Lab 5: Using Random Access Memory with the 8051

CpE 214: Digital Engineering Lab II

Last revised: March 11, 2013 (CAC)

In this lab, you will be provided with a Random Access Memory (RAM) module in the form of an EDIF file. The given RAM is double ported i.e. it has separate Read and Write ports, just like in the 8051 Module that we use. You are required to interface this RAM. Then you will be given some address locations and data to be written to the RAM. You will write a C program for the 8051 to execute the above Write operation, and a routine to then read those locations back to display the contents on seven-segment displays available on DE-2 FPGA board.

0.1 Outline and Concepts

1. Introduction to RAM
2. Creating Top Level Design
3. Writing the C program
4. Testing with Cyclone II FPGA

1 Introduction to RAM

Random Access Memories are volatile memories used in almost all the modern electronic devices on account of their ever-increasing storage capacity, ease of use, and speed of operation. They care called "Random Access" memories, because any address of a RAM can be accessed randomly at any time and can be read/written any number of times. There are two types of RAM: Static RAM (SRAM) and Dynamic RAM (DRAM). SRAM is extremely fast and dissipates less power but requires a high silicon area and is very expensive. It exclusively used as Cache Memory in Microprocessors. DRAM, on the other hand, is cheap, area efficient but is slow and requires a frequent refreshing. It is used as the main memory in modern personal computers. The choice between the use of SRAM and DRAM is made on the basis of cost, power and area requirements of a design. In this lab, you will learn the basic interface between a microprocessor and a RAM. You will also learn about the Read and Write cycles of a RAM. The RAM module will be provided to you by the instructor that has an 8-bit address bus and an 8-bit data bus. The read and write cycles are illustrated in Figure 1 and Figure 2.

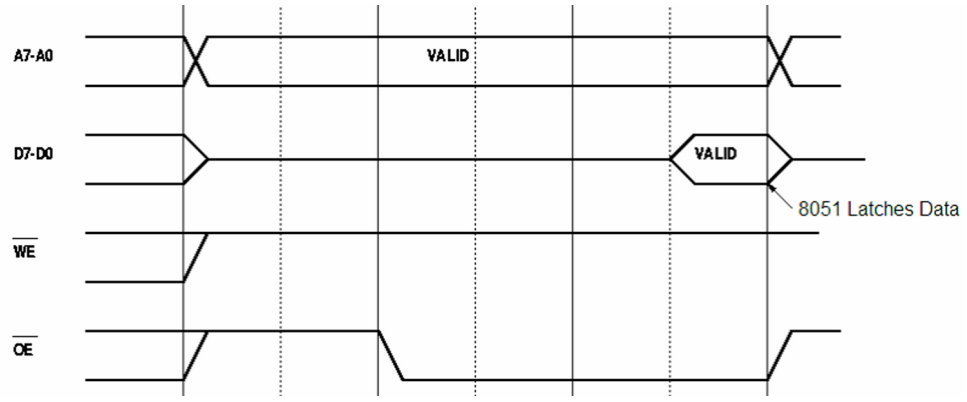

FIG. 1: RAM read cycle

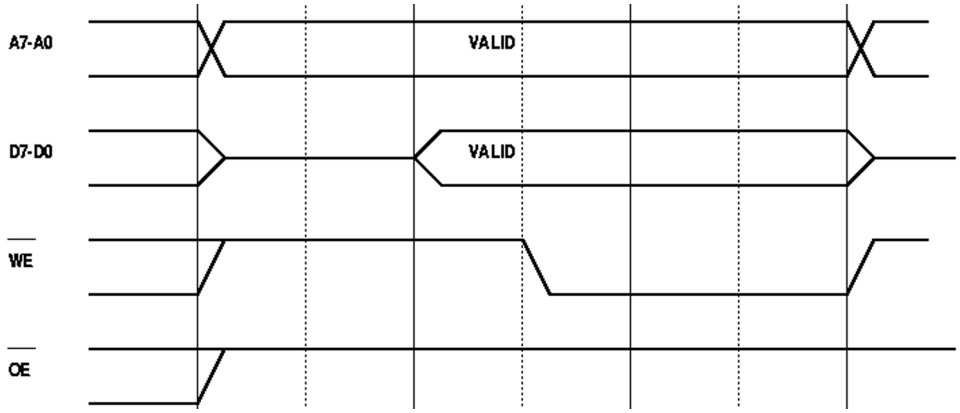

FIG. 2: RAM write cycle

2 Creating Top Level Design

The top level design in Quartus is very similar to what you have done for Lab 4. The only difference is you will remove the address decoder and latches, and replace it with the RAM module (provided on Blackboard) and the 7-seg decoder you made previously in Lab 2.

1. Make a new folder and copy all of the provided lab5 files on Blackboard into it first. Then create a new project in Quartus titled lab4. Create symbols for the 8051.edf module, and the RAM module.
2. Locate your 7-seg design file (.bdf or .edf) and corresponding symbol file (.bsf) in your lab2 folder, and copy both to your new lab4 folder. Add the design file to the file list.
3. Once you have all the files in place and imported, proceed to make a top-level block diagram schematic that looks like Figure 3.

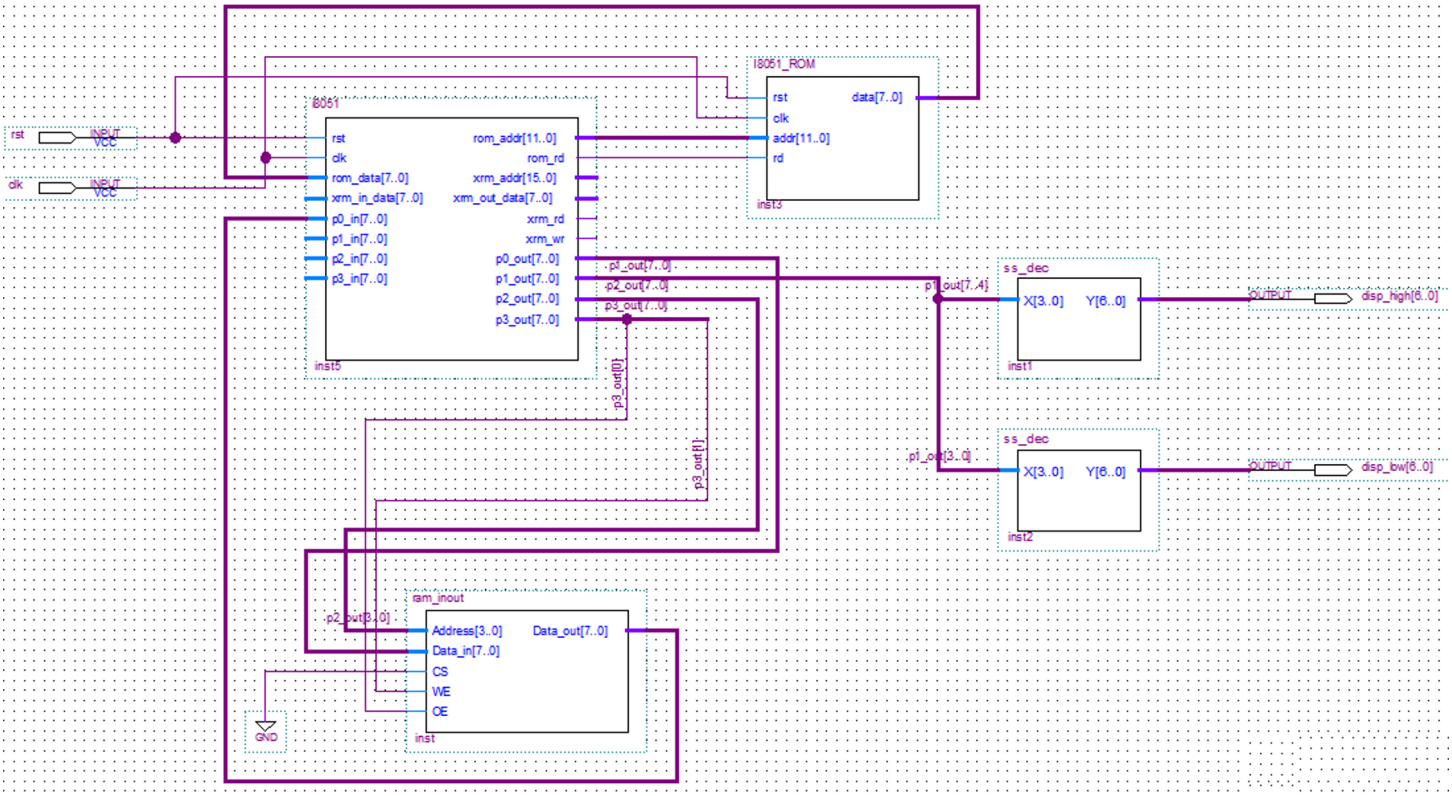

FIG. 3: Top level schematic design with RAM

When you finish, you should compile your design to check for any errors. *If the only error you receive is about I8051_ROM, then everything is GOOD and you can proceed to writing your code.*

3 Writing the C program

Just like Lab 4, you need to write a C program in Keil. A template is again provided for you on Blackboard to work with, but this time you will need to write more of it on your own. The code has two main functions: readRAM() and writeRAM(). You will use the writeRAM function to place the data at the locations in the table below first.

Data	Address
0xFF	0x00
0xAB	0x04
0x05	0x0A
0x7D	0x0C

Once this data is written to the memory addresses, you will then want to read them back with the readRAM function. readRAM should access the memory locations one by one, and then output the results to Port 1 and the seven-seg displays.

- While writing the C code, make sure that you use Port 0 of the 8051 as the RAM data port, Port 1 as the seven-seg display output, Port 2 as the RAM address port and Port 3 as the control signals such as OE (Output Enable) and WE (Write Enable) pins. *Both are ACTIVE LOW.*

The specific steps for the writeRAM function are:

1. Set OE' and WE' high to ensure no reading or writing is being done.
2. Send the address via Port 2.
3. Send the data to be written via Port 0.
4. Set WE' low to enable writing to the RAM.
5. Set WE' high to latch the data in RAM and disable writing.

Similarly, the steps for the readRAM function are:

1. Set OE' and WE' to 1.
2. Send the address out on Port 2.
3. Set OE' low which tells the RAM to output the data.
4. Read the data with Port 0.

After you finish your C code, make sure you have set Keil to output a hex file and then compile it. Then use the Hex to ROM generator on Blackboard on the hex output file to generate your i8051_rom.vhd code file like we did in Lab 4. You will then import this file into your Quartus project. It serves as the definition for the ROM symbol on the schematic diagram and will clear up the Quartus compile error in reference to it.

4 Testing with Cyclone II FPGA

Before you test on the FPGA, you should compile your design first after you import your VHDL code back to Quartus. This way we can fix any errors first before you get to the board and something doesn't work. You should use Appendix D, as usual, to make your pin definitions.

1. Assign input pins N25 as your reset, N2 as your clock.
2. Choose any two side by side 7-seg displays on the board to hook your decoders to. The highest numbered 7-seg (#7) is on the far left of the board, and decrease left to right. Compile once more.
3. Use the programmer to flash to the board and observe your program to see if it works correctly. You should see the data you wrote (from the table given earlier) display in sequence on the 7-segs.

5 Questions (attach at end of your report)

1. What is the size of the RAM module you were provided with?
2. A ROM module does not have one control signal that a RAM module has. Which one and why?
3. Was an ALE signal needed in your design and why/why not?