社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－171 (7)
2007－EMB－ 3 (7)
2007／1／22

高機能システムLSIを対象とする仮想化ハードウェアのアーキテクチャ

吉井 謙一郎† 矢尾 浩† 金井 達徳†

デジタルメディア機器に組み込まれるシステムLSIは，プロセッサコアだけでなくハードウェアエンジンやDSPなど様々な機能モジュールを混載し，ハードウェアとソフトウェアが高機能化そして複雑化している．このようなシステムLSIでは，各機能モジュールにおける処理間の不要な相互干渉等を防ぐために，システムLSI全体を対象とした安全で信頼性の高い実行環境が望まれている．そこで本報告では，システムLSI全体を仮想化することによりそのような実行環境を実現するハードウェアIPのアーキテクチャを提案する．このハードウェアIPは，ハイパーバイザと呼ぶ特権ソフトウェアと連携して，メモリおよびデバイスアクセスの制御そしてデバイス割り込みの適切な配送により，システムLSI全体の仮想化を実現する．ハイパーバイザを実行するための領域は，専用のハードウェアによって保護する．このハードウェアIPは，既存のシステムLSIのプラットフォームを変更することなく追加することができる．

An Architecture of a lightweight IP for system LSI virtualization

Ken-ichiro Yoshii,† Hiroshi Yao† and Tatsunori Kanai†

Nowadays, system LSIs embedded in digital media systems become more complicated and sophisticated because various functional modules are integrated into them. To prevent interference among processes of function modules, safe, secure, and dependable runtime environment that covers whole system LSI is important. Therefore, we propose a new virtualization hardware IP architecture that realizes such runtime environment for software and hardware. In cooperated with hypervisor, hardware IPs realize system LSI virtualization by access control for memories and devices, and delivery of interrupts from devices to appropriate destination. An extra hardware protects runtime areas for hypervisor. These hardware IPs can be added to system LSIs without changing the platform of them.

1. はじめに

近年，組込み機器への要求の高度化に従い，ハードウェアとソフトウェア双方の複雑化が進んでいる．そのような組込み機器に使用されるシステムLSIでは，汎用プロセッサ以外にも，例えば映像処理用のハードウェアエンジンや，様々な方式の音声処理をソフトウェアで行うためのDigital Signal Processor (DSP)といった機能モジュールをいくつも混載するようになっている．これに加えて，汎用プロセッサ上でリアルタイム処理用のOSだけでなくGUI処理などを行う汎用OSを動作させたいというニーズも存在する．

しかし機能が高度化する反面，様々な機能をひとつのシステムLSIに集積することで機能間の相互干渉の危険性や，データおよびソフトウェアの保護や他機能の不具合等の影響からの隔離の必要性，さらにはシステム開発時のデバッグの困難化など，課題も多い．

今後も，消費電力を削減するために処理をハードウェアエンジンで実現してシステムLSIに搭載したり，逆にハードウェアエンジンで実現していた処理をソフトウェアで実現した場合でもマルチコアによる並列処理によって必要な性能を達成する流れが予想され，この課題の重要性は変わらないと考えられる．

そしてこれらの課題の解決には，ソフトウェアおよびハードウェアで実現される機能に必要なメモリやプロセッサ時間そしてデータ入出力に使うデバイスを，システムLSI全体に渡って管理することが重要である．

そこで本報告では，制御用のファームウェアと連携することでシステムLSI全体を仮想化し，安全で信頼性の高いハードウェアとソフトウェアの実行環境を実現する仮想化ハードウェアIPを提案する．

この仮想化ハードウェアIPは，システムLSIが搭

† (株) 東芝 研究開発センター
Corporate Research & Development Center, Toshiba Corporation

載するハードウェアエンジンを含めた機能モジュールを対象とする仮想化機能を，既存プラットフォームを変更することなく追加する，軽量でかつ親和性の高い設計となっている．

2. システム LSI の仮想化

高機能なシステム LSI において，ハードウェアおよびソフトウェアに対して安全で信頼性の高い実行環境を提供するには，プロセッサだけでなく，混載された機能モジュールの動作も考慮する必要がある．例えばメディアプレーヤーに使用されるシステム LSI では，ハードウェアエンジンや DSP による映像や音声信号の復号化や加工そしてデバイスへの出力処理，さらにはプロセッサ上で動作している再生ソフトウェアによる全体処理の制御が並行して行われている．

このとき，信頼性の観点から，再生ソフトウェアなど一部の機能の不具合でシステム全体が停止しないように，各処理に必要なデータやソフトウェア本体を他の処理から保護したり，機能モジュールが特定の処理に不当に占有されないように制御する必要がある．

またセキュリティの観点から，悪意のあるソフトウェアやそのソフトウェアに操作されたデバイスによる処理中のデータの漏洩や，本来出力が許されないデバイスへの出力による漏洩を防ぐ必要がある．

そこで我々は，システム LSI 全体を仮想化することにより，安全で信頼性の高いハードウェアとソフトウェアの実行環境を実現することを考えた．ここで，図 1 を使用してこの仮想化のイメージを説明する．

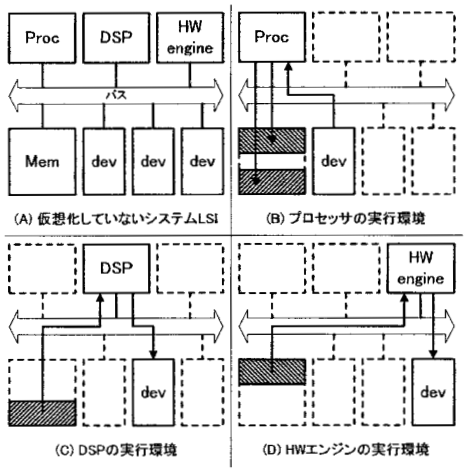

図 1 システム LSI 仮想化のイメージ
Fig. 1 Concept of system LSI virtualization

図 1 (A) は，プロセッサ，DSP，ハードウェアエンジン，メモリ，そして入出力デバイスを備えたシステム LSI のブロック図である．本仮想化技術は，図 1 (A) のプロセッサ，DSP，ハードウェアエンジンに対してそれぞれ図 1 (B)〜(D) のような実行環境を提供する．図中の実線はアクセスできる機能モジュールを，そして点線はアクセスできない機能モジュールを示す．

例えば，図 1 (D) はメモリからデータを読んで処理を行い結果をデバイスに出力するハードウェアエンジンの実行環境である．図の通り，ハードウェアエンジンは処理に必要なデータが格納されたメモリ領域と出力デバイスしかアクセスできない．また図 1 (B) と (C) の通り，プロセッサと DSP はハードウェアエンジンが処理するデータやデバイスへのアクセスを制限される．したがって，このハードウェアエンジンの処理は，他の機能モジュールの処理に不具合があってもその影響を受けず，また処理中のデータも保護される．このように，本仮想化ハードウェア IP は，プロセッサ上で動作するソフトウェアだけでなく，DSP やハードウェアエンジンといった全ての機能モジュールに対して仮想的な実行環境を提供するものである．

プロセッサを対象とした仮想計算機技術には，プロセッサにハードウェアによる仮想化支援機構[1)2)] を搭載する技術と，ソフトウェアによって仮想化を実現する技術[3)] が従来より存在する．これらの仮想計算機技術は，ともにプロセッサ上で動作するゲスト OS に対して仮想計算機を提供している．

本技術が目指すシステム LSI の仮想化は，従来の仮想計算機技術が実現しているプロセッサ上での複数 OS の動作を含め，図 1 で見たようにプロセッサ以外のハードウェアエンジンや DSP といった機能モジュールを仮想化の対象とする．また，システム LSI に組み込まれるプロセッサは仮想計算機の構築を支援するようなハードウェアを搭載していないものが多い．この 2 つの理由のため，システム LSI を仮想化するために従来の仮想計算機技術をそのまま適用することはできず，システム LSI ならではの工夫が必要となる．

3. 仮想化アーキテクチャの概要

前章で述べたシステム LSI の仮想化を実現するために，既存のプラットフォームに付加するハードウェア IP とそれを制御するファームウェアを設計した．これにより，システム LSI に搭載されるさまざまな組込みプロセッサと組み合わせて使用すること，そしてシステム LSI に搭載されたハードウェアエンジン等の機能モジュールを仮想化の対象とすることを実現している．本章では，この仮想化ハードウェア IP とその機能について説明する．

3.1 仮想化管理情報の保護

仮想化支援機構を備えないプロセッサを利用して図 1 (B)〜(D) のような仮想実行環境を実現する際の大きな課題は，メモリおよびデバイスへのアクセス制

御の設定情報等，仮想化のために重要な情報を，ゲスト OS 等のソフトウェアやハードウェアエンジンのような機能モジュールから保護することである．この保護が実現できないと，アクセス制御の設定情報を自由に操作され，他のゲスト OS やソフトウェアそして機能モジュールに割り当てられたメモリやデバイスが不正にアクセスされてしまう．

そこで，本仮想化ハードウェア IP を操作するファームウェアである**ハイパーバイザ**を導入し，このハイパーバイザが他のソフトウェアや機能モジュールに邪魔されずに動作することをハードウェアによって保証することで，そのハイパーバイザが動作している間のみアクセスできる領域を作成した．

このハイパーバイザが動作している状態を**HV モード**と呼び，それ以外を通常モードと呼ぶことにする．つまり，この仮想化ハードウェア IP が生成する保護領域にアクセスできるのは，HV モードで動作しているハイパーバイザのみである．

この HV モードを実現する IP を含む，本仮想化ハードウェア IP を付加したシステム LSI は以下の図 2 のようになる．

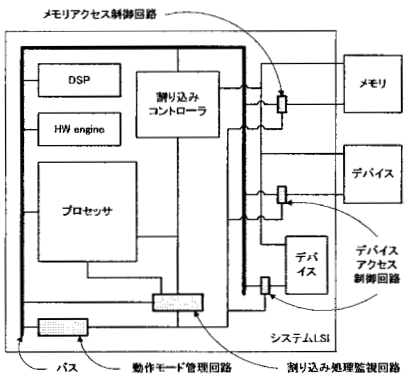

図 2 仮想化ハードウェア IP を付加したシステム LSI
Fig. 2 System LSI with hardware IPs for virtualization

図 2 中の網掛けの回路が仮想化ハードウェア IP であり，それぞれ**動作モード管理回路**，**メモリアクセス制御回路**，**デバイスアクセス制御回路**，そして**割り込み処理監視回路**である．この 4 種類のハードウェア IP によって，システム LSI の仮想化を実現している．以下ではこれらの IP に関してその役割を説明する．

3.2 動作モードの管理

動作モード管理回路は，HV モードと通常モード間の動作モードの遷移を管理する回路である．本技術によるシステム LSI の仮想化は，ハードウェア IP を既存のプラットフォームに付加するもので，動作モードの管理はプロセッサとハイパーバイザが格納されているメモリを結ぶバスを監視することで実現する．また，システムが HV モードかどうかを他の 3 種類の仮想化ハードウェア IP に対して通知する．この動作モードの識別信号を利用することで，仮想化のために重要な情報が格納されるメモリ領域のアクセス制御の保護を実現している．このメモリ領域には HV モードで動作するハイパーバイザしかアクセスできない．

動作モードによるこの保護領域の見え方の違いは図 3 のようになる．図 3 のように，保護領域には，ハイパーバイザのコード領域とデータ領域，そしてメモリアクセス制御情報とデバイスアクセス制御情報を設定するためのレジスタが存在する．

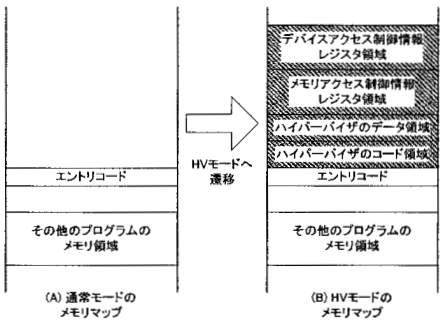

図 3 動作モードによる保護領域の見え方の違い
Fig. 3 Protected area with HV mode

プロセッサ上でゲスト OS が動作している時は動作モードは通常モードであり，この時は図 3 (A) のように保護領域は存在しないように見える．この結果，プロセッサを含む機能モジュールからこの保護領域へのアクセスは遮断される．

プロセッサが起動したりリセットした直後や，ゲスト OS がハイパーバイザの提供する機能を使用するためにハイパーバイザを呼び出した時には，エントリコードと呼ぶ HV モードへ遷移するためのチェックを行うコードが実行される．動作モード管理回路はそのエントリコードが正しく実行されたかどうかを監視している．エントリコードが正しく実行されたことを動作モード管理回路が確認すると，動作モードを HV モードへ遷移させる．すると図 3 (B) のように，ハイパーバイザのコードやデータ領域をはじめ，メモリアクセス制御回路およびデバイスアクセス制御回路が備える設定情報にアクセスできるようになる．

逆に HV モードから通常モードへの遷移は，あらかじめ決められた命令のフェッチを検出して行う．

図 3 の保護領域には，メモリアクセス制御回路とデバイスアクセス制御回路が備える設定用レジスタ群も配置する．動作モードが HV モードかどうかは，動作モード管理回路から通知される動作モード識別信号で判断する．この二つの回路は，それぞれメモリやデバイスとバスの間に配置する．そして，プロセッサ上で

動作するゲストOS等のソフトウェアやハードウェアエンジンをはじめとする，システムLSIが搭載している全ての機能モジュールからのアクセスをチェックし，その結果に従いアクセスの許可および不許可を制御する．

3.3 割り込み処理の監視

仮想化支援機構を備えないプロセッサを利用して仮想化する場合，デバイスの起こした割り込みを適切なゲストOSに配送しなければならない．

一般にプロセッサは，割り込み要因に対応する処理を記述した割り込みベクタテーブル（または処理ルーチンのアドレス）を参照して割り込み処理を行う．従来の仮想計算機技術では，仮想化層がデバイス割り込みを一旦全て受け取ってそれを適切なソフトウェアに配送している．しかし，本方式はプロセッサの仮想化支援機構を前提にできないため，割り込みベクタテーブルが配置されるメモリ領域を保護できず，デバイス割り込みの適切な配送ができない．そこで，割り込みベクタテーブルを保護する代わりに，全てのゲストOSは割り込みを受け取るとまずハイパーバイザを呼び出すようにする．

本方式における割り込み処理の様子を図4に示す．

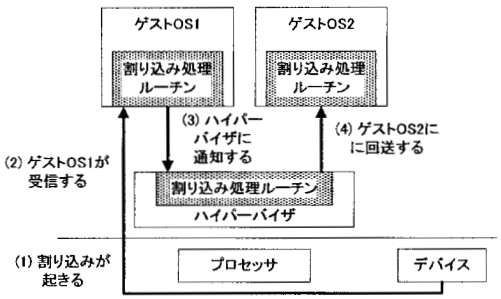

図4 デバイス割り込み処理概念図
Fig. 4 Device interrupt processing

デバイスが割り込みを起こすと，その時点でプロセッサ上で動作しているゲストOS（図4ではゲストOS1）に割り込みが通知され，ゲストOSの割り込み処理ルーチンが実行される．すると，ゲストOSはハイパーバイザにデバイス割り込みを受信したことを通知する．ゲストOSからデバイス割り込みの通知を受けたハイパーバイザは，その割り込みの内容を調べて適切なゲストOS（図4ではゲストOS2）に配送する．

もしゲストOSがこの取り決めを守らなかった場合は，プロセッサをリセットすることで間違いを防止する．

この仕組みによって，割り込みベクタテーブルが保護できなくても，ハイパーバイザがデバイスの起こした割り込みを全て把握することができ，適切なゲストOSへの配送が可能となる．

4. 仮想化ハードウェアIPの動作

第3章で説明したように，本仮想化技術は4種類のハードウェアIPによって安全で信頼性の高いハードウェアおよびソフトウェアの実行環境を実現している．本章では，これらのハードウェアIPに関して詳細に説明する．

4.1 HVモードへの遷移条件

動作モード管理回路で管理しているHVモードは保護された領域にアクセス可能な動作モードなので，HVモードと通常モードは厳格に区別し，動作モードの遷移は正確でなければならない．このため，ハイパーバイザが格納されているメモリ領域から命令をフェッチしている時のみHVモードとし，ハイパーバイザ以外のユーザコードが動作しないことを保証しなければならない．その上でさらに，HVモードへの遷移に関して考慮する必要がある．

まず，動作モードをHVモードへ遷移させる入口を**エントリコード**と呼ぶ命令列の一箇所に絞る．動作モードをHVモードへ切り替えることができる場所を一箇所に絞ることで，ユーザコードからハイパーバイザコードの途中にジャンプして動作モードをHVモードに切り替えられることを防ぐことができる．

また，動作モードがHVモードの時はプロセッサは割り込み禁止状態でなければならない．割り込みが有効の状態でハイパーバイザコードの実行中に割り込みが発生すると，HVモードのまま割り込み処理ルーチンへジャンプしてしまう．このとき，割り込みベクタテーブルや割り込み処理ルーチンが保護可能であってハイパーバイザの一部であれば問題はない．しかし，仮想化支援機構を備えないプロセッサではこれを実現できないため，割り込みを悪用されると任意のコードをHVモードで実行可能であり，結果として保護しようとしていた情報を操作されてしまうからである．

したがって，HVモードでユーザコードが実行されないことを保証するためにHVモードへの切り替え時にチェックすべきことは以下の2点となる．

(1) エントリコードが実行されること
(2) 割り込み禁止状態であること

さらに上記2点を確認する処理が不可分に行われることも条件となる．ここで「不可分に行われる」とは，2つの処理の間に別の処理が行われないことを意味する．

4.2 動作モード管理回路の実装

前節で説明した通り，HVモードへ切り替える条件は2点を確認することと，その確認処理が不可分に行われることである．

本技術では，この動作をプロセッサに外付けのハードウェアで実現する．つまり，プロセッサの外部から観測可能な現象を利用して上記条件を確認する．

システムLSIが搭載するプロセッサが前記2つの処理を1命令で実行できる場合は，エントリコードにその命令を配置して，その命令のフェッチを検知することで条件を満たしたと判断できる．しかし，上記2つの処理を1命令で実行できないプロセッサの場合，上記2つの処理の他に，その2つの処理が割り込まれなかったことを確認する必要がある．

後者の場合，以下の図5のように，プロセッサを割り込み禁止状態にする命令(A)，プロセッサの割り込み状態を含むデータを読み出す命令(B)，そして読み出したプロセッサの割り込み状態を含むデータをメモリに書き出す命令(C)を，この順にエントリコードとして実行する．

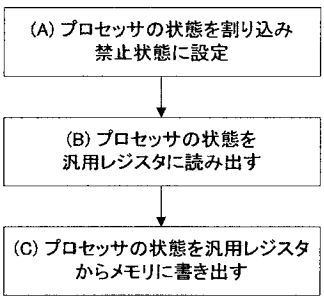

図5 HVモードエントリコードの処理手順
Fig. 5 Entry code sequence

そして以下の3点を確認することで，前記の2点を確認しかつこれらの処理が不可分に行われたことを保証できる．

(i) (A) → (B) → (C) の順にフェッチされること
(ii) ((C) の実行結果としてバスに流れたデータから) プロセッサが割り込み禁止状態であること
(iii) (A) のフェッチから (C) で割り込み禁止状態を確認するまでに要した時間が予め設定しておいた制限時間よりも短いこと

エントリコード実行中に割り込みが起きると，レジスタの退避やプロセッサ状態の変更といった付加処理が行われる．つまり，上記処理が割り込まれずに実行された場合に比べ時間がかかる．エントリコードの実行に要した時間をチェックしているのは，このことを利用している．

なお前記(i)のチェックのため，エントリコードはキャッシュされないメモリ領域に格納し，プロセッサからの命令フェッチがバスに出るようにする．

4.3 メモリアクセス制御回路

メモリアクセス制御回路は，ハイパーバイザによる設定に従って，ゲストOSやシステムLSIが搭載する機能モジュールに対して実メモリを仮想化したメモリ空間を提供し，かつある機能モジュールが他の機能モジュールに割り当てられたメモリ領域にアクセスしないように制御を行う回路である．

この機能の実現には，機能モジュールが出したメモリアクセスのアクセス元を正しく判別して，そのアクセス元がアクセス先アドレスへのアクセスが許可されているかどうかをチェックし，かつアクセスが許可された場合には実メモリ領域へのアクセスに変換する．このアクセス制御に使用される情報や情報を設定するためのレジスタは，図3の保護領域内に存在する．

また，アクセス元のチェックにはバスマスタIDを使用し，プロセッサ上で動作するゲストOS等のソフトウェアだけでなく，DSPやハードウェアエンジンといった機能モジュールによるメモリアクセスを含めた一括制御を実現している．

アクセス権のチェックや実メモリ空間とのマッピングの方法は，従来よりよく使用されているメモリ管理手法と同等のものであり，割り当てたメモリと機能モジュールとを対応付けて管理し，機能モジュールがバスに出したアドレスを実メモリ空間のアドレスに変換する．

ただし，本メモリアクセス制御回路では制御の単位がプロセッサ上で動作するゲストOSやプロセス単位ではなくバスマスタ単位となる．つまり，制御の単位がバスマスタ単位となるため，プロセッサのように同じバスマスタ上で複数のソフトウェアが動作する場合には，ハイパーバイザによるコンテクストスイッチ時に，そのバスマスタに対応するアクセス制御設定を入れ換える．またキャッシュを備えるプロセッサの場合は，次に動作するソフトウェアが，直前に動作していたソフトウェアがキャッシュに残したデータにアクセスすることがないように，キャッシュをフラッシュする．

4.4 デバイスアクセス制御回路

デバイスアクセス制御回路は，ハイパーバイザによる設定に従って，システムLSIが搭載する機能モジュールのデバイスへのアクセスを制御する回路である．

アクセス制御は，バスマスタIDによってアクセス元を判別し，各バスマスタ毎に設定されたアクセスの許可情報に基づいてアクセス先アドレスを操作することで実現している．このアクセス制御に使用される情報や情報を設定するためのレジスタは，図3に示した保護領域内に存在する．

メモリアクセス制御と同様に制御の単位がバスマスタ単位であるため，プロセッサのように同じバスマスタ上で複数のソフトウェアが動作する場合には，ハイパーバイザによるコンテクストスイッチ時に，そのバスマスタに対応するアクセス制御設定を入れ換える．

4.5 割り込み処理監視回路

本技術においては，ハイパーバイザがデバイス割り込みを適切なゲストOSに配送するために，デバイス割り込みを一旦ゲストOSに受信させ，その後全てハイパーバイザに通知させるようにしている．ただこの

ままではゲスト OS の不具合や悪意のゲスト OS によってデバイス割り込みを握り潰される（図 4 においてゲスト OS1 が (3) の処理をしない）可能性がある．そこで，デバイス割り込みが発生してから一定時間が経過してもハイパーバイザに通知されなかった場合は，プロセッサをリセットして，原因となっているゲスト OS の実行を強制的に停止させる．

ここで，この仮想化アーキテクチャにおけるデバイス割り込みの処理動作を図 6 に示す．この図 6 の処理において，割り込みを起こすデバイスはゲスト OS2 が管理しているものであり，この割り込みの配送先はゲスト OS2 である．

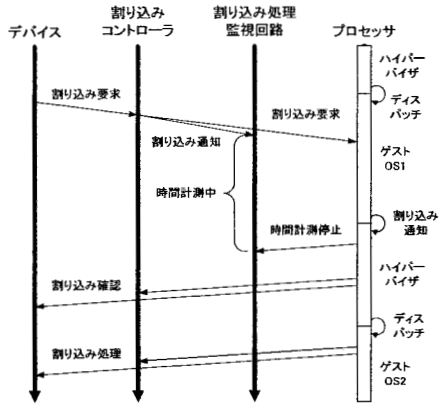

図 6 正常動作時のデバイス割り込み処理シーケンス
Fig. 6 Normal sequence of interrupt processing

この図 6 のように，割り込み処理監視回路はデバイス割り込みの通知を受け取ると時間計測を開始する．そのデバイス割り込みの要求を受け取ったゲスト OS1 が割り込みをハイパーバイザに通知すると，ハイパーバイザはまず割り込み処理監視回路の時間計測を停止させる．その後，ハイパーバイザは割り込みの確認やゲスト OS2 への割り込みの通知等の処理を行う．これが通常動作時の処理である．

ここでもしゲスト OS1 がハイパーバイザにデバイス割り込みを通知せずに割り込みを握り潰そうとした場合は，割り込み処理監視回路が制限時間の経過を検知してプロセッサにリセットをかけ，強制的にゲスト OS1 の実行を停止させる．そしてリセットしたプロセッサでハイパーバイザが動作し，制御を戻すことができる．したがって，デバイス割り込み要求を受けたゲスト OS に悪意があったり不具合が起きていて，受け取ったデバイス割り込み要求をハイパーバイザに正しく通知しなかった場合でも，ハイパーバイザはこのデバイス割り込みを正しい配送先であるゲスト OS2 に配送することができる．

この仕組みの実現にあたり，ゲスト OS を，デバイス割り込みを通知された際にハイパーバイザが用意した割り込み通知用のインターフェースを呼び出すように変更する必要がある．しかし，この変更のコストはそれらのソフトウェアをシステム LSI 上で動作させるために行った変更のコストと比較して大きいものではないと考えている．

5. おわりに

本報告では，ハイパーバイザとの連携によってシステム LSI 全体を対象とした仮想化を実現するハードウェア IP のアーキテクチャと，その仮想化方式について報告した．

今後，この方式のフィージビリティを検証するための試作を進めていく．

参考文献

1) IBM: *Book III: PowerPC Operation Environment Architecture* (2003).
2) Intel: *Intel Virtualization Technology Specification for the IA-32 Intel Architecture* (2005).
3) Braham, P., Dragovic, B., Fraser, K., Hard, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *In Proceeding of the ACM Symposium on Operating Systems Principles*, pp. 164–177 (2003).