Workshop on Computer Architecture Education
Saturday, June 17, 2006

Organizer: Ed Gehringer, North Carolina State U.

Program Committee:
Jim Conrad, University of North Carolina-Charlotte
Richard Enbody, Michigan State University
Mark Fienup, University of Northern Iowa
Nirav Kapadia, Independent consultant
Naraig Manjikian, Queen's University (Canada)
Aleksandar Milenkovic, University of Alabama-Huntsville
Yale Patt, University of Texas-Austin
Murray Pearson, University of Waikato
Antonio Prete, Università di Pisa
Kenny Ricks, University of Alabama
Anujan Varma, University of California, Santa Cruz
Wayne Wolf, Princeton University
Craig Zilles, University of Illinois

Session 1. Welcome and Keynote, 9:00–10:05

9:00 Welcome, Edward F. Gehringer, workshop organizer
9:05 Keynote, “A systems approach to teaching computer systems,” Frans Kaashoek, MIT .......... 2

Session 2. Embedded systems, 10:05–10:30

10:05 Experiences with the Blackfin architecture in an embedded systems lab, Michael Benjamin, David Kaeli, and Richard Platcow, Northeastern University .................................................. 3

Break 10:30–11:00

Session 3. Simulation environments 11:00–12:30

11:00 “PSATSim: An interactive graphical superscalar architecture simulator for power and performance analysis,” Clint W. Smullen, IV and Tarek M. Taha, Clemson University .......... 10
11:25 “An execution-driven simulation tool for teaching cache memories in introductory computer organization courses,” Salvador Petit, Noel Tomás, Julio Sahuquillo, and Ana Pont, Polytechnic University of Valencia ............................................................................................................. 18
11:50 Web memory hierarchy learning and research environment, José Leandro D. Mendes, Luiza M. N. Coutinho, and Carlos A.P. S. Martins, Pontifical Catholic University of Minas Gerais ............... 25
12:15 Discussion

Lunch 12:30–2:00

Session 4. Special Session on Writing an International Textbook, Denise Penrose 2:00–3:00

Break 3:00–3:30

Session 5. Regular Papers 3:30–4:20

3:30 Remote operation and control of computer engineering laboratory experiments, Helmut Bähring, Jörg Keller, and Wolfram Schiffmann, FernUniversität Hagen ................................................................. 33
3:50 “Students’ Experimental Processor: A processor integrated with different types of architectures for educational purposes,” L. S. K. Udugama and Janath C. Geeganage, Open University of Sri Lanka 41
4:10 Discussion

A systems approach to teaching computer systems

Jerry Saltzer and Frans Kaashoek
Massachusetts Institute of Technology
{saltzer,kaashoek}@mit.edu

At M.I.T. we teach a class titled "computer system engineering", a required class that provides an introduction to computer systems. It provides a broad and in-depth introduction to the main principles and abstractions of engineering computer systems, be it an operating system, a client/server application, a database application, a secure Web site, or a fault-tolerant disk cluster. These principles and abstractions are timeless and are of value to any computer science or computer engineering student, whether specializing in computer systems or not.

Many fundamental ideas such as design principles, modularity, abstraction, virtualization, concurrency, communication, fault tolerance, and atomicity are common to several of the upper-division electives of the computer science curriculum. Most curricula start with two beginning subjects (introduction to software and hardware) and then branch out to: operating systems, networks, database systems, distributed systems, programming languages, software engineering, etc. The problem is that this list has grown over the last 20 years and there isn’t time for most students to take all or even several of those subjects. The typical response is "choose three" or "take operating systems plus two more". The result is that most students end up with no background at all in the remaining topics.

As a solution, we have developed a text that cuts across all of these subjects, identifying common mechanisms and design principles, and explaining in depth a carefully chosen set of ideas in each area. This approach provides an opportunity to teach a core undergraduate subject that can be taken by all computer science students, followed by the same set of electives listed above, but those electives can now explore their topic more effectively and more thoroughly. We found this course structure to be effective at M.I.T., and it has been enthusiastically received by the students.
Experiences with the Blackfin Architecture in an Embedded Systems Lab

Michael Benjamin, David Kaeli, Richard Platcow
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115

Abstract

At Northeastern University we are building a number of courses upon a common embedded systems platform. The goal is to reduce the learning curve associated with new architectures and programming environments. The platform selected is based on the Analog Devices Blackfin digital signal processor.

In this paper we discuss our recent experience developing anew undergraduate embedded systems lab. Students learn to utilize the embedded DSP platform to address a number of different applications, including controller design, RS-232 communication, encryption, and image processing. This platform provides a rich design exploration sandbox replete with programming and simulation tools. We describe our use of this platform in our Microprocessor-based Design Laboratory and discuss how this platform can be used in a range of classes.

I. INTRODUCTION

At Northeastern University, our electrical and computer engineering students pursue 5-year B.S. degrees and gain workplace experience with up to 1.5 years of cooperative education. To better prepare our students for the hands-on work of the real world, we are building a common educational platform allowing development in real-world systems, while reducing the learning curve associated with working on a new platform.

Such a platform must support a range of classical topics including signal processing, controls, computer architecture, and embedded systems. But the platform’s tools must also be easy to use, and the associated documentation and pedagogic material must be sufficiently rich. Additionally, the platform must be practical enough that implementations using it at least hold the attentions of students and at best ignite their imaginations.

We have selected the Blackfin architecture from Analog Devices (ADI) to provide such a platform. The Blackfin combines the functionality of a digital signal processor with that of a micro-controller unit. The architecture supports a number of tool-chains, operating systems, and development environments.

In 2003, our Digital Signal Processing class began using the ADI Blackfin. Based on very positive student feedback with this platform, we have begun to deploy this platform in a number of classes. In the Summer/Fall of 2005, our Microprocessor-based Design lab was redesigned to use the Blackfin. The following Spring 2006 semester, the revised lab was offered for the first time. The lab presented the concepts of performing basic I/O, micro-control program design, RS-232 communication, encryption, and image processing.

It is our goal for this lab to provide a platform upon which our students quickly build. Our experience to date is that students taking the DSP course that utilizes this platform in the lab, utilize this same set of tools to implement their senior capstone projects.

The objective of this paper is to describe our very positive experiences with this platform, and to discuss our future plans to build off this experience by replicating it in other labs. The rest of the paper is organized as follows. Section II discusses related work. Section III gives a brief introduction to the Blackfin processor architecture. Section IV describes the hardware platform and the supporting materials for the lab. Section VI concludes the paper and Section V describes how this model can be extended to future labs.

II. RELATED WORK

ADI supports academic programs through their University Program Initiative [1]. This program makes available a series of online training modules, including videos, PDF transcripts, presentation slides, and some code examples; workshops and seminars in North America, Europe, and China; and training publications [2]–[4].

A number of universities around the world have begun to utilize the ADI Blackfin EZ-KIT platform to support their classes. EZ-KIT evaluation boards integrate a Blackfin processor with a range of peripherals. The ECE-ADI Project at University of Calgary has a rather extensive collection of audio, video, microcontroller labs, and presentation materials [5]. The University of Massachusetts Lowell new Handy Board [9] utilizes the Blackfin. A one-day course at the University of Parma, Italy provides hands-on experience using the Blackfin [6]. California Polytechnic State University has utilized the Blackfin in a course and has also described how it could be used to revamp an entire curriculum [7]. This most recent adoption has motivated the curricular changes at Northeastern to adopt the Blackfin EZ-KIT.

III. BLACKFIN PROCESSOR

The Blackfin was designed to provide micro-controller (MCU) and digital signal processing (DSP) functionality in a single processor, while allowing flexibility between the needs of control and DSP. With this duality in mind, the Blackfin...
incorporates a Single Instruction, Multiple Data (SIMD) processor with features such as a variable-length RISC instructions, software-programmable on-chip PLL, watchdog timer, real-time clock, memory management unit, 100 Mbps serial ports (SPORTs), UART controllers (with IRDA support), and SPI ports. The MMU supports multiple direct memory access (DMA) channels for data transfers between peripherals and SDRAM, FLASH, SRAM memory subsystems, and also supports configurable on-chip instruction and data caches. The Blackfin hardware supports 8-bit, 16-bit, and 32-bit arithmetic operations - but is optimized for 16-bit operations.

**A. Architecture**

The Blackfin architecture is based on the Micro Signal Architecture [15] developed jointly by Analog Devices (ADI) and Intel, which includes 32-bit RISC instruction set and 8-bit video instruction set with dual 16-bit multiply-accumulate (MAC) units. ADI has been able to achieve a balance between the needs of DSP and MCU with the Blackfin instruction set architecture. Using C/C++, a developer can rely on the compiler to generate highly dense yet computationally efficient code; or the developer can write targeted assembly. For real-time needs, operating system support becomes critical - the Blackfin supports memory protection and supports a number of operating systems.


Blackfin comes in both single-core (e.g., BF533, BF535 and BF537) and dual-core (e.g., BF561) models. The organization of the BF561 used in our Microprocessor-based Lab is shown in Figure 1. This chip provides for symmetric multiprocessing, while also providing low power consumption.

**IV. LAB ENVIRONMENT**

The Microprocessor Lab course was co-requisite to the Microprocessor-based Design course, which met twice weekly for 100-minute lectures. The lab section consisted of 15 undergraduates, most in their junior year. The students worked in groups of 3 or 4 in the laboratory.

The course was split into 5 individual experiments over a 14-week semester - each lab consisting of two 2-hour sessions. A sixth student-designed lab was also assigned, in which students proposed a project that could easily lead to an engineering capstone design project (all seniors are required to complete a capstone project in their senior year). The labs were conducted every other week to provide students ample time to write up their reports. In addition to the textbook [16] used in the course, a number of hardware and software manuals were used from ADI [17]–[20].

**A. Goals of the Lab**

In addition to our fundamental goal to strengthen students’ comprehension and retention of lecture material, additional goals for the lab course were:
To balance breadth and depth of embedded systems education.
To encourage students to develop skills for problem solving, group/time management, and self-learning.
To prepare students for technical writing in industry and academia.

To balance breadth and depth, we chose to use the C language and the VDSP++ IDDE. Assembly is discussed in the course lectures, and we show in lab how students can use inline assembly in C subroutines. To encourage independence in problem solving and learning, students were assigned a student-design lab and used Wikipedia articles as reference material. To strengthen technical writing, each report had an “executive summary” generally describing the lab and a “technical abstract” to address such detail.

B. Equipment

In this lab we utilize the Blackfin BF561 EZ-KIT Lite evaluation board (shown in Figure 2). The BF561 board integrates an ADSP-BF561 symmetric dual-core processor designed for consumer multimedia into an evaluation board which includes 64MB of SDRAM, 8MB FLASH, stereo audio channels (2 input/3 output), video jacks (3 input/3 output), UART/RS232 line driver/receiver, 20 LEDs, 5 push buttons (1 reset, 4 programmable flags), JTAG ICE 14-pin header and expansion interface.

Our lab setup includes an Ontrak ADR101 serial data acquisition interface board. This allows us to interface the BF561 with a relay using a RS-232 protocol and a basic string-as-command interface (see Figure 3).

C. Programming Environment

We utilize the ADI VDSP++ programming framework in the lab, shown in Figure 4. This framework is quite extensive and we are only able to touch on the basics in this lab (in the future we are looking to utilize this platform in a freshman programming class). Section V provides a discussion of how
advanced labs might further exploit the capabilities provided by VDSP++.

D. Experiments
Next we describe the series of experiments assigned to the students. They are presented in the order in which they are assigned.

1) Basic Input/Output: This lab introduces how to use the programming tools provided in order to produce input and output. In the first part of the lab, the students develop a simple C program for standard I/O. In the second part, they evaluate a definite integral using the Monte Carlo method. In the third part, students use push-buttons and LEDs on the BF561 EZKIT Lite board to implement simple board-level I/O. The students use Monte Carlo simulation code [21] to evaluate\[\int_{-1}^{0} e^{-\frac{x^2}{2}} dx\]corresponding to the probability density function for a Gaussian distribution (without the scaling factor $\frac{1}{\sigma\sqrt{2\pi}}$, where $\sigma^2$ denotes the variance). This exercise is intended as a demonstration of the available C functionality on the Blackfin and also as a conceptual introduction to the theoretical topics of stochastic and numerical methods.

To demonstrate simple I/O, one group’s solution was to develop a simple game similar to Simon. The game controls two LEDs in the bottom row, and are located close to a pushbutton. The game awards a point to the user if the user hits the correct button in time, and keeps score in binary on the top row of the LEDs.

A solution was given to the students that featured a tennis game. LEDs light one row at a time in succession, moving from one side to another. The direction changes as the appropriate racket button is pressed.

2) Washing Machine Controller: This lab covers how to implement an abstract finite state machine (FSM) model of a washing machine in an embedded controller design. In the first step, the students produce the FSM and implement it in C, using stdin and stdout to prototype sensors and control devices. An example FSM is given in Figure 5.

In the second step, students revise their I/O function to use a UART controller library which sends strings to a Windows XP Hyper Terminal session via an RS-232 interface. Finally, the students build a washing machine prototype using a breadboard, LEDs for outputs to the controller (such as valves to supply and drain water, and a motor to agitate the load) and switches for inputs to the controller (e.g., water-level sensors, door open/close sensors, etc.).
3) **RS-232 Communication:** In this lab students use the UART controller on the BF561 to interface with the ADR101 and the breadboard washing machine prototype they built in the previous lab. To verify their designs, the program writes the correct ADR commands to a Hyper Terminal session. In their reports, the students were asked to explain the RS232/UART communication step-by-step.

4) **Encryption:** In this lab, the students implement an RSA cryptography scheme that encrypts a text file and sends the encrypted characters from one BF561 to another. The data is then decrypted and displayed on the VSDP++ console. To encourage self-learning, the students were instructed to read Wikipedia articles that explained the overall RSA algorithm and numerical methods for exponentiation.

The basic RSA cryptography scheme discussed is summarized as follows. To transform a character with the ASCII value of $x$ into an encrypted number $c$ we can use $r$ (the public exponent), and $n$ (the public modulus, which is the product of two random, private prime numbers $n = p \times q$) using the following equation:

$$\text{encrypt}(x) = x^r \mod n = c \quad (1)$$

Similarly to transform a decrypted number $c$ into a character with the ASCII value of $x$, we use $s$ (the private exponent) and $n$ (again, the public modulus).

$$\text{decrypt}(c) = c^s \mod n = x \quad (2)$$

For very large values of $r$ and $s$, the exponentiation requires the use of a numerical method (such as the Square-and-Multiply algorithm, binary or modular exponentiation). The student reports discuss which numerical method was best, in terms of Big-O notation, execution time and number of operations, and also how to solve communication problems (e.g., detecting when the value of an encrypted character was greater than the range of character values, how to send integers as strings, etc.). Students also address the RSA Factoring Challenge [22] - in particular what the meaning of recent factorizations of challenge numbers such as (RSA-640 in November 2005) implies about the security of encryption keys.

5) **Image Processing:** Given the complexity of introducing image processing in this lab, limitations were placed on the scope of this experiment. We assumed the lab would use a single frame of video. The frame would be represented by a file in RGB24 format (24-bits for red, green, and blue – each one byte) with a pixel represented by a line, each RGB value comma-separated on the line.

The first part of the lab is to read the image, convert it to RGB565 using a given C macro, and display it in VDSP++’s Image Viewer. The file took some time to load because of the USB interface - but using a high-performance PCI (HPPCI) emulator the loading of the file was very fast. The second part of the lab is to develop code for filtering out pixels below a fixed threshold. The third part of the lab is to perform a fire detection given a video frame. The students applied the same simple filter, but this time the threshold corresponded to the colorspace occupied by fire. Figure 6 shows the before and after results for filtering based on the colorspace corresponding to fire.

6) **Student-Design:** This final lab was intended to be act in part as a segue to capstone design. Students were allowed to design their own final lab but were not required to fully implement the design. Each group wrote a proposal that included:

- the goals and motivations of the work
- related work
- problem(s) addressed
- solution(s) proposed
- justification of the approach (listing advantages and disadvantages)
- a time schedule for implementation (with Gantt charts)
- division of work between group members
E. Student Feedback

The Spring 2006 lab consisted of only 15 students, but students were asked to fill out an anonymous web survey to rate the lab. On average, most students found the lab to both difficult but very useful. Students answered that they had learned as much as expected, improved their time-management skills, found the group-based evaluations somewhat useful, found the Wikipedia articles very useful, and overall felt that the balance of breadth and depth was good.

V. EXTENSIONS

The new Microprocessor-based Design Lab is only in its first year and will continue to be refined. We plan to improve both the hardware and software aspects of the platform. We are also providing materials online for other labs to use and to encourage interested persons to help develop the work.

Some features of the VDSP++ IDDE were mentioned but not dealt with thoroughly. Advanced labs could incorporate profiling using the VDSP++ Profile Guided Optimization (PGO) Tool. Once students discovered hot code in their programs, assembly code could be hand tuned (a typical exercise of commercial implementations). Also, because the Blackfin can use a variety of development platforms, the lab would be well served to incorporate GCC/μClinux as an example of an alternative tool chain.

In addition to improving software-related topics, there remains room for improving hardware-related topics. Taking advantage of both BF561 cores for performance and power should be explored (for instance, the first lab might implement a doubles game, with the rows and buttons corresponding to a one-ball, four player game). In the future, we hope to make use of the extender boards available for the BF561 such as the USB-LAN board for disk I/O or network communication; and the FPGA board for acceleration. Indeed, we would like the students to utilize the FPGA more heavily in different steps in the labs. Also because the instruction and data caches are configurable, an advanced discussion of caching should be included.

Finally, as an outgrowth of the lab, we have developed a website for the Northeastern University Blackfin Labs NEUfin Project. The goal of the project is provide for an open discussion and development of relevant labs and tutorials for others. The material produced by the project is intended to be shared with other schools. The website is located at:
http://www.ece.neu.edu/groups/nucar/BFLab

VI. SUMMARY

Northeastern University is adopting a common embedded platform to be used in a number of undergraduate courses.

The Blackfin architecture has been selected upon which to base this platform because of its DSP and MCU functionality, its support of operating systems and programming languages, its documentation, its use in industry, and also for ADI’s commitment to support universities. Already this platform has been used in the Digital Signal Processing Lab and Microprocessor-based Design Lab. Both labs were received very well by the students and provided a wide range of educational material.

Future courses which will use this platform include the digital logic design lab and the hardware description language course (using the FPGA extender board). The goal will be to see the hardware/software co-design tradeoff of utilizing an FPGA versus high-level language. There are also plans to incorporate elements of these tools into the undergraduate Computer Architecture course.
ACKNOWLEDGMENT

The authors would like to thank Mimi Pichey, Giuseppe Olivadoti, Richard Gentile, and Ken Butler from Analog Devices for their generous donation of Blackfin EZ-KIT Lite evaluation boards, software, and extender boards, and for their support of this project. The authors would also like to acknowledge the efforts of Kaushal Sanghai who provided technical support on the BF561.

REFERENCES

[6] http://ee.unipr.it/SHARC2116x/, Hands-on ADI DSP 1-day course, University of Parma Italy.
PSATSim: An Interactive Graphical Superscalar Architecture Simulator for Power and Performance Analysis

Clint W. Smullen, IV
Clemson University
Dept. of Electrical and Computer Engineering
Clemson, SC 29634
csmulle@clemson.edu

Tarek M. Taha
Clemson University
Dept. of Electrical and Computer Engineering
Clemson, SC 29634
tarek@clemson.edu

Abstract

Two of the most important design issues for modern processors are power and performance. It is important for students in computer organization classes to understand the tradeoff between these two issues. This paper presents PSATSim, a graphical simulator that allows student to configure the design of a speculative out-of-order execution superscalar processor and see the effect of the design on both power and performance. The simulator explicitly shows the relationship between instructions within a processor by visually tagging instructions. The use of a graphical simulator makes it simple for instructors to demonstrate the execution of instructions within these architectures and the interactions among processor components.

1. Introduction

Given the emphasis on speculative out-of-order execution superscalar processors and power consumption in industry, it is important to introduce and cover these topics in computer organization classes. Ideas that students need to understand include the flow of instructions within the pipeline, dynamic scheduling, speculative execution, and the data dependencies between instructions. Students also need to appreciate the power consumption which offsets performance gains within processors. It is difficult to explain these concepts and the interactions among the architecture components without visual aids. Graphical simulations of these architectures allow students to easily grasp the concepts of the architectures by observing the flow of instructions in time. They allow students to explore the impact of different architecture options on the performance and power consumption of a processor.

The majority of existing graphical simulators target simple processor architectures. For instance, WinDLX [4] models the scalar pipelined architecture described in [5]. The IJVM simulator [8] models the stack based JVM described in [6]. Existing superscalar processor simulators with graphical front-ends such as SuperSim [7] and the Simplescalar Visualizer [10] model processors in great detail, but do not show the dependencies between instructions within the datapath that contribute to different instruction flows. At present the only classroom tool for modeling the power consumption of computer architectures is Quilt [2]. However, it does not graphically show the flow of instructions in a processor.

This paper presents PSATSim, a graphical simulator that explicitly shows the relationship between instructions within a superscalar out-of-order execution processor. It indicates both the power consumption and performance of a given architecture configuration. The simulator is based on ideas in the SATSim simulator described by Wolff and Wills [12]. Our simulator provides a wider variety of configuration options, explicitly shows speculative execution, provides a clear layout of the processor components, and provides power consumption values at the end of execution. PSATSim has been designed for ease of use and installation. Color tagging of instruction registers makes it easy to follow the flow of instructions. Incorrectly speculated instructions are indicated on screen, making it possible to see the effect of misspeculation. The simulator is trace driven, with the traces generated using SPEC benchmarks on the SimpleScalar simulator [10]. PSATSim is currently used in undergraduate and graduate courses in computer architecture at Clemson University.

2. Simulator Features

PSATSim models the dynamic power consumption of superscalar processor architectures. It incorporates a speculative execution model which gives relative accuracy to the power values. PSATSim allows students to experiment
with a wide range of architectural parameters and to observe both the power and performance results of the configuration. An example of the results screen from an execution of PSATSim is shown in Figure 1. The following sections will describe the architecture used by PSATSim, the model of speculative execution, and the power model.

2.1. Modeling instruction execution

PSATSim models a superscalar processor with out-of-order speculative execution. Out of order execution allows instructions not waiting on data values to execute ahead of other instructions. Speculative execution allows the processor to speculate on branch outcomes and continue executing instructions, without disrupting the final state of the processor. Figure 2 shows the graphical view of the processor presented by PSATSim. Instructions flow downward in this view. The architecture closely models the superscalar architecture described by Shen and Lipasti [9]. The architecture can be divided into three parts: the in-order front end, the out-of-order core, and the in-order back end (Table 1 shows the components within each part). Users are able to modify the configuration of the processor and view the effects on instruction flow through the simulator. At the end of execution, a results screen is displayed including the power consumption and the execution time. Configuration options are described in detail in section 3.

The three stages in the front end (fetch, decode, and dispatch) have an in-order flow. As instructions enter the processor through the fetch stage, they are serially numbered and color coded to easily identify their progress through the architecture. After moving through the decode and dispatch stages, instructions enter the out-of-order execution core. The branch predictor is not shown in the graphical view as it is modeled statistically. Since the simulator is trace driven, the outcome of all branch instructions are known a priori. This allows users to try different predictor characteristics and observe its effect on the architecture. The same applies to the memory hierarchy.

In the execution core, instructions are simultaneously entered into the reorder buffer and an available reservation station. They are also renamed through the renaming table and register remapping table (these tables can be minimized to reduce on-screen clutter if needed). The reservation stations indicate the instruction opcode and the source and destination registers. These registers are indicated by the serial number of the instruction producing them. If their value is not available, the registers are also color coded to their producer instruction’s color. Once the producer instruction completes execution, the register color is removed to indicate that its value is available. This makes it easy to identify the data dependencies between instructions in the different reservation stations and functional units. Once all of the input values are available and the front pipeline stage on an appropriate associated execution unit is free, the instruction begins execution. The instruction continues to occupy its reservation station until the instruction’s execution has completed. An execution unit may be pipelined or have a multi-cycle delay. Each stage in an execution unit pipeline is shown, while multi-cycle delays are only indicated by the multi-cycle residency in the execution unit.

After an instruction finishes execution, it is removed from the reservation station. In addition, the corresponding entries in the renaming table and reorder buffer are uncolored. This makes it easy to determine which instructions are ready to commit from the reorder buffer. The reorder buffer is placed on the left side of the display to provide an easy reference to all the instructions as they flow through the datapath.

Memory access occurs in the last pipeline stage of the memory unit. Therefore, if there is a data cache miss, every memory instruction in the execution unit will be delayed. The memory hierarchy is not visualized in the interface,
Figure 2. Normal Execution

Figure 3. Speculative Execution
though cache misses in each level are listed in the running statistics at the top of the simulator. The statistics at the top of the window also list the current cycle number, the number of instructions committed, the IPC, the number of cycles remaining on a fetch (shown as 'Fetch Latency'), the number of branches encountered, as well as the number of misspeculated branches.

2.2. Modeling speculative execution

PSATSim provides the option of simulating the effect of speculative execution, resulting in significantly more accurate power modeling. Since the simulator is trace driven, the instructions that would be fetched on a branch misprediction are not available in the trace. However, Bhargava et al. [1] show that it is possible to model the effect of misprediction by fetching instructions from the main trace file. This is due to the fact that a large percentage of the instructions fetched during misspeculation are fetched normally at other points during execution. In other words, instructions that would have been executed during a misspeculation are similar to those in the input trace which is already being utilized. PSATSim simulates the effect of misspeculation by loading instructions from the same trace file independent of the normal stream of execution. This provides the same general mix of instructions which would have occurred if the actual program had been executed and accounts for the energy consumed by instructions which, in the end, have no effect on the architectural state of the processor.

If the simulation of misspeculation is enabled, the misspeculated instructions are shown with a strikethrough on the display. Figure 3 shows an example of the simulator in this state. As with any architecture allowing out-of-order speculative execution, it is necessary to commit the instructions in order; in PSATSim, this is accomplished using a reorder buffer. Each cycle, up to the superscalar width of finished instructions in the reorder buffer will be committed, thereby permanently affecting the processor’s state. The instructions that will be committed in the current cycle are shown in the bottom row of the display in the 'Commit' row. Though misspeculation visualization may be disabled, the reorder buffer is always used to ensure that instructions are committed in-order.

2.3. Modeling power consumption

PSATSim incorporates the Cacti-based [11] Wattch high-level power modeling system [3]. Though the Wattch power model has been designed for the SimpleScalar architecture, we have adapted it to support the architectural flexibility of PSATSim. Because of the large number of differences between the two simulators’ architectures, power values cannot be directly compared between Wattch and PSATSim.

The Cacti system was designed for computing delay-optimal cache structures. The delay comes in the form of the effective resistance and capacitance of the cache. In computing dynamic power, all that is needed is the effective capacitance of a structure. The energy is consumed during switching of the transistor states, that is, in charging and discharging the transistors. The resistance is not needed, since it principally affects the maximum clock speed at which a circuit will operate. The Wattch power model uses Cacti, and its implementation of known semiconductor manufacturing process parameters, to compute the effective capacitances for a wider range of processor structures [3].

The dynamic power is defined by Wattch as: $P_d = CV_{dd}^2af$. The exclusion of a model for static power does not significantly impact the accuracy of PSATSim due to the use of 350 nm technology process parameters. The capacitance, C, is generated using Cacti based on the process parameters. The supply voltage, $V_{dd}$, as well as the clock frequency, $f$, are generally determined by the technology process used, though many architectural design decisions can significantly reduce the clock frequency. The activity factor, $a$, represents the average proportion of the structure that is switching each cycle. Modeling the activity factor well is the most difficult part of normalizing the power model to published power figures.

In modeling the effective capacitance for different components, there are three general categories which an architectural structure can fall into: array structures, content-associate memories, and complex logic blocks [3]. The last category is used to model functional units and will be discussed later. The first two categories are interrelated and are used to model the caches, the reorder buffer, the register renaming table, and the architecture register file. Array structures are modeled as sets of decoder, wordline drive, and bitline discharge circuitry, which consume most of the energy. Reservation stations are modeled as content-associate memories, which are similar to array structures, except that they use taglines and matchlines due to their fully-associative nature. The original implementation of Wattch inside the sim-outorder component of SimpleScalar [10] features a register update unit, which combines the features of the reorder buffer, a centralized reservation station, and the dispatch stage into a single unit. PSATSim models the structures separately, since that is much more common in industry. The PSATSim power model for each of these structures is correspondingly independent, resulting in a much wider range of possible configurations than with Wattch implemented in sim-outorder.

Caches are modeled using a hybrid between the two that is built out of array structure equations. This is due to the fact that caches are rarely fully-associative, due to their size and the need for performance. Since the memory hierarchy
is stochastically modeled in PSATSim, the cache size parameters which would actually provide the supplied hit rates is unknown. Therefore, it is necessary to assume certain cache sizings for the purposes of power modeling. Since PSATSim uses the baseline 350 nm technology parameters contained within the Watch model, the two L1 caches are modeled as direct-mapped 16 kB each, while the L2 cache is modeled as 256 kB four-way set associative. These cache parameters are typical of processors made using 350 nm processes, as specified by Watch [3].

For structures such as caches, which pre-charge and discharge the bitlines every cycle, regardless of being used, the activity factor is assumed to be 100%. For wide structures, such as the fetch, decode, dispatch, and commit stages, the activity factor can be scaled by the utilization. That is, if only one-half of the decode stage is being used, then the activity factor can be approximately halved. Watch introduces a number of conditional clocking schemes designed to mimic modern processors. The most important of which models the activity factor as a linear function of the number of units being utilized. Even with no utilization, there is still some power consumed, due to clock circuitry. At this time, PSATSim scales the activity factor for all regular structures linearly from 10% to 100% (this is the same assumption used in Watch). The 10% minimum is used by Watch to account for transistors that cannot be disabled by aggressive conditional clocking [3]. If a structure is too complicated to be modeled in this way, then an activity factor of 50% is assumed. This gives an equal likelihood for each gate to be switched.

The selection logic, found with each reservation station, and the dependency check logic, found in the dispatch stage, both feature complex logic which have models which were used within Watch [3]. Due to the complex nature of functional units, there is no way to construct a high-level model to support a wide range of functional unit types. It is therefore necessary to extract capacitance values for each functional unit type from low-level models for a specific technology process. In PSATSim, these values are stored, along with technology process parameters, within a file that defines the ISA and the capacitance of the circuitry to execute each instruction. At the moment, PSATSim uses extremely aggressive clocking which causes different opcodes that execute on the same functional unit to have different energy consumption.

3. Simulator Configuration

When the simulator is started up, the user is presented with a processor configuration dialog box to tailor the architecture. The configuration is divided into three main parts (through a tabbed interface): general, execution, and memory/branch. Figures 4, 5, and 7 show example screenshots of the three tabs. The settings specified can be saved and loaded later. The exact configuration used by the simulator is saved into the output file, including the random-number generator’s seed, making it possible to exactly recreate the output. The configurations are stored in XML format, making it simpler to edit and view manually if necessary. The following sections discuss the three configuration tabs and how they relate to the architecture modeled.

3.1. The General Tab

On the general tab (see Figure 4), the superscalar width specifies the bandwidth of the front and back ends of the processor (fetch, decode, dispatch, and commit stages). The number of reorder buffer entries, renaming table entries, the input trace file, and the output results file are also specified here.

3.2. The Execution Tab

On the execution tab (see Figure 5), the user can configure the reservation station and the execution unit configuration. Three reservation station structure types (as defined in Shen and Lipasti [9]) are offered: distributed, centralized, and hybrid. In the first case each functional unit has its own reservation station, while in the centralized case all functional units share a common reservation station buffer (see Figure 6). In the hybrid case, a group of functional units shares a buffer, as in the MIPS R10000 [13].
The execution tab also allows the user to configure the number of functional units. Depending on the details taught in class, students can pick different types of functional unit configurations: simple, standard, complex, and custom (see Table 2). In the simple case, all integer and floating point operations are carried out through a general ALU. In the standard case, separate integer and floating point ALUs are provided, while in the complex case, separate functional units are provided for each sub type of operation (such as floating point divide and floating point multiply).

In the custom case, the user can specify any arbitrary structure of functional units and reservation stations. It allows functional unit latencies and throughputs to be changed, along with the type of instructions executed by the units. In addition the functional units connected to each reservation station can be modified. The custom configuration is specified by hand editing an XML file.

### 3.3. The Memory/Branch Tab

The branch prediction accuracy and memory properties are defined in the memory/branch tab (see Figure 7). The user can choose to have just system memory, system memory with L1 caches, or system memory with both L1 and L2 caches. The access latency and hit-rate for each cache level is configurable, as is the branch speculation accuracy. A checkbox is provided to disable the visualization of branch misspeculation modeling. Since these components are modeled statistically, students can observe the effect of different branch hit-rates or memory latencies on processor performance.

#### Table 2. Functional Unit Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Functional Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>ALU, branch, and memory.</td>
</tr>
<tr>
<td>Standard</td>
<td>Integer, floating point, branching, and memory.</td>
</tr>
<tr>
<td>Complex</td>
<td>Integer addition / multiplication / division, floating point addition / multiplication / division / square-root, branching, and load/store.</td>
</tr>
</tbody>
</table>

![New Trace Options](image)

**Figure 5. Standard Execution Architecture**

Settings for a New Trace

- RSB Integer RSB Integer RSB
- RSB Branch RSB Memory

- **a)** Standard EU with distributed RSB architecture

- RSB ALU RSB ALU RSB
- RSB Branch RSB Branch RSB

- **b)** Simple EU with hybrid RSB architecture

- RSB
  - LA, LM, IDiv, FMAdd, FMult, FPMult, FPDiv, FPStd, Branch, Load, Store

- **c)** Complex EU with centralized RSB architecture

**Figure 6. Example EU and RS Configurations**
Figure 7. L1 and System Memory Architecture Settings for a New Trace

4. Implementation

PSATSim was implemented in C++. GTK+ v2 was used to provide the GUI functionality. The use of cross-platform libraries enables PSATSim to run under both Windows and Linux environments, as well as some other Unix-type systems, including Solaris and MacOS X. Full inheritance and polymorphism is used, allowing for ease of extension in the future. Each component in the system, be it an entry in the reorder buffer or an execution unit, is an independent object that is linked to other objects. This design allows for custom architectures and simplifies future expansion of the program.

The Windows installer for PSATSim uses the Nullsoft Scriptable Install System and was designed to conform to general norms of modern installers. Options are given to install additional trace files. An installer for GTK+ is included in the installer to help get PSATSim running on machines which do not have GTK+ installed already, which is a high probability for most Windows based machines. The simulator and it’s associated trace files can be downloaded from: http://www.ces.clemson.edu/~tarek/psatsim/

5. Use of PSATSim

PSATSim allows the user to explore different configurations of basic superscalar processors with ease. In a course on computer architecture, it can be used in class to visually demonstrate the flow of instructions in a modern processor. A large variety of superscalar architecture configurations can be demonstrated by modifying the configuration file. The color coding of instructions allows students to visualize the interaction between processor components. PSATSim provides the same structure as other simulators, but with a more flexible configuration system, allowing an instructor to both raise and lower the level of complexity to meet the needs of their students. Configurations can be developed and then saved for use later in class or to be given as an experiment to students.

It is important for students to understand how to investigate architectures that are optimized in terms of cost, performance, and power. Projects designed around PSATSim have been used in both undergraduate and graduate level courses at Clemson University to teach students these concepts. In these assignments, students are asked to explore architecture configurations that optimize for power, performance, or both within a given chip area budget (at present an equation for chip area based on architecture configurations is utilized). The assignments can be tailored by having students develop architectures that are targeted to a specific class of applications (such as integer, scientific, multimedia, or a combination of these).

6. Future Work

In its present form, PSATSim does not allow the user to directly alter the power model settings. The inclusion of an interface with which the user could select from a set of technology processes would increase the simulator’s flexibility and usefulness. This would also allow for the incorporation of more modern technology processes than the base 350 nm process. Also, the incorporation of leakage characteristics would improve PSATSim’s accuracy for more modern processes.

At the moment, the user input cache hit-rates do not impact the power consumption due to the use of a fixed cache organization power model that does not reflect the chosen hit-rates. Similarly, the efficacy of the branch predictor has no impact on the power consumption. Functional modeling of the branch predictor components would provide more accurate power and performance modeling. The incorporation of a model to reconstruct the run-time object code, such as that presented by Bhargava, et. al, would improve the accuracy of energy consumption statistics for misspeculated execution paths [1]. The inclusion of memory addresses into the trace files would facilitate the incorporation of functional cache modeling as well. This would make it possible to directly associate the power consumed by the caches and branch predictors and would significantly improve the accuracy of the power model across the range of input para-
The implemented power model has not been normalized against published figures, so it is not possible to compare the power values from PSATSim against those of other simulators. Normalization would improve the accuracy of the power model and extend the usefulness of PSATSim as part of a broader power modeling project. Additionally, the ability to easily input the energy values for each instruction type by the user would make it possible for students to incorporate values generated from SPICE models and the like into PSATSim and quickly see the results of low-level changes at a high-level.

7. Conclusion

PSATSim provides a unique tool for use in computer architecture courses. It allows students to try out different architectural parameters and interactively watch execution or quickly jump to the end of the execution and view the resultant power and performance figures. It can be used both to demonstrate the operation of speculative superscalar architectures and to teach students about the trade off between performance and dynamic power consumption.

References


An Execution-Driven Simulation Tool for Teaching Cache Memories in Introductory Computer Organization Courses

Salvador Petit, Noel Tomás
Computer Engineering Department
Faculty of Computer Science
Polytechnic University of Valencia
spetit@disca.upv.es, noetoar@eui.upv.es

Julio Sahuquillo, and Ana Pont
Computer Engineering Department
Technical School of Applied Computer Science
Polytechnic University of Valencia
{jsahuqui, apont}@disca.upv.es

Abstract

Cache memories are the most ubiquitous mechanisms devoted to hide memory latencies in current microprocessors. Due to this importance, they are a core topic in computer architecture curricula, both in graduate and undergraduate courses. As a consequence, traditional literature and current educational proposals devote important efforts to this topic. In this context, exercises dealing with simple algorithms, also known as code-based exercises, have a good acceptance among instructors because they permit students to realize how the accesses generated by the programs affect the cache’s state.

From about one decade ago, simulators have been extensively employed as a valuable pedagogical tool as they enable students to visualize how computer units work and interact each other. Unfortunately, there is no simple simulator allowing to perform code-based exercises for cache memories. Hence, students perform these exercises by means of the classic “paper and pencil” methodology.

In this paper we introduce Spim-cache, a simple execution-driven cache simulator to carry out such experiments, intended to use in undergraduate courses. The tool allows, in an intuitive and easy way, to select a given cache organization and run step-by-step the code proposed while visualizing dynamic changes in the cache’s state.

1 Introduction

The ever increasing gap between the memory and the microprocessor speeds has encouraged microprocessor architects across several decades to provide mechanisms in order to hide the long memory latencies. Cache memories have become the basic and ineludible mechanism implemented in current processors to hide these latencies and reduce the average data access time. Furthermore, the importance of caches grows as the memory-processor gap widens, which is the current trend; for instance, in 1980 some microprocessors were designed without caches while current microprocessors include two or even three levels of caches [1] on chip. These memory structures base their effectiveness in the exploitation of the principle of locality (i.e., temporal and spatial) that data exhibit, and have been employed by computer architects from about four decades ago [2].

Consequently, a large amount of research works in the computer architecture field have focused on cache memories and related issues. This research has provided efficient mechanisms to manage and exploit caches, and some of them have been implemented in modern microprocessors. For instance, the AMD Athlon [3] includes a victim cache [4], the Itanium 2 [1] incorporates a prevalidated tag structure to provide fast accesses, the HP 7200 implements the Assist Cache [5], a particular type of split data caches [6].

Concerning to international curricula recommendations, the joint IEEE Computer Society and ACM Computer Engineering Task Force has identified cache memories as a core topic in the Computer Organization and Architecture [7] area of knowledge. This fact is because the processor performance strongly depends on the cache performance. As a consequence, cache memories play an important role in Computer Organization/Architecture Courses mainly offered at Computer Engineering schools, although concepts concerning cache memories are also studied in a more simplified way in other Technical Schools, like Electrical Engineering or Computer Science. Computer Organization courses cover the study of the basic functional units of the computer and the way they are interconnected to form a complete computer. Computer Architecture courses mainly cover advanced processor architectures, advanced memory mechanisms, and multiprocessor systems; as well as the corresponding performance evaluation issues. Concerning to the study of caches, they involve a wide variety of con-
cepts and functionalities ranging from the basic functionality to advanced mechanisms implemented in modern microprocessors. The basis of caches, covered in initial Computer Organization courses, deals with concepts like cache organization, mapping functions, and replacement algorithms. Advanced mechanisms, like way prediction or the trace cache, are covered in a posterior Computer Architecture course.

To teach how caches work some widely referenced books, e.g., [8], [9], [10], and [11], use simple algorithms, for instance, the sum of the elements of an array (see Figure 1). For these algorithms, students must obtain some metrics concerning cache behavior, e.g., the hit ratio. These examples have a great pedagogical value to reinforce the learning process and have been commonly adopted by instructors around the world. Nevertheless, despite the undeniable pedagogical value, instructors train students performing these exercises using the classic paper and pencil method.

```plaintext
sum = 0;
for (i = 0; i < 100; i++)
{
    sum = sum + A[i];
}
```

Figure 1. Simple algorithm example

The massive use of computers in classrooms and laboratories has resulted in new teaching methodologies using computers as a pedagogical tool. In this context, the use of simulators is highly recommended as they enable students to visualize how the modelled system operates. A wide set of these simulators has been developed to teach the basic functional units of the computer in Computer Organization/Architecture courses, e.g., simple processors, computer arithmetic units, or cache memories. Concerning to cache memories there are simple simulators dealing with the basis of caches. Unfortunately, and to the knowledge of the authors, none of these basic simulators can be used to perform code-based exercises, i.e., the same kind of exercises that have traditionally proposed in the teaching literature. In this paper, Spim-cache is proposed, as a pedagogical tool intended to be used in undergraduate courses, attempting to fill the existing gap between current simulators and the reference guides.

Spim-cache has been developed by students and staff of the Computer Engineering School at the Polytechnic University of Valencia. In this University, computer organization subjects are split in two annual courses [12], namely Computer Fundamentals (first year) and Computer Organization (second year). In these courses, as in a high number of universities around the world, the MIPS approach is used to illustrate the corresponding topics, e.g., the processor pipeline, the memory hierarchy, and the input-output system, are studied using the same machine model. This means that students are trained with the MIPS assembly language and the corresponding Spim [13] simulator. This teaching context led us to extend the Spim simulator to implement our proposal.

Spim-cache provides support to perform code-based exercises detailing cache events in a friendly and easy way. Spim-cache allows students to run programs while visualizing, step-by-step, how the cache controller works, e.g., fetching memory blocks, or dealing with write policies. The simulator code is open source and can be found at http://www.disca.upv.es/spett/spim.htm.

The remainder of this paper is organized as follows. Section 2 summarizes the main features of current simulators and the reasons that encouraged us to deal with this work. Section 3 describes the proposed pedagogical training tool. Finally, Section 4 presents some concluding remarks.

## 2 Background and motivation

An interesting laboratory using real microprocessor’s caches is described in [14]. In this laboratory, students run a small benchmark on a given computer. The benchmark is mainly composed by a nested loop that reads and writes an array of data using different strides and array sizes. The benchmark measures the average data access time, which varies according to several factors, e.g., if this array is larger or smaller than the cache size. Analyzing the times provided by the benchmark execution, students must deduce the cache geometry (i.e., the cache size, the line size, and the associativity degree). Further details can be found in [10]. This kind of laboratory, really encourages students as they work directly on the real hardware. Unfortunately, in order to ease the analysis of the results, the benchmark must run in a relatively old processor like the Pentium II which includes a simple cache organization. In more recent microprocessors including advanced cache mechanisms, for instance, some kind of prefetching like the Intel Pentium 4 [15], or victim caches like the AMD Athlon, the cache geometry could not be deduced from the results provided by the benchmark.

The mentioned drawback jointly with the pedagogical value that simulators provide, led instructors to train students by using cache simulators. Table 1 shows a subset of current simulators dealing with caches that are being currently used for educational or research purposes, or both. An interesting survey of simulators can be found in [16]. Simulators can be classified in two main groups attending to the way they are fed: trace-driven simulators and execution-driven simulators. The first ones are fed by simple traces.
### Table 1. Educational simulators: an overview

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Complexity</th>
<th>Driven</th>
<th>Graphic Interface</th>
<th>Core Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCMSim</td>
<td>Low</td>
<td>Trace Driven</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Dinero</td>
<td>Medium</td>
<td>Trace driven</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>mlcache</td>
<td>Medium</td>
<td>Trace or execution driven</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Simplescalar</td>
<td>High</td>
<td>Execution driven</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

while the second ones, much more complex, are fed by the binary of a given benchmark.

To teach students in undergraduate courses instructors use trace driven simulators, e.g., [17]. A trace is composed by a set of lines, where each line represents an specific event, i.e., a write or read operation in a given memory address. The simulator shows how the cache contents change each time a new event occurs. In this way, students can follow how memory accesses miss or hit into the cache. Some trace driven simulators have been also used for research purposes like [18] and [19]. Nevertheless, as this kind of simulators are fed by traces they are not appropriate to perform code-based exercises.

Advanced execution-driven simulators are mainly used for research purposes. These sophisticated tools may concentrate only on cache related issues, e.g., the mlcache simulator [18], or in the complexity of the entire microprocessor, e.g., the simplescalar toolset [20], which models an aggressive out-of-order processor. This kind of tools can provide detailed information cycle-by-cycle, showing how the machine units interact each other. Due to the huge complexity involving current microprocessors, this kind of tools are organized in a structured way, including a specific cache memory module. The main advantage of advanced simulators is that they permit to know the details on each instruction in the pipeline, however, they usually fail in that they do not provide a friendly graphic interface and are quite difficult to use. This fact jointly with the complexity of the modeled processor make advanced execution-driven simulators unsuitable to perform code-based exercises in undergraduate courses.

In this paper we propose a simulator intended to use in undergraduate courses which provide support to perform code-based exercises. The tool attempts to get the best of both kinds of simulators, i.e., a friendly and easy-to-use simulator which visualizes, cycle-by-cycle, the architectural state of the machine. From the pedagogical point of view, the proposal provides a framework which permits to perform the typical exercises based on simple algorithms proposed in the literature.

### 3 Spim-cache

Spim is a simulator that runs MIPS32 assembly language programs. It has a wide acceptance among instructors of introductory Computer Organization courses and it has been widely referenced in [8]. UNIX, Linux, MS-DOS, and Windows versions are available. The user’s interface for Microsoft Windows platforms, also known as PCSpim, consists of a window that is composed by four frames, from up to bottom: the contents of the register file, the program being simulated (assembler and machine codes), the memory, and a list of log messages. This simulator permits to illustrate the instruction set architecture, i.e., the processor as seen by the programmer.

The proposed tool extends the Windows version providing support to simulate cache memories. The tool also visualizes step-by-step how the cache contents change. In addition, important statistics, e.g., number of hits or misses, are also provided on-the-fly. The tool supports the simulation of both data and instruction caches.

In order to start with the cache simulation, user should firstly select the Cache Simulation option in the Cache Settings dialog, which pops up after clicking on the settings entry of the simulator menu (see Figure 2). After doing so, a dialog with the different cache organization configuration options is displayed (see Figure 3). Users can select among
Figure 3. Cache configuration dialog

simulate a data cache, an instruction cache, or both, as explained below.

3.1 Cache Architecture Configuration

Current processors implement independent cache organizations to store instructions and data. In this context, Spim-cache permits to simulate only a data cache, only an instruction cache, or both together (i.e., Harvard Architecture) as implemented in modern processors.

The default configuration (i.e., the Harvard Architecture) can be changed through the proper dialog (see Figure 3), by clicking on the Cache Architecture Configuration entry of the Cache Simulation menu option. The data and instruction caches are displayed as two independent frames inside the main PCsSpim window. Both frames are only displayed if the Harvard Configuration is selected; otherwise, only one frame is shown.

3.2 Cache settings

To display the Cache Settings dialog, users should click on the Cache Settings entry of the Cache Simulation menu option. This dialog allows users to change the configuration of the caches. The same dialog is used to change both data and instruction cache configuration. To change between them, select the proper option from the Cache combo box in the dialog as shown in Figure 4.

The dialog shows the different options available. This interface allows users to choose the cache size (ranging from 128B to 1024B), the block size (4, 8 or 16B), the mapping function (direct mapping, 2 or 4 ways set associative or fully associative), the writing hit and miss policies (write-through or write-back, and write-allocate or write-no-allocate), and the replacement algorithm (LRU, FIFO). As instruction caches are not allowed to be modified by user programs, the writing hit and miss policies are only available for data caches. In addition, in order to provide pedagogical feedback, if the show rate option is selected, Spim-cache displays some statistics, like the number of misses, in a small frame below the cache frame.

The cache frame visualizes the cache contents and its layout, depending on the selected configuration. The direct-mapped is the default configuration and its layout is organized as a matrix. Each row in the matrix represents a cache line and the columns represent all the information associated with it. The first column contains the line identifier (or line number) while the remaining columns contain data and control information. The control information displayed for this configuration is the valid bit and the tag bits (in hexadecimal notation). In addition, for pedagogical purposes, a field Acc showing the result of the cache access (i.e., hit or miss) has been included.

When adding associativity to the cache (i.e., increasing the number of ways) the layout of the cache frame changes. To this end, Spim-cache replicates the columns of a direct-mapped cache as many times as the number of ways selected. Proceeding in this way, each row of the window represents a single set of the cache (i.e., the lines belonging to the set) and the first column states the set identifier.

To support the replacement policy, an additional column (LRU/FIFO) is added for each way, to represent the counter value of the replacement algorithm. In addition, if the user selects the write-back policy, a new column (M) is added for each way to represent the modified or dirty bit of each block.

In order to facilitate the user interaction and offer a better visualization, the basic strategy of replicating columns to provide a higher number of ways to the cache is not followed by a fully associative cache. On such case, the basic layout is transposed. In this configuration, the first column indicates the number of way and each row shows the con-
3.3 Running a program

Spin-cache allows the user to visualize how the processor’s state, the main memory, and the cache interact while a given program runs step-by-step. In addition, statistic results about the cache behavior are also presented. To run a program step-by-step, the user should load it and click on the single step menu option of the simulator menu (or use the F10 key function).

In the PCSpim simulator, a step of execution covers the entire execution of a single instruction. The cache simulation extension modifies this semantic in order to help students to follow how the cache memory works. In this sense, memory reference instructions (loads and stores) can take several steps to execute when the extension is activated. In this case, if the access hits into the cache, it takes one step to execute as the remaining instructions. Nevertheless, if the access misses, the number of steps depends on the type of instruction and the write policies.

For the sake of clarity, load and store misses are handled in a different number of steps. On one hand, a load miss is handled in three steps: a) detect and mark the miss in the corresponding set (all lines in the set are marked), b) fetch the block from main memory to the cache, and c) load the desired data into the corresponding register. On the other hand, a store miss is handled in two or three steps depending on the write miss policy (allocate or no-allocate) selected. If the policy is no-allocate, the steps are a) detect and mark the miss, and b) store the content of the register in main memory. Otherwise, the steps are a) detect and mark the miss in the corresponding set, b) fetch the block from main memory, and c) store the corresponding data into the cache. Finally, with respect to the instruction cache, a miss is handled analogously to a load miss in the data cache. In this case the third step consists in the execution of the instruction.

Figure 5. First step of execution
Let us see a working example in order to illustrate how Spim-cache handles cache misses. Assume a 128B size, 16B line size, direct-mapped instruction cache and a 128B size, 16B line size, 2-way set associative data cache with LRU replacement algorithm, and write-through write-allocate policies. Let us consider the execution of the store instruction highlighted in Figure 5. Notice that the execution of a memory reference instruction involves two cache accesses, one to the instruction cache to read the corresponding instruction and the other one to the data cache.

In the first step, there is instruction cache hit as the store is already in cache (see Figure 5). Also, in the first step when the store is executed and accesses to the data cache this access results in a miss. Since both ways in the set contain a valid line, the LRU algorithm selects the line in way 1 to replace as it contains the LRU line (higher LRU counter value). Hence, in the second step, the missing block is fetched from main memory to the data cache (see Figure 6). Finally, the content of register $14$ is written to the data cache. In parallel, it is also written to main memory because of the write-through selected policy.

Notice that on each step of the execution, if the show rate option is selected, Spim-cache visualizes the number of accesses, the number of hits and the cache’s hit rate. For the data cache the Spim-cache also visualizes the number of the different misses. Misses are classified as compulsory misses (when the block is accessed for the first time), conflict misses (when the target set is full but the cache is not full), and capacity misses (when the cache is full). The obtained statistics would allow the student to realize how the configuration of the cache affects performance.

4 Conclusions

Educational books and instructors use small fragments of code implementing simple algorithms that include memory references to teach cache memories. Students learn caches identifying these instructions and following, using paper
and pencil, the sequence of accesses to the cache. In this context, they observe when these instructions miss or hit into the cache.

This paper has presented Spim-cache as a pedagogical tool to perform this kind of exercises in undergraduate courses. The proposed tool has been implemented as an extension of the Windows version of the MIPS R2000 Spim simulator, because this framework is well known and widely used in a high number of universities around the world. Nevertheless, the same idea could be implemented in other basic tools.

From a pedagogical point of view, the proposed tool benefits the learning process, since it permits students to observe how cache information dynamically changes as the processor runs instructions, helping students to learn how the processor-memory work as a whole.

The current version of Spim-cache has been thoroughly tested on the Windows operating system and Linux operating system using the wine library (http://www.winehq.org/). The Spim-cache source code is publicly available at http://www.disca.upv.es/spetit/spim.htm. Users can directly either run the tool by using its binary file or modify its source code to add new functionalities.

Acknowledgements

This work has been partially supported by the Generalitat Valenciana under grant GV06/326 and by Spanish CICYT under Grant TIC2003-08154-C06-01.

References

Abstract

Learning the various structures and levels of memory hierarchy by means of conventional procedures is a complex subject. A memory hierarchy environment (Web-MHE) was proposed and developed as part of our undergraduate research, to serve not only as an auxiliary teaching tool for the Computer’s Architecture professor, but also as a learning facility tool for the student carrying out his undergraduate or graduate courses. The environment encompasses the knowledge of the various levels of the memory hierarchy, and can be either used by students in his studies or by senior researchers. In conclusion, the Web-MHE offers the user the possibility to simulate, experiment and learn the various concepts of the memory hierarchy.

1. Introduction

The learning of the memory hierarchy is a complex process mainly due to the difficulties faced when attempting to understand and visualize the process that take place during the accessing of the memory positions in the various hierarchy levels. Teachers are usually face with great difficulties in grabbing the students’ attention while using slide shows or other static teaching tools, and, from the student’s perspective, the classes gradually become boring. The teachers face difficulties in presenting, in a simple format, the various hierarchy levels using the available conventional teaching tools [1]. The use of simulation didactic tools may well contribute to the learning process steps, as well as to the designing of more objective exercises, enabling the students to compare the results of different configurations and memory traces, and to visualize the various hierarchy levels. The difficulty and complexity faced by the students when solving exercises and verifying questions may be reduced to a minimum with the use of didactic simulators, making the problem solving task a more pleasant one.

The Memory Hierarchy Learning and Research Environment with support the Web (Web-MHE) [2] is composed by a group of didactic materials with which the student can learn alone or to complement his study on the main memory hierarchy concepts, a simulation tool (Web-MHSim - Web Memory Hierarchy Simulator), where the student can test and to increase his knowledge and configurations and memory traces files previously defined, that can be chosen by the user and loaded so that an automatic simulation can be generated.

The Web-MHSim is a didactic memory hierarchy simulation tool of (cache, main and virtual memories), that it has as objective to be an ally of the teacher in the teaching and of the student in the learning, by providing easy ways to configuring and simulating possibilities for different structure options. The simulator is able to handle the different levels of the student’s knowledge on the subject, of beginners, in the process of getting familiarized with the theory of memory hierarchy, to the advanced researchers, that want a simulator to verify the results of their research work, in an efficient and quick way. The simulator Web-MHSim is an evolution of the MSCSim [1][3].

For verification of the results were considered some simulators and one environment as reference. The developed environment looked for improvements and new functionalities in relation to these. To facilitate the spread and platform independence, the environment was developed for Web.

In synthesis the Web-MHE has as goal to be an ally of the teacher in the teaching, to enlarge the degree of the students' knowledge and to facilitate the research of the several structures of the memory hierarchy. The users have the possibility to learn, to simulate and to try the several concepts and techniques of the memory
hierarchy, increasing his degree knowledge, to execute practical exercises, to accomplish experimentations and other.

2. The use of Simulators in Computer Architecture

The traditional teaching method, where the teacher presents the various concepts of memory hierarchy by means of static devices such as slides shows and text books is not enough to give to the majority of the students a precise understanding of what is being taught.

The integration of the traditional teaching method with the use of didactic simulators contributes to a better understanding of the subject since they are capable of translating to reality the theoretical concepts by introducing them in a clear and didactic fashion [4]-[9].

The Simulators comprise the creation of dynamic and simplified models of the real world. Simulators, in general, despite being simpler than real systems, often present a reasonable difficulty degree in their use, due to the knowledge level required to handle them.

Any didactic Simulator must be capable of offering to the student a possibility of choice of the simulation complexity according to his knowledge level on the subject, enabling him to gradually increase the simulations complexity degree. Other important features required to a simulator are the platform independence, easy installation, configuration and use of the tool, reducing the time spent in learning its operational requirements and procedures. The tool must offer a great amount of information in a clear and objective manner, therefore contributing to the student’s learning process while the student increases his critical evaluation and knowledge degree, through the execution of various exercises and experimentation of different hypothesis, as well as checking solutions for specific problems.

Some memory hierarchy Simulators such as Dinero IV [10], CacheSim [11], VirtualMemory [12], CACTI [13], LDA-Cache [14] and Prima [15] (reference simulators), are known in the academic world but do not possess all the features previously referred.

3. Related Works

HASE Dinero [16] stands out for being an environment learning Web and research of cache memory. It was considered as reference for comparison and definition of new functionalities of the Web-MHE. HASE Dinero was the only environment Web of cache memory with didactic resources found in the study of the state of the art.

The simulators CacheSim and Dinero IV have as main feature the cache simulation and VirtualMemory stands out for the simulation of virtual memory. The most interesting features from these simulators were selected to they be incorporated in the Web-MHSim. For verification of the evolution of the tool MSCSim, was used whose the base aided in the development of the Web-MHSim.

HASE Dinero is a teaching environment of cache memory that was developed at the Institute for Computing Systems Architecture of the University of Edinburgh. The environment is composed by a group of didactic materials, animations that can be visualized in the Web, simulator desktop that possesses all of the functionalities of Dinero IV (except multiple levels and multiple input formats) and previously defined files of configurations that can be used in the simulation. In this tool is not allowed the simulation in the Web, simulation of virtual memory, generation of memory traces files and others.

CacheSim is a tool Web of cache simulation that allows the users to specify a configuration and choose one of the memory trace existent, so that, soon afterwards, they can verify the acting of the cache. With CacheSim is not possible to accompany the simulation step by step, the time of access, performance graphics, some simulation types (only the simulation of unified, instruction or data cache memory it’s possible) and it doesn't possess memory trace generator and interaction with the user during the simulation. The memory traces files are previously defined and it doesn’t allow the configuration for the user.

Dinero IV was developed at the University of Wisconsin by Mark D. Hill. The results are obtained through the reading of a memory trace, parameters and configurations specified. It is possible to configure several types of caches (direct mapped, set associative and fully associative), that can compose a cache unified, separate and/or multiple levels. Block’s size, associative levels and other parameters can also be specified. In this version it is also possible to classify the type of miss (compulsory, conflict and capacity). Parameters such as time and type of access, oriented interface for apprenticeship as well as some types of simulations such as multiple levels and virtual memory are not found in this Simulator.

VirtualMemory was developed by Ngon Tran e Dr. Daniel A. Menasce’ at the George Mason University. This tool presents a didactic interface that permits the virtual memory simulation (TLB, main memory, page table and hard disk) and exhibits statistical data during
the simulation. Other type of simulation, configuration and memory trace isn’t possible.

MSCSim was designed and developed with the objective of to be a didactic simulator for desktop. In this, it is possible to simulate and to accompany the behavior of several combinations of structures, as unified or split cache, multiple levels and memory hierarchy (cache, main and virtual memories). The simulations offer a great wealth of details, showing step by step as each address allocated. Among its main features stands out memory trace generator, statistics, didactic animation of the hierarchy structures and interface guided to the learning. In this version the users can’t enter with data during the simulation, to accomplish simulation with support the separated main memory, to visualize statistical and other graphics.

4. Web-MHE

In computer systems, in order to achieve a high performance level in the handling of information (data and program instructions) to and from the computer memory, various types of memory (memory hierarchy) are required in each single computer. For some specific tasks, it’s of the utmost importance that the transfer of the information is to be carried out as fast as possible. It’s the case of the tasks carried out in the central processor (internally) where speed is the main requirement, but the amount of bits to be manipulated is very small. This feature characterizes one type of memory known as cache (physically closer to the processor), meaning any memory that, if used in conjunction with a slower one (for instance, the main memory) may reduce the average access time. This memory can be organized as unified, split (separated cache memory) and at various levels (multiple cache levels).

The main memory is the place where the data and instructions are stored and recovered by the processor. This, can be unified (Von Neumann architecture) or separate (Harvard architecture), where exists a main memory of data and another of instructions. Therefore, accesses that generate hits at the highest hierarchical level can be processed faster. The accesses that generate miss are complied with access lower hierarchical levels, which are slower and bigger. If the hit rate is high, the hierarchy has an average access time very close of the first level component, therefore being faster [17]-[21].

The Web-MHE is inserted in the memory hierarchy learning context, and more precisely in the use of didactic environments for this purpose. The main goal is to be an Web environment with the desirable didactic features (tutorials, didactic interface, Web simulator that has several types of configurations and simulations, inclusion of knowledge levels, configured previously files and animation). Other objectives were the incorporation of the main functionalities of the references and new resources that helps the student and the researcher.

The environment was developed in HTML and JAVA because it’s a language free, robust and platform independent that became possible an object oriented development and the use of applet resources. Some software engineering concepts were used as prerequisites definition, use case and class diagram.

The Memory Hierarchy Learning and Research Environment with support the Web (Web-MHE) is composed by a group of didactic materials with which the student can learn alone or to complement his study on the main hierarchy concepts, a simulation tool (Web Memory Hierarchy Simulator), where the student can test and to increase his knowledge and file of configurations and memory traces previously defined, that can be chosen by the user and loaded so that an automatic simulation can be generated.

The Web-MHE is a didactic tool of simulation (memory hierarchy (cache, main and virtual memories), that it has an objective to be an ally of the teacher in the teaching and of the student in the learning, by providing easy ways to configuring and simulating possibilities for different structure options. The simulator is able to handle the different levels of the student’s knowledge on the subject, of beginner, in the process of getting familiarized with the theory of memory hierarchy to the advanced researcher, that want a Simulator to verify the result of their research work, in an efficient and quick way.

Figure 1. Web-MHE general view.

The Web-MHE is a Web didactic environment that includes a simulation tool, pre-configured files, didactic materials and memory hierarchy animations (Figure 1). The main requirement of the environment is to offer a didactic tool of memory hierarchy simulation.

The main aim of the didactic materials is to offer to the student resources capable to aid him in the study or in the learning alone of memory hierarchy. These materials are composed by cache lectures (direct mapped, set associative, fully associative, multiple levels and split cache), main memory (unified and split), virtual (TLB, pages table and virtual memory) and others. During the lectures are presented pre-configured files where the user can simulate and to visualize the results of the theory. Also are available
animations about these themes, with the objective of helping the student's learning.

One of the main requirements of the Web-MHSim (Figure 2) is the possibility to simulate and to accompany the behavior of some combinations of structures, as unified cache, split cache, multiple levels and virtual memory. Other requirements considered in the design of the simulator were the possibility to offer a didactic tool, the user's possibility to inform relative data to the accesses during the simulation and support to Web, among others.

The simulations are offered with a wealth of details, showing, step by step each allocated address, also allowing the user to inform the relative data to the current address (block, slot, tag, miss/hit, fault type, access time, virtual page and frame), helping the student in the analysis and performance evaluation of the configured hierarchy.

The tool allows accomplish simulations of unified cache with main memory and unified cache with main and virtual memories. This tool can simulate a separated cache with some configuration type for their caches. In the simulation of multiple levels is possible to configure $n$ levels, which can be composed by unified or separated caches. The main memory can be configured as unified or separated, for all of the cases.

Parameters are offered for the configuration of the memory cache (Figure 2), such as access type (sequential or parallel), cache memory type (direct mapped, set associative and fully associative), Write Through (WT) and Write Back (WB) write policy, FIFO e LRU replacement policy, number of slots and cache levels, access time, associative degree and block size. Access time and word size can be defined as main memory parameters. The level of virtual memory allows the configuration of the number of slots and the TLB (Translation-look aside buffer) replacement policy, memory and virtual page size.

With the objective of facilitating the understanding of the theory and the use of the environment, resources were incorporate to the simulator, such as the memory trace generator (memory access sets), advanced statistics, calculations guide (demonstration of all calculations made by the simulator) and memory hierarchy animation.

The generation of the memory trace file can be carried out in manual or random mode. In the first, as it can be seen in the figure 3, it is necessary to specify three fields: address, write (W) or read (R), data (D) or instruction (I). For the random generation the required fields are the total number of addresses and the lower and higher value for random generation of addresses. The memory trace is stored in memory for use in the simulation phase. Another possibility is the use of memory trace files configured.

During the simulation is possible to accompany the performance of the structures analyzing the information of the addresses (block number, slot, tag, page and frame, miss/hit, fault type and access time), the remaining addresses in the structure and the statistics (Figure 5, 6, 7 and 8). The main statistical informations offered are: number of distinct blocks, hit rate (percentage of caches hits in relation to the number of occurred cache misses), cache memory, main memory, TLB and page table occupation rate, number of misses.
with replacement in the cache and TLB, cache memory, TLB and table pages hit rate, as well as average cache access time. At the same time it is possible to analyze a graphic with the rates obtained during the simulation, aiding the student in the verification of the simulated hierarchy performance.

The user can interact with the simulation, at some moment, informing the values of the block, slot, tag, miss/hit, fault type, access time, fault/hit, virtual page and frame number as it is shown in the figure 5. Later the simulator verify the informed values and it exhibits for the user their mistakes and successes. The student with the interaction can verify and to increase his knowledge level.

Didactic animations were developed to facilitate the student's understanding of the hierarchy as a completely. In this, it is possible to accompany the path of the addresses to the processor (Figure 4).

5. Results

The Web-MHSim is very flexible allowing the memory trace generation, configurations of parameters and simulation types. The user accompanies and it can interact to each step of the simulation through detailed and easy-to-understand screens.

The simulation begins after the configuration of the parameters and memory trace. After initiate, due to the interface guided to the learning, the user can visualize step by step the input of each address, their information and the actual state of each hierarchy, facilitating the study of the simulated hierarchies.

It is possible to verify the situation and the simulation of the caches with a great wealth of details, what is not found in the reference simulators. The results, statistics and graphics supplied in the screens help to understand if a configuration is the most suitable for the hierarchy.

With a memory trace composed by 100 addresses, were accomplished some simulations, where, in a first moment it is presented a simulation that possesses a basic configuration that can be understood even by a beginner student, in a second moment is accomplished a small one gets better in the simulated architecture and in a third moment a simulation more complex is presented, typical for a student that has got a wide knowledge of the theory or a researcher that wants a fast and correct answer of an architecture.

For all simulations were considered the block size of the cache to 32 bytes, size of the word to 4 bytes, write back policy, FIFO replacement, sequential access, unified main memory and main access time to 500 ns.

5.1 Basic Simulation

With the use of an unified cache memory, fully associative with 4 slots, access time of 15 ns, only 20 cache hits happened, due to the occurrence of various cache miss substitution (Figure 5).

5.2 Intermediary Simulation

The intermediary simulation was composed by a split cache (cache L1 – data, L2 – instruction), both fully associative with 4 slots, access time of 15 ns.
this happened 46 cache hits, due to elimination of the conflicts between data and instructions for the same slots (Figure 6).

**Figure 6. View of a split cache simulation.**

The hit rate obtained by the data cache was 44% and by the instruction cache was 47%, most of the cache miss was of the capacity type (data cache 74% and instruction cache 68% approximately) and the average access time was 287 ns, obtaining an improvement of 31% in relation to the average time of the basic simulation.

### 5.3 Advanced Simulation

The advanced simulation (Figure 7) was configured with 3 cache levels. The first level is composed by a split cache (cache L1 – data and cache L2 - instruction), both fully associate with 4 slots and access time of 15 ns, the second level (cache L3) was composed by a unified cache, direct mapped with 16 slots and access time of 30 ns and the third level (cache L4) was composed by set associative (four way) with 8 slots and access time of 60 ns. In this simulation the closest processor level has as objective avoids the conflict between the data and instructions for the same slots and the lower levels reduce the miss penalty.

**Figure 7. View of the second level of cache simulation.**

The hit rate obtained by the data cache of the first level was 44% and by the instruction cache was 47%. The second level also obtained a hit rate of 48%. Already the last level obtained a hit rate of 54%. The data and instruction cache occupation was totally busy. The second level cache was occupation rate of 69% and the third level presented an occupation rate of only 41%. The average access time was 112ns, showing a significant gain in relation to the simple and intermediay simulations (obtaining a reduction of 83% in relation to basic simulation and of 39% in relation to intermediary simulation). That time can be explained due to the low number of accesses (just caches of the compulsory miss type, totaling only 13 accesses) that took place in the main memory, possessing the longer access time.

**Figure 8. Statistics and graphic of the basic simulation.**

With the objective to reduction the faults of the capacity type of cache L2 (Instruction), the number of slots was modified from 4 to 8 and kept all the other parameters of the hierarchy, resulting in an average access time was 105 ns and the hit rate this cache
increased for 75%. This configuration allowed an improvement of 6% in the average access time.

Another possibility of reduction of the average access time is the elimination of the conflicts misses occurred in the second level cache, for this was modified the cache type, from direct mapped to fully associative. This was possible to obtain an average access time of 103 ns, a hit rate of cache L3 (second level) increased of 48 for 76% and occupation increased 69% for 81%.

With both alterations was obtained an average access time of 99 ns. The comparison of the original designs with the three modifications can be visualized in figure 9.

![Figure 9. Comparison between the different structures.](image)

All the mentioned metrics can be viewed at any time during the simulation process at the statistics screen (Figure 8), enabling an easy comparison amongst the structures.

As another advanced simulation example is also possible to simulate one complete hierarchy, with cache memories (unified or separate and multiple levels), main (unified or separate), virtual, TLB and page table.

Changing the memory trace and/or some parameters of the simulations as replacement policy, block size, associative degree, hierarchy levels and others, it is possible to obtain an improvement or a worsen in the performance of the simulated hierarchies.

Comparing the Web-MHE and HASE Dinero environments, can be verified that the first includes all of the functionalities presented in the second. The animation found in HASE is more detailed than an available in the Web-MHE, because the animation is the main objective of the reference. The simulator Web-MHSim possesses more features than HASE Dinero's simulator, because the last accomplishes just some simulation types (only allowing two cache levels and split) and it possesses just some configuration types, besides not being a web simulator.

Comparisons were accomplished using the CacheSim simulator, for being the only Web tool of cache simulation found in the state of the art, Dinero IV, for being known thoroughly in the academic field, VirtualMemory, for supporting simulation of virtual memory and MSCSim, that it was used with the purpose of verifying the evolution of the tool (Table 1).

### Table 1. Comparison between simulators

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Web-MHSim</th>
<th>MSCSim</th>
<th>CacheSim</th>
<th>Dinero IV</th>
<th>Virtual Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>Web-MHSim</td>
<td>MSCSim</td>
<td>CacheSim</td>
<td>Dinero IV</td>
<td>Virtual Memory</td>
</tr>
<tr>
<td>Simulation of Cache</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Simulation of Split Cache</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Simulation of Multilevel</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Simulation of Virtual</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Simulation of Cache, Main and Virtual</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Harvard architecture</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Access Time</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Write Policy</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multiple input formats</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Memory Trace generator</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Statistics graphics</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Didactic interface</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Step by step simulation</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Animation</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Web simulation</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>User interaction with the simulation</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

For including some knowledge levels the simulator was implanted at the laboratory of Computer Architecture of Computer Science in the Pontifical University Catholic of Minas Gerais. As one of the results of its use, it was observed on the part of the students, a larger easiness in the understanding of the theory taught at classroom. The tool also aided them in the accomplishment of the exercises and proof of the results. The teachers observed a better learning and a better evaluation with the use of this.

### 6. Conclusions

The results obtained with the Web-MHE were great, because were selected, improved and implemented the main features of the works in reference and allies to new resources, resulted in a web environment of aid to the learning and the research with new and more functionalities, details and efficiency, thus reaching the main goals. Some of the main features of the references simulators are multiple
levels simulation, split cache simulation, basic statistics and animation.

Among the main contributions is to offer an didactic web environment that possesses all the main features of the references simulators and also new resources as, the possibility to accomplish the simulation of multiple levels with support to split cache and virtual memory, separate main memory, calculations guide, group of didactic materials and possibility of the user's interaction, facilitating the learning and the fixation of the theory, statistics and advanced graphics and memory trace generator.

The improved and added resources contribute to facilitate the learning and the research of memory hierarchy, split cache and multiple levels to enlarge the critical and analytical thought on the results.

As future expansion will be developed a module of on-line tutor that it will be capable to accompany the level of the student's learning and to inform him which topic is with more difficulty, and also to suggest some student that is well in this topic to aid him. The Web-MHE it will support new resources as, micro architecture, ISA level and multiple input formats. A framework will be developed to facilitate the implementation of new resources for other users and/or developer. Another future resource will be the animation integrated with the simulation.

7. References

Remote Operation and Control of Computer Engineering Laboratory Experiments

Helmut Bähring  Jörg Keller  Wolfram Schiffmann
FernUniversität in Hagen
Fakultät für Mathematik und Informatik
58084 Hagen, Germany
{helmut.baehring, joerg.keller, wolfram.schiffmann}@fernuni-hagen.de

Abstract

We present a lab-in-a-box, a metal case needing only power supply and network access, to provide a complete infrastructure to establish a microprocessor laboratory for embedded applications in computer engineering. The laboratory is used in distance teaching and thus all instruments and devices are controlled and observed via the net. Our focus is on flexibility to setup experiments, and on scheduling access to such a laboratory.

1. Introduction

In computer engineering the education in classes should be complemented by laboratory sessions where students can get hands-on experience. A multitude of experiments can be thought of. However, lab instruments are expensive, so that a small number of them has to be shared by many students, who have to be scheduled for lab hours, and experiments' setups can only be changed when all groups have completed the previous experiment. Normally, this can still be handled by scheduling different groups to perform one experiment within one week, before changing the setup for the next week. However, our institution is Germany’s distance teaching university, serving German speaking students all over the world. Course materials are available online for registered students (a reason why we cannot make our materials publicly available), but a lab course either requires access to the instruments, i.e. physical presence, or is restricted to simulation experiments. While those still provide some insight, they do not show everything, especially in digital signal processing where real-time requirements have to be obeyed (see e.g. the clock decoding example in Section 4). It is possible albeit difficult to get our students to the campus for a few days, but they would complain, and with reason, if they would have to spend most of their time waiting for access to the lab instruments.

In order to attack all those problems, we established a microprocessor lab course with internet access, and reported about it at the Workshop on Computer Architecture Education two years ago [2]. Yet, our solution at that time was kind of a prototype. It needed an expensive front-end server to control all the instruments of an experiment, and changing of experiment setups was cumbersome. Also, the scheduling between different student groups was only on a first-come-first-serve basis. In a recent publication [3], we envisioned a so-called lab-in-a-box, i.e. a metal case with only a plug for power supply and a plug for network connection. The case should contain all the necessary functionality to connect the instruments and devices with the network, the power supply should be controllable via the net. The realization of this vision, on which we report here, was enabled by small devices called XPorts that bridge between network and instruments, and additionally contain the applets that enable the remote control via the net. Thus, after programming one XPort for each instrument device, a generic lab infrastructure is established that can be provided in larger numbers for small cost. Change of experiments only requires exchange of instruments, and replugging. With programmable switches, even this can be avoided.

Also, we have coupled the lab with a system for collaborative work, where it is possible to reserve experiments in advance, exchange experiences between students during and after the experiments, and where it is possible to pre-configure experiments by the system, i.e. if a student logs in to perform the next experiment in the sequence, the system can switch on exactly those components that the student needs.

While our experiments are not directed towards microarchitecture, they are oriented towards computer architecture by encompassing the use of different types of processors: microprocessors, digital signal processors, and microcontrollers, so that the students learn about their differences.
The remainder of this article is organized as follows. In Section 2 we discuss related work. In Section 3, we briefly present the XPort devices that enable encapsulation of the control software and network access to instruments. In Section 4, we report on experimental setups for microcontrollers and digital signal processors. In Section 5, we detail the steering and scheduling of student groups for the laboratory with the help of a tool for collaborative learning. In Section 6, we draw conclusions and give an outlook on future work.

2. Related work

The number of universities that provide remotely controllable laboratories is increasing steadily. While simulations are cheap and easy to establish they can only model a limited degree of realism. A satisfactory simulation of the dynamic behavior of complex systems requires much computing power and even if such computing capability would be available it cannot accomplish real-time quality. Especially, the simulation of embedded computers is very challenging and a simulation of an electron microscope would be still impossible.

Thus, in order to develop and test the real-time behavior of those systems we have to rely on the real hardware instead of simulations. Remote laboratories allow the access to experimental setups that can be operated and controlled 24 hours on 7 days a week. Beside the device that should be investigated, the experimental setups usually comprise several expensive instruments for stimulation of the device and measurement of the device’s reaction. The spectrum of remotely controllable instruments can range from a quite low budget oscilloscope up to an extremely expensive electron microscope.

In the recent literature different categories of remotely controllable experiments were described: Analysis of the characteristics of semiconductor devices [8], setup of electronic circuits and measurement of their behavior [6, 1], physical observations like mechanics experiments [9], or analysis of the characteristics of the upper atmosphere [14], different kind of control systems for studying the parameter setup and analysis of the dynamic behavior of the control loop [10], communication engineering experiments [5, 15], and circuits of digital programmable logic [13].

Beside the description of specific remote lab systems also general distance learning and pedagogical issues were addressed. In [12] a framework for analyzing the effectiveness of remote labs is proposed. [4] mainly focuses on issues that impinge on the specification and design of remote labs. He also identifies the open problems that must be solved to achieve a widespread acceptance of remote labs in education. The main deficits of current remote laboratories concern the lack of a unified appearance of the instruments’ remote control panels, the users’ accessibility and the support for collaboration among the lab’s participants. Only a few papers describe support for user management and toolkits for the experimental setups [8].

In this paper we offer solutions to the deficits cited above. We will demonstrate how traditional instruments can be made network-enabled in a unified fashion. We describe some computer engineering experiments that have been realized by means of a lab-in-a-box approach and we introduce a framework that allows not only for mutual exclusive lab access, but also for advance reservation of experiments, conducting the experiments and a collaborative data interpretation and discussion of the experimental results.

3. Network Interface for Instruments

In order to implement remotely controllable experiments we need network-enabled signal generators and measurement devices. The signal sources are used to excite the device under test (DUT) and its reactions are detected by the measurement devices. Of course, also the DUT setup must be network-enabled.

Unfortunately, most of the traditional laboratory in-
Instruments like function generators, oscilloscopes or digital multi-meters are not equipped with a network interface. Instead, usually serial interfaces (RS232) are provided. Therefore, we need a device that can act as a bridge between these two interface standards.

Lantronix’s XPort [11] provides an embedded ethernet device server that can easily adapt any electronic device with serial capability. Besides the bridging functionality, there is also a built-in webserver. XPort uses a 16 Bit CPU which is clocked at 48 MHz (Figure 1). It is equipped with 256 KByte RAM and 512 KByte flash memory. 384 KByte remain for user data and Java applets while the rest is used by the XPort’s operating system and the server applications (bridge and webserver). By means of a device installer or a DHCP-protocol the IP-address can be configured. As soon as the IP-address is assigned other configurations can be done by a web-frontend. Java applets and html pages are uploaded via the TFTP-utility.

By means of instrument-specific applets we can provide internet access to the instrument that is connected to the XPort. Users only need a web-browser and the URL of the Java applet. Figure 2 shows a screenshot of a remote control applet (RCA) for the Tektronix TDS 210 oscilloscope. This applet can easily be adapted to other 2-channel oscilloscopes by changing the control word sentences associated with the generic parameter functions (e.g., how to set channel amplification or set time base). In this way, Java applets for common laboratory instruments can be used as templates to create RCAs for instruments of other producers.

Beside the bridging and webserver capability, an XPort also provides three parallel output signals that can be used to switch between different DUTs or experimental setups. By means of a four-way analog multiplexer that needs two of those signals as control lines, we can choose between up to four configurations that reside in one Lab-in-a-box to be presented in the next section. The third output can be used as a reset signal. Of course, we can arbitrarily scale this switching possibilities by additional XPorts.

One may argue that the XPorts’ functions could also be implemented by one powerful PC that is equipped with multiple serial interfaces and that serves many instruments. Even though one can proceed in this way the use of multiple XPorts has several advantages: Due to the fact that they use a flash memory, XPorts can be switched on together with the associated instrument. Shortly after power-on they provide the desired network interface to the instrument. Another advantage is the lower power consumption of the XPorts and the better scaling of the webserver throughput. With respect to energy saving it is useful to switch on remotely controlled experiments only when they are actually needed. This is also much simpler with a dedicated XPort server.

4. A Remote Lab-in-the-Box for Embedded Applications

Although our concepts for remotely controllable experiments are suitable for laboratory courses in a wide range of engineering or natural sciences, we are, due to our background, especially interested in their realization in the domain of computer engineering. In a first step we built a lab-in-a-box, the structure of which is shown in Fig. 3, for use in our laboratory course on applied microprocessors, e.g., microcontrollers (µC) and digital signal processors (DSP).

The box contains two microprocessor systems, namely a microcontroller system with a small keyboard and a small LED display (shown bottom left) and a DSP with a stereo Codec to process audio input signals (bottom right). Output
signals of both systems can be watched on the screen of a
digital oscilloscope. A switching unit allows to choose one
of four selectable output signals of the $\mu$C or DSP. Analog
or digital input signals to both microprocessor systems can
be delivered by a function generator.

The two instruments, i.e. the oscilloscope and the func-
tion generator, as well as the two microprocessor units are
connected to the internet by means of the above described
XPorts (symbolized by an X in Fig. 3) via a multiport LAN
switch. In addition, one of those XPorts is used to operate
the above mentioned switching unit, which is a 4-to-1
analog multiplexer, by two of its parallel output signals.

Three webcams, which are directly connected to the in-
ternet, allow to directly observe the displays of the two in-
struments as well as the microcontroller system. This means
that the student has the impression of a real experiment, and
has the assurance that the web frontends of the instruments
and microprocessor units are not merely simulating the ex-
periment. A small lamp inside the box provides enough
light for the webcams to show the displays and their envi-
ronment, so that the user gets a real impression of the ex-
perimental setup. This lamp and all the other devices in the
box get their electrical energy by a power supply which is
controlled by an internet-connected switch. So, to perform
an experiment one first has to switch on the light and all the
necessary equipment and to switch them off when work is
done — just as in a real laboratory. A test installation of the
box is depicted in Fig. 4.

The lab-in-a-box allows to do lots of experiments in the
domain of embedded applications. Some of them can be
done with the same setup, only by re-adjusting the 4-to-
1 switch. Of course, for greater changes between exper-
iments, the lab’s tutor has to do some adaptation job, e.g.
to exchange some devices, or to connect specific hardware
modules to the input/output signals of the microprocessor
systems. The microprocessor system consists of a self-
developed microcomputer, integrating an 8-bit processor, a
keyboard and a seven-segment display. As peripherals it of-
fers two parallel ports, two serial interfaces and three 16-bit
timers. A small operating system (monitor program) allows
or eases to store programs or data and to control the display,
the keyboard and the peripherals. The RS232 Interface is
used for communication with a PC or the XPort described
above.

Fig. 5 shows in detail how an “augmented” application
can be executed with the $\mu$C system. Here, a small radio
controlled clock unit (DCF-77) is connected to the micro-
processor to deliver the exact time of day and date. Students
can watch the signals delivered by the clock module on the
screen of their PC by means of the webcam. There they see
the analog signal received by the antenna of the clock mod-
ule and the generated digital signal (as shown at the bottom
of Figure 5). Furthermore, they can acquire and record this
signal on harddisk by means of the applet controlling the
oscilloscope.

Let us assume that students have to write a program for
the $\mu$C to analyze the clock signal and to show time or date,
resp., on the display of the $\mu$C system. To do this they first
have to work through the course text, which describes the basics for the experiment. Then they can use — locally and perhaps off-line — an integrated software development tool (IDE) and a simulator. Figure 5 shows that the graphical interface of this simulator resembles very much the appearance of the used real microcomputer, so the student can easily operate both. When they believe their program is fault-free and fully operational they submit it to the µC in the lab-in-a-box via internet. The software (a Java applet) for accessing the µC system has also almost the same look as the physical system itself (see left side of Fig. 5). Besides upload or download of programs and data, it allows to remotely control every component of the µC system, including the display and keyboard: all keyboard inputs by the user are transferred to the remote system and all outputs to the display are transmitted back to the controlling applet. When the user operates the clear key C of the graphical interface a special signal is sent to the real microcomputer which is thereby reset into its initial state. Furthermore, the user can watch the display by a webcam — thus realizing that what he sees is not just the result of a simulation. So, when the job of analyzing the RC clock signal is done quite well, the ”real” microcomputer as well as the webcam and the Java applet on the PC screen will show the same exact European time, regardless where on earth the user lives.

Fig. 6 shows how experiments using the DSP can be done. One of the most popular and surely easiest experiments in digital signal processing is to write a program for FIR filtering (Finite Impulse Response). The DSP usually has no control functions to perform. Therefore, the applet to control it and its graphical user interface is rather simple: it only allows to reset the DSP, to upload programs to the DSP and start them, to request an interrupt or to download memory regions from the DSP for further inspection. All actions are monitored in one small window shown in the upper right corner of Fig. 6. To support the students in writing their DSP programs they can use a professional software development tool, including again a simulator.

Fig. 6 shows the execution and evaluation of the FIR program running on the DSP. In the upper left corner you can see the graphical user interface of the applet which controls the function generator. Here one can choose a certain form of output signals with its specific parameters (like frequency, impulse width, etc.). The determined input signal and the resulting (filtered) output signal of the DSP can be watched by the webcam attached to the oscilloscope (lower right corner) or on the virtual screen of the applet controlling the oscilloscope (lower left corner). Of course, the latter signal may also be further processed or stored on hard-disk. At the top of the right side you can see the window to select and upload programs from the client PC to the DSP.

As a last example Fig. 6 shows a more fundamental experiment in computer engineering. Its aim is to acquaint
the students with the programming and practical use of Programmable Logical Devices (PLD), especially the so-called GALs (Generic Array Logic). The deployed device is an in-system programmable (isp) GAL22V10. It consists of a circuit with up to 22 inputs and 10 outputs that can be used either in combinatorial or registered mode. The students can program this GAL chip by means of a Java application that allows a graphical programming of a matrix with approximately 6000 fuses. As shown in Figure 7 the students select the output pin that should be programmed (left upper corner of 7) and the GAL-editor displays the corresponding part of the fusemap that now can be easily programmed by clicking the intersections between input (or feedback) wires and an AND-term of the output function (right lower corner). After programming of all the output lines the final fusemap can be uploaded via the Internet to the remote lab-in-a-box where it will be used to configure the GAL-chip. Some of its outputs are connected to the inputs of an oscilloscope and can thus be observed by means of a webcam or the virtual oscilloscope shown on the screen of the user PC.

5. Collaborative Web Platform

An indispensable component of a remote lab is the web platform that supports both mutual exclusive access to the lab-in-a-box and collaboration of student working groups. In the GridLabs project supported by the German Ministry of Research and Education (BMBF) we developed a Collaborative Remote Laboratory (CRL) server that integrates collaboration support and remote access to real laboratory equipment. To provide an intuitive software interface we use the metaphor of a virtual building that consists of virtual rooms augmented with real experimental equipment [7]. As in real life these augmented virtual reality rooms are connected by floors that can create hierarchical structures. Virtual rooms can only be entered if one holds a valid virtual key. A remote lab for one or more related experiments comprises several virtual rooms for different purposes (see Figure 8):

- main entrance hall,
- administration room,
- communication and support room,
- materials room,
- show room, and
- one or more lab rooms.
All the rooms provide facilities for asynchronous and synchronous communication between the lab’s participants (threaded email and chat respectively). Moreover, in the communication and support room students can meet tutors to get more qualified help on specific problems. Also in all rooms of the CRL, so called *pages* of shared data can be stored and accessed by everybody who holds a virtual key for that room. Two kind of pages can be distinguished: web-editable content pages (WIKIs) and binary pages (files) for the exchange of data by means of external applications. In contrast to [7] CRL allows not only access to pure virtual rooms but CRL is an extension of CURE that simultaneously supports the access to real remotely controllable devices.

Every student gets a key for the main entrance hall. In the materials room the students find detailed descriptions of the experiments, assignments and user manuals of this laboratory’s instruments and DUTs. In the showroom slide shows and tutorial movies explain the essential procedures for remote experimentation. Students can also create private (sub)rooms for storing their individual results of laboratory sessions, which are conducted in one of the lab rooms. In order to collaborate with other students they can make copies of their personal keys and pass copies of it to their colleagues. In this way, the room becomes a (virtual) group room.

In contrast to the rooms described so far lab rooms can only be entered for a limited time period. During this time a mutual exclusive access must be guaranteed. To accomplish this we generate a bunch of virtual keys that are only valid for a given time slot. The time slots are fixed to a given time period (e.g. 15 minutes each) which can be easily changed by the guidelines of the laboratory’s organizer. In the administration room a key rack provides all time-limited keys that are still available. If there are $n$ equivalent lab rooms (actually each one requires a lab-in-a-box) there will also be $n$ keys for each time slot. Thus the operational availability can be easily adapted to the needs by adding new lab-in-a-box modules and corresponding lab rooms.

By taking keys from the rack (in a FCFS-manner) the students reserve a lab room for a specific time slot. Because there is only one key per lab room and time slot mutual exclusive access is guaranteed. The key holder can only act as an experimenter during the corresponding time slot of the key. He can also create and pass observer keys which allow the access of the lab room as an observer but not permit the control of the experimental setup. The results of an experiment will be stored in pages of a group room where they can be discussed within the working group and prepared for submission to their supervisor.

6. Conclusions

We have presented a lab-in-a-box, a flexible infrastructure for setting up experiments to be performed over the internet. The lab-in-a-box is operational and will be used in the next laboratory course, running in the coming fall semester. The experiments there concentrate on embedded systems, i.e. microcontrollers and digital signal processors, however the applicability is much wider. The initial effort to program control applets for all devices and instruments looks much higher than it really was, because after the first applet, we could re-use much of the code, as many instruments are controlled in similar ways, by sending control strings and receiving result strings.

In the future, we would like to incorporate our remote lab in the GridLabs project, which already was envisioned in [3]. In this effort, several remote labs are to be shared among several virtual organisations in order to be able to provide a much larger spectrum of instruments and experiments.

References


Students’ Experimental Processor: A Processor Integrated with Different Types of Architectures for Educational Purposes

L S K Udugama, PhD
Department of Electrical & Computer Engineering,
Faculty of Engineering Technology,
The Open University of Sri Lanka.
udugama@ou.ac.lk

Janath C Geeganage
Department of Electrical & Computer Engineering,
Faculty of Engineering Technology,
The Open University of Sri Lanka.
janathg@gmail.com

Abstract

Students who are beginners face difficulties in understanding basics of Computer Architecture. Their problems have been identified, and a project has been initiated to address these problems, with defined objectives.

The project is to be implemented in stages: design a processor; build a simulator; develop a compiler and develop an intergraded system. The paper presents the results of the first stage – design a processor. The features and characteristics of the processor are defined and the design process is described in detail.

The SEP (Students’ Experimental Processor) integrates different types of architectures: Memory-Memory, Accumulator, Extended Accumulator, Stack, Register Memory, and Load Store. It can be switched to any one of them. It was modeled using VHDL and is ready to be implemented on FPGAs.

The SEP will support the introductory level students to understand the characteristics of computer architectures and their operations. Future developments of Computer Architecture education using the processor are also discussed.

1. Introduction

The Department of Electrical and Computer Engineering of The Open University of Sri Lanka offers specialization in Computer Engineering in the engineering degree program. In the year 2002 a new curriculum was introduced where Computer Architecture is one of the major components in the computer-engineering stream.

Computer Architecture education in our department is basically covered in four different courses taught at different levels. The first course, Communication and Information Technology is taken by freshers, where they are introduced to the components of a processor, how they are interconnected and how they function. Even though theory is explained by using a hypothetical machine, students use a simulator for the 8051 microcontroller to do their exercises. The second course at the next level covers microprocessor-based systems. Here students use the 8051 microcontroller development boards to do their laboratory work. At the next level, students study the five classic components of a computer [6] and other related areas such as performance and peripherals. At the final level students will learn to design a processor and they implement their processors on FPGA development boards in the laboratory.

Students, especially at lower levels, face difficulties in understanding basics of Computer Architecture. Some of these difficulties have been identified, and remedies suggested [1, 5, 7] According to our experience, we can highlight some of the problems faced by our students:

- Understanding the functions of the basic components of a processor.
- Visualizing how a processor actually executes an instruction and how data flows through its components.
- Assembly language programming of complex commercially available microprocessors/ microcontrollers.
• Comprehending the basic types and concepts of Computer Architecture.

We have initiated a project to find solutions to these problems. Here, we have mainly focused on the lowest level course, targeting students who hardly have any knowledge of computers/processors. However we do not want to limit it to one particular course, but would like to see the possibilities of catering for higher level courses as well. Thus, the following objectives have been drawn up:

• To design a simple processor/s to teach different basic types of architectures, to demonstrate functions of components in a processor and how a processor functions.
• To use the same processor/s to do exercises in the subject.
• To build a simulator for the processor/s and use the same to demonstrate functions of the components in a processor and to write assembler programs and execute them.
• To keep provision to use the same processor for microprocessor development systems and to teach concepts in computer architecture.
• To use the processor in the course on, Processor Design, at the final level, as a case study.

We have searched the literature to try to locate a suitable system that meets all our objectives, but found that while some of the systems [3, 4, 8] and simulators [1, 2, 5] did cover our objectives severally no system could be found to cover all in our objectives.

We then decided to commence this project called Computer Architecture Learning System of The Open University of Sri Lanka (CALS-OU) to achieve our target. The project will be implemented in stages: design a simple processor; build a simulator to write, debug and execute assembly programs; develop a compiler to write programs in a high-level language; and finally develop a hardware/software integrated system facilitating students to learn concepts in Computer Architecture and to use the same for doing students’ experiments. The first stage of the project has been completed, and this paper presents its results – a simple processor designed and named Students’ Experimental Processor (SEP), which will accomplish our objectives.

2. How We Made the SEP a Reality

At the very outset of the design we defined the features and characteristics of the processor to be. The first is that the processor should comprise of four basic types of architectures: Accumulator, Memory-memory, Stack and Load Store. We then decided to include other two types, register memory and extended accumulator architectures [6]. We thought of including these as popular processors have these architectural styles. The next feature is that the processor should be able to change to one of the six architectures using hard or soft switches. To keep the processor simple and to make it easy for the programmer, we decided to limit the number of instructions to less than 32 and leave the I/O instructions out. However we did not want to limit the number of addressing modes as it is useful for students to learn to handle them in different situations.

The design process was comprised of the following key steps:

• Design of ISAs and applicable addressing modes, dimensions of data and address buses, size and the number of GPRs and depth of the stack.
• Design of Individual Data paths for the six architectures.
• Design of control sequences of the different instruction classes.
• Design of the top-level architecture. This is the architecture, which integrates all six architectures within one processor.
• VHDL modeling.
• Translator development.
• Testing.
• Documentation.

In the rest of the paper the design process, problems faced and the solutions arrived at are described.

3. Instruction Set Architecture

Basic instructions are included in the instruction set to provide an overall understanding of ISAs. All instructions can be grouped as Arithmetic & Logical, Control, Data Movement and miscellaneous instructions (Table 1).

Same set of instructions in each of the groups Arithmetic & Logical and Control Transfer is applicable to all architectures. Only the Data Movement instructions differ from machine to machine. Control Transfer instructions refer the Flag register. The Flag register consists of 5 bits corresponding to the Parity flag, Sign flag, Overflow flag, Carry flag, and Zero flag. Instruction count is less than 32 for all machines.
Seven addressing modes are implemented in this processor. They are: immediate, direct, indirect, register direct, register indirect, index and displacement. Except displacement addressing, others are available for the user for accessing operands. Displacement addressing is used by the hardware in Control Transfer instructions. Operands can be in the memory, register bank, stack, or accumulator, depending on the active architecture. Addressing modes are declared in a way that it includes general modes and some popular modes. For clarity and consistency, the same instruction format is applied in all architectures.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Arithmetic and Logic</th>
<th>Control Transfer</th>
<th>Data Movement</th>
<th>Misc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory to Memory Accumulator</td>
<td>ADD, SUB, INC, MUL, DIV, AND, OR, NOT, XOR, SHL, SHR, ROL, ROR</td>
<td>JC, JS, JP, IZ, JOF</td>
<td>LOAD, LOADacc, STOREacc, PUSH, POP, DPP, SWAP, ROTATE3, LOADacc, STOREacc, LOAD, STORE</td>
<td>nop</td>
</tr>
<tr>
<td>Stack</td>
<td></td>
<td>Call Ret</td>
<td>Looz</td>
<td></td>
</tr>
<tr>
<td>Extended Accumulator</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register Memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load Store</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instructions are in two-address format. Targeted implementation device has about 512K of 16-bit memory. A 19-bit address bus is required to address 512K locations. In an instruction there are two addresses (where applicable) and it needs 38-bits.

Instruction format (Figure 1)
- Opcode → 5-bits
- Addressing mode selection → 3-bits
- Addresses of two operands → 38-bits

Fig. 1. Allocation of bits in an instruction

Altogether 46-bits are required for an instruction. It occupies 3 memory locations. The length of the instruction is 48-bits.

4. Number Representation

Arithmetic and logic unit supports signed fixed-point numbers. Negative numbers are represented in two’s complement form. The sign flag is replicated over bits 16 to 19, because the ALU is 19-bit and the data path is 16-bit. It is achieved by coupling the MSB to the three higher order bits.

5. Data Path Design

The structure of these architectures was designed in a simple manner. It has only the basic entities, which clearly shows the operation. For example, memory-to-memory architecture consists of the following entities:
- IP; Instruction Pointer
- MAR; Memory Address Register
- IR; Instruction Register
- OP1; a 16-bit register
- SR; Status Register/Flag Register
- MDR; Memory Data Register
- 3S_buff; a three state buffer
- Six multiplexers

5.1. Entities

Each of the architectures has a different data path, which facilitates the characteristic operation of its style. Almost all entities are effective in all architectures. There are two types of entities, namely, synchronized and asynchronized. Synchronized entities will function in an edge of the clock signal. Others change the output whenever a respective input is changed. Multiplexers and the ALU are non-synchronised entities. All registers are synchronised to the rising edge of the clock. Types of entities used in the processor are master-slave registers, shift register, 3-state buffer, register bank-stack-accumulator combined unit, Asynchronous SRAM and
multiplexers. As the Register bank-stack-accumulator is a special unit, the block diagram is shown in the figure 2.

![Block Diagram](image)

Figure 2. Interface of the Register bank, stack and accumulator combined unit

5.2. Handling Different Sized Busses

In indirect addressing, the address of the location retrieved from memory is 16-bit. It has to go to the MAR. MAR is 19-bit wide. In this case only the lower 16-bits are replaced. The data path facilitates this by coupling 3-MSBs of output of MDR to the input. This technique is used in several places to overcome the above situation. It results some limitations to the design.

5.3. Control Signals

Figure 3 shows the data path of the top-level processor, which facilitates six architectures mentioned above. Control signals to each entity can be found in the same figure.

6. Control Unit Design

A timing diagram was drawn for the execution of each instruction class to determine the sequence of control signals. The Control unit was designed using these timing diagrams.

“Functioning of the entities by the control signals” was important in designing the control unit, as the right signal has to be sent at the right time. A document containing all the entities and their control signals was prepared at the commencement of the design of the control unit.

7. VHDL Modeling

We used the Xilinx ISE Webpack 6.3i for VHDL modeling and for the simulation ModelSim 6.0a.

7.1. Modeling Strategy

Initially, the lower level entities were developed and tested for the expected results. When all entities are working successfully as individual entities, they were connected together to make the data, address and control paths according to the design done in the data path design process.

Behavior models were used in developing individual entities. Structural modeling was used in creating the top-level entity.

Then the control unit was programmed. Control states declared in the earlier stage were programmed using VHDL. The Control unit was tested as a stand-alone unit, and as an integrated entity, to verify its behavior.

Integrated testing was carried out after construction of the control unit.

7.2. Register Bank, Stack and Accumulator - The 3 in 1 Unit

Register bank, stack and the accumulator in the combined architecture was developed in a different way. As far as these three units are concerned, it is clear that two of them won’t be available in a single architecture. Accumulator architecture doesn’t have a register bank or a stack. Stack architecture doesn’t have a register bank or an accumulator. Because of this, these three units were combined and developed as a single unit, which switches the operation according to the selected architecture. It optimizes resource usage.

8. The Assembly Code Translator

This system had to go through a testing process to confirm the proper execution of all instruction classes in the six architectures. In the simulation, machine level instructions had been fed to the memory module of the VHDL model. It needs converting instructions to operational code and addressing modes (if any) and operands to binary. The binary stream needs to be split into 16-bit segments to fit into the memory locations,
Figure 3. Data path of the top level architecture
starting from the initial memory address. This process was time consuming. One mistake in conversion would produce an entirely different result.

The translator was developed to overcome this situation. The major functions performed by the translator are:

- Validation of instructions, addressing modes and operands
- Automation of machine code generation
- Automation of VHDL memory initialization file creation
- Ensuring the reusability of programs

The Translator was developed using Microsoft Visual Basic 6. It uses a Microsoft Access database to store the written programmes for future reference. Figure 4 shows the interface of the translator.

![Figure 4. Interface of the translator](image)

The output of the translator is a text file, which also contains memory initialization data of the program written by the user. However, the user has to copy the contents of that file and paste it in the memory initialization section within the VHDL memory module. Using the translator makes the programming processes interesting.

Even though this was developed originally for testing, it provides a lot of benefits to the users, and helps to make their life easy. Moreover the students can use this translator to write their programmes and generate the binary code mapped to the memory. It makes life easy and saves a lot of time.

9. Testing

Testing was carried out to verify the proper operation of

- Each entity
- Each instruction class
- Each addressing mode under each instruction class
- ALU for status flag settings
- Synchronous operation
- Special cases such as register bank, stack and accumulator combined unit
9.1. Testing Strategy

A bottom-up testing strategy was followed. Bottom level entities were tested and verified first. Followed by integrated testing. A testing check-list was prepared to measure progress, and to make the process an organised one.

9.2. Test Results

Successfully tested programmes were stored under the respective architecture. The system was checked with those after a modification is done, to verify the results.

10. Implementation

The target implementation device of this processor is the Spartan-3 FPGA board developed by the Xilinx Corporation, with the following features:

- Affordable price
- 400,000-gate Platform FPGA – XC3S400
- Xilinx 2 Mbit Platform Flash configuration PROM – XCF02S
- 1 Mbyte of Fast Asynchronous SRAM
- Daughter card expansion ports

11. Documentation

As the target audience of this project is students, a considerable effort was made in producing proper documentation. It was decided to compile the documentation in separate documents. They are:

- Programmer’s Manual (for the six processors) consists of detailed descriptions of six ISAs
- Description of the Processor Architecture, which includes the Functioning of entities over the control signals, Processor Architecture diagrams, Processor architecture, Abstract views of six architectures, Detailed control sequences of the Memory to Memory and Load Store architectures.
- Source code listing

All documentations and related programs are available on web http://www.ou.ac.lk/fac_etec/elec/udugama/.

12. Conclusions and Future Work

All six architectures are working perfectly according to their characteristics.

There are some limitations of this design. Call Return and indirect addressing can only be used in the same segment. In case where a Call instruction is called within a Call, the processor expects the compiler to do the necessary bookkeeping. This is also the case in the Looz instruction. For simplicity of the system, some of them were kept as it is.

Register bank, stack and the accumulator were combined into one unit, to minimize the resource usage on the FPGA. It automatically switches to the relevant storage according to the active architecture.

The next stage of this project is to develop a simulator for the processor. At the same time, we hope to develop a compiler for a high-level language as well. Once they are ready, it will be a good tool for beginners to learn basics of Computer Architecture.

However we hope to use this processor as a case study during this academic year for the course in Processor Design. It is possible to examine the effect of each control signal in each clock cycle using a VHDL simulator. Therefore it will be very helpful to students who are familiar with ModelSim to appreciate the internal operations of a processor.

There is a possibility to develop a microprocessor development board using the SEP. It is possible to implement the design on the FPGA as we targeted for Xilinx Spartan 3. In addition to that the processor is designed to interface the onboard 1 Mbyte asynchronous RAM as well. Therefore Spartan 3 board can be used as a microprocessor development board. However the interfacing units, keypad and a display have to be designed and connected to the board.

This processor can be improved further by pipelining instruction execution.

13. References


