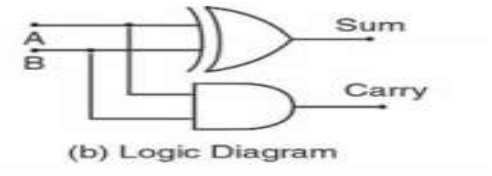
Verilog HDL Implementation of Different Combinational Circuit

1. Design of Half adder

i) Using Gate Level Modeling



(b) Logic Diagram

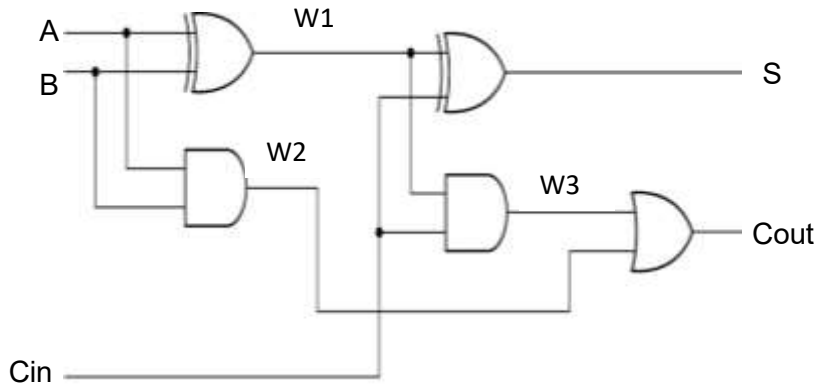
```
module HA( Sum,Carry,A,B);
output Sum,Carry;
input A,B;
xor x1(Sum,A,B);
and a1(Carry,A,B);
endmodule
```

ii) Using Data flow Modeling

```
module HA( Sum,Carry,A,B);
output Sum,Carry;
input A,B;
assign Sum=A^B;
assign Carry=A&B;
endmodule
```

2. Design of Full-Adder

i) Using Gate Level Modeling



Logic Diagram of Full Adder using Half adder

Prepared by A.Bakshi
Course: DSD

```
module FA( S,Cout,A,B,Cin);
output S,Cout;
input A,B,Cin;
wire W1,W2,W3;
xor x1(W1,A,B);
xor x2(S,W1,Cin);
and a1(W2,A,B);
and a2(W3,W1,Cin);
or o1(Cout,W2,W3);
endmodule
```

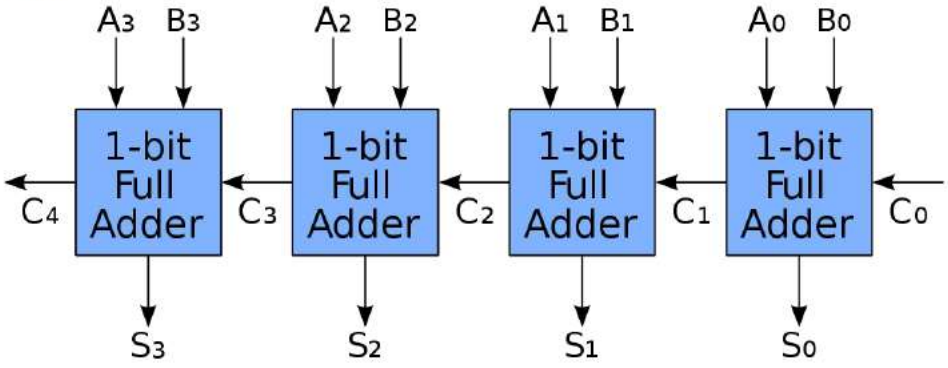
ii) Using Data flow Modeling

```
module FA( S,Cout,A,B,Cin);
output S,Cout;
input A,B,Cin;
assign S=A^B^Cin;
assign Cout=(A&B)|((A^B)&Cin);
endmodule
```

3. Design of four bit Full adder using single bit adder

RTL code for 4-bit RCA Adder.

Block Diagram:

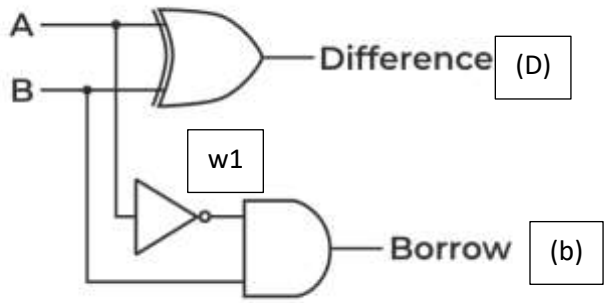



```
module FA4 (S,C4,A,B,C0);
output C4;
output [3:0] S;
input [3:0] A;
```

```
input [3:0] B;
input C0;
wire C1,C2,C3;  // internal carry
//code for 4 bit adder using 1 bit adder
FA fa0(S[0],C1,A[0],B[0],C0);
FA fa1(S[1],C2,A[1],B[1],C1);
FA fa2(S[2],S3,A[2],B[2],C2);
FA fa3(S[3],C4,A[3],B[3],C3);
endmodule
```

4. Design of Half-Subtractor

i) Using Gate Level Modeling



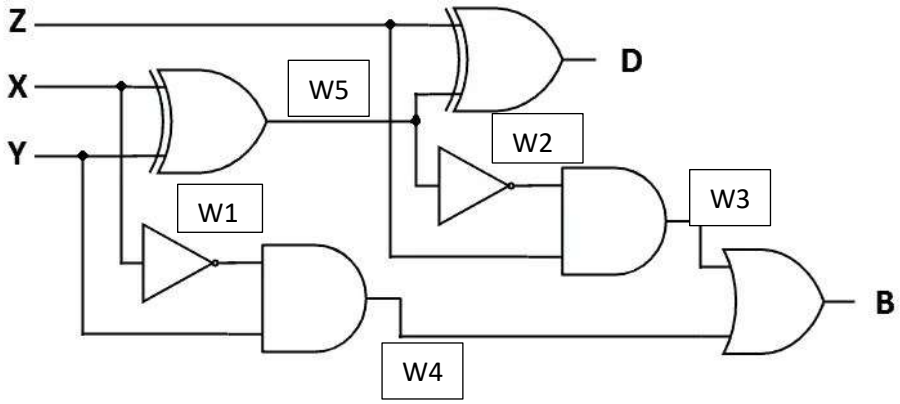

```
module HS( D,b,A,B);

output D,b;

input A,B;

wire w1;

xor x1(D,A,B);

not n1(w1,A)

and a1(b,w1,B);

endmodule
```

ii) Using Data flow Modeling

```
module HS( D,b,A,B);

output D,b;

input A,B;

assign D=A^B;

assign b=(~A)&B;

endmodule
```

5. Design of Full-Subtractor

i) Using Gate Level Modeling




```
module FS(output D, B, input X, Y, Z);
wire w1,w2,w3,w4,w5;
xor x1(w5,X,Y);
xor x2(D,w5,Z);
not n1(w1,X);
not n2(w2,w5);
and a1(w4,w1,Y);
and a2(w3,w2,Z);
or o1(B,w3,w4);
endmodule
```

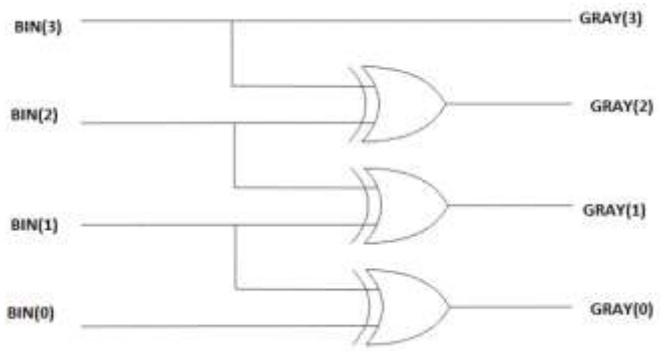
ii) Using Data flow Modeling

```
module FS(output D, B, input X, Y, Z);
assign D = X ^ Y ^ Z;
assign B = (~X &Y)| ~(X^Y) & Z;
endmodule
```

6. Design of code converters

Logic Diagram :(Binary To Gray Code Converter):

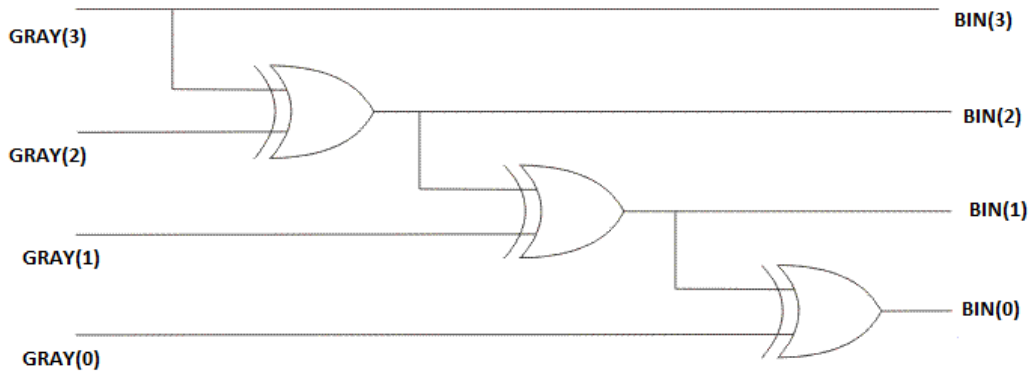
Verilog HDL Implementation of Different Combinational Circuit



i) Using Data flow Modeling

```
module bin2gray(input [3:0] bin, output [3:0] G);

assign G[3] = bin[3];
assign G[2] = bin[3] ^ bin[2];
assign G[1] = bin[2] ^ bin[1];
assign G[0] = bin[1] ^ bin[0];

endmodule
```

Logic Diagram for 4 bit Gray code to Binary converter:



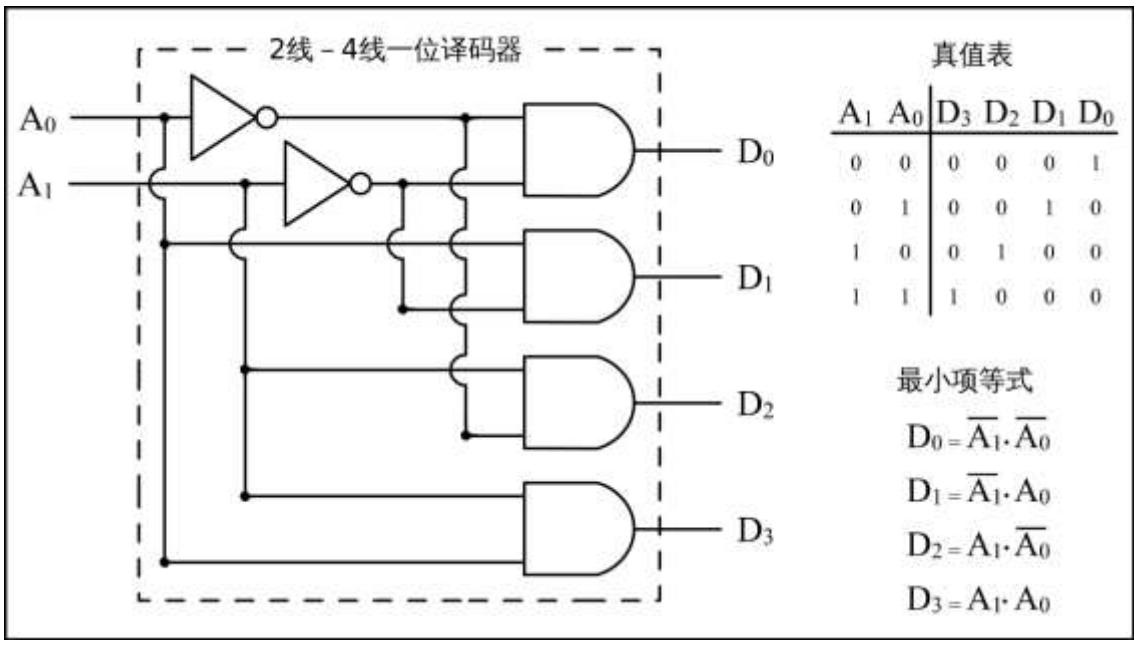
i) RTL Code for Gray code to Binary conversion:

```
module gray2bin(input [3:0] G, output [3:0]bin);

assign bin[3] = G[3];
assign bin[2] = G[3] ^ G[2];
assign bin[1] = G[3] ^ G[2] ^ G[1];
assign bin[0] = G[3] ^ G[2] ^ G[1] ^ G[0];

endmodule
```

Prepared by A.Bakshi
Course: DSD

7. Design of Decoders

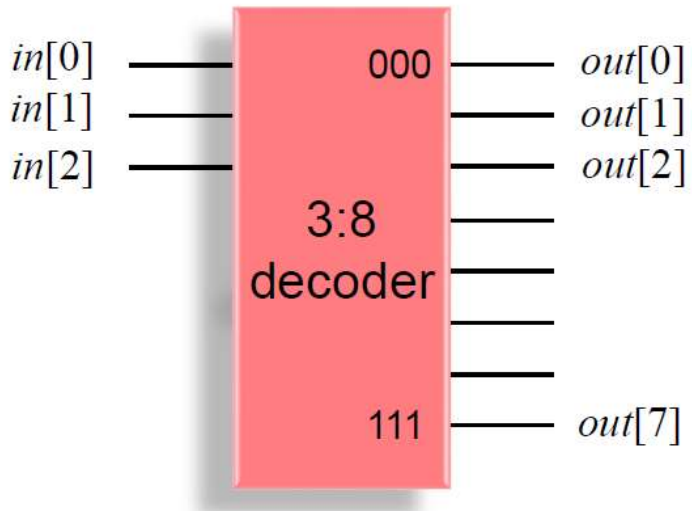
i) Verilog code for 2 to 4 line Decoder

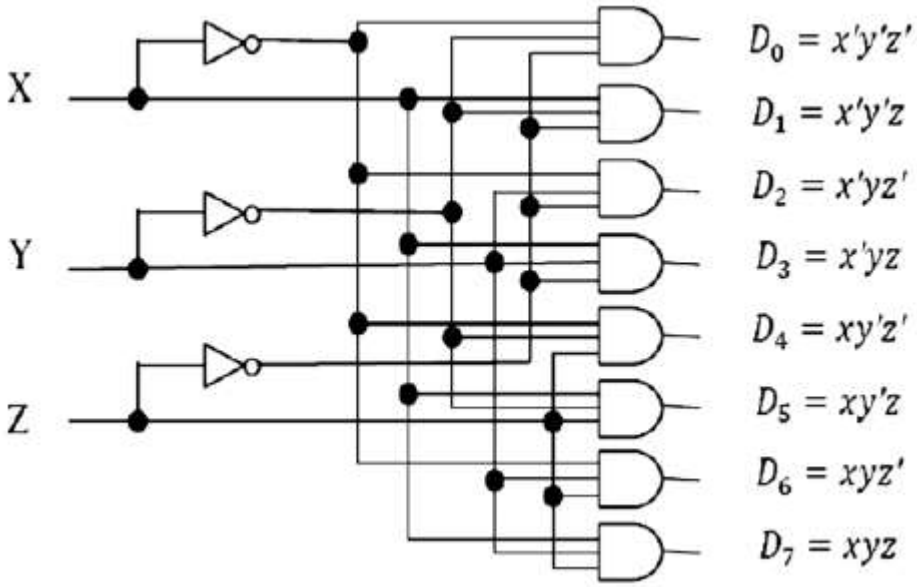



```
module decoder2_4(D0,D1,D2,D3,A0,A1);

output D0,D1,D2,D3;
input A0,A1;

assign D0 = (~A0) & (~A1);
assign D1 = (~A1) & A0;
assign D2 = A1 & (~A0);
assign D3 = A1 & A0;
endmodule
```

ii) Verilog code for 3 to 8 line Decoder

Verilog HDL Implementation of Different Combinational Circuit






```
module decoder3_8(out,in);

input [2:0]in;

output [7:0] out;

assign out[0] = ~in[2] & ~in[1] & ~in[0];

assign out[1] = ~in[2] & ~in[1] & in[0];

assign out[2] = ~in[2] & in[1] & ~in[0];

assign out[3] = ~in[2] & in[1] & in[0];

assign out[4] = in[2] & ~in[1] & ~in[0];

assign out[5] = in[2] & ~in[1] & in[0];
```

Prepared by A.Bakshi
Course: DSD

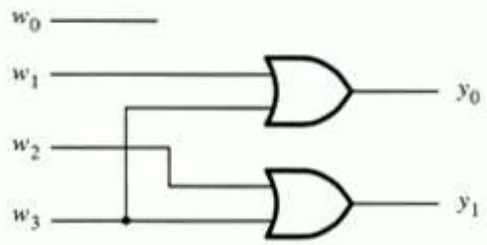
```
assign out[6] = in[2] & in[1] & ~in[0];

assign out[7] = in[2] & in[1] & in[0];

endmodule
```

9. Design of Encoders

i) 4 to 2 Encoder

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

(a) Truth table



(b) Circuit

```
module enc_2(W,Y);
input [0:3]W;
output [1:0] Y;
or o1(Y[0],W[1],W[3]);
or o2(Y[1],W[2],W[3]);
endmodule
```

II) 8 to 3 Encoder(Octal to Binary Encoder)

Octal-to-Binary take 8 inputs and provides 3 outputs, thus doing the opposite of what the 3-to-8 decoder does. At any one time, only one input line has a value of 1. The figure below shows the truth table of an Octal-to-binary encoder.

Inputs								Outputs		
D_0	**D_1**	**D_2**	**D_3**	**D_4**	**D_5**	**D_6**	**D_7**	**E_2**	**E_1**	**E_0**
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1

Inputs								Outputs	
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	1	1	1

Table: Truth Table of octal to binary encoder

For an 8-to-3 binary encoder with inputs I0-I7 the logic expressions of the outputs Y0-Y2 are:

E0 = I1 + I3 + I5 + I7

E1= I2 + I3 + I6 + I7

E2 = I4 + I5 + I6 +I7

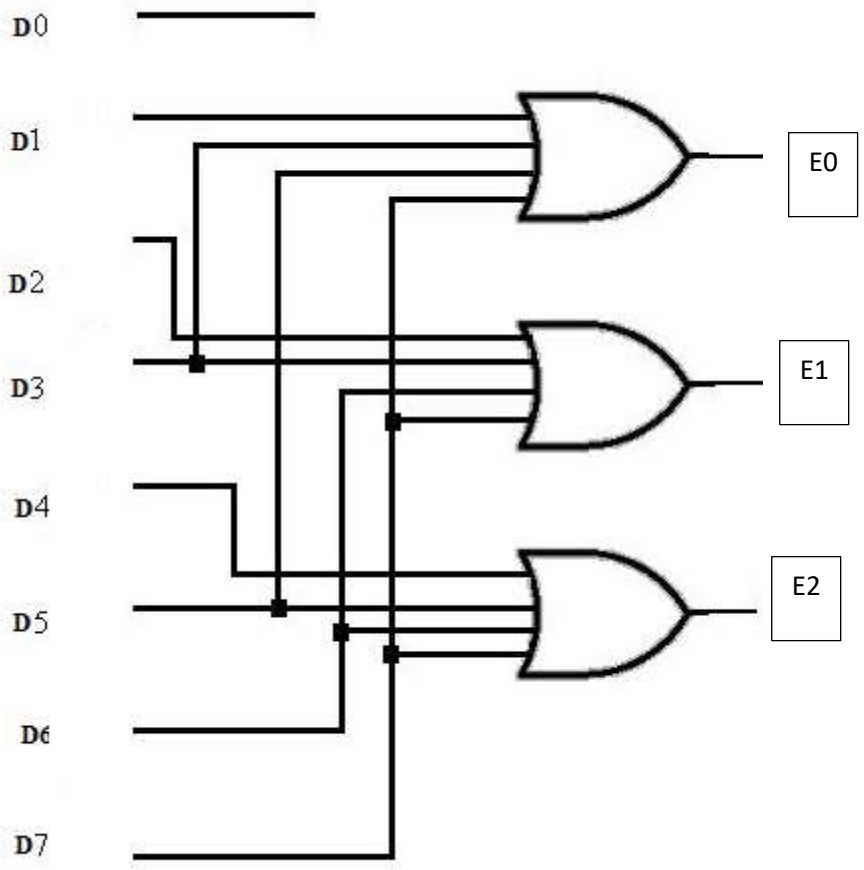


Fig. Logic Diagram of octal to binary encoder

Verilog Code:

```
module encoder8_3(D, E);
```

```
    input [7:0] D;
    output [2:0]E;
assign E[0]=D[1]|D[3]|D[5]|D[7];

assign E[1]= D[2]|D[3]|D[6]|D[7];

assign E[2]= D[4]|D[5]|D[6]|D[7];

endmodule
```