

# PLD Concepts and Types

# Building Logic Circuits

To build logic circuits with individual gates, flip flops, and other circuits, the Boolean logic equations are first written for the desired functions. Minimization techniques are applied to simplify the circuitry.

Then individual ICs are wired together on a printed circuit board to create the desired operation.

If circuit additions, changes, or modifications were required, a completely new printed circuit board had to be designed. This was a major time consuming and very expensive process.

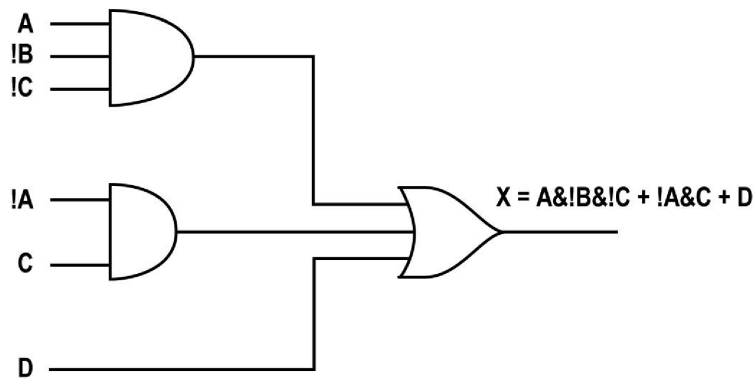
# The PLD Idea

The idea behind a PLD is to produce an IC that contains individual gates and flip flops and other circuits that can be programmed on a chip. That is, the individual circuits on the chip can be enabled or disabled, connected or disconnected, or turned off or on to create the desired Boolean function.

When some PLDs are programmed, they are permanently configured. If any changes are needed, a new chip must be programmed. These are referred to as one-time-programmable (OTP). Other PLD types may be reprogrammed to implement the changes or correct errors.

# Basic PLD Architecture

## SUM OF PRODUCTS (SOP)



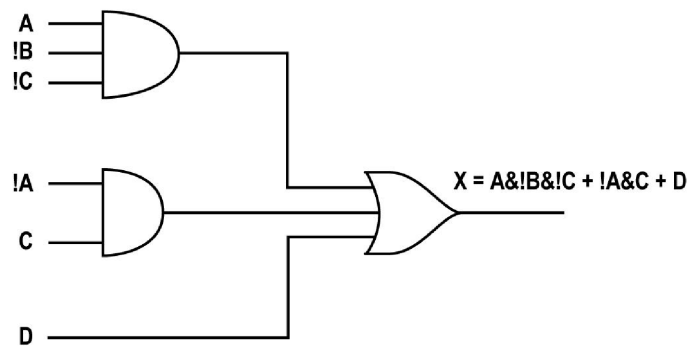
NOTE:    ! means NOT so !A means NOT A or the compliment of A  
          & means the AND logic function  
          + means the OR logic function

Most PLDs are made up of multiple gates, flip flops, and other logic circuits on a single chip with some mechanism for configuring them into a desired circuit.

PLDs are organized as arrays of AND and OR gates connected in a sum-of-products (SOP) format.

# Basic PLD Architecture: SOP Format

## SUM OF PRODUCTS (SOP)



NOTE: ! means NOT so !A means NOT A or the compliment of A  
& means the AND logic function  
+ means the OR logic function

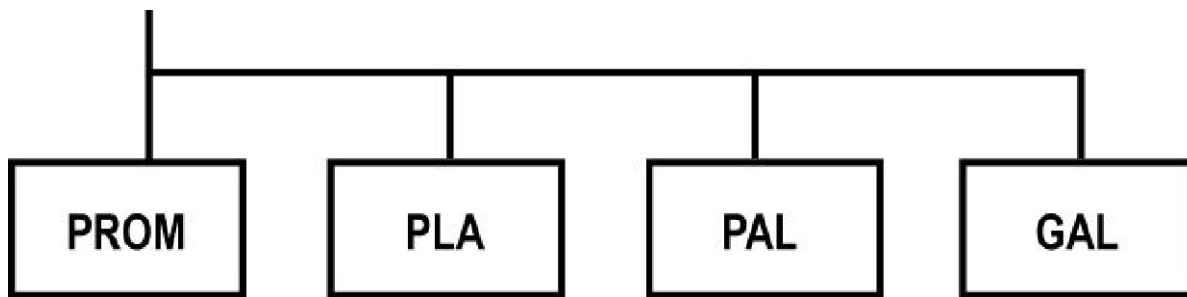
The SOP format, shown in the above example, means that input signals are connected to AND gates that in turn drive OR gates that produce the outputs. The AND array, the OR array or both arrays can be made programmable.

# Four Types of SPLDs

Some forms of PLDs also contain flip flops to create storage registers, counters, or shift registers. Most also contain inverters and three-state drivers as outputs.

The more advanced PLDs contain RAM or programmable ROM.

The four basic types of simple PLDs are programmable read-only memory (PROM), programmable logic array (PLA), programmable array logic (PAL), and generic logic array (GAL).



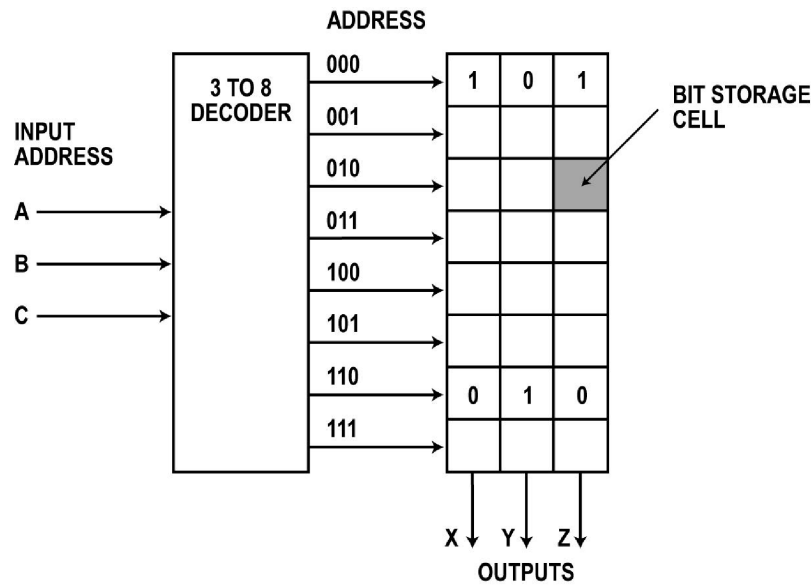
# Programmable Read-Only Memory (PROM)

Programmable read-only memory (PROM) is a type of semiconductor memory in which data is permanently stored or programmed into it. Even if power is removed from the circuit, the data remains.

Such a memory is said to be non-volatile. The data is stored as multibit words. There can be one bit or many bits in each word.

Each word in the ROM is accessed by applying an address input. When the address is applied, the stored data is outputted.

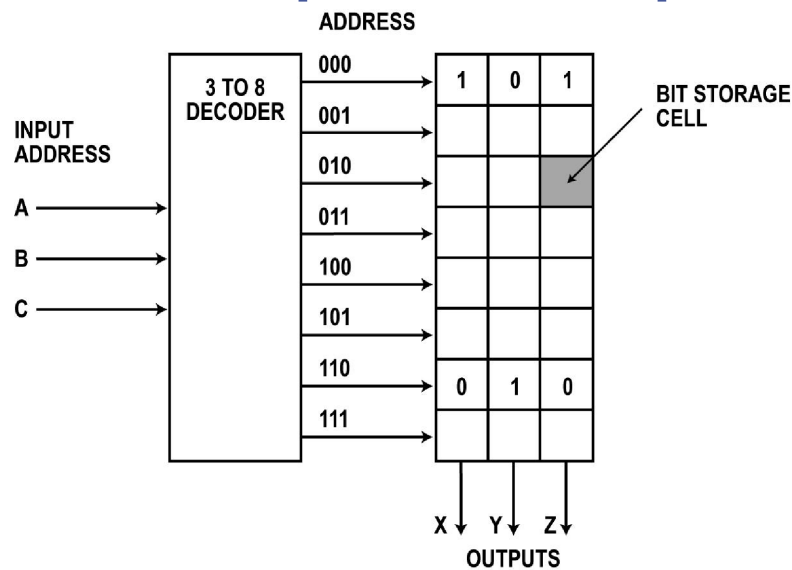
# PROM Example



The example here shows a simple ROM with eight memory data locations. Each location has three memory cells where three bits may be stored. A three bit address is applied to the input where a 3 to 8 decoder is used to select one of the memory locations. When the address 000 is applied, the PROM output is 101. When the address 110 is applied the output is 010.



# PROM Example: Look-up Table

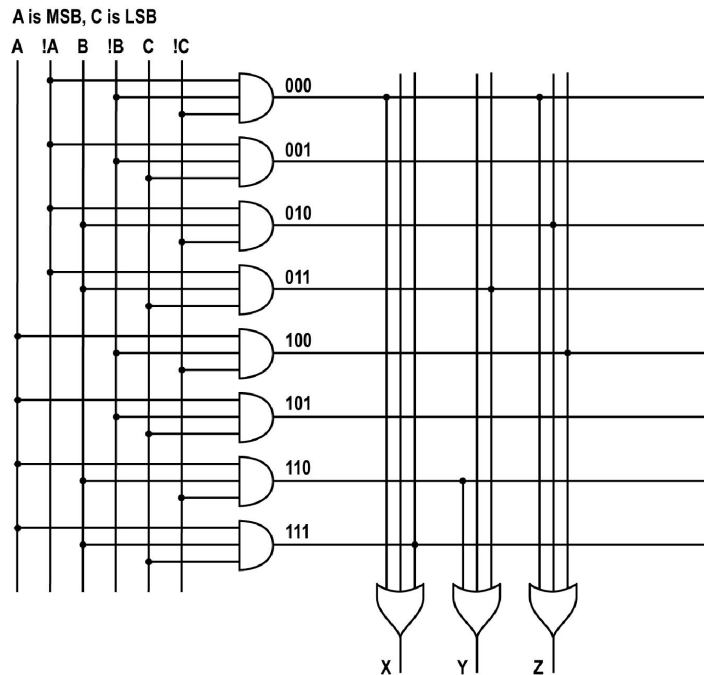


This type of PLD is equivalent to implementing logic with a look-up table (LUT).

The decoder address bits are the logic inputs. The decoder outputs enable one of many memory locations where binary words are stored.

In each memory location is that defines the desired outputs that should occur if the particular input code appears.

# Expanded View

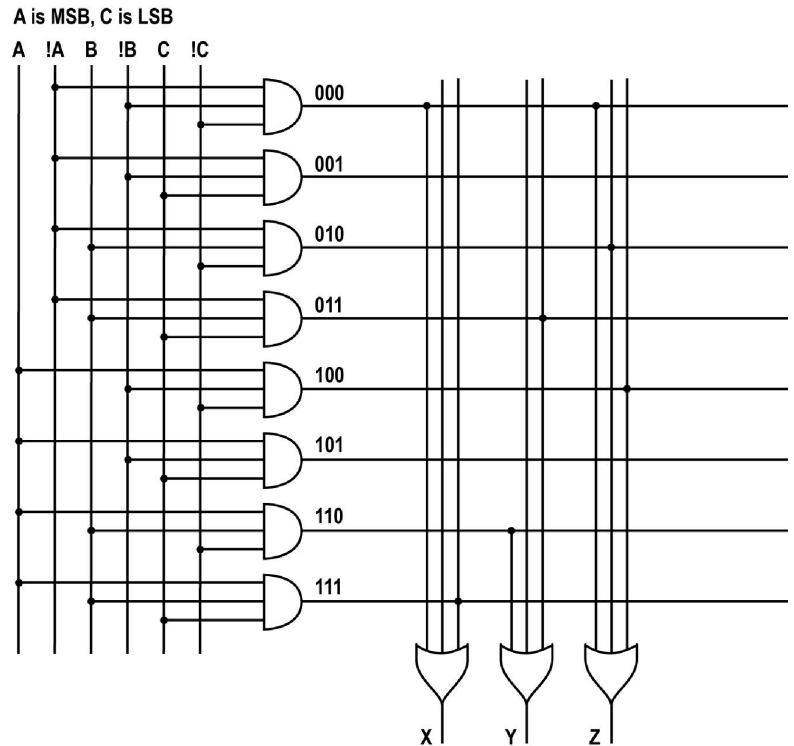


The PROM was the earliest form of a simple PLD. It is essentially a fixed AND input array (the address decoder) followed by a programmable OR output array. The figure shown here is an expanded version of the figure shown on the previous slide.

# PROM Inputs and Outputs

In a PROM, all input states are fully decoded by a fixed AND array.

The outputs of the AND decoder gates are then connected to the OR gates by programmable fuse type connections. Note that the AND outputs and OR inputs cross one another. A dot at the cross over indicates a connection, no dot means no connection. The OR gates provide the outputs.



# PROM Example

This circuit implements the following Boolean equations:

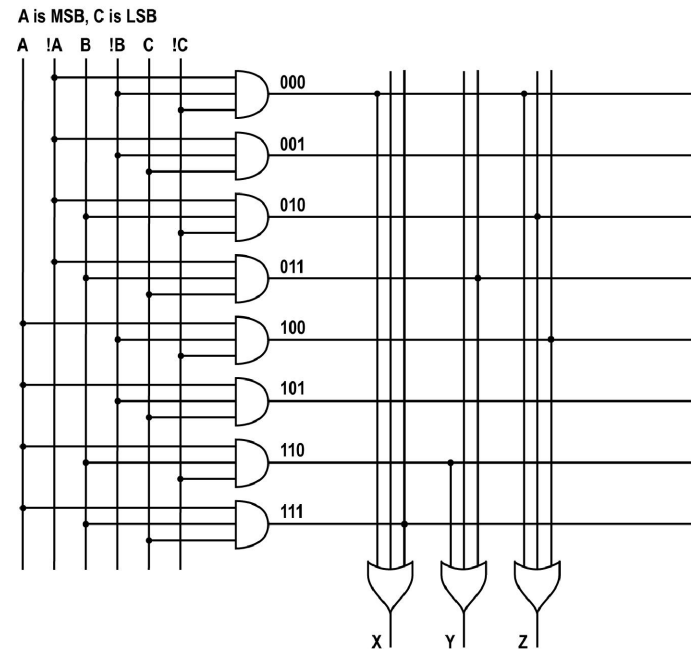
The following notation is used to express the logic: ! Means NOT, & is AND, and + is OR.

$$X = (!A \& !B \& !C) + (A \& B \& C)$$

$$Y = (!A \& B \& C) + (A \& B \& !C)$$

$$Z = (!A \& !B \& !C) + (!A \& B \& !C) + (A \& !B \& !C)$$

Study the figure to be sure you see how the AND outputs are connected to the OR inputs to produce each equation.



# Truth Table

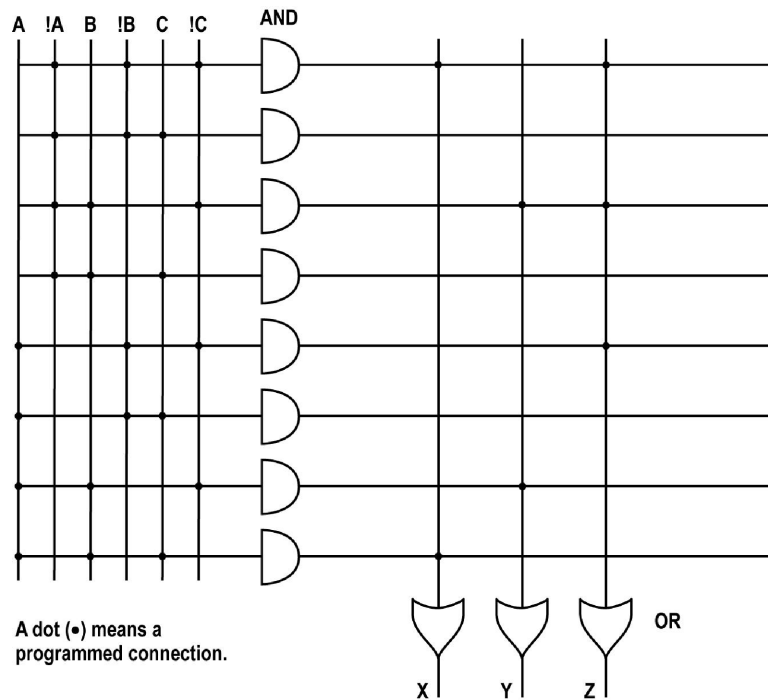
This circuit from the previous slide produces the truth table shown here.

This illustrates how logic can be designed with truth tables defining the operation rather than Boolean equations.

INPUTS A B C	OUTPUTS X Y Z
0 0 0	1 0 1
0 0 1	0 0 0
0 1 0	0 0 1
0 1 1	0 1 0
1 0 0	0 0 1
1 0 1	0 0 0
1 1 0	0 1 0
1 1 1	1 0 0

Truth table for PROM

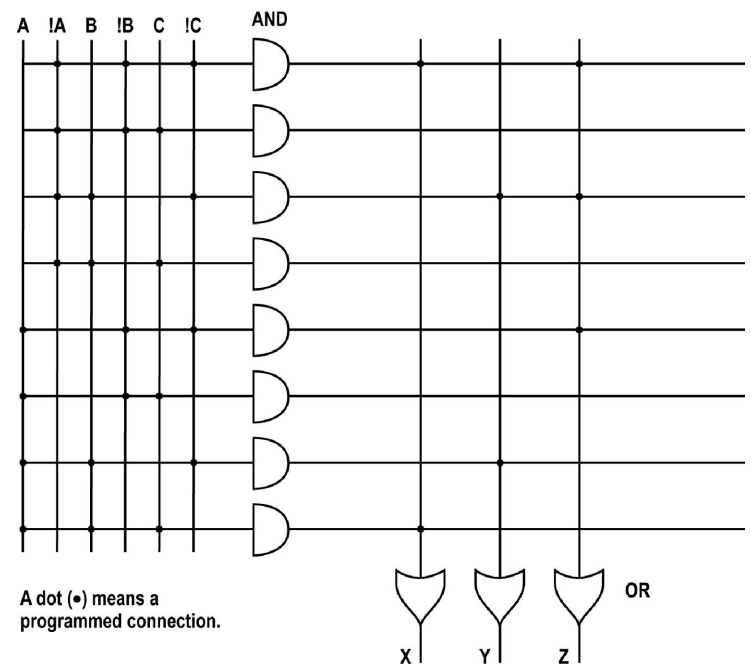
# Simplifying the Drawing



The earlier figure is a complex drawing even for simple logic functions like that shown. To simplify the drawings, a short-hand notation is used. The figure shown here is the same drawing as shown earlier but with the short-hand format.

# Simplifying the Drawing

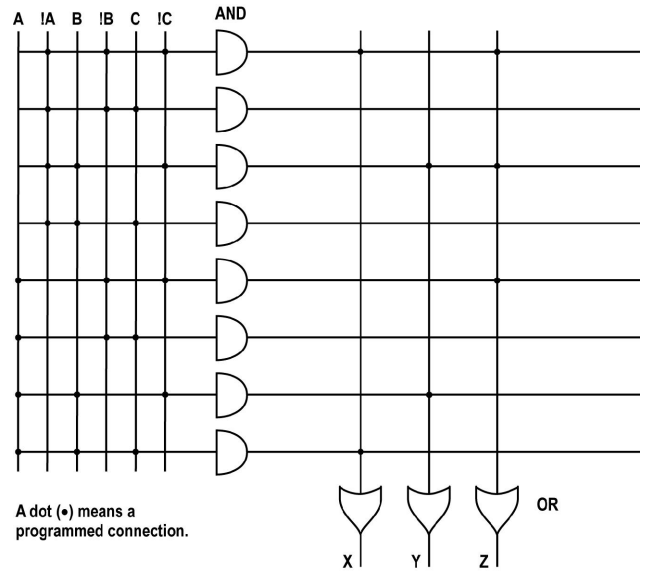
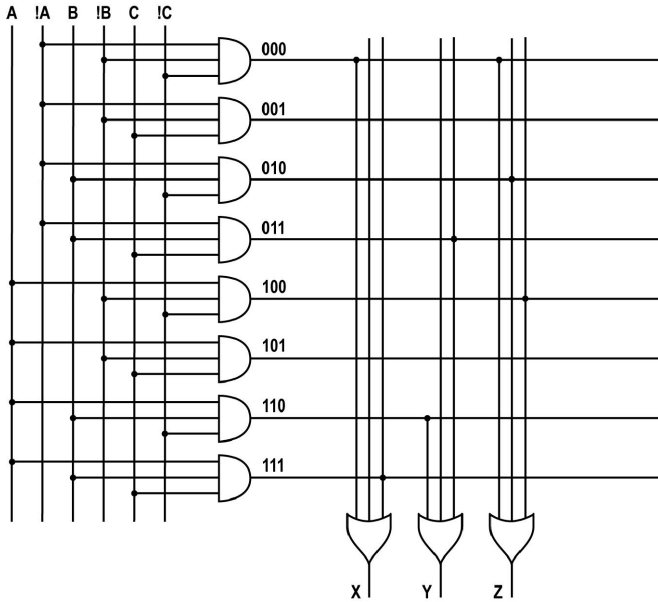
Notice that each AND gate shows only one input. This line represents all of the inputs and a programming dot is placed at the connection of each input. It does NOT mean that all inputs are connected together. It is simply a way of reducing the number of lines on the drawing. The OR gates also have only one input representing two, three or more actual inputs. Programming dots show the individual connections from AND outputs to the OR gate inputs.



# Comparing the Drawings

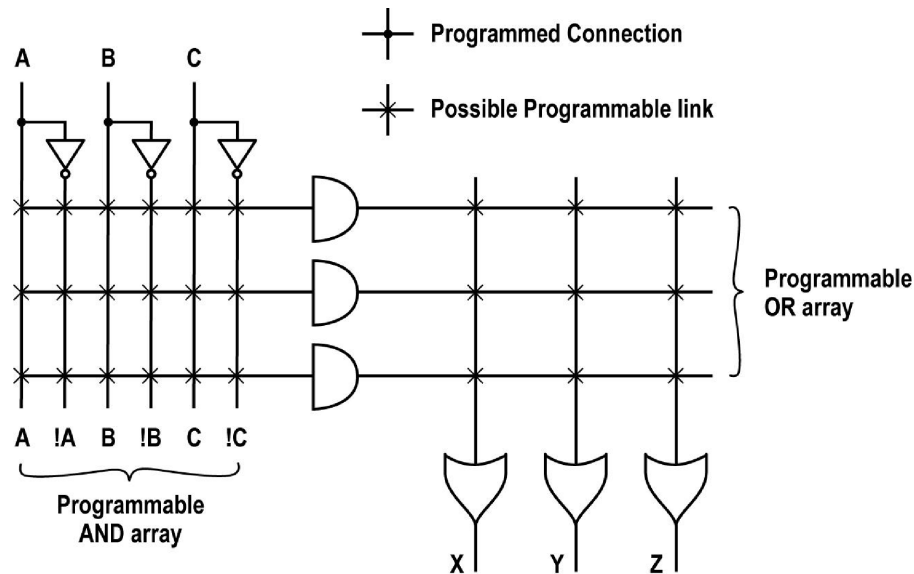
Compare the two figures to be sure you understand the notation.

A is MSB, C is LSB





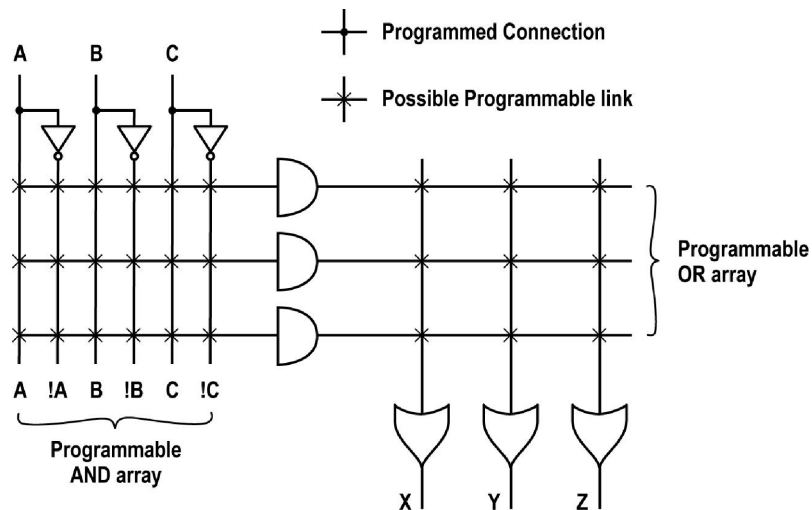
# Programmable Logic Array (PLA)



A programmable logic array (PLA) is a form of PLD in which both the AND and OR gate arrays are programmable. PLAs provide the utmost in programmability and flexibility implementing a desired logic function.

The figure shows an example of a PLA where connections to both AND and OR inputs are programmable.

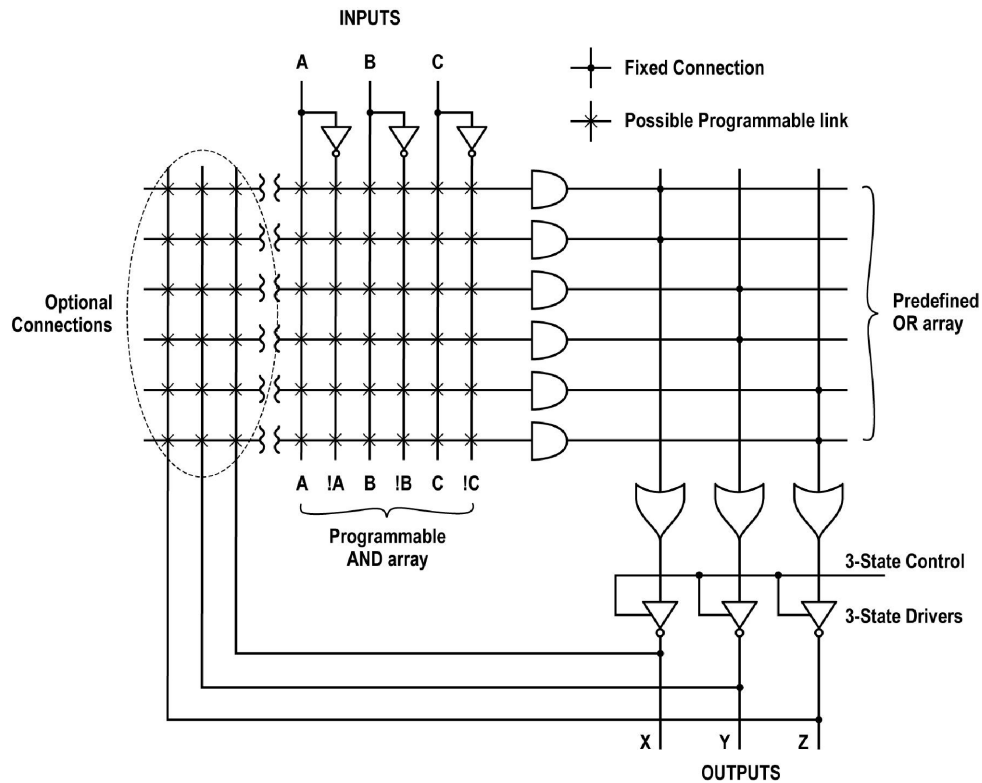
# Disadvantage of PLAs



PLAs have been less popular because the extra programming connections introduce additional propagation delays that slow down the logic. This problem led to the development of the programmable array logic (PAL).

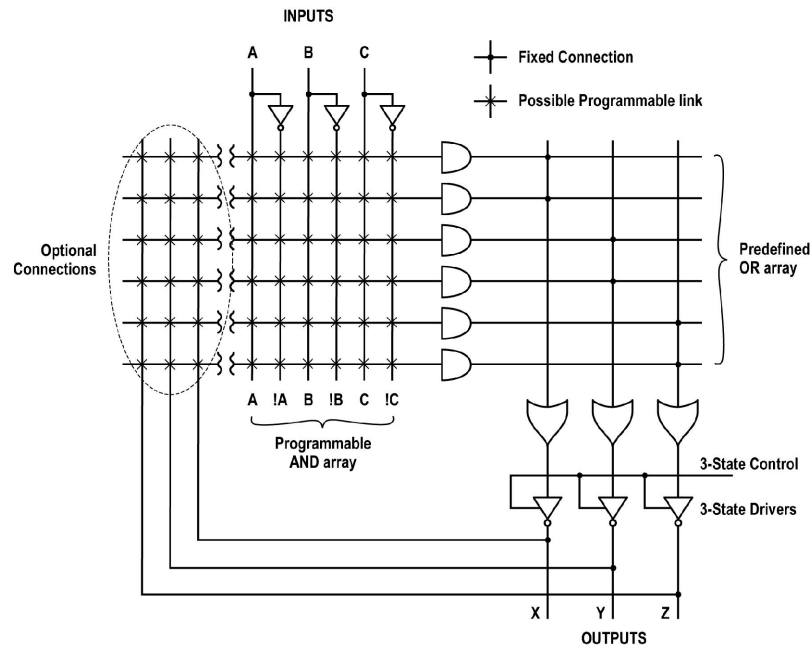
With two programmable interconnection sections, PLAs were never able to compete with the PROM or PAL for speed.

# Programmable Array Logic (PAL)



A programmable array logic (PAL) is a type of PLD in which the AND array is programmable and the OR array is fixed.

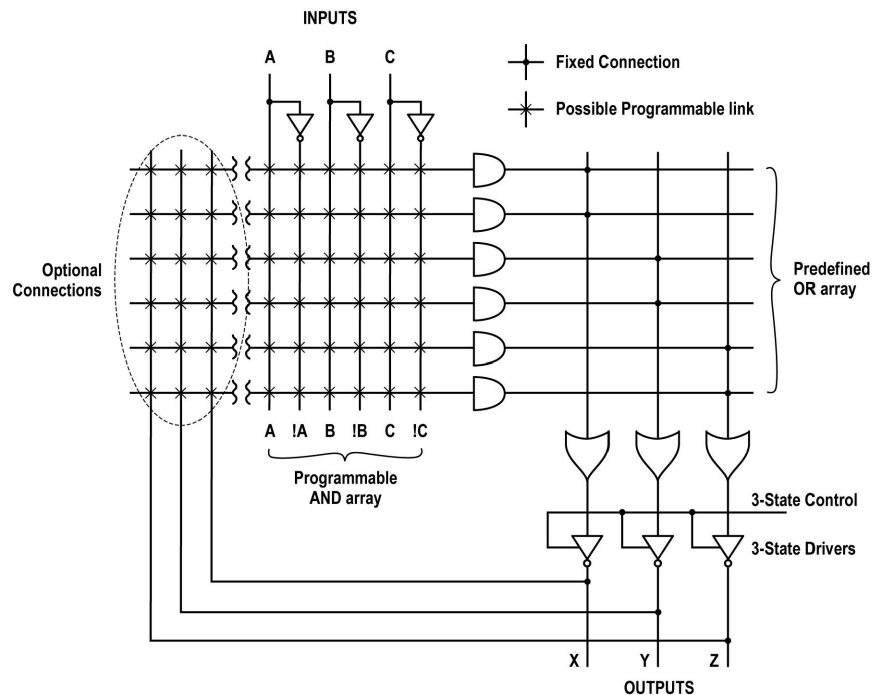
# PAL Inputs and Outputs



The inputs are applied to inverters so that both the inputs and their complements are available for programming. The inputs then go to AND gates as dictated by the Boolean equations.

The AND gates are already prewired to the OR gates which form the outputs.

# Connections



Some PALs also have 3-state output drivers so that the outputs can be used in bus connections.

In some designs, the outputs can also be used as inputs as the optional connections shown here.

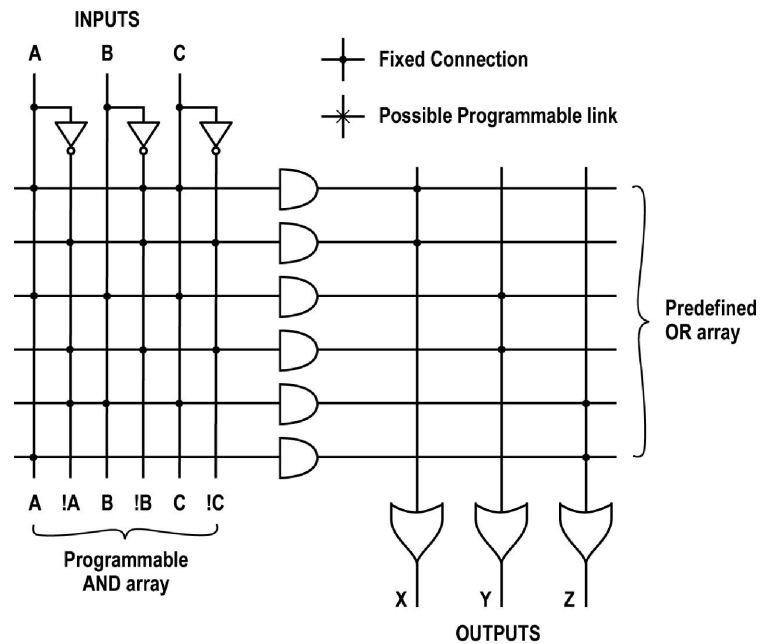
# PAL Example

This example shows how a PAL would be programmed to implement a specific example of Boolean expressions.

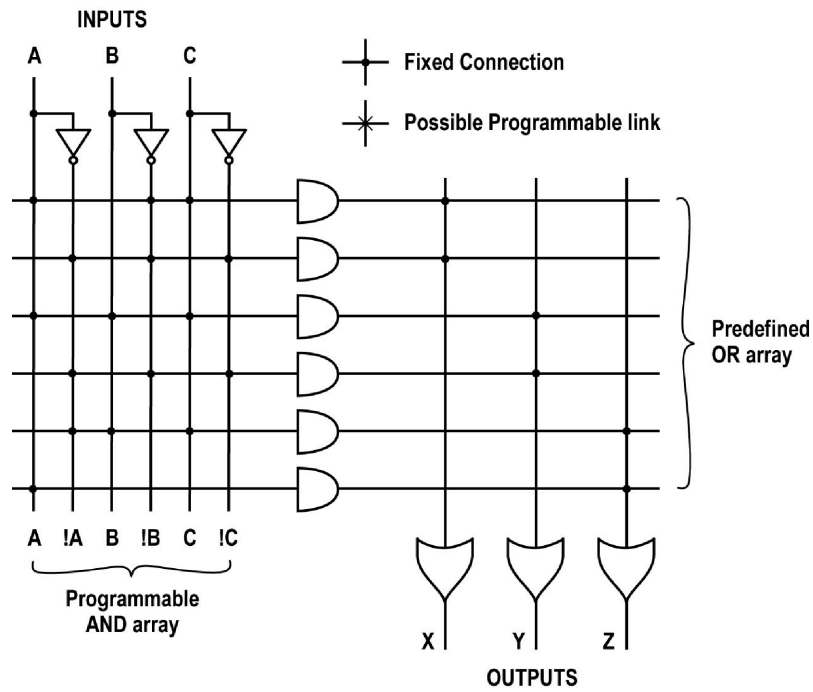
Remember in these expressions, the terms with an exclamation point (!) mean the NOT or inverted version of that term. NOT A is written as !A. For example, if  $A = 0$  then  $!A = 1$  and vice versa.

Looking at this figure, write the equations for X, Y and Z.

See the next frame for the answers.



# Answers to PAL Example

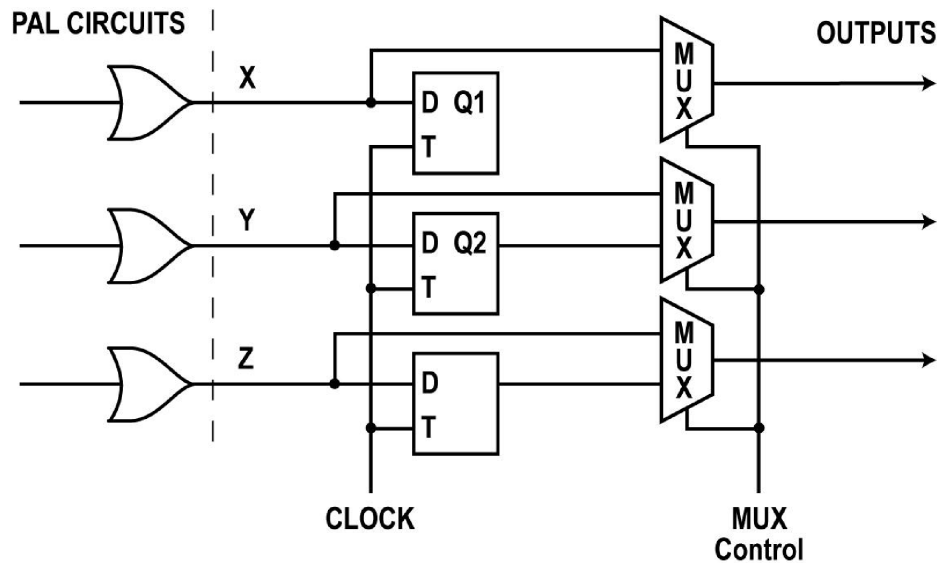


$$X = (A \& !B \& C) + (!A \& !B \& !C)$$

$$Y = (A \& B \& C) + (!A \& !B \& !C)$$

$$Z = (!A \& B \& C) + (A)$$

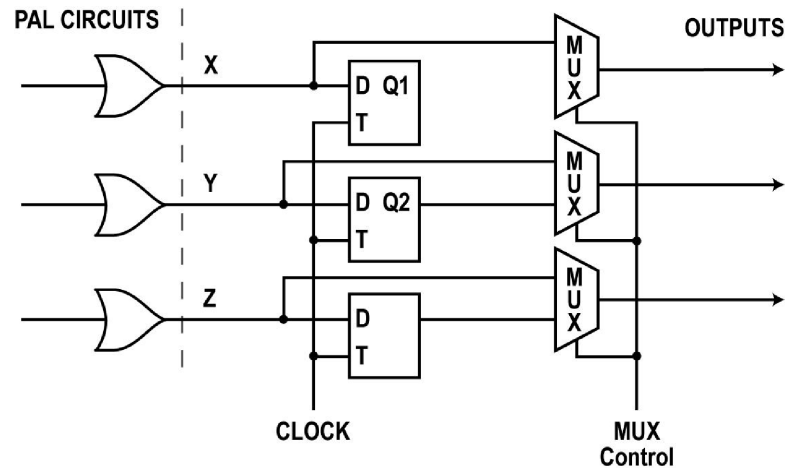
# Register Outputs



Some PLDs provide flip flops (FF) on the outputs of the logic gates to store the result of the logic operations as shown in the example here. Assume that the inputs to the D-type FF come from the X, Y, and Z outputs of the PAL in the previous figures. When a clock pulse occurs on the CLK input, the logic values of X, Y, and Z are stored in the FFs.



# Register Outputs Variation



A variation of this is the inclusion of a 2-input multiplexer (MUX) that can be used to select whether the output is simply from the OR gates or from the FFs. If the multiplexer (MUX) control input is binary 0, the outputs will come directly from the OR gates to the MUX output. If the MUX control input is binary 1, then the FF output is passed through the MUX to the output. The MUX control input is programmable.

# Generic Logic Array (GAL)

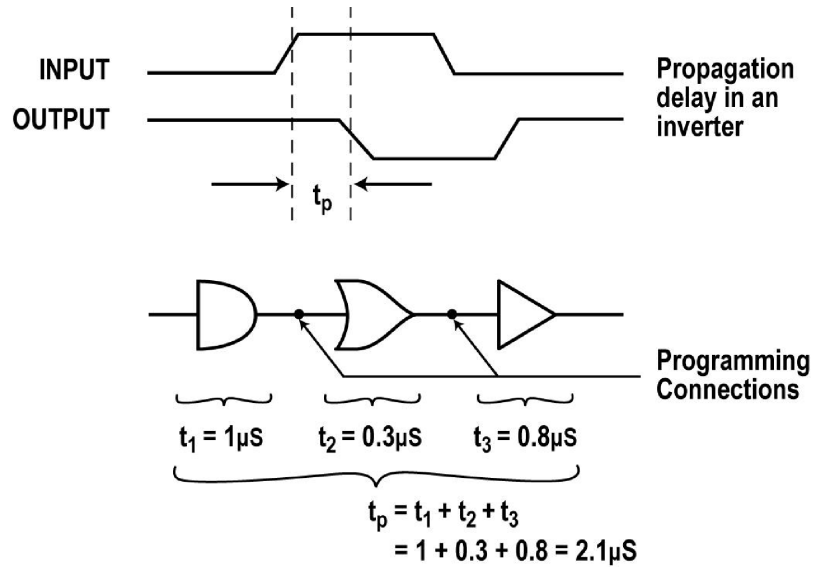
A generic logic array (GAL) is a variation of a PAL where the programming is not permanent. Programming a PROM, PLA, or PAL is usually a one time affair. Once it is programmed, it cannot be changed.

On the other hand, a GAL contains programming elements based upon electrically erasable programmable ROM (EEPROM) technology. Once the circuit is programmed, it remains programmed for an indefinite period. However, the EEPROM storage cells can be erased and reprogrammed.

This makes the GAL a far more useful and practical replacement for most PALs. It is faster and easier to design and provides similar performance to PALs. GALs are made by Lattice Semiconductor Corp.

# A Word About Performance

Performance in digital logic is usually defined by how fast a logic circuit accomplishes its function. Performance is determined by propagation delay ( $t_p$ ) in a circuit. This is the time it takes for an input pulse transition to cause the corresponding response at the output.



$t_1$  = prop. delay AND

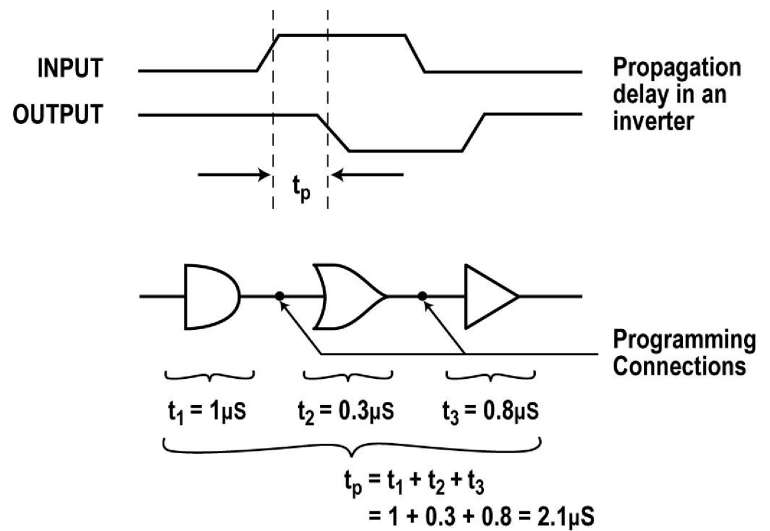
$t_2$  = prop. delay OR

$t_3$  = prop. delay driver

$t_p$  = total propagation delay

# Propagation Delay

Each logic gate or circuit has a specific propagation delay. If the signal must pass through two or more gates or other circuits, the propagation delays add together. The figure shows a logic circuit with three levels of logic that the signal must pass through. Note the propagation delay for each gate or circuit and their sum which is the overall speed. Programming connections and wiring also add to the delay.



$t_1$  = prop. delay AND

$t_2$  = prop. delay OR

$t_3$  = prop. delay driver

$t_p$  = total propagation delay

# Determining the Propagation Delay

Propagation delay is determined by the technology of the IC (TTL, CMOS, etc), the size of the IC components, and the circuit configuration. Most ICs today are CMOS so their size is the main controlling factor. The smaller the MOSFETs in the IC the faster they operate. Small size translates to less resistance and capacitance in the circuits therefore lower propagation delays. The number of transistors and other components in the circuit also determine the delay.

Modern CMOS ICs are made with one micron ( $1\ \mu\text{m}$ ) size transistors or less. The newer circuits have sizes of 0.5, 0.35, 0.25, 0.18, 0.13 and  $0.09\ \mu\text{m}$  geometries. The smaller devices have propagation delays of tens or hundreds of picoseconds per circuit.

# Test your knowledge

## Programmable Logic Devices Knowledge Probe 2 PLD Concepts and Types

Click on [Course Materials](#) at the top of the page.  
Then choose **Knowledge Probe 2**.