Faculty of Engineering and Technology Department of Electrical and Computer Engineering ENCS 2110

Digital Electronics and Computer Organization Lab

Experiment No. 9 - A Simple Security System Using FPGA

Prepared by:

Zainab Murad **NumberID : 1211020**

Partners:

Mohyeddin Eis **NumberID : 1230256**

Instructor: Ashraf Al-Rimawi

Teaching assistant: Eng. Raneem Alqaddi

Section:4

Date: 13/12/2024

Abstract

In this experiment, we designed and implemented a simple security system using Quartus software and an FPGA board. The system functions as a two-digit digital lock, allowing the user to input numbers ranging from 0 to 3 via a keypad. These inputs are displayed on two 7-segment displays and compared to predefined passwords. If the entered values match the passwords, a green LED is activated to grant access; otherwise, a red LED remains lit to deny access. The design integrates several components, including a priority encoder to convert inputs to binary codes, a 7-segment display driver to format the outputs for display, a memory system to retain the displayed values, comparators to match the inputs with reference values, and an AND gate to validate the combination. This experiment provided valuable experience in FPGA programming and the development of a cohesive digital system.

Table of Contents

Table of Figures

1. Theory

In this experiment, we will design a simple security system using the Altera Quartus software and implement it on an FPGA board. The system functions as a 2-digit digital lock, where the user enters a number consisting of two digits, with each digit ranging from 0 to 3. The input is provided through a keypad (using the 91 switch keys integrated into the FPGA), and each digit is displayed on a 7-segment display. If the total number entered matches a predefined value (XX), a green LED lights up, granting access. If the number does not match, a red LED stays on, blocking access.

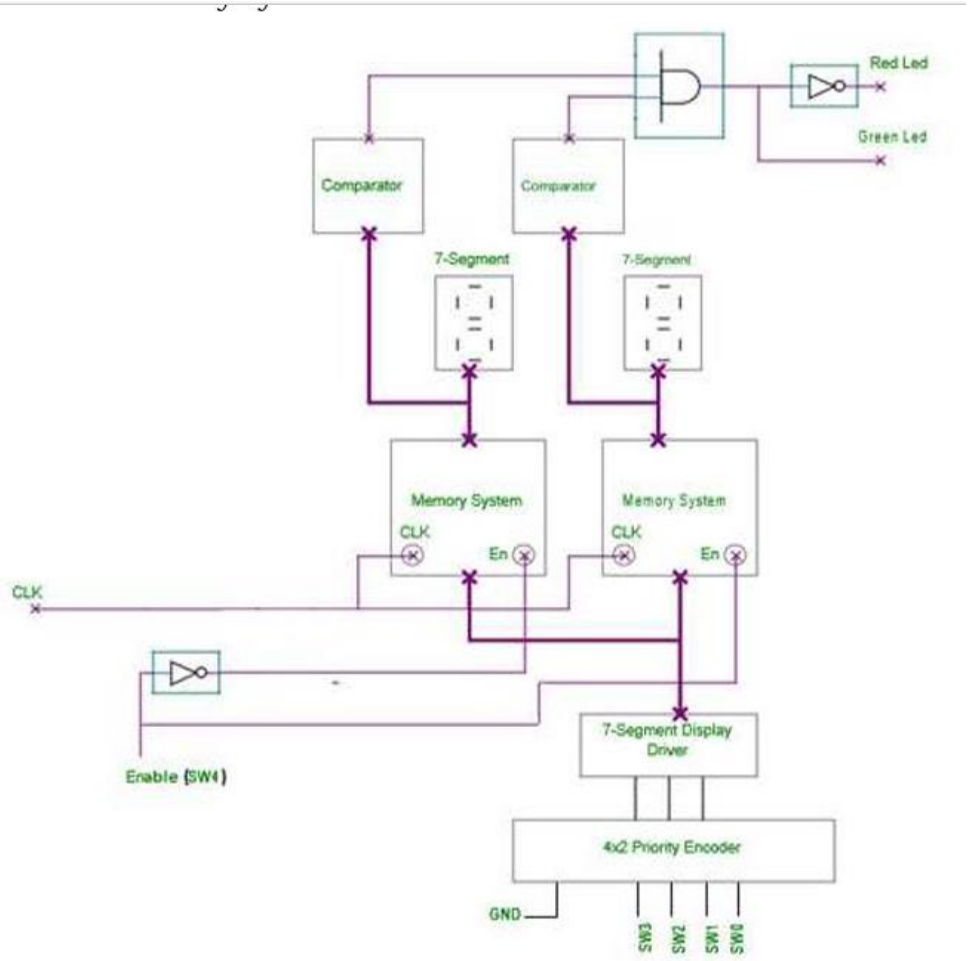
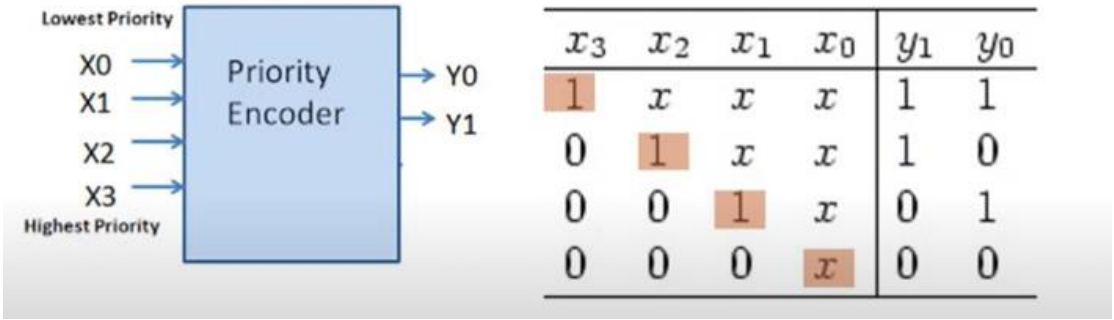


Figure 1.1: The Security System Architecture. (Lab Manual).

Figure 1.2: High level view of the system. (Lab Manual).

1.1 4x2 Priority Encoder

A normal digital encoder is a combinational circuit that converts 2^n input lines into n output lines, generating the binary code for the active input. However, it has a limitation: it only works correctly when one input is active. If multiple inputs are active, the encoder generates an incorrect code. To solve this, a priority encoder is used, which prioritizes the active input and outputs the code corresponding to the highest priority input. For example, if SW1 is high and SW2 and SW3 are low, the priority encoder will output b'01', representing the value to be displayed on a 7-segment display.



x_3	x_2	x_1	x_0	y_1	y_0
1	x	x	x	1	1
0	1	x	x	1	0
0	0	1	x	0	1
0	0	0	x	0	0

Figure 1.3: 4x2 Priority Encoder. (Lab Manual).

1.2 Enable Port

The enable port is used to let the user choose which memory system to activate, thereby selecting the corresponding 7-segment display. For instance, when SW4 is high, the En pin of the first memory system is activated, allowing it to read input from the 4x2 priority encoder. It's important to note that the decoder's enable pin must be active low when switching between the selection lines.

1.3 segment display driver

The output of the priority encoder is converted into the proper input for the 7-segment displays. This output is first stored in a memory unit before being transferred to the 7-segment display, depending on the memory system that is enabled by the 2x4 decoder. Figure 9.4 shows a 4x7 7-segment decoder, but in this experiment, a 2x7 7-segment decoder will be used.

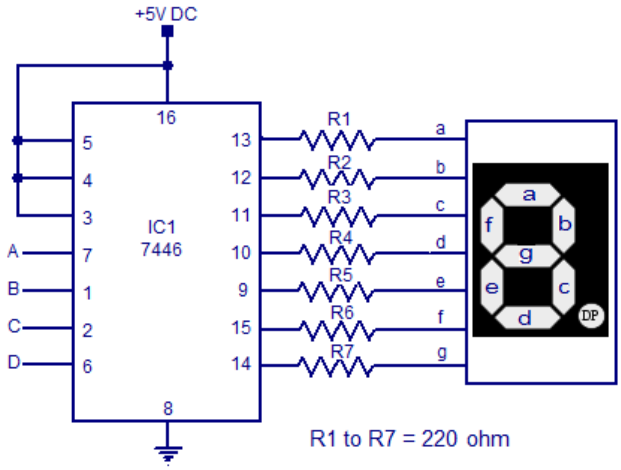


Figure 1.4 segment display driver system (circuitstoday.com)

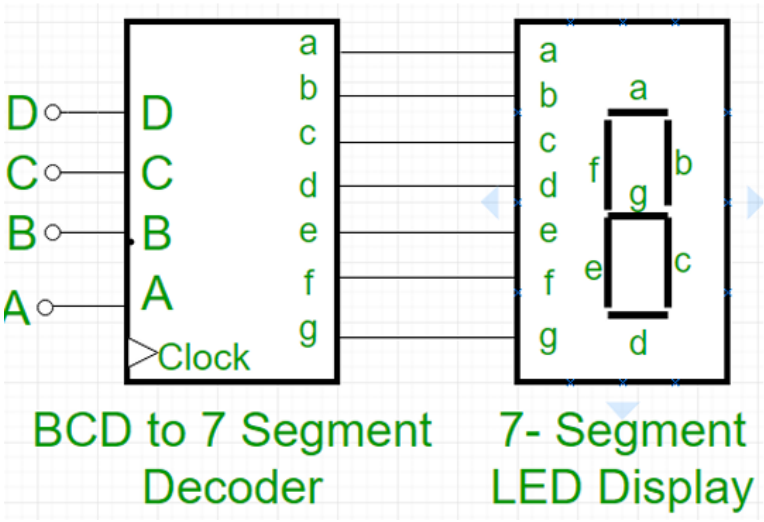


Figure 1.5 segment display driver. (Lab Manual).

1.4. Memory System

This system ensures that the value displayed on a specific 7-segment remains unchanged even when the user switches to another 7-segment. Each memory block includes seven D flip-flops and 2x1 multiplexers, as shown in Figure 9.5. When the enable pin is set to 0, the output of each flip-flop loops back as its input with every clock cycle. However, when the enable pin is set to 1, the data from the 7-segment driver is stored in the flip-flops. The output of these flip-flops is then sent via a data bus to the corresponding 7-segment display. Notably, each 7-segment display requires a dedicated memory block.

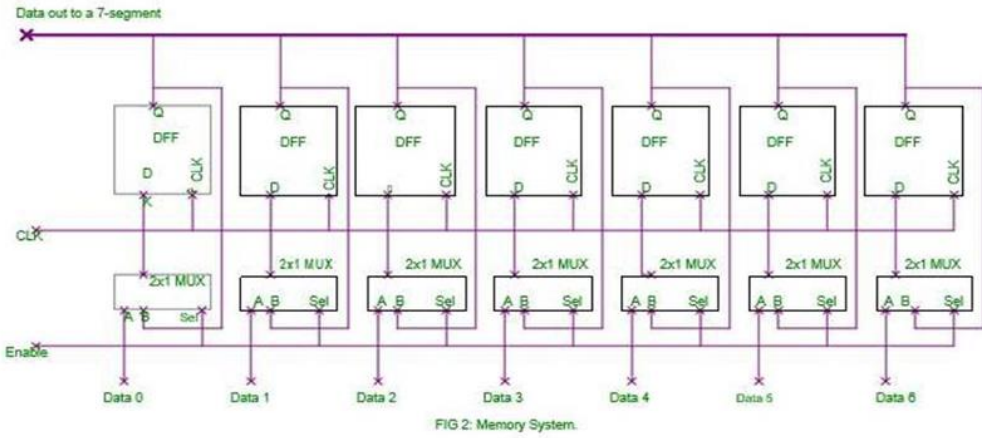


Figure 1.6 Memory System. (Lab Manual).

1.5 Comparator

Each 7-segment display is connected to a comparator, which compares the display's input value with a predefined reference value. If the input matches the reference value, the comparator outputs a logic "1"; otherwise, it outputs a logic "0." For instance, if a comparator has a reference value of 5, it will output "1" only when the input equals 7'b0100100 (the binary representation of 5 for a 7-segment display). The role of the comparator is to determine whether to lock or unlock the security system based on these comparisons.

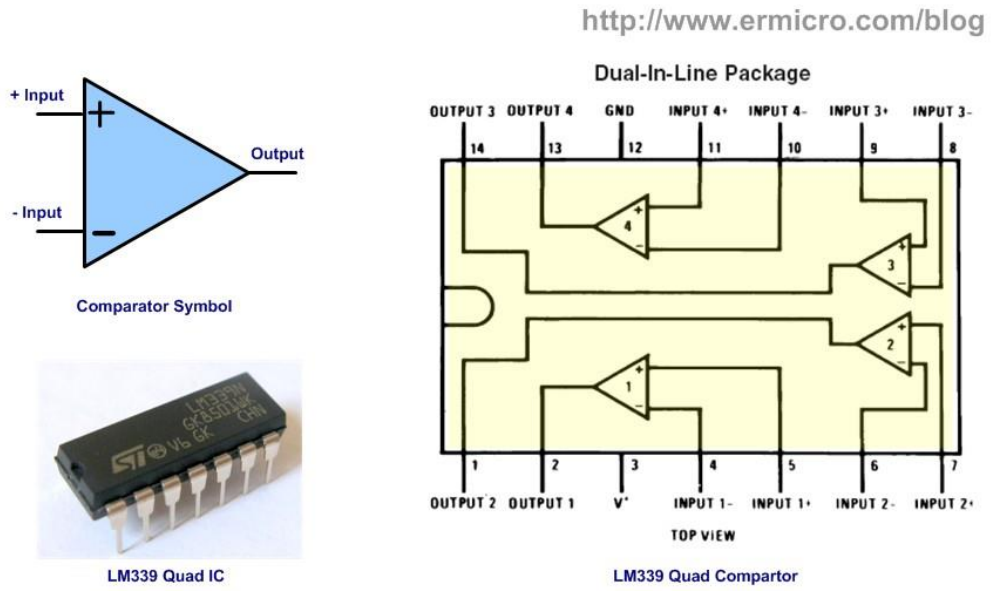
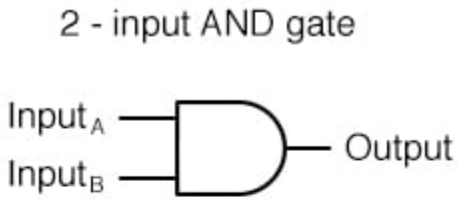


Figure 1.7 Memory System. (ermicro.com)

1.6 2-input AND gate

An AND gate ensures that both 7-segment displays show the correct combination. Specifically, if the outputs of both comparators are "1," the AND gate will output a "1," turning on the green light to indicate access is granted. If this condition is not met, the AND gate output remains "0," keeping the red light on to indicate access is denied.



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Figure 1.8 2-input AND gate. (allaboutcircuits.com)

2. Procedure and Discussion:

2.1 Design a 4*2 priority Encoder

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

```
//4 x 2 Priority encoder
module priority_encoder(out, in);

  input [3:0] in;
  output reg [1:0] out;
  always @ (in)

begin

  casex(in)
        4'b0001:out = 2'b00;
        4'b001x:out = 2'b01;
        4'b01xx:out = 2'b10;
        4'b1xxx:out = 2'b11;
        default:out = 2'b00;
    endcase

end

endmodule
```

Figure 2. 1 : 2x4 Priority Encoder – Code(Lab Manual)

The priority encoder waveform is shown in Figure 2.2 below:

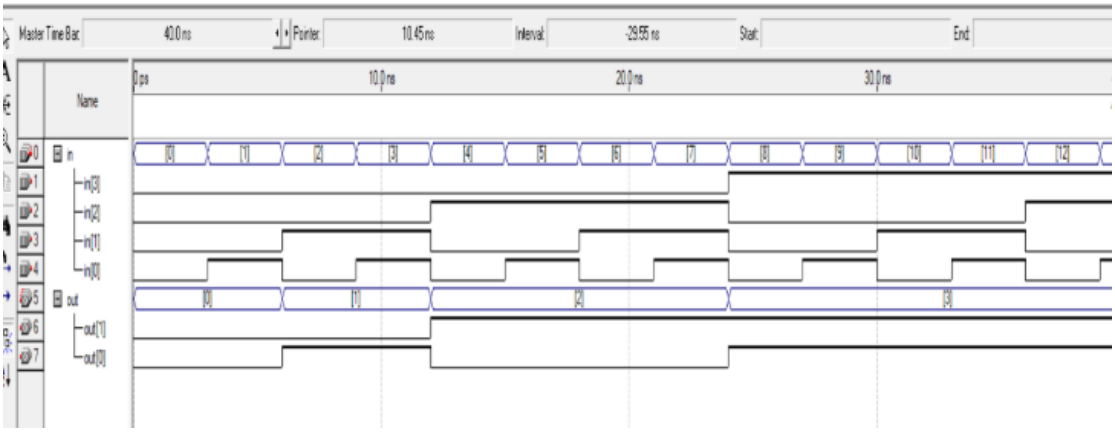


Figure 2. 2 : 2x4 Priority Encoder – Waveform

Discussion:

The results confirm that the 2×4 priority encoder functions correctly. It operates with four inputs and two outputs, determining the highest priority bit that is set to HIGH. The encoding process begins by checking the inputs sequentially from the most significant bit (MSB) to the least significant bit (LSB). The first input found to be HIGH is identified as the highest priority bit and is subsequently encoded.

2.2 Design a 7-segment display driver

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

```
//this will convert binary input
// to 7-segment input
module sevenSegmentDriver(in_v , out_v ) ;
input [1:0] in_v ;
output [6:0] out_v ;
wire[1:0] in_v ;
reg [6:0] out_v ;
always @ (in_v)
begin
    case(in_v)
        0 : out_v = 7'b1000000;
        1 : out_v = 7'b1111001;
        2 : out_v = 7'b0100100;
        3 : out_v = 7'b0110000;
    endcase
end
endmodule
```

Figure 2. 3 : 7-Segment Display Driver (Lab Manual)

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

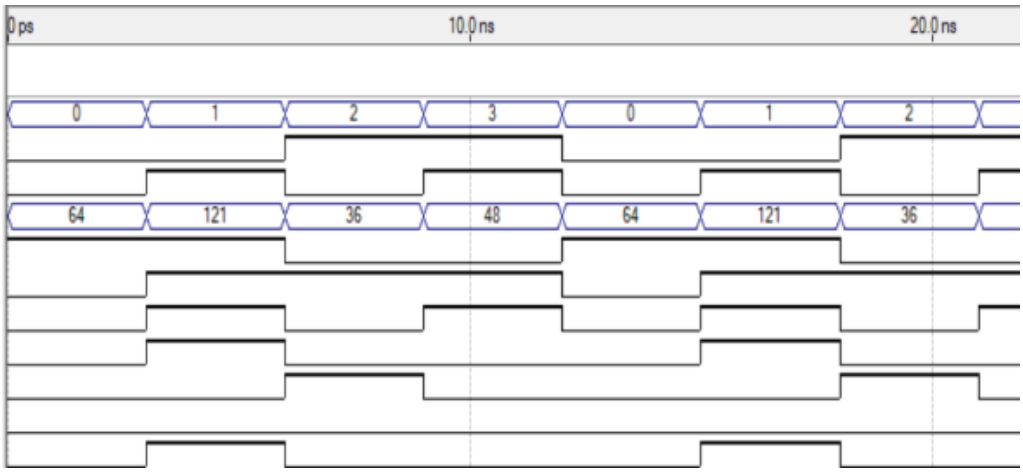


Figure 2. 4 : 7-Segment Display Driver – Waveform

Discussion

In this section, we create a Quartus code for a 2-bit to 7-segment decoder. With 2-bit inputs, the possible values range from 0 to 3 (`2'b00` to `2'b11`), resulting in four distinct cases (digits 0, 1, 2, and 3). Each digit corresponds to a unique code displayed on the 7-segment. The input value is analyzed, and based on it, the appropriate bits for the 7-segment display are set accordingly.

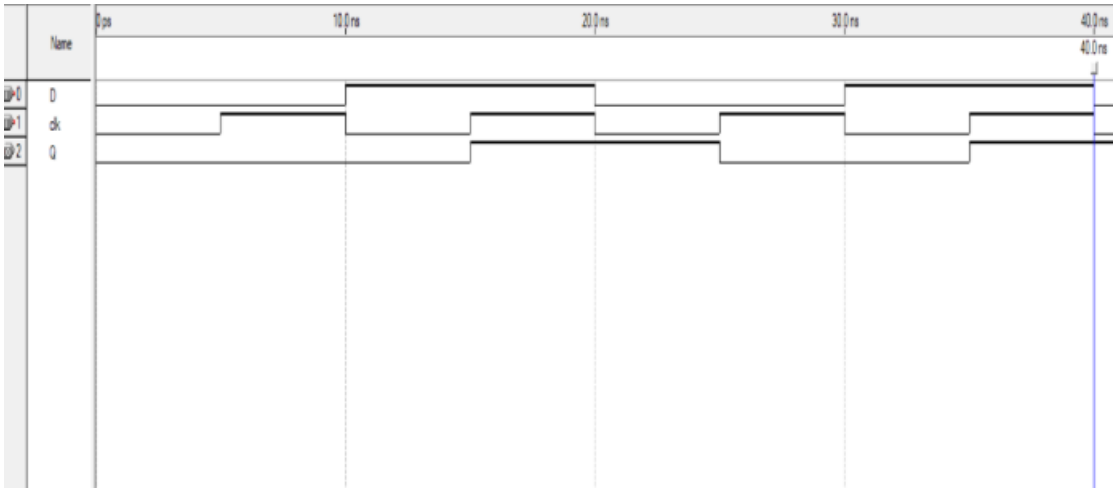
2.3 Design a D – Flip Flop

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

```
1
2   module D_flip_flop (output reg Q,input D,clk);
3
4   always @(posedge clk )
5
6   Q<=D;
7
8   endmodule
9
```

Figure 2. 5 : D - Flip Flop – Code

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



Discussion:

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

2.4 Design a 2*1 MUX

```
1
2   module MUX_2TO1 (output reg m,input z,n,l);
3
4   always @(posedge clk )
5
6   always @(*)
7   if(l==0)
8   m=n;
9   else
0   m=z;
1   endmodule
2
```

Figure 2. 7 : 2x1 MUX – Code

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

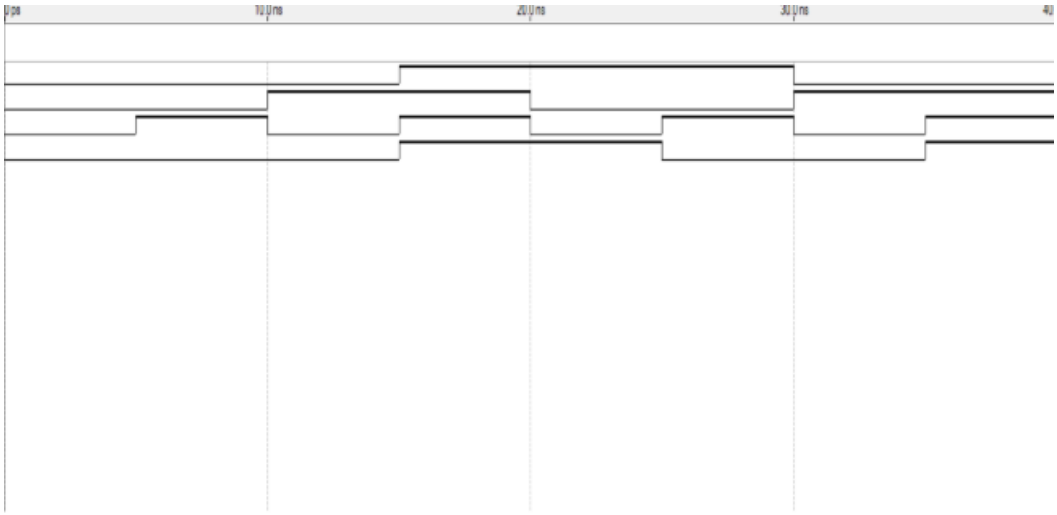


Figure 2. 8 : 2x1 MUX – Waveform

Discussion:

In this section, we use Quartus software to implement a 2×1 multiplexer with two inputs, A and B, and a selection input, Sel. The multiplexer outputs either A or B based on the value of the selection line. When Sel is 0, the output corresponds to the least significant input, B. Conversely, when Sel is 1, the output corresponds to the most significant input, A.

2.5 Design a Memory System

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

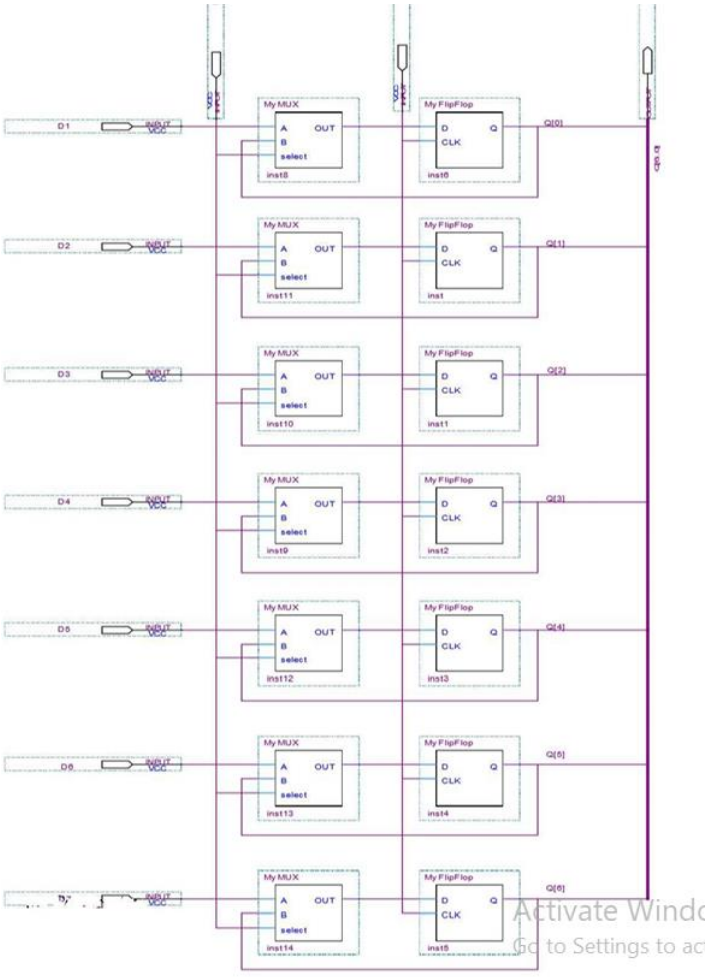


Figure 2. 9 : Memory System Block Diagram (Lab Manual).

Figure 2. 10 : Memory System – Waveform

Discussion:

the block diagram of a 2×1 multiplexer and a D flip-flop is utilized to construct a memory system. The stored memory value updates only when the enable signal is HIGH and a positive clock edge occurs. Otherwise, the memory retains its current value, as illustrated in the memory system waveform shown in Figure 2.10.

2.6 Design a Comparator

We designed a comparator using QUARTUS software as following:

```
module MyComp(CData,out);
input [6:0] CData;
wire [6:0] CData;
output out;
reg out;
always @ (CData)
begin
    //compare value in register with 2
    if(CData == 7'b0100100)
        out <=1'b1;
    else
        out <=1'b0;
    end
endmodule
```

Figure 2. 11 : Comparator – Code(Lab Manual).

The Comparator waveform is shown in Figure 2.10 below:

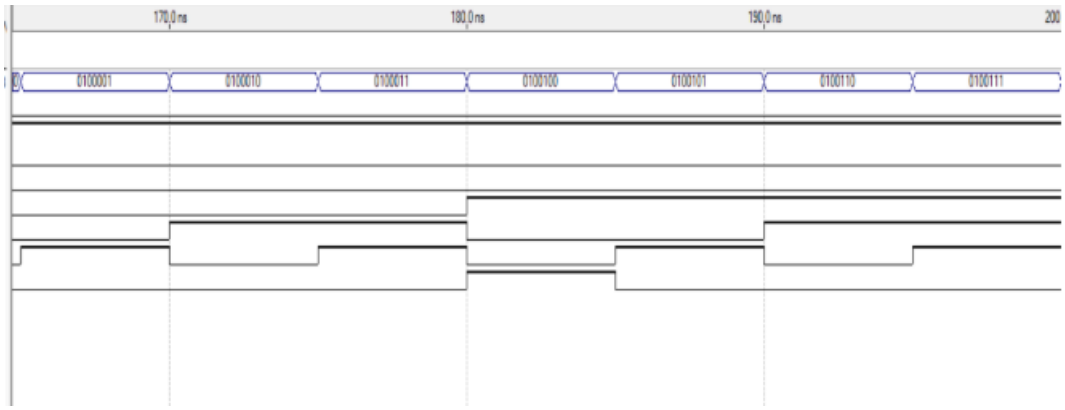


Figure 2. 12 : Comparator – Waveform

Discussion:

In this section, we design a comparator to compare the value stored in the register (memory system), representing the user's input, with the predefined correct value for the security system. In this implementation, the reference value is set to 2, meaning the correct password for the system is assumed to be 2.2

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:

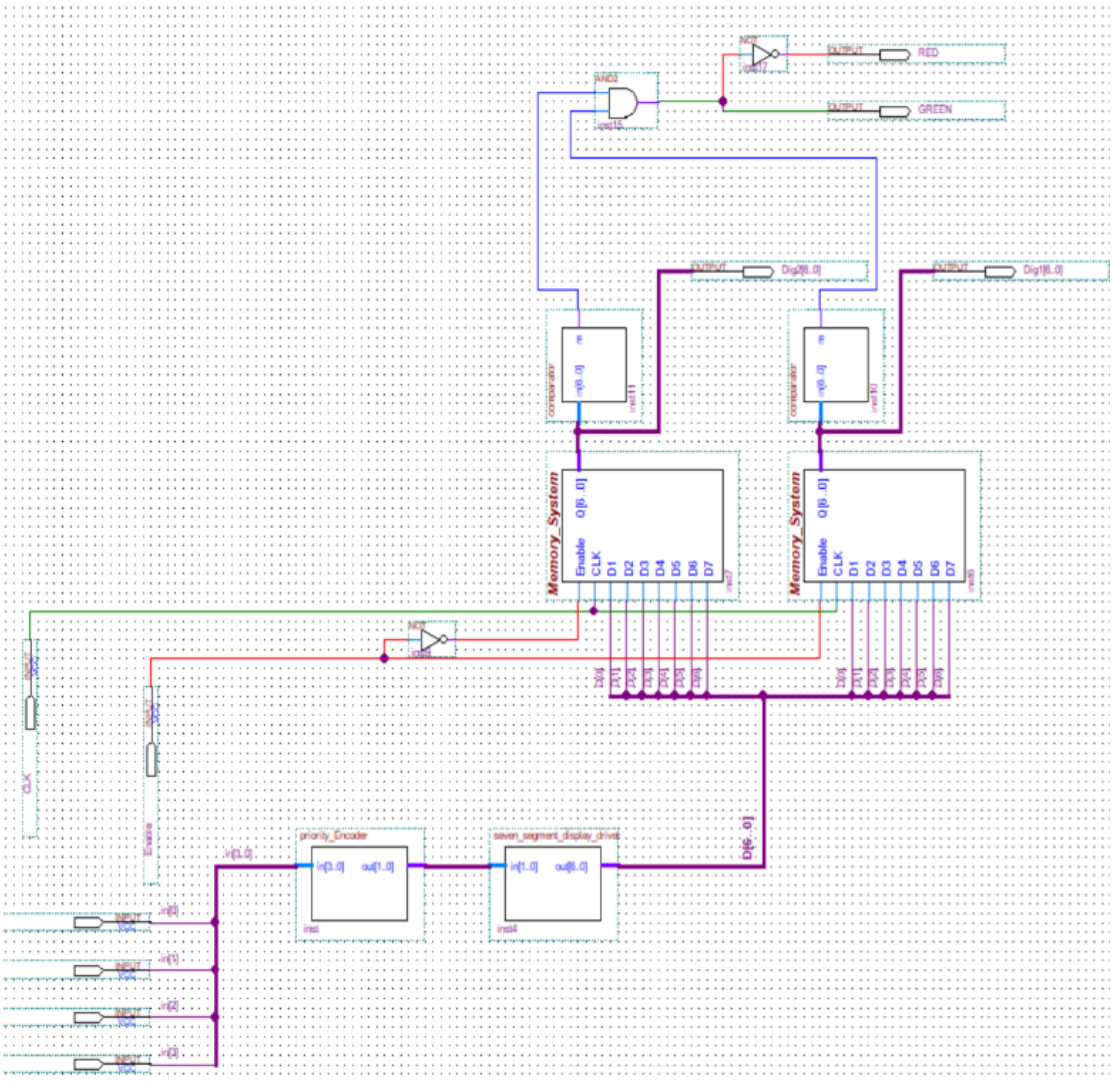


Figure 2. 13 : The Security System Final Block Diagram

The Security System waveform is shown in Figure 2.14 below :

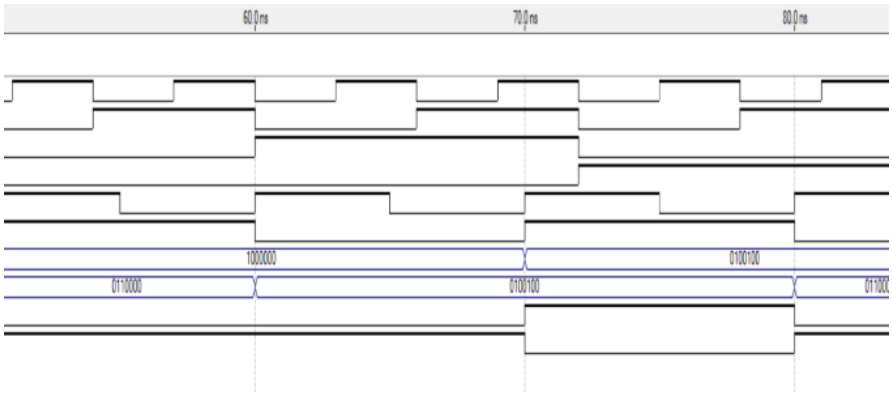


Figure 2. 14 : The Security System Final Block Diagram – Waveform

Discussion :

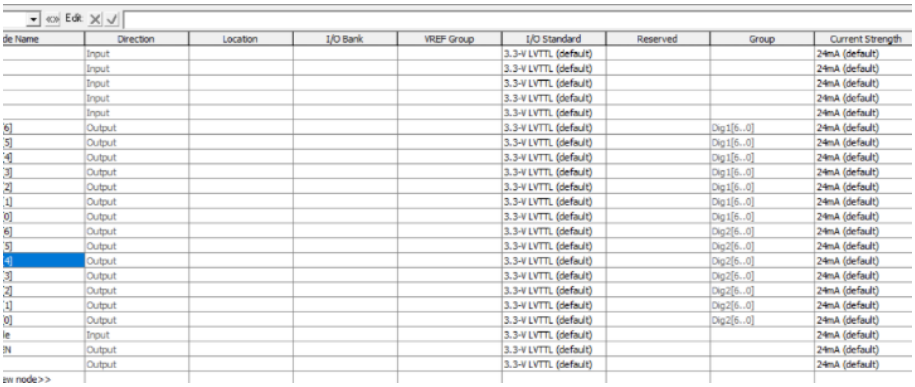
In this section, we constructed the final block diagram of the security system by integrating the components designed in previous sections, as shown in Figure 2.13. By tracing the waveform in Figure 2.14, we can observe how the system operates. When the two digits entered are (0100100), representing the value 2 in decimal, the green LED will be HIGH (ON), indicating that the password for the security system is 22. If the input does not match, 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 :



de Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Group	Current Strength
	Input				3.3-V LVTTL (default)			24mA (default)
	Input				3.3-V LVTTL (default)			24mA (default)
	Input				3.3-V LVTTL (default)			24mA (default)
	Input				3.3-V LVTTL (default)			24mA (default)
	Input				3.3-V LVTTL (default)			24mA (default)
6]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
5]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
4]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
3]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
2]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
1]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
0]	Output				3.3-V LVTTL (default)		Dig1[6..0]	24mA (default)
6]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
5]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
4]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
3]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
2]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
1]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
0]	Output				3.3-V LVTTL (default)		Dig2[6..0]	24mA (default)
le	Input				3.3-V LVTTL (default)			24mA (default)
N	Output				3.3-V LVTTL (default)			24mA (default)
	Output				3.3-V LVTTL (default)			24mA (default)
ew node>>								

Figure 2. 15 : Assign Pins Values to FPGA Board

In this section, we select the switches and LEDs to be used on the FPGA board by assigning the pin numbers (for switches, LEDs, and the 7-segment display) to the "Location" column, as shown in Figure 2.15.

2) Download the System to the FPGA Board:

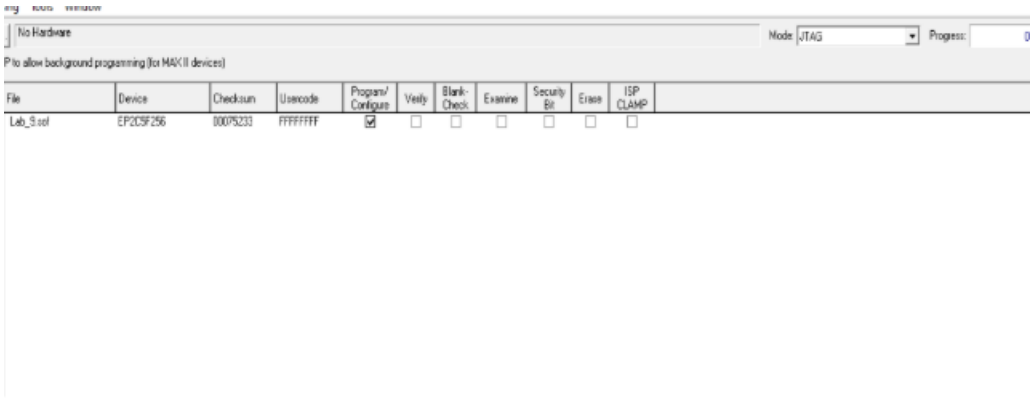


Figure 2. 16 : Download the System to FPGA

Discussion:

Finally, after implementing 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 and 7-segment displays on the FPGA board. Then, we downloaded the program (the system) to the FPGA and used it as the security system.

3. Conclusion

Upon completing this experiment, all the objectives have been achieved. I am now capable of implementing and simulating any program using the QUARTUS software. Additionally, I can assign and download any program to the FPGA board, utilizing it as needed. In this experiment, I first implemented the basic components, including the priority encoder, multiplexer, 7-segment display driver (decoder), D flip-flop, and comparator. Then, I designed the memory system using the multiplexer and D flip-flop components created earlier. Finally, I integrated all these components to construct the Final Block Diagram of the Security System, as described earlier.

After implementing the components and tracing the results, I observed that the outcomes aligned with the theoretical expectations. This confirms that the implementation was correct, with no issues or errors in any of the components or in the entire system (Final Block Diagram).

4. References

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

[2]: [online image] :[Accessed On 13th December 2024]

https://images.app.goo.gl/4vvaACnSd5qmyi566

[3]: [online image] :[Accessed On 13th December 2024]

https://images.app.goo.gl/YquSaZ5CBv8NSRXm9

[4]: [online image] :[Accessed On 13th December 2024]

https://images.app.goo.gl/Qb9Q3Wodix4icnRu7