# Chapter 11: Programmable Logic

This chapter provides an overview of programmable logic devices (PLDs). The term PLD is used as a generic description for any circuit that can be programmed to implement digital logic. The technology and architectures of PLDs have advanced over time. A historical perspective is given on how the first programmable devices evolved into the programmable technologies that are prevalent today. The goal of this chapter is to provide a basic understanding of the principles of programmable logic devices.

**Learning Outcomes**—After completing this chapter, you will be able to:

11.1 Describe the basic architecture and evolution of programmable logic devices.
11.2 Describe the basic architecture of Field Programmable Gate Arrays (FPGAs).

## 11.1 Programmable Arrays

### 11.1.1 Programmable Logic Array

One of the first commercial PLDs developed using modern integrated circuit technology was the **programmable logic array (PLA)**. In 1970, Texas Instrument introduced the PLA with an architecture that supported the implementation of arbitrary, sum of product logic expressions. The PLA was fabricated with a dense array of AND gates, called an *AND plane*, and a dense array of OR gates, called an *OR plane*. Inputs to the PLA each had an inverter in order to provide the original variable and its complement. Arbitrary SOP logic expressions could be implemented by creating connections between the inputs, the AND plane, and the OR plane. The original PLAs were fabricated with all of the necessary features except the final connections to implement the SOP functions. When a customer provided the desired SOP expression, the connections were added as the final step of fabrication. This configuration technique was similar to an MROM approach. Figure 11.1 shows the basic architecture of a PLA.

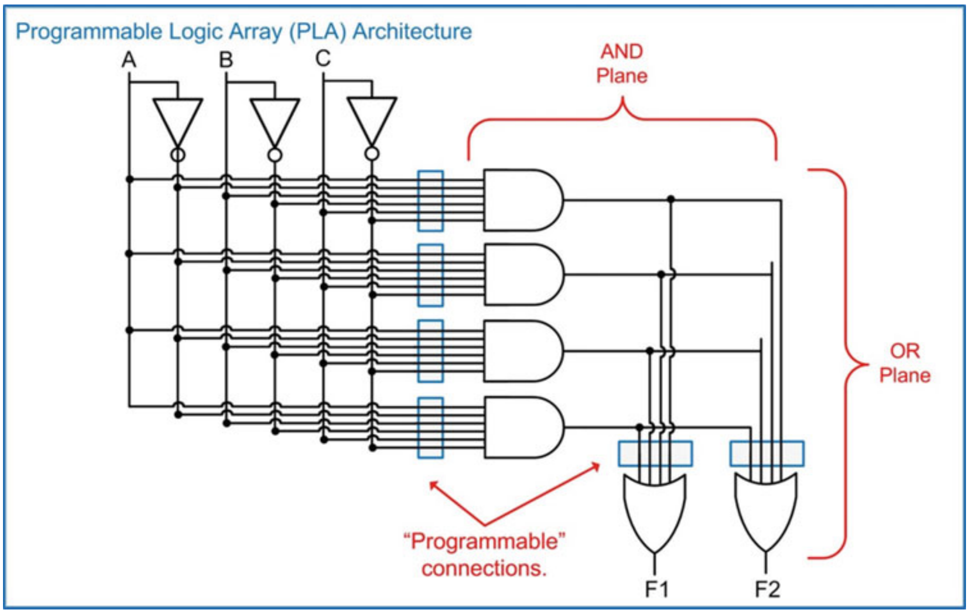

Programmable Logic Array (PLA) Architecture
A
B
C
AND Plane
OR Plane
"Programmable" connections.
F1
F2

**Fig. 11.1**
Programmable logic array (PLA) architecture

A more compact schematic for the PLA is drawn by representing all of the inputs into the AND and OR gates with a single wire. Connections are indicated by inserting Xs at the intersections of wires. Figure 11.2 shows this simplified PLA schematic implementing two different SOP logic expressions.

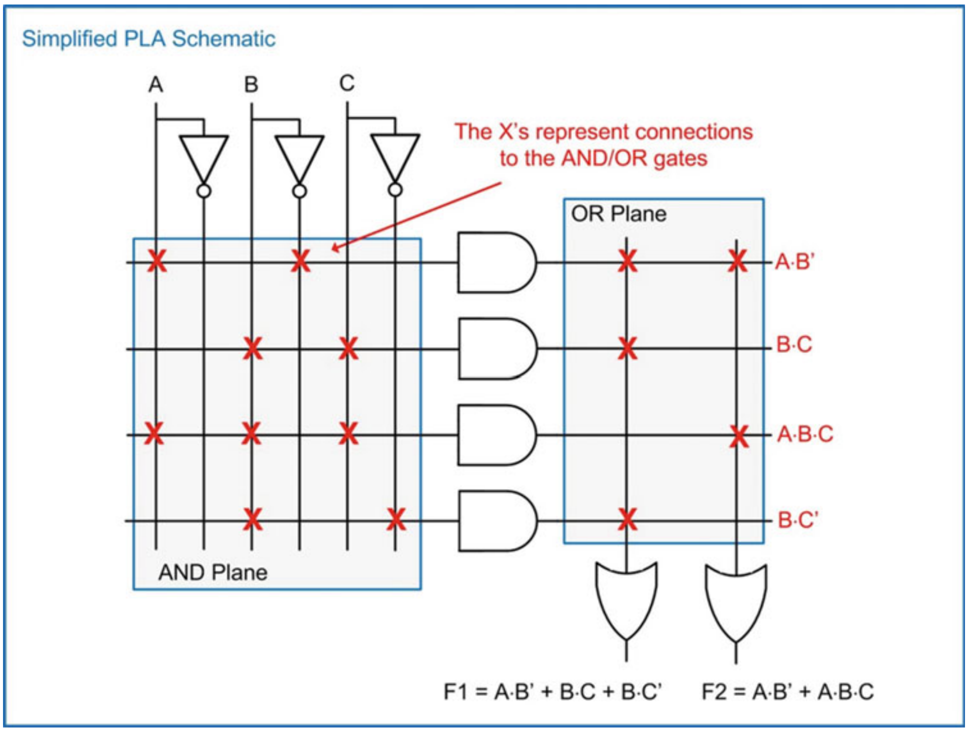


**Fig. 11.2**
Simplified PLA schematic

## 11.1.2 Programmable Array Logic

One of the drawbacks of the original PLA was that the programmability of the OR plane caused significant propagation delays through the combinational logic circuits. In order to improve on the performance of PLAs, the **programmable array logic (PAL)** was introduced in 1978 by the company *Monolithic Memories, Inc*. The PAL contained a programmable AND plane and a *fixed-OR* plane. The fixed-OR plane improved the performance of this programmable architecture. While not having a programmable OR plane reduced the flexibility of the device, most SOP expressions could be manipulated to work with a PAL. Another contribution of the PAL was that the AND plane could be programmed using fuses. Initially, all connections were present in the AND plane. An external programmer was used to blow fuses in order to disconnect the inputs from the AND gates. While the fuse approach provided one-time-only programming, the ability to configure the logic post-fabrication was a significant advancement over the PLA, which had to be programmed at the manufacturer. Figure 11.3 shows the architecture of a PAL.

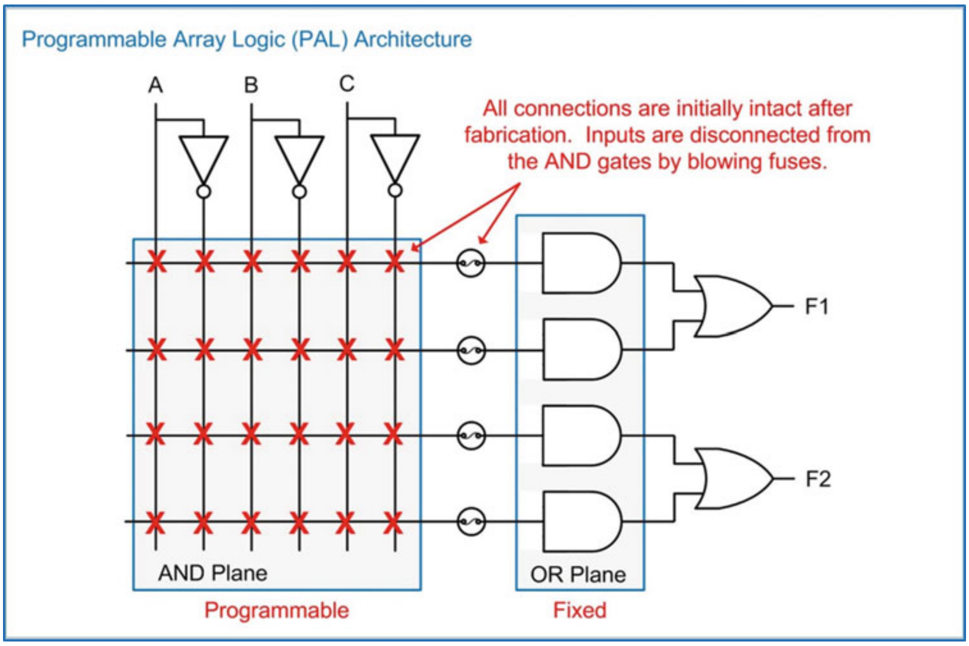

Programmable Array Logic (PAL) Architecture
A
B
C
All connections are initially intact after fabrication. Inputs are disconnected from the AND gates by blowing fuses.
F1
F2
AND Plane
Programmable
OR Plane
Fixed

**Fig. 11.3**
Programmable array logic (PAL) architecture

### 11.1.3 Generic Array Logic

As the popularity of the PAL grew, additional functionality was implemented to support more sophisticated designs. One of the most significant improvements was the addition of an *output logic macrocell (OLMC)*. An OLMC provided a D-flip-flop and a selectable mux so that the output of the SOP circuit from the PAL could be used either as the system output or the input to a D-flip-flop. This enabled the implementation of sequential logic and finite-state machines. The OLMC could also be used to route the I/O pin back into the PAL to increase the number of inputs possible in the SOP expressions. Finally, the OLMC provided a multiplexer to allow feedback from either the PAL output or the output of the D-flip-flop. This architecture was named a **generic array logic (GAL)** to distinguish its features from a standard PAL. Figure 11.4 shows the architecture of a GAL consisting of a PAL and an OLMC.

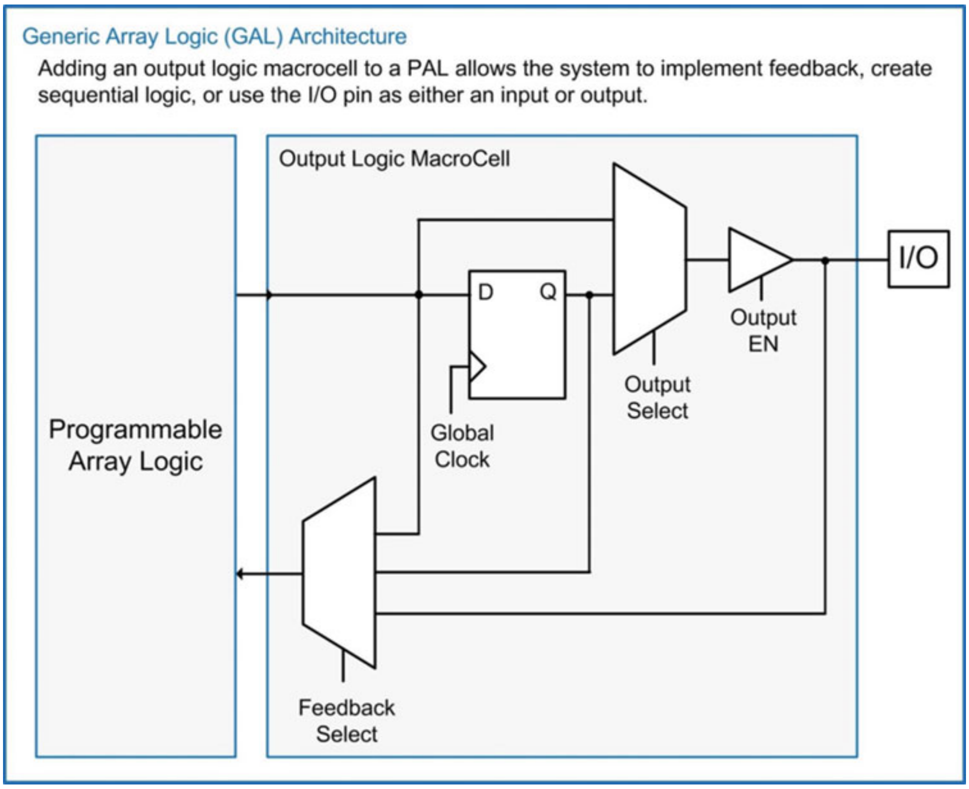

Generic Array Logic (GAL) Architecture
Adding an output logic macrocell to a PAL allows the system to implement feedback, create sequential logic, or use the I/O pin as either an input or output.
Programmable Array Logic
Output Logic MacroCell
D
Q
Global Clock
Output Select
Output EN
I/O
Feedback Select

**Fig. 11.4**
Generic array logic (GAL) architecture

## 11.1.4 Hard Array Logic

For mature designs, PALs and GALs could be implemented as a **hard array logic (HAL)** device. A HAL was a version of a PAL or GAL that had the AND plane connections implemented during fabrication instead of through blowing fuses. This architecture was more efficient for high-volume applications as it eliminated the programming step post-fabrication and the device did not need to contain the additional programming circuitry.

In 1983, *Altera Inc.* was founded as a programmable logic device company. In 1984, Altera released its first version of a PAL with a unique feature that it could be programmed and erased multiple times using a programmer and an UV light source similar to an EEPROM.

## 11.1.5 Complex Programmable Logic Devices

As the demand for larger programmable devices grew, the PAL's architecture was not able to scale efficiently due to a number of reasons: first, as the size of combinational logic circuits increased, the PAL encountered fan-in issues in its AND plane; secondly, for each input that was added to the PAL, the amount of circuitry needed on the chip grew geometrically due to requiring a connection to each AND gate in addition to the area associated with the additional OLMC. This led to a new PLD architecture in which the on-chip interconnect was partitioned across multiple PALs on a single chip. This partitioning meant that not all inputs to the device could be used by each PAL, so the design complexity increased; however, the additional programmable resources outweighed this drawback and this architecture was broadly adopted. This new architecture was called a **complex programmable logic device (CPLD)**. The term ***simple* programmable logic device (SPLD)** was created to describe all of the previous PLD architectures (i.e., PLA, PAL, GAL, and HAL). Figure 11.5 shows the architecture of the CPLD.

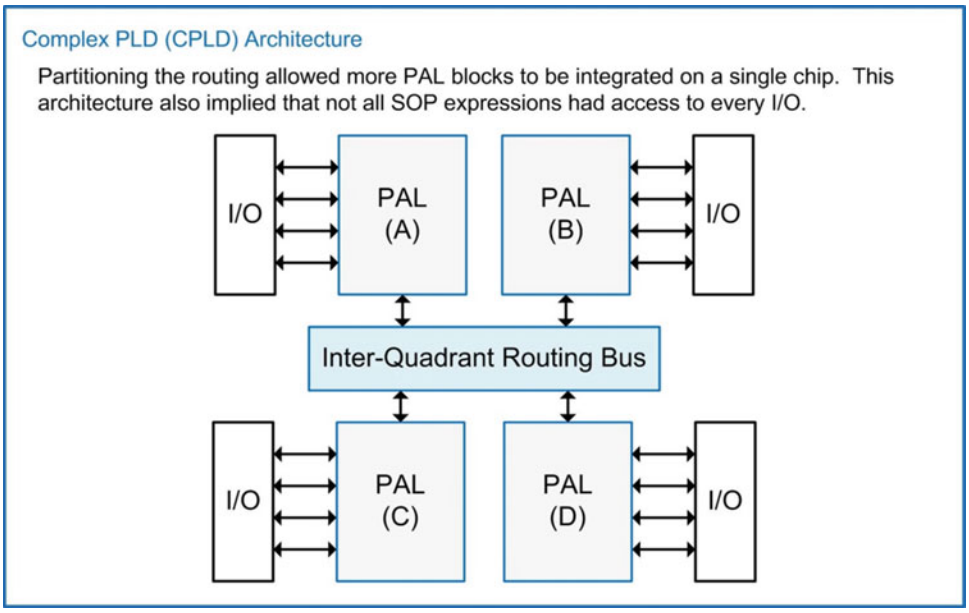

Complex PLD (CPLD) Architecture
Partitioning the routing allowed more PAL blocks to be integrated on a single chip. This architecture also implied that not all SOP expressions had access to every I/O.
I/O
PAL (A)
PAL (B)
I/O
Inter-Quadrant Routing Bus
I/O
PAL (C)
PAL (D)
I/O

**Fig. 11.5**
Complex PLD (CPLD) architecture

**CONCEPT CHECK**

**CC11.1** What is the only source of delay mismatch from the inputs to the outputs in a programmable array?

A) The AND gates will have different delays due to having different numbers of inputs.

B) The OR gates will have different delays due to having different numbers of inputs.

C) An input may or may not go through an inverter before reaching the A ND gates.

D) None. All paths through the programmable array have identical delay.

## 11.2 Field Programmable Gate Arrays

To address the need for even more programmable resources, a new architecture was developed by *Xilinx Inc*. in 1985. This new architecture was called a **field programmable gate array (FPGA)**. An FPGA consists of an array of programmable logic blocks (or logic elements) and a network of programmable interconnect that can be used to connect any logic element to any other logic element. Each logic block contained circuitry to implement arbitrary combinational logic circuits in addition to a D-flip-flop and a multiplexer for signal steering. This architecture effectively implemented an OLMC within each block, thus providing ultimate flexibility and providing significantly more resources for sequential logic. Today, FPGAs are the most commonly used programmable logic device, with Altera Inc. and Xilinx Inc. being the two largest manufacturers. Figure 11.6 shows the generic architecture of an FPGA.

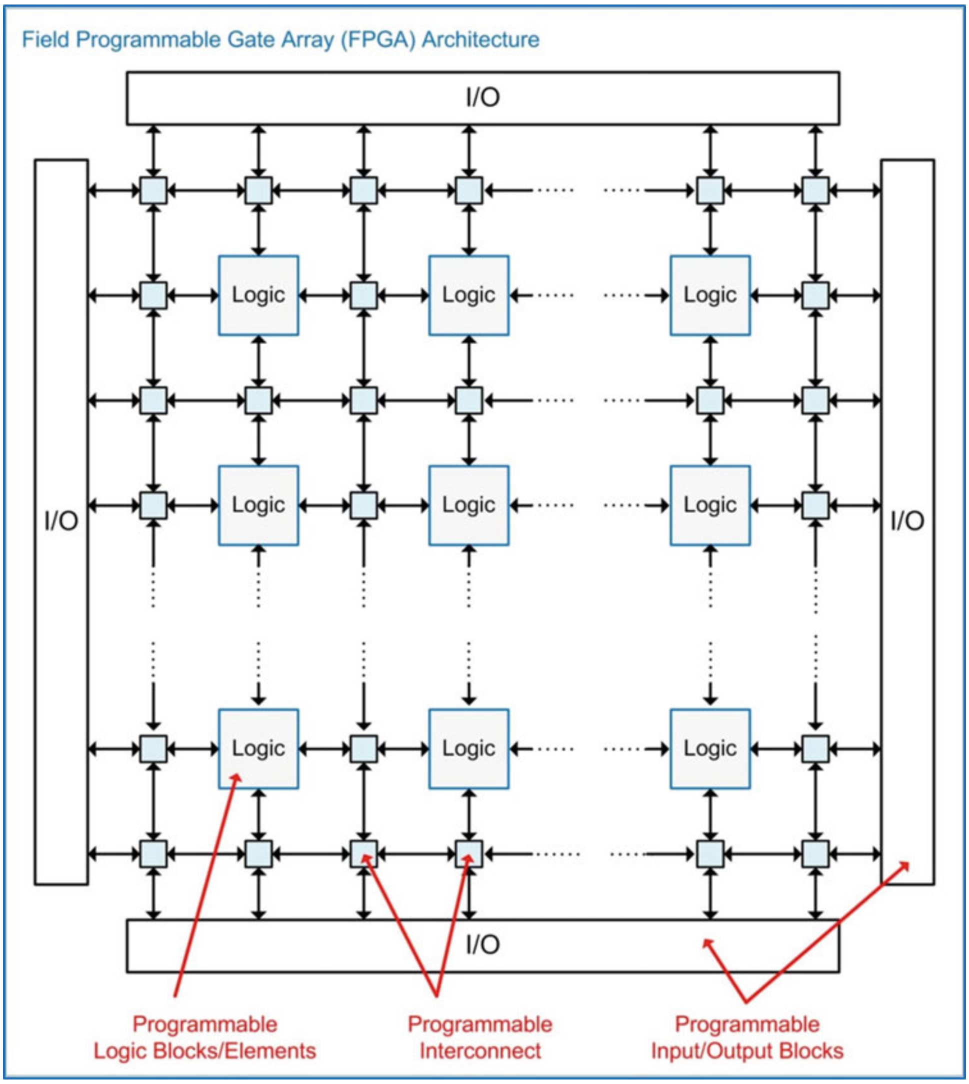
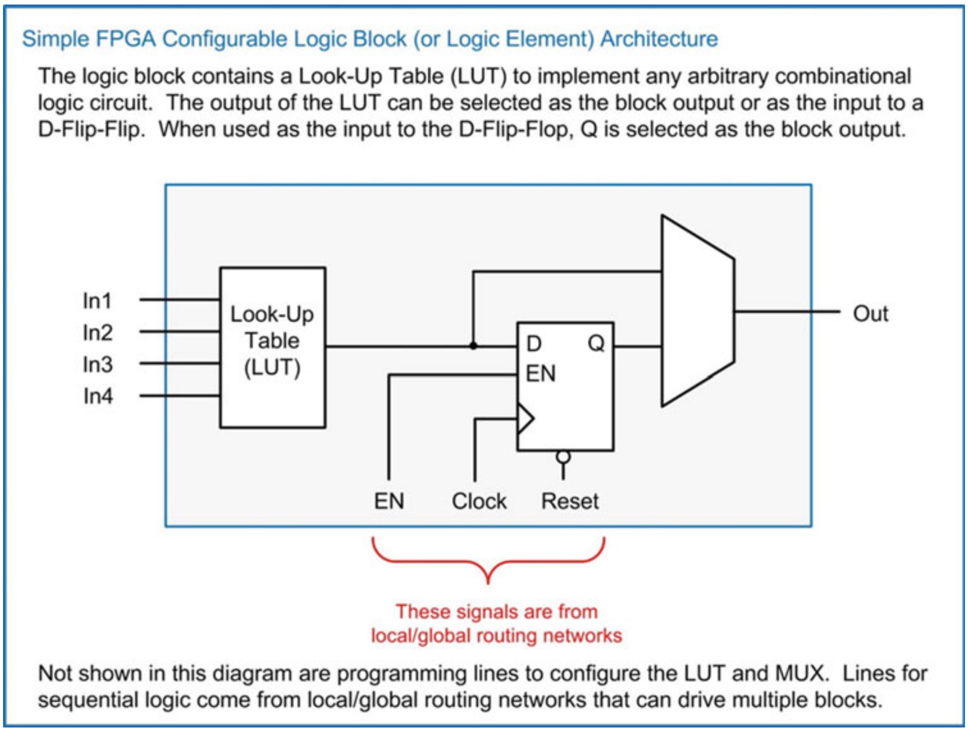

Field Programmable Gate Array (FPGA) Architecture
I/O
I/O
I/O
I/O
Logic
Programmable Logic Blocks/Elements
Programmable Interconnect
Programmable Input/Output Blocks

**Fig. 11.6**
Field programmable gate array (FPGA) architecture

## 11.2.1 Configurable Logic Block (or Logic Element)

The primary reconfigurable structure in the FPGA is the **configurable logic block (CLB)** or **logic element (LE)**. Xilinx Inc. uses the term CLB while Altera uses LE. Combinational logic is implemented using a circuit called a **Look-Up Table (LUT)**, which can implement any arbitrary truth table. The details of an LUT are given in the next section. The CLB/LE also contains a D-flip-flop for sequential logic. A signal steering multiplexer is used to select whether the output of the CLB/LE comes from the LUT or from the D-flip-flop. The LUT can be used to drive a combinational logic expression into the D input of the D-flip-flop, thus creating a highly efficient topology for finite-state machines. A global routing network is used to provide common signals to the CLB/LE such as clock, reset, and enable. This global routing network can provide these common signals to the entire FPGA or local groups of CLB/LEs. Figure 11.7 shows the topology of a simple CLB/LE.


Simple FPGA Configurable Logic Block (or Logic Element) Architecture

The logic block contains a Look-Up Table (LUT) to implement any arbitrary combinational logic circuit. The output of the LUT can be selected as the block output or as the input to a D-Flip-Flip. When used as the input to the D-Flip-Flop, Q is selected as the block output.

Not shown in this diagram are programming lines to configure the LUT and MUX. Lines for sequential logic come from local/global routing networks that can drive multiple blocks.

**Fig. 11.7**
Simple FPGA configurable logic block (or logic element)

CLB/LEs have evolved to include numerous other features such as carry in/carry out signals so that arithmetic operations can be cascaded between multiple blocks in addition to signal feedback and D-flip-flop initialization.

## 11.2.2 Look-Up Tables

An LUT is the primary circuit used to implement combinational logic in FPGAs. This topology has also been adopted in modern CPLDs. In an LUT, the desired outputs of a truth table are loaded into a local configuration SRAM memory. The SRAM memory provides these values to the inputs of a multiplexer. The inputs to the combinational logic circuit are then used as the select lines to the multiplexer. For an arbitrary input to the combinational logic circuit, the multiplexer selects the appropriate value held in the SRAM and routes it to the output of the circuit. In this way, the multiplexer *looks up* the appropriate output value based on the input code. This architecture has the advantage that any logic function can be created without creating a custom logic circuit. Also, the delay through the LUT is identical regardless of what logic function is being implemented. Figure 11.8 shows a 2-input combinational logic circuit implemented with a 4-input multiplexer.

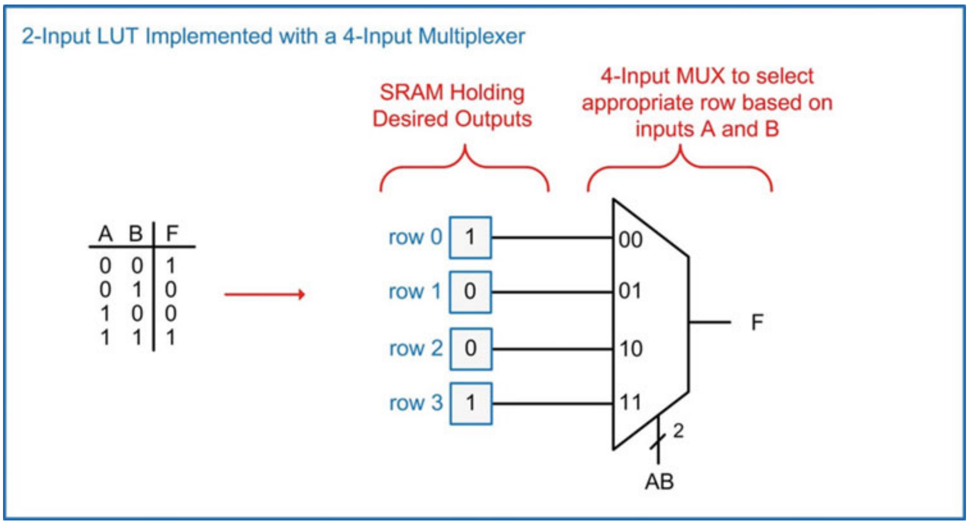

2-Input LUT Implemented with a 4-Input Multiplexer
SRAM Holding Desired Outputs
4-Input MUX to select appropriate row based on inputs A and B
A B | F
0 0 | 1
0 1 | 0
1 0 | 0
1 1 | 1
row 0 1
row 1 0
row 2 0
row 3 1
00
01
10
11
F
2
AB

**Fig. 11.8**
2-Input LUT implemented with a 4-input multiplexer

Fan-in limitations can be encountered quickly in LUTs as the number of inputs of the combinational logic circuit being implemented grows. Recall that multiplexers are implemented with an SOP topology in which each product term in the first level of logic has a number of inputs equal to the number of select lines plus one. Also recall that the sum term in the second level of logic in the SOP topology has a number of inputs equal to the total number of inputs to the multiplexer. In the example circuit shown in Fig. 11.8, each product term in the multiplexer will have three inputs and the sum term will have four inputs. As an illustration of how quickly fan-in limitations are encountered, consider the implication of increasing the number of inputs in Fig. 11.8 from two to three. In this new configuration, the number of inputs in the product terms will increase from three to four and the number of inputs in the sum term will increase from four to eight. Eight inputs is often beyond the fan-in specifications of modern devices, meaning that even a 3-input combinational logic circuit will encounter fan-in issues when implemented using an LUT topology.

To address this issue, multiplexer functionality in LUTs is typically implemented as a series of smaller, cascaded multiplexers. Each of the smaller multiplexers progressively chooses which row of the truth table to route to the output of the LUT. This eliminates fan-in issues at the expense of adding additional levels of logic to the circuit. While cascading multiplexers increase the overall circuit delay, this approach achieves a highly consistent delay because regardless of the truth table output value, the number of levels of logic through the multiplexers is always the same. Figure 11.9 shows how the 2-input truth table from Fig. 11.8 can be implemented using a 2-level cascade of 2-input multiplexers.

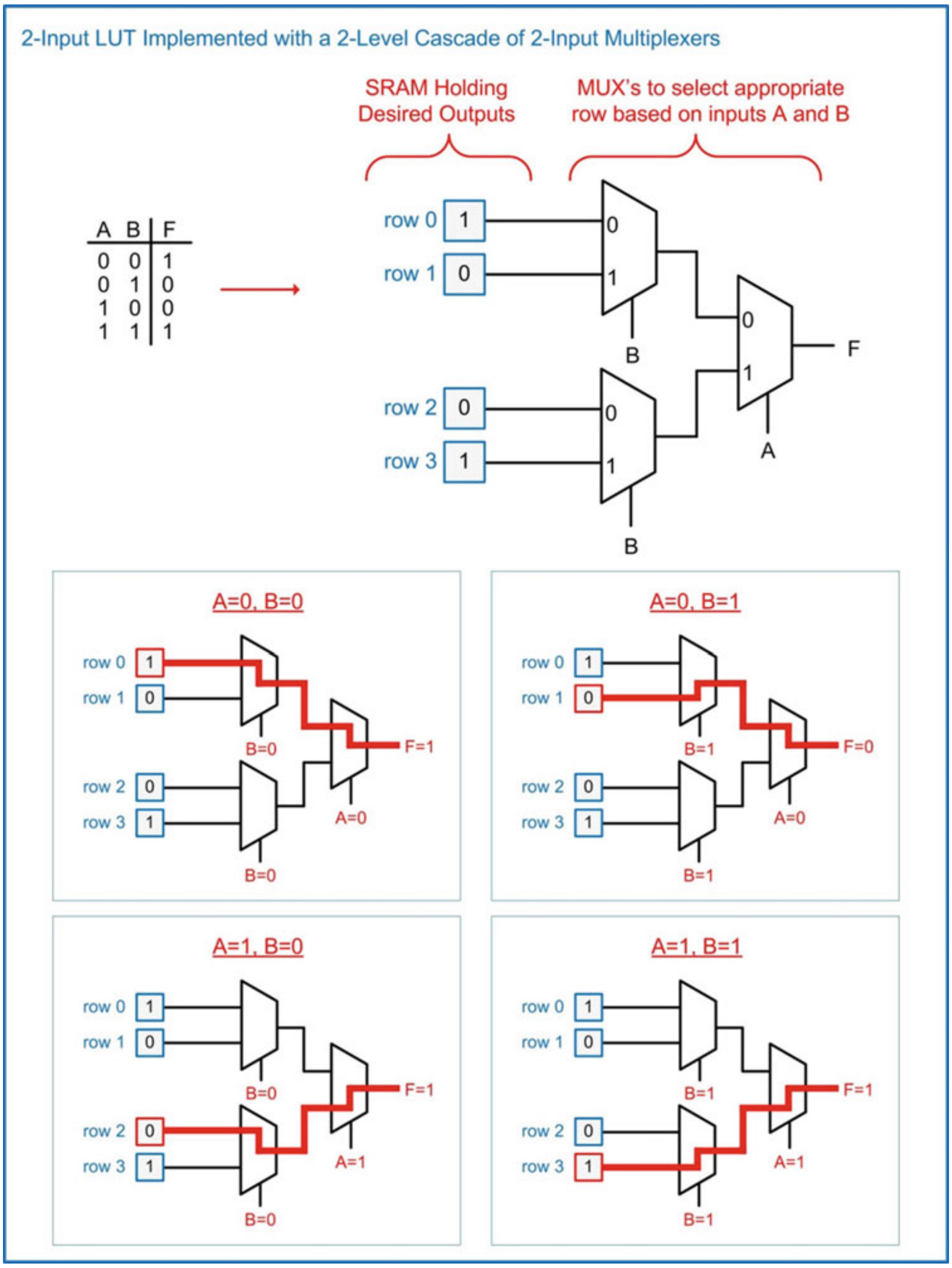

2-Input LUT Implemented with a 2-Level Cascade of 2-Input Multiplexers
SRAM Holding Desired Outputs
MUX's to select appropriate row based on inputs A and B
A B F
0 0 1
0 1 0
1 0 0
1 1 1
row 0 1
row 1 0
row 2 0
row 3 1
0
1
B
A
F
A=0, B=0
F=1
A=0, B=1
F=0
A=1, B=0
F=1
A=1, B=1
F=1

**Fig. 11.9**
2-Input LUT implemented with a 2-level cascade of 2-input multiplexers

If more inputs are needed in the LUT, additional MUX levels are added. Figure 11.10 shows the architecture for a 3-input LUT implemented with a 3-level cascade of 2-input multiplexers.

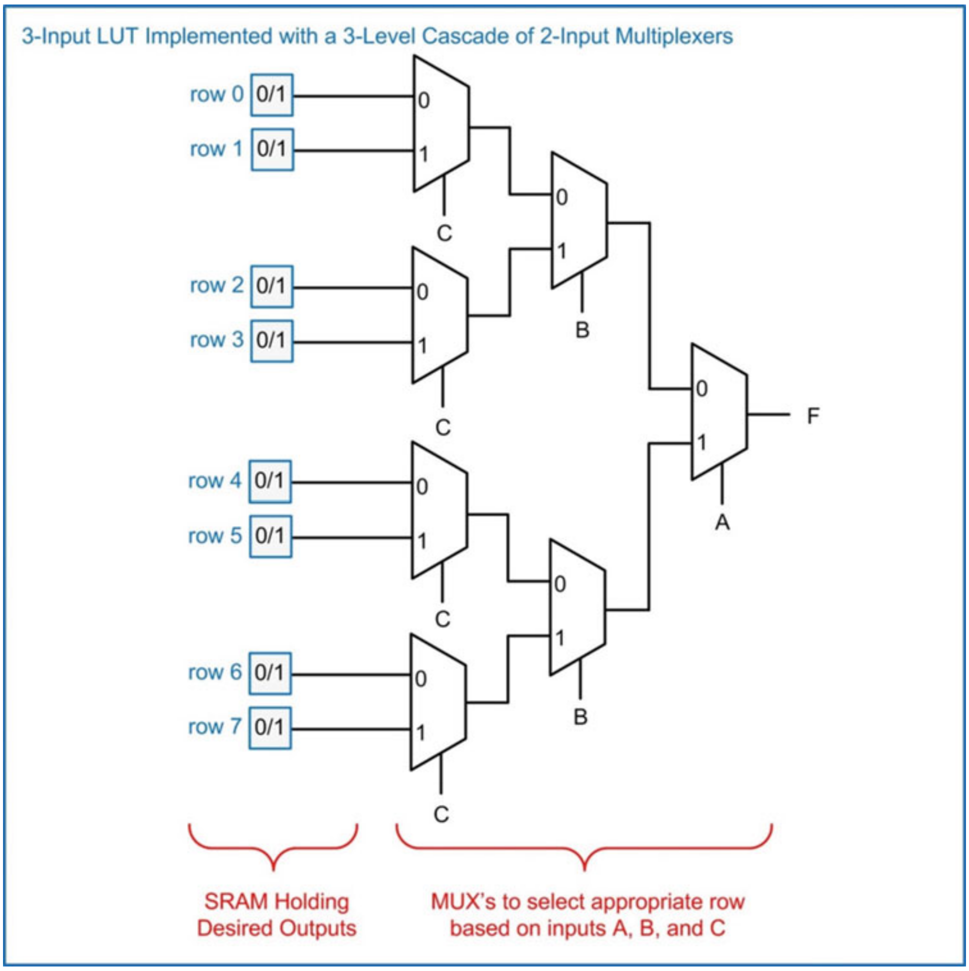

3-Input LUT Implemented with a 3-Level Cascade of 2-Input Multiplexers
row 0
0/1
row 1
0/1
row 2
0/1
row 3
0/1
row 4
0/1
row 5
0/1
row 6
0/1
row 7
0/1
0
1
C
B
A
F
SRAM Holding Desired Outputs
MUX's to select appropriate row based on inputs A, B, and C

**Fig. 11.10**
3-Input LUT implemented with a 3-level cascade of 2-input multiplexers

Modern FPGAs can have LUTs with up to 6 inputs. If even more inputs are needed in a combinational logic expression, then multiple CLB/LEs are used that form even larger LUTs.

## 11.2.3 Programmable Interconnect Points (PIPs)

The configurable routing network on an FPGA is accomplished using programmable switches. A simple model for these switches is to use an NMOS transistor. A configuration SRAM bit stores whether the switch is opened or closed. On the FPGA, interconnect is routed vertically and horizontally between the CLB/LEs with switching points placed throughout the FPGA to facilitate any arbitrary routing configuration. Figure 11.11 shows how the routing can be configured into a full cross-point configuration using programmable switches.

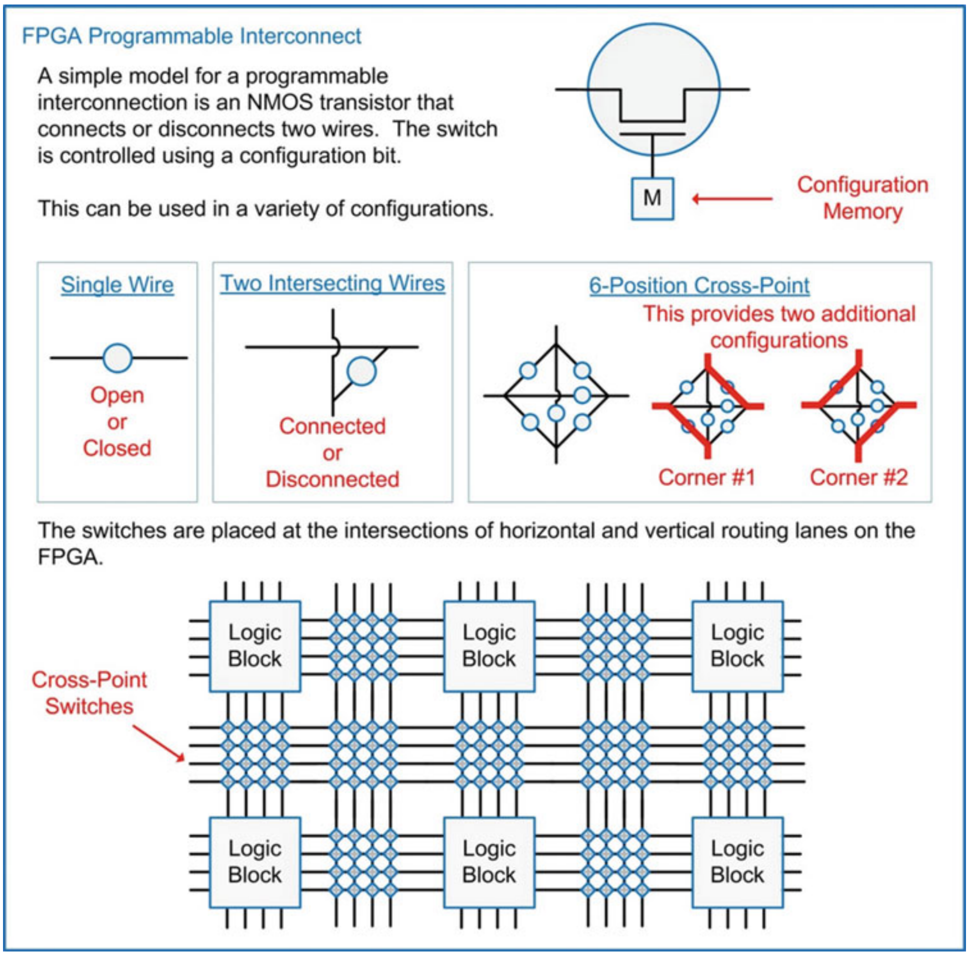

FPGA Programmable Interconnect
A simple model for a programmable interconnection is an NMOS transistor that connects or disconnects two wires. The switch is controlled using a configuration bit.
This can be used in a variety of configurations.
M
Configuration Memory
Single Wire
Open or Closed
Two Intersecting Wires
Connected or Disconnected
6-Position Cross-Point
This provides two additional configurations
Corner #1
Corner #2
The switches are placed at the intersections of horizontal and vertical routing lanes on the FPGA.
Logic Block
Cross-Point Switches

**Fig. 11.11**
FPGA programmable interconnect

### 11.2.4 Input/Output Block

FPGAs also contain **input/output blocks (IOBs)** that provide programmable functionality for interfacing to external circuitry. The IOBs contain both driver and receiver circuitry so that they can be programmed to be either inputs or outputs. D-flip-flops are included in both the input and output circuitry to support synchronous logic. Figure 11.12 shows the architecture of an FPGA IOB.

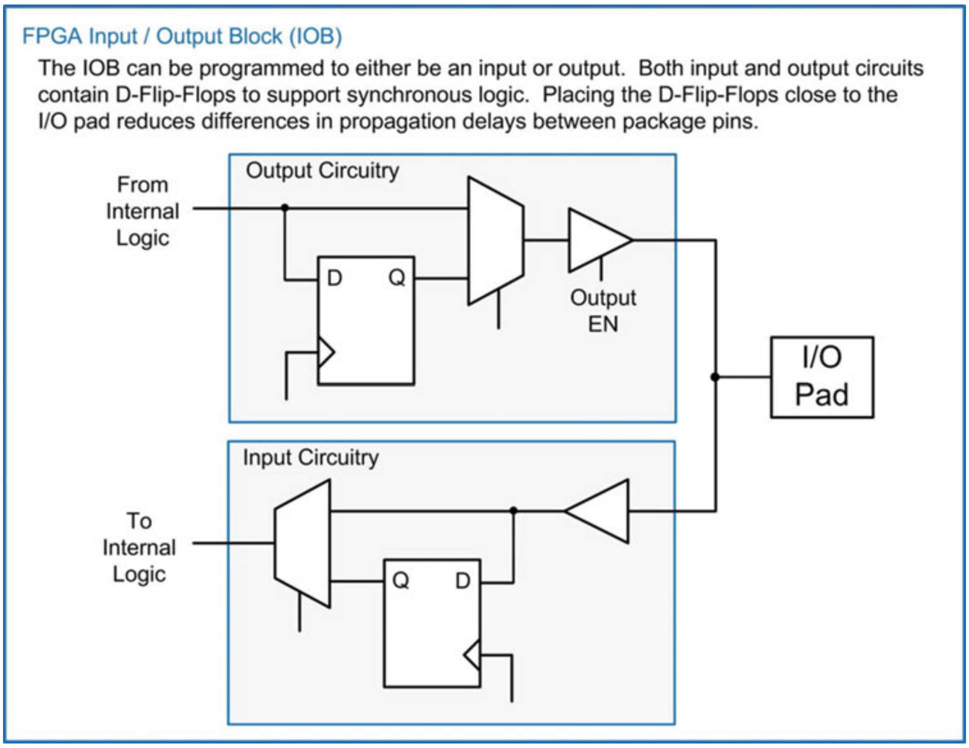

FPGA Input / Output Block (IOB)
The IOB can be programmed to either be an input or output. Both input and output circuits contain D-Flip-Flops to support synchronous logic. Placing the D-Flip-Flops close to the I/O pad reduces differences in propagation delays between package pins.
From Internal Logic
Output Circuitry
D
Q
Output EN
I/O Pad
Input Circuitry
To Internal Logic
Q
D

**Fig. 11.12**
FPGA input/output block (IOB)

### 11.2.5 Configuration Memory

All of the programming information for an FPGA is contained within configuration SRAM that is distributed across the IC. Since this memory is volatile, the FPGA will lose its configuration when power is removed. Upon power-up, the FPGA must be programmed with its configuration data. This data is typically held in a nonvolatile memory such as FLASH. The “FP” in FPGA refers to the ability to program the device in the *field*, or post-fabrication. The “GA” in FPGA refers to the array topology of the programmable logic blocks or elements.

**CONCEPT CHECK**

**CC11.2** What is the primary difference between an FPGA and a CPLD?

A) The ability to create arbitrary SOP logic expressions.

B) The abundance of configurable routing.

C) The inclusion of D-flip-flops.

D) The inclusion of programmable I/O pins.

## Summary

- ❖ A *programmable logic device* (PLD) is a generic term for a circuit that can be configured to implement arbitrary logic functions.
- ❖ There are a variety of PLD architectures that have been used to implement combinational logic. These include the PLA and PAL. These devices contain an AND plane and an OR plane. The AND plane is configured to implement the product terms of an SOP expression. The OR plane is configured to implement the sum term of an SOP expression.
- ❖ A GAL increases the complexity of logic arrays by adding sequential logic storage and programmable I/O capability.
- ❖ A CPLD significantly increases the density of PLDs by connecting an array of PALs together and surrounding the logic with I/O drivers.
- ❖ FPGAs contain an array of programmable logic elements that each consists of combinational logic capability and sequential logic storage. FPGAs also contain a programmable interconnect network that provides the highest level of flexibility in programmable logic.
- ❖ An LUT is a simple method to create a programmable combinational logic circuit. An LUT is simply a multiplexer with the inputs to the circuit connected to the select lines of the MUX. The desired outputs of the truth table are connected to the MUX inputs. As different input codes arrive on the select lines of the MUX, they *select* the corresponding logic value to be routed to the system output.

## Exercise Problems

### Section 11.1: Programmable Arrays

**11.1.1** Name the type of programmable logic described by the characteristic: *this device adds an output logic macrocell to a traditional PAL.*

**11.1.2** Name the type of programmable logic described by the characteristic: *this device combines multiple PALs on a single chip with a partitioned interconnect system*.

**11.1.3** Name the type of programmable logic described by the characteristic: *this device has a programmable AND plane and programmable OR plane.*

**11.1.4** Name the type of programmable logic described by the characteristic: *this device has a programmable AND plane and fixed OR plane.*

**11.1.5** Name the type of programmable logic described by the characteristic: *this device is a PAL or GAL that is programmed during manufacturing.*

**11.1.6** For the following unconfigured PAL schematic in Fig. 11.13, draw in the connection points (i.e., the Xs) to implement the two SOP logic expressions shown on the outputs.

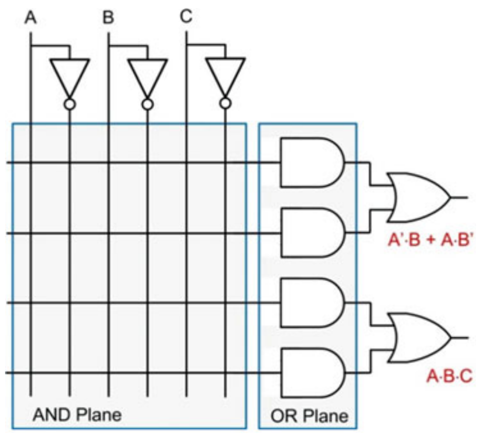


**Fig. 11.13**
Blank PAL Schematic

## Section 11.2: Field Programmable Gate Arrays

**11.2.1** Give a general description of an FPGA that differentiates it from other programmable logic devices.

**11.2.2** Which part of an FPGA is described by the following characteristic: *this is used to interface between the internal logic and external circuitry*.

**11.2.3** Which part of an FPGA is described by the following characteristic: *this is used to configure the on-chip routing*.

**11.2.4** Which part of an FPGA is described by the following characteristic: *this is the primary programmable element that makes up the array.*

**11.2.5** Which part of an FPGA is described by the following characteristic: *this part is used to implement the combinational logic within the array.*

**11.2.6** Draw the logic diagram of a 4-input LUT to implement the truth table provided in Fig. 11.14. Implement the LUT with only 2-input multiplexers. Be sure to label the exact location of the inputs (A, B, C, and D), the desired value for each row of the truth table, and the output (F) in the diagram.

| A | B | C | D | F |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

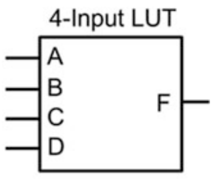


**Fig. 11.14**
4-Input LUT Exercise