組込み仮想化におけるハードウェアI/Oアドレスマップ変換の性能評価

請園　智玲†　　荒木　光一††

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．組込み仮想化は既存のソフトウェアを新しいプラットフォームに移植する際に，ソフトウェアの修正を最小限にし，ソフトウェアの再利用性を高めることができる．この恩恵から，組込み製品の生産性の向上が可能となり，ソフトウェアの修正に伴う不具合を抑えられる．また，移植するVMにCPUの特権モードを与えないことで，堅牢性が高く，セキュアなシステムの構築が可能となる．その一方で，仮想化に用いるVMMは命令エミュレーションによるオーバーヘッドが原因でCPUの演算効率を低下させる．このため，実時間処理が重要な組込みシステムへの仮想化の導入を妨げている．

我々の目的は汎用性を必要としない組込みの分野で，VMMをSoCの機能としてハードウェアで実装し，VMMの恩恵を，より高いCPUの演算効率で実現する組込み仮想化システムを構築することである．本稿では，特に組込み仮想化のI/Oアドレスマップ変換部に着目し，ハードウェアI/Oアドレスパップ変換器を提案する．また，ハードウェアI/Oアドレスパップ変換器によって得られる演算性能向上を評価する．

Performance Evaluation for Hardware Translation of I/O Address Map in Embedded Virtualization

Tomoaki Ukezono† and Koichi Araki††

Recently, exploiting virtualization to design of embedded system is getting a lot more attention lately. Embedded virtualizations can achieve minimal modification for software and higher software reusability when the software is ported to new platforms. From the benefits, improvement in productivity and bugs that is caused by software modification can be attained. Furthermore, robust and secure system can also be established preventing that CPUs give privileged mode to migrated VM. On the other hand, VMM which is used to virtualization decreases effectiveness of execution in CPUs, since the VMM has overhead for emulation of instructions. For this reason, introducing virtualization to embedded systems that are attached weight to real time execution is prevented.

Our objective is to establish an embedded virtualization system that can be benefited from VMMs and can be achieved more effective CPU execution implementing VMMs in hardware as a function of SoC. In this paper, in particular, we focus on the translation of I/O address map in embedded virtualization and propose a hardware I/O address map translator. In addition, we evaluate performance improvement by the hardware I/O address map translator.

1. はじめに

1.1 汎用システムの仮想化

仮想化は1967年のIBM社のメインフレームSystem360モデル40用に開発されたCP40[1)]が起源とされる．当時のメインフレームは大変高価であり，比較的小規模な演算を必要とするユーザには不向きであった．IBMはこれらユーザに対応するために，高価なハードウェアを複数のユーザで共有する仮想化システムを導入した．時を経て，現在，ハードウェアはプロセス技術の発展により，性能あたりの価格が激的に安くになったが，仮想化技術は依然として利用されている．特にデータセンターなどで大規模なハードウェアリソースを従量制で貸し出すクラウドシステムではSystem360の時代から発展した仮想化ソフトウェアがシステムの中核をなす技術となっている．

仮想化ソフトウェアはユーザが使用するOSをゲストOS(VM:Virtual Machine)と位置づけ，単一のハードウェア上に復数のゲストOSを同時に動作させる．各VMはハードウェアリソースの全てを自らが使える

† 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology (JAIST)
†† 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
Center for Highly Dependable Embedded Systems Technology, JAIST

と思い込み振る舞うが，仮想化ソフトウェアはさらにローレベルの制御を行い，複数の OS がハードウェアリソースを共有するように制御を行う．現在，汎用システム向けの仮想化ソフトウェア（ハイパバイザ）が多くのベンダーから多様な製品として提供されている．ハイパバイザの種類には大別して 2 種類ある． 1 つは Xen[2] や KVM[3] に代表されるベアメタルハイパバイザ (type1 ハイパバイザ) である．ベアメタルハイパバイザはハードウェア上で直接動作するソフトウェアである．2 つ目は Virtual BOX[4] や VMWare Workstation[5] に代表されるホスト型ハイパバイザ (type2 ハイパバイザ) である．ホスト型ハイパバイザは OS 上のアプリケーションとして実現される．以降，本稿ではベアメタルハイパバイザに関しての議論を行う．また，本稿ではハイパバイザを VMM(Virtual Machine Monitor) と表記する．

1.2 組込み仮想化

近年，組込みシステムの開発でも汎用システムと同様に VMM の利用が注目を浴びている．組込み VMM は，汎用とは異なり，VM 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が目的とされる．組込み仮想化がもたらす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組込みソフトウェアを移植する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なされず，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，ソフトウェア移植後のハードウェア上で動作する 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伴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[6][7] 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．そういったシステムでも，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 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もつ．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える．

この様に，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[8] を組み合わせたシステムを意味する．

マイクロカーネルと組合わせた代表的な組込み VMM には，PikeOS[9]，OKL4[10]，CODEZERO[12]，NOVA[11] などが挙げられる．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，スマートフォン等の 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，マイクロカーネルを用いない組込み VMM も存在する．蛍[13]，WindRiver Hypervisor[14]，RTS Hypervisor[15]，VMWare MVP[16] などがそれにあたる．

VMM は組込みシステム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．我々の最終目標は VMM を System-on-Chip(SoC) の機能の一つとしてハードウェアで実装し，この命令実

行オーバヘッドを削減することである．

SoCとは，組込みシステムに必要な機能を複数のIC／LSIを用いてボード上に実装せずに，一つのLSIの中に実装する設計方法である．これにより，組込み製品の小型化や製造コスト低減が実現できる．本研究はSoCの一つの機能としてハードウェア化したVMMの実装を提案する．汎用システムのように多機能なVMMをハードウェア実装することは容易ではない．しかしながら，我々は，組込みシステムにおいては，設計時に動作するVMの種類と数が静的に決まる特性や，メモリやCPU資源の種類と量が少ない特性をもつため，ハードウェア化の実現が可能であると考察している．本研究の最終的な成果物はSoCを構成する際にRTLまたはネットリストによるソフトマクロとしてIPコアの形態で利用され，LSIとして実現する．これまでにSoCのIPコアでVMMが提供される例はなく，完全にハードウェア化された事例もない．

本研究が目指す仮想化は完全仮想化である．完全仮想化の場合，VMはプログラムバイナリの修正なしに，直接ハードウェア上で実行されていると思い込み動作する．これはVMMがソフトウェア実装されていてもハードウェア実装されていても変わらない．このため，VMMのハードウェア化は組込み仮想化の最大の利点であるソフトウェアのポータビリティの向上に影響を与えない．

本稿では，ハードウェア化するVMMの機能の一つとして，I/Oアドレスマップ変換に着目し，そのハードウェア化で得られる実行性能向上を評価し議論する．多くの命令セットアーキテクチャでは，I/Oへの参照をメモリマップドI/Oで実現している．外部デバイスが割り当てられるI/Oアドレスマップはハードウェアによって異なり，特にアプリケーションレベルでこのアドレスを操作する組込みシステムでは，新しいプラットフォームへのソフトウェアの移植に多大な修正を要する．VMMはこの修正を抑えるために，I/Oアドレスマップの変換を動的に行う．また，VMMはI/Oアドレスマップ変換と同時に，複数VMのI/O参照の競合を調停する動作を行なっており，I/O制御を主眼とする組込みシステムの仮想化では，動作速度が非常に重要となる．

ハードウェアによるアドレス変換はTLBに代表されるようにハードウェアで容易に実現可能である上，その効果が大きい．このため，我々はVMMを完全にハードウェア化するための最初のコンポーネントをI/Oアドレスマップ変換に決定した．

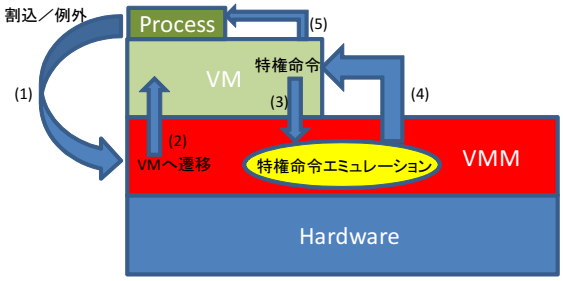

図 1 VMとVMMの階層構造と実行遷移.

2. 組込み仮想化のI/Oアドレスマップ変換

2.1 ベアメタルハイパバイザの動作概要

ベアメタルハイパバイザはハードウェア上に直接VMMがあり，OSの特定動作によりVMMのプログラムが実行される．図1に，この階層構造とその実行遷移を示す． 図1はIntel VT-x[17]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でのフルソフトウェア実装の例である．VMMが一つ以上のVMを管理し，VMは一つ以上のユーザプロセスを管理する形となっている．図中の番号に従いVMとVMMの動作遷移を説明する．

(1) ユーザプロセスが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．☆
(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，特権が必要な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．☆☆
(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からOSの各機能を実装したプログラムに遷移する．VMMは

☆ 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
☆☆ 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

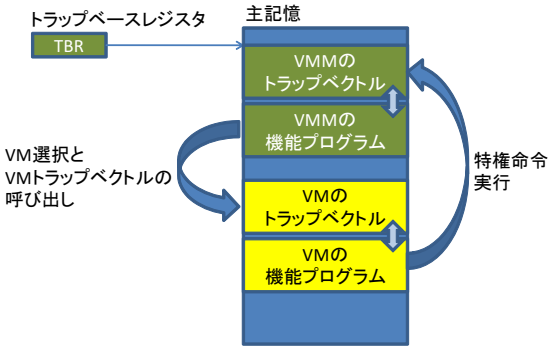

図 2 VM と VMM のトラップベクトルの構造の概要.

このトラップベクトルを乗っ取る形で実装される．図 2 に VMM のトラップベクトルを配置を示す．

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 I/O アドレスマップ変換の処理概要

VMM による I/O アドレスマップ変換の処理はターゲットとするマイクロアーキテクチャにより異なるが，通常，I/O ポートはバイト（またはワード）単位でアドレスが割り振られるため，VMM が TLB 等の既存のハードウェアを用いてページサイズ毎の I/O アドレスの変換を行った場合，変換粒度が粗すぎる問題が発生する．粒度が粗すぎる場合，1 つの物理ページが複数の I/O ポートを包含し，参照を許可していない I/O ポートを VM が参照可能になり，VMM の厳密な I/O 資源管理が困難になる．このことから，VM の I/O ポートへの参照は，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妥当である．VMM が命令単位の制御を行えば，厳密にどの VM がどの I/O ポートを参照するかを制御できる．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を図 3 に示す．

図 3 で示す処理は図 2 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，予め MMU を制御する．VMM が全ての 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容易である．本稿の評価では，データ MMU Fault

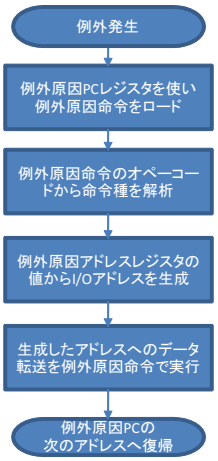

図 3 ソフトウェア I/O アドレスマップ変換処理のフローチャート.

例外を I/O アドレスマップ変換処理の起点にした．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．本稿の評価のソフトウェア I/O アドレスマップ変換処理はこの 2 つのレジスタを利用した．まず，例外原因 PC レジスタからアドレスを取得し，命令をロードする．次にオペコードを元に命令を解析し，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．これら動作を本稿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既に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．

上記の処理は，I/O アドレスマップ内を参照する全てのデータ転送命令で発生する．このため，I/O 参照が支配的な処理を行う場合は相応のオーバヘッドが必要となる．本稿では，このオーバヘッドを縮小するために，I/O アドレスマップの変換を行うハードウェアを提案し，その性能を評価する．

3. ハードウェア I/O アドレスマップ変換

本稿で提案する I/O アドレスマップ変換ハードウェアは MMU のさらに主記憶側に配置され，物理－物理アドレス変換を行う．提案する I/O アドレスマップ

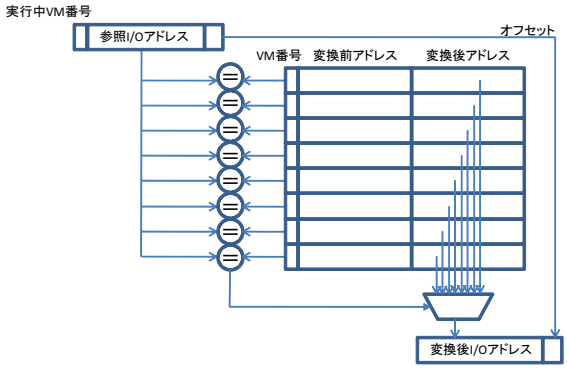

図 4 I/O アドレスマップ変換ハードウェア.

変換ハードウェアを図 4 に示す.

提案ハードウェアの構造は TLB と酷似している. 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テーブル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当該変換情報が有効な VM を識別する ID が格納されており，この VM の ID はハードウェア VMM がもつ，現在実行中の VM の ID レジスタと比較される. 複数の VM が同時に実行される場合，例えば，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.

テーブルのエントリ数は VM が参照可能な I/O ポート数で決まる. ここで重要なポイントは，本研究の目的が汎用システム向けではなく，組込み VMM を SoC の機能としてハードウェア実装する提案をしていることである. SoC を開発する際に，VM 移植前のプラットフォームの I/O アドレスの情報が分かれば，このテーブル内の情報を定数値で設計可能である. TLB のように，このようなテーブルを SRAM を用いて設計すると，実装されるテーブルのエントリ数に応じて回路面積と遅延が問題となるが，定数値比較回路と定数値選択器であれば，回路規模は比較的小さく設計できる. また，SoC 設計時に各 VM に割り当てた I/O ポートはソフトウェア実行などの要因で変えることができないため，セキュアなシステム設計にもつながる. 本稿の評価では，この定数値の I/O アドレス変換回路を実装し評価した.

SoC 設計時に移植される VM が決定していない場合，このテーブルは SRAM などで設計されなければならない. その場合，エントリ数を多く設定することは，回路面積及び遅延の観点から難しい. このため，少量のエントリを用意して，現在アクティブな I/O のアドレスマップ変換情報のみを格納し，システムの状態に合わせて情報を置換する必要がある. この SRAM を用いたテーブル実装は本稿では未評価であり，今後，実装・評価を進める予定である.

本稿の評価では，イーサネットの通信速度を対象に評価を行ってる. 例えば，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. 本稿の実験環境では，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. 今後，適切なエントリ粒度に関しての研究を進めていく.

4. 評　　価

4.1 評価環境

評価環境として Digilent 社製 Atlys^{TM} Spartan-6 FPGA 開発ボード[18] の FPGA 上に OpenRISC 1000[19] 命令セットを採用した OR1200 プロセッサ[20] と ORPSoC[21] を実装し，構築した. Spartan-6 FPGA 開発ボードには DDR2 SDRAM が 128MB, 1Gbit ether phy，HDMI 入出力，AC-97 audio，SPI Flash メモリが 16MB，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. 本稿の評価では，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. 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. イーサネット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. 本稿の評価で使用したハードウェア設計環境を表 1 に示す.

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. OpenCores プロジェクトによって GNU ツールチェイ

表 1 ハードウェア設計環境.

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

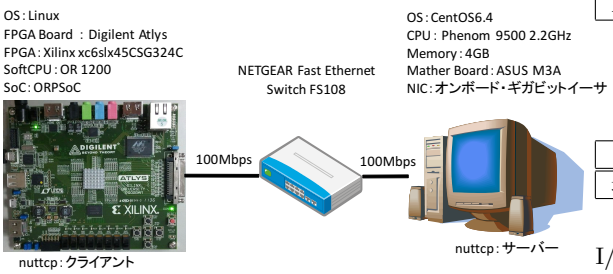

図 5 I/O 性能評価環境.

ンが移植されたため，Linux はカーネル 3.1 リリースから OpenRISC に正式対応し，OR1200 プロセッサ上で実行可能である．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 2 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全て 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 3 で示すフローチャートに沿って実装している．

イーサネットは図 5 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[22] 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 はクライアントー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．本稿の評価では，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．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．

4.2 ハードウェア量及び遅延の評価

提案手法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 2 に示す．表の VMM 実装手法はソフトウェアで

表 2 ハードウェア量の比較.

	VMM 実装手法	提案 HW 実装手法
スライス 数	6767	6750
LUT 数	12625	12564
RAM16BWER 数	41	41
RAM8BWER 数	3	3
DSP481A1 数	4	4

表 3 最大動作周波数の比較.

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

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等を除き，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．提案 HW 実装手法は 3 章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．この極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 2 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わかる．

提案手法のハードウェア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 3 に示す．両者の動作周波数の差は 1.129MHz 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，提案手法のハードウェアはクリティカルパスの要因になるほど遅延を発生させないことが確認できた．

この動作周波数の差は，回路設計の一般的な見地から見ると，論理合成／配置配線ツールの最適化による誤差の範疇である．表 2 の結果も併せて踏まえると，今回の提案ハードウェア以外の部分 (32 ビット乗算回路) でハードウェアサイズ増加と引き換えに動作周波数を増加させる最適化が VMM 実装手法のハードウェアに若干多く適用されたと推測できる．

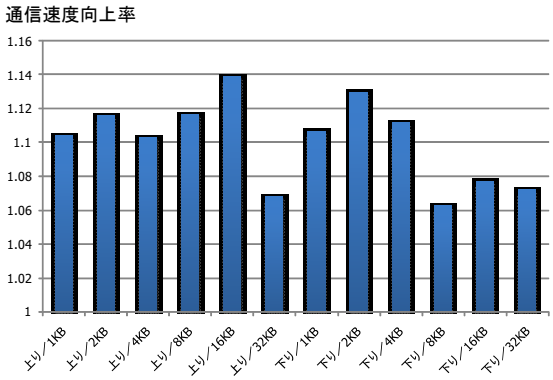

図 6 提案手法によるデータ通信速度向上率.

4.3 I/O 性能の評価

図 6 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 4.1 章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．例えば，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，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．

本稿計測では，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，全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 16KB で，約 14%の性能向上が観測された．全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 32KB で 6%であった．全計測条件の性能向上平均は 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付けるために，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 1K バイトである．表 4 に nuttcp を 5 回実行した命令実行数の平均を示す．

表 4 VMM 実装手法の命令実行数のカウント.

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

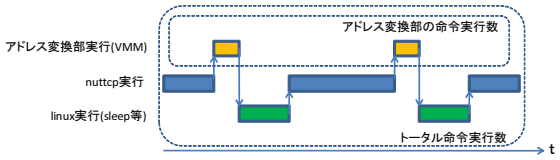

図 7 命令数計測の概要.

図 7 に表 4 で示すアドレス変換部の実行命令数とトータル命令実行数の計測の概要を示す．横軸は時間 (t:time) を示しており，横軸に沿って，その時に実行しているプログラムの種類が矩形で示される．プログラムの種類は 3 つあり，上からアドレス変換部の実行，nuttcp の実行，linux の実行である．実機での計測では，外部割り込みやタイマ割り込みなど，計測中に多様なイベントが発生するため，必ずしも図 7 の遷移をするわけではないが，表 4 で示す命令数が実行全体のどの部分を計測したのかを示している．アドレス変換部の実行命令数は VMM が I/O アドレスを変換する部分のみのプログラムの実行数で，トータル命令実行数は nuttcp の実行が開始されて exit システムコールを呼び出し，実行が完了するまでの命令実行数を示している．ただし，トータル命令実行数は nuttcp が Linux の sleep システムコールを呼び，OS が sleep 解除待ちの間に無限ループを実行した命令数も含まれるため，nuttcp のプログラムバイナリ内の命令実行数よりも多くカウントされている．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 1 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．また，sleep システムコールで加算されるトータル命令実行数も存在することを鑑みると，提案手法のオーバヘッド削減は狙い通りに実現されていると結論付けることができる．

5. おわりに

本研究の最終目的は VMM を SoC のひとつの機能

としてハードウェアで実装することである．これを達成することにより，仮想化のオーバヘッドを削減し，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リアルタイム性向上の両立を目指す．本稿では，ハードウェア VMM の I/O アドレスマップ変換機能に着目し，I/O アドレスマップ変換のハードウェア化の提案を行い，性能向上を評価した．

本稿では，高速な I/O 処理が必要とされるイーサネットに着目し，提案手法による通信速度向上を示した．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．

今後の課題として，3 章で提案したハードウェアの実装の検証が不足している点が挙げられる．本稿の評価の実装では，移植される VM がハードウェア設計時に決定される必要がある．しかしながら，実際の SoC 設計をこの制限で進めることは難しい．このことから，I/O アドレスマップの変換テーブルを SRAM で構成し，書き換え可能にするハードウェアを実装しなければならない．この実装を行うにはハードウェアサイズと遅延増加に注意を払い，適切なエントリ数の設定をした上で，テーブルエントリの置換手法を提案する必要がある．また，本稿の評価は 1 つのプログラムの実行に着目しているため，提案手法の汎用性を示せていないことから，更に多くのプログラムにおける性能を計測する必要がある．加えて，組込みシステム設計に重要な要素であるリアルタイム応答性を評価・検証する必要がある．今後はこれらの研究を進める．

本研究の最終目標である，VMM の完全なハードウェア化には，本稿の着目点である I/O の管理の他に，VM 実行の切り替えとスケジューリング，メモリ管理，外部割込み管理などの機能もハードウェア化しなければならない．これらのハードウェア化も今後併せて提案していくと共に，組込み VMM の利点であるポータビリティ／堅牢性／信頼性の評価基準を策定していく予定である．

謝辞 本研究は北陸先端科学技術大学院大学　井口研究室　井口　寧　教授　から FPGA 評価ボードの貸与協力を得て行われた．深く感謝する．

参 考 文 献

1) B. Bitner and S. Greenlee, “z/VM - A Brief review of Its 40 Year History”, http://www.vm.ibm.com/vm40hist.pdf, IBM Corporation, 2012.
2) B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne and J. N. Matthews, “Xen and the Art of Repeated Research”, Proc. of the annual conference on USENIX Annual Techinical Conference, pp.135-144, 2004.
3) A. Kivity, Y. Kamay and D. Laor, “KVM: The Linux Virtual Machine Monitor”, Proc. of the Linux Symposium, pp.225-230, 2007.
4) H. Davis, M. B. Skov, M. Stougaard and F. Vetere, “Virtual box: supporting mediated family intimacy through virtual and physical play”, Proc. of the 19th Australasian Conference on Computer-Human Interaction: Entertaining User Interfaces, pp. 151-159, 2007.
5) “Getting Started with VMware Workstation”, in http://www.vmware.com/pdf/desktop/ws90-getting-started.pdf, VMWare, Inc, 2012.
6) C. Clark, K. Fraser, S. Hand, J.G. Hansen,E. Jul, C. Limpach, I. Pratt, and A. Warfield, “Live migration of virtual machines”, Proc. of 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation, 2005.
7) M. Nelson, B.-H. Lim, and G.Hutchins, “Fast transparent migration for virtual machines”, Proc. of the annual conference on USENIX Annual Techinical Conference, pp.391-394, 2005.
8) J. Liedtke, “On µ-kernel construction”, Proc. of 15th ACM Symposium on Operating System Principles, pp.237-250, 1995.
9) R. Kaiser and S. Wagner, “Evolution of the PikeOS microkernel”, Proc. of first International Workshop on Microkernels for Embedded Systems, 2007.
10) G. Heiser, B. Leslie, “The OKL4 Microvisor: Convergence Point of Microkernels and Hypervisors”, Proc. of the first ACM Asia-Pacific Workshop on Systems, pp.19-24, 2010.
11) U. Steinberg, B. Kauer, “NOVA: a microhypervisor-based secure virtualization architecture”, Proc of the 5th European conference on Computer systems, pp.209-222, 2010.
12) “CODEZERO® Embedded Hypervisor”,B LABS, in http://b-labs.com/products/.
13) “ハイパバイザ 蛍”, AXE, Inc, in in http://www.axe-inc.co.jp/hotaru/index.html
14) “Wind River Hypervisor”, Wind River Systems, Inc, in http://www.windriver.com/products /product-notes/PN_Hypervisor_0611.pdf
15) “RTS Real-Time Embedded Hypervisor”, Real-Time Systems GmbH, in http://www.real-time-systems.com/real-time_hypervisor/index.php
16) “VMware and Trango”, VMware, Inc, in in http://www.vmware.com/company/acquisitions /trango/
17) “Intel® Virtualization Technology and Intel® Active management Technology in Retail Infrastructure”, Intel White Paper,

http://research.cs.wisc.
edu/areas/os/ReadingGroup/OS/papers
/vanderpool_ia32.pdf, 2006.
18) “Atlys™ Board Reference Manual”, Digilent, Inc, http://www.digilentinc.com/Data/Products/ATLYS/Atlys_rm.pdf
19) “OpenRISC Architecture Manual”, OPENCORES.ORG, http://www.da.isy.liu.se/courses/tsea44/OpenRISC/openrisc_arch3.pdf, 2003.
20) “OR1200 OpenRISC Processor”, OPENCORES.ORG, in http://opencores.org/or1k/OR1200_OpenRISC_Processor
21) “ORPSoc”, OPENCORES.ORG, in http://opencores.org/or1k/ORPSoC
22) “nuttcp”, nuttcp development team, in http://nuttcp.org/nuttcp/Welcome%20Page.html