Hacettepe University

Computer Engineering Department

BM233 Logic Design Lab - 2022 Fall

Experiment 5 - Sequential Circuits in Verilog

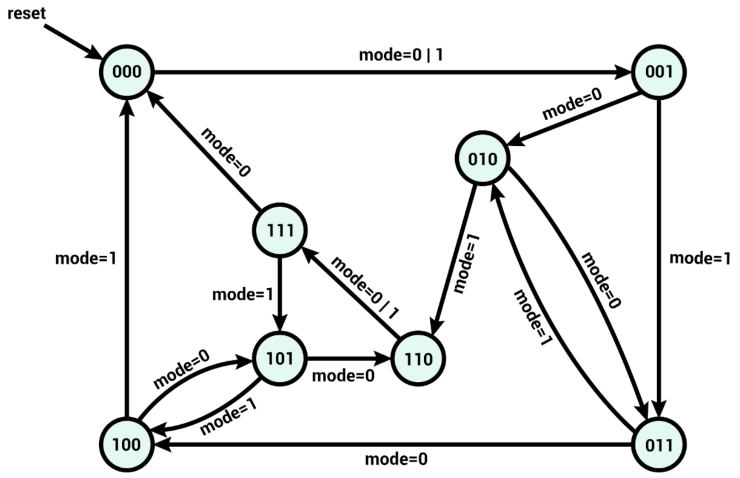
January 1, 2023

Student name:
İsmail Ulaş Ünal

Student Number:
b2200356001

1 Problem Definition

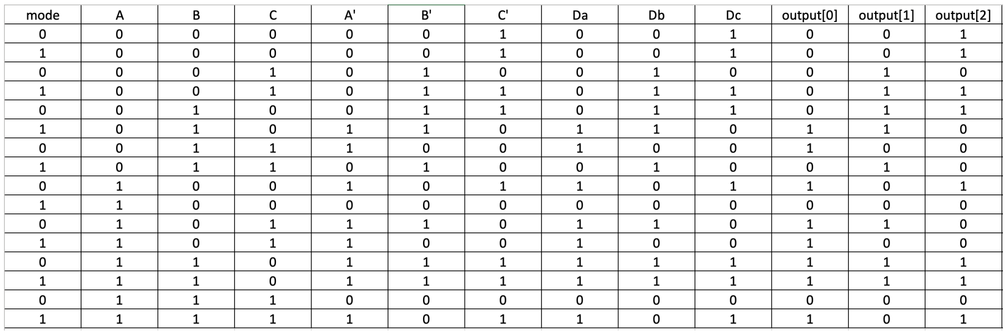
Design a Binary/Gray Code Counter circuit using Verilog. The circuit should be designed to count up in binary or gray code depending on the 1-bit mode input variable, as illustrated in the state diagram below. If mode is 0, it should count in natural binary, otherwise it should count in gray code. The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



2 Solution Implementation

firstly I created a JK flip flop and D flip flop as usual. Secondly, I draw table with state diagram for finding formulas of dA, dB, dC and jA, kA, jB, kB, jC, kC these formulas will be an input of counter d and counter jk, I used 3 flip flop because I have 8 statements $\log_2 8 = 3$

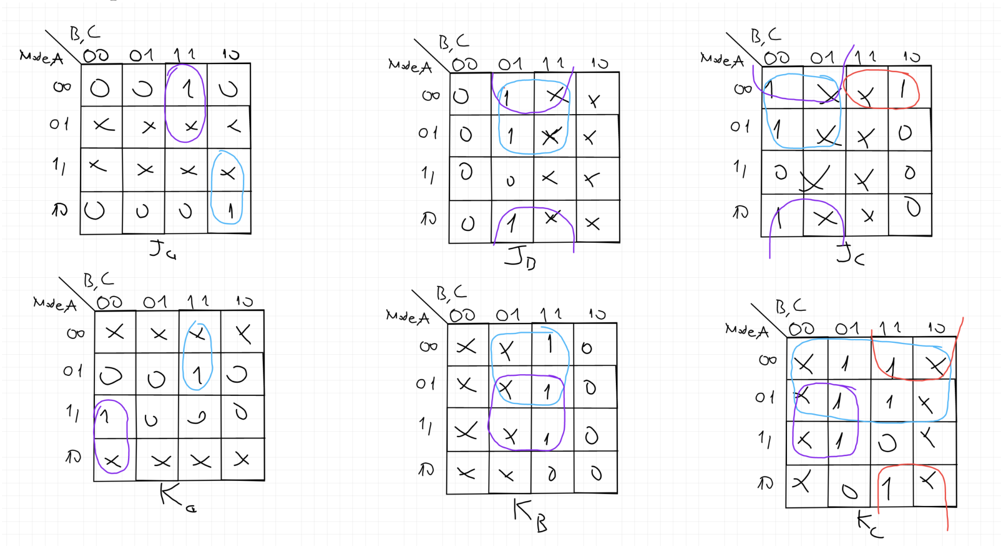
Tables

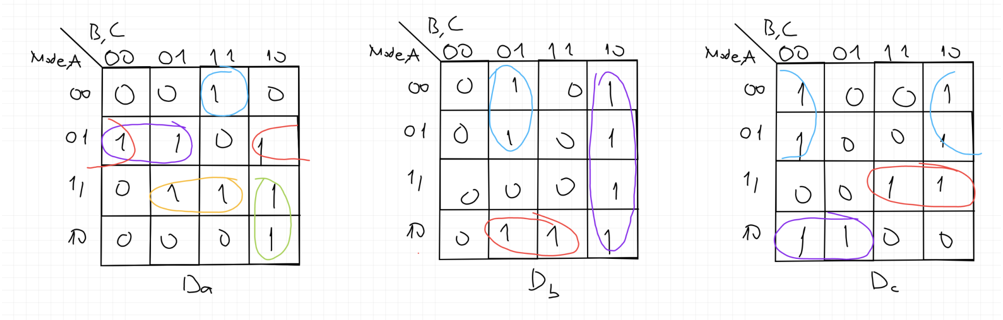


mode	A	B	C	A'	B'	C'	Da	Db	Dc	output[0]	output[1]	output[2]
0	0	0	0	0	0	1	0	0	1	0	0	1
1	0	0	0	0	0	1	0	0	1	0	0	1
0	0	0	1	0	1	0	0	1	0	0	1	0
1	0	0	1	0	1	1	0	1	1	0	1	1
0	0	1	0	0	1	1	0	1	1	0	1	1
1	0	1	0	1	1	0	1	1	0	1	1	0
0	0	1	1	1	0	0	1	0	0	1	0	0
1	0	1	1	0	1	0	0	1	0	0	1	0
0	1	0	0	1	0	1	1	0	1	1	0	1
1	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	0	1	1	0	1	1	0
1	1	0	1	1	0	0	1	0	0	1	0	0
0	1	1	0	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1
0	1	1	1	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	1	1	0	1	1	0	1

mode	A	B	C	A'	B'	C'	Ja	Ka	Jb	Kb	Jc	Kc	output[0]	output[1]	output[2]
0	0	0	0	0	0	1	0	X	0	X	1	X	0	0	1
1	0	0	0	0	0	1	0	X	0	X	1	X	0	0	1
0	0	0	1	0	1	0	0	X	1	X	X	1	0	1	0
1	0	0	1	0	1	1	0	X	1	X	X	0	0	1	1
0	0	1	0	0	1	1	0	X	X	0	1	X	0	1	1
1	0	1	0	1	1	0	1	X	X	0	0	X	1	1	0
0	0	1	1	1	0	0	1	X	X	1	X	1	1	0	0
1	0	1	1	0	1	0	0	X	X	0	X	1	0	1	0
0	1	0	0	1	0	1	X	0	0	X	1	X	1	0	1
1	1	0	0	0	0	0	X	1	0	X	0	X	0	0	0
0	1	0	1	1	1	0	X	0	1	X	X	1	1	1	0
1	1	0	1	1	0	0	X	0	0	X	X	1	1	0	0
0	1	1	0	1	1	1	X	0	X	0	1	X	1	1	1
1	1	1	0	1	1	1	X	0	X	0	1	X	1	1	1
0	1	1	1	0	0	0	X	1	X	1	X	1	0	0	0
1	1	1	1	1	0	1	X	0	X	1	X	0	1	0	1

$K - Maps$

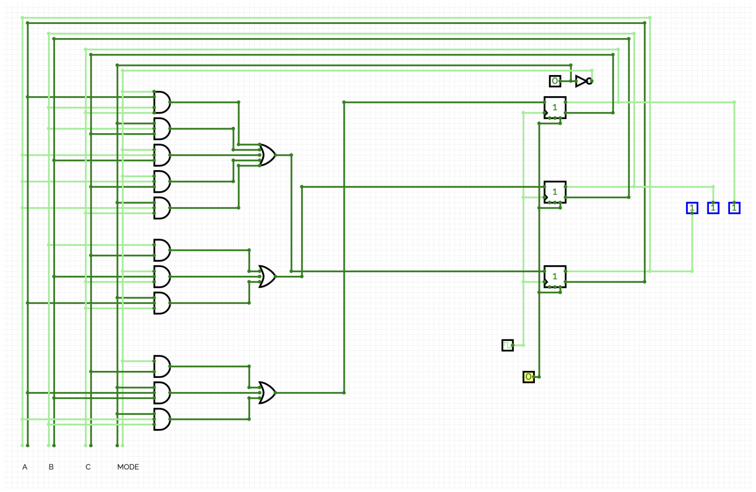


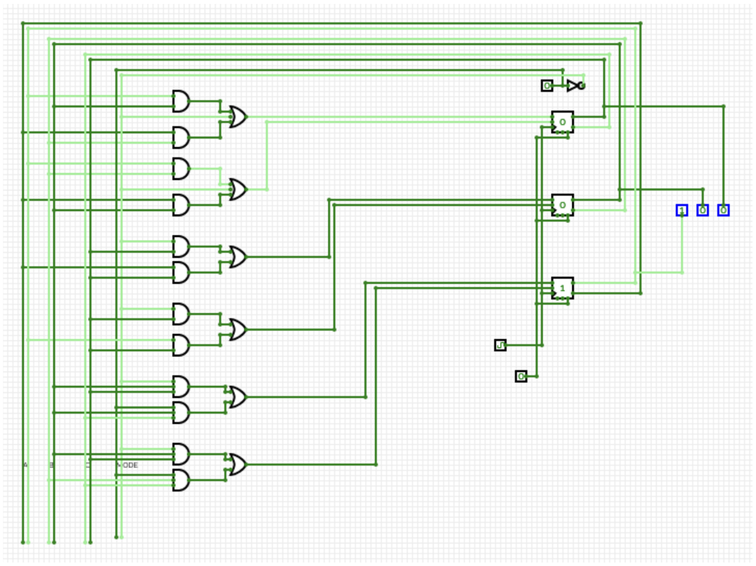


$$Da = (mode'A'BC) + (modeBC') + (mode'AB') + (mode'AC') + (modeAC)$$
$$Db = (BC') + (mode'B'C) + (modeA'C)$$
$$Dc = (mode'C') + (modeA'B') + (modeAB)$$

$Ja = (mode'BC) + (modeBC')$
$Ka = (mode'BC) + (modeB'C')$
$Jb = (mode'C) + (A'C)$
$Kb = (mode'C) + (AC)$
$Jc = (mode') + (A'B') + (AB)$
$Kc = (mode') + (A'B) + (AB')$

Circuits





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

    // Declare a flip-flop
    always @(posedge clk) begin
        if (sync_reset) begin // reset statement it assign into 0
            Q <= 1'b0;
        end else begin // if it didn't go to reset it assigns D
            Q <= D;
        end
    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;

    // Declare a JK flip-flop
    always @(posedge clk) begin
        if (sync_reset) begin // reset statement it assigns into 0
            Q <= 1'b0;
        end else begin
            if (J && K) begin // 11 -> complement so make it complement of Q
                Q <= ~Q;
            end else if (!J && K) begin // 01 -> reset so make it 0
                Q <= 1'b0;
            end else if (J && !K) begin // 10 -> set so make it 1
                Q <= 1'b1;
            end
            // if it didn't go anything it means that 00 so it stays same
        end
    end
endmodule
```

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

    wire da, db, dc;

    // for the input of count[2]'s D flip flop
    assign da = (~mode & ~count[2] & count[1] & count[0]) |
    (mode & count[1] & ~count[0]) |
    (~mode & count[2] & ~count[1]) |
    (~mode & count[2] & ~count[0]) |
    (mode & count[2] & count[0]);

    // for the input of count[1]'s D flip flop
    assign db = (count[1] & ~count[0]) |
    (~mode & ~count[1] & count[0]) |
    (mode & ~count[2] & count[0]);

    // for the input of count[0]'s D flip flop
    assign dc = (~mode & ~count[0]) |
    (mode & ~count[2] & ~count[1]) |
    (mode & count[2] & count[1]);


    // work with 3 D Flip Flop because I have 8 states log2^8 = 3.
    dff_sync_res dffA(da, clk, reset, count[2]);
    dff_sync_res dffB(db, clk, reset, count[1]);
    dff_sync_res dffC(dc, clk, reset, count[0]);

endmodule
```

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

    // ja, jb, and jc are the J inputs for the JK flip-flops used in the counter.
    // ka, kb, and kc are the K inputs for the JK flip-flops used in the counter.

    wire ja, jb, jc, ka, kb, kc;

    // for the input of count[2]'s jk flip flop
    assign ja = (~mode & count[1] & count[0]) | (mode & count[1] & ~count[0]);
    assign ka = (~mode & count[1] & count[0]) | (mode & ~count[1] & ~count[0]);

    // for the input of count[1]'s jk flip flop
    assign jb = (~mode & count[0]) | (~count[2] & count[0]);
    assign kb = (~mode & count[0]) | (count[2] & count[0]);

    // for the input of count[0]'s jk flip flop
    assign jc = (~mode) | (~count[2] & ~count[1]) | (count[2] & count[1]);
    assign kc = (~mode) | (~count[2] & count[1]) | (count[2] & ~count[1]);


    // work with 3 JK Flip Flop because I have 8 states log2^8 = 3.
    jk_sync_res jkA(ja, ka, clk, reset, count[2]);
    jk_sync_res jkB(jb, kb, clk, reset, count[1]);
    jk_sync_res jkC(jc, kc, clk, reset, count[0]);

endmodule
```

3 Testbench Implementation

In this code I tried whether the gray and binary counter code are working or not. For checking it, first, I will reset the system to 0 and see if the system has been reset, then I will try to see if it can count binary when mode is 0 and make it count at least 0 to 8, then set the mode to 1 in the remaining part. I check if the gray counter is working or not. Finally, I reset again and see if the reset works properly.

```
`timescale 1ns/1ps

module counter_tb;
    reg reset, clk, mode;
    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
        $dumpfile("result.vcd");
        $dumpvars;

        reset = 1; // in the beginning reset will be 1
        mode = 0; // in the beginning mode will be 0 because I will try to solve
        #75; // starts from here
        reset = 0; // make reset is 0 and run code
        #500;
        mode = 1; // for checking the gray code change mode into 1
        #500;
        reset = 1; // finish by making reset 1
        #75;
        $finish;
    end


    initial begin
        // clock implementation
        forever begin
            #20 clk = 1;
            #20 clk = 0;
        end

    end

endmodule
```

4 Results

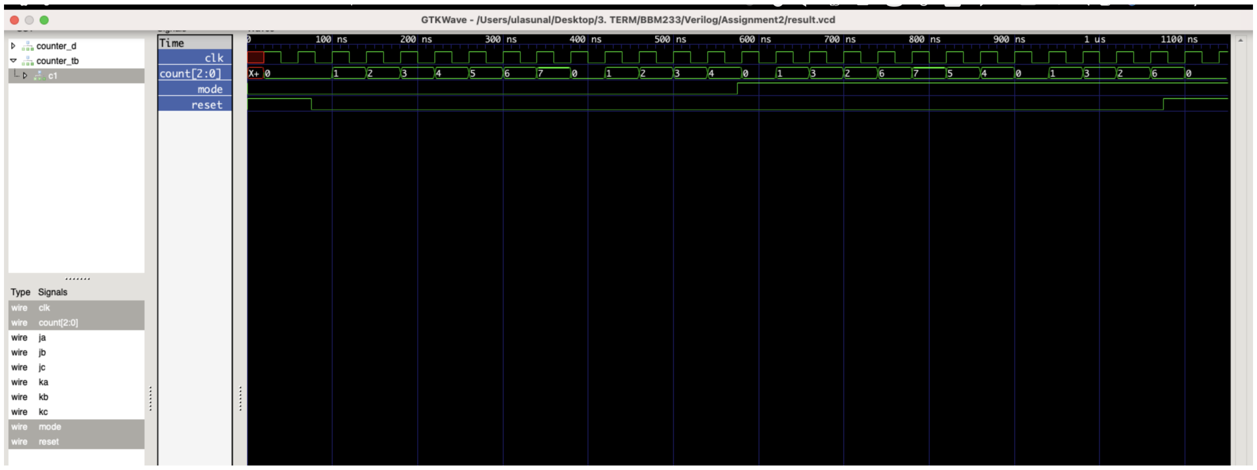


Figure 1: D Flip-Flop Resulting Waveform

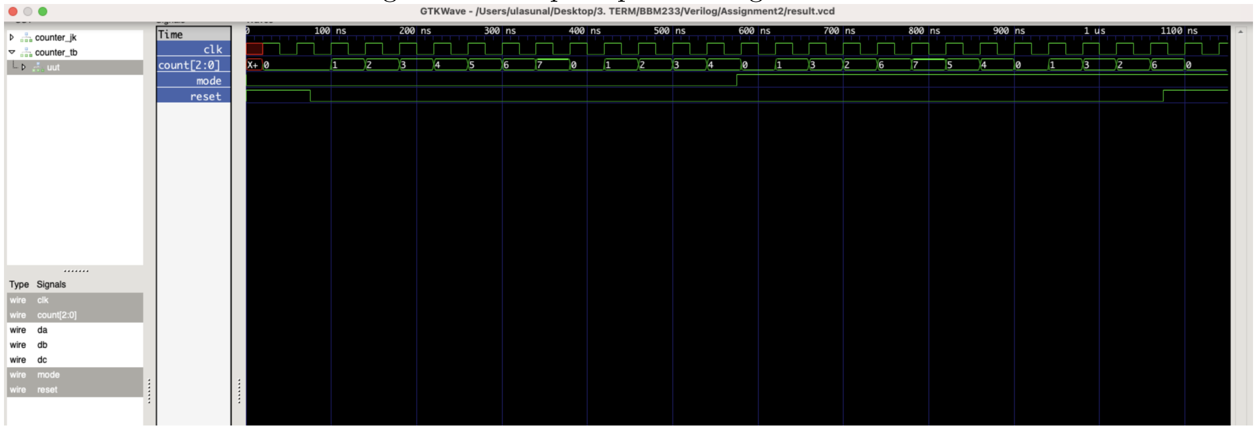


Figure 2: JK Flip-Flop Resulting Waveform

First, reset is 1 output stays as 0 then, reset is 0 and mode is 0, when mode is 0 natural binary code counter will work and count like 0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 0 - 1 - 2... then I make mode is 1 for counting gray counter it counts like 0 - 1 - 3 - 2 - 6 - 7 - 5 - 4 - 0 - 1 - 3 - 2... then make reset is 1 for finishing code. waveform screenshots are in next page