

# Using Rapid Prototyping in Computer Architecture Design Laboratories

James O. Hamblen, Henry Owen, Sudhakar Yalamanchili, and Binh Dao  
School of Electrical and Computer Engineering  
Georgia Institute of Technology  
Atlanta, Georgia 30332-0250

## Abstract

*This paper describes the undergraduate computer architecture courses and laboratories introduced at Georgia Tech during the past two years. A core sequence of six required courses for computer engineering students has been developed. In this paper, emphasis is placed upon the new core laboratories which utilize commercial CAD tools, FPGAs, hardware emulators, and a VHDL based rapid prototyping approach to simulate, synthesize, and implement prototype computer hardware.*

## 1 Introduction

The Bachelor of Computer Engineering degree has been offered at Georgia Tech by the School of Electrical and Computer Engineering since 1986. Three years ago, the undergraduate computer engineering curriculum was revised. A total of 203 quarter hours is required for the BCmpE degree. In addition to the CmpE classes, twenty three hours of electrical engineering courses and seventeen hours of computer science courses are required. Students must select one of three areas of concentration, Software and Applications, Systems and Architectures, or Devices and Circuits. Each area of concentration has a different list of courses from which students select eighteen additional hours of EE, CS, or CmpE courses.

A new required core sequence of six CmpE classes with parallel laboratories was developed for the new curriculum. The required core sequence is a total of twenty-six quarter hours of lecture and laboratories. The first two courses in the sequence are also required for electrical engineering students. A host of new CAD tools were introduced into the core sequence laboratories. VHDL was chosen to enable modeling, simulation, and synthesis of digital logic and computer systems throughout the core sequence. This paper describes the core sequence of required computer architecture courses.

## 2 Digital Computation I and II

Digital Computation I and II, Computer Engineering 2500 and 3500, are required four hour, one quarter, courses taken in sequence by computer and electrical engineering students. This course sequence is an updated digital logic course sequence which reflects many of the recent advances in VLSI. The first course in the sequence covers basic gates and simple state machines. The final design example is a

very simple computer. Annual enrollment is around five hundred students per year. The second course covers more complex state machine design examples and provides an introduction to technology, timing, and testing issues. A required textbook[1] and a laboratory manual developed at Georgia Tech are used for these courses. Laboratory work requires the construction of digital circuits using TTL gates, PALs, and FPGAs. CAD tools including Logiworks, Espresso, AMD Mach, and Viewlogic are used in the laboratories. Simple VHDL programs are introduced to students and VHDL models are used for gate level timing simulations.

## 3 Computer Architecture I and II

Computer Architecture I and II, Computer Engineering 2510 and 3510, are required four hour, one quarter, courses taken in sequence by computer and electrical engineering students. The goals of this course sequence are to introduce students to assembly language programming, computer arithmetic, computer architecture and to provide an introduction to modern digital CAD tools. At the end of the sequence, students should have a basic understanding of computer operations from the high level language programming level to the gate level implementation of the computer system. Students are already familiar with digital design, computer programming in a high level language, Field Programmable Gate Arrays, and VHDL based modeling and simulation of simple digital logic circuits from prerequisite courses. Introduction of modern VHDL based CAD tools into the curriculum allows more realistic evaluation through system implementation and simulation than has been previously possible.

Enrollment in Computer Architecture I and II is approximately 350 students per year. Lectures are held three hours a week and three hours of laboratory work per week are required. An overview of the complete design process developed for this laboratory and the associated CAD tools are shown in Fig. 1. These tools were collected from a variety of sources and many are available free of charge to educational users via the Internet.

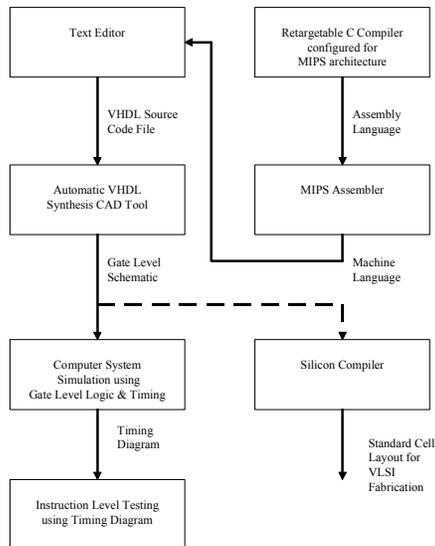


Figure. 1 Design Process and Associated CAD Tools

The lecture and laboratory outline for Computer Architecture I and II is shown in Table 1. The MIPS 32-bit RISC assembly language, integer and floating point computer arithmetic, and several hardware implementations of the MIPS processor are contained in the course textbook [2]. In Computer Engineering 2510, Computer Architecture I, students complete three assembly language programming laboratory assignments using SPIM, a MIPS assembler and instruction level simulator. SPIM is a public domain program available free via anonymous ftp for UNIX, PC and MAC platforms [2,3,4]. The first two assembly language program assignments demonstrate integer operations, control structures, stack operations, and subroutine linkage.

In the second assembly language program, students compare their hand written assembly code to the code generated by a C compiler that has compiled a program to perform the same operation. Lcc, a public domain retargetable C compiler, available free for UNIX workstations and PCs, is used for this step [5]. One of the target machines already supported by this compiler is the MIPS. MIPS assembly language source files generated by the compiler are then used by students to demonstrate compiler code generation.

In the final assembly language program, students implement a floating point function such as square root. The SPIM MIPS simulator supports floating point instructions and operations. IEEE Standard 754 floating point format is used in the MIPS processor [2,3,4].

After completing MIPS assembly language laboratory assignments, students have two laboratory assignments in which they modify a VHDL model of a

MIPS RISC processor. A synthesizable VHDL model of the MIPS was developed at Georgia Tech for use in this laboratory. It should be noted that existing synthesis tools impose a number of restrictions upon the VHDL source code and require the use of a subset of standard VHDL.

TABLE 1 Outline of Lecture and Laboratory Topics

Week	Computer Architecture I Laboratory Assignment	Computer Architecture I Lecture Topic
1	SPIM tutorial	RISC architecture
2	First assembly lang. program	MIPS instruction Set
3	"	SPIM assembly language
4	Second assembly lang. program	Stacks and subroutines
5	MIPS C compiler	Integer computer arithmetic
6	MIPS floating point program	Floating point arithmetic
7	VHDL CAD tool tutorial	Overview of VHDL
8	VHDL synthesis and simulation running a new test program	MIPS processor hardware implementation and VHDL model
9	VHDL synthesis and simulation adding a new instruction	Modifying hardware and VHDL to add new instructions and features
10	"	Alternative microcoded MIPS
Week	Computer Architecture II Laboratory Assignment	Computer Architecture II Lecture Topic
1	VHDL CAD tool tutorial	Overview of pipelining
2	Pipelining the MIPS design	Pipelining the MIPS hardware
3	"	Pipeline control problems
4	Adding a hazard unit to the MIPS	MIPS hazard detection hardware
5	"	MIPS forwarding unit
6	Adding forwarding to the MIPS	"
7	"	MIPS branch flushing
8	Add branch flushing to the MIPS	I/O devices and busses
9	"	Caches and virtual memory
10	Compile MIPS chip design	Introduction to parallel computers

In addition to a Pascal or ADA based syntax, VHDL includes several features which make it useful for modeling of digital circuits [6]. VHDL processes and modules execute in parallel. The WAIT statement suspends a process until the specified condition is true. A WAIT statement which waits for the clock signal to change is used to model and synthesize latches and flip-flops. Another VHDL feature is the AFTER keyword. The keyword, AFTER, followed by a time delay can be used to model the physical time delays present in digital circuits. These features permit VHDL to model the parallel operation of digital hardware even including physical time delays, if so desired. Behavioral VHDL based modeling of digital systems has been in use for several years. VHDL based logic synthesis is a newer development.

A simple VHDL synthesis example using the MIPS control unit is shown in Figs. 2 and 3. The VHDL source code is shown in Fig. 2 and the schematic automatically generated by the synthesis tool is shown in Fig. 3. The input to this module is the 6-bit instruction opcode, OP(5-0). The language support for bit vectors or arrays of bits, such as OP, greatly simplifies coding of the VHDL model. All busses and register values use the bit vector data type. It is possible to change the entire value of a bit vector with one assignment statement or to access smaller bit fields when needed. The outputs of this VHDL module are the various control signals used in the processor. The synthesis tool automatically performs logic minimization and multi-level logic synthesis. As an example, many of the intermediate level gate outputs are shared between the various output circuits shown in the schematic in Fig. 3. The original VHDL

code had logic equations with six inputs for the opcode bits. More than two levels of gating are required for implementation, since the logic technology used for synthesis allows a maximum of four inputs per gate.

```

-- control module (VHDL model of MIPS control unit)
library synth;
use synth.stdsynth.ALL;
entity control is
port( signal Op : in vbit_1D(5 downto 0);
      signal RegDst : out vbit;
      signal ALUSrc : out vbit;
      signal MemtoReg : out vbit;
      signal RegWrite : out vbit;
      signal MemRead : out vbit;
      signal MemWrite : out vbit;
      signal Branch : out vbit;
      signal ALUOp0 : out vbit;
      signal ALUOp1 : out vbit);
end control;
architecture behavior of control is
  signal Rformat, Lw, Sw, Beq : vbit;
  -- behavior of MIPS control
begin
  Rformat <= ((NOT Op(5)) AND (NOT Op(4)) AND (NOT Op(3))
             AND (NOT Op(2)) AND (NOT Op(1)) AND (NOT Op(0)));
  Lw <= (Op(5) AND (NOT Op(4)) AND (NOT Op(3)) AND
        (NOT Op(2)) AND (Op(1)) AND (Op(0)));
  Sw <= (Op(5) AND (NOT Op(4)) AND (Op(3)) AND
        (NOT Op(2)) AND (Op(1)) AND (Op(0)));
  Beq <= (NOT Op(5)) AND (NOT Op(4)) AND (NOT Op(3)) AND
         (Op(2) AND (NOT Op(1)) AND (NOT Op(0)));
  RegDst <= Rformat;
  ALUSrc <= Lw or Sw;
  MemtoReg <= Lw ;
  RegWrite <= Rformat or Lw;
  MemRead <= Lw ;
  MemWrite <= Sw ;
  Branch <= Beq ;
  ALUOp1 <= Rformat ;
  ALUOp0 <= Beq ;
end behavior;

```

Figure 2 VHDL Source Code for MIPS Control Unit

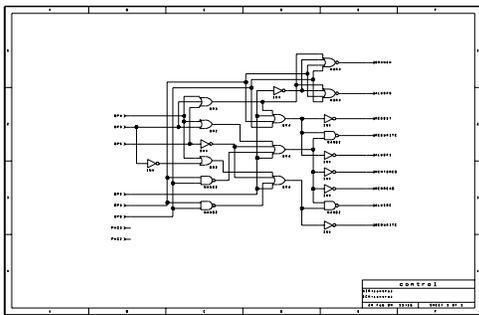


Figure 3 Control Unit Schematic Produced by VHDL Synthesis

In the two VHDL based laboratory assignments, new instructions and features are added to the VHDL model of the MIPS processor. Detailed block diagrams of the MIPS processor architecture are presented in the course textbook [2]. Changes and additions to the block diagram are made as new instructions are added to the processor. Students determine the hardware changes or additions needed and modify the appropriate source code in the MIPS VHDL model. Typical assignments have included the addition of unconditional branch instructions and unsigned add operations. The entire design is resynthesized to a gate level schematic, and a gate level timing simulation running a short machine

language test program is used to verify correct operation of the new instructions. The timing diagram from a gate level simulation of a short MIPS machine language program is shown in Fig. 4. Tutorial and batch command files are included to make the process simple enough for first time users of the CAD tools.

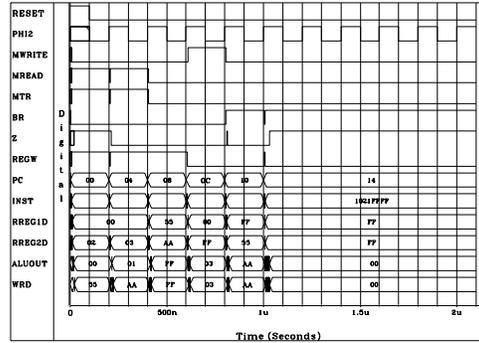


Figure 4 Timing Diagram from Gate Level Simulation

Student's VHDL models are automatically synthesized into gate level logic and simulated using a popular commercial CAD tool, Viewlogic. Several other VHDL based CAD synthesis tools could also be used such as Synopsys[7], Mentor, or Cadence; however, these are available only for UNIX platforms. Viewlogic's VHDL synthesis and simulation tools are currently available for UNIX or PC platforms at a substantial educational discount. For PC based Viewlogic, an 80486 based or faster PC with at least 16 megabytes of memory and 100 megabytes of available disk space is recommended. We currently have forty such machines available for student use in this laboratory. In addition to class handouts, several VHDL references are provided and a full set of CD-ROM on-line documentation for the VHDL CAD tool is available via a shared network drive [8,9]. The Xilinx 4000 ASIC, Application Specific Integrated Circuit, technology library is used for gate level synthesis and timing information [9]. With minor changes it is possible to resynthesize using other ASIC technologies. Gate count and timing information is automatically reported by the synthesis tool.

The VHDL synthesis tool has predefined VHDL based implementations of adders, subtractors, and flip-flops making the design process easier. Hardware for addition, subtraction, and register storage operations can be automatically generated by function calls in the VHDL code. Unfortunately, with current VHDL synthesis tools there is no universal industry standard for these functions. Since there is no VHDL synthesis standard these portions of the VHDL model would need to be modified if a different VHDL synthesis CAD tool is used.

It is possible for students to examine the synthesized schematics and to experiment with different hardware design tradeoffs such as time and area. When running simulations, a timing diagram can be produced and it is also possible to back annotate

the schematics with simulation values to aid in debugging.

A full 32-bit version of the MIPS requires several hours to synthesize on the PC platform. A smaller version with an 8-bit data path with full 32-bit instructions and control was developed to permit the design to synthesize and simulate quickly for student laboratory assignments. This version has minimal VHDL source level modifications from the 32-bit version. Changes are limited to the bit vector array size declarations for variables and signals in the data path. The smaller model requires machine language test programs that use 8-bit integer values. Since the hardware test programs in the textbook are just a few instructions long and relatively simple, this has not affected the pedagogical value of the MIPS VHDL model.

Machine language test programs can be hand assembled, assembled using the SPIM assembler, or even produced using the C compiler. This machine language file is then read into the VHDL model's memory with a read command or hardcoded into a VHDL synthesized ROM used to simulate instruction memory. On current generation PCs, the entire synthesis process for the smaller model requires several minutes of computer time. Approximately 3,500 gates are used to implement the basic MIPS VHDL model. After student modifications that add new features, the improved pipelined MIPS processor uses approximately 6,000 gates.

In Computer Engineering 3510, Computer Architecture II, students study pipelining techniques, caches, virtual memory, and I/O hardware. In laboratory assignments, students modify the synthesizable VHDL model to create an improved MIPS RISC processor with pipelining. In the first laboratory assignment, students pipeline the existing MIPS VHDL model. The VHDL model contains separate modules for each of the five MIPS pipeline stages making the student pipelining process much easier. In this assignment, students add the pipeline registers to each stage or module and modify the control unit. A top-level VHDL structural model is used to connect the five stages together. This structure generates two levels of hierarchy in the design. Breaking up the model into different pipeline stages makes the model much easier to understand and to synthesize. The textbook contains detailed block diagrams of the pipelined MIPS processor[2]. Instructions are traced one clock cycle at a time as they move through the pipeline stages and the value of all busses and control signals are shown in a detailed set of diagrams. Short machine language test programs identical to those presented in the course textbook are run in the simulator to verify correct operation. Using the simulation output, a timing diagram with bus values indicated in hexadecimal, students can check the computer's operation against the textbook's diagrams to verify correct operation. On the PC platform, the VHDL gate level timing

simulator can execute the MIPS test programs in a few seconds.

In the second laboratory assignment, automatic pipeline stalling using a hazard detection unit is added to the pipelined MIPS VHDL model. The textbook modifies the control unit to detect a data hazard between instructions in the pipeline and to stall or wait a clock cycle. An example data hazard would be an instruction in the pipeline that writes a register that is then read by the instruction that immediately follows it. The pipelined register write occurs after the read operation for the next instruction and the incorrect register data value would be used. This problem is solved by stalling and later by forwarding. These problems are carefully avoided by the MIPS machine language test programs until they are discussed in the textbook and corrected by adding additional hardware.

In the third laboratory assignment, forwarding is added between the pipeline stages to reduce the number of stalls. As outlined in the textbook, two forwarding multiplexers are added to the ALU inputs along with a more complex control unit that selects the appropriate input when it detects a data hazard. In the case of a register data hazard, the new value of the register is supplied to the ALU by the forwarding multiplexer before it is written back to the register file.

In the fourth assignment, branch flushing is added to the pipelined MIPS VHDL model. When a branch is taken several instructions after the branch are already in the pipeline and they must be automatically cleared out of the pipeline or flushed. The control unit and pipeline registers are modified to detect a branch flush operation and to clear the appropriate pipeline registers to avoid executing unwanted instructions already in the pipeline.

In the final assignment, additional features such as a cache, exception handling hardware, or a different branch instruction scheme are added to the pipelined MIPS VHDL model. On the MIPS processor exceptions, such as overflow, save the current PC in a special register and jump to a trap address. In each laboratory assignment, complete synthesis of the design to gate level, followed by a full gate level timing simulation is used to execute short machine language test programs.

Over the two quarter sequence, most students have encountered timing problems present in their VHDL models that cause errors in the VHDL gate level timing simulation. In every case, it was a real timing problem traced back to VHDL coding errors. They were able to detect and diagnose the problem using the simulation, modify the VHDL model, resynthesize, and eliminate the timing problems. Problems included uninitialized values in flip-flops, combinatorial loops, gating signals with the clock, and logic hazards on critical write control lines. These real world problems familiar to most experienced digital designers are not detected by most RTL or VHDL behavioral simulations.

Actual VLSI chip designs with higher clock rates could be automatically produced using the VHDL

synthesis generated schematic or netlist as the input to a silicon compiler [10]. The Lager silicon compiler automatically generates a standard cell layout. Interfaces to this tool have been developed for synthesized designs using Viewlogic VHDL. This tool is available upon request for UNIX platforms free of charge to educational users. In theory, a MIPS VLSI processor chip could then be fabricated using a VLSI fabrication service. The actual fabrication process would be costly and would add a substantial delay due to VLSI fabrication times. Actual fabrication of the chip would make design turnarounds difficult, if not impossible, in a one quarter course. It would also be possible to fit the entire MIPS VHDL synthesized design into a large Xilinx FPGA. In future offerings of Computer Architecture II, for the final laboratory project, MIPS processor designs will be compiled into a VLSI standard cell layout and then simulated to examine the resulting timing and area information.

#### 4 Computer Engineering Design I and II

CmpE 4500 and 4510 are required five hour design courses taken in sequence by computer engineering seniors at the Georgia Institute of Technology. Students are already familiar with digital design, FPGAs, VHDL modeling and synthesis, assembly language, C programming, operating systems, and computer architecture from earlier required coursework. Many of the students have also taken VLSI courses and a compiler class. Students work together in teams of two to four on the six month computer design project. In the first regular offering of this new sequence, students were assigned the task of developing hardware and software for a pipelined RISC processor of their own design. A total of fourteen teams worked on a wide variety of designs the majority of which were successfully implemented. The designs included several modified MIPS processors and two 2-way superscalar pipelined RISC processors.

This required two quarter course sequence is structured to enable students to apply all they have learned towards designing, documenting, and building a complete computing system. The sequence was developed with active feedback from industry, and is quite aggressive in its goals. The designs are sponsored by Intel Corporation which provides awards for the top two designs. Organizational, communication, presentation, and engineering skills are all required to be successful in this class. The introduction of this set of classes into the curriculum allows more realistic evaluation through actual system implementation than has been previously possible. Example projects, student feedback, and results from the first two offerings of this sequence are included in this paper as well as recommendations for broadening the design experience to include other implementations such as board level designs, custom chips, and real-time hardware interfaces.

#### 5 Laboratory Tools for Rapid Prototyping

The rapid prototyping methodology used in the design process in this laboratory is shown in fig. 5. VHDL synthesis [7] and a 30,000 gate hardware emulation device with automatic partitioning software are used to develop a working prototype on approximately thirty large FPGAs. A meta assembler is configured by students to produce machine language test programs. Lcc, a retargetable C compiler [5] with a code generator generator, lburg, is used to develop a cross compiler for the student processor design.

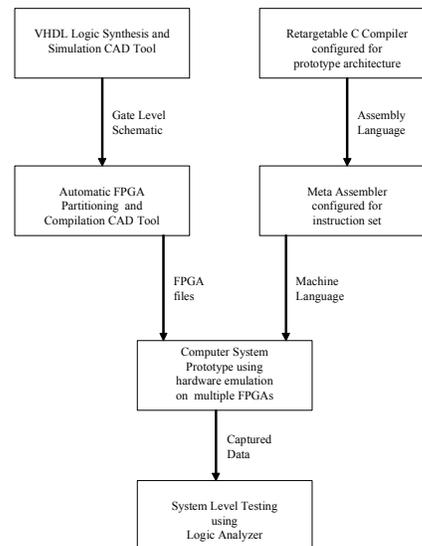


Figure 5. Rapid Prototyping Course Methodology

After VHDL synthesis, the resulting schematic or netlist is then implemented on the hardware emulation device using commercial software, ZyCAD's Concept Silicon and Xilinx's FPGA tools. This software automatically splits the design, a gate level netlist at this point, into multiple logic array chips. The hardware emulation system, a ZyCAD Paradigm RP, contains sixteen Xilinx 4010 10,000 gate FPGAs and eighteen Xilinx 3090s used to automatically interconnect the 4010 chips. A PC is used as the host machine to configure and download the hardware emulator. In theory over 200,000 gates are available for a prototype design; however, due to routing and pin out constraints designs of up to 30,000 gates are typical. On large emulation systems, it is not uncommon to obtain only ten to fifteen percent utilization of the total FPGA gate resources. The automatic interconnection using FPGAs eliminates the need for any manual wiring on the prototype system. This automatic interconnection feature makes this approach ideal for classroom projects. A number of companies produce such emulation hardware and a few have software for automatic partitioning of the design [11]. Automatic generation of the data files to

produce the prototype systems requires two to three hours of workstation time for each design. Designs can be downloaded to the hardware emulator in few minutes. Prototype digital designs of 10,000 to 30,000 gates are possible with clock rates of several megahertz. For larger designs, multiple hardware emulators can be connected together. This type of technology is widely used in industry to prototype ASICs, Application Specific Integrated Circuits, and recent microprocessor designs such as Intel's Pentium and AMD's K5 processor chips.

## 6 Computer Engineering Design Laboratory

In the first offering of CmpE 4500 and 4510 there were 30 students. Class met twice a week and the laboratory was open sixty hours a week. Remote access to workstations was available via the campus network. Students were assigned the task of designing a 32-bit RISC processor, developing a VHDL synthesizable model and simulation, prototyping the design on a hardware emulator, and developing an assembler and a compiler

TABLE 2 Outline of Lecture and Laboratory Topics

Week	Computer Engineering Design I Laboratory Assignment	Computer Engineering Design I Lecture Topic
1	VHDL Tutorial	Overview of Design Process
2	Design and Select Instruction Set	Designing an Instruction Set
3	Retarget Meta Assembler	Retargeting the Meta Assembler
4	Design Data Path	Design of Data Path Architecture
5	Develop VHDL model of Processor	Assembly of Benchmark Programs and Design Review
6	"	VHDL CAD Tools for Synthesis and Simulation
7	VHDL simulations	VHDL Instruction Level Test Simulation
8	VHDL instruction level test	VHDL Benchmark Program Simulation
9	Assembled benchmark program VHDL simulation	VHDL Synthesis of Design and Timing/Area Analysis
10	VHDL synthesis of processor timing and area reports	Design Presentation and Final Report
Week	Computer Engineering Design II Laboratory Assignment	Computer Engineering Design II Lecture Topic
1	Emulation Software Tutorial	Overview of Emulation Hardware
2	Logic Analyzer Tutorial	Overview of C Compiler
3	Compiler Retargeting and Hardware Emulation	Hardware Emulation CAD Tools
4	"	Retargeting the C Compiler
5	"	Compiler Test and Design Review
6	"	Emulation System Test and Debug
7	Hardware Emulation of Benchmark Program	Emulation System Test and Debug
8	Compilation of Benchmark Programs	Compiler Benchmarks
9	Simulation and Hardware Emulation of Compiled Programs	Running C Programs on the Hardware Emulator
10	Demonstration of Prototype Computer System	Design Presentation and Final Report

A benchmark program was proposed to compare functional designs. Metrics for the design included gate count and total execution time. Several possible instruction sets and architectures were suggested as a starting point [4,12-15].

Most students selected a MIPS instruction set and architecture since it had been studied in earlier coursework. Two groups developed their own instruction set and worked on superscalar designs. The lecture and laboratory outline for CmpE 4500 and 4510 is shown in Table 2.

In CmpE 4500, the first course in the sequence, students designed a synthesizable VHDL model of a pipelined RISC processor. Their VHDL design was then automatically synthesized into gate level logic and simulated using a popular commercial CAD tool, Synopsys. This tool has predefined VHDL based implementations of ALUs, multiplexers, and registers making the design process easier. Hardware for add, subtract and multiply operations can be automatically inferred from the symbols "+", "-", and "\*" appearing in VHDL code. The synthesis process requires several hours of workstation time for each group.

In the first two weeks of the course, students divide into groups ranging in size from two to four and design their instruction set. Then using a meta assembler, students develop an assembler for their new machine. The benchmark program is hand-compiled from C into assembly language for use in the first course of the two course sequence. At the same time as the assembler and benchmark programs are being developed, students design the data path architecture of a processor for their instruction set. Coding several benchmark programs at this point exposes any major deficiencies in the instruction set. Short machine language programs, assembled using the meta assembler, are run in the VHDL simulation to verify correct operation of the processor at the instruction level. A written report or design specification describing the instruction set and data path architecture with block diagrams is used as the basis to begin coding of the VHDL synthesis model of the processor. Most groups choose to divide the VHDL work up which each team member being responsible for a different pipeline stage. Typically each pipeline stage was a separate VHDL behavioral module. A top level VHDL structural model connects the stages together. Two to three levels of hierarchy were present in all designs. The VHDL model was then simulated and tested using short test programs produced using the meta assembler. After the simulation was successful, each group began the synthesis of the design to a gate level schematic. The netlist or schematic is then input to the automatic partitioning software which is used to divide the design between multiple FPGAs in the hardware emulator.

At midterm and the end of the quarter, each design team makes a formal presentation of their design and submits a written report. Design hierarchy was emphasized in the class as well as commenting methodology. In an industrial environment, communication of design details through well commented hierarchical hardware description languages such as VHDL is becoming very important. A news group was created to permit students and instructors the exchange ideas and tips about the various tools. Design teams were evaluated based on the correct operation of machine language instruction level test and benchmark programs, execution time and memory requirements of these benchmarks, and the number of gates used in their design. In addition

to instructor and TA input, students are encouraged to provide comments on other groups presentations and to rate the participation of other members of their group. Each team member is required to present a portion of the formal presentation.

In CmpE 4510, the second course in the sequence, students run machine language programs on the prototype machine and verify correct operation using an HP 16500 logic analysis system. The logic analyzer connections are left on predefined back panel connector pins that can be selected automatically in software by using the interconnect FPGAs with constraint files. This prevents students from needing to move the connections which would in time damage the connectors and also provides for quick changeovers from one design group to the next. It is possible to switch designs on the emulator and logic analyzer in less than five minutes. In parallel with this task, work starts on retargeting the C compiler for their machine. Lcc, a public domain retargetable C compiler available for UNIX workstations and PCs, is used for this step. Assembly language files generated by the compiler are then input to the meta assembler to produce machine language code. Machine language files are then used to run several benchmark C programs on the prototype machine.

Two formal presentations of the design and a written report is required from each group. In addition, informal design reviews with the instructor are held twice each quarter. Traces of programs running on the hardware emulator are displayed and recorded on the logic analyzer. Working computer designs must be demonstrated to the instructor or laboratory TA. Teams are also evaluated based on the correct compilation and execution time on the hardware prototype of several C benchmark programs.

The designs resulting from the first offering of this course are summarized in Table 3. Over the six month period each group's design reflects a total accumulated effort of approximately one thousand person hours. A total of nine teams completed the two course sequence. Six of the nine designs ran the required benchmark program successfully on the hardware emulator. Three designs were only partially successful. The benchmark program was a bubble sort program that sorted eight numbers in an array in memory. The sort program from [2] is a short C program that contains integer only add, subtract, and compare operations, nested loops, if statements, integer variables, array and stack operations, and a subroutine call. Thus it serves as a reasonable test of the basic operation of the compiler and the computer. Several other C programs were provided to test additional features of the compiler. Most groups divided up into a hardware design and emulation team and a compiler team for the second course in the sequence.

Two of the groups designed their own instruction set and two built a 2-way superscalar machine. They also wrote a code reorganizer to optimize the pipeline scheduling of assembly language code for their design.

These designs worked but they taxed the available computer and emulation resources to the limit.

Three of the groups' processor designs included hardware multiply and one group designed a hardware divide unit. Only small versions of the multiply and

TABLE 3 Summary of Student Processor Designs.

Machine Name	Inst.Set Arch.	Data Path Architecture	VHDL Sim	Hardware Emulation	Compiler Retargeted	Benchmark Results	Gate Count
Dueling Pipes	RISC	2-way Superscalar 16-bit Data Path 5 Stage Pipes 64 Registers	Yes	Ran Benchmark Program	Passed all test programs	357 clocks 12.34 MHz	36272(32) 8632(8)
Speed	RISC MIPS	5 Stage Pipe 32 Registers 1 Delay Slot	Yes	Ran Benchmark Program	Passed all test programs	490 clocks 6.2 MHz	21200(32) 9557(8)
Scotty	RISC VLIW	2-way Superscalar 5 Stage Pipes	Yes	Ran with minor problem	Passed main Benchmark	n/a 5 MHz	29000(32) 10853(8)
Dino	RISC MIPS	5 Stage Pipe 32 Registers	Yes	Ran Benchmark	Passed all test programs	920 clocks 5.68 MHz	25800(32) 9803(8)
Lightning	RISC MIPS	5 Stage Pipe 32 Registers	Yes	Ran Benchmark	Passed Benchmark	390 clocks 10 MHz	30100(32) 10379(8)
Melvin	RISC MIPS	5 Stage Pipe 32 Registers	Yes	ALU stage test only	Partial	827 clocks 4 MHz	28000(32) 8945(8)
Dinosaur	RISC MIPS	5 Stage Pipe no Data Forwarding 32 Registers	Yes	Ran Benchmark Program	Partial	672 clocks 4 MHz	24000(32) 9200(8)
TBM	RISC MIPS	5 Stage Pipe 32 Registers	Yes	ALU stage test only	Partial	n/a 5 MHz	19200(32) n/a(8)
Midget MIPS	RISC MIPS	5 Stage Pipe 32 Registers	Inst. Level test only	ALU stage test only	Partial	n/a	16050(32) 6067(8)

divide hardware were included on the emulator since a full 32-bit version of array multiply or divide would use most, if not all, of the 30,000 gates available on the hardware emulator. Divide hardware is not automatically produced by the synthesis tool. Due to gate count and time constraints floating point hardware was not included.

All of the groups were able to successfully develop the assembler for their design. A meta assembler definition file must be written to define the new instruction formats. This process is somewhat similar to writing macros. Once instruction formats and mnemonics are defined, an assembly language program can be written and assembled using the meta assembler. An object module in hex format and a listing file is created by the assembler. This machine language file is then read into the VHDL model's memory with a read command or hardcoded into a ROM used to simulate instruction memory. Three of the groups decided to write their own custom assembler.

The efforts to retarget the C compiler were for the most part successful. The Lcc compiler emits assembly language source files which are then assembled using the meta assembler. The machine specific features for the compiler's code generator are contained in a single machine definition file that must be modified for retargeting. Three of the groups ran numerous tests on the compiler and produced correct machine language automatically for several benchmark programs that ran on the prototype system. The remainder of the groups produced correct instruction sequences for short benchmark programs but they found it necessary to hand edit the assembly

language files prior to assembly to correct minor problems with assembly language register assignments, addressing modes, psuedo-ops and directives.

Using timing estimates from the VHDL synthesis tool and clock cycle counts from the emulation hardware, the sort program benchmark results are summarized in Table 3. The clock cycle times are based on gate delays for the Xilinx 4000 technology libraries. Clock cycle times are slower on the emulator due to programmable interconnect FPGAs and the physical dimensions of the emulator. Three of the designs were fully operational at 3.2 MHz on the emulator.

As shown in the last column of Table 3, two versions of the design were developed by most groups. A 32-bit version was run in the VHDL simulation tests. A smaller version with an 8-bit data path with full 32-bit instructions and control was developed by most groups to permit the design to synthesize quickly and to easily fit on the emulator. The limiting factor was not the total gate count rather it was the maximum number of available FPGA I/O interconnect pins. Data path models with 32-bit busses and large forwarding muxes quickly consume most of the available pins on the FPGAs in the emulator. So we quickly found that around 10,000 gates in a RISC data path design was a reasonable target instead of the maximum of 30,000 suggested for the hardware emulator. Larger hardware emulation systems which could handle the larger 32-bit designs are currently available. 32-bit designs could have been placed on the emulator at the expense of greatly increased computer and user effort. Locking the pins to pre-assigned locations for the logic analyzer also consumes additional interconnect resources. The smaller design also greatly reduced the computer requirements for each design cycle to just a few hours and made for a reasonable student project. Benchmark programs were selected to use small integer data values so that they still function correctly on the smaller models running on the prototype.

## 7 Future Enhancements

Based on our experiences in the first offerings of this course we plan several enhancements. The first offering was successful, but it would be possible to accomplish more now that we have become familiar with all of the various tools and have developed lab manuals, tutorials, and batch files for the students.

It became apparent that our current instructional computer resources were barely adequate for the task given our anticipated enrollment of around one hundred students per year. Our workstations could use more physical memory and swap space to run the CAD tools required. Currently we have a small number of UNIX workstations with 128M of memory and 375M of swap space. This is close to the minimum required and for some of the designs it was a limiting factor with our current CAD tools. Several hours of continuous workstation time are required to

synthesize the design and to map the design to the emulator. Design teams required four to six hours of workstation CPU time for each design cycle. Eight hour time slots were reserved for designs teams on two workstations. The minimal disk file space required for each design is around 150M. We are investigating several approaches to upgrade and obtain additional workstations.

Four of the groups detected timing problems that caused errors on the emulation hardware after successful VHDL behavioral simulation. In each case it was a real timing problem traced back to VHDL coding errors. They were able to detect and diagnose the problem, modify the VHDL, resynthesize and eliminate the timing problems. Problems included combinatorial loops, inferred latches, logic hazards on critical control lines, and mixing level triggered latches and edge triggered flip-flops in the same design. It is likely that a post synthesis full-timing gate level simulation would have detected these problems prior to emulation. Workstation availability and multi-vendor CAD tool interface problems prevented the widespread use of full-timing gate level simulations during the first offering of this course.

It is possible to attach external hardware to the emulation device via a back panel wire wrap connector containing 512 pins with TTL signal levels. No system clock is provided on the emulator so one was constructed at Georgia Tech. Using the back panel pins, an external variable speed clock circuit with a reset and single step switch reminiscent of old style computer front panels was connected to the special bus lines. These low skew clock bus lines are distributed directly to all Xilinx chips. The logic analyzer is used to monitor and check busses output to the back panel pins for correct operation. We plan to add external hardware such as lights and switches, a data terminal, expansion memory, and other I/O devices which would interact with the computer system and impose real-time constraints on the designs. The terminal could be easily be used to add human I/O to the C programs running on the emulator.

Once I/O devices are interfaced and functional and a small monitor or OS kernel program written in C could be ported to the design using the retargetted C compiler. A minimal set of assembly language functions for I/O would need to be rewritten and assembled using the meta assembler. A timer and interrupt hardware could easily be added to each design in VHDL to provide a time slice interrupt for multitasking.

Actual VLSI chips with higher clock rates could be automatically produced using the VHDL synthesis generated netlist as the input to a silicon compiler [10]. This process would add a substantial delay due to VLSI fabrication times and would make design turnarounds difficult if not impossible in a one quarter course. In future offerings of this course, designs will be fabricated using this technique after successful simulation and emulation.

Complex processor oriented VLSI chips fabricated using MOSIS have traditionally have had a high failure rate due to inadequate simulation and testing. MOSIS now recommends simulation and even emulation prior to fabrication. This would allow more of the design errors to be identified prior to chip fabrication. MOSIS chips fabricated from this methodology will have a higher success rate. The use of the hardware emulator in this course serves the purpose of greatly decreasing the design cycle and test time at the expense of a slower clockrate on the prototype system. Clock times of one to two orders of magnitude lower than the actual VLSI chip still permit the execution of systems level software that could not be simulated in a reasonable amount of time.

## 8 Conclusions

Rapid Prototyping provides a new paradigm for computer architecture classes. With the proper CAD tools and equipment it is possible for groups of undergraduate students to design, simulate, and develop working prototypes of complex computer systems as part of their laboratory coursework. The equipment and CAD tools required are available to schools at prices comparable to existing instructional laboratories. Students gain a sense of accomplishment and work harder on the computer design projects when given the opportunity to use real hardware rather than just a software simulation. Student and employer feedback has encouraged the continued use of new CAD tools, FPGAs and the new emulation hardware. A rapid prototyping design laboratory demonstrates and unifies many of the ideas taught in numerous undergraduate classes and serves as an ideal capstone design course for computer engineering students.

## Acknowledgments

The authors would like to thank the many students and teaching assistants who aided in the development of the new courses. Donations of workstations and logic analyzers from Hewlett Packard, Personal Computers from Intel, and CAD tools from AMD, Synopsys, Viewlogic, Xilinx, and ZyCAD made the development of these new laboratories possible

## References

- [1] R. H. Katz, *Contemporary Logic Design*, Redwood City, CA: Benjamin/Cummings Publishing, 1995.
- [2] D. Patterson, D. and J. Hennessy, *Computer Organization & Design The Hardware/Software Interface*, San Mateo, CA: Morgan Kaufmann, 1994.
- [3] J. Larus, *Assemblers, Linkers, and the SPIM Simulator*, University of Wisconsin, 1991. Reprinted in [1], manual and SPIM software available via ftp from ftp.cs.wisc.edu.
- [4] J. Heinrich, *MIPS R4000 User's Manual*, Englewood Cliffs, N.J.: Prentice Hall, 1993.
- [5] C. Fraser, C and D. Hanson, *A Retargetable C Compiler: Design and Implementation*, Redwood City, CA: Benjamin/Cummings Publishing, 1995. Information and compiler source code available from <http://www.cs.princeton.edu/software/lcc>
- [6] P. Ashenden, *The VHDL Cookbook*, University of Adelaide, 1990. Available via ftp from bears.ece.ucsb.edu.
- [7] P. Kurup, and T. Abbasi, *Logic Synthesis Using Synopsys*, Boston, Mass: Kulwer Academic Publishers, 1995. Synopsys information available from <http://www.synopsys.com>
- [8] *VHDL Reference Manual for Synthesis*, Marlboro Massachusetts: Viewlogic Systems, 1992. Viewlogic information available from <http://www.viewlogic.com>
- [9] *XACT Viewlogic Interface User Guide*, San Jose, CA: Xilinx Corporation, 1994. Xilinx information available from <http://www.xilinx.com>
- [10] R.W. Brodersen, editor., *Anatomy of a Silicon Compiler*, Boston, Mass: Kulwer Academic Publishers, 1992. Lager information available from <http://infopad.EECS.berkeley.edu/research/tools/lager>
- [11] H. Owen, U. Khan, and J. Hughes, "FPGA Based Emulator Architectures", *More FPGAs* (Will Moore and Wayne Luk, Eds.), Abbingondon, England: Abbingondon EE & CS Books, pp. 399-409 (1994). Emulator Information available from <http://www.zycad.com> and <http://www.quickturn.com>
- [12] D. Patterson, and J. Hennessy, *Computer Architecture A Quantitative Approach*, San Mateo, CA: Morgan Kaufmann, 1990.
- [13] R. L. Sites, ed, *Alpha Architecture Reference Manual*, Bedford, Mass: Digital Press, 1993.
- [14] D.L Weaver, and T. Gemond eds., *The SPARC Architecture Manual*, Prentice Hall, Englewood Cliffs, H.J. 1994.
- [15] S. Wiess and J. E. Smith, *Power and PowerPC: Principles, Architecture, Implementation*, San Mateo, CA: Morgan Kaufman, 1994.