



**Faculty of Engineering and Technology Electrical and
Computer Engineering Department DIGITAL
ELECTRONICS AND COMPUTER ORGANIZATION
LABORATORY ENCS2110**

Experiment No. 9

A Simple Security System Using FPGA

Prepared by:

Name: Anas Al Sayed

Number: 1221020

Partners:

Name: Ahmad Toffaha

Number: 1220476

Name: Layan Ayyash

Number: 1210859

Instructor: Dr. Hanya Radwan

Assistant: Ms. Katy Sadi

Section: 10

Date: 11/5/2025

Abstract:

This experiment focuses on the design and implementation of a simple digital security system using FPGA technology. The system functions as a 2-digit lock, allowing user input via a keypad and displaying values on 7-segment displays. Key components include a 4x2 priority encoder, segment display driver, memory units using D flip-flops and multiplexers, comparators, and a logic AND gate. The system grants access by lighting a green LED when the correct code ("2 2") is entered and a red LED for incorrect attempts. The project emphasizes practical application of digital logic design and FPGA programming using Quartus, enhancing understanding of real-world digital systems.

I. Table of Contents

Abstract:	I
II. Table of Figures	III
III. List of Tables.....	III
1.Theory	4
1.1 4x2 Priority Encoder:	4
1.2 Enable Port:.....	5
1.3 Segment display driver:.....	5
1.4 Memory System:	5
1.5 Comparator:	6
1.6 2-input AND gate:	6
2.Procedure & Discussion.....	7
2.1 Design a 4 x 2 priority encoder:	7
2.2 7-segment display driver:	8
2.3 D- Flip Flop:	9
2.4 2x1 MUX:.....	11
2.4 Memory System:	12
2.5 Comparator:.....	14
2.6 Security System:.....	15
2.7 Discussion:	17
3.Conclusion	17
4.References	18
5. Appendices.....	18

II. Table of Figures

Figure 1:block diagram of 4x2 Priority Encoder	4
Figure 2:segment decoder.	5
Figure 3:Memory System.	6
Figure 4:4 x 2 priority Block Diagram	7
Figure 5:Priority Encoder Verilog Code.	7
Figure 6:Simulation for 4 x 2 priority encoder	8
Figure 7:7-segment display driver block diagram	8
Figure 8:7-Segment Display Driver Verilog Code.....	9
Figure 9:7-segment display driver.	9
Figure 10:D Flip Flop Block Diagram.....	10
Figure 11:D Flip Flop Verilog Code.	10
Figure 12:D Flip Flop simulation	10
Figure 13:2x1 MUX Block Diagram	11
Figure 14:2x1 MUX Verilog Code.	11
Figure 15:2x1 MUX simulation.....	12
Figure 16:Memory System Block Diagram.....	13
Figure 17:Comparator Block Diagram	14
Figure 18:Comparator Verilog Code.....	14
Figure 19:Comparator simulation.	15
Figure 20:The Security System Final Block Diagram.	16
Figure 21:FPGA board.....	17
Figure 21:FPGA board.....	18

III. List of Tables

Table 1:4x2 Priority Encoder truth table.....	4
---	---

1.Theory

we will talk about the main components of our experiment:

1.1 4x2 Priority Encoder:

The priority encoder is a combinational logic circuit that contains 2^n input lines and n output lines and represents the highest priority input among all the input lines. When multiple input lines are active high at the same time, then the input that has the highest priority is considered first to generate the output[1].

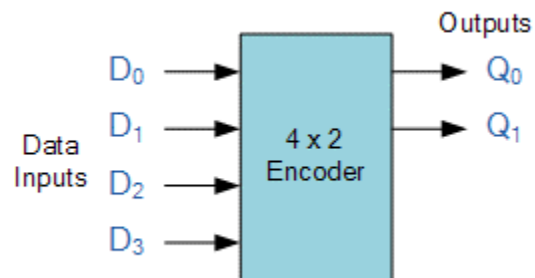


Figure 1:block diagram of 4x2 Priority Encoder

[2]

Table 1:4x2 Priority Encoder truth table.

D3	D2	D1	D0	Q1	Q0
1	X	X	X	1	1
0	1	x	X	1	0
0	0	1	X	0	1
0	0	0	X	0	0

1.2 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 4x2 priority encoder.

3 we put the output, the same thing in all gates in all ICs nether the inverter one because its take one input and one output[3].

1.3 Segment display driver:

A Flip-Flop is a better memory element for synchronous circuits, it solves the problem of latches in synchronous sequential circuits, A latch is sensitive to the level of the clock, but flip-flop is sensitive to the edge of the clock, it called an edge-triggered memory element, because it changes it output value at the edge of the clock[4]

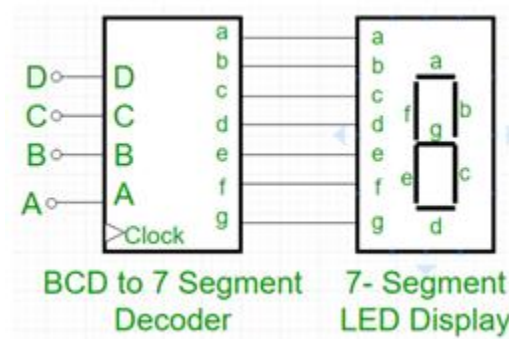


Figure 2:segment decoder.

1.4 Memory System:

The purpose of such 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 2x1 MUXs as shown in Figure 3. 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[3].

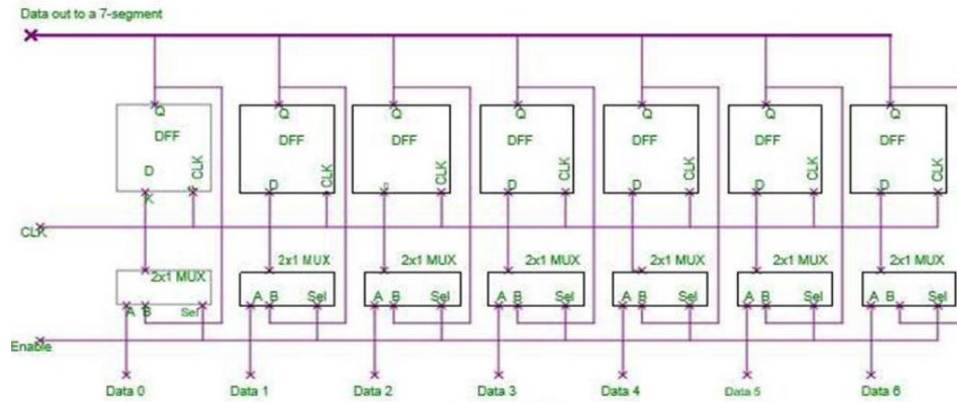


Figure 3:Memory System.

1.5 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 the 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[3].

1.6 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[3].

2.Procedure & Discussion

2.1 Design a 4 x 2 priority encoder:

In this part the 4 x 2 priority encoder module has been designed by writing and simulating the Verilog code shown in figure 5.

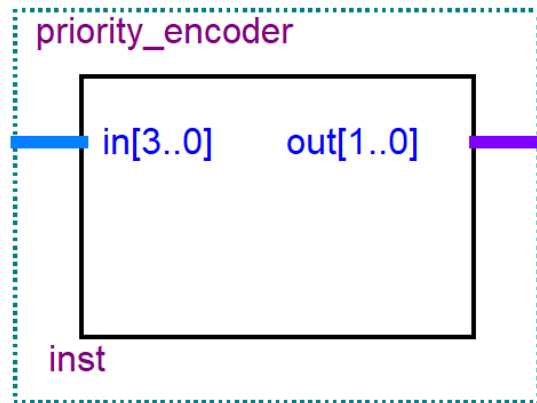


Figure 4:4 x 2 priority Block Diagram

```
1  module priority_encoder(out, in);
2      input [3:0] in;
3      output reg [1:0] out;
4      always @(in)
5      begin
6          casex(in)
7              4'b0001: out = 2'b00;
8              4'b001x: out = 2'b01;
9              4'b01xx: out = 2'b10;
10             4'b1xxx: out = 2'b11;
11             default: out = 2'b00;
12         endcase
13     end
14 endmodule
15
```

Figure 5:Priority Encoder Verilog Code.

First defines a 4-bit input 'in', and 2-bit output 'out', then whenever 'in' changes the case block executes as follows If 'in' is 0001, 'out' will be 00, If 'in' is 001x, 'out' will be 01, If 'in' is 01xx, 'out' will be 10, If 'in' is 1xxx, 'out' will be 11, and default case if none previous match.

Below we find figure for the simulation.

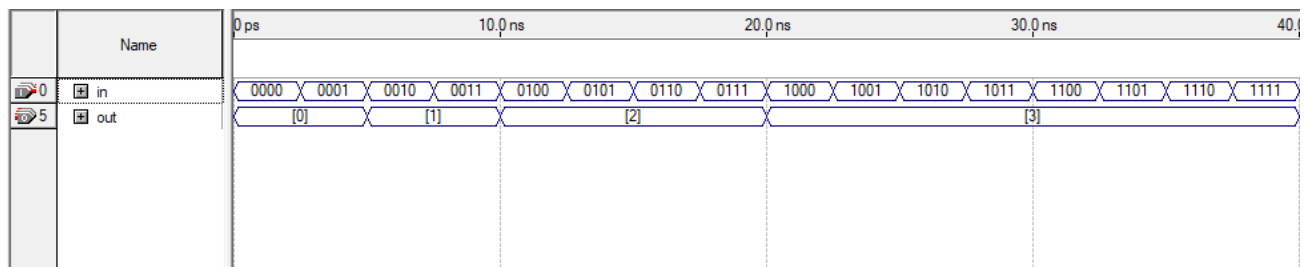


Figure 6:Simulation for 4 x 2 priority encoder

After testing the 4 x 2 priority, and collect this simulation data we verify that if multiple input lines are active, the output code will correspond to the input line with the highest priority and same as 4 x 2 priority truth table.

2.2 7-segment display driver:

In this part the 7-segment display driver module has been designed by writing and simulating the Verilog code shown in figure 8.

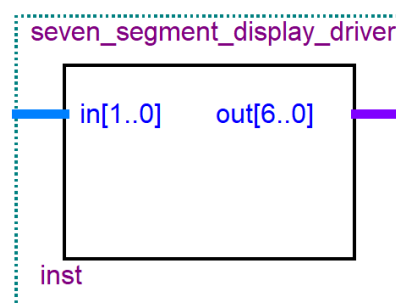


Figure 7:7-segment display driver block diagram

```

1  module seven_segment_display_driver(out, in)
2
3      input [1:0] in;
4      output reg [6:0] out;
5
6      always @(in)
7      begin
8          case(in)
9              0:out = 7'b0000001;
10             1:out = 7'b1001111;
11             2:out = 7'b0010010;
12             3:out = 7'b0000110;
13         endcase
14     end
15 endmodule
16

```

Figure 8:7-Segment Display Driver Verilog Code.

First defines a 2-bit input 'in', and 7-bit output 'out', then whenever 'in' changes the case block executes as follows If 'in' is 0, 'out' will be 0000001, If 'in' is 1, 'out' will be 1001111, If 'in' is 2, 'out' will be 0010010, and if 'in' is 3, 'out' will be 0000110.

Below we find figure for the simulation.

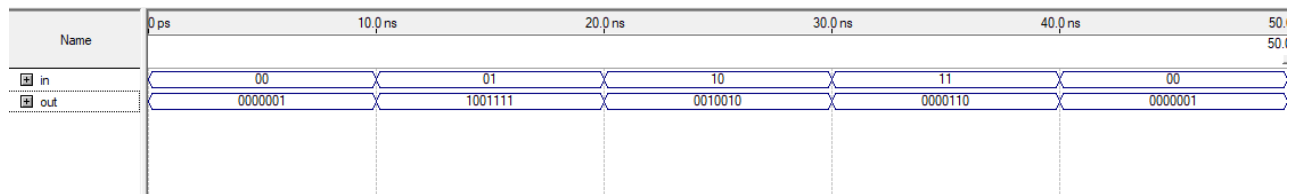


Figure 9:7-segment display driver.

After testing the 7-segment display driver, and collect this simulation data we verify that if any correct input enters, the output code will be the correct code for 7-segment LED display.

2.3 D- Flip Flop:

In this part the D- Flip Flop module has been designed by writing and simulating the Verilog code shown in figure 11.

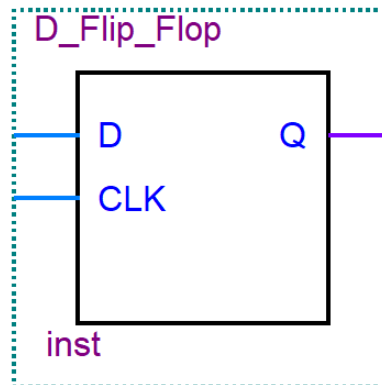


Figure 10:D Flip Flop Block Diagram.

```

1  module D_Flip_Flop (Q, D, CLK);
2      input D, CLK;
3      output Q;
4      reg Q;
5      always @(posedge CLK)
6          Q = D;
7  endmodule
8

```

Figure 11:D Flip Flop Verilog Code.

First defines 'D' as data input, 'CLK' as clock input, and 'Q' as the output, then declares 'Q' as a register to hold the state, and when the clock on the rising edge the always block executes to assigns the value of 'D' to 'Q'.

Below we find figure for the simulation.

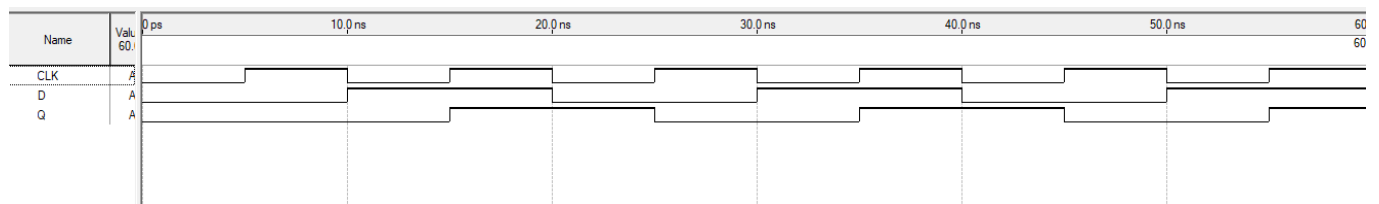


Figure 12:D Flip Flop simulation

After testing the D Flip Flop, and collect this simulation data we verify that the D Flip Flop captures and holds the input value at the moment of a clock signal's rising edge.

2.4 2x1 MUX:

In this part the 2x1 MUX module has been designed by writing and simulating the Verilog code shown in figure 14.

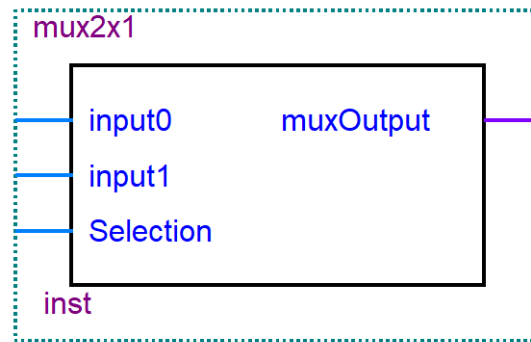


Figure 13:2x1 MUX Block Diagram

```
1 module mux2x1 (input0,input1,Selection,muxOutput);
2   input input0,input1,Selection;
3   output muxOutput;
4   reg muxOutput;
5   always @ (input0 or input1 or Selection)
6       if (Selection == 0)
7           muxOutput = input0;
8       else
9           muxOutput = input1;
10  endmodule
```

Figure 14:2x1 MUX Verilog Code.

First defines 'input0' and 'input1' as inputs, 'Selection' as the select line, 'muxOutput' as the output, and declares 'muxOutput' as a register to hold the state, then whenever any of the inputs change the always block executes as follows If 'Selection' is 0, 'muxOutput' will be 'input0' else if 'Selection' is not 0, 'muxOutput' will be 'input1'.

Below we find figure for the simulation.

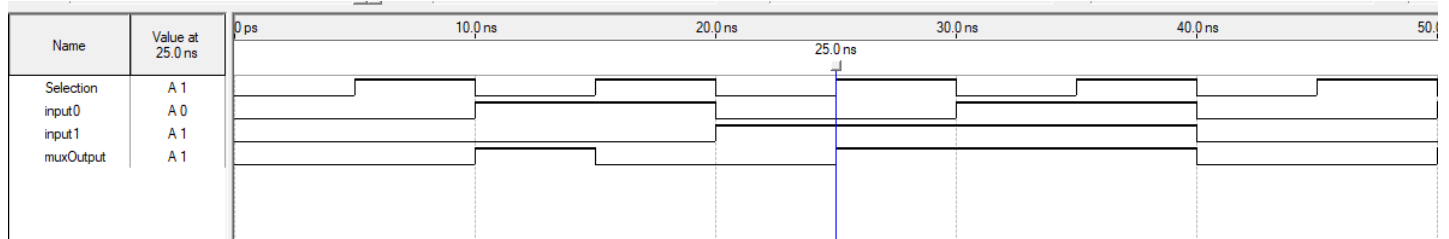


Figure 15:2x1 MUX simulation

After testing the 2x1 MUX, and collect this simulation data we verify that the MUX allows one of two inputs to be passed to the output based on a selection signal.

2.4 Memory System:

In this part the Memory System Block has been designed using seven of D- flip-flops and 2x1 MUXs as shown in Figure 12.

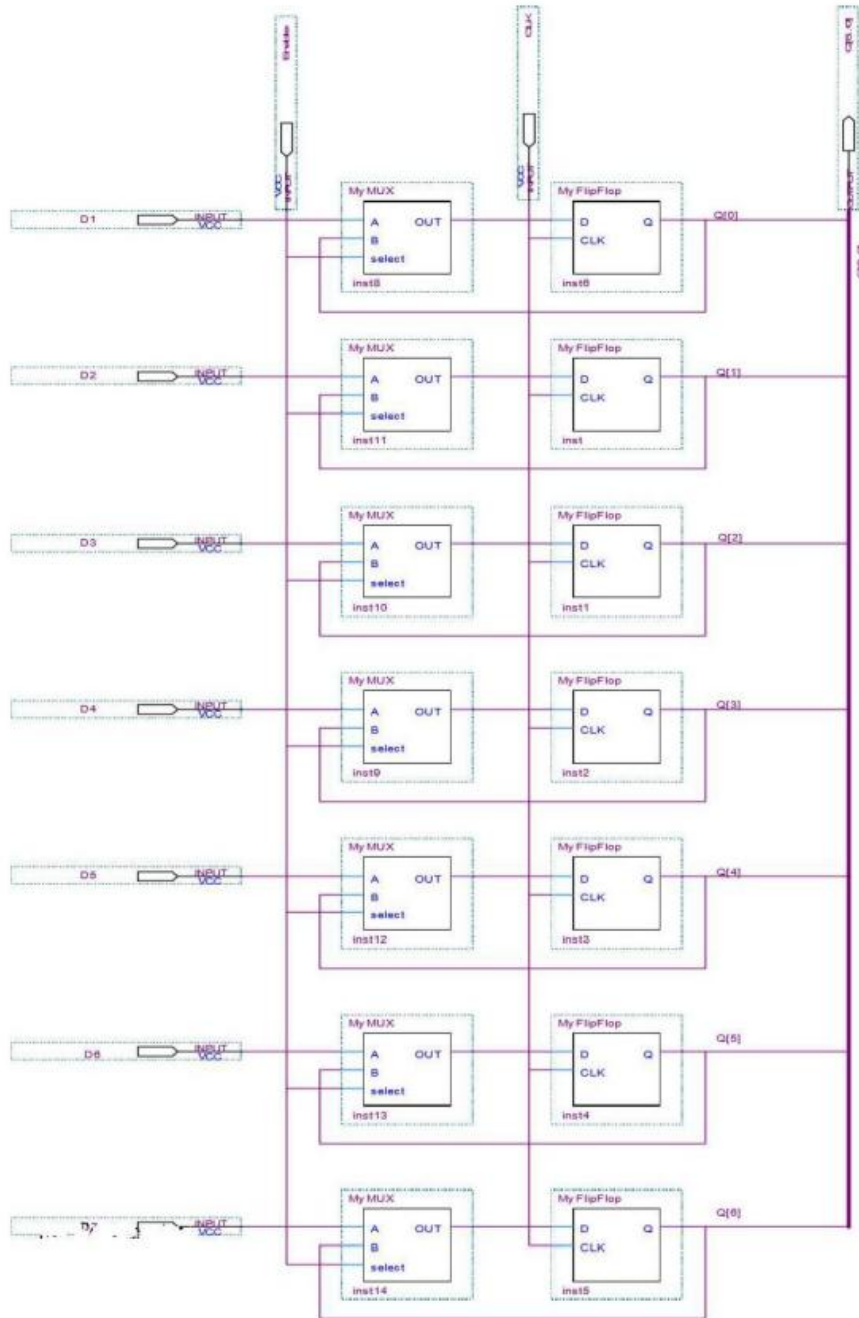


Figure 16:Memory System Block Diagram

2.5 Comparator:

In this part the Comparator module has been designed by writing and simulating the Verilog code shown in figure 18.

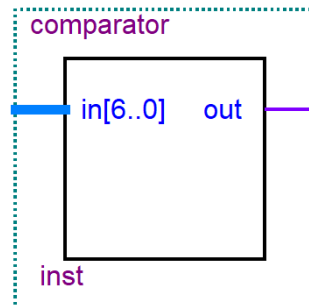


Figure 17:Comparator Block Diagram

```
1  module comparator(out, in);
2      input [6:0] in;
3      output reg out;
4
5      always @ (in)
6      begin
7          if (in == 7'b0100100)
8              out = 1'b1;
9          else
10             out = 1'b0;
11     end
12 endmodule
```

Figure 18:Comparator Verilog Code.

First defines a 7-bit input 'in', and a 1-bit output 'out', then whenever 'in' changes the always block executes as follows Checks if 'in' is equal to the binary number 0100100 If true, sets 'out' to 1, else if false, sets 'out' to 0.

Below we find figure for the simulation.

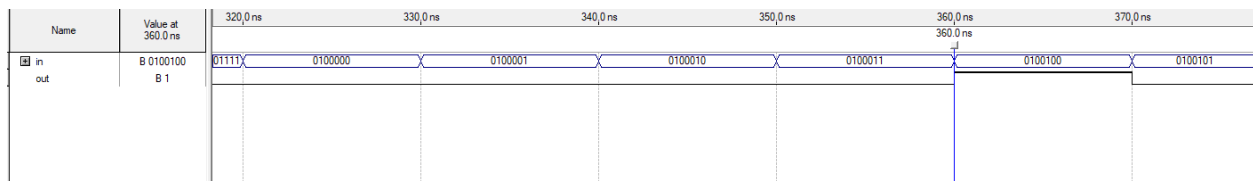


Figure 19:Comparator simulation.

In this part the final block of the security system has been designed using the components that built in the previous steps as shown in the figure below. Then Assign pins values to the security system design, and download the system on the FPGA board.

2.6 Security System:

In this part the final block of the security system has been designed using the components that built in the previous steps as shown in the figure below. Then Assign pins values to the security system design, and download the system on the FPGA board.

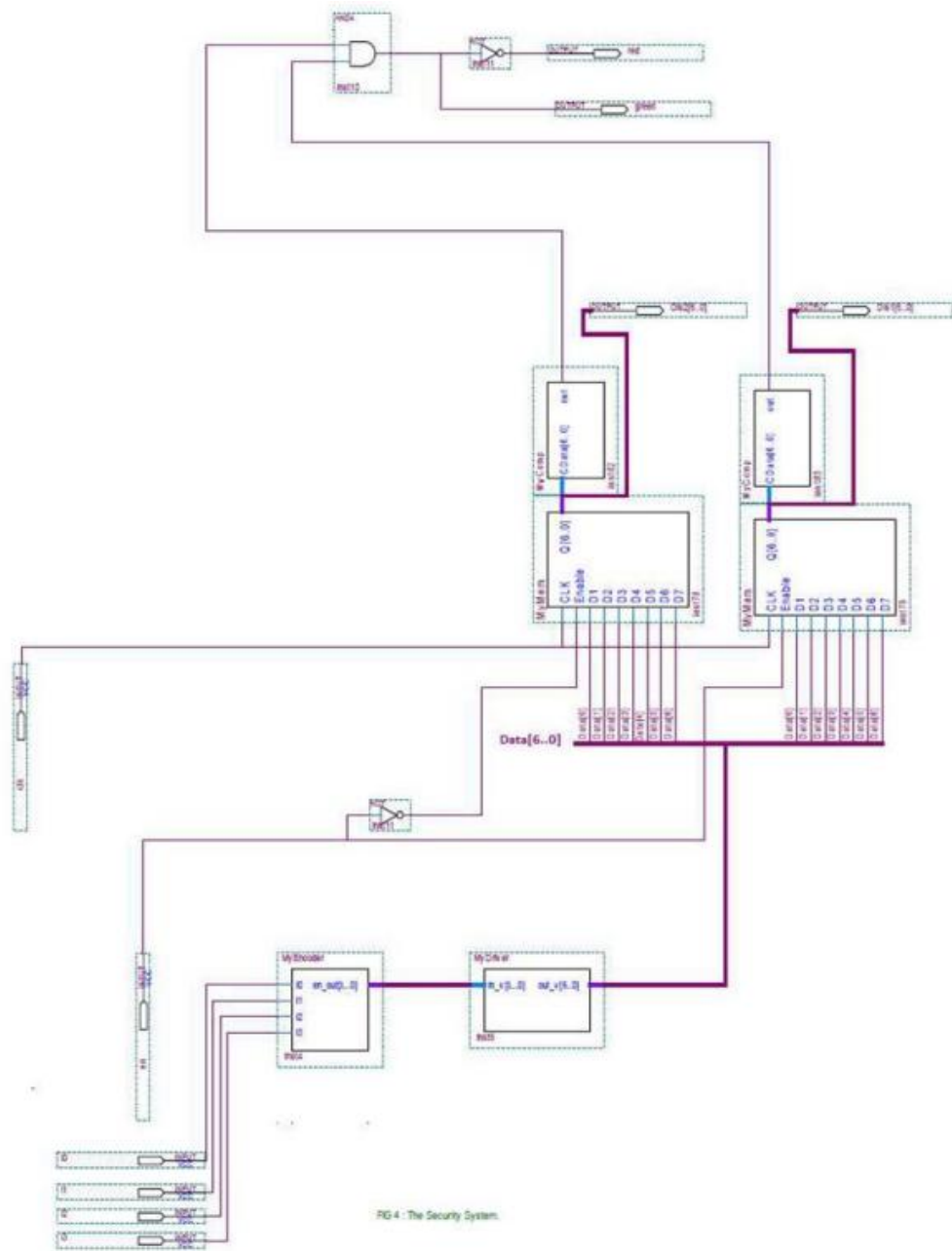


Figure 20: The Security System Final Block Diagram.

Here we can find a picture of the block in the lab.

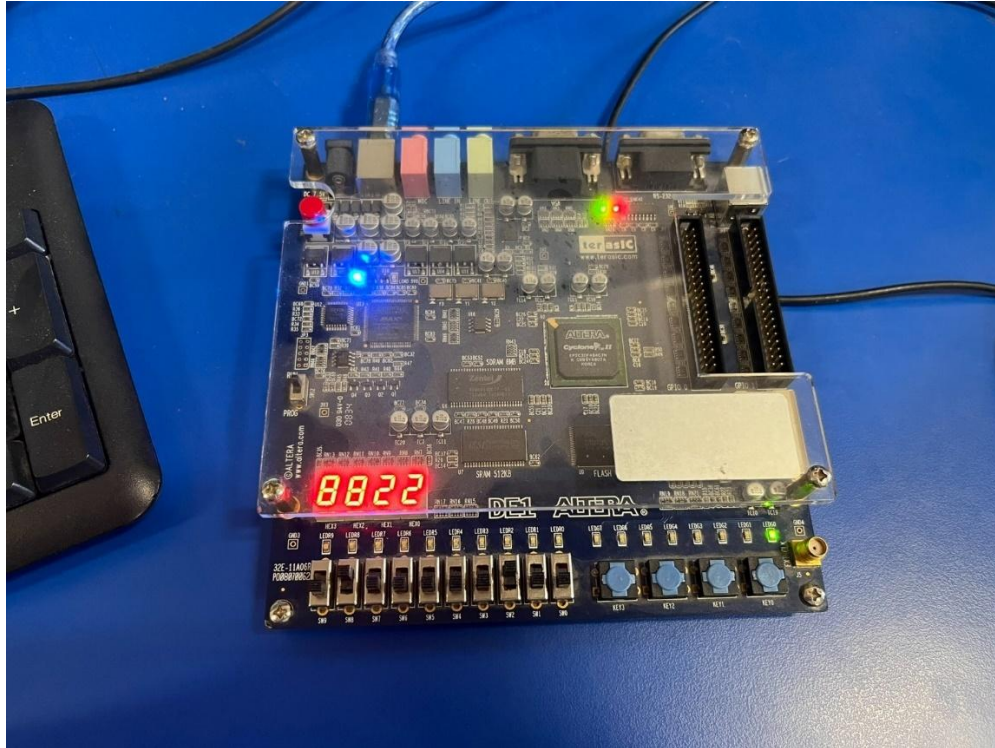


Figure 21:FPGA board

After download the system on the board which can be programmed after manufacturing to perform a wide variety of digital functions, and test it, we verify that if the user enters the correct password the green led turn on otherwise the red led still on, indicates that the security system works fine and correctly.

2.7 Discussion:

This experiment demonstrated the practical integration of digital components to build a simple FPGA-based security system. Each module—priority encoder, 7-segment display driver, memory system, and comparator—was designed and tested individually before final integration. The system successfully verified user input against a preset code, activating a green LED upon correct entry. Through simulation and hardware implementation, we deepened our understanding of combinational and sequential logic. Minor issues like pin misassignments were resolved during testing. Overall, the lab effectively connected theoretical concepts with real-world FPGA design.

3.Conclusion

In conclusion the experiment went smoothly with some complications at first but then it was fine, everything was giving the same answers as the theoretical datasheets. This experiment was simply

to understand how comparators, flip flops, Priority Encoders, and 7- segment displays work and how to implement each one. The implementation of each design was done through Quartus either by building a Verilog codes or Block diagrams.

4.References

- [1] : <https://www.elprocus.com/priority-encoder/> [Accessed 18 May 2024,17:28]
- [2] : https://www.electronics-tutorials.ws/combinational/comb_4.html [Accessed 18 May 2024,17:43]
- [3] : Manual for Digital Electronics and Computer Organization Lab.
- [4] : <https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html> [Accessed 13 April 2024,1:00]

5. Appendices

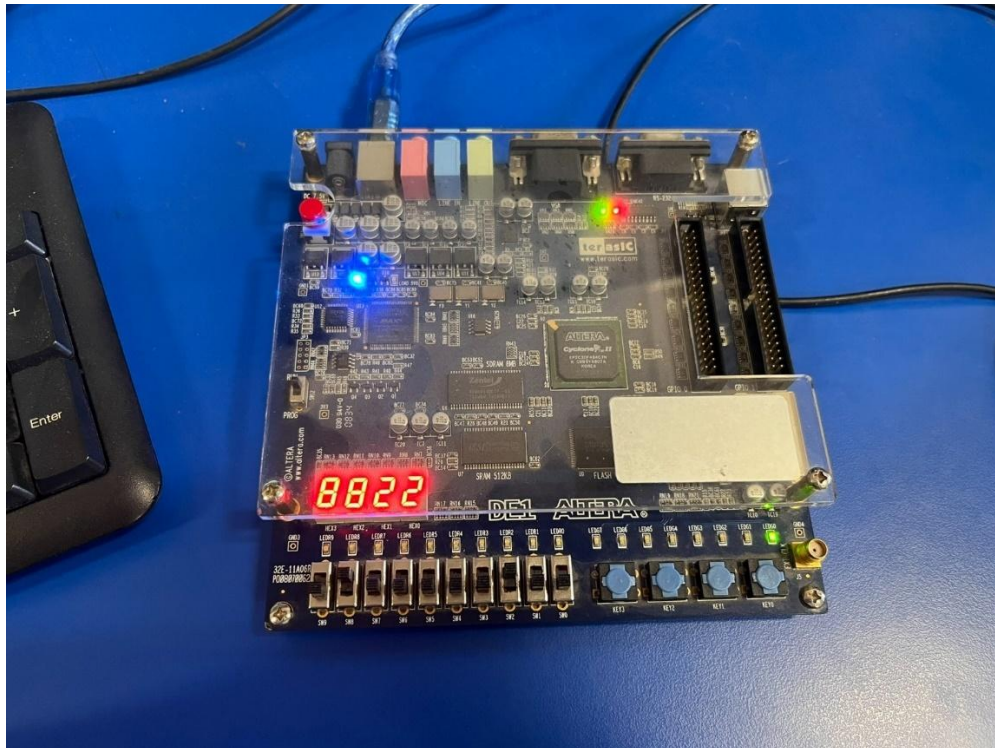


Figure 22:FPGA board