Hacettepe University

Computer Engineering Department

BM233 Logic Design Lab - 2021 Fall

Verilog Assignment 2

December 22, 2021

Student name:
Metin Eren Oktay

Student Number:
2210356137

1 Problem Definition of The Implementation with D Flip-Flops

We are required to create a sequential circuit that recognizes at least two 1's and an odd number of 0's using D flip-flops in Verilog.

2 Transition Table for Implementation Using D Flip-Flop

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

Figure 1: Transition Table for Implementation Using D Flip-Flop

3 D Flip-Flop And Output Equations Using K-Maps

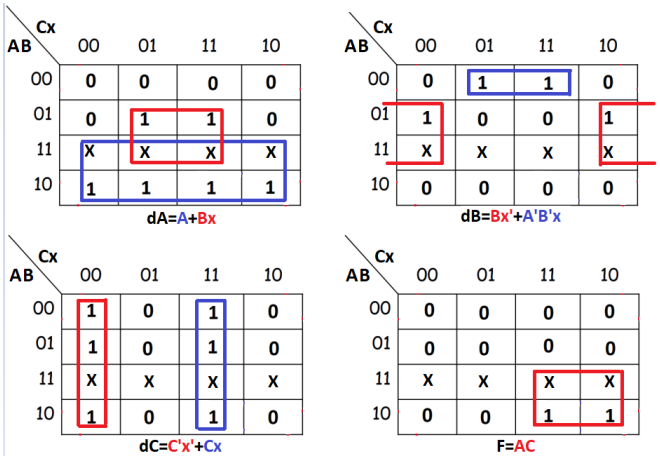

Figure 2: D Flip-Flop And Output K-Maps

4 Design Schematic with D Flip-Flops

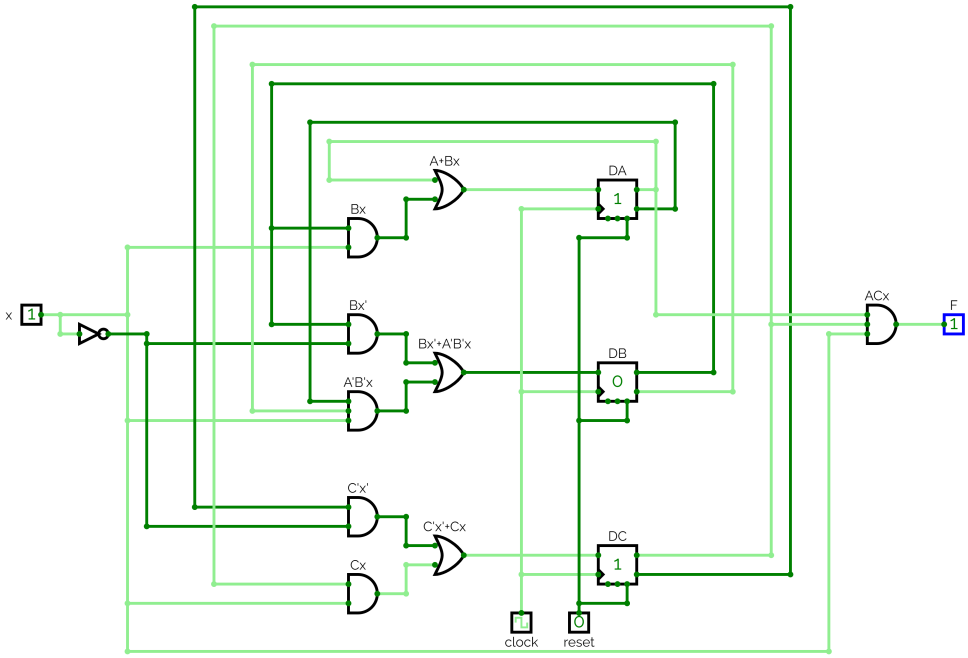

Figure 3: Design Schematic with D Flip-Flops

5 Verilog Code Solutions with D Flip-Flops

5.1 dff

```
1  `timescale 1ns / 1ps
2  module dff (input d,
3                input rst,
4                input clk,
5                output reg q);
6      always @(posedge clk or posedge rst)
7      begin
8          if(rst)
9              q <= 0;
10         else
11             q <= d;
12     end
13 endmodule
```

5.2 machine_d

```
`timescale 1ns / 1ps
module machine_d(
    input x,
    input rst,
    input clk,
    output F
    );
    //current_state[2] = A
    //current_state[1] = B
    //current_state[0] = C
    reg [2:0] current_state = 3'b000;
    //next_state[2] = d_A
    //next_state[1] = d_B
    //next_state[0] = d_C
    wire [2:0] next_state;
    dff d_A (
                .d((current_state[2])|(current_state[1]&x)),
                .clk(clk),
                .rst(rst),
                .q(next_state[2]));
    dff d_B (
                .d((current_state[1]&(~x))|(~current_state[2]&~current_state[1]&x)),
                .clk(clk),
                .rst(rst),
                .q(next_state[1]));
    dff d_C (
                .d((~current_state[0]&(~x))|(current_state[0]&x)),
                .clk(clk),
                .rst(rst),
                .q(next_state[0]));
    always @(rst or next_state) begin
        if(rst) begin current_state <= 3'b000; end
        else begin current_state <= next_state; end
    end
    assign F = (current_state[2]&current_state[0]);
endmodule
```

5.3 machine_d_tb

```
‘timescale 1ns / 1ps
module machine_d_tb;
    reg x;
    reg rst;
    reg clk;
    wire F;
    machine_d uut(.x(x), .rst(rst), .clk(clk), .F(F));
    reg[19:0] input_data_1;
    integer shift_amount_1;
    initial begin
        input_data_1=20’b11110000000111000100;
        shift_amount_1=0;
        x=0; rst=1; #22;
        rst=1; #5;
        rst=0; #98;
        rst=1; #2;
        rst=0; #93;
        $finish;
    end
    initial begin
        clk=1;
        forever begin
            #5;
            clk=~clk;
        end
    end
    always @(posedge clk or posedge rst) begin
        #2;x=input_data_1>>shift_amount_1;
        shift_amount_1=shift_amount_1+1;
    end
endmodule
```

6 Waveform of The Implementation with D Flip-Flops

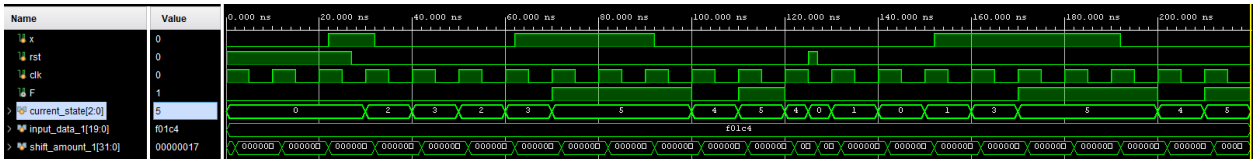

Figure 4: Waveform of machine_d

7 Results for The Implementation with D Flip-Flops

As you can see below; D flip-flops that assigns the values of states are triggered on the rising edges of the both clk and the rst signals, current_state values and output change according to the characteristic equations of the D flip-flops such that dA=A+Bx, dB=Bx'+A'B'x, dC=C'x'+Cx, F=AC and if a reset occurs, state goes back to the initial state.

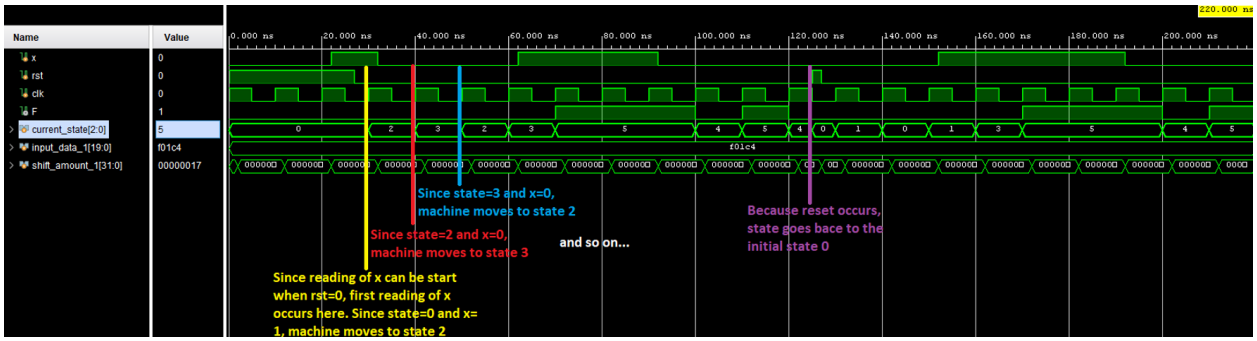

Figure 5: Waveform of machine_d with Explanations

8 Problem Definition of The Implementation with JK Flip-Flops

We are required to create a sequential circuit that recognizes at least two 1's and an odd number of 0's using JK flip-flops in Verilog.

9 Transition Table for Implementation Using JK Flip-Flop

Present State			Input	Next State			Output	JK Flip-Flop A		JK Flip-Flop B		JK Flip-Flop C	
A	B	C	x	A	B	C	F	JA	KA	JB	KB	JC	KC
0	0	0	0	0	0	1	0	0	X	0	X	1	X
0	0	0	1	0	1	0	0	0	X	1	X	0	X
0	0	1	0	0	0	0	0	0	X	0	X	X	1
0	0	1	1	0	1	1	0	0	X	1	X	X	0
0	1	0	0	0	1	1	0	0	X	X	0	1	X
0	1	0	1	1	0	0	0	1	X	X	1	0	X
0	1	1	0	0	1	0	0	0	X	X	0	X	1
0	1	1	1	1	0	1	0	1	X	X	1	X	0
1	0	0	0	1	0	1	0	X	0	0	X	1	X
1	0	0	1	1	0	0	0	X	0	0	X	0	X
1	0	1	0	1	0	0	1	X	0	0	X	X	1
1	0	1	1	1	0	1	1	X	0	0	X	X	0

Figure 6: Transition Table for Implementation Using JK Flip-Flop

10 JK Flip-Flop And Output Equations Using K-Maps

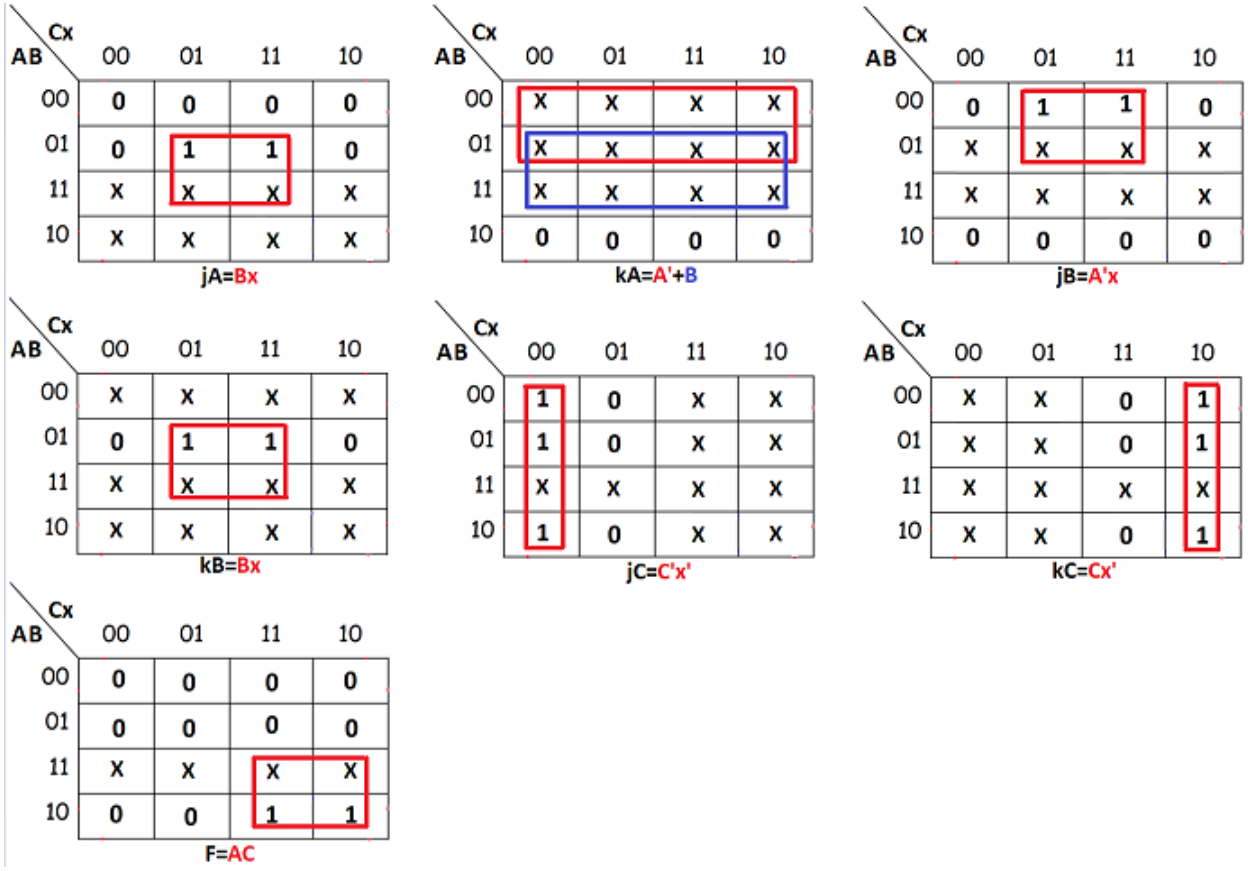

Figure 7: JK Flip-Flop And Output K-Maps

11 Design Schematic with JK Flip-Flops

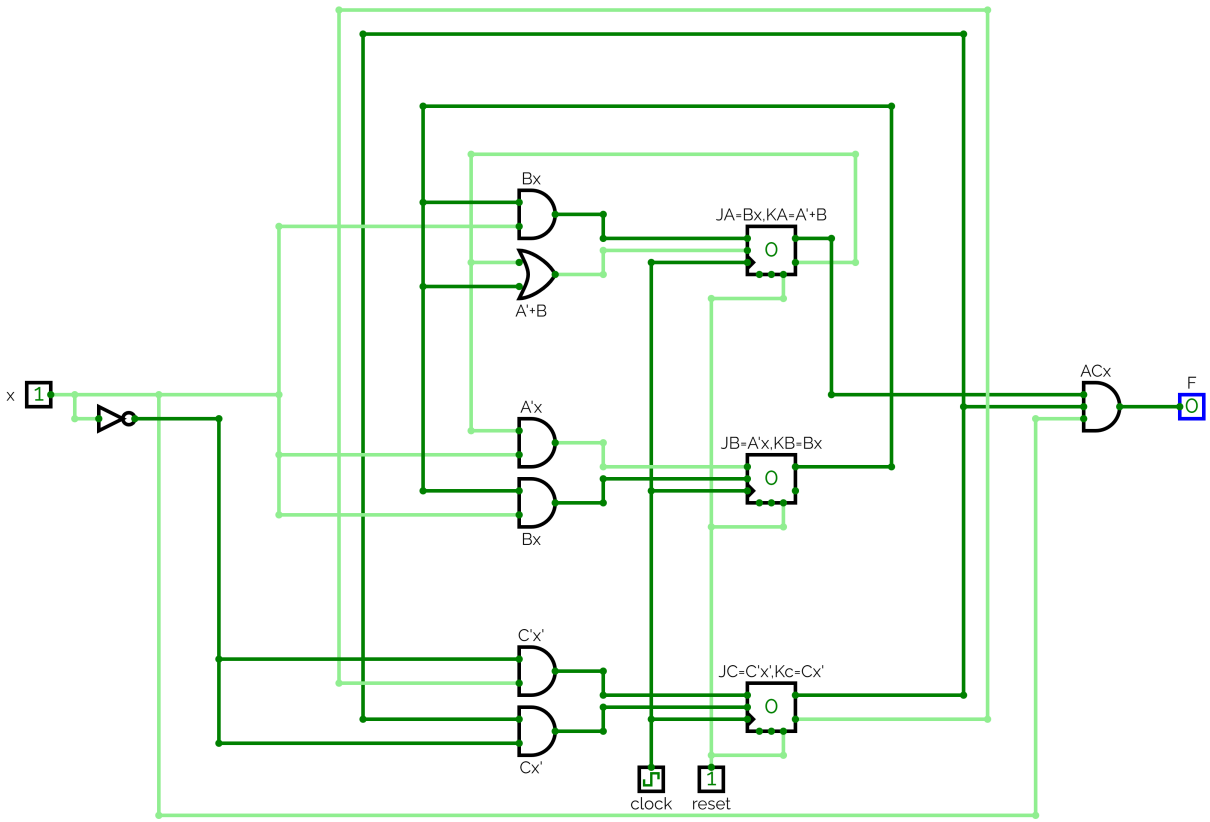

Figure 8: Design Schematic with JK Flip-Flops

12 Verilog Code Solutions with JK Flip-Flops

12.1 jkff

```
`timescale 1ns / 1ps
module jkff (input j,
               input k,
               input rst,
               input clk,
               output reg q);
    always @(posedge clk or posedge rst)
    begin
        if(rst)begin
            q <= 1'b0;
            end
        else begin
            case({j,k})
                2'b00: begin
                    q <= q;
                end
                2'b01: begin
                    q <= 0;
                end
                2'b10: begin
                    q <= 1;
                end
                2'b11: begin
                    q <= ~q;
                end
            endcase
        end
    end
endmodule
```

12.2 machine_jk

```
`timescale 1ns / 1ps
module machine_jk(
    input x,
    input rst,
    input clk,
    output F
    );
    //current_state[2] = A
    //current_state[1] = B
    //current_state[0] = C
    reg [2:0] current_state_jk = 3'b000;
    //next_state[2] = d_A
    //next_state[1] = d_B
    //next_state[0] = d_C
    wire [2:0] next_state_jk;
    jkff jk_A (
                .j(current_state_jk[1]&x),
                .k(~current_state_jk[2] | current_state_jk[1]),
                .clk(clk),
                .rst(rst),
                .q(next_state_jk[2]));
    jkff jk_B (
                .j(~current_state_jk[2]&x),
                .k(current_state_jk[1]&x),
                .clk(clk),
                .rst(rst),
                .q(next_state_jk[1]));
    jkff jk_C (
                .j(~current_state_jk[0]&~x),
                .k(current_state_jk[0]&~x),
                .clk(clk),
                .rst(rst),
                .q(next_state_jk[0]));
    always @(rst or next_state_jk) begin
        if(rst) begin current_state_jk <= 3'b000; end
        else begin current_state_jk <= next_state_jk; end
    end
    assign F = (current_state_jk[2]&current_state_jk[0]);
endmodule
```

12.3 machine_jk_tb

```
‘timescale 1ns / 1ps
module machine_jk_tb;
    reg x;
    reg rst;
    reg clk;
    wire F;
    machine_jk uut(.x(x), .rst(rst), .clk(clk), .F(F));
    reg[19:0] input_data_2;
    integer shift_amount_2;
    initial begin
        input_data_2=20’b11110000000111000100;
        shift_amount_2=0;
        x=0; rst=1; #22;
        rst=1; #5;
        rst=0; #98;
        rst=1; #2;
        rst=0; #93;
        $finish;
    end
    initial begin
        clk=1;
        forever begin
            #5;
            clk=~clk;
        end
    end
    always @(posedge clk or posedge rst) begin
        #2;x=input_data_2>>shift_amount_2;
        shift_amount_2=shift_amount_2+1;
    end
endmodule
```

13 Waveform of The Implementation with JK Flip-Flops

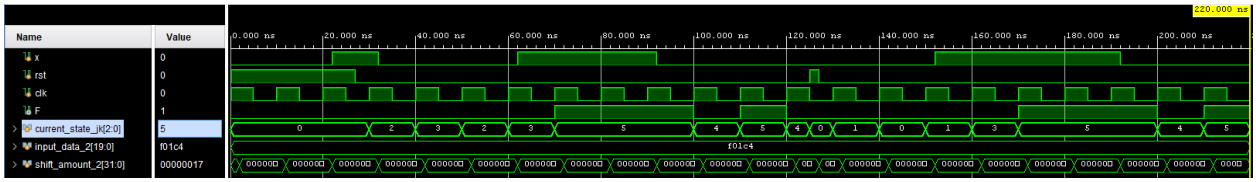

Figure 9: Waveform of machine_jk

14 Results for The Implementation with D Flip-Flops

As you can see below; JK flip-flops that assigns the values of states are triggered on the rising edges of the both clk and the rst signals. If a reset occurs, state goes back to the initial state. current_state values and output change according to the characteristic equations of the JK flip-flops such that jA=Bx, kA=A'+B, jB=A'x, kB=Bx, jC=C'x', kC=Cx', F=AC.

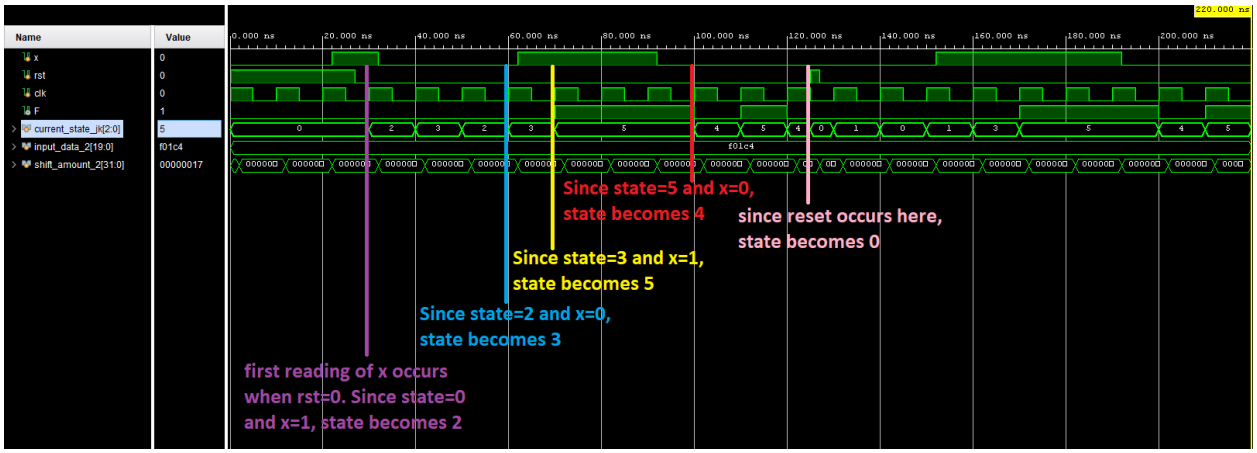

Figure 10: Waveform of machine_jk with Explanations

References

- Reference for Verilog Code for Rising Edge D Flip-Flop with Asynchronous Reset High Level
- Reference for Verilog Code for JK Flip-Flop