Kevin Karnadi Kirmansjah 紀維鑫

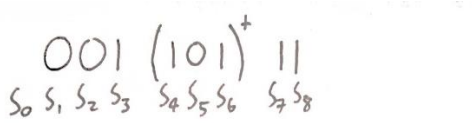
109006241

Logic Design

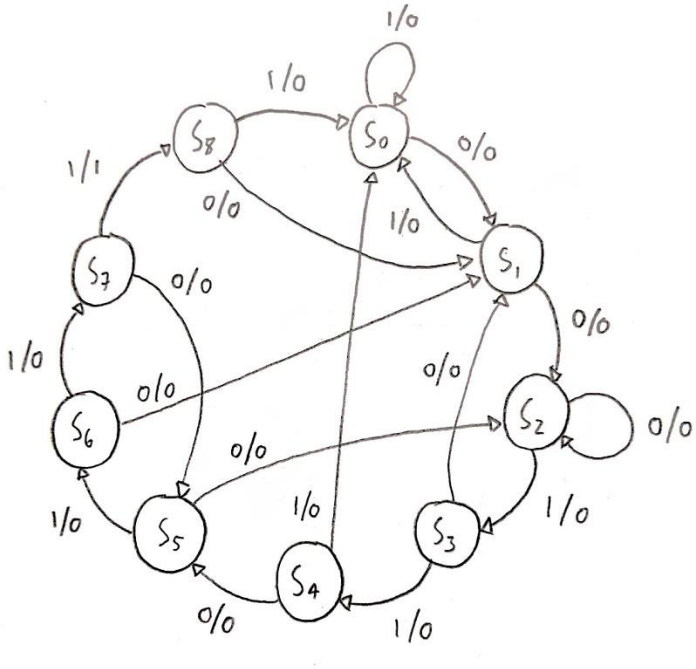
Lab 3: Pattern Matching

I. Design of my FSM

In this assignment, I'm asked to detect the pattern 001(101)11. To do this, I created a corresponding state table and state diagram for the sequence detector.



Q(t)	Q(t+1)		Output Y	
	X = 0	X = 1	X = 0	X = 1
S_0	S_1	S_0	0	0
S_1	S_2	S_0	0	0
S_2	S_2	S_3	0	0
S_3	S_1	S_4	0	0
S_4	S_5	S_0	0	0
S_5	S_2	S_6	0	0
S_6	S_1	S_7	0	0
S_7	S_5	S_8	0	1
S_8	S_1	S_0	0	0



In my FSM, each bit in the 001(101)11 pattern is represented as states S0 up to S8. S0 means that the pattern hasn't started yet, S1 represents the 1st bit (0), S2 represents the 2nd bit (0), S3 represents the 3rd bit (1), and so on. The input X will determine the next state of the machine. Output Y will be 1 when the state S8 is reached, which means that the pattern is detected. In order to be able to repeat the sequence of 101 in the middle of the pattern, when the machine is currently in state S7 and the input is 0, the pattern detecting process won't be returned to the start, but it will be returned to S5 instead. The rest of the pattern detection process is simpler, and it can be seen on the state table and state diagram above.

II. How I implement my FSM in Verilog

The module uses 3 inputs, which are clk, reset, data, and will produce 1 output, which is flag. I created new parameters to represent states S0 to S8, which is represented by its' 4-bit binary form. For example, S5 will be represented as 0101. I also created two new regs called cur and next, which is used to store the current value and the next value of the machine, respectively.

```
input clk, reset, data;
output flag;

parameter S0 = 4'b0000;
parameter S1 = 4'b0001;
parameter S2 = 4'b0010;
parameter S3 = 4'b0011;
parameter S4 = 4'b0100;
parameter S5 = 4'b0101;
parameter S6 = 4'b0110;
parameter S7 = 4'b0111;
parameter S8 = 4'b1000;
reg[3:0] cur, next;
```

Flag will be assigned as 1 if the current state S8, which means that the pattern is detected. Otherwise, the value of flag will be 0.

```
assign flag = (cur == S8) ? 1 : 0;
```

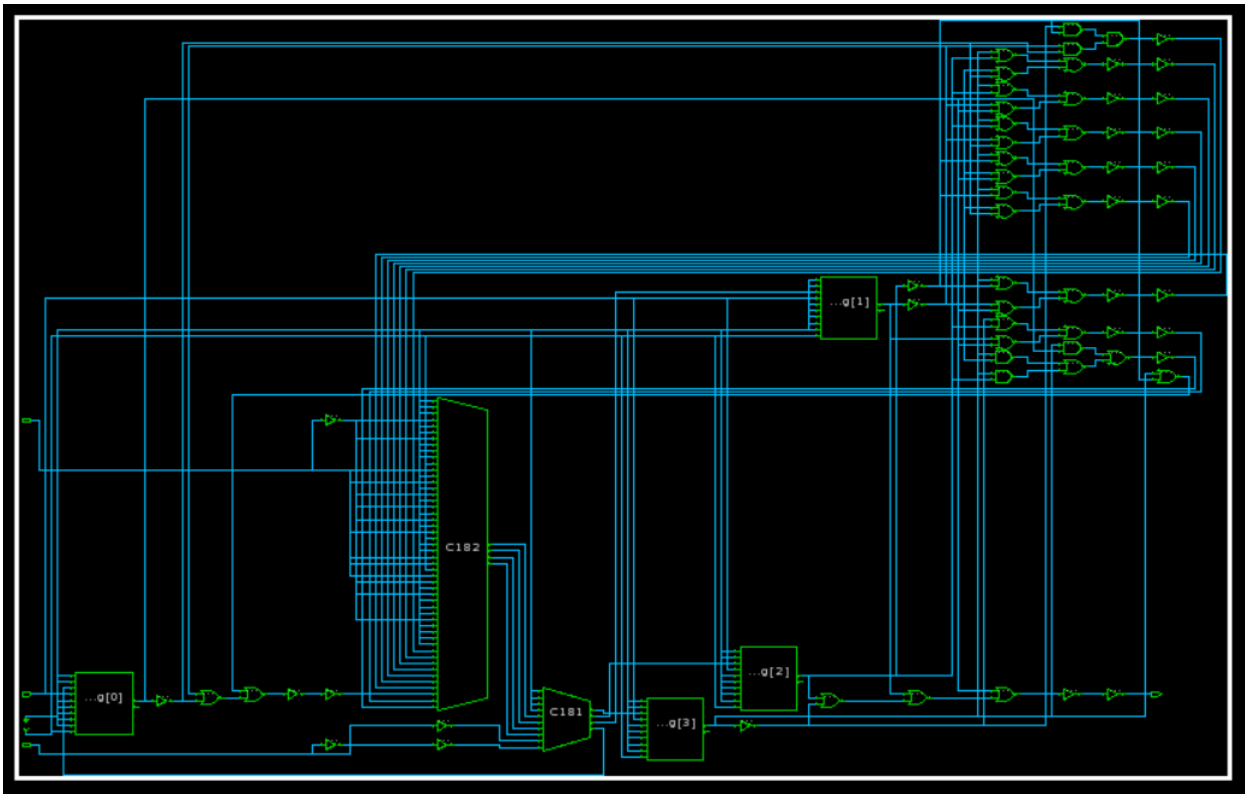
I used an always block that will be triggered at the posedge of clk to assign a value to the current state. If reset equals to 1, it means that the pattern detection process hasn't started yet, so the current state will be assigned as S0. On the other hand, if reset is 0, it means that the pattern detection process has already started, and current state will be assigned as the next state, that will be determined based on the machine's present state and input X.

```
always@(posedge clk)
begin
    if(reset)
        cur <= S0;
    else
        cur <= next;
end
```

To determine the values of the next state, I will use an always block which uses cur and data as the parameters. I used the state table that I've written earlier as my reference to help me decide the next state of my machine. Then, I used multiple case and if statements to determine which next state to take based on the value of the current state and the data input.

```
always@(cur, data)
begin
    case(cur)
        S0:
        begin
            if(data == 0)
                next = S1;
            else if(data == 1)
                next = S0;
        end
        S1:
        begin
            if(data == 0)
                next = S2;
            else if(data == 1)
                next = S0;
        end
        S2:
        begin
            if(data == 0)
                next = S2;
            else if(data == 1)
                next = S3;
        end
        S3:
        begin
            if(data == 0)
                next = S1;
            else if(data == 1)
                next = S4;
        end
        S4:
        begin
            if(data == 0)
                next = S5;
            else if(data == 1)
                next = S0;
        end
        S5:
        begin
            if(data == 0)
                next = S2;
            else if(data == 1)
                next = S6;
        end
        S6:
        begin
            if(data == 0)
                next = S1;
            else if(data == 1)
                next = S7;
        end
        S7:
        begin
            if(data == 0)
                next = S5;
            else if(data == 1)
                next = S8;
        end
        S8:
        begin
            if(data == 0)
                next = S1;
            else if(data == 1)
                next = S0;
        end
        default
            next = S0;
    endcase
end
```

This is what the design of the PAT, obtained by using Design Vision, looks like:



III. The problems I faced and how I deal with it

One of the biggest problems that I've faced in this assignment is about how to implement my sequence detector in Verilog. But after doing some research on the internet and watching some tutorial videos, now I know how to implement my sequence detector as a Verilog code.

Another problem that I've faced in this assignment is when I'm in the process of making the state table and state diagram. At first, I've made some mistakes about which next state to take, but after checking my state table and state diagram for some time and after I carefully did a rework on it, I finally created the correct state table and state diagram for this sequence detector.

IV. The questions I want to ask TAs

Is there any other way to implement a sequence detector in Verilog other than by using always blocks?