Vol.2013-SLDM-160 No.14
Vol.2013-EMB-28 No.14
2013/3/13

TLMu を用いた組込みマルチコアのシミュレーション

紅林修斗† 稗田拓路† 谷口一徹† 冨山宏之†

本研究では複雑化したマルチコア SoC に対応したシミュレーション環境を構築した。CPU 部に CPU エミュレータである QEMU を加工した TLMu を、その他のハードウェアに SystemC を用いることで、システムの詳細なモデリングができるソフトウェアシミュレーションを可能にしている。実験では 9 コアの SoC シミュレーション環境を構築し、その上で JPEG 変換プログラムのパイプライン処理をシミュレーションした。これにより、構築したシミュレーション環境の動作確認と性能評価を行った。

1. はじめに

近年では、1CPU での命令レベル並列処理の限界や、動作周波数の向上による消費電力の増大といった問題から、SoC (System on a Chip)でもマルチコア化が進んでいる。それに伴ってシステム全体の複雑度が増し、開発期間は延長してしまう。開発期間の延長は開発コストや製品の売り上げに悪影響を及ぼす。この問題に対してソフトウェアシミュレータは 1 つの有効な解決手段となり得る。ターゲットとなるアーキテクチャをシミュレータ上で構築し、その上でソフトウェアを実行させることで、ソフトウェア開発をハードウェアが完成する前から開始できる。これによりトータルの開発期間の短縮が可能となる。

本研究では、TLMu を用いてマルチコア SoC シミュレーション環境を構築した。TLMu とはオープンソースの CPU エミュレータである QEMU をシステムレベル設計言語である SystemC の CPU モジュールとして使用できるように加工したものである。これを用いることで SystemC によるハードウェアの詳細なモデリングと、QEMU によるソフトウェアシミュレートを同時に行うことが可能となる。

本論文の構成を示す。2 章では、SystemC と QEMU を用いたシミュレータの関連研究について述べる。3 章では，今回使用しているベース技術について述べる。4 章では，開発したシミュレーション環境について述べ、5 章では，それを用いた実験について述べる。最後に 6 章でまとめを述べる。

2. 関連研究

今回使用した TLMu 以外にも QEMU と SystemC を用いたシミュレーション環境が研究されている。2.1 節で NoC (Network on Chip)アーキテクチャを対象とした Naxim を、2.2 節で本研究と同じバス型のマルチコア SoC を対象とした Rabbit を紹介する。

2.1 Naxim

Naxim[1]は NoC 向けソフトウェアシミュレータである。各コアに QEMU、コア間のネットワーク部に SystemC を用いている。また QEMU と SystemC の通信には BSD ソケット通信を用いている。主にネットワーク部のシミュレーションを目的として開発されている。ネットワーク部以外は QEMU の内部で処理されており、高速なソフトウェアシミュレーションが可能となっている

2.2 Rabbit

Rabbit[2]はマルチコア SoC 向けソフトウェアシミュレータである。Rabbit は Naxim 同様、CPU 部を QEMU で、周辺部を SystemC でシミュレーションしているが、QEMU をそのまま使用せず、SystemC 上で使用できるように加工している点で Naxim とは異なっている。Rabbit ではオリジナルの QEMU に含有されていたキャッシュやメモリ等を SystemC で記述している。これにより、CPU のメモリアクセスやバスの挙動といった部分をより詳細にシミュレーションすることが可能となっている。また QEMU と SystemC に共有な時間軸を設定することで、QEMU と SystemC を同期している。

2.3 その他の研究

その他いくつかの研究で QEMU と SystemC を用いたシミュレーション環境について述べられている。文献[3]、[4]、[5]の QEMU-SC では Naxim のように、オリジナルの QEMU と SystemC とがホスト上で動作している。一方、文献[6]や[5]の QBOX では Rabbit のように、QEMU を SystemC 上で使用・管理できるように加工している。

3. ベース技術

3.1 QEMU

QEMU [7]は動的バイナリ変換を用いた ISS である。移植性に優れ ARM や x86 など多数のアーキテクチャに対応している。QEMU にはフルシステムエミュレーションモードとユーザーモードの動作モードがある。

フルシステムエミュレーションモードでは QEMU 上で OS を動作させることができる。OS の動作に必要な周辺機器も同時にエミュレートすることでこれを実現している。

† 立命館大学
Ritsumeikan University

Vol.2013-SLDM-160 No.14
Vol.2013-EMB-28 No.14
2013/3/13

ユーザーモードではホストとは異なるアーキテクチャの実行ファイルをホスト上で実行することができる。例えば、ARM 用にコンパイルされた実行ファイルを、x86 のホスト上で動作させることができる。

3.2 SystemC

SystemC[8]はシステムレベル設計言語であり C++のライブラリとして提供されている。RTL (Register Transfer Level)から SAM (System Architecture Model)に至るまで、広い抽象度のレベルで設計が可能で、近年の複雑化した電子システムの設計に対して高い生産性が期待できる。

C++の拡張である SystemC は専用のヘッダーファイルとライブラリを用いる以外は、C++と同じようにソースをコンパイルすればよい。出力された実行ファイルを実行することで、設計したシステムのシミュレーションを行うことができる。

3.3 TLMu

TLMu[9]は SystemC 用の CPU モデルライブラリである。QEMU の CPU に該当する部分が SystemC のモジュールとして使用できるようにラッピングされ、それ以外の周辺機器は SystemC で記述する。故に、QEMU と SystemC との関係性は Rabbit に近い。図 1 に TLMu を用いたサンプルのアーキテクチャを示す。図中の CPU が TLMu で提供されている QEMU を用いた CPU モデルである。ここでは 3 つの QEMU が 3 種類のアーキテクチャ(CRIS, ARM, MIPS)をエミュレートしている。モジュール間の通信には OSCI (Open SystemC Initiative)が提供している SystemC の通信用ライブラリである TLM[8]を使用している。

ハードウェア側(SystemC 側)のコンパイル方法は TLMu を使用するためのヘッダーファイルやライブラリをリンクする以外は、基本的に SystemC のそれと同じである。シミュレートするハードウェアのパラメータは各モジュールのインスタンス時に設定する。

ソフトウェア側(C 言語側)のコンパイルフローを図 2 に示す。

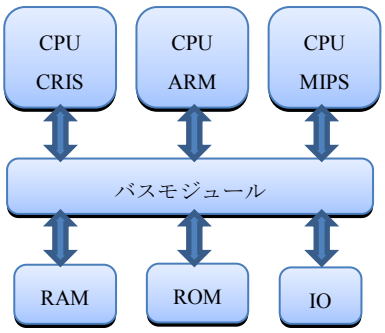


図 1　TLMu サンプルアーキテクチャ

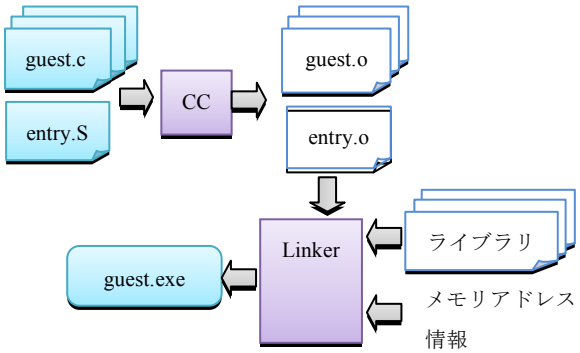
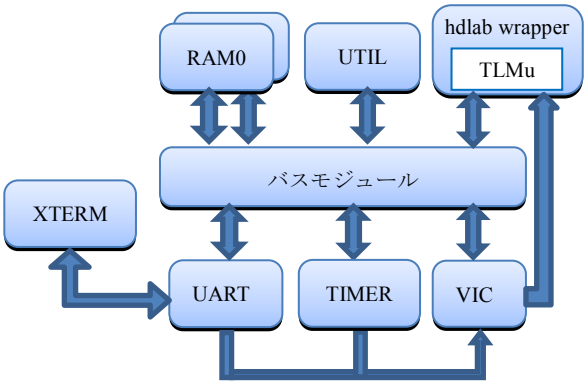


図 2　TLMu ソフトウェア側のコンパイルフロー

リンク時に生成する実行ファイルの命令メモリアドレスとデータメモリアドレスを、リンカオプションとして指定する必要がある。この値がシミュレータ上のメモリモジュールに格納されるアドレスとなる。図 2 の entry.S はスタートアップファイルである。この中でスタックポインタの初期値を割り当てたメモリに収まるように設定する。GCC のオプションでは「-nostartfiles」を付け、コンパイラが自動的にこれを生成しないように設定する必要がある。生成した guest.exe は SystemC の実行時に各 CPU モデルにロードされる。ソフトウェア側で設定される各パラメータは、SystemC 側で設定されるものとの整合性が必要である。

3.4 hdLab TLMu 環境

hdLab TLMu 環境 [11]は hdLab 社が公開している TLMu を拡張した SystemC 用のモジュールライブラリである。図 3 にこれと SCML(SystemC Modeling Library)[10]を組み合わせて構築されたサンプルのアーキテクチャを示す。



VIC=Versatile Interrupt Controller

図 3　hdLab TLMu 環境

hdLab TLMu 環境ではタイマや割り込みコントローラなどが用意されており、拡張した TLMu ARM モデルへの割り込み処理を実現している。またサンプルとしてシングルコア環境が提供されており、この上で Linux カーネル ver3.4 が動作する。

4. マルチコアシミュレーション環境の構築

本研究ではhdLab TLMu環境をベースに、マルチコアSoCソフトウェアシミュレーション環境の開発を行った。まずhdLab TLMu環境をシングルコアからマルチコアへ拡張した。さらに共有メモリを増設し、これを介したCPU間の通信用関数を実装した。

図 4にhdLab TLMu環境のCPUを2個に拡張し、CPU間の通信用共有メモリ(SHARED MEMORY)を増設したアーキテクチャを示す。このメモリは特定のCPUに割り振らず、全てのCPUからアクセス可能である。またCPUと共にXTERM, UART, TIMER, VICもそれぞれ増設している。

表 1 に図 4 で増設した共有メモリを介した通信用関数を示す。後述する評価実験用であり、int型のみではあるが、他のデータ型への対応も容易である。

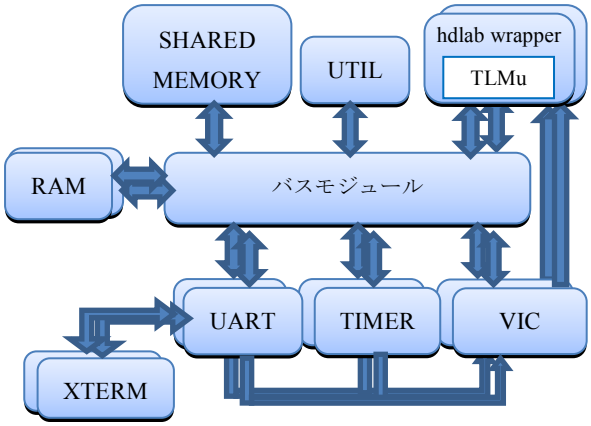


図 4　2CPU アーキテクチャ

表 1　通信用関数

関数名	用途	引数
int SendIntData (int *array_base, int length, int cpu_id)	int 型データ送信用	array_base 送信データ配列の先頭番地 length 送信データの要素数 cpu_id　　送信先 ID
int RecvIntData (int *array_base, int length)	int 型データ受信用	array_base 受信用配列の先頭番地 length 受信データの要素数

図 5に表 1のSendIntData()とRecvIntData()の動作の概念図を示す。共有メモリ内をCPU数に分割し、各CPUの受信バッファおよび制御用変数として使用する。制御用変数は各CPUの受信バッファへのアクセスを制御する。

まず送信側は、送信先の受信バッファが書き込み可能かどうかを判断するために制御用変数にアクセスする。書き込み可能な場合、受信バッファにデータを書き込む。書き込み不可の場合は、可能になるまで待機する。次に受信側は、自分の受信バッファが読み出し可能かを判断するために制御用変数にアクセスする。読み出し可能な場合、データを読み出し、受信バッファのデータを削除する。読み出し不可の場合は、可能になるまで待機する。

今回の仕様では、バッファにデータが1度書き込まれると、それが読み出されるまで新しいデータは書き込めない。また受信側はバッファのデータを読み出す際、そのデータの送信元を指定することはできず、ただ存在するデータを読み出す。データが存在しない場合は、何らかの書き込みがあるまで待機する。

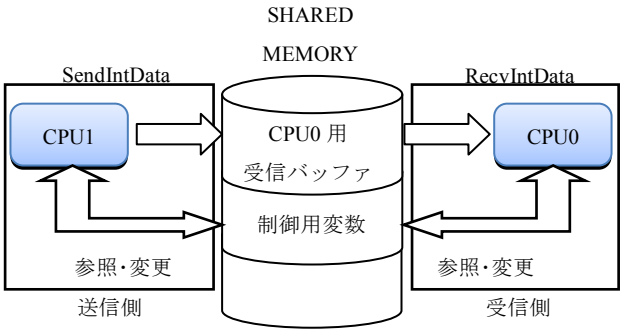


図 5　CPU 間の通信用関数の概念図

5. 実験

本研究で構築したマルチコアSoC用ソフトウェアシミュレータの評価実験として、JPEG変換プログラムを9段のタスクにパイプライン化したものを用いる。これは文献[1]でNaximの実験用として製作されていたもので、本研究のシミュレータ用に改変し、使用した。またオリジナルのQEMUを用いているNaximを性能比較対象にした。

5.1節で評価実験に用いるJPEGパイプライン圧縮プログラムについて、5.2節で9コアに拡張したシミュレーション環境について述べる。5.3節と5.4節で実験結果について述べる。

5.1 JPEG パイプライン圧縮プログラム

JPEGパイプライン圧縮プログラムではJPEG圧縮の全行程を9段階に分割し、各段階を1つのCPUが担当する。以下で各段階の処理内容を述べる。

第1段階では入力のBMP画像から色情報を取り出し、YCrCb変換とデータの間引きを施す。間引き後のデータ量はY : Cr : Cb = 4 : 1 : 1となる。これと同時にデータをMCU(Minimum Codec Unit = 6×8pixel×8pixel)ごとに分割し、1MCU分を生成する度に次の段階へ送信する。

第2段階から第7段階では、離散コサイン変換(Discrete Cosine Transform)を行う。DCTは他の処理に比べて時間が

かかるので、6 段階に分割し処理時間の均等を図っている。6 段階の DCT で 1 MCU 分の DCT が完了する。第 8 段階では 1MCU 毎に量子化を行う。

第 9 段階では 1 MCU 毎にハフマン符号化を行う。本来、ハフマン符号化は全データ中の出現頻度を元に処理を行うが、パイプラインプログラムでは同時刻にすべてのデータは存在しない。よって、あらかじめ 1MCU 分の出現頻度テーブルを製作し、これに従って符号化する。全 MCU の処理を待って JPEG 画像を出力する。

5.2 9 コア SoC シミュレーション環境の構築

図 6 は QEMU とホストの関係性を示している。オリジナルの QEMU はゲストプログラムがホスト OS 上の BMP ファイルおよび JPEG ファイルに直接アクセスが可能であった。しかし TLMu を用いる場合、QEMU 上からアクセスできるメモリは SystemC でエミュレートしているメモリのみで、ホスト OS 上にあるファイルに直接アクセスできない。よって図 7 のように SystemC のモジュールにホストとファイルのやり取りをするプロセスを用意し、ゲストプログラムがそのモジュールにアクセスすることで、ホストとゲスト間のデータのやり取りを実現した。このデータの仲介役を図 3 の UTIL に担当させた。

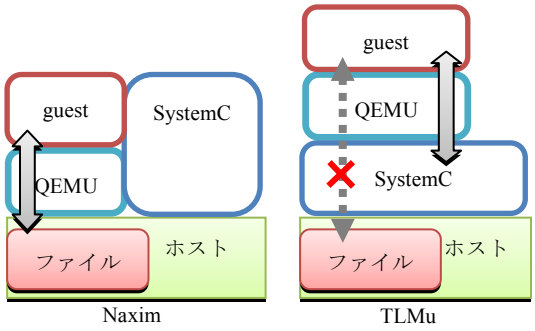


図 6　QEMU, SystemC, ホストの関係

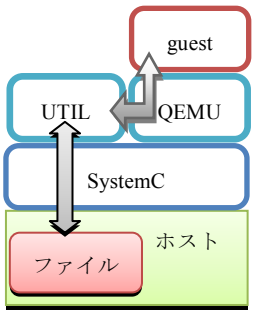


図 7　SystemC モジュールを介したホストとゲストプログラムのファイル受け渡し

図 8 はマルチコア hdLab TLMu 環境のタスク割り当てを示している。図 4 の CPU を 8 個に拡張し、ファイルの入出力に関係するタスクを UTIL に、その他を 8 個の CPU に担当させ Naxim 同様 9 コアで JPEG パイプライン圧縮プログラムを実行する。

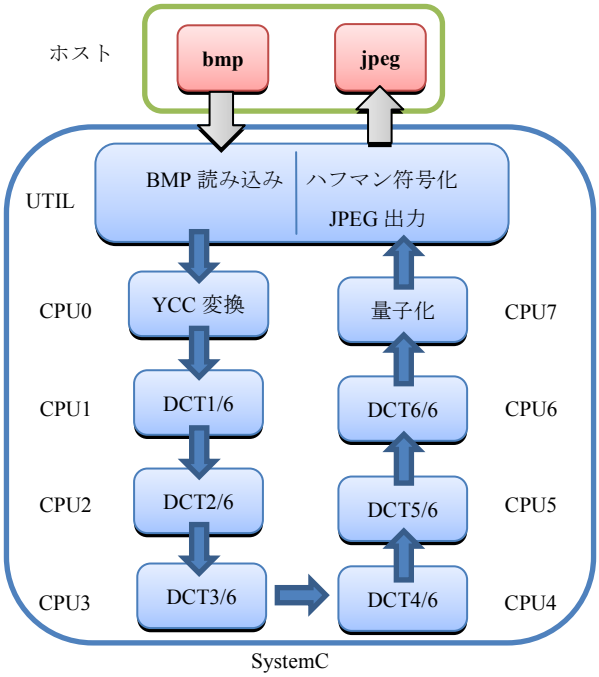


図 8　マルチコア hdLab TLMu 環境のタスク割り当て

5.3 動作確認実験結果・考察

本研究のシミュレーション環境が正しく動作することを確認するための実験を行った。表 2 に実験環境を示す。QEMU のアーキテクチャは ARM cortex-A9 とした。

表 2　実験環境

CPU	Core i7 990x 3.46GHz (6cores 12threads)
MEMORY	DDR3-1333 12GB
HDD	SATA3.0 2TB 5900rpm
OS	Ubuntu 12.04LTS 64bit
Kernel	Linux kernel 3.2.0-35-generic
QEMU	version 0.15.50
SystemC	version 2.2.0

まず JPEG パイプライン圧縮プログラムの入力画像の画素数を変化させてシミュレーションを行った。図 9 は画素数を 64×64, 128×128, 256×256 と変化させたときのシミュレーションサイクル数を示している。

Vol.2013-SLDM-160 No.14
Vol.2013-EMB-28 No.14
2013/3/13

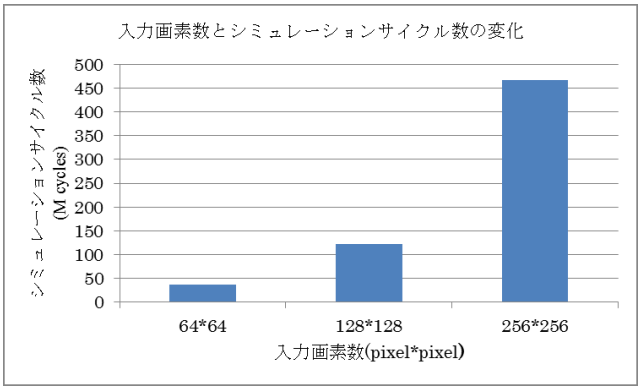


図 9　入力画素数とシミュレーションサイクル数の変化

次に CPU 数を 1, 2, 4, 8 と変更してシミュレーションを行った。図 10 に実験結果を示す。CPU 数の変化に伴って JPEG パイプライン圧縮プログラムのタスク割り当てを、各 CPU で行う処理量が均一になるように調整している。また、JPEG パイプライン圧縮プログラムの入力画像の画素数は 128*128 pixel に固定している。

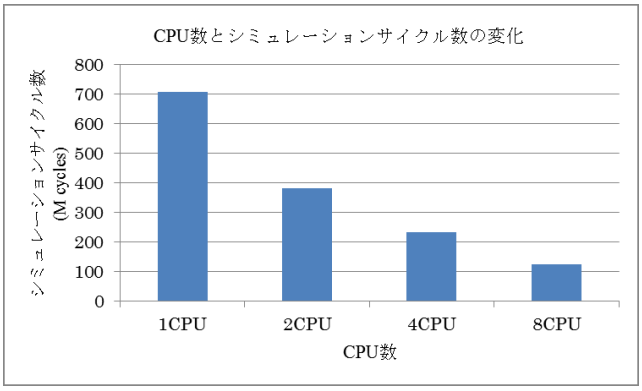


図 10　CPU 数とシミュレーションサイクル数の変化

まず図 9 において、入力画像の画素数に比例してシミュレーションサイクル数が増加している。これは予想通りの結果であるといえる。また図 10 でも CPU 数にシミュレーションサイクル数がおよそ反比例して変化していて、こちらも理想的である。これらの実験より TLMu を用いたシミュレーション環境がパラメータの変更を相対的ではあるが、シミュレーション結果に正しく反映していることが確認できた。

5.4 性能評価確認実験・考察

オリジナルの QEMU を用いた Naxim と比較して、TLMu を用いたシミュレータがどの程度処理に時間を要するか計測した。縦 3 コア、横 3 コアの計 9 コアの NoC プラットフォームを構築し、各コアに JPEG パイプライン圧縮プログラムの 1 段分を割り当てる。図 11 に 9 コア Naxim のタスク割り当てを示す。入力となる BMP 画像の画素数は 128×128pixel である。

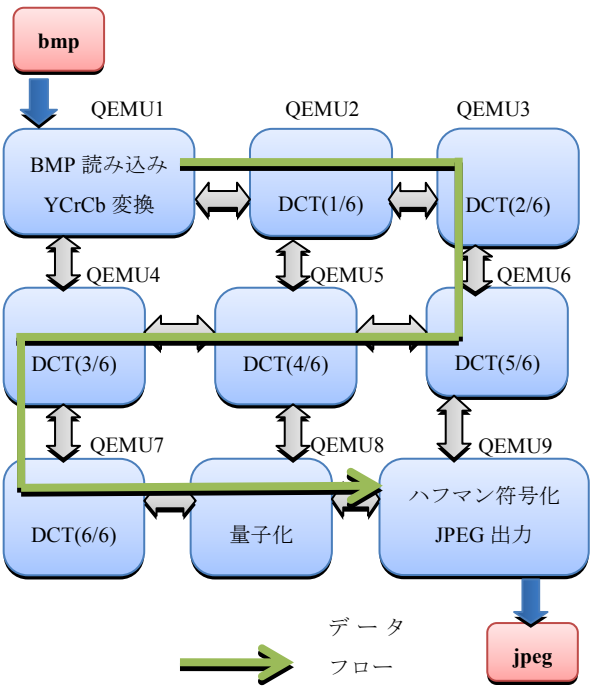


図 11　9 コア Naxim のタスク割り当て

表 3　Naxim と TLMu の性能比較実験結果

	シミュレーション時間(sec)
Naxim	24
TLMu	23466

表 3 より、Naxim に比べておよそ 1000 倍遅いことが分かる。元々QEMU 内で行っていたバスやメモリのシミュレーションを、SystemC で実装している点がこの結果を招いた理由だと考えられる。

次にシミュレートするハードウェア量がシミュレーション時間にどう影響するかを確認するため、5.3 節の CPU 数を変化させた実験のシミュレーション時間を計測した。図 12 に結果を示す。また図 13 に図 12 のシミュレーション時間と図 10 のシミュレーションサイクル数の関係を示す。

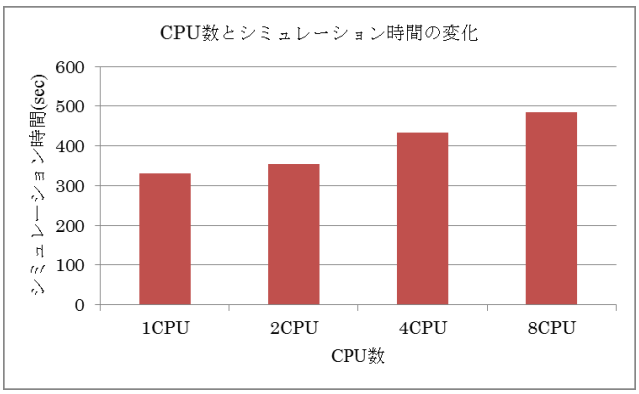


図 12　CPU 数とシミュレーション時間の変化

Vol.2013-SLDM-160 No.14
Vol.2013-EMB-28 No.14
2013/3/13

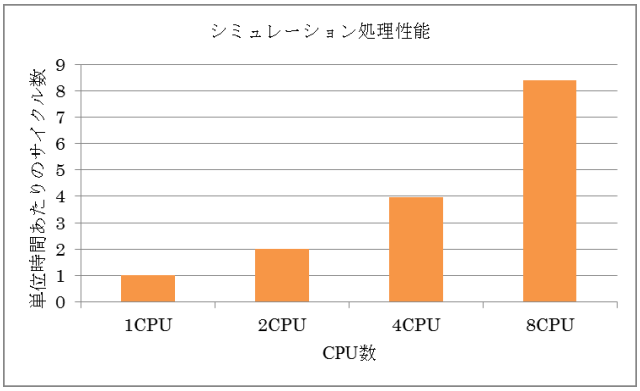


図 13　単位時間あたりシミュレーションサイクル数

図 13 では 1CPU の場合の結果を 1 とし、他の場合の結果と比較している。この実験ではメモリ以外のモジュール数は CPU に比例しているので、hdLab TLMu 環境も SystemC の処理性能がシミュレートするハードウェア量に依存していることが確認できた。

6. おわりに

本研究では SystemC と QEMU を加工した TLMu を用いてマルチコア SoC のシミュレーション環境を構築した。実験を通じて、TLMu を用いることでハードウェアの詳細なモデルリングが可能なソフトウェアシミュレーションを行えることを確認した。

今後の課題としては、ハードウェアの詳細な設定を可能にしたことで増大した、シミュレーション時間の削減が第 1 に挙げられる。これには、シミュレーションの詳細度をモジュールごとに設定し、詳しく解析する部分を選択できる機構が必要と考える。解析が必要な部分のみを詳細に設定し、その他の部分の抽象度を上げることで、全体の処理時間を削減し、より実用的な物に近づくことができる。

参考文献

[1] Keita Nakajima, Takuji Hieda, Ittetsu Taniguchi, Hiroyuki Tomiyama and Hiroaki Takada, “A Fast Network-on-Chip Simulator with QEMU and SystemC,” *International Workshop on Advances in Networking and Computing (WANC)*, 2012.

[2] Marius Gligor, Nicolas Fournel and Frédéric Pétrot, “Using Binary Translation in Event Driven Simulation for Fast and Flexible MPSoC Simulation,” *CODES+ISSS,* 2009.

[3] Ming-Chao Chiang, Tse-Chen Yeh, and Guo-Fu Tseng, “A QEMU and SystemC-Based Cycle-Accurate ISS for Performance Estimation on SoC Development,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, Apr. 2011.

[4] Cheng-Shiuan Peng, Li-Chuan Chang, Chih-Hung Kuo and Bin-Da Liu**.** “Dual-core Virtual Platform with QEMU and SystemC,” *International Symposium Next-Generation Electronics (ISNE),* 2010.

[5] Màrius Montón, Jakob Engblom, and Mark Burton, “Checkpointing for Virtual Platforms and SystemC-TLM,” *IEEE Trans. on Very Large Scale Integration (VLSI) Systems,*vol. 21, no. 1, Jan. 2013.

[6] Claude Helmstetter, Vania Joloboff and Hui Xiao**,** “SimSoC: A Full System Simulation Software for Embedded Systems,” *International Workshop* on *Open-source Software for Scientific Computation (OSSC),* 2009.

[7] QEMU Emulator User Documentation, http://qemu.weilnetz.de/qemu-doc.html.

[8] Accelera Systems Initiative, http://www.accellera.org/home/.

[9] Edgar E Iglesias, Transaction Level eMulator (TLMu), http://edgarigl.github.com/tlmu/.

[10] Synopsys, System-Level Design, http://www.synopsys.com/cgi-bin/slcw/kits/reg.cgi.

[11] hdLab, ARM CPU モデル環境, http://www.hdlab.co.jp/web/a050consulting/b009armcpumodel/.