教育用命令セットシミュレータの自動生成

柳澤秀明　上原稔　森秀樹

東洋大学工学部情報工学科

我々は、プロセッサのハードウェアとソフトウェア開発環境の自動生成を行うプロセッサ開発のための統合開発環境C-DASHの研究を行っている。C-DASHは、シミュレータ、アセンブラ、逆アセンブラ、コンパイラを生成することができるが、シミュレータは、コマンドラインで実行するCUIベースのシミュレータであった。本論文では、C-DASHより生成されるシミュレータのGUI化による教育用シミュレータの自動生成について述べる。

Automatic Generation of Instruction Set Simulator for Education

Hideaki Yanagisawa, Minoru Uehara, Hideki Mori

Department of Information and Computer Sciences, Toyo University

We have been developing C-DASH. C-DASH is a Hardware/Software codesign tool for processor. C-DASH can generate a software development environment (such as simulator, assembler, disassembler, compiler). But the instruction set level simulator is CUI based. In this paper, we construct GUI of simulator and describe utilization of simulator in education.

1. はじめに

近年、組込みシステムの開発が活発になり、組込みシステム開発者が不足している。組込みシステム開発者の養成が必要な状況となってきている。

組込みシステムの多くは、プロセッサ、メモリ、専用回路などをシステムの性能やコストに応じ組み合わせて実装される。組込みシステムを開発するためには、周辺機器を制御する能力が必要であり、高級言語では、直接扱うことの出来ない制御プログラムを開発する必要がある。制御プログラムを開発するためには、ハードウェアの制御に関する知識が必要になり、プロセッサに対するより深い知識が必要となる。プロセッサに対する深い知識を有することで効率的なプログラムの開発が可能となる。

プロセッサを理解するためには、シミュレータを用い実際にプログラムを書きながら動作を理解することが必要である。しかしながら、異なるプロセッサの動作を理解しようとした場合、プロセッサの命令セットやデータサイズの違いを考慮することと同時に、プロセッサごとに異なる開発ツールの利用法も学ばなければならない。

また、教育現場において、オリジナルプロセッサの動作を教育する場合や、新しく開発されたプロセッサの動作を教育する場合には、新しく開発されたプロセッサ用のソフトウェア開発環境（シミュレータ、アセンブラ、逆アセンブラ、コンパイラ）が必要となる。これらのソフトウェア開発環境を教師が独自に開発することは、大変な労力を必要とする。

このため、本研究では、同一のインターフェース、共通の利用方法で異なるプロセッサのシミュレーションを可能にするシミュレータの自動生成を行うプロセッサ開発環境C-DASH[1][2]の開発を行っている。

C-DASHは、プロセッサの仕様記述（命令のビットパターン、ニーモニック、動作定義など）を行うことでプロセッサのハードウェア（Verilog-HDL記述）とソフトウェア開発環境（シミュレータ、アセンブラ、逆アセンブラ、コンパイラ）の自動生成を行うシステムである。しかし、シミュレータは、コマンドラインより実行するCUIでの実装であり、GUIを備えていなかった。

本論文では、C-DASH より生成されるシミュレータを教育用シミュレータとして利用するための GUI 化について述べる。また、教育用シミュレータの自動生成について述べる。

2. 関連研究

関連研究として 2.1 で教育用として利用されているシミュレータについて述べ、2.2 で既存のプロセッサ開発環境について述べる。

2.1. 教育用シミュレータ

教育用として利用される命令セットシミュレータを調べるために、検索サイト Yahoo!と Google を利用し、キーワードとして CASL、COMET、シミュレータ、アセンブリ言語などを組み合わせることにより出力された検索結果より大学の授業として利用されているシミュレータやシラバスの中で紹介されているシミュレータの調査を行った。

神奈川大学横浜キャンパス情報工学の授業で利用されている CASLDV[7]では、レジスタ、メモリ、ソースコードの表示を行っている。

InfoCASL[8]は、神奈川工科大学[15]や日本大学文理学部[14]などで利用されている。InfoCASL では、レジスタ、メモリ、ソースコード、スタック領域が表示され、[assemble]と[Trace]ボタンが配置されている。

帝京大学では、WCASL-II[9]が利用されている。WCASL-II は、2 モードシミュレータとして、CASL のプログラムを習得するための CASL モードと CPU の動作を理解するための COMET モードを用意している。COMET モードでは、初学者にわかりやすい CPU のダイアグラムを用い、その図の上で、1 つの命令が実行される過程(命令取り出し→命令解読→アドレス生成→命令実行」)を視覚的に見ることができるようになっている。

三重大学[12]のシラバスでは、WCASL-II とキャッスルシミュレータ[13]紹介されている。キャッスルシミュレータは、Web 版 CASL II シミュレータでは、ソースプログラムを入力欄に入れ、[アセンブル&実行]ボタンを押すとアセンブルされたコードと実行時のレジスタの値を表示する。Windows 版では、ソースコードを表示し[最初から]、[実行]、[中断]ボタンを配置、レジスタ、メモリ、スタックの表示/非表示を切り替えられるようになっている。

大阪大学[10]で利用されているのは、コマンドベースの COMETII シミュレータ[11]のようである。

ここで述べたシミュレータをまとめると、授業で利用される GUI を利用したシミュレータの共通点として、エディタ機能、アセンブル機能、メモリ（コード）表示、レジスタ表示、実行、ステップ実行である。

本研究では、プログラムの習得を目的とした命令セットシミュレータの自動生成を目的とする。このためシミュレータに必要な機能として、レジスタやメモリ内容が確認できること、ステップ実行が可能なこと、シミュレータとエディタ機能を統合し、アセンブリ言語で記述したプログラムのアセンブルからシミュレーションがスムーズに行えること、また、ソースコートとアセンブルした結果を確認するための逆アセンブラの機能を組み込むことが必要であると考えた。

2.2. プロセッサ開発環境

プロセッサ開発環境は、3 つのタイプに分けることができる。（1）命令セットの動作のみを定義する（nML[3]）、（2）プロセッサのアーキテクチャを定義する（MIMOLA[4]）、（3）命令セットの動作とアーキテクチャを記述する（ASIP Meister[5]、LISA[6]）。

（1）では、パイプラインの段数などハードウェアの仕様を定義することは出来ないが、動作レベルのシミュレータを生成する場合などには適している。（2）は、ハードウェアの仕様を細かく定義する場合に適しているものの、（1）と比べると記述が複雑になり、理解し難くなる。（3）は、（1）と（2）を組み合わせたようなものであり、命令セットの動作記述とハードウェアのアーキテクチャの詳細記述の両方を行い、動作定義は、クロックベースで行われる。

シミュレータの実行速度とシミュレーションのレベル（クロックレベルや動作レベル）と関係し、詳細なシミュレーションほど実行速度が遅くなってしまう。

C-DASH では、命令セットの動作のみの記述、クロックを用いた記述の両方をサポートしている。本論文では、命令セットレベルでのシミュレータを生成するため、複雑な記述を排除した動作レベルでの命令セット定義を行う。

3. C-DASH

C-DASH は、プロセッサを記述するための専用言語 CHDL (C-DASH HDL)を用い ISA(Instruction Set Architecture)を基本としたプロセッサの定義を行う。CHDL 記述でのプロセッサ開発では、リソースの宣言（レジスタやメモリ）と命令セットの定義を行う。

C-DASH は、CHDL でのプロセッサ記述から、論理合成可能なプロセッサの Verilog-HDL 記述と生成したプロセッサのためのソフトウェア開発環境（シミ

ュレータ、アセンブラ、逆アセンブラ、コンパイラ）の自動生成が可能である。C-DASH を利用することで、CHDL での単一のプロセッサ記述から、ハードウェアとソフトウェアの開発環境を自動生成することができるためプロセッサ設計者の開発負担を減らすことができる。また、ISA でプロセッサを定義しているため、命令セットの変更を簡単に行うことができる。

C-DASH では、柔軟に命令セットの動作記述が可能なビヘイビアレベルでの記述と論理合成可能な HDL 記述を生成するためのクロックを基本とする動作記述を行うことができる。クロックを基本とする動作記述を行うことで、CISC, RISC, Stack アーキテクチャの論理合成可能なプロセッサの HDL 記述を生成することができる。

3.1. C-DASH による自動生成

C-DASH は、CHDL でのプロセッサ記述から以下のものを生成する。

- Instruction set level simulator (Java or C)
  命令セットレベルシミュレータは、設計者によって定義された命令セットの動作を確認するために利用される。
- Simulation Compiler
  コンパイル型シミュレーションを可能にするためのターゲットマシンの機械語をホストマシンの機械語に変換する
- Hardware simulator (SpecC)
  C-DASH は、SpecC でのハードウェアシミュレータの生成も行う。このシミュレータは、プロセッサのアーキテクチャのチェックを行うために利用される。
- Compiler (GCC)
  C-DASH では、GCC をクロスコンパイラとして利用するために, GCC のアーキテクチャ定義ファイルを自動生する。(現状では、RISC、CISC 型プロセッサに対する、命令セットの RTL 記述のみ出力)
- Assembler
  アセンブリ言語で記述されたプログラムをオブジェクトコードに変換する。
- Disassembler
  オブジェクトコードをアセンブリ言語に変換する。デバッグやアプリケーションプログラムから命令セットの抽出などに利用される。
- HDL description (Verilog-HDL)
  C-DASH は、ハードウェア記述として Verilog-HDL 記述を生成する。C-DASH から生成されるプロセッサの HDL 記述は、論理合成可能な記述である。

```
reg    gr    16    8;
reg    pc    16;
ram    m     16    65536;

instructin    m    pc;
```

図 1 リソース宣言
Fig. 1 Declaration of Resources

```
instruct adda {0010 0000 dddd ssss iiii iiii iiii iiii} {
    gr[d] = gr[d] + m[i];
}
asm adda {<label> <opcode> <reg> , <labelAddress>} {
    d = number($3);
    s = 0;
    i = address($5);
}
```

図 2 動作定義
Fig. 2 Definition of an Instruction

3.2. C-DASH におけるプロセッサ定義

プロセッサ定義の記述例として CASL のレジスタやメモリなどのリソース記述を図 1 に示し、加算命令（ADDA）の動作記述例を図 2 に示す。

CHDL では、"reg" でレジスタのビット幅や個数を定義する。また、"ram"によりメモリのビット幅とサイズを定義し、命令メモリとプログラムカウンタを"instructin"により明示する。

各命令の動作は、"instruct"命令により定義する。CHDL では、ニーモニック名（adda)、ビットパターン（0010・・・・）と命令の動作を定義し、命令のフォーマットを"asm"命令により定義する。"<label>"は、プログラム中で使われるラベルを表し、"<opcode>"でニーモニック名を表す。"<reg>"でオペランドがレジスタであることを表し、"<labelAddress>"でオペランドが即値または、ラベルであることを表す。"asm"中の"d,s,i"は、"instruct"のサブビット"dddd"、"ssss"、"iiii…"をそれぞれ表し、"number()"でレジスタ番号を"address()"で即値やアドレスをサブビットに埋め込むことを表している。"$N"（N=3,4…）は、","も含めた番号を表している。

4. 教育用シミュレータの自動生成

教育用シミュレータとして必要な機能として以下のことを考慮した。

- シミュレータの GUI 化

- プログラム編集（エディタ）機能
- アセンブル機能
- シミュレーションの実行機能
- ステップ実行機能
- ブレークポイント
- レジスタ表示
- メモリ表示
- インストールのしやすさ

教育用として利用するためには、シミュレータの機能や使いやすさは以外に、シミュレータのインストールの問題を考慮する必要がある。授業で利用する場合など一度に何十台ものPCを利用する。このような環境では、全てのPCに対してインストール作業が必要になる場合がある。

さらに、個人で学習することも考慮する必要がある。個人で学習する場合、それぞれのPCに対してシミュレータのインストールが必要となりPCの初心者でも問題なくインストールできる配布環境が必要である。

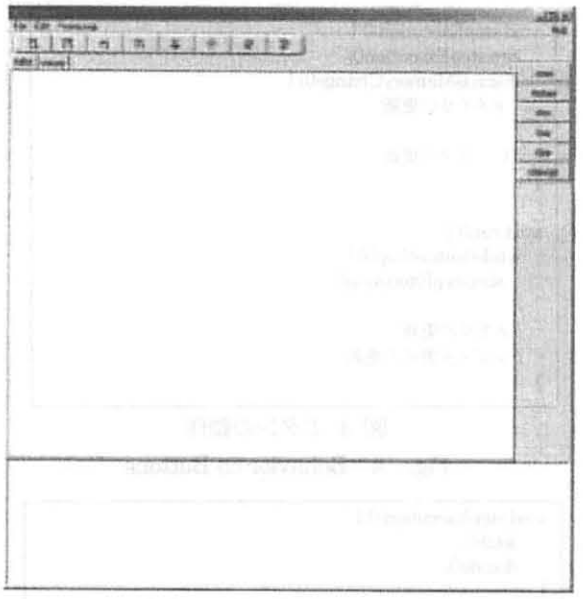
本研究では、インストールに関する問題点を解決するために、Java Web Start（JWS）[16]を利用した。JWSを利用することで、インストール作業の負担をなくすことが可能となる。また、自宅で学習する場合でも、JWSを利用することで簡単にインストールをすることが可能となる。JWSを利用することで、シミュレータのバージョンアップ時でも、ユーザは、意識することなく最新のものが利用可能な状態となる。

4.1. シミュレータGUI

本研究では、プロセッサ開発環境による自動生成を行っているため、シミュレータ本体とGUIを完全に独立させ、GUIのRunまたは、Stepボタンが押される度に、シミュレータの実行を行う構成とした。

構築したシミュレータのGUIを図3に示す。図3（1）は、エディタウインドウであり、ここでプログラムの編集を行う。（2）は、シミュレーションウインドウであり、この画面で、シミュレータの実行を行いメモリやレジスタの値の変化を確認することができる。エディタウインドウとシミュレーションウインドウの切り替えは、[Editor][Viewer]タブにより行う。

シミュレータのGUIとしてテキスト編集などに用いる機能（新規作成、上書き、ファイルを開く、コピー、カット、ペースト、元に戻す）をツールボタンとして上段に配置し、シミュレーションに用いる機能（ロード、リロード、実行、ステップ実行、ストップ）をウインドウの右側に配置した。シミュレータウインドウの左側に、レジスタの値を表示し、中央には、ブレークポイント、メモリアドレス、オブジェクトコード、逆アセンブルしたコードを表示している。



（1）エディタウインドウ

（2）シミュレーションウインドウ

図3 シミュレータのGUI
Fig. 3 Simulator GUI

Stepボタンと Runボタンの動作の概要を図4に示す。ボタンが押されると、シミュレータの実態であるsim.stepExecution()が呼び出される。図5には、シミュレータの動作を示す。シミュレータは、stepExecution()からfetch-execを呼び出すことで各命

```
void stepExecution 0 {
  sim.stepExecution0;
  if(sim.isMemoryChange0) {
    //メモリの更新
  }
  //レジスタの更新
}

void run0 {
  while(!sim.isStop0) {
    sim.stepExecution0;
  }
  //メモリの更新
  //レジスタ表示の更新
}
```

図 4 ボタンの動作

Fig. 4 Behavior on Buttons

```
void stepExecution 0 {
    fetch0;
    decode0;
}
void fetch0 {
    ir = memory[pc];
    pc = pc + 1;}

void exec0 {
    if((ir&ADDA_MASK)==ADDA_OP) {
        //ADDA の処理 }
    else if(ir& INSN_N_MASK)==INSN_N _OP) {
        //INSN_N 処理 }
}
```

図 5 シミュレータ動作

Fig. 5 Behavior of Simulator

令の動作を実行する。
INSN_N_MASK は、命令を特定するためのビットパターンであり ir との AND が命令の INSN_N_OP と等しいときに INSN_N の処理を行う。

4.2. シミュレータ

シミュレータの構成では、対象プロセッサが異なった場合でも、以下の機能を持たせることで GUI と独立して扱うことが出来る。

- メモリデータの受け渡し
- レジスタデータの受け渡し
- メモリデータの変更を監視するフラグ
- メモリデータの変更を調べるメソッド (isMemoryChange0)
- シミュレータのステップ実行 (stepExecution0)
- メモリの扱いは、ワード単位 (固定されていないときは、バイト単位)

4.3. シミュレータの自動生成

シミュレータを生成するためには、各命令のビットパターン、命令を特定するための MASK パターンを用い定義された命令の数だけ条件分岐と動作記述を出力し、exec0を完成させる。シミュレータの生成部を図 6 に示す。

```
for(int i=1;i<定義数;i++) {
    出力("if(ir&");
    出力(命令名+"_MASK==" + 命令名+"_OP) {");
    出力(// 処理);
    出力("}");
}
```

図 6 シミュレータの生成

Fig. 6 Generation of Simulator

5. 評価

本システムの評価を行うために、生成されたシミュレータのダウンロードページと CASL の命令を説明するページを用意した。そして、4 人の学生に例題を実行してもらい、以下のアンケートを行った。

1. JRE は、インストールされていたか?
2. JRE のインストールは、簡単だったか?
3. シミュレータのダウンロード時間
4. シミュレータの起動時間
5. シミュレータの実行速度
6. GUI について (たんなる見た目)
7. 分かり易さ (ボタンやメニューの機能)
8. シミュレータの使いやすさ
9. CASL についてどのくらい理解していたか?
10. シミュレータは、CASL の理解に役に立ったか?
11. シミュレータを使うことで勉強の効率が改善されたか?
12. 以前に何種類くらいのシミュレータを使ったことがあるか?

1 の質問では、二人が「はい」と答え、残りの二人が「自動インストールを行った」と答えた。問 2 では、全員が「はい」と答えている。3 から 8 に対して、5 段階 (良い、やや良い、普通、やや悪い、悪い) で評価してもらった。その結果を以下のグラフに示す。

問 3,4 で「悪い」と答えている者が居るが、これは、大学内のネットワークにトラブルが発生していたことを考慮するとまったく問題ないと思われる。

問 7 では、2 人が「やや悪い」1 人が「悪い」と答

えている。これは、問 12 とあわせて考えると全くシミュレータを使ったことが無い者に対してもボタンの機能を理解できるようにするために、各ボタンにツールチップを付け、何をするボタンかをわかるようにすることとした。

問8で1人「悪い」と答えているが、これは、問7でも「悪い」と答えている者であり、問7の問題を解決することで、改善されると思われる。

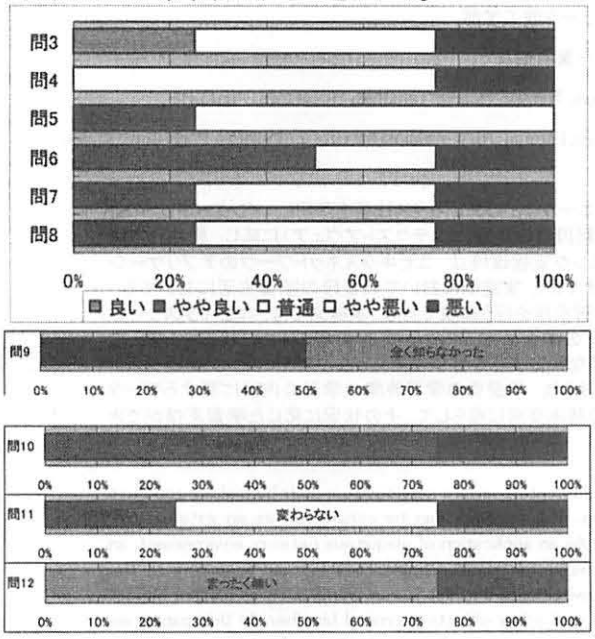


図 7 アンケート結果
Fig. 7 The Results of a questionnaire

問9では、2人が「過去に勉強したが忘れていた」、2人が「全く知らなかった」と答えている。

問10で、3人が「やや良い」と答えているにもかかわらず、問11では、1人が「やや良い」、2人が「変わらない」と答えているのは、シミュレータを使うことで命令セットの動作を理解することには、やや役に立ったが、問題を解くのに苦労したためだと思われる。問題の作り方に不備があったようである。

6. まとめ

本論文では、プロセッサ開発のための統合開発環境 C-DASH を用い、教育用シミュレータの自動生成について述べた。教育用のシミュレータでは、シミュレータの機能として、ステップ実行が行え、各命令実行後のレジスタやメモリ内容の変更が反映できることが必要である。

また、教育用ということを考えシミュレータのインストール作業の煩雑さを考慮する必要があり、今回 JWS を利用した。JWS を利用することにより、PC に Java の実行環境がインストールされていれば、ホームページの JNLP(Java Network Launching Protocol)へのリンクを作ることで、シミュレータのインストールからシミュレータの起動までを自動的に行うことができるようになる。さらに、サーバ上のシミュレータを置き換えるだけで、ユーザが意識することなくシミュレータのバージョンアップを行うことができ、メンテナンスの負担をなくすことができる。

今後の課題として、より内部の動作を理解するためのアーキテクチャレベルシミュレータの実装を行う。また、今回の評価では、数人の学生による評価であったが、実際に授業でシミュレータを用い、シミュレータの使いやすさなどを評価し改善していきたい。

参考文献

[1] Hideaki Yanagisawa, Minoru Uehara, Hideki Mori, "TSA based system design language in HW/SW co-design environment", In Proc. of 13th IEEE International Workshop on Rapid System Prototyping (RSP'02), pp.122-127, 2002.

[2] Hideaki Yanagisawa, Minoru Uehara, Hideki Mori, "Automatic Generation of a Simulation Compiler by a HW/SW Codesing System", In Proc. of 15th IEEE International Workshop on Rapid System Prototyping (RSP'04), pp.53-59,2004.

[3] A. Fauth, J. Van Praet, M. Freericks, "Describing Instruction Set Processors Using nML", IEEE, European Design and Test Conf., 1995.

[4] R. Leupers and P. Marwedel. "Retargetable code generation based on structural processor descriptions", Design Automation for Embedded Systems, 1998.

[5] Makiko Itoh, Yoshinori Takeuchi, Masaharu Imai, Akichika Shiomi, "Synthesizable HDL Generation for Pipelined Processor from a Micro-Operation Description", IEICE Trans. Fundamentals Vol. E83-A, No.3, pp.394-400, March 2000.

[6] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, O. Wahlen and H. Meyr., "A Methodology for the Design of Application Specific Instruction Set Processors (ASIP) Using the Machine Description Language LISA. "International Conference on Computer Aided Design (ICCAD), November 2001.

[7] CASLDV 内田研究室 http://www.inf.ie.kanagawa-u.ac.jp/

[8] InfoCASL 東京理科大学 情報処理技術者試験研究会 http://www.rs.kagu.tus.ac.jp/~infoserv/j-siken/infocasl/

[9] WCASL-II 帝京大学 http://www.ics.teikyo-u.ac.jp/wcasl2/

[10] 大阪大学 COMET II http://www-ise4.ist.osaka-u.ac.jp/~o-mizuno/index.php?Lecture%2F2004Fco

[11] CASLII シミュレータ独立行政法人情報処理推進機構（IPA） http://www.jitec.jp/1_20casl2/casl2dl_001.html

[12] 三重大学工学部情報工学科 http://www.cs.info.mie-u.ac.jp/~toshi/lectures/introcomp/index.html

[13] キャッスルシミュレータ http://www.chiba-fjb.ac.jp/fjb_labo/casl/

[14] 日本大学文理学部計算機室 http://www.cssa.chs.nihon-u.ac.jp/cr/install-2005.html

[15] 神奈川工科大学電子計算センター http://www.kanagawa-it.ac.jp/~l4001/cn16/image125/p2628.pdf

[16] Java Web Start ガイド http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide/javaws/developersguide/contents.html