AC 2010-45: HDL BASED DESIGN PROBLEMS FOR COMPUTER ARCHITECTURE

Chad Hager, United States Air Force Academy

Chad E. Hager received his B.S. in Electrical Engineering in December 2007 and his M.S. in Electrical Engineering in May 2009 from the University of Wyoming. His graduate work focused on the development of HDL based design problems for computer architecture. He has also developed and modified a teaching/robot platform used to educate undergraduate student in microprocessors. He is now a research associate with the Department of Electrical and Computer Engineering at the United States Air Force Academy, Colorado.

Steven Barrett, University of Wyoming

Steven F. Barrett, Ph.D., P.E. received the BS Electronic Engineering Technology from the University of Nebraska at Omaha in 1979, the M.E.E.E. from the University of Idaho at Moscow in 1986, and the Ph.D. from The University of Texas at Austin in 1993. He was formally an active duty faculty member at the United States Air Force Academy, Colorado and is now an Associate Professor of Electrical and Computer Engineering, University of Wyoming. He is a member of IEEE (senior) and Tau Beta Pi (chief faculty advisor). His research interests include digital and analog image processing, computer-assisted laser surgery, and embedded controller systems. He is a registered Professional Engineer in Wyoming and Colorado. He co-wrote with Dr. Daniel Pack five textbooks on microcontrollers and embedded systems. In 2004, Barrett was named "Wyoming Professor of the Year" by the Carnegie Foundation for the Advancement of Teaching and in 2008 was the recipient of the National Society of Professional Engineers (NSPE) Professional Engineers in Higher Education, Engineering Education Excellence Award.

Cameron Wright, University of Wyoming

Cameron H. G. Wright, Ph.D, P.E., is an Associate Professor with the Department of Electrical and Computer Engineering at the University of Wyoming, Laramie, WY. His research interests include signal and image processing, real-time embedded computer systems, biomedical instrumentation, and engineering education. He is a member of IEEE, ASEE, SPIE, BMES, NSPE, Tau Beta Pi, and Eta Kappa Nu. E-mail: c.h.g.wright@ieee.org Web page: http://wwweng.uwyo.edu/electrical/faculty/wright.html

Jerry Hamann, University of Wyoming

Jerry C. Hamann received the B.S. in Electrical Engineering with a Bioengineering Option from the University of Wyoming in 1984. He then worked for the Loveland Instrument Division of Hewlett-Packard before returning to the University of Wyoming to complete the M.S. in Electrical Engineering in 1988. Sharing time as a lecturer and National Science Foundation Graduate Fellow, he completed the Ph.D. in Electrical Engineering at the University of Wisconsin in 1993. As a faculty member at the University of Wyoming since 1993, Jerry has pursued research interests in applied robotics and control, signal processing, and higher education teaching and learning. He directed the University of Wyoming Hewlett Foundation Engineering Schools of the West Initiative until 2008, which is focused upon enhancing the recruitment, retention and quality of undergraduate engineering students. He now serves as Head, Department of Computer Science.

HDL Based Design Problems For Computer Architecture

Abstract

A computer architecture course is a necessary component in a computer engineering curriculum. Students in related disciplines such as electrical engineering and computer science may also value course concepts in the development of their elective coursework. There are many excellent computer architecture textbooks available to illuminate the difficult concepts encountered within the topic area. Many contain detailed designs of various architectures and configurations. To enhance the design skills and allow students to observe the dynamic operation of specific computer architectures, a series of Verilog Hardware Descriptive Language (HDL) design exercises were developed for a senior/graduate level course in computer architecture. The exercises allowed students to begin with basic review exercises on HDL design techniques and progress to fully operational computer architectures. The exercises were directly based and coupled with architectures presented in the course textbook by Mano and Kime. Student feedback indicated the exercises significantly enhanced their design skills and their overall understanding of computer architecture concepts. Students also demonstrated the capability to analyze more complex computer architectures and synthesize advanced components of a computer architecture and apply their knowledge to challenging open-ended design projects. Although originally developed for the Mano and Kime textbook, the design exercises described may be used with any computer architecture text.

Overview

There are many tools used in the design of microprocessors and microcontrollers to increase their speed and performance: manufacturing/processing, software development, and computer architecture. Previous advances in computer architecture were made possible by the reduction of the transistor size and performance and enhancement in architecture design. More recent computer architecture enhancements have focused on multiple cores and parallel processing in design. It is essential that computer architecture students understand the fundamental concepts as well as advanced techniques [1].

Computer architectures have evolved over many years and today there are many different types of computer architectures. Some are made for the general user, while others focus on a specific application. To help develop these architectures, engineers often use a Hardware Descriptive Language (HDL). In some cases, the HDL models can be compiled and implemented into field-programmable gate arrays FPGAs for further testing or even mask layouts for final or near-final production. Each processor manufacturer has its own specific procedures to move a design to the production phase. This being said, HDL based designs are clearly advantageous in academics, implying a need for homework based on HDL. This paper describes a series of homework assignments that have been recently developed to enhance the instruction of complex computer architectures using Verilog HDL as a design vehicle. The organization of these homework assignments attempts to show students the link between

software and hardware and to illuminate some of the more difficult architecture concepts discussed in class. The assignments are arranged to give students a gentle introduction into HDL, followed by the construction of memory. Later design exercises directs attention to arithmetic logic units (ALUs), single cycle computers, and pipelined processors. These assignments were developed for a senior/graduate level course in computer architecture for which the text by Mano and Kime is used [2]. The specific design assignments will be discussed and student reaction to the exercises is provided.

This project was planned and conducted by a graduate student as part of degree requirements for a master of science degree in electrical engineering. We have partnered with students on a number of related education projects over the past decade. We have dubbed this type of development project as "For Students By Students (FSBS)." In the FSBS model, students develop educational tools for use in the classroom and laboratory for use by fellow students. This approach has allowed us to custom design educational tools while providing students opportunities for hands-on development work. Additional information on this approach and related projects are provided in the literature [3-9].

In the next section we present background information for the reader to put into context the concepts related to the paper, followed by the methods used to develop the laboratory exercises. We then provide results of using the developed exercises in an elective senior/graduate level course during the Spring 2009 semester.

Background

Computer Architectures. There are two major categories of computer architectures: Harvard and Von Neumann architectures. Harvard architecture is defined by the separation of instruction memory from data memory. In this design there are two different physical locations of memory. Von Neumann architecture is defined by data and instruction memory existing within the same physical location. Some hybrid architectures, in which multiple specialized memory spaces are used, can also be found. Sub-categories of computer architectures also exist. Among these categories are, single cycle computers, pipelined computers, and multiple cycle computers. In some cases designs incorporate characteristics from several of these sub-categories, but the defining factor of which design should be used depends on the application. The single cycle computer completes instructions within one clock cycle. This type of architecture is typically slow because the propagation delay of all the internal units is summed. Pipelined computers use storage registers between a pre-determined set of units to store information and control signals for the next stage. This design, different from the single cycle, is only limited in speed by the slowest stage in the computer and the propagation delay in the storage registers. Multiple cycle computers utilize a state machine influenced by one instruction to instantiate a series of steps to complete a process. These instructions are complicated and in some cases, cannot be completed within one clock cycle. Logically the multiple cycle computer is necessary for complicated instructions. Rather than implementing multiple instructions to complete a task, the programmer can just implement one instruction, simplifying the programming [2].

Xilinx ISE. The Xilinx Integrated Software Environment (ISE) is a software package which is readily available to students to support the use of Xilinx programmable logic devices such as FPGAs and CPLDs. This software provides students with a link between hardware and HDL. Students can use this application to design complex systems and test them on several levels. Within ISE, there is a behavioral simulator where a design can be tested before synthesis and implementation on the chip level. A useful feature of this software is its hierarchal design solution, allowing the design to be implemented in sections and linked together on a higher level. Schematics, behavioral modeling, and gate-level modeling can all be used depending on the type of circuit that is being made [10].

Related Coursework. Computer Architecture is offered in the curriculum of the department of Electrical and Computer Engineering at the University of Wyoming. This senior/graduate level elective course has only has one prerequisite—Digital Verilog Design. In the sophomore level Digital Systems Design course, students are provided an introduction to Verilog HDL during their two-hours per week laboratory. This course is typically taken early in the student's undergraduate program, and Computer Architecture is usually taken as a graduate course. This results in a worst case time lag of two to three years between taking the two courses.

The Computer Architecture course uses the textbook written by M. Morris Mano and Charles R. Kime entitled "Logic and Computer Design Fundamentals [2]." The textbook provides an excellent coverage of computer architecture with many design examples provided. However, it was essential for students to experience the design process behind the examples and also observe the dynamic operation of the computer architecture examples. Therefore, we decided to develop a series of design exercises based on Verilog HDL. The exercises would allow students to implement and observe the dynamic operation of some of the example architectures described in the textbook. Also, students were required to develop and complete a final design project using Verilog HDL and the concepts learned within the course. We examined the literature and could not find HDL based exercises to support this textbook.

Students who elect to take this course are typically senior/graduate level students in electrical engineering, computer engineering, or computer science. In some cases students have only experienced HDL in the sophomore level Digital Systems Design course. A senior level elective design course in Verilog HDL is also available but is not a required pre-requisite to the Computer Architecture course. To accommodate students who do not have an extensive background in HDL, the assignments that were developed began with relatively simple HDL concepts and steadily increased in difficulty. A list of the assignments along with a short description is shown below:

- Introduction to HDL and Utilities—students are asked to design a half-adder using behavioral modeling. Next, they use their half-adder design and in a hierarchal approach along with gate-level modeling, they implement a full-adder unit. Finally, using the fulladder and a schematic approach, they design a four-bit adder-subtractor unit with carry out and overflow detection.

- RAM (Random Access Memory)—students are asked to design an 8x8 RAM array using any design method.

- ALU (Arithmetic Logic Unit)—students must design a sixteen operation ALU. The ALU must operate on two 8-bit operands and provide a single 8-bit solution. In addition, status bits (carry out, overflow, zero, and negative) must be added.

- Single Cycle Computer—student must design a computer that performs one operation in one clock cycle with the requirement that the bus size needs to be 8-bits.

- Stack Memory and Interrupts—students are asked to design a stack pointer which communicates with data memory. Then through proper procedures, design a test solution which simulates an interrupt event.

- Pipelined Computer—students are asked to create a two-stage pipelined computer using the single cycle computer design. In addition, the ability to jump to a particular place in instruction memory must be added. A test routine is completed to show a data hazard event.

- Expansion—students expand the single cycle computer to a 16-bit data bus and develop their own test routine.

- Project—students pick a design related to computer architecture. They develop and test this design followed by a presentation.

Related Efforts. The concept of using HDL to teach computer architecture concepts is not new. MacDonald et al. used Very High-Speed Integrated Circuit Hardware Descriptive Language (VHDL) as early as 1992 to develop a novel computer architecture designated the WM to teach high performance architecture [11]. Huang et al. also used VHDL to instruct computer architecture. Huang noted several challenges with the approach most notably VHDL tool availability to students and proper lecture material to coincide with course objectives [12]. Hyde used Verilog HDL companioned with the popular text "Computer Architecture: A Quantitative Approach" [13], to instruct computer architecture concepts [14]. Hyde noted "Students in computer architecture courses, especially undergraduates, need to *design* computer components in order to gain an in-depth understanding of architectural concepts. For maximum benefits, students must be active learners, engage the material and design, i.e., produce components to meet a specific need." Calazans et al. supported Hyde's view. They reported that "…students should learn how computers work not only by studying their inner details, but also by concomitantly building processors and computers or embedded systems" [15]. Hill at the University of Hartford has developed a series of hypothetical microprocessor architectures designated the "nod." The nod is implemented within an FPGA and provides performance similar to traditional small microcontrollers such as the Motorola/Freescale 68HC11 [16, 17]. Smith at the University of St. Thomas has developed a spreadsheet CPU that simulates the operation of a central processing unit for teaching purposes [18]. Most recently Hayne at the Citadel has developed VHDL homework exercises and a capstone design project to provide hands-on application of computer architecture course concepts [19]. We hold this entire body of related efforts in the highest regard. However, we needed a Verilog HDL based architecture to

directly support the architecture described in Mano and Kime [2], which resulted in the need for this project.

Methods

A series of gradually more challenging homework assignments were developed for the computer architecture course. An overview of each assignment follows.

Assignment 1—Introduction to HDL and Utilities

As an introductive exercise, students implement basic components using different designing schemes. Knowing how to effectively navigate through these design schemes assist them in future homework. The following strategies are given to the students:

Implement a half adder using dataflow modeling:

- Outputs: S-Sum, C-Carry
- Inputs: X-Bit 1, Y-Bit-2

Use hierarchal and gate-level modeling to implement a full adder:

- Outputs: S-Sum, C-Carry
- Inputs: X-Bit1, Y-Bit-2, Z-Carry In

Create a schematic symbol of the full adder and develop a 4-bit adder/subtractor with carryout and overflow detection:

- Outputs: Sum [3:0] – Sum of X and Y, V-Overflow, C- Carry Out
- Inputs: Sel-Selection between add and subtract, X [3:0] and Y [3:0] – Inputs in the form X - Y or X + Y

If done correctly, a solution similar to Figure 1 is made. Within this solution there are four full adder units (FullAdder). Full adders have the ability to add two binary digits and produce a sum bit (Sum) and a carry bit (Carry). In this arrangement, the add/subtract selection bit (M) is applied to XORs to invert the second operand (BInput) during a subtract operation. M is also applied to the carry input (Zin) of the first full adder unit. This design was tested in a behavioral simulation to verify its operation.

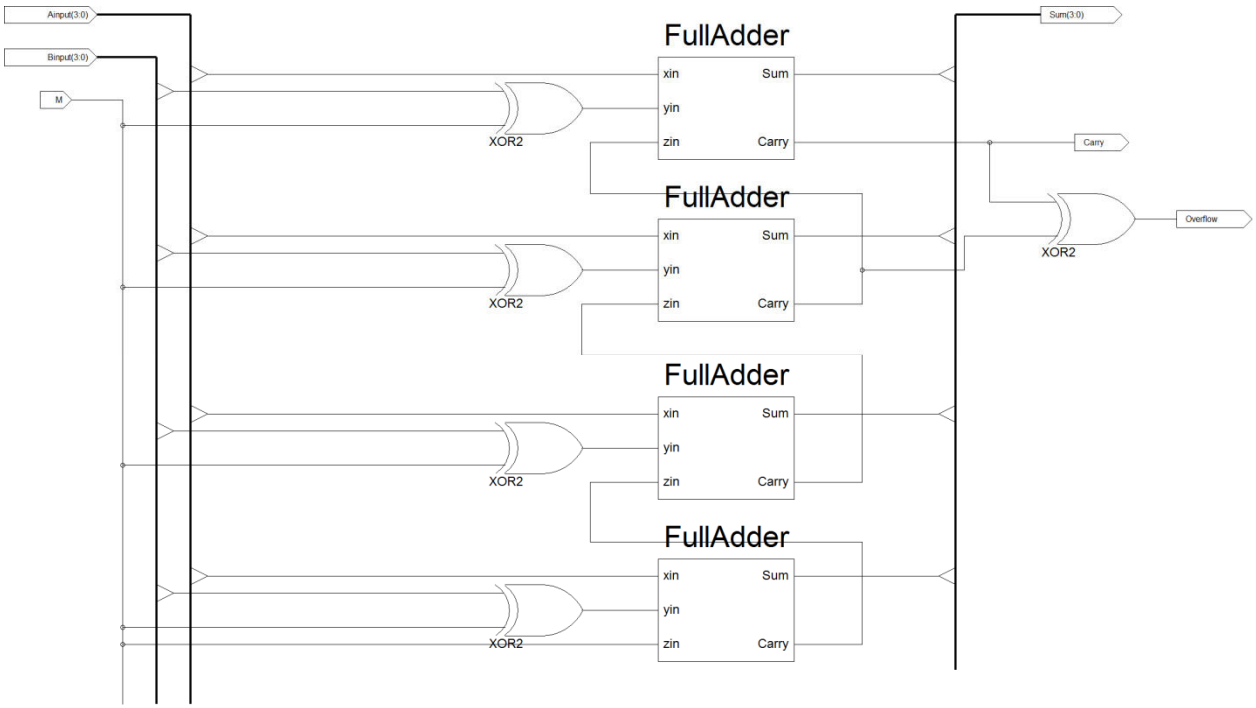


Figure 1. Four-bit adder/subtractor with carry out and overflow status bits.

Assignment 2—RAM

Using any HDL design method, students produce and 8x8 RAM array (8-8-bit words). Ultimately the design is left open ended so the student has the option to utilize the best means of development. Figure 2 is introduced to aid the students in the formatting of the RAM array. Note that there are typically four inputs to memory: Data In, Word Select, Read/Write, and Enable. Data In represents the data to be written to the memory. Word Select represents the address of the word that either a read or a write operation can effect. Read and Write are the commands for their respective operation. Enable is the controller for the entire RAM array. Enable is also very beneficial when expanding memory.

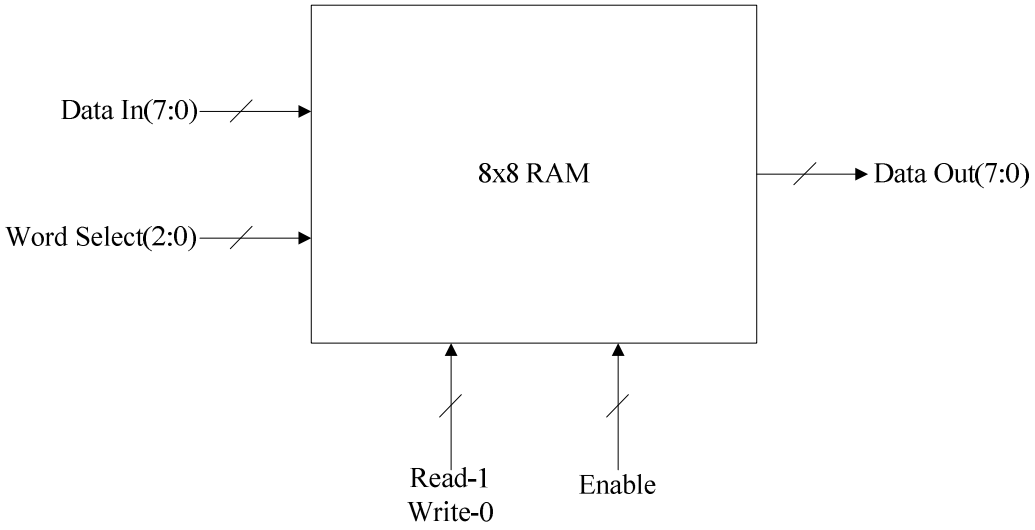


Figure 2. 8x8 RAM array with Enable

Several solutions exist for this design; however, using the "reg" command to instantiate registers then using a behavioral coding style with an "always" condition, produces a quick and effective solution. Alternative solutions might involve developing a memory cell by either a D-flip-flop or an S-R latch, then cloning the cells and linking in such a way to produce the 8x8 RAM array.

Assignment 3—ALU

Students develop and ALU with up to sixteen commands, implying that the Function Code must have four bits, Figure 3. The ALU must also handle two 8-bit operands A and B. Four true/false signals (i.e., status bits or flags) are added for when the operation shows: overflow, carry out, negative, and zero. Overflow is the event where the allotted memory cannot accurately represent the result. Carry Out is the carry bit from the last location in the addition or subtraction operation. Negative is where the last bit in the solution is a 1 (2's compliment notation). Zero is the indication that the result is zero.

Designs for an ALU might include a multiplexer approach where a series of select bits choose a particular operation then through a multiplexer the solution is routed to the output. This method contains an extra degree of control different from the behavioral method. Using the behavioral method the designer uses a "case" statement to select the appropriate operation. This design is easier to implement versus the multiplexer approach, but lacks control when adding the status bits. The status bits Carry Out and Overflow have to be implemented in a separate section and communicate with the case statement in order to detect this event. Status bits zero and negative are easily implemented with an "assign" statement typically involving a decision.

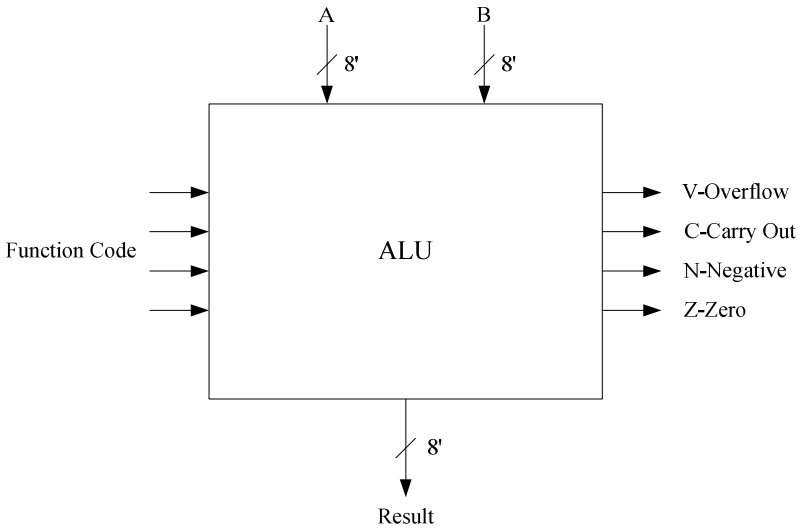


Figure 3. Block diagram of the 8-bit ALU with overflow, carry out, negative, and zero detection.

Assignment 4—Single Cycle Computer

Using the designs from assignments 2 and 3, students develop an 8-bit data bus single cycle computer. They follow the design in [2] with the following stipulations for inputs, outputs, and exceptions:

- Input(7:0) – Program counter control to set the program count to a particular value
- Input – Program counter control to enable the setting of the program counter
- Input – Clock applied to the program counter

- Output(7:0)– Datapath which is the output from the multiplexer connected to the ALU/Data Memory
- Output – Overflow detection
- Output – Carryout
- Output – Negative
- Output – Zero
- Output(15:0) – Instruction memory output which is the 16 bit instruction outputted from the instruction memory before it reaches the instruction decoder
- Output(x:x) – Any other output which might be helpful for debugging

- The Data Bus must be an 8 bit design
- Branch control can be removed for simplification
- Extend can be removed for simplification

The single cycle computer uses the data memory and ALU developed in assignments 2 and 3, respectfully. Apart from these units, the student must also develop the following parts:

- Program Counter

- Address Filter
- Instruction Memory
- Message Control Unit/Instruction Decoder
- Multiplexers
- Zero Fill
- Register File

Noticing Figure 4, the program counter (ProgramCounter) keeps track of the current instruction address. This address is applied to the address filter (AddressFilter) which crops the three least significant bits before applying the address to the instruction memory (InstructionMemory_EightInst). The instruction memory is hardcoded by an always condition and a case statement. The selected instruction is outputted to the instruction decoder (MessageControlUnit) which decodes the instruction and produces control bits for the datapath. In the datapath, the register file (RegisterFile_EightByEight) can hold up to 8-8-bit words and can output two simultaneous arguments while only writing one argument. Multiplexers control the inputs into the data path, whether they come from the data memory (DataMemory) or the immediate operand (ZeroFill). An ALU takes care of any mathematical or logical operations within the datapath. Bus taps are located throughout the single cycle to observe the operation during execution. The test routing for the single cycle computer is as follows:

- Instruction 0: R0 ← 5
- Instruction 1: R0 ← R0 + 1
- Instruction 2: R0 ← ~R0

Using the waveform in Figure 5, while observing the datapath section (DataPath[7:0]), the single cycle correctly places 8'h05 into the register file. Then the value of 8'h05 is incremented to 8'h06 and finally the value of 8'h06 is bit-by-bit inverted to the value of 8'hF9 (249 decimal). Status bits are working in this design but appear as indeterminate conditions before they are examined by the ALU.

Assignment 5— Stack Memory and Interrupts

Stack memory is constructed using data memory from Assignment 2. An additional unit, the stack pointer, is developed and implemented through some kind of flow control mechanism (multiplexers or tri-state logic), Figure 6. Through another flow control device, the Program Counter (PC) is applied to the data memory both as an input and output. The input to the PC, is the Interrupt Vector Address (IVA) and the return from the data memory. Using the behavior simulator, students act as the control unit for this circuit in attempt to simulate an interrupt. Different from previous assignments, this problem is not completely specified to give students the opportunity to explore the design procedures within the context of an open-ended problem.

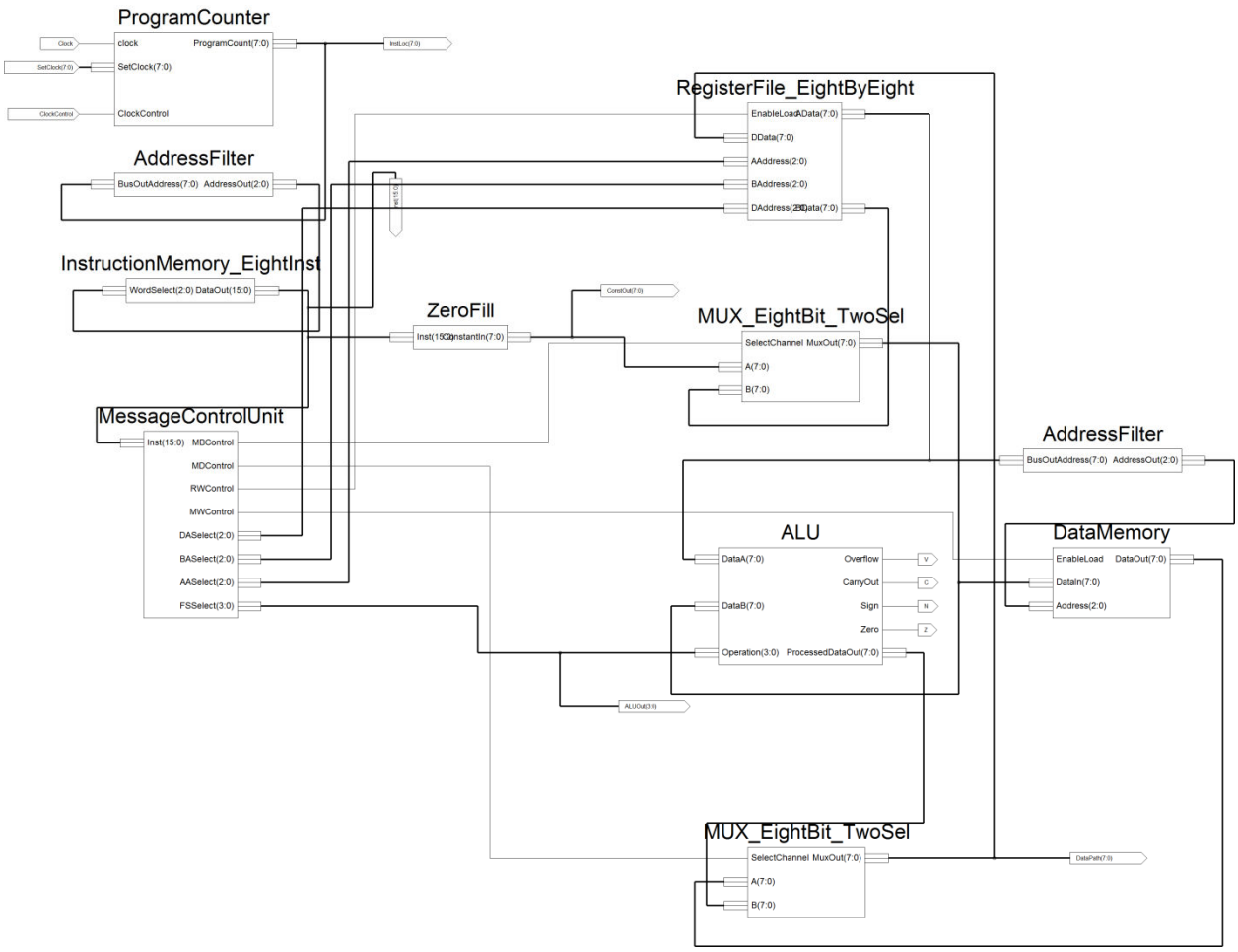


Figure 4. 8-bit Single Cycle Computer with hardcoded instruction memory.

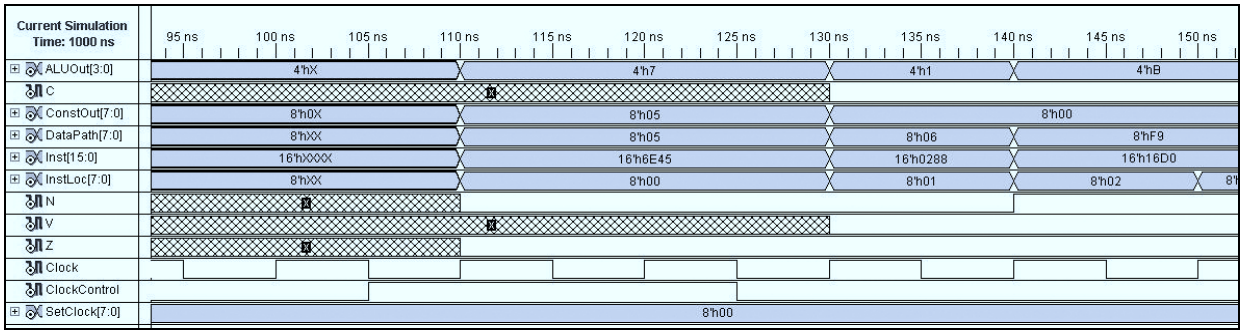


Figure 5. Waveform simulation of the 8-bit Single Cycle Computer.

Typically during an interrupt, important information such as the register file values, the program count, and status bits are all stored on the stack. When the interrupt is completed the values are replaced and normal operation continues. To examine an interrupt service routine students are given the following steps:

- Set the stack pointer to the bottom of data memory plus one
- Decrement the stack pointer and save the contents of the program counter to data memory
- (PUSH)

- Apply a new address to the program counter IVA
- Wait one clock cycle
- Restore the address of the program counter by accessing the stack (POP/PULL)

If done correctly a waveform similar to Figure 7 is produced. On DataIn[7:0] the value of the program counter (8'b01) is stored in data memory by a PUSH command. Then a jump routine is executed positioning the program counter to 8'hF0 (IVA). One clock cycle passes and the program counter continues to increment. Finally during the next clock cycle the program counter is reset to the original value through a POP/PULL command.

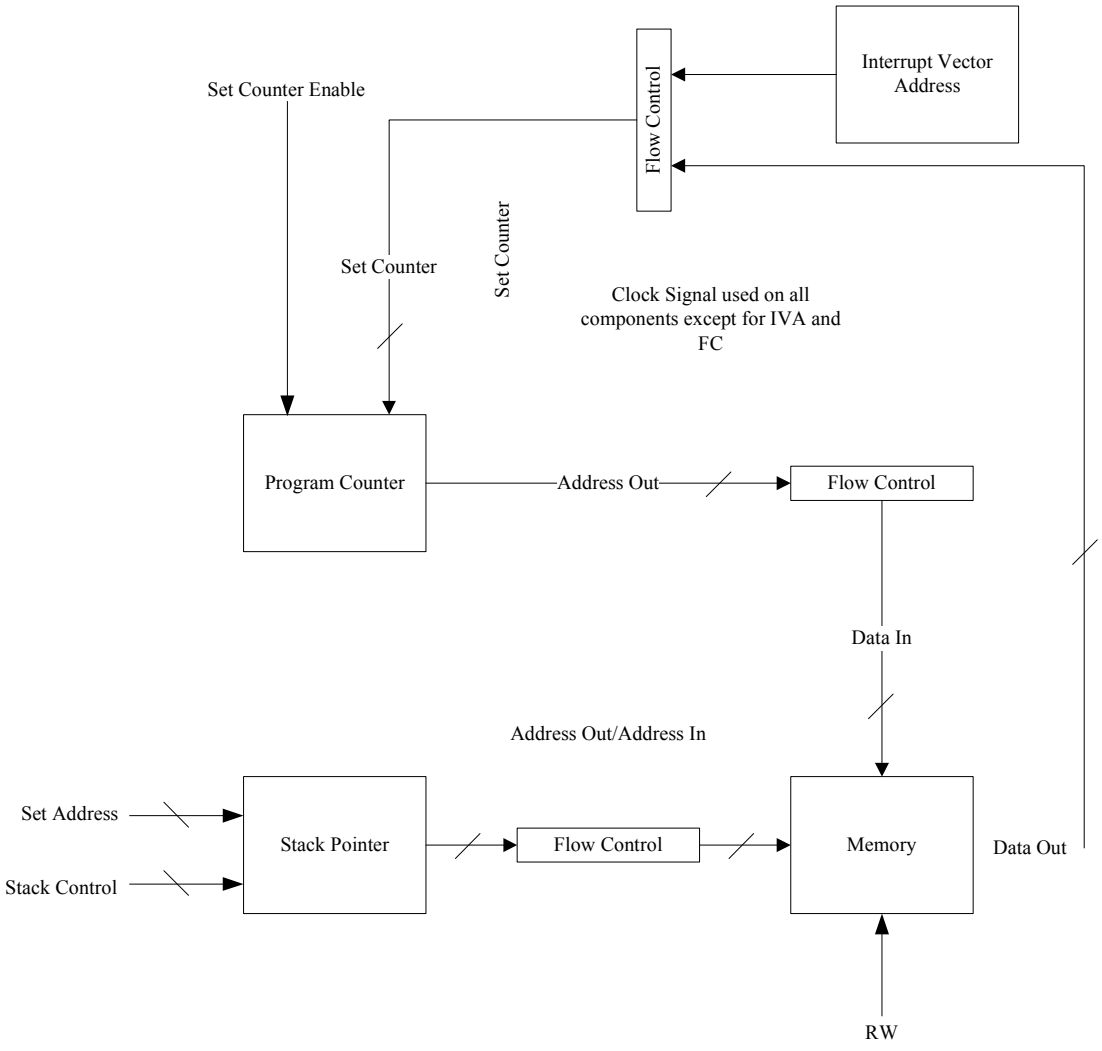


Figure 6. Data flow diagram of stack operation for an interrupt procedure.

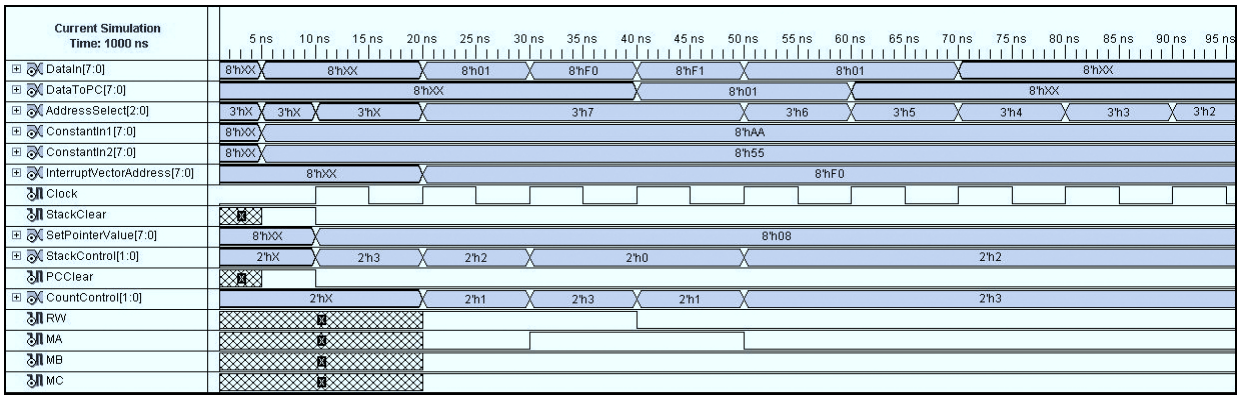


Figure 7. Interrupt routine using the stack pointer.

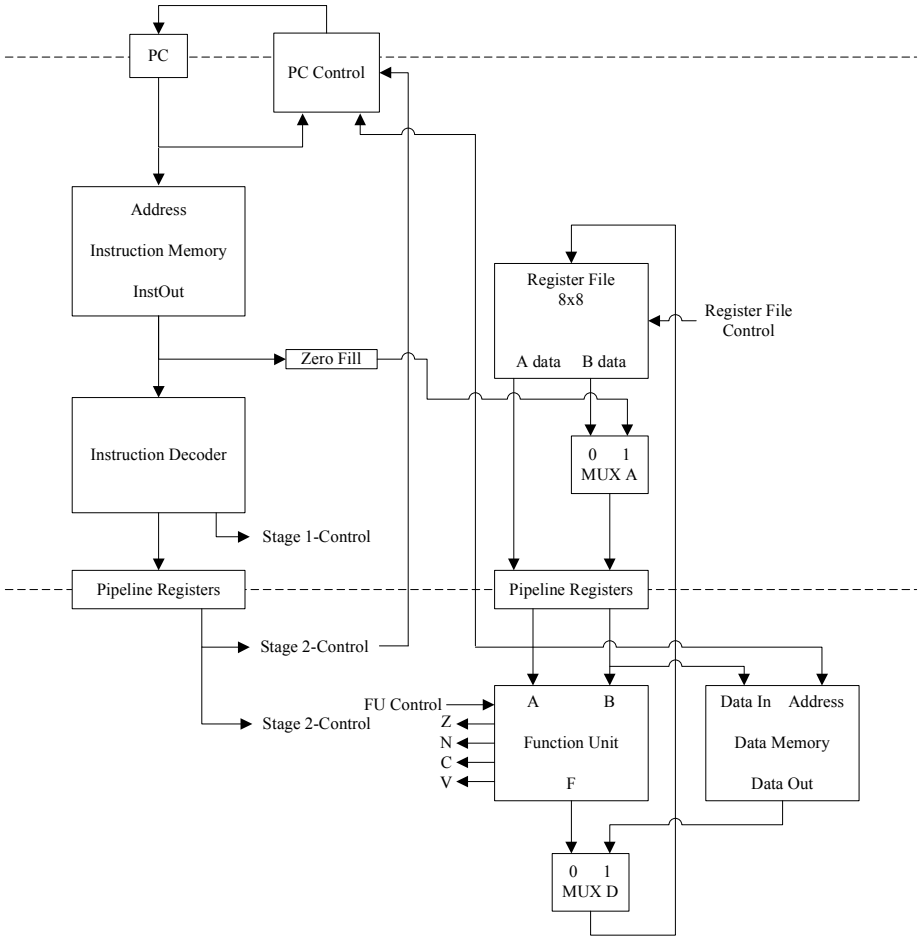


Figure 8. Data flow diagram of a Two-Stage RISC Pipelined Computer.

The two-state pipeline computer is modified slightly from Assignment 4, Figure 9. Added to the single cycle are the pipeline storage registers (Pipeline_ID_MUX) and the Jump routine which is a modification to the program counter (ProgramCounter_Mod). Control signals are applied strategically before and after the pipeline storage registers. In Figure 10, noticing the datapath (DataPath[7:0]), the value of 2 is initially stored on the register file (120 ns). Subsequently, the initial value of 0 is set to the R1 register (150 ns). For 10 ns a NOP follows

appearing as an indeterminate condition. At 170 ns the R1 register is incremented followed by a Jump routine back to the second instruction. The execution of the second Instruction 3 will not occur on the datapath until 220 ns. Students also test the pipelined computer without using the NOP instruction. The R1 register will not be valid and a data hazard has occurred. Students might also consider a read-after-write register file which should also fix the problem but will only work as a fix for the two-stage pipeline.

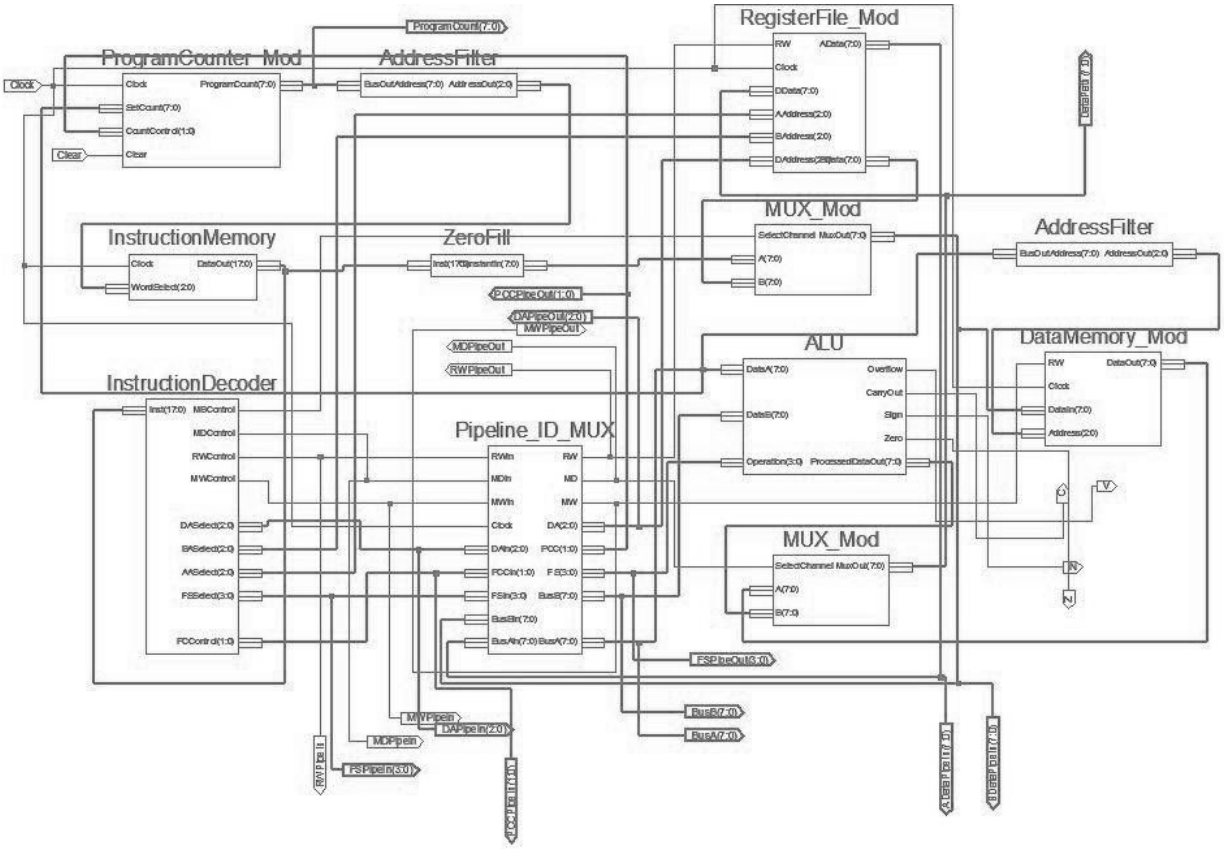


Figure 9. Two-Stage Pipeline Computer

Assignment 7—Expansion

Utilizing the result in Assignment 4, students expand the data bus to 16-bits and implement their own test routine for the single cycle computer. There are no new features to this design other than the expanded bus. The purpose is to show students how to make adaptable code. In some cases students can use a parameter to define the bus width and simply adjust the value to change the design. Using a parameter is favored in the development of all the units within the single cycle computer.

Essentially, the result should be similar to Figure 4 with the only change being the 16-bit data busses. Students are asked to implement their own test routine to exhaustively examine the interconnectivity of the single cycle. Note that students are not asked to test each of the internal units since they were already examined; however, the test routine must involve all pieces of the single cycle computer.

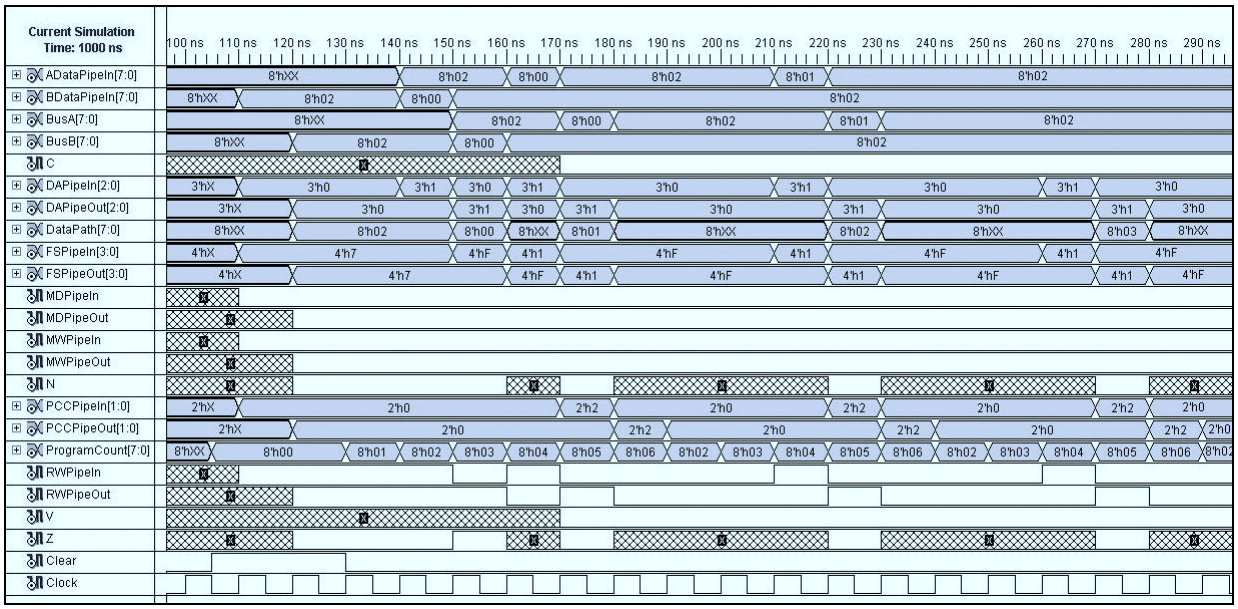


Figure 10. Test routine of the NOP coded Two-Stage Pipeline computer. Additional information is present to monitor the operation of the computer.

Assignment 8—Project

Students choose a project based on computer architecture which needs to involve an HDL based solution. Designs might include:

- Barrel Shifter
- Multi-CPU Computer Architecture
- Floating Pointer Precision Calculator
- Memory Management System

An appropriate design might involve a ground up implementation of a simple Multi-CPU Computer Architecture. Observed in Figure 11, the Multi-CPU Computer contains two CPU units interfaced to a common memory. Each CPU is similar to the design in assignment 4; however, the CPUs are changed to a Von Neumann type architecture, since data and instructions are stored in the same memory. The main memory produces the 24-bit instructions by a concatenation of several memory words. The behavior of the computer mimics the purpose of multi-CPU computers, meaning that multiple CPUs operate on the same memory. This design is simple since the memory hierarchy has been removed.

During startup each CPU addresses the first location in main memory (MainMemory). Accessing a jump command at this location, the CPUs point to a new location in memory anointed by the program set value (SetCountPCX[7:0]). At this location each CPU accesses its own specific program and begins to execute. In the event that more than one CPU attempts to access the same memory location, a memory access error bit is enabled (MAE). Future additions to this design might be a memory management system which defines a priority during a memory access error. A memory hierarchy can also be added pulling information from main memory into cache then operating on the memory in the cache.

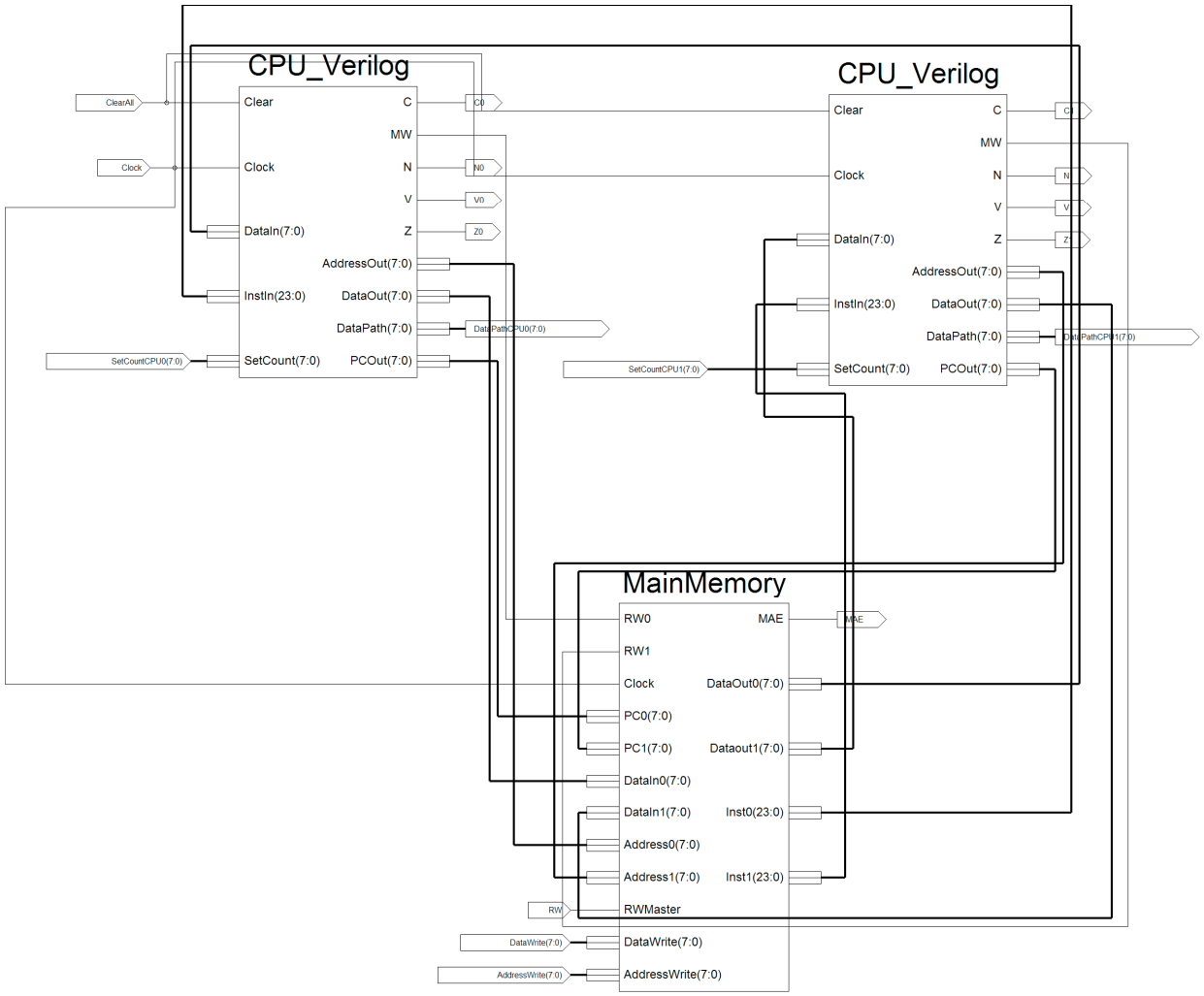


Figure 11. Simple Multi-CPU Computer Architecture.

Results and Discussion

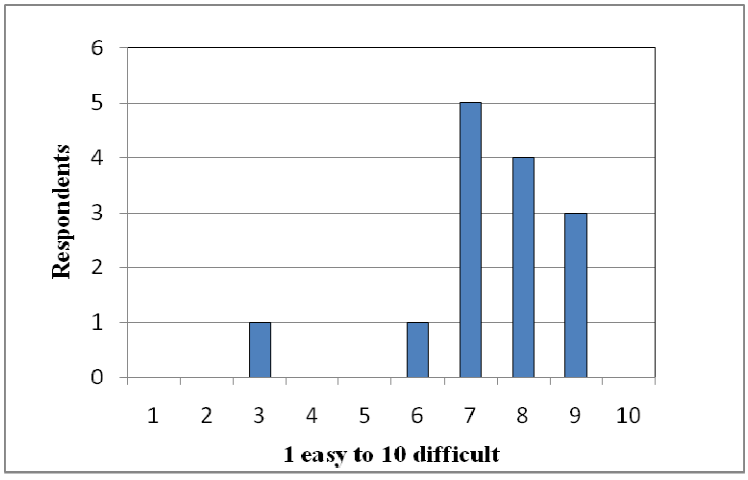
Approximately twenty students were given these assignments during the Spring 2009 semester. A post-course survey was administered to gage student response to the design exercises. Here is a brief snap shot of students' end-of-course comments about the Verilog HDL design exercises.

Question: In what ways do you feel the HDL assignments were advantageous in your understanding of Computer Architecture?

- "The HDL assignments are the foundation for understanding the material in the class."
- "Tested your knowledge of things discussed in class book problems were very complex."
- "Some of this stuff is really just hot air and theory until you hit the ground running in the assignments."

- "In doing them, it gives a more in depth [view] of the parts of a processor which is really handy and helpful."
- "It made you actually think about the implementation instead of just how stuff worked. This is a true difference between understanding the subject and actually being able to design something."
- "The assignments related the key concepts in an actual design."
- "It brought the design problems in our laps and we had to figure out how to do things on our own. Great for troubleshooting."
- "We actually designed what was needed in computer architecture so that in itself defined the course and we could say we knew how to do it."
- "I thought the HDL assignments were cool. They showed me how the concepts related to real world examples."
- "It applies the concepts to a real world application. Sometimes it's hard to tell exactly how something works until you implement it yourself."
- "Implementing/troubleshooting the HDL filled in the gaps and details on where to begin (usually the bottom level) and how parts interact."
- "Actually implementing them requires you put a lot more thought into what was taught in class. Pretty helpful in answering questions you didn't even know you had." "Very helpful, but not enough direction given when assignment is handed out."
- "HDL helped reinforce what was learned. Too hard though."

Question: On a scale of 1 to 10 (1 easy – 10 difficult), in general, rate the HDL assignments.



After a careful review, it was determined that incorporating the HDL based homework problems enhanced the pedagogy of the class, and resulted in deeper understanding by the students. Most students tended to prefer the 4th assignment (single cycle computer), while most students thought that assignment 6 was the most difficult. In all, students felt that the arrangement and difficulty of these assignments was necessary in the development of HDL skills, both in design and in testing. Students preferred these assignments as they gave them the opportunity to develop fully functional models.

In addition to the students' perception of the exercises, we also qualitatively compared the design sophistication of students. Students who completed the course prior to implementation of the design projects were able to answer advanced questions on specific computer architectures via examination. They were also required to report on an architecture of their choice.

Students completing the course with the Verilog HDL design exercises demonstrated the same aptitude as students in the earlier offering of the course but they also demonstrated the capability to **design** and analyze advanced components within a specific architecture on examination. Furthermore, they demonstrated the capability apply their new found skills in an open-ended end-of-course design project.

Conclusions

In conclusion, HDL based assignments are useful in improving the understanding of computer architecture compared to conducting a similar course without such assignments. Students also felt that the development of their HDL skill was increased by completing the assignments. As they completed the HDL based assignments, students began to understand simple computer architectures and the relationship between software and hardware. The skill acquired in this class can be extrapolated to larger projects with increased complexity. Although originally developed for the Mano and Kime textbook, the design exercises described may be used with any computer architecture text.

Acknowledgments

We would like to thank the students in Computer Architecture (EE 5390) at the University of Wyoming, in the departments of Electrical and Computer Engineering and Computer Science. Their commitment to this class gave us the opportunity to explore a new method of educating students in the field of computer architecture.

We would like to acknowledge M. Morris Mano from California State University, Los Angeles and Charles R. Kime from the University of Wisconsin, Madison. "Logic and Computer Design Fundamentals-Fourth Edition" has proved to be a valuable tool in the development of the HDL based assignments.

All coursework described in this paper is available for adoption by contacting Steven Barrett at steveb@uwyo.edu

References

[1] D. Soldan, J. Hughes, J. Impagliazzo, A. McGettrick, V. Nelson, P. Srimani, M. Theys, "Computer Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering – A Volume of the Computing Curricula Series December 12, 2004." *The Joint Task Force on Computing Curricula IEEE Computer Society, Association for Computing Machinery,* 2004.

[2] M. Morris Mano, Charles R. Kime, "Logic and Computer Design Fundamentals—Fourth Edition," Pearson Education, Inc., 2008.

[3] S.F. Barrett, A. Wells, C. Hernandez, T. Dibble, Y. Shi, T. Schei, J. Werbelow, J. Cupal, L. Sircin, G. Janack, "Undergraduate Engineers for Curriculum and Laboratory Equipment Development," *Computers in Education Journal,* Vol. XIII, No. 4, 2003, 46-58.

[4] A. Griffith, S. F. Barrett, D. Pack, "Verilog HDL Controlled Robot For Teaching Complex Systems Design," *Computers in Education Journal,* Vol XVIII, No. 1, Jan – Mar 2008, 63-72.

[5] C.H.G. Wright, D. Mares, S.F. Barrett, T. Welch, "Digital Signal Processing and Bioinstrumentation Using Labview, the New ELVIS Benchtop Platform, and BIOPAC," *Computers in Education Journal*, Vol. XVII, No. 2, 104-112, Apr-Jun, 2007.

[6] S. F. Barrett, D.J. Pack, P. Beavis, M. Sardar, A. Griffith, L. Sircin, G. Janack, "Using Robots to Teach Complex Real Time Embedded Systems Concepts," *Computers in Education Journal,* Vol. XVI, No. 4, 58-58, Oct-Dec 2006.

[7] S. F. Barrett, D.J. Pack, C. Straley, L. Sircin, G. Janack, "Real-Time Operating Systems: A Visual Simulator," *Computers in Education Journal*, April – June 2005.

[8] S.F. Barrett, A. Wells, C. Hernandez, T. Dibble, Y. Shi, T. Schei, J. Werbelow, J. Cupal, L. Sircin, G. Janack, "Undergraduate Engineers for Curriculum and Laboratory Equipment Development," *Computers in Education Journal,* Vol. XIII, No. 4, 2003, 46-58.

[9] S. Barrett, C. Hager, M. Yurkoski, R. Lewis, M. Jespersen, Z. Rubel, "Undergraduate Engineers for Curriculum and Laboratory Equipment Development: A Freescale S12 Microcontroller Trainer," *Computers in Education Journal*, Vol. XVIII, No.1 Apr-Jun 2008, 22-32.

[10] Anon, "Xilinx, Inc.", http://www.xilinx.com, 2009.

[11] R. MacDonald, S. Srinivasan, R. Williams, and J. Aylor, "A Novel VHDL-Based Computer Architecture Design Methodology," in *Rapid System Prototyping, 1992, International Workshop on Shortening the Path from Specification to Prototype*, Research Triangle Park, NC, June 1992.

[12] T. Huang, R. Melton, P. Bingham, C. Alford, and F. Ghannadian, "The Teaching of VHDL in Computer Architecture," in *Proceedings of the 1997 International Conference on Microelectronics Systems Education*, 1997.

[13] J. Hennessy and D. Patterson, Computer Architecture A Quantitative Approach, 3rd ed., Morgan Kaufmann, San Francisco, 2003.

[14] D. Hyde, "Using Verilog HDL to Teach Computer Architecture Concepts," in *Proceedings of the 25th International Symposium on Computer Architecture*, Barcelona, Spain, June 1998.

[15] N. Calazans, F. Moraes, C. Marcon, "Teaching Computer Organization and Architecture with Hands-On Experience," in *Proceedings of the ASEE/IEEE Frontiers in Education*

Conference, Boston, MA, Nov 2002.
[16] J. Hill, "Custom Processor Using an FPGA for Undergraduate Computer Architecture Courses," in *Proceedings of the American Society for Engineering Education*, Honolulu, HI, June 2007.
[17] J. Hill, "Microprocessor Architecture with FPGA Implementation for Undergraduate Computer Architecture Courses," *Computers in Education Journal*, Vol. XVIII, No.1 Jan-Mar 2008, 93-102.
[18] R. Smith, "A Spreadsheet-based Simulation of CPU Instruction Execution," in *Proceedings of the American Society for Engineering Education*, Honolulu, HI, June 2007.
[19] R. Hayne, "VHDL Projects to Reinforce Computer Architecture Classroom Instruction," *Computers in Education Journal*, Vol. XVIII, No.1 Apr - Jun 2008, 98-112.