1 秒間のデータ通信量を上りと下りに分けて計測することができる．本研究の評価では，TCP 接続において，上りと下りに分けて 100 回計測し，通信速度（Mbps）の算術平均をとって評価した．

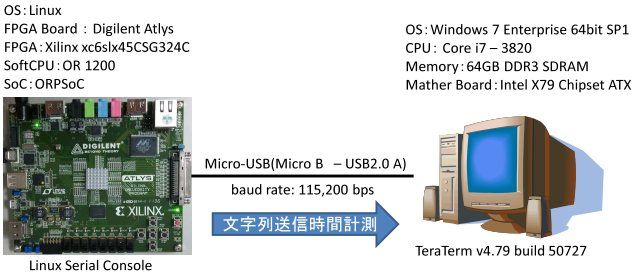


図 8 USB UART の I/O 性能評価環境
**Fig. 8** Environment for performance evaluation of USB UART.

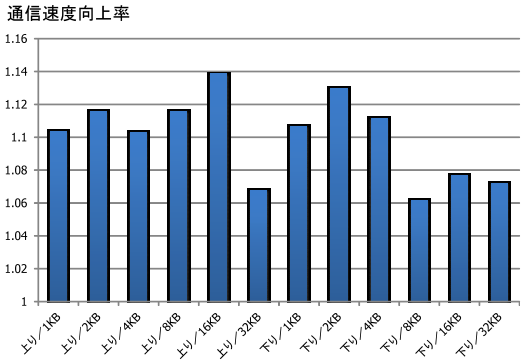


図 9 イーサネットの性能評価における提案手法によるデータ通信速度向上率
**Fig. 9** Increase of data communication speed using proposed technique in performance evaluation of Ethernet.

また，nuttcp は TCP の Read/Write バッファサイズを指定できる．そのサイズを変化させて通信速度の変化を計測した．

USB UART は**図 8** で示す接続を行い通信させた．通信性能を計測するために FPGA ボード側の Linux のシリアルコンソール上で文字列を送信するベンチマークプログラムを実行させ，その実行時間を計測した．ベンチマークプログラムは，1 から順に決められた数までの数字をカウントアップし改行コードを付与した文字列として，ホスト側のコンソールに送信する．評価では 1 から 10 までの文字列の送信，1 から 100 までの文字列の送信，1 から 1,000 までの文字列の送信，1 から 10,000 までの文字列の送信と 4 つケースで速度を計測した．本計測もイーサネットと同様に 100 回計測し，送信が終了する時間の平均をとって評価した．

## 3.2 性能評価

### 3.2.1 イーサネットの性能評価

**図 9** に提案手法のイーサネットにおける I/O 性能の向上を示す．図で示されるグラフは図 7 で示した計測環境で nuttcp を実行し，計測したデータを元に作成している．図の縦軸は VMM 実装手法の通信速度を 1 として，提案 HW 実装手法により同じ処理を行った場合の通信速度の向上比を示している．たとえば，縦軸で 1.1 を示していれば，ハー

表 2 イーサネットの性能評価における VMM 実装手法の命令実行数のカウント
**Table 2** Executed instruction count for VMM implementation technique in performance evaluation of Ethernet.

| アドレス変換部の命令実行数 | トータル命令実行数 | 割合 (%) |
|---|---|---|
| 9,801,763 | 121,946,838 | 8.03 |

ドウェアにより I/O アドレスマップ変換を行ったことで，通信速度が 10%向上していることになる．図の横軸は計測の条件を示しており，スラッシュで区切られた左側は計測が上りで行われたか，下りで行われたかを示している．また，スラッシュで区切られた右側はその計測を行うときの Read/Write バッファサイズを示している．ここで示す上りはクライアント側からサーバ側への転送，下りはサーバ側からクライアント側への転送を意味する．

本研究の計測では，VMM が I/O アドレスマップ変換に要する命令実行を除いて，すべて同じ命令実行である．そのため，観測されたハードウェア I/O アドレスマップ変換による性能向上は VMM が I/O アドレスマップ変換のために実行した命令実行オーバヘッドを削減したために得ている．

計測結果から，最大の性能向上は，上りのバッファサイズ 16 K バイトで，約 14%の性能向上が観測された．すべての計測で性能低下は観測されず，最小の性能向上は，上りのバッファサイズ 32 K バイトで約 6%であった．全計測条件の性能向上平均は約 10.03%であった．

この性能向上が実行命令数減少によるものをか裏づけるために，OR1200 プロセッサの回路中にカウンタを挿入し，VMM 実装手法の命令実行数を計測した．カウンタの観測には表 1 で示したロジック・アナライザを使用した．計測条件は nuttcp の上りの計測でバッファサイズ 1 K バイトである．**表 2** に nuttcp を 5 回実行した命令実行数の平均を示す．

この計測では，トータル命令実行におけるアドレス変換部の命令実行数の割合は 8.03%であった．同一計測条件の通信速度の性能向上が 10.36%であったことから，この実行命令数の増加分の実行の必要がなくなる提案手法においては，妥当な性能向上が得られていると考えて良い．OR1200 プロセッサは全命令を 1 クロックサイクルで実行できるわけではなく，特にデータ転送命令は複数クロックサイクルを要する可能性が高い．加えて，nuttcp はネットワークステータスに応じ sleep システムコールを呼ぶ構造となっており，アプリケーションのコード部以外の命令の実行がトータル命令実行数に加算されることを考慮すると，提案手法の命令実行オーバヘッド削減効果は狙い通りに実現されていると結論づけることができる．

### 3.2.2 USB UART の性能評価

**表 3** に USB UART におけるベンチマークの計測結果

表 3 USB UART の送信時間計測結果
**Table 3** Evaluation result for transmission time by USB UART.

| | 1–10 | 1–100 | 1–1000 | 1–10000 |
|---|---|---|---|---|
| VMM 実装手法（秒） | 0.0175 | 0.177 | 1.9502 | 18.066 |
| 提案 HW 手法（秒） | 0.0168 | 0.1482 | 1.5102 | 15.7452 |

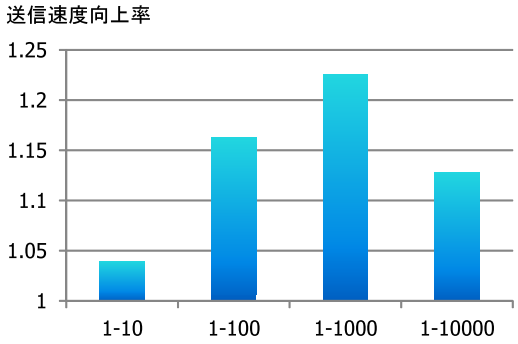


図 10 USB UART の性能評価における提案手法によるデータ送信速度向上率
**Fig. 10** Increase of data communication speed using proposed technique in performance evaluation of USB UART.

を，**図 10** に表 3 から算出した提案手法の I/O 性能の向上を示す．表 3 で示される数値は図 8 で示した計測環境で USB UART 経由の Linux シリアルコンソール上に文字列を送信し，送信し終わるまでの時間を計測した結果である．表中の 4 つの計測項目は実験時の条件を示している．たとえば，1–10 は 1 から 10 まで数をカウントアップし，その数字を示す文字列を順に送信した場合，1–100 は 1 から 100 まで数をカウントアップし，その数字を示す文字列を順に送信した場合である．図 10 の縦軸は VMM 実装手法の通信速度を 1 として，提案 HW 実装手法により同じ処理を行った場合の通信速度の向上比を示している．たとえば，縦軸で 1.1 を示していれば，ハードウェアにより I/O アドレスマップ変換を行ったことで，送信速度が 10%向上していることになる．図の横軸は表 3 で示した計測条件である．計測結果から，1–1000 の文字列送信の計測条件で，最大の性能向上である約 22.5%の送信性能向上を観測した．すべての計測で性能低下は観測されず，最小の性能向上は，1–10 の文字列送信の計測条件で約 4%であった．全計測条件の性能向上平均は約 13.91%であった．

USB UART における計測結果は図 9 で示したイーサネットにおける計測結果に比べ，改善率が大きい．この差異は各ベンチマークにおける I/O 操作処理とそれ以外の処理の実行命令の差異によるものである．USB UART におけるシリアルコンソールの通信はイーサネットに比べ，通信制御が単純であり，I/O 操作に係るプログラムコードの実行比率が高い．この差異が性能向上率の差になったと考えられる．

表 4 ハードウェア量の比較
Table 4 Comparison of hardware size.

| | VMM 実装手法 | 提案 HW 実装手法 |
|---|---|---|
| スライス数 | 6,767 | 6,750 |
| LUT 数 | 12,625 | 12,564 |
| RAM16BWER 数 | 41 | 41 |
| RAM8BWER 数 | 3 | 3 |
| DSP481A1 数 | 4 | 4 |

表 5 定数記述と BRAM 実装の 32 エントリの場合のハードウェア量および遅延
Table 5 Hardware size and latency of constant description method and BRAM implementation method when 32 entries are configured.

| | 定数実装 | BRAM 実装 |
|---|---|---|
| レジスタ数 | 0 | 31 |
| LUT 数 | 13 | 1,141 |
| 占有スライス数 | 8 | 667 |
| BRAM 数 | 0 | 43 |
| 動作周波数 | 412.2 | 101.82 |
| 遅延クロック数 | 1 | 1 |

### 3.3 ハードウェア量および遅延評価

本研究は VMM 処理の一部を専用の SoC の機能部品としてハードウェア化し，処理の効率化を図る．このため，提案手法による実行速度低下はなく，I/O ポートの参照頻度に応じて仮想化処理を高速化することができる．しかしながら，ハードウェアに専用の部品を導入することから，ソフトウェア処理と比べて，ハードウェア量と回路遅延の増大が懸念される．本節では，本研究で設計したハードウェアをハードウェア量と遅延の観点から確認し，トレードオフを議論する．

提案ハードウェアを含むチップ全体のハードウェア量を評価するために，表 1 で示した配置配線ツールの必要リソース数レポートを**表 4** に示す．表中に示される VMM 実装手法はソフトウェアで I/O アドレスマップ変換を行うため，性能計測用のカウンタなどを除き，OR1200 のメモリ参照の論理に何も手を加えていないハードウェアである．表中に示される提案 HW 実装手法は図 6 で示したハードウェアを導入し，ハードウェアによる I/O アドレスマップ変換機能を備えたハードウェアである．両手法に若干の差異が見られる．スライス数で 0.2%程度，LUT 数で 0.4%程度，VMM 実装手法の方がハードウェア量が多い．当初，定数値の比較回路と定数値の選択器を必要とする提案 HW 実装手法の方が多少のハードウェア量増加必要とすると推測したが，結果はそうならなかった．このきわめて微量の提案手法のハードウェア量削減は論理合成と配置配線ツールの最適化による影響であると考えられる．

**表 5** に図 6 のハードウェアのみを 32 エントリ構成にしてメモリ部（白の矩形部）を定数で実装した場合と，ブロック RAM（BRAM）で実装した場合のハードウェア量と遅延を示す．定数実装は非常に高速で，きわめて小さい回路で実現できていることが分かる．この結果から，表 4 で示した，提案ハードウェアのハードウェア量と遅延がチップ全体にほとんど影響しないものであることが裏づけられる．

表 6 最大動作周波数の比較
Table 6 Comparison of maximum operating frequency.

| | VMM 実装手法 | 提案 HW 実装手法 |
|---|---|---|
| 最大動作周波数（MHz） | 42.18 | 41.051 |

FPGA はランダムロジックを表現するために効率の良いメモリのアレイであり，動的部分再構成機能を用いて ROM をメモリとして利用可能であることから，FPGA で実装する SoC を対象にした場合，提案ハードウェアは定数記述で実装すべきであることが分かる．一方，チップのロジックを LSI で実装する場合は定数実装を行うことはできない．この場合は FPGA の BRAM に相当する SRAM セルを用いることになるが，32 エントリの 21 ビットタグを実現する連想メモリは一般的な TLB のエントリサイズ・タグとほぼ同等のメモリ量であり，この TLB は小さく実装でき，かつ高速に参照できることが，近年の CPU の設計から明らかになっていることから，この場合でも同様に実現は可能であると推測できる．

表 4 と表 5 の結果から，ハードウェア量に関して，少なくとも FPGA 上の実装に関してだが，I/O アドレスマップ変換回路は支配的な大きさとはならないことが明らかとなった．

提案ハードウェアを含むチップ全体の遅延を評価するために，表 1 で示した配置配線ツールの動作周波数レポートを**表 6** に示す．両者の動作周波数の差は 1.129 MHz で，VMM 実装手法のハードウェアの若作周波数が若干，提案 HW 実装手法を上回った．この差を知るために，両ハードウェアのクリティカルパスを調査した．共に 32 ビット乗算回路でクリティカルパスが形成されており，提案ハードウェアはクリティカルパスの要因になっていないことが確認できた．この差異はやはり，論理合成と配置配線ツールの最適化の影響であると考察する．

## 4. 関連研究

1.1 節で紹介したように，VMM は組込みシステム開発に様々な利点を持つ．このことから，2000 年代後半から現在にかけて，多様な組込み VMM が様々なベンダーから提供されてきた．広義の意味では組込み VMM は組込みシステム内に準備されたベアメタルハイパバイザを意味し，狭義の意味ではハイパバイザと L4 マイクロカーネル [4] を組み合わせたシステムを意味する．

L4 マイクロカーネルと組み合わせた代表的な組込み VMM には，PikeOS [5]，OKL4 [6]，CODEZERO [8]，

NOVA [7] などがあげられる．PikeOS は信頼性に特化した組込み VMM であり，エアバス A350 やエアバス A400M などの航空機で採用された実績がある．OKL4 は組込み VMM の先駆的存在であり，スマートフォンなどの Android 端末に採用されたことから，世界中で 10 億を超える搭載機器が存在するとされている．CODEZERO は ARM 向けプラットフォームで仮想ネットワーク内の通信の最適化に着目した VMM で，近年注目を浴びてきている．NOVA はセキュリティに重点を置いた組込み VMM で，複数の VMM を同時に動作させ，完全仮想化し，論理的にゲスト OS を分割することで，高いセキュリティ性能を実現している．また，L4 マイクロカーネルを用いない組込み VMM も存在する．蛍 [9]，WindRiver Hypervisor [10]，RTS Hypervisor [11]，VMWare MVP [12] などがそれにあたる．

これら組込み VMM は，本研究が対象とする SoC でカスタマイズされたハードウェアは対象とせず，ARM，MIPS などの標準的な組込み用マイクロアーキテクチャを採用したプロセッサを対象としている．組込み向けプロセッサは，Intel VT-d のようなの仮想化アクセラレータが搭載されないため，I/O アドレス変換をソフトウェアにより行っているため，本研究が対象とする問題を解決することはできない．組込み仮想化の機能を SoC の構成選択時の機能の 1 つとしてハードウェアで実現するアプローチの研究およびプロダクトは他に存在しないことから，本研究のアプローチは新規性を持つと考える．

本研究に近い着眼点の先行研究として Domain Partitioning [20] がある．Domain Partitioning はマルチコア環境において，PPC（Physical Partitioning Controller）と呼ばれるアクセス分割器が内部システムバス上のターゲットモジュールへの信頼性をチェックする機構を提案している．PPC はアドレスと SrcID と呼ばれるイニシエータモジュールと制御レジスタアドレスを対とした静的な ID を用いて，デバイスへの参照を制御する．Domain Partitioning と本研究の大きな違いとして，Logical Partitioning（仮想化）への適用があげられる．Domain Partitioning は Physical Partitioning と呼ばれる CPU コアや I/O ペリフェラルなどの SoC の機能を用途ごとに静的に分割実装する手法に着目し，その実装方式における効率的な資源割り当ての提案を行っている．一方，本研究は仮想化に着目し，マルチコア環境でなくとも，単一のハードウェア資源に複数のオペレーティングシステム環境が構築でき，資源の分割・共有を行うフレキシブルなシステム上での I/O 性能向上を実現している．先行研究では仮想化による目的の実現はソフトウェアオーバヘッドが大きいため，より現実的な Physical Partitioning を対象としたが，本研究はそのソフトウェアオーバヘッドの削減を対象としていることが，本研究の位置づけである．

## 5. おわりに

通常，仮想化ソフトウェアが行う I/O アドレスマップ変換処理を，専用ハードウェアを用意して処理することで，本研究は仮想化ソフトウェアの命令実行オーバヘッドを削減する提案を行った．このハードウェアを導入することで，ソフトウェアポータビリティの向上による組込みのシステム開発効率向上と動作速度低下抑制を両立できる．また，提案ハードウェアを SoC 用の IP コアとして位置づけることで，組込みシステム向けの SoC を設計する歳の機能選択の選択肢を与えることができる．

本研究では，高速な I/O 処理が必要とされるイーサネットと汎用性の高い USB UART に着目し，提案手法による性能向上を実機で計測した．イーサネットにおける評価で，最大で約 14%の通信速度向上を実現し，全計測条件の平均で約 10%の通信速度向上を実現した．USB UART における評価で最大で約 22.5%の通信速度向上を実現し，全計測条件の平均で約 13.91%の通信速度向上を実現した．また，実際に設計した回路を検証し，提案ハードウェアは FPGA 実装において軽微なものであり，ハードウェア量および遅延の観点から容易に導入できることを示した．

**謝辞** 本研究は平成 25 年度公益財団法人澁谷学術文化スポーツ振興財団「大学の新技術研究に対する奨励金」の助成により行われた（研究課題名：ハードウェアで実現する組込みシステム向けハイパーバイザの開発）．

### 参考文献

[1] Bitner, B. and Greenlee, S.: z/VM - A Brief Review of Its 40 Year History, IBM Corporation (2012), available from ⟨http://www.vm.ibm.com/vm40hist.pdf⟩.
[2] Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I. and Warfield, A.: Live migration of virtual machines, *Proc. 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation* (2005).
[3] Nelson, M., Lim, B.-H. and Hutchins, G.: Fast Transparent Migration for Virtual Machines, *Proc. Annual Conference on USENIX Annual Techinical Conference*, pp.391–394 (2005).
[4] Liedtke, J.: On $\mu$-kernel Construction, *Proc. 15th ACM Symposium on Operating System Principles*, pp.237–250 (1995).
[5] Kaiser, R. and Wagner, S.: Evolution of the PikeOS Microkernel, *Proc. 1st International Workshop on Microkernels for Embedded Systems* (2007).
[6] Heiser, G. and Leslie, B.: The OKL4 Microvisor: Convergence Point of Microkernels and Hypervisors, *Proc. 1st ACM Asia-Pacific Workshop on Systems*, pp.19–24 (2010).
[7] Steinberg, U. and Kauer, B.: NOVA: A Microhypervisor-based Secure Virtualization Architecture, *Proc. 5th European Conference on Computer Systems*, pp.209–222 (2010).
[8] CODEZERO® Embedded Hypervisor: B LABS, available from ⟨http://b-labs.com/products/⟩.

[9] ハイパバイザ 蛍：AXE, Inc., 入手先 ⟨http://www.axe-inc.co.jp/hotaru/index.html⟩.
[10] Wind River Hypervisor: Wind River Systems, Inc., available from ⟨http://www.windriver.com/products/product-notes/PN_Hypervisor_0611.pdf⟩.
[11] RTS Real-Time Embedded Hypervisor: Real-Time Systems GmbH, available from ⟨http://www.real-time-systems.com/real-time_hypervisor/index.php⟩.
[12] VMware and Trango: VMware, Inc., available from ⟨http://www.vmware.com/company/acquisitions/trango/⟩.
[13] Enabling Intel® Virtualization Technology and Benefits: Intel White Paper (2010), available from ⟨http://www.intel.in/content/dam/www/public/us/en/documents/white-papers/virtualization-enabling-intel-virtualization-technology-features-and-benefits-paper.pdf⟩.
[14] Intel® Virtualization Technology and Intel® Active management Technology in Retail Infrastructure: Intel White Paper (2006), available from ⟨http://research.cs.wisc.edu/areas/os/ReadingGroup/OS/papers/vanderpool_ia32.pdf⟩.
[15] Atlys™ Board Reference Manual: Digilent, Inc., available from ⟨http://www.digilentinc.com/Data/Products/ATLYS/Atlys_rm.pdf⟩.
[16] OpenRISC Architecture Manual: OPENCORES.ORG (2003), available from ⟨http://www.da.isy.liu.se/courses/tsea44/OpenRISC/openrisc_arch3.pdf⟩.
[17] OR1200 OpenRISC Processor: OPENCORES.ORG, available from ⟨http://opencores.org/or1k/OR1200_OpenRISC_Processor⟩.
[18] ORPSoc: OPENCORES.ORG, available from ⟨http://opencores.org/or1k/ORPSoC⟩.
[19] nuttcp: nuttcp development team, available from ⟨http://nuttcp.org/nuttcp/Welcome%20Page.html⟩.
[20] Nojiri, T., Kondo, Y., Irie, N., Ito, M., Sasaki, H. and Maejima, H.: Domain Partitioning Technology for Embedded Multicore Processor, *IEEE Micro*, Vol.29, No.6 (Nov./Dec. 2009).
[21] Huynh, K. and Theurer, A.: KVM Virtualized I/O Performance, Report of IBM Linux Technology Center (June 2013), available from ⟨http://www.novell.com/docrep/2013/05/kvm_virtualized_io_performance.pdf⟩.
[22] Intel Virtualization Technology for Directed I/O Architecture Specification Rev.2.2: Intel Corporation (Sep. 2013), available from ⟨http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/vt-directed-io-spec.pdf⟩.

**請園 智玲** （正会員）

昭和 53 年生．平成 21 年北陸先端科学技術大学院大学情報科学研究科博士後期課程情報システム学専攻修了．博士（情報科学）．平成 22 年より北陸先端科学技術大学院大学高信頼組込みシステム教育研究センター研究員．平成 23 年より北陸先端科学技術大学院大学情報科学研究科助教．プロセッサアーキテクチャ，組込みシステムの研究に従事．電子情報通信学会会員．