Vol.2009-EMB-14 No.11
2009/7/24

QEMUを利用したV850シミュレータの開発と評価

尾崎辰典[†1,*1] 中本幸一[†1] 藪内健二[†2]

自動車や航空機はいまや小型のコンピュータがネットワークで結合された大規模分散組込みシステムとなってきている．こうしたコンピュータシステム，特にソフトウェアの規模や複雑さが最近急速に増大している．このため，製品のハードウェアとソフトウェアを並行同時開発する必要があり，製品のハードウェアが存在してなくてもソフトウェアの開発を進めるための各種のシミュレータが利用され，それらを統合した仮想実行環境が必要となってきている．本研究ではその一部として，命令変換型のシミュレータであるQEMUを利用してNEC製マイクロプロセッサV850のシミュレータを開発した．本稿では，その実装と評価を述べる．

Implemenation and Evaluation of V850 Simulator using QEMU

Tatsunori Osaki,[†1,*1] Yukikazu Nakamoto[†1] and Kenji Yabuuchi[†2]

The size and complexity of large-scale distributed embedded systems such as automotive and process control have increased recently. In order to develop the large-scale distributed embedded systems with high productivity and quality, a virtual execution environment is required. This environment integrates numerous CPU simulators and various device simulators through the network and provides network-wide simulation functionalities in the distributed system. In this paper, we present implementation and evaluation of V850 CPU simulator using QEMU as one of simulator in the environment.

†1 兵庫県立大学大学院応用情報科学研究科
Graduate School of Applied Informatics, University of Hyogo
*1 現在，富士通テン株式会社
Presently with Fujitsu Ten Limited
†2 株式会社オクトパス
OCTOPATH Corporation

1. はじめに

自動車や航空機はいまや小型のコンピュータがネットワークで結合された大規模分散組込みシステムとなってきている．こうしたコンピュータシステム，特にソフトウェアの規模や複雑さが最近急速に増大している．さらに，これらのシステムは高信頼性，高環境性が要求され，そのため開発コストも膨大である．大規模分散組込みシステムは多くのCPUや物理空間と制御を行うデバイスから構成されること，さらに大規模なネットワークシステムであることが特徴である．制御を行うデバイスには物理空間での物体，自動車や航空機を構成する部品，様々なハードウェアがある．自動車システムでの例では数10個から100個程度のコンピュータ(車載システムではECUという)が使用され，センサーがデータを取得し，ECUがデータ処理，アクチュエータを通じて必要な制御コマンドを制御対象に送る．一方，市場からは組込みシステムの開発コストの削減，より短期間での開発が要請されている．特にソフトウェアの規模と複雑さが増大しており，これが開発期間の長期化と開発コストの増大の主要因となっている．このため，製品のハードウェアとソフトウェアを並行同時開発する必要があり，製品のハードウェアが存在してなくてもソフトウェアの開発を進めるための各種のシミュレータが利用されてきている．CPUボードの代替としてソフトウェアを稼動させるCPUシミュレータがその代表的なものである．これだけでなく，エンジンやブレーキといった制御デバイスをシミュレーションするデバイスシミュレータも利用されている．ここでは，CPUだけでなく制御デバイスまで含めたシミュレータを統合した環境を仮想実行環境と呼ぶことにする．すなわち，大規模分散組込みシステム開発のために，分散組込みシステム向け仮想実行環境が必要とされている．

本稿では分散組込みシステム向け仮想実行環境におけるCPUシミュレータをQEMUを用いて開発し評価したのでこれを報告する．ターゲットCPUはNEC製プロセッサV850である．このような仮想実行環境においてCPUシミュレータを自作したのは，廉価で拡張が容易なCPUシミュレータが必要であったからである．

2. 仮想実行環境と関連研究

仮想実行環境には以下の形態がある．Software-In-the-Loop Simulation（SILS）では，CPUシミュレータ上のプログラムがデバイスシミュレータからセンサーデータを取得，処理，制御を行うものである．一方，Hardware-In-the -Loop Simulation（HILS）では，ECU上のプログラムがデバイスシミュレータからセンサーデータを取得，処理，制御を行

Vol.2009-EMB-14 No.11
2009/7/24

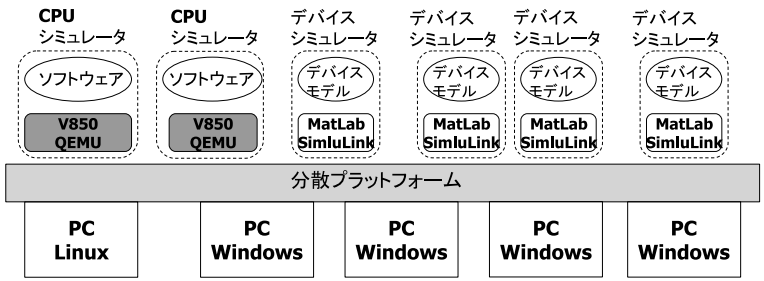

図 1 分散組込みシステム向け仮想実行環境

うものである．HILS は制御対象装置の代替として利用される．今後，より安全，より快適，環境に配慮したサービス提供のために，周辺車両も含めた多数のセンサーからデータ取得を行い，デバイスを制御することが必要となってくるため，組込みシステムは必然的にネットワーク化，分散化，大規模化してくる．このため，SILS，HILS が複雑化していく一方，必要性も高まってくると考えられる．また，制御のための入力データはこれまでのアナログデータだけでなく，インタネットなどサイバー空間からのデジタルデータも必要となる．例えば，オートクルーズ制御で，地図データや渋滞情報を利用すると言うものである．

以下に仮想実行環境に関する関連研究を述べる．これには制御対象のモデル化から行うトップダウンアプローチと既存のシミュレータを結合させていくボトムアップアプローチがある．前者には，カリフォルニア大バークレイ校の Chess プロジェクト（Center for Hybrid and Embedded Software Systems）での Actor-oriented model[11] がある．ここでは，従来からの物理モデル以外に，データフロー，離散イベントモデルなどを統合したシミュレーションモデルを構築し，Java 言語を使った開発環境として提供している．しかし，開発モデルが従来と異なることからソフトウェア開発者の技術修得に難があると考えられる．IBM の SysML は 物理モデルと UML による離散モデルの統合を目指している[2]．一方，後者には，既存の CPU シミュレータと制御モデルとを UNIX のパイプ機能を使って結合されたものがある[5]．

筆者らのグループでは，上述の大規模分散組込みシステムにおける高生産，高品質のソフトウェアの開発支援のために，CPU と制御対象デバイスのシミュレーション環境とこれらを統合させる分散プラットフォームの基盤技術の研究を行っている (図 1)[9]．

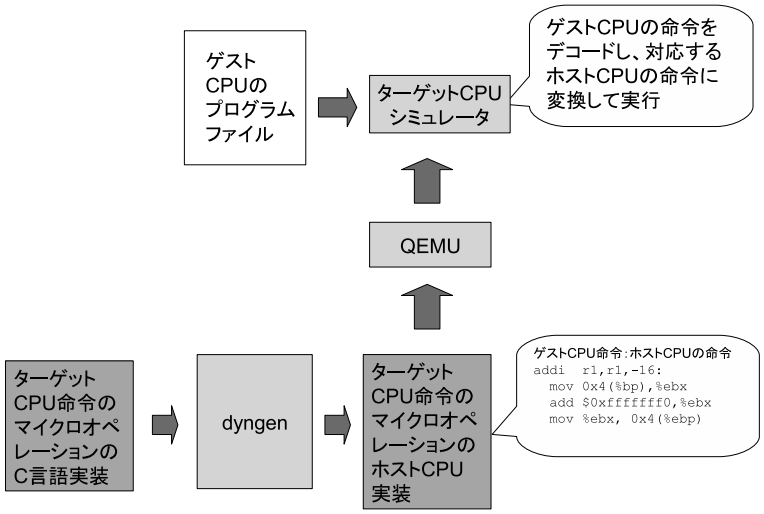

図 2 QEMU の機能

3. QEMU の機能

本研究で使用した QEMU は Fabrice Bellard によって開発されたオープンソースの CPU シミュレータである[3]．現在 Linux 上の仮想マシン KVM[6] や Android の開発環境 (例えば 7))，OS 教材[12] に利用されている．現在，x86，ARM，SH，MIPS，PowerPC，68000 などの CPU シミュレータが実現されている．

CPU シミュレータ開発のベースを QEMU としたのは次のような理由による．

- オープンソースであり，機能拡張が容易である．
- Linux，Windows，MacOS，FreeBSD のような多くの OS 上で利用可能なシミュレータを生成する．さらに，新たなマシン記述とエミュレートデバイスを加えることで，特別な組込み機器を比較的容易にシミュレートすることができる．
- 命令変換方式の CPU シミュレーションを行うため，高速なシミュレーション実行が期待できる．このため実時間でのシミュレーションの可能性もあり，ターゲットマシンとシミュレータが混在した環境でのシミュレーションも期待できる．
- 既に多くの CPU シミュレータが実現されており，仮想実行環境においてこれらの CPU

シミュレータの利用が期待できる．
- マルチコア CPU の対応がなされている．
- ポートアクセス時に起動する関数を定義できるなど組込みシステムの入出力のシミュレーションに有効な機能がある．
- 命令変換方式の技術を習得したかったこと．

一方、デバッガ機能の内蔵しているがブレークポイントでの停止，再開などであり，ターゲットプログラムの実行制御は別途用意する必要がある．ターゲットプログラムのデバッグという目的には適さないと考えられる．

QEMU における命令変換は，実行時にターゲット CPU の命令をホスト CPU 用に変換することによってシミュレーションを行う．この変換の概要を図 2 に示す．この変換方法では，まず，ターゲット CPU の命令をデコードしマイクロオペレーションの列に分割する．QEMU でのマイクロオペレーションは，ターゲット CPU より単純な命令であり，ホスト CPU の命令列に対応付けられている．マイクロオペレーションは C 言語で記述され，**dyngen** というツールとホスト CPU 上のコンパイラ (多くは x86 のコンパイラ) によってホスト CPU の命令列に変換される．マイクロオペレーションとそのホスト CPU 命令列の対応は QEMU の生成したシミュレータ内で保持される．QEMU の生成したシミュレータでは，ターゲット CPU の命令をデコードして，その命令に対応するマイクロオペレーションを求め，このオペレーションのホスト CPU コードを Translated Block(TB) というバッファにコピーする (図 3 参照)．この操作はターゲットプログラムにおいてジャンプ等の命令が現れるまで続けられる．ジャンプ命令が現れるとターゲットプログラムのデコードは中断して，TB 上にある変換されたホスト CPU の命令列を実行する．デコード時にターゲット CPU の命令列が既にバッファ内の TB にある場合はキャッシュとしてこれが再利用される．

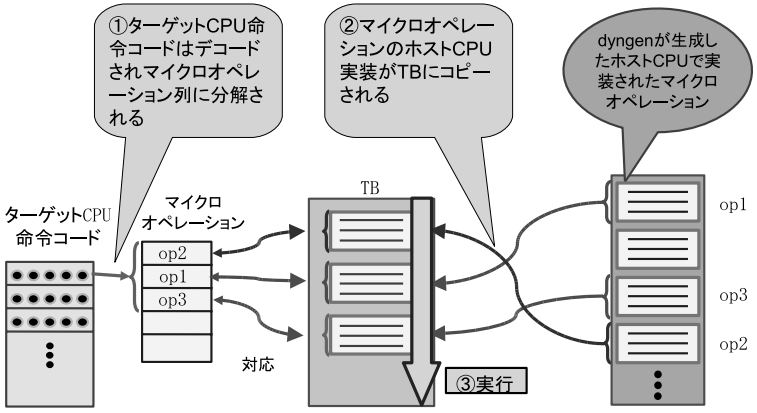

図 3 QEMU の命令変換

3.1 マイクロオペレーションの仕様

本研究で使用した QEMU はバージョン 0.9.0 である．このバージョンの QEMU のマイクロオペレーションを定義する上でいくつかの規則がある．

R1 `T0` から `T2` までの 3 つの仮想レジスタが利用可能であること．

R2 ターゲット CPU のレジスタ値は，変数 `env` の指す領域に保持されること．

R3 マイクロオペレーションは，`op_`で始まる関数名で，その型は `OPPROTO` であること．

R4 `op_`で始まる関数に対する実パラメタは，`PARAM1` のようなマクロ変数でアクセス可能なこと．

R5 マイクロオペレーションの関数名は，`RETURN()` で終わること．

R6 マイクロオペレーションは `op_`の前に `gen_`を付けて呼出しを行うこと．

QEMU は他のホスト CPU への移植が容易とされる．これはマイクロオペレーションが C 言語で書かれており，移植先のホスト CPU の C コンパイラで当該ホスト CPU の命令に変換されるからである．

4. V850 QEMU の開発

4.1 V850 の機能概要

V850 は NEC 製の組込み用 CPU で，シングルチップ・マイクロコンピュータである[10]．

V850 はユーザーモード（プロテクション）のない RISC プロセッサーである．また命令数は 74，32 ビット汎用レジスタは 32 本，ロング/ショート形式を持ったロード/ストア命令，3 オペランド命令，プログラム空間は 16M バイト・リニア，データ空間は 4G バイト・リニア，飽和演算命令，ビット操作命令といった特徴を持つ．ただし，特権モード/非特権モードの違いはなく，仮想空間機能もない．

4.2 V850 用マイクロオペレーションの仕様

上述した規則に従い，V850 の命令に対してマイクロオペレーションを定義し，これらを使って V850 命令を分解した．

算術演算命令: V850 の算術演算命令に対して，以下のようなマイクロオペレーションを定義した．

- op_mov_rN_T0(n)
  n で示される汎用レジスタの値を T0 レジスタに入れる．
- op_add_T0_rN(n)
  n で示される汎用レジスタに T0 レジスタの値を加算する．
- op_mov_T0_rN(n)
  T0 レジスタの値を n で示される汎用レジスタの値に入れる．

これらを使用し，V850 の add reg1, reg2 という命令は以下のようにマイクロオペレーションに分解される．

```
gen_op_mov_rN_T0(REG1);
gen_op_add_T0_rN(REG2);
```

REG1，REG2 はそれぞれ，add 命令中でレジスタ番号を示すビット列から抽出したものである．

ロード/ストア命令: V850 のロード/ストア命令に対して，以下のマイクロオペレーションを定義した．

- op_addr_T1(r,d)
  r レジスタの内容から d 加算した値を T1 に入れる．
- op_ldX_T1_T0()
  T1 の値を T0 レジスタにロードする．ここで X は b, ub, w，uw，l，ul のいずれかであり，それぞれ符号ありなしでのバイト，ワード，ロングワードでのアクセスを示す．
- op_stX_T1_T0()
  T0 レジスタの内容を T1 のアドレスにストアする．X は上と同じ．

これらを使用し，V850 の ld.w disp16[reg1], reg2 という命令は以下のようなマイクロオペレーションに分解される．

```
gen_op_addr_T1(REG1,DISP16);
gen_op_ldul_T1_T0();
gen_op_mov_T0_rN(REG2);
```

また，st.h reg2, disp16[reg1] という命令は以下のようなマイクロオペレーションに分解される．

```
gen_op_addr_T1(REG1,DISP16);
gen_op_mov_rN_T0(REG2);
gen_op_stw_T1_T0();
```

条件分岐命令: V850 の分岐命令をシミュレートするために以下のようなマイクロオペレーションを定義した．

- gen_op_chkCOND()
  現在の条件レジスタ (PSW) の内容を調べて，COND で指定された条件が成立しているかどうかを返す．COND には lt(less than signed)，v(overflow) など V850 の条件分岐命令に対応した条件が入る．

V850 の blt L という命令は以下のようなマイクロオペレーションに分解される．

```
gen_op_chklt();
l1 = gen_new_label();
gen_op_jT(l1);
gen_goto_tb(ctx, 0, PC);
gen_set_label(l1);
gen_goto_tb(ctx, 1, L);
```

gen_new_label() 以降の関数は QEMU で提供されている関数で以下のような機能を有する．gen_new_label()，gen_op_jT(l1) で新しいラベル l1 を生成する．gen_goto_tb(ctx, 0, PC) で偽の場合のジャンプ先 (この場合，ブランチ命令の次の命令を指す PC) を設定し，gen_set_label(l1)，gen_goto_tb(ctx, 0, L) で真の場合のジャンプ先 (この場合，ブランチ命令のジャンプ先のラベル) を設定する．

4.3 V850 のマイクロオペレーション

V850 のマイクロオペレーションを C 言語で実装した．その例として gen_op_add_rN_T0(REG1) の V850 マイクロオペレーションの実装を図 4 に示す．

このマイクロオペレーションは予め x86 命令にコンパイルされ，そのコードは op_add_T0_rN に対応づけられる．add 命令実行時に gen_op_add_T0_rN(REG2) が呼ばれた時に，対応づけられた x86 のコードが TB に転送され，実行される．

4.4 V850 QEMU の実装

V850 QEMU のための主な改造方針と改造箇所は以下である．

- QEMU のソースコードを調査し，もっとも簡単なコーディングである SH を改造母体とすることにした．

```
void OPPROTO op_add_T0_rN(void)
{
   int32_t reg1_a,reg2_a,reg2_b;

   T1 = env->regs[PARAM1];
   env->regs[PARAM1] += T0;

   reg1_a = (int32_t)T0;
   reg2_a = (int32_t)T1;
   reg2_b = (int32_t)env->regs[PARAM1];

   if((unsigned)reg2_a > (unsigned)reg2_b)   psw_set(env->psw,PSW_CY);
   else    psw_unset(env->psw,PSW_CY);

   if((reg1_a>=0&&reg2_a>=0&&reg2_b<0)||(reg1_a<0&&reg2_a<0&&reg2_b>=0))
      psw_set(env->psw,PSW_OV);
   else psw_unset(env->psw,PSW_OV);

   if(reg2_b < 0)   psw_set(env->psw,PSW_S);
   else psw_unset(env->psw,PSW_S);

   if(reg2_b == 0)  psw_set(env->psw,PSW_Z);
   else  psw_unset(env->psw,PSW_Z);
   RETURN();
}
```

図 4 gen_op_add_rN_T0 のマイクロオペレーションの実装

- V850 は仮想空間がないので，QEMU の仮想空間機構である softmmu を利用して仮想アドレスと物理アドレスを同一アドレスで固定させた．
- 上述したマイクロオペレーションを実装した．
- 命令コードのデコード部を追加した．

ソースコードサイズの改造母体規模と V850 に適応させた QEMU 開発規模を表 1 と表 2 にそれぞれ示す．改造工数は，コンピュータアーキテクチャの知識を持った者が行い，約 1 人月であった．

表 1 改造母体規模

モジュール名	行数 (KL)
共通部分	77
ハードウェアドライバエミュレーション (hw,slirp,fpu)	55
ターゲット CPU 命令実行部 (arm,mips,m68k,i386,ppc,sh)	58
その他	53
合計	243

表 2 V850QEMU の開発規模

ファイル名		概要	規模 (行)
TARGET_dis.c			-
hw	v850.c	ターゲットボードの定義	173
target-v850	cpu.h		109
	exec.h		69
	helper.c	割込みメモリ管理	163
	op.c	ターゲット命令をホスト命令で定義	1542
	op_helper.c	op.c で呼ばれるホスト命令実行でのサブルーチン	133
	op_mem.c	メモリロード・ストア時の glue	79
	translate.c	ターゲット命令をホスト命令に変換	771
合計			3039

5. 性能評価

開発した V850 シミュレータの性能測定を行った．性能測定を行った環境は表 3 のとおりである．

表 3 測定環境

CPU	Intel Core2 Duo CPU E8400 @ 3.00GHz,2.99GHz
メモリ	3.24 GB RAM
OS	Vine Linux 4.2

性能測定のために以下のプログラムを用意した．

T1: 10 万回の空ループするプログラム

T2: `mov` 命令を 10 万回のループするプログラム

T3: `add` 命令を 10 万回のループするプログラム

更に，4 からフラグ設定に多くのプログラムを要していることから，フラグ設定を省略したマイクロオペレーションを作成し，**T4** として実行した．

図 5 は QEMU の命令ルーチンを簡易的に表記したものである．この図において，$A-B_1-C$ 間を TB にヒットしなかった場合のデコード・命令実行時間，$A-B_2-C$ 間を TB にヒットした場合の命令実行時間を，上記の **T1** から **T4** までのプログラムを実行させ，計測した．実行時間測定は，`gettimeofday` を使用した．

計測結果をまとめると以下のようになる．

- TB にヒットしなかったとき 1 命令を 5〜40μ 秒で実行
- TB にヒットしたとき 1000 命令を 0.7〜7μ 秒で実行
- フラグ設定を省略した場合はフラグ設定を省略しない場合の 30%程度の速度で実行．

一命令に対して，TB にヒットしたときとしてないときで千倍程度の差がでている．高速なシミュレーション実行が可能となる一方，TB にヒットする場合としない場合で速度差が大きく，実時間でのリアルタイム処理のプログラムをシミュレーションする場合，注意が必要となる場合がある．しかし，プログラムの構造が比較的単純である場合，TB に全てのターゲットプログラムのコードを搭載することも可能であり，実時間のシミュレーションも可能となりうる．

6. おわりに

本稿では，現在の車載ソフトウェア開発に用いられているシミュレータについて述べ，現状の課題から要求されるであろうシミュレータの開発について述べた．シミュレータの開発では，QEMU を用いて，V850 をターゲット CPU として，QEMU のマイクロオペレーションを定義し，V850 のマイクロオペレーションを C 言語で記述し実装した．また開発したシミュレータの性能測定を行った．

今回の開発から今後解決すべき課題として，

- フラグセットの簡略化
- 命令コードの最適化
- TCG(Tiny Code Generator) への対応

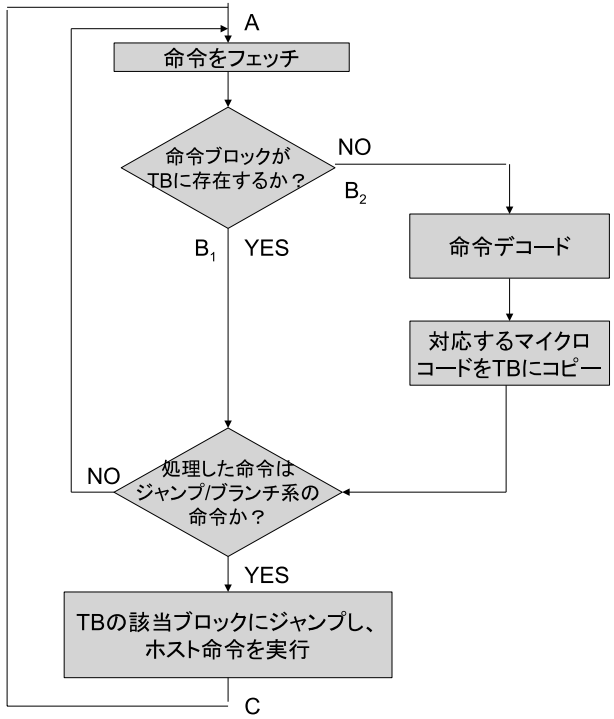

図 5 測定ポイント

- デバイスシミュレータ構築手法と CPU シミュレータとデバイスシミュレータを統合するソフトウェアプラットフォームの研究

があげられる．

現在，マイクロオペレーションの実装の中でフラグの判定を行っている．そのため命令のコード量が大きくなってしまっているためである．フラグセットの簡略化の必要性は性能測定の結果からもわかる．

現在ターゲット CPU の命令は各命令単位でマイクロオペレーションに分解している．しかし，複数命令をまとめてマイクロオペレーション分解することで，より最適化された命令に分解できる可能性がある．この実現にはコンパイラの最適化技術の適用が考えられる．

QEMU の最新版 (0.10.x) では，本稿で述べたようなマイクロコードを個別プロセッサ毎

に設計するのではなく，TCG(Tiny Code Generator) と呼ばれる共通なマイクロコードで実装している[4]．TCG の共通マイクロコードで実装することで，共通コードを利用した様々なサービスを享受することが可能となる (例えば，8))．

制御対象となるデバイスはアナログ，デジタル両方の動作挙動を有するため，これらの挙動を統一的に扱うことが可能なモデルとそれに使ったデバイスシミュレータ構築手法が必要となる．また，CPU シミュレータとデバイスシミュレータを統合するソフトウェアプラットフォームとして，筆者らのグループでは現在オープンソース CORBA である MICO を評価している[1]．

謝　　辞

本研究は科学研究補助金基盤研究 (C)(21500040) と富士通テン (株) の支援を受けている．

参 考 文 献

1) Abe, I., Nakamoto, Y., Terada, H. and Osaki, T.: Approaches to Evaluate Performance of MICO as Distributed Automotive Software Platform, *Proc. the 9th International Symposium on Autonomous Decentralized Systems* (2009). (to appear).
2) Balmelli, L.: An Overview of the Systems Modeling Language for Products and Systems Development, *Journal of Object Technology*, Vol.6, No.6 (2007).
3) Bellard, F.: QEMU, a Fast and Portable Dynamic Translator, *Proc. USENIX 2005 Annual Technical Conference*, pp.41–46 (2005).
4) Fabrice Bellard: qemu - Subversion Repositories (2008). http://savannah.nongnu.org/svn/?group=qemu.
5) Ishikawa, M., McCune, D., Saikalis, G. and Oho, S.: CPU Model-Based Hardware/Software Co-design, Co-simulation and Analysis Technology for Real-Time Embedded Control Systems, *Proc. 13th IEEE Real Time and Embedded Technology and Applications Symposium*, pp.3–11 (2007).
6) Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: kvm: the Linux Virtual Machine Monitor, *Proc. Ottawa Linux Symposium*, pp.225–230 (2007).
7) Leslie, B.: A quick look inside the Android emulator (2007). http://benno.id.au/blog/2007/11/29/android-qemu.
8) Nakamoto, I., Osaki, T. and Abe, I.: Proposing Universal Execution Trace Framework for Embedded Software using QEMU, *Proc. the 1st International Workshop on Software Technologies for Future Dependable Distributed Systems*, pp.173–176 (2009).
9) Nakamoto, Y., Abe, I., Osaki, T., Terada, H. and Moriyama, Y.: Toward Integrated Virtual Execution Platform for Large-scale Distributed Embedded Systems, *Proc. 6th IFIP WG 10.2 International Workshop, Software Technologies for Embedded and Ubiquitous Systems*, pp.317–322 (2008).
10) NEC: V850 ファミリ 32 ビット・シングルチップ・マイクロコンピュータ　ユーザマニュアル　アーキテクチャ編 (1994).
11) Zhao, Y., Liu, J. and Lee, E.A.: A Programming Model for Time-Synchronized Distributed Real-Time Systems, *Proc. 13th IEEE Real Time and Embedded Technology and Applications Symposium*, pp.259–268 (2007).
12) 川合秀美：OS 自作入門，朝日コミュニケーションズ (2006).