A JAVA FRAMEWORK TO TEACH COMPUTER ARCHITECTURE

Ricardo S. Ferreira[1], Antônio Carlos S. Beck[2], Luigi Carro[2], Andre Toledo[1], Aroldo Silva[1]

[1]*Depto de Informatica, Universidade Federal de Vi?osa*
36570-000, Vicosa, Brazil, Phone: +55 31 3899 1761, Fax +55 31 3899 2394
cacau@dpi.ufv.br

[2]*Instituto de Informatica, Universidade Federal do Rio Grande do Sul*
Av. Bento Goncalves, 9500, Porto Alegre, Brazil
caco@inf.ufrgs.br, carro@inf.ufrgs.br

Abstract: This work proposes a Java-based framework to teach computer architecture design. Our methodology allows students rapidly to explore many different concepts across multiple research and design domains. Our environment is based on Hades editor/simulator tool by showing how to profite from the Java portability and reusability, open source component behaviour description, and VHDL generation. In addition, we have improved Hades tool by increasing the component library, by adding a power consumption estimation, and by performing program behaviour instrumentation. Our approach have been validated by showing two processor examples. Morevover, we have used real workloads to show simulation performance and flexibility.

Keywords: Teaching Tool, Computer Architecture, Behavior Description, Java

1. INTRODUCTION

A digital design could be specified by its behavior and/or structural description. Schematic designs, state diagrams, hardware languages (VHDL,Verilog) or programming languages (C,C++,SystemC) are the most common specifications. There are a wide range of computer aided tools (Bormida et al., 1997; Figueiredo et al., 2001; Reddi et al., 2004) at both graduate and undergraduate teaching levels. (Bormida et al., 1997) shows that interactive tools and exercises are more effective than hypertext and visual animations. In addition, a simple tool is more suitable than a complex CAD one for the student to learn a specific subject (Bormida et al., 1997). Moreover, thanks to the fast technology improvements, a teaching environment should provide a dynamic and incremental structure to handle new architecture models and paradigms.

Based on these characteristics and constraints, this work proposes a framework to teach computer architecture design by extending a Java-based editor/simulator tool, called Hades (Hendrich, 2000), which have been developed at Hamburg University, since 1998. Our framework uses Java to issue portability, reusability, a design teaching and exploration. Java is used to specify the architecture and Register-level component behavior, to perform the simulation, and to perform program behaviour analysis. Our methodology allows students rapidly to explore many different concepts across multiple research and design domains. Furthermore, as power dissipation is an essential skill for both high performance desktop processor and mobile processors for embedded systems in these days, we also propose an extension to the Hades component library to handle the power consumption issue. In addition, we have improved Hades tool by showing how the student can easily perform a program behavior instrumentation. Our approach was validated by developing three processor examples: Java Microcontroller and PIC microcontroller as a single component in a high level of abstraction, and a Java Microcontroller processor at Resgister Transfer (RT) level. We use real workloads to show simulation performance and flexibility.

Section 2 introduces related work and contrasts it with our work. The methodology used is briefly outlined in Section 3. Section 4 illustrates how three practical applications can be modeled by using our environment. Finally, Section 5 exposes the main results and on-going work.

2. RELATED WORK

The first task when one has to develop a framework is to look around and examine carefully the available tools. Hades tool (Hendrich, 2000) have been chosen because it is simple, portable and dynamic, as a teaching tool must be. Hades is a pure-Java component-based simulator, and includes as main features: an user friendly interface, a discrete-event based simulation, an interactive and batch simulation (no recompilation is needed after changes), and a flexible waveform viewer. Hades is used for modeling and simulating at different design levels, from gate to high architecture levels. In addition, recent work (Marwedel and Sirocic, 2003; Ferreira et al., 2004; Rodrigues and Cardoso, 2005) have shown that Hades-based environment can handle many computer architecture subjects, and still have a good performance and easy interaction with tools and environments. Another Java-based environment is JHDL (Hutchings et al., 1999), which is a structurally based Hardware Description Language (HDL) implemented with Java. Our environment differs from JHDL structural approach, and proposes a behavior and/or structural description.

Also, visual(Marwedel and Sirocic, 2003; Uy et al., 2004) and quantitative simulators (Branovic et al., 2004) are available to help students to better understand the theorical concepts. Recently, (Marwedel and Sirocic, 2003) presented a Hades-based tool, called Ravi, to teach the basis of dynamic behavior of processors by using the following examples: microcoded MIPS, pipeline MIPS, the scoreboarding and tomasulo algorithm and multiprocessor protocol. However, Ravi focuses on visualization features, and it is not trivial for the student neither to change a current design nor to create a new one. Although Ravi uses the Hades simulation engine and interface, it uses its own library. This means that Ravi components cannot interact with Hades components and, as a consequence, the components of each library are not interchangeable. The main drawback is that the student can not reuse the rich parametrical RT component library of Hades. Furthermore, if the student would like to add a new forwarding line by adding an extra multiplexer, for instance, all control units must be written. This can be a complex task, depending on the design. Moreover, the source code of Ravi is not available for public use. This way, we propose a new approach by extending the large Hades library and by using an open source approach. This allows the student to rapidly create new design and new components thanks to the environment reusability and object-oriented paradigm.

In embedded systems, the power consumption evaluation in early stages of the design flow is fundamental for the search of optimal solutions within the design space. Recently, a cycle-accurate and configurable simulator was presented by (Beck et al., 2003), called as CACO-PS (Cycle-Accurate COnfigurable Power Simulator). CACO-PS is an ANSI C component-based simulator, and includes as main features: a structural system description at different levels of abstraction, a behavior and power estimation component description, and some other measures (execution time, memory footprint, number of instructions, etc). We propose an enhanced power estimation simulator based on Hades. Our approach presents an object-oriented behavior component and power description, an graphical interface including waveforms, an event-oriented simulator, a VHDL generation, and much more. Although Hades is Java-based, the preliminary results have shown that the simulation time is comparable to CACO-PS performance. Furthermore, Hades signals follows the VHDL standards, and a tri-state bus can be easily modeled. CACO-PS estimates the dynamic power consumption of the system based on the switching activity of the components, limiting the analysis to single bits in the component inputs. Hades can also handle dynamic power consumption estimation during a clock cycle. However, Hades can make the power consumption estimation of the components also in higher levels of abstractions, such as analyzing vector, integers, string or even JPEG images, for example.

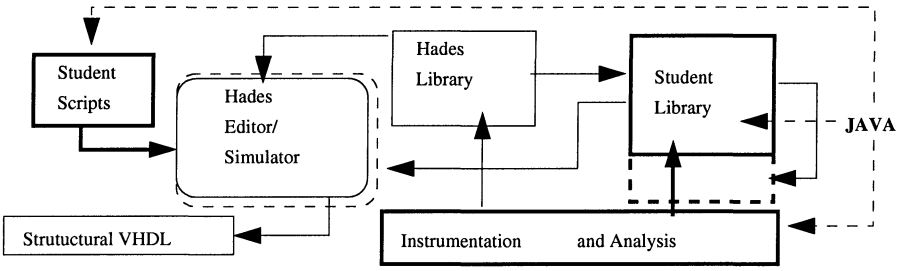

Figure 1. A Java-based Student Design/Simulator Environment

3. METHODOLOGY

We propose an interactive methodology, where the student can work in different concepts: architecture visualization and design, simulation environment and hardware event monitoring programs. We also propose a visual and Java-based approach to reduce the time spent by the student to learn our environment, and all concepts are integrated in a unique tool and language. Our approach focuses on component modeling, high level behavior description and monitoring. Moreover, this approach is time-saving because the students do not need a deep understanding of the simulation tool.

First, during the begining of the undergradutate course, the student can take advantage of a large component library and basic design available in Hades. The graphical interface and WEB tutorial provide an easy interation to enhance the student understanding. These designs can be at transistor level, gate level, lower and higher architecture level. Then, the students can start to build their own components, by writing few Java source lines thanks to the reuse concept of the object-oriented paradigm. In addition, the simulation can be interactive or managed by student scripts written in Java language. A VHDL script syntax-like and a quantitative approach are also available. Furthermore, the student can write their own instrumentation tools. Moreover, the student take benefit from Java resources and libraries to analyze and to resume the main results, as well as to avoid large trace files. Finally, the student could create more complex designs and components by reusing the previous work.

Different logic and architecture levels can be explored. Even at RT level, a fast simulation performance could be reached when compared against VHDL simulation. In addition, the Java behavior description provides a faster modeling. The environment could also export structural VHDL to commercial tools, as shown in Figure 1. However, a VHDL file for each component should be written or imported from another tool. At gate-level, a VHDL library is al-

ready available. The following sections will detail the component modeling, the simulation features, and the architecture and power instrumentation.

Student Library

Figure 1 shows the student incremental library development. A new component can be written by only specifying the I/O number, and their types and names. In fact, the student is just extending the component library. Then, the behavior is specified by the *evaluate* method. A Java library is available to handle bit and bit vector data. The following source code describes the behavior of a n-bit adder/sub which depends on a Select input. The output is scheduled after a delay that can be configured by the user. The *evaluate* method is called if an event arrives at the input ports A and/or B. The student has only to worry about the development of the component, since the simulator engine is automatically managed by Hades. Every component is an object and will be called dynamically during the simulation.

```
public void evaluate(Object arg) {
  /*
   * C = A + B if Select=0
   * C = A - B otherwise
   */
  StdLogicVector A   = PortA.getValue();
  StdLogicVector B   = PortB.getValue();
  StdLogic1164 Select = PortSelect.getValue();
   if (Select.is_1())
      vector = A.sub(B);
   else
      vector = A.add(B);
  double time = simulator.getSimTime() + delay;
  scheduleOuput(PortC,vector,time);
}
```

Simulation

During the simulation process, the student can interact by using various modes. First, a mouse-driven event can be used to apply a signal, for instance: a virtual keyboard (see Fig. 2), a vector or bit signals (see Fig. 2), or even a graphical interface (e.g.: menu option, action button, scrool bars, see Fig 3). A Java, Jynthon or VHDL-syntax script can automatically controls some or all of the input signals during a simulation process. The simulation can also run in batch mode. A graphical waveform window (see Fig. 4) can be opened to view a subset of signals, or all of them. The simulation trace file can be saved, and the waveform window has browser zoom features. Moreover, the

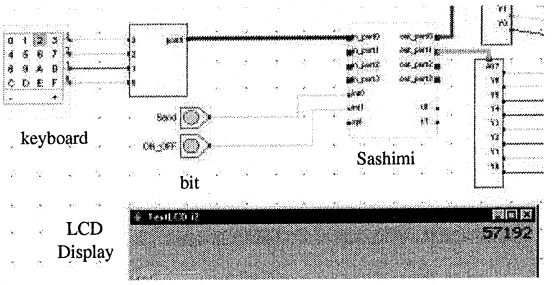

Figure 2. Calculator Design: A Simulation in Sashimi Microcontroller

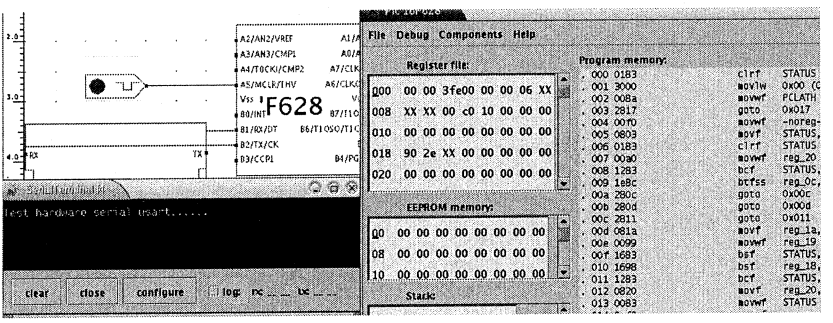

Figure 3. PIC16F628, a graphical window, and Serial Terminal window

components can have associated tool tips and/or component animation features (e.g. colors, draws or texts). For instance, Figure 4 illustrates the current bytecode name instruction and cycle number of FSM microcode control unit which was generated by the following extra lines in evaluate method:

```
public void evaluate(Object arg) {
    . . .
    String S = instructionName + " " + clkCounter ;
    Figcounter.setText(S); // add text to graphical animation
    showState();  // update the text
    . . .
}
```

The next section will show how to add some code instrumentation, as *instructionName* and *clkCounter* variable from the previous source example.

Program Behavior

The instrumentation is a useful resource to better understanding the processor skills. Learnning and writing a simulator could be a complex task to be developed during a undergraduate course. Due the high complexity, (Reddi

et al., 2004) have shown that computer architecture education should handle program behavior and real workload by using effective tools. Our approach is similar, however we use a object-oriented language. For instance, if the student would like to count how many RAM write accesses have been done in a stack based machine, as shown in section 4, only few lines have to be added in RAM class object file, as follows:

```
public class ram extends rtcomponent {
  protected int write_counter = 0;
  . . .
  MEM_RAM[address] = current_data; // perform a write
  write_counter++; // a counter is added !
  . . .
}
```

Thanks to the object-oriented paradigm, the student can also create a new component by extending a previous one, without knowing any implementation details. The instrumentation can be performed as follows:

```
public class studentram extends ram
 {
  protected int writecounter = 0;
   public void evaluate(Object arg)
  {
    super.evaluate(arg);
    StdLogic1164  CLK =  portCLK.getValueOrU();
    StdLogic1164    RW = portRW.getValueOrU();
    if (CLK.hasEvent() && CLK.is_1()  &&  RW.is_1())
      writecounter++;
  }
 }
```

As we can see, a new instrumentation component can be created by writing a Java class with few lines of code. Moreover, some additional source could be implemented to avoid large trace file and to view a post processing analysis, like a graphical histogram, at the end of the simulation.

Power Estimation

As CACO-PS (Beck et al., 2003), our approach is a cycle-accurate and configurable power simulator. The design environment is not restricted to a single architecture, as most instruction-level or compiled power simulators. The accuracy is also guaranteed by a component based simulation with power models based on the switching activity at the component inputs. Furthermore, as CACO-PS, our approach allows the definition of architectural components at different levels of abstraction (from transistors to whole processors). This

means that it is not necessary to have a RT description of the system. However, our approach improves CACO-PS by using object-oriented class and all component power behavior is handle inside a single component Java file. In CACO-PS approach, the designer have to handle the component file and the power file, and have to know some simulator engine details. Our approach allows the student to add some lines to a library component or to create a new one by using inheritance. For instance, the following source extends an n-bit register component to measure the cycle-driven power:

```
public class powerreg extends reg {
  public void evaluate(Object arg) {
      super.evaluate(arg); power(arg);
  }
    public void power(Object arg) {
     if (CLK.hasEvent() && CLK.is_1())
       if (Enable.is_1()) {
         power += StaticRegCapacitanceGate;
         if ( HasInputTransition(portDATA) )
           power += InputTransition(portDATA)
                  * DynamicRegCapacitanteGate;
       }
}
```

In addition, the Hades library has already more than three hundred components, and by using inheritance, it can be extended to measure power at different levels (from transistors to microcontrollers).

4. EXAMPLES

A Popular Microcontroller

This section presents the PIC 16F628 microcontroller from Microchip©. The Hades library has only the PIC 16F84. We have implemented the 16F628 by extending the 16F84. The graphical window (see Fig. 3), and instruction core simulator have been kept. The new features have been implemented by extending the Pic register class. For instance, the timer1 source code has only 65 lines. We have implemented: hardware serial USART, 3 timers, hardware PWM, hardware compare/capture registers, dual onboard comparators, and programmable voltage reference. We propose to use integer Hades signal to handle analog values. Figure 3 shows a screenshot and the graphical PIC interface which was inheritance. The simulation runs very fast, around 500.000 cycles per second. We have validated our PIC by using C source examples (compiled by Piclite Demo tool from Hitech©).

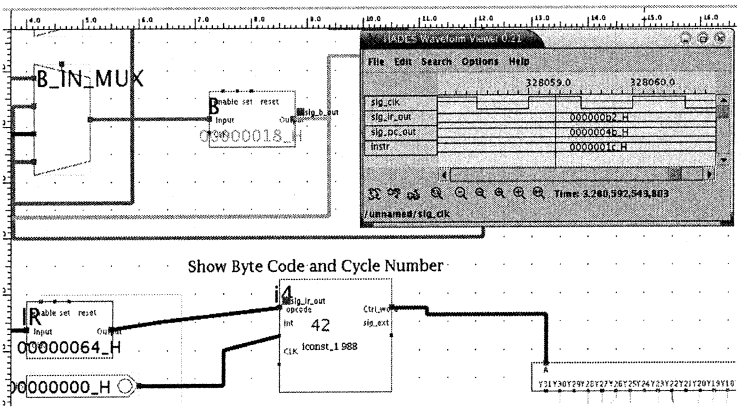

Figure 4. Fetmo Java processor at Register Tranfers Level

FemtoJava Processor

This section presents an high level and RT-level example of Java microcontroller (Ito et al., 2001) which have been integrated to our environment. Figure 2 illustrates a design screenshot where the high level microcontroller is connected to an external keyboard and a LCD display. The microcontroller component can load a Java source code built by the student and perform the simulation. This example shows a calculator design. The microcontroller is like a black box and the simulation runs very fast.

We have also implemented the Multicycle FemtoJava Microcontroller (Ito et al., 2001) at RT-level. The design has 35 component instances and 95 connections. Figure 4 illustrates a subset of the graphical design and shows some visual simulation features. This design is compatible with the VHDL version (Beck et al., 2003), and a structural VHDL can be generated. The student can run a step-by-step simulation, in order to understand the microcode operations. A script can start and stop the simulation in a specific break-point. Moreover, we have implemented some power estimation and instrumentation lines to perform a quantitative design analysis. Table 1 shows the performance in number of cycles, number of byte-code instructions, number of branches, number of memory writes, and FSM power estimation for each application. The benchmark set is composed of four sort algorithms and four versions of IMDCT (Inverse Modified Discrete Cosine Transformation), an important part of the MP3 decompression algorithm. The u1,u2 and u3 are unrolled versions of the IMDCT. Even at RT-level, the simulation is still fast, around 5.000 cycles per second. The execution time was under 25 seconds for all applications. The instrumentation allows a quantitative behavior analysis of different sort algorithms and loop unrolled versions.

Table 1. Instrumentation Table

bench	Cycle	Inst	Branches	Write	power
Sort/Bubble	6775	1124	123	920	227690
SortSelect	5347	955	123	697	187630
SortInsert	4094	709	67	546	138960
QuickSort	3941	695	104	542	139320
IMDCT	140300	25479	913	17142	4737100
IMDCTu1	97354	19893	59	12361	3526890
IMDCTu2	92882	18695	41	11747	3335480
IMDCTu3	51346	7312	8	5976	1537240

5. CONCLUSION

This work present an open source Java-based framework to teach computer architecture design, which is also suitable to e-learning courses. We propose a simple and powerful Computer-Aided Design and Computer-Aided Learning environment. Our approach is based on Hades, which is able to locate and load simulation models and accompanying software both from the local file system or the WEB. By using a portable object-oriented language and a simple and unified environment, the students can rapidly explore many different concepts across multiple research and design domains. We have shown how to create a new component library, how to perform code and RT instrumentation, and power estimation. The student can work from high modeling levels to RT-levels, and lower levels are also available. We have used real workloads (sort and IMDCT) to show simulation performance and flexibility. On-going work is also conducted to implement different versions of Java processors from (Beck and Carro, 2005), which including a five-stage pipeline, four VLIW versions, and a reconfigurable architecture.

REFERENCES

Beck, Antonio C. S. and Carro, Luigi (2005). Dynamic reconfiguration with binary translation: Breaking the ilp barrier with software compatibility. In *42th Design Automation Conference*, California, USA.

Beck, Antonio C. S., Mattos, Julio C. B., Wagner, Flavio R., and Carro, Luigi (2003). Cacops: A general purpose cycle-accurate configurable power simulator. In *16th Symposium on Integrated Circuits and Systems Design (SBCCI'03)*, page 349, São Paulo, Brazil.

Bormida, Giorgio Da, Ponta, Domenico, and Donzellini, Giuliano (1997). Methodologies and tools for learning digital electronics. *IEEE Transaction in Education*, 40(4).

Branovic, Irina, Giorgi, Roberto, and Prete, Antonio (2004). Web-based training on computer architecture: The case for jcachesim. In *Proceedings of the 5th International Workshop on Computer Architecture Education.*

Ferreira, Ricardo, Cardoso, João M. P., and Neto, Horácio C. (2004). An Environment for Exploring Data-driven Architectures. In Springer-Verlag, LNCS, editor, *14th International Conference on Field Programmable Logic and Applications*, pages 1022–1026.

Figueiredo, Renato J., Fortes, Jose A. B., Eigenmann, Rudolf, Kapadia, Nirav H., Taylor, Valerie, Choudhary, Alok, Vidal, Luis, and Chen, Jan-Jo (2001). On the use of simulation and parallelization tools in computer architecture and programming courses. *Computers in Education Journal*, 11(1).

Hendrich, N. (2000). A Java-based Framework for Simulation and Teaching. In Publishers, Kluwer Academic, editor, *3rd European Workshop on Microelectronics Education*, pages pp. 285–288, Aix en Provence, France.

Hutchings, Brad L., Bellows, Peter, Hawkins, Joseph, Hemmert, Scott, Nelson, Brent E., and Rytting, Mike (1999). A cad suite for high-performance fpga design. In *FCCM*, pages 12–24.

Ito, Sérgio A., Carro, Luigi, and Jacobi, Ricardo (2001). Making java work for microcontroller applications. *IEEE Design & Test of Computers,*.

Marwedel, Peter and Sirocic, Birgit (2003). Multimedia components for the visualization of dynamic behavior in computer architectures. In *Workshop on Computer Architecture Education (WCAE)*, San Diego, CA Sunday, June 8.

Reddi, Vijay Janapa, Settle, Alex, Connors, Daniel A., and Cohn, Robert (2004). Pin: A binary instrumentation tool in computer architecture research and education. In *Proceedings of the 7th International Workshop on Computer Architecture Education*, Germany.

Rodrigues, Rui and Cardoso, João M. P. (2005). An infrastructure to functionally test designs generated by compilers targeting fpgas. In *DATE 2005*, pages 30–31.

Uy, Roger Luis, Bernardo, Marizel, and Erica, Josiel (2004). Darc2: Second-generation dlx architecture simulator. In *Proceedings of the 7th International Workshop on Computer Architecture Education*, Germany.