ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION REPORT

PROJECT NO : 1

SUBMISSION DATE : 13.03.2019

GROUP NO : 12

GROUP MEMBERS:

150160151 : Esin Ece AYDIN

150170016 : Umit BAŞAK

150180704 : Cihat AKKİRAZ

150180705 : Batuhan Faik DERİNBAY

150180707 : Fatih ALTINPINAR

SPRING 2019

1 INTRODUCTION

In this project, registers and register files are designed for different purposes and implemented using the Logism Logic Simulator Software. In the Part-1, two registers will be designed, and these two have functionalities such that clear, increment, decrement, and load. The distinction between them is being 8-bit register and 16-bit register. In the Part-2, register files which is defined as a structure with a large number of registers will be designed based on the registers designed in Part-1.

GitHub is known as the software development platform, is used to do group work quickly and effectively. Thanks to GitHub, all group members have direct access to designed parts in anytime, anywhere. As a result, performance increases and time is used efficiently.

2 METHODOLOGY

This project consists of 2 main parts which are explained in detail in the following sections Part-1, and Part-2. Part-2 is split into 3 subsections as Part-2a, Part-2b, and Part-2c according to the requirements.

2.1 Part-1

In the first part of the project, 2 different registers (8-bit and 16-bit) with 4 functionalities are designed. These functionalities are controlled by two bit control signals (FunSel) and an enable input (E). Table 1 shows the characteristic table of these two registers.

E	FunSel	Q^+
O	Φ	Q (Retain Value)
1	00	I (Load)
1	01	Q-1 (Decrement)
1	10	Q+1 (Increment)
1	11	0 (Clear)

Table 1: Characteristic table of registers.

To start with, the circuit represented in Figure 1 was thought on and designed. It featured every required aspect of the assignment and worked flawlessly. It's concept was simple and it was driven by a decoder that is controlled by function selection inputs.

Delay flip flops were used as the memory units and inputs were fed through 4:1 multiplexers which were controlled by both the function selection decoder and the enable input.

Arithmetic operations (increment and decrement) were done via a full adder. Function selection decoder controller decided whether to perform incrementation or decrementation.

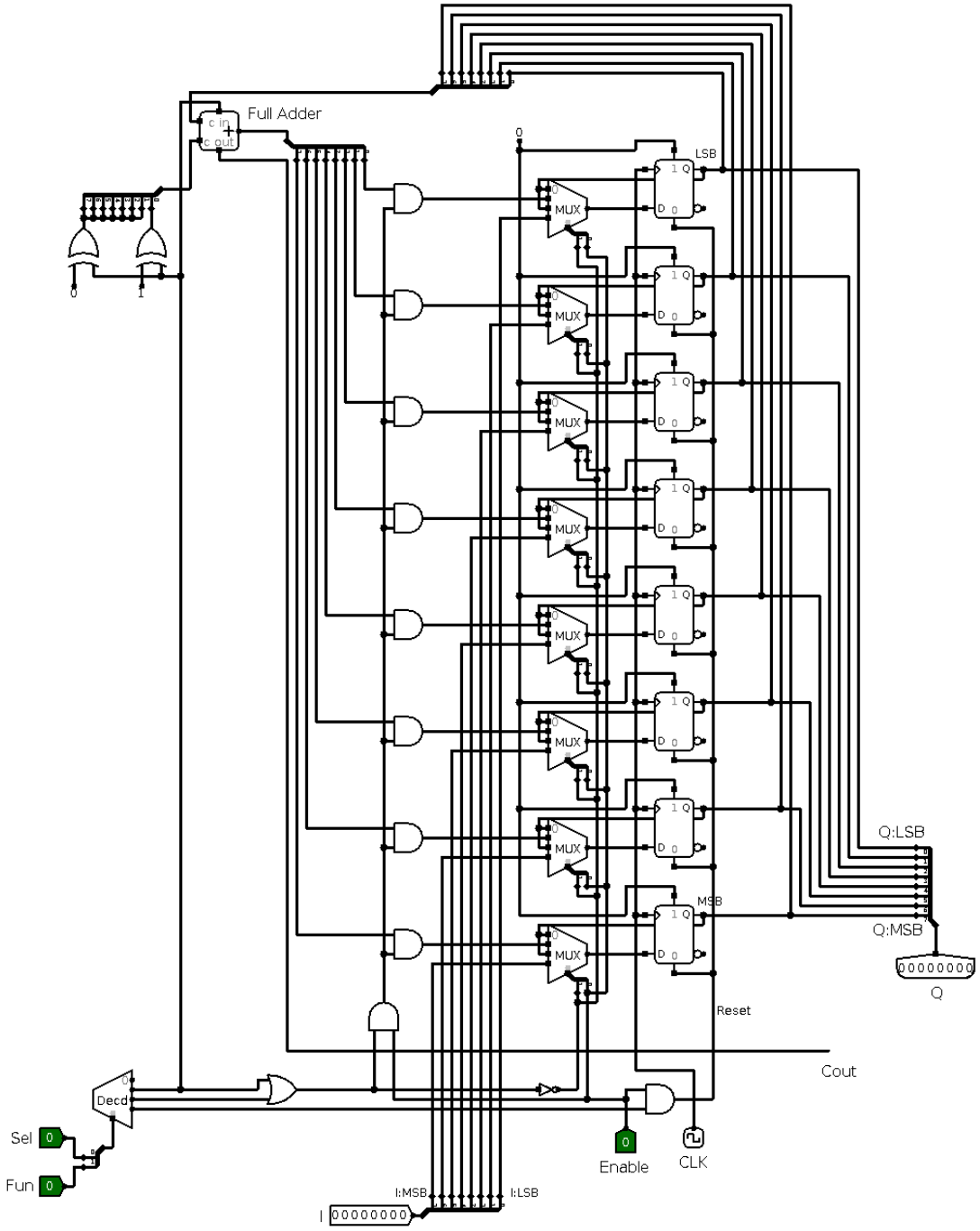


Figure 1: First attempt at designing the 8-bit register.

Once the design was completed, the circuit was tested extensively. After this first design process and necessary tests, the team decided to improve upon the idea and came up with the design seen in Figure 2.

This design not only gets rid of the full adder but also tackles the arithmetic challenge in a more ingenious way. It does so by using some additional AND, OR and XOR gates. Rest of the design is quite similar to the first version.

Logic behind the arithmetic part of the design is not that complicated. Arithmetic operations are performed by AND, OR and XOR gates.

In order to decrement the value stored in register:

1. First bit gets negated no matter what. ($\bar{Q_0}$)
2. Second bit gets negated if the first bit is 0. Checking this condition achieve by using an XOR gate that has inputs as Q_1 and $\bar{Q_0}$. ($\bar{Q_0} \oplus Q_1$)
3. Other bits get negated only if less significant bits are all 0. This check is completed by using OR gates. An OR gates gives 0 output if and only if all of it's inputs are 0. Negated output of this OR gate then given into the XOR gate which does the negation. ($(Q_0 + Q_1 + ... + Q_{n-1}) \oplus Q_n$)

To achieve incrementing the value stored:

1. First bit gets negated no matter what.($\bar{Q_0}$)
2. Second bit gets negated if the first bit is 1. Checking this condition achieve by using an XOR gate that has inputs as Q_1 and Q_0. ($Q_0 \oplus Q_1$)
3. Other bits get negated only if less significant bits are all 1. This check is completed by using OR gates. An AND gates gives 1 output if and only if all of it's inputs are 1. Output is given into the XOR gate which does the negation like in decrementing.($(Q_0 \cdot Q_1 \cdot ... \cdot Q_{n-1}) \oplus Q_n$)

With the correct implementation of arithmetic operations, the design was completed. After applying the concept on the 8-bit register and testing if it works as intended, the group extended it into a 16-bit register (see Figure 3).

In conclusion, an 8-bit and a 16-bit registers are designed and improved in Part-1. These registers are able to retain their value when inactive, load an external input value, decrement or increment the current value or clear the memory according to the values of function selection and enable signals. They can and will also be implemented in feature designs as seen in Figure 4.

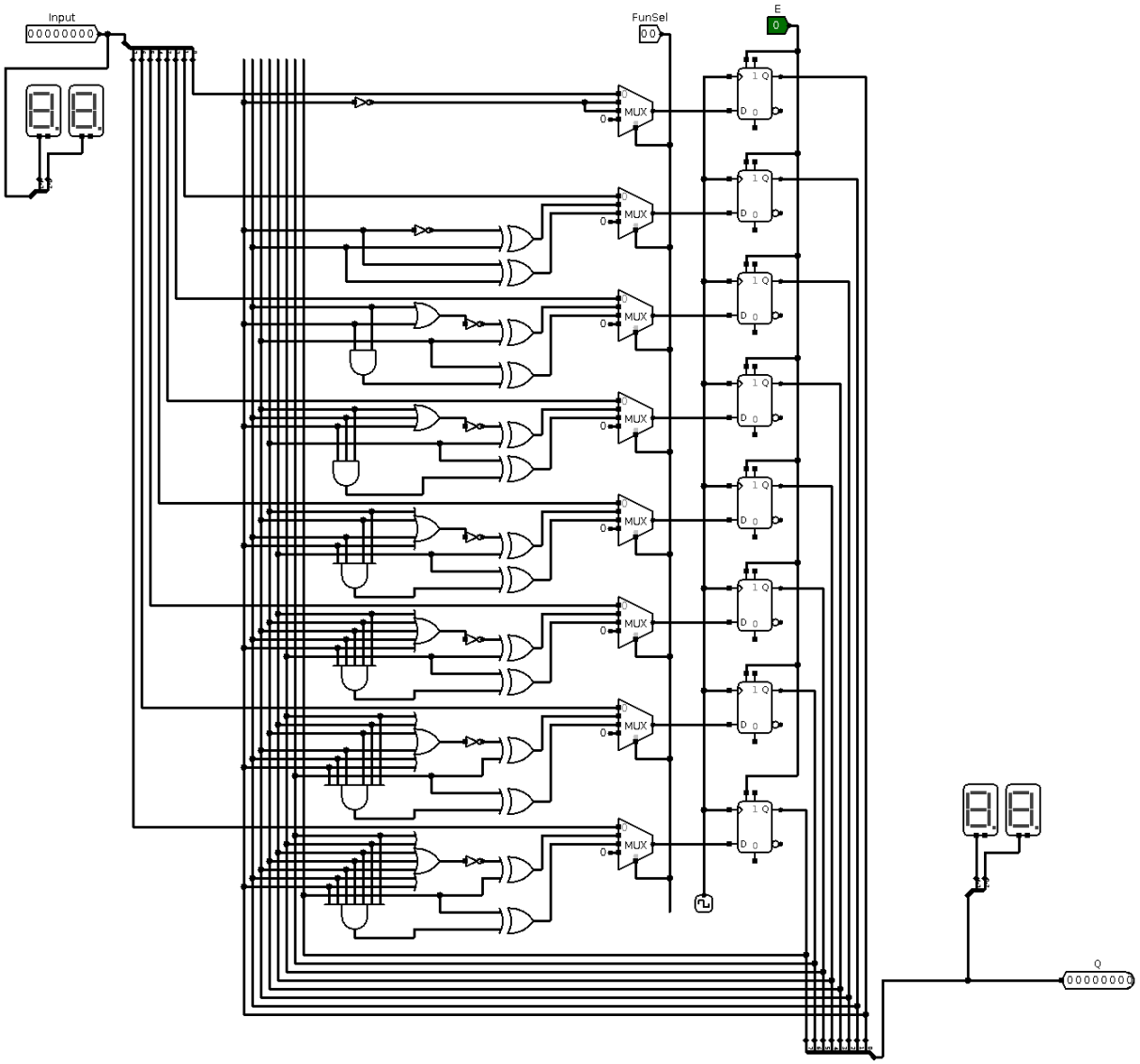


Figure 2: Final design of the 8-bit register.

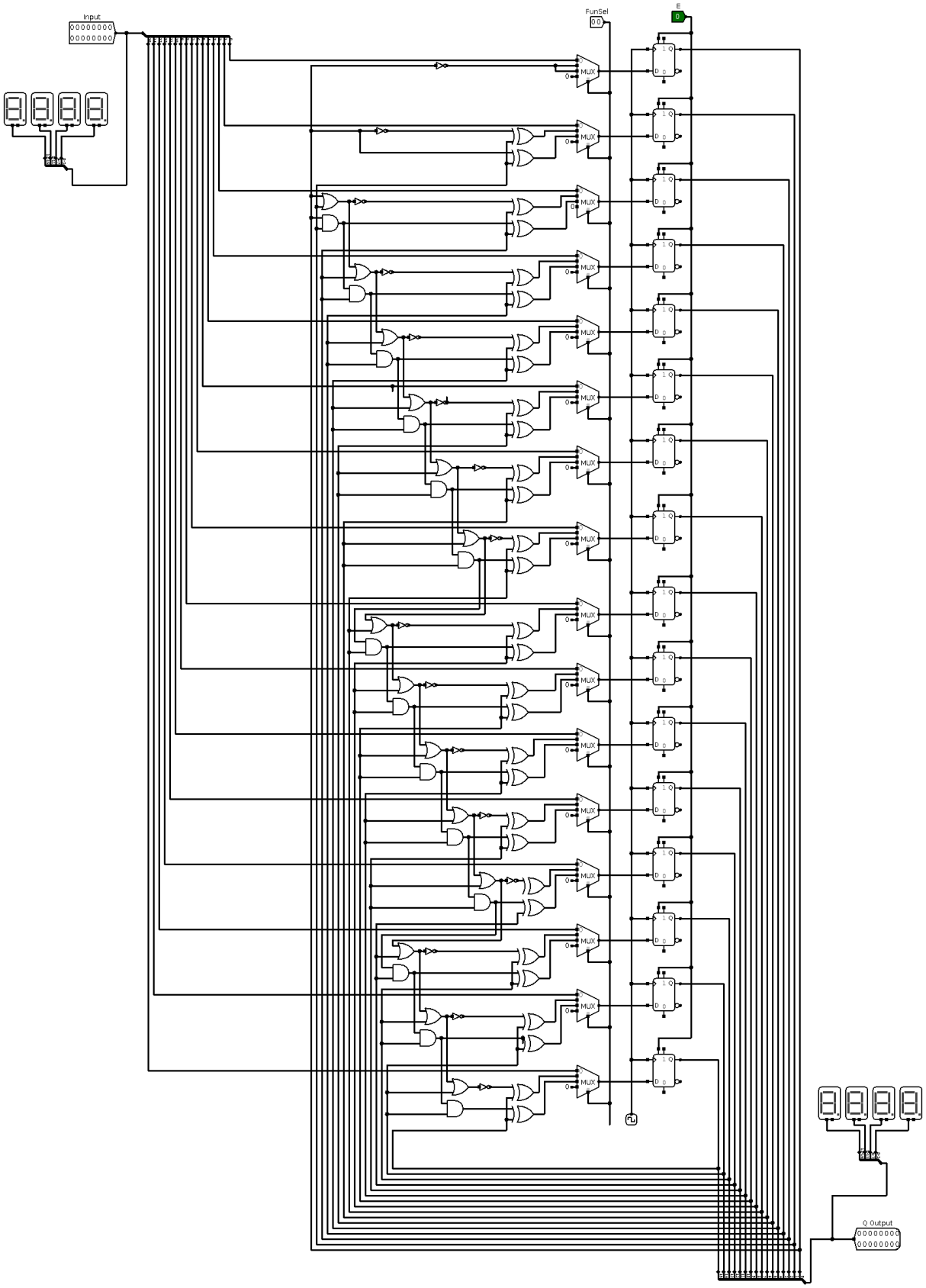
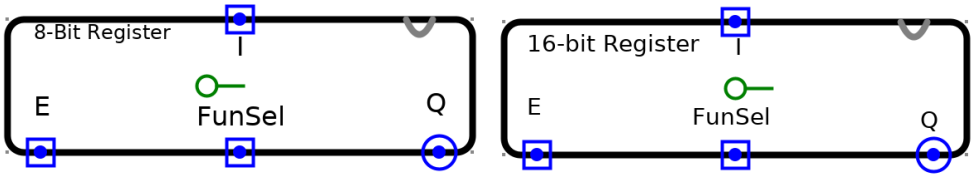


Figure 3: Final design of the 16-bit register.



(a) Subcircuit of 8-bit register (b) Subcircuit of 16-bit register

Figure 4: Subcircuit diagrams of registers in Part-1

2.2 Part-2

In the second part of the experiment, register files are designed for the purpose of selecting registers and implementing desired functions to these registers.

2.2.1 Part-2a

In this part, 4 register(8-bit registers are designed previously) are implemented to the circuit. The aim of this implementation is that selecting registers and applying selected functions to these registers. Also, displaying only desired registers's output values. Registers are selected via 2-bit RegSel signal and functions are selected with the 2-bit FunSel signal and applied to these registers with the next clock cycle. Registers displayed on the hex display are controlled by 2-bit OutASel and OutBSel signals.

OutASel	Output A
00	R3
01	R2
10	R1
11	R0

OutBSel	Output B
00	R3
01	R2
10	R1
11	R0

Table 2: OutASel and OutBSel controls.

On the two hex display two registers's output values can be seen and which registers will be displayed is controlled by OutASel and OutBSel signals as shown table above. To select displaying registers two 4x1 multiplexers are used. One for displaying register selected with OutASel signal and other one for dislaying register selected with OutBSel signal.

When 8-bit input signal entered to system, registers will take this information and only selected registers will be applied selected functions. Other registers are not affected

because of Enable signal. Enable value of selected registers are 1 and others enable value are 0. Which registers will be selected are controlled by 4-bit RegSel signal as shown table below.

RegSel	Enabled Registers
0000	N0 register is enabled
0001	Only R0 is enabled
0010	Only R1 is enabled
0011	R0 and R1 are enabled
0100	Only R2 is enabled
0101	R0 and R2 are enabled
0110	R1 and R2 are enabled
0111	R0, R1 and R2 are enabled
1000	Only R3 is enabled
1001	R0 and R3 are enabled
1010	R1 and R3 are enabled
1011	R0, R1 and R3 are enabled
1100	R2 and R3 are enabled
1101	R0, R2 and R3 are enabled
1110	R1, R2 and R3 are enabled
1111	R0, R1, R2 and R3 are enabled

Table 3: Regsel control inputs.

On the first version of the project, decoder were used for selecting registers. It worked properly for purpose of the project. But using splitter are seem easy and better way for selecting registers. Team decided to use splitter instead of decoder. The final version of the part can be seen in the next figure.

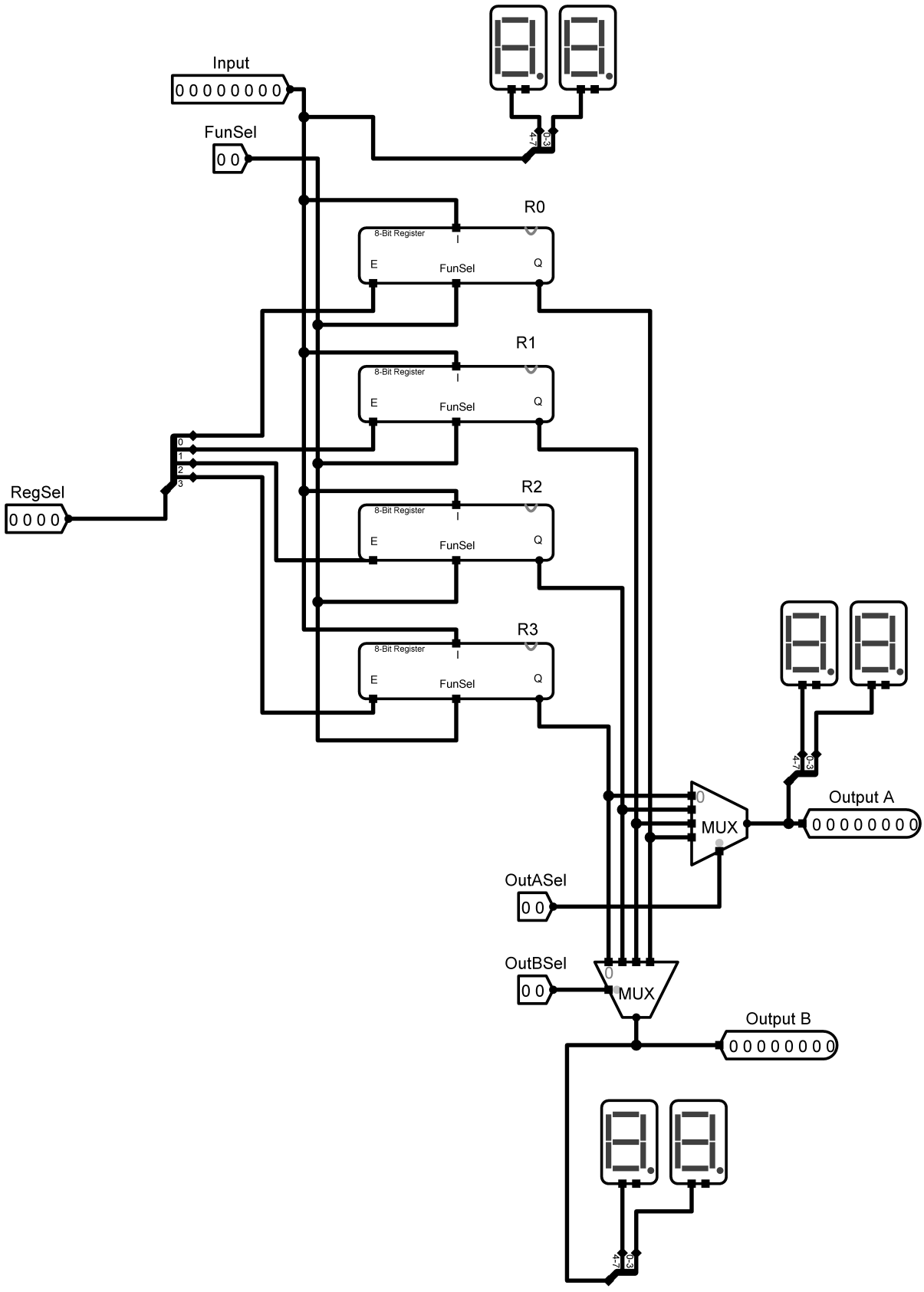


Figure 5: Final design of the Register Selector.

2.2.2 Part-2b

Second part of part 2 is similar to first part of part 2. In this part, three 8-bit registers are implemented to the circuit. When 8-bit input signal is entered to the circuit, registers that are selected with 3-bit RegSel signal are applied to selected functions with 2-bit FunSel signal. Output values on the hex display are controlled by 2-bit OutCSel signal as shown table below.

OutCSel	Output C
00	AR
01	SP
10	PC
11	PC

Table 4: OutCSel controls.

On the hex display registers's output values can be seen and which register will be displayed is controlled by OutCSel signal as shown table above. To select displaying registers 4x1 multiplexer is used.

Registers are selected with the 3-bit RegSel signals. When the least significant bit is 1 R0 will be selected, the second bit is 1 R1 will be selected, the most significant bit is 1 R2 will be selected. Only selected registers will be affected from selected functions with the next clock cycle. Other ones will stay same. The circuit can be seen in the next figure.

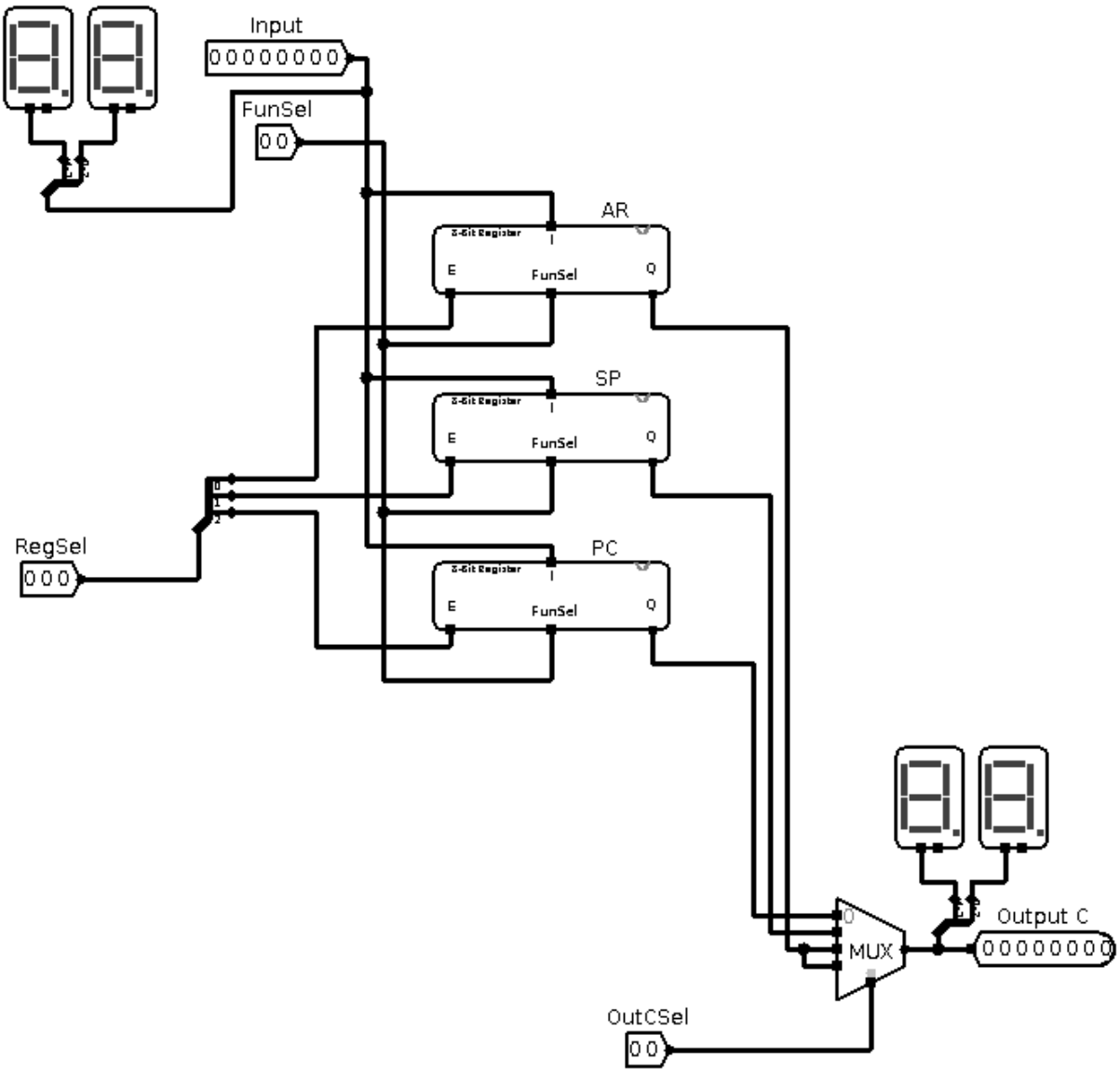


Figure 6: Final design of the Adress Selector.

2.2.3 Part-2c

In the final part of the project, a 16-bit IR register that has a 8-bit input with 5 functionalities is implemented to the circuit. These functionalities are controlled by a two bit control signal(FunSel), an enable input(E) and an input which indicates whether to load lower bits or higher bits of IR(L/H'). Table 4 shows the characteristic table of this register.

L/H'	E	FunSel	IR^+
Φ	0	$\Phi\Phi$	IR
1	1	00	IR(0-7) - I(Load)
0	1	00	IR(8-15) - I(Load)
ϕ	1	01	IR - 1
ϕ	1	10	IR + 1
ϕ	1	11	0

Table 5: Characteristic table of the IR register.

To start with, the given task of loading 8 data bits into required positions in the 16-bit register is analyzed. Relations between the 16-bit register seen in Figure 3 and the task is apparent. The only difference is that the input has 8-bits of data rather than 16-bits. In order to achieve the task, input 8-bits of data are channeled in to the corresponding output bits.

For that, writing 8-bit input data to the desired positions in the 16-bit register, 2:1 Multiplexers are used. If the $L/\bar{H}$ input is 0, bits from 8-15 takes new values from input and bits from 0-7 preserve their values. In order to write input values to the bits 0-7, $L/\bar{H}$ input should be 1.

The methodology of deciding input per D flip-flop can be explained as follows:

1. Depending on $L/\bar{H}$ signal a 2:1 multiplexer which decides between writing new state or preserving current state.

2. Output of this 2:1 multiplexer is connected to the first input of the corresponding 4:1 multiplexer which decides what will be next state of the D flip-flop.

Since only the first input of the 4:1 multiplexer changed, other functions remain same as they are in 16-bit register.

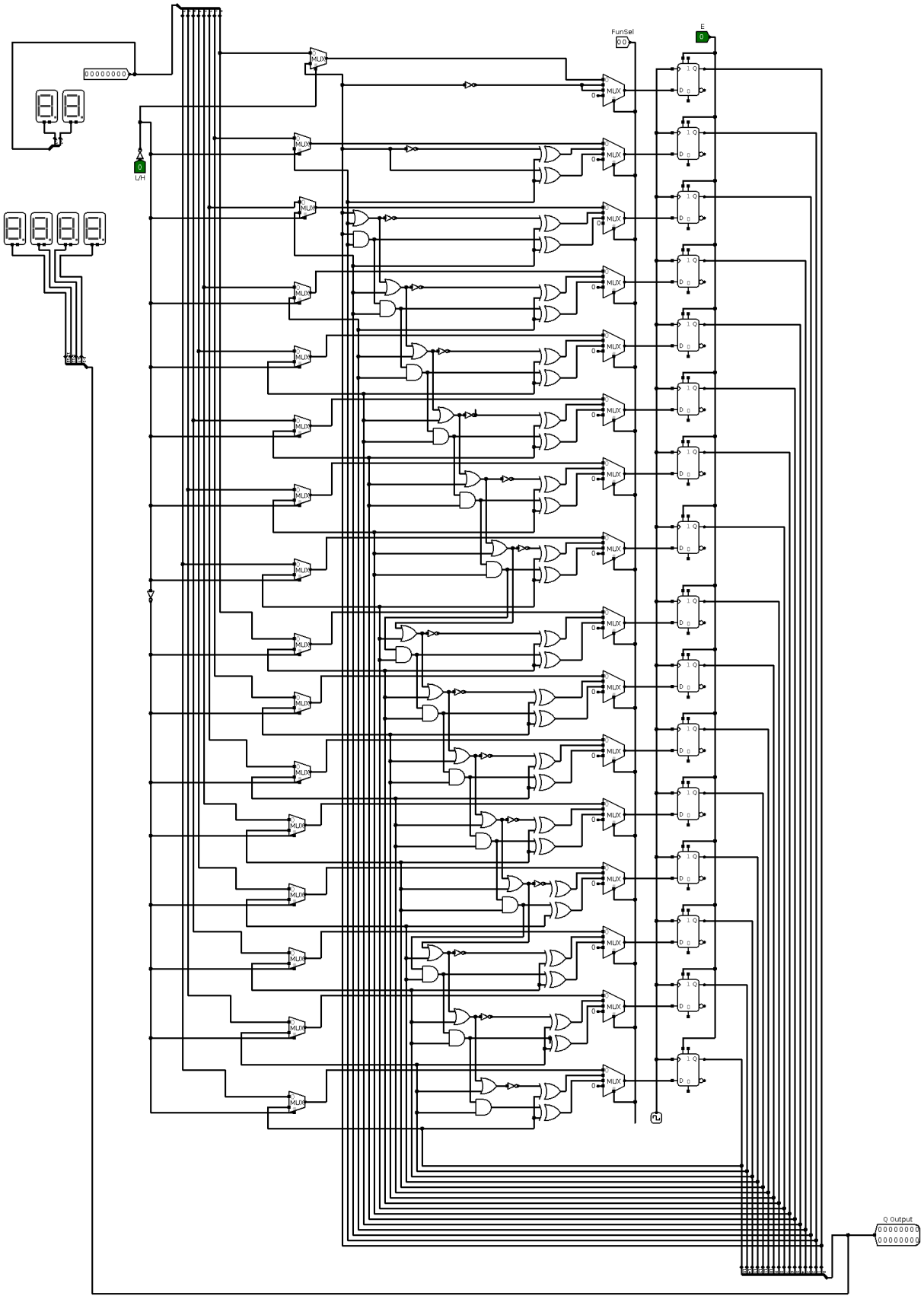


Figure 7: Design of the 16-bit IR register.

3 DISCUSSION

This project is useful for learning design process of registers and register files. It also has a pragmatic role; because, it is required to practice in the Logism software in order to possess the necessary abilities to finish the task. As a result of that, gained ability to use this software easily. For instance, we learnt how to design the circuit in simpler way by using splitters.

Another subject we have learned is, we can store the data thanks to registers. Moreover, when some operations are needed to be done, they can be done through appropriate registers. These functionalities can be used for many purposes.

4 CONCLUSION

In conclusion, different types of registers are designed considering their functionalities and control signals. When the design is done, other available combinational or sequential Logism units are used. For instance, in the first part of the project, 4:1 MUX, D Flip Flop, NOT, AND, OR, and XOR gates are added in a way which would have the least cost. In order to design 16-bit register in a clever way, the 8-bit register is extended into a 16-bit register. As a result, the connections are prevented from becoming more complicated and register is designed with a small number of connections. It means, it requires less wiring when this 16-bit register is implemented in real world. These registers are designed as a library unit; thus they can be reused in the second part.

In the second part, we used these registers to design register files. For example, in the Part-2a, four 8-bit Register are used. Also, three 8-bit registers are used as seen in the Part-2b. Like in the previous part, some combinational or sequential Logism units such as 2:1 MUX, 4:1 MUX, D flip-flop, NOT, AND, OR, XOR gates are added accordingly considering these register files' purposes.

To control the inputs of the designed registers and register files and also to test their functionalities, pins which can be found under Wiring group of Logism software, are connected. Similarly, for observing the outputs Hex Digit Displays are connected to inputs and outputs of the circuits.

This project report is written using the online latex site (www.overleaf.com), which allows group members to work on the document together allowing time to be used efficiently.

REFERENCES

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

[2] Detailed info on writing reports https://projects.ncsu.edu/labwrite/res/res-studntintro-labparts.html.

[3] Website lets collabration over projects. https://github.com/.