ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E
DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	7
EXPERIMENT DATE	:	06.05.2021
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 asked to design and implement Register Line, Decoder, Register File Design, ALU, Instruction Decoder, Program Counter, and a Mini Computer. We first learned the concepts of the each design, 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, and no restrictions are given.

2 MATERIALS AND METHODS

2.1 MATERIALS

Tools Used

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

Firstly we read and tried to comprehensively refresh our knowledge regarding the designs given. We then implemented the modules required, along with their respective appropriate simulations. Since we were allowed, we do the Boolean logic operations using their respective operators in Verilog. Since also there was no restriction regarding the use of any kind of gate within the module, we were free to implement our designs algorithmically. 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 required to implement an 4:16 decoder with enable input. If enable signal is logical low, all the outputs of decoder will be logical low. There was a slight issue in this part of this experiment. We are not sure if it is exactly an issue, but since the 4:16 decoder is normally a design with 4 input and 16 output, in this experiment we had to make it as one 4-bit input and one 16 bit input. Logically it is still a decoder, since it does what it is supposed to do successfully, however when we see the elaborated design schematics of the implementation. It is as seen as in Figure 1. It became a sequences multiplexer. Not because it is a multiplexer, but it is due to the design having only one output, even though it is 16 bit. Anyhow, our implementation strategy was simply to assign the output with values according to the 4:18 decoder's typical truth table. Enable is also taken into consideration with an AND operation, so the value is taken only when Enable is 0. Simulating the module, we got a correct implementation. It is as shown in Figure 2. Then, we were asked to implement a 16-bit register line module. The module takes lineselect, clock, reset, and 16-bit dataIn input and 16-bit storedata output. There were conditions such as the data is cleared when the falling edge of the reset signal, and new data is stored at the rising edge of clock signal when lineselect is high. We implemented this similar with the previous experiment's memory module, with the only different is we didn't use tristate buffer to store. The elaborated design schematics of the implementation is as shown in Figure 3. We simulated the design with some inputs that would prove its correctness, as shown in Figure 4.

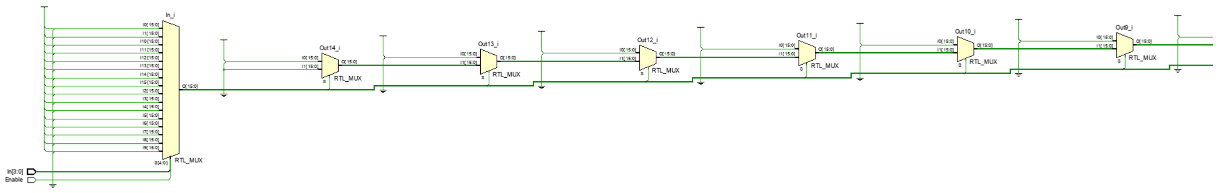

Figure 1: 4:18 Decoder Elaborated Design Schematic

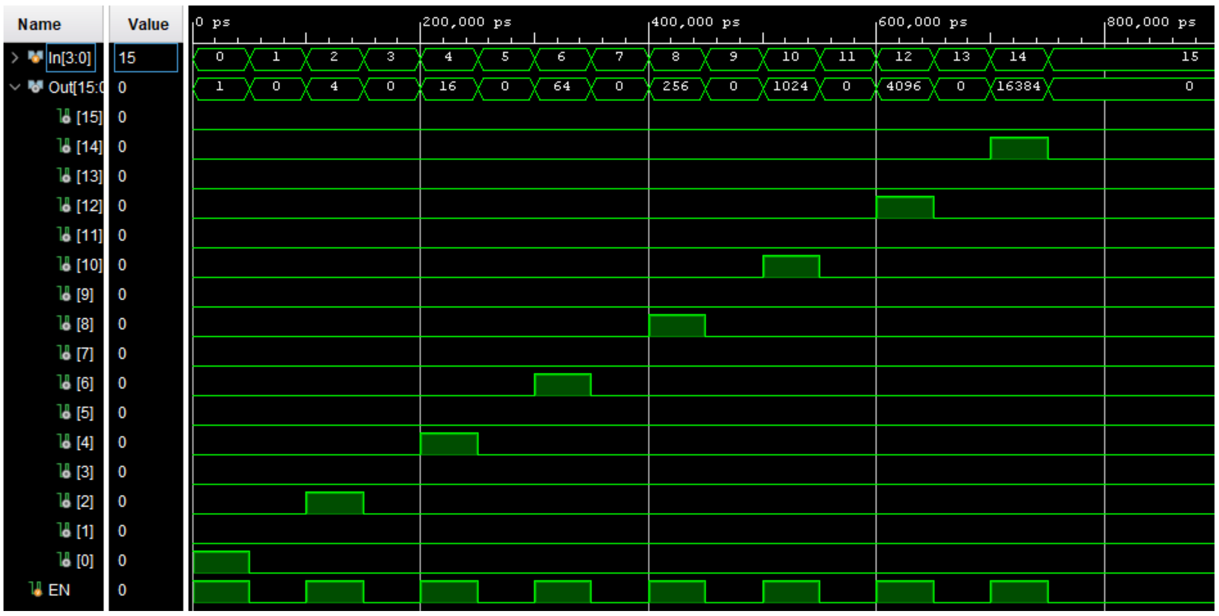

Figure 2: 4:18 Decoder Simulation

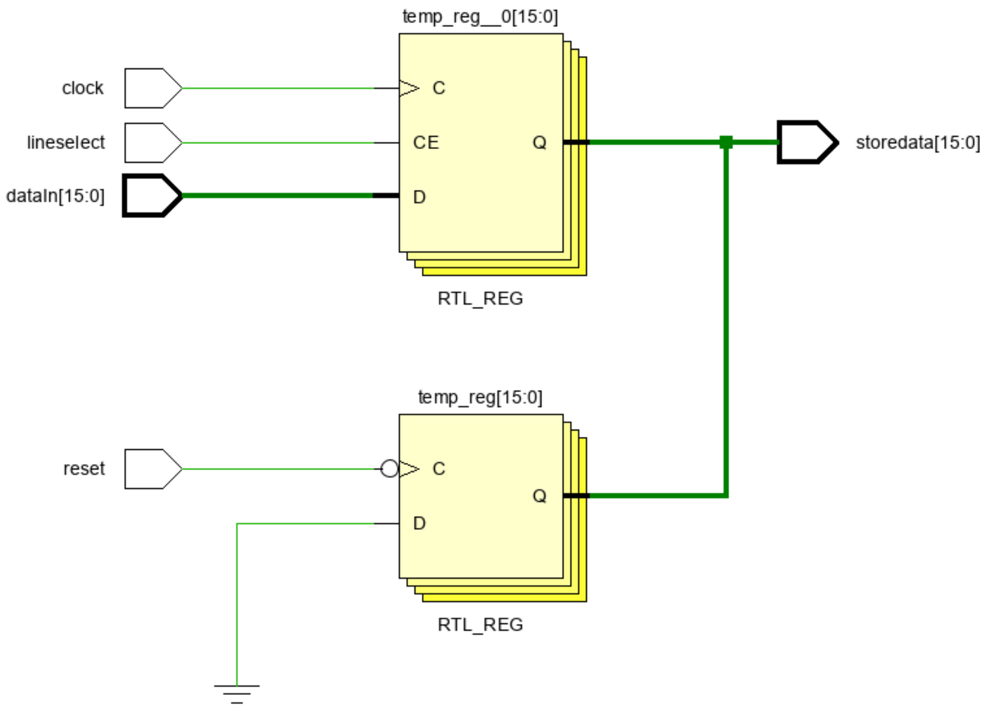

Figure 3: Part1B Elaborated Design Schematic

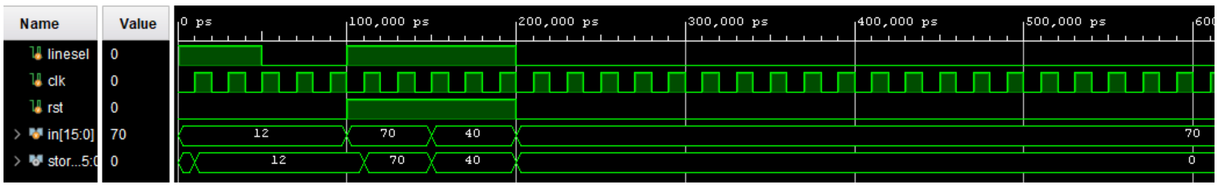

Figure 4: Part1B Decoder Simulation

2.3 PART 2

In the second part, we were asked to design a a 16 line 16-bit register file module (32 byte) using 4:16 decoder module and 16-bit register line module. This module should take 4-bit selA, 4-bit selB, 4-bit selWrite, 16-bit dataIn, reset, writeEnable, and clock as input and give 16-bit dataA and 16-bit dataB. The conditions such as store when clock signal's rising edge and reset at reset signal's falling edge is the same as in the Part 1, so we did not implement them here. We implemented only the last condition that is the register outputs are exported to dataA and dataB after selected by selA and selB inputs. We implemented this design by first decoding the selWrite input so we can better use each line for the 16 line of the register. We use that value to determine the state of lineselect parameter that goes to the Part1's Register module, alongside with the WriteEnable set. Instantiating the Part1 Register module 16 times, we store the data to an 16-bit array of aradata for each line. Then at last, we assigned aradata's selected line (ex. aradata[selA]) to dataA and dataB, respectively. The Elaborated Design Schematic of the design is as shown in Figure 5. The simulation of the design is also shown below in Figure 6.

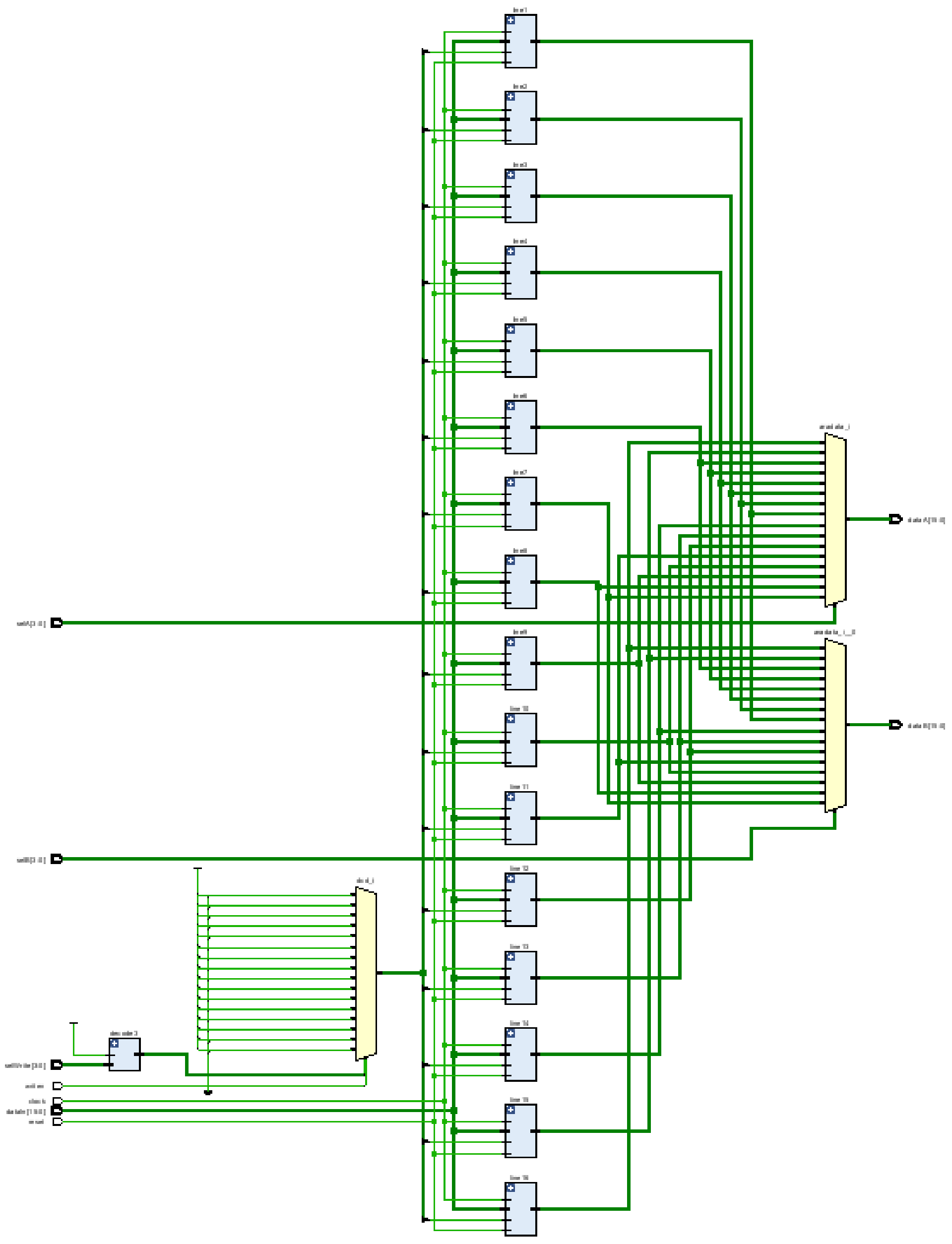

Figure 5: Part 2Elaborated Design Schematic

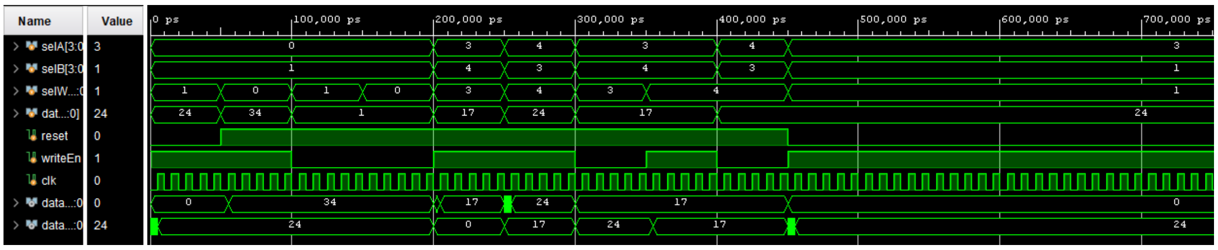

Figure 6: Part 2 Decoder Simulation

2.4 PART 3

In the second part, we were asked to design an ALU. It does some operations that is shown in Figure 7. The inputs of the ALU will be called srcA, srcB. The output will be called dst. The ALU will decide its operations with the help of 3-bit Op input. There is also an addition feature of zeroflag, that updates when the result of the ALU's operation of dst is 0 when it is either Addition or Subtraction. At the falling edge of the reset zeroFlag will be cleared. Taking all these into consideration, we implemented the design with two always blocks, one for positive edge clock and one for negative edge reset. In the first one we did the the operations, assigning the result of the operations to dst reg output. Meanwhile, we have two temp values of 'temp' and flag. Temp is used to store the updated/or undated zeroflag value before assigning it after the block. Flag is used to check whether the operation being done is chaning the zeroflag value or not, so when flag is 1, and when the dst or the result of the operation is 0, zeroflag is risen. The Elaborated Design Schematic of the design is as shown in Figure 5. The simulation of the design is also shown below in Figure 6.

OPCODE	**Operation Name**	**Operation**	**Update Flag**
0	AND	dst = srcA & srcB	No
1	OR	dst = srcA | srcB	No
2	Addition	dst = srcA + srcB	Yes
3	Subtraction	dst = srcA - srcB	Yes
4	XOR	dst = srcA ˆ srcB	No
5	Logical Shift Right	dst = srcA >> srcB	No
6	Logical Shift Left	dst = srcA << srcB	No
7	LOAD	dst = srcB	No

Figure 7: ALU Instruction Format

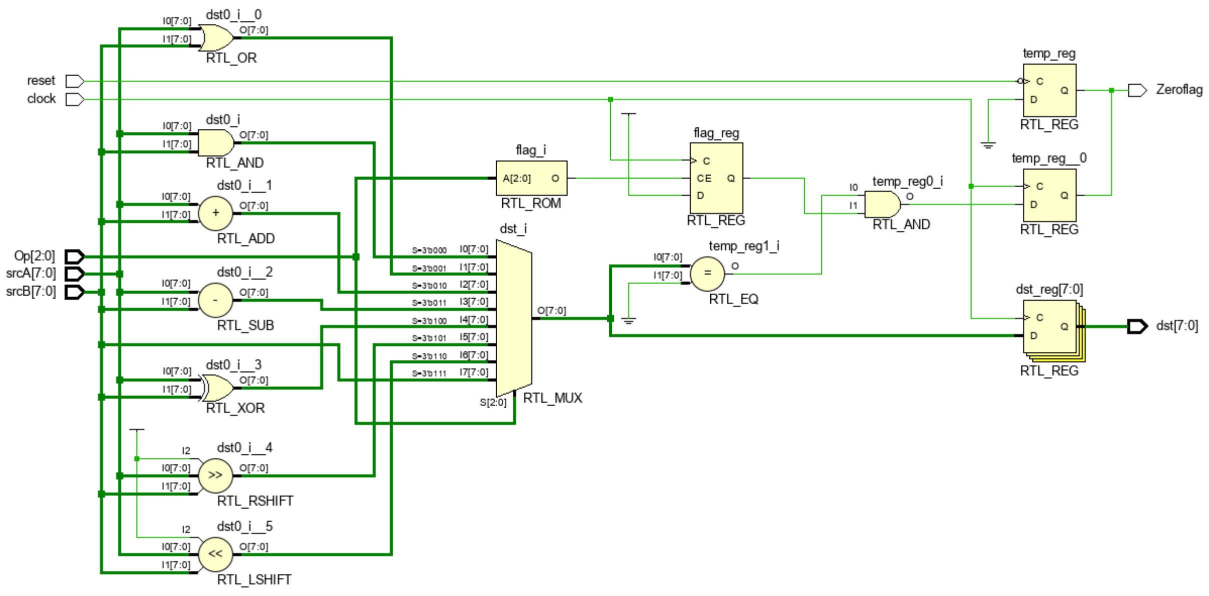

Figure 8: Part 3 ALU Elaborated Design Schematic

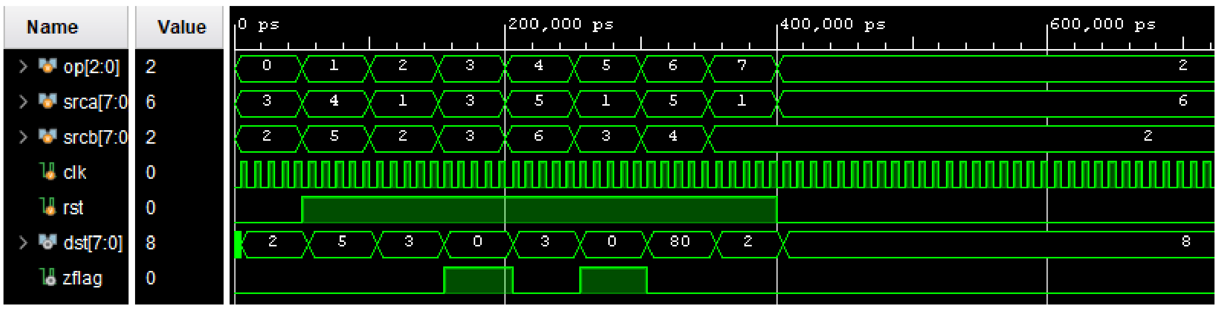

Figure 9: Part 3 ALU Simulation

3 RESULTS

We completed only 3 out of 6 we were given within this experiment 7. Due to the high amount of exams and homeworks for the other classes we take this week, especially between this week's Monday and Wednesday, we unfortunately could not get enough time to complete the whole experiment. 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 the designs of decoder, register line and others we were asked to implement in the experiment. Then later in the experiments' main parts, we implemented the Decoder first, then Part 1 using that. We also implemented Part 2 using Part 1's modules. Part 3's ALU is designed independent from the other modules. 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.

5 CONCLUSION

We successfully implemented the aforementioned parts' modules and validated their correctness by simulating them with appropriate samples. Overall, we learned how to design a decoder, a 16-bit register line, and an ALU, as well as what kinds of problems we might encounter while doing so. While certain parts of this report are missing, we will still implement them later for personal learning purposes.

REFERENCES

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

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