COMPUTER ARCHITECTURE DESIGN AND ANALYSIS INVOLVING RECONFIGURABLE HARDWARE PLATFORM

Muhammad Zafrul Hasan
Engineering Technology and Industrial Distribution Department
Texas A & M University
hasan@entc.tamu.edu

Abstract

Computer Architecture and Organization deals with both software and hardware aspects of computer systems. This is important both for a programmer and a system designer. Due to their wide spread penetration in all fields, it is almost obligatory for students in Electrical, Computer, and Telecommunication engineering programs to master the basics of these two areas. Although a typical microprocessor programming course covers the architecture part of it, organization of systems is not widely addressed. Memory interfacing and expansion techniques, cache memory organization, memory management unit for virtual memory organization, alternative ways of processor design, fast arithmetic circuits are some topics covered under organization. The purpose, concept, and design of these system elements can be covered in a lecture class whereas their hardware implementation can be supported in a laboratory environment. The laboratory exercises would certainly enhance experiential learning of the students. However, choosing a suitable platform to accommodate the laboratory exercises is challenging as it needs to satisfy peculiar needs of different types of designs. Field Programmable Gate Arrays (FPGAs) provide a flexible hardware platform to accommodate digital systems. FPGAs, such as the ones provided by Xilinx, are quite useful in applications requiring hardware changes to accommodate system behavior. As such, these devices offer the opportunity to implement different computer system components conveniently in hardware using VHDL (Very high speed integrated circuit Hardware Description Language). FPGAs can be easily reconfigured to evaluate alternative design approaches often encountered in computer systems. With such implementation data, more complex models can be formulated and simulated to predict and evaluate system performance. Thus, such a reconfigurable platform also enables architecture and organization research. This paper presents an outline of a course covering concepts and implementation of computer system elements, associated laboratory exercises involving reconfigurable logic, and course related research with simulation results.

1. Introduction

Motivation and rationale: In order to enhance students' learning in engineering programs, it is important to provide them with engaging laboratory and continuous assessment of learning outcomes [1, 2]. Also, providing examples and teaching subject matter through student-centered approaches ensure effective student learning [3]. These approaches promote activities valued by industry that encourage active student participation in the learning process [4, 5]. Moreover, it is also important for the students to be exposed to the open-ended nature of design problems [6]. These facts emphasize strong cohesion between the materials covered in a lecture class and its

associated laboratory activities [7]. In addition to this, students need to appreciate the practice of design trade-offs among several competing requirements [8].

Limitation of traditional courses: Normally a course covering computer architecture and organization uses built hardware as the platform that has little configurability for laboratory exercises [9, 10]. Hardware is programmed in either a high level language or processor specific assembly language. This bounds the students to only explore some architectural features by programming. Contrary to that, if the students could modify the hardware to implement a new feature, the exploration would be more comprehensive by programming several alternative approaches.

Way to overcome: Thus, a reconfigurable hardware (such as FPGA) extends the horizon of experiments with architectural and organizational features of embedded computers. Usually, FPGAs are populated in a board containing several interfacing components such as displays, switches, push-buttons, and standard ports. As such, these boards have comparable features found in traditional processor based hardware used for such laboratory exercises [9]. FPGAs can be configured to implement different designs in hardware using either schematics or any hardware description language. The implemented hardware in FPGA can be programmed by code chosen by the designer. Thus, this design process becomes very intuitive for the students as if they own it. This process also empowers them with the concept of hardware / software co-design and integration, an area that is in high demand in the industry. Thus a reconfigurable hardware platform becomes a preferred means for meeting the educational goals discussed above.

Benefit of incorporating research elements: Although undergraduate students may not be matured enough for research, a flavor of such could be introduced to them [11]. Performance and power consumption of a design, trade-offs among various metrics, and the issues of reliability and upgradability could be analyzed for a design implementation. These activities are expected to stimulate critical thinking in the students that would be beneficial in the capstone design project in their senior year as well as in the profession.

With the above points in view, this paper outlines both the lecture and laboratory contents of such a course, its evaluation strategy, course related research that involves students, justification and incorporation of the course in the curriculum. The paper ends with a conclusion suggesting future directions of the course.

2. Lecture Contents

The set of learning outcomes for the course under discussion are as follows:

1) To learn different computer system elements,
2) To understand design alternatives for these elements and their merits,
3) To use VHDL to implement such elements in reconfigurable logic,
4) To appreciate the process of hardware / software co-design, and
5) To expose research activity associated with the course contents

Lecture contents can be developed as described in the following paragraphs.

Hardwired control & Micro-sequencer control: Fundamental to any computer architecture and organization course is Central Processing Unit (CPU) design. This involves design of register or data-path section, ALU section, and the control unit. However, in designing the control unit one has two choices. A hardwired control unit that is normally employed in a Reduced Instruction Set Computer (RISC) can be implemented. Alternatively, a micro-sequencer based control unit that is the norm in a Complex Instruction Set Computer (RISC) can be implemented. For a hardwired control unit, counter-decoder approach is to be followed for state transitions as shown in Figure 1, although other valid approaches for state machine implementation would also be fine [12].

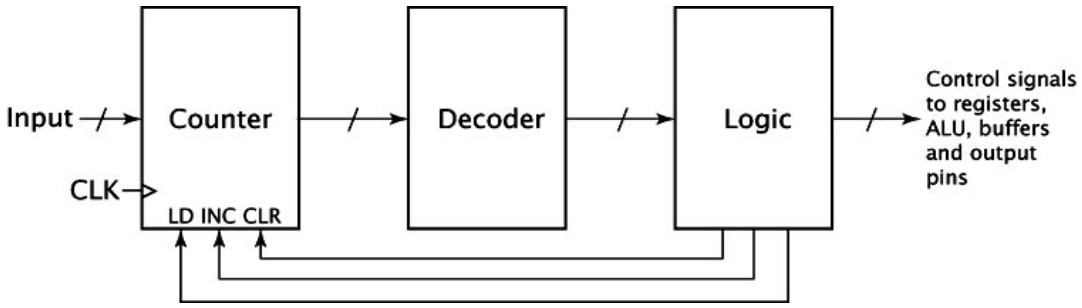

Figure 1: Generic Hardwired Control Unit

On the other hand, micro-sequencer based control unit can be implemented using multiplexer-register-control ROM approach as shown in Figure 2. In addition, the control ROM can be populated using any one of horizontal, vertical, or direct decoding techniques of microinstructions.

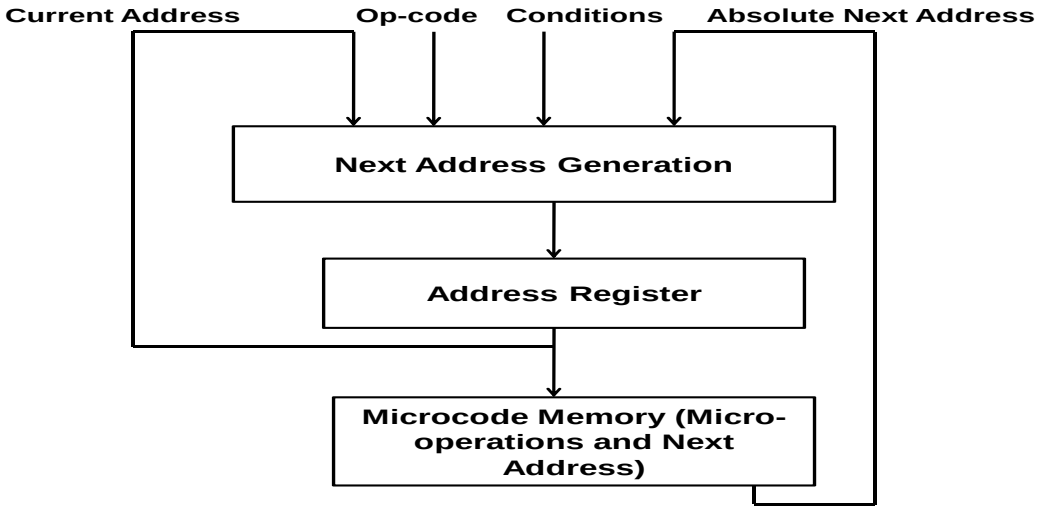

Figure 2: Generic Micro-sequencer Based Control Unit

Booth's multiplier: A software multiplier would normally use repeated addition method to carry out multiplication. A shift-add hardware multiplier would carry out the same multiplication

much faster. Moreover, a hardware multiplier following Booth's algorithm would carry out the operation faster when using 2's complement number representation. As discussed in [12, 13], a multiplicand and a multiplier register hold the inputs while a pair of shift-registers hold the final results. Control inputs for these registers are generated from the counter-decoder combination.

Non restoring divider: A software divider would normally use repeated subtraction method to carry out division. A shift-subtract hardware divider would carry out the same division much faster. Non restoring divider handles the division in a conservative way by subtracting only when the relevant portion of the dividend is greater than (or equal to) the divisor.

CSA & Wallace tree: In a multiplier, partial products are added two at a time using parallel binary adder. Carry Save Adder (CSA) is capable of adding three numbers at a time. Thus employing CSA to add the partial products yields multiplication results faster than standard multiplier. Wallace tree is built using multiple CSAs in parallel producing fast results through a purely combinational logic. An example tree for 3×3 multiplication having three partial products is shown in Figure 3.

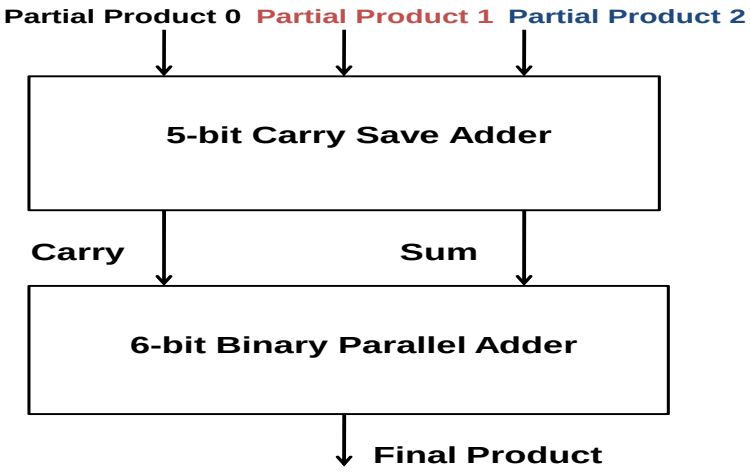

Figure 3: Wallace tree for 3×3 Multiplication

HOI & LOI: Memory system can be interfaced to a processor either using High Order Interleaving (HOI) or Low Order Interleaving (LOI). HOI offers easy expansion capability whereas LOI offers faster (pipelined) access to memory system.

Direct, Associative, & Set-associative cache: Fast access to recently used instructions and data is ensured by means of cache memory. Basis of cache design is Content Addressable Memory (CAM). Also called associative memory, CAM involves data, mask, match and output registers along with a memory as illustrated in Figure 4. However, different cache-tagging schemes have varied hardware complexities and hit rates under different applications. Also, different cache replacement policies and cache write-policies fit better in different situations.

Memory Management Unit (MMU): MMU has a key role in virtual memory implementation. MMU takes care of the page table and the logical-to-physical address translation process. Page replacement policies applied to the page table updating comprise of various algorithms. Recently

accessed frame numbers can be stored in a cache called Translation Look-aside Buffer (TLB). This ensures fast access to the recently used pages.

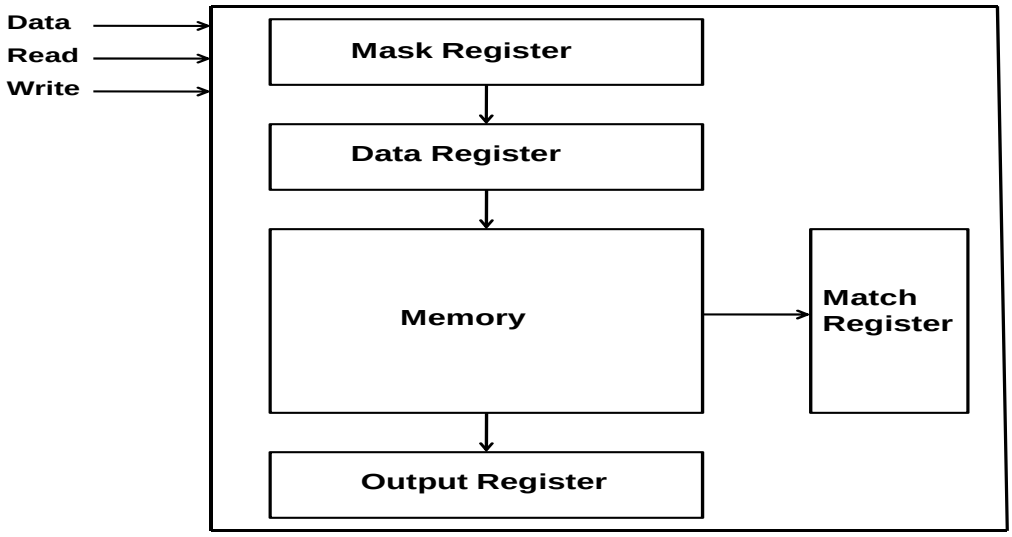

Figure 4: Organization of Associative Cache Memory

Instruction pipeline implementation: In order to speed-up the rate of execution of instruction stream, fetch-decode-execute cycle can be pipelined. Steady-state speed-up could be as high as the number of pipeline stages for a large stream of instructions. However, they are associated with data and control hazards that may result in wrong or undesired outputs. Techniques such as 'no-operation' insertion and instruction reordering could be employed to overcome such conflicts or hazards. These techniques may result in performance degradation as compared to non-existent conflicts.

3. Laboratory Contents

The following outline of experiments may be chosen to support the above mentioned objectives and the lecture contents. All the following experiments involve design of hardware (and associated software) elements. The hardware could be designed using VHDL and implemented on an FPGA for verification and for performance analysis. For all the experiments, references [12, 13, 14] can provide excellent background.

Hardwired control & Micro-sequencer control: The objective of this experiment would be to highlight the difference between the two techniques of control unit implementation. Hardwired control based processor is expected to run faster than the other. However, Micro-sequencer based control of the same processor would provide flexibility of adding instructions. An eight-bit processor as outlined in [2] could be used. For a hardwired control unit, counter-decoder approach is to be followed, whereas for micro-sequencer based control unit multiplexer-register-control ROM approach can be pursued [12]. This design would be used as a basis for performance comparison of several other hardware designs.

Booth's multiplier: The goal of this experiment is to appreciate the benefit of hardware multiplier in general and in specific the capability of Booth's algorithm to produce negative results correctly as compared to shift-add multiplier. This type of multiplier can be designed using registers, counters, and decoder. A software multiplier can be implemented using the above processor. Cycles needed by software approach can be compared to the cycles required by the Booth's hardware multiplier.

Non restoring divider: The purpose of this activity is to appreciate the benefit of hardware divider in providing with speed-up as compared to repeated subtraction. This type of divider can be designed using registers, counters, and decoder. A software divider can be implemented using the above processor. Cycles needed by software approach can be compared to the cycles required by the hardware divider.

CSA & Wallace tree: The objective of this experiment is to understand the basic building block of Wallace tree and how the tree provides speed-up compared to traditional multiplier. CSAs are built using standard binary full adders. Then Wallace tree is formed by parallel and series combination of CSAs and a binary full adder in the last stage. Its performance can be compared to software multiplier discussed above.

HOI & LOI: The goal of this experiment is to identify the alternative memory address distribution techniques among different memory banks. Also, it would enable the students to appreciate the advantage each technique has over the other. HOI can be achieved by selecting different banks using higher order address lines and provides more flexibility for expansion. Similarly, LOI can be achieved by selecting different banks using lower order address lines and provides fast access as compared to HOI.

Direct, Associative, & Set-associative cache: The purpose of this activity is to compare design complexities of each cache mapping technique. In addition to this, the students would also be able to appreciate the flexibility provided by various schemes.

4. Evaluation Strategy

The course outlined above may carry 4-credits (3 hours Lecture and 2 hours laboratory per week). 67% of the total weight is to be evaluated by means of coursework such as homework and quizzes, one midterm, and a final examination. The rest of the weight is assigned to the laboratory activities. The breakdown of the weight is shown in the Table 1.

Objective of the homework and quizzes is to test the understanding of the concepts developed so far and to prepare the students for the examinations. Examinations are non-accumulative, i.e. the topics covered are exclusive. 33% of the total point is to be evaluated by means of laboratory work. It includes evaluation of performance during the laboratory sessions and the reports submitted for each group of activities. Extra credits can be assigned for guest lectures and industry visits that enhance the concepts developed throughout the semester.

Table 1: Proposed Evaluation Rubric for the Course

Item	Weight
Homework 1	7
Homework 2	7
Homework 3	8
Midterm	22
Final	23
Laboratory	33

5. Research Connections

In order to extend student's learning experience, research activities involving new ideas to solve identified problems are important. One such research area is dynamic adaptation of architecture based on the application needs [15]. As an FPGA can be reconfigured easily, dynamic adaptation is close to reality. Many FPGAs (such as Xilinx Spartan and Virtex II) support partial reconfiguration. As such, adaptation can be done partially with minimal delay. However, identifying the right kind of architectural adaptation and its associated expense in terms of area and delay are still a challenge. A piece of research in this area was carried out during summer by students having a partial course background as sketched in this paper. A dynamic random application was formed using combination of five different benchmark kernels [16, 17]. The kernels are: Autocorrelation between two vectors, RGB to YIQ conversion, and High Pass Grey Filtering (HPG), 2D-DCT shuffling and FFT reordering. These kernels were designed for implementation in FPGA hardware. In running the applications, these kernels were programmed into FPGA by reconfiguring the existing architecture only when it provided performance improvement. The students extended such a simulation system as shown in Figure 5 and carried out performance studies [14].

We considered a system with multiple reconfigurable units. For such system, an appropriate distribution of a data set between the host and the reconfigurable hardware units could boost application performance. In this respect, we split the data set for each kernel of an application such that the complete execution time on the host and on the reconfigurable logic (including any overhead) is equal (or close). However, under this methodology we have two options to choose a reconfigurable module for hardware execution of the factored data set. We may choose a hardware module ensuring uniform utilization among all. This ensures better uniformity of usage. This policy is called ***load balancing with uniform utilization***. Alternatively, we may look-ahead in the task graph to find the next two upcoming kernels for execution. We choose a hardware module for execution that does not contain those upcoming kernels. This ensures reduced reconfiguration overhead and improved performance. This policy is called ***load***

balancing with look-ahead. We simulated both the variations of load balancing policies within our test bench.

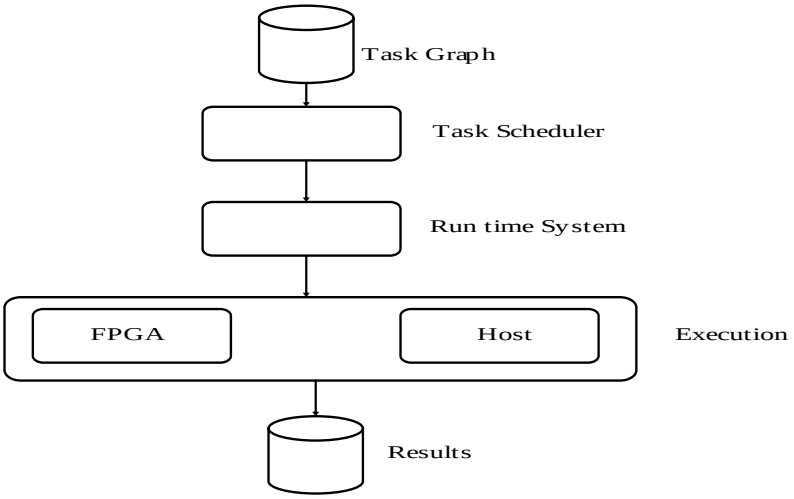

Figure 5: Developed Simulation Environment

Figure 6 shows the execution time for different application task graphs under these policies. As expected, the load balancing strategy provides performance gain, especially for large task graphs. The execution time savings are significant (about 35%) for task sizes of 249 under load balancing with look-ahead policy. However, the performance gain is not that prominent under load balancing with uniform utilization, as evident from Figure 6.

6. Plan to Incorporate

One possible placement of the proposed course could be in the junior year after Digital Logic Design and Microprocessor programming courses. However, VHDL needs to be incorporated in a prerequisite course. A course that involves VHDL programming and design implementation of an eight bit processor is already ongoing [2]. Student performance over three semesters in that course is summarized in Figure 7. As seen, the performance is skewed towards the high achievers end. As such, it is expected that a follow-on course as outlined above would be appropriate for these students having necessary background. To support the laboratory, FPGA boards need to be furnished in the laboratory. Necessary (complimentary) SW can be obtained from the vendors. The books (along with their supporting supplements) as listed in the bibliography could serve as the basis for developing both the lecture and the laboratory materials.

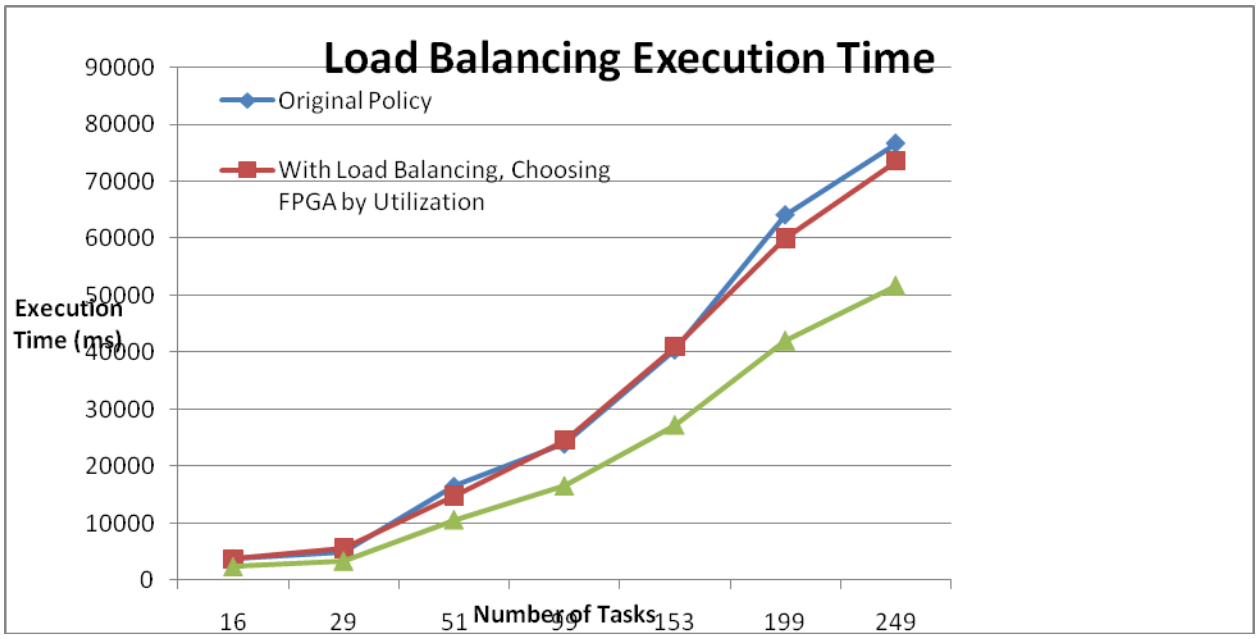

Figure 6: Execution Time under Different Policies for Load-balancing

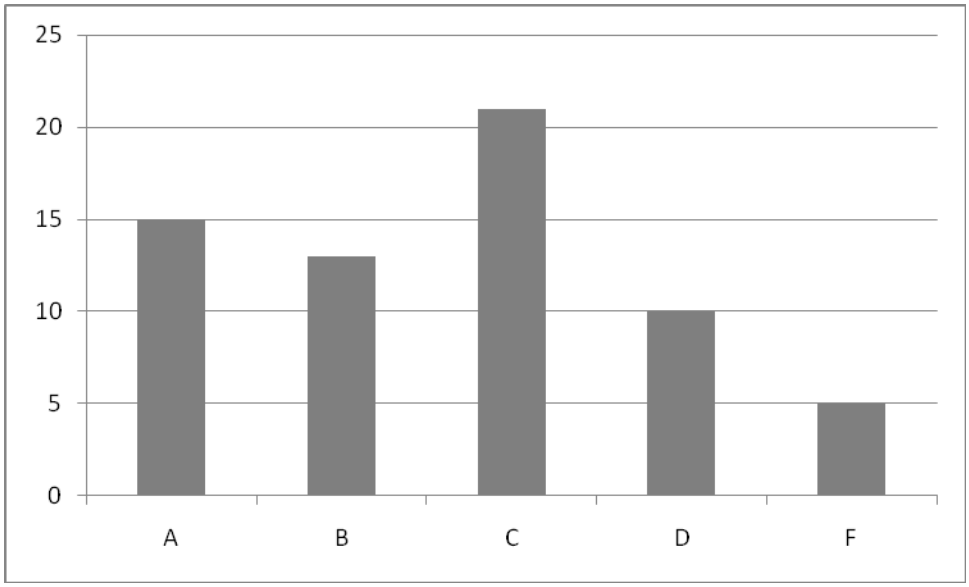

Figure 7: Summarized Student Performance in a Prerequisite Course

6. Conclusions

An outline of computer architecture and organization course that involves design and analysis utilizing reconfigurable hardware is presented in this paper. The focus is more on HW / SW co-design and integration rather than simple programming found in traditional courses. Significant emphasis is placed on laboratory work that involves design implementation and performance comparison. Reconfigurable hardware platform is proposed as a suitable vehicle for laboratory exercises as it can adapt well to the peculiar needs of system elements. Research elements associated with such a course has been highlighted that could extend the learning experience of the students. Future improvements to this proposed course could include advanced hardware debugging techniques using soft tools and proper code-documentation using version control. Dynamic (partial reconfiguration) could possibly be explored as well with the support of vendor tools. Moreover, advanced activities such as micro coded hardware design of conditional branch instructions, implementation of micro-subroutines and micro-code jumps, design of branch prediction unit and collision free scheduling could also be included both in the lecture and in the laboratory.

References

[1] Abul K. M. Azad, "Design and Development of an Introductory Digital Electronics Course within an Undergraduate Program", Journal of Engineering Technology, Spring 2008.

[2] M. Z. Hasan, "Course Development in Digital Systems Targeting Reconfigurable Hardware," ASEE Annual Conference, June, 2009.

[3] Carl J. Spezia, "A Task-Oriented Design Project for Improving Student Performance," Journal of Engineering Technology, Spring 2009.

[4] Stewart J. I., "Teaching and Assessing Using Project-based Learning and Peer Assessment," Journal of Engineering Technology, Spring 1999.

[5] Akins, Leah, M. and Ellena E. Reda, "Implementation of an Integrated Project for the Electrical Engineering Curriculum," Journal of Engineering Technology, Fall 1998.

[6] Gerhard, Glen, C., "Teaching Design with Behavior Modification Techniques in a Pseudocorporate Environment," IEEE Transactions on Education, November 1999.

[7] Wei Pan, S. Hossein Mousavinezhad, Kenyon Hart, "Digital Signal Processing: Theory and Practice, Hardware and Software," ASEE Annual Conference, June, 2009.

[8] James H. Aylor, "Teach Smarter, Cheaper," ASEE Prism, September, 2009.

[9] Eduardo Montanez, Michael Norman, "A Cost-Effective, Modular-Hardware Platform for Embedded Systems Design and Development," ASEE Annual Conference, June, 2009.

[10] Haluk Ozemek, Preetpal Kang, Albert Khanh Nguyen, Pradeep Badhan, "Sophomore-Level Programming Concepts and Methodology Course in Computer Engineering, Covering Both Hardware and Software," ASEE Annual Conference, June, 2009.

[11] Georgios Anagnostopoulos, Michael Georgiopoulos, Veton Kepuska, Kenneth Stanley, Alison Morrison-Shetlar, Pat Lancey, Paula Krist, Tace Crouse, "The Amalthea REU Program: Activities, Experiences, and Outcomes of a Collaborative Summer Research Experience in Machine Learning," ASEE Annual Conference, June, 2009.

[12] John D. Carpinelli, “Computer Systems Organization and Architecture”, Addison Wesley Longman, Inc. 2001.

[13] Behrooz Parhami, “Computer Arithmetic: Algorithms and Hardware Designs”, Oxford University Press, 2000.

[14] Behrooz Parhami, “Computer Architecture – from Microprocessors to Supercomputers”, Oxford University Press, 2005.

[15] M.Z. Hasan and S.G. Ziavras, “Reconfiguration Framework for Multi-kernel Embedded Applications,” 2nd Annual Reconfigurable and Adaptive Architecture Workshop (in conjunction with the 40th Annual *IEEE/ACM International Symposium on Microarchitecture*), Chicago, December 1, 2007.

[16] The EDN Consortium, http://www.eembc.org/.

[17] M.R. Guthaus, et al., “MiBench: A free, Commercially Representative Embedded Benchmark Suite**”,** *4th IEEE Annual Workshop on Workload Characterization*, Austin, Texas, Dec. 2, 2001.