ECS 409/609
Assignment 2
Behavioral Based Assignment Solutions

Kunwar Arpit Singh

Submission Deadline: September 10, 2025

For code repository of this assignment: **Verilog Assignments**

---

1. Problem 1

Problem statement

Write behavioral verilog code to multiply a 4-bit input by 2.

Verilog source (Behavioral)

```
module multiply_4_bit_by_2 (input [3:0] A, output [4:0] out);
  assign out = A << 1;
endmodule
```

Testbench

```
module multiply_4_bit_by_2_tb;
    reg [3:0] A;
    wire [4:0] out;
    reg [8*24-1:0] name;

    multiply_4_bit_by_2 dut (.A(A), .out(out));

    initial begin
        $display("Kunwar␣Arpit␣Singh");
        name = "␣Kunwar␣Arpit␣Singh␣22185";
        $dumpfile("assign2_problem1_4_bit_multiply_by_2.vcd");
        $dumpvars(1, multiply_4_bit_by_2_tb);
        $display("Kunwar␣Arpit␣Singh");

        $display("A␣(Binary␣&␣Decimal)␣|␣Output(Binary)␣|␣Output(Decimal)");
        $display("-----------------------------------------");

        A = 4'b0000; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0001; #10;
```

```
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0010; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0011; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0100; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0101; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0110; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0111; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1000; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1001; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1010; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1011; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1100; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1101; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1110; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1111; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        $finish;
    end
endmodule
```

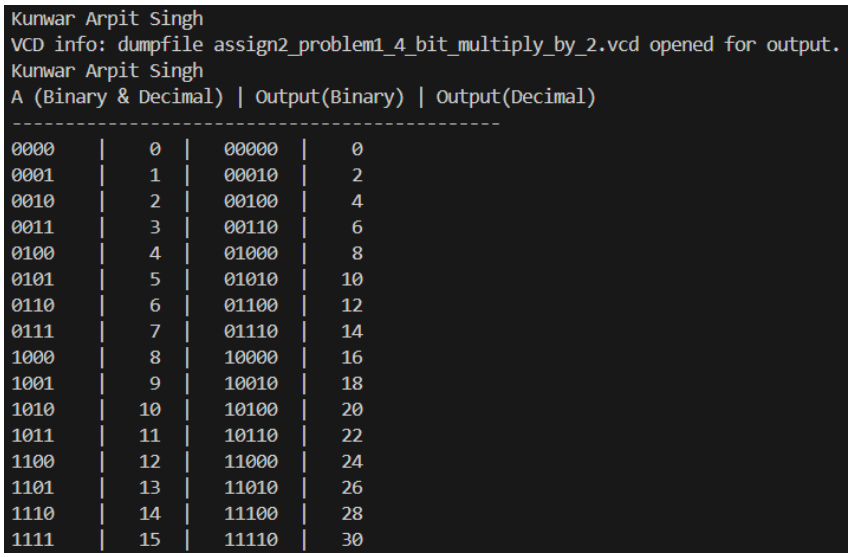
Output in Terminal

```
VCD info: dumpfile assign2_problem1_4_bit_multiply_by_2_tb.vcd opened for
    output.
Kunwar Arpit Singh
A (Binary & Decimal) | Output(Binary) | Output(Decimal)
---------------------------------------------
0000 | 0 | 00000 | 0
0001 | 1 | 00010 | 2
0010 | 2 | 00100 | 4
0011 | 3 | 00110 | 6
0100 | 4 | 01000 | 8
0101 | 5 | 01010 | 10
0110 | 6 | 01100 | 12
0111 | 7 | 01110 | 14
1000 | 8 | 10000 | 16
1001 | 9 | 10010 | 18
```

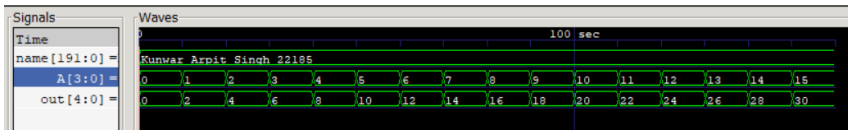
```
1010 | 10 | 10100 | 20
1011 | 11 | 10110 | 22
1100 | 12 | 11000 | 24
1101 | 13 | 11010 | 26
1110 | 14 | 11100 | 28
1111 | 15 | 11110 | 30
```

Output Screenshots

Terminal output



GTKWave waveform



2. Problem 2

Problem statement

Implement a behavioral verilog code for a 4-to-2 encoder and 2-to-4 decoder.

Verilog source (Behavioral)

4-to-2 Encoder:

```
module encoder_4_to_2 (input [3:0] A, output [1:0] out);
    assign out = (A == 4'b1000) ? 2'b11 :
                 (A == 4'b0100) ? 2'b10 :
                 (A == 4'b0010) ? 2'b01 :
                 (A == 4'b0001) ? 2'b00 : 2'bxx;
endmodule
```

2-to-4 Decoder:

```
module decoder_2_to_4(input [1:0] A, output [3:0] out);
    assign out = (A == 2'b00) ? 4'b0001 :
                 (A == 2'b01) ? 4'b0010 :
                 (A == 2'b10) ? 4'b0100 :
                 (A == 2'b11) ? 4'b1000 : 4'bxxxx;
endmodule
```

Testbench

4-to-2 Encoder:
This testbench shows two tables one for only the valid cases for 4 to 2 encoder and the second for all the cases for 4 to 2 encoder.

```
module encoder_4_to_2_tb;
    reg [3:0] A;
    wire [1:0] out;
    reg [8*24-1:0] name;

    encoder_4_to_2 dut (.A(A), .out(out));

    initial begin
        name = "␣Kunwar␣Arpit␣Singh␣22185";
        $dumpfile("assign2_problem2_encoder_4_to_2.vcd");
        $dumpvars(1, encoder_4_to_2_tb);
        $display("Kunwar␣Arpit␣Singh");
        $display("Output␣for␣only␣the␣valid␣cases␣for␣4␣to␣2␣encoder");
        $display("A␣(Binary␣&␣Decimal)␣|␣Output(Binary)␣|␣Output(Decimal)");
        $display("-----------------------------------------------");
        A = 4'b0001; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0010; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0100; #10;
```

```
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1000; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        $display("\nOutput␣for␣all␣the␣cases␣for␣4␣to␣2␣encoder");
        $display("A␣(Binary␣&␣Decimal)␣|␣Output(Binary)␣|␣Output(Decimal)");
        $display("-------------------------------------------");
        A = 4'b0000; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0001; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0010; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0011; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0100; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0101; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0110; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b0111; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1000; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1001; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1010; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1011; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1100; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1101; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1110; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 4'b1111; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        $finish;
    end
endmodule
```

2-to-4 Decoder:

```
module decoder_2_to_4_tb;
    reg[1:0] A;
    wire[3:0] out;
    reg[8*24-1:0] name;

    decoder_2_to_4 dut (.A(A), .out(out));

    initial begin
```

```
        name = "␣Kunwar␣Arpit␣Singh␣22185";
        $display("Kunwar␣Arpit␣Singh");
        $dumpfile("assign2_problem2_decoder_2_to_4.vcd");
        $dumpvars(1, decoder_2_to_4_tb);
        $display("Kunwar␣Arpit␣Singh");
        $display("A␣(Binary␣&␣Decimal)␣|␣Output(Binary)␣|␣Output(Decimal)");
        $display("-----------------------------------------------");
        A = 2'b00; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 2'b01; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 2'b10; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        A = 2'b11; #10;
        $display("%b␣␣␣␣|␣␣␣%d␣␣|␣␣␣%b␣␣|␣␣␣%d", A, A, out, out);
        $finish;
    end
endmodule
```

Output in Terminal

4-to-2 Encoder:

```
VCD info: dumpfile assign2_problem2_encoder_4_to_2.vcd opened for output.
Kunwar Arpit Singh
Output for only the valid cases for 4 to 2 encoder
A (Binary & Decimal) | Output(Binary) | Output(Decimal)
-----------------------------------------------
0001 | 1 | 00 | 0
0010 | 2 | 01 | 1
0100 | 4 | 10 | 2
1000 | 8 | 11 | 3

Output for all the cases for 4 to 2 encoder
A (Binary & Decimal) | Output(Binary) | Output(Decimal)
-----------------------------------------------
0000 | 0 | xx | x
0001 | 1 | 00 | 0
0010 | 2 | 01 | 1
0011 | 3 | xx | x
0100 | 4 | 10 | 2
0101 | 5 | xx | x
0110 | 6 | xx | x
0111 | 7 | xx | x
1000 | 8 | 11 | 3
1001 | 9 | xx | x
1010 | 10 | xx | x
1011 | 11 | xx | x
1100 | 12 | xx | x
1101 | 13 | xx | x
1110 | 14 | xx | x
```

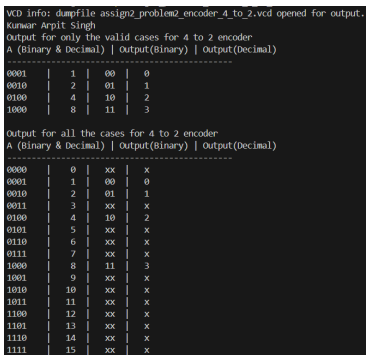
```
1111 | 15 | xx | x
```

2-to-4 Decoder:

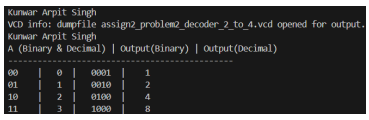
```
Kunwar Arpit Singh
VCD info: dumpfile assign2_problem2_decoder_2_to_4.vcd opened for output.
Kunwar Arpit Singh
A (Binary & Decimal) | Output(Binary) | Output(Decimal)
-----------------------------------------------
00 | 0 | 0001 | 1
01 | 1 | 0010 | 2
10 | 2 | 0100 | 4
11 | 3 | 1000 | 8
```

Output Screenshots

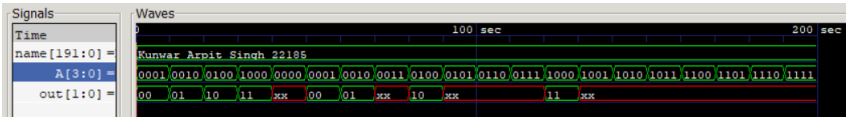
Terminal output (4-to-2 Encoder)



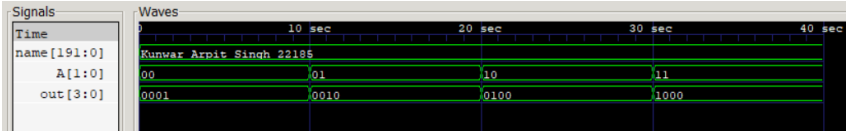
Terminal output (2-to-4 Decoder)



GTKWave waveform (4-to-2 Encoder)



GTKWave waveform (2-to-4 Decoder)



3. Problem 3

Problem statement

Design an 8-to-1 multiplexer using a case statement in behavioral verilog.

Verilog source (Behavioral)

```
module multiplexer_8_to_1(input [7:0] A, input [2:0] sel, output out);
    assign out = (sel == 3'b000) ? A[0] :
                 (sel == 3'b001) ? A[1] :
                 (sel == 3'b010) ? A[2] :
                 (sel == 3'b011) ? A[3] :
                 (sel == 3'b100) ? A[4] :
                 (sel == 3'b101) ? A[5] :
                 (sel == 3'b110) ? A[6] :
                 (sel == 3'b111) ? A[7] :
                 1'bx;
endmodule
```

Testbench

```
module multiplexer_8_to_1_tb;
    reg [7:0] A;
    reg [2:0] sel;
    wire out;
    reg [8*24-1:0] name;

    multiplexer_8_to_1 uut (.A(A), .sel(sel), .out(out));

    initial begin
        name = "␣Kunwar␣Arpit␣Singh␣22185";
        $display("Kunwar␣Arpit␣Singh");
        $dumpfile("assign2_problem3_mutliplexer_8_to_1.vcd");
        $dumpvars(1, multiplexer_8_to_1_tb);
        $display("Kunwar␣Arpit␣Singh");
        A = 8'b11001010;
        $display("Output␣for␣all␣the␣cases");
        $display("A␣(Binary␣&␣Decimal)␣|␣Output(Binary)␣|␣Output(Decimal)");
        $display("------------------------");
        sel = 3'b000; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b001; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b010; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b011; #10;
```

```
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b100; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b101; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b110; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b111; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);

        A = 8'b00110101;
        $display("Output␣for␣all␣the␣cases");
        $display("A␣(Binary␣&␣Decimal)␣|␣Output(Binary)␣|␣Output(Decimal)");
        $display("------------------------");
        sel = 3'b000; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b001; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b010; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b011; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b100; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b101; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b110; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        sel = 3'b111; #10;
        $display("%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣|␣␣%b␣␣|␣␣%d␣␣", A, A, sel, sel, out
            , out);
        $finish;
    end
endmodule
```

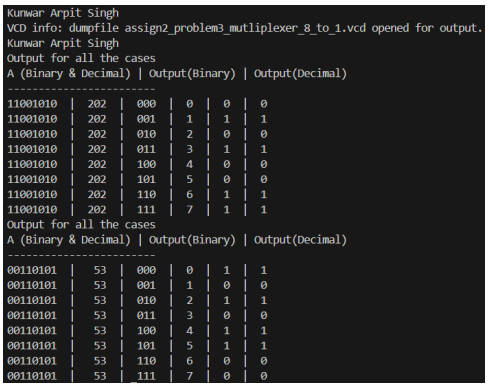
Output in Terminal

```
Kunwar Arpit Singh
```

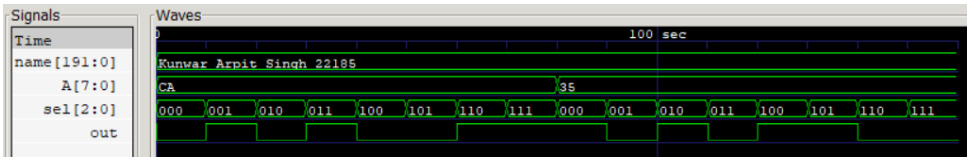
```
VCD info: dumpfile assign2_problem3_mutliplexer_8_to_1.vcd opened for output.
Kunwar Arpit Singh
Output for all the cases
A (Binary & Decimal) | Output(Binary) | Output(Decimal)
------------------------
11001010 | 202 | 000 | 0 | 0 | 0
11001010 | 202 | 001 | 1 | 1 | 1
11001010 | 202 | 010 | 2 | 0 | 0
11001010 | 202 | 011 | 3 | 1 | 1
11001010 | 202 | 100 | 4 | 0 | 0
11001010 | 202 | 101 | 5 | 0 | 0
11001010 | 202 | 110 | 6 | 1 | 1
11001010 | 202 | 111 | 7 | 1 | 1
Output for all the cases
A (Binary & Decimal) | Output(Binary) | Output(Decimal)
------------------------
00110101 | 53 | 000 | 0 | 1 | 1
00110101 | 53 | 001 | 1 | 0 | 0
00110101 | 53 | 010 | 2 | 1 | 1
00110101 | 53 | 011 | 3 | 0 | 0
00110101 | 53 | 100 | 4 | 1 | 1
00110101 | 53 | 101 | 5 | 1 | 1
00110101 | 53 | 110 | 6 | 0 | 0
00110101 | 53 | 111 | 7 | 0 | 0
```

Output Screenshots

Terminal output (8-to-1 Multiplexer)



GTKWave waveform (8-to-1 Multiplexer)



4. Problem 4

Problem statement

Design an 8-to-33 priority encoder using the if else statement in behavioral Verilog.

Verilog source (Behavioral)

8-to-3 Priority Encoder

```
module priority_encoder_8_to_3(input [7:0] A, output reg [2:0] out, output reg
    valid);
    always @(*) begin
        valid = |A;
        if (A[7]) out = 3'b111;
        else if (A[6]) out = 3'b110;
        else if (A[5]) out = 3'b101;
        else if (A[4]) out = 3'b100;
        else if (A[3]) out = 3'b011;
        else if (A[2]) out = 3'b010;
        else if (A[1]) out = 3'b001;
        else if (A[0]) out = 3'b000;
        else out = 3'bxxx;
    end
endmodule
```

Testbench

8-to-3 Priority Encoder

```
module priority_encoder_8_to_3_tb_small;
    reg [7:0] A;
    wire [2:0] out;
    wire valid;
    reg[8*24-1:0] name;
    integer i;
    priority_encoder_8_to_3 dut (.A(A), .out(out), .valid(valid));

    initial begin
        name = "␣Kunwar␣Arpit␣Singh␣22185";
        $display("Kunwar␣Arpit␣Singh");
        $dumpfile("assign2_problem4_priority_encoder_8_to_3_small.vcd");
        $dumpvars(1, priority_encoder_8_to_3_tb_small);
        $display("Kunwar␣Arpit␣Singh");
        $display("Outputs␣for␣Single␣Active␣Inputs");
        $display("Output␣for␣No␣Active␣Input␣is")
        $display("A(Input)␣␣|␣␣␣out(Output)␣␣␣␣|␣␣␣␣Valid");
        A = 8'b00000000; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b00000001; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b00000010; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b00000100; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
```

```
        A = 8'b00001000; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b00010000; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b00100000; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b01000000; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b10000000; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);

        $display("Outputs␣for␣Multiple␣Active␣Inputs");
        $display("A(Input)␣␣|␣␣␣out(Output)␣␣␣␣|␣␣␣␣Valid");
        A = 8'b00000001; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b00000011; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b00000110; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b00001100; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b00010001; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b00101000; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b01010000; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        A = 8'b11111111; #10; $display("%b␣|␣%b␣|␣%b␣|", A, out, valid);
        $finish;
    end
endmodule
```

Output in Terminal

8-to-3 Priority Encoder

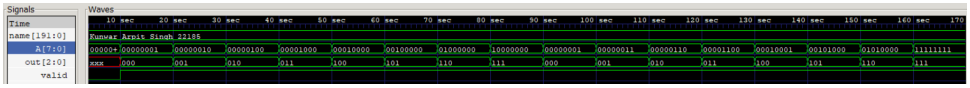
```
Kunwar Arpit Singh
VCD info: dumpfile assign2_problem4_priority_encoder_8_to_3_small.vcd opened
    for output.
Kunwar Arpit Singh
Outputs for Single Active Inputs
A(Input) | out(Output) | Valid
00000000 | xxx | 0 |
00000001 | 000 | 1 |
00000010 | 001 | 1 |
00000100 | 010 | 1 |
00001000 | 011 | 1 |
00010000 | 100 | 1 |
00100000 | 101 | 1 |
01000000 | 110 | 1 |
10000000 | 111 | 1 |
Outputs for Multiple Active Inputs
A(Input) | out(Output) | Valid
00000001 | 000 | 1 |
00000011 | 001 | 1 |
00000110 | 010 | 1 |
00001100 | 011 | 1 |
00010001 | 100 | 1 |
00101000 | 101 | 1 |
01010000 | 110 | 1 |
11111111 | 111 | 1 |
```

Output Screenshots

Terminal output (8-to-3 Priority Encoder)

```
Kunwar Arpit Singh
VCD info: dumpfile assign2_problem4_priority_encoder_8_to_3_small.vcd opened for output.
Kunwar Arpit Singh
Outputs for Single Active Inputs
A(Input)  |   out(Output)    |    Valid
00000000 | xxx | 0 |
00000001 | 000 | 1 |
00000010 | 001 | 1 |
00000100 | 010 | 1 |
00001000 | 011 | 1 |
00010000 | 100 | 1 |
00100000 | 101 | 1 |
01000000 | 110 | 1 |
10000000 | 111 | 1 |
Outputs for Multiple Active Inputs
A(Input)  |   out(Output)    |    Valid
00000001 | 000 | 1 |
00000011 | 001 | 1 |
00000110 | 010 | 1 |
00001100 | 011 | 1 |
00010001 | 100 | 1 |
00101000 | 101 | 1 |
01010000 | 110 | 1 |
11111111 | 111 | 1 |
```

GTKWave waveform (8-to-3 Priority Encoder)



5. Problem 5

Problem statement

Write behavioral Verilog code for a 4-bit Carry Look-Ahead Adder.

Verilog source (Behavioral)

4-bit Carry Look-Ahead Adder

```
module carry_look_ahead_adder_4_bit(input [3:0] A, input [3:0] B, input C_IN,
    output [3:0] SUM, output C_OUT);
    wire [3:0] P;
    wire [3:0] G;
    wire [3:0] C;

    assign P = A ^ B;
    assign G = A & B;

    assign C[0] = C_IN;
    assign C[1] = G[0] | (P[0] & C[0]);
    assign C[2] = G[1] | (P[1] & C[0]) | (P[1] & P[0] & C[0]);
    assign C[3] = G[2] | (P[2] & C[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] &
        P[0] & C[0]);
    assign C_OUT = G[3] | (P[3] & G[2]) | (P[3] & p[2] & G[1]) | (P[3] & P[2] &
         P[1] & G[0]) | (P[3] & P[2] & P[1] & P[0] & C[0]);

    assign SUM = P ^ C;
endmodule
```

Testbench

4-bit Carry Look-Ahead Adder

```
module carry_look_ahead_adder_4_bit_tb;
    reg [3:0] A;
    reg [3:0] B;
    reg C_IN;
    wire [3:0] SUM;
    wire C_OUT;
    reg [8*24-1:0] name;
    carry_look_ahead_adder_4_bit dut (.A(A), .B(B), .C_IN(C_IN), .SUM(SUM), .
        C_OUT(C_OUT));

    initial begin
        name = "␣Kunwar␣Arpit␣Singh␣22185";

        $display("Kunwar␣Arpit␣Singh");

        $dumpfile("assign2_problem5_carry_look_ahead_adder_4_bit.vcd");
        $dumpvars(1, carry_look_ahead_adder_4_bit_tb);
```

```
17
18          $display("Outputs␣for␣some␣sample␣input␣(in␣both␣Binary␣and␣Decimal)");
19
20          $display("A(B)|␣A(D)|␣B(D)␣|B(D)|␣␣Cin␣|␣Cout␣|␣Sum");
21          $display("---------------------------------");
22
23          A=4'b0101; B = 4'b0010; C_IN = 1'b0; #10;
24          $display("%04b␣|␣%d␣|␣%04b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%04b␣|␣%d␣|␣", A,
                A, B, B, C_IN, C_IN, C_OUT, C_OUT, SUM, SUM);
25
26          A=4'b1010; B = 4'b1000; C_IN = 1'b0; #10;
27          $display("%04b␣|␣%d␣|␣%04b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%04b␣|␣%d␣|␣", A,
                A, B, B, C_IN, C_IN, C_OUT, C_OUT, SUM, SUM);
28
29          A=4'b1001; B = 4'b0011; C_IN = 1'b1; #10;
30          $display("%04b␣|␣%d␣|␣%04b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%04b␣|␣%d␣|␣", A,
                A, B, B, C_IN, C_IN, C_OUT, C_OUT, SUM, SUM);
31
32          A=4'b1111; B = 4'b0000; C_IN = 1'b1; #10;
33          $display("%04b␣|␣%d␣|␣%04b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%04b␣|␣%d␣|␣", A,
                A, B, B, C_IN, C_IN, C_OUT, C_OUT, SUM, SUM);
34
35          A=4'b1111; B = 4'b1111; C_IN = 1'b1; #10;
36          $display("%04b␣|␣%d␣|␣%04b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%04b␣|␣%d␣|␣", A,
                A, B, B, C_IN, C_IN, C_OUT, C_OUT, SUM, SUM);
37
38          $finish;
39      end
40  endmodule
```

Output in Terminal

4-bit Carry Look-Ahead Adder

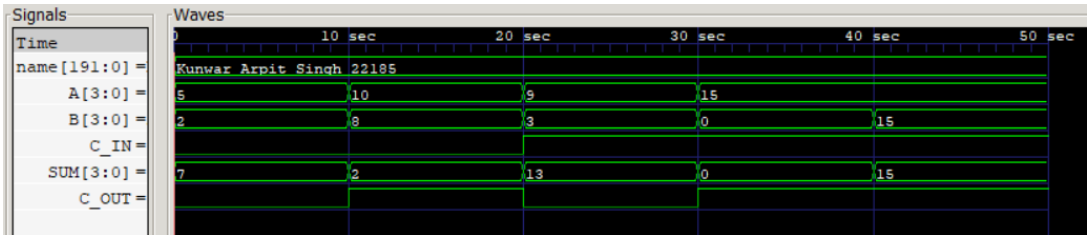
```
1  Kunwar Arpit Singh
2  VCD info: dumpfile assign2_problem5_carry_look_ahead_adder_4_bit.vcd opened
       for output.
3  Outputs for some sample input (in both Binary and Decimal)
4  A(B)| A(D)| B(D) |B(D)| Cin | Cout | Sum
5  ---------------------------------
6  0101 | 5 | 0010 | 2 | 0 | 0 | 0 | 0 | 0111 | 7 |
7  1010 | 10 | 1000 | 8 | 0 | 0 | 1 | 1 | 0010 | 2 |
8  1001 | 9 | 0011 | 3 | 1 | 1 | 0 | 0 | 1101 | 13 |
9  1111 | 15 | 0000 | 0 | 1 | 1 | 1 | 1 | 0000 | 0 |
10 1111 | 15 | 1111 | 15 | 1 | 1 | 1 | 1 | 1111 | 15 |
```

Output Screenshots

Terminal output (4-bit Carry Look-Ahead Adder)

```
Kunwar Arpit Singh
VCD info: dumpfile assign2_problem5_carry_look_ahead_adder_4_bit.vcd opened for output.
Outputs for some sample input (in both Binary and Decimal)
A(B)| A(D)| B(D) |B(D)|  Cin | Cout | Sum
--------------------------------
0101 |  5 | 0010 |  2 | 0 | 0 | 0 | 0 | 0111 |  7 |
1010 | 10 | 1000 |  8 | 0 | 0 | 1 | 1 | 0010 |  2 |
1001 |  9 | 0011 |  3 | 1 | 1 | 0 | 0 | 1101 | 13 |
1111 | 15 | 0000 |  0 | 1 | 1 | 1 | 1 | 0000 |  0 |
1111 | 15 | 1111 | 15 | 1 | 1 | 1 | 1 | 1111 | 15 |
```

GTKWave waveform (4-bit Carry Look-Ahead Adder)



6. Problem 6

Problem statement

Design behavioral Verilog code for a 4-bit Wallace Tree Multiplier.

Verilog source (Behavioral)

```
module wallace_tree_multiplier_4_bit(input [3:0] A, input [3:0] B, output
    [7:0] out);

    wire PP_00, PP_01, PP_02, PP_03;
    wire PP_10, PP_11, PP_12, PP_13;
    wire PP_20, PP_21, PP_22, PP_23;
    wire PP_30, PP_31, PP_32, PP_33;

    assign PP_00 = A[0] & B[0];
    assign PP_01 = A[0] & B[1];
    assign PP_02 = A[0] & B[2];
    assign PP_03 = A[0] & B[3];

    assign PP_10 = A[1] & B[0];
    assign PP_11 = A[1] & B[1];
    assign PP_12 = A[1] & B[2];
    assign PP_13 = A[1] & B[3];

    assign PP_20 = A[2] & B[0];
    assign PP_21 = A[2] & B[1];
    assign PP_22 = A[2] & B[2];
    assign PP_23 = A[2] & B[3];

    assign PP_30 = A[3] & B[0];
    assign PP_31 = A[3] & B[1];
    assign PP_32 = A[3] & B[2];
    assign PP_33 = A[3] & B[3];

    wire S1_1, C1_1;
    wire S2_1, C2_1;
    wire S3_1, C3_1;
    wire S4_1, C4_1;
    wire S5_1, C5_1;
    wire S6_1, C6_1;

    wire S2_2, C2_2;
    wire S3_2, C3_2;
    wire S4_2, C4_2;
    wire S5_2, C5_2;
    wire C6_sum_final, C6_carry_final;
    wire S3_temp, C3_temp, C3_temp2, C_to_C5;

    assign S1_1 = PP_01 ^ PP_10;
```

```
43      assign C1_1 = PP_01 & PP_10;
44
45      assign S2_1 = PP_02 ^ PP_11 ^ PP_20;
46      assign C2_1 = (PP_02 & PP_11) | (PP_11 & PP_20) | (PP_02 & PP_20);
47
48      assign S3_temp = PP_03 ^ PP_12 ^ PP_21;
49      assign C3_temp = (PP_03 & PP_12) | (PP_12 & PP_21) | (PP_03 & PP_21);
50      assign S3_1 = S3_temp ^ PP_30;
51      assign C3_temp2 = S3_temp & PP_30;
52      assign C3_1 = C3_temp ^ C3_temp2;
53      assign C_to_C5 = C3_temp & C3_temp2;
54
55      assign S4_1 = PP_13 ^ PP_22 ^ PP_31;
56      assign C4_1 = (PP_13 & PP_22) | (PP_22 & PP_31) | (PP_13 & PP_31);
57
58      assign S5_1 = PP_23 ^ PP_32;
59      assign C5_1 = PP_23 & PP_32;
60
61      assign S6_1 = PP_33;
62      assign C6_1 = 1'b0;
63
64      assign S2_2 = S2_1 ^ C1_1;
65      assign C2_2 = S2_1 & C1_1;
66
67      assign S3_2 = S3_1 ^ C2_1;
68      assign C3_2 = S3_1 & C2_1;
69
70      assign S4_2 = S4_1 ^ C3_1;
71      assign C4_2 = S4_1 & C3_1;
72
73      assign S5_2 = S5_1 ^ C4_1;
74      assign C5_2 = S5_1 & C4_1;
75
76      assign C6_sum_final = C5_1 ^ C5_2;
77      assign C6_carry_final = C5_1 & C5_2;
78
79      wire [7:0] VEC1, VEC2;
80
81      assign VEC1[0] = PP_00;
82      assign VEC1[1] = S1_1;
83      assign VEC1[2] = S2_2;
84      assign VEC1[3] = S3_2;
85      assign VEC1[4] = S4_2;
86      assign VEC1[5] = S5_2;
87      assign VEC1[6] = S6_1;
88      assign VEC1[7] = C6_carry_final;
89
90      assign VEC2[0] = 1'b0;
91      assign VEC2[1] = 1'b0;
92      assign VEC2[2] = 1'b0;
93      assign VEC2[3] = C2_2;
```

```
    assign VEC2[4] = C3_2;
    assign VEC2[5] = C4_2 | C_to_C5;
    assign VEC2[6] = C6_sum_final;
    assign VEC2[7] = 1'b0;

    assign out = VEC1 + VEC2;

endmodule
```

Testbench

```
module wallace_tree_multiplier_4_bit_tb;
    reg [3:0] A;
    reg [3:0] B;
    wire[7:0] out;
    reg [8*24-1:0] name;

    wallace_tree_multiplier_4_bit dut (.A(A), .B(B), .out(out));

    initial begin
        name = "␣Kunwar␣Arpit␣Singh␣22185";
        A = 0;
        B = 0;

        $display("Kunwar␣Arpit␣Singh");
        $dumpfile("assign2_problem6_wallace_tree_multiplier_4_bit.vcd");
        $dumpvars(1, wallace_tree_multiplier_4_bit_tb);
        $display("Kunwar␣Arpit␣Singh");
        $display("|␣A(B)␣|␣A(D)|␣B(B)␣|␣B(D)␣|␣Product(B)␣|␣Product(D)␣|");

        A = 4'b0001; B = 4'b0001; #10;
        $display("|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|", A, A, B, B, out, out);

        A = 4'b0101; B = 4'b1010; #10;
        $display("|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|", A, A, B, B, out, out);

        A = 4'b1111; B = 4'b1111; #10;
        $display("|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|", A, A, B, B, out, out);

        A = 4'b1001; B = 4'b0111; #10;
        $display("|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|", A, A, B, B, out, out);

        A = 4'b1100; B = 4'b0011; #10;
        $display("|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|", A, A, B, B, out, out);

        A = 4'b0100; B = 4'b1111; #10;
        $display("|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|", A, A, B, B, out, out);

        A = 4'b0000; B = 4'b1111; #10;
        $display("|␣%b␣|␣%d␣|␣%b␣|␣%d␣|␣%b␣|␣%d␣|", A, A, B, B, out, out);
```

```
40
41          $finish;
42      end
43  endmodule
```

Output in Terminal

```
1  Kunwar Arpit Singh
2  VCD info: dumpfile assign2_problem6_wallace_tree_multiplier_4_bit.vcd opened
       for output.
3  Kunwar Arpit Singh
4  | A(B) | A(D)| B(B) | B(D) | Product(B) | Product(D) |
5  | 0001 | 1 | 0001 | 1 | 00000001 | 1 |
6  | 0101 | 5 | 1010 | 10 | 00110010 | 50 |
7  | 1111 | 15 | 1111 | 15 | 11100001 | 225 |
8  | 1001 | 9 | 0111 | 7 | 00111111 | 63 |
9  | 1100 | 12 | 0011 | 3 | 00100100 | 36 |
10 | 0100 | 4 | 1111 | 15 | 00111100 | 60 |
11 | 0000 | 0 | 1111 | 15 | 00000000 | 0 |
```

Output Screenshots

Terminal output (4-bit Wallace Tree Multiplier)

```
Kunwar Arpit Singh
VCD info: dumpfile assign2_problem6_wallace_tree_multiplier_4_bit.vcd opened for output.
Kunwar Arpit Singh
| A(B) | A(D)| B(B) | B(D) | Product(B) | Product(D) |
| 0001 |  1 | 0001 |  1 | 00000001 |   1 |
| 0101 |  5 | 1010 | 10 | 00110010 |  50 |
| 1111 | 15 | 1111 | 15 | 11100001 | 225 |
| 1001 |  9 | 0111 |  7 | 00111111 |  63 |
| 1100 | 12 | 0011 |  3 | 00100100 |  36 |
| 0100 |  4 | 1111 | 15 | 00111100 |  60 |
| 0000 |  0 | 1111 | 15 | 00000000 |   0 |
```

GTKWave waveform (4-bit Wallace Tree Multiplier)