ECE/CS 3700

Spring 2018

Assigned Tuesday (April 3) onwards. Due date: Please try to demo the circuit during your regular lab hours for the week 4/17-4/23. Final report is due by Friday 4/27 on Canvas. In case any delayed check-offs are needed, I will ask the TAs to make themselves available on reading day, 4/25, for around 1.5 hours in the lab. Please coordinate with your TA in such cases.

Now that you have become familiar with both combinational and sequential logic design (adders, counters, latches, FFs, etc.) it is time to put these pieces together and build some large systems. In this lab you have an option of designing any one (not both) of the following two designs:

- Project 1: Verilog Design, Simulation, Synthesis, Implementation and Testing of a 4-bit CPU and control.
- Project 2: Verilog Design, Simulation, Synthesis Implementation and Testing of a music/tone generator.

Of the above two projects, you may choose the one that suits your interests. In my view, the CPU design project is more suitable for those students who have taken EE/CS 3810. However, that does not mean that those students who have not taken 3810 should avoid this project. Both projects require the implementation of a state-machine and interfacing it with other components. So, the choice is yours.

Described below is the design description for each of the projects.

I. THE 4-BIT CPU AND CONTROL

You will be designing a circuit very similar to the one shown in Fig. 7.9, pp. 431 (or Fig 7.73 in the second edition) in the textbook. Take a look at the figure and maybe you should also go through Sections 7.1.1 and 7.1.2 (or Sections 7.14.1 and 7.14.2 in the 2nd edition).

Your CPU will have the following:

- An Arithmetic and Logic Unit (ALU) that takes 2 4-bit numbers as inputs and produces an output (4-bit for logical ops, 5 bits for add, but you may choose to ignore the carry bit). The operations that the ALU performs are ADD and the following bit-wise logical ops: OR, XOR, NOT.
- There are three 4-bit registers R_1, R_2 and R_{out} connected to a 4-bit bus via tri-state buffers.
- I'm asking you to design a small finite-state-machine (FSM) that "hard-codes a few instructions", executes them, and then comes to a halt. The execution can be re-started when reset is pressed. This FSM is essentially a circuit that controls/assigns the signals to: i) the register- and tribuf-enable signals that transfer data between registers via the bus; and ii) select the required ALU operation. This FSM/CPU control (and the program to be hard-coded) is described in the next section.

(Note: In a general purpose computer, the bus is connected to a memory that holds a program. The CPU control then has the responsibility to fetch/decode/execute the instructions - usually done by enabling/disabling the buffers and registers, just as required above. The FPGA board does have an SDRAM where you can store

a large program. However this requires an interface via a memory controller and the subsequent design and synthesis would require a lot more time and effort than a 2-week lab project. It would, in fact, turn out to be a mini-3710 project. So, I'm asking you to just "hard-code" a program in an FSM, and implement the required CPU control. This way, you will get a fair idea of why CPU control is a state machine and how does it work!)

- The Bus is also connected to a 4-bit external input (say, a DIP switch on your board?) to read data that the FSM would then process. Note the tri-state buffer to isolate the switch.
- A block diagram of the design is shown in Fig. 1. The entire system is synchronized w.r.t. a common clock. The block-diagram is just a reference, you may (or may not) require to add/delete a few signals. I give you have enough flexibility to implement your design.
- Just make sure to implement: i) a global *reset* as an external combinational input connected to a push-button switch on the FPGA board; ii) connect a 7-segment display to R_{out} to view the result; and iii) 4 push-button or dip-switches to read external data as input.



The FSM for CPU control

Fig. 1. Block Diagram of the 4-bit CPU+Control

A. FSM: CPU control + Hardcoded Program

Your design will interface the CPU with a control FSM that has the following specifications:

- At Reset, it goes to State-0 (S_0). Here, it tri-states all bufs, and resets all registers to 4'b0000.
- When reset is de-activated, the machine transitions to a next state (from S_i to S_{i+1}) at every positive-edge of the clock.
- In State-1 (S_1), the machine reads the data on the external switch and loads it in register R_1 . In other words, the first instruction being executed is "LOAD R_1 EXTERNAL_DATA". How would you do this? Well, open the

respective buffers to put the data on the bus; enable R_1 so that at the next positive edge of the clock, R_1 loads the data corresponding to the external switch connections.

- In S_2 , load an integer-value 3 in R_2 . This is akin to a "load-immediate" instruction: LDI R2 4'B0011.
- In S_3 , ADD $R_1 + R_2$ and store the result in R_{out} . For this, you will have to give a signal to the ALU that it has to perform the add operation on the data available at its inputs, and you have to enable R_{out} so it will store the result at the next posedge clk. Instruction: $(R_{out} \leftarrow R_1 + R_2)$.
- In S_4 , transfer the data from R_{out} to R_2 : Mov $R_2 \leftarrow R_{out}$.
- In S_5 , $R_{out} \leftarrow R_1 \lor R_2$, (bit-wise OR).
- In S_6 , Mov $R_1 \leftarrow R_{out}$.
- In S_7 , $R_{out} \leftarrow \text{NOT}(R_1)$, (bit-wise complement).
- In S_8 , Mov $R_1 \leftarrow R_{out}$.
- In S_9 , $R_{out} \leftarrow R_1 \oplus R_2$, (bit-wise XOR).
- When you get to S_9 , you display the result on the 7-segment and stay in this state until the reset is pressed in which case re-start the process from S_0 .

Hints:

- One can test your program by adjusting the external-DIP switches to different 4-bit values and re-starting the program. For example, what is the value of R_{out} (7-segment display) in the terminal state, when $R_1 = 4'b1001$ in the initial state?
- Study tri-state buffers. Recall, bufif0 and bufif1 library modules that I showed in class? If you don't, try google! Or Table A.2, pp. 695 (pp. 785 in the 2nd edition).
- Should the 'bus' variable be declared as a wire? Or as something else? See Section 7.1 (Sec 7.14 in 2nd edition).
- Try to implement each element of the CPU as a separate module. Your top-level hierarchy would then interconnect all these modules via wires/tri-state buffers.

The Assignment: Verilog Design + Simulation + Synthesis + Mapping + Demo + Project report = the usual.

II. THE MUSIC/TONE GENERATOR

If we generate a square/sine-wave of some frequency (between 20Hz to 20KHz) and connect it to a speaker, you'll hear a tone. Musical notes (A, B, C, $C^{\#}$, etc.) can be generated by designing a clock corresponding to that frequency. If I give you a document that lists the frequency of all musical notes, then you've designed a rudimentary tone-generator. Of course, the quality of the sound depends upon amplification, filtering, signal processing, etc., but for this project the main idea behind such a circuit is generating clocks at different frequencies. Select different clocks at different time intervals, and what you'll hear is going to be "music to your ears". So how is this different from the stop-watch? Well, the assignment is described below; refer to Fig. 2 too.

Your design will operate as follows:

• Using the push-button switches S0, S1, S2 you can play each of the 7 notes (as in a trumpet?). We will generate only major notes.



Fig. 2. Block Diagram of the Music/Tone Generator

- When S3 is 0, your design should play the notes corresponding to the switches (S0, S1, S2) pressed. When S3 = 1, you'll ignore S0, S1, S2 and play a pre-recorded song. The pre-recorded song is an FSM design.
- First, let us consider the case when S3 = 0 in which case S0, S1, S2 act like your keyboard.
- When none of the switches are pressed, you will not play any note. When you press S0, you'll play note A. When you press S1, you can play B. When you press all three switches together you'll play the last note G. Therefore, by pressing a combination of these switches, you can create "music" (or "noise", depending on your tastes).
- When S3 = 1, you'll play a tune called *Habanera* from *Carmen*, an opera by Bizet. The composition (the notes, the frequency, and the duration of each note) is available as another document on the class web-page. Using the table, you'll build a state-machine that generates a clock at a particular frequency, for a particular duration (corresponding to the note being played). Looks kind of tedious at first glance, but the design consists of a bunch of counters selected by a "selector FSM"!
- A midi file corresponding to the tune is also uploaded on the class webpage so you can hear the tune.
- Design your machine such that it keeps on repeating the tune i.e., keeps on looping through (when S3 = 1). The moment S3 = 0, the machine should get into the input polling mode.
- You can implement S3 using a DIP switch, and S0, S1, S2 using the push-button switches.
- Speakers will be made available on the desks in the lab for testing and demo.
- For the "Note to Frequency" translation, refer to the site: https://pages.mtu.edu/~suits/notefreqs.html

The Assignment: Verilog Design + Simulation + Synthesis + Mapping + Demo + Project report = the usual.

A. Lab Report Submissions + Deadlines

As usual, you will be expected to document your labs in a professional manner. Show your design as a blockdiagram/schematic. State any assumptions, highlight important features, elaborate on any "optimizations" that you may have performed. Describe the testing/troubleshooting strategy. **Deadlines:** Classes end on Tue, 4/24. While you will need two weeks to complete this project, some of you may still have finishing touches left on the stop-watch assignment this week. So, let us keep the week of 4/17-4/23 for class demos. And in case of any delays, I will ask the TAs to be available on reading day April 25 for final check-offs. I'll be a nice guy and give you the opportunity to submit the final report also by Friday 4/27.

Have fun!