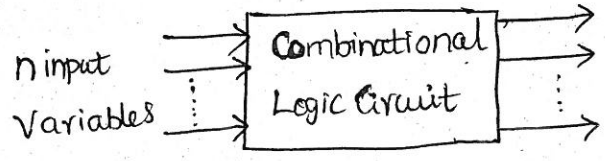
Unit- III - Combinational Logic Circuits.

Combinational Logic Circuit:

→ When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage device.

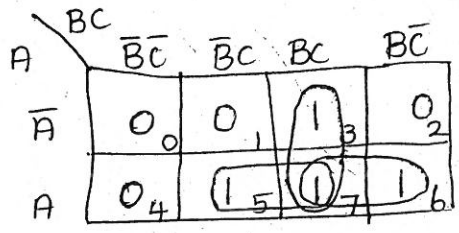


→ The combinational circuit consists of i/p variables, Logic gates, and output variables. The Logic gates accept signals from the i/p variables and generate output variables.

(1). Design a Combination Logic circuit with 3 input variables that will produce a logic 1 output when more than one input variables are logic 1.

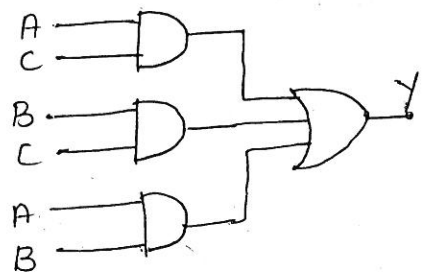
input variables A	B	C	output (Y)	
0	0	0	0	0
0	0	1	0	1
0	1	0	0	2
0	1	1	1	3
1	0	0	0	4
1	0	1	1	5
1	1	0	1	6
1	1	1	1	7

Simplified Boolean Expression,



$$Y = AC + BC + AB$$

Logic diagram,



Adders: → The simple addition consists of four possible basic operations,

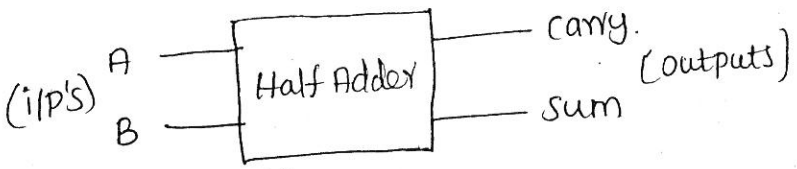
0+0=0 ; 0+1=1 ; 1+0=1 ; 1+1=10 ;

Types:

* Half Adder (The circuit which performs addition of 2 bits)
* Full Adder (The circuit which performs addition of 3 bits)

Half Adder:-

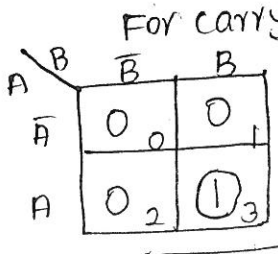
→ Half adder operation needs 2 binary inputs, and 2 binary outputs (sum & carry).



inputs		outputs	
A	B	carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

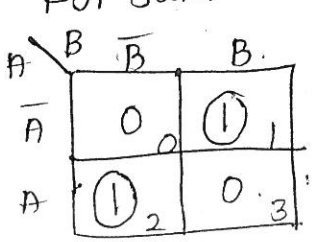
K map Simplification for carry & sum,

For carry



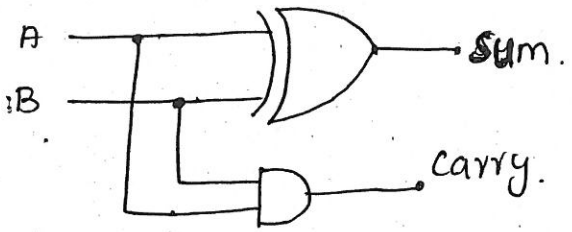
Carry = AB

For sum

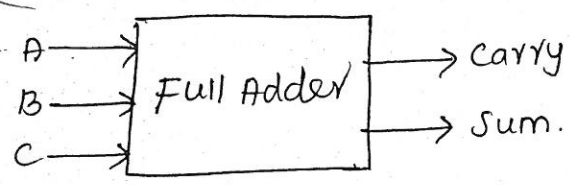


Sum = $A\bar{B} + \bar{B}\bar{A}$ ⇒ A ⊕ B.

Logic diagram,



Full-Adder: It contains 3 inputs (A, B, C) & Two outputs (sum & ca

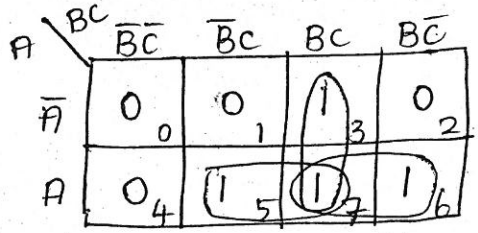


Truth Table

Inputs			output	
A	B	C	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

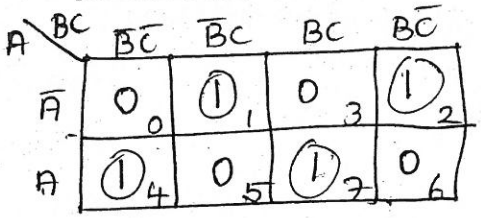
K-map Simplifications,

For carry (Cout)

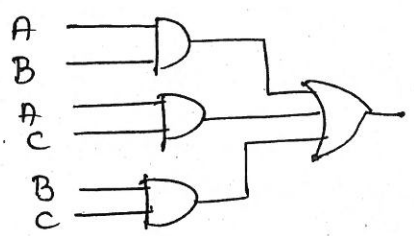


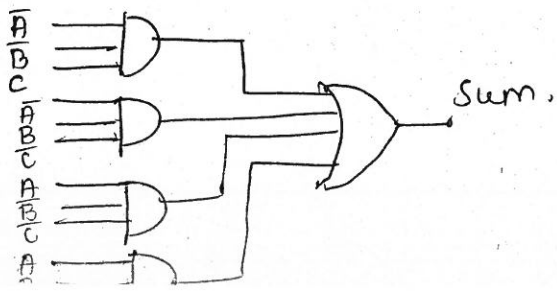
$$Cout = AB + AC + BC$$

For Sum



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





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

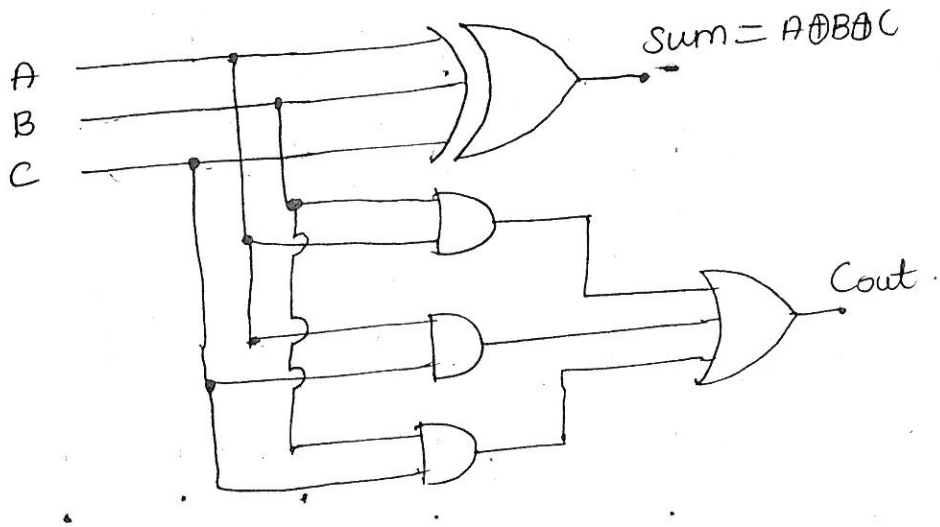
$= C(\bar{A}\bar{B} + AB) + \bar{C}(\bar{A}B + A\bar{B})$

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

$\boxed{Sum = C \oplus A \oplus B}$

$\boxed{\overline{A \oplus B} = \bar{A}\bar{B} + AB}$

Implementation of Full adder,



1) Implement Full Adder Using 2 Half Adders?

$Sum = A \oplus B \oplus C$

$Carry = AB + AC + BC$
(Cout)

$Cout = AB + AC + BC(A + \bar{A})$ $\quad AB(C + \bar{C}) + A(B + \bar{B})C + (A + \bar{A})BC$

$= AB + AC + ABC + \bar{A}BC \Rightarrow A\bar{B}\bar{C} + AB\bar{C} + A\bar{B}C + ABC + \bar{A}BC.$

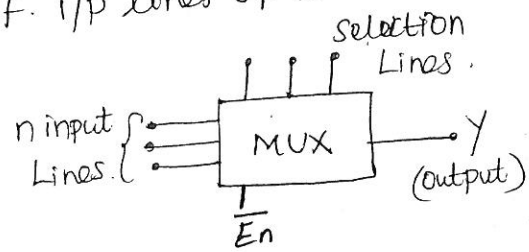
$= ABC + AB\bar{C} + A\bar{B}C + \bar{A}BC \Rightarrow AB(C + \bar{C}) + (A\bar{B} + \bar{A}B)C.$

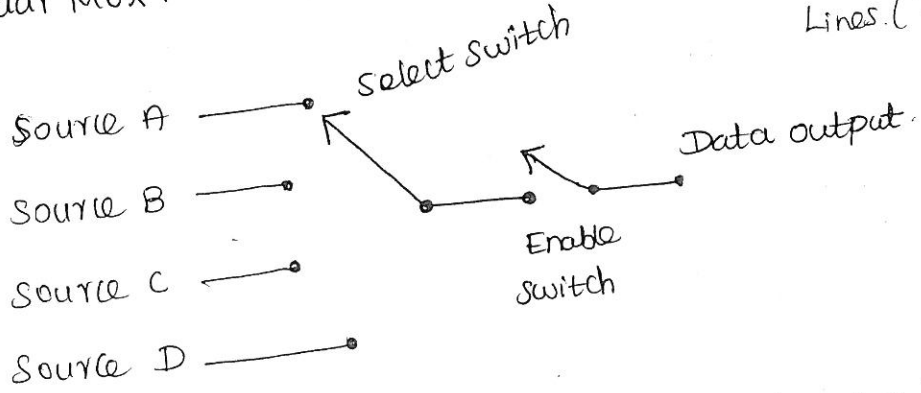
$\boxed{Cout = AB + C(A \oplus B)}$

Multiplexers :- [Data selector]

→ Multiplexer is a digital switch. It is a combinational circuit that select one digital information from several sources and transmit the selected information on a single o/p line.

→ The selection line is used to decide the no. of i/p lines of a particular MUX.

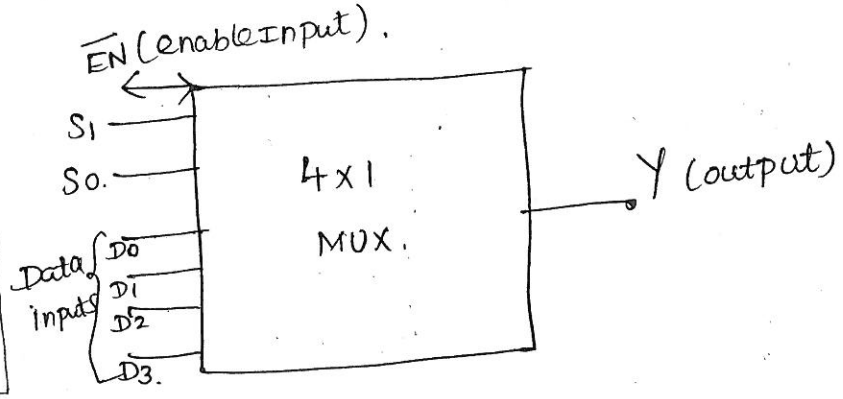


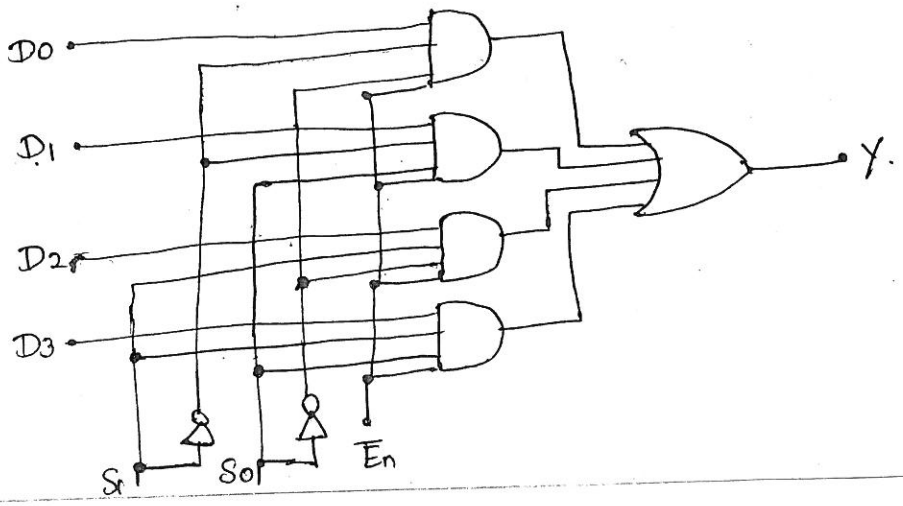


4 to 1 MUX

$n = 4\ (D_0 - D_3)$, $M = 2\ (S_1 \& S_0)$, $Y = 1$.

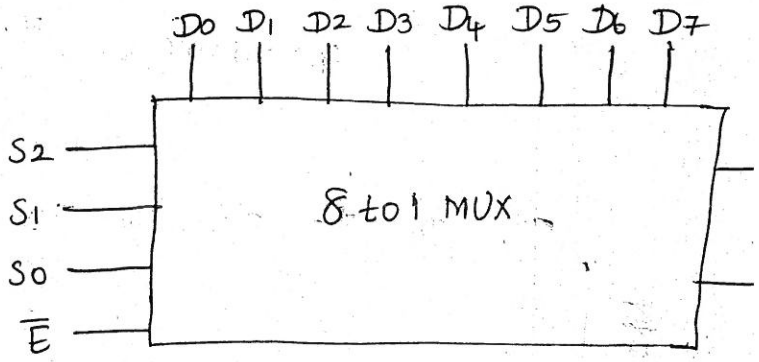
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



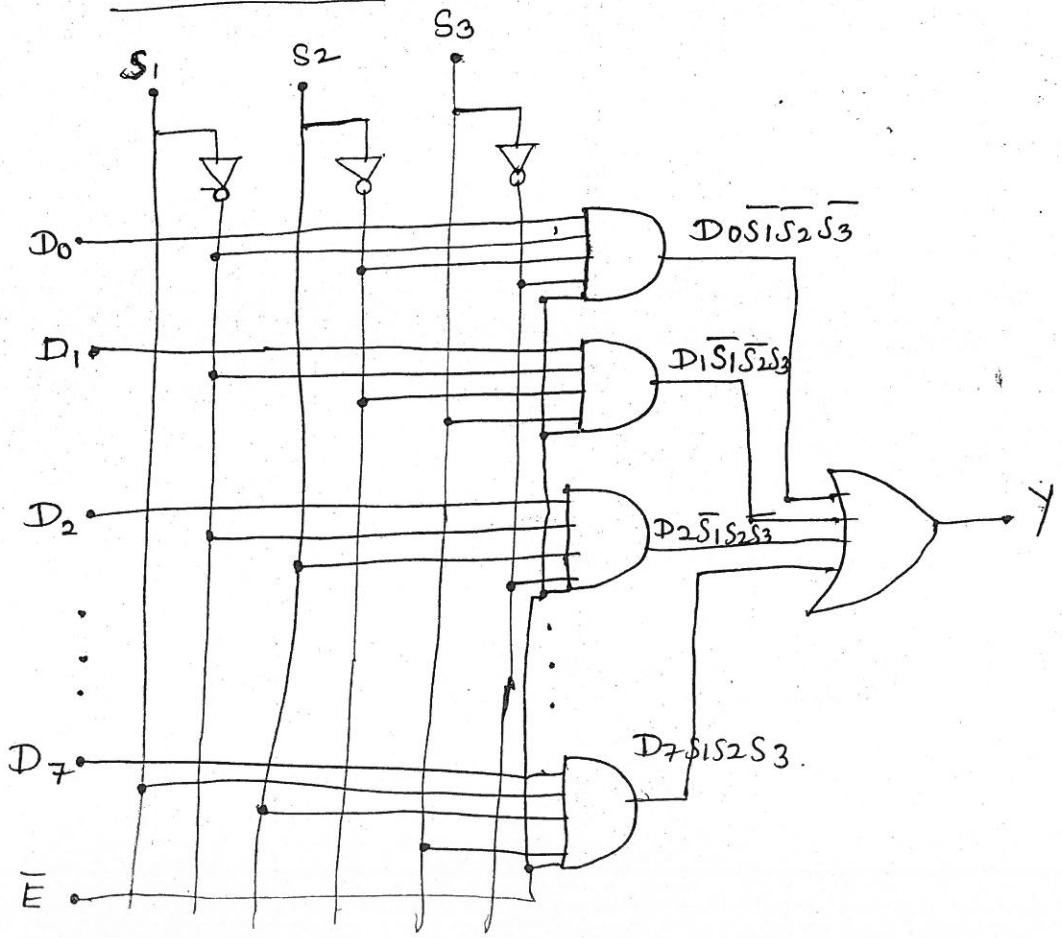


8 to 1 MUX :- $n=8(D_0-D_7)$, $Y=1$, $m=2^n=3(S_0,S$

Inputs		Outputs	
$S_2\ S_1\ S_0$	$\overline{E}$	Y	$\overline{Y}$
X X X	1	1	0
0 0 0	0	D_0	$\overline{D_0}$
0 0 1	0	D_1	$\overline{D_1}$
0 1 0	0	D_2	$\overline{D_2}$
0 1 1	0	D_3	$\overline{D_3}$
1 0 0	0	D_4	$\overline{D_4}$
1 0 1	0	D_5	$\overline{D_5}$
1 1 0	0	D_6	$\overline{D_6}$
1 1 1	0	D_7	$\overline{D_7}$



Logic Diagram



1) Implement the following boolean function with 8:1 MUX,

$F(A,B,C,D) = \Pi M(0,3,5,8,9,10,12,14)$

2) Implement the following boolean function with 8:1 MUX

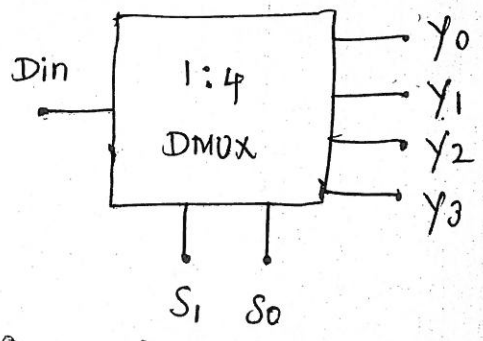
$F(A,B,C,D) = \Sigma m(0,2,6,10,11,12,13) + d(3,8,14)$.

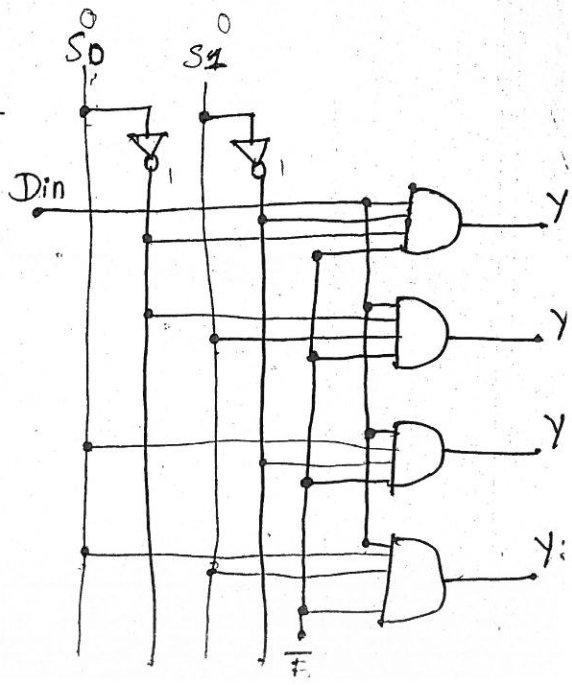
Demultiplexers :-

→ It receives the information on a single line and transmits this information on one of 2^n possible output lines. The selection of specific output line is controlled by the values of n selection lin

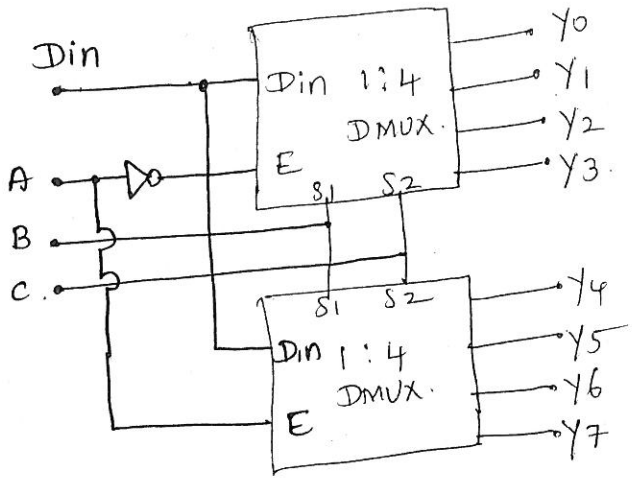
1 to 4 DMUX.

Enable	S_1	S_0	Din	y_0	y_1	y_2	y_3
1	X	X	X	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1





2) Design 1:8 demultiplexer using two 1:4 multiplexers.



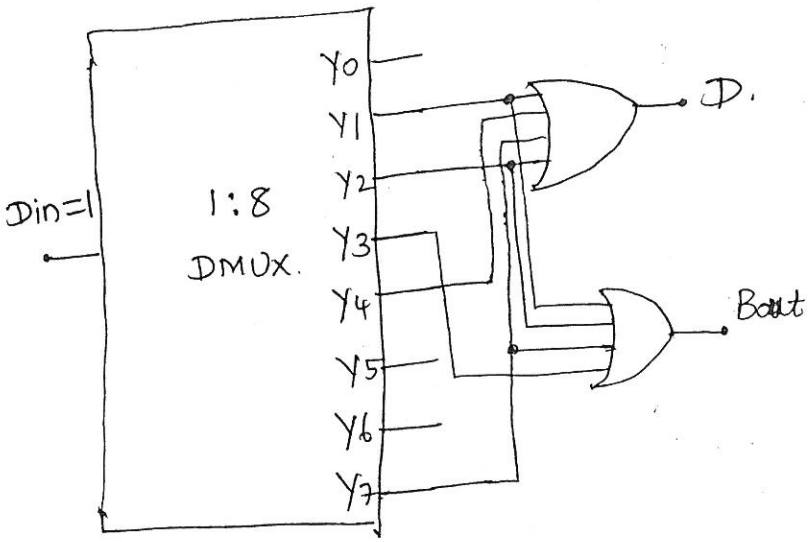
3) Implement full subtractor using demultiplexer.

Truth Table

A	B	Bin	D	Bout
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

$D \Rightarrow f(A,B,C) = \Sigma m(1,2,4,7)$.

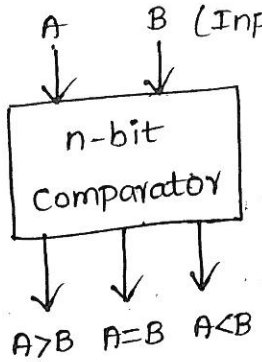
$Bout \Rightarrow f(A,B,C) = \Sigma m(1,2,3,7)$.



14/8/19

4, 6, 7, 8, 9, 12, 15, 21, 22, 23, 31, 32, 33, 34, 36, 39, 41, 44, 46, 52, 59, 60,

Magnitude Comparator :-



→ A comparator is a Special Combinational ckt designed primarily to compare the relative magnitude of 2 binary numbers.

→ It receives 2 n-bit numbers A & B as inputs and outputs are A>B, A=B, & A<B.

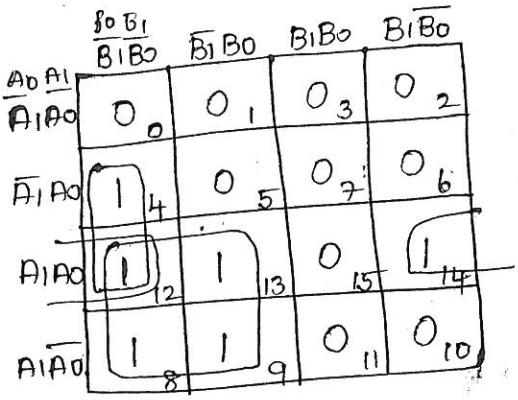
→ Depending upon the relative magnitudes of the 2 number, one of the outputs will be high.

1). Design 2-bit comparator using gates.

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

K-map Simplification,

(A>B)



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

(A_1; A_0B_0; B_1)

(A=B)

	$\bar{B}_1\bar{B}_0$	$\bar{B}_1B_0$	B_1B_0	$B_1\bar{B}_0$
$\bar{A}_1\bar{A}_0$	1 (0)	0 (1)	0 (3)	0 (2)
$\bar{A}_1A_0$	0 (4)	1 (5)	0 (7)	0 (6)
A_1A_0	0 (12)	0 (13)	1 (15)	0 (14)
$A_1\bar{A}_0$	0 (8)	0 (9)	0 (11)	1 (10)

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

$= \bar{A}_1\bar{B}_1(\bar{A}_0\bar{B}_0 + A_0B_0) + A_1B_1(A_0B_0 + \bar{A}_0\bar{B}_0)$

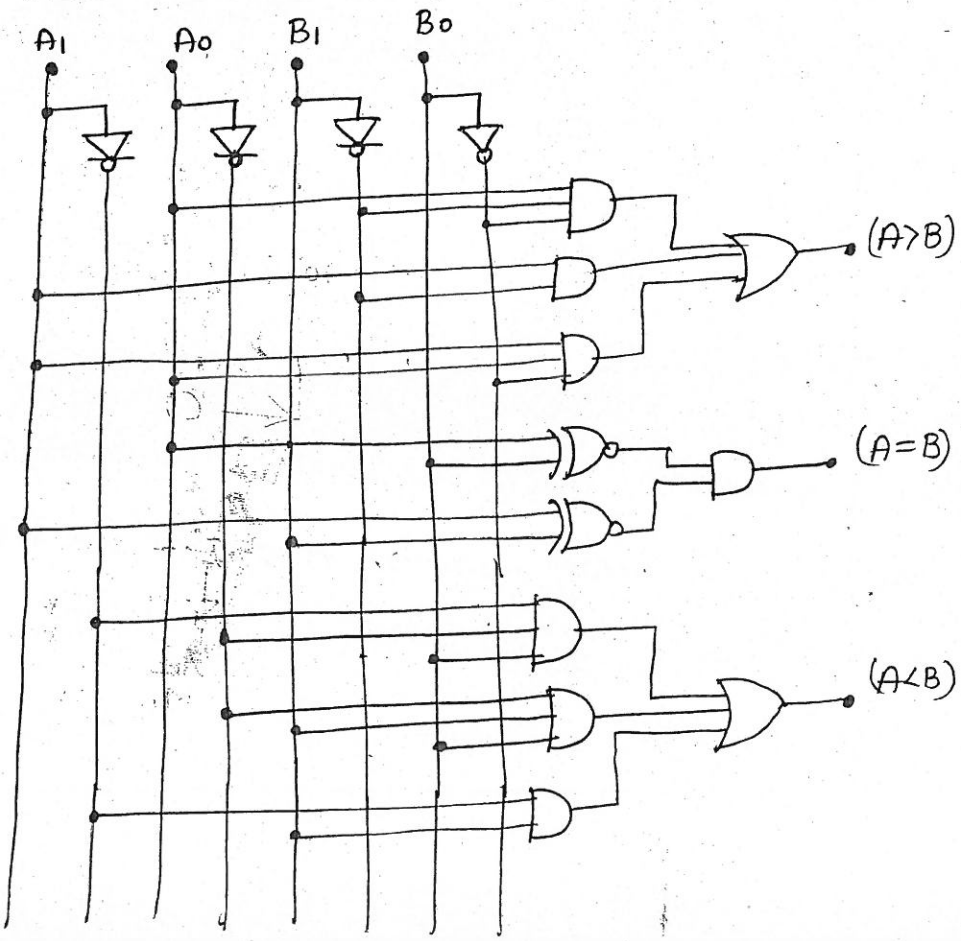
$$\boxed{(A=B) = (A_0 \odot B_0)(A_1 \odot B_1)}$$

(A<B)

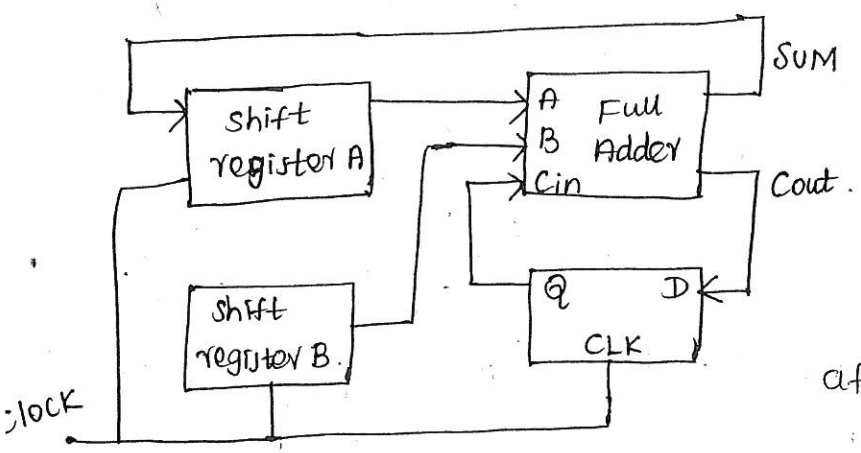
	$\bar{B}_1\bar{B}_0$	$\bar{B}_1B_0$	B_1B_0	$B_1\bar{B}_0$
$\bar{A}_1\bar{A}_0$	0 (0)	1 (1)	1 (3)	1 (2)
$\bar{A}_1A_0$	0 (4)	0 (5)	1 (7)	1 (6)
A_1A_0	0 (12)	0 (13)	0 (15)	0 (14)
$A_1\bar{A}_0$	0 (8)	0 (9)	1 (11)	0 (10)

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

Logic Diagram,



Serial Adder / subtractor :-

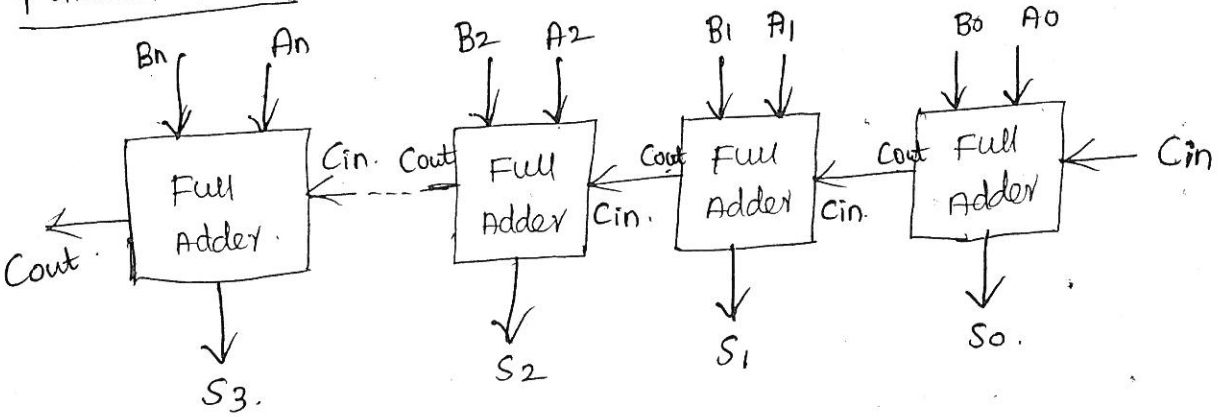


→ Full adder is used to perform bit by bit addition and D-flip flop is used to store the carry output generated after addition.

→ This carry is used as carry input for the next addition. Initially, the D-flipflop is cleared and addition starts with the least significant bits of both registers.

→ After each clock pulse data within the shift registers are shifted right 1-bit and the result SUM is stored bit by bit in the register A.

Parallel Adder :-

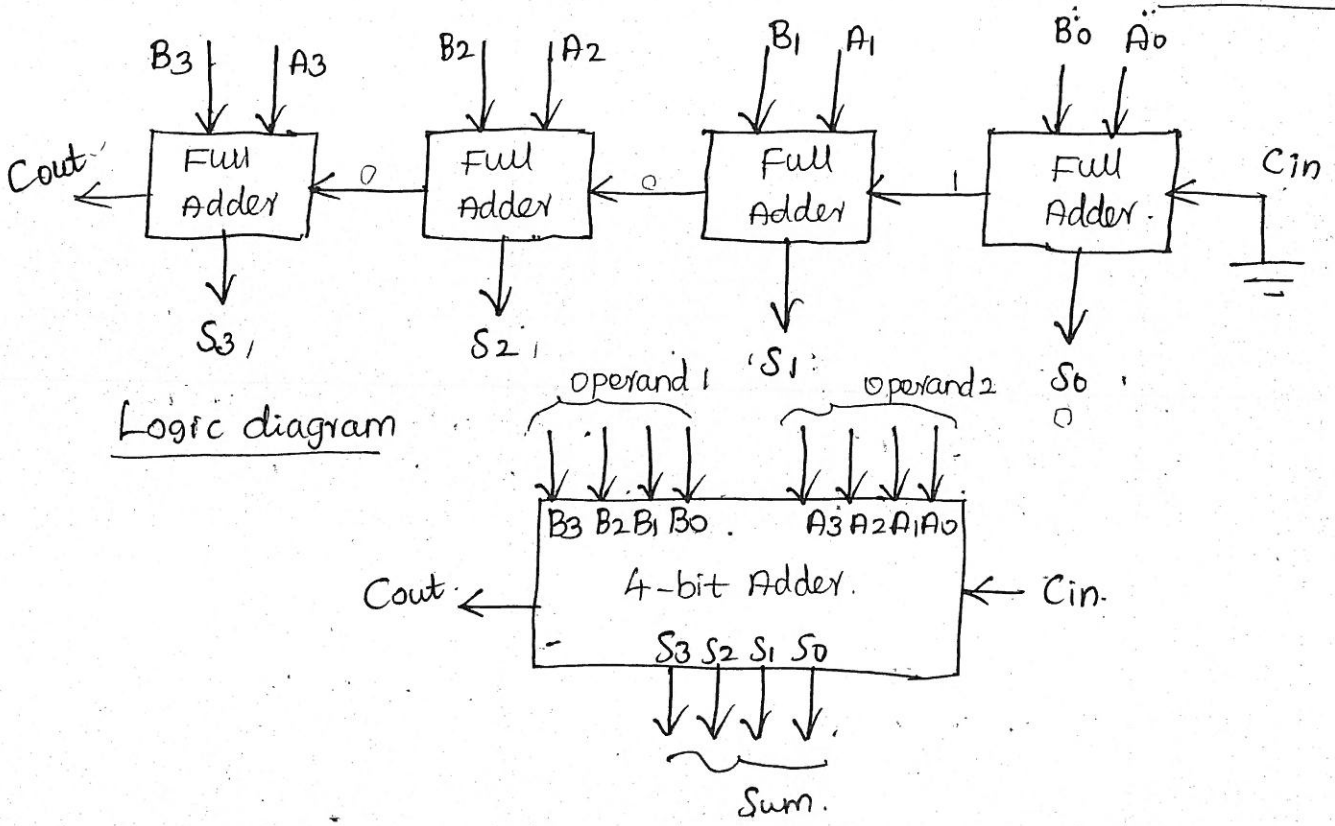


→ A single full-Adder is capable of adding two one-bit numbers and an input carry. In order to add binary numbers more than one bit, additional full-adders must be employed.

→ A n bit, parallel adder can be constructed using no. of. full ad circuits connected in parallel.

→ ie, the carry output of each adder is connected to the carry in of the next higher-order adder.

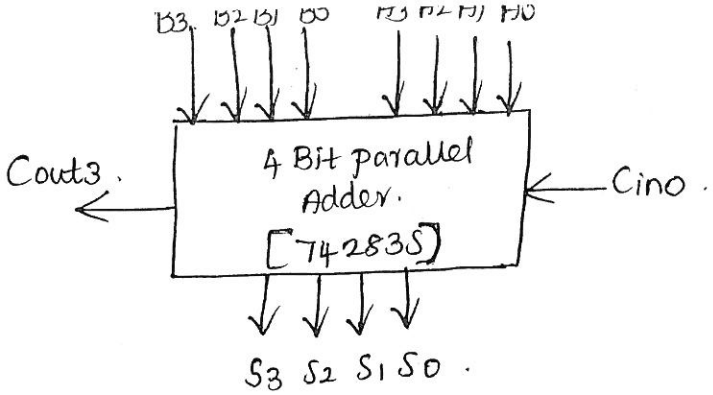
1) Design a 4-bit parallel adder using full-adders. (Block Diagra



Logic diagram

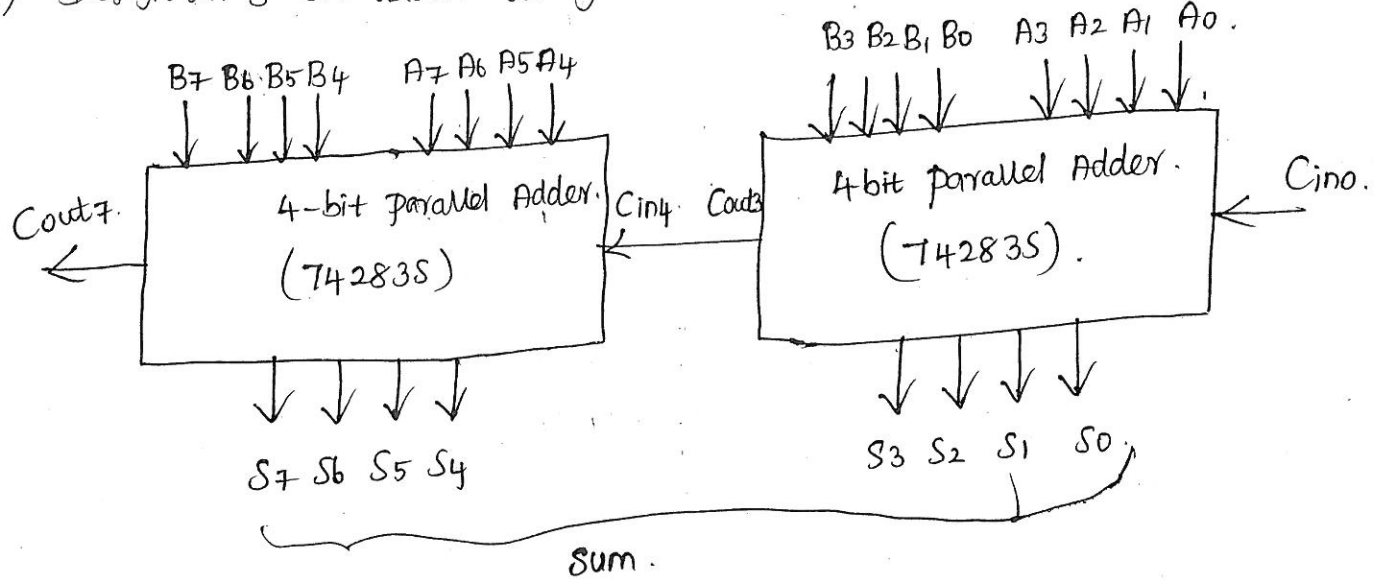
Binary parallel Adder. (IC 74LS83 / 74283)

→ Many high-speed adders available in Integrated Circuit form utilize the look-ahead carry or a similar technique for reducing overall Propagation delays. The most common is a 4-bit parallel adder IC (74LS83/74283) that contains 4 interconnected full adders and look ahead carry circuitry needed for high speed operation.

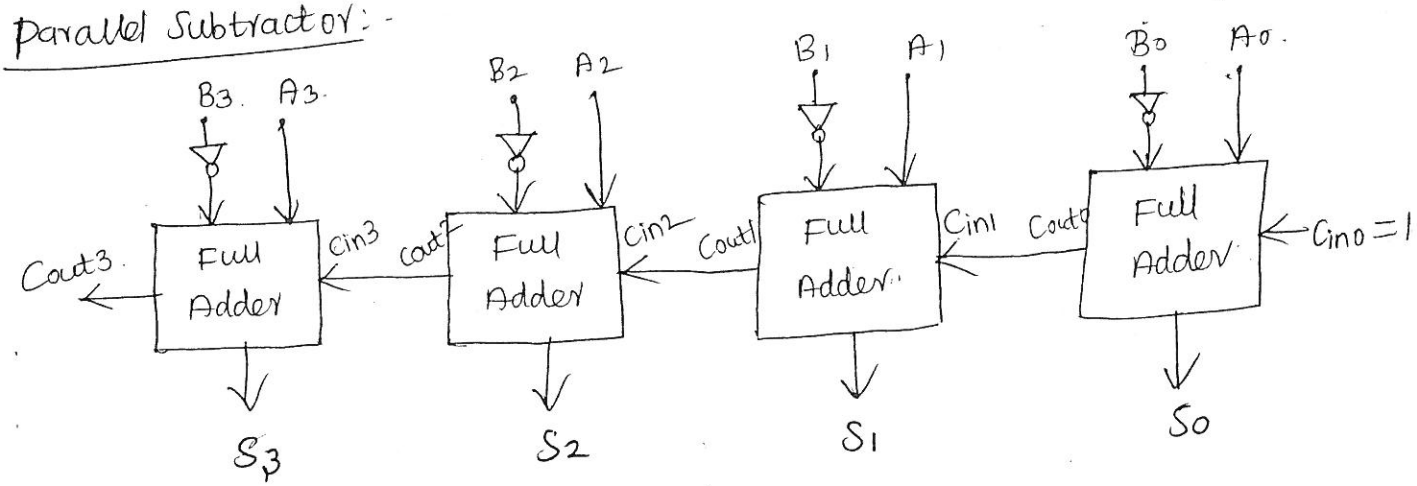


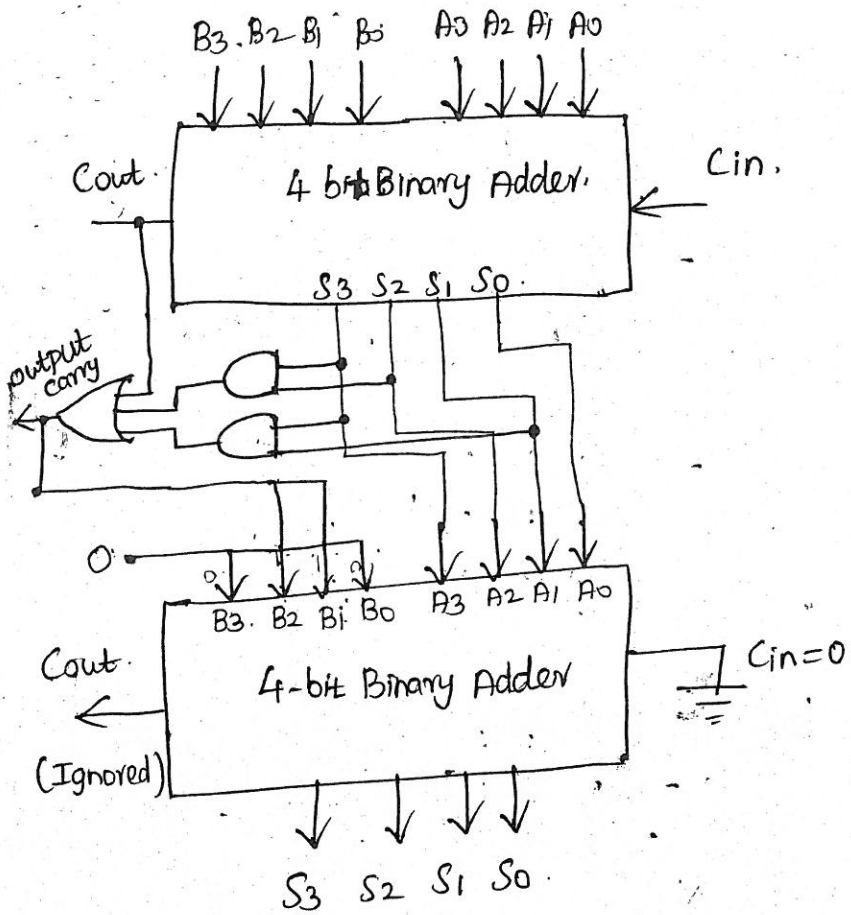
→ Input of this IC are two 4-bit numbers, $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ and the carry Cino, into the LSB position. The outputs are the sum bits $S_3S_2S_1S_0$ and the carry Cout3, output of the MSB position.

1) Design an 8-bit adder using two 74283S.



Parallel Subtractor:-





→ The 2 BCD numbers, togeth with i/p carry, are first added the top 4-bit binary adder to produce a binary sum.

→ When the o/p carry is equ to zero. [i.e, When Sum ≤ 9 c Cout = 0], nothing added to binary sum.

→ When the o/p carry is equal to 1. (i.e, When Sum > 9 and Cout = 1 binary 0110 is added to the binary sum. And the o/p carry generate from the bottom binary adder can be ignored.

BCD Adder:-

→ A BCD Adder is a circuit that adds the two BCD digits and produces a sum digit also in BCD.

* TO Implement BCD adder,

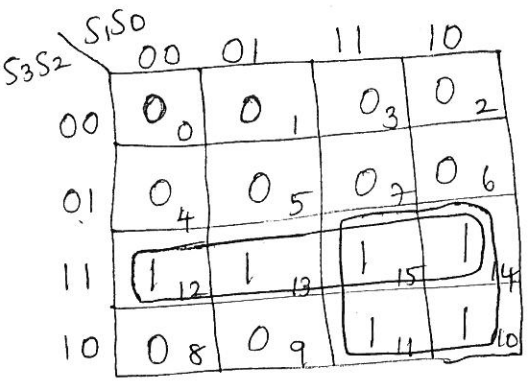
(i) 4 bit binary adder for initial addition.

(ii) Logic Circuit to detect Sum greater than 9,

(iii) One more 4 bit adder to add $(0110)_2$ in the sum if sum is greater than 9 (or) carry is 1.

Inputs				output
S_3	S_2	S_1	S_0	Y.
0	0	0	0	→ 0
0	0	0	1	→ 0
0	0	1	0	→ 0
0	0	1	1	→ 0
0	1	0	0	→ 0
0	1	0	1	→ 0
0	1	1	0	→ 0
0	1	1	1	→ 0
1	0	0	0	→ 0
1	0	0	1	→ 0
1	0	1	0	→ 1
1	0	1	1	→ 1
1	1	0	0	→ 1
1	1	0	1	→ 1
1	1	1	0	→ 1
1	1	1	1	→ 1

Kmap Simplification,



$$Y = S_3S_2 + S_3S_1$$

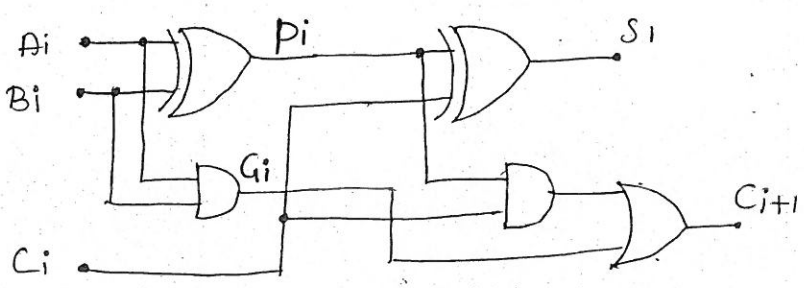
→ If sum is greater than 9, we add (0110) in the sum.

Carry Look Ahead Adder

→ In parallel adder, In which the carry output of each full-adder stag connected to the carry input of the next higher-order stage. So, t Sum & carry outputs of any stage cannot be produced until the inp carry occurs.

→ This leads time delay in the addition process → this delay known c carry propagation delay.

→ For speeding up this process by eliminating inter stage carry delay is called look ahead carry addition. This method utilizes logic gates to l at the lower-order bits. It uses two functions, (i) carry generate (ii) carry propagate.



$Pi = Ai \oplus Bi$

$Gi = Ai\,Bi$

The output Sum and carry can be expressed as,

$$Si = Pi \oplus Ci$$

$$C_{i+1} = Gi + PiCi$$

→ Here Gi is carry propagate & it produces on carry when both Ai & Bi are one, regardless of the input carry.

→ P_i is called carry propagate because it is term associated with the propagation of the carry from C_i to C_{i+1}.

→ Boolean function for the carry output of each stage,

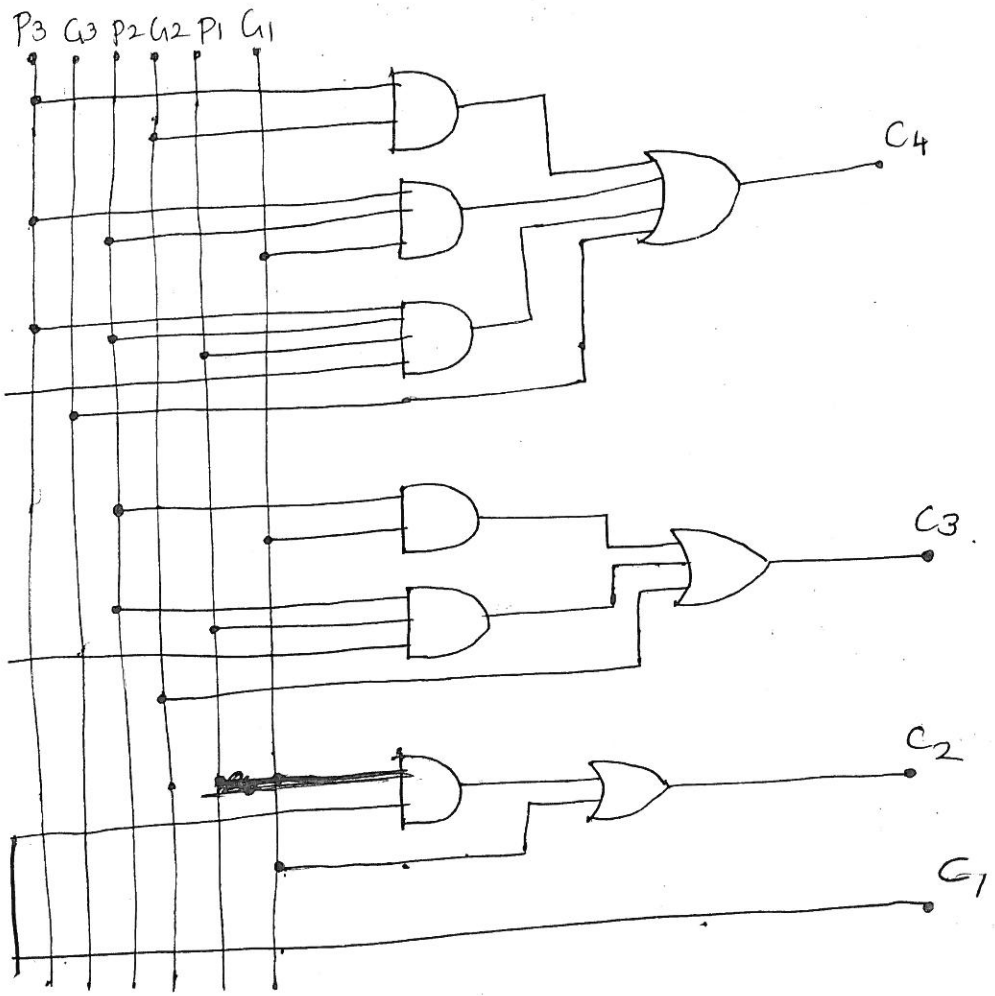
$$C_2 = G_1 + P_1 C_1$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 C_1) = G_2 + P_2 G_1 + P_2 P_1 C_1$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 C_1)$$

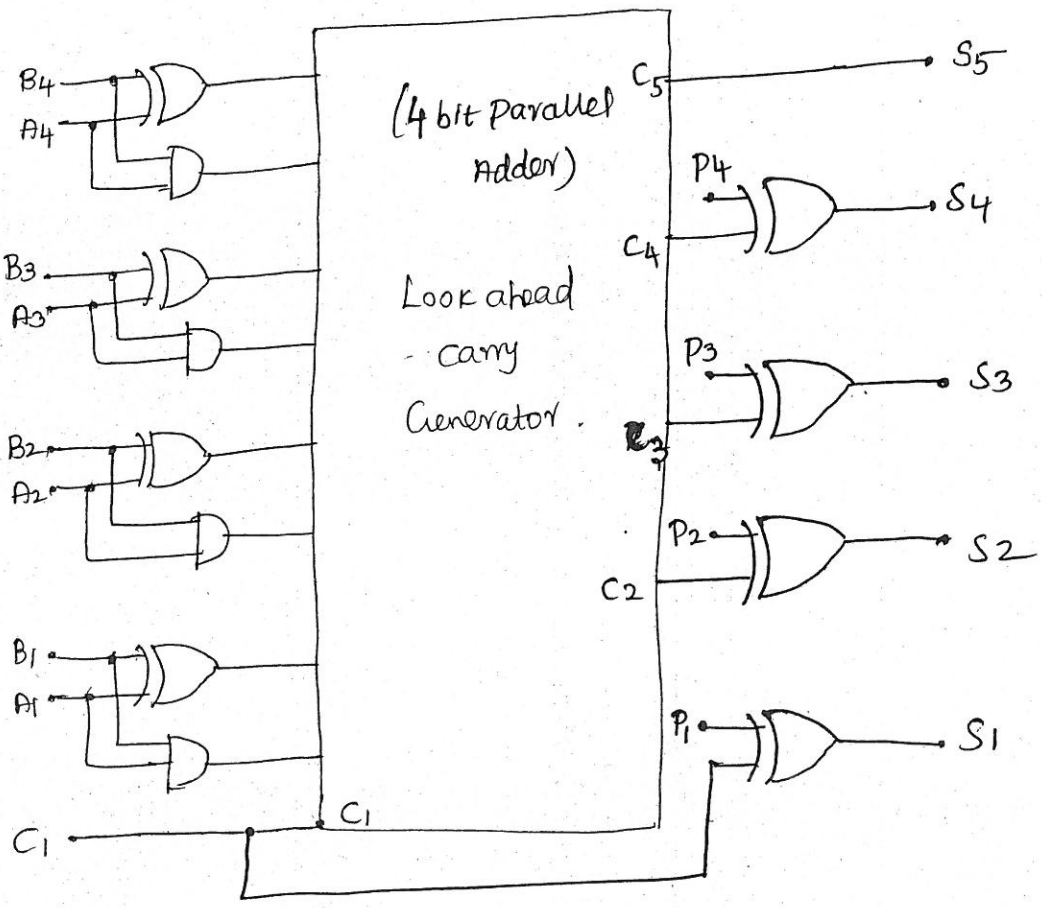
$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1 .$$

Logic Diagram:



→ From the above boolean Expression seen that C_4 does not have to wait for C_3 and C_2 to propagate; In fact C_4 is propagated at same time as C_2 and C_3.

4 bit parallel adder with look ahead carry generater.



Quine - McCluskey (or) Tabulation method

→ The minterms whose binary equivalent differ only in one place can be combined to reduce the minterms.

Forexample: $\bar{A}B\bar{C}D$, $\bar{A}B\bar{C}\bar{D} \Rightarrow 0000 + 0010 \Rightarrow$ Resultant term 00_0

implicant

→ The terms did not match during the procass called → prime implicants

Steps

(i). List all minterms in the binary form.

(ii). Arrange the minterms according to number of 1s.

(iii) Compare each binary no. with every term in adjacent next higher catag & if they differ only by one position.

(iv) Continue the procass until a single pass-through cycle yields no further elimination of variables.

(v) List all the prime implicants.

vi) Select the min. number of P.I's which must cover all the minterms

1). Find the minimal SOP for the boolean expression,

$f = \Sigma m(1,2,3,7,8,9,10,11,14,15)$ SOP using Quine mccluskey metho

TM Pos

$AB\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + AB\bar{C}\bar{D} + \bar{A}B\bar{C}D$

1111 0000 1110 0010

4 (0)

Step-1 List all the minterms in the binary form.

Minterms	Variables A	B	C	D
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
14	1	1	1	0
15	1	1	1	1

Step-2 Arrange the minterms according to the number of 1's.

Number of 1's	Minterms	Variables A	B	C	D
1	1	0	0	0	1
	2	0	0	1	0
	8	1	0	0	0
2	3	0	0	1	1
	9	1	0	0	1
	10	1	0	1	0
3	7	0	1	1	1
	11	1	0	1	1
	14	1	1	1	0
4	15	1	1	1	1

Step3:- 2 cell combination: Compare each binary no with every term in the adjacent next higher category & If they differ only by one position.

Combinations	Variables A	B	C	D
(1,3) →	0	0	–	1
(1,9) →	–	0	0	1
(2,3) →	0	0	1	–
(2,10) →	–	0	1	0
(8,9) →	1	0	0	–
(8,10) →	1	0	–	0
(3,7) →	0	–	1	1
(3,11) →	–	0	1	1
(9,11) →	1	0	–	1
(10,11) →	1	0	1	–
(10,14) →	1	–	1	0
(7,15) →	–	1	1	1
(11,15) →	1	–	1	1
(11,15) →	1	1	1	–

Step 4: (4 cell combination)

Combinations	Variables A	B	C	D
(1,3,9,11) →	–	0	–	1
(1,9,3,11) →	–	0	–	1
(2,3,10,11) →	–	0	1	–
(2,10,3,11) →	–	0	1	–
(8,9,10,11) →	1	0	–	–
(8,10,9,11) →	1	0	–	–
(3,7,11,15) →	–	–	1	1
(3,11,7,15) →	–	–	1	1
(10,11,14,15) →	1	–	1	–
(10,14,11,15) →	1	–	1	–

Step5:- (Prime Implicants)

P.I	Minterms									
	1	2	3	7	8	9	10	11	14	15
(1,3,9,11)	(X)		X			X		X		
(2,3,10,11)		(X)	X				X	X		
(8,9,10,11)					(X)	X	X	X		
(3,7,11,15)			X	(X)				X		X
(10,11,14,15)							X	X	(X)	X

$$Y = B\bar{D} + \bar{B}C + A\bar{B} + AC + CD$$

Simplify the boolean function by using a Quine-McCluskey method.

$F(A,B,C,D) = \Sigma m(0,2,3,6,7,8,10,12,13)$

Step1 Arrange the minterms in binary form.

Minterms	Variables A B C D
0 →	0 0 0 0
2 →	0 0 1 0
3 →	0 0 1 1
6 →	0 1 1 0
7 →	0 1 1 1
8 →	1 0 0 0
10 →	1 0 1 0
12 →	1 1 0 0
13 →	1 1 0 1

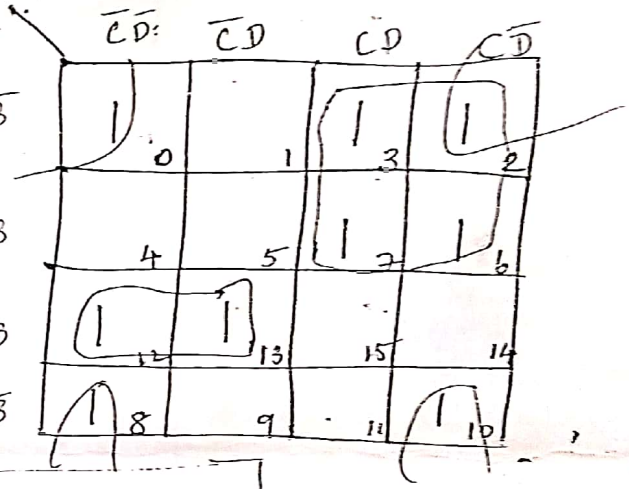
Step2:- Arrange minterms according to no. of 1's.

No. of 1's.	Minterms	Variables A B C D
0	0 →	0 0 0 0
1	2 →	0 0 1 0
	8 →	1 0 0 0
2	3 →	0 0 1 1
	6 →	0 1 1 0
	10 →	1 0 1 0
	12 →	1 1 0 0
3	7 →	0 1 1 1
	13 →	1 1 0 1

Prime Nos.	Variables A	B	C	D
(0,8) →	–	0	0	0
(0,2) →	0	0	–	0
(2,3) →	0	0	1	–
(2,6) →	0	–	1	0
(2,10) →	–	0	1	0
(8,10) →	1	0	–	0
(8,12) →	1	–	0	0
(3,7) →	0	–	1	1
(6,7) →	0	1	1	–
(12,13) →	1	1	0	–

Combinations	Variables A	B	C	D
(0,8,2,10) →	–	0	–	0
(0,2,8,10) →	–	0	–	0
(2,3,6,7) →	0	–	1	–
(2,6,3,7) →	0	–	1	–

$Y = \bar{B}\bar{D} + \bar{A}C + ABC$



top 5: Prime Implicants :-

P.I	Minterms 0	2	3	6	7	8	10	12	13
(8,12) →						X		X	
(12,13) →								X	(X)
(0,8,2,10) →	(X)	X				X	(X)		
(2,3,6,7) →		X	(X)	(X)	(X)				

$$Y = AB\bar{C} + \bar{B}\bar{D} + \bar{A}C$$

$= \Sigma m(0,1,3,7,8,9,11,15)$ Simplify using Quine mccluskey method.

Step-1): Write Binary no for the minterms.

Minterms	Variables A B C D
0 →	0 0 0 0
1 →	0 0 0 1
3 →	0 0 1 1
7 →	0 1 1 1
8 →	1 0 0 0
9 →	1 0 0 1
11 →	1 0 1 1
15 →	1 1 1 1

Step-2) Arrange no. of 1's.

No. of 1's	Minterms	Variables A B C D
0	0 →	0 0 0 0
1	1 →	0 0 0 1
	8 →	1 0 0 0
2	3 →	0 0 1 1
	9 →	1 0 0 1
3	11 →	1 0 1 1
	7 →	0 1 1 1
4	15 →	1 1 1 1

Step3: 2 cell combination.

Combination	Variables A B C D
✓(0,8) →	− 0 0 0
✓(0,1) →	0 0 0 −
✓(1,9) →	− 0 0 1
✓(1,3) →	0 0 − 1
✓(8,9) →	1 0 0 −
✓(3,11) →	− 0 1 1
✓(3,7) →	0 − 1 1
✓(9,11) →	1 0 − 1
✓(11,15) →	1 − 1 1
✓(7,15) →	− 1 1 1

Step4:- 4 cell combination

Combination	Variables A B C D
(0,1,8,9) →	− 0 0 −
(0,8,1,9) →	− 0 0 −
(1,3,9,11) →	− 0 − 1
(1,9,3,11) →	− 0 − 1
(3,7,11,15) →	− − 1 1
(3,11,7,15) →	− − 1 1

1) $f = \Sigma m(0,2,4,6,7,8,10,11,12,13,14,16,18,19,29,30)$.

2) $f = \Sigma m(1,3,4,5,9,11) + \Sigma d(6,8)$.

Step 1

Minterms	Variables A B C D.
1 →	0 0 0 1
3 →	0 0 1 1
4 →	0 1 0 0
5 →	0 1 0 1
9 →	1 0 0 1
10 →	1 0 1 0
11 →	1 0 1 1
6 →	0 1 1 0
8 →	1 0 0 0
10 →	1 0 1 0

Step 2.

No. of 1's	Minterms	Variables A B C D.
1	1 →	0 0 0 1
	4 →	0 1 0 0
	8 →	1 0 0 0
2	3 →	0 0 1 1
	5 →	0 1 0 1
	6 →	0 1 1 0
	9 →	1 0 0 1
	10 →	1 0 1 0
3	11 →	1 0 1 1

Step. 3 2 cell combination :-

Combination	Variables A B C D.
(1,3) →	0 0 – 1
(1,5) →	0 – 0 1
(1,9) →	– 0 0 1
(4,5) →	0 1 0 –
(4,6) →	0 1 – 0
(8,9) →	1 0 0 –
(8,10) →	1 0 – 0
(3,11) →	– 0 1 1
(9,11) →	1 0 – 1
(10,11) →	1 0 1 –

PI1

PI2

PI3.

Step. 4 :-

Combination	Variables A B C D.
(1,3,9,11) →	– 0 – 1
(8,9,10,11) →	1 0 – –

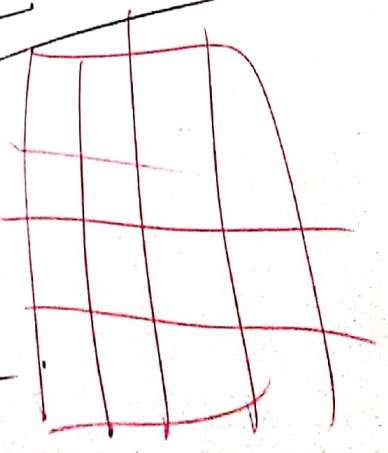
(b,d)

$Y = BD + A\bar{B}\bar{C} + \bar{A}BC$

$\bar{A}BC$

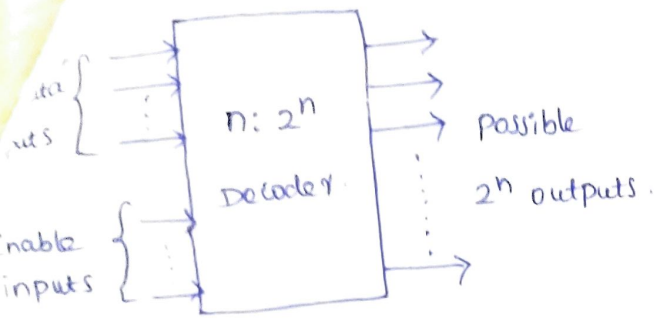
$+ BD$

$+ A\bar{B}$

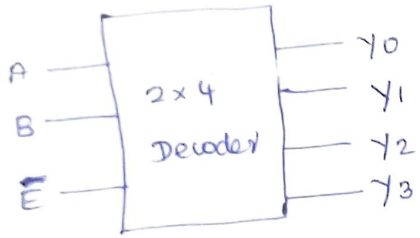


Decoders

Decoder

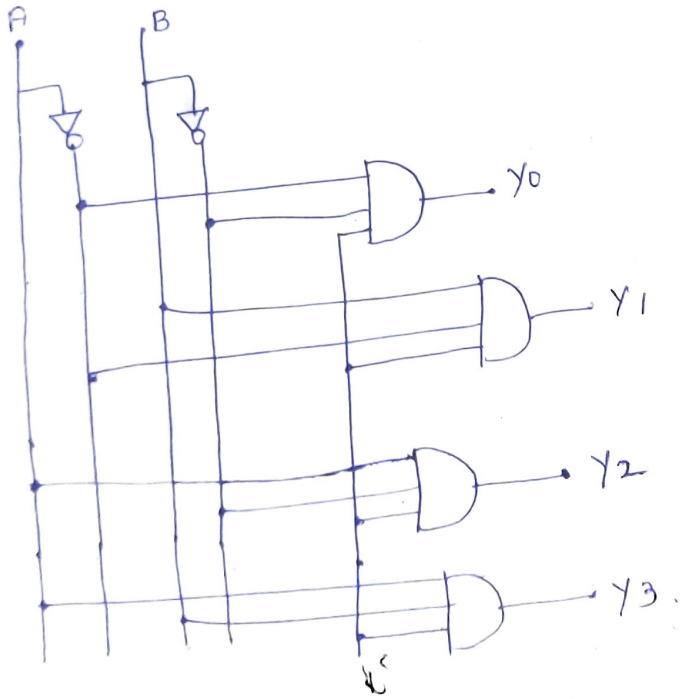


1) Design a 2 x 4 decoder.



Inputs			outputs			
EN	A	B	Y3	Y2	Y1	Y0
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

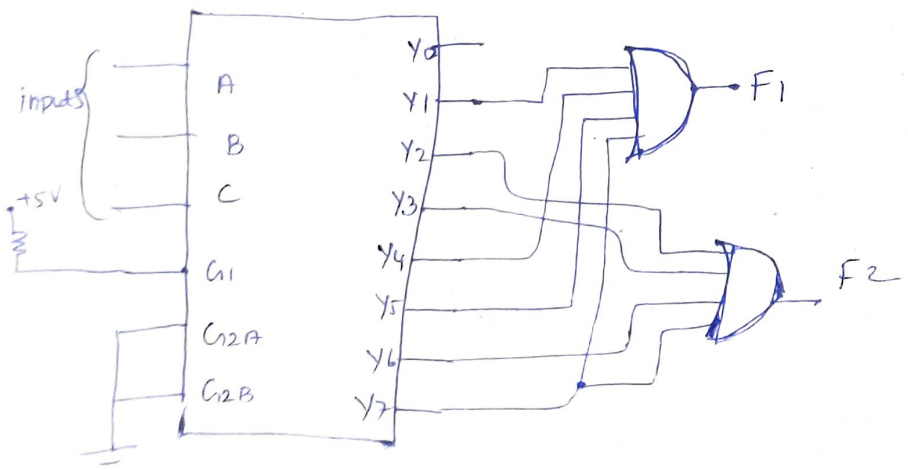
Logic diagram



2) Design 3-8 decoder

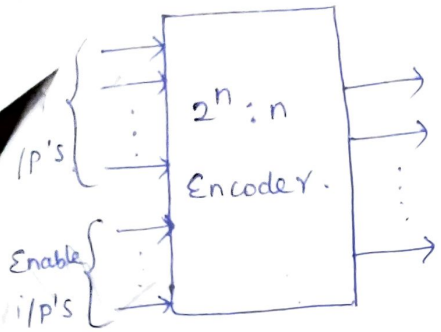
inputs				outputs							
EN	A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

1) Implement $F_1(A,B,C) = \Sigma m(1,4,5,7)$, $F_2(A,B,C) = \Pi M(2,3,6,7)$



ders

Encoder

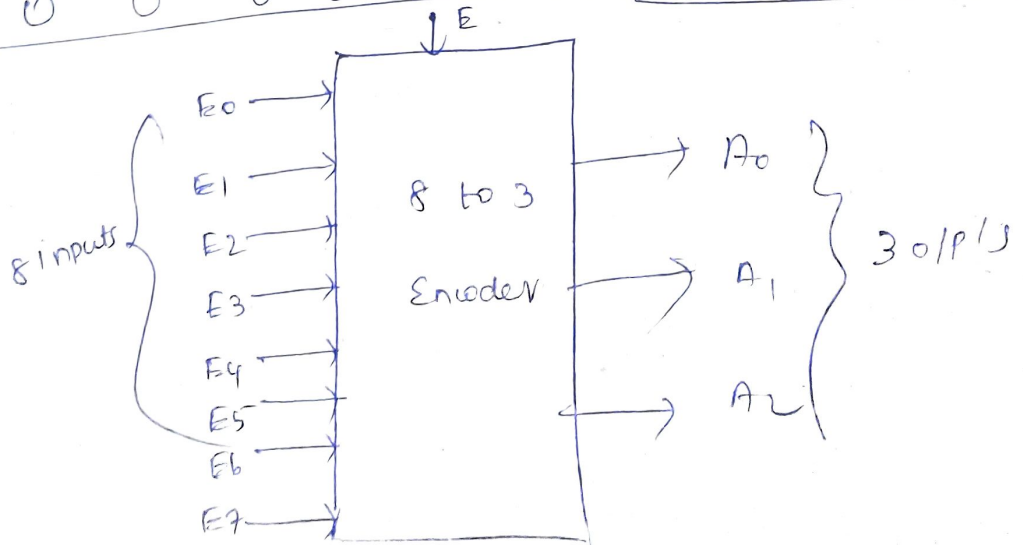


→ Inverse of Decoder.

→ 2^n inputs → n output lines

1) Octal to binary encoder. (8 to 3 Encoder)

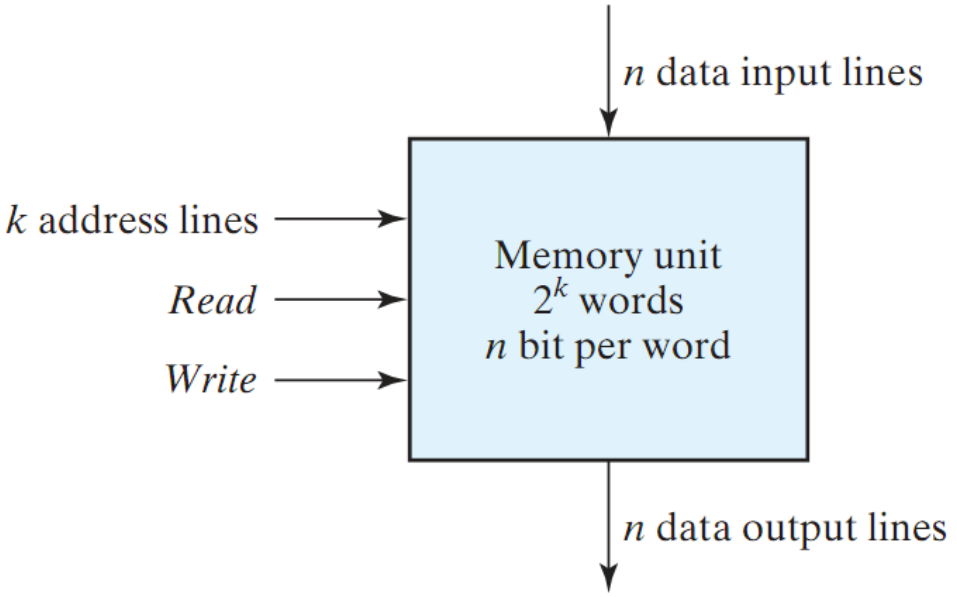
Inputs								outputs		
E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	A_0	A_1	A_2
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1



Memory

- **Memory:** A collection of cells capable of storing binary information (1s or 0s) – in addition to electronic circuit for storing (writing) and retrieving (reading) information.

- n data lines (input/output)
- k address lines
- 2^k words (data unit)
- Read/Write Control
- Memory size = 2^k X n



Memory (cont.)

Two Types of Memory:

- **Random Access Memory (RAM):**
  - Write/Read operations
  - **Volatile:** Data is lost when power is turned off
- **Read Only Memory (ROM):**
  - Read operation (no write)
  - **Non-Volatile:** Data is permanent.
  - PROM is programmable (allow special write)

Read-Only Memory (ROM)

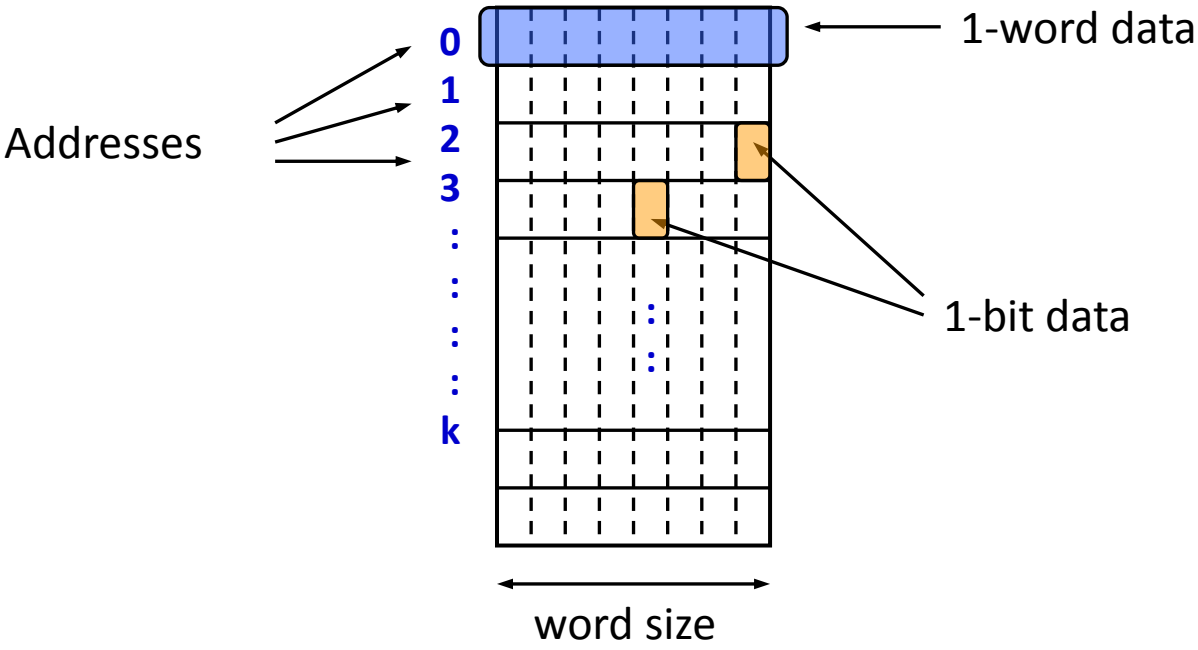
- **ROM:** A device in which "permanent" binary information is stored using a special device (programmer)

- k inputs (address) → 2^k words each of size n bits (data)
- ROM DOES NOT have a write operation ➔ ROM DOES NOT have data inputs

Word: group of bits stored in one location

Read-Only Memory (ROM)

- A semi-conductor memory is a device where data can be stored and retrieved.
- Logically, this memory device can be regarded as a table of memory cells (data).



Example 1

- **Example**: Design a combinational circuit using ROM (memory size). The circuit accepts a 3-bit number and generates an output binary number equal to the square of the number.

  Solution: Derive truth table:

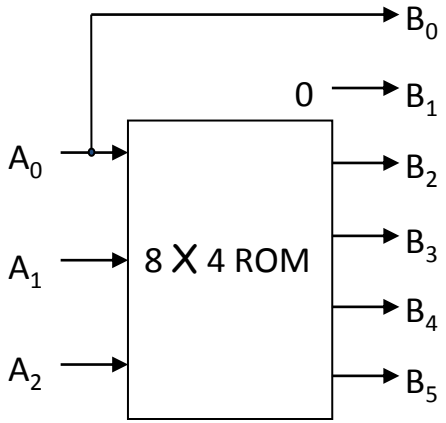
Inputs			Outputs						
A2	A1	A0	B5	B4	B3	B2	B1	B0	SQ
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

Example 1 (cont.)

Inputs			Outputs						
A2	A1	A0	B5	B4	B3	B2	B1	B0	SQ
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

ROM truth table – specifies the required connections

- B1 is ALWAYS 0 ➔ no need to generate it using the ROM
- B0 is equal to A0 ➔ no need to generate it using the ROM
- Therefore: The minimum size of ROM needed is 2^3X4 or 8X4



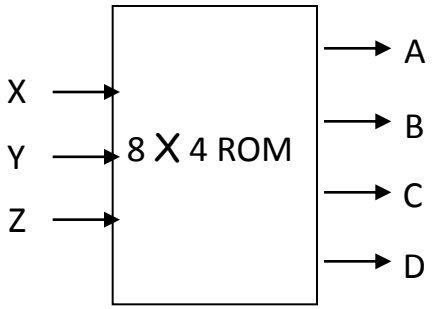
Example 2

- **Problem**: Tabulate the truth for an 8 X 4 ROM that implements the following four Boolean functions:

$A(X,Y,Z) = \Sigma m(3,6,7)$; $B(X,Y,Z) = \Sigma m(0,1,4,5,6)$

$C(X,Y,Z) = \Sigma m(2,3,4)$; $D(X,Y,Z) = \Sigma m(2,3,4,7)$

Solution:

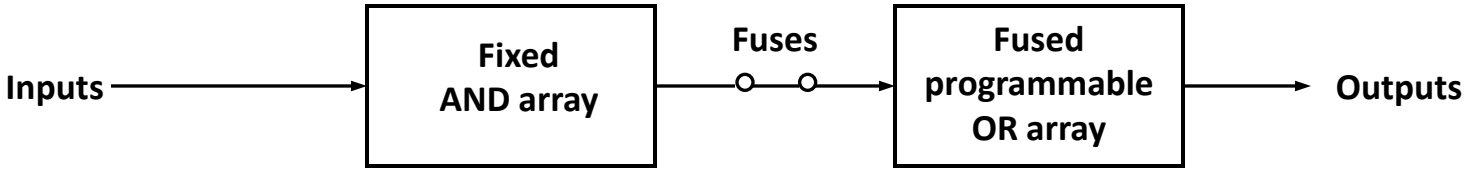


Inputs			Outputs			
X	Y	Z	A	B	C	D
0	0	0	0	1	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	0	0
1	1	0	1	1	0	0
1	1	1	1	0	0	1

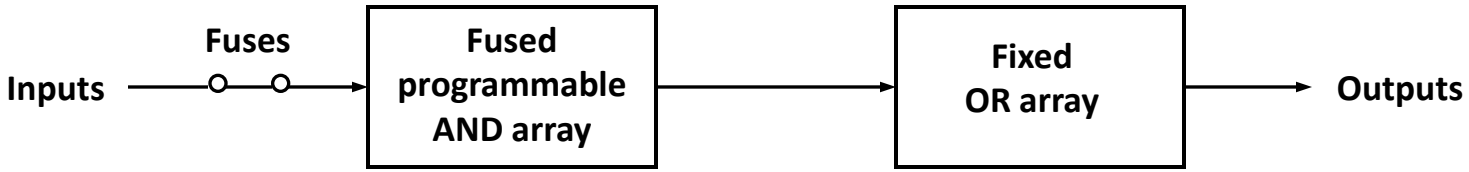
Types of ROMs

- A ROM programmed in four different ways:
- **ROM: Mask Programming**
  - By a semiconductor company
- **PROM (Programmable ROM)**
  - User can connect fuses with a special programming device (PROM programmer)
  - Only programmed once
- **EPROM (Erasable PROM)**
  - Can be erased using Ultraviolet Light
- **Electrically Erasable PROM (EEPROM or E^2PROM)**
  - Like an EPROM, but erased with electrical signal

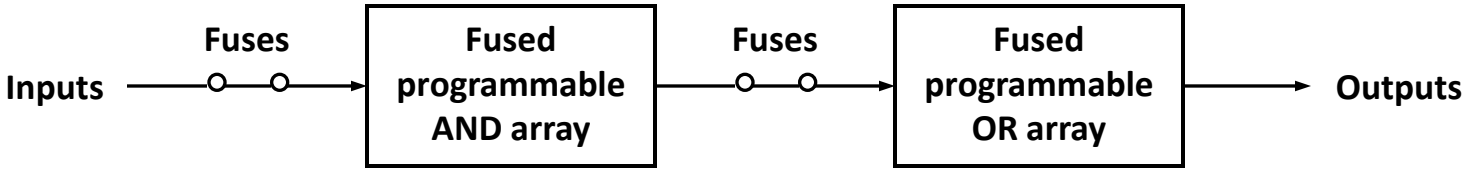
Programmable Logic Devices



Programmable Read Only Memory (PROM)



Programmable Array Logic (PAL)



Programmable Logic Array (PLA)

Programmable Logic Array (PLA)

- Combination of a **programmable AND** array followed by a **programmable OR** array.
- Example: Design a PLA to realise the following three logic functions and show the internal connections.

$f_1(A,B,C,D,E) = A'.B'.D' + B'.C.D' + A'.B.C.D.E'$

$f_2(A,B,C,D,E) = A'.B.E + B'.C.D'.E$

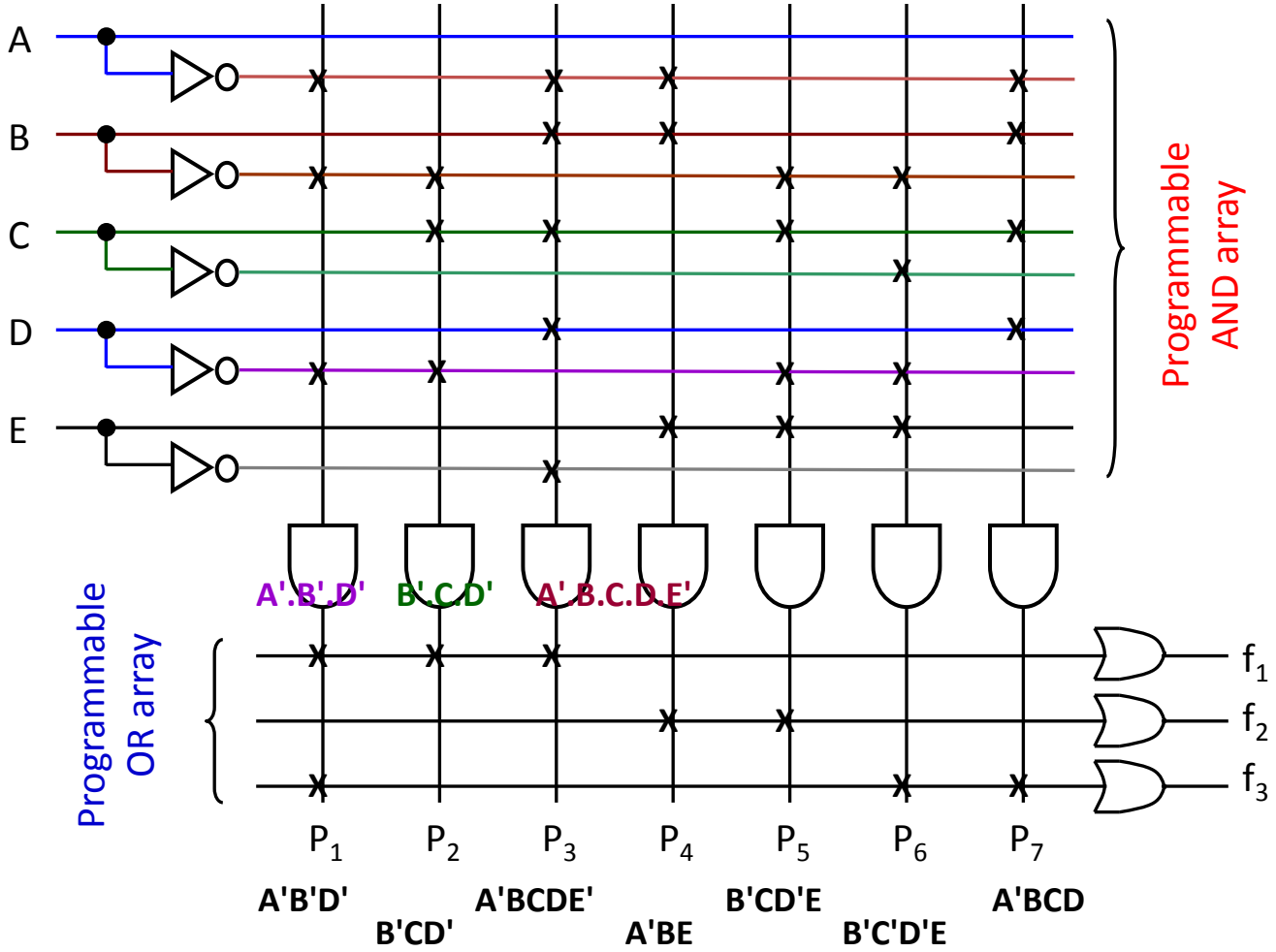
$f_3(A,B,C,D,E) = A'.B'.D' + B'.C'.D'.E + A'.B.C.D$

Realising Logic Functions with PLAs

$f_1(A,B,C,D,E) = A'.B'.D' + B'.C.D' + A'.B.C.D.E'$

$f_2(A,B,C,D,E) = A'.B.E + B'.C.D'.E$

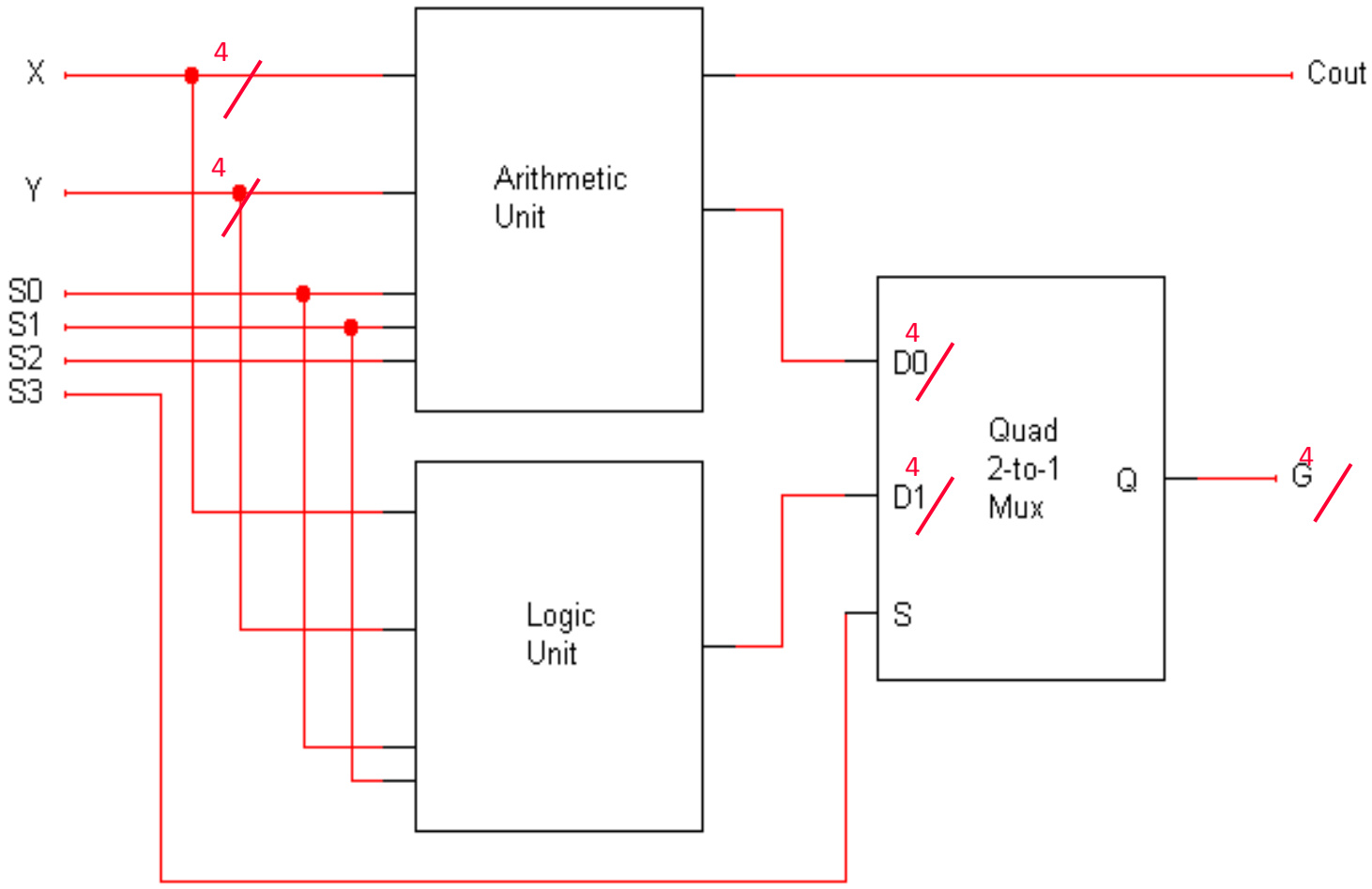
$f_3(A,B,C,D,E) = A'.B'.D' + B'.C'.D'.E + A'.B.C.D$



Arithmetic Logic Unit

- An arithmetic-logic unit, or ALU, performs many different arithmetic and logic operations. The ALU is the “heart” of a processor—you could say that everything else in the CPU is there to support the ALU.
- Here’s the plan:
  - We’ll show an arithmetic unit first, by building off ideas from the adder - subtractor circuit.
  - Then we’ll talk about logic operations a bit, and build a logic unit.
  - Finally, we put these pieces together using multiplexers.
- We use some examples from the textbook, but things are re-labeled and treated a little differently.

A complete ALU circuit



What does HDL stand for?

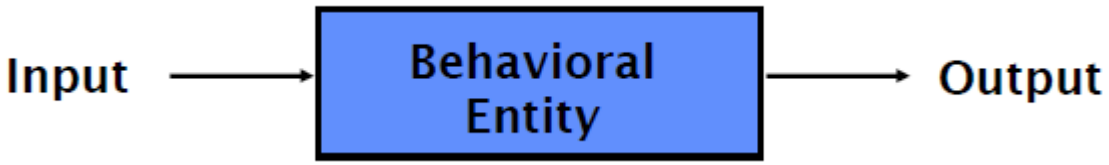
HDL is short for **Hardware Description Language**

VHDL – VHSIC (Very High Speed Integrated Circuit)

Hardware Description Language

Basic Form of VHDL Code

- Every VHDL design description consists of at least **one entity / architecture pair, or one entity with multiple architectures.**
- The entity section is used to declare I/O ports of the circuit. The architecture portion describes the circuit's behavior.
- A behavioral model is similar to a "black box".
- Standardized design libraries are included before entity declaration.



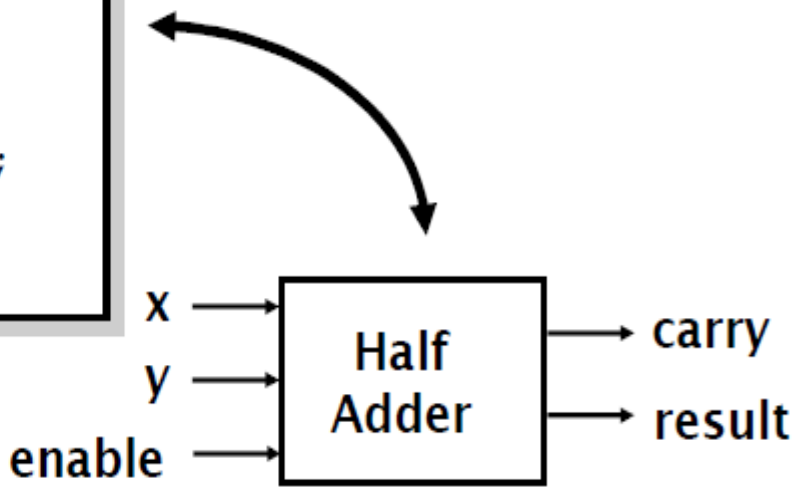
Standard Libraries

- Include *library ieee;* before entity declaration.
- **ieee.std_logic_1164** defines a standard for designers to use in describing interconnection data types used in VHDL modeling.
- **ieee.std_logic_arith** provides a set of arithmetic, conversion, comparison functions for signed, unsigned, std_ulogic, std_logic, std_logic_vector.
- **ieee.std_logic_unsigned** provides a set of unsigned arithmetic, conversion, and comparison functions for std_logic_vector.

Entity Declaration

- An entity declaration describes the interface of the component. Avoid using Altera's primitive names
- PORT clause indicates input and output ports.
- An entity can be thought of as a symbol for a component.

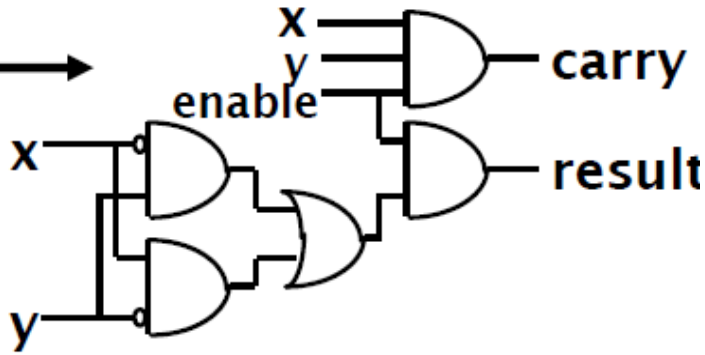
```
ENTITY half_adder IS

    PORT( x, y, enable: IN bit;
          carry, result: OUT bit);

END half_adder;
```



Architecture Declaration

- Architecture declarations describe the operation of the component.
- Many architectures may exist for one entity, but only one may be active at a time.
- An architecture is similar to a schematic of the component.

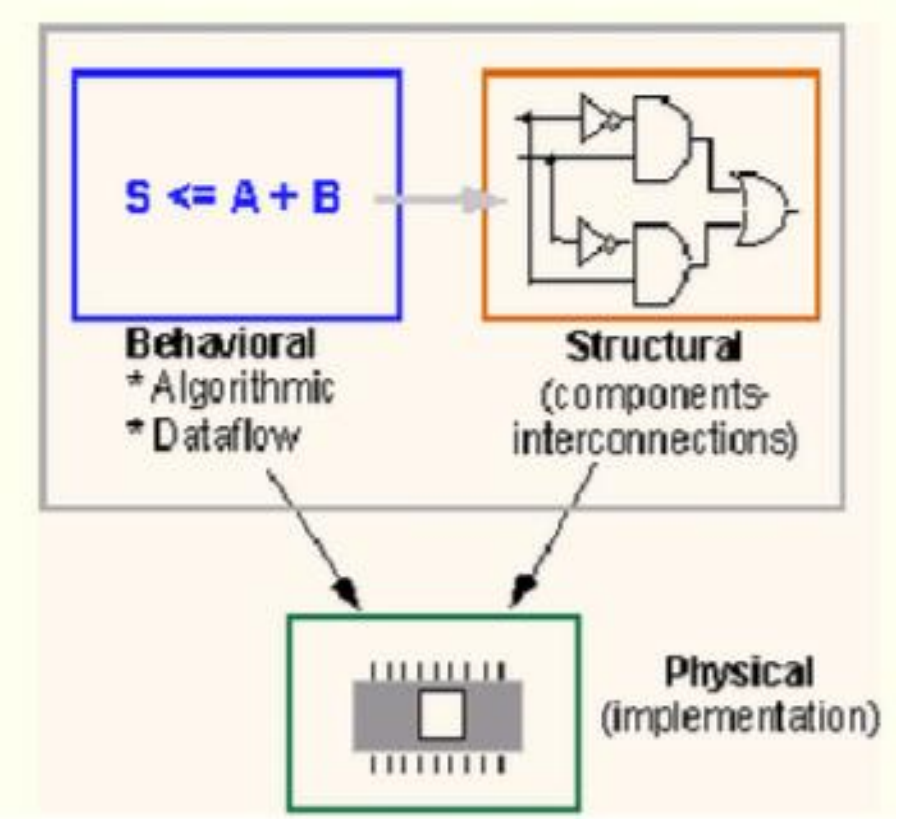
```
ARCHITECTURE behave OF half_adder IS
BEGIN
    PROCESS (enable, x, y)
    BEGIN
      IF (enable = '1') THEN
        result <= x XOR y;
        carry  <= x AND y;
      ELSE
        carry  <=  '0';
        result <=  '0';
      END IF;
    END PROCESS;
END behave;
```



Modeling Styles

There are three modeling styles:

- Behavioral (Sequential)
- Data flow
- Structural



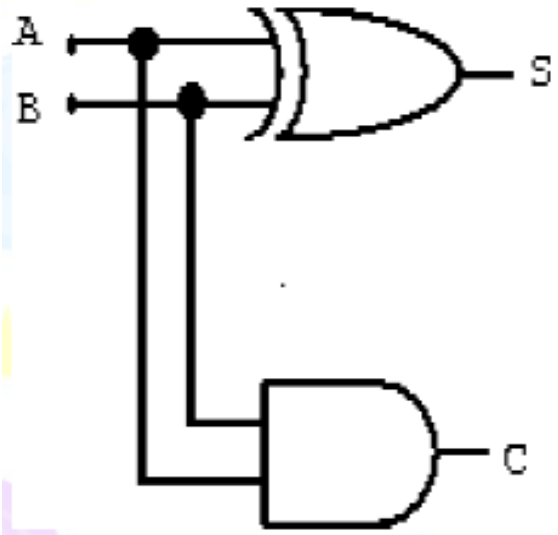
Styles in VHDL

- **Behavioral**
  - High level, algorithmic, sequential execution
  - Hard to synthesize well
  - Easy to write and understand (like high-level language code)
- **Dataflow**
  - Medium level, register-to-register transfers, concurrent execution
  - Easy to synthesize well
  - Harder to write and understand (like assembly code)
- **Structural**
  - Low level, netlist, component instantiations and wiring
  - Trivial to synthesize
  - Hardest to write and understand (very detailed and low level)

Dataflow modeling

❖ The internal working of an entity can be implemented using concurrent signal assignment.

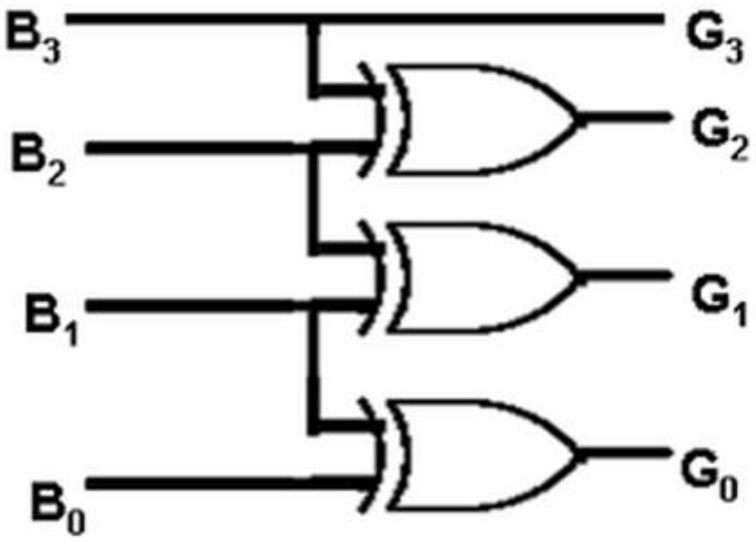
Half adder example




```
Library IEEE;
IEEE.STD_LOGIC_1164.all;
entity ha_en is
      port (A,B:in bit;S,C:out bit);
end ha_en;
architecture dataflow of ha_en is
         begin
             S<=A xor B;
             C<=A and B;
end ha_ar;
```

DATAFLOW MODELLING

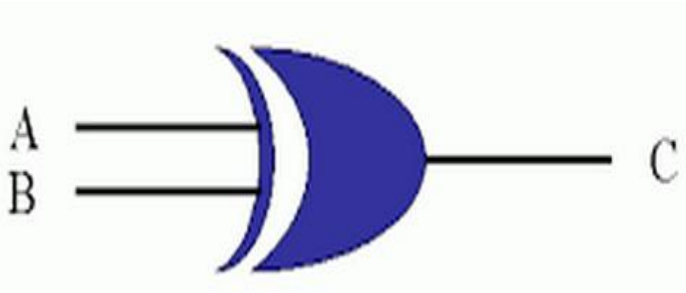
BINARY TO GRAY CODE CONVERTOR



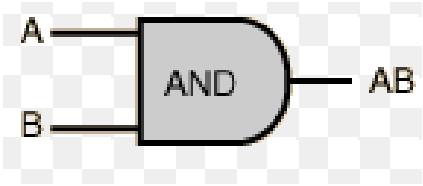

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity bintogray is
    Port ( b3 : in  STD_LOGIC;
           b2 : in  STD_LOGIC;
           b1 : in  STD_LOGIC;
           b0 : in  STD_LOGIC;
           g3 : out  STD_LOGIC;
           g2 : out  STD_LOGIC;
           g1 : out  STD_LOGIC;
           g0 : out  STD_LOGIC);
end bintogray;
architecture dataflow of bintogray is
begin
g3<=(b3);
g2<=(b3 xor b2);
g1<=( b2 xor b1);
g0<=(b1 xor b0 );
end dataflow;
```

xor gate

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity xorgate is
    Port ( a1 : in  STD_LOGIC;
           b1 : in  STD_LOGIC;
           c1 : out  STD_LOGIC);
end xorgate;
architecture dataflow of xorgate is
begin
c1<=(a1 and (not b1))or(b1 and(not a1));
end dataflow;
```

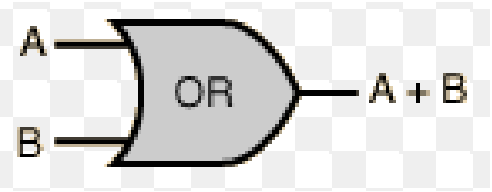


AND gate



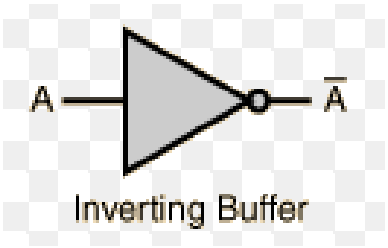

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity andgate is
    Port ( a2 : in  STD_LOGIC;
           b2 : in  STD_LOGIC;
           c2 : out  STD_LOGIC);
end andgate;
architecture dataflow of andgate is
begin
c2<= a2 and b2;
end dataflow;
```

OR gate



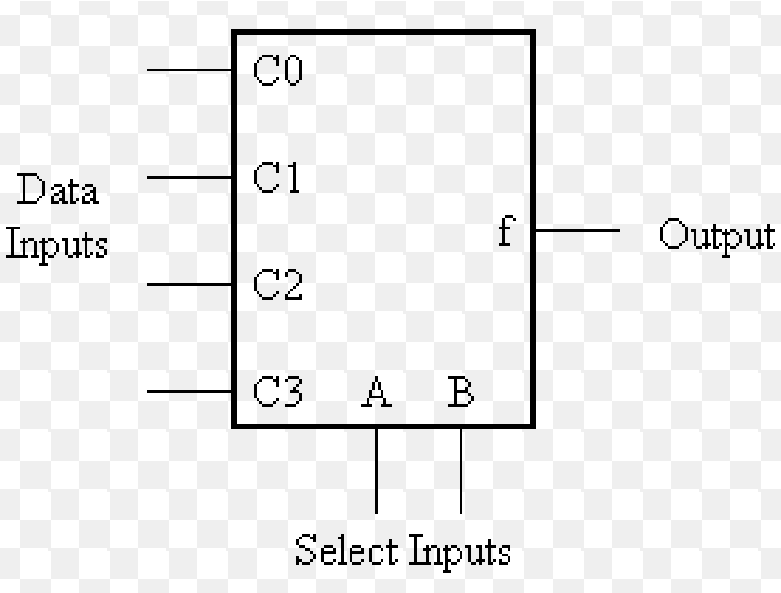

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity orgate is
   Port ( a1 : in  STD_LOGIC;
          b1 : in  STD_LOGIC;
          c1 : out  STD_LOGIC);
end orgate;
architecture Dataflow of orgate is
begin
c1<=( a1 or b1);
end Dataflow;
```

NOT gate




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity notgate is
    Port ( a2 : in  STD_LOGIC;
           c2 : out  STD_LOGIC);
end notgate;
architecture Dataflow of notgate is
begin
c2<=( not a2);
end Dataflow;
```

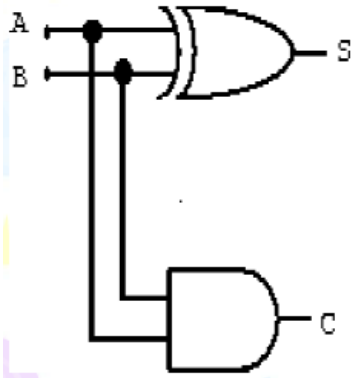
Multiplexer (8*1 MUX)




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mux8_1 is
   Port ( i0 : in  STD_LOGIC;
          i1 : in  STD_LOGIC;
          i2 : in  STD_LOGIC;
          i3 : in  STD_LOGIC;
          i4 : in  STD_LOGIC;
          i5 : in  STD_LOGIC;
          i6 : in  STD_LOGIC;
          i7 : in  STD_LOGIC;
          s2 : in  STD_LOGIC;
          s1 : in  STD_LOGIC;
          s0 : in  STD_LOGIC;
          y : out  STD_LOGIC);
end mux8_1;
architecture dataflow of mux8_1 is
begin
y<=(((not s2)and (not s1) and (not s0) and i0) or((not s2)and (not s1)
and ( s0) and (i1))  or ((not s2)and ( s1) and (not  s0) and (i2)) or
((not s2)and ( s1) and ( s0) and (i3)) or ((s2)and (not s1) and ( s0)
and (i4)) or (( s2)and (not s1) and ( s0) and (i5)) or ((not s2)and ( s1)
and (not  s0) and (i6)) or (( s2)and ( s1) and ( s0) and (i7)));
end dataflow;
```

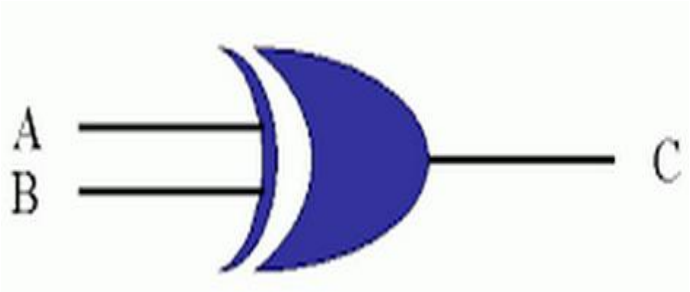
Behavioural modeling

- The internal working of an entity can be implemented using set of statements. It contains:
  - **Process statements**
  - **Sequential statements**
  - **Signal assignment statements**
  - **Wait statements**



BEHAVIORAL MODELLING

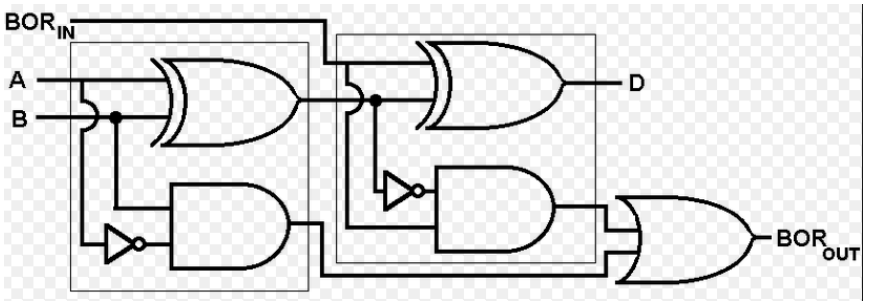
XOR GATE




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity xorgate is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : out  STD_LOGIC);
end xorgate;
architecture Behav of xorgate is
begin
process(a,b)
begin
if(a='1'and b='0')then
c<='1';
else if(a='0'and b='1')then
c<='1';
else
c<='0';
end if;
end if;
end process;
end xora;
```

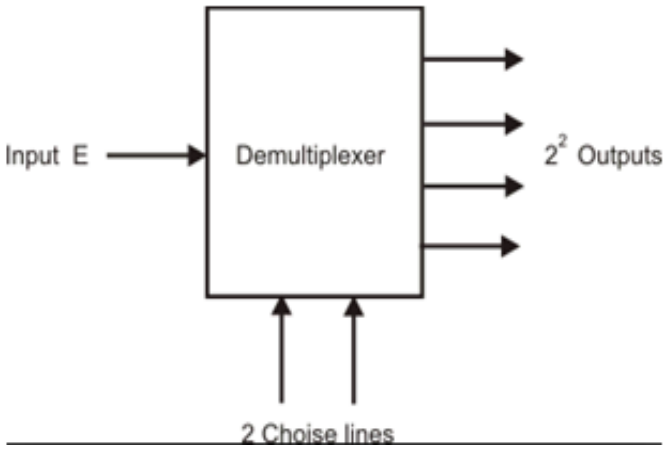
FULL SUBSTRACTOR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity fullsubst is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           bi : in  STD_LOGIC;
           d : out  STD_LOGIC;
           bo : out  STD_LOGIC);
end fullsubst;
architecture Behavioral of fullsubst is
begin
process(a,b,bi)
begin
if((a='0' and b='0' and bi='1')or(a='0' and b='1' and bi='0') or (a='1'
and b='1' and bi='1'))then
d<='1';
bo<='1';
else if(a='0' and b='1' and bi='1')then
d<='0';
bo<='1';
else if(a='1' and b='0' and bi='0')then
d<='1';
bo<='0';
else
d<='0';
bo<='0';
end if;
end if;
end if;
end process;
end Behavioral;
```



DEMULTIPLEXER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity decoder  is
   Port ( s : in  STD_LOGIC_VECTOR (2 downto 0);
          d : out  STD_LOGIC_VECTOR (7 downto 0));
end decoder;
architecture Behavioral of decoder is
begin
process(s)
begin
case s is
when "000"=>
d<="00000001";
when "001"=>
d<="00000010";
```




```
when "010"=>
d<="00000100";
when "011"=>
d<="00001000";
when "100"=>
d<="00010000";
when "101"=>
d<="00100000";
when "110"=>
d<="01000000";
when "111"=>
d<="10000000";
when others=>
null;
end case;
end process;
end Behavioral;
```

Structural modeling

- The implementation of an entity is done through set of interconnected components. It contains:
  - Signal declaration.
  - Component instances
  - Port maps.
  - Wait statements.

Component declaration:

Syntax:

Component component_name [is]

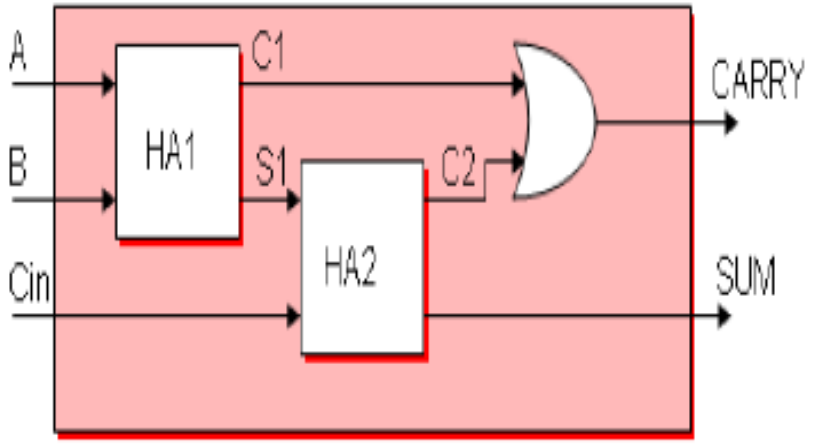
List_of_interface ports;

End component component_name;

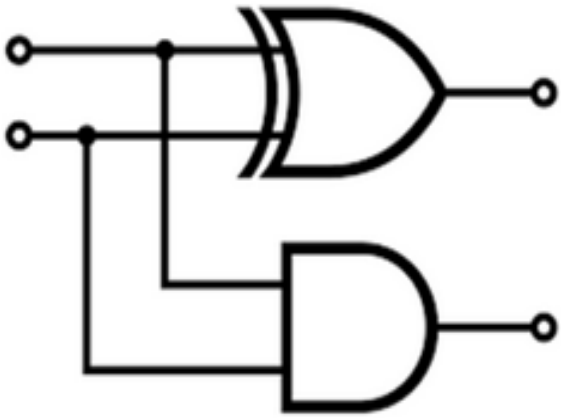
- Before instantiating the component it should be declared Using component declaration as shown above. Component Declaration declares the name of the entity and interface of a component.

Full adder example

```
library IEEE
use IEEE.STD_LOGIC_1164.all;
entity fa_en is
    port(A,B,Cin:in bit; SUM,
CARRY:out bit);
end fa_en;
architecture  struct   of fa_en is
component ha_en
    port(A,B:in bit;S,C:out bit);
end component;
    signal C1,C2,S1:bit;
    begin
        HA1:ha_en port
map(A,B,S1,C1);
        HA2:ha_en port
map(S1,Cin,SUM,C2);
        CARRY <= C1 or C2;
end fa_ar;
```



Half adder



```
library IEEE;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity halfadder_str_us_xor is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           sum : out  STD_LOGIC;
           carry : out  STD_LOGIC);
end halfadder_str_us_xor;
architecture Structural of halfadder_str_us_xor is
component xorgate
Port ( a1 : in  STD_LOGIC;
           b1 : in  STD_LOGIC;
           c1 : out  STD_LOGIC);

end component;
component andgate
Port ( a2 : in  STD_LOGIC;
           b2 : in  STD_LOGIC;
           c2 : out  STD_LOGIC);
          end component;
begin
L0:xorgate port map(a,b,sum);
L1:andgate port map(a,b,carry);
end Structural;
```