EE108B Prof. Dally
Winter 2010-11

# EE108B Lab Assignment #2
# Processor Datapath Design
# Due: Tue, Feb 8th

## 1. Introduction

Now that you have seen some of the benefits of the software approach to problems, we will spend the next three labs building a processor that is capable of executing MIPS instructions. In this lab, you will complete an implementation of a single cycle processor, so that you can run your program from Lab 1.

## 2. Requirements

You will be completing the MIPS processor implementation using the starter files provided to you. This will involve editing parts of the instruction fetch unit, instruction decoder, and ALU. In lab, TAs will run a test script on your processor to verify that it handles all instructions properly. In addition, you will need to run and show a demo of your Pong program from lab 1 on this processor.

## 3. Implementation Details

*Processor Model*

Before you start coding, look at each of the modules we have provided and try to understand what each one of them does. Ask one of the TAs if you have any questions about the model.

- MIPS.v - top level module
- MIPStest.v - Testbench (for simulation purpose alone)
- IF.v - fetches the current instruction from memory, and calculates the next PC address
- Regfile.v - reads the two input registers, and writes to the output register
- Decode.v - decode the instruction and determine control signals for rest of processor
- ALU.v - datapath for arithmetic/logic operations
- MemStage.v - interface with data memory and memory-mapped devices, and determine what data should be written back into the register file
- Sega_Ctl.v - memory-mapped gamepad with address 0xFD
- VGA_Ctl.v - memory-mapped VGA screen with address 0xFF
- DVI_Controller_Top.v , sync_generators.v, chip_data_parser.v - VGA to DVI conversion

- I2C_*.v – I2C module to initialize DVI controller on board
- irom files – Instruction memory initialized with the program
- framebuffer files – Framebuffer is used to store the colors/pixels to be displayed on the monitor
- tcgrom files – Pixels for char display (useful in debug mode)
- dataram files – Data Memory
- MIPS.ucf - Maps the signals from the top level module to pins on board
- asm108b, rename.awk - Useful for conversion of .s file into .coe file (memory initialization)

*Getting Started*

You will primarily be using two software tools in this project - ModelSim for simulation purpose and Xilinx ISE for FPGA synthesis, place and route. You might also want to use CoreGenerator in Xilinx ISE to generate memory files required for simulation and FPGA synthesis.

Before using the tools, you need to execute the following command from your UNIX terminal window in your home directory. This has to be executed only once for the labs. This is used to setup your environment to run the above tools.

```
echo source /usr/class/ee108b/ee108.cshrc >> .cshrc
```

Modelsim is aliased to vsim and Xilinx ISE is aliased to ise. It is recommended to create a separate folder for the project work.

Verilog Code

You will need to complete the modules IF.v, Decode.v, and ALU.v. Note that you will not need to add any new wires or registers. You will only be making assignments to the provided ones. Adding new wires or regs will only make things more complicated. This will make debugging your code a lot easier.

The processor we have provided will increment the program counter by 4 after each instruction, but does not allow for branches or jumps. In IF.v (check for the indication), calculate the program counter in the event of jumps or branches, and determine how to assign the next PC.

In Decode.v, you will have to determine which ALU operation should be performed for each instruction listed below. In many cases, the answer will be obvious (for the ADD instruction, select the add operation), but in some cases it will not be. You will also be calculating the 32-bit extended immediate value, which varies depending on the instruction. The instructions to decode are:

| | | | | | |
|---|---|---|---|---|---|
| ADD | ADDU | ADDI | ADDIU | SUB | SUBU |
| SLT | SLTU | SLTI | SLTIU | AND | ANDI |
| OR | ORI | XOR | XORI | NOR | SRL |
| SRA | SLL | SRLV | SRAV | SLLV | LW |
| SW | BEQ | BNE | BLTZ | BLEZ | BGTZ |
| BGEZ | LUI | | | | |

In ALU.v, you will have to calculate the result for the different operations you have specified in Decode.v. Once again, some results will be straightforward to implement, while others will require more thinking.

*Simulation Instructions:*

In Modelsim, you can create a new project and add all the .v files from the starter kit. Make sure you have the tcgrom, framebuffer, dataram, irom files in the same directory as your ModelSim project. In order to recognize Xilinx Core generator files in ModelSim, type the following commands in Modelsim command line.

```
vmap unisims_ver afs/ir.stanford.edu/class/ee/mentor/vsim66/modeltech/xilinx_libs/unisims_ver
vmap xilinxcorelib_ver
afs/ir.stanford.edu/class/ee/mentor/vsim66/modeltech/xilinx_libs/xilinxcorelib_ver
```

Also, to see the signals so that you can add them to the wave forms, issue the following command as well:

```
vsim -L xilinxcorelib_ver -L unisims_ver -t 1ps {-voptargs=+acc=bcglnprst -O0} work.MIPStest
```

*Generating .coe files:*

irom.v contains the instruction memory, which must be loaded with your MIPS program. Use Core Generator to specify a .coe file as the init file for the irom memory. After loading a new program, you will have to re-synthesize your project. Note that you need to generate new memory files only for irom.

EE108B Prof. Dally
Winter 2010-11

To generate a .coe file, copy the scripts asm108b and rename.awk from the website into your AFS working directory (you are advised to use a different folder for this purpose). Then, type the command:

```
./asm108b pong.s rename.awk // produces pong.coe file
```

You may then use Coregen utility associated with Xilinx ISE to generate memory blocks using the .coe file. You can open the CoreGenerator tool from the Tools menu in Xilinx ISE. Navigate to your Verilog project directory using the Browse button and then click OK. In the left window, expand the Memory & Storage Elements tree, click on RAMS &ROMS. Check the "All IP Versions" box. And double click on the **Block Memory Generator Version 3.3**. In the new window that appears type irom for Component Name, select Single Port ROM, 32 bits for Width and enter the number of lines (recommended 512) in your assembly code for Depth. In page 3, check Load Init File, and Click on Load File. Navigate to the .coe file (generated using asm108b and rename.awk) and click OK. Finally click Generate. If all goes well, a message would pop up informing you that the memory block has been successfully generated, and the generated irom.xco file can be found in your project directory.

*Synthesizing the FPGA configuration (.bit) file:*

Create a new project in Xilinx using the following settings:

Family: Virtex5

Device: XC5VLX110T

Package: FF1136

Speed grade: -1

Top-Level Source Type: HDL

Synthesis Tool : XST (VHDL/Verilog)

Simulator: ISim (VHDL/Verilog)

Preferred Language: Verilog

Property Specification in Project File : Store all values

In Xilinx ISE project, add all of the .v files, the mips.ucf file and .xco memory block files into your project. In order to generate the .bit file, double click on "Generate Programming File".

### *Hardware implementation*

After you have simulated the design, you will need to download your design onto the FPGA. We have provided the file mips.ucf, which contains the pin assignments for the inputs and outputs. You will be using the push buttons on the Xilinx board to control your paddle(s), and you should be able to work out the button-control correspondence using the Xilinx FPGA manual and the ucf file. When you are ready, download your *.bit onto the FPGA using iMPACT.

When you run the processor in hardware, there are two display modes and two speed modes, controlled by DIP switches (sw1 and sw2) on the Xilinx Virtex5 board. For the display, toggle switch number 1 of the DIP switch would allow you to toggle between the screen from lab 1, and a debugging screen which displays the current PC, instruction, ALU operands, and instruction result. For the speed, toggle switch number 2 of the DIP switch would allow you to either run the program at full speed or step through one instruction at a time using the center push button on the Xilinx board.

Refer to the .ucf file to find out which buttons on the Xilinx board correspond to the gameconsole buttons for A-B, Select etc.

### *Extensions*

If you have completed the above requirements, you may choose to implement one or more of the following extensions:

- Overflow detection - Provide an output from ALU.v that is asserted whenever an operation results in overflow (be careful to distinguish between signed and unsigned operations). Use one of the LEDs on the Xilinx board to display when overflow has occurred. You do not have to worry about overflow affecting subsequent instructions.
- Implement the instructions BGTZAL and BLEZAL - See Appendix A in the textbook for descriptions of these instructions.

## 4. Write-up and Submission

You must submit a write-up electronically. Please refer to the guidelines posted on the eeclass website.

Be sure to explicitly state what extensions you added. Additionally, in your conclusion, please answer the following questions:

1. According to the Xilinx synthesis report, what is the maximum frequency for your processor?
2. How could you get the processor to run at a higher frequency, while still keeping it as a single-cycle machine?

A demo of your pong game from lab 1 running on your completed processor will be required.