Vol.2010-ARC-189 No.3
Vol.2010-OS-114 No.3
2010/4/21

# 組み込み向けマルチコアプロセッサにおける複数 OS 実行環境の構築技術

杉本健 野尻徹 平松義崇 寺田光一†

我々は，マルチコアプロセッサ上で複数の OS を起動することによって，既存のアプリケーションを改変せずに並列実行することを可能にするソフトウェアとして，ExVisor を提案している．本研究では ExVisor の実用化に向け，実機環境における ExVisor の運用の検証を行った．各 OS に対する CPU，メモリ，I/O を割り当て，保護する設定を行い，複数の OS を動かすメモリマップを検討し，割り込みに対する対処を行った．複数の CPU コアと，メモリアクセス制御ハードウェア (PPC)を搭載する試作 SoC にて評価し，オーバーヘッドが 0.002%以下である事を確認して，ExVisor 実用化の見通しを得た．

# Study of Multi Operating System Platform for Embedded Multicore Processor

KEN SUGIMOTO　TOHRU NOJIRI
YOSHITAKA HIRAMATSU　KOICHI TERADA†

We propose ExVisor which is software that enables to execute existing applications in parallel without any modification by executing two or more OS on the multicore processor. In this study, we verify the operation of ExVisor on the real chip environment for the practical use of ExVisor. We examine the allocation and protection of CPU cores, memories, and I/Os for each OS, the assignment of memory map for two or more OS, and control of interrupt. We evaluate them with real SoC equipped with eight CPU cores and memory access control hardware (PPC), confirm the overhead is 0.002% or less, and obtain the prospect of the ExVisor practical use.

## 1. はじめに

近年組み込みシステムにおいて，並列に動作させるアプリケーションの数を増やしたいというニーズがある．例えば，自動車においては，エンジンやブレーキ等の制御に加え，カーナビやテレビ等の複数のマルチメディアのアプリケーションを並列に動作させたいというニーズがある[1]．一方テレビにおいては，複数の番組を映しながら番組の録画を行い，同時にインターネット経由で情報を取り出したい，というニーズがある．これらのニーズに対して，現在の制御系では，複数のアプリケーションそれぞれに対して複数のプロセッサ LSI を用意して，各 LSI でアプリケーションを動作させるといった方法が採られる．しかしこのままでは，アプリケーション数の増加に従って，必要なハードウェアコストが増大してしまうため，これを抑える一つの方法として，マルチコアプロセッサの利用が考えられている．

マルチコアを利用して複数のアプリケーションを並列に動作させる方法として，マルチコア用のマルチタスク OS を作成し，すべてのアプリケーションをその OS 上で動かす方法や，VMWare[2]，Intel VT[3]等の仮想化技術を利用して，仮想環境で複数の OS を動かす方法がある．しかし以下で述べるように，組み込みシステムでこれらの方法を用いると，ソフトウェアの開発コストが多大となることが懸念される．

組み込みシステムの OS には，μITRON 等，アプリケーションの目的に特化したものが多く，これらを 1 種類の OS に集約するには大きな工数がかかるという課題がある．また，VMWare や Intel VT 等の仮想化技術を利用する方法では，OS の種類の制約は無くなるが，これらの仮想化技術はハードウェアからソフトウェアまでの複数の階層に渡る仕組みを用いるため，実処理時間の予測が難しい．その為，リアルタイム性の確保が確実で無くなるという課題がある．

そこで，我々は既存のアプリケーションをその特性に関わらずそのまま再利用することを目指し，これを実現するために，マルチコア上で複数の OS を物理的に分離した形で立ち上げることを可能にする ExVisor の研究を進めている[4][5]．複数の OS を物理的に分離した形で立ち上げる為には，OS 間の干渉をコントロール出来る必要がある．その為の方法として ExVisor の研究ではまず，OS に CPU コア，外部メモリ，I/O デバイスを割り当てるアーキテクチャの検討を行い，この為に必要な機構として，PPC(Physical Partitioning Controller)というハードウェアを提案している．我々はその中で，ExVisor を実際のアプリケーションに適用する方法についての研究を進めている．今回，実際のアプリケーション及びその環境を，Linux とμITRON の 2 つの OS を起動し，Linux 上で OpenCV[7]を用いた顔検出を行うアプリケーションを動かし，μ

* † (株)日立製作所 中央研究所
Hitachi Ltd. Central Research Lab.

Vol.2010-ARC-189 No.3
Vol.2010-OS-114 No.3
2010/4/21

ITRON 上で映像再生アプリケーションを動かす，次世代高機能テレビアプリケーションの試作と定めた．そしてこの環境を構築するために，各 OS に対する CPU，メモリ，I/O を割り当て，保護する設定を行い，複数の OS を動かすメモリマップを検討し，割り込みに対する対処を行った．そして，それが CPU コアを複数搭載し，メモリアクセス制御を行うハードウェア (PPC)を含む SoC で動作することを確認した．

本稿では，まず開発プラットフォームとして使用したマルチコア SoC について述べた後，試作したアプリケーションソフトウェアの構成について説明する．続いて，具体的なソフトウェア実装と，さらにその評価結果について述べる．

## 2. マルチコア SoC

ExVisor を用いて既存アプリケーションを複数動作させる為の動作確認環境として，(株)ルネサステクノロジが開発した試作チップであるマルチコア SoC[6]を用いた．本章では，このマルチコア SoC チップについて述べる．

### 2.1 マルチコア SoC の概要

アーキテクチャの概要を図 1 に示す．また，チップ写真を図 2 に示す．CPU コアとしてクラスタあたり 4 個の SH-4A 相当のプロセッサで構成された 2 個のクラスタ，動的再構成プロセッサ(DRP）として 4 個の Flexible Engine (FE)，及び映像処理向けの映像プロセッサ Video Processing Unit（VPU）が搭載されている．以下に特徴をまとめる．

・クラスタ 0 は SH×4 個と VPU，クラスタ 1 は SH×4 個と FE×4 個のヘテロ構成．
・各クラスタに，それぞれ 1 チャネルの DDR3 インタフェース(DBSC)を装備．
・クラスタ間をバススイッチにより相互接続．
・動作周波数は，SH が 648MHz，FE が 324MHz，バスが 324MHz，VPU が 162MHz，共有メモリとして使われる DDR3-SDRAM は 162MHz．製造プロセスは 45nm．
・PPC は各スレーブとバス（SHwy）との間に配置．

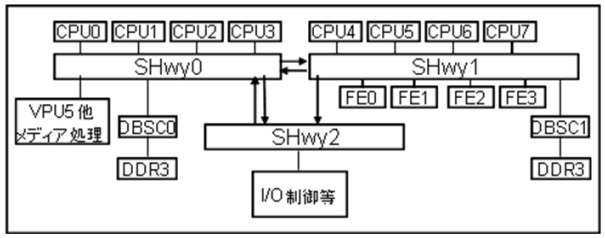


図 1 マルチコア SoC 構成図

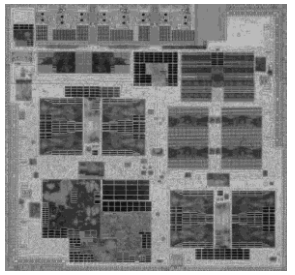

図 2 マルチコア SoC チップ写真

### 2.2 SH メモリアーキテクチャ

本節では，マルチコア SoC に CPU コアとして搭載されている，SH-4A プロセッサのメモリアーキテクチャの特徴について述べる．

SH-4A は図 3 に示すような，複数の物理アドレス空間モードを持つ．CPU からアクセスするための論理的なアドレス空間である「仮想アドレス空間」は 32bit 固定の空間である．この空間は，P0(U0), P1, P2, P3, P4 と呼ばれる 5 つの領域に分割されており，それぞれの領域は次のような特徴付けがなされている．

表 1 SH メモリアーキテクチャの領域分割

| 領域 | 主な用途 | Cache | MMU |
|---|---|---|---|
| P0(U0) | ユーザプログラム | On | On |
| P1 | OS | On | Off |
| P2 | OS | Off | Off |
| P3 | OS | On | On |
| P4 | I/O 空間マッピング | Off | Off |

SH-4A は，これら複数の領域に分かれている仮想アドレス空間のアドレスを，29bit や，32bit ないしそれ以上の領域となる物理アドレス空間にマッピングするための機構として，PMB (Privileged Mapping Buffer) と呼ばれる機構を持つ．この機構により，CPU は，任意の物理アドレス空間へのアクセスを行うことが可能となる．PMB の詳細については次節で述べる．

Vol.2010-ARC-189 No.3
Vol.2010-OS-114 No.3
2010/4/21

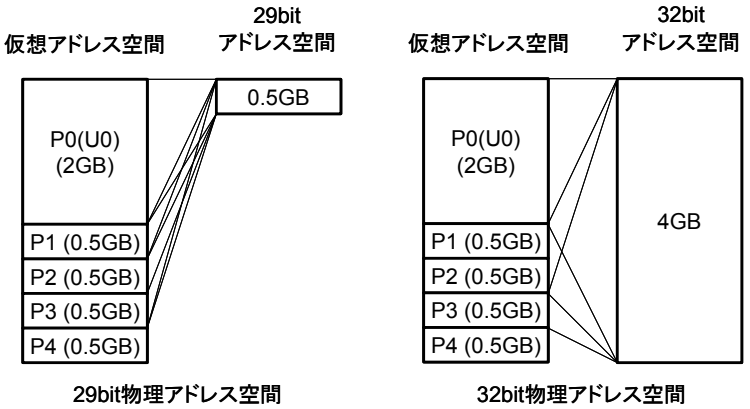


図 3　SH-4A メモリアーキテクチャ

### 2.3　PMB

マルチコア SoC に搭載された SH-4A は，TLB (Translation Look-aside Buffer)の類似機能である，PMB(Privileged Mapping Buffer)を持つ．TLB と PMB は，MMU (Memory Management Unit)に内蔵された仮想アドレスから物理アドレスへのアドレス変換を行うユニットであるという点で同じである．しかし，TLB は一般にユーザ空間のアドレス変換を行うのに対して PMB は特権空間のアドレス変換を行うという点で異なる．この PMB を用いることにより，μITRON などの仮想記憶機構を持たない OS であっても，アドレス変換を可能とすることができる．PMB 自体，OS からは隠ぺいされているアドレス変換機構として働くことが特徴であるといえる. PMB はこのような目的のために導入されているため, 仮想ページサイズが 16MB～512MB 単位と TLB のそれより大きいことも特徴である．

さて，OS が使用する特権アドレス空間は OS と強く結びついており，アドレスの変更が難しい場合も多い．このため，複数の OS を立ち上げる場合，それぞれの OS が使用する特権アドレス空間が重なる可能性がある．しかし，PMB を用いれば，特権アドレス空間から物理メモリアドレス空間への割り当てを OS ごとに変更することによって，対応可能となる．今回，実際に PMB を利用して複数 OS の立ち上げ環境を作成した．具体的には，4.4.2 節で述べる．

## 3.　試作アプリケーション

今回試作した次世代高機能テレビアプリケーションの概略構成を図 4 に示す．大きく２つのアプリケーションを同時に動かす構成であり，うち１つはμITRON OS 上で動作させる映像再生アプリケーション, もう１つは Linux OS 上で動作させる顔検出アプリケーションである．

以下，それぞれのアプリケーションについて説明する．

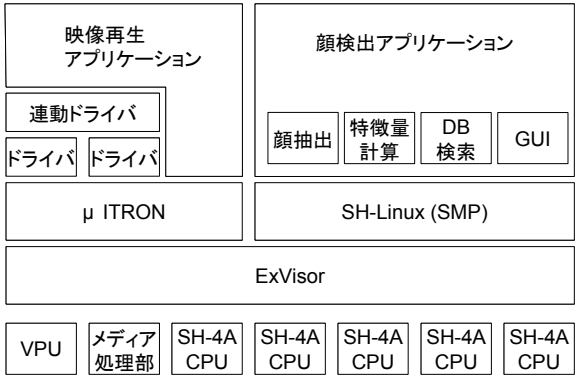


図 4　試作アプリケーションの構成

### 3.1　映像再生アプリケーション

マルチコア SoC にハードウェアとして搭載されているメディア処理モジュールを利用して，リアルタイムの映像再生を行うアプリケーションである．

利用するメディア処理モジュールとしては，映像デコードを行う VPU (Video Processing Unit)のほか，映像の拡大縮小や，色空間変換，グラフィックとの重ね合わせなどの機能に対応したモジュール群がある．ソフトウェアとしては，これらのハードウェアモジュールを制御する各ドライバソフトウェアや，ドライバ同士を連携させるための連動ドライバソフトウェア，およびそれらの全体制御を行うアプリケーションソフトウェアとから構成される．

### 3.2　顔検出アプリケーション

マルチコア SoC に搭載されている複数の CPU コアを利用する，SMP Linux OS 上に構築する画像処理アプリケーションである．画像に含まれる顔の検出を行い，その特徴量を計算し, あらかじめデータベースに記録している顔画像との比較を行うことで，類似の顔画像を検索する，という機能を持つ．

搭載する主な機能のうち，顔検出機能ソフトウェアとしては，OpenCV[7]を，特徴量計算機能と類似画像検索機能ソフトウェアとしては，EnraEnra[9]を，それぞれ利用

Vol.2010-ARC-189 No.3
Vol.2010-OS-114 No.3
2010/4/21

した．

### 3.3 試作アプリケーションの実装制約

以上述べた試作アプリケーションを実装するにあたり，我々はいくつかの実装上の制約を見出した．ここでは，それらのうち代表的なものを例にとり，説明する．

第一章に述べたように，我々は，既存アプリケーションを改変することなく，マルチコア SoC 上で再利用することを目標としている．今回再利用する対象としたアプリケーションソフトウェアのうち，特に映像再生アプリケーションは，映像処理のための複数のハードウェアモジュールを利用するための個別ドライバや，それらドライバを連動させるための連動ドライバなどから構成される，という特徴がある．

これらドライバソフトウェアは，映像データ処理に必要な高性能処理が求められることから，さまざまな性能チューニングが施されており，また複数のドライバが緊密に連携して動作している状況にある．このため，例えば使用する物理アドレスの改変が相当困難である，という制約があることが分かった．

なお，このような制約は，今回利用したソフトウェアに限らず，同様のリアルタイム性能が必要なアプリケーションソフトウェアであれば，類似の制約となるものと考えられる．

## 4. 実装

本章では，ExVisor の動作とそれを用いた複数 OS 実行環境実装の詳細について述べる．まず，ExVisor を用いた複数の OS のブートシーケンスについて述べる．次に，各ドメインに対する CPU やメモリ，I/O の割り当てと保護について述べる．その次に，ExVisor 環境における割り込み及び例外処理について述べる．最後に，実際のドメインの設定例として，マルチコア SoC 上で $\mu$ ITRON と Linux の２つのドメインを動作させる今回の試作例を述べる．

### 4.1 ブートシーケンス

一般に，IPL はハードウェアの初期化と OS のイメージのロードを行う．ExVisor を用いた複数のドメインのブートシーケンスでは，IPL を，ハードウェアを初期化するコードと OS のイメージをロードして起動するコードの２つに分割し，それぞれ別のタイミングで用いる．これらの 2 つの IPL と ExVisor を用いて，ドメインは次の手順で起動される．

(1) まず，ハードウェアを初期化する IPL が起動し，DDR メモリ等の初期化処理が実行される．
(2) IPL は動作終了後，ExVisor のアドレスにジャンプする．
(3) ドメインを起動する CPU のリセットベクタを ExVisor が設定する．リセットベクタは，OS イメージをロードする IPL のアドレスにセットされる．
(4) 各ドメインの PPC を ExVisor が設定する．
(5) 各ドメインを起動する CPU を ExVisor がキックする．ただし，ExVisor が動作しているコアで起動する OS は，ExVisor の全ての処理終了後に，その OS の IPL へジャンプする．IPL は各ドメインを立ち上げる．

以上のシーケンスで ExVisor より複数のドメインが立ち上がる．なお，ExVisor はそれぞれの OS の起動コアをキックするが，OS が SMP で利用する他のコアはキックしない．それらのコアをキックするのは，その OS 自身の役割である．

### 4.2 各ドメインに対するマルチコアの CPU，メモリ，I/O の割り当てと保護

ExVisor は, マルチコアの CPU, メモリ, I/O といった資源を各ドメインに割り当て，保護することによってドメイン間でリソースが破壊されることを防ぐ. ここで, CPU, メモリ, 周辺 I/O は, それぞれの割り当てと保護において異なった方法が用いられる．また，OS の改変も若干必要である．そこで，本節ではそれぞれのハードウェアの割り当てと保護の方法について詳細に述べる．

#### 4.2.1 CPU のドメインごとの割り当てと保護

まず，CPU をドメインごとに割り当てる為に，ドメインがそれぞれ占有する CPU を規定する方法を用いた．この為に ExVisor は，ドメインと，ドメインが使用する物理 CPU 番号及びリセットベクタとの対応表を持つ．また，OS が SMP で動作する場合は，OS も自身が使用する物理 CPU 番号を保持し，その番号を ExVisor の持つ値と一致させる必要がある．これは，OS が SMP で動作する場合，起動コア以外のコアのキックは OS の役割なので, 使用する物理 CPU 番号を OS が保持していないと, OS がどの CPU をキックすればよいのかわからなくなってしまうからである．なお, OS が UP で動作する場合，必ずしも OS は使用する物理 CPU 番号を保持する必要は無い．なぜなら, UP 動作では自分以外の CPU を触らないため, その情報が必要ないからである．

次に, CPU を保護する為には, その CPU を使用するドメイン以外のドメインから，当該 CPU の制御レジスタの変更を禁止すれば良い．これを実現するために，CPU の制御レジスタのアドレスを保護する PPC を用いる．

#### 4.2.2 メモリのドメインごとの割り当てと保護

まず，メモリをドメインごとに割り当てる為に，ドメイン間でメモリの衝突を防ぐ目的で, ドメインごとに使用するメモリ範囲が重ならないようにメモリを割り当てた．この為に，OS が自身で使用するメモリの範囲を設定可能な仕組みが必要である．

また，メモリの保護する為には，ExVisor から PPC を設定する．これによって，OS プログラムのバグ等によって，想定範囲外のメモリ範囲をアクセスしても，他のドメインの動作を阻害することがなくなる．

Vol.2010-ARC-189 No.3
Vol.2010-OS-114 No.3
2010/4/21

#### 4.2.3 周辺 I/O のドメインごとの割り当てと保護

まず，ドメインへの I/O の割り当ては，ある I/O の使用を特定のドメインに限定することとした．この為に，OS にて，自身が使用する外部 I/O の選択を可能にした．

また，I/O を保護するために，I/O それぞれのメモリマップのアドレス範囲に対応した PPC を用いる．

例えば，外部 I/O として PCI-Express（以下，PCI-e）があるが，この PCI-e を使用出来るドメインは予め一つに固定しておき，そのドメインからは PCI-e に自由にアクセス出来るが，その他のドメインからのアクセスは出来ない，という形にすれば良い．

### 4.3 割り込み及び例外処理

PPC アクセス違反による例外処理は ExVisor で処理される．しかし，既存の OS を用いた場合，割り込み及び例外が発生すると OS の割り込み処理のルーチンが起動する．

そこで，OS の割り込みルーチンを書き換えることにより，CPU が例外もしくは割り込みを受け取ると ExVisor の割り込みルーチンが立ち上がるように変更する．ExVisor はその後，内容が PPC アクセス違反以外の，例えば TLB 例外等の OS が処理すべき内容であれば，OS に制御を返す．これにより，OS の割り込み及び例外も正しく処理することが出来る．

### 4.4 マルチコア SoC を用いた実際の OS の設定例

本節では，前節までで述べた仕組みをマルチコア SoC 上で実際どのように実装したのかについて述べる．今回，Linux と μITRON の２つのドメインを立ち上げた．

#### 4.4.1 起動シーケンス

今回，図 5 に示すように，μITRON を CPU0 にて UP で動かし，Linux を CPU4 で起動し，CPU5 と合わせて SMP で動かす構成にした．また，μITRON と Linux それぞれの OS 起動用の IPL は適切なアドレスに配置し，そのアドレスをリセットベクタとして，ExVisor に書き込んだ．この場合，図 6 に示すように起動処理が行われる．起動手順は次の通りである．

(1) CPU0 は，ハードウェア初期化の為の IPL，ExVisor，OS ロードの為の IPL，μITRON を，この順番に起動する．
(2) CPU0 は ExVisor 動作中に CPU4 のリセットベクタを設定し，コアを起動する．コア起動後，CPU4 では OS ロードの為の IPL が動作し，その IPL より Linux が起動する．
(3) CPU5 は CPU4 で起動した Linux から起動される．

なお，ドメイン保護の為の PPC の設定については明記していないが，ExVisor 動作中に行われている．

#### 4.4.2 メモリマップ

次に，今回構築したドメインのメモリマップを図 7 に示す．先に述べた通り，複数の OS の上で複数のアプリケーションを動作させる場合，それぞれの OS やアプリケーションは他の OS やアプリケーションと並列に動作することを想定せずに動作する為，それらが使用する物理メモリが衝突する可能性がある．これを防ぐためには，適切なメモリマップを作成する必要がある．今回，具体的に以下のように考え，設定を行った．

**A) Linux カーネル使用領域**

今回は Linux をソースコードよりコンパイルして使用した．この Linux カーネルは，P0(U0)領域と P1 領域を使用する．P0(U0)領域で使用する物理アドレスは，コンパイル時にコンフィギュレーションで指定可能であった．よって，他の OS やアプリケーションが使用する物理メモリの範囲と重ならないようにコンフィギュレーションの設定を行った．P1 領域は，一部の論理アドレスを固定的に使用する仕様となっていた．この範囲は後で述べる，μITRON 上で動作する VPU のファームウェアが使用する論理アドレスと重なっていた．しかし，これは PMB を用いることによって回避できた．詳細は μITRON のアプリケーションの節で述べる．

**B) Linux 上で動くアプリケーションの使用領域**

Linux 上で動作するアプリケーションの物理アドレス領域の管理は Linux によって行われる．よって，今回 Linux 上で類似画像検索エンジンや OpenCV 等の既存のアプリケーションを動かしたが，Linux カーネルのメモリ範囲について十分考えられていれば，これらを動かす上で物理アドレス領域について考える必要は無い．

**C) μITRON カーネル使用領域**

μITRON カーネルもソースコードよりコンパイルして使用した．このカーネルはセクション情報を変更することによって使用する物理アドレス領域を変更することが可能である．セクションごとに P0(U0)領域に物理アドレス領域を設定しなければならないか P1 領域に設定しなければならないかそのどちらでも良いかが定まっており，この点を守りつつ，他と物理アドレス領域が重ならないように使用領域の設定を行った．

**D) μITRON 上で動作するアプリケーションの使用領域**

μITRON 上で動作させたアプリケーションは，VPU ファームウェア，画面出力ドライバソフトウェアである．これらのアプリケーションも全て，セクションの設定によって使用する物理アドレス領域を設定出来る．それぞれのアプリケーションのセクションごとに，P0(U0)や P1 等の使用可能な物理アドレス領域が定まっているので，その要求に合うように使用領域を割り当てた．この中で，VPU のファームウェアは動作するメモリの物理アドレス領域が Linux のカーネルが動作する物理アドレス領域と重なっていることが判明した．また，VPU ファームウェアは高度に性能チューニングされていることから改変が困難であり，これが使用する物理アドレス領域を変更するこ

Vol.2010-ARC-189 No.3
Vol.2010-OS-114 No.3
2010/4/21

とも出来ない．しかし，それぞれの OS で異なる PMB の設定を行い，それぞれの論理アドレスが指し示す物理アドレスが異なるように設定することによって，アドレスの衝突を防ぐ事が出来た．PMB を用いることで，変更がほぼ不可能な同じメモリ範囲を使用する２つの既存のアプリケーションを並列に動作させることが可能になったと言える．

**E)　Linux と μITRON の共有領域**

Linux と μITRON 間での情報の通信を行うために，これらの OS の共有領域を用意した．この領域は，Linux と μITRON 両方からアクセス可能な領域に他と重ならないように物理アドレス領域をとれば良い．今回，アプリケーションから共有領域のアドレスを直接指定することによって共有領域への読み書きを行った．

以上のように，必要な大きさのメモリをそれぞれの OS，アプリケーションごとに適切に割り当て，メモリマップを作成した．また，論理メモリアドレスの衝突を避けるために，PMB を活用した．

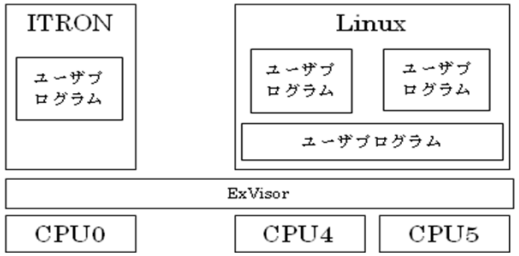


図 5 ブート環境構成図

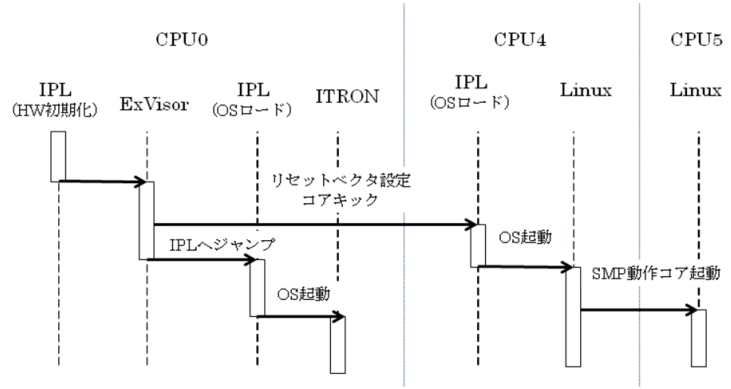


図 6 ブートシーケンス

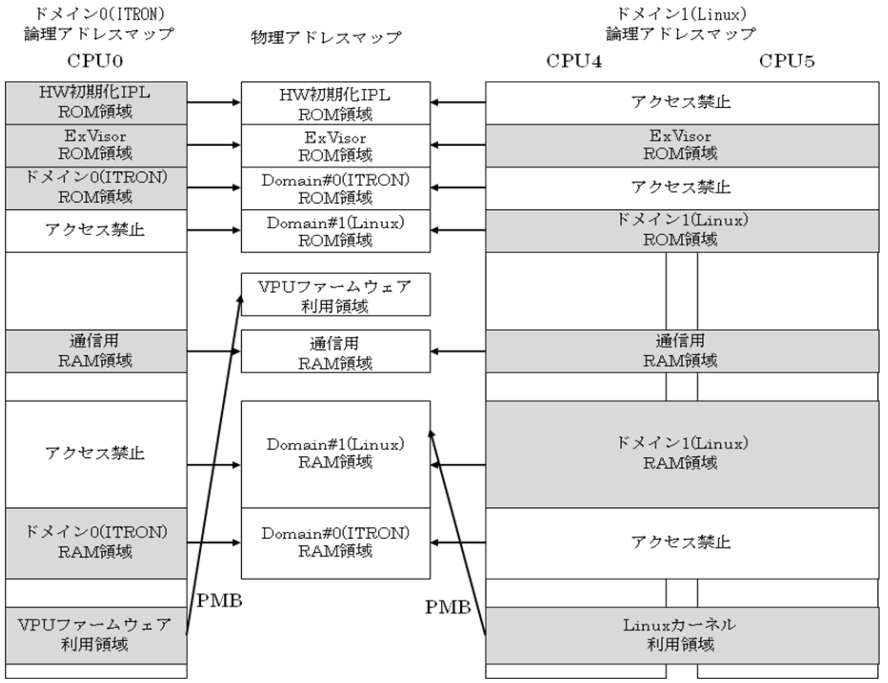


図 7　メモリマップ

**4.4.3 割り込みルーチン**

最後に，今回構築した割り込みルーチンの動作について述べる．以下，具体例を示す．図 8 は ExVisor が受け取った割り込みをドメインに転送する場合のシーケンスを示したものである．以下，文中の数値は図中の数値と同じ段階を示している．

(1)　最初，アプリケーションプログラムが実行されている．
(2)　デバイスから割り込み要求が発生すると，CPU コアが割り込みを受け付ける．
(3)　VBR(Vector Base Register:割り込みハンドラの位置を記憶するレジスタ)の値を OS の割り込みハンドラから ExVisor の割り込みハンドラに書き換えておくことにより，ExVisor の割り込みハンドラを開始する．この時，特定のレジスタを介して，OS の割り込みハンドラを ExVisor に渡す．
(4)　ExVisor の割り込みハンドラは，当該割り込みをドメインに転送するべきか判断する．
(5)　ドメインに転送するべきだと判断した場合には OS から渡された OS の割り込みハンドラのアドレスからエントリポイントを計算し，適切な割り込みハンドラを

Vol.2010-ARC-189 No.3
Vol.2010-OS-114 No.3
2010/4/21

実行する．このとき，割り込みからの戻りアドレスとして，ExVisor の割り込みハンドラのプログラムではなく，元々の割り込まれたプログラム（図 8 ではアプリケーションプログラム）のアドレスを設定して転送するため，この後の割り込み処理に ExVisor が関与することはない．

(6) OS の割り込みハンドラ内では，割り込み処理完了後，通常，割り込み要求元の割り込み要求をクリアする処理を行い，コンテキストスイッチの必要がなければ元のアプリケーションプログラムの実行が再開される．この動作の詳細は OS 実装依存であり，ExVisor はその詳細に関与しない．

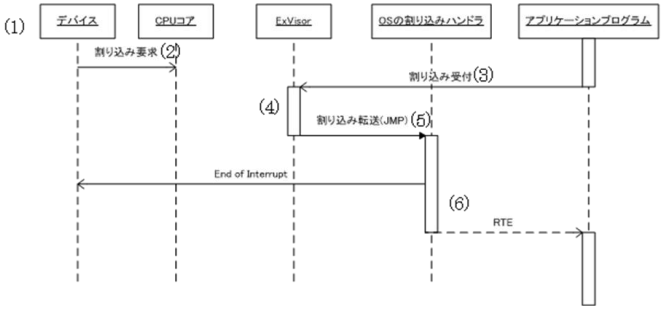


図 8 割り込み処理の転送シーケンス

## 5. 評価

### 5.1 アプリケーション再利用による開発期間の短縮

ExVisor によるアプリケーションの再利用によって，次世代高機能テレビアプリケーション試作の実装開発期間を大幅に短縮することができた. 実装は, マルチコア SoC 搭載の評価ボードを入手してから，既存のアプリケーションのポーティング期間が１ヶ月，情報制御連携部の実装に２ヶ月の合計３ヶ月という短期間で完了した．仮に，ExVisor によってアプリケーションの再利用が行えなかったとしたら，VPU のファームウェアをはじめとするμITRON 上で動作するアプリケーションを全て Linux に移植する必要がある．これを３ヶ月という短期間で終えることは困難である．そもそも，VPU のファームウェアのように，高度にチューニングされているため改変が困難なアプリケーションや，権利関係の都合でソースファイルが提供されないアプリケーションの場合、移植する事は出来ない．また，もし移植が出来たとしても，Linux 上で映像のリアルタイム処理を行うことは，OS の特性から困難である．

### 5.2 ExVisor の割り込み処理による性能への影響

ExVisor を用いた場合，割り込み処理でオーバーヘッドが発生する．オーバーヘッドはμITRON 等のリアルタイム OS で問題になる可能性がある．そこで，本節ではこのオーバーヘッドを評価する．

4.3 節で述べた通り, CPU は割り込みを受け取ったらまず ExVisor の割り込み処理のルーチンが起動する．そしてそのルーチンは割り込みの種類に応じて，自分自身で処理を行うか，若しくは割り込みを OS に通知する．ここで，ExVisor 自身が処理する割り込みは PPC 違反のみであるが，この例外は通常発生しないように設計される．その為，アプリケーションが異常動作しない限り，ExVisor は受け取った全ての割り込みを OS に通知する．よって，通常時の ExVisor の割り込み処理によるオーバーヘッドは，ExVisor が OS に処理を通知する部分である．図 8 で言えば，(4)のフェーズのみが ExVisor によるオーバーヘッドである．

そこで今回，割り込みを ExVisor が受け付けてから OS へ処理が移るまでに必要なクロック数を測定し，また，一定のサイクル間にアプリケーションにて発生する割り込みの回数を測定することによって，ExVisor の割り込み処理がアプリケーション全体に与えるオーバーヘッドを測定した．対象アプリケーションはリアルタイム OS で動作するものが望ましいので, 今回実装したμITRON 上で動作する映像再生アプリケーションを用いた．

まず，実装した ExVisor の割り込み処理の命令を計測した所，OS に処理が移るまでの処理は，命令にして 25 命令であり，実行に必要なクロック数はキャッシュミスや分岐予測ミスの可能性を考慮しても，最大で 100 サイクル程度であるとわかった．

次に，一定クロックサイクル中何回例外または割り込みが発生したかを，マルチコア SoC のパフォーマンスカウンタを用いて測定した結果，3,790,000,000 サイクル中割り込みと例外が合わせて 754 回発生していることが測定された．オーバーヘッドは

(754 * 100) / 3790000000 = 1.99 * 10^-5

より最大で 0.002%程度と計算できる．よって，リアルタイム OS を動かす上でも，ExVisor は実用上問題ないと言える．

## 6. おわりに

### 6.1 本研究の成果

マルチコアプロセッサ上で複数のOSを起動し, 既存アプリケーションの並列実行環境を構築するソフトウェアであるExVisorに関して，実際のアプリケーションの適用方法についての研究を進めた．今回は，ターゲットとして，Linuxとμ ITRONの2つのOSを起動し，Linux上でOpenCV[7]とEnraEnra[9]を用いた顔検出・検索を行うアプリケーションを動かし，μITRON上で映像再生アプリケーションを動かす，次世代高機能テレビアプリケーションの試作と定めた．そしてこの環境を構築するために，各OSに対

するCPU，メモリ，I/Oを割り当て，保護する設定を行い，複数のOSを動かすメモリマップを検討し，割り込みに対する対処を行った．さらに，それがCPUコアを複数搭載し，メモリアクセス制御を行うハードウェア (PPC)を含むSoCで動作することを確認した．また，ExVisorによる速度低下は0.002%以下であり，実用上問題ない事を確認した．

以上により，ExVisorによって既存のアプリケーションの並列化を行う事に対する実用化の見通しを得た．

### 6.2 今後の課題

#### 6.2.1 アプリケーション再利用による工数削減量の評価

今後アプリケーションの再利用が可能になったことによる，工数の削減量を評価する必要がある．これは，あるアプリケーションを ExVisor を用いて再利用した時と用いずに移植を行った場合それぞれの工数を算出し，比較すれば良い．例えば，アプリケーションを移植した実例とその時にかかった工数を入手し，次に，ExVisor を用いて移植を行った場合の工数を算出することによって，工数の比較を行うことが出来ると考えられる．

#### 6.2.2 キャッシュの制約の回避

ドメイン間共有メモリをキャッシュ可能領域に配置した場合，通常，ドメイン間でコヒーレンシを保つことは無いので，共有領域のコヒーレンシを保つハードウェア機構が存在しない，という問題がある．これは例えば，領域ごとにコヒーレンシを保つ設定を可能にすれば，解決可能である．さらに，PPC のアクセスコントロールリストをコアが認識し，その設定に応じて，特定の領域ごとにあるコアはどのコアとコヒーレンシを取るべきなのか自動的に判別して，設定を自動化する事も考えられる．

#### 6.2.3 デバイスの仮想化

割り込みなど，ドメインごとに分割できず，ドメイン間で共有せざるを得ないデバイスが幾つかある．オーバーヘッドとのトレードオフなど検討するべき項目が多く，今後の課題である．

#### 6.2.4 設定を簡素化する為のコンフィギュレータの開発

ExVisor を利用する場合，OS と ExVisor それぞれの設定値を揃えなければならない．現状ではこれは手作業で行われているため，設定ミスが入り込む余地が大きいうえに，作業工数の低減も難しい．実用化に向けては，これらの設定を自動で行うことが可能なコンフィギュレータが求められる．

**謝辞** 本研究の一部は，独立行政法人新エネルギー・産業技術総合開発機構(NEDO)の助成事業 P05020「情報家電用ヘテロジニアス・マルチコア技術の研究開発」の支援を受けて行われた．

## 7. 参考文献

[1]竹居 智久: 「トヨタの次世代車載情報端末基本アーキテクチャが明らかに」 日経エレクトロニクス 2008.1.28 pp.12-13

[2] Jeremy Sugerman et al:” Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor”,USENIX 2001 ,pp1-14

[3] Rich Uhlig et al: “Intel Virtualization Technology”, IEEE COMPUTER, Vol. 38, No.5, May 2005, pp48-56

[4] 高橋 明生 他: 「組込み向けマルチコア対応ドメイン分離技術」 FIT, 2008. Oct. pp.249-250

[5] Tohru Nojiri et al: "Domain Partitioning Technology for Embedded Multicore Processors”, IEEE MICRO, Vol.29, No6, Nov 2009, pp7-17

[6] Y.Yuyama et al.:”A 45nm 37.3GOPS/W Heterogeneous Multi-Core SoC”, Proc. of IEEE International Solid State Circuits Conference (ISSCC2010), Feb., 2010.

[7] http://opencv.willowgarage.com/wiki/

[8] J.E. Smith and R. Nair, ”Virtual Machines: Versatile Platforms for Systems and Processors,”' Morgan Kaufmann, 2005.

[9] 廣池 敦 他:「類似画像検索システム EnraEnra ～大規模画像アーカイブのための検索プラットフォーム～」 第 15 回画像センシングシンポジウム, Jun., 2009.