**MEMORY AND PROGRAMMABLE LOGIC**

**Introduction:**

A memory unit is a device to which binary information is transferred for storage and from which information is retrieved when needed for processing. When data processing takes place, information from memory is transferred to selected registers in the processing unit.

Intermediate and final results obtained in the processing unit are transferred back to be stored in memory. Binary information received from an input device is stored in memory, and information transferred to an output device is taken from memory. A memory unit is a collection of cells capable of storing a large quantity of binary information.

There are two types of memories that are used in digital systems: *random access memory* (RAM) and *read only memory* (ROM). RAM stores new information for later use. The process of storing new information into memory is referred to as a memory *write* operation. The process of transferring the stored information out of memory is referred to as a memory *read* operation. RAM can perform both write and read operations.

ROM can perform only the read operation. This means that suitable binary information is already stored inside memory and can be retrieved or read at any time. However, that information cannot be altered by writing.

ROM is a *programmable logic device* (PLD). The binary information that is stored within such a device is specified in some fashion and then embedded within the hardware in a process is referred to as *programming* the device. The word “programming” here refers to a hardware procedure which specifies the bits that are inserted into the hardware configuration of the device. ROM is one example of a PLD. Other such**‐** units are the programmable logic array (PLA), programmable array logic (PAL), and the field programmable gate array (FPGA).

A PLD is an integrated circuit with internal logic gates connected through electronic intact. Programming the device involves blowing those fuses along the paths that must be removed in order to obtain the particular configuration of the desired logic function.

We introduce the configuration of PLDs and indicate procedures for their use in the design of digital systems. We also present CMOS FPGAs, which are configured by downloading a stream of bits into the device to configure transmission gates to establish the internal connectivity required by a specified logic function (combinational or sequential).

A typical PLD may have hundreds to millions of gates interconnected through hundreds to thousands of internal paths. In order to show the internal logic diagram of such a device in a concise form, it is necessary to employ a special gate symbology applicable to array logic. Figure shows the conventional and array logic symbols for a multiple input OR gate. Instead of having multiple input lines into the gate, we draw a single line entering the gate. The input lines are drawn perpendicular to this single line and are connected to the gate through internal fuses. In a similar fashion, we can draw the array logic for an AND gate. This type of graphical representation for the inputs of gates will be used throughout the chapter in array logic diagrams.



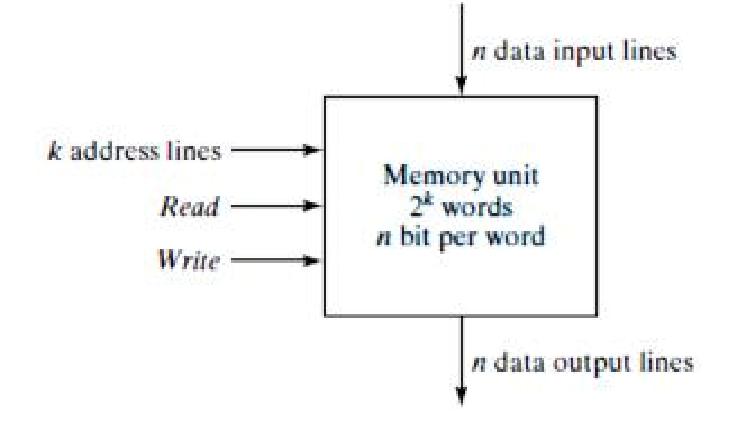
Conventional and array logic diagrams for OR gate

**RANDOM-ACCESS MEMORY:**

A memory unit is a collection of storage cells, together with associated circuits needed to transfer information into and out of a device. The architecture of memory is such that information can be selectively retrieved from any of its internal locations. The time it takes to transfer information to or from any desired random location is always the same—hence the name *random access* *memory,* abbreviated RAM. In contrast, the time required to retrieve information that is stored onmagnetic tape depends on the location of the data.

A memory unit stores binary information in groups of bits called *words*. A word in memory is an entity of bits that move in and out of storage as a unit. A memory word is a group of 1’s and 0’s and may**‐** represent a number, an instruction, one or more alphanumeric characters, or any other binary coded information. A group of 8 bits is called**‐** a *byte*. Most computer memories**‐** use words that are multiples of 8 bits in length. Thus, a 16 bit word contains two bytes, and a 32 bit word is made up of four bytes. The capacity of a memory unit is usually stated as the total number of bytes that the unit can store.

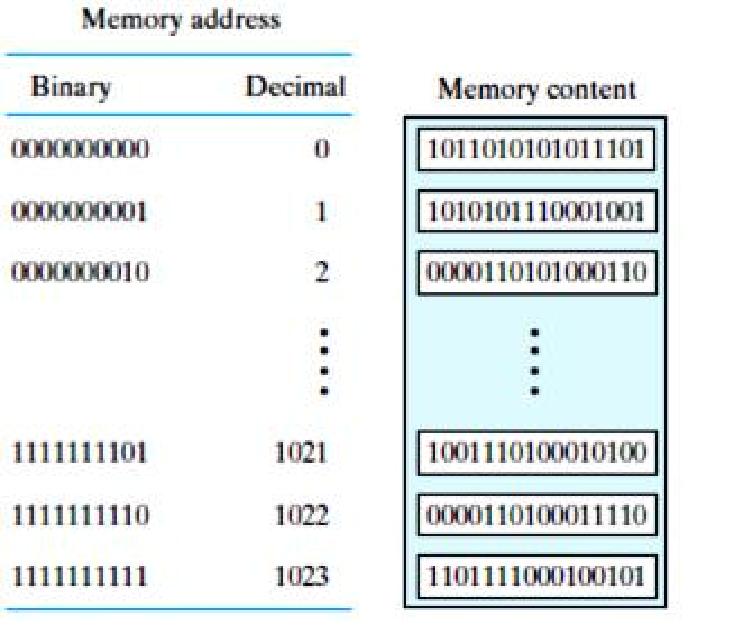
Communication between memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer. A block diagram of a memory unit is shown in Fig. below. The *n* data input lines provide the information to be stored in memory, and the *n* data output lines supply the information coming out of memory. The *k* address lines specify the particular word chosen among the many available. The two control inputs specify the direction of transfer desired: The *Write* input causes binary data to be transferred into the memory, and the *Read* input causes binary data to be transferred out of memory.



**Block Diagram of a Memory Unit**

The memory unit is specified by the number of words it contains and the number of bits in each word. The address lines select one particular word. Each word in memory is assigned an identification number, called an *address,* starting from 0 up to 2 *k* - 1, where *k* is the number**‐** of address lines. The selection of a specific word inside memory is done by applying the *k* bit address to the address lines. An internal decoder accepts this address and opens the paths needed to select the word specified. Memories vary greatly in size and may range from 1,024 words, requiring an address of 10 bits, to 232 words, requiring 32 address bits. It is customary to refer to the number of words (or bytes) in memory with one of the letters K (kilo), M (mega), and G (giga). K is equal to 210, M is equal to 220, and G is equal to 230. Thus, 64K = 216, 2M = 221, and 4G = 232.

Consider, for example, a memory unit with a capacity of 1K words of 16 bits each. Since 1K = 1,024 = 210 and 16 bits constitute two bytes, we can say that the memory can accommodate 2,048 = 2K bytes. Below figure shows possible contents of the first three and the last three words of this memory. Each word contains 16 bits that can be divided into two bytes. The words are recognized by their decimal address from 0 to 1,023. The equivalent binary address consists of 10 bits. The first address is specified with ten 0’s; the last address is specified with ten 1’s, because 1,023 in binary is equal to 1111111111. A word in memory is selected by its binary address. When a word is read or written, the memory operates on all 16 bits as a single unit.



**Contents of a 1024 \* 16 Memory**

**Write and Read Operations**

The two operations that RAM can perform are**‐** the write and read operations. As alluded to earlier, **‐**the write signal specifies a transfer in operation and the read signal specifies a transfer out operation. On accepting one of these control signals, the internal circuits inside the memory provide the desired operation.

The steps that must be taken for the purpose of transferring a new word to be stored into memory are as follows:

1. Apply the binary address of the desired word to the address lines.
2. Apply the data bits that must be stored in memory to the data input lines.
3. Activate the *write* input.

The memory unit will then take the bits from the input data lines and store them in the word specified by the address lines.

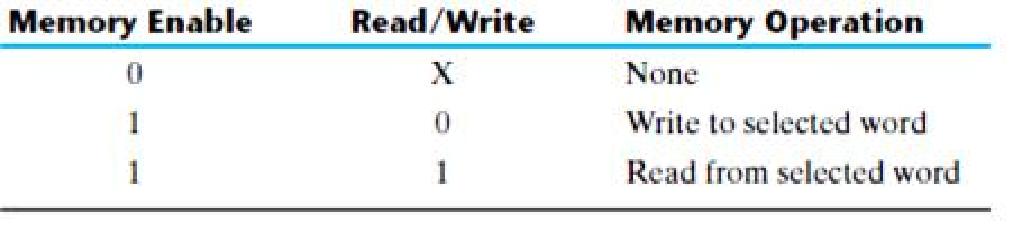
The steps that must be taken for the purpose of transferring a stored word out of memory are as follows:

1. Apply the binary address of the desired word to the address lines.
2. Activate the *read* input

The memory unit will then take the bits from the word that has been selected by the address and apply them to the output data lines. The contents of the selected word do not change after the read operation, i.e., the word operation is nondestructive.

Commercial memory components available in integrated**‐**circuit chips sometimes provide the two control inputs for reading and writing in a somewhat different configuration. Instead of having separate read and write inputs to control the two operations, most integrated circuits provide two other control inputs: One input selects the unit and the other determines the operation. The memory operations that result from these control inputs are specified in Table below.

**Control Inputs to Memory Chip**



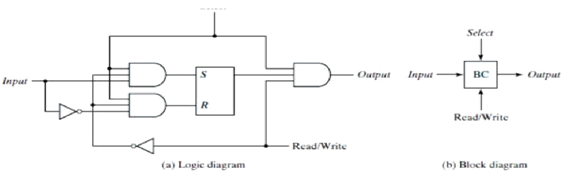
The memory enable (sometimes called the chip select) is used to enable the particular memory chip in a multichip implementation of a large memory. When the memory enable is inactive, the memory chip is not selected and no operation is performed. When the memory enable input is active, the read/write input determines the operation to be performed.

**Memory Decoding**

In addition to requiring storage components in a memory unit, there is a need for decoding circuits to select the memory word specified by the input address. In this section, we present the internal construction of a RAM and demonstrate the operation of the decoder. To be able to include the entire memory in one diagram, the memory unit presented here**‐** has a small capacity of 16 bits, arranged in four words of 4 bits each. An example of a two dimensional coincident decoding arrangement is presented to show a more efficient decoding scheme that is used in large memories. We then give an example of address multiplexing commonly used in DRAM integrated circuits.

**Internal Construction**

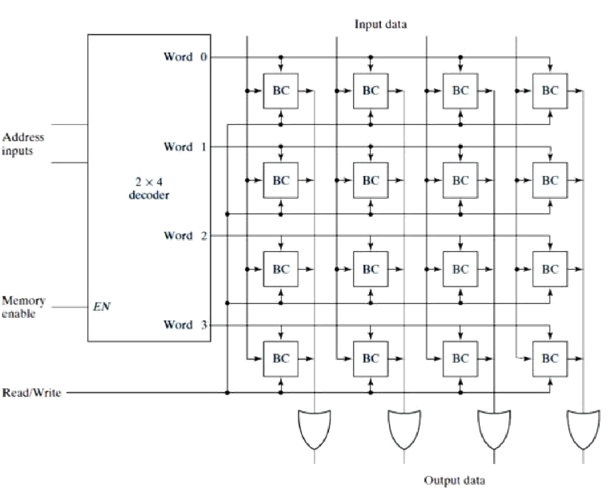
The internal construction of a RAM of *m* words and *n* bits per word consists of *m* \* *n* binary storage cells and associated decoding circuits for selecting individual words. The binary storage cell is the basic building block of a memory unit. The equivalent logic of a binary cell that stores one bit of information is shown in Fig. below. The storage part of the cell is modeled by an *SR* latch with associated gates to form a *D* latch. Actually, the convenient to model it in terms of logic symbols. A binary storage cell must be very small in order to be able to pack as many cells as possible in the small area available in the integrated circuit chip. The binary cell stores one bit in its internal latch. The select input enables the cell for reading or writing, and the read/write input determines the operation of the cell when it is selected. A 1 in the read/write input provides the read operation by forming a path from the latch to the output terminal. A 0 in the read/write input provides the write operation by forming a path from the input terminal to the latch.



**Memory Cell**

The logical construction of a small RAM is shown in Fig. below. This RAM consists of four words of four bits each and has a total of 16 binary cells. The small blocks labeled BC represent the binary cell with its three inputs and one output, as specified in Fig. above. A memory with four words needs two address lines. The two address inputs go through**‐** a 2 \* 4 decoder to select one of the four words. The decoder is enabled with the memory enable input. When the memory enable is 0, all outputs of the decoder are 0 and none of the memory words are selected. With the memory select at 1, one of the four words is selected, dictated by the value in the two address lines. Once a word has been selected, the read/write input determines the operation. During the read operation, the four bits of the selected word go through OR gates to the output terminals. During the write operation, the data available in the input lines are transferred into the four binary cells of the selected word. The binary cells that are not selected are disabled, and their previous binary values remain unchanged. When the memory select input that goes into the decoder is equal to 0, none of the words are selected and the contents of all cells remain unchanged regardless of the value of the read/write input.

Commercial RAMs may have a capacity of thousands**‐** of words, and each word may range from 1 to 64 bits. The logical construction of a large capacity memory would be a direct extension of the configuration shown here. A memory with 2*k* words of *n* bits per word requires *k* address lines that go into a *k* \* 2*k* decoder. Each one of the decoder outputs selects one word of *n* bits for reading or writing.

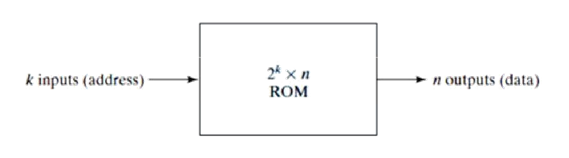


**Diagram of a 4 \* 4 RAM**

**READ ONLY MEMORY:**

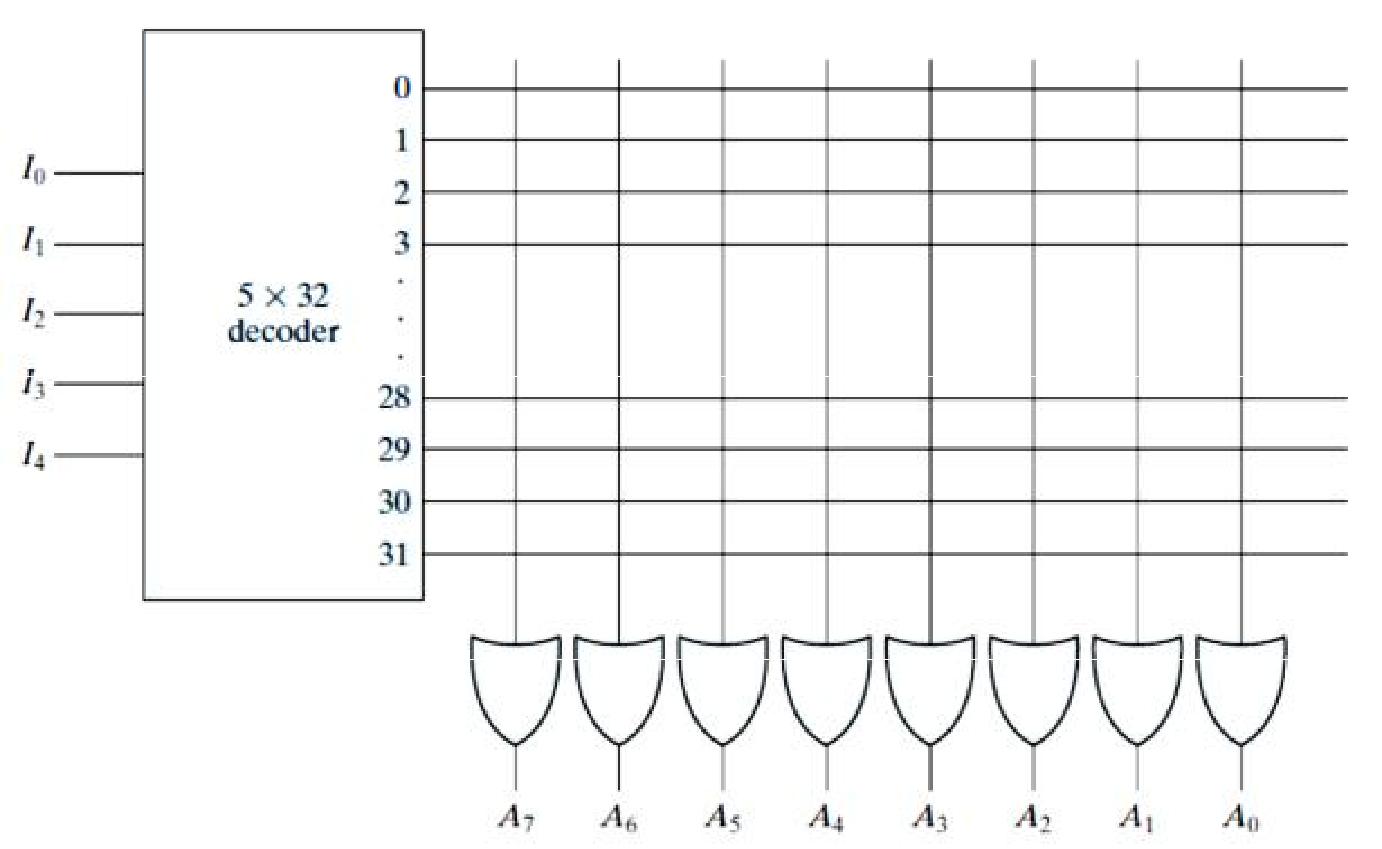
A read**‐**only memory (ROM) is essentially a memory device in which permanent binary information is stored. The binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern. Once the pattern is established, it stays within the unit even when power is turned off and on again.

A block diagram of a ROM consisting of *k* inputs and *n* outputs is shown in Fig. below. The inputs provide the address for memory, and the outputs give the data bits of the stored word that 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. Note that ROM does not have data inputs, because it does not have a write operation. **‐**Integrated circuit ROM chips have one or more enable inputs and sometimes come with three state outputs to facilitate the construction of large arrays of ROM.



**Block Diagram of ROM**

Consider, for example, 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. Below figure shows the internal logic construction of this ROM. 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 eight OR gates. The diagram shows the array logic convention used in complex circuits. 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* \* *n* ROM will have an internal *k* \* 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.



**Internal logic of a 32: 8 ROM**

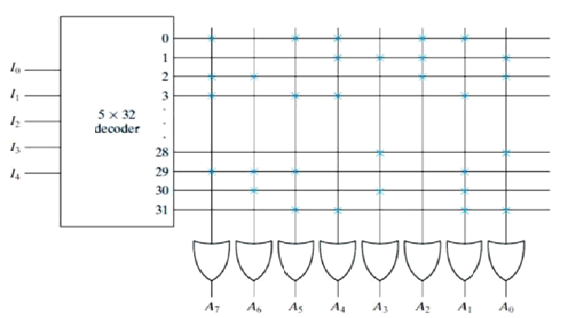
**Combinational Circuit Implementation**

It was shown that a decoder generates the 2*k* minterms of the *k* input variables. By inserting OR gates to sum the minterms of Boolean functions, we were able to generate any desired combinational circuit. The ROM is essentially a device that includes both the decoder and the OR gates within a single device to form a minterm generator. By choosing connections for those minterms which are included in the function, the ROM outputs can be programmed to represent the Boolean functions of the output variables in a combinational circuit.

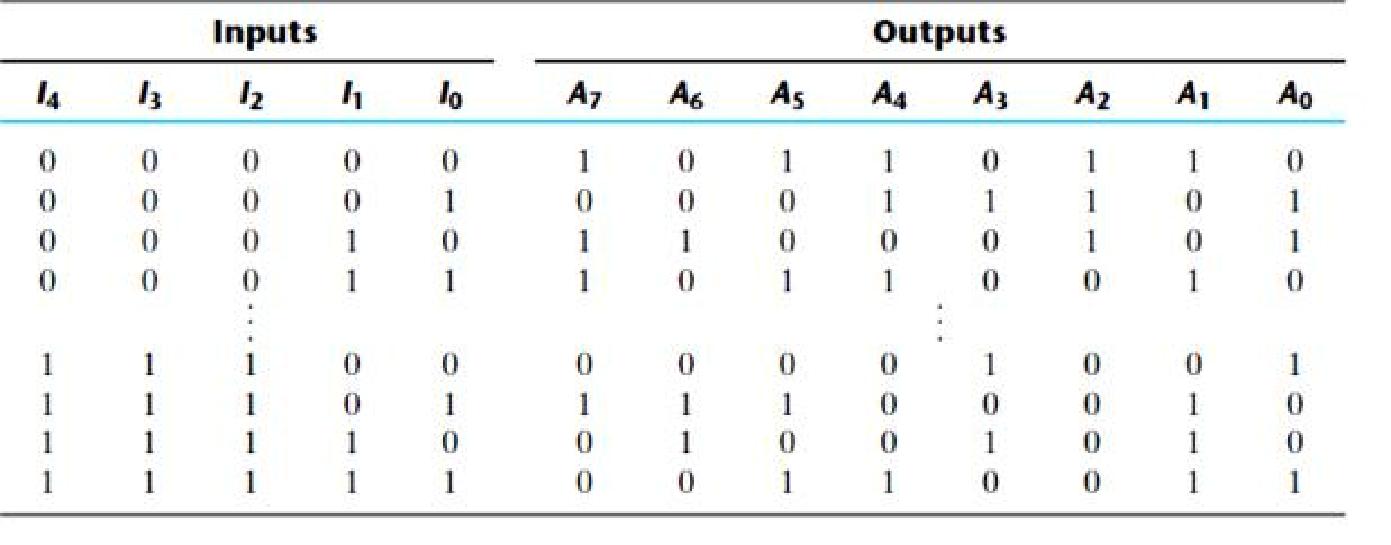
The internal operation of a ROM can be interpreted in two ways. The first interpretation is that of a memory unit that contains a fixed pattern of stored words. The second interpretation is that of a unit which implements a combinational circuit. From this point of view, each output terminal is considered separately as the output of a Boolean function expressed as a sum of minterms. For example, the ROM may be considered to be a combinational circuit with eight outputs, each a function of the five input variables. Output *A*7 can be expressed in sum of minterms as

*A*7(*I*4, *I*3, *I*2, *I*1, *I*0) =**∑**m(0, 2, 3, …., 29)

(The three dots represent minterms 4 through 27, which are not specified in the figure.) A connection marked with \* in the figure produces a minterm for the sum. All other crosspoints are not connected and are not included in the sum. In practice, when a combinational circuit is designed by means of a ROM, it is not necessary to design the logic or to show the internal gate connections inside the unit. All that the designer has to do is specify the particular ROM by its IC number and provide the applicable truth table. The truth table gives all the information for programming the ROM. No internal logic diagram is needed to accompany the truth table.



**Programming the ROM according to Table given above**



**Types of ROMs**

The required paths in a ROM may be programmed in four different ways. The first is called *mask* *programming* and is done by the semiconductor company during the last fabrication process ofthe unit. The procedure for fabricating a ROM requires that the customer fill out the truth table he or she wishes the ROM to satisfy. The truth table may be submitted in a special form provided by the manufacturer or in a specified format on a computer output medium. The manufacturer makes the corresponding mask for the paths to produce the 1’s and 0’s according to the customer’s truth table. This procedure is costly because the vendor charges the customer a special fee for custom masking the particular ROM. For this reason, mask programming is economical only if a large quantity of the same ROM configuration is to be ordered.

For small quantities, it is more economical to use a second type of ROM called *programmable* *read only memory,* or PROM. When ordered, PROM units contain all the fuses intact, giving all1’s in**‐** the bits of the stored words. The fuses in the PROM are blown by the application of a high voltage pulse to the device through a special pin. A blown fuse defines a binary 0 state and an intact fuse gives a binary 1 state. This procedure allows the user to program the PROM in the laboratory to achieve the desired relationship between input addresses and stored words. Special instruments called PROM programmers are available commercially to facilitate the procedure. In any case, all procedures for programming ROMs are hardware procedures, even though the word *programming* is used.

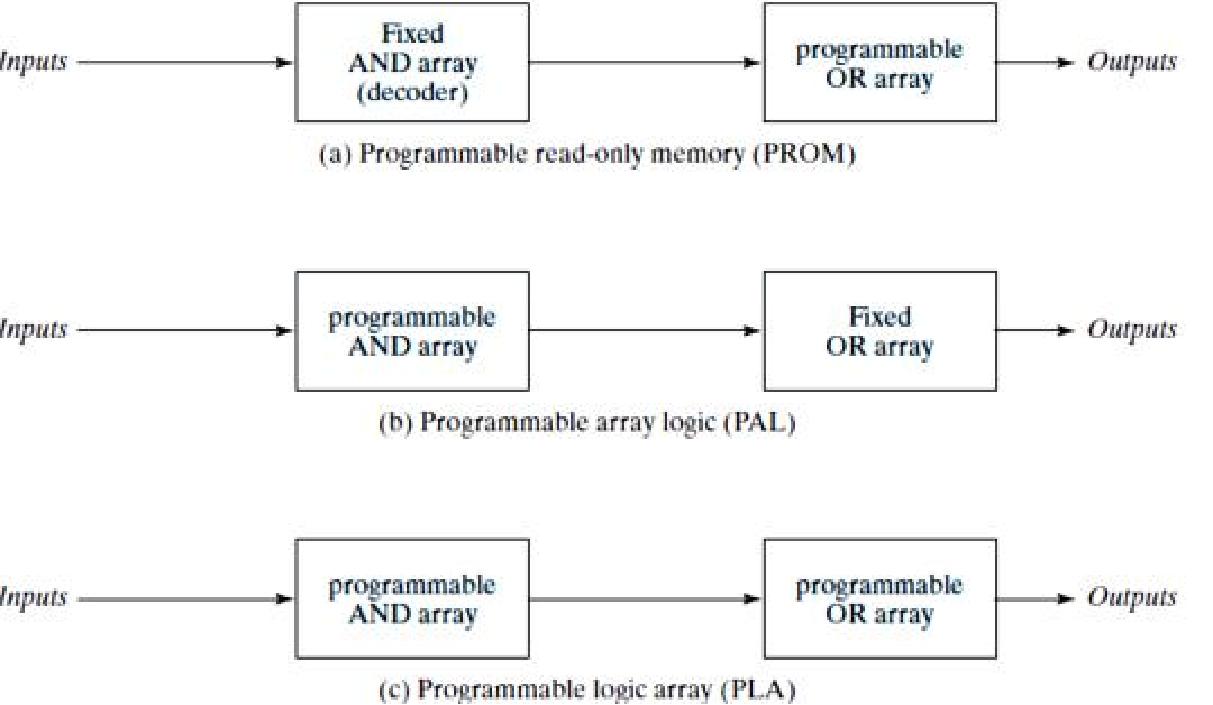
The hardware procedure for programming ROMs or PROMs is irreversible, and once programmed, the fixed pattern is permanent and cannot be altered. Once a bit pattern has been established, the unit must be discarded if the bit pattern is to be changed. A third type of ROM is the *erasable PROM,* or EPROM, which can be

restructured to the initial state even though it has been programmed previously. When the EPROM is placed under a special ultraviolet light for a given length of time, the shortwave radiation discharges the internal floating gates that serve as the programmed connections. After erasure, the EPROM returns to its initial state and can be reprogrammed to a new set of values.

The fourth type of ROM is the electrically erasable PROM (EEPROM or E2PROM). This device is like the EPROM, except that the previously programmed connections can be erased with an electrical signal instead of ultraviolet light. The advantage is that the device can be erased without removing it from its socket.

**Combinational PLDs**

The PROM is a combinational programmable logic device (PLD)—an integrated circuit with programmable**‐‐** gates divided into an AND array and an OR array to provide an AND–OR sum of product implementation. There are three major types of combinational PLDs, differing in the placement of the programmable connections in the AND– OR array. Below figure shows the configuration of the three PLDs. The PROM has a fixed AND array constructed as a decoder and a programmable**‐‐** OR array. The programmable OR gates implement the Boolean functions in sum of minterms form. The PAL has a programmable AND array and a fixed 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 most flexible PLD is the PLA, in which 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. The names PAL and PLA emerged from different vendors during the development of PLDs. The implementation of combinational circuits with PROM was demonstrated in this section. The design of combinational circuits with PLA and PAL is presented in the next two sections.



**Basic configuration of three PLDs**

**PROGRAMMABLE LOGIC ARRAY :**

The PLA is similar in concept to the PROM, except that the PLA does not provide full decoding of the variables and does not generate all the minterms. The decoder is replaced by an array of AND gates that can be programmed to generate any product term of the input variables. The product terms are then connected to OR gates to provide the sum of products for the required Boolean functions.

The internal logic of a PLA with three inputs and two outputs is shown in Fig. below. Such a circuit is too small to be useful commercially, but is presented here to demonstrate the typical logic configuration of a PLA. The diagram uses the array logic graphic symbols for complex circuits. Each input goes through a buffer–inverter combination, shown in the diagram with a composite graphic symbol, that has both the true and complement outputs. Each input and its complement is connected to the inputs of each AND gate, as indicated by the intersections between the vertical and horizontal lines. The outputs of the AND gates are connected to the inputs of each OR gate. The output of the OR gate goes to an XOR gate, where the other input can be programmed to receive a signal equal to either logic 1 or logic 0. The output is inverted when the XOR input is connected to 1 (since *x* XOR 1 = *x*’). The output does not change when the XOR input is connected to 0 (since *x XOR* 0 = *x)*.

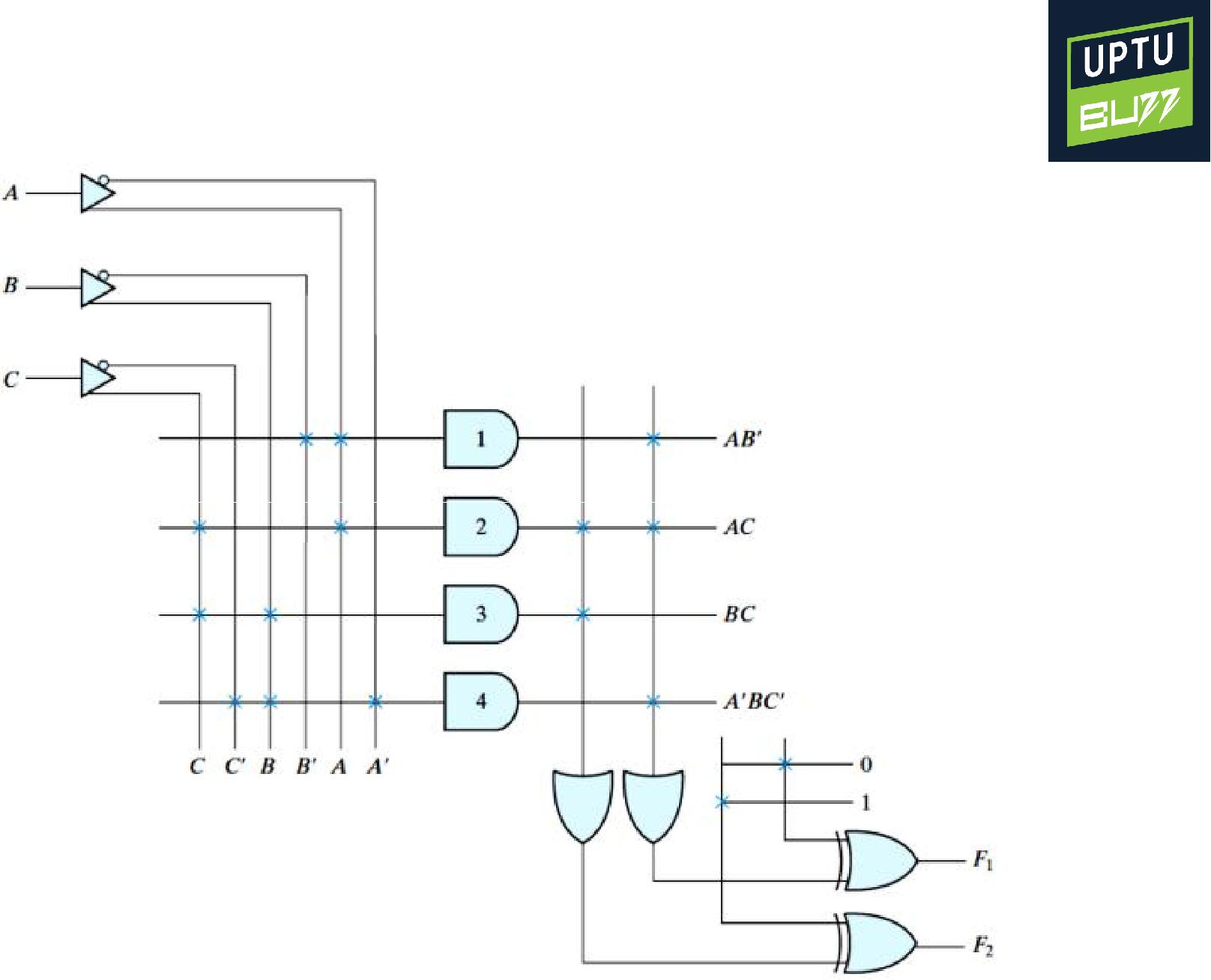
The particular Boolean functions implemented in the PLA of below Fig. are

*F*1 = *AB*’+ *AC* + *A*’*BC*’

*F*2 = (*AC* + *BC*)’

The product terms generated in each AND gate are listed along the output of the gate in the diagram. The product term is determined from the inputs whose crosspoints are connected and marked with a \*. The output of an OR gate gives the logical sum of the selected product terms. The output may be complemented or left in its true form, depending on the logic being realized

The fuse map of a PLA can be specified in a tabular form. For example, the programming table that specifies the PLA of above Fig. is listed in above Table. The PLA programming table consists of three sections. The first section lists the product terms numerically. The second section specifies the required paths between inputs and AND gates. The third section 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 XOR 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 the form in which it is true, 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 from the product term, it is marked with a dash.



**PLA with Three Inputs, Four Product Terms, and Two Outputs**

The paths between the inputs and the AND gates are specified under the column head “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 head “Outputs.” The output variables are marked with 1’s for those product terms which 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 XOR 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. A typical integrated circuit PLA may have 16 inputs, 48 product terms, and eight outputs. 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* XOR gates. There are 2*n* \* *k* connections between the inputs and the AND array, *k* \* *m* connections between the AND and OR arrays, and *m* connections associated with the XOR gates.

In designing a digital system with a PLA, there is no need to show the internal connections of the unit as was done in Fig. above. All that is needed is a PLA programming table from which the PLA can be programmed to supply the required logic. As with a ROM, the PLA may be mask programmable or field programmable. With mask programming, the customer submits a**‐** PLA program table to the manufacturer. This table is used by the vendor to produce a custom made PLA that has the required internal logic specified by the customer. A second type of PLA that is available is the field programmable logic array, or FPLA, which can be programmed by the user by means of a commercial hardware programmer unit.

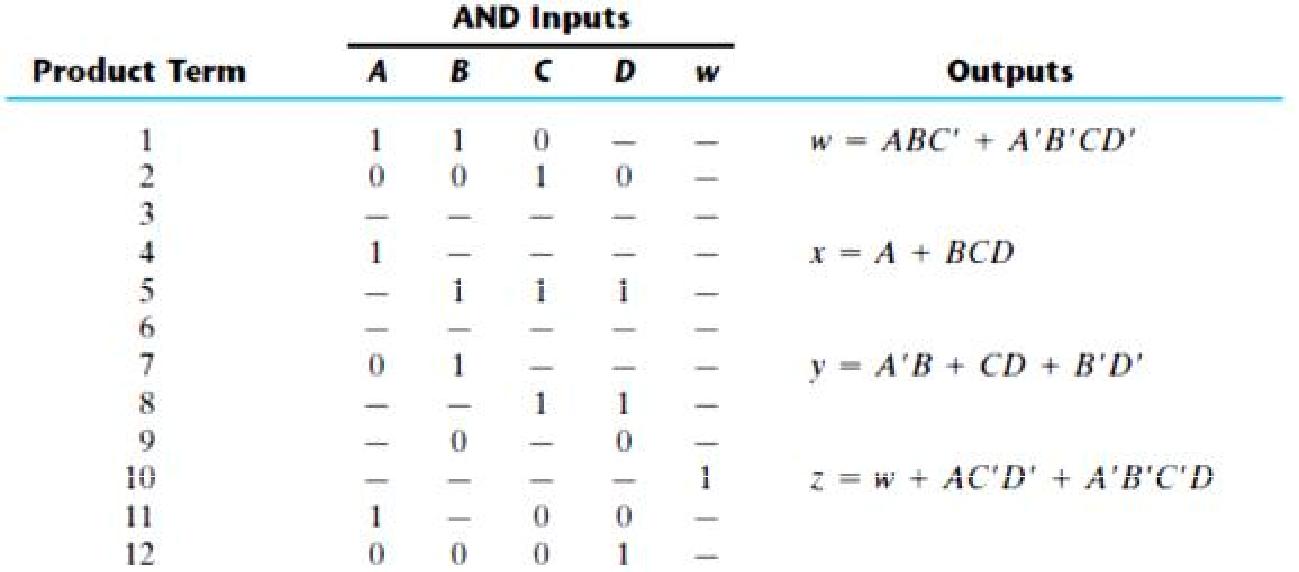
In implementing a combinational circuit with a PLA, careful investigation must be undertaken in order to reduce the number of distinct product terms, since a PLA has a finite number of AND gates. This can be done by simplifying each Boolean function to a minimum number of terms. The number of literals in a term is not important, since all the input variables are available anyway. Both the true value and the complement of each function should be simplified to see which one can be expressed with fewer product terms and which one provides product terms that are common to other functions.

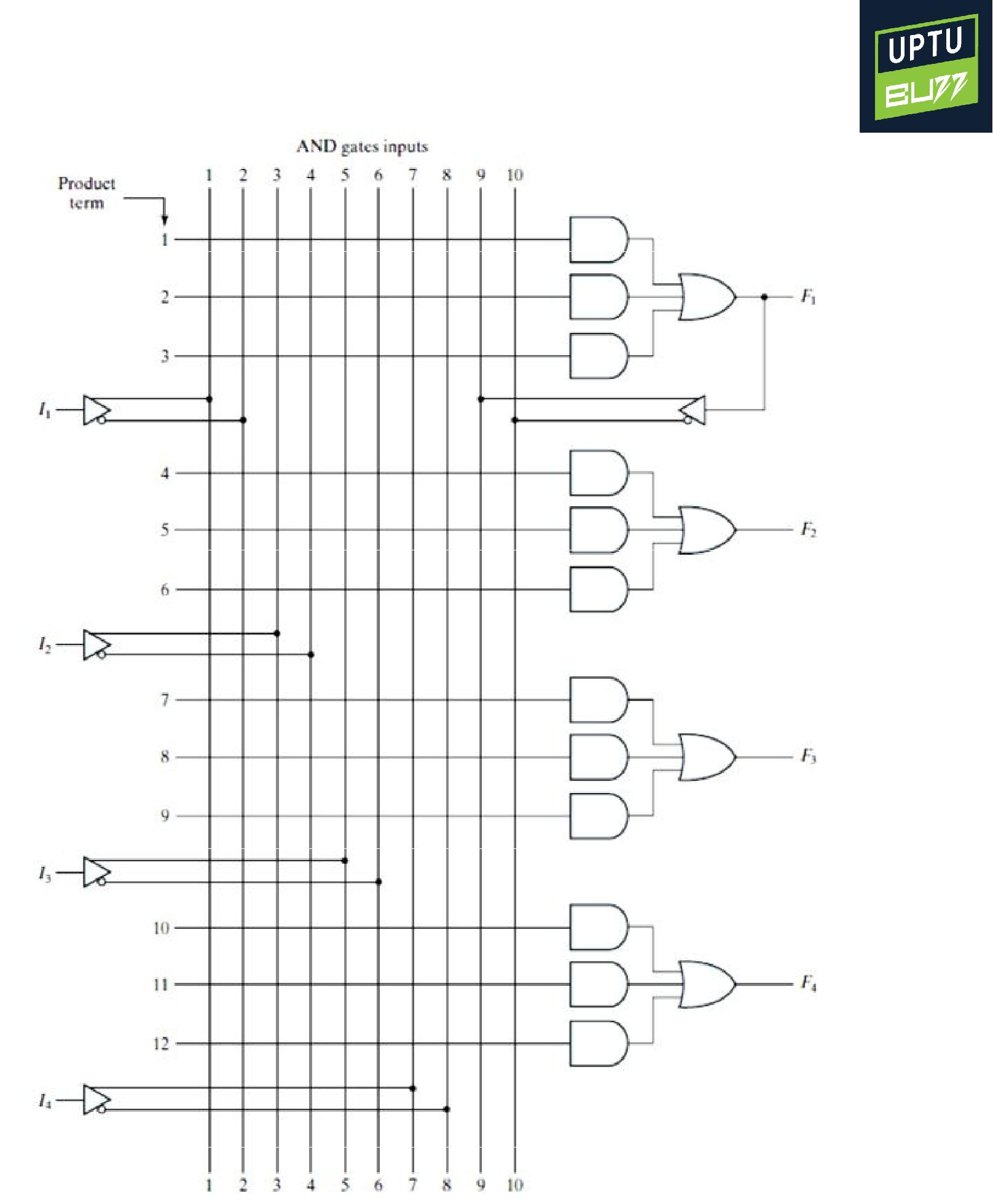
**PAL :**

The PAL is a programmable logic device with a fixed OR array and a programmable AND array. Because only the AND gates are programmable, the PAL is easier to program than, but is not as flexible as, the PLA. Figure 7.16 shows the logic configuration of a typical PAL with four inputs and four outputs. Each input has a buffer–inverter gate, and each output is generated by a fixed OR gate. There are four sections in the unit, each composed of an AND–OR array that is *three* *wide*, the term used to indicate that there are three programmable AND gates in each section andone fixed OR gate. Each AND gate has 10 programmable input connections, shown in the diagram **‐**by 10 vertical lines intersecting each horizontal line. The horizontal line symbolizes the multiple input configuration of the AND gate. One of the outputs is connected to a buffer– inverter gate and then fed back into two inputs of the AND gates.

In designing with a PAL, the Boolean functions must be simplified to fit into each section. Unlike the situation with a PLA, a product term cannot be shared among two or more OR gates. Therefore, each function can be simplified by itself, without regard to common product terms.

**PAL Programming Table**





**PAL with Four Inputs, Four Outputs, and a Three-wide AND–OR structure**