DIGITAL IC APPLICATIONS

III ECE,
DICA
4th unit

UNIT - IV

Combinational Logic Design

9621.25
33/-

Syllabus;

Binary adder-subtractor, Ripple adder, Look Ahead carry generator, ALU, Decoders, encoders, multiplexers and demultiplexers, Parity circuits, comparators, Barrel shifters, simple floating point encoder, Dual Priority encoder, Design considerations of above combinational logic circuits with relevant Digital ICs, modeling of above ICs using VHDL.

Binary adder - subtractor

A digital binary adder is a digital device that adds two binary numbers and gives its sum in binary format.

The two numbers to be added are known as "Augand" and "Addend".

The building block of digital adder is half adder. Half adders come together to form full adder.

A digital binary subtractor is a digital device that subtracts two binary numbers and gives difference in binary format.

www.Jntufastupdates.com

The two numbers that subtract are known as "minuend" and "subtrahend".

The building block of digital subtractor is Half subtractor.

Augend	6
Addend	3
Sum	9
Minuend	9
Subtrahend	3
Difference	6

Types of Binary Adder & Subtractor

1. Half Adder
2. Full adder
3. 4 bit Adder
4. Subtractor
5. Adder/Subtractor

Half adder:

Half adder can add 2 single bit numbers.

consider two numbers A, B and output being "sum" and "carry".

$Sum = \bar{A}B + A\bar{B}$

$Carry = AB$

INPUT		OUTPUT	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

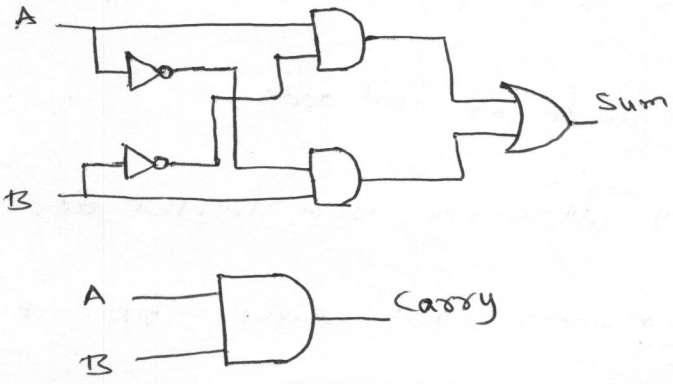


Fig: Sum & Carry using Discrete logic Gates

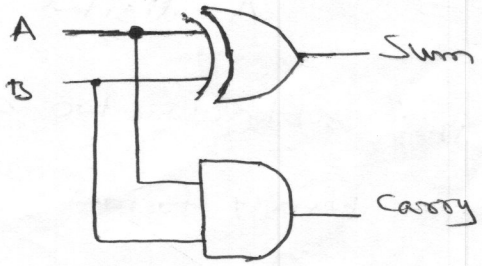


Fig: schematic design

www.Jntufastupdates.com

Behavioral Program

```
Library IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;
Entity HA is
  Port (A,B : in std_logic;
        S,C : out std_logic);
End HA;
Architecture behavioral of HA is
begin
  Process (A,B)
  begin
  if A='0' and B='1' then
      S<='0'; C<='0';
elsif A='0' and B='1' then
      S<='1'; C<='0';
else
      S<='0'; C<='0';
end if;
end process;
end behavioral;
```

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder

Inputs - A, B, C_{in}

outputs - Sum, Carry

C_{in}	A	B	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$Sum = \bar{C}_{in}\bar{A}B + \bar{C}_{in}A\bar{B} + C_{in}\bar{A}\bar{B} + C_{in}AB$$

$$= \bar{C}_{in}(\bar{A}B + A\bar{B}) + C_{in}(\bar{A}\bar{B} + AB)$$

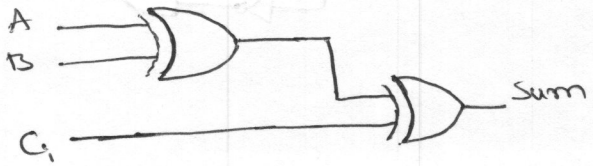
$$= \bar{C}_{in}(A \oplus B) + C_{in}(\overline{A \oplus B})$$

$$= A \oplus B \oplus C_{in}$$

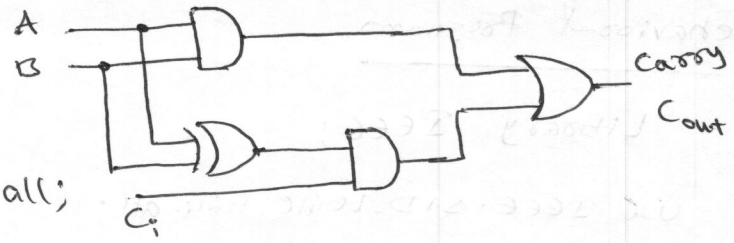
$$Cout = \bar{C}_{in}AB + C_{in}\bar{A}B + C_{in}A\bar{B} + C_{in}AB$$

$$= AB(\bar{C}_{in} + C_{in}) + C_{in}(\bar{A}B + A\bar{B})$$

$$= AB + C_{in}(A \oplus B)$$



Program




```
Library IEEE;
USE IEEE.STD_LOGIC_1164.all;
Entity FA is
  Port (A,B, Ci : in std_logic;
        S, Co : out std_logic);
End FA;
Architecture behavioral of FA is
begin
 Process (A,B, Ci)
 begin
  if (A='0' and B='0' and Ci='0') then
     S<='0'; Co<='0';
elsif (A='0' and B='1' and Ci='0') or
 (A='1' and B='0' and Ci='0') or (A='0' and B='0'
                and Ci='1') then
     S<='1'; Co<='0';
elsif (A='1' and B='1' and Ci='0') or (A='0' and
   B='1' and Ci='1') or (A='1' and B='0' and Ci='1')
                then
     S<='0'; Co<='1';
else
     S<='1'; Co<='1';
 end if;
 end process;
end behavioral;
```

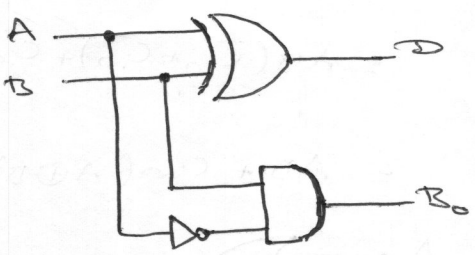
Half subtractor:

Inputs - A, B Outputs - Difference, Borrow

$D = \bar{A}B + A\bar{B}$

$B_o = \bar{A}B$

A	B	Diff (D)	Borrow (B_o)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



Program

```
Library IEEE;
USE IEEE.STD_LOGIC_1164.all;
Entity HS is
  Port (A,B : in std_logic;
        D,B0 : out std_logic);
End HS;
Architecture behavioral of HS is
begin
  Process (A,B)
  begin
    if (A='0' and B='1') then
      S<='1'; B0<='1';
    elsif (A='1' and B='0') then
      S<='1'; B0<='0';
    else
      S<='0'; B0<='0';
    end if;
  end process;
end behavioral;
```

Full subtractor:

↳ Inputs – A, B, C, outputs – Difference, Borrow

A	B	C	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$Diff = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

$= \bar{A}(\bar{B}C + B\bar{C}) + A(\bar{B}\bar{C} + BC)$

$= \bar{A}(B \oplus C) + A(\overline{B \oplus C})$

$= A \oplus B \oplus C$

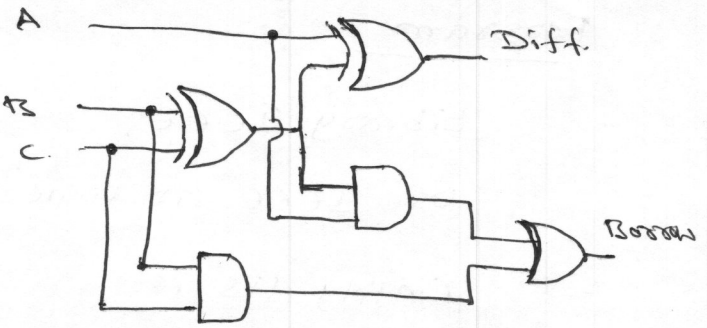
$Borrow = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + ABC$

$= \bar{A}(\bar{B}C + B\bar{C}) + BC(\bar{A} + A)$

$= \bar{A}(B \oplus C) + BC$

www.Jntufastupdates.com

Program

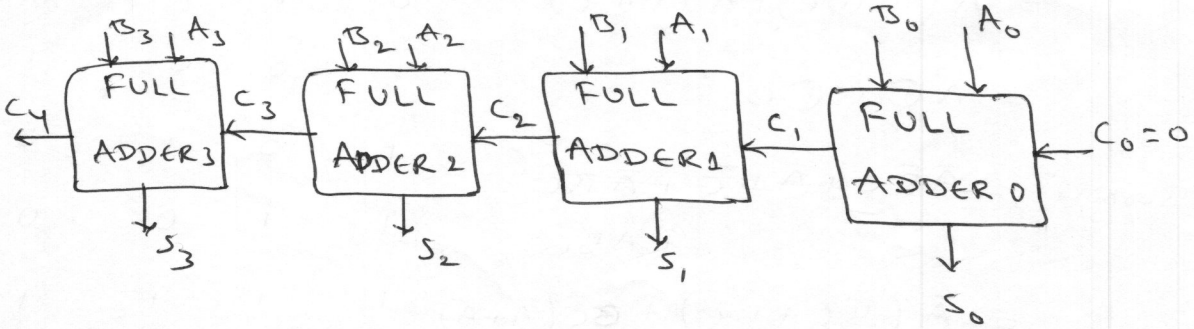



```
Library IEEE;
USE IEEE.STD_LOGIC_1164.all;
Entity FS is
   Port (A,B,C : IN STD_LOGIC;
         Diff, Borrow: OUT STD_LOGIC);
End FS;
Architecture behavioural of FS is
 begin
  Process (A,B,C)
   begin
    if ((A='0' and B='0' and c='0') or
      (A='1' and B='0' and c='1') or
      (A='1' and B='1' and c='0')) then
  Diff<='0'; Borrow<='0';
elsif (A='0' and B='1' and c='1')
   Then Diff<='0';
        Borrow<='1';
elsif (A='1' and B='0' and c='0')
   then Diff<='1'; Borrow<='0';
else
   Diff<='1'; Borrow<='1';
end if;
end Process;
end behavioural;
```

Ripple Adders (Ripple carry adder) or (Parallel adder)

It adds 2, n-bit numbers plus carry input and gives n-bit sum and carry output.

4-bit Ripple carry Adder circuit:



Here Full adder Adds ⇒ 3 (1-bit)

Half Adder Adds ⇒ 2 (1-bit

$C_3\ C_2\ C_1\ C_0 = 0$

$A = A_3\ A_2\ A_1\ A_0$

m-bit

4-bit $+ B = B_3\ B_2\ B_1\ B_0$

$C_4\ S_3\ S_2\ S_1\ S_0$

FA FA FA HA/FA

3 Full Adders
1 Half Adder } (or)

4 Full Adders

For n-bit ⇒ n Full Adders required

(or)

(n-1) Full Adders and 1 Half Adder

To set C_4, it is to wait for C_3, C_2, C_1 from Full adder 0, 1, 2, 3. So this is generating delay called Carry propagation delay.

To avoid carry propagation delay, carry look-ahead adder is implemented.

VHDL Program

```
Library IEEE;
USE IEEE STD_LOGIC_1164.ALL;

Entity RA is
   PORT (A : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
         B : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
         Cin : IN STD_LOGIC;
         S : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
         Cout : OUT STD_LOGIC);
End RA;
```

www.Jntufastupdates.com

```
Architecture ParallelAdder of RA is

    SIGNAL C : STD_LOGIC_VECTOR (3 downTO 1);

    COMPONENT FullAdder
        PORT ( F: IN STD_LOGIC_VECTOR (2 downto 0);
               Sum, carry: OUT STD_LOGIC);
    END COMPONENT;

BEGIN
    FA0 : FullAdder PORT MAP (Cin, A(0), B(0), S(0), C(1));
    FA1 : FullAdder PORT MAP (C(1), A(1), B(1), S(1), C(2));
    FA2 : FullAdder PORT MAP (C(2), A(2), B(2), S(2), C(3));
    FA3 : FullAdder PORT MAP (C(3), A(3), B(3), S(3), Cout);
END ParallelAdder;
```

COMPONENT FULLAdder

```
Library IEEE;
USE IEEE STD_LOGIC_1164.ALL;

Entity FULLAdder is
    PORT ( I: IN STD_logic;
           F: IN STD_LOGIC;
           G: IN STD_logic;
           Sum, carry: OUT STD_LOGIC );
END FULLAdder;

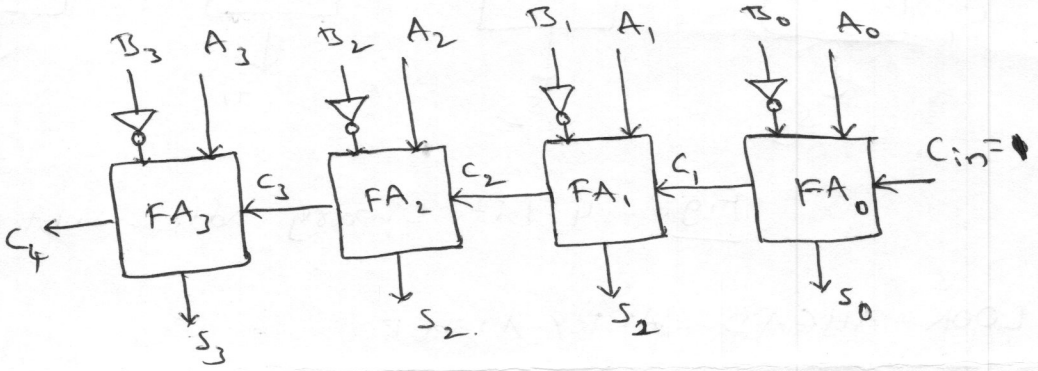
Architecture FA of FULLAdder is
begin
    Sum <= I xor F xor G;
    carry <= (I and F) or (F and G) or (G and I);
End FA;
```

4-BIT PARALLEL SUBTRACTOR

The subtraction A-B can be done by taking the 2's complement of B and adding it to A.

2's complement obtained by taking i's complement and adding 1 to the least significant pair of bits.

The is complement can be implemented with inverters.



Binary Adder - Subtractor

The addition and subtraction operations are combined into one circuit with one common binary adder. This is done by including an X-OR gate to each full adder.

The mode M controls the operation.

When M=0, the circuit is an adder, and when M=1, the circuit is an subtractor.

Each X-OR gate receives input M and one of input B. When M=0, $B \oplus 0 = B$. The full adder receives input value as B and input carry is 0 and circuit performs A+B.

www.Jntufastupdates.com

When $M=1$, $B \oplus 1 = \bar{B}$ and $C_1 = 1$. The B inputs are complemented and 1 is added through the carry input. The circuit performs operation A plus 2's complement of B.

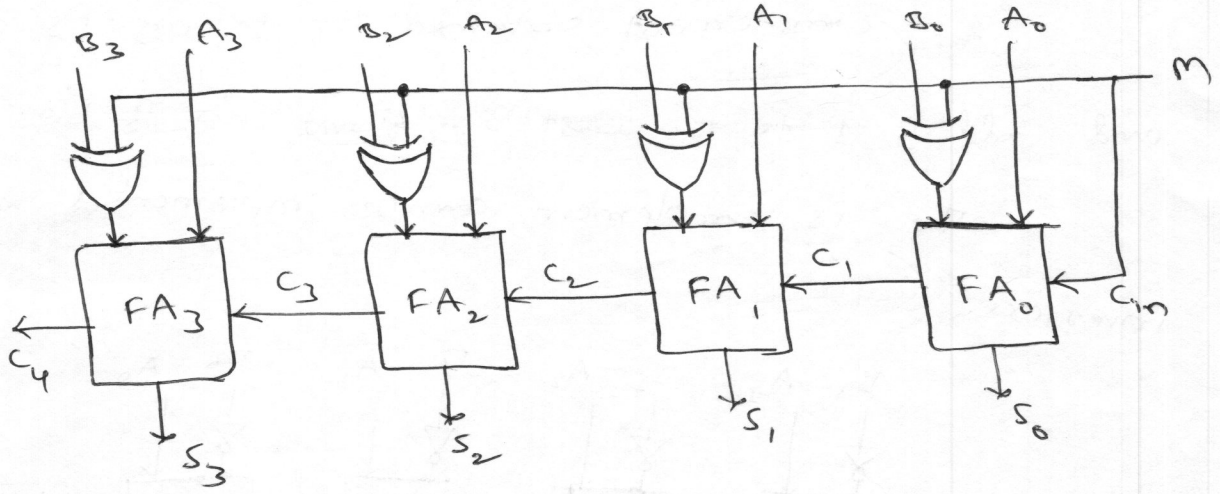


Fig: 4-bit binary adder-subtractor

LOOK-AHEAD-CARRY ADDER

In Parallel Adder, the speed with which addition can be performed is governed by the time required for the carries to propagate or ripple through all stages of adder.

The look-ahead-carry adder speeds up the process by eliminating this ripple carry delay. It examines all the input bits simultaneously and also generates the carry-in bits for all stages simultaneously.

The method of speeding up the addition process is based on two additional functions of the full-adder, called the carry generate and carry propagate functions.

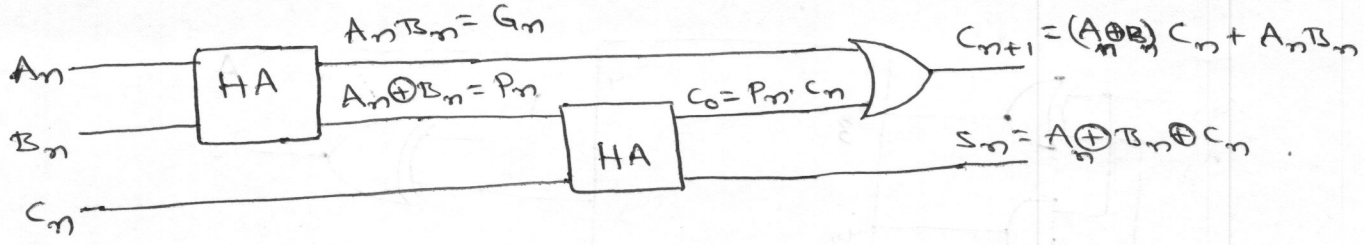
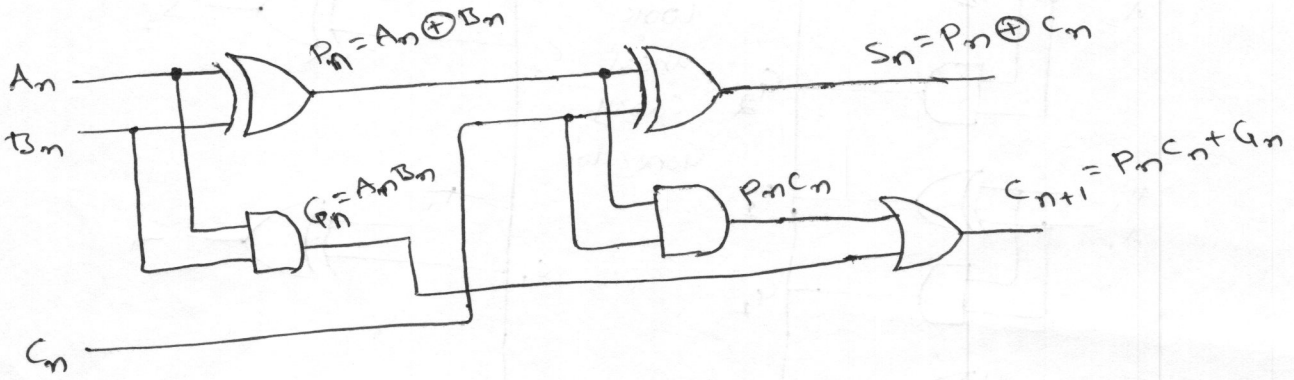


Fig: A full Adder (n^{th} stage of parallel Adder)



$$S_n = P_n \oplus C_n$$

output carry, $C_{n+1} = G_n + P_n C_n$

$$C_1 = P_0 C_0 + G_0$$

$$C_2 = P_1 C_1 + G_1 = P_1 (P_0 C_0 + G_0) + G_1$$

$$C_3 = P_2 C_2 + G_2 = P_2 \left[P_1 (P_0 C_0 + G_0) + G_1 \right] + G_2$$

$$C_4 = P_3 C_3 + G_3 = P_3 \left\{ P_2 \left[P_1 (P_0 C_0 + G_0) + G_1 \right] + G_2 \right\} + G_3$$

The general expression for n stages designated as 0 through (n-1) would be

$$C_n = G_{n-1} + P_{n-1} \cdot C_{n-1} = G_{n-1} + P_{n-1} \cdot G_{n-2} + P_{n-1} \cdot P_{n-2} \cdot G_{n-3} + \cdots + P_{n-1} \cdots P_0 C_0$$

Final output carry is expressed as a function of the input variables in SOP form, which is a two-level AND-OR or equt NAND-NAND form.

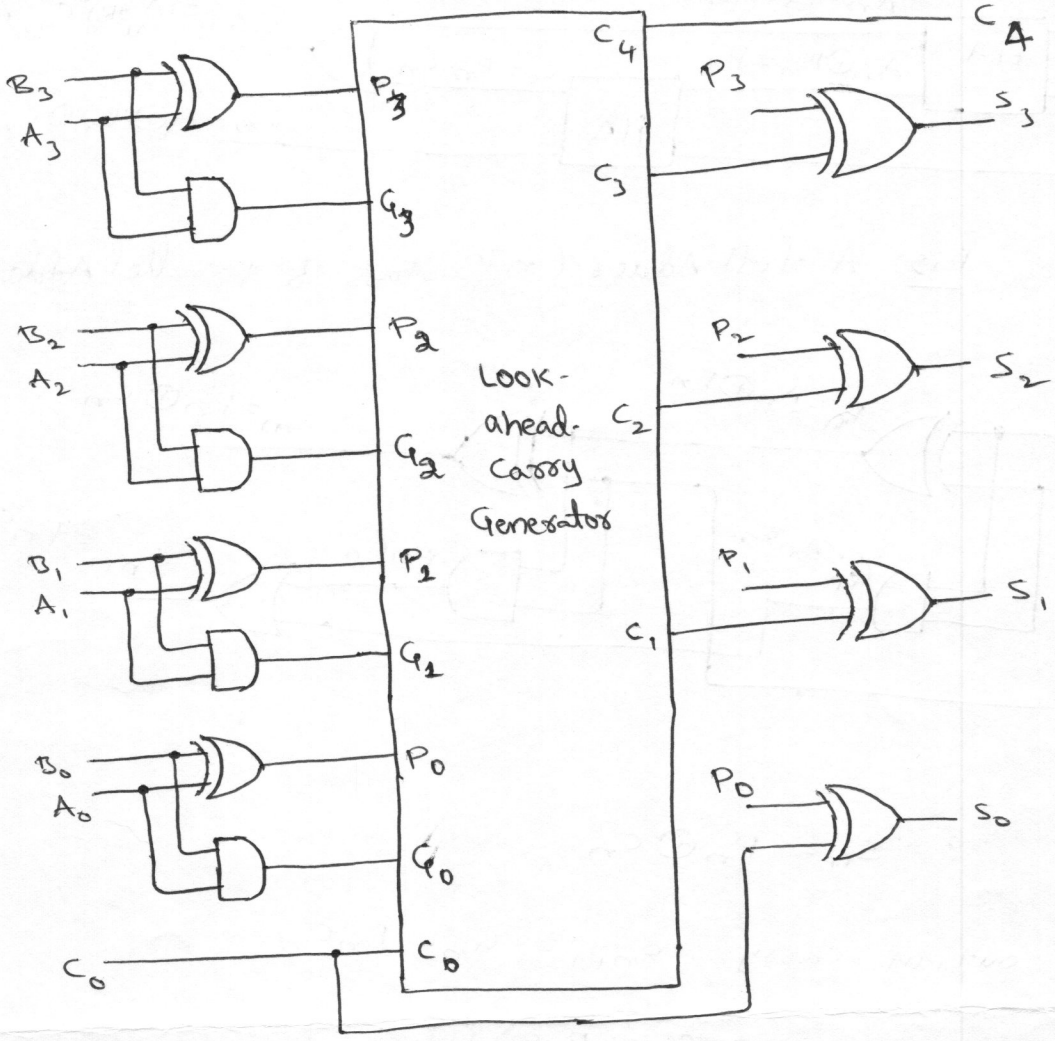


Fig: 4-bit look-ahead-carry generator

* VHDL code for look ahead carry adder

```
Library IEEE;
USE IEEE.STD_LOGIC_1164.all;
Entity LACAdd is
  Port (A,B : in std_logic_vector (3 downto 0);
        Cin : in std_logic;
        S : out std_logic_vector (3 downto 0);
        Cout : out std_logic);
End LACAdd;
```

```
Architecture Adder of LACAdd is
  signal P, G : std_logic_vector (3 downto 0) := "0000";
  signal C : std_logic_vector (4 downto 0) := "00000";
  Component LAC
    Port ( CP : in std_logic_vector (3 downto 0);
           CG : in std_logic_vector (3 downto 0);
           CC : out std_logic_vector (4 downto 0);
           CCin : in std_logic);
  End component;
  begin
    P <= A xor B;
    G <= A and B;
    gen_C : LAC Port map (P, G, Cin, C);
    S <= P xor C(3 downto 0);
    Cout <= C(4);
  End Adder
```

LAC.vhd

```
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
  Entity LAC is
    Port ( CP : in std_logic_vector (3 downto 0);
           CG : in std_logic_vector (3 downto 0);
           CC : out std_logic_vector (4 downto 0);
           CCin : in std_logic);
  End LAC;
```

Architecture Add of LAC is

begin

CC(0) <= CCin ;

CC(1) <= CG(0) or (CP(0) and CCin);

CC(2) <= CG(1) or (CP(1) and CG(0)) or (CP(1) and CP(0) and CCin);

CC(3) <= CG(2) or (CP(2) and CG(1)) or (CP(2) and CP(1) and CG(0)) or (CP(2) and CP(1) and CP(0) and CCin);

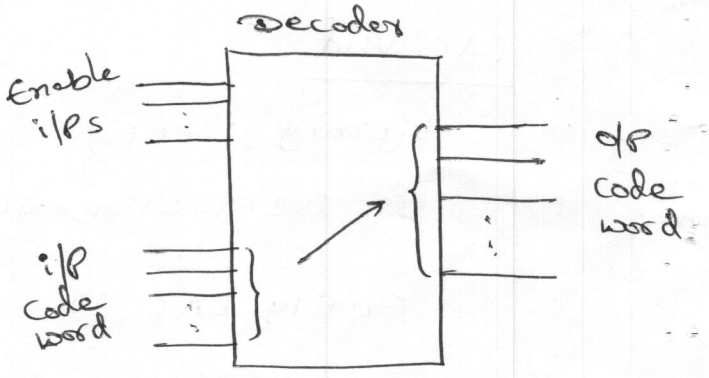
CC(4) <= CG(3) or (CP(3) and CG(2)) or (CP(3) and CP(2) and CG(1)) or (CP(3) and CP(2) and CP(1) and CG(0)) or (CP(3) and CP(2) and CP(1) and CP(0) and CCin);

End Add;

Decoders:

Decoder is a multiple i/p, multiple o/p logic circuit that converts coded inputs into coded outputs, where i/p & o/p codes are different.

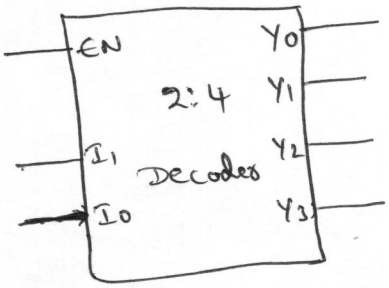
The enable must be asserted for the decoder to perform its normal function, otherwise decoder maps 'all' i/p code words into a single disabled o/p code word.



1. 74X139 (Dual 2 to 4 Decoder)

2. 74x138 (3 to 8 Decoder)

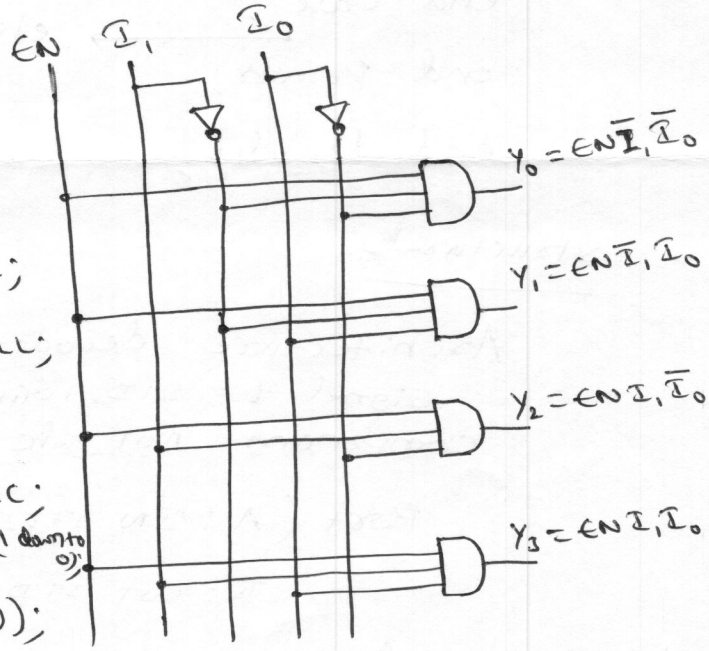
2 to 4 Binary decoder



EN	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

VHDL Code

~~Data flow~~




```
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
Entity dec2to4 is
Port ( EN : IN STD_LOGIC;
       I : IN STD_LOGIC_VECTOR(1 downto 0);
       Y : OUT STD_LOGIC_VECTOR
           (3 downto 0));
End dec2to4;
```

Data flow:

```
Architecture decoder of dec2to4 is
begin
    Y(0) <= EN and (NOT I(1)) and (NOT I(0));
    Y(1) <= EN and (NOT I(1)) and (I(0));
    Y(2) <= EN and (I(1)) and (NOT I(0));
    Y(3) <= EN and I(1) and I(0);
End decoder;
```

www.Jntufastupdates.com

Behavioral:

```
Architecture decoder of dec2to4 is
begin
Process (EN, I)
begin ⟶ if (EN='1') then
  case I is
    when "00" => Y <= "0001";
    when "01" => Y <= "0010";
    when "10" => Y <= "0100";
    when others => Y <= "1000";
  end case; ⟶ else Y <= "0000"; end if;
end process;
end decoder;
```

Structural:

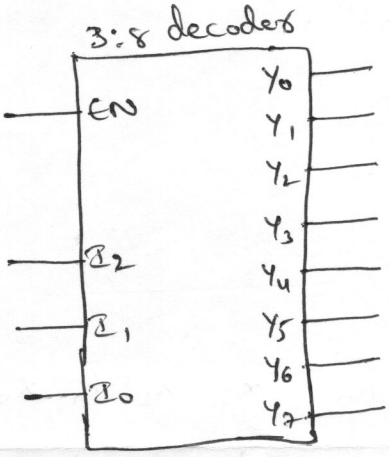
```
Architecture decoder of dec2to4 is
  signal t : STD_LOGIC_VECTOR(1 downto 0);
component NOT_gate
  Port (A : IN STD_LOGIC;
        B : OUT STD_LOGIC);   // B <= NOT A;
  End component;
component AND_gate
  Port (M,N,O : IN STD_LOGIC;
        P : OUT STD_LOGIC);   // P <= M and N and O;
  End component;
begin
U1 : NOT_gate Port map (I(0), t(0));
U2 : NOT_gate Port map (I(1), t(1));
```

www.Jntufastupdates.com

```
U3 : AND_gate Port map (EN, t(1), t(0), Y(0));
U4 : AND_gate Port map (EN, t(1), I(0), Y(1));
U5 : AND_gate Port map (EN, I(1), t(0), Y(2));
U6 : AND_gate Port map (EN, I(1), I(0), Y(3));

End decoder;
```

3 to 8 decoder

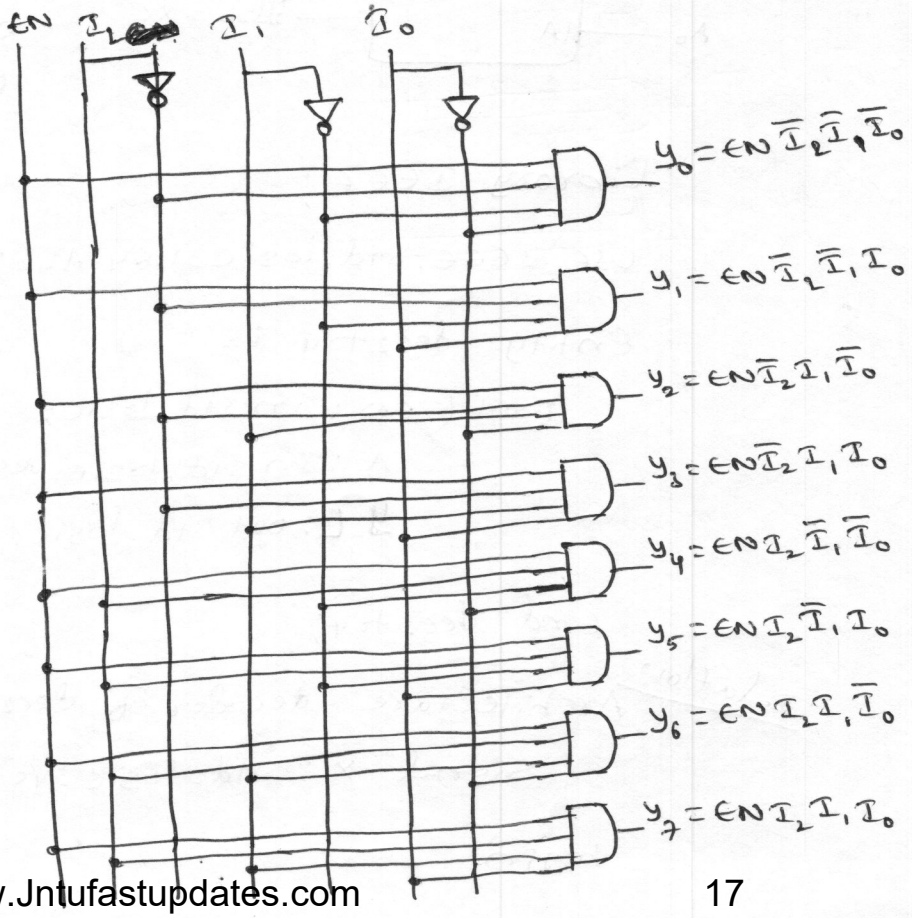


EN	I_2	I_1	I_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

```
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

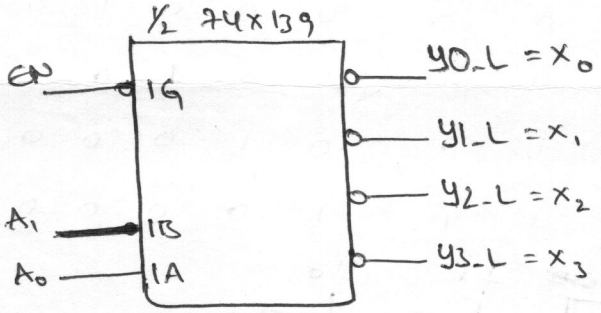
Entity dec3to8 is
Port ( EN: IN STD_LOGIC;
   I: IN STD_LOGIC_VECTOR
        (2 downto 0);
   Y: OUT STD_LOGIC_VECTOR
        (7 downto 0));
End dec3to8;

Architecture decoder of
    dec3to8 is
begin
  Signal ys : STD_LOGIC_
    VECTOR(7 downto 0);
```



```
begin
  with I select
    Ys <= "00000001" when "000",
          "00000010" when "001",
          "00000100" when "010",
          "10000000" when others;
  Y <= Ys when EN='1' else "00000000";
end decoder;
```

74 X 139 dual 2 to 4 decoder:



EN	A1	A0	X3	X2	X1	X0
1	X	X	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

```
Library IEEE;
USE IEEE.std_logic_1164.ALL;
Entity dec2to4 is
   Port( EN : In std_logic;
         A : In std_logic_vector(1 downto 0);
         Y_L: out std_logic_vector (3 downto 0));
End dec2to4;
```

Dataflow

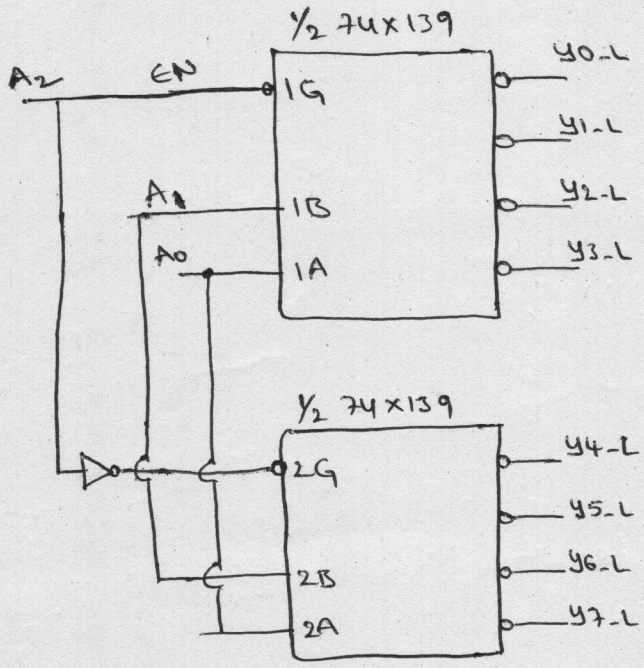
```
Architecture decoder of dec2to4 is
   Signal X : std_logic_vector (3 downto 0);
begin
```

```
with A select
    X<= "1110" when "00",
        "1101" when "01",
        "1011" when "10",
        "0111" when others;
Y_L <= X when EN = '0' else "1111";
End decoder;
```

Behavioural model:

```
Architecture decoder of dec2to4 is
begin
Process (EN, A)
begin
  case A is
when "00" => X<= "1110";
when "01" => X<="1101";
when "10" => X<="1011";
when others => X<="0111";
 end case;
if EN='0' then Y_L <= X;
 else Y_L <= "1111";
end if;
end Process;
end decoder;
```

3 to 8 decoder using 74x139 dual 2 to 4 decoder



A_2	A_1	A_0	$Y7_L$	$Y6_L$	$Y5_L$	$Y4_L$	$Y3_L$	$Y2_L$	$Y1_L$	$Y0_L$
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1
x	x	x	1	1	1	1	1	1	1	1

VHDL Program

```
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
Entity dec3to8 is
    Port ( A : In std_logic_vector(2 downto 0);
           Y_L : out std_logic_vector (7 downto 0));
End dec3to8;
```

Dataflow

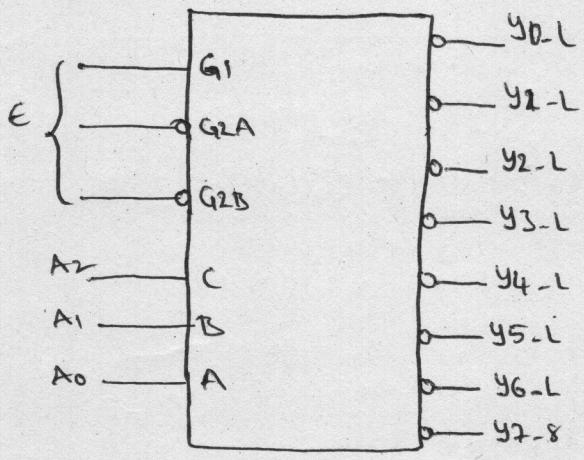
```
Architecture decoder of dec3to8 is
begin
  With A select
    Y_L <= "11111110" when "000",
           "11111101" when "001",
           "11111011" when "010",
           "11110111" when "011",
```

```
"11101111" when "100",
"11011111" when "101",
"10111111" when "110",
"01111111" when others;
End decoder;
```

Behavioural

```
Architecture decoder of decstor is
begin
Process (A)
begin
 case A is
  when "000" => y_L <= "10111110";
  when "001" => y_L <= "11111101";
  when "010" => y_L <= "11111011";
  when "011" => y_L <= "11110111";
  when "100" => y_L <= "11101111";
  when "101" => y_L <= "11011111";
  when "110" => y_L <= "10111111";
  when others => y_L <= "01111111";
End case;
End Process;
End decoder;
```

3 to 8 Decoder (74X138)



E	A	Y_L
100	000	11111110
100	001	11111101
100	010	11111011
100	011	11110111
100	100	11101111
100	101	11011111
100	110	10111111
100	111	01111111
000	XXX	11111111

VHDL Program

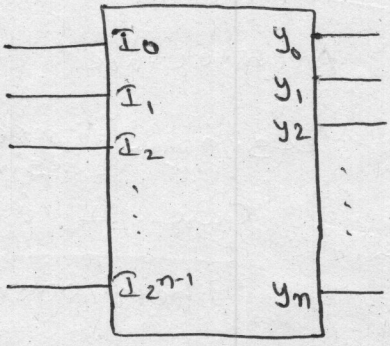
```
Library IEEE;
USE IEEE.std_logic_1164.ALL
Entity dec3to8 is
   Port (G1, G2A, G2B : In std_logic;
         A : In std_logic_vector (2 downto 0);
         Y_L : out std_logic_vector (7 downto 0));
End dec3to8;
Architecture decoder of dec3to8 is
begin
   with A select
      Y_L <= "11111110" when "000",
             "11111101" when "001",
             "11111011" when "010",
             "11110111" when "011",
             "11101111" when "100",
             "11011111" when "101",
             "10111111" when "110",
             "01111111" when others;
End decoder;
```

Encoders (74x148)

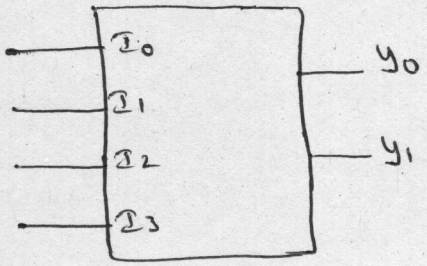
Encoder is a multiple inputs and multiple outputs.

device. i.e 2^n to n.

Binary encoder:



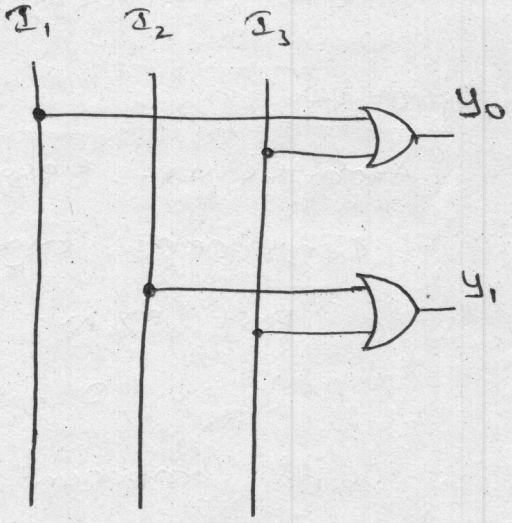
4 to 2 encoder



I3	I2	I1	I0	y1	y0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$y_1 = I_2 + I_3$

$y_0 = I_1 + I_3$




```
Library IEEE;
use IEEE.std_logic_1164.all;
Entity enc4to2 is
    Port (I: in std_logic_vector
            (3 downto 0);
          y: out std_logic_vector
            (1 downto 0));
End enc4to2;
```

Data flow

```
Architecture encoder of enc4to2 is
begin
    with I select
        y<="00" when "0001",
           "01" when "0010",
```

```
y(0)<= I(1) or I(3);
y(1)<= I(2) or I(3);
```

"10" when "0100",

"11" when others;

End encoder;

Behavioural:

Architecture Encoder of enc4to2 is

begin → {Process (I)
begin

Case I is

when "0001" => y<="00";

When "0010" => y<="01";

When "0100" => y<="10";

When others => y<="11";

End case;

End Process;

End encoder;

Structural:

Architecture Encoder of enc4to2 is

component orgate1

Port (m, n : In std_logic;

o : out std_logic);

End component;

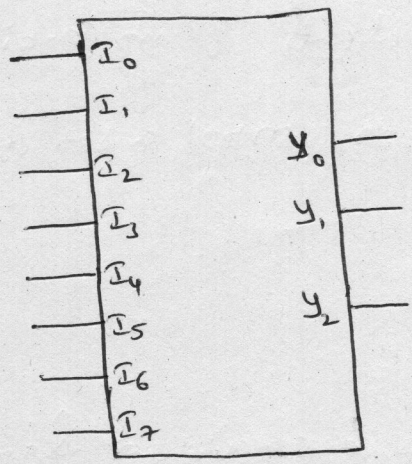
begin

U1 : orgate1 Port map (I(1), I(3), y(0));

U2 : orgate1 Port map (I(2), I(3), y(1));

End Encoder;

8 to 3 Encoder



I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	y_2	y_1	y_0
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

$$y_0 = I_1 + I_3 + I_5 + I_7$$

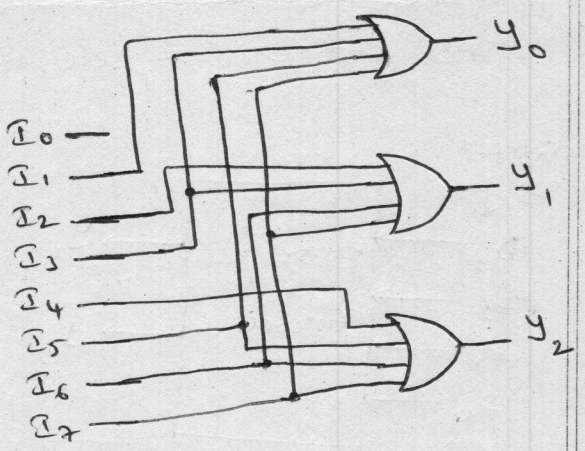
$$y_1 = I_2 + I_3 + I_6 + I_7$$

$$y_2 = I_4 + I_5 + I_6 + I_7$$

```
Library IEEE;
USE IEEE.std_logic_1164.all;

Entity Enc8to3 is
  Port (I: in std_logic_vector
        (7 downto 0);
        y: out std_logic_vector
        (2 downto 0));
End Enc8to3;

Architecture Encoder of
        Enc8to3 is
begin
  y(0) <= I(1) or I(3) or I(5) or I(7);
  y(1) <= I(2) or I(3) or I(6) or I(7);
  y(2) <= I(4) or I(5) or I(6) or I(7);
End Encoder;
```



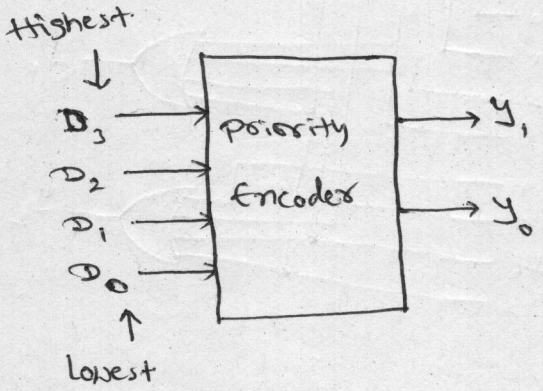
Priority Encoders

The 1 out of 2^n coded outputs of an n-bit binary decoder are generally used to control a set of 2^n devices.

4 to 2 Priority Encoders

The priority encoder is a circuit or algorithm that compresses multiple binary inputs into a smaller number of outputs.

Priority encoders can select the inputs with highest priority in many practical applications. This process of selection is called arbitration.

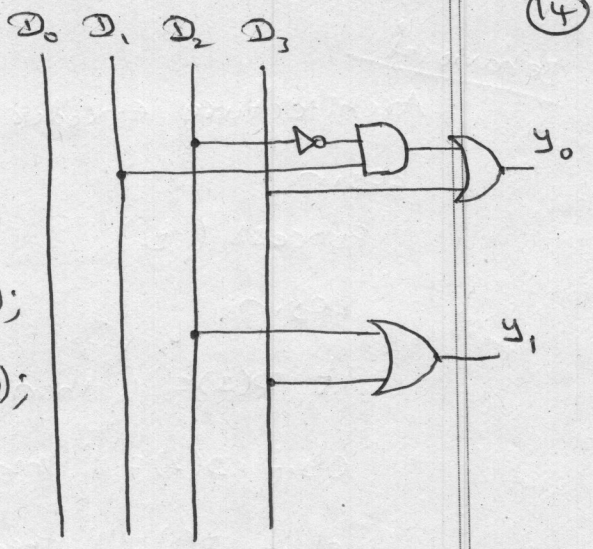


D_3	D_2	D_1	D_0	y_1	y_0
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

$$y_0 = \bar{D}_2 D_1 + D_3$$

$$y_1 = D_2 + D_3$$

www.Jntufastupdates.com




```
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

Entity PE4to2 is
   Port(D: in std_logic_vector(3 downto 0);
        y: out std_logic_vector(1 downto 0));
End PE4to2;
```

Dataflow

```
Architecture Encoder of PE4to2 is
begin
   y(0) <= (NOT D(2) and D(1)) or D(3);
   y(1) <= D(2) or D(3);
End Encoder;
```

Structural

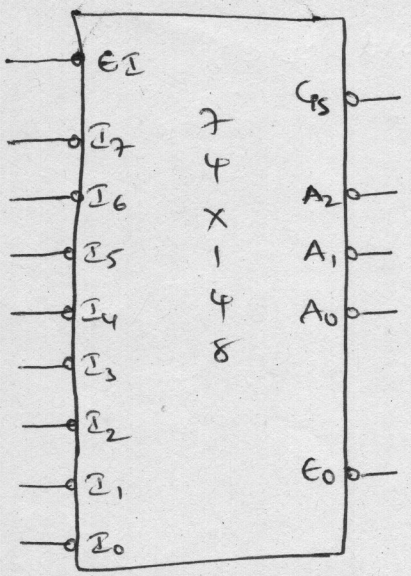
```
Architecture Encoder of PE4to2 is
  component NOTgate
     Port(Ni: in std_logic;
          No: out std_logic);
  End component;
  component ANDgate
     Port(A0,A1: in std_logic;
          A2: out std_logic);
  End component;
  component ORgate
     Port(R0,R1: in std_logic;
          R2: out std_logic);
  End component;
begin
  U1: NOTgate Port map (D(2), P);
  U2: ANDgate Port map (P, D(1), Q);
  U3: ORgate Port map (Q, D(3), y(0));
  U4: ORgate Port map (D(2), D(3), y(1));
End Encoder;
```

Signal P,Q: std_logic;

Behavioral

```
Architecture Encoder of PE4to2 is
begin
  Process (D)
  begin
  if D(3)='1' then y <= "11";
  elsif D(2)='1' then y<="10";
  elsif D(1)='1' then y<="01";
  else y<="00";
  End if;
 End Process;
End Encoder;
```

8 to 3 Priority Encoder (74×148):

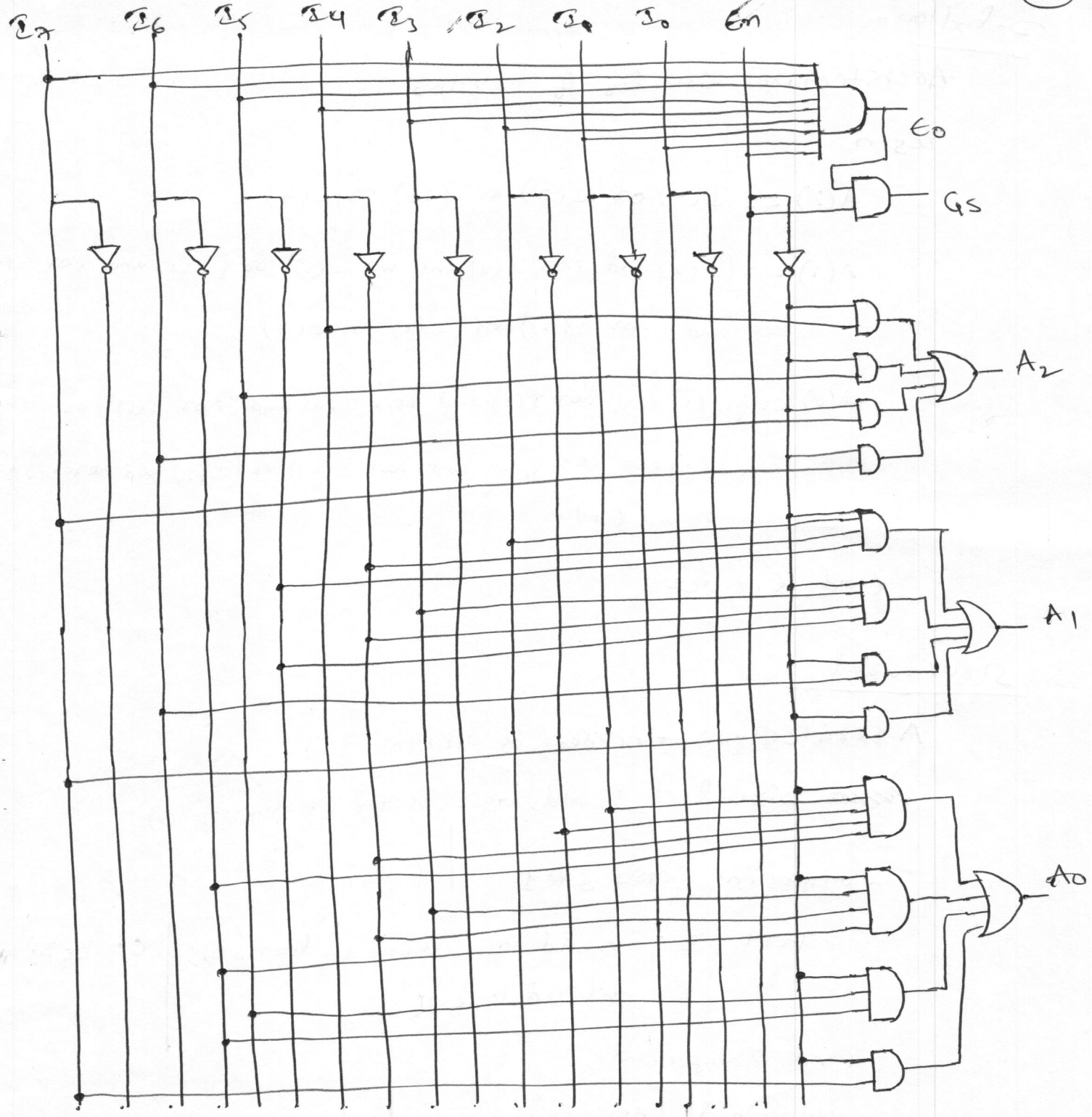


$\overline{EI}$	$\overline{I_7}$	$\overline{I_6}$	$\overline{I_5}$	$\overline{I_4}$	$\overline{I_3}$	$\overline{I_2}$	$\overline{I_1}$	$\overline{I_0}$	A_2	A_1	A_0	GS	EO
0	x	x	x	x	x	x	x	x	1	1	1	1	0
1	0	0	0	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	0	0	1	x	0	0	1	0	1
1	0	0	0	0	0	1	x	x	0	1	0	0	1
1	0	0	0	0	1	x	x	x	0	1	1	0	1
1	0	0	0	1	x	x	x	x	1	0	0	0	1
1	0	0	1	x	x	x	x	x	1	0	1	0	1
1	0	1	x	x	x	x	x	x	1	1	0	0	1
1	1	x	x	x	x	x	x	x	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

$$A0 = \overline{I_1}\,\overline{I_2}\,\overline{I_4}\,\overline{I_6} + \overline{I_3}\,\overline{I_4}\,\overline{I_6} + \overline{I_5}\,\overline{I_6} + \overline{I_7}$$

$$A1 = \overline{I_2}\,\overline{I_4}\,\overline{I_5} + \overline{I_3}\,\overline{I_4}\,\overline{I_5} + \overline{I_6} + \overline{I_7}$$

$$A2 = \overline{I_4} + \overline{I_5} + \overline{I_6} + \overline{I_7}$$




```
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
Entity PE8to3 is
    Port ( En : in std_logic;
           I : in std_logic_vector (7 downto 0);
           Eo : out std_logic;
           A : out std_logic_vector (2 downto 0));
End PE8to3;
```

Dataflow:

```
Architecture Encoder of PE8to3 is
begin
    A(2) <= I(4) or I(5) or I(6) or I(7)
    A(1) <= (I(2) and NOT I(4) and NOT I(5)) or (I(3) and NOT I(4)
            and NOT I(5)) or I(6) or I(7)
    A(0) <= (I(1) and NOT I(2) and NOT I(4) and NOT I(6)) or
            (I(3) and NOT I(4) and NOT I(6)) or (I(5) and NOT I(6))
            or I(7)
End Encoder;
```

Structural

```
Architecture Encoder of PE8to3 is
    Signal t : std_logic_vector (8 downto 0);
    Component AND_gate1
        Port (B : In std_logic_vector (3 downto 0);
              C : out std_logic);
    End component;
    Component AND_gate2
        Port (D : In std_logic_vector (2 downto 0);
              E : out std_logic);
    End component;
    Component AND_gate3
        Port ( F : In std_logic_vector (1 downto 0);
               G : out std_logic);
    End component;
```

$C = B_3 B_2 B_1 B_0$

$E = D_2 * D_1 * D_0$

$G = F_1 * F_0$

component OR_gate

Port (P : In std_logic_vector (3 downto 0);
Q : out std_logic);

$Q = P_3 + P_2 + P_1 + P_0$

End component;

~~begin~~

~~U1 : AND_gate Port map ();~~

Component NOT_gate

Port (R : In std_logic ;
S : out std_logic);

End component;

begin

U1 : NOT_gate Port map (I(2), t(2));

U2 : NOT_gate Port map (I(4), t(4));

U3 : NOT_gate Port map (I(6), t(6));

U4 : NOT_gate Port map (I(5), t(5));

U5 : AND_gate Port map (I(1), t(2), t(4), t(6), t(0));

U6 : AND_gate Port map (I(3), t(4), t(6), t(1));

U7 : AND_gate Port map (I(5), t(6), t(3));

U8 : AND_gate Port map (I(2), t(4), t(5), t(7));

U9 : AND_gate Port map (I(3), t(4), t(5), t(8));

U10 : OR_gate Port map (t(0), t(1), t(3), I(7), A(0));

U11 : OR_gate Port map (t(7), t(8), I(6), I(7), A(1));

U12 : OR_gate Port map (I(4), I(5), I(6), I(7), A(2));

End Encoder;

Behavioral

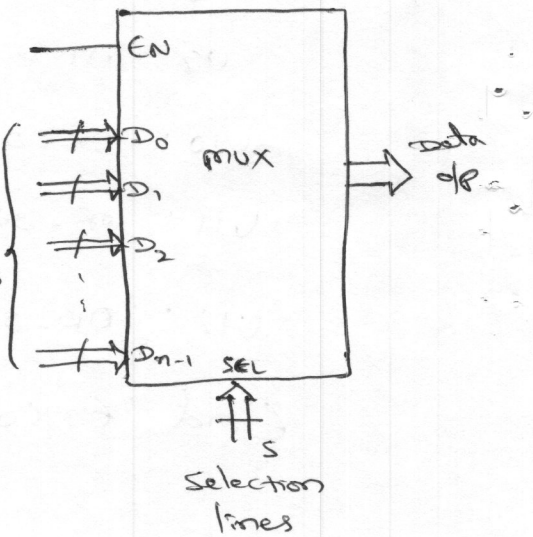
```
Architectural Encoder of PE8to3 is
begin
  Process (EI, I)
  begin
    If I(0) ='1' then A<="000"; GS<='0'; EO<='1';
  elsif I(1) ='1' then A<="001"; GS<='0'; EO<='1';
  elsif I(2) ='1' then A<="010"; GS<='0'; EO<='1';
  elsif I(3) ='1' then A<="011"; GS<='0'; EO<='1';
  elsif I(4) ='1' then A<="100"; GS<='0'; EO<='1';
  elsif I(5) ='1' then A<="101"; GS<='0'; EO<='1';
  elsif I(6) ='1' then A<="110"; GS<='0'; EO<='1';
  else A<="111"; GS<='0'; EO<='1';
  End if;
  End process;
  End Encoder;
```

Multiplexers:

Multiplexer is a digital switch, it converts data sources 1 of n to its o/p.

There are n sources of data, with 'b' bits each and S is the selection lines to select among n sources. so

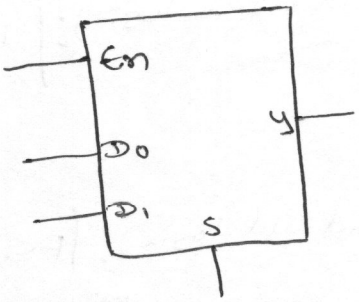
$$S = \log_2 N$$



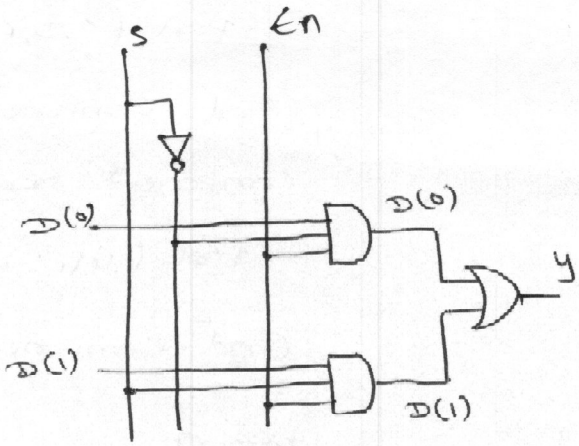
Types of multiplexers

1. 2:1 MUX 2. 4:1 MUX 3. 8:1 MUX

2:1 MUX



En	S	y
0	X	0
1	0	D(0)
1	1	D(1)



$y = En \cdot \bar{S} \cdot D(0) + En \cdot S \cdot D(1)$

VHDL Program

```
Library IEEE;
USE IEEE.std_logic_1164.All;
Entity mux2to1 is
  Port(S, En: in std_logic;
       D : in std_logic_vector (1 downto 0);
       y : out std_logic);
End mux2to1;
```

Data flow:

```
Architecture mux of mux2to1 is
  signal x : std_logic;
begin
  with S select
    x<= D(0) when '0',
        D(1) when '1';
  y<= x when En='1'
    else y<='0';
```

```
begin
  y <= En and NOT S and D(0)
       or En and S and D(1);
End mux;
```

structural

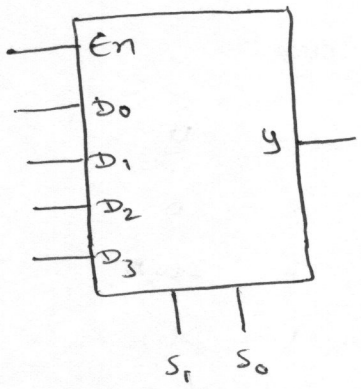
```
Architecture mux of mux2to1 is
  signal t: std_logic_vector (2 downto 0);
 component NOTgate
   Port (A: in std_logic ; B: out std_logic);  | B<= NOT A;
 End component;
 component ANDgate
   Port (C,D,E: in std_logic; F: out std_logic);  | F<= C and D and E;
 End component;
 component orgate
   Port (G,H: in std_logic ; I: out std_logic);  | I<= G or H;
 End component;
 begin
 U1: NOTgate Port map (s, t(0));
 U2: ANDgate Port map (En, t(0), D(0), t(1));
 U3: ANDgate Port map (En, S, D(1), t(2));
 U4: orgate Port map (t(1), t(2), y);
End mux;
```

Behavioural

```
Architectural mux of mux2to1 is
  signal x: std_logic;
 begin
  Process (En, s)
   begin
     case S is
       when '0' => x<= D(0);
       when '1' => x<= D(1);
```

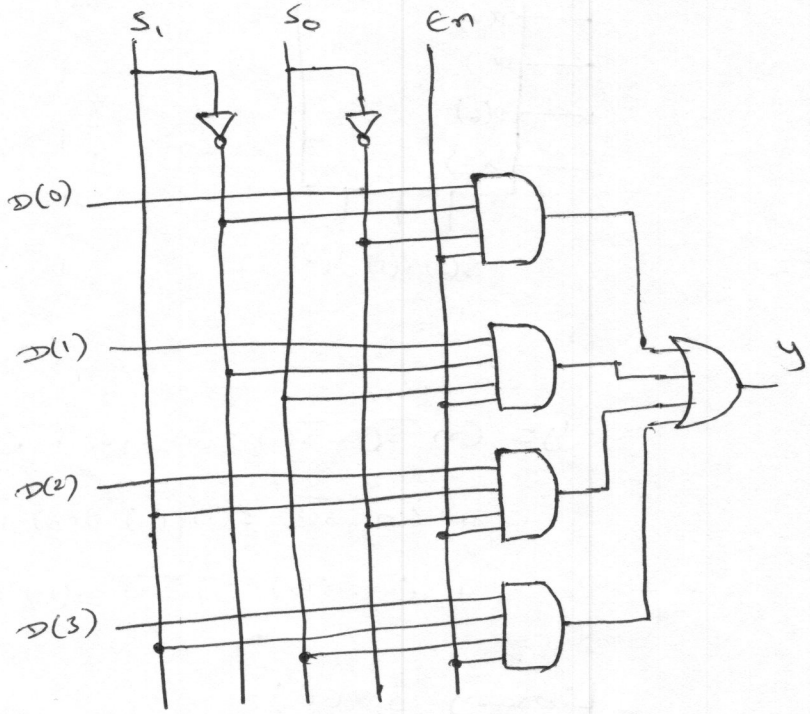
```
if En='1' then y<=x;
  else y<='0';
End if;
End case;
End process;
End mux;
```

4:1 MUX



En	S_1	S_0	y
0	x	x	0
1	0	0	D(0)
1	0	1	D(1)
1	1	0	D(2)
1	1	1	D(3)

$y = En\ \overline{S(1)}\ \overline{S(0)}\ D(0) + En\ \overline{S(1)}\ S(0)\ D(1) + En\ S(1)\ \overline{S(0)}\ D(2) + En\ S(1)\ S(0)\ D(3)$



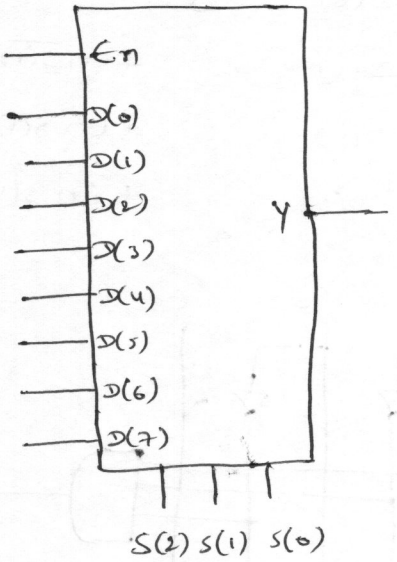

```
Library IEEE;
USE IEEE.std_logic_1164.ALL;
Entity MUX4to1 is
  Port (En: In std_logic;
        S: In std_logic_vector
           (1 downto 0);
        D: In std_logic_vector
           (3 downto 0);
        y: out std_logic);
End MUX4to1;
Architecture mux of MUX4to1 is
  Signal x : std_logic;
begin
```

```
with s select
    x <= D(0) when "00",
         D(1) when "01",
         D(2) when "10",
         D(3) when "11";

    y <= x when En='1' else y<='0';

end mux;
```

8:1 MUX



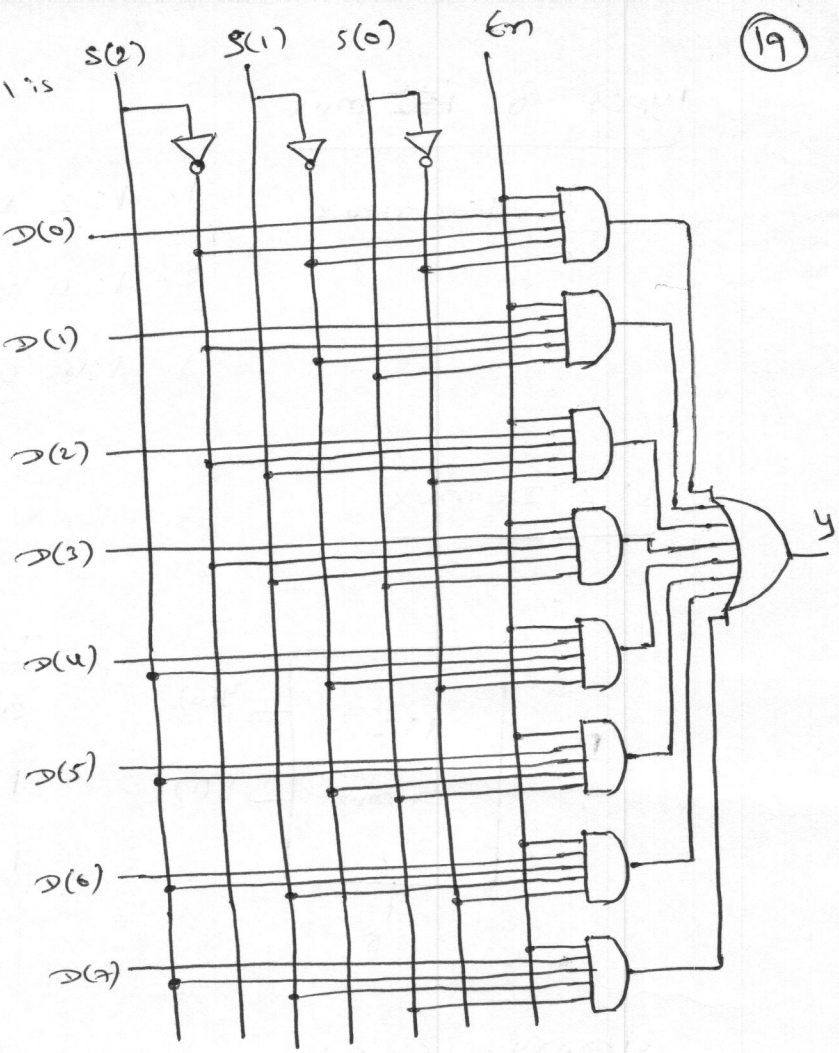
En	S(2)	S(1)	S(0)	y
0	x	x	x	0
1	0	0	0	D(0)
1	0	0	1	D(1)
1	0	1	0	D(2)
1	0	1	1	D(3)
1	1	0	0	D(4)
1	1	0	1	D(5)
1	1	1	0	D(6)
1	1	1	1	D(7)

$$y = En\,\overline{S(2)}\,\overline{S(1)}\,\overline{S(0)}\,D(0) + En\,\overline{S(2)}\,\overline{S(1)}\,S(0)\,D(1) + En\,\overline{S(2)}\,S(1)\,\overline{S(0)}\,D(2)$$
$$+ En\,\overline{S(2)}\,S(1)S(0)\,D(3) + En\,S(2)\,\overline{S(1)}\,\overline{S(0)}\,D(4) + En\,S(2)\,\overline{S(1)}\,S(0)\,D(5)$$
$$+ En\,S(2)\,S(1)\,\overline{S(0)}\,D(6) + En\,S(2)\,S(1)\,S(0)\,D(7)$$

```
Library IEEE;
use IEEE.std_logic_1164.ALL;

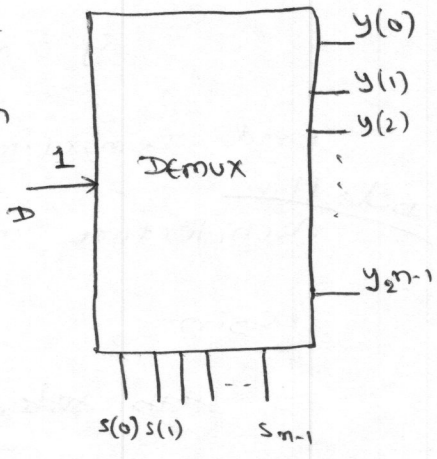
Entity mux8to1 is
    Port ( En : In std_logic; D : In std_logic_vector(7 downto 0);
           S : In std_logic_vector(2 downto 0); y : out std_logic);
End mux8to1;
```

```
Architecture mux of mux8to1 is
signal x : std_logic;
begin
 with s select
  x<= D(0) when "000",
      D(1) when "001",
      D(2) when "010",
      D(3) when "011",
      D(4) when "100",
      D(5) when "101",
      D(6) when "110",
      D(7) when "111";
 y<= x when En='1' else
      y<='0';
 End mux;
```



Demultiplexers:

The demultiplexer is a combinational logic circuit designed to switch one common input line to one of several output line by the application of selection line.



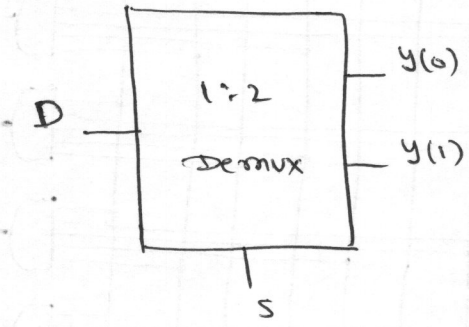
The demux takes one single i/p data line and then switches it to any one of a number of individual output lines one at a time.

Demux converts a serial data signal at the i/p to a parallel data at its o/p lines

Types of Demux

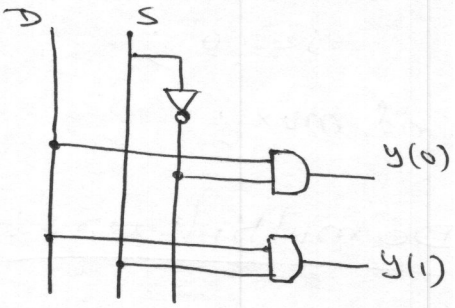
1. 1:2 Demux
2. 1:4 Demux
3. 1:8 Demux

1:2 Demux



D	S	y(1)	y(0)
0	x	0	0
1	0	0	1
1	1	1	0

$y(0) = \bar{S}D$, $y(1) = D \cdot S$




```
Library IEEE;
use IEEE.std_logic_1164.All;
Entity Demux1to2 is
  Port (D, S : In std_logic;
        y : out std_logic_vector (1 downto 0));
End Demux1to2;
```

Dataflow

```
Architecture Demux of Demux1to2 is
begin
  y(0) <= D and NOT S;
  y(1) <= D and S;
End Demux;
```

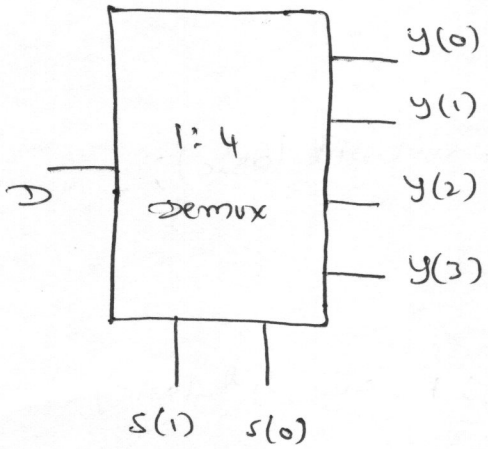
structural

```
Architecture demux of demux1to2 is
  component NOTgate
    port (A: In std_logic; B: out std_logic);
  End component;
  component ANDgate
    port (C, E: In std_logic; F: out std_logic);
  End component;
  [illegible]
    signal t: std_logic;
begin
U1: NOTgate port map (S, t);
U2: ANDgate port map (D, t, y(0));
U3: ANDgate port map (D, S, y(1));
End demux;
```

Behavioural

```
Architecture Demux of Demux1to2 is
  begin
    Process (D, S)
    begin
     if (S='0') then  y(0) <= D;
     else  y(1) <= D;
     End if;
     End process;
End Demux;
```

1:4 Demux

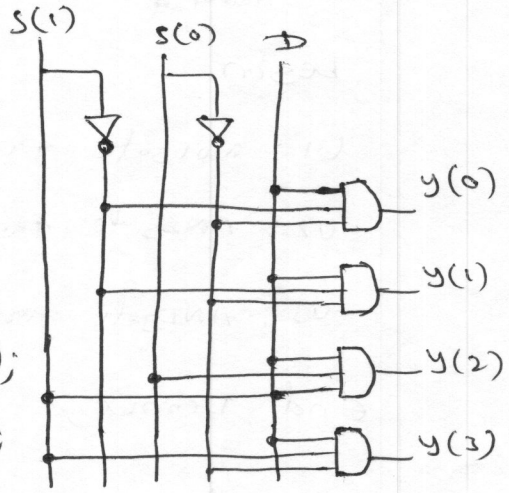


D	S(1)	S(0)	y(3)	y(2)	y(1)	y(0)
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

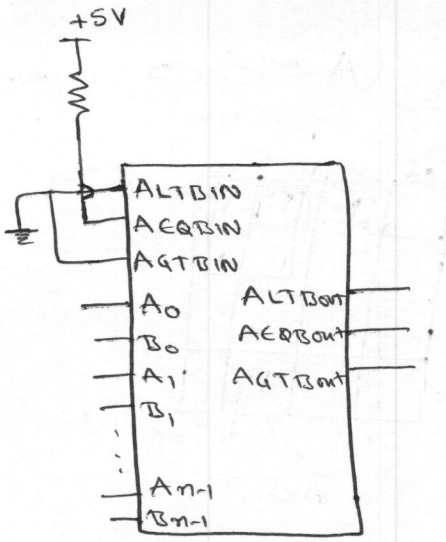
$y(0) = D \cdot \overline{S(1)}\ \overline{S(0)}$; $y(1) = D \cdot \overline{S(1)}\ S(0)$;

$y(2) = D \cdot S(1) \cdot \overline{S(0)}$; $y(3) = D \cdot S(1) \cdot S(0)$;

```
Library IEEE;
USE IEEE.std_logic_1164.ALL;
Entity Demux1to4 is
  Port ( D : in std_logic;
         S : in std_logic_vector (1 downto 0);
         y : out std_logic_vector (3 downto 0));
End Demux1to4;
Architecture Demux of Demux1to4 is
  begin
      y(0) <= D and NOT S(1) and NOT S(0);
      y(1) <= D and NOT S(1) and S(0);
      y(2) <= D and S(1) and NOT S(0);
      y(3) <= D and S(1) and S(0);
End Demux;
```



Comparators

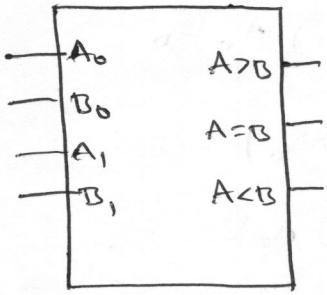


A circuit that compares two binary words and indicates whether they are equal is called a comparator.

Some comparators interpret their input words as signed or unsigned numbers and also indicate an arithmetic relationship (greater or less than) between the words. These devices are often called magnitude comparators.

1. 2-BIT COMPARATOR
2. 4-BIT COMPARATOR

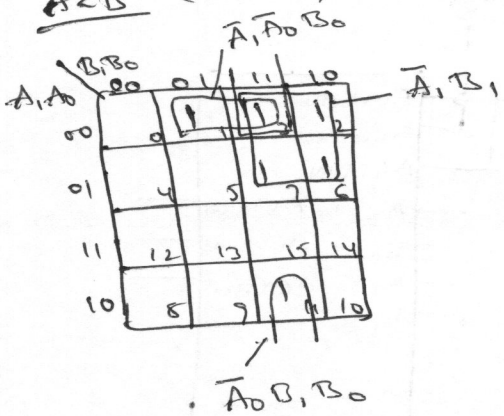
2-BIT COMPARATOR

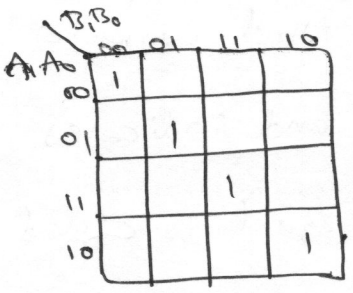


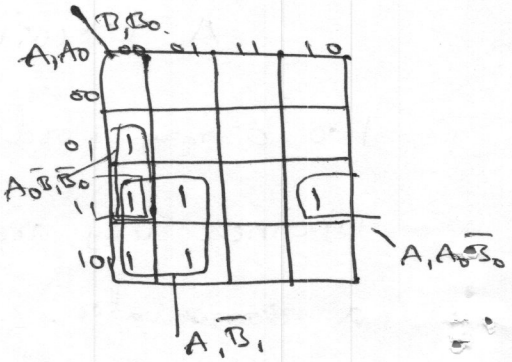
A_1	A_0	B_1	B_0	A>B	A=B	A<B
1	1	0	1	0	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

A_1	A_0	B_1	B_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0

A<B ~~(A<B)~~ ~~A,A0(A0B0)A~~ A=B A>B







$A<B = \bar{A}_1B_1 + \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0$

$(A=B) = \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 + A_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0$

$A>B = A_1\bar{B}_1 + A_0\bar{B}_1\bar{B}_0 + A_1A_0\bar{B}_0$

$(A<B) = \bar{A}_1B_1 + \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0$

$(A=B) = \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 + A_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0$

$(A>B) = A_1\bar{B}_1 + A_0\bar{B}_1\bar{B}_0 + A_1A_0\bar{B}_0$

(OR)

$(A<B) = \bar{A}_1B_1 + (A_1 \odot B_1)\bar{A}_0B_0$

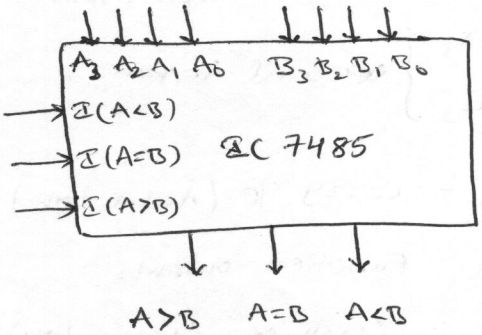
$(A=B) = (A_1 \odot B_1)(A_0 \odot B_0)$

$(A>B) = A_1\bar{B}_1 + (A_1 \odot B_1)A_0\bar{B}_0$

```
Library IEEE;
USE IEEE.std_logic_1164.ALL;
Entity comp2bit is
   Port ( A,B:in std_logic_vector(1 downto 0);
          AGTB, AEQB, ALTB: out std_logic);
End comp2bit;
```

```
Architecture comparator of comp2bit is
begin
    ALTB <= (NOT A(1) and B(1)) or ((A(1) xnor B(1)) and NOT A(0) and B(0));
    AEQB <= (A(1) xnor B(1)) and (A(0) xnor B(0));
    AGTB <= (A(1) and NOT B(1)) or ((A(1) xnor B(1)) and A(0) and NOT B(0));
End comparator;
```

4-bit comparator



comparing inputs A B	cascading inputs I(A>B)	I(A=B)	I(A<B)	outputs A>B	A=B	A<B
A>B	X	X	X	1	0	0
A=B	1	0	0	1	0	0
	X	1	X	0	1	0
	0	0	1	0	0	1
	0	0	0	1	0	1
	1	0	1	0	0	0
A<B	X	X	X	0	0	1

$$(A<B) = \overline{A}_3 B_3 + \epsilon_3 \overline{A}_2 B_2 + \epsilon_3 \epsilon_2 \overline{A}_1 B_1 + \epsilon_3 \epsilon_2 \epsilon_1 \overline{A}_0 B_0$$

$\epsilon_3 = A_3 \odot B_3$

$\epsilon_2 = A_2 \odot B_2$

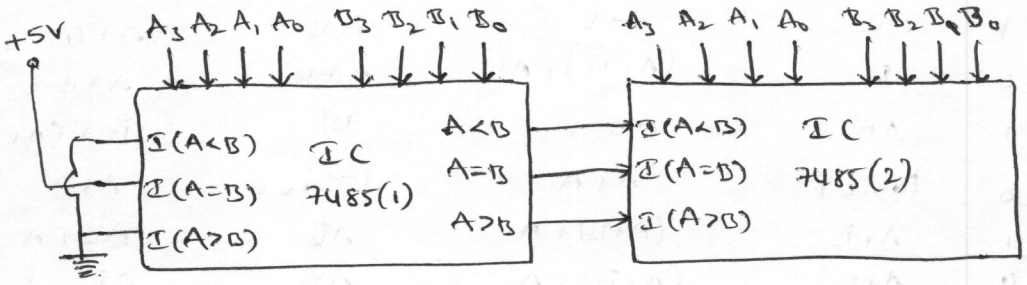
$\epsilon_1 = A_1 \odot B_1$

$\epsilon_0 = A_0 \odot B_0$

$$(A=B) = \epsilon_3 \cdot \epsilon_2 \cdot \epsilon_1 \cdot \epsilon_0$$

$$(A>B) = A_3 \overline{B}_3 + \epsilon_3 A_2 \overline{B}_2 + \epsilon_3 \epsilon_2 A_1 \overline{B}_1 + \epsilon_3 \epsilon_2 \epsilon_1 A_0 \overline{B}_0$$

8-bit comparator

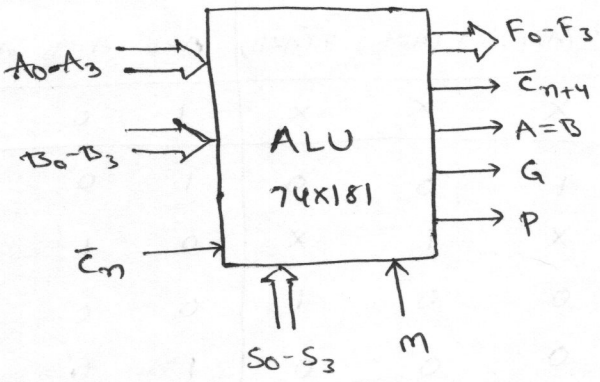


Arithmetic logic unit (ALU)

→ The ALU performs all the necessary arithmetic and logical operations.

→ It requires one or two operands upon which it operates and produces a result.

→ It is basically a multifunction combinational logic circuit. It provides select input to select the particular operation.



S_0-S_3 – select Function inputs

M – mode control input

A_0-A_3, B_0-B_3 } operand inputs

$\overline{C}_n$ – Carry i/p (Active Low)

F_0-F_3 Function outputs

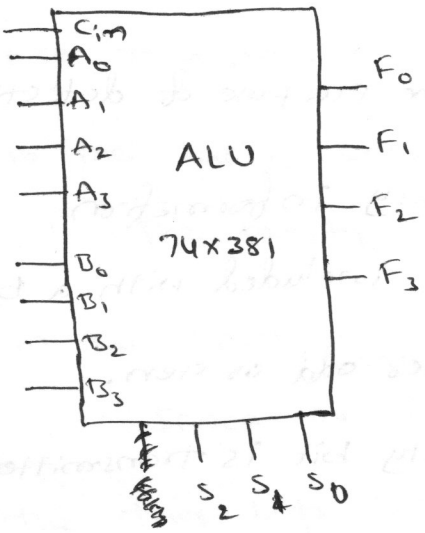
$\overline{C}_{n+4}$ – Carry o/p (Active Low)

A=B – comparator Equal o/p

P, G – Carry propagate, Generate o/p

				Active HIGH operands		Active LOW operands	
S_3	S_2	S_1	S_0	M=1	(M=0) ($\overline{C}_n$=1)	M=1	(M=0) ($\overline{C}_n$=0)
0	0	0	0	$F=\overline{A}$	F=A	$F=\overline{A}$	F=A-1
0	0	0	1	$F=\overline{A}+\overline{B}$	F=A+B	$F=\overline{AB}$	F=AB-1
0	0	1	0	$F=\overline{A}B$	$F=A+\overline{B}$	$F=\overline{A}+\overline{B}$	$F=A\overline{B}-1$
0	0	1	1	F=logic 0	F=-1	F=logic 1	F=-1
0	1	0	0	$\overline{A}\overline{B}$	$A+A\overline{B}$	$\overline{A}+\overline{B}$	$A+(A+\overline{B})$
0	1	0	1	$\overline{B}$	$(A+B)+A\overline{B}$	$\overline{B}$	$AB+(A+\overline{B})$
0	1	1	0	$A\overline{B}$	AB-1	$A\oplus\overline{B}$	A-B-1
0	1	1	1				
1	0	0	0	$\overline{A}+B$	A+AB	$A+\overline{B}$	$A+\overline{B}$
1	0	0	1	$\overline{A}\oplus\overline{B}$	A+B	$\overline{A}B$	A+(A+B)
1	0	1	0	B	$(A+\overline{B})+AB$	$A\oplus B$	A+B
1	0	1	1	AB	AB-1	B	$A\overline{B}+(A+B)$
1	1	0	0	logic 1	A+A	logic 0	A+A
1	1	0	1	$A+\overline{B}$	(A+B)+A	$A\overline{B}$	AB+A
1	1	1	0	A+B	$(A+\overline{B})+A$	AB	$A\overline{B}-A$
1	1	1	1	A	A-1	A	A

www.Jntufastupdates.com



Inputs			output
S_2	S_1	S_0	F
0	0	0	0000
0	0	1	B-A-1+Cin
0	1	0	A-B-1+Cin
0	1	1	A+B+Cin
1	0	0	A⊕B
1	0	1	A+B
1	1	0	A.B
1	1	1	1111

```
Library IEEE;
USE IEEE.std_logic_1164.ALL; USE IEEE.std_logic_unsigned.ALL;
Entity ALU is
    Port ( A,B : In std_logic_vector (3 downto 0);
           S : In std_logic_vector (2 downto 0);
           CIN : In std_logic ;
           F : out std_logic_vector (3 downto 0));
End ALU;
Architecture ALU of ALU is
begin
  process (S, A, B, CIN)
    begin
      case S is
        when "000" => F <= "0000";
        when "001" => F <= B-A-1+CIN;
        when "010" => F <= A-B-1+CIN;
        when "011" => F <= A+B+CIN;
        when "100" => F <= A xor B;
        when "101" => F <= A or B;
        when "110" => F <= A and B;
        when "111" => F <= "1111";
End process; End case;
End ALU;
```

www.Jntufastupdates.com

Parity Circuits

A parity bit is used for the purpose of detecting errors during transmission of binary information.

A parity bit is an extra bit included with a binary message to make the no. of 1's either odd or even.

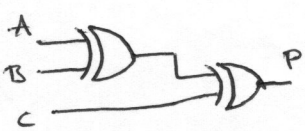
The message, including the parity bit is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity doesn't correspond with the one transmitted.

The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity in the receiver is called a parity checker.

In even parity the added parity bit will make the total number of 1s an even amount.

In odd parity the added parity bit will make the total number of 1s an odd amount.

3-bit message			odd parity bit	even parity bit
A	B	C		
0	0	0	1	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	0	1



$P_{even} = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

$= A \oplus B \oplus C$

$P_{odd} = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$

$= (\bar{A}B + A\bar{B})C + (\bar{A}\bar{B} + AB)\bar{C}$

$= A \odot B \odot C$

The three bits in the message together with Parity bit are transmitted to their destination, where they are applied to the parity checker circuit.

The parity checker circuit checks for possible errors in the transmission.

Since the information was transmitted with even Parity, the four bits received must have an even number of 1s. Then error occurs during transmission if four bits received have an odd no. of 1s.

The output of Parity checker, is denoted by PEC (Parity error check). It will be equal to 1, if an error occurs, i.e if the four bits received have an odd no. of 1's.

Four bits Received				Parity error check
A	B	C	D	PEC
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

$PEC = A \oplus B \oplus C \oplus D$

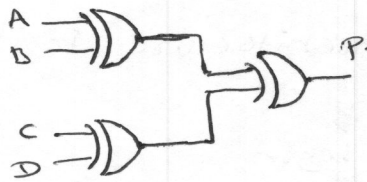
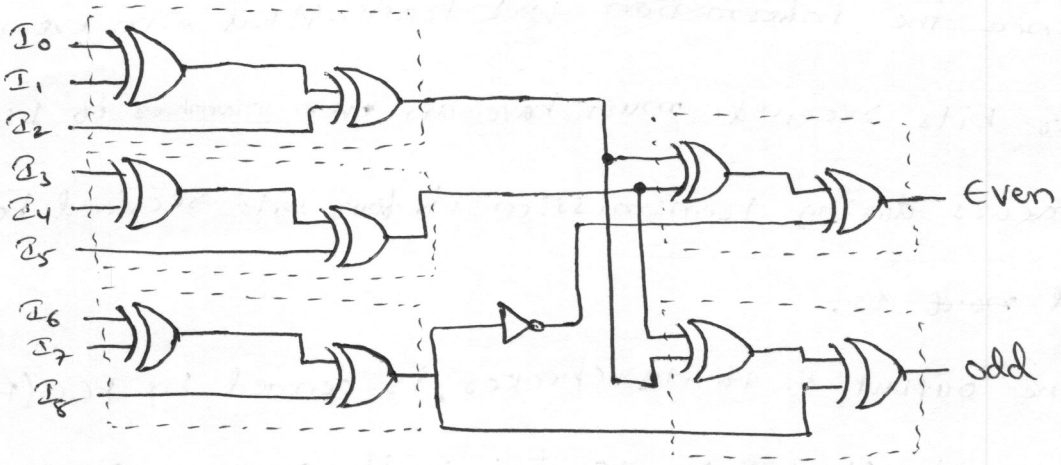


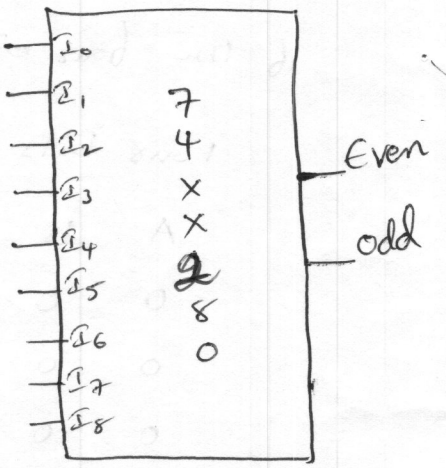
Table: For even parity checker

www.Jntufastupdates.com

9-Bit Parity Generator/checker (74xx280)

9-bit Parity generator or checker commonly used to detect errors in high speed data transmission or data retrieval systems.





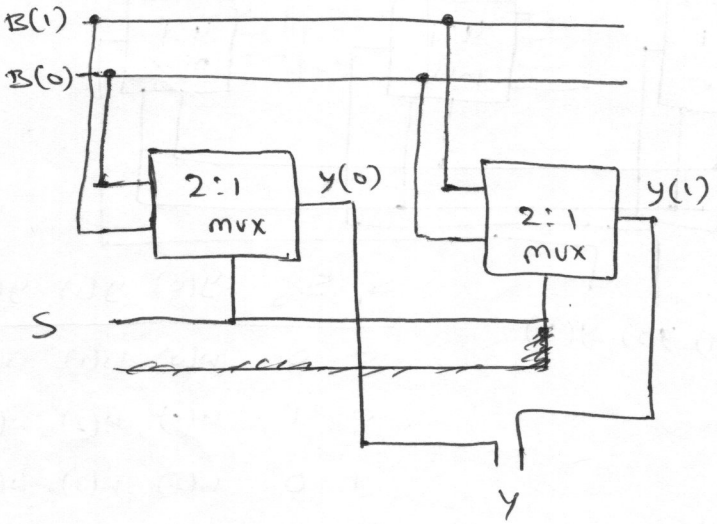

```
Library IEEE;
USE IEEE.std_logic_1164.ALL;
Entity Para9_bit is
  Port (I: In std_logic_vector(8 downto 0);
    Even, odd: out std_logic);
End Para9_bit;
Architecture Parity Of Para9_bit is
begin
  Process (I)   variable P: std_logic;
    begin
      P:=I(0);
    For J in 1 to 8 loop
      if I(J)='1' then P:=not P;
      End if;
    End loop;          ODD<=P; Even<=not P;
                       End Process;
                    End Parity;
```

Barrel shifter

→ A barrel shifter is a combinational logic circuit.

→ It has n data inputs, n data outputs, and a set of control inputs that specify how to shift the data between input and output.

2-bit



S	Y(0)	y(1)
0	B(0)	B(1)
1	B(1)	B(0)

```
Library IEEE;
USE IEEE. std. logic. 1164. ALL;

Entity BS2B
   Port ( B: In std_logic_vector(1 downto 0);
          S : In std_logic ;
          Y : out std_logic_vector(1 downto 0));
End BS2B;
Architecture BS of BS2B is
begin
   Process (B, S)
     begin
       case S is
          when S= '0' => Y<= B;
          when S='1' => Y <= B ROR 1 ;
```

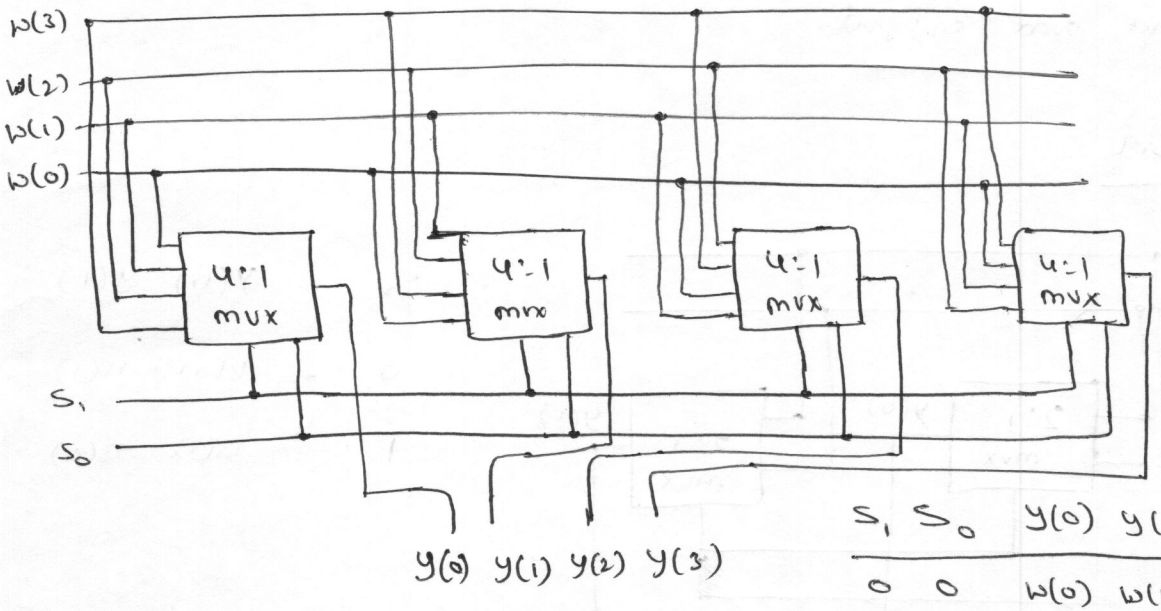
www.Jntufastupdates.com

End case;

End process;

End BS;

4 - bit



S_1	S_0	y(0)	y(1)	y(2)	y(3)
0	0	w(0)	w(1)	w(2)	w(3)
0	1	w(1)	w(2)	w(3)	w(0)
1	0	w(2)	w(3)	w(0)	w(1)
1	1	w(3)	w(0)	w(1)	w(2)

```
Library IEEE;
USE IEEE.std_logic_1164.ALL;
Entity BS4B is
    Port ( w: In std_logic_vector(0 to 3);
           s : In std_logic_vector (1 downto 0);
           y : out std_logic_vector (0 to 3));
End BS4B;
Architecture BS of BS4B is
 begin
   Process (B,S)
    begin
      case S is
       when "00" => y <= w;
       when "01" => y <= w(1) & w(2) & w(3) & w(0);
```

www.Jntufastupdates.com

```
When "10" => y <= w ROR 2;
When others => y <= w ROR 3;
End case;
End process;
End BS;
```

Simple floating-Point Encoder

In Simple floating-Point Encoder, the encoding of 11-bit fixed-point number to a 7-bit floating Point number with 4-bits for mantissa and 3-bits for exponent. Therefore, in this the smallest integer is $0 \cdot 2^0$ and the largest integer is $(2^4 - 1) \cdot 2^7$.

B_{10}	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	$m_3 m_2 m_1 m_0 \cdot e_2 e_1 e_0$	Truncation error
1	0	1	1	1	0	1	0	0	1	0	= 1011 · 111	+ 1010010
0	0	1	0	1	0	0	0	0	1	0	= 1010 · 101	+ 00100
0	0	0	0	0	0	1	1	1	0	0	= 1110 · 001	+ 0
0	0	0	0	0	0	0	0	1	0	0	= 0100 · 000	+ 0

The relationship between fixed-point number and floating point number can be

$$B = m \cdot 2^e + T \qquad 0 \le T \le 2^e$$

Ex: $B = 1010 \cdot 2^5 + 00100 = 00101000100$

1. 00010110110 $= 1011 \times 2^4 + 0110$

   m - 1011
   e - 4 - 100
   T - 0110

2. 10100111101 $= 1010 \times 2^7 + 0111101$

```
Library IEEE;
use IEEE.std_logic_1164.All;
use IEEE.std_logic_arith.All;
entity FPENC is
    Port (B: In std_logic_vector(10 downto 0);
          M: out std_logic_vector(3 downto 0);
          E: out std_logic_vector(2 downto 0));
End FPENC;
Architecture FP of FPENC is
 begin
   Process (B)
     begin
      if B(10) = '1' then M<=B(10 downto 7); E<="111";
      elsif B(9) = '1' then M<=B(9 downto 6); E<="110";
      elsif B(8) = '1' then M<=B(8 downto 5); E<="101";
      elsif B(7) = '1' then M<=B(7 downto 4); E<="100";
      elsif B(6) = '1' then M<=B(6 downto 3); E<="011";
      elsif B(5) = '1' then M<=B(5 downto 2); E<="010";
      elsif B(4) = '1' then M<=B(4 downto 1); E<="001";
      else                  M<=B(3 downto 0); E<="000";
      end if;
   End process;
 End FP;
```