**UNIT V**

# MEMORY and Programmable Logic

### 5.1 Introduction

The next three combinational components we will study are: ROM, PLA, and PAL. ROM's PLA's and PAL's are storage Components. This might seem like a contradition because earlier we said that combinational components don't have memory. This apparent contradiction results from an incomplete definition of the term combination component. A more precise definition is: a combinational component is a circuit that doesn't have memory of past inputs. (The outputs of a combinational component are completely determined by the current inputs.) The data in the storage components we are about to study are stored at design time before the component is added to a circuit. The data stored in sequential circuits comes from the inputs that are received while the component is active in the circuit.

The storage components that are the topic of this lecture can be used to implement Boolean functions. This may also be the most efficient implementation. If the function is moderately complex a SSI implementation (individual gates) may be expensive in terms of the number of gates that have to be purchased and in terms of the number of connections required to wire them together. When a function is implemented inside a ROM, PLA, or PAL there is only one IC to purchase and fewer connections.

### 5.2 ROM

A ROM is a combinational component for storing data. The data might be a truth table or the data might be the control words for a microprogrammed CPU. (This is a topic we will discuss later. Control words in a microprogrammed CPU interpret the macro instructions understood by the CPU.)

A ROM can be programmed at the factory or in the field.

The following image shows the generic form of a ROM:



An n x m ROM can store the truth table for m functions defined on log2n variables:



Example: Implement the following functions in a ROM:

F0 = A

F1 = A'B' + AB

Since a ROM stores the complete truth table of a function (or you could say that a ROM decodes every minterm of a function) the first step is to express each function as a truth table.

|  |  |
| --- | --- |
| **A B** | **F0 F1** |
| 0  0 | 0   1 |
| 0  1 | 0   0 |
| 1  0 | 1   0 |
| 1  1 | 1   1 |

For the discussion that follows it may be helpful to keep in mind the canonical form of the function also:

F0 = AB' + AB

F1 = A'B' + AB

We use a special notation to show the ROM implementation of a function:



The image above shows how a 4x3 ROM can be used to implement the two functions F0 and F1. (Note, there is room in the ROM for 3 functions of two variables. F2 isn't used. I'm showing a ROM with an unused portion to demonstrate that a ROM may still be the most efficient implementation even when large sections of the ROM go unused.) You can imagine a decoder inside the ROM that decodes the inputs A B. Just like the decoder we defined earlier, one output line is selected for every unique set of inputs. If the selected output line is connected to an OR gate the function associated with the OR gate will have a value of 1 for the particular set of inputs.

Note: the image above also shows a new way to indicate multiple connections with a single line. For example:



A single vertical line that intersects 4 horizontal lines represents potentially 4 different lines or connections. This is a notational convenience we will use when talking about ROM's, PLA's, and PAL's because it makes the diagrams much easier to read.

The circles at the intersection of two lines indicates a connection. Connections are either formed at the factory or in the field. If they are formed in the field a special programmable ROM is used. One type of programmable ROM is a ROM that has a fuse at every connection. Fuses at connections not wanted are burned by running high current through the fuse. What is left are the connections that define the data within the ROM.

**Observations**

* Not very efficient implementation of sparse functions.
* A ROM that implements two functions does not require twice the number of gates as a ROM that implements one function. (The decoder is shared by every output function.)

ROMs programmed at the factory are called mask ROMS because during fabrication the circuit patterns are determined by a mask. There are several different types of field programmable ROMS:

**5.3 Different Types Of Field Programmable ROMS:**

o PROM (Programmable Read-Only Memory) - This is the type that was discussed above. Connections are fused and burned in the field with a PROM programmer.

o EPROM (Erasable Programmable Read-Only Memory) - This type of ROM can be re-written by shining an ultraviolet light through a window on the IC.

o EEPROM (Electrically Erasable Programmable Read-Only Memory) - Rather than ultraviolet light an extra high voltage is used to re-write the contents of this type of ROM.

o Flash Memory - Instead of requiring extra high voltages flash memory devices work with regular device voltages.

### 5.4 Programmable Logic

A programmable logic device works like a ROM but is a more efficient solution for implementing sparse output functions. (Not all minterms are decoded.)

There are two types of programmable logic devices:

* PLA (Programmable Logic Array)
* PAL (Programmable Array Logic)

We suggested earlier that a ROM had a decoder inside it. You could visualize this as:



The image above also defines two terms we will use to distinguish between PLA and PAL devices:

* AND Array - this is the portion of the device that decodes the inputs. The AND array determines the minterms decoded by the device. A ROM decodes all possible minterms.
* OR Array - this is the portion of the device that combines the minterms for the definition of a function.

**5.5 PLA**

Now we are ready to define a PLA.

A PLA is a programmable logic device with a programmable AND array and a programmable OR array.

A PLA with n inputs has fewer than 2n AND gates (otherwise there would be no advantage over a ROM implementation of the same size). A PLA only needs to have enough AND gates to decode as many unique terms as there are in the functions it will implement.

Because we can control the AND array and there is a limit to the number of terms that can be specified in the AND array, it may be more economical to simplify the function before implementing it with a PLA. If you do simplify the function and intend to implement with a PAL device you should also keep in mind that product terms can be shared between functions. (Product sharing is when two functions share a product term decoded by the AND array. For example, in the image below the product term AB is shared between F0 and F1.)

Example: Implement the functions F0 F1 we introduced above using a PLA with 2 inputs, 3 product terms, and 2 outputs.

The unprogrammed PLA from the manufacture looks like:



After programming for the two functions F0 F1 the state of the PLA is:



Notice that we only need three AND gates because there are only three unique minterms in the functions F0 and F1. Also, notice that since we have control over the OR array we can share the minterm AB in the definitions of both functions.

Note, there may be an advantage to simplifying the functions before implementing. In the example used here there is no advantage. The simplified form of the functions F0 and F1 still require 3 unique product terms. Because product terms can be shared between functions its important to look for common product terms when simplifying.

**5.6 PAL**

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

A PAL has a fixed OR array. For example, here is what an unprogrammed PAL might look like straight from the manufacture:



A fixed OR array makes the device less expensive to manufacture. On the other hand, having a fixed OR array means you can't share product terms between functions.

Example: Implement the functions F0 F1 we introduced above using the PAL given above.

For this implementation we will need to simplify the functions F0 F1 because the PAL we are given has an output function that can accommodate only one product term. The simplified form of the functions are:

F0 = A

F1 = A'B' + AB

After programming for the two functions F0 F1 the state of the PAL is:



So, in summary:

* A PLA device has a programmable AND and programmable OR array
* A PAL device has a programmable AND and fixed OR array
* (You could also say that a ROM has a fixed AND and programmable OR array)

When implementing with a ROM there is no advantage to minimizing the functions since the input is fully decoded. When implementing with a PLA there may be an advantage to minimizing the expression but you also have to keep in mind that product terms can be shared between functions. So, when you are minimizing one function you need to consider the form of other functions and watch for product terms that can be shared. When implementing with a PAL there may also be some advantages to minimizing the function first. However, since you can't share product terms with a PAL you don't have to consider the form of other functions when minimizing.

**5.7 Programmable Logic Array (PLA)**

* A block diagram is shown in Fig. 5 – 25. It consists of *n* inputs, *m* outputs, *k* product terms, and *m* sum terms.
* PLA doesn’t provide full decoding of the variables and doesn’t generate all the minterms as in the ROM.
* The product terms constitute a group of *k* AND gates and the sum terms constitute a group of *m* or gates.
* The size of the PLA is specified by the number of inputs, the number or product terms, and the number of outputs (the number of sum terms is equal to the number of outputs.)
* A typical PLA has 16 inputs, 48 product terms, and 8 outputs.
* The number of programmed fuses is *2n x k + k x m + m* whereas that of ROM is 2*n* X *m*.
* Fig. 5 – 26 shows the internal construction of a specific PLA. It has 3 inputs, three product terms, and two outputs.

PLA program Table

* It is superior to ROM for circuits that have a large number of don’t-care conditions.
* Ex: Consider the truth table of the combinational circuit, shown in Fig. 5 – 27 (a).
* The simplified functions in sum of products are obtained from the maps of Fig. 5 – 27 (b).
* There are 3 distinct terms in this combinational circuit: *AB’*, *AC*, and *BC*.
* The circuit has three inputs and two outputs; so the PLA of Fig. 5 – 26 can be used to implement this combinational circuit.
* The PLA program table is shown in Fig. 5 – 27 (c).
* The first column lists the product terms numerically.
* The second column specifies the required paths between inputs and AND gates.
* The third column specifies the paths between the AND gates and the OR gates.
* Under each output variable, we write a *T* (for true) if the output inverter is to be bypassed, and *C* (for complement) if the function is to be complemented with the output inverter.

Ex: A combinational circuit is defined by the functions

 *F1 (A, B, C) = (3, 5, 6, 7)*

 *F2 (A, B, C) = (0, 2, 4, 7)*

Implement the circuits with PLA having three inputs, four product terms, and two outputs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 0 |  |  | 1 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 |  | 1 | 1 | 1 |

 *F1 = AC + AB + BC*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 0 | 1 |  |  | 1 |
| 1 | 1 |  | 1 |  |

 *F2 = B’C’ + A’C’ + ABC*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 0 | 0 | 0 |  | 0 |
| 1 | 0 |  |  |  |

 *F’1 = B’C’ + A’C’ + A’B’*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 0 |  | 0 | 0 |  |
| 1 |  | 0 |  | 0 |

 *F’2 = B’C + A’C + ABC*

PLA program Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Product Term | Inputs A B C | Outputs F1 F2 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *B’C’* | 1 | - | 0 | 0 | 1 | 0 |  |
| *A’C’* | 2 | 0 | - | 0 | 1 | 0 |  |
| *A’B’* | 3 | 0 | 0 | - | 1 | - |  |
| *ABC* | 4 | 1 | 1 | 1 | - | 1 |  |
|  |  |  |  |  | C | T | T/C |

**5.8 Programmable Array Logic (PAL)**

* Fig. 5 – 29 shows the conventional and array logic symbols for a multiple-input AND gate.
* 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, but is not flexible as the PLA.
* Figure 5-30 shows the logic array configuration of a typical PAL.
* It has 4 inputs and 4 outputs. Each has a buffer and an inverter gate.
* There are 4 sections in the unit, each being composed of a three wide AND-OR array.
* This is the term used to indicate that there are three programmable AND gates in each section and one fixed OR gate.
* Each AND gate has 10 fused programmable inputs.
* Unlike the 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.
* The number of product terms in each section is fixed, and if the number of the terms in the function is too large, it may be necessary to use two sections to implement one Boolean function.
* Ex: Using a PAL in the design of a combinational circuit, consider the following Boolean functions given in sum of minterms:

*w(A, B, C, D) = (2, 12, 13)*

*x (A, B, C, D) = (7, 8, 9, 10, 11, 12, 13, 14, 15)*

*y (A, B, C, D) = (0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)*

*z (A, B, C, D) = (1, 2, 8, 12, 13)*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 00 | 01 | C11 | C10 |  |
| 00 |  |  |  | 1 |  |
| 01 |  |  |  |  | B |
| A 11 | 1 | 1 |  |  | B |
| A 10 |  |  |  |  |  |
|  |  | D | D |  |  |

*w = ABC’ + A’B’CD’*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 00 | 01 | C11 | C10 |  |
| 00 |  |  |  |  |  |
| 01 |  |  | 1 |  | B |
| A 11 | 1 | 1 | 1 | 1 | B |
| A 10 | 1 | 1 | 1 | 1 |  |
|  |  | D | D |  |  |

*x = A + BCD*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 00 | 01 | C11 | C10 |  |
| 00 | 1 |  | 1 | 1 |  |
| 01 | 1 | 1 | 1 | 1 | B |
| A 11 |  |  | 1 |  | B |
| A 10 | 1 |  | 1 | 1 |  |
|  |  | D | D |  |  |

*y = A’B + CD + B’D’*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 00 | 01 | C11 | C10 |  |
| 00 |  | 1 |  | 1 |  |
| 01 |  |  |  |  | B |
| A 11 | 1 | 1 |  |  | B |
| A 10 | 1 |  |  |  |  |
|  |  | D | D |  |  |

*z = ABC’ + A’B’CD’ + AC’D’ + A’B’C’D’*

 *= w + AC’D’ + A’B’C’D*

* The table below shows the PAL programmable table for the 4 Boolean functions.

**PAL Programmable Table**

*Product AND Inputs*

*Term A B C D W Outputs*

*1 1 1 0 - - w = ABC’*

*2 0 0 1 0 - + A’B’CD’*

*3 - - - - -*

*4 1 - - - - x = A*

*5 - 1 1 1 - + BCD*

*6 - - - - -*

*7 0 1 - - - y = A’B’*

*8 - - 1 1 - + CD*

*9 - 0 - 0 - + B’D’*

*10 - - - - 1 z = w*

*11 1 - 0 0 - + AC’D’*

*12 0 0 0 1 - + A’B’C’D*

The table is divided into 4 sections with three products in each.

* The first two sections need only two product terms to implement the Boolean function.
* The last two sections need three product terms. (Function *w* was reduced to 3 terms.)
* The fuse map is shown in Fig. 5 – 31.
* For each 1 or 0 in the table, we mark the corresponding intersection in the diagram with the symbol for an intact fuse.
* For each dash, we mark the diagram with blown fuses in both the true and complement inputs.

If the AND gates is not used, we leave its

**5.9 DIGITAL LOGIC CIRCUITS**

Digital Logic Families

 **Resistor-Transistor Logic (RTL) family**



The resistor-transistor logic, also termed as RTL, was most popular kind of logic before the invention of IC fabrication technologies.As its name suggests, RTL circuits mainly consists of resistors and transistors that comprises RTL devices. The basic RTL device is a NOR gate, shown in figure aside. Inputs to the NOR gate shown above are ‘input1’ & ‘input2’. The inputs applied at these terminals represent either logic level HIGH (1) or LOW (0).
The logic level LOW is the voltage that drives corresponding transistor in cut-off region, while logic level HIGH drives it into saturation region. If both the inputs are LOW, then both the transistors are in cut-off i.e. they are turned-off. Thus, voltage

Vcc appears at output I.e. HIGH. If either transistor or both of them are applied HIGH input, the voltage Vcc drops across Rc and output is LOW.  RTL family is characterized by poor noise margin, poor fan out capability, low speed and high power dissipation. Due to these undesirable characteristics, this family is now obsolete.

**5.10 Diode-Transistor Logic (DTL) family**



The diode-transistor logic, also termed as DTL, replaced RTL family because of greater fan-out capability and more noise margin. As its name suggests, DTL circuits mainly consists of diodes and transistors that comprises DTL devices. The basic DTL device is a NAND gate, shown aside. Three inputs to the gate are applied through three diodes viz. D1, D2 and D3. The diode will conduct only when corresponding input is LOW. If any of the diode is conducting i.e. when at least one input is LOW, the voltage at cathode of didoe DA is such that it keeps transistor T in cut-off and subsequently,output of transistor is HIGH.
If all inputs are HIGH, all diodes are non-conducting, transistor T is in saturation, and its output is LOW. Due to number of diodes used in this circuit, the speed of the circuit is significantly low. Hence this family of logic gates is modified to transistor-transistor logic i.e. TTL family.

**5.11 Transistor-Transistor Logic (TTL) family**



TTL family is a modification to the DTL. It has come to existence so as to overcome the speed limitations of DTL family. The basic gate of this family is TTL NAND gate.
Modifications to DTL NAND

1.The diodes D1, D2 and D3 are replaced by emitter-base junctions of a multiple-emitter transis labeled T1.

2. Diode DA is replaced by collector-base junction of T1. Diode DB is replaced by emitter base junction of transistor labeled T2.
The working of this circuit is identical to that of DTL circuit.

Case1- When at least one input is logic LOW, transistor T2 and T3 are in cut-off and hence, output of T3 is HIGH.

Case2 When all inputs are HIGH, T1 operates in active inverse mode, driving T2 & T3 in saturation. Since T3 is ON, the output is LOW.

 Case3- While all inputs are HIGH, if any of the inputs suddenly goes LOW, then T2 and T3 will be turned off only when stored base charge is removed. The collector-base junction of T1 is back-biased and T1 operates in normal active region. A large collector current of T1 is in such direction that it helps removing base charge of T2 and T3. In this way, the circuit speed is increased in TTL over speed of DTL.

**5.12 Emitter Coupled Logic (ECL) Family**



ECL logic family implements the gates in differential amplifier configuration in which transistors are never driven in the saturation region thereby improving the speed of circuit to a great extent. The ECL family is fastest of all logic families.
The basic gate of ECL family is NOR gate (OR and NOR together) as shown in diagram. The output1 is OR output while ouput2 is NOR output.

Transistor T1 is applied with input and additional inputs are applied to transistors (T1’, T1’’, .. .) in parallel with T1. Thus transistor(s) T1 and T2 are connected in differential amplifier configuration.
Transistors T3 and T4 are emitter-followers used for DC level-shifting of output voltages.
The positive supply terminal of the circuit is grounded while negative supply terminal is at negative 5.2V. This is done to minimize the effect of noise introduced by the power supply and also to protect the gate from short-circuit that might occur accidently.
Both the outputs (HIGH/LOW) for OR and NOR are negative. Thus, to interface this logic family with other, a translator circuit is needed which converts negative voltages to compatible positive voltage levels.

**5.13 MOS Logic Family**



MOS logic family implements the logic gates using MOSFET devices. MOSFETs are high density devices which can easily and economically fabricated on ICs. MOS logic gates can be fabricated using either only NMOS or only PMOS devices.

MOS logic is vastly used in LSI and VLSI devices, such as microprocessor chips, due to their high density characteristic. NMOS NOR gate is shown in figure. If both transistors T1 and T2 are off i.e. A = B = LOW, then output is HIGH = VDD. If either of the inputs is HIGH, then corresponding transistor(s) is/are ON, thus connecting output to GND i.e. LOW.



If both inputs are ON, then only both T1 and T2 are ON and output is LOW; otherwise (when either or both transistors are OFF,) the output is HIGH. This is NAND operation on applied inputs.

!!Caution:

            If any of input of MOS is left unconnected, the open input terminal which has very high input impedance may take any stray electric charge as input and can develop extreme dangerous high voltage which can damage the device and may also harm the person handling the device. Thus, none of the un-used inputs of MOS device should be left unconnected. It must connected to ground or supply. Even for storage of device, all pins must be shorted.

CMOS Logic family

**5.14 CMOS NAND Gate**

CMOS stands for complementary-MOS, in which both p-channel and n-channel enhancement MOSFET devices are fabricated on same chip. This causes density to be reduced and complex fabrication process. However, CMOS devices consume negligible power and hence are preferred over MOS devices in battery operated applications.



A CMOS NAND gate is shown in figure aside.
T1 and T2 are n-channel MOSFETs while T3 and T4 are p-channel MOSFETs.
When both inputs A & B are HIGH, then T1 & T2 are ON while T3 & T4 are OFF. Hence,output is connected to GND i.e. LOW.
If either input is LOW, then either T3 or T4 is ON, connecting output is +Vcc i.e. HIGH.



Similar is working of CMOS NOR gate shown in figure aside. Here, p-channel devices are in series and n-channel devices are in parallel.

**5.15 Comparisons of Logic Families**



Major parameters for comparing logic families are the propagation delay, the power dissipation and the speed-power product. ‹