Combinational Logic with MSI and LSI

Contents:

- ✓ Binary Parallel Adder.
- ✓ BCD Adder.
- ✓ Magnitude Comparator.
- ✓ Decoders.
- ✓ Encoders.
- ✓ Multiplexer
- ✓ Demultiplexer.

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.

Input Carry	C4	C3	C2	C1
Augend	A4	A3	A2	A1
Addend	B4	B3	B2	B4
Sum	S4	S3	S2	S1
Output Carry	C5	C4	C3	C2

Fig. 1: Addition of two 4-bit numbers

Subscript i	4	3	2	1	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

Fig. 2: Addition of A=1011 and B=0011

Binary Parallel Adder (Cont.)

Figure-3, shows the interconnection of four full-adder (FA) circuits to provide a 4-bit binary parallel adder. The augend bits of A and the addend bits of B are designated by subscript numbers from right to left. The input carry to adder is C1 and the output carry is C5.

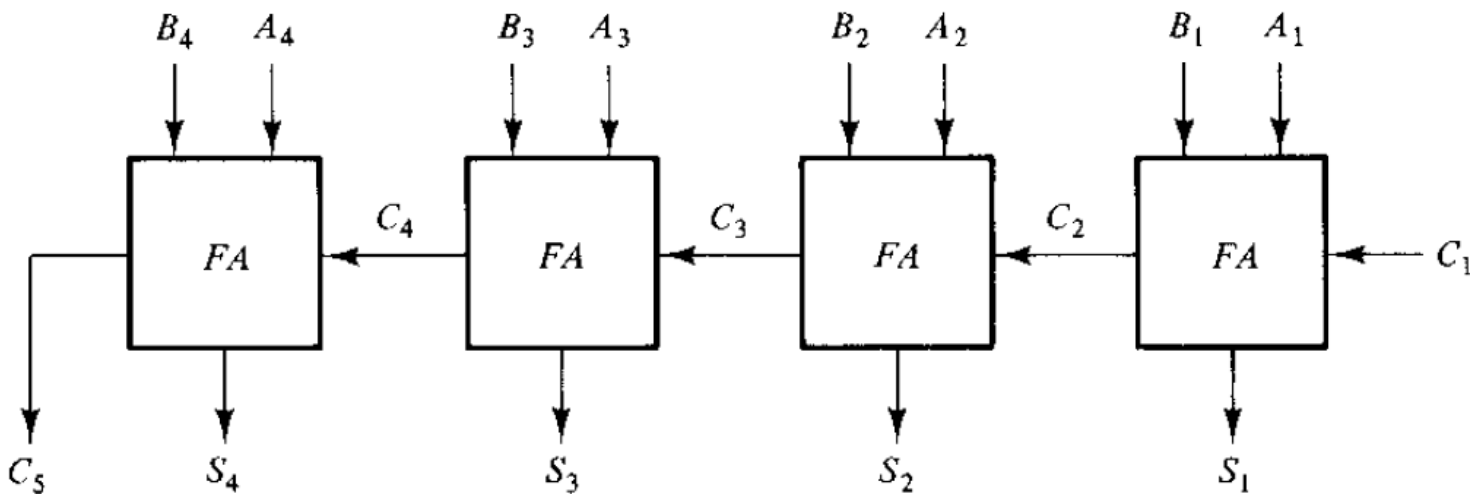
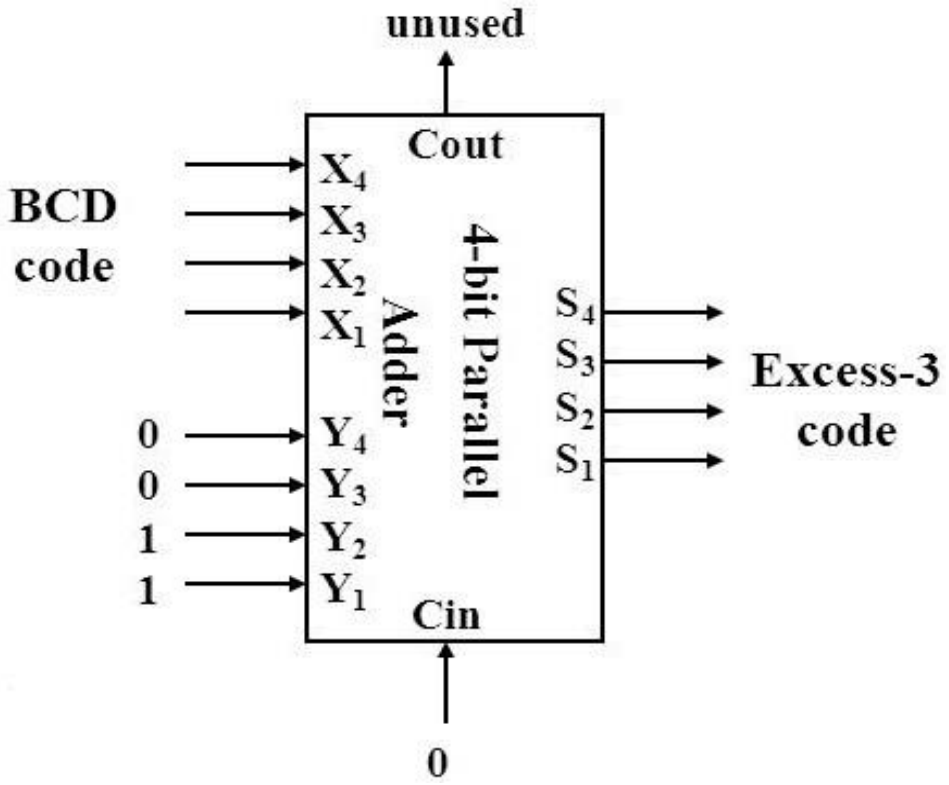


Fig. 3: 4-bit full adders

Binary Parallel Adder (Cont.)

Example 5-1: Design a BCD-to-excess-3 code converter using a 4-bit full adders MSI circuit.

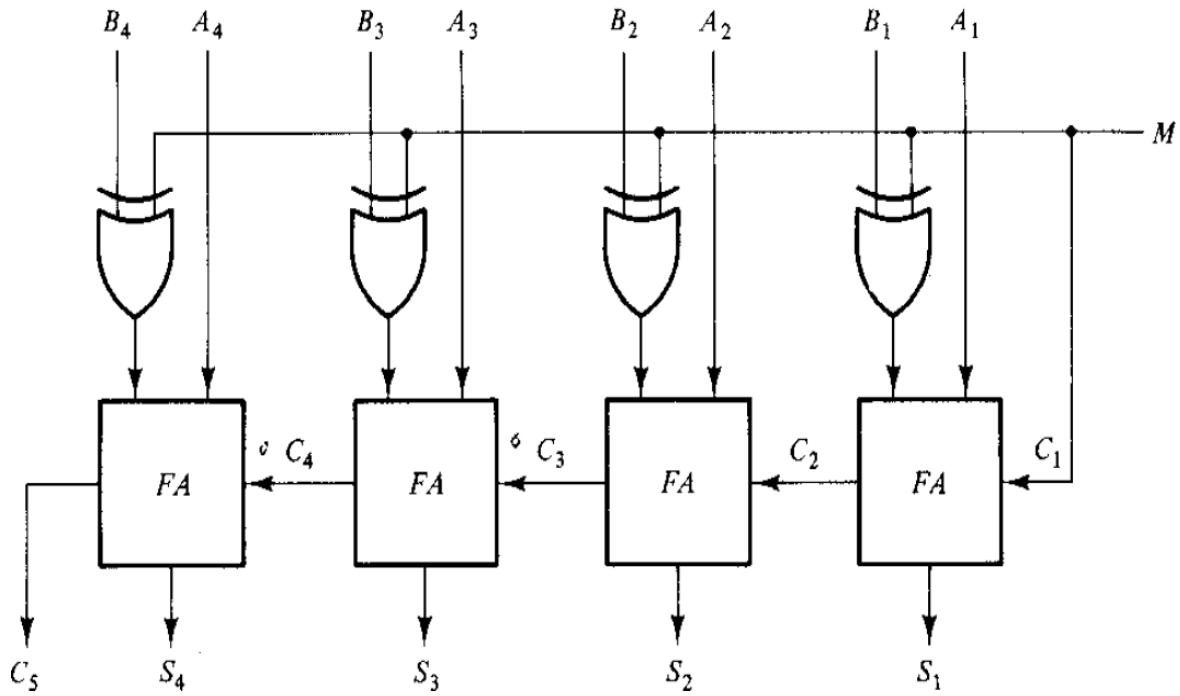
Excess-3 code = BCD Code + $(0011)_2$



Problem 5-1: Design an excess-3-to-BCD code converter using a 4-bit full adders MSI circuit
Hints: BCD code = Excess-3 Code – $(0011)_2$ => BCD Code = Excess-3 Code + $(1101)_2$

Binary Parallel Adder (Cont.)

Problem 5-3: Using 4 exclusive-OR gates and a full-adders MSI circuit, construct a 4-bit parallel adder/subtractor. Use an input select variable M so that when M=0, the circuits adds and when M=1, the circuit subtracts.



Analysis:
If M=1, then
A+ (1's complement of B)+1 appears as the result
If M=0, then
A+B appears as the result.

Fig. 4: 4-bit parallel adder/subtractor

Binary Parallel Adder (Cont.)

Carry Propagation:

- Since each bit of the sum output depends on the value of the input carry, the value of Si in any given stage in the adder will be in its steady-state final value only after the input carry to that stage has been propagated.
- Consider output **S4**,inputs **A4** and **B4** reach a steady value as soon as input signals are applied to the adder. But input carry **C4** does not settle to its final steady-state value until **C3** is available in its steady-state value.Similarly, **C3** has to wait for **C2**, and soon down to **C1**.Thus only after carry propagates through all stages will be the last output **S4** and carry **C5** settle to their final steady-state value.

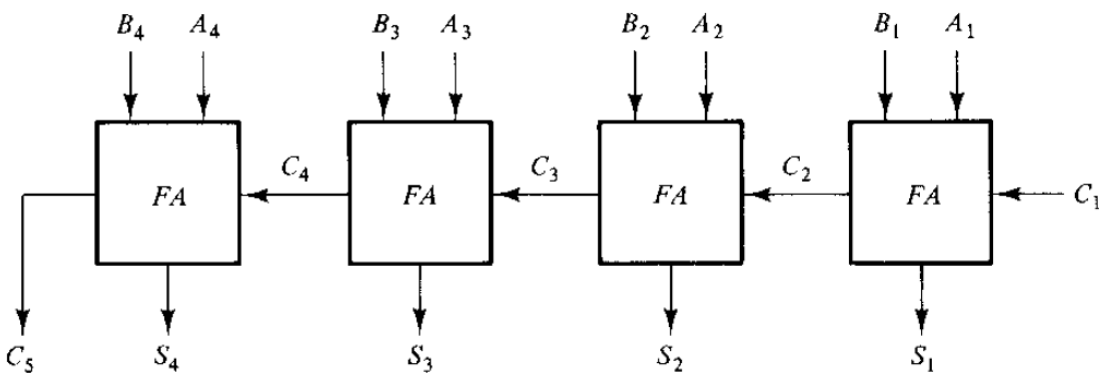


Fig. 5: 4-bit parallel adder

Binary Parallel Adder (Cont.)

Carry Propagation (Cont….):

- The number of gate levels for the carry propagation can be found from the circuit of the full-adder.

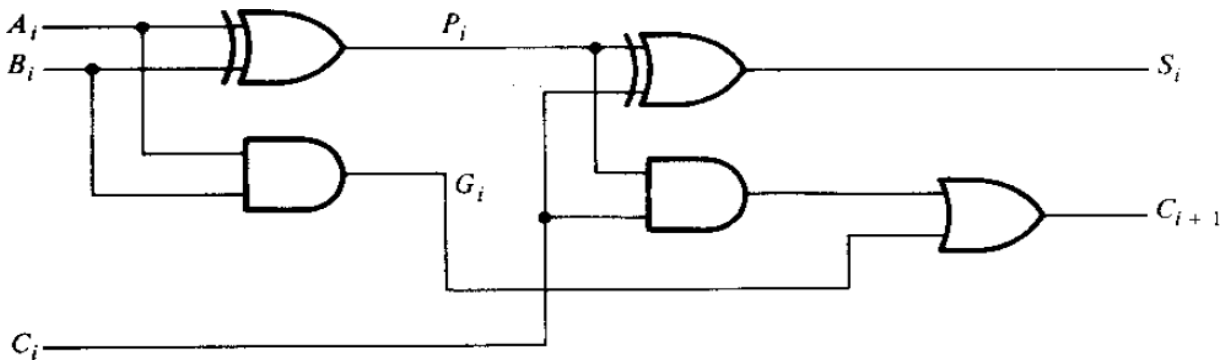


Fig. 6: Full adder

- The signals at **Pi** and **Gi** settle to their steady-state value after the propagation through the irrespective gates. These two signals are common to all FAs and depend only on the input augend and addend bits.
- The signals from the input carry C_i, to the output carry C_{i+1} propagates through an AND and an OR gate, which constitute two gate levels.
- For an n-bit parallel adder, there are 2n gate levels for the carry to propagate through.

Binary Parallel Adder (Cont.)

Carry Propagation (Cont….):

- There are several techniques for reducing the carry propagation time in aparallel adder. The most widely used technique employs the principle of Look-ahead carry.
- From previous circuit (Fig. 6), we can define two new variables,

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

The output sum and carry can be expressed as:

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

- Gi is called a **carry generate** and it produces an output carry when both Ai and Bi are one, regardless of the input carry. Pi is called a **carry propagate** because it is the term associated with the propagation of the carry from C_i to C_{i+1}
- We now write the Boolean function for the carry output of each stage:

$$C_2 = G_1 + P_1 C_1$$

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

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

Binary Parallel Adder (Cont.)

Carry Propagation (Cont….): Note that C_4 does not have to wait for C_3 and C_2 to propagate, in fact, C_4 is propagated at the same time as C_2 and C_3.

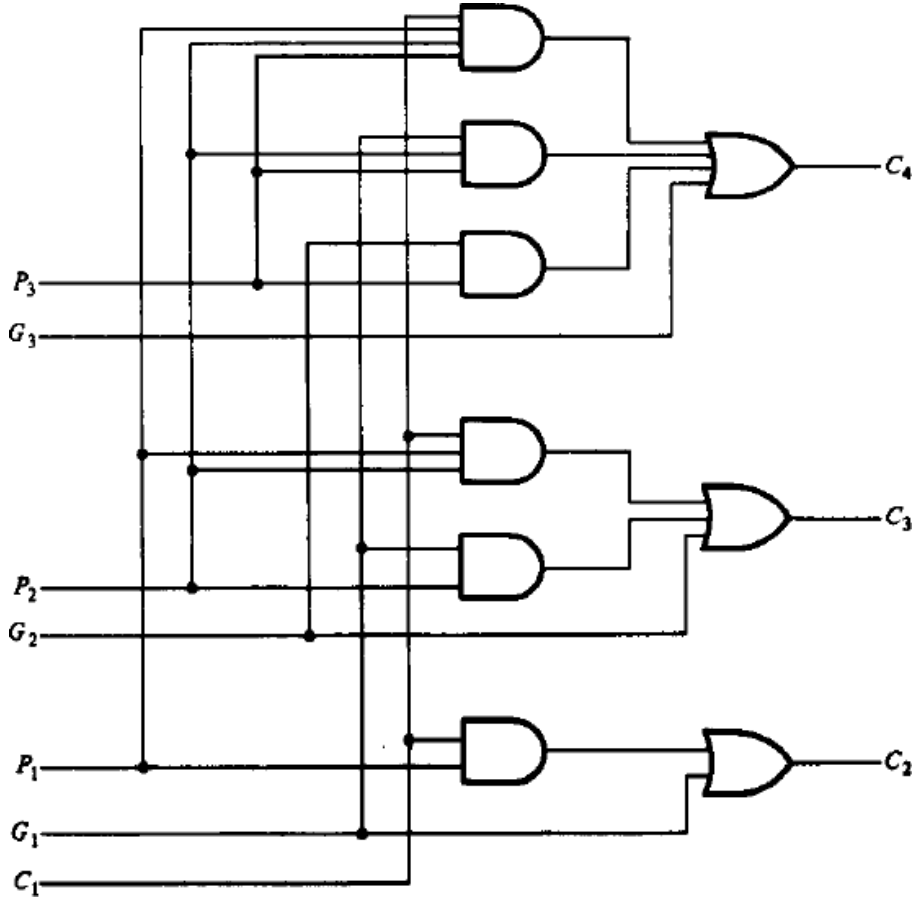


Fig. 7: Logic Diagram of a look-ahead carry generator

Binary Parallel Adder (Cont.)

Carry Propagation (Cont….):

- The construction of a 4-bit parallel adder with a look-ahead carry scheme is shown in Fig. 8.
- A typical look-ahead carry generator is the IC type 74128

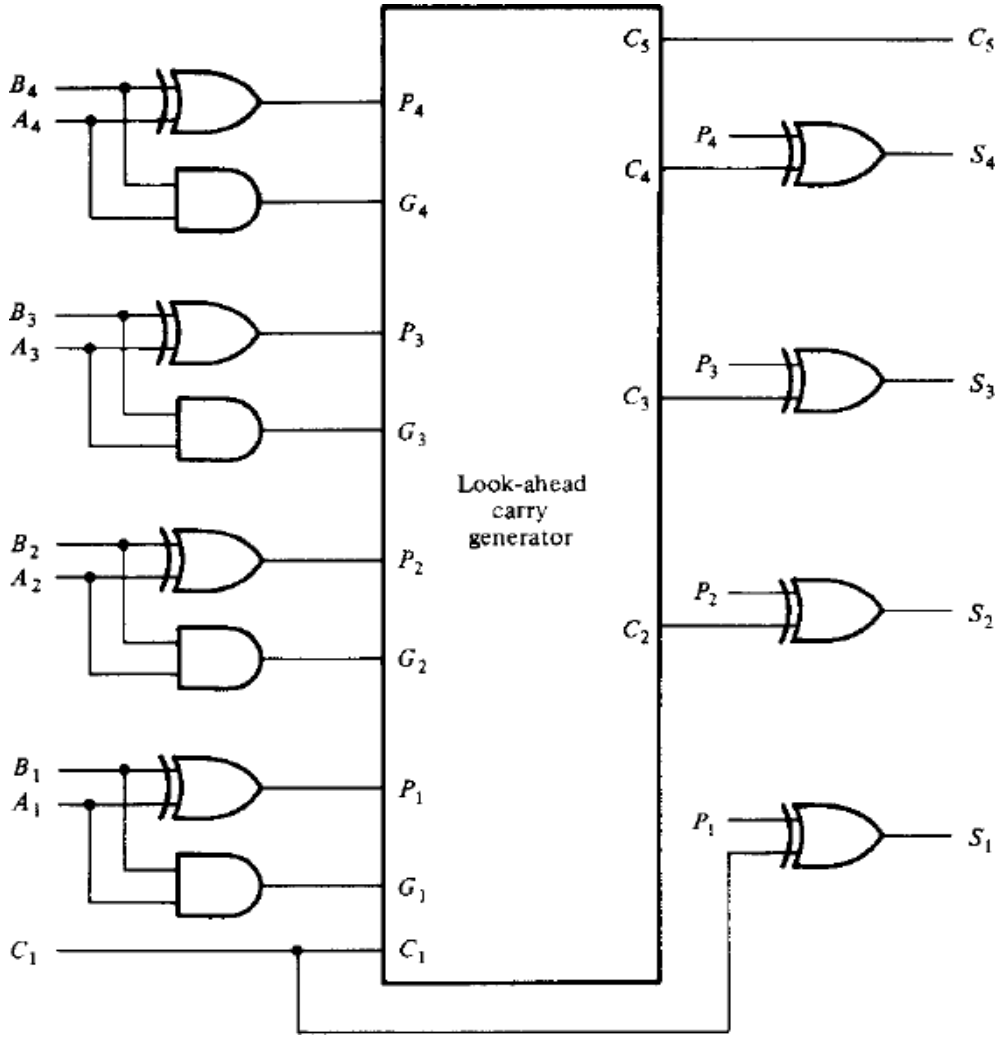


Fig. 8: 4-bit adder with look-ahead carry generator

BCD Adder

- A BCD adder is a circuit that adds two BCD digits in parallel and produces a sum digital so in BCD.
- Consider the arithmetic addition of two decimal digits in BCD, together with a previous carry from a previous stage. The output sum cannot 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 to19.

BCD Adder

- In examining the content of the table, it is apparent that when the binary sum is equal to or less than1001,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.
- The condition for a correction and an output carry can be expressed by the Boolean function:

$$C = K + Z_8Z_4 + Z_8Z_2$$

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

BCD Adder

C=0, when binary sum is less or equal to 1001. In this case binary sum and BCD sum are same. So, nothing is added to the binary sum.

C=1, when binary sum is greater than 1001. binary 0110 is added to the binary sum through the bottom 4-bit binary adder to convert the binary sum into BCD sum.

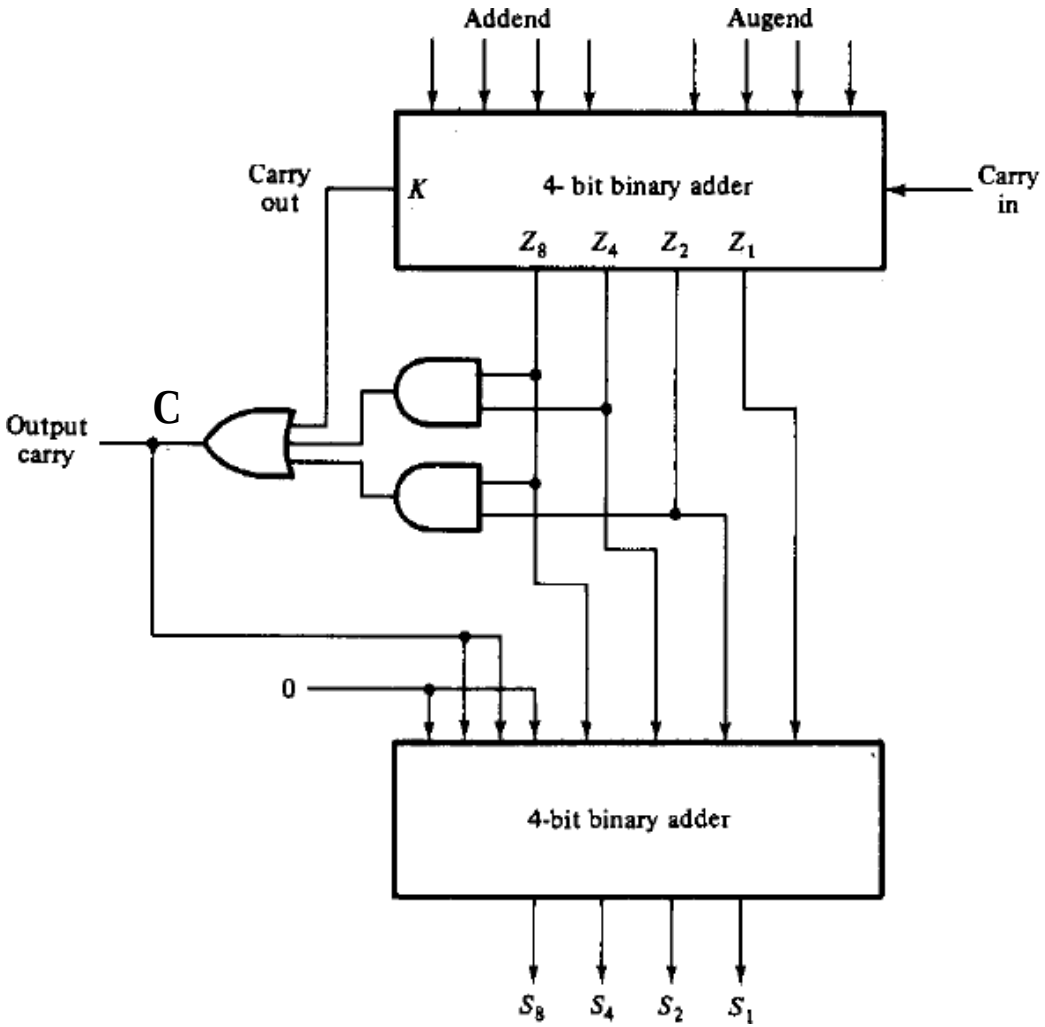


Fig. 9: Block diagram of a BCD adder

Magnitude Comparator

- **A magnitude comparator** is a combinational circuit that compares two numbers A & B and determines the irrelative 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.

$$A = A_3A_2A_1A_0$$

$$B = B_3B_2B_1B_0$$

- The two numbers are equal if all pairs of significant digits are equal i.e. A_3=B_3 and A_2=B_2 and A_1=B_1 and A_0=B_0.
- The equality relation of each pair of bits can be expressed:

$$x_i = A_iB_i + A_i'B_i' \quad , \text{ i=0,1,2,3}$$

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.

Magnitude Comparator (Cont.)

- For equity condition to exist, all x_i variables must be equal to 1.

  (A=B) = $x_3x_2x_1x_0$

A and B will be equal if $x_3 = x_2 = x_1 = x_0 = 1$ or, all pairs of digits of the two numbers are equal.

- To determine if A is greater than or less than B, we inspect the relative magnitude of pairs of significant digits starting from 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 unique digits is reached. If the digit of A is 1 and that of B is 0, we conclude A>B. If the corresponding digit of A is 0 and that of B is 1, we have that A<B.

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

Magnitude Comparator (Cont.)

Analysis

If this term is 1, it means A3=1 and B3=0. As in most significant bit, A is greater than B, we conclude A>B.
But, if this term is zero, it means A3=B3.We should check next bit.

x3=1 meansA3=B3.

$A_2B'_2$ =1, indicates A2>B2. If x3=1, we conclude A>B
$A_2B'_2$ =0, indicates A2=B2.We should check next bit to take decision.

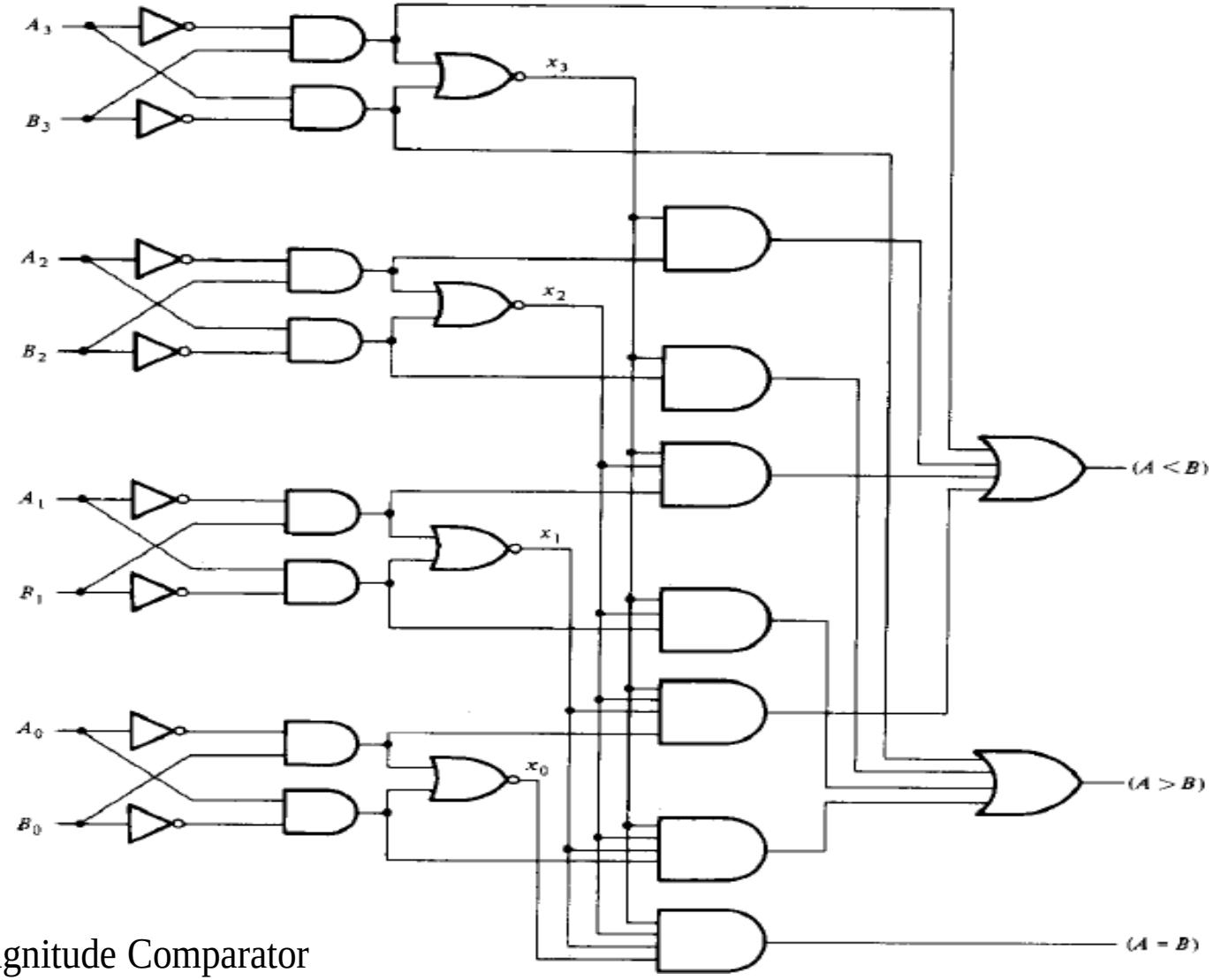
x3 x2=1 means A3=B3 and A2=B2.

$A_1B'_1$ =1, indicates A1>B1. If x3 x2=1, we conclude A>B
$A_2B'_2$ =0, indicates A1=B1.We should check next bit to take decision.

$$(A > B) = A_3B'_3 + x_3A_2B'_2 + x_3x_2A_1B'_1 + x_3x_2x_1A_0B'_0$$

x3 x2 x1=1 means A3=B3, A2=B2 and A1=B1.
$A_0B'_0$ =1, indicates A0>B0. If x3 x2 x1=1, we conclude A>B
$A_0B'_0$ =0, indicates A0=B0. All bits are equal. i.e. $x_3x_2x_1x_0$=1.

Magnitude Comparator (cont.)

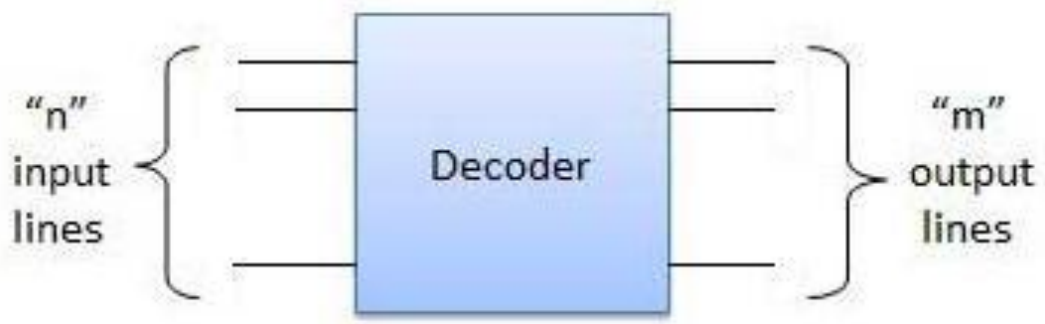


4-bit magnitude Comparator

Decoders

Definition:

- **A decoder** is a combinational circuit that converts binary information from **n input lines** to a maximum of **2^n unique output lines**.
- If n-bit decoded information has unused or don't-care combinations, the decoder output will have less than 2^n outputs.
- The decoders presented here are called n-to-m line decoders where $m \leq 2^n$. Their purpose is to generate the 2^n (or less) minterms of n input variables.



Decoders

Application:

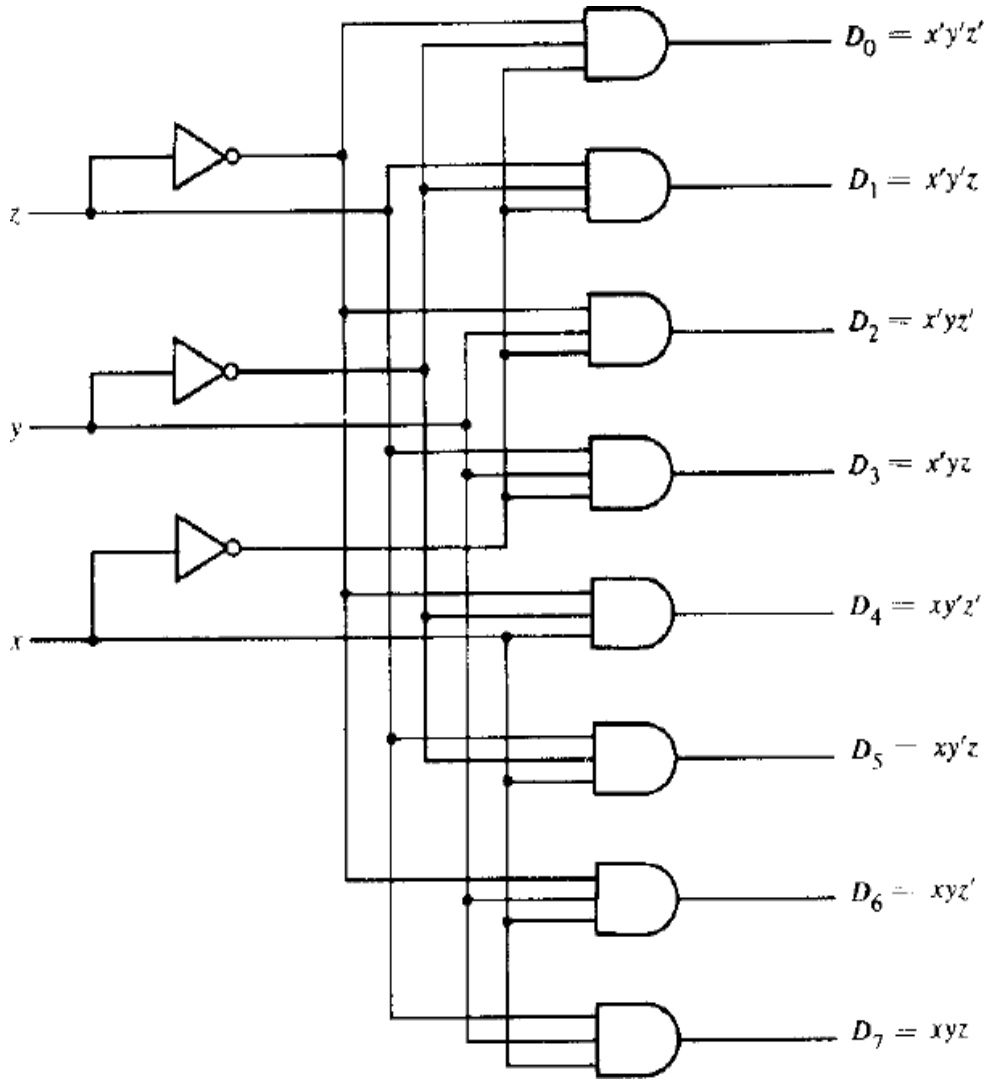
Decoders are greatly used in applications where the particular output or group of outputs to be activated only on the occurrence of a specific combination of input levels. Some important application of decoder circuit is given below-

- **Address Decoders:** Amongst its many uses, a decoder is widely used to decode the particular memory location in the computer memory system. Decoders accept the address code generated by the CPU which is a combination of address bits for a specific location in the memory.
- **Instruction Decoder:** Another application of the decoder can be found in the control unit of the central processing unit. This decoder is used to decode the program instructions in order to activate the specific control lines such that different operations in the ALU of the CPU are carried out.

Decoders

3 X 8 Decoder

- The three inputs are decoded into eight outputs, **each output representing one of the minterms** of the 3-input variables.
- A particular application of this decoder would be a **binary-to-octal conversion**. The input variable may represent a binary number and the outputs will then represent the eight digits in the octal number system



Decoders

Truth Table of 3 X 8 line decoder

From the truth table it is observed that the output variables are mutually exclusive because only one output can be equal to 1 at any one time. The output line whose value is equal to 1 represents the minterm equivalent of the binary number presently available in the input lines.

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

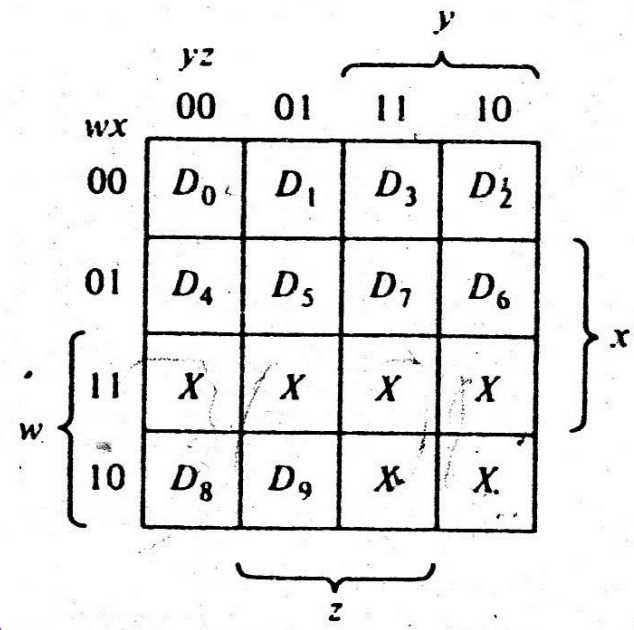
Decoders

Example 5-2: **Design a BCD-to-decimal decoder**

Input BCD				Output Decimal									
w	x	y	z	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1

Decoders

- Since the circuit has ten outputs, it would be necessary to draw ten maps to simplify each one of the output functions, Instead of drawing ten maps, we will draw only one map and write each of the output variables, D0 to d9, inside its corresponding minterm square.
- Six input combinations will never occur, so we mark them as don't care.



- **D0 and D1 is isolated, can't be grouped with don't care.**

  D2 to D7 form pair with their adjacent don't care.

  D8 and D9 form Quad with their adjacent don't care

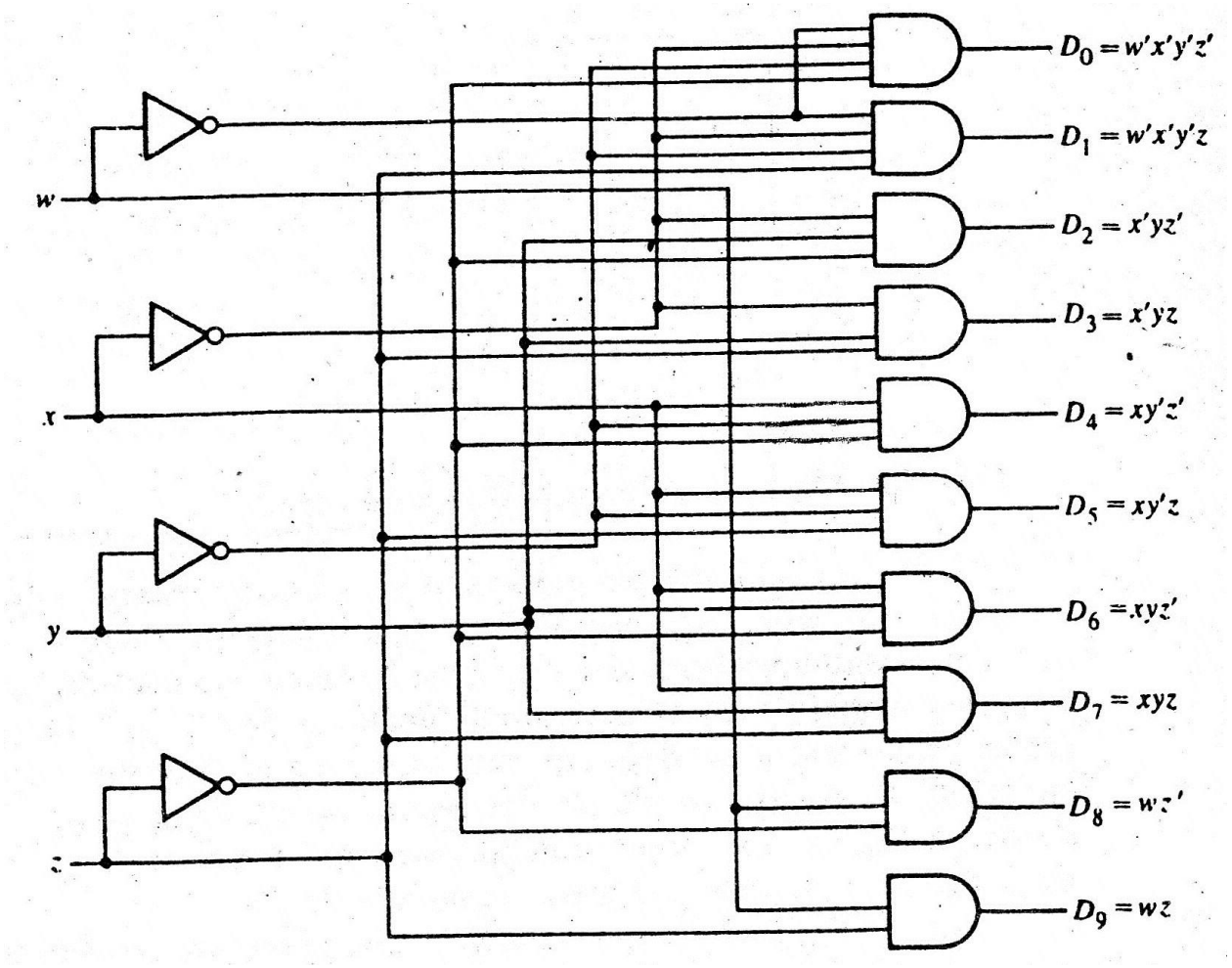
- **Finally we get**

$D_0 = w'x'y'z'$ $D_1 = w'x'y'z$ $D_2 = x'yz'$ $D_3 = x'yz$ $D_4 = xy'z'$

$D_5 = xy'z$ $D_6 = xyz'$ $D_7 = xyz$ $D_8 = wz'$ $D_9 = wz$

Decoders

Logic diagram of BCD-to-decimal decoder

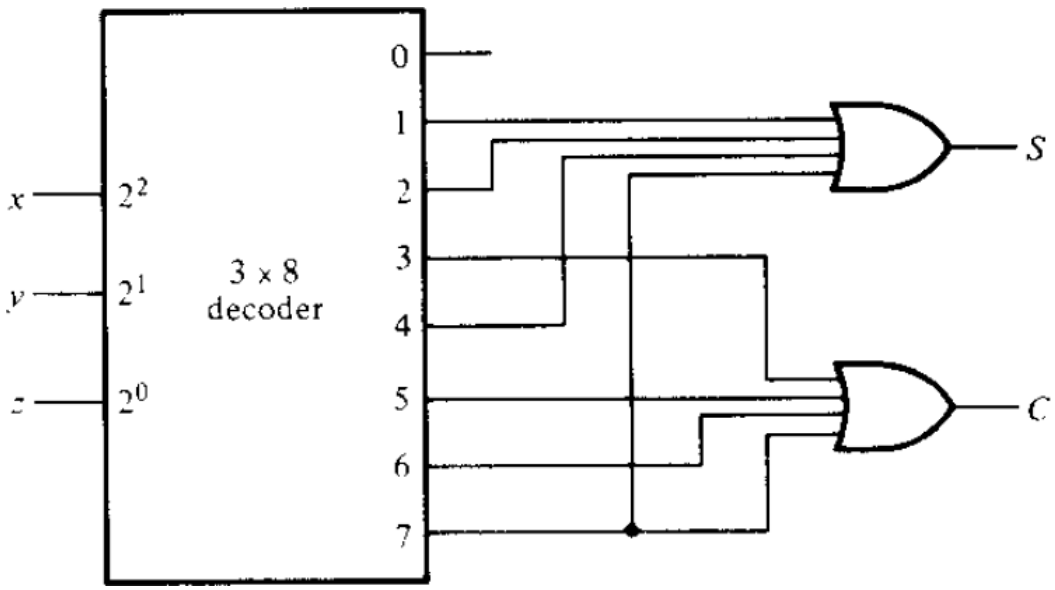


Decoders

Example 5-3: Implement a full-adder circuit with a decoder and two OR gates

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

x	y	z	C	S
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



$$S = \sum(1,2,4,7)$$

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

Encoders

An encoder is a digital function that produces a reverse operation from that of a decoder. An encoder has **2^n input lines and n output lines**.

The output lines generate the binary code corresponding to the input value.

Example: Octal to binary encoder which has 8 inputs and 3 outputs

Truth Table of binary to octal 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

From the truth table:

- Output bit z is 1 if octal digit is odd.
- Output y is 1 for octal digits 2,3,6 or 7.
- Output x is 1 for octal digits 4,5,6, 0r 7.

Encoders

Boolean function of output variables

$$x = D_4 + D_5 + D_6 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$z = D_1 + D_3 + D_5 + D_7$$

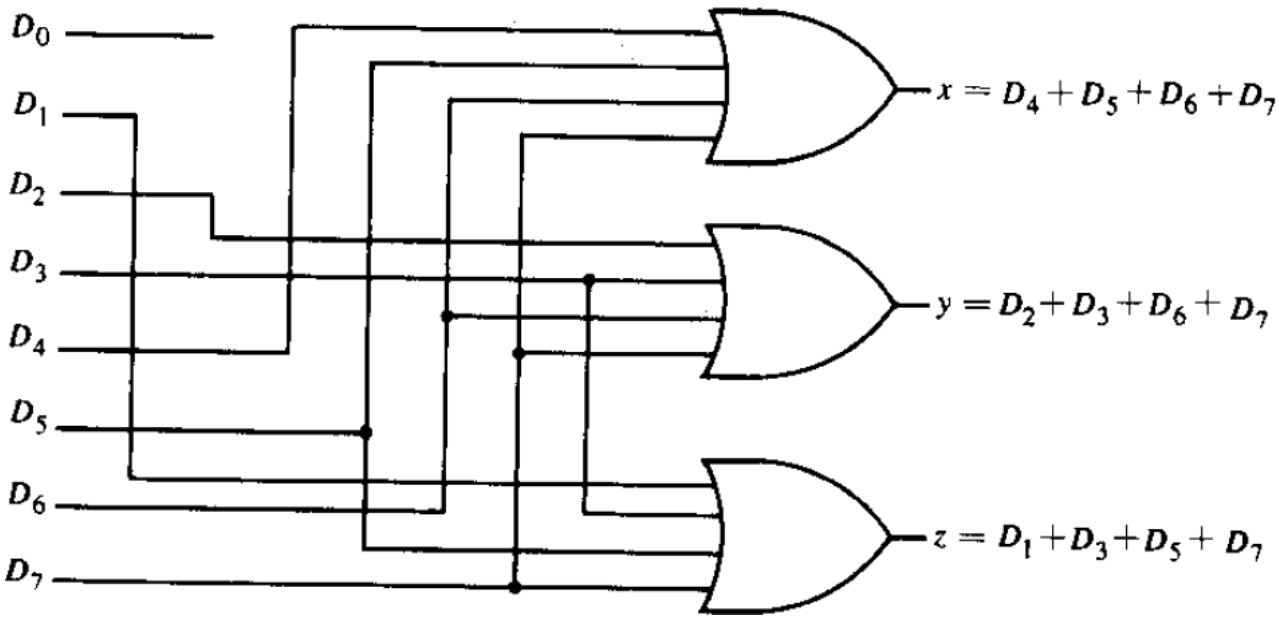
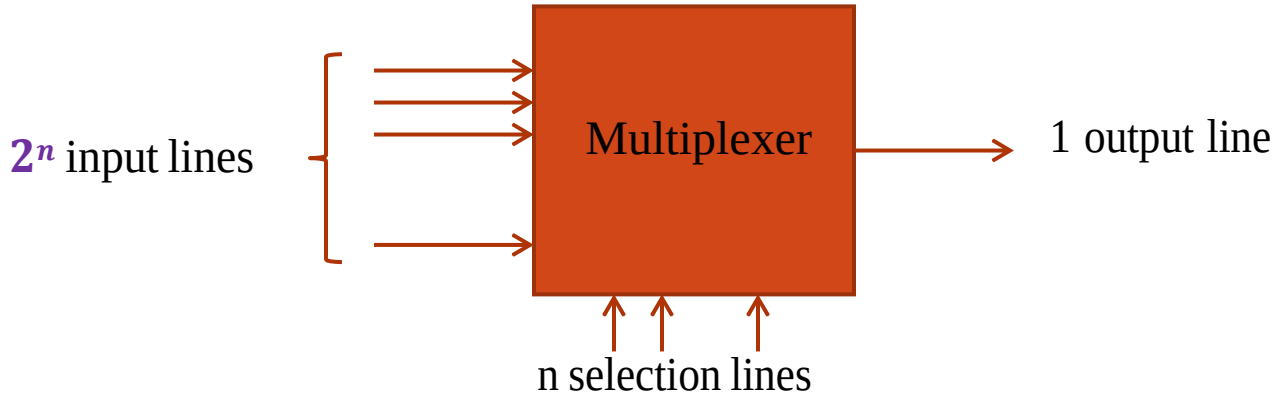


Fig. Octal to binary encoder

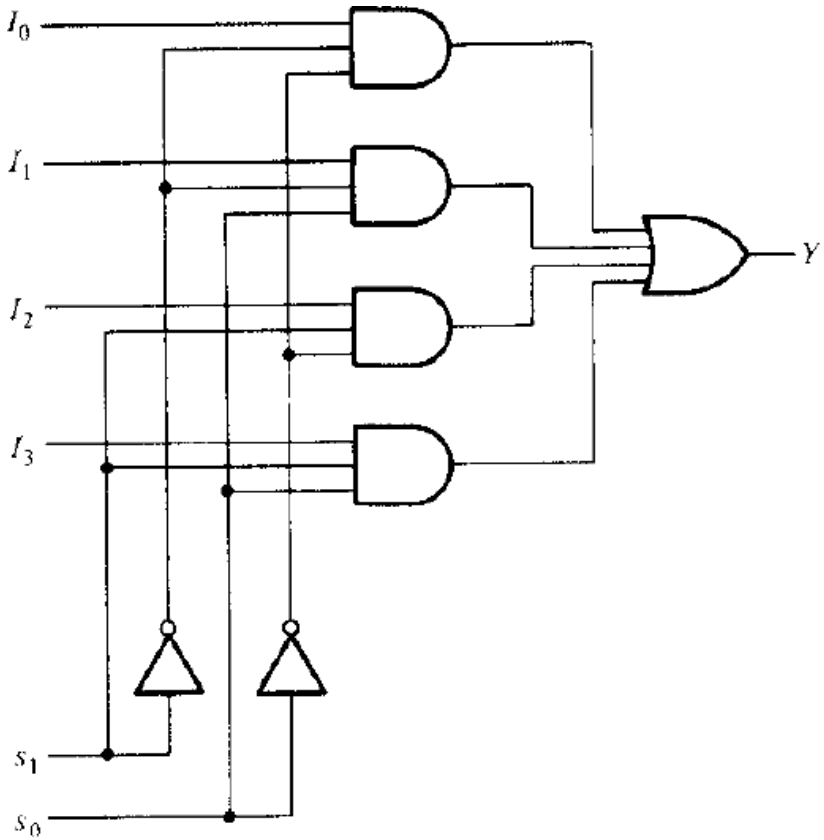
Multiplexer

- Multiplexer means transmitting a large number of information units over a smaller number of channels or lines.
- A digital multiplexer is 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 lines 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.
- A multiplexer is also called a data selector, since it selects one of many inputs and steers the binary information to the output line.



Multiplexer

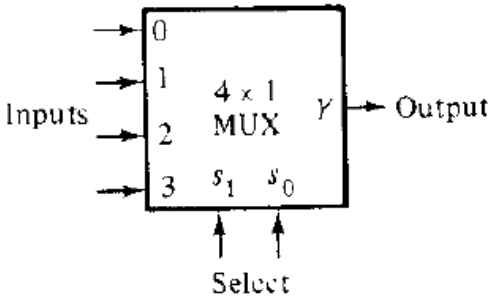
4-line to 1-line multiplexer



(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

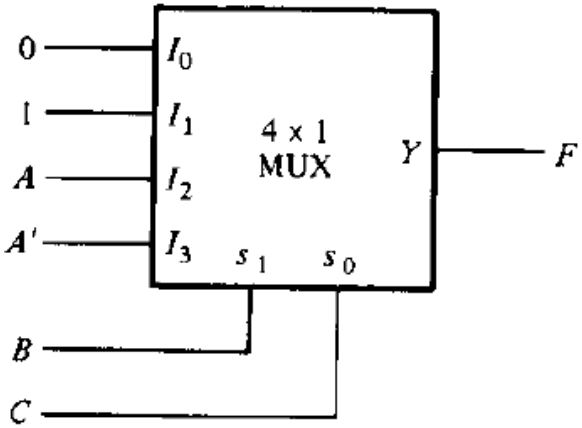
(b) Function table



(c) Block diagram

Multiplexer

Implementation of Boolean function $F(A, B, C) = \sum(1, 3, 5, 6)$ by multiplexer



(a) Multiplexer implementation

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

(b) Truth table

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

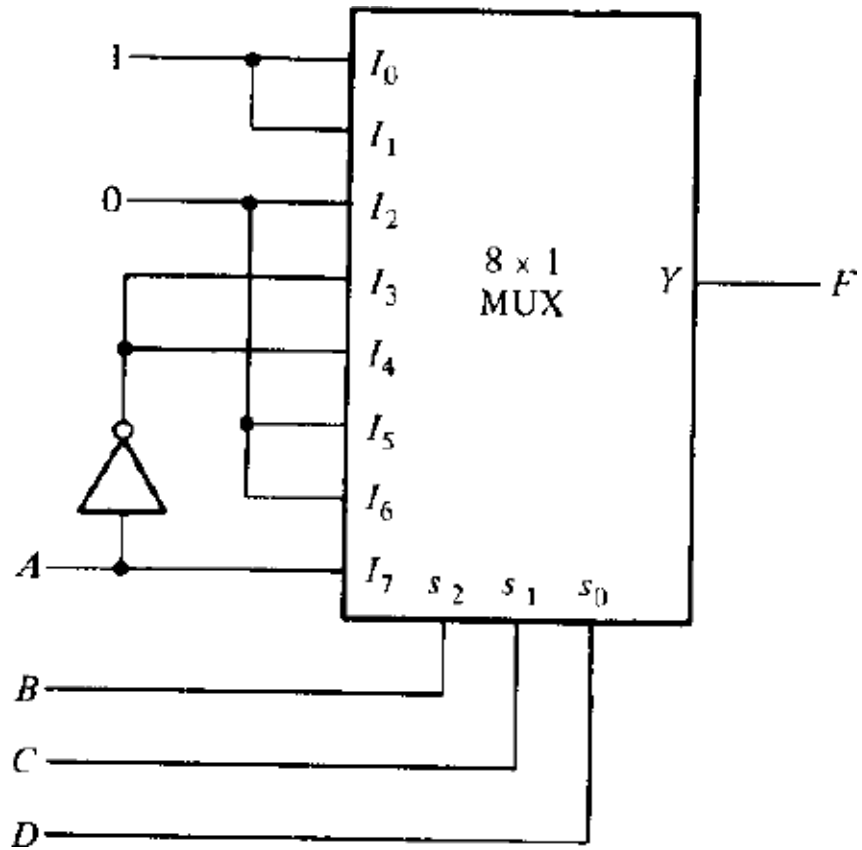
(c) Implementation table

Multiplexer

Example 5-4:

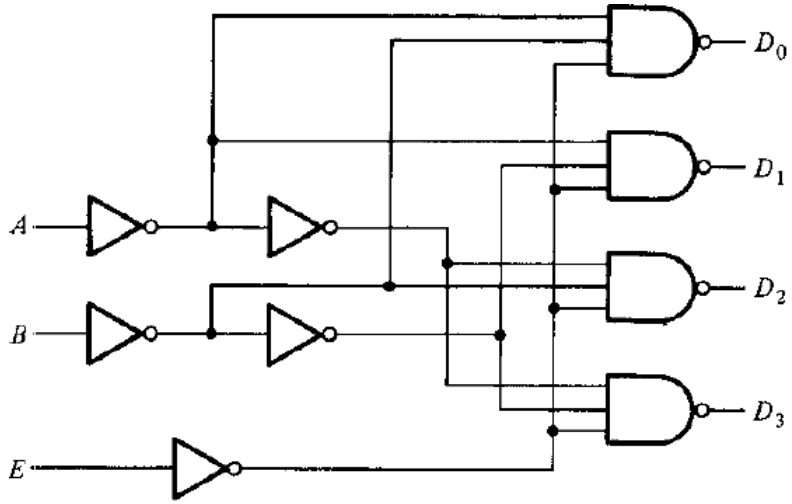
Implementation of Boolean function $F(A, B, C, D) = \sum(0, 1, 3, 4, 8, 9, 15)$ by multiplexer

	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



De-multiplexer

A decoder with an enable input can function as a de-multiplexer.
A de-multiplexer is a circuit that **receives information on a single line** and transmit this information on one of 2^n **possible output lines**.



(a) Logic diagram

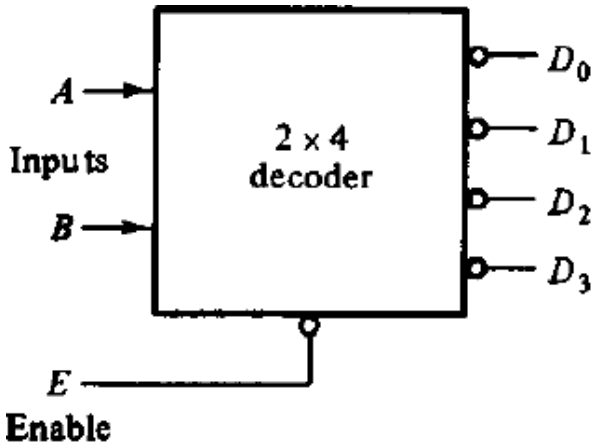
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

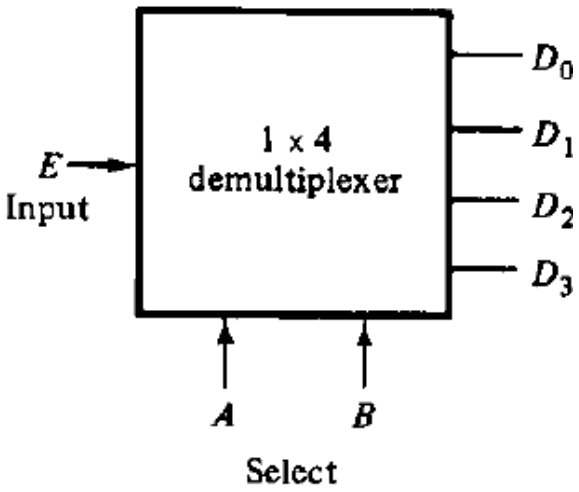
Fig: A 2-to-4 line decoder with enable (E) input

The decoder of fig can function as a de-multiplexer if the E line is taken as a data input line and lines A and B are taken as the selection lines.

De-multiplexer

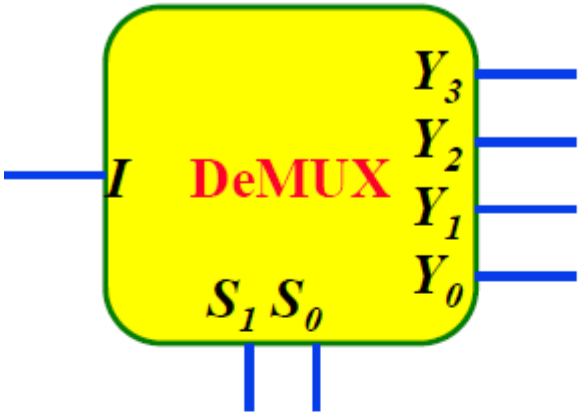


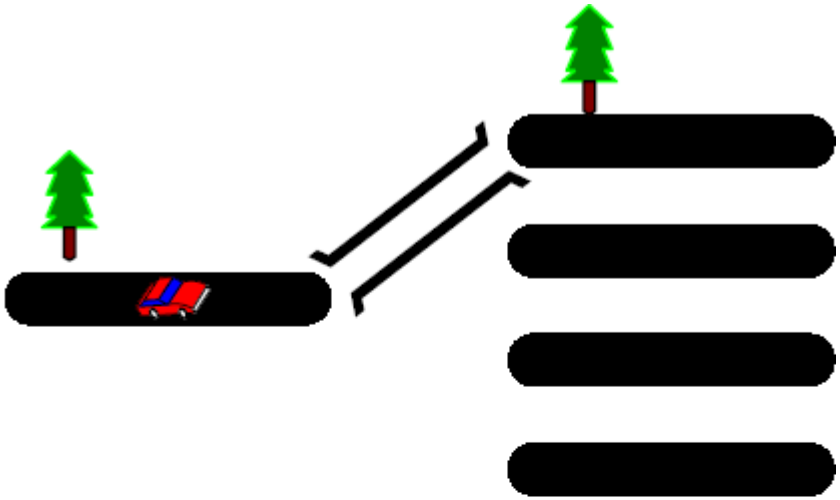
(a) Decoder with enable

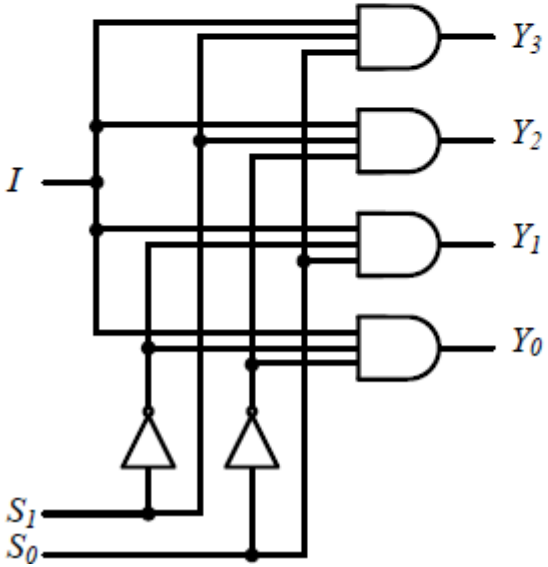


(b) Demultiplexer

De-multiplexer



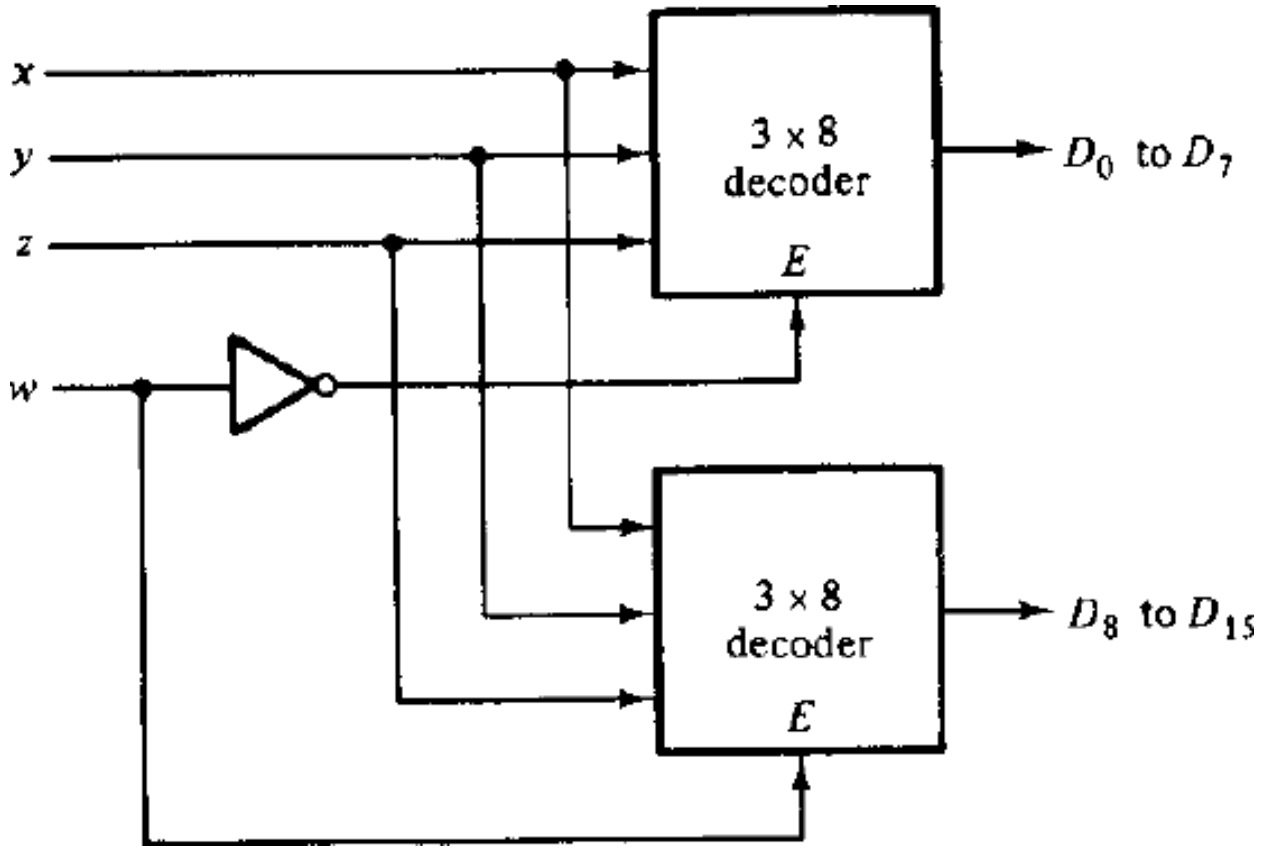




S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

De-multiplexer

Decoder with enable/demultiplexer circuits can be connected together to form a larger decoder circuit. Fig. shows two 3X8 decoders with enable inputs connected to form a 4X16 decoder.

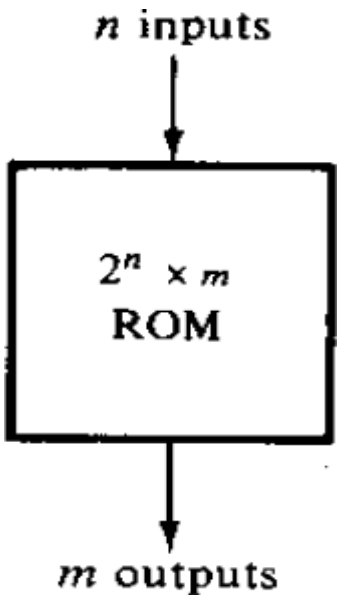


Read-Only Memory (ROM)

- A decoder generates the 2^n minterms of the n input variable. By inserting OR gates to sum the minterms of Boolean functions, we are able to generate any desired combinational circuit.
- **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 links that 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.

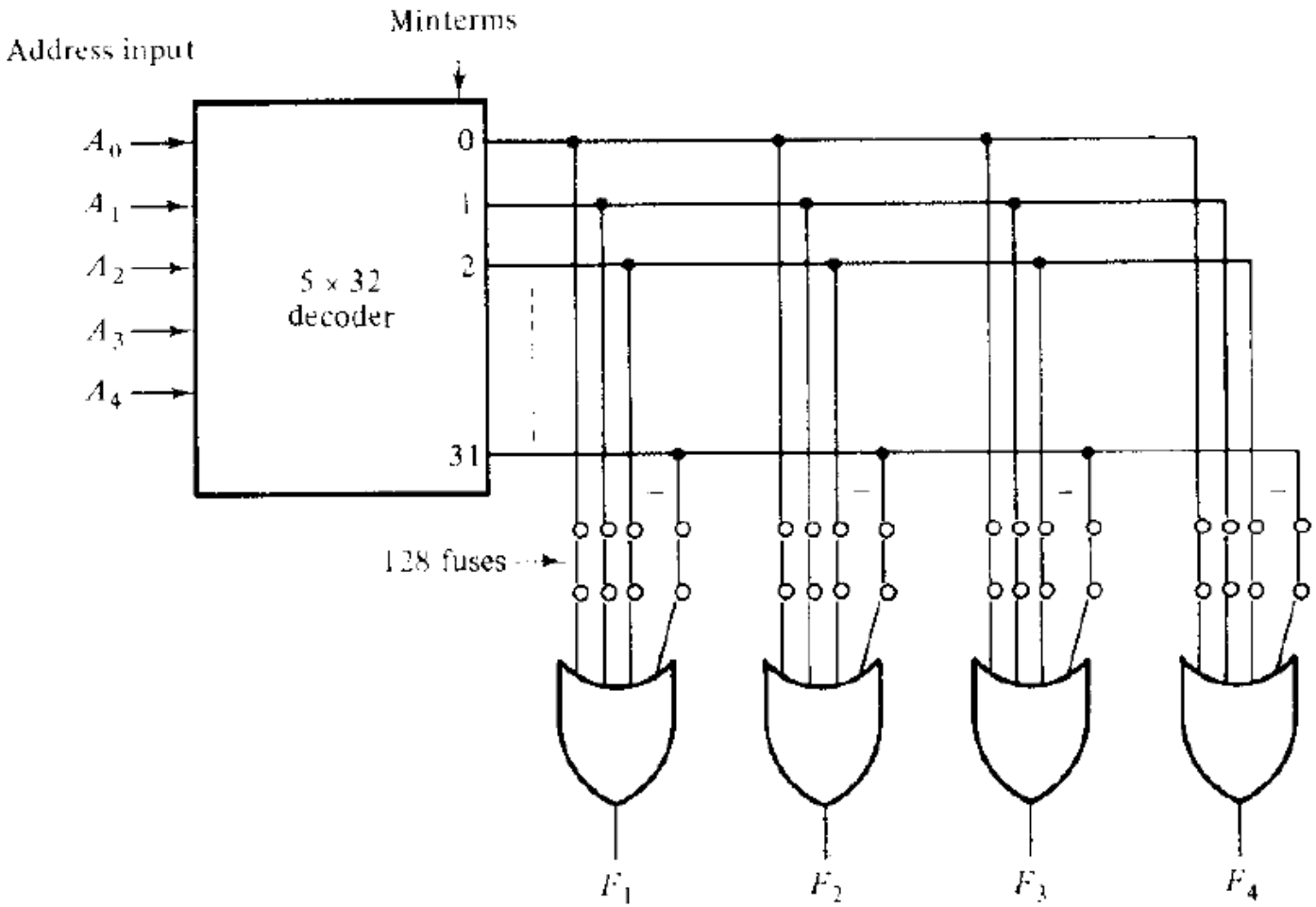
Read-Only Memory (ROM)

- 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^n distinct addresses, so there are 2^n distinct words which are said to be stored in the unit.



Read-Only Memory (ROM)

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. Figure shows the logic construction of a 32X4 ROM.



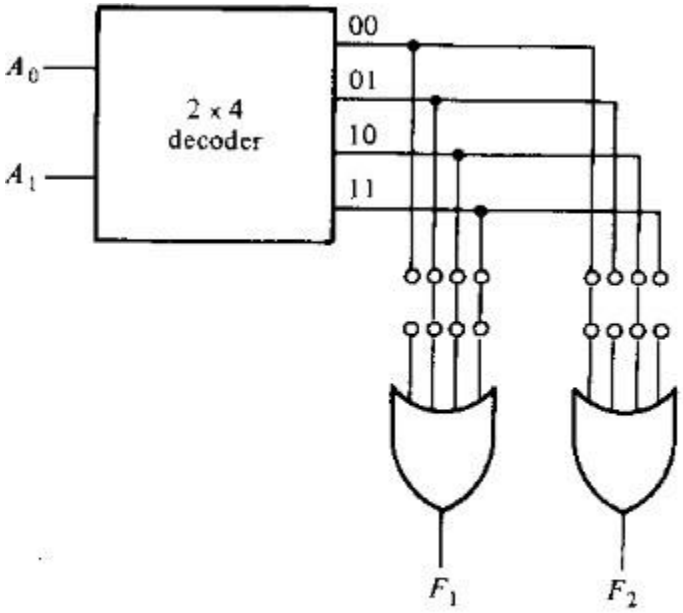
Read-Only Memory (ROM)

Combinational Logic Implementation

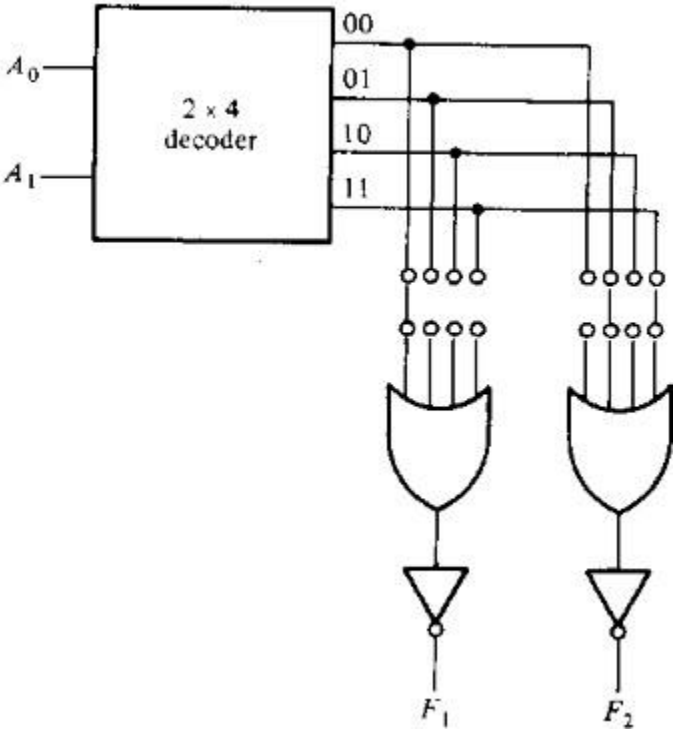
Implement the following combinational circuit with a 4X2 ROM

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

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



ROM with AND-OR gates



ROM with AND-OR-INVERT gates

Read-Only Memory (ROM)

Combinational Logic Implementation

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.

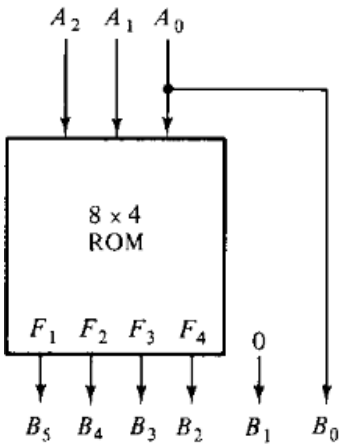
Solution:

Output B0 is always equal to input A0; so there is no need to generate B0 with a ROM since it is equal to an input variable. Moreover, output B1 is always 0, so this outputs is always known.

Truth Table

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

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

Programmable Logic Array (PLA)

Why PLA?

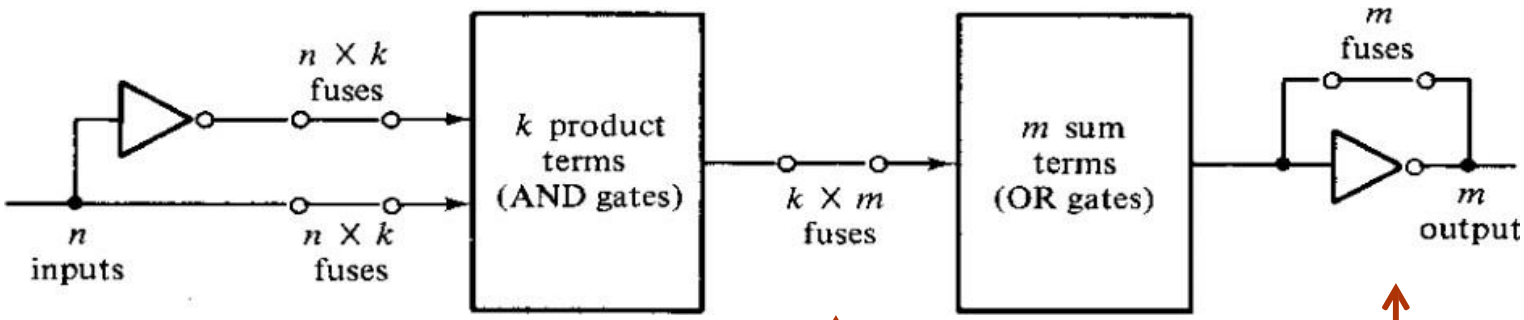
- A combinational circuit may occasionally have don't care conditions. When implemented with a ROM, a don't care condition becomes an address input that will never occur. The words at the don't care addresses need not be programmed and may be left in their original state (all 0's or all 1's). The result 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 X 6 (2^{12} X 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, **for cases where the number of don't care conditions is excessive,** it is more economical to use a second type of LSI component called Programmable Logic Array (PLA).

Programmable Logic Array (PLA)

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

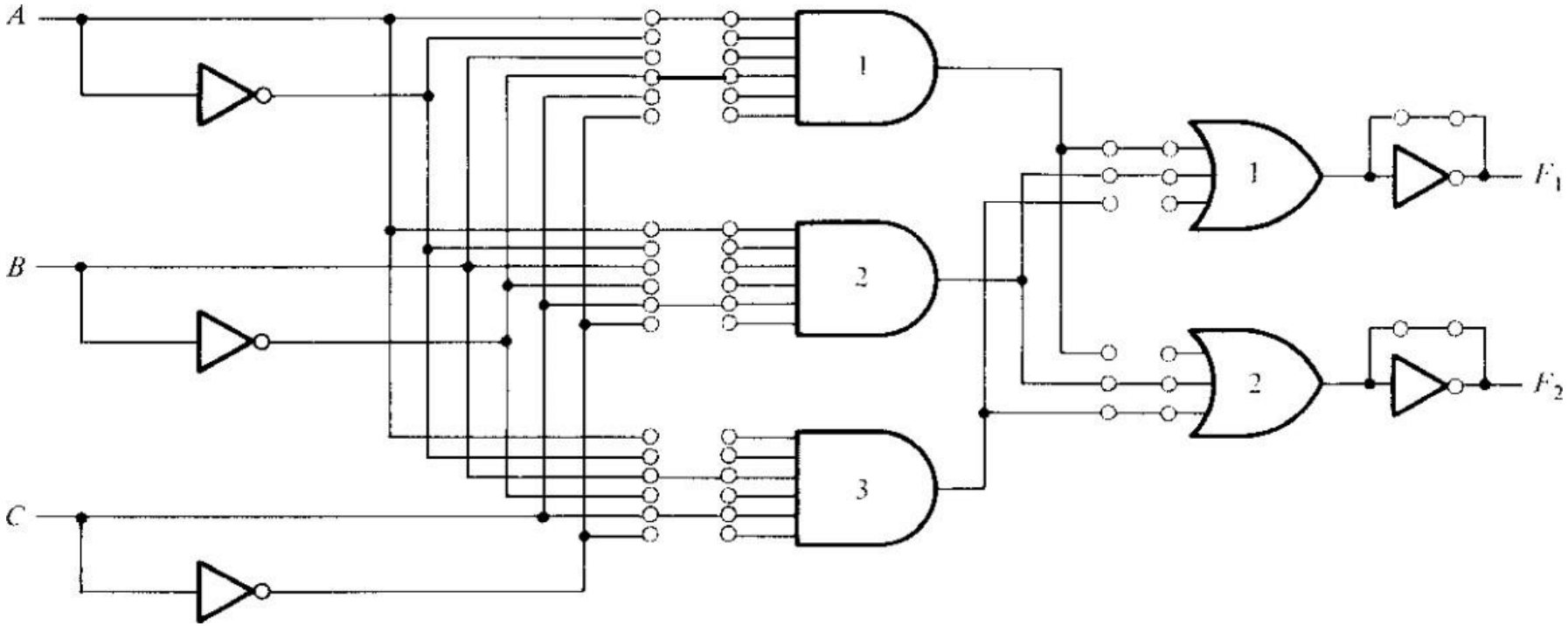
- The number of programmed links is 2n X k + k X m +m, whereas that of a ROM is $2^n X\ m$

Programmable Logic Array (PLA)

Implementation of combinational circuit by PLA.

$$F_1 = AB' + AC$$

$$F_2 = AC + BC$$



Programmable Logic Array (PLA)

PLA Types.

- PLA may be mask-programmable or field programmable.
- With a **mask programmable PLA**, the customer must submit a PLA program table to the manufacturer. This table is used by the vendor to produce a custom-made PLA that has the required internal paths between inputs and outputs.
- A second type of PLA available is called **field programmable logic array or FPLA**. The FPLA can be programmed by the user by means of certain recommended procedure. Commercial hardware programmable units are available for use in conjunction with certain FPLAs