<Lab 11-Exploring the Operation of a 2-Read, 1-Write Memory for use in a Basic Processor>

Name: Robert Bara

Section #: 003

Date: 4/9/2020

Summary/Abstract (10 points)

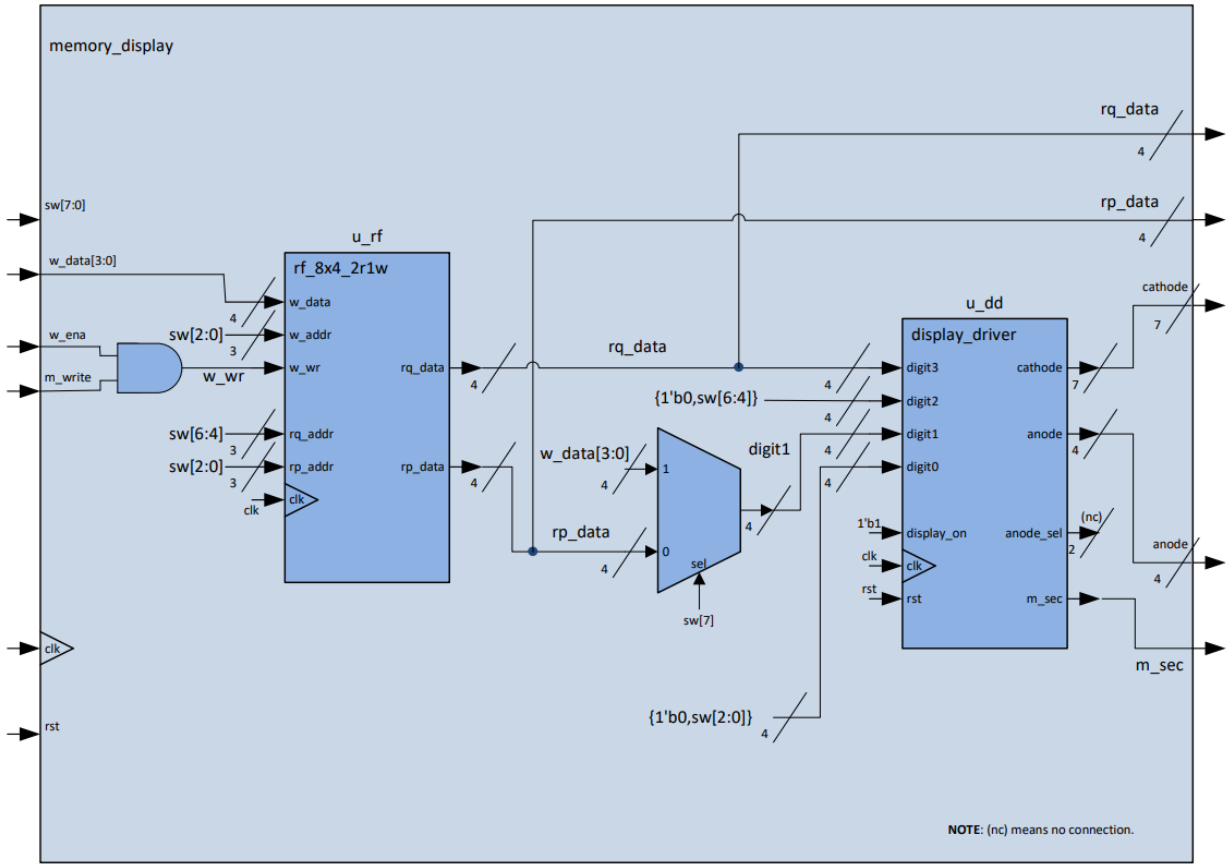
This lab introduces the logic of a basic processor by instantiating a 2-read, 1-write processor to create a small 8-word deep by 4-bit wide which will hold my TU ID into the memory location.

Introduction (10 points)

The idea of creating the 2-read, 1-write RAM is the fundamental of a register within a microprocessor. Most standard memories are single port which means they use a single address bus which selects the input and output data paths. Dual port contains a 1-read, 1-write memory which contains independent addresses for controlling to read or write to the memory cells. The 2-read, 1-write memory in this lab has three address busses that access the same internal memory so the processor can read two words in different memory locations at once. Without registers, any sort of electronic such as a computer or phone would be able to work because they rely on registers to update memory cells to read/write/store data to the device, therefore this lab is extremely relevant for any ECE major, since it is an introduction to processors.

Procedure (15 points)

The lab manual lays out a block diagram with the respective inputs and outputs below.



[Figure 1: Block Diagram for the Memory Display]

The syntax for this lab is straightforward. It requires two instantiations: the rf_8x4_2r1w module that was provided for the lab, and the instantiation for the display_driver module that was used in previous labs. From this the signals w_ena and m_write can be fed into the 8x4 2r1w module by using an assign statement or an AND gate, this should be placed before the instantiations. It is also worth noting digit1 and w_wr need to be defined to connect the block diagram correctly. Finally, the 2x1 multiplexer can easily be created through a simple if, else statement.

A functional Table describes how exactly to read and write data to the Basys3 board below. Essentially the basics of writing data is this: switches 0-2 control Digit 0 using a 3-bit binary input on the switches. This represents which memory cell we are working in. Switch 7 needs to be on to write information, and switches 8-11 can use a 4bit binary input to write a hexadecimal value to the memory cell, this is displayed on Digit 1. Pressing the left button then signifies you are done writing and switch 7 should be turned off. The middle button acts as a reset, and when reading data Digit 0 represents the 3'h address from p, Digit 1 is the 4'h data read from address p, Digit 2 is the 3'h address q, and Digit 3 is the 4'h data read from q:

Functional Table	
Function: reset	
reset	center pushbutton (***rst*** input signal in ***memory_display***)
Function: write data to memory	
Inputs:	
Select write function	switch 7 – on (***w_ena*** input signal in ***memory_display***)
Write address	3-bits: switches 2-0
Write data	4-bits: switches 11-8 (***w_data*** input signal in ***memory_display***)
Command: write data to address	left pushbutton (***m_write*** input signal in ***memory_display***)
Outputs:	
Digit 0	3-bits: hexadecimal, address to write to (switches 2-0)
Digit 1	4-bits: hexadecimal, data to write (switches 11-8)
Digit 2	(same as read function below)
Digit 3	(same as read function below)
Function: read data from memory	
Inputs:	
Select read function	switch 7 – off (***w_ena*** input signal in ***memory_display***)
Read address, port p	3-bits: switches 2-0
Read address, port q	3-bits: switches 6-4
Outputs:	
Digit 0	3-bits: hexadecimal, address p
Digit 1	4-bits: hexadecimal, data read from address p
Digit 2	3-bits: hexadecimal, address q
Digit 3	4-bits: hexadecimal, data read from address q

[Figure 1b: Functional Table from Lab report]

For the simulation the only work required was figuring out how to translate my TU ID that is 915614617 into the 8 addresses (it discards the last one). I was able to do this using the following table and logic:

4'b	h
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1110	C
1111	D

SW[2:0] SW[11:8]

Memory Address	Data
3'b 000 0	9 : 4'b 1001
3'b 001 1	1 : 4'b 0001
3'b 010 2	5 : 4'b 0101
3'b 011 3	6 : 4'b 0110
3'b 100 4	1 : 4'b 0001
3'b 101 5	4 : 4'b 0100
3'b 110 6	6 : 4'b 0110
3'b 111 7	1 : 4'b 0001

TU: ID: 915614617 → discard

[Figure 2: Logic for assigning TU ID into the 2r1w]

Results (45 points – see sub-sections)

This lab came pretty quickly for me, there wasn't a lot of code and personally I was able to verify the simulation and match the logic to the hardware implementation. It also makes sense for this lab to be straight forward since it is the start of the memory unit and building block for the basic processor that we will be creating next lab by running this into an ALU.

Design Code (15 points)

memory_display

```
//
// lab11 : version 04/05/2020
// Robert Bara
module memory_display (output logic [6:0] cathode, output logic [3:0] anode,
    output logic m_sec, output logic [3:0] rp_data, output logic [3:0] rq_data,
    input logic rst, input logic clk, input logic m_write,
    input logic w_ena, input logic [3:0] w_data, input logic [7:0] sw);

    //define wires
    logic w_wr;
    logic [3:0]digit1;
    assign w_wr=w_ena & m_write;    //AND Gate
    //Instantiate rf_8x4_2r1w
    rf_8x4_2r1w u_rf(.rq_data,.rp_data,.w_data,.w_wr(w_wr),.w_addr(sw[2:0]),.rq_addr(sw[6:4]),.rp_addr(sw[2:0]),.clk);
    display_driver udd(.cathode,.anode,.anode_sel(),.m_sec,.digit3(rq_data),.digit2({1'b0,sw[6:4]}),.digit1(digit1),.digit0({1'b0,sw[2:0]}),.display_on(1'b1),.clk,.rst);

    //2x1 multiplexer
    always_comb begin
        if(sw[7]==1'b0) digit1=rp_data;
        else digit1=w_data[3:0];
    end

endmodule

```

[Figure 3A: Memory_Display module design code]

Simulation Results (15 points)

```
Simulation complete - no mismatches!!!
$finish called at time : 48000005 ns : File "/home/tuj22026/2613_2020s/lab11/tb_memory_display.sv" Line 143
```

[Figure 3B: Simulation results]

Hardware Implementation (10 points)

My design was checked off by Yamin at 1:04pm on Thursday 4/9/2020 during the lab period.

Conclusion (10 points)

After completing such a big design that was labs 7, 8, and 10, I thought this lab was a nice, easy break compared to previous labs. The syntax was not hard at all, as long as you can understand a block diagram and basic combinational logic, primarily this lab was meant to justify the logic behind a register and I think it definitely helped by giving myself a hands on experience of physically reading and writing memory to a simple circuit board, something our phones, computers, video games, etc. do everyday in less than a nanosecond. Overall, my design was a success and serves as an introduction to memory, which will be continued next week.