Hacettepe University

Computer Engineering Department

BM233 Logic Design Lab - 2022 Fall

Sequential Circuits in Verilog

January 1, 2023

Student name:
Muhammed Tahir Akdeniz

Student Number:
b2200356027

1 Problem Definition

Sequential circuits are circuits whose outputs depend on both the present inputs and the sequence of past inputs (sequential circuits include memory elements). That is, the previous inputs or state of the circuit will have an effect on its present state.

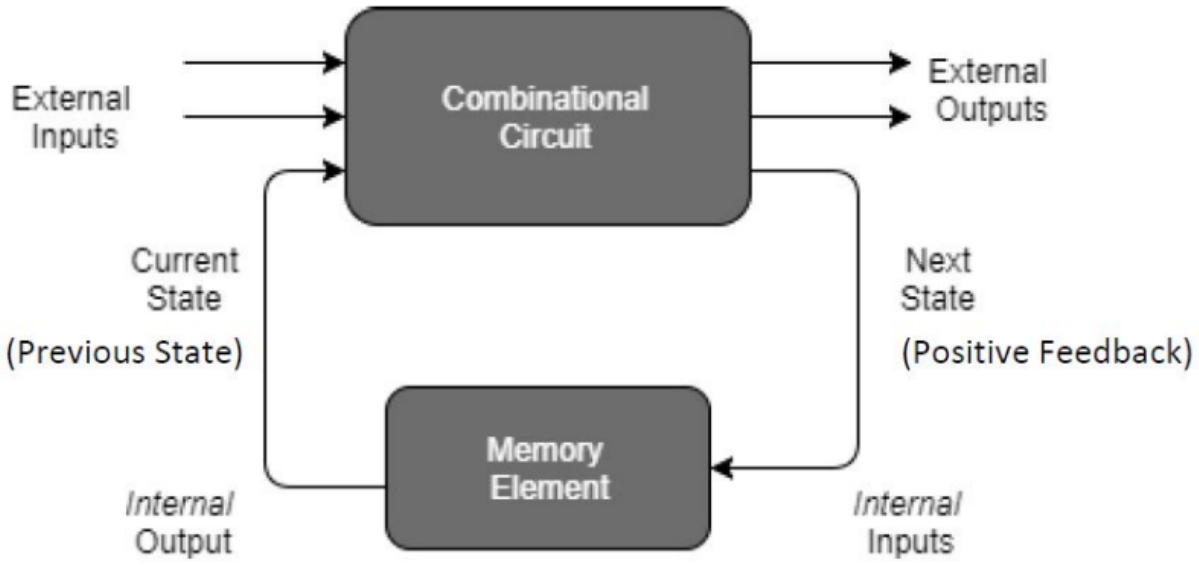

Figure 1: Sequential Circuits

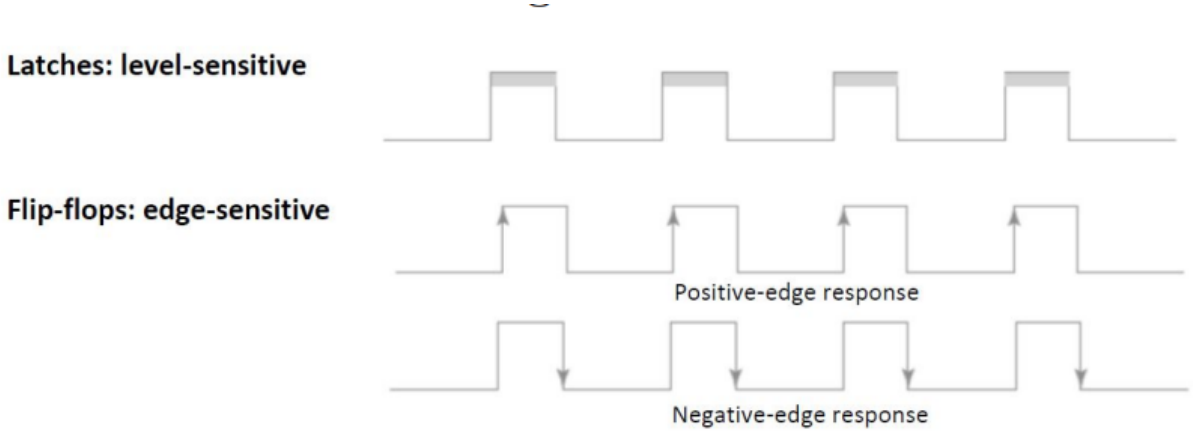

Figure 2: Storage Elements

A flip-flop is a basic building block of sequential logic circuits. It is a circuit that has two stable states and can store one bit of state information. The output changes state by signals applied to one or more control inputs.

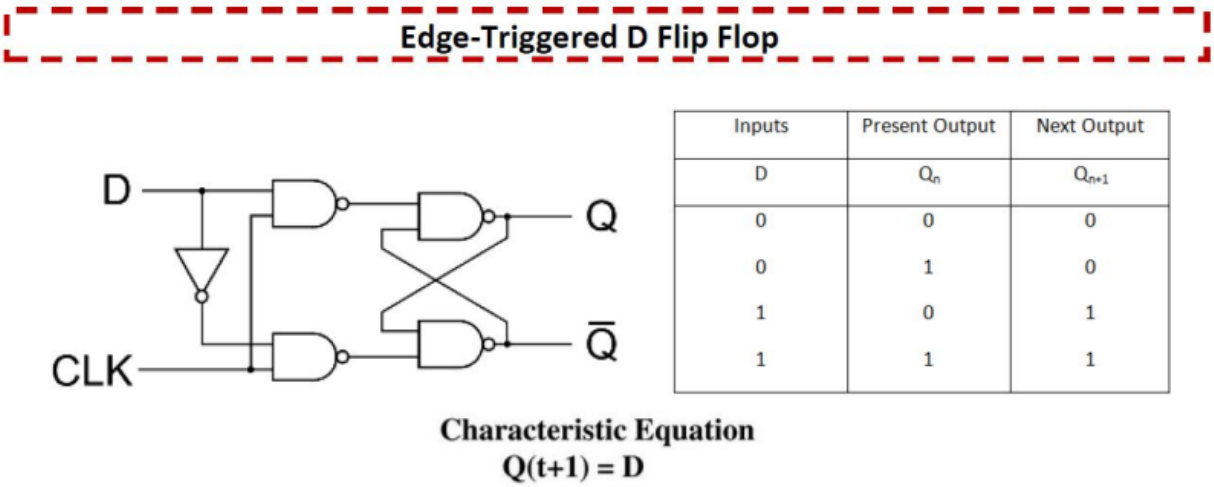

Figure 3: Edge-Triggered D flip flop

A basic D flip-flop has a D (data) input, a clock (CLK) input and outputs Q and Q' (the inverse of Q). Optionally it may also include the PR (Preset) and CLR (Clear) control inputs.

1.1 Experiment

Gray code is a form of binary that uses a different method of incrementing from one number to the next. With gray code, only one bit changes state from one position to another. Below table illustrates the difference between binary and gray code.

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

Figure 4: 3-Bit Binary/Gray Code Counter Table

Even though gray code seems like an innocent and theoretical encoding system invented only for fun, it has a lot of important use cases. Here are some example use cases of gray code: Axes of Karnaugh Maps, mathematical puzzles, telegraphy codes, ADC conversion, error correction, genetic algorithms.

For this experiment, it is 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 you design should also have another 1-bit input variable reset, which resets the circuit state to the start state 000.

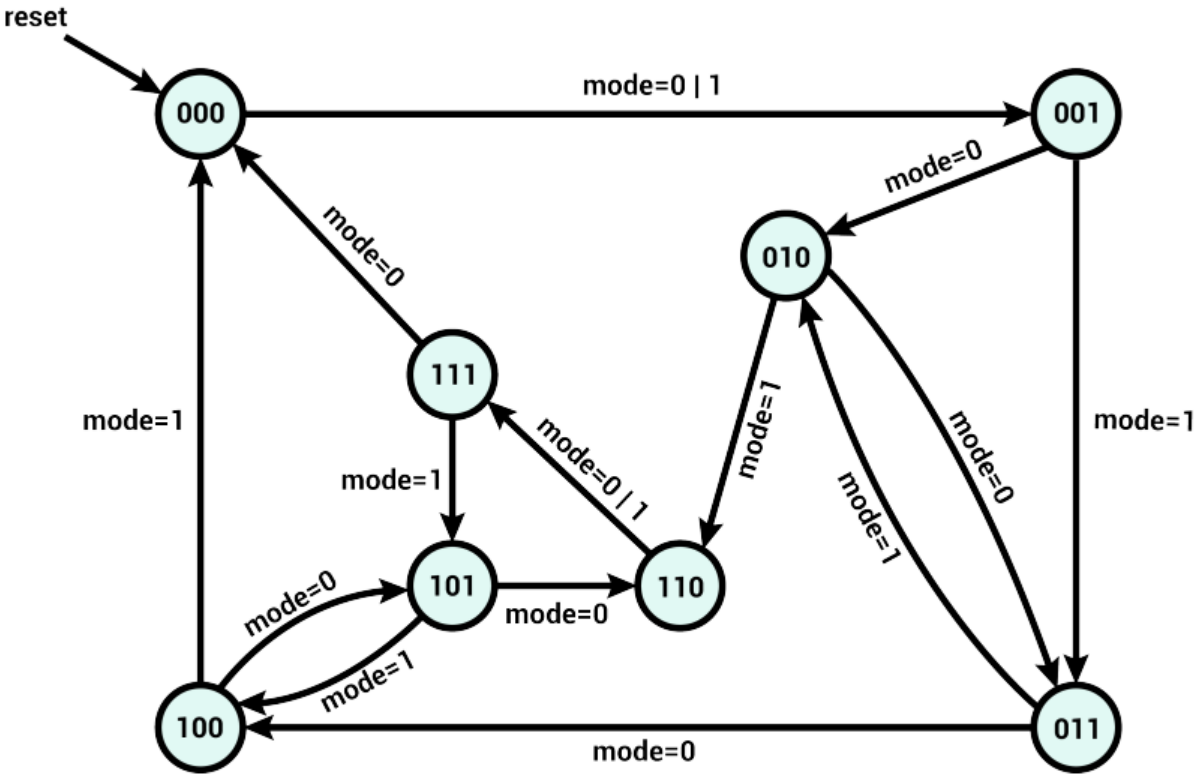

Figure 5: 3-Bit Binary/Gray Code Counter State Diagram

2 Solution Implementation

2.1 Truth Table, Karnaugh Map and Circut

present state			mode	next state			D_A	D_B	D_C
A	B	C		A	B	C			
0	0	0	0	0	0	1	0	0	1
			1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	0
			1	0	1	1	0	1	1
0	1	0	0	0	1	1	0	1	1
			1	1	1	0	1	1	0
0	1	1	0	1	0	0	1	0	0
			1	0	1	0	0	1	0
1	0	0	0	1	0	1	1	0	1
			1	0	0	0	0	0	0
1	0	1	0	1	1	0	1	1	0
			1	1	0	0	1	0	0
1	1	0	0	1	1	1	1	1	1
			1	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0	0
			1	1	0	1	1	0	1

Figure 6: Truth Table for D Flip Flop

present state			mode	next state			J_A	K_A	J_B	K_B	J_C	K_C
A	B	C		A	B	C						
0	0	0	0	0	0	1	0	X	0	X	1	X
			1	0	0	1	0	X	0	X	1	X
0	0	1	0	0	1	0	0	X	1	X	X	1
			1	0	1	1	0	X	1	X	X	0
0	1	0	0	0	1	1	0	X	X	0	1	X
			1	1	1	0	1	X	X	0	0	X
0	1	1	0	1	0	0	1	X	X	1	X	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	0	X	1	0	X	0	X
1	0	1	0	1	1	0	X	0	1	X	X	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	1	1	X	0	X	0	1	X
1	1	1	0	0	0	0	X	1	X	1	X	1
			1	1	0	1	X	0	X	1	X	0

Figure 7: Truth Table for JK Flip Flop

$$D_A(A,B,C,mode) = BC'mode + A'BCmode' + AB'mode' + ACmode + AC'mode' \quad (1)$$
$$D_B(A,B,C,mode) = BC' + B'Cmode' + A'Cmode \quad (2)$$
$$D_C(A,B,C,mode) = A'B'mode + C'mode' + ABmode \quad (3)$$
$$J_A(A,B,C,mode) = BC'mode + BCmode' \quad (4)$$
$$K_A(A,B,C,mode) = B'C'mode + BCmode' \quad (5)$$
$$J_B(A,B,C,mode) = A'C + Cmode' \quad (6)$$
$$K_B(A,B,C,mode) = Cmode' + AC \quad (7)$$
$$J_C(A,B,C,mode) = A'B' + mode' + AB \quad (8)$$
$$K_C(A,B,C,mode) = mode' + A'B + AB' \quad (9)$$
$$(10)$$

These equations define the outputs of 3-bit binary-to-Gray code converters implemented using D flip-flops and JK flip-flops. The D flip-flop equations specify the outputs D_A, D_B, and D_C

as functions of the inputs A, B, C, and mode. The JK flip-flop equations specify the outputs J_A, K_A, J_B, K_B, J_C, and K_C as functions of the inputs A, B, C, and mode.

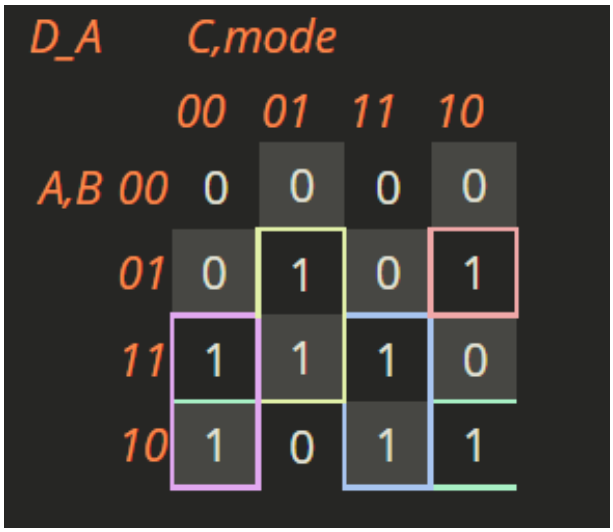

Figure 8: Karnaugh Map for D_A (1)

Figure 9: Karnaugh Map for D_B (2)

Figure 10: Karnaugh Map for D_C (3)

Figure 11: Karnaugh Map for J_A (4)

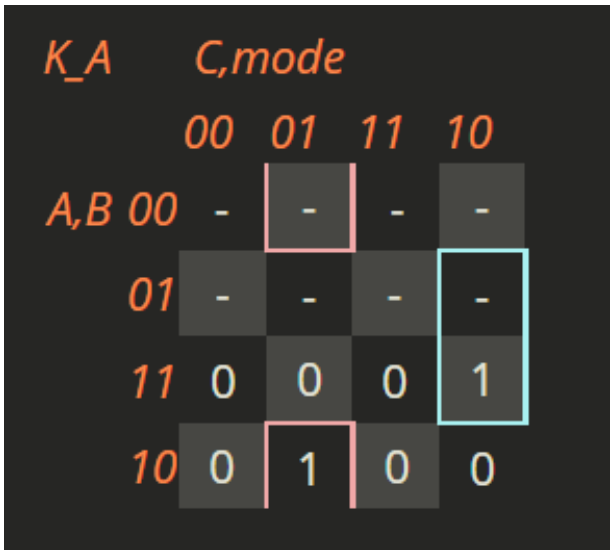

Figure 12: Karnaugh Map for K_A (5)

Figure 13: Karnaugh Map for J_B (6)

Figure 14: Karnaugh Map for K_B (7)

Figure 15: Karnaugh Map for J_C (8)

Figure 16: Karnaugh Map for K_C (9)

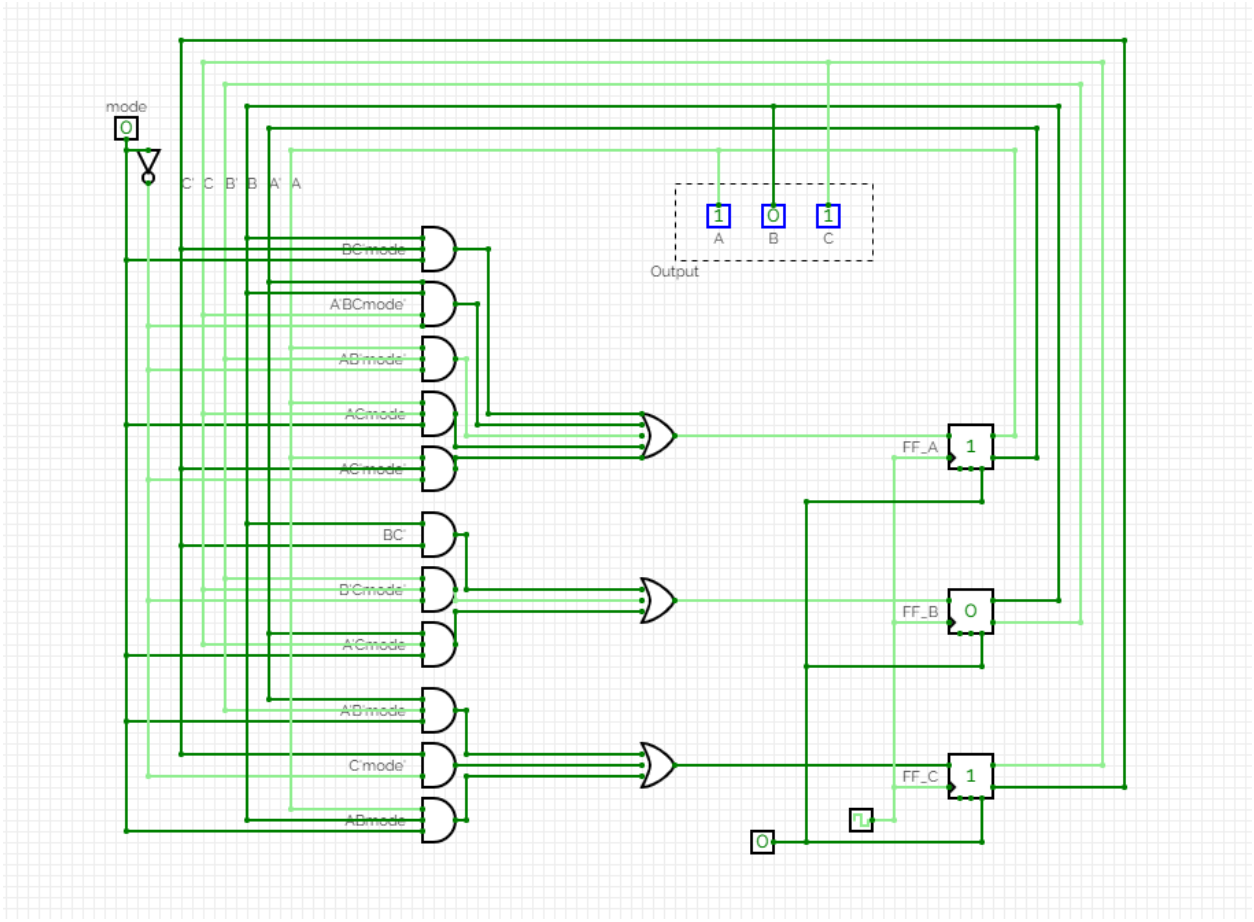

Figure 17: 3-Bit Binary/Gray Code Circuit Implementation with D Flip Flop

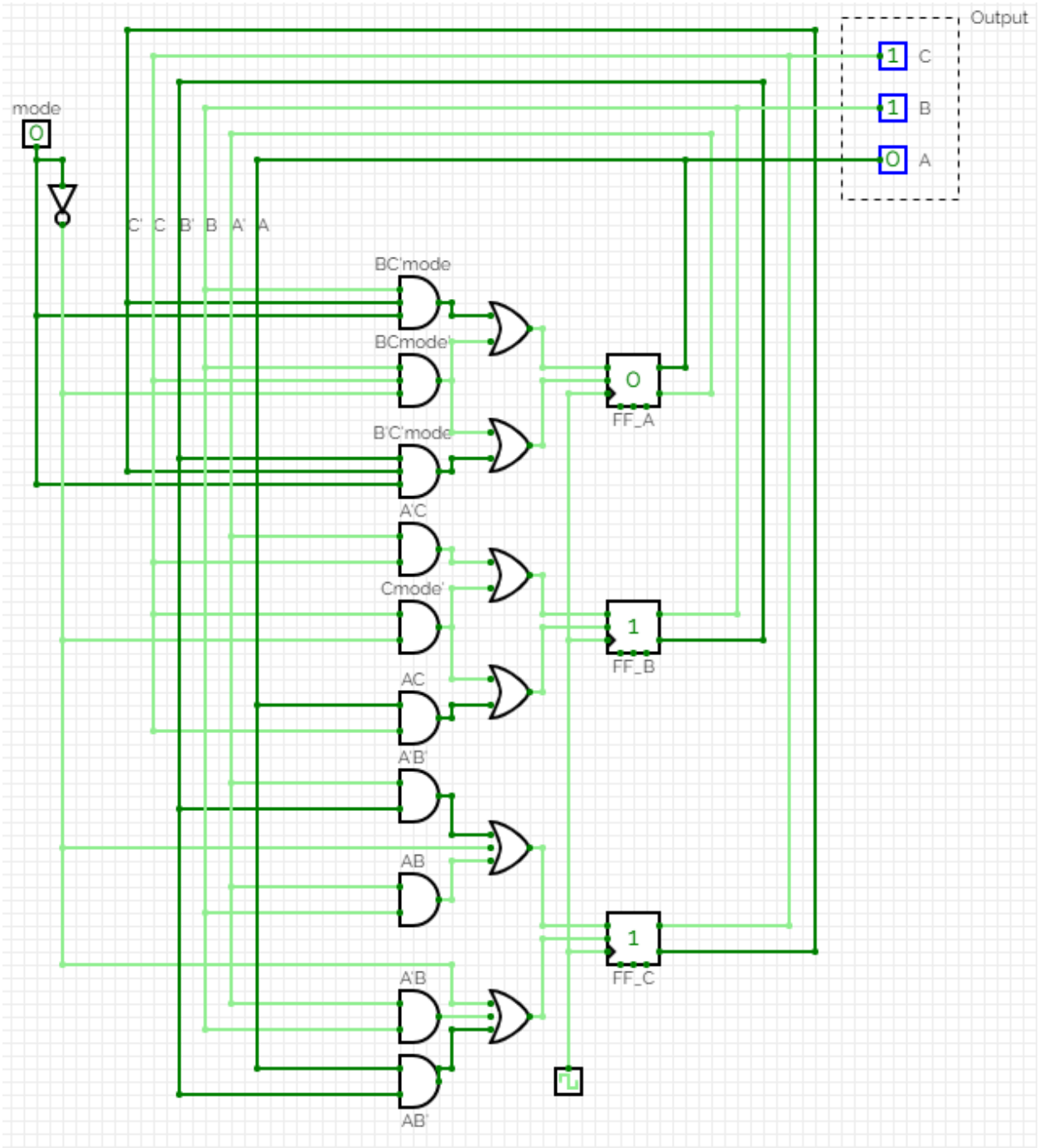

Figure 18: 3-Bit Binary/Gray Code Circuit Implementation with JK Flip Flop

2.2 Verilog Implementation

The counter_d and counter_jk modules are Verilog modules that implement a 3-bit binary counter and a 3-bit Gray code counter, respectively. Both modules have four input signals: reset, clk, mode, and count. The reset signal is a 1-bit input that, when set to 1, resets the counter to 0. The clk signal is a 1-bit clock signal that controls the counter's behavior. The mode signal is a 1-bit input that determines which mode the counter is in (binary or Gray code). The count signal is a 3-bit output that represents the current count value.

The counter_d module uses three D flip-flops (dff_sync_res) to implement the 3-bit binary counter, while the counter_jk module uses three JK flip-flops (jk_sync_res) to implement the 3-bit Gray code counter. In both modules, each flip-flop is responsible for one bit of the count value, with the most significant bit (A) represented by the first flip-flop, the second most significant bit (B) represented by the second flip-flop, and the least significant bit (C) represented by the third flip-flop. The input and output functions of each flip-flop are derived using Karnaugh maps.

The dff_sync_res module is a Verilog module that implements a D flip-flop with synchronous reset. It has four input signals: D, clk, sync_reset, and Q. The D signal is a 1-bit input that specifies the data to be stored, and the clk signal is a 1-bit input that specifies when the data should be stored. The sync_reset signal is a 1-bit input that, when set to 1, resets the flip-flop to 0. The Q signal is a 1-bit output that provides the current value of the stored data. The behavior of the flip-flop is controlled using an always block with a positive edge trigger on the clk input.

The jk_sync_res module is a Verilog module that implements a JK flip-flop with synchronous reset. It has five input signals: J, K, clk, sync_reset, and Q. The J and K signals are 1-bit inputs that control the behavior of the flip-flop. The clk signal is a 1-bit input that specifies when the J and K inputs should be applied. The sync_reset signal is a 1-bit input that, when set to 1, resets the flip-flop to 0. The Q signal is a 1-bit output that provides the current value of the stored data. The behavior of the flip-flop is controlled using an always block with a positive edge trigger on the clk input.

```
1  module counter_d(input reset, input clk, input mode, output [2:0] count);
2
3      // It is needed 3 flip flop to represent given states
4      // A = count[2]
5      // B = count[1]
6      // C = count[0]
7
8      // flip flop for A
9      dff_sync_res d_a (
10         .D(
11             // derived input and output functions using Karnaugh Maps for A
12             (count[1] & ~count[0] & mode)
```

```
            | (~count[2] & count[1] & count[0] & ~mode)
            | (count[2] & ~count[1] & ~mode)
            | (count[2] & count[0] & mode)
            | (count[2] & ~count[0] & ~mode)
        ),
        .clk(clk),
        .sync_reset(reset),
        .Q(count[2])
    );

    // flip flop for B
    dff_sync_res d_b(
        .D(
            // derived input and output functions using Karnaugh Maps for B
            (count[1] & ~count[0])
            | (~count[1] & count[0] & ~mode)
            | (~count[2] & count[0] & mode)),
        .clk(clk),
        .sync_reset(reset),
        .Q(count[1])
    );

    // flip flop for C
    dff_sync_res d_c(
        .D(
            // derived input and output functions using Karnaugh Maps for C
            (~count[2] & ~count[1] & mode)
            | (~count[0] & ~mode)
            | (count[2] & count[1] & mode)),
        .clk(clk),
        .sync_reset(reset),
        .Q(count[0])
    );

endmodule

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

    // It is needed 3 flip flop to represent given states
    // A = count[2]
    // B = count[1]
    // C = count[0]

    jk_sync_res j_A(
        // derived input and output functions (J AND K) using Karnaugh Maps for A
        .J((count[1] & ~count[0] & mode) | (count[1] & count[0] & ~mode)),
        .K((~count[1] & ~count[0] & mode) | (count[1] & count[0] & ~mode)),
        .clk(clk),
```

```
        .sync_reset(reset),
        .Q(count[2])
    );

    jk_sync_res j_B(
        // derived input and output functions (J AND K) using Karnaugh Maps for B
        .J((count[0] & ~mode) | (~count[2] & count[0])),
        .K((count[0] & ~mode) | (count[2] & count[0])),
        .clk(clk),
        .sync_reset(reset),
        .Q(count[1])
    );

    jk_sync_res j_C(
        // derived input and output functions (J AND K) using Karnaugh Maps for C
        .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
        // reset condition:
        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 reset = 1, then reset the output
        if(sync_reset == 1'b1)
            Q <= 1'b0;
        else begin
            case ({J,K})
                2'b00 :  Q <= Q;
                2'b01 :  Q <= 0;
                2'b10 :  Q <= 1;
                2'b11 :  Q <= ~Q;
            endcase
        end
    end
endmodule
```

3 Testbench Implementation

This is a Verilog testbench for a 3-bit counter that has two modes: a binary counter mode and a Gray code counter mode. The testbench instantiates either a module called counter_d or counter_jk, depending on which line is commented out. The counter_d module is a 3-bit binary counter, and the counter_jk module is a 3-bit Gray code counter implemented using JK flip-flops.

The testbench has four input signals:

- reset: a 1-bit signal that, when set to 1, resets the counter to 0
- clk: a 1-bit clock signal that controls the counter's behavior
- mode: a 1-bit signal that determines which mode the counter is in (binary or Gray code)
- count: a 3-bit output signal that represents the current count value

The testbench has two initial blocks, which are executed when the simulation begins. The first initial block contains the $dumpfile and $dumpvars commands, which create a VCD (Value Change Dump) file that records the values of all the signals during the simulation. The second initial block contains a series of statements that specify the values of the input signals over time.

Overall, this testbench sets up a simulation of a 3-bit counter in binary and Gray code modes, and creates a VCD file to record the values of the signals during the simulation.

```
`timescale 1ns/1ps

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

        // Comment the next line out when testing your JK flip flop implementation.
```

```
10      counter_d uut(reset, clk, mode, count);
11
12      // Uncomment the next line to test your JK flip flop implementation.
13      // counter_jk c1(reset, clk, mode, count);
14
15      initial begin
16          $dumpfile("result.vcd");
17          $dumpvars;
18
19          // wait to see results after change variables
20
21          i = 0;
22          mode = 0;
23          reset = 1;
24          #15;
25
26          reset = 0;
27          #200;
28
29          mode = 1;
30          #200
31
32          reset = 1;
33          #50
34
35          $finish;
36      end
37
38      // clock to test
39      initial begin
40          clk = 0;
41          forever begin
42              #10;
43              clk = ~clk;
44          end
45      end
46
47  endmodule
```

4 Results

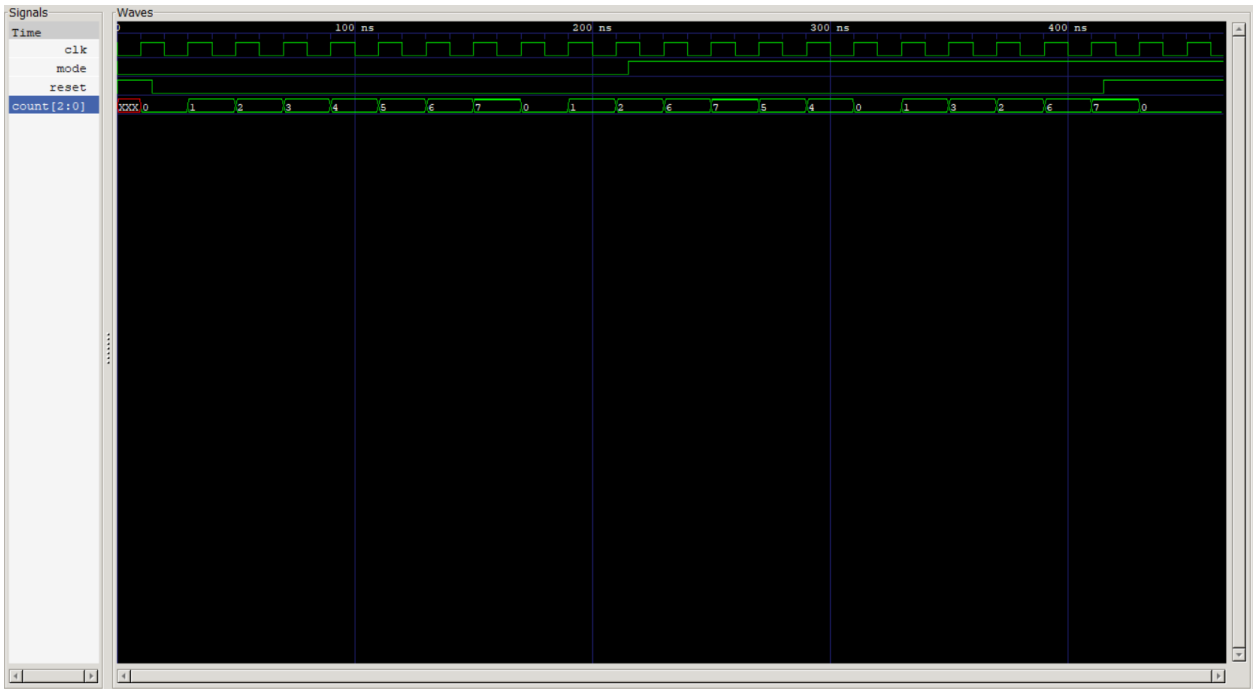

Figure 19: Resulting Waveform

References

- Verilog HDL, BB233 Lecture Notes, 2022
- BBM233 Fall 2022 Verilog Assignment 2, 2022
- https://web.cs.hacettepe.edu.tr/ bbm231/
- https://charlie-coleman.com/experiments/kmap/
- https://circuitverse.org/users/151469/projects/3-bit-binary-gray-code-circuit-implementation-with-jk-flip-flop
- https://circuitverse.org/users/151469/projects/d-flip-flop-gray-counter