Hacettepe University

Computer Engineering Department

BBM233 Logic Design Lab - 2022 Fall

Experiment 5 - Sequential Circuits

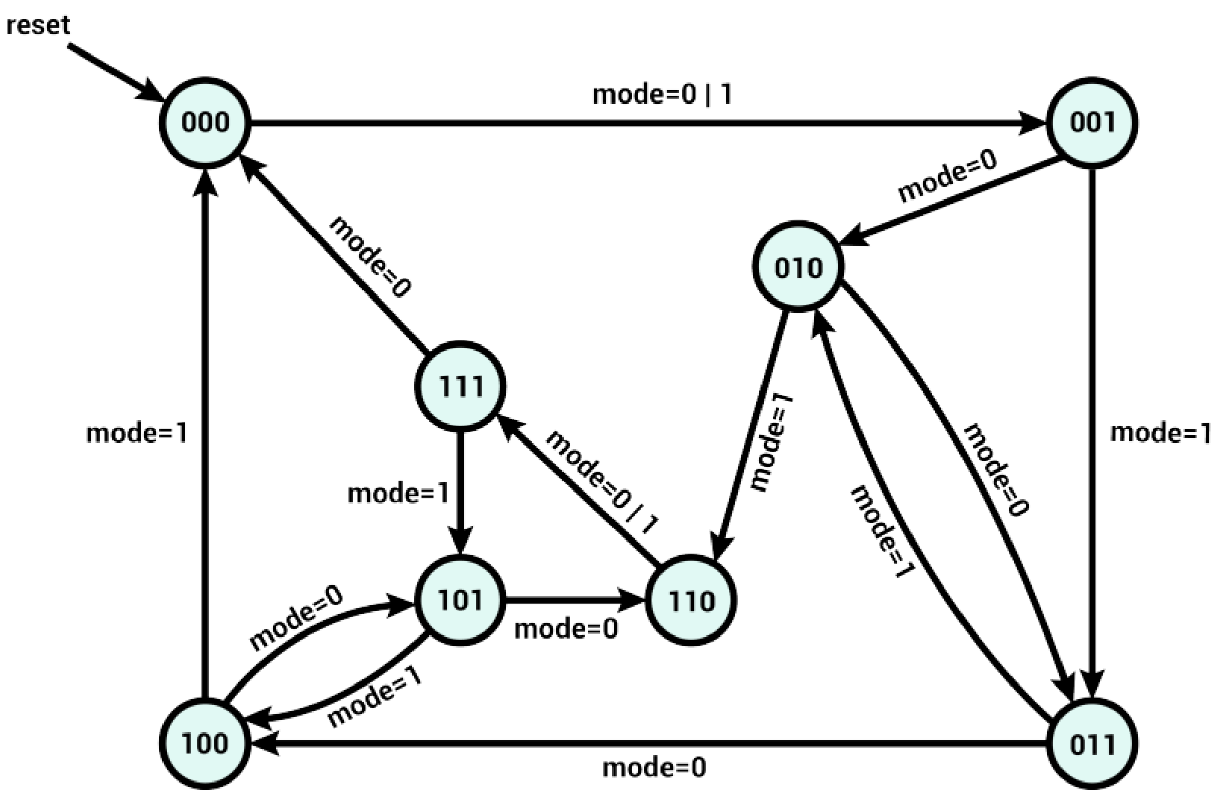
January 1, 2023

Student name:
Nurullah Başer

Student Number:
b2200356077

1 Problem Definition

For this experiment, I 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.

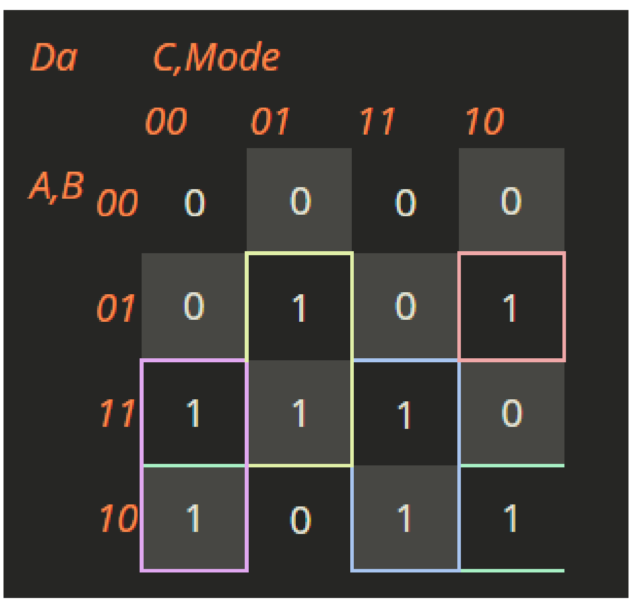


The counter state diagram is illustrated in the figure above. It has eight states, three 1-bit inputs, reset, mode, and clk, and a 3-bit output count that outputs the current state of the counter. Since there are eight states, we need to use three flip flops.

TABLE OF FLIP FLOPS

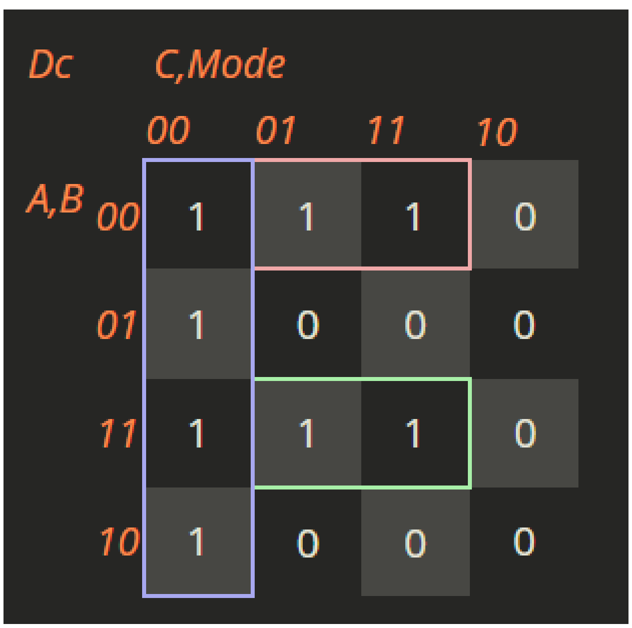
A	B	C	MODE	A'	B'	C'	Da	Db	Dc	Ja	Ka	Jb	Kb	Jc	Kc
0	0	0	0	0	0	1	0	0	1	0	x	0	x	1	x
0	0	0	1	0	0	1	0	0	1	0	x	0	x	1	x
0	0	1	0	0	1	0	0	1	0	0	x	1	x	x	1
0	0	1	1	0	1	1	0	1	1	0	x	1	x	x	0
0	1	0	0	0	1	1	0	1	1	0	x	x	0	1	x
0	1	0	1	1	1	0	1	1	0	1	x	x	0	0	x
0	1	1	0	1	0	0	1	0	0	1	x	x	1	x	1
0	1	1	1	0	1	0	0	1	0	0	x	x	0	x	1
1	0	0	0	1	0	1	1	0	1	x	0	0	x	1	x
1	0	0	1	0	0	0	0	0	0	x	1	0	x	0	x
1	0	1	0	1	1	0	1	1	0	x	0	1	x	x	1
1	0	1	1	1	0	0	1	0	0	x	0	0	x	x	1
1	1	0	0	1	1	1	1	1	1	x	0	x	0	1	x
1	1	0	1	1	1	1	1	1	1	x	0	x	0	1	x
1	1	1	0	0	0	0	0	0	0	x	1	x	1	x	1
1	1	1	1	1	0	1	1	0	1	x	0	x	1	x	0

K-MAPS AND FORMULAS

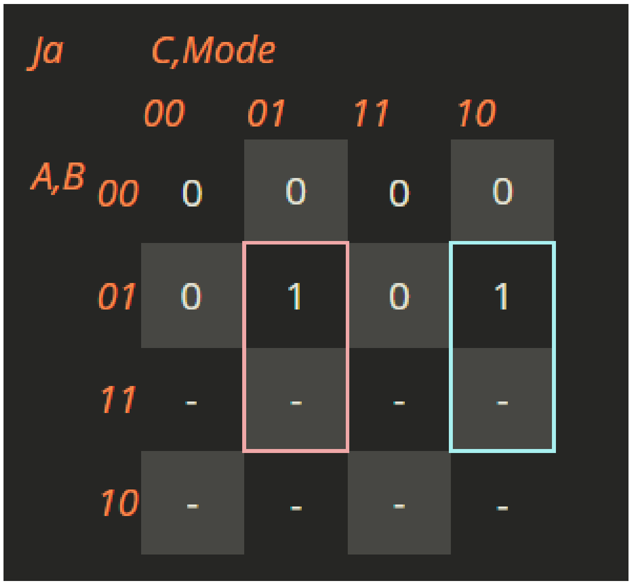


Da(A, B, C, Mode) = BC’Mode + A’BCMode’ + AB’Mode’ + ACMode + AC’Mode’

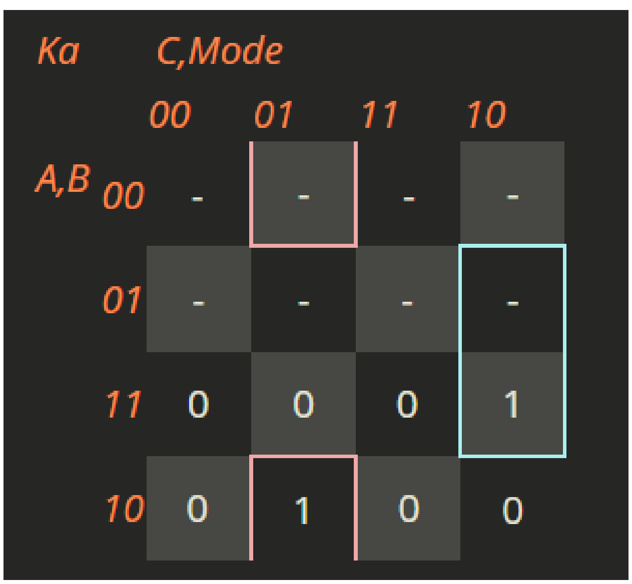
Db(A, B, C, Mode) = BC' + B'CMode' + A'CMode



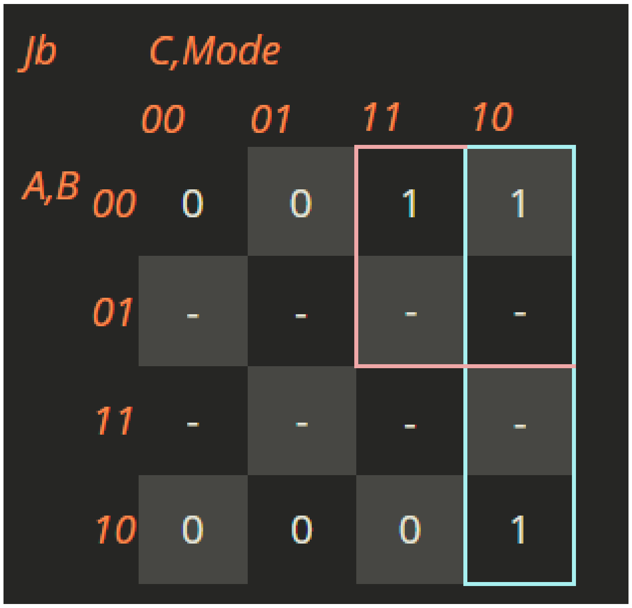
Dc(A, B, C, Mode) = A'B'Mode + C'Mode' + ABMode



Ja(A, B, C, Mode) = BC’Mode + BCMode’

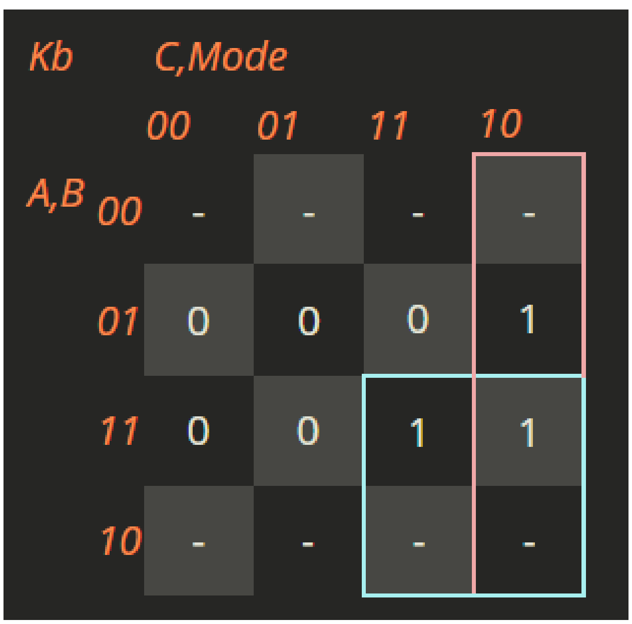


Ka(A, B, C, Mode) = B’C’Mode + BCMode’



Jb	C,Mode 00	01	11	10
A,B 00	0	0	1	1
01	-	-	-	-
11	-	-	-	-
10	0	0	0	1

Jb(A, B, C, Mode) = A'C + CMode'



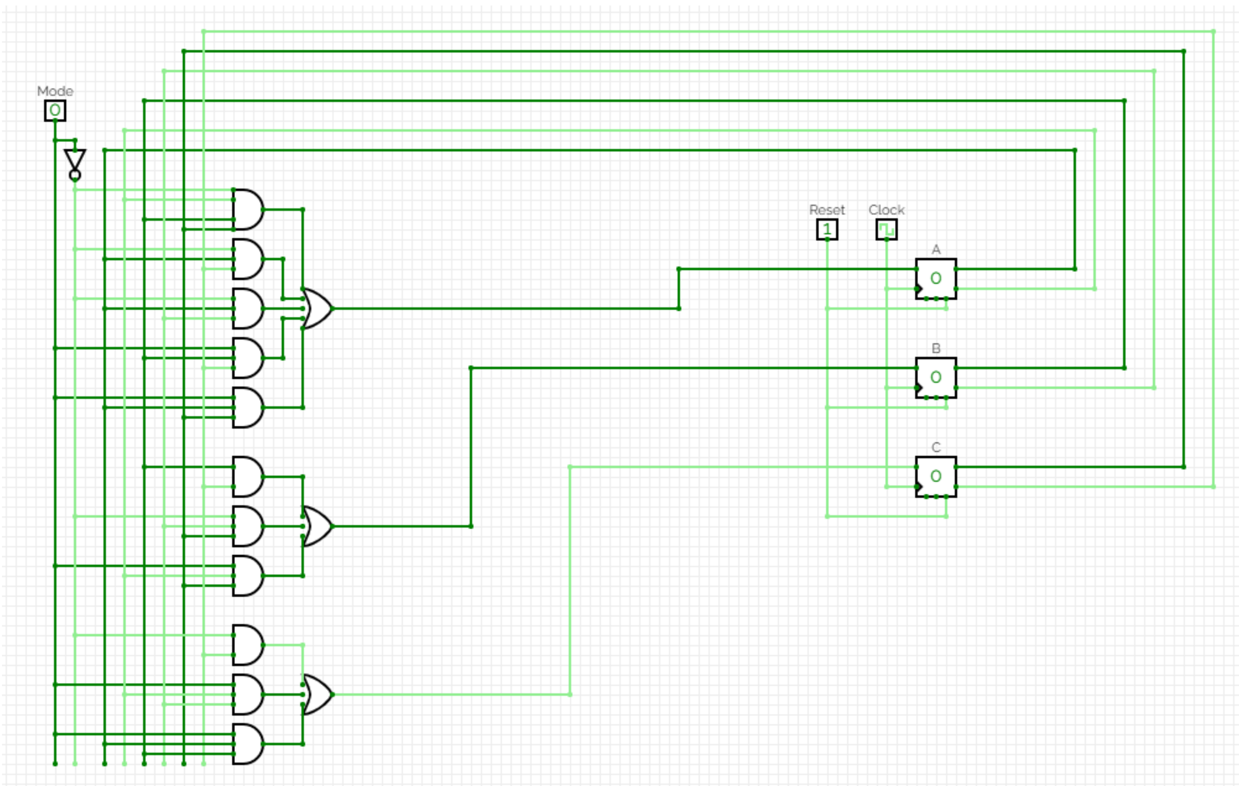
Kb	C,Mode 00	01	11	10
A,B 00	-	-	-	-
01	0	0	0	1
11	0	0	1	1
10	-	-	-	-

Kb(A, B, C, Mode) = CMode' + AC

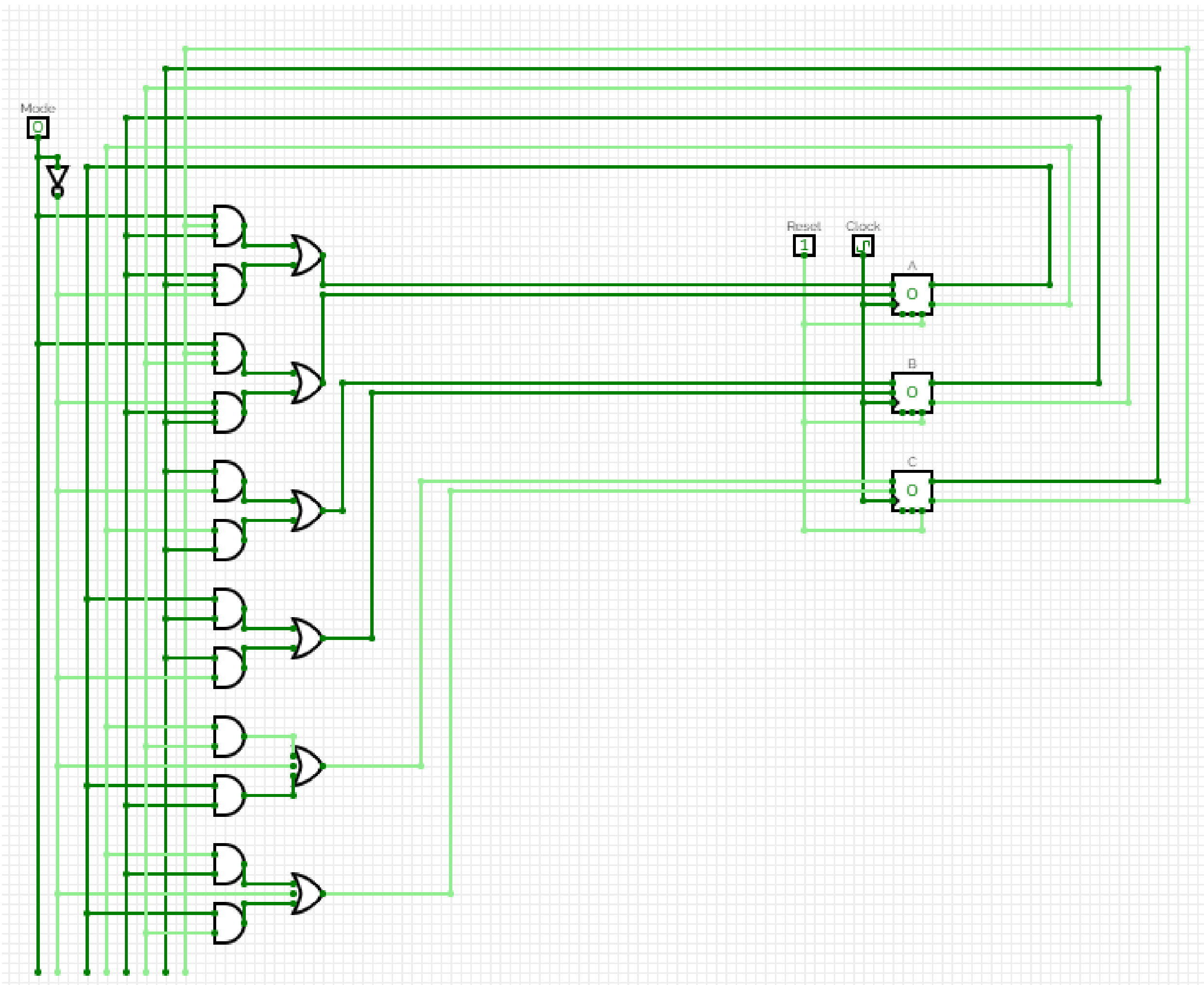
Jc(A, B, C, Mode) = A'B' + Mode' + AB

Kc(A, B, C, Mode) = Mode' + A'B + AB'

CIRCUITS OF FLIP FLOPS



CIRCUIT OF D FLIP FLOP



CIRCUIT OF JK FLIP FLOP

2 Solution Implementation

```
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) // if reset is 1 than Q equals 0
```

```
            Q <= 1'b0;
        else // if reset is 0 than Q equals D
            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) //if reset is 1 than Q = 0
            Q <= 1'b0;
        else begin
            if (J && K) // 11 = complement
                Q <= ~Q;
            else if (J && !K) // 10 reset
                Q <= 1'b1;
            else if (!J && K) // 01 set
                Q <= 1'b0;
        end
    end

endmodule
```

```
module counter_d(input reset, input clk, input mode, output [2:0] count);
    // This module implements a 3-bit counter using D flip-flops
    // Input determines count based on mode and current count values

    // First D flip-flop in the counter
    dff_sync_res dffA((~mode & ~count[2] & count[1] & count[0]) |
    (~mode & count[2] & ~count[0]) |
    (~mode & count[2] & ~count[1]) |
    (mode & count[1] & ~count[0]) |
    (mode & count[2] & count[0]),
    clk, reset, count[2]);

    // Second D flip-flop in the counter
    dff_sync_res dffB((count[1] & ~count[0]) |
    (~mode & ~count[1] & count[0]) |
    (mode & ~count[2] & count[0]),
    clk, reset, count[1]);

    // Third D flip-flop in the counter
```

```
    dff_sync_res dffC((~mode & ~count[0]) |
    (mode & ~count[2] & ~count[1]) |
    (mode & count[2] & count[1]),
    clk, reset, count[0]);

endmodule
```

```
module counter_jk(input reset, input clk, input mode, output [2:0] count);
    // This module implements a 3-bit counter using JK flip-flops
    // J and K inputs determine count based on mode and current count values

    // First JK flip-flop in the counter
    jk_sync_res jkffA(
        (count[1] & ~count[0] & mode) | (count[1] & count[0] & ~mode),
        (~count[1] & ~count[0] & mode) | (count[1] & count[0] & ~mode),
        clk,
        reset,
        count[2]
    );

    // Second JK flip-flop in the counter
    jk_sync_res jkffB(
        (count[0] & ~mode) | (~count[2] & count[0]),
        (count[2] & count[0]) | (count[0] & ~mode),
        clk,
        reset,
        count[1]
    );

    // Third JK flip-flop in the counter
    jk_sync_res jkffC(
        (~count[2] & ~count[1]) | (count[2] & count[1]) | (~mode),
        (~count[2] & count[1]) | (count[2] & ~count[1]) | (~mode),
        clk,
        reset,
        count[0]
    );

endmodule
```

3 Testbench Implementation

I start with a reset at first. Then I make reset to 0 and try binary counter and gray counter. After testing and seeing the accuracy, I make reset 1 again and activate it.

```
`timescale 1ns/1ps

```

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

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

    initial begin
        $dumpfile("counter.vcd");
        $dumpvars;
        reset = 1;#50; // in the beginning reset is 1 and system is closed
        reset = 0;#420; // reset is 0 for running code
        reset = 1;#50; // at the beginning reset is 1 and system is closed
        $finish;
    end

    initial begin // clock timer
        clk = 0;
        forever begin
            #10;
            clk = ~clk;
        end
    end

    always@(posedge clk) begin //if i is below than 12 make mode 0 for running
    //binary counter else make mode 1 for running gray counter
        #10;
        mode = i<12 ? 0 : 1;
        i = i+1;
    end

endmodule
```

4 Results

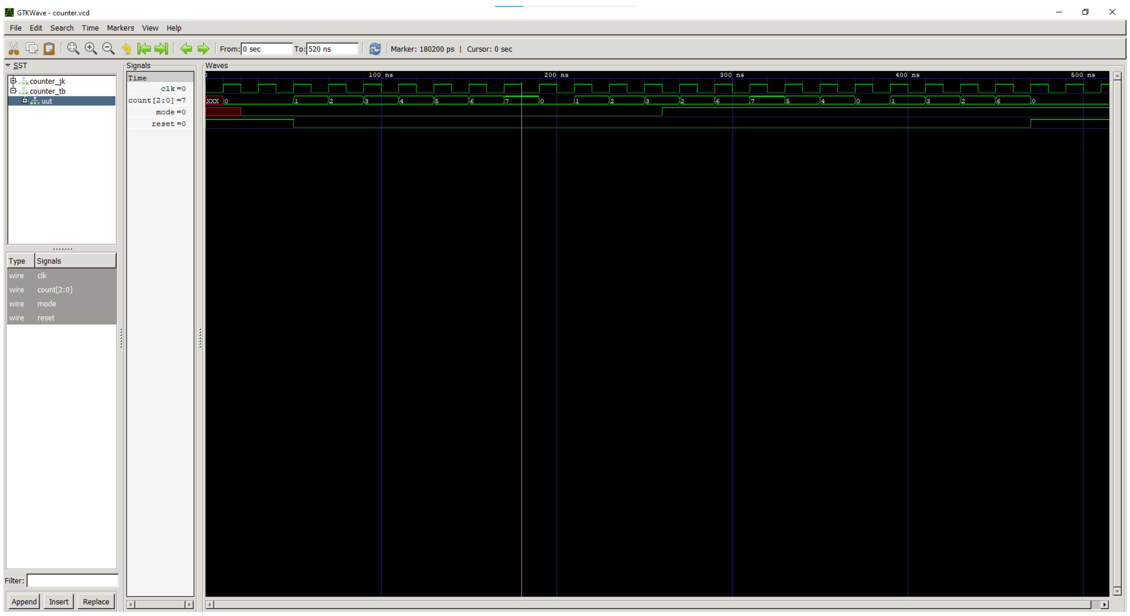


Figure 1: Resulting Waveform with D FlipFlop

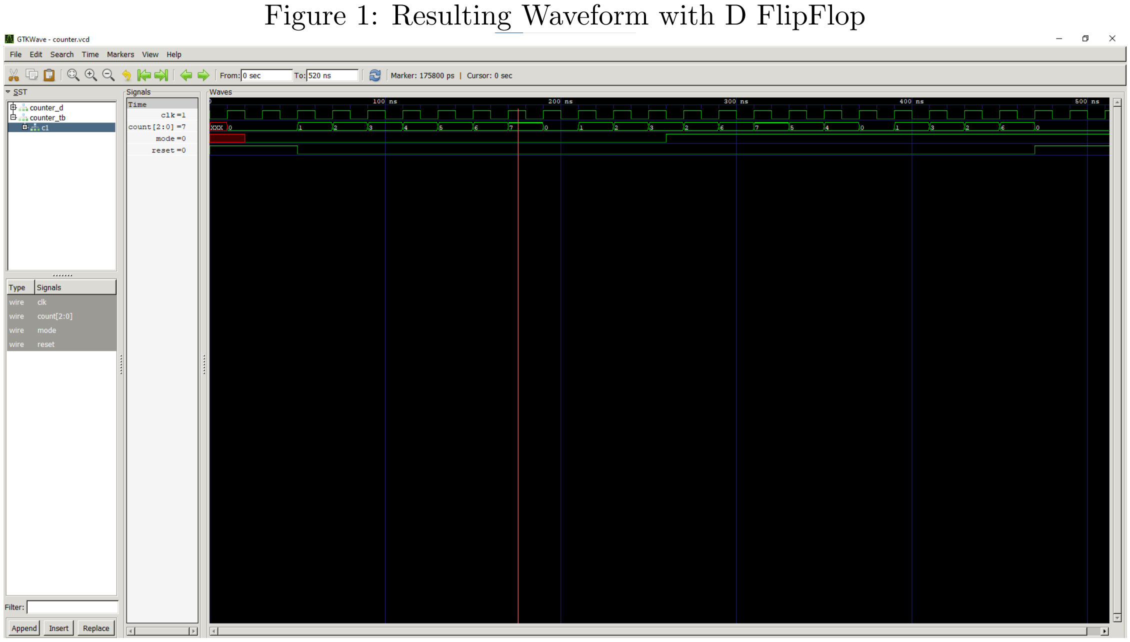


Figure 2: Resulting Waveform with J-K FlipFlop

If the reset value is 0, the system is closed. When reset is 1, we do the next state finding process according to the mode. If the mode is 0, we find it as binary, if the mode is 1, we find it as gray.

Natural Binary Counter = 0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 0 - ...

Gray Counter = 0 - 1 - 3 - 2 - 6 - 7 - 5 - 4 - 0 - ...

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