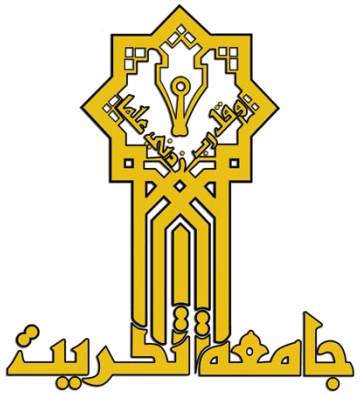


DIGITAL DEVICES AND LOGIC FAMILIES

ELECTRICAL DEPARTMENT

FOURTH STAGE

INSTRUCTOR: DR. ANAS MQDAD

LECTURE THREE

PROGRAMMABLE LOGIC

OUTLINES

- SRAM-Based FPGAs
- FPGA Cores
- Embedded Functions
- Specific FPGA Devices
- Programmable Logic Software

SRAM-Based FPGAs

- FPGAs are either nonvolatile because they are based on antifuse technology or they are volatile because they are based on SRAM technology. (The term *volatile* means that all the data programmed into the configurable logic blocks are lost when power is turned off.)
- Therefore, SRAM-based FPGAs include either a nonvolatile configuration memory embedded on the chip to store the program data and reconfigure the device each time power is turned back on or they use an external memory with data transfer controlled by a host processor.
- The concept of on-the-chip memory is illustrated in Figure 1 (a). The concept of the host processor configuration is shown in part (b).

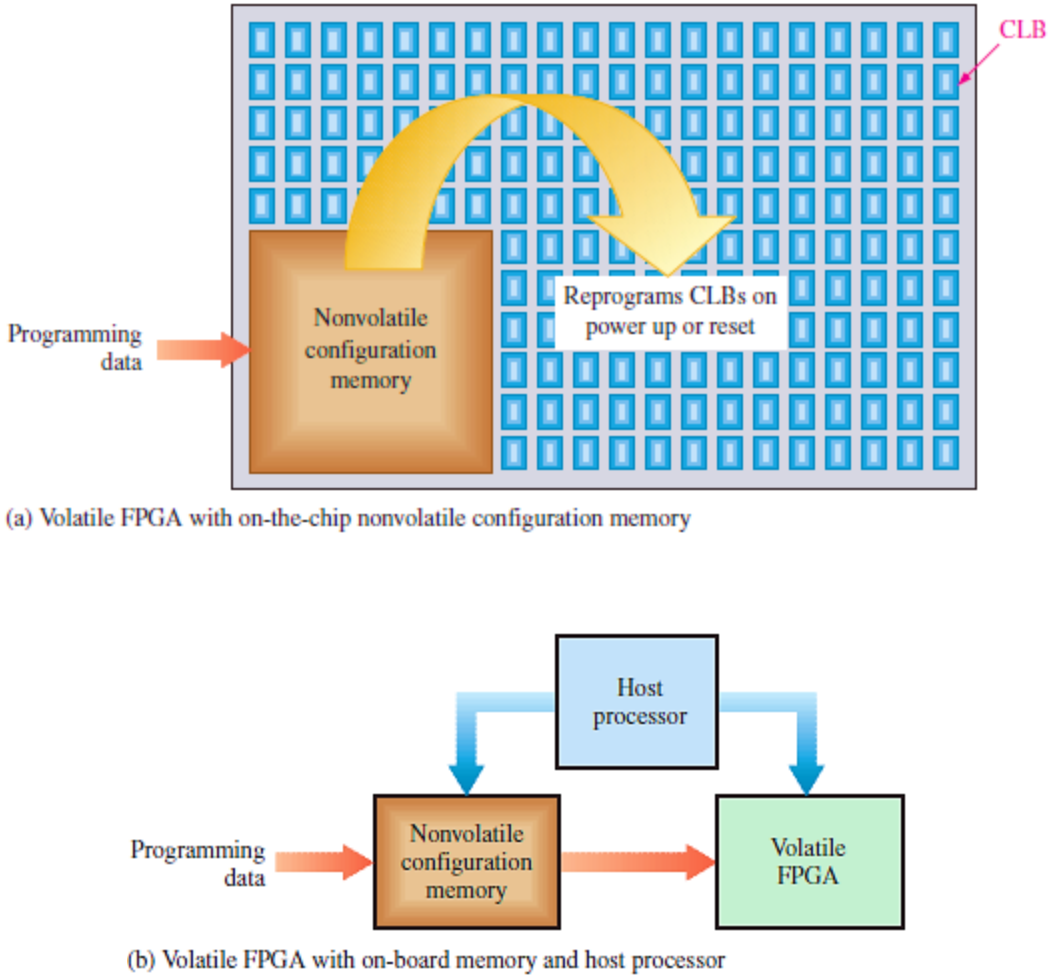


FIGURE 1 Basic concepts of volatile FPGA configurations.

FPGA Cores

- FPGAs, as we have discussed, are essentially like “blank slates” that the end user can program for any logic design. FPGAs are available that also contain hard-core logic.
- A **hard core** is a portion of logic in an FPGA that is put in by the manufacturer to provide a specific function and that cannot be reprogrammed. For example, if a customer needs a small microprocessor as part of a system design, it can be programmed into the FPGA by the customer or it can be provided as hard core by the manufacturer.
- If the embedded function has some programmable features, it is known as a **soft-core** function. An advantage of the hard-core approach is that the same design can be implemented using much less of the available capacity of the FPGA than if the user programmed it in the field, resulting in less space on the chip (“real estate”) and less development time for the user.
- Also, hard-core functions have been thoroughly tested. The disadvantage of the hard core is that the specifications are fixed during manufacturing and the customer must be able to use the hard-core logic “as is.” It cannot be changed later

- Hard cores are generally available for functions that are commonly used in digital systems, such as a microprocessor, standard input/output interfaces, and digital signal processors.
- More than one hard-core function can be programmed in an FPGA. Figure 2 illustrates the concept of a hard core surrounded by configurable logic programmed by the user. This is a basic embedded system because the hard-core function is embedded in the user-programmed logic.

- Hard core designs are generally developed by and are the property of the FPGA manufacturer. Designs owned by the manufacturer are termed **intellectual property** (IP).
- A company usually lists the types of intellectual property that are available on its website. Some intellectual properties are a mix of hard core and soft core.
- A processor that has some flexibility in the selection and adjustment of certain parameters by the user is an example.
- Those FPGAs containing either or both hard-core and soft-core embedded processors and other functions are known as **platform FPGAs** because they can be used to implement an entire system without the need for external support devices.

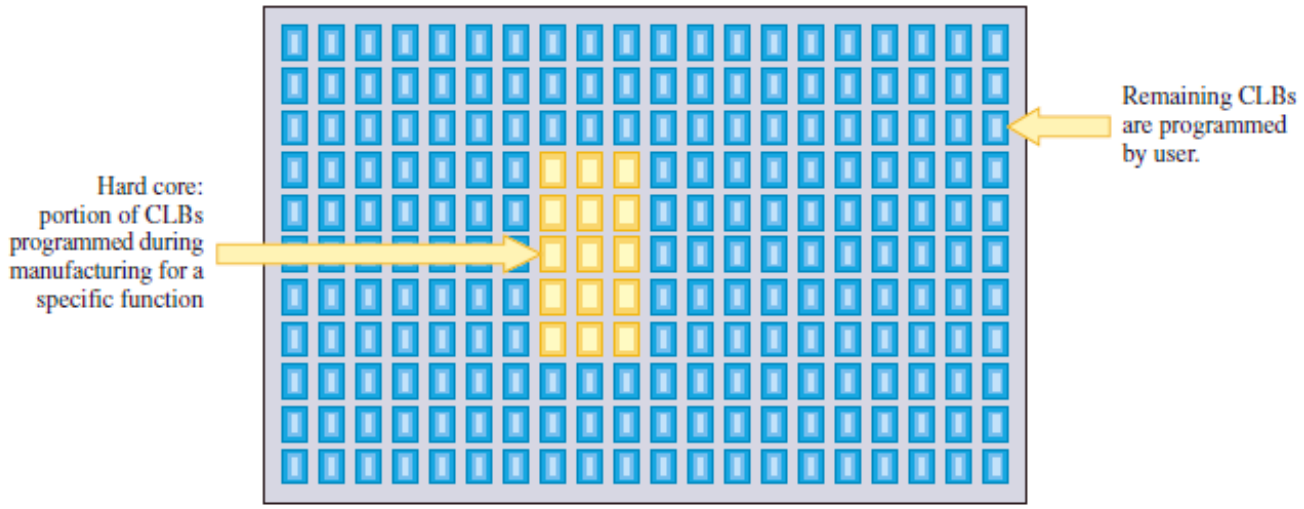


FIGURE 2 Basic idea of a hard-core function embedded in an FPGA.

Embedded Functions

- A block diagram of a typical FPGA is shown in Figure 3. The FPGA contains embedded memory functions as well as digital signal processing (DSP) functions.
- DSP functions, such as digital filters, are commonly used in many systems. As you can see in the block diagram, the embedded blocks are arranged throughout the FPGA interconnection matrix and input/output elements (IOEs) are placed around the FPGA perimeter.

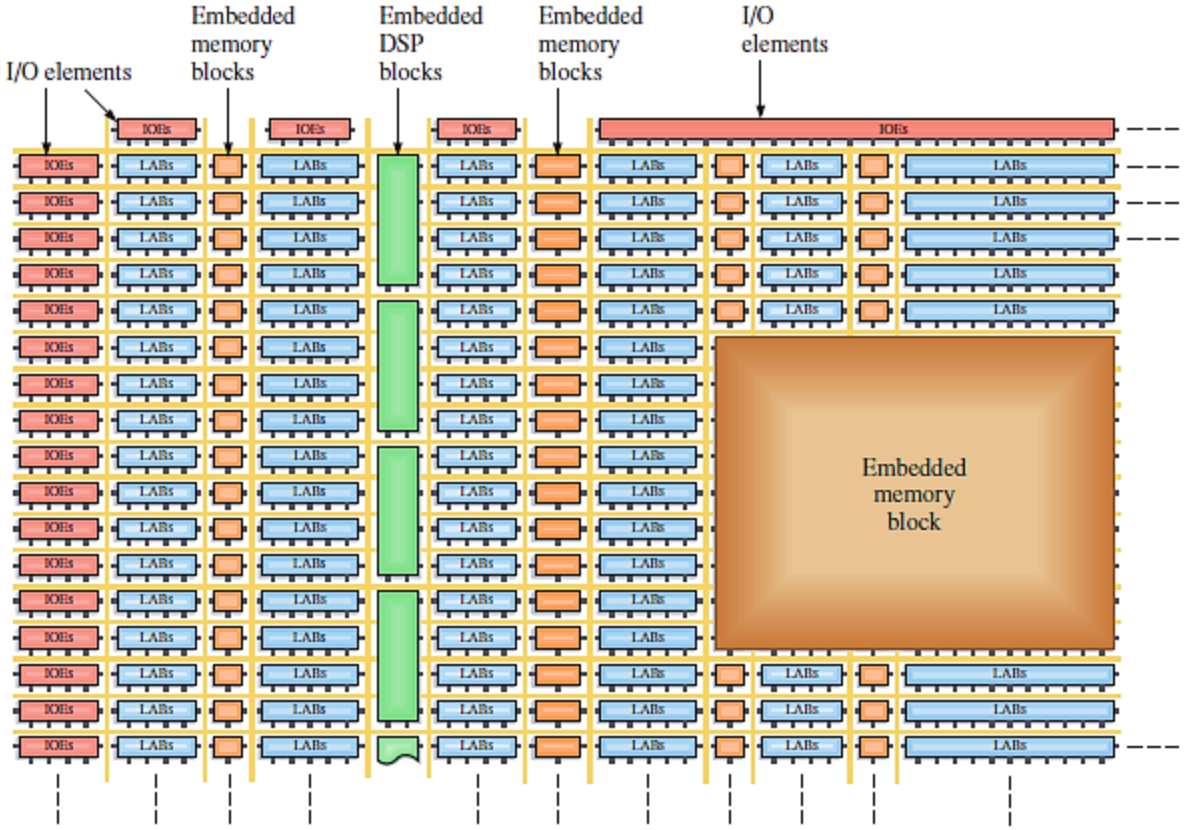


Figure 3 Example FPGA block diagram.

Specific FPGA Devices

- Several manufacturers produce FPGAs as well as CPLDs. Table 1 lists device families from selected companies. Check the website for the most current information.

- FPGAs vary greatly in terms of complexity. Table 2 lists some of the parameter ranges that are available. Keep in mind that these numbers are subject to change as technology advances.

Table 1

FPGA manufacturers.

Manufacturer	Series Name(s)	Design Software	Website
Altera	Stratix Aria Cyclone	Quartus II	Altera.com
Xilinx	Spartan Artix Kintex Virtex	ISE Design Suite	Xilinx.com
Lattice	iCE40 MachX02 Lattice ECP3 LatticeXP2 LatticeGC/M	Lattice Diamond iCEcube2	Latticesemi.com
Atmel	AT40	IDS	Atmel.com

Table 2

Selected FPGA parameters.

Feature	Range
Number of LEs	1,500–813,000
Number of CLBs	26–359,000
Embedded memory	26 kb–63 Mb
Number of I/Os	18–1200
DC operating voltage	1.8 V, 2.5 V, 3.3 V, 5 V

Programmable Logic Software

- In order to be useful, programmable logic must have both hardware and software components combined into a functional unit. All manufacturers of SPLDs, CPLDs, and FPGAs provide software support for each hardware device. These software packages are in a category of software known as computer-aided design (CAD).

- The programming process is generally referred to as **design flow**. A basic design flow diagram for implementing a logic design in a programmable device is shown in Figure 4. Most specific software packages incorporate these elements in one form or another and process them automatically. The device being programmed is usually referred to as the **target device**.

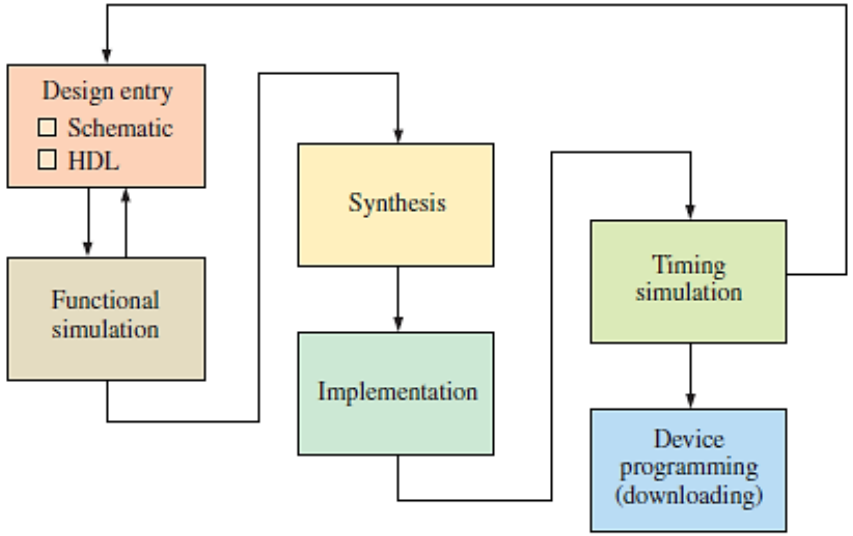


FIGURE 4 General design flow diagram for programming a SPLD, CPLD, or FPGA.

- You must have four things to get started programming a device: a computer, development software, a programmable logic device (SPLD, CPLD, or FPGA), and a way to connect the device to the computer. These essentials are illustrated in Figure 5.

- Part (a) shows a computer that meets the system requirements for the particular software you are using.

- Part (b) shows the software acquired either on a CD from the device manufacturer or downloaded from the device manufacturer's website. Most manufacturers provide free software that can be downloaded and used for a limited time (Examples are Altera Quartus II and Xilinx ISE.).

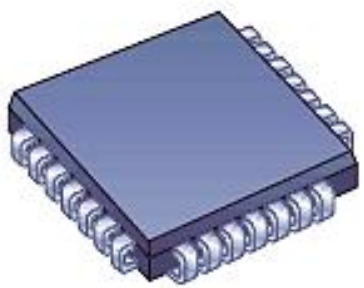
- Part (c) shows a programmable logic device.

- Part (d) illustrates two means of physically connecting the device to the computer via cable by using either the programming fixture into which the device is inserted or the development board on which the device is mounted

- After the software has been installed on your computer, you must become familiar with the particular software tools before attempting to connect and program a device.

(a) Computer

(b) Software (CD or Website download)



(c) Device

(d) Programming hardware (programming fixture or development board with cable for connection to computer port)

Figure 5 Essential elements for programming an SPLD, CPLD, or FPGA.
(d) photo courtesy of Digilent, Inc.

Design Entry

Assume that you have a logic circuit design that you wish to implement in a programmable device. You can enter the design on your computer in either of two basic ways: **schematic entry** or **text entry**. In order to use text entry, you must be familiar with an HDL such as VHDL, Verilog, or AHDL. Most programmable logic manufacturers provide software packages that support VHDL and Verilog because they are standard HDLs. Some also support AHDL, ABEL, or other proprietary HDLs. Schematic entry allows you to place symbols of logic gates and other logic functions from a library on the screen and connect them as required by your design. A knowledge of an HDL is not required for schematic entry.

Functional Simulation

The purpose of the **functional simulation** in the design flow is to make sure that the design you entered works as it should in terms of its logic operation, before synthesizing into a hardware design. Basically, after a logic circuit is compiled, it can then be simulated by applying input waveforms and checking the output for all possible input combinations. Functional simulation is accomplished graphically using a waveform editor or programmatically using a test bench. Graphical waveform editors allow drawing of test stimulus using waveform drawing features and drag and drop techniques.

Synthesis

Once the design has been entered and functionally simulated to verify that its logical operation is correct, the compiler automatically goes through several phases to prepare the design to be downloaded to the target device. During this synthesis phase of the design flow, the design is optimized in terms of minimizing the number of gates, replacing logic elements with other logic elements that can perform the same function more efficiently, and eliminating any redundant logic. The final output from the synthesis phase is a netlist that describes the optimized version of the logic circuit.

Implementation (Software)

After the design has been synthesized, the **compiler** implements the design, which is basically a "mapping" of the design so that it will fit in the specific target device based on its architecture and pin configurations. This process is called *fitting* or *place and routing*. To accomplish the implementation phase of the design flow, the software must "know" about the specific device and have detailed pin information. Complete data on all potential target devices are generally stored in the software library.

Timing Simulation

- This part of the design flow occurs after the implementation and before downloading to the target device. The **timing simulation** verifies that the circuit works at the design frequency and that there are no timing problems that will affect the overall operation. Since a functional simulation has already been done, the circuit should work properly from a logic point of view.

- The development software uses information about the specific target device, such as propagation delays of the gates, to perform a timing simulation of the design. For the functional simulation, the specification of the target device was not required; but for the timing simulation, the target device must be chosen.

Device Programming (Downloading)

Once the functional and timing simulations have verified that the design is working properly, you can initiate the download sequence. A **bitstream** is generated that represents the final design, and it is sent to the target device to automatically configure it. Upon completion, the design is actually in hardware and can be tested in-circuit

Device Programming (Downloading)

Once the functional and timing simulations have verified that the design is working properly, you can initiate the download sequence. A **bitstream** is generated that represents the final design, and it is sent to the target device to automatically configure it. Upon completion, the design is actually in hardware and can be tested in-circuit

Thank You