ECEN 340
Lab #8

Static Sequential Memories and Tri-State Buses

Purpose:

1. To learn how to design Static Memories
2. To learn how to deal with tri-state (3-state) data buses
3. To learn to trust simulation for sub modules
4. To gain experience understanding Verilog code written by someone else.

Overview:

The objective of this lab is to implement a synchronous memory project that will be used for future projects.

Synchronous memory simply means that it may be accessed at a pre-determined clock rate. Loading the memory in this lab will be done by hand via the switches. Therefore, the second half of this lab will require the use of switches, buttons, and LEDs to interface with the memory.

Procedure:

Part 1. Simulating a single 16 X 16 Static memory.

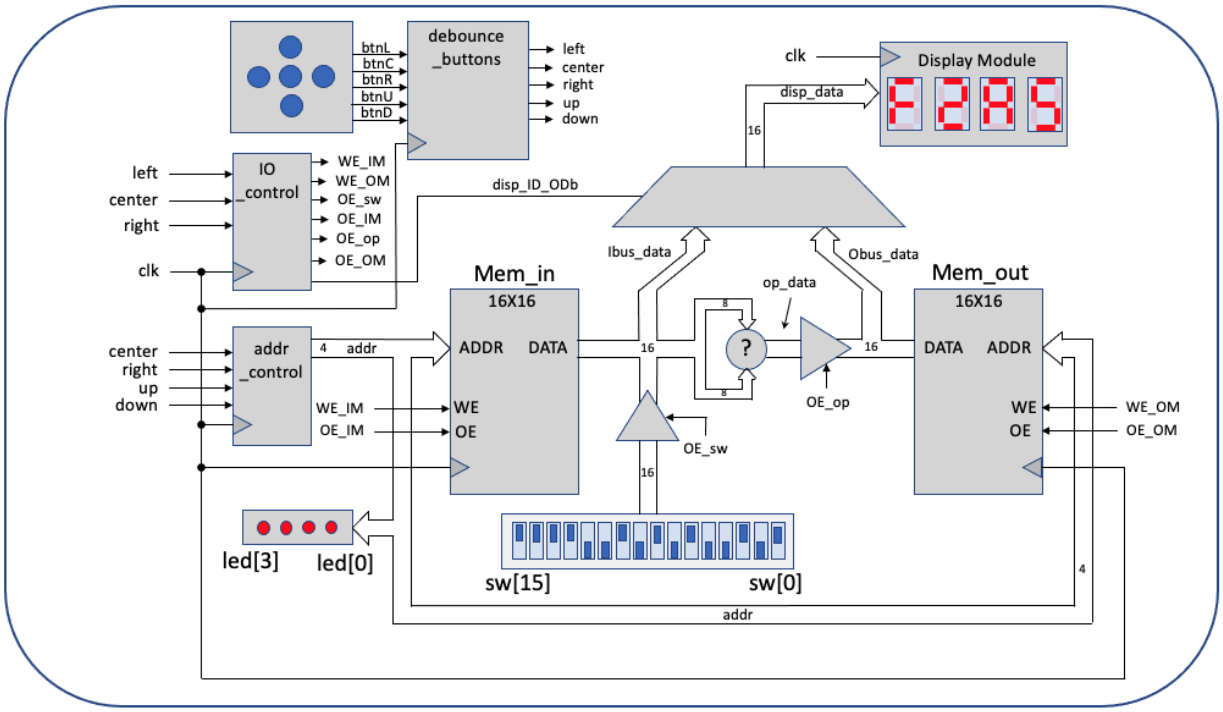
a) Design the 16 X 16 Static RAM module called "memory". Note that the cell ports are: we, oe, data, addr, and clk. The data is accessed via a bi-directional 16-bit bus. See class slides for details on memory arrays (class slides are duplicated below).

b) Write a test bench to verify functionality of the memory (class slides are duplicated below).

Part 2. Implement the complete lab in Vivado. Homework 8 will serve as the foundation for understanding this lab. You will need:

1) Your four digit 7-segment display modules.
2) Your "memory" module
3) The pre-written modules provided.

You will not need to write a test bench for this part of the lab. When the lab is completely implemented, you will observe that the top-level design instantiates two 16 X 16 SRAM cells, 5 push-button debouncers, and the seven-segment design of Lab 6.



Block Diagram of 3-state Memory Lab

Memory Module:

memory.v

C:/Users/young/OneDrive/Documents/- Hardware Labs/digital_systems-verilog/ecen340/Lab8_Static_Sequential_M

```
1  `timescale 1ns / 1ps
2
3  module memory(
4      input clk,              // Clock signal
5      input we,               // Write enable: 1 = write to memory
6      input oe,               // Output enable: 1 = read from memory
7      input [3:0] addr,       // 4-bit address bus (16 locations)
8      inout [15:0] data       // 16-bit bidirectional data bus
9  );
10
11 // Memory array: 16 locations, each 16-bit wide
12 reg [15:0] mem [15:0];
13
14 // Internal register to hold output data
15 reg [15:0] data_out;
16
17 // Connect data output to data bus when reading, otherwise set to high impedance
18 assign data = (oe && !we) ? data_out : 16'hZZZZ;
19
20 always @(posedge clk) begin
21     // Write operation: Store input data into memory at given address
22     if (we) begin
23         mem[addr] <= data;
24     end
25 end
26
27 always @(posedge clk) begin
28     // Read operation: Output memory contents to data bus
29     if (oe && !we) begin
30         data_out <= mem[addr];
31     end else begin
32         // Set data_out to high impedance when not reading
33         data_out <= 16'hZZZZ;
34     end
35 end
36
37 endmodule
38
```

Test Bench Code:

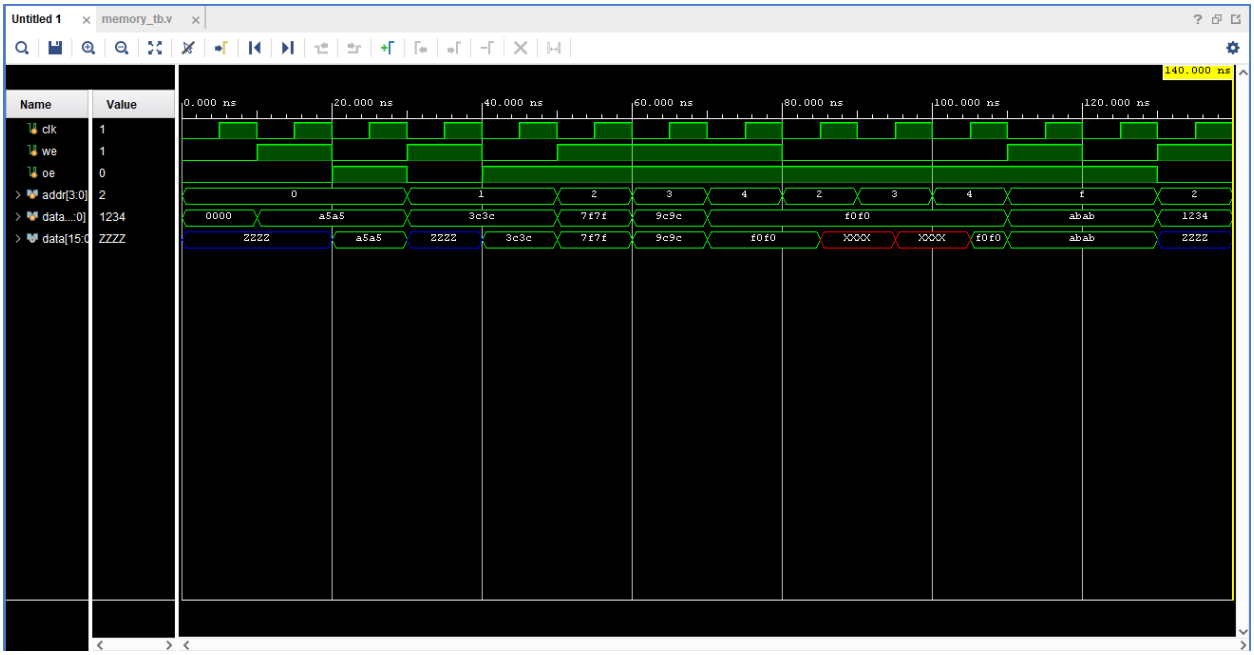
memory_tb.v *

/OneDrive/Documents/- Hardware Labs/digital_systems-verilog/ecen340/Lab8_Static_Sequential_Memories/Static_Sequential_Memory/Static_Sequential_

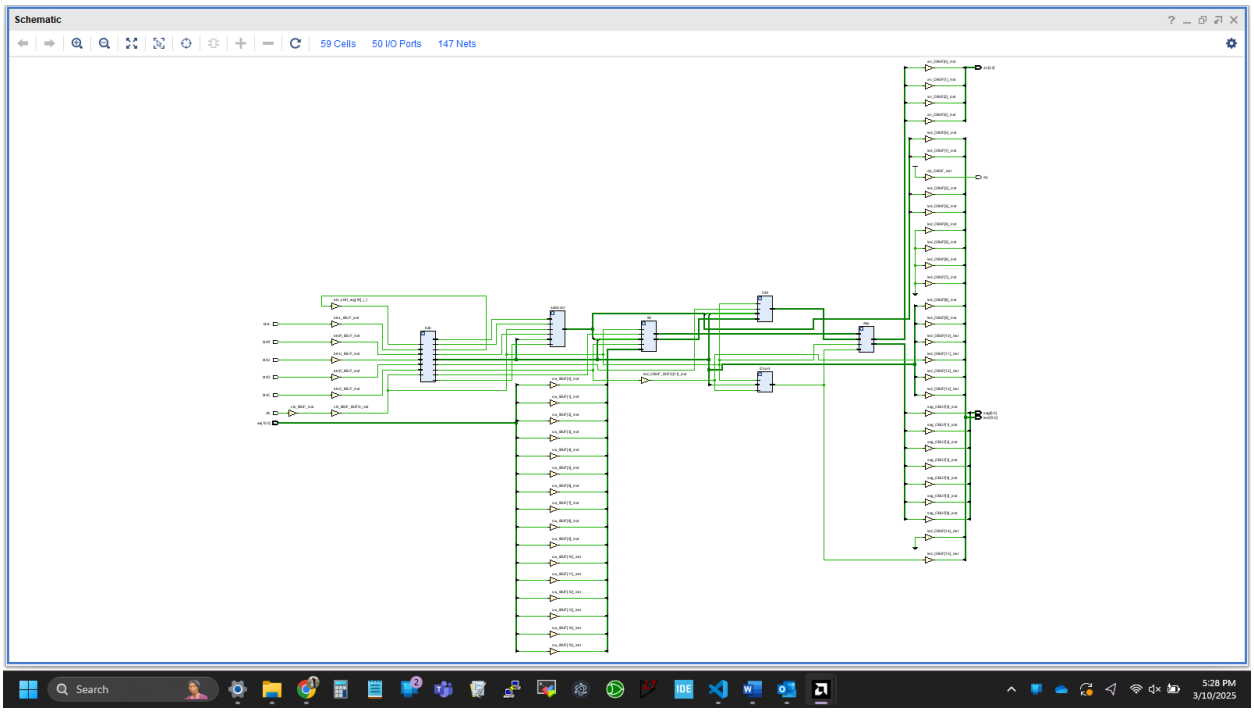
```
1  `timescale 1ns / 1ps
2
3  module memory_tb;
4      // Declare testbench variables
5      reg clk;                // Clock signal
6      reg we;                 // Write enable
7      reg oe;                 // Output enable
8      reg [3:0] addr;         // 4-bit address for memory locations (16 addresses)
9      reg [15:0] data_in;     // 16-bit Data input for write
10     wire [15:0] data;       // 16-bit Data wire for read (inout is now a wire)
11
12     memory uut (
13         .clk(clk),
14         .we(we),
15         .oe(oe),
16         .addr(addr),
17         .data(data)
18     );
19
20     always #5 clk = ~clk;  // 100 MHz clock (period = 10ns)
21
22     initial begin
23         clk = 0;
24         we = 0;              // Write Enable (initially disabled)
25         oe = 0;              // Output Enable (initially disabled)
26         addr = 4'b0000;      // Start with address 0
27         data_in = 16'h0000;  // Start with data value 0x0000
28         #10;
29
30         // Test 1: Write data to address 0
31         we = 1;              // Enable write
32         addr = 4'b0000;      // Set address to 0
33         data_in = 16'hA5A5;  // Set data to write (0xA5A5)
34         #10;                 // Wait 10ns
35
36         // Test 2: Read data from address 0
37         we = 0;              // Disable write
38         oe = 1;              // Enable output (read)
39         addr = 4'b0000;      // Set address to 0 (where we wrote data)
40         #10;                 // Wait 10ns
41
42         // Test 3: Write data to address 1
43         we = 1;              // Enable write
44         oe = 0;              // Disable output
45         addr = 4'b0001;      // Set address to 1
46         data_in = 16'h3C3C;  // Set data to write (0x3C3C)
47         #10;                 // Wait 10ns
```

```
        // Test 4: Read data from address 1
        we = 0;               // Disable write
        oe = 1;               // Enable output (read)
        addr = 4'b0001;       // Set address to 1 (where we wrote data)
        #10;                  // Wait 10ns

        // Test 5: Write data to multiple addresses
        we = 1;               // Enable write
        addr = 4'b0010;       // Set address to 2
        data_in = 16'h7F7F;   // Set data to write (0x7F7F)
        #10;

        addr = 4'b0011;       // Set address to 3
        data_in = 16'h9C9C;   // Set data to write (0x9C9C)
        #10;

        addr = 4'b0100;       // Set address to 4
        data_in = 16'hF0F0;   // Set data to write (0xF0F0)
        #10;

        // Test 6: Read all addresses
        we = 0;               // Disable write
        oe = 1;               // Enable output (read)
        addr = 4'b0010;       // Read from address 2
        #10;

        addr = 4'b0011;       // Read from address 3
        #10;

        addr = 4'b0100;       // Read from address 4
        #10;

        // Test 7: Edge case - Write at maximum address (4'b1111)
        we = 1;               // Enable write
        addr = 4'b1111;       // Set address to the maximum value (0xF)
        data_in = 16'hABAB;   // Set data to write (0xABAB)
        #10;

        // Test 8: Read from maximum address
        we = 0;               // Disable write
        oe = 1;               // Enable output (read)
        addr = 4'b1111;       // Read from maximum address
        #10;

        // Test 9: Verify that no data is output when we is high but oe is low
        we = 1;               // Enable write
        oe = 0;               // Disable output
        addr = 4'b0010;       // Set address to 2
        data_in = 16'h1234;   // Set data to write (0x1234)
        #10;

        $finish;
    end

    // Tri-state buffer control for data line
    // If output enable (oe) is active, drive data_in to data bus, otherwise high-Z
    assign data = oe ? data_in : 16'hZZZZ;
endmodule
```

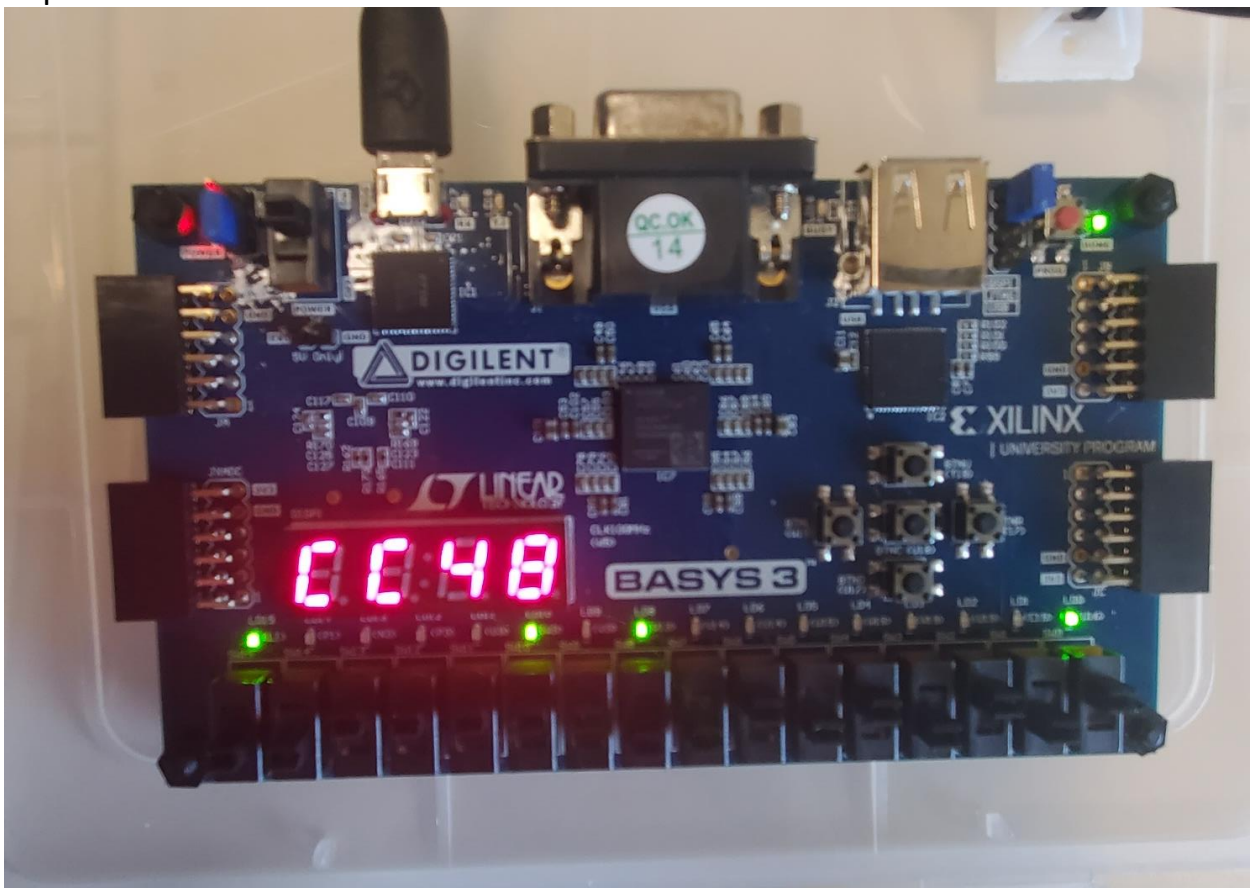
Simulation:



Schematic:



Implementation:



New Tools, Technologies, or Concepts

This lab was mainly about static memories and tri-state buses. One key thing was static RAM (SRAM), specifically a 16×16 memory module, which stores data in a stable manner without requiring refresh cycles. The lab also required the use of tri-state buses, which allow multiple components to share a common data bus by enabling or disabling outputs when needed. We also implemented button debouncing to ensure the signal received from buttons is correct.

Conclusion

We used the testbench to verify the memory module by performing a sequence of write and read operations, ensuring that data was correctly stored and retrieved at specific addresses. Then we programmed the FPGA and tested the memory module with push buttons and verified the seven-segment display and LEDs were what they should be. The system functioned 100% as expected, with data being written via switches and displayed correctly upon retrieval.