MiniMIPS: *An 8-Bit MIPS in an FPGA for Educational Purposes*

Cesar Ortega-Sanchez
Electrical and Computer Eng. Dept.
Curtin University
Perth, Western Australia
c.ortega@curtin.edu.au

***Abstract*—Expectations of Electrical Engineering students about their courses have changed over the years. Even though the basic principles of Digital Electronics remain the same, tools and laboratory activities need to accommodate more demanding expectations. This paper presents MiniMIPS: an 8-bit implementation of the MIPS's single-cycle architecture for educational purposes. The MiniMIPS is targeted to the BASYS Spartan 3E development board. The user interface consists of DIP switches, push-buttons, LEDs and four 7-segment displays. The instruction set consists of 9 instructions and 3 instruction formats. Programs for the MiniMIPS are developed in a custom-made assembler/simulator tool, also presented in this paper. A MiniMIPS assembly program to generate the Fibonacci series is presented as an example. A description of laboratory tasks currently used is offered.**

Keywords-Soft cores; MIPS architecture; embedded systems; FPGA applications; engineering education.

I. INTRODUCTION

Very few disciplines have evolved so much in a short period of time as Digital Electronics. The first working transistor was demonstrated at Lucent Labs in 1947. Sixty-four years later, microprocessors contain hundreds of millions of transistors in a single integrated chip; a major achievement. Computer architecture has also evolved very rapidly to exploit the advantages of ever-shrinking transistors and satisfy the demand for faster, more powerful computers. This evolution has promoted the arrival of strategies and techniques such as pipelining, cache management, multithreading, parallelism and heterogeneous chip multiprocessors, to mention just a few [1].

Despite the rapid evolution of computer architecture, university students still learn the basics of digital design and computer architecture following a syllabus that has changed little over the years. After all, the basics remain the same. However, tools and methodologies have evolved dramatically to cope with the ever-increasing complexity of modern electronic systems. Twenty years ago prototypes were built using discrete integrated circuits and breadboards. Nowadays, the complexity of most real-world digital systems makes them impossible to prototype using discrete components. The modern digital electronics designer needs to be current with software and hardware tools to verify and test complex systems. To satisfy this need, simulators and Field-Programmable Gate Arrays (FPGAs) must be at the frontline in the modern digital electronics laboratory.

Almost any book on digital design and computer architecture covers the theoretical part of most Digital Electronics courses. The practical aspects of the course, however, may differ greatly from one university to the next. In some faculties the emphasis may be on simulation using EDA (Electronic Design Automation) tools; while in other, a more hands-on approach may be preferred. It is then in the Digital Electronics laboratory where a great variety of approaches is usually found. At Curtin the prescribed book is [2] and an intensive hands-on approach with emphasis on FPGA technology is favored.

II. FPGAS IN THE DIGITAL ELECTRONICS LAB

Nowadays there is a multitude of FPGA development boards available from manufacturers and third-party companies [3, 4, 5]. These boards offer the same advantages as development boards for other technologies: low-cost, good support, a community of users sharing experiences, sample applications, etc. FPGAs are not committed to particular applications; they can be configured to implement a wide range of designs, from the demonstration of 2-input logic gates to complete microcontroller-based embedded systems. FPGA development boards are the ideal prototyping platform to support the digital design laboratory. Never before have Computer Engineering students had access to such a versatile tool.

This versatility gives academics freedom to create more challenging and, why not, more entertaining laboratory tasks. This last point is important if one takes into account that university students' expectations have also evolved on par with technology. Young people are so familiar with high-tech gadgets that a 1980's electronics laboratory would seem to them boring and uninteresting. The elements of novelty and challenge of FPGAs have the potential to quench the thirst for novelty of even the most demanding student, without sacrificing academic rigor.

To support the teaching in computer architecture a modified version of the MIPS presented in [2] has been developed at Curtin. The MiniMIPS architecture allows the implementation of the concepts covered in the book using the BASYS FPGA development board by Digilent [6]. The MiniMIPS and its associated software development tool have transformed Electrical Engineering students' understanding of computer architecture and how programs get executed by a processor.

978-0-7695-4551-6/11 $26.00 © 2011 IEEE
DOI 10.1109/ReConFig.2011.62

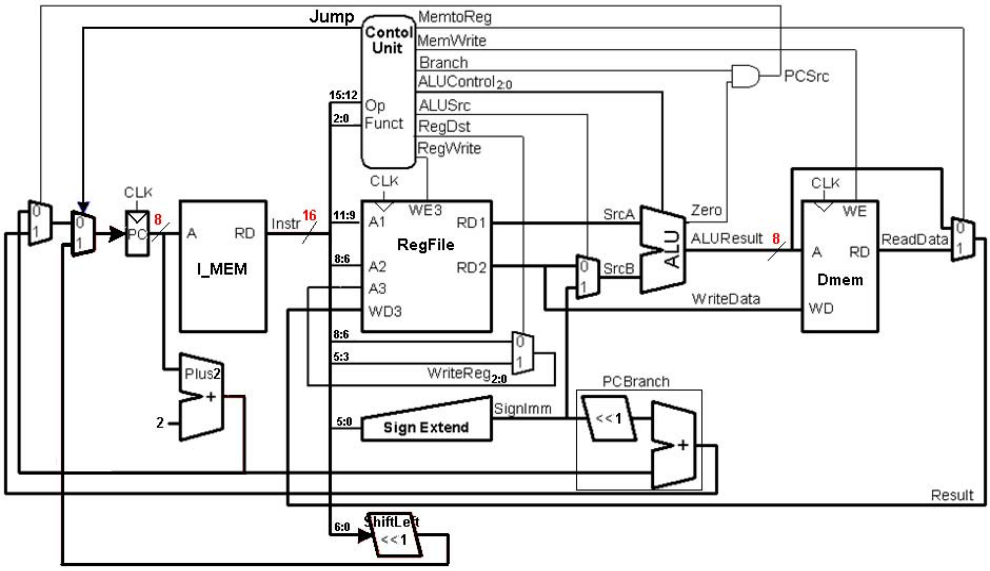


Figure 1. MiniMIPS datapath and control unit.

III. MINIMIPS ARCHITECTURE

The MIPS is a 32-bit, RISC processor invented by John Hennessy from Stanford University in the 1980s. It is used as a working example to explain microarchitecture principles in [2], Ch.7. In the MiniMIPS the original architecture was modified to make full use of the peripherals present in the BASYS FPGA development board. Table I shows the features of the MIPS and MiniMIPS processors.

A detailed description of the original MIPS processor can be found in [7]. Figure 1 shows the datapath and control unit of the MiniMIPS. The simplicity and elegance of this processor help students understand the basic principles of computer architecture.

In one of the laboratory assignments students are asked to describe all modules in Figure 1 in VHDL. Individual modules have to be fully simulated before simulating the complete processor. The MiniMIPS is then integrated into a high-level project where inputs and outputs are mapped to peripherals in the BASYS board. Once the configuration file has been loaded to the BASYS board, students can control and monitor the execution of programs in the MiniMIPS.

All MiniMIPS instructions are 16-bit wide. The original MIPS instruction formats were modified to fit in 16 bits. Figure 2 shows the modified instruction formats of the MiniMIPS.

TABLE I. COMPARISON OF MIPS AND MINIMIPS COMPUTERS

Feature	MIPS	MiniMIPS
Data bus width	32 bits	8 bits
Address bus width	32 bits	8 bits
Instruction length	32 bits	16 bits
Number of registers	32	8
# instructions	65	9 (expandable)
# clock cycles per instruction	1	1
Instruction formats	Register, Immediate, Jump	

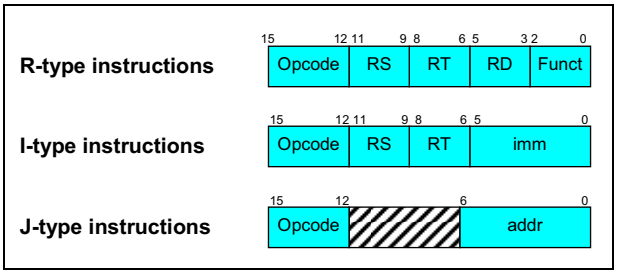


Figure 2. MiniMIPS instruction formats

The MiniMIPS has 8 registers, hence 3 bits are sufficient to encode register operands and destinations.

Immediates are 6-bit signed constants (-32 to 31).

The number of bits for the opcode was reduced from 6 to 4. Bits indicating the function in register operations were reduced from 6 to 3. Table II presents the instruction set of the MiniMIPS.

In the current implementation of the MiniMIPS only 9 instructions have been considered. However; students are taught how to incorporate new instructions by modifying the data path and control unit [2] p.377.

TABLE II. MINIMIPS INSTRUCTION SET

Assembly Instruction	Instr. Format	Opcode/ Funct	Operation
add $rd, $rs, $rt	Register	0000/000	$rd = $rs + $rt
sub $rd, $rs, $rt	Register	0000/010	$rd = $rs - $rt
and $rd, $rs, $rt	Register	0000/100	$rd = $rs AND $rt
or $rd, $rs, $rt	Register	0000/101	$rd = $rs OR $rt
addi $rt, $rs, imm	Immediate	0100	$rt = $rs + imm
lw $rt, imm($rs)	Immediate	1011	$rt = M[$rs + imm]
sw $rt, imm($rs)	Immediate	1111	M[$rs + imm] = $rt
beq $rt, $rs, imm	Immediate	1000	if ($rs == $rt) then PC = PC + imm * 2
j address	Jump	0010	PC = address * 2

TABLE III. MINIMIPS REGISTER SET

Name	Address	Use
$0	0	Constant 0. Read Only
$r1	1	General purpose register
$r2	2	General purpose register
$r3	3	General purpose register
$r4	4	General purpose register
$r5	5	General purpose register
$r6	6	General purpose register
SWs	7	8-bit value from switches. $rs or $rt. Read only
LEDs	7	8-bit value for LEDs. $rd. Write only

Due to the limited number of bits available to encode register addresses, the number of registers in the MiniMIPS has been reduced from 32 in the original MIPS to 8. Register 7 is used to access LEDs and switches from the BASYS board (marked as ***A*** and ***C*** in Figure 3). For compatibility with the MIPS, register 0 has a constant value of 0x00. All other registers are general purpose. Table III shows the set of registers in the MiniMIPS and how they are used.

IV. PROGRAMMING THE MINIMIPS

Programs for the MiniMIPS are written in assembly language. An editor and simulation tool was developed to support students in writing and verifying their programs. The tool offers the following functions:

Context-sensitive code editor. To write programs students select instructions from a drop-down menu and then the operands are validated accordingly, hence avoiding mistakes in typing or in the order of operands.

Save and load programs. Programs are stored as text files with extension .mips.

Assembly code simulator. Programs are simulated on the screen for verification. The content of registers, memory and I/O devices changes dynamically as instructions get executed.

Export to VHDL. Once programs have been verified in the simulator, the export button creates a VHDL description of the MiniMIPS instruction memory. This module has to be included in ISE's MiniMIPS project. To test the program in the BASYS board, the project is re-synthesised and downloaded. Figure 4 shows a screenshot of the MiniMIPS simulator.

V. USING THE MINIMIPS

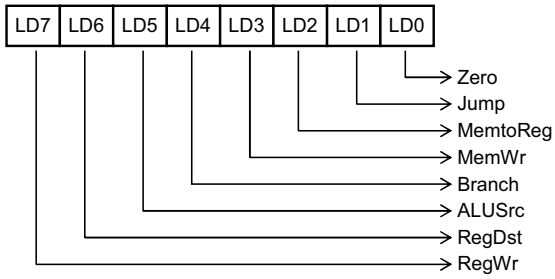
The system is controlled with the 4 push-buttons available on the board (marked as ***D*** in Figure 3). The function of each button is as follows:

Button 3 (BTN3)- Clock for the system. Every time BTN3 is pressed one clock pulse is sent to the processor. A debouncing circuit has been added to prevent the generation multiple clock cycles with a single push.

Button 2 (BTN2)- Pressing BTN 2 selects the display of the following signal values on the four 7 displays (marked as ***B*** in Figure 3) in sequence:

- Program Counter
- Instruction
- Operands A and B
- ALU result

Button 1 (BTN1)- Selects the content of the LEDs. If not pressed, the LEDs show the content of the LEDs register. When pressed, the LEDs present the control signals generated by the Control Unit (see Figure 1), as follows:



- LD7- ***RegWr***. Signal that enables writing in the register file.
- LD6- ***RegDst***. Signal that selects what bits in the instruction determine the destination register.
- LD5- ***ALUSrc***. Signal that selects between a register and a sign-extended immediate as operand B for the Arithmetic and Logic Unit (ALU).
- LD4- ***Branch***. This signal is asserted when the *Branch if Equal* instruction (*beq*) is executed. It is AND-ed with the ALU's Zero flag to select the address that will be loaded in the Program Counter (PC in Figure 1) on the next clock cycle.
- LD3- ***MemWr***. Signal that enables writing in the Data Memory.
- LD2- ***MemtoReg***. Signal that selects between the Data Memory and the ALU as the source for the destination register on the next rising-edge of the clock.
- LD1- ***Jump***. This signal is asserted when the Jump instruction (*j*) is executed. It selects the address that comes in the instruction as the next value to be loaded in the PC.
- LD0- ***Zero***. This signal is asserted when the output from the ALU is equal to 0x00.

Button 0 (BTN0)- Reset of the system. When pressed,the program counter is reset to 0x00.

Figure 3 shows the position of peripherals on the board.

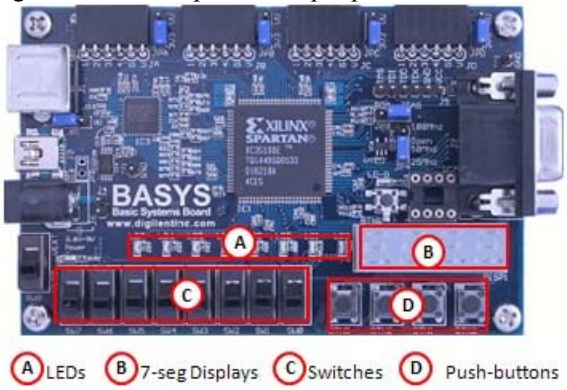


Figure 3. BASYS board

There are two independent registers associated to the address of register 7. When register 7 is used as an operand, the binary value set on the switches is used. When register 7 is used as destination, the LEDs display the content of this register.

The MiniMIPS and all its peripherals occupy 4% of the FPGA on the BASYS board, leaving 96% of reconfigurable logic available for MiniMIPS expansion or custom peripherals. The device utilization summary is as follows:

Selected Device :	3s250etq144-5	
Number of Slices:	115 out of 2448	4%
Number of Slice Flip Flops:	19 out of 4896	0%
Number of 4 input LUTs:	216 out of 4896	4%
Number used as logic:	168	
Number used as RAMs:	48	
Number of IOs:	74	
Number of bonded IOBs:	74 out of 108	68%
Number of GCLKs:	1 out of 24	4%

VI. EXAMPLE: THE FIBONACCI SERIES

As an example of the MiniMIPS application, the development of a program to generate the Fibonacci series is presented next.

The Fibonacci Series is a sequence of numbers first created by Leonardo Fibonacci in 1202. It is a deceptively simple series, but its ramifications and applications are nearly limitless [8]. The first two numbers in the series are zero and one. To obtain each number of the series, the two numbers that came before it are added. In other words, each number of the series is the sum of the two numbers preceding it.

$$f(n)= f(n-2) + f(n-1) \quad \text{for all } n>1 \qquad (1)$$

Where f(0)=0 and f(1)=1.

The flowchart in Figure 4 shows the algorithm to generate the first 20 numbers in the Fibonacci series. In this program the series is stored in the data memory, starting at location 0x00. Numbers in the series are displayed in the LEDs of the board. This is a good example because the program produces overflow and students need to explain it.

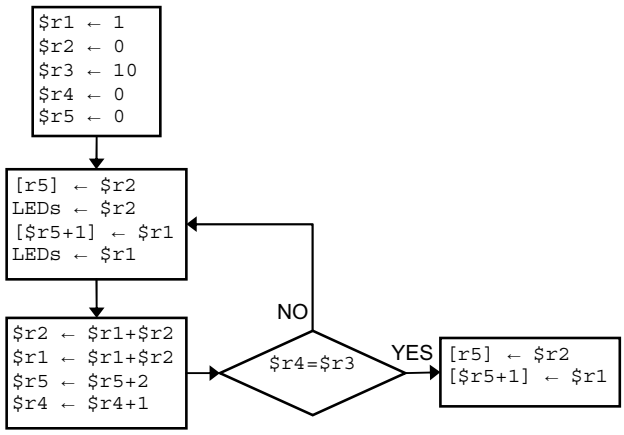


Figure 4. Algorithm to generate the first 20 numbers in the Fibonacci series

Figure 5 shows the MiniMIPS assembly code and simulation results for the Fibonacci series generator.

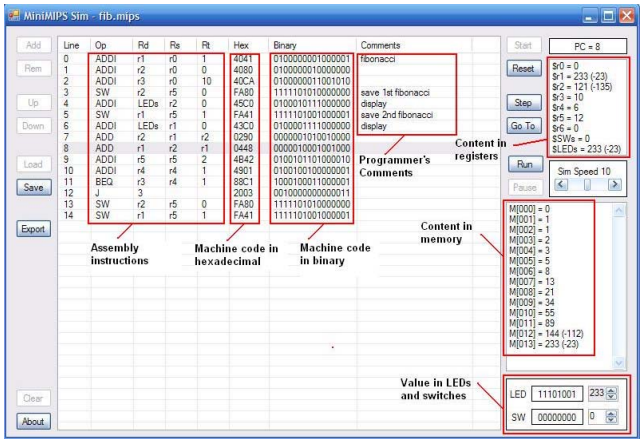


Figure 5. Assembly code and simulation of Fibonacci series generator

Figure 6 shows the MiniMIPS's instruction memory VHDL code generated by pressing the ***Export*** button in the assembler and simulation tool.

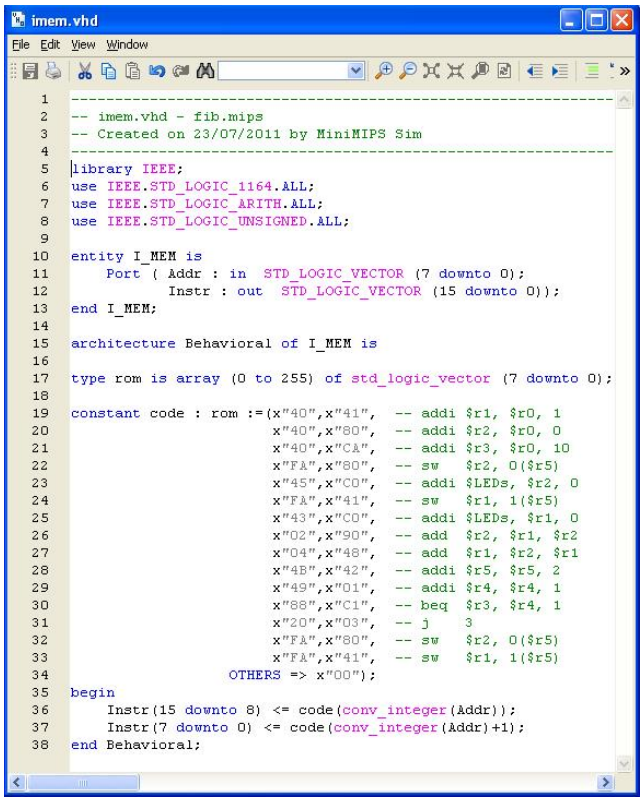


Figure 6. VHDL code of the MiniMIPS instruction memory

To test the program on the BASYS board, students add this VHDL file to their ISE projects and then re-generate and download the FPGA's configuration file.

VII. Conclusions and Future Work

Modern FPGAs allow fast prototyping of complex systems, making them excellent educational tools in the Digital Electronics Laboratory.

An 8-bit implementation of the MIPS microprocessor suitable for demonstration in the BASYS development board, and its assembler/simulation tool has been presented in this paper. A Fibonacci series generator was presented as an example of the problems students can solve using the MiniMIPS assembly language tools.

A description of the laboratory scripts used at Curtin University to teach the foundations of digital design is provided to make a case for the use of field-programmable logic devices in the digital electronics lab.

Describing and programming the MiniMIPS offer students the opportunity to learn in an interactive way the relationship between software and hardware. Also, learning VHDL and the ISE tool exposes students to the way modern digital systems are designed, verified and tested in industry.

A collection of resources including: lecture notes, all laboratory scripts described in this paper, VHDL source files describing the implementation of the MiniMIPS in the BASYS board, and assembler/simulation tools are available for educational purposes by request to Dr Ortega-Sanchez (c.ortega@curtin.edu.au).

References

[1] Jouppi N., "The Future Evolution of High-Performance Microprocessors", Hewlett-Packard Development Company, 2005. Last accessed 15/03/09 http://pact04.ac.upc.es/micro38/01_keynote2.pdf

[2] Harris D. and Harris S. (2007) "Digital Design and Computer Architecture", 1st Edition, Morgan Kaufmann. ISBN: 978-0-12-370497-9

[3] ALTERA web site. Last accessed 21/06/2011 http://www.altera.com/products/devkits/kit-dev_platforms.jsp

[4] XILINX web site. Last accessed 21/06/2011 http://www.xilinx.com/products/boards-and-kits/index.htm

[5] DIGILENT web site. Last accessed 21/06/2011 http://www.digilentinc.com/Products/Catalog.cfm?NavPath=2,400&Cat=10&FPGA

[6] BASYS development board web site. Last accessed 20/06/11 http://www.digilentinc.com/Products/Detail.cfm?NavTop=2&NavSub=457&Prod=BASYS

[7] MIPS Technologies web site. Last accessed 22/06/2011. http://www.mips.com/products/architectures/mips32/

[8] http://library.thinkquest.org/27890/. Last accessed 20/07/2011.

Appendix: Laboratory Teaching Materials

At Curtin University, Electrical and Computer student engineers start using FPGAs and VHDL in the unit Foundations of Digital Design 201 (FDD201) during the first semester of their second year. FDD201 covers the following topics:

- Binary Systems and logic gates
- Minimisation techniques
- Design of combinational circuits
- Positional numeric systems
- Computer arithmetic
- Design of sequential systems
- Hardware Description Languages
- Registers, memories and programmable logic
- Architecture and instruction sets
- Single-cycle processor
- Programming
- Commercial architectures
- Advanced architectures

Semesters at Curtin consists of 12 teaching weeks. In FDD201 students attend one 2-hour laboratory session, or "Lab", per week. Labs start on week 2 and finish on week 11. In week 12 students who either missed one of the labs or want to improve their marks in one of the assignments have the opportunity to re-do one lab. A brief description of FDD201 Labs is presented next.

Lab 1- Logic Gates or "Welcome to the 80's"

In this assignment students have to implement small combinational designs using integrated circuits (ICs) from the 74XX family, breadboards, and wires. The objective is to expose them to how prototypes were built in the 1980's so that they can reflect, after they have used the FPGA board, on how much the technology has evolved in the last 25 years. For most students this is the first time they have to use a breadboard. Some like the experience, but others get very frustrated when their circuits do not work because of a loose wire, or because they forgot to connect the integrated circuits' VCC and GND pins to the power supply.

Lab 2- Xilinx's Tools or "Back to the Future"

In this lab students learn how to use Xilinx's ISE design suite. They are told that if breadboarding was the way to go in the 1980's, then FPLDs are the way to implement prototypes in the 21st century. As pre-lab work, an "ISE quick reference guide" specifically prepared for this assignment has to be studied. The document summarises in 13 pages all the steps involved in the implementation, simulation and test of a small combinational circuit; from project creation to downloading of configuration files to the FPGA. The objective of the assignment is for students to gain confidence using ISE. For this purpose they have to open an existing project and download the configuration file to the FPGA board. Then they have to modify the original schematic diagram by adding more components, and generate the new configuration file for testing in the FPGA.

Lab 3- Decoders and Multiplexers

In this assignment students start using hierarchical design to implement combinational circuits. First they create a project in ISE and use a schematic diagram to implement, simulate and test a 2-to-4 decoder. Then they use two instances of the decoder to implement, simulate and test an 8-to-1 multiplexer. Finally, students have to design, simulate and test a 4-input, 1-output combinational circuit to solve a problem stated in English. In their designs they must use only the 8-to-1 multiplexer and one inverter.

Lab 4- Positional Numeric Systems

In this assignment students are chief engineers of an intergalactic expedition exploring our galaxy. In their travels they find a planet inhabited by little beings whose two hands have 2 fingers each. In order to help them advance their technology they have to design a device that adds together two 2-bit numbers and presents the result in the 7-segment displays of their prototyping FPGA board. Digits are entered using slide switches to represent their binary codes.

Lab 5- Arithmetic and Logic Unit

Students have to design, simulate and test a complete 4-bit Arithmetic and Logic Unit (ALU) capable of performing 4 arithmetic (addition, subtraction, increment, decrement), and 4 bitwise logic functions (AND, OR, XOR, NOT). The design must follow a hierarchical approach (nesting of components). The arithmetic part can be implemented using full-adders as the basic component and then the arithmetic operations can be achieved by modifying the second operand of the full adder according to the selection inputs. The logic part can be implemented with one multiplexer and logic gates. Students are free to explore their own designs as long as they follow the hierarchical approach.

Lab 6- Sequential Systems: The Traffic Lights

Students need to design, simulate and test the control for a set of traffic lights in a crossroad. One of the roads is a major highway and the other is a small street, hence, cars circulating on the highway must have preference. To achieve this, the small road has a sensor that detects when there is a car waiting to cross. Conveniently, the FPGA board has red, yellow and green LEDs which add an increased level of realism to this lab. In this session students use a clock for the first time. The on-board clock is 100 Mhz, hence students need to design a clock divider to get a 1 Hz frequency. The state machine for the controller must change state on every clock cycle.

Lab 7- Implementing Designs Using VHDL

Lab 7 is an introduction to how to create, verify and test designs using VHDL. In this introductory lab students are asked to analyse, simulate and test in the BASYS board selected VHDL descriptions. The objective is for students to familiarise with VHDL syntax and acquire a sense of confidence that will allow them to attempt their own designs in the following labs.

The designs to test are:

- 4-bit ALU to demonstrate combinational circuits.
- Moore state machine to demonstrate sequential circuits.
- 16 x 4-bit RAM to demonstrate how memories work.
- 4 bit multiplier to demonstrate the use or arithmetic libraries.

All designs are constrained to 4-bits to allow testing using the switches and lights available on the board. However, students are invited to reflect on how easy it would be to change designs in VHDL.

Lab 8- Counters

Students have to design a 4-bit binary counter that gets incrementally more complex. First, the counter do not use any external inputs, it simply cycles through an ascending sequence. Then, an input is added to alternate the sense of counting between ascending and descending. Finally, a new input is added and the counter should go through 1 of 4 different sequences, selected by the code presented in the two inputs. The objective of this lab is for students to realise how efficient their work becomes when their designs are described in VHDL. This would be a very long assignment if schematic diagrams were used to describe every design.

Lab 9- The MiniMIPS Architecture

In this lab students are given an incomplete VHDL hierarchical description of the MiniMIPS described in this paper and they are asked to complete the design. Students describe and simulate individual modules before integrating them into the MiniMIPS description. The completed MiniMIPS is then simulated. Students are asked to observe and describe the behaviour of every component of the MiniMIPS during the execution of instructions in a sample program.

Lab 10- Programming the MiniMIPS

This lab is a follow up for lab 9. Students are asked to use the MiniMIPS assembler/simulation tool described in this paper to write and simulate a set of simple programs. In their programs students have to implement programming structures like loops and arrays using MiniMIPS assembly instructions.

It is expected that students should do pre-lab work for a minimum of two hours before attending every lab session; however, even though students are reminded that digital design is 80% thinking and 20% implementation, some of them still decide to do the thinking during the lab. These students very rarely finish all the required tasks.