Instructions

Objective

The objective of this lab is to develop a set of test vectors for a MIPS ALU, which will be provided. The ALU contains one or more bugs, which your test vectors should successfully detect. After completing this lab, you should have the skills required to design and implement test benches for other SystemVerilog modules in the future.

Deliverables

- Your assignment directory, packed using the provided script.
  - Your updated `testbench.sv` file.
  - Your updated `vectors.dat` file.
- Your report in PDF format.
- **You will be turning in two separate files, one `.7z` generated by the provided script, and one PDF containing your report**.
- Each group only needs to submit once via either partner.

Specific Tasks

1. *Setup the FPGA tools for your Linux account*
   - (add `source /usr/local/3rdparty/cad_setup_files/altera.bash` to your .bashrc file and open a new terminal)
2. Download and extract the project skeleton.
3. Modify the `testbench.sv` file to be able to load and execute test vectors in the prescribed format. It should be able to load the provided example vectors, which should both pass.
   - **HINT**: areas where you should place your code are marked with a `TODO` comment.
4. Develop a set of test vectors for the provided ALU.
   - You must name your vectors file `vectors.dat`.
   - You may execute your tests using the shell command `./csce611.sh simulate`. (**HINT**: you can show messages in the Modelsim console by using the `$display()` macro).

5. (optional) use `./csce611.sh compile` and `./csce611.sh program` to run your code on the board. The provided top-level module will connect the `lo` ALU output to the HEX displays, although you will need to provide your own `hexdriver` module (you can re-use the one you made for lab 0). You may modify the `CSCE611_lab_alu.sv` file if you need to to accommodate your hex driver design.
   - This is not a graded part of the assignment, but is a good idea to practice using the boards.
6. (optional) read the `scripts/test.tcl` script which is used to control Modelsim when you run `./csce611.sh simulate`.
   - This is not a graded part of the assignment, but test and build automation are valuable skills to have.
7. Pack up your project for submission using `./csce611.sh pack`. Your submission file must be named `CSCE611_<Semester>_alu_<Your USC Username>.7z`. Your "USC username" is whatever you log into your university computer with. For example, a submission file might be named `CSCE611_Fall2019_alu_jsmith.7z`. Please only use the provided script to pack your assignment, do not do it manually.
8. Your report should be submitted as a **separate file** from your project submission. You should name it `CSCE611_<Semester>_alu_<Your USC Username>_Report.pdf`.

NOTE: you do not need to fix the bug(s) in the ALU, nor should you need to change the `alu.sv` file. In other words, your test cases should indicate that the ALU is faulty due to the bug(s) it contains.

Test Vector Format

The test vector file contains the inputs and expected outputs for your test bench.

Each test vector must be in hexadecimal format and comprise 148 bits each (37 hexadecimal digits). Use the following format:

value	marker (MSB)	a	b	shamt	op	hi	lo	zero (LSB)
direction	n/a	input	input	input	input	expected	expected	expected
width (bits)	4	32	32	8	4	32	32	4
width (hex digits)	1	8	8	2	1	8	8	1
bits	147:144	143:112	111:80	79:72	71:68	67:36	35:4	1:0

- The `marker` field is always `4'hf` and is intended for use in identifying whether a test vector is initialized or not. The first 4 bits of every test vector must be `1`.
- The field `zero` actually corresponds to the 1-bit `zero` pin in the ALU. It is padded to 4-bits to make the vectors more readable. Therefore, if the `zero` pin is to be high, the vector should make the last hex digit `1` (i.e. only the single least significant bit should be used).
- The field `shamt` is likewise padded from 5 bits up to 8 for the same reason. Thus `shamt[7:5]` is **unused**.

Example Vectors

```
// test bitwise AND with non-zero output
F_0F0F0F0F_FFFFFFFF_00_0_00000000_0F0F0F0F_0
// test bitwise AND with all zero output
F_0F0F0F0F_F0F0F0F0_00_0_00000000_00000000_1
^ ^        ^        ^   ^ ^        ^        ^
1 2        3        4   5 6        7        8

1: marker
2: a
3: b
4: shamt
5: opcode
6: hi
7: lo
8: zero
```

Note: only the first 4 lines of the above are syntactically valid, the remaining lines should not be copied to your `vectors.dat` file and are provided only for illustrative purposes.

Design Requirements

1. You must use the provided project skeleton. Your project should execute without error using `./csce611.sh simulate`.
2. Your test vector file **must** be named `vectors.dat` and use the format described above.
3. You should not add or remove any SystemVerilog files to the project, except for your hex display driver should you choose to complete step 5.

Report

You must write a report. The report should be written using a reasonable choice of font and other stylistic options. The report **must** be in PDF format.

The report must include the following information:

- A listing of your test vectors, annotated with general comments. For example:

```
// test bitwise AND with non-zero output
F_0F0F0F0F_FFFFFFFF_00_0_00000000_0F0F0F0F_0
// test bitwise AND with all zero output
F_0F0F0F0F_F0F0F0F0_00_0_00000000_00000000_1
```

- A brief (~ 1-3 paragraphs) justification of why your test vectors demonstrate correctness in the ALU design. This should be a general, high-level description of your approach, rather than a line-by-line description.
- A brief (~ 1-3 paragraphs) explanation of the bug(s) found within the ALU. If you were not able to find the bug(s), speculate on what you think it/they might be and why.

In total, your report should be not more than 1000 words, not including the test vector listing. It is not necessary to fill all allowed space with prose, the 1000 word limit is an upper boundary, not a target.

You may include figures, diagrams, drawings, testbench output listings, etc. in your report if you feel they add to the text, but please be sure any hand-drawn figures are clear and legible. The entire report must be in a single PDF document.

Rubric

- (A) 20 points – annotated test vectors
  - This category takes into account both the thoroughness of the test vectors, and the clarity of their annotations/comments.
- (B) 25 points – explanation of test vectors
  - The reader should clearly understand the testing strategy used to ensure the ALU functionality was exhaustively validated. The explanation should not include detailed line-by-line analysis of the ALU or test vectors, but specific edge case may be pointed out.
- (C) 25 points – explanation of bug(s)
  - (C.1) 10 points – bug(s) is/are correctly identified.

  - (C.2) 15 points – the bug(s) should be clearly explained, including your thought process and discussion on specific relevant test vectors or lines of SystemVerilog code from the ALU.

- (D) 10 points – stle
  - The report should feature a reasonable choice of font, line spacing, margins, etc. For example, the LaTeX default styling, or 12pt Times New Roman with 1 inch margins would both be reasonable choices of style. 38pt Fuchsia Comic Sans would not be considered reasonable.
  - The prose of the report should be clear and easy to follow.
  - The report should exhibit correct spelling and grammar (please use a text editor which features spell check).
  - **The report should be in PDF format. If it is not you will recieve 0 style points.**

Maximum score: 80 points.

Additionally, the following may cause you to lose points:

- If the code provided in your submission does not match what is shown in your report (i.e. in screenshots, code listings, etc.), you will be given a failing grade on the assignment.
- If your report is submitted without your project code, you will be given a failing grade on the assignment.
- Your code may be runnot including the test vector listing, through Moss, and your report through plagiarism detection software such as TurnItIn. Plagiarism and other forms of cheating will be reported to the academic honesty department, which may result in a grade penalty.
- **NOTE**: if you wish to instead write a code generator (or similar tool) to produce test cases, you may instead include it's source code with detailed comments, and a 1-3 paragraph justification and overview of it's design and correctness, in lieu of annotated test vectors. The code generator must be entirely your own work, and run on the Linux lab machines.

Project Structure

The structure of the provided skeleton is as follows:

- `CSCE611_lab_alu.htm`: log of pin assignments for the DE2-115 board.
- `CSCE611_lab_alu.qpf`: Quartus Project File for this project.
- `CSCE611_lab_alu.qsf`: Quartus Script File for this project – this is where project settings such as the device, top-level entity, pin mappings, and project file are configured.
- `CSCE611_lab_alu.sdc`: Synopsis Design Constraint file defining information about the system clock.
- `CSCE611_lab_alu.sv`: Top-level SystemVerilog module for the project.

- `alu.sv`: ALU implementation file.
- `testbench.sv`: test bench file for the ALU.
- `vectors.dat`: test vectors file.
- `alu_system_builder_settings.cfg`: Terasic System Builder settings used to generate the project file.
- `csce611.sh`: front-end wrapper for course-specific scripts.
- `scripts/`: course-specific scripts.

You should not directly modify any of the above files except for `vectors.dat` and `testbench.sv` (and possibly `CSCE611_lab_alu.sv`), however it is very important that you not change the top-level SystemVerilog module name, inputs, or outputs, or the test script will break and you will lose points. The QPF or QSF file may be modified using TCL commands, `csce611.sh` or the Quartus GUI, but should not be edited by hand.