ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E
DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	6
EXPERIMENT DATE	:	28.04.2021
LAB SESSION	:	FRIDAY - 10.30
GROUP NO	:	G1

GROUP MEMBERS:

150170908	:	MUHAMMAD RIZA FAIRUZZABADI
070170364	:	AİŞE HÜMEYRA BOZ

SPRING 2021

Contents

1 INTRODUCTION

In this project, we were required to design and implement firstly 3-state buffer, then buses and memory modules. We first learned the concepts of the 3-State Buffer and Bus, then we implemented the required circuit designs and the memory modules according to their respective uses. There was not a preliminary part of the experiment, apart from a short introduction-information regarding the concept of Bus and its implementation using 3-state buffers,

2 MATERIALS AND METHODS

2.1 MATERIALS

Tools Used

- Vivado Design Suite - Xilinx
- Latex (overleaf.com)

Firstly we read and tried to comprehensively understand the concept of the Bus and Three-State buffers. We then implemented each modules required, including the non-part 3-State buffer module itself. Since we were allowed, we do the boolean logic operations using their respective operators in Verilog. We implement and program the logic circuits constructed for this experiment in Vivado Design Suite, and lastly we used overleaf.com to prepare the report document in LaTex. Due to the complexity of the circuits, we did not use Logisim to design the circuits beforehand.

2.2 PART 1

In the first part, we were asked to implement an 8-bit bus, design given in the PDF, by using 3-state buffers. The 3-State Buffers itself was also to be implemented by ourselves separately. We first implemented the 3-State buffer, with an 8-bit input and 1-bit enable input, and an 8-bit output. To accomplish the tri-state buffer operation, we simply use one liner if operation. With output equals to 8-bit input when enable is 1, or Z (as 8'dz) when enable input is 0. The Elaborated Design Schematic of the buffer is as shown in the Figure 1. We then simulated this design according to truth table below. The simulation result is as shown in Figure 2. Lastly, we implemented the Part 1 Circuit design (8-bit data bus with 2 drivers with 3-state buffers) by instantiating the previously designed 3-State Buffer module twice. One for each input, with the first select input being negated. The

Elaborated Design Schematic of the circuit is as shown in Figure 3. Then we simulated the module with several inputs that would show its correctness, shown in Figure 4.

In	En	Out
0	0	Z
1	0	Z
0	1	0
1	1	1

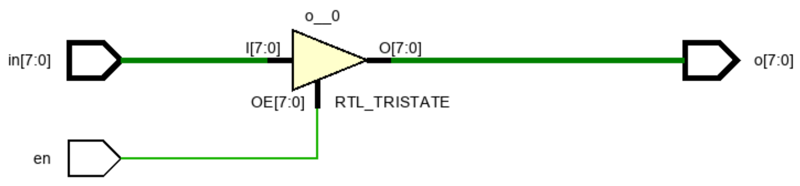

Figure 1: 3-State Buffer Elaborated Design Schematic

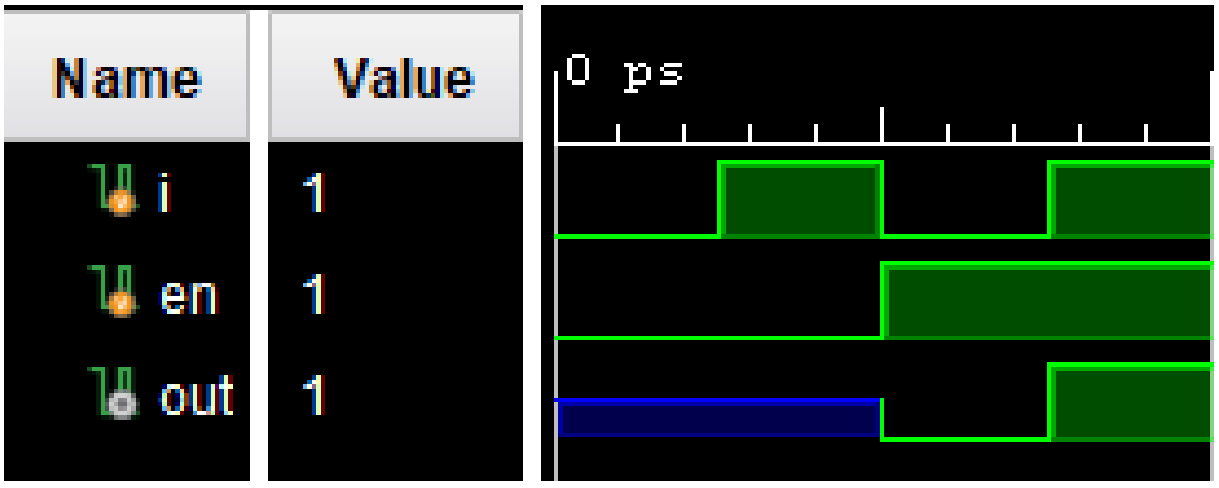

Figure 2: 3-State Buffer Simulation

2.3 PART 2

In the second part, we were asked to design an 8-bit data bus with 2 drivers and 2 readers, which circuit design is also given in the homework questions file. This is slightly similar to the first part, with the main difference being, one having a middle bus between two different part of part 1-like data bus, with also having two outputs instead of two. Seeing the similarity between the two, to implement this circuit we at first instantiate the

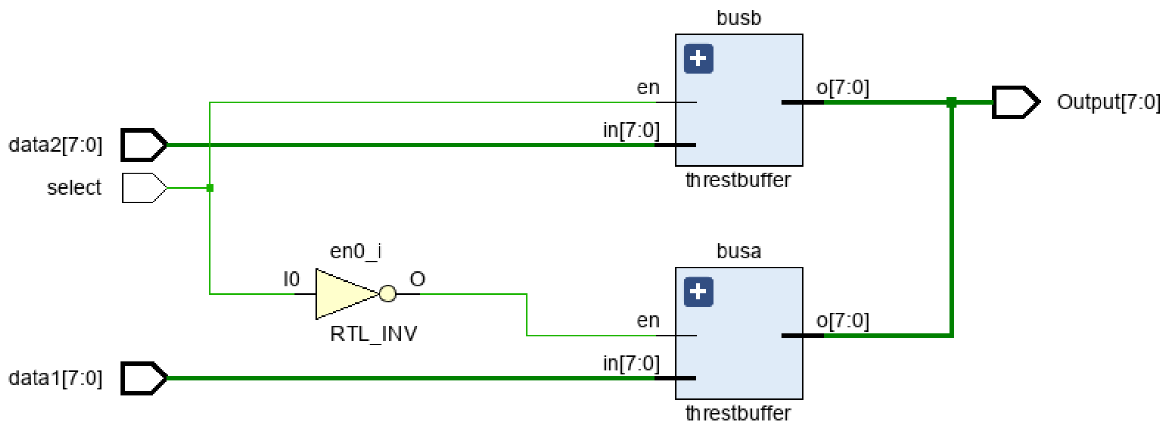

Figure 3: Part 1 Buffer Elaborated Design Schematic

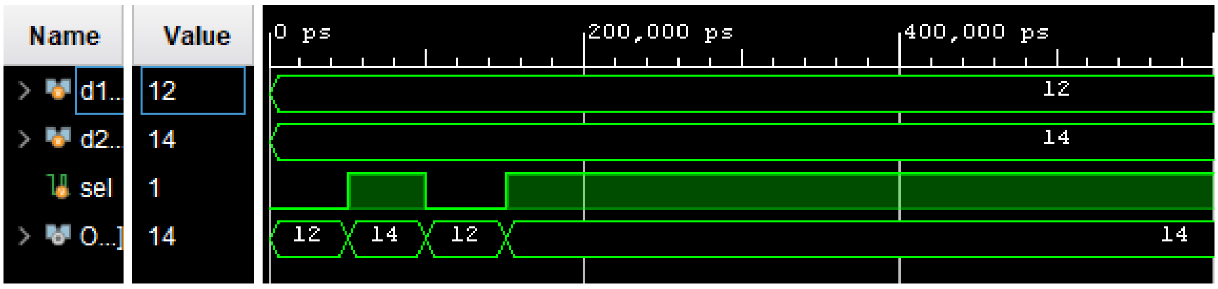

Figure 4: Part 2 Simulation

previously designed part 1, giving it inputs data1, data2, select and aradata as parameters. Aradata is the middle bus mentioned before. We then use that aradata wire value as an input again instantiate the general 3-State Buffer Module twice, with technique the same as in the part 1. Both instantiation of the module gives two distinct outputs of Output1 and Output2. The elaborated design schematic of the circuit is as shown in Figure 5, and again as can be seen in Figure 6, we simulated the module with specific inputs to show its correctness. Any particular simulation sample was not given, therefore it is random, just as in Part 1.

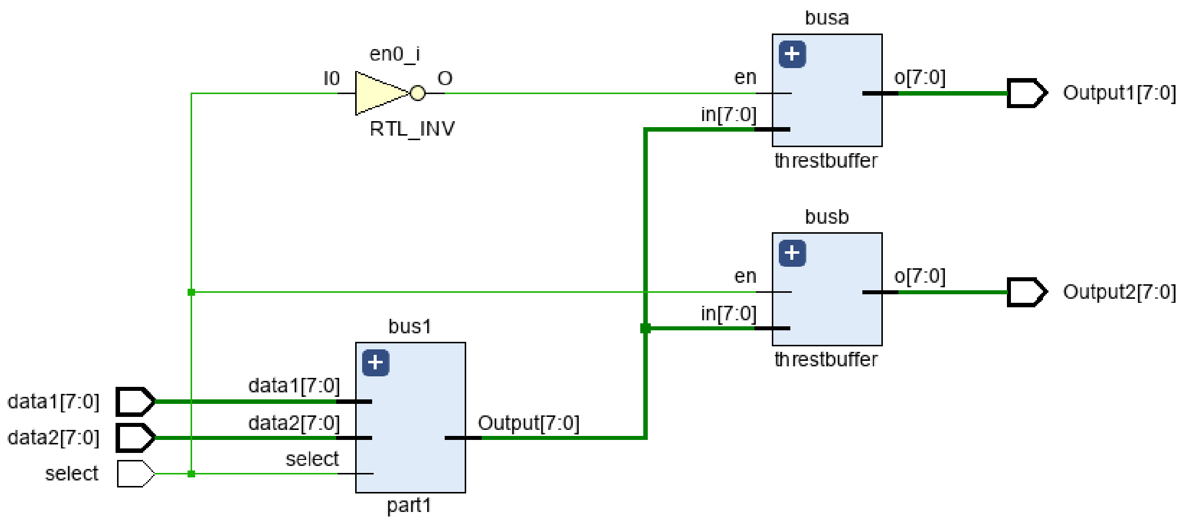

Figure 5: Part 2 Buffer Elaborated Design Schematic

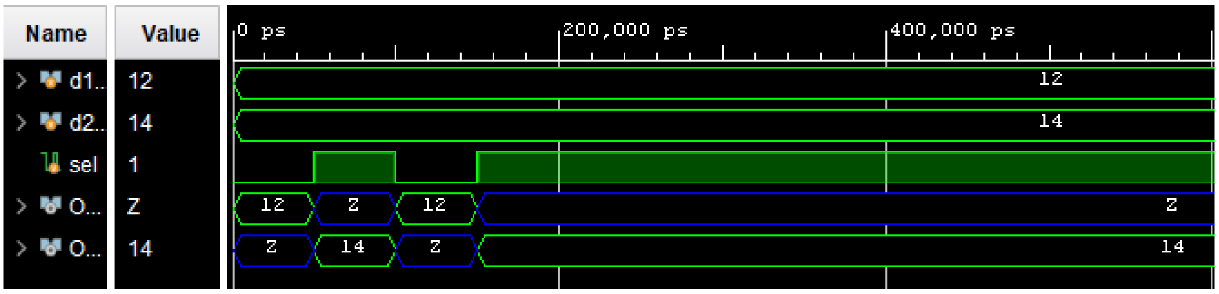

Figure 6: Part 2 Simulation

2.4 PART 3

In the third part, we were asked to implement an 8-bit memory line module, with 8-bit data input and output. It also has some special parameters of reset, line select, read enable, write enable and clock inputs. Those operators determine some particular conditions. Line select is essential here. So, while linesel (our Line select variable name) is 1, write enable is also 1, at the rising edge of the clock signal, then it is store operation. Meanwhile when readen is 1, then the output is read. The output is high impedance Z when read enable is 0. Lastly, all stored data is cleared at the falling edge of the reset signal. To implement all this, we at first created a temp reg variable, to store value inside the upcoming always block before assigning it as parameter later in the instantiation of triset buffer module. We then assign an outen variable, that takes the output value of read enable AND line select. Inside the always block, there is one if statement, that checks if both write and line select inputs are high, to determine the storing condition. Then below the always block we added one other always block to check the falling edge of reset using nededge reset. Inside it we assign the temp variable to 8-bit zero. Lastly, we instantiate our first 3-State buffer using the temp variable as input parameter. Also we use the previous outen variable as the enable parameter as well, so the output would be shown or not according to the read enable, as required. The elaborated design schematic of the circuit is as shown in Figure 7, and the result of our own simulation sample is as shown in Figure 8.

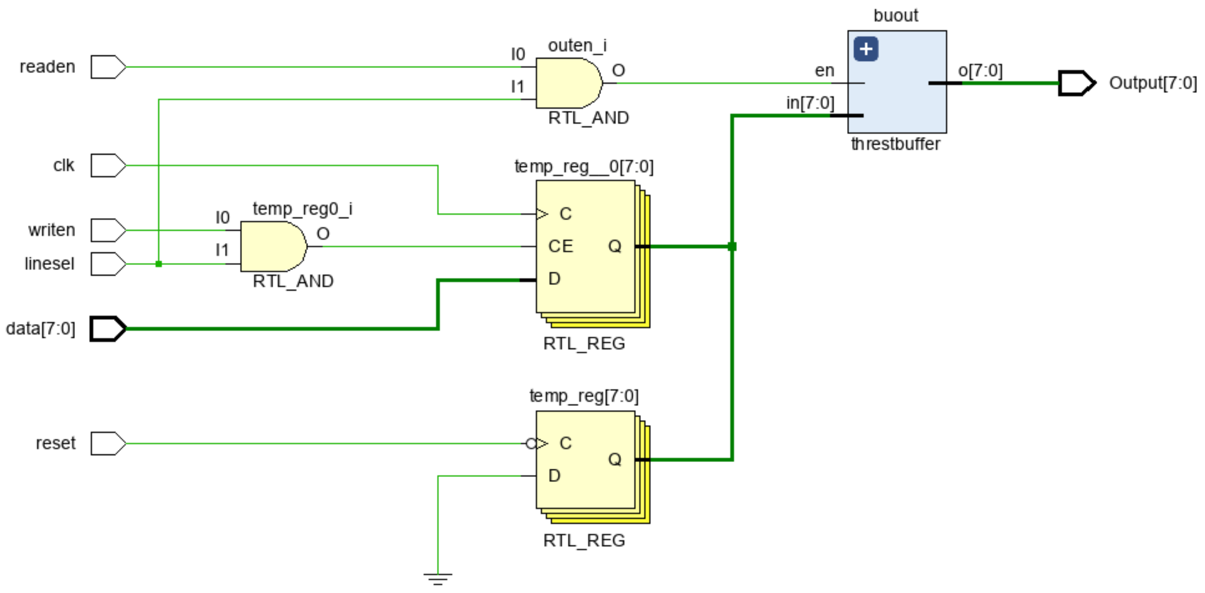

Figure 7: Part 3 Buffer Elaborated Design Schematic

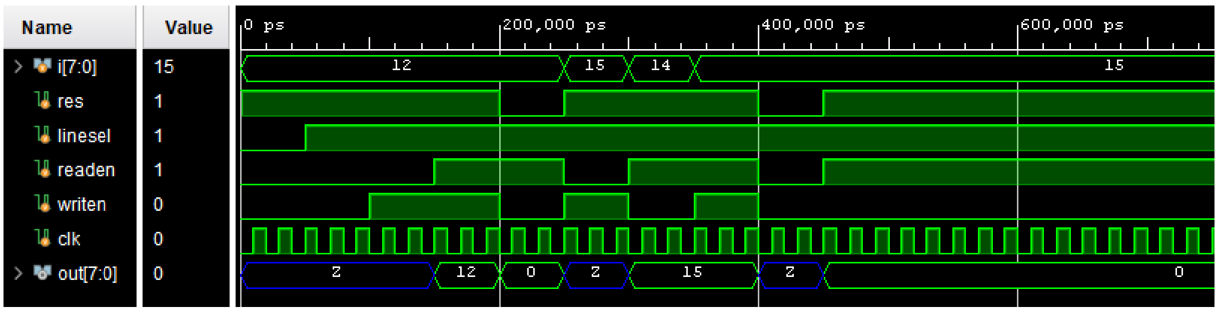

Figure 8: Part 3 Simulation

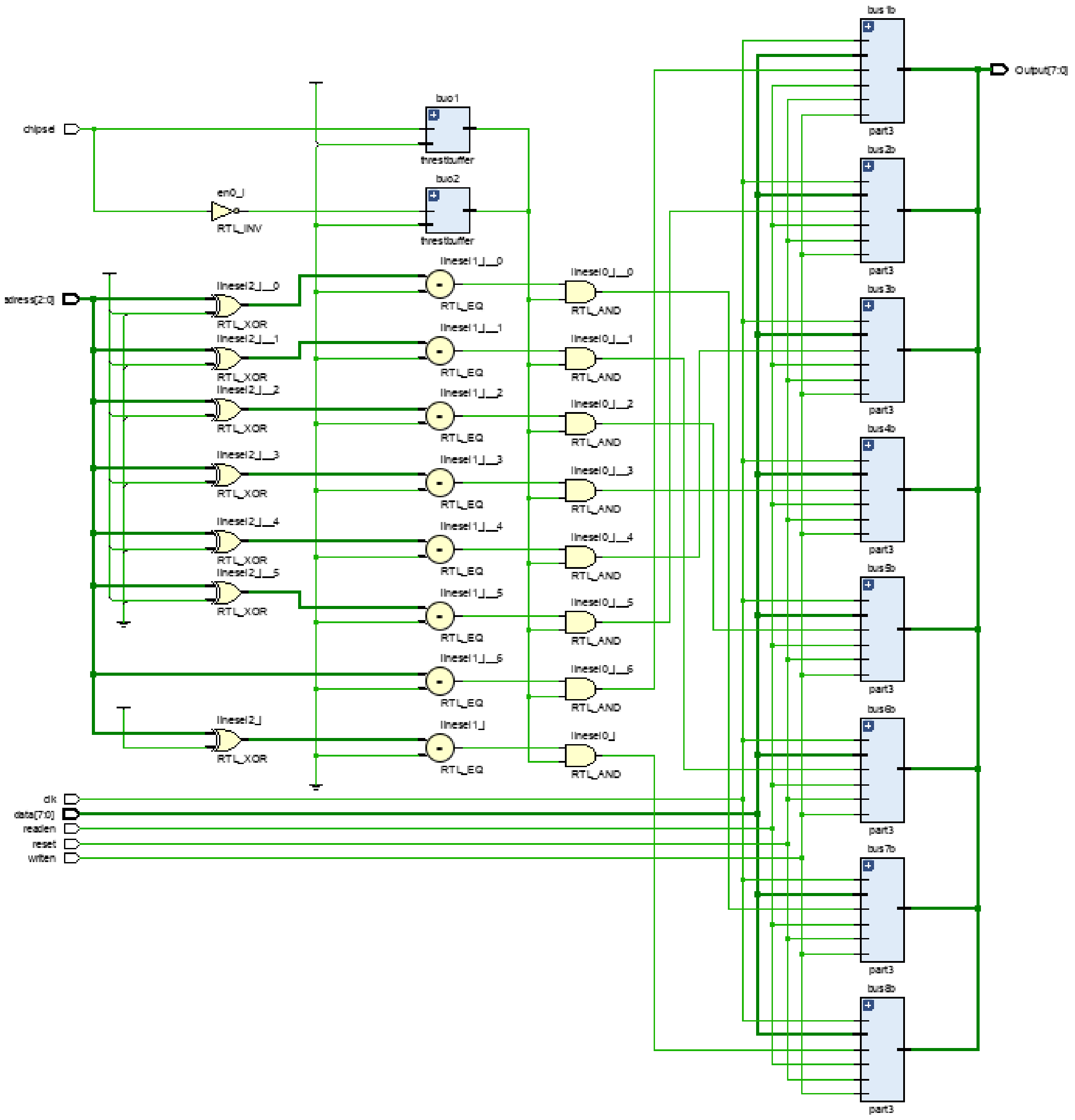

Figure 9: Part 4 Buffer Elaborated Design Schematic

2.5 PART 4

In the third part, we were asked to implement an 8 byte memory module using 8-bit memory line we designed previously. It has similar input and output as previous, in addition to 3-bit address and one chip select input instead of the previous line select. There are some conditions that had to be met, like store when clock is at rising edge plus write enable is high, alongside the previously mentioned reset and read enable dependent output. However, since we used the part 3 circuit within this module, we didn't have to recreate the same condition statements inside always block. We only have to check the chip select input since it is the requirement for the selection of the memory line. Before looking to that, we first had to determine the line select input of the previous part 3. Since we instantiate that module 8 times, we had to give the right values for it. Then we thought of the similar logic as in the part 2. So we made each chipsel parameter of each instantiation different according to each kind of combination it may be. From 0 to 8 in decimal. However, we also had to choose whichever of them are actually is the adress input. So to check that, we use XOR. We did the XOR operation between the adress input and each combination between 0 and 7, from that we can get that if it equals to 0, that both are the same value. Also with this technique we could get 1-bit input parameter we need, instead of 3. Then we did an AND operation between the result of that operation with the value of chip select, so the selection could only be valid only when it is high. That concludes our implementation. In addition, to prevent having a MUX in our design, we use two 3-state buffer to the chip select input instead of if statement. The elaborated design schematic of the circuit is as shown in Figure 9, and the result of our own simulation sample is as shown in Figure 10.

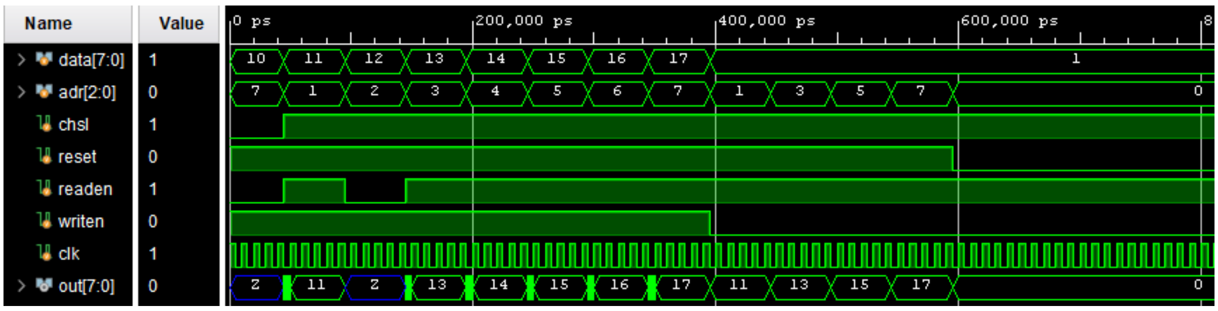

Figure 10: Part 4 Simulation

2.6 PART 5

In the third part, we were asked to implement a 32 byte memory module using the 8 byte memory module we implemented before. Again due to the similirity between the two, we used the same logic as before to implement this. Since it is 32 byte using 8 byte modules, we instantiate the module of part 4, four times. We take the most significant three bit of the adress input as the part 4's adress paramenter input, and we take the other bits of [4:3], check each of its equality with the combination of 00, 01, 10 and 11 with XNOR and a == equal operation as before, we use its output as part 4's chip select input parameter. And again, the store, read, reset operations are made in part 3 which is a section of part 4 that we instantiated here, so we did not implement those operations here as well. That concludes our implementation design. The elaborated design schematic of the circuit is as shown in Figure 11, and the result of our own simulation sample is as shown in Figure 12.

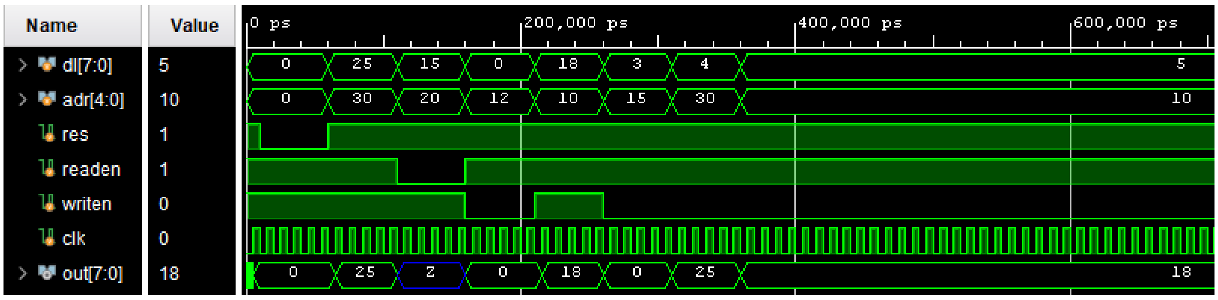

Figure 11: Part 5 Buffer Elaborated Design Schematic

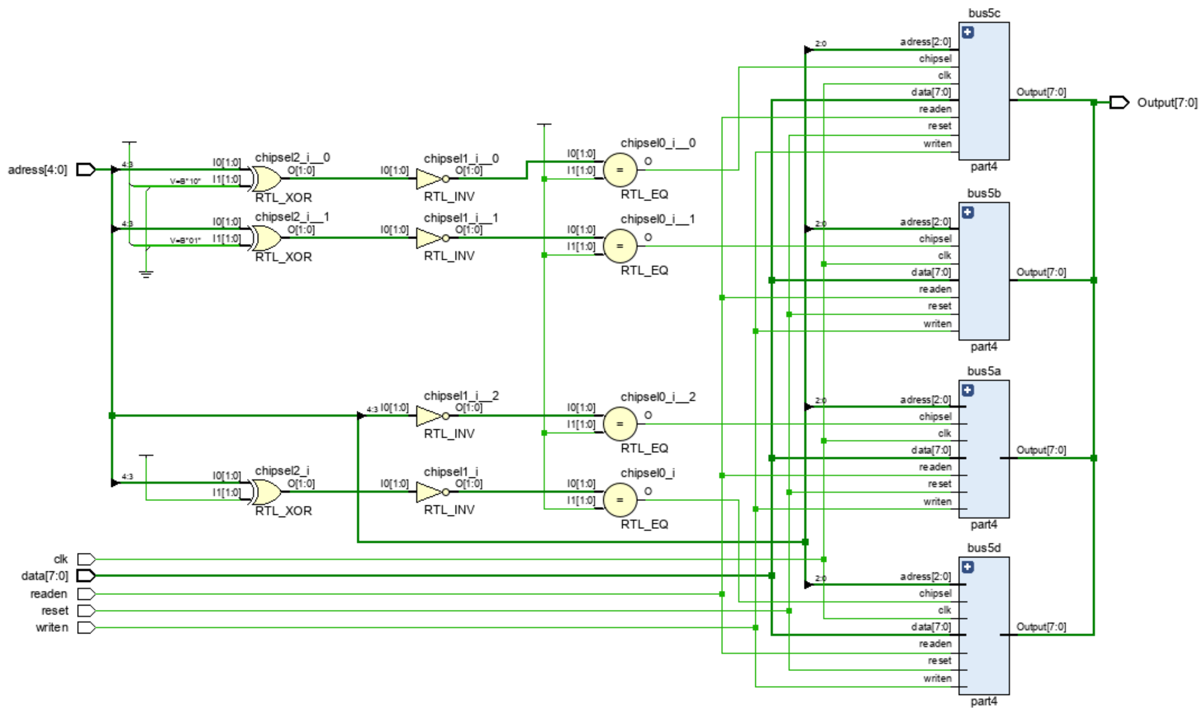

Figure 12: Part 5 Simulation

2.7 PART 6

In this last part, we were tasked to implement a 128 byte memory module using 32 byte memory module. It has 32-bit input and 32-bit data output, with the previously seen reset, read enable, write enable and clock inputs. We also did not implement the store, reset and read conditions inside this module since it is implemented in the previous parts' module. Thanks to part 5 not having a kind of line or chip select inputs, this module is rather simple. The only thing to be considered was that part 5 32 byte memory module has only 8-bit input and outputs. Therefore, we instantiated that 4 times, with input data being divided to 4. 0-7, 8-15, 16-23, and 24-31. We store each output of the instantiations to 8-bit wires outa, outb, outc, and outd. Lastly, we contatenate all of them and assign it to O, concluding our implementation of this part. The elaborated design schematic of the circuit is as shown in Figure 13, and the result of our own simulation sample is as shown in Figure 14.

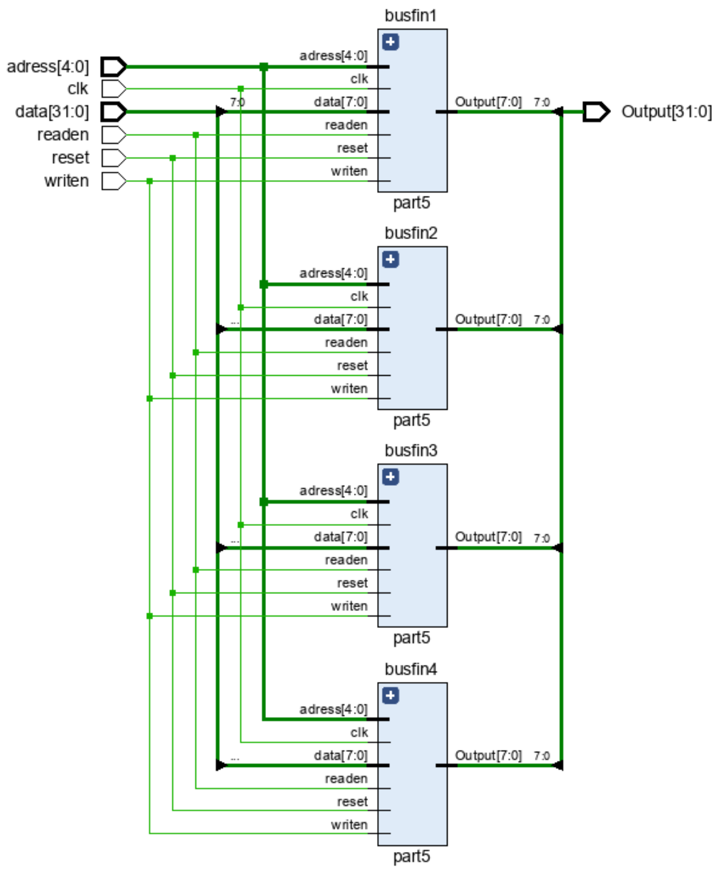

Figure 13: Part 6 Buffer Elaborated Design Schematic

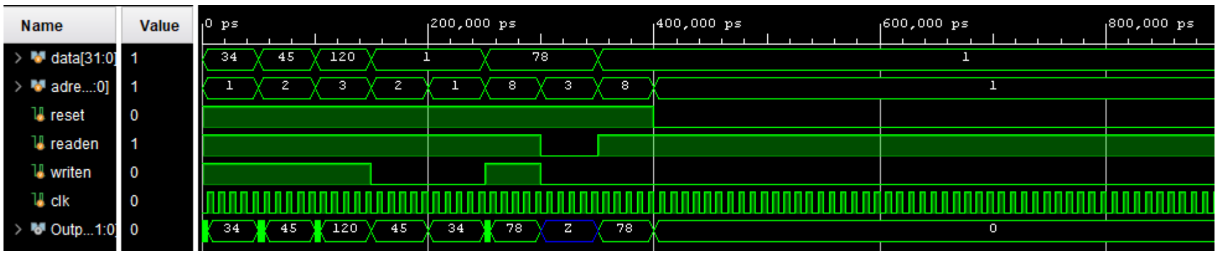

Figure 14: Part 6 Simulation

3 RESULTS

We completed the all the tasks we were given within this experiment 6. The results of the simulations are shown in the materials and methods part to prevent the deformation of the layout of the report. We did not design the circuits before the implementation in software such as Logisim, but due to the clear usage/objectives of each part, we were able to validate their correctness by setting the appropriate simulation samples/codes. As in the previous experiments, we supplemented our images namely elaborated design schematic, and simulations to the report for each part inside the Materials and Methods section.

4 DISCUSSION

As mentioned in the introduction part, in the beginning we recalled the concept of bus and 3-State Buffers, and their 'role' for one another. Being not a preliminary part, we were not tasked to do any specific implementations in that part other than to gain required understanding. Then later in the experiments' main parts, we implemented the 3-State Buffer at first, then Part 1 using that, Part 2 using Part 1, Part 3 using the 3-State Buffer module, Part 4 using Part 3 and the 3-State Buffer module, Part 5 using Part 4, and part 6 using part 5. What we mean by 'using' here is that we instantiated those particular modules within their pair. All Boolean logic operations are done with verilog default operators. We use always/code blocks when required since it was allowed. The experiment, even though rather simple in its core, was rather time consuming due to limitation of having Multiplexer inside the elaborated design. We thought that the main reason behind that restriction is to give us better understanding of the concept of bus and 3-state buffer to implement especially the memory modules where outputs are selected/able. However we think that other than that, the total restriction of the usage of MUX may be unnecessary. Since according to our general experience in implementing this experiment, MUX could appear out of any kind of operation and were significantly hard to get rid of in some occasions.

5 CONCLUSION

We successfully implemented the modules and simulated them using suitable samples to validate their correctnesses. All in all, we learnt the concept of Bus, how to implement them with 3-State Buffers, and then we learned how to design and implement the memory (line) modules of different capacity up to 128 bytes. Again, getting rid of MUX was challenging from time to time, however we managed to overcome it.

REFERENCES

[1] LogicLab. Logic lab. *An example journal*, 22(4):10–16, February 2020.

[2] Overleaf documentation https://tr.overleaf.com/learn.