Lab 1: Revisiting Digital Logic and Verilog Simulation

Due **01/29/2022** (midnight)

Instructor: Bingzhe Li
TA: Cale England

1. Introduction

In this lab warmup, you will take a quick tour of the Verilog tools that we will use in this class, and what it takes to run them by hand. You will then construct a simple register file in Verilog, which should be a review of material from ECEN 3233 or DLD. By the end of this warmup, you should be familiar with MGC ModelSim for simulation and for viewing the output of ModelSim. If you have not touched Verilog in a while, this lab should allow you to recall the basics. In later labs (beginning with Lab 3), you will be developing a more significant codebase in Verilog and simulating it with these tools.

2. GitHub

I highly encourage you to use GitHub to manage your codes for this semester. GitHub provides hosting for software development and version control using Git. It offers the distributed version control and source code management (SCM) functionality of Git, plus its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and wikis for every project. Git it used heavily in the industry for all kinds of files and all sizes of teams. Learning to use this tool now will be very useful, and if you already know how to use Git then this will be a nice review.

If you have taken ECEN 4303 - Digital Integrated Circuit Design, then you already have used GitLab (similar to the way GitHub works, but it runs the site on the local machines). If you do not have an account on GitHub, you can sign up for a GitHub account by visiting `https://github.com/`.

The following guides are used for GitLab, but I think the GitHub has the same commands for the code management:

Once you have a GitLab account, you will be added to a repository that will be where you will store your Lab 1 progress. To use this repo on your local machine, you will need the tool `git` which can be downloaded at `https://git-scm.com/downloads`. `git` is a command-line tool for accessing and working with repositories like the ones you will have access to on GitLab. You will then `clone` your GitLab repository by copying the 'Clone with HTTPS' string (shown in Figure 1) and running `git clone <copied Clone with HTTPS string>` on the command line wherever you want the repository to be stored.

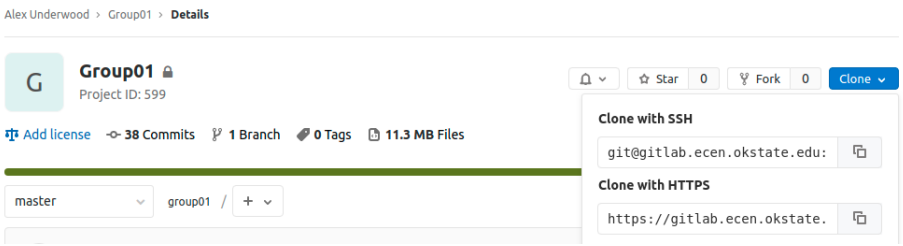


Figure 1: Cloning a Project.

When you sit down to work on your project, the first thing you'll want to do is run `git pull` on the command line inside the cloned directory. This will go out to GitLab and check if anything has changes in the repository since the last time you interacted with it and, if it has, grab the changed and update your local copy. Then you can begin working on your project with the updated files.

When you are done working, you will need to commit your changed to the repository. This is a multi-step process in which you `add` the files you want to commit, `commit` them to the repository, and `push` the commit to GitLab so you and your future partner in later labs can access the changes you just made.

1. `git status`
   (Optional) Running this in the top of your repository directory will list all the files you have added/changed/removed since the last commit in the repository. This can be useful for selecting which files you want to commit if you only want to commit some files. If you want to commit all files or you know exactly which files you want to commit already, you can skip this step.

2. `git add <files you want to commit>`
   If you want to commit all of the files that you've added/changed, you can run `git add .` at the top of your repository directory and that will include everything. If you only want to commit some files that you've added/changed and leave others uncommitted, you can add them specifically with multiple specific `git add <file>` calls.

3. `git commit -m "<helpful commit message>"`
   This will commit all of the files you added with the previous step and attach the <helpful commit message> you wrote so that you or your partner can quickly see what it was you did during the commit. Remember, the more descriptive your commit message is, the more helpful it will be later when you come back to the project after a long weekend!

4. `git push`
   This will push the commit(s) that you made to the repository on GitLab, meaning that if your partner runs `git pull`, they'll get all the new and modified files you just committed. It also means that we (the instructor and TAs) will be able to see your work, as **the files in your GitLab repository will be what we use to grade your lab assignments**.

Again, I highly encourage you to use GitHub for each lab. For the submission, you should upload the zip file including all codes to Canvas (which you can directly download from GitHub if you use that). Your simulation files and Vivado FPGA project files should be separated for easy file management (this will make more sense during Lab 3 when keeping files separate will greatly help with organizing larger projects) and your lab report PDF should be submitted to Canvas as well.

3. Part I : Simple Finite State Machine

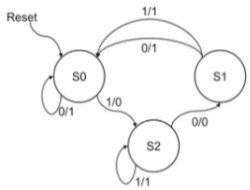
For the first part of this lab, we will look at the Verilog for a simple Mealy finite state machine. The state transition diagram and Verilog description of this Mealy finite state machine with 1-bit input and 1-bit output is given below. The Verilog and testbench will be given to you for this part - all you need to do is simulate it. However, we will go through the Verilog during the lab introduction so you can get a review as to the design approach for Verilog.

3.1 Simulation

Simulation is key to making sure your hardware for any architecture or digital system works. It is often the difference between something that works and something that is a pure speculation. Therefore, it is vital that for lab that you understand well how to simulate and get results from ModelSim.

Mentor Graphics (MGC) ModelSim is a popular Hardware Descriptive Language (HDL) compiler and simulator that is widely used in the industry. Although there are similar tools form other vendors (e.g., Synopsys VCS or Cadence Design Systems NCsim), the ideas are rather similar between Electronic Design Automation (EDA) tools. Regardless of the choice of simulator, the use of testbenches and batch files to invoke simulation are the staple of digital designers and architects.

Testbenches are essential to HDL designs and they are not unique to simulators. They are part of the Verilog standard and are typically written in a behavioral manner to make sure your simulated hardware works. Although many types of testbenches are utilized, using a testbench that verifies what the **true** result is essential. Therefore, it is encouraged that you utilize a self-checking style in your testbench. Although testbenches are usually written by the architect, for this part of the lab a sample testbench is provided to you. You should modify this testbench to make sure the design behaves just like the state transition diagram indicated in Figure 2.




```
module FSM (
    output reg Out,
    input reset_b, clock, In);

parameter [1:0] S0 = 2'd0, S1 = 2'd1, S2 = 2'd2;
reg [1:0] state, nextState;

always @ * begin
       case (state)
              S0:    begin  nextState = In? S2: S0;
                            Out = In?0:1; end
              S1:    begin  nextState = S0;
                            Out = 1; end
              S2:    begin  nextState = In? S2: S1;
                            Out = In?1:0; end
              default: begin nextState = 2'bx;
                             Out = 1'bx; end
       endcase
end

always @(posedge clock or negedge reset_b) begin
       if (~reset_b)
              state <= S0;
       else state <= nextState;
end

endmodule
```

Figure 2: Sample Mealy Finite State Machine.

The next part of the simulation environment is called a `DO` file and is basically a batch file for ModelSim that allows the simulation to run regardless of a users set up. A sample `DO` file is given to you, but you should modify to make sure it runs your finite state machine design and its appropriate testbench. To run ModelSim with a `DO` file, type the following command at a command prompt.

```
vsim -do FSM.do
```

In order for `vsim` to work on the terminal, you will need to make sure the appropriate exeutable directories are in your `PATH` variable. You are encouraged to consult your TA or the Internet to learn how to get that terminal in Microsoft Windows, so you can run your `DO` file correctly.

4. Part II: Register File

In the second half of this practice lab, you will construct a register file (RF) in Verilog. Although this lab is sort of practice of what you should already know, the module that you write in this section will be useful to you in Lab 3 if you write it properly.

4.1 Requirements

The register file is a unit in the processor which supplies other functional units (for example, the ALU) with operand values and stores the results of computation for subsequent use. Modern instruction sets typically supply the programmer with 8-32 architecturally-visible registers for integer computation. In this lab, you will implement a register file that is suitable for executing one ARM instruction per cycle. To support one instruction per cycle, the register file must allow two concurrent reads and one write per cycle because a ARM instruction can require up to two input operands and can produce one result value (e.g., from an Arithmetic Logic Unit). This is sometimes called dual-porting – in other words, it can read two values from the register file at the same time through two separate ports.

Your register file should contain 32 registers, each of which holds 32 bits. It should have have the following input and output ports:

- Inputs: Two 5-bit source register numbers (one for each read port), one 5-bit destination register number (for the write port), one 32-bit wide data port for writes, one write enable signal, and clock.
- Outputs: Two 32-bit register values, one for each of the read ports.

The register file should behave as follows:

- Writes should take effect synchronously on the rising edge of the clock and only when write enable is also asserted (active high).
- The register file read port should output the value of the selected register combinatorially.
- When reading and writing the same register, the read port will update on the rising clock edge. The read port is still combinatorial.
- Reading register zero ($0) should always return (combinatorially) the value zero.
- Writing register zero has no effect.

4.2 Implementation

Go ahead and write a module RF with the specification given above, and build a testbench for your register file. In order to help you get started, your repository should have an empty register file that is missing some pieces. However, it should have the ports to help you figure out what is an input or output. Although we have not covered register files in general, the idea should be similar to the idea of a register which you all should be familiar with. **More information on the register file can be found within the Appendix of your textbook in Section A.8.**

Ensure that you test all reasonable cases (e.g., read a register while it is being updated; read the same register with both read ports at the same time; reset the register file and ensure that all registers read zero). Use the waveform viewer in ModelSim as in the first part of this lab in order to examine the behavior of your register file. You should use the FSM testbench and `DO` file as a basis to create your own files and verify your design properly.

5. Handin

All files you want considered during grading should be compressed the whole directory as a zip file and uploaded to Canvas. Your code should be readable and well-documented. It is recommended that you pick a formatting style to write your Verilog in and stick with that style for the remainder of the semester, as consistency is key when it comes to formatting styles. In addition, please turn in additional test cases or any other added item that you used. Please also remember to document everything in your Lab Report using the information found in the Grading Rubric and submit a copy of your Lab Report to Canvas.