Lecture (1)
Introduction to FPGA

1. The History of Programmable Logic

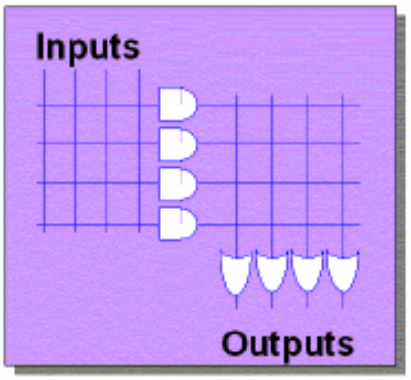
By the late 1970s, standard logic devices were all the rage, and printed circuit boards were loaded with them. Then someone asked, “What if we gave designers the ability to implement different interconnections in a bigger device?” This would allow designers to integrate many standard logic devices into one part.

To offer the ultimate in design flexibility, Ron Cline from Signetics™ (which was later purchased by Philips and then eventually Xilinx) came up with the idea of two programmable planes. These two planes provided any combination of “AND” and “OR” gates, as well as sharing of AND terms across multiple Ors.

This architecture was very flexible, but at the time wafer geometries of 10 µm made the input-to-output delay (or propagation delay) high, which made the devices relatively slow.

1.1 What is a CPLD?

1.1.1 Programmable Logic Array (PLA)



- Two programmable planes
- Any combination of ANDs/Ors
- Sharing of AND terms across multiple OR’s
- Highest logic density available to user
- High fuse count, slower than PALs

MMI (later purchased by AMD™) was enlisted as a second source for the PLA array. After fabrication issues, it was modified to become the programmable array logic (PAL) architecture by fixing one of the programmable planes.
This new architecture differed from that of the PLA in that one of the Programmable planes was fixed – the OR array. PAL architecture also had the added benefit of faster Time of Propagation Delay (Tpd) and less complex software, but without the flexibility of the PLA structure.

Other architectures followed, such as the PLD. This category of devices is often called Simple PLD.

1.1.2 Programmable Array Logic (PAL)

- One programmable plane – AND/Fixed OR
- Finite combination of ANDs/Ors
- Medium logic density available to user
- Lower fuse count, faster than PLAs (at this time fabricated on a 10 um process)

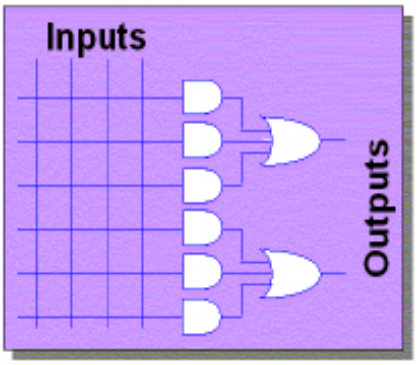


Figure 1.2 Programmable Array Logic (PAL)

The architecture had a mesh of horizontal and vertical interconnect tracks. At each junction was a fuse. With the aid of software tools, designers could select which junctions would not be connected by "blowing" all unwanted fuses. (This was done by a device programmer, but more commonly these days is achieved with In System Programming (ISP)).
Input pins were connected to the vertical interconnect. The horizontal tracks were connected to AND-OR gates, also called "product terms". These in turn connected to dedicated flip-flops, whose outputs were connected to output pins.
PLDs provided as much as 50 times more gates in a single package than discrete logic devices! This was a huge improvement, not to mention fewer devices needed in inventory and a higher reliability over standard logic.
PLD technology has moved on from the early days with companies such as Xilinx producing ultra-low-power CMOS devices based on flash memory technology. Flash PLDs provide the ability to program the devices time and time again, electrically programming and *erasing* the device. Gone are the days of erasing for more than 20 minutes under an UV eraser.

1.1.3 Complex Programmable Logic Devices (CPLDs)

Complex programmable logic devices (CPLDs) extend the density of SPLDs.
The concept is to have a few PLD blocks or macrocells on a single device with a general-purpose interconnect in-between. Simple logic paths can be implemented within a single block. More sophisticated logic requires multiple blocks and uses the general-purpose interconnect in-between to make these connections.

1.1.3.1 CPLD Architecture

- Central, Global Interconnect
- Simple, Deterministic Timing
- Easily routed
- PLD Tools and only interconnect
- Wide, fast complex gating

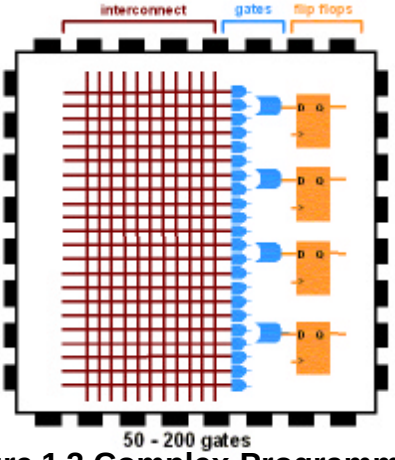


Figure 1.3 Complex Programmable Logic Device (CPLD)

CPLDs are great at handling wide and complex gating at blistering speeds – 5 nanoseconds, for example, which is equivalent to 200 MHz.
The timing model for CPLDs is easy to calculate so before starting your design you can calculate your input-to-output speeds.

1.2 Why use a CPLD?

CPLDs enable ease of design, lower development costs, and more product revenue for your money, and the opportunity to speed your products to market.

1.2.1 Ease of Design: CPLDs offer the simplest way to implement a design. Once a design has been described, by schematic and/or HDL entry, you simply use CPLD development tools to optimize, fit, and simulate the design.
The development tools create a file that is used to customize (that is, program) a standard off-the-shelf CPLD with the desired functionality. This pro- vides an instant hardware prototype and allows the debugging process to begin.
If modifications are needed, you can enter design changes into the CPLD development tool, and re-implement and test the design immediately.

1.2.2 Lower Development Costs: CPLDs offer very low development costs. Because CPLDs are re-programmable, you can easily and very inexpensively change your designs. This allows you to optimize your designs and continue to add new features to enhance your products.
CPLD development tools are relatively inexpensive (or in the case of Xilinx, free). Traditionally, designers have had to face large cost penalties such as rework, scrap, and development time. With CPLDs, you have flexible solutions, thus avoiding many traditional design pitfalls.

1.2.3 More Product Revenue: CPLDs offer very short development cycles, which means your products get to market quicker and begin generating revenue sooner. Because CPLDs are re-programmable, products can be easily modified using ISP over the Internet. This in turn allows you to easily introduce additional features and quickly generate new revenue. (This also results in an expanded time for revenue).
Thousands of designers are already using CPLDs to get to market quicker and stay in the market longer by continuing to enhance their products even after they have been introduced into the field. CPLDs decrease TTM and extend TIM.

1.2.4 Reduced Board Area: CPLDs offer a high level of integration (that is, a large number of system gates per area) and are available in very small form factor packages.
This provides the perfect solution for designers whose products which must fit into small enclosures or who have a limited amount of circuit board space to implement the logic design.

1.2.5 Cost of Ownership: Cost of Ownership can be defined as the amount it costs to maintain, fix, or warranty a product.
For instance, if a design change requiring hardware rework must be made to a few prototypes, the cost might be relatively small. However, as the number of units that must be changed increases, the cost can become enormous.

Because CPLDs are re-programmable, requiring no hardware rework, it costs much less to make changes to designs implemented using them. Therefore cost of ownership is dramatically reduced.

Don't forget that the ease or difficulty of design changes can also affect opportunity costs. Engineers who spend time fixing old designs could be working on introducing new products and features ahead of the competition.

There are also costs associated with inventory and reliability. PLDs can reduce inventory costs by replacing standard discrete logic devices. Standard logic has a predefined function. In a typical design, lots of different types have to be purchased and stocked. If the design is changed, there may be excess stock of superfluous devices. This issue can be alleviated by using PLDs. You only need to stock one device; if your design changes, you simply reprogram. By utilizing one device instead of many, your board reliability will increase by only picking and placing one device instead of many.

Reliability can a lso be increased by using ultra-low-power CoolRunner CPLDs. Their lower heat dissipation and lower power operation leads to decreased FIT.

1.3 Field Programmable Gate Arrays (FPGAs)

In 1985, a company called Xilinx introduced a completely new idea: combine the user control and time to market of PLDs with the densities and cost benefits of gate arrays. Customers liked it – and the FPGA was born. Today Xilinx is still the number-one FPGA vendor in the world.

An FPGA is a regular structure of logic cells (or modules) and interconnect, which is under your complete control. This means that you can design, program, and make changes to your circuit whenever you wish.

With FPGAs now exceeding the 10 million gate limit (the Xilinx Virtex™-II FPGA is the current record holder), you can really dream big.

1.3.1 FPGA Architecture

- Channel Based Routing
- Post Layout Timing
- Tools More Complex than CPLDs
- Fine Grained
- Fast register pipelining

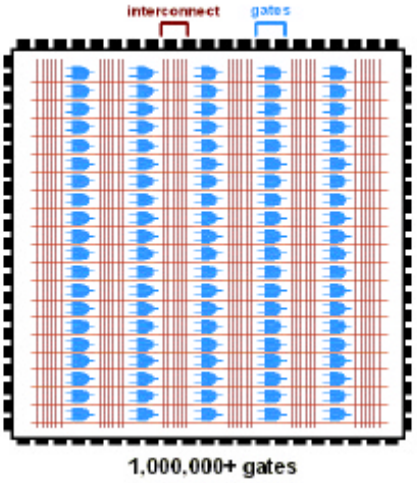


Figure 1.4 Field Programmable Gate Array Logic (FPGA)

There are two basic types of FPGAs: SRAM-based reprogrammable (Multi-time Programmed MTP) and (One Time Programmed) OTP. These two types of FPGAs differ in the implementation of the logic cell and the mechanism used to make connections in the device.

The dominant type of FPGA is SRAM-based and can be reprogrammed as often as you choose. In fact, an SRAM FPGA is reprogrammed every time it's powered up, because the FPGA is really a fancy memory chip. That's why you need a serial PROM or system memory with every SRAM FPGA.

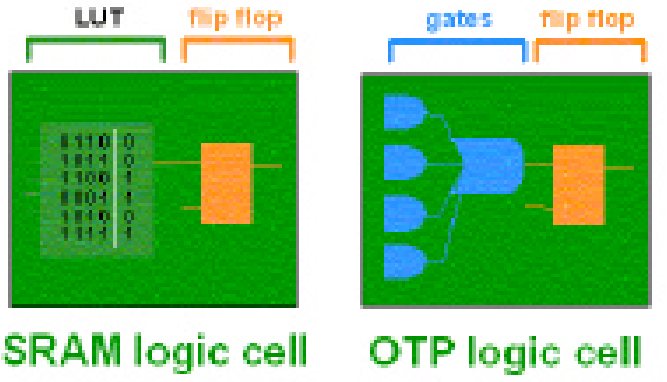


Figure 1.5 Types of FPGA

In the SRAM logic cell, instead of conventional gates, an LUT determines the output based on the values of the inputs. (In the "SRAM logic cell" diagram above, six different combinations of the four inputs determine the values of the output.) SRAM bits are also used to make connections.

OTP FPGAs use anti-fuses (contrary to fuses, connections are made, not "blown," during programming) to make permanent connections in the chip. Thus, OTP FPGAs do not require SPROM or other means to download the program to the FPGA. However, every time you make a design change, you must throw away the chip! The OTP logic cell is very similar to PLDs, with dedicated gates and flip-flops.

Table 1.1: Comparison between OTP FPGA and MTP FPGA

Property	OTP FPGA	MTP FPGA
Speed	Higher (current flows in wire)	Lower (current flows in transistors)
Size	smaller	Larger
Power Consumption	Lower	Higher
Working Environment (Radiation)	Radiation hardened	NO radiation hardened
Design Cycle	Programmed once only	Many times
Price	Almost the same	Almost the same
Reliability	More (single Chip)	Less (2 Chips, FPGA & PROM)
Security	More secure	Less secure

2. ASIC & FPGA Devices

Standard “off-the-shelf” integrated circuits have a mixed functional operation defined by the chip manufacturer. Contrary to this, both FPGAs & ASIC are types of integrated circuits whose function is not fixed by the manufacturer. The function is defined by the designer for a particular application. An ASIC requires a final manufacturing process to customize its operation while an FPGA does not.

2.1 ASICs

An Application Specific Integrated Circuit is a device that is partially manufactured by an ASIC vendor in generic form. This initial manufacturing process is the most complex, time consuming and expensive part of the total manufacturing process. The result is silicon chips with an array of unconnected transistors.

The final manufacturing process of connecting the transistors together is then completed when a chip designer has a specific design he or she wishes to implement in the ASIC. An ASIC vendor can usually do this in a couple of weeks and is known as the turn-round time. There are two categories of ASIC devices: Gate Array and standard cells.

2.1.1 Gate Array

There are two types of gate array: a channeled gate array and a channel-less gate array. A channeled gate array is manufactured with single or double rows of basic cells across the silicon. A basic cell consists of numbers of transistors. The channels between the rows of cells are used for interconnecting the basic cells during the final customization process. A channel-less gate array is manufactured with a “sea” of basic cells across the silicon and there are no dedicated channels for interconnections.
The library of cells provided by a gate array vendor will contain: primitive logic gates, registers, hard-macros, soft-macros. Hard-macros and soft-macros are usually of MSI and LSI complexity such as multiplexers, comparators and counters. Hard-macros are defined by the manufacturer while soft-macros are defined by the designer (e.g. specifying the width of a particular counter).

2.1.2 Standard Cells

Standard cell devices do not have the concept of a basic cell and no components are prefabricated on the silicon chip. The manufacturer creates custom masks for every stage of the device’s process and means silicon is utilized much more efficiently than for gate arrays.
Manufacturer supplies hard-macro and soft-macro libraries containing elements of LSI and VLSI complexity, such as controllers, ALUs and microprocessors. Additionally, soft-macro libraries contain RAM functions that cannot be implemented efficiently in gate array devices; ROM functions are more efficiently implemented in cell primitives.

2.2 FPGAs

The Field Programmable Gate Array is a device that is completely manufactured, but that remains design independent. Each FPGA vendor manufactures devices to a proprietary architecture. However the architecture will include a number of programmable logic blocks that are connected to programmable switching matrices. To configure a device to a particular functional operation these switching matrices are programmed to route signals between the individual logic blocks.

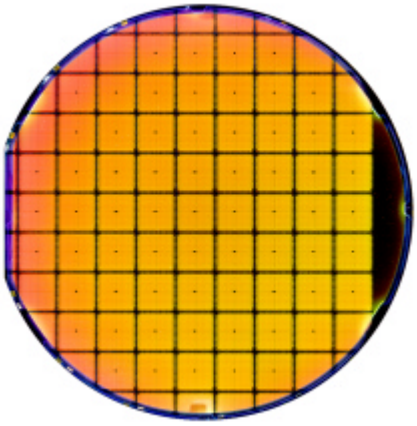


Figure 1.6 Silicon Wafer containing 10,000 gate FPGAs

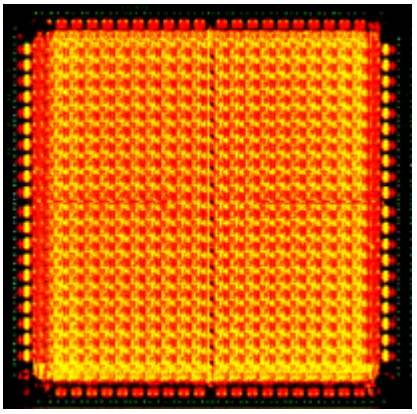


Figure 1.7 Single FPGA Die

Table 1.2: Comparison between FPGAs and ASICs

Property	FPGAs	ASICs
Digital and Analog Capability	Digital Only	Digital and Analog
Size	Larger	More Smaller
Operating Frequency	Lower (up to 400 MHz)	Higher (up to 3 GHz)
Power Consumption	Higher	Lower
Design Cycle	Very Small (few mins)	Very Long (about 12 wks)
Mass Production	Higher Price	Lower Price (more suitable)
Security	More Secure	Not Secure

3. The Internal Structure of FPGA

A typical FPGA is composed of three major components: logic modules, routing resources, and input/output (I/O modules) Figure 1.8 depicts the conceptual FPGA model. In an FPGA, an array of logic modules is surrounded or overlapped by general routing resources bounded by I/O modules. The logic modules contain combinational and sequential circuits that implement logic functions. The routing resources comprise pre-fabricated wire segments and programmable switches. The interconnections between the logic modules and the I/O modules are user-programmable.

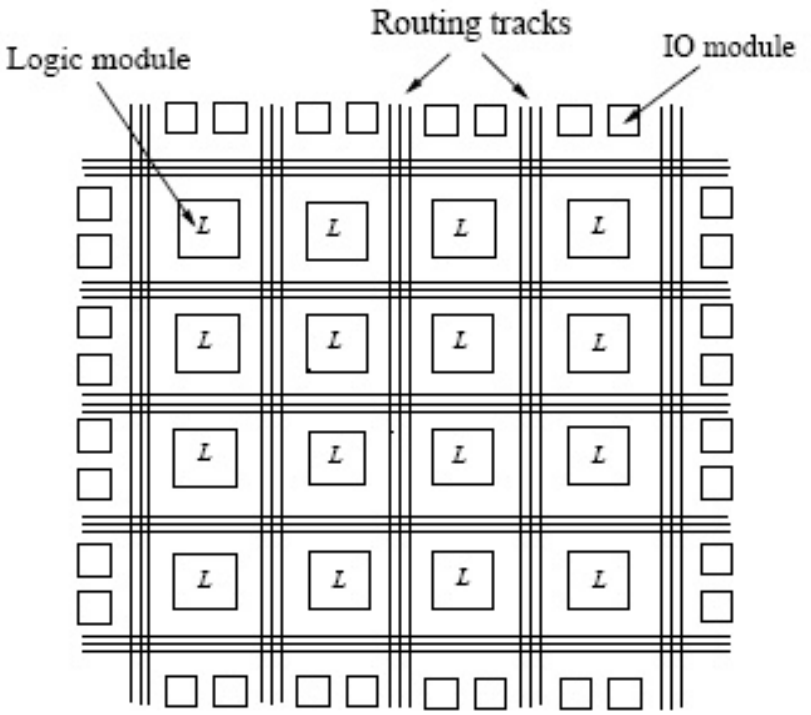


Figure 1.8: A typical FPGA architecture with three major components: logic modules, routing resources, and I/O modules.

A logic circuit is implemented in an FPGA by partitioning logic into individual logic modules and then interconnecting the modules by programming switches. A large circuit that cannot be accommodated into a single FPGA is divided into several parts each part is realized by an FPGA and these FPGAs are then interconnected by a Field-Programmable Interconnect Component (FPIC) (see Figure 1.9).

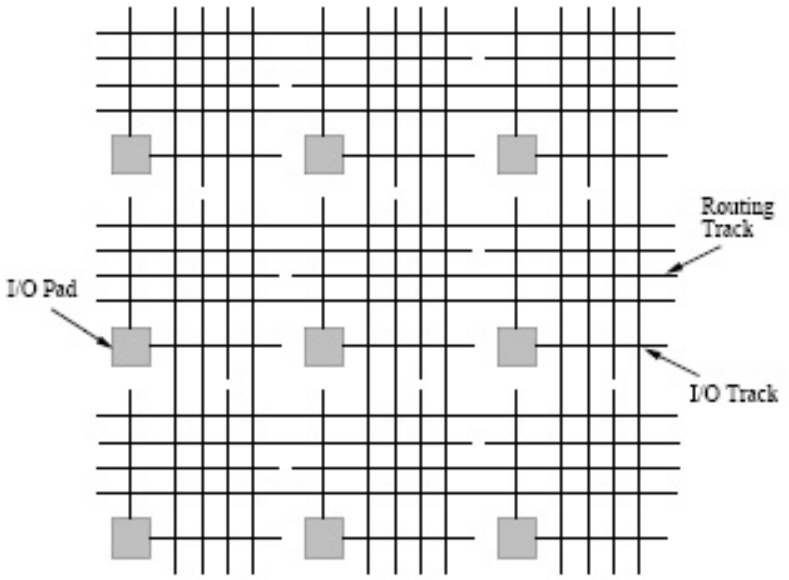


Figure 1.9: An FPIC Architecture

3.1 Programming Technologies

3.1.1 SRAM Programming Technology

The SRAM programming technology employs SRAM cells to control pass transistors or multiplexers as shown in Figures 1.10a and 1.10b. For the pass-transistor element in Figure 1.10a, the state of the SRAM cell controls the ON and OFF of the transistor (switch). When ON, the pass transistor exhibits a relatively low resistance (< 2k Ω) between the two adjacent routing wires, and thus the switch is closed; when OFF, the switch is open and the transistor incurs a very high resistance between the two routing wires. For the multiplexer approach in Figure 1.10b, the states of the SRAM cells serve as select signals and control the choice of the multiplexer's inputs.

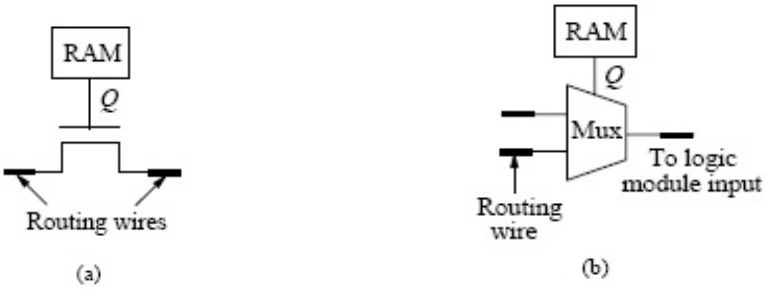


Figure 1.10: The SRAM Programming Technology.
(a) A pass-transistor switch. (b) A multiplexer switch

SRAM programming technology has the following major advantages: simple manufacturing, low, fast in-circuit reprogrammability, and low power consumption. However, a major disadvantage of SRAM programming technology is its large physical size.

One interesting property of SRAM is its volatility - a configuration is lost when its power is off. On one hand, this implies a system overhead that external permanent memory such as a Read-Only Memory (ROM). Programmable Read - Only Memory (PROM) or a disk is needed to store and provide programming configurations. On the other hand_ volatility gives SRAM-based FPGAs better design security because a design configuration is lost when power is removed.

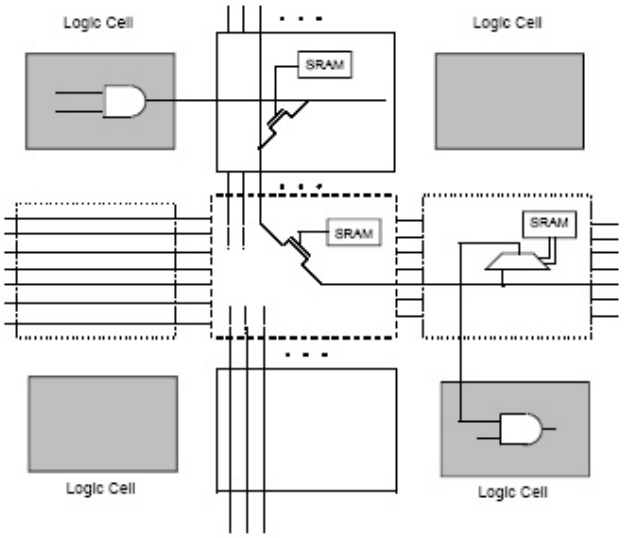


Figure 1.11: SRAM-Controlled Programmable Switch

3.1.2 Anti-fuse Programming Technology

An anti-fuse is a two-terminal, one-time programmable circuit element with high resistance (> 100 MΩ) between its terminals in the unprogrammed state and low resistance (= 500 Ω) in the programmed state.
Programming is performed by applying a higher-than-operating voltage (say, 18 V) across the anti-fuse's terminals, causing the dielectric breakdown and drastically reducing the device resistance.

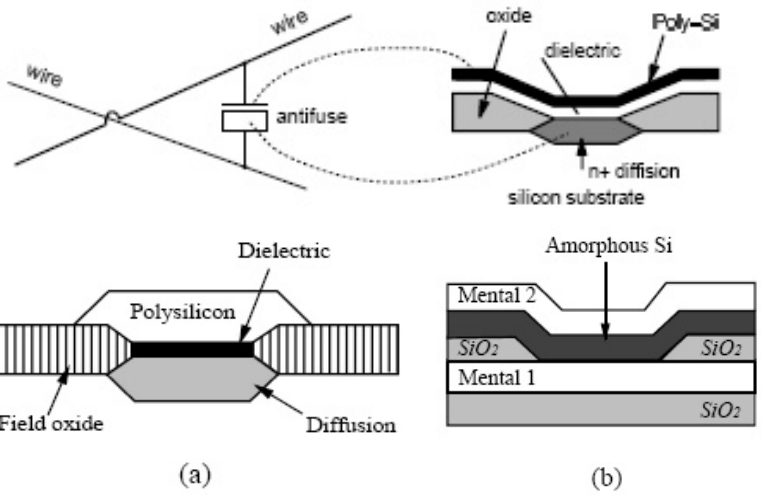


Figure 1.12: The Anti-fuse Programming Technology
(a) The ONO anti-fuse structure (b) The amorphous anti-fuse structure

Two major advantages of the anti-fuse are its small size and relatively low ON resistance and OFF capacitance; for example, the size of an amorphous anti-fuse is approximately 1 μm^2 in a 0.65 micron process. These advantages allow a much denser switch population and thus could alleviate the routing constraints imposed by the limited connectivity of routing resources. However, anti-fuse programming technology has the following major disadvantages: relatively complex manufacturing process and non-reprogrammability.

3.1.3 EPROM and EEPROM Programming Technology

An ultraviolet (UV)-erasable PROM (EPROM) is typically based on a floating gate structure as illustrated in Figure (1.13). If sufficient charge is trapped on the floating gate by applying higher-than-operating voltages between the control gate (13-14 V) and the drain of the transistor (12 V), the floating gate becomes charged negatively. The process increases the threshold voltage of the transistor to around 7 V, setting the transistor to the OFF state for all normal circuit voltages, maximally 5-6 V. The process can be reversed out-of-circuit by exposing the floating gate to UV light, giving the trapped charge sufficient energy to escape from the gate dielectric; this process reduces the threshold voltage and makes the transistor function normally.

The Electrically Erasable PROM (EEPROM) programming technology typically uses two transistors, one access and one programmed transistors, in a ROM cell. The programmed transistor performs the same function as the floating gate in an EPROM, with both charge and discharge being done electrically in-circuit without UV light, The access transistor allows programming of a cell.

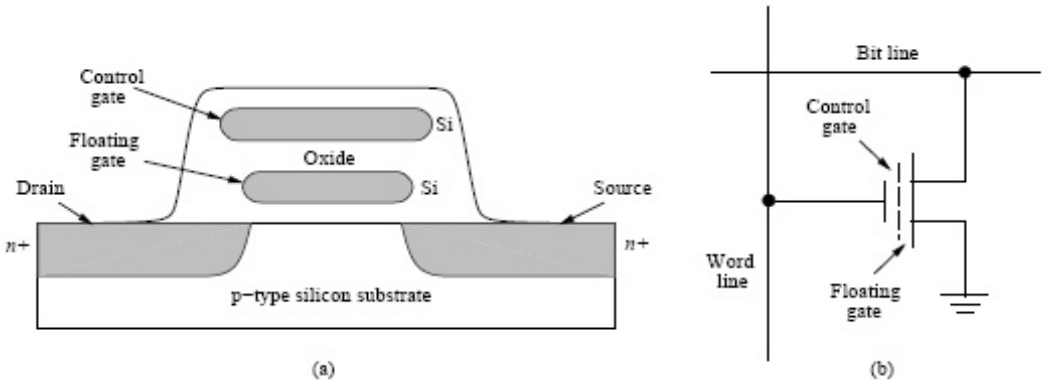


Figure 1.13: The EPROM Programming Technology
(a) UV-erasable EPROM Structure (b) Circuit Symbol for a floating-gate EPROM

The major advantages of EPROM and EEPROM technologies are their reprogrammability and full testability before shipment, especially for EEPROM. Also, unlike the SRAM technology, EPROM or EEPROM requires no external permanent memory to program the chip at power-up. However, the EPROM, EEPROM technology suffers from some drawbacks such as relatively high ON-resistance, high static power consumption, and complicated manufacturing processes

Table 1.3: Comparison of Programming Technologies

Programming Technology	SRAM	ONTO Anti-fuse	Amorphous Anti-fuse	EPROM	EEPROM
Manufacturing complexity	+++	+	+	-	--
Reprogrammable?	Yes In Circuit	No	No	Yes Out of Circuit	Yes In Circuit
Physical Size	Large (20X)	Small (2X)	Small (1X)	Large (40X)	Large (80X)
ON Resistance (O)	600-800	100-500	30-80	1-4 K	1-4 K
OFF Resistance (O)	10-50	3-5	1	10-50	10-50
Power Consumption	++	+	+	-	-
Volatile?	Yes	No	No	No	No

3.2 Logic Module Architecture

A logic circuit is implemented in a CPLD/FPGA by partitioning logic into individual logic modules and then interconnecting the modules by programming switches. A logic module has a fixed number of inputs and outputs. Certain set of functions can be implemented using a logic module_ depending on the functionality of the module. The logic modules of a high-capacity commercial programmable device are typically based on lookup tables, multiplexers, transistor pairs, basic logic gates, or SPLDs, with the first four types being popular in FPGAs and the last in CPLDs. We detail lookup table, and multiplexer-based logic modules in the following.

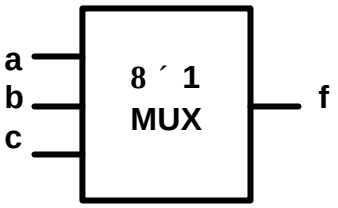
3.2.1 Lookup Table-based Logic Modules

A lookup table (LUT) based logic module is a segment of SRAM. The programming data defining the logic configuration are loaded into the SRAM at power-up. A K-input LUT is a $2^K \times 1$ SRAM with K address lines served as inputs and the 1-bit SRAM output as the LUT output. For example, if the function f = ab + a!c needs to be implemented using a 3-input LUT, then the truth table shown in Figure 1.14(a) is

loaded into the LUT, and thus the $2^3 \times 1$ SRAM would have a 0 stored at address 000, a 1 at 001, etc_ as given in the truth table.

a	B	c	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

(a)



(b)

Figure 1.14: Look-up Table Logic (a) Truth Table (b) An 8 ´ 1 SRAM

The main advantage of LUTs lies in their high functionality, a K-input LUT can realize any function of up to K inputs and there are 2 ^ 2^k such functions. However, the LUTs are only feasible for small values of K because 2^K memory cells are required for a K-input LUT with only combinational logic; typically, the value of K is 3, 4, 5 or 6. Further, the value of K may be even smaller in order to implement multi-output or sequential logic.

3.2.2 Multiplexer-based Logic Modules

A multiplexer-based logic module is typically composed of a tree of 2-to-1 multiplexers. For example, the Actel ACT 3 C-Module, shown in Figure (1.15) consists of three 2-to-1 multiplexers.
The inputs to the logic module are either data inputs d_0, d_1, d_2, d_3 or the multiplexer select inputs s_0, s_1, s_2, s_3. Therefore, the logic module can be used to implement a wide range of different functions of up to eight variables, (specifically, 766 functions for the logic module); for example_ the function f = ab + a!c can be realized by setting d_0 =1 x, d_1 = x, d_2 = c, d_4 = b, s_0 = a, s_1 = 1, s_2 = 1 and s_3 = x, where x means don't-care. Also flip-flops can be incorporated into the multiplexer-based logic module to implement sequential logic.

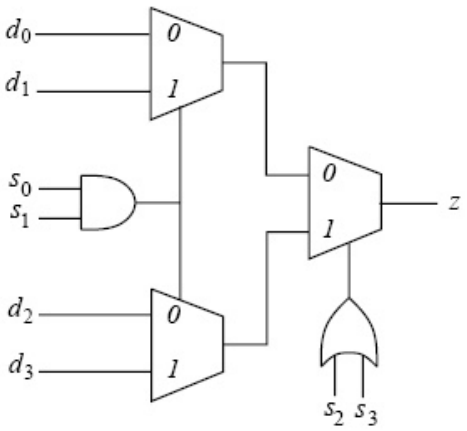


Figure 1.15: Multiplexer-based Logic Module

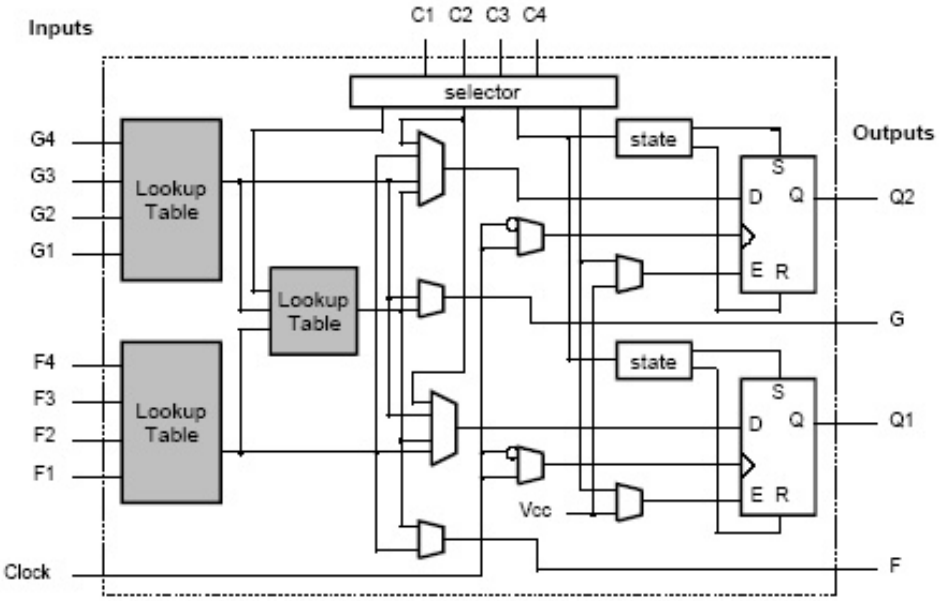


Figure 1.16: FPGA Configurable Logic Block

3.3 Routing Architecture

The interconnect architecture of a CPLD/FPGA includes routing resources and their interconnection topology on the CPLD/FPGA. CPLD/FPGA routing resources consist of pre-fabricated wire segments and programmable switches; routing in a CPLD/FPGA is performed by programming the switches to connect the wire segments. There are four major interconnection topologies for commercial CPLDs/FPGAs: array-based, row-based, sea-of-gates, and hierarchical models.

An array-based FPGA consists of a two-dimensional array of logic modules which can be connected by general routing resources (See Figure 1.17a). As mentioned earlier, the logic modules house circuits that implement logic functions. The routing resources comprise horizontal and vertical routing tracks and user-programmable switches.
A row-based CPLD/FPGA consists of multiple rows of logic modules (See Figure 1.17b). Routing wires run in the channels between two adjacent rows and also inside the channels vertically. There are additional global vertical wires providing connections among different rows (not shown in Figure 1.17b).
Unlike the array- and row-based architectures where there are routing channels separating logic modules, all logic modules in a sea-of-gates structure directly abut (See Figure 1.17c). The close proximity of logic modules allows direct connections between adjacent modules, significantly improving circuit performance. A routing network runs over the top of the modules, enabling efficient use of the available chip area.
The hierarchical structure is composed of logic modules connected by multiple levels of interconnect (See Figure 1.17d). At the first hierarchical level, several logic modules form a group, and local routing resources for the modules are provided. Together, the logic modules and their interconnect form a local component. Again, the second level of interconnect ties together a group of the components of the first level, with shared routing resources for the components. The construction for higher levels of components proceeds in the same manner.

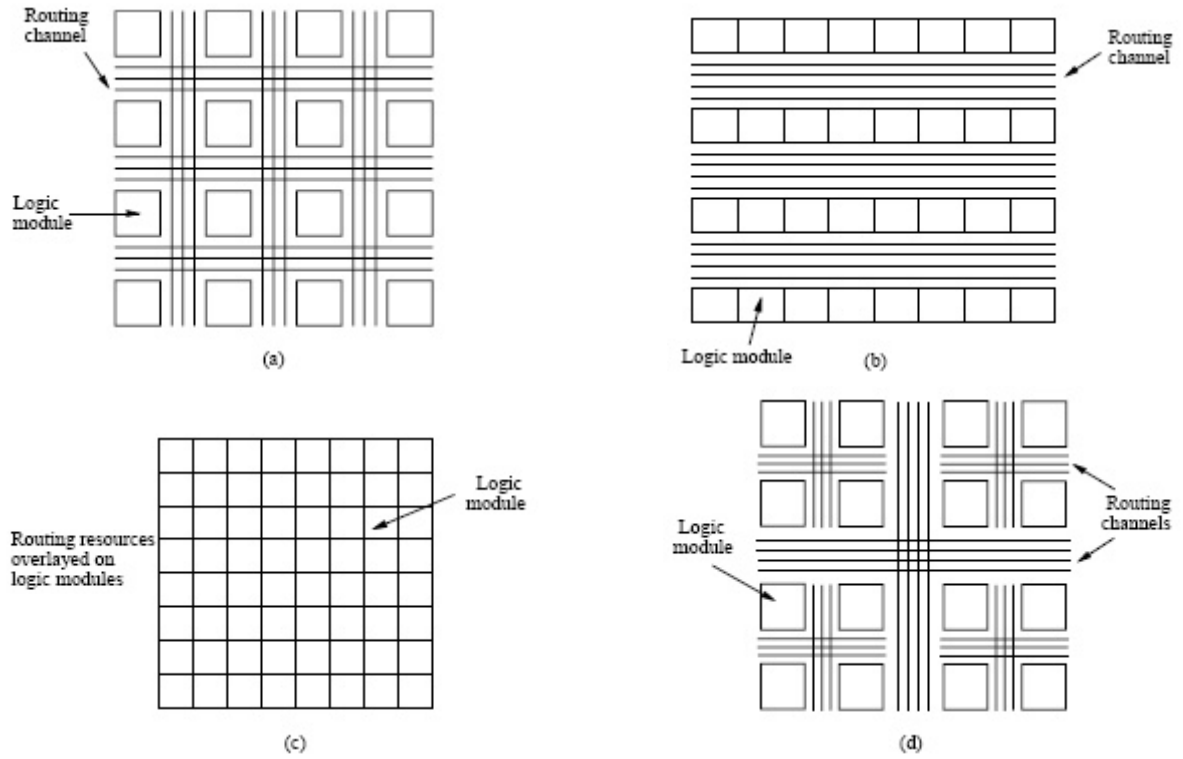
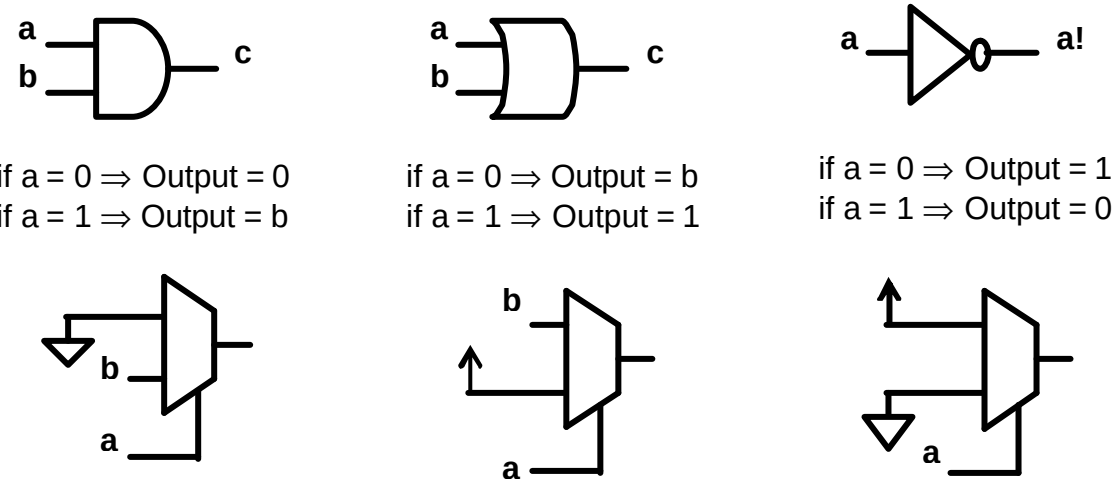


Figure 1.17: The four major CPLD/FPGA architectures
(a) The array-based model (b) The row-based model
(c) The sea-of-gates model (d) The hierarchal model

3.4 Logic Gates implementation with multiplexers

The basic building block of any combinational circuit in FPGA is the multiplexer. Using one multiplexer, we can implement all the combinational circuits. The following figure shows how AND, OR and NOT gates are implemented inside an FPGA. Of course, any other combinational circuit can be represented in a SOP form and hence will be a combination of multiplexers with appropriate connections.



if a = 0 ⇒ Output = 0
if a = 1 ⇒ Output = b

if a = 0 ⇒ Output = b
if a = 1 ⇒ Output = 1

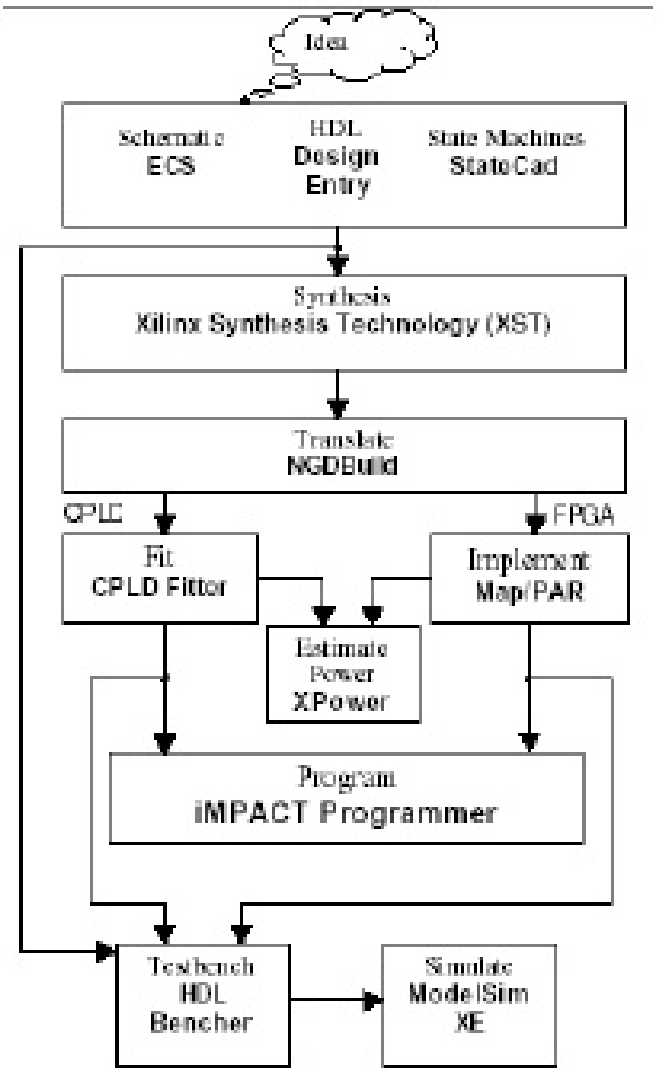
if a = 0 ⇒ Output = 1
if a = 1 ⇒ Output = 0

Figure 1.18: Logic Gate Implementation with multiplexers

4. Top-Down Design Methodology

FPGA Design Steps

1. Read customer specs.
2. Translate these specs into engineering specs.
3. Design a complete architecture for your design.
4. Design a test structure.
5. Design the data entry (Block-diagrams, HDL files, state-machines, ... etc)
6. Functional simulation
7. Choose the vendor, family, device and speed grade of the FPGA chip.
8. Run the synthesizer (Xilinix Synthesis Technology, Leonardo-Spectrum tool)
9. Place and Route.
10. Extract Delays (Standard Delay Format (SDF) files).
11. Post Layout Timing simulation (*.hdl & *.sdf files simulation).
12. Critical Path reduction (Pipelined processing or Parallel processing).
13. Download the design on the target chip.
14. Hardware testing.



5. Applications of FPGAs

FPGAs have gained rapid acceptance and growth over the past decade because they can be applied to a very wide range of applications. A list of typical applications includes: random logic, integrating multiple SPLDs, device controllers, communication encoding and filtering, small to medium sized systems with SRAM blocks, and many more.

Other interesting applications of FPGAs are prototyping of designs later to be implemented in gate arrays, and also emulation of entire large hardware systems. The former of these applications might be possible using only a single large FPGA (which corresponds to a small Gate Array in terms of capacity), and the latter would entail many FPGAs connected by some sort of interconnect; for emulation of hardware, QuickTurn [Wolff90] (and others) has developed products that comprise many FPGAs and the necessary software to partition and map circuits.

Another promising area for FPGA application, which is only beginning to be developed, is the usage of FPGAs as custom computing machines. This involves using the programmable parts to “execute” software, rather than compiling the software for execution on a regular CPU. The reader is referred to the FPGA-Based Custom Computing W orkshop (FCCM) held for the last four years and published by the IEEE.

When designs are mapped into CPLDs, pieces of the design often map naturally to the SPLD-like blocks. However, designs mapped into an FPGA are broken up into logic block-sized pieces and distributed through an area of the FPGA. Depending on the FPGA’s interconnect structure, there may be various delays associated with the interconnections between these logic blocks. Thus, FPGA performance often depends more upon how CAD tools map circuits into the chip than is the case for CPLDs.

We believe that over time programmable logic will become the dominant form of digital logic design and implementation. Their ease of access, principally through the low cost of the devices, makes them attractive to small firms and small parts of large companies. The fast manufacturing turn-around they provide is an essential element of success in the market. As architecture and CAD tools improve, the disadvantages of FPDs compared to Mask-Programmed Gate Arrays will lessen, and programmable devices will dominate.