KIIT, Deemed to be University

School of Electronics Engineering

Digital System Design Laboratory [EC 29005]

Verilog Design code

Experiment 1:

SOP implementation of Boolean function using $F(A, B, C) = \sum(1,3,6,7)$ using NAND gates.

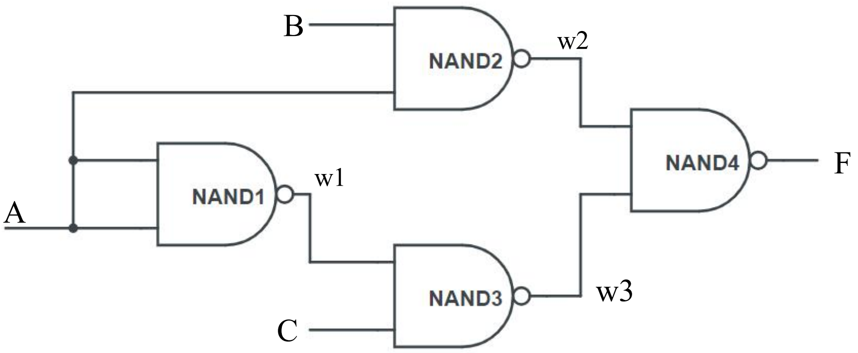


Figure 1: SOP implementation of Boolean function using NAND gates.

```
module SOP ( output F,  input A, input B, input C   );
        wire w1,w2,w3 ;
        nand  NAND1 (w1,A,A) ;
        nand  NAND2 (w2,A,B) ;
        nand  NAND3 (w3,w1,C) ;
        nand  NAND4 (F,w2,w3) ;
endmodule
```

POS implementation of Boolean function using $F(A, B, C) = \prod(0,2,4,5)$ using NOR gates.

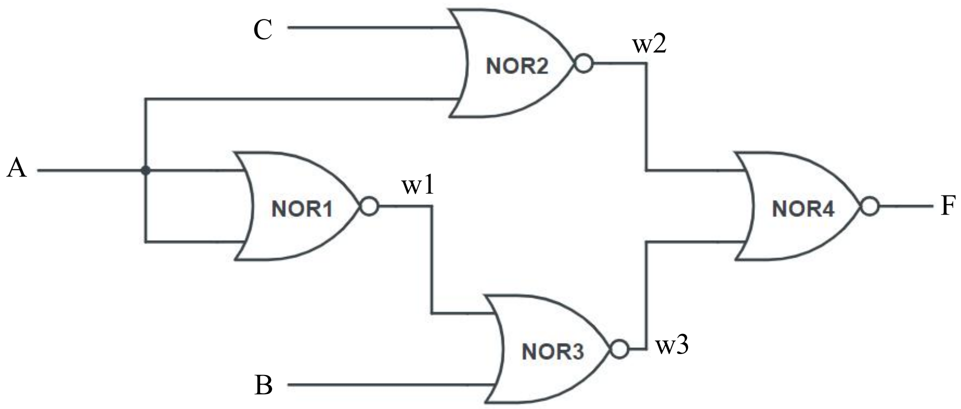


Figure 2: POS implementation of Boolean function using NOR gates.

```
module POS ( output F, input A, input B, input C  ) ;
        wire w1,w2,w3 ;
        nor  NOR1 (w1,A,A) ;
        nor  NOR2 (w2,A,C) ;
        nor  NOR3 (w3,w1,B) ;
        nor  NOR4 (F,w2,w3) ;
endmodule
```

Design Problem :

Warning indicator using NAND and NOR gates.

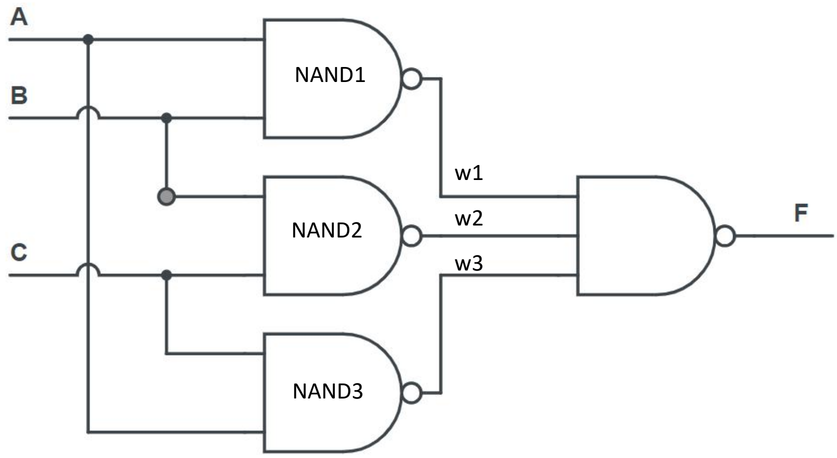


Figure 3: Implementation of warning indicator using NAND gates.

```
module   warning_indicator  ( output F, input A, input B, input C  );
        wire w1,w2,w3;
        nand  NAND1   (w1,A,B);
        nand  NAND2   (w2,B,C);
        nand  NAND3   (w3,A,C);
        nand  NAND4   (F,w1,w2,w3);
endmodule
```

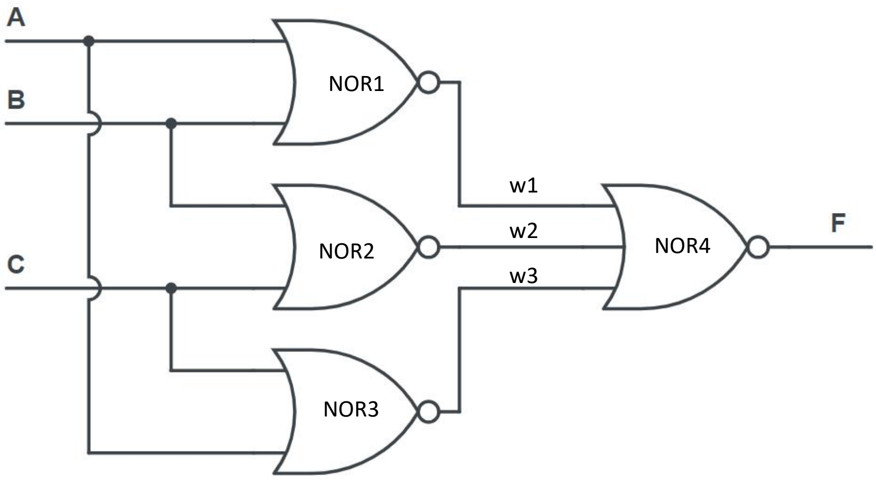


Figure 4: Implementation of warning indicator using NOR gates.

```
module   warning_indicator  ( output F, input A, input B, input C  );
         wire w1,w2,w3;
         nor  NOR1  (w1,A,B);
         nor  NOR2  (w2,B,C);
         nor  NOR3  (w3,A,C);
         nor  NOR4  (F,w1,w2,w3);
endmodule
```

Experiment 2:

Full Adder using Logic Gates

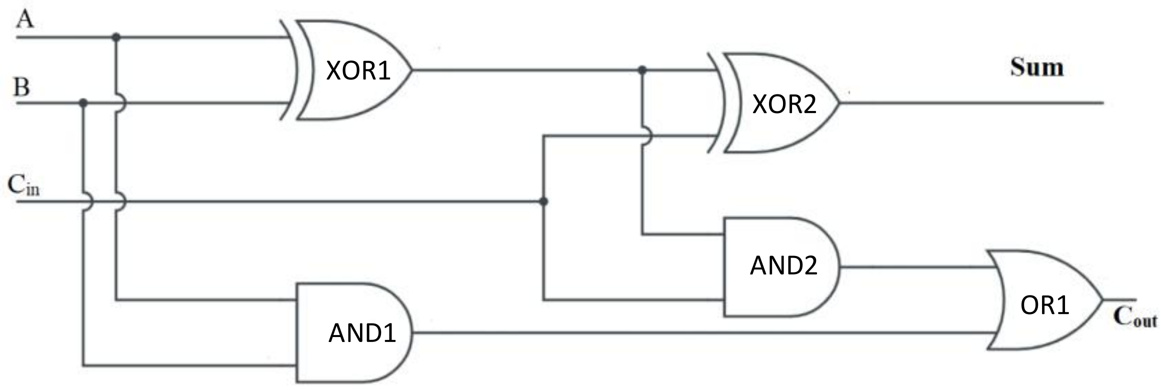


Figure 5: Full Adder using Logic Gates

```
module Full_Adder ( output Cout, output Sum, input A, input B,input Cin ) ;
        wire w0,w1,w2 ;
        xor   XOR1  (w0,A,B) ;
        and   AND1  (w1,A,B) ;
        xor  XOR2   (Sum,w0,Cin) ;
        and  AND2  (w2,w0,Cin) ;
        or    OR1    (Cout,w1,w2) ;
endmodule
```

Design Problem:

Full Subtractor using Logic Gates

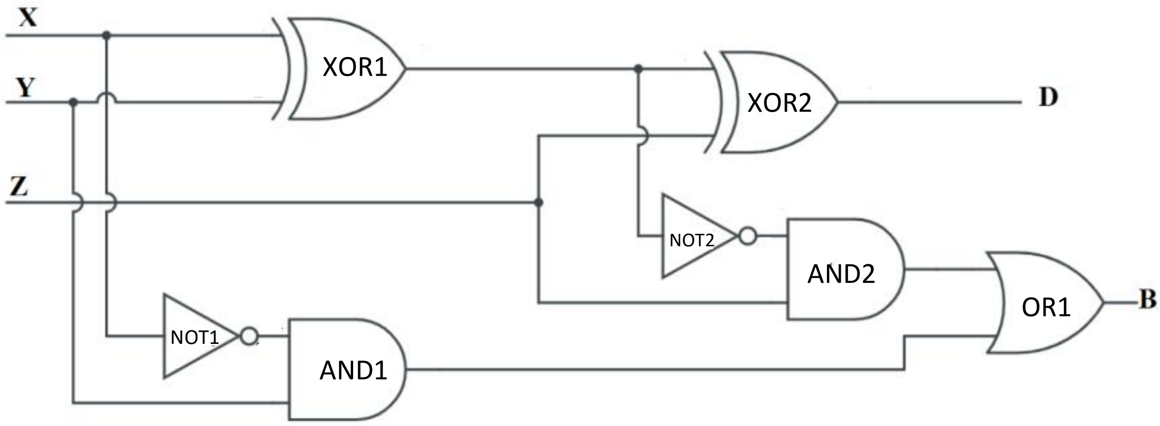


Figure 6: Full Subtractor using Logic Gates

```
module Full_Sub ( output B, input D, input X, input Y, input Z ) ;
        wire w0,w1,w2,w3,w4,w5 ;
        xor   XOR1  (w0,X,Y) ;
        not   NOT1  (w1,X) ;
        and   AND1  (w2,w1,Y) ;
        xor   XOR2  (D,w0,Z) ;
        not   NOT2  (w4,w0) ;
        and   AND2  (w5,w4,Z) ;
         or    OR1    (B,w2,w5) ;
endmodule
```

Experiment 3:

3 line to 8 line Decoder using Logic Gates

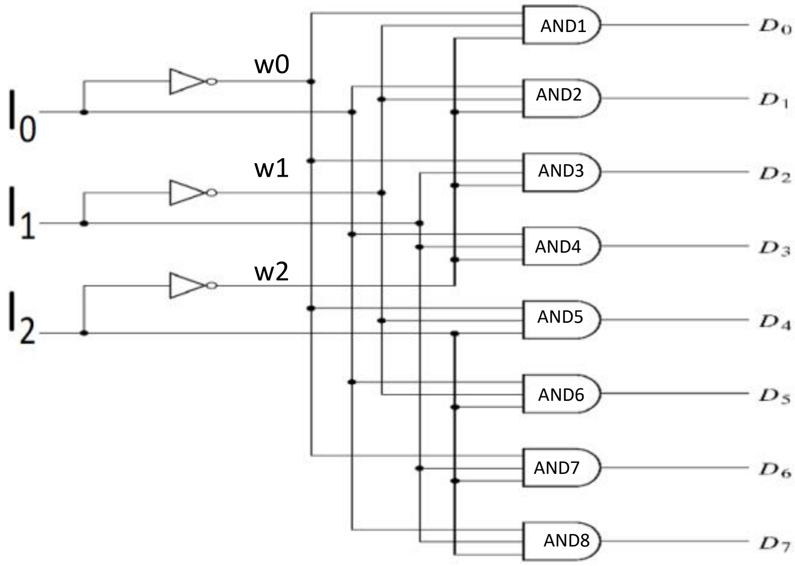


Figure 7: 3 line to 8 line Decoder using Logic Gates

```
module   Decoder_three_to_eight ( output D0, output D1, output D2, output D3,
                                  output D4, output D5, output D6, output D7,
                                  input I2, input I1, input I0  ) ;


        wire w0,w1,w2;
        not  NOT1  (w0,I0);
        not  NOT2  (w1,I1);
        not  NOT3  (w2,I2);

        and  AND1  (D0,w2,w1,w0);
        and  AND2  (D1,w2,w1,I0);
        and  AND3  (D2,w2,I1,w0);
        and  AND4  (D3,w2,I1,I0);
        and  AND5  (D4,I2,w1,w0);
        and  AND6  (D5,I2,w1,I0);
        and  AND7  (D6,I2,I1,w0);
        and  AND8  (D7,I2,I1,I0);
endmodule
```

Design Problem:

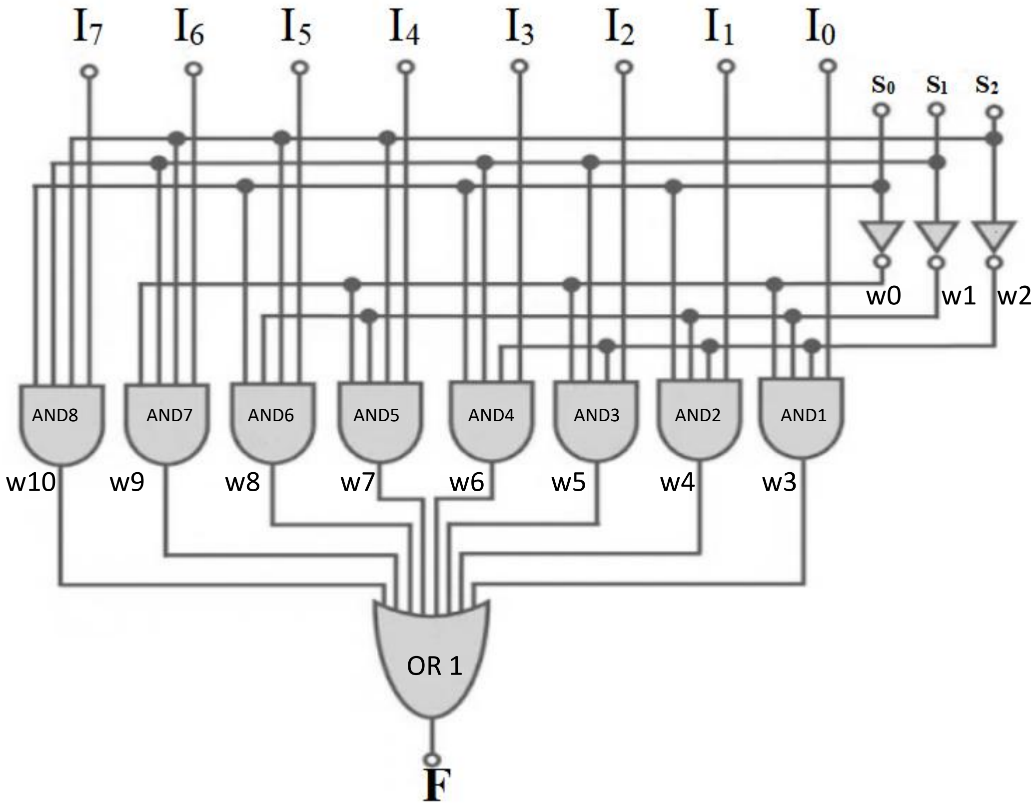
Implementation of 3 bit Binary to Gray Code-converter using Decoder.

```
module   Binary_to_Gray    ( output G2, output G1,output G0,input B2,input B1,input B0 ) ;
            wire    w0,w1,w2,w3,w4,w5,w6,w7 ;
            Decoder_three_to_eight  Decoder1  ( w0,w1,w2,w3,w4,w5,w6,w7,B2,B1,B0) ;
            or  OR1    (G2,w4,w5,w6,w7) ;
            or  OR2    (G1,w2,w3,w4,w5) ;
            or  OR3     (G0,w1,w2,w5,w6) ;
endmodule
```

Note: To simulate 3 bit Binary to Gray code-converter using 3:8 Decoder the module of 3:8 Decoder should be in the same project. Here the 3:8 Decoder module is called to implement the 3 bit Binary to Gray code-converter .

Experiment 4:

8 X 1 MUX using Logic Gates




```
module MUX_eight_to_one (  output F, input S2, input S1, input S0, input I7, input I6,
                           input I5, input I4, input I3, input I2, input I1, input I0  ) ;
        wire w0,w1,w2,w3,w4,w5,w6,w7,w8,w9,w10 ;
        not  NOT1 (w0,S0) ;
        not  NOT2 (w1,S1) ;
        not   NOT3 (w2,S2) ;
        and   AND1 (w3,I0,w2,w1,w0) ;
        and   AND2 (w4,I1,w2,w1,S0) ;
        and   AND3 (w5,I2,w2,S1,w0) ;
        and   AND4 (w6,I3,w2,S1,S0) ;
        and   AND5 (w7,I4,S2,w1,w0) ;
        and   AND6  (w8,I5,S2,w1,S0) ;
        and  AND7  (w9,I6,S2,S1,w0) ;
        and   AND8  (w10,I7,S2,S1,S0) ;
        or     OR1     (F,w3,w4,w5,w6,w7,w8,w9,w10) ;
endmodule
```

Design Problem:

4 X 1 MUX using the 2 X 1 MUX.

```
module two_to_one_mux  (output F,  input S0,  input I1,  input I0) ;

        wire w0,w1,w2;

        nand   NAND1  (w0,S0,S0) ;
        nand   NAND2  (w1,w0,I0) ;
        nand   NAND3  (w2,S0,I1) ;
        nand   NAND4  (F,w1,w2) ;

endmodule
```

Simulation of 4 X 1 MUX using 2 X 1 MUXs:

```
module  four_to_one_mux  ( output F, input S1,  input S0,  input I3,  input I2, input I1,
                                        input I0 ) ;
        wire w1,w2;

        two_to_one_mux   Mux1  (w1,S0,I1,I0) ;
        two_to_one_mux   Mux2  (w2,S0,I1,I0) ;
        two_to_one_mux   Mux3  (F,S1,w2,w1) ;

endmodule
```

Note: To simulate 4X1 MUX using 2X1 MUXs the module of 2X1 MUX should be in the same project. Here the 2X1 MUX module is called three times namely Mux1, Mux2, Mux3 to implement the 4X1 MUX.