Digital System Design
Fourth Class
EE 404

Diyala University College
of Engineering Department
of Electronics

Theoretical:2 Hrs/Wk
Tutorial: 1 Hrs/Wk
Practical: Hrs/Wk

Topic	Hours
Simplification of Boolean Function using K-map and Tabulation	6 Hrs
Digital Circuit Design using Logic Circuits (LSI, SSI, MSI)	6 Hrs
Design using Programmable Logic Circuits (ROM, PLA, PAL)	6 Hrs
Synchronized Sequential Circuits (Analysis and Design)	6 Hrs
ASM Diagrams.	6 Hrs
Analysis and Design of Sequential Circuits using ASM Diagrams	6 Hrs
Asynchronous Circuits (Analysis)	6 Hrs
Luminescent pulse phenomenon in logic circuits (Static and Dynamic)	6 Hrs
Microprocessors-Component and Architecture	6 Hrs
Microprocessors Hardware	~ 10 Hrs

4-, 8-, 16- and 32-bit Microprocessors, Single Chip Microcomputer 8085, 8088, MPU details.

Rom and Programmable Logic Devices (PLDs)

Digital ICs are often *categorized* according to the complexity of their circuits, as measured by the number of logic gates in a single package. The differentiation between those chips which have a few internal gates and those having hundreds of thousand of gates is made by customary reference to package as:

Small-scale integration (SSI) devices contain several independent gates in a single package. The inputs and outputs of the gates are connected directly to the pins in the package. The number of gates is usually fewer than 10 and is limited by the number of pins available in the IC.

Medium-scale integration (MSI) devices have a complexity of approximately 10 to 1000 gates in a single package. They usually perform specific elementary digital operations.

Large-scale integration (LSI) devices contain thousands of gates in a single package. They include digital systems such as processor, memory chips, and programmable logic devices.

Very Large-scale integration (VLSI) device contained hundred of thousand of gates within a single package. Examples are large memory array and complex microcomputer chips.

Read Only Memory (ROM):

A 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 Figure (1).

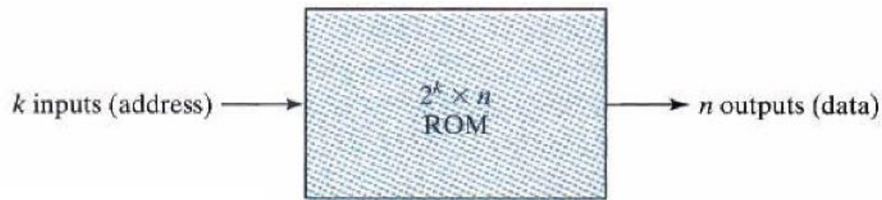


Fig. ( )

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. Consider, for example, a 32 X 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. Figure (2) shows the internal logic construction of this ROM. The five inputs are decoded into 32 distinct outputs by means of a 5 X 32 decoder. Each output of the decoder represents a memory address.

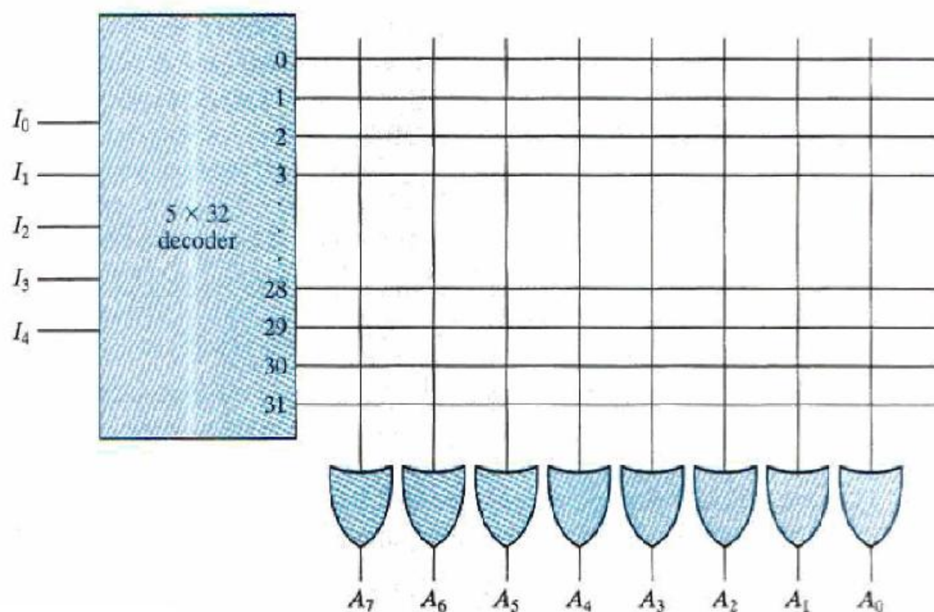
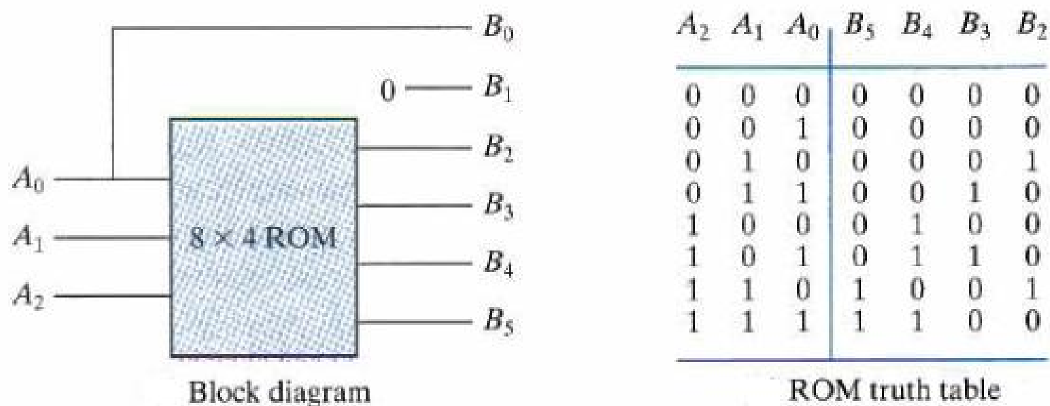


Fig. ( ) Internal logic of a 32 × 8 ROM

EX: Design a combinational circuit using a ROM. The circuit accepts a three-bit number and outputs a binary number equal to the square of the input number?

Sol: Three inputs specify eight words, so the ROM must be of size 8 X 4. The ROM implementation is shown as:

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



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

Block diagram ROM truth table

Types of ROMs

The type of ROM is determined by the way the switches are set or reset (i.e., programmed).

The Mask ROM: The mask ROM is usually referred to simply as a ROM. It is permanently programmed during the manufacturing process to provide widely used standard functions, such as popular conversions, or to provide user-specified functions, once the memory is programmed. It cannot be changed. Most IC ROMs utilize the presence or absence of a transistor connection at a row /column junction to represent a 1 or a 0.

Figure (3) shows MOS ROM cells. The presence of a connection from a row line to the gate of a transistor represents a 1 at that location because when the row line is taken HIGH; all transistors with a gate connection to that row line turn on and connect the HIGH (1) to the associated column lines. At row/column junctions

where there are no gate connections, the column lines remain LOW (0) when the row is addressed.

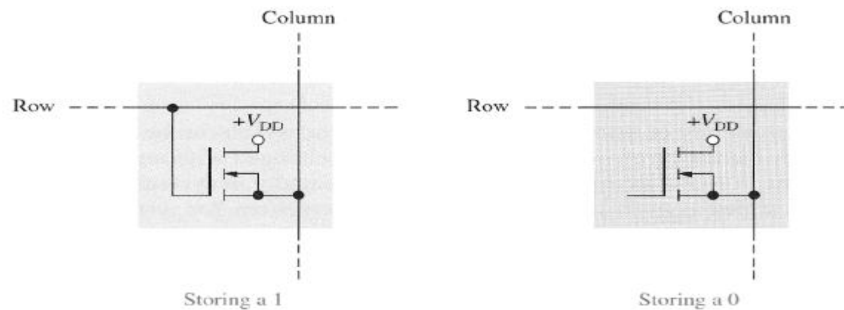


Figure (3)

PROMs: A PROM uses some type of fusing process to store bits, in which a memory *link* is burned open or left intact to represent a 0 or a 1. The fusing process is irreversible: once a PROM is programmed, it cannot be changed.
Figure (4) illustrates a MOS PROM array with fusible links. The fusible links are manufactured into the PROM between the source of each cell's transistor and its column line. In the programming process, a sufficient current is injected through the fusible link to burn it open to create a stored 0. The link is left intact for a stored 1.

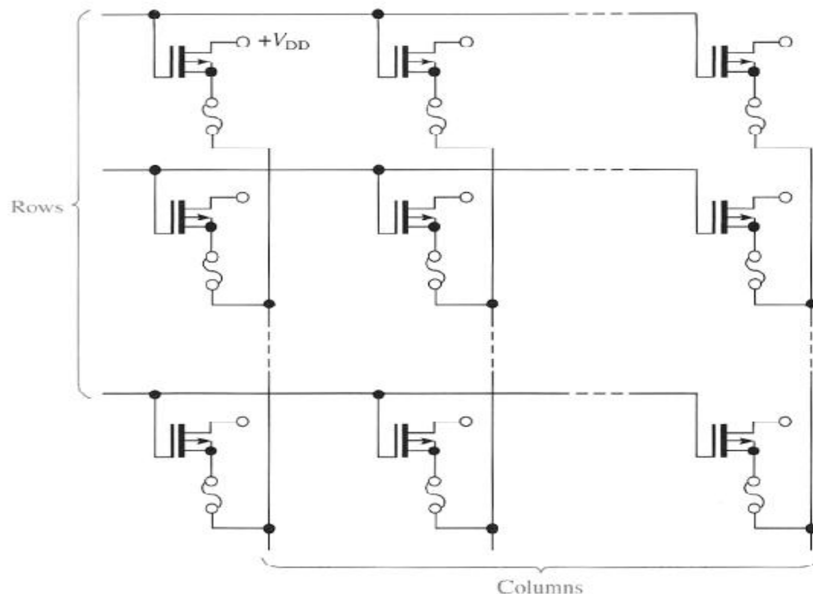


Figure (4)

EPROMs: An EPROM is an erasable PROM. Unlike an ordinary PROM, an EPROM can be reprogrammed if an existing program in the memory array is erased first.

An EPROM uses an NMOSFET array with an isolated-gate structure. The isolated transistor gate has no electrical connections and can store an electrical charge for indefinite periods of time. The data bits in this type of array are represented by the presence or absence of a stored gate charge. Erasure of a data bit is a process that removes the gate charge. Two basic types of erasable PROMs are the ultraviolet erasable PROM (UV EPROM) and the electrically erasable PROM (EEPROM).

UV EPROMs You can recognize the UV EPROM device by the transparent quartz lid on the package, as shown in Figure (5). The isolated gate in the FET of an ultraviolet EPROM is "floating" within an oxide insulating material. The programming process causes electrons to be removed from the floating gate. Erasure is done by exposure of the memory array chip to high-intensity ultraviolet radiation through the quartz window on top of the package. The positive charge stored on the gate is neutralized after several minutes to an hour of exposure time.

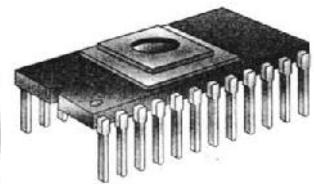


Figure (5) Ultraviolet erasable PROM package

EEPROMs An electrically erasable PROM can be both erased and programmed with electrical pulses. Since it can be both electrically written into and electrically erased, the EEPROM can be rapidly programmed and erased in-circuit for reprogramming.

Word-Length Expansion

To increase the word length of a memory, the number of bits in the data bus must be increased. For example, an 8-bit word length can be achieved by using two memories, each with 4-bit words as illustrated in Figure (6- a). As you can see in part (b), the 16-bit address bus is commonly connected to both memories so that the combination memory still has the same number of addresses (2^{16} = 65,536) as each individual memory. The 4-bit data buses from the two memories are combined to form an 8-bit data bus. Now when an address is selected, eight bits arc produced on the data bus—four from each memory. Example 12-2 shows the details of 65,536 x 4 to 65,536 x 8 expansions.

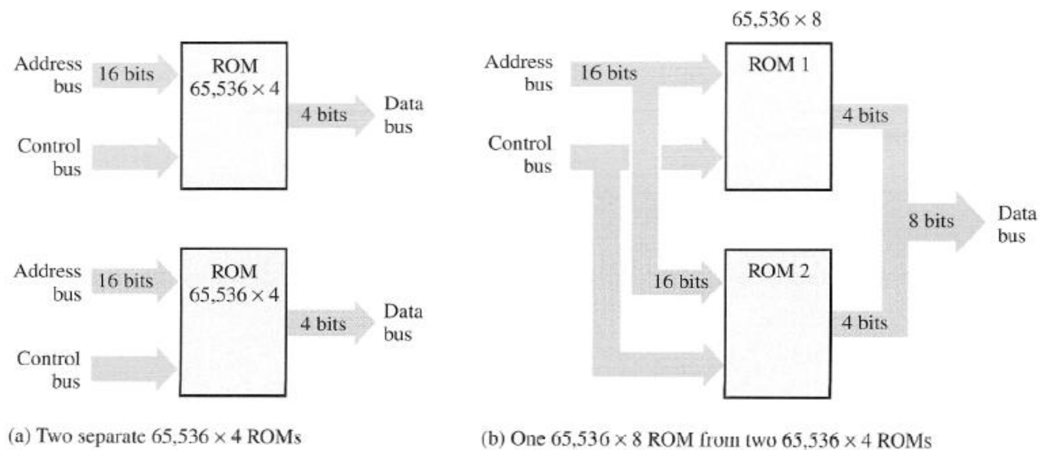


Figure (6)

EXAMPLE: Expand the 65,536 x 4 ROM (64k x 4), to form a 64k x 8 ROM?

Solution Two 64k x 4 ROMs are connected as shown in Figure (7). Notice that a spa-ilk address is accessed in ROM 1 and ROM 2 at the same time. The four bits from a selected address in ROM 1 and the four bits from the corresponding address in ROM 2 go out in parallel to form an 8-bit word on the data bus. Also notice that a LOW on the chip enable line, E. which farms a simple control bus, enables both memories.

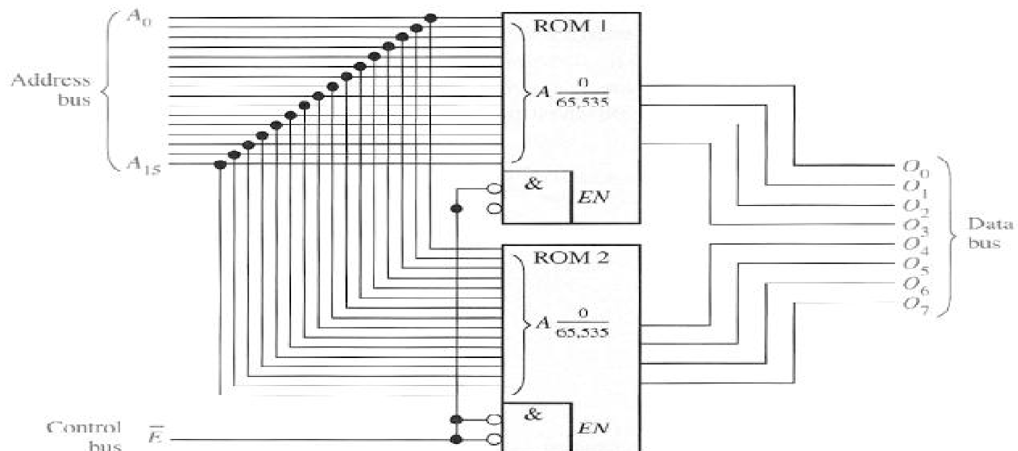


Figure (7)

Programmable Logic Devices (PLDs)

The three major types of programmable logic are SPLD, CPLD, and FPGA. Each major type generally has several manufacturer-specific subcategories.

SPLDs (simple programmable logic devices) are the least complex form of PLDs. An SPLD can typically replace several fixed-function SSI or MSI devices and their interconnections. The SPLD was the first type of programmable logic available A few categories of SPLD are listed below, some of which are unique to a specific manufacturer. A typical package has 24 to 28 pins.

PAL (programmable array logic)

GAL (generic array logic)

PLA (programmable logic array)

PROM (programmable read-only memory)

CPLDs (complex programmable logic devices) have a much higher capacity than SPLDs, permitting more complex logic circuits to be programmed into them. A typical CPLD is the equivalent of from two to sixty-four SPLDs. The development of these devices followed the SPLD as advances in technology permitted higher-density chips to be implemented. There are several forms of CPLD, which vary in complexity and programming capability. CPLDs typically come in 44-pin to 160-pin packages depending on the complexity.

FPGAs (field-programmable gate arrays) are different from SPLDs and CPLDs in their internal organization and have the greatest logic capacity. FPGAs consist of an array of anywhere from sixty-four to thousands of logic-gate groups that are sometimes called logic blocks. Two basic classes of FPGA are course-grained and fine-grained. The course-grained FPGA has large logic blocks, and the fine-grained FPGA has much smaller logic blocks. FPGAs come in packages ranging up to 1000 pins or more.

Programmable Arrays

All PLDs consist of programmable arrays. A programmable array is essentially a grid of conductors that form rows and columns with a fusible link at each cross point. Arrays can be either fixed or programmable. The earliest type of programmable array, dating back to the 1960s, was a matrix with a diode at each cross point of the matrix.

The OR Array The original diode array evolved into the integrated OR array, which consists of an array of OR gates connected to a programmable matrix with fusible links at each cross point of a row and column, as shown in Figure (8- a). The array is programmed by blowing fuses to eliminate selected variables from the output functions, as illustrated in part (b) for a specific case. For each input to an

OR gate, only one fuse is left intact in order to connect the desired variable to the gate input. Once a fuse is blown, it cannot be reconnected.

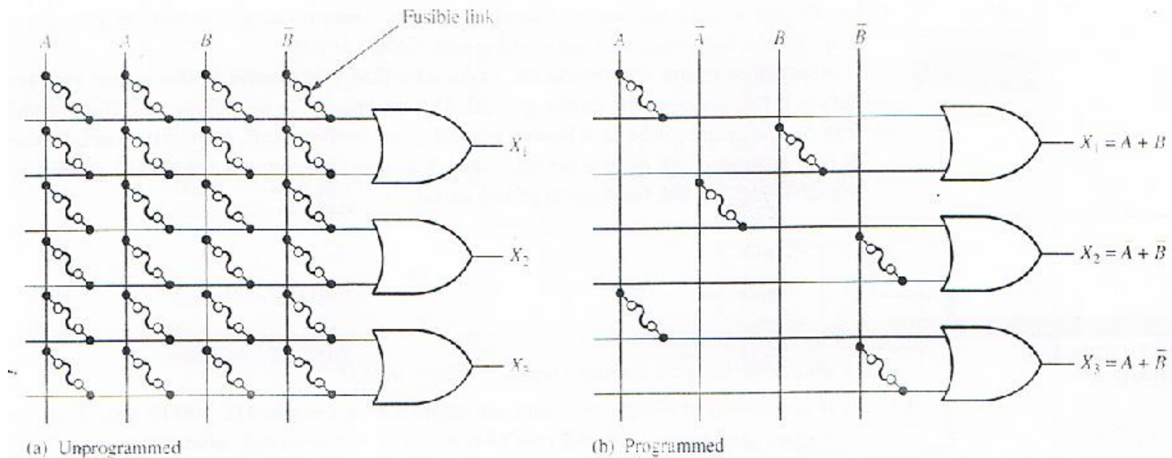


Figure (8)

The AND Array This type of array consists of AND gates connected to a programmable matrix with fusible-links at each cross point, as shown in Figure (9-a). Like the OR array, the AND array is programmed by blowing fuses to eliminate variables from the output function, as illustrated in part (b). For each input to an AND gate, only one fuse is left intact in order to connect the desired variable to the gate input. Also like the OR array, the AND array with fusible links is one-time programmable.

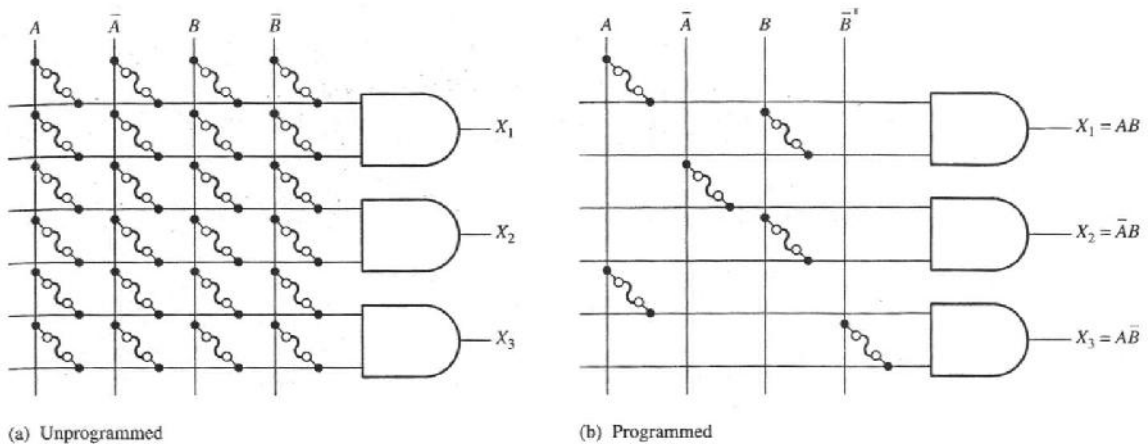


Figure (9)

Classifications of PLDs

PLDs are classified according to their ***architecture,*** which is basically the internal functional arrangement of the elements that give a device its particular operating characteristic.

Programmable Read-Only Memory The ***PROM*** consists of a set of fixed (nonprogrammable) AND gates connected as a decoder and a programmable OR array, as shown in the generalized block diagram of Figure (10). The PROM is used primarily as an addressable memory and not as a logic device because of limitations imposed by the fixed AND gates.

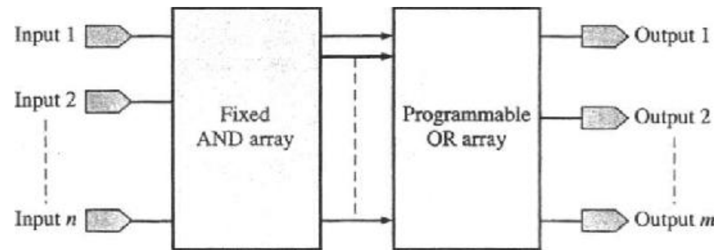


Figure (10)

Programmable Logic Array (PLA) The ***PLA*** is a PLD that consists of a programmable AND array and a programmable OR array, as shown in Figure (11). The PLA was developed to overcome some of the limitations of the PROM. The PLA is also called an FPLA (field-programmable logic array) because the user in the field, not the manufacturer, programs it.

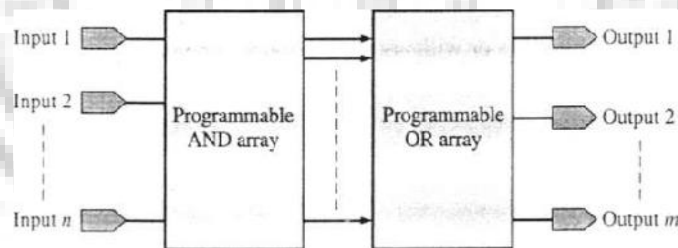


Figure (11)

Programmable Array Logic (PAL) The ***PAL*** is a PLD that was developed to overcome certain disadvantages of the PLA, such as longer delays due to the additional fusible links that result from using two programmable arrays and more circuit complexity. The basic PAL consists of a programmable AND array and a fixed OR array with output logic, as shown in Figure (12). The PAL is the most common one-time programmable logic device (OTP) and is implemented with bipolar technology (TTL or ECL).

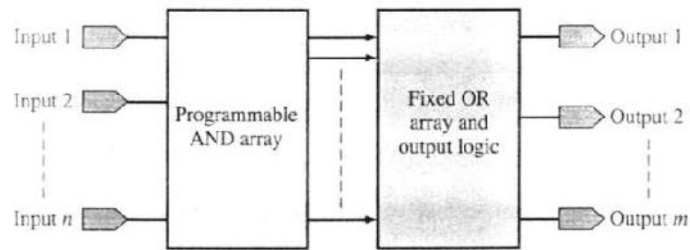


Figure (12)

Generic Array Logic (GAL) The GAL has a reprogrammable AND array and a fixed OR array with programmable output logic. The two main differences between GAL and PAL devices are (a) the GAL is *reprogrammable* and (b) the GAL has programmable output configurations. The GAL can be reprogrammed again and again because it uses E^2CMOS (electrically erasable CMOS) technology instead of bipolar technology and fusible links. The block diagram of a GAL is shown in Figure (13).

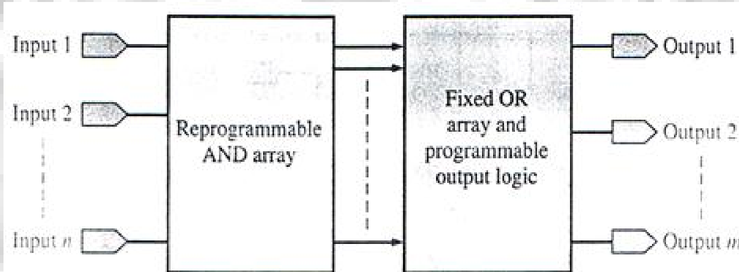


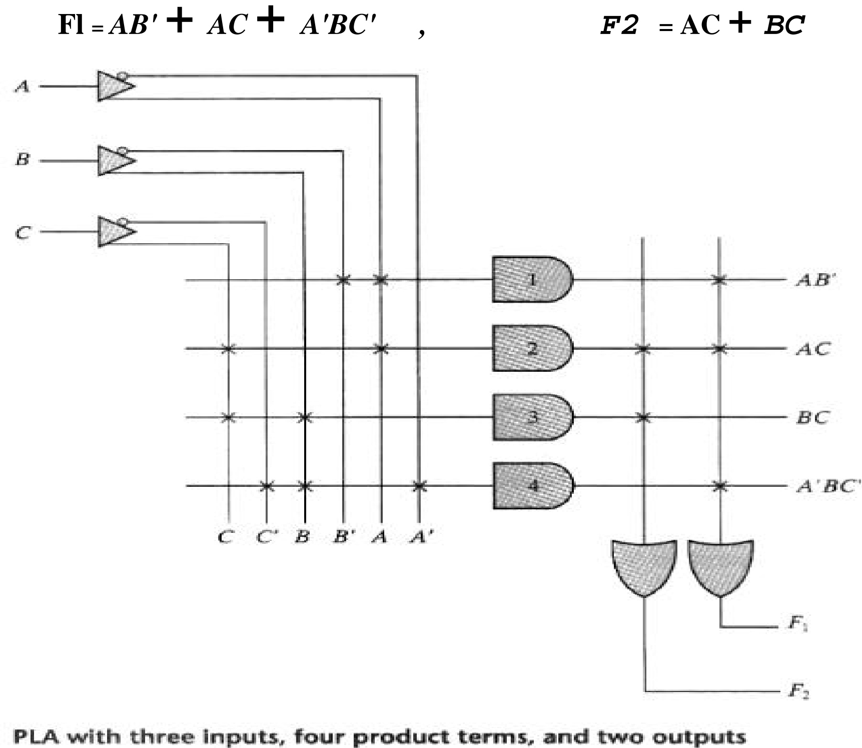
Figure (13)

Programmable Logic Array (PLA)

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 minters, 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 Figure bellow. 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 particular Boolean functions implemented in the PLA of Fig. bellow are:

$$F1 = AB' + AC + A'BC' \quad , \qquad F2 = AC + BC$$



PLA with three inputs, four product terms, and two outputs

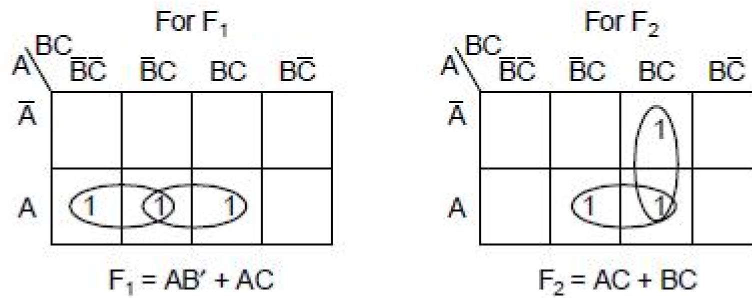
<u>Ex:</u> Implement the following functions using 3-input, 3 product terms and 2output PLA.

F1 = (4, 5, 7)
F2 = (3, 5, 7)

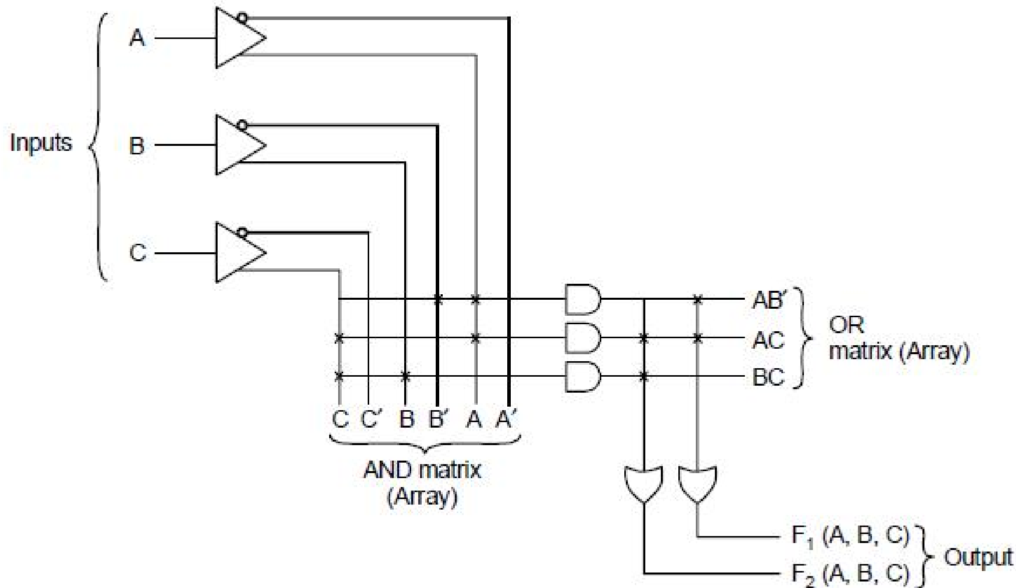
Solution: Step 1. Derive the truth table for the combinational circuit.

Inputs			*Outputs*	
A	*B*	*C*	F_2	F_1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

Step 2. Simplify functions using K-map.



Step 3. Draw the logic diagram



Programmable Array Logic (PAL)

The PAL consists of a programmable array of AND gates that connects to a fixed array of OR gates. This structure allows any sum-of-products (SOP) logic expression with a defined number of variables to be implemented. The basic structure of a PAL is illustrated in Figure (14) for two input variables and one output although most PALs have many inputs and many outputs. As you know, a programmable array is essentially a grid of conductors forming rows and columns with a fusible link at each cross point. Each fused cross point of a row and column is called a cell and is the programmable element of a PAL. Each row is connected to the input of an AND gate and each column is connected to an input variable or its complement. By using the presence or absence of fused connections created by programming, any combination of input variables or complements can be applied to an AND gate to form any desired product term.

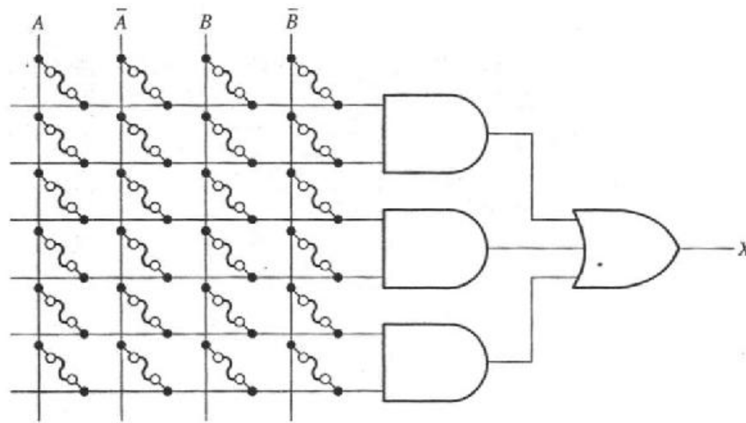


Figure (14)

Implementing** a **Sum-of-Products Expression In its simplest form, each cell in a basic AND array consists of a fusible link connecting a row and a column as represented in Figure (14). When the connection between a row and column is required, the fuse is left intact. When no connection between a row and column is required, the fuse is blown open during the programming process.

As an example, a simple array is programmed as shown in Figure (15) so that the product term *AB* is produced by the top AND gate, *AB* by the middle AND gate, and *A B* by the bottom AND gate. As you can see, the fusible links are left intact to connect the desired variables or their complements to the appropriate AND gate inputs. The fusible links are opened where a variable or its complement is not used in a given product term. The final output from the OR gate is the SOP expression,

$$X = AB + A\grave{B} + \grave{A}\grave{B}$$

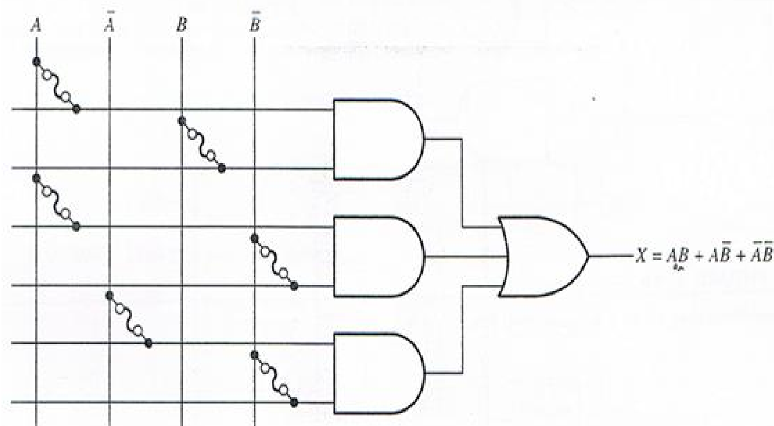


Figure (15)

Simplified Symbols

What you have seen so far represents a small segment of a typical PAL. Actual PALs have many AND gates and many OR gates in addition to other circuitry and are capable of handling many input variables and their complements. Since PALs are very complex integrated circuit devices, manufacturers have adopted a simplified notation for the logic diagrams to keep them from being overwhelmingly complicated.

Input Buffers The input variables to a PAL are buffered to prevent loading by the large number of AND gate inputs to which a variable or its complement may be connected. An inverting buffer produces the complement of an input variable. The symbol representing the **buffer** circuit that produces both the variable and its complement on its outputs is shown in Figure (16) where the bubble output is the complement.

AND Gates A typical PAL AND array has an extremely large number of interconnecting lines, and each AND gate has multiple inputs. PAL logic diagrams show an AND gate that actually has several inputs by using an AND gate symbol with a single input line representing all of its input lines, as indicated Figure (16). Also, multiple input lines are sometimes indicated by a slash and the number of lines as shown in the top AND gate for the case of four lines.

PAL Connections To keep a logic diagram as simple as possible, the fusible links in a programmed AND array are indicated by an X at the cross point if the fuse is left intact and by the absence of an X if the fuse is blown, as indicated in Figure (16). Fixed connections use the standard dot notation, as also indicated.

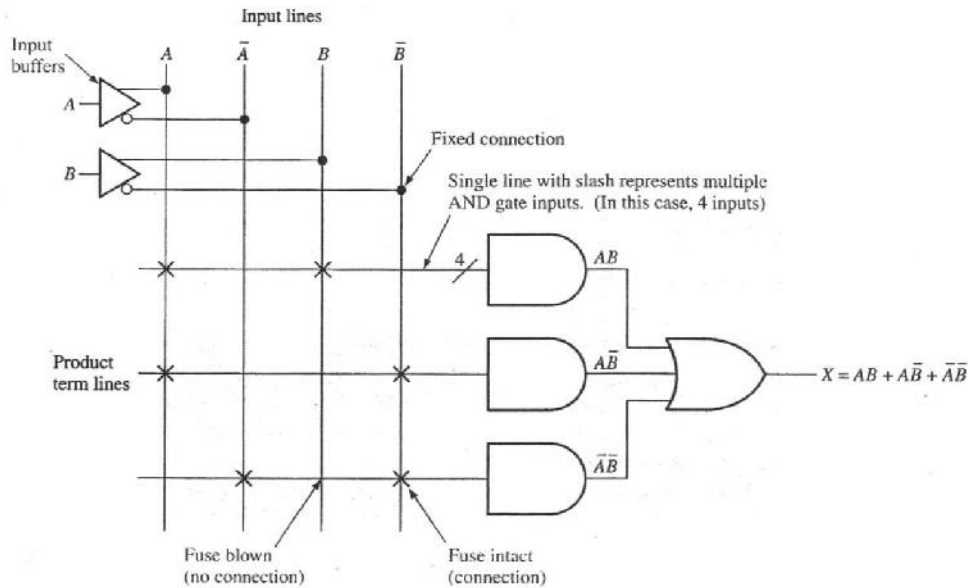


Figure (16)

<u>*EX:*</u> Show how a PAL is programmed for the following 3-variable logic function:

$$X = A\grave{B}C + \grave{A}B\grave{C} + \grave{A}\grave{B} + AC$$

Solution The programmed array is shown in Figure (17). The intact fusible links are indicated by small Xs. The absence of an X means that the fuse has been blown.

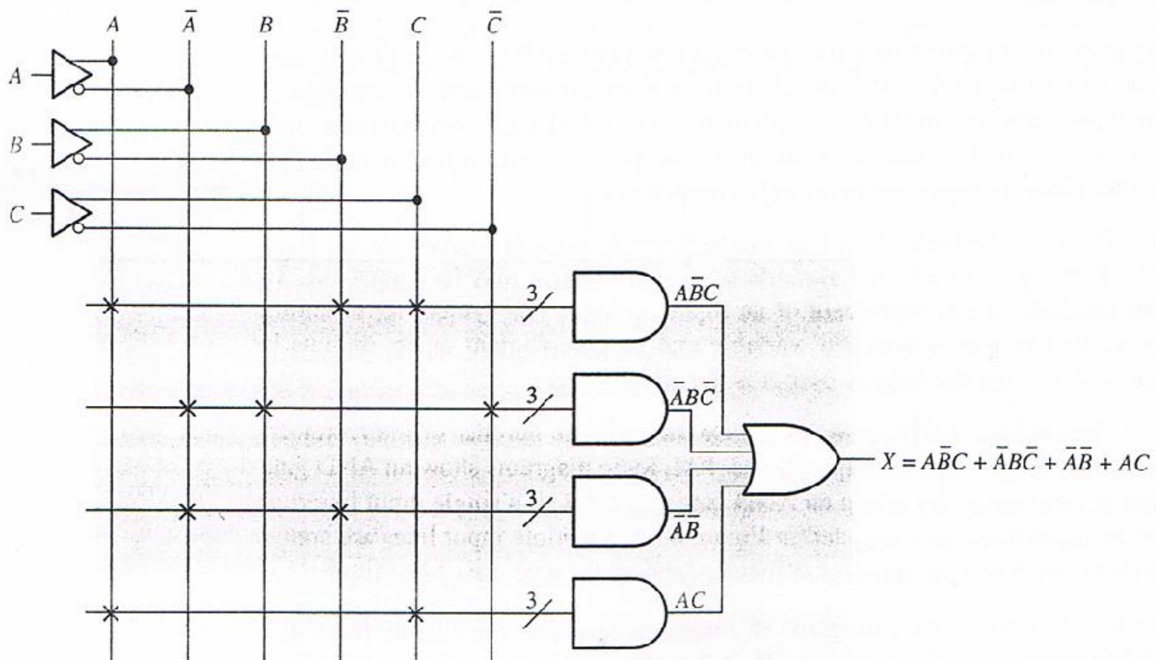
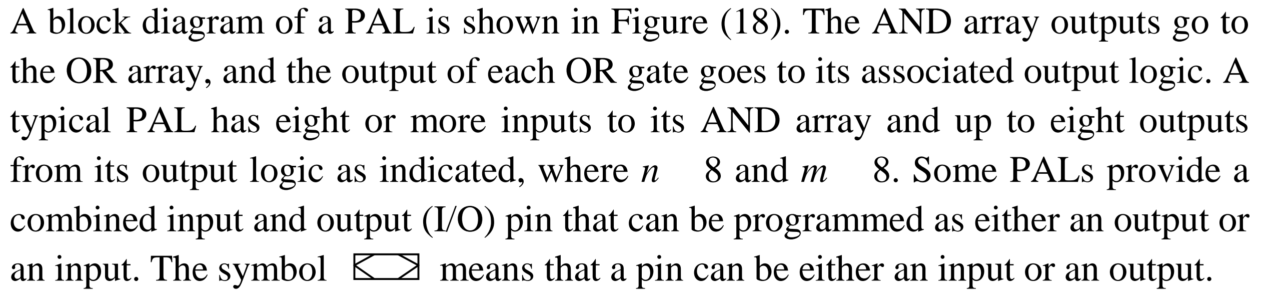


Figure (17)

The PAL Block Diagram



A block diagram of a PAL is shown in Figure (18). The AND array outputs go to the OR array, and the output of each OR gate goes to its associated output logic. A typical PAL has eight or more inputs to its AND array and up to eight outputs from its output logic as indicated, where *n* 8 and *m* 8. Some PALs provide a combined input and output (I/O) pin that can be programmed as either an output or an input. The symbol means that a pin can be either an input or an output.

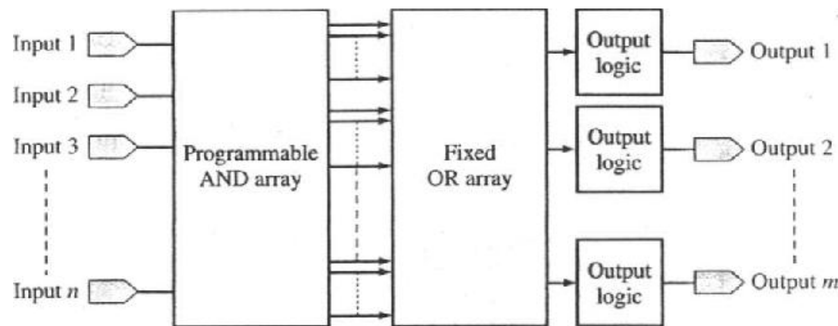


Figure (18)

PAL Output Combinational Logic

There are several basic types of PAL output logic that allows you to configure the device for a specific application. In this chapter, only the aspects of the output logic related to combinational logic functions are discussed. Figure (19) shows three basic types of combinational output logic with tri-state outputs and the associated OR gate. The following are types of PAL output logic:

Combinational output: This output is used for an SOP function and is usually available as either an active-LOW or an active-HIGH output.

Combinational input/output (I/O): This output is used when the output function must feed back to be an input to the array or be used to make the I/O pin an input only)

Programmable polarity output This output is used for selecting either the output function or its complement by programming the exclusive-OR gate for inversion or no inversion. The fusible link on the exclusive-OR input is blown open for inversion and left intact for no inversion.

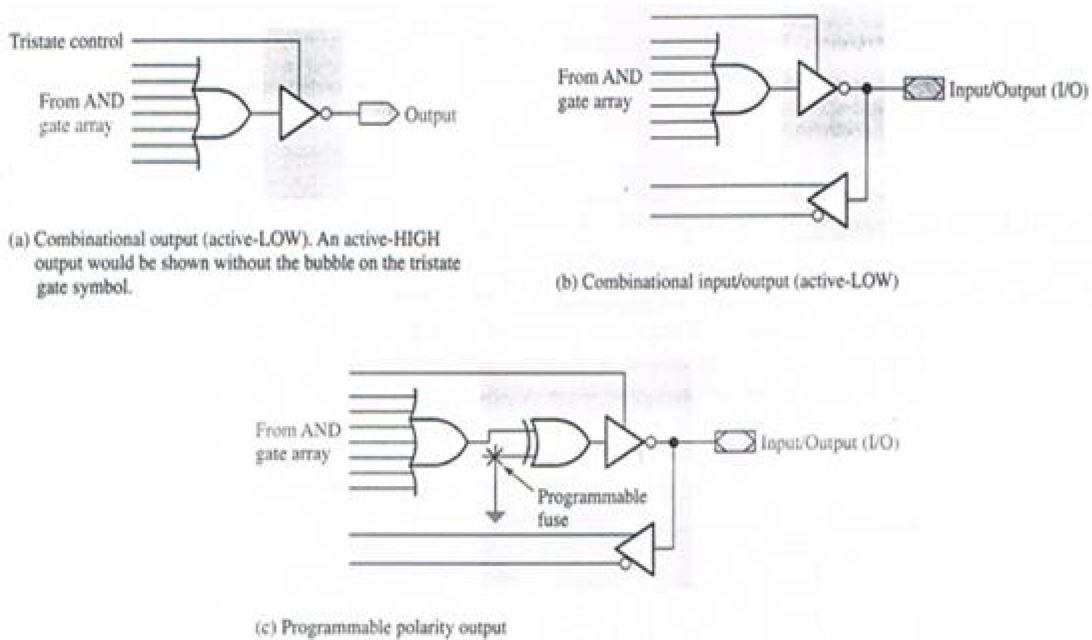
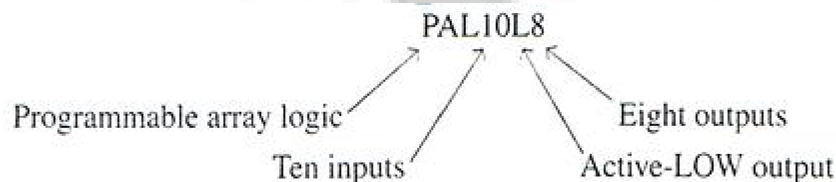


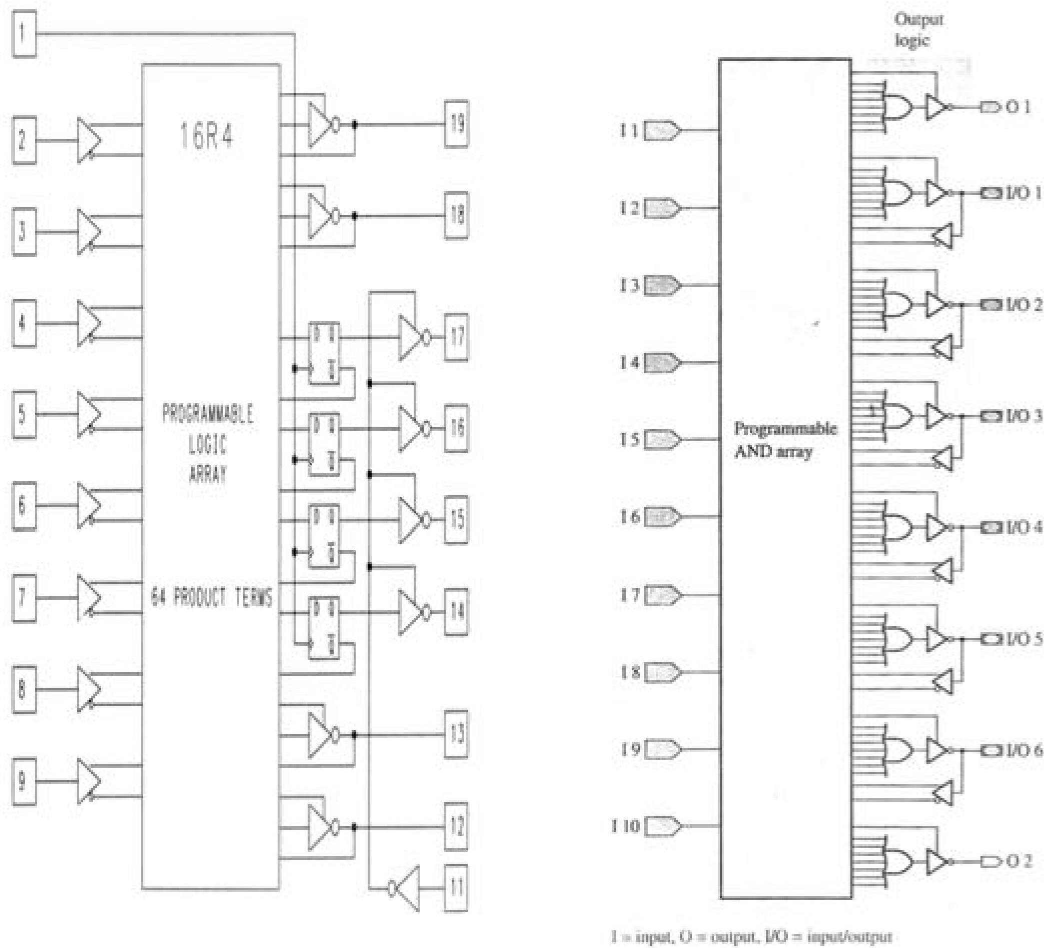
Figure (19)

Standard PAL Numbering

Standard PALs come in a variety of configurations, each of which is identified by a unique part number. This part number always begins with the prefix PAL. The first two digits following PAL indicate the number of inputs, which includes outputs that can be configured as inputs. The letter following the number of inputs designates the type of output: L—active- LOW, H—active-HIGH, or P—programmable polarity. The one or two digits that follow the output type is the number of outputs. The following number is an example.



In addition, a PAL part number may carry suffixes that specify speed, package type, and temperature range. As an example of a PAL configuration, a block diagram of the PAL16L8 and PAL16R4 are shown in Figure (20).



Block diagram of the PAL16R4 **Block diagram of the PAL16P8**

Figure (20)

Generic Array Logic (GAL)

The GAL basically consists of a reprogrammable array of AND gates that connects to a fixed array of OR gates. Just as in a PAL, this structure allows any sum-of-products (SOP) logic expression with a defined number of variables to be implemented.

The basic structure of a GAL is illustrated in Figure (21) for two input variables and one output although most GALs have many inputs and many outputs. The reprogrammable array is essentially a grid of conductors forming rows and columns with an electrically erasable CMOS (E^2CMOS) cell at each cross point, rather than a fuse as in a PAL. These cells are shown as blocks in the figure.

Each row is connected to the input of an AND gate, and each column is connected to an input variable or its complement. By programming each E^2CMOS

cell to be either *on* or *off,* any combination of input variables or complements can be applied to an AND gate to form any desired product term. A cell that is *on* effectively connects its corresponding row and column, and a cell that is *off* disconnects the row and column. The cells can be electrically erased and reprogrammed. A typical E^2CMOS cell can retain its programmed state for 20 years or more.

Implementing* a *Sum-of-Products Expression As an example, a simple GAL array is programmed as shown in Figure (22) so that the product term *AB* is produced by the top AND gate, AB by the middle AND gate, and A B by the bottom AND gate. As shown, the E2CMOS cells are on to connect the desired variables or their complements to the appropriate AND gate inputs. The E2CMOS cells are off where a variable or its complement is not used in a given product term. The final output from the OR gate is an SOP expression.

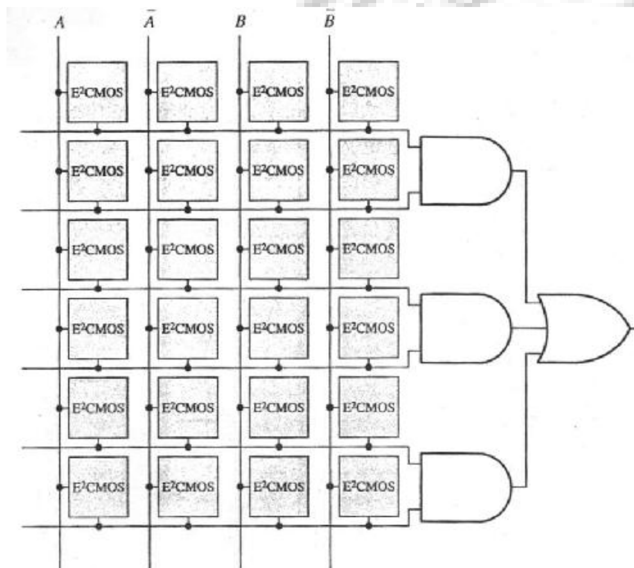


Figure (21) *Basic E^2CMOS array structure of a GAL*

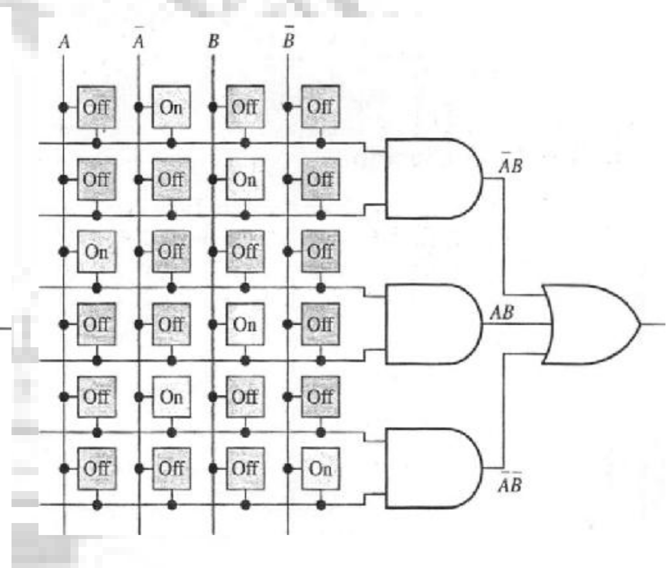
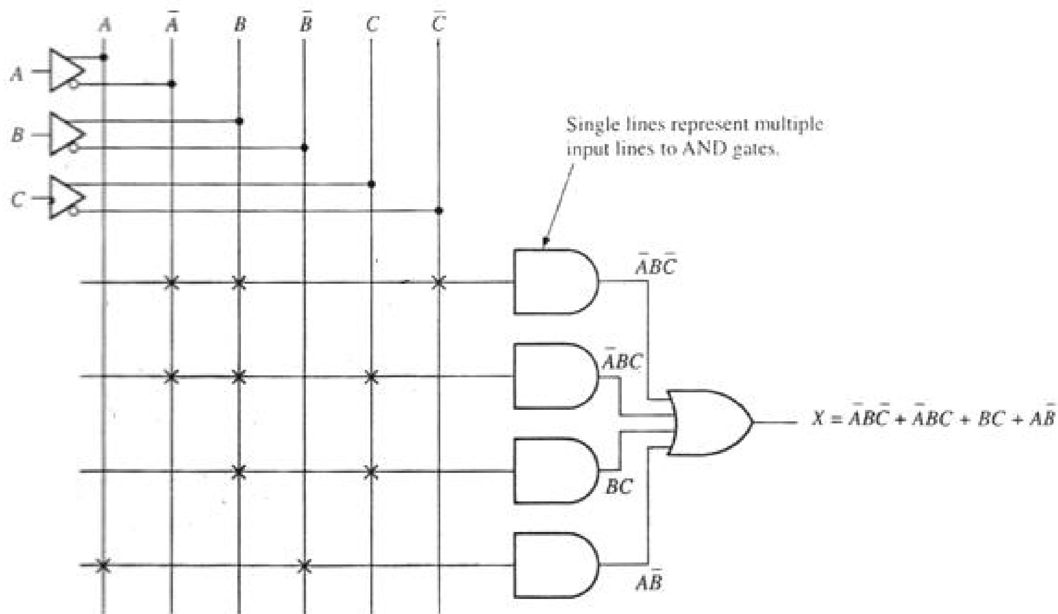


Figure (22) *GAL implementation of a sum-of-products expression*

<u>*Ex:*</u> Show how a GAL is programmed for the following 3-variable logic function:

$$X = \grave{A}B\grave{C} + \grave{A}BC + BC + A\grave{B}$$

Solution The programmed array using simplified notation is shown bellow. Cells that are *on* are indicated by small Xs. The absence of an X means that the cell is *off*



The GAL Block Diagram A block diagram of a GAL is shown in Figure (23). The AND array outputs go to the output logic macrocells (OLMC), which contain the OR gates and programmable logic. A typical GAL has eight or more inputs to its AND array and eight or more input/outputs (I/Os) from its OLMCs as indicated, where *n* 8 and *m* 8. The OLMC is made up of logic circuits that can be programmed as either *combinational logic* or as *registered logic.* The OLMC provides much more flexibility than the fixed output logic in a PAL.

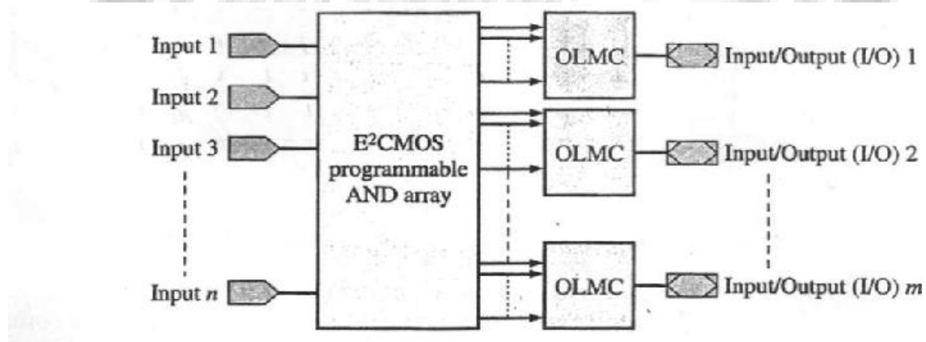
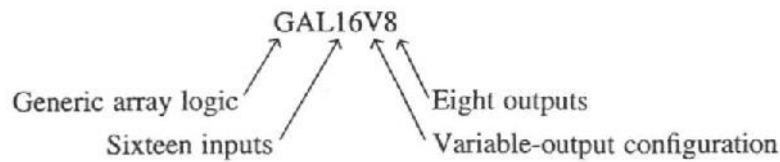


Figure (23)

Standard GAL Numbering GALs come in a variety of configurations, each of which is identified by a unique part number. This part number always begins with the prefix GAL. The first two digits following the prefix indicate the number of inputs, which includes outputs that can be configured as inputs. The letter V following the number of inputs designates a variable-output configuration. The

one or two digits that follow the output type is the number of outputs. The following



THE GAL22V10

The GAL22V10 contains twelve dedicated inputs and ten input/outputs (I/Os) as shown in the block diagram of Figure 7-18. This device is available in either a 24-pin DIP (dual in-line package) or a 28-pin PLCC (plastic chip carrier), as shown in Figure (24). This device is also available in a low-voltage version, the GAL22LV10.

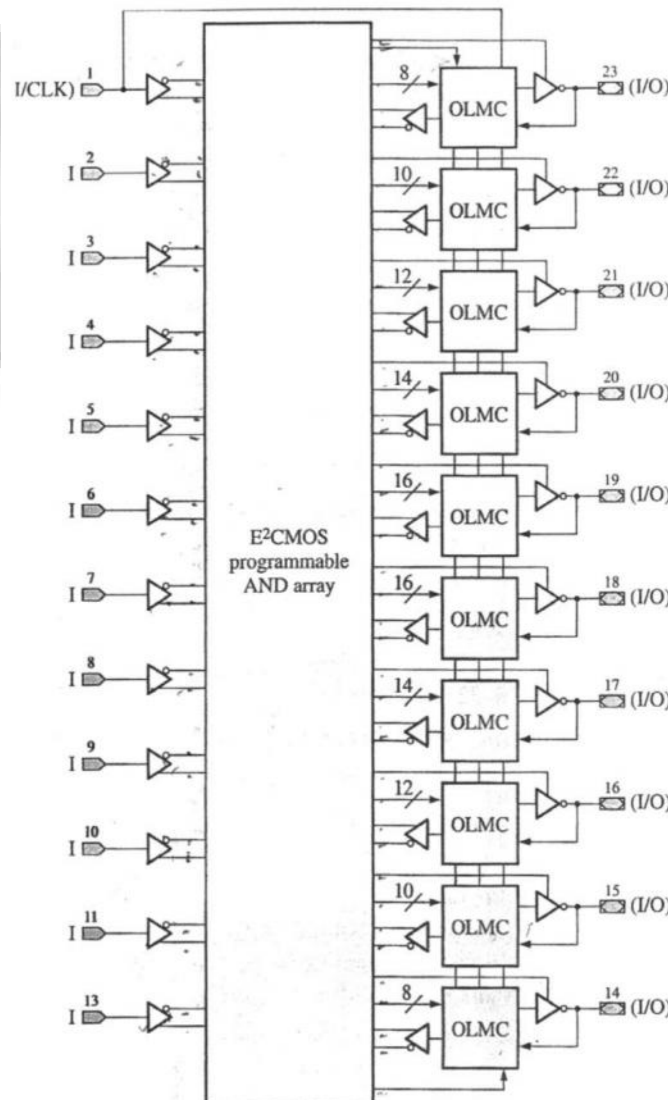


Figure (24)

The Output Logic Macrocells (OLMCs)

As stated in the discussion of GALs, an OLMC contains programmable logic circuits that can be configured either for a combinational output or input or for a registered output. In the registered mode, the output comes from a flip-flop.

As indicated by the notation in the block diagram of Figure (24), of the ten available GAL22V10 OLMCs, two have eight product terms (lines from the AND array to the OR gate), two have ten product terms, two have twelve product terms, two have fourteen product terms, and two have sixteen product terms. Each OLMC can be programmed for either an active-HIGH or an active-LOW output. Also, each OLMC can be programmed as an input.

Logic Diagram A basic logic diagram for the GAL22V10 OLMC is shown in Figure (25). The inputs from the AND gates to the OR gate vary from ten to sixteen, as mentioned. The logic inside the gray box consists of a flip-flop and two multiplexers.

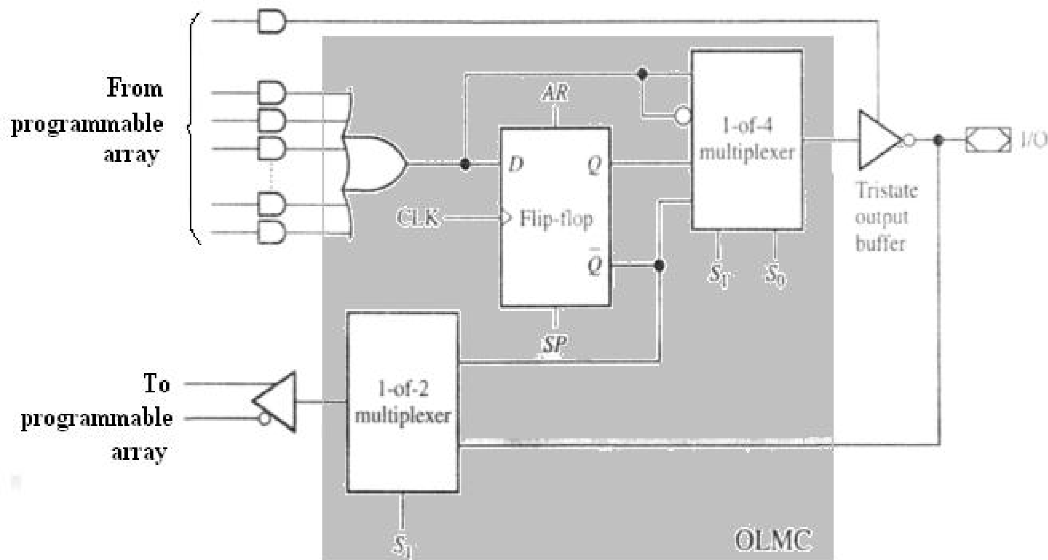
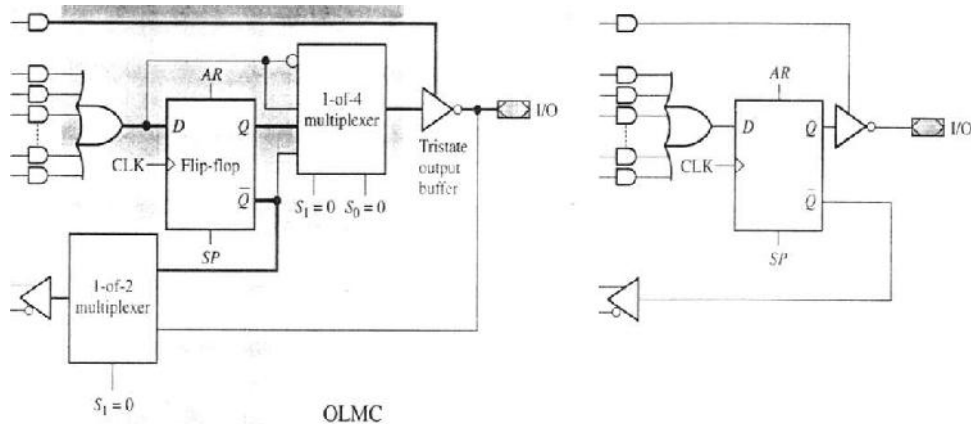


Figure (25)

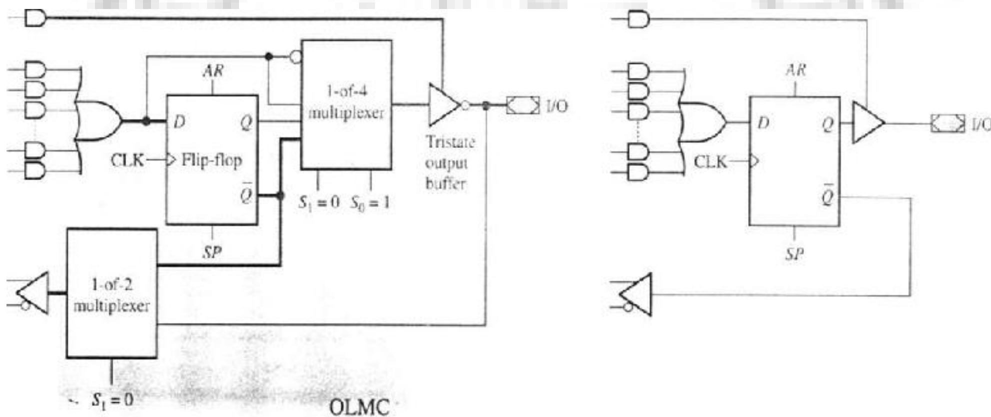
The l-of-4 multiplexer connects one of its four input lines to the tri-state output buffer, based on the states of two select inputs, S_0 and S_1. The inputs to the l-of-4 multiplexer are the OR gate output, the complement of the OR gate output, the flip-flop output Q, and the complement of the flip-flop output $\bar{Q}$. This allows the output of the OLMC to be either active-HIGH or active-LOW in each mode. The l-of-2 multiplexer connects either the output of the tri-state buffer or the $\bar{Q}$ output of the flip-flop back through a buffer to the AND array based on the state of S_1. The select bits, S_0 and S_1, for each OLMC are programmed into a dedicated group of cells in the array by the compiler software, so the user does not have to directly manipulate these bits.

The OLMC of the type shown in Fig.(25) can be configured to produce four different outputs depending upon the selection inputs. These include the following:

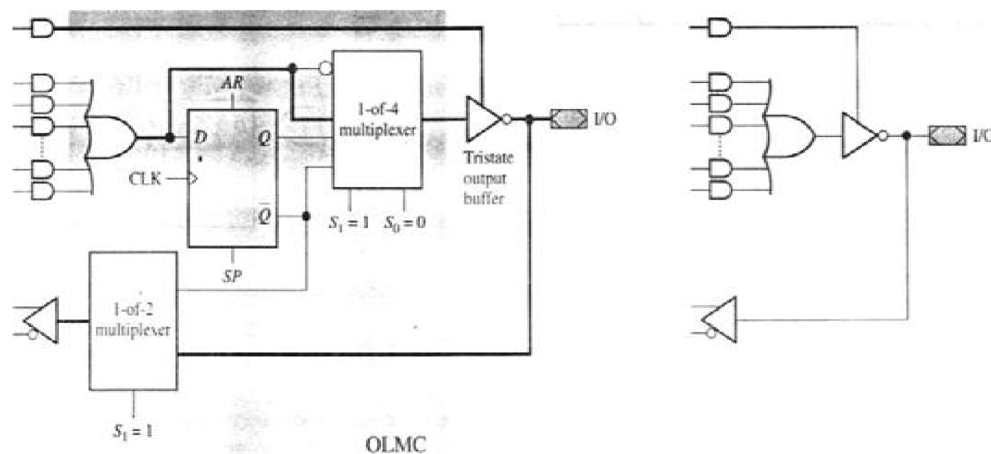
1. $S_1S_0= 00$: registered mode with active LOW output, Fig. (26- a).
2. $S_1S_0= 01$: registered mode with active HIGH output, Fig. (26- b).
3. $S_1S_0= 10$: combinational mode with active LOW output, Fig. (26- c).
4. $S_1S_0= 11$: combinational mode with active HIGH output, Fig. (26- d).



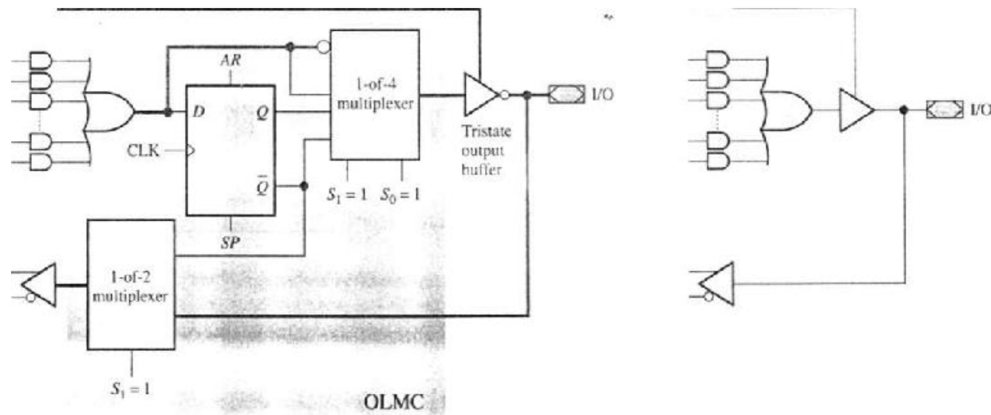
(a) OLMC in the active-LOW registered mode and the effective logic diagram



(b) OLMC in the active-HIGH registered mode and the effective logic diagram



(c) OLMC in the active-LOW combinational mode and the effective logic diagram



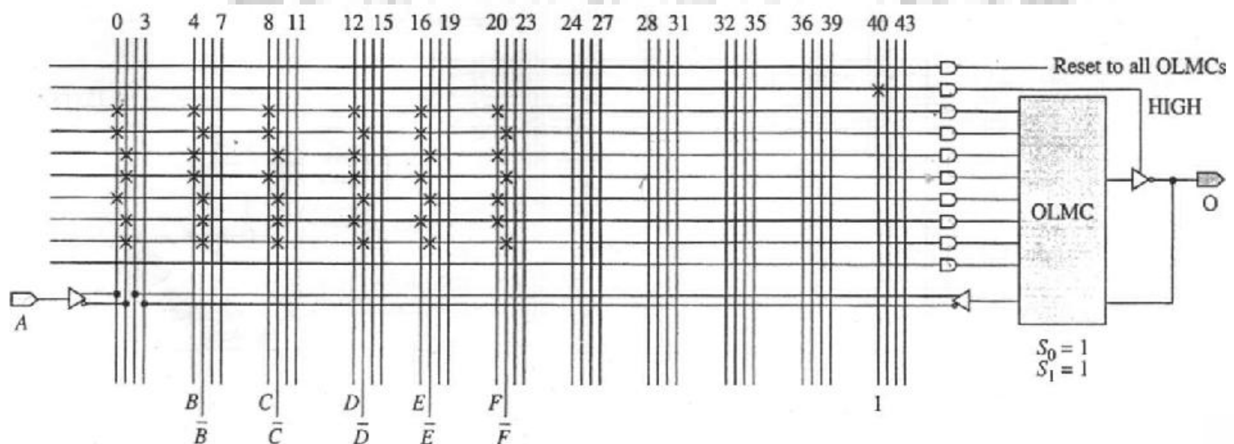
(d) OLMC in the active-HIGH combinational mode and the effective logic diagram

Figure (26)

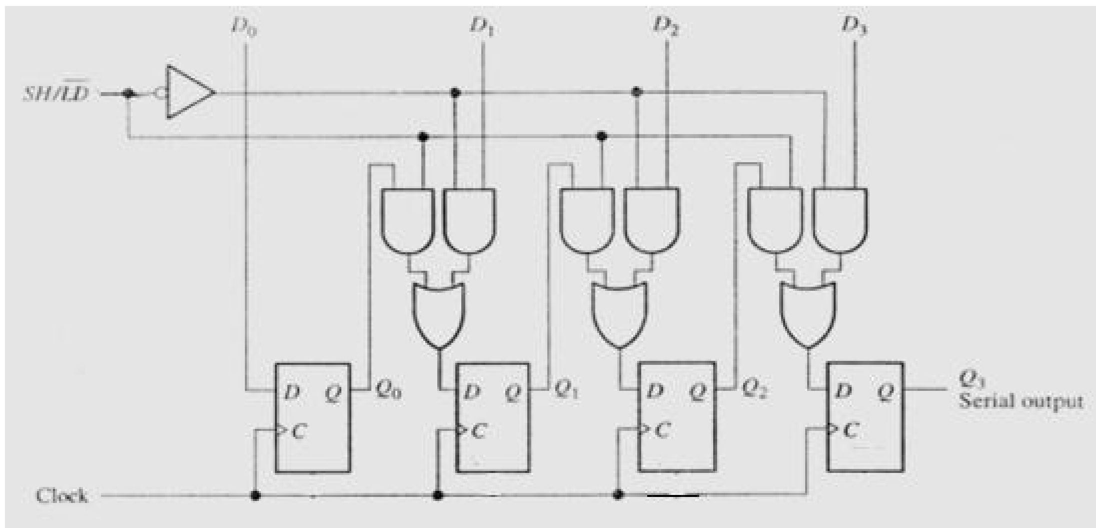
EX: Show how the following 6-variable SOP function is implemented with the GAL22V10.

$$X = ABCDEF + A\bar{B}C\bar{D}E\bar{F} + \bar{A}B\bar{C}D\bar{E}F + \bar{A}BCDE\bar{F} + A\bar{B}\bar{C}\bar{D}\bar{E}F + \bar{A}\bar{B}\bar{C}DEF + \bar{A}\bar{B}\bar{C}\bar{D}\bar{E}\bar{F}$$

Solution The programmed portion of the array with the associated OLMC is shown in Figure bellow. The rest of the array remains unused in this case. The Xs represent E^{2}CMOS cells that are programmed to the *on* state.



EX: Implement [illegible] 4-bit parallel in/serial out shift register in Figure bellow by using (a[illegible]PLA and Flip-flops, (b) - GAL22V10. The $SH/\overline{LD}$ input is the $Shift/\overline{Load}$. When this input is LOW, the parallel data are loaded into the register, and when it is HIGH, data are shifted from left to right. D_0 through D_3 are the parallel data inputs and Q_3 is the serial data output.


D0
D1
D2
D3
SH/LD
D Q
C
Q0
Q1
Q2
Q3
Serial output
Clock

FIELD-PROGRAMMABLE GATE ARRAYS (FPGA)

Field Programmable Gate Array (FPGA) devices were introduced by Xilinx in the mid 1980s. They differ from CPLDs in architecture, storage technology, number of built-in features, and cost, and are aimed at the implementation of high performance, large-size circuits.

FPGAs are flexible, programmable devices with a broad range of capabilities. Their basic structure consists of an array of universal, programmable logic cells embedded in a configurable connection matrix. There are three key parts of FPGA structure: logic blocks, interconnect, and I/O blocks. The I/O blocks form a ring around the outer edge of the part. Each of these provides individually selectable input, output, or bidirectional access to one of the general-purpose I/O pins on the exterior of the FPGA package.

Inside the ring of I/O blocks lies a rectangular array of logic blocks. And connecting logic blocks to logic blocks and I/O blocks to logic blocks is the programmable interconnect wiring.

In FPGAs, CPLD's PLDs are replaced with a much smaller *logic block.* The logic blocks in an FPGA are generally nothing more than a couple of logic gates or a look-up table (LUT) and a flip-flop. The FPGAs use a more flexible and faster interconnection structure than the CPLDs. In the FPGAs, the logic blocks are embedded in a mesh or wires that have programmable interconnect points that can be used to connect two wires together or a wire to a logic block as shown in Fig. (1).

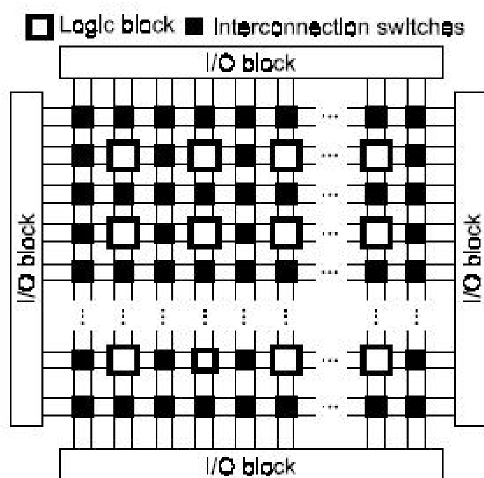


Figure (1)

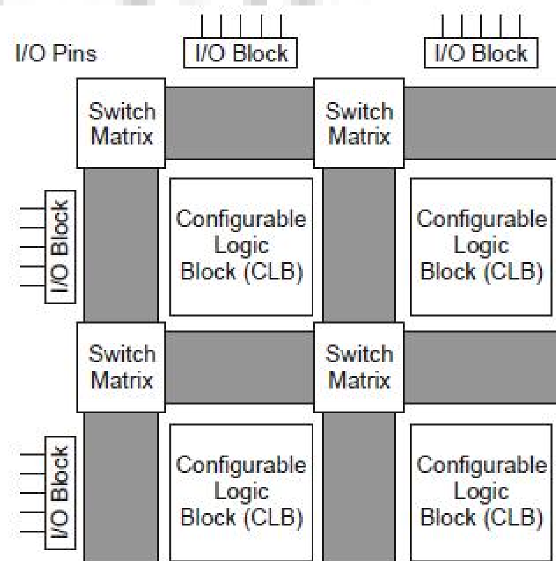


Figure (2)

There are several architectures for FPGAs available but the two popular architectures are that, used by Xilinx and Altera. The Xilinx chips utilize an "island-type" architecture, where logic functions are broken up into small islands of 4–6 term arbitrary functions, and connections between these islands are computed. Fig.(2) illustrates the basic structure of the Xilinx FPGA. Altera's architecture ties the chip inputs and outputs more closely to the logic blocks, as shown in Fig.(3). This architecture places the logic blocks around one central, highly connected routing array. The circuits used to implement combinational logic in logic blocks are lookup tables (LUT). The structure of LUT's in Altera FPGAs is as shown in Fig. (4).

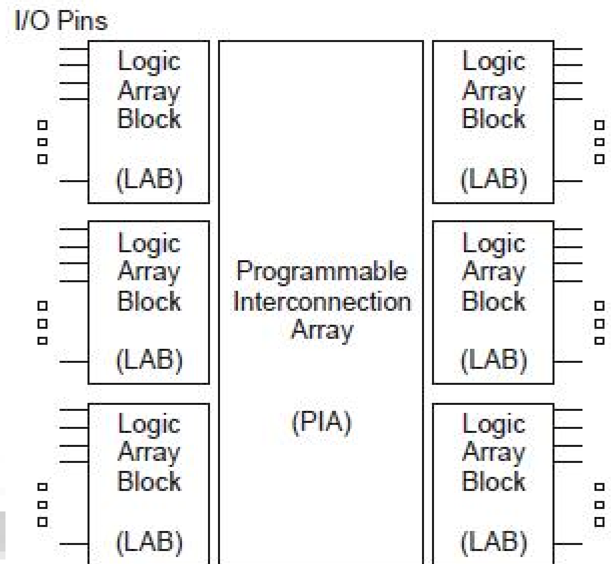


Figure (3)

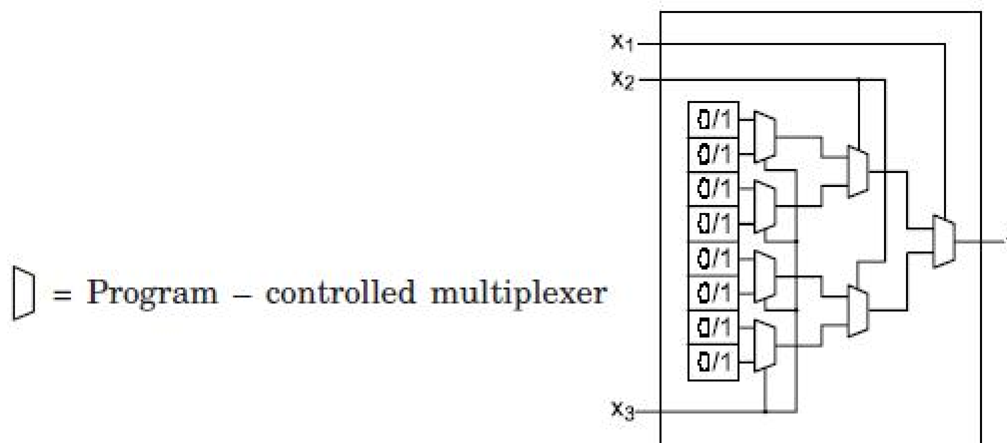


Figure (4)

Figure (5) shows a typical logic block of an FPGA. It consists of a four-input look-up table (LUT) whose output feeds a clocked flip-flop. The output can either be a registered output or an unregistered LUT output. Selection of the output takes place in the multiplexer. An LUT is nothing but a small one-bit wide memory array with its address lines representing the inputs to the logic block and a one-bit output acting as the LUT output. An LUT with n inputs can realize any logic function of n inputs by programming the truth table of the desired logic function directly into the memory.

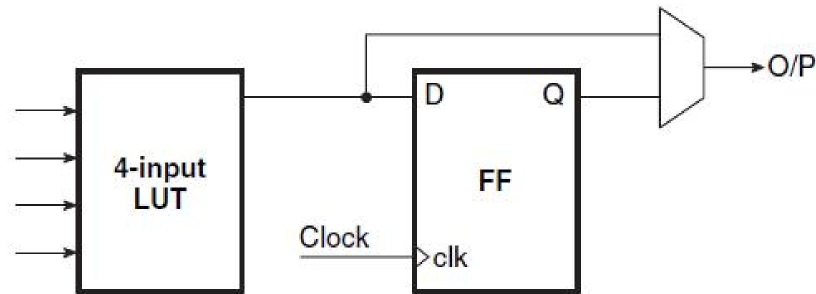
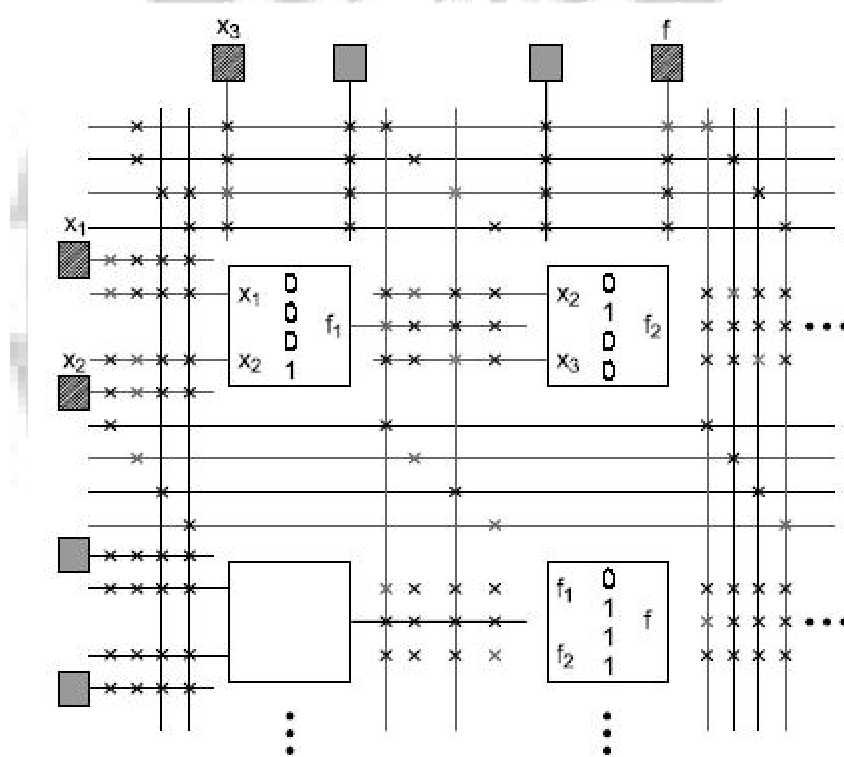


Figure (5)

- As example of programmed FPGA, when :

$$f_1 = x_1 x_2 \ , \quad f_2 = \bar{x}_2 x_3 \ , \quad f_3 = f_1 f_2$$



A few examples of FPGA packages are illustrated in Fig.(6), which shows one of the smallest FPGA packages on the left (64 pins), a medium-size package in the middle (324 pins), and a large package (1,152 pins) on the right.

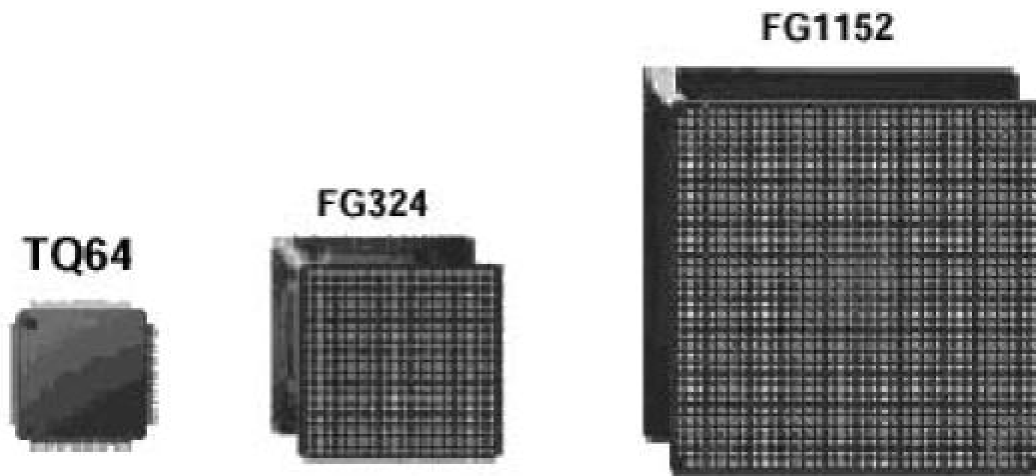


Figure (6)

State Machine Design with SM Charts

Algorithmic State Machine (ASM)

Finite state machines are a powerful tool for designing sequential circuits, but they are lacking in that they do not explicitly represent the algorithms that compute the transition or output functions, nor is timing information explicitly represented. Flowcharts are useful in the hardware design of digital systems. A special type of flowchart is called a state machine flowchart, or SM chart for short. SM charts are also called ASM (algorithmic state machine) charts. The SM chart offers several advantages:

- It is often easier to understand the operation of a digital system by inspection of the SM chart instead of the equivalent state graph.
- A given SM chart can be converted into several equivalent forms, and each form leads directly to a hardware realization.

An SM chart differs from an ordinary flowchart in that certain specific rules must be followed in constructing the SM chart. When these rules are followed, the SM chart is equivalent to a state graph, and it leads directly to a hardware realization. Figure (1) shows the three principal components of an SM chart.

The state of the system is represented by a state box. The state box may contain an output list, and a state code may be placed outside the box at the top. The state name is placed in a circle to the left of the state box. A decision box is represented by a diamond-shaped symbol with true and false branches. The condition placed in the box is a Boolean expression that is evaluated to determine which branch to take. The conditional output box, which has curved ends, contains a conditional output list. The conditional outputs depend on both the state of the system and the inputs.

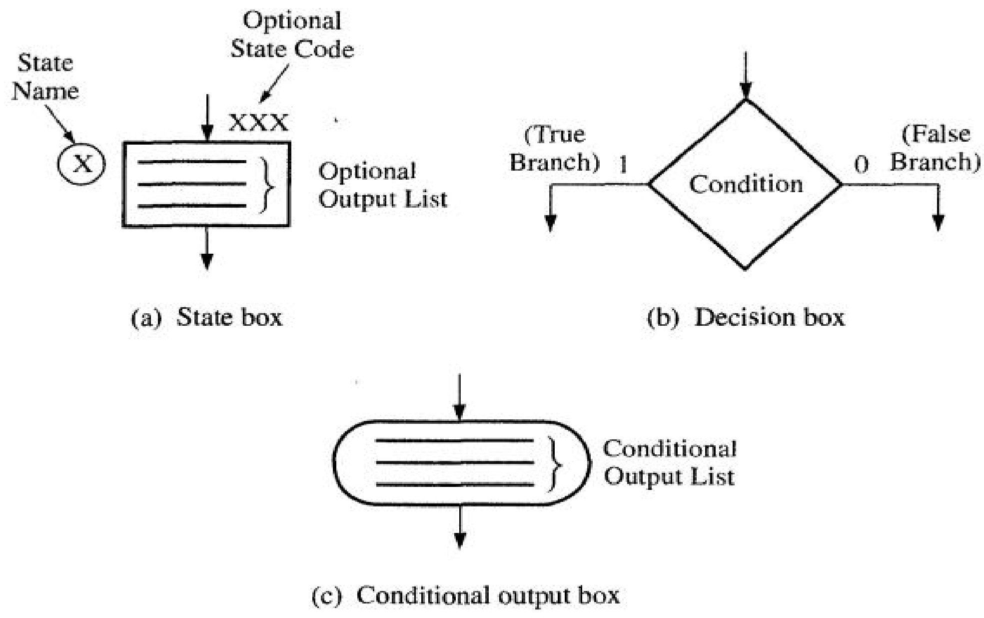


Figure (1) Components of an SM Chart

Figure (2), when state S_1 is entered, outputs Z_1 and Z_2 become 1. If inputs X_1 and X_2 are both equal to 0, Z_3 and Z_4 are also 1, and at the end of the state time the machine goes to the next state via exit path 1. On the other hand, if $X_1 = 1$ and $X_3 = 0$, the output Z_5 is 1 and exit to the next state will occur via exit path 3.

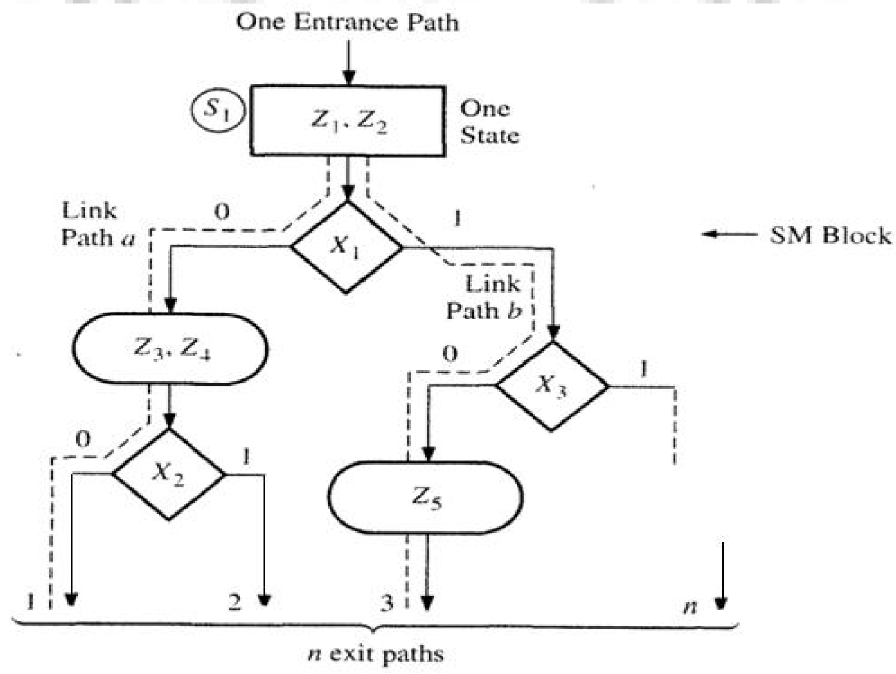


Figure (2) Example of an SM Block

Figure (3) shows two equivalent SM blocks. In both (a) and (b), the output Z_2= 1 if X_1 = 0; the next state is S_2 if X_2 = 0 and S_3 if X_2 = 1.

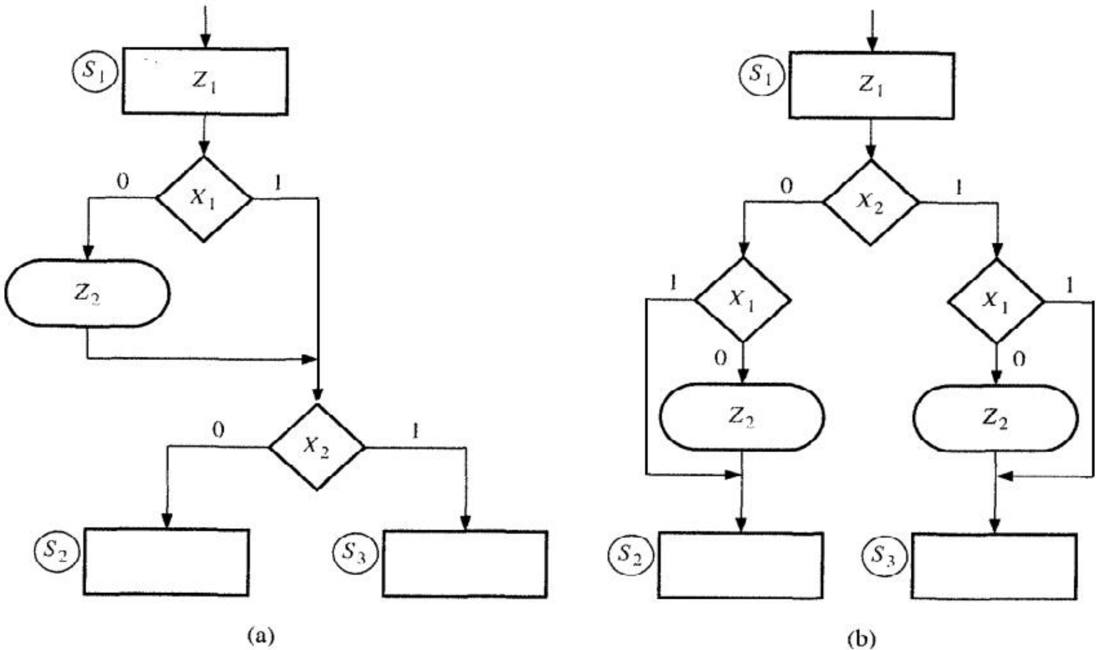


Figure (3) Equivalent SM Blocks

Figure (4) shows two equivalent ASM blocks. (a) Using a single decision box. (b) Using several decision boxes.

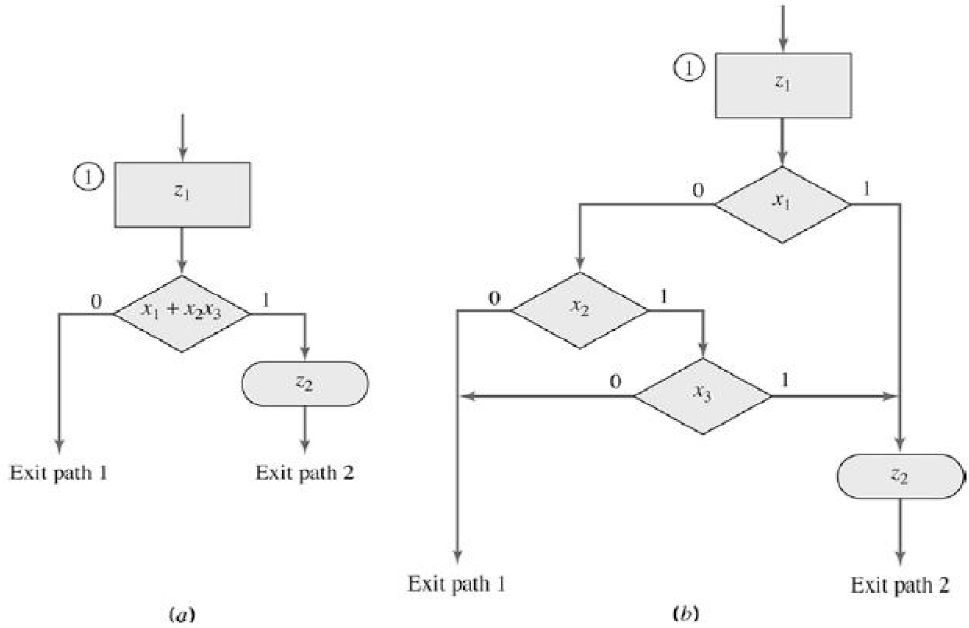
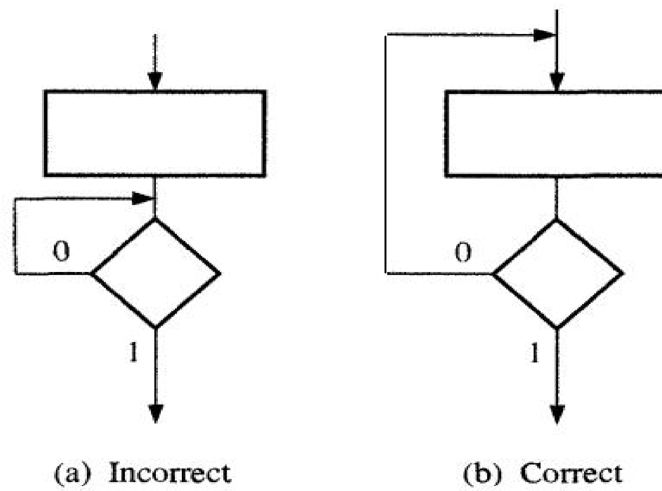


Figure (4)

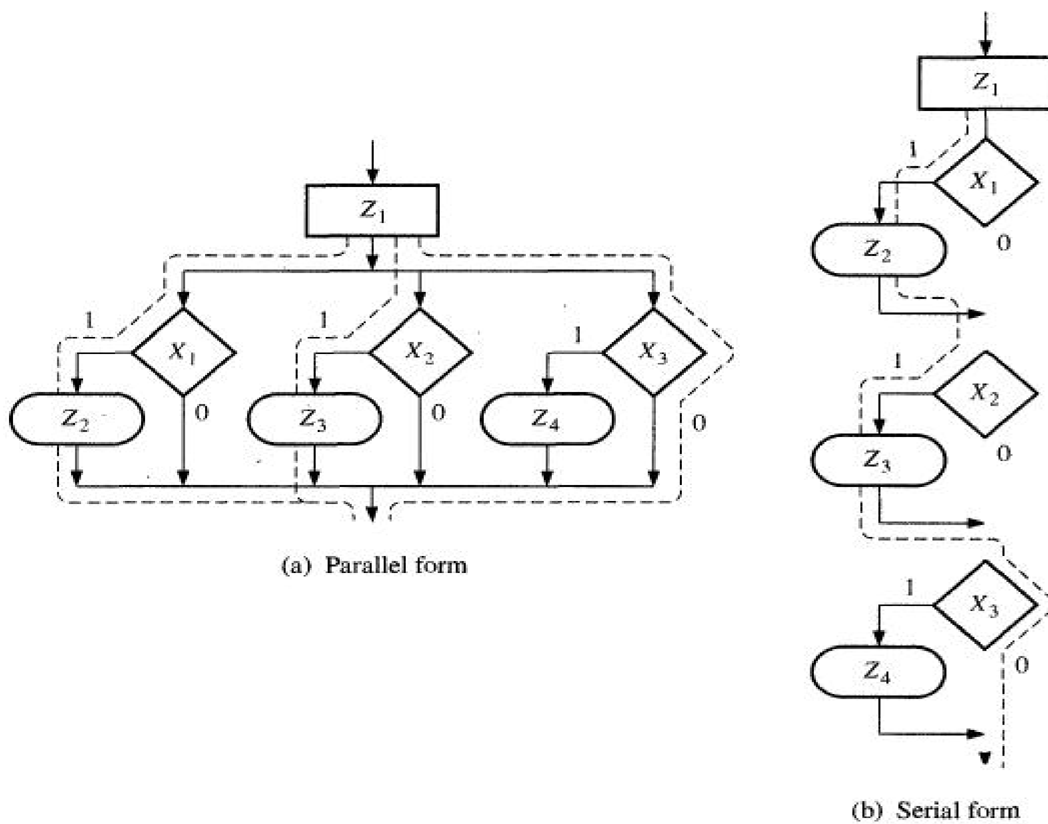
Figure (5) shows an incorrect and correct way of drawing an SM block with feedback.



(a) Incorrect (b) Correct

Figure (4) SM Block with Feedback

Figure (6-a) shows a parallel SM block, which is equivalent to Fig. (6-b). The link path for $X_1 = X_2 = 1$ and $X_3 = 0$ is shown with a dashed line, and the outputs encountered on this path are Z_1, Z_2 , and Z_3.



(a) Parallel form (b) Serial form

Figure (5) Equivalent SM Blocks

Figure (7) shows ASM sharing:

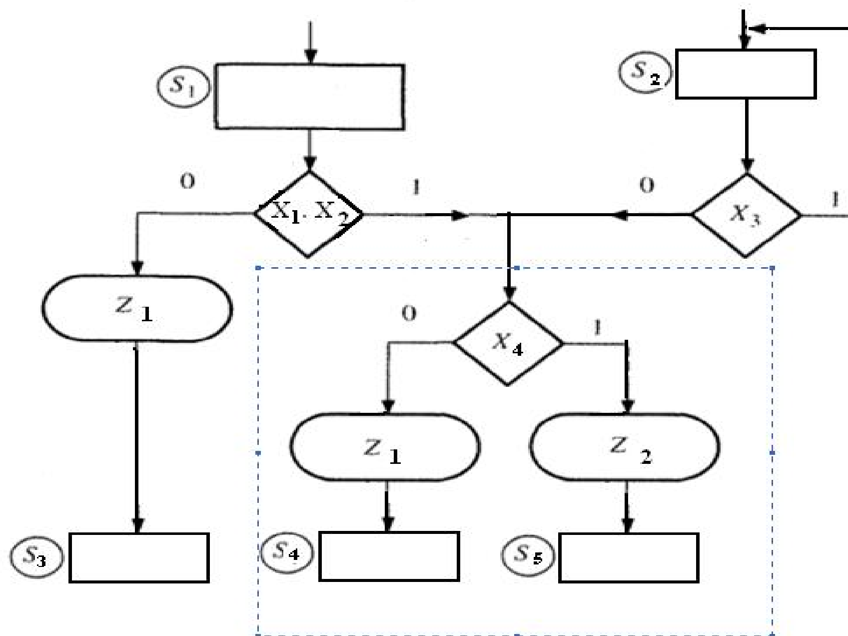
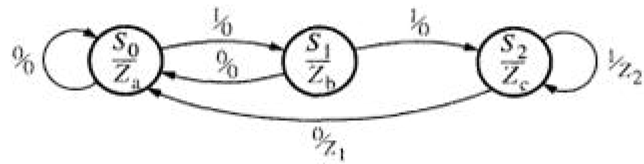
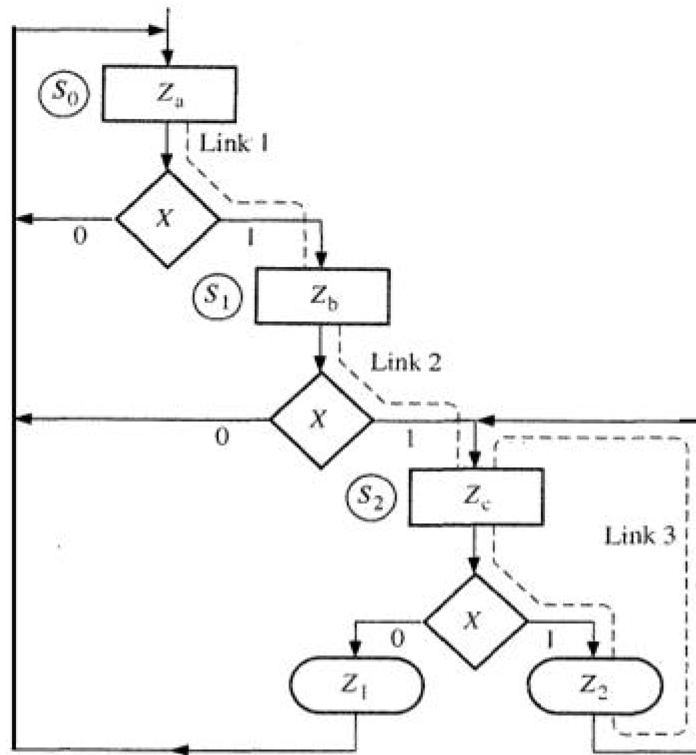


Figure (6) ASM chart with two blocks sharing condition

A state graph for a sequential machine is easy to convert to an equivalent SM chart. The state graph of Fig.(7-a) has both "Moore" and "Mealy" outputs. The equivalent SM chart has three blocks- one for each state. The "Moore" outputs (Z_a, Z_b, Z_c) are placed in the state boxes since they do not depend on the input. The "Mealy" outputs (Z_1 , Z_2) appear in conditional output boxes since they depend on both the state and input. In this example, each SM block has only one decision box since only one input variable must be tested. For both the state graph and SM chart, Z_c is always 1 in state S_2. If X = 0 in state S_2, Z_1 = 1 and the next state is S_0. If X = 1, Z_2 = 1 and the next state is S_2.



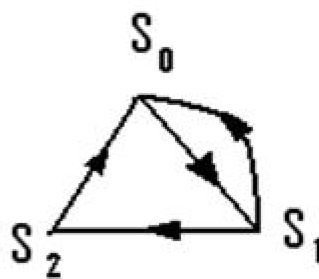
(a) State graph



(b) Equivalent SM chart

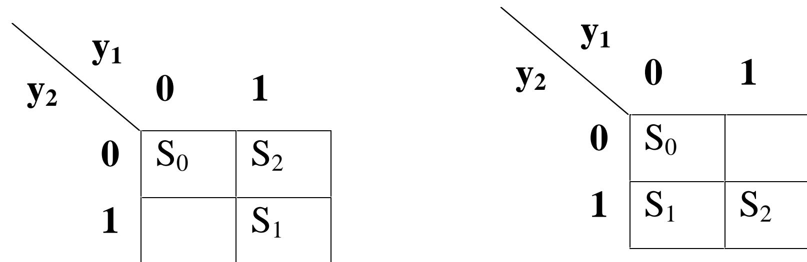
Figure (5) Conversion of a State Graph to an SM Chart

- ***State Assignment:*** we write the transition without slink as:



Assignment graph

Now we draw K-maps :



Map (b) Map (a)

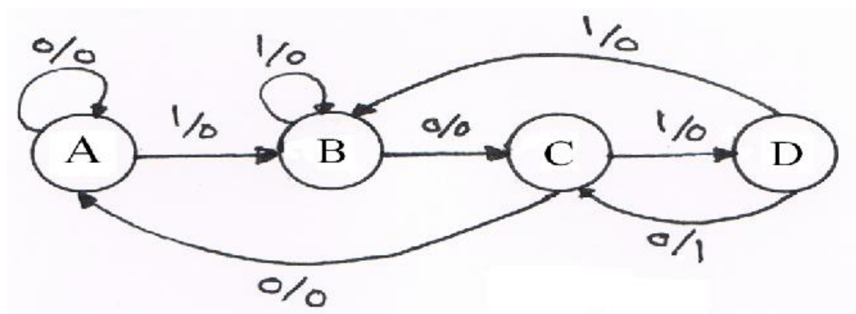
Now, fond a minimum state locus assignment for the ASM chart:

	bit distance map(a)	***bit distance map(b)***
$S_0 \longrightarrow S_1$	1	2
$S_1 \longrightarrow S_0$	1	2
$S_1 \longrightarrow S_2$	1	1
$S_2 \longrightarrow S_0$	2	1
State locus	**5**	**6**

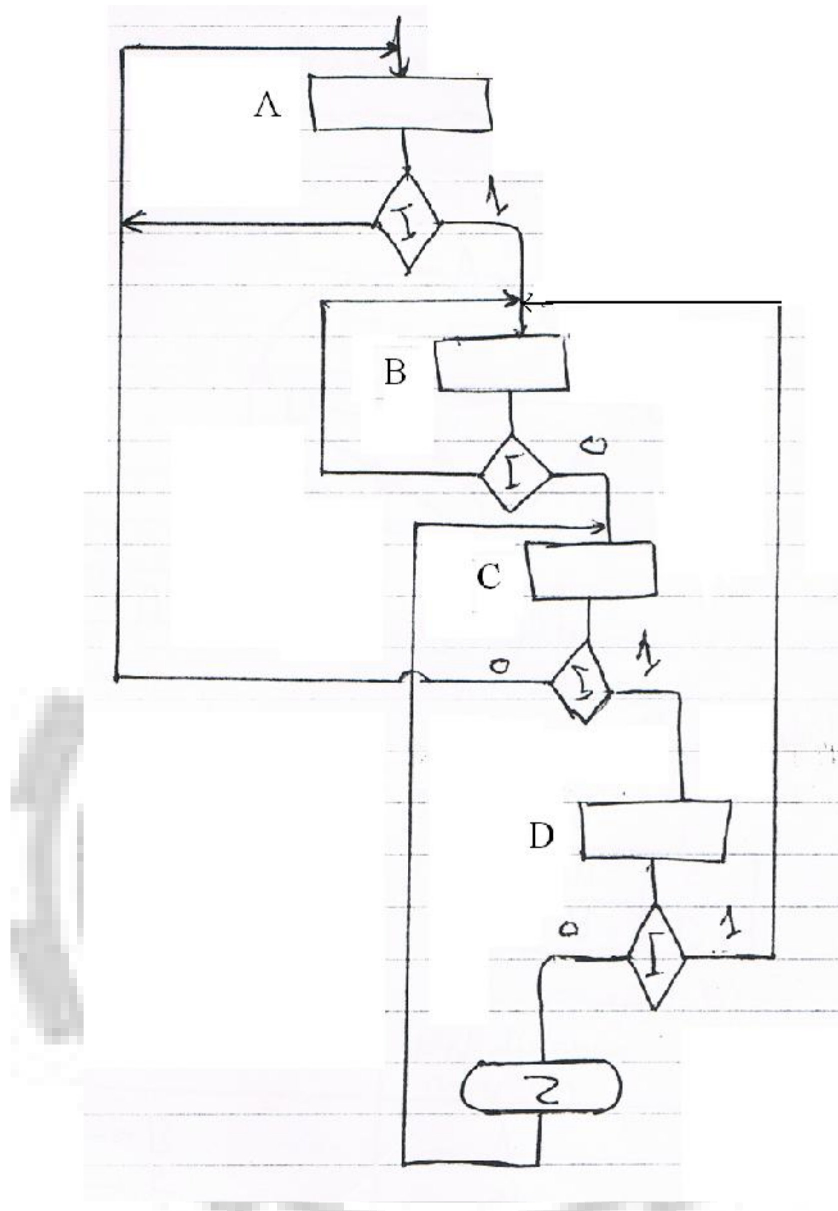
We choose the less map distance, therefore map **(a), from map (a)**

	y_1y_2
S_0	00
S_1	01
S_2	11

EX: A circuit is required to produce an output (1) when the sequence (1010) is detected and zero output at all other inputs?

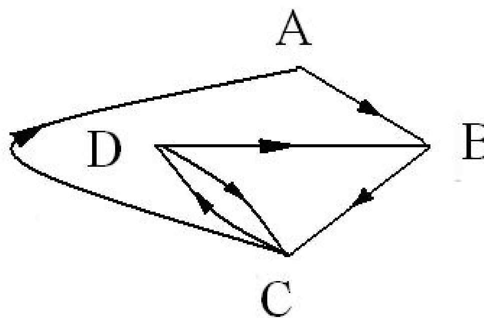


(a) State diagram



(b) ASM chart

● ***State Assignment:*** we write the transition without slink as:



Now we draw K-maps:

Map (a)

y_2 \ y_1	0	1
0	A	C
1	B	D

Map (b)

y_2 \ y_1	0	1
0	A	D
1	B	C

Map (c)

y_2 \ y_1	0	1
0	A	D
1	C	B

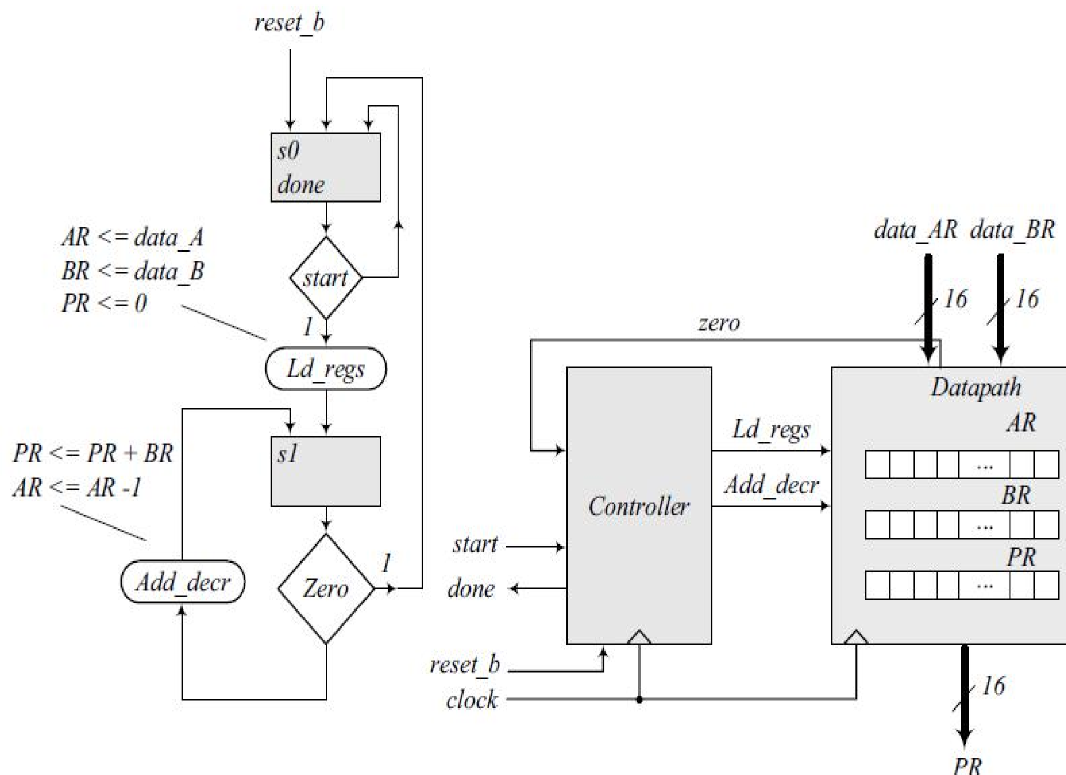
	bit distance map(a)	***bit distance map(b)***	***bit distance map(c)***
A ⟶ B	1	1	2
B ⟶ C	2	1	1
D ⟶ B	1	2	1
C ⟶ A	1	2	1
C ⟶ D	1	1	2
D ⟶ C	1	1	2
State locus	**7**	**8**	**9**

We choose the less map distance, therefore map **(a)** because it should be minimums, **from map (a):**

	y_1y_2
A	00
B	01
C	10
D	11

Ex: Develop a block diagram and an ASM chart for a digital *circuit* it multiply two binary numbers by the repeated-addition method. *For* example, *to* multiply 5 *X* 4, the digital system evaluates the product by adding the multiplication four times: 5 + 5 + 5 + 5 = 20. Design the circuit. Let the multiplicand be in register *BR.* the multiplier in register *AR.* add the product in register PR. An adder circuit adds the contents of BR to PR. A zero detection signal indicates whether AR is 0.

Sol: The internal architecture of the data path consists of a double-width register to hold the product (PR), a register to hold the multiplier (AR), a register to hold the multiplicand (BR), a double-width parallel adder, and single-width parallel adder. The single-width adder is used to implement the operation of decrementing the multiplier unit. Adding a word consisting entirely of 1s to the multiplier accomplishes the 2's complement subtraction of 1 from the multiplier. Figure below shows the ASM chart.



As another example of SM chart construction, we will design an electronic dice game. Figure (8) shows the block diagram for the dice game. Two counters are used to simulate the roll of the dice. The rules of the game are as follows:

1. After the first roll of the dice, the player wins if the sum is 7 or 11. He loses if the sum is 2, 3, or 12. Otherwise, the sum which he obtained on the first roll is referred to as his "point" and he must roll the dice again.

2. On the second or subsequent roll of the dice, he wins if the sum equals his point, and he loses if the sum is 7. Otherwise, he must roll again until he finally wins or loses.

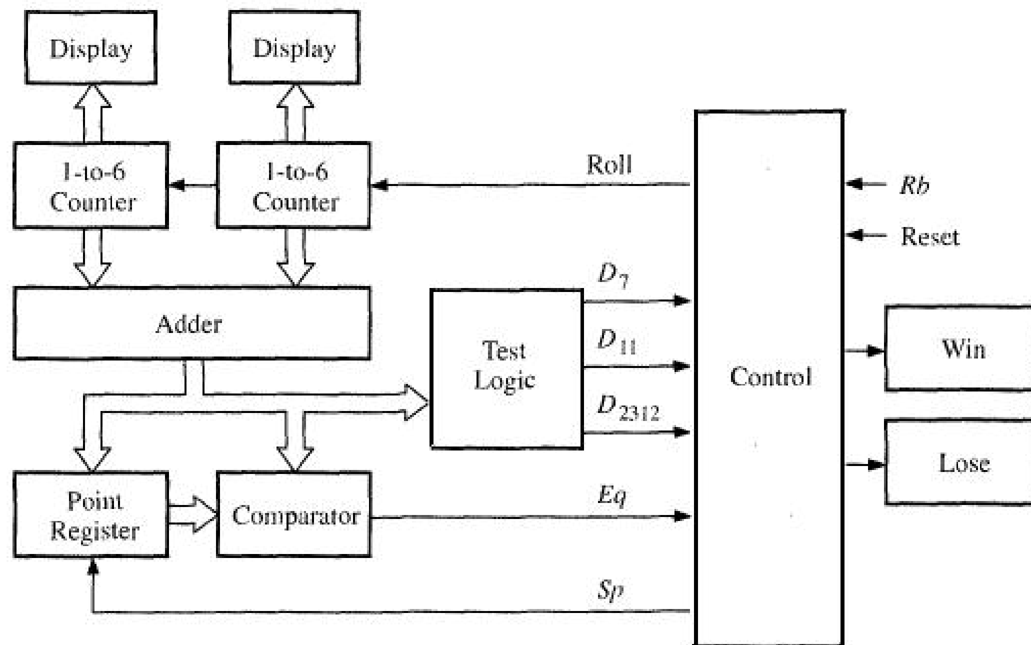


Figure (8) Block Diagram for Dice Game

The inputs to the dice game come from two push buttons, $\mathbf{R_B}$ (roll button) and ***Reset***. ***Reset*** is used to initiate a new game. When the roll button is pushed, the dice counters count at a high speed, so the values cannot be read on the display. When the roll button is released, the values in the two counters are displayed and the game can proceed. If the ***Win light*** or ***Lose light*** is not on, the player must push the roll button again.

Figure (9) shows a flowchart for the dice game. After rolling the dice, the sum is tested. If it is 7 or 11, the player wins; if it is 2, 3, or 12, he *loses*. Otherwise the sum is saved in the point register and the player rolls again. If the new sum equals the point, he *wins*; if it is 7, he *loses*. Otherwise, he *rolls* again. After winning or losing, he must push *Reset* to begin a new game.

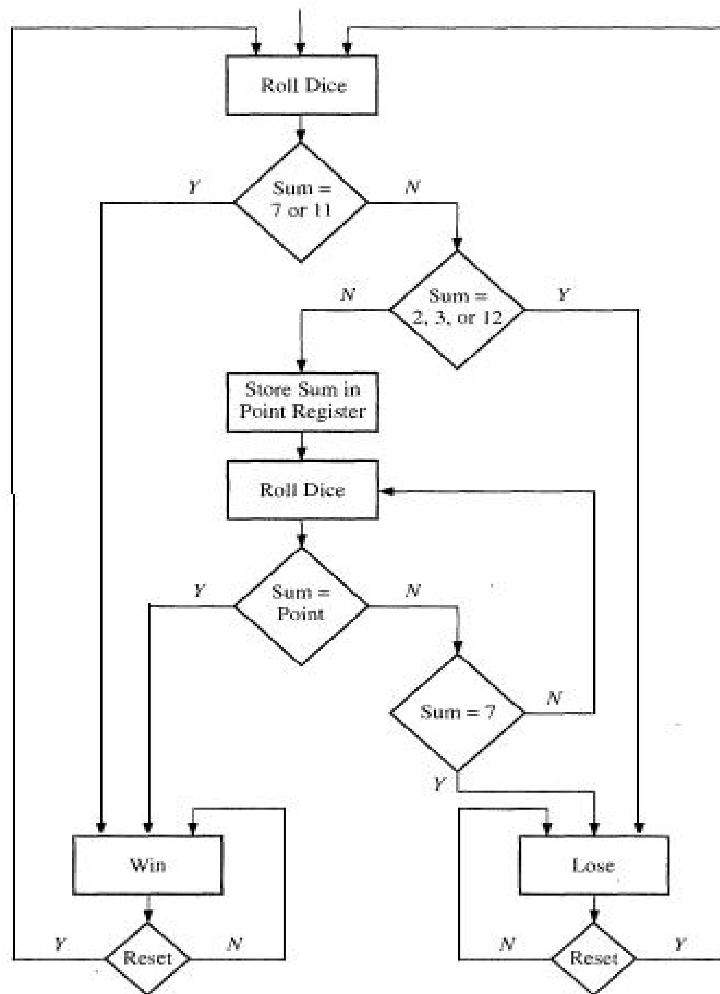


Figure (9) Flowchart for Dice Game

Input signals to the control network are defined as follows:

D_7 = 1 if the sum of the dice is 7

D_{11} = 1 if the sum of the dice is 11

$D_{2,3,12}$ = 1 if the sum of the dice is 2, 3, or 12

E_q = 1 if the sum of the dice equals the number stored in the point register

R_B = 1 when the roll button is pressed

Reset = 1 when the reset button is pressed

Outputs from the control network are defined as follows:

Roll = 1 enables the dice counters

Sp = 1 causes the sum to be stored in the point register

Win = 1 turns on the win light

Lose = 1 turns on the lose light

now convert the flowchart for the dice game to an SM chart for the control network using the control signals defined above.

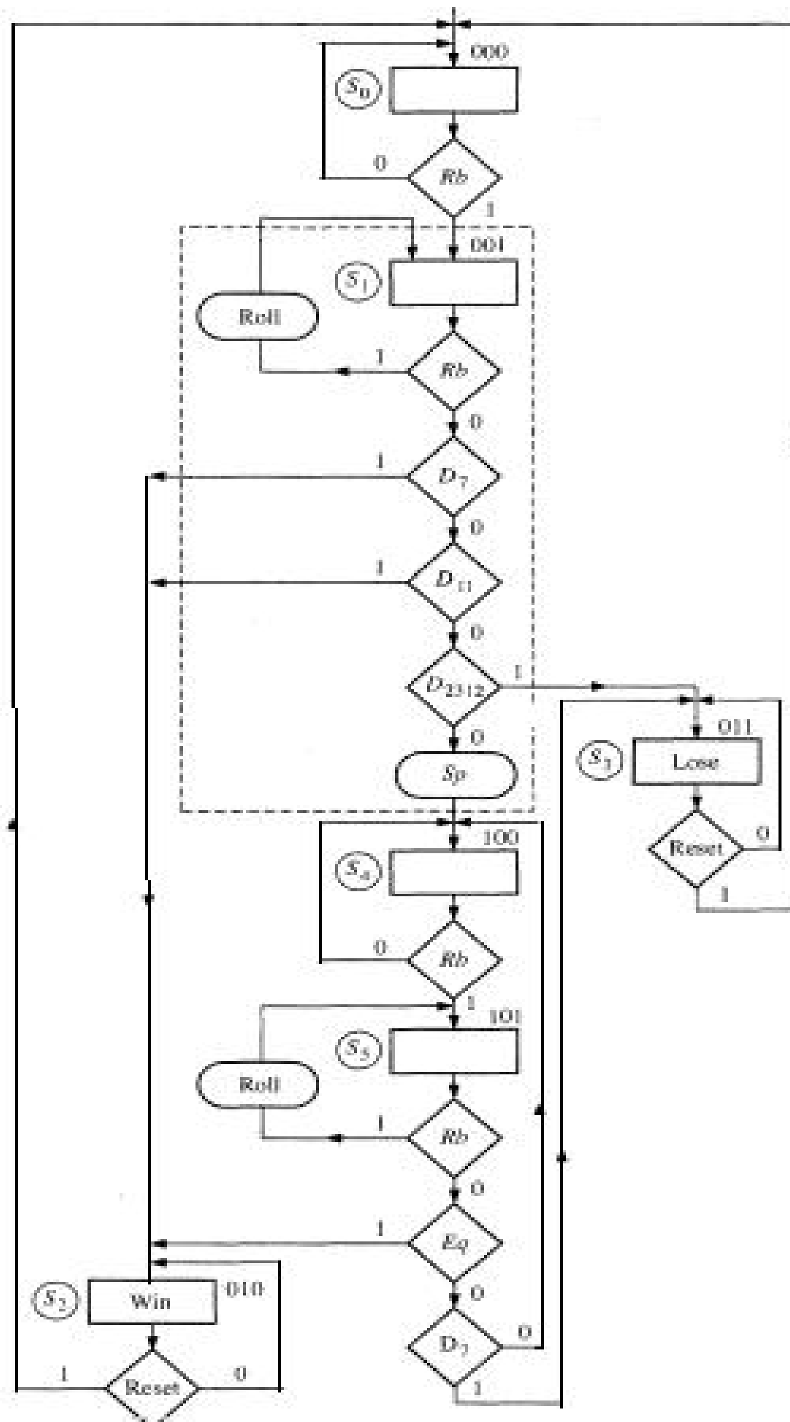
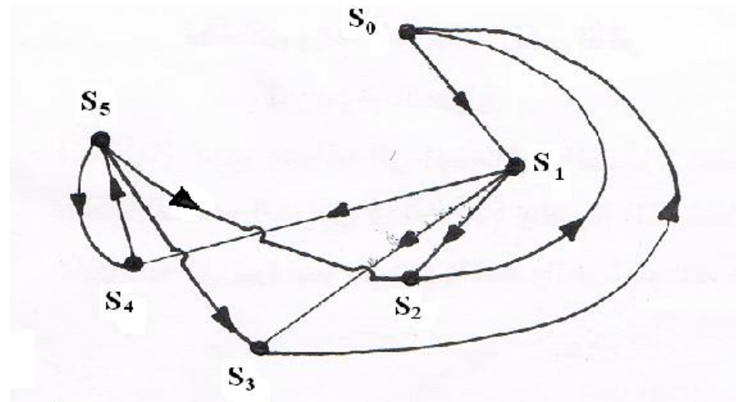


Figure ( ) SM chart for Dice game

- ***State Assignment:*** we write the transition without slink as:



Now we draw K-maps:

C \ AB	00	01	11	10
0	S_0	S_2		S_4
1	S_1	S_3		S_5

Map (a)

C \ AB	00	01	11	10
0	S_0	S_2		
1	S_1	S_3	S_5	S_4

Map (b)

C \ AB	00	01	11	10
0	S_0	S_2	S_5	S_3
1	S_1	S_4		

Map (c)

	bit distance map(a)	***bit distance map(b)***	***bit distance map(c)***
$S_0 \rightarrow S_1$	1	1	1
$S_1 \rightarrow S_2$	2	2	2
$S_1 \rightarrow S_3$	1	1	2
$S_1 \rightarrow S_4$	2	1	1
$S_4 \rightarrow S_5$	1	1	2
$S_5 \rightarrow S_2$	3	2	1
$S_5 \rightarrow S_3$	2	1	1
$S_5 \rightarrow S_4$	1	1	2
$S_2 \rightarrow S_0$	1	1	1
$S_3 \rightarrow S_0$	2	2	1
State locus	**16**	**13**	**14**

Therefore ***map (b)*** should be used for state assignment.

	ABC
S_0	000
S_1	001
S_2	010
S_3	011
S_4	101
S_5	111

P.S		ABC	Rb	Reset	D_7	D_{11}	D_{2312}	Eq	A^+	B^+	C^+	Win	Lose	Roll	Sp
S_0	1	000	0	–	–	–	–	–	0	0	0	0	0	0	0
	2	000	1	–	–	–	–	–	0	0	1	0	0	0	0
S_1	3	001	1	–	–	–	–	–	0	0	1	0	0	1	0
	4	001	0	–	1	–	–	–	0	1	0	0	0	0	0
	5	001	0	–	0	1	–	–	0	1	0	0	0	0	0
	6	001	0	–	0	0	1	–	0	1	1	0	0	0	0
	7	001	0	–	0	0	0	–	1	0	1	0	0	0	1
S_2	8	010	–	0	–	–	–	–	0	1	0	1	0	0	0
	9	010	–	1	–	–	–	–	0	0	0	1	0	0	0
S_3	10	011	–	0	–	–	–	–	0	1	1	0	1	0	0
	11	011	–	1	–	–	–	–	0	0	0	0	1	0	0
S_4	12	101	0	–	–	–	–	–	1	0	1	0	0	0	0
	13	101	1	–	–	–	–	–	1	1	1	0	0	0	0
S_5	14	111	1	–	–	–	–	–	1	1	1	0	0	1	0
	15	111	0	–	–	–	–	1	0	1	0	0	0	0	0
	16	111	0	–	0	–	–	0	1	0	1	0	0	0	0
	17	111	0	–	1	–	–	0	0	1	1	0	0	0	0

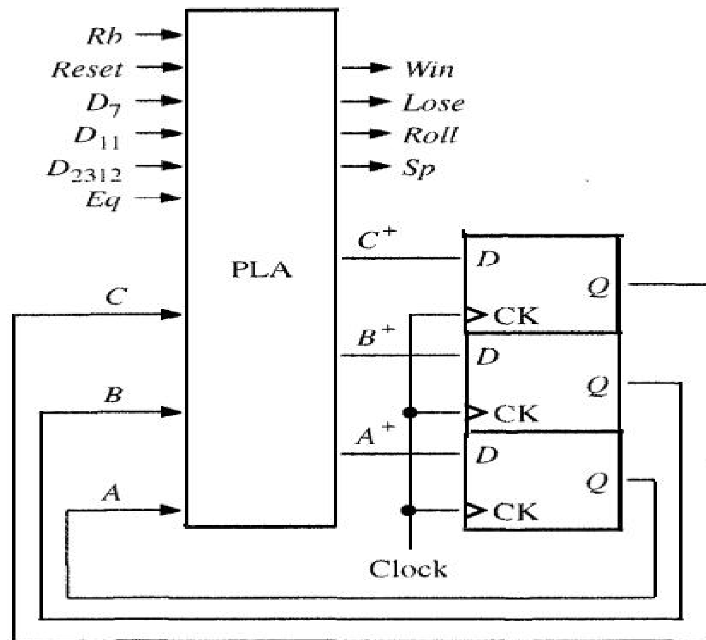
PLA Table for Dice Game

$$A^+ = \grave{A}\grave{B}C\grave{R}_B\grave{D}_7\grave{D}_{11}D_{2\,\grave{3}\,12} + A\grave{B}C\grave{R}_B + A\grave{B}CR_B + ABCR_B + ABC\grave{R}_B\grave{D}_7$$
$$= \grave{A}\grave{B}C\grave{R}_B\grave{D}_7\grave{D}_{11}D_{2\,\grave{3}\,12} + A\grave{B}C + ABCR_B + ABC\grave{R}_B\grave{D}_7$$

$$B^+ = \grave{A}\grave{B}C\grave{R}_BD_7 + \grave{A}\grave{B}C\grave{R}_B\grave{D}_7 + \grave{A}\grave{B}C\grave{R}_B\grave{D}_7\grave{D}_{11}D_{2\,3\,12} + \grave{A}B\grave{C}\grave{R}_{eset}$$
$$+ \grave{A}BC\grave{R}_{eset} + A\grave{B}CR_B + ABCR_B + ABC\grave{R}_B + ABC\grave{R}_BD_{11}$$
$$= \grave{A}\grave{B}C\grave{R}_B + \grave{A}\grave{B}C\grave{R}_B\grave{D}_7\grave{D}_{11}D_{2\,3\,12} + \grave{A}B\grave{R}_{eset} + A\grave{B}CR_B + ABC$$
$$+ ABC\grave{R}_BD_{11}$$

$$C^{+} = \grave{A}\grave{B}\grave{C}R_B + \grave{A}\grave{B}CR_B + \grave{A}\grave{B}C\grave{R}_B\grave{D}_7\grave{D}_{11}D_{2\,3\,12} + \grave{A}\grave{B}C\grave{R}_B\grave{D}_7\grave{D}_{11}\grave{D}_{2\,3\,12} + \grave{A}BC\grave{R}_{eset} + A\grave{B}C\grave{R}_B + A\grave{B}CR_B + ABCR_B + ABC\grave{R}_BD_7 + ABC\grave{R}_B\grave{D}_7 = \grave{A}\grave{B}R_B + \grave{A}\grave{B}C\grave{R}_B\grave{D}_7\grave{D}_{11}D_{2\,3\,12} + \grave{A}\grave{B}C\grave{R}_B\grave{D}_7\grave{D}_{11}\grave{D}_{2\,3\,12} + \grave{A}BC\grave{R}_{eset} + A\grave{B}C + ABC$$

$win = \grave{A}B\grave{C},\ lost = \grave{A}BC,\ Roll = \grave{A}\grave{B}CR_B + ABCR_B$,
$Sp = \grave{A}\grave{B}C\grave{R}_B\grave{D}_7\grave{D}_{11}\grave{D}_{2\,3\,12}$



PLA Realization of Dice Game Controller

A sequential logic network will be used to control the motor of a tape player. The logic network, shown as follows, will have five inputs and three outputs. Four of the inputs are the control buttons on the tape player. The input PL is 1 iff the play bu n is pressed, the input RE is 1 iff the rewind bu pressed, the input FF is 1 iff the fast forward button is pressed, and the ir ST is 1 iff the stop button is pressed. The fifth input to the control netw M, which is 1 iff the special "music sensor" detects music at the current tape position. The three outputs of the control network are P, R, and F, which make the tape play, rewind, and fast forward, respectively, when 1. No m than one output should ever be on at a time; all outputs, off cause the motor to stop.

The buttons control the tape player as follows: If the play button is pressed, the tape player will start playing the tape (output P = 1). If the play butto held down and the rewind button is pressed and released, the tape player will rewind to the beginning of the current song (output R = 1 until and then start playing. If the play button is held down and the fast forward button is pressed and released, the tape player will fast forward to the end of the current song (output F = 1 until M = 0) and then start playing. If rewir or fast forward is pressed while play is released, the tape player will rewind or fast forward the tape. Pressing the stop button at any time should stop the tape player motor.