Digital Electronics

(19-204-0302)

Module - 4

Syllabus - Module 4

- Memory and Programmable Logic: Introduction to Programmable Logic Devices, Read Only Memory, Programmable Logic Arrays (PLA), Programmable Array Logic (PAL)
- Logic Families: Transistor Transistor Logic(TTL), Emitter Coupled Logic(ECL), MOSFET Logic, TTL Gates.

Programmable logic devices (PLDs).

- A programmable logic device is an IC that is user configurable and is capable of implementing logic functions.
- It is an LSI chip that contains a ‘regular’ structure and allows the designer to customize it for any specific application,
- i.e. it is programmed by the user to perform a function required for his application.
- A PLD contains a large number of gates, flip-flops, and registers that are interconnected on the chip.
- The IC is said to be programmable because the specific function of the IC for a given application is determined by the selective breaking of some of the interconnections while leaving others intact.

Advantages of PLDs over fixed function ICs

1. Low development cost
2. Less space requirement
3. Less power requirement
4. High reliability
5. Easy circuit testing
6. Easy design modification
7. High design security
8. Less design time
9. High switching speed

READ-ONLY MEMORY (ROM)

- A read-only memory (ROM) is a memory device in which permanent binary information is stored.

- The binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern.

- Once the pattern is established, it stays within the unit when the power is turned off and on again.

- As the name suggests it is meant only for reading the information from it.

- A ROM which can be programmed is called a PROM.

- The process of entering information in a ROM is known as programming.

- ROMs can be used for designing combinational logic circuits.

- Depending on the type of ROM used, a user can design and modify the circuits easily.

ROM ORGANIZATION

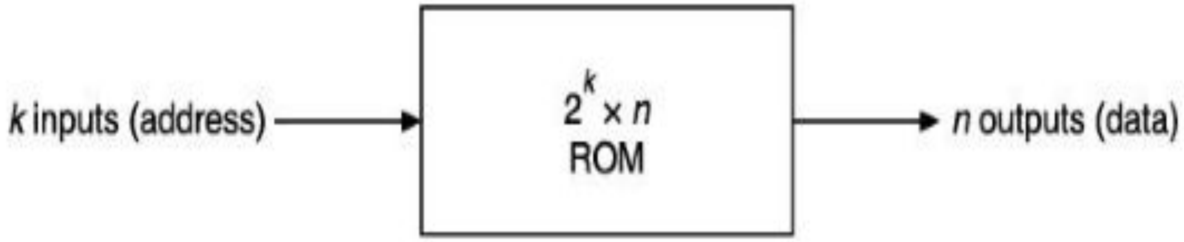
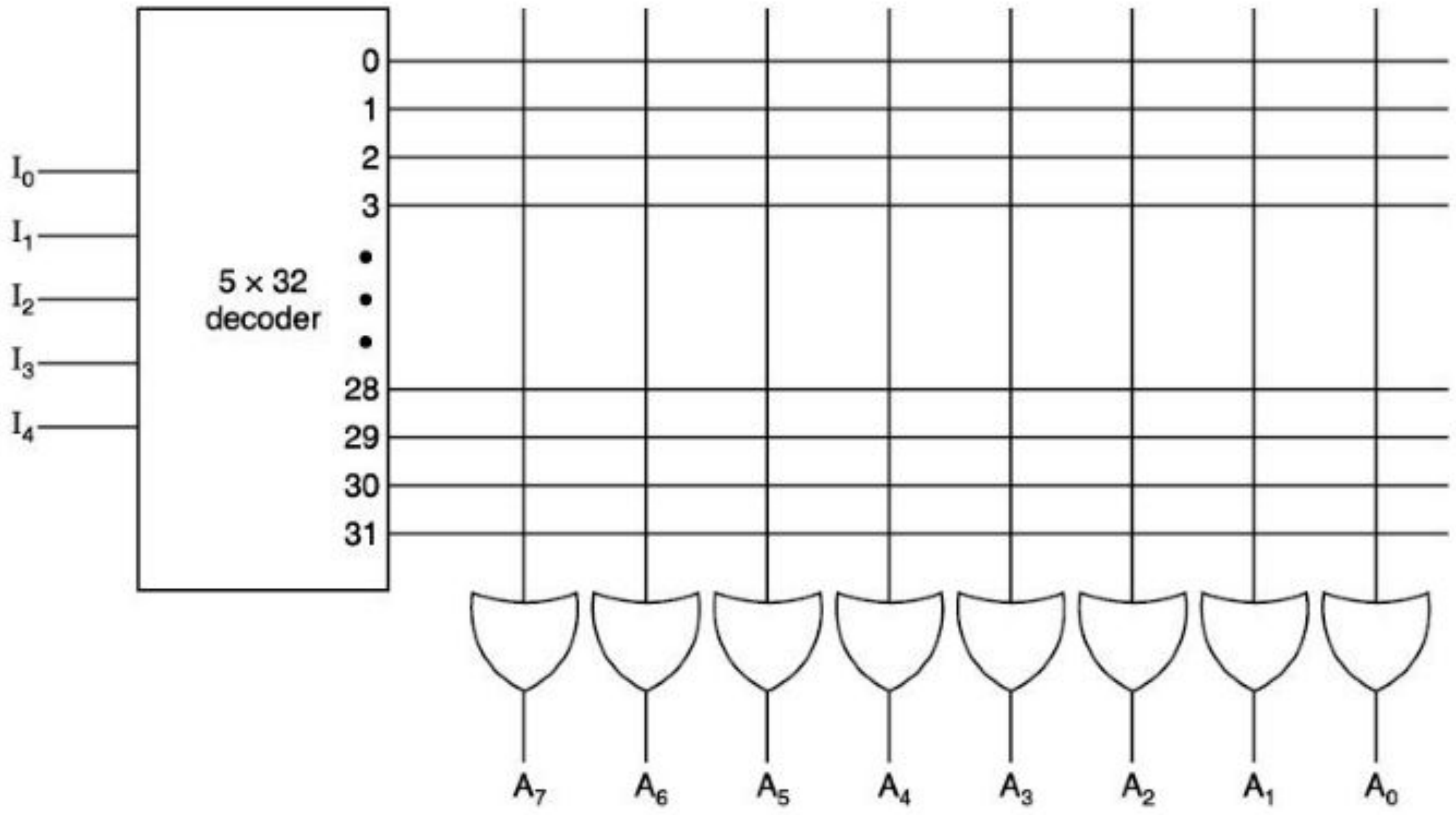


Figure 8.1 ROM block diagram.

- It consists of k inputs and n outputs.
- The inputs provide the address for the memory and the outputs give the data bits of the stored word which is selected by the address.
- The number of words in a ROM is determined from the fact that k address input lines are needed to specify 2^k words.

A 32 × 8 ROM.



Internal logic of a 32 × 8 ROM.

- The unit consists of 32 words of 8 bits each.
- There are five input lines that form the binary numbers from 0 through 31 for the address.
- The five inputs are decoded into 32 distinct outputs by means of a 5 × 32 decoder.
- The 32 outputs of the decoder are connected to each of the 8 OR gates.
- Each OR gate must be considered as having 32 inputs.
- Each output of the decoder is connected to one of the inputs of each OR gate.
- Since each OR gate has 32 input connections and there are 8 OR gates, the ROM contains 32 × 8 = 256 internal connections.

- In general, a $2^k \times n$ ROM will have an internal $k \times 2^k$ decoder and n OR gates.

- Each OR gate has 2^k inputs, which are connected to each of the outputs of the decoder.

- The 256 intersections in a 32 x 8 ROM are programmable.

- The programmable intersection between two lines is called a <u>cross point</u>.

ROM Truth Table

Table 8.1 ROM truth table (Partial)

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

- The truth table shows the five inputs under which are listed all 32 addresses.

- At each address, there is stored a word of 8 bits, which is listed under the outputs columns.

- The table shows only the first four and the last four words in the ROM.

- The complete table must include the list of all 32 words.

Programming a ROM

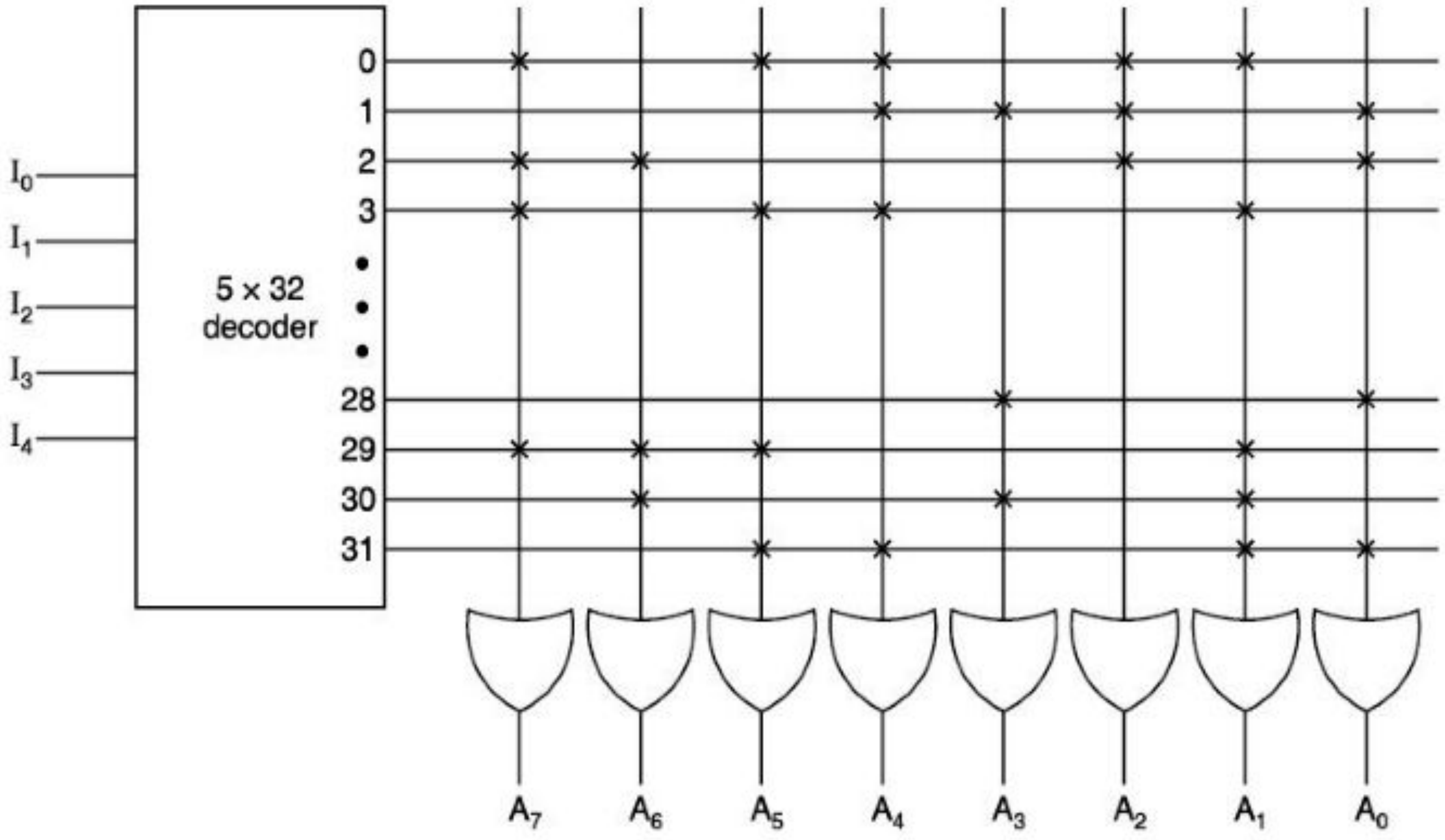


Figure 8.3 Programming the ROM according to Table 8.1.

- The hardware procedure that programs the ROM results in blowing fuse links according to the given truth table.
- Every 0 listed in the truth table specifies a no connection and every 1 listed specifies a path that is obtained by a connection.
- For example, the table specifies the 8-bit word 10110010 for permanent storage at address 3.
- The four 0s in the word are programmed by blowing the fuse links between output 3 of the decoder and the inputs of the OR gates associated with outputs A6, A3, A2 and A0.
- The four 1s in the word are marked in the diagram with a × to denote a connection in place of a dot used for permanent connection in logic diagrams.

- When the input of the ROM is 00011, all the outputs of the decoder are 0 except for output 3, which is at logic 1.

- The signal equivalent to logic 1 at decoder output 3 propagates through the connections to the OR gate outputs of A7, A5, A4 and A1.

- The other four outputs remain at 0.

- The result is that the stored word 10110010 is applied to the eight data outputs.

COMBINATIONAL CIRCUIT IMPLEMENTATION USING ROM

- The ROM is essentially a device that includes both the decoder and the OR gates within a single device.
- By choosing connections for those minterms that are included in the function, the ROM outputs can be programmed to represent the Boolean functions of the output variables in a combinational circuit.
- Each output terminal is considered separately as the output of a Boolean function expressed as a sum of minterms.

- A 32 X 8 ROM may be considered as a combinational circuit with eight outputs, each being a function of the five-input variables.

- Output A7 can be expressed in sum of minterms as

- $A7(I_4, I_3, I_2, I_1, I_0) = \sum(0, 2, 3, ..., 29)$

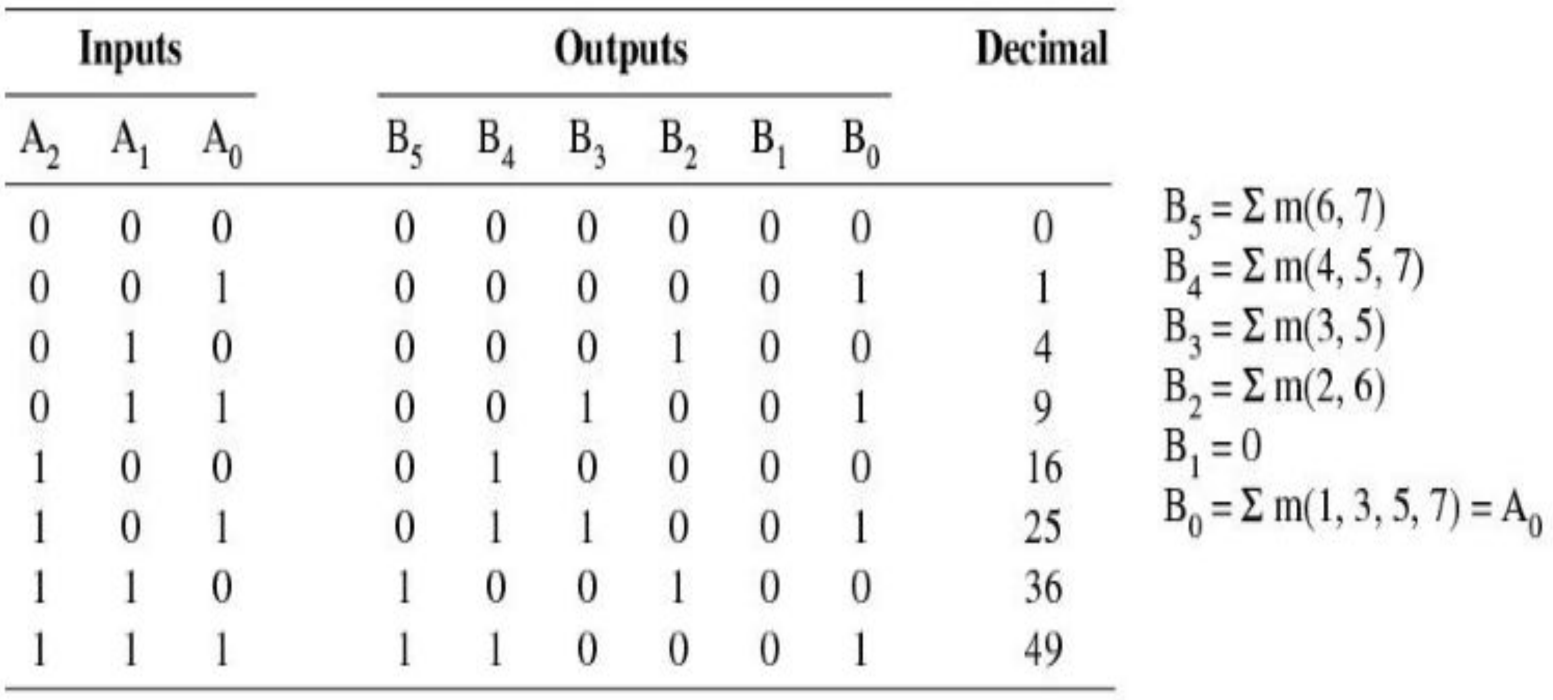
- A connection marked with × in the figure produces a minterm for the sum.

- Question

- 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.

- <u>Solution</u>
- The first step in the design is to derive the truth table of the combinational circuit.

Table 8.2 Example 8.1: Truth table for circuit



Inputs			Outputs						Decimal
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

$B_5 = \Sigma\, m(6, 7)$
$B_4 = \Sigma\, m(4, 5, 7)$
$B_3 = \Sigma\, m(3, 5)$
$B_2 = \Sigma\, m(2, 6)$
$B_1 = 0$
$B_0 = \Sigma\, m(1, 3, 5, 7) = A_0$

- Three inputs and six outputs are needed to accommodate all possible binary numbers.
- Output B_0 is always equal to input A_0; so there is no need to generate B_0 with a ROM since it is equal to an input variable.
- Output B1 is always 0, so this output is a constant.
- We actually need to generate only four outputs with the ROM; the other two are readily obtained.

- The minimum size ROM needed must have three inputs and four outputs.

- Three inputs specify eight words, so the ROM must be of size 8 × 4.

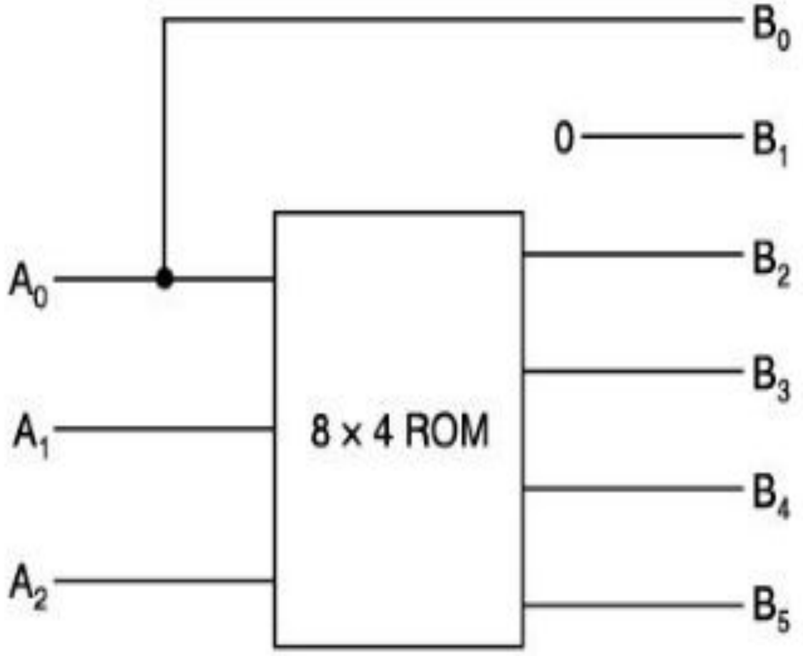
- The three inputs specify eight words of four bits each.

- The ROM truth table in Figure 8.4a specifies the information needed for programming the ROM.

- The block diagram of Figure 8.4b shows the required connections of the combinational circuit.

A_2	A_1	A_0	B_5	B_4	B_3	B_2
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

(a) ROM truth table



(b) Block diagram

Figure 8.4 Example 8.1: ROM implementation.

- <u>Question</u>

- Obtain the logic implementation of a 32×4 bit ROM using a decoder of a suitable size.

- <u>Solution</u>
- A 32 × 4 bit ROM is to be implemented.
- It consists of 32 words of four bits each.
- There must be five input lines that form the binary numbers from 0 through 31 for the address.
- The five inputs are decoded into 32 distinct outputs by means of a 5 × 32 decoder.
- Each output of the decoder represents a memory address.
- The 32 outputs of the decoder are connected to each of the four OR gates.

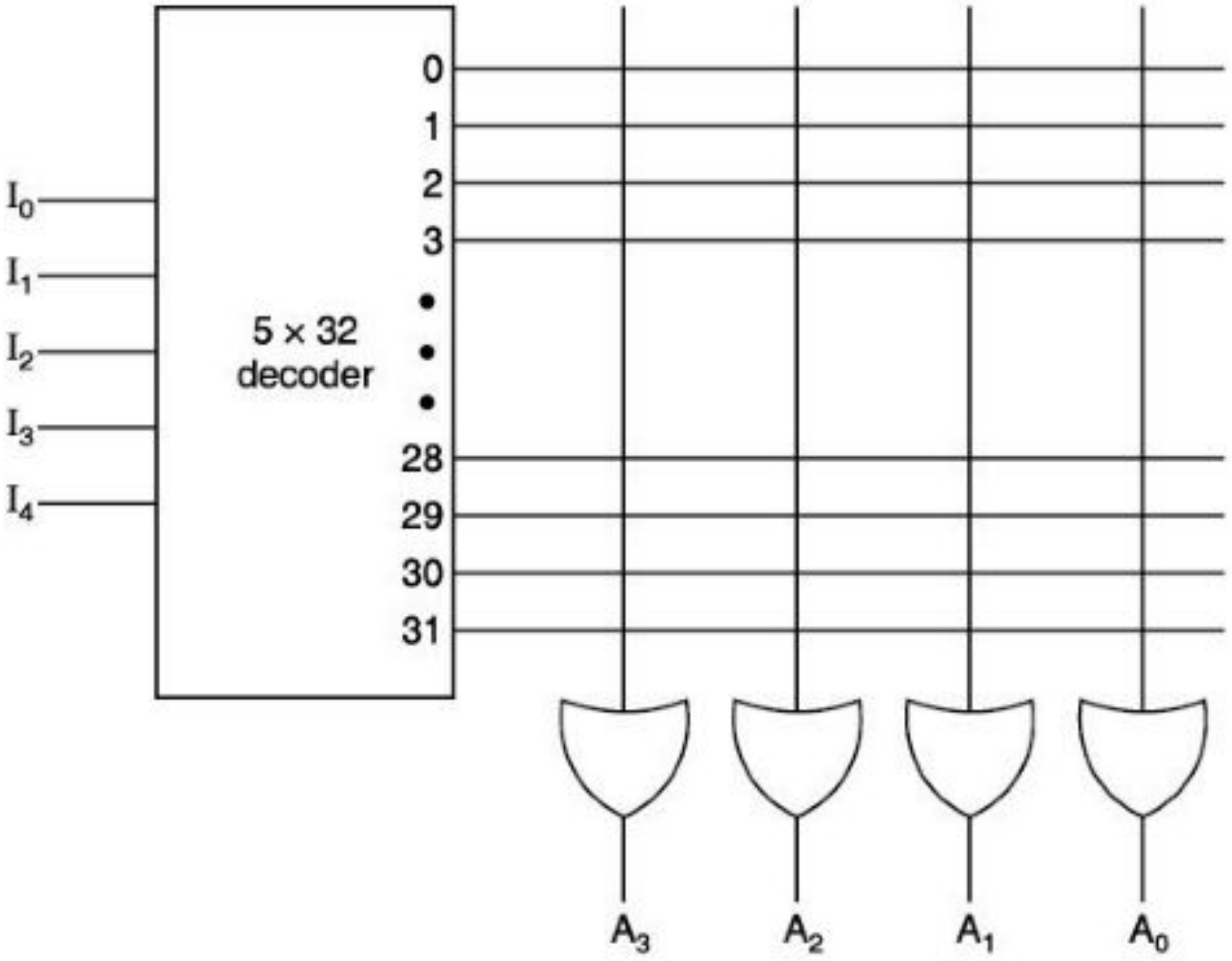


Figure 8.5 Implementation of 32 × 4 bit ROM.

- <u>Question</u>

- Obtain the logic implementation of a 8 × 4 bit ROM using a decoder of a suitable size.

- Solution
- An 8 × 4 bit ROM is to be implemented.
- It consists of eight words of four bits each.
- There must be three input lines that form the binary numbers from 0 through 7 for the address.
- The three inputs are decoded into 8 distinct outputs by means of a 3 × 8 decoder.
- Each output of the decoder represents a memory address.
- The eight outputs of the decoder are connected to each of the four OR gates.
- Figure 8.6 shows the logic implementation of the 8 × 4 ROM using a 3 × 8 decoder.

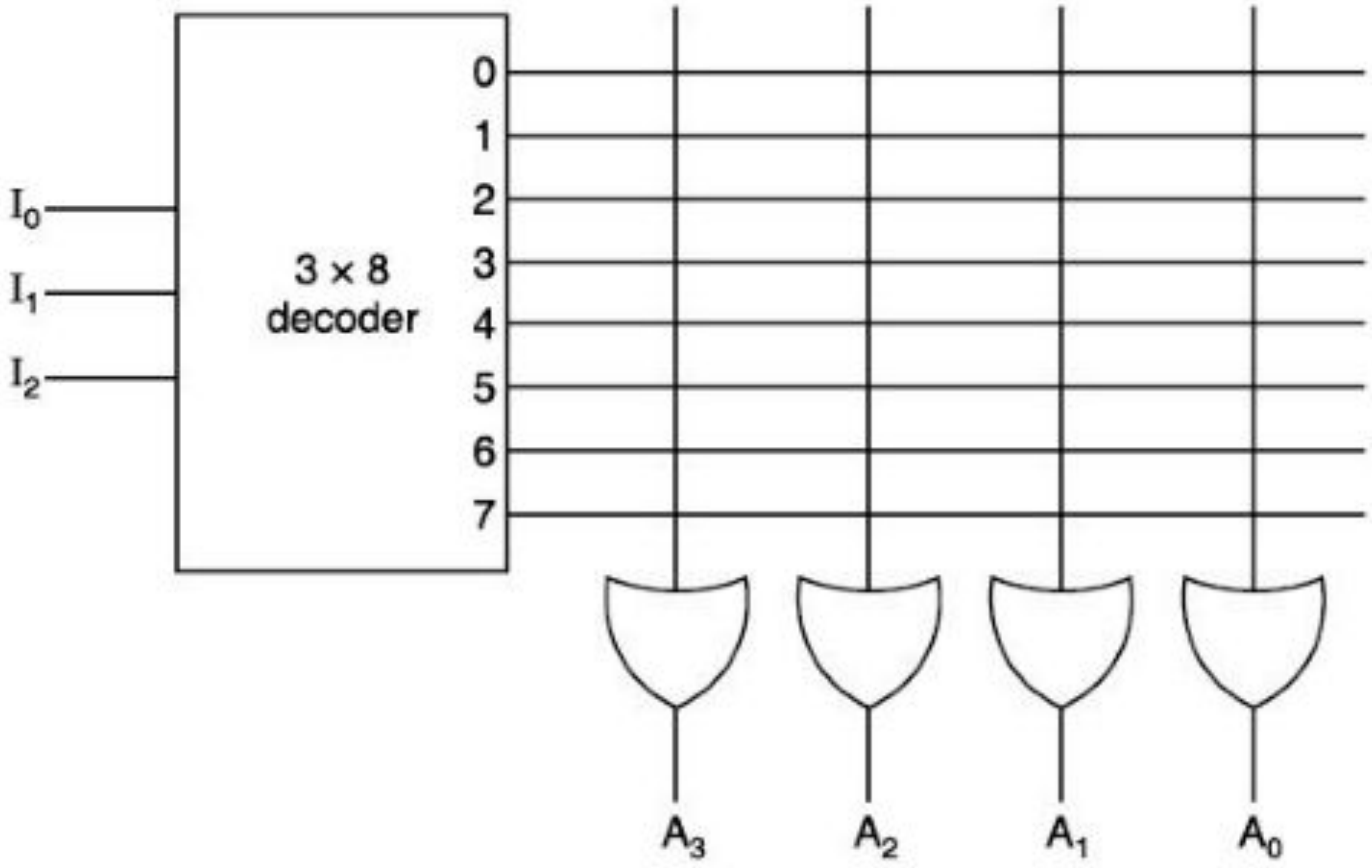


Figure 8.6 Implementation of 8 × 4 bit ROM.

COMBINATIONAL PROGRAMMABLE LOGIC DEVICES

- A combinational PLD is an integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND-OR sum of products implementation.

- There are three major types of combinational PLDs and they differ in the placement of the programmable connection in the AND-OR array.

- The various PLDs used are

- 1. PALs (programmable array logics),
- 2. PLAs (programmable logic arrays) and
- 3. PROMs (programmable read only memories).

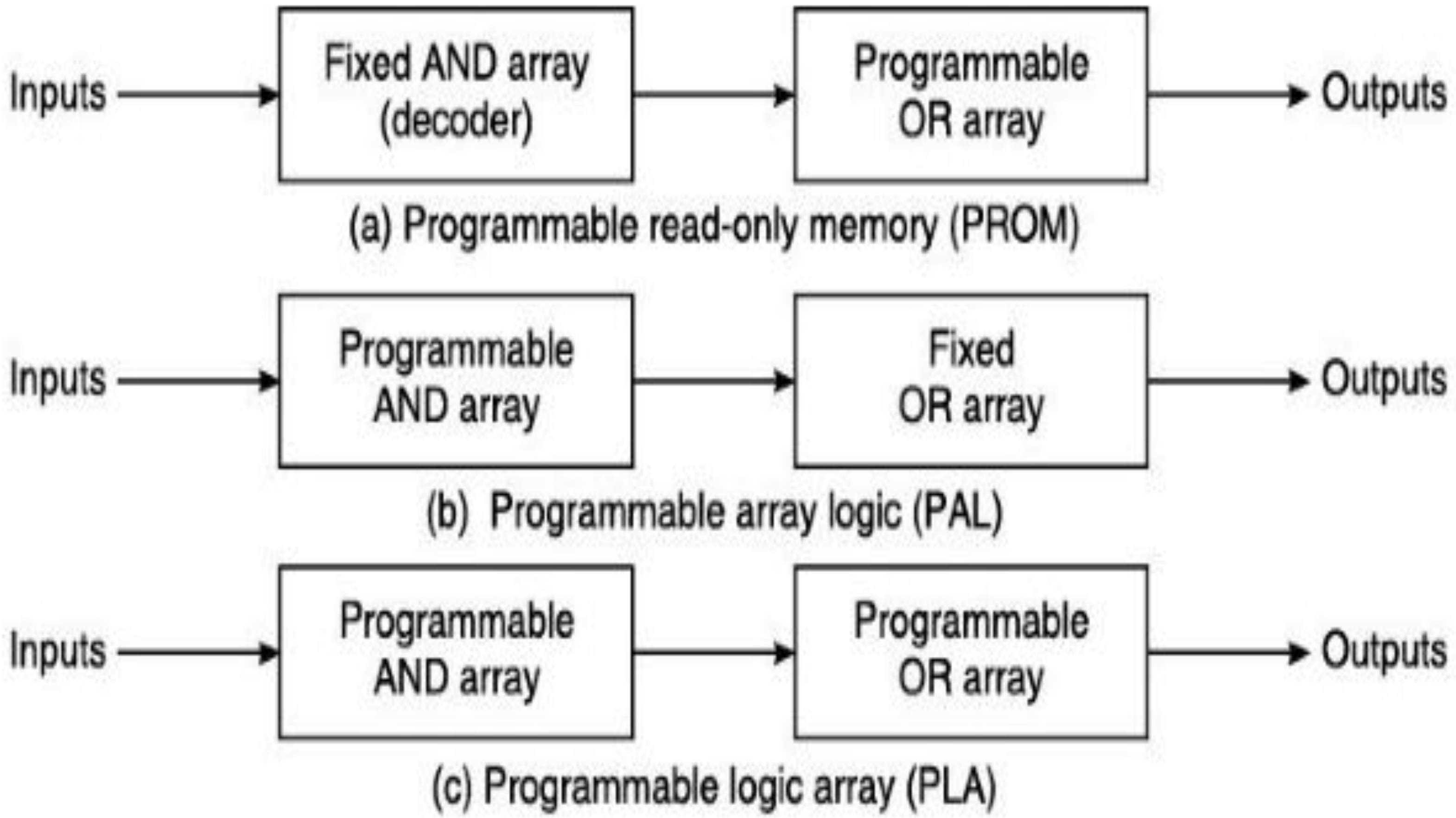


Figure 8.7 Basic configuration of three PLDs.

- The programmable read-only memory (PROM) has a fixed AND array constructed as a decoder and a programmable OR array.

- The AND gates are programmed to provide the product terms for the Boolean functions, which are logically summed in each OR gate.

- The programmable array logic (PAL) has programmable AND array and a fixed OR array.

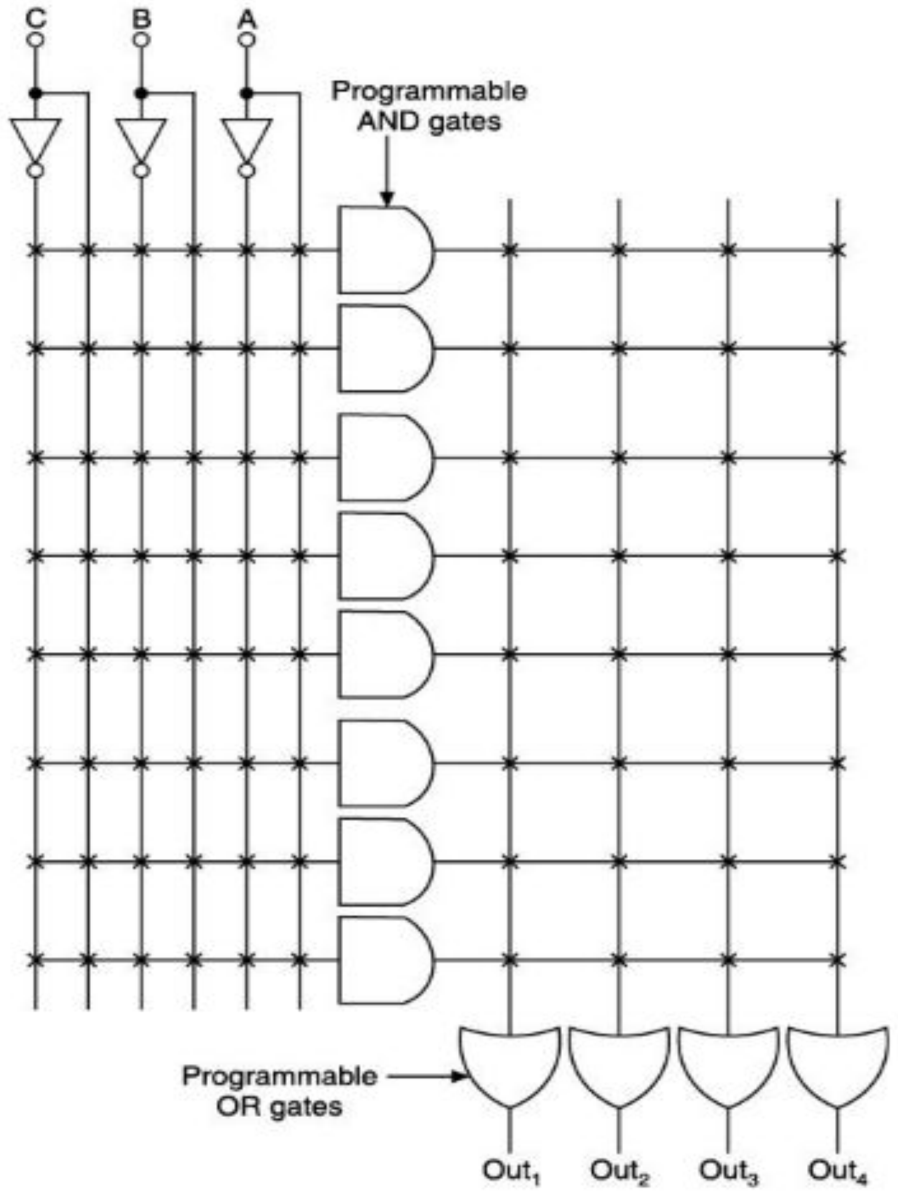
- The most flexible PLD is the programmable logic array (PLA) where both the AND and OR arrays can be programmed.

- The product terms in the AND array may be shared by any OR gate to provide the required sum of products implementation.

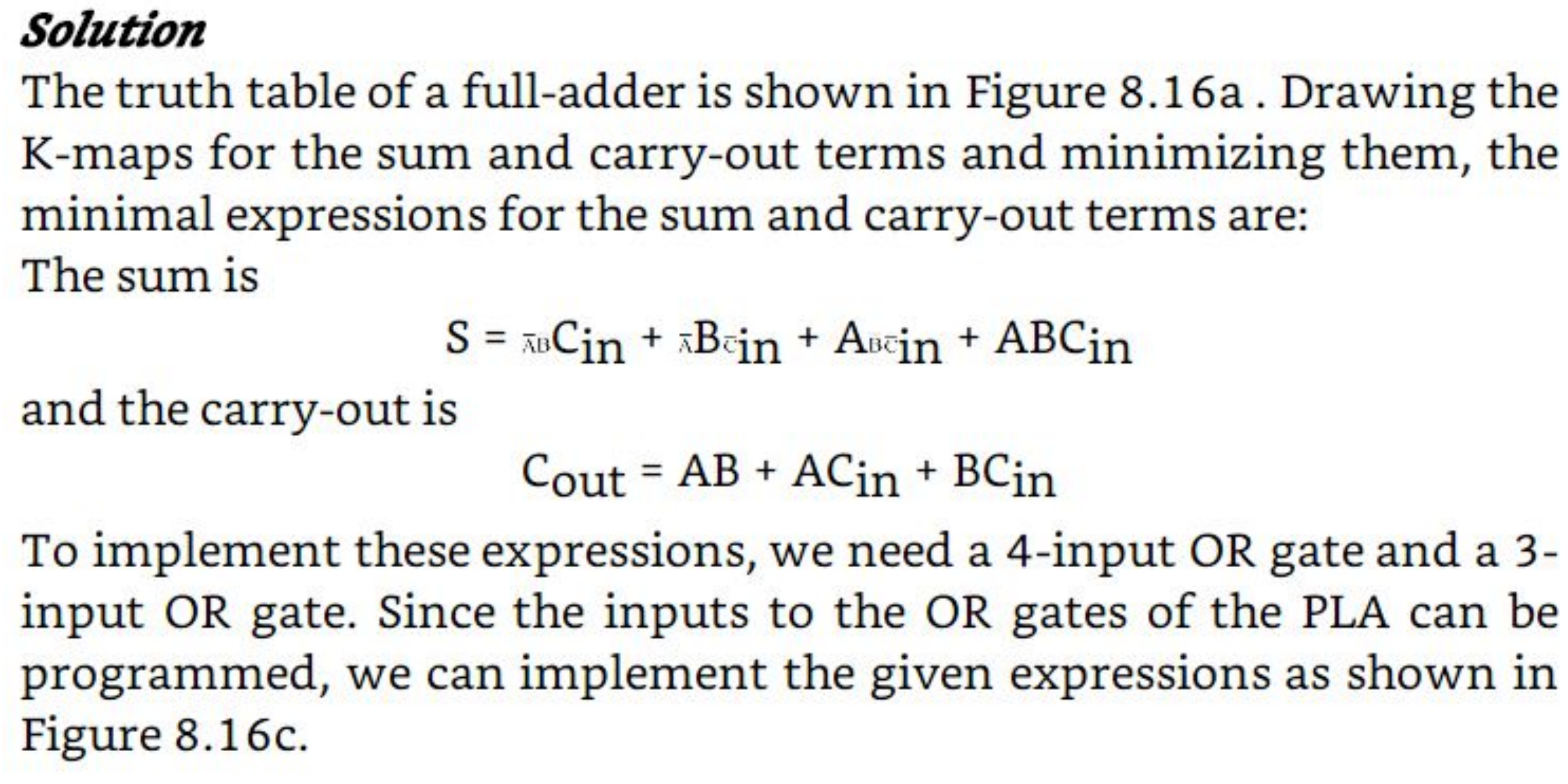
PROGRAMMABLE LOGIC ARRAY (PLA)

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

- i.e. in a PLA both AND gates and OR gates have fuses at the inputs.


C
B
A
Programmable AND gates
Programmable OR gates
Out$_1$
Out$_2$
Out$_3$
Out$_4$

- <u>Question</u>
- Show how the PLA circuit would be programmed to implement the sum and carry outputs of a full adder.



Solution

The truth table of a full-adder is shown in Figure 8.16a . Drawing the K-maps for the sum and carry-out terms and minimizing them, the minimal expressions for the sum and carry-out terms are:

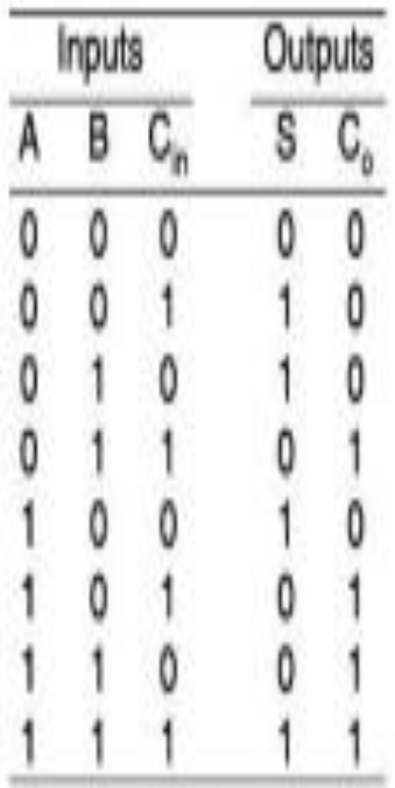
The sum is

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

and the carry-out is

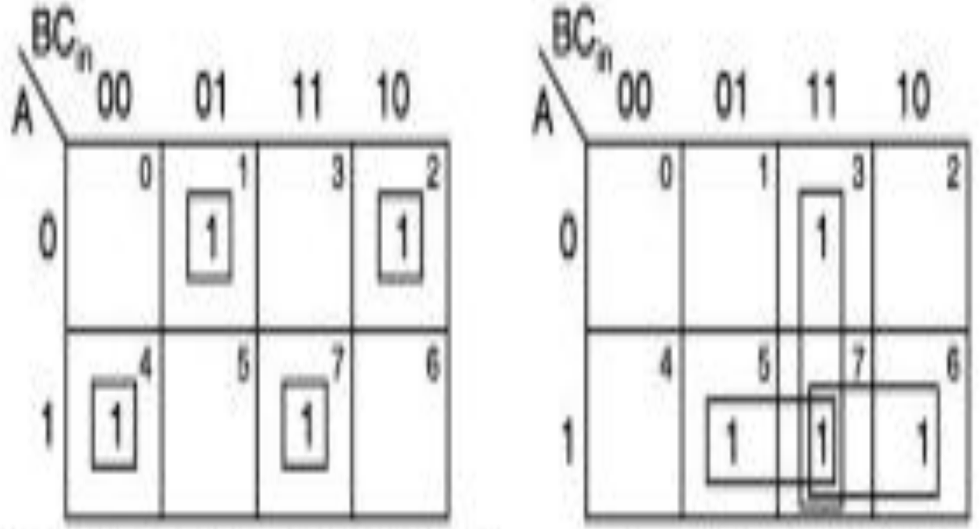
$$C_{out} = AB + AC_{in} + BC_{in}$$

To implement these expressions, we need a 4-input OR gate and a 3-input OR gate. Since the inputs to the OR gates of the PLA can be programmed, we can implement the given expressions as shown in Figure 8.16c.



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

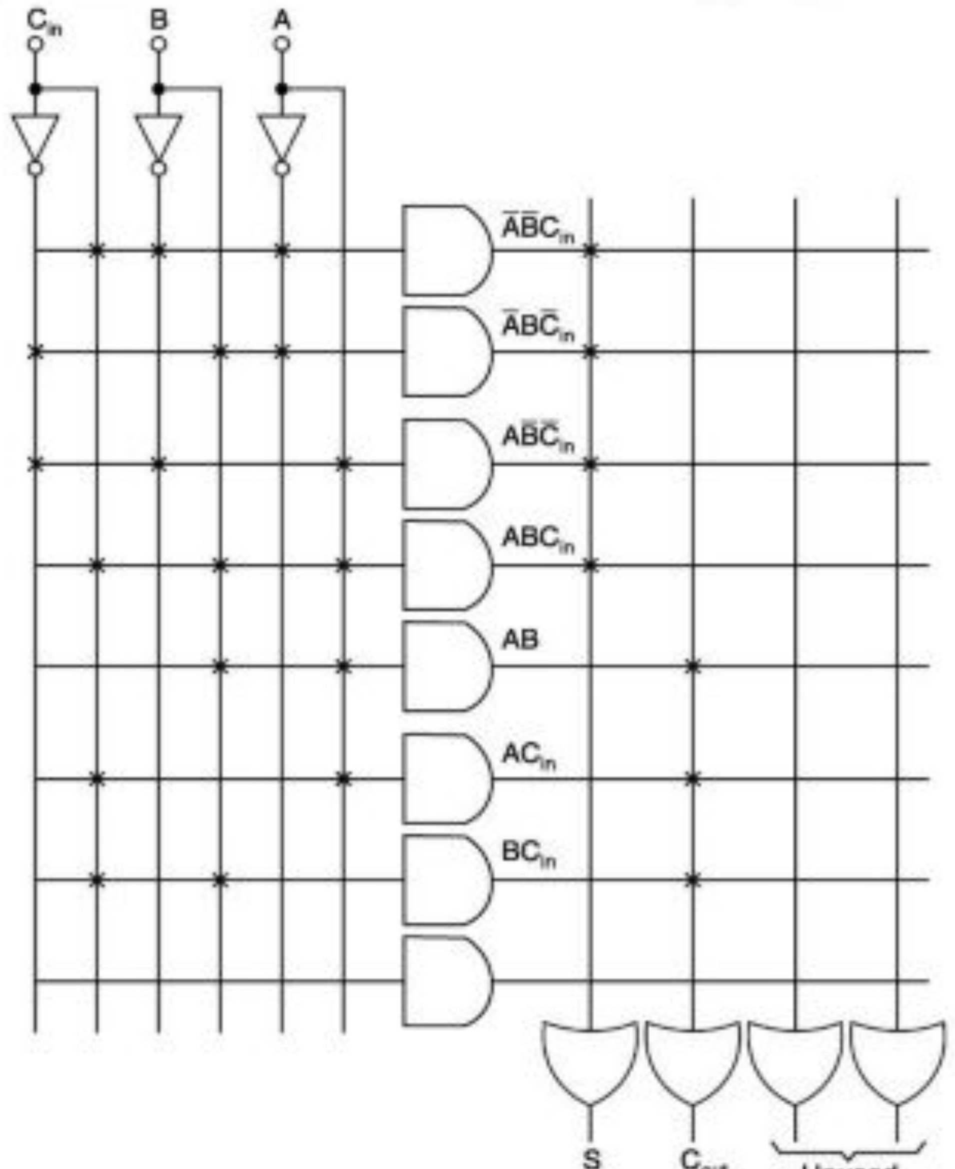
(a) Truth table



$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$

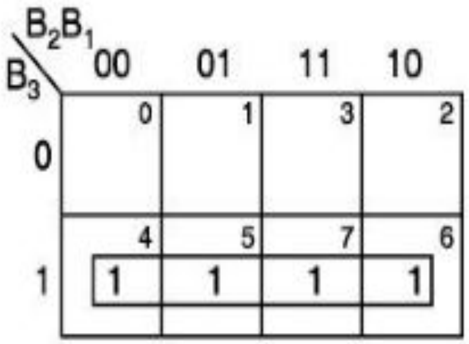
$C_{out} = AC_{in} + BC_{in} + AB$

(b) K-maps

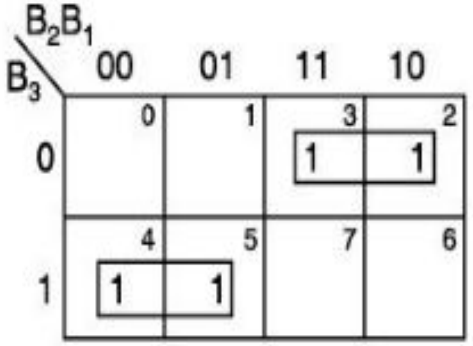

C_{in}
B
A
$\overline{A}\overline{B}C_{in}$
$\overline{A}B\overline{C}_{in}$
$A\overline{B}\overline{C}_{in}$
ABC_{in}
AB
AC_{in}
BC_{in}
S
C_{out}
Unused

(c) Implementation of full-adder

Qn: Show how the PLA circuit can be programmed to implement the 3-bit binary-to-gray conversion.



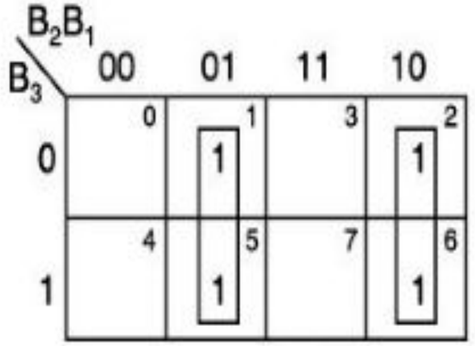
$G_3 = B_3$



$G_2 = B_3\overline{B}_2 + \overline{B}_3B_2$

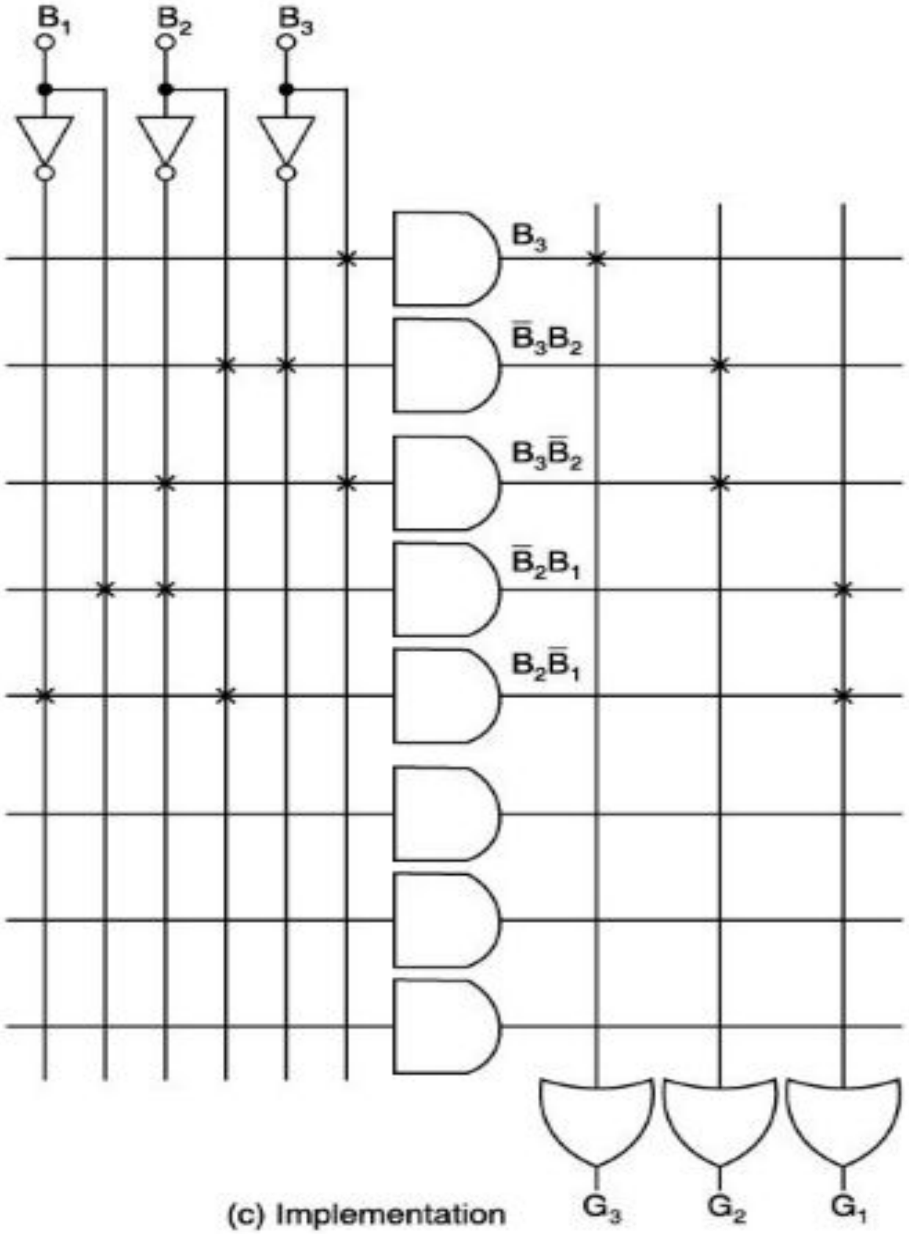
Binary			Gray		
B_3	B_2	B_1	G_3	G_2	G_1
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

(a) Conversion table



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

(b) K-maps



(c) Implementation

PLA Programming Table

- The fuse map of a PLA can be specified in a tabular form.
- The PLA programming table consists of three columns.
- The first column lists the product term numerically.
- The second column specifies the required paths between inputs and AND gates.
- The third column specifies the paths between the AND and OR gates.

- For each output variable, we may have a T (for true) or C (for complement) for programming the X-OR gate.
- The product terms listed on the left are not part of the table; they are included for reference only.
- For each product term the inputs are marked with 1, 0, or – (dash).
- If a variable in the product term appears in its true form, the corresponding input variable is marked with a 1.
- If it appears complemented, the corresponding input variable is marked with a 0.
- If the variable is absent in the product term, it is marked as a – (dash).

- The paths between the inputs and the AND gates are specified under the column heading inputs in the programming table.
- A 1 in the input column specifies a connection from the input variable to the AND gate.
- A 0 in the input column specifies a connection from the complement of the variable to the input of the AND gate.
- A – (dash) specifies a blown fuse in both the input variable and its complement.
- It is assumed that an open terminal in the input of an AND gate behaves like a 1.

- The paths between the AND and OR gates are specified under the column heading outputs.
- The output variables are marked with 1s for those product terms that are included in the function.
- Each product term that has a 1 in the output column requires a path from the output of the AND gate to the input of the OR gate.
- Those marked with a – (dash) specify a blown fuse.
- It is assumed that an open terminal in the input of an OR gate behaves like a 0.
- Finally, a T (true) output dictates that the other input of the corresponding X-OR gate be connected to 0, and a C (complement) specifies a connection to 1.

- The size of a PLA is specified by the number of inputs, the number of product terms, and the number of outputs.

- <u>For n inputs, k product terms, and m outputs the internal logic of the PLA consists of n buffer inverter gates, k AND gates, m OR gates, and m X-OR gates.</u>

- When designing a digital system with a PLA, there is no need to show the internal connections of the unit .

- All that is needed is a PLA programming table from which the PLA can be programmed to supply the required logic.

- Consider The implementation of the Boolean functions

$$F_1 = {}_{A}B + A_{c} + {}_{A}B_{c}$$

$$F_2 = \overline{A\bar{C} + \bar{B}C}$$

Product term		Inputs			Outputs (T)	Outputs (C)
		A	B	C	F	F
1	$\bar{A}B$	0	1	–	1	–
2	$A\bar{C}$	1	–	0	1	1
3	$\bar{A}B\bar{C}$	0	1	0	1	–
4	$\bar{B}C$	–	0	1	–	1

(a) PLA programming table

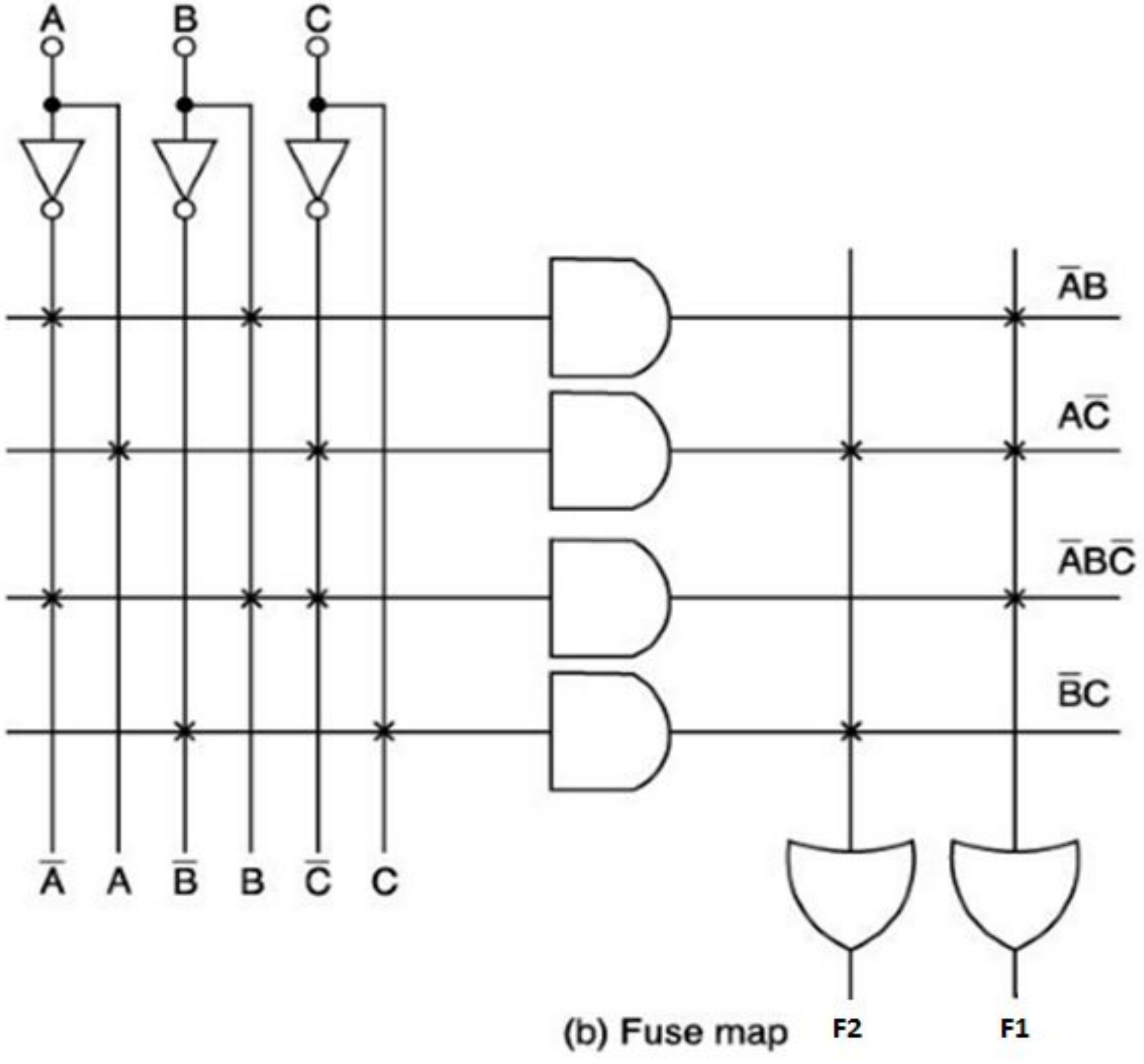
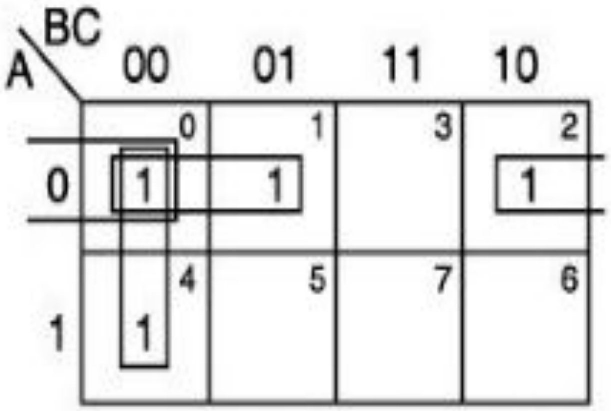


Figure 8.19 Programmable logic array.

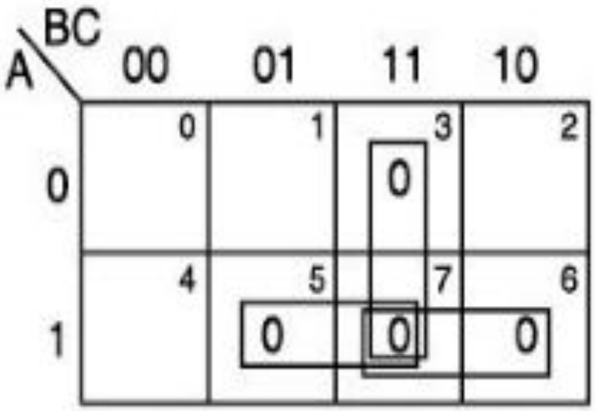
- <u>Question</u>

- Implement the following two Boolean functions with a PLA:

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



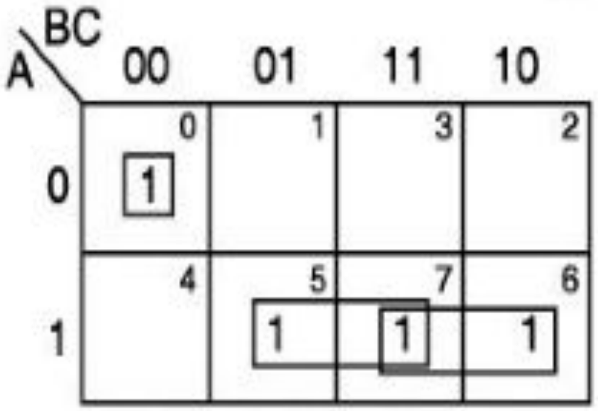
$F_1(T) = \bar{A}\bar{C} + \bar{B}\bar{C} + \bar{A}\bar{B}$



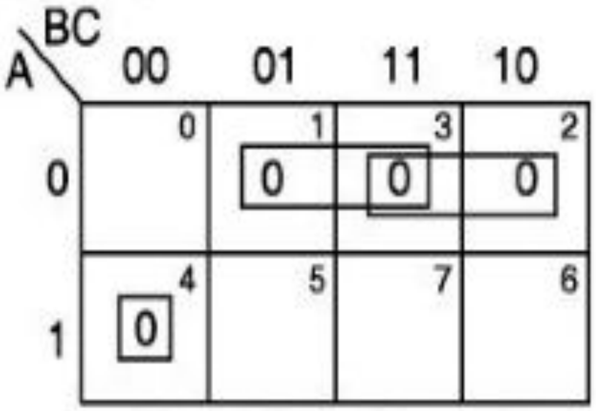
$\bar{F}_1 = AB + AC + BC$

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

(a) K-map for F_1



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



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

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

(b) K-map for F_2

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 8.20. 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

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

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

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

Considering the 0s of F_2

$$\bar{F}2 = A\text{B}\bar{C} + \bar{A}B + \bar{A}C$$

Therefore,

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

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

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

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

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

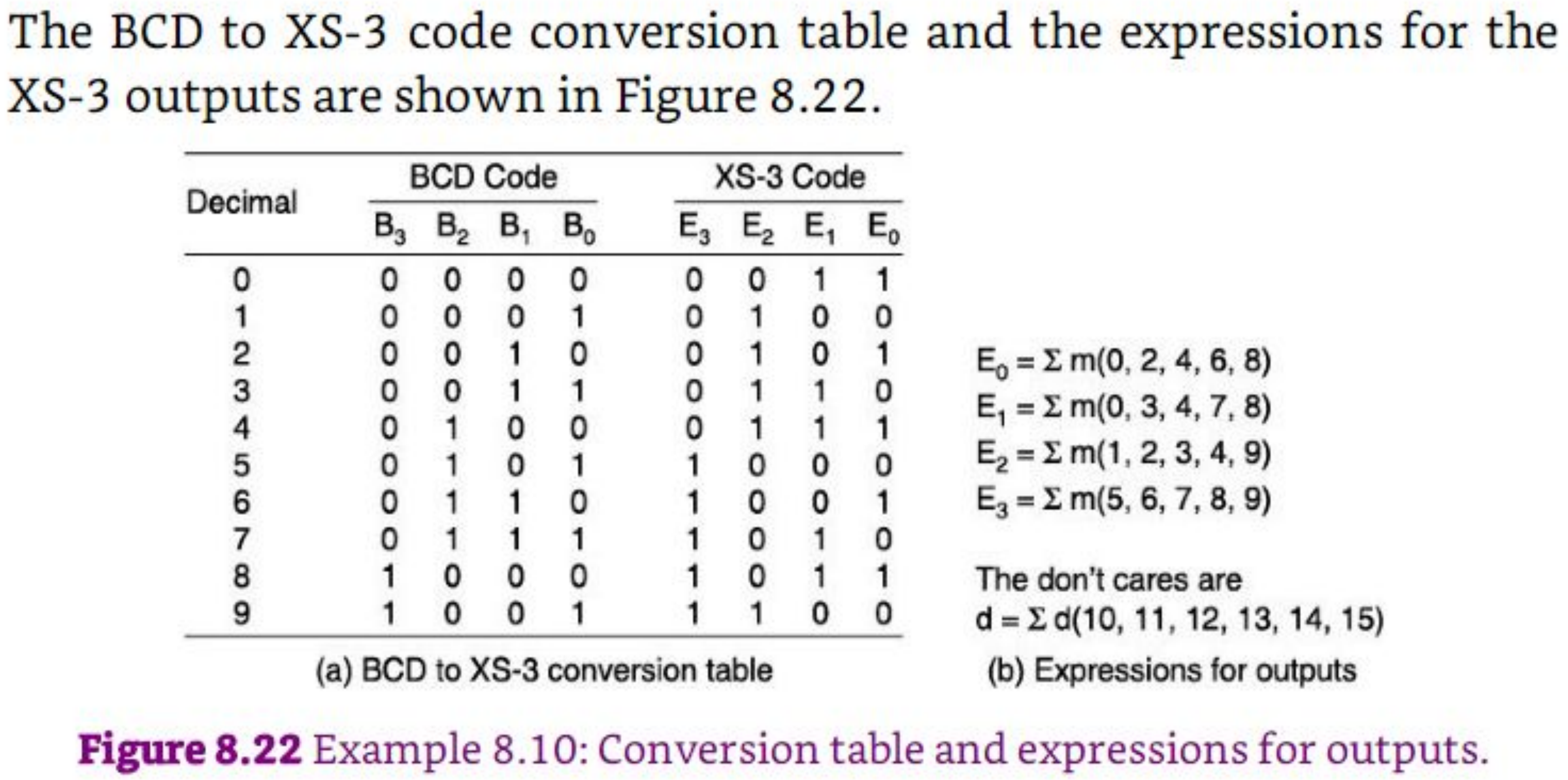
Figure 8.20 Example 8.9: K-maps.

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

Figure 8.21 Example 8.9: Programming table and fuse map (Contd.)

- <u>Question</u>
- Write the program table to implement a BCD to XS-3 code conversion using a PLA.



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

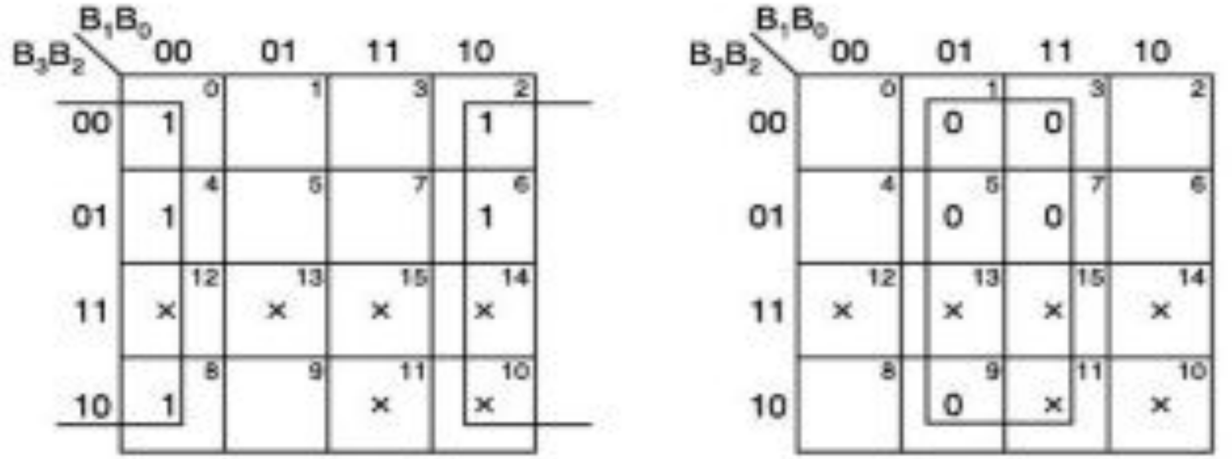
(a) BCD to XS-3 conversion table

$E_0 = \Sigma m(0, 2, 4, 6, 8)$
$E_1 = \Sigma m(0, 3, 4, 7, 8)$
$E_2 = \Sigma m(1, 2, 3, 4, 9)$
$E_3 = \Sigma m(5, 6, 7, 8, 9)$

The don't cares are
$d = \Sigma d(10, 11, 12, 13, 14, 15)$

(b) Expressions for outputs

Figure 8.22 Example 8.10: Conversion table and expressions for outputs.

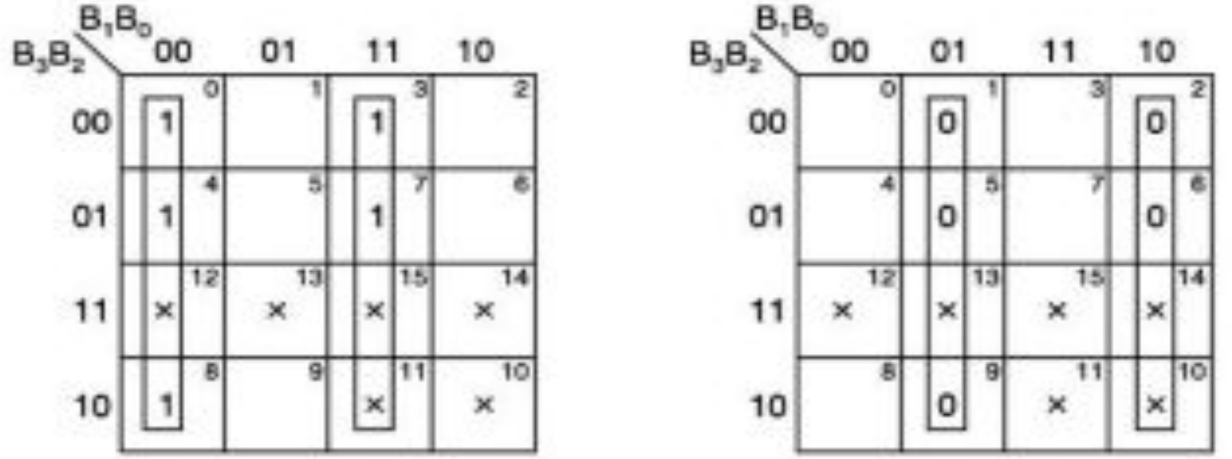


$E_0(T) = \overline{B}_0$

$\overline{E}_0 = B_0$

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

(a) K-maps for E_0

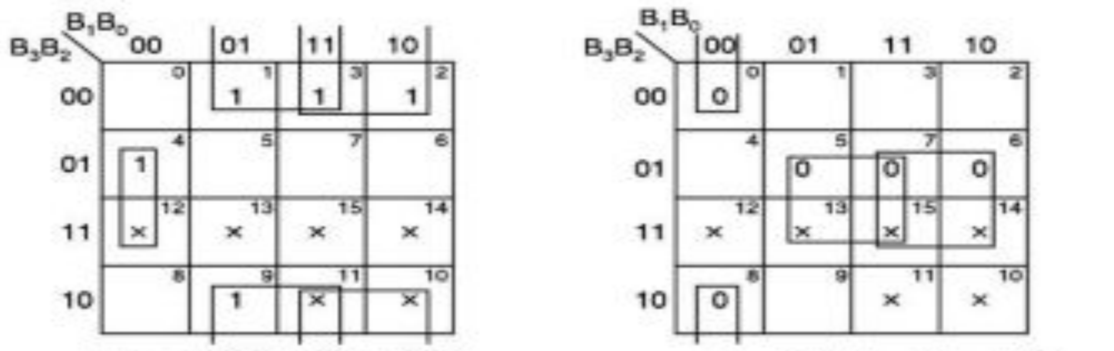


$E_1(T) = \overline{B}_1\overline{B}_0 + B_1B_0$

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

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

(b) K-maps for E_1

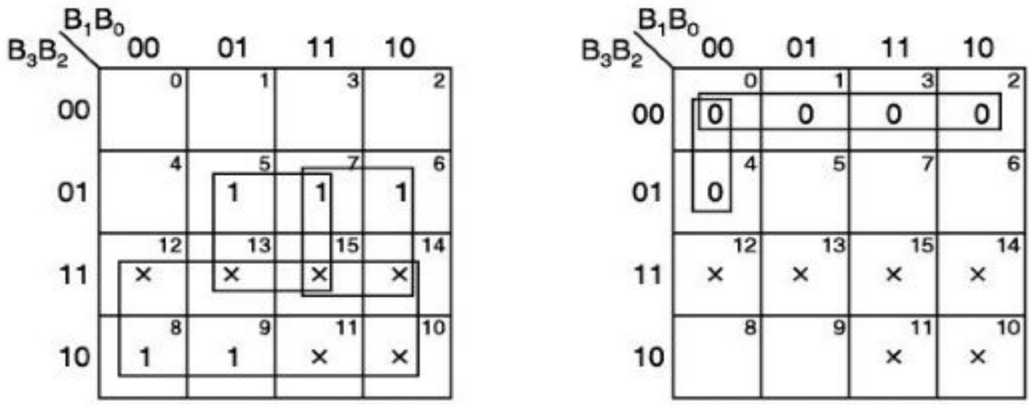


$E_2(T) = B_2\bar{B}_1\bar{B}_0 + \bar{B}_2B_0 + \bar{B}_2B_1$

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

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

(c) K-maps for E_2



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

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

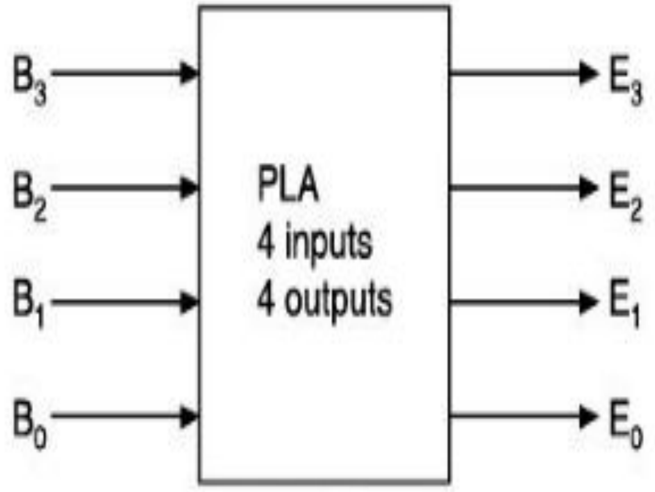
$E_3(C) = \overline{\bar{B}_3\bar{B}_2 + \bar{B}_3\bar{B}_1\bar{B}_0}$

(d) K-maps for E_3

Figure 8.23 Example 8.10: K-maps.

Observing the expressions in true and complement form for E_0, E_1, E_2 and E_3, we notice that two terms in $E_2(C)$ and $E_3(T)$ are common. So minimal combinations are as follows:

$E_0(T) = B_0$, $E_1(T) = B_1B_0 + B_1B_0$, $E_2(C) = \overline{\bar{B}_2\bar{B}_1\bar{B}_0 + B_2B_0 + B_2B_1}$, $E_3(T) = B_3 + B_2B_0 + B_2B_1$



(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	$\overline{B}_2\overline{B}_1\overline{B}_0$	–	0	0	0	–	1	–	–
5	$\overline{B}_1\overline{B}_0$	–	–	0	0	–	–	1	–
6	B_1B_0	–	–	1	1	–	–	1	–
7	$\overline{B}_0$	–	–	–	0	–	–	–	1

(b) PLA programming table