組込みシステム仮想実行環境の拡張

山本 整†, 大谷 寿賀子††, 落合 真一†

組込みシステム開発において，ハードウェアとソフトウェアを並列開発するために仮想実行環境が必要とされてきている．これまでに我々は，オープンソースの仮想実行環境である QEMU を利用してルネサス製組込み用途 CPU である RX ファミリの仮想実行環境を実現してきた．これに対して将来的な拡張を見込み，対称型マルチプロセッサ(Symmetric Multiprocessing：SMP)構成を可能とする拡張開発を実施した．SMP 拡張開発においては，SMP 動作させるために必要となる機能を検討した後に，設計，実装し，評価を行った．その結果，QEMU により実存しないハードウェア機能を模擬して，その上でソフトウェア開発が可能であることを確認できた．本論文では，RX アーキテクチャ対応 QEMU の SMP 拡張における実装と評価について述べる．

A prototyping of virtual execution environment for embedded systems

Hitoshi Yamamoto†, Sugako Ohtani†† and Shinichi Ochiai†

Virtual execution environments are required for embedded system development, because it is necessary to develop hardware and software concurrently. To enable starting software development before the hardware comes out, we developed the virtual execution environment for RENESAS RX CPU family using QEMU. In addition, we have developed SMP (Symmetric Multiprocessing) function in the QEMU future CPU extension. As a result, we confirmed that we can develop software, using the extended QEMU. In this paper, we present design and implementation of the QEMU that emulates the SMP function of RX CPU.

1. はじめに

組込みシステム開発において，ハードウェアとソフトウェアを並列開発するために仮想実行環境が必要とされてきている．仮想実行環境として，オープンソースの QEMU™[1]が存在する．QEMU は，模擬できる CPU アーキテクチャや，QEMU 自身が実行されるホストマシン環境が多様であるという特長を有する．近年では，QEMU を組込みソフトウェアの開発環境の一部として提供する例や，Android™ などオープンソースプロジェクトの参照用仮想ボードとして利用する[2]などの採用事例が増加している．我々は QEMU に対して，ルネサス製の組込み用途 CPU である RX ファミリ[6]へ対応する開発を実施している．この RX ファミリ対応の QEMU を RX-QEMU と呼ぶ．

ハードウェアとソフトウェアを並列開発する場合，ハードウェア仕様に基づき仮想実行環境を拡張する必要がある．ここで，近年の組込み用途 CPU においては，省電力と高性能を両立させるために対称型マルチプロセッサ(Symmetric Multiprocessing：SMP)構成を可能とした製品が存在する．我々は将来的に RX ファミリにおいても SMP 構成可能な製品が開発されることを想定し，RX-QEMU に対して SMP 拡張開発を実施した．SMP 拡張開発では，まず SMP 構成を可能とするために必要となるハードウェア仕様を策定し，次に該仕様に基づき RX-QEMU を拡張し動作を確認した．

本論文では，まず 2 章で関連研究について，3 章で RX-QEMU について説明し，4 章で本研究の目的を説明した後に，5 章では RX-QEMU を SMP 構成可能とする仕様の検討結果，6 章にて設計について，7 章では評価内容とその結果を述べ，最後に8章にてまとめと

† 三菱電機株式会社 情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation
†† ルネサス エレクトロニクス株式会社 第一事業本部
1st Solution Business Unit, Renesas Electronics Corporation

今後の課題について記述する.

2. 関連研究

本章では, 本研究の関連研究について述べる.

QEMU には, フルシステムエミュレーションと, ユーザモードエミュレーションの 2 つのエミュレーションモードが存在する. フルシステムエミュレーションは, ターゲット CPU と I/O デバイスも含めたハードウェア全体の模擬を行う. ユーザモードエミュレーションは, 一つのターゲット用プロセスを QEMU 経由で Linux®や BSD 上で動作させ, 該プロセスにより実行されるシステムコールをシステムコールインタフェースレベルで QEMU が動作しているホストマシンのシステムコールへ変換して実行する. この時, デバイスやターゲット CPU の特権命令はエミュレーションしない. 本研究では, 組込みシステム仮想実行環境として QEMU のフルシステムエミュレーションを利用する. フルシステムエミュレーションを使用した仮想実行環境としては, Android Emulator[2]が有名である. また, 組込み用途 CPU に対する QEMU の適用については, ルネサス製 V850 用の QEMU を開発し評価した事例[3]が存在する. V850 用 QEMU 開発は, RX-QEMU の目的と同様に組込みシステムのハードウェアとソフトウェアの並列同時開発を狙いとしている.

3. RX-QEMU

本章では, 本研究において拡張のベースとした RX-QEMU について説明する. RX-QEMU は, 複数の種類のボードを模擬することができ, そして実際の CPU と同等の性能を達成している.

3.1. 仮想ボード

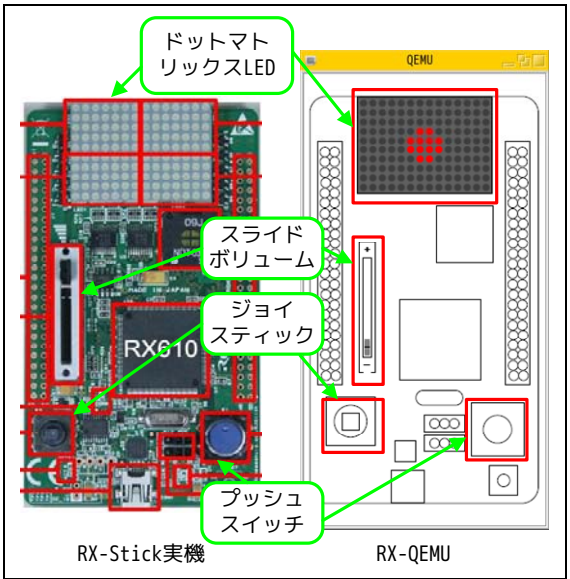
RX-QEMU が模擬する仮想ボードについて表 1へ示す. モデル名は, RX-QEMU 起動時に仮想ボードを指定する名称である. ボード名は, 実際のボードの名称である.

表 1 : RX-QEMU 仮想ボード一覧

#	モデル名	ボード名	概要
1	rxstick	RX-Stick	Renesas RX-Stick for RX610
2	rskrx610	RSKRX610	Renesas Starter Kit for RX610
3	rdkrx62n	RDKRX62N	Renesas Demonstration Kit for RX62N

表 1の各ボードは, RX ファミリの CPU である RX610 や RX62N のプロモーションまたはトレーニング用途のボードである. RX-Stick[7]には RX610 が搭載されており, ドットマトリックス LED による表示やジョイスティックによる入力が可能である. RSKRX610 は RX610 が搭載されており, 内蔵シリアルインタフェース(SCI)にて外部との接続が可能である. RDKRX62N には RX62N が搭載されており, 内蔵イーサネットコントローラ(ETHERC)により外部と接続でき, LED, LCD が搭載されている.

RX-QEMU の一例として, RX-Stick の実機と RX-QEMU による模擬ボード rxstick の外観を図 1へ示す. RX-QEMU におけるジョイスティック, スライドボリューム, プッシュスイッチの操作はマウスにより行うことができる. ジョイスティック, プッシュスイッチ, ドットマトリックス LED は RX610 内蔵の I/O ポート(PORT)へ, スライドボリュームは RX610 内蔵の A/D コンバータ(AD)へそれぞれ接続されている. RX-QEMU 上で RX-Stick 実機と同じプログラムを動作させることで, ジョイスティック, スライドボリューム, プッシュスイッチの操作に従い, RX-Stick 実機と同等の表示がドットマトリックス LED 上に表示される.



Renesas Electronics, RX-Stick User's Manual, p.4 の Figure4-1 を元に著者が手を加え作成した

図 1 : RX-Stick 実機と仮想ボード

3.2. 内部実装

RX-QEMU を実現するにあたり, RX 命令をホストマシン上で模擬する命令セットシミュレータ(Instruction Set Simulator：ISS)部, および各CPUが内蔵しているデバイスとボードに搭載されている I/O デバイスを模擬するデバイスモデルを開発している. 以下, RX-QEMU における命令セットシミュレータおよびデバイスモデルの概要について述べる.

3.2.1. 命令セットシミュレータ

RXファミリは90種類の命令を持つ. RXは出現頻度の高い命令の命令コードサイズを小さくすることで高いコード効率を実現している. オペコード長は, 5 ビットから24ビットまで10種類, オペランド数は0個から3個, アドレッシングモードは10種類存在する. 命令コードの数は191種類である. ISSでは, 命令デコード処理において 191 種類の命令コードから命令種別を特定, 命令変換処理においてホスト命令へ変換し Translation Block (TB)へ記憶する. 変換されたTBをTB実行処理により実行することでISSが実現される(図 2).

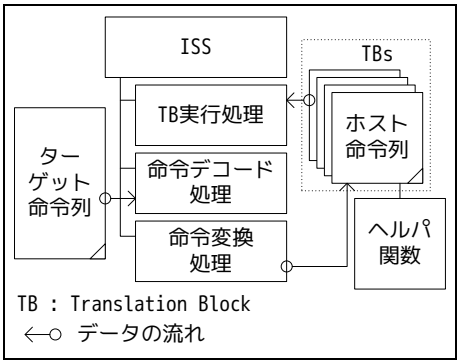


図 2 : QEMU 命令セットシミュレータ

命令デコード処理では, ターゲット命令を 1 バイトずつ読出し, 命令種別が一意に特定できるまで読出しを繰返す. RX 命令の場合は, 最短 1 バイト, 最長 3 バイト読み出すことで, 命令種別を一意に特定することが可能である. 命令変換処理では, QEMU の TCG Frontend Ops (TCG 関数) [5]を使用してRX命令の操作を記述する. ここで TCG (Tiny Code Generator)はQEMUのJITコード生成機構である. TCG 関数で記述されたターゲット命令は, TCG によりホスト命令に変換され, TB へ記録される. TCG 関数で記述が困難な命令については, C 言語を用いたヘルパ関数により実装することが可能である. その場合, 命令変換処理では, ヘルパ関数呼び出しを TB 上に記憶する. ターゲット CPU が TB 上の該アドレスを実行した場合に, ヘルパ関数が呼び出される.

3.2.2. デバイスモデル

RXファミリは, 組込み用途CPUであるため豊富な周辺デバイスを内蔵している. これらの中から, RX-QEMU にて模擬するデバイスとして, 実際のボード用に準備されているサンプルプログラムが使用するデバイスを選択して実装している. デバイスを模擬するデバイスモデルは, QEMU のデバイスフレームワークである qdev を利用して作成している (図 3).

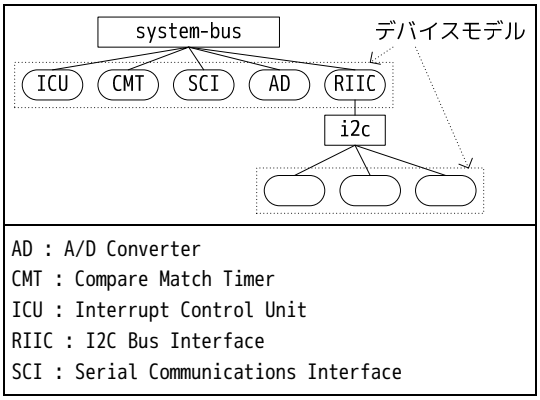


図 3 : QEMU デバイスモデル

qdev は, デバイスモデルを統一的に記述し管理するためのフレームワークである. qdev では, デバイスとバスを分離した形でモデル化している. 個々のデバイスモデルがバスモデルに接続される形で, システムのデバイス構成をツリー構造で管理できる. デバイスモデルをまたがる参照や操作は, qdev フレームワーク, あるいはバスモデルの提供するインタフェースを介して行う. デバイスは, ターゲット CPU のアドレス空間へマッピングされ, デバイスモデル側では該アドレスへの読出し, 書込み操作に従って呼び出されるコールバック関数を実装する.

3.3. 性能

RX-QEMU の性能について示す. ベンチマーク CoreMark[9]を用いてRX-QEMUの性能を測定した. 実機性能と比較した結果を図 4へ示す. CoreMark値は, 値が大きいほど性能が良い. ここで RX610 および RX62N の CoreMark 値は CoreMark の Web サイト[9]に公開されているスコアである. 両CPUともに100MHz動作であり, コンパイル環境は RX610 が GNU ツール環境, RX62N は IAR システムズ社製のツール環境である.

RX-QEMU は 3GHz の CPU を搭載したホストマシン上で動作させており，コンパイル環境は RX610 と同じく GNU ツールを使用している．RX-QEMU を動作させたホストマシン環境を表 2へ示す．実機である RX610 と RX62N の命令実行速度は同じであるため，それらの CoreMark 測定結果の違いはベンチマークのコンパイル環境の違いに起因すると考えられる．RX-QEMU の CoreMark 値は RX610 と比較して約 3.6 倍，RX62N と比較して約 2.7 倍である．次世代 RX ファミリの最高周波数は 240MHz であるが，単純に性能を周波数換算し比較した場合，次世代 RX ファミリに対しても RX-QEMU の性能の方が高いことがわかる．

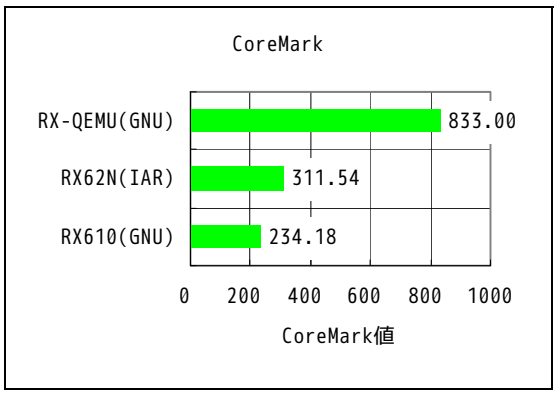


図 4 : CoreMark 結果

表 2 : ホストマシン環境

#	分類	諸元
1	H/W	CPU：Intel Core 2 Quad Q9650 3.00GHz メモリ：8GB
2	S/W	OS：Debian GNU/Linux 6.0.7 amd64 gcc：4.4.5 (Debian 4.4.5-8) binutils：2.20.1-system.20100303 glibc：Debian EGLIBC 2.11.3-4 qemu：qemu v1.2.0 クロス環境：KPIT GNURX v11.02

4. 研究の目的

本章では，本研究の目的について説明する．

これまで RX-QEMU において，実存するボードを模擬する機能を実現してきた．組込みシステムのハードウェアとソフトウェアの並行同時開発を行う場合，ハードウェア仕様に基づいて，仮想実行環境が模擬するハードウェアについても機能拡張する必要がある．I/O デバイスの追加程度であれば，前述のデバイスモデルを開発することで実現することが可能である．これに対して，本研究では CPU アーキテクチャの拡張を伴う，実存しない CPU を開発する場合においても RX-QEMU が仮想実行環境として利用できることの検証を目的とする．検証するために，RX ファミリにおいて実存しない SMP 構成可能な CPU を RX-QEMU 上で実現する．最終的には Linux のような SMP 動作可能なオペレーティングシステム(OS)の開発を SMP 拡張した RX-QEMU 上で可能とすることを目標とする．

5. 検討

本章では，RX-QEMU を SMP 構成対応するための機能仕様についての検討結果について説明する．

現状の RX ファミリには SMP 構成の CPU は存在しない．そのため，まず RX-QEMU における SMP 構成対応に必要な機能を検討した．我々はこれまでにも SMP に関する研究を実施している[8]．検討の結果，RX-QEMU の SMP 構成対応に必要な機能は以下の 4 点と考えた．実際の CPU においては，CPU 間のキャッシュ一貫性保証などの機能も必要であるが，RX-QEMU ではキャッシュを模擬しないため不要であると判断している．

(1) 排他制御機能
(2) CPU 間通知機能
(3) 複数 CPU 起動機能
(4) CPU 識別機能

QEMU において，新たなハードウェア機能を実現する方式として，2 方式考えられる．QEMU 自身を改造し実現する方法，もしくは QEMU とネットワークなどのインタフェース経由で接続されたデバイス模擬プログラムを拡張する方法である．ネットワーク接続されたデバイス模擬プログラムを使用する例としては[4]が提案されている．仮想ボードへ新たなデバイスを追加する場合は，どちらの方法でも実現が可能であると考える．本研究では，RX-QEMU の SMP 構成対応について検討しており，SMP 構成を実現するためには，CPU への命令追加など基本部分を拡張開発する必要がある．そのため，前述のインタフェースにより外部接続されたデバイス模擬プログラムを拡張する方法では実現が不可能である．従って SMP 対応開発は，RX-QEMU を改造して実現することとした．

5.1. 排他制御機能

排他制御機能は，各 CPU 上で動作している複数のソフトウェアからのアクセスが競合した場合に，問題となる共有資源へのアクセスを排他的に行うことを可能とする機能である．排他制御を実現するためには，一般的に変数を不可分操作するための命令(排他制御命令)を提供する必要がある．代表的な排他制御命令には Load-Linked / Store-Conditional 命令や，Compare-and-Swap 命令などが存在する．RX-QEMU では排他制御機能用命令として，LDL (Load-Linked) / STC (Store-Conditional)命令を追加する．LDL/STC 命令仕様概略をそれぞれ表 3，表 4へ示す．

表 3 : LDL 命令仕様概略

#	項目	説明
1	ニーモニック	LDL [Rsrc], Rdest
2	動作	LDL ビット付きロード LDLbit = 1; Rdest = *(int *)Rsrc;
3	機能	LDL ビットへ"1"を格納する． Rsrc で指定した番地のワードデータを，Rdest へ格納する． LDL ビットが"0"クリアされる条件 ・STC 命令実行時 ・RTE 命令実行時(割込み処理)

表 4 : STC 命令仕様概略

#	項目	説明
1	ニーモニック	STC Rsrc, [Rdest]

2	動作	LDL ビットクリア付きストア
		if (LDLbit == 1) {
		*(int *)Rdest = Rsrc;
		Rsrc = 0;
		} else {
		Rsrc = 1;
		}
		LDLbit = 0;

3	機能	LDL ビットが"1"の時，Rsrc を Rdest で指定された番地のメモリへ格納，Rsrc を"0"クリアする． LDL ビットが"0"の時，Rsrc に"1"を格納する． LDL ビットを"0"クリアする．

ここでは各命令の動作説明を容易化するために CPU 内部に LDL ビットが存在するとしている． LDL 命令にて LDL ビットを"1"に設定し，STC 命令にて LDL ビットが"1"の場合にストア動作を成功させる．STC 命令にて LDL ビットが"0"の場合はストア動作を行わない．LDL ビットの"0"クリア条件は，STC 命令もしくは RTE 命令が実行された場合である．RTE 命令は，割込み処理または例外処理の場合に実行される命令である．RTE 命令実行時も LDL ビットを"0"クリアすることにより，LDL 命令と STC 命令の間で，操作中の変数が割込みや例外に伴う処理により変更されることを防止する．LDL/STC 命令を使用したスピンロック関数 llsc_spin_lock()，スピンアンロック関数 llsc_spin_unlock()をリスト 1へ示す．

```
void llsc_spin_lock(int *lock)
{
    int tmp;
    __asm__ __volatile__ (
    "1:                 \n"
    "   ldl [%1], %0    \n"
    "   cmp #0, %0      \n"
    "   bne 1b          \n"
    "   mov #1, %0      \n"
    "   stc %0, [%1]    \n"
    "   cmp #0, %0      \n"
    "   bne 1b          \n"
    : "=&r" (tmp)
    : "r" (lock)
    : "memory"
    );
}

void llsc_spin_unlock(int *lock)
{
    *lock = 0;
}
```

リスト 1： スピンロック，アンロック関数

5.2. CPU 間通知機能

CPU 間通知機能は，ある CPU 上で動作しているソフトウェアが他の CPU 上で動作しているソフトウェアに対して，処理を依頼する場合に必要となる通知機能である．具体的な利用例としては，他の CPU からはアクセスできないリソース(CPU ローカルリソース)に対する処理を依頼する場合などがある．CPU 間通知機能として CPU 間割込み(Inter Processor Interrupt：IPI)機能を追

加する. RX-QEMU には, IPI 操作用に IPI 送信レジスタおよび IPI ステータスレジスタを搭載する. IPI 送信レジスタにより宛先の CPU 識別子を指定して割込みを送信することを可能とする. 割込みを受信した CPU は, IPI ステータスレジスタを読み出すことでどの CPU からの IPI を受信したかを判断可能とする. また IPI 受信時に動作を開始する IPI 受信ベクタを設ける.

5.3. 複数 CPU 起動機能

複数 CPU 起動機能は, 電源投入後やハードウェアリセット後のチップ起動時に最初に動作する CPU である Bootstrap Processor (BSP)から, BSP 以外の CPU である Application Processor (AP)を起動し, SMP 動作を開始する機能である. RX-QEMU 起動後は BSP がリセットベクタから動作し, BSP から IPI を送信し AP を起動させ, AP が IPI 受信ベクタから動作を開始する仕様とする. RX-QEMU では, AP となる CPU は起動後に IPI 受信待ち状態とさせる.

5.4. CPU 識別機能

CPU 識別機能は, チップ内で一意である CPU 識別子を得る機能である. 各 CPU 上で動作しているソフトウェア(OS など)が, どの CPU 上で自身が動作しているか判断できるようにするための機能である. RX-QEMU へ CPU 識別子レジスタを搭載し, ソフトウェアは該レジスタを読み出すことで自身が動作している CPU の識別番号を得ることを可能とする.

6. 設計

SMP 拡張におけるそれぞれの機能について設計した. 本章では, SMP 拡張における主要機能である排他制御機能の設計について説明する.

排他制御機能では RX-QEMU へ新たに LDL/STC 命令の命令模擬を追加している. RX-QEMU における LDL 命令模擬, STC 命令模擬の処理フローについてそれぞれ図 5, 図 6へ示す. LDL/STC 命令模擬では, LDL 命令と STC 命令を紐づけるために演算対象アドレスと該アドレスの変数値を使用する. CPU 毎のコンテキスト(env)へ LDL/STC 命令により操作されるアドレスを記憶する変数(env->lladdr)と該アドレスの値を記憶する変数(env->llval)を追加する. CPU 毎のコンテキスト(env)は, RX-QEMU 内で CPU 毎に準備するコンテキスト(CPURXState)であり, 汎用レジスタやシステムレジスタの情報を保持している.

図 5 : RX-QEMU LDL 命令模擬処理

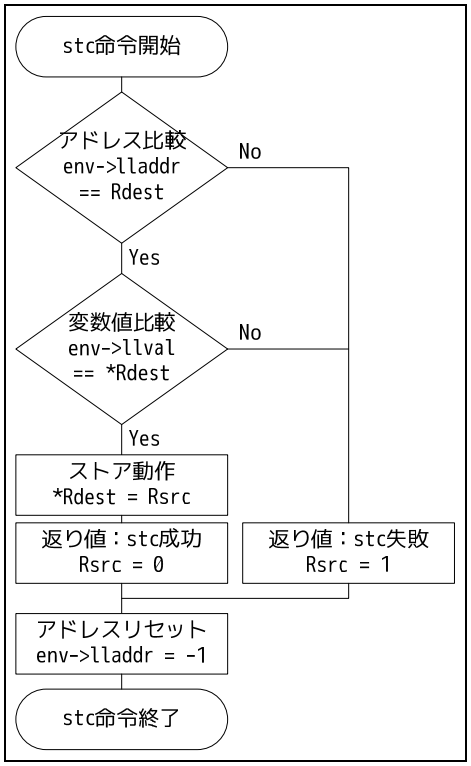


図 6 : RX-QEMU STC 命令模擬処理

LDL/STC 命令模擬では, まず LDL 命令実行時に演算対象アドレスを記憶(env->lladdr)し, さらに該アドレスから読み出した変数値を記憶(env->llval)する. その後の STC 命令実行時には, STC 命令の演算対象アドレスが記憶済のアドレス(env->lladdr)と等しく, さらに該アドレスを再度読み出した値が記憶済の変数値(env->llval)から変更されていない場合にのみ, ストア動作を成功させる. STC 命令実行後は記憶済のアドレ

ス(env->lladdr)を"-1"へリセットする.

また, QEMU では TB を数珠繋ぎにして連続実行することで高速化するチェイニングという手法が採られている. そのため, LDL/STC 命令模擬においてチェイニングが生じ, LDL/STC 命令模擬が連続実行され, CPU スケジューリングが停止してしまうことが想定される. 本研究では, この問題に対して QEMU 全体でチェイニングを無効化することで対処する. LDL/STC 命令模擬時のチェイニングを効率よく制御できるようにして, 性能劣化を防ぐことは今後の課題である.

7. 評価

SMP 対応拡張におけるそれぞれの機能についてテストプログラムを作成し, 評価を実施した. 本章では, 排他制御命令に関する動作評価を通じて, SMP 拡張した RX-QEMU 上で SMP 対応ソフトウェアの開発が可能なことを示す.

動作評価では, 排他制御機構を用いてクリティカルセクションを保護できることを確認した. 評価に用いたクリティカルセクションを以下に示す. ここで, count 変数は全 CPU からアクセスされる変数である.

(1) count 変数をテンポラリ変数へ読み出し
(2) テンポラリ変数を 1 インクリメント
(3) テンポラリ変数を count 変数へ書込み

SMP 模擬した RX-QEMU 上で, クリティカルセクションの前後をリスト 1にて示したスピンロック/アンロックにより囲んで排他制御した場合と, 排他制御なしの場合で, count 変数の値が期待値通りとなるかを評価した. ここで期待値は, CPU 数×ループ回数である.

SMP 拡張の動作検証構成を図 7へ示す. RX-QEMU により 4 個の RX CPU を動作させた状態で試験プログラム smp-test を実行する. そして, RX-QEMU の gdbstub へデバッガ(rx-elf-gdb)をリモート接続することで変数値を確認した. 試験プログラム smp-test ではクリティカルセクションを繰返し実行し, 排他制御ありの場合の count 変数値を count1 変数へ, 排他制御なしの場合の count 変数値を count2 変数へ記憶して終了する.

ここで, デバッガからはそれぞれの CPU をスレッドとして操作することができる. デバッガ操作の結果をリスト 2へ示す. gdb の print コマンドにて, 排他制御ありの場合の count1 変数, 排他制御なしの場合の count2 変数を表示している.

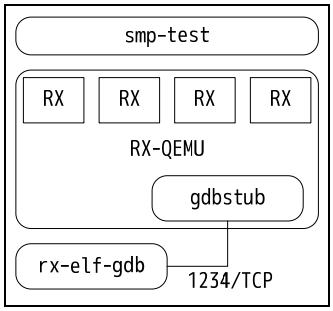


図 7 : SMP 拡張動作検証構成

```
$ rx-elf-gdb
<<snip>>
(gdb) target remote localhost:1234
(gdb) symbol-file ./smp-test
(gdb) continue
## Ctrl+C 入力で停止
(gdb) info threads
  4 Thread 4 (CPU#3 [running])
  3 Thread 3 (CPU#2 [running])
  2 Thread 2 (CPU#1 [running])
* 1 Thread 1 (CPU#0 [running])
(gdb) print count1
$1 = 40000000
(gdb) print count2
$2 = 36411357
```

リスト 2 : 排他制御命令動作検証

評価の結果, count 変数の初期値 0, CPU 毎にリードモディファイライトを 1,000 万回実行した場合に, 排他制御ありの場合(count1)は期待値通り 4,000 万(ループ回数×CPU 数)となるが, 排他制御なしの場合(count2)は, 約 3,600 万となり約 1 割クリティカルセクションの実行が不成功となることを確認した. これにより, RX-QEMU における排他制御命令の実装が正しいと判断した.

また, SMP 拡張した RX-QEMU が模擬する複数の CPU 上で動作するソフトウェアの操作を, デバッガ経由にてできることを確認した. これにより, SMP 拡張した RX-QEMU 上で SMP 対応ソフトウェアの開発が可能であると考える.

8. おわりに

本稿では, オープンソースの QEMU を利用するルネサス社製 RX ファミリの仮想実行環境 RX-QEMU について, 実際には搭載されていない機能を仮想実行環

境で実現可能かを検証するために，SMP 拡張を実施した．

今後評価すべき事項について述べる．現状では，RX ファミリの SMP 対応製品は存在しないが，SMP 拡張した RX-QEMU を用いて SMP 動作可能な OS の開発を実施し，該 OS が SMP 対応 RX ファミリ上で動作することを確認することが必要である．そして，本研究での評価は排他制御命令にとどまっているが，RX-QEMU の SMP 性能を評価することも必要と考える．

3.3節に示した RX-QEMU のベンチマーク結果により，次世代 RX ファミリの性能までは，現在の RX-QEMU でも対応可能であると考えている．ただし，更に高性能な RX ファミリが製品化されることも想定し，RX-QEMU の性能向上を狙うチューニングを行う．

参考文献

[1] QEMU，入手元(http://wiki.qemu.org) (参照 2013-05-28)
[2] Android Emulator，入手元 (http://developer.android.com/tools/help/emulator.html) (参照 2013-05-29)
[3] 尾崎 展典，中本 幸一，藪内 健二：QEMU を利用した V850 シミュレータの開発と評価，情報処理学会研究報告，Vol.2009-EMB-14 No.11 2009/7/24
[4] 中島 啓太，稗田 拓路，谷口 一徹，冨山 宏之：QEMU と SystemC を用いた NoC 向け仮想プラットフォームの開発，情報処理学会研究報告，Vol.2012-EMB-24 No.1 2012/3/2
[5] QEMU TCG Frontend Ops，入手元 (http://wiki.qemu.org/Documentation/TCG/frontend-ops) (参照 2013-05-29)
[6] RX ファミリ ユーザーズマニュアル ソフトウェア編，ルネサス エレクトロニクス (2011).
[7] RX-Stick User's Manual, Renesas Electronics (2010).
[8] 山本 整，高田 浩和：シングルチップマルチプロセッサへの Linux の適用，情報処理学会第 66 回全国大会
[9] CoreMark an EEMBC Benchmark，入手元 (http://www.coremark.org/) (参照 2013-05-28)