Pipelined CPU Design With FPGA in Teaching Computer Architecture

Jong Hyuk Lee, Seung Eun Lee, Heon Chang Yu, and Taeweon Suh, *Member, IEEE*

***Abstract*—This paper presents a pipelined CPU design project with a field programmable gate array (FPGA) system in a computer architecture course. The class project is a five-stage pipelined 32-bit MIPS design with experiments on the Altera DE2 board. For proper scheduling, milestones were set every one or two weeks to help students complete the project on time. The goal of the project is to educate students effectively via hands-on learning, rather than having them achieve a complete and flawless CPU design. This study reveals that 21 MIPS instructions are enough to achieve the purpose. With the addition in 2010 of the properly enforced scheduling and the FPGA system, many more students successfully completed the class project than was the case in 2009. A student survey and the independent samples t-test reveal the effectiveness of the methodology with the FPGA system. This work differs from previous work in that the devised project requires the implementation of a real CPU instead of utilizing simulators or just experimenting with ready-made complete CPU models.**

***Index Terms*—Computer architecture, education, field programmable gate array (FPGA), hands-on learning, incremental learning, pipeline, problem-based learning (PBL).**

I. Introduction

COMPUTER architecture courses are among the most crucial of those offered in Computer Science and Computer Engineering departments, in that they lay the foundations for any further studies in the computer science area. Nevertheless, such courses typically cover theoretical topics in a pen-and-paper fashion and, at best, use architecture simulators to demonstrate the internal operations of computer organization. A recent study [1] used a field programmable gate array (FPGA)-based system to provide an opportunity for students to experiment with a real system, but this was limited to tracing and debugging by observing the pipeline register contents of the CPU. To educate students more practically and improve their motivation, it is imperative to devise a curriculum offering them hands-on experience based on the theoretical background.

As hardware complexity increases according to Moore's law, hardware description language (HDL) has become commonplace in digital logic design. HDL-based design offers dramatic time savings, allowing highly complex designs to be realized within a reasonable time. FPGA is another enabler, offering an environment where students can rapidly exercise and validate their design of interest. A study [2] reveals that a single Virtex-4 can accommodate an entire Pentium processor. The capacity of the latest FPGAs far exceeds that of the Virtex-4, supplying more than enough logic for the needs of the course. The two leading FPGA companies offer free versions of the computer-aided design (CAD) tools, adequate for course use. They also provide free educational boards to those universities qualified via their educational programs [3], [4].

The combination of HDL and FPGA yields an ideal environment for students to implement and exercise computer architecture in diverse and creative ways. The students acquire a deeper understanding of computer organization by considering and reasoning out the clock-by-clock operation of computers. The ability gained not only profoundly increases their confidence with hardware, but also directly influences their perspective on, and understanding of, software optimization and system software. It has also been observed that students focus more on the study of the architecture by visually examining the impact of their design on the FPGA board. Among the various elements of computer organization, this study focuses especially on the CPU core because this is a key constituent of computer architecture.

This paper presents a pipelined CPU design project using an off-the-shelf FPGA board in a computer architecture course. The class project is a five-stage pipelined 32-bit MIPS design. It requires students to implement the basic and essential instructions of the 32-bit MIPS. The final design should be able to execute a high-level program written in C. The goal of the project is not that students should design a flawless CPU (which is not necessary for the educational purposes), but that they should acquire the practical knowledge and skills needed for CPU design and use hands-on learning to understand the hardware and software interface. To help students schedule their projects and complete them on time, milestones were set every one or two weeks. Since they were using a commercial FPGA board, they were immediately able to download the designed CPU and visually examine its behavior.

The same project was assigned in both the 2009 and 2010 Fall semesters at Korea University, Seoul, Korea. The only difference between the two assignments was that the 2010 project adopted the FPGA board and guided students with milestones. None of the students finished the project in 2009, whereas roughly 40% successfully completed the project in 2010. A course survey and the independent samples t-test indicate the effectiveness of the implemented methodology. The course material is freely available online at http://esca.korea.ac.kr from Korea University.

Manuscript received May 17, 2011; revised September 30, 2011; accepted October 27, 2011. Date of publication November 30, 2011; date of current version July 31, 2012. This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2011-0003422).

J. H. Lee is with the Creative Informatics and Computing Institute, Korea University, Seoul 136-701, Korea.

S. E. Lee is with the Department of Electronic and Information Engineering, Seoul National University of Science and Technology, Seoul 139-743, Korea.

H. C. Yu and T. Suh are with the Department of Computer Science Education, Korea University, Seoul 136-701, Korea (e-mail: suhtw@korea.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TE.2011.2175227

TABLE I
MIPS INSTRUCTIONS IMPLEMENTED IN SINGLE-CYCLE CPU

Instruction category	Instructions implemented in single-cycle MIPS
Data processing	`add, addu, addi, addiu, sub, subu, and, or, ori, slt, lui`
Memory access	`lw, sw`
Branch	`beq, j`

The organization of this paper is as follows. Section II summarizes the related work and highlights the uniqueness of this work. Section III discusses the lab environment including the hardware model, software, and the FPGA board. Then, the main contribution, a pipelined CPU design with the FPGA board, is presented in Section IV. Section V introduces the curriculum of the computer architecture course. Section VI reports the course statistics, student survey, and feedback, and conclusions are drawn in Section VII.

II. RELATED WORK

There is a large body of literature [5]–[9] on teaching computer architecture courses, with most of these studies adopting architecture simulators. Djordjevic *et al.* [6] created an educational environment known as `CAL2`, where students can exercise assembly programming, monitor program execution, and inspect implementation details at the register transfer level. Nikolic *et al.* [7] surveyed and evaluated simulators suitable for teaching a computer architecture course. More recently, Oztekin *et al.* [8] introduced yet another architecture simulator called `BZK.SAU`. With the rise in popularity and convenience of FPGAs, several papers adopt the technology in developing an architecture course. Gustin *et al.* [9] developed a somewhat peculiar processor referred to as `Move` and proposed implementing a CPU downloadable to an FPGA. Bulic *et al.* [1] also introduced an FPGA-based environment, based on their own `HIP` processor, which implemented a five-stage pipelined processor. Based on the capability of tracing and debugging the CPU core, students could observe pipeline registers on the fly.

To improve teaching, universities have adopted various pedagogical methods such as problem-based learning (PBL) [10], [11], experiential learning [12], [13], and incremental learning [13], [14]. Linge *et al.* [11] reports their experience of using PBL for teaching computer network design. Ozturk *et al.* [12] describe experiential learning with simulation for a project-oriented multicore education. Schaumont [14] introduced a senior-level course using an incremental design approach for a hardware and software codesign. The adoption of incremental learning has been driven by the following benefits [14], [15]. First, it is useful for students whose upfront experience is limited, compared to that of professional engineers. Second, a course using incremental learning is likely to be perceived as easier than one in which many weeks of background introduction are required before students can attempt practical exercises. Third, it avoids the fault line between prior knowledge and new knowledge that students can experience in the middle of courses with longer timescales.

This paper differs from previous work in several ways. First, the project described here requires the actual implementation of the 32-bit MIPS rather than using simulators or just experimenting with ready-made complete CPU models. With hands-on learning, students are able to build their practical knowledge of, and skills in, computer operation and design. Second, students can use an open-source off-the-shelf MIPS compiler since the project targets a commercially successful MIPS. Third, students can instantly examine the operation of the designed CPU visually with their own software thanks to the system model and the FPGA system. Fourth, the number of assigned instructions was minimized to lessen the burden of implementation while maximizing the educational efficiency. This study reveals that roughly 21 instructions are required in MIPS. Fifth, the project combines three pedagogical methods: PBL, hands-on learning, and the incremental approach.

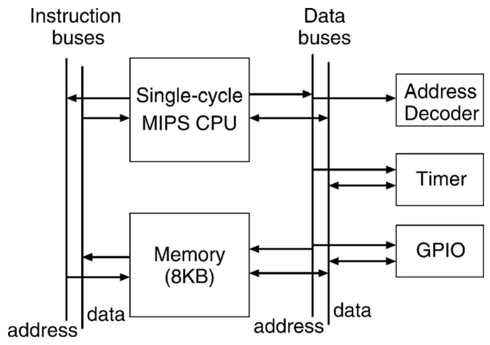

Fig. 1. Single-cycle MIPS based system model in Verilog-HDL.

III. LAB ENVIRONMENT

A. Hardware Model of the MIPS-Based System

For experiments on the DE2 board, a simple model of a computer system was devised as shown in Fig. 1. The model, composed of five components [Single-cycle MIPS CPU, Memory, Address Decoder, Timer, and General Purpose Input/Output (GPIO)], is provided to students as a base system for them to play with. Using the system model, they can turn on and off seven-segment displays and LEDs and take input from switches and push-buttons on the DE2 board by writing simple MIPS assembly and/or C code.

The 32-bit MIPS was chosen as a target CPU for the project as per a renowned architecture textbook [16]. The single-cycle MIPS design comes with the textbook [17], but only implements a subset of the MIPS instructions, as listed in Table I. As a result, it is hardly able to execute even a simple C program since the compiled binary would be likely to contain unimplemented instructions. The single-cycle MIPS executes each instruction in one clock cycle without pipelining. Therefore, the critical path is much longer than the pipelined CPU, limiting the operating clock frequency. The main task of the project is to add new instructions and convert the single-cycle CPU to the pipelined version.

A memory is used to store instructions and data and is configured to be 8 kB in size. Since the system model should be

TABLE II
MEMORY MAP OF THE BASE COMPUTER SYSTEM

Hardware components	Size	Base Address	End Address
Reserved		0xFFFF_3000	0xFFFF_FFFF
GPIO	4KB	0xFFFF_2000	0xFFFF_2FFF
Reserved	4KB	0xFFFF_1000	0xFFFF_1FFF
Timer	4KB	0xFFFF_0000	0xFFFF_0FFF
Reserved		0x0000_2000	0xFFFE_FFFF
Memory	8KB	0x0000_0000	0x0000_1FFF

downloaded to an FPGA on the DE2 board, the memory is created with Megafunction in Quartus-II instead of using a generic Verilog model. The Megafunction creates the user-configured memory with memory elements inside the FPGA. A dual-port memory, one port for accessing instructions and the other for accessing data, was created to allow for two simultaneous accesses from the CPU, avoiding the structural hazard. The memory is preloaded with the compiled MIPS binary and merged into a bitstream, downloadable to the FPGA.

Other than the single-cycle MIPS and memory, the components were designed from scratch using Verilog-HDL. The timer is an essential component in virtually any digital system; its major role is to periodically notify the system of an event, so that the system can properly manage resources and tasks. For example, operating systems use the timer interrupt extensively for scheduling tasks. The timer in Fig. 1 has a internal counter and can generate an interrupt periodically according to the counter value set by a user. GPIO is a module for interacting with the external world. By programming the GPIO module, students can take input from switches and push-buttons on the DE2 board, display numbers on seven-segment displays, and turn on/off LEDs.

Table II shows the memory map of the system model, which has a 4-GB memory space since the implemented MIPS has a 32-bit address bus. Each peripheral device occupies a 4-kB space, even though the whole 4 kB is not completely consumed by its internal registers. The address decoder in Fig. 1 assigns a memory space to each peripheral device in the system. Since the source code is based on Verilog-HDL, students can easily change the memory map and perform experiments with assembly and/or C-programming if desired.

B. *Software Environment*

Based on the GNU bin utility [18], the MIPS cross-compiler was built under Cygwin [19]. The Eclipse Integrated Development Environment (IDE) [20] is used for the software development. The Eclipse provides a convenient environment for programming, so it is widely used in both academia and industry. Once software is written using assembly and/or C-language, the MIPS cross-compiler generates a binary file in the Executable and Linkable Format (ELF). Since the Megafunction-generated memory requires a special file format—Memory Initialization File (MIF)—for initialization, a Perl script was written to convert the ELF binary file to the MIF format; this was provided to the students. As the name implies, the MIF is used to initialize the memory in Altera FPGA, preloading a software program into the memory in Fig. 1. To automate the compilation to the MIF conversion process, the make utility [21] is used with Cygwin under Eclipse.

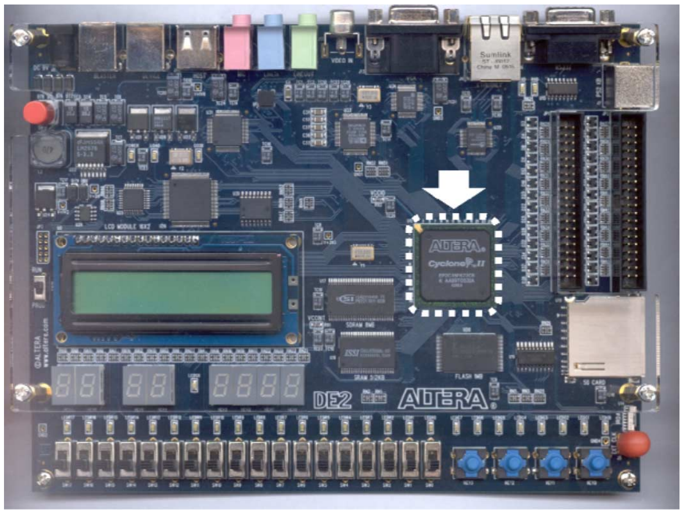
Fig. 2. Altera DE2 board.

C. *Experiment Environment*

Fig. 2 shows the Altera DE2 board. The main component, marked with a dotted line, is a Cyclone-II FPGA, to which a software program downloads the hardware system model. The board features various input and output components such as switches, push-buttons, seven-segment displays, a serial port, LEDs, and USB connector. Those I/O devices can be controlled by designing digital logic inside the FPGA. In the system model, the GPIO module provides interfaces to the I/O devices.

Altera provides a CAD tool, Quartus-II [22], which is used to synthesize, place and route, and download the logic design to the Altera FPGAs. As mentioned, the MIF is merged with the hardware design, and the generated bitstream is downloaded to the Cyclone-II from a PC via the USB connection. The free version of the Quartus-II is adequate for these experiments. For simulating the CPU design for validation, the ModelSim Altera Starter Edition [23] is used throughout the course.

IV. PIPELINED 32-b MIPS DESIGN

The class project is a five-stage pipelined MIPS design. The objective of the project is to have students build their practical knowledge of, and skills in, CPU design and pipelining via hands-on learning, rather than to have them design a flawless and complete CPU (which is a daunting task). Thus, it is not necessary for them to implement every single instruction that the MIPS ISA supports. From the numerous instructions available, essential basic instructions for software programming were chosen for implementation, such as data processing, memory access, and branch instructions. Considering that the computer architecture course is offered for second-year

TABLE III
MILESTONES FOR FIVE-STAGE PIPELINED MIPS IMPLEMENTATION

Milestones	Purposes	Newly added instructions	Milestone duration
#1	Gaining familiarity with experimental environment	None	1 week
#2	Manual schematic drawing of single-cycle MIPS	None	1 week
#3	Addition of new MIPS instructions to single-cycle MIPS	`sltu, bne, jal, jr, nop`	2 weeks
#4	Pipeline implementation with data hazard detection and handling	None	2 weeks
#5	Pipeline implementation with control hazard detection and handling	None	1 week
#6 (One-on-one Interview)	Further implementation of both single-cycle and pipelined MIPSs	`slti`	2 weeks

students at Korea University, the single-cycle MIPS design was provided to students as a base CPU design. As already mentioned, the single-cycle MIPS provides and implements only a subset of MIPS instructions, as listed in Table I, and executes each instruction in one clock cycle without pipelining. Six milestones, listed in Table III and described in what follows, were established at one- or two-week intervals to help students with scheduling and completing their pipelined MIPS design project by the end of the semester.

- Milestone 1 familiarizes students with the experimental environment. As explained in Section III, the system model is composed of a single-cycle MIPS, a memory, a timer, and a GPIO. This milestone requires each student to display their own student ID on seven-segment displays with the MIPS program under the Eclipse environment. To display numbers on a seven-segment display, students must understand the memory map of the system model and GPIO registers and program the appropriate numbers to the GPIO ports. One tricky point is that they can only use the instructions provided by the single-cycle MIPS listed in Table I. Even a simple C code could well generate a binary file with unimplemented instructions in the single-cycle MIPS if it were not very carefully programmed. Thus, this constraint forces students to study the binary file generated by the cross-compiler and consider the relationship between hardware and software.
- Milestone 2 requires students to draw schematic diagrams of the system model and the single-cycle MIPS design. The drawing of the system model written in Verilog-HDL lets students explicitly view the "big picture" of the hardware system: the memory map and connections/interactions between a CPU and peripheral devices. The drawing of the single-cycle MIPS prepares students for adding new instructions and implementing the pipeline in the subsequent milestones. By observing and drawing the CPU implementation, students can explicitly assimilate the data path and control path of the single-cycle MIPS. They will use this schematic in the following milestones as a reference guide when adding new instructions and implementing the pipelining.
- Milestone 3 provides two C programs written to display hexadecimal numbers (h'5 and h'A) alternatively on a seven-segment display at a certain time interval (about 1 s). Table IV lists one of the C programs. The cross-compilation generates a binary file containing several instructions that are not implemented in the single-cycle MIPS. Those new instructions are listed in Table III. Note that `move`, `li`, and `bnez` in Table IV are pseudo-instructions and are equivalent to `addu`, `addiu`, and `bne`, respectively. To display the numbers on the DE2 board, the instructions should be added to the single-cycle MIPS, which requires the modification of the decoder logic and data/control paths. `jal` (jump and link) and `jr` (jump register) instructions, in particular, are used for the subroutine call and return in MIPS. By implementing new instructions and performing simulations with ModelSim, students become thoroughly familiar with the internal organization of a CPU. Rather than assigning the pipeline implementation directly, it is worth first spending time on the single-cycle CPU; this dramatically raises students' confidence in their understanding of a CPU and helps motivate them to complete the remaining milestones successfully.
- Milestone 4 assigns the five-stage pipeline implementation from the single-cycle MIPS. It provides an assembly program that displays the numbers 1–8 on seven-segment displays if the pipelined version has been correctly implemented. To reduce the complexity of the implementation, this milestone is confined to splitting the data and control paths to the five-stage pipeline and resolving the data hazard. The assembly code contains neither any branch instructions that incur the control hazard nor any new instructions. However, the code is contrived to intensively generate data dependency. The pipelining can be implemented simply by adding pipeline registers in appropriate places, according to the students' discretion and under the instructor's guidance. The implemented MIPS, an in-order CPU, incurs only read-after-write (RAW) dependency. The data hazard is resolved by adding forwarding logic and a hardware interlock. Fig. 3 shows the outcome if the pipelining has been correctly implemented with the data hazard handling.
- Milestone 5 assigns the implementation of control hazard detection and handling. It also provides a MIPS assembly code that intensively generates the control hazard. The delayed branch is not assigned for implementation, to alleviate the workload. Since the MIPS adopts the delayed

TABLE IV
C-CODE AND ITS CORRESPONDING MIPS MACHINE CODE FOR MILESTONE 3

```
#include "SevenSeg.h"

void display (int);

int SevenSeg()
{
unsigned int i;

while (1){

    display(SEG_5);
    for (i=0; i<0xFFFFF; i++) ;

    display(SEG_A);
    for (i=0; i<0xFFFFF; i++) ;
}

return 0;
}


void display (int num)
{
    unsigned int * seg0_addr = (unsigned int *)
SevenSeg0;

    *seg0_addr = num;

    return;
}
```

```
int SevenSeg()
{
  10:   27bdffe0        addiu   sp,sp,-32
  14:   afbf001c        sw      ra,28(sp)
  18:   afbe0018        sw      s8,24(sp)
  1c:   03a0f021        move    s8,sp
unsigned int i;

while (1){

    display(SEG_5);
  20:   24040012        li      a0,18
  24:   0c000028        jal     a0 <display>
  28:   00000000        nop
    for (i=0; i<0xFFFF; i++) ;
  2c:   afc00010        sw      zero,16(s8)
  30:   08000011        j       44
<SevenSeg+0x34>
  34:   00000000        nop
  38:   8fc20010        lw      v0,16(s8)
  3c:   24420001        addiu   v0,v0,1
  40:   afc20010        sw      v0,16(s8)
  44:   8fc30010        lw      v1,16(s8)
  48:   3c02000f        lui     v0,0xf
  4c:   3442ffff        ori     v0,v0,0xffff
  50:   0062102b        sltu    v0,v1,v0
  54:   1440fff8        bnez    v0,38
<SevenSeg+0x28>
  58:   00000000        nop

    display(SEG_A);
  5c:   24040008        li      a0,8
  60:   0c000028        jal     a0 <display>
  64:   00000000        nop
```

`move, li,` and `bnez` are pseudo instructions:
- `move s8, sp == addu $s8, $sp, $0`
- `li a0, 18 == addiu $a0, $0, 18`
- `bnez v0, 38 == bne $v0, $0, -32`

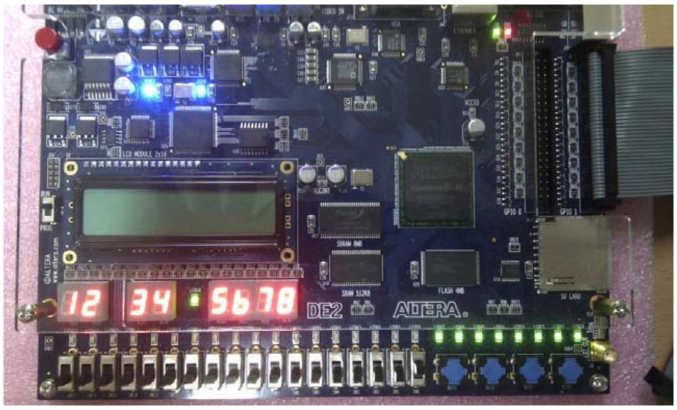
Fig. 3. Outcome of milestones 3 and 4.

branch, the assembly code is manipulated to fill the delay slots with `nops`. Given that the main goal of the course is for students to acquire practical knowledge and skills in the CPU architecture and design within a single semester, it is better to avoid consideration of the more advanced techniques such as delayed branch, which could potentially take too much time. The assembly code provided displays the same numbers as Milestone 4 if the control hazard is resolved correctly.

- Milestone 6, the final milestone, is a one-on-one interview that checks students' comprehension of computer architecture by discussing their experiences in meeting the milestones. Before the interview, students are given an assignment that requires them to further study the relationship between architecture and hardware/software. The assignment provides a C program that displays numbers from 0 to 9 on a seven-segment display at a certain time interval. The C code uses subroutines and pointers. Students are asked to observe the disassembled binary, especially focusing on how the pointer statement in C is translated to the MIPS machine code. The compiled binary file contains a new instruction (`slti`), which should be implemented in the pipelined version of the MIPS.

V. Computer Architecture Curriculum

Table V summarizes the curriculum devised for the computer architecture course. The prerequisite courses to take this course are C-programming and Computer Logic Design. The Computer Logic Design course is offered in the Spring semester before the Fall semester offering of the computer architecture course and covers digital logic design and Verilog-HDL. Thus, students are prepared to design any digital logic, including CPU, if guided properly.

As shown in Table V, the 16 weeks of lectures include MIPS instructions, single-cycle MIPS design, pipelined MIPS design, caches, TLB, and I/Os. By week 6, students have become familiar with the MIPS instructions through ISA lectures and assembly programming assignments. The milestones begin after the lecture on the single-cycle MIPS design at week 7 and finish

TABLE V
CURRICULUM FOR COMPUTER ARCHITECTURE COURSE

Weeks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Lectures	Course Intro	ISA & Performance	MIPS Instructions (data processing, memory access, branch, miscellaneous)				Single-cycle MIPS design			Pipelined MIPS design		Caches		TLB	I/O	Final Week
Milestones							#1	#2	#3		#4		#5	#6		

before the final week. Each milestone corresponds to the topics being covered in that week's lectures, so students can self-review each topic and put it into practice by performing the milestones. One week each is allocated for the milestones 1, 2, and 5, and two weeks each are allocated for the more complex and difficult milestones 3, 4, and 6. Milestone 3 requires adding new instructions to the single-cycle MIPS. Milestone 4 requires implementing pipelining and data hazard detection/handling. Both lead to major changes in the hardware design with Verilog-HDL.

VI. COURSE SURVEY AND EVALUATION

The course was offered in Fall semesters of 2009 and 2010, with the same project being assigned in both semesters. The only difference was that in 2009 students only used the ModelSim simulation as a validation method, and in 2010 the FPGA system and the milestones were adopted. The students were given nine weeks to complete the project, with the final goal being the same pipelined MIPS design able to execute the compiled binary of the given C code. Since the FPGA system was not adopted in 2009, those students only used ModelSim hardware simulation to debug and validate their CPU implementation. Thus, in 2009, the system model was composed of two components: a MIPS CPU and a memory.

Table VI shows the course statistics. Of 22 students enrolled in 2009, only five finished the implementation of the basic in-order MIPS with pipelining and data/control hazard detection and handling logic. None of the students met the final goal of executing the given C code on their design. In 2010, more than half of the students successfully implemented the basic in-order MIPS, and 12 students met the final goal. This dramatic difference is due to two factors: the enforced schedule management via milestones and the adoption of the FPGA board for visual validation and for increasing student engagement. According to the one-on-one interviews held in 2009, students tended to defer starting the project until the last minute; many underestimated the project workload and thus became frustrated a few days before the deadline. In 2010, the milestones guided students with their scheduling by distributing the workload evenly over two months. The adoption of the FPGA board and the system model also contributed to the success in 2010. The visual monitoring of the hardware and software interaction on the real system, rather than just using the ModelSim simulation for the design validation, gives students a real motivation to complete the milestones. It also lets students see the "big picture" of a computer system (not only the CPU) by observing the hardware implementation of peripheral devices and the programming to access them.

To prove the effectiveness of the methodology implemented in 2010, the independent samples t-test [24] was performed with SPSS [25]. It is assumed that the two groups (2009 and 2010)

TABLE VI
COURSE STATISTICS

Year	2009	2010
Project Goal	5-stage pipelined 32-bit MIPS CPU design	
Class size	22 students	29 students
Student management	None	Incremental methodology with Milestones
DE2 board (FPGA)	Not Used	Used
# Students implemented pipelining & hazard handling	5 students	18 students
# Students finished successfully	0 students	12 students

are independent of each other. Then, the following hypotheses were formed to prove.

- Null hypothesis: The means of the two groups are not significantly different.
- Alternative hypothesis: The means of the two groups are significantly different.

Table VII shows the descriptive statistics for the two groups. The mean of the 2010 group is roughly 20 points higher than that of the 2009 group. The standard deviation indicates that there is a stark contrast between successful and unsuccessful students both in 2009 and 2010. It is observed that the students who complete Milestone 3 tend to complete the remaining milestones successfully, and the other students tend to fail. However, with the FPGA system and the properly enforced scheduling in 2010, the distribution of the scores was moved in a positive direction by roughly 20 points. Table VIII shows the results of the independent samples t-test. The significance of Levene's Test for Equality of Variances reports 0.111, which is greater than 0.05. It indicates that the variances of the two groups are approximately equal. Then, the significance of the t-test for Equality of Means is 0.0325 (i.e., 0.065/2), which is less than 0.05, which proves that there is a significant difference between the 2009 and 2010 groups. In other words, students in 2010 achieved significantly higher scores than the ones in 2009, confirming the effectiveness of the implemented methodology.

Table IX shows the survey result taken in 2010. For each survey statement, students selected one of five choices: Strongly Disagree (1 point), Disagree (2 points), Neutral (3 points), Agree (4 points), and Strongly Agree (5 points). As shown, students agreed that the course is well organized (4.32). Students also agreed that it is necessary to design a CPU when studying computer architecture (4.27), even though it takes time and effort during the semester. They also think that it is effective to use the FPGA board in studying architecture for the visual verification (4.14). After taking the course, the vast majority of students believed that they understood CPU

TABLE VII
DESCRIPTIVE STATISTICS FOR THE TWO GROUPS

Year	# Students	Mean	Std. Deviation	Std. Error Mean
2009	22	41.5909	32.94969	7.02490
2010	29	60.5517	37.28901	6.92440

TABLE VIII
RESULTS OF INDEPENDENT SAMPLES t-TEST

	Levene's Test for Equality of Variances		t-test for Equality of Means						
	F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
								Lower	Upper
Equal variances assumed	2.635	.111	1.889	49	0.065	18.96082	10.03538	-1.20603	39.12766
Equal variances not assumed			1.922	47.793	0.061	18.96082	9.86390	-0.87409	39.79572

TABLE IX
COURSE SURVEY RESULTS (2010)

Statements	Average	Standard Deviation
The course was well organized	4.32	0.72
It is necessary to design CPU when studying computer architecture	4.27	0.77
I was surprised to know the capability of FPGAs	3.95	0.78
It was effective to use the FPGA board in learning computer architecture	4.14	0.77
I understand CPU and computer systems much better than before	4.32	0.72
I recommend this course to other students	4.18	0.85

Strongly Disagree (1 point), Disagree (2 points), Neutral (3 points), Agree (4 points), and Strongly Agree (5 points)

and computer systems much better than before (4.32). They agreed that they would recommend the course to other students (4.18). In the anonymous survey, only three students mentioned that the CPU design was too much of workload for a single semester. A majority of students said that the course is by far the most informative and practical, and they were satisfied with the course materials and organization. This implies that the hands-on learning with the FPGA system and the milestones has a tremendous impact on the practical understanding of computer architecture. The experiments with the FPGA board using the system model not only enhance students' understanding of computer systems, but also increase their motivation beyond that found when relying on computer simulations.

Judging from the previously published literature, most architecture courses use architecture simulators or ready-made complete CPU models with FPGA at best. Considering that an architecture course lays the foundations for further studies in the computer science area, it is imperative for students to improve their practical knowledge and experience of computer architecture. Nevertheless, CPU design is sometimes thought to involve an overwhelming amount of work; this paper addresses an important pedagogical issue in this regard. By minimizing the number of instructions while maximizing the educational efficiency, the devised project makes it feasible for students to complete a pipelined CPU design within roughly two months, thus enabling hands-on active learning via actual implementation. The project in 2010 also combines the three pedagogical methods of PBL, hands-on learning, and the incremental approach: The project assigns the CPU design problem (PBL); this is performed via hands-on design and practice (hands-on learning), and the milestones evenly distributed the workload over two months and properly guided students with their scheduling (incremental approach). Even though PBL is considered to be an effective pedagogical method in some areas of engineering education [11], the 2009 project (the PBL-only approach) implies that the appropriate pedagogical method should be selected at the instructor's discretion according to the workload of the assigned problem.

Several lessons learned from applying this CPU design project were the following. First, the students' tendency to defer their work should be tackled appropriately (as in 2010, with the milestones) when assigning the project. Second, many students are better motivated by experimenting with the actual system rather than only using simulators. Last but not least, the project devised in 2010 was proved to be more effective than that in 2009. Nevertheless, a number of students were still not successful in performing the project. Many students responded in the survey that they were sometimes puzzled with observing and debugging the clock-by-clock operation of the designed pipelined CPU with the ModelSim simulation. Special biweekly debugging sessions would be one option to overcome this difficulty and improve the success rate.

VII. Conclusion

This paper presents a pipelined CPU design project with an FPGA system for use in a the computer architecture course. The final goal is to design a five-stage pipelined MIPS, resolving data and control hazards. Well-defined milestones guided students to complete the project within the scheduled timeline. The adoption of the FPGA board and the system model provides motivation and fun with visual monitoring of hardware and software interaction. Furthermore, it helps students examine and digest not only the CPU, but also the whole computer system.

Designing a flawless CPU could be a daunting task if all the instructions were to be implemented since the different combinations of instructions can lead to aberrant states in conjunction with asynchronous events such as interrupt. When devising the project, one of the main considerations was to minimize the number of assigned instructions in order to lessen the students' workload. The other constraint when designing the project was to maximize the educational efficiency with those instructions. This study reveals that 21 MIPS instructions are adequate to achieve this purpose.

The methodology implemented in 2010 facilitated a dramatic increase in the number of students successfully completing the project. The majority of students responded in the course survey that the course is by far the most informative and practical they had taken. The independent samples t-test confirms the effectiveness of the implemented methodology. The course material is freely available online at http://esca.korea.ac.kr from Korea University.

References

[1] P. Bulic, V. Gustin, D. Sonc, and A. Strancar, "An FPGA-based integrated environment for computer architecture," *Comput. Appl. Eng. Educ.*, 2010, to be published.

[2] S.-L. L. Lu, P. Yiannacouras, T. Suh, R. Kassa, and M. Konow, "A desktop computer with a reconfigurable Pentium," *Trans. Reconfig. Technol. Syst.*, vol. 1, pp. 1–15, 2008.

[3] "Xilinx university program," Xilinx, San Jose, CA, 2011 [Online]. Available: http://www.xilinx.com/university/

[4] "Altera university program—Learning through innovation," Altera Corporation, San Jose, CA, 2011 [Online]. Available: http://www.altera.com/education/univ/unv-index.html

[5] A. Clements, "The undergraduate curriculum in computer architecture," *IEEE Micro*, vol. 20, no. 3, pp. 13–21, May–Jun. 2000.

[6] J. Djordjevic, B. Nikolic, T. Borozan, and A. Milenkovie, "CAL2: Computer aided learning in computer architecture laboratory," *Comput. Appl. Eng. Educ.*, vol. 16, pp. 172–188, 2008.

[7] B. Nikolic, Z. Radivojevic, J. Djordjevic, and V. Milutinovic, "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization," *IEEE Trans. Educ.*, vol. 52, no. 4, pp. 449–458, Nov. 2009.

[8] H. Oztekin, F. Temurtas, and A. Gulbag, "BZK.SAU: Implementing a hardware and software-based computer architecture simulator for educational purpose," in *Proc. 2nd Int. Conf. Comput. Design Appl.*, 2010, pp. 490–497.

[9] V. Gustin and P. Bulic, "Learning computer architecture concepts with the FPGA-based "Move" microprocessor," *Comput. Appl. Eng. Educ.*, vol. 14, pp. 135–141, 2006.

[10] M. Abdulwahed and W. Balid, "An assessment rich PBL vs classical teaching approach: A case of an embedded systems course," in *Proc. 2nd Int. Res. Symp. PBL*, 2009, pp. 1–10.

[11] N. Linge and D. Parsons, "Problem-based learning as an effective tool for teaching computer network design," *IEEE Trans. Educ.*, vol. 49, no. 1, pp. 5–10, Feb. 2006.

[12] O. Ozturk, "Multicore education through simulation," *IEEE Trans. Educ.*, vol. 54, no. 2, pp. 203–209, May 2011.

[13] H. Lim, H. Yu, and T. Suh, "Using virtual platform in embedded system education," *Comput. Appl. Eng. Educ.*, 2009, DOI: 10.1002/cae.20401.

[14] P. Schaumont, "A senior-level course in hardware-software codesign," *IEEE Trans. Educ.*, vol. 51, no. 3, pp. 306–311, Aug. 2008.

[15] H. Osborne, "The postroom computer: Teaching introductory undergraduate computer architecture," in *Proc. 33rd SIGCSE Tech. Symp. Comput. Sci. Educ.*, 2002, pp. 157–161.

[16] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design*. San Mateo, CA: Morgan Kaufmann, 2009.

[17] D. Harris and S. Harris, *Digital Design and Computer Architecture*. San Mateo, CA: Morgan Kaufmann, 2007.

[18] "GCC, the GNU compiler collection," GCC, 2011 [Online]. Available: http://gcc.gnu.org/

[19] "Cygwin," Red Hat, Raleigh, NC, 2011 [Online]. Available: http://www.cygwin.com/

[20] "Eclipse," Eclipse Foundation, Inc., Ottawa, ON, Canada, 2011 [Online]. Available: http://www.eclipse.org/

[21] "GNU Make," Free Software Foundation, Boston, MA, 2011 [Online]. Available: http://www.gnu.org/software/make/

[22] "Quartus-II," Altera Corporation, San Jose, CA, 2011 [Online]. Available: http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html

[23] "ModelSim–Altera starter edition," Altera Corporation, San Jose, CA, 2011 [Online]. Available: http://www.altera.com/products/software/quartus-ii/modelsim/qts-modelsim-index.html

[24] D. Freedman, R. Pisani, and R. Purves, *Statistics*, 4th ed. New York: Norton, 2007.

[25] "Statistical package for the social sciences," IBM, Armonk, NY, 2011 [Online]. Available: http://www-01.ibm.com/software/analytics/spss/

Jong Hyuk Lee received the M.S. and Ph.D. degrees in computer science education from Korea University, Seoul, Korea, in 2006 and 2011, respectively.

He is currently a Research Professor with the Creative Informatics and Computing Institute, Korea University. His current research interests include grid computing, cloud computing, distributed system, parallel architecture, embedded system, and computer science education.

Seung Eun Lee received the Ph.D. degree in electrical and computer engineering from the University of California, Irvine, in 2008.

He is currently an Assistant Professor with the Department of Electronic and Information Engineering, Seoul National University of Science and Technology, Seoul, Korea. Prior to joining Seoul Tech in 2010, he spent two years as a Platform Architect with Intel Labs, Hillsboro, OR, and six years as a Researcher with Korea Electronics Technology Institute, Seongnam, Korea. His current research interests include computer architecture, multiprocessor system-on-chip, low-power and resilient VLSI, and hardware acceleration for emerging applications.

Heon Chang Yu received the B.S., M.S., and Ph.D. degrees in computer science and engineering from Korea University, Seoul, Korea, in 1989, 1991, and 1994, respectively.

He has been a Professor of computer science education with Korea University since 1998. From February 2004 to January 2005, he was a Visiting Professor of computer science with the Georgia Institute of Technology, Atlanta. Since 2011, he has been the Executive Director of Korean Institute of Information Technology, Daejeon, Korea. His research interests include cloud computing, grid computing, distributed computing, and fault-tolerant systems.

Prof. Yu was the Vice President of the Korean Association of Computer Education and an Editor of *Korean Institute of Information Scientists and Engineers*. He was awarded the Okawa Foundation Research Grant of Japan in 2008.

Taeweon Suh (M'08) received the B.S. degree in electrical engineering from Korea University, Seoul, Korea, in 1993, the M.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1995, and the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, in 2006.

He is currently an Associate Professor with the Department of Computer Science Education, Korea University. Prior to joining academia, he was a Systems Engineer with Intel Corporation, Hillsboro, OR. His research interests include embedded systems, computer architecture, parallel computer architecture and programming, many-core, and computer science education.