

# Lab Assignment 6

ECE/CS 3700

Spring 2009

Assigned Mon (April 14) onwards, circuit demo during the week 4/21-4/24, final report due by 4/29.

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. So, in this lab, your assignment is:

- Verilog Design, Simulation, Synthesis, Implementation and Testing of a **4-bit CPU and control**.

The project requires the implementation of a state-machine and interfacing it with other components. Given below is the design description:

## I. THE 4-BIT CPU AND CONTROL

You will be designing a circuit very similar to the one shown in Fig. 7.73 in the textbook. Take a look at the figure and maybe you should also go through Sections 7.14.1 and 7.14.2.

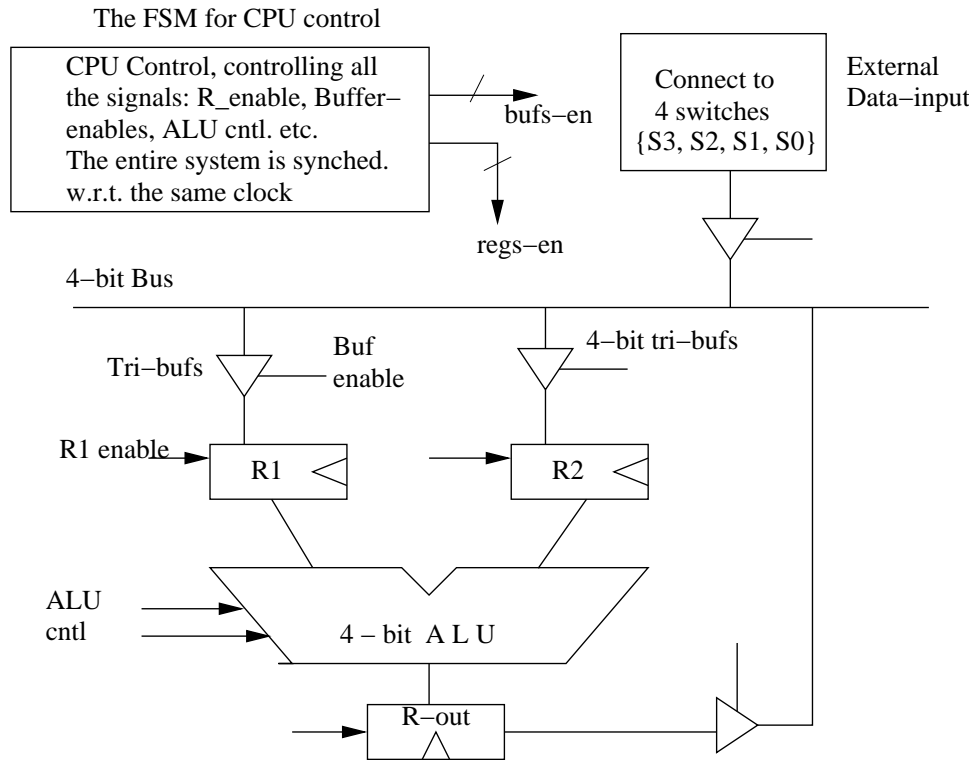
Your CPU will have the following:

- 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 1-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. 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 XESS 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.



#### 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  $S_5$ ,  $R_{out} \leftarrow R_1 \vee R_2$ , (bit-wise OR).
- In  $S_6$ ,  $Mov R_1 \leftarrow R_{out}$ .
- In  $S_7$ ,  $R_{out} \leftarrow NOT(R_1)$ , (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!
- Should the 'bus' variable be declared as a wire? Or as something else? Check Section 7.14.
- 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.

*B. Lab Report Submissions + Deadlines*

As usual, you will be expected to document your labs in a professional manner. Show your design as a block-diagram/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 Wed, 4/29. So I will ask you complete the design within 1 week, i.e. during the week 4/20 - 4/24, and then you can submit the lab report by 4/29. This way, you can use the next few days to work on the assignment, and submit the report later.

Have fun!