PROGRAMMABLE LOGIC DEVICES

Programmable logic devices (PLDs) are used for designing logic circuits. PLDs can be configured by the user to perform specific functions. The different types of PLDs available are:

- ROM and EPROM.
- Programmable Logic Arrays (PLA)
- Programmable Array Logic (PAL)
- Field Programmable Gate Arrays (FPGA)

ROM

ROM consists of an array of semiconductor devices interconnected to store an array of memory data. Data can be only read, it cannot be changed under normal operating conditions.

TYPES OF ROM.

- Mask programmable ROM
- Erasable programmable ROM (EPROM)
- Electrically erasable PROM (EEPROM)
- Flash

BASIC ROM STRUCTURE

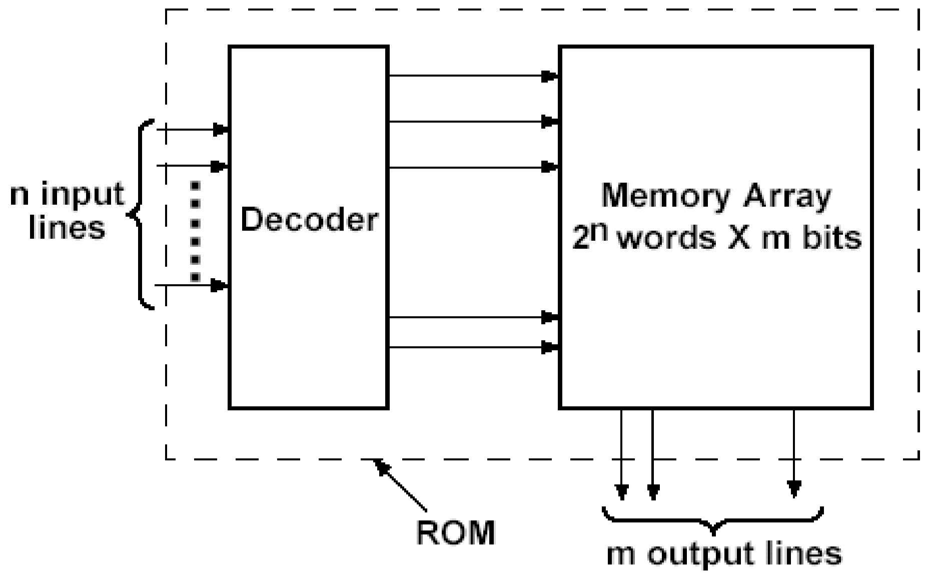


Fig.1 ROM Block diagram

A block diagram of the ROM is shown in the figure 1. There are n inputs and m outputs. The inputs provide the address for the memory, and the outputs give the data bits of the stored word that is selected from the address. The number of words in the ROM device is determined from the fact that n address input lines can specify 2^n words. For example a 4 x 8 ROM has 2 (ie., n=2) Input Address lines. & Can store 4 words each of 8 (ie., m=8) bits.

Internally a ROM consists of a decoder and a memory array. When a pattern of n 0s and 1s is applied to the decoder inputs, exactly one of the 2^n-decoder outputs is '1'. This decoder output line selects one of the words in the memory array, and the bit pattern stored in this word is transferred to the memory output lines. 2^n x m ROM can realize m functions of n variables , since it can store a truth table with 2^n rows and m columns.

Note: ROM doesn't have data inputs, because it doesn't have write operation.

TYPES OF ROM

Four technologies are used for **ROM** programming.

Mask programmable ROM

If mask programming is used, then the data array is permanently stored at the time of manufacture. Preparation of the mask is expensive, so mask programmable ROMs are economically feasible if large quantity are required within the same data array.

Erasable programmable ROM (EPROM)

If a small quantity of ROMs are required within a given data array, then **EPROMs** may be used. EPROMs allow the modification of the data stored as they use a special charge storage mechanism to enable or disable the switching elements in the memory array. The data stored in the EPROM is generally permanent until erased using ultraviolet light.

EEPROM

The electrically erasable PROM **(EEPROM)** is similar to EPROM except that the erasure of data is accomplished using electrical pulses instead of ultraviolet light. An EEPROM can be erased and reprogrammed only a limited number of times**.**

Flash

Flash memories are similar to EEPROMs except that they use a different charge storage mechanism. They also have built in programming and erase capability so that the data can be written to the flash memory while it is in place in a circuit without the need for a separate programmer.

Realization of a Sequential Network with ROM

A sequential network can easily be designed using a ROM and flip-flops. The combinational part of the sequential network can be realized using a ROM. The ROM can be used to realize the output functions and the next state functions. The state of the network can then be stored in a register of D flip-flops and fed back to the input of the ROM.

Note:

Use of D flip flops is preferable to J-K flip flops, since use of 2 input flip flops would require increasing the number of outputs. The fact that the D flip flop input equations would generally require more gates than the J-K equations is of no consequence, since the size of the ROM depends only on the number of inputs and outputs and not on the complexity of the equations being realized. For this reason, the state assignment used is also of little importance, and generally a state assignment in straight binary order is as good as any.

Recap:

MEALY SEQUENTIAL MACHINE

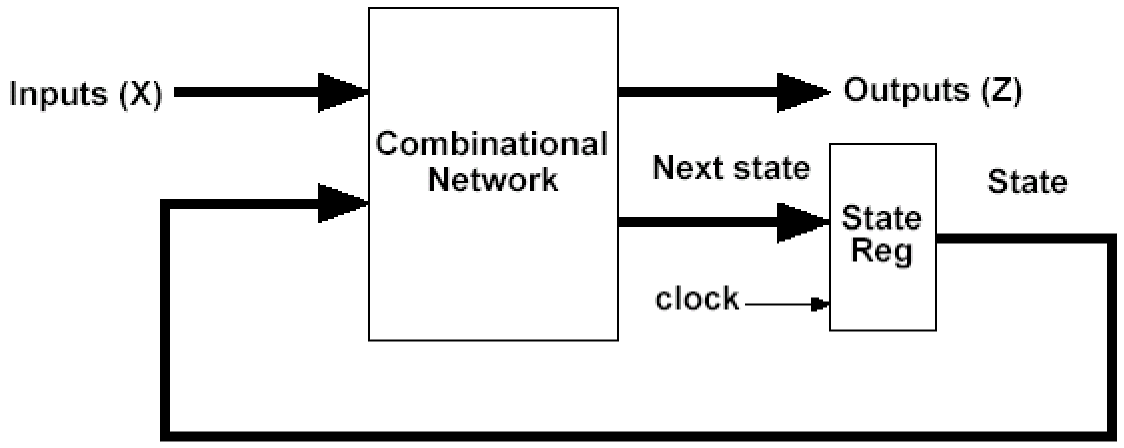
- **All programmable logic designs can be specified in Boolean form. However some designs are easier to conceptualize and implement using non-Boolean models. The State Machine model is one such model. A state machine represents a system as a set of states, the transitions between them, along with the associated inputs and outputs.**

- So, a state machine is a particular conceptualization of a particular sequential circuit. State machines can be used for many other things beyond logic design and computer architecture.
- Any circuit with memory is a finite state machine
  - Even computers can be viewed as huge FSMs
- Design of FSMs involves
  - Defining states
  - Defining transitions between states
  - Optimization / minimization
- Above approach is practical for small FSMs only

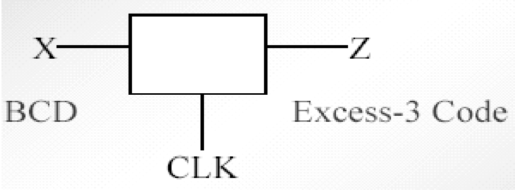
The Mealy State Machine generates outputs based on:

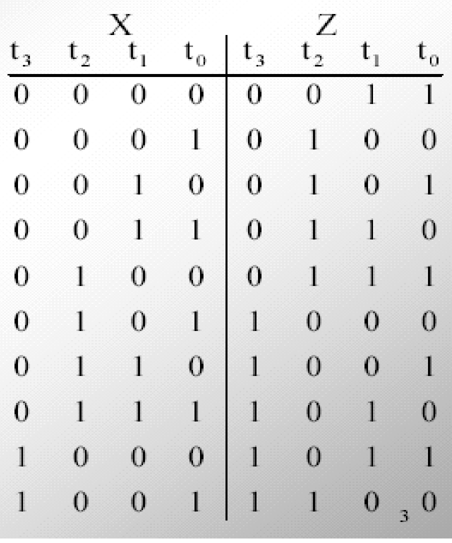
- The Present State, and
- The Inputs to the M/c.

So, it is capable of generating many different patterns of output signals for the same state, depending on the inputs present on the clock cycle.



MEALY FSM EXAMPLE : BCD to excess 3 code converter





X				Z			
t_3	t_2	t_1	t_0	t_3	t_2	t_1	t_0
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

Figure 3 shows the implementation of BCD to Excess-3 converter using combinational logic and D flip-flops

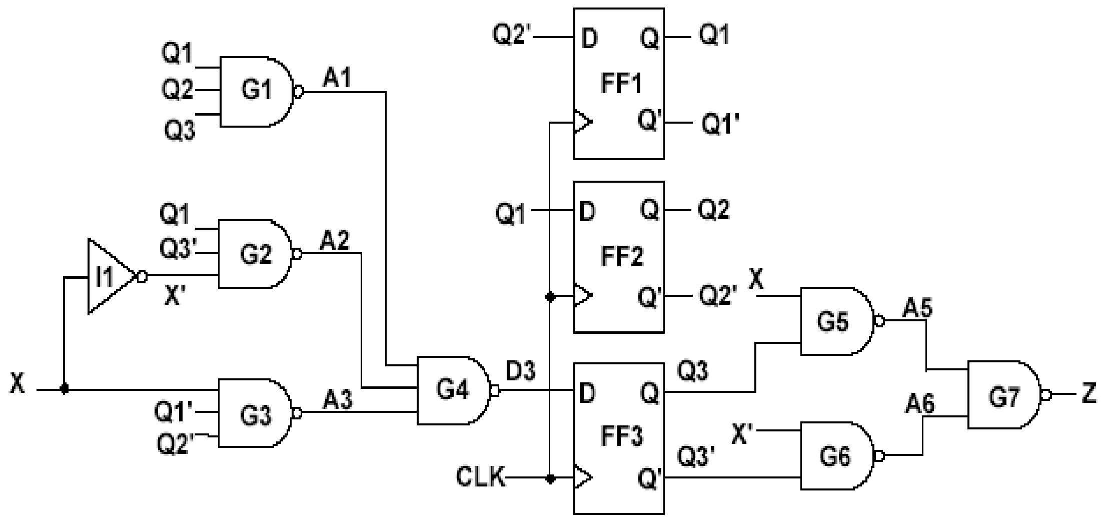


Fig 3 – Realization of MEALY Sequential Network for BCD to Excess3 with gates and flip-flops

We can realize the same sequential machine for a BCD to excess-3 code converter as shown in fig-3 using a ROM and three D flip flops, which is as shown in Fig.4. Table shown in figures 5 gives the truth table for the ROM, which implements the transition of fig.6 with the don't cares replaced by 0s. Since the ROM has four inputs, it contains 2^4 = 16 words. In general, a mealy sequential network with i inputs, j outputs and k state variables can be realized using k D flip-flops and a ROM with i+k inputs (2^{i+k} words) and j+k outputs.

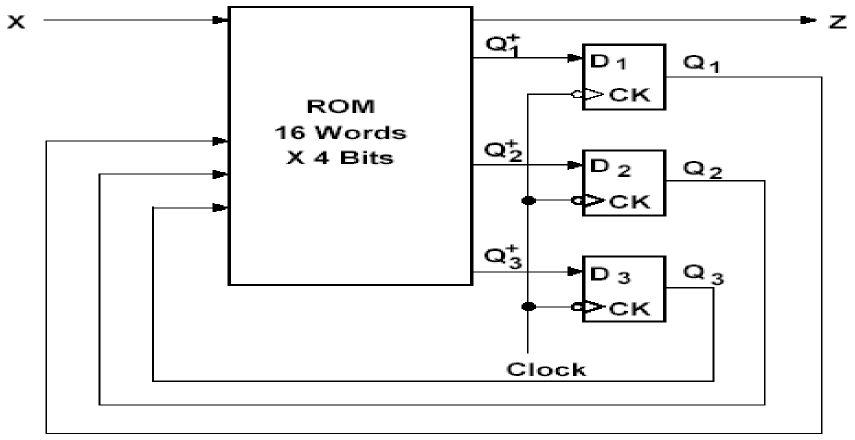
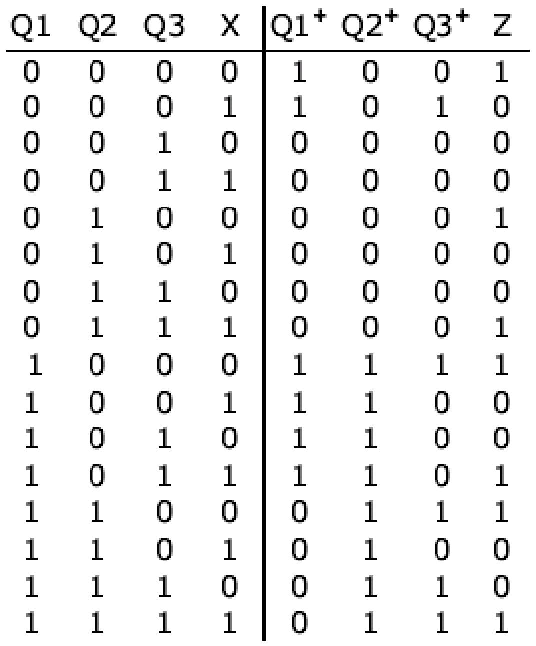
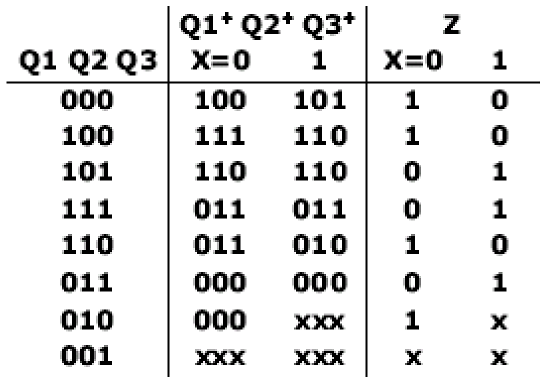


Fig 4 – Realization of MEALY Sequential Network for BCD to Excess3 with a ROM



Q1	Q2	Q3	X	$Q1^+$	$Q2^+$	$Q3^+$	Z
0	0	0	0	1	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	0	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	0	1	1	1
1	1	0	1	0	1	0	0
1	1	1	0	0	1	1	0
1	1	1	1	0	1	1	1

Fig 5 – Truth Table of ROM



	$Q1^+ Q2^+ Q3^+$		Z	
Q1 Q2 Q3	X=0	1	X=0	1
000	100	101	1	0
100	111	110	1	0
101	110	110	0	1
111	011	011	0	1
110	011	010	1	0
011	000	000	0	1
010	000	xxx	1	x
001	xxx	xxx	x	x

Fig 6 State table of BCD to Excess

Programmable Logic Array

A programmable logic array (PLA) performs the same basic function as a ROM. It is the most flexible device in the family of PLDs. The internal organization of a PLA is different from that of the ROM. The decoder of the ROM is replaced with an AND array that realizes elected product terms of the input variables. The AND array is followed by an OR array which Ors together the product terms needed to form the output functions. Both the AND and the OR arrays are programmable giving a lot of flexibility for implementing logic design.

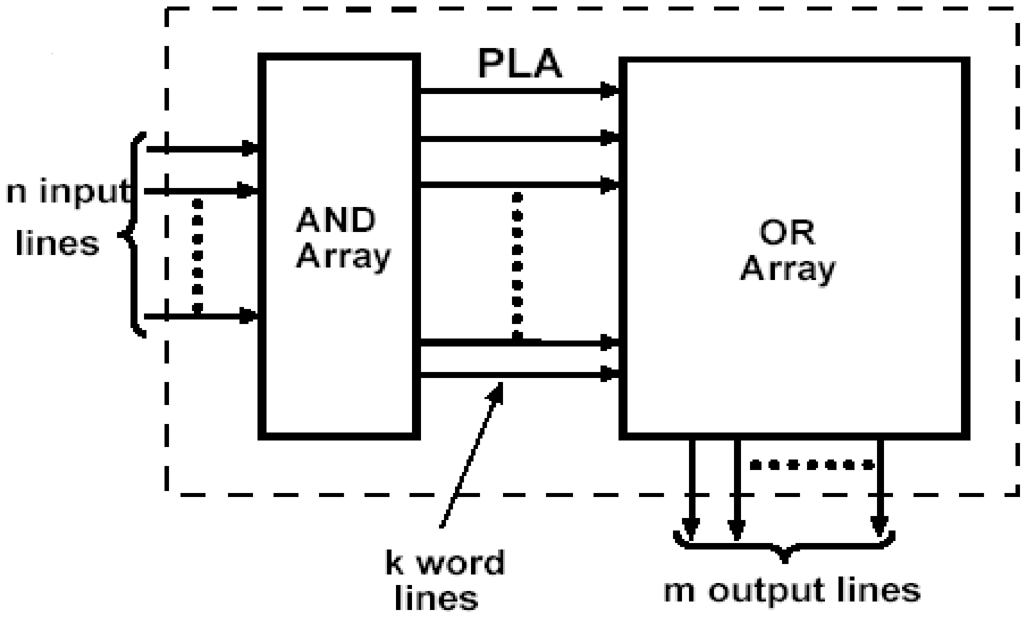


Fig 8 – Internal Logic Diagram of a PLA

Internally the PLA uses NOR-NOR logic but the added input and output inverting buffers make it equivalent to AND-OR logic. Logic gates are formed in the array by connecting NMOS switching transistors between the column line and row line. The transistors act as switches, so if the gate input is a logic zero, the transistor is turned off whereas if the gate input is a logic one, the transistor provides a conducting path to ground.

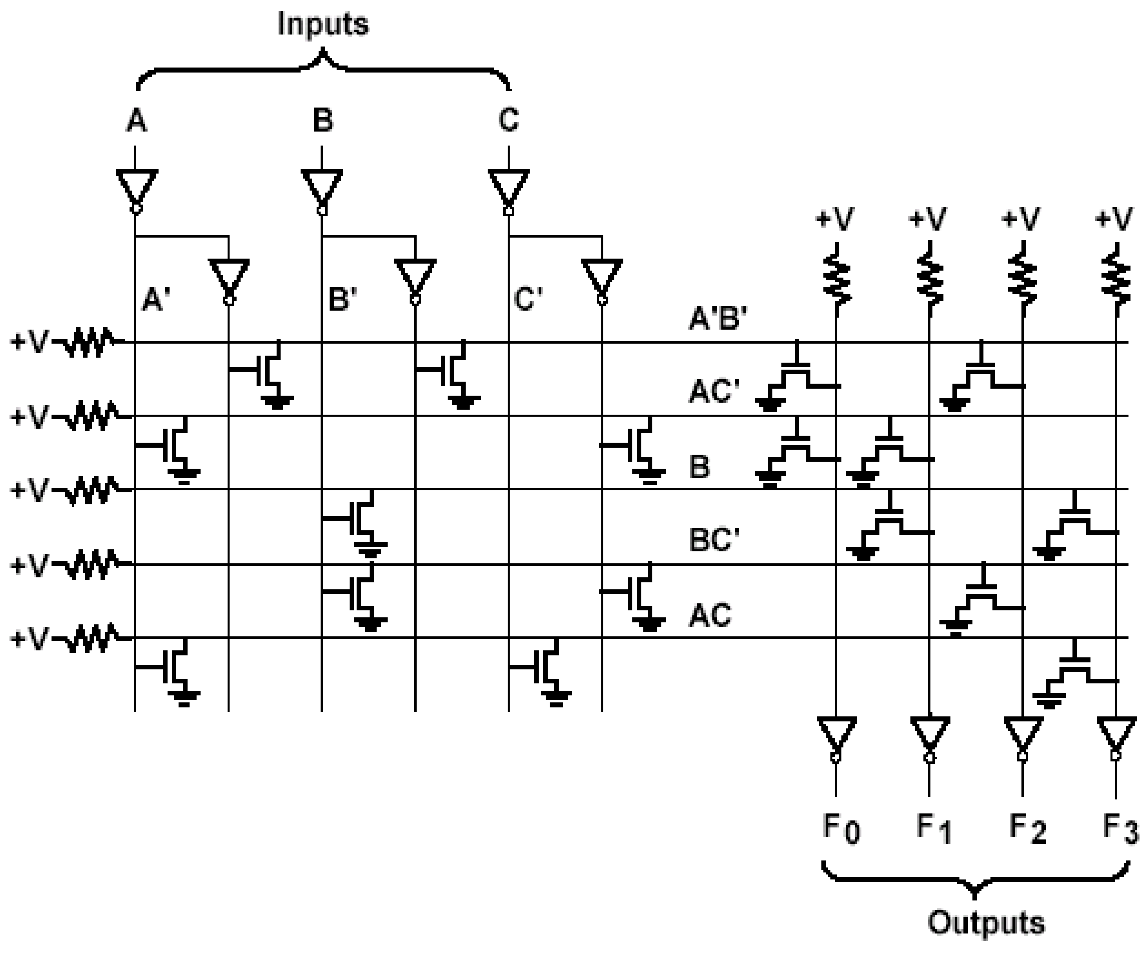


Fig 9 –PLA with 3 inputs , 5 product terms and 4 outputs

$$F0 = \sum m(0,1,4,6) = A'B'+AC'$$

$$F1 = \sum m(2,3,4,6,7) = B+AC'$$

$$F2 = \sum m(0,1,2,6) = A'B'+BC'$$

$$F3 = \sum m(2,3,5,6,7) = AC+B$$

The above set of formulas are implemented using NOR-NOR logic of PLA as shown in Fig. 9 by placing the NMOS switching transistors wherever the connection has to be established. The same set of equations can be implemented using an AND-OR array equivalent as shown in Fig. 10.

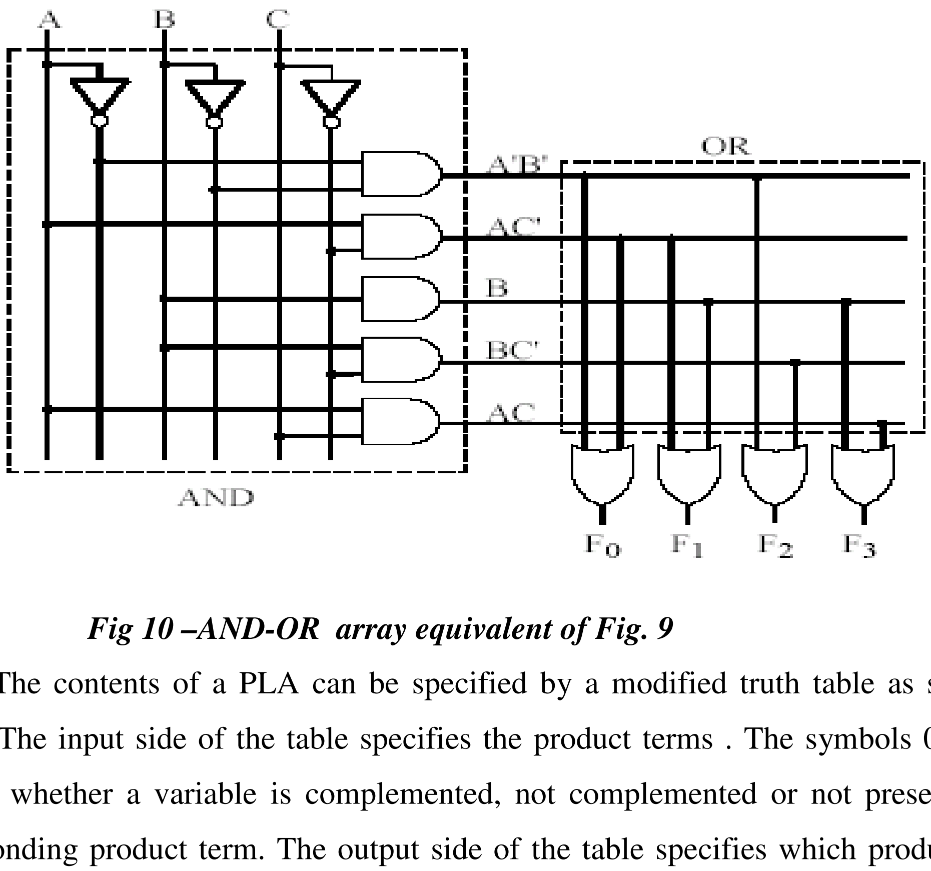


Fig 10 –AND-OR array equivalent of Fig. 9

The contents of a PLA can be specified by a modified truth table as shown in Fig.11. The input side of the table specifies the product terms . The symbols 0,1 and – indicate whether a variable is complemented, not complemented or not present in the corresponding product term. The output side of the table specifies which product terms appear in which output function. A 1 or 0 in the output terms indicate whether a given product term is present or not present in the corresponding output function.

	Inputs			Outputs			
Product	A	B	C	F0	F1	F2	F3
A'B'	0	0	-	1	0	1	0
AC'	1	-	0	1	1	0	0
B	-	1	-	0	1	0	1
BC'	-	1	0	0	0	1	0
AC	1	-	1	0	0	0	1

fig 11 – PLA Table for Fig. 10

The first row of the table indicates that the term A’B’ is present in output functions F0 and F2. The second row indicates that AC’ is present in F0 and F1. This PLA table can be written directly using the given set of equations to be realized using PLA logic.

Realization of a given function using min number of rows in the PLA

$$F1 = \sum m(2,3,5,7,8,9,10,11,13,15) \quad \text{….(1)}$$
$$F2 = \sum m(2,3,5,6,7,10,11,14,15) \quad \text{…. (2)}$$
$$F3 = \sum m(6,7,8,9,13,14,15) \quad \text{…. (3)}$$

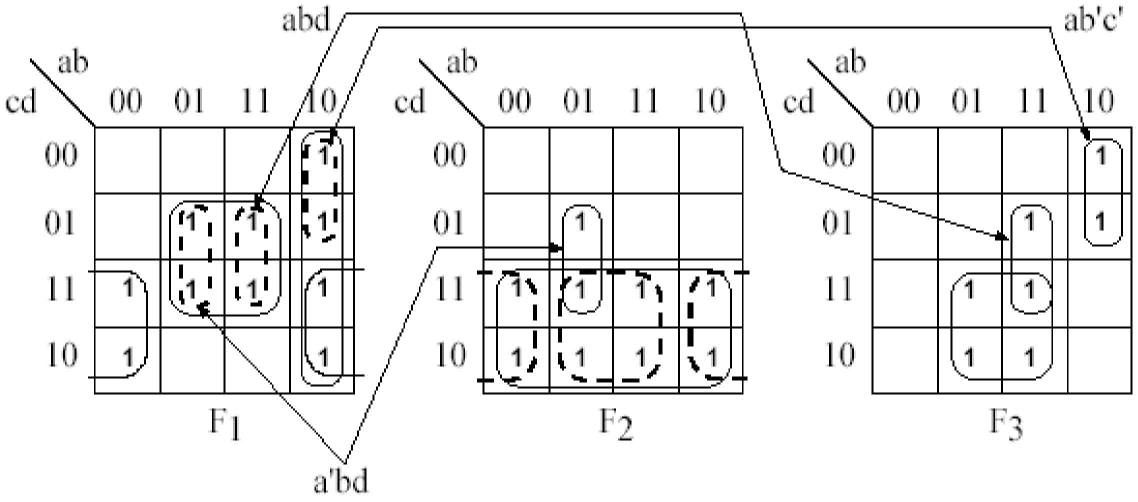


Fig 12 – Multiple Output Karnaugh Map

F1 = BD+B’C+AB’ ….(4)

F2 = C+A’BD ….(5)

F3 = BC+AB’C’+ABD ….(6)

Equations 1,2 and 3 can be reduced to equations 4, 5 and 6 respectively using Karnaugh map as shown in Fig.12.

If we implement these reduced equations 4,5 and 6 in a PLA then a total of 8 different product terms (including C) are required. So instead of minimizing each function separately, we have to minimize the total number of rows in the PLA table. When we are trying to design a logic using PLA, the number of terms in each equation is not important since the size of the PLA does not depend on the number of terms. The term AB’C’ is already present in function F3. So we can use it in F1 instead of AB’ by writing AB’ as AB’(C+C’) . F1 can be now written as

F1 = BD + B’C + AB’(C+C’)

= BD + B’C + AB’C + A’B’C’

= BD + B’C (1+ A) + A’B’C’

= BD + B’C + A’B’C’

This simplification of F1 eliminates the need to use a separate row for the original term AB’ present in F1 of equation (4).

Since the terms A’BD and ABD are needed in F2 and F3 respectively, we can replace the term BD in F1 with A’BD + ABD. This eliminates the need for a row to implement the term BD in PLA. Similarly, since B’C and BC are used in F1 and F3 respectively, w can replace C in F3 with B’C + BC. Now the equations for F1, F2 and F3 with the above said changes can be written as :

F1 = BD(A+A’) + B’C + AB’(C+C’)

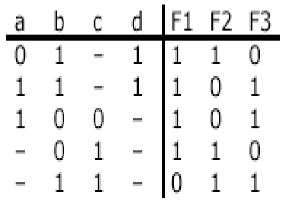
= ABD + A’BD + B’C +AB’C’ ….(7)

F2 = C(B+B’) +A’BD

= BC + B’C + A’BD ….(8)

F3 = BC+AB’C’+ABD ….(9)

The current equations for F1, F2 and F3 as shown in equations 7,8 and 9 respectively have only 5 different product terms. So the PLA table can now be written with only 5 rows. This is a significant improvement over the equations 4, 5 and 6, which resulted in 8 product terms. The reduced PLA table corresponding to equations 7, 8 and 9 is as shown in the figure 13.



a	b	c	d	F1	F2	F3
0	1	-	1	1	1	0
1	1	-	1	1	0	1
1	0	0	-	1	0	1
-	0	1	-	1	1	0
-	1	1	-	0	1	1

Fig 13 – Reduced PLA table

PLA table is significantly different from that of ROM truth table. In a truth table, each row represents a minterm ; therefore, one row will be exactly selected by each combination of input values. The 0s and 1s of the output portion of the selected row determine the corresponding output values. On the other hand, each row in a PLA table represents a general product term. Therefore, 0, 1 or more rows may be selected by each combination of input values. To determine the value of F for a given input combination, the values of F in the selected rows of the PLA table must be ORed together. For example, if **abcd=0001** is given as input , no rows are selected as this combination does not exist in the PLA table; and all the F outputs are 0. If abcd = 1001, only the third row is selected resulting in F1F2F3 = 101. If **abcd = 0111,** the first and the fifth rows are selected. Therefore, to get the values for F1, F2 and F3 is got by ORing the respective values of F1, F2 and F3 in the corresponding rows resulting in F1 = 1 + 0 = 1, F2 = 1+1 = 1 and F3 = 0 + 1 = 1.

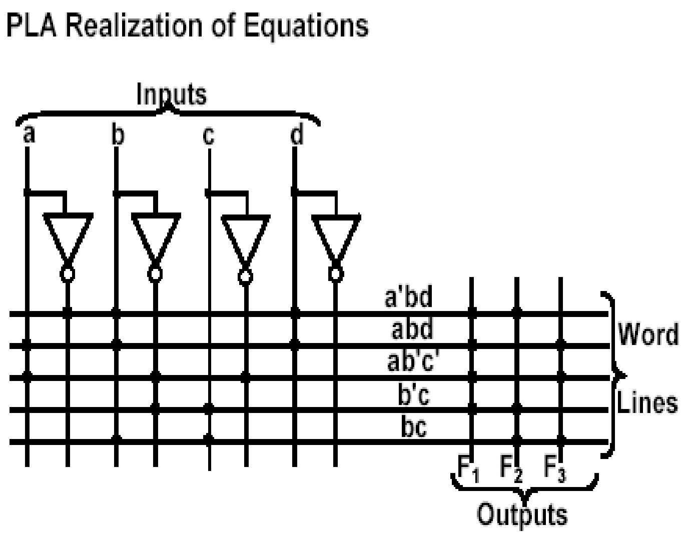


Fig 14–PLA Realization of Equations 7, 8 and 9.

Fig 14 shows the PLA structure, which has four inputs, five product terms and three outputs as shown in equation 7,8 and 9. A dot at the intersection of word line and an input or output line indicates the presence of a switching element in the array.

Implementation of BCD to Excess-3 using PLA

Product Term	Q1	Q2	Q3	X	$Q1^+$	$Q2^+$	$Q3^+$	Z
Q2'	–	0	–	–	1	0	0	0
Q1	1	–	–	–	0	1	0	0
Q1Q2Q3	1	1	1	–	0	0	1	0
Q1Q3'X'	1	–	0	0	0	0	1	0
Q1'Q2'X	0	0	–	1	0	0	1	0
Q3'X'	–	–	0	0	0	0	0	1
Q3X	–	–	1	1	0	0	0	1

Fig 15 –PLA Table for BCD to Excess-3 Converter

$$Q1^{+} = Q2'$$
$$Q2^{+} = Q1$$
$$Q3^{+} = Q1Q2\,Q3 + X'Q1Q3' + XQ1'Q2'$$
$$Z = X'Q3' + XQ3$$

Fig16 –Reduced Equations for BCD to excess-3

We can realize the sequential machine for BCD to excess 3 using a PLA and three D f/f. The network structure is same as that realized with ROM, except that the ROM is replaced by PLA. The required PLA table is as shown in figure 15 is derived from equations shown in figure 16 for BCD to excess-3.

Reading the output of the PLA in VHDL is somewhat more difficult than reading the ROM output. Since the input to the PLA can match several rows, and the output from those rows must be ORed together. The realization of the equations shown in fig 16 is as shown in Fig 17.

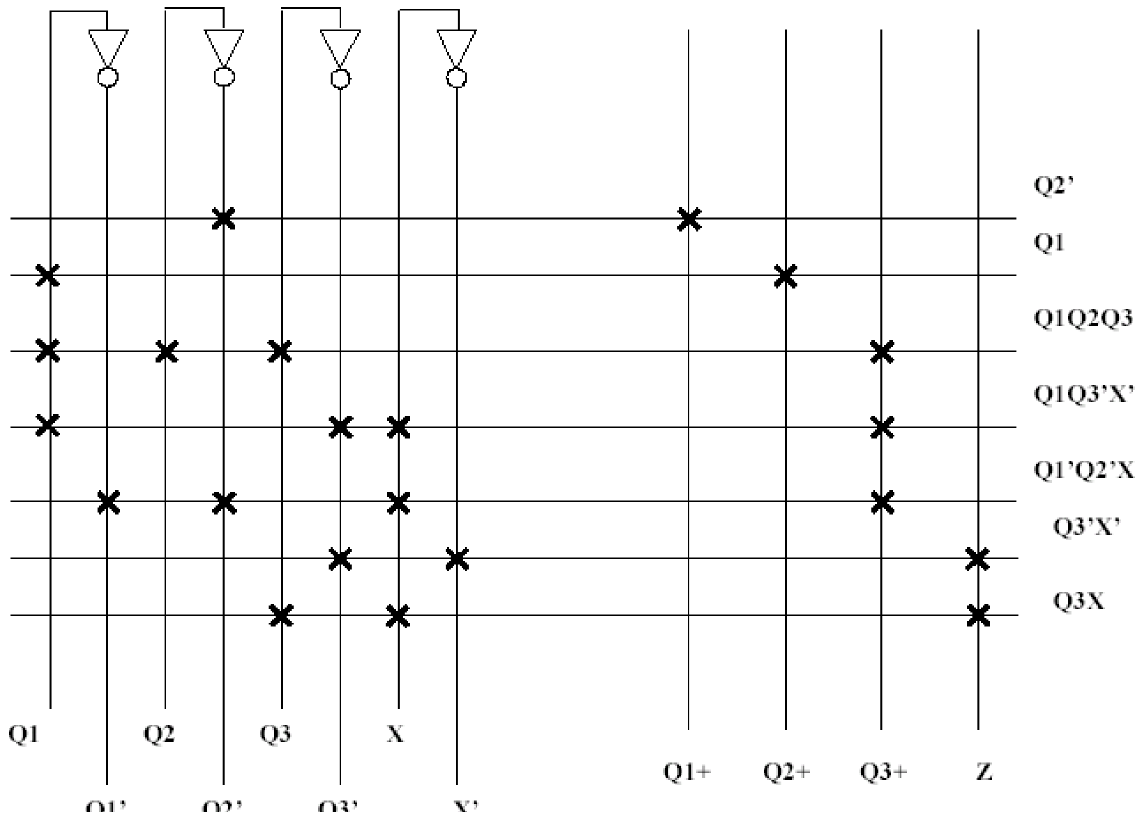


Fig : 17 Realization of BCD to Excess-3 using PLA.