# Harnessing FPGAs for Computer Architecture Education

Mark Holland, James Harris, Scott Hauck
Department of Electrical Engineering
University of Washington, Seattle, WA 98195, USA
{mholland, harrisjs, hauck}@ee.washington.edu

## Abstract

*Computer architecture is often taught by having students use software to design and simulate individual pieces of a computer processor. We have developed a method that will take this classwork beyond software simulation into actual hardware implementation. Students will be able to design, implement, and run a single-cycle MIPS processor on an FPGA. This paper presents the major steps in this work: the FPGA implementation of a MIPS processor, a debugging tool which provides complete control and observability of the processor, the reduction of the MIPS instruction set into the eight instructions that will be used by the processor, and an assembler that can map any MIPS non-floating point instruction into this set of eight supported instructions.*

## Introduction

Computer Design and Organization is a common upper-level engineering course that is offered at universities throughout the world. In this class students often learn computer design by implementing individual pieces of a computer processor. This approach has important limitations: while students can complete and simulate their designs using software, they do not have the chance to implement and run their designs in hardware.

This paper presents our solution to this problem: an FPGA implementation of the MIPS processor that we will introduce into the classroom (the processor is from *Computer Organization and Design* by David A. Patterson and John L. Hennessy [1]). The MIPS processor is commonly used in the teaching of computer architecture, so our work will be very portable. In addition to the processor, this paper presents a debugging tool that will be used to debug the processor and an assembler that reduces the MIPS instruction set down to a functionally complete subset of 8 instructions that will allow a simpler classroom implementation.

These tools can be used to enhance computer architecture education through the following process: we implement selected CPU parts onto the processor, give the incomplete processors to students, and allow the students to design and integrate the missing pieces. The processor debugging tool will provide the students with complete control over their processor, allowing them to debug and fix their designs. Reducing the MIPS instruction set to eight instructions allows us to use a simpler implementation of the processor, and it also simplifies the parts that the students will design. Our assembler can reduce a file of any non-floating point/coprocessor instructions into an implementation using only the set of eight supported instructions, outputting a machine language file that can be loaded directly into the processor.

Combined in the classroom, these tools allow students to implement and run their processor designs in hardware. Additionally, the reprogrammable nature of FPGAs makes them perfect for educational purposes because they can be reused year after year, resulting in low costs.

## FPGA Implementation of the Processor

The most important task in our work was to implement a single-cycle processor onto an FPGA so that it would look and behave like the processor presented in *Hennessey and Patterson*. We chose to use a XESS-XSV board for our implementation [2]. This platform includes one XILINX Virtex XCV300 FPGA, two independent 512K x 16 SRAM banks, one parallel port interface, and one push-button switch.

Nearly all of the control and the datapath of the processor can be implemented on the FPGA. The only exceptions are the data and instruction memories, which are implemented in SRAM due to a lack of space on the FPGA. The parallel port is used to load configurations to the FPGA as well as communicate with the PC, which is necessary for the debugging tool. The push-button is used to reset the processor.

Communication between the FPGA components and the off chip SRAM is controlled by a high frequency clock that is hidden from the students. A lower frequency clock (16 times slower) is used to control the actual processor, and is the clock that the students observe and control.

## Debugging Tool

The debugging tool is necessary for making our FPGA-implemented processor usable in the classroom. The tool allows the students to observe and control the internal states of their processors, which is critical for the designing and debugging processes. We chose to create the debugging tool using Visual Basic.

Communication between the debugging platform (a PC) and the processor occurs via the parallel port. The debugger provides complete control and observability of the processor. The tool supplies two methods for controlling the clock: one runs the clock for a specified number of cycles and one runs the clock until the user gives a stop signal. The student can read data lines simply by selecting one of the defined sources: the program counter, instruction memory, data memory, register 1, register 2, ALU, MISC1, or MISC2. MISC1 and MISC2 are lines that the student can connect to any part of the processor, providing added debugging flexibility. Students can write to the program counter, register file and memories by supplying the address and data value to be written, with the exception of the program counter which does not require an address. They can perform reads of the register file and memory simply by supplying the read address.

## Reduction of MIPS Instruction Set

We felt that implementing a processor with the complete MIPS instruction set would be both overly time-consuming as well as extremely taxing on the available FPGA resources. We feel that the students will obtain the same educational benefit using a reduced MIPS instruction set. This set of eight instructions provides functionality identical to the original set. The eight instructions are: NOR (bitwise NOR), SUBU (subtract unsigned), LW (load word), SW (store word), BGEZ (branch on greater than or equal to zero), JALR (jump and link register), SYSCALL (system call), and BREAK.

Different approaches are used for each type of instruction that we wanted to reduce out of the set. For example, an ADDU (add unsigned) can be accomplished by subtracting one of the values from 0 and then subtracting that value from the other. Similarly, an AND is accomplished by negating both values and then performing a NOR.

## Assembler

We have designed an assembler that is capable of mapping any of the MIPS non-floating point/coprocessor instructions to our set of eight instructions. The assembler is written using LEX and YACC.

The assembler accepts an input file of assembly language code and goes through two basic steps. First, it maps the instructions to the set of eight that is supported by the processor. Secondly, it converts the mapped assembly language code into machine language. The machine language file outputted by the assembler can be directly loaded into the processor on the FPGA.

Most of the MIPS instructions can be directly mapped to the supported set of eight, with the exception of immediate instructions. Since our processor does not support any immediate instructions, there can be no direct way to access the immediate value. Thus whenever the assembler sees an immediate instruction, it writes the immediate value into a data memory section that appears after its instruction memory section. A load instruction is then added that transfers the immediate value from memory into a register, allowing the processor to access the immediate as if it were a simple registered value.

## Classroom Integration

We will provide students with access to the following: an XSV board, access to a computer with Xilinx Foundation Software, our processor debugging tool, a parallel port connector, a Foundation project containing the base framework for our processor, and a power supply for powering the board. With these tools the students will be able to design and implement a processor on their FPGA.

An early assignment for the students will likely be to design the register file. Each student will create their register file and add it to the Foundation project, then implement the design onto their FPGA. Once completed, they can easily use the debugging tool to verify their register file design and fix any errors they find. Other projects would be the designing of an ALU and control logic as well as progressing through a single-cycle to a pipelined processor design.

## Results and Discussion

We have implemented both a single-cycle and pipelined processor using our proposed framework. For the single-cycle implementation we were able to fit our design quite easily onto a XILINX Virtex XCV300 PFGA, using only 22% of the LUTs and 25% of the BRAM units. This FPGA runs at a maximum frequency of 25MHz, which results in a processor speed of 1.5MHz (25/16). The pipelined processor produced similar utilization numbers, using 29% of the LUTs, 25% of the BRAMs, and running at 23 MHz (1.4MHz processor clock).

The debugging tool has also been tested, and successfully supplies a user with complete control of the processor even when the processor is incomplete. Additionally, an undergraduate student has successfully implemented designs onto an FPGA using our tools and provided us with feedback that was applied to our designs.

Further information on this work can be found in [3].

## References


[1] Patterson, D. A., J. L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, San Francisco, 1998.

[2] XESS Corp., *XSV Board V1.0 Manual*, March 1 2000.

[3] Holland, M. *Harnessing FPGAs for Computer Architecture Education,* M.S. Thesis, University of Washington, 2002.