Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Digital Electronics and Computer Organization Laboratory

ENCS2110

Report #3

Experiment No. 9 – A Simple Security System Using FPGA

Prepared by :

Mohammed Jamil Saada **Number : 1221972**

Partner :

Laith Nimer **Number : 1213046**

Instructor : **Dr. Abualseoud Hanani**

Teaching assistant: **Eng. Hanan Awawdeh**

Section : **4**

Date : **04 / 05 / 2024**

Abstract

In this experiment, we will review our knowledge of Verilog HDL in general, and QUARTUS Software in particular. In this experiment we will simulate a simple security system using Quartus software. Then, we will program and download this system on the FPGA board. This security system is a 2-digit digital lock, the user enter number of two digits, each digit has a range from 0 to 3, then if the number that user entered is correct the green led will be on; and the system allowing us to pass. Otherwise the red led will be on; and the system blocking us from passing.

At the end of this lab, my theoretical background will be proved practically so, I will be able to simulate, and use Verilog HDL and Quartus software to implement any program.

Table of Contents

Table of Figures

List of Tables

1. Theory

1- The Security System Architecture

This Security System is simply a 2-digit digital lock. The user enters a number of two digits, such that, the digit ranges from 0 to3. Thus, every digit has a lower limit of 0 and upper limit of 3. The number is entered using a keypad (using the 91 switch keys build in FPGA). Each digit is represented by a 7-segment display and if the total number entered on the displays equals to XX a green led is on; allowing us to pass. Otherwise, a red led is always on; blocking us from passing [1].

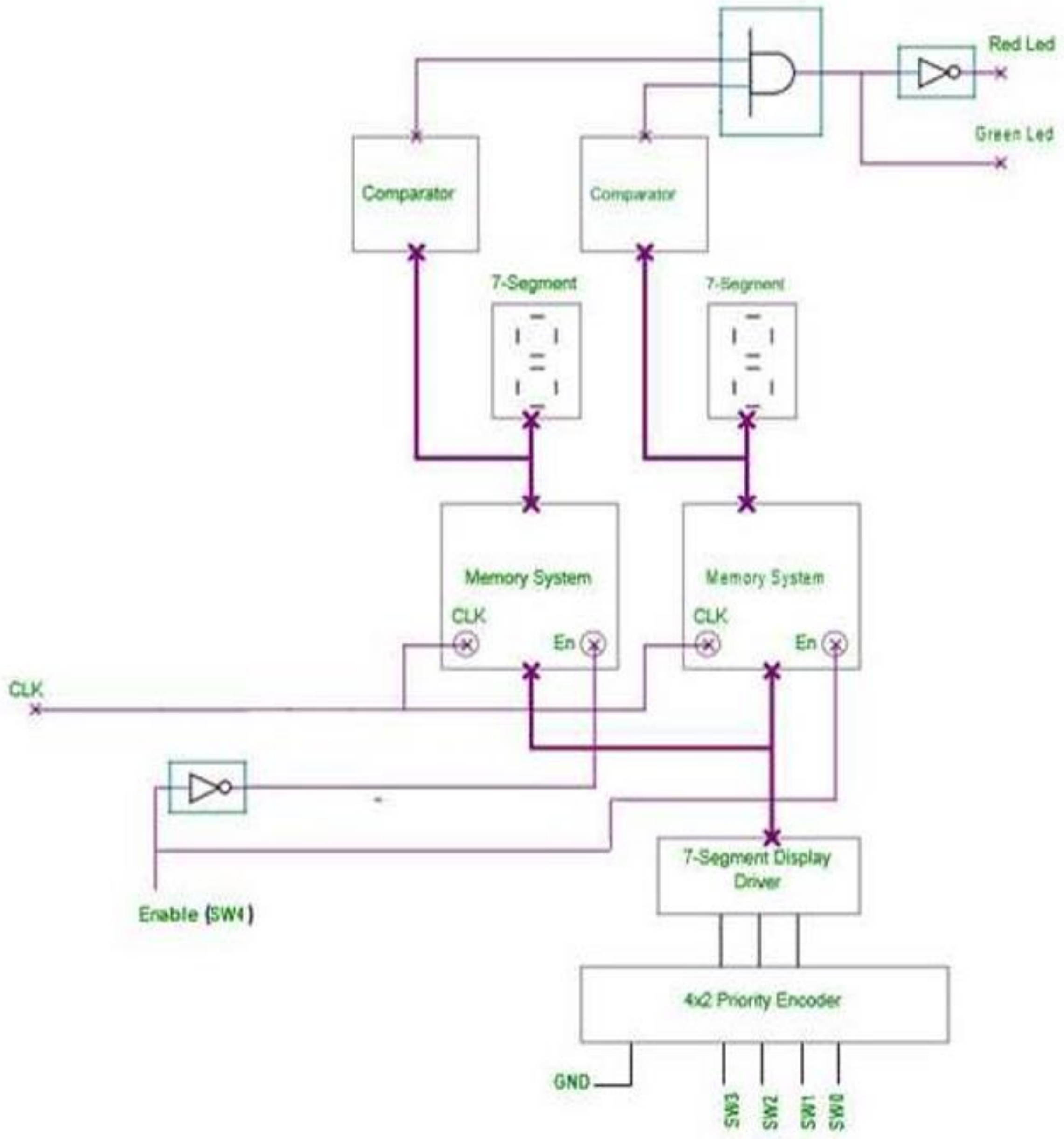


Figure 1 : The Security System Architecture (Source : Lab Manual)

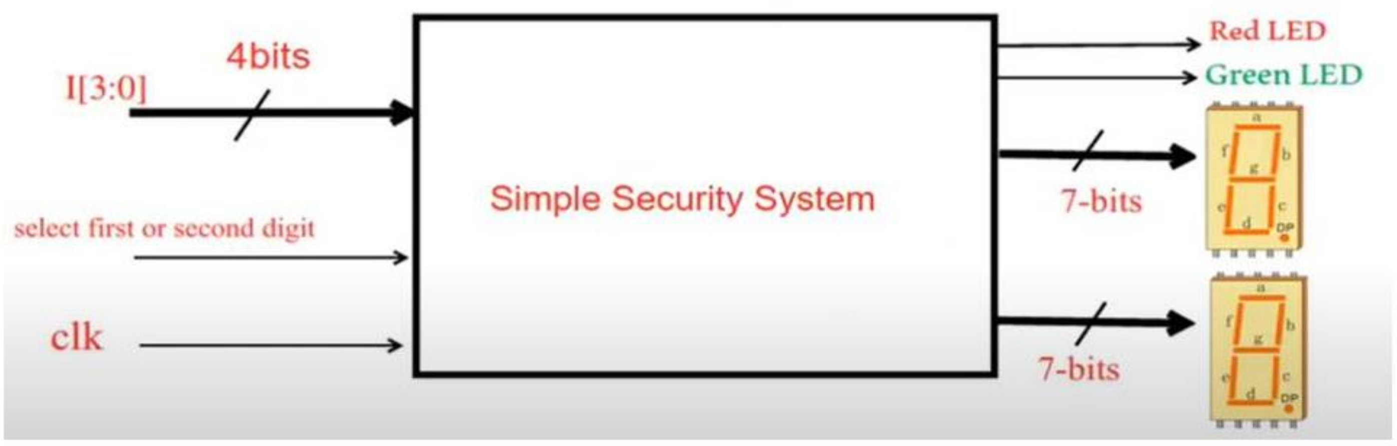


Figure 2 : High level view of the system (Source : Lab Manual)

2- 4×2 Priority Encoder

A 4 to 2 priority encoder has 4 inputs: Y3, Y2, Y1 & Y0, and 2 outputs: A1 & A0. Here, the input, Y3 has the highest priority, whereas the input, Y0 has the lowest priority. In this case, even if more than one input is '1' at the same time, the output will be the (binary) code corresponding to the input, which is having higher priority. The truth table for the priority encoder is as follows [2] :

Inputs				Outputs	
Y3	**Y2**	**Y1**	**Y0**	**A1**	**A0**
1	**X**	**X**	**X**	**1**	**1**
0	**1**	**X**	**X**	**1**	**0**
0	**0**	**1**	**X**	**0**	**1**
0	**0**	**0**	**X**	**0**	**0**

Table 1 : Truth Table of 2x4 priority encoder

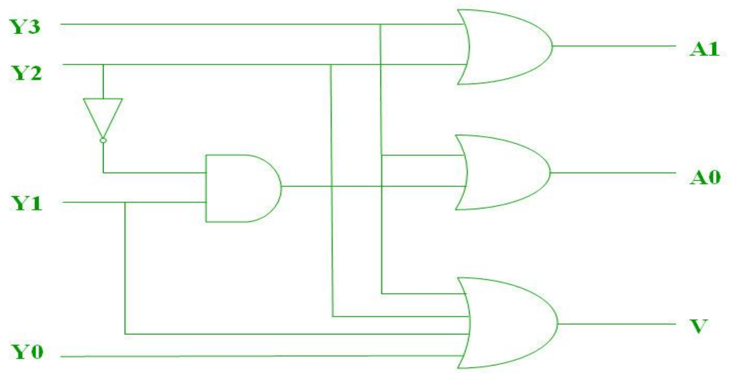


Figure 3 : 2x4 priority Encoder circuit

3- Enable Port

The purpose of this port is to allow the user to select which memory system is active, and hence which 7-segment display to use, for example, if SW4 is high then the En pin of the first memory system is enabled and ready to read the user input on the 4×2 priority encoder [1].

4- Segment display driver

Seven segment decoder / driver is a digital circuit that can decode a digital input to the seven segments format and simultaneously drive a 7 segment LED display using the decoded information. What that will be displayed on the 7 segments display is the numerical equivalent of the input data. For example, a BCD to seven segments decoder driver can decode a 4-line BCD (binary coded decimal) to 8-line seven segment format and can drive the display using this information. For example, if the input BCD code is 0001, the display output will be 1, for 0010 the display output will be 2 and so on [3].

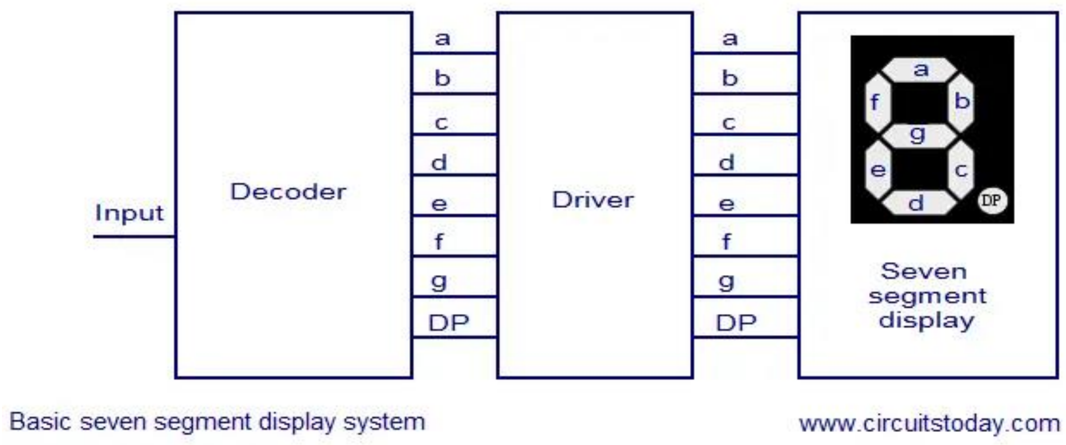


Figure 4 : 7-segment decoder.

5- Memory System

The purpose of this system is to ensure that the value selected by the user to display on a certain 7-segment is kept there when the user switches to select another 7-segment. Each memory system consists of seven D-flip-flops and 2×1 MUXs as shown in Figure 5. When the Enable pin equals 0, the output of each DFF becomes its input at every clock cycle, On the other hand, when the Enable pin becomes 1, the data coming from the 7-segment driver is stored in the each DFF. The output of each DFF is sent on a data bus to a 7-segment display [1].

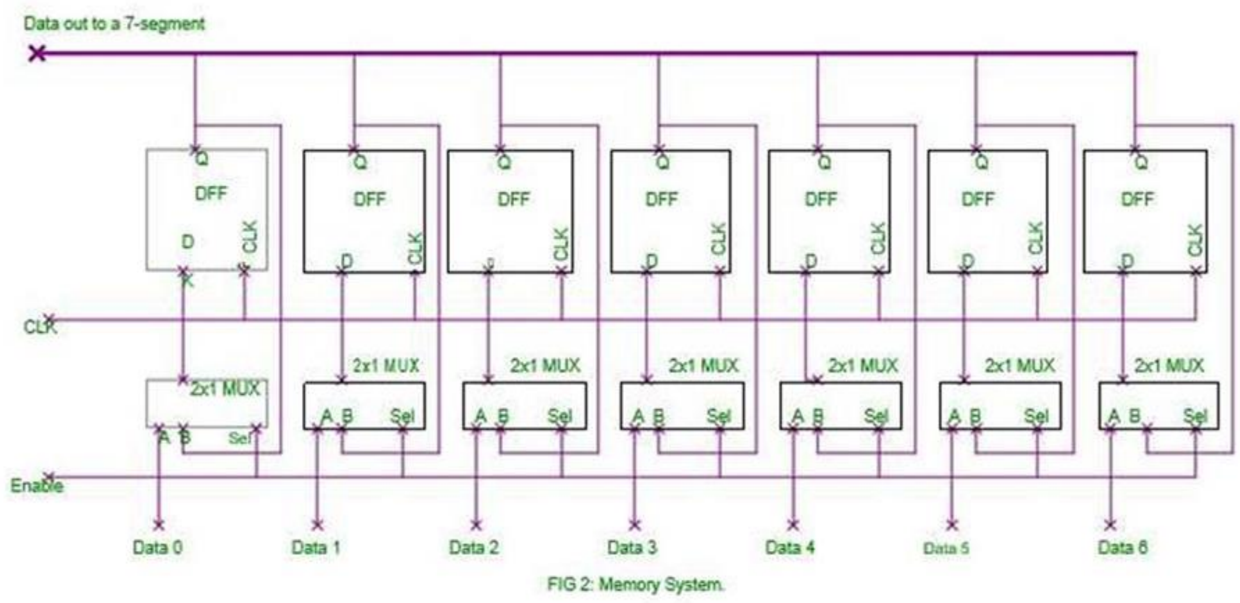


Figure 5 : Memory System (Source : Lab Manual)

6- Comparator

The input of each 7-segment display is connected also to a comparator. Every comparator has a built-in value (reference) which is compared with the value of the 7-segment display. If both values are equal, then the output of comparator is 1, and it is 0, otherwise. For example, if one of the comparators has a reference value equals 5, then its output will be 1 if and only if the input is equal to 7’b0100100 (which is the value of 5 in the 7-segment display). The purpose of the comparator is to lock / unlock the security system [1].

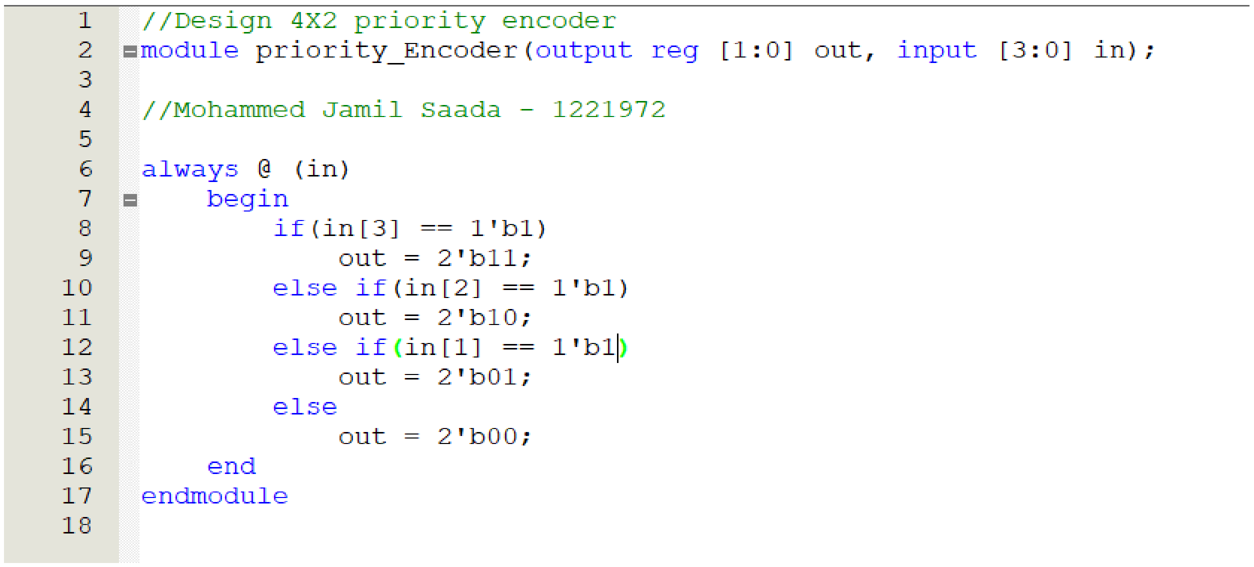
7- 2-input AND gate

This AND gate will make sure that the two 7-segment displays have the correct combination. In other words, if each comparator output is “1”, then the AND gate output will be “1”, and the green light is ON. Otherwise, the red light will be always ON [1].

2. Procedure and Discussion

2.1 Design a 4×2 priority Encoder

We designed a 4×2 priority encoder using QUARTUS software as following :



```
//Design 4X2 priority encoder
module priority_Encoder(output reg [1:0] out, input [3:0] in);

//Mohammed Jamil Saada - 1221972

always @ (in)
    begin
        if(in[3] == 1'b1)
            out = 2'b11;
        else if(in[2] == 1'b1)
            out = 2'b10;
        else if(in[1] == 1'b1)
            out = 2'b01;
        else
            out = 2'b00;
    end
endmodule
```

Figure 2. 1 : 2x4 Priority Encoder – Code

The priority encoder waveform is shown in Figure 2.2 below :

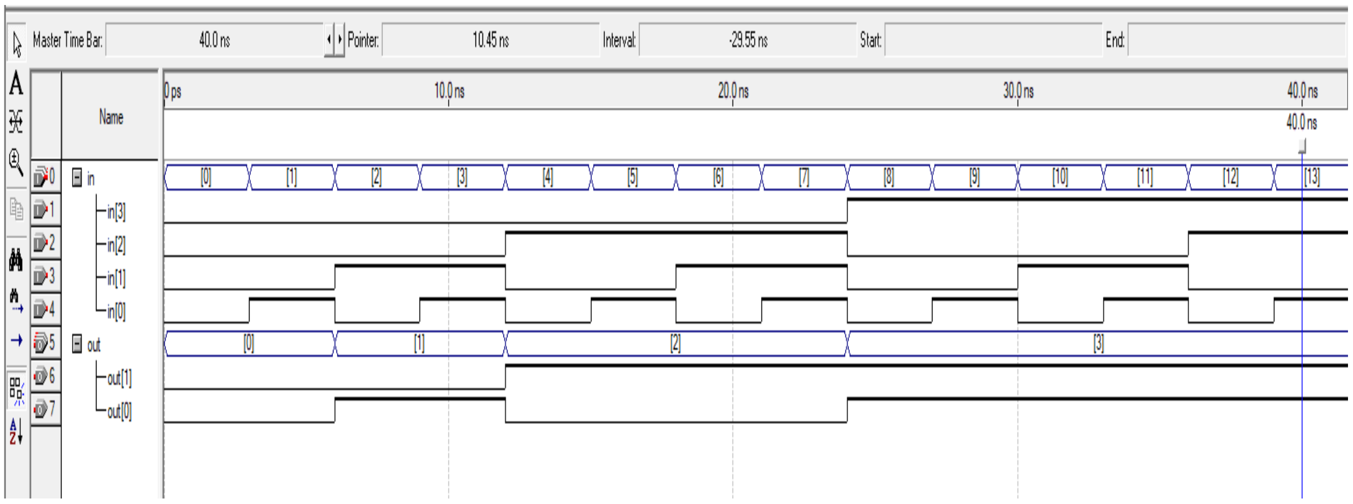


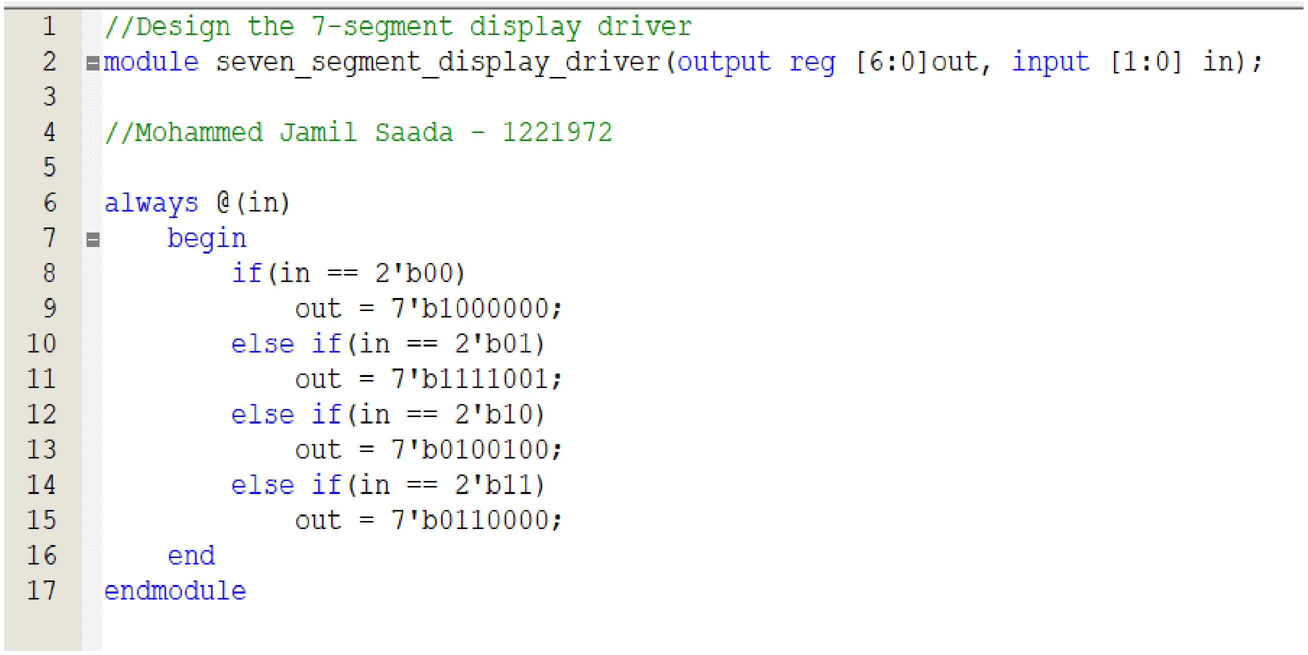
Figure 2. 2 : 2x4 Priority Encoder – Waveform

Discussion :

As the result shown above, the 2×4 priority encoder works properly. There is four inputs and two outputs, and to check which highest priority bit is HIGH to be encoded, then start the check process from the most significant bit to the least one, and when discovery the first occurrence of HIGH, so this is the highest priority bit to be encoded.

2.2 Design a 7-segment display driver

We designed a 7-segment display driver using QUARTUS software as following :



```
//Design the 7-segment display driver
module seven_segment_display_driver(output reg [6:0]out, input [1:0] in);

//Mohammed Jamil Saada - 1221972

always @(in)
    begin
        if(in == 2'b00)
            out = 7'b1000000;
        else if(in == 2'b01)
            out = 7'b1111001;
        else if(in == 2'b10)
            out = 7'b0100100;
        else if(in == 2'b11)
            out = 7'b0110000;
    end
endmodule
```

Figure 2. 3 : 7-Segment Display Driver – Code

The 7-Segment display driver waveform is shown in Figure 2.4 below :

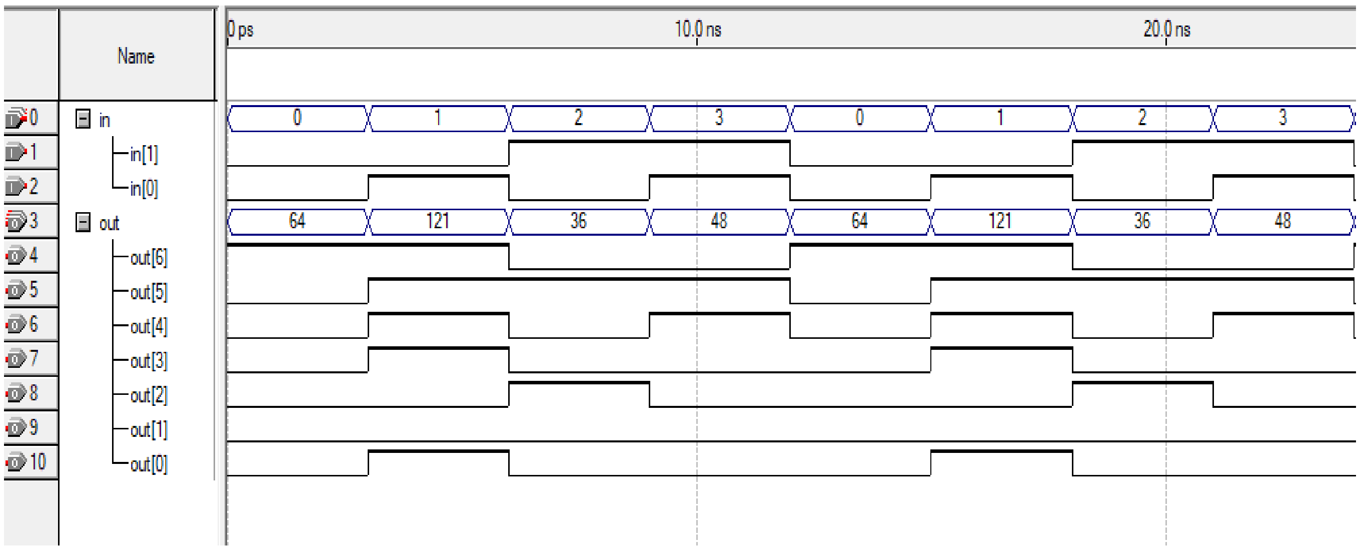


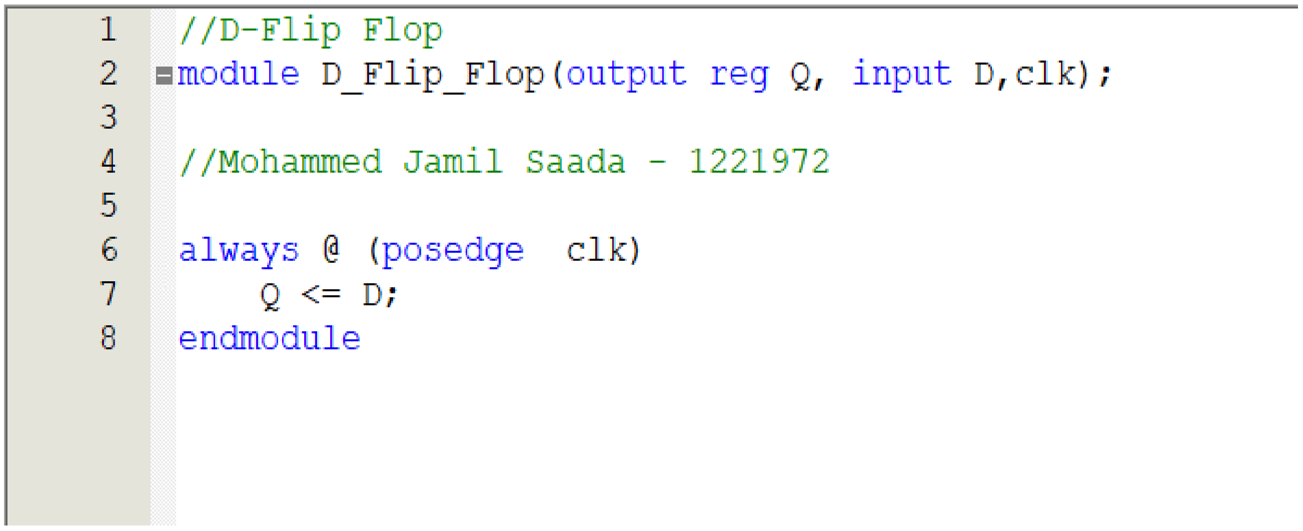
Figure 2. 4 : 7-Segment Display Driver - Waveform

Discussion :

In this section we design a Quartus code for 7-Segment decoder (2-bit to 7-segment decoder). Then, with 2 bits input the digit can be in range 0 to 3 [2’b00 , 2’b11], which mean that we have four cases (four digits – 0,1,2,3) each of them has a unique code in 7-Segment. So, we check the input value, then depends on it to determine the value of each bit in the 7-segment display.

2.3 Design a D – Flip Flop

We designed a D – Flip Flop using QUARTUS software as following :



```
//D-Flip Flop
module D_Flip_Flop(output reg Q, input D,clk);

//Mohammed Jamil Saada - 1221972

always @ (posedge  clk)
    Q <= D;
endmodule
```

Figure 2. 5 : D - Flip Flop – Code

The D – Flip Flop waveform is shown in Figure 2.6 below :

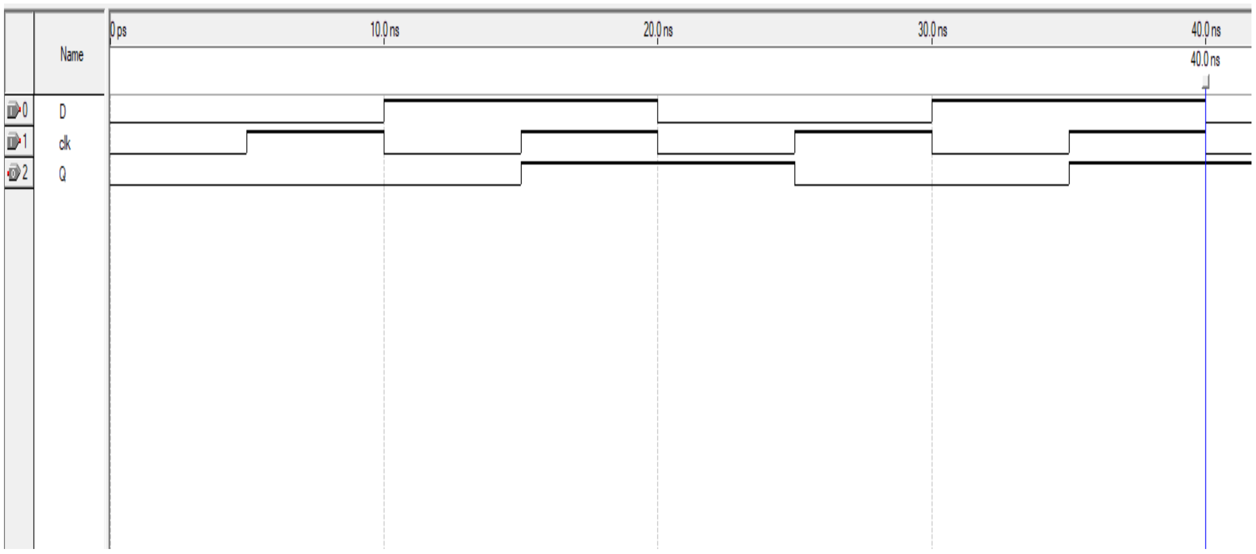


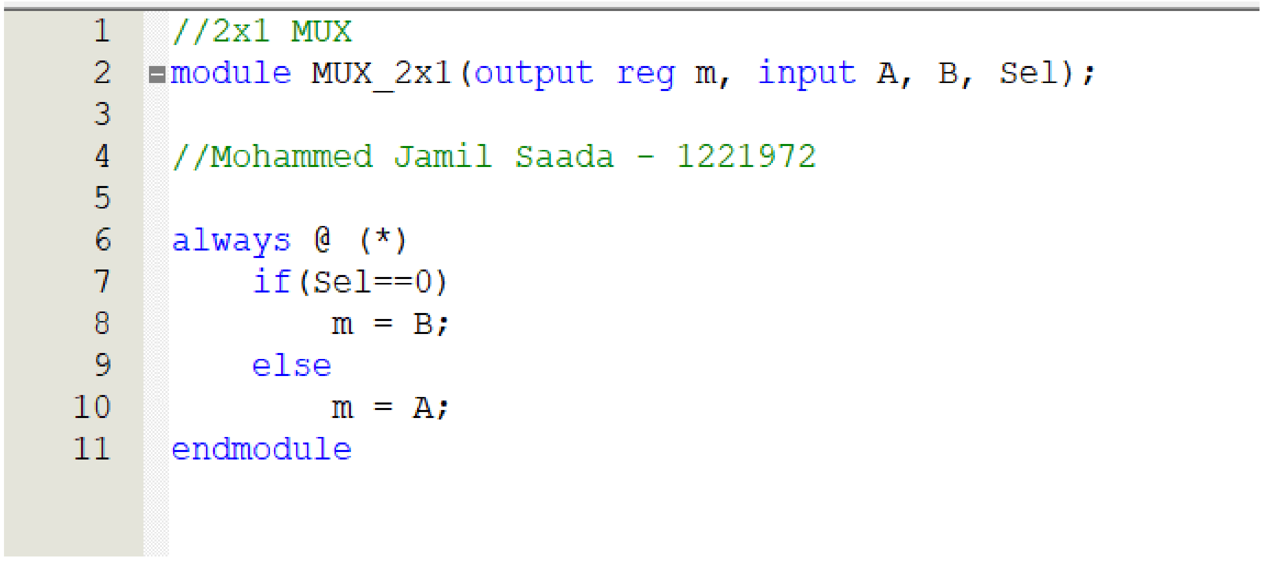
Figure 2. 6 : D - Flip Flop - Waveform

Discussion :

As we shown in the Figure 2.6 above [Waveform of the D – Flip Flop module], the output Q change at every positive edge of the clock to be the same of the value of input D. This is the D – Flip Flop that we know, so this code is work properly for it.

2.4 Design a 2×1 MUX

We designed a 2×1 MUX using QUARTUS software as following :



```
//2x1 MUX
module MUX_2x1(output reg m, input A, B, Sel);

//Mohammed Jamil Saada - 1221972

always @ (*)
    if(Sel==0)
        m = B;
    else
        m = A;
endmodule
```

Figure 2. 7 : 2x1 MUX – Code

The 2×1 MUX waveform is shown in Figure 2.8 below :

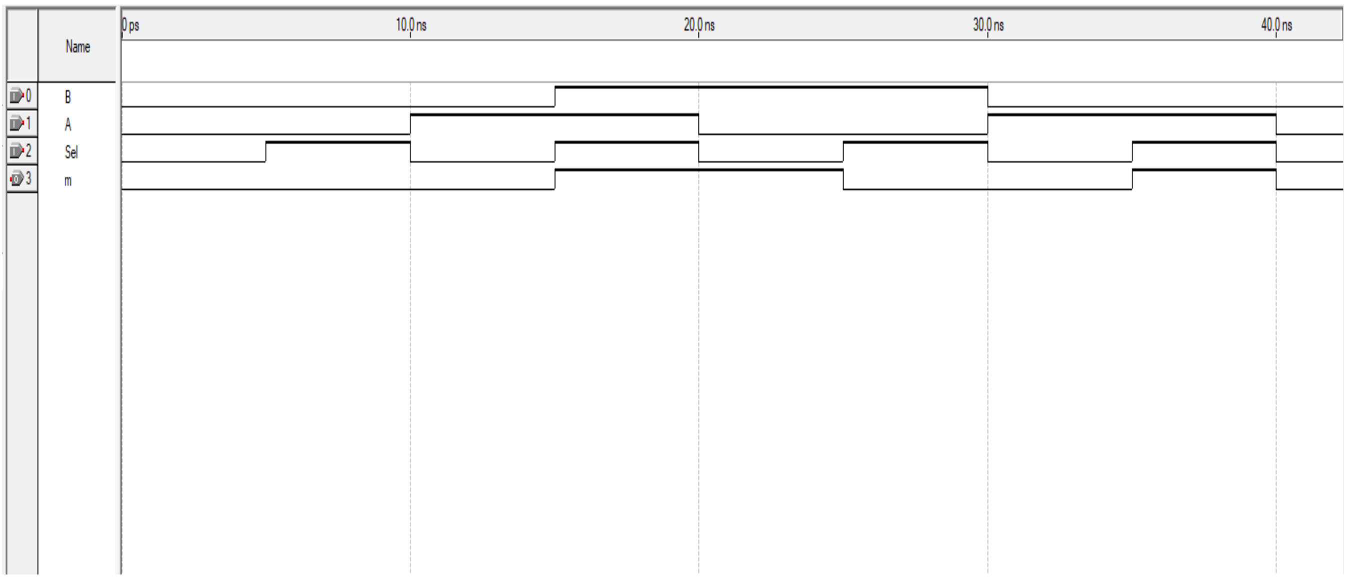


Figure 2. 8 : 2x1 MUX – Waveform

Discussion :

In this section we implement a 2×1 Multiplexer using Quartus software, that has two inputs A and B and the third input Sel that represent the selection line. So, this MUX determines the value of the output if equal to B or A based on the selection line value, if the Sel=0, then the output equal to the least significant bit input which is B, and if the Sel=1, then the output equal to the most significant bit input which is A.

2.5 Design a Memory System

We Designed a Memory System using the block diagrams for the above modules as follow :

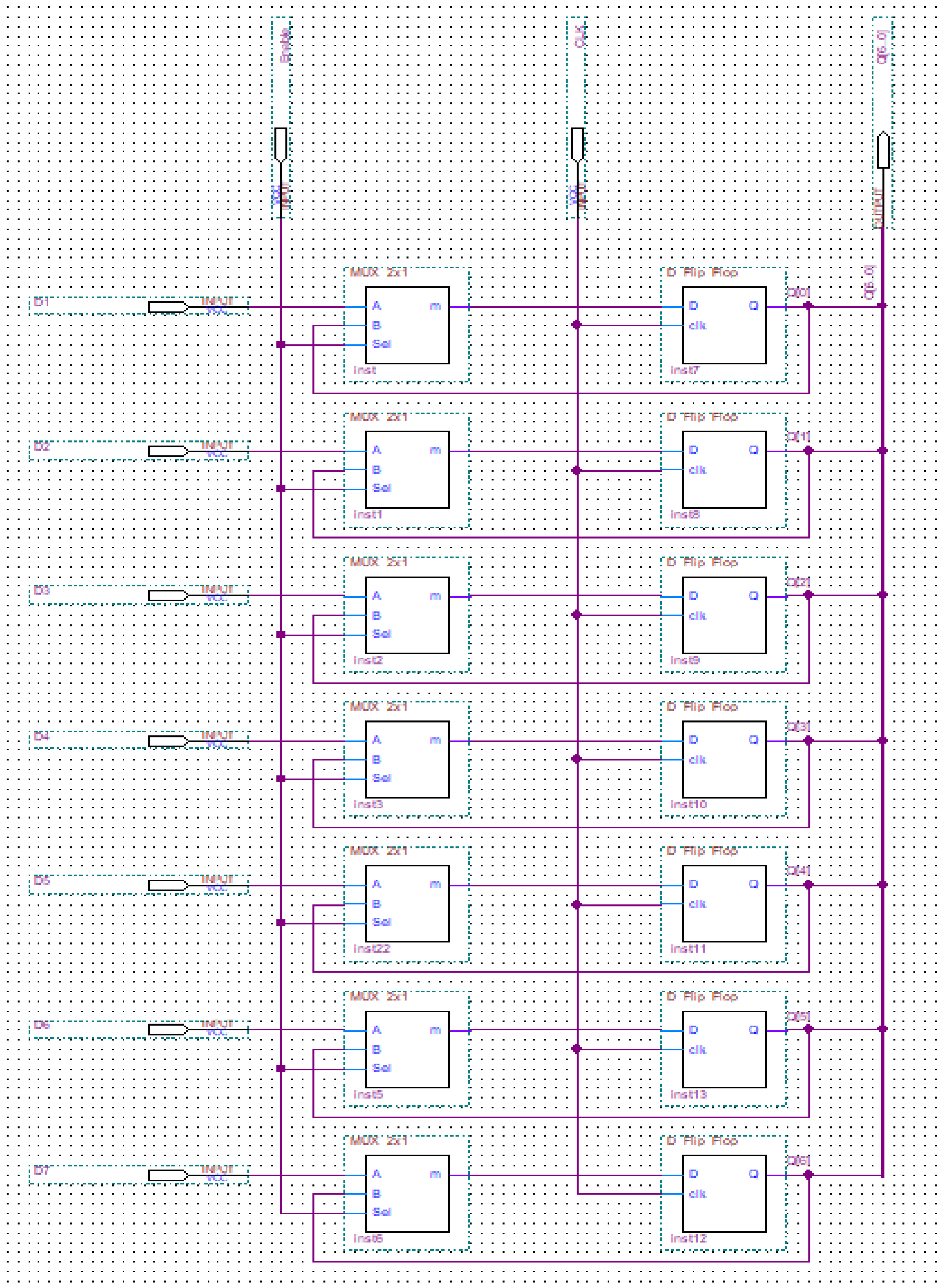


Figure 2. 9 : Memory System Block Diagram

The Memory System waveform is shown in Figure 2.10 below :

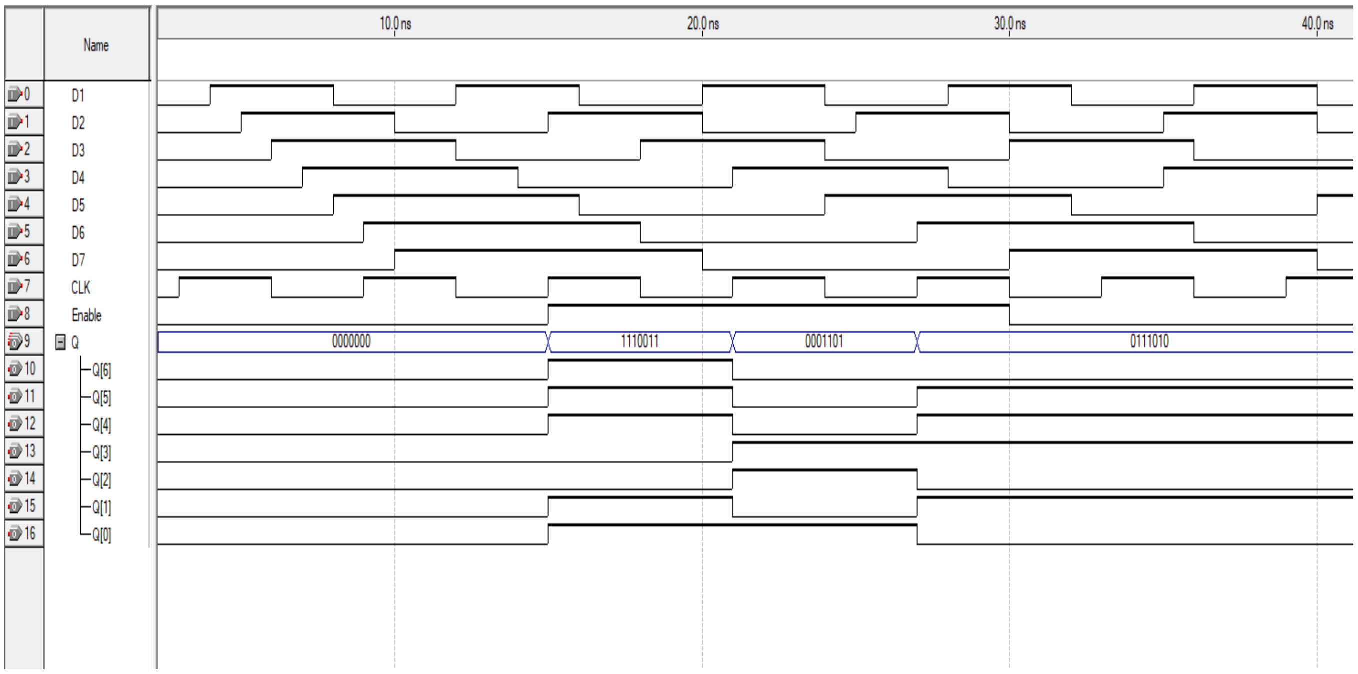


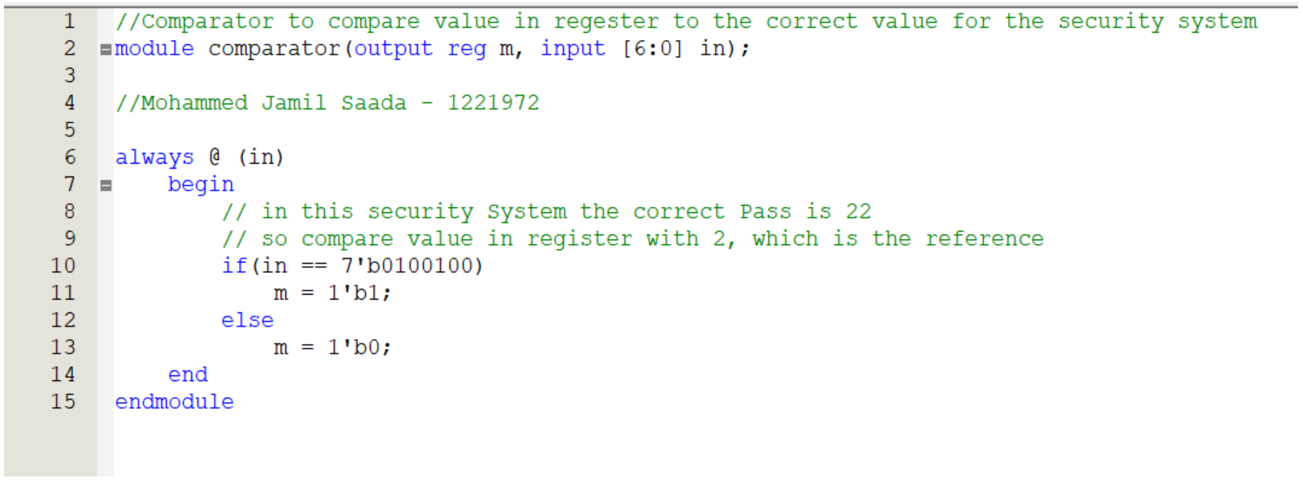
Figure 2. 10 : Memory System – Waveform

Discussion :

As we show above, we use the Block diagram of 2×1 MUX and D – Flip Flop to build a Memory System Block diagram. The memory value change when the enable is HIGH and there is a positive edge of clock. Otherwise, the memory keeps its value without change as we show this in the waveform of the Memory System in Figure 2.10 above.

2.6 Design a Comparator

We designed a comparator using QUARTUS software as following :



```verilog
//Comparator to compare value in regester to the correct value for the security system
module comparator(output reg m, input [6:0] in);

//Mohammed Jamil Saada - 1221972

always @ (in)
    begin
        // in this security System the correct Pass is 22
        // so compare value in register with 2, which is the reference
        if(in == 7'b0100100)
            m = 1'b1;
        else
            m = 1'b0;
    end
endmodule
```

Figure 2. 11 : Comparator – Code

The Comparator waveform is shown in Figure 2.10 below :

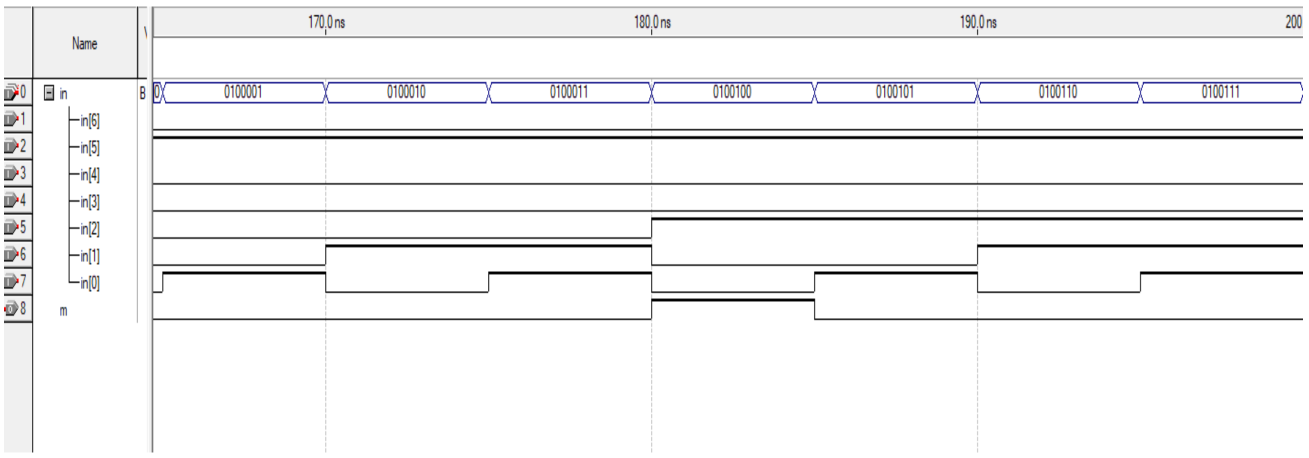


Figure 2. 12 : Comparator – Waveform

Discussion :

In this section we implement a comparator, to compare the value in the register (memory system) – value that the user entered – with the correct value of the security system. In this implementation we compare the value with 2 as a reference, in other words we suppose that the correct password for this system is 22.

2.7 Build the Full Design

We built and designed the security system Using the components that we built in the previous sections, as the following figure 2.13 :

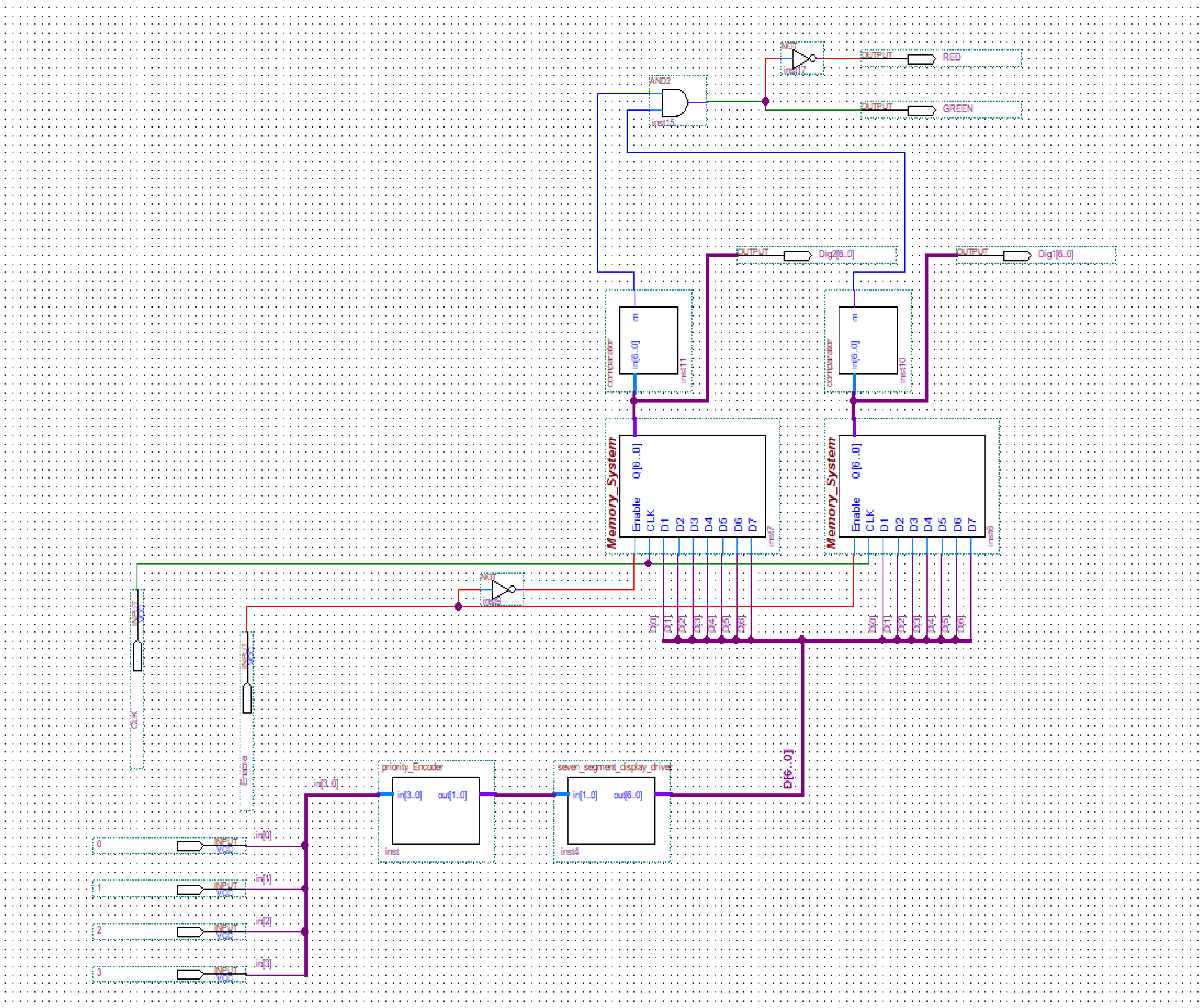


Figure 2. 13 : The Security System Final Block Diagram

The Security System waveform is shown in Figure 2.14 below :

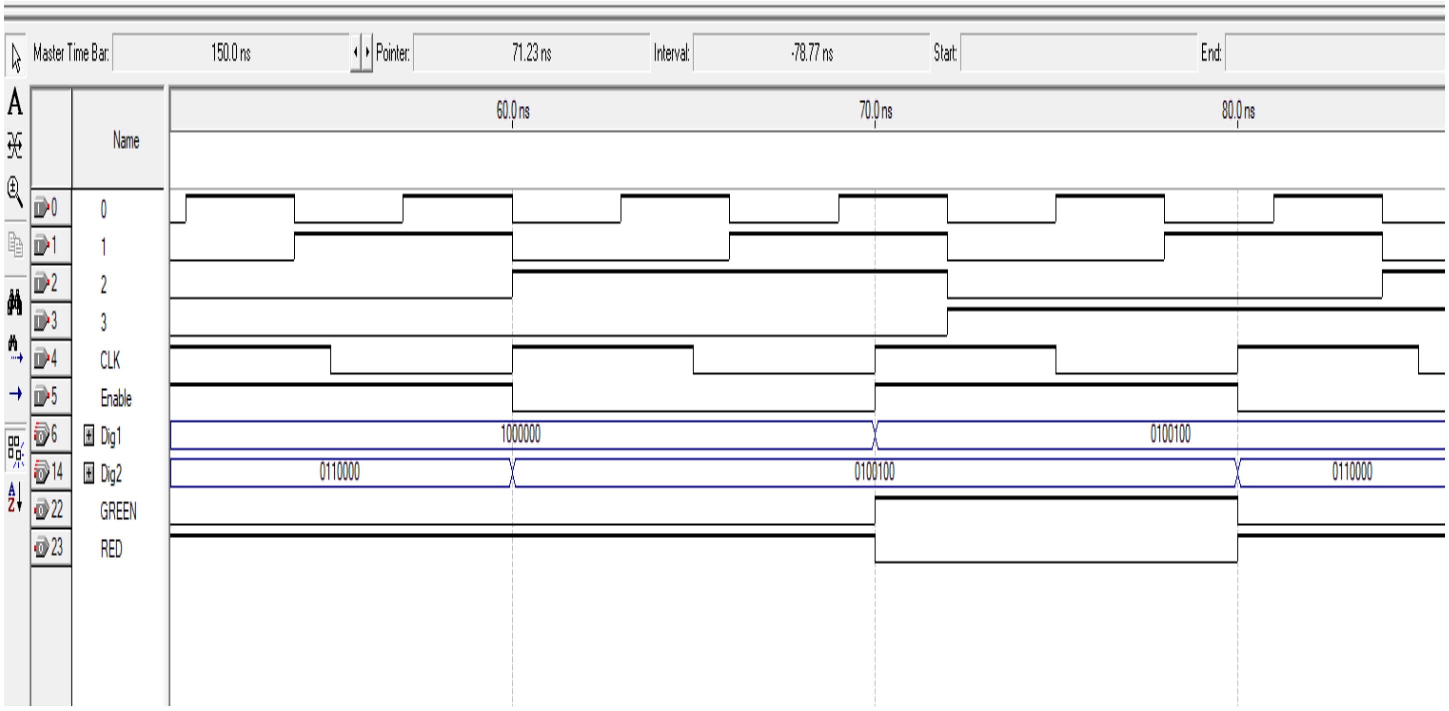


Figure 2. 14 : The Security System Final Block Diagram - Waveform

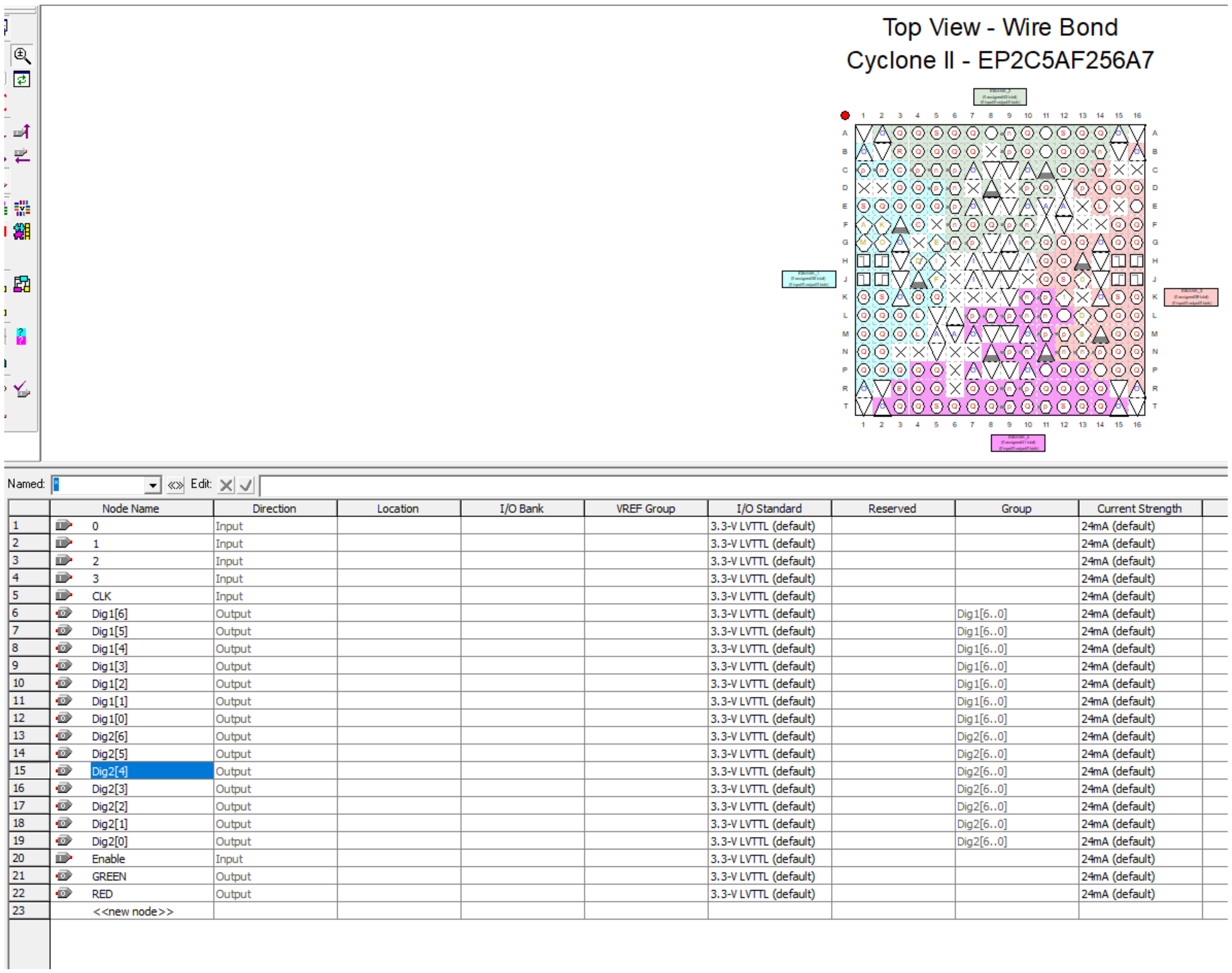
Discussion :

In this section, we built and designed the Security System Final Block Diagram, by using the components that we built in the previous sections, as we show in the figure 2.13 above. We can see how this security system work by trace the waveform in figure 2.14 above, so we can see that the Green LED will be HIGH (ON) when the two digits are (0100100) which mean 2 in decimal. So, the password for this security system is (22). Otherwise, the RED LED will be HIGH (ON).

2.8 Download System on the FPGA Board

Finally, after we design and build the Security System Final Block Diagram, we assign pins values to the Switches and LEDs in the FPGA board, and then download it to the FPGA board.

1) Assign pins values to the switches / LEDs in the FPGA board :

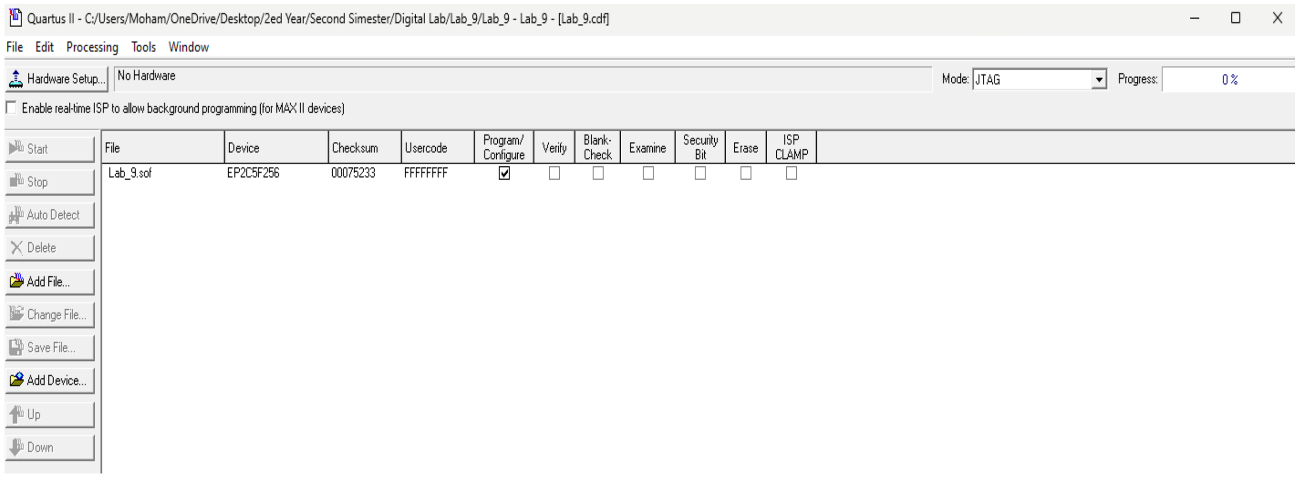


	Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Group	Current Strength
1	0	Input				3.3-V LVTTL (default)			24mA (default)
2	1	Input				3.3-V LVTTL (default)			24mA (default)
3	2	Input				3.3-V LVTTL (default)			24mA (default)
4	3	Input				3.3-V LVTTL (default)			24mA (default)
5	CLK	Input				3.3-V LVTTL (default)			24mA (default)
6	Dig1[6]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
7	Dig1[5]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
8	Dig1[4]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
9	Dig1[3]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
10	Dig1[2]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
11	Dig1[1]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
12	Dig1[0]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
13	Dig2[6]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
14	Dig2[5]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
15	Dig2[4]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
16	Dig2[3]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
17	Dig2[2]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
18	Dig2[1]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
19	Dig2[0]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
20	Enable	Input				3.3-V LVTTL (default)			24mA (default)
21	GREEN	Output				3.3-V LVTTL (default)			24mA (default)
22	RED	Output				3.3-V LVTTL (default)			24mA (default)
23	<<new node>>								

Figure 2. 15 : Assign Pins Values to FPGA Board

Here, we choose the Switches and LEDs that we want to use in the FPGA board, by enter the number of pin (Switch / LED / Seven Segment display) to the column “Location” as we see in the figure 2.15 above.

2) Download the System to the FPGA Board :


Quartus II - C:/Users/Moham/OneDrive/Desktop/2ed Year/Second Simester/Digital Lab/Lab_9/Lab_9 - Lab_9 - [Lab_9.cdf]

Hardware Setup... No Hardware

Mode: JTAG Progress: 0 %

☐ Enable real-time ISP to allow background programming (for MAX II devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase	ISP CLAMP
Lab_9.sof	EP2C5F256	00075233	FFFFFFFF	✓	☐	☐	☐	☐	☐	☐

Figure 2. 16 : Download the System to FPGA

After we assigned the pins, we download the program (Security System) to the FPGA Board, by connecting the board with the computer using USB cable, and then click start as we see in the figure 2.16 above, and wait until the progress becomes 100%. Then the FPGA Programed to the security system.

Discussion :

Finally, after we implement all the components and the Security System Final Block Diagram, we assigned the input Pins to the switches, and the output Pins to the LEDs or seven segment display on the FPGA board. Then, we downloaded the program (this system) to the FPGA and use it as a security system.

3. Conclusion

After completing this experiment all the objectives are obtained. Now, I can implement and simulate any program using QUARTUS software. And then I can assign and download any program on the QUARTUS to the FPGA board to use this board as I need. In this experiment, first I implemented the basic components such as priority encoder, multiplexer, seven segment driver (decoder), D – Flip Flop and comparator. Then I implement the memory system using the components of the multiplexer and D – Flip Flop that I implemented previously. Finally, I use all the previous components to build and design the Final Block Diagram of the Security System as we explained previously.

After we implement the components and trace the results, we noticed that the results are similar to theoretical results that we expected, so we implement them correctly and there is no problem or any issue in the implementation for any component or on the whole system (Final Block Diagram).

4. References

[1] : Manual for Digital Electronics and Computer Organization Lab, 2024, Birzeit University.

[2] : GeeksforGeeks : Priority Encoder in digital logic, Retrieved May 3, 2024 from https://www.geeksforgeeks.org/encoder-in-digital-logic/

[3] : Circuits To Day : segment display driver in digital logic, Retrieved May 3, 2024 from https://www.circuitstoday.com/7-segment-display-driver