Unit 5

Combinational Logic with MSI and LSI

Content:

- Binary Parallel Adder
- Decimal Adder
- Magnitude Comparator
- Decoders
- Multiplexers
- Read- Only- Memory (ROM)
- Programmable Logic array (PLA)

Binary Adder

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

The sum of *A* and *B* can be generated in two ways:

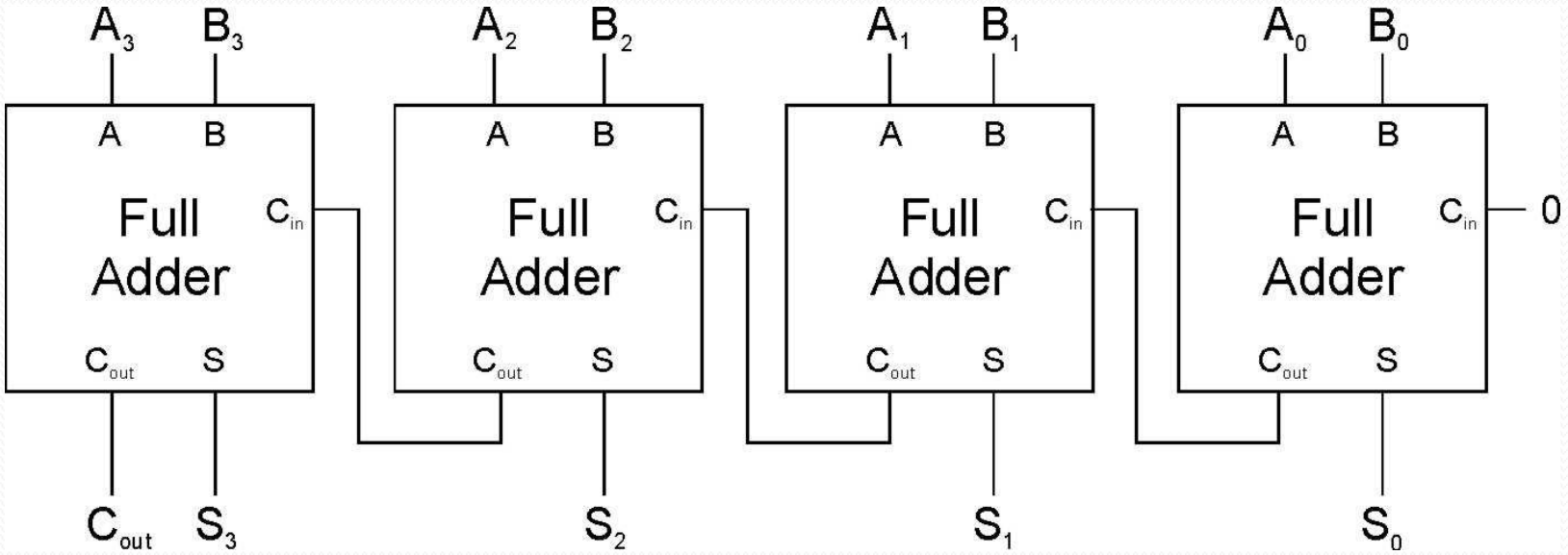
- The *serial addition method* uses only one full-adder circuit and a storage device to hold the generated output carry.

- The *parallel method* uses *n* full-adder circuits, and all bits of *A* and *B* are applied simultaneously

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 a chain, with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.
- Successive addition occurs.

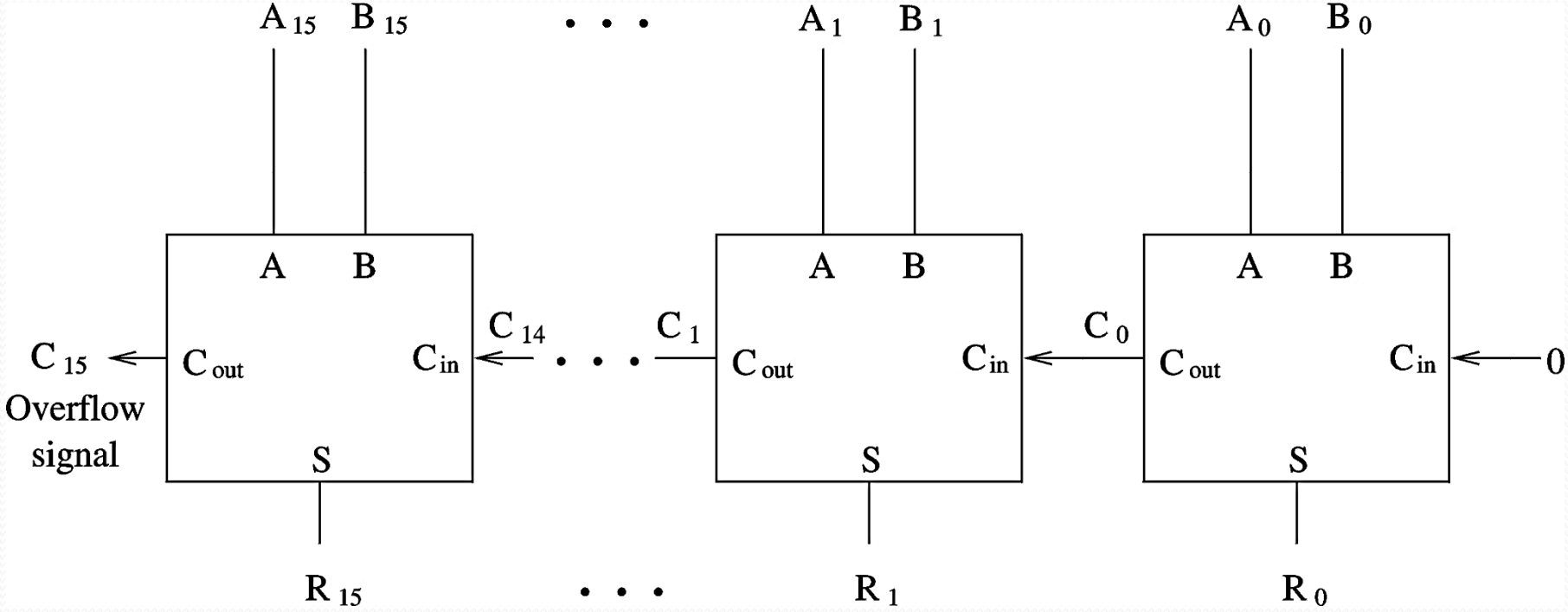
- Four-bit Adder or 4-Bit Ripple Carry Adder



Input Carry,	C_{in}	0	1	0	0
Augend	A	1	0	1	0
Addend	B	+ 0	0	1	1
Sum	S	1	1	1	1
Output Carry.	C_{out}	0	0	1	0

Input Carry,	C_{in}	0	1	1	0
Augend	A	1	0	1	1
Augend	B	+ 0	0	1	1
Sum	S	1	1	1	0
Output Carry.	C_{out}	0	0	1	1

- A 16-bit ripple-carry adder



Note:

The carry propagation time is a limiting factor on the speed with which two numbers are added in parallel. There are several techniques for reducing the carry propagation time in a parallel adder. The most widely used technique employs the principle of ***look-ahead*** **carry**.

Logic For Look Ahead Carry:

We know,

For full adder:

$$S_i = (A_i \oplus B_i) \oplus C_i$$

$$C_{i+1} = A_i B_i + (A_i \oplus B_i) C_i$$

Let,

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

Then,

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

4-bit full adders with Look-ahead Carry (Fast Adder)

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

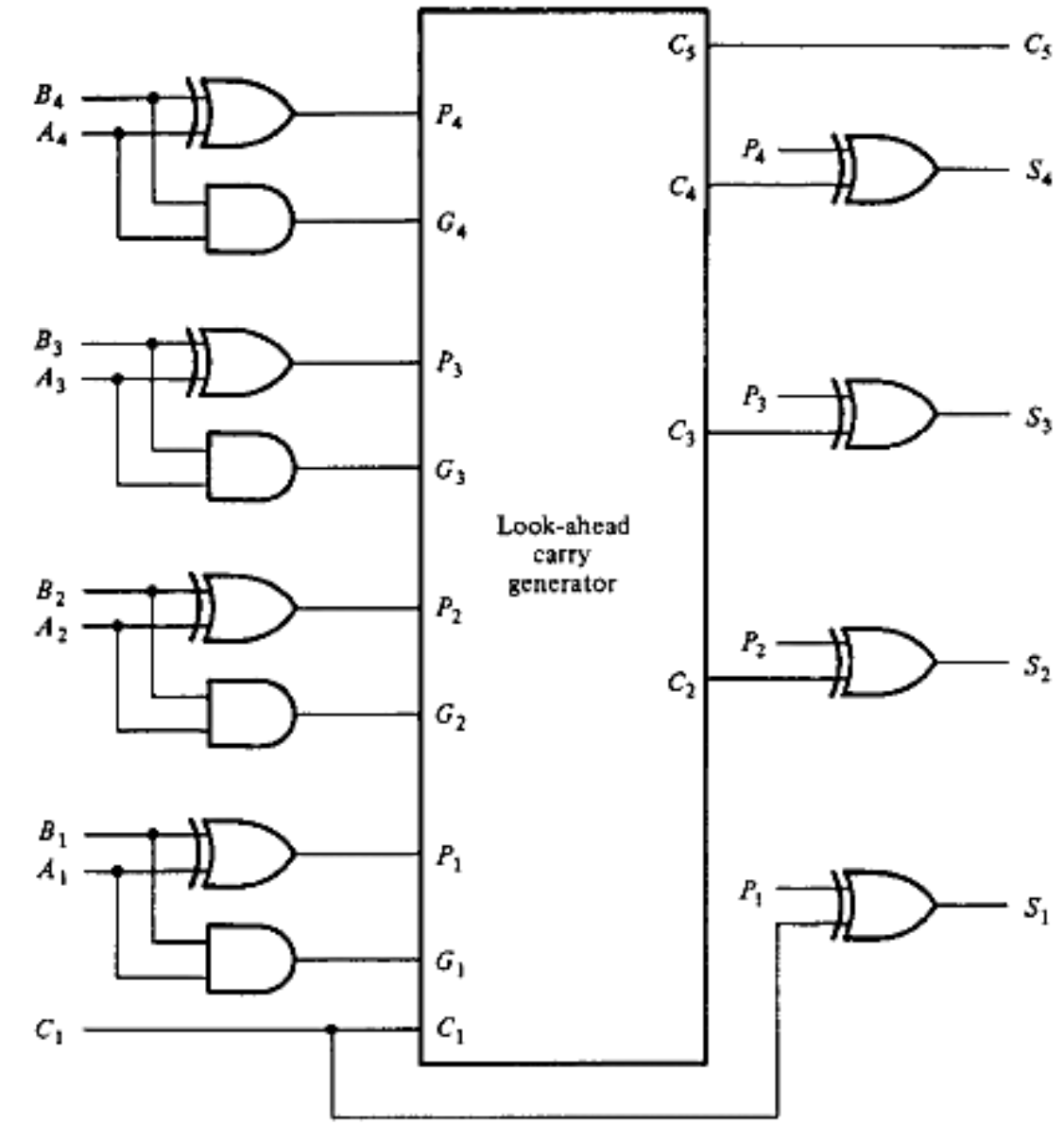
Where,

P= Propagation Carry

G= Generation Carry

The output sum and carry can be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$



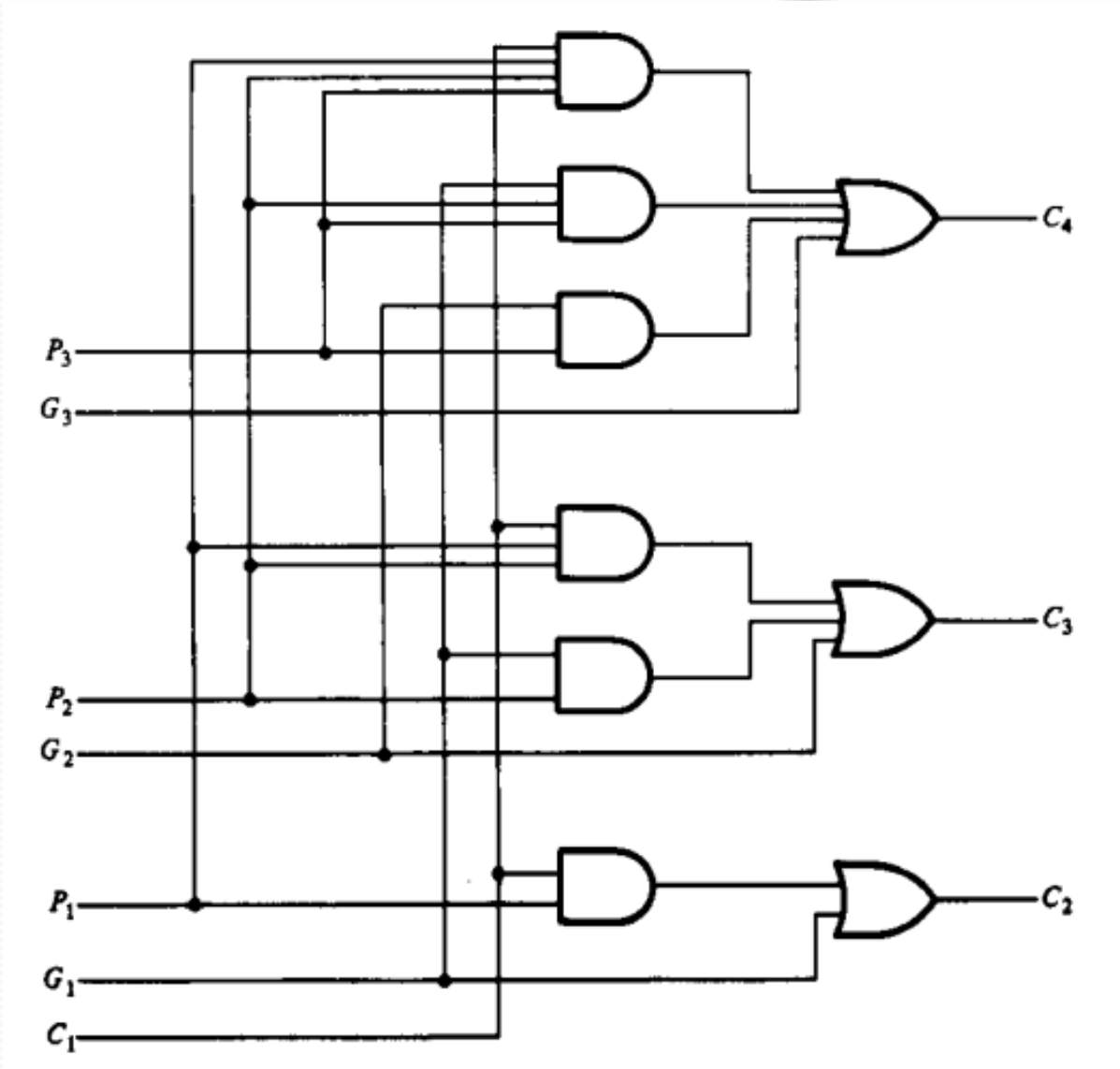
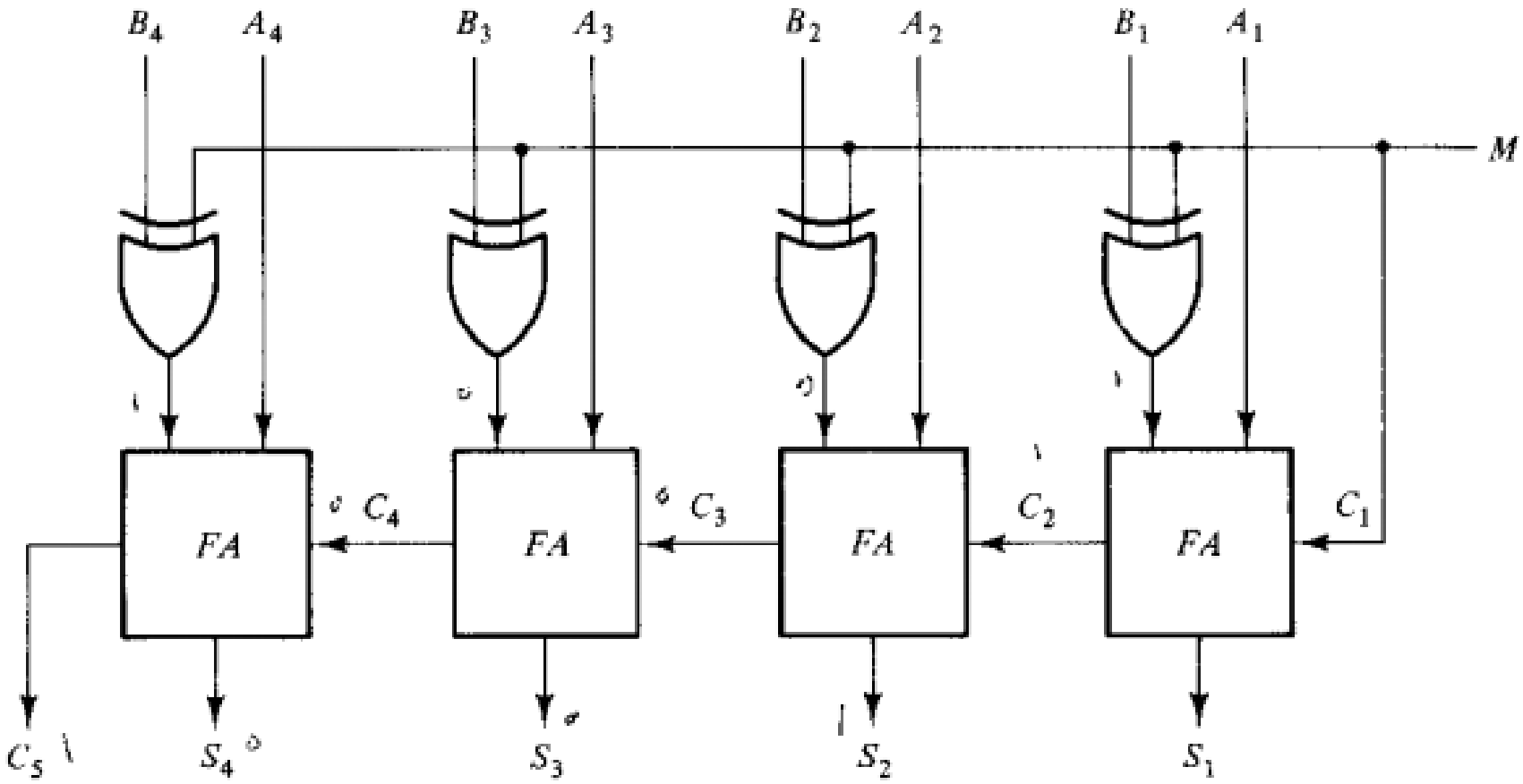


Fig: Logic Diagram of look-ahead carry generator

Q. Construct a 16-bit parallel adder with four MSI circuits, each containing a 4-bit parallel adder. Use a block diagram with nine inputs and five outputs for each 4-bit adder. Show how the carries are connected between the MSI circuits.

- **4-Bit Adder-Subtractor**



(b) 4-bit adder-subtractor

The addition and subtraction operations can be combined into one circuit with one common binary adder. This is done by including an exclusive-OR gate with each full- adder.
A 4-bit adder-subtractor circuit is shown in Fig. above.

The mode input M controls the operation.
Each exclusive-OR gate receives input M and one of the inputs of B.

- When M = 0, the circuit is an adder.

When M = 0, we have B $\oplus$ 0 = B. The full-adders receive the value of B , the input carry is 0, and the circuit performs A plus B.

- When M = 1, the circuit becomes a subtractor.

When M = 1, we have B $\oplus$ 1 = B ' and input carry = 1. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2’s complement of B.

Decimal Adder

Computers or calculators that perform arithmetic operations directly in the decimal number system represent decimal numbers in binary-coded form(BCD).

- **Decimal adder** is a combinational circuit that sums up two decimal numbers adopting particular encoding technique.
- A decimal adder requires a *minimum of* nine inputs(4+4+1) and five outputs, since four bits are required to code each decimal digit and the circuit must have an input carry and output carry.
- Of course, there is a wide variety of possible decimal adder circuits, dependent upon the code used to represent the decimal digits.

BCD Adder

This combinational circuit adds up two decimal numbers when they are encoded with binary-coded decimal (BCD) form.

- Adding two decimal digits in BCD, together with a possible carry, the output sum cannot be greater than 9 + 9 + 1 = 19.
- Applying two BCD digits to a 4-bit binary adder, the adder will form the sum in *binary* ranging from 0 to 19. These binary numbers are listed in Table below and are labeled by symbols K, Z_8, Z_4, Z_2, Z_1.

K is the carry, and the subscripts under the letter Z represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code.

Derivation of a BCD Adder

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

- The first column in the table lists the binary sums as they appear in the outputs of a 4-bit binary adder.
- The output sum of two decimal digits must be represented in BCD and should appear in the form listed in the second column of the table.
- The problem is to find a simple rule by which the binary number, in the first column can be converted to the correct BCD-digit representation of the number in the second column.

Looking at the table, we see that:

- When (binary sum) <= 1001

  Corresponding BCD number is identical, and therefore no conversion is needed.

- When (binary sum) > 1001

  Non-valid BCD representation is obtained. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

The logic circuit that detects the necessary correction can be derived from the table entries.

- **Correction** is needed when
  - The binary sum has an output carry $K = 1$.
  - The other six combinations from 1010 to 1111 that have $Z_8 = 1$. 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 a 1. The condition for a correction and an output carry can be expressed by the Boolean function

$$C = K + Z_8 Z_4 + Z_8 Z_2$$

- When C = 1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage.

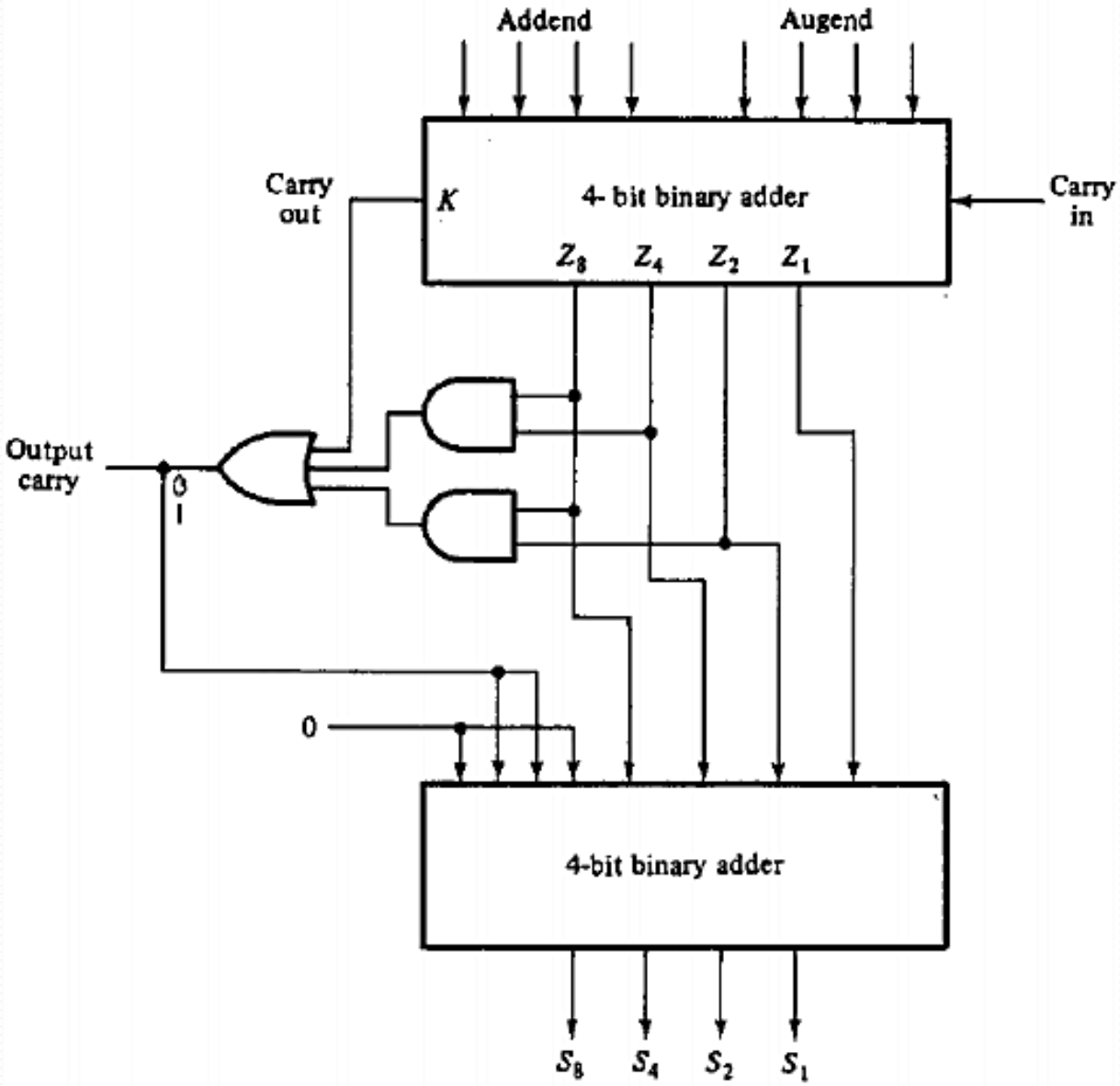


Fig: Block diagram of BCD adder

Note:

A decimal parallel adder that adds *n* decimal digits needs *n* BCD adder stages. The output carry from one stage must be connected to the input carry of the next higher order stage

Magnitude Comparator

A *Magnitude comparator* is a combinational circuit that compares two numbers, *A* and *B*, and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicate whether $A > B$, $A = B$, or $A < B$.

Consider two numbers, *A* and *B*, with four digits each. The coefficients of the numbers with descending significance as follows:

$$A = A_3A_2A_1A_0$$
$$B = B_3B_2B_1B_0$$

Where each subscripted letter represents one of the digits in the number, the two numbers are equal if all pairs of significant digits are equal, i.e., if $A_3 = B_3$ and $A_2 = B_1$ and $A_1 = B_1$ and $A_0 = B_0$

When the numbers are binary, the digits are either 1 or 0 and the equality relation of each pair of bits can be expressed logically with an equivalence function obtained from the truth table:

Truth Table:

A	**B**	**X**
0	0	1
0	1	0
1	0	0
1	1	1

Boolean Expression:

$$X_i = A_iB_i + A_i'B_i', \qquad i = 0, 1, 2, 3$$

Where X_i = 1 only if the pair of bits in position i are equal, i.e., if both are 1's or both are 0's.

Algorithm :

(A = B)

For the equality condition to exist, all X_i variables must be equal to 1. This dictates an AND operation of all variables:

$$(A = B) = X_3 X_2 X_1 X_0$$

The *binary* variable *(A = B)* is equal to 1 only if all pairs of digits of the two numbers are equal.

(A<B) or (A>B)

To determine if *A* is greater than or less than *B*, we check the relative magnitudes of pairs of significant digits starting from the most significant position. If the two digits are equal, we compare the next lower significant pair of digits. This comparison continues until a pair of unequal digits is reached.

A > B: If the corresponding digit of A is 1 and that of B is 0.

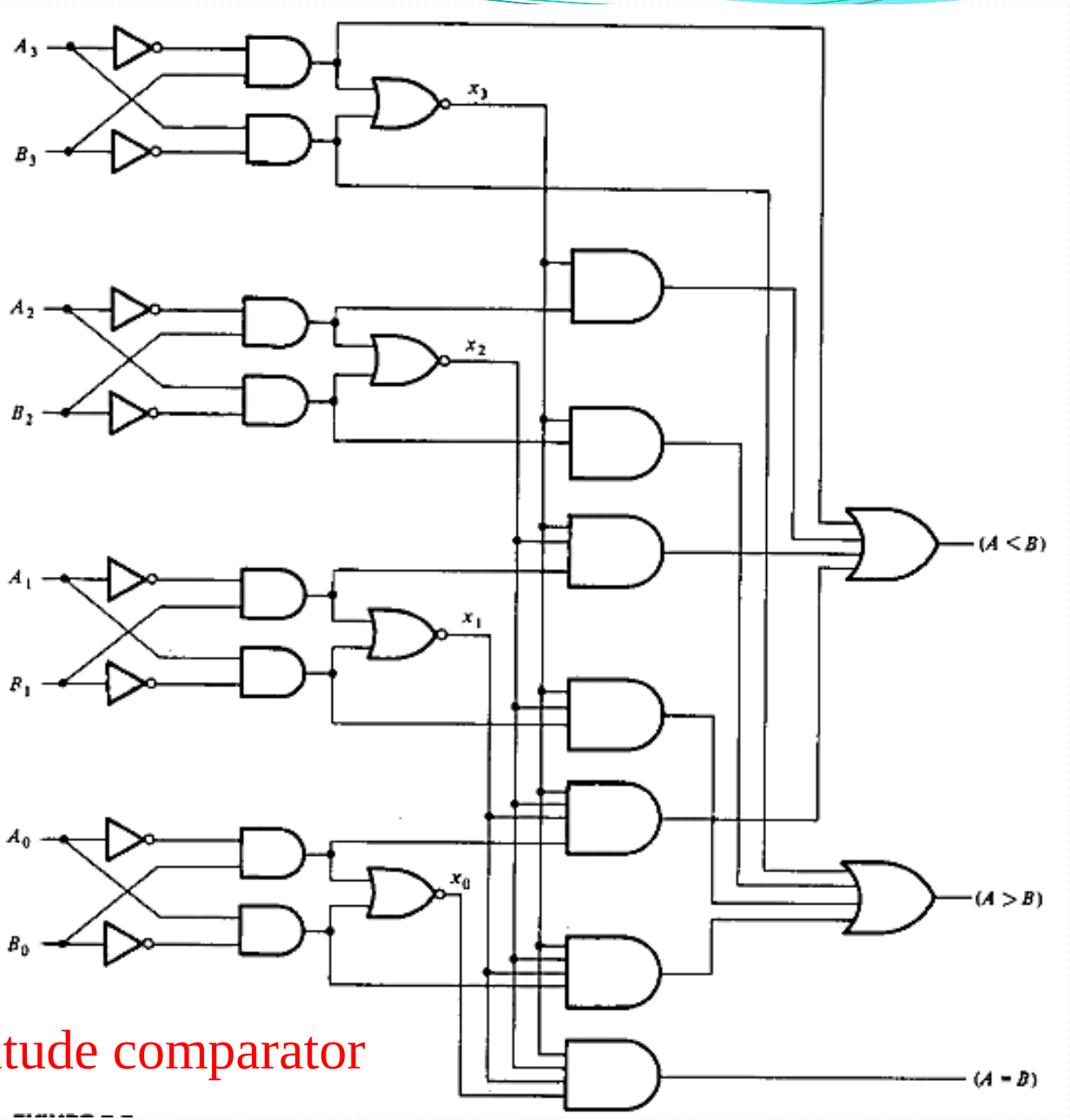
A < B: If the corresponding digit of A is 0 and that of B is 1.

The sequential comparison can be expressed logically by the following two Boolean functions:

$$(A > B) = A_3B_3' + X_3 A_2B_2' + X_3 X_2A_1B_1' + X_3 X_2X_1 A_0B_0'$$

$$(A < B) = A_3'B_3 + X_3 A_2'B_2 + X_3 X_2A_1'B_1 + X_3 X_2X_1 A_0'B_0$$

The symbols *(A > B)* and *(A < B)* are *binary* output variables that are equal to 1 when *A > B* or *A < B* respectively.



A<B

A>B

A=B

Fig: 4-bit magnitude comparator

Decoders

- a combinational circuit that converts binary information from *n* input lines to a maximum of 2^n unique output lines.

Example: **3-to-8 line decoder**

The 3 inputs are decoded into 8 outputs, each output representing one of the minterms of the 3-input variables.

Inputs			Outputs							
x	*y*	*z*	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Table: Truth-table for 3-to-8 line Decoder

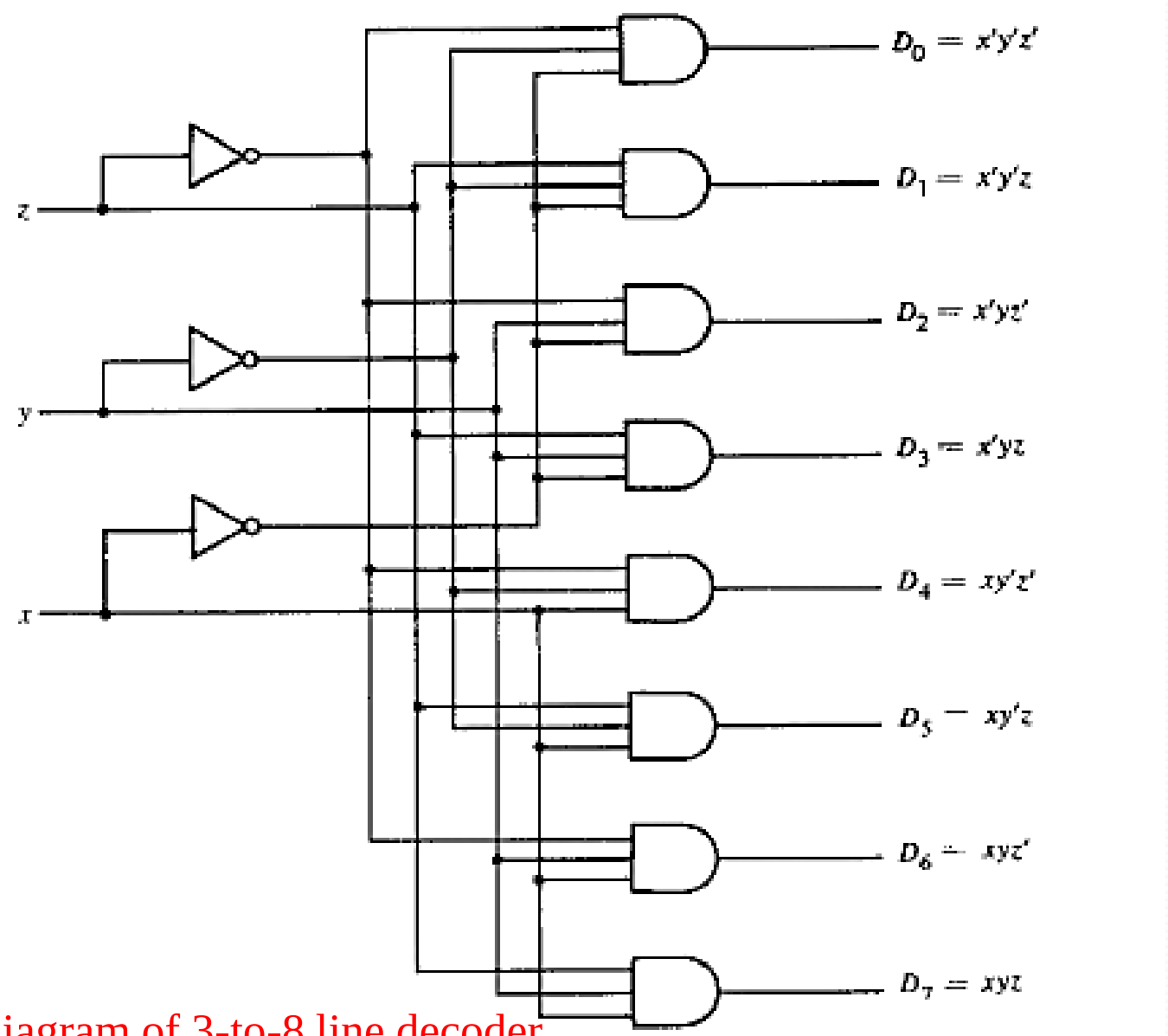


Fig: logic diagram of 3-to-8 line decoder

Combinational logic Implementation

A decoder provides the 2^n minterm of *n* input variables. Since any Boolean function can be expressed in sum of minterms canonical form, one can use a decoder to generate the minterms and an external OR gate to form the sum.

- Any combinational circuit with *n* inputs and *m* outputs can be implemented with an n-to- 2^n line decoder and *m* OR gates.

- Boolean functions for the Decoder-implemented-circuit are expressed in sum of minterms. This form can be easily obtained from the truth table or by expanding the functions to their sum of minterms.

Example: *Implement a full-adder circuit with a decoder.*

Solution: From the truth table of the full-adder, we obtain the functions for this combinational circuit in sum of minterms as:

$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$

$$C(x, y, z) = \Sigma(3, 5, 6, 7)$$

Since there are three inputs and a total of eight minterms, we need a 3-to-8-line decoder.

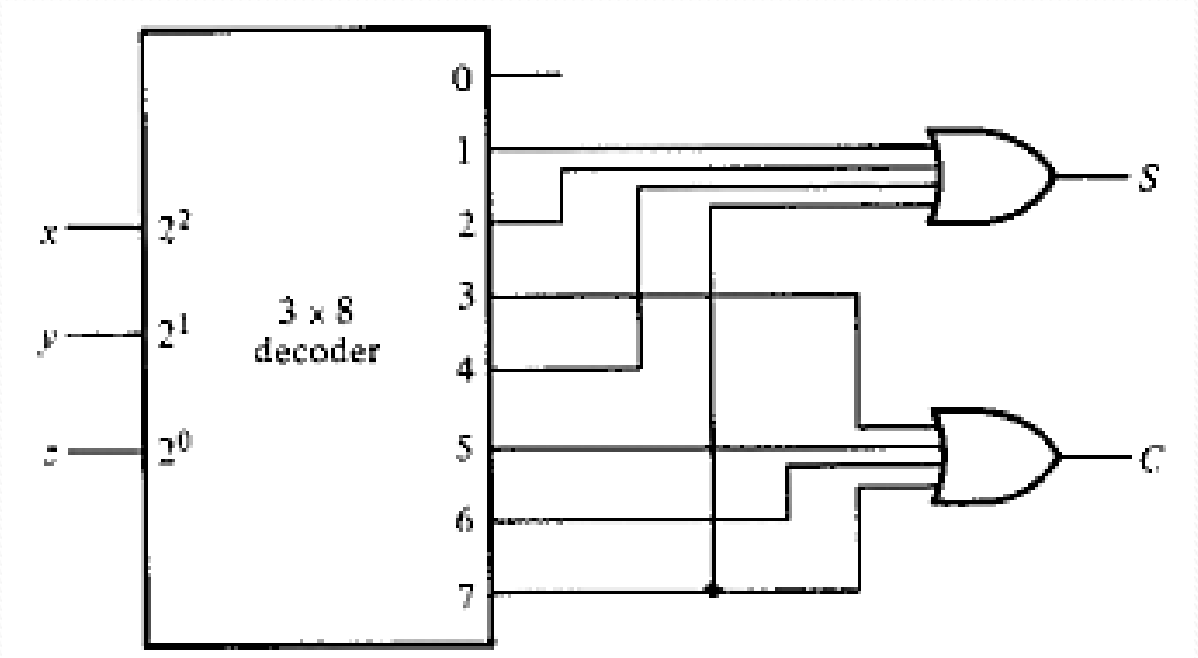


Fig: Implementation of Full-adder with a decoder circuit

Encoder

- 2^n (or fewer) input lines and *n* output lines.
- A combinational circuit that converts an active input signal into a coded output signal.
- An example of an encoder is the octal-to-binary encoder which has eight inputs, one for each of the octal digits, and three outputs that generate the corresponding binary number.

Truth Table of Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

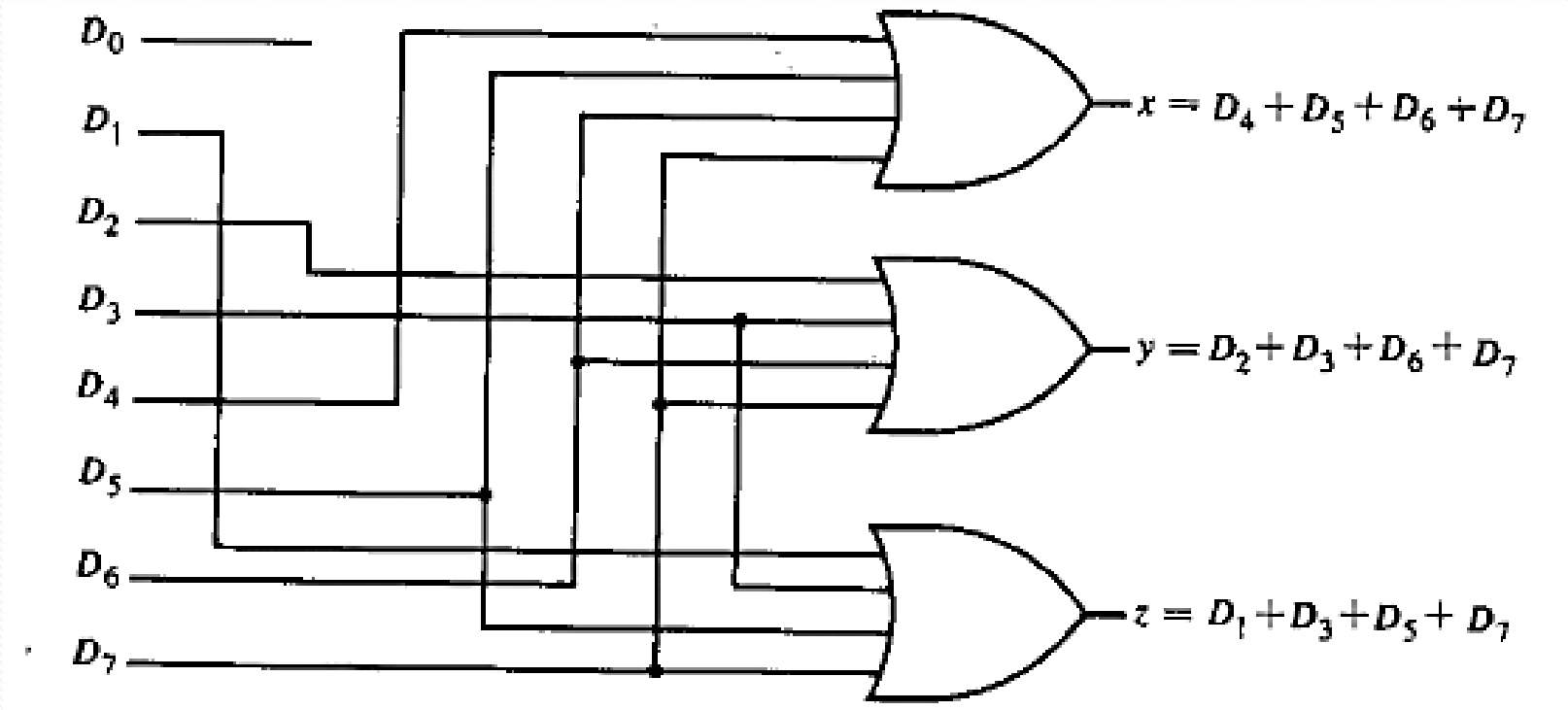


Fig: Octal-to-binary encoder

Note: The encoder defined in above truth table has the limitation that only one input can be active at any given time. If two inputs are active simultaneously, the output produces an undefined combination

For example, if D_3 and D_6 are 1 at same time, the output of the encoder will be 111 because all three outputs are equal to 1. This does not represent binary 3 nor binary 6. To resolve this ambiguity, encoder circuits must establish a priority to ensure that only one input is encoded. If we establish a higher priority for inputs with higher subscript numbers, and if both D_3 and D_6 are 1 at the same time, the output will be 110 because D_6 has higher priority than D_3.

Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0. The problem is that an output with all 0's is also generated when Do is equal to 1. This ambiguity can be resolved by providing an additional output that specifies the condition that none of the inputs are active.

Priority Encoder

- an encoder circuit that includes the priority function.
- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.
- Eg: In 4 x 2 encoder, Input D_3 has the highest priority; so regardless of the values of the other inputs, when this input is 1, the output is 11

Note: X 's are don't-care conditions

Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Boolean functions: $x = D_2 + D_3$

$y = D_3 + D_1 D'_2$

$V = D_0 + D_1 + D_2 + D_3$

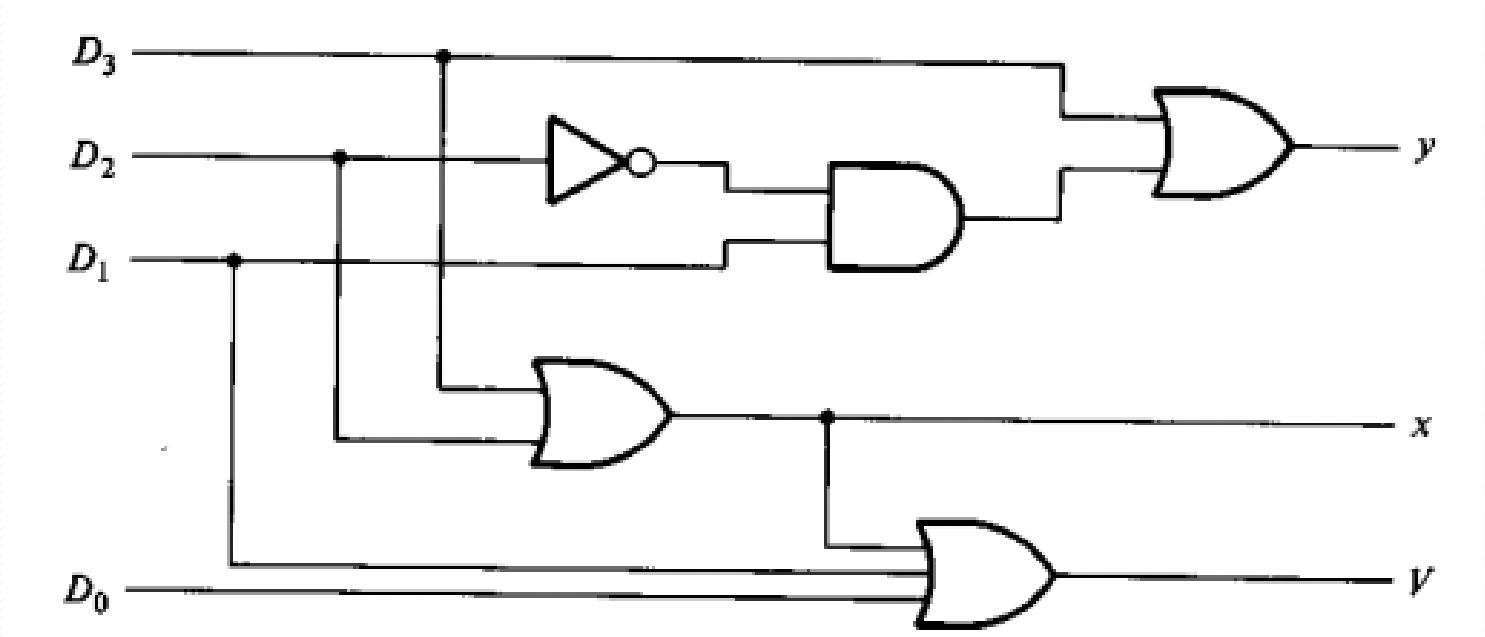


Fig: Logic Diagram of 4-input priority encoder

Multiplexer

- Multiplex means ‘many to one’.
- a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.
- Also called as a data selector since selects one of many inputs and steers the information to the output.
- Eg: 4 x 1 MUX

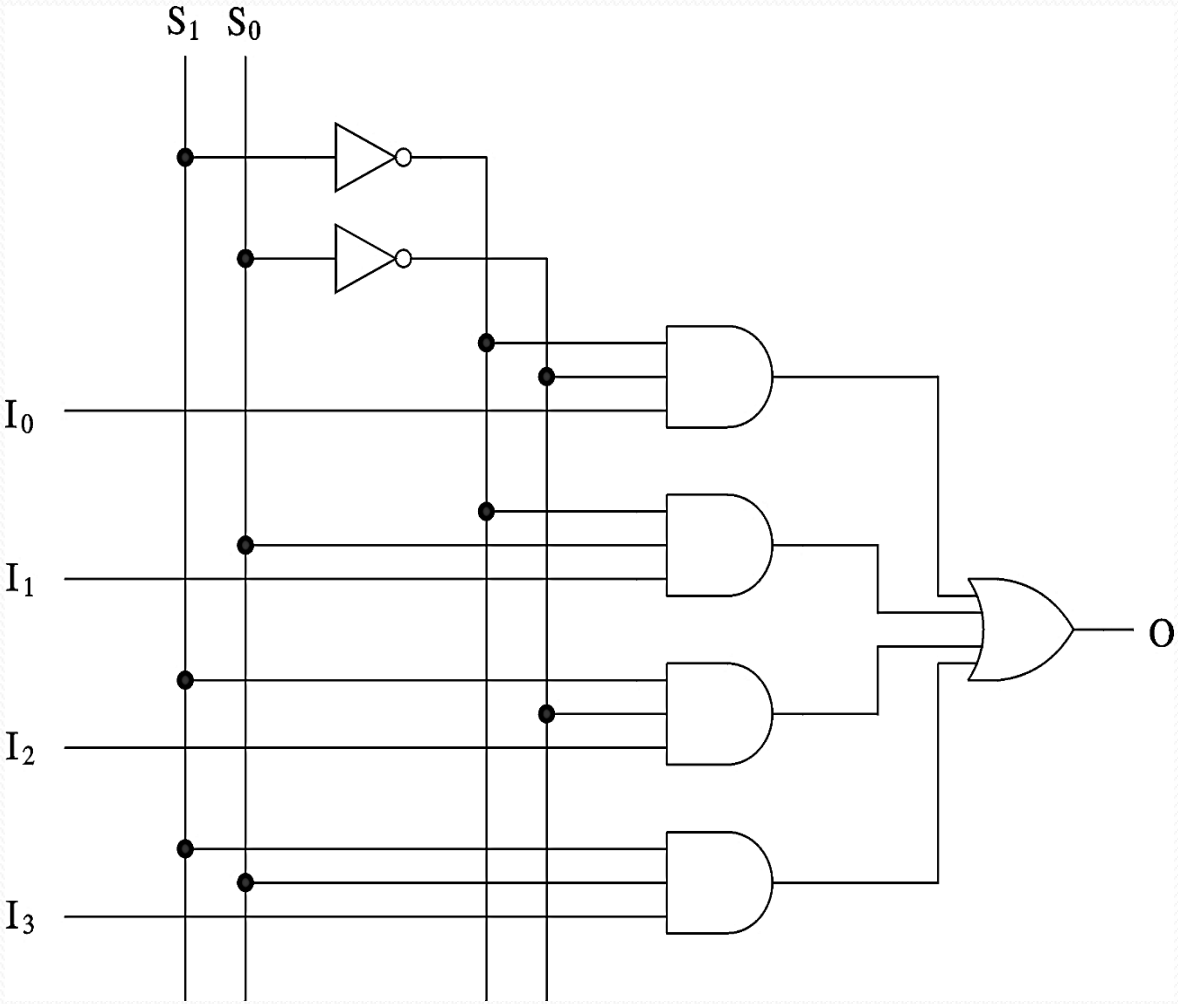


Fig: Logic Diagram: 4-to-1 line Multiplexer

S_1	S_0	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table: Function table

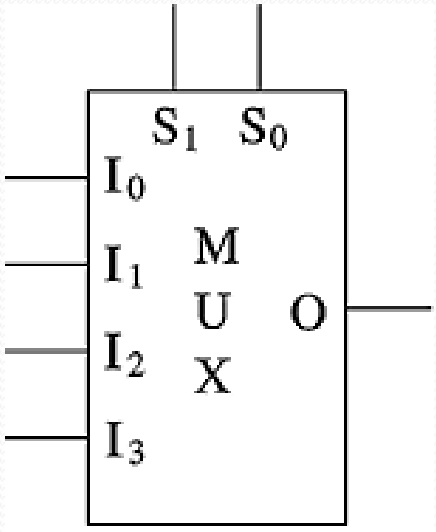


Fig: Block Diagram of Multiplexer

Here: $Y = O = s_1's_0'I_0 + s_1's_0I_1 + s_1s_0'I_2 + s_1s_0I_3$

To demonstrate the above circuit,

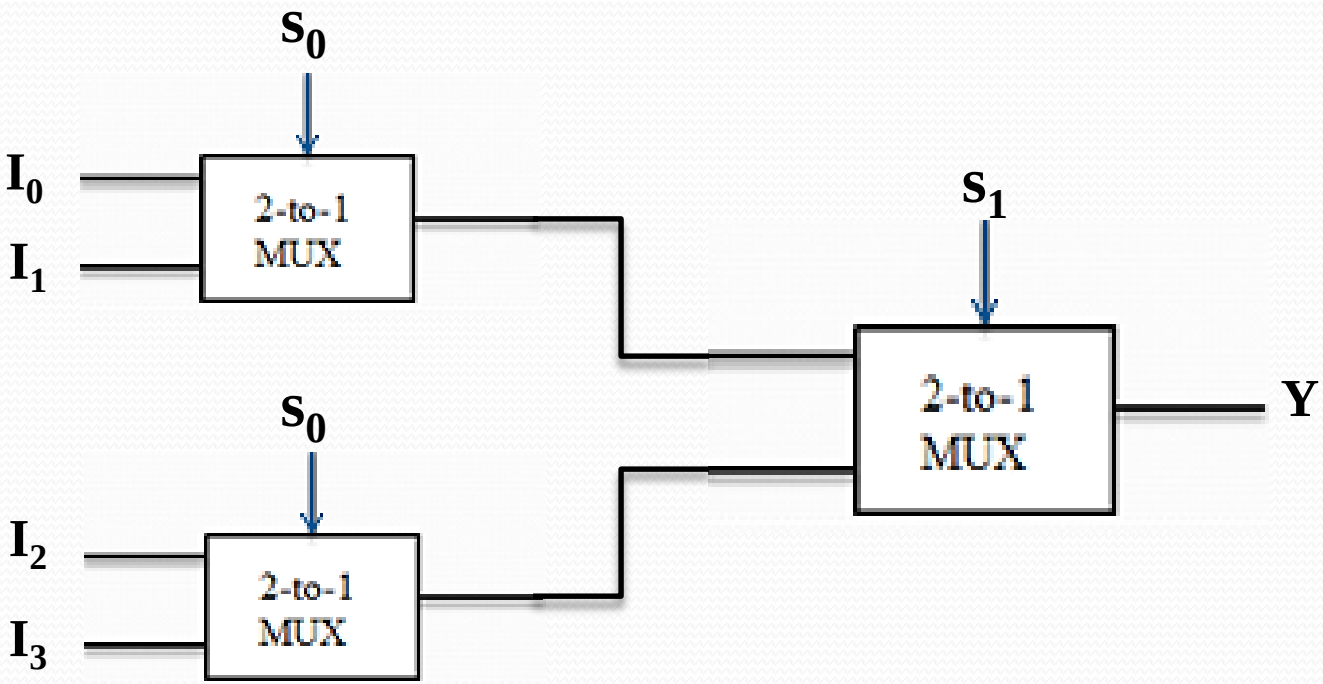
- Consider the case when $s_1s_2 = 10$.
- The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2.
- The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0.
- The OR gate output is now equal to the value of I_2, thus providing a path from the selected input to the output.

Q. Show how 4-to-1 multiplexer can be obtained using 2-to-1.

Solution:

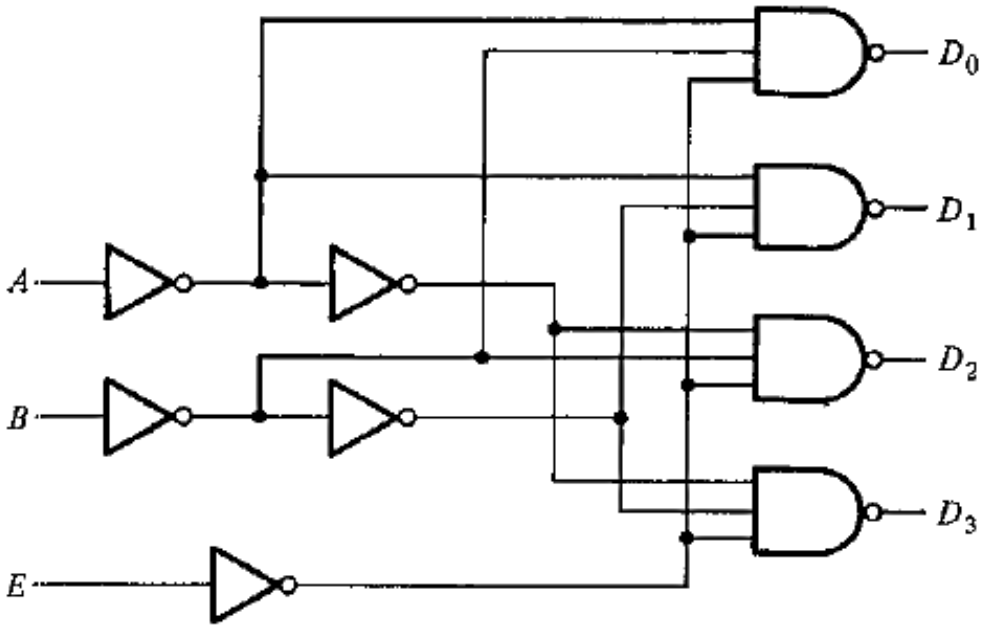
Logic equation for 2-to-1 MUX $Y = s'I_0 + sI_1$

Logic equation for 4-to-1 MUX $Y = s_1's_0'I_0 + s_1's_0I_1 + s_1s_0'I_2 + s_1s_0I_3$



De-multiplexer

- De-multiplex means “one to many”.
- A Decoder with an enable input can function as a demultiplexer.
- is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines. The selection of a specific output line is controlled by the bit values of *n* selection



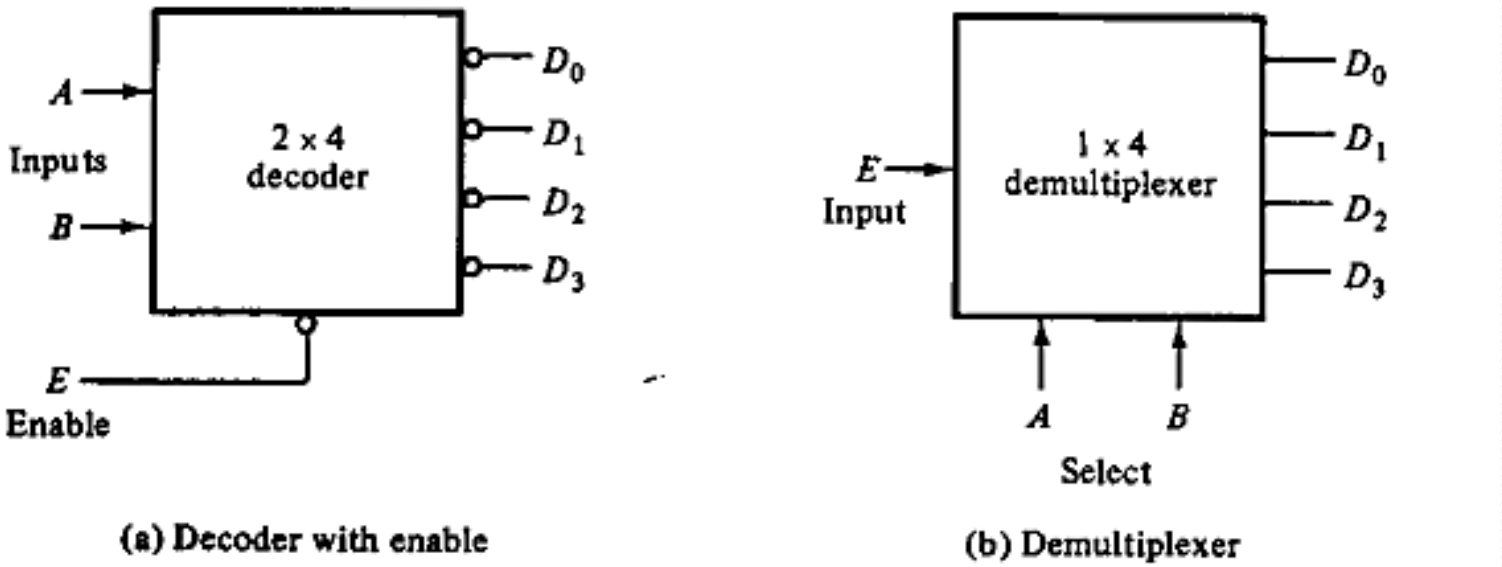
(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

Fig: 2-to-4-line decoder with enable (E) input

- The decoder of figure (a) below can function as a demultiplexer if the E line is taken as a data input line and lines A and B are taken as the selection lines. This is shown in figure (b). The single input variable E has a path to all four outputs, but the input information is directed to only one of the output lines, as specified by the binary value o f the two selection lines, A and B



(a) Decoder with enable

(b) Demultiplexer

This can be verified from the truth table of this circuit, shown in previous slide. For example, if the selection lines AB = 10, output D_2 will be the same as the input value E, while all other outputs are maintained at 1.

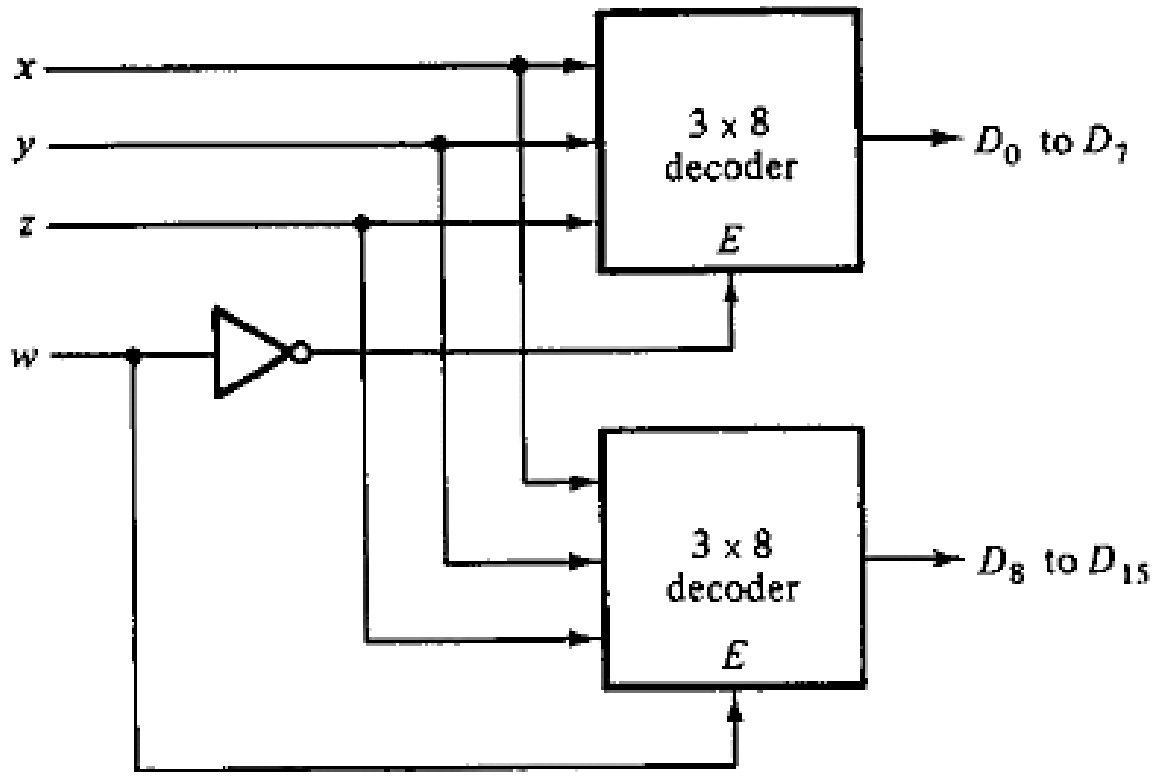


Fig: A 4 x 16 decoder constructed with two 3 x 8 decoders

Explanation of above figure:

Decoder/demultiplexer circuits can be connected together to form a larger decoder circuit. Figure 5-12 shows two 3 x 8 decoders with enable inputs connected to form a 4 x 16 decoder. When w = 0, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's, and the top eight outputs generate minterms 0000 to 0111, When w = 1, the enable conditions are reversed; the bottom decoder outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 0's.

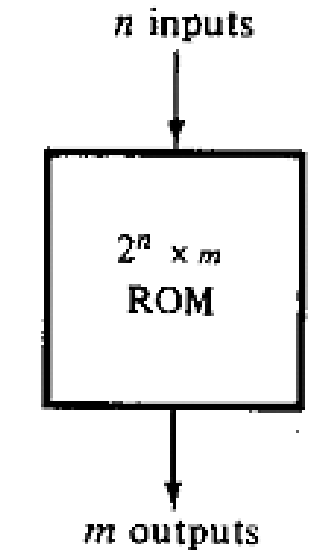
Assignment

- Design an encoder using universal gates.
- Design the full subtractor circuit with using Decoder and explain the working principle.
- Explain the decoder and design with universal gates.
- Implement the following using decoder.
  a) F (W X Y Z) = $\sum$ (0, 1, 3, 4, 8, 9, 10)
  b) F (W X Y Z) = $\sum$ (1, 3, 5, 6, 11, 13, 14)
- Design a 3 to 8 line decoder using two 2 to 4 line decoder and explain it.
- Realize Y = A'B+B'C'+ABC using an 8-to-1 MUX.
- Design 8 x 1 MUX using two 4 x 1 MUX.
- Differentiate between a MUX and DEMUX.
- Design a multiplexer 4 x 1 using only universal gates.
- Draw a block diagram, truth table and logic circuit of 1x16 Demultiplexer and Multiplexer and explain its working principle.

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. The ROM is used to implement complex combinational circuits within one IC package or as permanent storage for binary information.

A ROM is essentially a memory (or storage) device in which permanent binary information is stored. The binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern. ROMs come with special internal electronic fuses that can be "programmed" for a specific configuration. Once the pattern is established, it stays within the unit even when power is turned off and on again.



Block diagram of ROM

It consists of *n* input lines and *m* output lines. Each bit combination of the 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*.

$$2^n * m = 2^n \text{ words of m bit}$$

An address is essentially a binary number that denotes one of the minterms of *n* variables. The number of distinct addresses possible with *n* input variables is 2^n.

The number of addressed words in a ROM is determined from the fact that n input lines are needed to specify 2^n words. A ROM is sometimes specified by the total number of bits it contains, which is 2^n x m. For example, a 2048-bit ROM may be organized as 512 words of 4 bits each. This means that the unit has four output lines and nine input lines to specify $2^9 = 512$ words. The total number of bits stored in the unit is 512 x 4 = 2048.

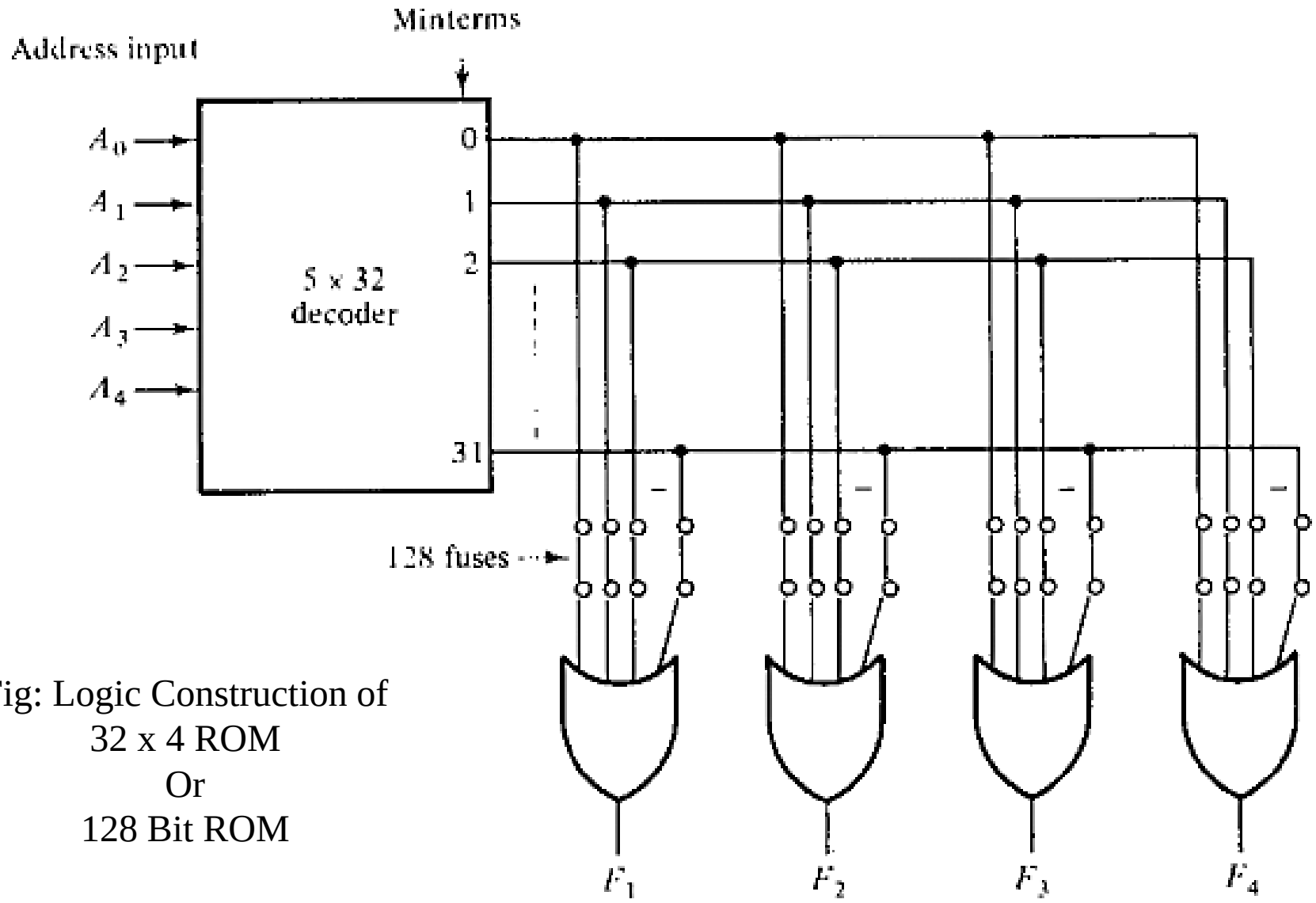


Fig: Logic Construction of
32 x 4 ROM
Or
128 Bit ROM

Explanation:

The five input variables are decoded into 32 lines. Each output of the decoder represents one of the minterms of a function of five variables. Each one of the 32 addresses selects one and only one output from the decoder. The address is a 5-bit number applied to the inputs, and the selected minterm out of the decoder is the one marked with the equivalent decimal number. The 32 outputs of the decoder are connected through fuses to each OR gate. Only four of these fuses are shown in the diagram, but actually each OR gate has 32 inputs and each input goes through a fuse that can be blown as desired.

If the input address is 00000, word number 0 is selected and it appears on the output lines. If the input address is 11111, word number 31 is selected and applied to the output lines. In between, there are 30 other addresses that can select the other 30 words.

Combinational Logic Implementation

From the logic diagram of the ROM, it is clear that each output provides the sum of all the minterms of the *n* input variables. Remember that any Boolean function can be expressed in sum of minterms form. By breaking the links of those minterms not included in the function, each ROM output can be made to represent the Boolean function.

- For an n-input, *m-output* combinational circuit, we need a 2n x *m* ROM.
- The blowing of the fuses is referred to as *programming the ROM*.
- The designer need only specify a ROM program table that gives the information for the required paths in the ROM. The actual programming is a hardware procedure that follows the specifications listed in the program table.

Example: Consider a following truth table:

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

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

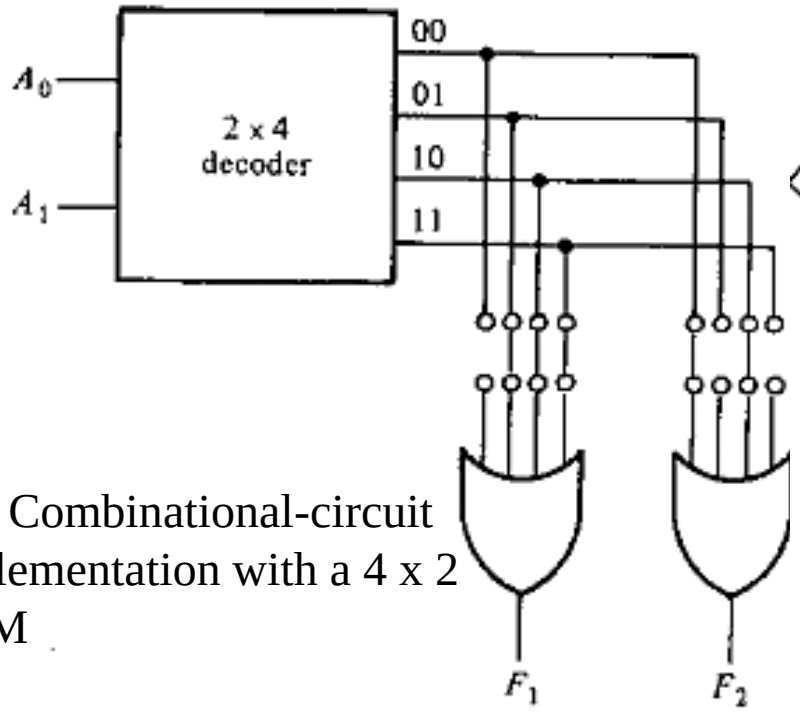


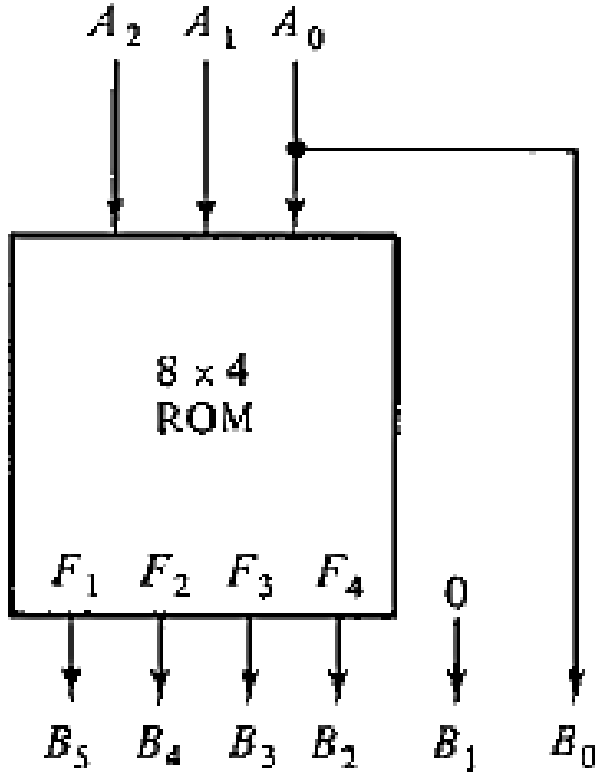
Fig: Combinational-circuit implementation with a 4 x 2 ROM

Diagram shows the internal construction of a 4X2 ROM. It is now necessary to determine which of the eight available fuses must be blown and which should be left intact. This can be easily done from the output functions listed in the truth table. Those minterms that specify an output of 0 should not have a path to the output through the OR gate. Thus, for this particular case, ti1e truth table shows three 0's, and their corresponding fuses to the OR gates must be blown.

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.

The first step is to derive the truth table for the combinational circuit

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



(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

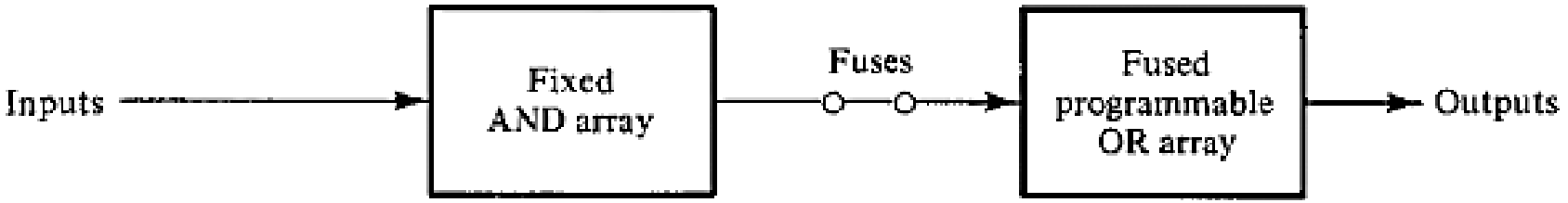
Note: **ROMs**: For simple ROMs, *mask programming* is done by the manufacturer during the fabrication process of the unit. The procedure for fabricating a ROM requires that the customer fill out the truth table the ROM is to satisfy. The truth table may be submitted on a special form provided by the manufacturer. The manufacturer makes the corresponding mask for the paths to produce the 1's and 0's according to the customer's truth table. This procedure is costly because the vendor charges the customer a special fee for custom masking a ROM. For this reason, mask programming is economical only if large quantities of the same ROM configuration are to be manufactured.

Programmable Logic Array (PLA)

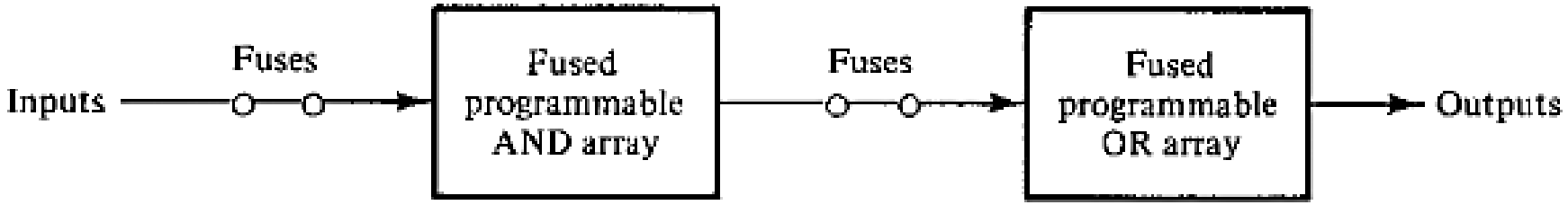
A combinational circuit may occasionally have don't-care conditions. When implemented with a ROM, a don't care condition becomes an address input that will never occur. The words at the don't-care addresses need not be programmed and may be left in their original state. The result is that not all the bit patterns available in the ROM are used, which may be considered a waste of available equipment.

For the cases where the number of don’t-care conditions is excessive, it is more economical to use a second type of LSI component called a *programmable logic array*, or PLA. A PLA is similar to a ROM in concept; however, the PLA does not provide full decoding of the variables and does not generate all the minterms as in the ROM.

In the PLA, the decoder is replaced by a group of AND gates, each of which can be programmed to generate a product term of the input variables. The AND and OR gates inside the PLA are initially fabricated with fuses among them. The specific Boolean functions are implemented in sum of products form by blowing appropriate fuses and leaving the desired connections.



(a) Programmable read-only memory (PROM)



(c) Programmable logic array (PLA)

Difference between ROM and PLA

ROM	PLA
1. ROM generates all the minterms as an output of decoder.	1. PLA does not provide full decoding of the variables.
2. Uses decoder	2. Decoder is replaced by group of AND gates each of which can be programmed to generate a product term of the input variables.
3. The size of the PLA is specified by the number of inputs (n) and the number of outputs (m).	3. The size of the PLA is specified by the number of inputs (n), the number of product terms (k), and the number of outputs (m) (=number of sum terms)
4. No. of programmed fuses = $2^n * m$	4. No. of programmed fuses = $2n * k + k * m + m$

Block Diagram of PLA

A block diagram of the PLA is shown in Fig. below. It consists of *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. Fuses are inserted between all *n* inputs and their complement values to each of the AND gates. Fuses are also provided between the outputs of the AND gates and the inputs of the OR gates.

Another set of fuses in the output inverters allows the output function to be generated either in the AND-OR form or in the AND-OR-INVERT form. With the inverter fuse in place, the inverter is bypassed, giving an AND-OR implementation. With the fuse blown, the inverter becomes part of the circuit and the function is implemented in the AND-OR-INVERT form.

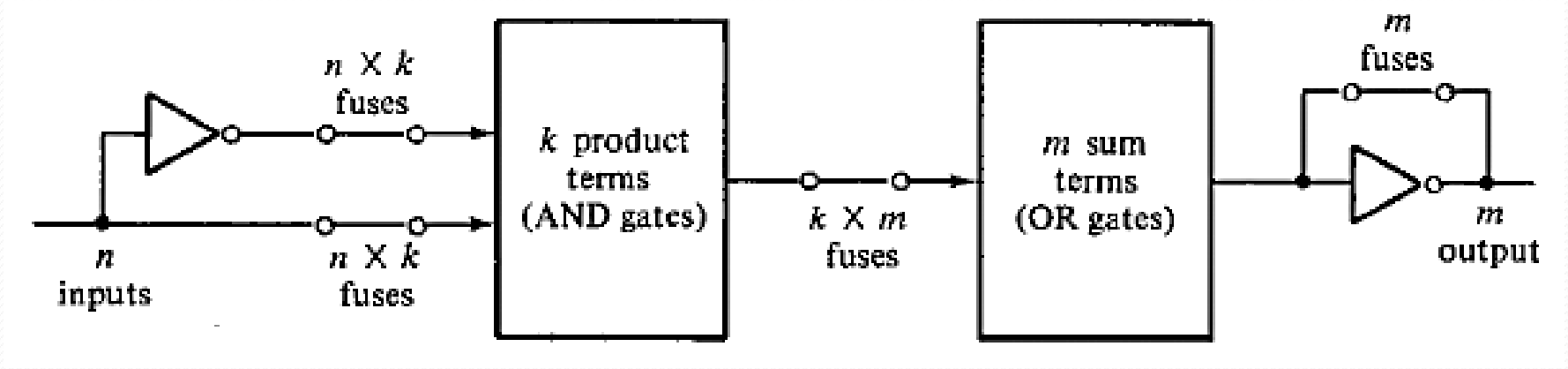


Fig: PLA block diagram

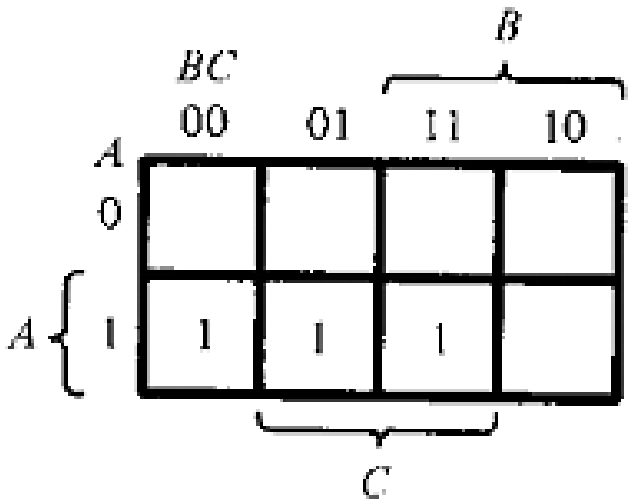
PLA program table and Boolean function Implementation

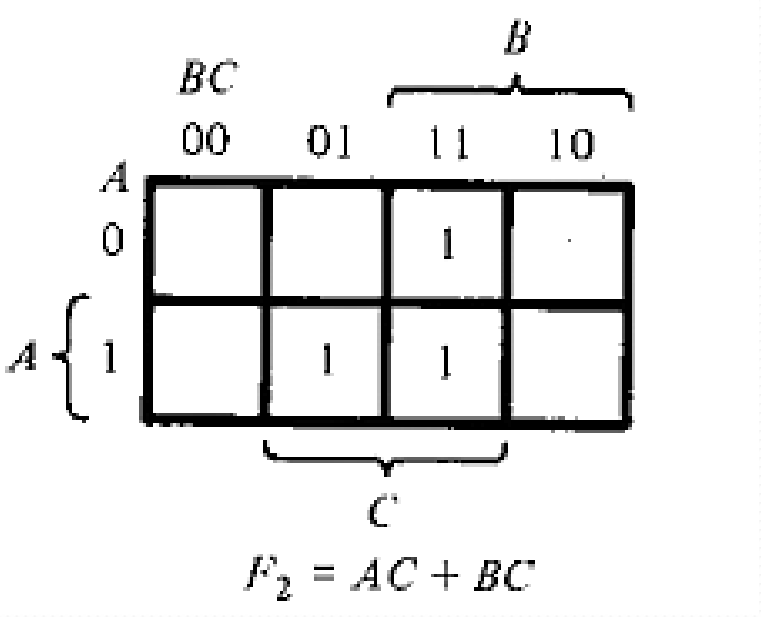
The use of a PLA must be considered for combinational circuits that have a large number of inputs and outputs. It is superior to a ROM for circuits that have a large number of don't-care conditions. Let me explain the example to demonstrate how PLA is programmed.

Consider a truth table of the combinational circuit:

A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

(a) Truth 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

(c) PLA program table

Note:

T (true) output dictates that the fuse across the output inverter remains intact.

C (complement) specifies that the corresponding fuse be blown.

Programming the PLA means, we specify the paths in its AND-OR-NOT pattern. A typical **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.

	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

(c) PLA program table

There are three distinct product terms in this combinational circuit: *AB'*, *AC* and *BC*. The circuit has three inputs and two outputs; so the PLA can be drawn to implement this combinational circuit.