Module 4

Design of Embedded Processors

Lesson 20

Field Programmable Gate Arrays and Applications

Instructional Objectives

After going through this lesson the student will be able to

- Define what is a field programmable gate array (FPGA)
- Distinguish between an FPGA and a stored-memory processor
- List and explain the principle of operation of the various functional units within an FPGA
- Compare the architecture and performance specifications of various commercially available FPGA
- Describe the steps in using an FPGA in an embedded system

Introduction

An **FPGA** is a device that contains a matrix of reconfigurable gate array logic circuitry. When a FPGA is configured, the internal circuitry is connected in a way that creates a hardware implementation of the software application. Unlike processors, FPGAs use dedicated hardware for processing logic and do not have an operating system. FPGAs are truly parallel in nature so different processing operations do not have to compete for the same resources. As a result, the performance of one part of the application is not affected when additional processing is added. Also, multiple control loops can run on a single FPGA device at different rates. FPGA-based control systems can enforce critical interlock logic and can be designed to prevent I/O forcing by an operator. However, unlike hard-wired printed circuit board (PCB) designs which have fixed hardware resources, FPGA-based systems can literally rewire their internal circuitry to allow reconfiguration after the control system is deployed to the field. FPGA devices deliver the performance and reliability of dedicated hardware circuitry.

A single FPGA can replace thousands of discrete components by incorporating millions of logic gates in a single integrated circuit (IC) chip. The internal resources of an FPGA chip consist of a matrix of configurable logic blocks (CLBs) surrounded by a periphery of I/O blocks shown in Fig. 20.1. Signals are routed within the FPGA matrix by programmable interconnect switches and wire routes.

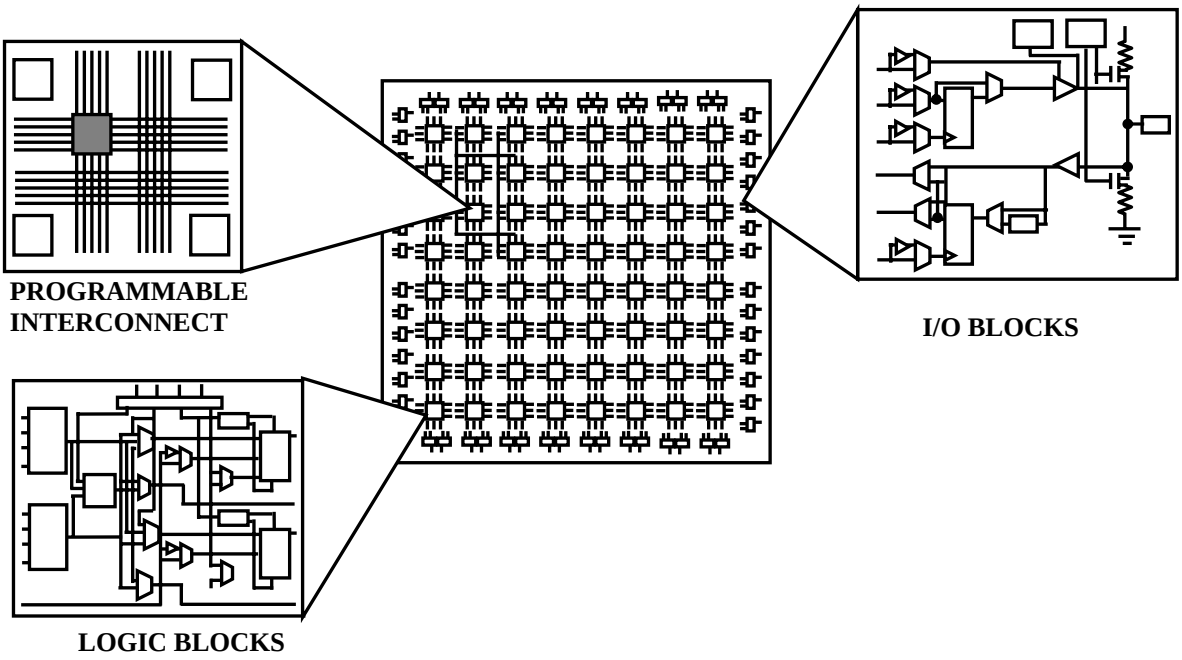


Fig. 20.1 Internal Structure of FPGA

In an FPGA logic blocks are implemented using multiple level low fan-in gates, which gives it a more compact design compared to an implementation with two-level AND-OR logic. FPGA provides its user a way to configure:

1. The intersection between the logic blocks and
2. The function of each logic block.

Logic block of an FPGA can be configured in such a way that it can provide functionality as simple as that of transistor or as complex as that of a microprocessor. It can used to implement different combinations of combinational and sequential logic functions. Logic blocks of an FPGA can be implemented by any of the following:

1. Transistor pairs
2. combinational gates like basic NAND gates or XOR gates
3. n-input Lookup tables
4. Multiplexers
5. Wide fan-in And-OR structure.

Routing in FPGAs consists of wire segments of varying lengths which can be interconnected via electrically programmable switches. Density of logic block used in an FPGA depends on length and number of wire segments used for routing. Number of segments used for interconnection typically is a tradeoff between density of logic blocks used and amount of area used up for routing. Simplified version of FPGA internal architecture with routing is shown in Fig. 20.2.

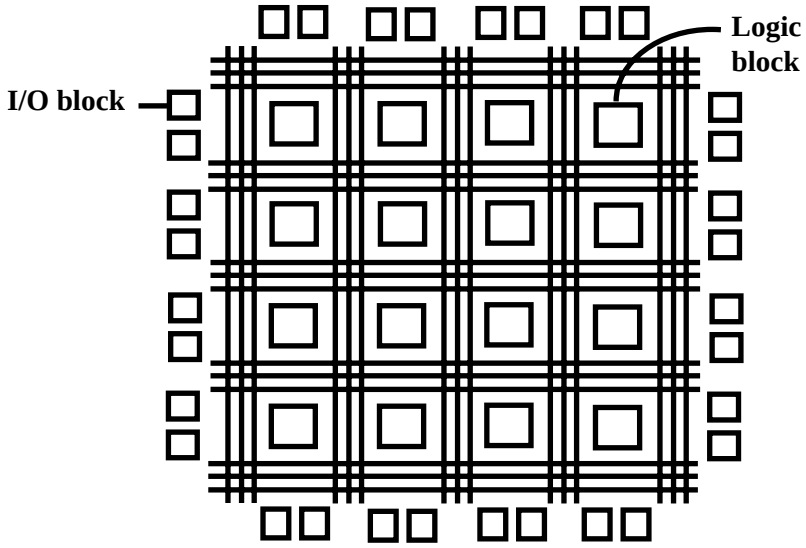


Fig. 20.2 Simplified Internal Structure of FPGA

Why do we need FPGAs?

By the early 1980's **large scale integrated circuits (LSI)** formed the back bone of most of the logic circuits in major systems. Microprocessors, bus/IO controllers, system timers etc were implemented using integrated circuit fabrication technology. Random "glue logic" or interconnects were still required to help connect the large integrated circuits in order to:

1. Generate global control signals (for resets etc.)
2. Data signals from one subsystem to another sub system.

Systems typically consisted of few large scale integrated components and large number of SSI (small scale integrated circuit) and MSI (medium scale integrated circuit) components.Intial attempt to solve this problem led to development of **Custom ICs** which were to replace the large amount of interconnect. This reduced system complexity and manufacturing cost, and improved performance. However, custom ICs have their own disadvantages. They are relatively very expensive to develop, and delay introduced for product to market (time to market) because of increased design time. There are two kinds of costs involved in development of custom ICs

1. Cost of development and design
2. Cost of manufacture

(A tradeoff usually exists between the two costs)

Therefore the custom IC approach was only viable for products with very high volume, and which were not time to market sensitive.FPGAs were introduced as an alternative to custom ICs for implementing entire system on one chip and to provide flexibility of reporogramability to the user. Introduction of FPGAs resulted in improvement of density relative to discrete SSI/MSI components (within around 10x of custom ICs). Another advantage of FPGAs over Custom ICs is that with the help of computer aided design (CAD) tools circuits could be implemented in a short amount of time (no physical layout process, no mask making, no IC manufacturing)

Evaluation of FPGA

In the world of digital electronic systems, there are three basic kinds of devices: memory, microprocessors, and logic. Memory devices store random information such as the contents of a

spreadsheet or database. Microprocessors execute software instructions to perform a wide variety of tasks such as running a word processing program or video game. Logic devices provide specific functions, including device-to-device interfacing, data communication, signal processing, data display, timing and control operations, and almost every other function a system must perform.

The first type of user-programmable chip that could implement logic circuits was the Programmable Read-Only Memory (PROM), in which address lines can be used as logic circuit inputs and data lines as outputs. Logic functions, however, rarely require more than a few product terms, and a PROM contains a full decoder for its address inputs. PROMS are thus an inefficient architecture for realizing logic circuits, and so are rarely used in practice for that purpose. The device that came as a replacement for the PROM's are programmable logic devices or in short PLA. Logically, a PLA is a circuit that allows implementing Boolean functions in sum-of-product form. The typical implementation consists of input buffers for all inputs, the programmable AND-matrix followed by the programmable OR-matrix, and output buffers. The input buffers provide both the original and the inverted values of each PLA input. The input lines run horizontally into the AND matrix, while the so-called product-term lines run vertically. Therefore, the size of the AND matrix is twice the number of inputs times the number of product-terms.

When PLAs were introduced in the early 1970s, by Philips, their main drawbacks were that they were expensive to manufacture and offered somewhat poor speed-performance. Both disadvantages were due to the two levels of configurable logic, because programmable logic planes were difficult to manufacture and introduced significant propagation delays. To overcome these weaknesses, Programmable Array Logic (PAL) devices were developed. PALs provide only a single level of programmability, consisting of a programmable "wired" AND plane that feeds fixed OR-gates. PALs usually contain flip-flops connected to the OR-gate outputs so that sequential circuits can be realized. These are often referred to as **Simple Programmable Logic Devices (SPLDs)**. Fig. 20.3 shows a simplified structure of PLA and PAL.

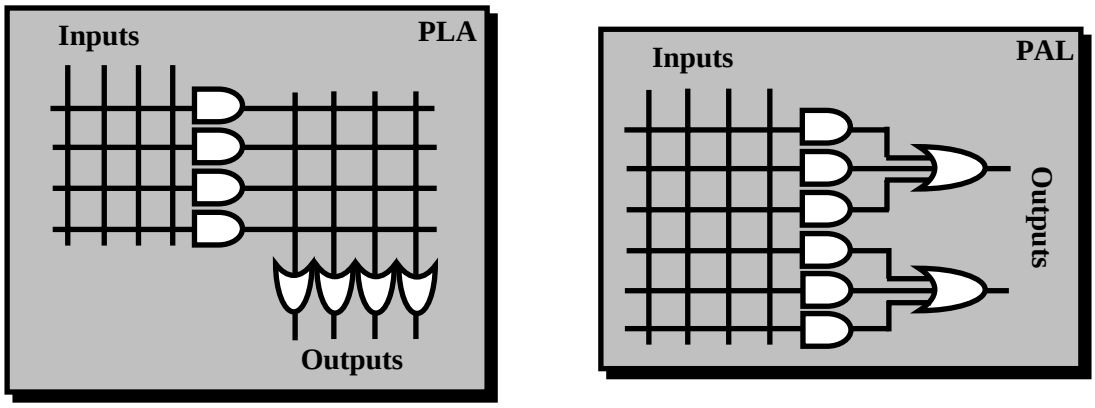


Fig. 20.3 Simplified Structure of PLA and PAL

With the advancement of technology, it has become possible to produce devices with higher capacities than SPLD's.As chip densities increased, it was natural for the PLD manufacturers to evolve their products into larger (logically, but not necessarily physically) parts called Complex Programmable Logic Devices (CPLDs). For most practical purposes, CPLDs can be thought of as multiple PLDs (plus some programmable interconnect) in a single chip. The larger size of a CPLD allows to implement either more logic equations or a more complicated design.

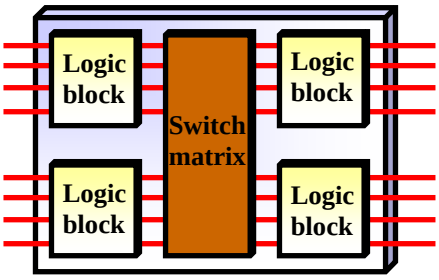


Fig. 20.4 Internal structure of a CPLD

Fig. 20.4 contains a block diagram of a hypothetical CPLD. Each of the four logic blocks shown there is the equivalent of one PLD. However, in an actual CPLD there may be more (or less) than four logic blocks. These logic blocks are themselves comprised of macrocells and interconnect wiring, just like an ordinary PLD.

Unlike the programmable interconnect within a PLD, the switch matrix within a CPLD may or may not be fully connected. In other words, some of the theoretically possible connections between logic block outputs and inputs may not actually be supported within a given CPLD. The effect of this is most often to make 100% utilization of the macrocells very difficult to achieve. Some hardware designs simply won't fit within a given CPLD, even though there are sufficient logic gates and flip-flops available. Because CPLDs can hold larger designs than PLDs, their potential uses are more varied. They are still sometimes used for simple applications like address decoding, but more often contain high-performance control-logic or complex finite state machines. At the high-end (in terms of numbers of gates), there is also a lot of overlap in potential applications with FPGAs. Traditionally, CPLDs have been chosen over FPGAs whenever high-performance logic is required. Because of its less flexible internal architecture, the delay through a CPLD (measured in nanoseconds) is more predictable and usually shorter.

The development of the FPGA was distinct from the SPLD/CPLD evolution just described.This is apparent from the architecture of FPGA shown in Fig 20.1. FPGAs offer the highest amount of logic density, the most features, and the highest performance. The largest FPGA now shipping, part of the Xilinx Virtex™ line of devices, provides eight million "system gates" (the relative density of logic). These advanced devices also offer features such as built-in hardwired processors (such as the IBM Power PC), substantial amounts of memory, clock management systems, and support for many of the latest, very fast device-to-device signaling technologies. FPGAs are used in a wide variety of applications ranging from data processing and storage, to instrumentation, telecommunications, and digital signal processing. The value of programmable logic has always been its ability to shorten development cycles for electronic equipment manufacturers and help them get their product to market faster. As PLD (Programmable Logic Device) suppliers continue to integrate more functions inside their devices, reduce costs, and increase the availability of time-saving IP cores, programmable logic is certain to expand its popularity with digital designers.

FPGA Structural Classification

Basic structure of an FPGA includes logic elements, programmable interconnects and memory. Arrangement of these blocks is specific to particular manufacturer. On the basis of internal arrangement of blocks FPGAs can be divided into three classes:

Symmetrical arrays

This architecture consists of logic elements (called CLBs) arranged in rows and columns of a matrix and interconnect laid out between them shown in Fig 20.2. This symmetrical matrix is surrounded by I/O blocks which connect it to outside world. Each CLB consists of n-input Lookup table and a pair of programmable flip flops. I/O blocks also control functions such as tri-state control, output transition speed. Interconnects provide routing path. Direct interconnects between adjacent logic elements have smaller delay compared to general purpose interconnect

Row based architecture

Row based architecture shown in Fig 20.5 consists of alternating rows of logic modules and programmable interconnect tracks. Input output blocks is located in the periphery of the rows. One row may be connected to adjacent rows via vertical interconnect. Logic modules can be implemented in various combinations. Combinatorial modules contain only combinational elements which Sequential modules contain both combinational elements along with flip flops. This sequential module can implement complex combinatorial-sequential functions. Routing tracks are divided into smaller segments connected by anti-fuse elements between them.

Hierarchical PLDs

This architecture is designed in hierarchical manner with top level containing only logic blocks and interconnects. Each logic block contains number of logic modules. And each logic module has combinatorial as well as sequential functional elements. Each of these functional elements is controlled by the programmed memory. Communication between logic blocks is achieved by programmable interconnect arrays. Input output blocks surround this scheme of logic blocks and interconnects. This type of architecture is shown in Fig 20.6.

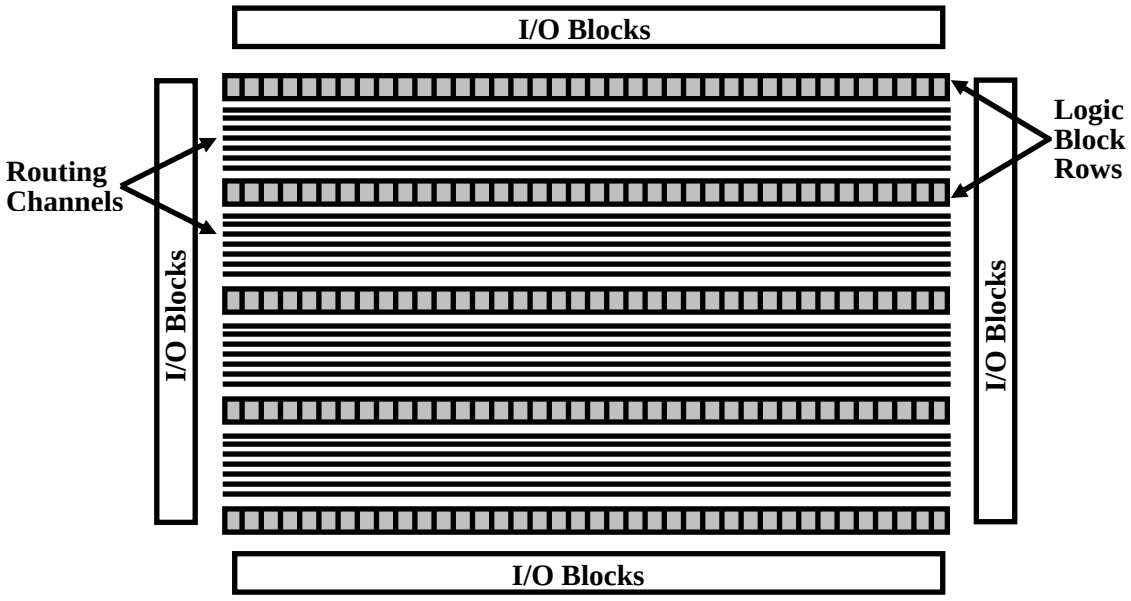


Fig. 20.5 Row based Architecture

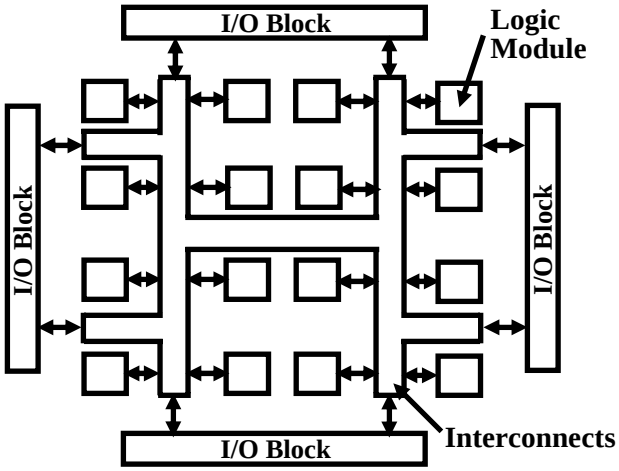


Fig. 20.6 Hierarchical PLD

FPGA Classification on user programmable switch technologies

FPGAs are based on an array of logic modules and a supply of uncommitted wires to route signals. In gate arrays these wires are connected by a mask design during manufacture. In FPGAs, however, these wires are connected by the user and therefore must use an electronic device to connect them. Three types of devices have been commonly used to do this, pass transistors controlled by an SRAM cell, a flash or EEPROM cell to pass the signal, or a direct connect using antifuses. Each of these interconnect devices have their own advantages and disadvantages. This has a major affect on the design, architecture, and performance of the FPGA. Classification of FPGAs on user programmable switch technology is given in Fig. 20.7 shown below.

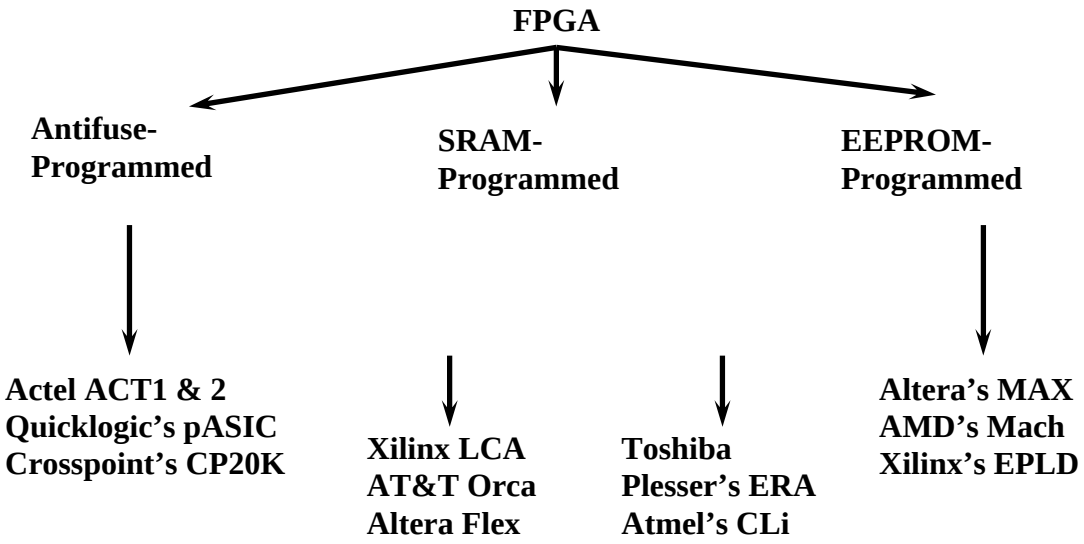


Fig. 20.7 FPGA Classification on user programmable technology

SRAM Based

The major advantage of SRAM based device is that they are infinitely re-programmable and can be soldered into the system and have their function changed quickly by merely changing the contents of a PROM. They therefore have simple development mechanics. They can also be changed in the field by uploading new application code, a feature attractive to designers. It does however come with a price as the interconnect element has high impedance and capacitance as well as consuming much more area than other technologies. Hence wires are very expensive and slow. The FPGA architect is therefore forced to make large inefficient logic modules (typically a look up table or LUT).The other disadvantages are: They needs to be reprogrammed each time when power is applied, needs an external memory to store program and require large area. Fig. 20.8 shows two applications of SRAM cells: for controlling the gate nodes of pass-transistor switches and to control the select lines of multiplexers that drive logic block inputs. The figures gives an example of the connection of one logic block (represented by the AND-gate in the upper left corner) to another through two pass-transistor switches, and then a multiplexer, all controlled by SRAM cells . Whether an FPGA uses pass-transistors or multiplexers or both depends on the particular product.

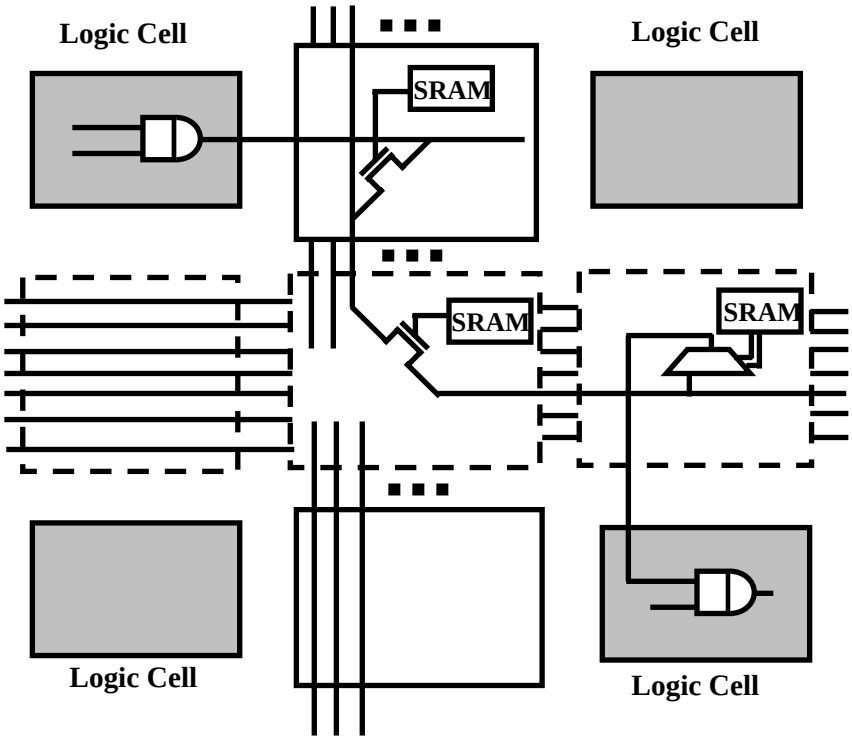


Fig. 20.8 SRAM-controlled Programmable Switches.

Antifuse Based

The antifuse based cell is the highest density interconnect by being a true cross point. Thus the designer has a much larger number of interconnects so logic modules can be smaller and more efficient. Place and route software also has a much easier time. These devices however are only one-time programmable and therefore have to be thrown out every time a change is made in the design. The Antifuse has an inherently low capacitance and resistance such that the fastest parts are all Antifuse based. The disadvantage of the antifuse is the requirement to integrate the fabrication of the antifuses into the IC process, which means the process will always lag the SRAM process in scaling. Antifuses are suitable for FPGAs because they can be built using modified CMOS technology. As an example, Actel's antifuse structure is depicted in Fig. 20.9. The figure shows that an antifuse is positioned between two interconnect wires and physically consists of three sandwiched layers: the top and bottom layers are conductors, and the middle layer is an insulator. When unprogrammed, the insulator isolates the top and bottom layers, but when programmed the insulator changes to become a low-resistance link. It uses Poly-Si and n+ diffusion as conductors and ONO as an insulator, but other antifuses rely on metal for conductors, with amorphous silicon as the middle layer.

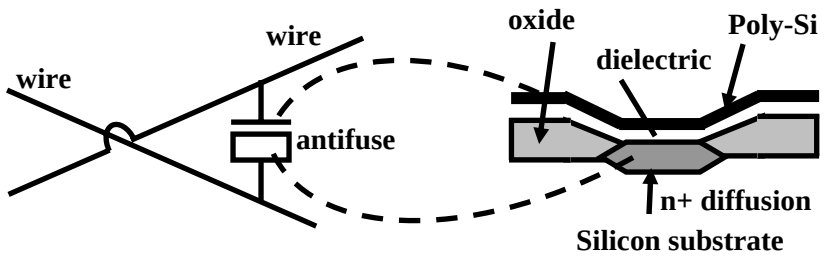


Fig. 20.9 Actel Antifuse Structure.

EEPROM Based

The EEPROM/FLASH cell in FPGAs can be used in two ways, as a control device as in an SRAM cell or as a directly programmable switch. When used as a switch they can be very efficient as interconnect and can be reprogrammable at the same time. They are also non-volatile so they do not require an extra PROM for loading. They, however, do have their detractions. The EEPROM process is complicated and therefore also lags SRAM technology.

Logic Block and Routing Techniques

Crosspoint FPGA: consist of two types of logic blocks. One is transistor pair tiles in which transistor pairs run in parallel lines as shown in figure below:

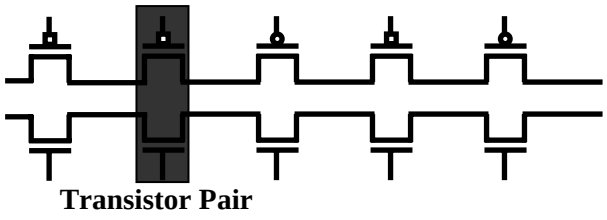


Fig. 20.10 Transistor pair tiles in cross-point FPGA.

second type of logic blocks are RAM logic which can be used to implement random access memory.

Plessey FPGA: Basic building block here is 2-input NAND gate which is connected to each other to implement desired function.

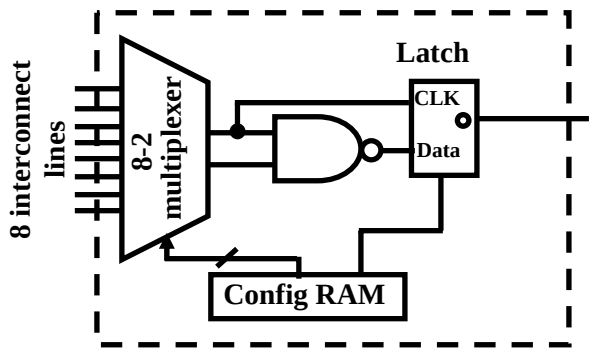


Fig. 20.11 Plessey Logic Block

Both Crosspoint and Plessey are fine grain logic blocks. Fine grain logic blocks have an advantage in high percentage usage of logic blocks but they require large number of wire segments and programmable switches which occupy lot of area.

Actel Logic Block: If inputs of a multiplexer are connected to a constant or to a signal, it can be used to implement different logic functions. For example a 2-input multiplexer with inputs a and b, select, will implement function ac + bc´. If b=0 then it will implement ac, and if a=0 it will implement bc´.

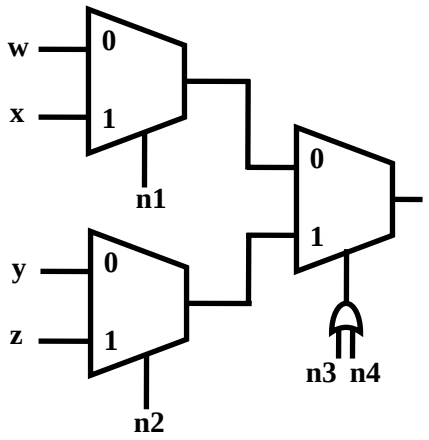


Fig. 20.12 Actel Logic Block

Typically an Actel logic block consists of multiple number of multiplexers and logic gates.

Xilinx Logic block

In Xilinx logic block Look up table is used to implement any number of different functionality. The input lines go into the input and enable of lookup table. The output of the lookup table gives the result of the logic function that it implements. Lookup table is implemented using SRAM.

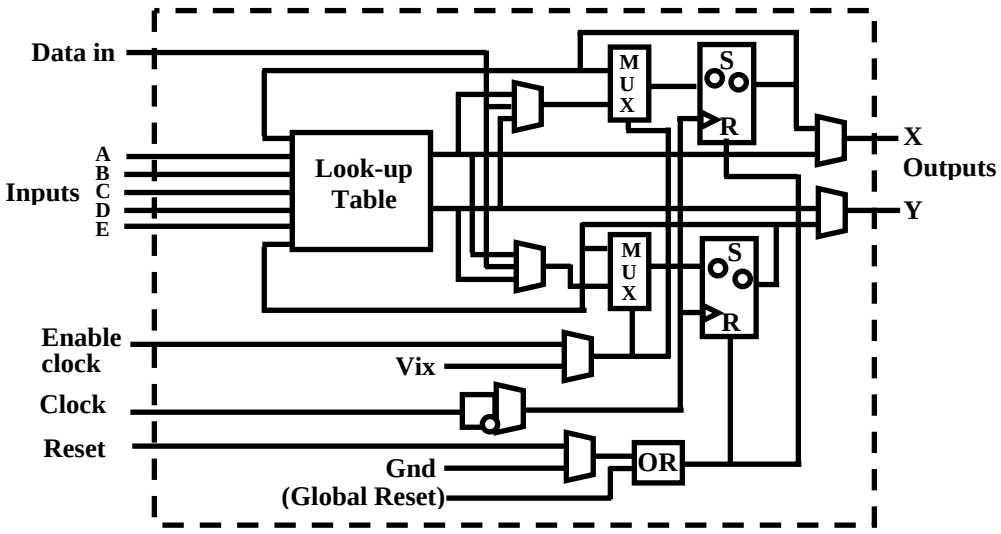
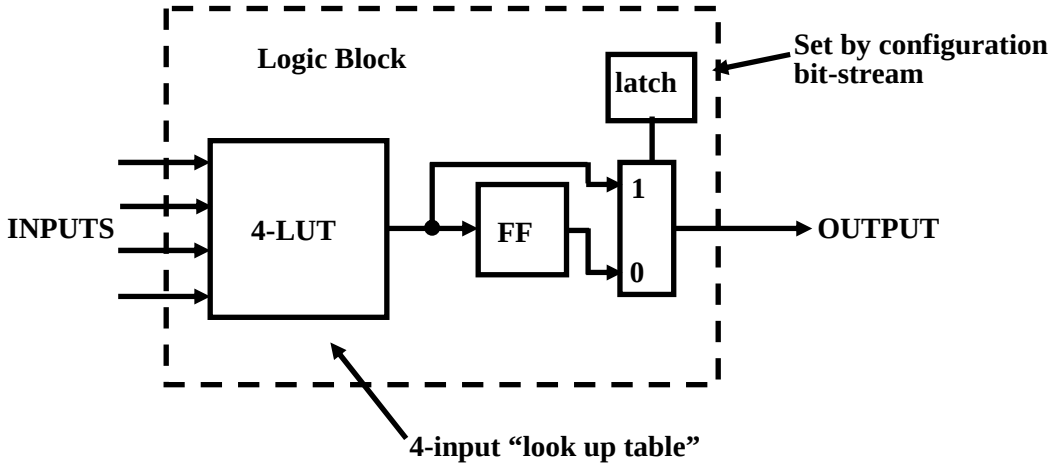


Fig. 20.13 Xilinx - LUT based

A k-input logic function is implemented using 2^k * 1 size SRAM. Number of different possible functions for k input LUT is 2^2^k. Advantage of such an architecture is that it supports implementation of so many logic functions, however the disadvantage is unusually large number of memory cells required to implement such a logic block in case number of inputs is large. Fig. 20.13 shows 5-input LUT based implementation of logic block LUT based design provides for better logic block utilization. A k-input LUT based logic block can be implemented in number of different ways with tradeoff between performance and logic density.



An n-lut can be shown as a direct implementation of a function truth-table. Each of the latch holds the value of the function corresponding to one input combination. For Example: 2-lut shown in figure below implements 2 input AND and OR functions.

Example: 2-lut

INPUTS	AND	OR
00	0	0
01	0	1
10	0	1
11	1	1

Altera Logic Block

Altera's logic block has evolved from earlier PLDs. It consists of wide fan in (up to 100 input) AND gates feeding into an OR gate with 3-8 inputs. The advantage of large fan in AND gate based implementation is that few logic blocks can implement the entire functionality thereby reducing the amount of area required by interconnects. On the other hand disadvantage is the low density usage of logic blocks in a design that requires fewer input logic. Another disadvantage is the use of pull up devices (AND gates) that consume static power. To improve power manufacturers provide low power consuming logic blocks at the expense of delay. Such logic blocks have gates with high threshold as a result they consume less power. Such logic blocks can be used in non-critical paths.
Altera, Xilinx are coarse grain architecture.

Example: Altera's FLEX 8000 series consists of a three-level hierarchy. However, the lowest level of the hierarchy consists of a set of lookup tables, rather than an SPLD like block, and so the FLEX 8000 is categorized here as an FPGA. It should be noted, however, that FLEX 8000 is

a combination of FPGA and CPLD technologies. FLEX 8000 is SRAM-based and features a four-input LUT as its basic logic block. Logic capacity ranges from about 4000 gates to more than 15,000 for the 8000 series. The overall architecture of FLEX 8000 is illustrated in Fig. 20.14.

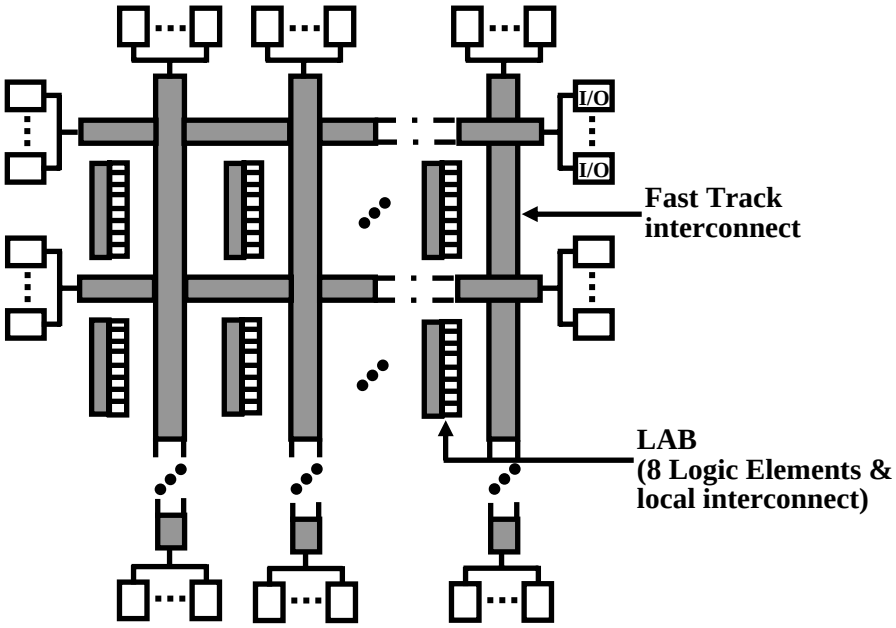


Fig. 20.14 Architecture of Altera FLEX 8000 FPGAs.

The basic logic block, called a Logic Element (LE) contains a four-input LUT, a flip-flop, and special-purpose carry circuitry for arithmetic circuits. The LE also includes cascade circuitry that allows for efficient implementation of wide AND functions. Details of the LE are illustrated in Fig. 20.15.

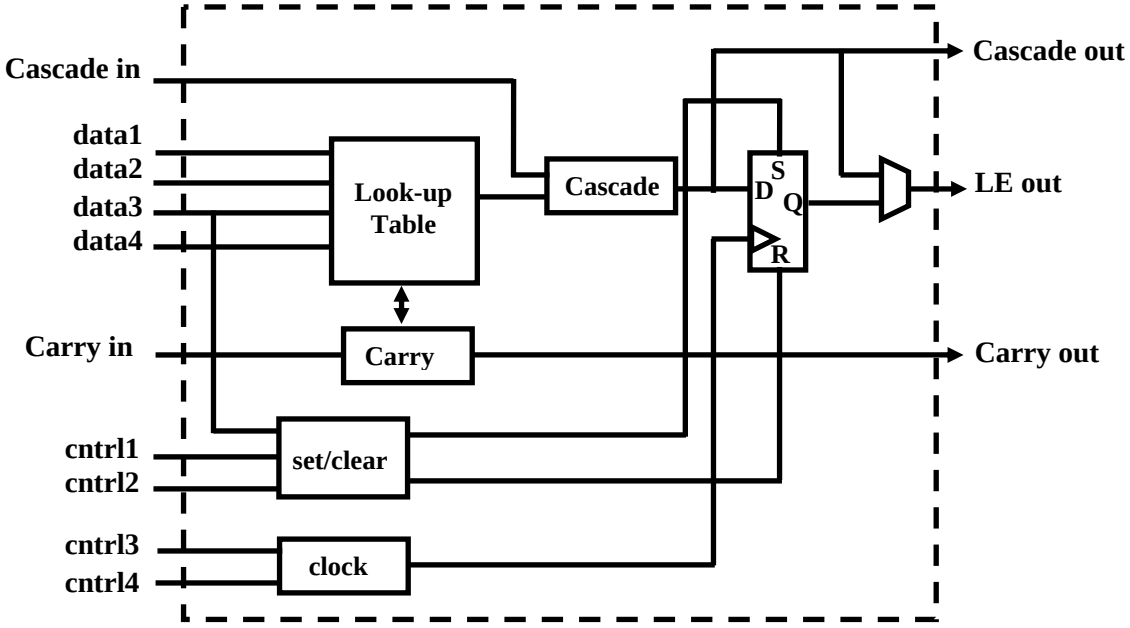


Fig. 20.15 Altera FLEX 8000 Logic Element (LE).

In the FLEX 8000, LEs are grouped into sets of 8, called Logic Array Blocks (LABs, a term borrowed from Altera's CPLDs). As shown in Fig. 20.16, each LAB contains local interconnect and each local wire can connect any LE to any other LE within the same LAB. Local interconnect also connects to the FLEX 8000's *global interconnect*, called FastTrack. All FastTrack wires horizontal wires are identical, and so interconnect delays in the FLEX 8000 are more predictable than FPGAs that employ many smaller length segments because there are fewer programmable switches in the longer path

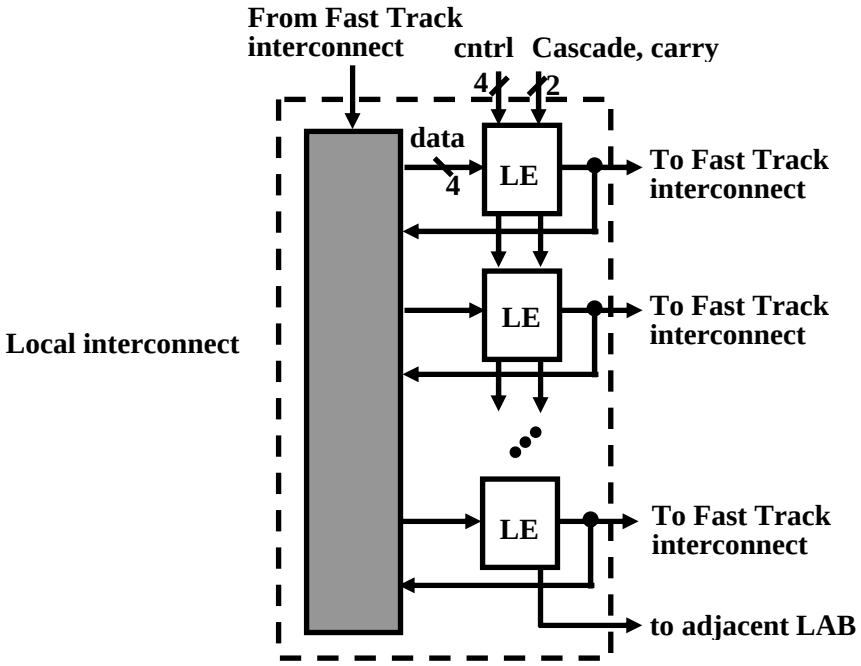


Fig. 20.16 Altera FLEX 8000 Logic Array Block (LAB).

FPGA Design Flow

One of the most important advantages of FPGA based design is that users can design it using CAD tools provided by design automation companies. Generic design flow of an FPGA includes following steps:

System Design

At this stage designer has to decide what portion of his functionality has to be implemented on FPGA and how to integrate that functionality with rest of the system.

I/O integration with rest of the system

Input Output streams of the FPGA are integrated with rest of the Printed Circuit Board, which allows the design of the PCB early in design process. FPGA vendors provide extra automation software solutions for I/O design process.

Design Description

Designer describes design functionality either by using schematic editors or by using one of the various Hardware Description Languages (HDLs) like Verilog or VHDL.

Synthesis

Once design has been defined CAD tools are used to implement the design on a given FPGA. Synthesis includes generic optimization, slack optimizations, power optimizations followed by placement and routing. Implementation includes Partition, Place and route. The output of design implementation phase is bit-stream file.

Design Verification

Bit stream file is fed to a simulator which simulates the design functionality and reports errors in desired behavior of the design. Timing tools are used to determine maximum clock frequency of the design. Now the design is loading onto the target FPGA device and testing is done in real environment.

Hardware design and development

The process of creating digital logic is not unlike the embedded software development process. A description of the hardware's structure and behavior is written in a high-level hardware description language (usually VHDL or Verilog) and that code is then compiled and downloaded prior to execution. Of course, schematic capture is also an option for design entry, but it has become less popular as designs have become more complex and the language-based tools have improved. The overall process of hardware development for programmable logic is shown in Fig. 20.17 and described in the paragraphs that follow.

Perhaps the most striking difference between hardware and software design is the way a developer must think about the problem. Software developers tend to think sequentially, even when they are developing a multithreaded application. The lines of source code that they write are always executed in that order, at least within a given thread. If there is an operating system it is used to create the appearance of parallelism, but there is still just one execution engine. During design entry, hardware designers must think-and program-in parallel. All of the input signals are processed in parallel, as they travel through a set of execution engines-each one a series of macrocells and interconnections-toward their destination output signals. Therefore, the statements of a hardware description language create structures, all of which are "executed" at the very same time.

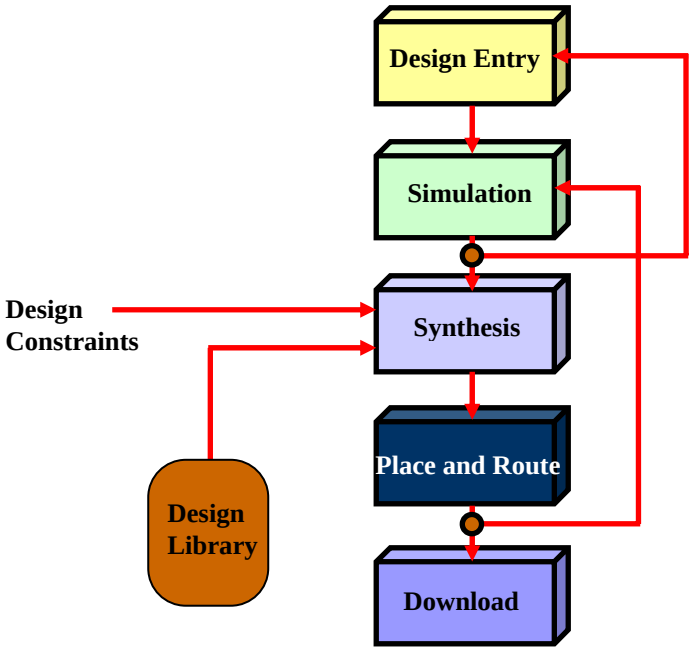


Fig. 20.17 Programmable logic design process

Typically, the design entry step is followed or interspersed with periods of functional simulation. That's where a simulator is used to execute the design and confirm that the correct outputs are produced for a given set of test inputs. Although problems with the size or timing of the hardware may still crop up later, the designer can at least be sure that his logic is functionally correct before going on to the next stage of development.

Compilation only begins after a functionally correct representation of the hardware exists. This hardware compilation consists of two distinct steps. First, an intermediate representation of the hardware design is produced. This step is called synthesis and the result is a representation called a netlist. The netlist is device independent, so its contents do not depend on the particulars of the FPGA or CPLD; it is usually stored in a standard format called the Electronic Design Interchange Format (EDIF).

The second step in the translation process is called place & route. This step involves mapping the logical structures described in the netlist onto actual macrocells, interconnections, and input and output pins. This process is similar to the equivalent step in the development of a printed circuit board, and it may likewise allow for either automatic or manual layout optimizations. The result of the place & route process is a bitstream. This name is used generically, despite the fact that each CPLD or FPGA (or family) has its own, usually proprietary, bitstream format. Suffice it to say that the bitstream is the binary data that must be loaded into the FPGA or CPLD to cause that chip to execute a particular hardware design.

Increasingly there are also debuggers available that at least allow for single-stepping the hardware design as it executes in the programmable logic device. But those only complement a simulation environment that is able to use some of the information generated during the place & route step to provide gate-level simulation. Obviously, this type of integration of device-specific information into a generic simulator requires a good working relationship between the chip and simulation tool vendors.

Things to Ponder

Q.1 Define the following acronyms as they apply to digital logic circuits:

- ASIC
- PAL
- PLA
- PLD
- CPLD
- FPGA

Q2.How granularity of logic block influences the performance of an FPGA?

Q3. Why would anyone use programmable logic devices (PLD, PAL, PLA, CPLD, FPGA, etc.) in place of traditional "hard-wired" logic such as NAND, NOR, AND, and OR gates? Are there any applications where hard-wired logic would do a better job than a programmable device?

Q4.Some programmable logic devices (and PROM memory devices as well) use tiny fuses which are intentionally "blown" in specific patterns to represent the desired program. Programming a device by blowing tiny fuses inside of it carries certain advantages and disadvantages - describe what some of these are.

Q5. Use one 4 x 8 x 4 PLA to implement the function.

$$F_1(w,x,y,z) = wx'y'z + wx'yz' + wxy'$$

$$F_2(w,x,y,z) = wx'y + x'y'z$$

Source: http://www.nptel.ac.in/courses/Webcourse-contents/IIT%20Kharagpur/Embedded%20systems/Pdf/Lesson-20.pdf