Hacettepe University

Computer Engineering Department

BM233 Logic Design Lab - 2022 Fall

Sequential Circuits in Verilog

December 30, 2022

Student name:
Eren Gül

Student Number:
b2200356037

1 Problem Definition

For this experiment, we are supposed to design a Binary/Gray Code Counter circuit using Verilog. The circuit should be designed to count up in binary or gray code depending on the 1-bit mode input variable, as illustrated in the state diagram below. If mode is 0, it should count in natural binary, otherwise it should count in gray code. The circuit we design should also have another 1-bit input variable reset, which resets the circuit state to the start state 000.

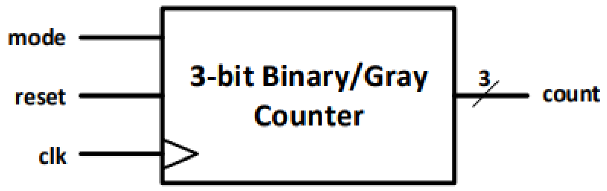


Figure 1: 3-bit Binary/Gray Counter

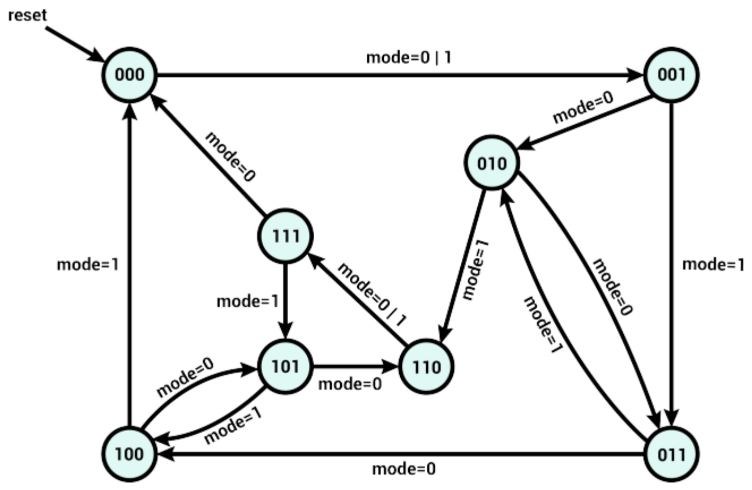
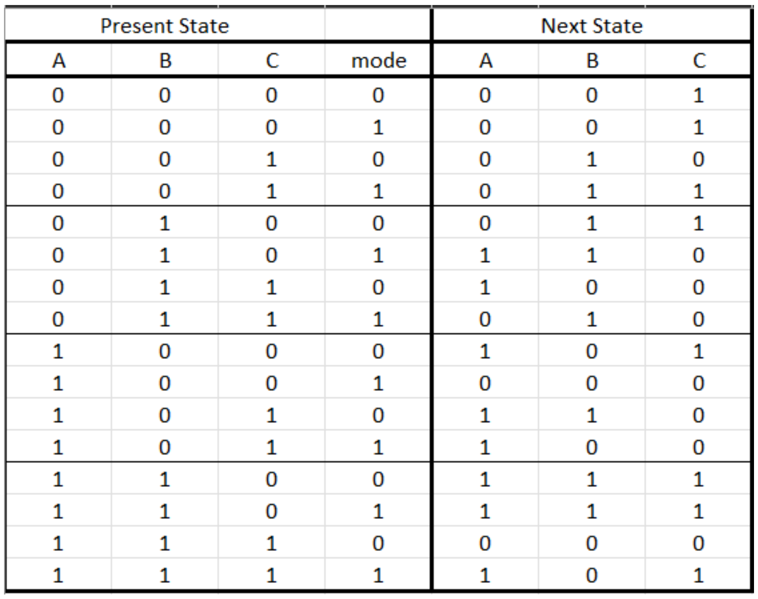


Figure 2: Counter State Diagram

2 Solution Implementation

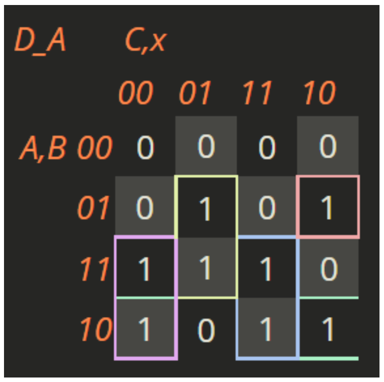
The state transition table is given below.



Present State				Next State		
A	B	C	mode	A	B	C
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	1	1
0	1	0	1	1	1	0
0	1	1	0	1	0	0
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	0	1	0	0	0
1	0	1	0	1	1	0
1	0	1	1	1	0	0
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	0	0	0
1	1	1	1	1	0	1

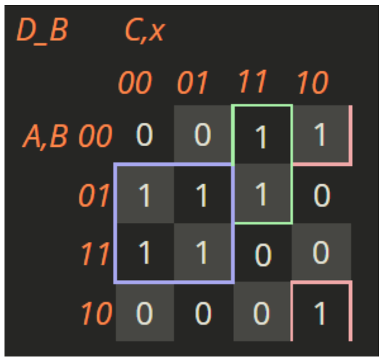
Figure 3: Counter State Table

For the first part of this experiment, we are going to need 3 D flip flops. The K-maps and input equations are given below for each one of them. (Renamed mode to x)



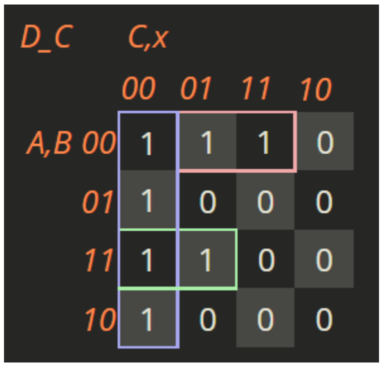
D_A	C,x 00	01	11	10
A,B 00	0	0	0	0
01	0	1	0	1
11	1	1	1	0
10	1	0	1	1

Figure 4: D_A K-map



D_B	C,x 00	01	11	10
A,B 00	0	0	1	1
01	1	1	1	0
11	1	1	0	0
10	0	0	0	1

Figure 5: D_B K-map



D_C	C,x 00	01	11	10
A,B 00	1	1	1	0
01	1	0	0	0
11	1	1	0	0
10	1	0	0	0

Figure 6: D_C K-map

$$D_A = BC'x + A'BCx' + AB'x' + ACx + AC'x'$$
$$D_B = BC' + B'Cx' + A'Cx$$
$$D_C = A'B'x + C'x' + ABx$$

The design schematic using D flip flops is given below.

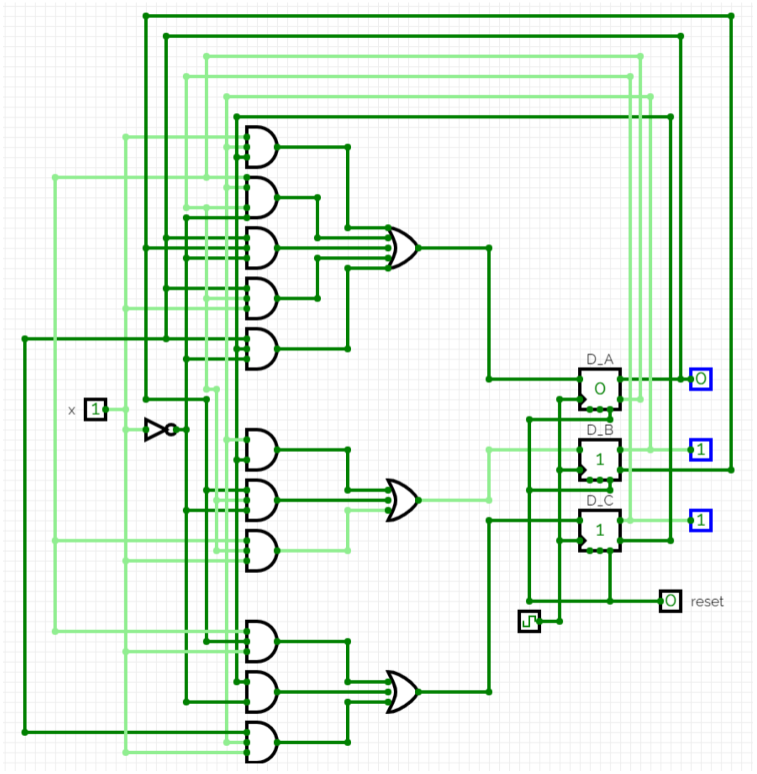
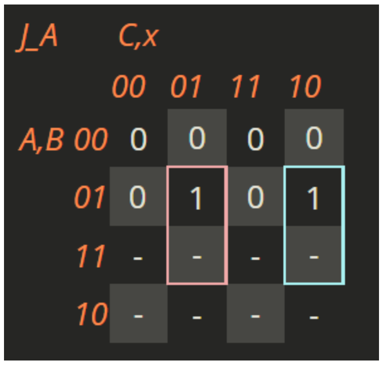


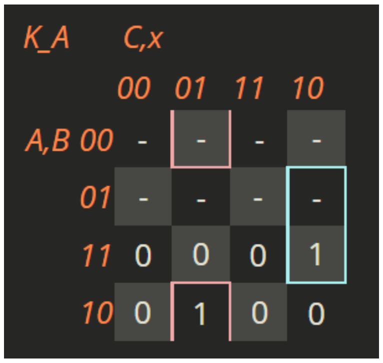
Figure 7: Design Schematic that uses D flip-flops

For the second part of this experiment, we are going to need 3 JK flip flops. The K-maps and input equations are given next page for each one of them.



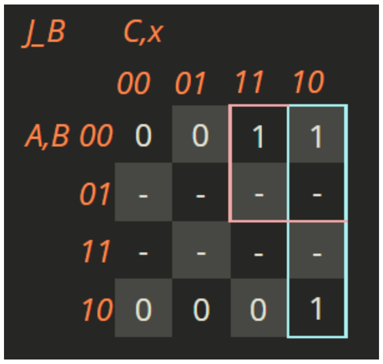
J_A	C,x			
	00	01	11	10
A,B 00	0	0	0	0
01	0	1	0	1
11	-	-	-	-
10	-	-	-	-

Figure 8: J_A K-map



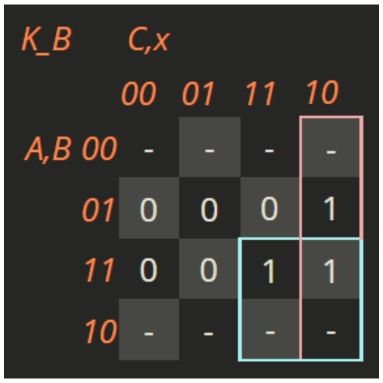
K_A	C,x			
	00	01	11	10
A,B 00	-	-	-	-
01	-	-	-	-
11	0	0	0	1
10	0	1	0	0

Figure 9: K_A K-map



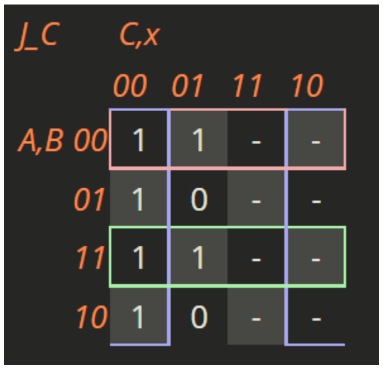
J_B	C,x			
	00	01	11	10
A,B 00	0	0	1	1
01	-	-	-	-
11	-	-	-	-
10	0	0	0	1

Figure 10: J_B K-map



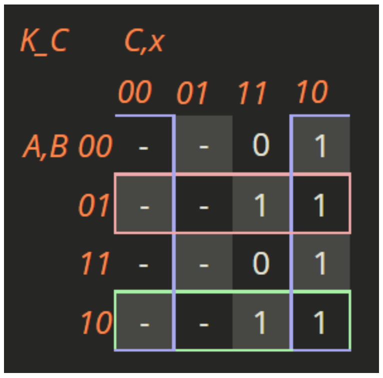
K_B	C,x			
	00	01	11	10
A,B 00	-	-	-	-
01	0	0	0	1
11	0	0	1	1
10	-	-	-	-

Figure 11: K_B K-map



J_C	C,x			
	00	01	11	10
A,B 00	1	1	-	-
01	1	0	-	-
11	1	1	-	-
10	1	0	-	-

Figure 12: J_C K-map



K_C	C,x			
	00	01	11	10
A,B 00	-	-	0	1
01	-	-	1	1
11	-	-	0	1
10	-	-	1	1

Figure 13: K_C K-map

$$J_A = BC'x + BCx'$$
$$K_A = B'C'x + BCx'$$
$$J_B = A'C + Cx'$$
$$K_B = Cx' + AC$$
$$J_C = A'B' + x' + AB$$
$$K_C = x' + A'B + AB'$$

The design schematic using JK flip flops is given below.

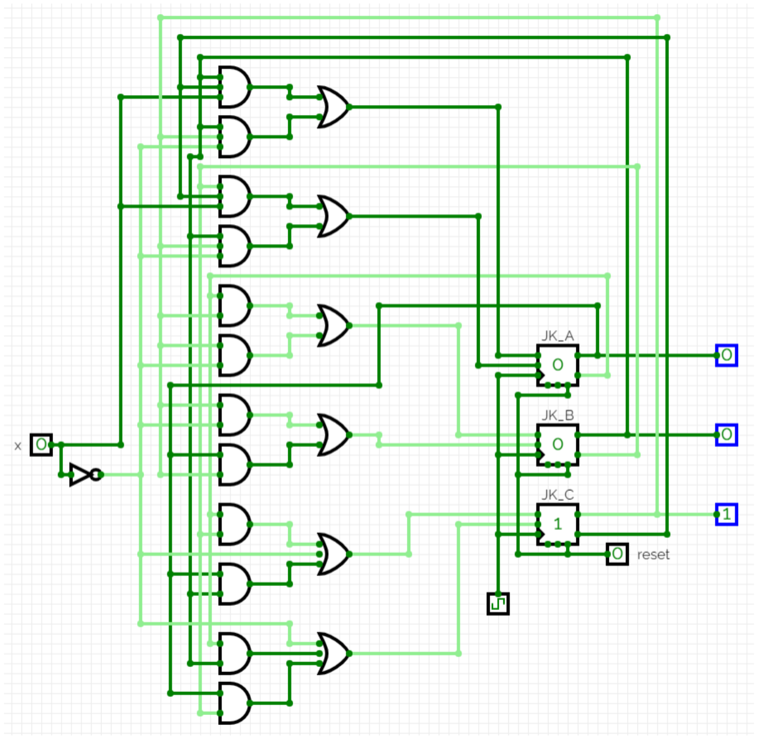


Figure 14: Design Schematic that uses JK flip-flops

Verilog code solutions for all the specified modules are given below.

```verilog
module dff_sync_res(D, clk, sync_reset, Q);
    input D;
    input clk;
    input sync_reset;
    output reg Q;

    // Change state at positive edge of clock
    always @(posedge clk)
    begin
        if(sync_reset==1'b1)
            Q <= 1'b0;
        else
            Q <= D;
    end

endmodule
```

```verilog
module jk_sync_res(J, K, clk, sync_reset, Q);
    input J;
    input K;
    input clk;
    input sync_reset;
    output reg Q;

    // Change state at positive edge of clock
    always @(posedge clk)
    begin
        if(sync_reset==1'b1)
            Q <= 1'b0;
        else
            case ({J,K})
                2'b00 : Q <= Q;
                2'b01 : Q <= 1'b0;
                2'b10 : Q <= 1'b1;
                2'b11 : Q <= ~Q;
            endcase
    end

endmodule
```

```verilog
module counter_d(input reset, input clk, input mode, output [2:0] count);

    // present_state[2] = A
    // present_state[1] = B
    // present_state[0] = C
    reg [2:0] present_state = 2'b00;
    // next_state[2] = D_A
    // next_state[1] = D_B
    // next_state[0] = D_C
    wire [2:0] next_state;

    // Used D flip flops in this part for structural design with explicit
       association
    dff_sync_res D_A(.D((present_state[1] & ~present_state[0] & mode) | (~
       present_state[2] & present_state[1] & present_state[0] & ~mode) | (
       present_state[2] & ~present_state[1] & ~mode) | (present_state[2] &
        present_state[0] & mode) | (present_state[2] & ~present_state[0] &
        ~mode)), .clk(clk), .sync_reset(reset), .Q(next_state[2]));

    dff_sync_res D_B(.D((present_state[1] & ~present_state[0]) | (~
       present_state[1] & present_state[0] & ~mode) | (~present_state[2] &
        present_state[0] & mode)), .clk(clk), .sync_reset(reset), .Q(
       next_state[1]));

    dff_sync_res D_C(.D((~present_state[2] & ~present_state[1] & mode) |
       (~present_state[0] & ~mode) | (present_state[2] & present_state[1]
       & mode)), .clk(clk), .sync_reset(reset), .Q(next_state[0]));

    // Whenever there is a next state, update present state
    always @(next_state) begin
        present_state <= next_state;
    end

    // Assign present state to output count
    assign count[2] = present_state[2];
    assign count[1] = present_state[1];
    assign count[0] = present_state[0];

endmodule
```

```
module counter_jk(input reset, input clk, input mode, output [2:0] count);

    // present_state[2] = A
    // present_state[1] = B
    // present_state[0] = C
    reg [2:0] present_state = 2'b00;
    // next_state[2] = JK_A
    // next_state[1] = JK_B
    // next_state[0] = JK_C
    wire [2:0] next_state;

    // Used JK flip flops in this part for structural design with explicit
        association
    jk_sync_res JK_A(.J((present_state[1] & ~present_state[0] & mode) | (
        present_state[1] & present_state[0] & ~mode)), .K((~present_state[1
        ] & ~present_state[0] & mode) | (present_state[1] & present_state[0
        ] & ~mode)), .clk(clk), .sync_reset(reset), .Q(next_state[2]));

    jk_sync_res JK_B(.J((~present_state[2] & present_state[0]) | (
        present_state[0] & ~mode)), .K((present_state[0] & ~mode) | (
        present_state[2] & present_state[0])), .clk(clk), .sync_reset(reset
        ), .Q(next_state[1]));

    jk_sync_res JK_C(.J((~present_state[2] & ~present_state[1]) | (~mode)
        | (present_state[2] & present_state[1])), .K((~mode) | (~
        present_state[2] & present_state[1]) | (present_state[2] & ~
        present_state[1])), .clk(clk), .sync_reset(reset), .Q(next_state[0]
        ));

    // Whenever there is a next state, update present state
    always @(next_state) begin
        present_state <= next_state;
    end

    // Assign present state to output count
    assign count[2] = present_state[2];
    assign count[1] = present_state[1];
    assign count[0] = present_state[0];

endmodule
```

3 Testbench Implementation

```
`timescale 1ns/1ps

module counter_tb;
    reg reset, clk, mode; // Inputs
    wire [2:0] count; // Output
    integer i; // Integer for the for loop

    reg counter = 1'b0;

    // Instantiate UUTs
    counter_d uut(reset, clk, mode, count);
    counter_jk c1(reset, clk, mode, count);

    // Increment mode by 1
    initial begin
        for (i = 0; i < 2; i++) begin
            mode = counter;
            counter += 1;
            #122;
        end
        $finish;
    end

    // Update reset values
    initial begin
        $dumpvars;
        reset <= 1; #22;
        reset <= 0; #200;
        reset <= 1; #20;
        $finish;
    end

    initial begin // Generate clock
        clk = 0;
        forever begin
            #5;
            clk = ~clk; // Change every 10ns
        end
    end

endmodule
```

4 Results

The obtained waveforms are given below.

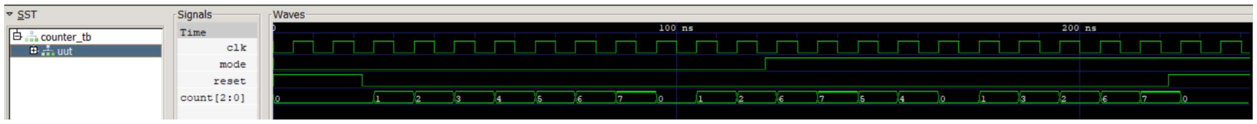


Figure 15: Obtained Waveform for the First Part

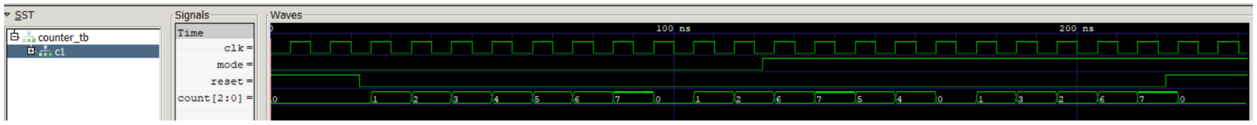


Figure 16: Obtained Waveform for the Second Part

From results, we see that at every positive edge of clock, the states change. When mode is 0, the circuit counts up in natural binary and when the mode is 1, the circuit counts up in gray code. If reset is 1, the state becomes 000.

References

- https://cdn-uploads.piazza.com/paste/itmvemweb267cd/1e748d391563184df77907a5a81401e0692b
  VerilogIntro.pdf
- https://cdn-uploads.piazza.com/paste/itmvemweb267cd/2e69727151844bf5c4accaa1d315bcfd4a98
  More_Verilog_Examples_of_Sequential_circuits.pdf
- https://piazza.com/class/l8n34kf3w15df/post/78