Hacettepe University

Computer Engineering Department

BM233 Logic Design Lab - 2022 Fall

Verilog Assignment 2

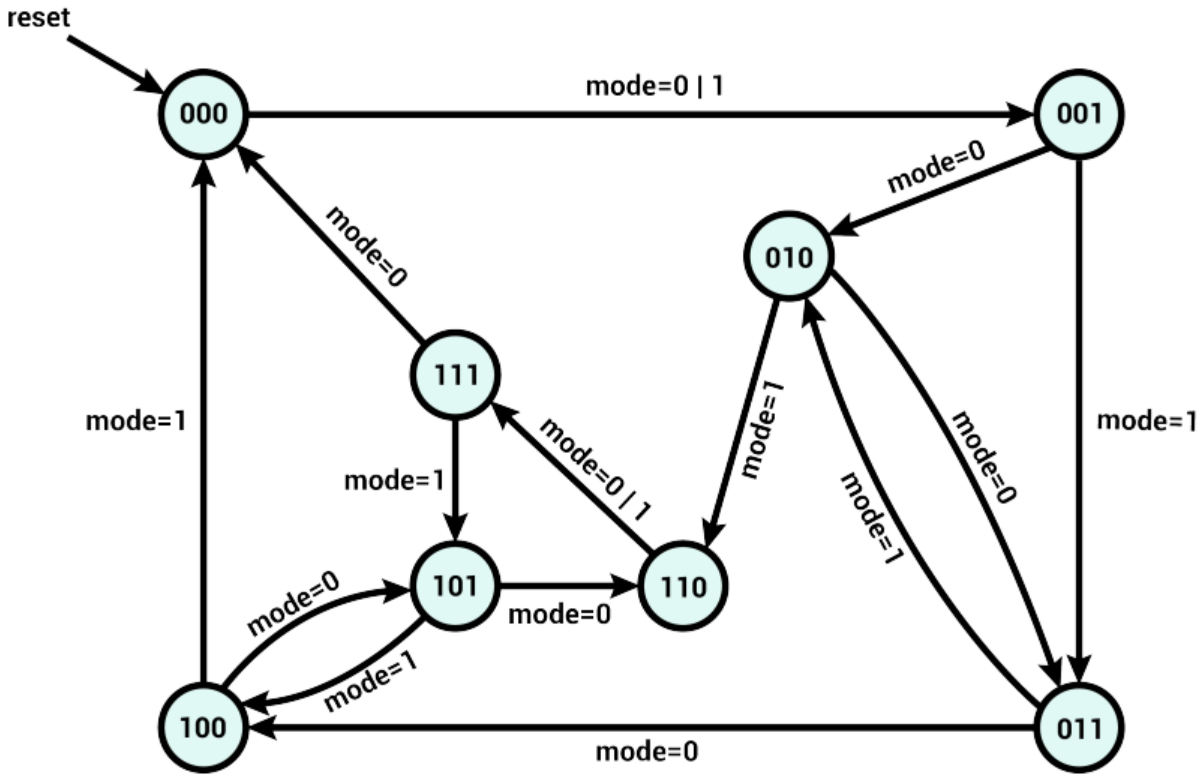
January 1, 2023

Student name:
Zeynep YEŞILKAYA

Student Number:
b2210356048

1 Problem Definition

A Binary/Gray Code Counter circuit is to be designed using Verilog for this experiment. The 1-bit mode input variable will determine whether the circuit counts up in binary or gray code. When the mode is 0, the natural binary code should be used for counting, and when it is 1, the gray code should be used. In addition, the circuit should have a 1-bit input variable called reset, which will reset the circuit state to the start state of 000 when activated.



2 Solution Implementation

Firstly, a truth table was created using the given diagram. Using this truth table, k-maps were drawn. Using k-maps, the equations of D type and JK type flip-flops were found.

A	B	C	mode	A'	B'	C'	D_A	D_B	D_C
0	0	0	0	0	0	1	0	0	1
0	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	0
0	0	1	1	0	1	1	0	1	1
0	1	0	0	0	1	1	0	1	1
0	1	0	1	1	1	0	1	1	0
0	1	1	0	1	0	0	1	0	0
0	1	1	1	0	1	0	0	1	0
1	0	0	0	1	0	1	1	0	1
1	0	0	1	0	0	0	0	0	0
1	0	1	0	1	1	0	1	1	0
1	0	1	1	1	0	0	1	0	0
1	1	0	0	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0	0
1	1	1	1	1	0	1	1	0	1

Figure 1: D Type Flip-Flop Truth Table

A	B	C	mode	A'	B'	C'	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	0	1	0	X	0	X	1	X
0	0	0	1	0	0	1	0	X	0	X	1	X
0	0	1	0	0	1	0	0	X	1	X	X	1
0	0	1	1	0	1	1	0	X	1	X	X	0
0	1	0	0	0	1	1	0	X	X	0	1	X
0	1	0	1	1	1	0	1	X	X	0	0	X
0	1	1	0	1	0	0	1	X	X	1	X	1
0	1	1	1	0	1	0	0	X	X	0	X	1
1	0	0	0	1	0	1	X	0	0	X	1	X
1	0	0	1	0	0	0	X	1	0	X	0	X
1	0	1	0	1	1	0	X	0	1	X	X	1
1	0	1	1	1	0	0	X	0	0	X	X	1
1	1	0	0	1	1	1	X	0	X	0	1	X
1	1	0	1	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	0	X	1	X	1	X	1
1	1	1	1	1	0	1	X	0	X	1	X	0

Figure 2: JK Type Flip-Flop Truth Table

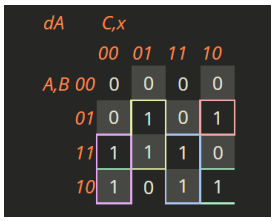


Figure 3: DA K-Map

Figure 4: DB K-Map

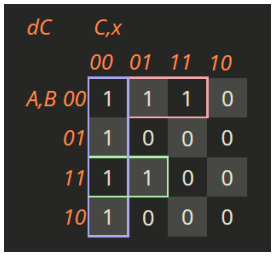


Figure 5: DC K-Map

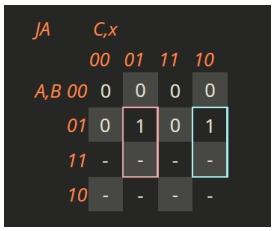


Figure 6: JA K-Map

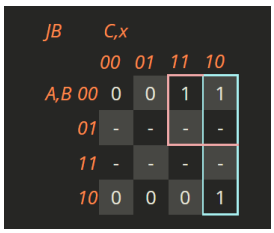


Figure 7: JB K-Map

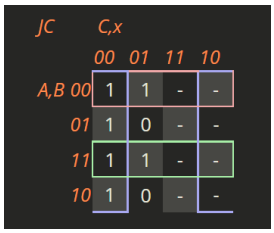


Figure 8: JC K-Map

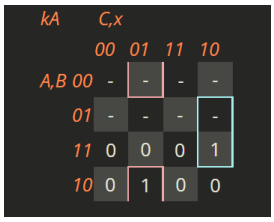


Figure 9: KA K-Map

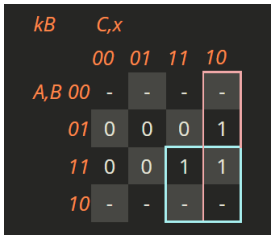


Figure 10: KB K-Map

Figure 11: KC K-Map

```
1
2  module counter_d(input reset, input clk, input mode, output [2:0] count);
3
4  // D flip-flop for bit 2
5  // The D input of the flip-flop is determined by several conditions
6  // depending on the values of count[1], count[0], and mode
7  dff_sync_res dff0(.D((count[1] & ~count[0] & mode)
8                       | (~count[2] & count[1] & count[0] & ~mode)
9                       | (count[2] & ~count[1] & ~mode)
10                      | (count[2] & count[0] & mode)
11                      | (count[2] & ~count[0] & ~mode)), .clk(clk),
12                      .sync_reset(reset), .Q(count[2]));
13
14 // D flip-flop for bit 1
15 // The D input of the flip-flop is determined by several conditions
16 // depending on the values of count[1], count[0], and mode
17 dff_sync_res dff1(.D((count[1] & ~count[0]) | (~count[1] & count[0] & ~mode)
18                   | (~count[2] & count[0] & mode)),
19                   .clk(clk), .sync_reset(reset), .Q(count[1]));
20
21 // D flip-flop for bit 0
22 // The D input of the flip-flop is determined by several conditions
23 // depending on the values of count[2], count[1], and mode
24 dff_sync_res dff2(.D((~count[2] & ~count[1] & mode) | (~count[0] & ~mode)
25                   |(count[2] & count[1] & mode)),
26                   .clk(clk), .sync_reset(reset), .Q(count[0]));
27 endmodule
```

```
1
2  module counter_jk(input reset, input clk, input mode, output [2:0] count);
3
4      // JK flip-flop for bit 2
5      // The J and K inputs of the flip-flop are determined by several conditions
6      // depending on the values of count[1], count[0], and mode
7      jk_sync_res jk0(.J((count[1] & ~count[0] & mode) | (count[1] &count[0]&~mode)),
8                      .K((~count[1] & ~count[0] & mode) |
9                      (count[1] & count[0] & ~mode)),
10                     .clk(clk), .sync_reset(reset), .Q(count[2]));
11
12     // JK flip-flop for bit 1
13     // The J and K inputs of the flip-flop are determined by several conditions
14     // depending on the values of count[0] and mode, and count[2]
15     jk_sync_res jk1(.J((count[0] & ~mode) | (~count[2] & count[0])), .K((count[0]
16     & ~mode) | (count[2] & count[0])),
17                     .clk(clk), .sync_reset(reset), .Q(count[1]));
18
19     // JK flip-flop for bit 0
20     // The J and K inputs of the flip-flop are determined by several conditions
```

```
    // depending on the values of count[2], count[1], and mode
    jk_sync_res jk2(.J((~count[2] & ~count[1]) | (~mode) | (count[2] & count[1])),
                    .K((~mode) | (~count[2] & count[1]) | (count[2] & ~count[1])),
                    .clk(clk), .sync_reset(reset), .Q(count[0]));


endmodule
```

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

    always@(posedge clk)
    begin
        if(sync_reset == 1'b1)
            Q<=1'b0;
        else
            Q <= D;
    end
endmodule
```

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

always @(posedge clk) begin
// If sync_reset is high, reset Q to 0
    if (sync_reset == 1'b1) begin
        Q <= 1'b0;
    end
     // Otherwise, update the value of Q based on the J and K input
    else begin
    case ({J,K})
        2'b00 :  Q <= Q; // No change
        2'b01 :  Q <= 0; // Reset
        2'b10 :  Q <= 1; // Set
        2'b11 :  Q <= ~Q; // Toggle
    endcase
    end
end

endmodule
```

3 Testbench Implementation

State for which cases you are testing and their meaning. Example how to add Verilog code:

```
`timescale 1ns/1ps

module counter_tb;
    reg reset, clk, mode;
    reg[19:0] input_data;
    wire [2:0] count;
    integer i;

    //Comment the next line out when testing your JK flip flop implementation.
    //counter_d uut(reset, clk, mode, count);
    // Uncomment the next line to test your JK flip flop implementation.
    counter_jk c1(reset, clk, mode, count);

    initial begin
        // Set dumpfile for simulation waveform output
        $dumpfile("result.vcd");
         // Enable dumping of variables
        $dumpvars;
        // Set input data
        input_data = 20'b11111111110000000000;
        i = 0;
        // Set reset to 1 and wait 30 time units
        reset = 1;#30;
        // Set reset to 0 and wait 400 time units
        reset = 0;#400;
        // Set reset to 1 and wait 30 time units
        reset = 1;#30;
        $finish;

    end

    initial begin

        // Generate clock
        clk = 0;
         // Generate clock indefinitely
        forever begin
            // Wait 10 time units
            #10;
            // Toggle clock
            clk = ~clk;
        end
    end

```

```
45      always@(posedge clk) begin
46          #1;
47          // Assign mode value from input_data
48          mode = input_data >> i;
49          // Increment loop variable
50          i = i+1;
51      end
52
53  endmodule
```

4 Results

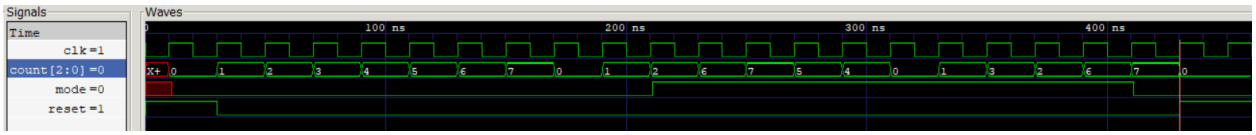


Figure 12: Resulting Waveform For JK Flip-Flop

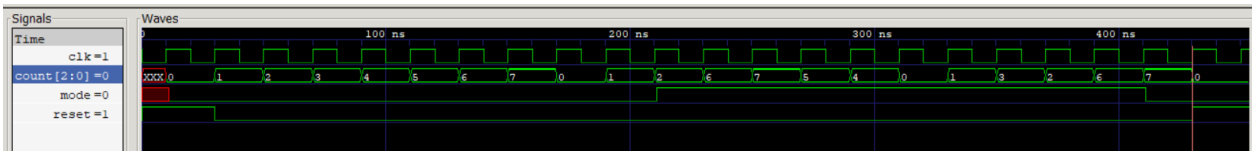


Figure 13: Resulting Waveform For D Flip-Flop

Firstly, because mode is equal to zero, it is counted according to the natural binary code.For example, 1,2,3,4,5,6,7,0,1... Then, since the mode is equal to one, it is counted according to the gray code. For example 0,1,3,2,6,7,5,4...

Decimal	Gray Code	Natural Binary
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

References

- https://piazza.com/class/l8n34kf3w15df/post/78