

Comp 541 Digital Logic and Computer Design

Fall 2014

Lab #8: Working with Memories (RAM)

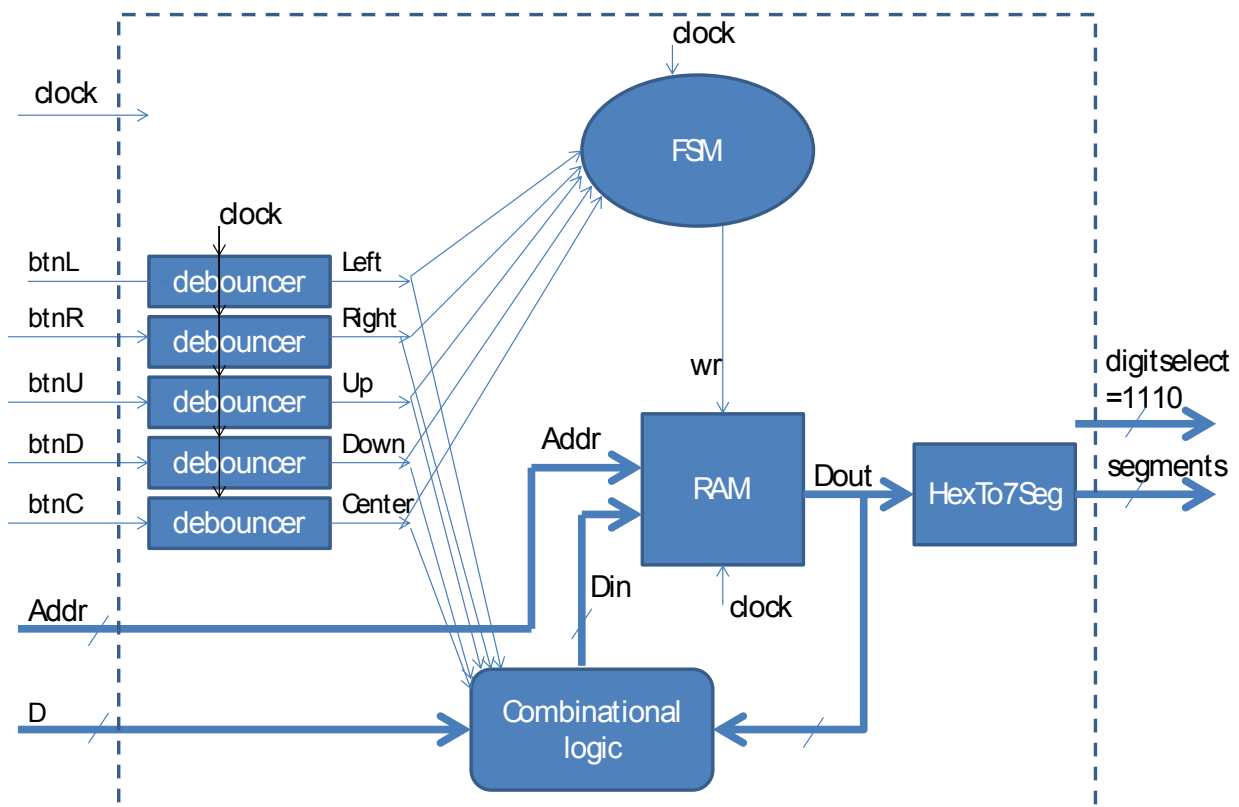
Issued Wed. 10/8/14; Due Wed. 10/22/14 (submit by 11:59pm)

You will learn the following in this lab:

- More practice with top-down hierarchical design (no top-level template provided!)
- Designing a simple memory unit (RAM)
- More practice with state machines (FSM)
- More practice with switch debouncers and 7-segment displays

In this lab assignment, you are to design a system containing a small RAM module and test it by filling it with values, and repeatedly reading them and modifying them. The data inputs to your system are: (i) an *address* to select which memory location is being read/written; and (ii) *data* to be written into the memory location (if writing). The output from your system is the data value at the location the memory is being addressed (i.e., read value), displayed on a 7-segment display.

The following block diagram shows the complete system (called memdemo). *Please study it carefully!*



The inputs and outputs of the system are as follows:

- *clock*: Delivered to all of the clocked components inside the system.
- Push buttons: Inputs from the five push buttons on the board, labeled *btnL*, *btnR*, *btnU*, *btnD* and *btnC* (center, sometimes labeled *btnS* in documentation).
- *Addr*: A 4-bit value that provides the address of the memory location being referenced. This value is entered on the board via slider switches #4-7. Thus, this assignment only needs a memory unit with 16 locations.
- *D*: A 4-bit value that provides the data for storing into memory. Thus, each word in memory is a single hex digit [0..F]. This value is entered on the board via the four rightmost slider switches (#0-3).
- *segments*: The value read from memory is encoded for output on the 7-segment display.
- *digitselect*: A constant output value that selects the rightmost character (out of four on Nexys 3, or eight on Nexys 4) on the 7-segment display.

NOTE: A *.ucf* file is provided with the appropriate pin bindings for all of the above input/output signals.

The behavior of the system is as follows:

- The user selects the memory location (0 to 15) by sliding the switches corresponding to *Addr*.
- When the user provides *Addr*, the memory unit outputs the value stored at that location, *mem[Addr]*. This is a 4-bit value that shows up as one character on the segment display.
- *If the user presses one of the push-buttons*: The value at the reference memory location is modified. Its new value depends upon which button was pressed, as follows:
  - *center* button: memory value is set to zero
  - *up* button: memory value is incremented by 1
  - *down* button: memory value is decremented by 1
  - *left* button: memory value is bitwise AND'ed with the value *D*
  - *right* button: memory value is bitwise OR'ed with the value *D*
  - NOTE: For each of these scenarios, the memory write operation should take place when the button is pressed; subsequently, your state machine should wait until the button is released before going back to its initial state.

Your design must follow these guidelines:

- The top-level design must correspond to the structure shown in the block diagram, and its Verilog description must be called `module memdemo` in file `memdemo.v`.
- Use the *hexto7seg* encoder module from Lab 5 Part III.
- Use the *debouncer* module from Lab 6. You will need five separate instances.
- Code is provided for *ram\_module*. First read it carefully and make sure you understand it.

- For the FSM, first draw a diagram with states and transitions. You should need no more than six states: one initial state, and one state each to represent each of the five button *presses*. The FSM should go back to its initial state once the button is *released*. This is important! Otherwise, similar to Lab 6, you will end up writing the memory thousands of times during each button press. This FSM is easier than that for Lab 6. You will need to scan it (or take a picture) and submit it along with the other files for this lab. The Verilog description must be in a separate module (`module fsmLab8` in file `fsm.v`).
- For the combinational logic at the bottom of the block diagram, write a simple piece of code that generates the correct value to be stored in memory, i.e., either 0, or the current value + 1, or the current value - 1, or ... etc. This should be in a separate module (`module comb`) in a separate file (`comb.v`).

The following files are provided on the website.

- **ram.v**: This file has the Verilog description of a parameterized RAM module. The parameters allow three attributes to be specified: number of address bits, number of data bits, and number of memory locations.
- **Lab8.ucf**: This file has the pin assignments.

---

***What to submit:***

- **A picture of the state diagram for the FSM (e.g., `fsm.png`).**
- **Your Verilog source for top-level design (`memdemo.v`), the FSM (`fsm.v`) and the combinational logic block (`comb.v`).**
- **Show a working demo of your design during the lab session on Oct 24, 2014.**

***How to submit:*** Please submit your work by email by **11:59pm, Oct 22, 2014**, as follows:

- **Send email to: [comp541submit-cs@cs.unc.edu](mailto:comp541submit-cs@cs.unc.edu)**
  - **Use subject line: **Lab 8****
  - **Include the four attachments as specified above**
-