

# Observations (Basic Gates):

- RTL Source Code

The screenshot displays three Verilog modules and their corresponding testbenches in Visual Studio Code. The top row shows the gate modules: AND.v, OR.v, and NOT.v. The bottom row shows the testbenches: AND\_test.v, OR\_test.v, and NOT\_test.v. The testbenches include logic to initialize signals, apply test cases, and monitor the output.

```
module AND(input A,B,output Y);
    assign Y = A & B;
endmodule

module OR(input A,B, output Y);
    assign Y = A | B;
endmodule

module NOT(input X,output Y);
    assign Y = ~X;
endmodule

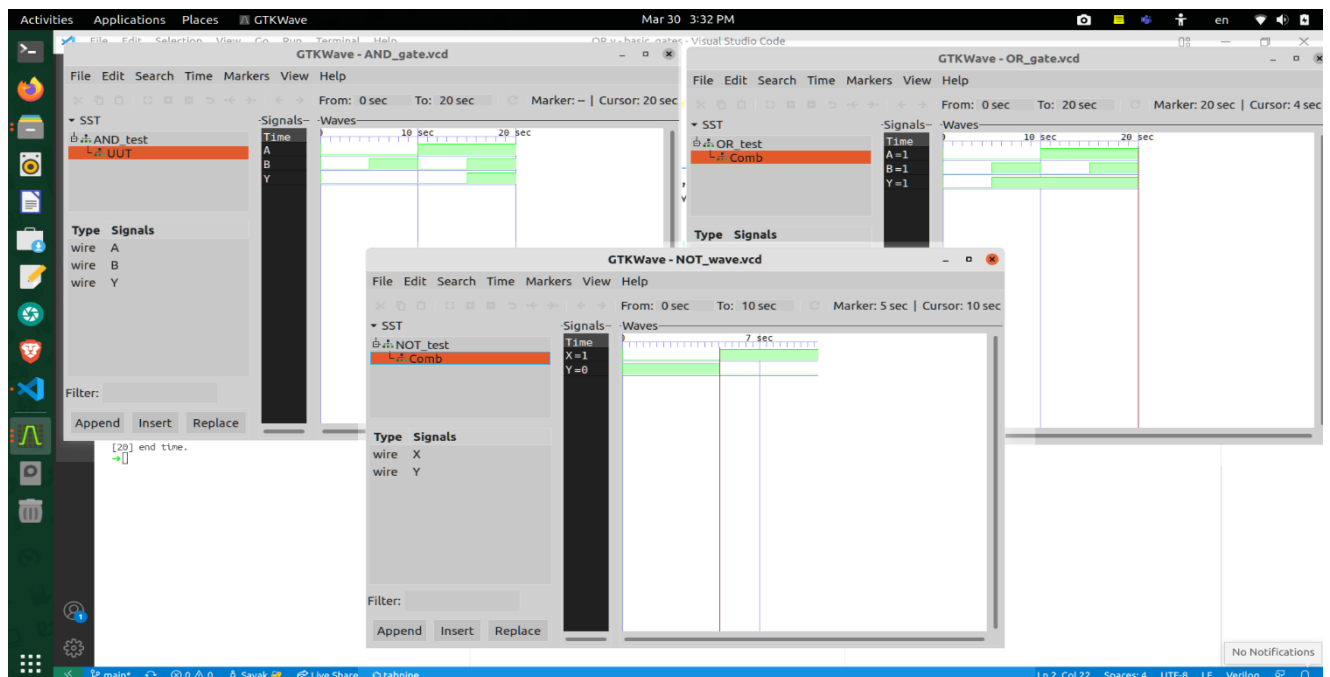
module AND_test();
    reg A,B;
    wire Y;
    AND UUT (.A(A),.B(B),.Y(Y));
    initial begin
        $dumpfile("AND_gate.vcd");
        $dumpvars(0,AND_test);
        $monitor("%d AND %d -> %d",
            A,B,Y);
        A=0; B=0; #5;
        A=0; B=1; #5;
        A=1; B=0; #5;
        A=1; B=1; #5;
        $finish;
    end
endmodule

module OR_test();
    reg A,B;
    wire Y;
    OR Comb(.A(A),.B(B),.Y(Y));
    initial begin
        $dumpfile("OR_gate.vcd");
        $dumpvars(0,OR_test);
        $monitor("%d OR %d -> %d",
            A,B,Y);
        A=0; B=0; #5;
        A=0; B=1; #5;
        A=1; B=0; #5;
        A=1; B=1; #5;
        $finish;
    end
endmodule

module NOT_test();
    reg X;
    wire Y;
    NOT Comb(.X(X),.Y(Y));
    initial begin
        $dumpfile("NOT_wave.vcd");
        $dumpvars(0,Comb);
        $monitor("NOT %d -> %d",
            X,Y);
        X=0; #5;
        X=1; #5;
        $finish;
    end
endmodule
```

(from left to right) Verilog code for AND gate, OR gate and NOT gate

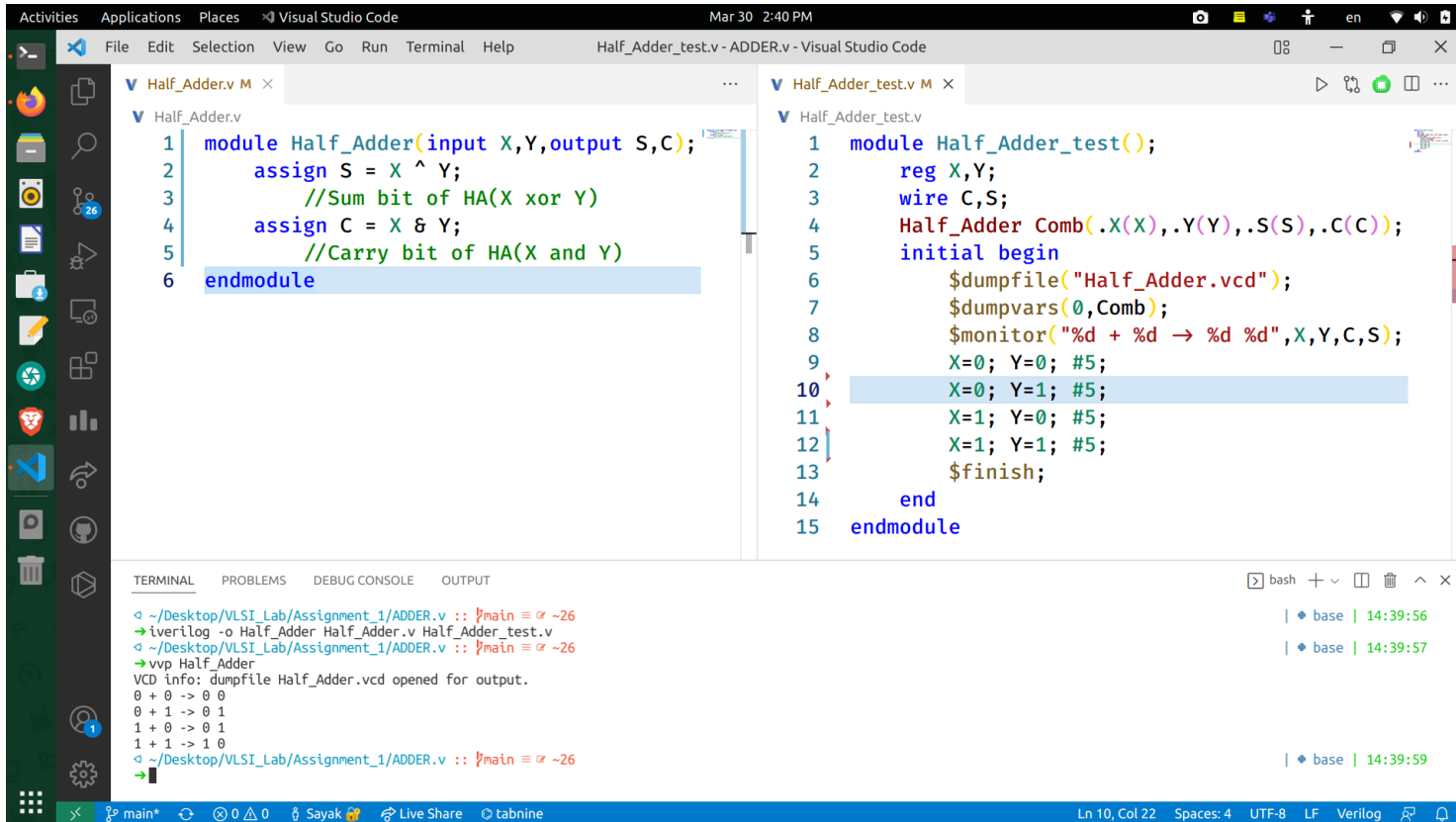
- GTKWave Output



(from top-left to right-bottom) gtkWAVE output for AND gate, OR gate and NOT gate

# Observations (Half Adder and Full Adder):

## • RTL Source Code



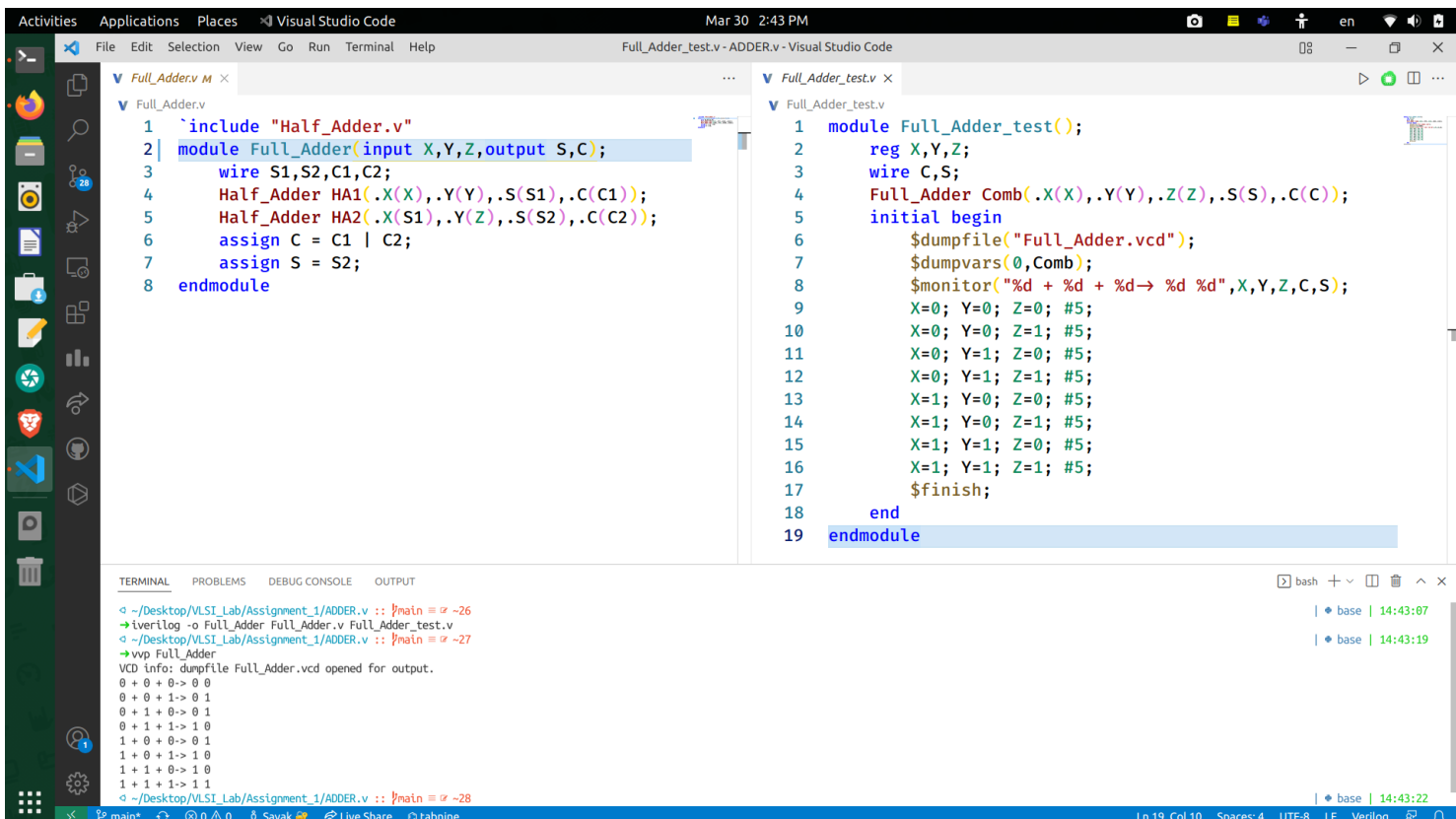
The screenshot shows the Visual Studio Code interface with two files open: `Half_Adder.v` and `Half_Adder_test.v`. The `Half_Adder.v` file contains the RTL code for a half adder, and the `Half_Adder_test.v` file contains the test bench code. The terminal at the bottom shows the execution of the Verilog code.

```
Half_Adder.v
1 module Half_Adder(input X,Y,output S,C);
2     assign S = X ^ Y;
3     //Sum bit of HA(X xor Y)
4     assign C = X & Y;
5     //Carry bit of HA(X and Y)
6 endmodule

Half_Adder_test.v
1 module Half_Adder_test();
2     reg X,Y;
3     wire C,S;
4     Half_Adder Comb(.X(X),.Y(Y),.S(S),.C(C));
5     initial begin
6         $dumpfile("Half_Adder.vcd");
7         $dumpvars(0,Comb);
8         $monitor("%d + %d -> %d %d",X,Y,C,S);
9         X=0; Y=0; #5;
10        X=0; Y=1; #5;
11        X=1; Y=0; #5;
12        X=1; Y=1; #5;
13        $finish;
14    end
15 endmodule

Terminal
~/Desktop/VLSI_Lab/Assignment_1/ADDER.v :: $main == ~26
iverilog -o Half_Adder Half_Adder.v Half_Adder_test.v
~/Desktop/VLSI_Lab/Assignment_1/ADDER.v :: $main == ~26
vvp Half_Adder
VCD info: dumpfile Half_Adder.vcd opened for output.
0 + 0 -> 0 0
0 + 1 -> 0 1
1 + 0 -> 0 1
1 + 1 -> 1 0
~/Desktop/VLSI_Lab/Assignment_1/ADDER.v :: $main == ~26
```

(from left to right) Verilog code for Half Adder and test bench



The screenshot shows the Visual Studio Code interface with two files open: `Full_Adder.v` and `Full_Adder_test.v`. The `Full_Adder.v` file contains the RTL code for a full adder, and the `Full_Adder_test.v` file contains the test bench code. The terminal at the bottom shows the execution of the Verilog code.

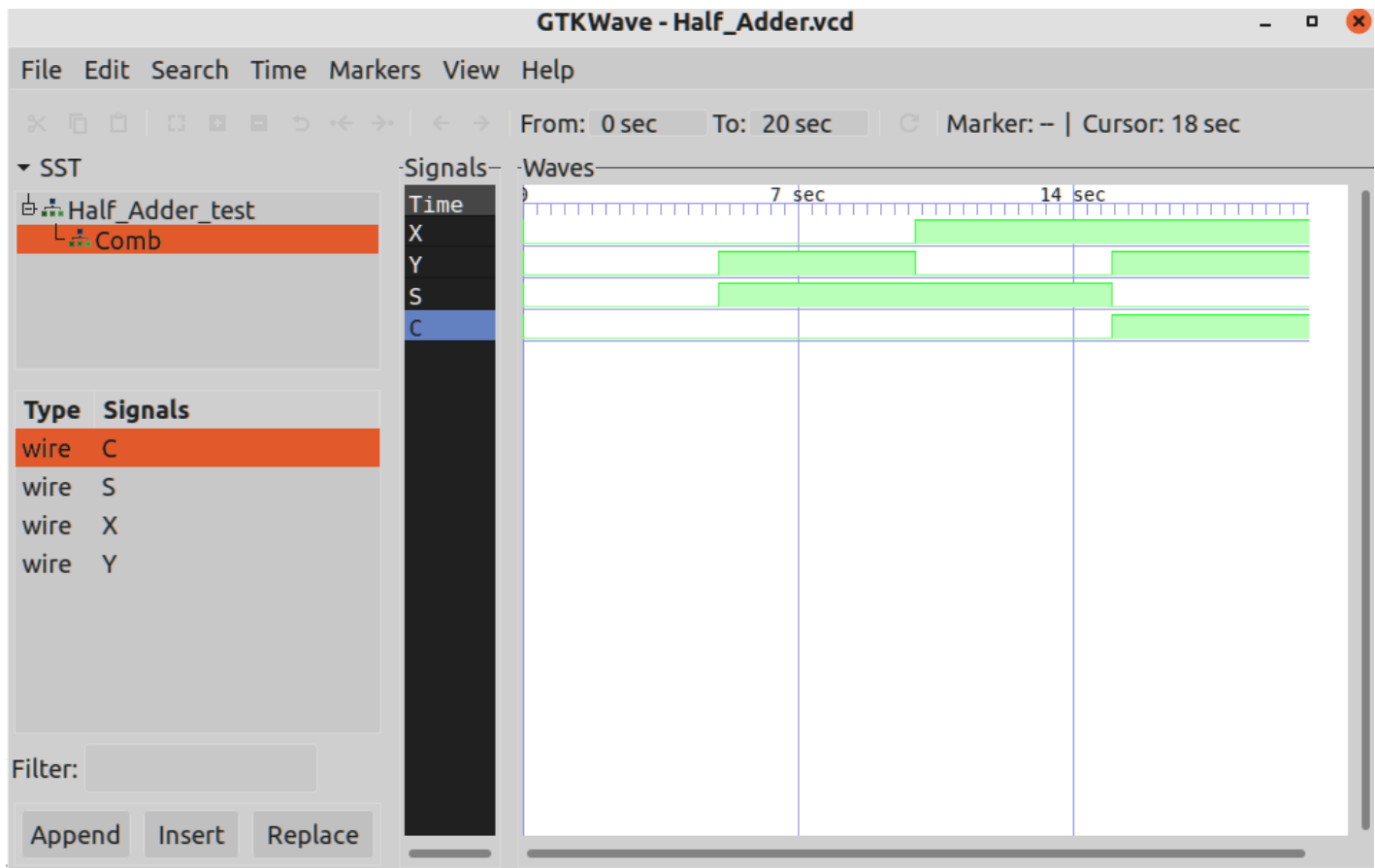
```
Full_Adder.v
1 `include "Half_Adder.v"
2 module Full_Adder(input X,Y,Z,output S,C);
3     wire S1,S2,C1,C2;
4     Half_Adder HA1(.X(X),.Y(Y),.S(S1),.C(C1));
5     Half_Adder HA2(.X(S1),.Y(Z),.S(S2),.C(C2));
6     assign C = C1 | C2;
7     assign S = S2;
8 endmodule

Full_Adder_test.v
1 module Full_Adder_test();
2     reg X,Y,Z;
3     wire C,S;
4     Full_Adder Comb(.X(X),.Y(Y),.Z(Z),.S(S),.C(C));
5     initial begin
6         $dumpfile("Full_Adder.vcd");
7         $dumpvars(0,Comb);
8         $monitor("%d + %d + %d -> %d %d",X,Y,Z,C,S);
9         X=0; Y=0; Z=0; #5;
10        X=0; Y=0; Z=1; #5;
11        X=0; Y=1; Z=0; #5;
12        X=0; Y=1; Z=1; #5;
13        X=1; Y=0; Z=0; #5;
14        X=1; Y=0; Z=1; #5;
15        X=1; Y=1; Z=0; #5;
16        X=1; Y=1; Z=1; #5;
17        $finish;
18    end
19 endmodule

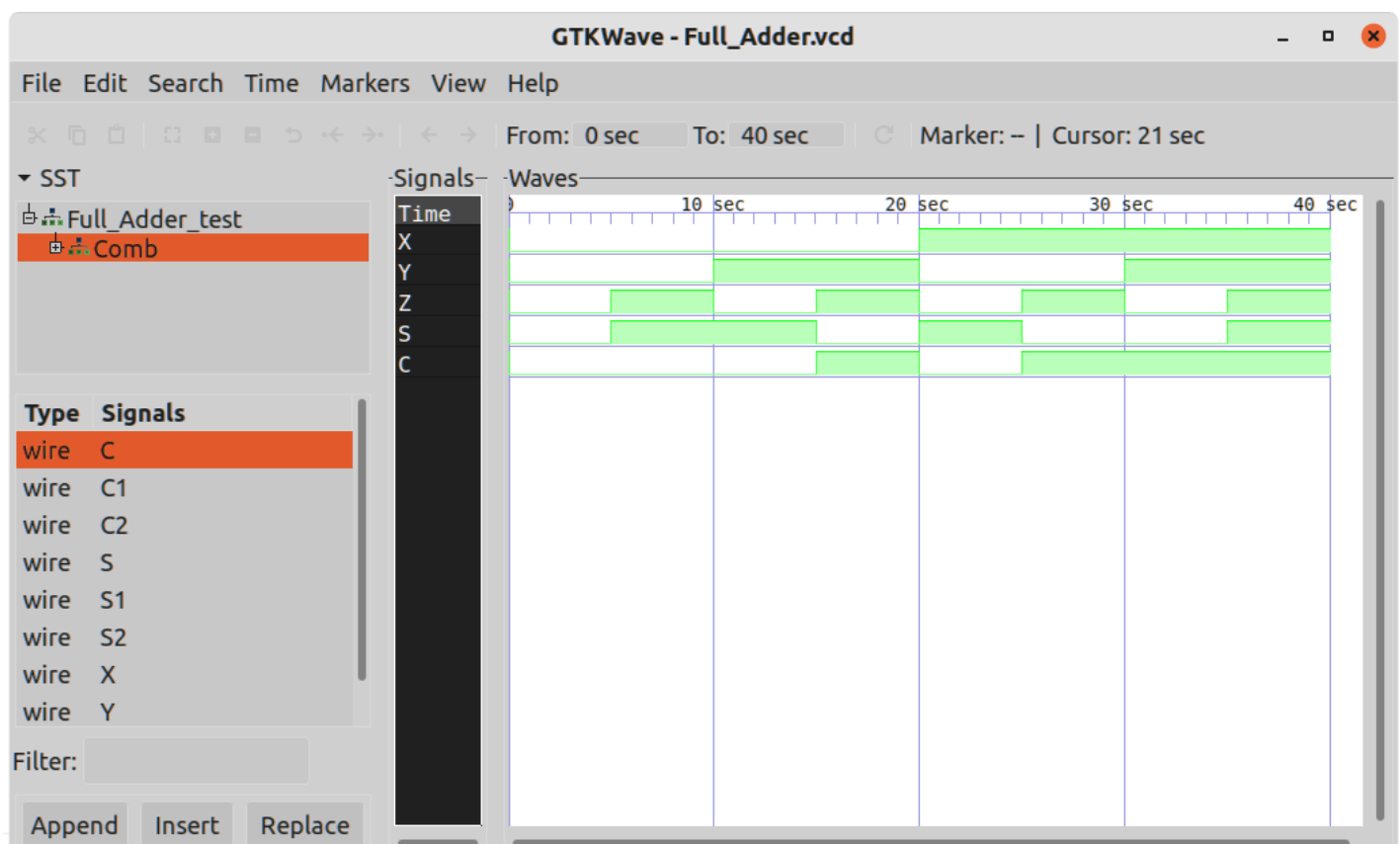
Terminal
~/Desktop/VLSI_Lab/Assignment_1/ADDER.v :: $main == ~26
iverilog -o Full_Adder Full_Adder.v Full_Adder_test.v
~/Desktop/VLSI_Lab/Assignment_1/ADDER.v :: $main == ~27
vvp Full_Adder
VCD info: dumpfile Full_Adder.vcd opened for output.
0 + 0 + 0 -> 0 0
0 + 0 + 1 -> 0 1
0 + 1 + 0 -> 0 1
0 + 1 + 1 -> 1 0
1 + 0 + 0 -> 0 1
1 + 0 + 1 -> 1 0
1 + 1 + 0 -> 1 0
1 + 1 + 1 -> 1 1
~/Desktop/VLSI_Lab/Assignment_1/ADDER.v :: $main == ~28
```

(from left to right) Verilog code for Full Adder and test bench

- **GTKWave Output**



gtkWAVE output of Half Adder Circuit



gtkWAVE output of Full Adder Circuit

# Observations (Half Subtractor and Full Subtractor):

- RTL Source Code

The screenshot shows the Visual Studio Code interface with two files open: `Half_Subtractor.v` and `Half_Subtractor_test.v`. The `Half_Subtractor.v` file contains the following Verilog code:

```
1 module Half_Subtractor(input X,Y,output D,B);
2     assign D = X ^ Y;
3     assign B = (~X) & Y;
4 endmodule
```

The `Half_Subtractor_test.v` file contains the following Verilog code:

```
1 module Half_Subtractor_test();
2     reg X,Y;
3     wire B,D;
4     Half_Subtractor Comb(.X(X),.Y(Y),.B(B),.D(D));
5     initial begin
6         $dumpfile("Half_Subtractor.vcd");
7         $dumpvars(0,Comb);
8         $monitor("%d - %d -> %d %d",X,Y,B,D);
9         X=0; Y=0; #5;
10        X=0; Y=1; #5;
11        X=1; Y=0; #5;
12        X=1; Y=1; #5;
13        $finish;
14    end
15 endmodule
```

The terminal output shows the execution of the test bench:

```
~/Desktop/VLSI_Lab/Assignment_1/Subtractor :: $main == 28
~verilog -o Half_Subtractor Half_Subtractor.v Half_Subtractor_test.v
~/Desktop/VLSI_Lab/Assignment_1/Subtractor :: $main == 30
~vvp Half_Subtractor
VCD info: dumpfile Half_Subtractor.vcd opened for output.
0 - 0 -> 0 0
0 - 1 -> 1 1
1 - 0 -> 0 1
1 - 1 -> 0 0
~/Desktop/VLSI_Lab/Assignment_1/Subtractor :: $main == 31
```

(from left to right) Verilog code for Half Subtractor and test bench

The screenshot shows the Visual Studio Code interface with two files open: `Full_Subtractor.v` and `Full_Subtractor_test.v`. The `Full_Subtractor.v` file contains the following Verilog code:

```
1 `include "Half_Subtractor.v"
2 module Full_Subtractor(input X,Y,Z,output D,B);
3     wire D1,D2,B1,B2;
4     Half_Subtractor HS1(.X(X),.Y(Y),.B(B1),.D(D1));
5     Half_Subtractor HS2(.X(D1),.Y(Z),.B(B2),.D(D2));
6     assign B = B1 | B2;
7     assign D = D2;
8 endmodule
```

The `Full_Subtractor_test.v` file contains the following Verilog code:

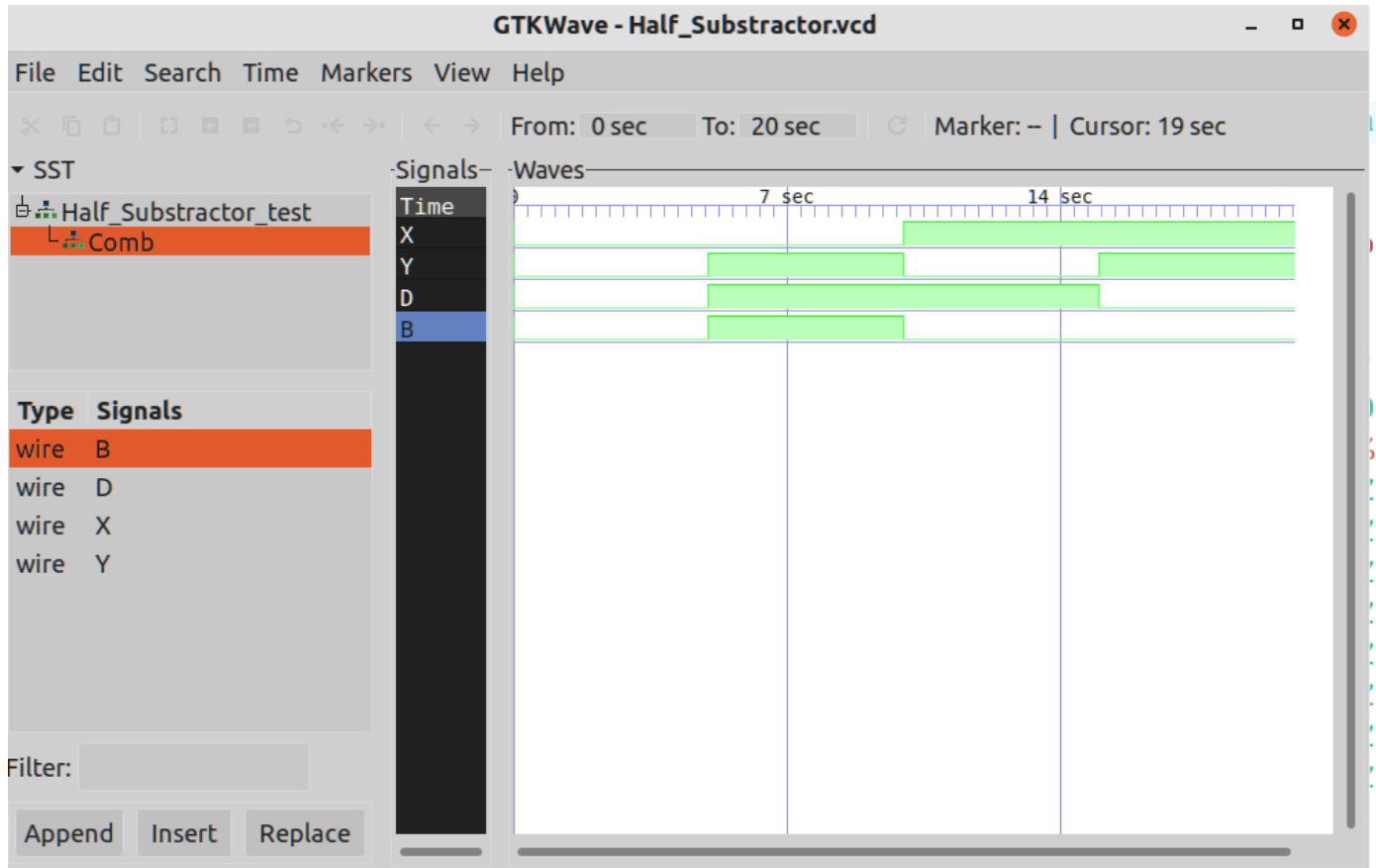
```
1 module Full_Subtractor_test();
2     reg X,Y,Z;
3     wire D,B;
4     Full_Subtractor Comb(.X(X),.Y(Y),.Z(Z),.D(D),.B(B));
5     initial begin
6         $dumpfile("Full_Subtractor.vcd");
7         $dumpvars(0,Comb);
8         $monitor("%d - %d - %d -> %d %d",X,Y,Z,B,D);
9         X=0; Y=0; Z=0; #5;
10        X=0; Y=0; Z=1; #5;
11        X=0; Y=1; Z=0; #5;
12        X=0; Y=1; Z=1; #5;
13        X=1; Y=0; Z=0; #5;
14        X=1; Y=0; Z=1; #5;
15        X=1; Y=1; Z=0; #5;
16        X=1; Y=1; Z=1; #5;
17        $finish;
18    end
19 endmodule
```

The terminal output shows the execution of the test bench:

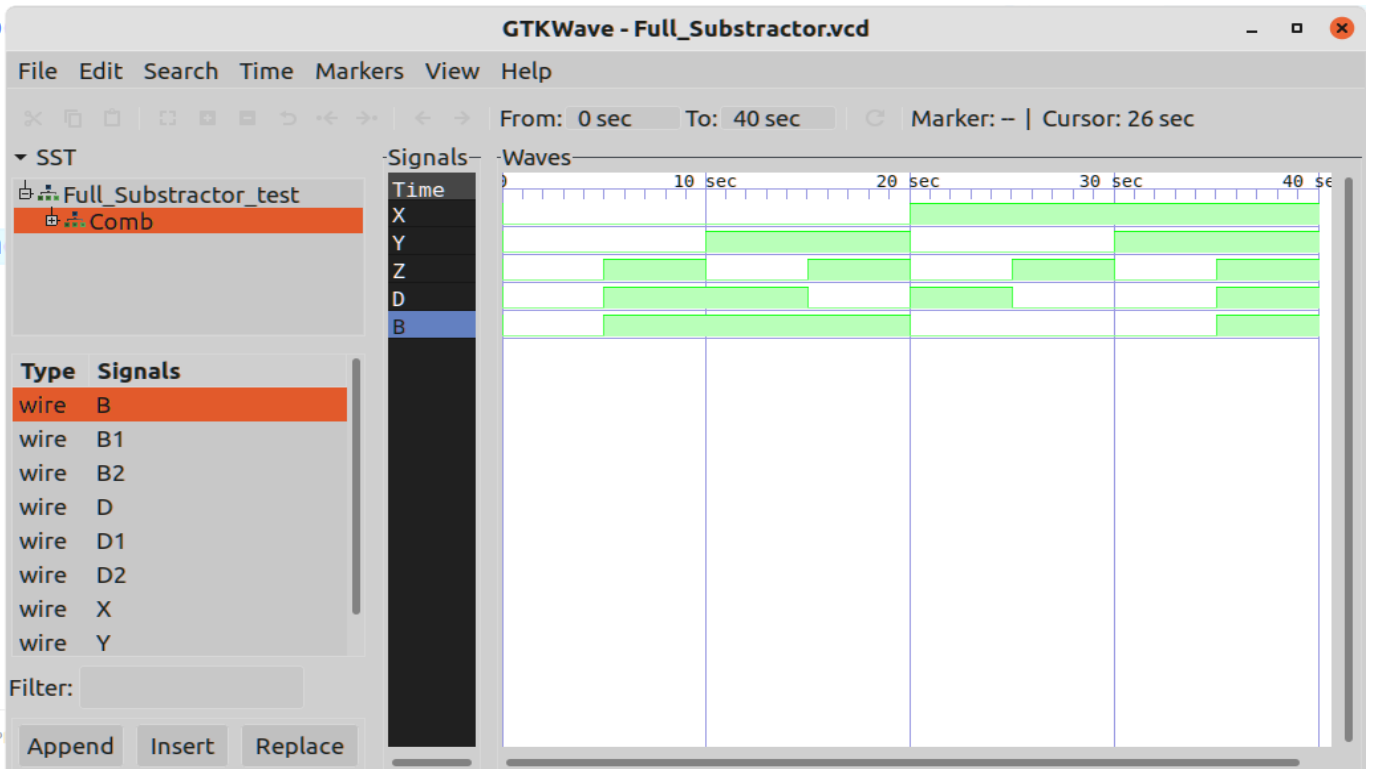
```
~/Desktop/VLSI_Lab/Assignment_1/Subtractor :: $main == 34
~verilog -o Full_Subtractor Full_Subtractor.v Full_Subtractor_test.v
~/Desktop/VLSI_Lab/Assignment_1/Subtractor :: $main == 34
~vvp Full_Subtractor
VCD info: dumpfile Full_Subtractor.vcd opened for output.
0 - 0 -> 0 0
0 - 0 -> 1 1
0 - 1 -> 1 1
0 - 1 -> 1 0
1 - 0 -> 0 1
1 - 0 -> 1 0
1 - 1 -> 0 0
1 - 1 -> 0 0
1 - 1 -> 1 1
```

(from left to right) Verilog code for Full Subtractor and test bench

- **GTKWave Output**



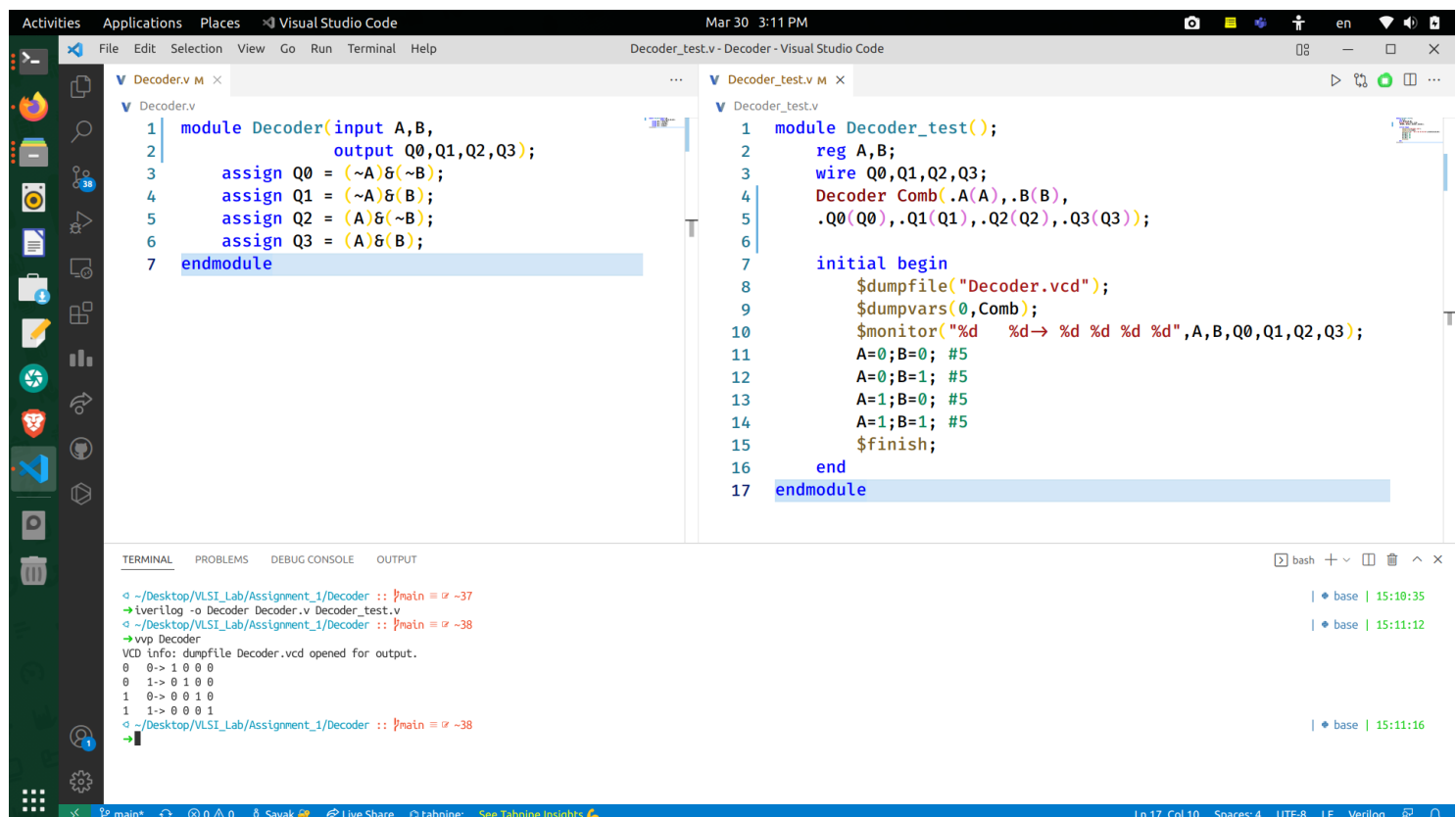
gtkWAVE output of Half Subtractor Circuit



gtkWAVE output of Full Subtractor Circuit

# Observations (2:4 Decoder):

- RTL Source Code



The screenshot shows the Visual Studio Code interface with two Verilog files open: `Decoder.v` and `Decoder_test.v`. The `Decoder.v` file contains the logic for a 2:4 decoder, and the `Decoder_test.v` file contains a test bench to verify its functionality. The terminal at the bottom shows the execution of the Verilog code using Icarus Verilog, including the generation of a VCD file and the simulation results.

```
1 module Decoder(input A,B,
2                 output Q0,Q1,Q2,Q3);
3     assign Q0 = (~A)&(~B);
4     assign Q1 = (~A)&(B);
5     assign Q2 = (A)&(~B);
6     assign Q3 = (A)&(B);
7 endmodule

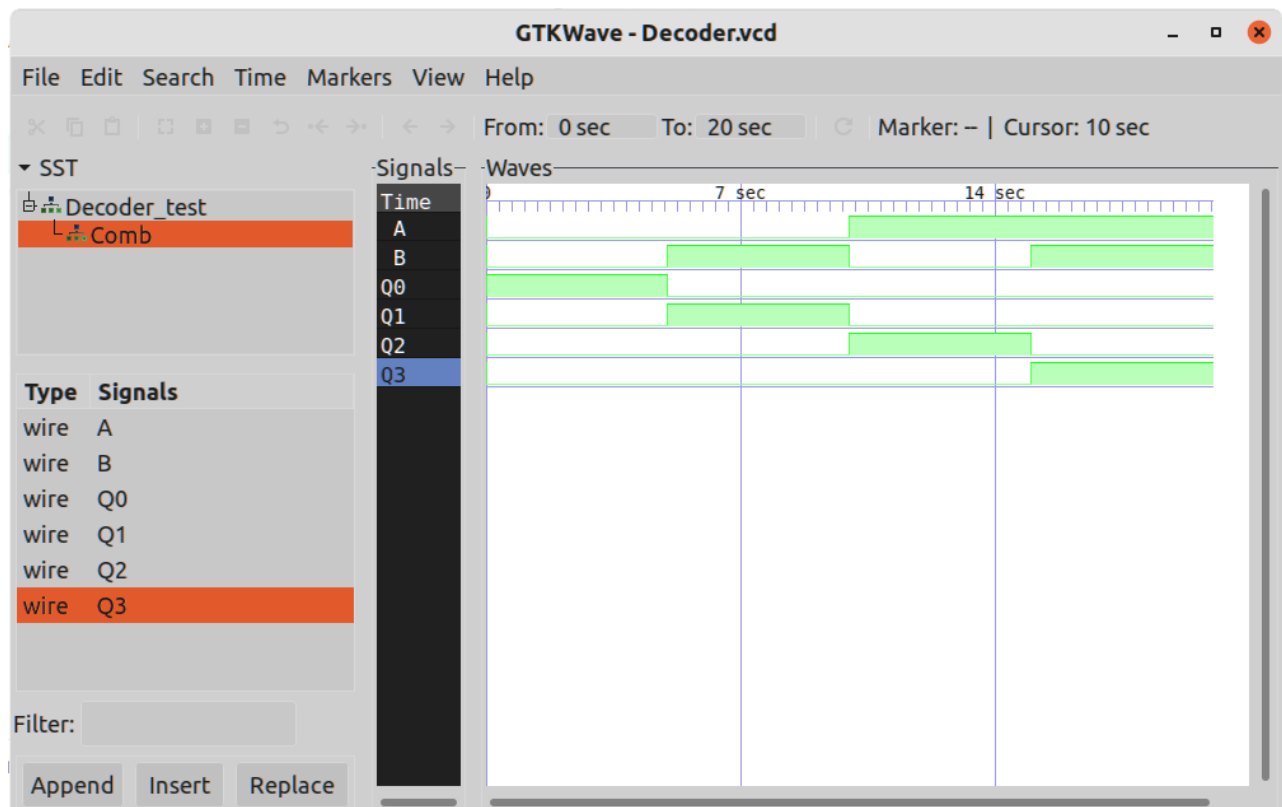
1 module Decoder_test();
2     reg A,B;
3     wire Q0,Q1,Q2,Q3;
4     Decoder Comb(.A(A),.B(B),
5                 .Q0(Q0),.Q1(Q1),.Q2(Q2),.Q3(Q3));
6
7     initial begin
8         $dumpfile("Decoder.vcd");
9         $dumpvars(0,Comb);
10        $monitor("%d %d -> %d %d %d %d",A,B,Q0,Q1,Q2,Q3);
11        A=0;B=0; #5
12        A=0;B=1; #5
13        A=1;B=0; #5
14        A=1;B=1; #5
15        $finish;
16    end
17 endmodule
```

Terminal Output:

```
~/Desktop/VLSI_Lab/Assignment_1/Decoder :: $main = 37
iverilog -o Decoder Decoder.v Decoder_test.v
~/Desktop/VLSI_Lab/Assignment_1/Decoder :: $main = 38
vvp Decoder
VCD info: dumpfile Decoder.vcd opened for output.
0 0-> 1 0 0 0
0 1-> 0 1 0 0
1 0-> 0 0 1 0
1 1-> 0 0 0 1
~/Desktop/VLSI_Lab/Assignment_1/Decoder :: $main = 38
```

(from left to right) Verilog code for 2:4 Decoder and test bench

- GTKWave Output



gtkWAVE output of 2:4 Decoder Circuit

# Observations (8:3 Encoder):

- RTL Source Code

```
1 module Encoder(input Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,
2               output A,B,C);
3     assign A = (Q4 | Q5) | (Q6 | Q7);
4     assign B = (Q2 | Q3) | (Q6 | Q7);
5     assign C = (Q1 | Q3) | (Q5 | Q7);
6 endmodule

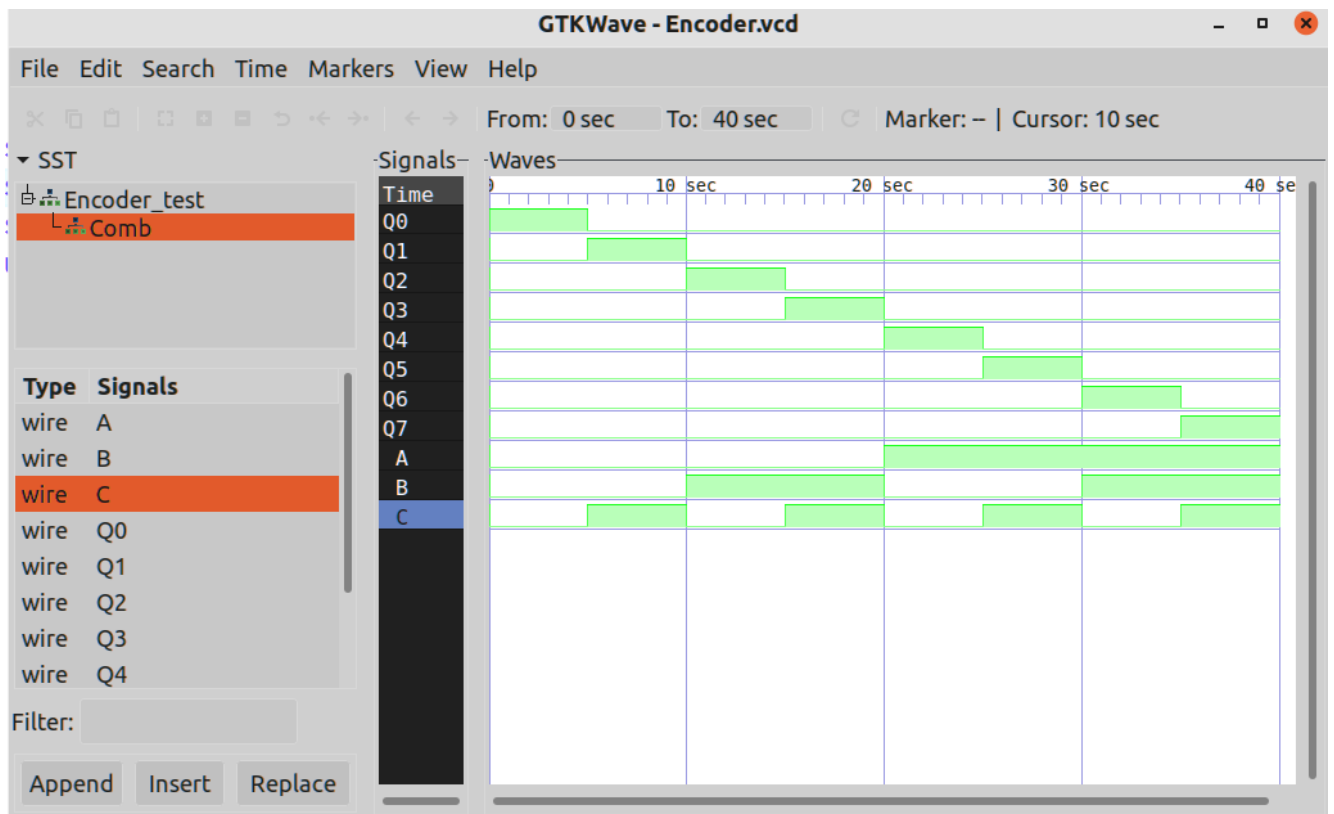
1 module Encoder_test();
2     reg Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7;
3     wire A,B,C;
4     Encoder Comb(.Q0(Q0),.Q1(Q1),.Q2(Q2),.Q3(Q3),.Q4(Q4),.Q5
5     (Q5),.Q6(Q6),.Q7(Q7),.A(A),.B(B),.C(C));
6     initial begin
7         $dumpfile("Encoder.vcd");
8         $dumpvars(0,Comb);
9         $monitor("%d %d %d %d %d %d %d %d → %d %d %d",
10        Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0,A,B,C);
11        Q7=0;Q6=0;Q5=0;Q4=0;Q3=0;Q2=0;Q1=0;Q0=1; #5
12        Q7=0;Q6=0;Q5=0;Q4=0;Q3=0;Q2=1;Q1=0;Q0=0; #5
13        Q7=0;Q6=0;Q5=0;Q4=0;Q3=1;Q2=0;Q1=0;Q0=0; #5
14        Q7=0;Q6=0;Q5=0;Q4=1;Q3=0;Q2=0;Q1=0;Q0=0; #5
15        Q7=0;Q6=0;Q5=1;Q4=0;Q3=0;Q2=0;Q1=0;Q0=0; #5
16        Q7=0;Q6=1;Q5=0;Q4=0;Q3=0;Q2=0;Q1=0;Q0=0; #5
17        Q7=1;Q6=0;Q5=0;Q4=0;Q3=0;Q2=0;Q1=0;Q0=0; #5
18    end
19 endmodule
```

TERMINAL

```
→iverilog -o Encoder Encoder.v Encoder_test.v
→./Desktop/VLSI_Lab/Assignment_1/Encoder :: main = 0 -41
→vvp Encoder
VCD info: dumpfile Encoder.vcd opened for output.
0 0 0 0 0 0 0 1 -> 0 0 0
0 0 0 0 0 0 1 0 -> 0 0 1
0 0 0 0 0 1 0 0 -> 0 1 0
0 0 0 0 1 0 0 0 -> 0 1 1
0 0 0 1 0 0 0 0 -> 1 0 0
0 0 1 0 0 0 0 0 -> 1 0 1
0 1 0 0 0 0 0 0 -> 1 1 0
1 0 0 0 0 0 0 0 -> 1 1 1
```

(from left to right) Verilog code for 8:3 Encoder and test bench

- GTKWave Output



gtkWAVE output of 8:3 Encoder Circuit