Unit-5
Combinational Logic with MSI and LSI

Binary Adder

This circuit sums up two binary numbers A *and* B of n-bits using full-adders to add each bit pair and carry from previous bit position.

Binary Parallel Adder:

A binary parallel adder is a digital circuit that produces the arithmetic sum of two binary numbers in parallel. It consists of full adders connected in cascade, with the output carry from one full adder connected to the input carry of the next full adder. An n bit parallel adder requires n full adders.

4-bit binary parallel adder:

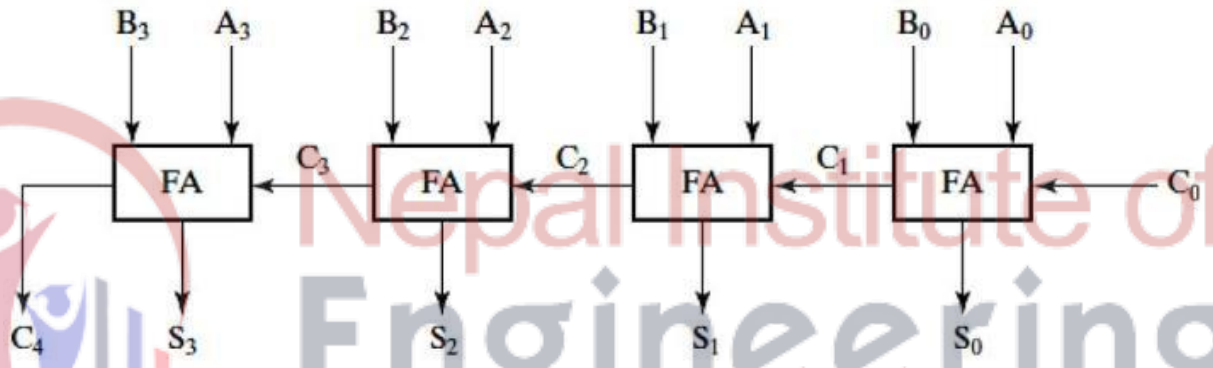


Fig: 4-bit binary parallel adder

A 4-bit binary parallel adder consists of 4-full adder. The augend bits are A_4, A_3, A_2, A_1 and addend bits are B_1, B_2, B_3, B_4. This parallel adder produces their sum as $C_4S_3S_2S_1S_0$ where C_4 is the final carry. The carries are connected in chain through the full-adders. The input carry to the first full adder is C_1 and the output carry from MSB position of full adder is C_4.

Q. Design a BCD-to-excess-3 code converter using a 4-bit full adders MSI circuit.

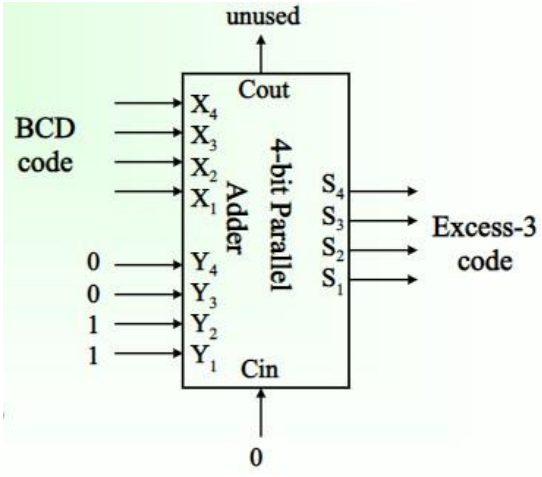
Sol^n:

$Excess - 3\ code = BCD\ code + (0011)_2$

Augend bits = $X_4X_3X_2X_1$ *(Input bits)*

Addend bits = $Y_4Y_3Y_2Y_1 = 0011$

Excess-3 code = $S_4S_3S_2S_1$ *(output)*



Decimal adder/BCD adder:

BCD adder is a combinational digital circuit that adds two BCD digits in parallel and produces sum which is also BCD.

- In BCD adder, each input digit does not exceed 9, so the output sum can't be greater than $9 + 9 + 1 = 19$, the 1 in the sum being an input carry.
- Suppose we apply two BCD digits to a 4-bit binary adder. The adder will form the sum in binary and produce a result which may range from 0 to 19.

Truth table for BCD adder is:

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

- In examining the content of the table, it is apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed.
- When the binary sum is greater than 1001, we obtain a non- valid BCD representation. The addition of binary 0110 (6 in decimal) to the binary sum converts it to the correct BCD representation and also produces an output carry.
- It is obvious from the table that a correction is needed when the binary sum has an output carry $k = 1$.
- The other six combination from 1010 to 1111 that need a correction have a 1 in position Z_8. To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_8, we specify further that either Z_4 or Z_2 must have 1.
- The condition for a correction and an output carry can be expressed by the Boolean function: $\boldsymbol{C = K + Z_8Z_4 + Z_8Z_2}$
- When output carry $C = 0$, nothing is added to the binary sum.

- When output carry $C = 1$, binary 0110 is added to the binary sum through the bottom 4-bit binary adder to convert the binary sum into BCD sum. (*In fig. below*)

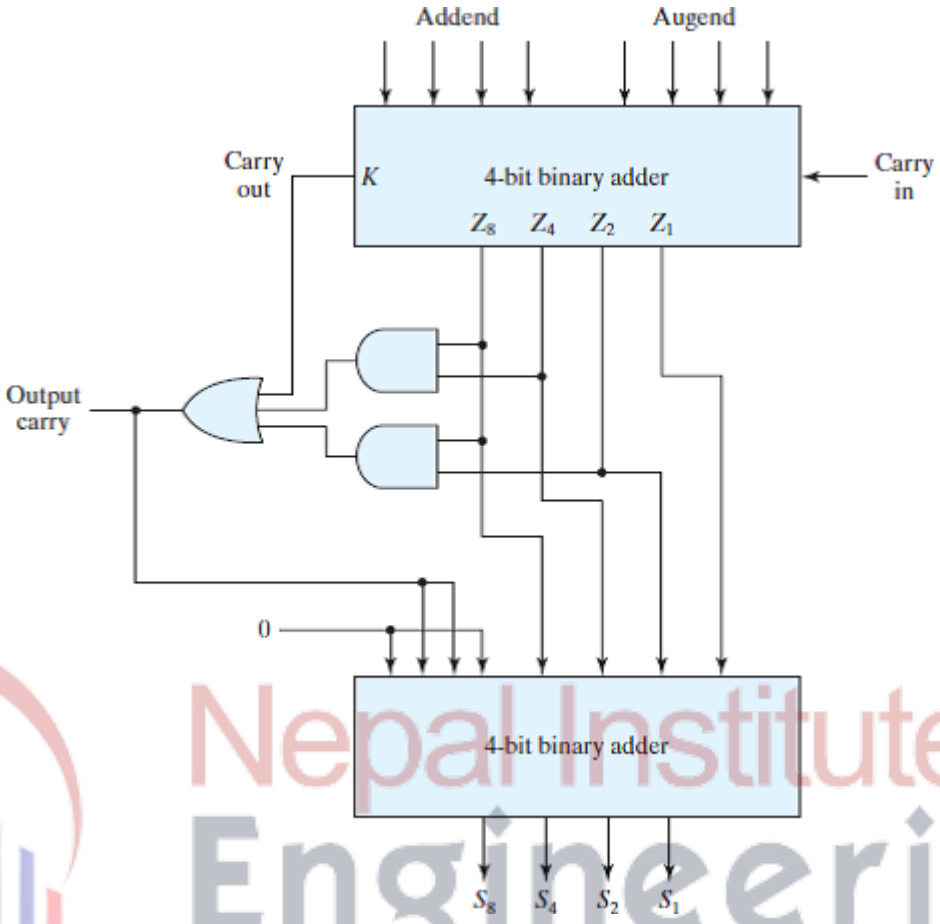


Fig: BCD adder

4-bit magnitude comparator:

4-bit magnitude comparator is a combinational logic circuit that compares two binary numbers each of 4-bits.

Consider two numbers A & B with four digits each.

$A = A_3A_2A_1A_0$

$B = B_3B_2B_1B_0$

Verification of $(\boldsymbol{A = B})$***:***

- The equality relation of each pair of bits can be expressed:

$$x_i = A_iB_i + \bar{A}_i\bar{B}_i, \ i = 0, 1, 2, 3$$

  Where $x_i = 1$ only if $A_i = B_i$ and $x_i = 0$ only if $A_i \neq B_i$.
- For equality condition to exist, all x_i variables must be equal to 1. A & B will be equal if $x_3x_2x_1x_0 = 1$.

$$\therefore (\boldsymbol{A = B}) = \boldsymbol{x_3x_2x_1x_0}$$

Verification of $(\boldsymbol{A > B})$***:***

- If $A_3 > B_3$ then $A > B$, it means $A_3 = 1$ & $B_3 = 0$. Therefore A is greater than B if $\boldsymbol{A_3\bar{B}_3 = 1}$.
- If $A_3 = B_3 (i.e\ x_3 = 1)$ and $A_2 > B_2$ then $A > B$. Therefore A is greater than B if $\boldsymbol{x_3A_2\bar{B}_2 = 1}$.
- If $A_3 = B_3 (i.e\ x_3 = 1)$ & $A_2 = B_2 (i.e\ x_2 = 1)$ and $A_1 > B_1$ then $A > B$. Therefore A is greater than B if $\boldsymbol{x_3x_2A_1\bar{B}_1 = 1}$.
- If $A_3 = B_3 (i.e\ x_3 = 1)$ & $A_2 = B_2 (i.e\ x_2 = 1)$ & $A_1 = B_1 (i.e\ x_1 = 1)$ and $A_0 > B_0$ then $A > B$. Therefore A is greater than B if $\boldsymbol{x_3x_2x_1A_0\bar{B}_0 = 1}$.

$$\therefore (\boldsymbol{A > B}) = \boldsymbol{A_3\bar{B}_3 + x_3A_2\bar{B}_2 + x_3x_2A_1\bar{B}_1 + x_3x_2x_1A_0\bar{B}_0}$$

In the same manner we can derive the expression for $(\boldsymbol{A < B})$.

$$\therefore (\boldsymbol{A > B}) = \boldsymbol{\bar{A}_3B_3 + x_3\bar{A}_2B_2 + x_3x_2\bar{A}_1B_1 + x_3x_2x_1\bar{A}_0B_0}$$

Logic Diagram:

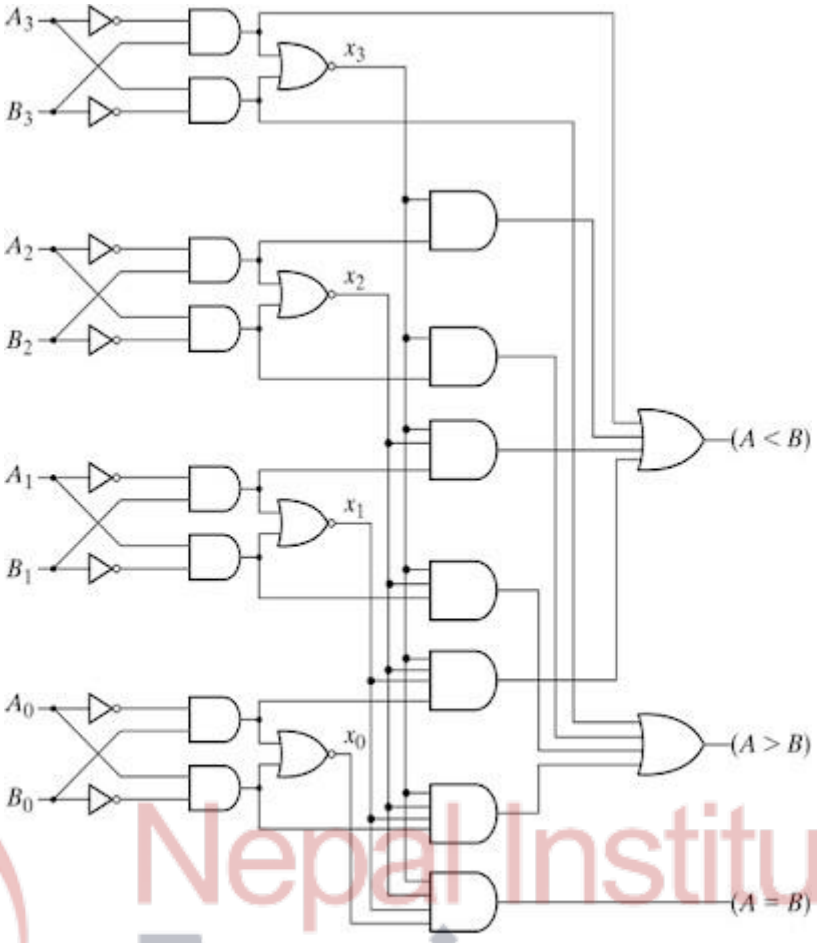


Fig. 4-17 4-Bit Magnitude Comparator

Q. Implement a full-adder circuit with a decoder and two OR gates.

Sol^n:

The truth table for full adder:

Input			Output	
A	B	Cin	Sum	Carry
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

From the truth table

$S(A, B, C_{in}) = \sum(1, 2, 4, 7)$

$C(A, B, C_{in}) = \sum(3, 5, 6, 7)$

Since there are three inputs and a total of eight minterms. So we need 3-to-8 line decoder. The decoder generates the eight minterms for A, B & C_{in}. The OR gate for output sum (S) forms the sum of minterms 1, 2, 4 & 7. The OR gate for the output carry (C) forms the sum of minterms 3, 5, 6 & 7.

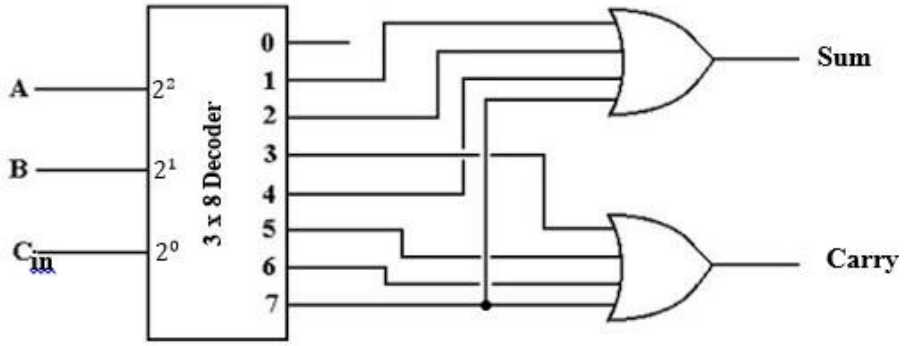


Fig: Full adder implementation with decoder

Q. Design a 3 to 8 line decoder using two 2 to 4 line decoder and explain it.
Solⁿ:

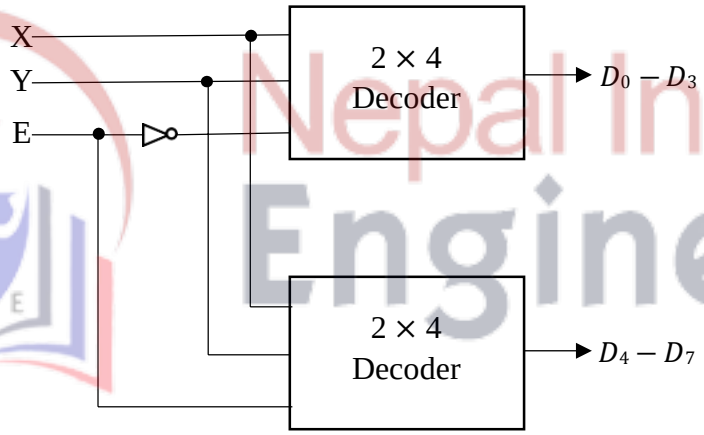
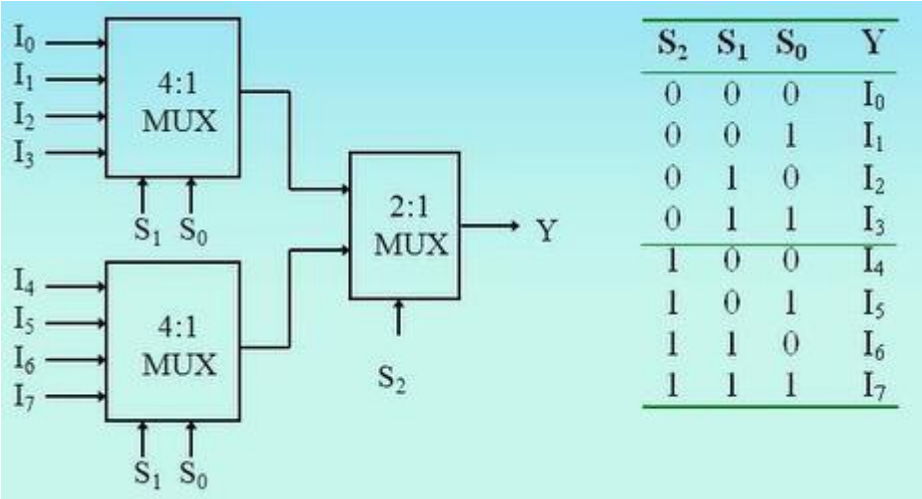


Fig: 3 to 8 decoder using two 2 to 4 decoder

The figure shows two 2 × 4 decoder with enable input (E) connected to form a 3 × 8 decoder. When $E = 0$, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's and the top four outputs generate minterms 000 to 001. When $E = 1$, the enable conditions are reversed. The bottom decoder outputs generate minterms 100 to 111 while the outputs of the top decoder are all 0's.

Q. Design a 8-to-1 line multiplexer using lower order multiplexers and explain it.

Soln:



S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

The same **selection lines, s_1 & s_0** are applied to both 4x1 Multiplexers. The data inputs of upper 4x1 Multiplexer are I_0 to I_3 and the data inputs of lower 4x1 Multiplexer are I_4 to I_7. Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, s_1 & s_0.

The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_2** is applied to 2x1 Multiplexer.

- If s_2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I_0 to I_3 based on the values of selection lines s_1 & s_0.
- If s_2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I_4 to I_7 based on the values of selection lines s_1 & s_0.

Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

Q. Implement the Boolean function $F(A, B, C) = \sum(1, 3, 5, 6)$ with multiplexer.

Soln:

The multiplexer can be implemented with 4 to 1 multiplexer.

Note: *It is possible to generate n+1 variables with 2^n to 1 mutiplexer.*

Now, truth table for the given function is:

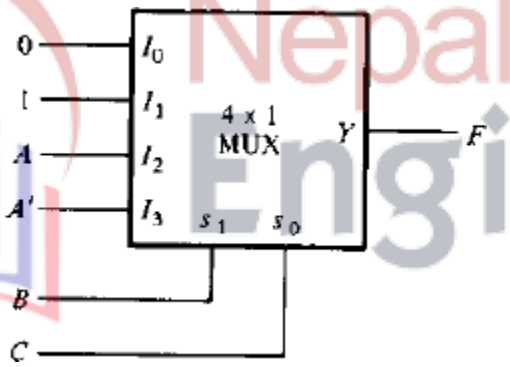
Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Now the implementation table is

	I_0	I_1	I_2	I_3
A'	0	①	2	③
A	4	⑤	⑥	7
	0	1	A	A'

- If the minterms in a column are not circled, then apply 0 to the corresponding multiplexer unit.
- If the 2 minterms are circled, then apply 1 to the corresponding multiplexer unit.
- If the bottom minterm is circled, and top is not circled then apply A to the corresponding multiplexer unit.
- If the top minterm is circled, and bottom is not circled then apply A' to the corresponding multiplexer unit.

Multiplexer implementation:



Q. Implement the Boolean function $F(A, B, C, D) = \sum(0, 1, 3, 4, 8, 9, 15)$ by multiplexer.

Sol^n:

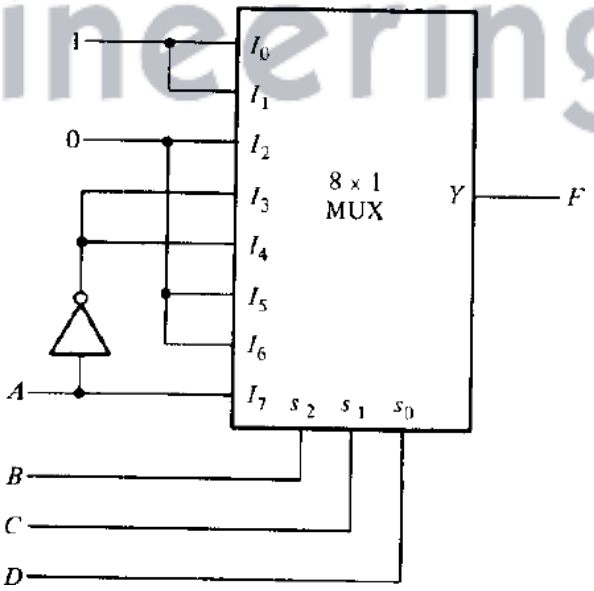
This function can be implemented with 8 to 1 MUX.

The truth table for the function is

Minterm	A	B	C	D	F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1

Now the implementation table and multiplexer implementation are given below:

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A'	⓪	①	2	③	④	5	6	7
A	⑧	⑨	10	11	12	13	14	⑮
	1	1	0	A'	A'	0	0	A



Read Only Memory (ROM)

- **A read-only memory (ROM)** is a device that includes **both the decoder and the OR gates** within a **single IC package**. The connections between the outputs of the decoder and the inputs of the OR gates can be specified for each particular configuration by "programming" the ROM.
- A ROM is essentially a memory (or storage) device in which a fixed set of binary information is stored.
- The binary information must first be specified by the user and is then embedded in the unit to form the required interconnection pattern. ROM's come with special internal linksthat can be fused or broken. The desired interconnection for a particular application requires that certain links be fused to form the required circuit paths. Once a pattern is established for a ROM, it remain fixed even when power is turned off and then on again.
- A ROM consists of n input lines and m output lines.
- Each bit combination of input variables is called an address.
- Each bit combination that comes out of the output lines is called a word. The number of bits per word is equal to the number of output lines m.
- A ROM with n input lines has 2^ndistinct addresses, so there are 2^n distinct words which

- Internally, the ROM is a combinational circuit with AND gates connected as a decoder and a number of OR gates equal to the number of outputs in the unit.

Combinational Logic implementation of ROM:

When a combinational circuit is implemented by means of ROM the function must be expressed in sum of min terms or better yet by a truth table.

Q. Implement the following combinational logic function with a 4X2 ROM.

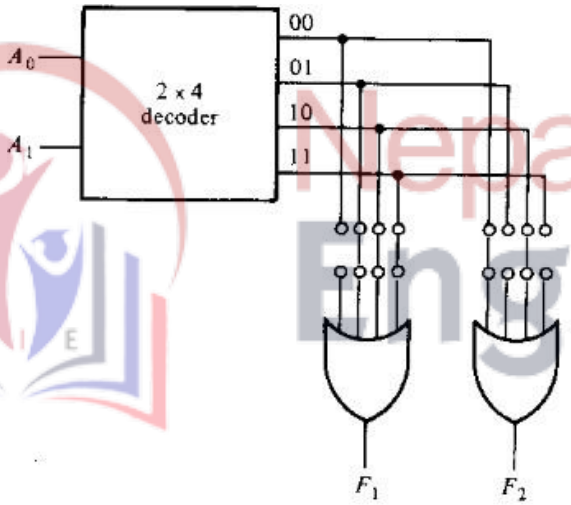
A_1	A_0	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

Sol^n:

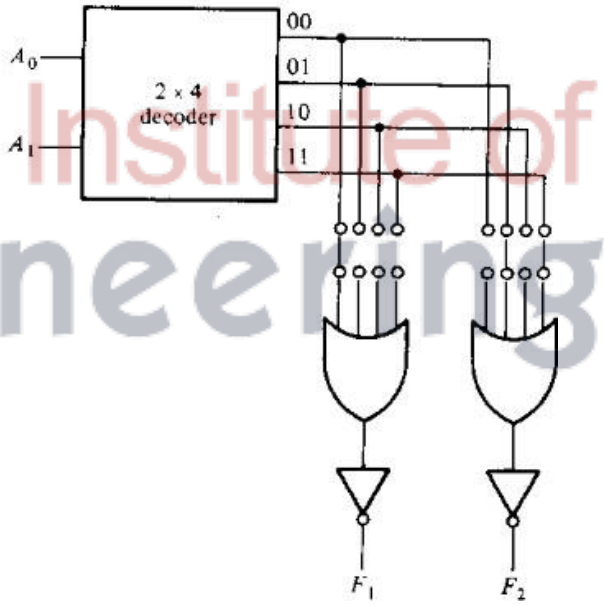
Truth table specifies a combinational circuit with 2 inputs and 2 outputs. The Boolean function can be represented in SOP as;

$$F_1(A_1, A_0) = \Sigma(1, 2, 3)$$
$$F_2(A_1, A_0) = \Sigma(0, 2)$$

Combinational-circuit implementation with a 4 x 2 ROM:



ROM with AND-OR gates



ROM with AND-OR-INVERT gates

Q. Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and generates an output binary number equal to the square of the input number.

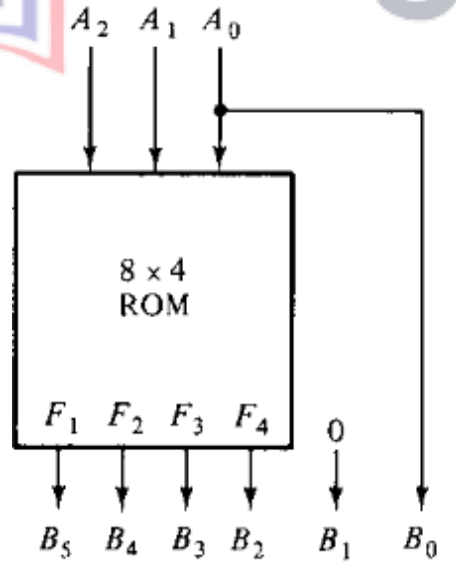
Sol^n:

First step is to derive the truth table for the combinational circuit

Inputs			Outputs						
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
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

Output B_0 is always equal to input A_0; so there is no need to generate B_0 with a ROM since it is equal to an input variable. Moreover, output B1 is always 0, so this outputs is always known.

Implementation by ROM:



(a) Block diagram

A_2	A_1	A_0	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

(b) ROM truth table

Types of ROM:

1. Mask ROM

- Permanent programming done at fabrication time
- Fabrication take place at factory as per customer order
- Very expensive and therefore feasible only for large quantity orders
- Once the memory is programmed during the manufacturing process, the user cannot alter the programs.

2. PROM (Programmable ROM)

- A blank chip which can be programmed only once using a special device called programmer.
- Once it's programmed its content cannot be modified or erased.

3. EPROM (Erasable Programmable ROM)

- Can be programmed multiple times.
- Its content can be erased by using UV (ultra violet) light.
- Exposure to the UV light will erase all contents.

4. EEPROM (Electrically Erasable Programmable ROM)

- Similar to EPROM but its contents can be electrically erased and re-written without having to remove it from the computer.

Programmable Logic Array (PLA)

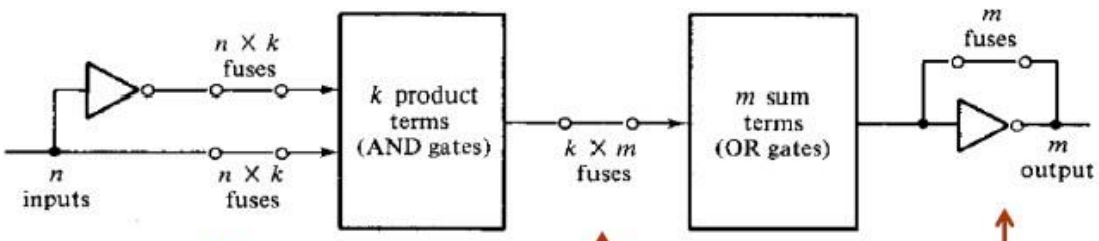
A combinational circuit may occasionally have don't care conditions. When implemented with a ROM, a don't care condition becomes an address input that will never occur. The words at the don't care addresses need not be programmed and may be left in their original state (all 0's or all 1's). The result is that not all the bit patterns available in the ROM are used, which may be considered as waste of available equipment.
For example, a combinational circuit that converts a 12-bit card code to a 6-bit internal alphanumeric code.
* It consists 12 inputs and 6 outputs. The size of the ROM must be 4096×6 ($2^{12} \times 6$).
* There are only 47 valid entries for the card code, all other input combinations are don't care. The remaining 4049 words of ROM are not used and are thus wasted.

So, ***Programmable Logic Array*** is a LSI component that can be used in economically as an alternative to ROM where number of don't-care conditions is excessive.

✓ PLA does not provide full decoding of the variables and does not generate all the minterms as in the ROM.

Block diagram of PLA:

A block diagram is shown in fig. It consists n inputs, m-outputs, k product terms and m sum terms. The product terms constitute a group of k AND gates and the sum terms constitute a group of m OR gates.



Links between all n inputs and their complement values to each of the AND gates.
2n X k links

Links between outputs of the AND gates and the inputs of the OR gates.
k X m links

Links to generate AND-OR form or, AND-OR-INVERT form
m links

✓ The number of programmed links is $2n \times k + k \times m + m$, whereas that of a ROM is $2^n \times m$.

Implementation of combinational circuit by PLA:

$F_1 = AB' + AC$

$F_2 = AC + BC$

PLA program table:

Product term		Inputs A	B	C	Outputs F_1	F_2
AB'	1	1	0	–	1	–
AC	2	1	–	1	1	1
BC	3	–	1	1	–	1
					T	T

T/C

Input side:
1=uncomplemented in term
0=complemented in term
- = does not participate
Output side:
1= term connected to output
- = no connection to output

PLA Logic Circuit:

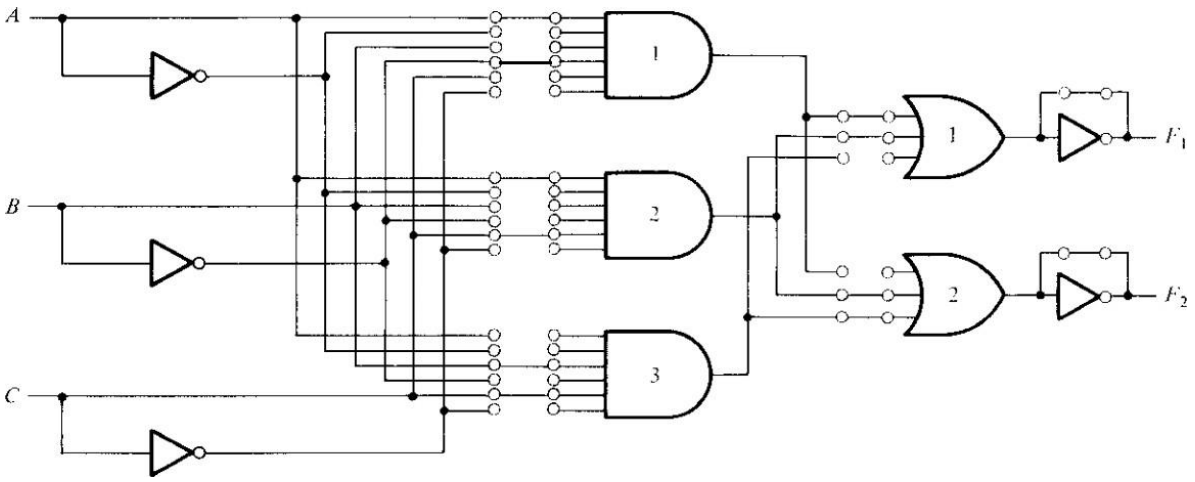


Fig: PLA with 3 inputs, 3 product terms, and 2 outputs

PLA program table consists of three columns:
- ***First column:*** lists the product terms numerically.
- ***Second column:*** specifies the required paths between inputs and AND gates.
- ***Third column:*** specifies the paths between the AND gates and the OR gates.

Under each output variable, we write a *T* (for true) if the output inverter is to be bypassed, and C (for complement) if the function is to be complemented with the output inverter.

Note: PLA implements the functions in their sum of products form (standard form, not necessarily canonical as with ROM). Each product term in the expression requires an AND gate. It is necessary to simplify the function to a minimum number of product terms in order to minimize the number of AND gates used.

Q. A combinational circuit is defined by the functions:

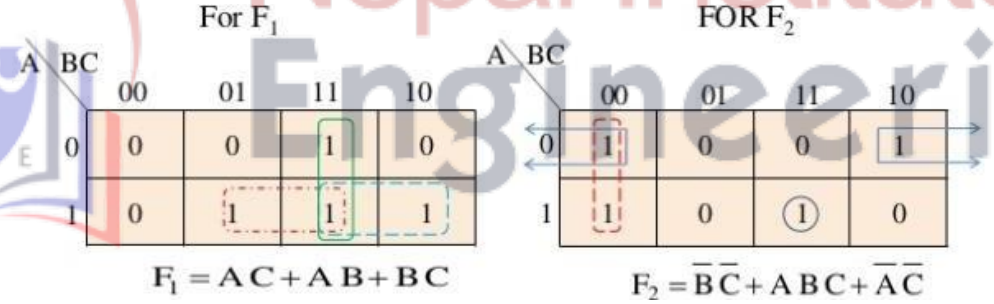
$$F_1(A, B, C) = \sum(3, 5, 6, 7)$$

$$F_2(A, B, C) = \sum(0, 2, 4, 7)$$

Implement the circuit with a PLA having three inputs, four product terms, and two outputs.

Sol^n:

First of all we have to write the function in minimize SOP form:



There are six product terms in F_1 and F_2, but only four product terms are allowed to use.

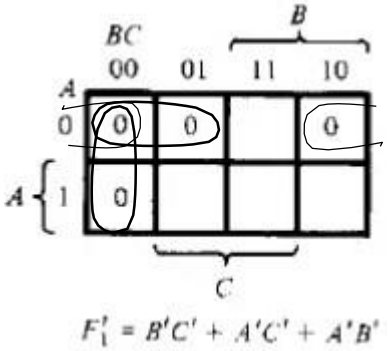
Now implement $F_1'(A, B, C)$

$$F_1'(A, B, C) = \sum(0, 1, 2, 4)$$

$$F_2(A, B, C) = \sum(0, 2, 4, 7)$$

From these equation it is clear that the minterms 0, 2 and 4 are common.

Now obtain the minimized expression by using them



Now four product terms are $B'C', A'C', A'B'$ *and* ABC.

$F_1 = B'C' + A'C' + A'B'$
$F_2 = B'C' + ABC + A'C'$

Now, PLA program table:

	Product term	Inputs A	B	C	Outputs F_1	F_2
$B'C'$	1	–	0	0	1	1
$A'C'$	2	0	–	0	1	1
$A'B'$	3	0	0	–	1	–
ABC	4	1	1	1	–	1
					C	T	T/C

Note that output F_1 is the normal (or true) output even though a C is marked under it. This is because F_1' is generated prior to the output inverter. The inverter complements the function to produce F_1 in the output.
Draw PLA circuit yourself.

(Refer to class notes for PLA and PLA questions)