Hacettepe University

Computer Engineering Department

BM233 Logic Design Lab - 2020 Fall

Experiment 5 - Sequential Circuits in Verilog

December 26, 2020

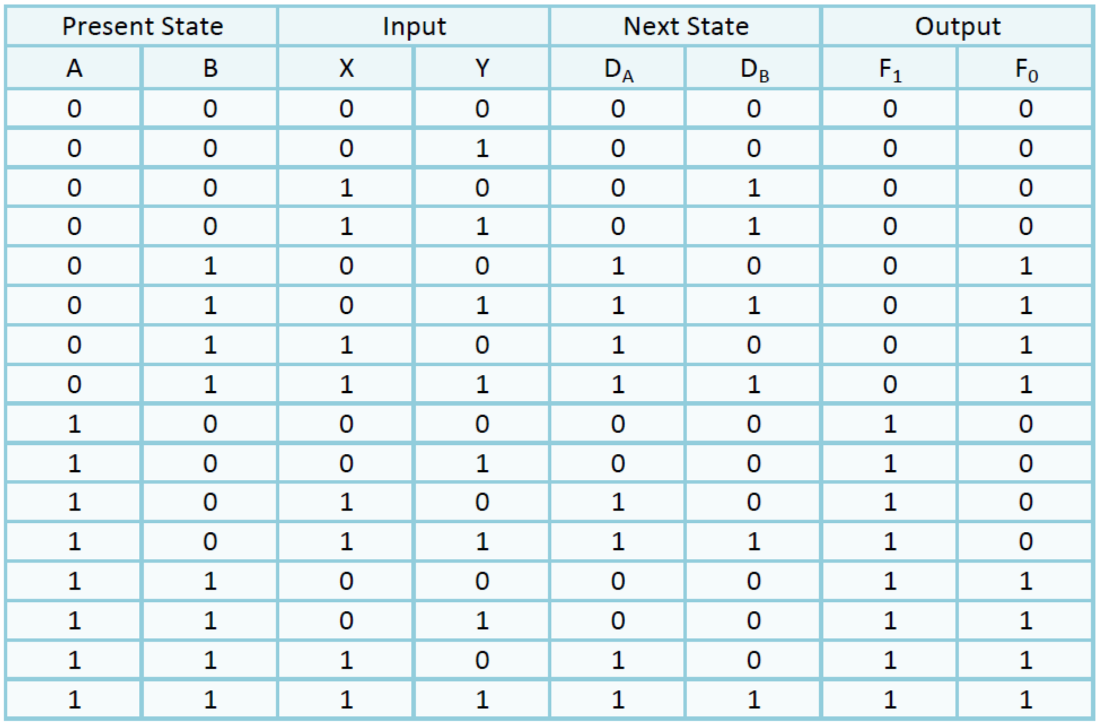
Student name:
Deniz Gönenç

Student Number:
b21946146

1 Problem Definition

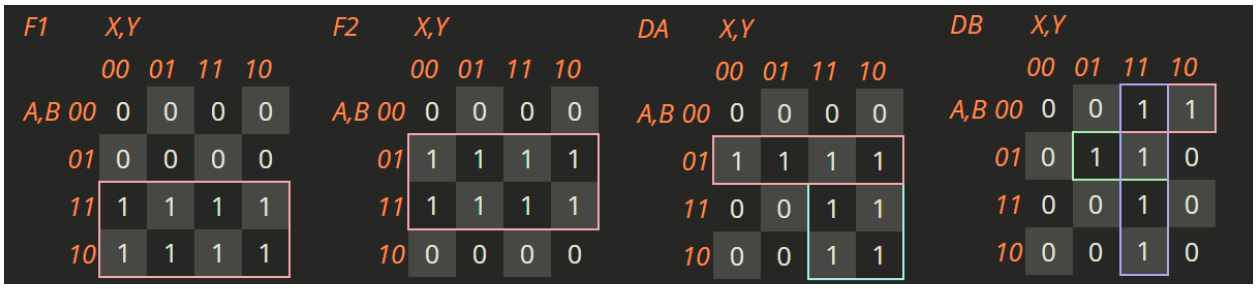
Designing and simulating a sequential circuit that uses D flip flops in Verilog HDL.

2 State Transition Table



Present State		Input		Next State		Output	
A	B	X	Y	D_A	D_B	F_1	F_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	1	0	0
0	0	1	1	0	1	0	0
0	1	0	0	1	0	0	1
0	1	0	1	1	1	0	1
0	1	1	0	1	0	0	1
0	1	1	1	1	1	0	1
1	0	0	0	0	0	1	0
1	0	0	1	0	0	1	0
1	0	1	0	1	0	1	0
1	0	1	1	1	1	1	0
1	1	0	0	0	0	1	1
1	1	0	1	0	0	1	1
1	1	1	0	1	0	1	1
1	1	1	1	1	1	1	1

3 K-Maps



F1	X,Y			
	00	01	11	10
A,B 00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

F2	X,Y			
	00	01	11	10
A,B 00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

DA	X,Y			
	00	01	11	10
A,B 00	0	0	0	0
01	1	1	1	1
11	0	0	1	1
10	0	0	1	1

DB	X,Y			
	00	01	11	10
A,B 00	0	0	1	1
01	0	1	1	0
11	0	0	1	0
10	0	0	1	0

4 Input Output Equations

We will use 2 D flip flops.

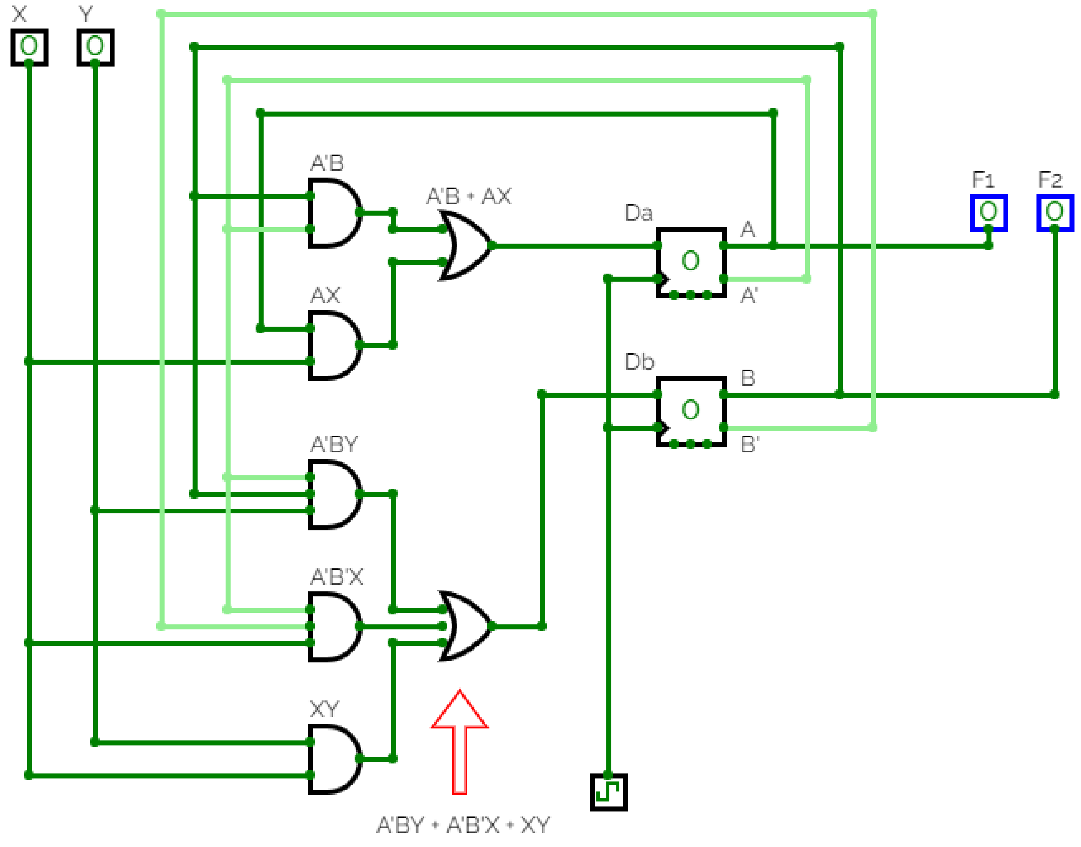
$$D_A = AX + A'B$$

$$D_B = A'BY + A'B'X + XY$$

$$F_1 = A$$

$$F_0 = B$$

5 Circuit Diagram



6 D Flip Flop Implementation

```
`timescale 1ns / 1ps

module Dflipflop(Q, Q_inv, D, clock, reset);
    input D, clock, reset;
    output reg Q, Q_inv;
    always @(posedge clock or posedge reset)
    if (reset) begin
        Q <= 0;
        Q_inv <= 1; end
    else begin
        Q <= D;
        Q_inv <= ~D;
    end

endmodule
```

7 Controller Implementation

```
`timescale 1ns / 1ps

`include "Dflipflop.v"

module controller(X, Y, F1, F0, clock, reset);
    input X, Y, clock, reset;
    output F1, F0;
    wire q_a, q_a_inv, q_b, q_b_inv, and_1, and_2, and_3, and_4, and_5, or_1, or_2;

    Dflipflop upper_ff(q_a, q_a_inv, or_1, clock, reset);
    Dflipflop lower_ff(q_b, q_b_inv, or_2, clock, reset);

    and(and_1, q_b, q_a_inv);
    and(and_2, q_a, X);
    or(or_1, and_1, and_2);
    and(and_3, q_a_inv, q_b, Y);
    and(and_4, q_a_inv, q_b_inv, X);
    and(and_5, X, Y);
    or(or_2, and_3, and_4, and_5);
    assign F1 = q_a;
    assign F0 = q_b;
endmodule
```

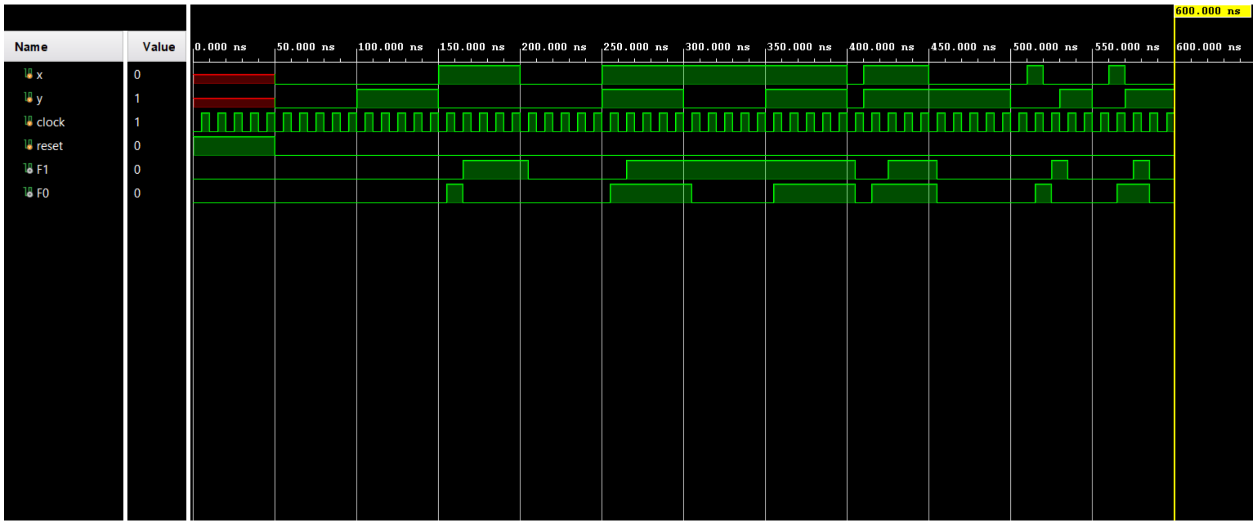
8 Testbench Implementation

Test for all 16 state changes.

```
`timescale 1ns / 1ps

`include "controller.v"

module controller_tb();
    reg x, y, clock, reset;
    wire F1, F0;
    controller UUT(.X(x), .Y(y), .F1(F1), .F0(F0), .clock(clock), .reset(reset));

    always begin
        # 5 clock = 1;
        # 5 clock = 0;
    end

    initial begin
    reset = 1; clock = 0;
    #50 reset = 0; x = 0; y = 0;
    #50 y = 1;
    #50 x = 1; y = 0;
    #50 x = 0; y = 0;
    #50 x = 1; y = 1;
    #50 x = 1; y = 0;
    #50 x = 1; y = 1;
    #50 x = 0; y = 0;
    #10 x = 1; y = 1;
    #40 x = 0;
    #50 y = 0;
    #10 x = 1;
    #10 x = 0;
    #10 y = 1;
    #20 x = 0; y = 0;
    #10 x = 1;
    #10 x = 0; y = 1;

    #30 $finish;
    end

endmodule
```

9 Results

9.



Resulting Waveform

The state changes in the order they happen in the waveform:
$00 \rightarrow 00(input : 00)$
$00 \rightarrow 00(input : 01)$
$00 \rightarrow 01(input : 10)$
$01 \rightarrow 10(input : 10)$

$10 \rightarrow 10(input : 10)$
$10 \rightarrow 00(input : 00)$
$00 \rightarrow 10(input : 11)$
$01 \rightarrow 11(input : 11)$

$11 \rightarrow 10(input : 10)$
$10 \rightarrow 11(input : 11)$
$11 \rightarrow 00(input : 00)$
$11 \rightarrow 11(input : 11)$

$11 \rightarrow 00(input : 01)$
$01 \rightarrow 10(input : 00)$
$10 \rightarrow 00(input : 01)$
$01 \rightarrow 11(input : 01)$

My design is working correctly because my results are parallel to the state diagram.