The Progressive Learning Platform

David J. Fritz, Wira D. Mulia, Sohum A. Sohoni
Oklahoma State University
{david.fritz, wira.mulia, sohum.sohoni}@okstate.edu

Abstract

The progressive learning platform is an FPGA based computer architecture learning platform. It is intended to expose students to the a wider scope of computer architecture design principles beyond the usual topics covered in an undergraduate course. This includes the typical core concepts - data path, control, pipelining, etc., with other additional, critical concepts such as full system integration, programmability, and other design trade-offs on a wider scope than the core architecture.

This is accomplished with a problem-based curriculum that takes teams of students through a complete system on a chip (SoC) design. The included reference design includes a MIPS-like CPU with surrounding system components including a VGA controller with framebuffer, UART, gpio, switches, buttons, leds, interrupt controller, memory controller, and more all connected through a standard front side bus. The system is designed to be used with student generated designs. Additionally, software tools, including a simple 2-pass assembler, board communication tools, and a cycle accurate simulator are included. All host software tools are written in Java to be OS agnostic. The reference design is implemented in Verilog and is fully synthesizable in Xilinx Webpack ISE.

1. Introduction

Choosing and effectively implementing a Computer Engineering laboratory component is a challenging task. It is important to choose relevant and engaging laboratory work that students can use and apply in future coursework without overwhelming them. Students learn best from experience gained using real programs on real systems [1]. Unfortunately, many laboratory components rely on small example problems and abstract simulations, from which students have difficulty relating to the real operation of computing machines.

In this paper we introduce the Progressive Learning Platform (PLP), an FPGA based system on a chip with a complete software stack including an assembler and cycle accurate simulator, and an accompanying curriculum. The PLP system is designed to be used in a number of Computer Engineering courses, beginning with an introductory logic design course and ending with a graduate level computer architecture course. Using a single system throughout a number of courses reduces the learning curve students often face in design laboratories, and provides a means for students to make meaningful learning connections between courses.

The PLP system comprises of a target FPGA development platform (many can be used, as the reference design is simple to port), a reference hardware design defined in behavioral Verilog including the core, bus interface, and numerous modules (VGA controller, timers, UART, GPIO, etc.), a complete Java based software stack including an assembler, cycle accurate simulator, and programming tools, board software libraries, and an accompanying curriculum. This paper describes the current state of the PLP system, which includes the second revision of the hardware and software stack, designed for use in an undergraduate computer architecture course. We also describe how we use the PLP system in the classroom.

2. The Progressive Learning Platform

The progressive learning platform (PLP) is a complete, FPGA based system-on-a-chip designed for use in a computer engineering curriculum. The PLP system includes the hardware design itself, an assembler, cycle accurate simulator, programming tools, and example and module interface codes. All hardware is defined in Verilog and the host software is written in Java. All components are freely available, and licensed under the GNU General Public License.

2.1 Hardware

All hardware is defined with Verilog HDL. Currently the platform targets the Digilent Nexys 2 FPGA development kit. Although a target board is used for development, and we use proprietary tools for synthesis, the system is designed to be platform agnostic. All HDL is defined behaviorally and special structures such as block RAMs are generically defined to aid in proper inference for a number of targets. PLP is also designed to be highly portable. In general, porting the reference design to another target board requires only that unsupported modules be removed from the module build manifest, timing constraints be updated, and pin assignments be made.

The hardware design includes the CPU, a bus arbiter, and a number of memory mapped modules, as shown in Figure 1. The CPU is a MIPS-like machine with a 5-stage pipeline and 23 instructions. A CPU block diagram is shown in Figure 2, and a table of instructions in Table 1. The datapath for the CPU is shown in Figure 2. All hazards are forwarded except for the load/use hazard, which generates a single cycle stall.

The bus arbiter provides a dual instruction/data bus for interfacing modules with the CPU. All modules are multiplexed on the bus, and are selected by a compile time generated memory map. All modules follow a common bus interface that requires two ports (instruction and data). The bus arbiter is automatically generated at compile time by a script that reads memory map information from each of the modules. This is done to simplify adding any new modules. A new module can be added to the system by conforming to the module port and timing specification, and adding a special memory map declaration, embedded as a Verilog comment in the module. The memory map declaration specifies the start address and segment length in memory. Figure 3 shows the declaration of a PLP module in Verilog. The first several ports are predefined, allowing additional ports as necessary. The comment immediately before the module declaration assigns the memory map base address and length.

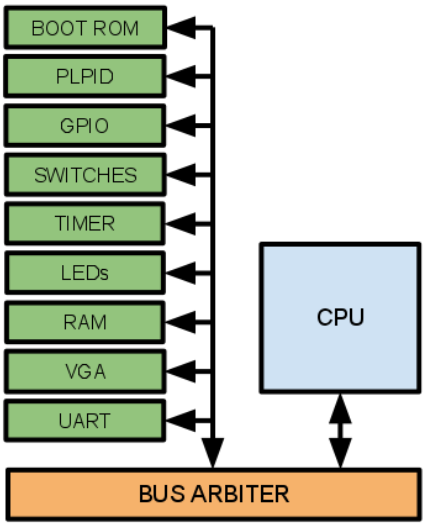

Figure 1 – The Progressive Learning Platform block diagram.

Table 1 – PLP ISA

Instruction	Syntax
Unsigned add	`addu $rd, $rs, $rt`
Unsigned subtract	`subu $rd, $rs, $rt`
Logical AND	`and $rd, $rs, $rt`
Logical OR	`or $rt, $rs, $rt`
Logical NOR	`nor $rt, $rs, $rt`
Signed compare	`slt $rd, $rs, $rt`
Unsigned compare	`sltu $rd, $rs, $rt`
Shift Left Logical	`sll $rd, $rt, shamt`
Shift Right Logical	`srl $rd, $rt, shamt`
Jump Register	`jr $rs`
Jump and Link Reigster	`jalr $rd, $rs`
Branch on Equal	`beq $rt, $rs, label`
Branch on Not Equal	`bne $rt, $rs, label`
Add Immediate Unsigned	`addiu $rt, $rs, imm`
Logical AND Immediate	`andi $rt, $rs, imm`
Logical OR Immediate	`ori $rt, $rs, imm`
Signed compare immediate	`slti $rt, $rs, imm`
Unsigned compare immediate	`sltiu $rt, $rs, imm`
Load Upper immediate	`lui $rt, imm`
Load word	`lw $rt, imm($rs)`
Store word	`sw $rt, imm($rs)`
Jump	`j label`
Jump and Link	`jal label`

The modules provide support for board level I/O, memory, and other internal components such as timer and interrupt hardware. The core set of components is driven by the available hardware on the reference target development kit, as well as the perceived educational value of certain components. Modules included in the core set are a UART, timer, identifier module, interrupt controller, bootloader ROM, and memory (either as a block RAM or off-chip interface). Additional support modules include a VGA controller which operates at 640x480x8 with frame buffer support, LED controller, GPIO, switch controller, and seven segment driver.

2.2 Software

The PLP software tool is a collection of Java programs that constitutes a computer architecture development and simulation framework. The framework supports development of an ISA with major

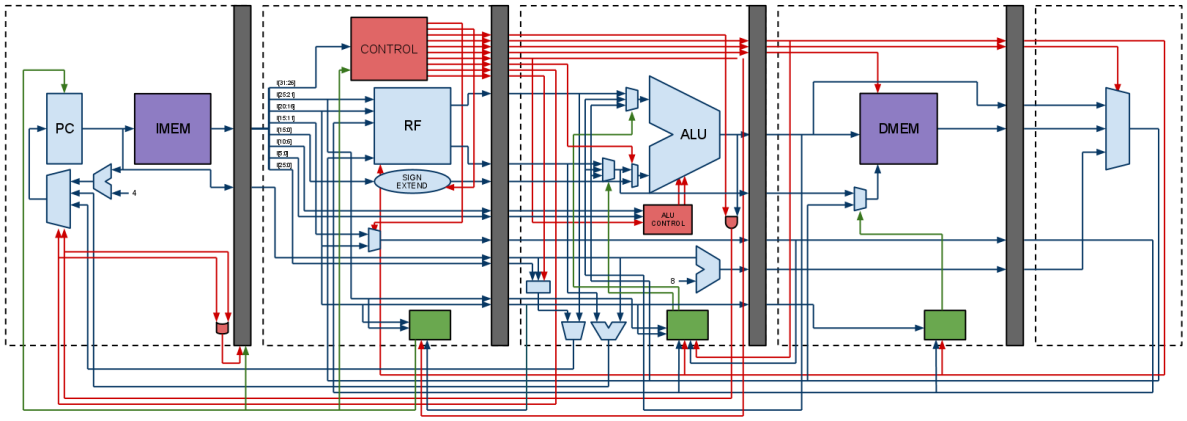

Figure 2 – Block Diagram for the PLP CPU. Student teams work on designing and implementing a functionally equivalent design.

focus in education by providing a set of intuitive interfaces for computer engineering faculty and students to develop, extend, and ultimately share back to the larger community. We develop a MIPS-like architecture package using this framework to be used along with PLP hardware in various computer engineering courses. These courses can range from introductory computer systems course to advanced computer architecture. This package will include a visualization tool to be used by educators to help convey computer architecture concepts such as data path, CPU control logic, pipelining, hazards, and performance evaluation. The software is written in Java and licensed under GPLv3 to fulfill the openness vision for the project.

2.2.1 The PLP Software Framework

The framework consists of a collection of Java classes that allow developers to build a development and simulation environment for their CPU architecture. Figure 4 provides a general overview of the system, and shows the structure of the framework.

The development framework consists of an assembler interface, a linker, and a GUI development environment. Developers will need to implement the functions listed in figure 4 to integrate with PLP software stack. User program flows as each object is instantiated, i.e., each assembly source file will be attached to an assembler object, and in turn these objects will be passed on to the linker. The simulation core can either receive a linker object or the assembler object, depending on the level of abstraction the system requires. A binary image might not even be necessary for the purpose of simulation; in fact, the source code can be passed on to the simulation core as is and processed during simulation, if the developer chooses to. Users can refer to javadoc-generated documentation [2] of PLP software tool under the root `plptool` Java package for class definitions.

The simulation framework consists of a single Java class implementing member methods of the PLP simulation core abstract. These methods include a program load procedure, a reset handler, and a step function. How a "step" is defined is up to the simulation core developer: it could be a full cycle of the clock, or a phase of the clock. Currently, developers will have to implement the simulation core from the ground up using Java. Future work will include a simulation shell with a well-defined interface where users can drop in HDL modules to drive the simulation. The final goal for this environment is an intuitive GUI core builder usable by undergraduate computer engineering students to generate custom simulation cores.

The framework also implements a module interface that can be used to attach modules to the bus of the simulation core as shown in figure 5. The bus will call read or write functions of these modules whenever the CPU tries to access an address mapped to the module.

```
/* PLP2MODULE start:0xf0200000 length:0x00000100 */
module my_module(rst, clk, ie, de, iaddr, daddr, drw, din, iout, dout);
```

Figure 3 – Defining a PLP hardware module. Notice that a Verilog comment is used to assign the memory map, which is generated at build time.

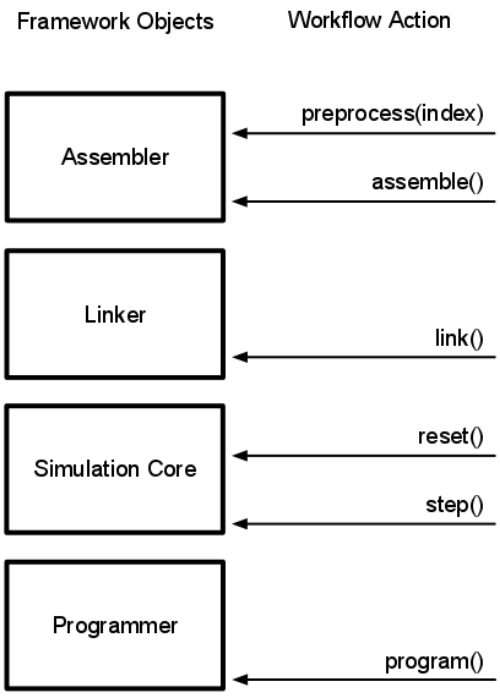

Figure 4 – PLP software framework. The function listing lists all the program methods that developers need to implement to integrate with PLP tool.

The access functions can be overridden by the developers to reflect the actual module or simulated device behavior. Evaluation functions are called by the simulation core whenever outputs are needed to be displayed to the user. Modules implemented using this interface can range from common I/O devices such as switches and LEDs, to tracers and cache simulators.

A GUI frame can be attached to the module for display, or control, purposes. The `plptool.mods` package in the source code tree contains modules already written for the system as examples. This includes an LED array, switches, and a memory access tracer. `IORegistry.java` contains registration information to load or unload modules during run-time. Figure 6 shows a simulation with the LED and Switches modules attached.

2.2.2 PLP ISA Implementation

PLP software tool includes an implementation of a MIPS-like ISA, development tools, and simulation core to be used along with the hardware implementation of the architecture. This implementation is a cycle-accurate simulation of the PLP CPU described in the hardware section of this paper. This implementation also includes an assembler that generates an object code compatible with the PLP architecture Verilog HDL and simulated CPUs. This allows users to write programs and run them in the simulator or download them to the FPGA board to be run on real hardware.

The simulation core implements a cycle-accurate 5-stage pipelined MIPS-like processor. A forwarding unit is also included with the option of turning it off if needed (for educational purposes).

A "step" in this simulation core simulates a full cycle of the clock. Refer to figure 7 on how this event is abstracted. The simulator assumes that the simulation is paused between and falling and rising edges of the clock, where all combinational logic outputs are assumed to be stable. The pipeline and program counter registers are assumed to be positive-edge triggered, and the bus and register file are negative-edge triggered. Once new values are propagated across

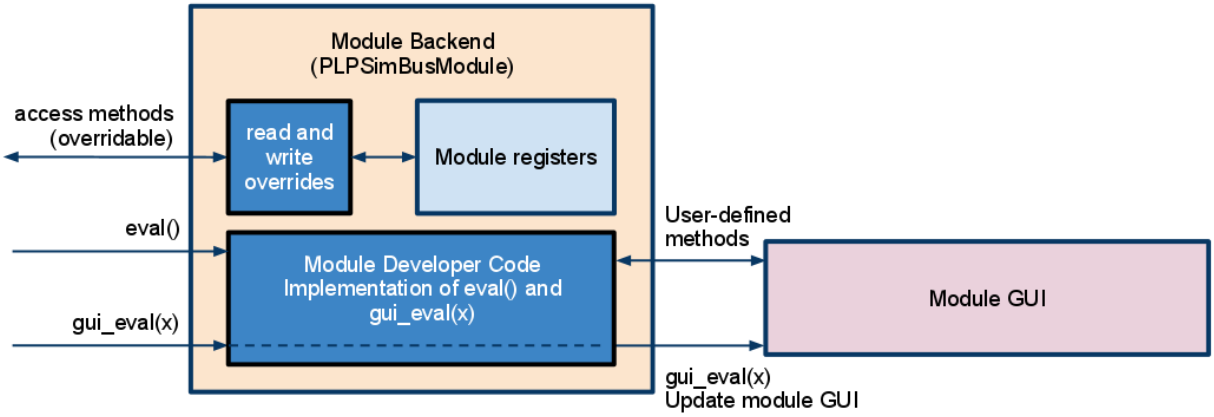

Figure 5 – PLP software module architecture, showing abstract methods and members of the class.

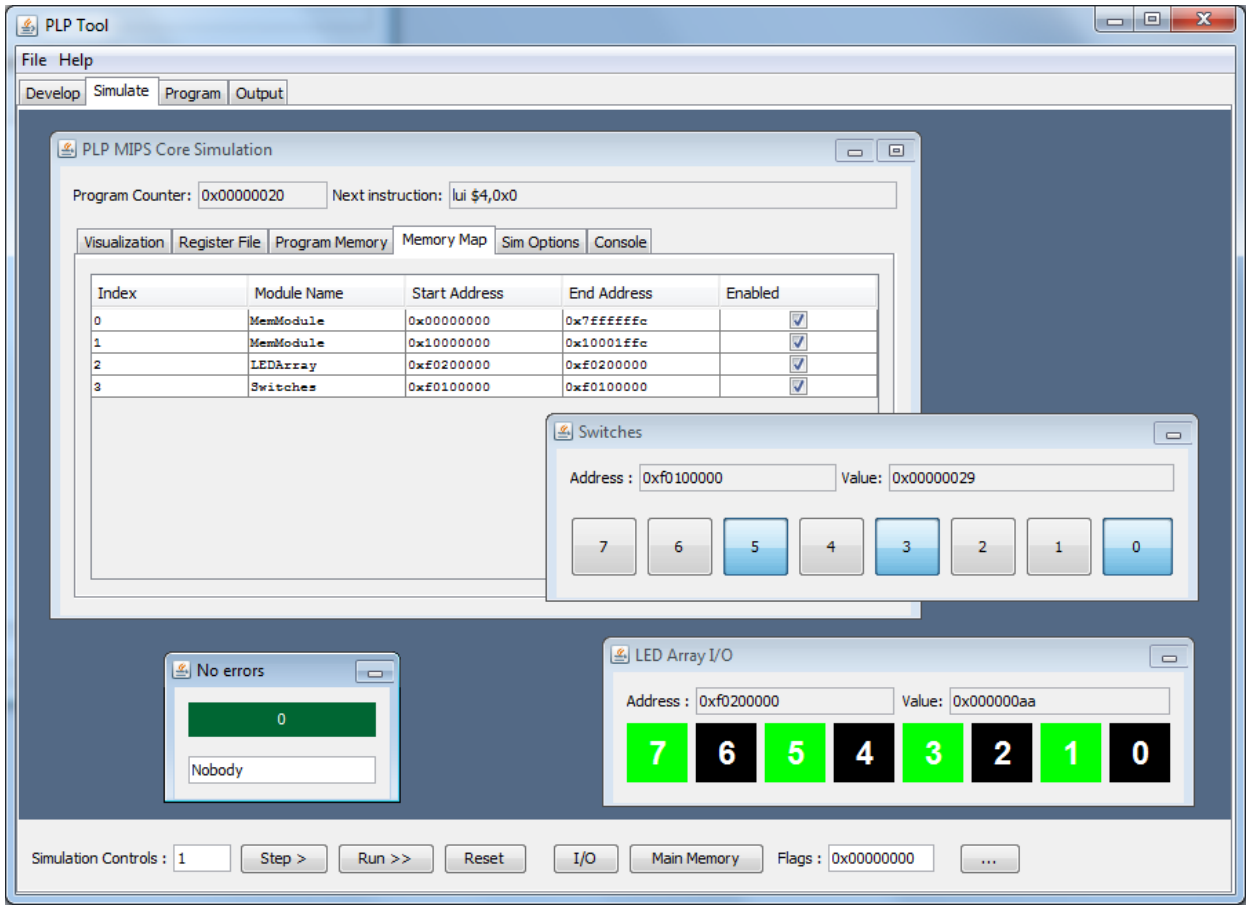

Figure 6 – Main PLP software window displaying a simulation with LED array and Switches I/O attached.

the pipeline stages, evaluation functions are called to evaluate the combinational logic in the circuit, and a new instruction is fetched by the fetch function.

The simulation core also includes a GUI window in the simulation pane to allow users to interact with the processor. Users can view and change values of the register file, modify program memory, and enable or disable forwarding events. An example of program disassembly listing is shown in figure 8.

The simulation core GUI also exposes a command console to the simulation backend for access to more features, such as printing out pipeline register values and code injection. Figure 9 shows the console displaying the values of the output side of the pipeline registers.

3. How we use PLP in the classroom

Many Computer Architecture laboratories are conducted with individual or small team assignments. While individual and small team laboratory assignments are effective in many contexts, it can be difficult for a small team to design and implement a complete processor in a single semester.

In our undergraduate Computer Architecture course, students are asked to design and implement functionally equivalent processor to that defined in the reference design. The course generally consists of roughly 30 students, who are divided into 5 teams, each of which is responsible for a subsection of the CPU design. The teams include a meta-team, which has administrative control over the other teams, and is responsible for the control logic, a front-end team, an execution engine team, a forwarding and hazards team, and a test and measurement team. The test and measurement team creates test fixtures for simulation and on-chip for each of the other teams, the individual pipeline stages, and the integrated CPU design.

The class as a whole completes a single design. An emphasis is placed on highly collaborative inter and

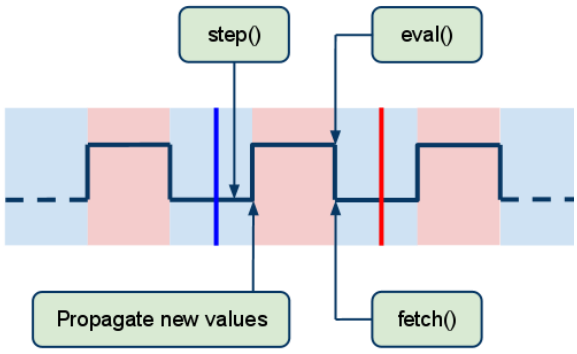

Figure 7 – Abstraction of the step function as implemented by the simulation core.

intra-team work. The meta-team has special administrative rights over the other teams in the class in that they are responsible for coordinating inter-team communication, signal and timing definitions, and other collaborative needs. By having students take charge over this part of the design process, the instructor is able to take on the role of the mediator.

The project consists of four phases: team-building, research, implementation, and integration. The team-building phase has students assign team roles, clarify their assign subsection in greater detail, and complete pre-implementation certifications (code management software and wiki training). The research phase allows students to learn course content relevant to their team, create and document block-diagrams, signal and timing definitions, and lecture to the class about their work. The research phase is the most collaborative part of the design process, as teams must work together to ensure that later integration will be successful. The implementation phase consists of teams implementing and testing designs made during the research phase. The integration phase begins with teams self-reassigning into an integration team, demonstration team, and communication team. The integration team carries out the final integration of the design, the demonstration team writes software to use on the PLP system during an end of semester "design day" hosted by the department, and the communication team finalizes documentation and creates a course video.

The focus of the course is on the project. Most of the course objectives set by the Computer Engineering faculty and ABET are satisfied by the successful completion of the course project. Since so much emphasis and time is dedicated to the project, other components such as exams and homework assignments have been removed. Weekly quizzes are still used as feedback to the instructor as to how students are progressing. Lectures generally follow the needs of students in regards to the project. Later in the semester when the project is well underway, lectures tend towards topics that aren't immediately relevant to the project such as performance evaluation and caches.

Since many solutions to this kind of design exist, including a near complete design in the course textbook, assessment is not done on the completion or quality of the design and implementation of the course

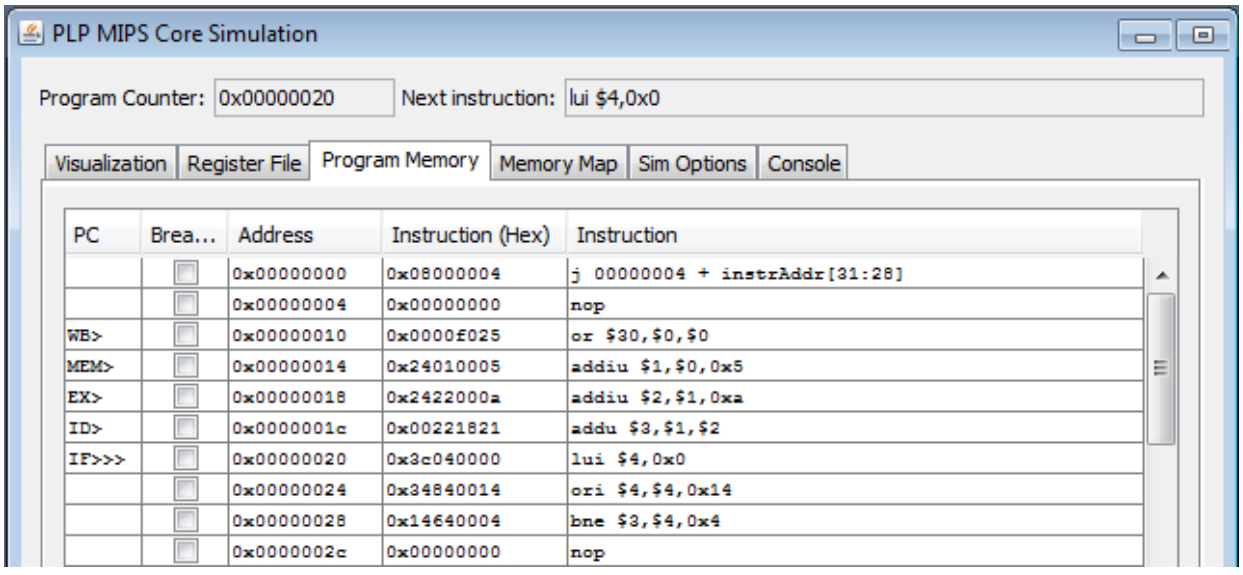

Figure 8 – Program listing view in the simulator. This view also displays which instruction is in each pipeline stage.

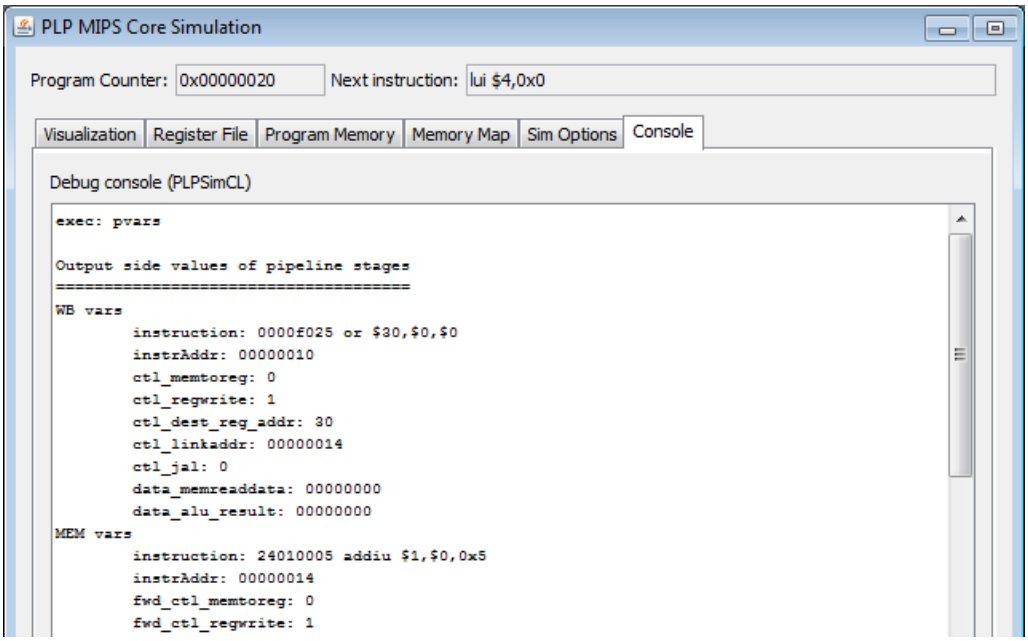

Figure 9 – Simulator console. This shows the output of the 'pvars' command, which prints out the signals on the output side of the pipeline registers.

project. Instead, assessment is made on the ability of students to communicate their understanding of their design, using communication as a proxy measure of student learning. Communication methods include documentation of the design and implementation process on a course wiki, lectures by teams in class, team-based oral examination at the end of each phase, and dissemination of results during the end of semester "design day" via the course video and other demonstration components.

4. Related Work

There exist several FPGA based SoC designs. Holland et. al [3] suggests a reduced MIPS instruction set design for use in the classroom. The design uses an 8 instruction variant of MIPS, and allows full observability and controllability through a host tool. This design however does not provide a host-independent product, requiring the host tool for running the processor in the classroom (beyond just programming). The PLP system provides a rich set of I/O, requiring the host tool for nothing more than initial programming. Additionally, the PLP MIPS design is more robust, while still simple enough for design and implementation in the classroom.

Nagaonkar and Manwaring [4] discuss a complete FPGA and micro controller based SoC for use in research and academia. Their design uses a very flexible custom hardware design. While its use in the classroom is mentioned, no explicit educational use is defined. The PLP system is designed only for use in the classroom, with special emphasis on exposing students to critical foundational components of the Computer Engineering curriculum.

Other than the FPGA hardware solutions mentioned above, processor simulators have been used in computer engineering classrooms in many universities as education tools. Simulators such as WebMIPS [5] and MipsIt [6] are used to teach concepts during program execution. Others such as MARS [7], SPIM [8] and TExaS [9] provide integrated development, simulation, and a debugging environment. The PLP system extends this to allow students to actually run their programs on real hardware, and to allow the students and faculty to modify the hardware / software as required.

5. Future Work

The current revision of the PLP system is designed for use in an undergraduate Computer Architecture classroom. The PLP development roadmap is in lockstep with course schedules for other Computer Engineering courses. The next revision of PLP, due Summer of 2011 will target an undergraduate Computer Based Systems course, and will focus on simulation visualization, greater features in the

assembler, and a debug framework in the reference hardware. Future revisions will include support for an embedded systems course, an introductory logic design course, and graduate Computer Architecture courses.

6. Obtaining PLP

The PLP system is licensed under the GNU GPL version 3 license, and is free to download and use. Media components, including lectures from the classroom, lecture slides, in-class assignments, and other documents are all licensed under a Creative Commons Attribution license.

The project is hosted at http://plp.okstate.edu

7. References

[1] R. E. Bryant and D. R. O'Hallaron, "Introducing computer systems from a programmer's perspective," presented at the Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, Charlotte, North Carolina, United States, 2001.

[2] W. Mulia and D. Fritz. (2011). *PLP Software Tool Javadoc Documentation*. Available: http://plp.okstate.edu/javadoc

[3] M. Holland, *et al.*, "Harnessing FPGAs for computer architecture education," in *Microelectronic Systems Education, 2003. Proceedings. 2003 IEEE International Conference on*, 2003, pp. 12-13.

[4] Y. Nagaonkar and M. L. Manwaring, "An FPGA-based Experiment Platform for Hardware-Software Codesign and Hardware Emulation," presented at the Proceedings of 2006 International Conference on Computer Design (CDES'06/ISBN #:1-60132-009-4/CSREA), Las Vegas, Nevada, 2006.

[5] I. Branovic, *et al.*, "WebMIPS: a new web-based MIPS simulation environment for computer architecture education," presented at the Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture, Munich, Germany, 2004.

[6] M. Brorsson, "MipsIt: a simulation and development environment using animation for computer architecture education," presented at the Proceedings of the 2002 workshop on Computer architecture education: Held in conjunction with the 29th International Symposium on Computer Architecture, Anchorage, Alaska, 2002.

[7] K. Vollmar and P. Sanderson, "MARS: an education-oriented MIPS assembly language simulator," *SIGCSE Bull.*, vol. 38, pp. 239-243, 2006.

[8] J. Larus. *SPIM: A MIPS32 Simulator*. Available: http://spimsimulator.sourceforge.net/

[9] J. W. Valvano, *Embedded Microcomputer Systems: Real Time Interfacing*: Brooks/Cole Publishing Co., 2000.