Faculty of Engineering

International Credit Hours Engineering Programs

Communication Systems Engineering Program

Academic Year 2023/2022 – Summer 2023

CSE 111

Logic Design

Project Report

Name	ID	Program
Youssef Hany Youssef Elboukhary Saad	21P0398	COMM
Sama Ashraf Mabrouk Abdelwahab	21P0066	COMM
Amir Raed Mories Shouky	21P0380	COMM

Table of Contents

Phase 1
Designing a 4-bit ALU

1. Introduction

The objective of this project is to create a 4-by-4-bit ALU (Arithmetic Logic Unit) that can execute a variety of arithmetic and logical operations between two binary values, much like the early computers created in the early days of computer technology. The ALU will function as a small, pre-elementary computer that can conduct calculations and display the results.

2. Designing steps

To achieve this, the project will involve different levels of abstraction and design. The following stages will be followed:

- **Requirement Analysis**: Specify the arithmetic and logical operations the 4 by 4-bit ALU should be able to do. The inputs and outputs needed for each operation should be determined.
- **Logic Design**: Use multiplexers and Boolean logic gates (AND, OR, NOT, etc.) to create a high-level logic design for the ALU. Make a list of the control signals required to choose the desired operation.
- **Circuit Layout**: Create a circuit schematic by converting the high-level logic design. List the components or integrated circuits (ICs) needed to assemble the ALU. Think about things like component cost, performance, and availability.
- **Component Selection and Procurement**: Using the circuit design and specifications, choose the proper integrated circuits (ICs), resistors, capacitors, and other components. Purchase the necessary components from reputable vendors.
- **Circuit Assembly**: Put the circuit parts together on a breadboard. Make sure the components are stated and connected correctly. At each level of the assembling process, thoroughly test and verify.
- **Testing and debugging**: Check that the correct arithmetic and logical operations are carried out when you input binary numbers to the ALU to

test its functionality. Fix any problems or mistakes that appeared during the testing procedure.

- **Documentation**: Prepare comprehensive documentation, like this one, that includes the circuit schematic, component list, assembly instructions, and testing procedures. This documentation will serve as a reference for future use by any learner and troubleshooting.

3. Design

3.1. Inputs

The first step in any design process is to determine the inputs of the system to be used in further operations. In our case, we have 4 main inputs – numbers away from selections – which represent two binary numbers A and B.

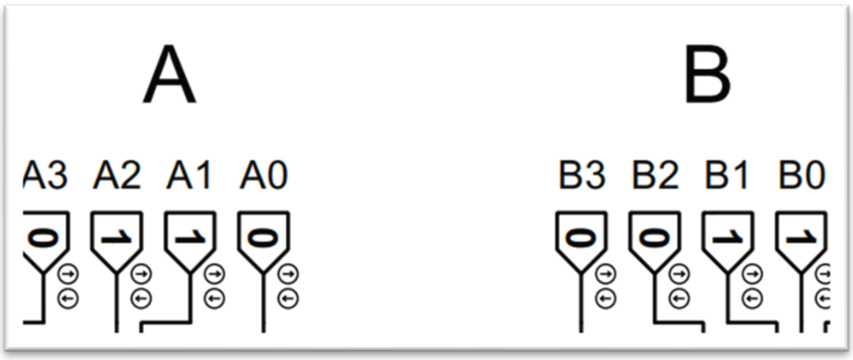


Figure (1): Inputs design in proteus

For getting the inputs from the user, we use Dip switches connected with a pull-down circuit to avoid any noise or interference. The user will need to move these dip switches to get the calculations needed in the further steps.
This input can be replaced by connecting the inputs to the VCC and GND to represent 1 and 0, respectively. This method will cause noise and interference in the circuit.

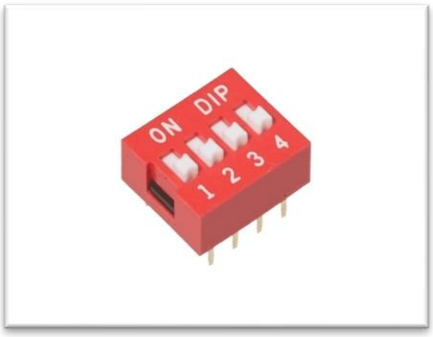


Figure (2): Dip switch used in the circuit for inputs.

3.2 Selectors

Other types of inputs other than numbers A and B are used to control the output of the circuit. For his part we have 5 selectors – S_0 to S_4 – to control our circuit. These selectors can be divided into 2 main groups according to the MUXs where we used them. The First two selectors S_0 and S_1 are controlling the input from the B inputs.

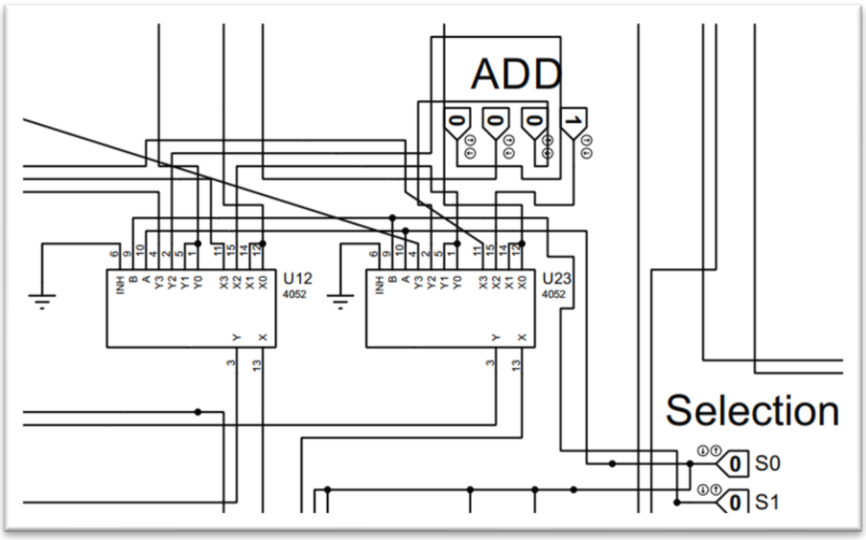


Figure (3): First 2 selectors and 4-1 MUXs connection on Proteus

As shown in **Figure (3)**, the 2 selection bits are used to control the 2 double 4 to 1 MUX. This mux can have 4 cases according to the input of the selector. We can have a group of cases for the B according to our goal for each case; the use of this selection will be clarified in the coming sections.
The following table represents the cases for B.

S_0	S_1	Mux's output
0	0	Input B
0	1	Input B
1	0	0001 (decimal 1)
1	1	Input A

For the other 3 selectors, they are used in determining the operation to be represented to the user. These 3 selectors are passed to a 8-1 MUX that has the output of the 4 operations. The MUX's connection with the selectors is shown in **Figure (4)**. It's important to note that the connection used to form the 8 operations is to be formed using specific codes of the 5 selectors. This part will be clarified step by step in this report.

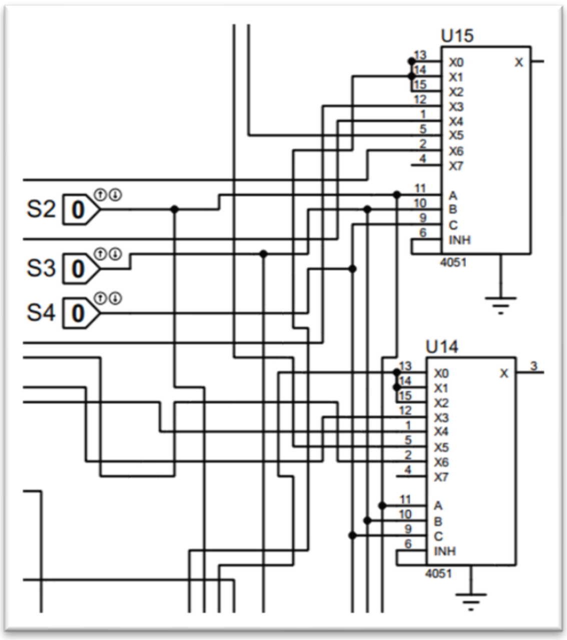


Figure (4): First 3 selectors and 8-1 MUXs connection on Proteus

S_2	S_3	S_4	Mux's output
0	0	0	Adder/subtractor output
0	0	1	Adder/subtractor output
0	1	0	Adder/subtractor output
0	1	1	A complement
1	0	0	A AND B
1	0	1	OR circuit
1	1	0	Multiplication circuit
1	1	1	Division circuit

3.3. Adder/Subtractor operation

For the adder subtractor, we have first to choose the design of the Full adder, we may use only NAND or AND, XOR and OR. After checking the usability of each design, we chose the 3 gates design. To use it me used the K-map for addition.

AB / C_{in}	00	01	11	10
0		1		1
1	1		1	

Figure (5): Sum K-map for full adder design

$$S = A \oplus B \oplus C_{in}$$

AB / C_{in}	00	01	11	10
0			1	
1		1	1	1

Figure (6): Carry out K-map for full adder design.

$$C_{out} = A\,C_{in} + B(\,C_{in} \oplus A)$$

Inputs			Outputs	
A	B	C_{in}	S(Sum)	C_{out} (Carry)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The design of each full adder was used based on this concept and design. The 4 bits by 4-bit adder was then constricted using 4 full adders of the same shape. The design of the 4 by 4 bits adder using 4 Full adders is shown in **Figure (7).**

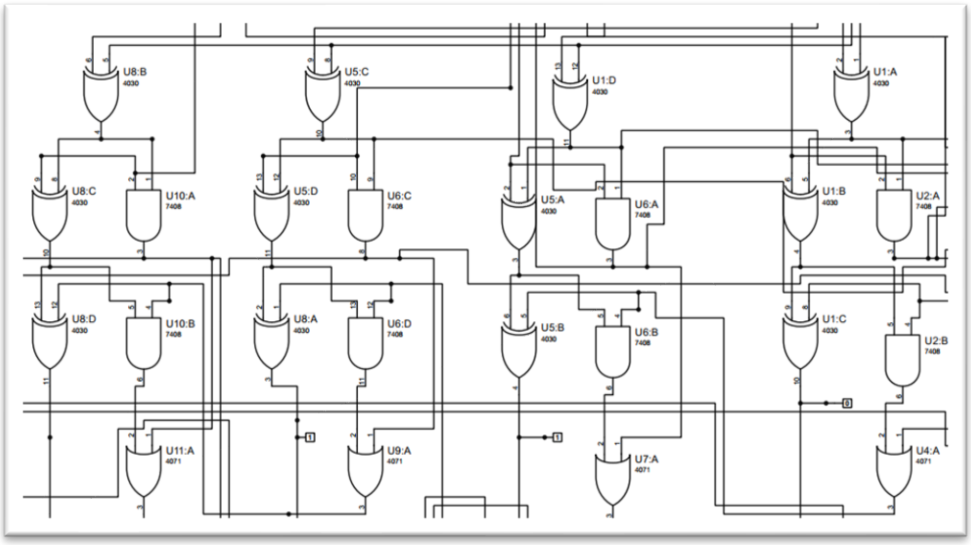


Figure (7): 4 bits adder using proteus.

It's important to note that the control bit in this design is the S_0 bit so that it works as an adder in case S_0 equals zero and as subtractor when it equals one. The main problem with negative numbers is that it will be represented as the two's complement of the number. For this case we needed to take it back to its main form to be represented on the 7-segment display. For this cause, we First used an XOR that takes the last carry out and the control bit. This XOR is supposed to represent the positive number with the extra bit because it's a part of the representation and hide the negative because it only represents sign.

Last carry	Control Bit	Output	Cause
0	0	0	Adding two numbers with no carry means that the number could be represented in 4 bits
0	1	1	Subtracting Two numbers with no carry out makes this a sign bit
1	0	1	Adding two numbers with carry means that the number should be represented in 5 bits
1	1	0	Subtracting two numbers with a carry means that the number is positive, so sign is zero

After controlling the sign bit, it will be used in another adding operation.
First the 4 bits of the answer will be XORed with the sign bit. This will yield the same number in case of positive number and it's 1's complement in case of a negative number.
The 4 bits answer will be then added to the 4 zeroes using a 4 bits adder, and the carry out bit will be the sign bit.
In case the result is negative, the 1's complement generated by the XOR will be added to 1 to generate the main number itself.

In case of a positive number, it will be just added to zeroes, so we have no effect. The circuit responsible for this is shown in **Figure (8).**

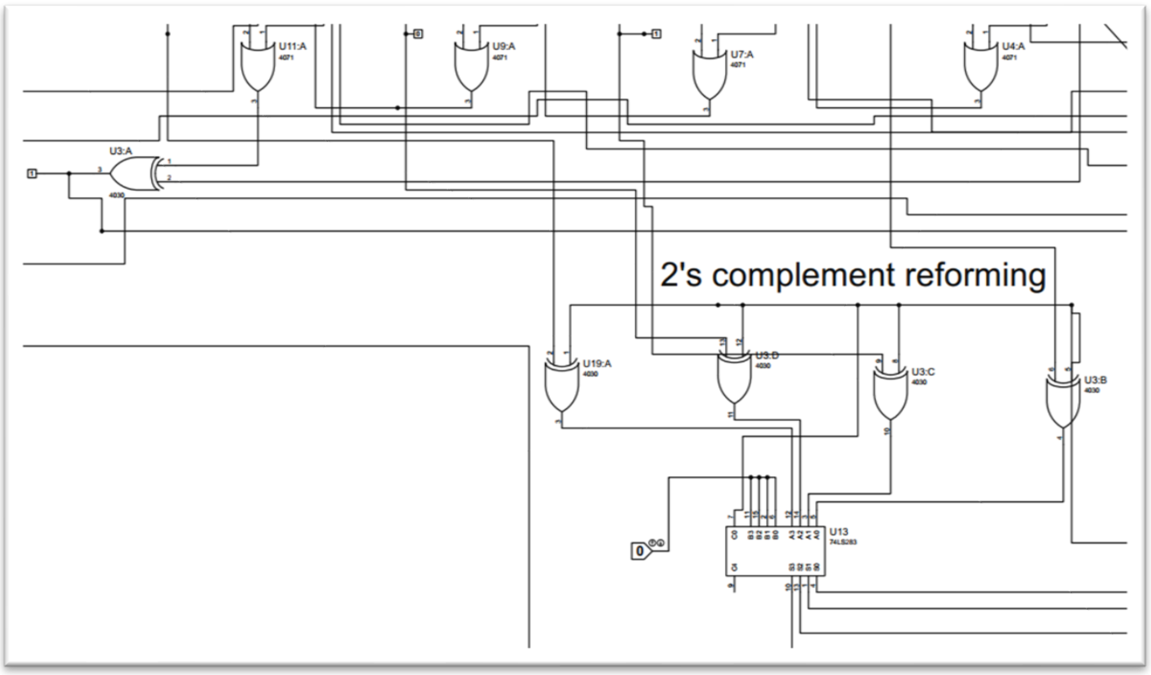


Figure (8): Negative number reforming circuit.

This final answer will be passed to the 8-1 MUX for the output. For the sign, we needed to distinguish between the extra bit in large addition results and the negative sign. To do this, we used an AND gate that has the XOR result and the control bit as inputs.

XORed bit	Control Bit	Output	Cause
0	0	0	Adding means a positive result
0	1	0	Subtracting two numbers with no sign bit means a positive number
1	0	0	Adding means a positive result
1	1	1	Subtracting two numbers with a sign bit means that the number is negative

After this operation, the sign is represented in a single 7 segments where only the "g" segment is connected to the AND output. This means that the negative sign will be shown only with negative numbers. The output circuit is shown **Figure (9).**

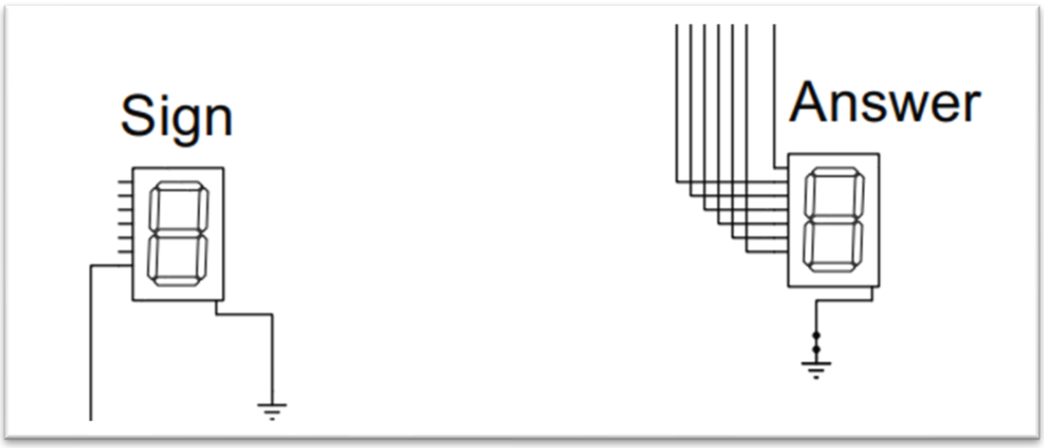


Figure (9): Output circuit for both sign and digit.

3.4 Increment operation

For the increment, we will use the adder subtractor circuit. The only difference is that input B will not be used but instead the binary number "0001" will be used. This will be applied using the 4-1 MUX by adapting the selection bits as shown in **Figure (3)**.

3.5 AND operation

For the AND operation, we will also use the Adder subtractor circuit. In this case, the Adder will just work for addition mode, but instead of taking the output from the last part of the circuit, we will take the output from the First level AND in the Full adder design.
<u>As you may observe, in case of addition, the adder inputs will be just A and B. For each digit, the first AND in this case is directly A AND B for each digit in the 4 bits.</u>

To achieve this, a wire will be directly taken from this AND to 8-1 MUX as a specific pin, so that we can present the result of just the AND not the whole ADD process. The circuit for this operation is shown in **Figure (10)**.

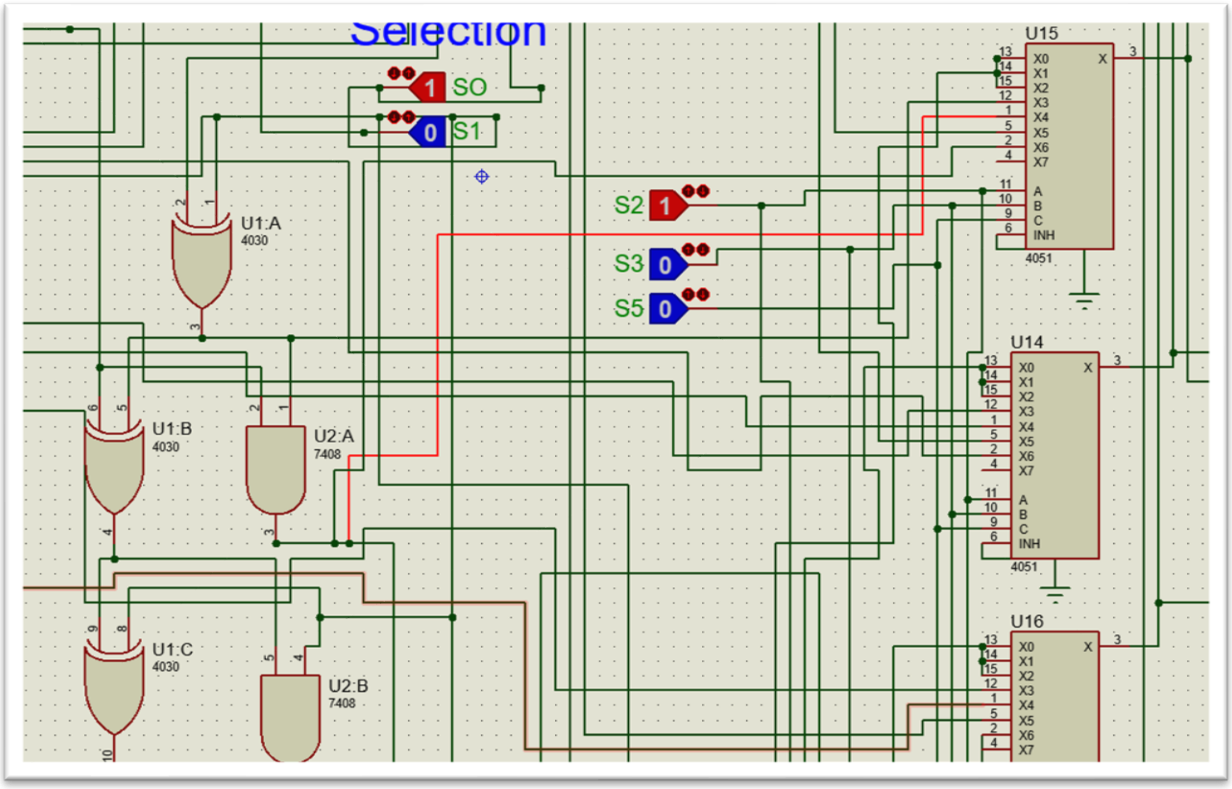


Figure (10): AND output directly connected to the MUX to use the Adder circuit for the AND representation.

3.6 NOT operation

For this operation, we will again use the Adder-Subtractor circuit. In this time the 4-1 Mux will pass the A as input. As it may seem illogical to add A to A or subtract A from A, but in this case, we will use it only for a specific part not the whole circuit – as we did in the AND operation.
<u>The control pin must be 1 in this case to work as Subtraction. In this case the first level of XOR will work as inverter for A</u>.
The inverted A will be taken as output for a specific Pin in the 8-1 MUX to be represented in the output.
The circuit for this operation is shown in **Figure (11)**.

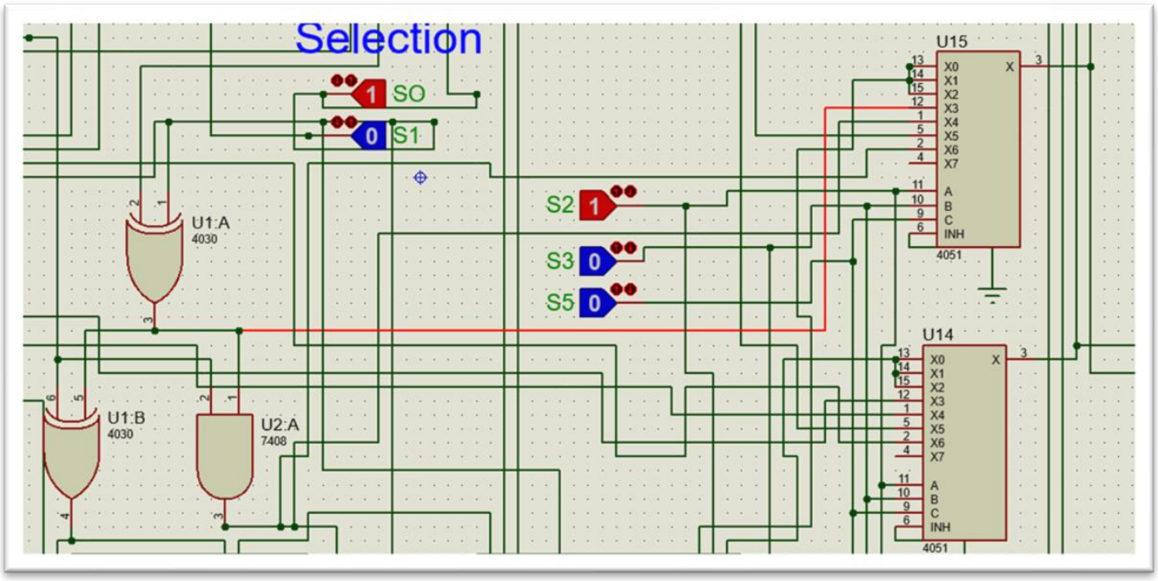


Figure (11): XOR output directly connected to the MUX to use the Adder circuit for the Complement representation.

3.7 OR operation

The OR operation will simply be constructed using OR gates. Simply each input will be connected to an OR gate and its output will be passed to the MUX for its specific pin. The circuit for this operation is shown in **Figure (12)**.

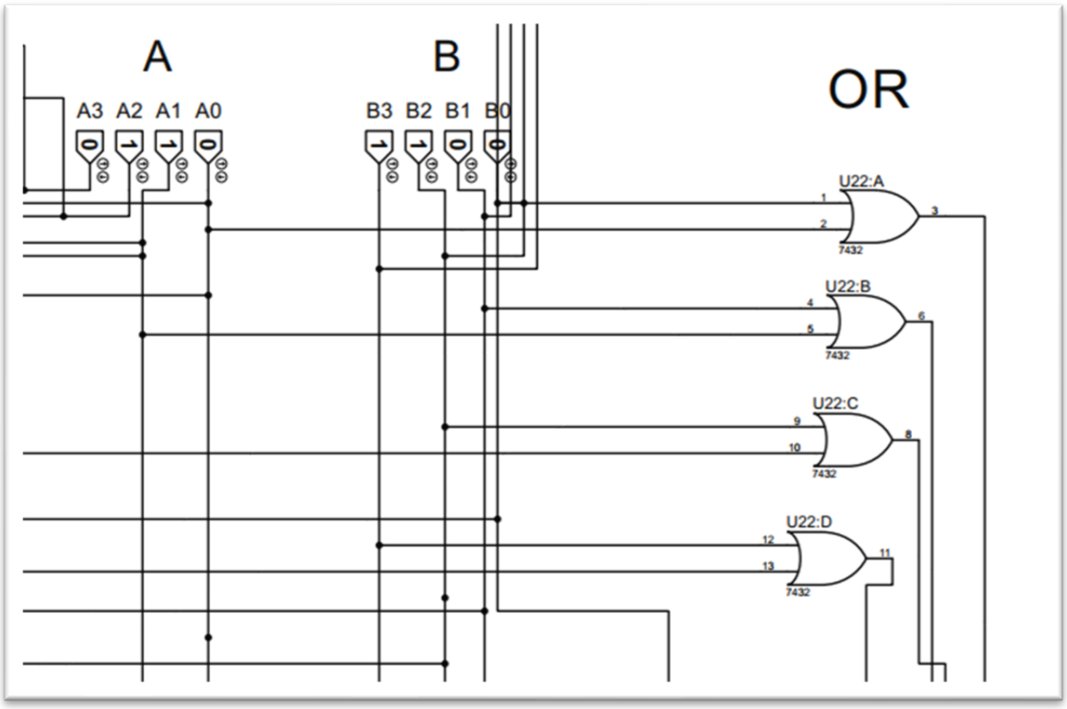
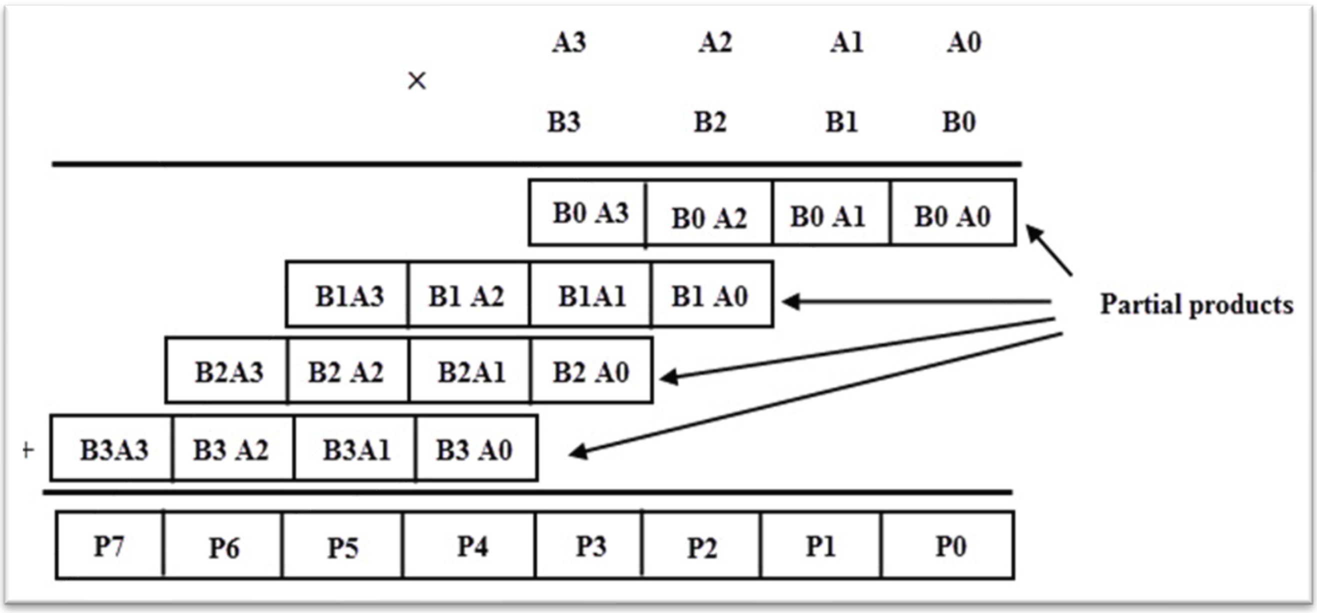


Figure (12): OR operation for the input directly

3.8 Multiplication operation

The multiplication operation will be designed using a behavioral model. In this case we observe how the process is being made by a human and mimic this design. To present this process, we used the model shown in **Figure (13).**



A3 A2 A1 A0
×
B3 B2 B1 B0
B0 A3 | B0 A2 | B0 A1 | B0 A0
B1A3 | B1 A2 | B1A1 | B1 A0
B2A3 | B2 A2 | B2A1 | B2 A0
+ B3A3 | B3 A2 | B3A1 | B3 A0
P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0
Partial products

Figure (13): Multiplication operation model

As we may observe, the process is just a mix of using AND gates and 4 bits adders. For each level we need 4 AND gates and an adder.
To minimize, we may get the direct AND terms [A_0 B_0, A_1 B_1, A_2 B_2, A_2 B_2] from the adder subtractor, and by this we eliminate the use of a full AND IC.
It's important to note that the output of this circuit will be represented in 8 bits. Since all other outputs have maximum 5 bits output 5 of the 8-1 MUXs will have Multiplication as input and the other 3 bits (most significant) will be represented directly on lamps.
The circuit for this operation is shown in **Figure (14)**.

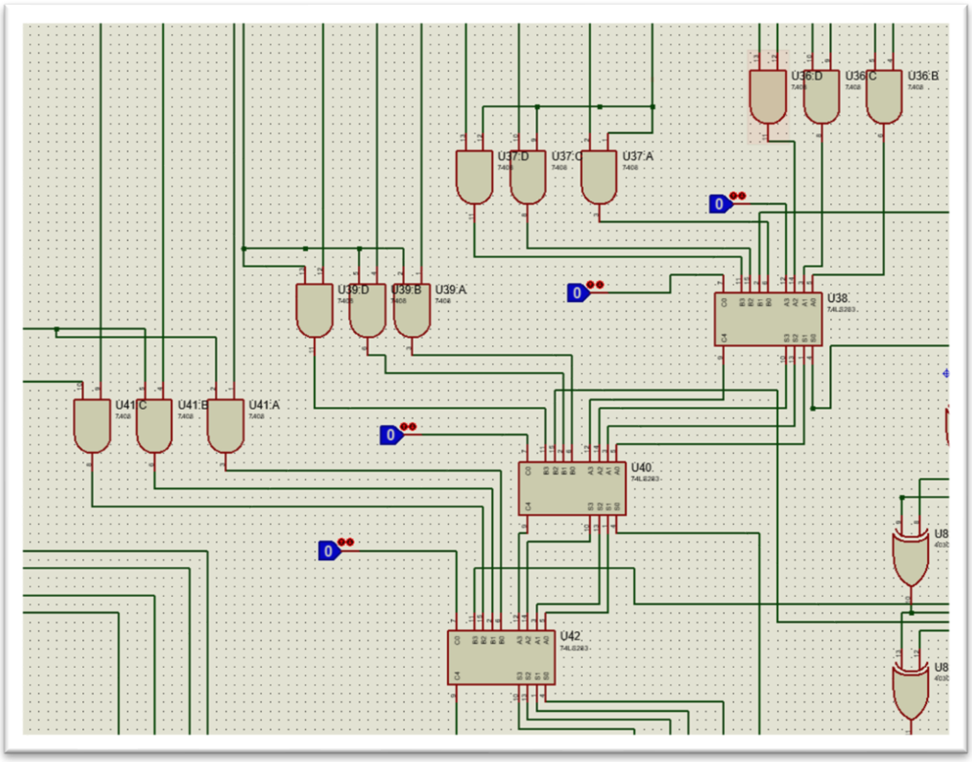


Figure (14): Multiplication operation simulation on proteus

3.9 Division operation

The main idea of the division is actually the subtraction, first we try to subtract the MSB with the number of division and that is the first level in the process, if this level of subtraction has an positive output we take the MSB of result as 1 and then add the digit after the MSB from the input to the remaining of subtraction(not addition just shifting), and if the output is negative we take the MSB of result as 0 but what is important is to restore the MSB of input before the subtraction so we use mux 2in1 and has an input of the result of subtraction and our input and we use the carry from output of subtraction as selector when it is 0 it passes our input and when it is 1 it pass the output of subtraction.
As example 1101 divided by 1. MSB is 1 so the first level of subtraction is 1-1 which is 0 and it is correct subtraction so we take the MSB of result as 1 and the mux will pass 0 and after the shifting it will become 010.
Another example 1010 divided by 11 , the first level of subtraction will be 01-11 which result in a negative number so the MSB of result will be 0 and the mux will pass 1 and after shifting it will be 10.

After the shifting we do the same process again until we find output of n level of subtraction is negative and there is no number to be shifted, here the mux will pass the value before subtracted it and it will be the remaining of the division.

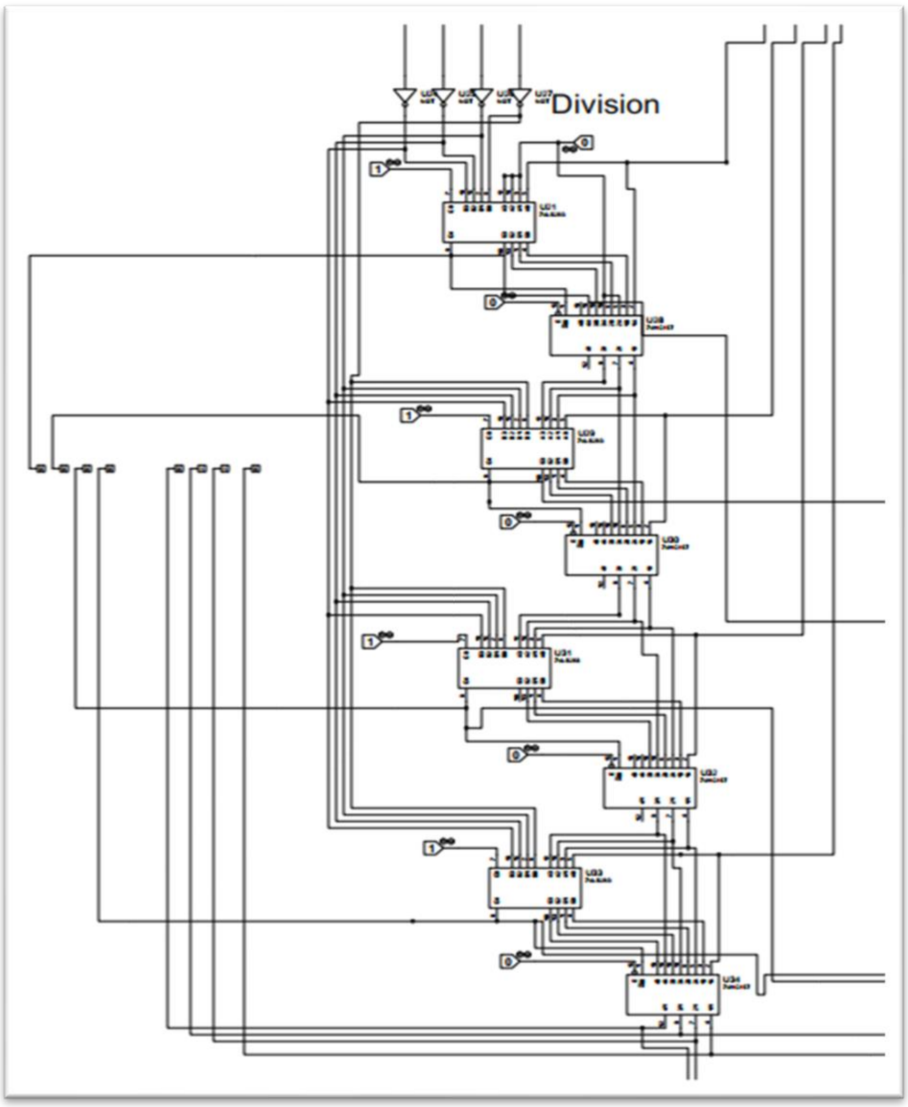


Figure (14): Division operation circuit in Proteus

3.10. Operations codes

As noted, we have 5 selection bits that control the circuit's output. To sum this, we have built a table representing the bit number for each form of output (As shown in the following table).

S_0	S_1	S_4	S_3	S_2	Output
0	0	0	0	0	A + B
1	0	1	0	0	A - B
0	1	1	0	0	Increment A
1	1	1	1	0	A complement
0	0	0	0	1	A AND B
X	X	1	0	1	A OR B
0	0	0	1	1	A multiply B
1	0	1	1	1	A divided by B

3.11 Final Design

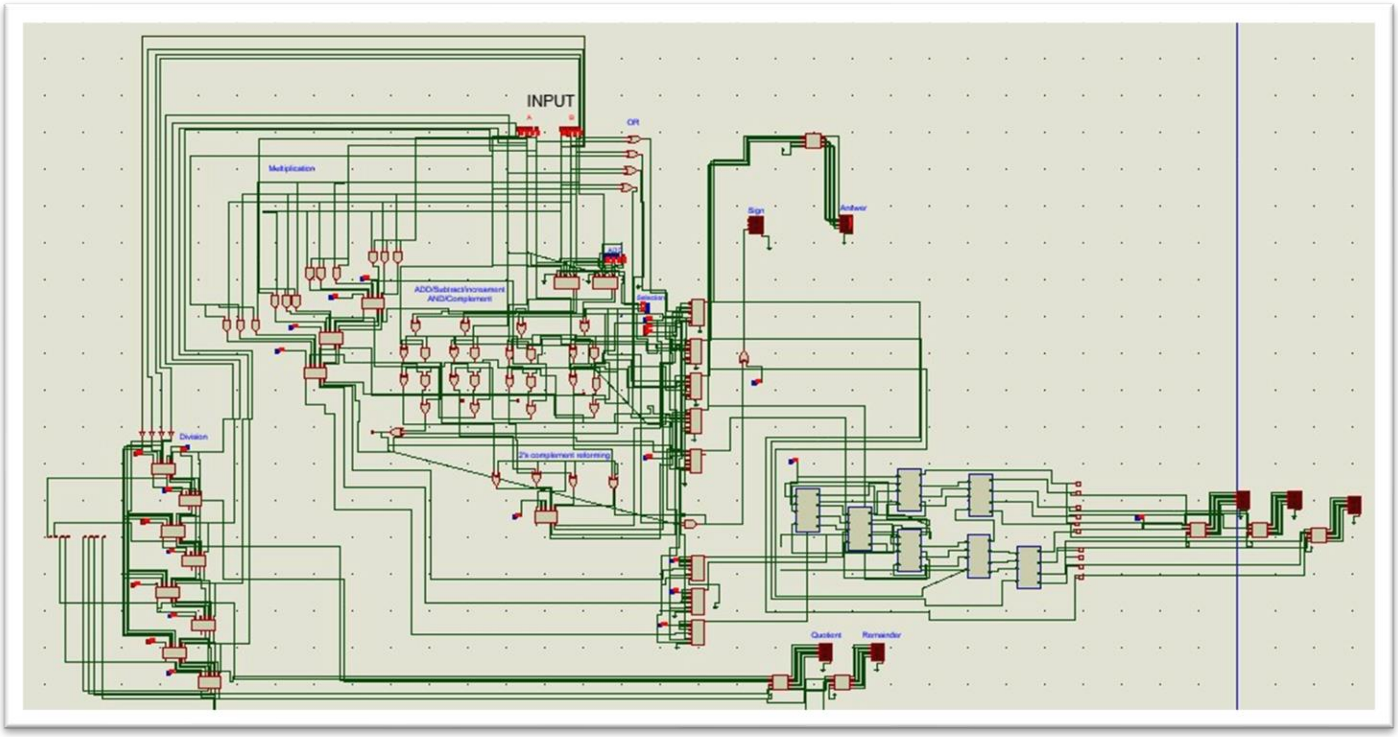


Figure (15): Final design for phase 1

4. Hardware design and implementation

It’s important to note that there are some small details that were not implemented in the hardware for technical reasons and for financial causes. The following Figure represents the Implemented circuit for this Phase.

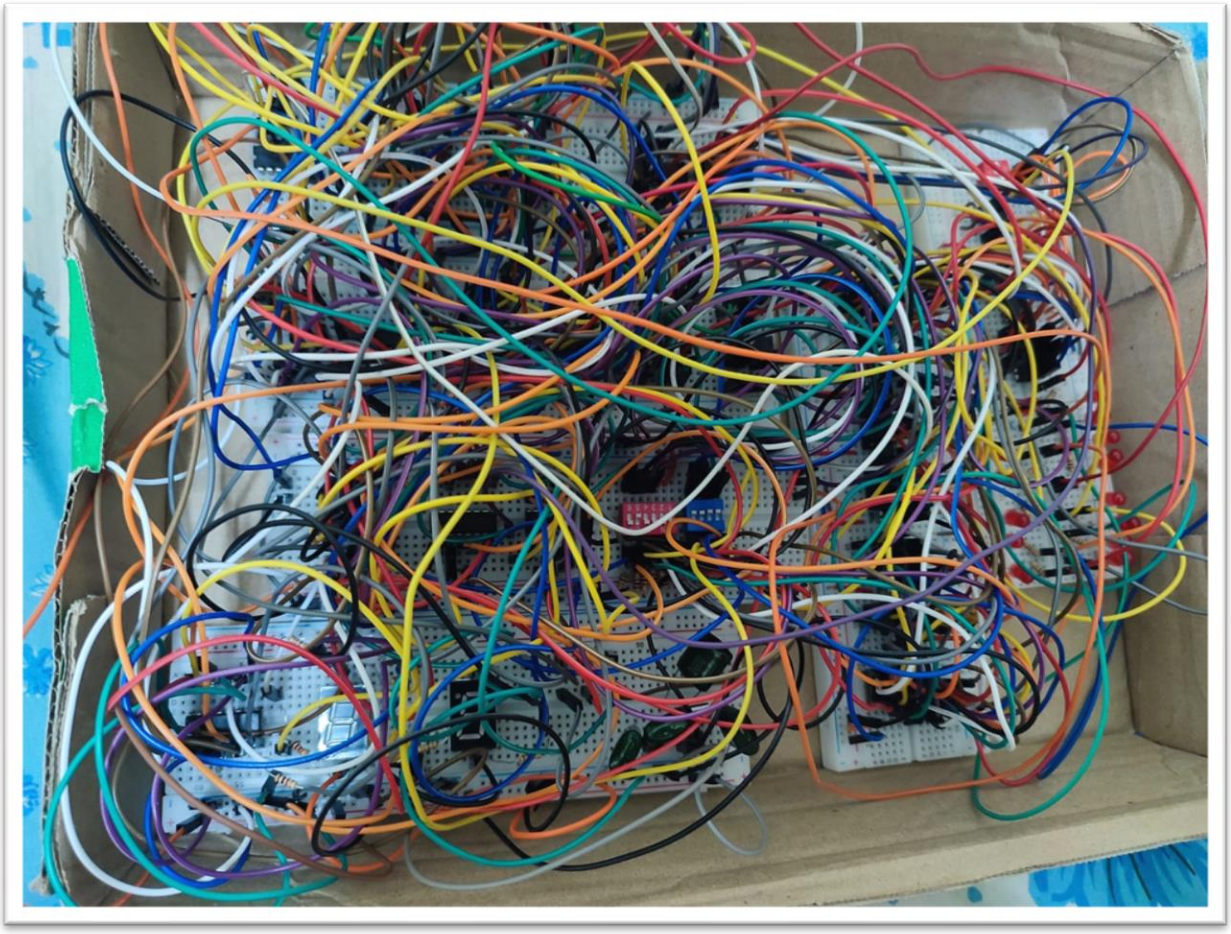
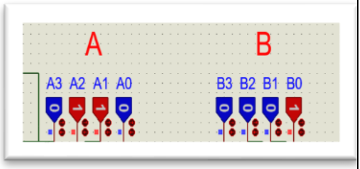
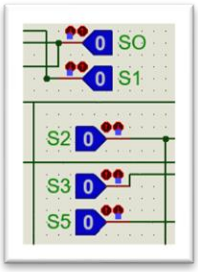
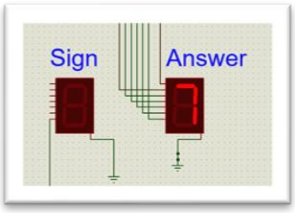
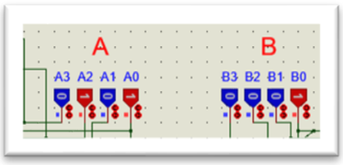
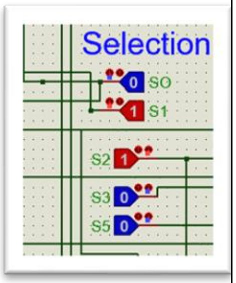
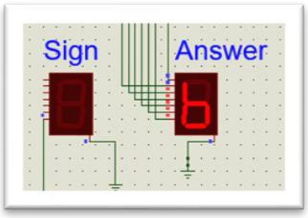
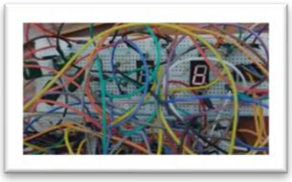
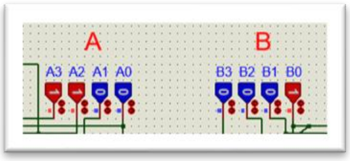
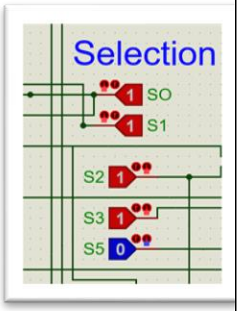
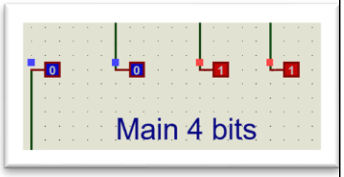
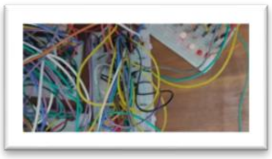
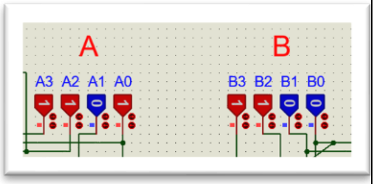
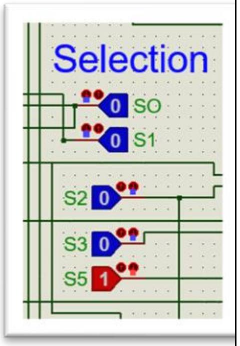
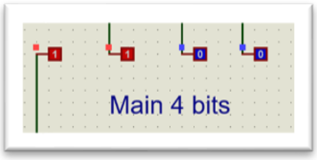
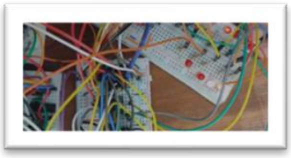
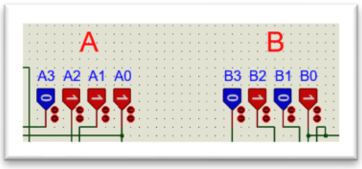
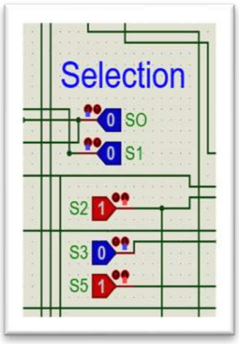
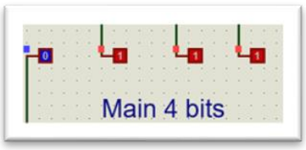
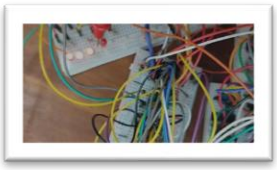
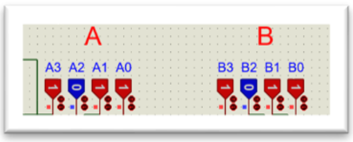
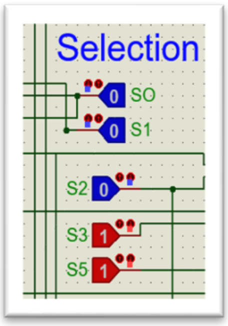
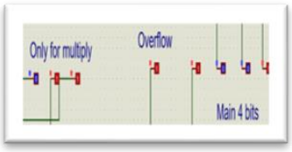
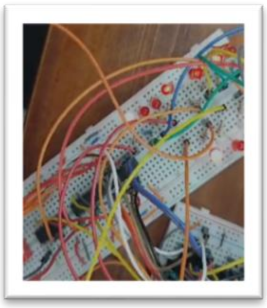
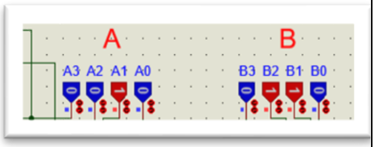
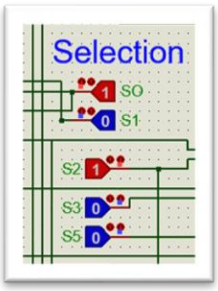
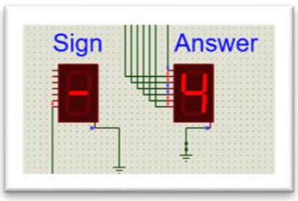
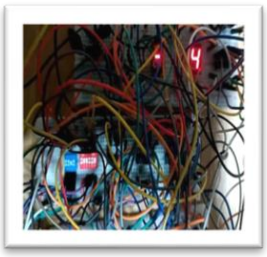


Figure (16): Full implemented circuit for Phase 1

5. Test cases and results

Input	Selection	Simulation output	Real output
			
			
			
			

6. Strengths

Our design has a group of strengths, we will present a group of them in the following points:

- **Portable**: Using a voltage regulator circuit, we can use the circuit in any possible place if we have a battery without the need of electricity resource.

- **Minimum gates**: As we mentioned thought the design process, we have already minimized the circuit massively. We have made 5 operations using only Adder circuit!

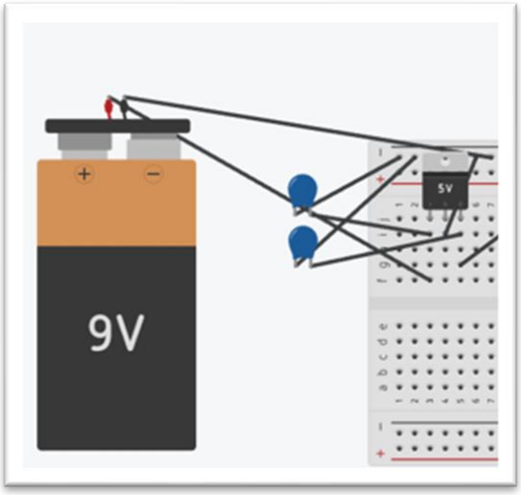


Figure (17): Voltage regulator circuit using tinker cad.

- **Power consumption and fade out:** By using minimum number of gates, we have minimized the power consumption without facing any Fading out.

- **No Noise:** Having a minimum number of tracks for power, we have avoided any noise in the circuit.

- **Representing negative numbers:** We have represented negative numbers in 2 forms. First, 2's complement using LEDs and second, number and sign 7 segment.

- **Division simulation**: Although we couldn't construct the division circuit in hardware, we have designed the circuit on Proteus.

- **Covered Range**: All numbers ranging from -15 to 225 can be represented for the user using LEDs for large numbers and 7 segments for 1 digit ranging from -9 to 9.

7. Cost

Item	Price	Amount	Total
Dip switch 4 input	4	2	8
Dip switch 6 input	5	1	5
74153 (Dual 4-Input to 1-Line Multiplexer)	20	2	40
74151 (8 Input Multiplexer)	12	4	48
LED	4	14	56
Resistance	0.25	44	11
7432 (Quad 2-Input OR Gate)	9	2	18
7408 (Quadruple 2-input AND gates)	9	6	54
# 7486 (Quad 2-input EXCLUSIVE-OR (XOR) Gate)	8	5	40
9V battery	40	1	40
voltage regulator	10	1	10
7483 (4-Bit Binary Full Adder with Fast Carry)	25	4	100
Wires	1	450	450
Capacitors 330nF	3	4	12
7 segments	5	2	10
7448 (BCD to 7-Segment Decoder/Driver Common CATHODE)	25	1	25
Breadboard	6	30	180
Total			**1,107**

Phase 2
Sequence representer

1. Introduction

This report's objective is to describe the sequential circuit's design and implementation, which serves to represent a particular code. Digital circuits known as sequential circuits use memory components to store and process data in a sequential order. They are extensively utilized in many different applications, such as data storage, counters, and control systems.
Our goal in this project is to create a sequential circuit that faithfully depicts a particular code. The representation of the code can be in binary, decimal, or any other format that is preferred. The sequential circuit will oversee storing the code, permitting input and output operations, and carrying out any required modifications or operations on the code.

2. Designing steps

To achieve this, the project will involve different levels of abstraction and design. The following stages will be followed:

- **Determining the code**: to be represented. In our case it was already stated by the TA to be “0312746”
- **State Diagram**: To see how the sequential circuit behaves and changes states, draw a state diagram. Identify the many states that the circuit may be in and specify the circumstances under which states may change. A high-level summary of the circuit's operation is provided by this diagram.
- **Create a state table** based on the state diagram that lists the current state, inputs, the following state, and outputs for each state transition. This table is used as a guide while creating the combinational logic and identifying the memory components (flip-flops) needed to store the code.

- **Designing Combinational Logic**: Create the combinational logic that uses the inputs and current state to determine the outputs and the future state. To implement the necessary logic functions, use Boolean algebra and logic gates. Any required modifications or operations on the code are carried out with this logic.
- **Determine the kind and quantity of memory elements** required to store the code before choosing them. We are already assigned to use D-Flip flop because our last digit is even.
- **Verification and Simulation**: To check the sequential circuit's functionality, use simulation software or tools. Apply various input codes to it, check the accuracy of the storage and retrieval, and confirm the appropriate actions or transformations to validate its behavior. Make that the circuit performs as expected and complies with the requirements.
- The circuit design, the state diagram, the state table, the schematics, the testing processes, and any pertinent notes or observations should all be included in the **documentation**. The sequential circuit's documentation can be used as a guide for future upkeep, troubleshooting, or development.

3. Design

3.1 State diagram

We start by defining the state diagram for our circuit. This diagram will include the states at which the system reaches in a sequential matter. The state diagram for our circuit is shown in **Figure (18).**

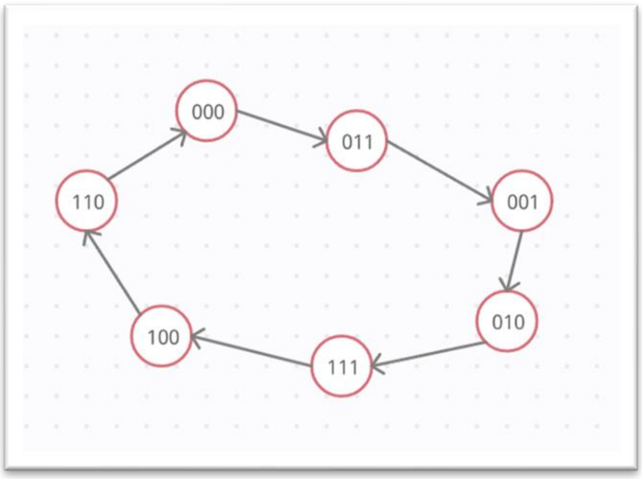


Figure (18): State diagram for our circuit

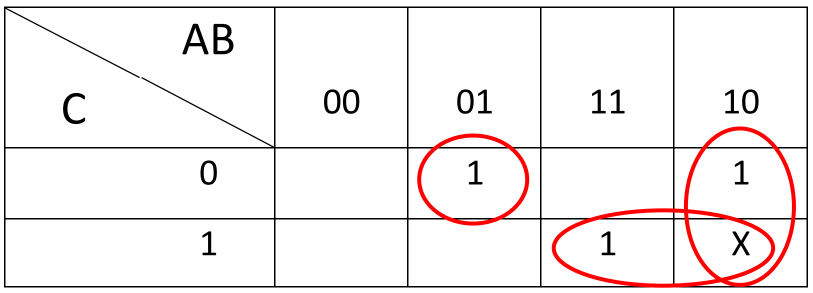
3.2 State table

The state table represents the last and next state for the system at time t and at time t+1 which is the time after a click raise. The state table is shown in the following table.

Current state			Next state		
A	**B**	**C**	**A(t+1)**	**B(t+1)**	**C(t+1)**
0	0	0	0	1	1
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	X	X	X
1	1	0	0	0	0
1	1	1	1	0	0

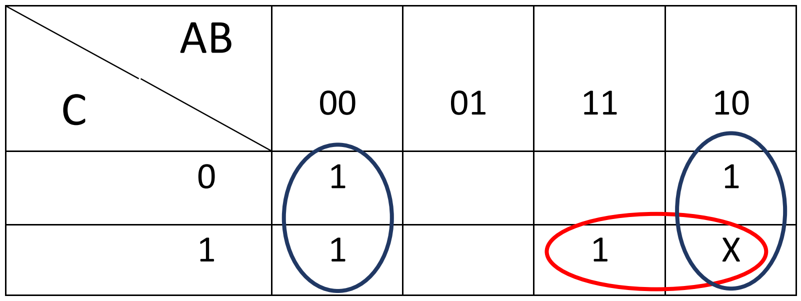
3.3 Flip Flop equation and minimization

From the table, we can obtain the form for the flip flop.



C \ AB	00	01	11	10
0		1		1
1			1	X

$$D_A = AB' + AC + A'BC' = A(B' + C) + A'BC'$$



C \ AB	00	01	11	10
0	1			1
1	1		1	X

$$D_B = B' + A'C'$$

AB / C	00	01	11	10
0	1	1		
1		1		X

$$D_C = A'B + A'C'$$

3.4 Circuit implementation

Using the equations obtained, we used the following implementation for the circuit. First, we connected the NE555 to get the clock for the frequency of the clock. Then the clock was connected to the gates to produce the sequence. The simulation for the circuit is shown in **Figure (19).**

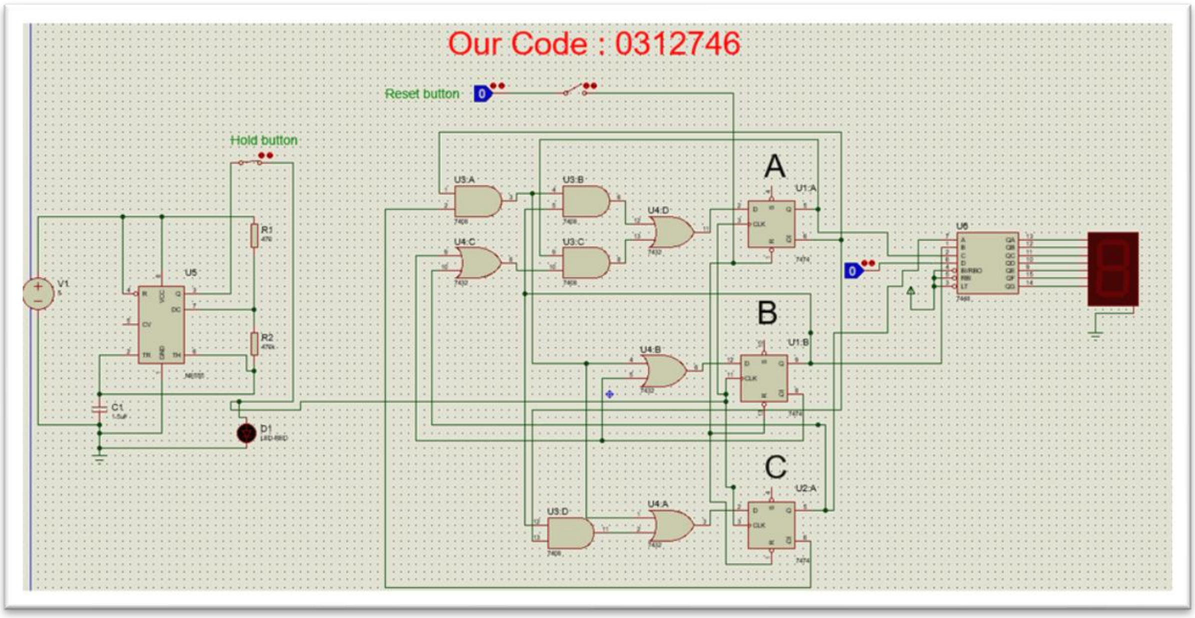


Figure (19): Phase 2 simulation

4. Hardware design and implementation

The implemented circuit is shown in **Figure (20)**.

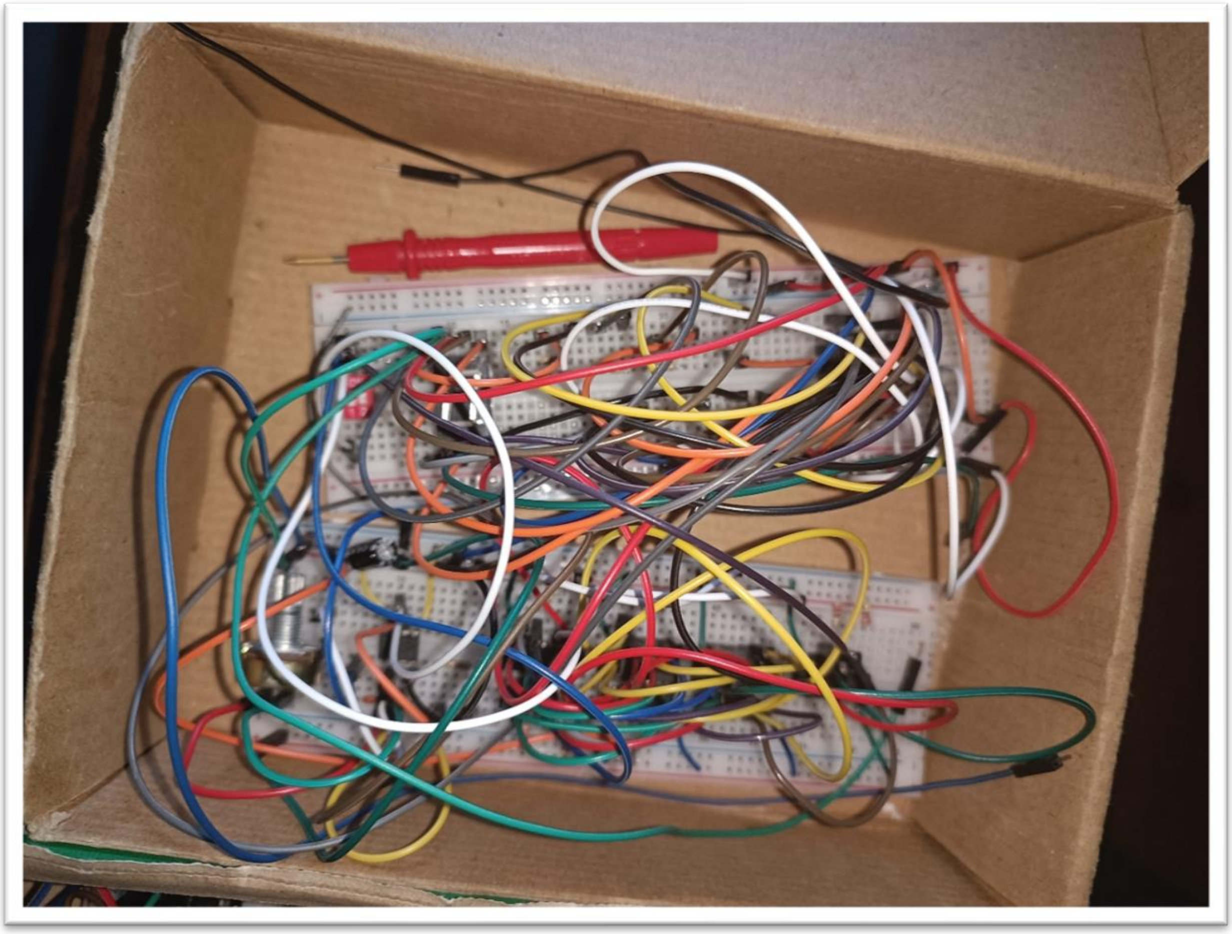


Figure (20): Phase 2 Hardware

5. Test cases and results

For our project, it was important to not only test the sequence but also to test the unused state of the system to check if it's self-correct. The unused case in our case is 101 or 5 in decimal. It was declared to be a don't care condition in our design. The next table represents the system's behavior in that case using the obtained equations:

Loop	A	B	C	A(t+1)	B(t+1)	C(t+1)
1	1	0	1	1	1	0
2	1	1	0	0	0	0

From the previous table, we can conclude that the system directly enters the sequence again after getting an invalid input. Thus, we can describe our system as being self-correct itself.

6. Strengths

- **Frequency controller**: Using a rheostat in the NE555 circuit, we can control the frequency of the clock. This can be helpful for different kinds of representation. In some cases, it would be easier to observe the system in a slow manner and in other cases, slow motion is needed in observing any noise during transition.

- **Hold switch:** The hold switch can be used to hold the clock to get the last presented number with no change. This hold button is based on stopping the clock from reaching the flip-flop to stop the system on a specific state. This would be useful for the user to stop the system at a chosen state.

- **Reset button**: The reset button will get the sequence to 0 and start again.

- **Operation ICs**: Only 2 ICs are used for the operations with the flip flops.

7. Cost

Item	Price	Amount	Total
DIP Switch 3-input	3	1	3
NE555	6	1	6
LED	0.25	1	0.25
Resistance	4	10	40
Wires	1	100	100
Capacitors	5	2	10
Rheostat	1	4	4
7432 (Quad 2-Input OR Gate)	9	1	9
7408 (Quadruple 2-input AND gates)	9	1	9
7 segments	5	2	10
7448 (BCD to 7-Segment Decoder/Driver Common CATHODE)	25	1	25
Total			216.25

Conclusion

As a result, the goals of this logic design project—creating an arithmetic logic unit (ALU) that can add, subtract, multiply, and divide two 4-bit numbers, as well as creating a sequential circuit to display a specific code on a 7-segment display—were both successfully attained.

The initial stage of the project involves carefully considering the necessary arithmetic and logical operations for designing the ALU.

The ALU was able to carry out the required operations precisely and effectively by combining logic gates, multiplexers, and decoders.

The requirements were examined, the requisite logic functions were created, and the circuit was implemented utilizing the proper components.

The ALU proved to be trustworthy and accurate in carrying out the required arithmetic operations after comprehensive testing and validation.

Understanding the code that would be displayed and turning it into appropriate control signals were the tasks of the second phase of the sequential circuit design for the 7-segment display.

The sequential circuit successfully displayed the code on the 7-segment display using flip-flops, logic gates, and segment drivers.

To ensure proper representation of the code, synchronization and timing issues were added into the architecture.
The sequential circuit proved to be fully functional and capable of displaying the code as planned after extensive testing.

Overall, **this logic design project demonstrated the use of basic digital logic principles and methods**. The accomplishment of both phases reveals the capacity to create intricate circuits that carry out mathematical operations and display codes. The project also made clear how crucial it is to plan carefully, use logic, and solve problems effectively to achieve project requirements and design constraints.

The sequential circuit and ALU that were built have real-world uses in a variety of industries, including computer architecture, embedded systems, and digital signal processing.

Recommendations

After working on this project, we found a group of recommendations that we want to represent to any future groups that will implement this project or any similar project.

1. Using short wires for most connections

Tracing is a very important step in working with hardware circuits. It will be easier to trace a circuit that has small, long connections wires. As shown in our project, using long wires results in a very chaotic design for the circuit. Using the wires kit to use short wires will reduce this problem massively.

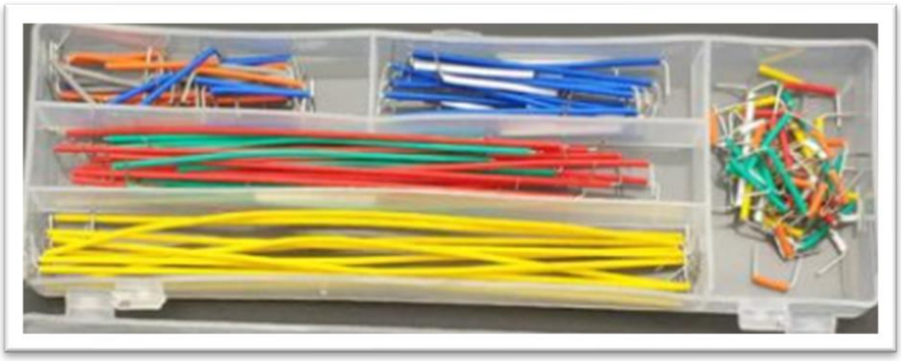


Figure (21): Wires kit with different sizes

2.Using load option in sequential circuit

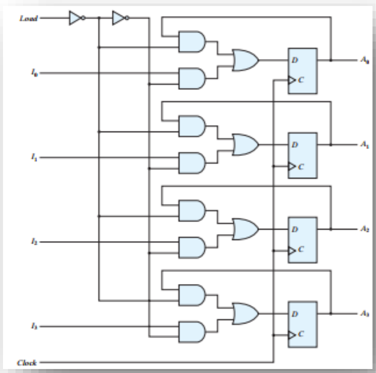


Figure (22): Example of a parallel load circuit

For any project to be usable, it needs to be user friendly. To achieve this in Phase 2, you can use a load option to give the user the ability to give the system a specific input and state to start with. This can easily be achieved using the parallel load very known architecture as shown in **Figure (22)**.

Documentation

Videos on drive link showing how practically the circuits that we mentioned in the report driven.

Our Drive

https://drive.google.com/drive/folders/13KJ6ngf8XgM9ke00TUScwY0I6gTspQdy?usp=sharing

Project on GitHub

https://github.com/Sama289/ALU_and_Custumized_Counter

Acknowledgments

At the end of this report, we would like to thank the people who helped us to reach this point and complete this project in the best possible way:

1. **Dr. Mohammed Omar Mohammed**: As our instructor for this course, he helped us in grasping the concepts of this course and build this project.

2. **Eng. Osama Samy Abdel-Hamid:** As out TA, he helped us in each stage of this project. He can be described as our reference for specific technical problems.

3. **Digital design community on YouTube:** Every video for a concept, an IC, a design or even a quick recap was very useful and helpful. We, as a group, hope to contribute to this community in the next years.

Team

Positions

For any team it's important to notice the work done by each member. These positions were chosen at the start of the project to help in clarifying the work of each member.
Knowing that we were working together in parallel in both phases in online meetings and on campus.

Position	**Person**
Hardware keeping	Amir
Hardware documentation: Mapping Proteus and revise	Bokhary
Report documentation	Sama
Laptop, AVO	Sama Amir Bokhary
Hardware documentation: Resources and uses	Amir

Position	**Person**
Design of Phase 2 **All the remaining Hardware Parts** **7 segment and Decoder in Both Phases Hardware**	Bokhary
Multiplication and division Design simulation **Multiplication hardware implementation** **Mux 8 to 1** **All Phase 2 Hardware**	Amir
Adder/ Subtractor, OR , increment simulation and hardware implementation **LEDs (Outputs) Hardware** **Optimization in Phase 2**	Sama

IEEE references

REFERENCES

[1] Mano, M. M., & Ciletti, M. D. (2019). Digital Design: With an introduction to the Verilog HDL. Pearson.

[2] YouTube. (2013). YouTube. Retrieved August 15, 2023, from https://www.youtube.com/watch?v=6L9hDBu3pyE.

[3] YouTube. (2022). YouTube. Retrieved August 17, 2023, from https://www.youtube.com/watch?v=ap0RMkqHWHQ

[4] John F. Wakerly, Digital Design: Principles and Practices. 4th Ed. Pearson, 2005.

[5] R. Katz and G. Boriello, Contemporary Logic Design. 2nd Ed. Pearson, 2005.

[6] S. Brown and Z. Vranesic, Fundamentals of Digital Logic with Verilog Design. 3rd Ed. SEM, 2013

[7] Virtual lab - IIT kharagpur. Virtual Lab for Computer Organisation and Architecture. (n.d.). https://cse.iitkgp.ac.in/~chitta/coldvl/alu.html

[8] Brown, S. D., & Vranesic, Z. G. (2014). Fundamentals of digital logic with Verilog design. McGraw-Hill Higher Education