Hacettepe University

Computer Engineering Department

BM233 Logic Design Lab - 2021 Fall

Experiment 5 - Sequential Circuits in Verilog

December 22, 2021

Student name:
İrem CENGİZ

Student Number:
b2200356006

1 Problem Definition

In the movie, Bender's Big Score, a level 87 binary code is discovered. The code is used for paradox-free time travel. The problem is to design a finite state machine that would be used to test binary sequences generated by a generator module installed to Bender, which has a chance to trigger a lower level supernatural effect.

2 Solution Implementation

2.1 given state transition diagram

We will use this state diagram:

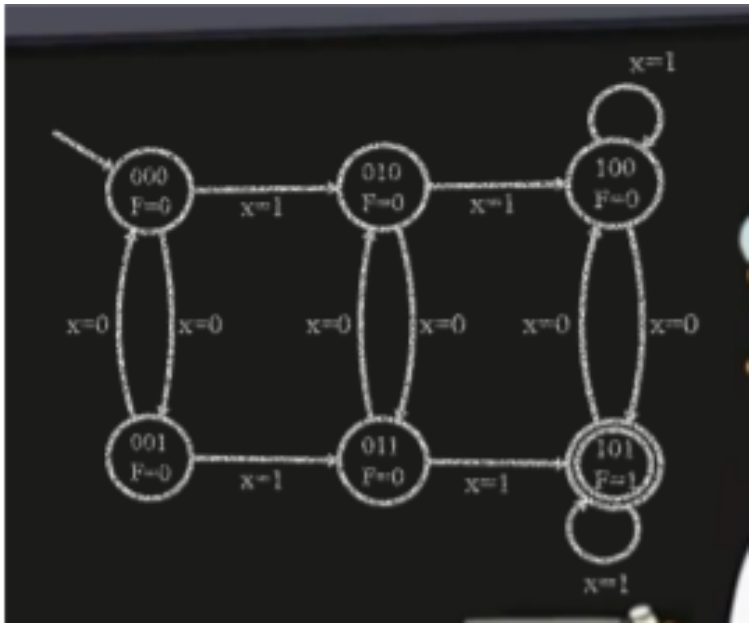

Figure 1: Diagram of the machine which recognizes at least two 1's and an odd number of 0's

2.2 convert the state transition diagram into a state transition table (binary coded state table).

Q2Q1Q0 = ABC , I : input

Q_2	Q_1	Q_0	I	Q_2*	Q_1*	Q_0*	output
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	0	0	0
0	1	1	0	0	1	0	0
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	1
1	0	1	1	1	0	1	1

present state next state

Figure 2:

There are 6 states and no reduction.
Number of bits is 3 and we will use 3 D flip flops.
Formula for determine number of d D flip flops:
$(2^n >= 6 - - - - - - - > n = 3)$

2.3 truth tables of D(A), D(B), D(C), Y

Take Present state:ABC , input:x , output:Y

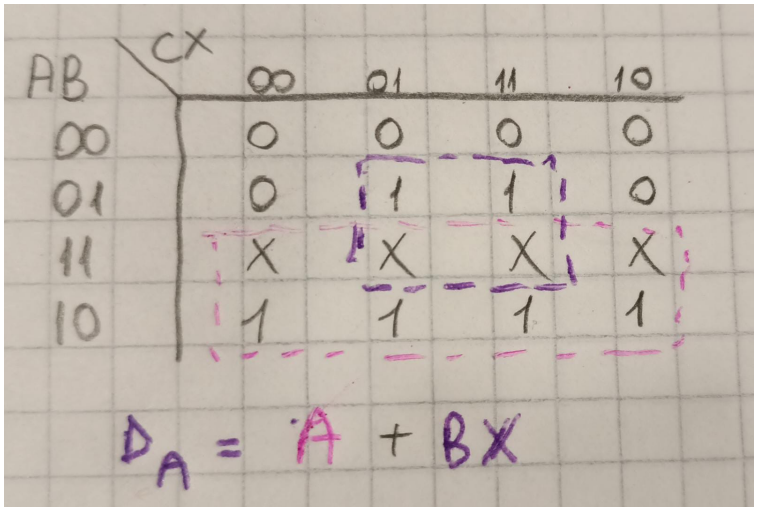

Figure 3: Function for D(A)

SOP form D(A) $=F(A,B,C,x) = \sum(5,7,8,9,10,11) + d(12,13,14,15)$

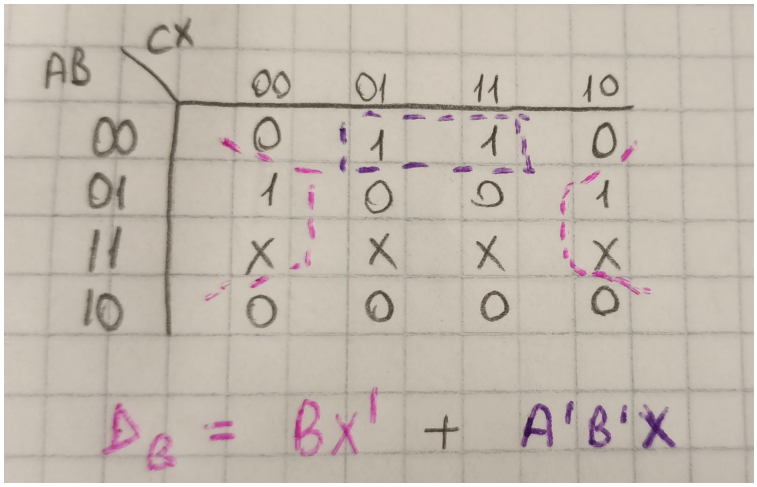

Figure 4: Function for D(B)

SOP form D(B) $=F(A,B,C,x) = \sum(1,3,4,6) + d(12,13,14,15)$

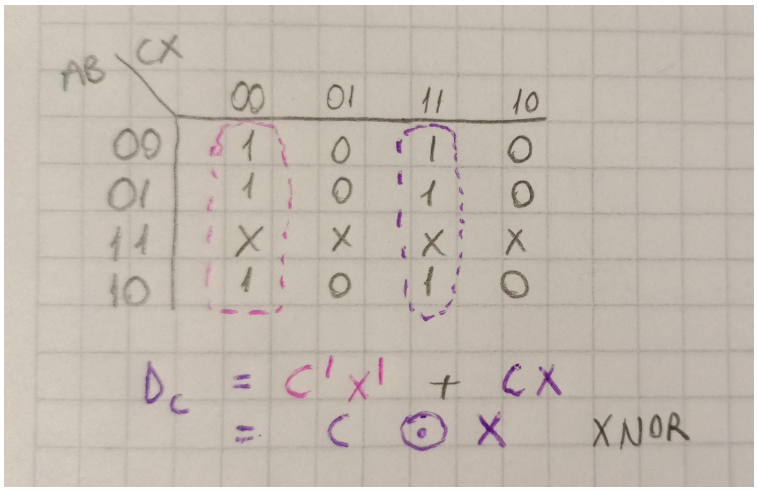

Figure 5: Function for D(C)

SOP form D(C) $= F(A, B, C, x) = \sum(0, 3, 4, 7, 8, 11) + d(12, 13, 14, 15)$
SOP form Y $= F(A, B, C) = \sum(5) + d(6, 7)$

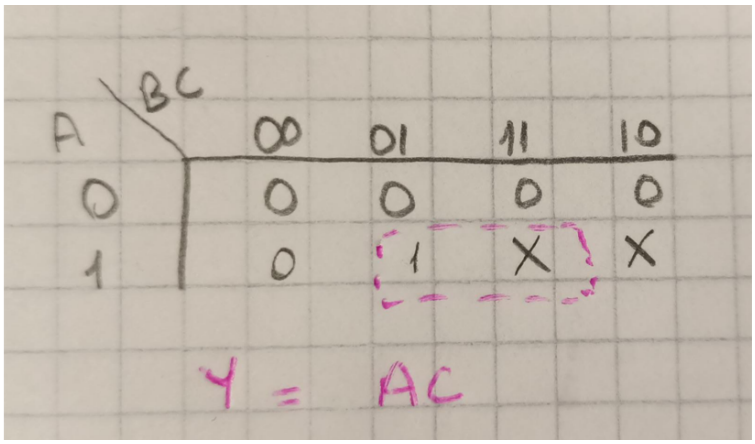

Figure 6: Function for output

2.4 Circuit diagram

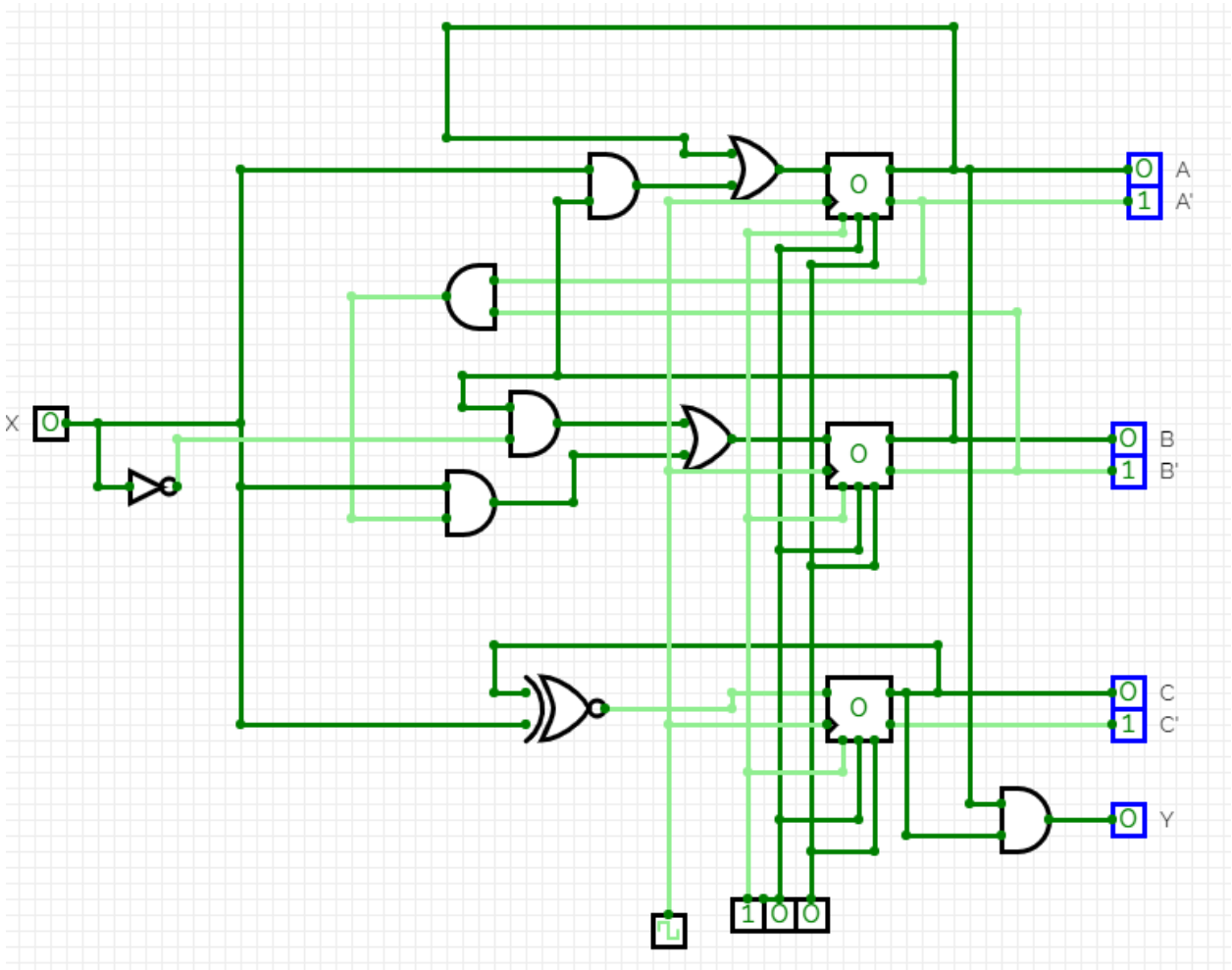

Figure 7: circuit

3 VERILOG CODES

3.1 code : dff.v

```
module dff (input d,
            input rst,
            input clk,
            output reg q);

 always @(posedge clk or posedge rst)
 begin
  if(rst)
   q <= 0;
  else
   q <= d;
 end
endmodule
```

3.2 code : machine-d.v

```
module machine_d(
    input x,
    input rst,
    input clk,
    output F
    );
    reg[2:0] present_state;     //ABC
    wire[2:0] next_state;       //D(A)D(B)D(C)

    //instantiating D flip flops (structural description and explicit association)
    //Functions of D(A),D(B),D(C),Y are used here
    dff D2(  //for A
        .d(present_state[2] | (present_state[1]&x)),
        .rst(rst),
        .clk(clk),
        .q(next_state[2])
    );
    dff D1(  //for B
        .d((present_state[1]&~x) | (~present_state[2]&(~present_state[1])&x)),
         .rst(rst),
        .clk(clk),
        .q(next_state[1])
    );
    dff D0(  //for  C
        .d((~present_state[0]&~x) | (present_state[0]&x)),
        .rst(rst),
        .clk(clk),
        .q(next_state[0])
    );

    always@ (rst or next_state) begin
        if (rst) begin present_state <= 3'b000; end
        else begin present_state <= next_state; end
    end

    assign F=(present_state[2] & present_state[0] )  ;
endmodule
```

4 Testbench Implementation

My test case is 11110111110000101010001 and I will take input from rightward.

```
module machine_d_tb();
//inputs
    reg x;
    reg clk;
    reg rst;
//output
    wire F;
    integer i;
    //instantiate the uut
    machine_d uut (.x(x), .rst(rst), .clk(clk), .F(F));

  //to give input to fsm I will use shifting
    integer shift_amount;
    reg [22:0] input_data;

    initial begin
        clk=1;
        forever begin
           #5; clk=~clk;
        end
    end

    initial
       #300
       $finish;
    initial begin
       $dumpfile("machine_d.vcd");
       $dumpvars;
        input_data=23'b11110111110000101010001;
        shift_amount=0;
        rst=1; #15;
        rst=0; #97;
        rst=1; #3;
        rst=0; #100;
        rst=1; #4;
        rst=0; #50;
    end
   //input sequence not completely synced with the clock
    always@ (posedge clk ) begin
        #1;
        x=input_data >> shift_amount;
        shift_amount=shift_amount+1;
    end
endmodule
```

5 Results

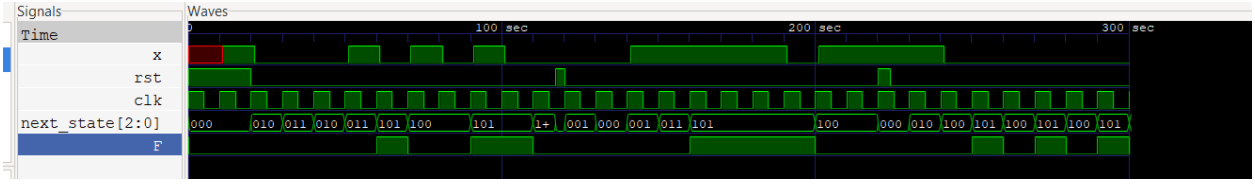

Figure 8: Resulting Waveform

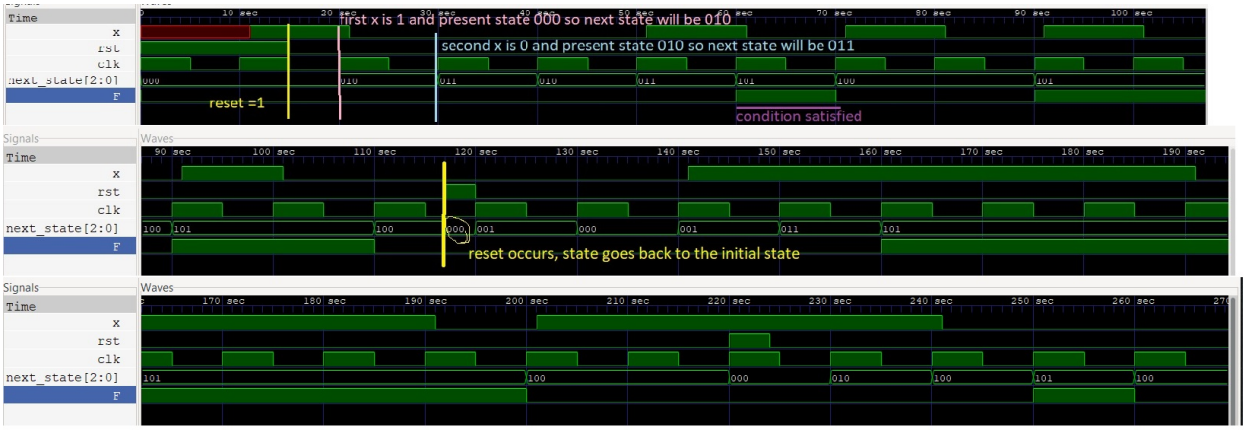

Figure 9: Resulting Waveform

In my input 11110111110000101010001:(from right to left)
first state 000: state will stay the same until reset time expires than:
present state + input — next state
000 + 1 — 010
010 + 0 — 011
011 + 0 — 010
010 + 0 — 011
011 + 1 — 101
101 + 0 — 100
100 + 1 — 100
100 + 0 — 101
101 + 1 — 101
101 + 0 — 100
reset — state = 000
000 + 0 — 001

001 + 0 — 000
000 + 0 — 001
001 + 1 — 011
011 + 1 — 101
101 + 1 — 101
101 + 1 — 101
101 + 1 — 101
101 + 0 — 100
100 + 1 — 100
100 + 1 — 100
reset — state = 000
000 + 1 — 010
010 + 1 — 100
.....
all the case tested.
(6 states take input 0 and 1, give correct next states (we can check using state diagram))

References

- Reference 1 https://www.fpga4student.com/2017/02/verilog-code-for-d-flip-flop.html
- Reference 2 https://piazza.com/class$_profile/get_resource/kska8e2rsbg1ja/kwt8vokjhja3pa$

6 IMAGES in figure 9

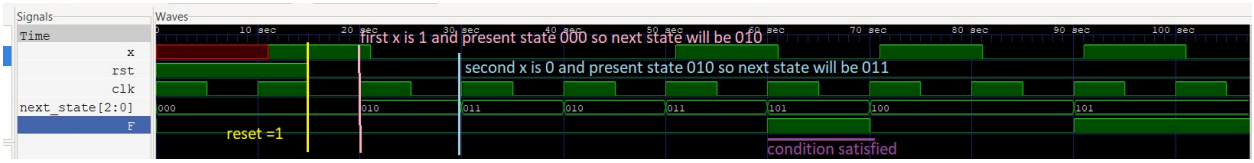

Figure 10: Resulting Waveform

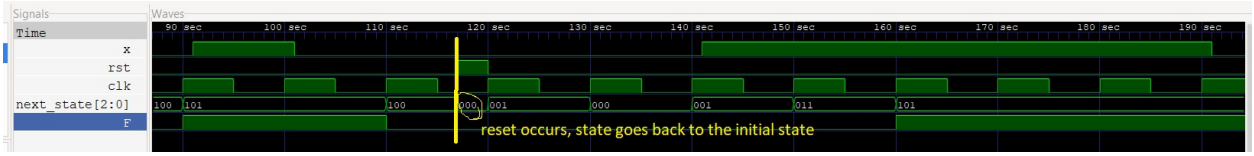

Figure 11: Resulting Waveform

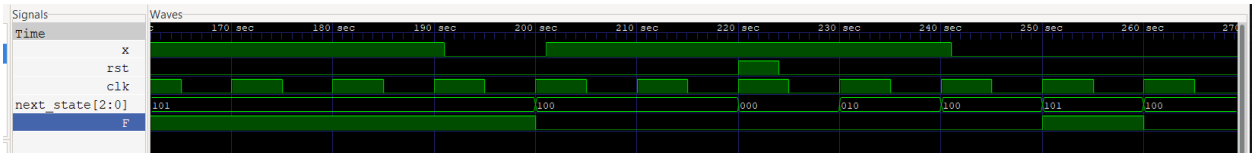

Figure 12: Resulting Waveform