Bir zeit University

Faculty of Information Technology

Computer Systems Engineering Department

Digital Lab ENCS 211 EXP. No. 9

A Simple Security System Using FPGA

9.1 Objectives:

- To practice building different digital components using Quartus either by building a Verilog codes & Block diagrams.
- Learning how to put some of the digital components you've studied and build in pervious lab sessions, together to build useful systems.
- To become more familiar with FPGA programming.

9.2 Apparatus:

- A Desk\Lab top with Quartus II (7.2 +) and USB driver installed.
- Altera DE1 system with its datasheets. (For FPGA pins map).

9.3 Pre Lab: (Bring a soft copy of your pre lab with you to the lab)

Prepare each part of the procedure section where it says **(Pre Lab).**

NOTE: ***It's important that you come prepared, as this will reflect your work time during the lab plus it will be a critical variable in the evaluation of your lab report.***

9.4 Theory:

In this experiment we are going to build a simple security system using Altera Quartus software, then we will program and download our system to DE1 Board (FPGA board).

Our security system is simply a 4 digit digital lock, User enter a number of 4 digits (digit range: 0 to 6, so every digit will has a lower limit of 0 and an upper limit of 6) using a keypad (using the six switch keys build in our FPGAs) . Each digit is represented by a 7-segment display and if the total number entered on the displays equals to XXXX a green led is on;

allowing us to pass. Else a red led is always on; blocking us from passing.

The following computer based design will explain the architecture of our system:

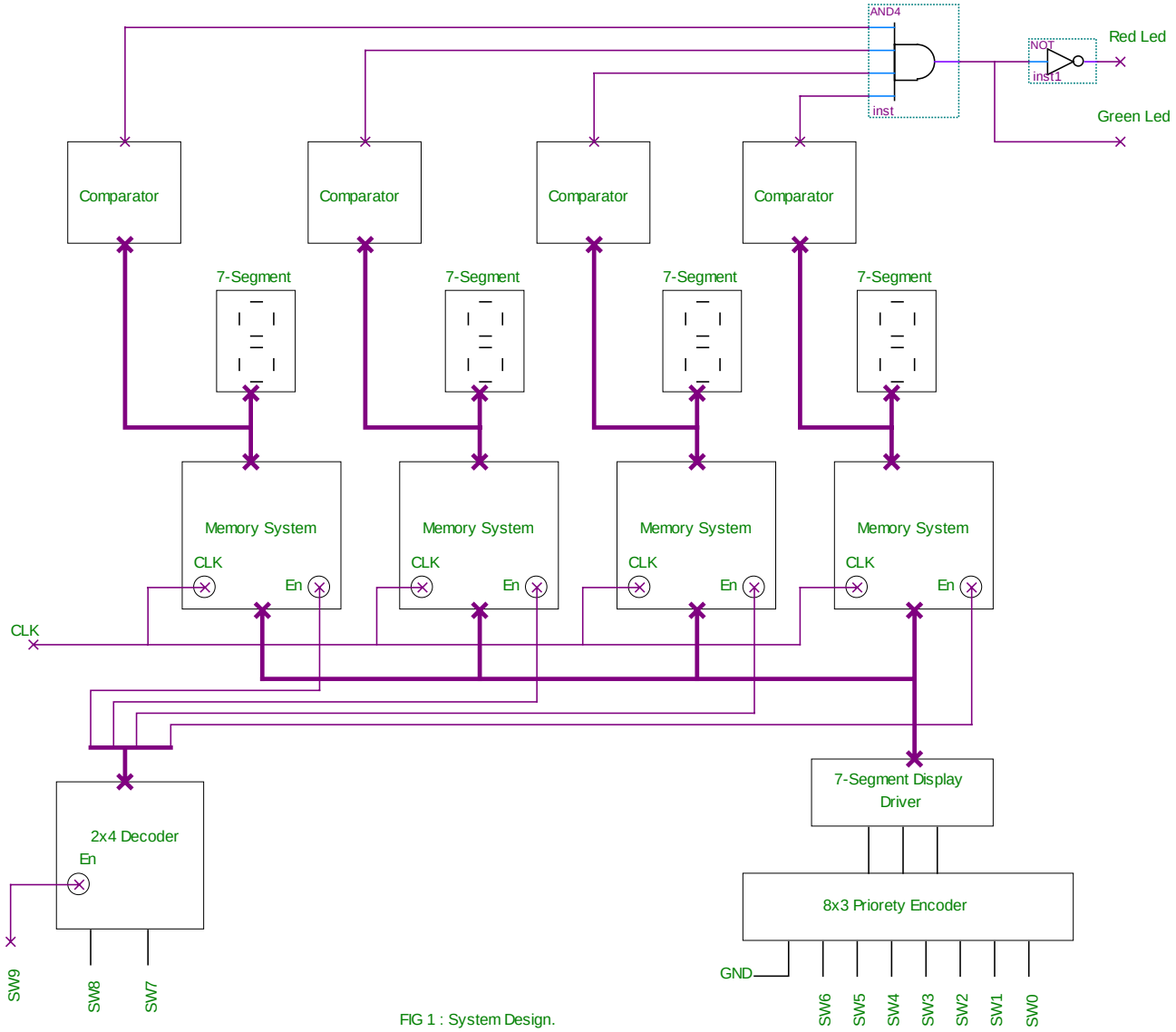


FIG 1 : System Design.

As we can see in the above figure (FIG 1) the system consist of the following components:

1) 8x3 Priority Encoder:

The user will use this priority encoder to choose what value to view on a 7-segment display (values range from 0 to 6 in decimal) , for example if the user switch SW4 to high and keep SW5 and SW6 low then the output of the encoder will be b'100.

2) 2x4 Decoder:

The purpose of this decoder is to let the user select which memory system is active thus which 7-segment display to use, for example if SW8 and SW7 are active high then the En pin of the 4^{th} memory system is enabled and ready to read user input on the 8x3 priority encoder.

Note: The enable pin of the decoder must be active low while switching between selection lines of the decoder.

3) 7-segment display driver:

This driver is used to convert the output of the priority encoder to the proper input for the 7-segment displays, the output of the driver is first stored in a memory unit before transferring to a 7- segment (depends on which memory system is enabled using the 2x4 decoder).

4) Memory System:

The purpose of such system is to ensure that the value selected by user to display on a certain 7-segment is kept there when the user switch to select another 7-segment.

Each memory system is consisting of seven D- flip flops and 2x1 MUXs as seen in the following figure (FIG 2):

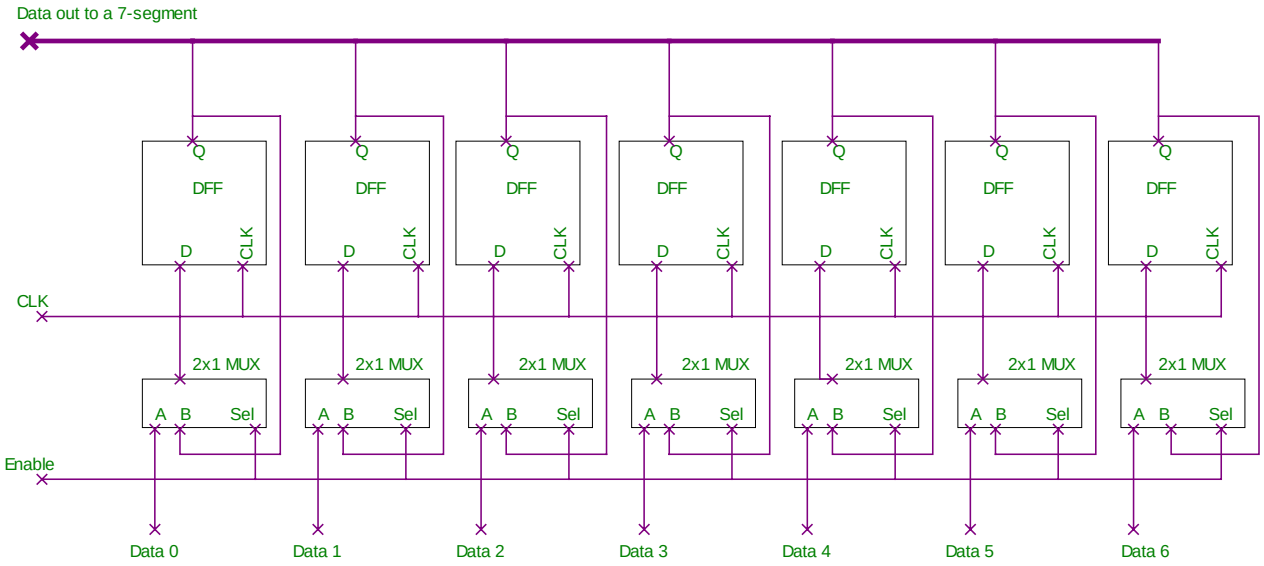


FIG 2 : Memory System.

When the Enable pin =0, the output of each DFF becomes its input at every clock cycle, when the Enable pin becomes 1 the data coming from the 7-segment driver is then stored in the each DFF. The output of each DFF is sent as a data bus to a 7-segment display.
Note: For each 7-segment display we need a memory system block.

5) Comparator:

The input of each 7-segment display is connected also to a comparator , every comparator has a build in value (reference) which is compared with the value of the 7-segment display , if both are equal then the output of the comparator is 1 else the output will be 0 ; for example if one of the comparators has a reference value = 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 7-segment display).

The purpose of the comparator is to lock/unlock our security system.

6) 4-input AND gate:

This AND gate will make sure that all 4 7-segment displays have the correct combination ; if each comparator output = 1 , then the AND gate output will be 1 , thus a green light is on, else a red light will be always on.

9.5 Procedure :

After we understood the architecture of our security system it's time to start programming and designing it (get our feet wet ☺).

NOTE: ***Create a Symbol for each component you build.***

1) **8x3 priority encoder.**
Write down the following code, compile and simulate it **(Pre Lab).**

```
3   module MyEncoder(i0,i1,i2,i3,i4,i5,i6,i7,en_out);
4   input i0,i1,i2,i3,i4,i5,i6,i7;
5   output [2:0] en_out;
6   reg [2:0] en_out;
7
8   always @ (i0 or i1 or i2 or i3 or i4 or i5 or i6)
9
10  if (i6==1'b1)
11   en_out <= 3'b110;
12  else if (i5==1'b1)
13   en_out <= 3'b101;
14  else if (i4==1'b1)
15   en_out <= 3'b100;
16  else if (i3==1'b1)
17   en_out <= 3'b011;
18  else if (i2==1'b1)
19   en_out <= 3'b010;
20  else if (i1==1'b1)
21   en_out <= 3'b001;
22  else
23   en_out <= 3'b000;
24
25  endmodule
```

2) 2x4 Decoder:

Write down the following code, compile and simulate it **(Pre Lab).**

```
module MyDecoder (en,select,out);
input[1:0] select;
input en;
wire[1:0] select;
output[3:0] out;
reg[3:0] out;
always@(select)
begin
if(en==1)
case(select)
0:out = 4'b0001;
1:out = 4'b0010;
2:out = 4'b0100;
3:out = 4'b1000;
endcase
else
out <=4'b0000;
end
endmodule
```

3) 7-segement driver:

Write down the following code, compile and simulate it **(Pre Lab).**

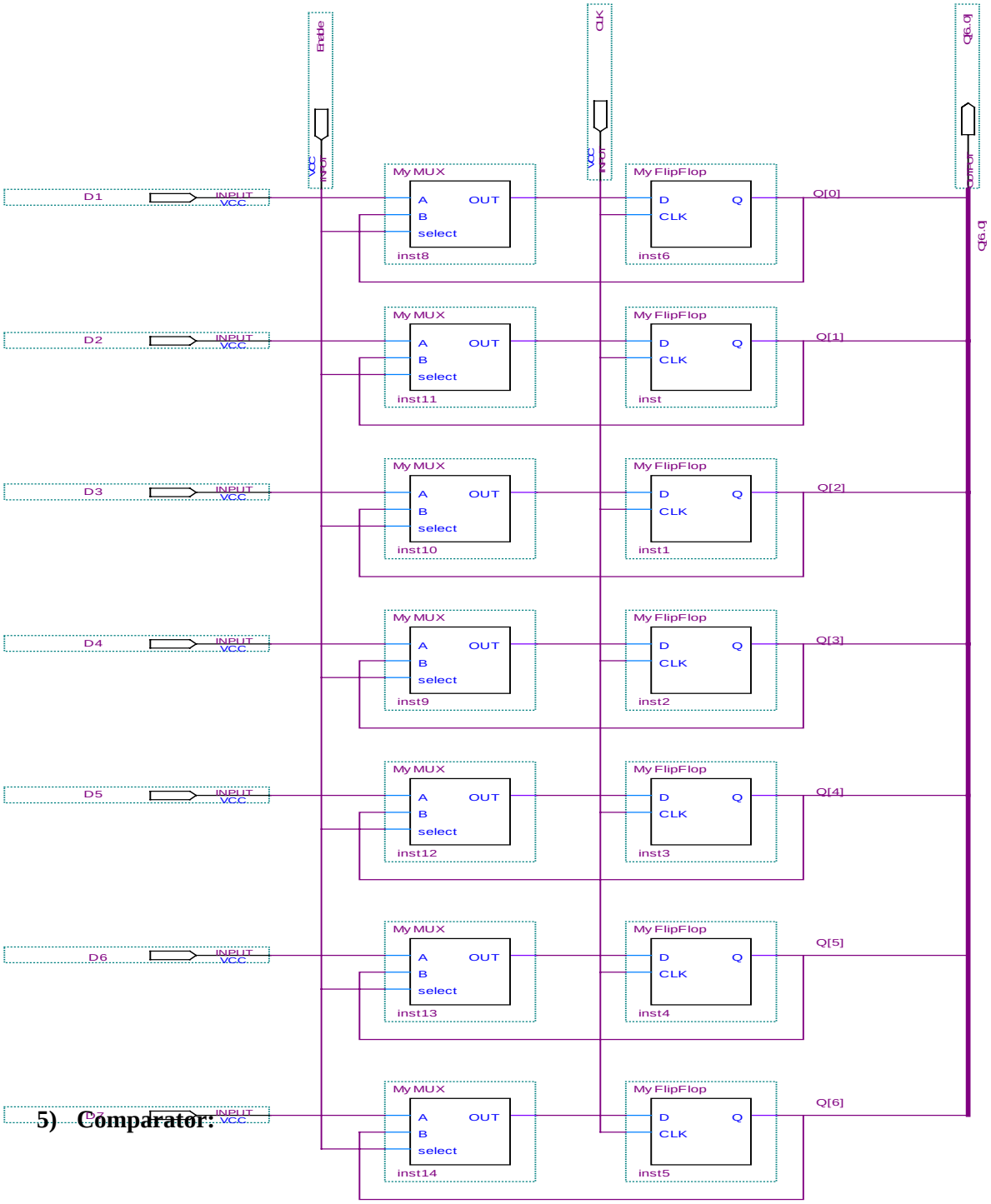
```
module MyDriver(in_v,out_v);
input[2:0]in_v;
wire[2:0]in_v;
output[6:0]out_v;
reg[6:0] out_v;
always@(in_v)
begin
case(in_v)
0:out_v = 7'b0000001;
1:out_v = 7'b1001111;
2:out_v = 7'b0010010;
3:out_v = 7'b0000110;
4:out_v = 7'b1001100;
5:out_v = 7'b0100100;
6:out_v = 7'b0100000;
7:out_v = 7'b0001111;
endcase
end
endmodule
```

4) Memory System:

1- Write the code of a D- Flip Flop, compile and simulate it.

2- Write the code of a 2x1 MUX, compile and simulate it **(Pre Lab).**
Note: ***It's important that your MUX behaves as explained in the memory system section (check back the theory).***

3- Use a block diagram to build the following design :



5) Comparator:

Write down the following code, compile and simulate it **(Pre Lab).**

```
3   module MyComp(CData,out);
4   input [6:0] CData;
5   wire [6:0] CData;
6   output out;
7   reg out;
8   always @(CData)
9   begin
10  if(CData == 7'b0100100)
11   out <=1'b1;
12  else
13  out <=1'b0;
14  end
15  endmodule
```

Note: ***For simplifying reasons we will build one comparator based on a reference value X (in this example X = 5), you can build four different comparator with four different values to compare with.***

6) Putting everything together:

Build and design the security system using the components you build during the previous sections.

The final block design should look as the one in FIG 4 (check the next page please).

7) Assign pins values to the security system design you just build and then download the system to the FPGA board.

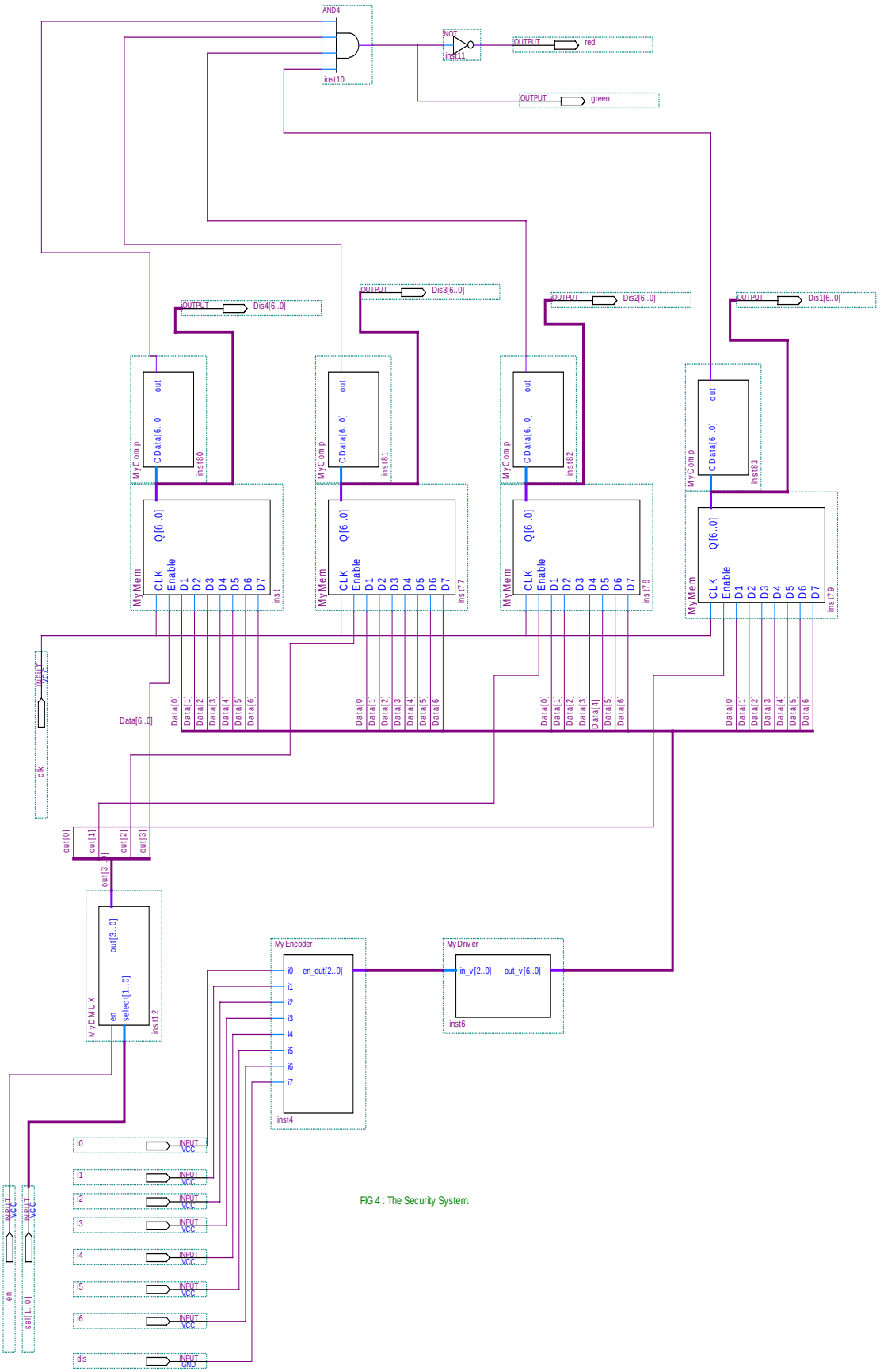


FIG 4 : The Security System.