UNIT-III: Combinational Logic Design

Combinational Circuits

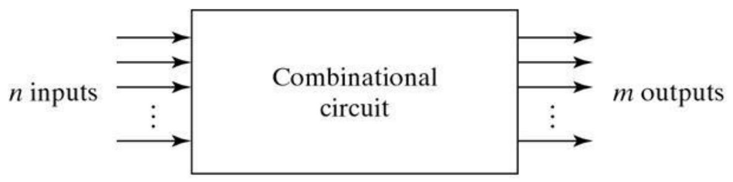


Fig. Block Diagram of Combinational Circuit

Logic circuits for digital systems may be combinational or sequential. The output of a combinational circuit depends on its present inputs only. Combinational circuits perform a specific information processing operation fully specified logically by a set of Boolean functions. A combinational circuit consists of input variables, logic gates and output variables. Both input and output data are represented by signals, i.e., they exists in two possible values. One is logic–1 and the other logic-0.

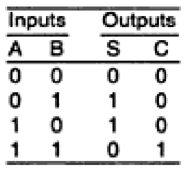
For n input variables, there are 2n possible combinations of binary input variables. For each possible input Combination, there is one and only one possible output combination. A combinational circuit can be described by m Boolean functions one for each output variables. Usually the inputs come from flip-flops and outputs go to flip-flops.

Design Procedure:

1.The problem is stated
2. The number of available input variables and required output variables is determined. 3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationship between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

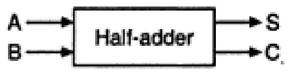
Adders:

Digital computers perform variety of information processing tasks, the one is arithmetic operations. And the most basic arithmetic operation is the addition of two binary digits. i.e, 4 basic possible operations are: 0+0=0, 0+1=1, 1+0=1, 1+1=10
The first three operations produce a sum whose length is one digit, but when augends and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a carry. A combinational circuit that performs the addition of two bits is called a **half- adder**. One that performs the addition of 3 bits (two significant bits & previous carry) is called a **full adder** & 2 half adder can employ as a full-adder.



Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table



(b) Block diagram

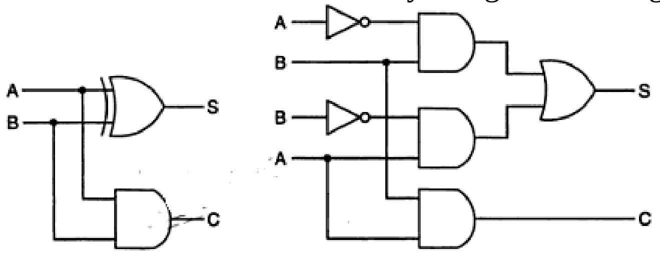
The Half Adder: A Half Adder is a combinational circuit with two binary inputs (augends and addend bits and two binary outputs (sum and carry bits.) It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits. It is an arithmetic operation of addition of two single bit words (i.e) Number of inputs=2(A,B) and Number of outputs=2(sum and carry).

The Sum(S) bit and the carry (C) bit, according to the rules of binary addition, the sum (S) is the X-OR of A and B (It represents the LSB of the sum). Therefore,

S= AB#+A#B = $A \oplus B$

The carry (C) is the AND of A and B (it is 0 unless both the inputs are 1).Therefore, C=AB

A half-adder can be realized by using one X-OR gate and one AND gate as

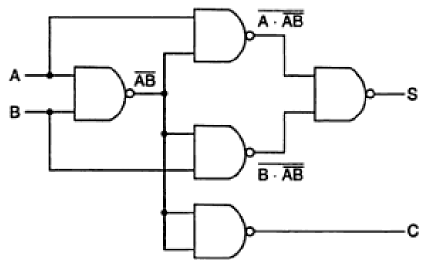


(a) (b)

Fig: Logic diagrams of half-adder

NAND Logic:

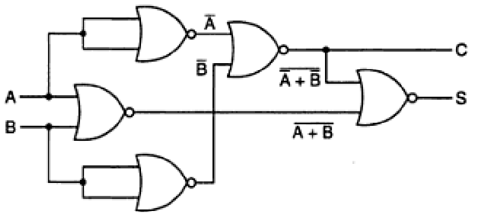
$$S = A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B}$$
$$= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$$
$$= A \cdot \overline{AB} + B \cdot \overline{AB}$$
$$= \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$
$$C = AB = \overline{\overline{AB}}$$


A
B
$\overline{AB}$
$\overline{A \cdot \overline{AB}}$
$\overline{B \cdot \overline{AB}}$
S
C

Logic diagram of a half-adder using only 2-input NAND gates.

NOR Logic:

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



Logic diagram of a half-adder using only 2-input NOR gates.

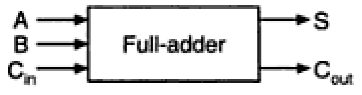
The Full Adder:

A Full-adder is a combinational circuit that adds two bits and a carry and, outputs a sum bit and a carry bit. To add two binary numbers, each having two or more bits, the LSBs can be added by using a half-adder. The carry resulted from the addition of the LSBs is carried over to the next significant column and added to the two bits in that column. So, in the second and higher columns, the two data bits of that column and the carry bit generated from the addition in the previous column need to be added.

The full-adder adds the bits A and B and the carry from the previous column called the carry-in Cin and outputs the sum bit S and the carry bit called the carry-out Cout . The variable S gives the value of the least significant bit of the sum. The variable Cout gives the output carry. The eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have. The 1s and 0s for the output variables are determined from the arithmetic sum of the input bits. When all the bits are 0s, the output is 0. The S output is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1. The Cout has a carry of 1 if two or three inputs are equal to 1.(i.e) Number of inputs=2(A,B,C) and Number of outputs=2(sum and carry)

Inputs			Sum	Carry
A	B	C_{in}	S	C_{out}
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

(a) Truth table



(b) Block diagram

Full-adder.

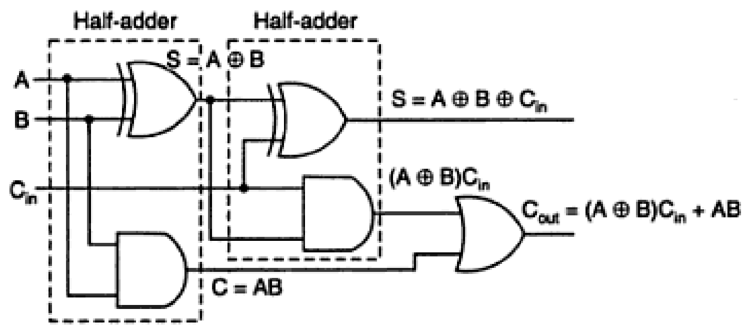
From the truth table, a circuit that will produce the correct sum and carry bits in response to every possible combination of A,B and Cin is described by

$$S \# ABC_{in} \# ABC_{in} \# ABC_{in} \# ABC_{in} = C_{in}(A\# B\#+AB)+ C\#_{in}(A\#B+AB\#)$$
$$C_{out} \# ABC_{in} \# ABC_{in} \# ABC_{in} \# ABC_{in} = C_{in}(A \# B)+AB$$

and

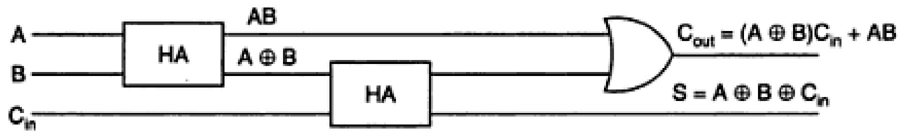
$S \# A \# B \# C_{in}$

$C_{out} \# C_{in}(A \# B)+AB$



Logic diagram of a full-adder using two half-adders.

The block diagram of a full-adder using two half-adders is


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

Block diagram of a full-adder using two half-adders.

The sum term of the full-adder is the X-OR of A,B, and Cin, i.e, the sum bit the modulo sum of the data bits in that column and the carry from the previous column. The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e, Two half adders) and one OR gate is

Even though a full-adder can be constructed using two half-adders, the disadvantage is that the bits must propagate through several gates in accession, which makes the total propagation delay greater than that of the full-adder circuit using AOI logic.

$$A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

Then

$$S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) \cdot \overline{(A \oplus B)C_{in}}} \cdot \overline{C_{in} \cdot \overline{(A \oplus B)C_{in}}}}$$

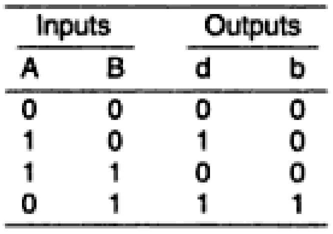
The Full-adder neither can also be realized using universal logic, i.e., either only NAND gates or only NOR gates as

Subtractors:

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this, the subtraction operation becomes an addition operation and instead of having a separate circuit for subtraction, the adder itself can be used to perform subtraction. This results in reduction of hardware. In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position, that has been borrowed must be conveyed to the next higher pair of bits by means of a signal coming out (output) of a given stage and going into (input) the next higher stage.

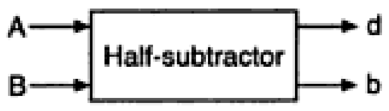
The Half-Subtractor:

A Half-subtractor is a combinational circuit with two inputs A and B and two outputs d and b. d indicates the difference and b is the output signal generated that informs the next stage that a 1 has been borrowed. When a bit B is subtracted from another bit A, a difference bit (d) and a borrow bit (b) result according to the rules (i.e) Number of inputs=2(A, B) and Number of outputs=2(difference and borrow)given as



Inputs		Outputs	
A	B	d	b
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

(a) Truth table



(b) Block diagram

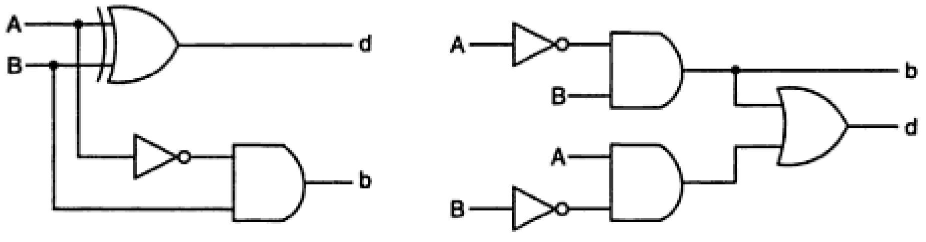
Half-subtractor.

The output borrow b is a 0 as long as A#B. It is a 1 for A=0 and B=1.

A circuit that produces the correct difference and borrow bits in response to every possible combination of the two 1-bit numbers is,

d=A#B+AB#= $A \oplus B$ and b=A#B

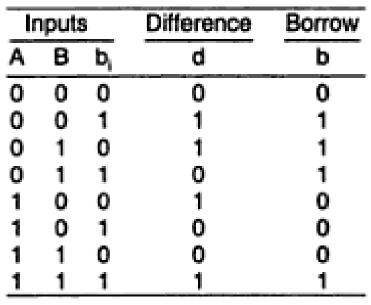
That is, the difference bit is obtained by X-OR ing the two inputs, and the borrow bit is obtained by ANDing the complement of the minuend with the subtrahend.



Logic diagrams of a half-subtractor.

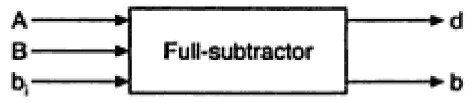
The Full-Subtractor:

The half-subtractor can be only for LSB subtraction. If there is a borrow during the subtraction of the LSBs, it affects the subtraction in the next higher column; the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column. Such a subtraction is performed by a full-subtractor. It subtracts one bit (B) from another bit (A), when already there is a borrow bi from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit (b) required from the next d and b. The operation for input is A-B-bi. The two outputs present the difference and output borrow. (i.e) Number of inputs=3(A,B,bi) and Number of outputs=2(d,b)



Inputs			Difference	Borrow
A	B	b_i	d	b
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

(a) Truth table



(b) Block diagram

Full-subtractor.

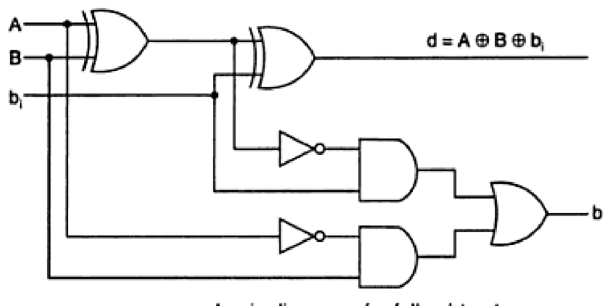
From the truth table, a circuit that will produce the correct difference and borrow bits in response to every possible combinations of A,B and bi is

$$\begin{aligned} d &= \bar{A}\bar{B}b_i + \bar{A}B\,\bar{b}_i + A\bar{B}\,\bar{b}_i + ABb_i \\ &= b_i(AB + \bar{A}\bar{B}) + \bar{b}_i(A\bar{B} + \bar{A}B) \\ &= b_i(\overline{A \oplus B}) + \bar{b}_i(A \oplus B) = A \oplus B \oplus b_i \end{aligned}$$

and

$$\begin{aligned} b &= \bar{A}\bar{B}b_i + \bar{A}B\,\bar{b}_i + \bar{A}Bb_i + ABb_i = \bar{A}B(b_i + \bar{b}_i) + (AB + \bar{A}\bar{B})b_i \\ &= \bar{A}B + (\overline{A \oplus B})b_i \end{aligned}$$

A full-subtractor can be realized using X-OR gates and AOI gates as

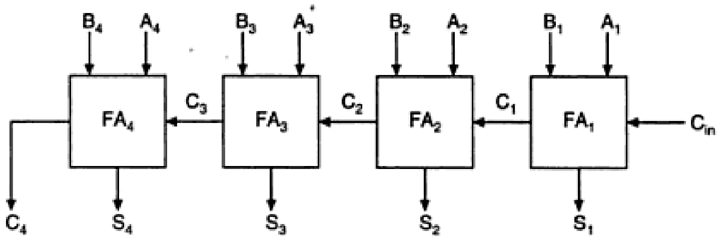


Logic diagram of a full-subtractor.

Binary Parallel Adder:

A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form. 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.

The interconnection of four full-adder (FA) circuits to provide a 4-bit parallel adder. When the 4-bit full-adder circuit is enclosed within an IC package, it has four terminals for the augends bits, four terminals for the addend bits, four terminals for the sum bits, and two terminals for the input and output carries. An n-bit parallel adder requires n-full adders. The output carry from one package must be connected to the input carry of the one with the next higher –order bits. The 4-bit full adder is a typical example of an MSI function.



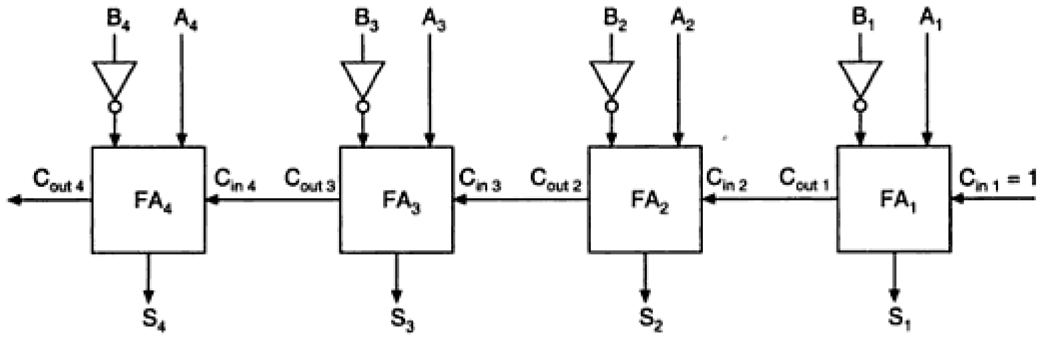
Logic diagram of a 4-bit binary parallel adder.

Ripple carry Adder:

The carry propagation delay for each full-adder is the time between the application of the carry-in and the occurrence of the carry-out (i.e) c4 is waiting for c3, c3 is waiting for c2, and c2 is waiting for c1. That is one Full Adder is waiting for another Full Adder

4-Bit Parallel Subtractor:

The subtraction of binary numbers can be carried out most conveniently by means of complements, the subtraction A-B can be done by taking the 2‘s complement of B and adding it to A . The 2‘s complement can be obtained by taking the 1‘s complement and adding 1 to the least significant pair of bits. The 1‘s complement can be implemented with inverters as

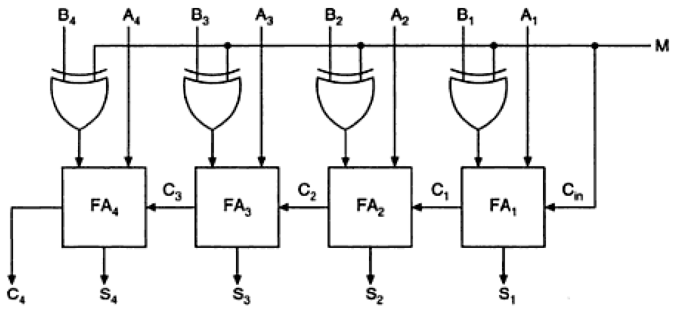


Logic diagram of a 4-bit parallel subtractor.

Binary-Adder Subtractor:

A 4-bit adder-subtractor, the addition and subtraction operations are combined into one circuit with one common binary adder. This is done by including an X-OR gate with each full-adder. The mode input M controls the operation. When M=0, the circuit is an adder, and when M=1, the circuit becomes a subtractor. Each X-OR gate receives input M and one of the inputs of B. When M=0, $B\oplus 0=B$.The full-adder receives the value of B, the input carry is 0 and the circuit performs A+B. when $B\oplus 1=B'$

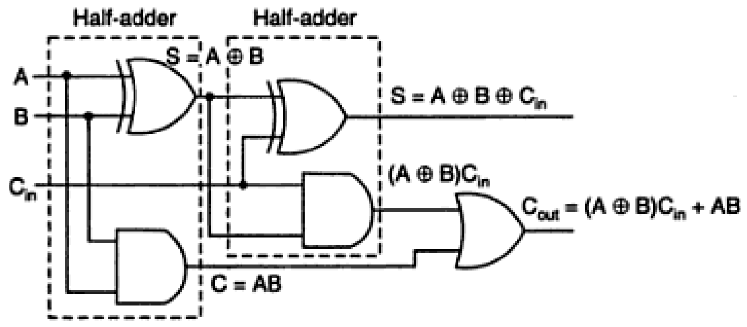
and C1=1. The B inputs are complemented and a 1 is through the input carry. The circuit performs the operation A plus the 2‘s complement of B.



Logic diagram of a 4-bit binary adder-subtractor.

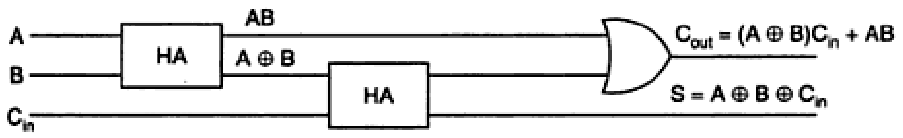
The Look-Ahead –Carry Adder:

The look-ahead carry adder speeds up the process by eliminating this ripple carry delay..



Logic diagram of a full-adder using two half-adders.

The block diagram of a full-adder using two half-adders is



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

Block diagram of a full-adder using two half-adders.

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

Observe that the final output carry is expressed as a function of the input variables in SOP form. Which is two level AND-OR or equivalent NAND-NAND form. Observe that the full look-ahead scheme requires the use of OR gate with (n+1) inputs and AND gates with number of inputs varying from 2 to (n+1).

BCD Adder:

The BCD addition process:
1. Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.
2. For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.
3. When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result. This will produce a carry to be added to the next decimal position.

A BCD adder circuit must be able to operate in accordance with the above steps. In other words, the circuit must be able to do the following:
1. Add two 4-bit BCD code groups, using straight binary addition.

2. Determine, if the sum of this addition is greater than 1101 (decimal 9); if it is, add 0110 (decimal 6) to this sum and generate a carry to the next decimal position.

The two BCD code groups A3A2A1A0and B3B2B1B0 are applied to a 4-bit parallel adder, the adder will output S4S3S2S1S0 , where S4 is actually C4 , the carry –out of the MSB bits.
The sum outputs S4S3S2S1S0 can range anywhere from 00000 to 100109when both the BCD code groups are 1001=9). The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001, so that the correction can be added in. Those cases, where the sum is greater than 1001 are listed as:

Let us define a logic output X that will go HIGH only when the sum is greater than 01001 (i.e, for the cases in table). If examine these cases ,see that X will be HIGH for either of the following conditions:
1. Whenever S4 =1(sum greater than15)
2. Whenever S3 =1 and either S2 or S1 or both are 1 (sum 10 to 15). This condition can be expressed as

X=S4+S3(S2+S1)

Whenever X=1, it is necessary to add the correction factor 0110 to the sum bits, and to generate a carry. The circuit consists of three basic parts. The two BCD code groups A3A2A1A0 and B3B2B1B0 are added together in the upper 4-bit adder, to produce the sum S4S3S2S1S0. The logic gates shown implement the expression for X.

The lower 4-bit adder will add the correction 0110 to the sum bits, only when X=1, producing the final BCD sum output represented by #3#2#1#0. The X is also the carry-out that is produced when the sum is greater than 01001.

When X=0, there is no carry and no addition of 0110. In such cases, #3#2#1#0= S3S2S1S0.

Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the second BCD adder, the carry-out of the second BCD adder is connected as the carry-in of the third BCD adder and so on.

Multiplexer:

Multiplexing means sharing.
There are two types of multiplexing: time multiplexing and frequency multiplexing.
A multiplexer (MUX) or data selector is a logic circuit that accepts several data inputs and allows only one of them at a time to get through to the output (i.e) 2^n inputs, n select lines and 1 output.

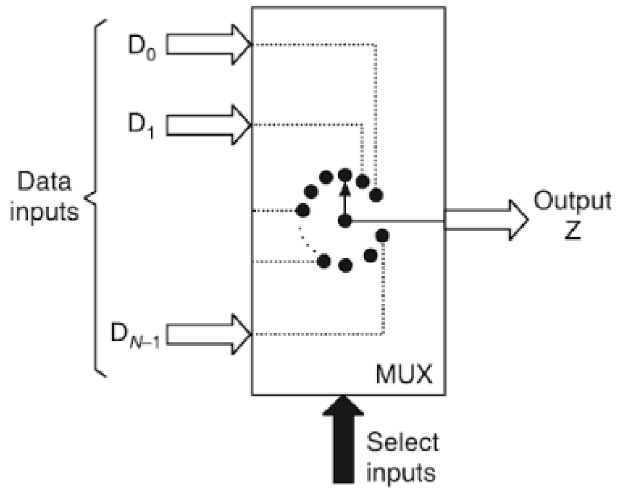
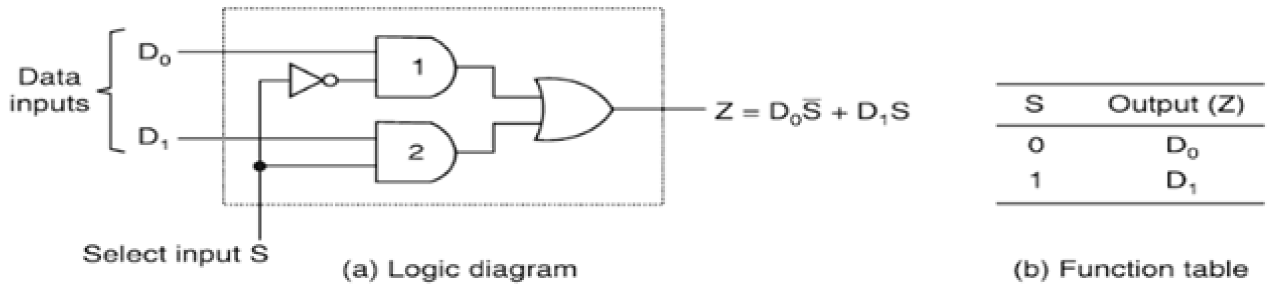


Fig: Functional diagram of a digital multiplexer

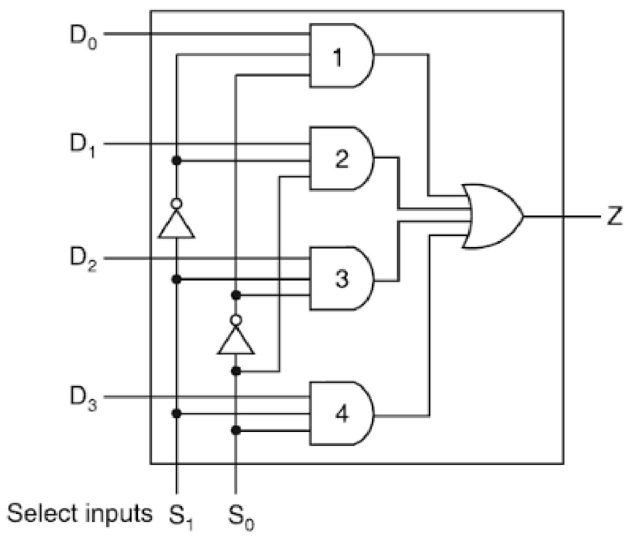
Basic 2 input multiplexor:
When S=0, AND gate 1 is enabled and AND gate 2 is disabled. So, $Z=D_0$.
When S=1, AND gate 1 is disabled and AND gate 2 is enabled. So, $Z=D_1$.



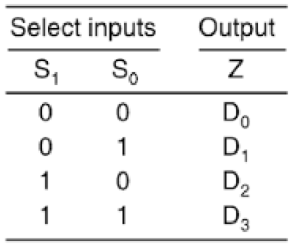
S	Output (Z)
0	D_0
1	D_1

Fig: 2 input multiplexor

The 4 input multiplexor:



(a) Logic diagram

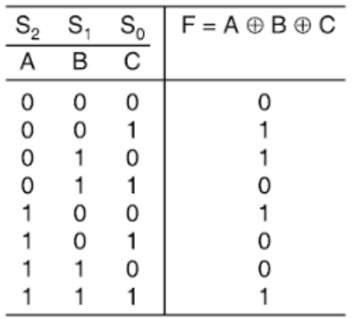


Select inputs		Output
S_1	S_0	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

(b) Function table

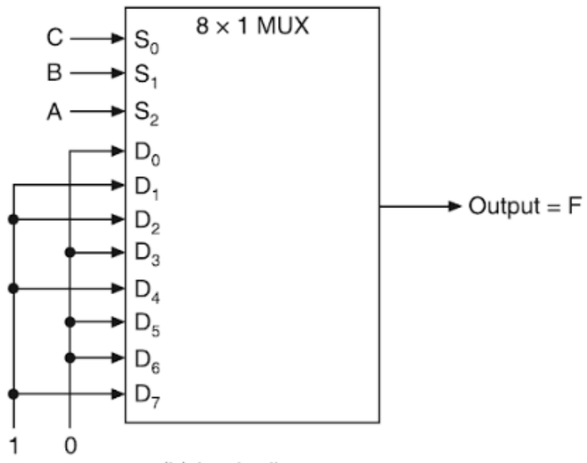
Fig: 4 input multiplexor

Example1: Use multiplex to implement the logic function F = $A{\scriptstyle A\oplus B}B{\scriptstyle A\oplus B}C$



S_2	S_1	S_0	$F = A \oplus B \oplus C$
A	B	C	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(a) Truth table

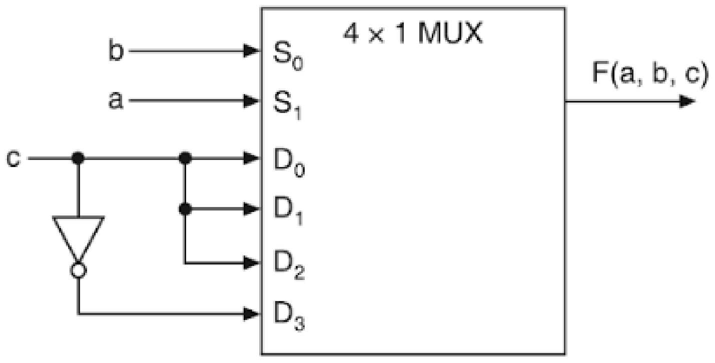


(b) Logic diagram

Example 2: Implement the following function with a MUX: F(a,b,c)=#m(1,3,5,6)
i)Select a, b, c as inputs ii) Choose a and b as select inputs

S_1	S_0			
a	b	c	F	
0	0	0	0	F = c
0	0	1	1	
0	1	0	0	F = c
0	1	1	1	
1	0	0	0	F = c
1	0	1	1	
1	1	0	1	$F = \overline{c}$
1	1	1	0	

ii) (a) Truth table



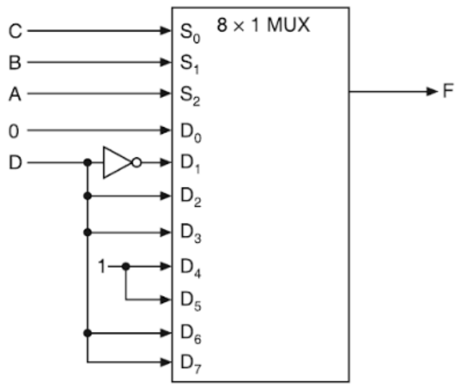
(b) Multiplexer implementation

Example 3: Implement the following Boolean function using an (i) 16:1, (ii) 8:1 multiplexer considering D as the input and A, B, C as the selection lines: F(a, B, C, D)=AB#+BD+B#CD#

(ii)

S_2	S_1	S_0			
A	B	C	D	F	
0	0	0	0	0	F = 0
0	0	0	1	0	
0	0	1	0	1	$F = \overline{D}$
0	0	1	1	0	
0	1	0	0	0	F = D
0	1	0	1	1	
0	1	1	0	0	F = D
0	1	1	1	1	
1	0	0	0	1	F = 1
1	0	0	1	1	
1	0	1	0	1	F = 1
1	0	1	1	1	
1	1	0	0	0	F = D
1	1	0	1	1	
1	1	1	0	0	F = D
1	1	1	1	1	

(a) Truth table



(b) Logic diagram

Decoder:
A decoder is a combinational circuit that converts binary information from ‘n’ input lines to a maximum of ‘2^{n}’ unique output lines.

3 x 8 Decoder:

2 x 4 Decoder: Some decoders are constructed with NAND gates. The use of NAND gates as the decoding element, results in an active-“LOW” output while the rest will be “HIGH”. The NAND gate produces the AND operation with an inverted output. Decoders may include enable inputs to control the circuit operation. A two-to-four-line decoder with an enable input constructed with NAND gates is shown below. The circuit operates with complemented outputs and a complement enables input. The decoder is enabled when E is equal to 0

4 x 16 Decoder: Number of inputs-4, Number of outputs-16

It can be constructed with two 3 x 8 decodes

Combinational Logic Implementation for full adder:

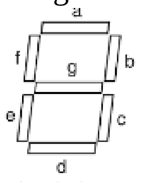
S(x, y, z) = #(1, 2, 4, 7)
C(x, y, z) = #(3, 5, 6, 7)

Design BCD to decimal decoder:

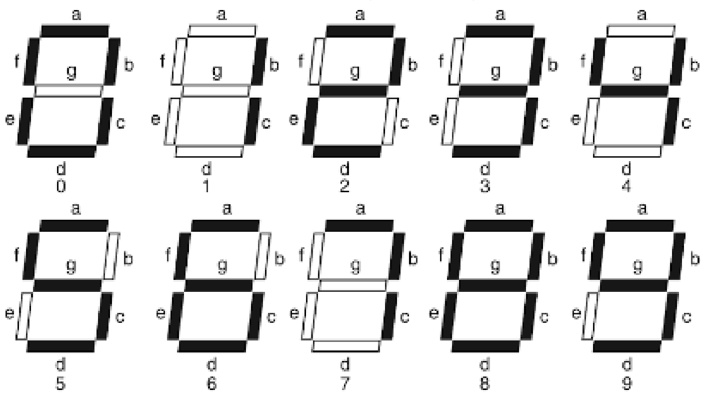
Number of digits in a BCD system=10
Each digit represented with 4 bits.

Number of inputs=4, Number of outputs=2^n = 16, where 10(0-9) are used and remaining 6 are unused.

Design BCD to 7 segment decoder:

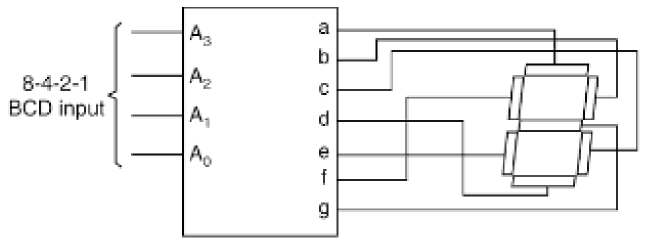


(a) Letters used to designate the segments

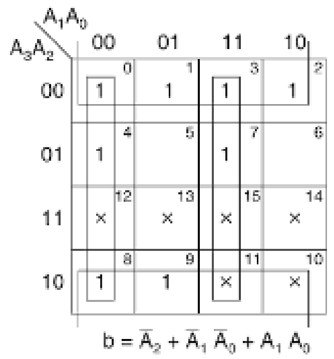


b) By causing different combinations of the segments to illuminate (shown with solid black), the numerals 0–9 can be displayed.

Number of digits in a BCD system=10
Each digit represented with 4 bits.
Number of inputs=4, Number of outputs=10 with 7(a-g) segments

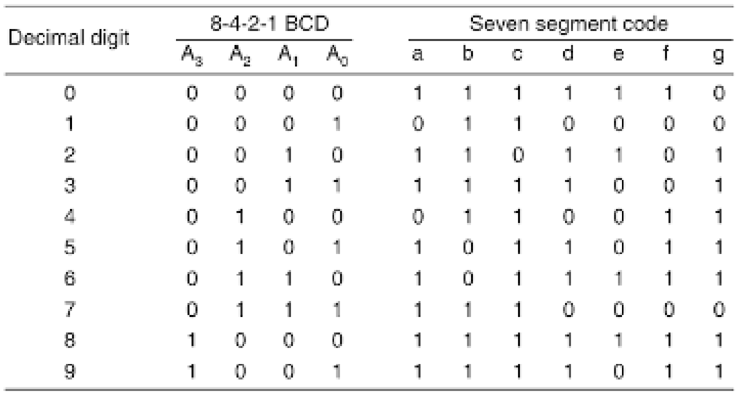


(a) Logic circuit



$b = \overline{A}_2 + \overline{A}_1\,\overline{A}_0 + A_1\,A_0$

(c) K-map to derive simplified expression for driving segment (b)



Decimal digit	8-4-2-1 BCD				Seven segment code						
	A_3	A_2	A_1	A_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

(b) Function table

$$\begin{aligned} b &= \overline{A}_3\overline{A}_2\overline{A}_1\overline{A}_0 + \overline{A}_3\overline{A}_2\overline{A}_1A_0 + \overline{A}_3\overline{A}_2A_1\overline{A}_0 + \overline{A}_3\overline{A}_2A_1A_0 + \overline{A}_3A_2\overline{A}_1\overline{A}_0 \\ &\quad + \overline{A}_3A_2A_1A_0 + A_3\overline{A}_2\overline{A}_1\overline{A}_0 + A_3\overline{A}_2\overline{A}_1A_0 \\ &= \Sigma\, m(0, 1, 2, 3, 4, 7, 8, 9) \end{aligned}$$

Don't cares,

$$d = \Sigma\, m(10, 11, 12, 13, 14, 15)$$

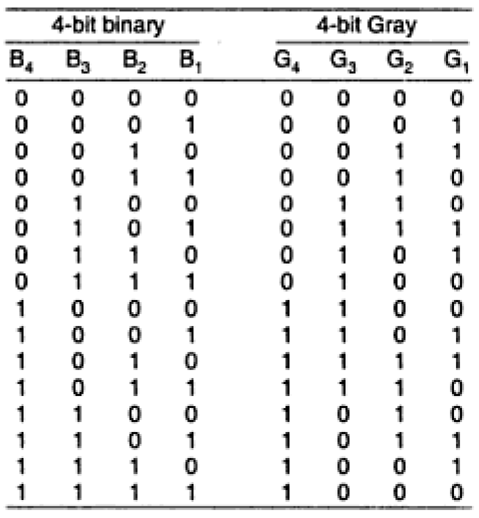
Code converters:

A code converter is a logic circuit whose inputs are bit patterns representing numbers (or character) in one code and whose outputs are the corresponding representation in a different code. Code converters are usually multiple output circuits.

To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B. A combinational circuit performs this transformation by means of logic gates.

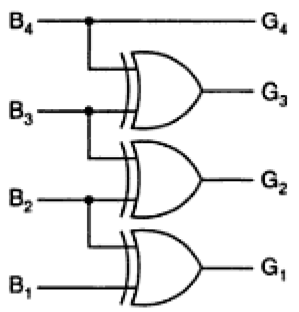
For example, a binary–to-gray code converter has four binary input lines B4, B3, B2, B1 and four gray code output lines G4, G3, G2, G1. When the input is 0010, for instance, the output should be 0011 and so forth. To design a code converter, we use a code table treating it as a truth table to express each output as a Boolean algebraic function of all the inputs.

This code converter is actually equivalent to four logic circuits, one for each of the truth tables.



4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

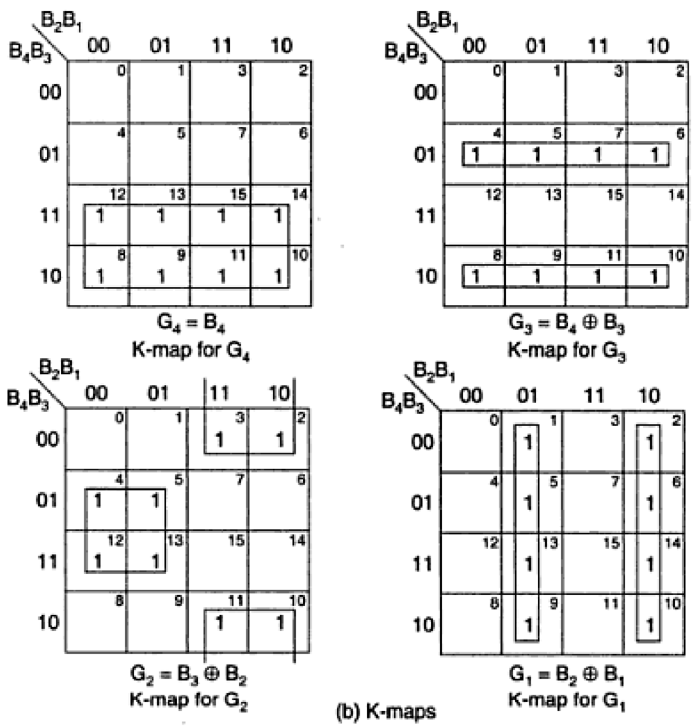
(a) Conversion table



(c) Logic diagram

4-bit binary-to-Gray code converter

Design of a 4-bit binary to gray code converter:



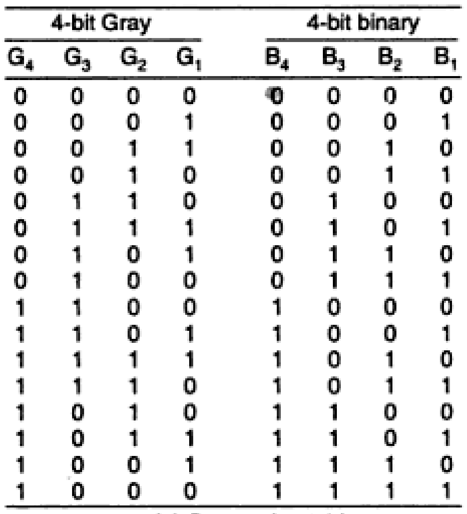
(b) K-maps

4-bit binary-to-Gray code converter.

Design of a 4-bit gray to binary code converter:

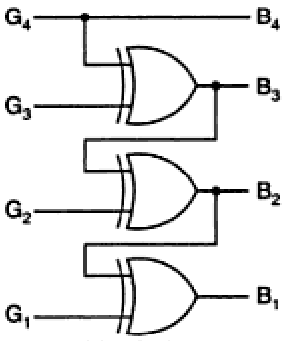
$B_4 = \Sigma\, m(12, 13, 15, 14, 10, 11, 9, 8) = \Sigma\, m(8, 9, 10, 11, 12, 13, 14, 15)$
$B_3 = \Sigma\, m(6, 7, 5, 4, 10, 11, 9, 8) = \Sigma\, m(4, 5, 6, 7, 8, 9, 10, 11)$
$B_2 = \Sigma\, m(3, 2, 5, 4, 15, 14, 9, 8) = \Sigma\, m(2, 3, 4, 5, 8, 9, 14, 15)$
$B_1 = \Sigma\, m(1, 2, 7, 4, 13, 14, 11, 8) = \Sigma\, m(1, 2, 4, 7, 8, 11, 13, 14)$

$B_4 = G_4$
$B_3 = \bar{G}_4 G_3 + G_4 \bar{G}_3 = G_4 \oplus G_3$
$B_2 = \bar{G}_4 G_3 \bar{G}_2 + \bar{G}_4 \bar{G}_3 G_2 + G_4 \bar{G}_3 \bar{G}_2 + G_4 G_3 G_2$
$\quad = \bar{G}_4 (G_3 \oplus G_2) + G_4 (\overline{G_3 \oplus G_2}) = G_4 \oplus G_3 \oplus G_2$
$B_1 = \bar{G}_4 \bar{G}_3 \bar{G}_2 G_1 + \bar{G}_4 \bar{G}_3 G_2 \bar{G}_1 + \bar{G}_4 G_3 G_2 G_1 + \bar{G}_4 G_3$
$\quad + G_4 G_3$

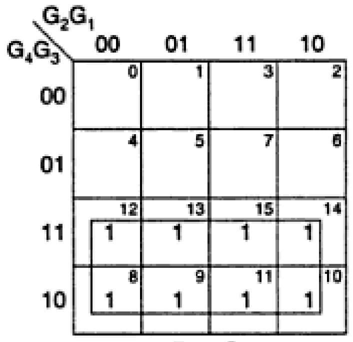


4-bit Gray				4-bit binary			
G_4	G_3	G_2	G_1	B_4	B_3	B_2	B_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

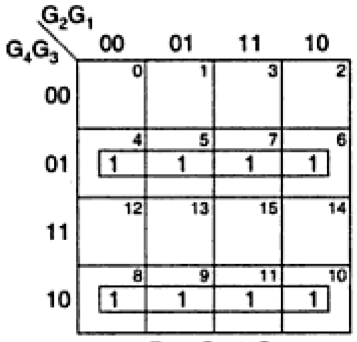
(a) Conversion table



(c) Logic diagram

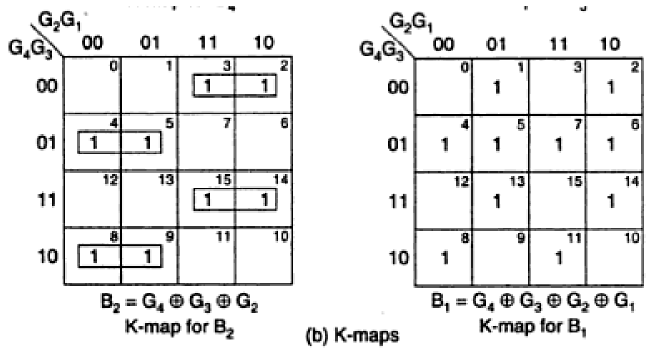


$B_4 = G_4$
K-map for B_4



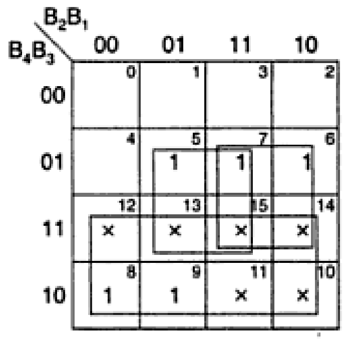
$B_3 = G_4 \oplus G_3$
K-map for B_3

$$= \bar{G}_4\bar{G}_3(G_2 \oplus G_1) + G_4G_3(G_2 \oplus G_1) + \bar{G}_4G_3(\overline{G_2 \oplus G_1}) + G_4\bar{G}_3(\overline{G_2 \oplus G_1})$$
$$= (G_2 \oplus G_1)(\overline{G_4 \oplus G_3}) + (\overline{G_2 \oplus G_1})(G_4 \oplus G_3)$$
$$= G_4 \oplus G_3 \oplus G_2 \oplus G_1$$

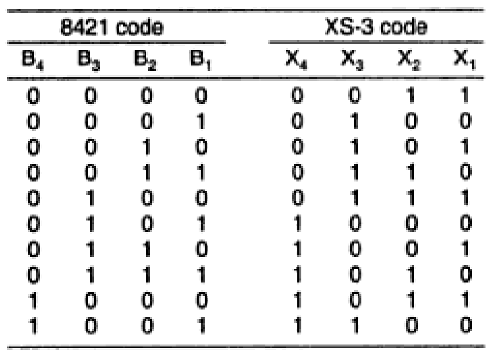

G_2G_1
G_4G_3
$B_2 = G_4 \oplus G_3 \oplus G_2$
K-map for B_2
(b) K-maps
$B_1 = G_4 \oplus G_3 \oplus G_2 \oplus G_1$
K-map for B_1

4-bit Gray-to-binary code converter.

Design of a 4-bit BCD to XS-3 code converter:



$X_4 = B_4 + B_3B_2 + B_3B_1$
K-map for X_4



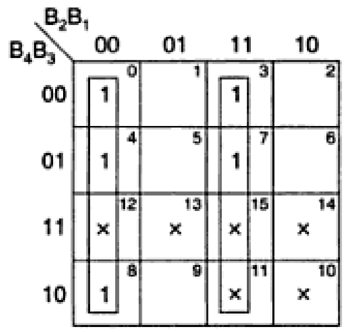
8421 code				XS-3 code			
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

(a) Conversion table

$X_4 = \Sigma\, m(5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$
$X_3 = \Sigma\, m(1, 2, 3, 4, 9) + d(10, 11, 12, 13, 14, 15)$
$X_2 = \Sigma\, m(0, 3, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$
$X_1 = \Sigma\, m(0, 2, 4, 6, 8) + d(10, 11, 12, 13, 14, 15)$

The minimal expressions are
$X_4 = B_4 + B_3B_2 + B_3B_1$
$X_3 = B_3\overline{B}_2\overline{B}_1 + \overline{B}_3B_1 + \overline{B}_3B_2$
$X_2 = \overline{B}_2\overline{B}_1 + B_2B_1$
$X_1 = \overline{B}_1$

(b) Minimal expressions



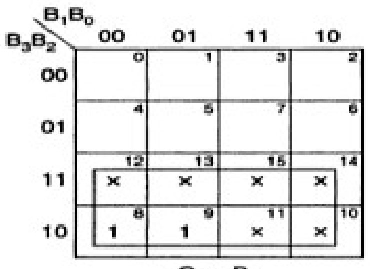
$X_2 = \overline{B}_2\overline{B}_1 + B_2B_1$
K-map for X_2

(c) K

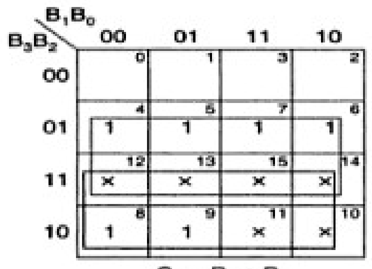
4-bit BCD-to-XS-3 code converter

4-bit BCD-

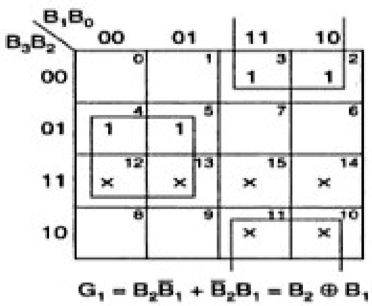
Design of a BCD to gray code converter:



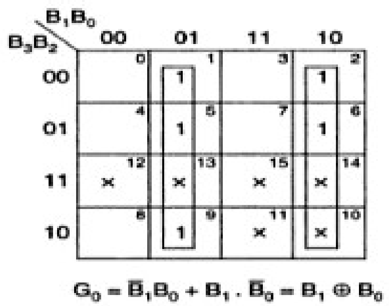
$G_3 = B_3$



$G_2 = B_2 + B_3$

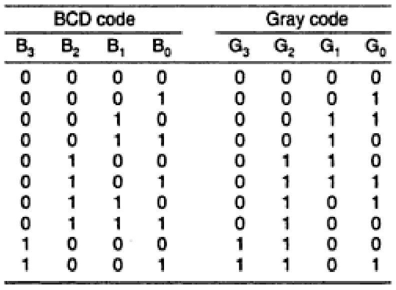


$G_1 = B_2\overline{B}_1 + \overline{B}_2B_1 = B_2 \oplus B_1$



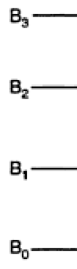
$G_0 = \overline{B}_1B_0 + B_1 \cdot \overline{B}_0 = B_1 \oplus B_0$

K-maps for a BCD-to-Gray code converter.



BCD code				Gray code			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

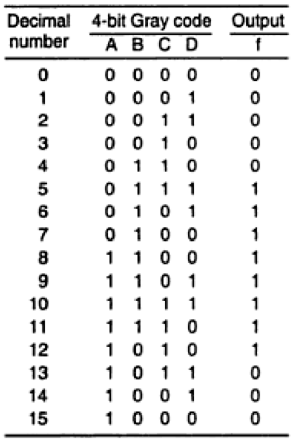
(a) BCD-to-Gray code conversion table



(b

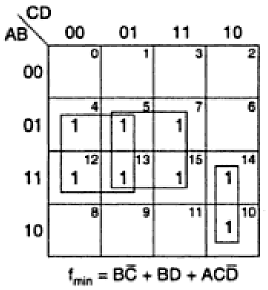
BCD-to-Gray code converte

Design of a SOP circuit to Detect the Decimal numbers 5 through 12 in a 4-bit gray code Input:



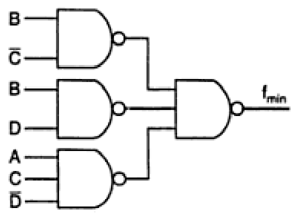
Decimal number	4-bit Gray code A	B	C	D	Output f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	1	0
3	0	0	1	0	0
4	0	1	1	0	0
5	0	1	1	1	1
6	0	1	0	1	1
7	0	1	0	0	1
8	1	1	0	0	1
9	1	1	0	1	1
10	1	1	1	1	1
11	1	1	1	0	1
12	1	0	1	0	1
13	1	0	1	1	0
14	1	0	0	1	0
15	1	0	0	0	0

(a) Truth table



$f_{min} = B\overline{C} + BD + AC\overline{D}$

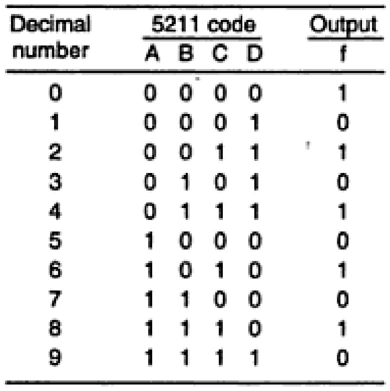
(b) K-map



(c) NAND logic

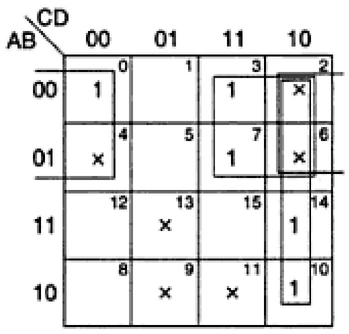
Truth table, K-map and logic diagram for the SOP circuit.

Design of a SOP circuit to detect the decimal numbers 0,2,4,6,8 in a 4-bit 5211 BCD code input:



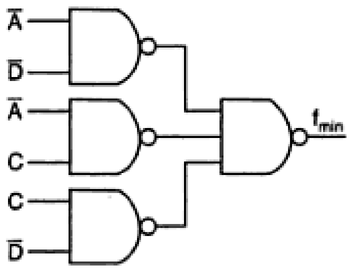
Decimal number	5211 code A	B	C	D	Output f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	1	1
3	0	1	0	1	0
4	0	1	1	1	1
5	1	0	0	0	0
6	1	0	1	0	1
7	1	1	0	0	0
8	1	1	1	0	1
9	1	1	1	1	0

(a) Truth table



$f_{min} = \overline{A}\overline{D} + \overline{A}C + C\overline{D}$

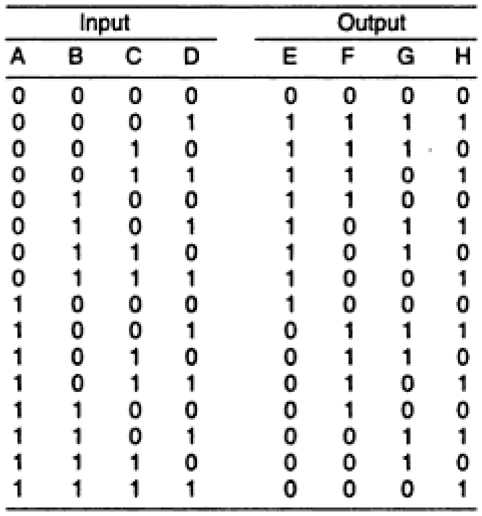
(b) K-map



(c) Logic diagram

Truth table, K-map and logic diagram for the SOP circuit.

Design of a Combinational circuit to produce the 2's complement of a 4-bit binary number:



Input A	B	C	D	Output E	F	G	H
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	0
0	0	1	1	1	1	0	1
0	1	0	0	1	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	0	1	1	1
1	0	1	0	0	1	1	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

(a) Conversion table

Draw K-maps for E, F, G, H

Expressions are

E=AB#C#D#+A#B+A#D+A#C
F=BC#D#+B#D+B#C
G=C#D+CD#
H=D

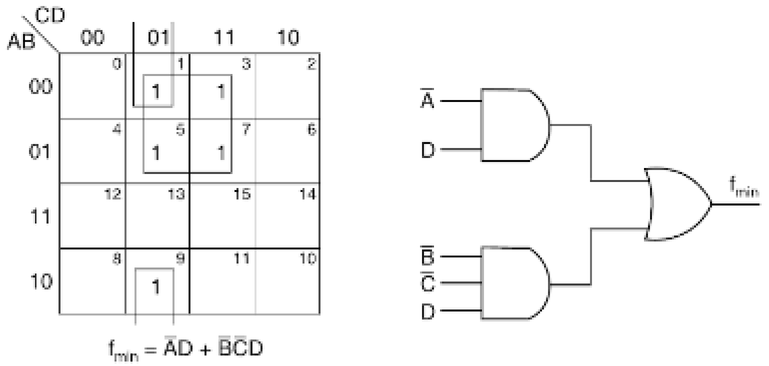
EXAMPLE Design each of the following circuits that can be built using AOI logic and outputs a 1 when:

(a) A 4-bit hexadecimal input is an odd number from 0 to 9.

(b) A 4-bit BCD code translated to a number that uses the upper right segment of a seven-segment display.

Solution

(a) The output is a 1 when the input is a 4-bit hexadecimal odd number from 0 to 9. There are 16 possible combinations of inputs, and all are valid. The output is a 1 only for the input combinations 0001, 0011, 0101, 0111, and 1001. For all other combinations of inputs, the output is a 0. The problem may thus be stated as $f = \Sigma\, m(1, 3, 5, 7, 9)$. The SOP K-map, its minimization, the minimal expression obtained from it, and the realization of the minimal expression in AOI logic are shown in Figures 4.46a and b.



(a) K-maps (b) Logic diagram

(b) We see from Figure 4.47a that display of digits 0, 1, 2, 3, 4, 7, 8, and 9 requires the upper-right segment of the seven-segment display. Since the input is a 4-bit BCD, inputs 1010

through 1111 are invalid, and therefore, the corresponding outputs are don't cares. The problem may be stated as

$$f = \Sigma\, m(0, 1, 2, 3, 4, 7, 8, 9) + d\,(10, 11, 12, 13, 14, 15)$$

The K-map, its minimization, the minimal expression obtained from it and the realization of the minimal expression using AOI logic are shown in Figures 4.47b and c respectively.

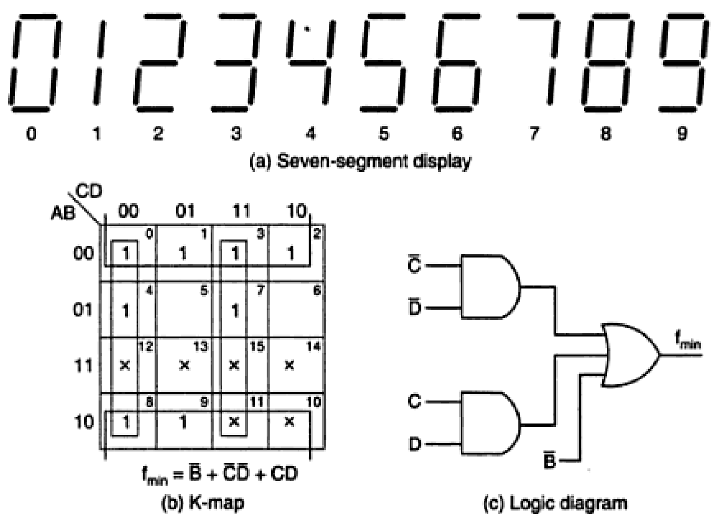


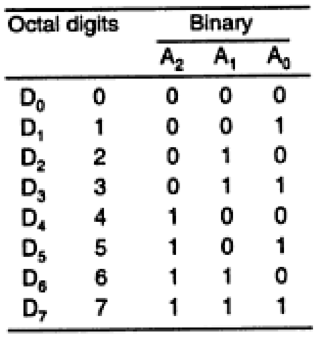
Fig 4.47

Encoders:

It is a device whose inputs are decimal digits or alphabetical characters and whose outputs are the coded representation of corresponding inputs.

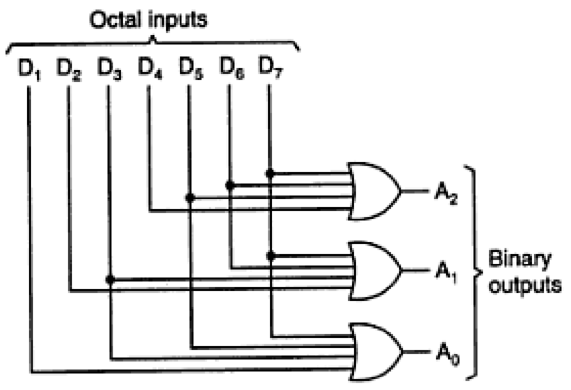
Encoder (2n x n) is a combinational logic circuit which performs the reverse operation of decoder

Octal to Binary Encoder (8 x 3 encoder):



Octal digits		Binary		
		A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

(a) Truth table



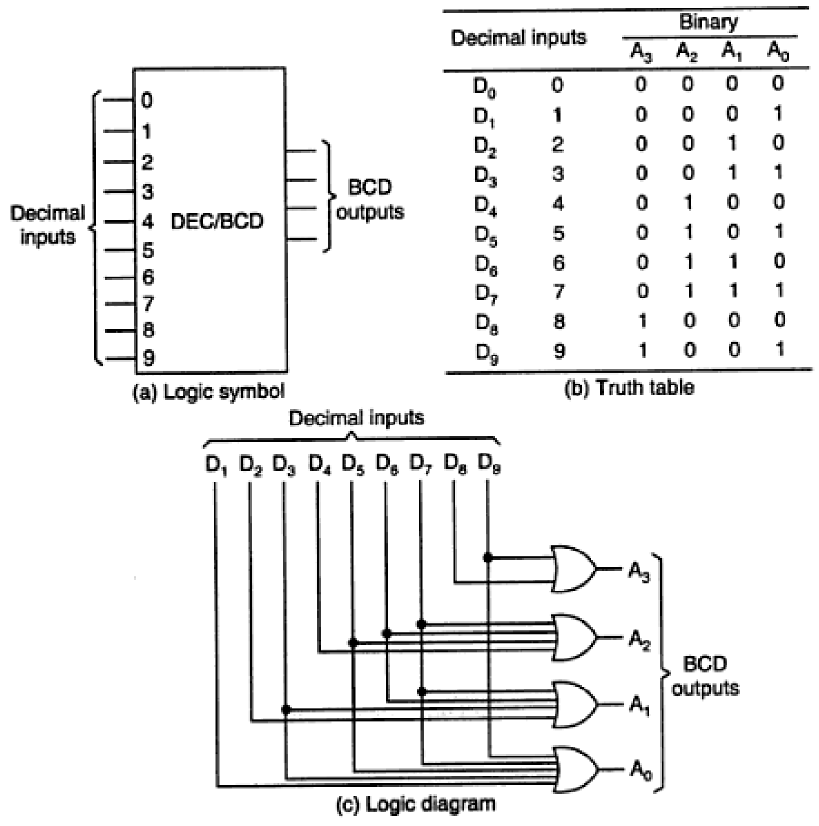
(b) Logic diagram

Octal-to-binary encoder.

$z/A_0 = D1 + D3 + D5 + D7$
$y/A_1 = D2 + D3 + D6 + D7$

x/A_2 = D4 + D5 + D6 + D7

Decimal to BCD Encoder:



Decimal inputs		Binary			
		A_3	A_2	A_1	A_0
D_0	0	0	0	0	0
D_1	1	0	0	0	1
D_2	2	0	0	1	0
D_3	3	0	0	1	1
D_4	4	0	1	0	0
D_5	5	0	1	0	1
D_6	6	0	1	1	0
D_7	7	0	1	1	1
D_8	8	1	0	0	0
D_9	9	1	0	0	1

Decimal-to-BCD encoder.

Priority Encoder:
A priority encoder is a logic circuit which responds to one input according to some priority system.

Convert 4 x 2 priority encoder:
D_0-least, D_3-highest
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.
Number of inputs=4
Number of outputs=2 and additional output to check validity of inputs.
Assume the inputs are D_0, D_1, D_2, D_3.
Assume the outputs are x, y, V(validation bit).
If V = 0, inputs are invalid and output is also invalid (i.e) outputs are specified as don't-care conditions
If V = 1, inputs are valid and based on priority among the inputs it generates the output x, y.

Instead of listing all 16 minterms of four variables, the truth table uses an X to represent either 1 or 0 in input columns. For example, X 100 represents the two minterms 0100 and 1100.
Input D_3 has the highest priority, so, regardless of the values of the other inputs, when this input is 1, the output for xy is 11 (binary 3).
D_2 has the next priority level. The output is 10 if D2 = 1, provided that D3 = 0, regardless of the values of the other two lower priority inputs.
The output for D1 is generated only if higher priority inputs are 0, and so on down the priority levels.

T

The minterms for the two functions are derived from the above Table. Although the table has only five rows, when each X in a row is replaced first by 0 and then by 1, we obtain all 16 possible input combinations. For example, the fourth row in the table, with inputs XX10, represents the four minterms 0010, 0110, 1010, and 1110.

The simplified Boolean expressions for the priority encoder are obtained from the maps. The condition for output V is an OR function of all the input variables. The priority encoder is implemented in below Fig. according to the following Boolean functions:

$x = D_2 + D_3$

$y = D_3 + D_1 D'_2$

$V = D_0 + D_1 + D_2 + D_3$

De-Multiplexer:

It is also known as data distributors. It takes a single input and distributed it over several outputs based on the selection lines. It is a reverse operation of Multiplexer.

Number of Inputs=1, Number of select lines=n, Number of outputs=2^n

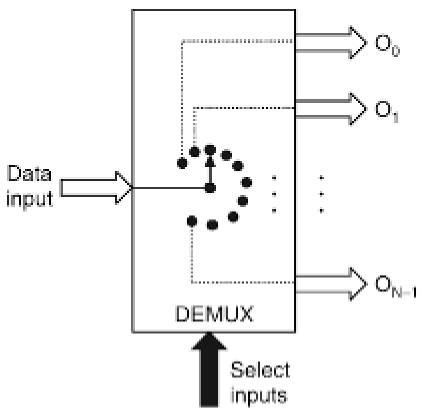
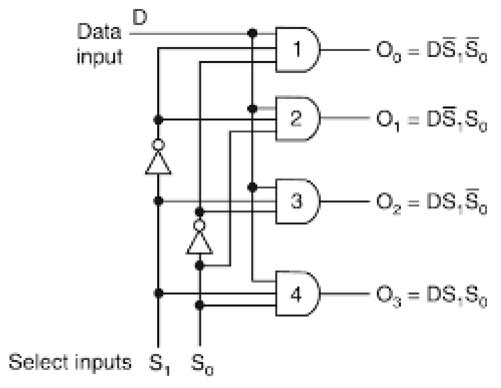


Fig: Functional diagram of a general demultiplexer

1-line to 4-line demultiplexer:



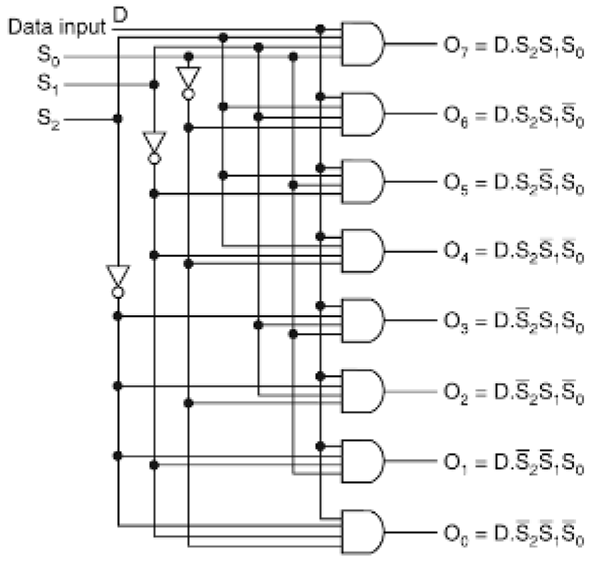
(a) Logic diagram

Select code		Outputs			
S_1	S_0	O_3	O_2	O_1	O_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

(b) Truth table

Fig: 1-line to 4-line demultiplexer

1-line to 8-line demultiplexer:


Data input
D
S_0
S_1
S_2
$O_7 = D.S_2S_1S_0$
$O_6 = D.S_2S_1\bar{S}_0$
$O_5 = D.S_2\bar{S}_1S_0$
$O_4 = D.S_2\bar{S}_1\bar{S}_0$
$O_3 = D.\bar{S}_2S_1S_0$
$O_2 = D.\bar{S}_2S_1\bar{S}_0$
$O_1 = D.\bar{S}_2\bar{S}_1S_0$
$O_0 = D.\bar{S}_2\bar{S}_1\bar{S}_0$

(a) Logic diagram

Select code			Outputs							
S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	D
0	0	1	0	0	0	0	0	0	D	0
0	1	0	0	0	0	0	0	D	0	0
0	1	1	0	0	0	0	D	0	0	0
1	0	0	0	0	0	D	0	0	0	0
1	0	1	0	0	D	0	0	0	0	0
1	1	0	0	D	0	0	0	0	0	0
1	1	1	D	0	0	0	0	0	0	0

(b) Truth table

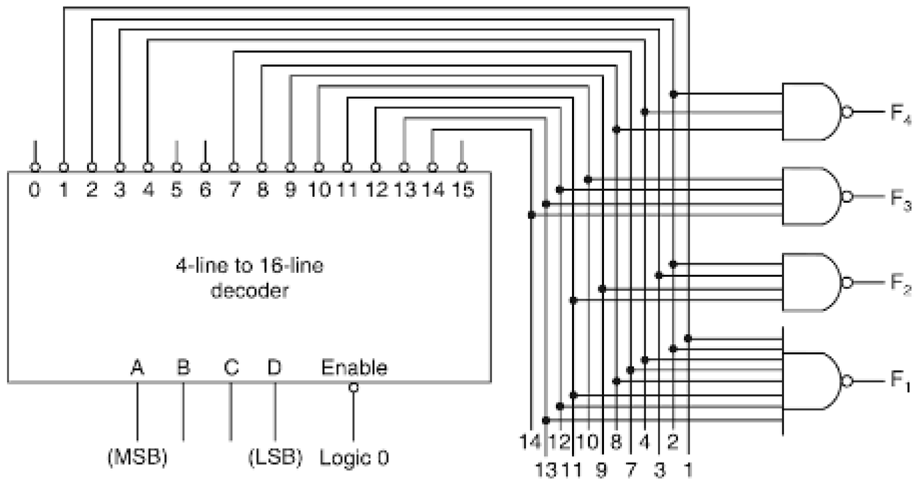
Fig: 1-line to 8-line demultiplexer

Example: Implement the following multiple output combinational logic circuit using a 4-line to 16-line decoder.

$$F_1 = \Sigma\, m(1, 2, 4, 7, 8, 11, 12, 13)$$
$$F_2 = \Sigma\, m(2, 3, 9, 11)$$
$$F_3 = \Sigma\, m(10, 12, 13, 14)$$
$$F_4 = \Sigma\, m(2, 4, 8)$$

Solution

The realization of the given multiple output logic circuit using a 4-line to 16-line decoder is shown in Figure 4.91. The decoder's outputs are active LOW; therefore, a NAND gate is required for every output of the combinational circuit. In combinational logic design using a multiplexer, additional gates are not required, whereas the design using a demultiplexer requires additional gates. However, even with this disadvantage, the decoder is more economical in cases where non-trivial, multiple-output expressions of the same input variables are required. In such cases, one multiplexer is required for each output, whereas it is likely that only one decoder supported with a few gates would be required. Therefore, using a decoder could have advantages over using a multiplexer.



Demultiplexer Tree: 5-line to 32-line decoder/demultiplexer using two 4-line to 16-line decoders.

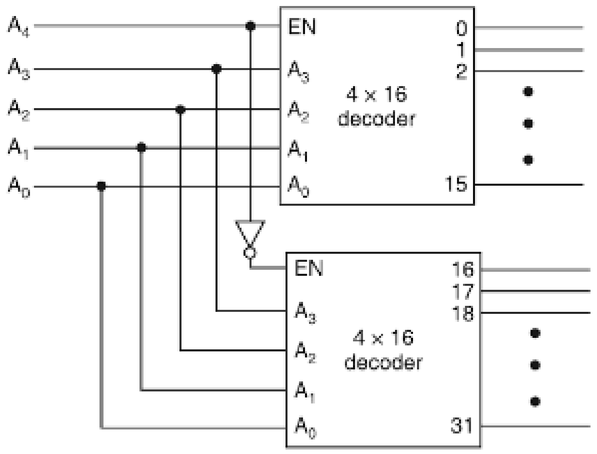
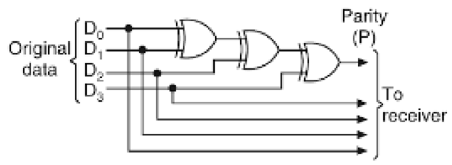


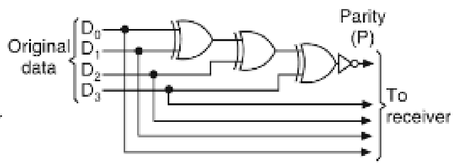
Fig: 5:32 decoder from two 4:16 decoders.

Parity bit generators:

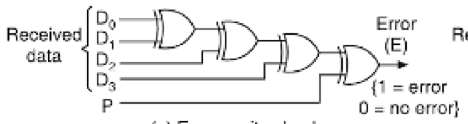
The circuit that generates the parity bit in the transmitter is called a parity generator. The circuit that checks the parity in the receiver is called a parity checker.



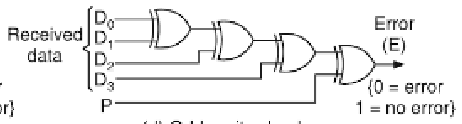
(a) Even parity generator



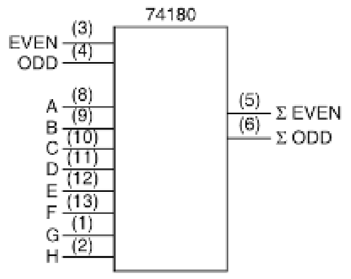
(b) Odd parity generator



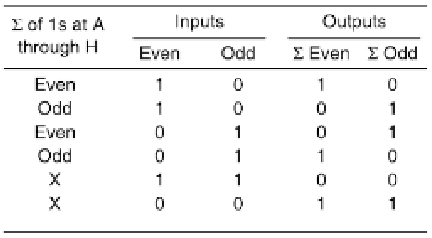
(c) Even parity checker



(d) Odd parity checker



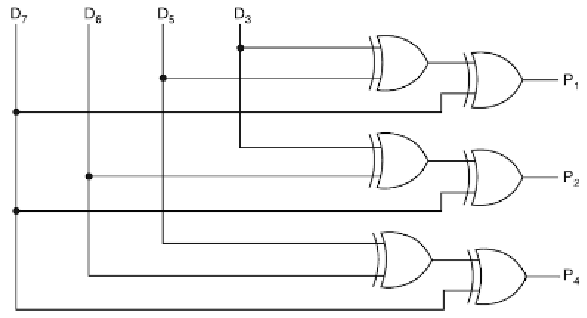
(e) Logic symbol of IC 74180



Σ of 1s at A through H	Inputs		Outputs	
	Even	Odd	Σ Even	Σ Odd
Even	1	0	1	0
Odd	1	0	0	1
Even	0	1	0	1
Odd	0	1	1	0
X	1	1	0	0
X	0	0	1	1

(f) Truth table of IC 74180 (X = don't care)

Parallel parity bit generator for hamming code:
Consider the 7-bit Hamming code ($P_1P_2D_3P_4D_5D_6D_7$)



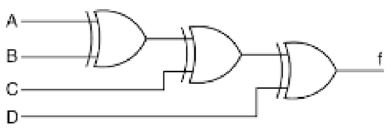
Design of an even parity bit generator for a 4-bit input:

4-bit data input				Output
A	B	C	D	parity bit (f)
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

(a) Truth table

AB \ CD	00	01	11	10
00	^{0}	1 ^{1}	^{3}	1 ^{2}
01	1 ^{4}	^{5}	1 ^{7}	^{6}
11	^{12}	1 ^{13}	^{15}	1 ^{14}
10	1 ^{8}	^{9}	1 ^{11}	^{10}

(b) K-map for f



(c) Logic diagram

Figure 4.52 Even parity bit generator.

From the K-map, the output is

$$f = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BCD + AB\bar{C}D + ABC\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD$$
$$= \bar{A}\bar{B}(C \oplus D) + \bar{A}B(\overline{C \oplus D}) + AB(C \oplus D) + A\bar{B}(\overline{C \oplus D})$$
$$= (C \oplus D)(\overline{A \oplus B}) + (\overline{C \oplus D})(A \oplus B)$$
$$= A \oplus B \oplus C \oplus D$$

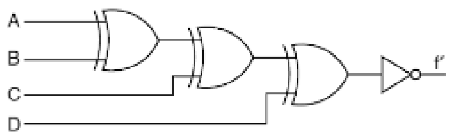
Design of an odd parity bit generator for a 4-bit input:

4-bit data input				Output
A	B	C	D	parity bit f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

(a) Truth table

AB \ CD	00	01	11	10
00	1 ^{0}	^{1}	1 ^{3}	^{2}
01	^{4}	1 ^{5}	^{7}	1 ^{6}
11	1 ^{12}	^{13}	1 ^{15}	^{14}
10	^{8}	1 ^{9}	^{11}	1 ^{10}

(b) K-map for f′



(c) Logic diagram

Figure 4.53 Odd parity bit generator.

From the K-map, the output is

$$f = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + AB\bar{C}\bar{D} + ABCD + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D}$$
$$= \bar{A}\bar{B}(\overline{C \oplus D}) + AB(\overline{C \oplus D}) + \bar{A}B(C \oplus D) + A\bar{B}(C \oplus D)$$
$$= (\overline{C \oplus D})(\overline{A \oplus B}) + (C \oplus D)(A \oplus B)$$
$$= \overline{(A \oplus B) \oplus (C \oplus D)}$$

PLD's-ROM:

PLD (Programmable Logic Devices):

It is an IC chip that is user configurable and it's capable of implementing logic functions.
A PLD contains large number of gates, flip flops and registers that are interconnected on the chip. Many of the connections are fusable that is the link can be broken or disconnected. The IC is said to be programmable the specific function of IC of given application is determined by selective breaking of sum of interconnection while leaving others in contact.

ROM:

It is a memory device in which permanent binary information is stored. A ROM which can be programmed is called PROM.

The process of entering information into a ROM is known as programming.
Size of the ROM = 2^k x n
k = number of input lines (address lines)
n = number of output lines (data line)

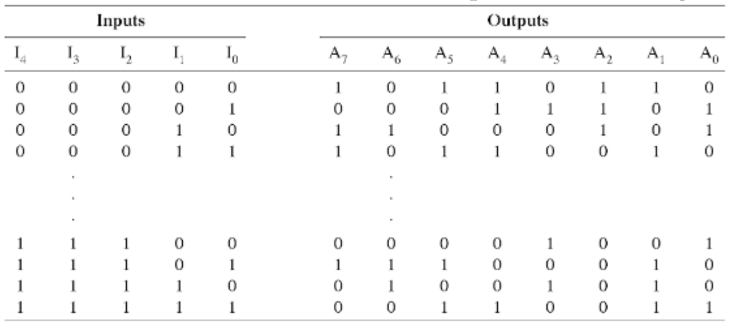
Fig: ROM block diagram

Example: 32 x 8 ROM

k x 2^k decoder to decode input address
  - o Number of input lines = 5
  - o Decoder size = 5 x 32
  - o Number of output lines = 8

n OR gates with 2^k input each
# Decoder output is connected to all n OR gates through fuses
# ROM $\rightarrow$ 2^k x n programmable connections

For example, the table specifies the 8-bit word 10110010 for permanent storage at address 3.



Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
	.					.						
	.					.						
	.					.						
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

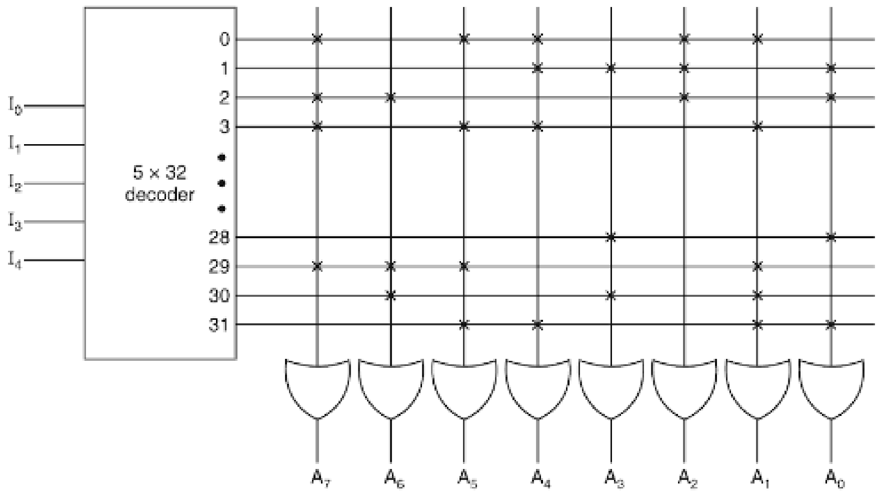
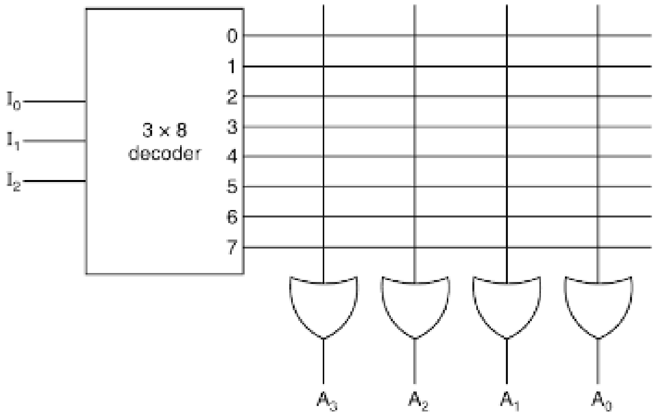
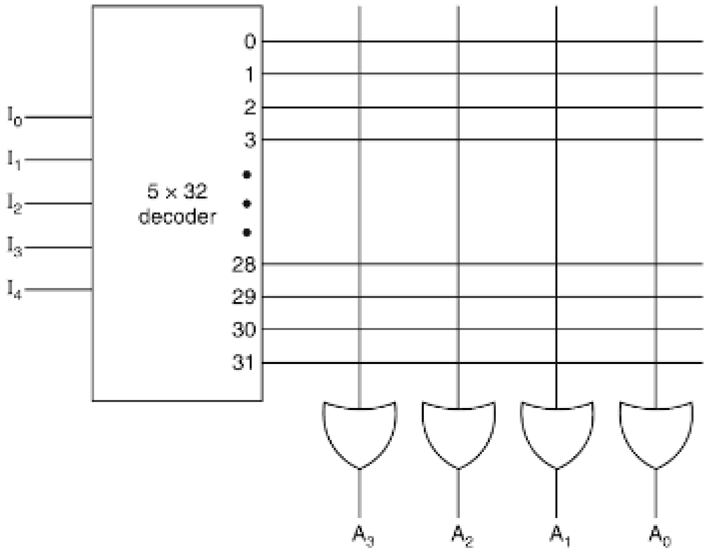


Fig: Programming the ROM

Example 1: Give the logic implementation of a 8 x 4 bit and 32 x 4 bit ROM using a decoder of a suitable size





Example 2: Design ROM of size 8 x 3 outputs.

$F1=\sum m(0,4,7)$

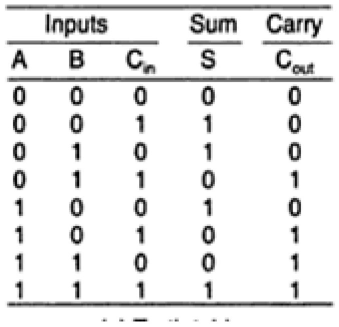
$F2=\sum m(1,3,6)$

$F3=\sum m(2,4,6)$

Number of inputs=3, ROM size= 8 x 3, Number of outputs=3

Example 3: Design a full adder using ROM.

Number of inputs = 3, ROM size = 8 x 2, Number of outputs = 2



Inputs			Sum	Carry
A	B	C_{in}	S	C_{out}
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

Example 4: Design a combinational circuit using a ROM the circuit accepts 3 bit number and generates an output binary number equal to the square of input number.[Note: Use the ROM size as 8 x 4]

PROM:

The address inputs to the PROM serve as logic variable inputs and the data output as the node where the output of a logic function is realized.

EXAMPLE .1 Show how an 8 × 1 PROM can be programmed to implement the logic function whose truth table is shown in Figure

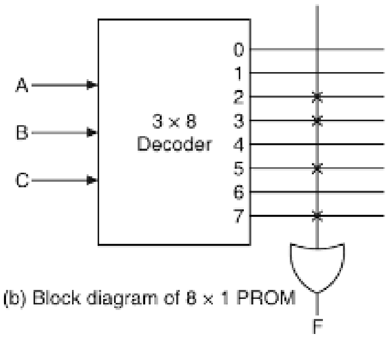
Solution

From the truth table shown in Figure , we observe that the logic function is

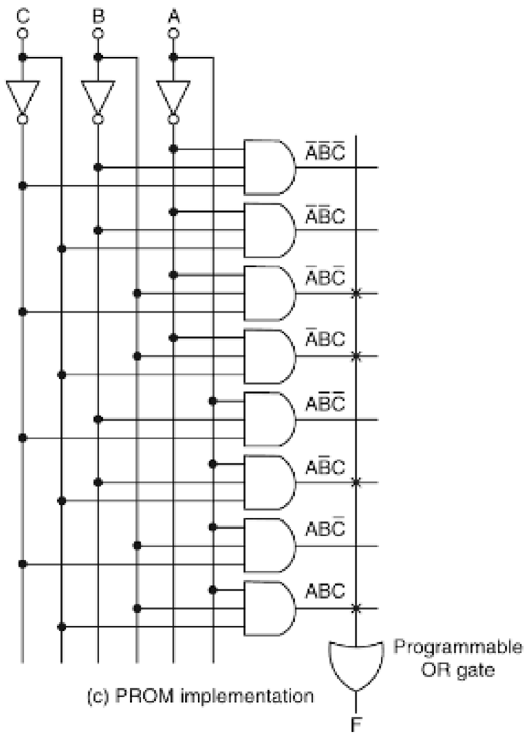
$$F = \Sigma\, m(2, 3, 5, 7) = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$$

Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

(a) Truth table



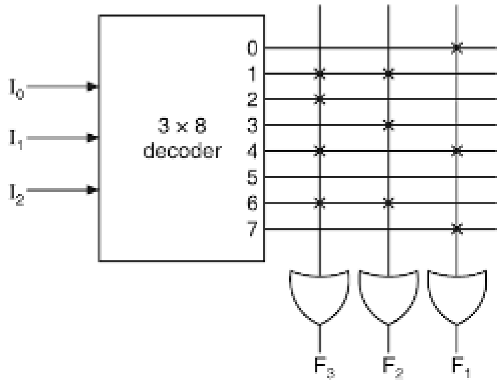
(b) Block diagram of 8 × 1 PROM



(c) PROM implementation

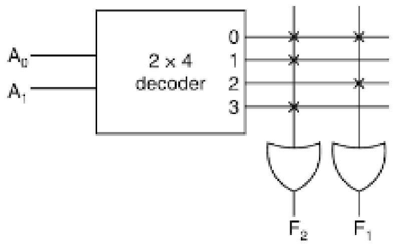
EXAMPLE Realize the following functions using a PROM of size 8 × 3

$$F_1 = \Sigma\, m(0, 4, 7)$$
$$F_2 = \Sigma\, m(1, 3, 6)$$
$$F_3 = \Sigma\, m(1, 2, 4, 6)$$



EXAMPLE Realize two outputs F_1 and F_2 using a 4 × 2 PROM:

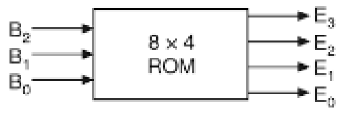
$$F_1(A_1, A_0) = \Sigma\, m(0, 2)$$
$$F_2(A_1, A_0) = \Sigma\, m(0, 1, 3)$$



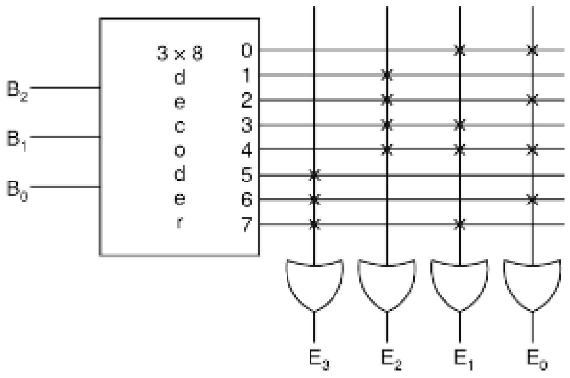
EXAMPLE Design a combinational circuit using a PROM. The circuit accepts a 3-bit binary number and generates its equivalent XS-3 code.

Inputs			Outputs			
B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	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	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

$E_0 = \Sigma\, m(0, 2, 4, 6)$
$E_1 = \Sigma\, m(0, 3, 4, 7)$
$E_2 = \Sigma\, m(1, 2, 3, 4)$
$E_3 = \Sigma\, m(5, 6, 7)$



(a) Block diagram



(b) Realization of the circuit

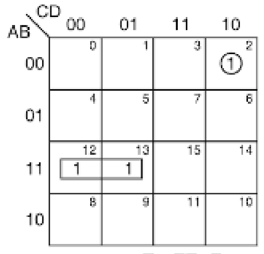
PAL:

Fig: Programmable array logic (PAL) device

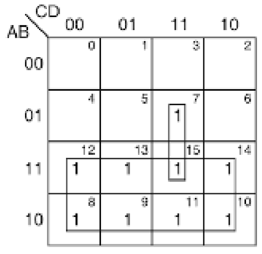
It is a programmable logic device with a fixed OR array and programmable AND array.

EXAMPLE Implement the following Boolean functions using PAL with four inputs and 3-wide AND-OR structure. Also write the PAL programming table.

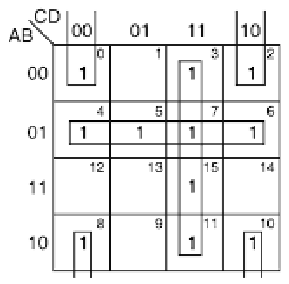
$F_1(A, B, C, D) = \Sigma\, m(2, 12, 13)$
$F_2(A, B, C, D) = \Sigma\, m(7, 8, 9, 10, 11, 12, 13, 14, 15)$
$F_3(A, B, C, D) = \Sigma\, m(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$
$F_4(A, B, C, D) = \Sigma\, m(1, 2, 8, 12, 13)$.



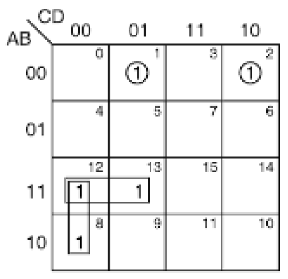
$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$



$F_2 = A + BCD$

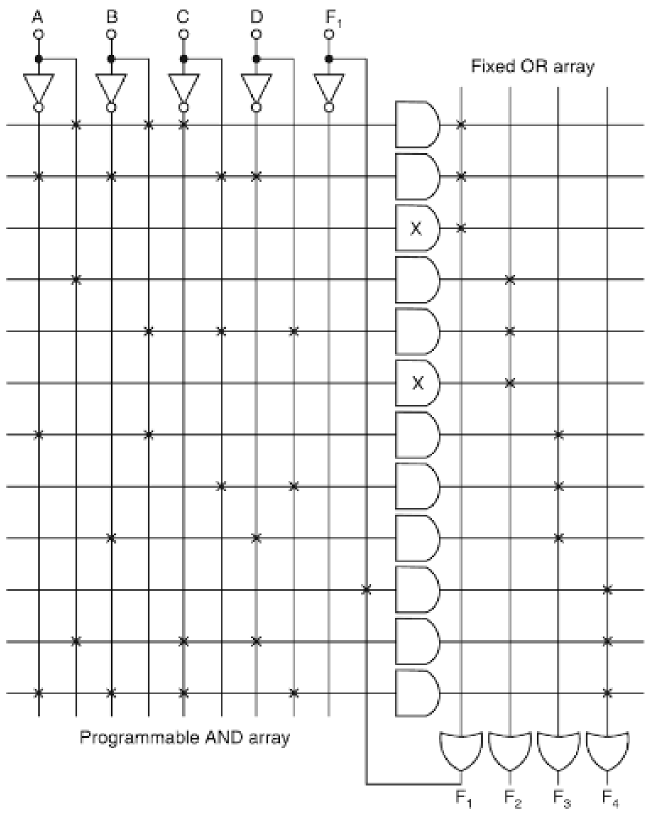


$F_3 = CD + \bar{A}B + \bar{B}\bar{D}$



$F_4 = AB\bar{C} + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D}$
$= F_1 + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$

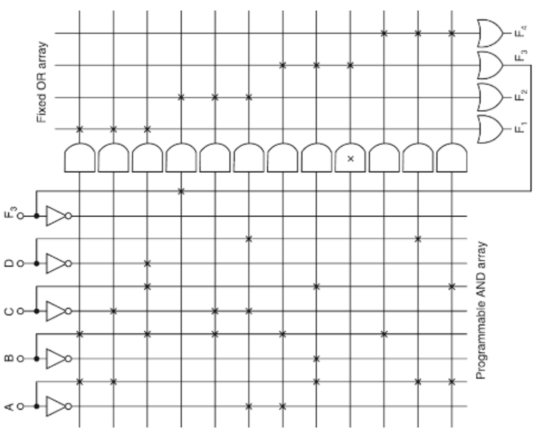
Product term	AND Inputs					Outputs
	A	B	C	D	F_1	
1	1	1	0	–	–	$F_1 = AB\bar{C} + \bar{A}\bar{B}C\bar{D}$
2	0	0	1	0	–	
3	–	–	–	–	–	
4	1	–	–	–	–	
5	–	1	1	1	–	$F_2 = A + BCD$
6	–	–	–	–	–	
7	0	1	–	–	–	$F_3 = \bar{A}B + CD + \bar{B}\bar{D}$
8	–	–	1	1	–	
9	–	0	–	0	–	
10	–	–	–	–	1	$F_4 = F_1 + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$
11	1	–	0	0	–	
12	0	0	0	1	–	



EXAMPLE Realize the following functions using a PAL with four inputs and 3-wide AND-OR structure. Also write the PAL programming table.

$F_1(A, B, C, D) = \Sigma\, m(6, 8, 9, 12\text{–}15)$ $F_2(A, B, C, D) = \Sigma\, m(1, 4\text{–}7, 10\text{–}13)$
$F_3(A, B, C, D) = \Sigma\, m(4\text{–}7, 10\text{–}11)$ $F_4(A, B, C, D) = \Sigma\, m(4\text{–}7, 9\text{–}15)$

Product term	AND Inputs					Outputs
	A	B	C	D	F_3	
1	1	1	–	–	–	
2	1	–	0	–	–	$F_1 = AB + A\overline{C} + BC\overline{D}$
3	–	1	1	0	–	
4	–	–	–	–	1	$F_2 = \overline{A}B + A\overline{B}C + \overline{A}\overline{C}D + B\overline{C}$
5	0	–	0	1	–	$= F_3 + B\overline{C} + \overline{A}\overline{C}D$
6	–	1	0	–	–	
7	0	1	–	–	–	
8	1	0	1	–	–	$F_3 = \overline{A}B + A\overline{B}C$
9	–	–	–	–	–	
10	–	1	–	–	–	
11	1	–	–	1	–	$F_4 = B + AD + AC$
12	1	–	1	–	–	



PLA:

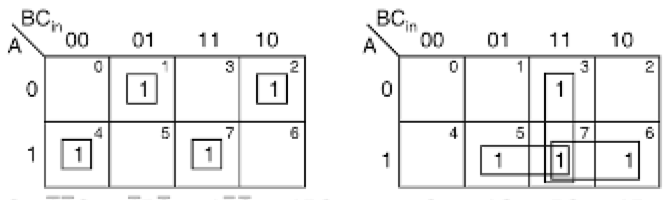
Fig: Programmable logic array (PLA) device

The PLA combines the characteristics of the PROM and the PAL by providing both a programmable OR array and a programmable AND array.

EXAMPLE Show how the PLA circuit would be programmed to implement the sum and carry outputs of a full adder.

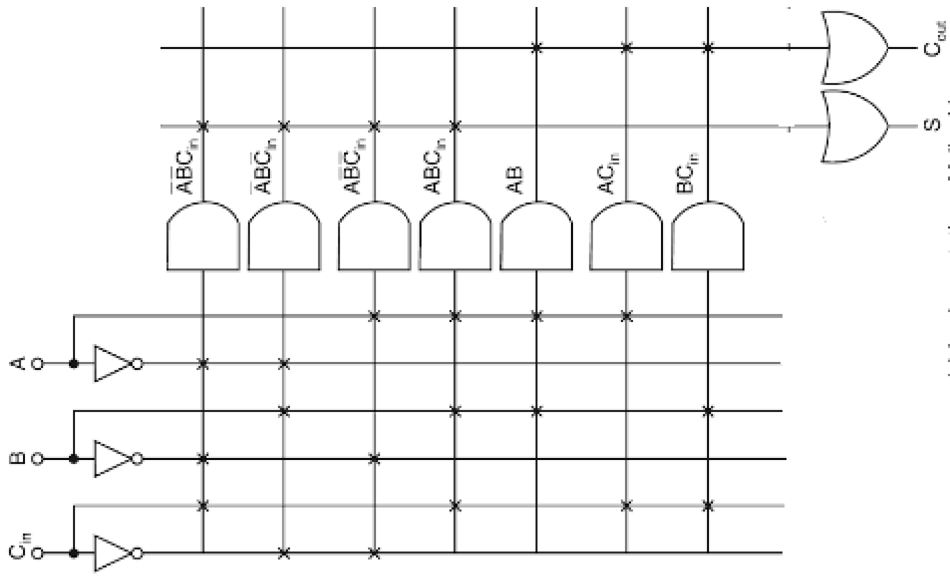
Inputs			Outputs	
A	B	C_{in}	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Truth table



$S = \overline{A}\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\overline{C}_{in} + ABC_{in}$ $C_{out} = AC_{in} + BC_{in} + AB$

(b) K-maps



(c) Implementation of full-adder

EXAMPLE Implement the following two Boolean functions with a PLA:

$$F_1(A, B, C) = \Sigma\, m(0, 1, 2, 4)$$
$$F_2(A, B, C) = \Sigma\, m(0, 5, 6, 7)$$

Solution

The K-maps for the functions F_1 and F_2, their minimization, and the minimal expressions for both the true and complement forms of those in sum of products are shown in Figure
For finding the minimal in true form, consider the 1s on the map and for finding the minimal in complement form consider the 0s on the map.

Considering the 1s of F_1

$$F_1(T) = \bar{A}\bar{C} + \bar{B}\bar{C} + \bar{A}\bar{B}$$

Considering the 0s of F_1

$$\bar{F}_1 = AB + AC + BC$$

Therefore,

$$F_1(C) = \overline{(AB + AC + BC)}$$

Considering the 1s of F_2

$$F_2(T) = \bar{A}\bar{B}\bar{C} + AB + AC$$

Considering the 0s of F_2

$$\overline{F}_2 = A\overline{B}\overline{C} + \overline{A}B + \overline{A}C$$

Therefore,

$$F_2(C) = \overline{A\overline{B}\overline{C} + \overline{A}B + \overline{A}C}$$

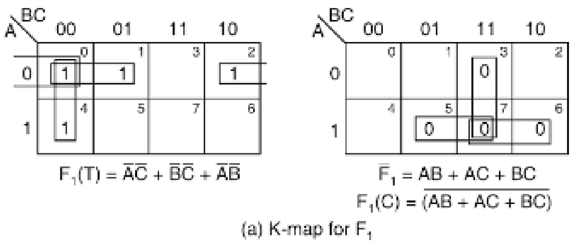
Out of $F_1(T)$, $F_1(C)$, $F_2(T)$, $F_2(C)$, the combination that gives the minimum number of product terms is

$$F_1(C) = \overline{(AB + AC + BC)}$$

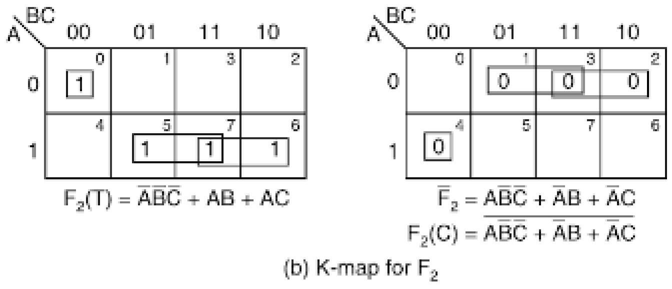
$$F_2(T) = AB + AC + \overline{A}\overline{B}\overline{C}$$

This gives four distinct terms: AB, AC, and BC and $\overline{A}\overline{B}\overline{C}$. The PLA programming table for this combination is shown in Figure 8.21a. The implementation using a PLA is shown in Figure 8.21b.

F_1 is the true output even though a C is marked over it in the table. This is because F_1 is generated with an AND-OR circuit and is available at the output of the OR gate. The X-OR gate complements the function to produce the true F_1 output.



$F_1(T) = \overline{A}\overline{C} + \overline{B}\overline{C} + \overline{A}\overline{B}$
$\overline{F}_1 = AB + AC + BC$
$F_1(C) = \overline{(AB + AC + BC)}$
(a) K-map for F_1

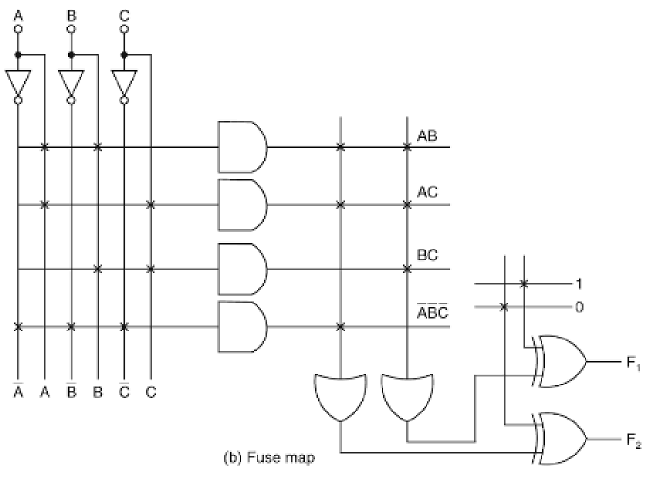


$F_2(T) = \overline{A}\overline{B}\overline{C} + AB + AC$
$\overline{F}_2 = A\overline{B}\overline{C} + \overline{A}B + \overline{A}C$
$F_2(C) = \overline{A\overline{B}\overline{C} + \overline{A}B + \overline{A}C}$
(b) K-map for F_2

Figure 8.20 Example 8.9: K-maps.

Product term		Inputs			Outputs	
		A	B	C	(C) F_1	(T) F_2
1	AB	1	1	–	1	1
2	AC	1	–	1	1	1
3	BC	–	1	1	1	–
4	$\overline{A}\overline{B}\overline{C}$	0	0	0	–	1

(a) PLA programming table



A
B
C
AB
AC
BC
$\overline{A}\overline{B}\overline{C}$
1
0
F_1
F_2
$\overline{A}$ A $\overline{B}$ B $\overline{C}$ C

(b) Fuse map

EXAMPLE Write the program table to implement a BCD to XS-3 code conversion using a PLA.

Solution

The BCD to XS-3 code conversion table and the expressions for the XS-3 outputs are shown in Figure 8.22.

Decimal	BCD Code				XS-3 Code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

(a) BCD to XS-3 conversion table

$E_0 = \Sigma\, m(0, 2, 4, 6, 8)$

$E_1 = \Sigma\, m(0, 3, 4, 7, 8)$

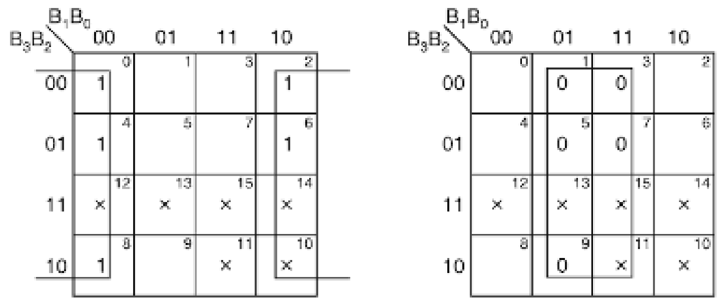
$E_2 = \Sigma\, m(1, 2, 3, 4, 9)$

$E_3 = \Sigma\, m(5, 6, 7, 8, 9)$

The don't cares are

$d = \Sigma\, d(10, 11, 12, 13, 14, 15)$

(b) Expressions for outputs

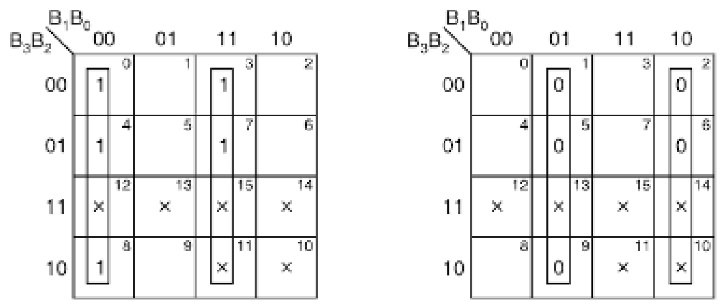


$E_0(T) = \bar{B}_0$

$\bar{E}_0 = B_0$

$E_0(C) = \bar{B}_0$

(a) K-maps for E_0

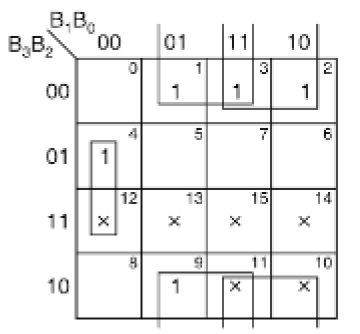


$E_1(T) = \bar{B}_1\bar{B}_0 + B_1B_0$

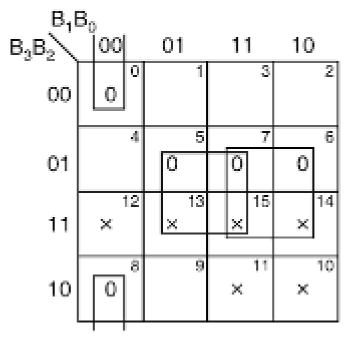
$\bar{E}_1 = \bar{B}_1B_0 + B_1\bar{B}_0$

$E_1(C) = \overline{\bar{B}_1B_0 + B_1\bar{B}_0}$

(b) K-maps for E_1



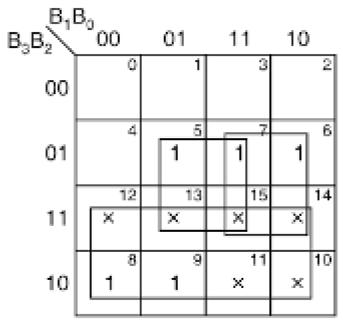
$E_2(T) = B_2\bar{B}_1\bar{B}_0 + \bar{B}_2B_0 + \bar{B}_2B_1$



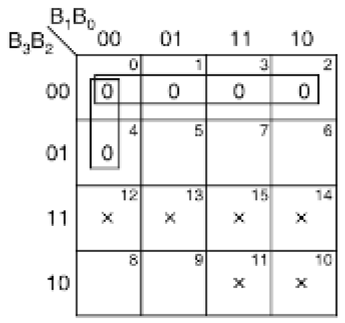
$\bar{E}_2 = \bar{B}_2\bar{B}_1\bar{B}_0 + B_2B_0 + B_2B_1$

$E_2(C) = \overline{\bar{B}_2\bar{B}_1\bar{B}_0 + B_2B_0 + B_2B_1}$

(c) K-maps for E_2



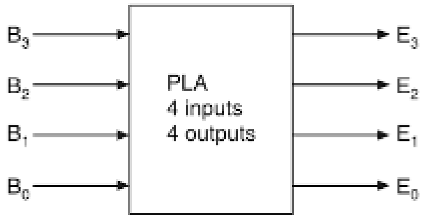
$E_3(T) = B_3 + B_2B_0 + B_2B_1$



$\bar{E}_3 = \bar{B}_3\bar{B}_2 + \bar{B}_3\bar{B}_1\bar{B}_0$

$E_3(C) = \overline{\bar{B}_3\bar{B}_2 + \bar{B}_3\bar{B}_1\bar{B}_0}$

(d) K-maps for E_3



(a) Block diagram

Product term		Inputs				Outputs			
		B_3	B_2	B_1	B_0	E_3 (T)	E_2 (C)	E_1 (T)	E_0 (T)
1	B_3	1	–	–	–	1	–	–	–
2	B_2B_0	–	1	–	1	1	1	–	–
3	B_2B_1	–	1	1	–	1	1	–	–
4	$\bar{B}_2\bar{B}_1\bar{B}_0$	–	0	0	0	–	1	–	–
5	$\bar{B}_1\bar{B}_0$	–	–	0	0	–	–	1	–
6	B_1B_0	–	–	1	1	–	–	1	–
7	$\bar{B}_0$	–	–	–	0	–	–	–	1

(b) PLA programming table