CHAPTER 1

What Is an FPGA and What Can It Do?

Field-***p***rogrammable ***g***ate ***a***rrays (FPGA) are a special type of integrated circuits (ICs) or chip that can be programmed in the field after manufacture and has three basic building blocks: logic gates, flip-flops + memories, and wires. In this chapter we will quickly review what FPGA is and some of the things it can do.

An IC has input and/or output pins (the gray color in Figure 1-1). The black box is the "brain" of the IC and most ICs have white or gray markings on top of the black box. Most of the ICs mention their specific functions in their datasheet (e.g., is it an amplifier, a processor, a counter, an Ethernet MAC, or a combination of the lot). If you read the FPGA's datasheets and are looking for a specific function, you will probably get very frustrated. That's because they don't mention the purpose or any of the FPGA's feature sets. All you will be able to find is how many logic gates, how much memory, and how to program the FPGA, but you won't find functions or features. You just cannot figure them out from the datasheet.

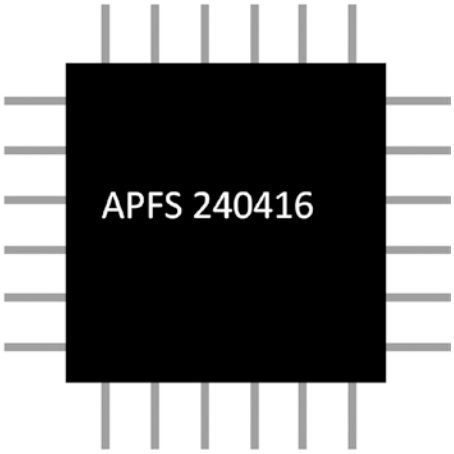

Figure 1-1. *Integrated circuits, or chip, look like this*

A. Pang and P. Membrey, *Beginning FPGA: Programming Metal*, DOI 10.1007/978-1-4302-6248-0_1

FPGAs allow designers to modify their designs very late in the design cycle —even after the end product has been manufactured and deployed in the field. Sound familiar? It should sound a lot like Windows updates or Android/Apple phone software updates. That's one of the most powerful and compelling features of an FPGA but please don't treat FPGA design as a software programming exercise for a microcontroller or processor. An FPGA is not a processor with software. It is an amazing device that allows the average person to create his or her very own digital circuit. You are designing a hardware digital circuit when you are creating an FPGA design. You are going to use a hardware description language (which, incidentally, is used by Intel CPU chip designers too) to design your FPGA. This is a very important concept. We will provide more details when we are putting together some example designs in the later chapters. In the following sections we'll provide a little bit more detail about field-programmable, gates, and arrays and what they can do.

1.1 Field-Programmable

The most valuable FPGA feature is that the end user can program or configure it within seconds. This means that the end user can change the hardware design in the FPGA chip quickly and at will. For example, the FPGA can change from temperature sensor to LED (light-emitting diode) driver within a few seconds. This means that FPGAs are useful for rapid product development and prototyping. This field-programmable magic is done by a configuration file, often called a bit file which is created by a designer (you). Once loaded, the FPGA will behave like the digital circuit you designed!

1.1.1 Configuration Technology

Table 1-1 lists the three types of configuration technologies: static random access memory (SRAM), flash memory, and antifuse.

Table 1-1. *Different Types of Configuration Technology*

	SRAM	Flash	Antifuse
Achronix	YES	--	--
Altera	YES	--	--
Lattice	YES	--	--
Microsemi	--	YES	YES
Xilinx	YES	--	--

Most of the FPGA vendors are using SRAM technology. It is fast and small, and it offers unlimited reprogrammability. One of the drawbacks though is that the FPGA needs time to reload the entire design into SRAM every time you power up the FPGA. This approach also takes more power.

Flash and antifuse technologies are non-volatile (meaning that they can retain data even if the power is turned off) so that they provide the benefit of "instant on" without needing to reload the FPGA bit file every time we power up the FPGA or the system. They also draw less power than the SRAM approach.

Antifuse technology can only be programmed once and can't match the performance of SRAM technology. The only reasons to use an antifuse technology FPGA today is due to its super high reliability and security.

■ **Tips** The world's largest and most powerful particle accelerator—Large Hadron Collider (LHC)—uses antifuse FPGAs to implement radiation protection digital circuits.

This book focuses on SRAM technology because it is the most common technology and is the easiest to program. Most of the SRAM-based FPGAs have an external EEPROM (electrically erasable programmable read-only memory) for storing the bit file, similar to how a computer stores programs on disk. All you need to do is "burn" your bit file into the EERPOM and the FPGA will autoload the bit file from the EEPROM when it powers up.

1.2 Gates = Logic

The gate is the most basic element in digital logic and is more formally known as a logic gate. All modern digital designs are based on CMOS (complementary metal oxide semiconductor) logic gates. To support complex digital designs, FPGAs contain tens of thousands, hundreds of thousands, or even more individual logic elements which are built by logic gates.

1.2.1 The Basic Gate Design Block No. 1: Logic Element

The logic element (LE) is one of the smallest elements in FPGA design. It basically consists of a look-up table (LUT), flip-flop, and multiplexer (which we will cover shortly). FPGAs are used extensively for compute problems that can benefit from parallel computing architectures—for example, cleaning up images being received from an image sensor, local processing on image pixels, and computing difference vectors in H.264 compression. LEs can form any complex or even simple digital function inside the FPGA. Figure 1-2 shows the basic configurable logic elements. Although different FPGA venders have their own LE designs, they are very similar to the one shown in Figure 1-2. Most of the inputs to a LE are connected to the LUT and followed with a flip-flop (register). The output of the LE is selected by a multiplexer (MUX). The LUT and MUX are the major configurable blocks in the LE. Don't worry, this will all make sense after you read the next section.

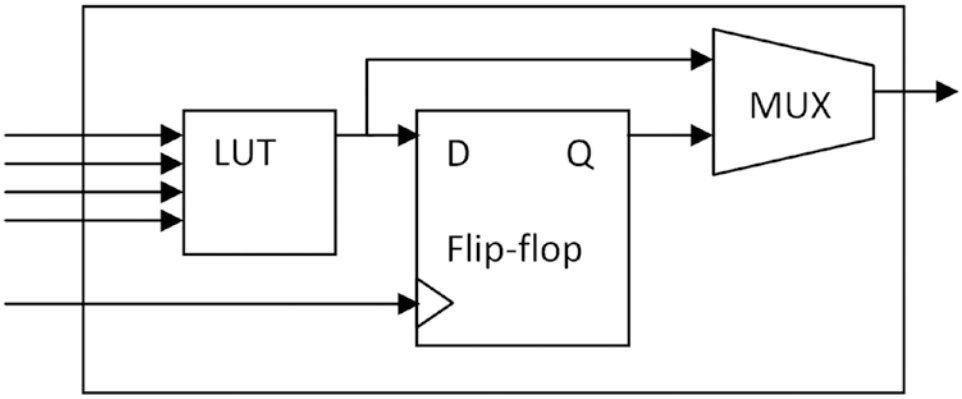

Figure 1-2. *Basic configurable LEs*

1.2.1.1 The Magic Block: LUT

The LUT is basically just a small amount of read-only memory. A four-input, one output LUT, can generate any four-input Boolean function (AND/OR/XOR/NOT) (Figure 1-2). We will provide more details on Boolean function in Chapters 8 and 9. When you configure the FPGA, the contents of the LUT will be configured accordingly. Table 1-2 shows an OR gate with all four possible inputs the LUT supports. The four inputs act as an address bus, addressing one of 16 (2 ^4) stored bits; therefore it can emulate any Boolean function that can be handled in 4 bits (Figure 1-3). In some FPGA LE designs, LUTs can be combined to form a 16-bit shift register or memory block. There is another similar block, the MUX, in the basic LE. It is used to select the output source from a register or directly from LUT. In reality, the LEs are a bit more complex in that they may very well have more than one LUT and MUX.

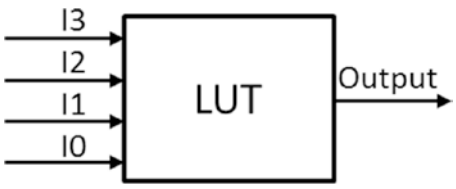

Figure 1-3. *Four inputs, one output Boolean function*

Table 1-2. *Four-Input, One-Output LUT*

Address	I3	I2	I1	I0	Output
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

If you are serious about FPGA digital design, then you may want to read the LUT datasheet from the FPGA vendor. You can do more with the same FPGA, if you know how to fully use the LUT blocks.

1.2.1.2 The 1-bit storage block: Flip-flop

The flip-flop (or register) is the smallest storage unit in an FPGA. Each flip-flop can store 1 bit of information at a time. Like a light switch, it can either be on (a value of 1) or off (a value of 0). The basic function of a flip-flop is to hold information and make it available to the logic elements for the computing process. This flip-flop is one of the most important blocks in a computer. Figure 1-4 shows the digital design of a D-type flip-flop, which is a basic flip-flop used in FPGA.

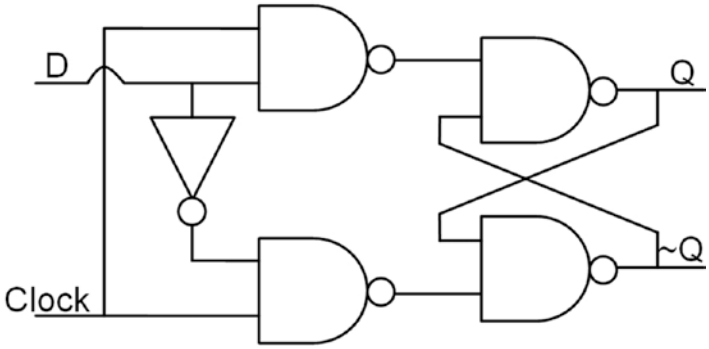

Figure 1-4. *Flip-flop logic gate design (D-type flip-flop)*

There are two inputs (D and Clock) and two outputs (Q and ~Q) in a flip-flop. The D flip-flop captures the value of the D input at a defined stage of the clock cycle (such as the rising edge of the clock input). That captured value becomes the Q output. At other times, the output Q does not change. The D flip-flop can be viewed as a memory cell, a zero-order hold, or a delay line. Multiple flip-flops connected together can form a shift register.

■ **Tip** You can go to the following web site for more details about D-type flip-flop: `http://electronics-course.com/d-flip-flop`.

Most of the FPGA digital designs are synchronous design. It means that all the inputs and outputs are synchronous to one clock or more. All the examples in this book are synchronous design with the rising edge of the clock because most of the FPGAs are built for implementing synchronous design.

■ **Note** A synchronous circuit is a digital circuit in which the changes in the state of memory elements are synchronized by a clock signal. In a sequential digital logic circuit, data is stored in memory devices called flip-flops or latches. Source: `www.wikipedia.org`. It is like shooting pictures. Each picture stores the "data" when you hit the shutter button. The picture is the storage elements and the button is the clock signal.

1.2.2 The Basic Gate Design Block No. 2: Configurable IO Block

Another basic logic design element is configurable input/output block (IOB). It is used to connect the LE to the outside world. It can be configured as an input or output, as b-directional, or not connected. The IOB can support different types of electrical input/output (I/O) specifications (e.g., TTL logic, 3.3V CMOS, and PCIe) and add internal pull-up or pull-down resisters. Remember to select the correct I/O specification. It is because different specifications have different driving strength (voltage and current) and most of the time they are **not** compatible. The worst is overload the FPGA IO and break the FPGA.

The latest FPGA IOB can support bandwidth higher than 10 Gbps. Some FPGAs even have direct optical input and output physical connections (Source: www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01161-optical-fpga.pdf). Chapter 12 provides a detailed discussion on how to use IOBs.

1.2.3 The Basic Gate Design Block No. 3: Internal RAM

The internal RAM (random access memory) block is another configurable unit in the FPGA. The main configuration parameter is number of read/write ports. You can read and write the same memory with different locations at the same time. It is like one instruction to read and write on the same physical memory with different address. Can you think of a use case of this type of configuration (read and write at the same time/clock cycle)? One possibility is FIFO (first in, first out memory) which is used to pass data from module A to module B. There is an FPGA vendor that puts DRAM (dynamic random access memory) (which is the one you'll find in your computer because of its density—it can pack more bits into the same physical size) inside the FPGA and allows the rest of the logic to access it directly. Some FPGA vendors put flash memory (which is a non-volatile memory such as you'd find in a memory card or USB stick) into their FPGA. If your application needs non-volatile memory, then an FPGA with built-in flash memory would be a good fit for you. In Parts 3 and 4, we will show how to configure internal RAM.

1.3 Arrays Have Many Connections

An array is a large group of things put together with a particular order. At this moment, we know that FPGA has a lot of LE, IOB, and internal memory. All of them can be configured to do whatever you would like them to do. There is a last piece of kit inside the FPGA that can be configured; the connections between the LEs and IOBs. To make the connection efficient, FPGA vendors put the LEs and IOBs into a two-dimensional array (some newer FPGA's have three-dimensional arrays instead). Figure 1-5 shows an FPGA array example. All of the IOBs are close to the IC edge such that it has the shortest distance to the outside world. The wires between each LE and IOB are the configurable connection (wires). These are extremely flexible and easy to use. Most of the time, the FPGA vendor tools take care of all the connections for you.

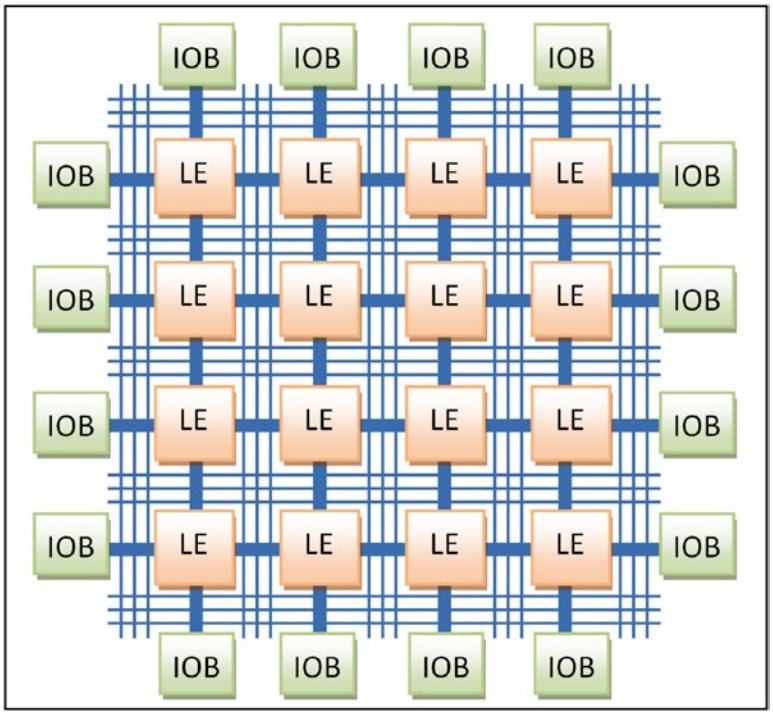

Figure 1-5. *Gate array*

Digital logic theory: If you have enough NAND gates then you can build anything that you want. In an ideal world we don't need to have complicated LE blocks. All we need to have are NAND gates and connections in between.

In the real world: The routing between NAND gates will very quickly run out of room to finish even a simple job. Traditional FPGAs are optimized for more complex configurable logic element blocks with a relatively limited number of interconnections between them. Today, FPGAs have added more interconnections by using 3D arrays. These 3D arrays are stacked like multiple chips with connections between chips.

Keep in mind: It is possible to run out of connections between LEs and IOB and thus fail to generate a viable FPGA design.

1.4 What Can It Do?

Generally speaking, all the FPGA does is generate ones (3.3 V) and zeroes (0 V) which means it can do everything or nothing. That doesn't sound terribly impressive does it? However, you need to ask the right question in order to get the right answer. The right question is, "What you want it to do?" The more you do, the more it can do for you! Think of the FPGA as a piece of clay. You can mold that clay into any number of shapes, make a plate, make a statue, or make a tiny house. Its potential is limited only by your imagination, and that's also true of an FPGA—it has the capability to take on practically any function you can imagine!

It is very difficult to design your own chip completely from scratch. The big difference between an FPGA and every other chip you can buy on the market is that an FPGA doesn't actually do anything. It has no intended function when you buy it. It is not like a microcontroller that has a defined function or a generic process ready to run software. An FPGA gives you the ultimate in flexibility, allowing you to design anything you can imagine in the digital domain. If you want to turn your FPGA into an 8051 microcontroller, you can do it. You can configure your FPGA to be a custom LED driver for 1,000 LEDs, a 100 PWM (Pulse Width Modulator), or a universal asynchronous receiver/transmitter (UART or COM port) device, which is a common computer interface. However, just because it can be done with an FPGA does not mean that it will necessarily be easy to do so.

When you play a join-the-dots puzzle, you need to follow the numbers (features) to draw lines (design). It is like you get a processor; you have a fixed amount of counters, timers, and interface types (features); and you write your code (design) to follow the features. FPGAs are like join-the-dots puzzles but without any numbers at all (Figure 1-6). You need to design all of the features and rules in the FPGA to make it "work." Another way to look at it is rather than you writing software for a predetermined feature set, you have the ability with an FPGA to actually define the feature set yourself—you don't have to follow the numbers because with an FPGA, there simply aren't any numbers to follow!

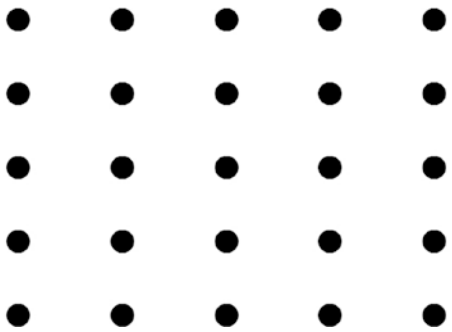

Figure 1-6. *Connect-the-dots puzzles without number*

Some fun: Try to join all 25 dots in Figure 1-6 by drawing no more than eight straight lines. The straight lines must be continuous.

■ **Tips** Think out of the box. This tip is valid for designing FPGA too!

1.5 It Can Get the Job Done Fast!

FPGAs can get the job done faster than a computer, if you can separate your job into discrete pieces. This is due to order reduction in FPGA tools and parallelism operation in FPGA. The FPGA tool (mention in Chapter 3) is smart enough to reduce complicated operations to simpler ones. This operation is order reduction! The tools can reduce complex operation like multiplying by 2 into an addition which runs faster and uses less energy. An FPGA can parallelize a task that was already much slower to run as software on a CPU (central processing unit). Once the designer starts to realize the following, "I can perform in 25 FPGA clock cycles a task which takes my CPU 200,000 clock cycles, and I can do this task in parallel 5 items at a time," you can easily see why an FPGA could be a heck of a lot faster than a CPU! Reversing the order of bits inside a 32-bit integer is a very good example for order reduction. Computers need to use a `FOR` loop to do it which will take a number of clock cycles to complete. You can accomplish the same thing in a single cycle with an FPGA.

Data flow into and out of the FPGA is expandable too! So you can send a lot of data to an FPGA to get more jobs done at the same time. Image, video, and Ethernet packet processing all need this kind of high bandwidth data flow.

For example, suppose you want to do some processing work on an image, and let's say you want to rotate it. Let's also say that it takes one second to do this, and you have 10,000 images to do. A single processor would take nearly three hours to process this workload. Of course, modern CPUs are multicore, so let's be generous and assume we have eight cores at our disposal. Now we can do the same work in just over 20 minutes. With the FPGA, though, we might have our "image processing" unit, and maybe 500 of those fit on a single FPGA. If we could keep those units full of data, we'd complete our task in only 20 seconds. If we also consider that it is now being processed in hardware rather than software, we could say that it only takes half a second to process an image, dropping our total time down to ten seconds. Okay, this example is a bit contrived and there are plenty of ifs, buts, and maybes, but it gives you an idea of the power of an FPGA.

1.6 FPGA vs. Processor

The major difference between a processor and an FPGA is that an FPGA doesn't have a permanent hardware configuration; on the contrary, it is configurable according to the end-user needs. However, processors have a permanent hardware configuration which means that all the transistors, registers, interface structures, and all of the connections are permanent. A processor can only do predefined tasks (accumulation, multiplication, I/O switch, etc.). Designers make the processor do these tasks "in a consecutive manner" by using software, in accordance with their own functions.

Hardware configuration in the FPGA is not fixed so it is defined by the end user. Although logic elements are fixed in FPGA, functions they achieve and the interconnections between them are controlled by the user. So tasks that FPGAs can do are not predefined. You can have the task done according to the written hardware description language (HDL) code "in parallel," which means concurrently. The capability of parallel processing is one of the most important features that separates FPGAs from processors and makes them superior in many areas.

Processors are generally more useful for repetitive control of specific circuits. For example, using an FPGA for simple functions such as turning on or off a device from a computer may be overkill. This process can easily done with many conventional microcontrollers. However, FPGA solutions are more reasonable, if you want to process 4K video data on the computer. Table 1-3 compares FPGAs and processors in a few more ways.

Table 1-3. *FPGA vs. Computer*

	FPGA	Processor
Cost	High	Low
Hardware structure	Flexible	Fix
Execution	Concurrent	Sequential
Programming	HDL	Assembly language
Development time	Long	Short
Power	Efficient	Not efficient

There is something that the both have in common and we will find out more about that together in Chapter 2.

1.7 Summary

Field-***p***rogrammable ***g***ate ***a***rrays have three basic building blocks.

1. Logic gates
2. Flip-flops + memories
3. Wires

FPGAs can do many things, or they can do nothing, with these three building blocks. It all depends on you. You need to design the hardware structure to handle the task you want. FPGAs can concurrently process your tasks and this is the main difference between FPGA and a processor. Chapters 2, 3, and 4 will get you ready to design real digital logic.

In Chapter 2, you will get hands on with the hardware platform we will be using.

In Chapter 3, you will install FPGA digital design environment software.

In Chapter 4, you will create your first digital design in the FPGA world!

As you work your way through this book (Part 2 and Part 3), you will create a number of design blocks; each block can be reused. Each block is like LEGO, you can reuse all the blocks you create in other designs.

"Nothing will work unless you do."

—Maya Angelou

Electronic supplementary material The online version of this chapter (doi:10.1007/978-1-4302-6248-0_1) contains supplementary material, which is available to authorized users.