Interface Development for Web-based Instruction Set Simulator

Hideaki Yanagisawa[1], Minoru Uehara[2] and Hideki Mori[3]
[1] Computer Science and Electronic Engineering, Tokuyama College of Technology
Gakuendai, Shunan, Yamaguchi, 745-8585, Japan
[2,3] Department of Information and Computer Sciences, Toyo University
2100, Kujirai, Kawagoe, Saitama, 350-8585, Japan
E-mail: [1]yanagisawa@tokuyama.ac.jp, [2]uehara@toyonet.toyo.ac.jp, [3]mori@toyonet.toyo.ac.jp

Abstract:
HW/SW codesign tools are necessary to develop new processors in a short period of time, because both the hardware and a software development environment that includes a simulator, assembler, disassembler and compiler, need to be designed for the new processor.

We have therefore develped C-DASH, a HW/SW codesign tool for designing processors. In addition, we have developed SSC-DASH (Server-Side C-DASH), which provides a web-based interface enabling processor designers to use C-DASH on the web.

This paper describes the interface development for a web-based instruction set simulator in a web-based processor development environment.

1. Introduction

This paper discusses the interface development for a web-based instruction set simulator (WISS).

In developing a new processor, it is necessary to design the HW (Hardware) of the processor and to implement a SWDE (Software development environment) that includes a simulator, assembler, disassembler and compiler.

It is, however, difficult to develop both the HW and SWDE in a short period of time. Therefore PDEs (Processor Development Environments) have been developed.

We have developed a PDE (called C-DASH [1]), which is a HW/SW codesign system for designing processors. C-DASH can generate both the HW (Hardware) and SW (Software) from processor definition files based on an ISA (Instruction Set Architecture). In terms of the HW, C-DASH generates synthesizable Verilog-HDL descriptions, while for the SW, C-DASH generates a SWDE that includes a GUI, simulator, assembler, disassembler and compiler (the compiler generator is not yet complete). This SWDE is an IDE (Integrated Development Environment) for the defined processor written in Java.

We have also developed SSC-DASH [2][3], which is a web-based execution environment for C-DASH. SSC-DASH receives a processor definition file via a web browser and calls C-DASH on the server with the processor definition file.

The generated SWDE implements JWS (Java Web Start) [4]. By using JWS, the download and installation of application programs are performed automatically by selecting a link to a JNLP (Java Network Launching Protocol) [5] file on a web page. Signed JAR files of application programs are required to use JWS.

In a PDE, developers write footprint assembly programs to verify the behavior of the defined processor. Currently, processor developers have to download the generated SWDE onto their local machine every time it is required, even if only small changes have been made. Version checking and downloading the SWDE becomes a time consuming part of the verification process.

To verify the behavior of the generated simulator quickly and easily, we have implemented a web-based interface to the generated SWDE using Ajax. By using Ajax, the web-based interface does not need to download the SWDE and web-based interactive simulation is possible without page transfer.

2. Related Works

Related works with regard to PDEs (Processor Development Environments) are described in this section. A HW/SW codesign system is necessary when developing a new processor, because SWDEs depend on the structure of the hardware, such as the ISA and resources, and the hardware cannot function without software.

PDEs have been developed to assist in the development of both the HW and a specific SWDE for a new processor in a short period of time.

PDEs are classified into three types, namely an ISA (instruction set architecture) level description based method such as nML [6]; an architecture level description based method like MIMOLA [7]; and a mixed level description based method that incorporates both the ISA and architecture level descriptions. Examples of the mixed level description method include the CoWare Processor Designer [8] and ASIP Meister [9].

The systems described above are not collaborative development environments. To develop a new processor quickly in limited time, a collaborative development environment is essential.

In a collaborative development environment, resources and the progress of the project are managed appropriately. Using a collaborative development environment for processor designers, it should be possible to develop a new processor efficiently and quickly in limited time.

3. Web-based Collaborative Processor Development Environment

3.1 Processor Development Environment

We have developed a PDE called C-DASH, that adopts a mixed (ISA and architecture based) processor description for the CHDL (C-DASH HDL), which is an original language for describing processors. The CHDL description is composed of resource declarations and instruction descriptions for the target processor.

C-DASH can generate the HW (synthesizable HDL descriptions of a processor) and a SWDE (including a simulator, assembler, disassembler and machine description file for compiler generation) for the processor by extracting the necessary information from the processor description written in CHDL.

C-DASH attempts to generate a compiler and adapts GCC (Gnu Compiler Collection) [10] as a cross compiler for the target processor. (It should be noted that this feature is not yet complete).

C-DASH can describe RISC, CISC and stack architectures based on a single clock and it supports different description levels (behavior and clock based). For more details, refer to [1].

The CHDL provides three methods for describing instruction sets; a behavior description, sequential description and pipeline description.

In the behavior description, it is possible to describe instruction behavior using common C language functions. A behavior level ISA simulator can be generated based on these behavior descriptions; however, the behavior description does not generate an HDL description. From the sequential description, it is possible to generate a clock based ISA level simulator, which is used to check the behavior of the processor instruction set. It is also possible to generate the HDL description of a processor consisting of four fixed stages (FT, DC, EX, WB).

In the pipeline description, the processor designer can change the pipeline stage. C-DASH generates the HDL description of a processor that is used to check the behavior of the HW. Processor designers can then evaluate the performance of the HW and customize the architecture.

3.2 Example of Processor Definition

In defining a processor, resource declarations (such as registers, memory, etc.) and the behavior of the instruction set are required.

To illustrate the processor description at the behavior level, the behavior of the "suba" instruction (subtraction operation) is shown in the code below.

Example of Processor Definition File:

```
/* resource declaration */
reg   gr   16   8;
reg   pc   16;
ram   m    16   65536;
instructin   m    pc ir;

/* instruction set description */
instruct suba {0011 0000 dddd ssss iiii iiii iiii iiii} {
   gr[d] = gr[d] - m[i]; }
asm adda {<label> <opcode> <reg> , <label.address>} {
   d = number($3);
   s = 0;
   i = address($5); }
```

In the CHDL, bit-width and the number of registers are defined by "reg". Memory definitions (bit-width and size) are defined by "ram". To indicate the instruction memory, program counter and instruction register, "instructin" is used.

To define the instruction set, instruction descriptions are used that consist of simulator and assembler descriptions.

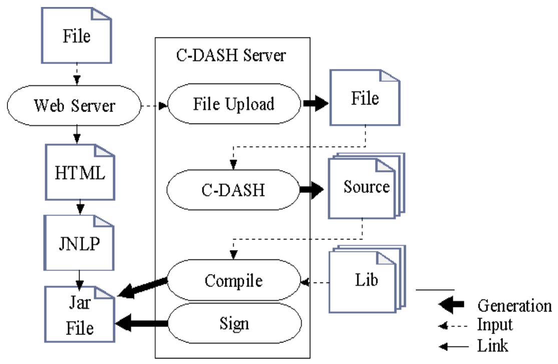

Fig. 1. Generation of SWDE

The example code shows an instruction definition using an "instruct", which consists of a mnemonic name (suba), a bit pattern (0011…) corresponding to the mnemonic and the behavior of the instruction.

The mnemonic name, instruction format and operand description are used to define the assembly language format, "asm". In the assembly program, "<label>" represents a label name, while "<opcode>" represents a mnemonic name, which in this case is “suba”. "<reg>" represents an operand register and "<label.address>" is an immediate value or label name.

The "d, s, i" in the "asm" code represent sub-bits "dddd", "ssss" and "ii…" in the bit pattern (“d” and “s” are each 4 bits, and identify register numbers, while “i” is 16 bits and represents an immediate value). "gr[ ]" represents the register file. "number($N)" gets a register number from $N and "address($N)" gets the immediate value or label address from $N where "$N" (N = 3,4…) represents the Nth operand, counting the ",".

3.3 Web-based Processor Development Environment

To develop a processor in SSC-DASH, processor designers only need to upload a file. Figure 1 shows the generation flow of the SWDE.

(1) The C-DASH server stores the processor definition file that has been uploaded by the developer into a previously created project directory. (2) SSC-DASH calls C-DASH with the processor definition file as input. C-DASH generates source files for the simulator, assembler, disassembler and compiler. (3) SSC-DASH compiles the source files, including a GUI library, and generates Java class files. (4) SSC-DASH creates a JAR file from the Java class files generated previously. (5) Finally, SSC-DASH signs the JAR file with a previously prepared certificate.

3.4 Collaborative Processor Development Environment

To make web-based collaborative development possible, SSC-DASH is implemented using a Java servlet. The SSC-DASH servlet enables processor designers to develop a processor from anywhere in the world, using only a web browser and a text editor.

Processor designers access SSC-DASH via a web browser and upload a processor definition file onto the server. SSC-DASH then generates the SWDE as a signed Java Archive (JAR file) and prepares a JNLP file and a link on the web page for the JNLP file.

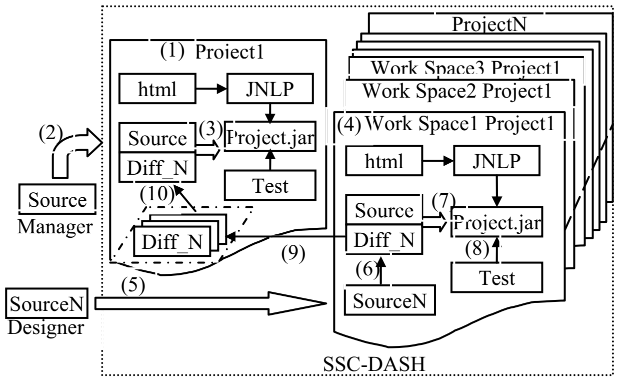

Fig. 2. Behavior of WISS

To implement a WCDE (web-based collaborative development environment), SSC-DASH includes:

- User Management
- Project Management
- File Management using CVS
- Access Control to files
- Information Management using BBS
- Library Management
- File Release System

The behavior of SSC-DASH can be described as: (1) Project manager creates a new project. SSC-DASH creates a project directory on the server machine and the necessary files (HTML and JNLP files) for the project are generated automatically. (2) The developer uploads a processor definition file (Source) which is then stored in the project directory. (3) SSC-DASH generates a JAR file (Project.jar) from the processor definition file. (4) Each designer creates a work space to develop additional features and/or fix bugs. (5) Once the designer has written and/or fixed features, these are uploaded to SSC-DASH via a web browser and stored in the directory. (6) The behavior of the uploaded features is checked and Diff_N is created by SSC-DASH. Diff_N is a difference file between the Source and SourceN. (7) Designer generates a new SWDE based on the updated processor definition file. (8) Designer checks the behavior of the SWDE with test programs and accepts the updated file if the behavior is correct. (9) Designer submits a new function and/or bug fix to the project manager. (10) After the functions submitted by each designer, have been checked for correct behavior, SSC-DASH releases the new SWDE.

4. Interface Development for ISS Using Ajax

4.1 Structure of WISS

Figure 3 shows the structure of a WISS (Web-based ISS) that contains the following components: Web server: Apache 2.0; Servlet Container: Tomcat 5.5.17; Server OS: CentOS 2.6.9-34.EL.

(1) A user accesses the web server. (2) The web server connects to the servlet container and (3) calls the SSC-DASH. (4) User uploads source files written in assembly language to verify behavior of processor. SSC-DASH then calls the SWDE with the source file. The SWDE assembles the source file and generates machine code and executes it. (5) The SWDE returns the execution results to SSC-DASH. (6)(7)(8) The web browser then loads the execution results and renews values asynchronously.

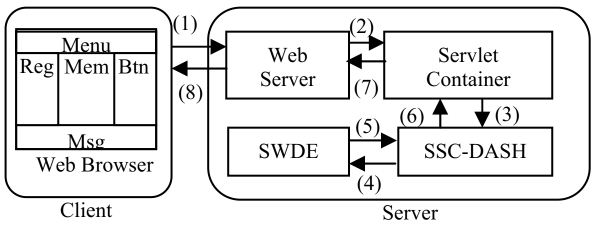

Fig. 3. Structure of WISS

4.2 The WISS Interface

The WISS interface is divided into 5 parts (***Menu***, ***Reg***, ***Mem***, ***Msg*** and ***Btn***), as shown in Fig. 3. The ***Menu*** shows information (such as processor name and related information) and a menu of features (such as open file, close file, save, etc.). ***Reg*** is a register viewer that shows the name and value of each register (such as general purpose, program counter, flag, etc.). At the center of the browser window is the memory viewer (***Mem***) that contains a check box for each execution step, address information, object and source codes. ***Btn*** contains the buttons for executing simulations and these are placed to the right. The system log and messages are displayed in the Message window (***Msg***).

4.3 Data Format

When execution of the simulation terminates or is interrupted, the SWDE returns the results of the simulation in XML format, the structure of which is given below.

```
<CPU name="CPU_NAME" source="FILE_NAME">
<REGISTERS>
  <REG id="REGI_NAME">REG_VALUE</REG>
    …
</REGISTERS>
<MEMORY>
  <ADDRESS id="ADDRESS_N">MEM_VALUE</ADDRESS>
    …
</MEMORY>
</CPU>
```

This XML format contains a CPU tag that contains REGISTERS and MEMORY tags. The REGISTERS tag contains REG tags with parameters for 'register name by id' (REG_NAME) and its value: REG_VALUE. The MEMORY tag contains ADDRESS tags with parameters for 'memory address by id' (ADDRESS_N) and its value: (MEM_VALUE).

Interactivity in the WISS interface is achieved using Ajax, which makes asynchronous communication possible without page transfers. As an example, given below is the communication code to obtain a register value.

Line 02 checks whether data transfer is successful. In line 04, register data is retrieved by Tag name. In line 06, the register name is retrieved, and in line 07, the register node value is obtained. Line 08 renews the register value.

The code to renew a memory value is almost identical to the above.

```
function displayXml() {
  if((httpObj.readyState == 4) && (httpObj.status == 200)) {
    xmlData = httpObj.responseXML;
    RegTags = xmlData.getElementsByTagName("REG");
    for(i=0;i<RegTags.length;i++) {
      name = RegTags[i].childNodes[0].nodeValue;
      value = RegTags[i].childNodes[0].nodeValue;
      document.getElementById(name).innerHTML = value;
    }
  }
}
```

5. Evaluation

The evaluation environment is given in Table 1. A signed SWDE JAR file including 16 bit words and 24 instructions was prepared and placed on both the Server1 and Server2. The file size was 73KB. Table 2 gives the results of our evaluation. We measured the elapsed time (JWS in Table 2) from clicking on the link to the JNLP file until loading of the SWDE window is complete. For comparison a direct link to the SWDE without using JWS was also timed (Download in Table 2). The ISS column in Table 2 gives the execution time to execute the SWDE on each host. The WISS column gives the execution times of the web-based instruction set simulation.

Use of the "*" in Table 2 denotes that less than one second ISS execution time for the client is required to compare local ISS execution and WISS. The execution times of the ISS are low because the test programs are small. The execution times for the WISS also depend on network traffic. Based on the results shown here, it is clear that the WISS is very useful.

Almost all assembly language programs that verify the behavior of defined processors have LOC (Line of Code) footprints of less than 30 lines. This is because these programs are not actual application programs but test programs that check the defined instruction behavior.

Furthermore, the web-based interface for the simulator improves simulation performance in web-based processor development environments. It is also possible to use web-based instruction set simulators for educational purposes. As it is difficult for inexperienced PC users to install application software, the simulator can be used without installation of any software.

6. Conclusion

In this paper, we have described the interface for a web-based instruction set simulator in a web-based collaborative processor development environment.

We have developed the web-based collaborative PDE called SSC-DASH. SSC-DASH obtains the processor definition files via a web-browser, and generates a SWDE including GUI, simulator, assembler and disassembler. The SWDE is written in Java, and adapted for JWS. JWS is suitable for application distribution because version checks, downloads and installation of the application are performed automatically. JWS is, however, not suitable as a test environment in a PDE, where the processor definition files are uploaded frequently.

Table 1. Evaluation Environment

	Server1	Server2	Client
CPU	500MHz (Pentium 3)	1.0GHz (UltraSPARC T1)6/24	1.8GHz (Core2Duo)
Memory	128MB	4GB	1GB
LAN	100BT	1000BT	100BT
OS	Cent OS	Solaris	Windows XP
Browser			IE7

Table 2. Results of Execution

	JWS (sec)	Download (sec)	ISS (sec)	WISS
Server1	7~10	1*	1*	1*
Server2	7~10	1*	1*	1*
Client	N/A	N/A	1*	N/A

Version check, download and installation time are required every time the processor definition file is uploaded even if the change is to the footprint. So, the version check, download and installation can become very time consuming in the test environment.

By implementing the interface for the web-based instruction set simulator using Ajax, it is possible to eliminate the time consuming part (the version check, download and installation of the SWDE). True interactive simulation is then possible.

In future works, we aim to enhance the functions of the collaborative development environment.

References

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

[2] Hideaki Yanagisawa, Minoru Uehara, Hideki Mori, "Web-based Collaborative Development Environment for an ISA Simulator", In Proc. of Fourth International Conference on Cooperative Internet Computing (CIC 2006), pp.142-151, 2006.

[3] Hideaki Yanagisawa, Minoru Uehara, Hideki Mori, "Server Side C-DASH : Ubiquitous Hardware/Software Codesign Environment", In Proc. of IEEE International Workshop on Future Mobile and Ubiquitous Information Technologies (FMUIT'06), (CD-ROM), Nara, Japan, 2006.

[4] Java Web Start Guide, http://java.sun.com/javase/6/docs/technotes/guides/javaws/developersguide/contents.html

[5] Java Network Launching Protocol and API, http://jcp.org/aboutJava/communityprocess/mrel/jsr056/index2.html

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

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

[8] CoWare Processor Designer, http://www.coware.com/products/processordesigner.php

[9] ASIP Meister, http://www.asip-solutions.com/index.html

[10] GNU Compiler Collection (GCC) Internals, http://gcc.gnu.org/onlinedocs/gccint/