UNIT – V

SEMICONDUCTOR INTEGRATED CIRCUIT DESIGN: PLAs, FPGAs, CPLDs, Standard Cells, Programmable Array Logic, Design Approach, Parameters influencing low power design.

Programmable Logic Devices:

Logic devices constitute one of the three important classes of devices used to build digital electronics systems, memory devices and microprocessors being the other two. Memory devices such as ROM and RAM are used to store information such as the software instructions of a program or the contents of a database, and microprocessors execute software instructions to perform a variety of functions, from running a word-processing program to carrying out far more complex tasks. Logic devices implement almost every other function that the system must perform, including device-to-device interfacing, data timing, control and display operations and so on. So far, we have discussed those logic devices that perform fixed logic functions decided upon at the manufacturing stage. Logic gates, multiplexers, demultiplexers, arithmetic circuits, etc., are some examples. Sequential logic devices such as flip-flops, counters, registers, etc., to be discussed in the following chapters, also belong to this category of logic devices. In the present chapter, we will discuss a new category of logic devices called programmable logic devices (PLDs). The function to be performed by a programmable logic device is undefined at the time of its manufacture. These devices are programmed by the user to perform a range of functions depending upon the logic capacity and other features offered by the device. We will begin with a comparison of fixed and programmable logic, and then follow this up with a detailed description of different types of PLDs in terms of operational fundamentals, salient features, architecture and typical applications.

5.1 Fixed Logic Versus Programmable Logic

As outlined in the introduction, there are two broad categories of logic devices, namely fixed logic devices and programmable logic devices. Whereas a fixed logic device such as a logic gate or a multiplexer or a flip-flop performs a given logic function that is known at the time of device manufacture, a programmable logic device can be configured by the user to perform a large variety of logic functions. In terms of the internal schematic arrangement of the two types of device, the circuits or building blocks and their interconnections in a fixed logic device are permanent and cannot be altered after the device is manufactured.

A *programmable logic device* offers to the user a wide range of logic capacity in terms of digital building blocks, which can be configured by the user to perform the intended function or set of functions. This configuration can be modified or altered any number of times by the user by reprogramming the device. Figure 5.1 shows a simple logic circuit comprising four three-input AND gates and a four-input OR gate. This circuit produces an output that is the sum output of a full adder. Here, A and B are the two bits to be added, and C is the carry-in bit. It is a fixed logic device as the circuit is unalterable from outside owing to fixed interconnections between the various building blocks.

Figure 5.2 shows the logic diagram of a simple programmable device. The device has an array of four six-input AND gates at the input and a four-input OR gate at the output. Each AND gate can handle three variables and thus can produce a product term of three variables.
The three variables (A, B and C in this case) or their complements can be programmed to appear at the inputs of any of the four AND gates through fusible links called antifuses. This means that each AND gate can produce the desired three-variable product term. It may be mentioned here that an antifuse performs a function that is opposite to that performed by a conventional electrical fuse. A fuse has a low initial resistance and permanently breaks an electrically conducting path when current through it exceeds a certain limiting value. In the case of an antifuse, the initial resistance is very high and it is designed to create a low-resistance electrically conducting path when voltage across it exceeds a certain level. As a result, this circuit can be programmed to generate any three-variable sum-of-products Boolean function having four minterms by activating the desired fusible links. For example, the circuit could be programmed to produce the sum output resulting from the addition of three bits (the sum output in the case of a full adder) or to produce difference outputs resulting from subtraction of two bits with a borrow-in (the difference output in the case of a full subtractor).

We can visualize that the logic circuit of Fig. 5.2 has a programmable AND array at the input and a fixed OR gate at the output. Incidentally, this is the architecture of programmable logic devices called programmable array logic (PAL). Practical PAL devices have a much larger number of programmable AND gates and fixed OR gates to have enhanced logic capacity and performance capability.

5.1.1 Advantages and Disadvantages

1. If we want to build a fixed logic device to perform a certain specific function, the time required from design to the final stage when the manufactured device is actually available for use could easily be several months to a year or so. PLD-based design requires much less time from design cycle to production run.

2. In the case of fixed logic devices, the process of design validation followed by incorporation of changes, if any, involves substantial nonrecurring engineering (NRE) costs, which leads to an enhanced cost of the initial prototype device. In the case of PLDs, inexpensive software tools can be used for quick validation of designs. The programmable feature of these devices allows quick incorporation of changes and also a quick testing of the device in an actual application environment. In this case, the device used for prototyping is the same as the one that would qualify for use in the end equipment.
3. In the case of programmable logic devices, users can change the circuit as often as they want to until the design operates to their satisfaction. PLDs offer to the users much more flexibility during the design cycle. Design iterations are nothing but changes to the programming file.

4. Fixed logic devices have an edge for large-volume applications as they can be mass produced more economically. They are also the preferred choice in applications requiring the highest performance level.

Figure 5.2 Simple programmable logic circuit.

5.2 Programmable Logic Devices – An Overview

There are many types of programmable logic device, distinguishable from one another in terms of architecture, logic capacity, programmability and certain other specific features. In this section, we will briefly discuss commonly used PLDs and their salient features. A detailed description of each of them will follow in subsequent sections.

5.2.1 Programmable ROMs

PROM (Programmable Read Only Memory) and EPROM (Erasable Programmable Read Only Memory) can be considered to be predecessors to PLDs. The architecture of a programmable ROM allows the user to hardware-implement an arbitrary combinational function of a given number of inputs. When used as a memory device, n inputs of the ROM (called address lines in this case) and m outputs (called data lines) can be used to store $2^n \cdot m$-bit words. When used as a PLD, it can be used to implement m different combinational functions, with each function being a chosen function of n variables.
Any conceivable n-variable Boolean function can be made to appear at any of the m output lines. A generalized ROM device with n inputs and m outputs has \(2^n\) hard-wired AND gates at the input and m programmable OR gates at the output. Each AND gate has n inputs, and each OR gate has \(2^n\) inputs. Thus, each OR gate can be used to generate any conceivable Boolean function of n variables, and this generalized ROM can be used to produce m arbitrary n-variable Boolean functions. The AND array produces all possible minterms of a given number of input variables, and the programmable OR array allows only the desired minterms to appear at their inputs. Figure 5.3 shows the internal architecture of a PROM having four input lines, a hard-wired array of 16 AND gates and a programmable array of four OR gates. A cross (×) indicates an intact (or unprogrammed) fusible link or interconnection, and a dot (•) indicates a hard-wired interconnection. PROMs, EPROMs and EEPROMs (Electrically Erasable Programmable Read Only Memory) can be programmed using standard PROM programmers. One of the major disadvantages of PROMs is their inefficient use of logic capacity. It is not economical to use PROMs for all those applications where only a few minterms are needed. Other disadvantages include relatively higher power consumption and an inability to provide safe covers for asynchronous logic transitions. They are usually much slower than the dedicated logic circuits. Also, they cannot be used to implement sequential logic owing to the absence of flip-flops.

5.2.2 Programmable Logic Array

A programmable logic array (PLA) device has a programmable AND array at the input and a programmable OR array at the output, which makes it one of the most versatile PLDs. Its architecture differs from that of a PROM in the following respects. It has a programmable AND array rather than a hard-wired AND array. The number of AND gates in an m-input PROM is always equal to \(2^m\). In the case of a PLA, the number of AND gates in the programmable AND array for m input variables is usually much less than \(2^m\), and the number of inputs of each of the OR gates equals the number of AND gates. Each OR gate can generate an arbitrary Boolean function with a maximum of minterms equal to the number of AND gates. Figure 5.4 shows the internal architecture of a PLA device with four input lines, a programmable array of eight AND gates at the input and a programmable array of two OR gates at the output. A PLA device makes more efficient use of logic capacity than a PROM. However, it has its own disadvantages resulting from two sets of programmable fuses, which makes it relatively more difficult to manufacture, program and test.

5.2.3 Programmable Array Logic

Programmable array logic (PAL) architecture has a programmable AND array at the input and a fixed OR array at the output. The programmable AND array of a PAL device is similar to that of a PLA device. That is, the number of programmable AND gates is usually smaller than the number required to generate all possible minterms of the given number of input variables. The OR array is fixed and the AND outputs are equally divided between available OR gates. For instance, a practical PAL device may have eight input variables, 64 programmable AND gates and four fixed OR gates, with each OR gate having 16 inputs. That is, each OR gate is fed from 16 of the 64 AND outputs. Figure 5.5 shows the internal architecture of a PAL device that has four input lines, an array of eight AND gates at the input and two OR gates at the output, to introduce readers to the arrangement of various building blocks inside a PAL device and allow them a comparison between different programmable logic devices.
Figure 5.4 Internal architecture of a PLA device.

Figure 5.5 Internal architecture of a PAL device.
5.2.4 Generic Array Logic
A generic array logic (GAL) device is similar to a PAL device and as invented by Lattice Semiconductor. It differs from a PAL device in that the programmable AND array of a GAL device can be erased and reprogrammed. Also, it has reprogrammable output logic. This feature makes it particularly attractive at the device prototyping stage, as any bugs in the logic can be corrected by reprogramming. A similar device called PEEL (Programmable Electrically Erasable Logic) was introduced by the International CMOS Technology (ICT) Corporation.

5.2.5 Complex Programmable Logic Device
Programmable logic devices such as PLAs, PALs, GALs and other PAL-like devices are often grouped into a single category called simple programmable logic devices (SPLDs) to distinguish them from the ones that are far more complex. A complex programmable logic device (CPLD), as the name suggests, is a much more complex device than any of the programmable logic devices discussed so far. A CPLD may contain circuitry equivalent to that of several PAL devices linked to each other by programmable interconnections. Figure 5.6 shows the internal structure of a typical CPLD. Each of the four logic blocks is equivalent to a PLD such as a PAL device. The number of logic blocks in a CPLD could be more or less than four. Each of the logic blocks has programmable interconnections. A switch matrix is used for logic block to logic block interconnections. Also, the switch matrix in a CPLD may or may not be fully connected. That is, some of the possible connections between logic block outputs and inputs may not be supported by a given CPLD. While the complexity of a typical PAL device may be of the order of a few hundred logic gates, a CPLD may have a complexity equivalent to tens of thousands of logic gates. When compared with FPGAs, CPLDs offer predictable timing characteristics owing to their less flexible internal architecture and are thus ideal for critical control applications and other applications where a high performance level is required. Also, because of their relatively much lower power consumption and lower cost, CPLDs are an ideal solution for battery-operated portable applications such as mobile phones, digital assistants and so on. A CPLD can be programmed either by using a PAL programmer or by feeding it with a serial data stream from a PC after soldering it on the PC board. A circuit on the CPLD decodes the data stream and configures it to perform the intended logic function.

![CPLD architecture](image-url)
5.2.6 Field-Programmable Gate Array

A field-programmable gate array (FPGA) uses an array of logic blocks, which can be configured by the user. The term ‘field-programmable’ here signifies that the device is programmable outside the factory where it is manufactured. The internal architecture of an FPGA device has three main parts, namely the array of logic blocks, the programmable interconnects and the I/O blocks. Figure 5.7 shows the architecture of a typical FPGA. Each of the I/O blocks provides an individually selectable input, output or bidirectional access to one of the general-purpose I/O pins on the FPGA package. The logic blocks in an FPGA are no more complex than a couple of logic gates or a look-up table feeding a flip-flop. The programmable interconnects connect logic blocks to logic blocks and also I/O blocks to logic blocks.

FPGAs offer a much higher logic density and much larger performance features compared with CPLDs. Some of the contemporary FPGA devices offer a logic complexity equivalent to that of eight million system gates. Also, these devices offer features such as built-in hard-wired processors, large memory, clock management systems and support for many of the contemporary device-to-device signaling technologies. FPGAs find extensive use in a variety of applications, which include data processing and storage, digital signal processing, instrumentation and telecommunications.

FPGAs are also programmed like CPLDs after they are soldered onto the PC board. In the case of FPGAs, the programmed configuration is usually volatile and therefore needs to be reloaded whenever power is applied or a different functionality is required.

![FPGA Architecture Diagram]

Figure 5.7 FPGA architecture.
5.4 Programmable Logic Array

A programmable logic array (PLA) enables logic functions expressed in sum-of-products form to be implemented directly. It is similar in concept to a PROM. However, unlike a PROM, the PLA does not provide full decoding of the input variables and does not generate all possible minterms. While a PROM has a fixed AND gate array at the input and a programmable OR gate array at the output, a PLA device has a programmable AND gate array at the input and a programmable OR gate array at the output. In a PLA device, each of the product terms of the given Boolean function is generated by an AND gate which can be programmed to form the AND of any subset of inputs or their complements. The product terms so produced can be summed up in an array of programmable OR gates. Thus, we have a programmable OR gate array at the output. The input and output gates are constructed in the form of arrays with input lines orthogonal to product lines and product lines orthogonal to output lines.

Figure 5.11 shows the internal architecture of a PLA device with four input lines, eight product lines and four output lines. That is, the programmable AND gate array has eight AND gates. Each of the AND gates here has eight inputs, corresponding to four input variables and their complements. The input to each of the AND gates can be programmed to be any of the possible 16 combinations of four input variables and their complements. Four OR gates at the output can generate four different Boolean functions, each having a maximum of eight minterms out of 16 minterms possible with four variables. The logic diagram depicts the unprogrammed state of the device. The internal architecture shown in Fig. 5.11 can also be represented by the schematic form of Fig. 5.12. PLAs usually have inverters at the output of OR gates to enable them to implement a given Boolean function in either AND-OR or AND-OR-INVERT form.

Figure 5.13 shows a generalized block schematic representation of a PLA device having n inputs, m outputs and k product terms, with n, m and k respectively representing the number of input variables, the number of OR gates and the number of AND gates. The number of inputs to each OR gate and each AND gate are k and 2n respectively.

A PLA is specified in terms of the number of inputs, the number of product terms and the number of outputs. As is clear from the description given in the preceding paragraph, the PLA would have a total of $2^{kn} + km$ programmable interconnections. A ROM with the same number of input and output lines would have $2n \times m$ programmable interconnections.

A PLA could be either mask programmable or field programmable. In the case of a mask-programmable PLA, the customer submits a program table to the manufacturer to produce a custom-made PLA having the desired internal paths between inputs and outputs. A field-programmable logic array (FPLA) is programmed by the users themselves by means of a hardware programmer unit available commercially.
Limitations of PLAs

PLAs come in various sizes:

Typical size is 16 inputs, 32 product terms, 8 outputs

Each AND gate has large fan-in this limits the number of inputs that can be provided in a PLA

16 inputs $3^{16} = \text{possible input combinations; only 32 permitted (since 32 AND gates) in a typical PLA}$

32 AND terms permitted large fan-in for OR gates as well this makes PLAs slower and slightly more expensive than some alternatives to be discussed shortly

8 outputs could have shared minterms, but not required Programmable Array Logic (PAL)

Also used to implement circuits in SOP form

The connections in the AND plane are programmable

The connections in the OR plane are NOT programmable
The PAL (Programmable Array Logic):
The PAL device is a PLD with a fixed OR array and a programmable AND array.
As only AND gates are programmable, the PAL device is easier to program but it is not as flexible as the PLA.

![Internal architecture of a PAL device](image)

The device shown in the figure has 4 inputs and 4 outputs. Each input has a buffer-inverter gate, and each output is generated by a fixed OR gate.

The device has 4 sections, each composed of a 3-wide AND-OR array, meaning that there are 3 programmable AND gates in each section.

Each AND gate has 10 programmable input connections indicating by 10 vertical lines intersecting each horizontal line. The horizontal line symbolizes the multiple input configuration of an AND gate.

One of the outputs $F_1$ is connected to a buffer-inverter gate and is fed back into the inputs of the AND gates through programmed connections.

Designing using a PAL device, the Boolean functions must be simplified to fit into each section.

The number of product terms in each section is fixed and if the number of terms in the function is too large, it may be necessary to use two or more sections to implement one Boolean function.
Example:

Implement the following Boolean functions using the PAL device as shown above:

\[ W(A, B, C, D) = \sum m(2, 12, 13) \]
\[ X(A, B, C, D) = \sum m(7, 8, 9, 10, 11, 12, 13, 14, 15) \]
\[ Y(A, B, C, D) = \sum m(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15) \]
\[ Z(A, B, C, D) = \sum m(1, 2, 8, 12, 13) \]

Simplifying the 4 functions to a minimum number of terms results in the following Boolean functions:

\[ W = ABC' + A'B'CD' \]
\[ X = A + BCD \]
\[ Y = A'B + CD + B'D' \]
\[ Z = ABC' + A'B'CD + AC'D' + A'B'C'D \]
\[ = W + AC'D' + A'B'C'D \]

Note that the function for \( Z \) has four product terms. The logical sum of two of these terms is equal to \( W \). Thus, by using \( W \), it is possible to reduce the number of terms for \( Z \) from four to three, so that the function can fit into the given PAL device.

The PAL programming table is similar to the table used for the PLA, except that only the inputs of the AND gates need to be programmed.

<table>
<thead>
<tr>
<th>Product term</th>
<th>AND Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>—</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>9</td>
<td>—</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>—</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The figure shows the connection map for the PAL device, as specified in the programming table.

Since both \( W \) and \( X \) have two product terms, third AND gate is not used. If all the inputs to this AND gate left intact, then its output will always be 0, because it receives both the true and complement of each input variable i.e., \( AA' = 0 \).
Comparing PALs and PLAs

PALs have the same limitations as PLAs (small number of allowed AND terms) plus they have a fixed OR plane less flexibility than PLAs.

PALs are simpler to manufacture, cheaper, and faster (better performance) PALs also often have extra circuitry connected to the output of each OR gate.

The OR gate plus this circuitry is called a *macrocell*.

Macrocell:
Macrocell Functions:

Enable = 0 can be used to allow the output pin for \( f \) to be used as an additional input pin to the PAL.

Enable = 1, Select = 0 is normal for typical PAL operation.

Enable = Select = 1 allows the PAL to synchronize the output changes with a clock pulse.

The feedback to the AND plane provides for multi-level design.

Multi-Level Design with PALs

\[
f = A'B'C + A'B'C' + ABC' + AB'C = A'g + Ag' \text{ where } g = BC + B'C' \text{ and } C = h \below{eq}
\]

Programming SPLDs

PLAs, PALs, and ROMs are also called SPLDs – Simple Programmable Logic Devices, SPLDs must be programmed so that the switches are in the correct places.

CAD tools are usually used to do this: A fuse map is created by the CAD tool and then that map is downloaded to the device via a special programming unit.

There are two basic types of programming techniques

1. Removable sockets on a PCB
2. In system programming (ISP) on a PCB

This approach is not very common for PLAs and PALs but it is quite common for more complex PLDs,

An SPLD Programming Unit: The SPLD is removed from the PCB, placed into the unit and programmed there

Removable SPLD Socket Package: PLCC (plastic-leaded chip carrier)
In System Programming (ISP):
- Used when the SPLD cannot be removed from the PCB
- A special cable and PCB connection are required to program the SPLD from an attached computer
- Very common approach to programming more complex PLDs like CPLDs, FPGAs, etc.

Programmable Interconnect Technologies:
The programmable features of every PLD, be it simple programmed logic devices (SPLDs) such as PLAs, PALs and GALs or complex programmable logic devices (CPLDs) or even field-programmable gate arrays (FPGAs), come from their programmable interconnect structure. Interconnect technologies that have evolved over the years for programming PLDs include fuses, EPROM or EEPROM floating gate transistors, static RAM and antifuses.

1. Fuse
A fuse is an electrical device that has a low initial resistance and is designed permanently to break an electrically conducting path when current through it exceeds a specified limit. It uses bipolar technology and is nonvolatile and one-time programmable. It was the first user-programmable switch developed for use in PLAs. They were earlier used in smaller PLDs and are now being rapidly replaced by newer technologies.

2. Floating-Gate Transistor Switch
This interconnect technology is based on the principle of placing a floating-gate transistor between two wires in such a way as to facilitate a WIRE-AND function. This concept is used in EPROM and EEPROM devices, and that is why the floating-gate transistor is sometimes referred to as an EPROM or EEPROM transistor. Figure shows the use of floating-gate transistor interconnects in the AND plane of a CPLD or SPLD. All those inputs that are required to be part of a particular product term are activated to drive the product wire to a logic ‘0’ level through the EPROM transistor. For inputs that are not part of the product term, relevant transistors are switched off.
This technology is commonly used in SPLDs and CPLDs. A floating-gate transistor based switch matrix, however, requires a large number of interconnects and therefore transistors. For example, a CPLD with 128 macrocells with four inputs and one output each would require as many as 65 536 interconnects for 100% routability. A large number of interconnects also adds to the propagation delay.
3. **Static RAM-Controlled Programmable Switches**

Static RAM (SRAM) is basically a semiconductor memory, and the word ‘static’ implies that it is a nonvolatile memory. That is, the memory retains its contents as long as power is on. A SRAM with \(m\) address lines and \(n\) data lines is referred to as a \(2^m \times n\) memory and is capable of storing \(2^m\ n\)-bit words. Figure 1 shows the basic SRAM cell comprising six MOSFET switches, with four of them connected as cross-coupled inverters. A basic SRAM cell can store one bit of information. The reading operation is carried out by precharging both the bit lines (BL and BL?) to logic ‘1’ and then asserting the WL line. The writing operation is done by giving the desired logic status to the BL line and its complement to the BL line and then asserting the WL line.

Figure 5.32 shows the use of SRAM-controlled switches. SRAMs are used to control not only the gate nodes but also the select inputs of multiplexers that drive the logic block inputs. The figure illustrates the routing scheme for feeding the output of one logic block to the input of another via SRAM-controlled pass transistor switches and a SRAM-controlled multiplexer. It may be mentioned here that a SRAM-controlled programmable interconnect matrix does not necessarily use both pass transistors and multiplexers. Whether it uses pass transistors or multiplexers or both is product specific.

4. **Antifuse**

An antifuse is an electrical device with a high initial resistance and is designed permanently to create an electrically conducting path typically when voltage across it exceeds a certain level. Antifuses use CMOS technology, which is one of the main reasons for their wide use in PLDs, FPGAs in particular. A typical antifuse consists of an insulating layer sandwiched between two conducting layers. In the unprogrammed state, the insulating layer isolates the top and bottom conducting layers. When programmed, the insulating layer is transformed into a low-resistance link. Typically, metal is used for conductors and amorphous silicon for the insulator. The application of high voltage across amorphous silicon permanently transforms it into a polycrystalline silicon–metal alloy having a low resistance. There are other antifuse structures too, such as that used in the Actel antifuse. This antifuse, known as PLICE, uses polysilicon and n+ diffusion as conductors and ONO as insulator. Figure shows the construction. This type of antifuse is usually triggered by a small current of the order of a few milliamperes. The high current density produced in the thin insulating layer produces heat, thus melting the insulating layer and creating an irreversible resistive silicon link.
CPLD

Complex Programmable Logic Devices (CPLD)

SPLDs (PLA, PAL) are limited in size due to the small number of input and output pins and the limited number of product terms.

Combined number of inputs + outputs < 32 or so CPLDs contain multiple circuit blocks on a single chip.

Each block is like a PAL: PAL-like block, Connections are provided between PAL-like blocks via an interconnection network that is programmable. Each block is connected to an I/O block as well.

Internal Structure of a PAL-like Block Includes macrocells, Usually about 16. Each Fixed OR planes: OR gates have fan-in between 5-20. XOR gates provide negation ability: XOR has a control input.
More on PAL-like Blocks:
CPLD pins are provided to control XOR, MUX, and tri-state gates.
When tri-state gate is disabled, the corresponding output pin can be used as an input pin.
The associated PAL-like block is then useless.
The AND plane and interconnection network are programmable. Commercial CPLDs have
between 2-100 PAL-like blocks.
Programming a CPLD: CPLDs have many pins – large ones have > 200, so removal of CPLD
from a PCB is difficult without breaking the pins instead. Use ISP (in system
programming) to program the CPLD by using JTAG (Joint Test Action Group) port used
to connect the CPLD to a computer.

Example CPLD: Use a CPLD to implement the function
\[ f = x_1x_2x_6' + x_1x_4x_5x_6' + x_2x_3x_7 + x_2x_4x_5x_7 \]
Field Programmable Gate Arrays (FPGAs):

The FPGA consists of 3 main structures:

1. Programmable logic structure,
2. Programmable routing structure, and
3. Programmable Input/output (I/O).

1. **Programmable logic structure**

The programmable logic structure FPGA consists of a 2-dimensional array of configurable logic blocks (CLBs).

Each CLB can be configured (programmed) to implement any Boolean function of its input variables. Typically CLBs have between 4-6 input variables. Functions of larger number of variables are implemented using more than one CLB. In addition, each CLB typically contains 1 or 2 FFs to allow implementation of sequential logic.

Large designs are partitioned and mapped to a number of CLBs with each CLB configured (programmed) to perform a particular function.

These CLBs are then connected together to fully implement the target design.

Connecting the CLBs is done using the FPGA programmable routing structure.

2. **Programmable routing structure:**

To allow for flexible interconnection of CLBs, FPGAs have 3 programmable routing resources:

1. Vertical and horizontal routing channels which consist of different length wires that can be connected together if needed. These channel run vertically and horizontally between columns and rows of CLBs as shown in the Figure.
2. Connection boxes, which are a set of programmable links that can connect input and output pins of the CLBs to wires of the vertical or the horizontal routing channels.
3. Switch boxes, located at the intersection of the vertical and horizontal channels. These are a set of programmable links that can connect wire segments in the horizontal and vertical channels.

3. Programmable I/O:
These are mainly buffers that can be configured either as input buffers, output buffers or input/output buffers.

They allow the pins of the FPGA chip to function either as input pins, output pins or input/output pins.

Field Programmable Gate Arrays (FPGA) can handle larger circuits, No AND/OR planes and provide logic blocks, I/O blocks, and interconnection wires and switches Logic blocks provide functionality. Interconnection switches allow logic blocks to be connected to each other and to the I/O pins.
LUTs: Logic blocks are implemented using a lookup table (LUT). LUT's contain a small number of inputs, one output. Contains storage cells that can be loaded with the desired values. A 2 input LUT uses 3 MUXes to implement any desired function of 2 variables.

“Shannon’s expansion at work!

$ f = x_1'x_2 + x_1x_2$, or using Shannon's expansion:

$ f = x_1'(x_2') + x_1(x_2) = x_1'(1' + 0) + x_1(1' + 1)$

3 Input LUT: 7 2x1 MUXes and 8 storage cells are required, Commercial LUTs have 4-5 inputs, and 16-32 storage cells.

Example FPGA

Use an FPGA with 2 input LUTS to implement the function $f = x_1x_2 + x_2'x_3$

$f_1 = x_1x_2$, $f_2 = x_2'x_3$, $f = f_1 + f_2$
Another Example FPGA

Use an FPGA with 2 input LUTS to implement the function \( f = x_1x_3x_6' + x_1x_4x_5x_6' + x_2x_3x_7 + x_2x_4x_5x_7 \)

Fan-in of expression is too large for FPGA (this was simple to do in a CPLD)

Factor \( f \) to get sub-expressions with max fan-in = 2

\[ f = x_1x_6'(x_3 + x_4x_5) + x_2x_7(x_3 + x_4x_5) = (x_1x_6' + x_2x_7)(x_3 + x_4x_5) \]

Could use Shannon's expansion instead, Goal is to build expressions out of 2-input LUTs

**FPGA Implementation:** \( f = (x_1x_6' + x_2x_7)(x_3 + x_4x_5) \)
**Standard Cells:**
Rows of logic gates can be connected by wires in the *routing channels.* Designers (via CAD tools) select prefab gates from a library and place them in rows. Interconnections are made by wires in routing channels. Multiple layers may be used to avoid short circuiting. A hard-wired connection between layers is called a *via.*

![Standard Cells Diagram]

**Example: Standard Cells**
\[ f_1 = x_1 x_2 + x_1' x_2' x_3 + x_1 x_3', \quad f_2 = x_1 x_2 + x_1' x_2' x_3 + x_1 x_3 \]

**Sea of Gates Gate Array:**
A Sea of Gates gate array is just like a standard cell except all gates are of the same type. Interconnections are run in channels and use multiple layers, Cheaper to manufacture due to regularity.

![Sea of Gates Gate Array Diagram]
PAL - Example PAL16L8
Testing:
Design of logic integrated circuits in CMOS technology is becoming more and more complex since VLSI is the interest of many electronic IC users and manufacturers. A common problem to be solved by both users and manufacturers is the testing of these ICs. Testing can be expressed by checking if the outputs of a functional system (functional block, Integrated Circuit, Printed Circuit Board or a complete system) correspond to the inputs applied to it. If the test of this functional system is positive, then the system is good for use. If the outputs are different than expected, then the system has a problem: so either the system is rejected (Go/No Go test), or a diagnosis is applied to it, in order to point out and probably eliminate the problem’s causes.

After chips have been made, they must be tested (usually before cutting up the wafers and putting the chips in packages). These test patterns can usefully be derived from the test data used when the circuit was simulated previously. However, it becomes even more important that the test patterns should exercise the circuit fully. This has resulted in testers that compare a newly made chip against a known gold chip which has been characterized previously.

Why We Need Testing?
- Not all die on a wafer operate correctly due to ic manufacturing process.
- The aim of testing is to determine which parts of dies are good.
- Testing a die (chip) can occur at and at malfunction detection cost of
  - Wafer level(0.1$)
  - Packaged chip level(1$)
  - Board level(10$)
  - System level(100$)
  - In the field(1000$)
- Manufacturing cost is very low when malfunctioning chip fault is detected at early level.

Roles of Testing
Detection: Determination of whether or not the device under test (DUT) has some fault.
Diagnosis: Identification of a specific fault that is present on DUT.
Device characterization: Determination and correction of errors in design and/or test procedure.
Failure mode analysis (FMA): Determination of manufacturing process errors that may have caused defects on the DUT.

Testing principle
Apply inputs and compare “outputs” with the “expected outputs”
Testing is applied to detect faults after several operations: design, manufacturing, packaging and especially during the active life of a system, and thus since failures caused by wear-out can occur at any moment of its usage. Design for Testability (DfT) is the ability of simplifying the test of any system. DfT could be synthesized by a set of techniques and design guidelines where the goals are:
- minimizing costs of system production
- minimizing system test complexity: test generation and application
- improving quality
- Avoiding problems of timing discordance or block nature incompatibility.

The Rule of Ten
In the production process cycle, a fault can occur at the chip level. If a test strategy is considered at the beginning of the design, then the fault could be detected rapidly, located and eliminated at a very low cost. When the faulty chip is soldered on a printed circuit board, the cost of fault remedy would be multiplied by ten. And this cost factors continues to apply until the system has been assembled and packaged and then sent to users.

Cost of replacing defective component increases by an order of magnitude with each stage of manufacture.

If you don’t test it, it won’t work!
Levels of testing
- Chip
- Board
- System
  - Boards put together
  - System-on-Chip (SoC)
  - System in field

Classification of tests
A test is a procedure which allows one to distinguish between good and bad parts. Tests can be classified according to the technology they are designed for, the parameters they measure, the purpose for which the test results are used, and the test application method. The type of tests to be performed depends heavily on the technology of the circuit to be tested: analog, digital, or mixed-signal.

Digital circuits have the property that the domain of values of the input and output signals is binary (usually referred to as digital); i.e., the signals can only take on the value ‘logic 0’ or ‘logic 1’.
Functional Testing versus manufacturing Testing

IC test is comprised of two primary approaches: functional testing and structural (MANUFACTURING TEST) testing.

There are two basic forms of validation

**Functional test:**
Does this chip design produce the correct results?

Functional test seeks logical correctness, several year effort, up to 50 people, to ensure that the design is good. Ferociously expensive, Often a software approach. But, may comprise: Scanning Electron Microscope tests, Bright-Lite detection of defects, Electron beam testing, Artificial intelligence (expert system) methods, repeated functional tests

Functional testing verifies that the circuit performs as it is designed to perform. For example, if we assume that the design is an adder circuit. Functional testing verifies that this circuit performs the addition function and computes the correct results over the range of values tested. However, exhaustive testing of all possible input combinations grows exponentially as the number of inputs increases. To maintain a reasonable test time, one needs to focus functional test patterns on the general function and corner cases.

**Manufacturing test:**
Does this particular die work? Can I sell it?

Manufacturing test is done on each die prior to market release, Send your parts through a burn-in oven and a tester before selling them. The distinction is in the testing, not in the problem. Determines if manufactured chip meets specs, Must cover high % of modeled faults, Must minimize test time (to control cost), No fault diagnosis, Tests every device on chip, Tests are functional or at speed of application or speed guaranteed by supplier

- Verify the physical operation of every gate in the chip.
  - It is done during chip fabrication so that functional correctness is assumed.
  - Typical manufacturing defects are

- Layer to layer shorts. Shorts to VDD or VSS, shorts between to nodes.
- Discontinuous wires. Floating inputs, disconnected outputs.
- Thin-oxide shorts to substrate. Shorts to VDD or GND.
  - I/O integrity tests
- Level tests: Noise margins for the TTL, ECL, CMOS etc. I/O pads.
- Speed test
- IDD test

Critical factor is to incorporate methods of testing circuits.
Consider a combinational logic with n inputs. A sequence of 2n inputs (test vectors) are required to exhaustively test the logic. This may take years if n is large.
Fault types and Models

Faults
System -- Mixed electronic, electromechanical, chemical, and photonic system (MEMS technology).
Failure: Incorrect or interrupted system behavior
Error: Manifestation of fault in system
Fault: Physical difference between good & bad system behavior

Fault types:
Permanent: System is broken and stays broken the same way indefinitely
Transient: Fault temporarily affects the system behavior, and then the system reverts to the good machine -- time dependency, caused by environmental condition
Intermittent: Sometimes causes a failure, sometimes does not.

Failure Mechanisms
Permanent faults:
Missing/Added Electrical Connection Broken Component (IC mask defect or silicon-to-metal connection) Burnt-out Chip Wire Corroded connection between chip & package Chip logic error (Pentium division bug OR Pentium FDIV bug)

Transient Faults:
Cosmic Ray_ An a particle (ionized Helium atom)_ Air pollution (causes wire short/open)_ Humidity (temporary short)_ Temperature (temporary logic error)_ Pressure (temporary wire open/short)_ Vibration (temporary wire open)_ Power Supply Fluctuation (logic error)_ Electromagnetic Interference (coupling)_ Static Electrical Discharge (change state)_ Ground Loop (misinterpreted logic value)

Intermittent Faults:
Loose Connections_ Aging Components (changed logic delays)_ Hazards and Races in critical timing paths (bad design)_ Resistor, Capacitor, Inductor variances (timing faults)_ Physical Irregularities (narrow wire – high resistance)_ Electrical Noise (memory state changes)

Fault Modeling
In order to deal with the existence of good and bad parts it is necessary to propose a “fault model”, i.e. a model for how faults occur and their impact on circuits. Chip testing is usually multipurpose and attempts to detect faults in fabrication, design, and failures due to stressful operating conditions, i.e. reliability problems. The input vectors are applied to devise under test (DUT) and circuit under test (CUT) as its stimuli. Its output are measured and compared with the desired outputs and if they match then quality of DUT or CUT is determined. Mainly the faults are caused due to physical defects. The examples of physical defects are:
- Defects in silicon substrate
- Photolithographic defects
- Mask contaminated and scratches
- Process variations and abnormalities
- Oxide defects.
The physical defects can cause electrical faults and logical faults.

The **electrical faults** include: Shorts (bridging faults) & opens—Transistor stuck-on, stuck-open—Resistive shorts and opens—Excessive change in threshold voltage—Excessive steady-state currents

The **logical faults** include: Logical struck-at-o or struck-at-1—Slower transition—AND-bridging, OR bridging

- **Electrical Faults**
  
  **Stuck-On, Stuck-Open**

  ![Stuck-On, Stuck-Open Diagram](image)

  **Bridging Faults**

  1) Two circuit nodes shorted together

  2) Usually assumed to be a low resistance path (hard short)

  3) Three classes are typically considered:
     
     i. Bridging within a logic element (transistor gates, sources, or drains shorted together)

     ii. Bridging between logic nodes (i.e. inputs or outputs of logic elements) without feedback

     iii. Bridging between logic nodes with feedback

  4) Typically not considered is bridging of non-logical nodes between logic elements (transistor shorts across logic elements)

- **Advantages**

  1) Covers a large percentage of physical defects - some research indicates that bridging faults account for up to 30% of all defects

- **Disadvantages**

  1) ATPG algorithms are more complex - testing requires setting the two bridged nodes to opposite values and observing the effect

  2) Requires a lower level circuit description for bridging faults within logic elements

- **Logical Faults**

  **Logical struck-at-o or struck-at-1**

  The most popular model is called the “Struck-at” model. With this model, a faulty gate input is modeled as a “struck at 0” or “stuck at 1”.

  These faults are of two types:

  1) Single Stuck-at Fault Model &

  2) Multiple Stuck-at Fault Model.
Single Stuck-at Fault Model:

Fault-Free Gate

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Fault: A s-a-l

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Assumptions

1) Only one line in the circuit is faulty at a time
2) The fault is permanent (as opposed to transient)
3) The effect of the fault is as if the faulty node is tied to either Vcc (s-a-1), or Gnd (s-a-0)
4) The function of the gates in the circuit is unaffected by the fault

Advantages

1) Can be applied at the logic level or module level
2) Reasonable numbers of faults 2n (n=number of circuit nodes)
3) Algorithms for automatic test pattern generation (ATPG) and faults simulation are well developed and efficient
4) Research indicates that the single stuck-at fault model covers about 90% of the possible manufacturing defects in CMOS circuits
5) Source-drain shorts, oxide pinholes, missing features, diffusion contaminants, metallization shorts, etc.
6) Other useful fault models (stuck-open, bridging faults) can be mapped into (sequences of) stuck-at faults

Disadvantages

1) Does not cover all defects in CMOS or other devices

Multiple Stuck-at Fault Model:

Assumptions:

1) Same as single stuck-at faults except:
2) 2 or more lines in the circuit can be faulty at the same time

Advantage

1) If used in conjunction with single stuck-at faults, it covers a greater percentage of physical defects

Disadvantages

1) Large number of faults 3n-1 (n=number of circuit nodes)
2) Algorithms for ATPG and fault simulation are much more complex and not as well developed
3) Does not cover a significantly larger number of detects that single stuck-at faults
Design Strategies for Test:
• Design for Testability (DFT)
Controllability and observability are the two key concepts for designing circuits that are testable.
• Controllability is the ability to set every internal node to logic-1 and logic-0.
• Observability is the ability to observe either directly or indirectly the state of any node in the circuit.
Controllability and observability together with predictability are the most important factors that determine the complexity of deriving a test set for a circuit. Controllability is the ability to establish a specific signal value at each node in a circuit by setting values on the circuit’s inputs. Observability, on the other hand, is the ability to determine the signal value at any node in a circuit by controlling the circuit’s inputs and observing its outputs. DFT techniques, used to improve a circuit’s controllability and observability, can be divided into three major categories:

Three main approaches to Design For Testability
• Ad-hoc (based on experience)
• Scan based
• Self-test and Built-in test

Ad-Hoc Testing
Aim is to reduce complexity of testing. Common techniques are;
• Partitioning large sequential circuits
• Adding test points
• Adding multiplexers to provide alternative signal paths
• Providing for easy state reset
The techniques involve tricks that are developed over the years to avoid the overheads of systematic approach to testing.
Example: Consider a long counter (n-bit). Possible ad-hoc techniques are;
• Add parallel-load feature to test carry propagation.
• Reduce counter length to k bits, so that (n/k) 2k output vectors to test instead of 2n output vectors.
Example: In a bus oriented system, the bus can be used for observing and controlling the internal nodes for testing.

Scan-Based Test Techniques
Level Sensitive Scan Design (LSSD)
• In this technique, every input combinational logic may be controlled and every output may be observed. Also, running a serial sequence of “110010” through scan registers can test these registers. The disadvantage is the complexity of the scan registers that impacts IC performance.
  ➢ Serial Scan
  ➢ Partial Serial Scan
  ➢ Parallel Scan
BUILT-IN SELF-TEST
The main idea behind a BIST approach is to eliminate the need for the external tester by integrating active test infrastructure onto the chip. A typical BIST architecture consists of a test pattern generator (TPG), usually implemented as a linear feedback shift register (LFSR), a test response analyzer (TRA), implemented as a multiple input shift register (MISR), and a BIST control unit (BCU), all implemented on the chip (Figure 1). This approach allows applying at-speed tests and eliminates the need for an external tester. Furthermore, the BIST approach is also one of the most appropriate techniques for testing complex Systems on chip, as every core in the system can be tested independently from the rest of the system. It reduces test pattern generation cost, volume of test data and test time.

Fig. 1  A typical BIST architecture

PSEUDO RANDOM PATTERN GENERATOR
To test the circuit, test patterns first have to be generated either by using a pseudo random generator, a weighted test generator, an adaptive test generator, or other means. A pseudo random test generator circuit can use a linear feedback shift register (LSFR) as shown in Figure 2.

Fig. 2  A pseudo random sequence generator using LFSR
LFSR AS AN OUTPUT RESPONSE ANALYSER (ORA):
To reduce the chip area penalty, data compression schemes are used to compare the compacted test responses instead of the entire raw test data. Signature analysis is a popular data compression scheme based on the concept of cyclic redundancy checking. It uses polynomial division, which divides the polynomial representation of the test output data by a characteristic polynomials and then finds the remainder as the signature. The signature is then compared with the expected signature to determine whether the device under test is faulty. Compression causes some loss of fault coverage. It is possible that the output of a faulty circuit can match the output of a fault-free circuit, thus, the fault can go undetected in the signature analysis. Such a phenomenon is called aliasing.

The above circuit is a single-input linear feedback shift register (LFSR) in which all the latches are edge-triggered. In this case, the signature is the content of this register after the last input bit has been sampled. The input sequence \( \{an\} \) is represented by polynomial \( G(x) \) and the output sequence by \( Q(x) \). It can be shown that \( G(x) = Q(x)P(x) + R(x) \), where \( P(x) \) is the characteristic polynomial of LFSR and \( R(x) \) is the remainder, the degree of which is lower than that of \( P(x) \). For the above case characteristic polynomial is \( P(x) = 1 + x^2 + x^4 + x^5 \).

For the 8-bit input sequence \( \{1 1 1 1 0 1 0 1\} \), the corresponding input polynomial is \( G(x) = x^7 + x^6 + x^5 + x^4 + x^2 + 1 \)

And the remainder term becomes \( R(x) = x^4 + x^2 \), which corresponds to the register contents of \( \{0 0 1 0 1\} \).

Fig. 3 Combine scan with PRSG and signature analysis

OUTPUT RESPONSE ANALYSER:
Another simple method is to compare the outputs of two identical circuits for the same input, with one of them regarded as reference. However, if both circuits have the same faults, their outputs can still match. Also a checker circuit is inserted such that the checker generates and sends out a signal when on-line fault occurs. The use of self-checking circuits simplifies the development of software diagnostic programs. A disadvantage is that additional hardware is required and the checker itself needs to have self-checking capability.
BUILT-IN LOGIC BLOCK OBSERVER:
The built-in logic block observer (BILBO) register is a form of ORA which can be used in each cluster of partitioned registers. The BILBO operation allows monitoring of circuit operation through exclusive-OR ing into LSFR at multiple points, which corresponds to the signature analyser with multiple inputs.

Fig. 4 BILBO application

• IDDQ Testing
It is a method of testing for bridging faults. VDD current is monitored. Static CMOS logic gate does not draw DC current (except leakage). But, when bridging fault occurs, a measurable DC IDD flows. Current measuring is a slow process, hence test time increases.

Automatic test pattern generation (ATPG) attempts to analyze the logic of a circuit and, in particular, the access paths to observe and control internal state, and formulate test stimuli that should give a high fault coverage.
The main difficulty in testability lies in internal state registers, and one solution is to make these easily accessible by connecting them into a long shift register. The chip can then run in a number of modes:
  _ Normal – the registers are disconnected from the shift register.
  _ Shift – the shift register steps along.
  _ Read – the state registers are copied into the shift register.
  _ Write – the shift register is copied into the state registers.

ATPG Systems
  ➢ Increase fault coverage
  ➢ Reduce overall effort (CPU time)
  ➢ Fewer test vectors (test application time)

Testability Measures
  ➢ A powerful heuristic used during test generation (more on its uses in later slides)

Reduce cost of fault simulation
  ➢ Fault list reduction
  ➢ Efficient and diverse fault simulation methods
  ➢ suited for specific applications and environment
  ➢ Fault sampling method
The process of generating the test patterns on a design via a CAD (Computer Aided Design) tool is called ATPG or Automatic Test Pattern Generation. Test patterns, sometimes called test vectors, are sets of 1s and 0s placed on primary input pins during the manufacturing test process to determine if the chip is functioning properly. When the test pattern is applied, the Automatic Test Equipment (ATE) determines if the circuit is free from manufacturing defects by comparing the fault-free output which is also contained in the test pattern with the actual output measured by the ATE.

The goal of ATPG is to create a set of patterns that achieves a given test coverage, where test coverage is the total percentage of testable faults the pattern set actually detects. The ATPG run itself consists of two main steps i.e generating patterns and performing fault simulation to determine which faults the patterns detect [17]. The two most typical methods for pattern generation are random and deterministic. Additionally, the ATPG tools can fault simulate patterns from an external set and place those patterns detecting faults in a test set.

**Random Test Pattern Generation**
An ATPG tool uses random pattern test generation when it produces a number of random patterns and identifies only those patterns necessary to detect faults. It then stores only those patterns in the test pattern set. The type of fault simulation used in random pattern test generation cannot replace deterministic test generation because it can never identify redundant faults [16]. Nor can it create test patterns for faults that have a very low probability of detection. However, it can be useful on testable faults aborted by deterministic test generation. Using a small number of random patterns as the initial ATPG step can improve ATPG performance.

**Deterministic Test Pattern Generation**
An ATPG tool uses deterministic test pattern generation when it creates a test pattern intended to detect a given fault. The procedure is to pick a fault from the fault list, create a pattern to detect the fault, fault simulate the pattern, and check to make sure the pattern detects the fault. More specifically, the tool assigns a set of values to control points that force the fault site to the state opposite the fault-free state, so there is a detectable difference between the fault value and the fault-free value. The tool must then find a way to propagate this difference to a point where it can observe the fault effect. To satisfy the conditions necessary to create a test pattern, the test generation process makes intelligent decisions on how best to place a desired value on a gate. If a conflict prevents the placing of those values on the gate, the tool refines those decisions as it attempts to find a successful test pattern. If the tool exhausts all possible choices without finding a successful test pattern, it must perform further analysis before classifying the fault. Faults requiring this analysis include redundant, ATPG-untestable, and possible-detected-untestable categories. Identifying these fault types is an important by-product of deterministic test generation and is critical to achieving high test coverage. For example, if a fault is proven redundant, the tool may safely mark it as untestable. Otherwise, it is classified as a potentially detectable fault and counts as an untested fault when calculating test coverage.
External Test Pattern Generation
An ATPG tool uses external pattern test generation when the preliminary source of ATPG is a preexisting set of external patterns that already exists. The tool analyzes this external pattern set to determine which patterns detect faults from the active fault list. It then places these effective patterns into an internal test pattern set [17]. The “generated patterns”, in this case, include the patterns (selected from the external set) that can efficiently obtain the highest test coverage for the design.

Chip-Level Test Techniques
• Testing approach to the main types of circuit structure;– Regular logic arrays
• Partial Serial Scan or Parallel Scan–Memories
• Self-Testing– Random Logic
• Full Serial Scan or Parallel Scan

System-Level Test Techniques
• At the board level– Bed-of-nails are used.
• At the chassis level– Software programs are used
• Complex boards– Scan-based test techniques – IEEE 1149 standard (Boundary Scan) is used to test multiple boards and IC’s.

Boundary Scan (IEEE 1149 standard)– IC has a serial scan path through its I/O pins.
– At the board level, IC’s are connected in a variety of series and parallel combinations. Then the following types of tests can be done;
• Connectivity tests among components
• Sampling and setting chip I/O’s
• Distribution and collection of self-test or built-in test results.
– IC has a Test Access Port (TAP). The port has the pins;
• TCK : Test Clock Input
• TMS : Test Mode Select
• TDI : Test Data Input
• TDO : Test Data Output
• TRST : Test Reset  Signal (Optional)

Boundary Scan (IEEE 1149 standard)
– The test circuitry is composed of
• TAP pins

I/O Pad and Boundary
Scan Cell
• Test-data registers
• Instruction register
• TAP controller