International Journal of Engineering Education Vol. 29, No. 1, pp. 215–229, 2013
Printed in Great Britain

0949-149X/91 $3.00+0.00

PLP: A Community Driven Open Source Platform for Computer Engineering Education*

WIRA D. MULIA, DAVID J. FRITZ, SOHUM A. SOHONI, KERRI KEARNEY and MWARUMBA MWAVITA
Oklahoma State University, Stillwater, Oklahoma, USA. E-mails: wira.mulia, david.fritz, kerri.kearney, mwarumba.mwavita@okstate.edu; sohum.sohoni@asu.edu

This paper is a detailed technical description of the Progressive Learning Platform (PLP). The PLP system is a System on a Chip design with accompanying tools reflecting a contemporary CPU architecture. The paper is intended to be a reference for users who are interested in using the platform, developers who intend to contribute to the project, educators who would like to adopt PLP in their computer engineering course, and engineering education researchers who would like to use PLP as a vehicle for conducting research. All hardware components of PLP are written in Verilog HDL, are open source, and are freely available. To support the hardware components, a unified assembler, cycle accurate emulator, and board interface software package is included. The software is written in Java, works on Linux, Windows, and Mac OS, is open source, and is freely available. The PLP hardware and software components are licensed under the General Public License version 3 to encourage open access and contribution. All parts of the system are publicly hosted and a public mailing list is used to serve as a communication channel between users and developers of the system.

Keywords: instruction set design; computer systems organization; collaborative learning; computer science education; modeling of computer architecture; learning technologies; educational simulations; user generated learning content

1. Introduction

The Progressive Learning Platform [1] is an open, community driven project that consists of a System-on-Chip platform and curriculum for computer engineering courses. System-on-Chip platforms and simulators already exist to aid the teaching of computer engineering topics in classrooms, but they are mostly standalone systems and do not provide the students with a cohesive framework that spans multiple courses. The aim for this project is to provide a comprehensive education platform and an open, dynamic development process by having a public website, a mailing list, and an open license of the work that allows unrestricted collaboration.

The curriculum that is based on the PLP project is a departure from the traditional lecture-style course to a social constructivist design [2]. Courses in this curriculum promote cooperative learning, communications skills, and other soft engineering skills in addition to technical skills. The PLP system centers on the design and implementation of a CPU. A CPU is something to which all computer engineering students can relate, which makes it an obvious choice for the focus of many computer engineering topics. This system utilizes Field-Programmable Gate Array (FPGA) development kits. FPGA is a reconfigurable integrated logic circuit that allows the hardware to be configured using a hardware description language (HDL), as opposed to the "final" nature of an Application-specific Integrated Circuit (ASIC) hardware. The adaptability of the system allows the use of the system in multiple courses. For example, a microcontroller course instructor can use the system as is to be utilized as a CPU development board with MIPS-like Instruction Set Architecture (ISA). An instructor for a graduate computer architecture course can use the system as a platform for the students to meet course objectives by actually making changes to the hardware itself or to the simulation and visualization environments.

The openness of PLP prevents the system from becoming obsolete. The design currently supports the Digilent Nexys2 [3] and Nexys3 [4] FPGA development kits, but can be developed to support other FPGA technologies. The hardware of PLP is written in behavioral Verilog [5] and the community is encouraged to adapt the system to as many hardware platforms as possible.

This paper describes in detail the design and implementation of version 3 of PLP and suggests ideas on how this system can be implemented in various computer engineering courses while maintaining a common context. This paper also serves as a reference for individuals or groups interested in developing any part of the PLP system. Section 3 includes a description of a reference hardware design including the core, bus interface, and numerous modules (VGA controller, timers, UART, GPIO, etc.). This section also describes PLPTool, a complete Java based software stack including an assembler, cycle accurate simulator, and programming tools.

* Accepted 6 July 2012.

PLP is also a platform for computer engineering education research. This paper provides some preliminary results based on a pilot study of how PLP enhanced students' academic efficacy in the computer architecture course. This course applied teaching philosophies that include social constructivism and cooperative learning [6] and is a departure from how the class was taught before: a traditional lecture class with disjointed lab assignments. PLP is also currently used in two other courses in Oklahoma State University: an embedded systems programming and an advanced computer architecture course.

The rest of the paper is organized as follows. Section 2 summarizes the work related to simulators and platforms for teaching computer engineering. Section 3 provides a detailed description of our platform. Section 4 provides a context of the courses in which the platform is used and describes how it is used in these courses. Section 5 describes the qualitative and quantitative studies carried out to gauge the effects of PLP on student learning, and Section 6 lists our conclusions and future work.

2. Related work

There exist several FPGA-based SoC designs. Holland *et al.* [7] suggest a reduced MIPS instruction set design for use in the classroom. The design uses an eight-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 to run 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 [8] discuss a complete FPGA and microcontroller-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 [9] and MipsIt [10] are used to teach concepts during program execution. Others such as MARS [11], SPIM [12] and TExaS [13] provide integrated development, simulation, and a debugging environment. LC-3 [14] is an ISA with an assembler and simulator built for it to be used in class projects. The PLP system extends the idea of using simulators as a teaching aid by allowing students to run their program on real hardware. The open nature of the hardware also allows instructors to suit the system to any specific requirement of a particular course.

Hades [15] is a logic circuit simulator with a powerful visualization tool and extensive components library. PLP focuses more on the architectural level with the inclusion of a toolchain for the ISA and integration into computer engineering courses.

PLP is an open source project, allowing anybody to contribute to and benefit from any part of the project with one restriction: all code contributions to the project will have to be licensed under GPL version 3. Brett [16] leveraged the open source paradigm to create an education environment where students can use and modify works that already exist to enhance their learning experience. The use of open source software also allows students to contribute improvements to the system itself. The openness of PLP uses the same idea to allow students to actually make modifications that will be incorporated into the project and provide a sense of ownership for the work that has been done.

3. Technical description

PLP development is hosted in our Google Code site [17]. The project uses Mercurial [18] as our source control management system. See the development site for how to obtain your own copy of the project.

3.1 CPU design

The PLP CPU implements a MIPS-like Instruction Set Architecture (ISA) with instructions listed in Table 1. The goal is to provide a simple, yet relevant, architecture that is applicable to a wide range of Computer Engineering courses. The differences between the PLP CPU and MIPS as described by Hennessy *et al.* [19] are the reduced instruction set, the lack of co-processors and there are no exceptions in PLP. The PLP CPU supports interrupts, but instead of utilizing a co-processor, two of the 32 registers in the CPU are used for an interrupt vector and to save the return address when an interrupt occurs. As in the pipelined MIPS micro-architecture, all hazards are forwarded except for the load/use hazard, which generates a single cycle stall. A block diagram is shown in Fig. 1. The datapath for the CPU is shown in Fig. 2.

3.1.1 Hardware implementation

All hardware is defined with Verilog HDL. Currently the platform targets the Digilent Nexys 2 and Nexys 3 FPGA development kits. Although a target

Table 1. PLP instruction set

Instruction	Syntax
Unsigned add	**addu $rd, $rs, $rt**
Unsigned subtract	**subu $rd, $rs, $rt**
Logical AND	**and $rd, $rs, $rt**
Logical OR	**or $rd, $rs, $rt**
Logical NOR	**nor $rd, $rs, $rt**
Multiply high-word	**mulhi $rd, $rs, $rt**
Multiply low-word	**mullo $rd, $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**

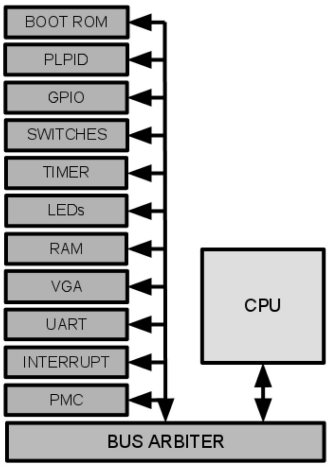

Fig. 1. PLP block diagram. All of the modules are connected to the CPU via the bus arbiter.

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.

3.1.2 Hardware modules

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.

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 was driven by the available hardware on the reference target development kit, as well as the perceived educational value of certain compo-

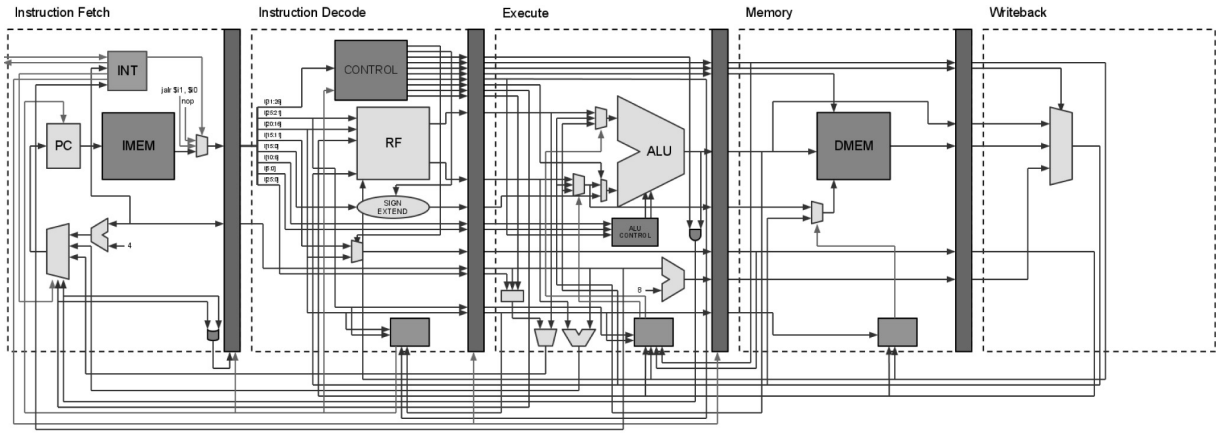

Fig. 2. Block diagram for the PLP CPU.

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

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

nents. 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 $640 \times 480 \times 8$ with frame buffer support, LED controller, GPIO, switch controller, and seven segment driver.

3.2 Extending the hardware

As shown in Fig. 1, the PLP CPU core connects to the rest of the system via a simple bus arbiter, which directs I/O to and from the CPU to the various modules based on a defined memory map. The arbiter is designed to be easily extended to accommodate new hardware modules that can be easily memory mapped to the PLP system. All modules, including memories, interrupt controllers, and I/O modules follow a standard module interface, as defined in Table 2. The bus arbiter facilitates both instruction and data access to each module. Even if a module does not support certain access (instruction reads for example), it must provide the necessary ports to the arbiter (returning undefined data if used incorrectly).

3.2.1 Detailed module port descriptions

rst—Synchronous reset
Global synchronous reset. The PLP CPU core also resets on this signal.

clk—Clock
Global clock. The PLP CPU core also uses this clock.

ie, de—Instruction / data bus enable
Bus enable signals. Modules see all traffic to every module on each cycle. These signals are raised by the arbiter to inform a specific module that the traffic in that cycle is intended for that module. For example, on an instruction that reads the value of the LEDs, the ie signal for the instruction memory would be raised, and the de signal for the LEDs module would be raised.

iaddr, daddr—Instruction / data address
Address ports. Represents the memory address to read from / write to for the instruction or data bus.

drw—Data read/write
2-bit read/write signal. 2'b00 indicates an idle cycle (neither read nor write), 2'b01 a write cycle, and 2'b10 a read cycle. 2'b11 is not a possible state. drw must be evaluated with de to determine if a read/write is intended for that module.

din—Data input
Data input to the module. This signal is only used on a store word instruction, and is only valid in a module when both de and drw[0] is asserted.

iout, dout—Instruction / data output
Instruction and data outputs. Used on instruction / data reads. Asserting values on these busses are ignored by the arbiter unless ie / de are asserted.

Table 2. Required module port declaration. Additional ports are declared after these ports

Port name	Direction	Size	Description
rst	input	1	Synchronous reset
clk	input	1	Clock
ie	input	1	Instruction bus enable
de	input	1	Data bus enable
iaddr	input	32	Instruction bus address
daddr	input	32	Data bus address
drw	input	2	Data read/write
din	input	32	Data bus data input
iout	output	32	Instruction bus output
dout	output	32	Data bus data output

3.2.2 Describing a new module

Developing a new module involves creating a new Verilog module that follows the port description shown in Table 2. An example skeleton module implementation is shown in Fig. 4. If the module communicates with other modules, or off-chip, additional ports are declared at the end of the port declaration list. Additional ports must then be routed up to the necessary level (to other modules in the arbiter for module-module communication, or to the top level module for off-chip communication).

For example we shall create a simple timer module that is memory mapped at 0xf0f00000, and contains a single writable register. The timer module will increment on each cycle, unless written to. To begin, we declare a new module named mod_timer using the required port list, as shown in Fig. 5.

3.3 Software tools and library

PLP has a suite of software tools (PLPTool) that includes an assembler, a simulator, and an interface to load a program image to the target board. These tools are written in Java to maximize the number of computing platforms they can run on without

```
module mod_timer(rst, clk, ie, de, iaddr, daddr, drw, din, iout, dout);
   input rst;
   input clk;
   input ie,de;
   input [31:0] iaddr, daddr;
   input [1:0] drw;
   input [31:0] din;
   output [31:0] iout, dout;

   wire [31:0] idata, ddata;
   assign iout = idata;
   assign dout = ddata;

   /* all data bus activity is
    * negative edge triggered
    */
   always @(negedge clk) begin
       if (drw[0] && de && !rst) begin
           /* write cycle */
       end else if (rst) begin
          /* idle / read cycle */
       end
   end
endmodule
```

Fig. 4. A skeleton module.

```
module mod_timer(rst, clk, ie, de, iaddr, daddr, drw, din, iout, dout);
   input rst;
   input clk;
   input ie,de;
   input [31:0] iaddr, daddr;
   input [1:0] drw;
   input [31:0] din;
   output [31:0] iout, dout;

   wire [31:0] idata, ddata;
   assign iout = idata;
   assign dout = ddata;

   reg [31:0] timer;
   assign ddata = timer;

   /* all data bus activity is
    * negative edge triggered
    */
   always @(negedge clk) begin
       if (drw[0] && de && !rst) begin
           /* write cycle */
           timer <= din;
       end else if (rst) begin
          /* idle / read cycle */
           timer <= timer + 1;
       end
   end
endmodule
```

Fig. 5. Timer module.

modification. The tools are written with extensibility in mind, and are organized such that developers familiar in Java can extend the tools in an object-oriented manner. PLP also includes a code library that users can use to interface with the I/O and to perform other routine functions.

3.3.1 PLPTool

The software tools can be used without any modification as a development tool for the PLP CPU. Students can download the package from the website and use it in a classroom to program an embedded system. Figure 6 shows a view of the editor with a PLP project open.

PLPTool uses the PLP project file format to store information of the project including the source files, compiled binary, and some simulation states. This makes it easier for students to pass their project file among team members and for course instructors to evaluate their work.

Clicking the simulation button will launch PLPTool into a simulation mode as shown in Fig. 7. This mode allows the users to simulate their program: stepping through instructions, running

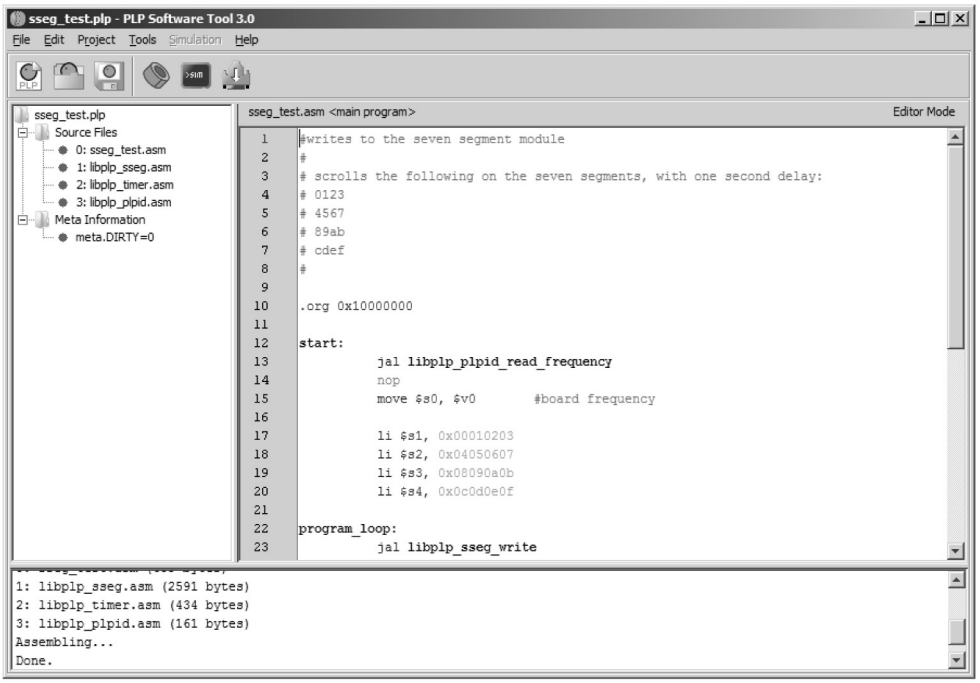

Fig. 6. The default view of the PLPTool editor window. The left pane displays the source assembly files contained in the project. The editor window should be familiar to users who have used a development environment for a programming language.

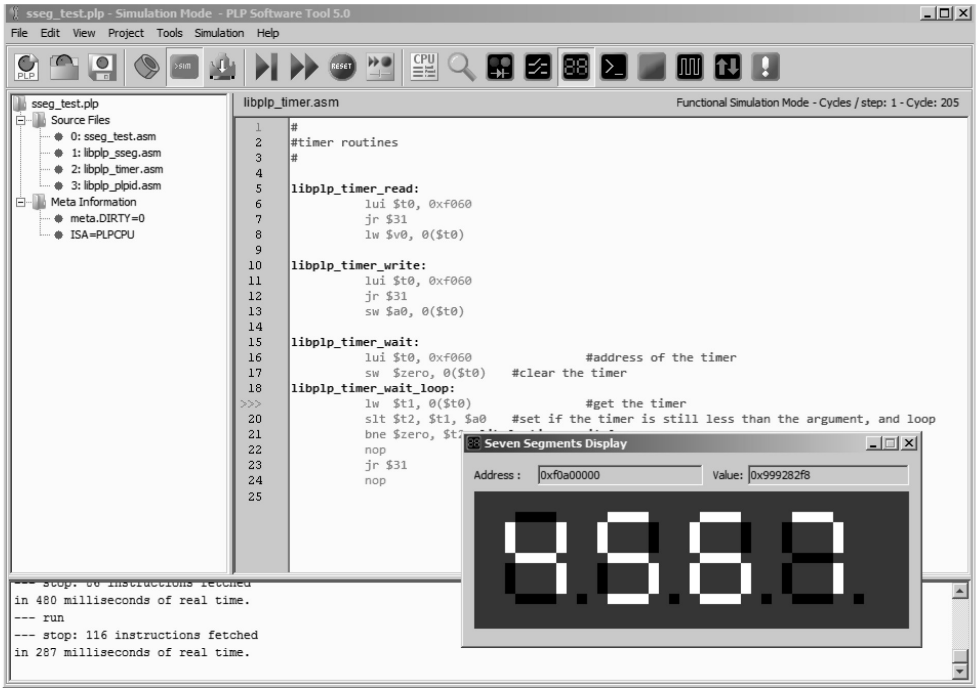

Fig. 7. PLPTool window running a simulation showing the seven segments I/O.

the CPU, modifying register and memory values, interacting with I/O devices, and observing the CPU operation. PLPTool simulates all the I/O devices implemented by the hardware. Figure 8 shows the watcher window of the simulator where users can directly observe the changes to CPU registers, memory locations, and I/O device registers.

PLPTool also allows users to download their program to the CPU board using a serial port. This capability uses RXTX serial library [20]

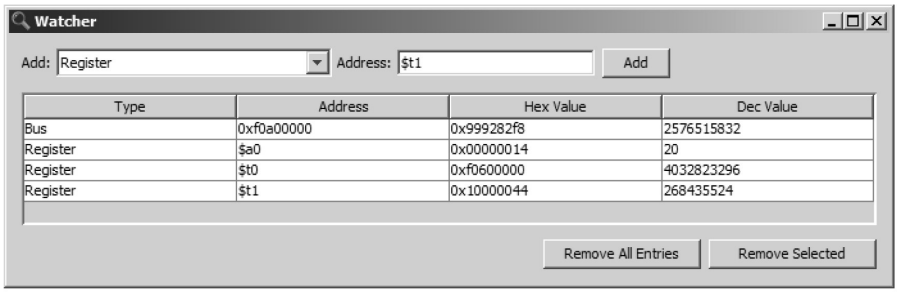

Fig. 8. Watcher window of the simulator. This window allows users to monitor memory addresses and CPU registers during simulation.

Table 3. Java packages in PLPTool and a description of what each package contains

Package name	Description
plptool	This package contains global classes such as constants and global configuration. It also contains abstract classes that define the ISA: the assembler, simulator, and programmer classes
plptool.gui	User interface classes
plptool.gui.frames	GUI window frames
plptool.mips	Assembler, simulator and board programmer implementation for PLP CPU
plptool.mods	PLPTool simulation modules

which may not be available in some computing platforms.

3.3.2 Developing and extending PLPTool

Table 3 lists the packages that are in PLPTool and gives a short description on what is contained within each one. PLPTool ISA framework is located in plptool as described in Section 3.3.3. plptool.mips is an example use of the framework and is currently used to implement an assembler, a simulator, and board programming tools for the PLP CPU. PLPTool is developed as a Netbeans 6.9.1 project [21].

3.3.3 ISA 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 9 provides a general overview of the system, and shows the structure of the framework. The plptool.mips package contains the current implementation of the PLP CPU ISA in PLPTool.

Developers will need to implement the functions listed in Fig. 9 to integrate with the PLP software stack. The 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 simulator. 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. Users can refer to javadoc-generated documentation [22] of the 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 (PLPSimCore.java). These methods include a program load procedure (loadProgram), a reset handler (reset), and a simulation step (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,

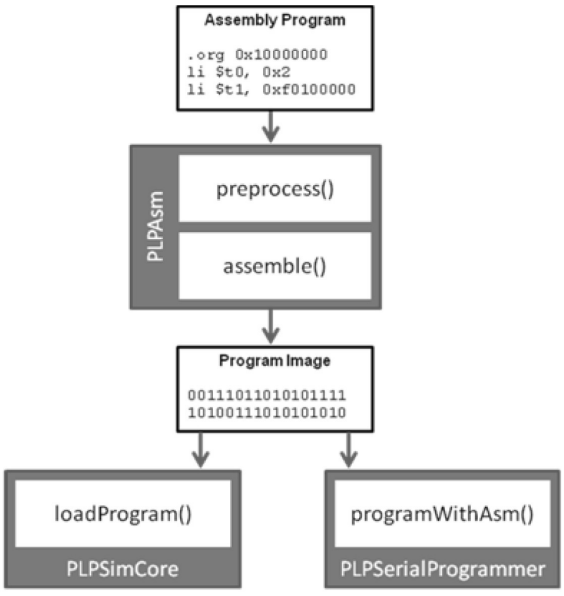

Fig. 9. The assembler, simulator, and board programmer abstract methods. Developers wishing to port an ISA to PLPTool will have to at least implement these methods and the methods mentioned in Section 3.3.3.

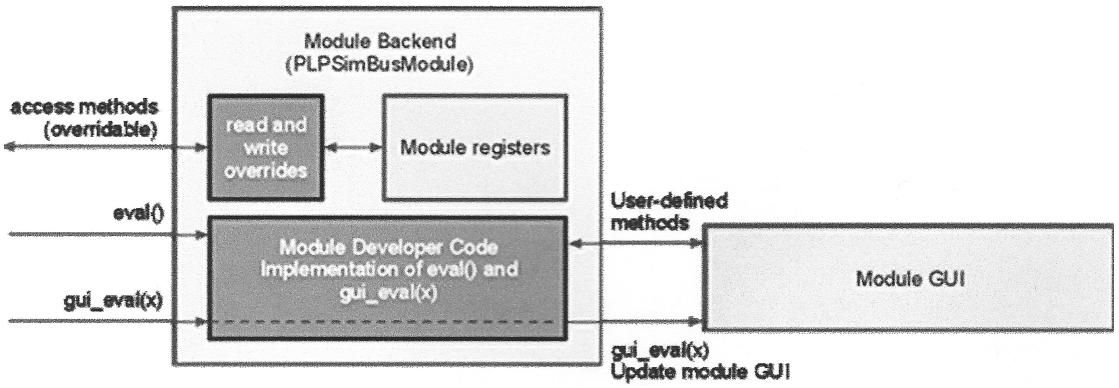

Fig. 10. PLP software module architecture, showing abstract methods and members of the class.

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 Fig. 10. The bus will call read or write functions of these modules whenever the CPU tries to access an address mapped to the module. 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 runtime.

3.3.4 Software library

PLP includes a software library that makes it easier to program by reusing common functions, such as accessing I/O devices and math operations. The library can be found under the reference/sw/libplp directory in the repository. The documentation for the library is available on the development website [17]. All the library functions use the argument registers ($a0 to $a3) for function parameters and they use the $v0 and $v1 registers for return values. Saved temporaries ($s0–$s7) and pointer registers are not modified. All other registers, including the temporaries ($t0–$t9) are subject to modification and may need to be saved before a library function is called.

4. Applying PLP to the classroom

This section describes the use of the PLP system in various courses at Oklahoma State University. PLP is currently used in three classes: Computer Based Systems (an engineering science class that introduces students to computer systems, microprocessors and assembly programming), Computer Architecture (a senior-level course that teaches processor design), and Digital Computer Design (a graduate-level computer architecture course that delves deeper into processor design). A pilot study was done for the Computer Architecture class in spring 2011. A detailed description of the course and the results of the study are presented later in this section. We first describe how PLP is used in each of the classes.

4.1 PLP in the Computer Based Systems course

PLP is used as a programming platform in Computer Based Systems where students use the software tools to write and simulate their programs. The students also use the board to see their program running in real hardware. We also modified the PLP system to drive a robot we name PLPBot [23] for the final project [24] where the students form groups to write the control software. PLPBot is an excellent example of the extensibility of the PLP system. PLPBot uses two PLP devices, one on the robot itself, and another device acting as a base station for communication relay to the driver. The robot PLP device was modified to support two additional UARTs, replacing the GPIO pins. One UART is used to interface a simple wireless transceiver, and another is used to drive the motor controller circuitry. The base station is also modified with an extra

UART, to drive another wireless transceiver. Since the PLP system already has a UART implementation, the only modifications needed were in the memory map and arbiter modules. In total, only ten lines of Verilog were added / modified to support this new configuration. Details of how the course was redesigned around PLP are available in another paper [25].

4.2 PLP in the Digital Computer Design course

Advanced computer architecture students can extend the system to reinforce their understanding of computer engineering concepts. Some project ideas include modifying the cache in the CPU design, encrypting the main memory, design and implement a superscalar version of PLP, or implement a MMU for the CPU. Students can also extend the software tools such as extending the simulator and writing optimization routines for the assembler. Instead of just developing toy projects, students are contributing to a real system and have the opportunity to see their additions to the project in action. They also have the opportunity to develop something that may be used, or extended, by future students.

4.3 PLP in the Computer Architecture course

While many engineering design laboratories are conducted in small teams, it is difficult and impractical for a small team to design and implement a complete processor in a single semester. In our undergraduate introductory Computer Architecture course, students are grouped into five large teams of five to seven students each. The teams are asked to work collaboratively on a single deliverable processor design to be used in the PLP system. The design is divided among the five teams including a front-end team, an execution engine team, a hazards and forwarding team, a test and measurement team, and a meta-team. Emphasis is placed on the highly collaborative inter- and intra-team work implied by the project assignment. To facilitate communication, the meta-team has special administrative rights over the other teams in that they are responsible for communicating inter-team communication, signal and timing definitions, and other collaborative needs. Moving the control of these components to a student team enables the instructor to further assume the role of facilitator and students to gain additional levels of real-world-applicable experience.

Each team consists of particular team roles including a team leader, documentation expert, and lead engineer. All team members must have at least one distinct role. The team leaders from each team meet regularly to ensure that proper communication of design efforts is made. The team leader for the meta-team is the primary liaison to the instructor and teaching assistants (TAs), and he or she is effectively the project leader. In our course, the meta-team leader's role as project leader is made explicit, and he or she serves as the final decision maker on conflicts in design decisions. Again, this provides a unique opportunity to extend student learning to address the interpersonal communication challenges of teaming, a critical real-world set of skills that is too often not addressed in engineering education.

The course emphasizes the course project and does not include any mid-term or final examinations. Weekly quizzes are used to monitor ongoing learning. Additionally, there are many in-class assignments, and teams provide weekly in-class oral status reports.

4.3.1 Course project

The course project, which lasts for the entire semester, is split into four phases: team-building, research, implementation, and integration. During the team-building phase in the first two weeks of the semester, students meet and exchange contact information, form basic team structure such as meeting times, and create a team behavioral contract. Students also complete required certifications for using the collaborative tools for the course, which includes a course Wiki and the code management software. Once students are certified, they are given administrative access to both tools, which allows them to modify all information, including that of other students, on the Wiki and code repository. All team-building phase information is documented on the course Wiki as the deliverable for that phase.

The research phases lasts for approximately one month and teams learn in great detail the aspects of their part of the overall design. It is during this phase that general instruction over computer architecture is provided in a lecture format. Teams are asked to learn about material relevant to their part of the design, create block diagrams, fully define signals that impact other teams, and document all of their work on the course Wiki. At the end of the research phase, teams deliver formal presentations of their findings. Other students, as well as an assessment board made up of the instructor, other knowledgeable instructors, and key graduate students, are also present for the presentation. The assessment board is responsible for assessing the team on the effectiveness and clarity of communication of their part of the design, as well as their understanding of the overall design. Other students are encouraged to ask questions as well, particularly about how that team's design impacts their own. The grade for the research phase is based on the combined perspectives of the assessment board.

The implementation phase is the longest phase of the design; within this phase, students implement their designs from the research phase in a hardware description language (in our course, using Verilog). Communication is also critical in this phase, as even moment-to-moment changes can have significant impact on the work of other teams. The meta-team is responsible for coordinating all cross-team information. All teams are responsible for providing up to the minute documentation on the course Wiki, which enables teams to use and build upon each others' work. The implementation phase ends with the test and measurement team evaluating the design based on the most recent version of the project specifications. This evaluation may result in teams being required to modify their designs. All implementations must meet specification in order to move on to the integration phase. Teams that have outstanding issues with their implementation do not, however, delay the overall project as the other teams may use components from the reference design to progress to the next phase.

The integration phase is the final phase of the project. In this phase, students are assembled into new teams: an integration team, a documentation team, a demonstration team, and a video team. The integration team has the task of integrating the implementations (or components from the reference design) into the final deliverable. The documentation team completes all Wiki based documentation of the design. The demonstration team uses the PLPTool to create a high-quality program to run on their design; this work will be demonstrated during the end-of-semester College of Engineering Design Day where students demonstrate their semester projects in the hallways of the Engineering College. Finally, the video team works to create a video-based documentary of the class project and student experiences with the class.

4.3.2 Assessment

Assessment practices in the course are based on the ability of students to communicate their understanding of the design and its implementation effectively. As noted above, this is accomplished through four major communication metrics:

1. Documentation of all work on a publicly accessible Wiki.
2. In-class demonstrations of the outcomes of each phase.
3. An end-of-term video detailing the course project.
4. An end-of-term, high-quality program for the College of Engineering Design Day.

The primary assessment metric is the course Wiki. A Wiki is a website that is driven by a powerful and simple markup language and is intended for rapid development of deeply connected content. The most prevalent example of a Wiki is Wikipedia, a free encyclopedia that anyone can contribute to or edit. Owing to the rapid development of content and ease of use, Wiki software is used in numerous contexts including project development portals, documentation efforts, and in education. Wiki software facilitates collaborative development, as anyone with access to the Wiki can edit it. This allows for information to develop in an evolutionary way from multiple users. Side discussions about the development of particular Wiki articles often develop as students work to resolve conflicts of information among users. Additionally, Wiki software saves revision history of every edit to an article, allowing users to revert a particular edit to any previous point in time. Wiki software is used extensively in the engineering industry and has merit in design courses intended to be representative of industry practice.

5. Educational research methodology to assess PLP usage in computer architecture course

This pilot study used a mixed method, case study design that, by definition, included both qualitative and quantitative research methods. This presented an epistemological challenge as each type of research flows from a very different worldview. For this study constructivism, which flows from interpretivism, dominated and closely associated with qualitative research design. Constructivists hold that "all knowledge, and therefore all meaningful reality as such, is contingent upon human practices, being constructed in and out of interaction between human beings and the world" [26]. Thus people may experience the same phenomena differently and thus have differing reactions, understandings, and conclusions with each person viewing his or her experience through the lenses of personal perspective and previous experience. This requires that the researcher use data collection techniques designed to capture variations in participant experiences and constructed meanings; in this study, free-flowing focus groups that encouraged extensive participant interaction and evolving perspectives were used. In addition the researcher kept field notes that included observations and emerging insights.

The combination of quantitative and qualitative methodologies can enhance and deepen understanding. Quantitative methodology flows from a positivist tradition that posits that meaning exists externally to the person and can be definitively discovered, understood, classified and measured.

This study used pre- and post-test assessment as a quantitative method for measuring changes in student knowledge.

5.1 Qualitative focus groups

The Progressive Learning Platform was used twice in the Computer Architecture (ECEN 4243) course. At the conclusion of these courses, a researcher from outside the college of engineering conducted a focus group with student volunteers who completed the course. Discussion focused on PLP and its perceived impact on the course experience. The focus group was designed to be free-form and interactive, allowing students to voice any opinions about their experiences in the course. However a set of guiding questions was used to provide direction for the discussion:

- How does your learning experience in the class compare with other learning experiences in your other engineering coursework?
- In what ways were you able to link learning from other classes to this one?
- How would you describe your interest level within and across the time of this class?
- Describe the level of collaboration that was required with your classmates.
- In what aspects did this class help you to prepare to work in engineering in the "real world?"

Sessions were audio-taped and transcribed. Open coding and researcher memos served as primary data analysis methods with particular attention paid to emerging patterns and themes. Peer review and researcher discussion, preservation of all data and notes, and analytic memos increased the trustworthiness [27] of the qualitative outcomes. Data were collected into four primary themes as reported below.

5.2 Pre–post assessment

We used pre/post tests data to evaluate the efficacy of PLP in the classroom [28]. The pre/post tests were identical to each other and administered at the beginning of the first lecture of the semester, as well as with the final examination (or last lecture in courses that had no final examination) at the end of the semester. The pre/post tests consisted of a number of questions designed to cover all listed course objectives and pre-identified "Be Able To's" (BATs).

A subset of test questions is explicitly covered by the PLP system. The success of the use of PLP in the classroom was determined by comparing pre-test and post-test scores. Specifically, we were interested in finding out if the use of the PLP system impacted students' knowledge of the relevant general course material and if the use of the PLP system in the classroom impacted knowledge of concepts explicitly covered by PLP. Tests were scored by calculating the mean, standard deviation, and standard error mean of the overall test and the subset of questions identified as being explicitly covered by PLP. The overall test calculations included the PLP questions. A paired dependent t-test design was used to analyze the results. The paired dependent t-tests uses repeated measures. Thus it is used to examine mean differences of test scores between two waves of assessments on the same participants. The t-test assesses whether the mean difference between paired/matched observations is significantly different from zero, that is, if the mean difference between the pre-test and post-test is greater or lesser than zero, as well as whether this difference is statistically significant or is noise (error). Furthermore, for each group of students, the Cronbach's alpha coefficient was calculated to determine the reliability of the instrument [29]. Reliability in this context was used to measure whether or not the items on the test measured the intended variable (topic). Generally, a Cronbach's alpha coefficient of > 0.5 is considered statistically significant; however, with this test it is important to note that a very high Cronbach's alpha coefficient is not necessarily a good thing. Too high a value may indicate that the test items do not vary enough to provide a meaningful test (equivalent to asking the same question with slightly different wording). Furthermore, there are a number of topics in the taught courses (especially ECEN 4243) that have independent topics (digital logic vs. high level programming) that, when measured, may not indicate a high Cronbach's alpha coefficient. Nonetheless, topics in these courses can be considered to be broadly linked, making it important to measure their correlations.

5.3 Findings

5.3.1 Qualitative case studies

Overall, students in the focus groups characterized their experience in the classes as "different," in a positive way, from previous engineering course experiences. The focus group members specifically noted four observable themes that they felt improved their understanding of Computer Architecture and Computer Engineering as a whole:

Building and bridging knowledge

Participants noted that the course concepts were serial and had a backwards dependency, including into previous courses in the Computer Engineering curriculum. Students had to understand the previous material (specifically in lab assignments) in order to proceed to the next. This dependency was not artificial (such as not being allowed as per the

syllabus to proceed until previous lab work was accomplished), but rather was a natural progression of learning the assigned material. As one student noted:

> In usual testing, I can take test 1 and test 2, and it doesn't matter on test 2 what grade I had on test 1. But in this, if my research wasn't correct, I must catch it up. [sic]

Use of PLP in the classroom also facilitated the bridging of the knowledge gained from previous (and listed pre-requisite) coursework, as well as from the current classwork to lab work:

> [It was] interesting to have one central hub that reinforced all of the materials. If you had it in class, the PLP system had something to do with it. So, as you moved on, you got more and more knowledge. [sic]

Another student noted that the course

> really bridged the gap between digital logic and coding, integrated hardware and software class we had prior to [this]. We actually had to know the knowledge from the pre-reqs. [sic]

A primary goal of PLP is to provide a mechanism to facilitate students' understanding of the connections among course content in the Computer Engineering curriculum. Student feedback appears to support PLP's ability to do this.

Increased levels of communication

Participants noted that the use of PLP required and facilitated an increased level of communication. Requirements for inter-team communication were reported as "much higher." Communication "with more than just your partner" was expected. Specifically, the use of the Wiki was a significant factor in facilitating communication:

> With this class we had Wiki dedicated to reference design and PLP so we could go to the Wiki. This is different because there was information readily available right there. That was good. [sic]

PLP explicitly uses communication as a resource throughout the curriculum as well as a primary assessment metric. This result was expected, and is in keeping with previous research regarding how critical communication is to understanding [30–35].

Increased student engagement

Participants generally agreed that students were more engaged with the course content, both because the format of the course facilitated engagement and because of student interest in the course, project, and PLP:

> We were working on a processor and I didn't really know a lot about it—the basics, yes, but actually going through it and building it from the ground up, learning about all of the different aspects of it. I'd say that interested me a lot. [sic]

One student reported that his increased interest caused him to expand his usual focus:

> [This class] got me interested—I normally only look at a system level as a computer engineer. This class got me interested in diving in deeper to how the chip works. [sic]

Another student noted that the format encouraged all students to be much more vocal, and another associated the format with having "fun":

> This was fun. I didn't worry about my grade this entire semester . . . it was more about learning and having fun and understanding it rather than "will I pass this class." [sic]

Clearly student engagement was reported as positively impacted by the use of the PLP system.

Perceived authenticity

The authenticity of the learning process, or the degree to which it mimicked the real world, was a prominent factor of the use of PLP on which numerous students commented. Reports collected into two major themes related to authenticity of task and authenticity of environment. One student summed up the overall impression:

> Being able to put your hands on it mattered. [sic]

At a greater depth, however, student perceptions about the authenticity of the environment varied across four areas:

1. *Management*. The instructor and teaching assistant were generally viewed as functioning in a role similar to managers in the real world; however, participants generally agreed that supervisory infrastructure needed to be increased. These observations evolved from focus group discussion about the need to address student complaints about unequal participation among team members. Additionally, students noted that critical issues raised in peer evaluations were not addressed by instructors in any observable way. The boss knows if there is an employee not pulling their weight—it's bad business to not pay attention.
2. *Team versus individual performance and assessment*. Students were concerned with the structure of and messages implied by the grading system. Because the emphasis was on teamwork, students could "game the system"—contributing little to the course project (and ostensibly not learning), and still receiving a passing grade. This was seen as in opposition to the real world where "if you don't show up for work in industry, you don't keep the job."
3. *Team Structure*. Participants reported that the use of PLP required more inter-team collaboration. In terms of helping them prepare for real-

Table 4. Descriptive statistics

Class / semester	Pre/post-test	Mean	*N*	σ	Std. error mean
ECEN 4243 Spring 2010	Pre (overall)	5.04	25	1.51	0.303
	Post (overall)	6.88	25	1.67	0.333
	Pre (PLP-specific)	1.72	25	1.14	0.227
	Post (PLP-specific)	3.4	25	1.44	0.289
ECEN 4243 Spring 2011	Pre (overall)	3.33	33	1.915	0.333
	Post (overall)	5.76	33	2.21	0.384
	Pre (PLP-specific)	1.50	33	1.37	0.239
	Post (PLP-specific)	3.51	33	1.52	0.265

world relationships, one student remarked "I would rank [the class] very high, one of the top."

4. *Accountability toward the goal/task*. Participants reported that deadlines in the class were more realistic—more "real world." Students also noted that the self-management required was more authentic or like their experiences in internships in the real world. If one group fails, then the whole class fails with it just like you would be in industry. [sic]

Another student remarked:

> We were able to see from the start of the class what the goal of this class was. This is very unusual. We started knowing what the end result should be if we did everything we were supposed to do. Classes do not usually work that way. [sic].

Students viewed the existence of a stated, specific goal as a significant positive as it "gives students something to work toward."

In addition to student comments about their experience in the classroom, focus group members provided feedback and suggestions for future iterations of the course and the use of PLP. Regarding teams, students suggested: smaller teams, even at the expense of having two, simultaneous course projects, in order to alleviate problems inherent to large teams.

Of significant importance is the last point mentioned, as teams report challenges with assessing individual performance and perceived authenticity. Students noted the desire for more emphasis on soft skills as well as the inclusion of assessment on technical achievement; they also stated a desire to have assessment better reflect industry practice of performance-based assessment. Qualitative results are promising, as students appear to embrace the merits of PLP in the classroom. Several deficiencies are noted, and plans to alleviate those concerns all are in place for the spring 2012 Computer Architecture course. A subsequent focus group will follow to assess the resolution of those concerns and additional needed improvements.

5.3.2 *Pre-post assessment*

Pre/post-test data were scored and analyzed using paired t-tests and Cronbach's alpha coefficient. Table 4 shows the mean, standard deviation, and standard error mean for each sample set. This data is used in the paired t-tests to determine if the difference in mean between the two tests is significant.

A subset of the questions on the pre/post-test is explicitly covered by PLP. To quantify this, we use Cronbach's alpha to determine the reliability of questions in the overall question set, and the questions in the PLP question set. In other words, is there an indication that the questions on the pre/post exam, overall and PLP, evaluate the same topic? As shown in Table 5, the overall and PLP-specific question sets are somewhat significantly reliable. The loss in reliability indicated by the Cronbach's alpha is probably due to the multiple variables examined in the pre/post-test. That is, the pre/post-test asks a number of questions about Computer Architecture, covering each of the course objectives, but the questions do not necessarily intersect as the breadth of course objectives is significant.

Table 6 shows the difference in mean, and t-test values from each of the studied courses. The t-test values for each of the courses indicate a significant difference, which implies that students in the course had a positive and significantly different understanding of the course objectives at the end of the course than at the beginning. This is expected of both a traditionally taught course and one that uses PLP. We also measure the subset of post-test questions that are explicitly and only covered by the use of PLP in the course. These too show a significant difference in mean.

The implications of these results allow us to assert that the use of PLP is an effective way to teach the stated course objectives. To determine whether or not the use of PLP is an effective way to teach the soft engineering skills, communication, and other goals of PLP, we defer to the qualitative results. We

Table 5. Reliability statistics

ECEN 4243 Spring 2011	Pre (overall)	0.687
	Post (overall)	0.685
	Pre (PLP-specific)	0.528
	Post (PLP-specific)	0.578

Table 6. Paired samples t-test

Class / semester	Pre/post-test	Mean difference	*s*	*t*	*df*	Sig. (2-tailed)
ECEN 4243 Spring 2010	Overall	1.84	1.49	6.17	24	0.000
	PLP-specific	1.68	1.57	5.33	24	0.000
ECEN 4243 Spring 2011	Overall	2.42	1.92	7.25	32	0.000
	PLP-specific	2.03	1.38	8.45	32	0.000

also note that the overall difference in mean for the 2011 ECEN 4243 course was greater than the difference in mean in the 2010 data.

6. Conclusion and future work

In summary, PLP is an open project that adapts to the needs of computer engineering education and it is still in an early stage of implementation. It was created to connect core concepts learned in various computer engineering courses, and is aimed at improving the learning experience for students. It is grounded in the theories of social constructionism and situated cognition. Results from the pilot study show that PLP is highly effective in engaging students and in helping them gain valuable skills. Since this is a work in progress, we have collected data in subsequent semesters and are currently transcribing and analyzing it. A longitudinal study is also in progress and will be completed by spring 2013. One clear advantage that we are beginning to see is that students, instructors and teaching assistants all found it very convenient to use the same system (PLP) in multiple courses.

One of the weaknesses of PLP in its current state is that it might prove to be a challenge for other instructors to readily adopt it in their courses. We are addressing this by improving the documentation, standardizing and rewriting parts of the code, and putting together a proposal for beta-testing PLP in three or four other universities. We hope to involve more people in the development of this project and have this system used in more classes and universities. Once PLP gains traction, we expect the open-source model to nurture and sustain its growth.

The PLP system is licensed under the GNU GPL version 3 license [36], 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 and the development site is hosted at http://code.google.com/p/progressive-learning-platform/.

Acknowledgment—This work was supported in part by NSF award EEC 1136934.

References

1. D. Fritz, W. Mulia and S. Sohoni, *The Progressive Learning Platform Website*. Accessed July 2012. http://plp.okstate.edu
2. L. S. Vygotskii, V. Davidov and R. Silverman, *Educational Psychology*, St. Lucie Press, 1997.
3. Digilent, *Digilent Nexys2 Spartan-3E FPGA Board*. Accessed July 2012. http://www.digilentinc.com/Products/Detail.cfm?Prod=NEXYS2
4. Digilent, *Digilent Nexys3 Spartan-6 FPGA Board*. Accessed July 2012. http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,897&Prod=NEXYS3
5. IEEE Standard Verilog Hardware Description Language, *IEEE Std 1364-2001,* pp. 0_1-856, 2001.
6. D. Fritz, W. Mulia, S. Sohoni, K. Kearney and M. Mwavita, The Progressive Learning Platform for computer engineering, presented at the *American Society for Engineering Education National Conference*, June 26, 2011.
7. M. Holland, J. Harris and S. Hauck, Harnessing FPGAs for computer architecture education, in *Microelectronic Systems Education, 2003. Proceedings. 2003 IEEE International Conference on*, 2003, pp. 12–13.
8. Y. Nagaonkar and M. L. Manwaring, An FPGA-based experiment platform for hardware-software codesign and hardware emulation, *Proceedings of 2006 International Conference on Computer Design*, Las Vegas, Nevada, 2006.
9. I. Branovic, R. Giorgi and E. Martinelli, WebMIPS: a new web-based MIPS simulation environment for computer architecture education, *Proceedings of the 2004 Workshop on Computer Architecture Education: held in conjunction with the 31st International Symposium on Computer Architecture*, Munich, Germany, 2004.
10. M. Brorsson, MipsIt: a simulation and development environment using animation for computer architecture education, *Proceedings of the 2002 Workshop on Computer Architecture Education: Held in conjunction with the 29th International Symposium on Computer Architecture*, Anchorage, Alaska, 2002.
11. K. Vollmar and P. Sanderson, MARS: an education-oriented MIPS assembly language simulator, *SIGCSE Bull.,* **38**, 2006, pp. 239–243.
12. J. Larus. *SPIM: A MIPS32 Simulator*. Accessed July 2012. http://spimsimulator.sourceforge.net
13. J. Valvano. *TExaS: Test Execute and Simulate*. Accessed July 2012. http://www.ece.utexas.edu/~valvano/sim.html
14. Y. N. Patt and S. Patel, *Introduction to Computing Systems: From Bits and Gates to C and Beyond*, McGraw-Hill Higher Education, New York, NY, 2003.
15. C. Bienia, S. Kumar, J. P. Singh and K. Li, The PARSEC benchmark suite: characterization and architectural implications, *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, Toronto, Ontario, Canada, 2008.
16. E. S. Brett, A Frankenstein approach to open source: The construction of a 3D game engine as meaningful educational process, *IEEE Transactions on Learning Technologies,* **3**, 2010, pp. 85–90.
17. D. Fritz and W. Mulia. *Progressive Learning Platform Development Website*. Accessed July 2012. http://code.google.com/p/progressive-learning-platform
18. *Mercurial Source Control Management*. Accessed July 2012. http://mercurial.selenic.com/about
19. J. Hennessy, N. Jouppi, S. Przybylski, C. Rowen, T. Gross, F. Baskett and J. Gill, MIPS: A microprocessor architecture,

Proceedings of the 15th annual workshop on Microprogramming, Palo Alto, California, USA, 1982.
20. *RXTX: The Prescription for Transmission*. Accessed July 2012. http://users.frii.com/jarvi/rxtx
21. *NetBeans IDE 6.9.1 Release Information*. Accessed July 2012. http://netbeans.org/community/releases/69
22. W. Mulia and D. Fritz. *PLP Software Tool Javadoc Documentation*. Accessed July 2012. http://plp.okstate.edu/javadoc
23. D. Fritz and W. Mulia. *PLPBot*. Accessed July 2012. http://code.google.com/p/plpbot
24. *ENSC 3213 Course Webpage*. Accessed July 2012. http://plp.okstate.edu/ensc3213
25. S. Sohoni, D. Fritz and W. Mulia, Transforming a microprocessors course through the progressive learning platform, *ASEE Midwest Conference*, Russelville, AR, 2011.
26. M. Crotty, *The Foundations of Social Research: Meaning and Perspective in the Research Process*, Sage Publications Ltd, 1998.
27. Y. S. Lincoln and E. G. Guba, *Naturalistic Inquiry,* **75**, Sage Publications, Inc., 1985.
28. L. R. Gay, G. E. Mills and P. W. Airasian, *Educational Research: Competencies for Analysis and Applications*, Pearson Merrill Prentice Hall, 2006.
29. L. J. Cronbach, Coefficient alpha and the internal structure of tests, *Psychometrika,* **16**, 1951, pp. 297–334.
30. E. F. Crawley, D. R. Brodeur and D. H. Soderholm, The education of future aeronautical engineers: conceiving, designing, implementing and operating, *Journal of Science Education and Technology,* **17**, 2008, pp. 138–151.
31. J. Lave and E. Wenger, *Situated learning: Legitimate Peripheral Participation*, Cambridge University Press, 1991.
32. M. Maram, P. Prabhakaran, S. Murthy and N. Domala, Sixteen Roles Performed by Software Engineers in First One Year, 2009, pp. 212–215.
33. D. Q. Nguyen, The essential skills and attributes of an engineer: a comparative study of academics, industry personnel and engineering students, *Global J. of Engng. Educ,* **2**, 1998, pp. 65–75.
34. L. B. Resnick, *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*, Lawrence Erlbaum, 1989.
35. J. Shimazoe and H. Aldrich, Group work can be gratifying: Understanding and overcoming resistance to cooperative learning, *College Teaching,* **58**, 2010, pp. 52–57.
36. F. S. Foundation. *GNU General Public License 3.0*. Accessed July 2012. http://www.gnu.org/licenses/gpl.html

Wira D. Mulia is a Doctoral Candidate in Electrical and Computer Engineering (ECE) at Oklahoma State University (OSU). He received his MS in ECE from OSU in 2009. His research interests are in Computer Architecture and Systems. He has been instrumental in the development of the Progressive Learning Platform, and has led the work on PLPTool.

David J. Fritz is a Doctoral Candidate in Electrical and Computer Engineering (ECE) at Oklahoma State University (OSU). He received his MS in ECE from OSU in 2008. His research interests are in Computer Engineering Education and Computer Architecture, with an emphasis on memory systems and virtualization. He is lead developer for the Progressive Learning Platform.

Sohum A. Sohoni is an Assistant Professor in Computing Studies in the College of Technology and Innovation at Arizona State University. Prior to joining ASU in 2012, he was an Assistant Professor at Oklahoma State University. He received his Ph.D. in Computer Engineering from the University of Cincinnati in 2004 and his Bachelors in Electrical Engineering from COEP, Pune University in 1998. His research interests are in STEM Education and Computer Architecture.

Kerri Kearney is an associate professor of educational leadership at Oklahoma State University. Her professional experience is a hybrid of both education and organizational consulting in leadership, organizational and team development, and executive coaching. She holds an MBA in management and an Ed.D. in educational leadership.

Mwarumba Mwavita is a Visiting Assistant Professor of Research, Evaluation, Measurement, and Statistics at Oklahoma State University.