システムソフトウェアと
オペレーティング・システム 87－8
（２００１．６．２８）

組み込み用 OS『開聞』の MIPS プロセッサへの移植

堀口　努†　萱嶋　志門†　並木　美太郎†

東京農工大学大学院工学研究科†

近年，組込みシステムでもソフトウェアの規模が大きくなってきており，プログラムの再利用性が要求されるようになった．しかし,組込みシステムでは様々な CPU が用いられており，例外処理方式などの違いが組込み用 OS の再利用性を低下させる要因となっている．本論文では, PowerPC403GA で動作していた組込み用 OS『開聞』を MIPS プロセッサに移植することにより，組込みシステムの OS 本体とアプリケーションの再利用性を向上するための検討を行った．『開聞』は，タスク管理部，割込み管理部，システムコール部から構成され，CPU 依存部分と独立部分が混在する．タスク管理部では主に，タスク生成時における TCB のエントリポインタや，ステータスレジスタ等の各種パラメータ設定処理，タスクのスタック確保処理等を MIPS 用に改変した．また, MIPS プロセッサの例外処理を，例外ベクタと例外ハンドラを対応付け，例外ベクタテーブルで管理することで，ユーザによる性能トレードオフの選択を容易にした．移植した結果,例外管理部のシステムコールは CPU 依存であり，タスク管理部のシステムコールに移植性を持たせることができた．ソースコードの 59.6%が CPU 独立部分であり，40.4%が CPU 依存部分であった．また，CPU 依存部分の 42.0%がアセンブリ言語であった．

Porting "KAIMON" Embedded Operating System to the MIPS Processor Architecture

Tsutomu Horiguchi†　Shimon Kayashima†　Mitarou Namiki†

Tokyo University of Agriculture and Technology†

In recent years, as softwares are more complex and larger scales in embedded systems, they are demanded to be portable. But many kinds of processors are used in enbedded sysutems and it causes to lost portability of embedded operating systems. In this paper, Porting "Kaimon" to the MIPS processor and considering how to improve portability of embedded OS and its aplications. "Kaimon" is composed of 3 parts which are task management, interrupt management and systemcalls. In task management, TCB parametors and securing stack managements were need to exchange. Managing MIPS interrupt by InterruptVectorTable made easy of selecting the paformance trade-off. A result of porting to MIPS processor, systemcalls of interrupt managements depend on CPU. But systemcalls of task management are indipendent of CPU. 59.6% of source code is depend on CPU and 40.4% is independent of CPU. And 42.0% of a part of depending on CPU is written in assembly language.

1 はじめに

近年，組込みシステムでもソフトウェアの規模の拡大とともに，プログラムの再利用性が要求されるようになった．組込みシステムは，様々なアーキテクチャのプロセッサが用られているが，アーキテクチャごとに例外処理方式やレジスタセットをはじめ，様々な違いがある．組込み用 OS『開聞』は当初,PowerPC403GA プロセッサ用に設計され,プログラマが性能トレードオフの設計が容易という特徴を持つ．『開聞』を多くの種類の CPU で稼働する移植可能な OS にするために，本論文では MIPS プロセッサで『開聞』を実際に動作させ，組込みシステムのアプリケーションと OS 本体の再利用性を向上させるための検討を行う．

2 『開聞』の概要と移植上の問題点

MIPS へ移植する際に，『開聞』の設計方針を損なわないようにする考慮する必要があった．まず，『開聞』の概要を示す．そして,実際に移植する際に生じた問題点を述べる．

2.1 『開聞』の設計方針

『開聞』の設計方針を次に示す．

(1)必要最小限の機能を持つ

ハードウェアの抽象化レベルに対する考察を行うことを目的とし，提供する機能を限定する．

(2)ハードウェアの抽象化のレベルを下げることでオーバヘッドを削減する．

ハードウェアの性能を最大限に引き出すことを目指す．

(3)性能制約に応じて,仮想化された機能を選択するインタフェースを提供する．

システム開発者が性能制約を考えて，必要な機能を選択可能にするために，仮想化のレベルごとのインタフェースを設計する．

移植する際に，上記の設計方針を損なわないように，考慮した．

2.2 『開聞』の機能の全体構成

『開聞』は，大きく分けて次の三つの部分に分けることができる．構成図を図 2.1 に示す．

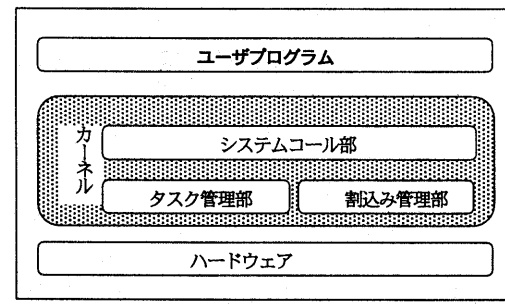

図 2.1 『開聞』の全体構成

(1)タスク管理部

タスクの生成，起床，休眠，消滅などの操作を行う．

(2)システムコール部

ユーザプログラムが OS の機能を使用するためのインタフェースである．

(3)割込み管理部

割込み要因を特定し，割込みハンドラへ処理を移行させる．従来 OS が行っていた処理をユーザプログラムでも実行させるために，仮想ベクタテーブルを実装しており，システムコールを介して，ユーザプログラムから変更可能である．

(1)～(3)のすべてに，ハードウェア依存部分と，独立部分が混在しており，明確に切り分けられてないことが移植上問題となった.次節では，MIPS に移植する際に生じた問題点を明確にする．

2.3 割込み管理部における移植上の問題

『開聞』の構成要素の中で，ハードウェアを直接管理する部分が割込み管理部である．本節では,『開聞』の割込み管理部を移植する際に生じる問題点を示す．

(1)割込みベクタテーブル

『開聞』の割込み処理を，割込み要因と割込みハンドラを結びつけた表で管理しており，この表を割込みベクタテーブルという．『開聞』では，PowerPC の割込みを三種類の割込みベクタテーブルで管理している．しかし，『開聞』によるハー

ドウェアの仮想化レベルが低いため，割込みベクタテーブルがハードウェア依存となっている．PowerPC の外部割込みを管理するために，外部割込みベクタテーブルが実装されているが，外部割込みという概念はMIPS にはない.また，割込みベクタテーブルでは，割込み要因を割込み ID として管理しているが，割込みIDが,PowerPC アーキテクチャに依存している．

(2)システムコール

『開聞』では，割込みベクタテーブルをユーザに開放することにより，性能トレードオフの選択を容易にしており，『開聞』の特徴的な機能である.しかし，(1)で述べたように，割込みベクタテーブルがハードウェア依存であるため，それらを設定するシステムコールのユーザインタフェースがCPU 依存になっている．実装面から見ると，メモリ上の例外ベクタへ直接命令を書き込むため,MIPS 用に改変する必要があった．

(3)割込み要因特定処理

割込みベクタテーブルは，割込み要因特定処理によって結び付けられている．要因特定は，完全に CPU に依存している．PowerPC では，割込みベクタや各種レジスタを用いて要因特定を行う．割込みベクタの割当て方は，PowerPC と MIPS では異なるので，移植上問題であった．

割込みベクタテーブル,割込み要因特定処理,システムコールは,互いに依存しており，ある部分を改変すると，他の部分も改変する必要がある.『開聞』は，割込み処理の仮想化レベルが低く，PowerPC アーキテクチャを完全に吸収していない．このことがMIPS へ移植する際に問題となった．

2.4 タスク管理部における移植上の問題

タスク管理部における移植上の問題点は，『開聞』の仮想化レベルを引き上げることによって，解決する問題ではなく，アーキテクチャの差異が原因で，必然的に生じる問題である．当初，改変すべき点が不明確であったので，移植を通じて明確にすれば，OS 本体の移植性を向上することができる．そこで，本節では，タスク管理部を移植する際に，必然的に変更を加えなければならなかった点をまとめる．

2.4.1 タスク管理

『開聞』では，タスクをシステム上に生成する時に次に示す処理を行う．

(1)新たに生成する TCB のメモリ領域を確保する．
(2)TCB を待ち行列の最後に接続する．
(3)TCB パラメータの初期設定を行う．

移植時に(3)が問題となった．移植時に問題となるパラメータを次に示す．

(a)スタックポインタ
(b)タスクのエントリポインタ
(c)ステータスレジスタの初期状態

スタックポインタ(a)は，使用するコンパイラの差異による汎用レジスタへの機能割当て方の相違が問題となる．

上記(b)(c)は，CPU アーキテクチャ依存によって生じる問題である．(b)のエントリポインタは，MIPS では，設定すべきレジスタの種類や機能が異なるので変更する必要があった．

ステータスレジスタ(c)に関しては，MIPS にも同名称のレジスタがあるが，初期設定値がことなる．(a)(b)(c)の部分はどれも，レジスタに対して直接操作を行うため，移植時に問題となった．

2.4.2 TCB(タスクコントロールブロック)

『開聞』はタスクを TCB で定義し，管理している．移植上の問題点となったのが，TCB のコンテキストである．『開聞』のコンテキストとは，レジスタのことである．PowerPC と MIPS が実装しているレジスタの詳細は，第 3 章で述べる． MIPS に『開聞』を移植するためには， PowerPC と MIPS アーキテクチャの違いを考慮しなければならなかった．そこで，次章では，両者のアーキテクチャの概要について述べる．

3 PowerPC と MIPS アーキテクチャ

OS 本体の移植時に，アーキテクチャの違いを考慮する必要がある．本章では，PowerPC と MIPS アーキテクチャについてまとめる．

3.1 PowerPC の割込み処理

PowerPC では, 割込みが発生すると, 割込みごとに, 固有の割込みベクタへ処理が移る. PowerPC の場合, 割込みベクタへ分岐するまえに, MSR の内容と, 割込みの起きた命令のアドレスをSRRに保存する. 割込みに応じて割込みハンドラを起動する. 割込みハンドラからは, rfi 命令により元のプログラムへ復帰する. 割込みハンドラを終了するときには, 保存しておいた MSR の値を戻し, 元のアドレスへ制御が移る. 割込み処理の手順を図 3.1 にまとめる.

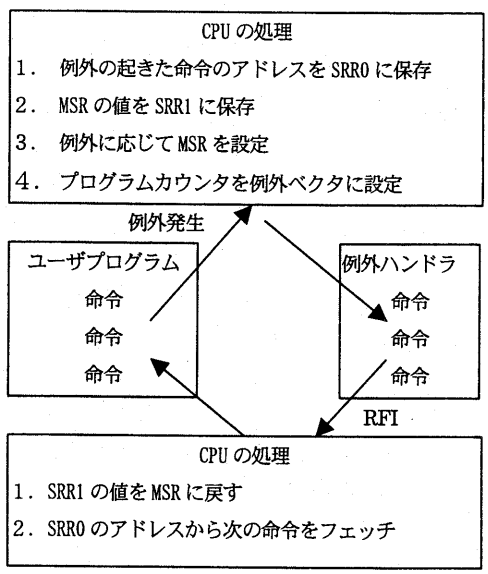

図 3.1 PowerPC の例外処理の手順

3.2 MIPS の例外処理

PowerPC の割込みを MIPS では、例外という。そして、例外処理の一部として割込みが定義されている。本節では, 『開聞』 で使用頻度が低いリセット例外やTLB不一致例外については述べない. 例外が発生すると, まず, 原因レジスタが設定され, EPC に PC がセットされる. そして, PC に例外ベクタが設定され, 例外ハンドラに処理が移行する. 例外ハンドラからは, ERET 命令で復帰する. 例外ハンドラを終了するときには, ステータスレジスタの, EXL がクリアされる. 例外処理の手順を図 3.2 にまとめる.

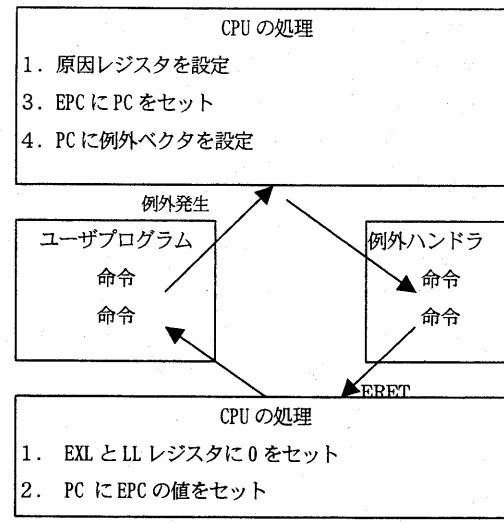

図 3.2 MIPS の例外処理の手順

3.3 例外ベクタ

PowerPC では, 割込みごとに割込みベクタを割り当て, 外部割込み以外の場合, 割込み要因を割込みベクタで判別する方式ことができる. PowerPC の割込みベクタを表 3.1 と表 3.2 にまとめる.

表 3.1 PPC 例外ベクタ 1

	ベクタ
クリティカル	0x0100
マシンチェック	0x0200
保護違反	0x0300
外部割込み	0x0500
アライメントエラー	0x0600

表 3.2 PPC 例外ベクタ 2

	ベクタ
プログラム	0x0700
SC	0x0C00
PIT	0x1000
FIT	0x1010
WDT	0x1020
デバッッグ	0x2000

一方, MIPS の場合, 例外ベクタは 3 種類なので, 例外の種類を判別するために, レジスタを参照する必要がある. 表 3.3 に MIPS の例外と例外ベクタの関係をまとめる.

表 3.3 MIPS 例外ベクタ

	ベクタ
コールドリセット, ソフトリセット, NMI	0xBFC0 0000
TLB 不一致 EXL=0	0x8000 0000
その他	0x8000 0180

例外ベクタの仕様の差異は, ベクタテーブルの実装方式や, 割込み要因特定処理, システムコールなどに影響を及ぼすので, 割込み管理部の構築時に考慮する必要がある.

3.4 レジスタ

はじめに, 『開聞』が保存している PowerPC のコンテキストを示す. そして, MIPS が実装しているレジスタについてまとめる.

3.4.1 PowerPC のレジスタセット

PowerPC403GA は, 104 個レジスタを実装しており, すべて格納すると, 416 バイトメモリを消費する. 『開聞』では, 39 個のコンテキストを退避することで, メモリ消費を 312 バイトに抑えた. 『開聞』のコンテキストを表 3.4 にまとめる.

表 3.4 PowerPC 版『開聞』のコンテキスト

分類	細目	レジスタ名	機能
ユーザレベル	固定小数点ユニット	GPR0～GPR31	汎用レジスタ
		XER	キャリ, オーバフロー
	分岐ユニット	CR	演算結果
		LR	サブルーチンの戻り先番地
		CTR	分岐とループ
特殊用途	例外処理	SRR0	非クリティカル割込み
		SRR1	コンテキストの保存
		SRR2	クリティカル割込み
		SRR3	コンテキストの保存

3.4.2 MIPS のレジスタセット

MIPS には, CPU レジスタと CP0 レジスタがある. CPU レジスタをすべて退避するためには, 536 バイトのメモリ領域を要する. さらに, CP0 には, 例外処理や, メモリ保護のためのレジスタが32個実装されており, すべて退避すると, 256 バイトメモリを消費する. CPU と CP0 のレジスタをすべて退避すると, 792 バイトメモリを消費する. 『開聞』では, 資源消費を抑えるために, 必要なコンテキストのみ退避する必要があった. コンテキストの詳細は 4 章で述べる.

以上に, PowerPC と MIPS アーキテクチャを概説した. 次章では, アーキテクチャの違いを考慮し, MIPS に移植した『開聞』の詳細を示す.

4 MIPS 版『開聞』の実現

本章では, MIPS に移植した『開聞』について述べる. 例外管理部では, ベクタテーブルの実装方法, 例外要因特定処理を中心に述べる. タスク管理部では, 改変が必要であった部分を中心に述べる.

4.1 例外管理部

本節では, MIPS 版『開聞』の例外管理部について述べる. まず, 例外が発生してから, 例外ハンドラへ処理が移るまでに『開聞』が行う処理を述べる.

4.1.1 MIPS 例外管理方式概要

例外が発生し, 例外ベクタに処理が移行すると, 対応するハンドラへジャンプする. 実行速度を重視する場合は, 通常この時点で, 『開聞』が行うコンテキスト退避/復元を, ユーザ側で行うことができる. 『開聞』がコンテキスト退避処理を行った場合, 例外要因特定処理を行い, さらにベクタテーブルを参照する. MIPS における例外管理の概要を図 4.1 にまとめる.

4.1.2 MIPS の例外ベクタテーブル

PowrPC では三つのベクタテーブルで割込みを管理していた. しかし, PowerPC と MIPS では, 割込み処理のアーキテクチャが異なるので, MIPS で三つのベクタテーブルで管理すると, 例外要因の判別方法が複雑になり, ユーザが混乱すると考えた. そこで, MIPS の例外を四つのベクタテーブルで管理することで, ユーザが性能トレードオフの選択を可能にした. 次に, ベクタテーブルの解説を示す.

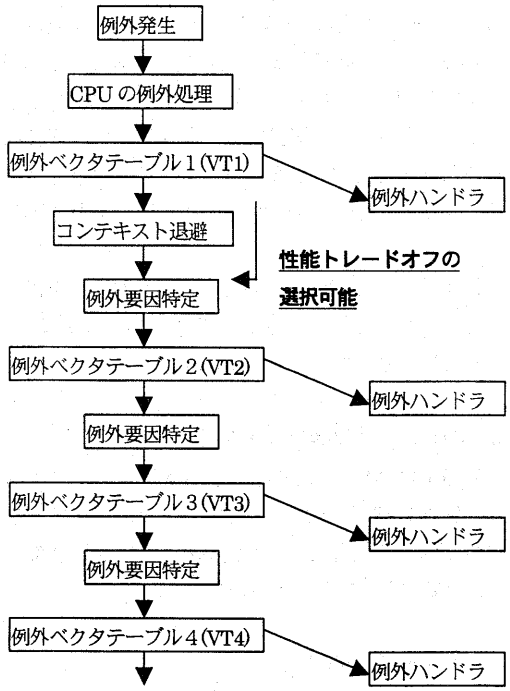

図 4.1 MIPS の例外管理

(1)VT1

3. 3 節で示した, 三つの例外ベクタと, 例外ハンドラの対応をこのテーブルで管理している. コンテキスト退避処理を『開聞』が行うか否かをユーザは VT1 を設定することで選択可能である.

(2)VT2

『開聞』にコンテキスト退避処理を行わせた場合, このベクタテーブルを参照する. 例外ベクタと例外ハンドラの対応を管理している.

(3)VT3

その他の例外用ベクタテーブルである. コンテキスト退避後, 通常このベクタテーブルを参照することになる.

(4)VT4

割込み例外用ベクタテーブルである. 発生した割込み種類を特定した後, このベクタテーブルを参照する.

次節では, VT を結び付けている例外要因特定処理の詳細を述べる.

4.1.3 例外要因の特定方法

本節では, MIPS の例外要因の特定方法について述べる.

(1)コールド・リセット, ソフトリセット, NMI 例外の場合(例外ベクタ=0xBFC0 0000)

ステータスレジスタの ERL ビットが 1 であり, かつ, 自己診断ステータス領域の SR ビットが 1 であるかによって判別する. ERL ビットは, エラーが生じた場合 1 にセットされる. SR ビットはソフトリセットまたは NMI 例外が発生したときに 1 にセットされる.

(2)TLB 不一致例外(例外ベクタ=0x8000 0000)

原因レジスタの例外コード領域に TLBL または TLBS がセットされている場合, TLB 不一致例外であると解析する.

(3)その他の例外(例外ベクタ=0x8000 0180)

上記以外の場合その他の例外であると判別する. その他の例外は, すべて例外ベクタが同じなので, PowerPC のようにベクタで原因を特定することができず, さらに, 原因レジスタを参照することで例外を判別する. MIPS 版『開聞』では, 原因レジスタの例外コード領域を参照することで例外原因を特定している. 割込み例外の場合, さらに, 原因レジスタの IP 領域を参照する. 各種例外の原因を追求したら, 例外ベクタテーブルを検索し, 例外ハンドラへ処理が移行する.

4.2 タスク管理部

タスク管理部は, アーキテクチャの差異による, 必然的な改変部分が, 移植上問題であった. それは, TCB のコンテキスト とタスク生成機能におけるパラメータ設定処理であった. まず, TCB における改変部分を明確にする.

4.2.1 TCB

本節では, MIPS 版『開聞』で変更を加えた TCB コンテキストについて述べる. 『開聞』は, 浮動小数点演算や, メモリ管理などをサポートしていないので全レジスタを退避する必要はなく, タスクごとに切り替える必要のある 46 個のレジスタを, 『開聞』が退避するようにし, 360 バイトのメモリ消費に抑

えた. MIPS 版『開聞』のコンテキストを表 4.1 に示す.

表 4.1 MIPS 版『開聞』のコンテキスト

分類	レジスタ名	消費メモリ
CPU	汎用×32, 整数乗除算結果上位 DW 格納×1, 整数乗除算結果下位 DW 格納×1	272byte
CP0	コンテキスト, BadVAddr, カウント, 比較, ステータス, 原因, EPC, ウォッチ Lo, ウォッチ Hi, X コンテキスト, エラーEPC	88byte
	計	360byte

4.2.2 タスクの生成

本節では, 移植する際に変更した, タスクの生成処理について述べる. タスクの生成では, タスクに必要なスタック領域や TCB を確保し, 指定されたタスクを使用可能にする. MIPS に移植する際に, 次のパラメータの設定に関して変更を加えた.

(a)エントリポインタ

例外から復帰する際に, MIPS では, EPC レジスタのアドレスから再開される. したがって, コンテキストのエントリポインタは EPC に格納するようにした.

(b)スタックポインタ

生成されたタスクのスタック確保のために, スタックポインタを GPR29 に設定するようにした.

(c)ステータスレジスタ

MIPS のステータスレジスタの初期値を 0x34000000 に設定するようにした.

5 『開聞』の移植結果

本章では, アプリケーションの移植性について考察する. また, カーネルサイズや, 実行性能について定量的な比較考察を行う.

5.1 アプリケーションの移植性の考察

MIPS で移植性を確保することができた, タスク管理部のシステムコールを表 5.1 にまとめる. タスク管理部では, ハードウェアの差異を, 『開聞』が吸収していることがわかる. OS 本体については, システム依存のパラメータなどの必然的な改変をすればよい事がわかり, 依存部分が明確になった. そして, タスク管理部のシステムコールを用いたアプリケーションは, システムの違いを考慮する必要がない. 次に, MIPS 版『開聞』の例外関連のシステムコールについて表 5.2 にまとめる.

表 5.1 移植性があるシステムコール

名称	機能
CreateTask	タスクを生成
DeleteTask	タスクを削除
SuspendTask	タスクを休眠
ResumeTask	タスクを起床
P	セマフォの P 命令
V	セマフォの V 命令

表 5.2 MIPS 版『開聞』の例外関連システムコール

名称	機能
ChCpuSigVecTable	CPU 独立ベクタテーブを変更する
ChExpVecTable	例外ベクタテーブルを変更する
ChIntVecTable	割込みベクタテーブルを変更する
ChMemVecTable	CPU 依存ベクタテーブルを変更する

例外管理部は, 仮想化のレベルが低く, CPU の差異を吸収しておらず, システムコールは, CPU 依存である. SH4 にも移植が行われ, 三つのベクタテーブルを実装しているが, すべて CPU 依存となっており, AP の移植性を確保しておらず, 仮想化のレベルが低いことを示している.

5.2 カーネルサイズ

コンパイル後のカーネルサイズを, 表 5.3 にまとめる.

表 5.3 カーネルサイズ

CPU	サイズ(byte)
MIPS	45808
PowerPC403GA	66048

PowerPC 版の方が, 20240 バイト大きい. MIPS 版のソースコードの 59.6%が CPU 独立部分であり, 主にタスク管理部であった. また, 40.4%が CPU 依存部分であり, 主に割込み管理部であった. CPU 依存部分のうち, 42%がアセンブリ言語であった.

5.3 実行性能の計測

システムコールとコンテキスト退避/復元実行クロック数について, 10 回測定を行い, 最悪値を抽出した. システムコールについて表 5.4 にまとめる.

表 5.4 システムコール実行クロック

	PowerPC	MIPS
Null_SC	345	155
CreateTask	739	277
ResumeTask	436	257
SuspendTask	410	290
DeleteTask	425	123
P	402	133
V	413	291
ChCpuSigVecTable	395	193
ChMemVecTable	402	226

最悪値はすべて, MIPS 版の方が小さくなっている. 当初, 『開聞』は PowerPC 用に設計されていたが, MIPS でも, 性能を確保することができた.

表 5.5 にコンテキスト処理についてまとめる.

表 5.5 コンテキスト処理実行クロック

	PowerPC	MIPS
コンテキスト退避	109	59
コンテキスト復元	65	55

PowerPC では, コンテキスト退避/復元時に, フレームポインタの作成処理を行っている. MIPS ではこの処理を省略し, クロック数を減少させた.

6 おわりに

『開聞』の MIPS への移植を通じて, OS 本体とアプリケーションの再利用性について考察を行った. タスク管理部は, TCB のコンテキストとタスク生成処理が CPU 依存であった. 割込み管理部は仮想化のレベルが低く MIPS 用に再設計する必要があった. MIPS 版では, 例外要因特定処理と四つのベクタテーブルを作成し, 性能トレードオフ選択機能を実現した. タスク管理部のシステムコールは, アプリケーションの移植性を持たせることができたが, 例外管理部のシステムコールは, 仕様が CPU 依存であった. また, MIPS 版『開聞』の性能を, PowerPC 版と定量的な比較評価を行い MIPS で性能を確保できることを確認した.

関連研究として, ITRON がある. ITRON では, ハードウェアに特化した実装を行えるような仕様策定を行っている. ITRON の割込み管理は, ISR の起動方法が実装依存なので, ISR の移植性が高い実装になるとはいえない.

今後の課題として, 『開聞』にリアルタイム性を持たせる事が挙げられる. 現在, 周期タスクとスケジューラの観点から研究を進めている. また, 複数の組込みシステムが互いに連携して処理を行うために, 『開聞』にネットワーク機能を実装する研究も進められている.

参考文献

[1]緒方 正暢, PowerPC403GA 搭載 CPU ボード徹底解説, Interface, CQ 出版, pp. 142-pp. 152, June. 1997.

[2]PowerPC マイクロプロセッサ・ファミリプログラミング環境, May. 1996

[3]Gerry Kane 著, 前川 守監訳, mips RISC アーキテクチャ-R2000/R3000-, 共立出版, Oct. 1992

[4]MIPS TM , VR4305 TM , VR4310 TM 64 ビット・マイクロプロセッサ ユーザーズ・マニュアル, NEC, March. 1999.

[5]監修 坂村 健, μITRON4.0 仕様, トロン協会 ITRON 部会, 1999