Lab 1 - Register File

1 Account Setup

First, we will set up your class account. Each student will have his or her own account. Enter the MG account login name and password (given to you by your lab TA in your first lab session) in the login window on the Linux workstations. If you are unable to log into the machine, please let your TA know immediately.

NOTE

The tools you will use only work on Linux machines, specifically the machines in the 437 lab and the machines in msee190. While some may work on other machines, they are only supported on the machines specified above.

NOTE

The script `setup` will prompt you to change your password. If for whatever reason you need to run this script more than once, answer no or Ctrl+c when it prompts you to change password (second time or more). ECN has implemented a 1 year lockout on previously used passwords. If you accidentally change your password, you must wait 1 year to reuse it.

The next thing that needs to be performed is to set up your account so that you may access the variety of tools that will be required for this class throughout the remainder of the semester. Pull up a terminal window and run the script:

```
>  ~ece437l/tools/setup
```

and follow the instructions.

NOTE

Check that your `.cshrc` file (it is located in your home directory but hiden, just try to open it with editor) contains the following:

```
setenv SUB_ece437l 'sec#'
```

Where sec# is your section (i.e. sec1,sec2,sec3,...) the first lab section starts at 1, regardless of what ssinfo/courses says. If your `.cshrc` file contains more than one `setenv SUB_` statements remove all extra.

By sourcing your `.cshrc` file, all environmental variables needed to run Quartus II, Model-Sim and HDL Designer have been updated. Whenever you open a new terminal window all the proper environment variables will be setup correctly for you to use the necessary tools.

1.1 Project 1 Setup

Now issue the following commands to setup your directory structure for the first lab of this course.

```
>  cd ~/ece437/project1
```

To set up the `project1` directory, within the `project1` directory, type:

```
>  dirset
```

Notice that several directories were created. You must maintain this directory structure. Their uses are as follows:

asmFiles This directory is where you will keep all your assembly files for testing your CPU design.

scripts This directory is where you will store the pin assignment files that allow you to map signals to buttons on your fpga for your designs.

source This directory is where you should place all your VHDL source code for the current design you are working on. This will give you one place to look for your source code in the given directory. It will also help the tools in searching for source code.

mapped This directory is where the results of synthesis on your VHDL source code will be placed. A synthesized circuit is simply the results of linking or mapping your VHDL source code to a given set of logic gates or fpga cells in a design library. The synthesized code will be used to verify that your source code would most likely translate into hardware.

work This directory is where Modelsim keeps all the compiled entities. It is the default working library where entities are accessed.

1.2 Design Flow

The design flow in this course will have three parts.

1. Code the design and test the software version in ModelSim. You will need to create a test-bench for any design you want to test if it is not already provided.
2. Synthesize the design, which turns your code into hardware elements that will end up on the fpga, and test mapped version in ModelSim.
3. Make a .sof file, which contains the synthesized design in a format that can be programmed into an fpga board, and download that .sof file to the fpga. This programs the logic elements in the fpga to correspond to your design. Finally verify the correctness of your design in hardware via Quartus to complete the design flow.

1.3 SVN Setup

We will now create an SVN repository. SVN is a version control system and it is highly recommended that you use it. This will allow you to go back to a working version of your code if something is to break.

Go back to your home directory and modify your `.cshrc` file. Issue the following commands:

```
>  cd
>  vi .cshrc
```

NOTE

You may use your favorite editor instead of **vi**.

First add the following line to your `.cshrc` in your home directory.

```
setenv EDITOR vi
```

NOTE

Feel free to replace vi with your favorite editor in the editor line above. Emacs provides a lot of helpful functionality when writing VHDL as can vi. But pico is fine for just editing your CVS log. Once again, remember to place an end line after your last line or the last line will not execute.

This will let applications know your default editor of choice. Now to set up the repository. Be sure to close and reopen your terminal, so these changes will take effect. Make sure you are in the home directoy and type:

```
>  cd
>  svnadmin create SVN
```

This will create a directory `SVN` which will hold your repository. Then to import the directory structure of your `project1` directory type the following. (Replace mg### with your mg account)

```
>  cd ~/ece437
>  svn import project1 file:///home/ecegrid/a/mg###/SVN/ece437l/trunk/project1
 -m "Singlecycle Project Initial Import"
>  rm -r project1
>  svn checkout file:///home/ecegrid/a/mg###/SVN/ece437l/trunk/project1
>  cd ~/ece437/project1
>  svn log
```

The svn import line imports your directory structure into svn. The -m "Singlecycle ..." is the log message associated with this import. It is a good idea to create a short but informative log message whenever you commit any changes to your SVN repository. Then you can check later what kind of changes you had made for any of your commits. The last line checks out Project1 into you current directory. Issuing the log command will give you a summary of the actions on a working copy.

Now whenever you create a new file use the command:

```
svn add filename
```

```
svn commit -m "informative log message"
```

For more information consult the svn man pages.

NOTE

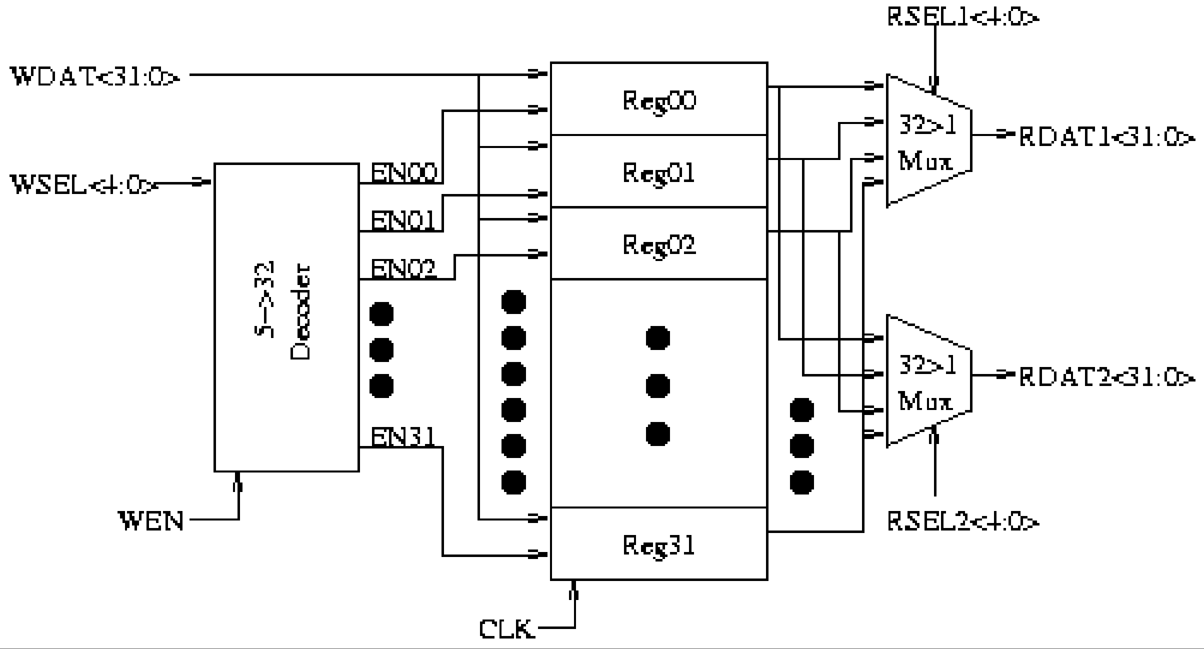
Throughout the first part of the semester, you will work ONLY under the project1 directory. Also, MAKE SURE that you use the EXACT case and names of files and directories that are specified. This is required by the grading scripts in order to recognize your files.

Most of you have had experience with the software used this class, so we are going to jump right in and begin designing a core piece of our processor: the register file.

2 Register File

The register file is the central storage of a microprocessor. Most operations involve using or modifying data stored in the register file. Since the register file runs at the full speed of the processor, it must be small and fast. The register file we will be designing has 32 locations. Figure 1 is the suggested block diagram of the register.

Figure 1 Register File Diagram



2.1 Register File Description

Each register is 32 bits wide, and there are 32 such registers in the register file. Each of the thirty two registers has a clock input (positive edge triggered for now), a data input, an enable, and a data output. This collection of registers is managed by the muxes and decoders surrounding it. There are two output (read) ports to the register file, so two registers can be read simultaneously. Each output of the registers is connected to two RDAT muxes. By changing the RSELx lines, we can choose what two register we want to output to the RDATx buses.

On the front side, we control the writing to registers. There is one write port. Since only one register can be written at a time, the WDAT bus can be connected to all the registers. The 5 to 32 decoder then takes the WSEL number and decides which register to update. If the WEN line is '1', then on the rising clock edge the selected register is updated. If WEN is '0', nothing is updated.

In order to reset the processor to a known state, we will be using the nReset line. nReset is an active low reset, so when it is '0', all the registers will asynchronously be set to ZERO. When returned to '1', the register file will function normally.

The only other aspect of the register file is that R0 is a special register. It will always output zero, no matter what is written to it. When the RSELx lines are set to "00000", ZERO should be output of the RDATx buses.

Download the skeleton file, namely `registerFile.vhd`, from the course web page and put it into your source directory. The entity declaration has been copied below. Note that the file registerFile.vhd does not contain a complete register file design, even if it compiles correctly. You have to make modifications to this file to create a complete register file design according to specifications.

```
entity registerFile is
  port ( wdat:   in STD_LOGIC_VECTOR(31 downto 0);
         wsel:   in STD_LOGIC_VECTOR(4 downto 0);
         wen:    in STD_LOGIC;
         clk:    in STD_LOGIC;
         nReset: in STD_LOGIC;
         rsel1:  in STD_LOGIC_VECTOR(4 downto 0);
         rsel2:  in STD_LOGIC_VECTOR(4 downto 0);
         rdat1:  out STD_LOGIC_VECTOR(31 downto 0);
         rdat2:  out STD_LOGIC_VECTOR(31 downto 0)
       );
end registerFile;
```

NOTE

A good habit to develop is to name the entity of the VHDL source code the same name as the file name without the "`.vhd`" extension(this is required for some scripts to recognize your design). For example, the `registerFile.vhd` should have an entity name of "registerFile".

2.2 Compiling and Simulating your registry file using VSIM

Download the registerFile test bench from the web page, `registerFile_tb.vhd`, and place it into the source directory. In this class, you will be expected to generate your own test benches for individual components, but for the first lab one will be provided for you.

The first thing you should do is add your new files to SVN. Check the status of the repository and add the file.

```
>  cd ~/ece437/project1
>  svn status
>  svn add source/registerFile.vhd source/registerFile_tb.vhd
>  svn commit
```

This will cause the editor you selected earlier to pop up. This will be a description that is associated with this file. For this first file just type: "1 word 32 element register with one input and two outputs." Save and close the editor. If you want to avoid this popping up of the editor provide a log message with your commit command, that is replace **svn commit** with **svn commit -m**"Initial commit: 1 word 32 element register with one input and two outputs".

Download the `Makefile` from the web page to your `project1` directory. A makefile is a macro tool for compiling large projects. You will maintain this makefile over the course of the semester. (You must use this makefile).

First we'll bring up VSIM. At a command prompt type:

```
>  vsim -i &
```

This should bring up a VSIM command window. From the command window you can compile and simulate your design. To compile, type:

```
ModelSim>  make lab1
```

A bunch of errors may or may not come up. Either way, note that the given registerFile design is not complete. Finish the design of registerFile.vhd in your favorite editor and then re-compile. (In particular, you need to complete reset, set/write, decoder, read1 and read2.)

To view waveforms of your design, click on the **work** folder in the library window(if closed, go view->library) of the main VSIM window. If the work item is missing or closed, First try typing:

```
ModelSim>  vlib work
```

Otherwise, extend the work folder to view its files(or ask your TA). Double click on the `registerFile_tb` entity. The terminal window should show several lines of "Loading XXXX". After this, if Objects window does not show up, go to **View** → **Objects**

Then the Objects window should give you a list of the signals in the test bench. Select these signals, then right chick and **Add** → **To Wave** → **Selected Signals**

To simulate, type

```
Modelsim>  run 3000
```

in the command window again. You should see your register file being used. As each part is tested, a text message is displayed. This indicates when something is wrong. Future test benches that you create should follow this pattern. Continue typing **run** *`time`* until the simulation is complete.

To exit out of the simulation window, type

```
Modelsim>  quit -sim
```

This should bring you back to the main VSIM window.

3 Synthesizing the Register File

Before you begin, make sure that you have a `registerFile.vhd` file in the `~/ece437/project1/source/` directory, which contains the VHDL code for your register file. Also make sure your Makefile is up-to-date.

3.1 Generating a Synthesized Design

In your `project1` directory type:

```
>  compile -s registerFile
```

All the output of synthesis is saved to a file called `compile.log`. You should run the following command to check for errors or warning in synthesis.

```
>  syschk compile
```

Check the `compile.log` for resources that your register file will take up on the chip. This will be under the table labeled "Analysis & Synthesis Resource Usage Summary". Copy this number onto your evaluation sheet. You will also want to know some timing information for your register file. This is under the table "Timing Analyzer Summary".

This will create a `registerFile.vhd` file in the `~/ece437/project1/mapped/` directory. This is the synthesized version of your registerFile. It must have the same functionality. Many designs no longer function correctly after synthesizing, so you are required to simulate this file to be sure that it works correctly. If it doesn't work you may want to check the `compile.log` file.

Go into modelsim again and this time simulate you mapped version. It is recommended that you remove your previous work library to avoid confusion on which version of your code you are simulating by typing:

```
ModelSim>  rm -r work
```

Now you are ready to simulate your mapped version. Recreate the work library using vlib and type:

```
ModelSim>  vlib work
```

```
ModelSim>  vcom mapped/registerFile.vhd
```

Now in VSIM, reload the `registerFile`'s test bench. By typing:

```
ModelSim> vcom source/registerFile_tb.vhd
```

If you see errors about a missing library, you need to restart VSIM. Reload and run the register file test bench. It will now test the synthesized version. Run the simulation. If something doesn't work, go back and look at your code. Check you entities. Is there something that you couldn't do in a circuit? Not all VHDL constructs will synthesize correctly. Watch out for loops and long nested if statements. If you need help, ask your TA.

To elaborate on the synthesis process, ECE437 is using an FPGA compiler called Quartus from Altera. The compile script you called to synthesize (the -s option) your design creates a file dependency list, creates a new Quartus project and runs the synthesis process using Quartus. Quartus project files for a particular synthesis run are located under the directory that should have the same name as the entity you were trying to synthesize (in this case registerFile will be the directory name for the project files). The synthesis process consists of running a .tcl file to configure Quartus, mapping your source code, running a fitter for the hardware, running timing analysis, running an assembler and generating a VHDL net list of the mapped design. The original VHDL net list is generated by Quartus inside the `simulation/modelsim` directory of the project directory with the name *`entity`*.vho. The synthesize script copies that file to your mapped directory and change the file extension to .vhd.

4 Synthesizing the Register File for Hardware Testing

4.1 Place & Mapping Your Design

There are two files you need to run the script. The first file, `regTest.pins`, contains the pin map. This file maps the signals in your VHDL to the pins in the hardware. The second file, `regTest.vhd`, contains a mapping of signals to switches and buttons on the fpga.

NOTE

It is important that all .pins files share the same name as the mapping file, i.e. regTest.pins and regTest.vhd. Also regTest.vhd should have an entity called regTest.

Download `regTest.pins` and `regTest.vhd` from the web page. Place the `.pins` file in the `scripts` directory and the `.vhd` file in your source directory.

After the files are in the correct location, we are ready to generate the hardware file by executing the following command from the project1 directory:

```
>  compile regTest
```

All the informational output of the fpga mapping is saved to a file called `compile.log`. You should run the following command to check for errors or warning in hardware synthesis. If the eval sheet asks for a clean synthesis the only way to get points is to have no inferred latches or combinational feedback loops show up in the log file.

```
>  syschk compile
```

regTest is the design name. Note that it is exactly the same name as files you downloaded sans the extensions. If any errors are printed, ask your TA.

To elaborate on the fpga mapping process, the same flow used in the synthesis process is used again here. The difference is that now, the hardware programming file is generated. This is a `.sof` file with the same name as your *`entity`*. The compile script leaves this `entity.sof` in the project directory.

5 Downloading the design to the prototyping system.

We will use Quartus to download the design to the prototyping system. Though this has a graphical interface, we have written a script to configure the system for download and actually download the file. Feel free to thank us at any point(no really do thank us). Also be sure to *turn on your developement board*. At the command prompt type:

```
>  compile -d regTest
```

NOTE

The script will then download the design and you will get a success message on completion.

Patiently wait for the design to download to the FPGA. Once the download is complete, you may begin testing the design.

Remember your register file? The register file you coded for this lab had several inputs:

wsel Notice the black switches. From right to left the first five switches are wsel (SW4 - SW0).

rsel1 The next five switches are rsel1 (SW9 - SW5). The data appears on the red leds (LEDR7 - LEDR5).

rsel2 The next five are rsel2 (SW14 - SW10). The data appears on the red leds (LEDR12 - LEDR10).

wdat The last three switches are wdat (SW17 - SW15).

wen Notice the four blue push buttons (KEY3 - KEY0). The first push button KEY3(the leftmost) is your write enable signal.

nReset The second push button KEY2(from the left) is the nReset. It is active low, so pressing the button will be sending a ’0’, hence a reset.

clk The clock was tied to the internal clock.

Please have your TA check the behavior of the design.

6 Questions

To complete this lab, please answer the following question on your evaluation form.

1. Thought Question (OPTIONAL): What would be the advantages or disadvantages of having writes occur on only the rising clock edge, only the falling clock edge, or on either clock edge?

7 Turn-in

Execute the following command from your `project1` directory

```
>  submit -p lab1
```

NOTE

For future reference, if you ever need to submit or re-submit a lab after the submission deadline (usually every Friday Midnight) upon TA’s permission under specical circumstances, you have to execute the following command instead: (just replace the number in ”lab1” with the corresponding lab number)

```
>  submit -p latelab1
```

NOTE

Remember, you should have all of lab 2 and lab1 done by the beginning of the second week.

WARNING

Make sure your lab 1 file is called registerFile.vhd (default) before you do your submission.

8 Makefile

You will be required to maintain this make file throughout the semester. This means you will need insert code for compiling each of the labs. All the code necessary for the first lab is included (you may download this from the website).

WARNING

The grading script uses the Makefile to compile your design. Hence failing to update the Makefile accordingly for all following labs may result your lab submission to recieve a grade of zero.

```
# ECE437 Makefile
.SUFFIXES: .vhd
COMPILE.VHDL = vcom
COMPILE.VHDLFLAGS = -93
SRCDIR = ./source
WORKDIR = ./work
VPATH= $(WORKDIR)

#Rules
%.vhd:  $(WORKDIR) $(SRCDIR)/%.vhd
$(COMPILE.VHDL) $(COMPILE.VHDLFLAGS) $<
touch $*.vhd

work:
vlib $(WORKDIR)
        vmap lpm $(WORKDIR)

# begin VHDL files (keep this)
```

```
registerFile_tb.vhd: registerFile.vhd
regTest.vhd: registerFile.vhd

# end VHDL files (keep this)

# Lab Rules
lab1: registerFile_tb.vhd
lab2: tb_alu.vhd
lab3: meminterface_tb.vhd
lab4: tb_cpu.vhd
lab5: tb_cpu.vhd
lab6: tb_cpu.vhd
lab7: tb_cpu.vhd
lab8: tb_cpu.vhd
lab9: tb_cpu.vhd
lab10: tb_cpu.vhd
lab12: tb_cpu.vhd
lab13: tb_cpu.vhd
lab14: tb_cpu.vhd

# Time Saving Rules
clean:
$(RM) -r $(WORKDIR)
$(RM) *.vhd
```

. These are all the constants for the Makefile.

. These are the rules for different file types. They tell the Makefile how to compile a .vhd file. You shouldn't have to edit this section.

. This is the area you may need to change. This tells the Makefile how to compile your files. Both the lab grading script and the synthesis script depend on this section being correct. Make sure to keep the begin and end comment. The syntax for a dependency is to put all the dependent files to the right of the colon and any required commands underneath the colon.

  For instance the registerFile_tb.vhd is dependent on registerFile.vhd. So the dependency line is:

  ```
  registerFile_tb.vhd : registerFile.vhd
  ```

  If you had a vhdl file with more than one component you would place all those components to the right of the colon separated by spaces. For instance if you had a barrel shifter with 4 constant shifters the syntax would be:

  ```
  barrelshifter.vhd : shifter1.vhd shifter2.vhd shifter4.vhd shifter8.vhd
  ```

. These are the rules for compiling and synthesizing the labs. The grading scripts will compile these entities for testing.

. This is to help clean up files if something goes wrong.

Figure 2 The Evaluation Board