Hacettepe University

Computer Engineering Department

BM233 Logic Design Lab - 2021 Fall

Experiment 5 - Sequential Circuits in Verilog

December 19, 2021

Student name:
Mert Ali Yalçın

Student Number:
21946682

1 Problem Definition

Sequential circuits are circuits whose outputs depend on both the present inputs and the sequence of past inputs (sequential circuits include memory elements). That is, the previous inputs or state of the circuit will have an effect on its present state.
Storage Elements
A flip flop is a basic building block of sequential logic circuits. It is a circuit that has two stable states and can store one bit of state information. The output changes state by signals applied to one or more control inputs. A basic D Flip Flop has a D (data) input, a clock (CLK) input and outputs Q and Q' (the inverse of Q). Optionally it may also include the PR (Preset) and CLR (Clear) control inputs.

To design a sequential circuit, start with the problem definition and use the following steps:

- Create a state transition diagram from the description. Reduce the number of states if necessary, and assign binary values to the states.
- Convert the state transition diagram into a state transition table (binary coded state table).
- Determine the number of flip-flops needed and choose flip-flop types. Derive their excitation tables (if designing a sequential circuit with flip-flops other than the D type), and derive input and output equations from the state table. Minimize the functions for the flip-flop inputs (e.g. using Karnaugh Maps).
- Determine the combinatorial circuit to represent the output (if any).
- Finally, use simplified functions to design sequential circuit and obtain the design schematic.

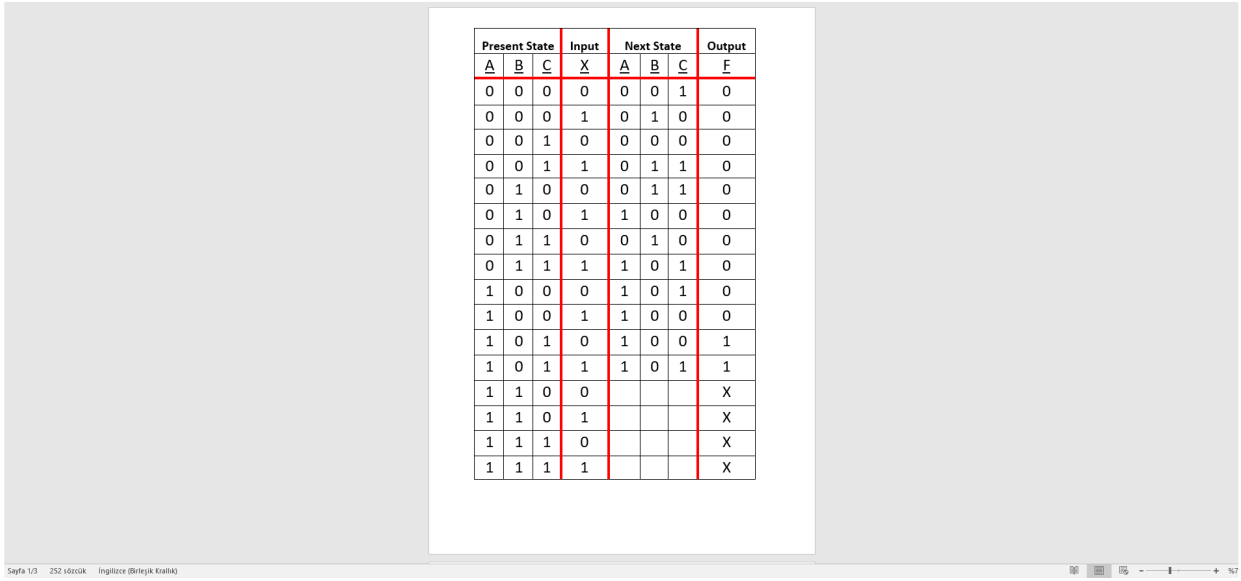
You should follow the instructions below:

- Use sequential circuit design steps described on the previous page to obtain the design schematic. Use D flip flops as storage elements to store the state information.
- Implement the design in Verilog using STRUCTURAL design approach following the circuit schematic. Behavioral design will be graded as 0. Name your D flip flop module as dff.v and machine module as machine_d.v
- You must use a Rising Edge D Flip-Flop with Asynchronous Reset on High Level. I.e., it should be triggered on the rising edges of the both clk and the rst signals, as can be seen in the waveform below.
- You MUST download and use these starter code files before you start working! Do NOT change the I/O port names!

- Verify the Verilog model of the machine by writing an appropriate testbench machine_d_tb.v for all possible test cases.
- Make sure to obtain a similar waveform which shows the correctness of your design. And explain the obtained results in your report. You MUST test for all possible state transitions

2 State Transition Table

Fig. 1.

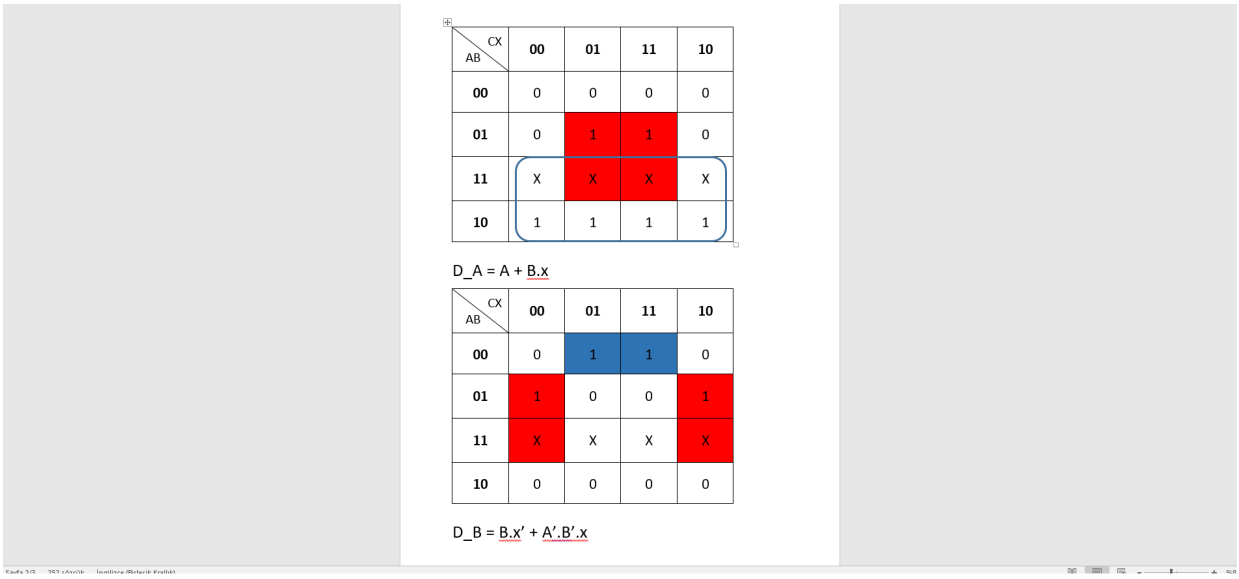


Present State			Input	Next State			Output
A	B	C	X	A	B	C	F
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	0	0	0
0	1	1	0	0	1	0	0
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	1
1	0	1	1	1	0	1	1
1	1	0	0				X
1	1	0	1				X
1	1	1	0				X
1	1	1	1				X

Figure 1: State Transition Table

3 Functions

Since I have three states, I need three D flip flops to represent them.
Fig. 2.



CX / AB	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	X	X	X	X
10	1	1	1	1

D_A = A + B.x

CX / AB	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	X	X	X	X
10	0	0	0	0

D_B = B.x' + A'.B'.x

Figure 2: K-maps for D_A and D_B

Fig. 3.

AB \ CX	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	X	X	X	X
10	1	0	1	0

D_C = C'.x' + C.x

AB \ CX	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	X	X	X	X
10	0	0	1	1

F = A.C

Figure 3: K-maps for D_C and F

4 Design schematic

Fig. 4.

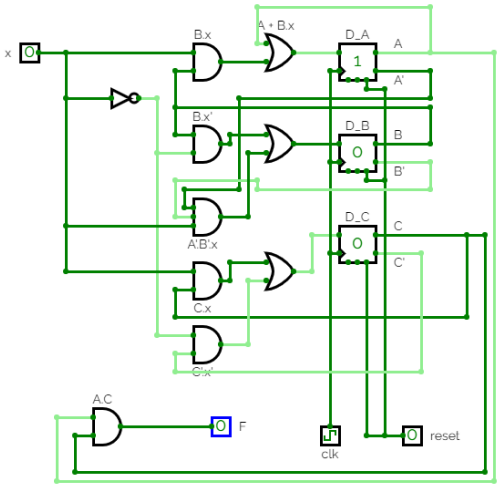


Figure 4: Design Schematic

5 Solution Implementation

dff.v

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

               always @(posedge clk or posedge rst)
               begin
               if(rst == 1'b1)
                 q <= 1'b0;
               else
                 q <= d;
               end
endmodule
```

machine_d.v

```
module machine_d(
    input x,
    input rst,
    input clk,
    output F);

    reg [2:0] present_state = 3'b000;
    wire [2:0] next_state;

    dff DA
    (
        .d(present_state[2] | (present_state[1] & x)),
        .rst(rst),
        .clk(clk),
        .q(next_state[2]));

    dff DB
    (
        .d((present_state[1] & ~x) | (~present_state[2] & ~present_state[1] & x)),
        .rst(rst),
        .clk(clk),
        .q(next_state[1]));

    dff DC
    (
        .d((~present_state[0] & ~x) | (present_state[0] & x)),
        .rst(rst),
        .clk(clk),
        .q(next_state[0]));
```

```
30
31      always @(rst or next_state) begin
32          if(rst == 1'b1)
33              present_state <= 3'b000;
34          else
35              present_state <= next_state;
36      end
37
38      assign F = present_state[2] & present_state[0];
39      assign state = present_state;
40  endmodule
```

machine_d_tb.v

```
1   module machine_d_tb;
2   //Inputs
3   reg clk;
4   reg rst;
5   reg x;
6   //Output
7   wire F;
8   machine_d UUT(.x(x), .rst(rst), .clk(clk), .F(F));
9   reg [21:0] input_data;
10  integer shift_amount;
11  initial begin
12      input_data = 22'b0001111000000111000100;
13      shift_amount = 0;
14      x = 0;
15      //second output
16      rst = 1; #27;
17      rst = 0; #98;
18      rst = 1; #2;
19      rst = 0; #95;
20      //First output
21      //rst = 1; #15;
22      //rst = 0; #160;
23      //rst = 1; #10;
24      //rst = 0; #75;
25      $finish;
26  end
27  initial begin //generate clock
28      clk = 0;
29      forever begin
30          #5;
31          clk = ~clk;
32      end
33  end
34  always @(posedge clk) begin
35      x = input_data>>shift_amount;
```

```
36      shift_amount=shift_amount+1;
37  end
38  endmodule
```

6 Resulting Waveform

Fig. 5

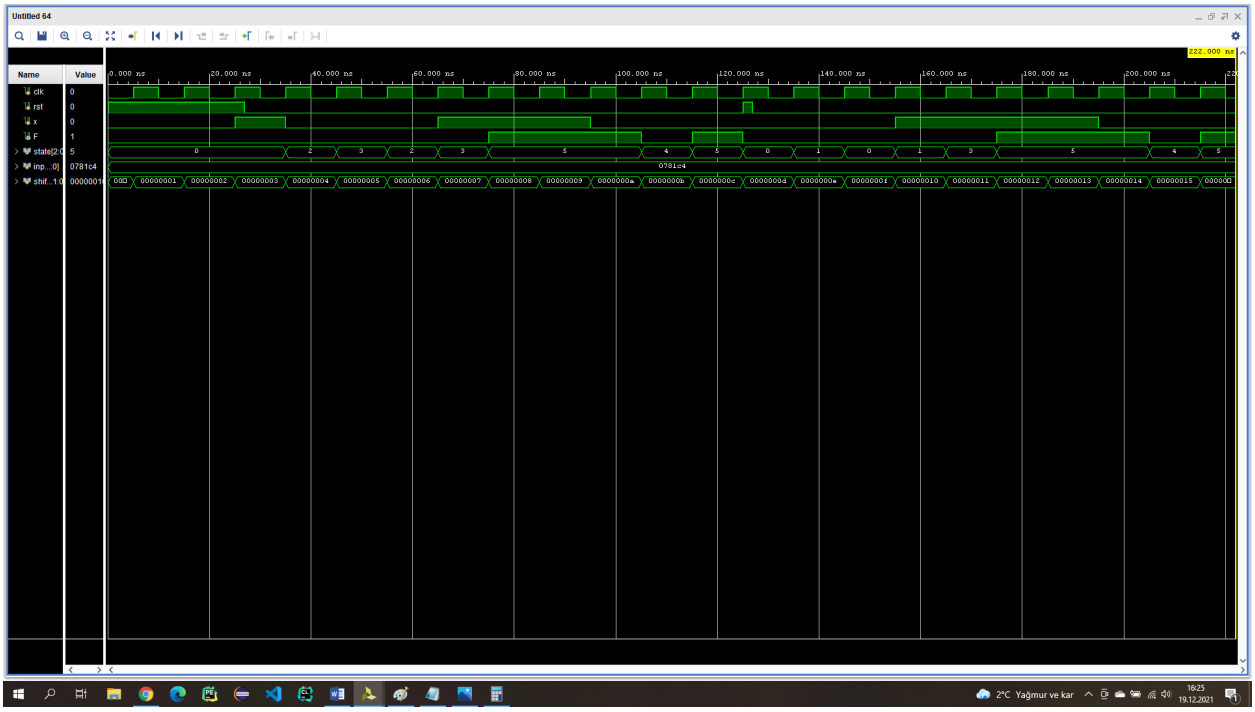


Figure 5: Resulting Waveform

7 Results

Fig. 6.

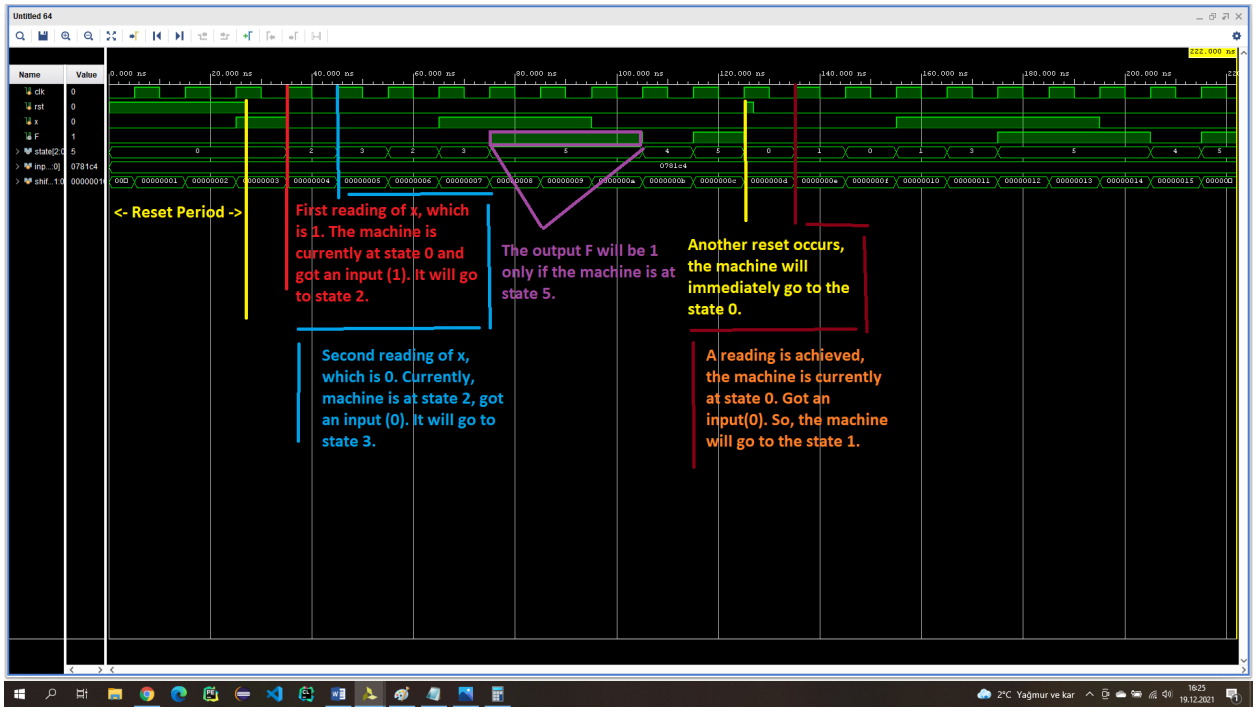


Figure 6: Resulting waveform explained

References

- BBM231 Lecture Notes
- BBM233 Lecture Notes