ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E

DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	6
EXPERIMENT DATE	:	30.04.2021
LAB SESSION	:	FRIDAY - 14.00
GROUP NO	:	G14

GROUP MEMBERS:

150180112 : ÖMER MALİK KALEMBAŞI

150190014 : FEYZA ÖZEN

150190108 : EKİN TAŞYÜREK

SPRING 2021

Contents

1 INTRODUCTION [10 points]

In this experiment, we implemented data buses and basic memory by using three-state. buffers.

2 MATERIALS AND METHODS [40 points]

2.1 PART 1

In this part, we implemented an 8-bit bus by using three-state buffers. Firstly, we designed a three-state buffer. The Verilog code of the TSB module is shown in Figure1. A three-state buffer is a circuit component that allows input to be transferred into output when Enable input is 1. When Enable is 0, the output is high-impedance. In the code(Figure 1), A is 8-bit input data, E is Enable input, and Z is high-impedance. RTL Schematic of the three-state buffer is shown in Figure 2.

```
module tsb(
   input [7:0] A,
   input E,
   output reg [7:0 ]out
);

   always @(A or E) begin
    if (E) begin
       out = A;
    end
    else begin
       out = 8'bZ;
    end
   end
endmodule
```

Figure 1: Three-State Buffer Module

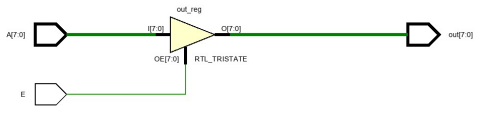

Figure 2: Three-State Buffer RTL Schematic

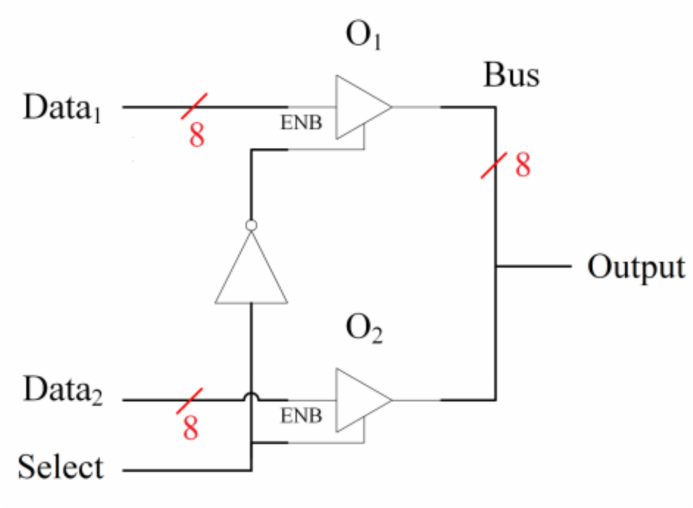

Figure 3: 8-bit data bus with 2 drivers with 3-state buffers

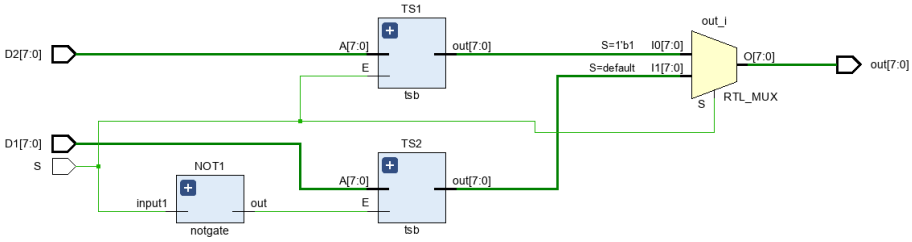

Figure 4: Part 1 RTL Schematic

2.2 PART 2

In this part, we implemented the circuit given in Figure 5. This circuit contains a bus with two distinct outputs and inputs.

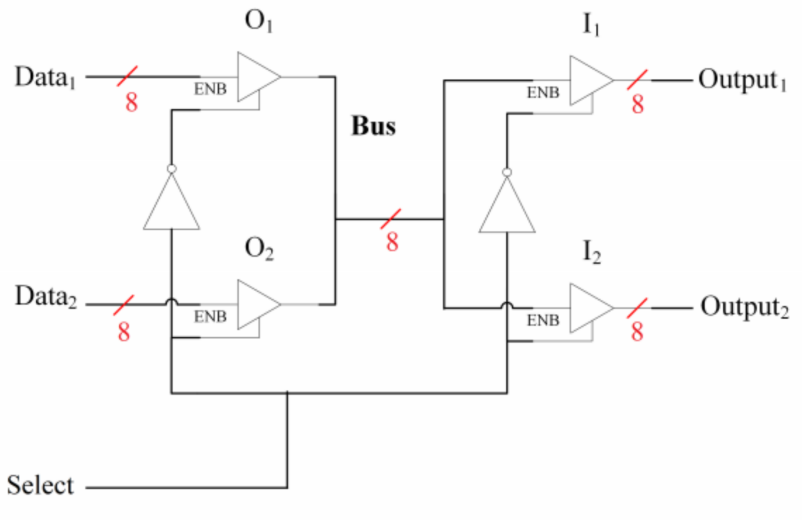

Figure 5: 8-bit data bus with 2 drivers and 2 readers

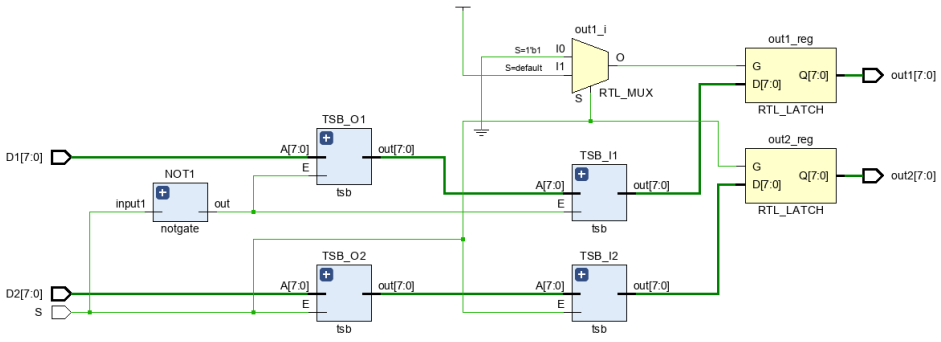

Figure 6: Part 2 RTL Schematic

2.3 PART 3

In this part, we implemented an 8-bit memory line module. This module takes 8-bit data as input. It also takes reset, line select, read enable, write enable, and clock inputs for some operations. It gives 8-bit data as output. Also, the module should take Required operations are given below.

When the clock signal is on the rising edge, Enable and Line Select inputs are high, the module stores the data value.

When the reset signal is on the falling edge, the module clears the stored data.

When read enable and line select inputs are high, the output of the module stores the data. Otherwise, the output is high impedance.

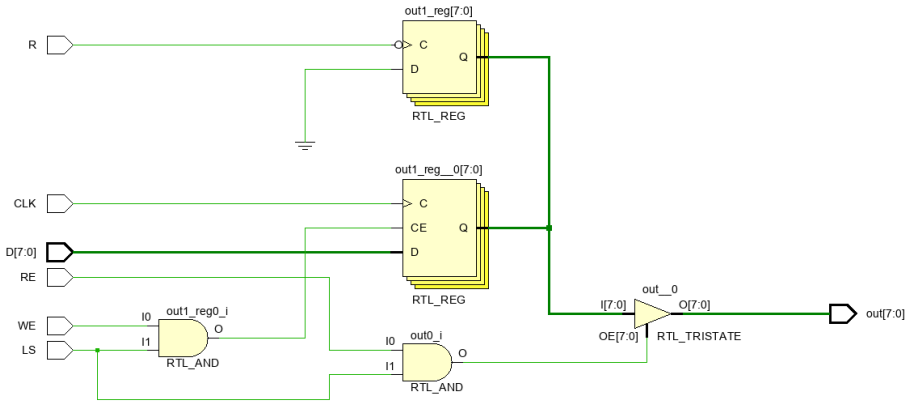

Figure 7: Part 3 RTL Schematic

2.4 PART 4

In this part, we implemented an 8-byte memory module using an 8-bit memory line module. 8-byte memory module takes 8-bit data, 3-bit address, chip select, reset, read enable, write enable, and clock as input and gives 8-bit data as output. When the chip select input is high, the Nth memory line is selected. When the Write Enable input is high and the clock signal is at the rising edge, the selected memory line stores the data. Reset signal provides to clear stored data in the memory lines at its falling edge. When Read Enable is high, the output of the module is the data of the selected memory line.

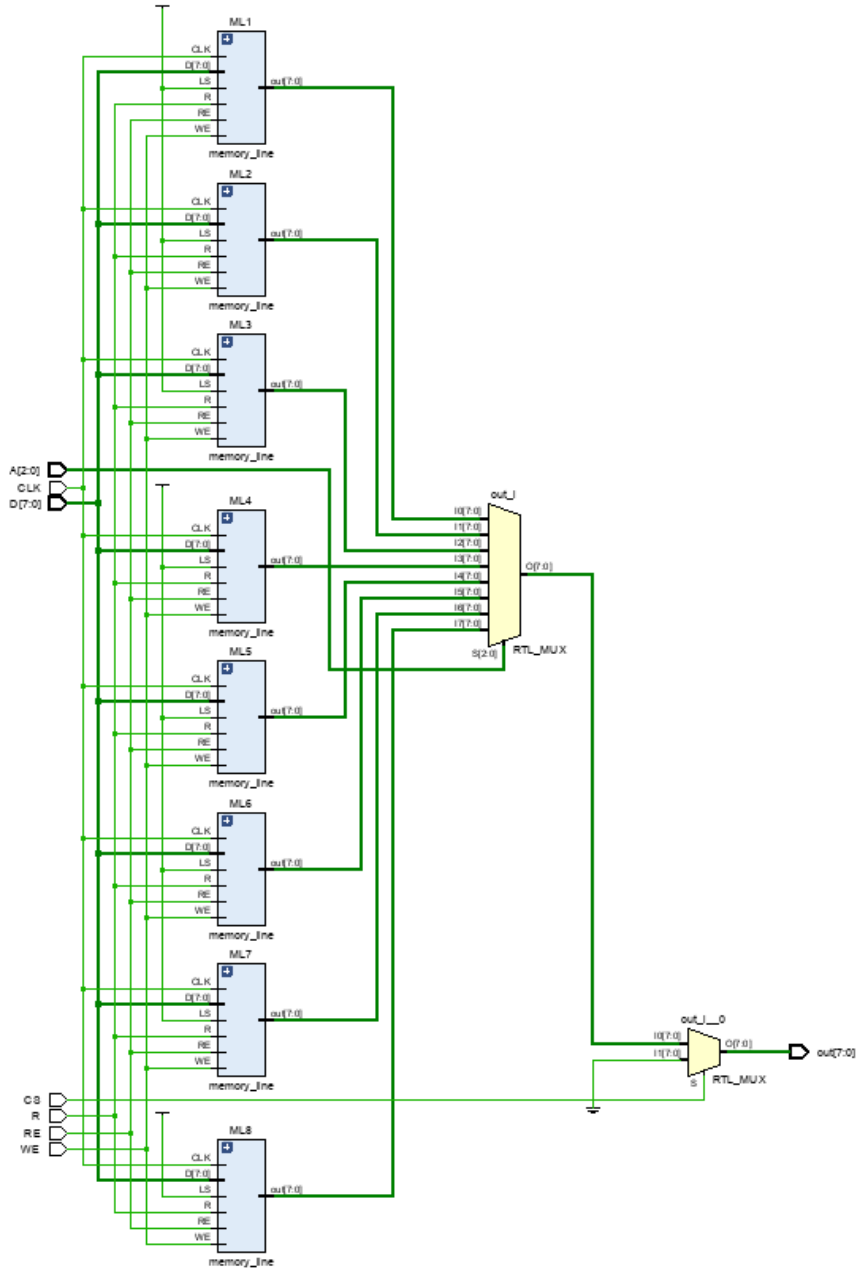

Figure 8: Part 4 RTL Schematic

2.5 PART 5

In this part, we implemented a 32 byte memory module using 8 byte memory module. The Memory module takes 8-bit data, 5-bit address, reset, read enable, write enable, and clock signal as input and gives 8-bit data as output. 2 bits of the address input are used for chip selection and the rest bit is used for selecting line.

When the clock is at rising edge and Write Enable is high, the selected memory line stores the input data value. When read enable is high, the output of the module is the stored data of the selected memory line. Also, there is a reset input that clears stored data in the memory modules when it is at its falling edge.

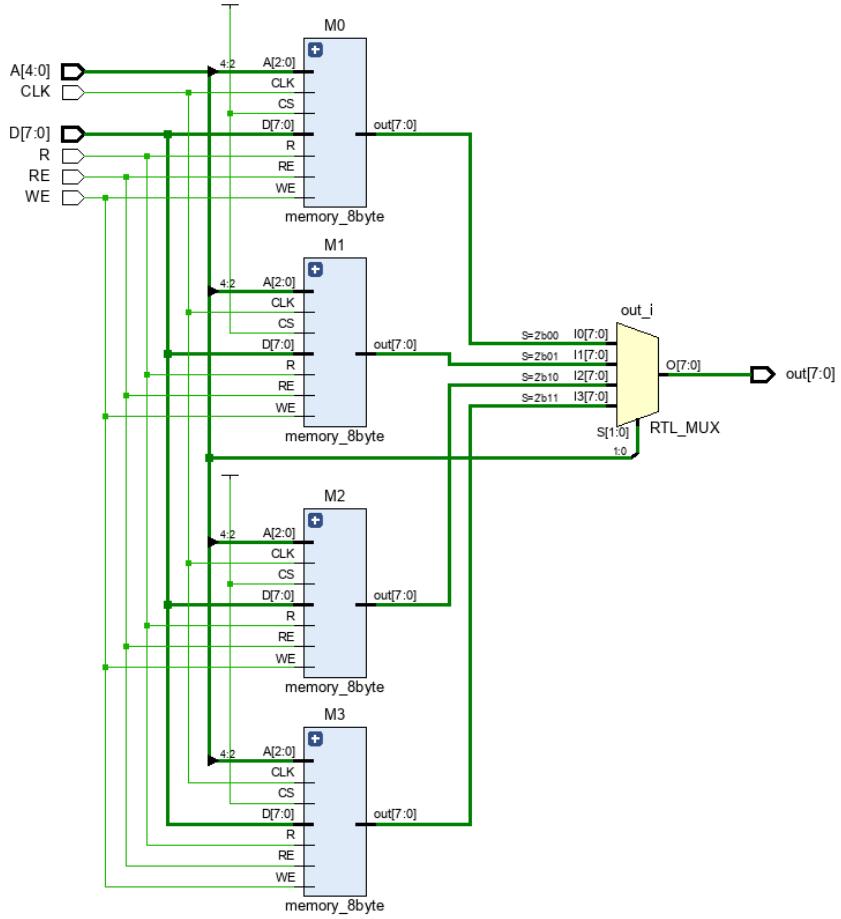

Figure 9: Part 5 RTL Schematic

2.6 PART 6

In this part, we implemented a 128-byte memory module using 32-byte memory module that we implemented before in part5. The memory module takes 32-bit, address, reset, read enable, write enable, and clock as inputs; and gives 32-bit data as output.

When the clock signal is at the rising edge and write enable is high, the selected memory line stores the input data. When read enable is high the output of the module is the stored data of the selected memory line. Also, there is a reset input that clears stored data in the memory modules when it is at its falling edge.

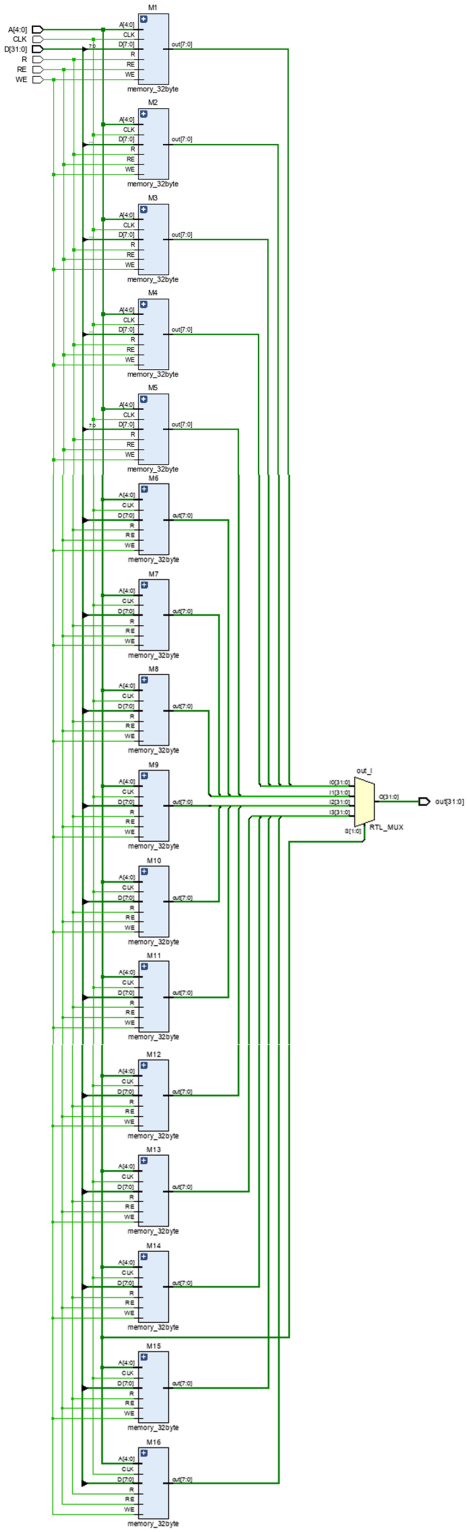

Figure 10: Part 6 RTL Schematic

3 RESULTS [15 points]

3.1 PART 1

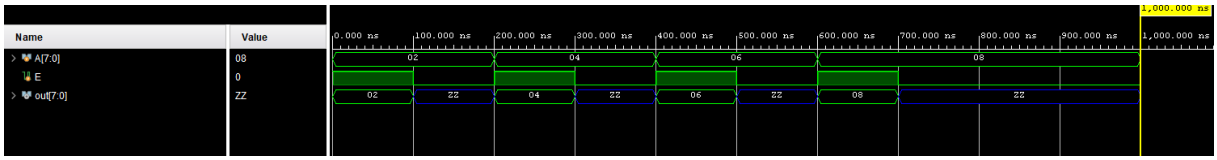

Figure 11: TSB Simulation

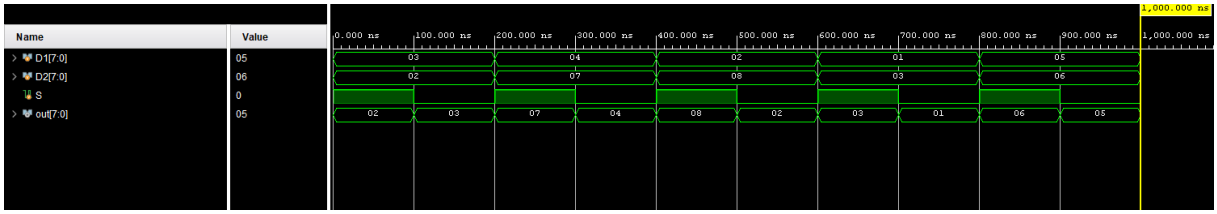

Figure 12: Part 1 Simulation

3.2 PART 2

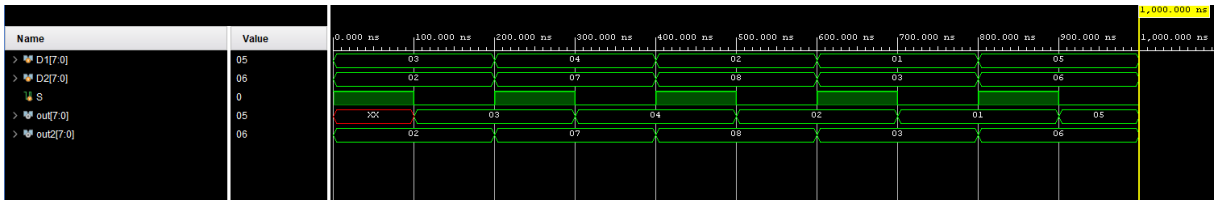

Figure 13: Part 2 Simulation

3.3 PART 3

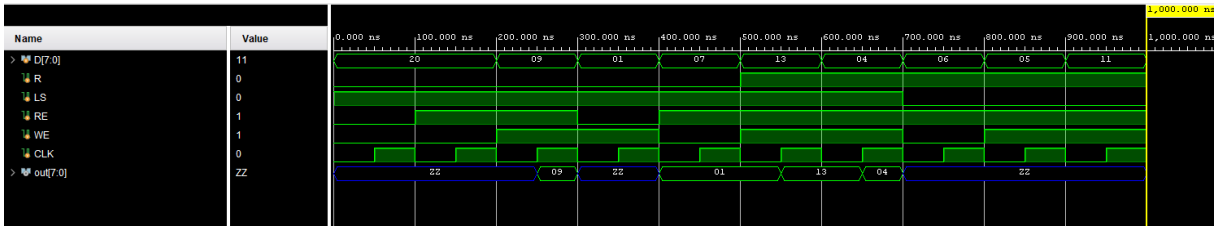

Figure 14: Part 3 Simulation

3.4 PART 4

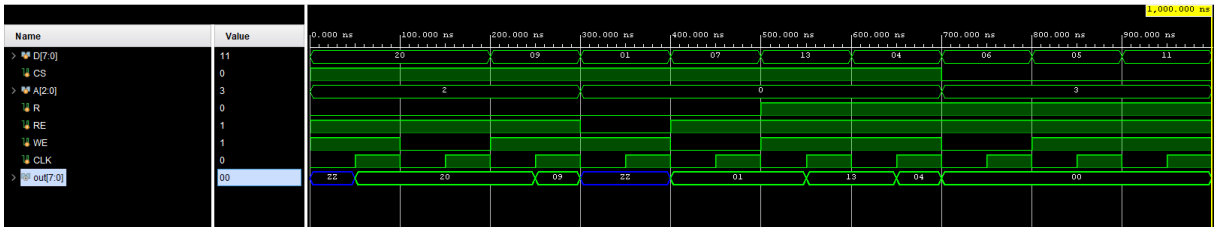

Figure 15: Part 4 Simulation

3.5 PART 5

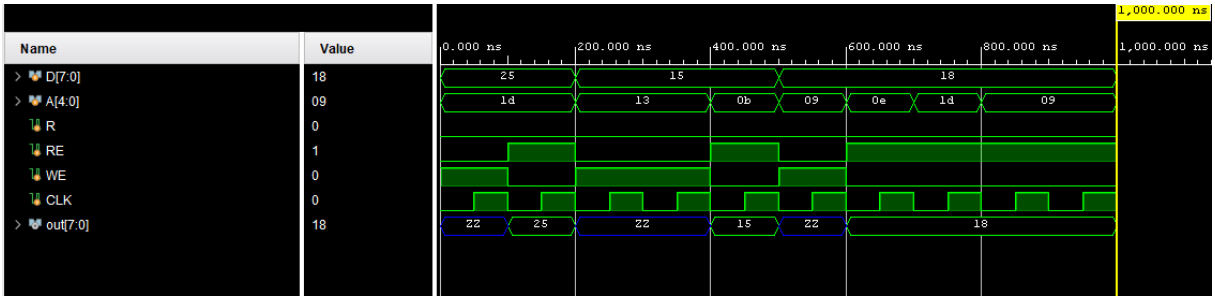

Figure 16: Part 5 Simulation

3.6 PART 6

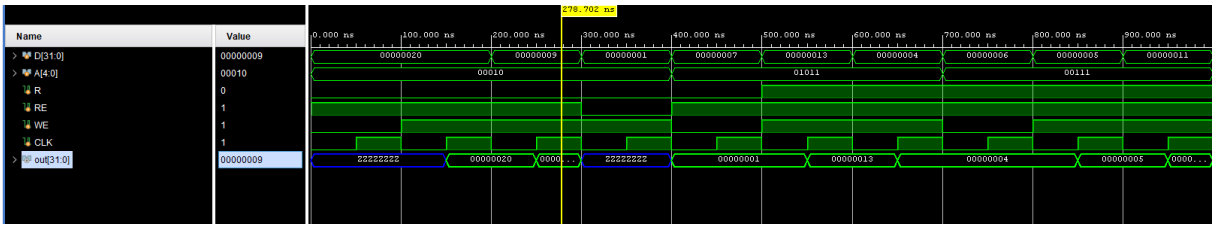

Figure 17: Part 6 Simulation

4 DISCUSSION [25 points]

In Part 1, we implemented an 8-bit bus by using three-state buffers. In our code, there are 2 8-bit data inputs named D_1 and D_2. We tested the output using some test cases. Some of them are; D_1=3 D_2=2, D_1=4 D_2=7. There are also a 1-bit select input S and 8-bit output out, in the code. When S is 1, D_2 is selected. When S is 0, D_1 is selected. The simulation results are the same as what we expected from the circuit.

In Part 2, we implemented the circuit given in Figure 5. In our code, there are 2 8-bit data inputs named D_1 and D_2. We tested the output using some test cases. Some of them are; D_1=3 D_2=2, D_1=4 D_2=7. There are also a 1-bit select input S and 2 8-bit output out1 out2, in the code. When S is at the falling edge, out1 equals D_1. When S is at the rising edge, out2 equals D_2.

In Part 3, we implemented an 8-bit memory line module. In our code, there is an 8-bit data input named D. There are also reset input R, line select LS, read enable RE, write enable WE, and the clock signal CLK. The 8-bit output is named out. We used always block to operate the cases that we need to implement. The first always block is active at the rising edge of the clock signal CLK. We made the circuit to load data D to loaded value, when write enable and line select is 1. The second always block is active at the negative edge of the reset signal R. It resets the loaded value and makes it 0. The third always block is active in "else" situation. If Read Enable and Line Select are 1, loaded value loads to out.

In Part 4, we implemented an 8-byte memory module using an 8-bit memory line module. In our code, there is an 8-bit data input named D. There are also reset input R, line select LS, read enable RE, write enable WE, and the clock signal CLK. The 8-bit output is named out. There are 2 more extra input variables that are chip select CS and 2-bit address A. We used 1 always block and switch case block. The code works when CS equals 1. The code selects and loads the output according to address A. Also, we have an "else" situation that makes output 0.

In Part 5, we implemented a 32-byte memory module using 8-byte memory modules. In our code, there is an 8-bit data input named D. There are also reset input R, line select LS, read enable RE, write enable WE, and the clock signal CLK. The 8-bit output is named out. There is 5-bit address input A. Its first 2 digit is for chip selection, last 3 digit is for line selection. We used the 8-bit memory line module 4 times to get a 32-bit

memory module. We used always and switch-case blocks. The switch case works with the first 2 digits of A. Then, loads needed data to out.

In Part 6, we implemented a 128-byte memory module. We have 6 inputs: 32-bit D for data, 5-bit A for address (first 2 bit is for chip selection, last 3 bit is for line selection), R for reset, RE for read-enable, WE for write-enable, CLK for clock. We used 16 different 32-byte memory modules. We used 4 32-byte memory modules to represent each 32 line because the module memory_32byte has 8-bit input and output. We used case blocks for chip selection just like we did in Part 5. In our test benches for Part 6, we firstly wrote 20 to line 3 (00101:first chip, third line). Then we read the 3rd line. Secondly we wrote 1 to line 3 again and read it. Then we wrote 7 to line 34 (01001:second chip, second line) and read the 34th line. We then wrote 13 and 4 to the 34th line again and read them. After that, we respectively wrote 6, 5 and 11 to line 8 (00111: first chip, eighth line) and read them.

5 CONCLUSION [10 points]

In this experiment we learned to implement data bus using three-state buffer and create memory modules in Verilog. Also we learned some syntactic properties of Verilog, such as switch case. We used negedge for the first time.